Projeto UC – Inteligência Artificial

Professores Responsáveis:

- Cleber Silva
- Vinicius

Grupo:

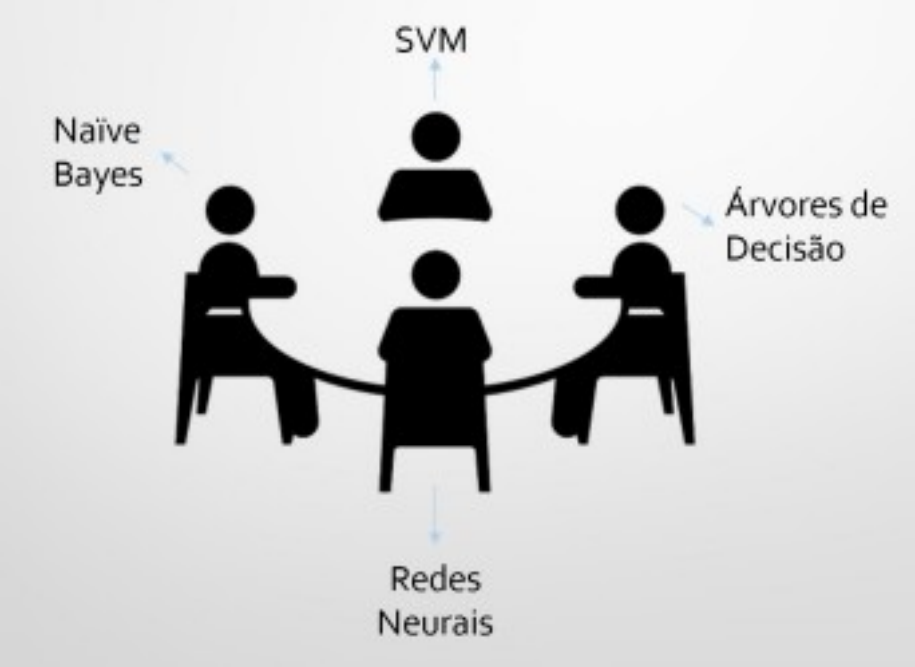
Aluno1: Jorge Leandro Piva RA: 820268722

Aluno2: Valério Vanuzi Pereira RA: 32213195

O que será o projeto?

O projeto da UC de Inteligência Artificial será fazer um Comitê de Classificadores

Ou seja, cada grupo analisará uma base de dados e aplicará métodos de IA para chegar a algum resultado, após a aplicação desses métodos, deverá compará-los para chegar à conclusão de qual teve o melhor resultado, melhor custo benefício, etc.



Quais os métodos que poderão ser aplicados?

- Árvores de Decisão
- Naive Bayes
- Redes Neurais
- SVM
- KNN
- etc

Base de dados

Foi escolhida um dataset no Kaggle sobre Fraudes bancárias. Esta UC está sendo ministrada simultaneamente com a UC de Análise de dados e Big Data, desta forma utilizamos o mesmo dataset para as duas atividades A3, porém, estamos utilizando técnicas de análise exploratória com Pandas e Numpy além de criação de alguns gráficos com Matplotlib e Seaborn para facilitar algumas visualizações, estas técnicas nos permite conhecer melhor os dados que serão utilizados nos algoritmos.

O dataset pode ser encontrado no site do Kaggle no link.

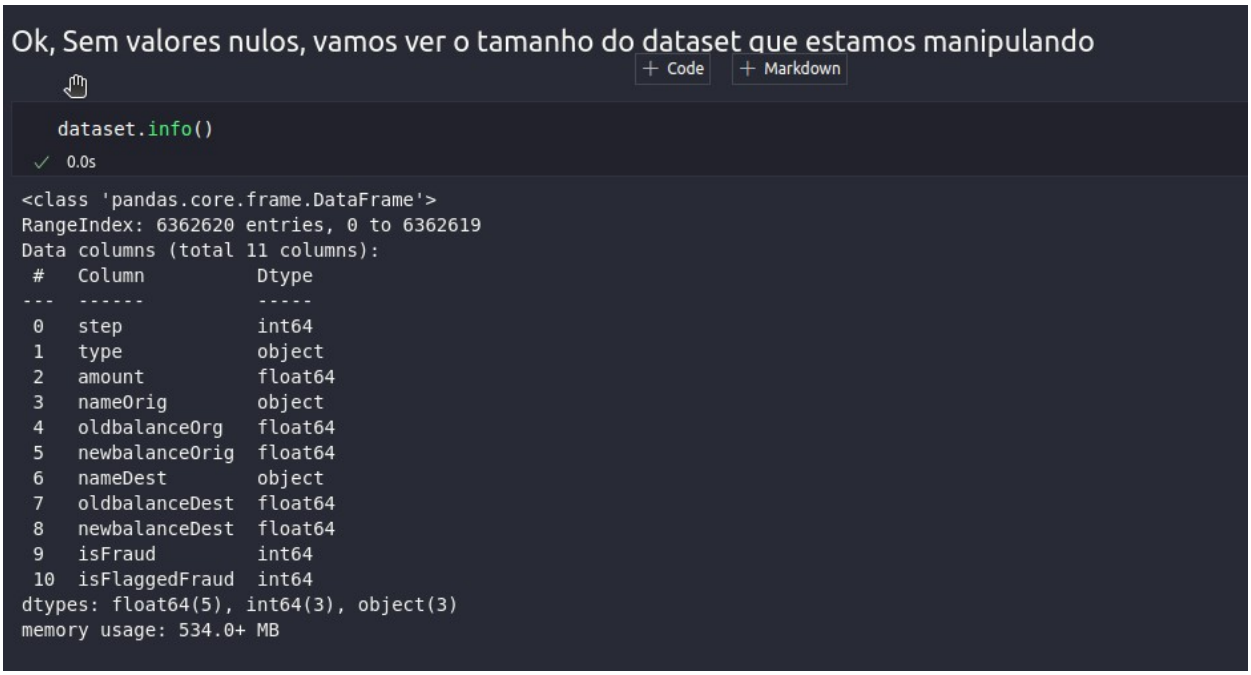
https://www.kaggle.com/datasets/chitwanmanchanda/fraudulent-transactions-data

Se por algum motivo o Kaggle remover o dataset do ar fizemos um backup no google drive que pode ser baixado por qualquer pessoa através do link:

https://drive.google.com/file/d/1Uwtd9_sW51Au0M1RtDBCGHL7Q9Ar14Gu/view?usp=sharing

Este dataset é muito grande, já que estamos falando de IA e Análise de Dados e “Big Data” tentamos pelo menos trazer um V desse big data com um conjunto de dados que não seria possível nem abri-lo no excel, ou seja, teriamos que analisar com técnicas mais avançadas.

