Introdução a MLOps e Ciclo de Vida de Modelos

- O que é MLOps: integração entre Data Science, Engenharia de Dados e Engenharia de Software para operacionalizar ML de forma reprodutível e escalável.
- Problemas que MLOps resolve: “notebook-hell”, fragilidade em produção, reprodutibilidade, debuggabilidade, dívida técnica de ML.

O que é o “notebook hell”

Em equipes de ciência de dados, é comum o uso intenso de notebooks Jupyter para:

- Explorar dados;
- Testar hipóteses;
- Treinar modelos;
- Gerar gráficos e métricas.

Com o tempo, isso leva a uma situação caótica:

- Cada cientista tem **versões diferentes** dos notebooks;
- É difícil **reproduzir resultados** (ex.: “por que esse modelo deu 92% ontem e 85% hoje?”);
- O código fica **espalhado e não versionado corretamente**;
- Há **dependências não documentadas** e **execuções manuais**;
- A transição de *prototipagem* → *produção* é dolorosa (muito retrabalho).

Daí o termo: **notebook hell**

Como o MLOps resolve isso

MLOps (Machine Learning Operations) aplica práticas de **DevOps** ao ciclo de vida do *machine learning*, automatizando e padronizando etapas.

Problema (notebook hell)	Solução com MLOps
Código solto em notebooks	Pipelines versionadas e reprodutíveis (ex.: com **MLflow**, **Kubeflow**, **Airflow**, **DVC**)
Falta de controle de versões	Versionamento de **dados**, **modelos** e **código** (Git + DVC/MLflow)
Difícil reproduzir resultados	Ambientes **containerizados** (Docker) e **infraestrutura como código**
Treinamentos manuais	**Automação** de *training pipelines* e *CI/CD* para modelos
Falta de rastreabilidade	**Monitoramento e logging** de experimentos e métricas
Dificuldade em colocar modelo em produção	**Serviços de deploy automatizado** (CI/CD, APIs, Kubernetes, etc.)

MLOps, ou Operações de Machine Learning, é um conjunto de práticas que automatizam e simplificam o ciclo de vida completo de modelos de machine learning, desde o desenvolvimento até a implantação e o monitoramento contínuo.

Ele une o desenvolvimento (Dev) com a implantação e operação (Ops), permitindo que equipes de ciência de dados e de TI colaborem de forma mais eficiente para colocar modelos em produção de maneira rápida, escalável e confiável.

O termo MLops é uma combinação de aprendizado de máquina (ML) e DevOps. O termo foi cunhado em 2015 em um artigo chamado "Hidden technical debt in machine learning systems", que descrevia os desafios inerentes ao tratamento de grandes volumes de dados e como usar processos de DevOps para incutir melhores práticas de ML.

