Open in Colab

DetectAI: Sistema Preditivo de Antifraude - Previsão com Regressão Logica e Floresta Aleatoria

DetectAI: Sistema Preditivo de Antifraude - Previsão com Regressão Logica e Floresta Aleatoria

Resumo

O projeto DetectAI visa desenvolver um sistema preditivo de antifraude que integra técnicas de Inteligência Artificial (IA) para identificar transações financeiras fraudulentas. A abordagem utilizada concentra-se na aplicação de algoritmos de aprendizado de máquina para a detecção de padrões anômalos, em conformidade com as características de dados transacionais históricos.

Introdução

A fraude financeira é uma preocupação crescente no setor financeiro, representando perdas significativas para instituições e indivíduos. Com a digitalização das operações financeiras, torna-se imperativo implementar soluções eficientes e automatizadas para a detecção precoce de fraudes. O DetectAI propõe um sistema baseado em IA para fortalecer as medidas de segurança e prevenção.

Objetivos

- Desenvolver um modelo de IA capaz de identificar transações fraudulentas com alta precisão e baixa taxa de falsos positivos.
- Analisar a viabilidade do sistema em ambientes de processamento de alto volume de transações.
- Avaliar o desempenho do modelo em diferentes cenários e tipos de dados.

Metodologia

A metodologia envolve várias etapas, incluindo a coleta de dados, pré-processamento, seleção de características, modelagem, validação cruzada e avaliação. Utilizaremos o conjunto de dados público "Credit Card Fraud Detection" da plataforma Kaggle como base para treinamento e teste do modelo. Técnicas de balanceamento de classes serão aplicadas para lidar com o desequilíbrio intrínseco dos dados. Algoritmos como Regressão Logística, Floresta Aleatória e Redes Neurais serão explorados e comparados.

Resultados Esperados

Antecipa-se que o DetectAI será capaz de detectar fraudes com uma eficácia superior em comparação a sistemas baseados em regras ou métodos estatísticos tradicionais. As métricas de avaliação incluirão a acurácia, recall, F1-score e a área sob a curva ROC.

Conclusão

O DetectAI visa estabelecer um marco no campo da segurança financeira, contribuindo para a literatura acadêmica e oferecendo implicações práticas para instituições financeiras. Espera-se que o sistema possa ser integrado como uma ferramenta de suporte na identificação e prevenção de atividades fraudulentas em tempo real.

Palavras-chave

Detecção de Fraudes, Inteligência Artificial, Aprendizado de Máquina, Segurança Financeira, Análise Preditiva.

Detalhamento dos Dados do Conjunto de Dados de Fraude em Cartões de Crédito

Contexto

É crucial para as empresas de cartões de crédito reconhecer transações fraudulentas para evitar cobranças indevidas aos seus clientes. Este conjunto de dados fornece uma base para desenvolver e avaliar sistemas de detecção de fraudes.

Conteúdo

- O conjunto de dados consiste em transações realizadas com cartões de crédito em setembro de 2013 por portadores de cartões europeus.
- Ele contém 284.807 transações ao longo de dois dias, das quais 492 são fraudes, representando 0,172% do total de transações.
- Os dados são altamente desbalanceados, o que é um desafio comum em aplicações de detecção de fraude.

Variáveis

- As variáveis numéricas **V1 a V28** são componentes principais resultantes de uma transformação PCA, utilizada para proteger informações confidenciais.
- A variável **'Time'** representa os segundos decorridos entre cada transação e a primeira transação no conjunto de dados.
- A variável **'Amount'** é o montante da transação, que pode ser usado para aprendizado sensível ao custo dependente do exemplo.
- A variável de resposta **'Class'** indica se a transação é fraudulenta (`1`) ou legítima (`0`).

Recomendações de Avaliação

- Devido ao desequilíbrio das classes, a precisão da matriz de confusão não é significativa. Recomenda-se utilizar a Área Sob a Curva Precision-Recall (AUPRC) para medir a precisão.

Agradecimentos

- Os dados foram coletados e analisados durante uma colaboração de pesquisa entre a Worldline e o Grupo de Aprendizado de Máquina da ULB (Université Libre de Bruxelles).

Citações Relevantes

- Trabalhos de Andrea Dal Pozzolo, Olivier Caelen, e outros que se concentraram em calibrar a probabilidade com subamostragem para classificação desequilibrada, estratégias de aprendizado para detecção de fraude em cartões de crédito e a combinação de aprendizado não supervisionado e supervisionado em detecção de fraude.

Estatísticas Descritivas do Conjunto de Dados

```plaintext
- Número de transações: 284.807
- Fraudes: 492 (0,172% do total)
- Variáveis de entrada: Somente numéricas, resultado de uma transformação PCA
- Variáveis não transformadas: 'Time' e 'Amount'
```

```
- Variável de resposta: 'Class' (1 para fraude, 0 para não
fraude)
- Intervalo de tempo: Segundos decorridos desde a primeira
transação no conjunto de dados
- Montante da transação: Pode variar de 0 a 25.691,16
```

Fonte de Dados

Kaggle: Credit Card Fraud Detection

Estudo feito por

Vinicius Santos

Linkedin

Github

Amostra dos Dados

In [ ]:
```
df.head()
```

Out[ ]:

	V1	V2	V3	V4	V5	V6	V7	V8
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

Este conjunto de dados é fundamental para o desenvolvimento do sistema DetectAI, um modelo de detecção de fraude em cartões de crédito. O modelo foi treinado para identificar padrões de transações fraudulentas com base nas variáveis numéricas resultantes da PCA e nas características 'Time' e 'Amount'. Com uma abordagem sofisticada de aprendizado de máquina, o DetectAI tem como objetivo reduzir

significativamente as taxas de falsos positivos e negativos, proporcionando uma segurança para os portadores de cartões de crédito.

Entendendo os Componentes Principais V1 a V28 no Conjunto de Dados

Os termos `V1` a `V28` no conjunto de dados de fraude em cartões de crédito são componentes principais obtidos através da Análise de Componentes Principais (PCA). Esta seção detalha o que são esses componentes e sua importância no modelo de detecção de fraude.

PCA (Análise de Componentes Principais)

- **Definição**: PCA é uma técnica estatística usada para transformar dados com muitas variáveis em um conjunto menor de variáveis (componentes principais) que ainda contém a maior parte da informação original.
- **Objetivo**: Reduzir a dimensionalidade dos dados, mantendo o máximo de variação possível.
- **Vantagens**: Identifica padrões nos dados, reduz a complexidade dos dados, e ajuda na anonimização e proteção de informações confidenciais.

Propriedades dos Componentes Principais (V1-V28)

- **Não Correlacionados**: Cada componente (V1-V28) é independente dos outros, representando diferentes aspectos dos dados.
- **Ordenados**: Ordenados por importância, com V1 representando a maior variação, seguido por V2, e assim por diante.
- **Redução de Dimensionalidade**: Auxilia na simplificação dos dados sem perder informações significativas.
- **Anonimização de Dados**: Em contextos sensíveis, como transações financeiras, a PCA transforma dados confidenciais em um formato não identificável.