As dimesões desse relatório são 6362620 linhas e 11 colunas, após subir o CSV em um dataset do pandas foi possível utilizar a função info para descobrir essas dimensões.



Este dataset possuí 11 colunas sendo elas

step - mapeia uma unidade de tempo no mundo real. Neste caso, 1 passo é 1 hora de tempo. Total de etapas 744 (simulação de 30 dias).

type- CASH-IN, CASH-OUT, DEBIT, PAYMENT e TRANSFER.

amount- valor da transação em moeda local.

nameOrig - cliente que iniciou a transação

oldbalanceOrg - saldo inicial antes da transação

newbalanceOrig - novo saldo após a transação

nameDest - cliente que é o destinatário da transação

oldbalanceDest - destinatário do saldo inicial antes da transação. Observe que não há informações para clientes que começam com M (Comerciantes).

newbalanceDest - novo destinatário do saldo após a transação. Observe que não há informações para clientes que começam com M (Comerciantes).

isFraud - São as transações feitas pelos agentes fraudulentos dentro da simulação. Neste conjunto de dados específico, o comportamento fraudulento dos agentes visa lucrar tomando controle ou contas de clientes e tentando esvaziar os fundos transferindo para outra conta e depois sacando do sistema.

isFlaggedFraud - O modelo de negócios visa controlar transferências massivas de uma conta para outra e sinaliza tentativas ilegais. É uma flag criada pelo próprio modelo.

Como chegar aos resultados?

● Estudar a base para entende-la

Toda a análise exploratória, os gráficos gerados além dos algoritmos utilizados estão disponíveis no github no repositório: https://github.com/jorgelpiva/A3_IAvsBigData

A análise está sendo conduzida em um jupyter notebook analise_exploratoria.ipynb

Primeiramente importamos o arquivo.

Depois verificamos se haviam valores nulos:

Coletamos maiores informações sobre a base de dados.

Ok, Sem valores nulos, vamos ver o tamanho do dataset que estamos manipulando

```
dataset.info()
```

[5] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            object
 2   amount          float64
 3   nameOrig        object
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        object
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64
 10  isFlaggedFraud  int64
dtypes: float64(5), int64(3), object(3)
memory usage: 534.0+ MB
```

E demos uma olhada nos dados

Existem 3 colunas de texto o restante são colunas numéricas, vamos dar uma olhada nestes dados, as primeiras linhas, as ultimas e uma amostra geral

```
dataset.head(15)
```

✓ 0.0s Python

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	0	0
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	0	0
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	1	0
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.00	0.00	1	0
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	0	0
5	1	PAYMENT	7817.71	C90045638	53860.00	46042.29	M573487274	0.00	0.00	0	0
6	1	PAYMENT	7107.77	C154988899	183195.00	176087.23	M408069119	0.00	0.00	0	0
7	1	PAYMENT	7861.64	C1912850431	176087.23	168225.59	M633326333	0.00	0.00	0	0
8	1	PAYMENT	4024.36	C1265012928	2671.00	0.00	M1176932104	0.00	0.00	0	0
9	1	DEBIT	5337.77	C712410124	41720.00	36382.23	C195600860	41898.00	40348.79	0	0
10	1	DEBIT	9644.94	C1900366749	4465.00	0.00	C997608398	10845.00	157982.12	0	0
11	1	PAYMENT	3099.97	C249177573	20771.00	17671.03	M2096539129	0.00	0.00	0	0
12	1	PAYMENT	2560.74	C1648232591	5070.00	2509.26	M972865270	0.00	0.00	0	0
13	1	PAYMENT	11633.76	C1716932897	10127.00	0.00	M801569151	0.00	0.00	0	0
14	1	PAYMENT	4098.78	C1026483832	503264.00	499165.22	M1635378213	0.00	0.00	0	0

Tam bém utilizamos a função describe para entender os valores.

```
dataset.describe()
```

✓ 1.6s Python

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
count	6362620.00	6362620.00	6362620.00	6362620.00	6362620.00	6362620.00	6362620.00	6362620.00
mean	243.40	179861.90	833883.10	855113.67	1100701.67	1224996.40	0.00	0.00
std	142.33	603858.23	2888242.67	2924048.50	3399180.11	3674128.94	0.04	0.00
min	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	156.00	13389.57	0.00	0.00	0.00	0.00	0.00	0.00
50%	239.00	74871.94	14208.00	0.00	132705.66	214661.44	0.00	0.00
75%	335.00	208721.48	107315.18	144258.41	943036.71	1111909.25	0.00	0.00
max	743.00	92445516.64	59585040.37	49585040.37	356015889.35	356179278.92	1.00	1.00

Depois criamos mais 3 colunas no dataset o prefixo do nameDest, o Prefixo do nameOri que seria a primeira letra de cada nome.