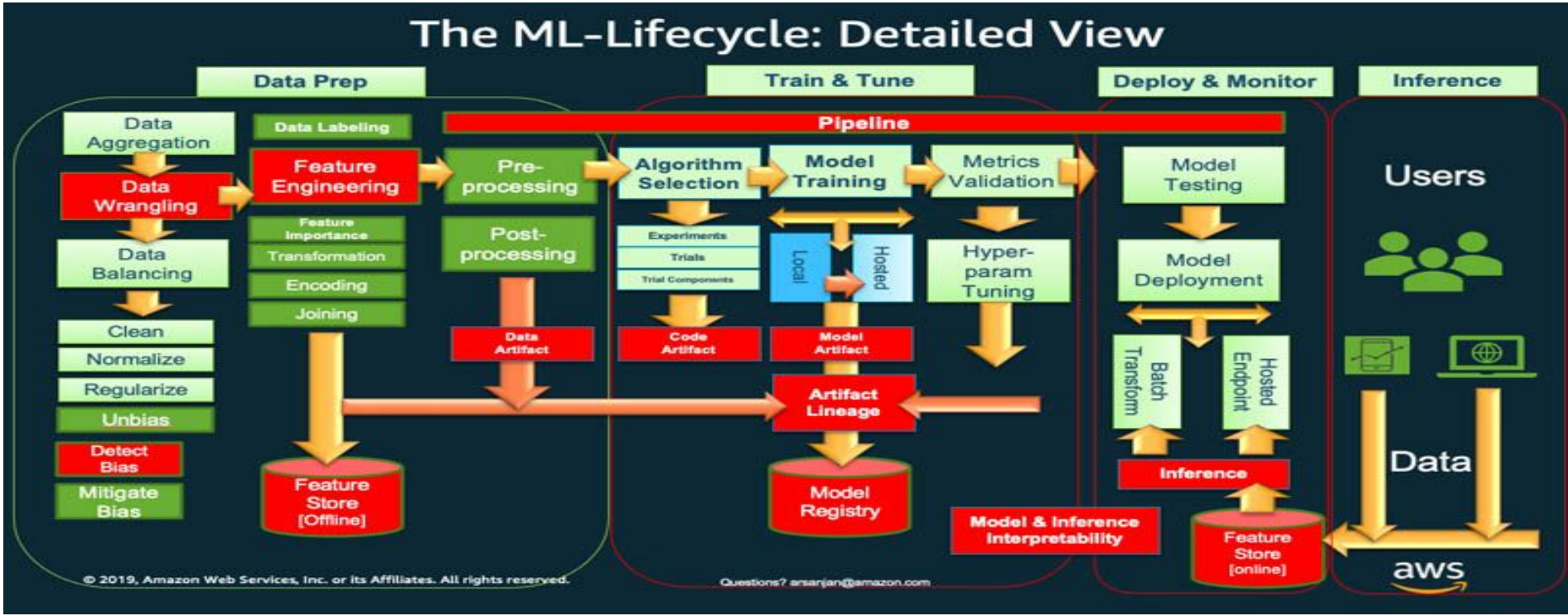
A criação de um processo de MLOps incorpora a metodologia de integração contínua e entrega contínua (CI/CD) do DevOps para criar uma linha de montagem para cada etapa da criação de um produto de aprendizado de máquina.

O que envolve o MLOps

- **Automação**: Automatiza processos para agilizar o desenvolvimento, testes, implantação e retreinamento de modelos.
- **Monitoramento**: Monitora o desempenho dos modelos em produção e detecta desvios ou degradação para garantir que continuem precisos.
- **Reprodução**: Garante que os modelos e os pipelines de treinamento sejam reproduzíveis, o que é crucial para a transparência e conformidade.
- **Colaboração**: Promove a colaboração entre cientistas de dados, engenheiros de ML e equipes de operações (DevOps).

Benefícios do MLOps

- **Eficiência**: Permite o desenvolvimento e a implantação mais rápidos e de maior qualidade de modelos de ML.
- **Escalabilidade**: Permite o gerenciamento e monitoramento de milhares de modelos em escala.
- **Redução de riscos**: Aumenta a transparência e a capacidade de resposta a solicitações de conformidade e auditoria.



https://aws.amazon.com/pt/what-is/mlops/

Ciclo de vida do MLOps

O MLOps opera através de um ciclo contínuo e iterativo, que inclui:

- **Rastreamento de experimentos:** Monitorar experimentos para encontrar os melhores modelos.
- **Implantação de modelos:** Implementar modelos para que possam ser acessados por aplicativos.
- **Monitoramento de modelos:** Observar modelos em produção para detectar falhas no desempenho.
- **Retreinamento de modelos:** Treinar modelos com novos dados para melhorar seu desempenho ao longo do tempo

Principais componentes do MLOps

O MLOps envolve a aplicação de princípios de automação, versionamento e colaboração em todas as fases do ciclo de vida de um modelo de ML.

- **Integração e entrega contínuas (CI/CD):** Estende os princípios de CI/CD para o universo do ML, automatizando o build, o teste e a implantação contínua do modelo.
- **Versionamento de dados, código e modelos:** É fundamental para garantir a reprodutibilidade. Permite rastrear alterações em dados, código e artefatos de modelos, possibilitando reverter para versões anteriores, se necessário.

