Engenharia de Software para Ciência de Dados

Prof. Marcos Kalinowski
Prof. Tatiana Escovedo
Prof. Hugo Villamizar

Apresentações

Software Science Lab

Marcos Kalinowski

kalinowski@inf.puc-rio.br

@prof_kalinowski

linkedin.com/in/kalinowski

- Professor do Quadro Principal e Coordenador de Pós-Graduação do **Departamento de Informática da PUC-Rio** (http://www.inf.puc-rio.br)
- Coordenador do **Software Science Lab**, onde orienta pesquisas de mestrado e doutorado
- Coordenador na iniciativa **ExACTa PUC-Rio** de Transformação Digital (http://www.exacta.inf.puc-rio.br)
- Senior Advisor do Programa Nacional **MPS.BR**
- Membro da rede de excelência em pesquisa **ISERN** (International Software Engineering Research Network)
- Mais informações em http://www.inf.puc-rio.br/~kalinowski

Apresentações

Software Science Lab

Tatiana Escovedo

tatiana@inf.puc-rio.br

@tatianaesc

linkedin.com/in/tatianaescovedo

- Professora do **Departamento de Informática da PUC-Rio** e coordenadora de cursos de pós-graduação lato sensu
- Colaboradora do **Software Science Lab**, onde atua em pesquisas nas áreas de Ciência de Dados e Engenharia de Software
- Gerente da área de Tecnologia, Gestão de Dados e Conhecimento na diretoria de Comercialização e Logística da **Petrobras**
- Autora dos **livros** "Introdução a Data Science - Algoritmos de Machine Learning e Métodos de Análise", "Jornada Java" e "Jornada Python"

Apresentações

Software Science Lab

Hugo Villamizar

hvillamizar@inf.puc-rio.br

@Richard_Guarin

linkedin.com/in/hrguarinv

- Doutorando do **Departamento de Informática da PUC-Rio**
- Colaborador do **Software Science Lab**, onde atua em pesquisas nas áreas de Ciência de Dados e Engenharia de Software
- Analista de Pesquisa e Desenvolvimento da iniciativa **ExACTa PUC-Rio** de Transformação Digital

Conhecendo a Turma

Quem são vocês?

Nome?

Experiência em Engenharia de Software?

Experiência em Ciência de Dados?

Expectativas?

Ementa e Conteúdo Programático

Software Science Lab

- Introdução à Engenharia de Software e à Ciência de Dados
- Gestão Ágil de Projetos de Software
- Engenharia de Sistemas de Software Inteligentes
  - Conceitos de BizDev, DevOps/MLOps e Experimentação Contínua
  - Boas práticas da Engenharia de Software
- Definição do Problema e Requisitos
  - Ideação de MVPs de Sistemas de Software Inteligentes (Lean Inception, MVP Canvas)
  - Requisitos de Sistemas de Software Inteligentes (ML Canvas, PerSpecML)
- Visão geral das tecnologias envolvidas
  - Revisão da Linguagem Python e Orientação a Objetos
  - Principais Bibliotecas Python utilizadas em Ciência de Dados
  - Boas Práticas de Projeto e Construção de Sistemas (princípios SOLID e boas práticas de codificação)
- Análise Exploratória e Pré-Processamento de Dados
- Construção de Modelos de Machine Learning (Modelagem e Inferência):
  - Algoritmos de Machine Learning de Classificação e Regressão
  - Avaliação e Comparação de Modelos
- Princípios de Projeto e Boas Práticas Aplicadas a Machine Learning
  - Reprodutibilidade de Notebooks
  - Aplicação dos Princípios SOLID no contexto de ML
  - Aplicando Análise Estática e Testes
  - Exemplo Prático de Machine Learning com POO
  - Gerência de Configuração
- Arquitetura e Deploy de Modelos de Machine Learning

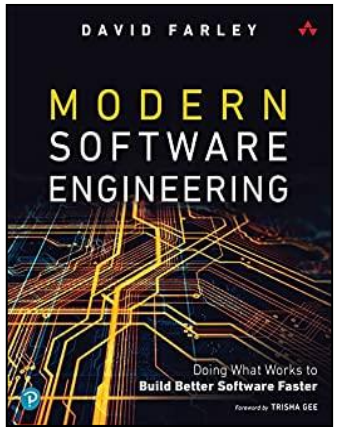
Avaliação

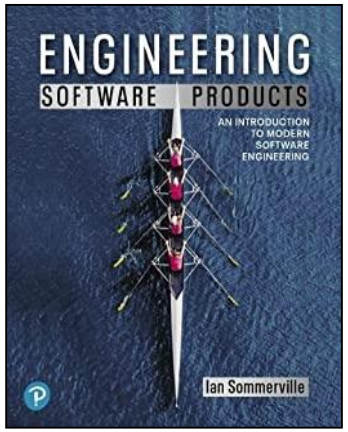
- A1 – Especificação de um sistema de Machine Learning
- A2 – Projeto de ciência de dados, seguindo boas práticas da engenharia de software

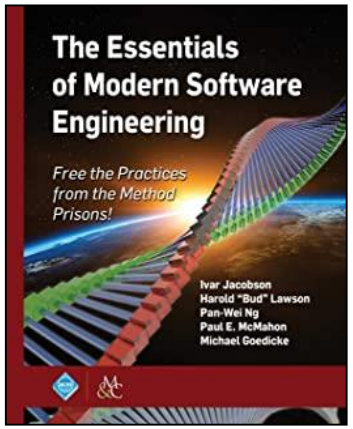
- Nota final = A1 x 0,5 + A2 x 0,5

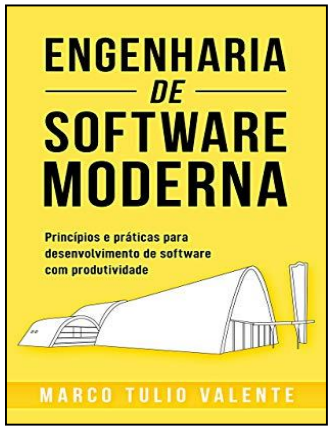
Alguns Livros Indicados (Eng. de Software)

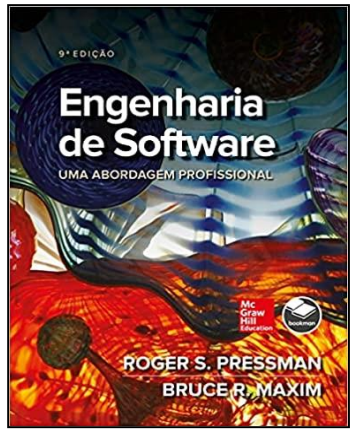
Software Science Lab





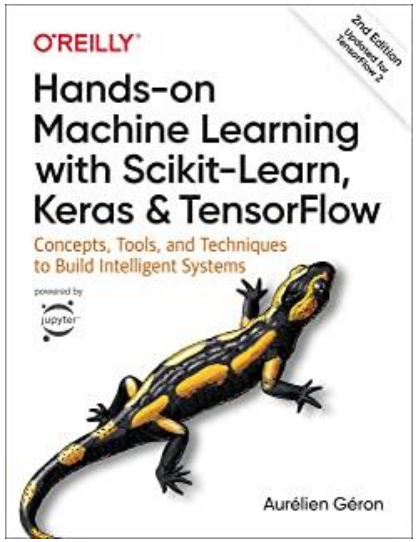


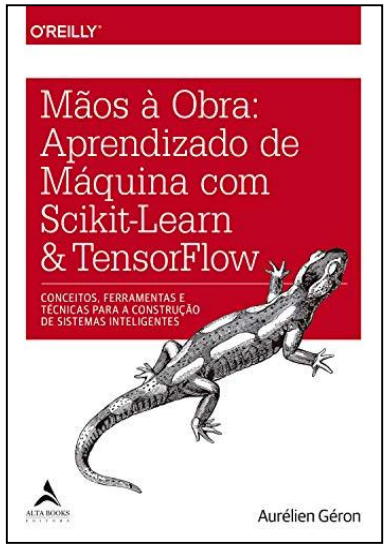


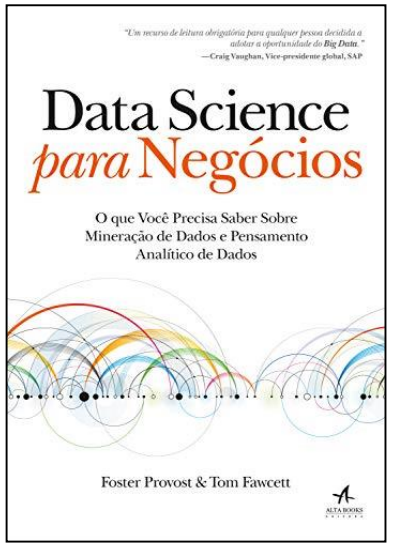


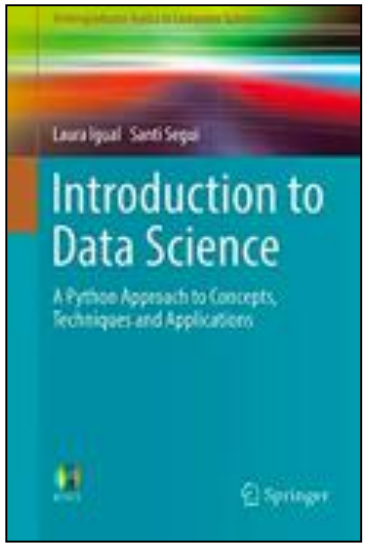
Alguns Livros Indicados (Ciência de Dados)

Software Science Lab









Software Science Lab

Introdução à Engenharia de Software

Engenharia de Software para Ciência de Dados

O que é Engenharia de Software?

“Engenharia de Software é a aplicação de uma abordagem sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção de software”

IEEE Std 610.12 (1990)

“If one end of the software development competency spectrum is occupied by code-and-fixe development, the other end is occupied by software engineering...”

[McConnel99]

Necessidade

- **Por que** preciso de **Engenharia de Software?**
  - **Programação** é parte **importante** do processo de Engenharia de Software, **mas não é tudo!**

- Precisamos também saber...
  - **o que** programar,
  - **como** programar,
  - se o que foi programado está **certo**,
  - etc.

Fatores envolvidos

- Fazer software no “mundo real” deve considerar fatores como:
  - Custo
  - Prazo
  - Qualidade

- Em função do tamanho do software, esses fatores se tornam difíceis de garantir!

Mas fazer software não é arte?

Software Science Lab

- Parte arte, parte engenharia...
  - Se o cantor/ator/pintor errar, a audiência fica chateada
  - Se o engenheiro civil errar o prédio pode cair
  - Se o médico errar o paciente pode morrer

- Se o desenvolvedor de software errar, o que pode acontecer?

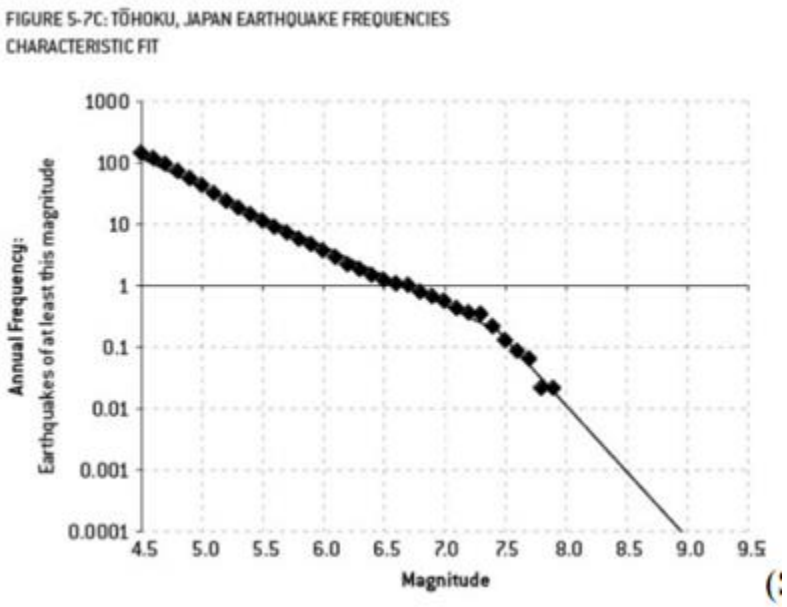
Caso Real 1: Desastre de Fukushima

Segundo pior desastre nuclear da história: 154.000 pessoas evacuadas e mais de 18.000 mortes

Overfitting no Desastre de Fukushima

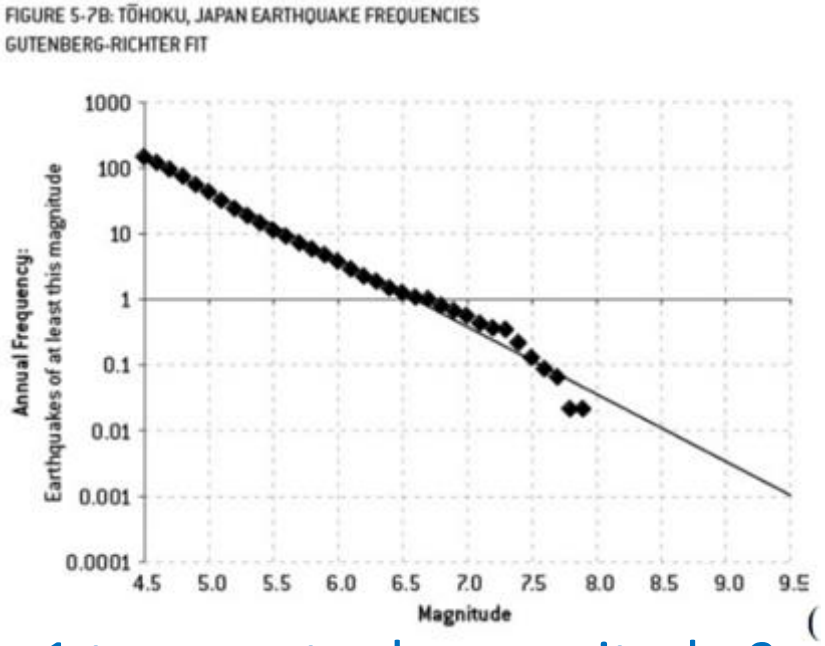
Software Science Lab

Modelo utilizado (com Overfitting)



1 terremoto de magnitude 9 a cada **13.000** anos

Modelo linear (sem Overfitting)



1 terremoto de magnitude 9 a cada **300** anos

A Usina foi construída para suportar um terremoto de magnitude 8,6.

O terremoto de 2011, que devastou a usina, foi de magnitude 9.

Fonte:
https://medium.com/@ml.at.berkeley/machine-learning-crash-course-part-4-the-bias-variance-dilemma-a94e60ec1d3

Caso Real 2: Reconhecimento Facial

Software Science Lab

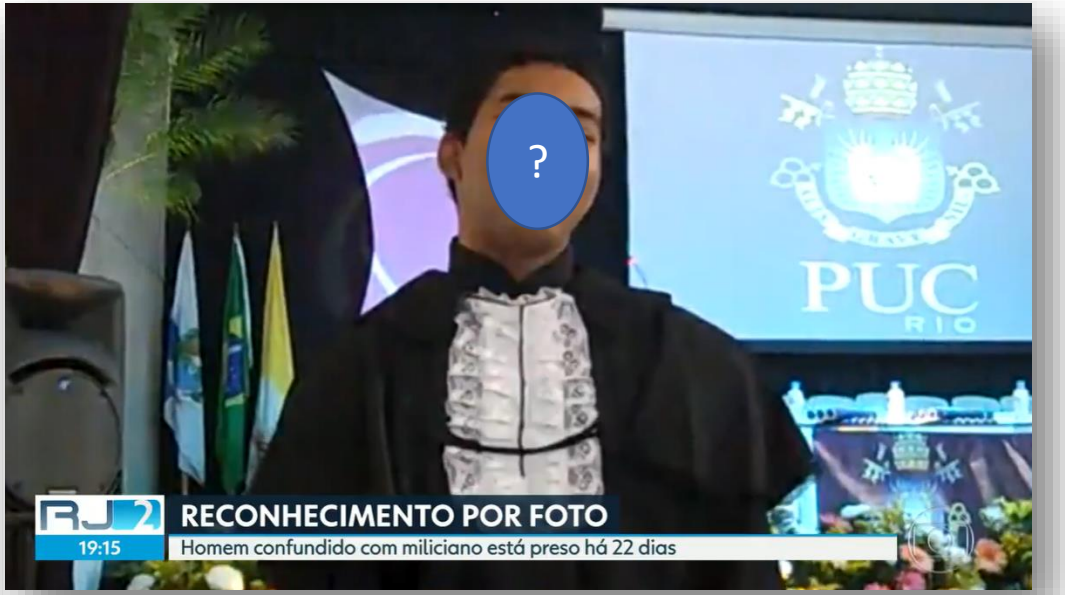
MENU g1 RIO DE JANEIRO BUSCAR

Polícia reconhece erro em prisão de cientista de dados, há 22 dias na cadeia acusado de ser miliciano

Raoni Lázaro Barbosa foi acusado pela polícia de integrar uma milícia em Caxias, mas a própria corporação afirma que testemunhas desfizeram reconhecimento por fotos e que pediu a soltura do homem.

Por Ben-Hur Correia, RJ2

08/09/2021 19h37 · Atualizado há um mês



Sistema de reconhecimento facial da PM do RJ falha, e mulher é detida por engano

Secretaria reconheceu o erro e lamentou o fato. Segundo a corporação, a pessoa foi levada para a delegacia, onde foi confirmado que não se tratava da criminosa procurada.

Por G1 Rio

11/07/2019 05h00 · Atualizado há 2 anos

Sections The Washington Post Democracy Dies in Darkness Get one year for $29 Sign in

Technology

Wrongfully arrested man sues Detroit police over false facial recognition match

The case could fuel criticism of police investigators' use of a controversial technology that has been shown to perform worse on people of color

Caso Real 3: Carros Autônomos

Software Science Lab

Tesla: acidente com carro 'sem motorista' mata 2 pessoas nos EUA

19 abril 2021

Caso bem sucedido 1: *Apollo Space Program*

Margaret Hamilton, Director of the Software Engineering Division of the MIT Instrumentation Laboratory, which developed on-board flight software for the Apollo space program

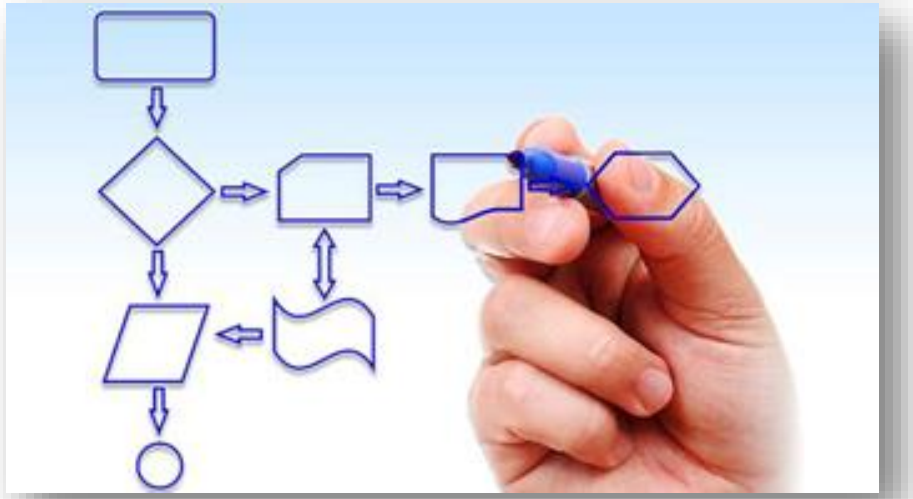
Caso bem sucedido 2: ExACTa

Processo de Engenharia de Software

Tem como objetivo garantir a produção de software de alta qualidade que está de acordo com as necessidades dos seus usuários finais com um cronograma e custo previsível.

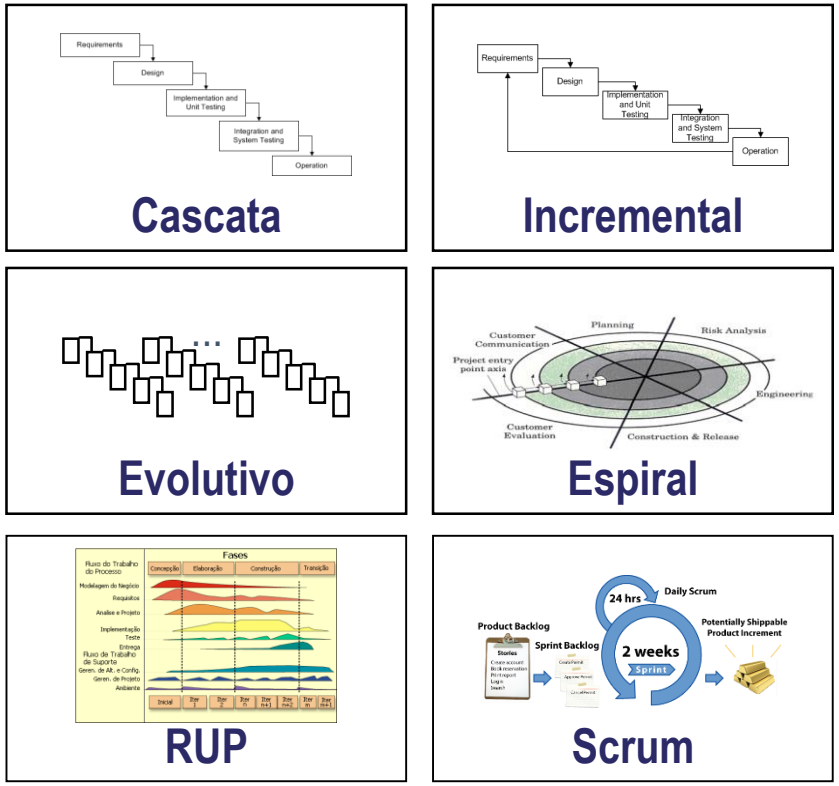
Processo de Engenharia de Software

- Conjunto de atividades:
  - com **modelo de ciclo de vida**
  - bem definidas
  - com responsáveis
  - com artefatos de entrada e saída
  - com dependências entre as mesmas e ordem de execução
  - descrição sistemática de como devem ser realizadas



Modelos de Ciclo de Vida de Processos

- Principais modelos de ciclo de vida:
  - Modelo Cascata
  - Modelo Incremental
  - Modelo Evolutivo
  - Modelo Espiral
  - Ciclo de Vida Associado ao RUP
  - Ciclo de Vida Associado ao Scrum



Modelo Incremental

- Requisitos são segmentados em uma série incremental de produtos. O processo se repete até que um produto completo seja produzido
- A segmentação de requisitos é usualmente planejada a priori
- Adotado quando há necessidade de entrega de um produto funcional em pouco tempo
- A cada incremento é produzida uma versão operacional do software

Modelo Incremental

Vantagens

- Menor custo e menos tempo são necessários para se entregar a primeira versão
- Riscos associados ao desenvolvimento de incrementos são menores, devido ao seu tamanho reduzido
- Número de mudanças nos requisitos pode diminuir devido ao curto tempo de desenvolvimento dos incrementos

Desvantagens

- Se os requisitos não são tão estáveis ou completos quanto se esperava, alguns incrementos podem precisar ser retirados de uso e retrabalhados
- O gerenciamento de custo, cronograma e configuração é mais complexo

Modelo Evolutivo

- Versões parciais são desenvolvidas que atendem aos requisitos conhecidos inicialmente
- A primeira versão é usada para refinar os requisitos para uma segunda versão
- A partir do conhecimento sobre os requisitos, obtido com o uso, continua-se o desenvolvimento, evoluindo o produto

Modelo Evolutivo

Vantagens

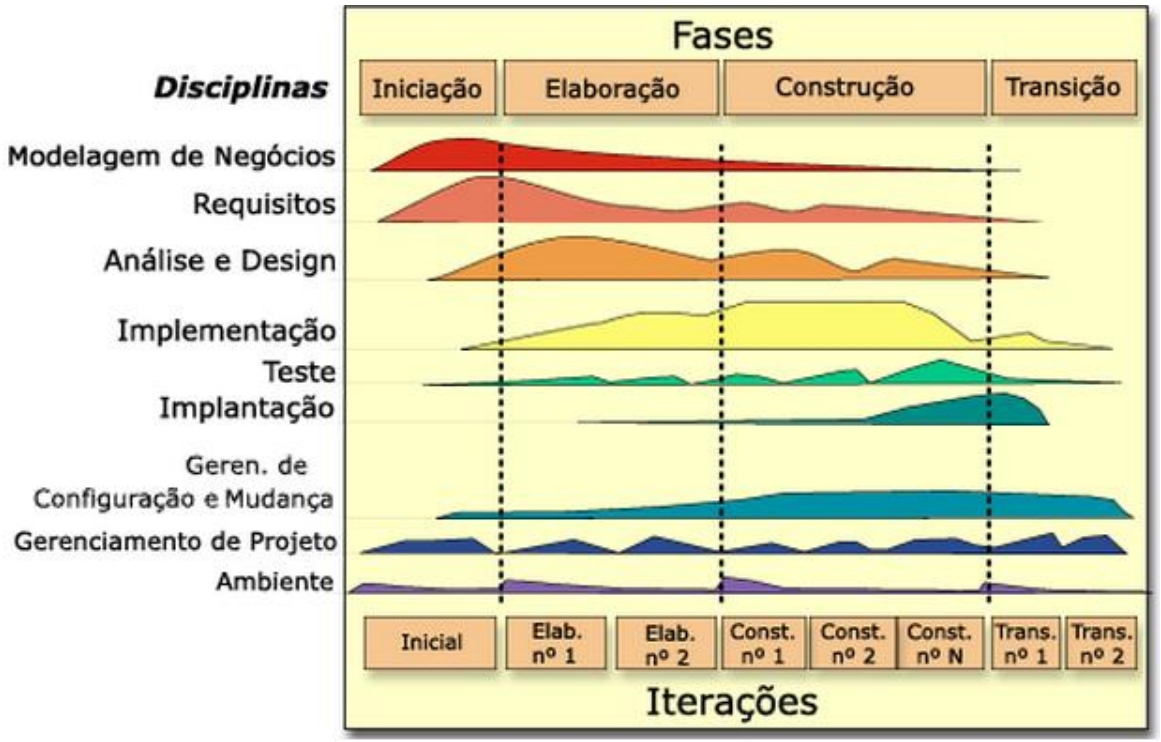
- Adequado quando os requisitos não podem ser completamente especificados de início
- O uso do sistema pode aumentar o conhecimento sobre o produto e melhorar os requisitos

Desvantagens

- Necessária uma forte gerência de custo, cronograma e configuração
- Usuários podem não entender a natureza da abordagem e se decepcionar quando os resultados são não satisfatórios

Exemplo Iterativo Incremental Orientado a Planos: *Rational Unified Process (RUP)*

- O Processo Unificado (*Unified Process - UP*) é um nome genérico para uma família de modelos de processo que atendem a determinados critérios:
  - Ser iterativo incremental
  - Ser orientado a casos de uso
  - Ter foco em tartar riscos cedo (plan-driven)
- O UP define quatro fases: Concepção (Iniciação), Elaboração, Construção e Transição
- O ***Rational Unified Process* (RUP)** é um refinamento do processo unificado criado pela Rational Software (atualmente da IBM)

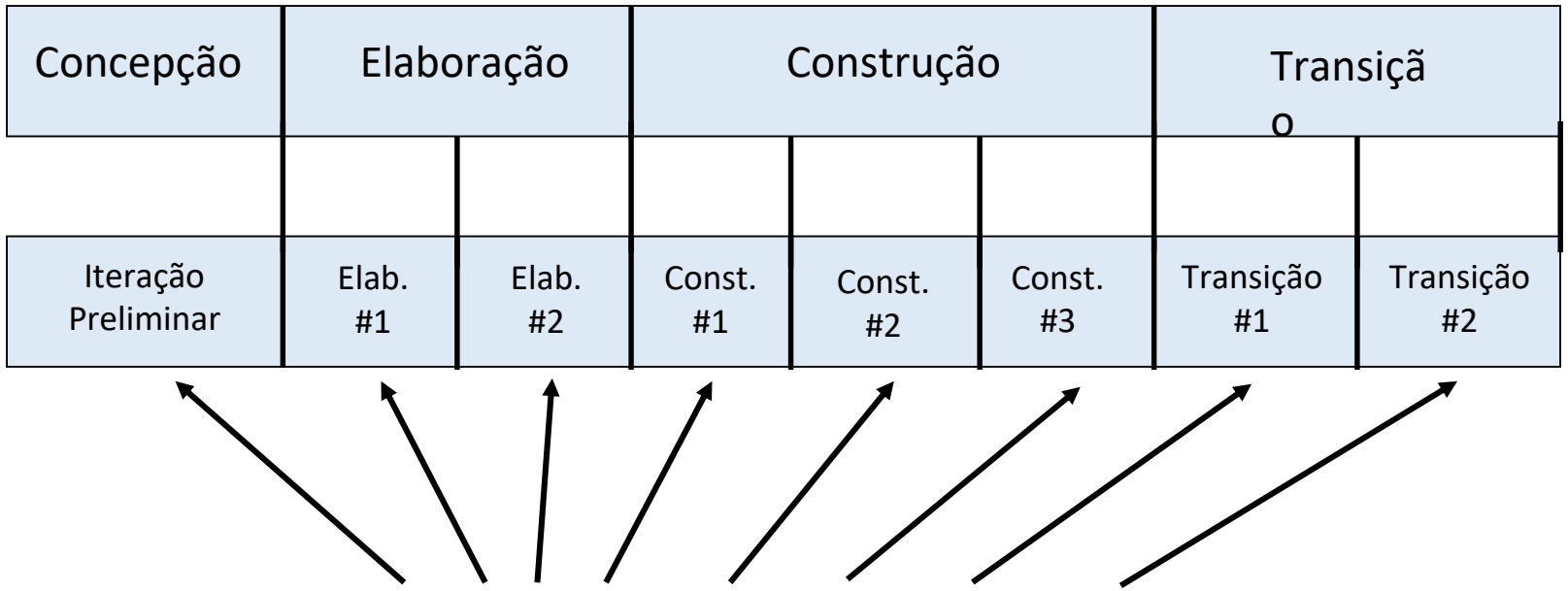


RUP

- O ***RUP (Rational Unified Process)*** é um framework de processos que pode ser adaptado e estendido, bastante aplicado no contexto de desenvolvimento de software orientado a planos
- O desenvolvimento de software é feito de forma iterativa e as iterações são planejadas em número, duração e objetivos
- Orientado a casos de uso

O RUP é Iterativo e Incremental

- Cada fase é dividida em iterações:



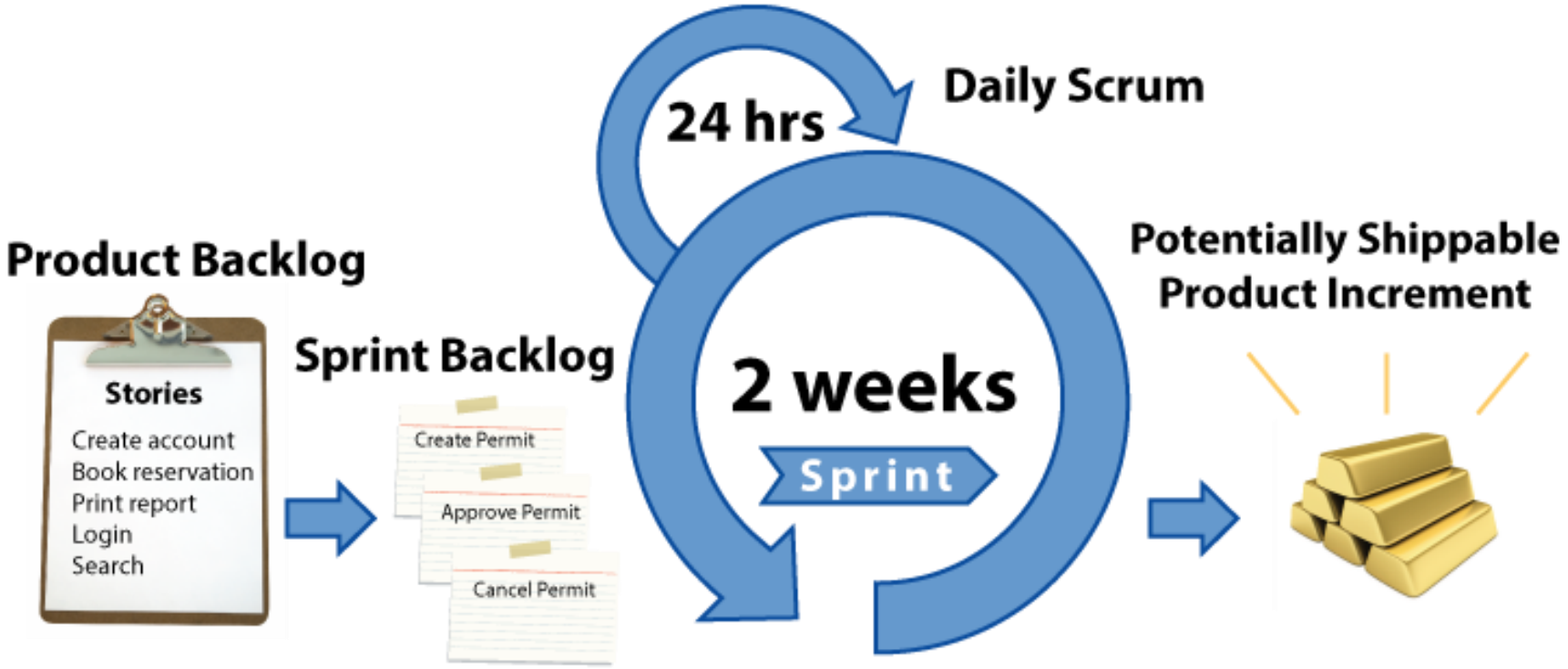
Fases do RUP

- Concepção
  - Comunicação e Planejamento
  - Entender os requisitos gerais e determinar o escopo do esforço de desenvolvimento
- Elaboração
  - Planejamento e Modelagem
  - Planejar as atividades e recursos necessários; especificar as características e projeto da arquitetura
- Construção
  - Construir o produto e evoluir a visão, arquitetura e planos, até que o produto esteja pronto para entrega
- Transição
  - Implantação
  - Garantir que o sistema tem o nível correto de qualidade para atingir os objetivos; realizar correções, treinamento de usuários, ajustes e adição de elementos que estavam faltando. O produto final é produzido e entregue.

RUP: Fases e Artefatos

Concepção	Elaboração	Construção	Transição
▪ Documento de Visão ▪ Casos de Negócio ▪ Lista de Riscos ▪ Plano de Desenvolvimento de Software ▪ Plano de Iteração (para a primeira iteração) ▪ Casos de Desenvolvimento ▪ Ferramentas ▪ Glossário ▪ Modelo de Caso de Uso (identificação de atores e UCs mais importantes, fluxos de eventos para casos de uso críticos)	▪ Protótipo Arquitetural ▪ Lista de Riscos (revisada e analisada) ▪ Casos de Desenvolvimento ▪ Ferramentas ▪ Documento de Arquitetura ▪ Modelo de Projeto ▪ Modelo de Dados ▪ Modelo de Implementação ▪ Visão (refinada) ▪ Plano de desenvolvimento de software ▪ Plano de Iteração (p/ fase de construção e transição) ▪ Modelo de Casos de Uso (80% concluído) ▪ Especificações Suplementares ▪ Conjunto de Testes ▪ Modelo de Análise	▪ Software (componentes) ▪ Plano de Implantação ▪ Modelo de Implementação (expandido) ▪ Conjunto de testes (implementados e executados) ▪ Materiais de Treinamento (manuais de usuário, etc) ▪ Plano de Iteração (para a fase de transição concluído) ▪ Modelo de Projeto ▪ Caso de Desenvolvimento ▪ Ferramentas ▪ Modelo de Dados ▪ Documentos de apoio (manuais de usuário, instalação)	▪ Incremento de Software (*Build*) ▪ Notas de release ▪ Artefatos de instalação ▪ Material para treinamento ▪ Material de suporte ao usuário final ▪ Relatório de Testes (beta) ▪ Realimentação geral do usuário

Exemplo Iterativo Incremental/Evolutivo Ágil: Scrum



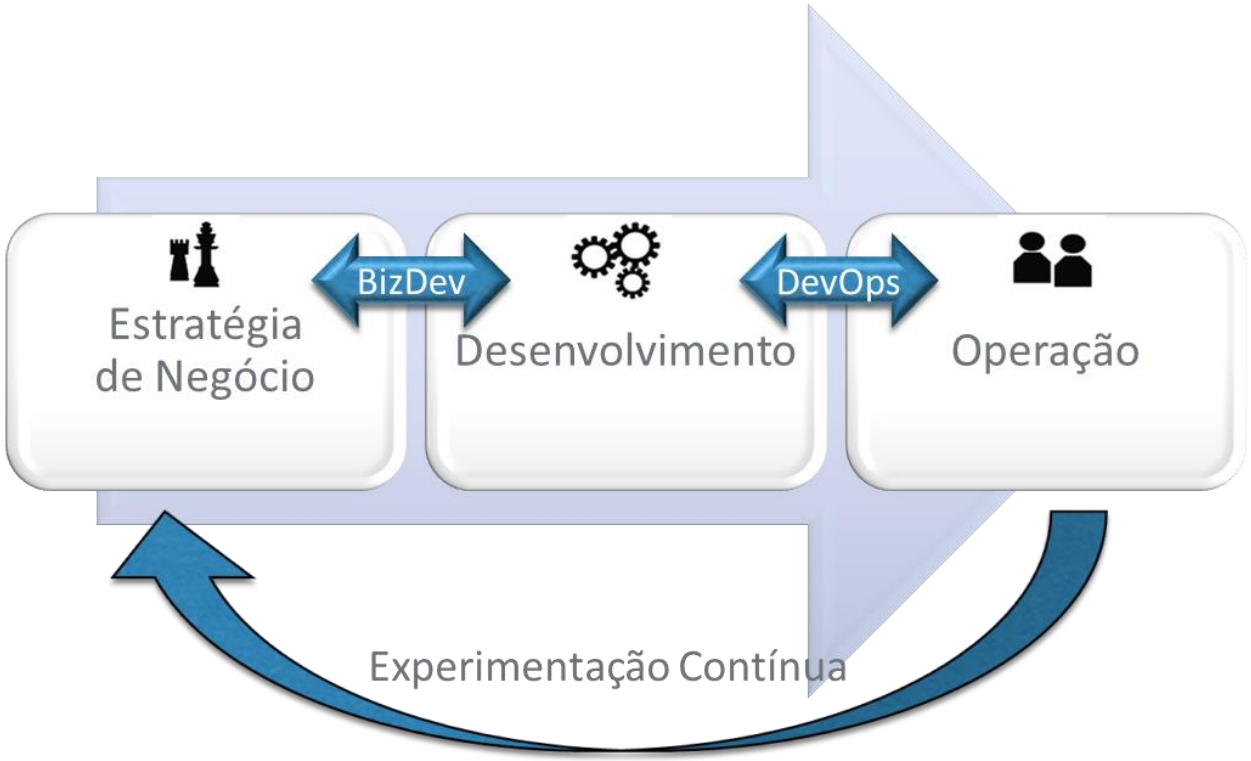
Scrum

- **Scrum** é um framework de gestão **iterativo e incremental/evolutivo** bastante aplicado no contexto de desenvolvimento ágil de software
- O ciclo de vida do SCRUM utiliza tempo fixo – ***sprints*** (ao invés de escopo fixo) para determinar seus incrementos
- O Scrum é facilitado por um ***Scrum Master***, que tem como função primária remover qualquer impedimento à habilidade de uma equipe de entregar o objetivo do *sprint*
- Cada *sprint* corresponde a uma iteração que entrega um incremento de software

Dinâmica do Modelo de Ciclo de Vida Associado ao Scrum

- Um ***backlog*** é conjunto de requisitos, priorizado pelo ***Product Owner*** (responsável por conhecer as necessidades do cliente)
- Entregas de um conjunto fixo de itens do *backlog* ocorrem em *sprints*
- Os itens do *backlog* para um *sprint* são definidos em uma sessão de planejamento
- Durante o *sprint*, ocorrem breves reuniões diárias, em que cada participante fala sobre o progresso conseguido, o trabalho a ser realizado e/ou o que o impede de seguir avançando
- Ao final de um *sprint*, ocorre a revisão e a retrospectiva, na qual todos os membros da equipe revisam a entrega e refletem sobre o *sprint* passado

Engenharia de Software Contínua



- **BizDev** é o alinhamento da estratégia de negócio com o desenvolvimento
- **DevOps** é o alinhamento do desenvolvimento de software com a implantação do software em operação

B. Fitzgerald, and K.J. Stol, “Continuous software engineering: A roadmap and agenda,” Journal of Systems and Software, vol. 123, pp.176-189, 2017.

Modelos de Referência de Processos e Corpos de Conhecimento em Engenharia de Software

Software Science Lab

Softex MpsBr

CMMI Capability Maturity Model Integration

ISO 12207

SWEBOK®

O Programa MPS.BR

Software Science Lab

- O objetivo do programa **MPS.BR** (Melhoria de Processo do Software Brasileiro) é o aumento da competitividade das organizações pela melhoria de seus processos

- Modelos de Referência MPS:
  - MPS-SW (MPS para Software)
  - MPS-SV (MPS para Serviços)
  - MPS-RH (MPS para Gestão de Pessoas)

Softex MpsBr

O Modelo de Referência MPS para Software

- Processos do MPS-SW:

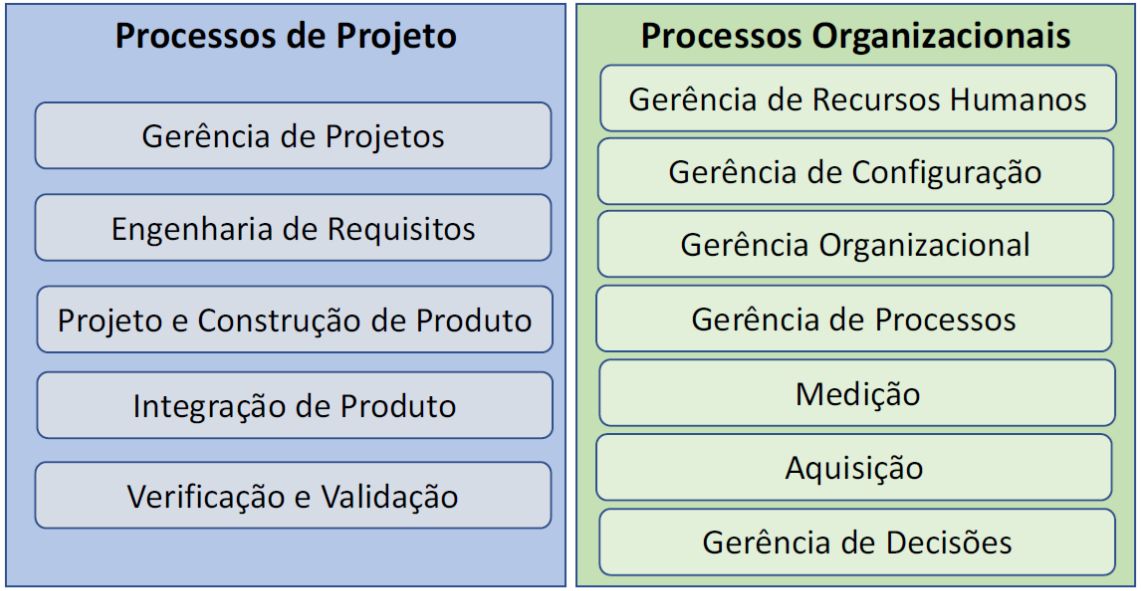


Figura 1 - Conjunto de Processos de Projetos e de Processos Organizacionais

SOFTEX, 2021. **Guia Geral MPS de Software**. Disponível em www.softex.br/mpsbr

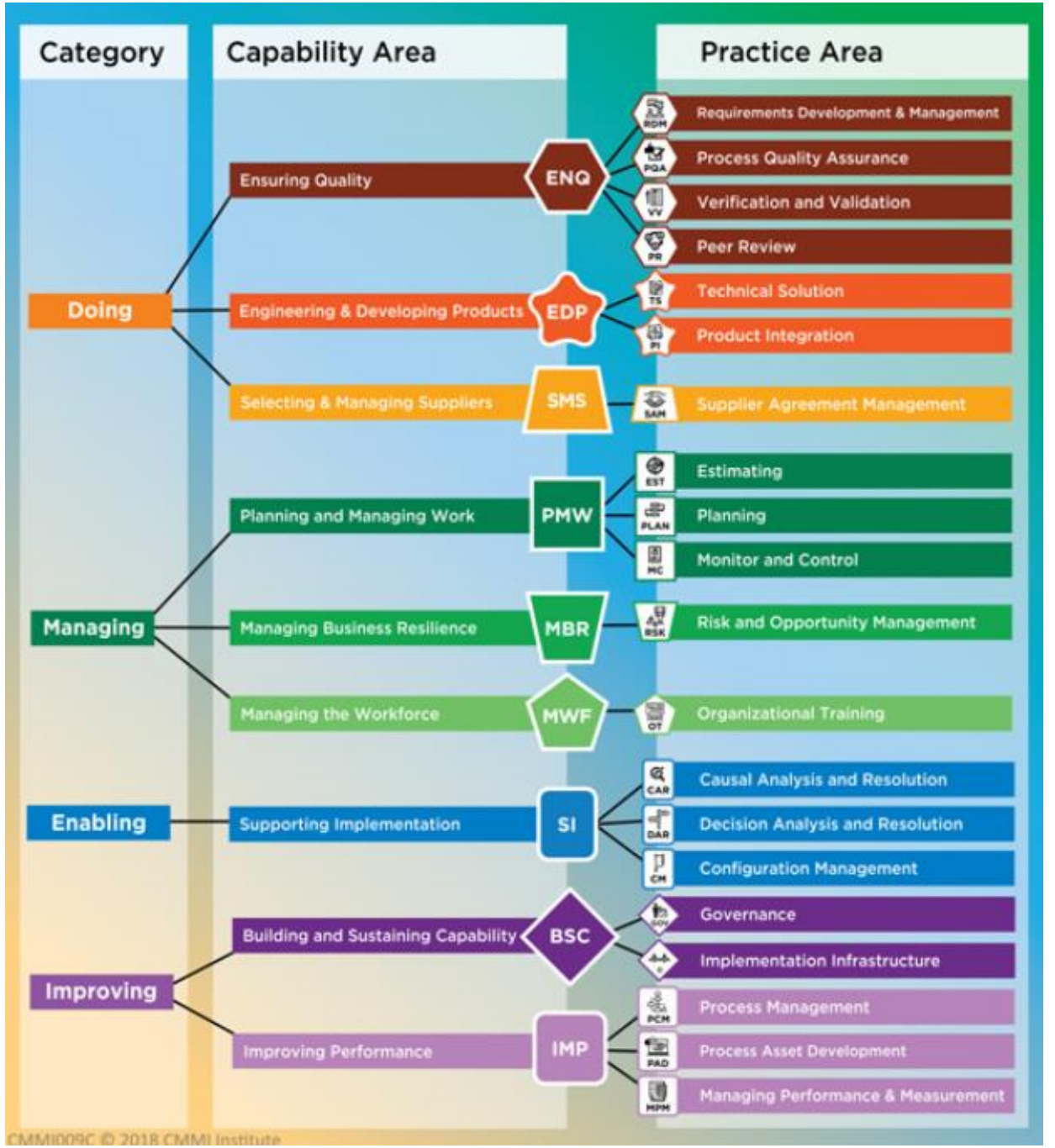
CMMI

- O **CMMI** (*Capability Maturity Model® Integration*) fornece às organizações elementos essenciais de processos eficazes

- Modelos da suíte de produtos CMMI:
  - CMMI-DEV
  - CMMI-SVC
  - CMMI-ACQ

CMMI

- ***Practice Areas*** (Processos) do CMMI:



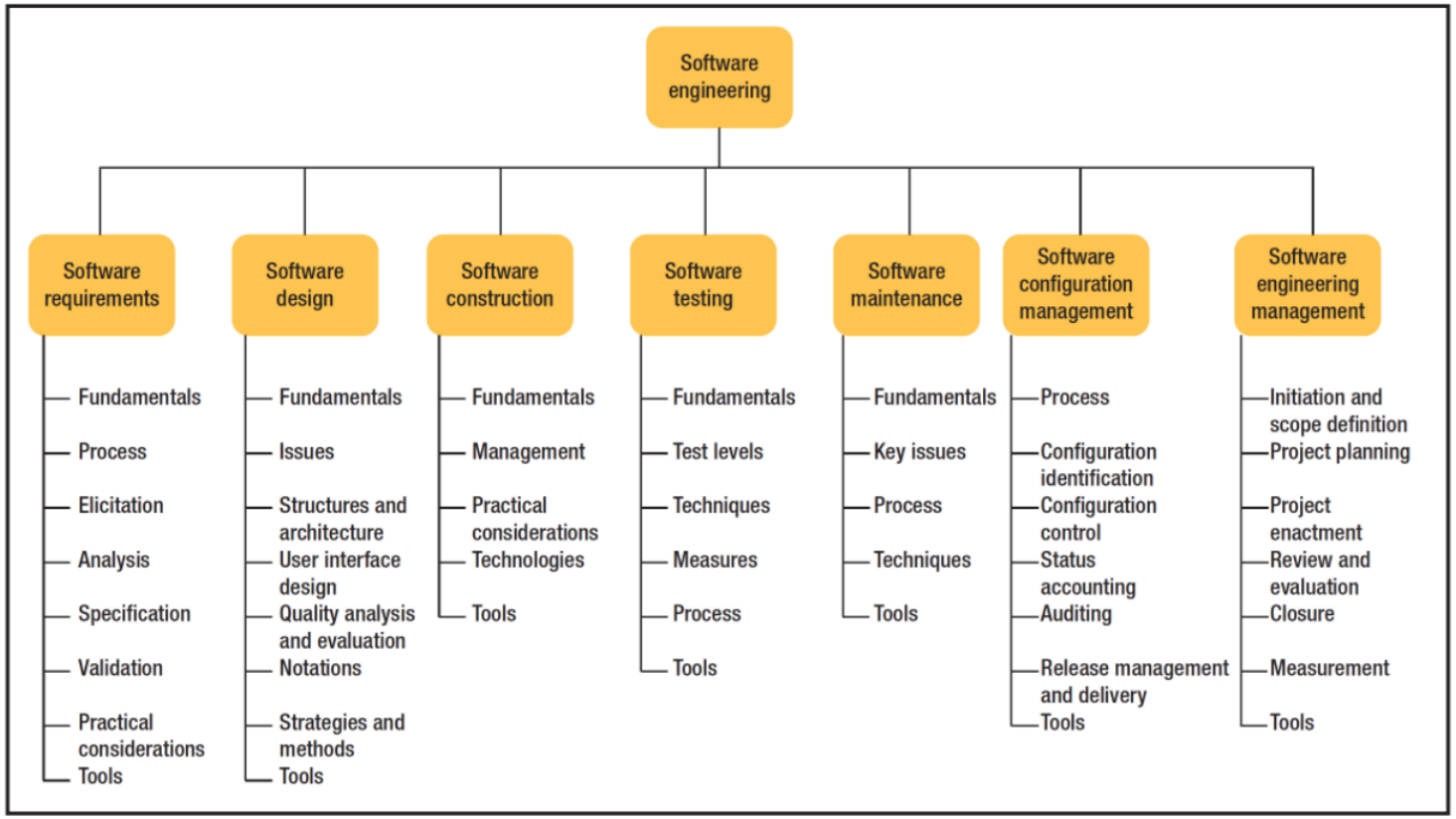
ISACA, 2018. **CMMI V2.0 Development Model**

SWEBOK

- O **Guia SWEBOK** (Guia do Conhecimento em Engenharia de Software) descreve conhecimento geralmente aceito sobre engenharia de software
- Suas 15 áreas de conhecimento (KAs) resumem os conceitos básicos e incluem uma lista de referências que aponta para informações mais detalhadas

SWEBOK

Table I.1. The 15 SWEBOK KAs
Software Requirements
Software Design
Software Construction
Software Testing
Software Maintenance
Software Configuration Management
Software Engineering Management
Software Engineering Process
Software Engineering Models and Methods
Software Quality
Software Engineering Professional Practice
Software Engineering Economics
Computing Foundations
Mathematical Foundations
Engineering Foundations



P. Bourque and R.E. Fairley, eds., **Guide to the Software Engineering Body of Knowledge**, ver. 3.0, IEEE CS, 2014; www.swebok.org.

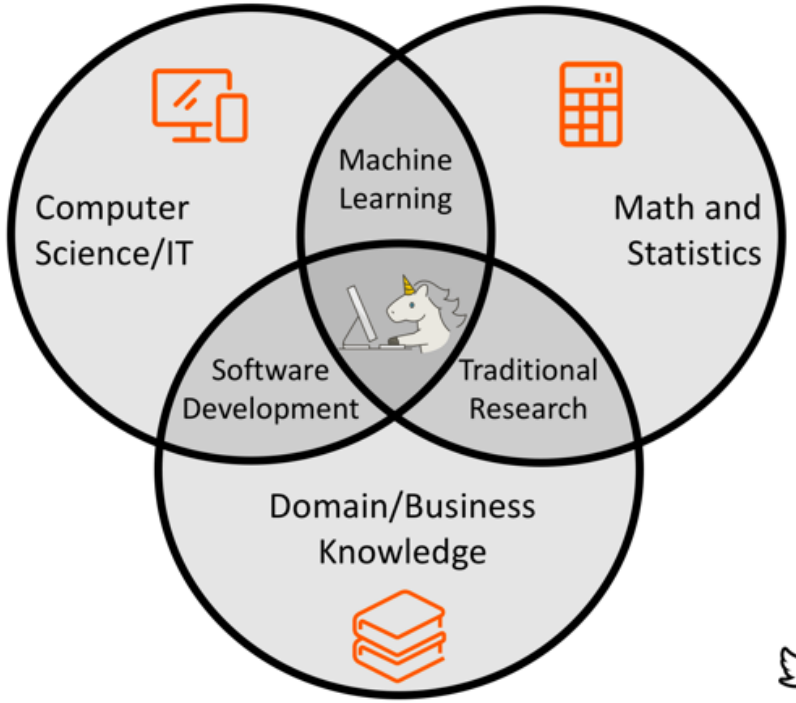
Leituras Sugeridas

- Valente, M.T., 2020. **Engenharia de software moderna: princípios e práticas para desenvolvimento de software com produtividade.** (Capítulos 1 e 2)
- Boehm, B. and Turner, R., 2003. **Balancing agility and discipline: A guide for the perplexed.** Addison-Wesley Professional.
- Kuhrmann, M., Tell, P., Hebig, R., Klunder, J.A.C., *et al.*, 2021. **What Makes Agile Software Development Agile.** *IEEE Transactions on Software Engineering*.
- SOFTEX, 2021. **Guia Geral MPS de Software**. Disponível em www.softex.br/mpsbr
- CMMI Institute, 2018. **CMMI V2.0 Development Model.**
- Bourque, P. and Fairley, R.E. (*eds.*), 2014. **Guide to the Software Engineering Body of Knowledge**, ver. 3.0, IEEE CS. www.swebok.org.
- B. Fitzgerald, and K.J. Stol, 2017. **Continuous software engineering: A roadmap and agenda.** *Journal of Systems and Software*, vol. 123, pp.176-189.

Software Science Lab

Introdução à Ciência de Dados

Engenharia de Software para Ciência de Dados



bouvet

@markawest

Software Science Lab

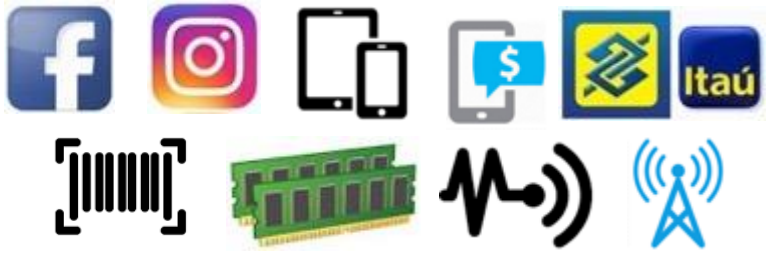
Um bom cientista de dados precisa...

Software Science Lab

Definir o problema → Preparar os dados → Avaliar um conjunto de algoritmos → Realizar ajustes para melhorar os resultados → Finalizar o modelo e apresentar os resultados

Bando de Dados?

- **Cenário atual:** criação e crescimento de inúmeras bases de dados, em fluxo contínuo
- Em 2017, Cerca de **90%** dos dados armazenados na web tinham sido gerados nos últimos **2 anos** e em 2020, a internet já alcançava **59%** da população mundial



"Decisões baseadas em emoções não são decisões, são instintos."

Motivação

Necessidade: processar e obter informação útil a partir dos dados

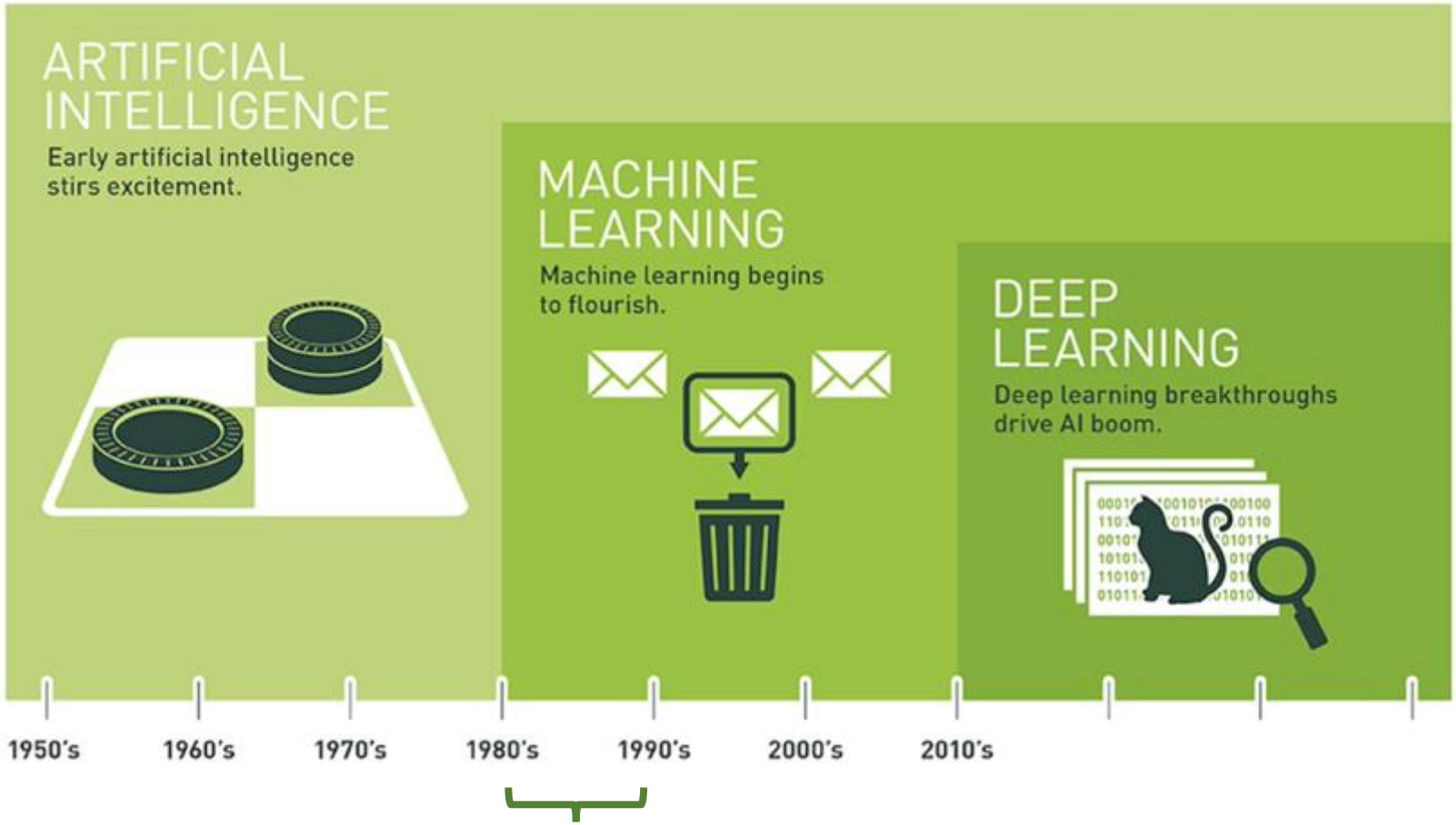
Problema: inviável realizar processamento e análise manualmente

Solução: automatização de tarefas, simulando o comportamento humano

Big Data

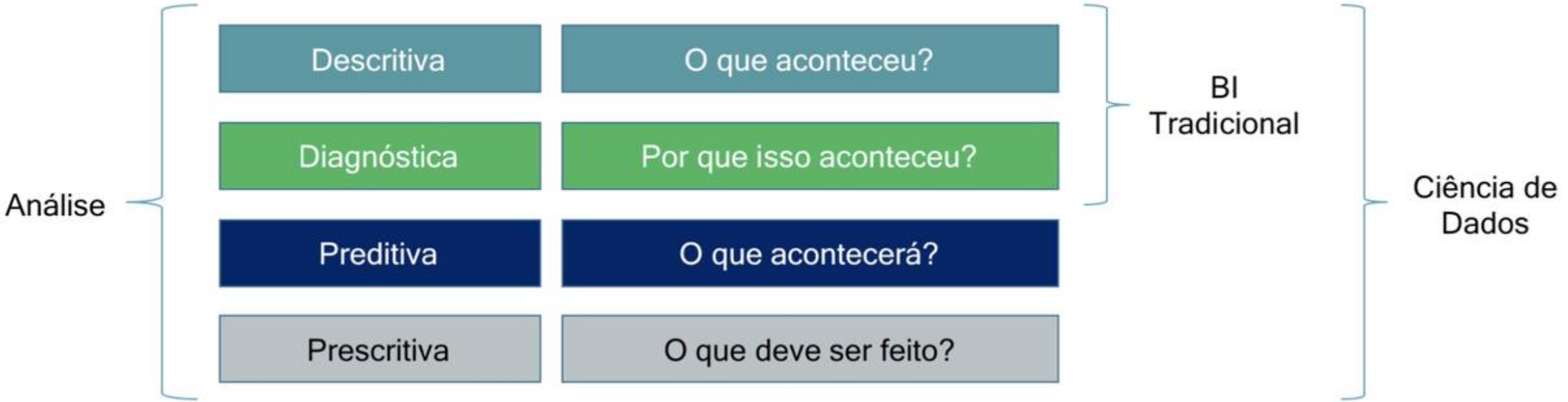
Inteligência Artificial

Terminologia



- Visão Computacional
- Processamento de Linguagem Natural

Business Intelligence e Ciência de Dados



Ciência de Dados é algo novo?

A maioria das técnicas é **antiga** e proveniente da **Estatística**, mas só passaram a ser efetivamente usadas para exploração de dados nos **últimos anos**.

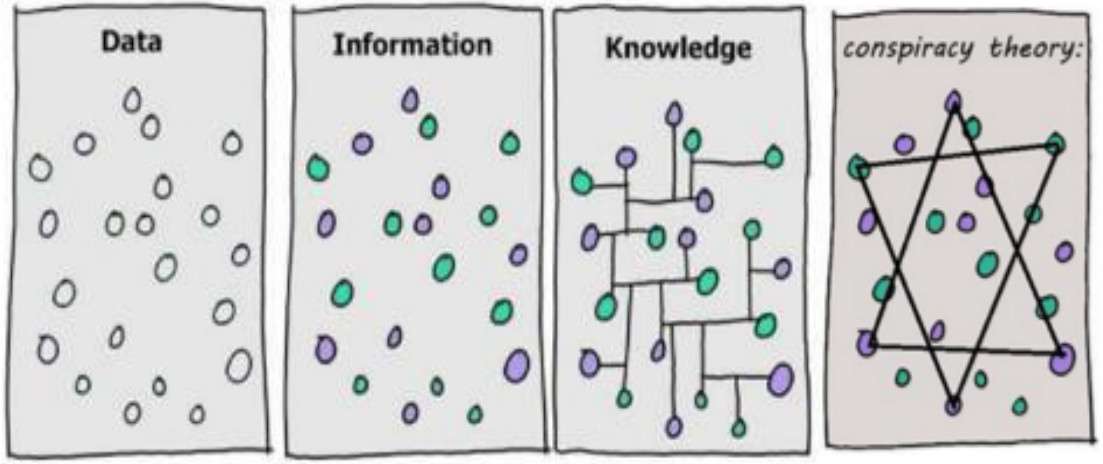
Big Data

Mas por quê?

- Volume de dados disponível
- *Data warehouses* (organização voltada para decisão)
- Recursos computacionais potentes
- Competição empresarial
- Softwares especializados
- Dados como ativos

Dados x Informação x Conhecimento

- O valor dos dados armazenados está ligado à capacidade de se **extrair conhecimento** de alto nível a partir deles: transformar **dados brutos** em **informações úteis**



⍰ CUIDADO!

Knowledge Discover in Databases (KDD)

Software Science Lab

"Um **processo** de várias etapas, **não trivial**, **interativo** e **iterativo**, para identificação de **padrões compreensíveis**, **válidos**, **novos** e **potencialmente úteis** a partir de grandes conjuntos de **dados**" [FAYAD et al., 1996]

Terminologia mais antiga, pode ser considerada uma parte de Ciência de Dados

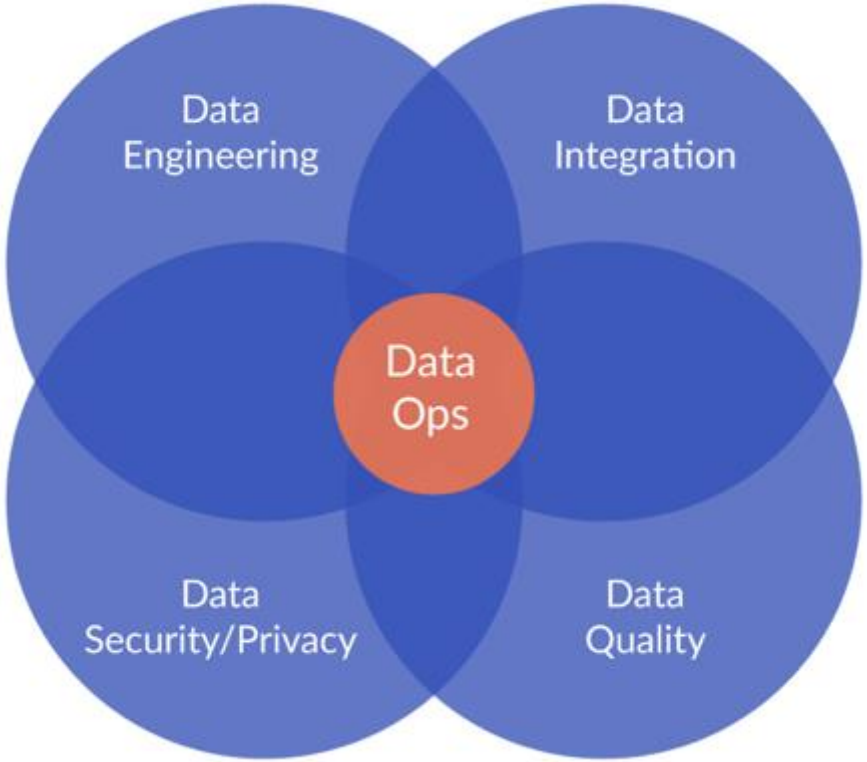
- Deve ser norteado por **objetivos** (definições da tarefa a ser executada e expectativa dos conhecedores do domínio): taxa de erro aceitável, necessidade do modelo ser transparente para os especialistas, natureza das variáveis envolvidas, etc.

Tipos de Dados

DataOps

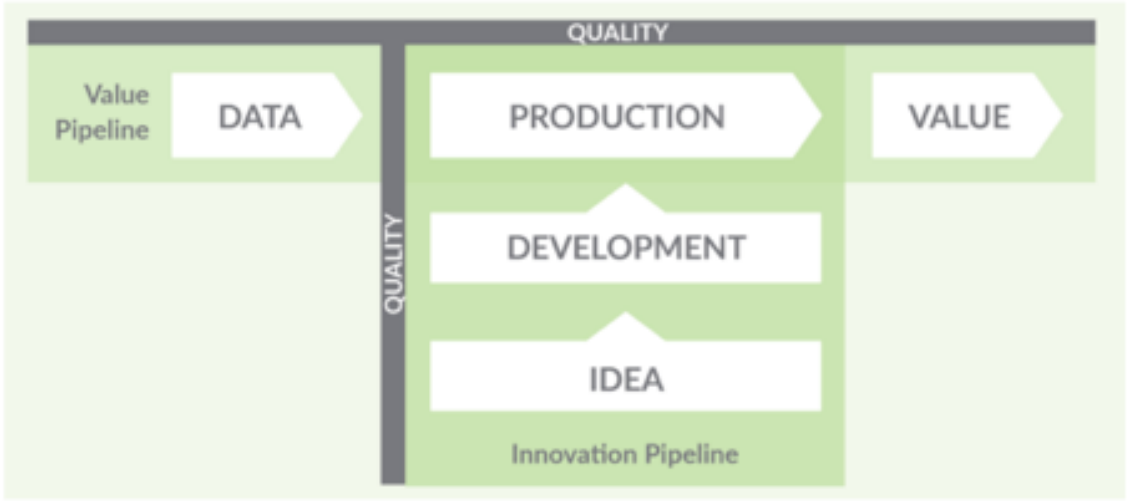
- Infraestrutura necessária para suportar a **quantidade**, **velocidade** e **variedade** dos dados disponíveis, diferente das abordagens de gerenciamento de dados tradicionais

- Deve saber lidar com fontes de dados diferentes e dados **gerados em tempo real**



DataOps

DevOps

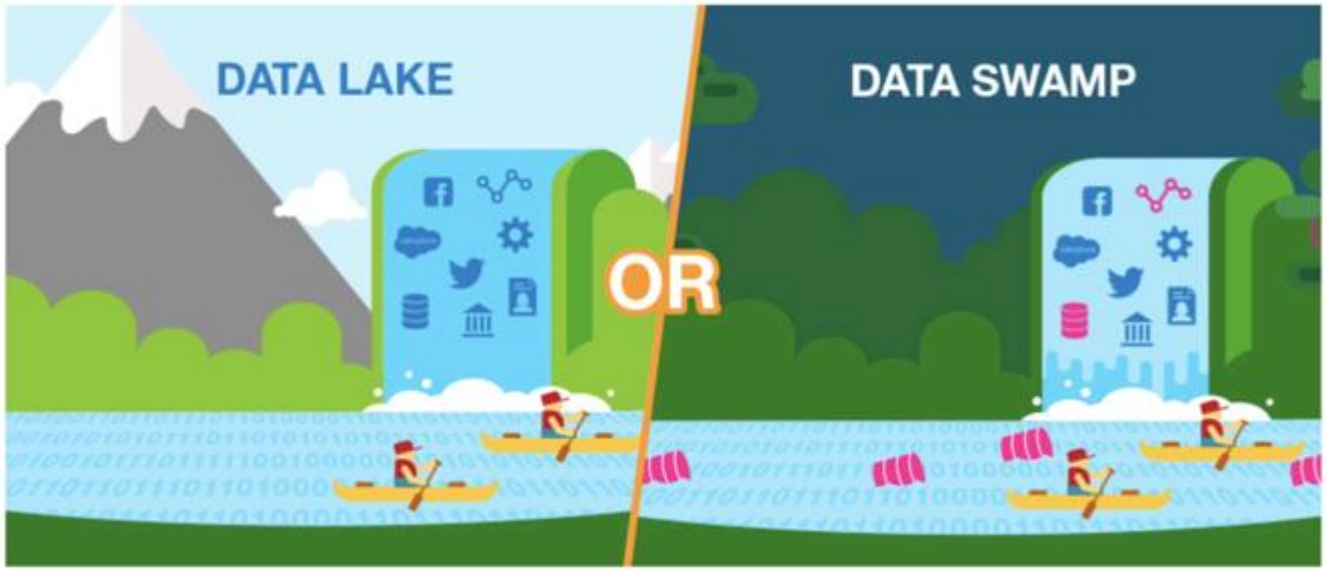


DataOps

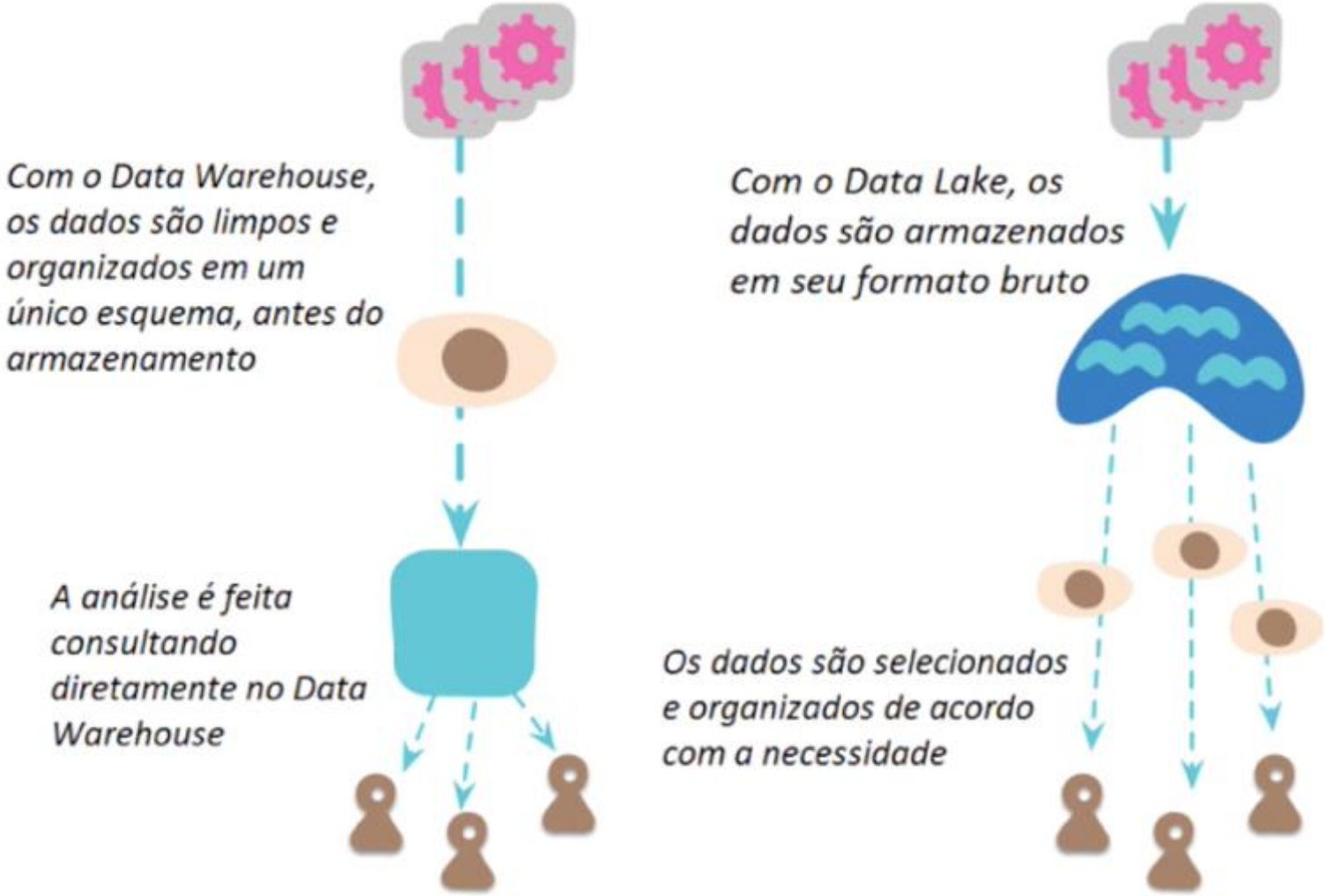
Leitura recomendada:
https://medium.com/data-ops/dataops-is-not-just-devops-for-data-6e03083157b7

Data Lake

- Um único repositório de dados **brutos** na empresa, disponível a qualquer pessoa que queira analisá-los
- Trabalha com processamento e armazenamento **distribuído** → Big Data já não cabe em uma única máquina!
- Necessidade de gestão e organização de dados para que o lago não vire um **pântano**



Data Lake x Data Warehouse



Aprendizado

Capacidade de se adaptar, modificar e melhorar seu comportamento e suas respostas, sendo uma das propriedades mais importantes dos seres inteligentes (humanos ou não).