```
# Criar a coluna "prefixNameOrig" no dataset
dataset['prefixNameOrig'] = dataset['nameOrig'].apply(lambda x: x[0])

# Criar a coluna "prefixNameDest" no dataset
dataset['prefixNameDest'] = dataset['nameDest'].apply(lambda x: x[0])
```

[11] ✓ 1.9s Python

Criamos uma faixa para os valores das transações.

Já que temos poucas variáveis categóricas iremos criar mais uma, uma faixa de valor das transações onde:

a primeira faixa vai de 0 a 100.000

a segunda faixa vai de 100.000 a 500.000

a terceira faixa vai de 500.000 a 1.000.000

a quarta faixa vai de 1.000.000 a 10.000.000

a quinta faixa são valores maiores que 10.000.000

+ Code + Markdown

```
# Definir as faixas de valores e os rótulos
faixas = [
    (0, 100000, 'Faixa 1'),
    (100000, 500000, 'Faixa 2'),
    (500000 , 1000000, 'Faixa 3'),
    (1000000, 10000000, 'Faixa 4'),
    (10000000, float('inf'), 'Faixa 5')
]

# Função para atribuir a categoria com base no valor
def assign_category(amount):
    for i, faixa in enumerate(faixas):
        if faixa[0] <= amount < faixa[1]:
            return faixa[2]
    return None

# Criar a coluna "amount_category" com as faixas de valores no dataset numérico
dataset['amount_category'] = dataset['amount'].apply(assign_category)
```

[12] ✓ 3.2s Python

Depois demos mais uma olhadinha de como ficou:

Vamos ver como ficou

```
dataset.sample(15)
```

[13] ✓ 1.5s Python

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud	prefixNameOrig	prefixNameDest	amount_category
2350702	189	CASH_IN	90846.32	C1444696899	8760699.79	8851546.11	C1398316663	810517.34	719671.02	0	0	C	C	Faixa 1
4345139	308	TRANSFER	701950.80	C409278596	10986.00	0.00	C203082427	2499751.49	3201702.29	0	0	C	C	Faixa 3
3855445	283	PAYMENT	10493.57	C396691123	0.00	0.00	M948261636	0.00	0.00	0	0	C	M	Faixa 1
2119350	183	CASH_OUT	274863.65	C1479280927	0.00	0.00	C2089629621	858464.63	1133328.28	0	0	C	C	Faixa 2
652460	35	CASH_OUT	141250.31	C385779760	10616.00	0.00	C850426597	160405.73	301656.04	0	0	C	C	Faixa 2
3796382	281	PAYMENT	6417.39	C1909879077	32809.28	26391.89	M1776126386	0.00	0.00	0	0	C	M	Faixa 1
2562812	206	CASH_IN	146514.23	C624998793	159475.14	305989.37	C784176608	2963025.24	2816511.01	0	0	C	C	Faixa 2
2698619	211	CASH_IN	75083.02	C324782483	160.00	75243.02	C2102476101	253037.18	280777.91	0	0	C	C	Faixa 1
809653	40	CASH_OUT	87276.18	C1214955450	0.00	0.00	C531072524	95981.45	183257.63	0	0	C	C	Faixa 1
3203715	249	CASH_IN	191058.19	C955209936	7724984.90	7916043.09	C1985253789	2463637.45	2272579.26	0	0	C	C	Faixa 2
1964281	178	CASH_IN	24282.39	C533353136	5844751.87	5869034.26	C1246952231	1742991.74	1796343.01	0	0	C	C	Faixa 1
1089938	129	CASH_OUT	357373.42	C1345509589	16977.00	0.00	C1369138601	705353.19	1062726.62	0	0	C	C	Faixa 2
1556702	154	CASH_OUT	262130.37	C1818298457	0.00	0.00	C889904914	1084710.66	1346841.03	0	0	C	C	Faixa 2
1000724	45	CASH_OUT	144122.81	C82478056	24253.03	0.00	C1992329654	397688.18	541810.99	0	0	C	C	Faixa 2
676461	36	PAYMENT	7810.96	C892336401	151249.36	143438.39	M23600363	0.00	0.00	0	0	C	M	Faixa 1

Aí fizemos o describe só nas colunas de texto.

```
dataset[['type', 'nameOrig' , 'nameDest', 'prefixNameOrig' , 'prefixNameDest', 'amount_category']].describe()
```

14] ✓ 16.2s

	type	nameOrig	nameDest	prefixNameOrig	prefixNameDest	amount_category
count	6362620	6362620	6362620	6362620	6362620	6362620
unique	5	6353307	2722362	1	2	5
top	CASH_OUT	C1902386530	C1286084959	C	C	Faixa 1
freq	2237500	3	113	6362620	4211125	3525256

E somente nas colunas numéricas também.

```
dataset[['step', 'amount' , 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest', 'isFraud', 'isFlaggedFraud']].describe()
```

✓ 1.6s

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
count	6362620.00	6362620.00	6362620.00	6362620.00	6362620.00	6362620.00	6362620.00	6362620.00
mean	243.40	179861.90	833883.10	855113.67	1100701.67	1224996.40	0.00	0.00
std	142.33	603858.23	2888242.67	2924048.50	3399180.11	3674128.94	0.04	0.00
min	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	156.00	13389.57	0.00	0.00	0.00	0.00	0.00	0.00
50%	239.00	74871.94	14208.00	0.00	132705.66	214661.44	0.00	0.00
75%	335.00	208721.48	107315.18	144258.41	943036.71	1111909.25	0.00	0.00
max	743.00	92445516.64	59585040.37	49585040.37	356015889.35	356179278.92	1.00	1.00

Então descobrimos que existiam dois prefixos de nameDest, C e M

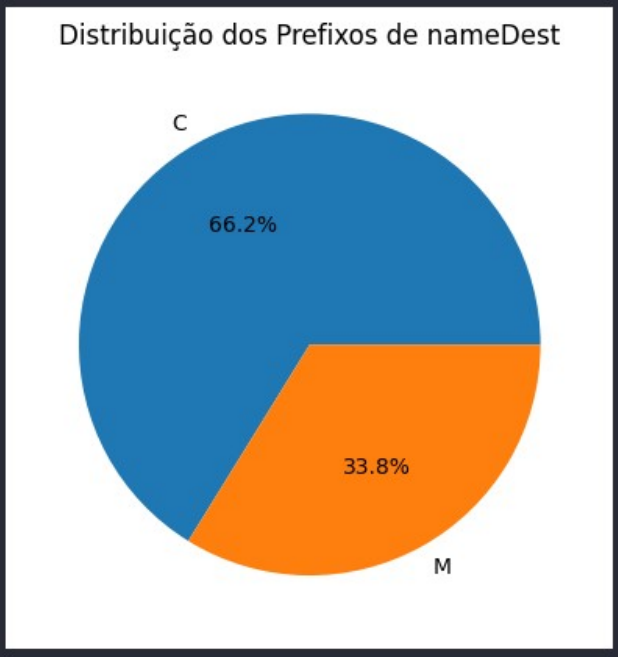
```
dataset['prefixNameDest'].value_counts()
```

✓ 0.2s

