Grupo 8

# Projeto Final: Bank Marketing.

**Discentes:**

- Arthur Calado
- Pedro Sarmento
- Gabriel Carvalho

**Docente:** Leandro Maciel Almeida

IF699 - APRENDIZAGEM DE MÁQUINA

# *Otimização de Campanhas de Marketing com Machine Learning*

Este projeto visa aumentar a taxa de conversão em campanhas de telemarketing bancário, utilizando técnicas avançadas de machine learning para identificar clientes com maior probabilidade de realizar depósitos a prazo.

# Etapa 1: Entendimento do Negócio.

# Etapa 1: Entendimento do Negócio.

- **Contexto do Projeto**

  Aumento da eficiência em campanhas de marketing direto focadas em telemarketing para depósitos a prazo.

- **Desafio**

  As campanhas atuais possuem uma baixa taxa de conversão, resultando em desperdício de recursos financeiros e operacionais.

- **Objetivo**

  Desenvolver um modelo preditivo que permita a segmentação da base de clientes, identificando aqueles com maior probabilidade de responder positivamente às campanhas de marketing.

- **Benefício Esperado**

  Aumentar a taxa de conversão e, consequentemente, a rentabilidade das campanhas de marketing, otimizando o uso dos recursos do banco.

# Etapa 2: Entendimento dos Dados.

# Etapa 2: Entendimento dos Dados.

## DATASET

O conjunto de dados utilizado contém 45.211 registros referentes a campanhas de marketing realizadas por um banco, com 16 características para cada cliente.

```
# Verificando as 5 primeiras linhas de x
print(X.head())
```

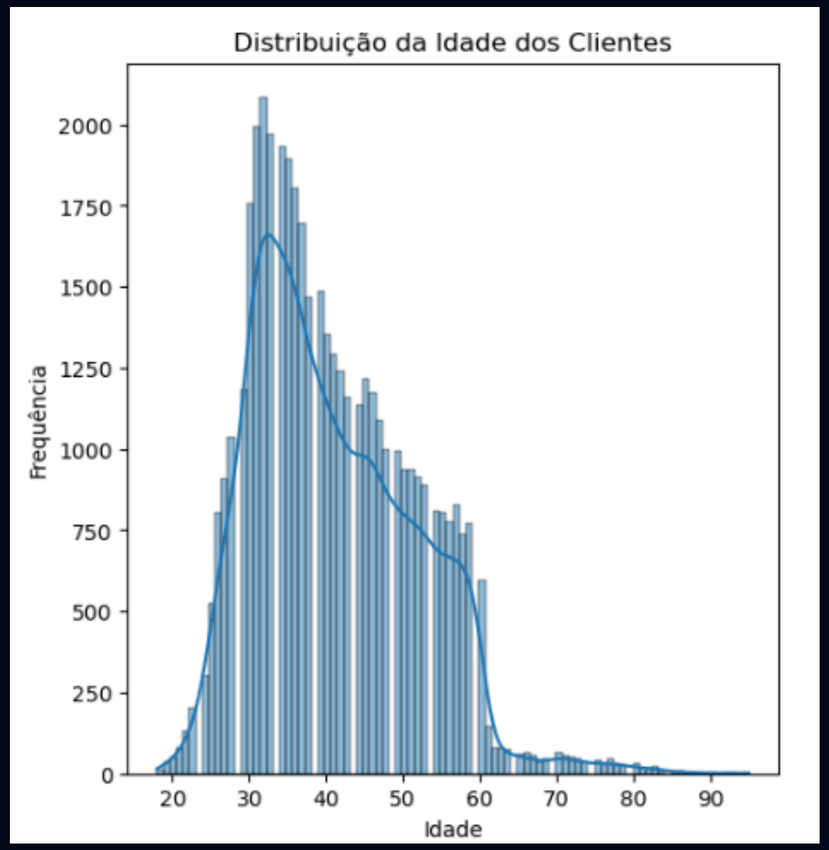
```
   age           job  marital  education default  balance housing loan  \
0   58    management  married   tertiary      no     2143     yes   no
1   44    technician   single  secondary      no       29     yes   no
2   33  entrepreneur  married  secondary      no        2     yes  yes
3   47   blue-collar  married        NaN      no     1506     yes   no
4   33           NaN   single        NaN      no        1      no   no

  contact  day_of_week month  duration  campaign  pdays  previous poutcome
0     NaN            5   may       261         1     -1         0      NaN
1     NaN            5   may       151         1     -1         0      NaN
2     NaN            5   may        76         1     -1         0      NaN
3     NaN            5   may        92         1     -1         0      NaN
4     NaN            5   may       198         1     -1         0      NaN
```

# Etapa 2: Entendimento dos Dados.

## Análise Exploratória: Distribuição de Idade

- Distribuição de Idade: A maioria dos clientes está na faixa etária de 30 a 40 anos
- Impacto: Influencia a estratégia de marketing, indicando um foco em faixas etárias específicas.

```python
# Criação do grafico histograma e boxplot
plt.figure(figsize=(12,6))
plt.subplot(1, 2, 1)
sns.histplot(X["age"], kde=True)
plt.title("Distribuição da Idade dos Clientes")
plt.xlabel("Idade")
plt.ylabel("Frequência")
plt.subplot(1, 2, 2)
sns.boxplot(X["age"], orient='h', notch=True, showcaps=False,
            boxprops={"facecolor": (0, .5, .7, .5)},
            medianprops={"color": "r", "linewidth": 1})
plt.title("Boxplot da Idade dos Clientes")
plt.xlabel("Idade")
plt.show()
```

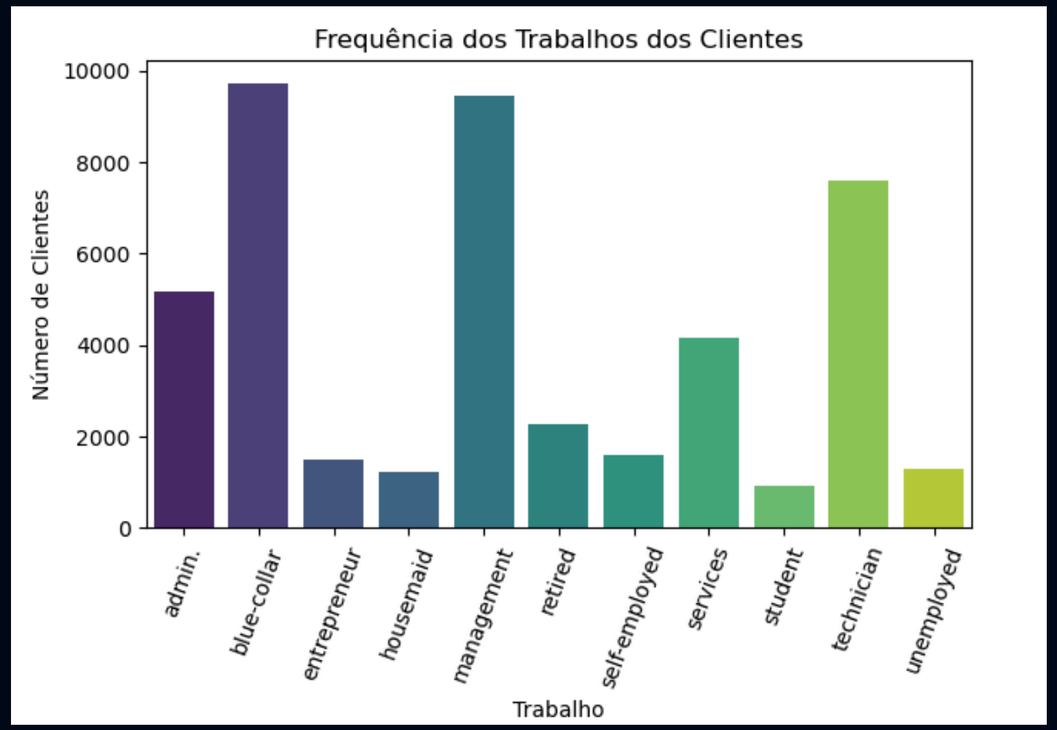


# Etapa 2: Entendimento dos Dados.

## Análise Exploratória: Status de Emprego

- Setores Dominantes: A maioria dos clientes trabalha em setores de gestão, serviços e técnicos.
- Impacto: Sugere um público-alvo com estabilidade financeira, o que pode afetar a receptividade às campanhas de marketing.