- Diz-se que se está **aprendendo** (treinando, construindo, formulando ou induzindo um modelo de conhecimento) a partir de um conjunto de dados quando se procura por padrões nestes dados
- Quando se faz uma estimativa (teste, predição) dos valores desconhecidos para atributos do conjunto de dados, diz-se que o modelo está sendo **aplicado**

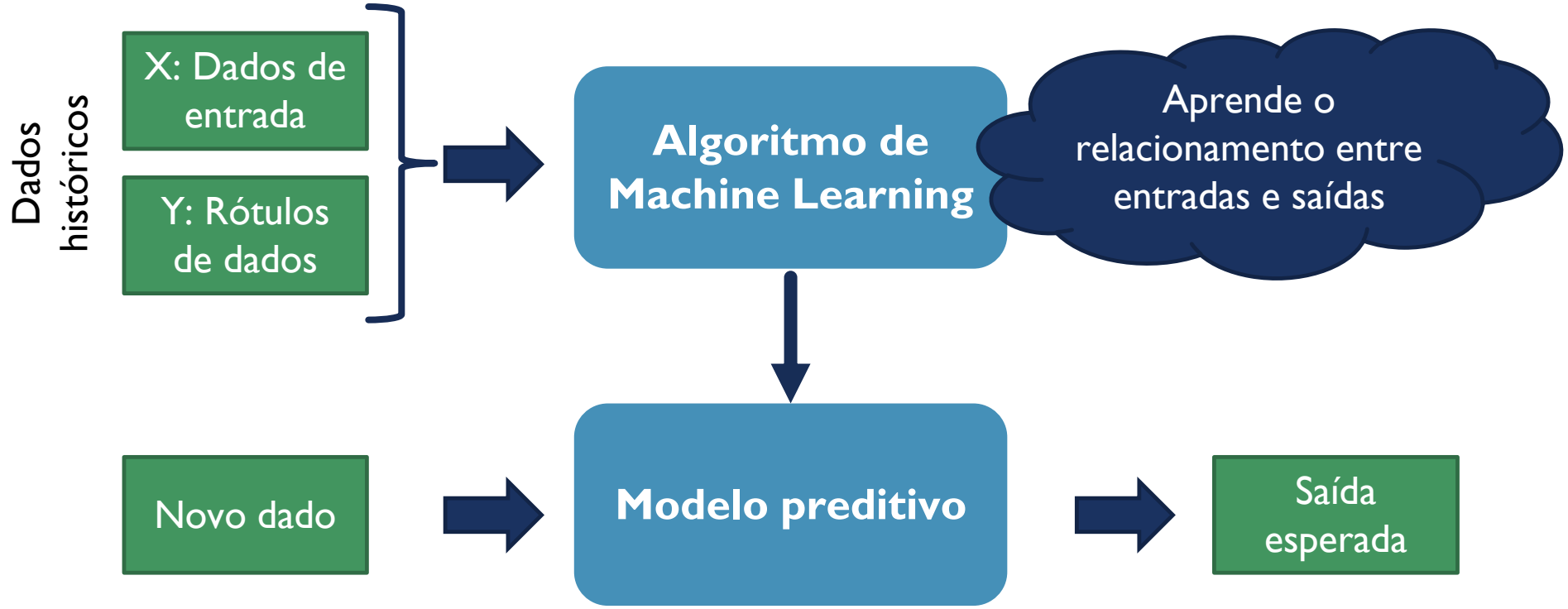
Aprendizado Supervisionado

- O modelo de conhecimento é construído a partir dos dados (dataset) apresentados na forma de **pares ordenados** (entrada - saída desejada) rotulados

- Apresentamos para o algoritmo um número suficiente de **exemplos** (também chamados de registros ou instâncias) de entradas e saídas desejadas (já rotuladas previamente). O objetivo do algoritmo é aprender uma **regra geral** que mapeie as entradas nas saídas corretamente

- Os **dados de entrada** podem ser divididos em dois grupos:
  - **X**, com os atributos (também chamados de características) a serem utilizados na determinação da classe de saída (também chamados de atributos previsores ou de predição)
  - **Y**, com o atributo para o qual se deseja fazer a predição do valor de saída categórico ou numérico (também chamado de atributo-alvo ou target)

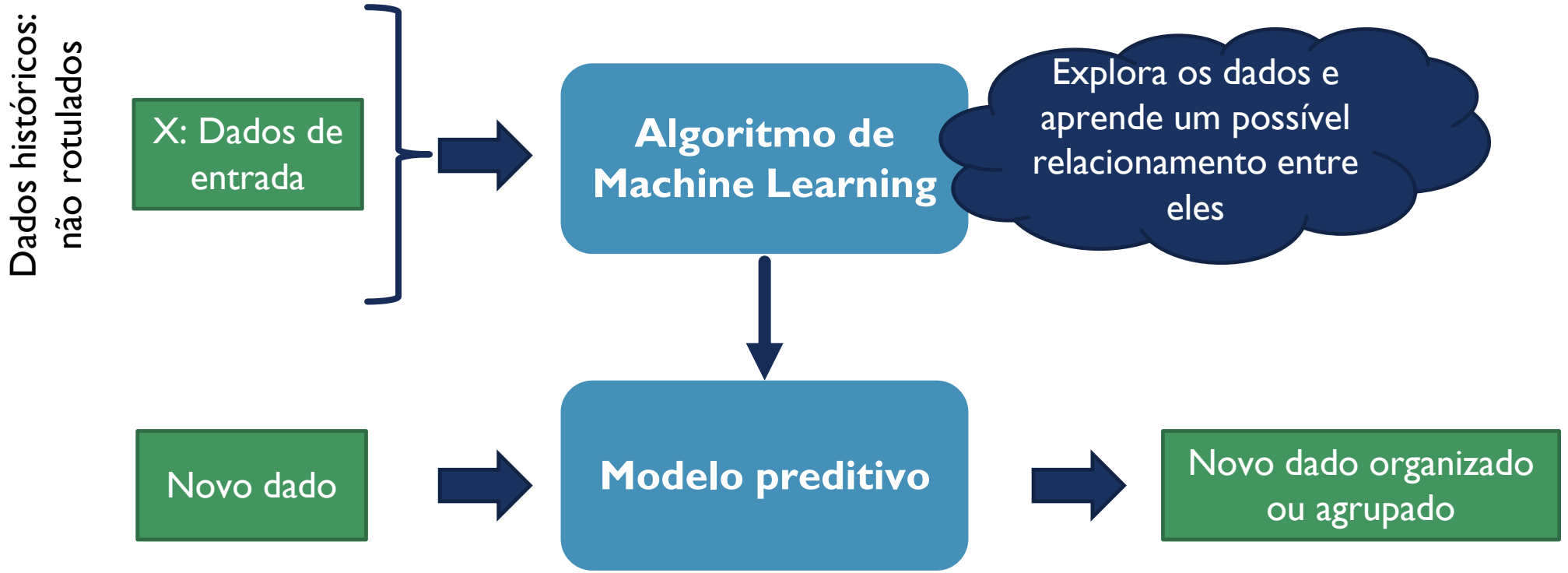
Aprendizado Supervisionado



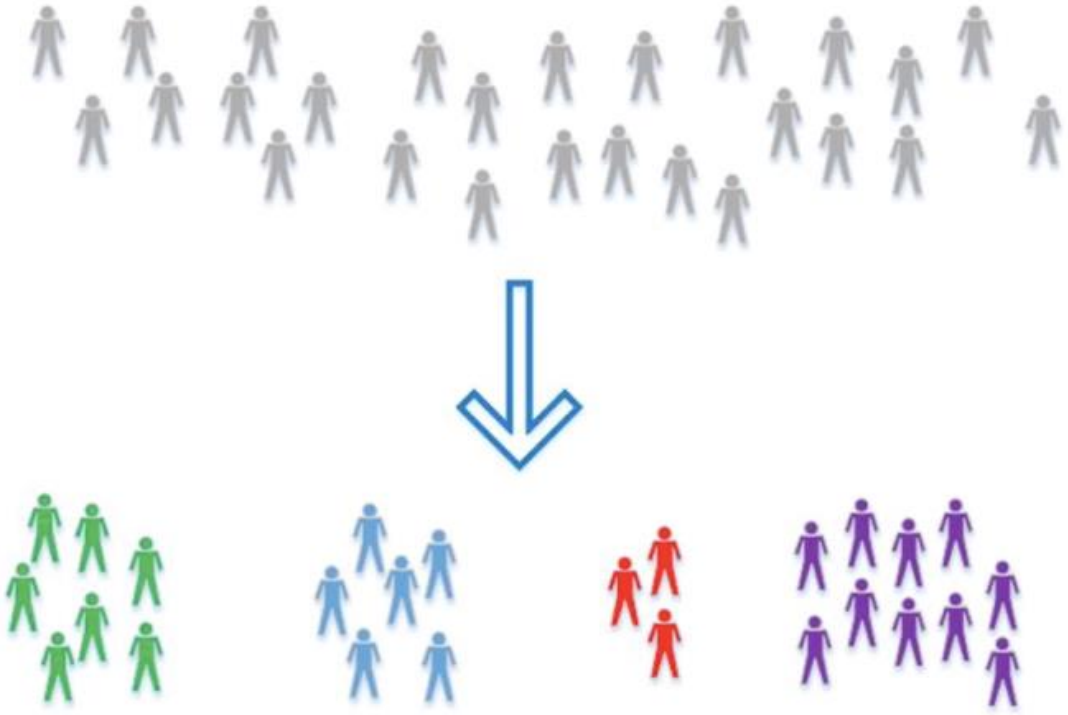
Aprendizado Não-Supervisionado

- Não existe a informação dos **rótulos históricos**, ou seja, não temos as saídas desejadas a serem estimadas
- O algoritmo não recebe durante o treinamento os possíveis resultados e ele deve **descobrir por si só**, explorando os dados e encontrando um possível relacionamento entre eles
- O processo de aprendizado busca identificar **regularidades** entre os dados a fim de agrupá-los ou organizá-los em função das similaridades que apresentam entre si
- Não há necessidade de realizar particionamento em conjuntos de treino e teste

Aprendizado Não-Supervisionado



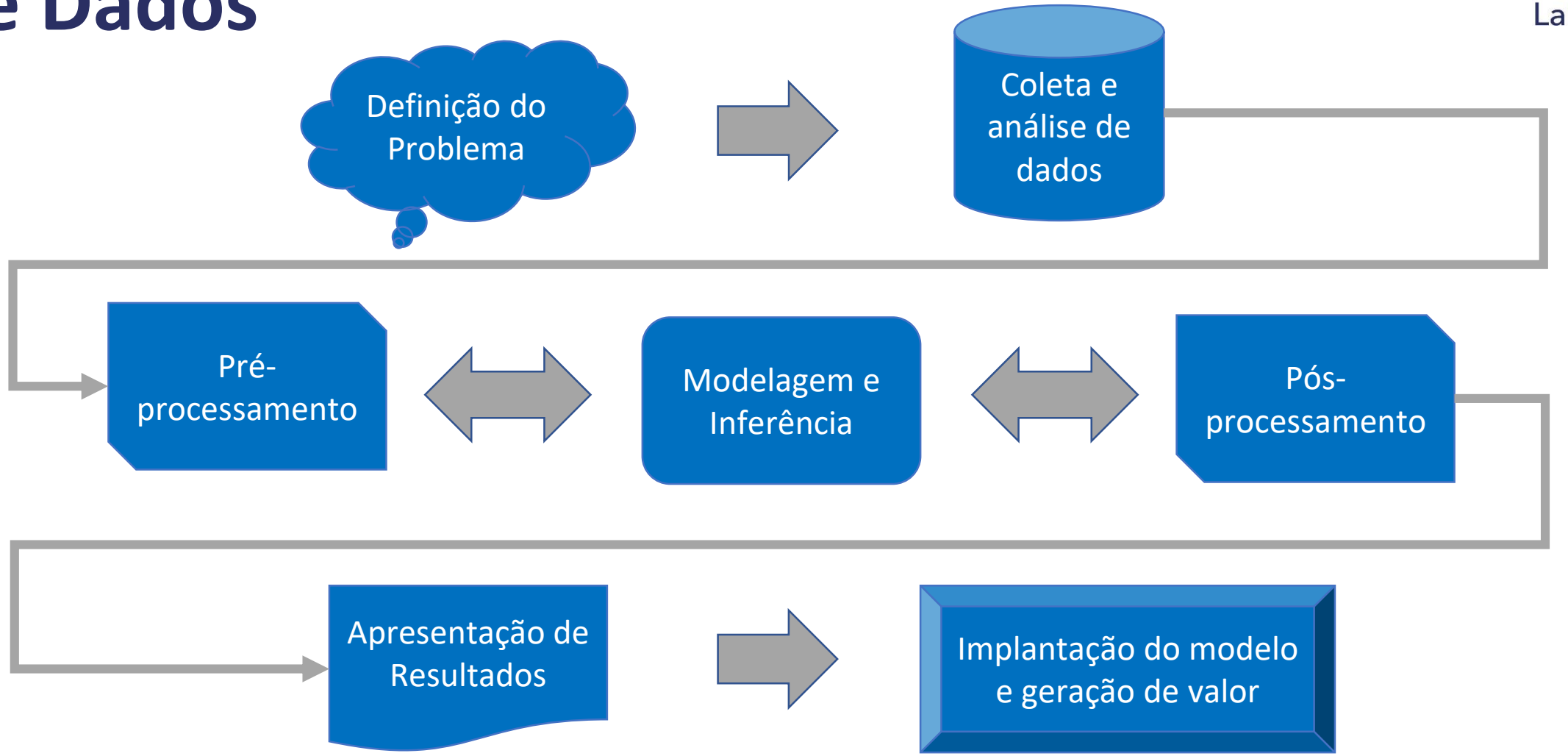
Aprendizado Não-Supervisionado



Ciclo de Vida de Projetos de Ciência de Dados

1. **Entender o problema e definir objetivos** - *Que problema estou resolvendo?*
2. **Coletar e analisar os dados** - *De que informações preciso?*
3. **Preparar os dados** - *Como preciso tratar os dados?*
4. **Construir o modelo** - *Quais são os padrões nos dados que levam a soluções?*
5. **Avaliar e criticar o modelo** - *O modelo resolve meu problema?*
6. **Apresentar resultados** - *Como posso resolver o problema?*
7. **Distribuir o modelo** - *Como resolvo o problema no mundo real?*

Esquema Básico de um Projeto de Ciência de Dados



Etapa 1-2: Problema

1. **Elencar** as perguntas dos gestores, Requisitos funcionais e não-funcionais
2. **Identificar** as variáveis que desejam ser preditas ou descritas, assim como as que possivelmente são relacionadas
3. **Classificar** cada pergunta em um dos tipos de problemas de CD

1. **Verificar** a disponibilidade das variáveis elencadas na etapa anterior
2. **Modelar** (se não existir) o DW/Data Mart, definir o processo de ETL e integrá-lo a uma ferramenta
3. **Analisar** os dados

Etapa 3-5: Recursos

Software Science Lab

Etapa mais demorada e trabalhosa

Pré-processamento

1. **Remover ou inputar** dados faltantes e **tratar** dados inconsistentes
2. **Corrigir ou amenizar** *outliers* e desbalanceamento entre classes
3. **Selecionar** as variáveis e instâncias para compor o(s) modelo(s)

Modelagem e Inferência

1. **Elencar** os modelos possíveis e passíveis para cada tipo de problema
2. **Estimar** os parâmetros que compõem os modelos, baseando-se nas instâncias e variáveis pré-processadas
3. **Avaliar** os resultados de cada modelo, usando métricas e um processo justo de comparação

Pós-processamento

1. **Combinar** heurísticas de negócio com os modelos ajustados
2. **Pós-avaliar** tendo em vista os pontos fortes e dificuldades na implementação de cada um dos modelos

Etapa 6-7: Resultados

Software Science Lab

Apresentação de Resultados

1. **Relatar** a metodologia adotada para endereçar a solução às demandas dos gestores
2. **Comparar** os resultados do melhor modelo com o *benchmark* atual (caso haja)
3. **Planejar** os passos para a implantação da solução proposta

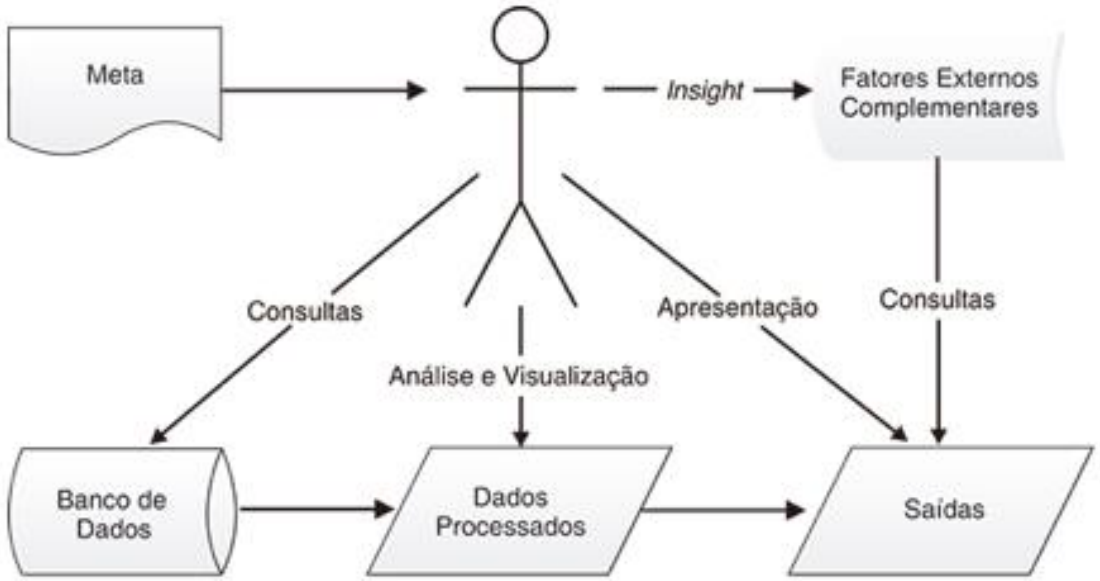
Implantação do modelo e geração de valor

1. **Implantar** o modelo em produção
2. **Calcular** os ganhos qualitativos (ganhos operacionais e de recursos humanos) e quantitativos (ROI e outras métricas)
3. **Monitorar** o modelo implantado

Papel do Ser Humano

Software Science Lab

Formulação precisa dos objetivos	Escolha de algoritmos
Escolha de técnicas de pré-processamento de dados	Parametrização de algoritmos
Experiência e Conhecimento	Intuição



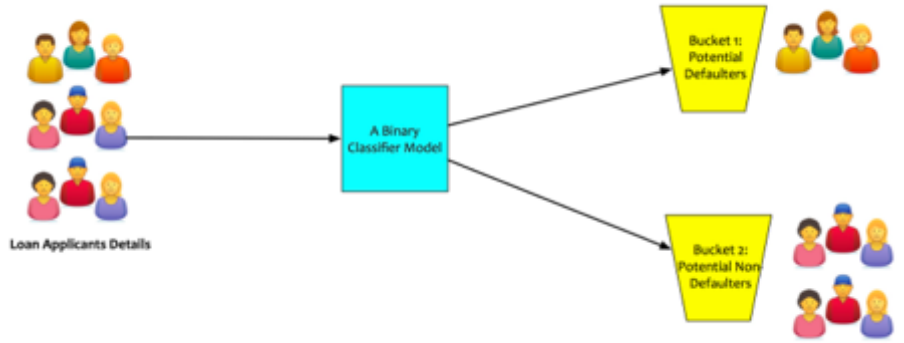
O **sentimento do especialista** não pode ser dispensado, mesmo que as mais sofisticadas técnicas sejam utilizadas.

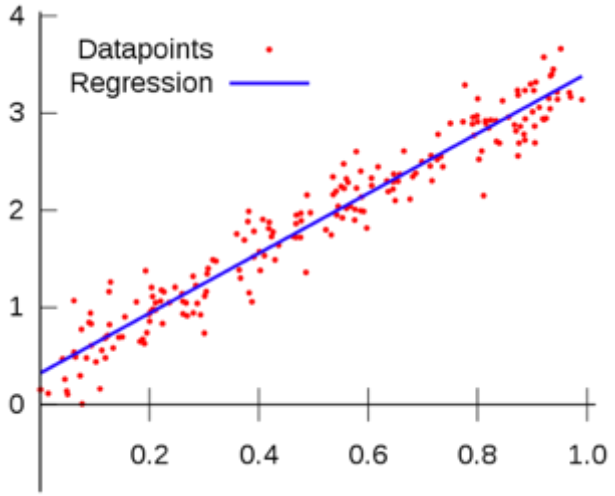
Tipos de Problemas de Ciência de Dados

Software Science Lab

SUPERVISIONADOS

- **Classificação:** Detecção de Clientes com Perfis Fraudulentos
- **Regressão:** Predição do Valor das Vendas em uma nova Filial





Problema de Classificação

Software Science Lab

Detecção de clientes com perfis fraudulentos

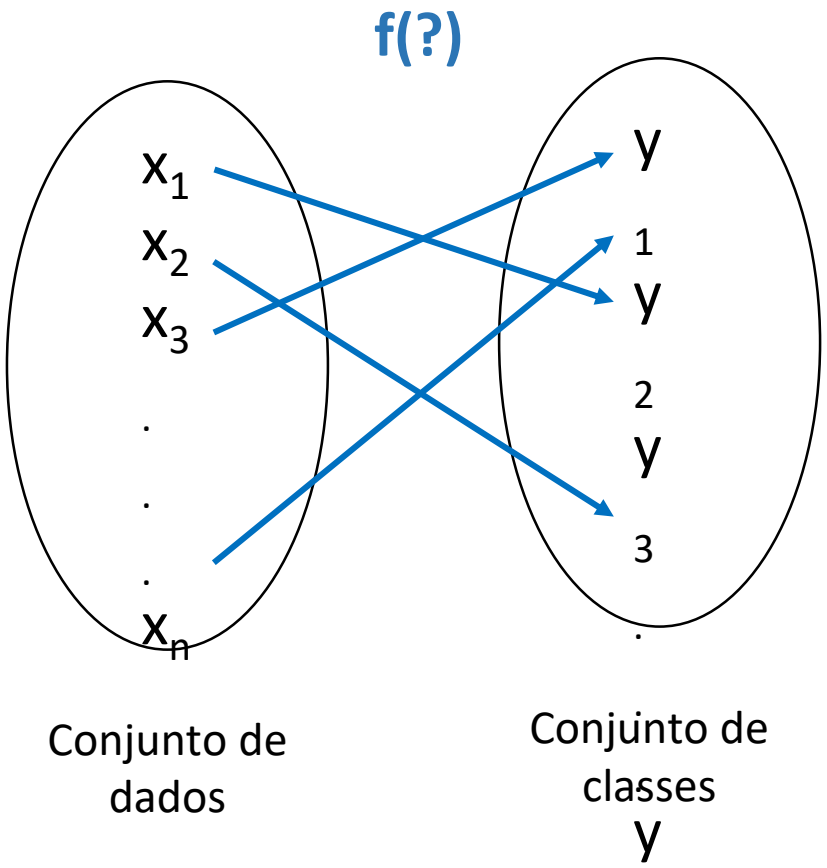
Cliente A deseja obter um empréstimo de R$ 1000,00

Pergunta do Gestor:

- Será que este cliente vai pagar o empréstimo?

Problema de Classificação

- Busca por uma **função matemática** que permita associar corretamente cada exemplo x_i de um conjunto de dados a um único rótulo categórico, y_i, denominado **classe**
- Uma vez identificada, esta função pode ser aplicada a novos exemplos para **prever** as classes em que eles se enquadram



Problema de Classificação

Software Science Lab

1

A partir de uma base de dados **rotulada**, geram-se dois subconjuntos disjuntos: a **base de treino** (i.e. 70% dos dados originais) e a **base de teste** (i.e. 30% dos dados originais).

2

A base de **treino** é submetida ao classificador para treinamento do modelo, que é calibrado de acordo com os dados apresentados.

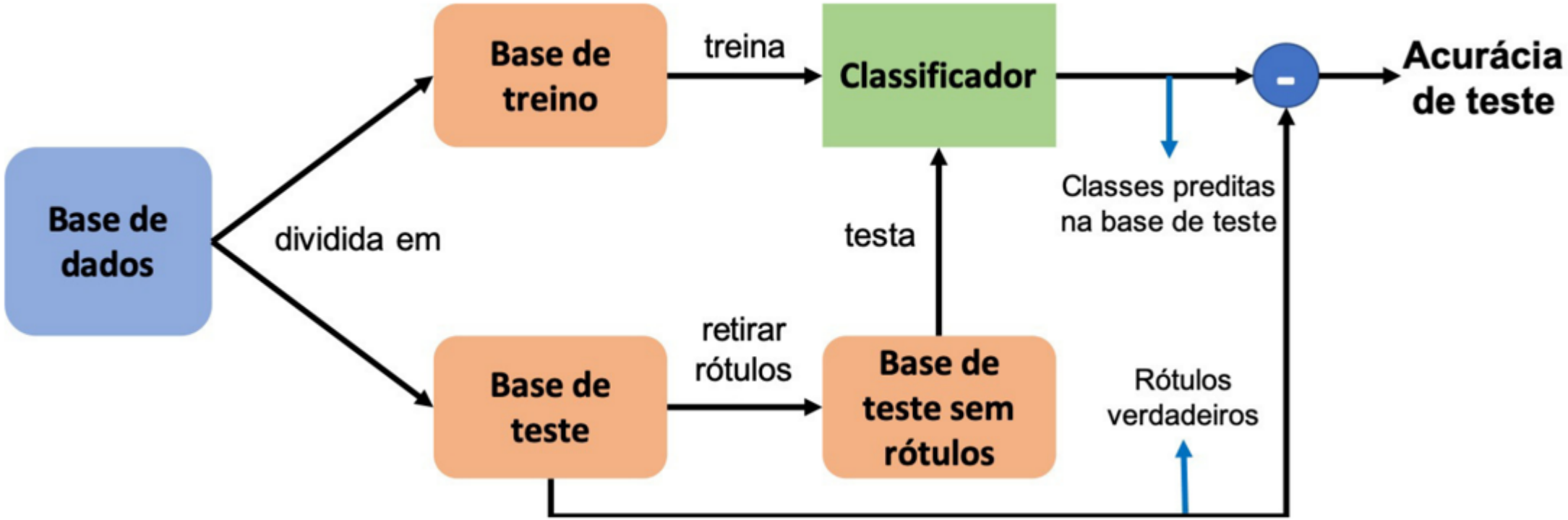
3

Apresentam-se os exemplos da base de **teste** para o modelo, que deverá realizar a predição das classes dos mesmos.

4

Comparando-se as classes preditas com as classes verdadeiras da base de **teste**, pode-se medir a qualidade do modelo (sua habilidade em classificar corretamente exemplos não vistos durante o treinamento).

Problema de Classificação



Problemas

- Uma vez identificada uma hipótese (classificador), esta pode ser **muito específica** para o conjunto de treinamento utilizado
- Caso este conjunto não corresponda a uma amostra suficientemente representativa da população, o classificador pode ter um bom desempenho no conjunto de **treinamento**, mas não no conjunto de **teste**

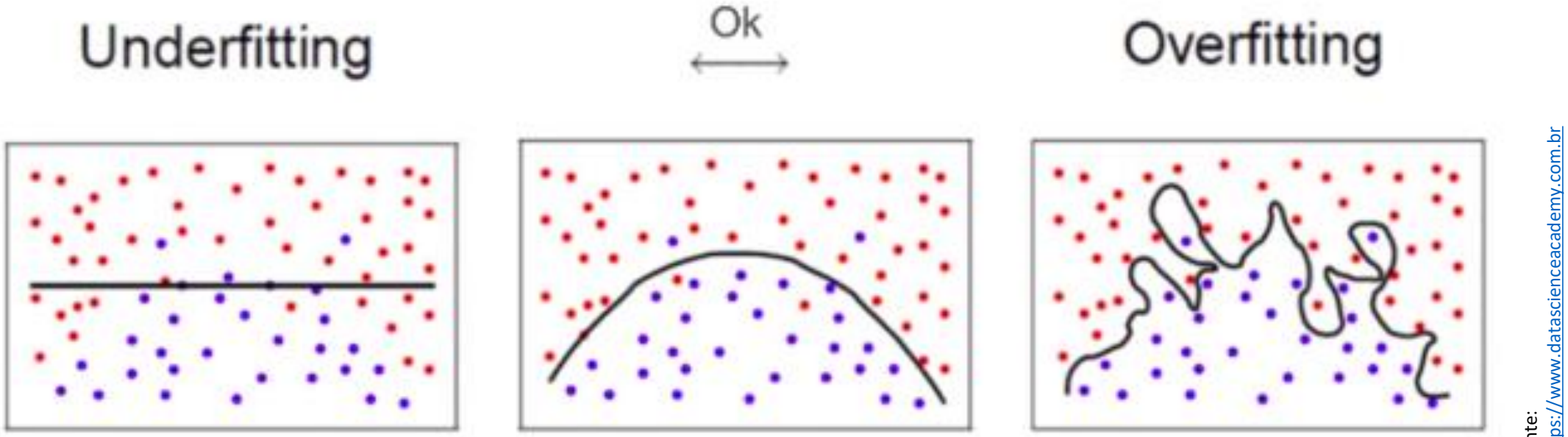
Problemas

- ***Overfitting:*** o classificador se ajustou em excesso ao conjunto de treinamento
- Geralmente, o erro de generalização (teste) é maior que o erro de treinamento. Idealmente, estes erros são próximos.
- Se o erro de generalização é grande demais, pode estar ocorrendo ***overfitting***: o modelo está memorizando os padrões de treinamento em vez de descobrir regras ou padrões generalizados

Problemas

- Outra situação que pode acontecer é se o algoritmo de aprendizado tiver parametrizações inadequadas
- Neste caso, diz-se que ocorreu ***underfitting***: o classificador se ajustou pouco ao conjunto de treinamento, não sendo adequado para realizar predições no conjunto de teste

Problemas



Fonte: https://www.datascienceacademy.com.br

Underfitting: modelo com falta de complexidade para o problema

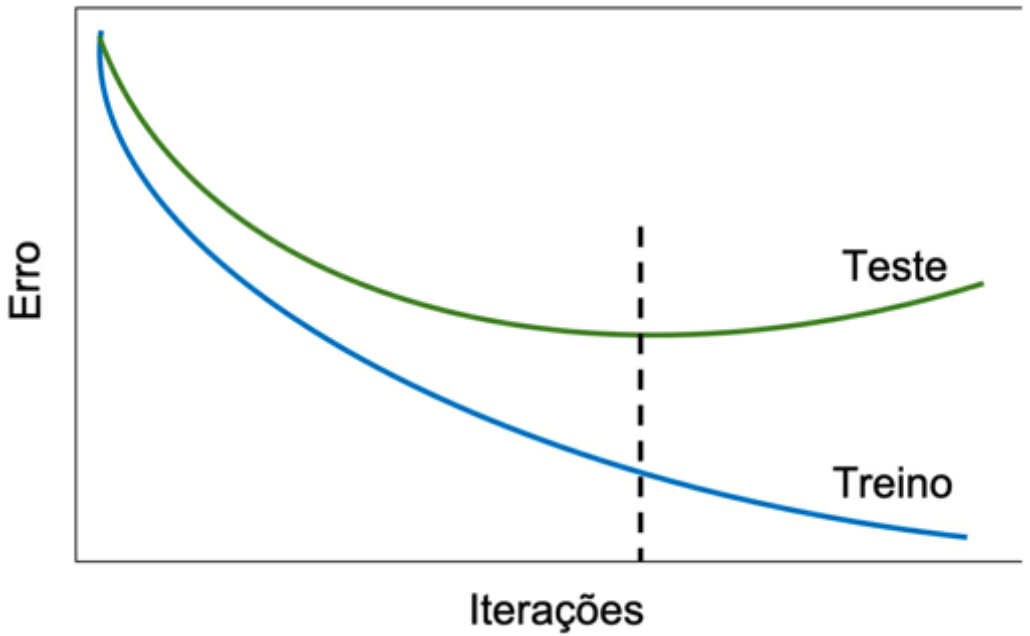
Modelo adequado para o problema

Overfitting: modelo com excesso de complexidade para o problema

Dilema Bias x Variância (Flexibilidade x Qualidade)

Software Science Lab

- O modelo preditivo precisa ser suficientemente **flexível** para aproximar os dados de treinamento, mas deve **evitar** que o modelo absorva os **ruídos** da base
- O modelo deve **capturar** as regularidades dos dados de treinamento, mas também **generalizar** bem para dados desconhecidos
- **Solução:** escolher o momento adequado para interromper o treinamento e/ou utilizar validação cruzada



Problema de Regressão

Predição de Faturamento

Pergunta do Gestor:

- Qual o faturamento esperado para o próximo trimestre?

Problema de Regressão

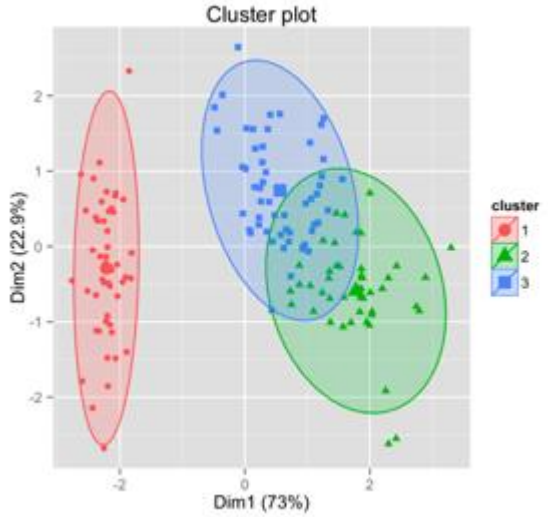
- Em vez do resultado ser categórico, como é na classificação, o resultado é **numérico** (contínuo ou discreto)
  - Por exemplo, um problema de classificação seria "Conceder ou não crédito para um cliente?" enquanto que um problema de regressão seria "Conceder qual valor de crédito para um cliente?"
- A **saída** de um modelo de regressão é um valor **numérico** que deve ser o mais próximo possível do valor desejado, e a diferença entre esses valores fornece uma medida de erro de estimação do algoritmo

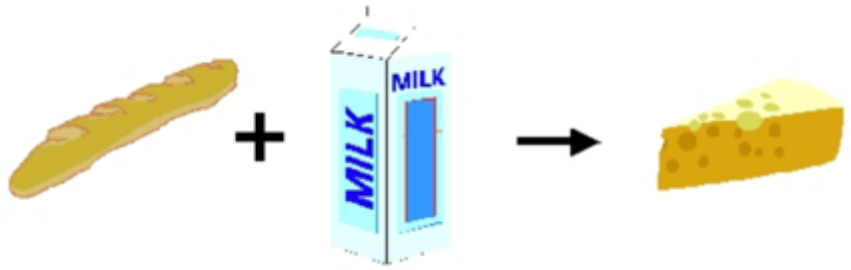
Tipos de Problemas de Ciência de Dados

Software Science Lab

NÃO-SUPERVISIONADOS

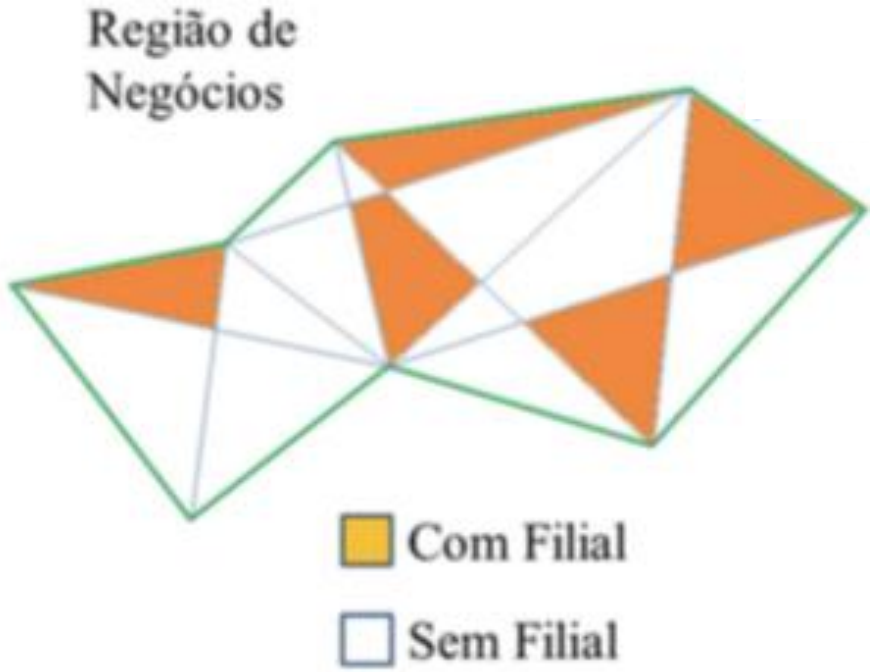
- **Agrupamento (Clustering):** Determinação de localidades promissoras para abertura de novas Filiais
- **Associação:** Oferta de novos serviços e produtos (Netflix, Amazon, Americanas.com, etc.)





Problema de Agrupamento

Localidades promissoras para novas filiais



Pergunta do Gestor:

- Qual o local mais promissor para abrir uma nova filial?

Problema de Associação

Oferta de novos serviços e produtos

Pergunta do Gestor:

- Quem observa esse produto, tem interesse em ver qual outro?

Leituras Sugeridas

- Escovedo, T. and Koshiyama, A., 2020. **Introdução a Data Science: Algoritmos de Machine Learning e métodos de análise.** Casa do Código. (Capítulo 1)

- Géron, A., 2019. **Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.** O'Reilly Media. (Capítulo 1)

Software Science Lab

Gestão Ágil de Projetos

Engenharia de Software para Ciência de Dados

Agenda

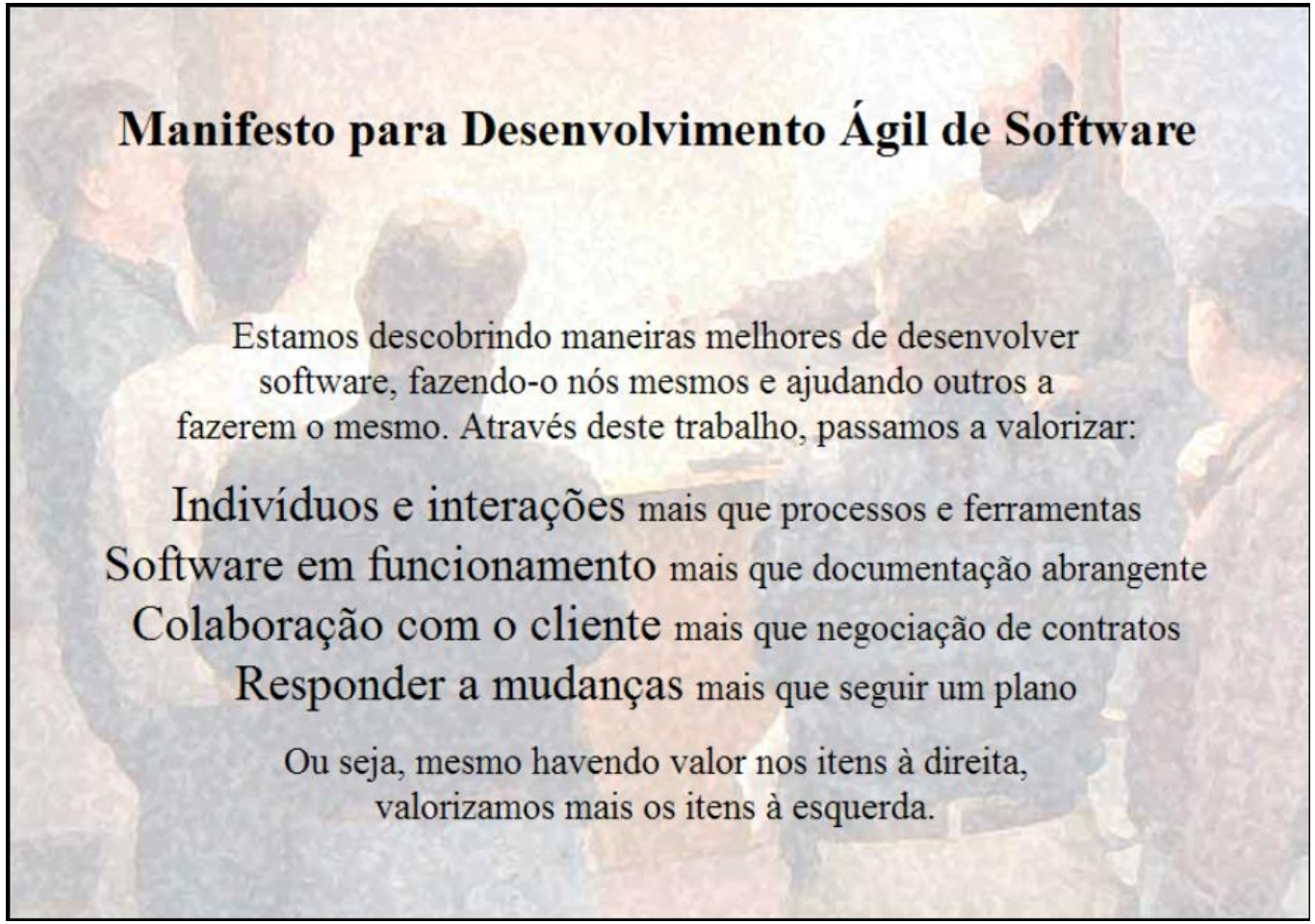
- Agilidade
  - Manifesto Ágil. Princípios Ágeis.
- Visão Geral do Scrum
  - Ciclo de Vida. Natureza Iterativa Incremental e Evolutiva.
- Framework Scrum
  - Papéis e Cerimônias

Software Science Lab

Agilidade

Manifesto Ágil. Princípios Ágeis.

Manifesto Ágil (2001)



12 Princípios Ágeis

Software Science Lab

1. Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
3. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
7. Software funcionando é a medida primária de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9. Contínua atenção à excelência técnica e bom design aumenta a agilidade.
10. Simplicidade - a arte de maximizar a quantidade de trabalho não realizado - é essencial.
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Leituras Sugeridas

- Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. **Manifesto for agile software development.** *Agile Alliance (2001).*

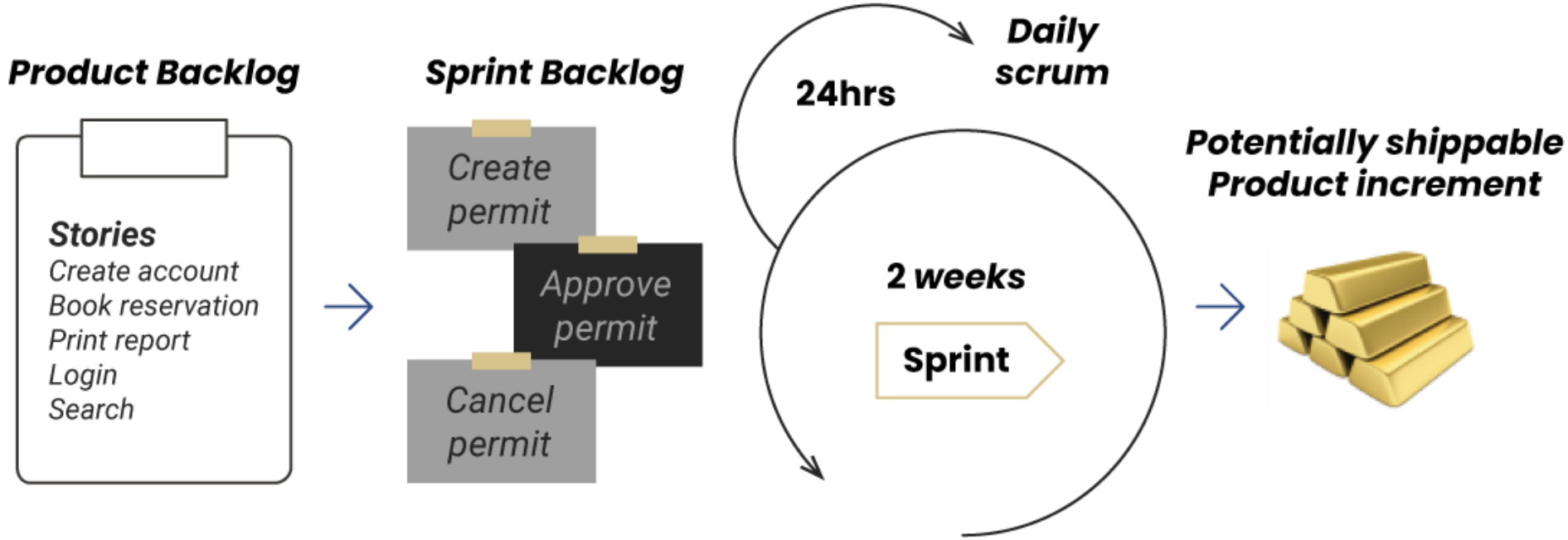
- Kuhrmann, M.; Tell, P.; Hebig, R.; Klunder, J. A.; Munch, J.; Linssen, O.; Pfahl, D.; Felderer, M.; Prause, C.; Macdonell, S.; Nakatumba-Nabende, J.; Raffo, D.; Beecham, S.; Tuzun, E.; Lopez, G.; Paez, N.; Fontdevila, D.; Licorish, S.; Kupper, S.; Ruhe, G.; Knauss, E.; Ozcan-Top, O.; Clarke, P.; Mc Caffery, F. H.; Genero, M.; Vizcaino, A.; Piattini, M.; Kalinowski, M.; Conte, T.; Prikladnicki, R.; Krusche, S.; Coskuncay, A.; Scott, E.; Calefato, F.; Pimonova, S.; Pfeiffer, R.; Pagh Schultz, U.; Heldal, R.; Fazal-Baqaie, M.; Anslow, C.; Nayebi, M.; Schneider, K.; Sauer, S.; Winkler, D.; Biffl, S.; Bastarrica, C.; and Richardson, I. **What Makes Agile Software Development Agile.** *IEEE Transactions on Software Engineering* (2021).

Software Science Lab

Visão Geral do Scrum

Ciclo de Vida. Natureza Iterativa Incremental e Evolutiva.

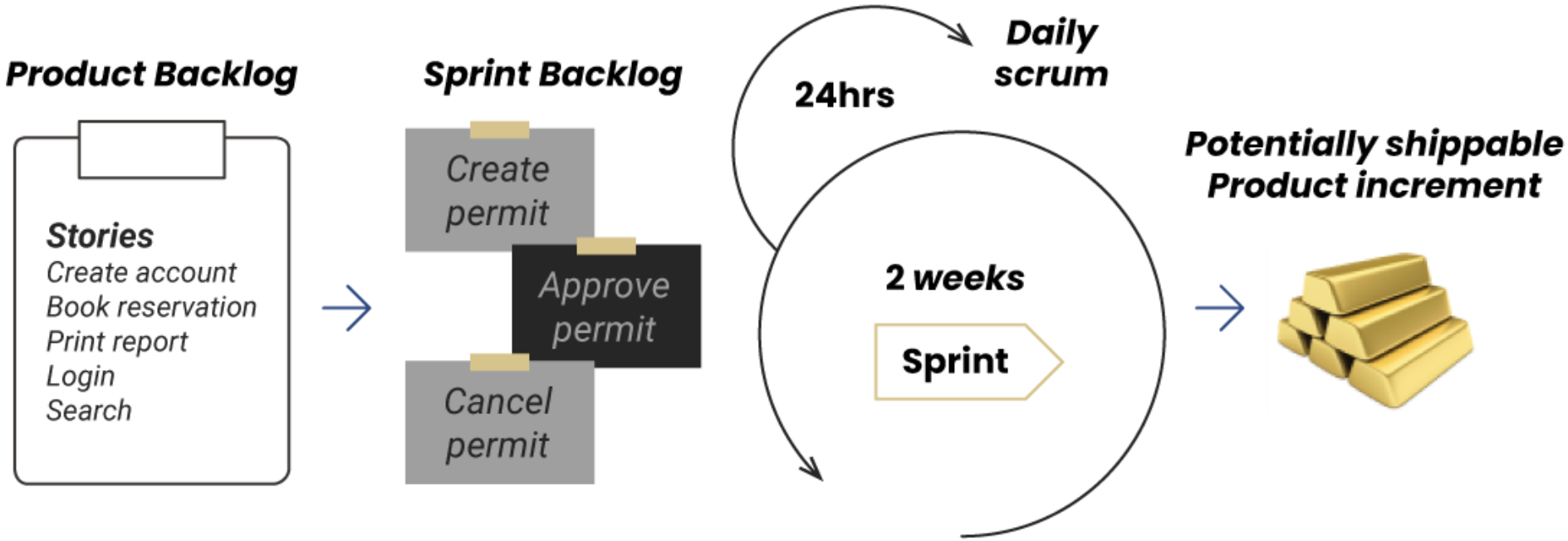
Scrum: Incremental e Evolutivo



Scrum

- **Scrum** é um framework de gestão **iterativo e incremental / evolutivo** bastante aplicado no contexto de desenvolvimento ágil de software
- O ciclo de vida do SCRUM utiliza tempo fixo – ***sprints*** (ao invés de escopo fixo) para determinar seus incrementos
- O Scrum é facilitado por um ***Scrum Master***, que tem como função primária remover qualquer impedimento à habilidade de uma equipe de entregar o objetivo do *sprint*
- Cada *sprint* corresponde a uma iteração que entrega um incremento de software

Scrum: Incremental e Evolutivo



Scrum

- Um ***backlog*** é conjunto de requisitos, priorizado pelo ***Product Owner*** (responsável por conhecer as necessidades do cliente)
- Entregas de um conjunto fixo de itens do *backlog* ocorrem em *sprints*
- Os itens do *backlog* para um *sprint* são definidos em uma sessão de planejamento
- Durante o *sprint*, ocorrem breves reuniões diárias, em que cada participante fala sobre o progresso conseguido, o trabalho a ser realizado e/ou o que o impede de seguir avançando
- Ao final de um *sprint*, ocorre a revisão e a retrospectiva, na qual todos os membros da equipe revisam a entrega e refletem sobre o *sprint* passado

Software Science Lab

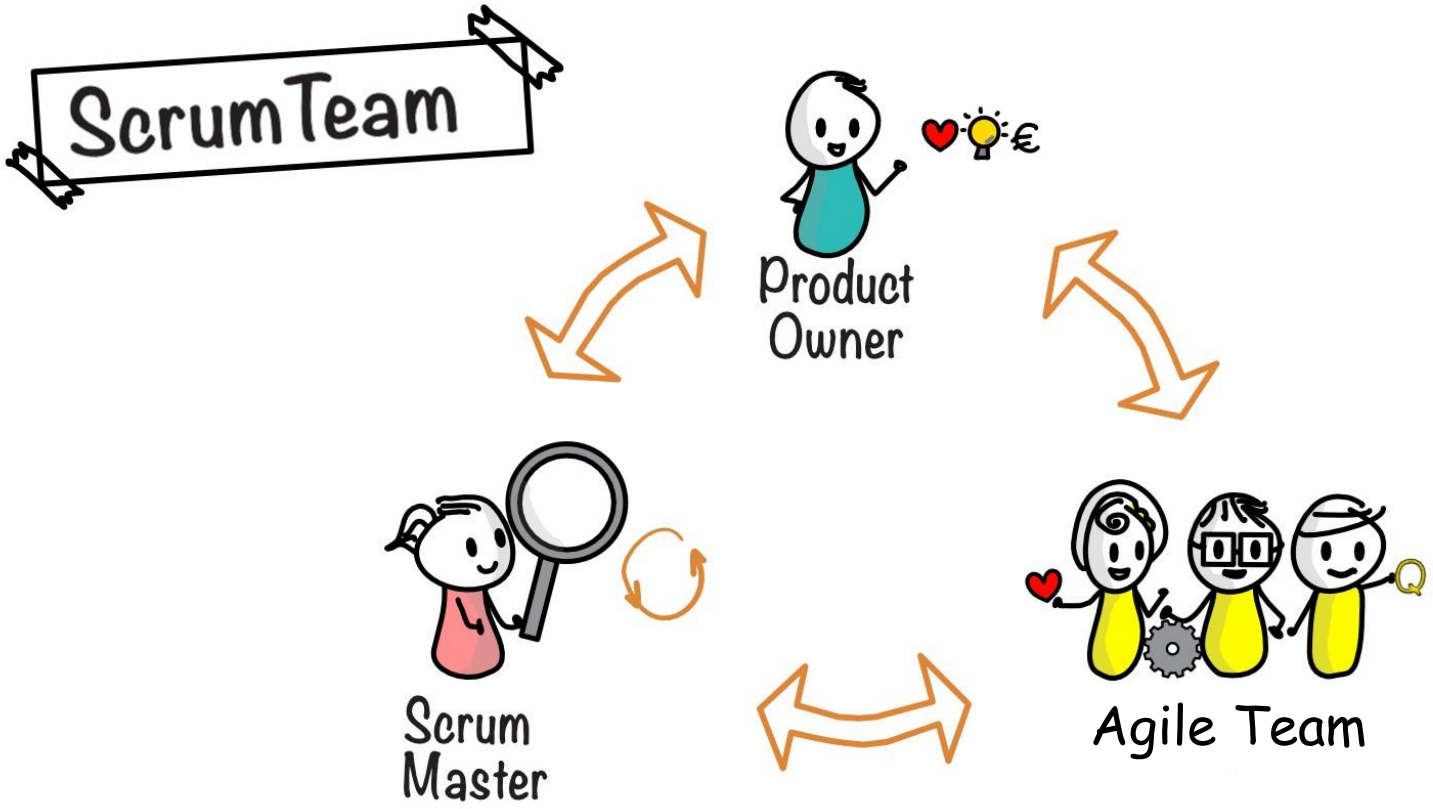
Framework Scrum

Papéis e Cerimônias

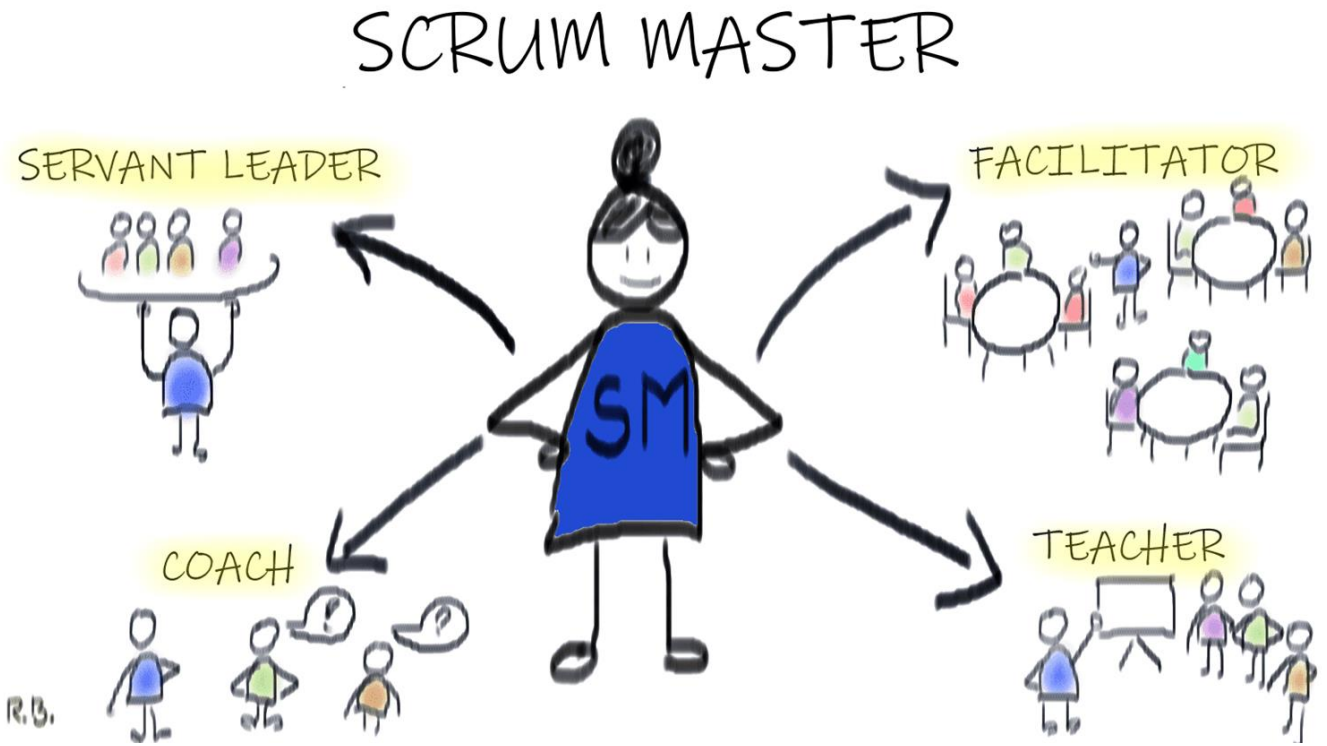
Os Pilares do Scrum

- Transparência
  - Aspectos significativos devem estar visíveis aos responsáveis pelos resultados
- Inspeção
  - Os usuários Scrum devem, frequentemente, inspecionar os artefatos Scrum e o progresso para detectar variações
- Adaptação
  - Se um ou mais aspectos desviou do esperado, o processo ou o material sendo produzido deve ser ajustado

Papeis do Scrum



Vocês conhecem esses papéis?



Mitos e Fatos sobre o Scrum Master

Software Science Lab

1. O Scrum Master trabalha para que o Scrum seja corretamente utilizado, ensinando-o e reforçando seus valores e regras
2. O Scrum Master remove impedimentos ao trabalho do time de entrega, mas apenas aqueles que não exigem mudanças organizacionais
3. Scrum Master é o papel mais importante em um projeto que usa Scrum
4. O Scrum Master atua como um líder servo do time de desenvolvimento
5. O Scrum Master não precisa participar das reuniões diárias, já que ela é um alinhamento interno do time de desenvolvimento
6. O Scrum Master em geral pode acumular o papel do P.O. sem problemas
7. O Scrum Master deve proteger o Time de Desenvolvimento de interferências externas que possam atrapalhar o seu trabalho
8. O Scrum Master deve gerenciar o trabalho do time de desenvolvimento
9. Um Scrum Master é eficaz quando se prova necessário para o bom desempenho do time de desenvolvimento
10. É benéfico para o Scrum Master envolvido em projetos de software entender de engenharia de software e boas práticas de desenvolvimento

Mitos e Fatos sobre o Scrum Master

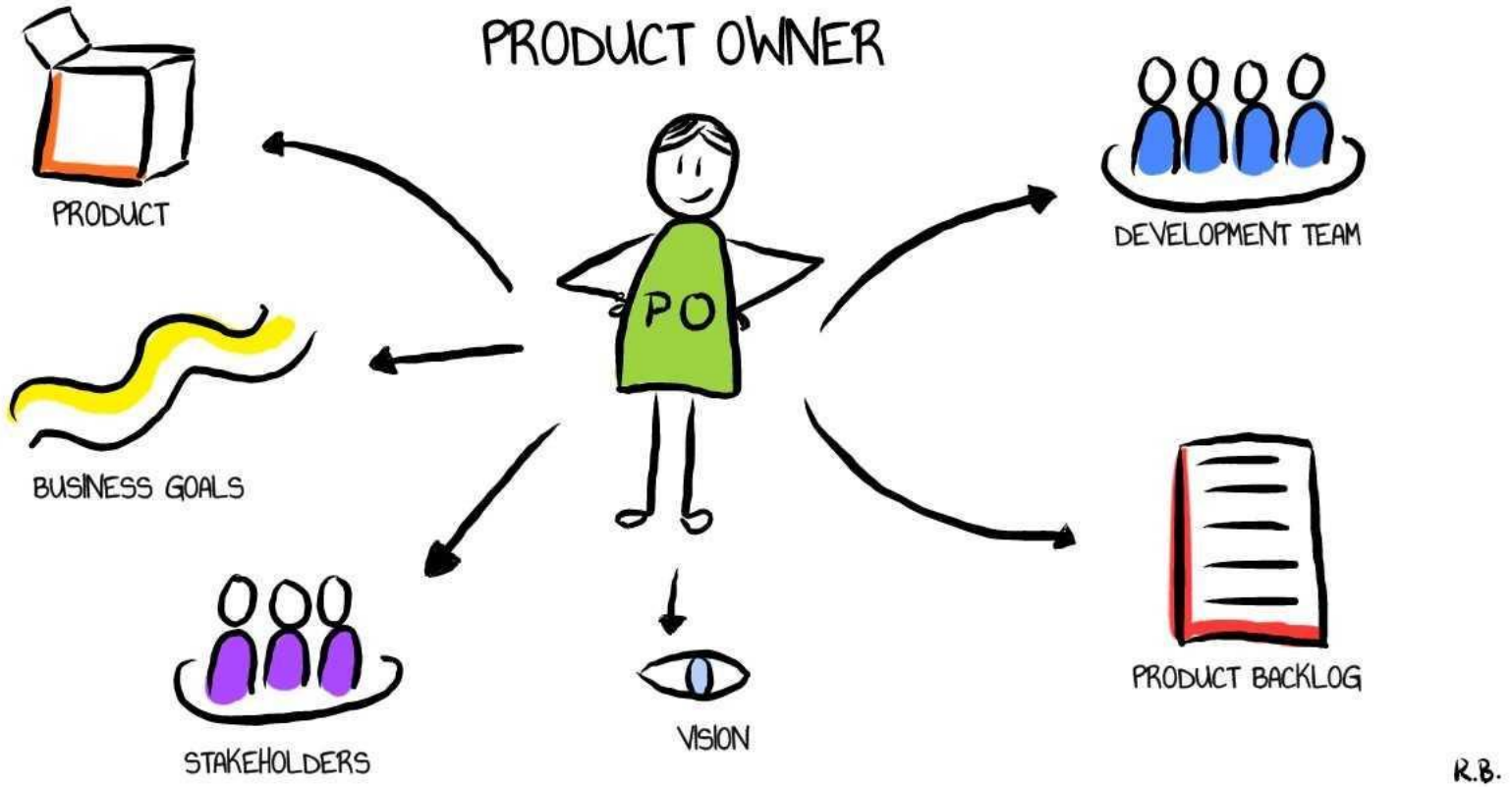
Software Science Lab

1. O Scrum Master trabalha para que o Scrum seja corretamente utilizado, ensinando-o e reforçando seus valores e regras
2. O Scrum Master remove impedimentos ao trabalho do time de entrega, mas apenas aqueles que não exigem mudanças organizacionais
3. Scrum Master é o papel mais importante em um projeto que usa Scrum
4. O Scrum Master atua como um líder servo do time de desenvolvimento
5. O Scrum Master não precisa participar das reuniões diárias, já que ela é um alinhamento interno do time de desenvolvimento
6. O Scrum Master em geral pode acumular o papel do P.O. sem problemas
7. O Scrum Master deve proteger o Time de Desenvolvimento de interferências externas que possam atrapalhar o seu trabalho
8. O Scrum Master deve gerenciar o trabalho do time de desenvolvimento
9. Um Scrum Master é eficaz quando se prova necessário para o bom desempenho do time de desenvolvimento
10. É benéfico para o Scrum Master envolvido em projetos de software entender de engenharia de software e boas práticas de desenvolvimento

Scrum Master

- Aplica o Scrum e facilita os eventos
- Remove impedimentos do time
- Ensina e lidera o time rumo aos objetivos do PO

Vocês conhecem esses papéis?



Mitos e Fatos sobre o Product Owner

Software Science Lab

1. O P. O. é o “proxy” ou representante do cliente, transmitindo seus desejos ao Time de Desenvolvimento
2. Embora deva ser influenciado por diferentes pessoas, o P. O. é o único que pode modificar o Product Backlog
3. O P. O. deve balancear necessidades e desejos dos diferentes stakeholders do projeto
4. O melhor P. O. é o próprio cliente ou alguém escolhido por ele
5. O P. O. deve ser apenas uma pessoa – deve haver apenas um ponto de decisão sobre o produto para o Time de Desenvolvimento
6. O P. O. em geral pode acumular o papel do ScrumMaster sem problemas
7. O P. O. deve colaborar de perto com o Time de Desenvolvimento para maximizar o valor entregue ao cliente
8. O P. O. deve cobrar do Time de Desenvolvimento o compromisso de que todos os itens escolhidos para o Sprint sejam desenvolvidos
9. A presença do P. O. não é obrigatória na reunião de Sprint Planning – basta o Time de Desenvolvimento escolher os itens do alto do Product Backlog
10. O P. O. é responsável por definir o produto certo a ser desenvolvido

Mitos e Fatos sobre o Product Owner

Software Science Lab

1. O P. O. é o “proxy” ou representante do cliente, transmitindo seus desejos ao Time de Desenvolvimento
2. Embora deva ser influenciado por diferentes pessoas, o P. O. é o único que pode modificar o Product Backlog
3. O P. O. deve balancear necessidades e desejos dos diferentes stakeholders do projeto
4. O melhor P. O. é o próprio cliente ou alguém escolhido por ele
5. O P. O. deve ser apenas uma pessoa – deve haver apenas um ponto de decisão sobre o produto para o Time de Desenvolvimento
6. O P. O. em geral pode acumular o papel do ScrumMaster sem problemas
7. O P. O. deve colaborar de perto com o Time de Desenvolvimento para maximizar o valor entregue ao cliente
8. O P. O. deve cobrar do Time de Desenvolvimento o compromisso de que todos os itens escolhidos para o Sprint sejam desenvolvidos
9. A presença do P. O. não é obrigatória na reunião de Sprint Planning – basta o Time de Desenvolvimento escolher os itens do alto do Product Backlog
10. O P. O. é responsável por definir o produto certo a ser desenvolvido

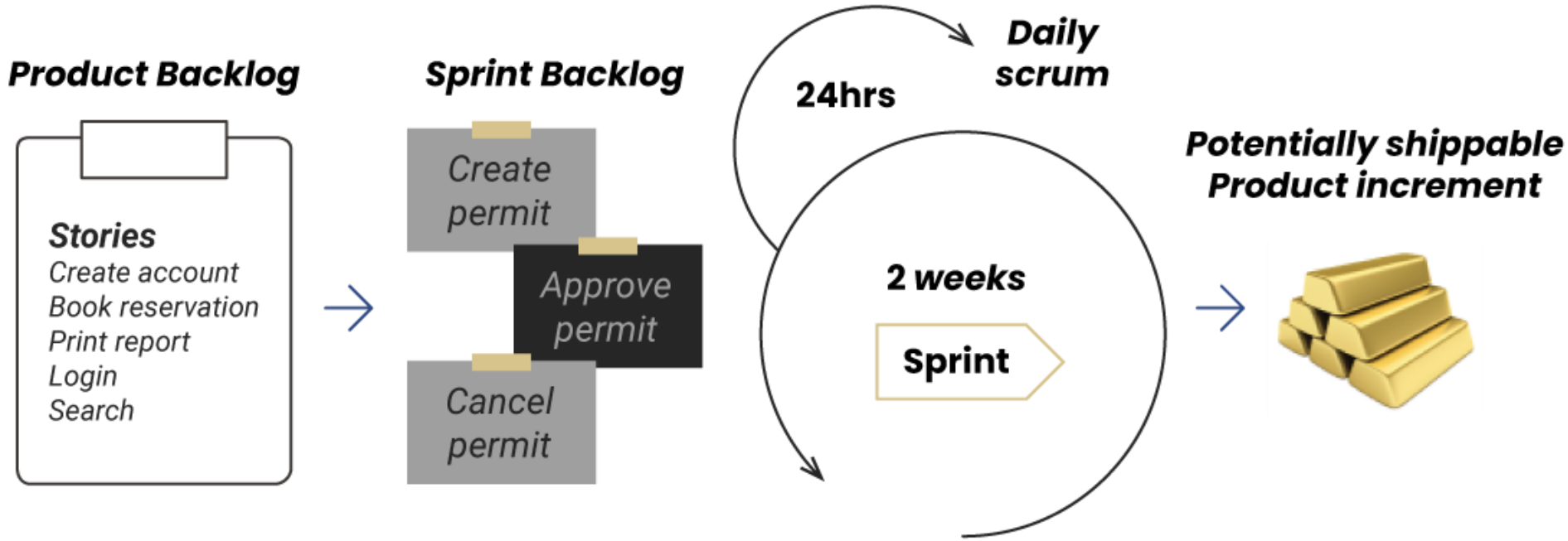
Product Owner

- Foca no Retorno de Investimento (ROI) e Valor Agregado para o Cliente
- Único que gerencia o Product Backlog
- O time deve respeitar as suas decisões
  - As decisões devem considerar os pontos de vista dos diferentes interessados

Time Ágil de Entrega

- Busca ser auto-gerenciável
- Multifuncionais (dentro do possível e de sua expertise)
- Responsabilidade compartilhada

Cerimônias do Scrum



Sprint

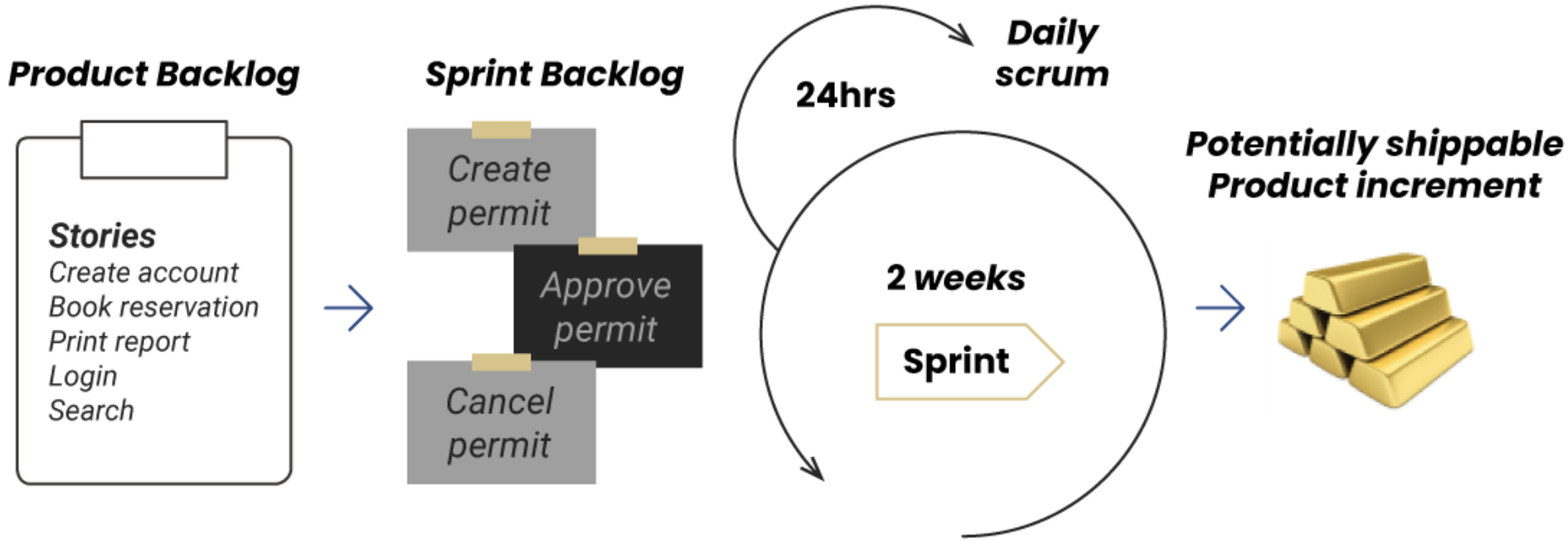
- Período (normalmente de duas semanas) durante o qual um incremento utilizável do produto é criado
- Inclui planejamento, reuniões diárias, o desenvolvimento, a revisão e a retrospectiva

Planejamento do Sprint

- O trabalho a ser realizado é planejado
  - O que foi priorizado para ser feito primeiro?
  - O que está preparado para entrar no sprint?
  - O que pode ser incluído no sprint considerando as estimativas e a capacidade da equipe?

- O resultado é o Sprint Backlog

Cerimônias do Scrum



Reunião Diária

- Reunião diária de 15 minutos, para que o Time de Desenvolvimento possa sincronizar as atividades e criar um plano para as próximas 24 horas

- Apenas três perguntas:
  - O que eu fiz ontem?
  - O que eu vou fazer hoje?
  - Tenho algum impedimento?

Sprint Review

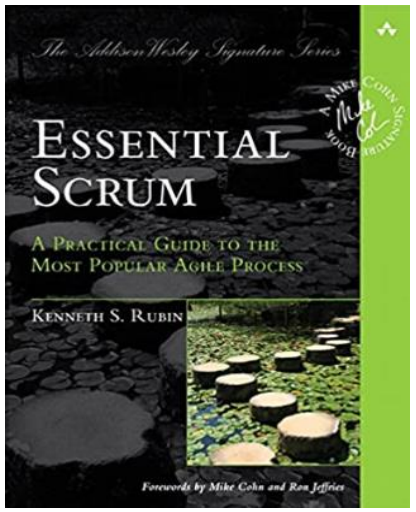
- Executada no final da Sprint para inspecionar e fazer o aceite dos itens desenvolvidos
- Pode resultar em adaptações para o Backlog do Produto

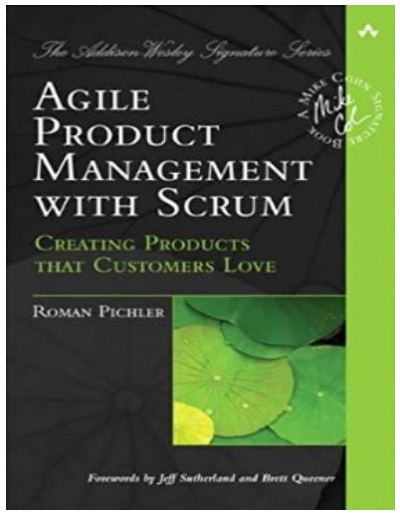
Retrospectiva do Sprint

- Oportunidade para equipe refletir sobre melhorias a serem aplicadas na próxima Sprint
  - O que foi bom?
  - O que pode melhorar?
  - Ações concretas para melhoria?

Livros para quem quiser saber mais sobre Scrum

Software Science Lab



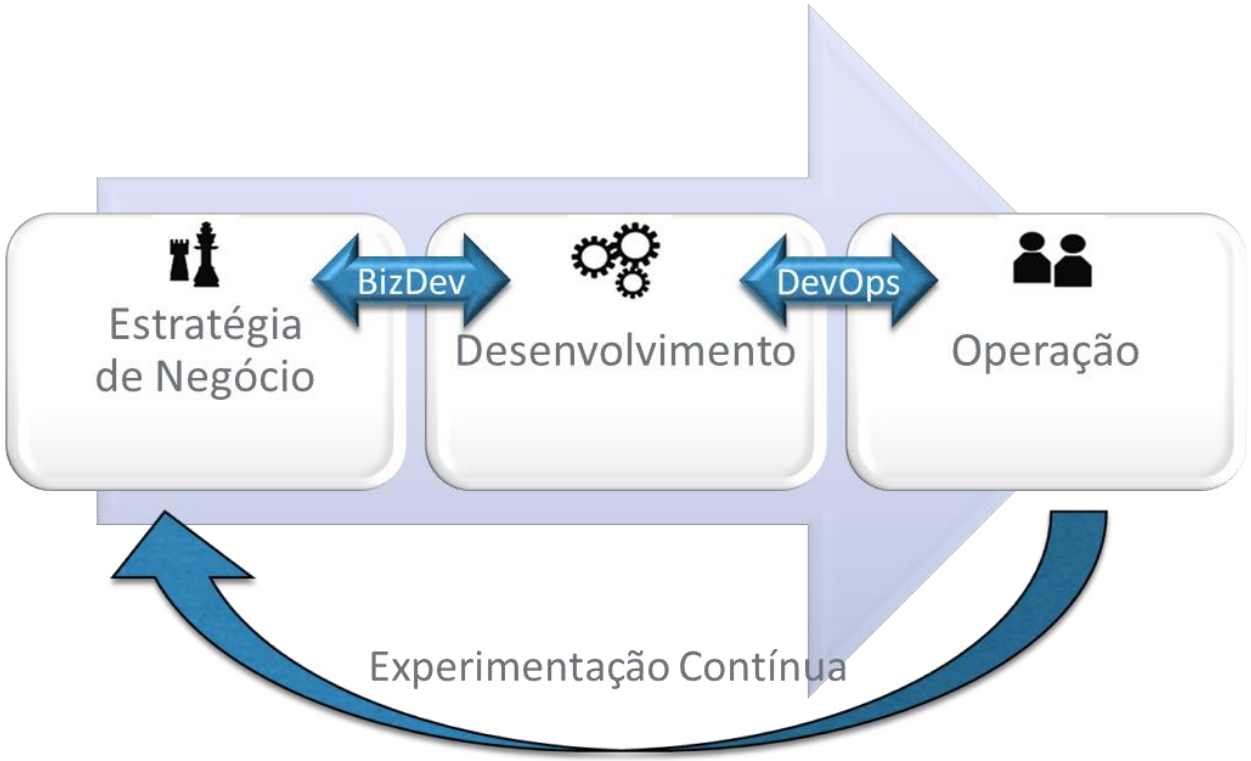


Software Science Lab

Engenharia de Sistemas de Software Inteligentes

Engenharia de Software para Ciência de Dados

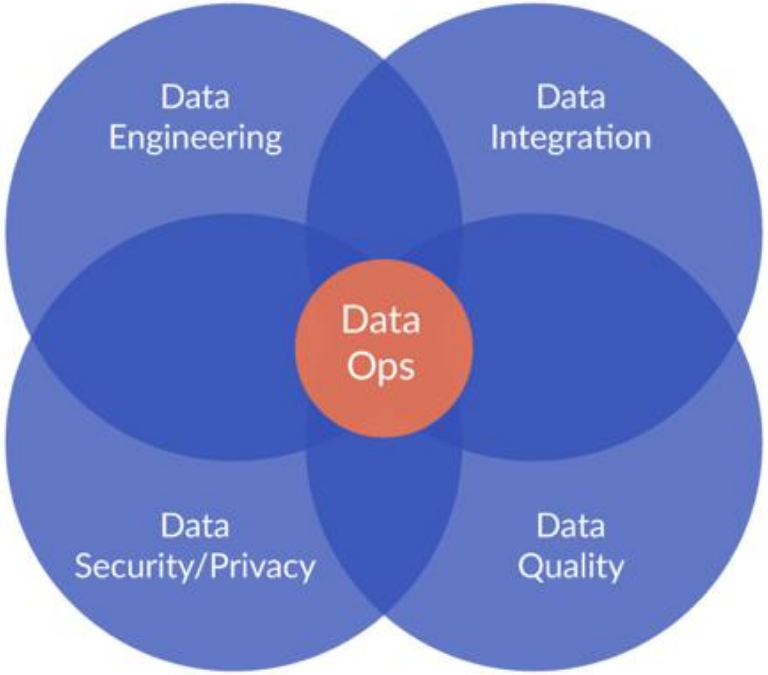
BizDev, DevOps e Experimentação Contínua



- **BizDev** é o alinhamento da estratégia de negócio com o desenvolvimento
- **DevOps** é o alinhamento do desenvolvimento de software com a implantação do software em operação

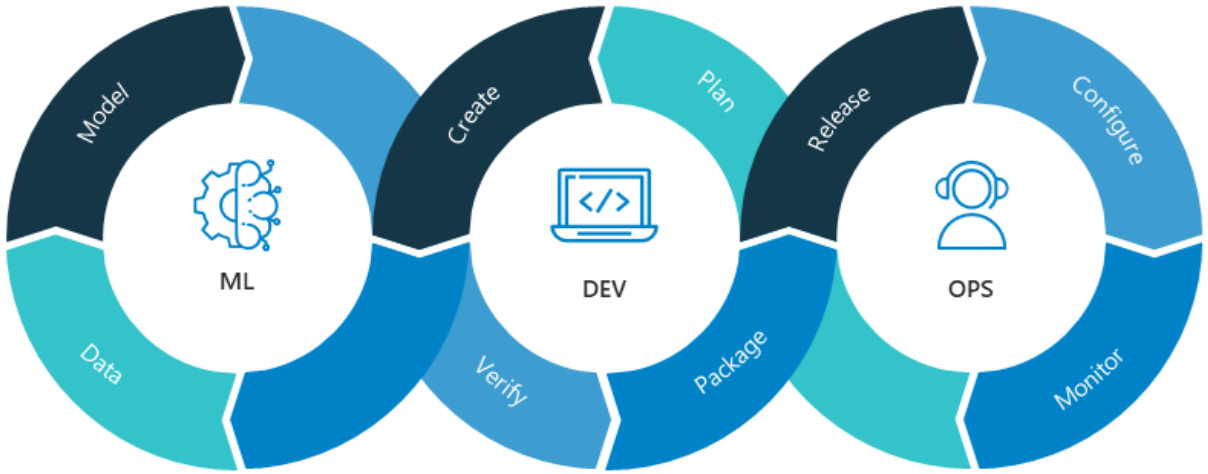
B. Fitzgerald, and K.J. Stol, "Continuous software engineering: A roadmap and agenda," Journal of Systems and Software, vol. 123, pp.176-189, 2017.