```
C    4211125
M    2151495
Name: prefixNameDest, dtype: int64
```

```
#grafico para mostrar a relação do M e do C
counts = dataset['prefixNameDest'].value_counts()
plt.pie(counts, labels=counts.index, autopct='%1.1f%%')
plt.title('Distribuição dos Prefixos de nameDest')
plt.show()
```

✓ 0.3s



Descobrimos que 1/3 dos prefixos era M mas nenhum deles foi fraudado.

Temos quase dois tercos cravados que seria 66.6666%, qual será o percentual dentro dos clientes fraudados?

```
dataset[dataset['isFraud'] == 1]['prefixNameDest'].value_counts()
```

✓ 0.0s Python

```
C    8213
Name: prefixNameDest, dtype: int64
```

Aparentemente clientes com prefixo M não foram fraudados aqui já temos também então a quantidade de Fraudes total aparentemente um percentual baixíssimo, vamos ver este percentual?

```
dataset['isFraud'].value_counts()
```

✓ 0.0s Python

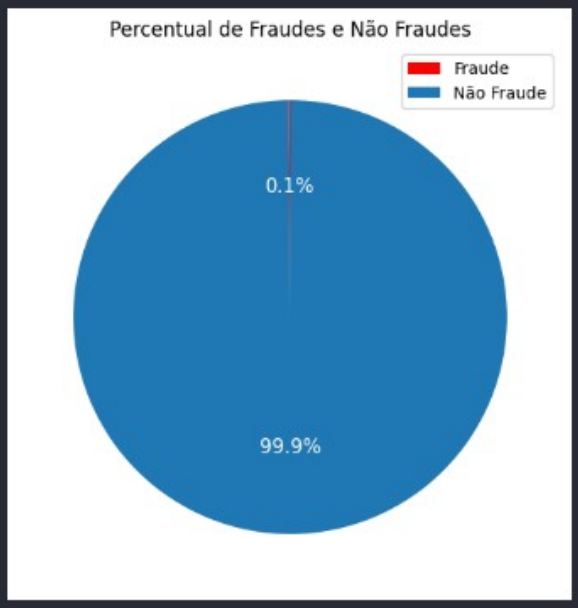
```
0    6354407
1       8213
Name: isFraud, dtype: int64
```

E junto com esta descoberta veio uma informação bombástica, a quantidade de fraudes era muito pequena.

```python
# Contagem das ocorrências de fraudes e não fraudes
fraud_count = dataset['isFraud'].value_counts()

# Criar uma lista com as contagens
count_list = [fraud_count[1], fraud_count[0]]
# Definir os rótulos para o gráfico
labels = ['Fraude', 'Não Fraude']

# Definir as cores das fatias
colors = ['#FF0000', '#1f77b4']  # Vermelho para fraude, azul para não fraude

# Plotar o gráfico de pizza
plt.figure(figsize=(6, 6))  # Definir tamanho da figura
plt.figure(figsize=(6, 6))  # Definir tamanho da figura
patches, _, text = plt.pie(count_list, labels=labels, autopct='%1.1f%%', startangle=90, colors=colors,
                           textprops={'color': 'white', 'fontsize': 12})
# Adicionar título ao gráfico
plt.title('Percentual de Fraudes e Não Fraudes')
plt.legend()
plt.show()
```

✓ 0.2s

```
<Figure size 600x600 with 0 Axes>
```

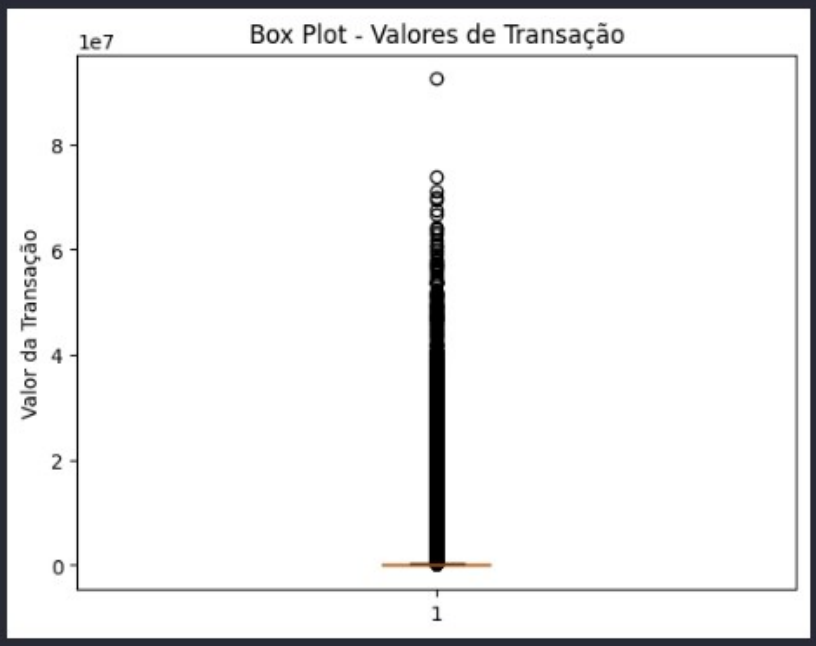


Dentro de uma base de 6 milhões de registros apenas 8 mil eram fraudulentas, o que dificulta muito para os algoritmos de IA e para nós mesmo conduzirmos análises, a partir deste momento, deveríamos trabalhar sempre olhando para o geral e para as operações fraudulentas de forma apartada.

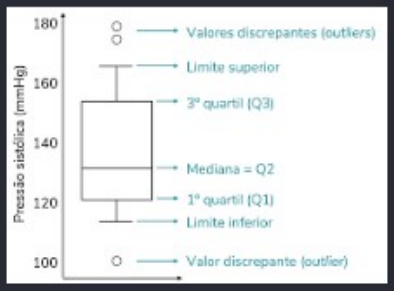
Tentamos identificar outliers através de um box-plot