- **Orquestração de pipelines:** Gerencia o fluxo de trabalho de ponta a ponta, desde a ingestão e preparação de dados até o treinamento e a implantação do modelo.
- **Monitoramento:** Acompanha o desempenho do modelo em produção para detectar desvios de dados (quando a distribuição dos dados de produção muda) ou degradação de desempenho, que podem indicar a necessidade de retreinamento.
- **Treinamento contínuo (CT):** Aciona o retreinamento automático do modelo com novos dados para garantir que ele permaneça preciso e relevante ao longo do tempo.

Ferramentas de MLOps

O ecossistema de ferramentas para MLOps é amplo e em constante evolução, com opções de código aberto e plataformas de nuvem.

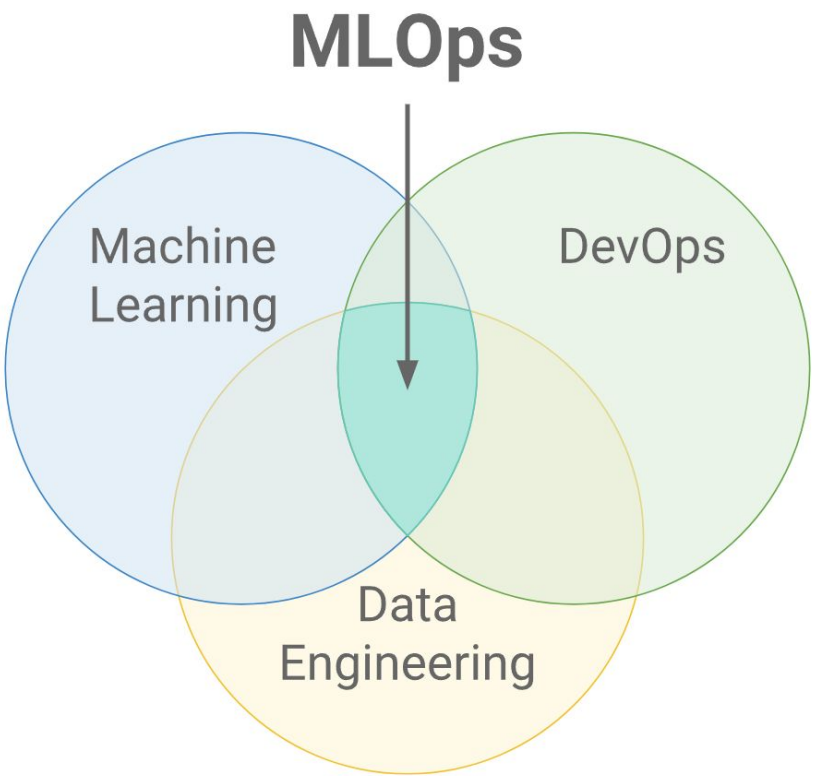
- **Plataformas de nuvem:** Serviços como Google Cloud Vertex AI, Amazon SageMaker e Azure Machine Learning oferecem soluções completas e gerenciadas para todo o ciclo de vida do MLOps.
- **Orquestradores de pipeline:** Ferramentas como o Kubeflow permitem automatizar e orquestrar fluxos de trabalho de ML em clusters Kubernetes.

- **Gerenciamento de experimentos:** O MLflow e o Neptune.ai ajudam a rastrear experimentos, modelos, métricas e artefatos de forma centralizada, facilitando a comparação de resultados.
- **Versionamento de dados:** O DVC (Data Version Control) aplica conceitos semelhantes ao Git para versionar grandes conjuntos de dados e modelos, garantindo a reprodutibilidade.
- **Armazenamento de recursos:** Uma feature store, como o Feast, permite compartilhar e reutilizar recursos de ML de forma consistente em diferentes modelos e equipes.

MLOps vs. DevOps

Embora o MLOps se baseie nos princípios do DevOps, a principal diferença está no escopo e na complexidade do ambiente.