DataOps



- **DataOps** visa em unir as áreas de engenharia de dados, segurança de dados, qualidade de dados e integração de dados, para o sucesso de seus projetos

MLOps



Fonte: https://nealanalytics.com/expertise/mlops/

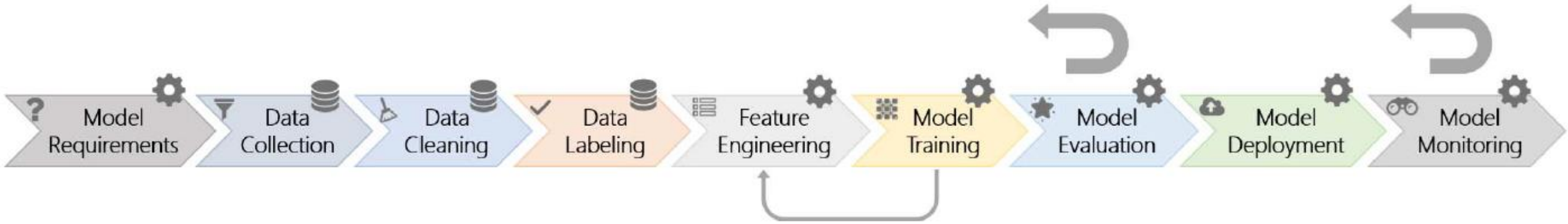
- **MLOps** é o alinhamento do desenvolvimento dos modelos com o desenvolvimento de software e a operação
- Contempla as melhores práticas para colaboração entre cientistas de dados, desenvolvedores e profissionais de operação

MLOps

- A necessidade …
  - Muitas vezes não se pode simplesmente implantar um modelo de ML treinado off-line como um serviço de previsão, **será necessário um pipeline de várias etapas para retreinar e implantar automaticamente um modelo**
  - Esse pipeline adiciona complexidade porque você precisa **automatizar etapas que os cientistas de dados executam manualmente** antes da implantação para treinar e validar novos modelos

- Fases do MLOps:
  - *Data gathering*
  - *Data analysis*
  - *Data transformation/preparation*
  - *Model training & development*
  - *Model validation*
  - *Model serving*
  - *Model monitoring*
  - *Model re-training*

ML Workflow



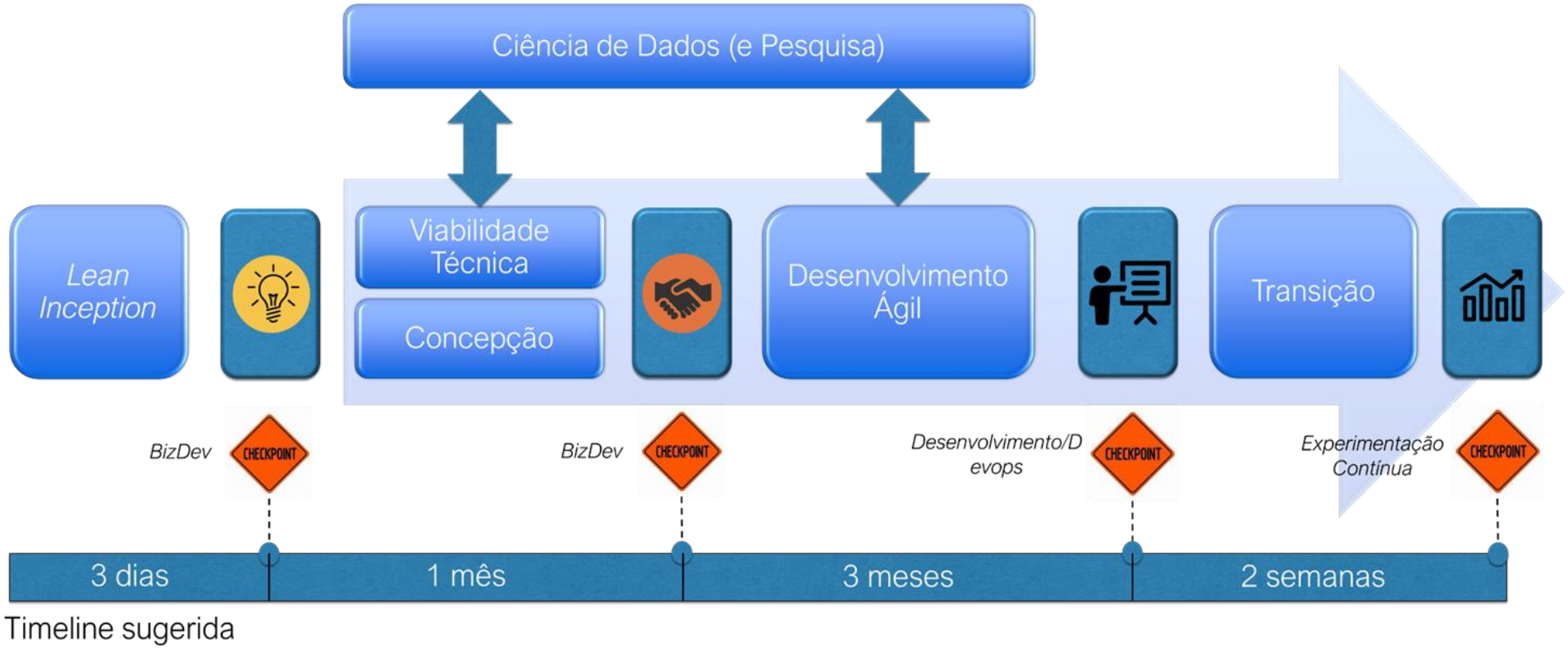
Machine Learning Workflow (Amershi et al., 2019)

Diferenças da engenharia de software tradicional (exemplos):

- Descobrir, gerenciar e versionar dados requeridos para aplicações de ML é muito mais complexo e difícil
- Customização e reutilização de modelos requer habilidades muito diferentes das normalmente encontradas em equipes de software

Amershi, S., Begel, A., Bird, C., *et al.*, **Software engineering for machine learning: A case study.** In *IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2019*

Exemplo de Abordagem: Lean R&D



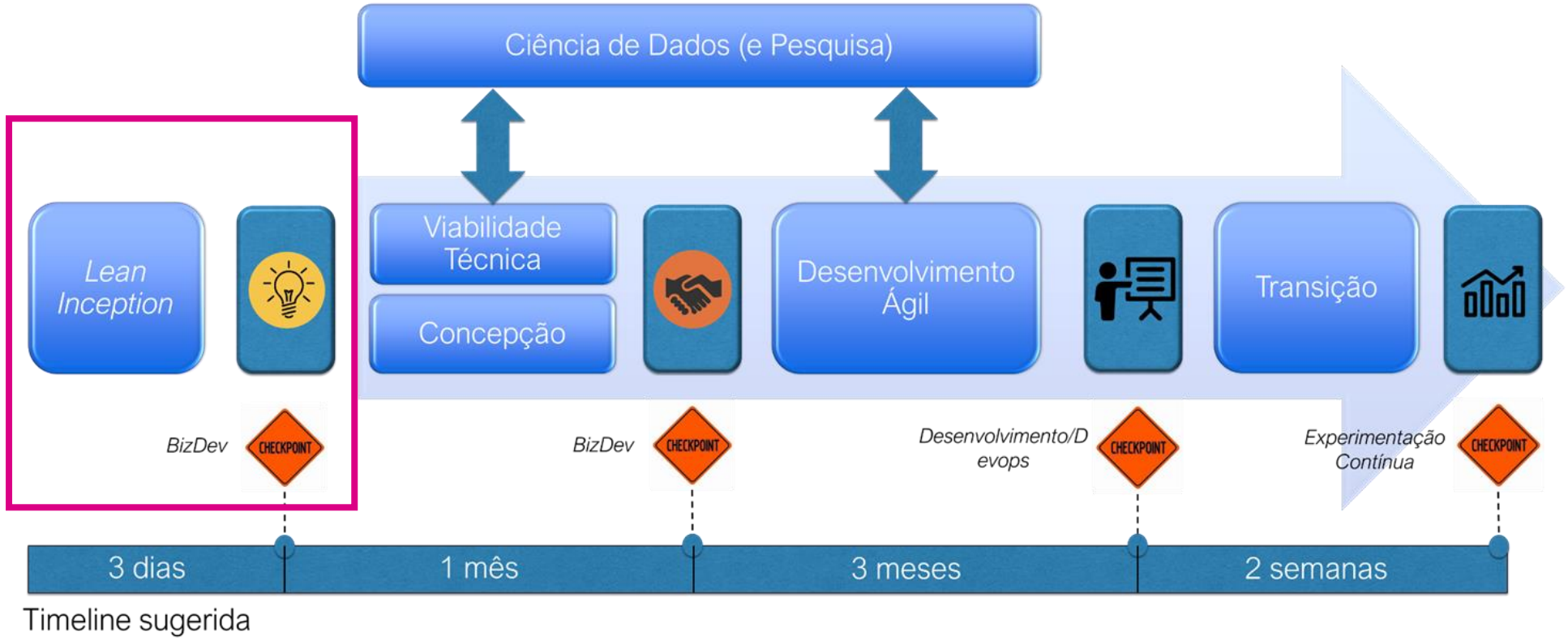
Timeline sugerida

KALINOWSKI, M. *et al.* **Lean R&D: An agile research and development approach for digital transformation**. In: International Conference on Product-Focused Software Process Improvement. Springer, Cham, 2020. p. 106-124.

Lean R&D Papéis

- **Comitê Gestor:** Avalia os projetos nos pontos de verificação descritos. Esta avaliação visa: (i) permitir a 'falha rápida' de ideias que não entregariam o valor de negócio esperado; e (ii) assegurar que a abordagem está sendo usada para enfrentar desafios relevantes de inovação e transformação digital
- **Project Manager (Scrum Master):** Facilita as *Lean Inceptions* e gerencia as equipes de pesquisa e desenvolvimento ágeis, garantindo que a abordagem geral de P&D Ágil seja seguida de forma adequada
- **Product Owners (POs) e representantes do cliente.** Assim como no Scrum tradicional, os POs são responsáveis por maximizar o valor comercial do produto resultante do trabalho da equipe de desenvolvimento
- **Desenvolvedores.** A equipe de desenvolvimento
- **Cientistas de Dados.** Apoiam a equipe de desenvolvimento em uma avaliação inicial de viabilidade técnica e em tarefas relacionadas com ciência de dados durante o desenvolvimento (e.g., investigando técnicas de aprendizado de máquina a serem usadas, elaborando modelos de previsão)
- **Equipe de design de UX/UI.** Responsável por projetar *mock-ups* de interação com o usuário e protótipos de alta fidelidade para subsidiar o desenvolvimento do *front-end*
- **Analista de DevOps e infraestrutura.** Responsável por fornecer a infraestrutura DevOps para as equipes de desenvolvimento

Exemplo de Abordagem: Lean R&D



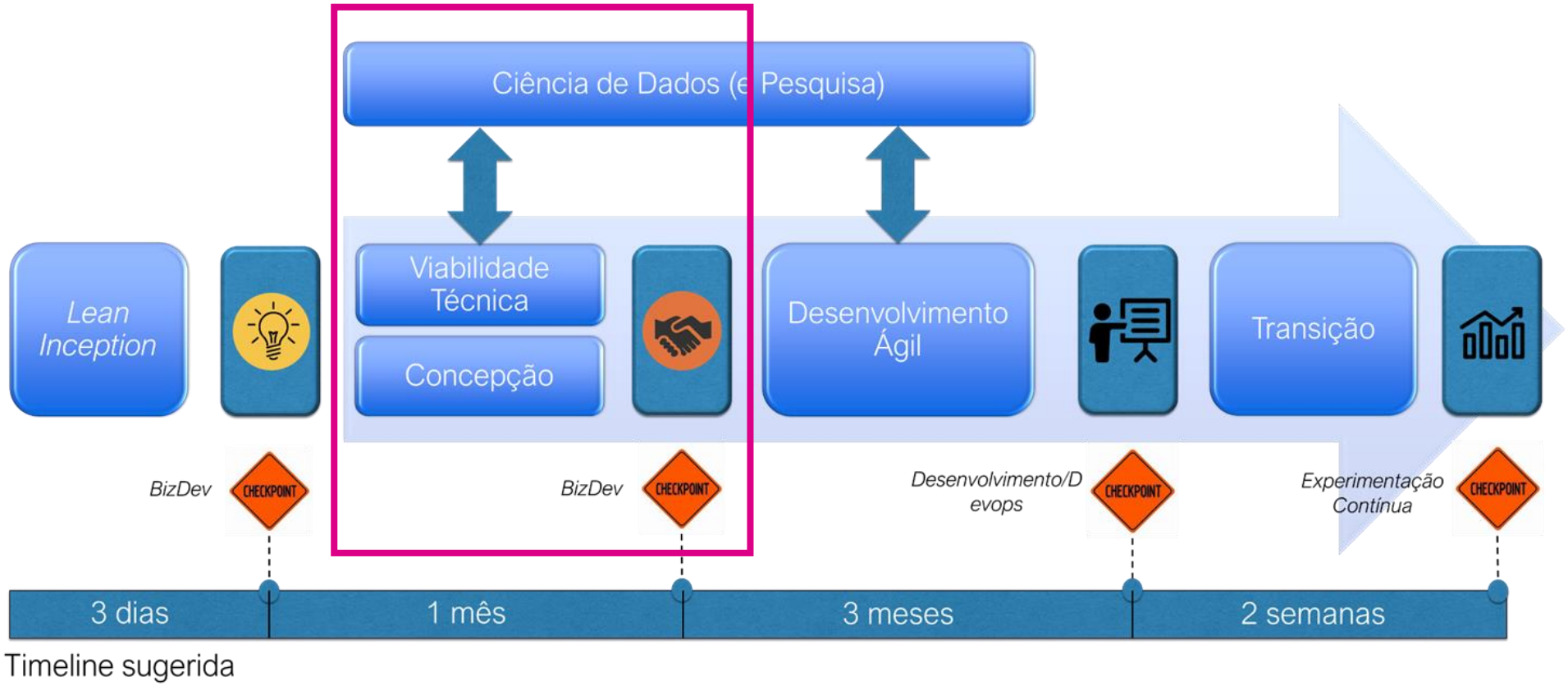
Timeline sugerida

Lean R&D Atividades

- A abordagem começa com uma ***Lean Inception*** para permitir que as partes interessadas definam em conjunto a visão de um MVP que pode ser usado para testar hipóteses de negócios

- No primeiro ponto de verificação, **o comitê gestor deve aprovar a visão definida para o MVP**
  - Se for **rejeitada**, uma nova *Lean Inception* deve ser conduzida, potencialmente focando em um problema diferente
  - Se for **aprovada**, duas fases começam em paralelo: Viabilidade Técnica e Concepção

Exemplo de Abordagem: Lean R&D



Timeline sugerida

Lean R&D Atividades

- Durante a **Viabilidade Técnica**, a equipe de desenvolvimento, auxiliada pela equipe de ciência de dados e o analista de infraestrutura, começa a investigar a viabilidade técnica de implementação das features identificadas durante a *Lean Inception*
- Seguindo a estratégia *tracer bullet*, esta fase normalmente serve como uma prova de que a arquitetura é compatível e viável e que há uma maneira de resolver o problema com eficácia razoável, além de fornecer um esqueleto funcional demonstrável (MVP0) contendo algumas implementações iniciais

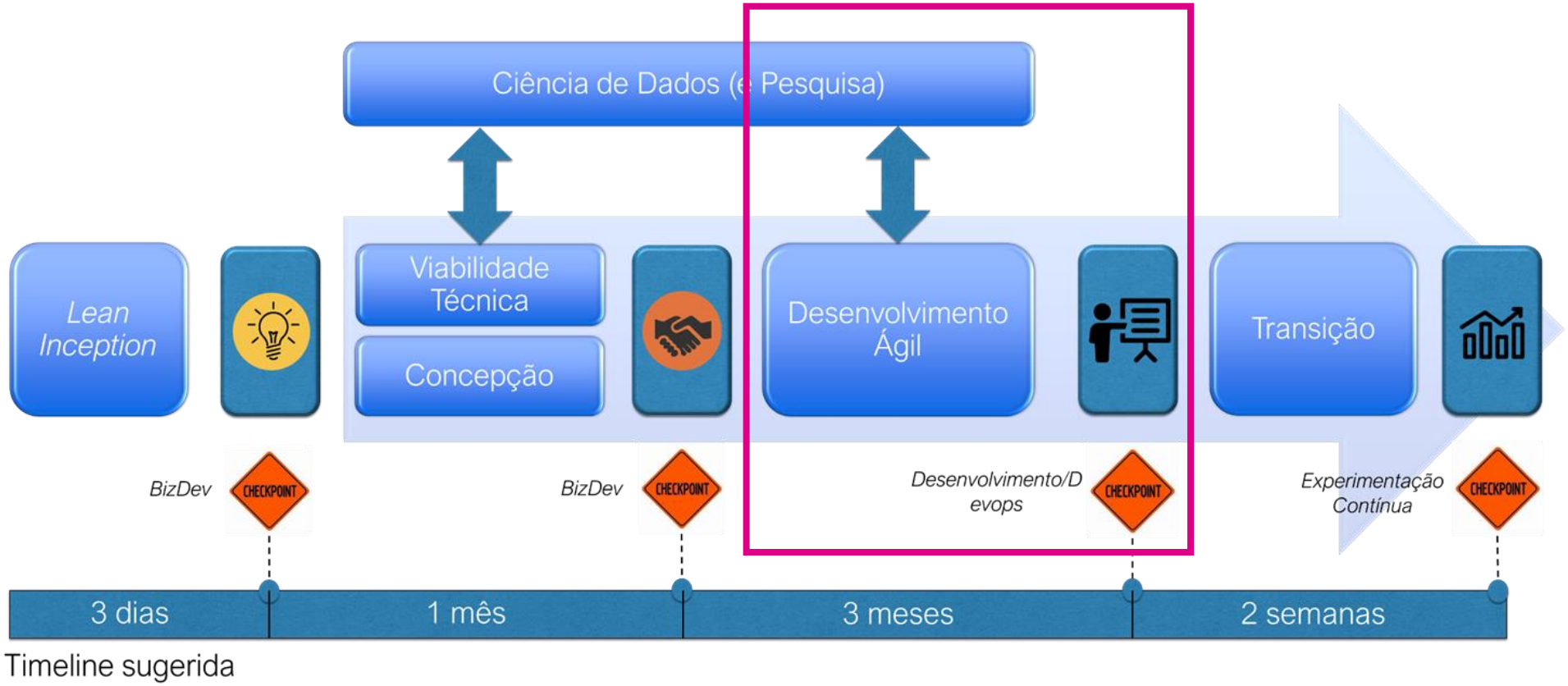
Lean R&D Atividades

- A **Concepção** envolve o PO detalhando as features do MVP identificadas durante a Lean Inception aplicando dinâmicas de *Product Backlog Building* com os representantes do cliente, seguida por outras técnicas típicas de elicitação de requisitos (por exemplo, entrevistas), a fim de especificar histórias de usuário. A equipe de UX/UI participa da criação de protótipos de baixa fidelidade para validação de requisitos e protótipos de UI de alta fidelidade para testes de usabilidade
- As histórias de usuário devem ser complementadas com cenários de *Behavior-Driven Development* (BDD) que devem ser usados como critério de aceitação

Lean R&D Atividades

- Ao final da concepção, ocorrem a revisão dos requisitos ágeis (histórias de usuários, cenários BDD e mock-ups) e os testes de usabilidade nos protótipos de alta fidelidade, ambos junto ao cliente
- No segundo ponto de verificação, **o comitê gestor deve decidir se o MVP deve ser realmente desenvolvido**, analisando a especificação de requisitos concebida, juntamente com os resultados da avaliação de viabilidade técnica, da revisão de requisitos e dos testes de usabilidade

Exemplo de Abordagem: Lean R&D

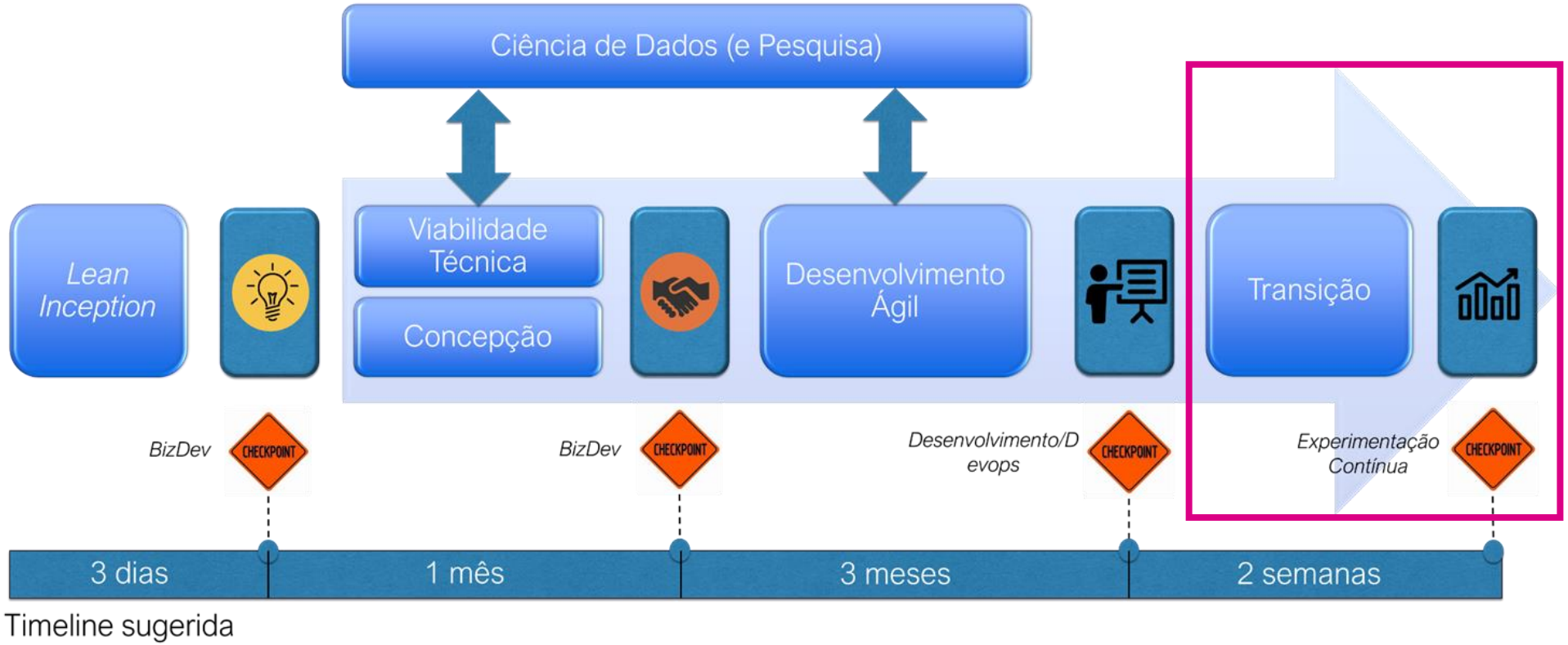


Timeline sugerida

Lean R&D Atividades

- A partir daí, a fase de **Desenvolvimento Ágil** envolve os desenvolvedores, com o apoio da equipe de ciência de dados, implementando o MVP. Esta fase segue o desenvolvimento baseado em Scrum padrão com planejamento de sprint, reuniões diárias, revisões e retrospectivas de *sprint*
- Para fins de controle da qualidade, recomendamos o uso de revisões de código modernas, que permitem identificar defeitos, melhorar soluções e compartilhar conhecimento e propriedade do código. Recomenda-se também a criação de um painel de gestão ágil que permite monitorar o progresso geral da equipe
- No terceiro ponto de verificação, **o comitê gestor deve decidir sobre a transição do MVP para produção**, após apresentação do PO

Exemplo de Abordagem: Lean R&D



Timeline sugerida

Lean R&D Atividades

- A fase de **Transição** envolve a equipe de desenvolvimento e infraestrutura preparando o MVP para o teste beta em seu ambiente final e avaliando as hipóteses de negócios
- A equipe de pesquisa deve projetar o plano de experimento, que deve delinear como instrumentar o produto para permitir a coleta das medidas necessárias para testar as hipóteses de negócios e, eventualmente, construir outros instrumentos de avaliação (e.g., questionários para medir a satisfação do usuário)
- No último ponto de verificação **o comitê gestor analisa os resultados da experimentação contínua**, i.e., se as hipóteses de negócios foram alcançadas e decide sobre investir em outro ciclo para melhorar o MVP

Boas Práticas de Engenharia de Software

- Gestão Ágil, DevOps, MLOps
- Integração de cientistas de dados, analistas de UX/UI e analistas de DevOps/Infra no processo de desenvolvimento
- Avaliação inicial da viabilidade técnica e arquitetural
- Especificação de requisitos ágeis com cenários de aceitação
- Especificação da UX/UI
- Revisão de requisitos e testes de usabilidade
- Boas práticas de projeto e construção
- Análise estática e testes automatizados
- Revisões de código modernas
- Integração contínua e gerência de configuração
- ...

Leituras Sugeridas

- Alonso, S., Kalinowski, M., Viana, M., Ferreira, B. and Barbosa, S. D. J. **A Systematic Mapping Study on the Use of Software Engineering Practices to Develop MVPs**, *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2021
- Amershi, S., Begel, A., Bird, C., *et al.*, **Software engineering for machine learning: A case study.** In *IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2019*
- Correia, J.L., Pereira, J.A., Mello, R., Garcia, A., Fonseca, B., Ribeiro, M., Gheyi, R., Kalinowski, M., Cerqueira, R. and Tiengo, W., **Brazilian Data Scientists: Revealing their Challenges and Practices on Machine Learning Model Development.** In *19th Brazilian Symposium on Software Quality*, 2020.
- Kalinowski, M. *et al.* **Lean R&D: An agile research and development approach for digital transformation**. In: International Conference on Product-Focused Software Process Improvement. Springer, Cham, 2020. p. 106-124.

Software Science Lab

Requisitos de Sistemas de Software Inteligentes

Engenharia de Software para Ciência de Dados

Software Science Lab

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems."

Frederick P. Brooks Jr.

Requisitos de Sistemas de Software Inteligentes

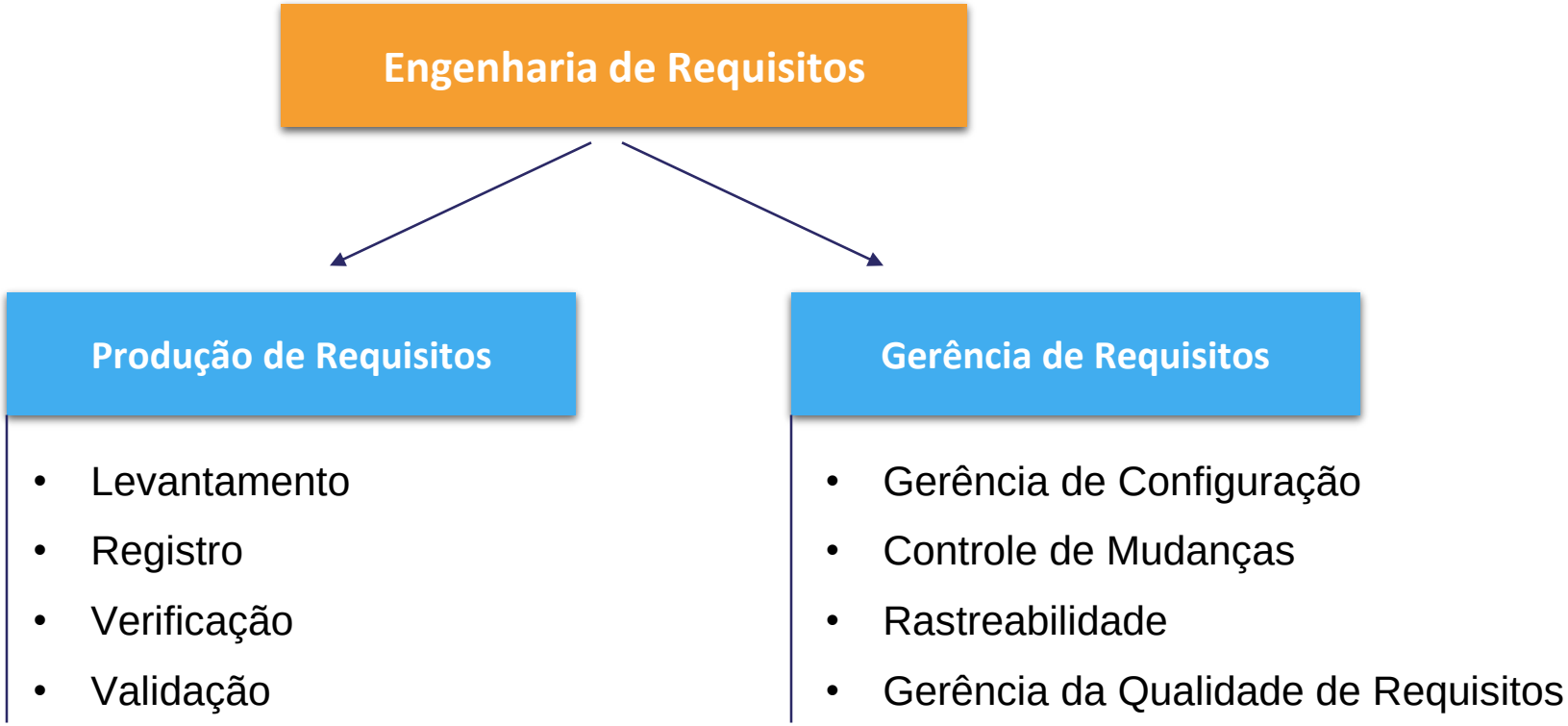
- Agenda:
  - **Conceitos Básicos de Requisitos de Software**
    - Engenharia de Requisitos
    - Tipos de Requisitos
  - Requisitos em Contextos Ágeis
    - Ideação (*Lean Inception, MVP Canvas*)
    - *Product Backlog Building*
    - User Stories e Cenários BDD
    - Gestão de Requisitos Ágil
  - Requisitos de Sistemas de Software Inteligentes
    - Visão Geral
    - ML Canvas
    - PerSpecML

Engenharia de Requisitos

- Definição
  - Para Glinz *et al.* (2020), Engenharia de Requisitos é:
    - "A abordagem sistemática e disciplinada para a **especificação e gerenciamento** de requisitos com o objetivo de compreender os desejos e necessidades das partes interessadas e minimizar o risco de entregar um sistema que não atenda a esses desejos e necessidades"

- Um requisito denota 3 conceitos (Glintz *et al.*, 2020):
  1. Uma necessidade percebida por um interessado
  2. Uma capacidade ou propriedade que um sistema deve possuir
  3. Uma representação documentada de uma necessidade, capacidade ou propriedade

Engenharia de Requisitos – Visão Geral



Tipos de Requisitos

- Requisitos Funcionais
- Requisitos Não Funcionais
- Requisitos do Domínio (Regras de Negócio)

Sommerville, I., **Engenharia de Software**. 10a Edição, Pearson, 2019.

- Requisitos Funcionais
- Requisitos de Qualidade
- Restrições

Glinz, M., van Loenhoud, H., Staal, S. and Bühne, S., **Handbook for the CPRE Foundation Level according to the IREB Standard**. 2020.

Requisitos Funcionais

- Requisitos Funcionais
  - Requisitos diretamente ligados a funcionalidade do software, descrevem as **funções que o software deve executar**

- Regra de formação (boa prática)

  [ID] O software deve permitir que o [responsável pela ação] + [ação]

- Exemplo:

  [RF1] O software deve permitir que o gerente financeiro realize a predição do faturamento mensal da loja.

Requisitos Não Funcionais

Software Science Lab

- Em vez de informar o que o sistema fará, os requisitos não-funcionais descrevem **características de qualidade** ou **restrições**
- Requisitos não-funcionais podem ser mais críticos que requisitos funcionais. Em muitas situações, o sistema que **não os satisfaz se torna um sistema inútil**

- Exemplo:

  [RNF1] A predição do modelo deve ser realizada dentro de três segundos

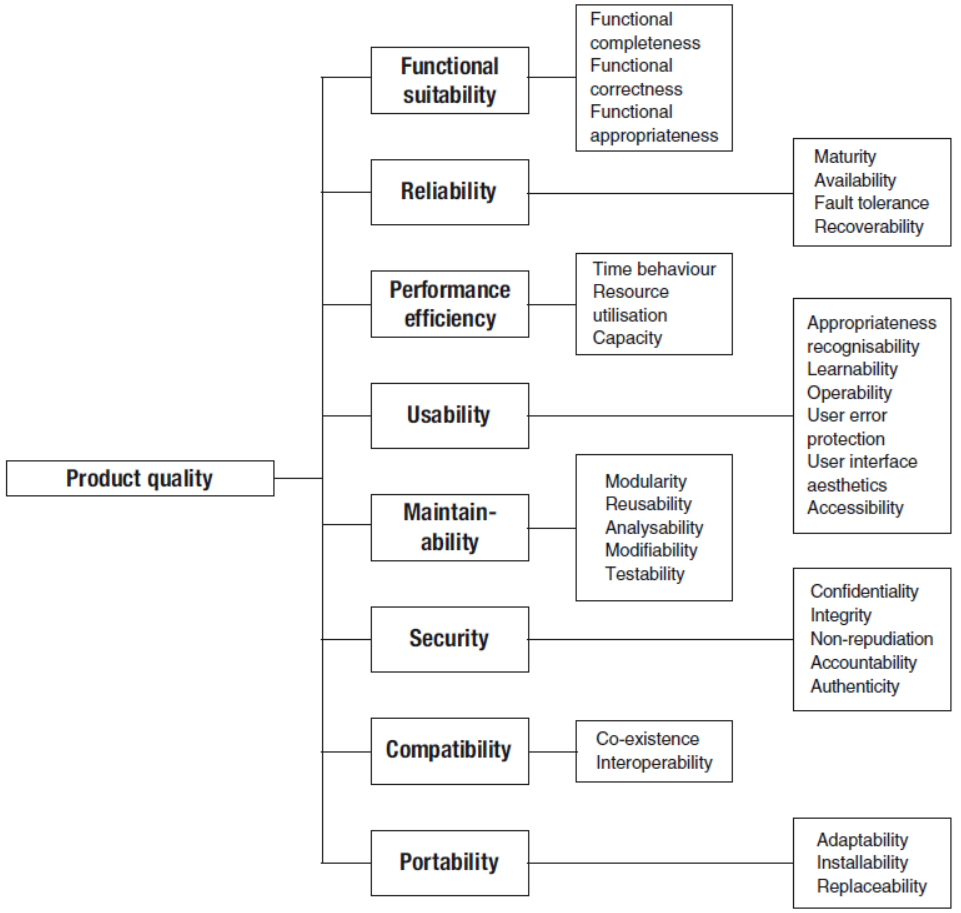
Requisitos Não Funcionais

Software Science Lab

- Requisitos não-funcionais **devem ser mensuráveis e verificáveis**
- **Incluem** requisitos do produto, como**:**
  - requisitos de **usabilidade**
    - e.g., usuários experientes devem ser capazes de utilizar todas as funções do sistema após duas horas de treinamento
  - requisitos de **confiabilidade**
    - e.g., o sistema deve estar disponível 99% das vezes;
  - requisitos de **segurança**
    - e.g., o acesso ao dados deve ser protegido
  - requisitos de **desempenho**
    - e.g., o sistema deve processar X requisições por segundo
  - requisitos de **capacidade**
    - e.g., o sistema deve suportar X usuários concorrentemente
  - requisitos de **portabilidade**
    - e.g., o sistema deve ser capaz de executar nas plataformas X e Y

Características de Qualidade

- Norma ISO 25010 – *Software Product Quality Model*
  - Sobrepõe a antiga série ISO 9126
  - Possui 8 características de qualidade principais para produtos de software e suas subcaracterísticas



Modelo de qualidade do produto da ISO/IEC 25010 (ISO/IEC, 2011)

Requisitos de Domínio

Software Science Lab

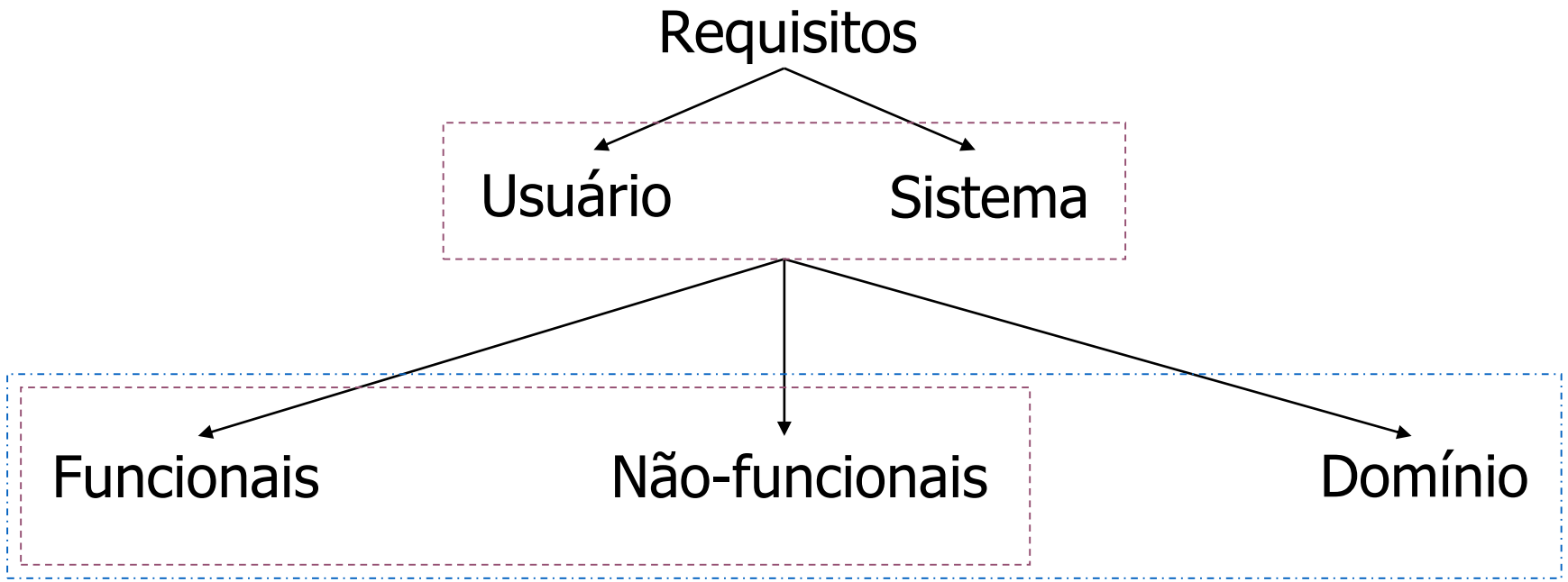
- Derivados do domínio da aplicação, comumente descrevem regras de negócio que devem ser atendidas independente do sistema
- Normalmente essas regras ajudam a detalhar requisitos funcionais com restrições específicas sobre requisitos existentes ou detalhes de computações específicas
- Se requisitos de domínio não forem satisfeitos, **o sistema pode tornar-se não prático**

- Exemplos:

  [RN1] A retenção dos impostos de uma nota fiscal é calculada da seguinte forma: ...

  [RN2] Não podem ser concedidos empréstimos para pessoas físicas que não tenham comprovante de rendimentos.

Visões para Requisitos



Visões: Requisitos do Sistema e do Software

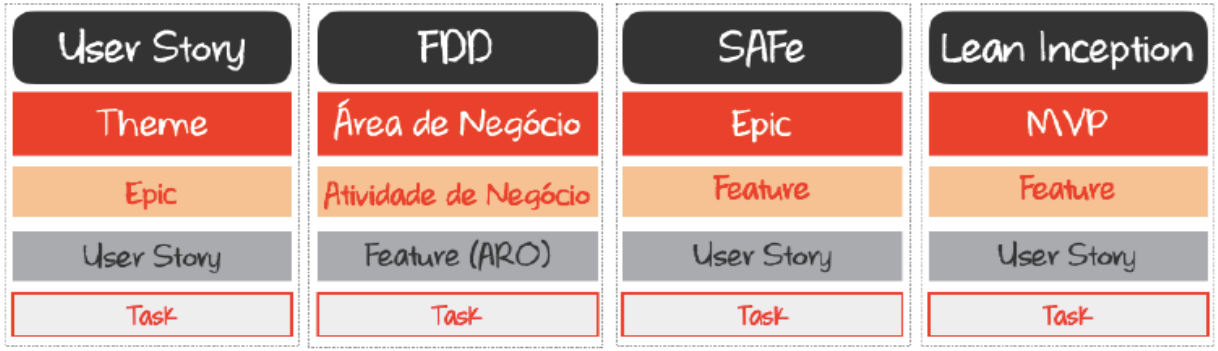
- Sistema
  - Combinação interativa de elementos (hardware, software, dados,...) para realizar um objetivo definido
- Requisitos do Sistema
  - São os requisitos do sistema como um todo
- Requisitos do Software
  - São os requisitos dos componentes de software derivados dos requisitos do sistema

Requisitos de Sistemas de Software Inteligentes

- Agenda:
  - ✓ Conceitos Básicos de Requisitos de Software
    - ✓ Engenharia de Requisitos
    - ✓ Tipos de Requisitos
  - ➢ **Requisitos em Contextos Ágeis**
    - Ideação (*Lean Inception, MVP Canvas*)
    - *Product Backlog Building*
    - User Stories e Cenários BDD
    - Gestão de Requisitos Ágil
  - Requisitos de Sistemas de Software Inteligentes
    - Visão Geral
    - ML Canvas
    - PerSpecML

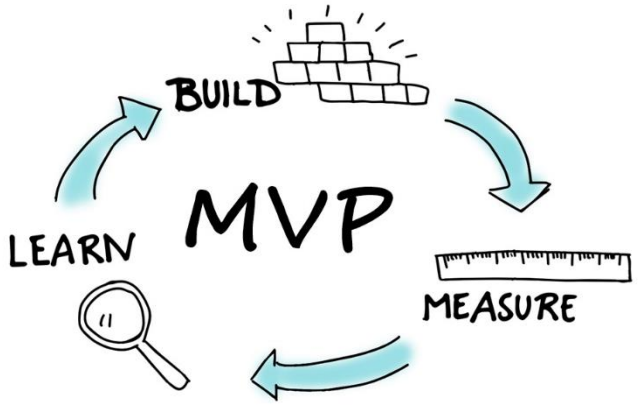
Requisitos em Contextos Ágeis

- Documento de Visão → Ideação e *MVP Canvas*
- Documento de Requisitos do Sistema → *Product Backlog* com *User Stories*



Ideação: O Ponto de Partida

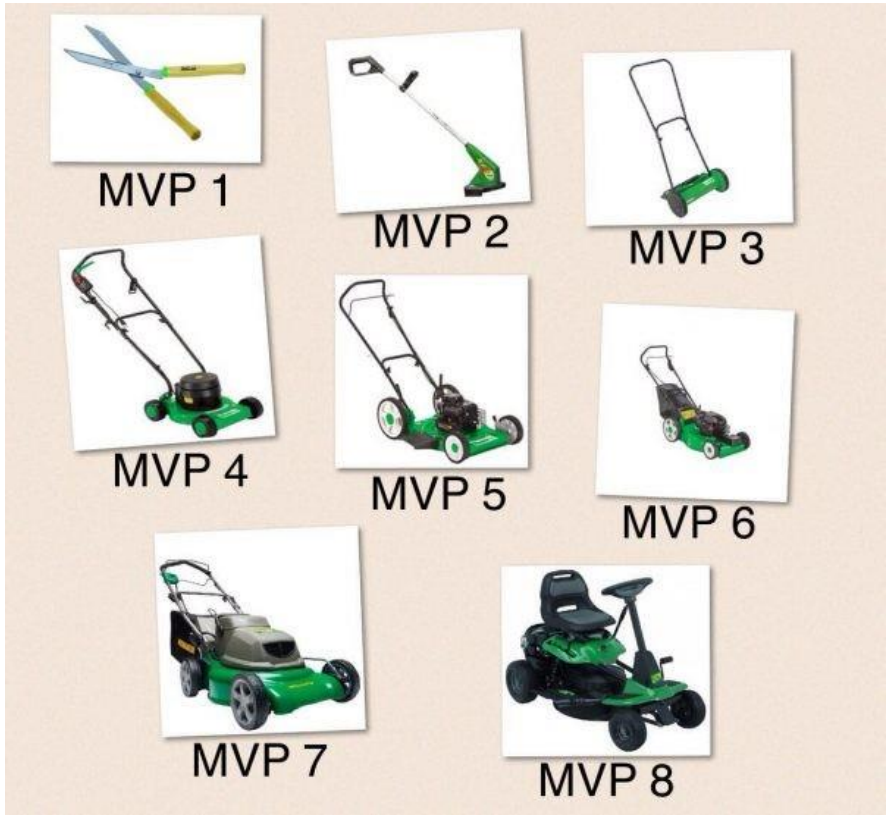
- Diversas Abordagens:
  - *Design Thinking*
  - *Design Sprints*
  - *Lean Startup*
  - *Lean Inception*



- *Lean Inception* é a combinação do *Design Thinking* com *Lean Startup* para decidir o **Produto Mínimo Viável** (MVP) (Caroli, 2018)

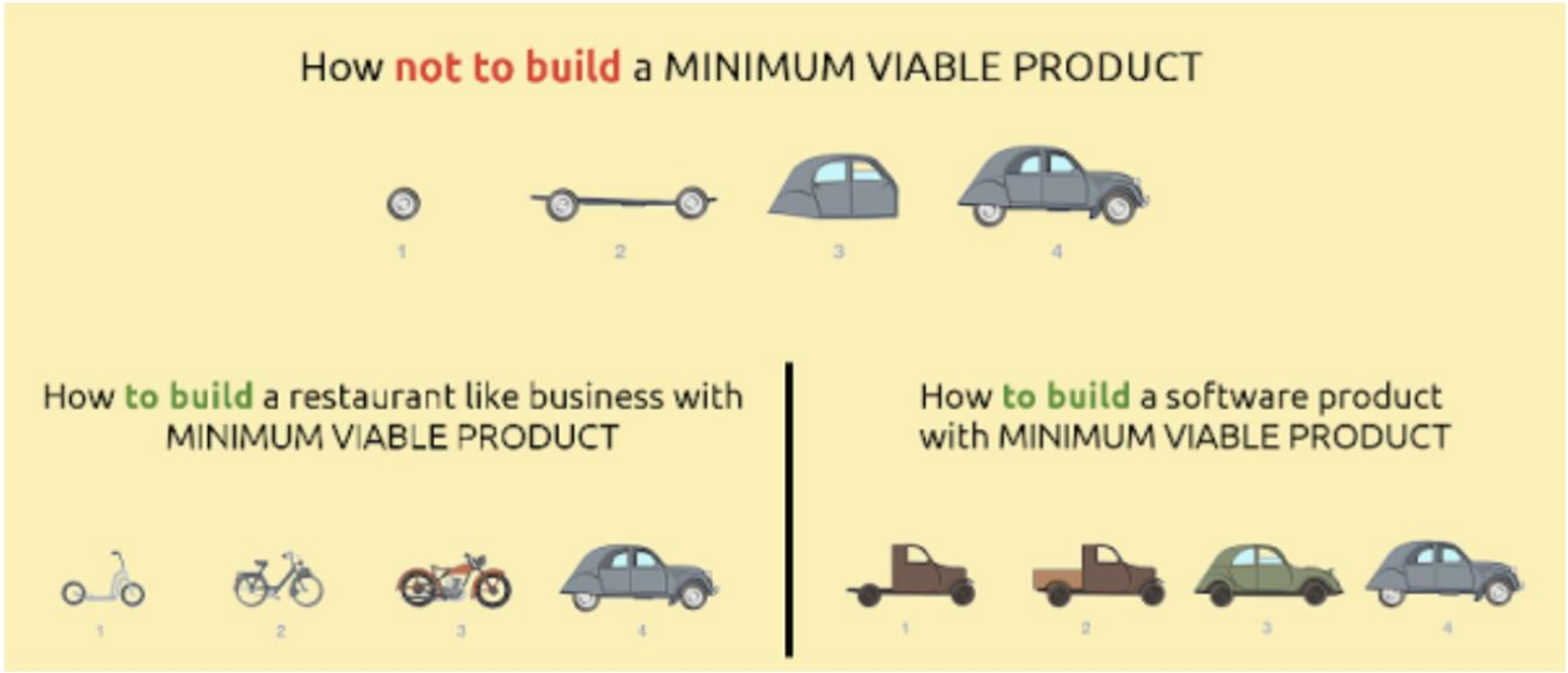
- Um MVP precisa permitir avaliar as hipóteses de negócio

Produto Mínimo Viável (MVP)



Produto Mínimo Viável (MVP)

- Atenção para MVPs na área de software deve-se pensar em **prototipação evolutiva** para evitar grandes perdas com retrabalho



Lean Inception (Agenda)

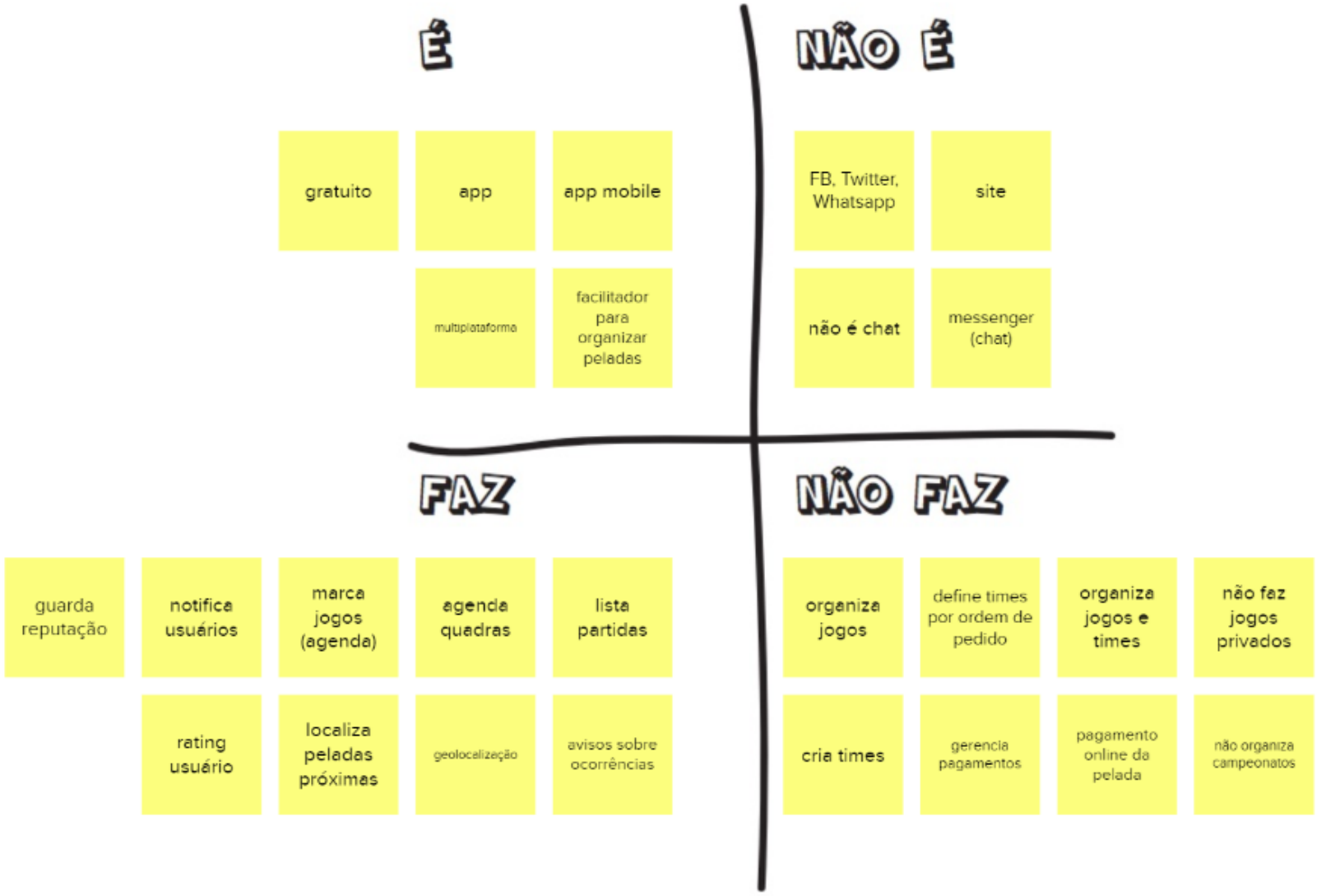
Exemplo de Lean Inception

Software Science Lab

- Aplicativo Easy-bola que visa facilitar que peladeiros encontrem jogos para maximizar as chances de acontecimento de partidas.

Visão do Produto

É – NÃO É | FAZ – NÃO FAZ

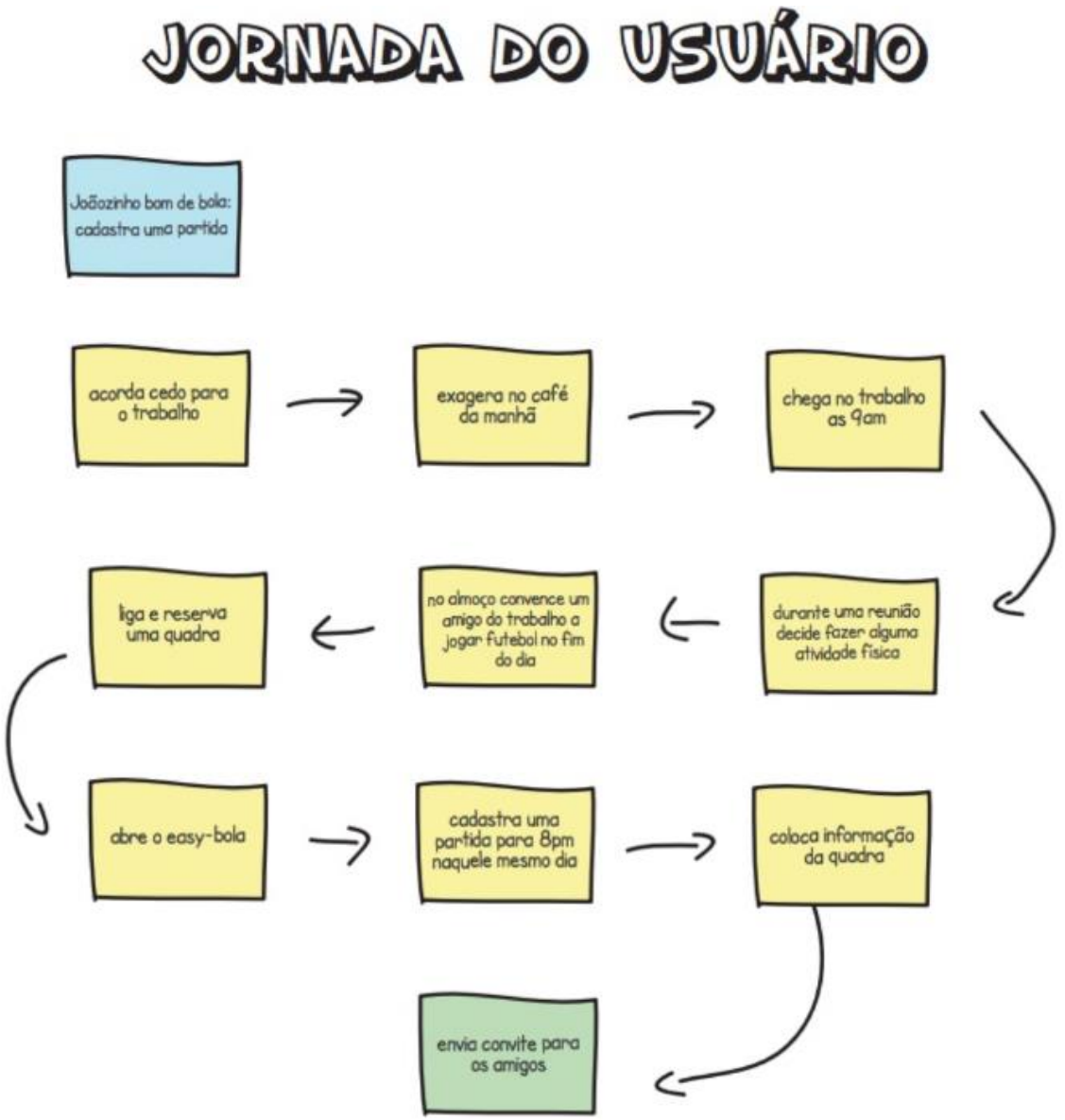


Objetivos do Produto

Personas

Software Science Lab

Jornadas



Software Science Lab

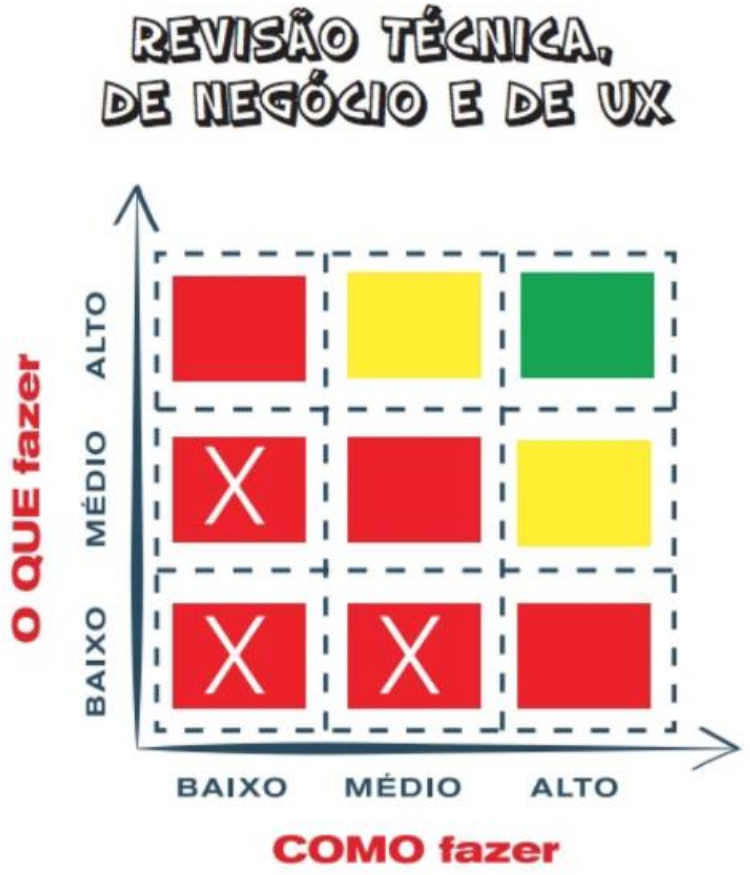
Brainstorm de Funcionalidades

Software Science Lab

consulta de peladas com geolocation	consulta de peladas sem geolocation	classificação de quadras	rankear um jogador	cadastro da pelada	convidar amigo para pelada	filtro detalhado	módulo de notificação
detalhamento da pelada (local, horário e data)	ranking dos jogadores (visualização)	detalhes financeiros da pelada	cadastro do peladeiro	confirmação de presença	notificação de pelada confirmada	notificação de pelada cancelada	cancelar presença
cancelar pelada							

Revisão Técnica de Negócios e de UX

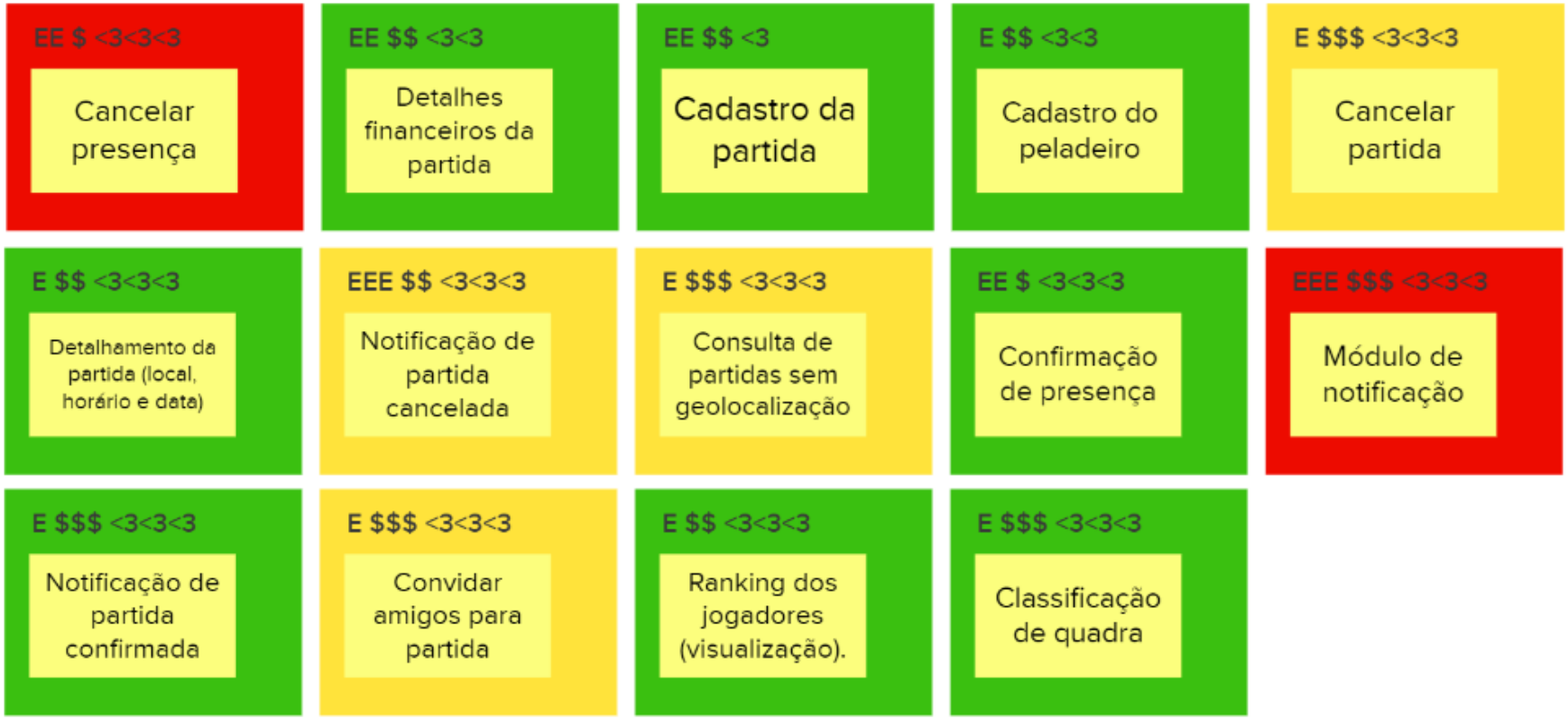
Software Science Lab



ESFORÇO	E	EE	EEE
NEGÓCIO	$	$$	$$$
UX	♥	♥♥	♥♥♥

Revisão Técnica de Negócios e de UX

Software Science Lab



Sequenciador

Regras:

Regra 1:
Uma onda pode conter no máximo três, cartões.

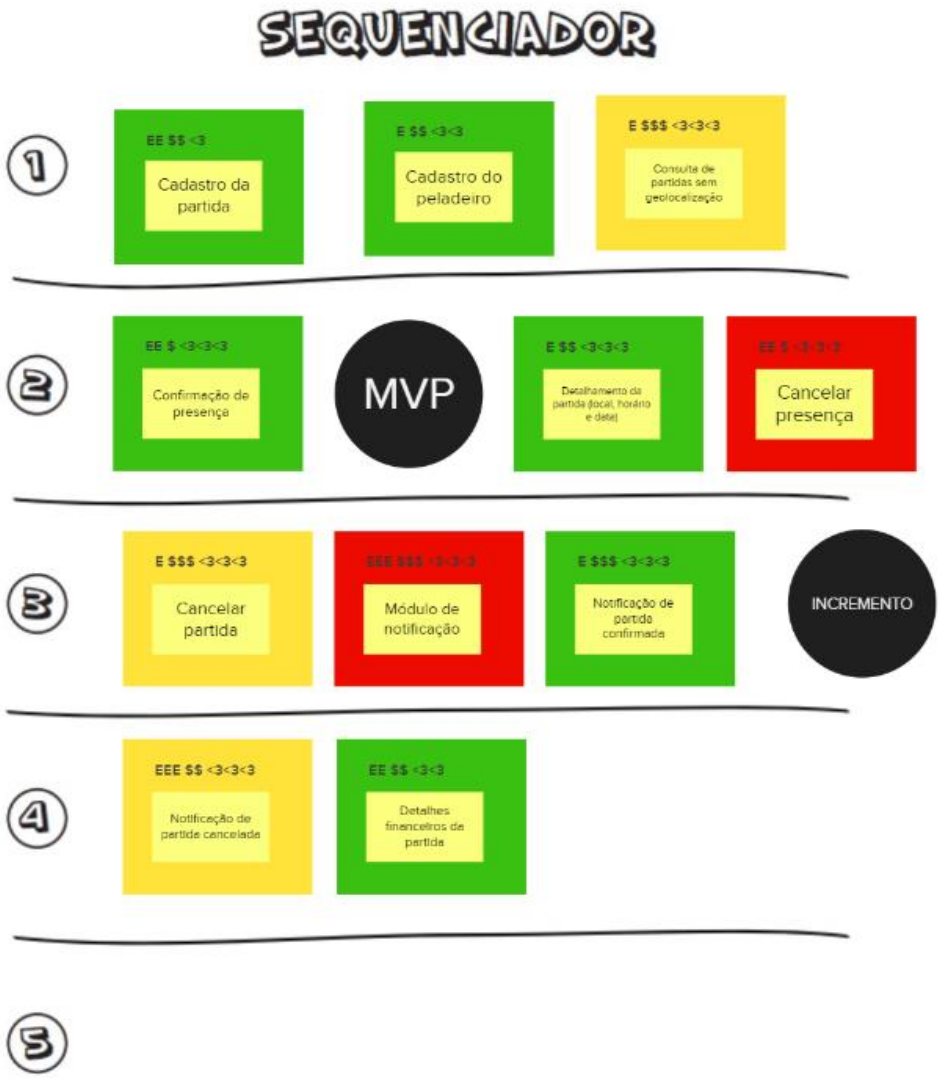
Regra 2:
Uma onda não pode conter mais de uma cartão vermelho.

Regra 3:
Uma onda não pode conter três cartões somente amarelos ou vermelho.

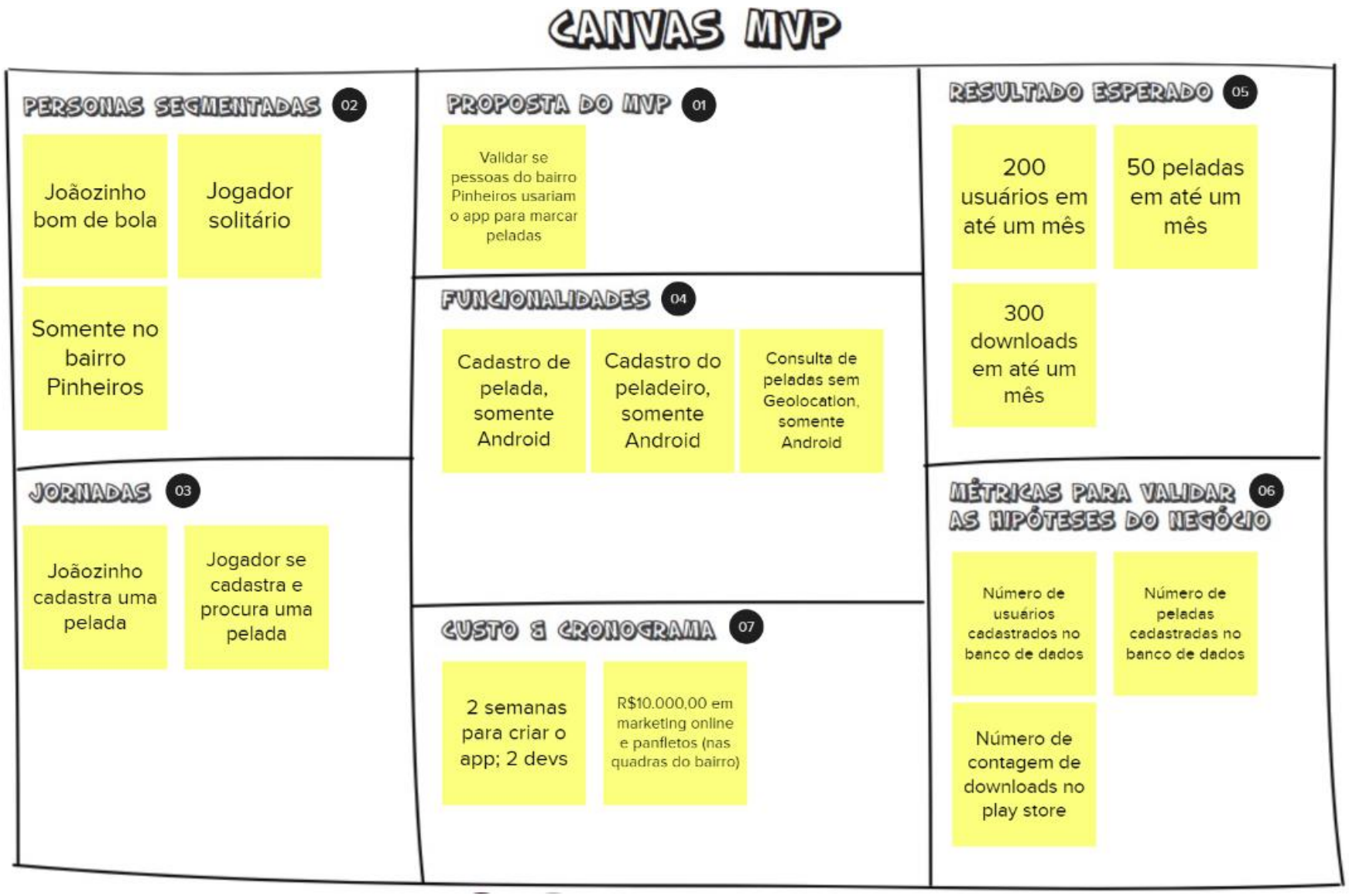
Regra 4:
A soma de esforço dos cartões não pode ultrapassar cinco Es.

Regra 5:
A soma de valor dos cartões não pode ser menos de quatro $s e quatro corações.

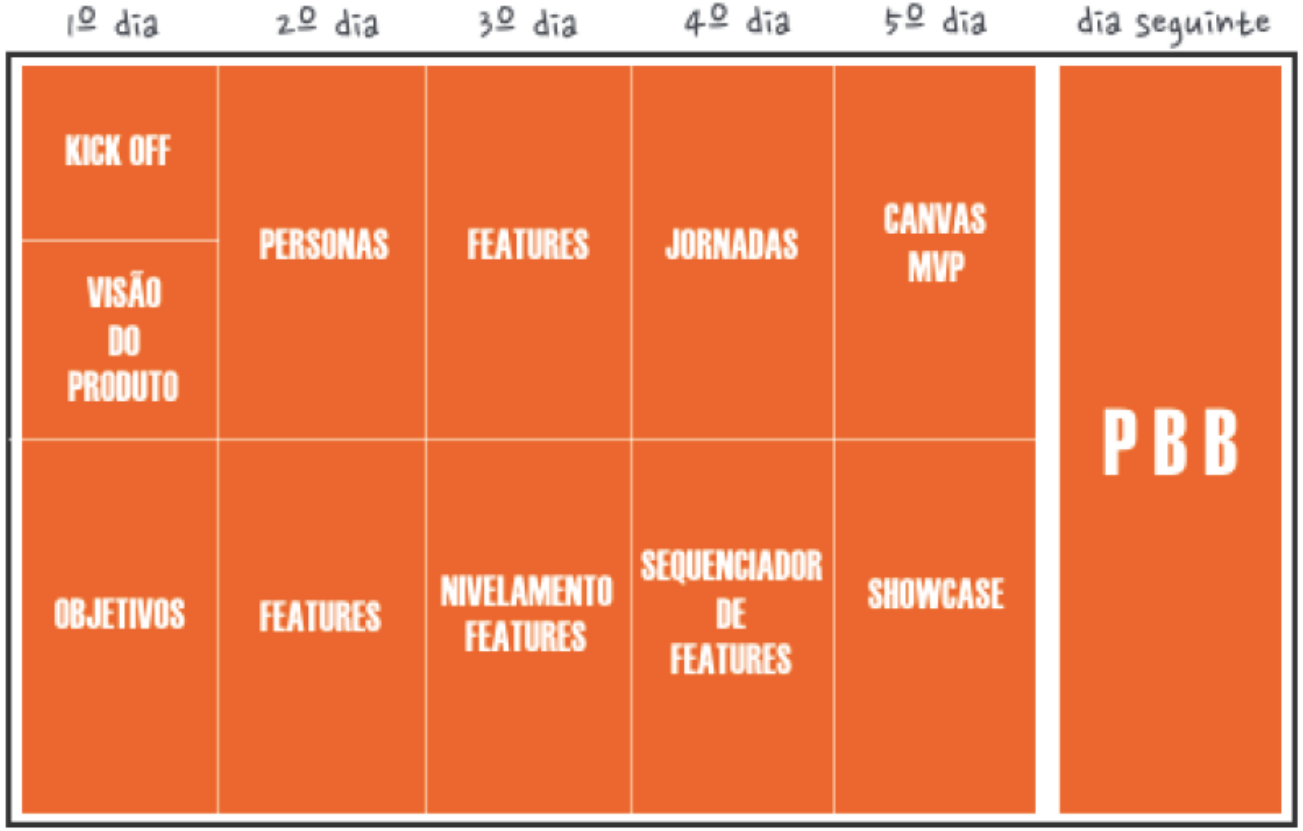
Regra 6:
Se um cartão depende de outro, esse outro deve estar em alguma onda anterior.



Canvas MVP


CANVAS MVP
PERSONAS SEGMENTADAS 02
Joãozinho bom de bola
Jogador solitário
Somente no bairro Pinheiros
PROPOSTA DO MVP 01
Validar se pessoas do bairro Pinheiros usariam o app para marcar peladas
FUNCIONALIDADES 04
Cadastro de pelada, somente Android
Cadastro do peladeiro, somente Android
Consulta de peladas sem Geolocation, somente Android
RESULTADO ESPERADO 05
200 usuários em até um mês
50 peladas em até um mês
300 downloads em até um mês
JORNADAS 03
Joãozinho cadastra uma pelada
Jogador se cadastra e procura uma pelada
CUSTO E CRONOGRAMA 07
2 semanas para criar o app; 2 devs
R$10.000,00 em marketing online e panfletos (nas quadras do bairro)
MÉTRICAS PARA VALIDAR AS HIPÓTESES DO NEGÓCIO 06
Número de usuários cadastrados no banco de dados
Número de peladas cadastradas no banco de dados
Número de contagem de downloads no play store

Product Backlog Building (PBB)



Product Backlog Building – Detalhando *Features*

Software Science Lab

Da Feature para o PBI

Representação dos passos:

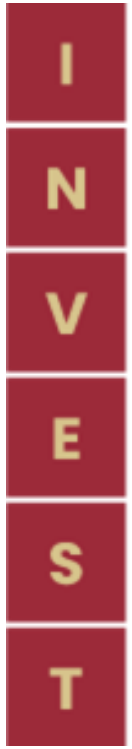
Modelo ARO

Realizar a consulta do produto

<ação><resultado><objeto>

Do PBI para a História de Usuário

User Stories



- Devem ser **I**ndependentes: dadas duas histórias X e Y, deve ser possível implementá-las em qualquer ordem
- Devem ser abertas para **N**egociação: são convites para conversas entre clientes e desenvolvedores durante uma iteração
- Devem agregar **V**alor para o negócio dos clientes
- Deve ser viável **E**stimar o tamanho de uma história
- Devem ser **S**ucintas para facilitar o entendimento e estimativa das mesmas
- Devem ser **T**estáveis: devem ter critérios de aceitação objetivos

Exemplos de User Stories

Feature: consultar partidas sem geolocalização

User story 1

- **Como** peladeiro não cadastrado
- **Quero** consultar partidas próximas a um endereço informado
- **Para** encontrar um jogo próximo ao meu local atual

User story 2

- **Como** peladeiro cadastrado
- **Quero** consultar partidas próximas ao meu trabalho
- **Para** encontrar um jogo próximo ao meu trabalho

User story 3

- **Como** peladeiro cadastrado
- **Quero** consultar partidas próximas à minha residência
- **Para** encontrar um jogo próximo à minha residência

Cenários BDD (Behavior-Driven Development)

Descreve um conjunto ordenado de comportamentos baseado em uma entrada para alcançar um resultado específico de uma funcionalidade. Costumam ser descrições adicionais utilizadas para a aceitação da história de usuário.

- Formato padrão (*template* ***given-when-then***)
  - **Dado que** <contexto?> **quando** <ação?> **então** <resultado?>
    - <contexto?> descreve em qual ponto o seu cenário irá iniciar.
    - <ação?> define a ação/evento que está sendo desempenhada no contexto descrito.
    - <resultado?> define o resultado esperado de suas ações.