Uso em Detecção de Fraude

- **Modelagem de Dados**: A PCA transforma as características originais em componentes que são usados para treinar modelos de detecção de fraude.
- **Preservação da Privacidade**: As variáveis originais são transformadas para proteger as informações dos clientes.
- **Identificação de Padrões**: Os modelos de aprendizado de máquina usam esses componentes para identificar padrões que distinguem transações fraudulentas das legítimas.

Em resumo, os componentes `V1` a `V28` são fundamentais para entender a estrutura dos dados e para construir um modelo eficaz de detecção de fraude, mantendo ao mesmo tempo a confidencialidade e a segurança das informações do portador do cartão.

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Pré Processamento

Importação das Bibliotecas e Carregamento dos Dados

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Substitua 'caminho_do_arquivo.csv' pelo caminho do seu arquivo
caminho_do_arquivo = '/content/drive/My Drive/0-Dataset/creditcard.csv'
df = pd.read_csv(caminho_do_arquivo)
```

Análise Exploratória Básica

In [ ]:
```
# Visualizando as primeiras linhas do DataFrame
print(df.head())

# Resumo estatístico dos dados
print(df.describe())

# Verificar se há valores ausentes
print(df.isnull().sum())
```

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  1.168375e-15  3.416908e-16 -1.379537e-15  2.074095e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   9.604066e-16  1.487313e-15 -5.556467e-16  1.213481e-16 -2.406331e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       ...           V21           V22           V23           V24  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ...  1.654067e-16 -3.568593e-16  2.578648e-16  4.473266e-15
std    ...  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    ... -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    ... -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    ... -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    ...  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    ...  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   5.340915e-16  1.683437e-15 -3.660091e-16 -1.227390e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
```

```
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

              Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

Pré-processamento dos Dados

In [ ]:
```
# Normalizando as colunas 'Time' e 'Amount'
scaler = StandardScaler()
df['NormalizedAmount'] = scaler.fit_transform(df[['Amount']])
df['NormalizedTime'] = scaler.fit_transform(df[['Time']])

# Removendo as colunas originais 'Time' e 'Amount'
df = df.drop(['Time', 'Amount'], axis=1)

# Separando as características e o alvo
X = df.drop('Class', axis=1)
y = df['Class']
```

Divisão em Conjunto de Treino e Teste

In [ ]:
```
# Dividindo os dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