- **Foco:** O DevOps é centrado em código, enquanto o MLOps lida com uma camada adicional de complexidade que inclui dados, modelos e código.
- **Iteração:** Em DevOps, as iterações geralmente envolvem atualizações de código. No MLOps, as iterações podem ser desencadeadas por novas versões de dados, mudanças no código ou degradação do modelo.
- **Testes:** Além dos testes de unidade e integração de código (DevOps), o MLOps requer testes adicionais para validar dados, avaliar a qualidade do modelo e verificar seu desempenho em produção.


MLOps
Machine Learning
DevOps
Data Engineering

Estratégias de Deploy

- Deploy Batch:
  - Características: processamento em lote
  - Use cases: relatórios diários, recomendações semanais
  - Exemplo: sistema de fraud detection
- Deploy em Tempo Real (5min):
  - APIs REST: Flask, FastAPI
  - Microserviços: containerização
  - Exemplo: aprovação de crédito instantânea
- Deploy em Dashboard:
  - Streamlit, Dash, Shiny
  - Use cases: ferramentas internas, demonstrações
  - Exemplo: painel de análise de sentimentos

```python
#Preparação do modelo
import pickle
from sklearn.ensemble import RandomForestClassifier


# Modelo exemplo
model = RandomForestClassifier()
# ... treinamento ...
pickle.dump(model, open('model.pkl', 'wb'))
```

```python
#API com FLASK
from flask import Flask, request, jsonify
import pickle

app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))

@app.route('/predict', methods=['POST'])
def predict():
    data = request.json
    prediction = model.predict([data['features']])
    return jsonify({'prediction': prediction.tolist()})
```

```python
#Dashboard com Streamlit
import streamlit as st
import pickle

model = pickle.load(open('model.pkl', 'rb'))

st.title('Classifier Demo')
features = st.text_input('Enter features (comma-separated)')
if features:
    prediction = model.predict([list(map(float, features.split(',')))])
    st.write(f'Prediction: {prediction[0]}')
```

Monitoramento de Modelos - Drift e Recalibração

1. Introdução ao Monitoramento

- Por que monitorar?:
  - Caso real: modelo de crédito com mudança econômica
  - Impacto business: perda financeira, reputação
  - Regulatory requirements
- O que monitorar? :
  - Data quality: missing values, distributions
  - Model performance: accuracy, precision, recall
  - Business metrics: ROI, conversion rates

Tipos de Drift

- Data Drift:
  - Definição: mudança na distribuição dos dados de entrada
  - Detecção: KS test, PSI, PCA
  - Exemplo: mudança sazonal em e-commerce
- Concept Drift:
  - Definição: mudança na relação features-target
  - Detecção: performance monitoring, DDM (Drift Detection Method)
  - Exemplo: pandemia afetando padrões de compra
- Model Drift:
  - Definição: degradação gradual do modelo
  - Detecção: tracking metrics over time
  - Exemplo: desgaste de modelo de recomendação

Como Detectar Drift

1. **Estatísticas descritivas**
   Compare médias, desvios padrão e distribuições de features entre conjunto de treino e dados atuais.

2. **Testes estatísticos**

   - KS-Test (*Kolmogorov-Smirnov*)
   - Chi-Square
   - PSI (*Population Stability Index*)

3. **Monitoramento contínuo**
   Ferramentas observam os fluxos de dados e disparam alertas quando a distribuição ultrapassa um limite aceitável.

KS-Test (Kolmogorov–Smirnov Test)

O **teste KS** compara **duas distribuições contínuas** (ex: valores numéricos) e mede a **diferença máxima entre as curvas de distribuição acumulada** (CDF – *Cumulative Distribution Function*).

Em outras palavras: ele verifica **quanto uma distribuição "se deslocou" em relação à outra.**

Aplicação

- Detectar *data drift* em **features numéricas contínuas**.
- Avaliar se a distribuição atual ainda segue a mesma forma da original.