Exemplos de Cenários BDD

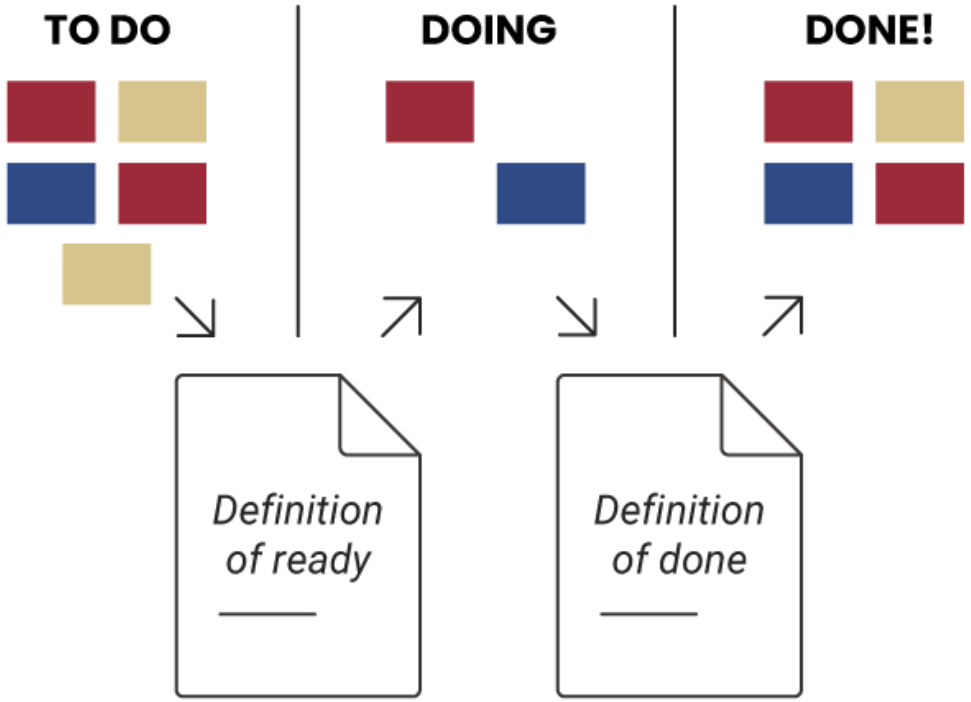
Software Science Lab

Critérios de aceitação da *user story 2*

- Dado que existe uma partida a menos de 10 quilômetros do meu trabalho
- Quando procuro por uma partida próxima ao meu trabalho
- Então encontro tal partida

- Dado que não existe nenhuma partida a menos de 10 quilômetros do meu trabalho
- Quando procuro por uma partida próxima ao meu trabalho
- Então não encontro nenhuma partida

Definition of Ready (DoR) e Definition of Done (DoD)

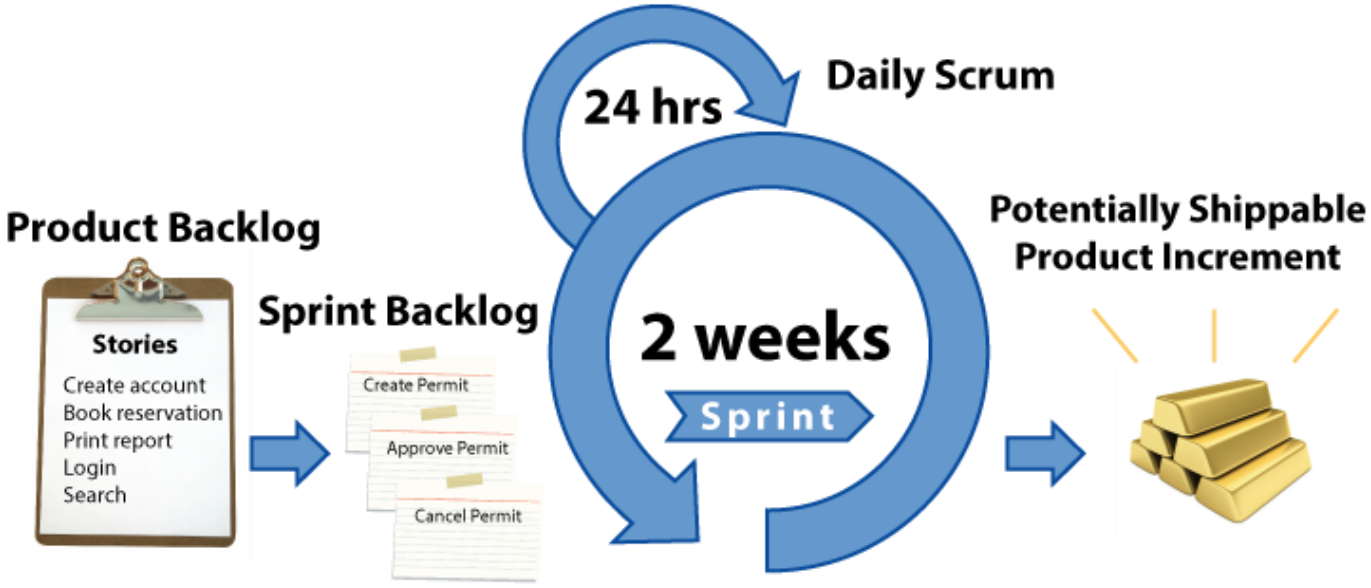


Definition of ready (preparado): descreve os requisitos que devem ser atendidos para que a *user story* possa ser movida do *backlog* do produto para o desenvolvimento

Definition of done (feito): descreve os requisitos que devem ser atendidos para que a implementação da *user story* possa ser classificada como concluída

Gerência de Requisitos no Contexto Ágil

- Em contextos ágeis mudanças de requisitos são normalmente tratadas atualizando o Backlog do Produto (Wagner et al., 2019)



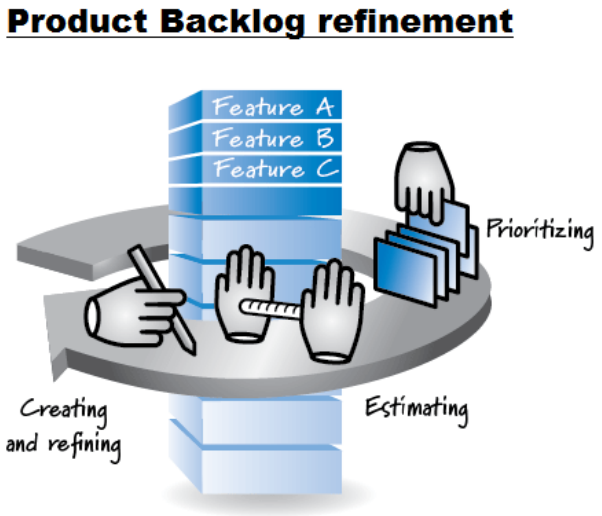
Refinamento do Backlog

Preparar os itens de *backlog* do produto para *sprints* futuras, refinando-os durante a *sprint* corrente para satisfazer a DoR

↓

Envolve organização, ordenação e limpeza do *backlog* do produto, incluindo tarefas como:

- Priorização dos itens do *backlog* do produto
- Descoberta de novos itens
- Preparação e refinamento dos itens mais importantes com as informações necessárias (critérios de aceitação, regras do negócio, fluxos, protótipos, aspectos não funcionais a serem considerados, etc.)
- Estimativa dos itens do *backlog* do produto (em caso de alterações)



Leituras Sugeridas

- WAGNER, S.; FERNÁNDEZ, D. M.; FELDERER, M.; VETRO, A.; KALINOWSKI, M.; WIERINGA, R.; PFAHL, D.; CONTE, T.; CHRISTIANSSON, M.; GREER, D.; LASSENIUS, C.; MÄNNISTÖ, T.; NAYEBI, M.; OIVO, M.; PENZENSTADLER, B.; PRIKLADNICKI, R.; RUHE, G.; SCHEKELMANN, A.; SEN, S.; SPÍNOLA, R. O.; TUZCU, A.; DE LA VARA, J. L.; AND WINKLER, D. **Status Quo in Requirements Engineering: A Theory and a Global Family of Surveys.** *ACM Transactions on Software Engineering and Methdology*, 28(2): 9:1-9:48. 2019.
- MENDES, T. S.; SOARES, H. F.; FARIAS, M. A. F.; MENDONCA, M.; KALINOWSKI, M.; SPINOLA, R. O. **Impacts of Agile Requirements Documentation Debt on Software Projects: A Retrospective Study.** In: ACM Symposium on Applied Computing (ACM SAC), p. 1300-1306, Pisa, Italy, 2016.
- WAGNER, S., MÉNDEZ-FERNÁNDEZ, D., KALINOWSKI, M. AND FELDERER, M., 2018. **Agile requirements engineering in practice: Status quo and critical problems**. *CLEI Electronic Journal*, *21*(1), p.15.

Requisitos de Sistemas de Software Inteligentes

Software Science Lab

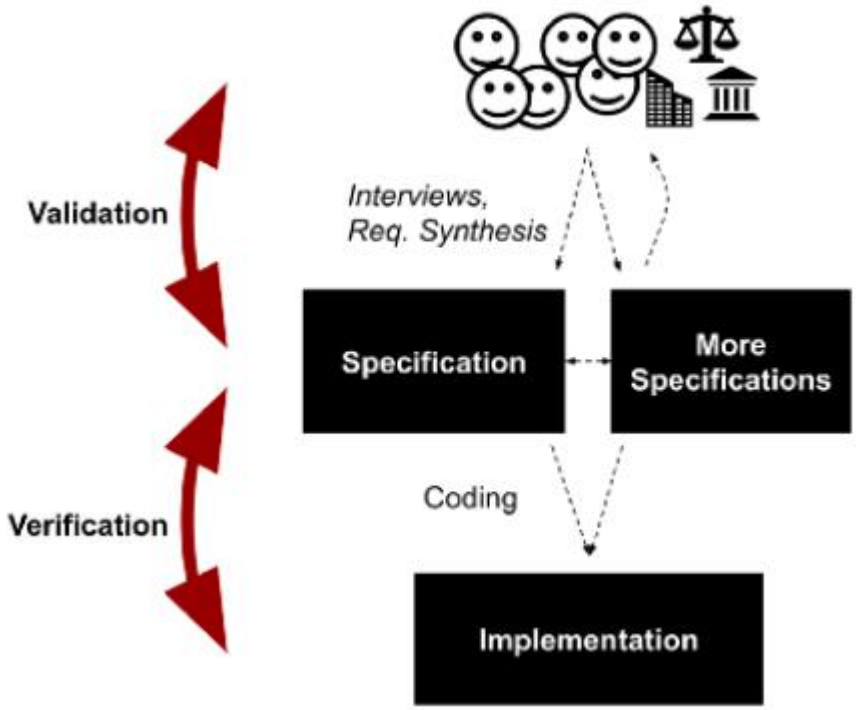
- Agenda:
  - ✓ Conceitos Básicos de Requisitos de Software
    - ✓ Engenharia de Requisitos
    - ✓ Tipos de Requisitos
  - ✓ Requisitos em Contextos Ágeis
    - ✓ Ideação (*Lean Inception, MVP Canvas*)
    - ✓ *Product Backlog Building*
    - ✓ User Stories e Cenários BDD
    - ✓ Gestão de Requisitos Ágil
  - ➢ **Requisitos de Sistemas de Software Inteligentes**
    - Visão Geral
    - ML Canvas
    - PerSpecML

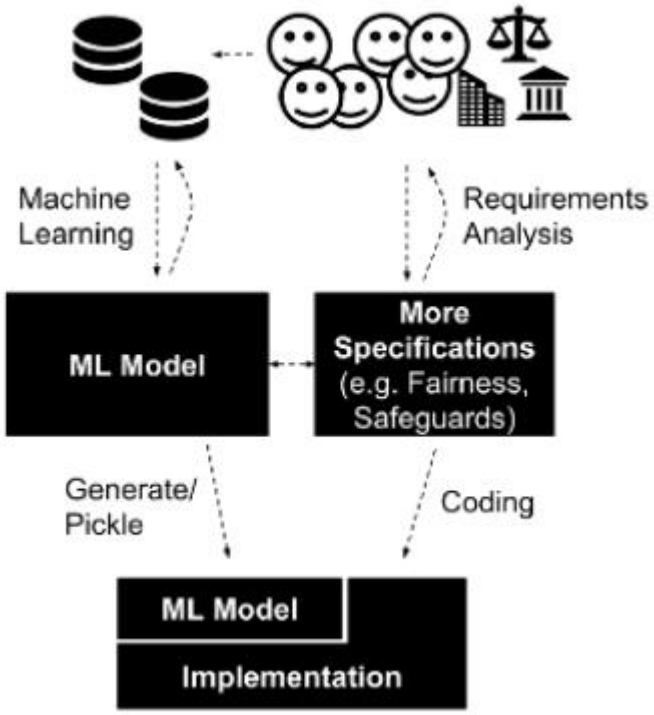
Requisitos de Sistemas de Software Inteligentes

- Contexto:
  - Modelos de ML são **especificações aprendidas a partir de dados**
  - Dados guiam o aprendizado de máquina
  - Dados substituem parcialmente o código em sistemas de ML

- O comportamento aprendido pode estar incorreto, mesmo se o algoritmo de aprendizado estiver corretamente implementado

Requisitos de Sistemas de Software Inteligentes





C. Kastner, “Machine learning is requirements engineering—on the ¨ role of bugs, verification, and validation in machine learning,” 2020. [Online]. Available: https://medium.com/analytics-vidhya/ machine-learning-is-requirements-engineering-8957aee55ef4

Requisitos de Sistemas de Software Inteligentes

- Engenharia de Requisitos (ER) é considerada a **atividade mais difícil do desenvolvimento de sistemas baseados em ML** [1].
- Técnicas de ER podem ajudar a:
  - Encontrar **métricas de qualidade** além da acurácia
  - Lidar melhor com **expectativas do cliente**
  - Entender por que **modelos não atendem** e para quem não atendem
  - Entender **quais dados estão faltando** e como os dados são analisados e generalizados

[1] H. Kuwajima, H. Yasuoka, and T. Nakae, "Engineering problems in machine learning systems," *Machine Learning*, vol. 109, no. 5, pp. 1103–1126, 2020.

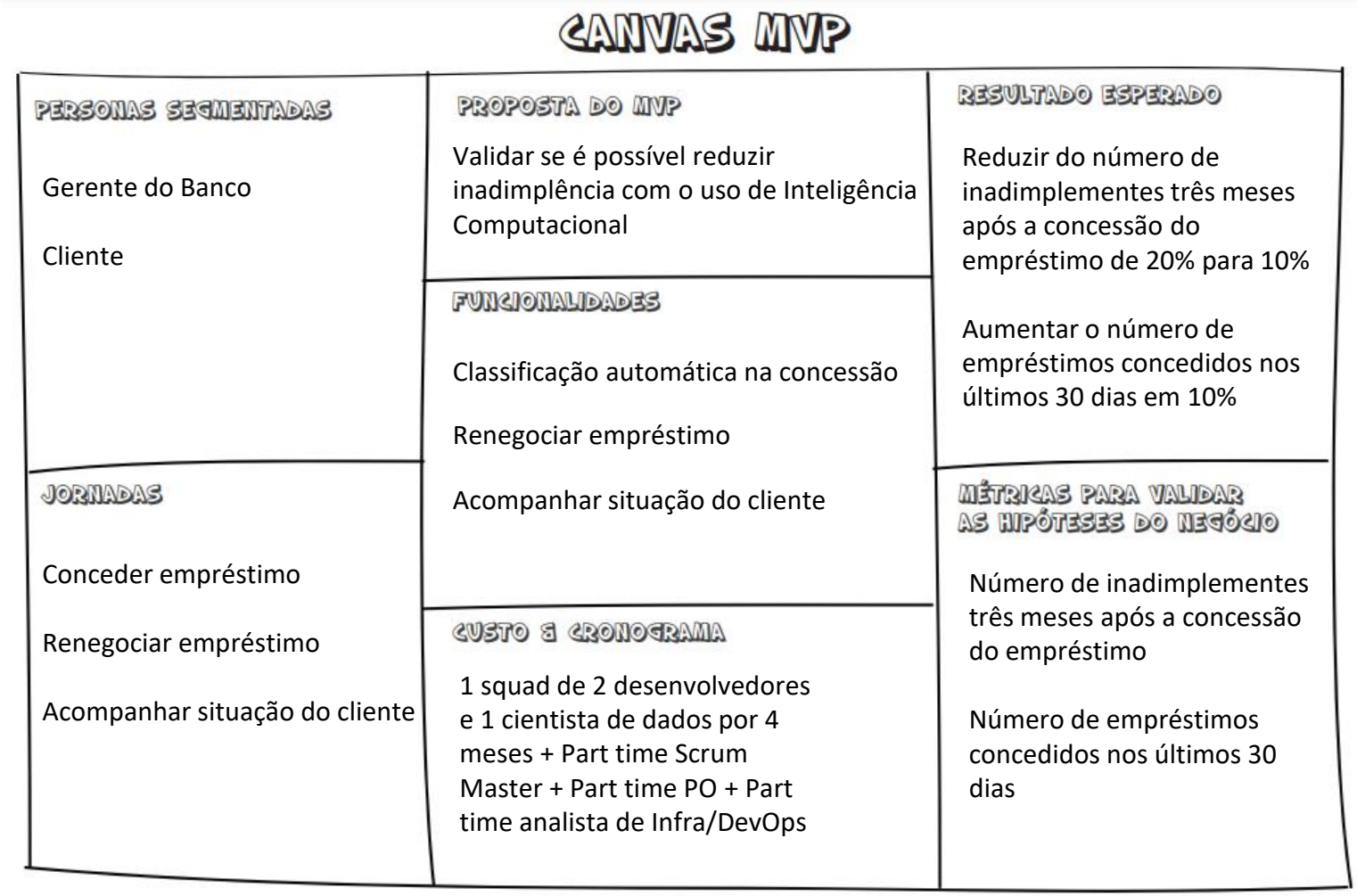
Características de Qualidade em Sistemas Baseados em ML

- Segurança
- **Explicabilidade**
- Privacidade
- **Qualidade dos Dados**
- ***Fairness***
- Transparência
- Confiabilidade
- ***Safety***
- Desempenho
- Manutenibilidade
- Requisitos legais
- *Testability*
- *Accountability*
- **Ética**
- Acurácia
- Adequação, Incerteza, Autonomia, Robustez, Modularidade, Escalabilidade, Usabilidade

Villamizar, H.; Escovedo, T.; and Kalinowski, M. **Requirements Engineering for Machine Learning: A Systematic Mapping Study.** In 47th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2021, Palermo, Italy, Sep 1-3, 2021.

Exemplo: MVP de Inteligência Computacional para Reduzir Inadimplências

Software Science Lab



Exemplo: MVP de Inteligência Computacional para Reduzir Inadimplências

Software Science Lab

CANVAS MVP

PERSONAS SEGMENTADAS

Gerente do Banco

Cliente

JORNADAS

Conceder empréstimo

Renegociar empréstimo

Acompanhar situação do cliente

PROPOSTA DO MVP

Validar se é possível reduzir inadimplência com o uso de Inteligência Computacional

FUNCIONALIDADES

Classificação automática na concessão

Renegociar empréstimo

Acompanhar situação do cliente

CUSTO E CRONOGRAMA

1 squad de 2 desenvolvedores e 1 cientista de dados por 4 meses + Part time Scrum Master + Part time PO + Part time analista de Infra/DevOps

RESULTADO ESPERADO

Reduzir do número de inadimplementes três meses após a concessão do empréstimo de 20% para 10%

Aumentar o número de empréstimos concedidos nos últimos 30 dias em 10%

MÉTRICAS PARA VALIDAR AS HIPÓTESES DO NEGÓCIO

Número de inadimplementes três meses após a concessão do empréstimo

Número de empréstimos concedidos nos últimos 30 dias

Features e User Stories

- Feature

  [F01] Classificação automática na concessão

- User Story

  [US01] **Como** Gerente do Banco **Quero** realizar a classificação automática de um cliente **Para** saber se o cliente irá pagar um empréstimo e decidir sobre a sua concessão

Detalhamento de User Stories de ML

[US01] **Como** Gerente do Banco **Quero** realizar a classificação automática de um cliente **Para** saber se o cliente irá pagar um empréstimo e decidir sobre a sua concessão

- Descrição detalhada:
  - **Tipo de problema de ML:** Classificação
  - **Entrada(s):** Renda mensal, valor de empréstimos vigentes, número de dependentes, idade, ...
  - **Saída(s):** Classificação automática de clientes novos em bons ou maus pagadores
  - **Fontes de dados:** Banco de dados do sistema de concessão de empréstimos e pagamentos
  - **Coleta dos dados:** Coletar dados diretamente do banco de dados de produção através de cópias mensais dos dados de até 6 meses atrás
  - **Dinâmica de criação/atualização do modelo:** Mensalmente com dados de até 6 meses atrás
  - **Dinâmica de execução do modelo:** Sob demanda
  - **Avaliação *offline* do modelo:** Utilização de base de testes para validar o modelo em dados não vistos durante o treinamento (*hold-out*)
  - **Monitoramento *online* do modelo:** Acurácia em tempo real referente aos últimos 6 meses
  - **Características de qualidade desejadas:** Qualidade dos dados (representatividade estatística da amostra, estratificação correta da base de treino e teste); Fairness/Ética (evitar viés de discriminação – quais devem ser testados?); Explicabilidade? (variáveis?); Confiabilidade (acurácia – quanto?); Desempenho (quanto?).

Cenários BDD

[US01] **Como** Gerente do Banco **Quero** realizar a classificação automática de um cliente **Para** saber se o cliente irá pagar um empréstimo e decidir sobre a sua concessão

- **Critérios de aceite da História US01**

  Dado que um bom pagador solicita um empréstimo
  Quando consulto a classificação automática
  Então o sistema me sugere conceder o empréstimo em pelo menos 95% dos casos

  Dado que um mau pagador solicita um empréstimo
  Quando consulto a classificação automática
  Então o sistema me sugere não conceder o empréstimo em pelo menos 95% dos casos

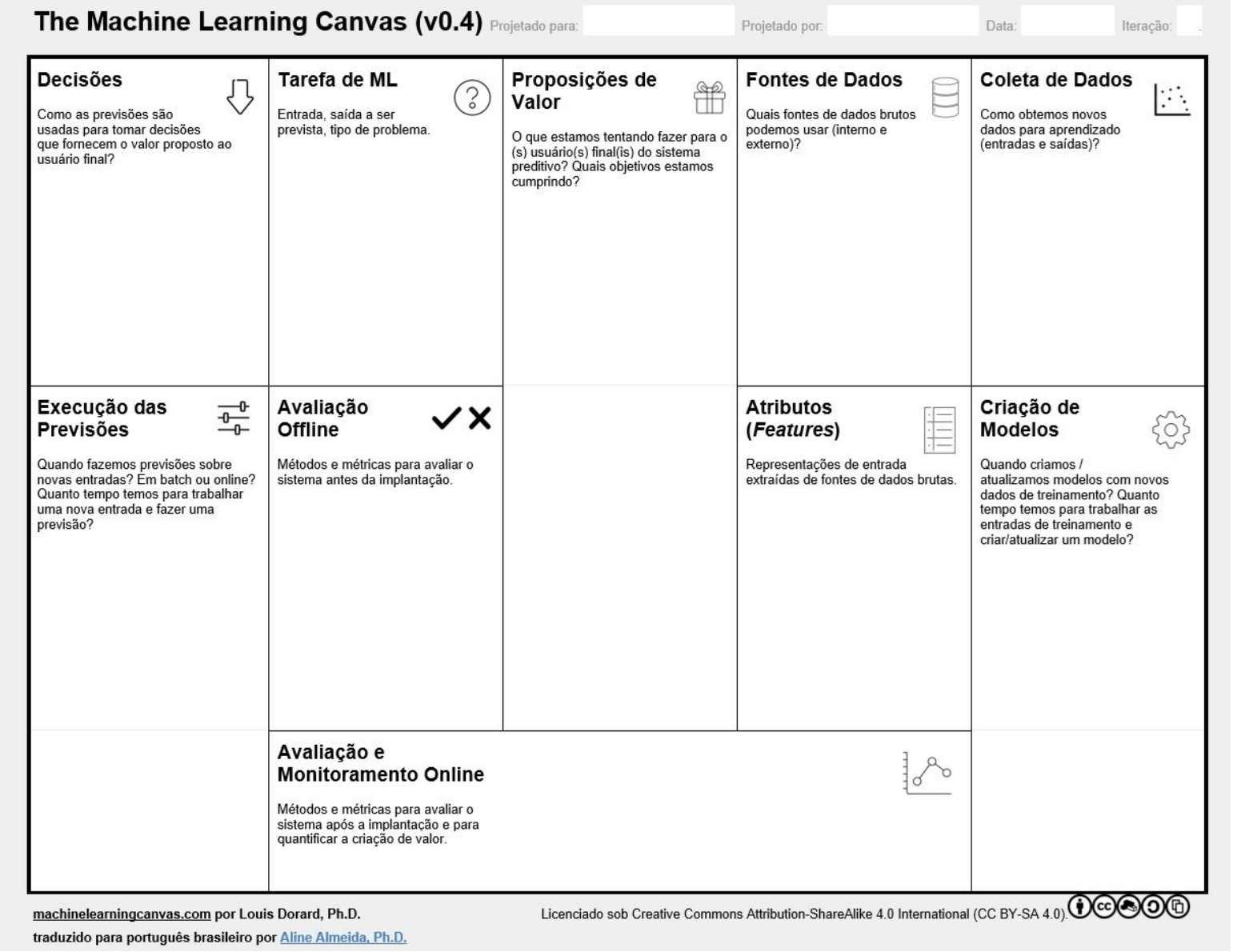
Mais Exemplos de Features e User Stories no Contexto de ML

- Classificação
  - Feature: [F01] Classificação automática na concessão de bons e maus pagadores
  - User story: [US01] **Como** Gerente do Banco **Quero** realizar a classificação automática de um cliente **Para** saber se o cliente irá pagar um empréstimo e decidir sobre a sua concessão
- Regressão
  - Feature: [F01] Predição do Faturamento
  - User story: [US01] **Como** diretor financeiro da empresa **Quero** consultar o faturamento esperado para o próximo trimestre **Para** apoiar o planejamento de investimentos

Mais Exemplos de Features e User Stories no Contexto de ML

- Agrupamento
  - Feature: [F01] Identificação de Localidade para Novas Filiais
  - User story: [US01] **Como** diretor de expansão **Quero** consultar o local mais promissor para abrir uma nova filial **Para** decidir sobre o local de abertura
- Associação
  - Feature: [F01] Oferta de novos produtos
  - User story: [US01] **Como** cliente da loja X **Quero** visualizar produtos relacionados aos meus interesses **Para** encontrar o produto ideal para minha necessidade

O Machine Learning Canvas



The Machine Learning Canvas (v0.4) Projetado para: Projetado por: Data: Iteração:

Decisões Como as previsões são usadas para tomar decisões que fornecem o valor proposto ao usuário final?	**Tarefa de ML** Entrada, saída a ser prevista, tipo de problema.	**Proposições de Valor** O que estamos tentando fazer para o (s) usuário(s) final(is) do sistema preditivo? Quais objetivos estamos cumprindo?	**Fontes de Dados** Quais fontes de dados brutos podemos usar (interno e externo)?	**Coleta de Dados** Como obtemos novos dados para aprendizado (entradas e saídas)?
Execução das Previsões Quando fazemos previsões sobre novas entradas? Em batch ou online? Quanto tempo temos para trabalhar uma nova entrada e fazer uma previsão?	**Avaliação Offline** Métodos e métricas para avaliar o sistema antes da implantação.		**Atributos (*Features*)** Representações de entrada extraídas de fontes de dados brutas.	**Criação de Modelos** Quando criamos / atualizamos modelos com novos dados de treinamento? Quanto tempo temos para trabalhar as entradas de treinamento e criar/atualizar um modelo?
	Avaliação e Monitoramento Online Métodos e métricas para avaliar o sistema após a implantação e para quantificar a criação de valor.			

machinelearningcanvas.com por Louis Dorard, Ph.D.
traduzido para português brasileiro por Aline Almeida, Ph.D.

- Vantagens:
  - Prático, captura informações essenciais para apoiar a construção de modelos de ML
- Desvantagens:
  - Desintegrado do restante do sistema e do processo de desenvolvimento
  - Claramente incompleto do ponto de vista de engenharia de software, não captura perspectivas essenciais do ponto de vista prático

PerSpecML

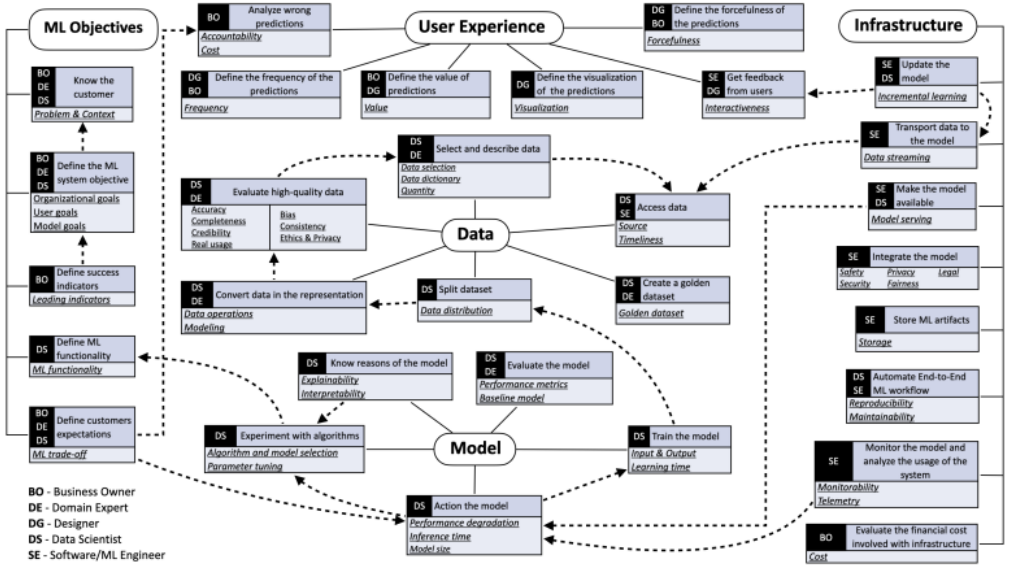
- É uma abordagem para apoiar a especificação de sistemas de software inteligentes que envolvem ML
- A abordagem é baseada na análise de 51 preocupações agrupadas em cinco perspectivas: **objetivos de ML, experiência do usuário, infraestrutura, modelo e dados**
- Juntas, essas perspectivas servem para mediar a comunicação entre *representantes do negocio, especialistas de domínio, designers, engenheiros de software/ML e cientistas de dados*

H. Villamizar, M. Kalinowski, H. Lopes, D. Mendez, L. Real, R. Archanjo, J. Pizani. **PerSpecML: A Perspective-Based Approach for Specifying Machine Learning-Enabled Systems.** International Conference on Software Engineering. In practice track (ICSE-SEIP), 2023 (submitted)

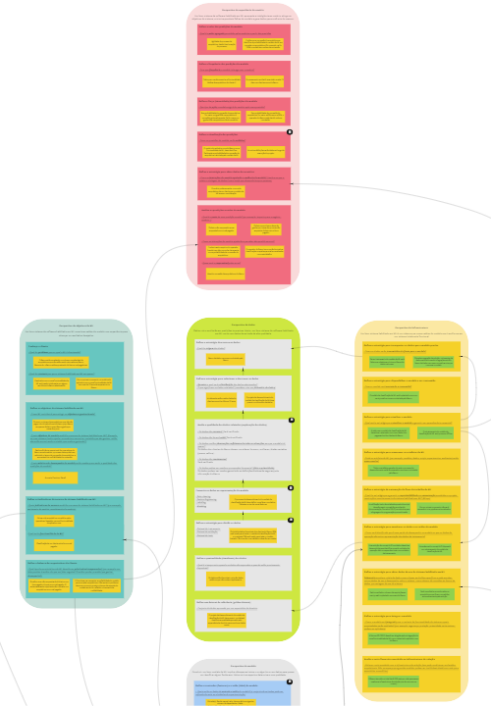
PerSpecML

Possui dois artefatos a serem usados:

- *Diagrama de tarefas e preocupações de ML baseado em perspectivas:*



- *Template de especificação de ML baseado em perspectivas:*



Perspectiva de objetivos de ML

- Fazer a ponte entre os objetivos de alto nível e as propriedades detalhadas dos modelos de ML é uma das causas mais comuns de falha.

- O sucesso em sistemas inteligentes é difícil de definir com uma única métrica. Uma boa prática é definir o sucesso em diferentes níveis.

Perspectiva de objetivos de ML

Software Science Lab

Preocupação	Abordar esta preocupação envolve especificar...
Problema & Contexto	o problema que ML abordará e seu contexto antes da codificação. ML deve ser direcionado ao problema certo.
Metas da organização	benefícios mensuráveis que ML deve trazer para a organização. Por exemplo, aumentar a receita em X%, aumentar o número de unidades vendidas em Y%.
Metas dos usuários	o que os usuários desejam alcançar usando ML. Por exemplo, para sistemas de recomendação, isso pode envolver ajudar os usuários a encontrar conteúdo de que gostem.
Metas do modelo de ML	métricas e medidas aceitáveis que o modelo de ML deve alcançar (por exemplo, para problemas de classificação, isso pode envolver precisão > X%, precisão > Y%, recall > Z%).
Indicadores de sucesso	medidas correlacionadas com o sucesso futuro. Isso pode incluir os estados afetivos dos usuários ao usar o sistema habilitado para ML (por exemplo, sentimento e engajamento do usuário).
Funcionalidade de ML	os resultados em termos de funcionalidade que o modelo de ML preverá (por exemplo, classificar clientes, prever probabilidades).
Trade-off de ML	o equilíbrio das expectativas do cliente (por exemplo, tempo de inferência versus precisão, falso positivo versus falso negativo).

Perspectiva de experiencia de usuário

- Um bom sistema de software habilitado ML inclui a criação de melhores experiências de uso de ML.

- O objetivo desta perspectiva é apresentar as previsões do modelo aos usuários de uma forma que atinja os objetivos do sistema e colete feedback dos usuários para melhorar o modelo.

- Conectar as saídas do modelo com os usuários é fundamental para alcançar o sucesso. No entanto, isso não é trivial e requer uma análise profunda.

Perspectiva da experiencia do usuário

Preocupação	Abordar esta preocupação envolve especificar...
Responsabilidade	quem é responsável pelos resultados inesperados do modelo de ML ou pelas ações tomadas com base nos resultados inesperados do modelo.
Custo	o impacto do usuário de uma predição errada do modelo.
Força	quão fortemente o sistema força o usuário a fazer o que o modelo de ML indica que ele deve fazer (por exemplo, ações automáticas ou assistidas).
Frequência	com que frequência o sistema deve interagir com os usuários (por exemplo, interaja sempre que o usuário solicitar ou sempre que o sistema achar que o usuário responderá).
Interatividade	quais interações os usuários terão com o sistema habilitado em ML (por exemplo, para fornecer novos dados para o aprendizado incremental do modelo).
Valor	o valor agregado percebido pelos usuários das predições ao seu trabalho.
Visualização	quais resultados de ML serão apresentados para que os usuários possam entendê-los (por exemplo, saída de um classificador, probabilidade da saída, especificar o painel e os protótipos de visualização para validação).

Perspectiva de infraestrutura

- Os modelos produzidos por cientistas de dados devem ser transformados em sistemas funcionais e conectados

- Isso implica cuidar de vários componentes, como ingerir e processar dados, criar novos aprendizados, implantá-los e monitorá-los

- É por isso que os engenheiros de software/ML desempenham um papel importante no desenvolvimento e orquestração de componentes de ML

Perspectiva de infraestrutura

Preocupação	Abordar esta preocupação envolve especificar...
Streaming dos dados	qual estratégia de streaming de dados será usada (por exemplo, em tempo real ou em lotes).
Custo	o custo financeiro envolvido com a infraestrutura que pode afetar as decisões de arquitetura. Grandes modelos podem ser inutilizáveis devido ao custo para executá-los e mantê-los.
Aprendizado incremental	a necessidade do sistema de aprender continuamente com novos dados, estendendo o conhecimento do modelo existente.
Integração	a integração que o modelo de ML terá com o restante da funcionalidade do sistema (por exemplo, segurança, safety, privacidade, justiça, legal)
Disponibilização de modelo	como o modelo de ML será executado e consumido (por exemplo, do lado do cliente, back-end, ponto final de serviço da Web).
Manutenibilidade	a necessidade de modificar o sistemas com componentes de ML para melhorar o desempenho ou se adaptar a um ambiente diferente.
Monitorabilidade	a necessidade de monitorar os dados e as saídas do modelo de ML para alertar/detectar quando os dados flutuam ou mudam.
Reproducibilidade	a necessidade de executar repetidamente um algoritmo/processo de ML em determinados conjuntos de dados/experimentos e obter os mesmos resultados (ou semelhantes).
Armazenamento	onde os artefatos de ML (por exemplo, modelos, dados, scripts) serão armazenados.
Telemetria	quais dados do sistema habilitado em ML precisam ser coletados. A telemetria envolve a coleta de dados como cliques em botões específicos e pode envolver outros dados de uso.

Perspectiva do modelo

- Um modelo de ML é um arquivo que foi treinado com um conjunto de dados para reconhecer certos tipos de padrões

- Construir um modelo implica não apenas treinar um algoritmo com dados para prever ou classificar bem algum fenômeno

- Muitos outros aspectos da qualidade de um modelo podem ser importantes ao operar um sistema

Perspectiva do modelo

Preocupação	Abordar esta preocupação envolve especificar...
Algoritmo & seleção do modelo	o conjunto de algoritmos que podem ser usados/investigados, com base no problema de ML e outras preocupações a serem consideradas (por exemplo, restrições de explicabilidade ou desempenho do modelo).
Ajuste de parâmetros dos algoritmos	a necessidade de escolher um conjunto de hiperparâmetros ótimos para um algoritmo de aprendizado. Um hiperparâmetro é um parâmetro cujo valor é usado para controlar o processo de aprendizado.
Modelo base	o modelo base que atua como referência (opcional). Sua principal função é contextualizar os resultados dos modelos treinados.
Explicabilidade	a necessidade de entender as razões para as inferências do modelo. O modelo pode precisar ser capaz de resumir as razões de suas decisões. Outras preocupações relacionadas, como transparência, podem ser aplicadas.
Interpretabilidade	
Tempo de inferência	o tempo aceitável para executar o modelo de ML e retornar as predições.
Entradas & Saída	as entradas (features) e resultados esperados do modelo. Obviamente, o conjunto de entradas pode ser refinado/aprimorado durante as atividades de pré-processamento, como a seleção das features.
Tempo de aprendizado	o tempo aceitável para treinar o modelo.
Tamanho do modelo	o tamanho do modelo em termos de armazenamento e sua complexidade (por exemplo, para árvores de decisão pode haver necessidade de poda).
Métricas de desempenho	as métricas usadas para avaliar o modelo (por exemplo, precisão, recall, erro quadrático médio) e expectativas de desempenho mensuráveis.
Degradação da performance	a consciência da degradação do desempenho do modelo de ML. Com o tempo, o desempenho preditivo de muitos modelos diminui à medida que um determinado modelo é testado em novos conjuntos de dados em ambientes em rápida evolução.

Perspectiva dos dados

- Os dados são essenciais para sistemas com componentes de ML
- Dados ruins resultarão em previsões imprecisas
- Por isso, ML requer dados de entrada de alta qualidade e tratamento adequado dos mesmos
- Do ponto de vista da requisitos, fica claro que os dados constituem um novo tipo de requisitos que precisam ser considerados ao projetar sistemas com componentes de ML

Perspectiva dos dados

Preocupação	Abordar esta preocupação envolve especificar...
Acurácia	a necessidade de obter dados corretos.
Vias	a necessidade de obter amostras justas de dados e distribuições representativas.
Completude	a necessidade de obter dados contendo observações suficientes de todas as situações em que o modelo irá operar.
Consistência	a necessidade de obter dados consistentes em um contexto específico.
Credibilidade	a necessidade de obter dados verdadeiros que sejam críveis e compreensíveis pelos usuários.
Operações nos dados	quais operações devem ser aplicadas nos dados (por exemplo, limpeza e rotulagem de dados).
Distribuição dos dados	as distribuições de dados esperadas e como os dados serão divididos em dados de treinamento, validação e teste.
Dicionário de dados	a coleção de nomes, definições e atributos para elementos e modelos de dados.
Seleção dos dados	o processo de determinar o tipo de dados apropriado e amostras adequadas para coletar dados.
Ética e privacidade	a necessidade de obter dados para evitar impactos adversos na sociedade.
Dataset golden	a necessidade de um conjunto de dados de referencia aprovado por um especialista de domínio.
Modelagem	o que é necessário para converter os dados na representação do modelo.
Quantidade	a quantidade de dados esperada de acordo com o tipo de problema e a complexidade do algoritmo.

PerSpecML Stakeholders

O **Business owner** geralmente entende como planejar projetos para tomar decisões. Como conectar objetivos de negócios com resultados de ML? O que representa o sucesso do ponto de vista empresarial?

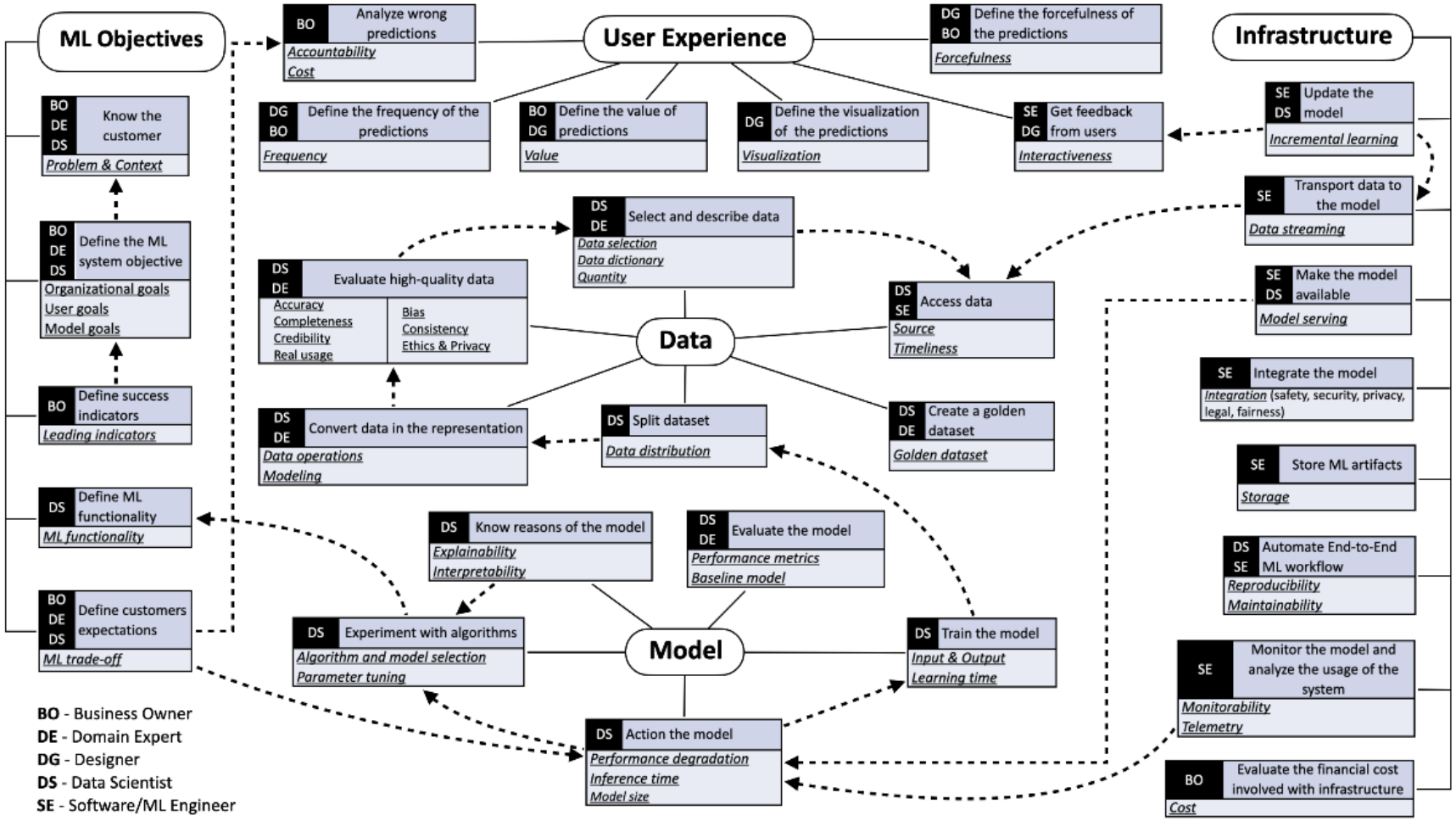
O **especialista de domínio** desempenha um papel importante na definição precisa do problema, proporciona insights sobre os dados e interpreta os resultados do modelo de ML.

O **designer** geralmente entende como melhorar a experiência dos usuários finais com base nas saídas do modelo de ML. Onde e com que frequência os resultados de ML devem aparecer? Com que força elas devem aparecer?

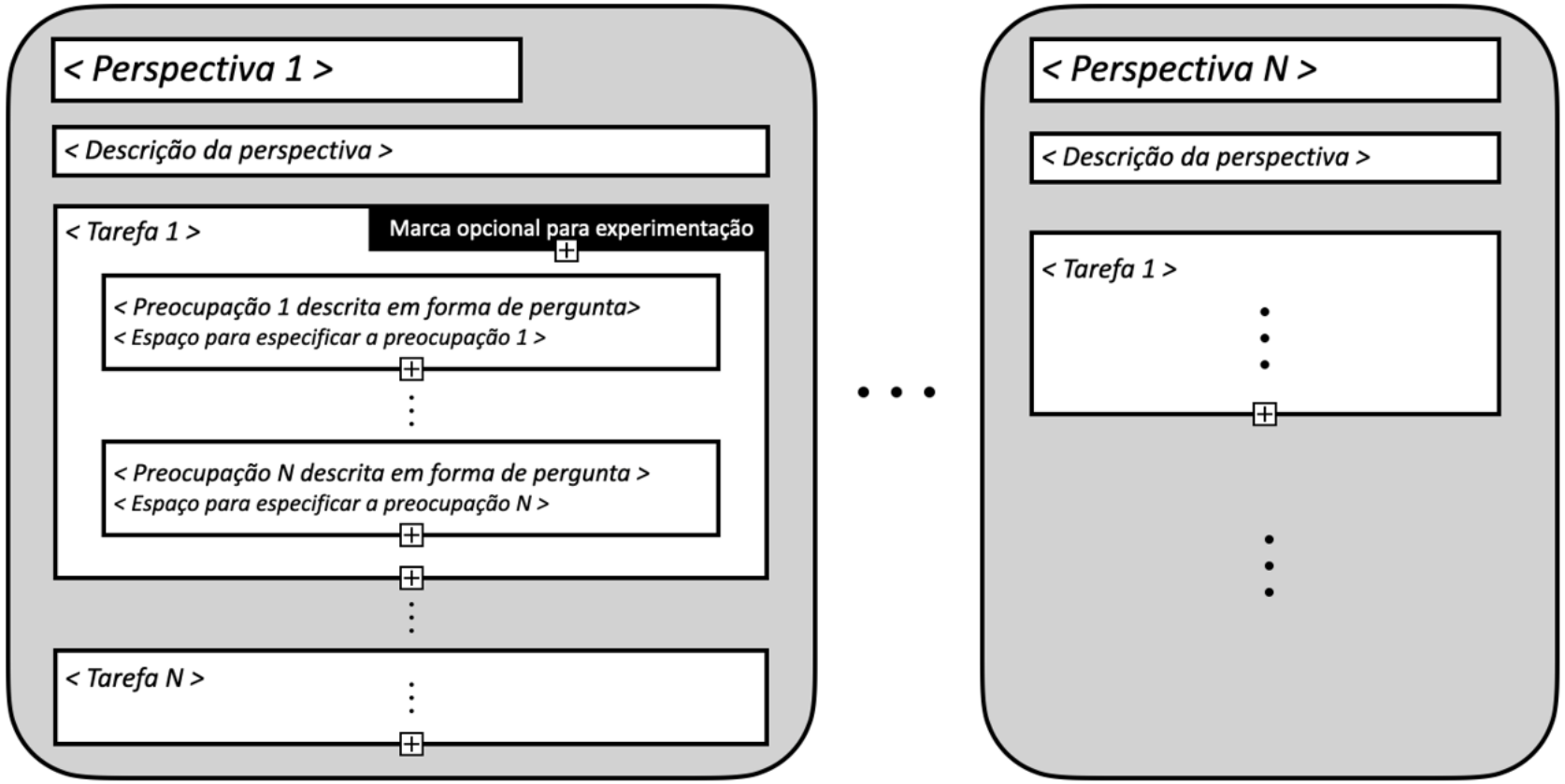
O **cientista de dados** normalmente entende as restrições dos modelos. Por exemplo, que tipo de algoritmos e modelagem podem ser usados para obter melhores resultados? Quais métricas representam melhor a qualidade do modelo?

O **engenheiro de software/ML** leva o modelo da fase de desenvolvimento para a fase de operação. Quais são os prós e os contras de disponibilizar o modelo como uma dependência ou como um serviço separado?

Diagrama de tarefas e preocupações



Template de especificação de ML



Resumindo

PerSpecML ajuda a:

- *Especificar e validar* requisitos de sistemas de software inteligentes
- *Fornecer uma visão geral do fluxo de trabalho* envolvido na construção de sistemas de software inteligentes
- *Comunicar os times envolvidos em projetos de ML* ao sinalizar as tarefas e sugerir os diferentes responsáveis

Exercício Prático

- Analisar o problema de reduzir inadimplências e especificar a funcionalidade de classificação automática da concessão, considerando as perspectivas e as preocupações

- Arquivos no Moodle

Leituras Sugeridas

- Capítulo sobre requisitos de algum livro texto de engenharia de software.
- F. Aguiar, P. Caroli, **Product Backlog Building**, Editora Caroli, 2019.
- S. Alonso, M. Kalinowski, M. Viana, B. Ferreira, and S.D.J. Barbosa, **A Systematic Mapping Study on the Use of Software Engineering Practices to Develop MVPs**, *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2021
- S. Amershi, A. Begel, C. Bird, *et al.*, **Software engineering for machine learning: A case study.** In *IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2019*
- D. M. Berry, **Requirements Engineering for Artificial Intelligence: What Is a Requirements Specification for an Artificial Intelligence?** Requirements Engineering: Foundation for Software Quality (REFSQ), pp. 19-25, 2022
- P. Caroli, **Lean Inception: how to align people and build the right product**, Editora Caroli, 2018.
- J.L. Correia, J.A. Pereira, R. Mello, A. Garcia, B. Fonseca, M. Ribeiro, R. Gheyi, M. Kalinowski, R. Cerqueira, W. Tiengo, **Brazilian Data Scientists: Revealing their Challenges and Practices on Machine Learning Model Development.** In *19th Brazilian Symposium on Software Quality*, 2020.
- B. Ferreira, M. Kalinowski, M., M.V. Gomes, M.C. Marques, H. Lopes, S.D.J. Barbosa, **Investigating Problem Definition and End-User Involvement in Agile Projects that Use Lean Inceptions.** In Proceedings of the XX Brazilian Symposium on Software Quality, SBQS'21, Brazil, November 8-11, pages 1-10, 2021.
- G. HULTEN, **Building Intelligent Systems: A Guide to Machine Learning Engineering**. Apress, 2018.
- M. Kalinowski, H. Lopes, A.F. Teixeira *et al.*, **Lean R&D: An agile research and development approach for digital transformation**, In *International Conference on Product-Focused Software Process Improvement* (PROFES), pp. 106-124, 2020.
- H. Villamizar, M. Kalinowski, H. Lopes, **A Catalogue of Concerns for Specifying Machine Learning-Enabled Systems.** Workshop on Requirements Engineering (WER), 2022
- H. Villamizar, M. Kalinowski, H. Lopes, **Towards Perspective-based Specification of Machine Learning-Enabled Systems.** Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2022
- H. Villamizar, T. Escovedo, M. Kalinowski, **Requirements Engineering for Machine Learning: A Systematic Mapping Study.** In 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2021
- S. Wagner, D.M. Fernández, M. Felderer, A. Vetrò, M. Kalinowski, R. Wieringa, D. Pfahl, T. Conte, M.T. Christiansson, D. Greer, C. Lassenius, **Status quo in requirements engineering: A theory and a global family of surveys**. ACM Transactions on Software Engineering and Methodology (TOSEM), 28 (2), 2019.

Software Science Lab

Introdução a Python e ao Google Colab

Engenharia de Software para Ciência de Dados

Prática: Introdução a Python

- https://colab.research.google.com/drive/1RVt_lKLXvRm_sglQqi5ts_u2GUYm87H0?usp=sharing

Princípios de Projeto e Boas Práticas de Codificação

Software Science Lab

- Agenda:
  - Introdução à Orientação a Objetos
  - Princípios SOLID
  - Guias de Estilos
  - Clean Code e Boas Práticas

Software Science Lab

Introdução à Orientação a Objetos com Python

Engenharia de Software para Ciência de Dados

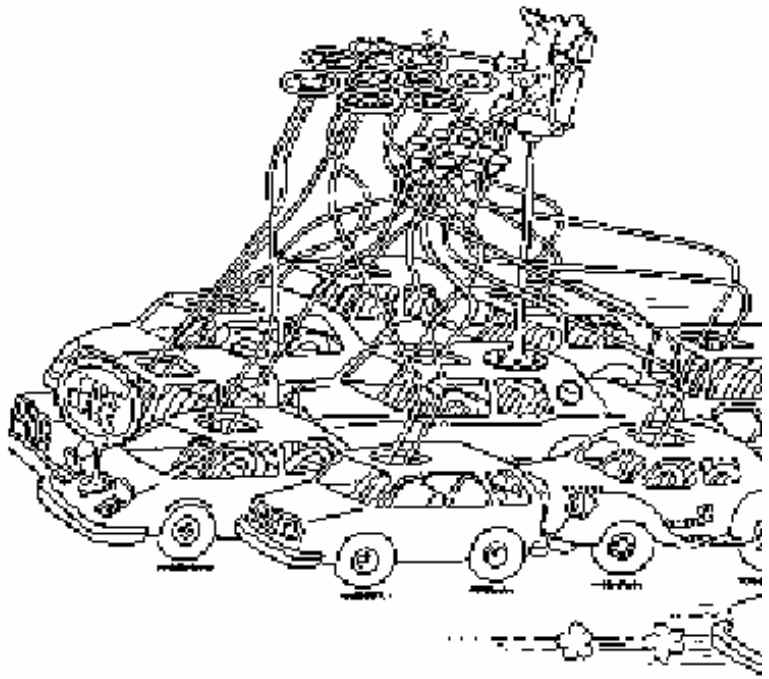
Paradigmas de Programação

Software Science Lab

- **Programação procedural:** baseada no conceito de chamadas a procedimentos (rotinas, métodos, funções)
- **Programação Orientada a Objetos (POO):** aproximação para a visão de mundo que temos: transformação, manipulação e utilização de objetos
  - Um **programa OO** é estruturado como um conjunto de objetos, cada um deles provê serviços ou efetua ações e é usado por outros objetos
  - Na **POO** uma ação é realizada por meio da **troca de mensagens** entre objetos

Classes e Objetos

- **Classes** estão para **objetos** assim como **formas** estão para **bolos**: podemos preparar vários bolos a partir de uma mesma forma, assim como podemos criar vários objetos a partir de uma mesma classe
- Cada objeto terá sua própria **identidade**



Orientação a Objetos

Software Science Lab

Classe

- Estrutura que integra **dados** e **comportamento**
- Representa o conjunto de **operações válidas** que acessam e manipulam os seus respectivos valores
- Usada para construir **objetos**: é o **projeto** dos objetos, e fornece a descrição do comportamento (operações) comum e do estado (dados) dos objetos

Objeto

- Instância de uma classe
- Tem **estado** (seus dados, o que o objeto sabe sobre si mesmo) e **comportamento** (o que o objeto faz)
- Os objetos construídos a partir de uma classe compartilham **comportamento comum**
- Sinônimo de **instância**

Vantagens da Orientação a Objetos

Software Science Lab

- Objetos são representados de forma **clara**, com **responsabilidades bem definidas** e **independentes dos outros objetos**
- Isto permite um código **mais claro** e de **fácil manutenção**, além de permitir que trechos de códigos sejam **facilmente reutilizados** em outros sistemas

Os 4 Pilares OO

- **Encapsulamento:** funciona como uma "embalagem" ao redor do objeto, protegendo-o do acesso indevido por outros objetos
- **Abstração:** projetar a classe pensando nos comportamentos e estados que queremos que os objetos que serão criados a partir dela tenham (programar usando um nível de pensamento mais abstrato)
- **Herança:** permite que as características e comportamentos descritos para caracterizar uma classe "pai" (superclasse) possam ser herdados a sua classe "filha" (subclasse)
- **Polimorfismo:** permite que métodos com a mesma semântica tenham múltiplas implementações

Classe

- Estrutura de dados que une variáveis (propriedades) e funções (comportamentos)
- Em Python, tudo deriva de uma classe, inclusive tipos básicos como tipos numéricos e textos
- Veja o que acontece quando executamos o comando ***help(int)***:

```
>>> help(int)
Help on class int in module builtins:

class int(object)
 |  int(x=0) -> integer
 |  int(x, base=10) -> integer
 |
 |  Convert a number or string to an integer, or return 0 if no arguments
 |  are given.  If x is a number, return x.__int__().  For floating point
 |  numbers, this truncates towards zero.
 |
 |  If x is not a number or if base is given, then x must be a string,
 |  bytes, or bytearray instance representing an integer literal in the
 |  given base.  The literal can be preceded by '+' or '-' and be surrounded
 |  by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
 |  Base 0 means to interpret the base from the string as an integer literal.
 |  >>> int('0b100', base=0)
```

- O tipo **int** deriva da classe **int** que, por sua vez, herda da classe **object** atributos e métodos comuns a todas as classes

Classe

- Em Python, classes podem ser definidas em **qualquer lugar**, inclusive dentro de funções

- Usualmente, classes são definidas mais próximas do **início do programa** ou em **módulos próprios**, o que costuma deixar o código mais limpo e fácil de manter

- A sintaxe para definir uma classe é:

```
class NomeDaClasse:
  DECLARAÇÕES e COMANDOS
```

- A palavra-chave **class** é usada para definir uma classe, e recomenda-se que seu nome utilize o padrão **CamelCase**

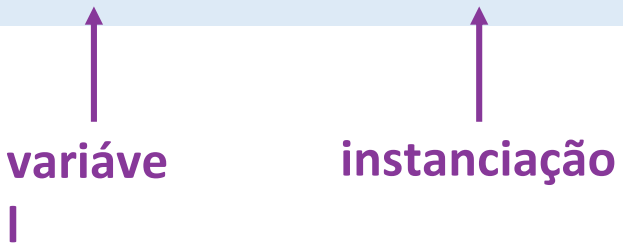
Exemplo

- Suponha que estamos escrevendo o código-fonte de um jogo de RPG
- Vamos começar escrevendo uma classe para definir um personagem:

```
>>> class PersonagemDeRPG:
...     pass
```

- Quando esta classe é usada para **criar** um personagem específico, estamos **instanciando** um objeto
- Quando este novo objeto é **atribuído** a uma variável, diz-se que essa variável **referencia** este objeto

```
>>> pesonagem =  PersonagemDeRPG()
```



Métodos

- Dentro de classes é possível definir **métodos**, que representam as ações que os objetos criados a partir desta classe poderão realizar
- Um método recebe pelo menos um parâmetro (a instância da classe - parâmetro ***self***), que permite ao método saber a qual instância se referem aqueles atributos
  - Na declaração do método é preciso declarar ***self*** explicitamente, mas ao chamar o método em um objeto, ***self*** é passado automaticamente
  - O parâmetro ***self*** pode ter outro nome, mas ***self*** é uma convenção (padrão PEP8)
- Métodos podem receber qualquer quantidade de parâmetros, que podem ser obrigatórios ou não (declarados com um valor *default*)

Exemplo

- Definimos uma classe **Pessoa** que possui dois métodos:
  - ***__init__***, inicializa o atributo nome durante a criação da instância
  - ***__str__***, converte uma **Pessoa** para **string**
- Quando passamos um objeto para o comando **print**, este vai buscar pelo método *__str__* para saber como representar o objeto na saída padrão

```
>>> class Pessoa:
...     def __init__(self, nome):
...         self.nome = nome
...     def __str__(self):
...         return self.nome

>>> pessoa_1 = Pessoa("João")
>>> print(pessoa_1)
João
>>> pessoa_2 = Pessoa("Maria")
>>> print(pessoa_2)
Maria
```

- As variáveis **pessoa_1** e **pessoa_2** referenciam instâncias (objetos) da classe Pessoa, com valores distintos para o atributo ***nome***

Exemplo

- Para chamar um método em um objeto, basta escrever o nome da variável que o referencia e o nome do método separados por um ponto, e passar os parâmetros cabíveis
- No exemplo, estamos chamando os métodos ***soma*** e ***media*** no objeto referenciado por ***minha_lista***

```
>>> class ListaDeNumeros:
...     def __init__(self, numeros):
...         self.numeros = numeros
...     def __str__(self):
...         return ", ".join([str(n) for n in self.numeros])
...     def soma(self):
...         return sum(self.numeros)
...     def media(self):
...         return self.soma()/len(self.numeros)

>>> minha_lista = ListaDeNumeros([1,3,5,7,9])

>>> print(f"A soma de {minha_lista} é {minha_lista.soma()}")
A soma de 1, 3, 5, 7, 9 é 25

>>> print(f"A média de {minha_lista} é {minha_lista.media()}")
A média de 1, 3, 5, 7, 9 é 5.0
```

Atributos

- Existem três tipos de variáveis que podem ser declaradas dentro de uma classe: **atributos de classes**, **atributos de instâncias** e **variáveis de métodos**:
  - **Atributos de classes** são atributos ligados à classe, e todas as instâncias da classe compartilham o mesmo atributo
  - **Atributos de instância** são específicos para cada instância, podendo ter valores diferentes de uma instância para outra
  - **Variáveis de métodos** existem apenas enquanto a função está sendo executada

Exemplo

- Criamos a classe Pessoa com o **atributo de classe** *num_pessoas*, utilizado para contar quantas instâncias de Pessoa foram criadas
- No método *__init__* criamos um **atributo de instância** *nome* para guardar o nome de cada indivíduo criado e incrementamos em uma unidade o **atributo de classe** *num_pessoas*

```
>>> class Pessoa:
...     num_pessoas = 0
...     def __init__(self, nome):
...         self.nome = nome
...         Pessoa.num_pessoas += 1
```

- Cada instância de Pessoa terá um valor distinto de *nome*, mas o valor de *num_pessoas* será o mesmo em todas, uma vez que atributos de classe são compartilhados entre todas as instâncias

Exemplo

- Agora criamos uma lista com quatro nomes e utilizamos uma *list comprehension* para inicializar quatro instâncias da classe Pessoa com cada um desses nomes
- Depois, imprimimos:
  - o valor de *Pessoas.num_pessoas* (note que o **atributo de classe** é acessado fazendo referência ao nome da classe); e
  - o valor do nome da instância localizada na primeira posição da lista (note que o **atributo de instância** é acessado fazendo referência à variável que referencia a instância)

```
>>> nomes = ['João', 'Paulo', 'George',
'Ringo']
>>> pessoas = [Pessoa(nome) for nome in nomes]
>>> Pessoa.num_pessoas
4
>>> pessoas[0].nome
'João'
```

Exemplo

- Note que a variável *res*, declarada dentro do método *soma* não aparece em nenhum dos atributos *__dict__*, nem no da instância, nem no da classe, pois a **variável de método** *res* só existe enquanto a função *soma* é executada.

```
>>> class Calculadora:
...     def soma(self, a, b):
...         res = a + b
...         return res
>>> calculadora = Calculadora()
>>> calculadora.soma(2,3)
5
```

Encapsulamento

- **Encapsulamento** é um recurso da OO que permite agrupar código e os dados que este código manipula em uma única entidade, protegendo estes elementos contra interferências externas, acesso indevido e/ou utilização/modificação inadequada por outro trecho de código definido fora do código encapsulado
- Uma **classe** naturalmente encapsula o código e os dados que a constituem, e é possível restringir o acesso a um membro (atributo ou método) da classe, permitindo acesso apenas para a classe em questão (**private**) ou em qualquer ponto do código (**public**)

Encapsulamento em Python

- Em OO, é uma boa prática tornar os atributos **privados**, uma vez que é responsabilidade de cada classe controlar os seus atributos
- Algumas linguagens utilizam o conceito de palavras chave para definir o nível de visibilidade, e Python considera que existem os tipos **public** e **non-public**.
  - Para marcar como *non-public* usa-se dois underscores ('__') antes do nome do atributo ou método.
  - Não se usa o termo *private* para Python porque nenhum atributo é de fato mantido como *private*.

Exemplo

- Como **numero** é *public*, ele pode ser acessado sem erros:

```
>>> class Conta:
...     def __init__(self, numero, saldo):
...         self.numero = numero
...         self.__saldo = saldo

>>> conta1234 = Conta(1234, 750.84)
>>> conta1234.numero
1234
```

- O interpretador acusou que o atributo **__saldo** (*non_public*) não existe na classe **Conta**.

```
>>> conta1234.__saldo

AttributeError: type object 'Conta' has no
attribute '__saldo’
```

- OBS: Em Python não existem atributos realmente privados (apenas um alerta de que você não deveria estar tentando acessá-lo), sendo possível acessar **__saldo** fazendo:

```
>>> conta1234._Conta__saldo
750.84
```

Boas Práticas de Encapsulamento

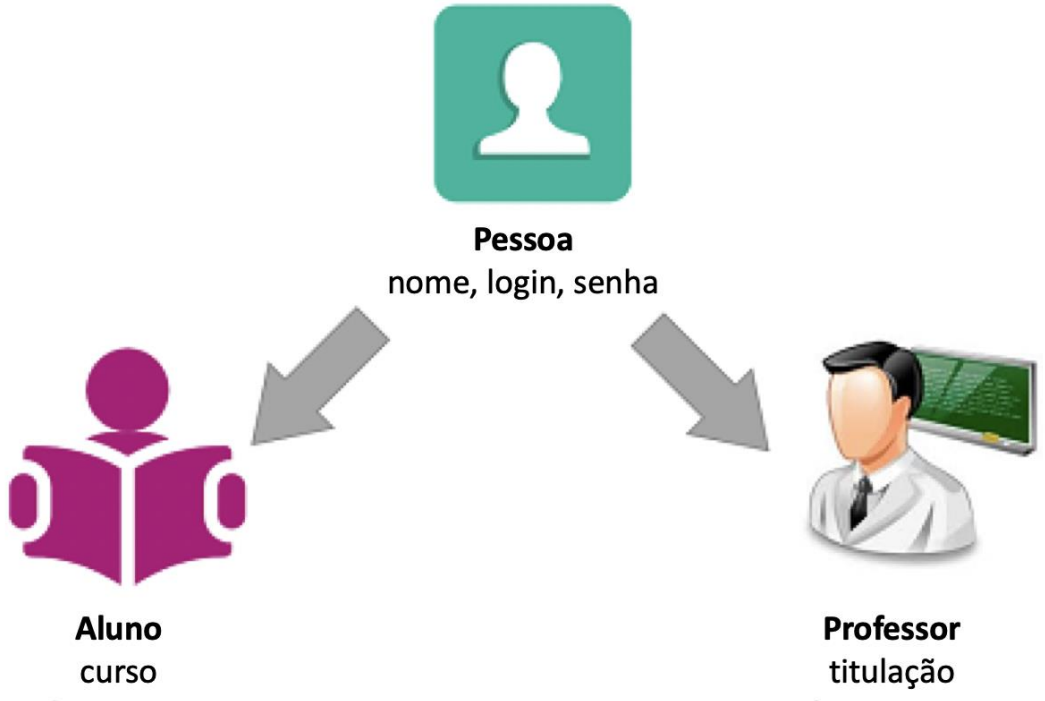
Software Science Lab

- As mesmas regras de acesso para atributos valem para os métodos
- Como **boa prática**, os atributos de instância são *non-public* e a maioria dos métodos, *public*
- Isto possibilita que a conversa entre objetos seja feita por troca de mensagens, ou seja, acessando seus métodos, em vez de um objeto manipular diretamente atributos que não sejam seus

```
>>> class Conta:
...     def __init__(self, numero, saldo):
...         self.__numero = numero
...         self.__saldo = saldo
...     def consulta_saldo(self):
...         return self._saldo
>>> conta1234 = Conta(1234, 750.84)
>>> conta1234.consulta_saldo()
750.84
```

Herança

- A **herança** permite que as características e comportamentos descritos para caracterizar uma classe "pai" (superclasse) possam ser herdados a sua classe "filha" (subclasse)
- Com a herança, é possível concentrar as partes **comuns** na classe pai e as partes **específicas** nas classes filhas



Exemplo

```
>>> # Classe Pessoa
>>> class Pessoa:
...     def __init__(self, nome, login, senha):
...         self.__nome = nome
...         self.__login = login
...         self.__senha = senha
...     def consulta_nome(self):
...         return self.__nome

>>> # Classe Aluno
>>> class Aluno(Pessoa):
...     def __init__(self, nome, login, senha, curso):
...         Pessoa.__init__(self, nome, login, senha)
...         self.__curso = curso
...     def consulta_curso(self):
...         return self.__curso

>>> # Classe Professor
>>> class Professor(Pessoa):
...     def __init__(self, nome, login, senha, titulacao):
...         Pessoa.__init__(self, nome, login, senha)
...         self.__titulacao = titulacao
...     def consulta_titulacao(self):
...         return self.__titulação
```

```
>>> pessoa1 = Pessoa('Maria', 'mary', 'm123')
>>> print(pessoa1.consulta_nome())
Maria

>>> aluna1 = Aluno('Viviane', 'vivi', 'v123',
'Informática')
>>> print(aluna1.consulta_nome())
Viviane
>>> print(aluna1.consulta_curso())
Informática

>>> prof1 = Professor('Tatiana', 'tati',
't123', 'Doutorado')
>>> print(prof1.consulta_nome())
Tatiana
>>> print(prof1.consulta_titulacao())
Doutorado
```

OBS: Ao definir o método *__init__* nas classes **Aluno** e **Professor**, o método *__init__* da classe **Pessoa** não é herdado.

Polimorfismo

- Podemos afirmar que todo *Aluno* é uma *Pessoa* (e todo *Professor* também), pois *Aluno* é uma extensão de *Pessoa*, e herda suas propriedades e métodos. Assim, podemos nos referir a um *Aluno* (ou um *Professor*) como sendo uma *Pessoa*.
- **Polimorfismo** é a capacidade de um objeto poder ser referenciado de várias formas (o que não significa que o objeto pode se transformar em outro tipo!)
  - Se tivermos um método que espera receber um objeto do tipo *Pessoa*, ele pode receber no lugar um objeto do tipo *Aluno* ou do tipo *Professor*.

Exemplo

```
>>> class EntradaUniversidade:
...     def __init__(self):
...         pass
...     def permite_entrada(self, pessoa):
...         print("Pode entrar, " +
pessoa.consulta_nome())
>>> entrada = EntradaUniversidade()
>>> entrada.permite_entrada(prof1)
Pode entrar, Tatiana
```

```
>>> # Classe Coordenador
>>> class Coordenador(Pessoa):
...     def __init__(self, nome, login, senha):
...         Pessoa.__init__(self, nome, login, senha)
>>> coord1 = Coordenador(Marcos', ‘mk', ‘m123')
>>> entrada.permite_entrada(coord1)
Pode entrar, Marcos
```

OBS: Repare que a funcionalidade representada pelo método **permite_entrada** foi construída antes mesmo que todos os tipos de **Pessoa** fossem criados, mas, ainda assim, ela continuou funcionando sem problemas após a criação da nova subclasse **Coordenador**, sem que fosse necessária nenhuma alteração no código.

Polimorfismo

Software Science Lab

- O polimorfismo é um recurso que permite projetarmos nossos códigos em um **alto nível de abstração** (pensando em superclasses), sem nos preocuparmos com os níveis mais baixos (subclasses) que poderão, inclusive, ser criados posteriormente
- Isto permite a criação de um código **flexível**, **simples** e de **fácil manutenção**
- Usando o polimorfismo, é possível diminuir o **acoplamento** entre as classes, evitando que modificações no código resultem em modificações em diversos outros lugares

Classes Abstratas

Software Science Lab

- Não faz sentido termos no nosso sistema uma classe **Pessoa**, pois os usuários do sistema serão de tipos específicos como **Aluno**, **Professor**, **Coordenador**... mas nunca tão genéricos como **Pessoa**
- Se removermos a classe **Pessoa** do nosso programa, perderemos duas importantes vantagens: o reuso de código entre as subclasses de **Pessoa** e a flexibilidade de termos um argumento polimórfico no método **permite_entrada**
- Podemos tornar **Pessoa** uma **classe abstrata**, que **não pode ser instanciada**

Classes Abstratas

Software Science Lab

- Em Python, uma classe abstrata deve conter pelo menos um método abstrato, e podemos criar uma classe abstrata herdando da superclasse para classes abstratas ABC (*Abstract Base Classes*), do módulo abc

```
>>> import abc
>>> class Pessoa(abc.ABC):
...     def __init__(self, nome, login, senha):
...         self.nome = nome
...         self.login = login
...         self.senha = senha
...     @abc.abstractmethod
...     def consulta_nome(self):
...         raise NotImplementedError()
```

- Se tentarmos instanciar agora um objeto do tipo **Pessoa**:

```
>>> pessoa1 = Pessoa()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class
Pessoa with abstract methods consulta_nome
```

OBS: Não é possível instanciar uma subclasse de **Pessoa** sem implementar o método abstrato **consulta_nome**. Ao torná-lo abstrato, ele deverá ser obrigatoriamente implementado em todas as subclasses de **Pessoa**.