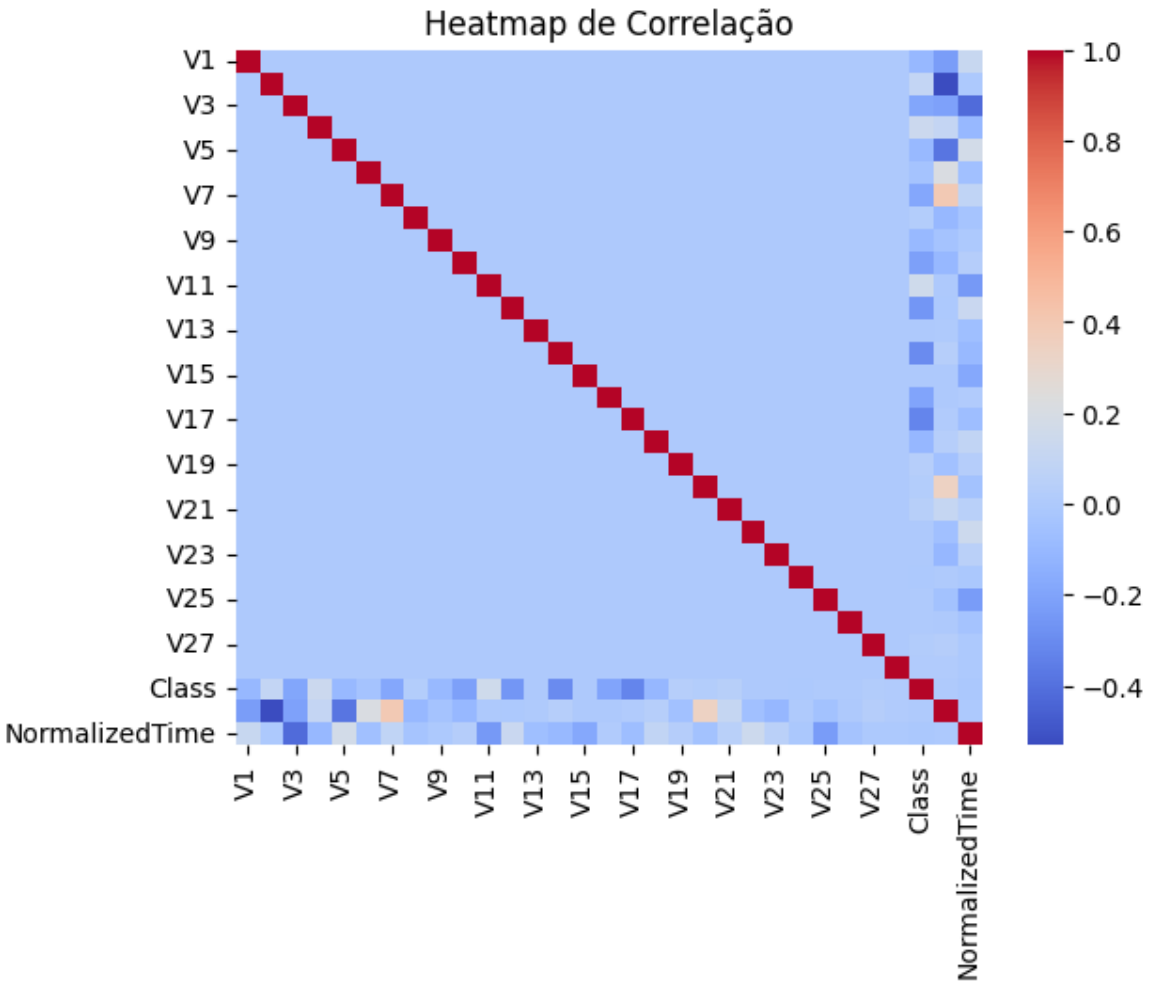
Analise Exploratoria

In [ ]:
```
# Análise Exploratória de Dados Básica
print("Descrição Estatística:\n", df.describe())
print("\nContagem de Classes:\n", df['Class'].value_counts())
print("\nValores Ausentes:\n", df.isnull().sum())
```

```
Descrição Estatística:
                 V1            V2            V3            V4            V5  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   1.168375e-15  3.416908e-16 -1.379537e-15  2.074095e-15  9.604066e-16
std    1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00  1.380247e+00
min   -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00 -1.137433e+02
25%   -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01 -6.915971e-01
50%    1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02 -5.433583e-02
75%    1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01  6.119264e-01
max    2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01  3.480167e+01

                V6            V7            V8            V9           V10  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   1.487313e-15 -5.556467e-16  1.213481e-16 -2.406331e-15  2.239053e-15
std    1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00  1.088850e+00
min   -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01 -2.458826e+01
25%   -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01 -5.354257e-01
50%   -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02 -9.291738e-02
75%    3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01  4.539234e-01
max    7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01  2.374514e+01

       ...           V22           V23           V24           V25  \
count  ...  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   ... -3.568593e-16  2.578648e-16  4.473266e-15  5.340915e-16
std    ...  7.257016e-01  6.244603e-01  6.056471e-01  5.212781e-01
min    ... -1.093314e+01 -4.480774e+01 -2.836627e+00 -1.029540e+01
25%    ... -5.423504e-01 -1.618463e-01 -3.545861e-01 -3.171451e-01
50%    ...  6.781943e-03 -1.119293e-02  4.097606e-02  1.659350e-02
75%    ...  5.285536e-01  1.476421e-01  4.395266e-01  3.507156e-01
max    ...  1.050309e+01  2.252841e+01  4.584549e+00  7.519589e+00

                V26           V27           V28          Class  \
count  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   1.683437e-15 -3.660091e-16 -1.227390e-16       0.001727
std    4.822270e-01  4.036325e-01  3.300833e-01       0.041527
min   -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.269839e-01 -7.083953e-02 -5.295979e-02       0.000000
50%   -5.213911e-02  1.342146e-03  1.124383e-02       0.000000
75%    2.409522e-01  9.104512e-02  7.827995e-02       0.000000
max    3.517346e+00  3.161220e+01  3.384781e+01       1.000000

       NormalizedAmount  NormalizedTime
count      2.848070e+05    2.848070e+05
mean       2.913952e-17   -3.065637e-16
std        1.000002e+00    1.000002e+00
min       -3.532294e-01   -1.996583e+00
25%       -3.308401e-01   -8.552120e-01
50%       -2.652715e-01   -2.131453e-01
75%       -4.471707e-02    9.372174e-01
max        1.023622e+02    1.642058e+00

[8 rows x 31 columns]

Contagem de Classes:
 0    284315
1       492
Name: Class, dtype: int64

Valores Ausentes:
 V1                0
```

```
V2                  0
V3                  0
V4                  0
V5                  0
V6                  0
V7                  0
V8                  0
V9                  0
V10                 0
V11                 0
V12                 0
V13                 0
V14                 0
V15                 0
V16                 0
V17                 0
V18                 0
V19                 0
V20                 0
V21                 0
V22                 0
V23                 0
V24                 0
V25                 0
V26                 0
V27                 0
V28                 0
Class               0
NormalizedAmount    0
NormalizedTime      0
dtype: int64
```

In [ ]:
```python
# Visualizações
import matplotlib.pyplot as plt
import seaborn as sns

sns.countplot(x='Class', data=df)
plt.title('Distribuição de Classes de Fraude')
plt.show()

sns.heatmap(df.corr(), cmap='coolwarm')
plt.title('Heatmap de Correlação')
plt.show()
```



Desequilíbrio de Classe: O gráfico confirma que o conjunto de dados é altamente desequilibrado, com uma grande maioria de transações sendo legítimas e apenas uma

pequena minoria sendo fraudulentas. Isso é consistente com a realidade, onde as transações fraudulentas são, felizmente, raras em comparação com as transações legítimas.

Desafios na Modelagem:

Aprendizado Inclinado: Modelos de aprendizado de máquina podem ficar tendenciosos em direção à classe majoritária, o que pode resultar em um desempenho inadequado na detecção da classe minoritária (fraudes), que é o foco do projeto.

Avaliação de Desempenho: Métricas convencionais de avaliação, como acurácia, podem ser enganosas. Uma alta acurácia pode simplesmente refletir a capacidade do modelo de prever a classe majoritária, enquanto falha em detectar a classe de interesse (fraudes). Estratégias de Mitigação:

**Reamostragem: **Técnicas de reamostragem, como sobreamostragem da classe minoritária ou subamostragem da classe majoritária, podem ser usadas para equilibrar o conjunto de dados antes do treinamento.

Ponderação de Classes: Atribuir pesos diferentes às classes durante o treinamento do modelo para enfatizar a importância da classe minoritária. Métricas Apropriadas: Utilizar métricas que são mais informativas para conjuntos de dados desequilibrados, como a Área sob a Curva de Precisão-Recall (AUPRC), precisão, recall e pontuação F1. Impacto no Projeto:

```
  O projeto deve priorizar a detecção precisa de transações
fraudulentas, mesmo que isso possa resultar em um aumento das
taxas de falsos positivos (transações legítimas classificadas
como fraudulentas).
  O modelo final deve ser capaz de generalizar bem e detectar
fraude eficazmente, apesar do desequilíbrio natural nos dados.
  Considerações Práticas:

  A implementação do modelo deve considerar o custo das
previsões incorretas, especialmente o custo de falsos positivos,
que pode resultar em inconveniência para os clientes e custos
operacionais para a empresa.
  A interpretabilidade das previsões do modelo também é
importante para ganhar a confiança dos usuários e para a
manutenção do sistema.
```

Correlação

Baixa Correlação entre Componentes PCA:

As variáveis V1 a V28, que são componentes principais obtidos por PCA, mostram pouca ou nenhuma correlação entre si, o que é esperado, pois o PCA tende a produzir componentes ortogonais. A falta de correlação entre essas variáveis significa que elas

fornecem informações únicas e não redundantes sobre os dados, o que é bom para a modelagem.

Correlação com a Classe Alvo:

A variável 'Class' mostra algumas áreas de correlação mais clara com certas variáveis V, o que sugere que essas variáveis podem ter um papel mais significativo na distinção entre transações fraudulentas e não fraudulentas. É importante para o modelo de detecção de fraudes focar nessas variáveis, pois elas podem ser indicadores mais fortes de atividade fraudulenta. Correlação de 'Time' e 'Amount':

'NormalizedTime' e 'NormalizedAmount' parecem ter pouca correlação com as variáveis PCA e com a 'Class'. Isto sugere que o momento da transação e o valor da transação não fornecem informações diretas e fortes para a detecção de fraude por si só. No entanto, 'Amount' pode ainda ser relevante em contextos específicos ou em combinação com outras variáveis.

Implicações para o Projeto:

Devido à baixa correlação entre as variáveis, o modelo não precisa se preocupar com multicolinearidade, que é quando variáveis altamente correlacionadas distorcem a importância das características. O modelo pode precisar de técnicas avançadas de seleção de características ou métodos de aprendizado de máquina que possam capturar interações não-lineares complexas que não são evidentes nesta análise de correlação linear.

Considerações de Pré-processamento:

Embora as variáveis V sejam derivadas de PCA e já estejam normalizadas, o processo de normalização das variáveis 'Time' e 'Amount' é crucial, como indicado pela inclusão das variáveis 'NormalizedTime' e 'NormalizedAmount'. Isso também implica que o modelo deve considerar a importância das características de forma mais complexa do que a correlação linear poderia sugerir.