Interpretação

- **H_0 (hipótese nula):** as distribuições são iguais.
- **H_1 (alternativa):** as distribuições são diferentes.

```python
###KS
from scipy.stats import ks_2samp
import numpy as np

# Exemplo: feature mudou de média 5 → 6
x_train = np.random.normal(5, 1, 200)
x_prod  = np.random.normal(6, 1, 200)

stat, p_value = ks_2samp(x_train, x_prod)
print(f"KS={stat:.3f}, p-value={p_value:.4f}")

if p_value < 0.05:
    print(" Drift detectado!")
else:
    print(" Distribuições semelhantes.")
```

Chi-Square Test (Qui-quadrado)

O **teste do qui-quadrado** compara **distribuições de frequências categóricas** (por exemplo, proporção de classes, gêneros, regiões, etc.).
Ele mede o **grau de diferença entre as frequências observadas e esperadas**.

Aplicação

- Detectar *drift* em **variáveis categóricas** (nominais).
- Avaliar se a proporção de categorias mudou ao longo do tempo.

Interpretação

- **H_0:** as distribuições categóricas são iguais.
- **H_1:** são diferentes.
- **$p < 0.05$** → existe *drift*.

```python
#Chi-Quadrado
import pandas as pd
from scipy.stats import chi2_contingency

# Frequências de categorias (exemplo: tipos de clientes)
treino = pd.Series(['A', 'A', 'B', 'B', 'C'] * 20)
producao = pd.Series(['A', 'B', 'B', 'C', 'C'] * 20)

# Tabela de contingência
contingencia = pd.crosstab(index=treino, columns=producao)
chi2, p, dof, expected = chi2_contingency(contingencia)

print(f"χ²={chi2:.3f}, p-value={p:.4f}")

if p < 0.05:
   print("Mudança significativa na distribuição categórica.")
else:
   print(" Distribuições semelhantes.")
```

PSI (Population Stability Index)

O **PSI** mede a **diferença entre distribuições de duas populações** (por exemplo, dados de treino vs dados em produção).
Ele é muito usado em **bancos e crédito**, pois permite acompanhar variações graduais ao longo do tempo.

O PSI é calculado com base em **bins (faixas de valor)** da variável e mede o quanto a proporção de cada faixa mudou.

Fórmula

$$PSI = \sum (P_i - Q_i) \times \ln\left(\frac{P_i}{Q_i}\right)$$

Onde:

- P_i = proporção no conjunto de referência (treino)
- Q_i = proporção no conjunto atual (produção)

Valor PSI	Interpretação
< 0.1	Estável
0.1 – 0.25	Atenção (mudança moderada)
> 0.25	Drift significativo

```python
#PSI
import numpy as np
import pandas as pd

def calculate_psi(expected, actual, bins=10):
    expected_perc, _ = np.histogram(expected, bins=bins)
    actual_perc, _ = np.histogram(actual, bins=bins)

    expected_perc = expected_perc / len(expected)
    actual_perc = actual_perc / len(actual)

    psi = np.sum((expected_perc - actual_perc) * np.log((expected_perc +1e-8) / (actual_perc + 1e-8)))
    return psi

# Exemplo de uso
train = np.random.normal(50, 5, 1000)
prod  = np.random.normal(55, 5, 1000)
```

```
psi_value = calculate_psi(train, prod)
print(f"PSI = {psi_value:.3f}")

if psi_value < 0.1:
    print("Estável")
elif psi_value < 0.25:
    print(" Mudança moderada")
else:
    print(" Drift significativo")
```

Métrica	Interpretação	Faixa típica
PSI (Population Stability Index)	Mede a mudança de distribuição entre dois períodos.	PSI < 0.1 = estável; 0.1–0.25 = atenção; >0.25 = drift significativo
Wasserstein Distance	Mede a “distância” entre duas distribuições.	Valores maiores indicam maior drift.

```
#Alternativa sem evidently
import numpy as np
from scipy.stats import ks_2samp
from sklearn.datasets import load_iris
import pandas as pd

iris = load_iris(as_frame=True)
train = iris.frame.sample(100, random_state=42)
test = train.copy()
test['sepal width (cm)'] *= 1.3  # simula drift

# Teste KS (Kolmogorov-Smirnov)
for col in train.columns[:-1]:
   stat, p = ks_2samp(train[col], test[col])
   print(f"{col}: KS={stat:.3f}, p={p:.4f}")
###
###Interpretação:
## Se p < 0.05, há evidência de drift (a distribuição mudou).
## Se p ≥ 0.05, a distribuição é estável.
```