Associação, Agregação e Composição

- Os conceitos de **associação**, **agregação** e **composição** permitem o **relacionamento** entre objetos
- Imagine que queremos ampliar a classe **Aluno** para guardar nela qual é o seu professor orientador, criando nela um atributo **nome_orientador** para guardar o nome do professor orientador e outros atributos para guardar as demais informações do professor orientador, como a sua titulação
- Esta alternativa, além de não ser interessante porque iria fazer com que a classe **Aluno** começasse a ter atributos em excesso, não está de acordo com a programação OO

Associação

- Podemos criar na classe **Aluno** um atributo **orientador**, do tipo **Professor**, que irá guardar todas as informações referentes ao orientador deste **Aluno**
- Assim, os atributos de uma classe também podem ser referências para outras classes
- Com isso, estamos fazendo uma **associação** de um professor a um aluno

```
>>> class Aluno(Pessoa):
...     def __init__(self, nome, login,
senha, curso, orientador):
...         Pessoa.__init__(self, nome,
login, senha)
...         self.__curso = curso
...         self.__orientador = orientador
...     def consulta_curso(self):
...         return self.__curso
...     def consulta_orientador(self):
...         return self.__orientador
...     def consulta_nome(self):
...         return self.nome
```

Associação

Software Science Lab

- Agora ao criarmos um **Aluno**, precisaremos passar um **Professor** como orientador:

```
>>> professorMarcos = Professor('Marcos', 'mk', 'm123', 'Doutor')
>>> novoAluno = Aluno('Isabela', 'isa', 'i123', 'Engenharia', professorMarcos)
```

- O professor orientador existe independente do aluno ao qual está associado, então dizemos que as classes **Professor** e **Aluno** estão relacionadas por uma **Associação** simples

- Quando semanticamente uma classe **faz parte** da outra temos uma **Agregação**, quando uma **é membro** da outra temos uma **Composição**

Associação 1:N

- Imagine agora que o aluno tem um **histórico de ocorrências**, com a data de matrícula e as ocorrências (notas obtidas, disciplinas cursadas, reprovações, etc):

```
>>> import datetime
>>> class Historico():
...     def __init__(self):
...         self.__data_matricula = datetime.datetime.today()
...         self.__ocorrencias = []
...     def imprime(self):
...         print("Matriculado em {}".format(self.__data_matricula))
...         print("Ocorrências:")
...         for o in self.__ocorrencias:
...             print("- ", o)
```

Associação 1:N

Software Science Lab

- Vamos modificar a classe **Aluno** para incluir um atributo do tipo **Historico**
- Também iremos incluir dois novos métodos, **gera_ocorrencia()** e **consulta_historico()**

```
>>> # Classe Aluno
>>> class Aluno(Pessoa):
...     def __init__(self, nome, login, senha, curso, orientador):
...         Pessoa.__init__(self, nome, login, senha)
...         self.__curso = curso
...         self.__orientador = orientador
...     def consulta_curso(self):
...         return self.__curso
...     def consulta_orientador(self):
...         return self.__orientador
...     def consulta_nome(self):
...         return self.nome
...     def gera_ocorrencia(self, ocorrencia):
...         self.__historico.__ocorrencias.append(ocorrencia)
...     def consulta_historico(self):
...         self.__historico.imprime()
```

Associação 1:N

Software Science Lab

- Criação de um novo aluno, inclusão de duas ocorrências no seu histórico e consulta do seu histórico:

```
>>> novoAluno = Aluno('Isabela', 'isa', 'i123', 'Engenharia', professorMarcos)
>>> novoAluno.gera_ocorrencia("Matriculou-se em Calculo 1")
>>> novoAluno.gera_ocorrencia("Nota final de Calculo 1: 9,7")
>>> novoAluno.consulta_historico()
Matriculado em 2021-11-18 19:23:03.908857
Ocorrências:
-  Matriculou-se em Calculo 1
-  Nota final de Calculo 1: 9,7
```

Agregação e Composição

- Tanto a agregação quanto a composição são relacionamentos **todo-parte**:
  - Na **agregação**, o objeto que compõe o todo tem uma parte do tipo de outro objeto e ambos podem existir separadamente.
    - Exemplo: **Revista** e **Artigo**
  - Na **composição**, se o objeto que representa o todo deixar de existir, o objeto que é uma parte relacionada a ele também deixará de existir.
    - Exemplo: **Livro** e **Capítulo**

Software Science Lab

Princípios SOLID

Engenharia de Software para Ciência de Dados

SOLID

- Os princípios SOLID buscam ajudar a realizar um bom projeto OO para um software de código flexível, escalável, sustentável e reutilizável
- SOLID é um acrônimo dos cinco princípios:
  - ***S**ingle Responsibility Principle (SPR)* - Princípio de Responsabilidade Única
  - ***O**pen/Closed Principle (OCP)* - Princípio Aberto / Fechado
  - ***L**iskov Substitution Principle (LSP)* - Princípio da Substituição de Liskov
  - ***I**nterface Segregation Principle (ISP)* - Princípio de Segregação de Interface
  - ***D**ependency Inversion Principle (DIP)* - Princípio da Inversão de Dependência

Princípio de Responsabilidade Única

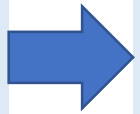
Software Science Lab

- *Uma classe não deve ter mais de um motivo para ser alterada*
- Uma classe deve ser especializada em um **único assunto** e possuir apenas **uma responsabilidade** dentro do software
- Quando uma classe tem mais de uma responsabilidade e surge uma necessidade de alteração, será difícil modificar uma dessas responsabilidades sem comprometer as outras
- Violar este princípio pode trazer problemas como **alto acoplamento**, **baixa coesão**, dificuldades na implementação de **testes automatizados** e para **reaproveitar o código**

Exemplo

- A classe **Animal** possui mais de uma responsabilidade. Ela representa um animal e seus comportamentos e também é responsável por salvá-lo no banco de dados

```
>>> class Animal:
...     def __init__(self, nome):
...         self.__nome = nome
...     def get_nome(self):
...         return self.__nome
...     def salvar(self):
...         # Salva o animal no banco de dados
...         pass
```



- Aplicando o princípio SRP, teremos uma classe para cada responsabilidade:

```
>>> class Animal:
...     def __init__(self, nome):
...         self.__nome = nome
...     def get_nome(self):
...         return self.__nome

>>> class AnimalDAO:
...     def salvar(self, animal: Animal):
...         # Salva o animal no banco de dados
...         pass
```

Princípio Aberto/Fechado

- ***Uma classe deve estar aberta para extensão, porém fechada para modificação***
- Quando novos comportamentos e recursos precisam ser adicionados no software, devemos **estender** e não alterar o código fonte original
- Por exemplo, organizar o seu código com uma superclasse genérica e abstrata o suficiente permitirá que o programa seja **estendido** acrescentando novas subclasses sem precisar alterar as classes existentes

Exemplo

- À medida que tivermos novos animais no nosso programa, teremos que modificar a classe **Animal**. Aplicando o OCP, teremos uma classe para cada tipo de animal:

```
>>> class Animal:
...     def __init__(self, nome):
...         self.__nome = nome
...     def get_nome(self):
...         return self.__nome
...     def faz_som(self):
...         if self.__nome == “Cachorro”:
...             print(“Au Au”)
...         if self.__nome == “Gato”:
...             print(“Miau”)
```

```
>>> class Animal:
...     def __init__(self, nome):
...         self.__nome = nome
...     def get_nome(self):
...         return self.__nome
...     def faz_som(self):
...         pass

>>> class Cachorro(Animal):
...     def faz_som(self):
...         print(“Au Au”)

>>> class Gato(Animal):
...     def faz_som(self):
...         print(“Miau”)
```

Princípio da Substituição de Liskov

Software Science Lab

- ***Uma subclasse deve poder ser substituída pela sua superclasse***
- Se **S** é um subtipo de **T**, então os objetos do tipo **T** podem ser substituídos pelos objetos de tipo **S** em tempo de execução
- Quando o LSP é seguido, a herança representa uma relação semântica que **garante a interoperabilidade entre os tipos da hierarquia**. Métodos públicos definidos nas classes filhas devem estar também presentes na classe pai, mesmo que de forma abstrata (***strong behavioral subtyping***)

Exemplo

Sem LSP:

```
class Animal:
  pass

class Cachorro(Animal):
  def latir(self):
    print("Au Au")

class Gato(Animal):
  def miar(self):
    print("Miau")
```

```
class Dono:

  def passear(cachorro: Cachorro):
    ...
    cachorro.latir()
```

Com LSP:

```
class Animal:
  def faz_som(self):
    pass

class Cachorro(Animal):
  def faz_som(self):
    __latir()
  def __latir(self):
    print("Au Au")

class Gato(Animal):
  def faz_som(self):
    __miar()
  def __miar(self):
    print("Miau")
```

```
class Dono:

  def passear(animal: Animal):
    ...
    animal.faz_som()
```

Princípio da Segregação de Interfaces

Software Science Lab

- ***Várias interfaces específicas são melhores do que uma interface genérica***
- Este princípio define que uma classe não deve conhecer nem depender de métodos que não necessite
- Interfaces devem ser segregadas com base nos requisitos ao invés de fornecer interfaces grandes de uso genérico

Exemplo

- Podemos utilizar a classe **ImpressoraFazTudo** como base a classe **ImpressoraMultifuncional**, porque uma impressora multifuncional imprime, digitaliza e envia fax
- Se utilizarmos a classe **ImpressoraFazTudo** como base se criar uma classe para uma **ImpressoraPadrao** violaremos o princípio ISP, porque ela irá herdar métodos que não utilizará (*digitaliza* e *envia_fax*)

```
>>> class ImpressoraFazTudo:
...     def imprime(self):
...         pass
...     def digitaliza(self):
...         pass
...     def envia_fax(self):
...         pass
```

```
>>> class ImpressoraMultifuncional(ImpressoraFazTudo):
...     def imprime(self):
...         pass
...     def digitaliza(self):
...         pass
...     def envia_fax(self):
...         pass
```

Exemplo

Software Science Lab

- Utilizando o princípio **ISP**, vamos dividir a classe **Impressora** em classes menores, para que as classes cliente implementem somente o que precisam:

```
>>> class Impressora:
...     def imprime(self):
...         pass
...
>>> class Digitalizadora:
...     def digitaliza(self):
...         pass
...
>>> class Fax:
...     def envia_fax(self):
...         pass
```

```
>>> class ImpressoraMultifuncional(Impressora, Digitalizadora, Fax):
...     def imprime(self):
...         pass
...     def digitaliza(self):
...         pass
...     def envia_fax(self):
...         pass
...
>>> class ImpressoraPadrao(Impressora):
...     def imprime(self):
...          pass
```

Princípio da Inversão de Dependências

- ***Devemos depender de abstrações e não de implementações***

- Programe para uma interface (o supertipo de maior abstração) e não para uma implementação (subtipo), isto permitirá desacoplar classes clientes de implementações específicas

Exemplo

```
class Animal:
  def faz_som(self):
    pass

class Cachorro(Animal):
  def faz_som(self):
    latir()
  def latir(self):
    print("Au Au")

class Gato(Animal):
  def faz_som(self):
    miar()
  def miar(self):
    print("Miau")
```

Obs: Note que a estrutura das classes não segue o LSP, permitindo violar o DIP

Sem DIP:

```
class Dono:

  def passear(cachorro: Cachorro):
    ...
    cachorro.latir()
```

Com DIP:

```
class Dono:

  def passear(animal: Animal):
    ...
    animal.faz_som()
```

Software Science Lab

Guia de Estilos

Engenharia de Software para Ciência de Dados

Motivação

- Um bom programador deve seguir **boas práticas** para escrever seus códigos, além de manter seu código **organizado** e fácil para que outros programadores consigam **entender** e dar **manutenção**
- Os **guias de estilo** contém práticas recomendadas para escrever e manter seus códigos de forma organizada, e são escritos e organizados por profissionais experientes
- O principal autor do guia de estilos da linguagem Python é o próprio **Guido Von Rossum**, criador do Python

PEPs

Software Science Lab

- A comunidade Python mantém atualizadas as características e processos da linguagem por meio das ***PEPs (Python Enhancement Proposal)***

- **Principais PEPs**
  - **PEP-0:** Indexação de todas as PEPs
  - **PEP-8:** Padrões de formatação de código mais indicadas para se utilizar com Python
  - **PEP-20:** Zen do Python (*Zen of Python*), uma lista de curtos pensamentos e ideais da programação com Python
  - **PEP-257:** Convenções para Docstrings

The Zen of Python

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Software Science Lab

Easter Egg

```
>>> import this
```

Layout de Código

- *“Bonito é melhor que feio”*
- Como você formata seu código tem papel fundamental na forma que ele é entendido
  - **Indentação:** pelo padrão da linguagem Python, devemos usar 4 espaços (ou 1 tab) por nível de indentação
  - **Tabulações ou Espaços:** nunca misture tabulações e espaços (a forma mais popular de indentar um código Python é somente com espaços, a segunda forma é somente com tabulações)
  - **Comprimento máximo de linhas:** limite todas as linhas de seu programa em um máximo de 79 caracteres

Longos Blocos

- Para **longos blocos de texto** (*docstrings* ou comentários), devemos limitar seu comprimento em 72 colunas.
- A melhor maneira de continuar linhas longas é usando a continuação implícita, entre parênteses, colchetes e chaves
- Se necessário, você pode adicionar um par extra de parênteses em uma expressão, mas, em alguns casos, uma barra invertida fica melhor

```
>>> class Rectangle():
...     def __init__(self, width, height,
...         color='black', emphasis=None, highlight=0):
...         if width == 0 and height == 0 and \
...             color == 'red' and emphasis == 'strong' or \
...             highlight > 100:
...             raise ValueError("sorry, you lose")
...         if width == 0 and height == 0 and \
...         (color == 'red' or emphasis is None):
...             raise ValueError("I don't think so")
```

Linhas em Branco

- Sempre separe as funções e definições de classes com duas linhas em branco, enquanto os métodos dentro de uma classe devem ser separados somente por uma única linha em branco
- Quando linhas em branco são usadas para separar métodos, deve haver também uma linha em branco entre a linha de definição da classe e o primeiro método da classe
- Use linhas em branco também para separar blocos lógicos dentro de métodos e funções

Imports

- Os imports devem ser sempre feitos em linhas separadas, nunca na mesma linha, mesmo que separados por “,”. Exemplo:

```
>>> import package1
>>> import package2
```

- Também podem ser usados da seguinte forma:

```
>>> from package1 import subpackage1, subpackage2
```

- Todos os imports devem ser sempre colocados no **topo do arquivo**, logo depois de quaisquer comentários ou *docstrings*, e antes de constantes ou globais
- Coloque uma linha em branco entre cada grupo
- Devem ser agrupados seguindo a ordem:
  1. Módulos da biblioteca padrão;
  2. Módulos terceiros relacionados entre si (por exemplo, todos os módulos de chamada de uma API proprietária, usadas na aplicação);
  3. Aplicações locais/bibliotecas específicas da aplicação.

Nomenclatura

Software Science Lab

- *"Explícito é melhor que implícito"*
- É muito importante deixar claro o que queremos desenvolver naquele bloco de código e nomear as variáveis, funções, classes, pacotes, etc, de forma clara e consistente
- *OBS:* evite utilizar letras únicas como l ou O para definir uma variável, pois pode ser facilmente confundida com 1 ou 0, por exemplo:

```
>>> O = 2
>>> # Isso pode parecer que você está tentando atribuir 2 a zero
```

Estilos de Nomenclatura

Software Science Lab

Tipo	Convenção de Nomeação	Exemplos
Função	Use uma palavra ou palavras minúsculas. Palavras separadas por sublinhados para melhorar a legibilidade.	*function, my_function*
Variável	Use uma letra, palavra ou palavras minúsculas. Palavras separadas com sublinhados para melhorar a legibilidade.	*x, var, my_variable*
Classe	Comece cada palavra com uma letra maiúscula. Não separe palavras com sublinhados.	*Model, MyClass*
Método	Use uma palavra ou palavras minúsculas. Palavras separadas com sublinhados para melhorar a legibilidade.	*class_method, method*
Constante	Use uma letra, palavra ou palavras maiúsculas. Palavras separadas com sublinhados para melhorar a legibilidade.	*CONSTANT, MY_CONSTANT, MY_LONG_CONSTANT*
Módulo	Use uma palavra ou palavras curtas e minúsculas. Palavras separadas com sublinhados para melhorar a legibilidade.	*module.py, my_module.py*
Pacote	Use uma palavra ou palavras curtas e minúsculas. Não separe palavras com sublinhados.	*package, mypackage*

Comentários

- *“Se a implementação é difícil de explicar, é uma má ideia”*
- Um programador deve documentar de forma bem objetiva e precisa seu código, pois facilita a sua manutenção e interpretabilidade
- Alguns pontos-chave sobre comentários de código:
  - Limite o comprimento da linha de comentários e docstrings em 72 caracteres;
  - Use frases completas, começando com uma letra maiúscula;
  - Certifique-se de atualizar os comentários se você alterar seu código.

Comentários

- Use **blocos de comentário** para documentar as seções de seu código, eles ajudam a entender o propósito e a funcionalidade de um determinado bloco de código.

- A PEP 8 fornece as seguintes regras para escrever comentários em blocos:

  - Bloqueie comentários de recuo para o mesmo nível do código que eles descrevem;
  - Inicie cada linha com um # seguido por um único espaço;
  - Parágrafos devem ser separados por uma linha contendo um único #.

```
>>> def quadratico (a, b, c, x):
...     # Calcular a solução para uma equação quadrática usando a
...     # Fórmula.
...     #
...     # Há sempre duas soluções para uma equação quadrática, x_1 e x_2.
...     x_1 = (- b+(b**2-4*a*c)**(11/2)) / (2*a)
...     x_2 = (- b-(b**2-4*a*c)**(1/2)) / (2*a)
...     return x_1, x_2
```

Comentários

Software Science Lab

- Comentários em linha (*inline*) explicam uma única declaração no próprio código. Eles são úteis para lembrá-lo, ou explicar aos outros, o que faz determinada linha de código. Porém, cuidados são necessários, tais como:
  - Use comentários *inline* com moderação;
  - Escreva comentários *inline* na mesma linha que a declaração a que se refere;
  - Separe os comentários *inline* por dois ou mais espaços na sua declaração;
  - Inicie os comentários *inline* com um # e um único espaço, assim como os comentários de blocos;
  - Não use comentários para explicar o óbvio.

```
>>> idade = 5  # Este é um comentário inline
>>> nome = 'Fafá de Belém'  # Nome do Estudante
```

Software Science Lab

Clean Code e Boas Práticas

Engenharia de Software para Ciência de Dados

Clean Code e Boas Práticas

- **Clean code** são instruções direcionadas ao modo como escrevemos nosso código. Não significa que se você não utilizar os métodos de clean code seu código ou programa não irá funcionar, mas significa que, provavelmente, seu código terá legibilidade baixa e consequentemente complexidade mais alta na hora que você mesmo, posteriormente, ou outro desenvolvedor, for dar manutenção neste código.

- **Boas práticas** é uma expressão que denomina técnicas identificadas como as melhores para realizar determinada tarefa.

Clean Code e Boas Práticas

Software Science Lab

Nomes são importantes

- Seja para variáveis, classes ou métodos, os nomes são muito importantes no entendimento do que o código faz, ou o que a variável guarda, utilize sempre **nomes significativos** e que sejam **pronunciáveis**.

Não use abreviações

- Abreviações fazem com que o leitor faça um esforço maior na hora de ler o código. Veja como é difícil entender este código:

```
>>> ct_n = len([1,2,3,4,5])  # ruim
>>> quantidade_de_numeros = len([1,2,3,4,5])  # bom
>>> nmdapsa = 'Josivaldo'  # horroroso
>>> nome_da_pessoa = 'Josivaldo'  # bem melhor
```

Clean Code e Boas Práticas

Software Science Lab

Utilize nomes descritivos

- Não tenha preguiça na hora de escrever seu código e não tenha medo de utilizar nomes grandes. Se for necessário para que o código seja legível, utilize-os, escrevendo tanto quanto necessário para que o nome seja fácil de entender:

```
>>> # ruim
>>> def ltTParaMl(x):
...     return x * 1000
>>> # bom
>>> def transforma_litro_para_mililitro(valor_em_lt):
...     return valor_em_lt * 1000
```

Clean Code e Boas Práticas

Software Science Lab

Evite números mágicos

- Não utilize valores constantes soltos pelo código, sem identificação do que significam. Eles não deixam claro o que o valor representa além de permitir que o valor seja reutilizado de maneira mais fácil:

```
>>> # ruim
>>> if soma > 1250:  # O que significa este número ?
...     faca_algo()
>>> # bom
>>> MINIMO_NECESSARIO_PARA_DESCONTO = 1250
>>> if soma > MINIMO_NECESSARIO_PARA_DESCONTO:
...     faca_algo()
>>> # ruim
>>> if variavel in (1,15,99,120):
...     raise Exception('Não aceito')
>>> # bom
>>> planos_nao_permitidos = (1,15,99,120)
>>> if variavel in planos_nao_permitidos:
...     raise Exception('Não aceito')
```

Clean Code e Boas Práticas

Software Science Lab

- Na hora de instanciar os objetos também é importante deixar claro o que cada um deles é:

```
>>> class Pessoa():
...     def falar(self):
...         print('Oi')
>>> # ruim
>>> p1 = Pessoa()
>>> p2 = Pessoa()
>>> # bom
>>> pai = Pessoa()
>>> filho = Pessoa()
```

- Uma boa prática também é utilizar verbos na nomeação de métodos e, substantivos nas classes e objetos.

Clean Code e Boas Práticas

Software Science Lab

Sempre pergunte o que você quer saber de forma clara

- Não faça perguntas invertidas

```
>>> #ruim
>>> def lista_usuarios():
...     if request not in
('POST','PUT','DELETE'):
...       return get_users_list()
...     else:
...       return "Method not allowed"
>>> #bom
>>> def lista_usuarios():
...     if request.method == 'GET':
...       return get_users_list()
...     return "Method not allowed"
```

- Não utilize booleanos negativos, pois isso complica o entendimento. Utilize os operadores de comparação para inverter quando necessário:

```
>>> # ruim
>>> nao_deve_processar = True
>>> if not nao_deve_processar:
...     # faz algo aqui
>>> # bom
>>> deve_processar = True
>>> # ou
>>> deve_processar = False

>>> if deve_processar:
...     # faz algo aqui
>>> # ou
>>> if not deve_processar:
...     # faz algo aqui
```

Clean Code e Boas Práticas

Software Science Lab

Utilize booleanos de forma implícita

- Eles ficam mais claros e rápidos de entender quando utilizados assim:

```
>>> # ruim
>>> def contem_letra(letra, nome):
...     if letra in nome:
...         return True
...     else:
...         return False
>>> # bom
>>> def contem_letra(letra, nome):
...     return letra in nome
>>> # ou
>>> # ruim
>>> if conta.numero in get_lista_de_contas_bloqueadas():
...     conta.bloqueada = True
>>> # bom
>>> conta.bloqueada = (conta.numero in get_lista_de_contas_bloqueadas())
```

Clean Code e Boas Práticas

Software Science Lab

Cuidado com a quantidade de parâmetros

- Se precisamos receber muitos parâmetros em um método, devemos optar por receber estruturas de dados que carregam os valores (e não uma listagem imensa de parâmetros) como Dicionários (dict), listas (list), tuplas (tuple) e instâncias de alguma classe

```
>>> # ruim
>>> def metodo_com_muitos_parametros(
...     nome, idade, data_nascimento, salario,
funcao
... ):
...     # faz algo aqui...
>>> # bom
>>> def
metodo_com_muitos_parametros(dados_do_usuario):
...     nome = dados_do_usuario.get('nome',
None)
...     # faz algo aqui...
```

Sempre que for necessário adicionar ou remover parâmetros neste método, não será necessário mudar todas as suas chamadas

Clean Code e Boas Práticas

Software Science Lab

Comentários são vilões quando não são bem utilizados

- Comentários que estão ali para explicar um código mal escrito e difícil de entender devem ser evitados. Neste caso, o código deve ser reescrito.

```
>>> # ruim
>>> # verifica se a venda entra na regra do desconto promocional
>>> if venda.valor > 120 and venda.itens > 12:
...     venda.desconto = 12
>>> # bom
>>> if venda.permite_desconto_promocional():
...     venda.aplica_desconto_promocional()
```

- Também não devemos utilizar comentários óbvios, que só atrapalham a leitura:

```
>>> def soma(a,b):
...     # retorna a soma de a com b
...     return a + b
```

Leituras Recomendadas

- **PEP-8:** https://www.python.org/dev/peps/pep-0008/
- **PEP-20:** https://www.python.org/dev/peps/pep-0020/
- Martin, Robert C. ***Código limpo: Habilidades práticas do Agile software***. Alta Books, 2019.

Vamos praticar?

- Exercícios de PEP8: 

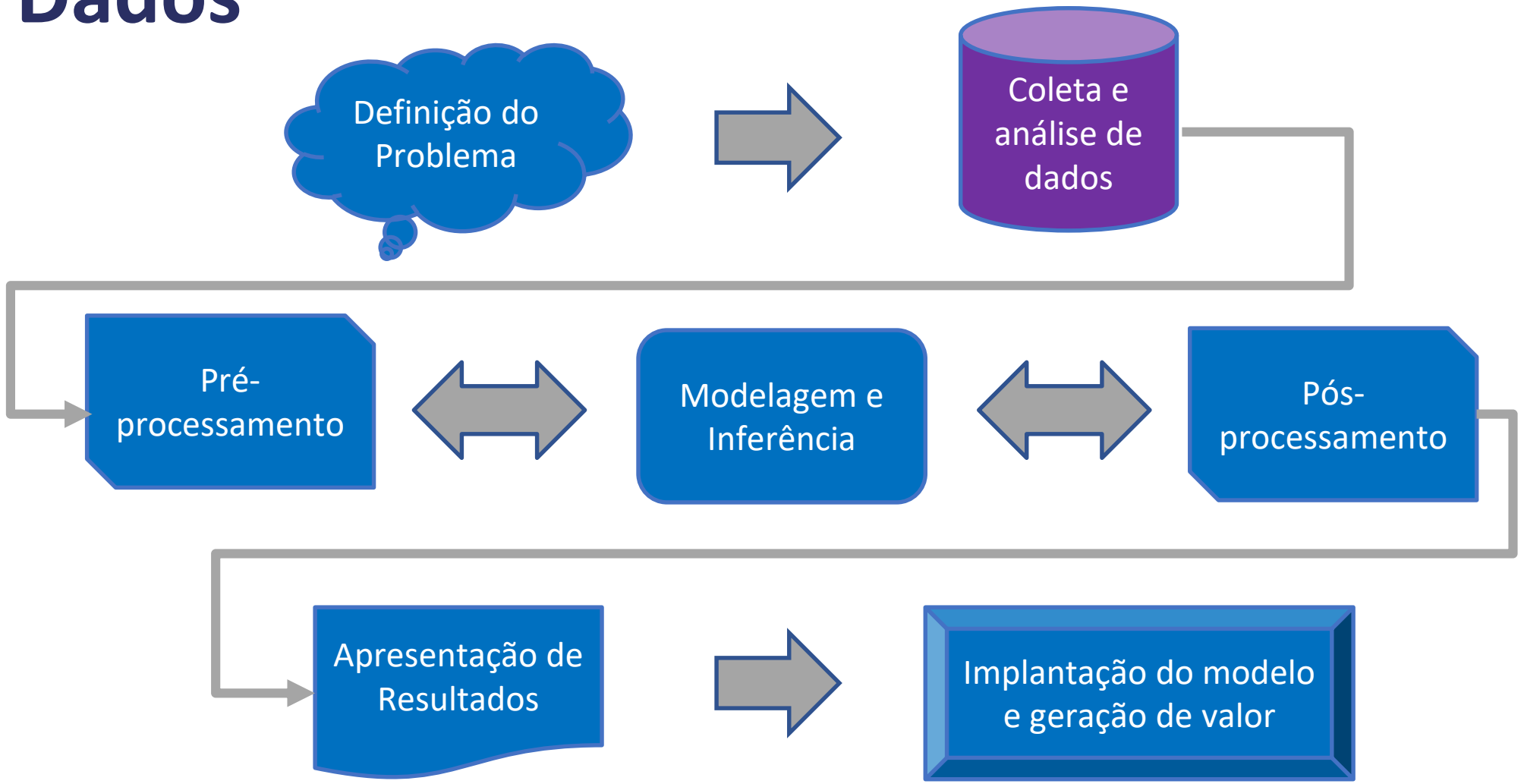
https://colab.research.google.com/drive/1RPQSFxsX8WFXsdMhiMn63vski8vCBhP3?usp=sharing

Software Science Lab

Análise Exploratória de Dados

Engenharia de Software para Ciência de Dados

Esquema Básico de um Projeto de Ciência de Dados

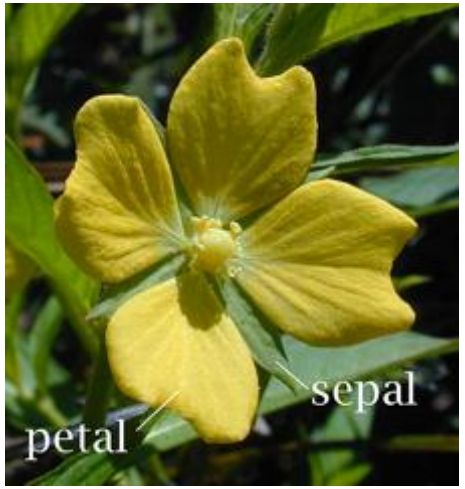


A Importância da Etapa de Análise Exploratória

- Primeiro **compreenda bem** os seus dados para depois explorar as possíveis soluções
- Após entendê-los, você poderá **limpar**, **transformar** e **apresentar** melhor os dados que você possui
- Entender os dados ajuda a obter os melhores resultados possíveis nos **algoritmos** de **Machine Learning**
- Trabalhar com **visualização de dados** (gráficos) ajuda a entender melhor seus dados e:
  - Identificar valores discrepantes, faltantes ou inválidos
  - Verificar a necessidade de alguma transformação de dados
  - Identificar atributos redundantes, entre outros

Exemplo de Análise exploratória com o Dataset Iris

- Um dos datasets mais utilizados em Machine Learning, criado em 1936.
- **150** instâncias contendo **4** variáveis numéricas (largura e comprimento da sépala e da pétala em cm) e **1** variável categórica (espécie da flor: setosa, versicolor ou virginica).



Setosa

Versicolor

Virginica

Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	*I. setosa*
4.9	3.0	1.4	0.2	*I. setosa*
4.7	3.2	1.3	0.2	*I. setosa*
4.6	3.1	1.5	0.2	*I. setosa*

Iris: Análise exploratória com Estatística Descritiva

- Ao começar a trabalhar com um novo dataset, geralmente é uma boa ideia dar uma olhada nas suas dimensões e principais medidas de estatística descritiva.
- Veja um exemplo com a linguagem R:

```
> dim(iris)
[1] 150   5
```

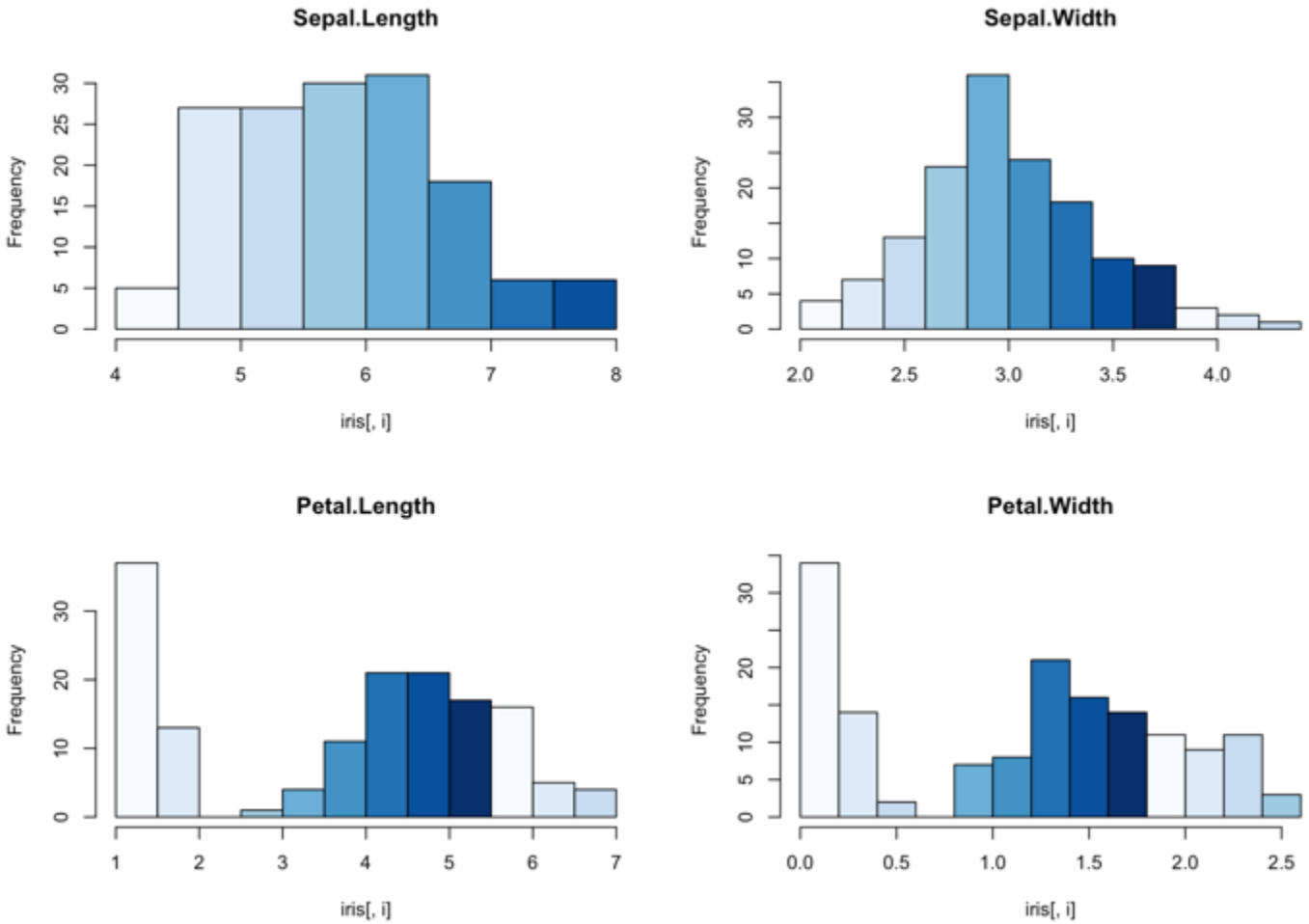
150 linhas e 5 colunas

```
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa     :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

A distribuição de classes é equilibrada

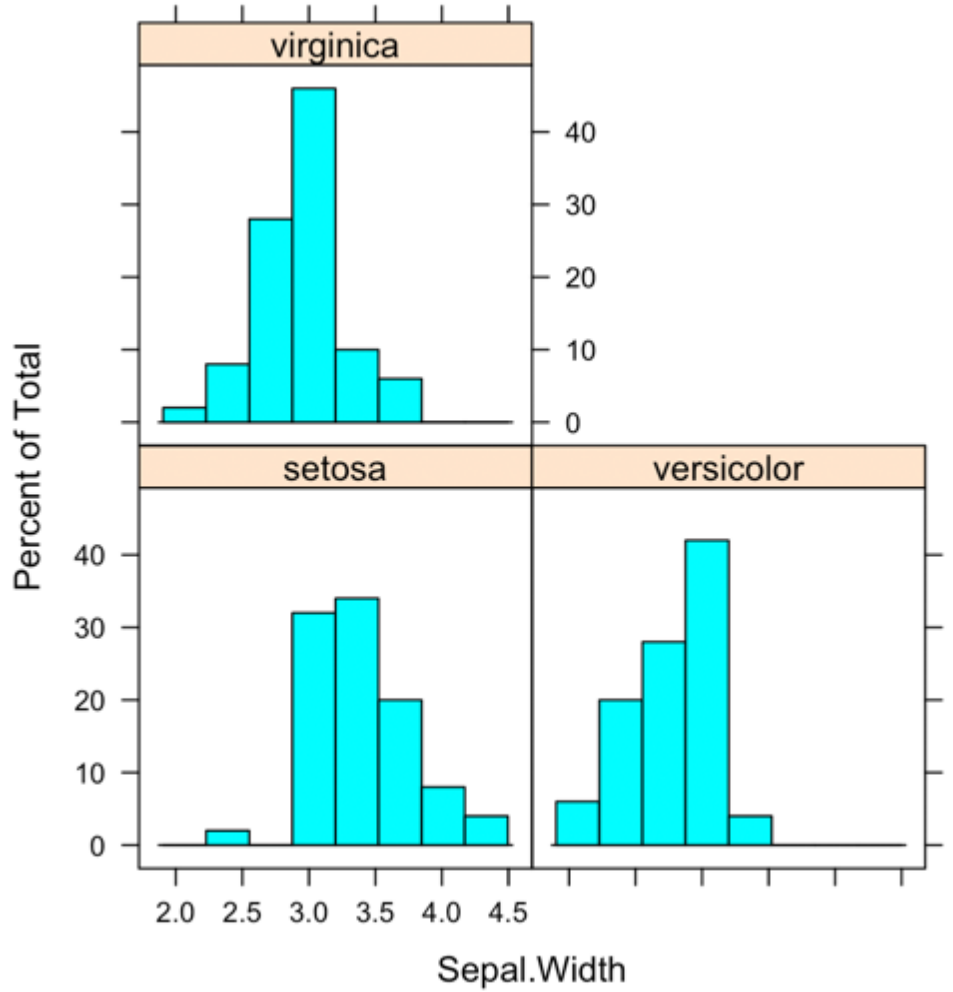
Iris: Análise Exploratória com Histograma

- O **Histograma** é um gráfico de barras contíguas, com as bases proporcionais aos intervalos de classe e a área do retângulo proporcional à respectiva frequência.
- Um histograma por atributo: podemos ter uma ideia da distribuição de cada atributo.



Iris: Análise Exploratória com Histograma

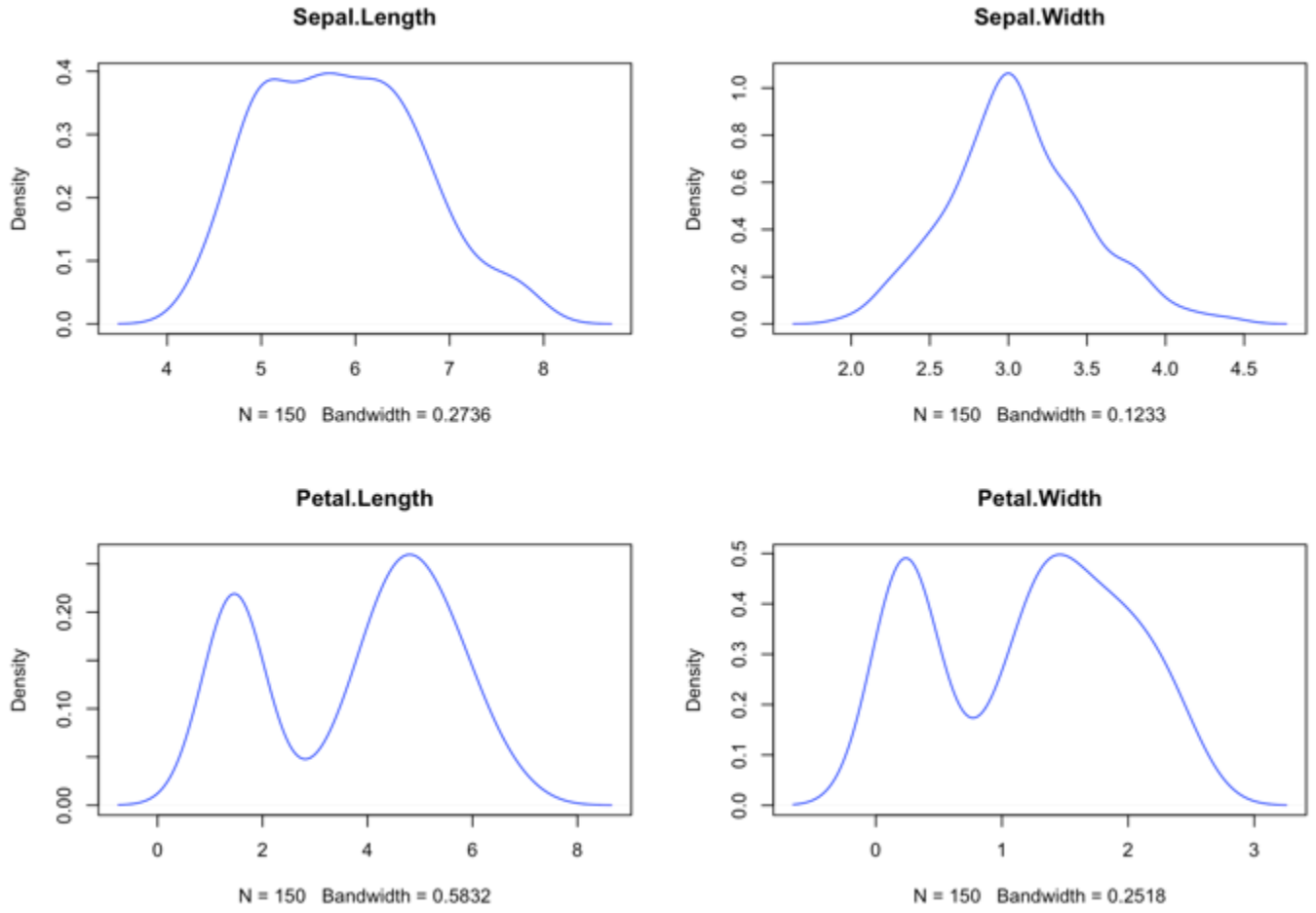
- Histogramas do atributo *Sepal.Width*, separados por classe: Podemos ter uma ideia da distribuição de cada atributo, segmentado por classe.



Iris: Análise Exploratória com Density Plot

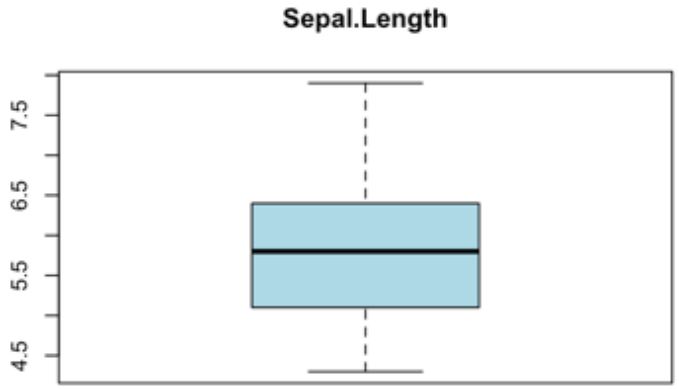
Software Science Lab

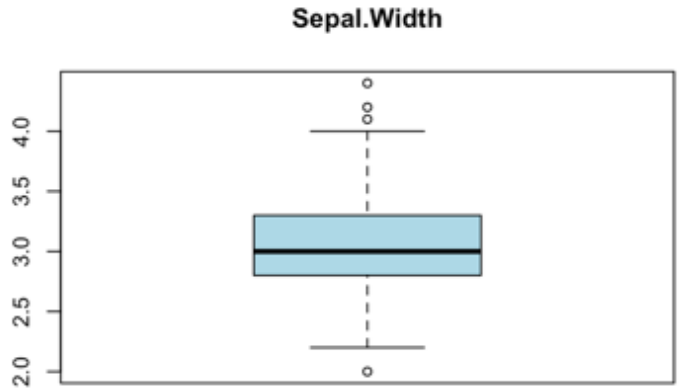
- Uma forma melhor de analisar distribuições é com o **density plot**, ou gráfico de densidade.

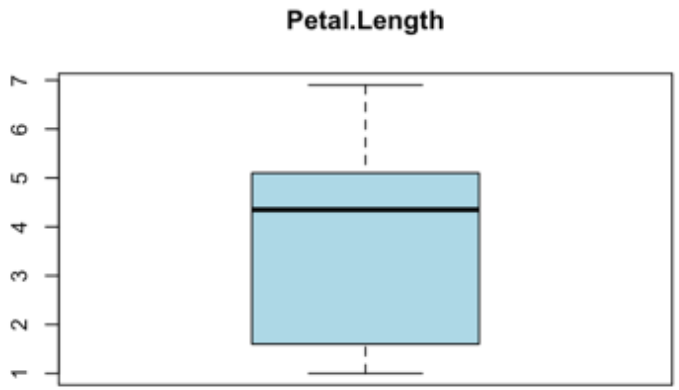


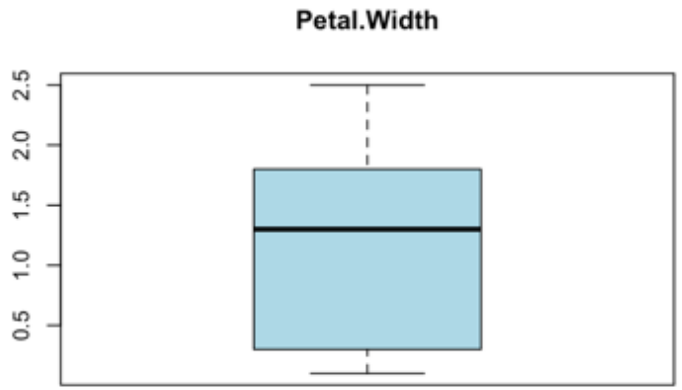
Iris: Análise Exploratória com Boxplot

- O **boxplot** pode ser usado para comparar visualmente as distribuições de uma variável numérica agrupada conforme uma variável categórica.



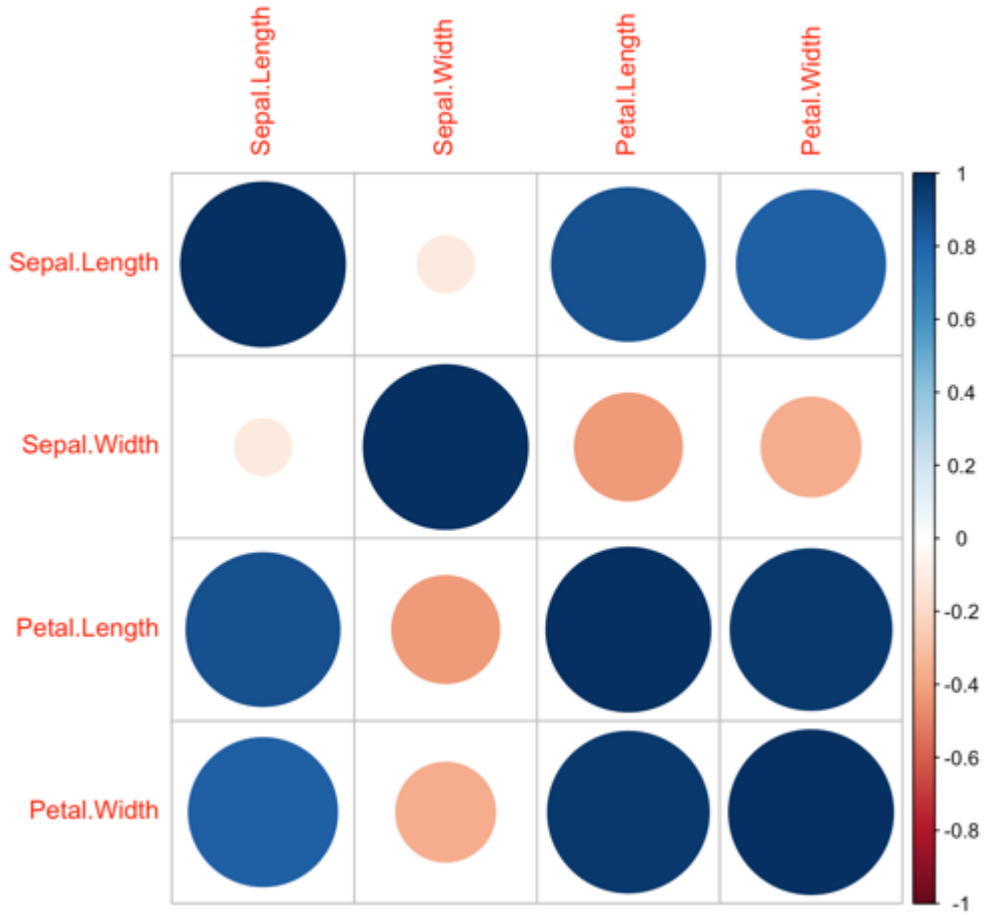






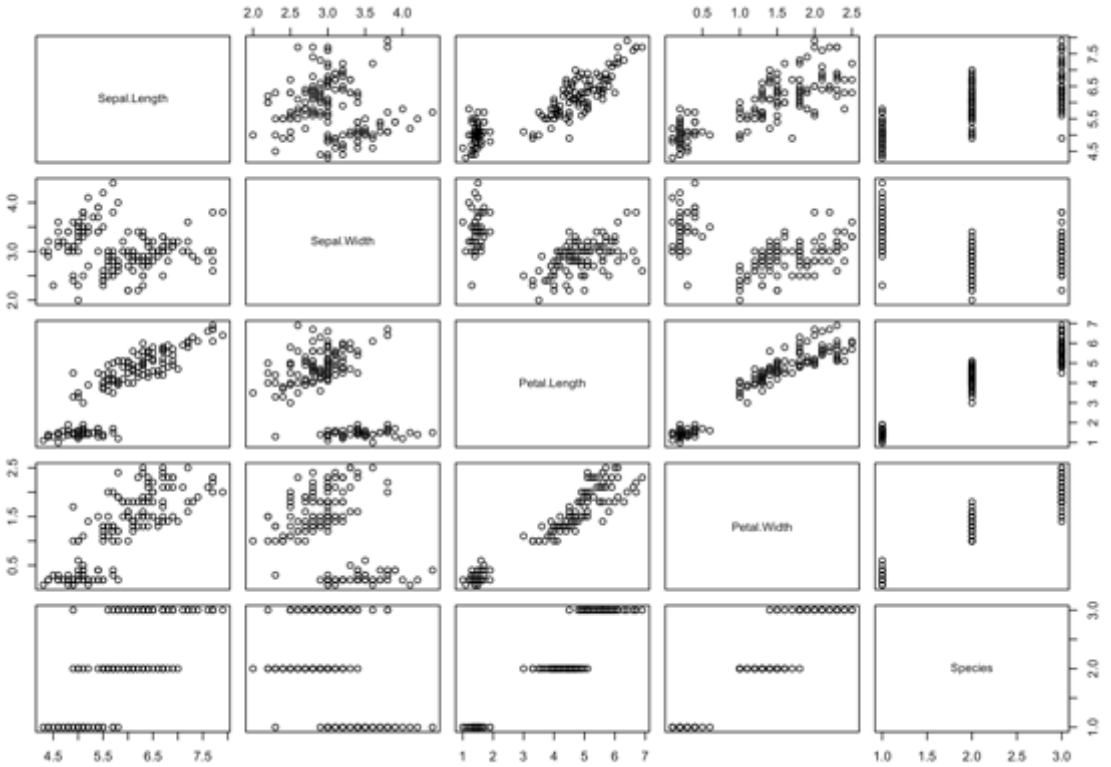
Iris: Análise Exploratória com Gráfico de Correlação

- O **gráfico de correlação** mostra a relação entre os atributos, e é formado a partir do cálculo da correlação entre cada par de atributos numéricos, que é plotada em uma matriz
  - Na figura, a cor **azul** representa correlação positiva e a cor **vermelha** correlação negativa, e quanto maior o círculo, maior a correlação
  - A matriz é simétrica e que os atributos diagonais são perfeitamente correlacionados positivamente (mostram a correlação de cada atributo consigo mesmo)
  - Nota-se que alguns dos atributos são altamente correlacionados



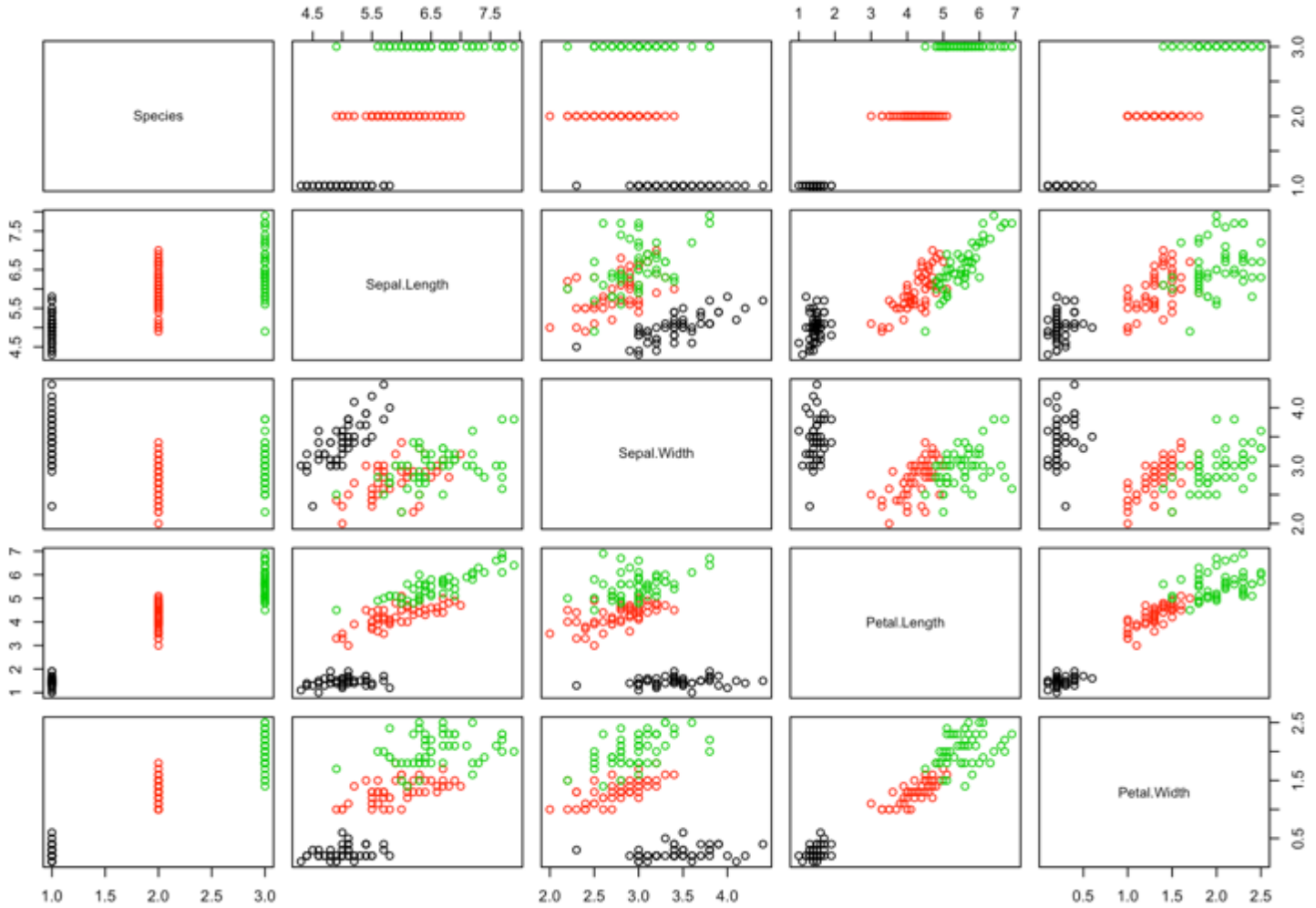
Iris: Análise Exploratória com Gráfico de Dispersão

- O **gráfico de dispersão** representa duas variáveis juntas, uma em cada um dos eixos x e y com pontos mostrando a sua interação. A propagação dos pontos indica a relação entre os atributos
- A figura ilustra, de uma só vez, um gráfico de dispersão para cada um dos pares de atributos do dataset
  - *Note que a matriz resultante é simétrica, e exibe os mesmos gráficos com os eixos invertidos, possibilitando analisar os dados de várias perspectivas*
  - *É possível notar uma relação linear, representada pela linha diagonal, entre o comprimento e a largura da pétala.*



Iris: Análise Exploratória com Gráfico de Dispersão

- Em problemas de classificação, pode ser útil **colorir** os pontos do gráfico de dispersão pela classe para ter uma ideia da facilidade da separação das classes do problema

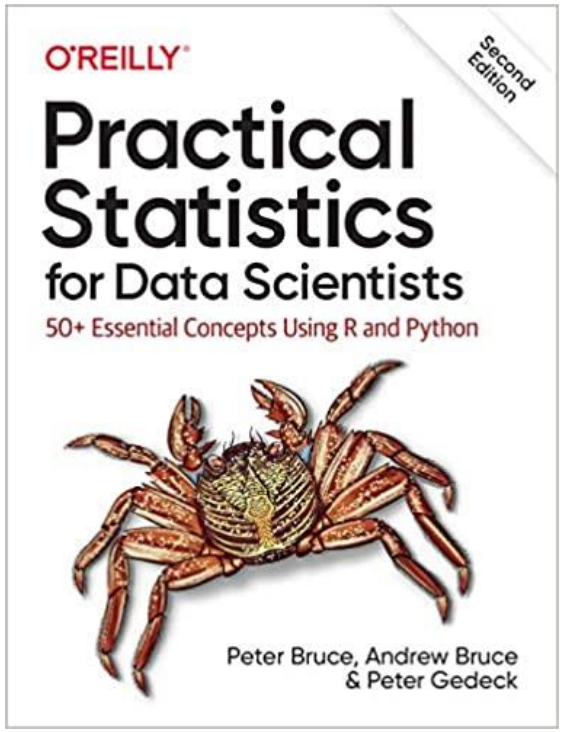


Análise Exploratória

Software Science Lab

- Os exemplos desta aula ilustram **algumas** das possíveis formas de exploração de dados, mas é importante ressaltar que eles não são exaustivos
- Não existe uma **receita pronta** para o trabalho de análise exploratória de dados, uma vez que ele dependerá dos dados a serem analisados e dos *insights* que cada etapa de análise provocará
- Também vale destacar que existem **diversos pacotes** no Python para criação de gráficos, além dos exemplificados nesta aula

Para saber mais: Estatística



http://passyworldofmathematics.com/misleading-graphs/
https://www.tylervigen.com/spurious-correlations

Python Open Data Science Stack

Software Science Lab

Módulos Python para Análise de Dados:

- **NumPy**: computação matemática (arrays)
- **SciPy**: computação científica (álgebra linear)
- **Pandas**: manipulação e análise de dados, o "excel" do Python
- **Matplotlib**: visualização de dados (gráficos)
- **Seaborn**: visualização de dados (gráficos)

Vamos praticar?

Software Science Lab

- Análise exploratória com Pandas e o dataset Iris:

https://colab.research.google.com/drive/19FMe4m37xDNZPHtz-40ZA-MMvZpMpCyt?usp=sharing

- Visualização de Dados com Matplotlib, Pandas e Seaborn e o dataset Iris:

https://colab.research.google.com/drive/1Sc3U18GOq-8_IVX5votSmH0aQgivdV8w?usp=sharing

- Análise exploratória e visualização de dados com Python e o dataset Diabetes

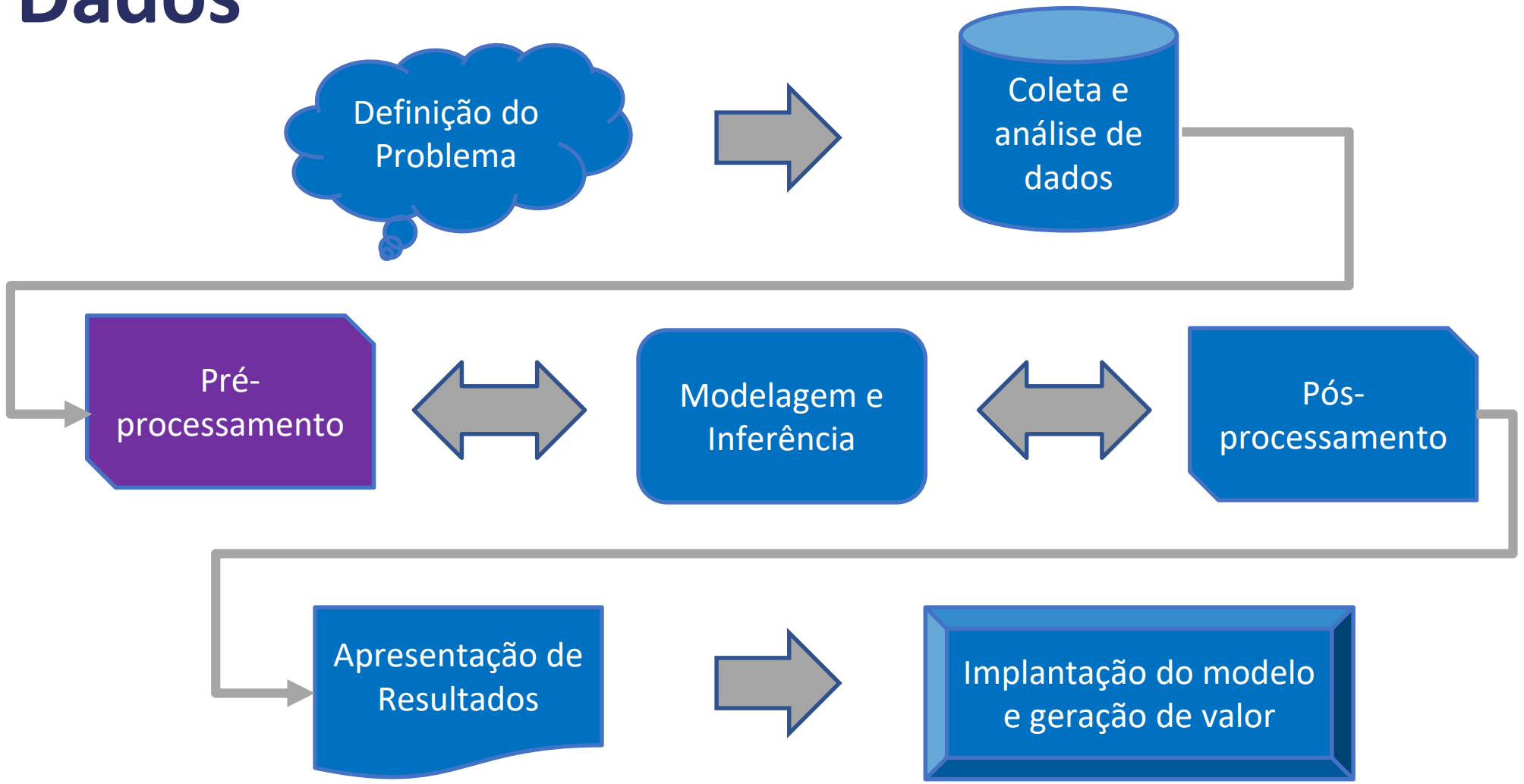
https://colab.research.google.com/drive/1yfjbPHDVq0dJOqLPq8IhcvDzRR_J-qCX?usp=sharing

Software Science Lab

Pré-Processamento de Dados

Engenharia de Software para Ciência de Dados

Esquema Básico de um Projeto de Ciência de Dados



Motivação

Software Science Lab

Após examinar com cuidado os seus dados, você pode identificar alguns problemas que devem ser tratados na etapa de **pré-processamento**, tais como:

- **Unidades de medida:** observar a unidade das variáveis (idade medida em meses ou anos, renda em dólares ou reais, etc) para fazer as devidas conversões, se necessário
- **Valores faltantes:** se um atributo tem muitos registros faltantes, ele não deve ser utilizado como entrada de um modelo sem um tratamento apropriado

Motivação

- **Valores inconsistentes:** é importante entender se os valores inconsistentes são valores **inválidos** ou **outliers**.
  - **Valor inválido:** um valor negativo em um campo que só pode ser positivo, ou um texto quando se espera um número.
  - **Outliers:** valores fora do intervalo esperado.

Motivação

- **Intervalo dos valores:** preste atenção no intervalo dos valores das variáveis.
  - Atributos como **salário** podem conter valores de 0 a mais de meio milhão de reais, por exemplo. Este grande intervalo pode ser um problema para alguns modelos, e pode ser necessário transformar esta coluna usando uma **transformação logarítmica** para reduzir o seu intervalo de valores.
  - Já variáveis com todos os valores em **intervalos pequenos** (exemplo: idades entre 50 e 55) provavelmente não serão uma informação útil para os modelos.
  - **O intervalo de valores adequado varia de acordo com o domínio da aplicação.**

Exemplo – Dataset

Software Science Lab

- Dataset simplificado do **Banco de crédito alemão**, disponível no UCI Machine Learning Repository.

	custid	sex	is.employed	income	marital.stat	health.ins	housing.type	recent.move	num.vehicles	age	state.of.res
1	2068	F	*NA*	11300	Married	TRUE	Homeowner free and clear	FALSE	2	49	Michigan
2	2073	F	*NA*	0	Married	TRUE	Rented	TRUE	3	40	Florida
3	2848	M	TRUE	4500	Never Married	FALSE	Rented	TRUE	3	22	Georgia
4	5641	M	TRUE	20000	Never Married	FALSE	Occupied with no rent	FALSE	0	22	New Mexico
5	6369	F	TRUE	12000	Never Married	TRUE	Rented	TRUE	1	31	Florida
6	8322	F	TRUE	180000	Never Married	TRUE	Homeowner with mortgage/loan	FALSE	1	40	New York
7	8521	M	TRUE	120000	Never Married	TRUE	Homeowner free and clear	TRUE	1	39	Idaho

Exemplo – Sumário dos Dados

- Vamos examinar os dados para verificar a necessidade de operações de pré-processamento

```
> # carrega os dados do banco de crédito alemão
> dados4 <- read.table('custdata.tsv',header = T,sep = '\t')
> summary(dados4)
     custid          sex     is.employed         income                   marital.stat health.ins                            housing.type
 Min.   :   2068   F:440   Mode :logical   Min.   : -8700   Divorced/Separated:155   Mode :logical   Homeowner free and clear    :157
 1st Qu.: 345667   M:560   FALSE:73        1st Qu.: 14600   Married           :516   FALSE:159       Homeowner with mortgage/loan:412
 Median : 693403           TRUE :599       Median : 35000   Never Married     :233   TRUE :841       Occupied with no rent       : 11
 Mean   : 698500           NA's :328       Mean   : 53505   Widowed           : 96                   Rented                      :364
 3rd Qu.:1044606                           3rd Qu.: 67000                                            NA's                        : 56
 Max.   :1414286                           Max.   :615000

 recent.move      num.vehicles        age             state.of.res
 Mode :logical   Min.   :0.000   Min.   :  0.0   California  :100
 FALSE:820       1st Qu.:1.000   1st Qu.: 38.0   New York    : 71
 TRUE :124       Median :2.000   Median : 50.0   Pennsylvania: 70
 NA's :56        Mean   :1.916   Mean   : 51.7   Texas       : 56
                 3rd Qu.:2.000   3rd Qu.: 64.0   Michigan    : 52
                 Max.   :6.000   Max.   :146.7   Ohio        : 51
                 NA's   :56                      (Other)     :600
```

Exemplo – Sumário dos Dados

- Vamos inicialmente examinar os dados para verificar potenciais problemas.

```
> # carrega os dados do banco de crédito alemão
> dados4 <- read.table('custdata.tsv',header = T,sep = '\t')
> summary(dados4)
     custid          sex         is.employed        income                   marital.stat health.ins
 Min.   :   2068   F:440   Mode :logical   Min.   : -8700   Divorced/Separated:155   Mode :logical
 1st Qu.: 345667   M:560   FALSE:73        1st Qu.: 14600   Married           :516   FALSE:159
 Median : 693403           TRUE :599       Median : 35000   Never Married     :233   TRUE :841
 Mean   : 698500           NA's :328       Mean   : 53505   Widowed           : 96
 3rd Qu.:1044606                           3r
 Max.   :1414286                           Max.

                    housing.type
 Homeowner free and clear    :157
 Homeowner with mortgage/loan:412
 Occupied with no rent       : 11
 Rented                      :364
 NA's                        : 56

 recent.move      num.vehicles        age
 Mode :logical   Min.   :0.000   Min.   :  0.0   Ca
 FALSE:820       1st Qu.:1.000   1st Qu.: 38.0   New York     : 71
 TRUE :124       Median :2.000   Median : 50.0   Pennsylvania: 70
 NA's :56        Mean   :1.916   Mean   : 51.7   Texas        : 56
                 3rd Qu.:2.000   3rd Qu.: 64.0   Michigan     : 52
                 Max.   :6.000   Max.   :146.7   Ohio         : 51
                 NA's   :56                      (Other)      :600
```

1/3 dos dados faltantes

Possíveis soluções:

- Excluir esta coluna
- Substituir os valores pelas categorias sim/não/desconhecido

Exemplo – Sumário dos Dados

Software Science Lab

```
> # carrega os dados do banco de crédito alemão
> dados4 <- read.table('custdata.tsv',header = T,sep = '\t')
> summary(dados4)
     custid          sex     is.employed        income                marital.stat health.ins
 Min.   :   2068   F:440   Mode :logical   Min.   : -8700   Divorced/Separated:155   Mode :logical
 1st Qu.: 345667   M:560   FALSE:73        1st Qu.: 14600   Married            :516   FALSE:159
 Median : 693403           TRUE :599       Median : 35000   Never Married      :233   TRUE :841
 Mean   : 698500           NA's :328       Mean   : 53505   Widowed            : 96
 3rd Qu.:1044606                           3rd Qu.: 67000
 Max.   :1414286                           Max.   :615000

                                housing.type
 Homeowner free and clear    :157
 Homeowner with mortgage/loan:412
 Occupied with no rent       : 11
 Rented                      :364
 NA's                        : 56

 recent.move      num.vehicles        age              state.of.res
 Mode :logical   Min.   :0.000   Min.   :  0.0   California   :100
 FALSE:820       1st Qu.:1.000   1st Qu.: 38.0   New York     : 71
 TRUE :124       Median :2.000   Median : 50.0   Pennsylvania: 70
 NA's :56        Mean   :1.916   Mean   : 51.7   Texas        : 56
                 3rd Qu.:2.000   3rd Qu.: 64.0   Michigan     : 52
                 Max.   :6.000   Max.   :146.7   Ohio         : 51
                 NA's   :56                      (Other)      :600
```

Valores de salário negativos, potencialmente inválidos: erro na entrada de dados ou cliente tem um débito?

Possíveis soluções:

- Excluir as linhas
- Substituí-los por 0

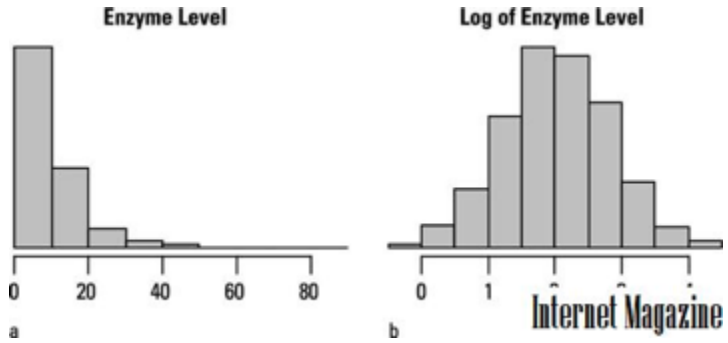
Exemplo – Sumário dos Dados

```
> # carrega os dados do banco de crédito alemão
> dados4 <- read.table('custdata.tsv',header = T,sep = '\t')
> summary(dados4)
     custid          sex     is.employed       income                marital.stat health.ins
 Min.   :   2068   F:440   Mode :logical   Min.   : -8700   Divorced/Separated:155   Mode :logical
 1st Qu.: 345667   M:560   FALSE:73        1st Qu.: 14600   Married           :516   FALSE:159
 Median : 693403           TRUE :599       Median : 35000   Never M
 Mean   : 698500           NA's :328       Mean   : 53505   Wid
 3rd Qu.:1044606                           3rd Qu.: 67000
 Max.   :1414286                           Max.   :615000

                    housing.type
 Homeowner free and clear    :157
 Homeowner with mortgage/loan:412
 Occupied with no rent       : 11
 Rented                      :364
 NA's                        : 56

 recent.move      num.vehicles        age              state.of.res
 Mode :logical   Min.   :0.000   Min.   :  0.0   California   :100
 FALSE:820       1st Qu.:1.000   1st Qu.: 38.0   New York     : 71
 TRUE :124       Median :2.000   Median : 50.0   Pennsylvania: 70
 NA's :56        Mean   :1.916   Mean   : 51.7   Texas        : 56
                 3rd Qu.:2.000   3rd Qu.: 64.0   Michigan     : 52
                 Max.   :6.000   Max.   :146.7   Ohio         : 51
                 NA's   :56                      (Other)      :600
```

O range da variável *income* é muito grande, o que pode afetar alguns modelos de ML

Possível solução:

- Transformação logarítmica



Exemplo – Sumário dos Dados

Software Science Lab

```
> # carrega os dados do banco de crédito alemão
> dados4 <- read.table('custdata.tsv',header = T,sep = '\t')
> summary(dados4)
     custid          sex         is.employed         income                    marital.stat health.ins
 Min.   :   2068   F:440   Mode :logical   Min.   : -8700   Divorced/Separated:155   Mode :logical
 1st Qu.: 345667   M:560   FALSE:73        1st Qu.: 14600   Married           :516   FALSE:159
 Median : 693403           TRUE :599       Median : 35000   Never Married     :233   TRUE :841
 Mean   : 698500           NA's :328       Mean   : 53505   Widowed           : 96
 3rd Qu.:1044606                           3rd Qu.: 67000
 Max.   :1414286                           Max.   :615000

                        housing.type
 Homeowner free and clear    :157
 Homeowner with mortgage/loan:412
 Occupied with no rent       : 11
 Rented                      :364
 NA's                        : 56

 recent.move      num.vehicles        age              state.of.res
 Mode :logical   Min.   :0.000   Min.   :  0.0   California   :100
 FALSE:820       1st Qu.:1.000   1st Qu.: 38.0   New York     : 71
 TRUE :124       Median :2.000   Median : 50.0   Pennsylvania: 70
 NA's :56        Mean   :1.916   [illegible]
                 3rd Qu.:2.000
                 Max.   :6.000   [illegible]     Ohio         : 51
                 NA's   :56                      (Other)      :600
```

56 valores faltantes (missings)

Possíveis soluções:

- Excluir as 56 linhas
- Substituí-los pela média dos valores

Exemplo – Sumário dos Dados

Software Science Lab

```
> # carrega os dados do banco de crédito alemão
> dados4 <- read.table('custdata.tsv',header = T,sep = '\t')
> summary(dados4)
     custid          sex         is.employed        income                marital.stat health.ins
 Min.   :   2068   F:440   Mode :logical   Min.   : -8700   Divorced/Separated:155   Mode :logical
 1st Qu.: 345667   M:560   FALSE:73        1st Qu.: 14600   Married           :516   FALSE:159
 Median : 693403           TRUE :599       Median : 35000   Never Married
 Mean   : 698500           NA's :328       Mean   : 53505   Widowed
 3rd Qu.:1044606                           3rd Qu.: 67000
 Max.   :1414286                           Max.   :615000

                                                                housing.type
 Homeowner free and clear    :157
 Homeowner with mortgage/loan:412
                                pied with no rent  : 11
                                                   :364

 recent.move      num.vehicles        age
 Mode :logical   Min.   :0.000   Min.   :  0.0
 FALSE:820       1st Qu.:1.000   1st Qu.: 38.0
 TRUE :124       Median :2.000   Median : 50.0
 NA's :56        Mean   :1.916   Mean   : 51.7
                 3rd Qu.:2.000   3rd Qu.: 64.0
                 Max.   :6.000   Max.   :146.7
                 NA's   :56