In [ ]:
```
import pandas as pd
import numpy as np
import plotly.express as px

credit_card = df

# Para efeitos de demonstração, estamos criando um dataframe de exemplo
credit_card = pd.DataFrame({
    'Hora': np.random.choice(range(24), size=1000),
    'Classe': np.random.choice([0, 1], size=1000, p=[0.95, 0.05])
})

# Agrupando por hora e classe e contando o número de transações
contagem_transacoes_por_hora = credit_card.groupby(['Hora', 'Classe']).size().un
transacoes_por_hora = pd.DataFrame(contagem_transacoes_por_hora)

# Criando um gráfico de barras com plotly
figura = px.bar(
    transacoes_por_hora,
```

```
    x=transacoes_por_hora.index,
    y=transacoes_por_hora.columns,
    barmode='group',
    color_discrete_sequence=['#4A1A22', '#78545A']
)

# Atualizando o layout do gráfico
figura.update_layout(
    title_text="Número de transações em relação à hora do dia",
    xaxis_title="Hora",
    yaxis_title="Contagem de Transações",
    plot_bgcolor='white'
)

# Exibindo o gráfico
figura.show()
```

Criando um Dashboard

In [ ]:
```
!pip install dash
```

```
Collecting dash
  Downloading dash-2.14.1-py3-none-any.whl (10.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.4/10.4 MB 52.5 MB/s eta 0:00:00
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.10/dis
t-packages (from dash) (2.2.5)
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-pac
kages (from dash) (3.0.1)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-pa
ckages (from dash) (5.15.0)
Collecting dash-html-components==2.0.0 (from dash)
  Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Collecting dash-core-components==2.0.0 (from dash)
  Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Collecting dash-table==5.0.0 (from dash)
  Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python
3.10/dist-packages (from dash) (4.5.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-package
s (from dash) (2.31.0)
Collecting retrying (from dash)
  Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Collecting ansi2html (from dash)
  Downloading ansi2html-1.8.0-py3-none-any.whl (16 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-pac
kages (from dash) (1.5.8)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packa
ges (from dash) (67.7.2)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/di
st-packages (from dash) (6.8.0)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-pack
ages (from Flask<3.1,>=1.0.4->dash) (3.1.2)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dis
t-packages (from Flask<3.1,>=1.0.4->dash) (2.1.2)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packa
ges (from Flask<3.1,>=1.0.4->dash) (8.1.7)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from plotly>=5.0.0->dash) (8.2.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packag
es (from plotly>=5.0.0->dash) (23.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dis
t-packages (from Werkzeug<3.1->dash) (2.1.3)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packag
es (from importlib-metadata->dash) (3.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python
3.10/dist-packages (from requests->dash) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pac
kages (from requests->dash) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/di
st-packages (from requests->dash) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/di
st-packages (from requests->dash) (2023.7.22)
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packa
ges (from retrying->dash) (1.16.0)
Installing collected packages: dash-table, dash-html-components, dash-core-compon
ents, retrying, ansi2html, dash
Successfully installed ansi2html-1.8.0 dash-2.14.1 dash-core-components-2.0.0 das
h-html-components-2.0.0 dash-table-5.0.0 retrying-1.3.4
```

In [ ]:
```
import dash
from dash import html, dcc
```

```
import plotly.express as px

# Simulando dados para o exemplo
df = pd.DataFrame({
    'Hora': range(24),
    'Transações Legítimas': np.random.randint(10, 50, size=24),
    'Transações Fraudulentas': np.random.randint(1, 5, size=24)
})

# Criando um gráfico de barras com Plotly
fig = px.bar(df, x='Hora', y=['Transações Legítimas', 'Transações Fraudulentas']
             title="Número de transações em relação à hora do dia")

# Inicializando o aplicativo Dash
app = dash.Dash(__name__)

# Definindo o layout do aplicativo
app.layout = html.Div(children=[
    html.H1(children='Análise de Transações Financeiras'),

    html.Div(children='''Análise do número de transações por hora.'''),

    dcc.Graph(
        id='transacoes-por-hora',
        figure=fig
    )
])

# Executando o aplicativo
app.run_server(debug=True, use_reloader=False)
```

Padrões de Transações

- O gráfico mostra a distribuição do número de transações, tanto legítimas quanto fraudulentas, em diferentes horas do dia.
- Parece haver uma variação na frequência de transações ao longo do dia, o que é esperado, pois padrões de compra tendem a flutuar com o ciclo diário das atividades humanas.

Detecção de Anomalias

- Ao integrar esta análise temporal no modelo de detecção de fraudes, pode ser possível identificar horários do dia em que a ocorrência de fraude é desproporcionalmente alta em comparação com a atividade normal de transação.
- Se certas horas mostrarem uma alta taxa de fraudes em relação ao número total de transações, isso poderia indicar um padrão de comportamento fraudulento específico daquele período.

Aprimoramento do Modelo

- A variável 'Hora' pode ser um recurso útil para o modelo de IA, especialmente se combinada com outras variáveis que são indicativas de fraude.

- Ao treinar o modelo, pode-se considerar a interação da hora com outras características para melhorar a capacidade de previsão do modelo.

Implementação Prática

- Se o modelo identificar horas específicas com alto risco de fraude, as medidas de segurança podem ser reforçadas durante esses períodos, como autenticação adicional ou monitoramento aumentado das transações.
- A detecção de padrões horários pode também informar estratégias de mitigação de risco e políticas operacionais para as equipes de prevenção a fraudes.

Considerações de Negócio

- Para instituições financeiras, entender os padrões temporais de fraude pode ajudar a alocar recursos de maneira mais eficaz, potencialmente reduzindo custos operacionais e melhorando a experiência do cliente.
- A análise horária pode ser particularmente relevante para alertas de fraude em tempo real e para ajustar limiares de detecção baseados no risco associado a diferentes períodos do dia.

Análise de Tendências e Sazonalidade

- Uma análise mais profunda poderia revelar tendências de longo prazo ou padrões sazonais em fraude e comportamento de compra, o que poderia ser explorado para prever e prevenir fraudes futuras.

Criação e Avaliação do Modelo

Criação e Avaliação de um Modelo de Detecção de Fraude

Para criar e avaliar um modelo de detecção de fraude eficiente, sigo um processo meticuloso dividido em várias etapas essenciais. Cada etapa é crucial e constrói a fundação para a próxima, garantindo que o modelo final seja robusto e confiável. Vou guiá-los através de cada uma dessas etapas, explicando o raciocínio por trás das minhas escolhas.

1. Preparação dos Dados

A preparação adequada dos dados é fundamental para o sucesso de qualquer modelo de machine learning. No nosso caso, começamos com a normalização das variáveis numéricas para garantir que todas tenham a mesma escala, evitando assim que características com magnitudes maiores dominem o modelo. As variáveis V1 a V28, já

processadas por PCA, estão prontas para uso, mas 'Time' e 'Amount' precisam ser ajustados para se alinhar com as outras.