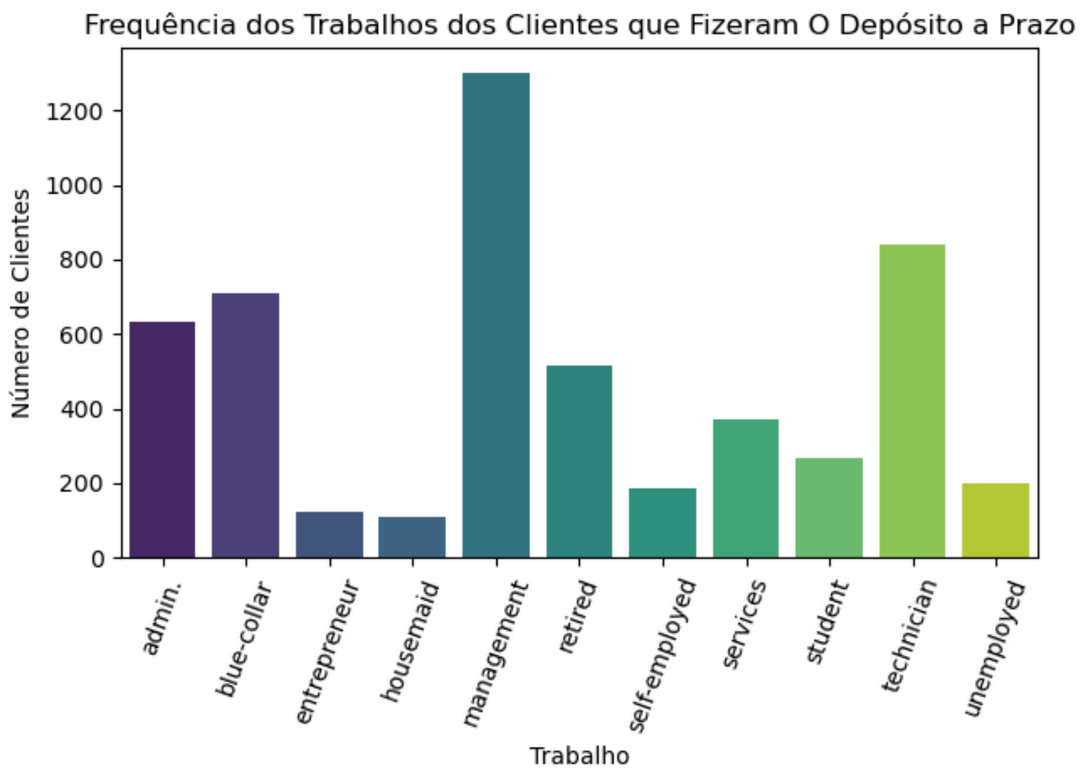
```
Admin.: Administrativo
Blue-collar: Trabalhador manual
Entrepreneur: Empresário
Householdant: Trabalhador doméstico
Management: Gerência
Retired: Aposentado
Self-employed: Autônomo
Services: Serviços gerais
student: Estudante
Technician: Técnico
Unemployed: Desempregado
Unknown: Desconhecido
```



# Etapa 2: Entendimento dos Dados.

## Análise Exploratória: Status de Emprego

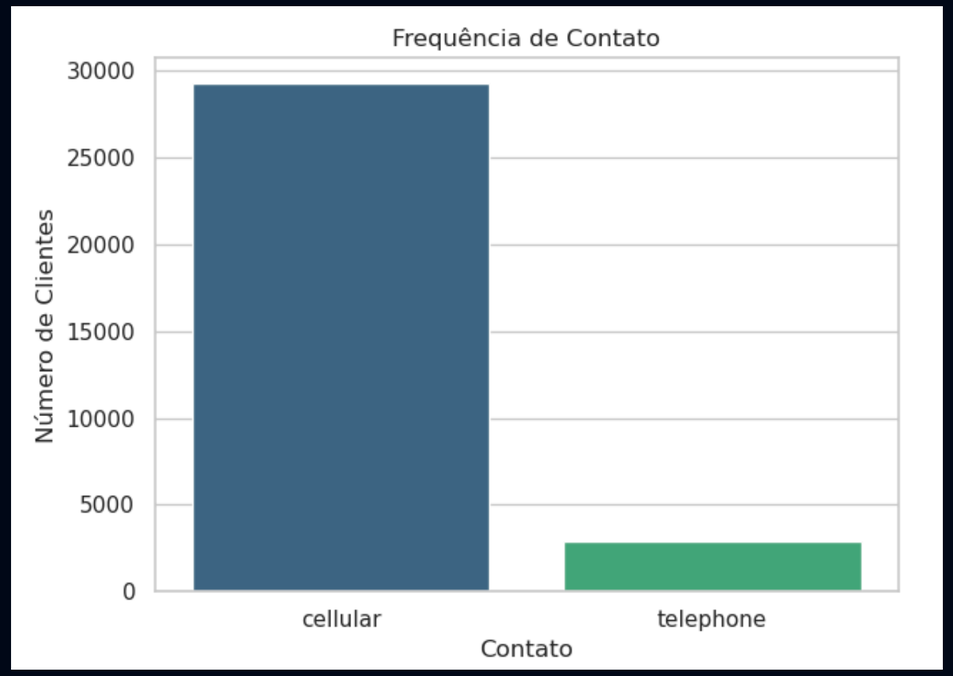
```
# Criando um gráfico de frequência para clientes que aceitaram o produto
sns.barplot(x=job_yes_counts.index, y=job_yes_counts.values, palette='viridis')
plt.title('Frequência dos Trabalhos dos Clientes que Fizeram O Depósito a Prazo')
plt.xlabel('Trabalho')
plt.ylabel('Número de Clientes')
plt.xticks(rotation=70)
plt.tight_layout()
```

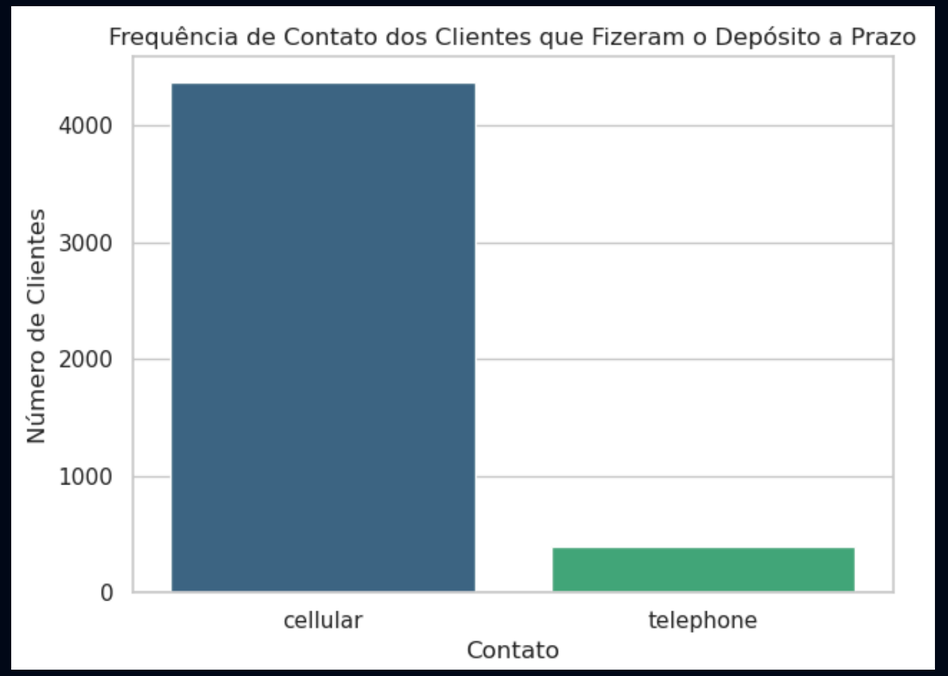


# Etapa 2: Entendimento dos Dados.

## Análise Exploratória: Meio de Contato

- Preferência de Contato: A maior parte dos contatos foi realizada por telefone celular.
- Impacto: Alta taxa de tentativas entre maio e julho sugere períodos de maior receptividade, o que pode ser usado para otimizar a programação das campanhas

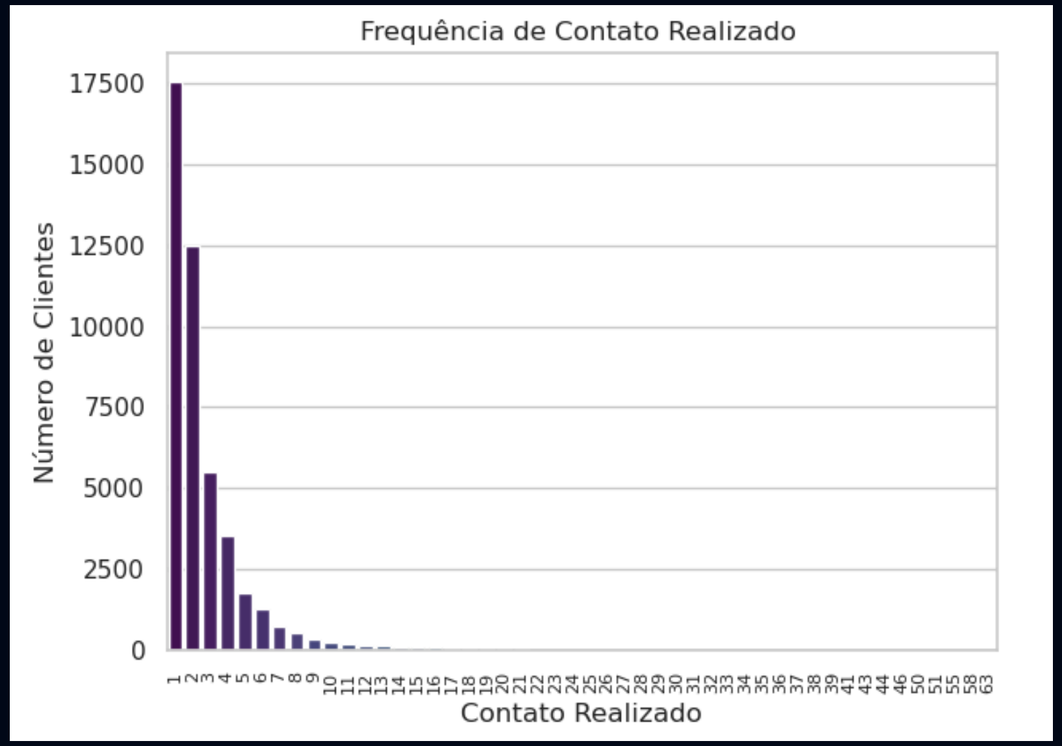


# Etapa 2: Entendimento dos Dados.

## Análise Exploratória: Histórico de Campanhas

- Frequência de Contato: Clientes contatados com maior frequência em campanhas anteriores tendem a apresentar uma probabilidade menor de resposta positiva
- Impacto: Importante para determinar a abordagem ideal nas campanhas futuras, evitando o desgaste do contato excessivo.



