Abaixo estarei mostrando como foi realizado o estudo de reconhecimento de fraude do dataset Credit Card Fraud Detection que está presente no Kaggle pelo link abaixo e também explicarei o código que foi feito.

Kaggle Dataset: https://www.kaggle.com/mlg-ulb/creditcardfraud

1- Importando os arquivos necessários

Utilizei as seguintes bibliotecas do sklearn e as padrões para realizar o estudo:

```
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import recall_score, accuracy_score, roc_curve, auc, precision_score, confusion_matrix, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
```

2- Importação do arquivo

O arquivo está sendo importado, e caso não consiga ele entrará nas exceptions de arquivo não encontrado ou então na exceção geral.

```
try:
    arquivo_completo = pd.read_csv("creditcard.csv", sep=",")
except FileNotFoundError:
    print("ARQUIVO NAO FOI ENCONTRADO")
except Exception as e:
    print(f"ERRO AO SUBIR ARQUIVO: {e}")
```

3- Limpeza e tratamento dos dados

Limpeza geral como substituir pontuação, remoção de acentos e pontos, e padronizar todo o dataset para Uppercase. Padrão que utilizo normalmente em meus datasets.

```
arquivo_completo = arquivo_completo.replace(".", "").replace(",", ".")
```

```
arquivo_completo.columns = arquivo_completo.columns.str.upper()


cols = arquivo_completo.select_dtypes(include=["object"]).columns
arquivo_completo[cols] = (arquivo_completo[cols].
                          apply(lambda x1: x1.str.normalize("NFKD").str.encode('ascii',
errors='ignore').
                                str.decode('utf-8')))
```

4- Análises para poder tomada de decisão

Realizei uma contagem para verificar quantos casos de fraude estão no arquivo, e pude conferir que apenas 0.17% da base é composta de casos de fraude, como está bem desbalanceada o passo seguinte que realizei foi de balanceamento e separação da base para estudo.

```
arquivo_completo["CLASS"].value_counts()

"0  ==  284315  ||   1   ==    492"
"APENAS 0.17% DA BASE É FRAUDE"
```

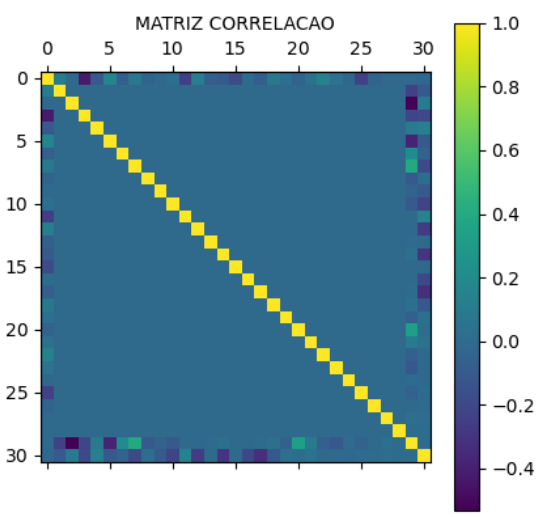
5- Balanceamento da base

Retirei então uma parte do arquivo que continha fraude e uma parte sem para então poder montar um novo DataFrame balanceado.

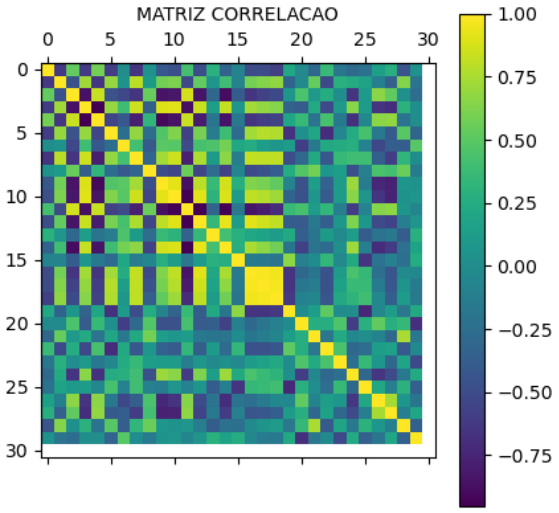
```
arquivo_fraudes = arquivo_completo.loc[np.where(arquivo_completo["CLASS"] == 1)][:50]

arquivo_sem_fraudes = arquivo_completo.loc[np.where(arquivo_completo["CLASS"] ==
0)][:50]
```

6- Visualização entre bases

Pelo Matplotlib podemos conferir a diferença de correlação de variáveis entre uma base que está balanceada e outra que não está:



Base Desbalanceada



Base Balanceada

Assim podemos ver realmente quais as colunas que estão com maior correlação e escolher as melhores para o modelo.