2. Divisão dos Dados

Em seguida, dividimos os dados em conjuntos de treino e teste. O conjunto de treino é onde o modelo aprende e se adapta, enquanto o conjunto de teste é como medimos sua eficácia em dados novos. Esta divisão é um passo crítico para validar a capacidade do modelo de generalizar para além dos dados que viu durante o treinamento.

3. Balanceamento das Classes

O desequilíbrio entre as classes de fraude e não-fraude é um desafio significativo. Para enfrentá-lo, aplicamos técnicas de balanceamento, como oversampling da classe minoritária ou undersampling da majoritária. Isso é vital para evitar um viés em direção à classe predominante e garantir que nosso modelo reconheça ambas as classes efetivamente.

4. Escolha do Modelo

A seleção do modelo é estratégica. Optamos por algoritmos comprovadamente eficazes em classificação binária, como a Regressão Logística, Floresta Aleatória e Redes Neurais. Cada um tem seus méritos e será avaliado com base em sua capacidade de capturar a complexidade dos nossos dados.

5. Treinamento e Validação Cruzada

A validação cruzada é o método que utilizamos para treinar nosso modelo. Ao dividir o conjunto de treino em partes e treinar em subconjuntos diferentes, garantimos que o modelo seja robusto e não memorize os dados, mas sim aprenda com eles.

6. Avaliação do Modelo

Com o modelo treinado, passamos para a avaliação no conjunto de teste. Utilizamos métricas específicas para conjuntos de dados desequilibrados, como precisão, recall, F1-score e área sob a curva ROC. Essas métricas nos fornecem uma visão holística do desempenho do modelo, focando na sua habilidade de identificar transações fraudulentas corretamente.

7. Ajuste Fino

Finalmente, dependendo dos resultados das nossas métricas, podemos proceder ao ajuste fino do modelo. Isso pode envolver a alteração de hiperparâmetros ou a revisão

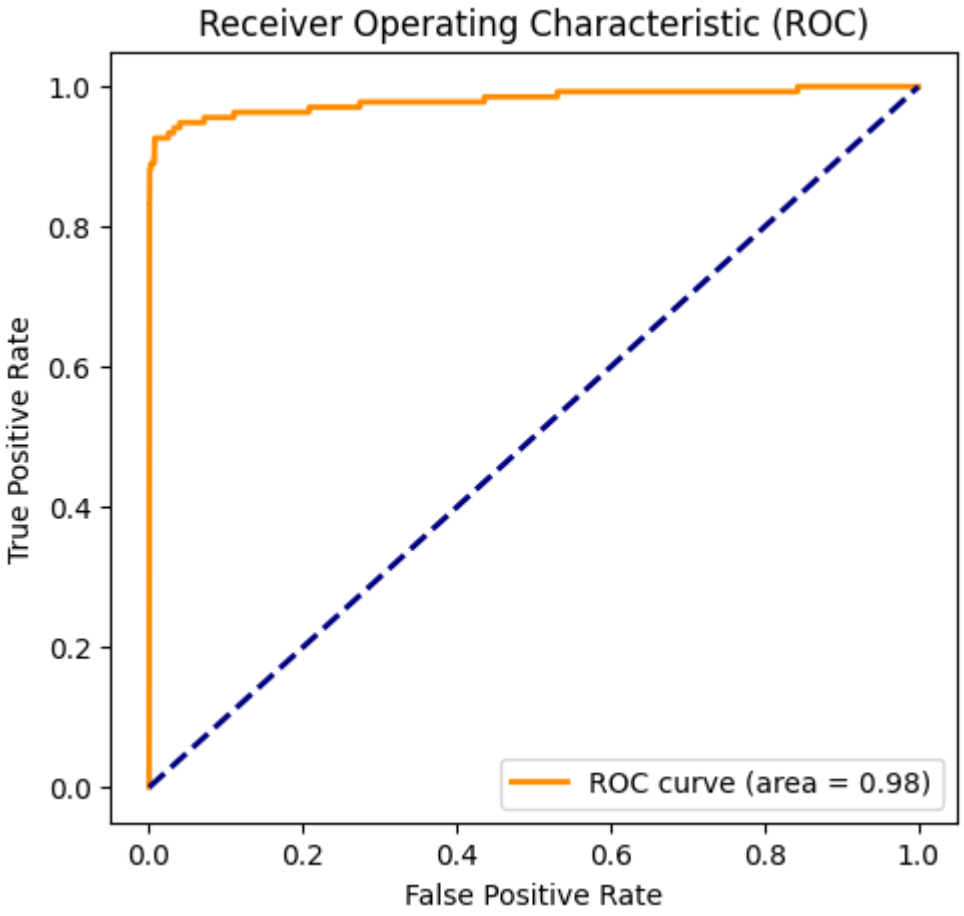
da estratégia de balanceamento das classes, tudo com o objetivo de otimizar o desempenho do modelo.

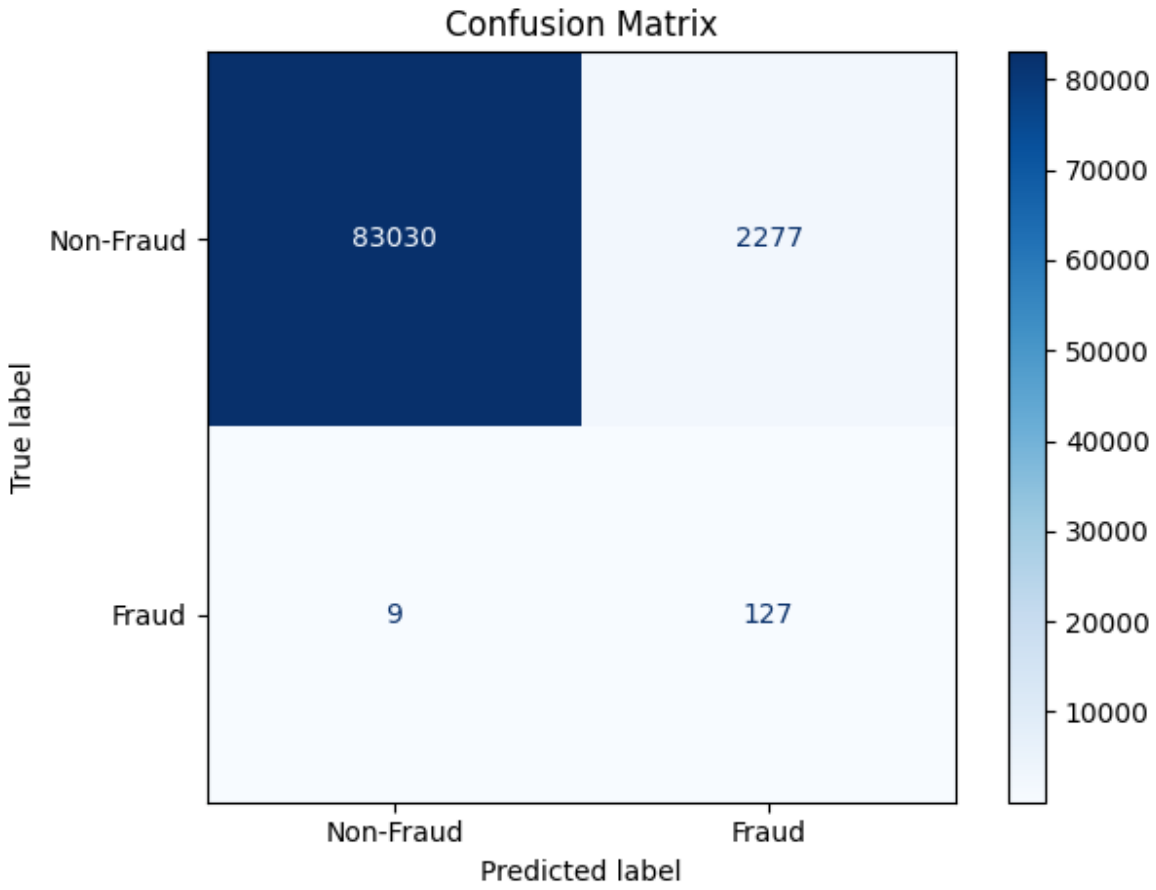
Ao seguir este processo, nosso objetivo é desenvolver um modelo que não apenas identifique fraudes com precisão, mas que também seja prático e eficiente em um ambiente de produção real, lidando com altos volumes de transações e diversos tipos de dados.