# Etapa 3: Preparação dos Dados.

# Etapa 3: Preparação dos Dados.

## Limpeza de Dados

- Identificação e Tratamento de Valores Ausentes: Algumas variáveis, como job, education e contact, continham valores ausentes. Estes foram preenchidos utilizando as categorias mais comuns nessas variáveis. Exemplo: job foi preenchido com a categoria "management" e education com "secondary".

```
filtered_job_bc = X[X['job'] == 'blue-collar'][['balance', 'age']]
filtered_job_mt = X[X['job'] == 'management'][['balance', 'age']]
filtered_job_null = X[X['job'].isnull()][['balance', 'age']].sort_index()

```

```
Xt['job'] = Xt['job'].fillna('management')
```

```
filtered_ed_s = X[X['education'] == 'secondary'][['job', 'age']]
filtered_ed_t = X[X['education'] == 'tertiary'][['job', 'age']]
filtered_ed_null = X[X['education'].isnull()][['job', 'age']].sort_index()

```

```
filtered_ed_s = X[X['contact'] == 'cellular'][['balance', 'age']]
filtered_ed_t = X[X['contact'] == 'telephone'][['balance', 'age']]
filtered_ed_null = X[X['contact'].isnull()][['balance', 'age']].sort_index()
```

```
Xt['contact'] = Xt['contact'].fillna('cellular')
```

# Etapa 3: Preparação dos Dados.

## Limpeza de Dados

- Remoção de Colunas Irrelevantes: A coluna poutcome (resultado de umacampanha de marketing anterior) foi removida por ser considerada irrelevantepara o modelo atual.

```
Xt.drop(columns=['poutcome'], inplace=True)
```

# Etapa 3: Preparação dos Dados.

## Engenharia de Variáveis

- Criação de Novas Variáveis: Novas variáveis foram criadas para capturar interações entre as características existentes, como a criação de uma variável que combina age e balance para capturar a estabilidade financeira em diferentes faixas etárias

```
# Criando variável que vai ter as variáveis transformada
Xt = X.copy()
```

- Conversão de Variáveis Categóricas: Variáveis categóricas como job, marital, e education foram convertidas em variáveis dummy (one-hot encoding) para serem utilizadas nos modelos de machine learning.

```
# Convertendo variáveis categóricas para numéricas usando codificação one-hot
df_encoded = pd.get_dummies(X, columns=['job','marital',
                                        'education','default',
                                        'housing','loan',
                                        'contact','month',
                                        'poutcome'], drop_first=True)

# Matriz de correlação
df_encoded.corr()
```

# Etapa 3: Preparação dos Dados.

## Transformação de Variáveis e Normalização

- Winsorization: Aplicada a técnica de Winsorization para limitar os valores extremos de age e balance, reduzindo o impacto de outliers.
- Normalização Min-Max: As variáveis contínuas, como age e balance, foram normalizadas utilizando a técnica de Min-Max Scaling para garantir que todas as variáveis estivessem na mesma escala (0 a 1).
- Transformação Logarítmica: Variáveis como age e balance passaram por transformação logarítmica para reduzir a assimetria e melhorar a distribuição.

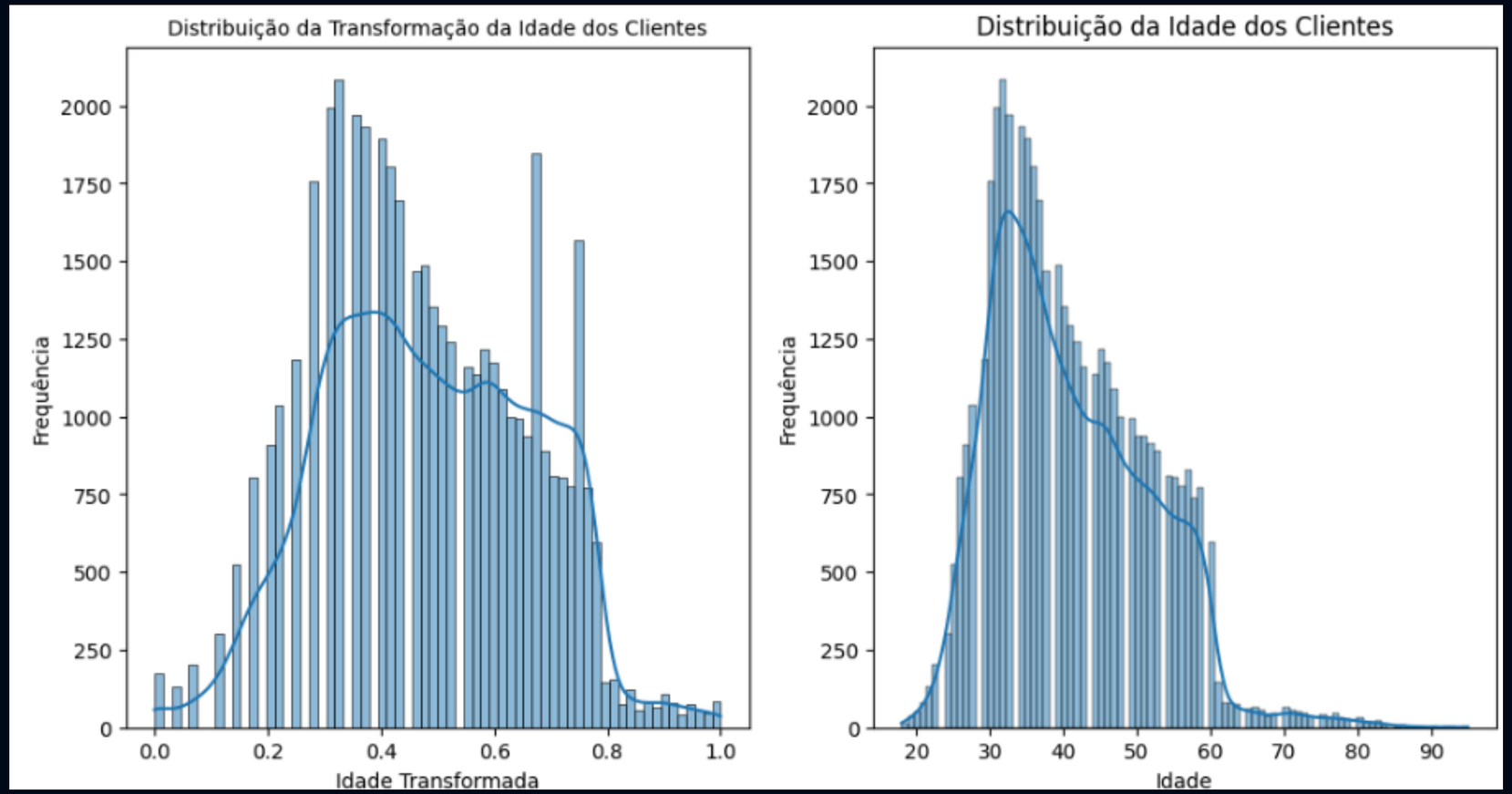
```
Xt['age'] = mstats.winsorize(np.log(X['age'])**(1/2.15), limits=[0.003, 0.001])
Xt['age'] = (Xt['age'] - Xt['age'].min()) / (Xt['age'].max() - Xt['age'].min())
```

```
Xt['balance'] = mstats.winsorize(np.sqrt(X['balance']**2) ** (1/7), limits=[0.1, 0.03])
print(Xt['balance'].min())
print(Xt['balance'].max())
Xt['balance'] = (Xt['balance'] - Xt['balance'].min()) / (Xt['balance'].max() - Xt['balance'].min())
```

```
Xt['duration'] = mstats.winsorize(X['duration']**(1/5), limits=[0.01, 0.02])
print(Xt['duration'].min())
print(Xt['duration'].max())
Xt['duration'] = (Xt['duration'] - Xt['duration'].min()) / (Xt['duration'].max() - Xt['duration'].min())
```

# Etapa 3: Preparação dos Dados.

## Transformação de Variáveis e Normalização



# Etapa 3: Preparação dos Dados.

## Balanceamento das Classes e Divisão de Dados

- Desbalanceamento das Classes: O conjunto de dados apresentava um desbalanceamento significativo entre as classes (clientes que realizaram o depósito versus aqueles que não realizaram).
- Técnica de SMOTE (Synthetic Minority Over-sampling Technique): Utilizada a técnica de SMOTE para gerar exemplos sintéticos da classe minoritária e assim balancear o conjunto de dados.
- Divisão em Conjuntos de Treinamento e Teste: O dataset foi dividido em conjuntos de treinamento (80%) e teste (20%) utilizando a técnica de stratified sampling para garantir que a proporção entre as classes fosse mantida em ambos os conjuntos.