```
plt.boxplot(dataset['amount'])
plt.ylabel('Valor da Transação')
plt.title('Box Plot - Valores de Transação')
plt.show()
```

[21] ✓ 0.7s



Este Gráfico está muito feio, pra ter uma idéia o normal seria um gráfico assim:

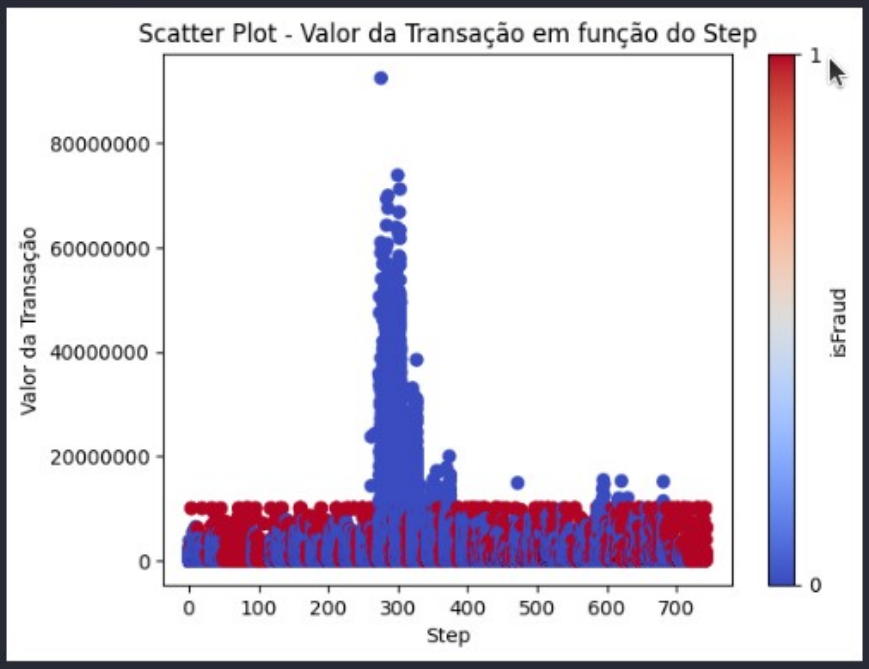


E descobrimos que tinhamos muitos outliers.

Então plotamos um scatterplot e ficou evidente que tínhamos um outlier, um momento de sazonalidade entre as transações ok e havia regularidade entre as transações fraudadas.

```
plt.scatter(dataset['step'], dataset['amount'], c=dataset['isFraud'], cmap='coolwarm')
plt.xlabel('Step')
plt.ylabel('Valor da Transação')
plt.title('Scatter Plot - Valor da Transação em função do Step')

plt.colorbar(label='isFraud', ticks=[0, 1])

# Ajustar o formato dos rótulos do eixo y
plt.gca().set_yticklabels(['{:.0f}'.format(x) for x in plt.gca().get_yticks()])

# Ajustar o formato dos rótulos do eixo x
plt.gca().set_xticklabels(['{:.0f}'.format(x) for x in plt.gca().get_xticks()])

plt.show()
```

[22] ✓ 1m 28.7s



Aparentemente as transações fraudulentas possuem uma regularidade e ela está bem abaixo das transações de maior valor, porém, na regularidade, as trasações fraudulentas possuem uma média de valor maior do que o dataset geral.

Isso fica evidente quando dividimos o dataset entre transações fraudadas e não fraudadas, quando aplicamos a função describe a média fica evidente.

```
[30]  dataset_fraude['amount'].describe()

...   count       8213.00
      mean     1467967.30
      std      2404252.95
      min            0.00
      25%       127091.33
      50%       441423.44
      75%      1517771.48
      max     10000000.00
      Name: amount, dtype: float64

[31]  dataset_no_fraude['amount'].describe()

...   count    6354407.00
      mean      178197.04
      std       596236.98
      min            0.01
      25%        13368.40
      50%        74684.72
      75%       208364.76
      max     92445516.64
      Name: amount, dtype: float64
```

Então tendo a noção da complexidade do que acontece, começamos a aplicar os algoritmos de machine learning em nossa variável target que foi escolhida o **isFraud**.

Criamos um novo dataset para gravar os dados como tempo de excecução e os resultados.

Fizemos alguns pré processamentos, primeiramente com a transformação do dataset geral para um dataset apenas com tipos numéricos com a biblioteca label encoder do sklearn

Já vimos que este dataset é complicado, tem um percentual muito baixo de fraudes poucas variáveis categóricas, as fraudes ocorreram em todos os momentos da série temporal com valores bem distribuídos que não chamam a atenção inclusive bem abaixo dos outliers e dos momentos sazonais, vamos tentar criar uma forma mais fácil de prever essas fraudes com algumas técnicas de machine lerning

Agora que não iremos adicionar mais nenhuma coluna no dataset, deixamos para o final fazer o pré-processamento de dados, uma vez que os datasets para machine learning precisam estar em variáveis numéricas

```python
# Criar cópia do dataset original
dataset_ml = dataset.copy()

# Criar instância do LabelEncoder
label_encoder = LabelEncoder()

# Codificar as variáveis categóricas e labels
dataset_ml['type_encoded'] = label_encoder.fit_transform(dataset_ml['type'])
dataset_ml['nameOrig_encoded'] = label_encoder.fit_transform(dataset_ml['nameOrig'])
dataset_ml['nameDest_encoded'] = label_encoder.fit_transform(dataset_ml['nameDest'])
dataset_ml['prefixNameOrig_encoded'] = label_encoder.fit_transform(dataset_ml['prefixNameOrig'])
dataset_ml['prefixNameDest_encoded'] = label_encoder.fit_transform(dataset_ml['prefixNameDest'])
dataset_ml['amount_category_encoded'] = label_encoder.fit_transform(dataset_ml['amount_category'])

# Remover colunas não utilizadas
columns_to_drop = ['type', 'nameOrig', 'nameDest', 'prefixNameOrig', 'prefixNameDest', 'amount_category']
dataset_ml = dataset_ml.drop(columns=columns_to_drop)
```

Python

Dividimos o dataset em 70% para treino e 30% para teste com a biblioteca train_test_split

Empty markdown cell, double-click or press enter to edit.