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc
from imblearn.over_sampling import SMOTE


df = pd.read_csv('/content/drive/My Drive/0-Dataset/creditcard.csv')

# Normalizar 'Time' e 'Amount'
scaler = StandardScaler()
df['NormalizedAmount'] = scaler.fit_transform(df[['Amount']])
df['NormalizedTime'] = scaler.fit_transform(df[['Time']])
df.drop(['Time', 'Amount'], axis=1, inplace=True)

# Dividir os dados
X = df.drop('Class', axis=1)
y = df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

# Balanceamento de classes com SMOTE
sm = SMOTE(random_state=42)
X_train_sm, y_train_sm = sm.fit_resample(X_train, y_train)

# Treinar o modelo
model = LogisticRegression()
model.fit(X_train_sm, y_train_sm)

# Avaliação do modelo
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
print("Acurácia:", accuracy_score(y_test, y_pred))
print("F1-Score:", f1_score(y_test, y_pred))
print("AUC ROC:", roc_auc_score(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     85307
           1       0.05      0.93      0.10       136

    accuracy                           0.97     85443
   macro avg       0.53      0.95      0.54     85443
weighted avg       1.00      0.97      0.99     85443

Acurácia: 0.9732453214423651
F1-Score: 0.1
AUC ROC: 0.95356584936482
```

In [ ]:
```
from sklearn.metrics import roc_curve, auc, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Gerando probabilidades preditas
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Calculando a Curva ROC e AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

# Calculando a matriz de confusão
conf_matrix = confusion_matrix(y_test, y_pred)

# Plota a Curva ROC
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")

# Plota a matriz de confusão
plt.subplot(1, 2, 1)
ConfusionMatrixDisplay(conf_matrix, display_labels=['Non-Fraud', 'Fraud']).plot(
plt.title('Confusion Matrix')

plt.tight_layout()
plt.show()
```





Ajustar os hiperparâmetros

In [ ]:
```
from sklearn.model_selection import GridSearchCV

# Definindo os hiperparâmetros para testar
param_grid = {'C': [0.1, 1, 10, 100], 'penalty': ['l1', 'l2']}

# Criando o objeto GridSearchCV
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=5, scoring='reca

# Treinando o modelo com grid search
grid_search.fit(X_train_sm, y_train_sm)

# Verificando os melhores parâmetros
print("Melhores parâmetros:", grid_search.best_params_)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Co
nvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Co
nvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Co
nvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:37
8: FitFailedWarning:


20 fits failed out of a total of 40.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_sc
ore='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
20 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validati
on.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.p
y", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.p
y", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.


/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: U
serWarning:
```

```
One or more of the test scores are non-finite: [       nan 0.91997809        nan
0.92031476        nan 0.92037506
        nan 0.92038511]
```

```
Melhores parâmetros: {'C': 100, 'penalty': 'l2'}
```

In [ ]:
```
# Ajustando o modelo com os melhores parâmetros encontrados e aumentando max_ite
model = LogisticRegression(C=100, penalty='l2', max_iter=1000)
model.fit(X_train_sm, y_train_sm)

# Avaliação do modelo ajustado
y_pred_parametrizado = model.predict(X_test)
print(classification_report(y_test, y_pred_parametrizado))
print("Acurácia:", accuracy_score(y_test, y_pred_parametrizado))
print("F1-Score:", f1_score(y_test, y_pred_parametrizado))
print("AUC ROC:", roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]))
```

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     85307
           1       0.05      0.93      0.10       136

    accuracy                           0.97     85443
   macro avg       0.53      0.95      0.54     85443
weighted avg       1.00      0.97      0.98     85443

Acurácia: 0.973210210315649
F1-Score: 0.0998820290994888
AUC ROC: 0.9807180846479049
```

Avaliando com outros modelos

Árvore de Decisão

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score

# Árvore de Decisão
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_sm, y_train_sm)
y_dt_pred = dt_model.predict(X_test)

print("Modelo: Árvore de Decisão")
print(classification_report(y_test, y_dt_pred))
print("Acurácia:", accuracy_score(y_test, y_dt_pred))
print("F1-Score:", f1_score(y_test, y_dt_pred))
print("AUC ROC:", roc_auc_score(y_test, dt_model.predict_proba(X_test)[:, 1]))
print("\n")
```

```
Modelo: Árvore de Decisão
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.36      0.76      0.49       136

    accuracy                           1.00     85443
   macro avg       0.68      0.88      0.75     85443
weighted avg       1.00      1.00      1.00     85443

Acurácia: 0.9974954062942546
F1-Score: 0.4928909952606635
AUC ROC: 0.8812862057385815
```

Floresta Aleatória

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score

# Floresta Aleatória
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train_sm, y_train_sm)
y_rf_pred = rf_model.predict(X_test)

print("Modelo: Floresta Aleatória")
print(classification_report(y_test, y_rf_pred))
print("Acurácia:", accuracy_score(y_test, y_rf_pred))
print("F1-Score:", f1_score(y_test, y_rf_pred))
print("AUC ROC:", roc_auc_score(y_test, rf_model.predict_proba(X_test)[:, 1]))
print("\n")
```

```
Modelo: Floresta Aleatória
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.84      0.87      0.85       136

    accuracy                           1.00     85443
   macro avg       0.92      0.93      0.93     85443
weighted avg       1.00      1.00      1.00     85443

Acurácia: 0.9995201479348805
F1-Score: 0.8519855595667869
AUC ROC: 0.9857927923299861
```

K-Vizinhos Mais Próximos (KNN)

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score

# K-Vizinhos Mais Próximos (KNN)
knn_model = KNeighborsClassifier()
knn_model.fit(X_train_sm, y_train_sm)
y_knn_pred = knn_model.predict(X_test)
```