# Etapa 3: Preparação dos Dados.

## Balanceamento das Classes

```
# Construindo conjuntos de treinamento, validação e teste
X_train, X_test, y_train, y_test = train_test_split(X_sample, y_sample, stratify=y_sample, test_size=0.2, random_state

# Dividindo o conjunto de treinamento em conjunto de treinamento e validação
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, stratify=y_train, test_size=0.25, random_state

# Balanceamento do dataset utilizando SMOTE
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)

# Inicializar uma lista para armazenar os resultados de cada classificador
results = []
```

# Etapa 4: Modelagem.

# Etapa 4: Modelagem.

## Seleção dos Modelos

- Modelos testados: Foram testados vários algoritmos de machine learning para encontrar o modelo que melhor se ajusta ao problema. Os modelos testados incluem KNN, LVQ, Decision Tree, SVM, Random Forest, MLP, e Comitês de Modelos
- Critérios de Avaliação: Cada modelo foi avaliado com base em métricas de Acurácia, F1-Score e AUC-ROC, considerando o equilíbrio entre precisão e recall como fatores críticos para selecionar o modelo mais eficaz.

# Etapa 4: Modelagem.

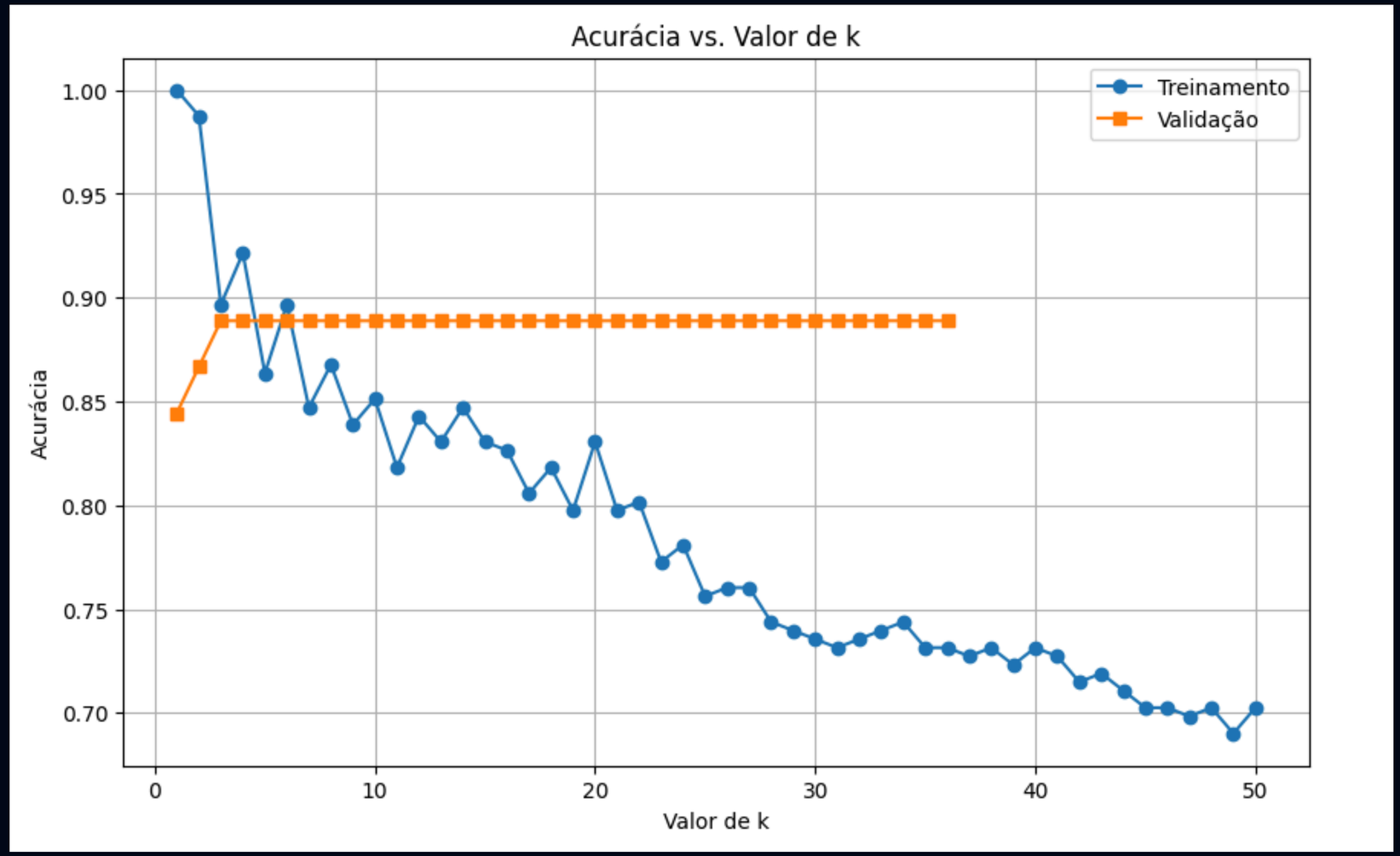
## Modelo K-Nearest Neighbors (KNN)

- Implementação do KNN: O algoritmo KNN foi utilizado para prever a classe dos clientes com base nas características dos vizinhos mais próximos.
- Resultados:
  - Acurácia: 65,22%
  - F1-Score: 0,71
  - AUC-ROC: 0,59
- Discussão: O KNN apresentou um desempenho insatisfatório, com baixa acurácia e F1-Score, o que indica que não é adequado para este problema.

# Etapa 4: Modelagem.

## Modelo K-Nearest Neighbors (KNN)

```
Classificador: kNN
Número de Vizinhos: 3
Acurácia de Treinamento: 89.67%
Acurácia de Validação: 88.89%
Acurácia de Teste: 65.22%
Precision: 81.95%
Recall: 65.22%
F1: 71.50%
AUC-ROC: 58.78%
```



# Etapa 4: Modelagem.

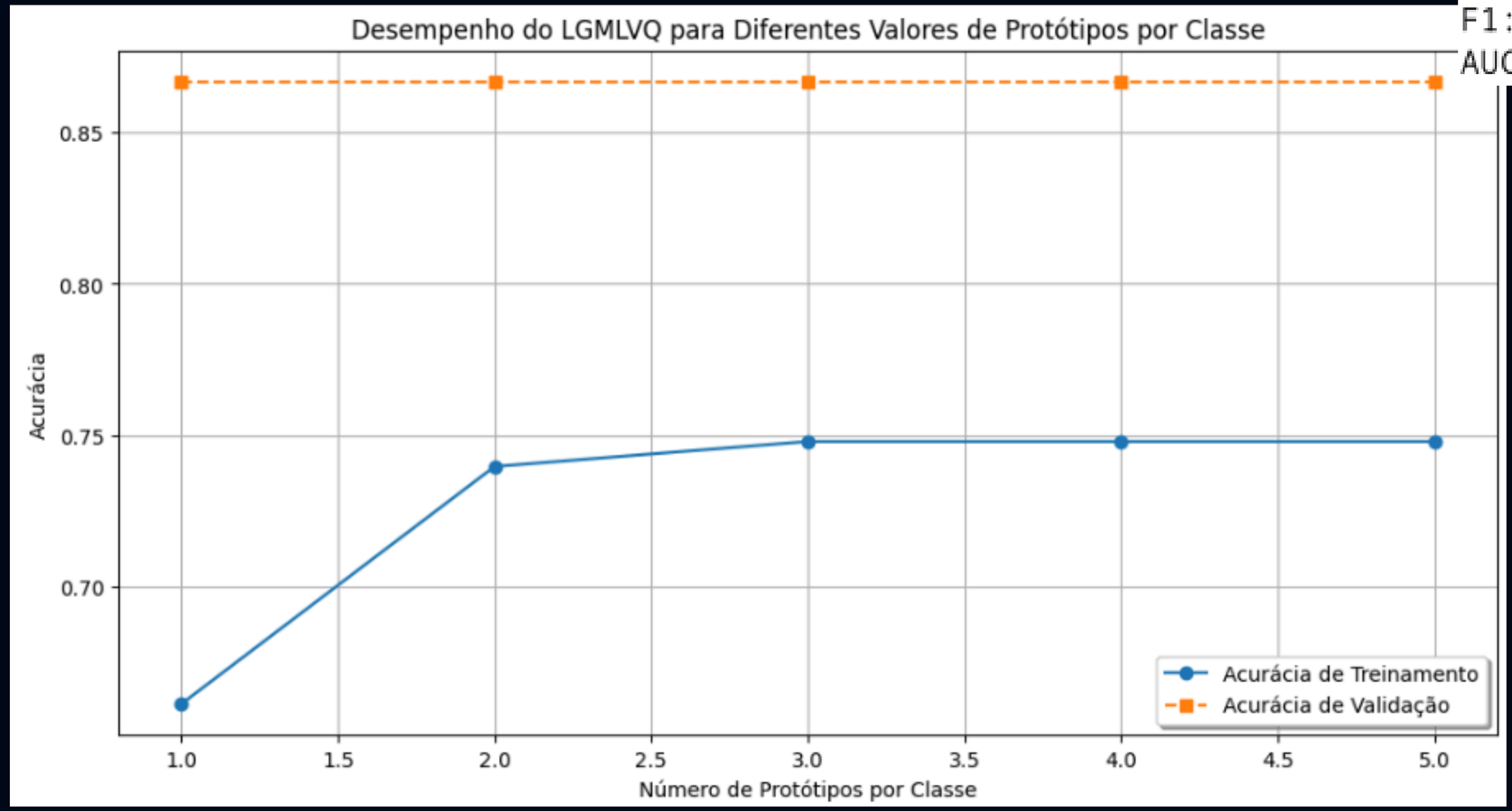
## Modelo Learning Vector Quantization (LVQ)