Versionamento e CI/CD

1. Versionamento em ML

- Versionamento de Código:
  - Git basics: commits, branches, tags
  - Git LFS **(Large File Storage)** para dados grandes
  - Exemplo: estrutura de repositório ML
- Versionamento de Dados (5min):
  - DVC (Data Version Control)
  - Delta Lake, LakeFS
  - Exemplo prático com DVC
- Versionamento de Modelos (5min):
  - MLflow Model Registry
  - DVC para modelos
  - Model naming conventions

2. CI/CD para Machine Learning

- Continuous Integration):
  - Testes automatizados: data validation, model tests
  - GitHub Actions exemplo yaml:

```yaml
name: ML Pipeline
on: [push]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Run tests
        run: |
          python -m pytest tests/
          python validate_data.py
```

2. CI/CD para Machine Learning

- Continuous Integration):
  - Testes automatizados: data validation, model tests
  - GitHub Actions exemplo yaml:

```yaml
name: ML Pipeline
on: [push]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Run tests
        run: |
          python -m pytest tests/
          python validate_data.py
```

- Continuous Deployment :
  - Automated training pipelines
  - Model deployment automation
  - Canary deployments para modelos

3. Ferramentas e Boas Práticas

- Ferramentas Populares):
  - MLflow: experiment tracking, model registry
  - Kubeflow: pipelines em Kubernetes
  - Airflow: orchestration
- Boas Práticas):
  - Project structure
  - Documentation
  - Reproducibility checks (**Reprodutibilidade** significa **obter os mesmos resultados** ao repetir um experimento **nas mesmas condições**).

Ética e Vieses em Modelos

1. Ética e Vieses em ML

- Tipos de Vieses:
  - Sample bias: dados não representativos
  - Label bias: problemas na rotulagem
  - Algorithmic bias: viés no algoritmo
- Casos Reais):
  - Hiring algorithms: gender bias
  - Loan approval: racial bias
  - Facial recognition: accuracy disparities

O que é Demographic Parity?

Demographic Parity (Paridade Demográfica) é uma **métrica de fairness** usada para medir se um modelo de machine learning trata **grupos sensíveis** de maneira **equivalente**, independentemente de sua classificação real.

Em outras palavras: todos os grupos têm a **mesma taxa de previsão positiva**, independentemente de diferenças nos dados reais.

Conceito matemático

Seja:

- $\hat{Y}$ = predição do modelo (1 = positivo, 0 = negativo)
- A = grupo sensível (ex: gênero, raça, idade)

A **Demographic Parity** requer:

$$P(\hat{Y} = 1 \mid A = 0) = P(\hat{Y} = 1 \mid A = 1)$$

Ou seja, **a proporção de previsões positivas deve ser igual entre grupos.**

O **Demographic Parity Difference (DPD)** mede **quanto a paridade é violada**, e é definido como:

$$DPD = P(\hat{Y} = 1 \mid A = 0) - P(\hat{Y} = 1 \mid A = 1)$$

- Valor **0** → perfeita paridade demográfica
- Valor **>0** → grupo $A = 0$ recebe mais previsões positivas
- Valor **<0** → grupo $A = 1$ recebe mais previsões positivas

Imagine um modelo de aprovação de empréstimos:

Grupo	Aprovações	Total	Taxa de aprovação
Homens (A=0)	70	100	0.7
Mulheres (A=1)	50	100	0.5

- **Demographic Parity Difference**: 0.7 - 0.5 = 0.2
- Interpretação: homens estão **recebendo mais aprovações** que mulheres — existe **viés**.

```python
import numpy as np

# Previsões do modelo
y_pred = np.array([1,0,1,1,0,1,0,1,0,0])  # 1=positivo
# Grupo sensível: 0=homem, 1=mulher
A = np.array([0,0,0,0,0,1,1,1,1,1])

# Taxas de aprovação por grupo
p0 = y_pred[A==0].mean()
p1 = y_pred[A==1].mean()

dpd = p0 - p1
print("Demographic Parity Difference:", dpd)
```

Interpretação prática

- Quanto **mais próximo de 0**, mais **justo** o modelo em termos de paridade demográfica.
- Valores distantes de 0 indicam **viés no modelo**, mesmo que o desempenho geral (acurácia, F1) seja bom.