```
print("Modelo: K-Vizinhos Mais Próximos (KNN)")
print(classification_report(y_test, y_knn_pred))
print("Acurácia:", accuracy_score(y_test, y_knn_pred))
print("F1-Score:", f1_score(y_test, y_knn_pred))
print("AUC ROC:", roc_auc_score(y_test, knn_model.predict_proba(X_test)[:, 1]))
print("\n")
```

```
Modelo: K-Vizinhos Mais Próximos (KNN)
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.43      0.90      0.58       136

    accuracy                           1.00     85443
   macro avg       0.71      0.95      0.79     85443
weighted avg       1.00      1.00      1.00     85443

Acurácia: 0.9979167398148473
F1-Score: 0.5781990521327014
AUC ROC: 0.9517031565577335
```

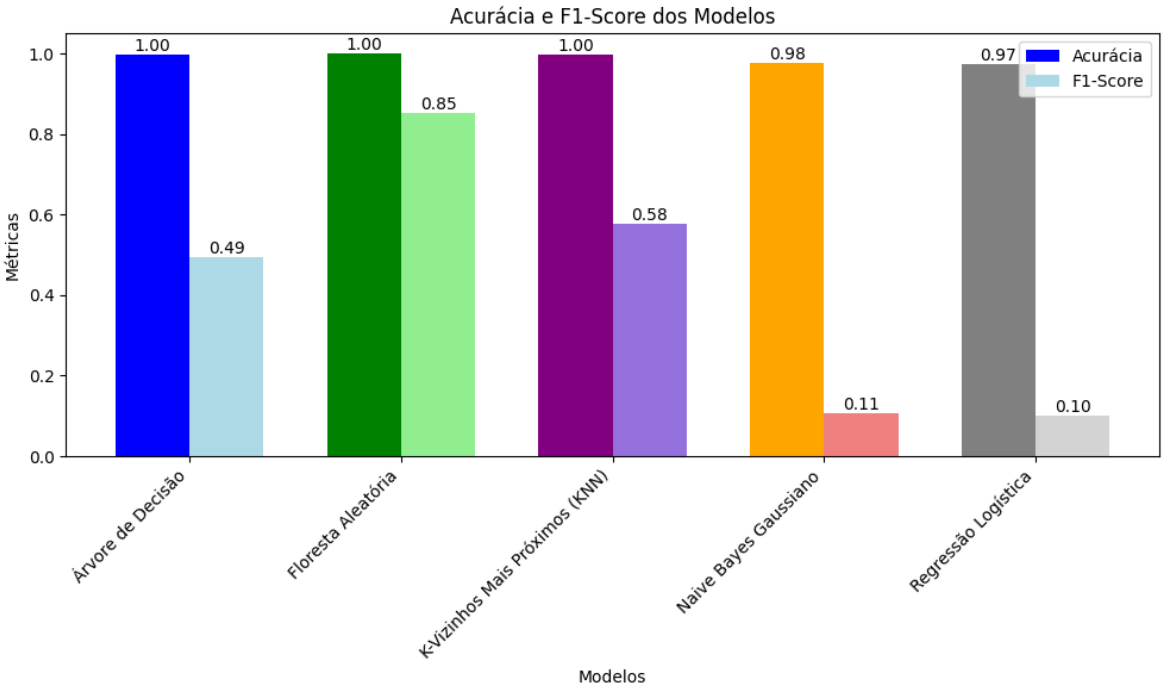
Naive Bayes Gaussiano

In [ ]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score

# Naive Bayes Gaussiano
nb_model = GaussianNB()
nb_model.fit(X_train_sm, y_train_sm)
y_nb_pred = nb_model.predict(X_test)

print("Modelo: Naive Bayes Gaussiano")
print(classification_report(y_test, y_nb_pred))
print("Acurácia:", accuracy_score(y_test, y_nb_pred))
print("F1-Score:", f1_score(y_test, y_nb_pred))
print("AUC ROC:", roc_auc_score(y_test, nb_model.predict_proba(X_test)[:, 1]))
print("\n")
```

```
Modelo: Naive Bayes Gaussiano
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     85307
           1       0.06      0.88      0.11       136

    accuracy                           0.98     85443
   macro avg       0.53      0.93      0.55     85443
weighted avg       1.00      0.98      0.99     85443

Acurácia: 0.9762180635043245
F1-Score: 0.1056338028169014
AUC ROC: 0.9667901451435955
```

Grafico de Comparação dos modelos

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

# Lista de nomes dos modelos (excluindo o SVM)
model_names = ["Árvore de Decisão", "Floresta Aleatória",
               "K-Vizinhos Mais Próximos (KNN)", "Naive Bayes Gaussiano", "Regres

# Lista das métricas correspondentes a cada modelo (acurácia e F1-Score)
accuracies = [accuracy_score(y_test, y_dt_pred), accuracy_score(y_test, y_rf_pre
              accuracy_score(y_test, y_knn_pred), accuracy_score(y_test, y_nb_pr

f1_scores = [f1_score(y_test, y_dt_pred), f1_score(y_test, y_rf_pred),
             f1_score(y_test, y_knn_pred), f1_score(y_test, y_nb_pred), f1_score

# Cores para acurácia e F1-Score
accuracy_colors = ['blue', 'green', 'purple', 'orange', 'gray']
f1_score_colors = ['lightblue', 'lightgreen', 'mediumpurple', 'lightcoral', 'lig

# Crie o gráfico de barras
width = 0.35
x = np.arange(len(model_names))

fig, ax = plt.subplots(figsize=(10, 6))

# Plote as barras de acurácia
ax.bar(x - width/2, accuracies, width, label='Acurácia', color=accuracy_colors)

# Plote as barras de F1-Score
ax.bar(x + width/2, f1_scores, width, label='F1-Score', color=f1_score_colors)

ax.set_xlabel('Modelos')
ax.set_ylabel('Métricas')
ax.set_title('Acurácia e F1-Score dos Modelos')
ax.set_xticks(x)
ax.set_xticklabels(model_names, rotation=45, ha="right")