- Implementação do LVQ: O LVQ é um modelo baseado em redes neurais que foi configurado para classificar os clientes com base em uma representação vetorial
- Resultados:
  - Acurácia: 89,13%
  - F1-Score: 0,54
- Discussão: O LVQ mostrou-se mais eficaz que o KNN, mas o F1-Score ainda é insuficiente para uma boa discriminação entre as classes, especialmente quando comparado a outros modelos mais complexos.

# Etapa 4: Modelagem.

## Modelo Learning Vector Quantization (LVQ)

```
Classificador: LVQ
Protótipos por Classe: 1
Acurácia de Treinamento: 66.12%
Acurácia de Validação: 75.56%
Acurácia de Teste: 89.13%
Precision: 50.00%
Recall: 60.00%
F1: 54.55%
AUC-ROC: 0
```



# Etapa 4: Modelagem.

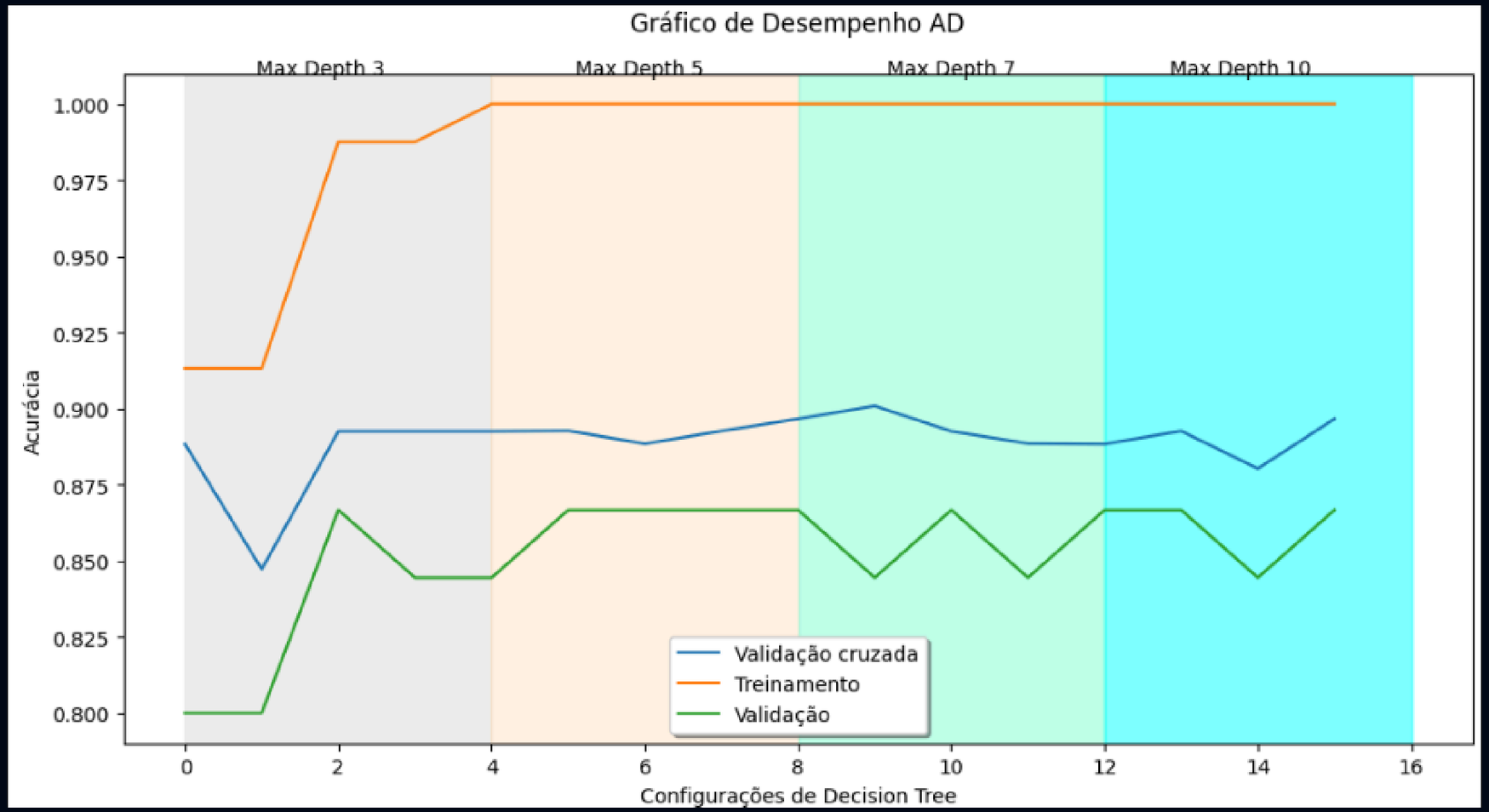
## Modelo Decision Tree

- Implementação do Decision Tree: **Um modelo de árvore de decisão foi testado para entender a hierarquia das características na previsão de respostas dos clientes.**
- Resultados:
  - Acurácia: 86,96%
  - F1-Score: 0,50
  - AUC-ROC: 0,75
- Discussão: O Decision Tree apresentou um desempenho equilibrado, com uma boa acurácia. No entanto, como árvore única, pode ser mais suscetível a overfitting, o que limita sua capacidade de generalização.

# Etapa 4: Modelagem.

## Modelo Decision Tree

```
Classificador: AD
Max Depth: 15
Criterion: entropy
Acurácia de Treinamento: 100.00%
Acurácia de Validação: 84.44%
Acurácia de Teste: 86.96%
Precision: 42.86%
Recall: 60.00%
F1: 50.00%
AUC-ROC: 75.12%
```



# Etapa 4: Modelagem.

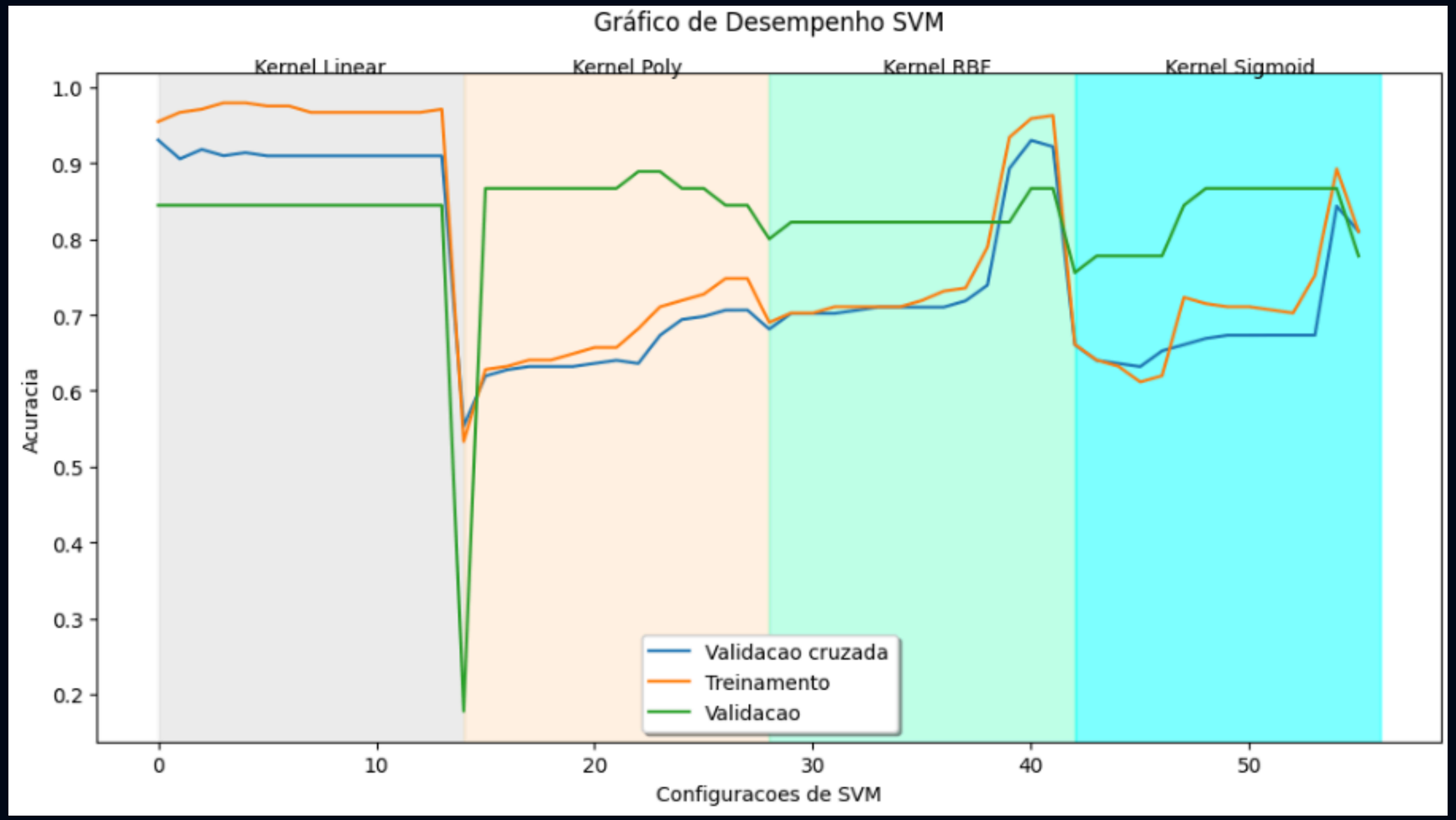
## Modelo Support Vector Machine (SVM)