                 California  :100
                 New York    : 71
                 Pennsylvania: 70
                 Texas       : 56
                 Michigan    : 52
                 Ohio        : 51
                 (Other)     :600
```

Apesar da média de idade parecer razoável, os valores de mínimo e máximo são estranhos. Podem ser outliers, mas 0 pode significar idade desconhecida ou não informada.

Possíveis soluções:

- Normalizar as idades pela média se a idade absoluta não for importante
- Converter em categorias (bins)

Técnicas de Pré-Processamento de Dados

- Limpeza
- Agregação
- Amostragem
- Redução de dimensionalidade
- Seleção de características
- Criação de recursos
- Transformação
- Enriquecimento

IMPORTANTE: Recomenda-se fazer uma **cópia** dos dados fontes originais e organizá-los em um único conjunto tratado.

Particionamento do Conjunto de Dados

Software Science Lab

- Após construído, um modelo precisa ser **validado** com dados diferentes dos utilizados na sua construção.
- Devem ser utilizados pelo menos dois conjuntos de dados: um de **treino** (para construir o modelo) e um de **teste** (para avaliar o modelo construído).
- Métodos de partição de dados:
  - **Holdout:** p% para treino e (1-p)% para teste, considerando geralmente p > ½.

Holdout

Cliente	Ano	Mês	Dia	Atributos	Pagou?
João	2012	Janeiro	1	...	Sim
Oswaldo	2012	Janeiro	9	...	Sim
Maria	2012	Janeiro	21	...	Não
Roberta	2012	Fevereiro	4	...	Sim
...	...	...	...	...	...
Thiego	2013	Novembro	29	...	Sim
Dalva	2013	Dezembro	2	...	Não
Josy	2013	Dezembro	11	...	Não
Érica	2014	Janeiro	11	...	Sim
Renata	2014	Janeiro	19	...	Sim
...	...	...	...	...	...
Dorival	2014	Dezembro	20	...	Sim

Treinamento

Teste

Validação Cruzada

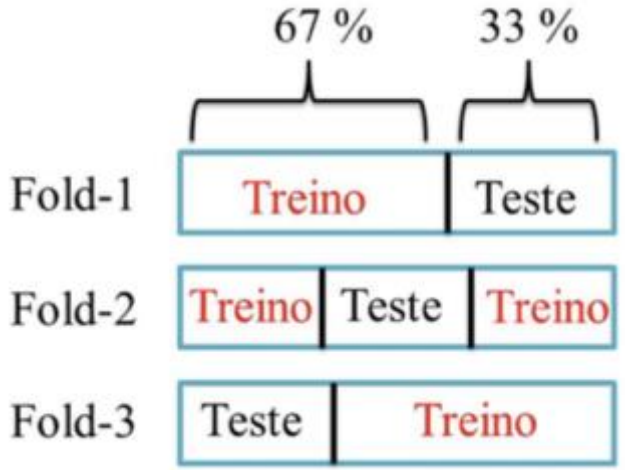
- **Validação cruzada com K conjuntos (K-Fold CrossValidation):** dividir aleatoriamente o conjunto de dados com *N* elementos em *K* subconjuntos disjuntos (folds) com aproximadamente o mesmo número de elementos (*N/K*). Cada um dos *K* subconjuntos é usado como conjunto de teste e os restantes são reunidos em um conjunto de treino. O processo é repetido *K* vezes, sendo gerados e avaliados *K* modelos de conhecimento. Valores mais comuns: 3, 5 e 10.
- **Validação cruzada com K conjuntos estratificada (Stratified K-Fold CrossValidation):** similar ao anterior, mas ao gerar os subconjuntos, a proporção de exemplos em cada uma das classes é considerada durante a amostragem.
- **Leave-One-Out:** validação cruzada com *K*= *N*. Muito custoso.

Exemplo: Validação Cruzada 3-fold

Índice	Nome	...
1	Adriano	...
2	Mateus	...
3	Thiago	...
4	João	...
5	Pedro	...
6	Lucas	...
7	Paulo	...
8	Timóteo	...
9	Maria	...
10	Martha	...
11	Madalena	...
12	Ester	...

1º Passo: Embaralhar os Índices

Índice	Nome	...
9	Maria	...
5	Pedro	...
6	Lucas	...
8	Timóteo	...
12	Ester	...
1	Adriano	...
10	Martha	...
2	Mateus	...
7	Paulo	...
11	Madalena	...
3	Thiago	...
4	João	...



Pasta	Acurácia	Exemplo
1	$AccPasta_1$	0,65
2	$AccPasta_2$	0,70
3	$AccPasta_3$	0,75
Final	$AccTeste$	$\frac{(0,65+0,70+0,75)}{3} = 0,70$

Exemplo: Validação Cruzada 3-fold

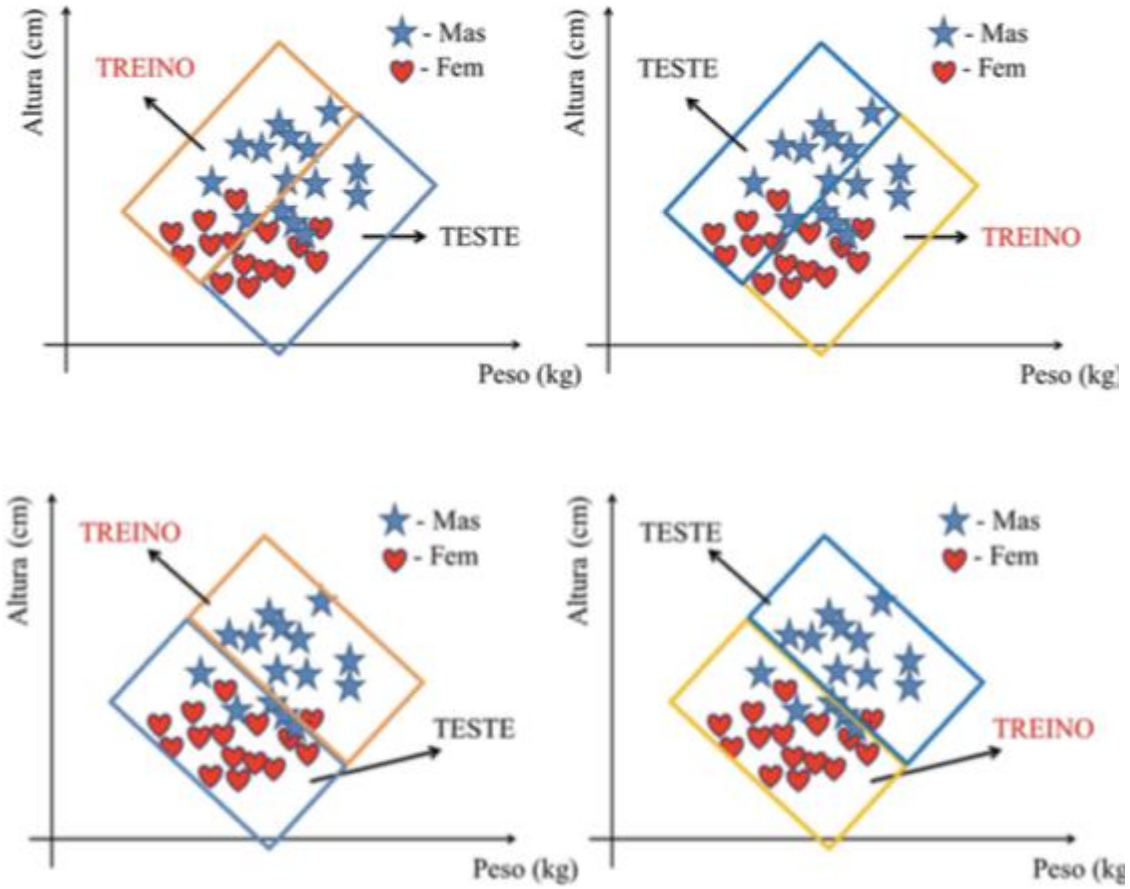
Software Science Lab

Índice	Nome	...	Índice*	Pasta 1
9	Maria	...	1	Treino
5	Pedro	...	2	Treino
6	Lucas	...	3	Treino
8	Timóteo	...	4	Treino
12	Ester	...	5	Treino
1	Adriano	..	6	Treino
10	Martha	..	7	Treino
2	Mateus	...	8	Treino
7	Paulo	...	9	Teste
11	Madalena	...	10	Teste
3	Thiago	...	11	Teste
4	João	...	12	Teste

Índice	Nome	...	Índice*	Pasta 2
9	Maria	...	1	Treino
5	Pedro	...	2	Treino
6	Lucas	...	3	Treino
8	Timóteo	...	4	Treino
12	Ester	...	5	Teste
1	Adriano	..	6	Teste
10	Martha	..	7	Teste
2	Mateus	...	8	Teste
7	Paulo	...	9	Treino
11	Madalena	...	10	Treino
3	Thiago	...	11	Treino
4	João	...	12	Treino

Índice	Nome	...	Índice*	Pasta 3
9	Maria	...	1	Teste
5	Pedro	...	2	Teste
6	Lucas	...	3	Teste
8	Timóteo	...	4	Teste
12	Ester	...	5	Treino
1	Adriano	..	6	Treino
10	Martha	..	7	Treino
2	Mateus	...	8	Treino
7	Paulo	...	9	Treino
11	Madalena	...	10	Treino
3	Thiago	...	11	Treino
4	João	...	12	Treino

Problema da Validação Cruzada Tradicional



Conjuntos de Treino e Teste desequilibrados!

Exemplo: Validação Cruzada 3-fold Estratificada

Índice	Nome	Peso (kg)	Altura (cm)	Sexo
1	Adriano	80	185	M
2	Mateus	67	175	M
3	Thiago	58	187	M
4	João	82	176	M
5	Pedro	74	168	M
6	Lucas	78	174	M
7	Paulo	71	172	M
8	Timóteo	92	179	M
9	Maria	54	167	F
10	Martha	58	170	F
11	Madalena	62	171	F
12	Ester	57	164	F

8 Homens = 67% da Base de Dados

4 Mulheres = 33% da Base de Dados

Índice	Nome	...
1	Adriano	...
2	Mateus	...
3	Thiago	...
4	João	...
5	Pedro	...
6	Lucas	...
7	Paulo	...
8	Timóteo	...
9	Maria	...
10	Martha	...
11	Madalena	...
12	Ester	...

1º Passo: Embaralhar os Índices de Forma Estratificada

Índice	Nome	...
5	Pedro	...
4	João	...
8	Timóteo	...
3	Thiago	...
6	Lucas	...
7	Paulo	...
1	Adriano	...
2	Mateus	...
11	Madalena	...
9	Maria	...
12	Ester	...
10	Martha	...

Exemplo: Validação Cruzada 3-fold Estratificada

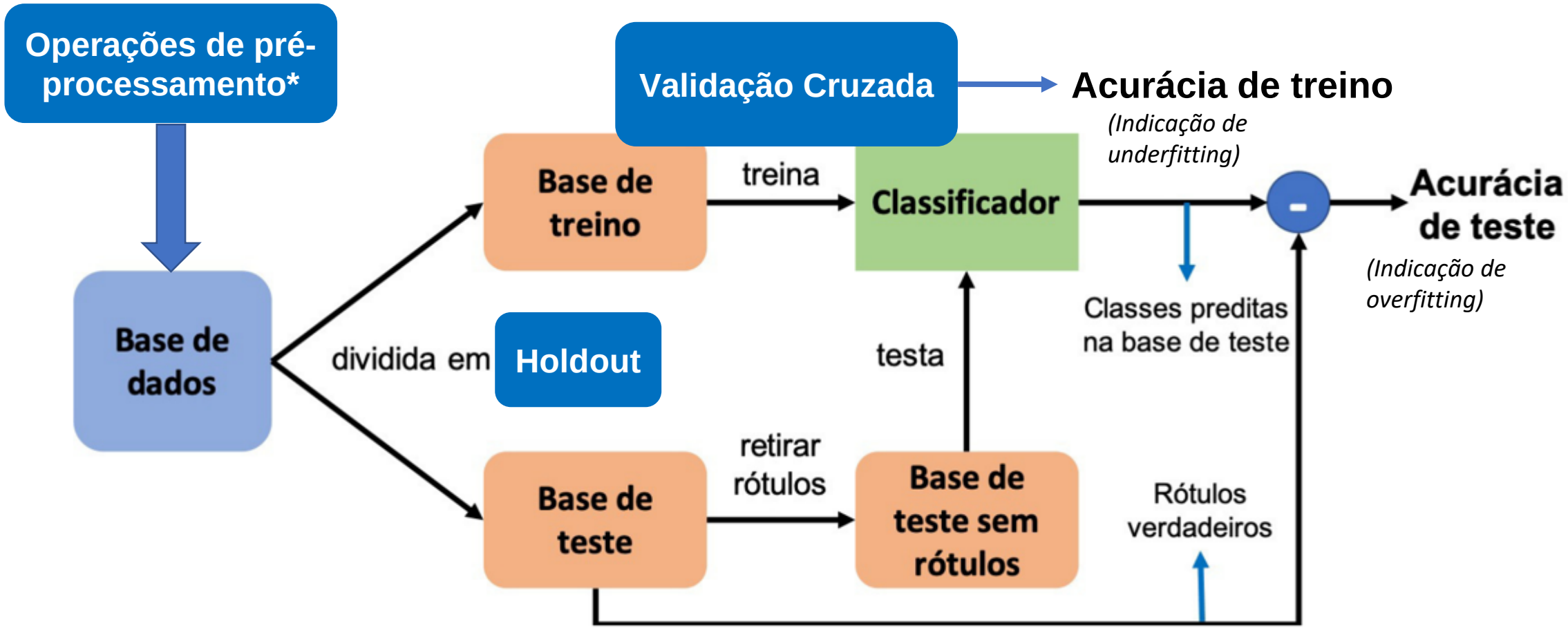
Software Science Lab

Índice	Nome	...	Índice*	Pasta 1
5	Pedro	...	1	Treino
4	João	...	2	Treino
8	Timóteo	...	3	Treino
3	Thiago	...	4	Treino
6	Lucas	...	5	Treino
7	Paulo	..	6	Teste
1	Adriano	..	7	Teste
2	Mateus	...	8	Teste
11	Madalena	...	9	Treino
9	Maria	...	10	Treino
12	Ester	...	11	Treino
10	Martha	...	12	Teste

Índice	Nome	...	Índice*	Pasta 2
5	Pedro	...	1	Treino
4	João	...	2	Treino
8	Timóteo	...	3	Teste
3	Thiago	...	4	Teste
6	Lucas	...	5	Teste
7	Paulo	..	6	Treino
1	Adriano	..	7	Treino
2	Mateus	...	8	Treino
11	Madalena	...	9	Treino
9	Maria	...	10	Treino
12	Ester	...	11	Teste
10	Martha	...	12	Treino

Índice	Nome	...	Índice*	Pasta 3
5	Pedro	...	1	Teste
4	João	...	2	Teste
8	Timóteo	...	3	Treino
3	Thiago	...	4	Treino
6	Lucas	...	5	Treino
7	Paulo	..	6	Treino
1	Adriano	..	7	Treino
2	Mateus	...	8	Treino
11	Madalena	...	9	Teste
9	Maria	...	10	Teste
12	Ester	...	11	Treino
10	Martha	...	12	Treino

Problema de classificação: Pipeline Simplificado



Vamos praticar?

- Pré-processamento de dados:

https://colab.research.google.com/drive/1BTORgj7Ul2pLMI1lU-BxbBOo_UqO7wXM?usp=sharing

- Pré-processamento de dados com o datasets Diabetes e Breast Cancer:

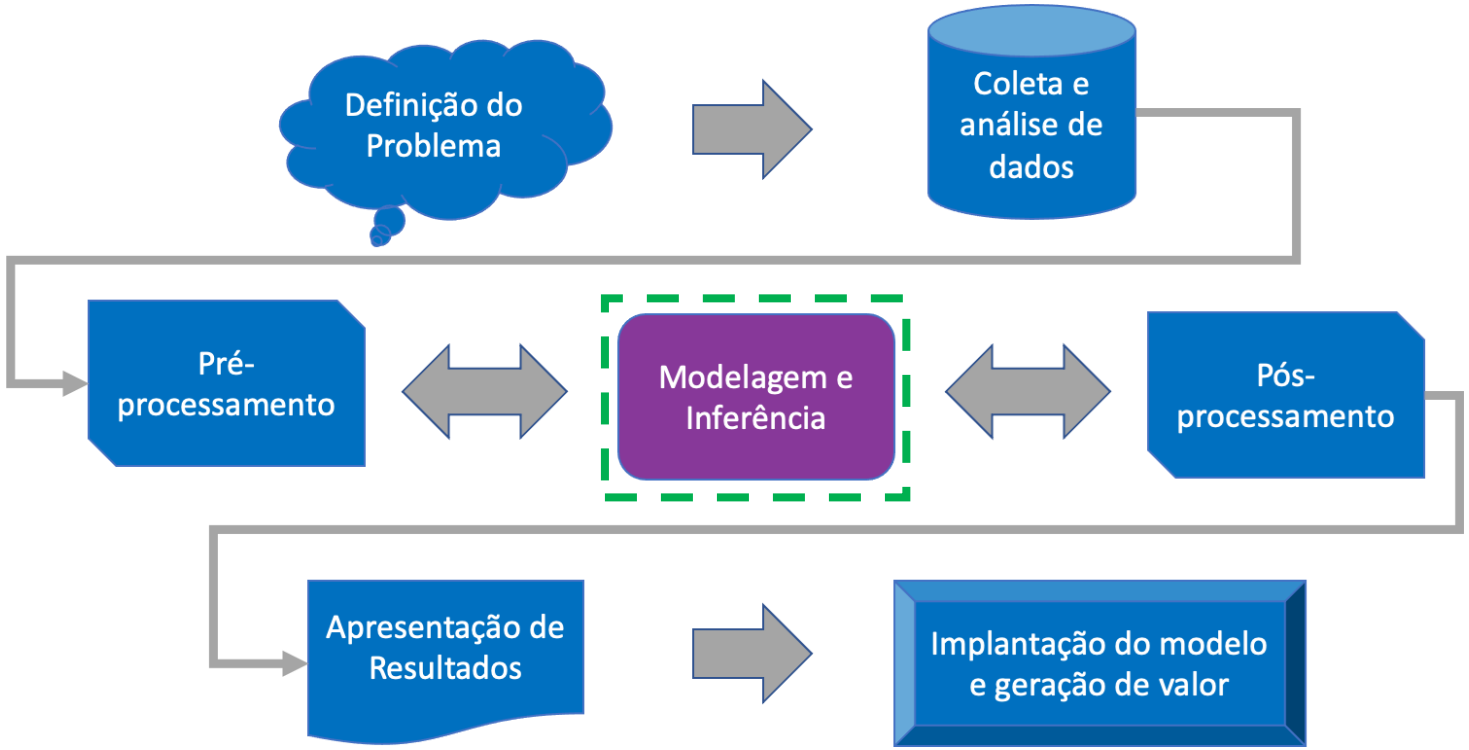
https://colab.research.google.com/drive/174mD9VGa311FK9yxkDpZs2AiM3KUUFbU?usp=sharing

Software Science Lab

Machine Learning – Problemas de Classificação

Engenharia de Software para Ciência de Dados

Esquema Básico de um projeto de CD



- 1 **Elencar** os modelos possíveis e passíveis para cada tipo de problema
- 2 **Estimar** os parâmetros que compõem os modelos, baseando-se nas instâncias e variáveis pré-processadas
- 3 **Avaliar** os resultados de cada modelo, usando métricas e um processo justo de comparação

Mas o que é um modelo (Fawcett, 2013)?

- Uma representação **simplificada** da realidade criada para servir a um propósito
- Baseado em **pressupostos** sobre o que é e o que não é importante para a finalidade específica ou com base nas **limitações** de informações ou tratabilidade

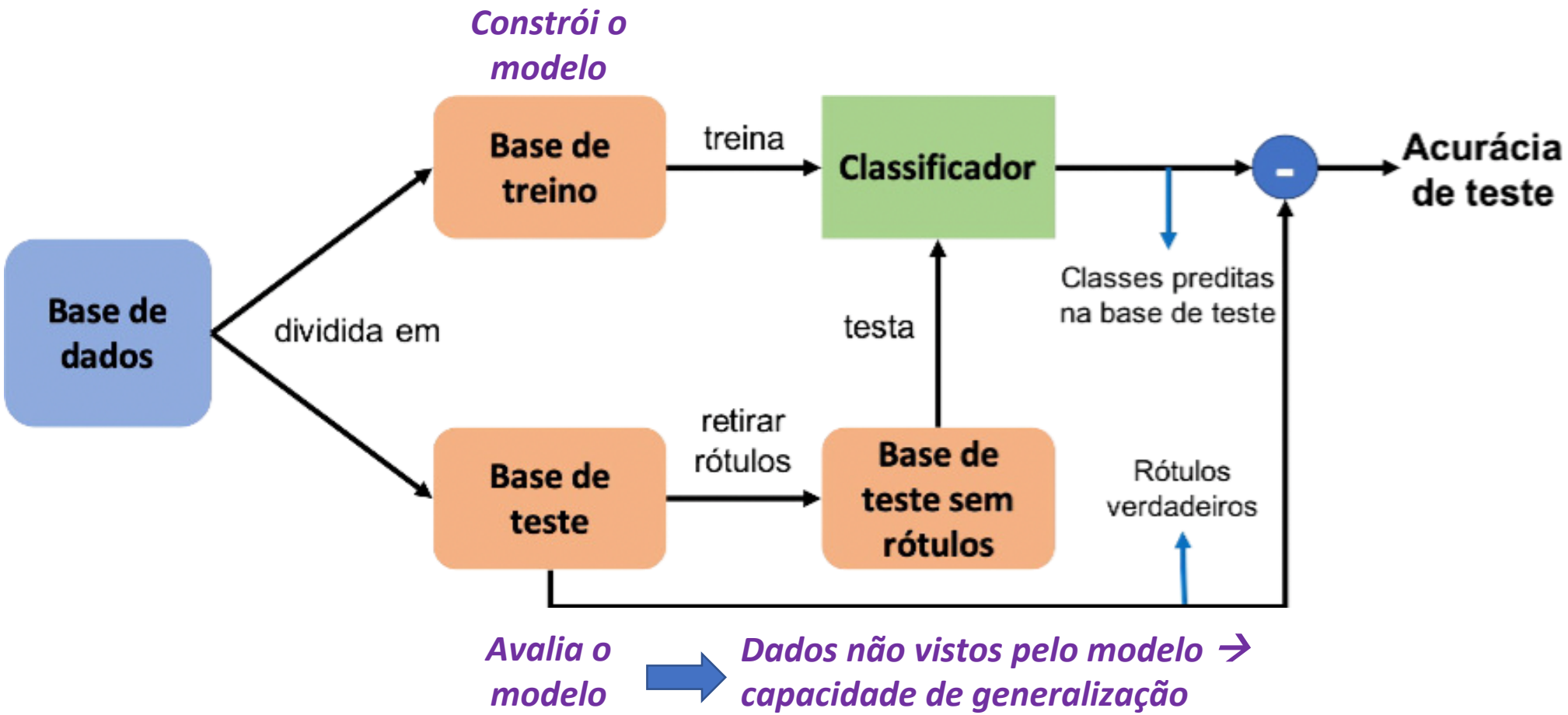
- Uma **fórmula** para estimar o valor desconhecido de interesse (target)
- A fórmula pode ser **matemática**, uma declaração lógica baseada em **regras** ou um **híbrido** dos dois
- A criação de modelos a partir de dados é conhecida como **indução** (generalização a partir de casos específicos)

[FAWCETT, 2013] "Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking".

Classificação

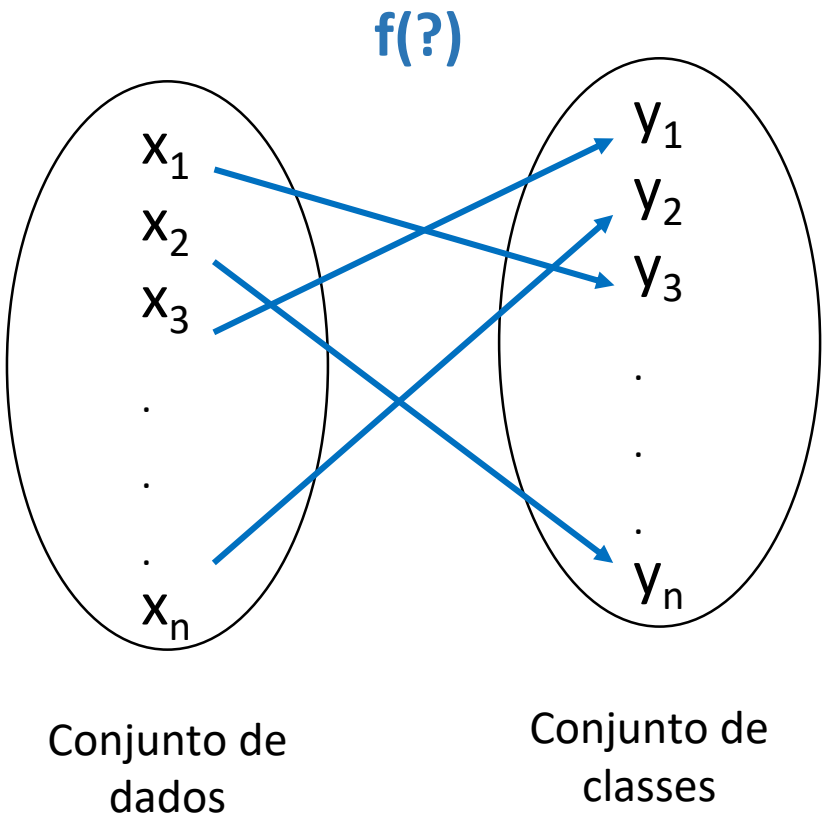
- Uma das tarefas de ML mais importantes e mais populares
- Utiliza aprendizado **supervisionado**
- Utiliza dois grupos:
  - **X**, com os atributos a serem utilizados na predição do valor
  - **Y**, com o atributo para o qual se deve fazer a predição do valor (atributo-alvo)
- O atributo alvo é **categórico**

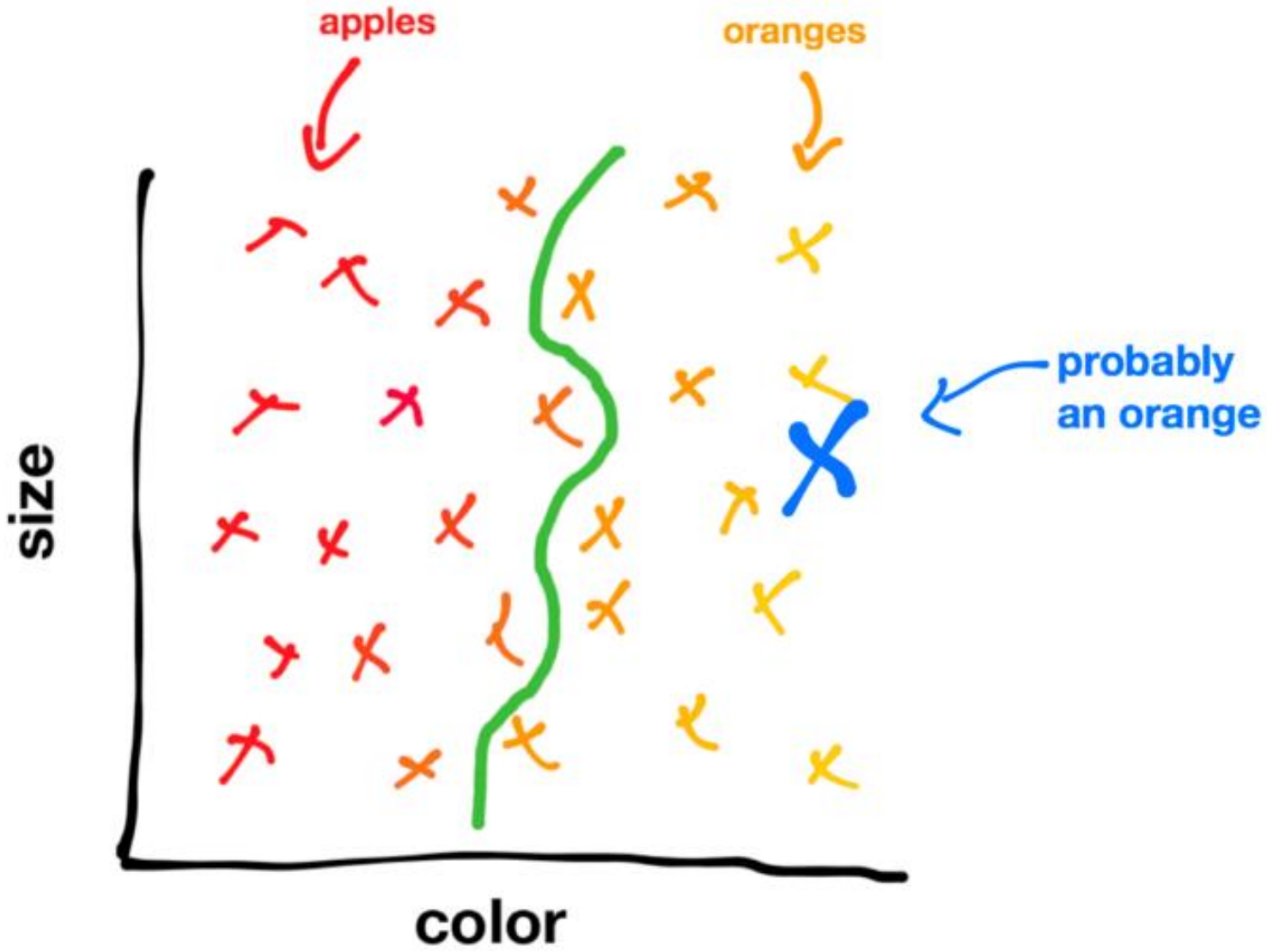
Classificação



Definição Informal

- Busca por uma **função matemática** que permita associar corretamente cada exemplo x_i de um conjunto de dados a um único rótulo categórico, y_i, denominado **classe**.
- Uma vez identificada, esta função pode ser aplicada a novos exemplos para **prever** as classes em que eles se enquadram.





© Machine Learning @ Berkeley

Definição Formal

- Dada uma coleção de exemplos ***f***, obter uma função (hipótese) ***h*** que seja uma aproximação de ***f***.
- A imagem de ***f*** é formada por rótulos de classes retirados de um conjunto finito e toda hipótese ***h*** é chamada de **Classificador**.
- O aprendizado consiste na busca da hipótese ***h*** que mais se aproxime da função original ***f***.

Avaliação: Acurácia do Classificador

- Medida de desempenho: **acurácia**, ou taxa de acerto do classificador:

$$Acc(h) = 1 - Err(h)$$

- A acurácia é uma função da **taxa de erro** (ou taxa de classificação incorreta):

$$Err(h) = \frac{1}{n}\sum_{i=1}^{n} \|y_i \neq h(i)\|$$

OBS: esta é UMA DAS possíveis métricas!

Onde:

- O operador ***||E||*** retorna 1 se a expressão *E* for verdadeira e 0 em caso contrário.
- ***n*** é o número de exemplos (registros da base de dados)
- y_i é a classe real associada ao i-ésimo exemplo
- ***h(i)*** é a classe indicada pelo classificador para o i-ésimo exemplo

Estamos interessados na Generalização

- **Generalização:** quando aplicamos um modelo aos dados que não foram utilizados para construí-lo
- Cada dataset é uma amostra finita de uma população
- Os dados de treinamento também vieram desta população
- Queremos que os modelos sejam bons não apenas para o conjunto de treinamento (usado para construir o modelo), mas para a população em geral
- Para isso podemos usar a técnica de **holdout** (separação do dataset em bases de treino e teste): o modelo é construído com um conjunto de dados e seu **erro de generalização** é avaliado com outro (não utilizado para o treinamento)

Classificação: Problemas

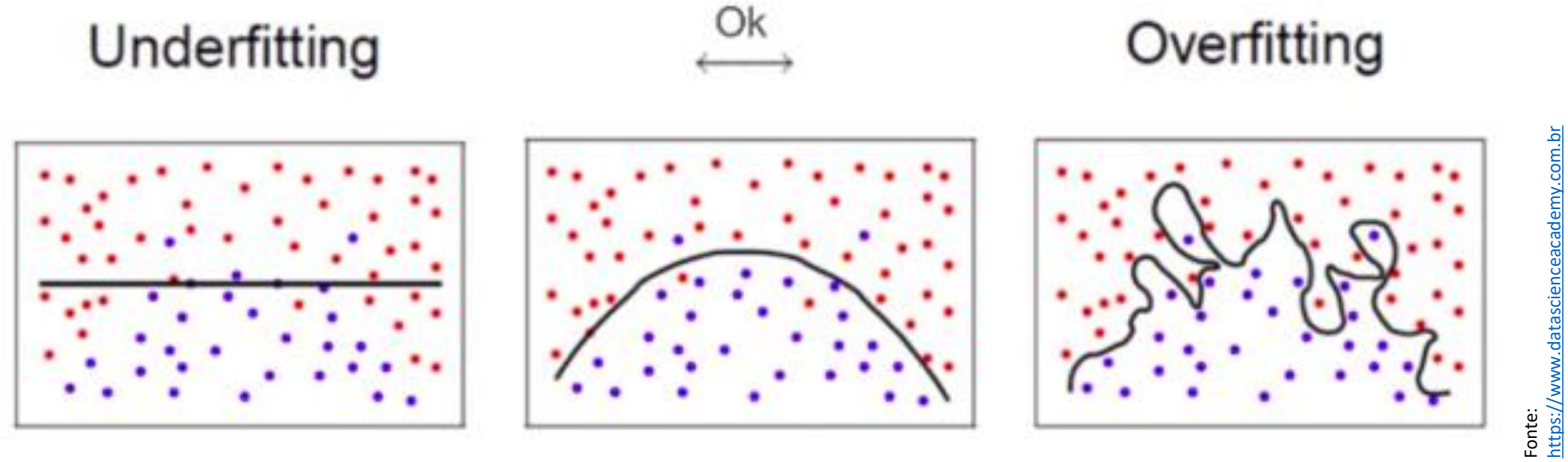
- Uma vez identificada uma hipótese (classificador), esta pode ser muito **específica** para o conjunto de treinamento utilizado.
- Caso este conjunto não corresponda a uma amostra suficientemente representativa da população, o Classificador pode ter um bom desempenho no conjunto de **treinamento**, mas não no conjunto de **teste**.
  - ***Overfitting:*** o classificador se ajustou em excesso ao conjunto de treinamento.
- O algoritmo de aprendizado pode ter parametrizações inadequadas.
  - ***Underfitting:*** o classificador se ajustou pouco ao conjunto de treinamento

Overfitting e Underfitting

- Geralmente, o erro de generalização (teste) é maior que o erro de treinamento. Idealmente, estes erros são próximos.
- Se o erro de **generalização** é grande demais, pode estar ocorrendo ***overfitting***: o modelo está memorizando os padrões de treinamento em vez de descobrir regras ou padrões generalizados.
- Se o erro de **treino** é grande demais, provavelmente está ocorrendo ***underfitting***.

OBS: Modelos mais simples tendem a generalizar melhor.

Overfitting e Underfitting



Fonte: https://www.datascienceacademy.com.br

Underfitting: modelo com falta de complexidade para o problema

Modelo adequado para o problema

Overfitting: modelo com excesso de complexidade para o problema

Dilema Bias x Variância (Underfitting x Overfitting)

- Na aprendizagem supervisionada, ao mesmo tempo em que o modelo preditivo precisa ser suficientemente **flexível** para aproximar os dados de treinamento, o processo de treinamento deve **evitar** que o modelo absorva os **ruídos** da base.
- O modelo deve **capturar** as regularidades dos dados de treinamento, mas também **generalizar** bem para dados desconhecidos.

OBS: *Sempre há uma parcela de erro irredutível, que não pode ser reduzido por nenhum modelo, e é resultado da aleatoriedade inerente ou de um conjunto de dados incompletos.*

Métricas de Avaliação: Matriz de confusão

- A **matriz de confusão** oferece um detalhamento do desempenho do modelo de classificação: mostra, para cada classe, o número de classificações corretas em relação ao número de classificações indicadas pelo modelo.

	spam (predito)	não spam (predito)
spam (real)	23 (VP)	1 (FN)
não spam (real)	12 (FP)	556 (VN)

Fonte: The 100 Page Machine Learning Book

Classes	Predita C1	Predita C2
Verdadeira C1	Verdadeiros Positivos	Falsos Negativos
Verdadeira C2	Falsos Positivos	Verdadeiros Negativos

Classes	Predita C1	Predita C2	...	Predita Cn
Verdadeira C1	M(C1, C1)	M(C1, C2)	...	M(C1, Cn)
Verdadeira C2	M(C2, C1)	M(C2, C2)	...	M(C2, Cn)
...	...	...	...	...
Verdadeira Cn	M(Cn, C1)	M(Cn, C2)	...	M(Cn, Cn)

Métricas de Avaliação: Acurácia

- A **acurácia** é o número de exemplos classificados corretamente dividido pelo número total de exemplos classificados.
- É uma métrica útil quando os erros nas predições de todas as classes são igualmente importantes.

$$\text{Acurácia} = \frac{VP+VN}{VP+VN+FP+FN}$$

- ***OBS:*** *A* ***acurácia sensível ao custo*** *pode ser usada quando as classes têm importância diferente. É atribuído um custo aos dois tipos de erros: FP e FN.*

Métricas de Avaliação: Precisão e Recall

- A **precisão** é a razão entre o número de predições **VP** sobre a quantidade total de **predições positivas**.
- O **recall** é a razão entre o número de predições **VP** sobre a quantidade total de **exemplos positivos** no dataset.

$$\text{Precisão} = \frac{VP}{VP+FP}$$

$$\text{Recall} = \frac{VP}{VP+FN}$$

- ***Exemplo:*** *No problema de detecção de spam, queremos ter uma* ***alta precisão*** *(evitar cometer erros detectando uma mensagem verdadeira como spam) e podemos tolerar um* ***recall mais baixo*** *(algumas mensagens de spam em nossa caixa de entrada).*
- *Na prática, quase sempre temos que* ***escolher*** *entre uma alta precisão ou um alto recall. Tipicamente, é impossível ter ambos.*

Métricas de Avaliação: Área sob a Curva ROC (AUC)

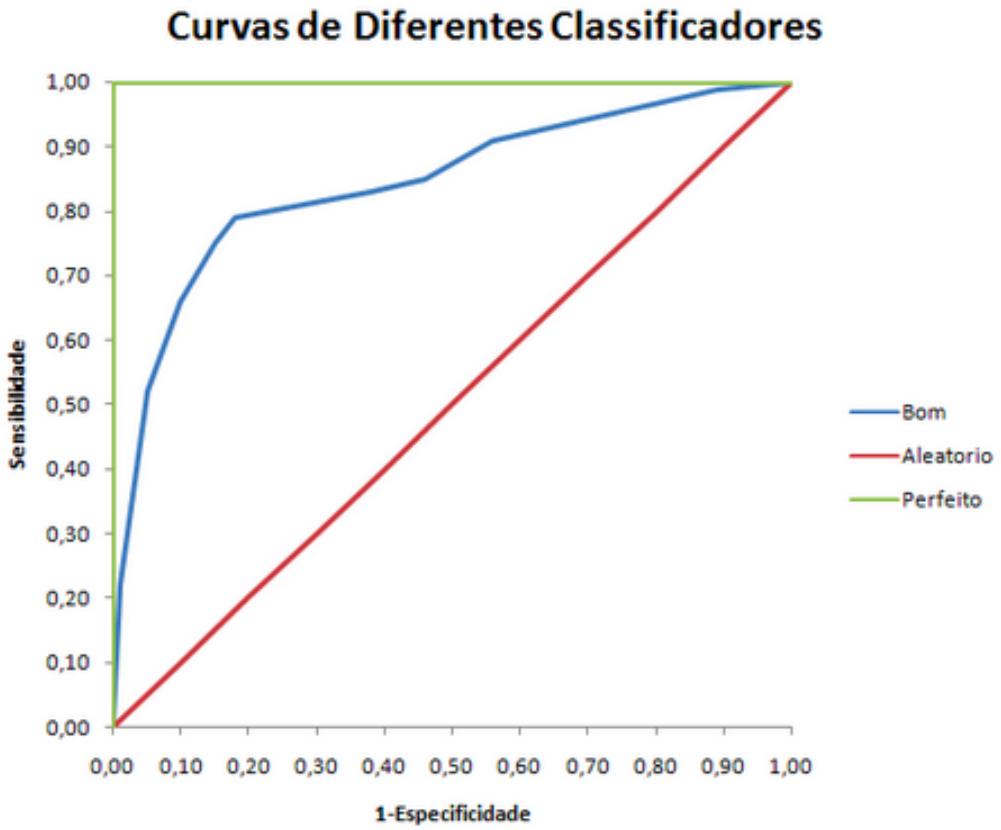
- **Curva ROC** (*Receiver Operating Characteristic*): contrasta os benefícios de uma classificação correta (TVP, sensibilidade ou recall) e o custo de uma classificação incorreta (TFP ou 1-especificidade)
  - ***Sensibilidade:*** capacidade de identificar corretamente os indivíduos que **apresentam** a característica de interesse
  - ***Especificidade:*** capacidade em identificar corretamente os indivíduos que **não** apresentam a característica de interesse

$$\text{Sensibilidade} = \text{TVP} = \frac{VP}{VP+FN}$$

$$\text{Especificidade} = \frac{VN}{FP+VN}$$

$$1\text{–Especificidade} = \text{TFP} = \frac{FP}{FP+VN}$$

Métricas de Avaliação: Área sob a Curva ROC (AUC)



Fonte: (Souza, 2009)

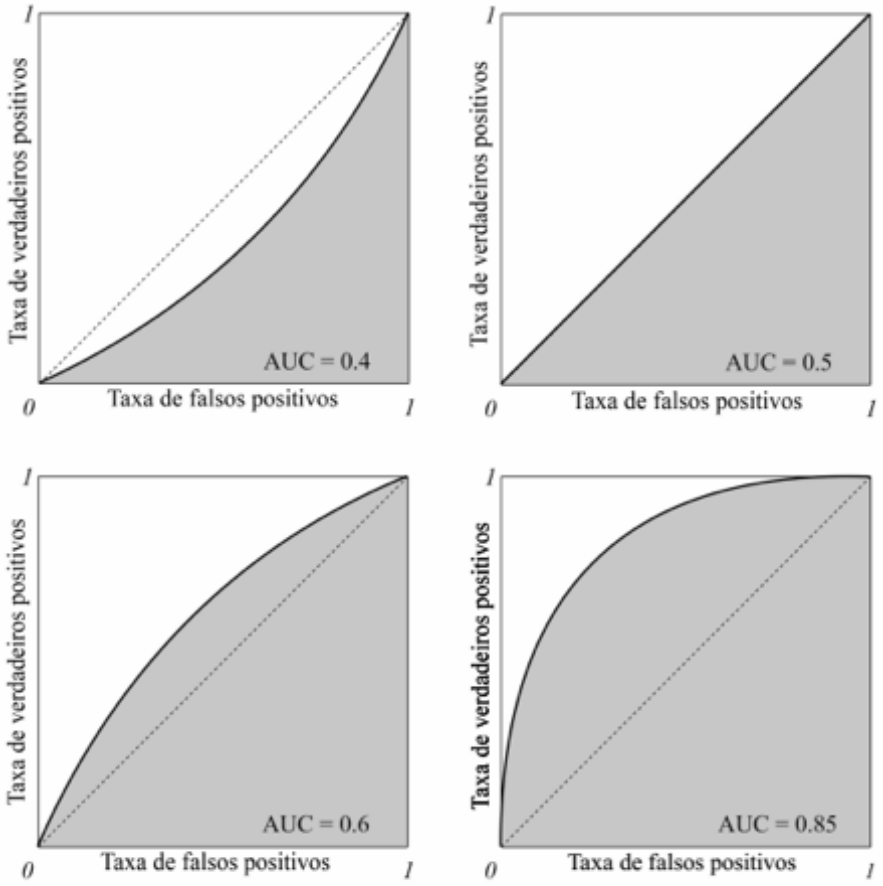


Figura 5.3: A área sob a curva AUC (mostrada em cinza).

Fonte: The 100 Page Machine Learning Book

Teorema NFL (No Free Lunch)

Software Science Lab

- **Não existe** um algoritmo de aprendizado que seja **superior** a todos os demais quando considerados todos os problemas de classificação possíveis.
- A cada problema, os algoritmos disponíveis devem ser **experimentados** a fim de identificar aqueles que obtêm melhor desempenho.
- **Classificador nulo (ou dummy):** sempre retorna apenas uma classe (geralmente a mais frequente). Nosso classificador deve ser **pelo menos** melhor que o nulo!

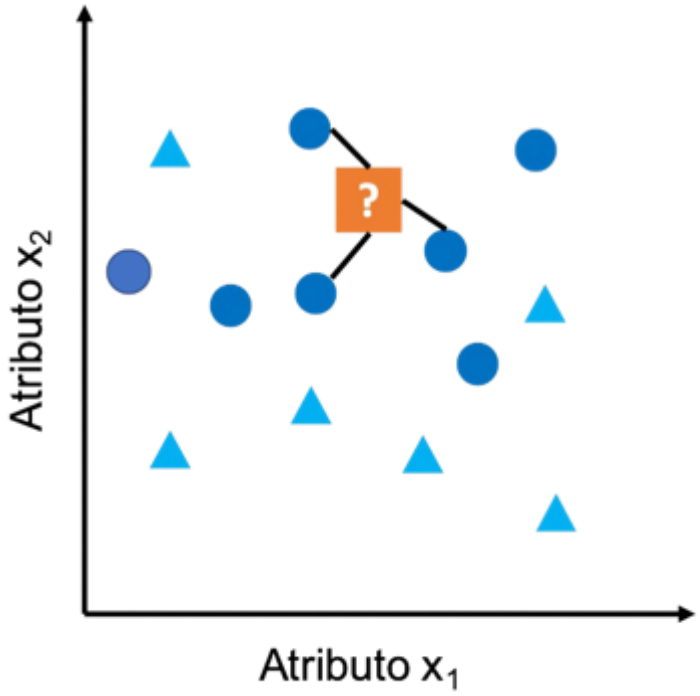
Software Science Lab

Algoritmos para Classificação

Engenharia de Software para Ciência de Dados

KNN

- **KNN: k-Nearest Neighbours (k-Vizinhos Mais Próximos)**
- Algoritmo **simples** de entender e que **funciona** muito bem na prática
- Algoritmo **não-paramétrico**: não assume premissas sobre a distribuição dos dados
- Considera que os exemplos **vizinhos** são **similares** ao exemplo que se deseja classificar



Considera que os registros do conjunto de dados correspondem a pontos no R^n, em que cada atributo corresponde a uma dimensão deste espaço

KNN: Funcionamento

Software Science Lab

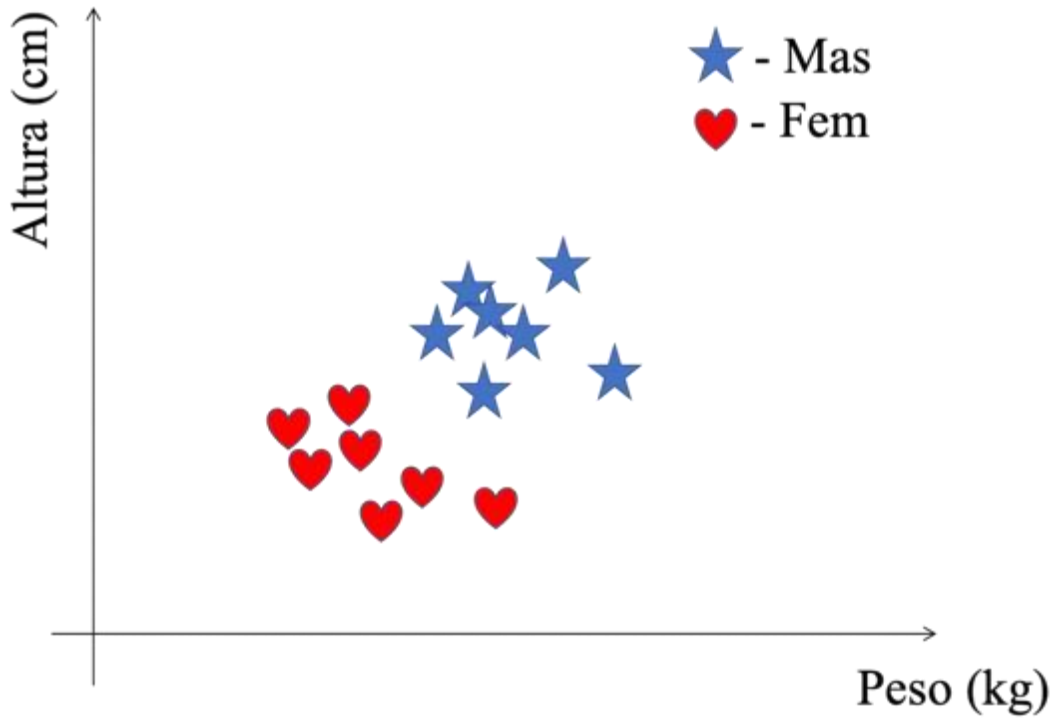
- O conjunto de dados (rotulado) é armazenado. Quando um novo registro deve ser classificado, ele é comparado a todos os registros do conjunto de treinamento para identificar ***k*** (parâmetro de entrada) **vizinhos mais próximos** (mais semelhantes) de acordo com alguma métrica de distância
- A classe do novo registro é determinada por **inspeção** das classes desses vizinhos mais próximos, de acordo com a métrica selecionada
- Na maioria das implementações, os atributos são **normalizados**, para que tenham a mesma contribuição na predição da classe

KNN: Algoritmo

1. Definição da métrica de distância, critério de desempate e valor de *k*
2. Cálculo da distância do novo registro a cada um dos registros existentes no conjunto de referência
3. Identificação dos *k* registros do conjunto de referência que apresentaram menor distância em relação ao novo registro (mais similares)
4. Apuração da classe mais frequente entre os *k* registros identificados no passo anterior (usando votação majoritária)

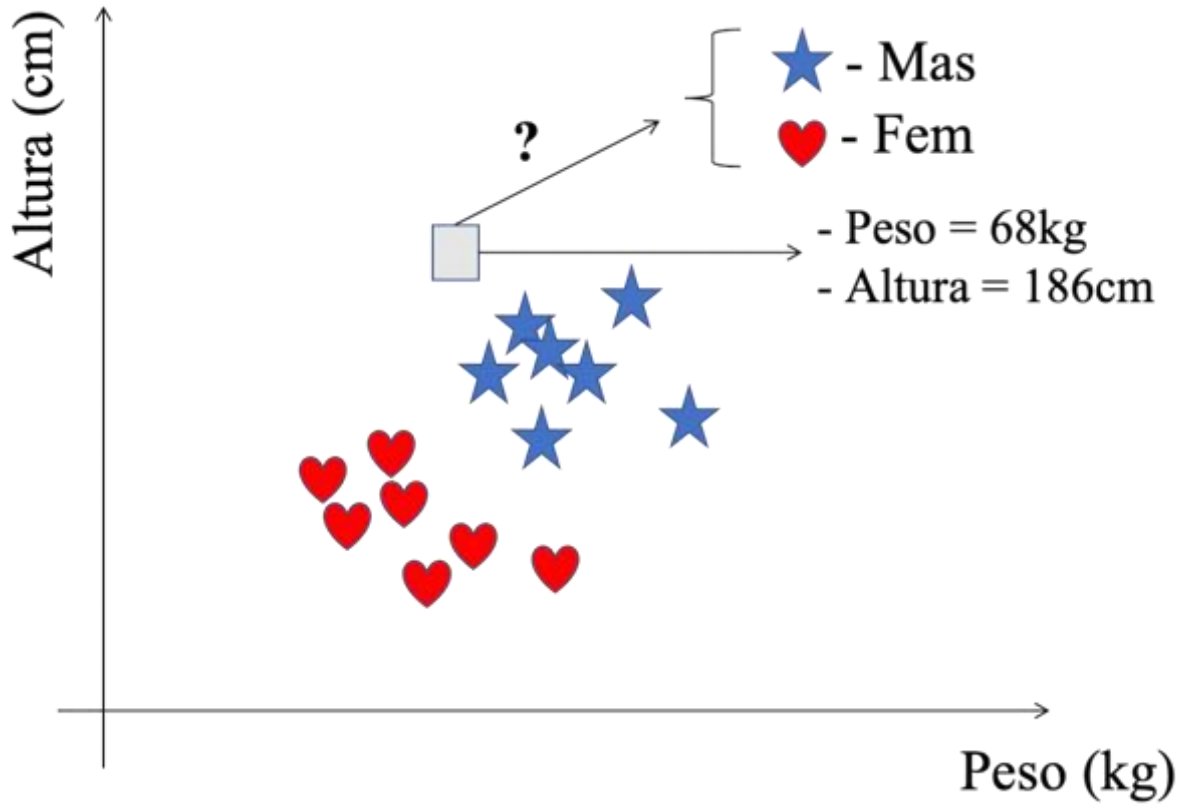
KNN: Exemplo

id	Peso (kg)	Altura (cm)	Sexo
1	55	167	F
2	52	160	F
3	48	155	F
...	...	...	...
12	70	175	M
13	74	170	M
14	80	180	M



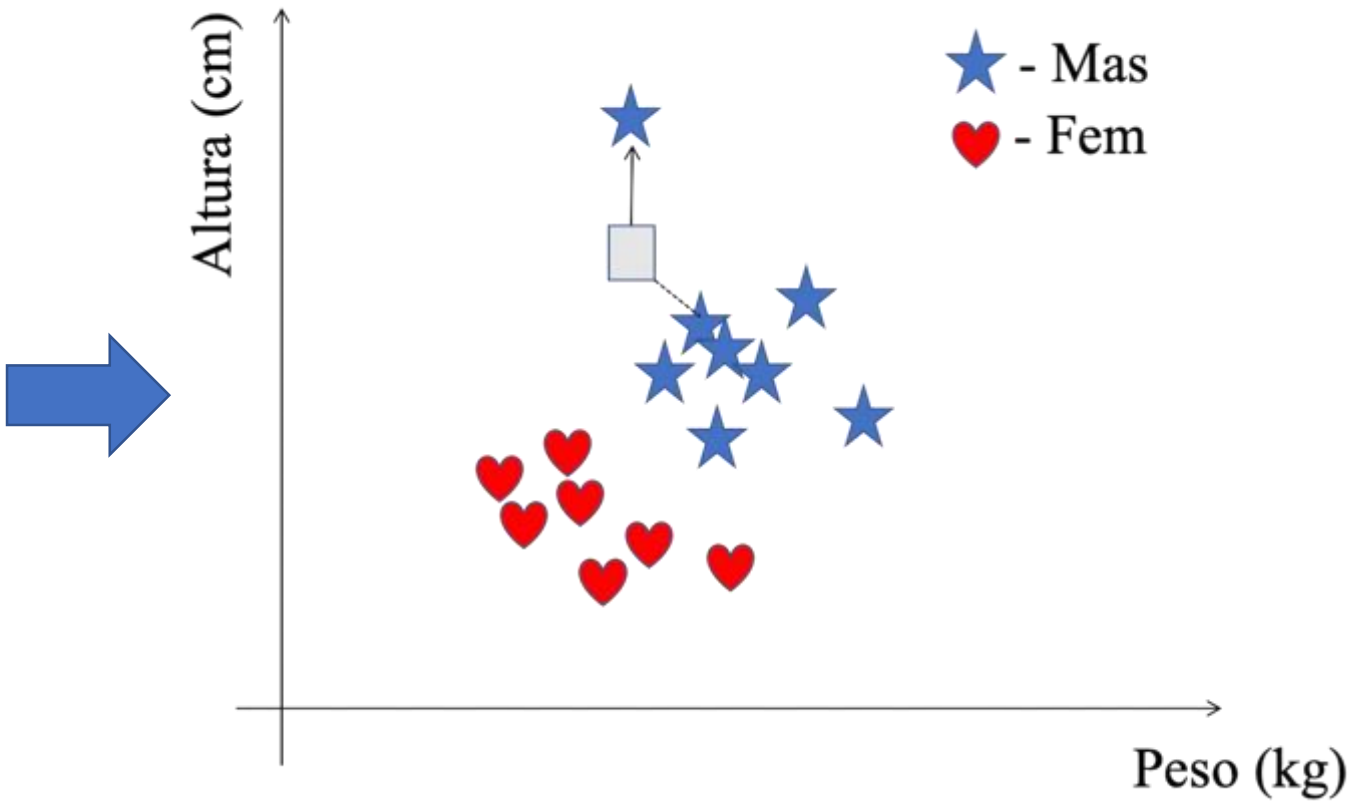
KNN: Exemplo

- Imagine que neste momento chega um novo indivíduo, com 68 kg e altura 186 cm, e deseja-se determinar o sexo deste indivíduo.



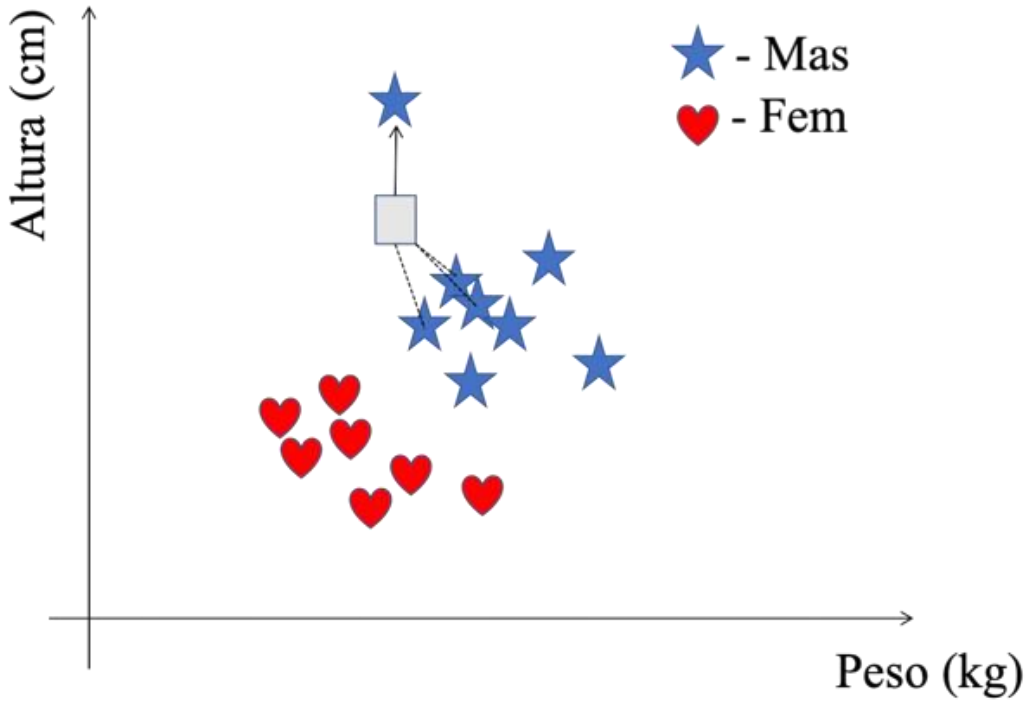
KNN: Exemplo

- Podemos utilizar o KNN para determinar o sexo deste novo indivíduo, que é dado pela informação dos k vizinhos mais próximos.
- Precisamos então determinar o valor de k.
- Para **k = 1**:

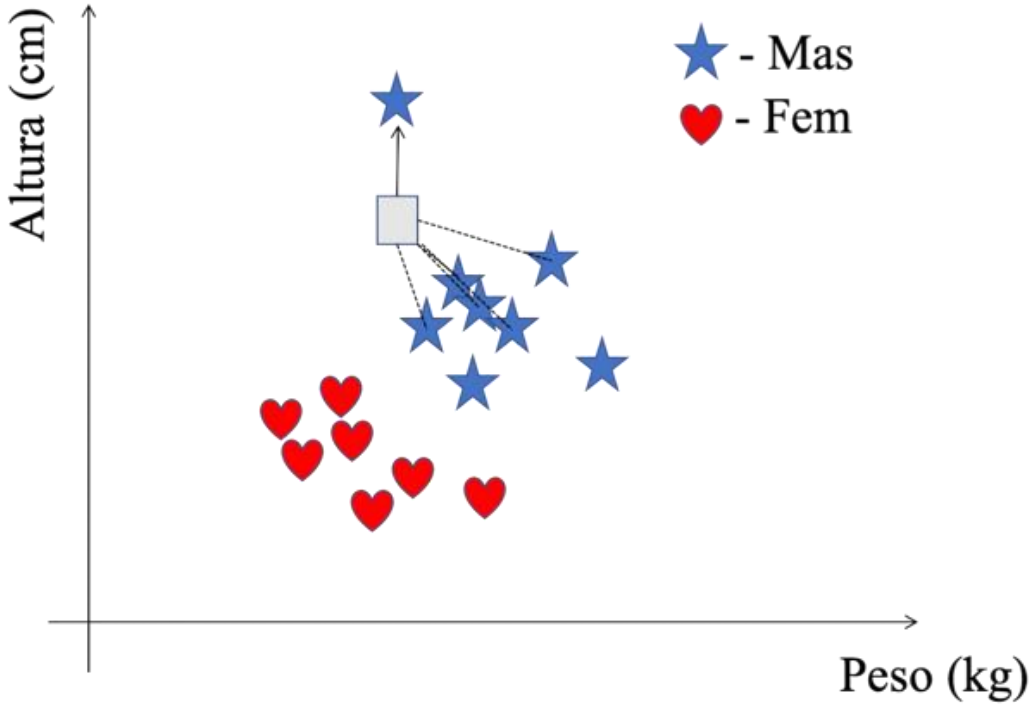


KNN: Exemplo

- Para **k = 3**:



- Para **k = 5**:



KNN: Perguntas

Software Science Lab

1. Que tipo de **distância** usar?

2. Qual o **valor** adequado de ***k***?

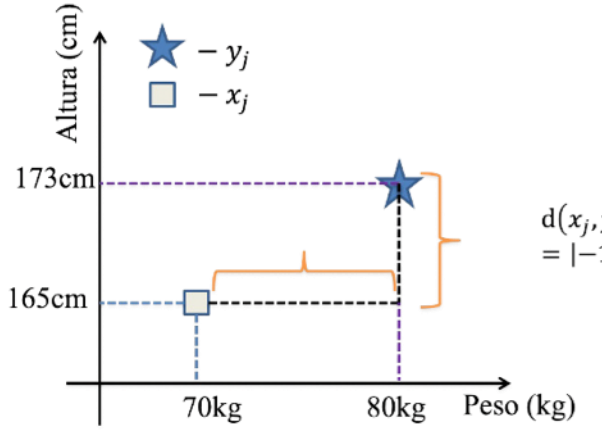
Pergunta 1) Que tipo de Distância usar?

Software Science Lab

Euclidiana: $d(x_j, y_j) = \sqrt{\sum_{j=1}^{J}(x_j - y_j)^2}$

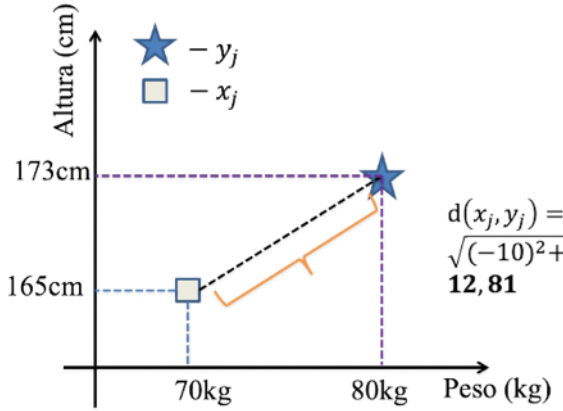
Manhattan: $d(x_j, y_j) = \sum_{j=1}^{J}|x_j - y_j|$

Minkowski: $d(x_j, y_j) = \max(|x_j - y_j|)$



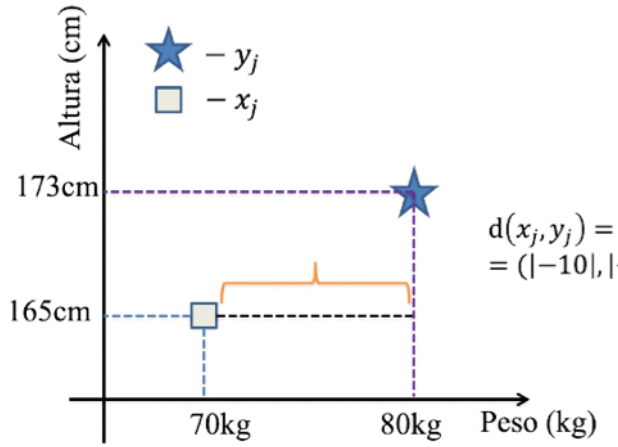
Manhattan

$d(x_j, y_j) = |70 - 80| + |165 - 173|$
$= |-10| + |-8| = 10 + 8 = \mathbf{18}$



Euclidiana

$d(x_j, y_j) = \sqrt{(70 - 80)^2 + (165 - 173)^2} =$
$\sqrt{(-10)^2 + (-8)^2} = \sqrt{100 + 64} = \sqrt{164} =$
$\mathbf{12,81}$



Minkowski

$d(x_j, y_j) = \max(|70 - 80|, |165 - 173|) =$
$= (|-10|, |-8|) = \max(10,8) = \mathbf{10}$

Pergunta 1) Que tipo de Distância usar?

Software Science Lab

- Para cada distância, há um resultado **diferente**!
- Qual distância escolher?
  - A decisão é **experimental**: devem ser criados diversos modelos diferentes, variando os parâmetros de distância, e aquele que apresentar melhor resultado (considerando, por exemplo, a acurácia de teste) será o melhor candidato

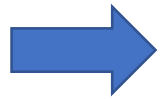
Pergunta 2) Qual o valor adequado de k?

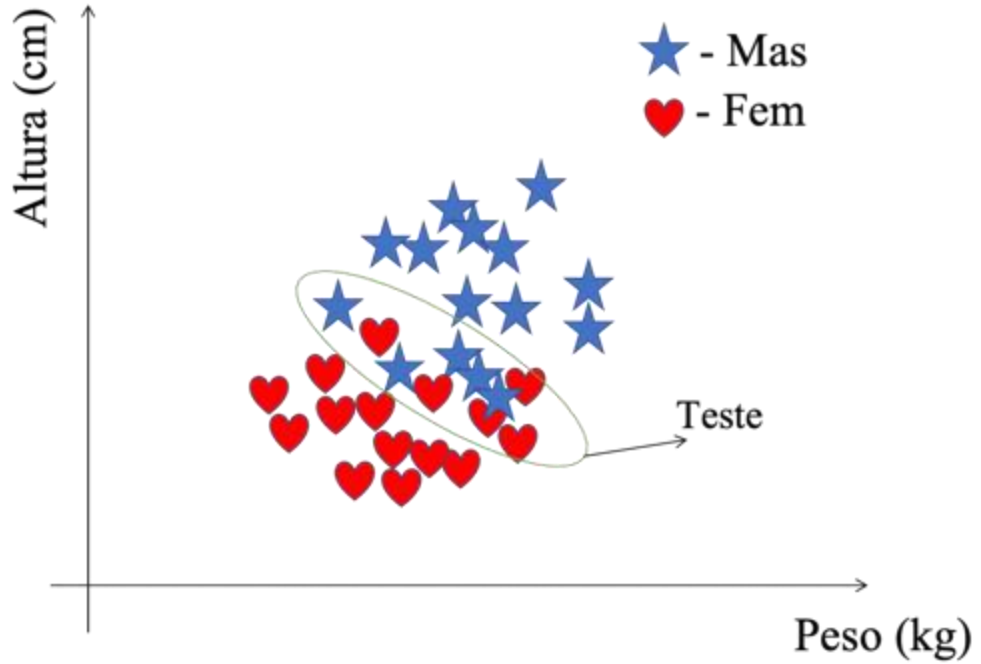
- Normalmente determinado em função do conjunto de dados.
- Em geral, quanto maior o valor de *k*, menor o efeito de eventuais ruídos no conjunto de referência, mas valores grandes de *k* tornam mais difusas as fronteiras entre as classes existentes
- Também é decidido **experimentalmente**

Como determinar o valor de k

1. Dividir os dados em **treino** e **teste**:

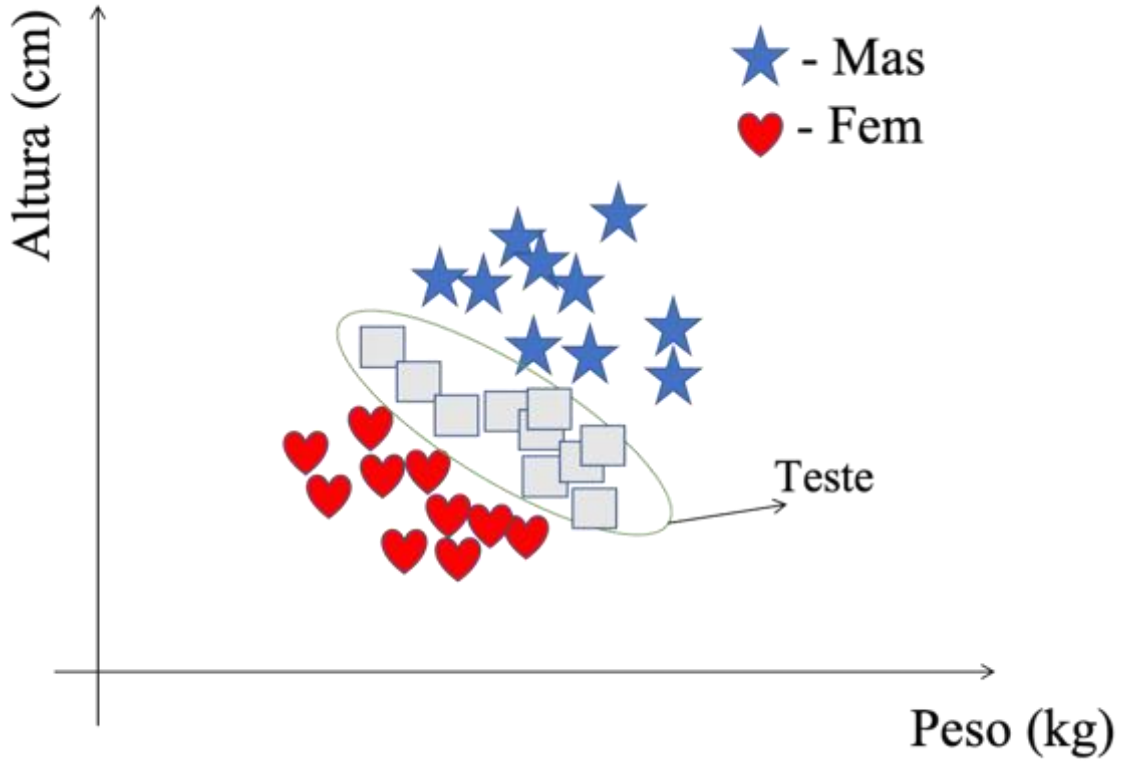
	id	Peso (kg)	Altura (cm)	Sexo
Treinamento	1	55	167	F
	2	52	160	F
	3	48	155	F
	...	...	...	...
	9	58	165	F
	10	64	169	F
	11	78	179	M
	...	...	...	...
	18	70	175	M
	19	74	170	M
	20	80	180	M
Teste	21	57	162	F
	22	59	161	F
	...	...	...	...
	29	72	170	M
	30	69	165	M





Como determinar o valor de k

2. Assumir que não conhecemos os rótulos dos exemplos do conjunto de teste:



Como determinar o valor de k

3. Aplicar o modelo e calcular a acurácia de teste para ***k = 1***, ***k = 3*** e ***k = 5***:

K	Acurácia
1	6/10 = 60%
3	7/10 = **70%**
5	6/10 = 60%

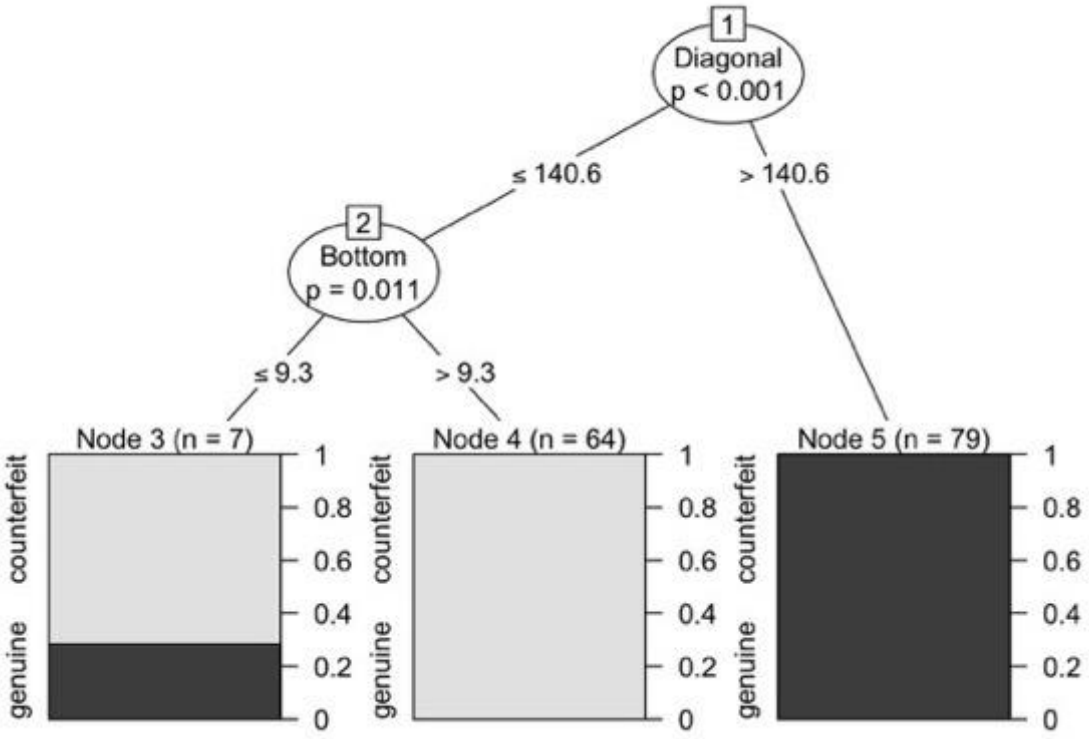
- Seleciona-se o **k** com maior acurácia com os dados de teste (no caso, 3)
- Sugere-se variar **k** de 1 a 20 e também as diferentes distâncias

Árvore de Decisão

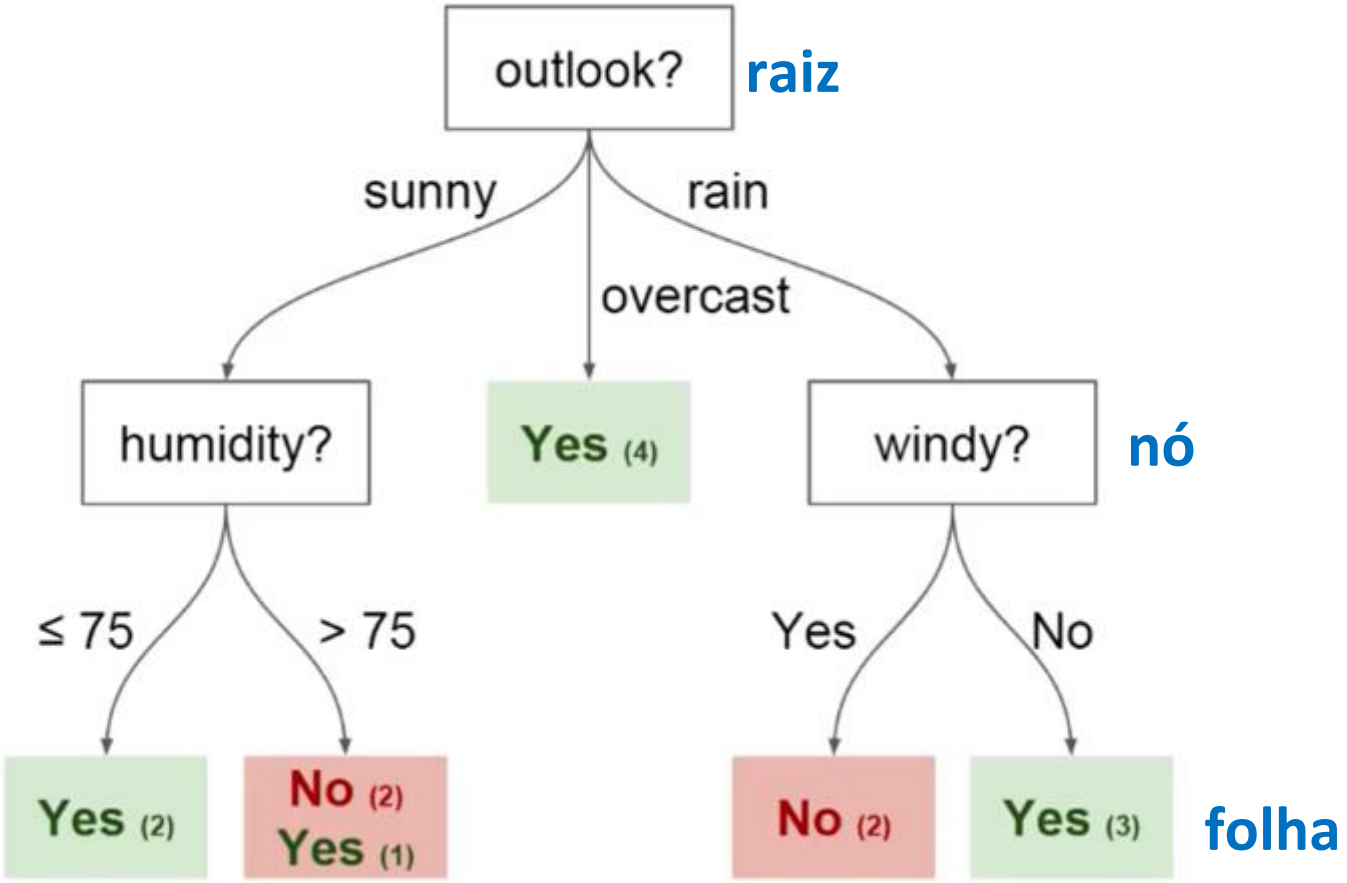
- Um dos modelos de decisão mais **simples de ser interpretado**
- Usam amostras das características dos dados para criar **regras** de decisão no formato de **árvore**
- Inspiradas na forma que **humanos** tomam decisão
- Apresentam a informação **visualmente**, de uma forma fácil de entender
- Podem ser usadas para problemas de **classificação ou regressão**
- Reduzem os dados em um **conjunto de regras** que podem ser usadas para uma decisão de classificação ou gerar uma predição de valor

Árvore de Classificação

- Possibilitam **seleção automática de variáveis** para compor suas estruturas
- Cada nó interno representa uma **decisão** sobre um atributo que determina como os dados estão particionados pelos seus nós filhos
- Para classificar um novo exemplo, basta **testar** os valores dos atributos na árvore e percorrê-la até se atingir um nó folha (classe predita)



Árvore de Classificação



Árvore de Classificação

- Há diferentes algoritmos para sua elaboração, todos bem parecidos:
  - ID3 (extensão do D3)
  - C4.5 (successor do ID3)
  - C5.0 (C4.5 aprimorado)
  - CART

- Em geral, a construção da árvore é realizada de acordo com alguma abordagem **recursiva** de particionamento do conjunto de dados
- A principal distinção está nos processos de:
  - Seleção de variáveis
  - Critério para particionar
  - Critério de parada para o crescimento da árvore

Árvore de Classificação

- Para construir uma árvore de decisão, precisamos decidir **que perguntas fazer** e em **qual ordem** para particionar os dados
  - Em cada etapa, eliminamos algumas possibilidades
- Gostaríamos de escolher perguntas cujas respostas nos dessem **muita informação** sobre o que esta árvore deveria prever
- A **entropia** representa a **incerteza** associada com os dados:
  - Se todos os pontos de dados pertencem predominantemente a uma única classe, então não há incerteza real, o que significa que deve haver **baixa entropia**

Árvore de Classificação – Algoritmo Geral

- Inicialmente, a **raiz** da árvore contém todo o conjunto de dados com exemplos misturados de várias classes.
- Um **predicado** (ponto de separação) é escolhido como sendo o atributo que melhor separa as classes, e induz uma divisão do conjunto de dados em dois ou mais conjuntos disjuntos, cada um deles associado a um **nó filho**.
- Cada novo **nó** abrange um subconjunto do conjunto de dados original que é recursivamente separado até que o subconjunto associado a cada **nó folha** consista inteira ou predominantemente de registros de uma mesma classe.

Algoritmo Resumido

Software Science Lab

1. Calcular a entropia do conjunto *T* completo

2. Para cada atributo

2.1. Calcular o ganho de informação se escolhido para particionamento

3. Selecionar o atributo com maior ganho de informação para o nó raiz da árvore

4. Subdividir o conjunto *T*

5. Repetir o procedimento para cada nó gerado

Ideia geral: *usa uma abordagem de divisão e conquista, começando com o conjunto de dados inteiro e aplicando a seleção de atributos para tentar criar os subgrupos "mais puros" possíveis em relação à variável alvo.*

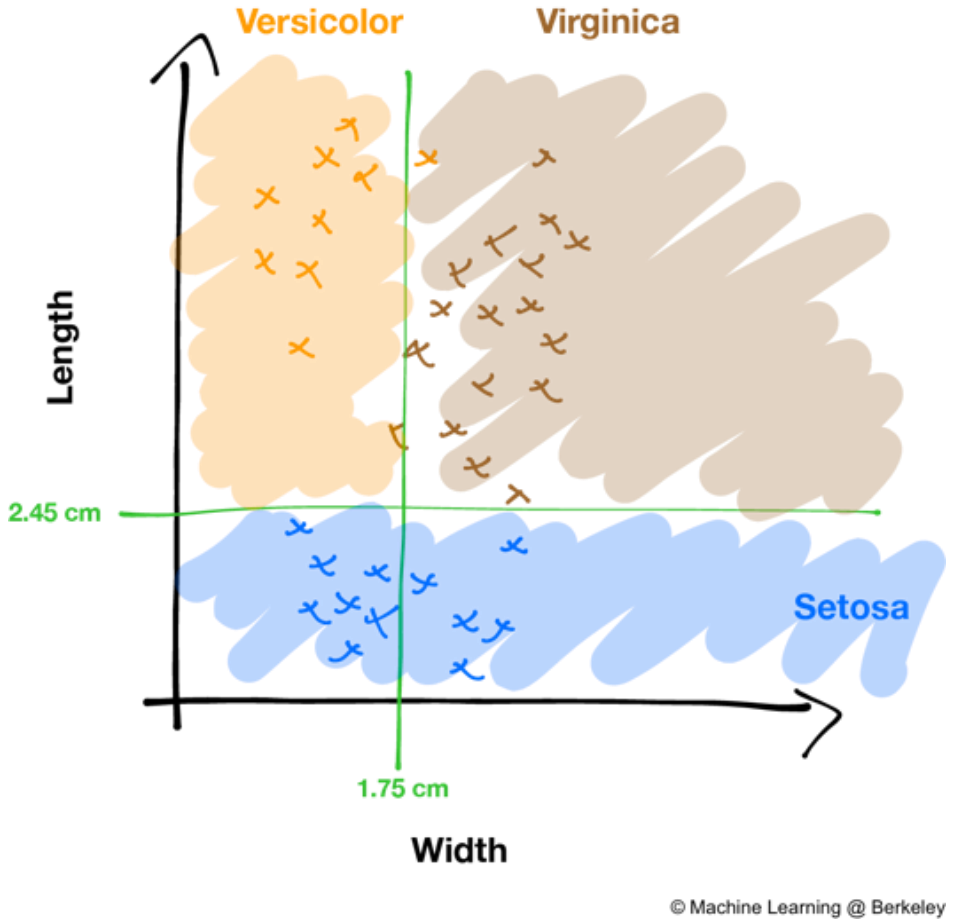
OBS: *Podemos usar algoritmos de poda (pruning) para reduzir a profundidade e a complexidade da árvore*

Medidas de Seleção de Atributos

Critério pode variar de acordo com o algoritmo. Alguns exemplos:

- Entropia
- Ganho de Informação
- Gini index
- Redução da variância

Árvore de Classificação – Dataset Iris



Naïve Bayes: Motivação

- **Naïve (Ingênuo)** porque desconsidera completamente a correlação entre os atributos (características).
  - Se um determinado animal é considerado um “Gato” se tiver bigodes, orelhas em pé e aproximadamente 30 cm de altura , o algoritmo não vai levar em consideração a correlação entre esses fatores, tratando cada um de forma independente.
- **Bayes** porque baseia-se no **Teorema de Bayes**, estando relacionado com o cálculo de probabilidades condicionais.

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)},$$

em que A e B são eventos e $P(B) \neq 0$.

Naïve Bayes: Exemplo

Software Science Lab

Neste universo:

Fruta	Longa	Doce	Amarela	Total
Banana	400	350	450	**500**
Laranja	0	150	300	**300**
Outra	100	150	50	**200**
Total	**500**	**650**	**800**	**1000**

Sabendo que uma nova fruta é longa, doce e amarela, ela será uma banana, uma laranja ou outra?

Naïve Bayes: Exemplo

Fruta	Longa	Doce	Amarela	Total
Banana	400	350	450	**500**
Laranja	0	150	300	**300**
Outra	100	150	50	**200**
Total	**500**	**650**	**800**	**1000**

O que sabemos?

- 50% das frutas são bananas
- 30% são laranjas
- 20% são outras frutas

1

- P(Y=Banana) = 500 / 1000 = 0,50
- P(Y=Laranja) = 300 / 1000 = 0,30
- P(Y=Outra) = 200 / 1000 = 0,20

Naïve Bayes: Exemplo

Fruta	Longa	Doce	Amarela	Total
Banana	400	350	450	**500**
Laranja	0	150	300	**300**
Outra	100	150	50	**200**
Total	**500**	**650**	**800**	**1000**

- 50% das frutas são longas
- 65% são doces
- 80% são amarelas

2

- P(x1=Longa) = 500 / 1000 = 0,50
- P(x2=Doce) = 650 / 1000 = 0,65
- P(x3=Amarela) = 800 / 1000 = 0,80

Naïve Bayes: Exemplo

Fruta	Longa	Doce	Amarela	Total
Banana	400	350	450	**500**
Laranja	0	150	300	**300**
Outra	100	150	50	**200**
Total	**500**	**650**	**800**	**1000**

- Das 500 bananas, 400 (0,8) são longas, 350 (0,7) são doces e 450 (0,9) são amarelas

3

- P(x1=Longa | Y=Banana) = 400 / 500 = 0,80
- P(x2=Doce | Y=Banana) = 350 / 500 = 0,70
- P(x3=Amarela | Y=Banana) = 450 / 500 = 0,90

Naïve Bayes: Exemplo

Fruta	Longa	Doce	Amarela	Total
Banana	400	350	450	**500**
Laranja	0	150	300	**300**
Outra	100	150	50	**200**
Total	**500**	**650**	**800**	**1000**

4

- Das 300 laranjas, 0 são longas, 150 (0,5) são doces e 300 (1) é amarela

- P(x1=Longa | Y=Laranja) = 0 / 300 = 0
- P(x2=Doce | Y= Laranja) = 150 / 300 = 0,50
- P(x3=Amarela | Y= Laranja) = 300 / 300 = 1

Naïve Bayes: Exemplo

Fruta	Longa	Doce	Amarela	Total
Banana	400	350	450	**500**
Laranja	0	150	300	**300**
Outra	100	150	50	**200**
Total	**500**	**650**	**800**	**1000**

5

- Das outras 200 frutas, 100 (0,5) são longas, 150 (0,75) são doces e 50 (0,25) são amarelas

- P(x1=Longa | Y=Outra) = 100 / 200 = 0,50
- P(x2=Doce | Y= Outra) = 150 / 200 = 0,75
- P(x3=Amarela | Y= Outra) = 50 / 200 = 0,25

Naïve Bayes: Exemplo

Teorema de Bayes: calcula a probabilidade de um evento dado que outro evento já ocorreu (probabilidade condicional)

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)},$$

em que A e B são eventos e $P(B) \neq 0$.

"Fato" que ocorreu (f): a nova fruta é longa, doce e amarela

$$P(c|f) = \frac{P(f|c)P(c)}{P(f)}$$

P(banana | f) = ?
P(laranja | f) = ?
P(outra | f) = ?

O MAIOR VALOR DE PROBABILIDADE DETERMINARÁ A CLASSE!

Naïve Bayes: Exemplo

- $\mathbf{x_1}$ = longa
- $\mathbf{x_2}$ = doce
- $\mathbf{x_3}$ = amarela

$$P(c|X) = \frac{P(x_1|c)\ P(x_2|c)P(x_3|c)P(c)}{P(x_1)P(x_2)P(x_3)P(xn)}$$

É CONSTANTE!

$$P(c|X) \propto P(x_1|c)\ P(x_2|c)P(x_3|c)P(c)$$

Naïve Bayes: Exemplo

- 50% das frutas são bananas (0,5)
- Das 500 bananas, 400 (0,8) são longas, 350 (0,7) são doces e 450 (0,9) são amarelas

$$P(banana|longa, doce, amarela) \propto$$
$$P(longa|banana)\ P(doce|banana) P(amarela|banana) P(banana)$$

$$P(banana|longa, doce, amarela) \propto 0{,}8 \text{ x } 0{,}7 \text{ x } 0{,}9 \text{ x } 0{,}5 = \mathbf{0{,}252}$$

Naïve Bayes: Exemplo

- 30% das frutas são laranjas (0,3)
- Das 300 laranjas, 0 (0) são longas, 150 (0,5) são doces e 300 (1) são amarelas

$$P(laranja|longa, doce, amarela) \propto$$
$$P(longa|laranja)\ P(doce|laranja)P(amarela|laranja)P(laranja)$$

$$P(laranja|longa, doce, amarela) \propto 0 \text{ x } 0{,}5 \text{ x } 1 \text{ x } 0{,}3 = \mathbf{0}$$

Naïve Bayes: Exemplo

- 20% das frutas são outras (0,2)
- Das 200 outras, 100 (0,5) são longas, 150 (0,75) são doces e 50 (0,25) são amarelas

$$P(outra|longa, doce, amarela) \propto$$
$$P(longa|outra)\ P(doce|outra)P(amarela|outra)P(outra)$$

$$P(outra|longa, doce, amarela) \propto 0{,}5 \text{ x } 0{,}75 \text{ x } 0{,}25 \text{ x } 0{,}2 = \mathbf{0{,}019}$$

Naïve Bayes: Exemplo

$P(banana|longa, doce, amarela) \propto$ **0,252**

$P(laranja|longa, doce, amarela) \propto$ **0**

$P(outra|longa, doce, amarela) \propto$ **0,019**

Naïve Bayes: Definição Formal

Sejam:

- $X(A_1, A_2, \ldots, A_n, C)$ um conjunto de dados
- $c_1, c_2, \ldots, c_k$ são as **classes** do problema (valores possíveis do atributo alvo C)
- *R* um **registro** que deve ser classificado
- $a_1, a_2, \ldots, a_k$ os valores que *R* assume para os **atributos** previsores $A_1, A_2, \ldots, A_n$, respectivamente.

Naïve Bayes: Definição Formal

Software Science Lab

O algoritmo consiste em:

1. Calcular as **probabilidades condicionais** $P(C=c_i|R)$, $i = 1, 2, \ldots, k$
2. Indicar como **saída** do algoritmo a classe *c* tal que $P(C=c|R)$ seja **máxima**, quando considerados todos os valores possíveis do atributo alvo *C*

- A intuição por trás do algoritmo é dar **mais peso** para as classes **mais frequentes**

SVM: Support Vector Machines (Máquinas de Vetor de Suporte)

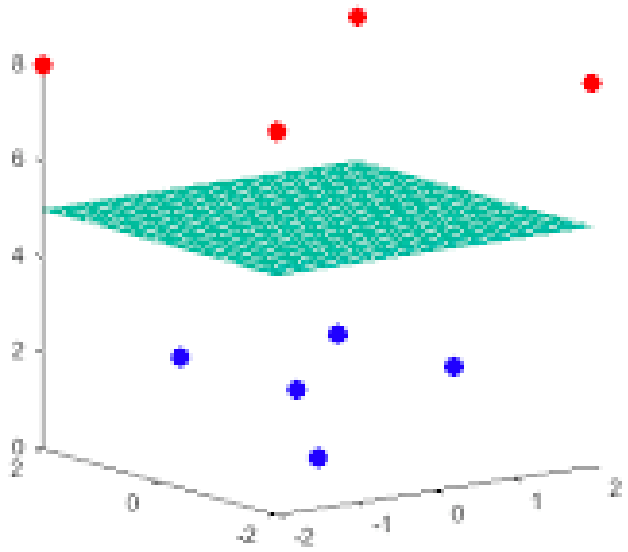
- Realiza um mapeamento **não linear** para **transformar** os dados de treino originais em uma **dimensão maior**
- São adicionados **novos termos**, mais complexos, ao problema (por exemplo, o quadrado ou o produto das características) para este mapeamento em dimensão maior
- Nesta nova dimensão, o algoritmo busca pelo **hiperplano** que separa os dados linearmente de forma ótima
  - Com um mapeamento apropriado para uma dimensão suficientemente alta, dados de duas classes podem ser **sempre** separados por um hiperplano
- O SVM encontra este hiperplano usando **vetores de suporte** (exemplos essenciais para o treinamento) e **margens**, definidas pelos vetores de suporte.

SVM: Support Vector Machines (Máquinas de Vetor de Suporte)

Software Science Lab

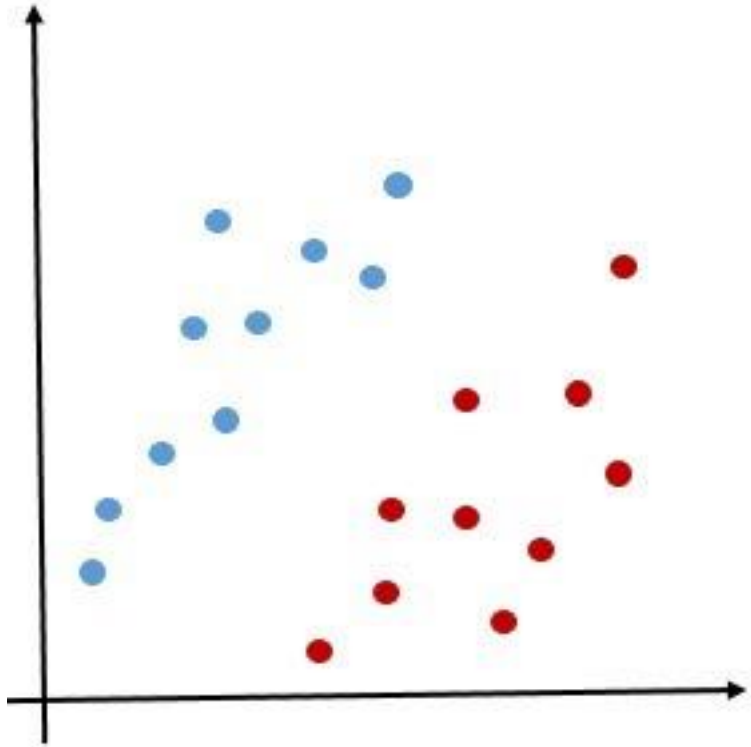
- O SVM constrói **classificadores lineares**, que separam os conjuntos de dados por meio de um **hiperplano** (generalização do conceito de plano para dimensões > 3)

- Em um espaço *p*-dimensional, um **hiperplano** é um subespaço achatado de dimensão ***p*-1**



SVM: Exemplo

- Considere um conjunto de dados de treinamento **linearmente separável** na forma $\{x_i, y_i\}$, em que:
  - x_i corresponde ao vetor de 2 atributos previsores
  - $y_i \in \{-1,1\}$, às duas classes possíveis do problema.
- O conjunto de dados de entrada é utilizado para construir uma função de decisão $f(x)$ tal que:
  - Se $f(x) > 0$, então $y_i = 1$
  - Se $f(x) < 0$, então $y_i = -1$

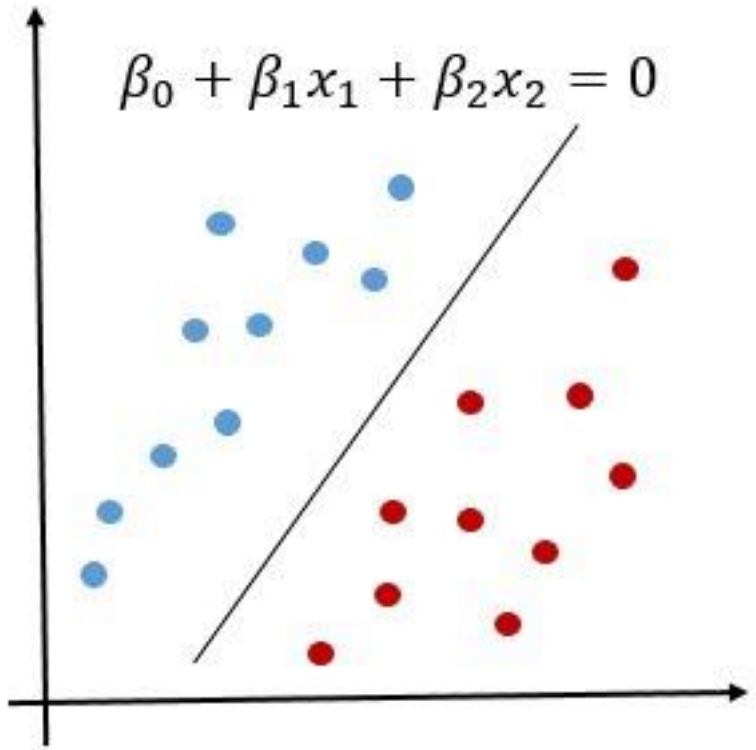


SVM: Exemplo

Para **d = 2**:

- O hiperplano é uma **reta** e sua equação é $\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$,

(β_1 e β_2 são parâmetros que determinam a inclinação da reta e β_o o ponto de corte do eixo y)



- O **propósito** do SVM é determinar parâmetros da reta (β_0, β_1 e β_2), que permitem **separar** os conjuntos dos dados de treinamento em duas classes possíveis:
  - $\beta_0 + \beta_1 x_1 + \beta_2 x_2 < 0$
  - $\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0$

SVM: Exemplo

- Da mesma forma que um plano tem **2** dimensões e divide um conjunto tridimensional em 2 espaços de dimensão **3**, um hiperplano em um espaço **n**-dimensional tem **n-1** dimensões e divide este espaço em 2 subespaços de dimensão **n**.

Para **d = p**:

- A equação do hiperplano é

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ \ldots + \beta_n x_n = 0$$

- Neste caso, o hiperplano também divide o espaço p-dimensional em 2 metades:
  - $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ \ldots + \beta_n x_n < 0$
  - $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ \ldots + \beta_n x_n > 0$

SVM: Exemplo

Voltando ao nosso exemplo…

- Como afirmamos que o problema é linearmente separável, **é possível** construir um hiperplano que separe as observações de treino perfeitamente de acordo com seus rótulos de classe.

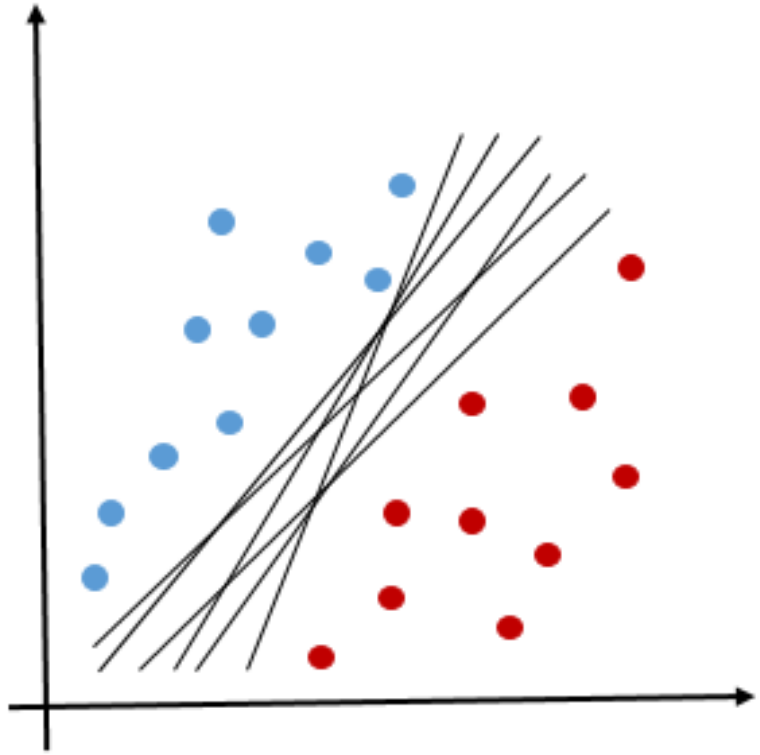
Este hiperplano tem as seguintes propriedades:

- $\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0, \text{se } y_i = 1$
- $\beta_0 + \beta_1 x_1 + \beta_2 x_2 < 0, \text{se } y_i = -1$

- O hiperplano pode então ser usado para construir um classificador: o exemplo de teste recebe a classe dependendo de que lado do hiperplano estiver localizado.

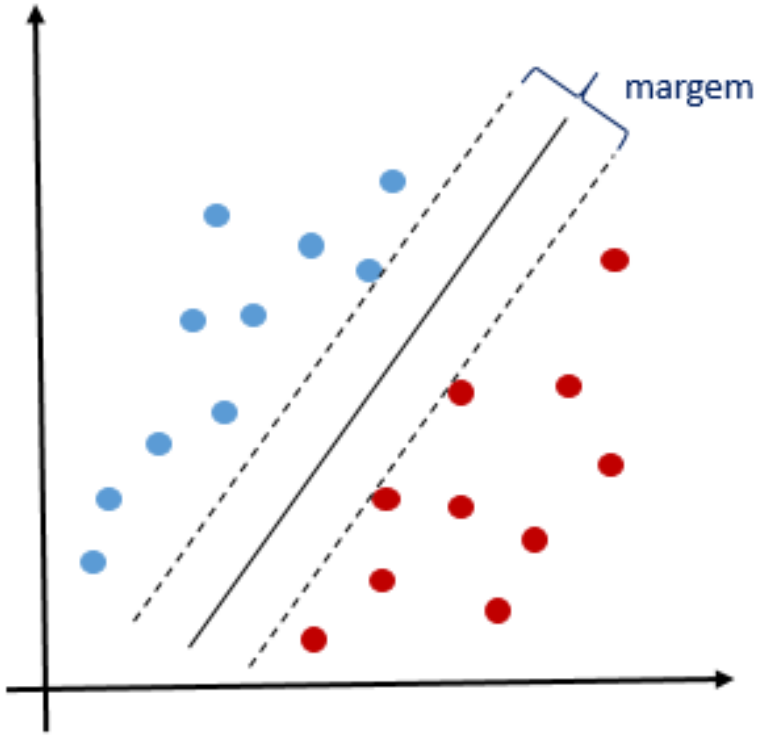
SVM: Exemplo

- Porém, no exemplo, **infinitas** retas (ou hiperplanos) dividem corretamente o conjunto de treinamento em duas classes.
- **É preciso definir qual delas usar…**
- O SVM deve realizar um processo de **escolha** da reta separadora, dentre o conjunto infinito de retas possíveis.



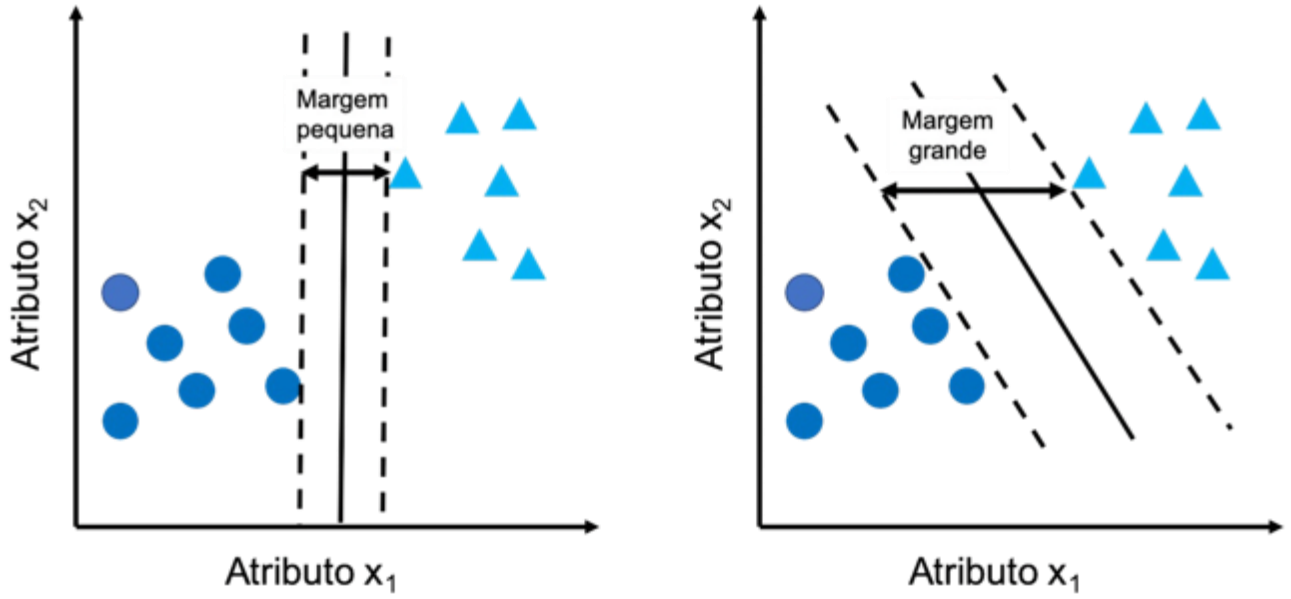
SVM: Margem e Vetor de suporte

- Na Figura, é apresentado apenas um **Classificador Linear** (reta sólida) e duas retas paralelas ao classificador.
- Cada uma delas é movida a partir da posição da reta sólida, e determina quando a reta paralela intercepta o **primeiro ponto** do conjunto de dados.
- A **margem** é a distância construída entre estas duas retas paralelas pontilhadas.



SVM: Margem e Vetor de suporte

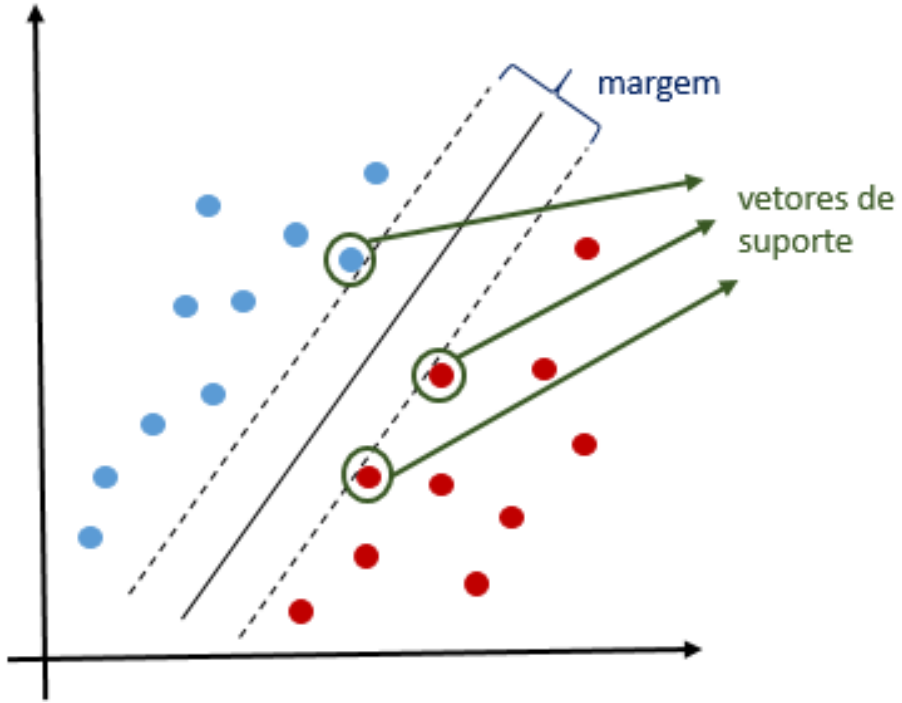
- Assim como existem infinitas retas que separam os pontos em duas classes, há diversos **tamanhos de margem possíveis** dependendo da reta escolhida como classificador.



SVM: Margem e Vetor de suporte

Software Science Lab

- O classificador associado ao valor máximo de margem é denominado **Classificador Linear de Margem Máxima**.
- Os pontos do conjunto de dados de treinamento que são interceptados pelas linhas da margem são denominados **vetores de suporte** e são os pontos mais difíceis de classificar.



SVM: Otimização

Software Science Lab

- O SVM realiza um processo de **otimização**, por meio do qual são determinados os parâmetros do classificador linear (no exemplo, β_0, β_1 e β_2) usando técnicas de programação quadrática

- O objetivo é determinar os valores dos parâmetros que produzam o **valor máximo** para o comprimento da margem

SVM: Otimização

Software Science Lab

- A reta correspondente ao classificador linear com margem de comprimento máximo é dita **ótima** porque se ela for deslocada em alguma das duas direções das retas perpendiculares a ele, a probabilidade é menor de haver um erro de classificação

- Esta reta é a **mais segura possível** com relação a eventuais erros de classificação: quanto maior a distância de *x* para o hiperplano, maior a confiança sobre a classe a que *x* pertence.

SVM: Soft-Margin

Software Science Lab

- Na prática, os dados reais **não costumam ser perfeitamente separáveis** por um hiperplano
- Além disso, o Hiperplano Margem Máxima é extremamente **sensível a mudanças** em uma única observação, o que sugere que pode ocorrer *overfitting* nos dados de treino
- Podemos querer considerar um classificador baseado em um hiperplano que não separe **perfeitamente** as duas classes, com o objetivo de aumentar a robustez nas observações individuais e melhorar a classificação na **maioria** das observações de treino
  - Pode valer a pena classificar erroneamente **algumas** observações de treino a fim de **melhorar** a classificação nas observações restantes.

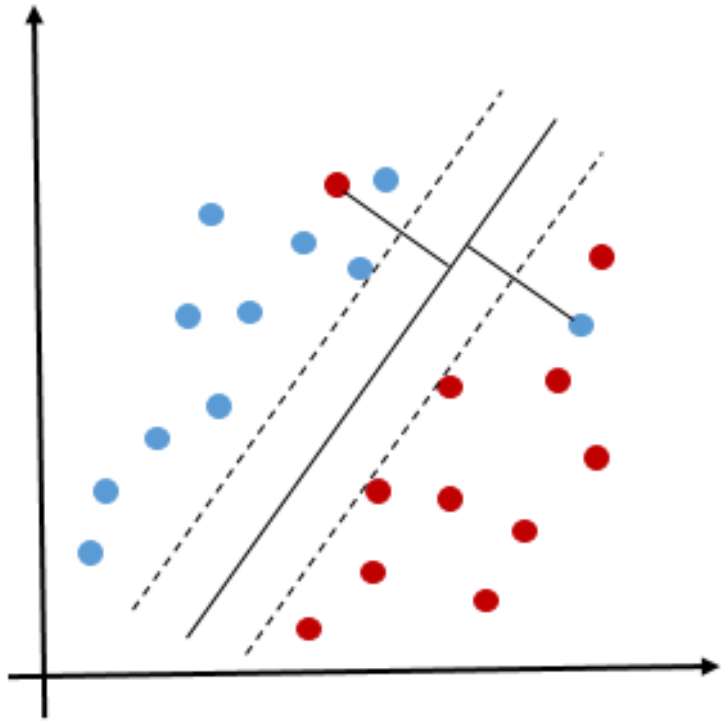
SVM: Soft-Margin

Software Science Lab

- O **classificador soft-margin** permite que algumas observações do conjunto de treino **violem** a linha de separação:
  - O hiperplano é escolhido para separar corretamente **a maior parte** das observações em duas classes, mas pode classificar incorretamente algumas observações
  - Um conjunto adicional de coeficientes é introduzido na otimização para permitir uma “folga” à margem, mas aumentam a complexidade do modelo

SVM: Soft-Margin

- Neste exemplo, o classificador iria cometer **erros** na classificação para os dois pontos destacados, que podem ser considerados ruído.



SVM: Soft-Margin

- O parâmetro de custo **C** define a "rigidez" da margem e controla o *trade-off* entre o **tamanho da margem** e o **erro do classificador**
- Quanto maior o valor de C, mais pontos podem ficar dentro da margem e maior o erro de classificação, mas menor é a chance de *overfitting*
- Se **C = 0**, a margem é rígida e temos o **Classificador de Margem Máxima**, que na prática, não produz bons resultados

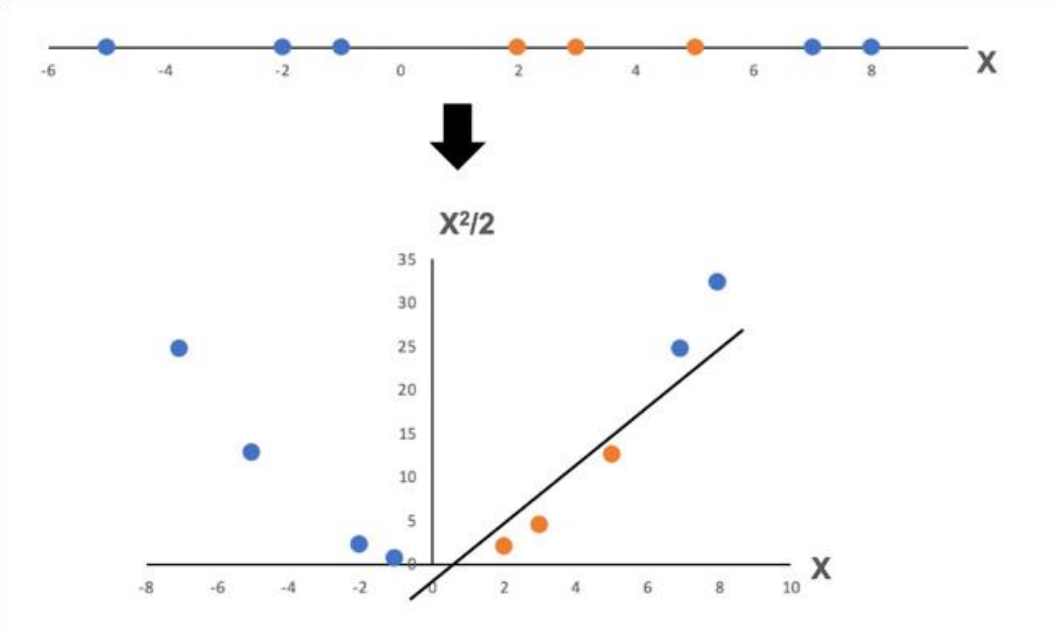
SVM: Número de Vetores de Suporte

Software Science Lab

- O **número total de vetores de suporte** depende da quantidade de **folga** permitida nas margens e da **distribuição** dos dados
  - Quanto **maior** a folga permitida, **maior** o número de vetores de suporte e **mais lenta** será a classificação dos dados de teste, pois a complexidade computacional do SVM está relacionada com o número de vetores de suporte

SVM: Kernel Trick

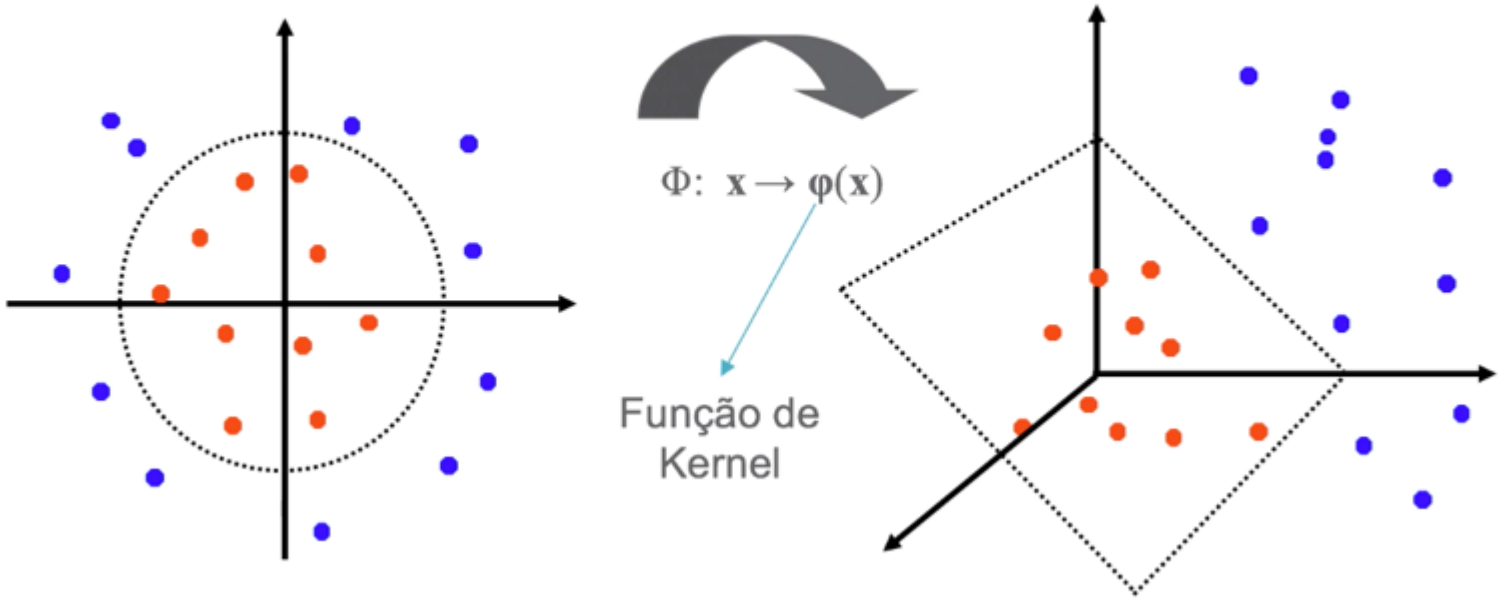
- Na prática, o SVM é implementado usando funções **kernel**
- Para um conjunto de dados que não é linearmente separável, **funções kernel** são utilizadas para mapear o conjunto de dados para um espaço de dimensão **maior** que a original e o classificador é ajustado neste novo espaço
- O SVM é, na verdade, a combinação do **classificador linear** com um **kernel não-linear**



SVM: Kernel Trick

- Embora estejamos aumentando a dimensão do espaço, a **complexidade diminui,** pois antes a classificação só era possível usando classificadores não lineares, e agora pode ser feita apenas com um hiperplano, que é uma superfície de decisão linear

$$\phi(x_1, x_2) \rightarrow (z_1, z_2, z_3) = (x_1^2, \sqrt{2x_1x_2}, x_2^2)$$



Fonte: *Data Science Academy*

SVM: Resumo

- Dado um conjunto de treinamento, deseja-se realizar uma estimativa da **margem de comprimento máximo**
  - Os **vetores de suporte** são os pontos (do conjunto de dados) que delimitam a margem e determinam a localização do classificador linear: são os pontos mais próximos do hiperplano do classificador
  - O **comprimento da margem** é igual à distância entre as duas retas paralelas ao classificador linear e que forma a fronteira de classificação
  - Por construção, todos os vetores de suporte possuem a **mesma distância** em relação a reta do classificador linear (a metade do comprimento da margem)

Software Science Lab

Problemas de Regressão

Engenharia de Software para Ciência de Dados

Problema de Regressão: Definição

Software Science Lab

- Dado um conjunto de *n* **padrões**, em que cada padrão é composto por informação de **variáveis explicativas** (X) e de uma **variável resposta contínua/discreta** (y)

- **Objetivo:** construir um modelo de regressão que dado um novo padrão estime o **valor mais esperado** para a variável resposta

Padrão	Explicativas			Resposta
$\mathbf{x}_1$	x_{11}	...	x_{1J}	y_1
$\mathbf{x}_2$	x_{21}	...	x_{2J}	y_2
$\mathbf{x}_3$	x_{31}	...	x_{3J}	y_3
...	...	...	...	...
$\mathbf{x}_i$	x_{i1}	...	x_{iJ}	y_i
...	...	...	...	...
$\mathbf{x}_n$	x_{n1}	...	x_{nJ}	y_n

Classificação x Regressão

- **Classificação:** Resultado categórico
  - **Conceder ou não** crédito para um cliente?
- **Regressão:** Resultado numérico (contínuo ou discreto)
  - Conceder **qual valor** de crédito para um cliente?

Tarefas como:

- Preparação da base de dados;
- Separação em conjuntos de treino e teste;
- Definição dos critérios de parada do algoritmo;
- Treinamento e teste;

são feitas de forma **equivalente**.

Regressão

- Também chamada de **Estimação**
- Aprendizagem **Supervisionada** a partir de **dados históricos**
- Diferença na **avaliação de saída**: em vez da acurácia, estima-se a **distância** ou o **erro** entre a saída do estimador e a saída desejada: o processo de treinamento do estimador tem por objetivo **corrigir o erro observado**, buscando minimizar um critério de maneira que os valores estimados estejam próximos dos valores reais no sentido estatístico
- A Classificação pode ser vista como um **caso particular** da Regressão

Métricas de Desempenho para Regressão

Software Science Lab

- **MSE** (mean squared error, ou erro quadrático médio): quanto **mais próximo de 0**, melhor o modelo. Fornece uma ideia da **magnitude** do erro, mas nenhuma ideia da direção.

$$MSE = \frac{1}{n}\sum_{j=1}^{n} e_j^2$$

- **RMSE** (*root mean squared error*, ou raiz do erro quadrático médio): quanto **menor**, melhor o modelo. A raiz quadrada faz com que seja medido na mesma unidade do erro, facilitando a interpretação.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n} e_j^2}$$

- **Coeficiente de Determinação (R^2)**: (*R square*): explica o quanto a variável *target* pode ser explicada pelo modelo (o quanto y é explicado por X). Varia entre 0 e 1 e quanto **mais próximo de 1**, melhor o ajuste do modelo:

$$R^2 = 1 - \frac{SQE}{\sum_{j=1}^{n}(y_j - \bar{y})^2}$$

- Sendo **SQE** a soma dos erros quadráticos (*sum of squared errors*):

$$SQE = \sum_{j=1}^{n} e_j^2$$

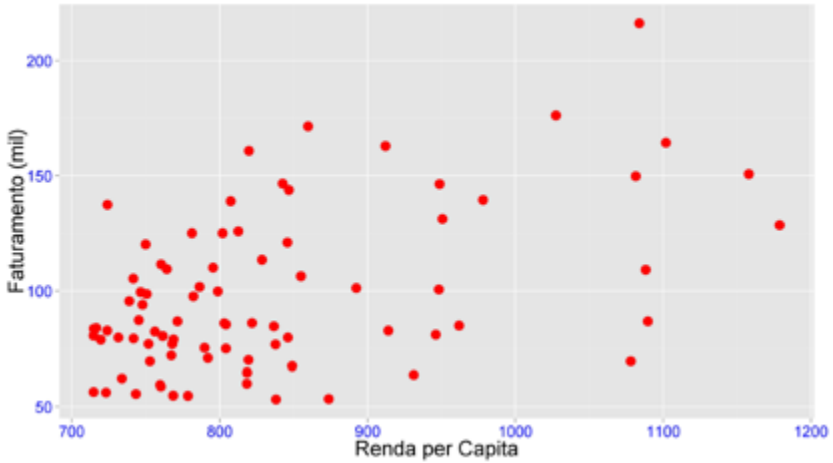
$$Sendo:\ e = y_j - \hat{y}_j$$

Regressão Linear

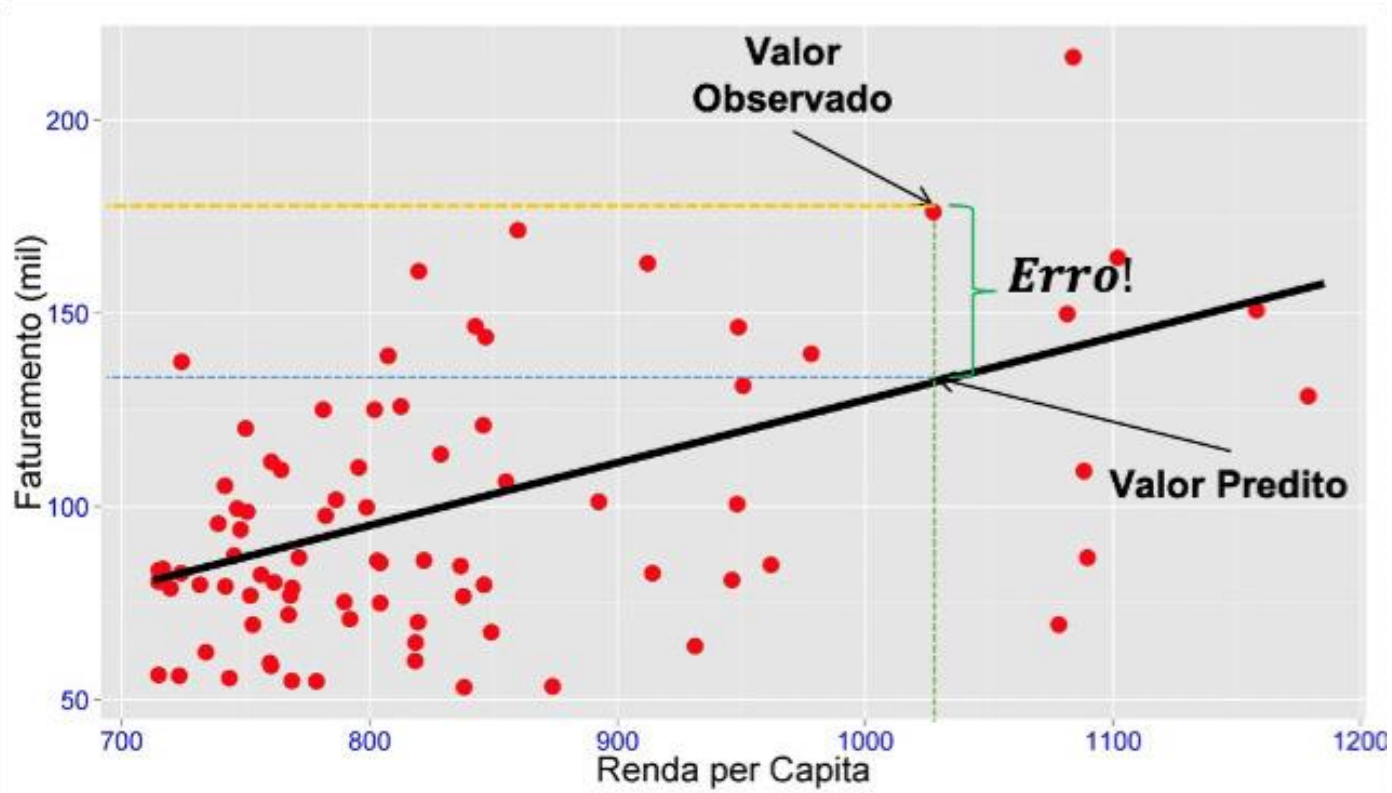
- **Problema:** determinar o faturamento de uma loja com base na renda/hab da sua localidade.

- **Observação trivial:** Quanto maior a Renda per Capita do Bairro, maior o Faturamento no Bairro. Mas como formular esta relação matematicamente?

Bairro	Renda/Hab.	Faturamento
A	1500	105.000
H	2400	180.000
...	...	...
L	3400	150.000



Regressão Linear



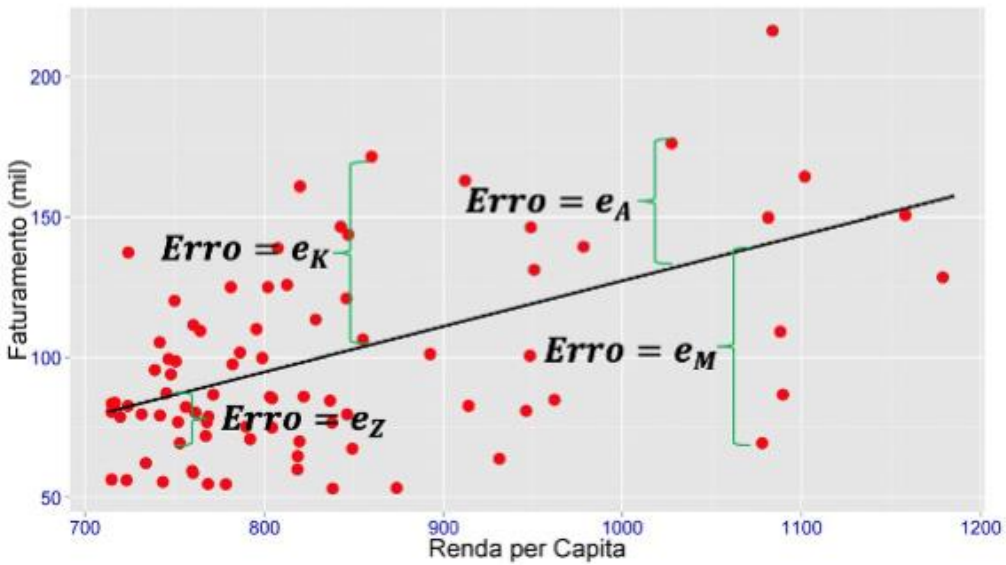
Esta solução é dita **ótima** porque passa mais perto dos pontos (considerando a distância euclidiana)

Regressão Linear

- Para cada escolha dos parâmetros β_o e β_1 na equação

$$\widehat{Fat}_{Bairro} = \beta_0 + \beta_1 \times RendaPerCapita_{Bairro}$$

- é possível calcular os erros (ou desvios) dessa escolha



$$\widehat{Fat}_{Bairro} = -24{,}49 + 0{,}15 \times RendaPerCapita_{Bairro}$$

Regressão Linear

- Porém, observe que se somarmos os erros para calcular o erro total do modelo, pelos erros individuais serem positivos e negativos, eles **se anulam**

Bairro	Fat	$\hat{Fat}$	$e = Fat - \hat{Fat}$
A	180	130	50
K	175	115	60
M	65	120	-55
...	...	...	...
Z	70	82	-12

- Assim, a melhor prática é trabalhar com a **magnitude do erro** (erro ao quadrado, por exemplo):

$$\sum_{i \in Bairro} e_i^2 = (Fat_i - \hat{Fat}_i)^2 = 50^2 + 60^2 + (-55)^2 + ... + (-12)^2 = 230$$

Regressão Linear

Software Science Lab

- **Moral da história:** a Regressão Linear consiste em escolher β_o e β_1 para construir uma reta que minimize a soma dos quadrados dos erros (SQE, ou outra métrica de regressão):

$$SQE = \sum_{i \in Bairro}^{n} e_i^2$$

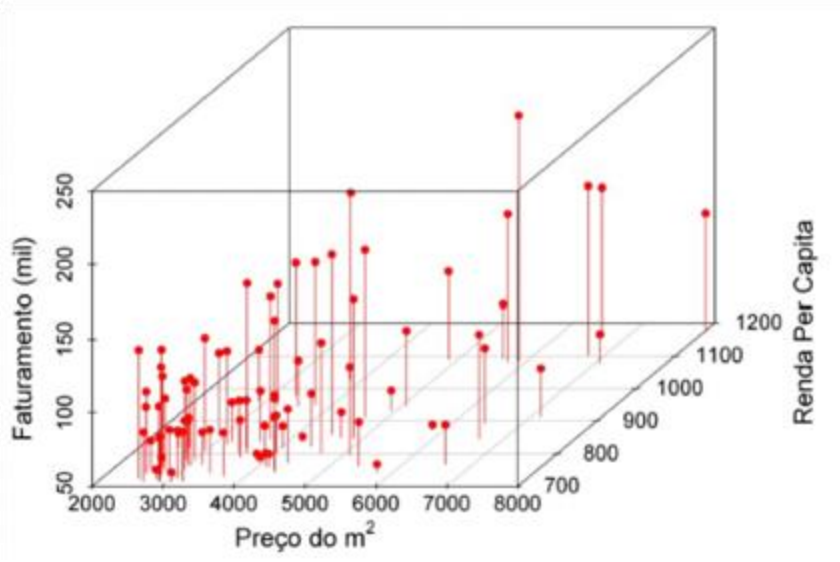
- Neste exemplo consideramos apenas a relação entre Faturamento e Renda Per Capita. **Em problemas reais, dificilmente haverá uma única variável x capaz de prever o y!**

Regressão Linear Múltipla

- Se quiséssemos adicionar uma nova variável (como por exemplo preço do m^2), teríamos o modelo de **regressão linear múltipla**:

$$\widehat{Fat}_i = \beta_0 + \beta_1 \times RendaPerCapita_i + \beta_2 \times PrecoM^2$$

- Basta adicionar as demais variáveis preditoras (x_1, x_2, ..., x_n) e seus coeficientes correspondentes.
- A solução seria construir um **plano** (em vez de uma reta) que melhor se ajuste aos pontos



Regressão Linear: Resumo

Software Science Lab

- A Regressão Linear modela a **relação** entre uma variável de **resposta** (y) e os **preditores** (X).
- Assume-se um relacionamento **linear** entre X e y, ou seja, que y pode ser calculado através de uma **combinação linear** de X.
  - Apenas um x: regressão linear simples
  - Mais de um x: regressão linear múltipla

Regressão Linear: Resumo

Software Science Lab

- Corresponde ao problema de estimar uma **função** a partir de pares entrada-saída.
  - **Simples:** $y = \beta_o + \beta_1 x$, sendo β_0 e β_1 os coeficientes de regressão (especificam o interceptor do eixo y e a inclinação da reta)
  - **Múltipla:** a equação deve ser estendida para equação de plano/hiperplano: $y = \beta_o + \beta_1 x_1 + \beta_2 x_2 + \ldots$
- A solução da tarefa de regressão consiste em **encontrar valores** para os **coeficientes de regressão** de forma que a reta (ou plano/hiperplano) se **ajuste** aos valores assumidos pelas variáveis no conjunto de dados

Regressão Linear: Avaliação

- A saída de um estimador é um valor numérico contínuo que deve ser **o mais próximo possível** do valor desejado, e a diferença entre esses valores fornece uma **medida de erro** de estimação do algoritmo.
  - Seja $d_j = 1, ..., n$ a resposta desejada para o objeto j e y_j a resposta predita do algoritmo, obtida a partir da entrada x_j. Então, $e_j = d_j - y_j$ é o **erro** observado na saída do sistema para o objeto j.

Regressão Linear: Avaliação

Software Science Lab

- O processo de treinamento do estimador tem por objetivo **corrigir** este erro observado e, para tal, busca **minimizar** um critério (função objetivo) baseado em e_j, de maneira que os valores de y_j estejam próximos dos de d_j no sentido estatístico.
- Se a equação de regressão aproxima suficiente bem os dados de treinamento, então ela pode ser usada para **estimar** o valor de uma variável (y) a partir do valor da outra variável (x).
- **Resumindo:** a regressão linear procura pelos coeficientes da reta que minimizam a distância dos objetos à reta.
- *OBS: apesar da sua simplicidade, modelos lineares são surpreendentemente competitivos em relação a modelos não lineares.*

Regressão Logística

Software Science Lab

- Não se confunda! É um algoritmo de **classificação** e não de regressão.

- Usado para estimar valores discretos (e.g., valores binários como 0/1, sim/não, verdadeiro/falso) com base em um conjunto de variáveis independentes.

- Calcula a **probabilidade de ocorrência** de um evento, ajustando os dados a uma função **logit**.

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = \log(p) - \log(1-p).$$

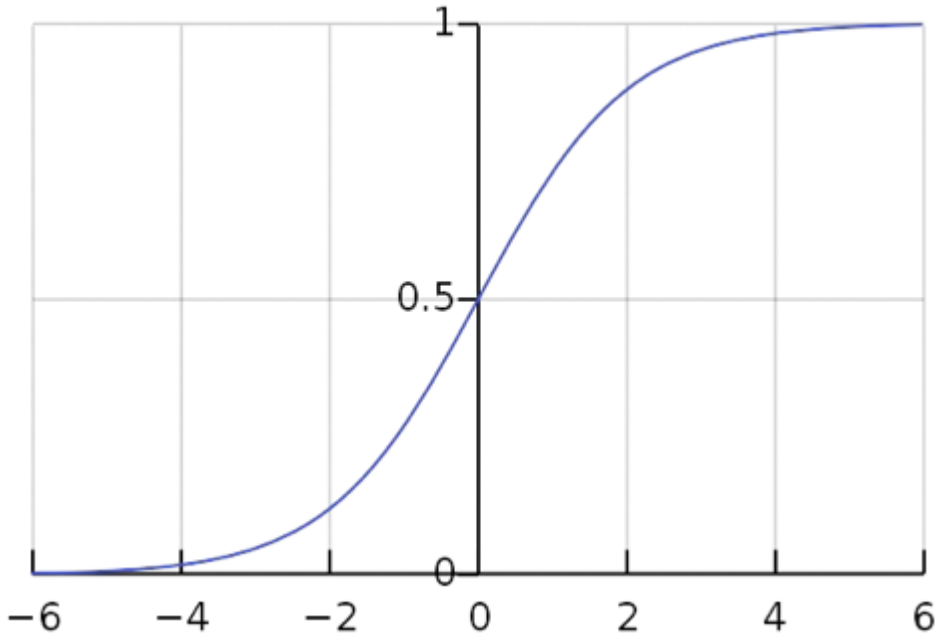
- Como prevê a probabilidade, seus valores de saída estão **entre 0 e 1**.

Regressão Logística

- Utiliza função **logística**, também chamada de **sigmóide**: uma curva em forma de S que pode mapear qualquer número em um intervalo entre 0 e 1.

$$f(x) = \frac{1}{1 + e^{-x}}$$

- onde f(x) é a saída prevista.



Transformação logística entre -6 e 6 (Fonte: Wikipedia)

Regressão Logística

Software Science Lab

- De forma similar à regressão linear, usa uma **equação** como representação.

- Os valores de entrada (X) são combinados linearmente usando **coeficientes** para predizer um valor de saída (y):

$$\hat{y} = \frac{1}{1 + e^{-(b0 + b1 \times x1)}}$$

OBS: Cada característica de entrada é associada a um coeficiente.

- O valor de saída é **arredondado** para um valor binário (0 ou 1) e mapeado na classe correspondente.

Algoritmos também usados para Regressão

Software Science Lab

- **Árvore de Regressão:** a homogeneidade é medida por estatísticas como variância, desvio padrão. A predição é a média dos valores dos exemplos de cada folha.
- **KNN para Regressão:** a saída é a média aritmética dos k vizinhos mais próximos.
- **SVR (SVM para regressão):** mapeia os dados em um espaço multidimensional usando funções kernel e realiza uma regressão linear neste espaço transformado, considerando apenas os pontos que estão dentro da margem. O melhor modelo é o hiperplano que possui o número máximo de pontos.

Software Science Lab

Ensembles

Engenharia de Software para Ciência de Dados

Ensembles

- Ajudam a melhorar os resultados, combinando vários modelos
- Geralmente têm melhor desempenho preditivo em comparação com um único modelo, sendo os primeiros colocados em muitas competições de aprendizado de máquina de prestígio
- **São meta-algoritmos que combinam várias técnicas de aprendizado de máquina em um modelo preditivo**

Ensembles Sequenciais e Paralelos

Software Science Lab

- Métodos **sequenciais**, em que os modelos base são gerados sequencialmente
  - Exploram a **dependência** entre os modelos de base
  - O desempenho geral pode ser aprimorado ponderando os exemplos previamente rotulados incorretamente com maior peso
- Métodos **paralelos**, em que os modelos base são gerados em paralelo
  - Exploram a **independência** entre os modelos base

Ensembles Homogêneos e Heterogêneos

Software Science Lab

- A maioria dos métodos ensemble usa **um único algoritmo básico**, produzindo **ensembles homogêneos**, mas também há métodos que usam algoritmos diferentes (**ensembles heterogêneos**)
- Para que os métodos de ensemble sejam **melhores do que qualquer um de seus modelos individuais**, os modelos base precisam de **boa precisão** e ser **diversificados**

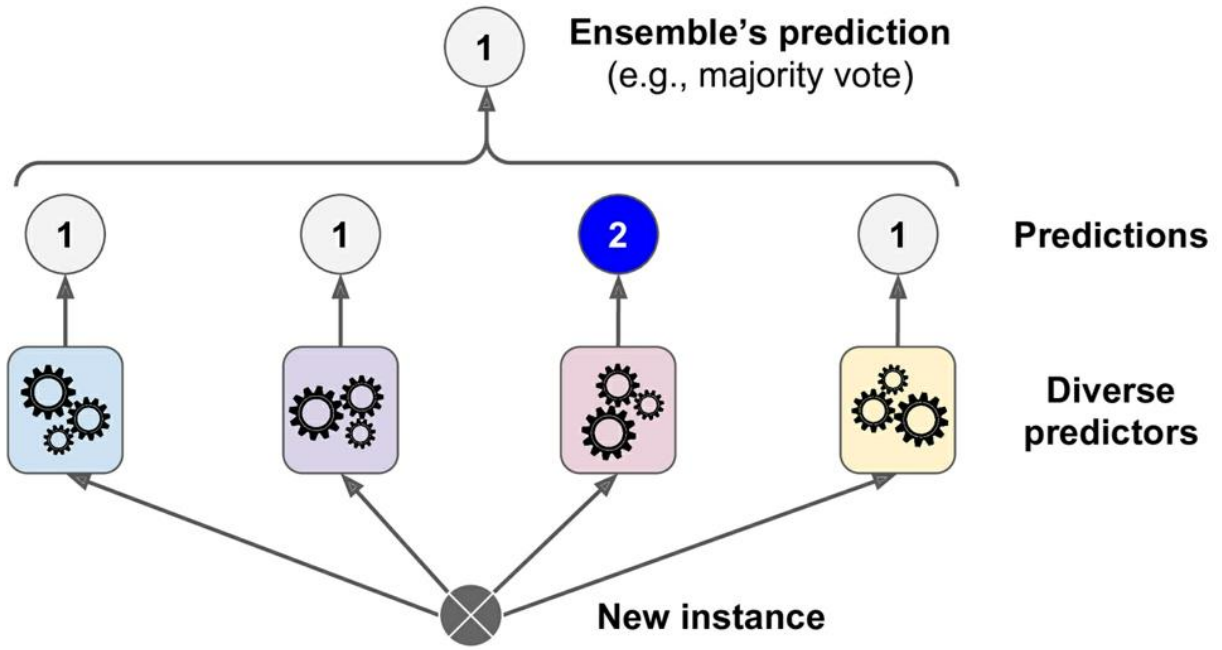
Métodos Ensemble mais Populares

Software Science Lab

- **Voting:** constrói vários modelos (geralmente de **tipos diferentes**) e calcula estatísticas simples (ex: média, moda) para **combinar** as predições
- **Bagging:** constrói vários modelos (geralmente do **mesmo tipo**) a partir de diferentes **sub-amostras** do conjunto de dados de treinamento
- **Boosting:** constrói vários modelos (geralmente do **mesmo tipo**) e cada um deles aprende a **corrigir** os erros de predição de um modelo anterior na sequência de modelos

Voting

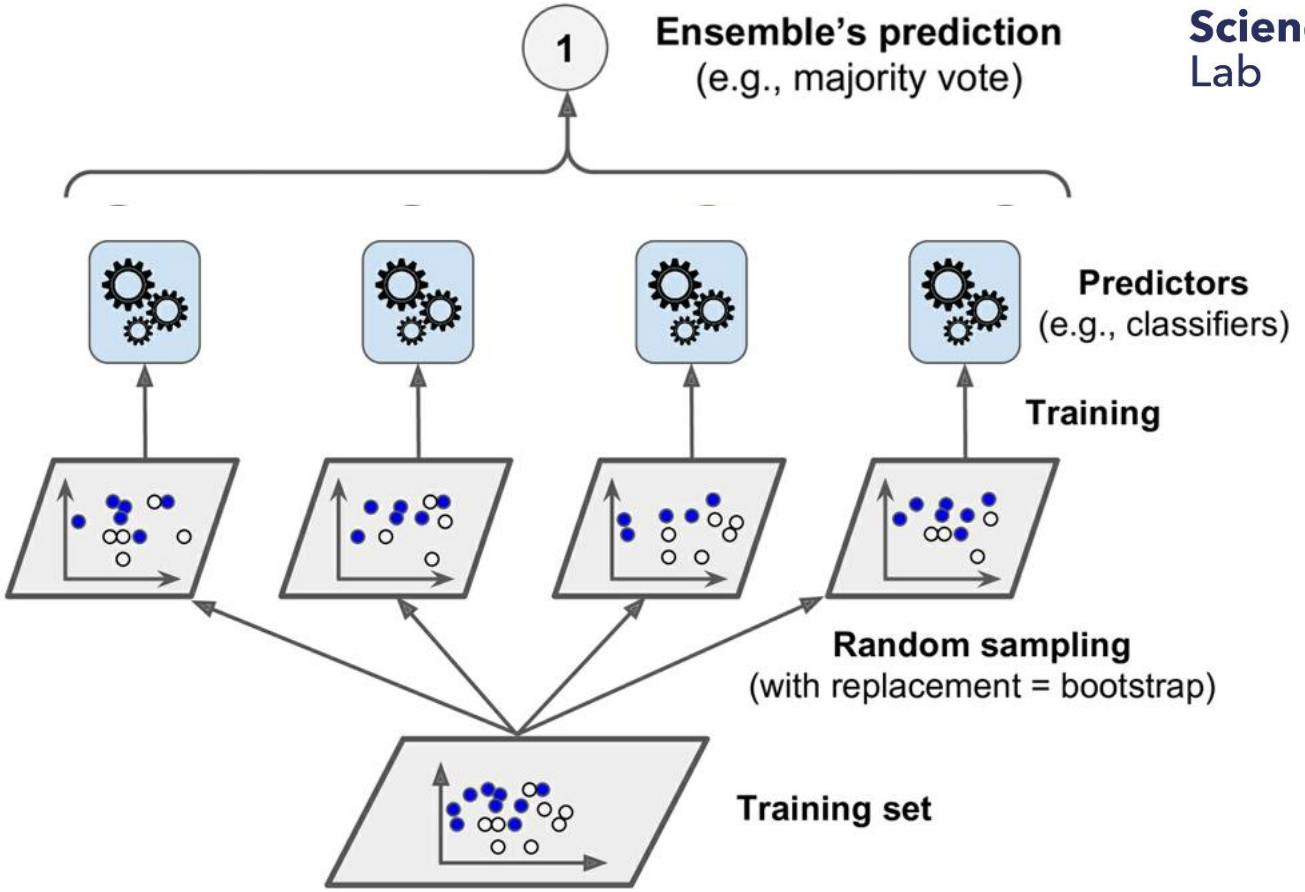
- A classe predita é **a mais votada** entre os classificadores
- Geralmente tem melhor acurácia que o melhor classificador do ensemble
- Mesmo que todos os classificadores sejam “fracos” (um pouco melhores que escolha aleatória), o ensemble ainda pode ser um classificador “forte” (se os classificadores forem numerosos e diversos o suficiente



Fonte: *Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn and TensorFlow . O'Reilly Media.*

Bagging

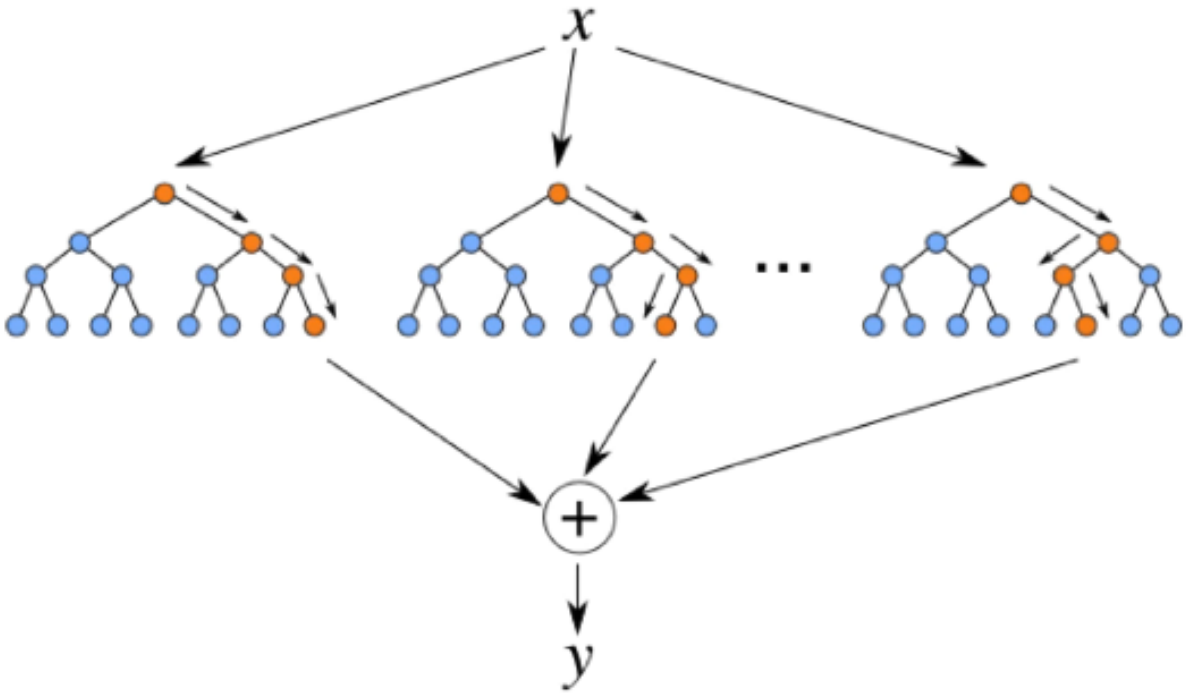
- Usa o mesmo algoritmo para cada modelo, treinando cada um deles em um **subconjunto aleatório** do conjunto de treino.
- Após o treinamento de cada modelo, a predição de uma nova instância é a **agregação** das predições individuais.
- Exemplos de algoritmos: **Random Forest** e **Extra Trees**



Fonte: *Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn and TensorFlow . O'Reilly Media.*

Exemplo: Random Forests (Florestas Aleatórias)

- Melhoria em relação ao Bagging com árvores de decisão.
- Para fazer a divisão em cada nó procura o melhor atributo dentre um subconjunto aleatório dos atributos.
- Para esta amostra de atributos, o algoritmo busca o ponto de corte ótimo para cada atributo.
- Isso resulta em uma **maior diversidade de árvores**, geralmente produzindo um ensemble melhor.

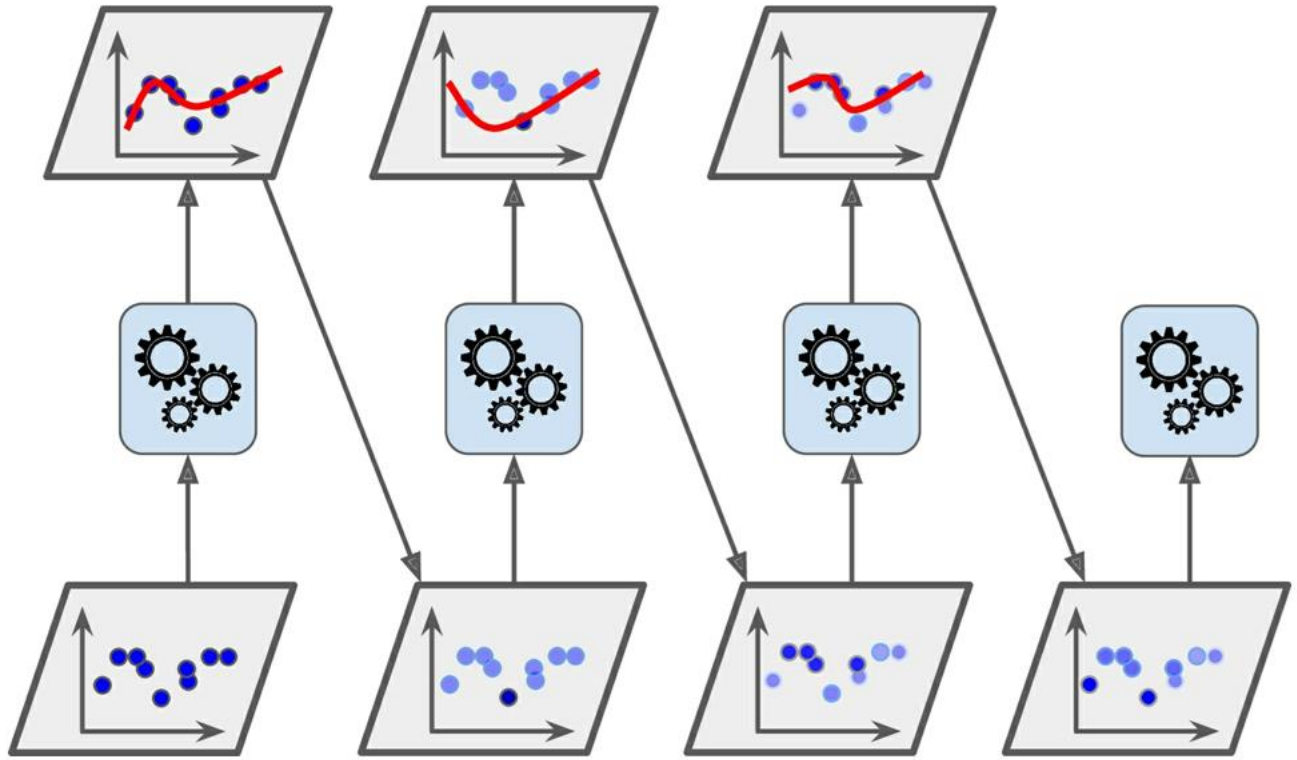


Extra Trees

- Também conhecido como ***Extremely Randomized Trees*** (Árvores Extremamente Aleatórias).
- Ao contrário do *Bagging* e do *Random Forest* (que constroem cada árvore de decisão a partir de uma amostra *bootstrap* do conjunto de dados de treinamento), o *Extra Trees* ajusta cada árvore de decisão em **todo o conjunto de dados de treinamento**.
- As divisões são realizadas **aleatoriamente:**
  - A divisão a realizar é a melhor divisão usando pontos de corte **aleatórios** em um subconjunto de atributos selecionado **aleatoriamente** para a árvore corrente.
- Devido a aleatoriedade, é um modelo **mais rápido** computacionalmente do que as Random Forests, pois não é necessário avaliar onde realizar as divisões.

Boosting

- Cria uma **sequência** de modelos na qual um modelo tenta **corrigir** os erros do modelo anterior
- Busca converter modelos **fracos** em modelos **fortes**
- Geralmente usado com árvores de decisão
- Exemplos de algoritmos: **Adaboost** e **Gradient Boosting**



Fonte: *Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn and TensorFlow . O'Reilly Media.*

Vamos praticar?

- ML Classificação:

https://colab.research.google.com/drive/12Un8HR-VEXQAPbefwWhJrChGaDd5LAVx?usp=sharing

- ML Regressão:

https://colab.research.google.com/drive/11SVgYiD8Q7Jn-fQjXQvD9UwqVFs8eU7v?usp=sharing

- ML Ensembles:

https://colab.research.google.com/drive/10VGEu6qjr0gnVZ5QCtllWRBUlIlqqPre?usp=sharing

- ML Projeto Completo de Classificação – Diabetes

https://colab.research.google.com/drive/1rEzxfxvDqqHmXVJjQM6LXi-ZhShMhWsx?usp=sharing

Software Science Lab

Seleção de Atributos

Engenharia de Software para Ciência de Dados

Seleção de Atributos (*Feature Selection*)

- Os atributos (X) dos dados usados para treinamento dos modelos de ML têm uma enorme influência no seu desempenho.
- Atributos irrelevantes ou parcialmente relevantes podem afetar negativamente o desempenho do modelo, especialmente algoritmos lineares como regressão linear e logística.
- O processo de seleção de atributos realiza a seleção automática dos atributos (X) que mais contribuem para a saída (y).

Seleção de atributos (*Feature Selection*)

Vantagens:

- **Redução de *Overfitting*:** dados menos redundantes = menor risco de tomar decisões com base em ruído.
- **Melhoria da precisão:** dados menos "enganosos" = melhor a precisão da modelagem.
- **Redução do tempo de treinamento:** menos dados = treinamento mais rápido.
- **Melhoria da interpretabilidade do modelo:** melhor entendimento do processo que gerou os dados.

Seleção de Atributos X Redução de Dimensionalidade

Software Science Lab

- Ambos os métodos tendem a reduzir o número de atributos no conjunto de dados.
- Um método de redução de dimensionalidade cria novas combinações dos atributos (*feature transformation*), enquanto os métodos de seleção de atributos excluem atributos do dataset.
- Exemplos de métodos de redução de dimensionalidade: PCA (*Principal Component Analysis*) e SVD (*Singular Value Decomposition*).

Algumas Técnicas de Seleção de Atributos

Software Science Lab

Seleção Univariada

- Testes estatísticos são usados para selecionar os atributos mais correlacionados com a variável de saída.

Eliminação Recursiva de atributos

- Remove recursivamente os atributos e vai construindo um modelo com os que permanecem, e usa a acurácia do modelo para identificar quais atributos mais contribuem para a predição da variável de saída.

Importância de atributos

- Técnicas como Random Forest ou Extra Trees podem ser usadas para estimar a importância dos atributos.

Vamos praticar?

- Feature Selection:

https://colab.research.google.com/drive/1oa3A4rE10lZvkAfx6h3ORuRjwj7yWbAd?usp=sharing

Software Science Lab

Princípios de Projeto e Boas Práticas de Codificação no Contexto de Machine Learning

Engenharia de Software para Ciência de Dados

Princípios de Projeto e Boas Práticas de Codificação no Contexto de Machine Learning

Software Science Lab

- Agenda:
  - Reprodutibilidade de Notebooks
  - Aplicação dos Princípios SOLID no contexto de ML
  - Aplicando Análise Estática e Testes

Reprodução e Boas Práticas para Reprodutibilidade de Notebooks

- Na área de ciência de dados é extremamente comum realizar tarefas como análise exploratória, pré-processamento e avaliação de algoritmos de Machine Learning utilizando programação literária interativa (notebooks)
- A forma interativa pode prejudicar a reprodutibilidade de experimentos (Pimentel *et al.*, 2021)
  - Em torno de 25% dos notebooks disponíveis no GitHub permitem a execução de todas as suas células
  - Em torno de 10% produziram resultados próximos aos armazenados nos notebooks

Pimentel, J.F., Murta, L., Braganholo, V. and Freire, J., 2021. **Understanding and improving the quality and reproducibility of Jupyter notebooks.** *Empirical Software Engineering*, *26*(4), pp.1-55.

Reprodução e Boas Práticas para Reprodutibilidade de Notebooks

Software Science Lab

- Boas práticas para notebooks (Pimentel *et al.*, 2021):
  - Usar títulos descritivos com um conjunto restrito de caracteres (A-Z a-z 0-9 . _) e células *Markdown* com títulos mais detalhados no corpo
  - Prestar atenção ao final do notebook e verificar se as células do final foram executadas e se não cabe nenhuma célula de *Markdown* para concluir a análise
  - Abstrair funções, classes e módulos que possam ser importados por diversos notebooks e testados externamente
  - Declarar dependências em arquivos apropriados (*requirements.txt*) e fixar a versão dos módulos

Pimentel, J.F., Murta, L., Braganholo, V. and Freire, J., 2021. **Understanding and improving the quality and reproducibility of Jupyter notebooks.** *Empirical Software Engineering*, *26*(4), pp.1-55.

Requirements.txt

- Arquivos *requirements.txt* especificam quais pacotes (e qual versão) são requeridos para executar um projeto
  - Podem ser usados para instalar diversos pacotes de uma só vez