```python
# Separar os conjuntos de treino e teste
X = dataset_ml.drop('isFraud', axis=1)
y = dataset_ml['isFraud']
X_treino, X_teste, y_treino, y_teste = train_test_split(X, y, test_size=0.3, random_state=42)
```

Python

Então processamos os resultados com a árvore de decisão:

```python
dtInicio = datetime.datetime.now()
# Criar uma instância do classificador de árvore de decisão
arvoreDeDecisao = DecisionTreeClassifier()

# Treinar o classificador utilizando os dados de treinamento
arvoreDeDecisao.fit(X_treino, y_treino)

# Fazer previsões utilizando os dados de teste
previsoes = arvoreDeDecisao.predict(X_teste)

# Calcular a acurácia do modelo
acuracia_arvore = accuracy_score(y_teste, previsoes)
acuracia_arvore_no_fraude = accuracy_score(y_teste[y_teste == 0], previsoes[y_teste == 0])
acuracia_arvore_fraude = accuracy_score(y_teste[y_teste == 1], previsoes[y_teste == 1])
print("Acurácia do modelo de Arvore de decisão foi:", acuracia_arvore)
print("Acurácia do modelo de Arvore de decisão para operações fraudulentas foi:", acuracia_arvore_fraude)
print("Acurácia do modelo de Arvore de decisão para operações não fraudulentas foi:", acuracia_arvore_no_fraude)
dtFin = datetime.datetime.now()
tempoExecucao = dtFin - dtInicio

df_resultados = df_resultados.append({
    'Algoritmo': 'Árvore de Decisão',
    'Acuracia_Geral': acuracia_arvore,
    'Acuracia_Fraude': acuracia_arvore_fraude,
    'Acuracia_No_Fraude': acuracia_arvore_no_fraude,
    'Tempo_Execucao': tempoExecucao
}, ignore_index=True)
```

Python

```
Acurácia do modelo de Arvore de decisão foi: 0.999676234004231
Acurácia do modelo de Arvore de decisão para operações fraudulentas foi: 0.864476386036961
Acurácia do modelo de Arvore de decisão para operações não fraudulentas foi: 0.9998489260372303
```

Como comentado anteriormente medimos o resultado geral, de forma a olhar os acertos apenas nas fraudes e olhando também os acertos dos não fraudados, para nossa supresa a árvore de decisão entregou um resultado

geral de 99,99% e de fraudados de 86% um número fomidável para o nível de complexidade do dataset, porém, fizemos os outros modelos.

Cabe ressaltar que como temos 99,99% dos registros como transações normais, caso um algoritmo colocasse todas as ocorrências como isFraud = 0 o algoritmo acertaria 99,99% do geral e 0% dos fraudados por isso temos que dividir os acertos para mensurar a eficácia.

Antes de prosseguir, fizemos um pré processamento para padronização dos valores, do contrário, alguns algoritmos como o KNN iriam dar erro, esse pré processamento foi feito com a biblioteca StandardScaler também do sckitlearn

```
# Pré-processamento dos dados
scaler = StandardScaler()
X_treino_scaled = scaler.fit_transform(X_treino)
X_teste_scaled = scaler.transform(X_teste)
```

Então fizemos a regressão logística.

```
dtInicio = datetime.datetime.now()
Criação do modelo de Regressão Logística
logreg = LogisticRegression()

# Treinamento do modelo
logreg.fit(X_treino, y_treino)

# Previsões no conjunto de teste
y_pred_logreg = logreg.predict(X_teste)

# Cálculo da acurácia geral
acuracia_Logistica_geral = accuracy_score(y_teste, y_pred_logreg)

# Filtrar os dados para calcular a acurácia para a classe de fraude
acuracia_Logistica_Fraude = accuracy_score(y_teste[y_teste == 1], y_pred_logreg[y_teste == 1])

# Filtrar os dados para calcular a acurácia para a classe sem fraude
acuracia_Logistica_No_Fraude = accuracy_score(y_teste[y_teste == 0], y_pred_logreg[y_teste == 0])

print("Acurácia do modelo de Regressão Logística geral foi:", acuracia_Logistica_geral)
print("Acurácia do modelo de Regressão Logística em operações fraudulentas foi:", acuracia_Logistica_Fraude)
print("Acurácia do modelo de Regressão Logística para operações não fraudulentas foi:", acuracia_Logistica_No_Fraude)
dtFin = datetime.datetime.now()
tempoExecucao = dtFin - dtInicio

df_resultados = df_resultados.append({
    'Algoritmo': 'Regressão Logística',
    'Acuracia_Geral': acuracia_Logistica_geral,
    'Acuracia_Fraude': acuracia_Logistica_Fraude,
    'Acuracia_No_Fraude': acuracia_Logistica_No_Fraude,
    'Tempo_Execucao': tempoExecucao
}, ignore_index=True)
```

[se] Python

```
Acurácia do modelo de Regressão Logística geral foi: 0.9991329567589033
Acurácia do modelo de Regressão Logística em operações fraudulentas foi: 0.43983572895277206
Acurácia do modelo de Regressão Logística para operações não fraudulentas foi: 0.9998473523501181
```

Tivemos um resultado interessante, cerca de 44% de acerto para as fraudes mas bem abaixo da árvore de decisão.

Então partimos para a rede neural MLP, e aí tivemos uma grande decepção:

```python
dtInicio = datetime.datetime.now()
# Criar uma instância do modelo MLPClassifier
mlp = MLPClassifier()

# Treinar o modelo usando os dados de treino
mlp.fit(X_treino, y_treino)

# Fazer previsões para os dados de teste
y_pred = mlp.predict(X_teste)

# Calcular a acurácia geral
acuracia_MLP_geral = accuracy_score(y_teste, y_pred)

# Filtrar os dados para calcular a acurácia para a classe de fraude
acuracia_MLP_Fraude = accuracy_score(y_teste[y_teste == 1], y_pred[y_teste == 1])

# Filtrar os dados para calcular a acurácia para a classe sem fraude
acuracia_MLP_No_Fraude = accuracy_score(y_teste[y_teste == 0], y_pred[y_teste == 0])

print("Acurácia do modelo de Rede Neural MLP geral foi:", acuracia_MLP_geral)
print("Acurácia do modelo de Rede Neural MLP em operações fraudulentas foi:", acuracia_MLP_Fraude)
print("Acurácia do modelo de Rede Neural MLP para operações não fraudulentas foi:", acuracia_MLP_No_Fraude)

dtFin = datetime.datetime.now()
tempoExecucao = dtFin - dtInicio

df_resultados = df_resultados.append({
    'Algoritmo': 'Rede Neural MLP Class',
    'Acuracia_Geral': acuracia_MLP_geral,
    'Acuracia_Fraude': acuracia_MLP_Fraude,
    'Acuracia_No_Fraude': acuracia_MLP_No_Fraude,
    'Tempo_Execucao': tempoExecucao
}, ignore_index=True)
```

Python

```
Acurácia do modelo de Rede Neural MLP geral foi: 0.9987693748801594
Acurácia do modelo de Rede Neural MLP em operações fraudulentas foi: 0.03696098562628337
Acurácia do modelo de Rede Neural MLP para operações não fraudulentas foi: 0.9999979017505171
```

Foi um resultado de menos de 4% inconformados, mudamos alguns parâmetros e rodamos de novo.

Esta acurácia da rede neural MLP ficou muito baixa, quando isso acontece o ideal é fazer uma experimentação mudando alguns parametros, neste caso mudaremos todos e adicionaremos profundidade com 3 camadas de neurônios.

```python
dtInicio = datetime.datetime.now()
# Criar uma instância do modelo MLPClassifier
mlpNdef = MLPClassifier(hidden_layer_sizes=(64, 64, 64), random_state=42, activation='identity', solver='lbfgs')

# Treinar o modelo usando os dados de treino
mlpNdef.fit(X_treino, y_treino)

# Fazer previsões para os dados de teste
y_pred = mlpNdef.predict(X_teste)

# Calcular a acurácia geral
acuracia_MLPNdef_geral = accuracy_score(y_teste, y_pred)

# Filtrar os dados para calcular a acurácia para a classe de fraude
acuracia_MLPNdef_Fraude = accuracy_score(y_teste[y_teste == 1], y_pred[y_teste == 1])

# Filtrar os dados para calcular a acurácia para a classe sem fraude
acuracia_MLPNdef_No_Fraude = accuracy_score(y_teste[y_teste == 0], y_pred[y_teste == 0])

print("Acurácia do modelo de Rede Neural MLP geral foi:", acuracia_MLPNdef_geral)
print("Acurácia do modelo de Rede Neural MLP em operações fraudulentas foi:", acuracia_MLPNdef_Fraude)
print("Acurácia do modelo de Rede Neural MLP para operações não fraudulentas foi:", acuracia_MLPNdef_No_Fraude)

dtFin = datetime.datetime.now()
tempoExecucao = dtFin - dtInicio

df_resultados = df_resultados.append({
    'Algoritmo': 'Rede Neural MLP Class Alterado',
    'Acuracia_Geral': acuracia_MLPNdef_geral,
    'Acuracia_Fraude': acuracia_MLPNdef_Fraude,
    'Acuracia_No_Fraude': acuracia_MLPNdef_No_Fraude,
    'Tempo_Execucao': tempoExecucao
}, ignore_index=True)
```

[58] Python

```
Acurácia do modelo de Rede Neural MLP geral foi: 0.951825924959634
Acurácia do modelo de Rede Neural MLP em operações fraudulentas foi: 0.002053388090349076
Acurácia do modelo de Rede Neural MLP para operações não fraudulentas foi: 0.9530390783229321
```

Então percebemos que precisamos estudar mais os parametros para conseguir uma melhor avaliação, como os tempos de execução estavam

crescendo à medida que estávamos aumentando o número de camadas da rede neural estávamos caminhando em direção ao *DeepLearning*, decidimos passar para o próximo algoritmo o SVC .

E o SVC se saiu muito melhor que a rede neural MLP mas abaixo da regressão logística e muito abaixo da Árvore de Decisão com apenas 33% de acertos nas operações fraudulentas.

Ficamos convictos que teríamos um melhor aproveitamento com o naive bayes então seguimos para o mesmo.

```
dtInicio = datetime.datetime.now()
# Criar uma instância do modelo GaussianNB
nb = GaussianNB()

# Treinar o modelo usando os dados de treino
nb.fit(X_treino, y_treino)

# Fazer previsões para os dados de teste
y_pred = nb.predict(X_teste)

# Calcular a acurácia geral
acuracia_NB_geral = accuracy_score(y_teste, y_pred)

# Filtrar os dados para calcular a acurácia para a classe de fraude
acuracia_NB_Fraude = accuracy_score(y_teste[y_teste == 1], y_pred[y_teste == 1])