- Implementação do SVM: O SVM foi testado utilizando o kernel RBF para capturar relações não lineares entre as variáveis.
- Resultados:
  - Acurácia: 89,13%
  - F1-Score: 0,54
  - AUC-ROC: 0,81
- Discussão: O SVM demonstrou um excelente desempenho, especialmente em termos de AUC-ROC, tornando-se uma das opções mais fortes para o modelo final.

# Etapa 4: Modelagem.

## Modelo Support Vector Machine (SVM)

```
Classificador: SVM
Kernel: sigmoid
Acurácia de Treinamento: 66.53%
Acurácia de Validação: 77.78%
Acurácia de Teste: 89.13%
Precision: 50.00%
Recall: 60.00%
F1: 54.55%
AUC-ROC: 80.98%
```



# Etapa 4: Modelagem.

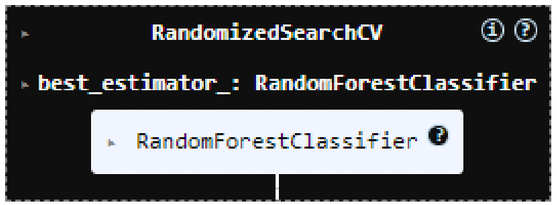
## Modelo Random Forest

- Implementação do Random Forest: O Random Forest foi utilizado por ser um ensemble de árvores de decisão, o que melhora a robustez e a generalização do modelo.
- Resultados:
  - Acurácia: 95,65%
  - F1-Score: 0,75
  - AUC-ROC: 0,90
- Discussão: O Random Forest foi o modelo com melhor desempenho entre todos os testados, proporcionando um ótimo equilíbrio entre precisão e recall, e mostrando-se o mais robusto para este problema.

# Etapa 4: Modelagem.

## Modelo Random Forest

```python
param_dist = {'n_estimators': randint(50,1000),
              'max_depth': randint(1,1000)}

# Create a random forest classifier
rf = RandomForestClassifier(random_state=10)

# Use random search to find the best hyperparameters
rand_search = RandomizedSearchCV(rf,
                                 param_distributions = param_dist,
                                 n_iter=5,
                                 cv=5, random_state=10)

# Fit the random search object to the data
rand_search.fit(X_train, y_train)
```




```
Desempenho da melhor configuração testada:
Classificador: RF
Melhores Hiperparâmetros: {'max_depth': 124, 'n_estimators': 206}
Acurácia de Treinamento: 100.00%
Acurácia de Validação: 95.65%
Acurácia de Teste: 95.65%
Precision: 100.00%
Recall: 60.00%
F1: 75.00%
AUC-ROC: 90.00%
```

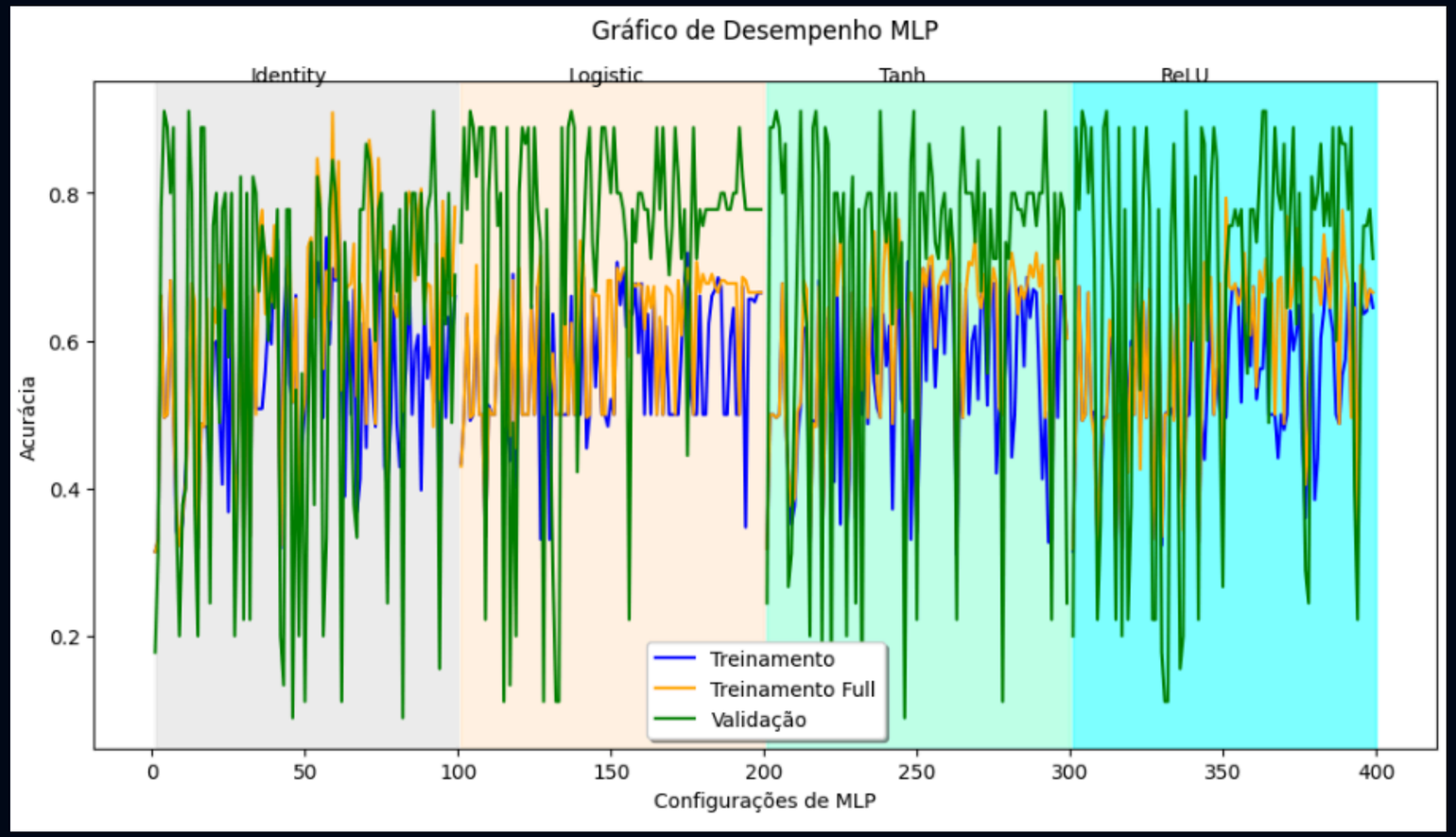
# Etapa 4: Modelagem.

## Modelo Multi-Layer Perceptron (MLP)

- Implementação do Random Forest: O MLP foi testado como um modelo de rede neural feedforward, configurado para múltiplas camadas ocultas
- Resultados:
  - Acurácia: 86,96%
  - F1-Score: 0,0
  - AUC-ROC: 0,0
- Discussão: Embora o MLP tenha oferecido resultados razoáveis, seu desempenho não superou o SVM ou o Random Forest, tornando-o menos adequado para o problema.