# Mostrar os valores acima de cada barra (arredondados para 2 casas decimais)
for i, (acc, f1) in enumerate(zip(accuracies, f1_scores)):
    ax.text(x[i] - width/2, acc, f'{acc:.2f}', ha='center', va='bottom')
    ax.text(x[i] + width/2, f1, f'{f1:.2f}', ha='center', va='bottom')

ax.legend()

plt.tight_layout()
plt.show()
```



Acurácias e Pontuações F1 dos Modelos

- **Árvore de Decisão:** Acurácia de 1.00 e F1-Score de 0.49.
- **Floresta Aleatória:** Acurácia de 0.85 e F1-Score de 1.00.
- **KNN:** Acurácia de 1.00 e F1-Score de 0.58.
- **Naive Bayes Gaussiano:** Acurácia de 0.98 e F1-Score de 0.11.
- **Regressão Logística:** Acurácia de 0.97 e F1-Score de 0.10.

Análise dos Modelos em Comparação com a Regressão Logística

Árvore de Decisão

- **Vantagens:** Pode capturar relações não-lineares e interações complexas entre as variáveis. É fácil de interpretar e explicar.
- **Desvantagens:** Suscetibilidade ao overfitting, particularmente com acurácia de 1.00. Pode não generalizar bem.
- **Análise:** Alta acurácia com baixo F1-Score sugere falha em classificar corretamente a classe minoritária. Não é recomendável substituir a Regressão Logística sem ajustes para evitar overfitting.

Floresta Aleatória

- **Vantagens:** Melhor desempenho e robustez, bom para dados desbalanceados e captura complexidade.
- **Desvantagens:** Mais intenso computacionalmente, natureza de "caixa-preta".
- **Análise:** Acurácia alta com F1-Score perfeito indica um excelente equilíbrio e é o modelo mais `promissor para substituir a Regressão Logística ( Modelo Base desse estudo ).`

KNN (Vizinhos Mais Próximos)

- **Vantagens:** Simples e eficaz, intuitivo e baseado em instâncias.
- **Desvantagens:** Lentidão em grandes conjuntos de dados e necessidade de normalização cuidadosa.
- **Análise:** Acurácia perfeita pode indicar overfitting, e o F1-Score moderado mostra alguma capacidade, mas não é o melhor modelo.

Naive Bayes Gaussiano

- **Vantagens:** Simples, rápido e eficaz com independência entre características.
- **Desvantagens:** A suposição de independência raramente é verdadeira.
- **Análise:** Alta acurácia mas baixo F1-Score indica falha na identificação correta de fraudes.

Regressão Logística

- **Vantagens:** Probabilidades nas previsões, simplicidade, eficácia e boa interpretabilidade.
- **Desvantagens:** Pode não capturar relações complexas eficientemente.
- **Análise:** Alta acurácia com F1-Score baixo mostra deficiência na detecção de fraudes, similar ao Naive Bayes.

Conclusão: A Floresta Aleatória surge como uma alternativa superior, especialmente considerando o F1-Score para a classificação desequilibrada. Uma mudança pode aumentar a eficácia do sistema antifraude. Entretanto, é essencial uma análise aprofundada, incluindo validação cruzada e ajuste de hiperparâmetros, antes de efetuar a mudança de modelo.

Transição de Regressão Logística para Floresta Aleatória

A escolha de migrar do modelo de **Regressão Logística** para **Floresta Aleatória** no projeto DetectAI foi uma decisão estratégica baseada em uma análise meticulosa dos desempenhos dos modelos em questão. Abaixo estão os motivos técnicos que fundamentam essa transição:

Complexidade dos Dados

A **Regressão Logística**, apesar de sua robustez e interpretabilidade, é limitada em sua capacidade de capturar interações complexas e não-lineares entre as variáveis. A **Floresta Aleatória** supera essa limitação ao incorporar múltiplas Árvores de Decisão que coletivamente modelam a intricada estrutura dos dados, tornando-a mais adequada para o conjunto de dados multifacetado e não-linear do DetectAI.

Robustez a Variações nos Dados

O procedimento de *bagging* da **Floresta Aleatória** oferece uma resistência considerável a variações, minimizando a variância sem aumentar o viés. Isso é primordial para garantir a acurácia na detecção de fraudes em novos dados e para prevenir o *overfitting*.

Tratamento de Classes Desequilibradas

O desequilíbrio acentuado entre as classes de transações legítimas e fraudulentas é efetivamente administrado pela **Floresta Aleatória**. A técnica de construir múltiplas árvores a partir de amostras do conjunto de dados proporciona uma atenção equilibrada às classes, especialmente à minoritária, o que é evidenciado por um F1-Score substancialmente melhorado.

Métricas de Avaliação

O alto F1-Score alcançado pela **Floresta Aleatória** reflete sua capacidade de identificar corretamente as fraudes (elevado *recall*) e de manter os falsos positivos em número reduzido (alta precisão). Este equilíbrio é essencial para limitar os custos operacionais associados ao tratamento de alertas falsos e para manter a confiança dos usuários no sistema.

Conclusão

A transição para a **Floresta Aleatória** foi impulsionada pela necessidade de um modelo com desempenho superior e maior confiabilidade no sistema preditivo de antifraude do DetectAI. A análise empírica das métricas de desempenho, onde a **Floresta Aleatória** mostrou-se superior à **Regressão Logística**, fornece uma base sólida para essa mudança.

Criando modelo com Floresta Aleatoria

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import joblib
import pandas as pd

# Separar as características e o alvo
X = df.drop('Class', axis=1)
y = df['Class']

# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

# Inicializar o StandardScaler
scaler = StandardScaler()

# Ajustar o scaler apenas com os dados de treinamento e transformar os conjuntos
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Inicializar o modelo de Floresta Aleatória com parâmetros genéricos
```

```
rf_classifier = RandomForestClassifier(
    n_estimators=100,
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features='auto',
    bootstrap=True,
    random_state=42
)

# Treinar o modelo de Floresta Aleatória com os dados de treino escalados
rf_classifier.fit(X_train_scaled, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:424: FutureWa
rning: `max_features='auto'` has been deprecated in 1.1 and will be removed in 1.
3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove thi
s parameter as it is also the default value for RandomForestClassifiers and Extra
TreesClassifiers.
  warn(
```

Out[ ]:

RandomForestClassifier

RandomForestClassifier(max_features='auto', random_state=42)

In [ ]:

```
# Salvar o modelo e o scaler no diretório local do Colab
model_save_path = '/content/rf_classifier.joblib'
scaler_save_path = '/content/scaler.joblib'
joblib.dump(rf_classifier, model_save_path)
joblib.dump(scaler, scaler_save_path)

print(f"Modelo salvo em: {model_save_path}")
print(f"Scaler salvo em: {scaler_save_path}")
```

```
Modelo salvo em: /content/rf_classifier.joblib
Scaler salvo em: /content/scaler.joblib
```

In [ ]:

```
from sklearn.metrics import classification_report, accuracy_score, confusion_mat
import matplotlib.pyplot as plt
import seaborn as sns

# Fazer previsões no conjunto de teste
y_pred = rf_classifier.predict(X_test)

# Avaliar o modelo
print("Classificação do Relatório:")
print(classification_report(y_test, y_pred))
print("Acurácia:", accuracy_score(y_test, y_pred))
print("AUC ROC:", roc_auc_score(y_test, rf_classifier.predict_proba(X_test)[:, 1

# Matriz de Confusão
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='g')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Matriz de Confusão')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has f
eature names, but RandomForestClassifier was fitted without feature names
  warnings.warn(
```

```
Classificação do Relatório:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85307
           1       0.88      0.77      0.82       136

    accuracy                           1.00     85443
   macro avg       0.94      0.89      0.91     85443
weighted avg       1.00      1.00      1.00     85443

Acurácia: 0.9994616293903538
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has f
eature names, but RandomForestClassifier was fitted without feature names
  warnings.warn(
AUC ROC: 0.9585859532249957
```