# Filtrar os dados para calcular a acurácia para a classe sem fraude
acuracia_NB_No_Fraude = accuracy_score(y_teste[y_teste == 0], y_pred[y_teste == 0])

print("Acurácia do modelo de Algoritmo Naive Bayes geral foi:", acuracia_NB_geral)
print("Acurácia do modelo de Algoritmo Naive Bayes em operações fraudulentas foi:", acuracia_NB_Fraude)
print("Acurácia do modelo de Algoritmo Naive Bayes para operações não fraudulentas foi:", acuracia_NB_No_Fraude)

dtFin = datetime.datetime.now()
tempoExecucao = dtFin - dtInicio

df_resultados = df_resultados.append({
    'Algoritmo': 'Naive Bayes',
    'Acuracia_Geral': acuracia_NB_geral,
    'Acuracia_Fraude': acuracia_NB_Fraude,
    'Acuracia_No_Fraude': acuracia_NB_No_Fraude,
    'Tempo_Execucao': tempoExecucao
}, ignore_index=True)
```

Python

```
Acurácia do modelo de Algoritmo Naive Bayes geral foi: 0.991427011723682
Acurácia do modelo de Algoritmo Naive Bayes em operações fraudulentas foi: 0.17330595482546202
Acurácia do modelo de Algoritmo Naive Bayes para operações não fraudulentas foi: 0.9924720054176802
```

E novamente tivemos uma leve decepção, quando o Naive Bayes acertou apenas 17% das fraudes nossa útlima esperança para elevar a taxa de acerto era o KNN.

Rodamos o KNN e ele bateu todos os anteriores, menos a Árvore de Decisão chegando a quase 53% de acertos.

```python
dtInicio = datetime.datetime.now()
# Criar uma instância do modelo KNN
knn = KNeighborsClassifier()

# Treinar o modelo usando os dados de treino
knn.fit(X_treino, y_treino)

# Fazer previsões para os dados de teste
y_pred = knn.predict(X_teste)

# Calcular a acurácia geral
acuracia_KNN_geral = accuracy_score(y_teste, y_pred)

# Filtrar os dados para calcular a acurácia para a classe de fraude
acuracia_KNN_Fraude = accuracy_score(y_teste[y_teste == 1], y_pred[y_teste == 1])

# Filtrar os dados para calcular a acurácia para a classe sem fraude
acuracia_KNN_No_Fraude = accuracy_score(y_teste[y_teste == 0], y_pred[y_teste == 0])

print("Acurácia do modelo de Algoritmo KNN geral foi:", acuracia_KNN_geral)
print("Acurácia do modelo de Algoritmo KNN em operações fraudulentas foi:", acuracia_KNN_Fraude)
print("Acurácia do modelo de Algoritmo KNN para operações não fraudulentas foi:", acuracia_KNN_No_Fraude)

dtFin = datetime.datetime.now()
tempoExecucao = dtFin - dtInicio

df_resultados = df_resultados.append({
    'Algoritmo': 'K Nearest Neighbors',
    'Acuracia_Geral': acuracia_KNN_geral,
    'Acuracia_Fraude': acuracia_KNN_Fraude,
    'Acuracia_No_Fraude': acuracia_KNN_No_Fraude,
    'Tempo_Execucao': tempoExecucao
}, ignore_index=True)
```

Python

```
Acurácia do modelo de Algoritmo KNN geral foi: 0.9993037459411375
Acurácia do modelo de Algoritmo KNN em operações fraudulentas foi: 0.526488706365503
Acurácia do modelo de Algoritmo KNN para operações não fraudulentas foi: 0.9999076770227519
```

Confrome mencionado, criamos um dataset para computar os valores do experimento e no final da execução ele ficou da seguinte forma:

```python
df_resultados.head(10)
```

	Algoritmo	Acuracia_Geral	Acuracia_Fraude	Acuracia_No_Fraude	Tempo_Execucao
0	Árvore de Decisão	1.00	0.86	1.00	0 days 00:01:21.276526
1	Regressão Logística	1.00	0.44	1.00	0 days 00:00:38.399384
2	Rede Neural MLP Class	1.00	0.04	1.00	0 days 00:20:33.739344
3	Rede Neural MLP Class Alterado	0.95	0.00	0.95	0 days 00:10:44.489893
4	Support Vector Classifier	1.00	0.33	1.00	0 days 01:15:57.424972
5	Naive Bayes	0.99	0.17	0.99	0 days 00:00:02.423080
6	K Nearest Neighbors	1.00	0.53	1.00	0 days 00:04:37.504406

Além da primeira execução ter sido a mais assertiva ela também foi uma das execuções mais rápidas.

Conclusão

Neste caso então podemos dizer que a primeira opção era a opção mais viável, mas como saber sem testar? Tivemos que fazer diversas baterias de teste para saber que a primeira opção era a mais viável, e se tivéssemos começado pelo KNN teríamos um valor bom, mas que poderia ser melhorado. Se tivéssemos rodado a árvore de decissão por último, só então teríamos encontrado o melhor aproveitamento, e ainda rodamos o MLP duas vezes, essa é rotina de um Cientista de Dados, de um Engenheiro de IA, experimentação e teste, vale lembrar que a ciência vive de testes, pesquisas e experimentos e vale muito a pena fazer todo este caminho mesmo que seja para descobrir no final que o primeiro era o mais assertivo, sabendo que você utilizou de todo o seu conhecimento e recursos disponíveis a fim de entregar o melhor resultado, todos os testes e resultados obtidos são execução de algoritmos complexos que demandam grande poder computacional em um dataset de mais de 6 milhões de linhas, este projeto pode ser aprimorado futuramente com o estudo mais especializado dos parâmetros presentes na biblioteca do sklearn e também com o uso de novas bibliotecas como o tensor flow e o pytorch, porém, é muito satisfatório saber que treinamos 7 modelos de aprendizado de máquina e conseguimos uma acurácia de 86% em identificar fraudes, com uma especialização maior podemos trazer dados ainda mais assertivos, mas para este estudo preliminar ficamos muito satisfeitos com os resultados e interessados em testar os modelos em outros datasets talvez com mais variáveis categóricas, talvez com uma variável target com um percentual maior de positivos, nestas situações, pode ser que a rede neural MLP saia melhor que a Árvore de Decisão ou o Modelo SVC? É possível, mas só será possível descobrir se houver, experimentação, teste e metodologia científica e vontade de tirar informações úteis dos dados.