# Etapa 4: Modelagem.

## Modelo Multi-Layer Perceptron (MLP)




```
Desempenho da melhor configuração testada:
Classificador: MLP
Ativação: identity
Camadas Ocultas: 4
Acurácia de Treinamento: 49.59%
Acurácia de Validação: 91.11%
Acurácia de Teste: 86.96%
Precision: 0.00%
Recall: 0.00%
F1: 0.00%
AUC-ROC: 18.54%
```

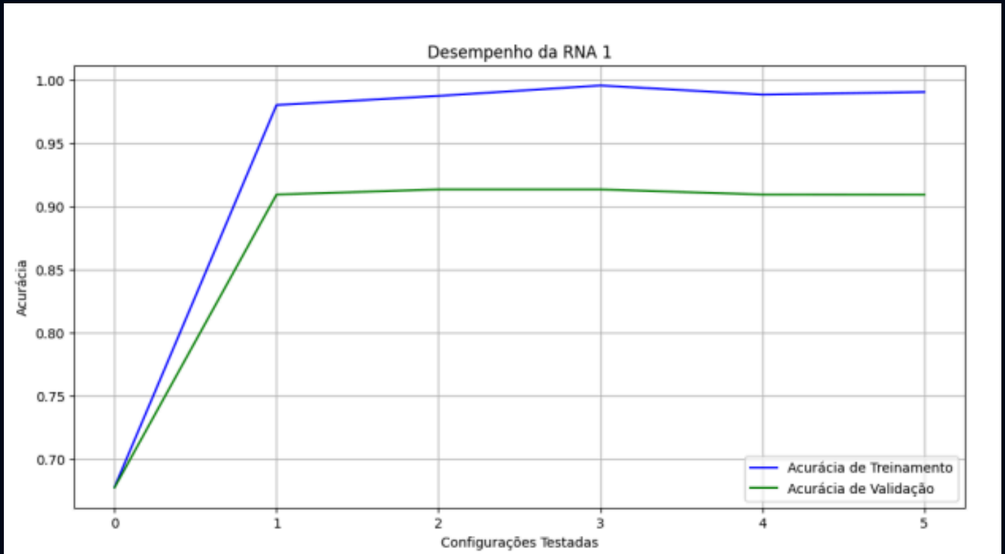
# Etapa 4: Modelagem.

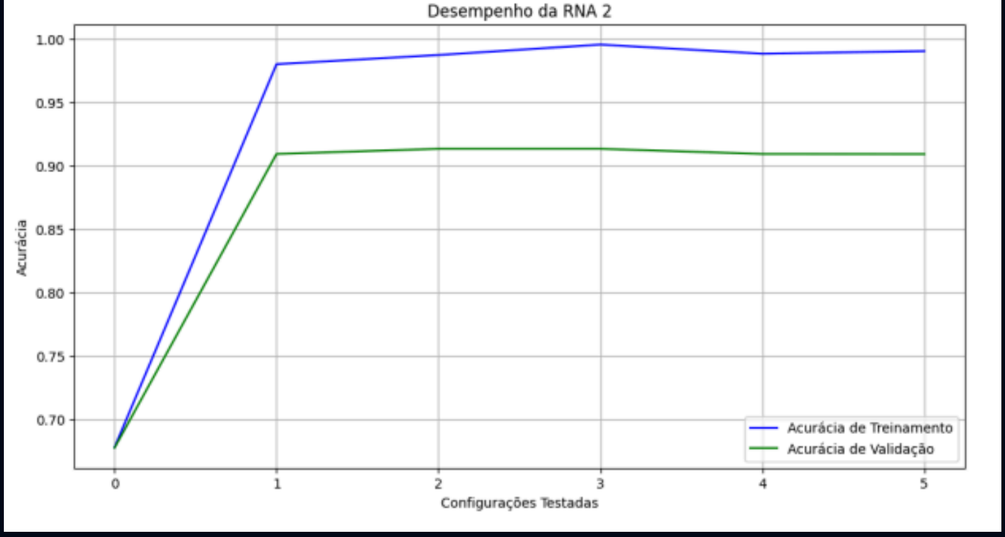
## Comitê de Redes Neurais

- Implementação do Comitê de Redes Neurais: Um comitê de redes neurais (Voting Classifier) foi utilizado para combinar as previsões de diferentes MLPs, visando melhorar a robustez do modelo.
- Resultados:
  - Acurácia: 91,30%
  - F1-Score: 0,33
  - AUC-ROC: 0,90
- Discussão: O Comitê de Redes Neurais demonstrou um desempenho competitivo, mas não superou o Random Forest, sendo uma solução viável, porém não a melhor

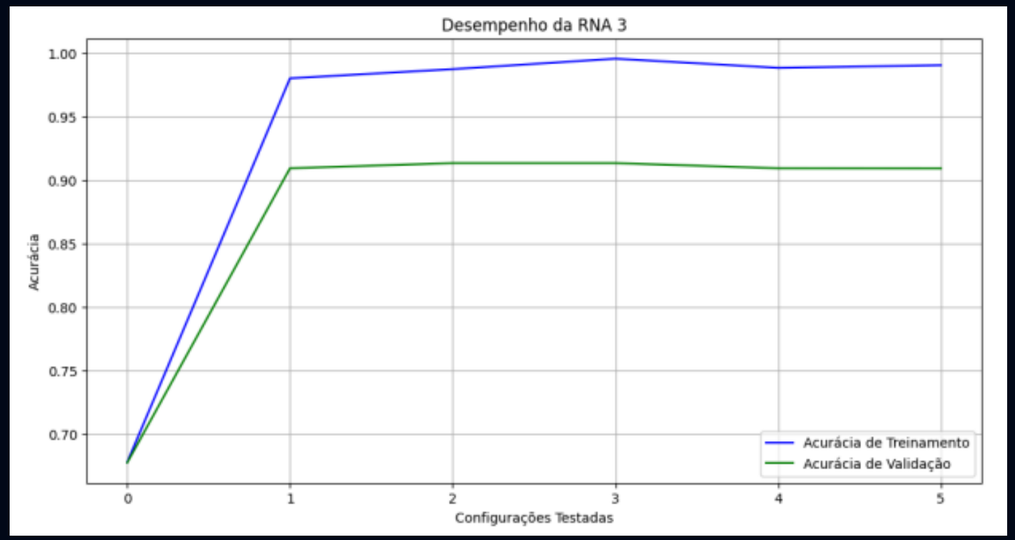
# Etapa 4: Modelagem.

## Comitê de Redes Neurais




```
Desempenho do Comitê de Redes Neurais Artificiais:
Classificador: Comitê RN
Acurácia de Treinamento: 98.35%
Acurácia de Validação: 91.11%
Acurácia de Teste: 91.30%
Precision: 100.00%
Recall: 20.00%
F1: 33.33%
AUC-ROC: 90.24%
```



# Etapa 4: Modelagem.

## Comitê Heterogêneo

- Implementação do Comitê Heterogêneo: : Este comitê combinou diferentes tipos de modelos (Random Forest, SVM, MLP) para aumentar a robustez e a generalização das previsões.
- Resultados:
  - Acurácia: 93,48%
  - F1-Score: 0,67
  - AUC-ROC: 0,88
- Discussão: Embora o Comitê Heterogêneo tenha demonstrado bom desempenho, ainda ficou aquém do Random Forest em termos de precisão e generalização.

```
Desempenho do Comitê Heterogêneo:
Classificador: Comitê Heterogêneo
Acurácia de Treinamento: 97.11%
Acurácia de Validação: 80.00%
Acurácia de Teste: 93.48%
Precision: 75.00%
Recall: 60.00%
F1: 66.67%
AUC-ROC: 87.80%
```