Limitações

- Não leva em conta **a realidade das classes reais**: se a distribuição de verdadeiros positivos for diferente entre grupos, um modelo que iguala previsões pode ser **injusto de outra forma**.
- É apenas **uma das métricas de fairness**; outras incluem:
    - **Equal Opportunity** (igualdade de TPR entre grupos)
    - **Equalized Odds** (igualdade de TPR e FPR entre grupos)

```python
#Detecção de Vieses :

from fairlearn.metrics import demographic_parity_difference
import numpy as np

# Exemplo de dados (substitua com seus dados reais)
y_true = np.array([0, 1, 0, 1, 0, 1, 0, 1, 0, 1]) # True labels
y_pred = np.array([0, 0, 0, 1, 0, 1, 1, 1, 0, 0]) # Predicted labels
gender = np.array(['Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male',
'Female', 'Male', 'Female']) # Sensitive feature

# Exemplo de fairness check
dp_diff = demographic_parity_difference(y_true, y_pred, sensitive_features=gender)
print(f"Demographic Parity Difference: {dp_diff}")
```

Mitigação e Governança

- Técnicas de Mitigação :
  - Pre-processing: reweighting, resampling
  - In-processing: fairness constraints
  - Post-processing: threshold adjustment
- Governança de Modelos (5min):
  - Model cards
  - Documentation
  - Audit trails

Mitigar vieses significa **intervir em diferentes momentos do ciclo de vida do modelo para torná-lo mais justo**. As principais técnicas são:

Pre-processing (pré-processamento)

- Atua **antes do treinamento**, ajustando os dados.
- Técnicas comuns:
  - **Reweighting (reponderação):** dá mais peso a exemplos de grupos sub-representados.
  - **Resampling (reamostragem):** aumenta ou diminui a quantidade de exemplos de determinados grupos para equilibrar os dados.
- Exemplo: Se há 80% homens e 20% mulheres em um dataset de crédito, você pode duplicar ou ponderar os exemplos de mulheres para reduzir o viés de gênero.

In-processing (durante o treinamento)

- Intervenção **direta no algoritmo ou na função de perda** para tornar o modelo mais justo.
- Técnicas comuns:
  - **Fairness constraints (restrições de equidade):** adiciona penalidades no modelo quando ele age de forma desigual entre grupos.
- Exemplo: Uma função de perda que penaliza previsões diferentes para grupos demográficos diferentes.

Post-processing (pós-processamento)

- Ajusta **os resultados do modelo depois que ele foi treinado**.
- Técnicas comuns:
  - **Threshold adjustment (ajuste de limiar):** muda o ponto de decisão para equilibrar métricas entre grupos.
- Exemplo: Se um modelo de aprovação de crédito é mais rigoroso com mulheres, você ajusta o limiar de aprovação para equilibrar taxas de aprovação.

Governança de Modelos

Governança trata de **documentar, monitorar e auditar modelos**, garantindo responsabilidade e transparência.

Model Cards

- Documentos que descrevem:
  - Finalidade do modelo
  - Métricas de desempenho
  - Dados utilizados
  - Limitações e potenciais vieses
- Objetivo: **facilitar a compreensão e uso responsável do modelo** por outros times ou usuários.

Documentation (Documentação)

- Inclui:
  - Descrição de features
  - Processo de treinamento
  - Versões de dados e código
- Importante para **reproduzibilidade e manutenção do modelo**.

Audit Trails (Rastro de Auditoria)

- Registro de **quem fez o quê e quando**, como:
  - Alterações no modelo
  - Treinamentos realizados
  - Decisões de deployment
- Permite **investigação de falhas, accountability e compliance**.