```
##### Core scientific packages
matplotlib==3.5.0
numpy==1.21.1
pandas==1.3.0
scipy==1.7.3

##### Machine Learning
packages
scikit-learn==1.0.1
```


```
!pip install -r requirements.txt
```

https://www.idkrtm.com/what-is-the-python-requirements-txt/

Reprodução e Boas Práticas para Reprodutibilidade de Notebooks

Software Science Lab

- Boas práticas para notebooks (Pimentel *et al.*, 2021) (cont.):
  - Usar um ambiente limpo para testar dependências e verificar se todas estão declaradas corretamente
  - Colocar importações no início do notebook
  - Usar caminhos relativos para acessar dados no repositório
  - Re-executar notebooks do início ao fim antes de enviá-los para algum repositório

Pimentel, J.F., Murta, L., Braganholo, V. and Freire, J., 2021. **Understanding and improving the quality and reproducibility of Jupyter notebooks.** *Empirical Software Engineering*, *26*(4), pp.1-55.

Boas Práticas de Programação OO para ML

Software Science Lab

- Princípios como SOLID e princípios abstratos como baixo acoplamento, alta coesão, podem e devem ser aplicados neste contexto

Os exemplos a seguir foram baseados em: https://github.com/musikalkemist/solidforml/tree/main/solidforml

Princípios SOLID

Software Science Lab

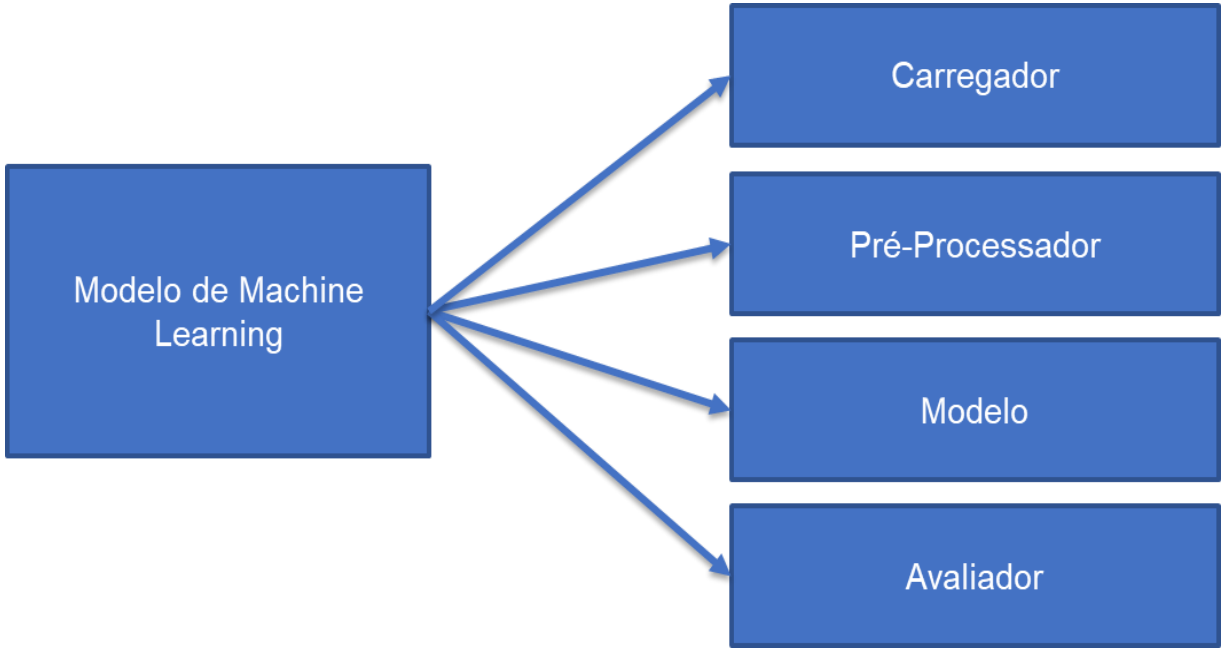
- Buscam ajudar a realizar um bom projeto orientado a objetos para que um software tenha código flexível, escalável, sustentável e reutilizável. SOLID é um acrônimo para cinco princípios:
  - ***S****ingle Responsibility Principle* (SPR) - Princípio de Responsabilidade Única
  - ***O****pen/Closed Principle* (OCP) - Princípio Aberto / Fechado
  - ***L****iskov Substitution Principle* (LSP) - Princípio da Substituição de Liskov
  - ***I****nterface Segregation Principle* (ISP) - Princípio de Segregação de Interface
  - ***D****ependency Inversion Principle* (DIP) - Princípio da Inversão de Dependência

S – Single Responsibility Principle

Software Science Lab

Uma classe não deve ter mais de um motivo para ser alterada

Uma classe deve ser especializada em um **único assunto** e possuir apenas **uma responsabilidade** dentro do software



S – Single Responsibility Principle

Software Science Lab

```python
class Carregador:

    def carregar_dados(self, url: str, atributos: list):
        """ Carrega e retorna um DataFrame. Há diversos parâmetros
        no read_csv que poderiam ser utilizados para dar opções
        adicionais.
        """
        return pd.read_csv(url, names=atributos)
```

```python
class Modelo:

    def treinar_SVM(self, X_train, Y_train):
        """ Cria e treina um modelo SVM. Poderia ter um Grid Search
        com cross_validation para escolher os melhores hiperparâmetros, etc.
        """
        modelo = SVC()
        modelo.fit(X_train, Y_train)
        return modelo
```

```python
class PreProcessador:

    def pre_processar(self, dataset, percentual_teste, seed=7):
        """ Cuida de todo o pré-processamento. """
        # limpeza dos dados e eliminação de outliers

        # feature selection

        # divisão em treino e teste
        X_train, X_test, Y_train, Y_test = self.__preparar_holdout(dataset,
                                                                   percentual_teste,
                                                                   seed)
        # normalização/padronização

        return (X_train, X_test, Y_train, Y_test)

    def __preparar_holdout(self, dataset, percentual_teste, seed):
        """ Divide os dados em treino e teste usando o método holdout.
        Assume que a variável target está na última coluna.
        O parâmetro test_size é o percentual de dados de teste.
        """
        dados = dataset.values
        X = dados[:, 0:-1]
        Y = dados[:, -1]
        return train_test_split(X, Y, test_size=percentual_teste, random_state=seed)
```

S – Single Responsibility Principle

Software Science Lab

```
class Avaliador:

    def avaliar_acuracia(self, modelo, X_test, Y_test):
        """ Faz uma predição e avalia o modelo. Poderia parametrizar o tipo de
        avaliação, entre outros.
        """
        predicoes = modelo.predict(X_test)
        return accuracy_score(Y_test, predicoes)
```

```
# Instanciação das Classes
carregador = Carregador()
pre_processador = PreProcessador()
modelo = Modelo()
avaliador = Avaliador()
```

```
# Carga
dataset = carregador.carregar_dados(url_dados, atributos)
# Pré-processamento
X_train, X_test, Y_train, Y_test = pre_processador.pre_processar(dataset,
                                                                 percentual_teste)
# Treinamento do modelo
model = modelo.treinar_SVM(X_train, Y_train)
# Impressão do resultado da avaliação
print(avaliador.avaliar_acuracia(model, X_test, Y_test))
```

O – Open Closed Principle

Software Science Lab

Uma classe deve estar aberta para extensão, porém fechada para modificação

Quando novos comportamentos e recursos precisam ser adicionados no software, **devemos estender e não alterar o código fonte original**

Por exemplo, organizar o seu código com uma superclasse genérica e abstrata o suficiente permitirá que o programa seja estendido adicionando novas subclasses sem precisar alterar as classes existentes

O – Open Closed Principle

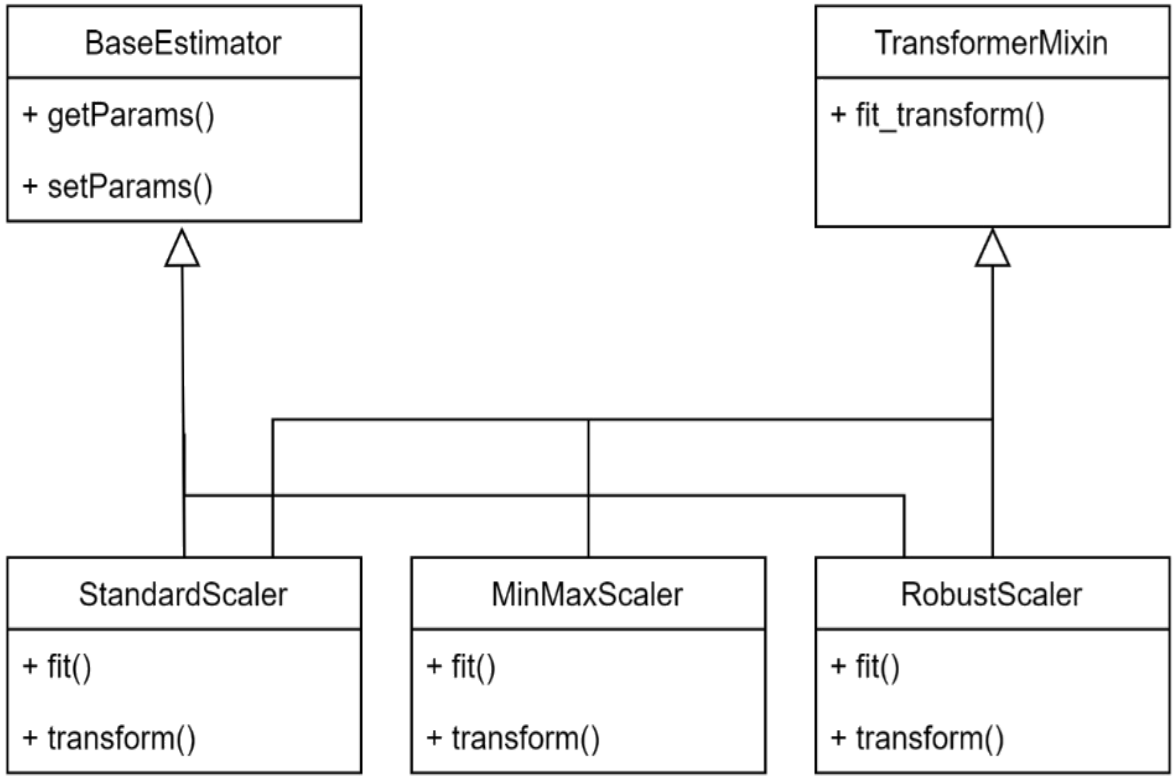
- Diversas aplicações deste princípio podem ser encontrados na própria biblioteca Scikit-learn
- Por exemplo, temos um conjunto de classes para tratar diferentes transformações de dados durante o pré-processamento, tais como normalização (classe *MinMaxScaler*) e padronização (classe *StandardScaler*)
- Se todas as diferentes formas de transformação estivessem implementadas em uma única classe, ela teria que ser modificada toda vez que se quisesse acrescentar uma nova forma de transformação de dados à biblioteca

O – Open Closed Principle

Software Science Lab

- A solução adotada no Scikit-learn foi, de acordo com o princípio OCP, criar uma superclasse mais genérica para permitir que a biblioteca pudesse ser estendida acrescentando novas subclasses, sem precisar alterar as classes existentes
- Essa superclasse mais genérica é a classe *TransformerMixin*
  - Novas classes que buscam realizar transformações sobre dados (assim como a *MinMaxScaler* e a *StandardScaler*) devem herdar dessa classe e implementar os métodos *fit* e *tranform*
- Isso permitirá a todo o restante do Scikit-learn tratar a nova classe de forma genérica como um *TransformerMixin* (uma classe que identifica todos os transformadores de dados)

O – Open Closed Principle



- O princípio do aberto e fechado (OCP) se manifesta neste contexto ao assegurar que novos *Scalers* possam ser criados em classes completamente novas, sem afetar o código das demais
- Basta criar uma nova subclasse que também herde de *TransformerMixin* (e de *BaseEstimator*, como todo estimador do Scikit-learn) e que implemente os métodos *fit* e *transform*
- Para o restante do sistema objetos da nova classe serão utilizados de forma transparente como se fossem objetos da classe *TransformerMixin* e nenhuma outra linha de código terá que ser alterada

L – Liskov Substitution Principle

Software Science Lab

Uma subclasse deve poder ser substituída pela sua superclasse

Parece estar invertido, uma vez que sabemos que se S é uma subclasse de C, então os objetos do tipo C podem ser substituídos pelos objetos de tipo S em tempo de execução. Mas, quando o LSP é seguido, os métodos públicos definidos nas subclasses devem estar também presentes na superclasse, mesmo que de forma abstrata (o que também é conhecido como ***strong behavioral subtyping***).

O LSP encoraja o uso das classes de nível de abstração mais alto, estruturando as heranças do código de forma a ampliar as possibilidades de aplicação do princípio de inversão de dependência (DIP), uma vez que as abstrações (superclasses) terão os métodos necessários para expressar toda a semântica pretendida

L – Liskov Substitution Principle

Software Science Lab

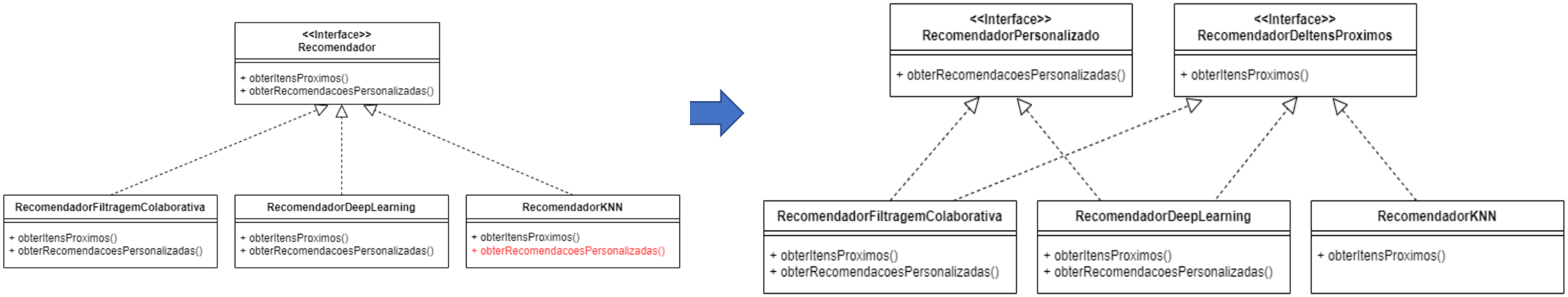
- É importante que saibamos quando vale a pena investir neste princípio
- Repare que no exemplo de OCP, o princípio LSP não foi seguido, uma vez que a classe *TransformerMixin* não tem os métodos *fit* e *transform,* que aparecem somente nas subclasses
  - Esta decisão provavelmente foi tomada com base em alguma certeza de que o uso pretendido da superclasse *TransformerMixin* de forma abstrata fosse restrito ao método *fit_transform*, que pela definição de herança é herdado também pelas subclasses

I – Interface Segregation Principle

Software Science Lab

Várias interfaces específicas são melhores do que uma interface genérica

Este princípio define que uma classe não deve conhecer nem depender de métodos que não necessite



I – Interface Segregation Principle

Software Science Lab

```python
class RecomendadorPersonalizado(ABC):
    @abstractmethod
    def obterRecomendacoesPersonalizadas(self, user):
        pass


class RecomendadorDeItensProximos(ABC):
    @abstractmethod
    def obterItensProximos(self, item):
        pass


class ReomendadorFiltragemColaborativa(RecomendadorPersonalizado,
                                       RecomendadorDeItensProximos):

    def obterRecomendacoesPersonalizadas(self, user):
        print("Recomendações Personalizadas.")

    def obterItensProximos(self, item):
        print("Recomendação de itens próximos.")


class ReomendadorKNN(RecomendadorDeItensProximos):

    def obterItensProximos(self, item):
        print("Recomendação de itens próximos.")
```

D – Dependency Inversion Principle

Software Science Lab

Devemos depender de abstrações e não de implementações

Programar para uma interface (a superclasse de maior abstração) e não para uma implementação (subclasse) permitirá desacoplar classes clientes de implementações específicas

Um exemplo de aplicação do princípio de inversão de dependência (DIP) da biblioteca Scikit-learn é a classe ***Pipeline***, que permite concatenar um conjunto de transformadores, desde que eles sigam uma interface predefinida

D – Dependency Inversion Principle

Software Science Lab

- O ***Pipeline*** recebe tuplas de nomes e transformadores (*name*, *transform*) e sequencialmente aplica os transformadores e um estimador final
  - Os elementos *transform* das tuplas intermediárias do *pipeline* precisam implementar os métodos *fit* e *transform*
  - O elemento *transform* da tupla final precisa implementar somente o método *fit*
- O código a seguir mostra a construção de um *Pipeline* que aplica sequencialmente uma padronização nos dados e depois treina o modelo KNN com os dados já padronizados e imprime os resultados de acurácia obtidos com a predição dos dados de teste. Note que o Pipeline foi construído com as tuplas *('StandardScaler', StandardScaler())* e *('KNN', KNeighborsClassifier())*, que contém respectivamente instâncias das classes *StandardScaler* e *KNeighborsClassifier*. Ambas as classes

D – Dependency Inversion Principle

Software Science Lab

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

model = Pipeline([('StandardScaler', StandardScaler()),
                  ('KNN', KNeighborsClassifier())])

model.fit(X_train, Y_train)
accuracy_score(model.predict(X_test), Y_test)
```

- Como a classe Pipeline é implementada considerando apenas a abstração, seria possível montar pipelines com outros pré-processamentos e outros tipos de modelo sem muita dificuldade

- O código mostra a construção de um *Pipeline* que aplica sequencialmente uma padronização nos dados e depois treina o modelo KNN com os dados já padronizados e imprime os resultados de acurácia obtidos com a predição dos dados de teste
- Note que o Pipeline foi construído com as tuplas *('StandardScaler', StandardScaler())* e *('KNN', KNeighborsClassifier())*, que contém respectivamente instâncias das classes *StandardScaler* e *KNeighborsClassifier*. Ambas as classes possuem os métodos *fit* e *transform*.