# Etapa 5: Avaliação e Implantação .

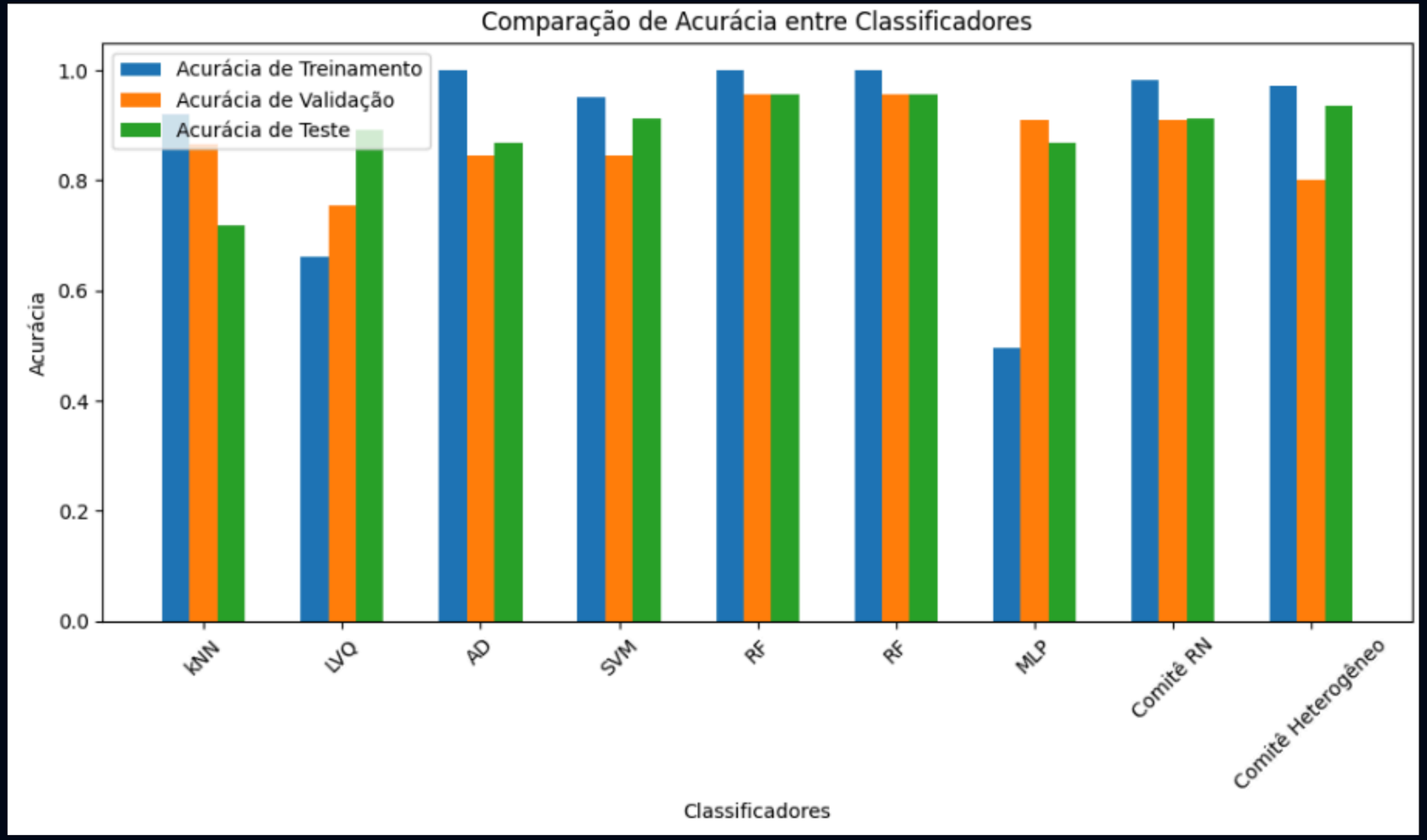
# Etapa 5: Avaliação e Implementação.

## Comparação Final e Escolha do Modelo

- O Random Forest foi selecionado como o modelo mais eficaz com base na comparação das métricas de Acurácia, F1-Score e AUC-ROC.
  - Acurácia: Melhor desempenho geral com 95,65%.
  - F1-Score: Bom equilíbrio entre precisão e recall com 0,75.
  - AUC-ROC: Alta discriminação entre as classes com 0,90.
- O Random Forest ofereceu o melhor equilíbrio entre precisão e recall, mostrando-se o mais robusto e adequado para o problema.

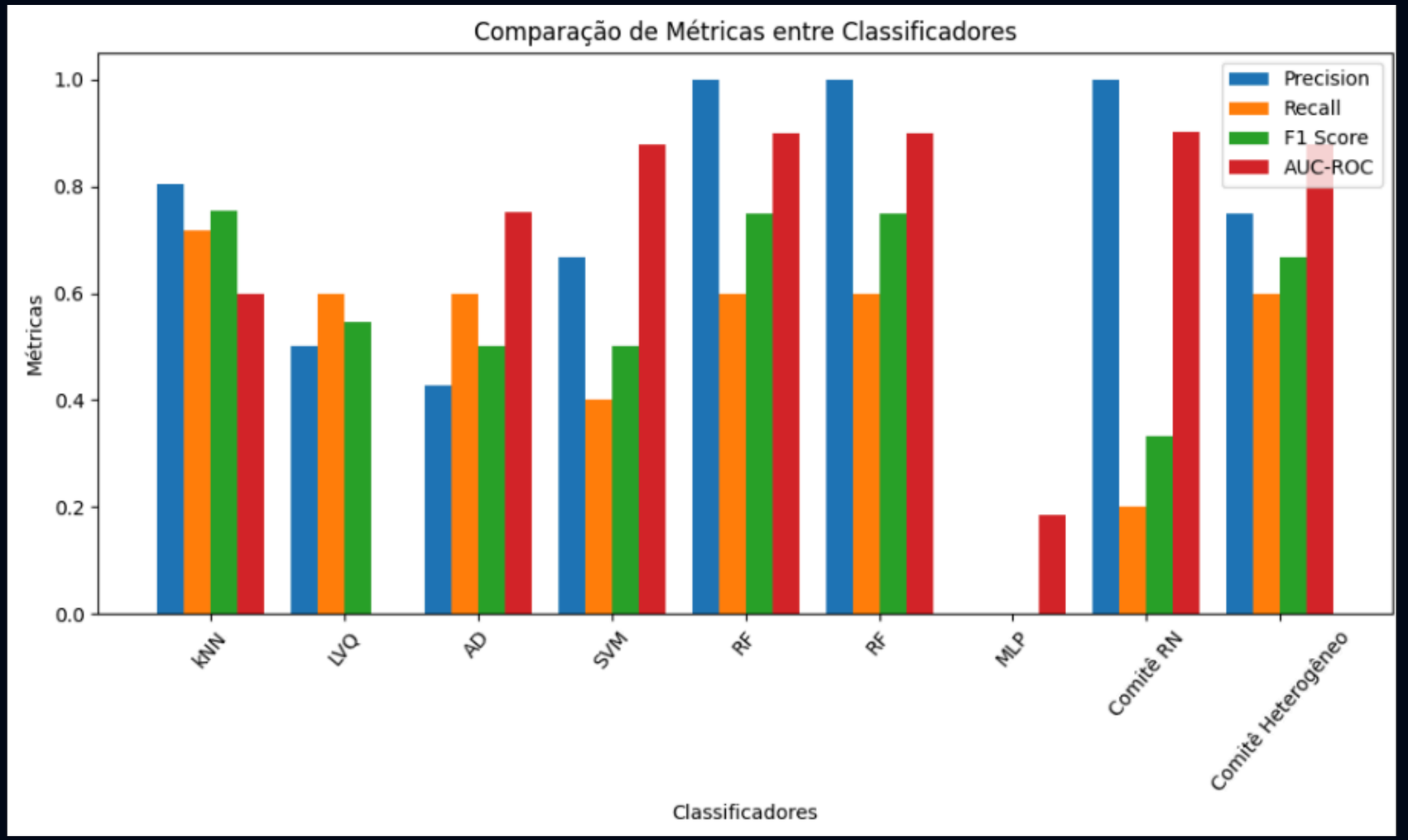
# Etapa 5: Avaliação e Implementação.

## Comparação Final e Escolha do Modelo



# Etapa 5: Avaliação e Implementação.

## Comparação Final e Escolha do Modelo



# Etapa 5: Avaliação e Implementação.

## Utilização do Modelo em Produção

- Exemplo de Implementação: O modelo foi carregado e aplicado a um conjunto de dados de clientes. O resultado das previsões foi utilizado para segmentar os clientes e direcionar as campanhas de marketing de forma mais eficiente.

```python
# Inicializa um contador para clientes propensos
propensos_count = 0

print("Analisando 100 clientes aleatórios...")

for i in range(100):
    # Seleciona um índice aleatório
    random_index = np.random.randint(0, len(X))

    # Seleciona a linha correspondente do DataFrame
    single_row_df = X.iloc[[random_index]]

    # Faz a previsão
    prediction = best_rf.predict(single_row_df)

    # Verifica se a previsão é 1
    if prediction[0] == 1:
        print(f'O cliente com índice {random_index} é propenso a fazer um depósito a prazo.')
        propensos_count += 1

# Imprime o número total de clientes propensos encontrados
print(f'\nTotal de clientes propensos encontrados: {propensos_count} de 100 analisados.')
```

resultado

# Etapa 5: Avaliação e Implementação.

## Utilização do Modelo em Produção

- Exemplo de Implementação: O modelo foi carregado e aplicado a um novo conjunto de dados de clientes. O resultado das previsões foi utilizado para segmentar os clientes e direcionar as campanhas de marketing de forma mais eficiente.

resultado

```
Analisando 100 clientes aleatórios...
O cliente com índice 10403 é propenso a fazer um depósito a prazo.
O cliente com índice 33312 é propenso a fazer um depósito a prazo.
O cliente com índice 29020 é propenso a fazer um depósito a prazo.
O cliente com índice 40618 é propenso a fazer um depósito a prazo.
O cliente com índice 40853 é propenso a fazer um depósito a prazo.
O cliente com índice 43102 é propenso a fazer um depósito a prazo.

Total de clientes propensos encontrados: 6 de 100 analisados.
```

# Etapa 5: Avaliação e Implementação.

## Conclusão e Próximos Passos

- O modelo Random Forest foi o mais eficaz, proporcionando alta precisão e bom equilíbrio entre as métricas
  - Monitoramento contínuo do desempenho do modelo em produção.
  - Ajustes no modelo com a incorporação de novos dados para manter sua eficácia.
  - Exploração de novas técnicas de modelagem para potencialmente melhorar os resultados.

# Obrigado.