D – Dependency Inversion Principle

```
def _validate_steps(self):
    names, estimators = zip(*self.steps)

    # validate names
    self._validate_names(names)

    # validate estimators
    transformers = estimators[:-1]
    estimator = estimators[-1]

    for t in transformers:
        if t is None or t == "passthrough":
            continue
        if not (hasattr(t, "fit") or hasattr(t, "fit_transform")) or not hasattr(
            t, "transform"
        ):
            raise TypeError(
                "All intermediate steps should be "
                "transformers and implement fit and transform "
                "or be the string 'passthrough' "
                "'%s' (type %s) doesn't" % (t, type(t))
            )

    # We allow last estimator to be None as an identity transformation
    if (
        estimator is not None
        and estimator != "passthrough"
        and not hasattr(estimator, "fit")
    ):
        raise TypeError(
            "Last step of Pipeline should implement fit "
            "or be the string 'passthrough'. "
            "'%s' (type %s) doesn't" % (estimator, type(estimator))
        )
```

- Embora seja um bom exemplo do princípio de inversão de dependência (DIP), já que envolve programar para a abstração, a classe *Pipeline* verifica internamente se os elementos que pretende utilizar possuem os métodos *fit* e *tranform*, ou seja, se a abstração está sendo atendida, e alerta no caso de problemas, sem utilizar explicitamente o conceito de interface

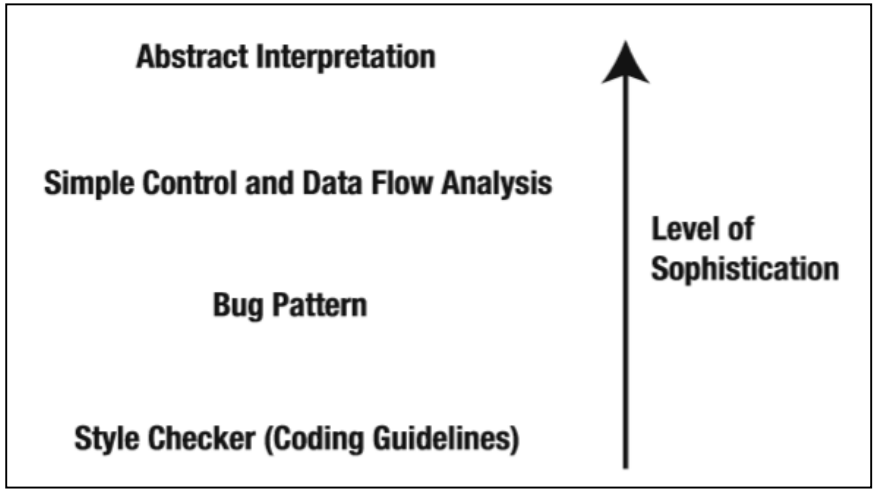
- Provavelmente esta decisão de projeto foi tomada considerando que as exigências para o seu último elemento são diferentes das para os demais

Análise Estática

- Análise estática é o processo de avaliar um sistema ou componente (e.g., código) com base na sua forma, estrutura, conteúdo ou documentação

ISO/IEC/IEEE 24765:2010: Systems and software engineering – vocabulary (2010)

Classificação de ferramentas de análise estática (Wagner, 2013)



Wagner, S., **Software Product Quality Control**, Springer, 2013.

Análise Estática em Notebooks

Software Science Lab

- Linters mais usados:
  - Flake8 “verifica pep8 (estilo), pyflakes (erros lógicos) e complexidade circular”
    - `pip install flake8`
  - PyLint “verifica pep8 (estilo) e mais alguns itens de verificação e opções”
    - `$ pip install pylint`
  - Embora haja alguma sobreposição, não há problema em combinar ferramentas de análise estática


```
!pip install pycodestyle pycodestyle_magic
!pip install flake8
%load_ext pycodestyle_magic

A célula que quiser verificar começar com
%%pycodestyle
```

Unit Testing

- Idealmente, as classes são testadas externamente aos notebooks
- PyTest é um framework para teste unitário em Python
- Os testes de código orientado a objetos serão mais eficientes se forem construídos utilizando técnicas de teste estrutural, incluindo técnicas de fluxo de controle e de fluxo de dados (fora do escopo deste curso)
- Para testar a predição dos modelos, deve se escolher métricas adequadas e *thresholds* aceitáveis (e.g., acurácia acima de 85%, R2 acima de 0.7)

PyTest

Software Science Lab

- Como o PyTest descobre os testes que deve executar?
  - Arquivos com testes devem ser nomeados como **test_*.py** ou ***_test.py**
  - Funções de teste devem iniciar com **test_**

```
%%file test_math.py

import math
def test_add():
    assert 1+1 == 2

def test_mul():
    assert 6*7 == 42

def test_sin():
    assert math.sin(0) == 0
```

Vamos praticar?

- Machine Learning com POO:

https://colab.research.google.com/drive/1nFykTTVO73tXLwkzDLlGUKaBwGfQNQrx?usp=sharing

- Para saber mais:
  - https://towardsdatascience.com/clean-machine-learning-code-bd32bd0e9212
  - https://towardsdatascience.com/pytest-for-machine-learning-a-simple-example-based-tutorial-a3df3c58cf8
  - https://github.com/musikalkemist/solidforml/tree/main/solidforml

Software Science Lab

Gerência de Configuração

Engenharia de Software para Ciência de Dados

O que é Gerência de Configuração?

Software Science Lab

- **Gerência de configuração (GC) é o processo de identificar, organizar e controlar modificações ao software sendo construído**
- A ideia é maximizar a produtividade minimizando os enganos

- Objetivos da GC:
  - Definir políticas para controle de versões, garantindo a consistência dos artefatos produzidos
  - Definir procedimentos para **solicitações de mudanças**
  - Auditar mudanças

Conceitos Básicos de GC

- Configuração
- Baseline
- Build
- Integração Contínua
- Release
- Tag
- Branch
- Merge

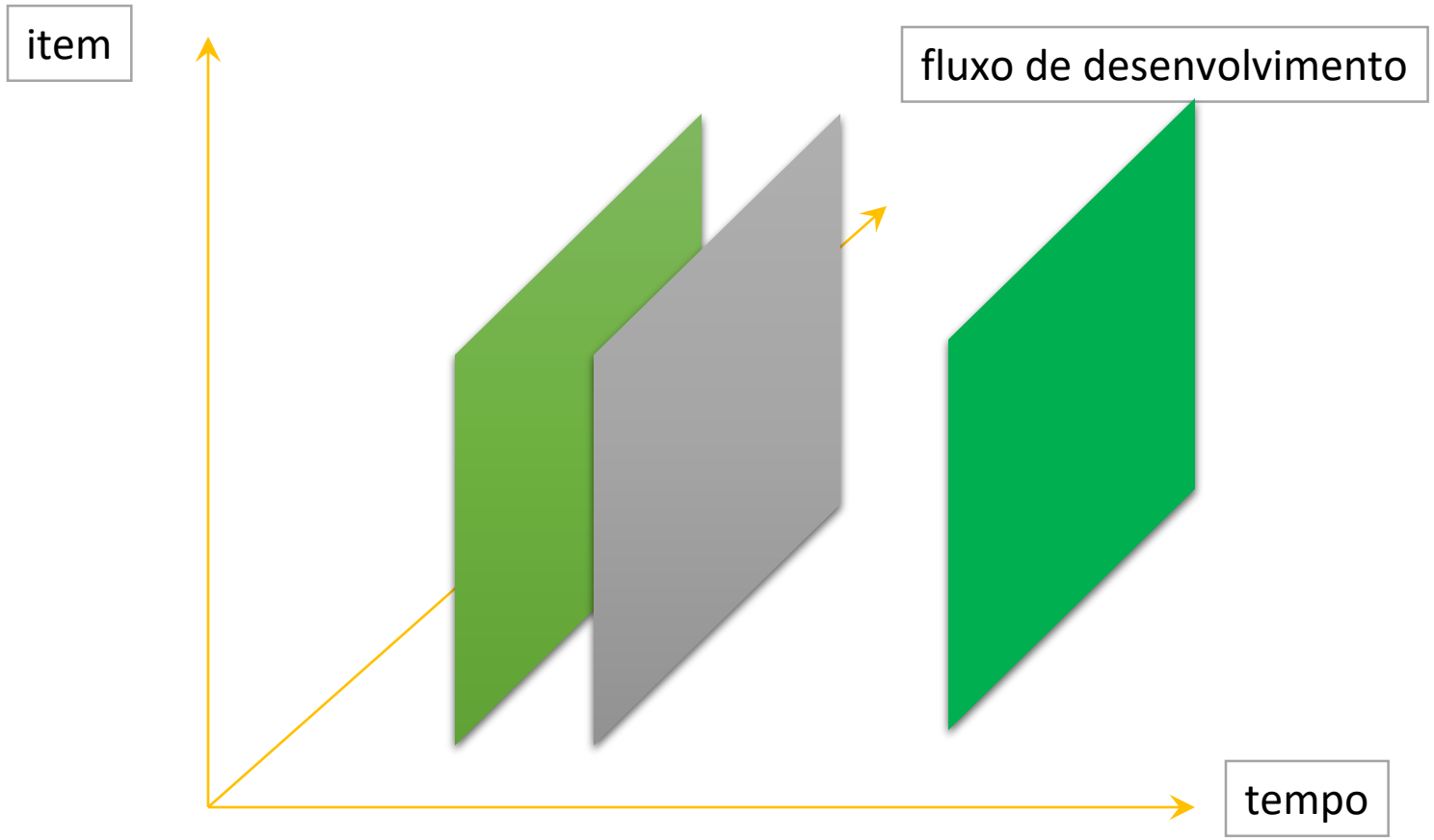
Configuração

- Um projeto de desenvolvimento de software produz os seguintes itens:
  - Programas (código fonte, programas executáveis, bibliotecas de componentes, etc.)
  - Documentação (manuais do usuário, documento de requisitos, modelo de análise e projeto, etc.)
  - Dados (dados de teste e do projeto)
- Esses conjuntos de itens são chamados, coletivamente, de configuração do software

Baseline

- Uma especificação ou produto que foi formalmente revisado e aceito
  - Serve como base para os passos posteriores do desenvolvimento
- A configuração do software em um ponto discreto no tempo
- Só pode ser modificado através de procedimentos formais (solicitações de mudança)

Baseline



Razões para Criar um Baseline

- **Reprodutibilidade:** a habilidade de reproduzir uma versão anterior do sistema
- **Geração de Relatórios:** A comparação dos conteúdos de dois baselines ajuda na depuração e criação de documentação
- **Controle de Mudanças:** referencial para comparações, discussões e negociações

Build

- Representa uma versão não necessariamente completa do sistema em desenvolvimento, mas com certa estabilidade
- Inclue não só código fonte, mas documentação, arquivos de configuração, base de dados, etc.
- A política de geração dos builds deve ser bem definida na estruturação do ambiente

Integração Contínua

- Geração frequente (pelo menos diária) de builds do sistema
  - As partes do sistema são integradas constantemente
  - Problemas de integração passam a ser encontrados logo que introduzidos, na maioria dos casos
- Considerada uma das “melhores práticas” no desenvolvimento de software
- A geração de builds deve ser automatizada e realizada com frequência adequada

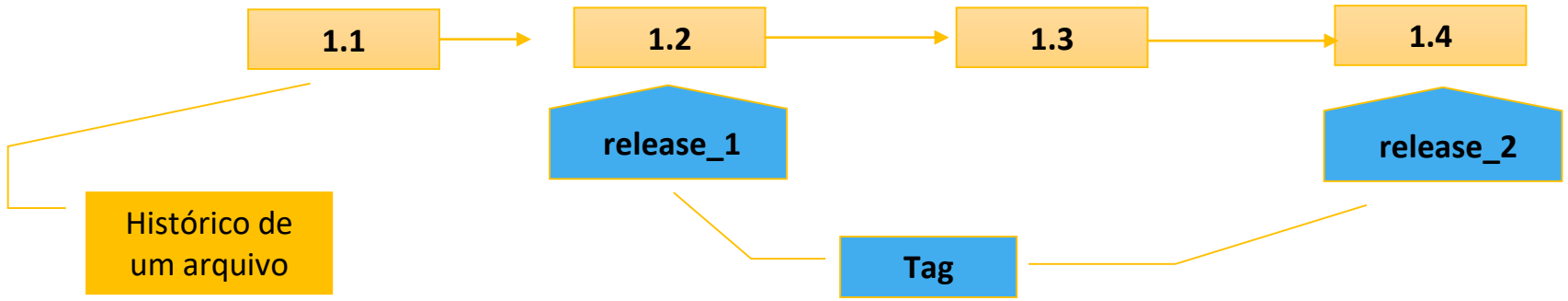
Release

- Identificação e empacotamento de artefatos entregues ao cliente (interno ou externo)
- Um release implica no estabelecimento de um novo baseline, de produto
- Produto de software supostamente sem erros
  - Versão do sistema validada após os diversos tipos de teste
  - Garantia de que todos os itens de configuração foram devidamente testados, avaliados, aceitos e estão disponíveis no novo *baseline*

- Processo iterativo/incremental produz, em geral, mais de um release

Tags

- Rótulos que são associados a conjuntos de arquivos
- Um tag referencia um ou mais arquivos em um ou mais diretórios
  - Costuma-se usar tags para denominar uma versão do projeto (um build ou release) rotulando todos os arquivos associados



Branch

- Criação de um fluxo alternativo para atualização de versões de itens de configuração
- Recurso muito poderoso
- Devem existir regras bem definidas para criação de branches
  - Por que e quando devem ser criados?
  - Quando retornar ao fluxo principal?
- Branches normalmente se originam de correções em versões anteriores

Merge

- Unificação de diferentes versões de um mesmo item de configuração
- Integração dos itens de configuração de um branch com os itens de configuração do fluxo principal
- Algumas ferramentas fornecem um mecanismo automático para realização de merges
  - Mesmo com o uso de ferramentas, em vários casos há necessidade de intervenção humana

O que é Controle de Mudanças?

- **Controle de Mudanças é o processo de avaliar, coordenar e decidir sobre a realização de mudanças propostas a itens de configuração**

- Envolve controlar as mudanças de maior relevância
  - Todas as mudanças são analisadas
    - Análises de custo x benefício produzidas pelos stakeholders
  - Apenas as aprovadas são priorizadas e realizadas
  - Sempre se sabe quem modificou o que, onde e quando

- É recomendada a integração entre os sistemas de controle de versão e de controle de mudanças

Solicitações de Mudança

- Algumas informações que podem estar incluídas em uma solicitação de mudança:
  - Identificação única
  - Solicitante
  - Sistema/Projeto
  - Item a ser modificado
  - Classificação (melhoria, correção de defeito, outra)
  - Prioridade
  - Descrição
  - Situação (nova, atribuída, finalizada, verificada, fechada)

Correções Emergenciais

- Em algumas situações (e.g., defeitos), não há tempo para seguir os procedimentos padrão para a realização de mudanças
- Mesmo nessas situações, porém, é muito importante que seja criada uma solicitação de mudança
  - O objetivo é garantir um mínimo de ordem, mesmo em uma situação caótica

Ferramentas de Apoio à Gerência de Configuração

Software Science Lab

Ferramenta de Controle de Versões (e.g., GIT, SVN, CVS)

- Manter todos os arquivos em um repositório central
- Controlar o acesso a esse repositório, de modo a garantir a consistência dos artefatos

Ferramentas de Geração de Builds e de Integração Contínua (e.g., BitBucket Pipelines, Jenkins, AWS CodePipeline, Azure Pipelines, GitLab CI/CD)

- Automatizar o processo de geração de *builds* e o *deploy*

Ferramentas de Gestão de Solicitações de Mudanças (e.g., JIRA, Azure DevOps, Redmine)

- Automatizar o processo de submissão e a gestão do ciclo de vida das solicitações de mudança

Software Science Lab

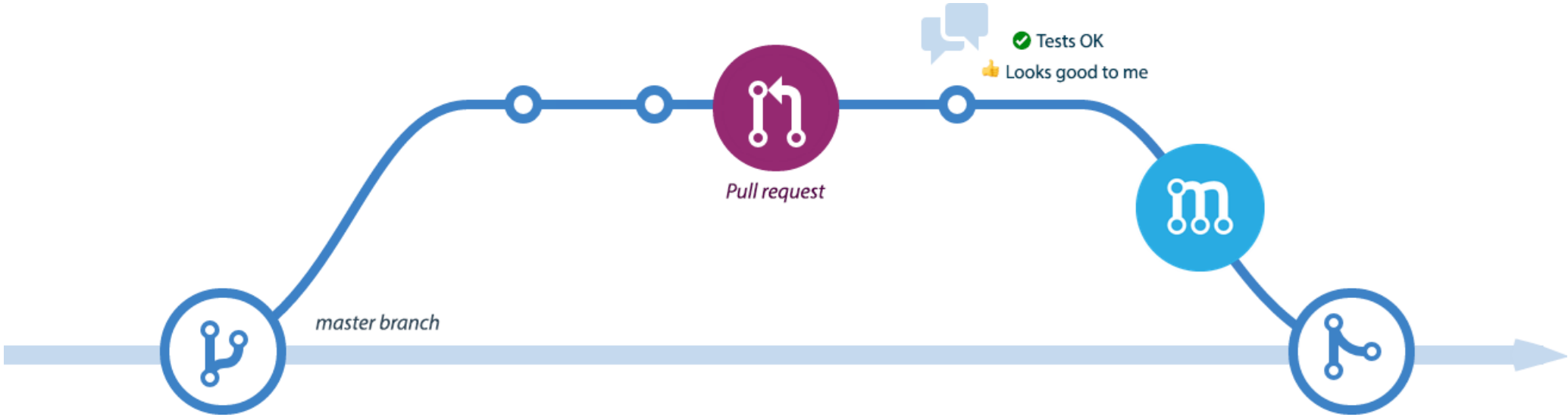
In case of fire

1. git commit

2. git push

3. leave building

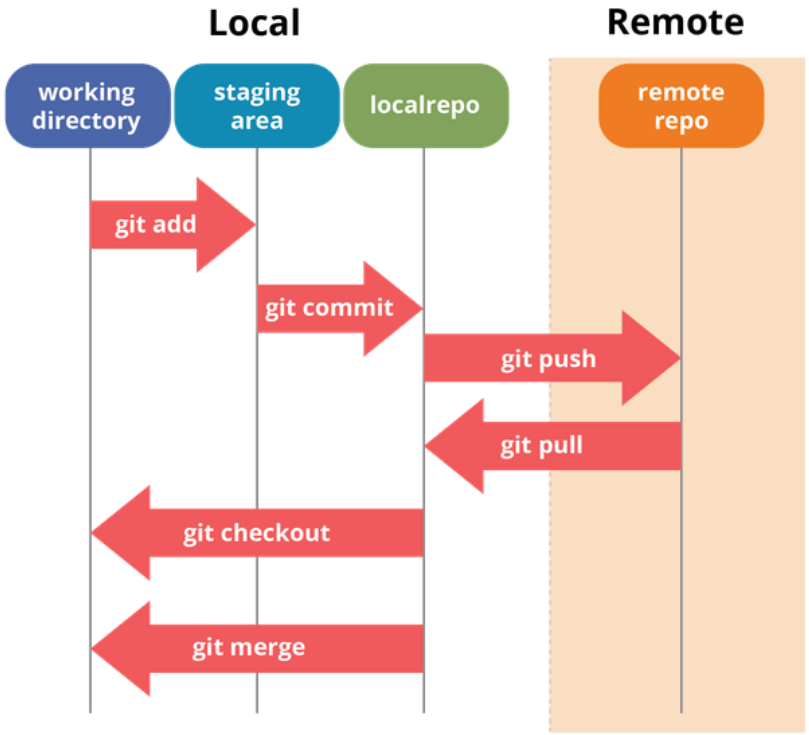
Pull Requests



Processo para merge *de um* Pull/Merge Request

Git Comandos Básicos

- **init** – cria um repositório vazio
- **clone** – cria um repositório vazio e copia para ele um repositório remoto



Software Science Lab

Implantação (*Deployment*) e Arquitetura de Sistemas de Software Inteligentes

Engenharia de Software para Ciência de Dados

Implantação (Deployment)

- Uma vez encontrado um modelo adequado, precisamos decidir como ele será usado em produção
- A implantação de modelos de Machine Learning em produção envolve disponibilizar modelos treinados, avaliados e aprovados

Implantação de Modelos de ML

- **Modelo "embarcado"** → o modelo é construído e empacotado na aplicação que o consome (*e.g.*, no backend de uma aplicação desenvolvida)

- **Modelo implantado como um serviço separado** → o modelo é provido através de um serviço que pode ser implantado de forma independente das aplicações que o consomem (e.g., *endpoint, container*)

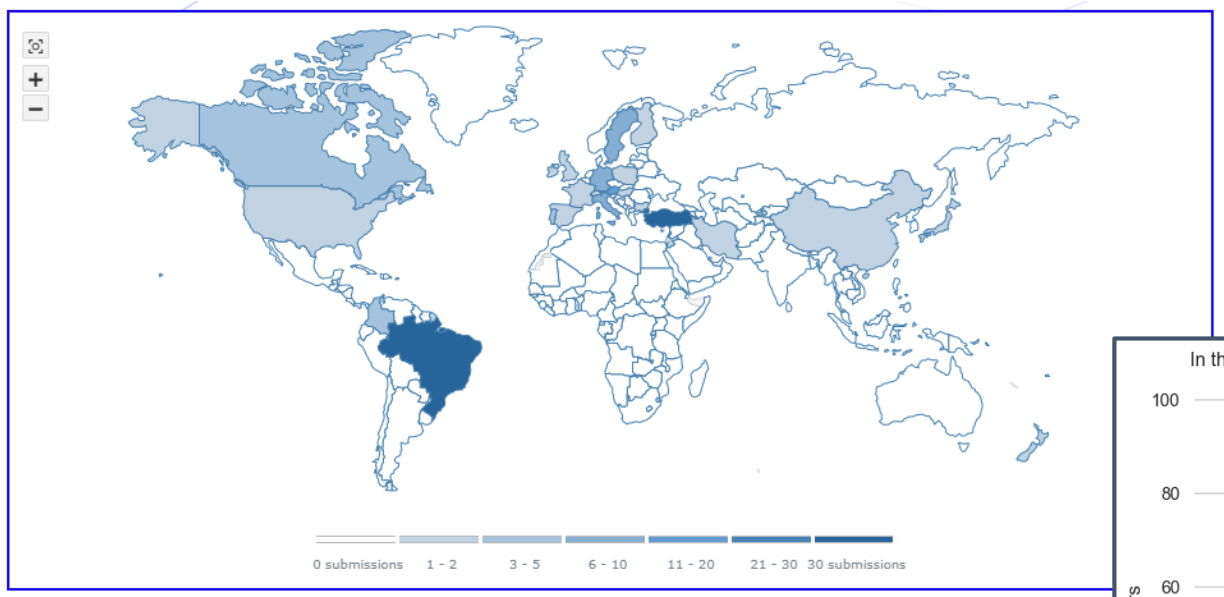
- **Platform-as-a-Service** (PaaS) → frameworks/ferramentas/ambientes são utilizadas para a implantação do modelo de forma autônoma, tornando ele disponível para o usuário final (e.g., Amazon SageMaker, Google Cloud Platform, Microsoft Azure)

Implantação de Modelos de ML

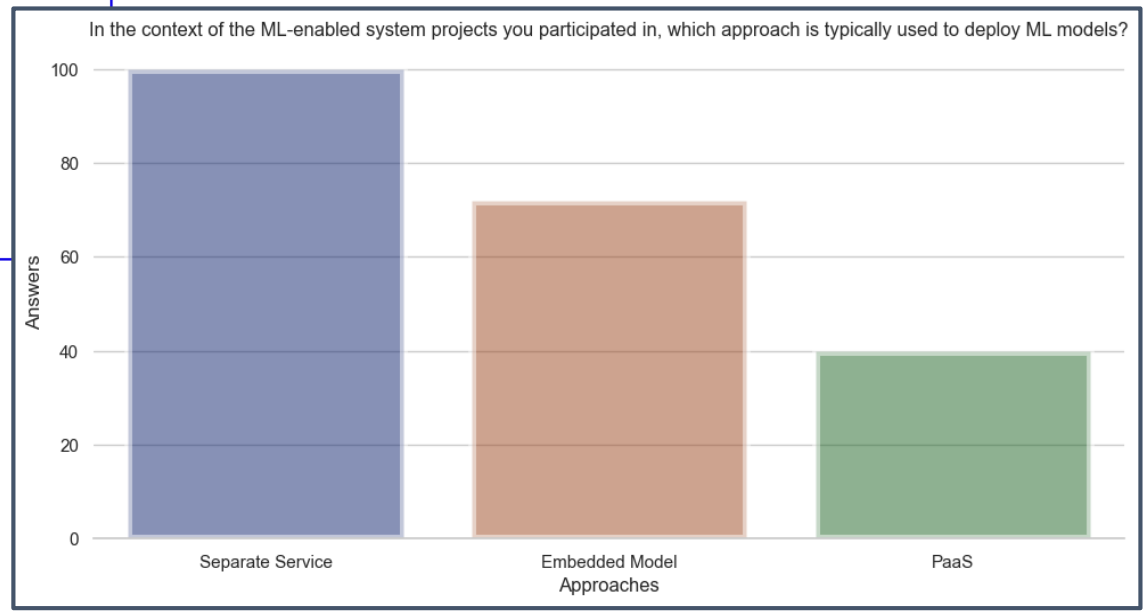
Software Science Lab

Forma de Implantação	Vantagens	Desvantagens
Modelo "embarcado"	• Maior Controle • Mais recursos customizados	• Maior esforço de desenvolvimento e • Gerenciamento de ambiente
Modelo implantado como um serviço separado (Serverless)	• Facilidade de disponibilização • Maior escalabilidade	• Envolve custo de processamento em nuvem
Platform-as-a-Service	• Facilidade de implementação • Menor esforço de gerenciamento dos recursos de ML	• Envolve custo de processamento em nuvem • A aplicação é pouco customizável

Estado da Prática

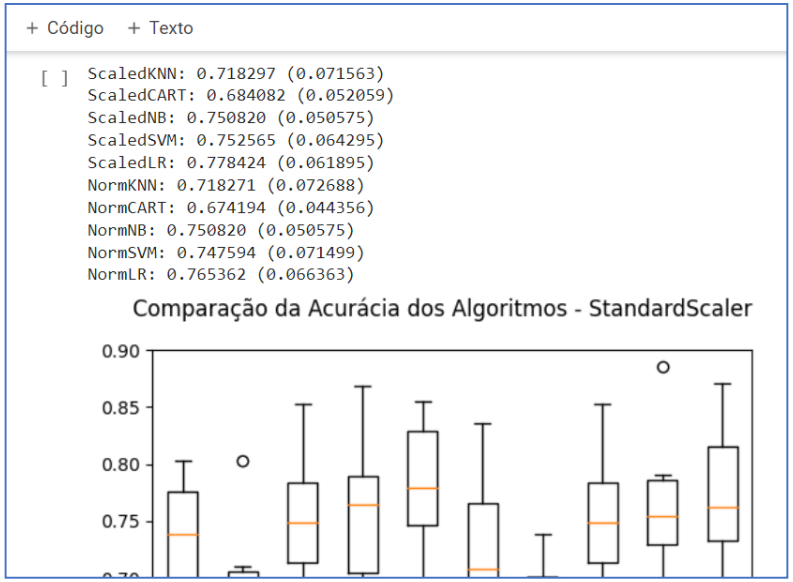


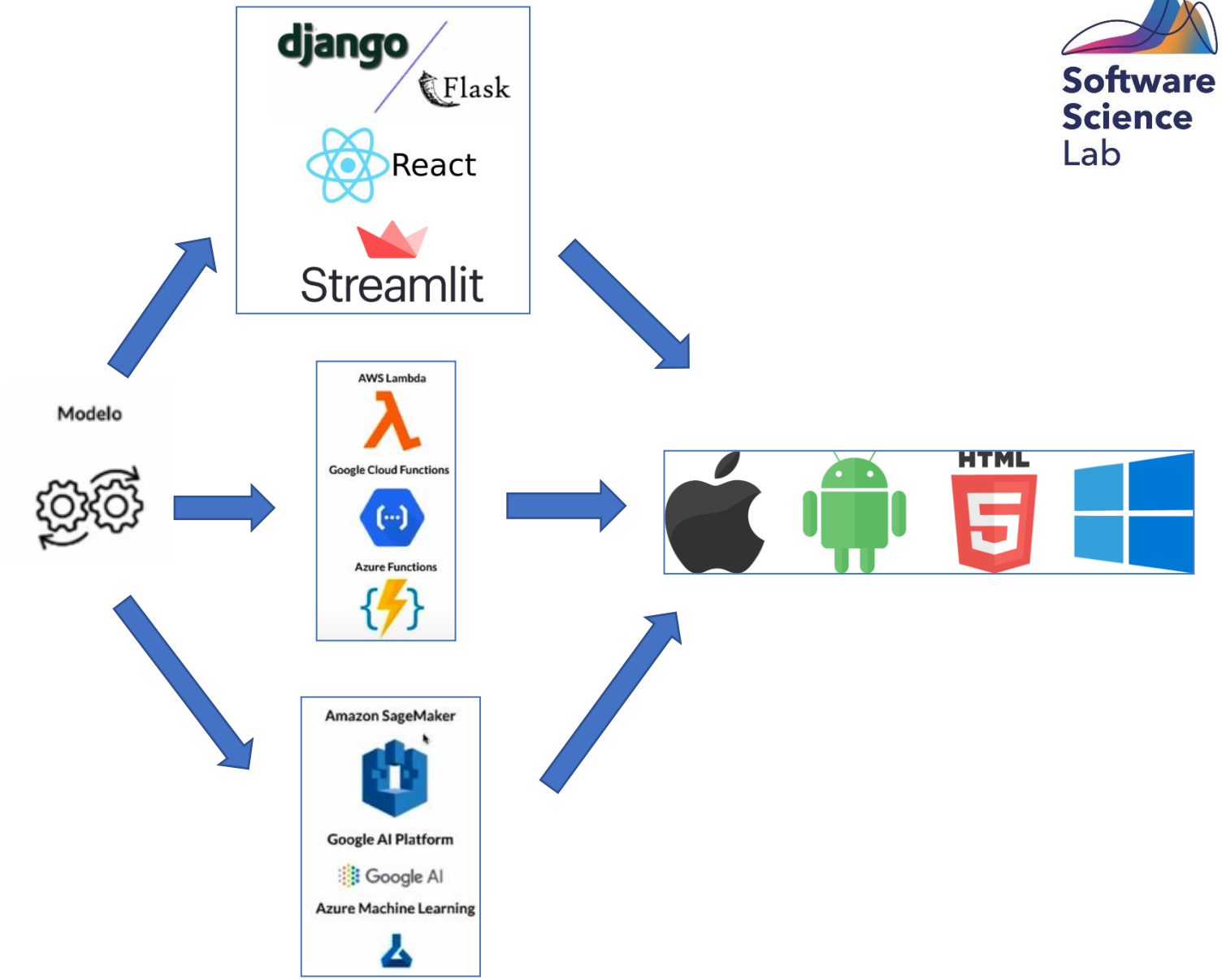
https://machine-learning-survey.web.app/



Implantação

Notebook: Análise exploratória, pré-processamento, treinamento e avaliação





Exportando um modelo como um arquivo

- Após treinar o modelo é possível salvá-lo usando bibliotecas como **pickle e joblib** (e.g., ***model.pkl, model.joblib***), e usa-lo como um script **.py** (e.g., ***model.py***)

```
# saving the model
import pickle
pickle_out = open("classifier.pkl", mode = "wb")
pickle.dump(model, pickle_out)
pickle_out.close()
```

```
import pickle
```

...

```
# loading the trained model
pickle_in = open('classifier.pkl', 'rb')
classifier = pickle.load(pickle_in)
```

...

```
# Making predictions
prediction = classifier.predict(
  [[Gender, Married, ApplicantIncome, LoanAmount,
  Credit_History]])
```

Implantação do modelo em nuvem

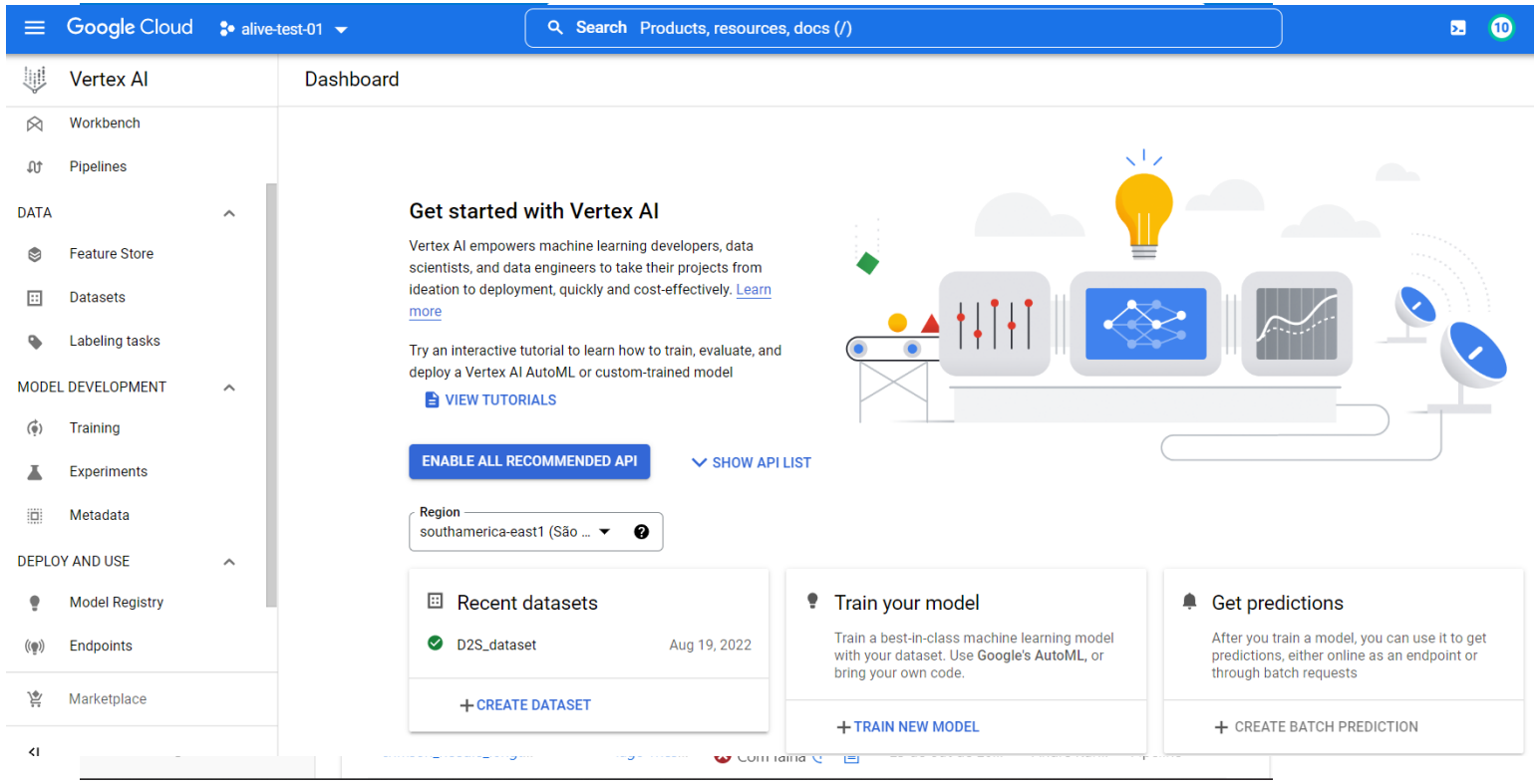
- É necessário criar uma conta em alguma plataforma de computação
- Geralmente envolve o uso da interface gráfica ou o SDK para:
  - Criar / Importar um modelo (e.g., representação em arquivo, AutoML)
  - Registrar o modelo na plataforma de computação
  - Definir as configurações dos recursos de computação que serão usados
  - Criar um endpoint para hospedar o modelo
  - Implantar o modelo no endpoint

```
model = aiplatform.Model('/projects/my-project/locations/us-central1/models/{MODEL_ID}')

endpoint.deploy(model,
                min_replica_count=1,
                max_replica_count=5
                machine_type='n1-standard-4',
                accelerator_type='NVIDIA_TESLA_K80',
                accelerator_count=1)
```

Serviços de computação em nuvem

Software Science Lab

- No modelo ***Serverless***, o provedor em nuvem aloca e fornece dinamicamente os recursos necessários



BigQuery ML

Software Science Lab

- Recurso que permite criar e implantar modelos de ML para realizar analises preditivas na origem, onde já são armazenados os dados

Características:

- Diferentes modelos de ML
- Pré-processamento de features
- Criação de modelos
- Ajuste de hiperparâmetros
- Inferência
- Avaliação
- Publicação de modelos

```
1 CREATE MODEL numbikes.model
2 OPTIONS
3   (model_type='linear_reg', labels=['num_trips']) AS
4 WITH bike_data AS
5 (
6 SELECT
7   COUNT(*) as num_trips,
```

Run Query

```
1 SELECT
2   predicted_num_trips, num_trips, trip_date
3 FROM
4   ml.PREDICT(MODEL `numbikes.model`,(WITH bike_data AS
5 (
6 SELECT
7   COUNT(*) as num_trips,
```

Vamos praticar?

Software Science Lab

- Salvando e carregando um modelo usando Pickle
- Fazendo um *deploy* de um modelo como um *endpoint* na Google Cloud

https://colab.research.google.com/drive/1fF3AJw7vn17fyyLCa3MsnIUlGfnrK_jN?usp=sharing

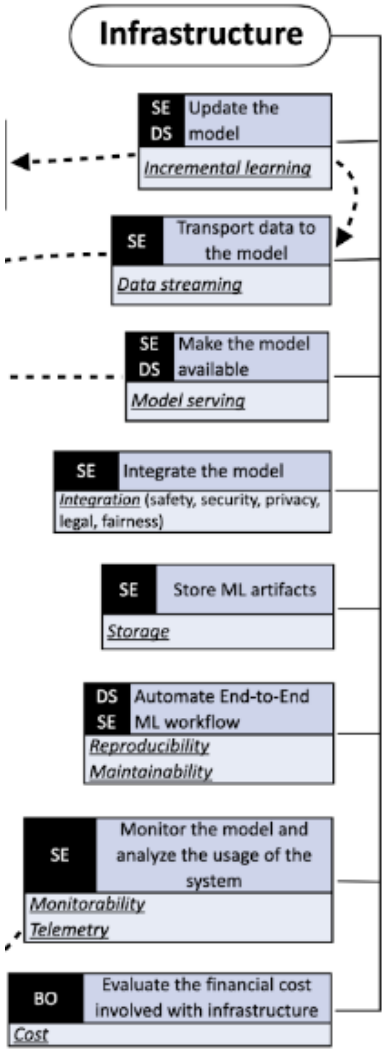
Arquitetura de Sistemas Inteligentes

Software Science Lab

- É uma boa prática pensar na operação do sistema inteligente desde o início do projeto
  - Qual é a limitação da infraestrutura que pode estar envolvida com seu modelo?
  - Quais recursos tenho que criar para suportar a infraestrutura que o modelo precisa?
  - A infraestrutura suportará inferências em tempo real ou em lotes?
- Pensar nessas preocupações desde o inicio ajuda a desenhar um melhor fluxo de trabalho de ML e evita falhas na produção

Não se esqueça: grandes modelos podem ser inutilizáveis devido a restrições de recursos para executá-los e mantê-los!

Relembrando PerSpecML



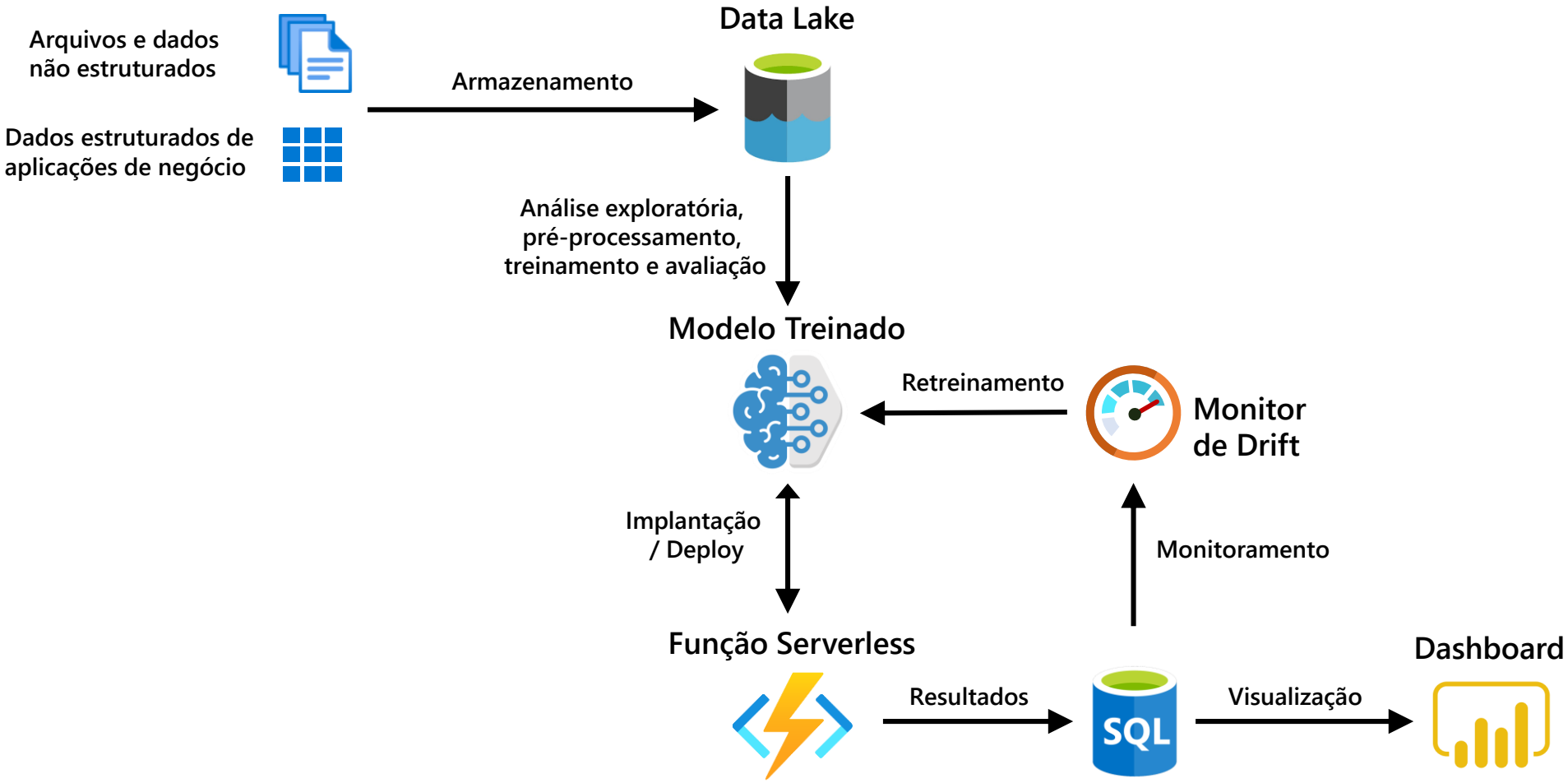
Machine Learning frequentemente impõe requisitos de infraestrutura

- Atualização do modelo
- Transferência dos dados
- Deploy do modelo
- Integração do modelo
- Armazenamento e gestão de artefatos de ML
- MLOps
- Monitoramento do modelo
- Custo financeiro

Arquitetura de Sistemas de Software Inteligentes

- A arquitetura depende fortemente de:
  - Requisitos de infraestrutura
  - Requisitos não funcionais (desempenho preditivo do modelo vs latência da predição)
  - Tipo de predições: em tempo real ou offline

Arquitetura de alto nível de um sistema de software inteligente em nuvem

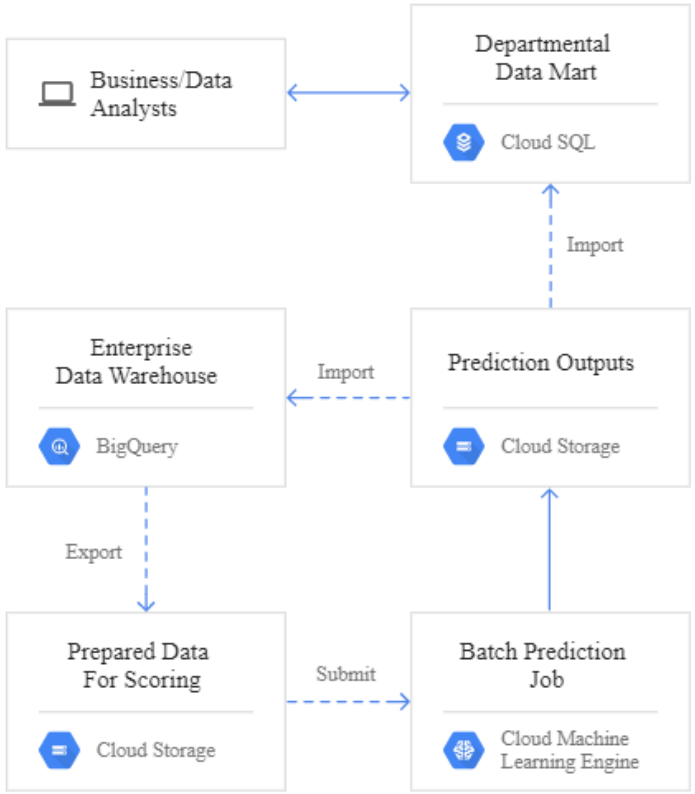


Tipo de previsões

- **Offline:** o modelo de ML é usado para processar dados em lote, em que as previsões não são necessárias em tempo real. Por exemplo:
  - Previsão de demanda: estimar a demanda de produtos por loja diariamente para otimização de estoque e entrada
  - Análise de sentimento: identificar o sentimento geral em relação aos seus produtos analisando o feedback do cliente

- **Online:** o modelo de ML é usado para fornecer previsões em tempo real, com base em solicitações online. Os dados são adquiridos na sequencia. Por exemplo:
  - Detecção de fraude: identificar se uma transação é fraude
  - Manutenção preditiva: prever se uma determinada peça da máquina falhará nos próximos N minutos, dados os dados do sensor em tempo real

Previsões Offline (batch)



Fluxo dos dados:

1. Dados armazenados no Cloud Storage

2. Requisição em lote com os dados para um modelo salvo no workspace de ML

3. Saídas do modelo importadas para um data warehouse ou um banco de dados SQL

https://cloud.google.com/architecture/minimizing-predictive-serving-latency-in-machine-learning

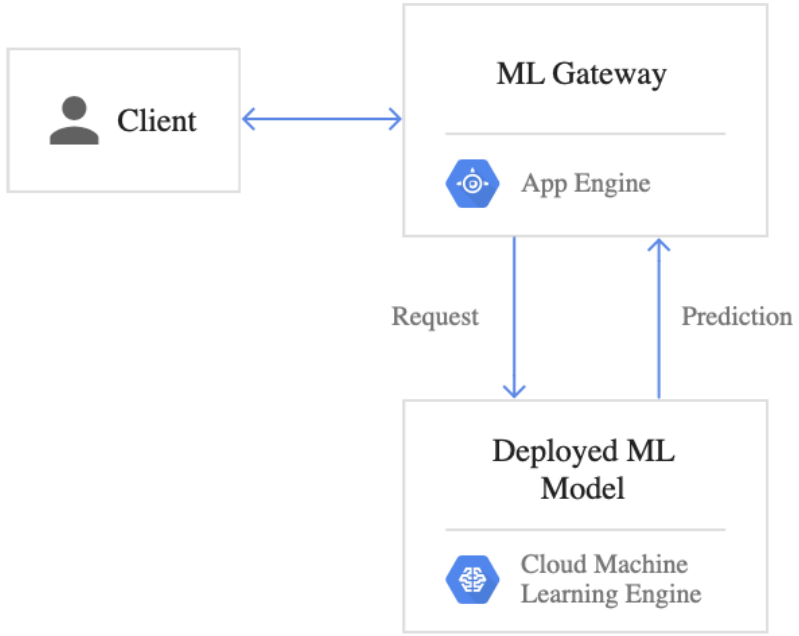
Previsões Online (tempo real)

Previsões em tempo real podem ser entregues de várias maneiras:

- **Síncrona:** Tem uma solicitação explicita de previsão. Aqui a solicitação e a previsão são executadas em sequência

- **Assíncrona:** As previsões são entregues independentemente da solicitação de previsão

Previsões Online - Síncrona



Fluxo dos dados:

1. Um aplicativo envia solicitações HTTP para um ponto de extremidade (endpoint)
2. O aplicativo recebe a resposta assim que o endpoint produz a previsão

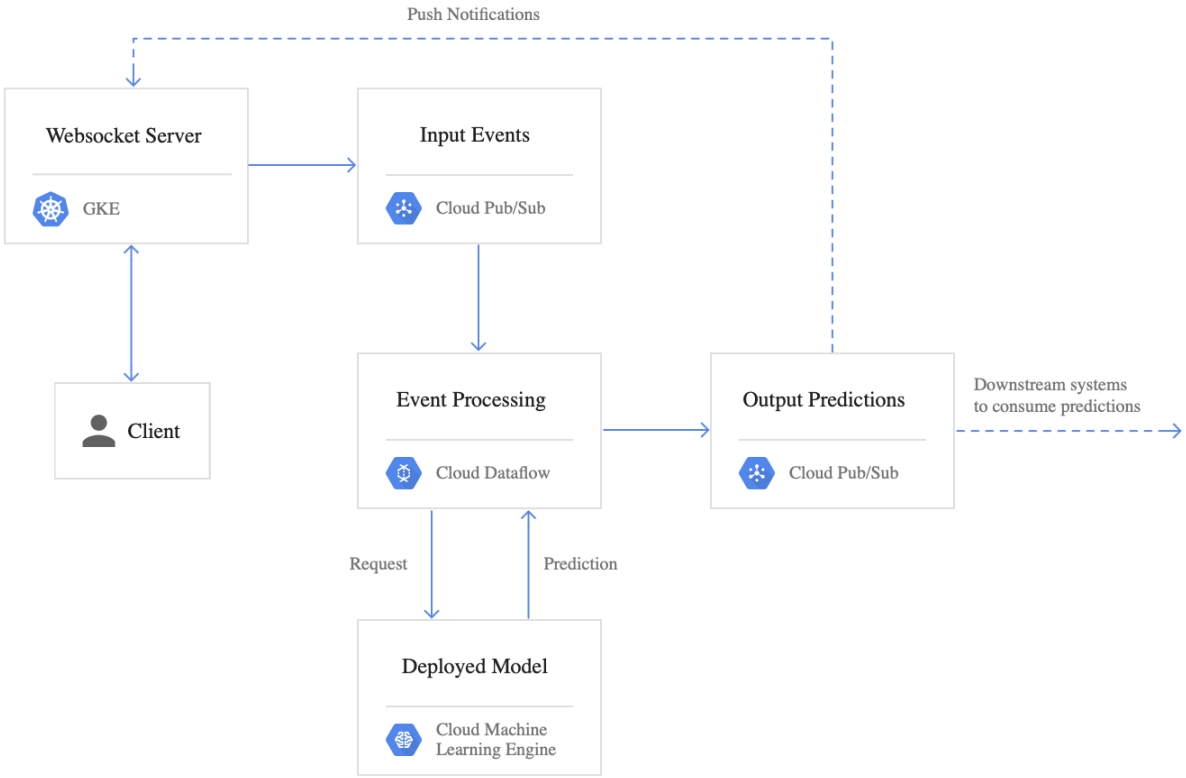
Não esquecer:

Deve ser implementada a lógica no back-end do aplicativo como parte do sistema inteligente para:

- Executar o pré-processamento dos dados
- Executar o pós-processamento na previsão de saída antes de enviar a resposta de volta ao caller

https://cloud.google.com/architecture/minimizing-predictive-serving-latency-in-machine-learning

Previsões Online - Assíncrona



Fluxo dos dados:

1. Um aplicativo envia envia eventos para o PUB/SUB (um serviço de mensagens em tempo real)
2. Esses eventos são processados em tempo real pelo Dataflow (um serviço de processamento unificado de dados)
3. O Dataflow invoca o modelo e envia as previsões para o PUB/SUB
4. As mensagens do PUB/SUB contendo as previsões são enviadas de volta ao aplicativo

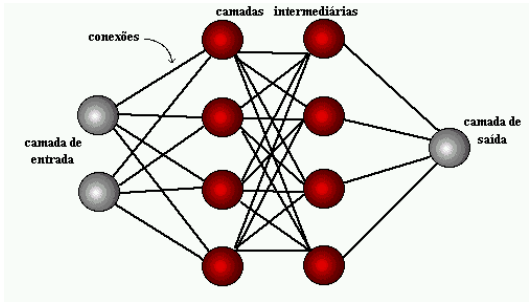
https://cloud.google.com/architecture/minimizing-predictive-serving-latency-in-machine-learning

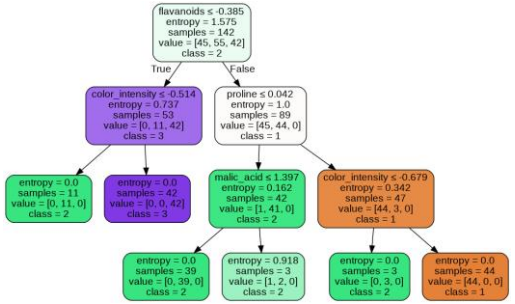
Otimizando previsões em tempo real

- Para previsões em tempo real, é importante minimizar o tempo de resposta, porque a ação esperada deve acontecer imediatamente

A nível do modelo

- Reduzir a complexidade do modelo. Por exemplo: redução de camadas em redes neurais, ou níveis em arvores de decisão





- Remoção de partes não utilizadas, redundantes ou irrelevantes do modelo (otimizar o passo da fase de treinamento até a fase de operação do modelo

Otimizando previsões em tempo real

Software Science Lab

- Para previsões em tempo real, é importante minimizar o tempo de resposta, porque a ação esperada deve acontecer imediatamente

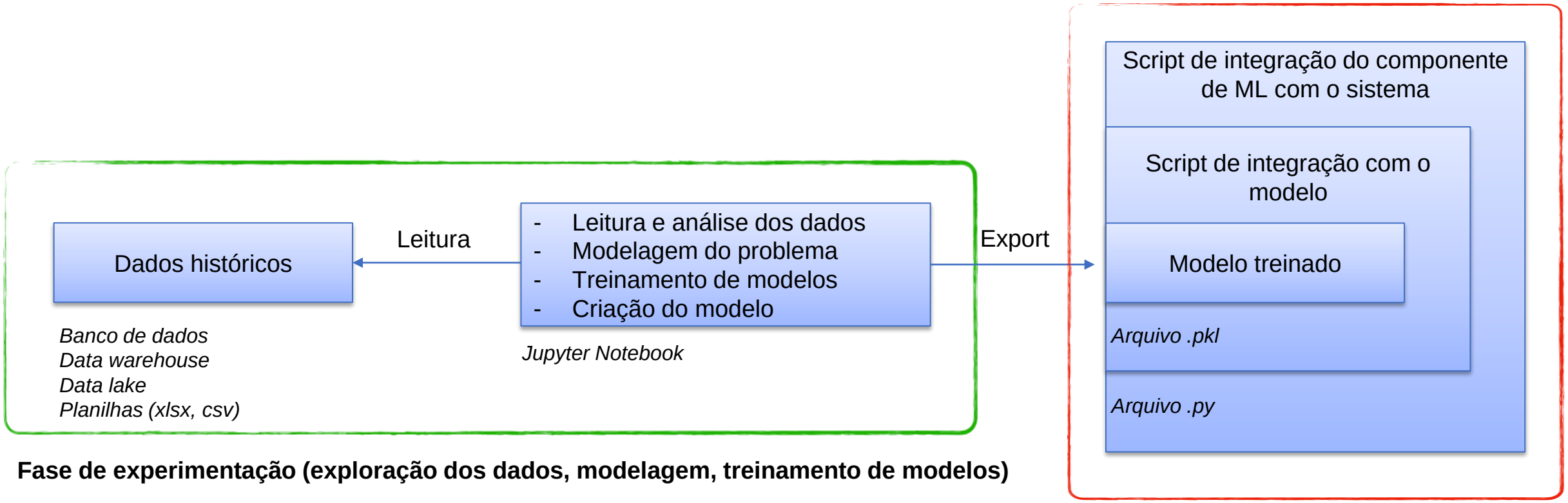
Outras estratégias

- armazenando dados em recursos de baixa latência de leitura. Por exemplo: bancos de dados não SQL

- Usando aceleradores para servir os modelos. Por exemplo: GPU, TPU

Portanto, deve haver um trade-off entre a eficácia preditiva do modelo e sua latência de previsão.

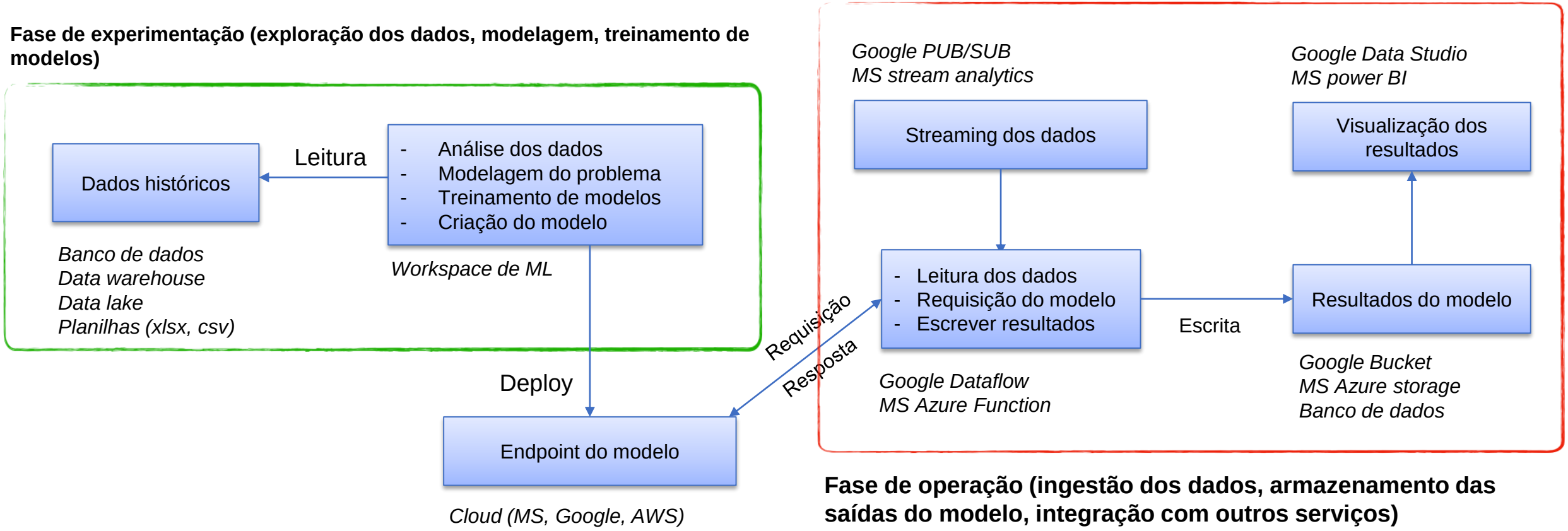
Transição fase de treinamento para produção – modelo embutido no backend da aplicação



Fase de experimentação (exploração dos dados, modelagem, treinamento de modelos)

Fase de operação (ingestão dos dados, armazenamento das saídas do modelo, integração com outros serviços)

Transição fase de treinamento para produção – modelo em nuvem



Serverless/Cloud

Software Science Lab

- Para saber mais:
  - Watson: https://towardsdatascience.com/deploy-a-machine-learning-model-from-a-jupyter-notebook-9257ae5a5f7c
  - Google Cloud: https://towardsdatascience.com/how-to-do-serverless-machine-learning-with-scikit-learn-on-google-cloud-ml-engine-db26dcc558a2
  - AWS Lambda: https://towardsdatascience.com/deploying-sklearn-machine-learning-on-aws-lambda-with-sam-8cc69ee04f47

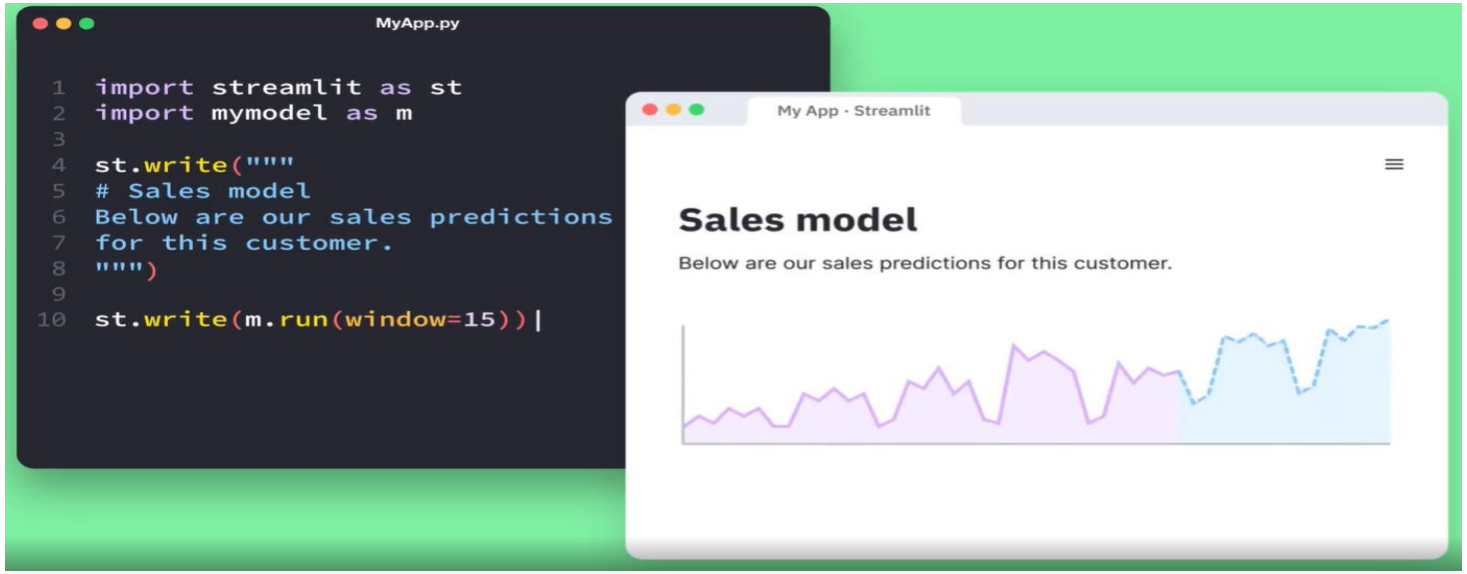
Aplicação Web

- Criar uma aplicação web (e.g. ***app.py***) que irá utilizar o modelo para fazer predições e exibir os resultados

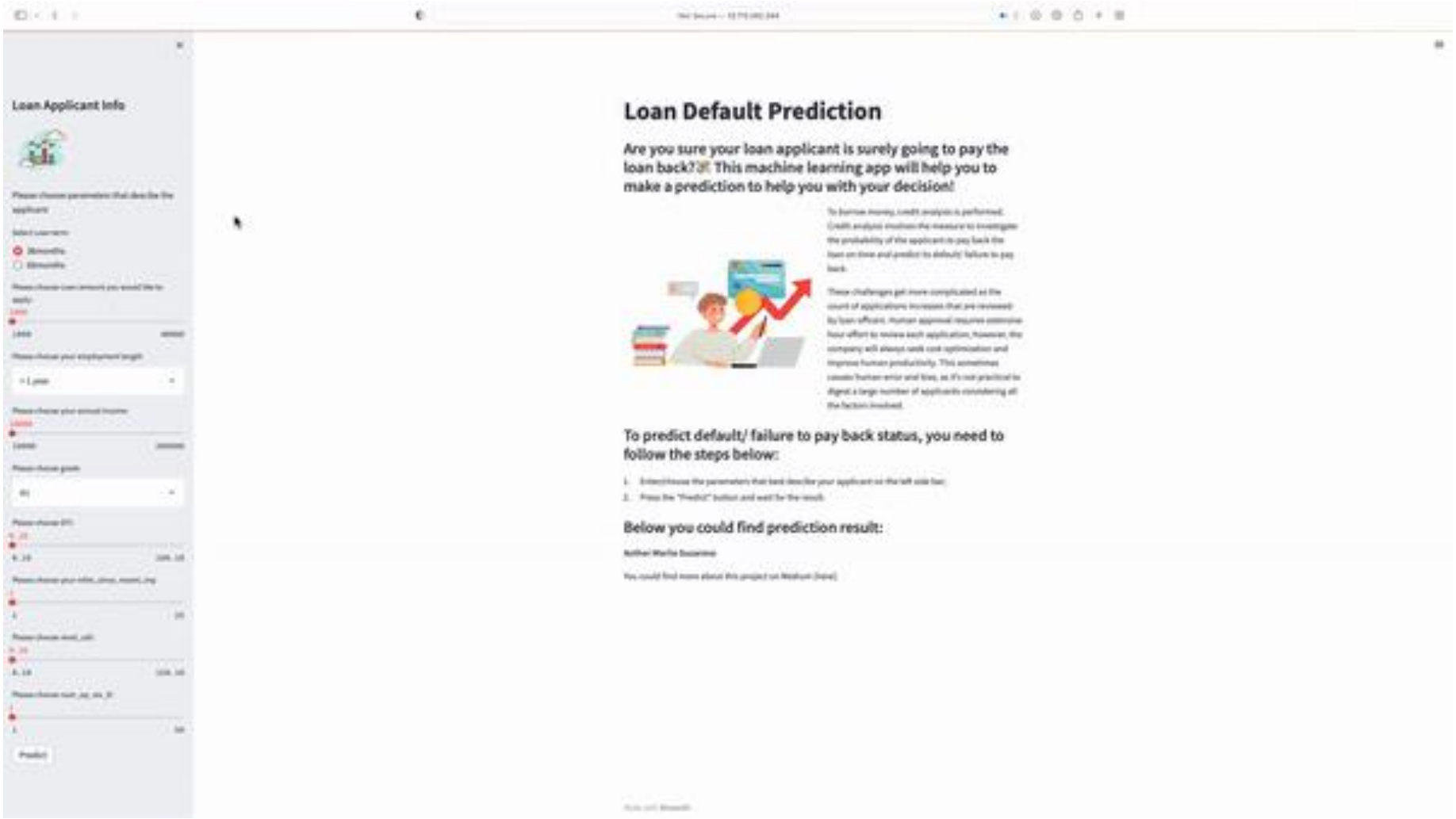
- A aplicação web pode ser criada, por exemplo, usando bibliotecas como **Streamlit** ou **Flask**

Streamlit

- É usado para criar de uma maneira mais rápida aplicativos baseados em dados
- É simples, gratuito e oferece *templates* prontos
- Também oferece um ambiente para implantar as aplicações



Exemplo Streamlit



Aplicação Web

- Para saber mais:
  - https://medium.com/@data.science.enthusiast/create-web-apps-for-your-ml-model-using-python-and-streamlit-cc966142633d
  - https://towardsdatascience.com/how-to-build-an-automl-app-in-python-e216763d10cd
  - https://medium.datadriveninvestor.com/machine-learning-model-deployment-using-flask-from-scratch-d2e8f622ed79

Engenharia de Software para Ciência de Dados

Prof. Marcos Kalinowski
Prof. Tatiana Escovedo
Prof. Hugo Villamizar