7- Mantendo o arquivo original sem os dados de treino

Para que não tenha overfit nos resultados removi da base original os dados que vamos utilizar para montar o estudo.

```
arquivo_resto_fraudes = arquivo_completo.loc[np.where(arquivo_completo["CLASS"] == 1)][50:]

arquivo_resto_sem_fraudes = arquivo_completo.loc[np.where(arquivo_completo["CLASS"] == 0)][50:]

arquivo = pd.concat([arquivo_resto_fraudes, arquivo_resto_sem_fraudes])
```

8- Tratando valores Nan

Substituímos valores Nan em variáveis categóricas com sua frequência em que aparecem e para as variáveis numéricas substituímos com a mediana do resultado de cada coluna para evitar uma média muito distorcida do resultado.

```
num_att = arquivo_fraudes.select_dtypes(exclude=["object", "datetime"]).columns.to_list()

imputer_mediana = SimpleImputer(strategy='median')

scaler = MinMaxScaler()
```

```
for num in num_att:
    arquivo_fraudes[num] =
imputer_mediana.fit_transform(np.array(arquivo_fraudes[num]).reshape(-1, 1))
    arquivo_fraudes[num] = scaler.fit_transform(np.array(arquivo_fraudes[num]).reshape(-1,
1))

arquivo_fraudes.dropna()
```

9- Verificando as melhores colunas para serem utilizadas no modelo

Com a ajuda do gráfico que fizemos no Matplot e com a função de KBest podemos verificar as melhores colunas e sua pontuação para assim inserirmos no modelo.
Caso o modelo tenha algum resultado abaixo do esperado podemos então ir alterando as colunas até chegar num resultado satisfatório.

```
X = arquivo_fraudes
y = arquivo_fraudes["CLASS"]

bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X, y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)

featureScores = pd.concat([dfcolumns, dfscores], axis=1)
featureScores.columns = ['Specs', 'Score']
print(featureScores.nlargest(20, 'Score'))

"""TOP 10 COLUNAS COM MAIOR PONTUAÇÃO DO DF"""
"""    Specs       Score
30  CLASS  492.000000
4      V4   53.974577
11    V11   44.133827
14    V14   31.839693
12    V12   28.666539
16    V16   20.267755
17    V17   15.792533
3      V3   12.678611
10    V10   11.473454
18    V18   11.305559"""
```

10- Escolha do melhor modelo

Criei uma função que mostra o resultado de cada algorítmo que eu colocar e assim pode me mostrar qual o melhor modelo que eu possa usar, com os dados de treino então o resultado pode ser conferido logo abaixo.

```
X = arquivo_fraudes[["V4", "V11", "V16", "V3", "V10", "V18", "V9"]]
y = arquivo_fraudes[["CLASS"]]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1, shuffle=True)


lista_modelos = [RandomForestClassifier(random_state=1),
LogisticRegression(random_state=1),
                 DecisionTreeClassifier(), GaussianNB()]

for tipo_modelo in lista_modelos:
    retorna_resultado_modelo(tipo_modelo)


def retorna_resultado_modelo(tipo_modelo_parametro):
    # REALIZANDO O FIT E PREDICT DO MODELO

    y_pred_parametro = tipo_modelo_parametro.fit(X_train, y_train).predict(X_test)

    # REALIZANDO TESTES DE DESEMPENHO DE MODELO

    """F1 SCORE"""
    f1 = f1_score(y_test, y_pred_parametro, average='macro')

    """RECALL"""
    recall = recall_score(y_test, y_pred_parametro, average='macro')

    """ACURÁCIA"""
    acuracia = accuracy_score(y_test, y_pred_parametro)

    """PRECISÃO"""
    precisao = precision_score(y_test, y_pred_parametro, average='macro')

    lista_testes[tipo_modelo] = [f1, recall, acuracia, precisao]

    return print(f"MODELO {tipo_modelo_parametro}\nF1: {f1}\nRECALL:
{recall}\nACURACIA: {acuracia}\n"
                 f"PRECISAO: {precisao}\n")
```

O resultado de cada modelo pode ser observado abaixo:

MODELO RandomForestClassifier(random_state=1)
F1: 0.9188311688311688
RECALL: 0.9333333333333333

ACURACIA: 0.92
PRECISAO: 0.9166666666666667

MODELO LogisticRegression(random_state=1)
F1: 0.8792270531400967
RECALL: 0.9
ACURACIA: 0.88
PRECISAO: 0.8846153846153846

MODELO DecisionTreeClassifier()
F1: 0.8792270531400967
RECALL: 0.9
ACURACIA: 0.88
PRECISAO: 0.8846153846153846

MODELO GaussianNB()
F1: 0.9589490968801313
RECALL: 0.9666666666666667
ACURACIA: 0.96
PRECISAO: 0.9545454545454546

Pelos resultados pude concluir então que o melhor modelo a ser utilizado na base real seria o algoritmo de **Naive Bayes Gaussiano**, com altos valores de acurácia, recall e precisão também.

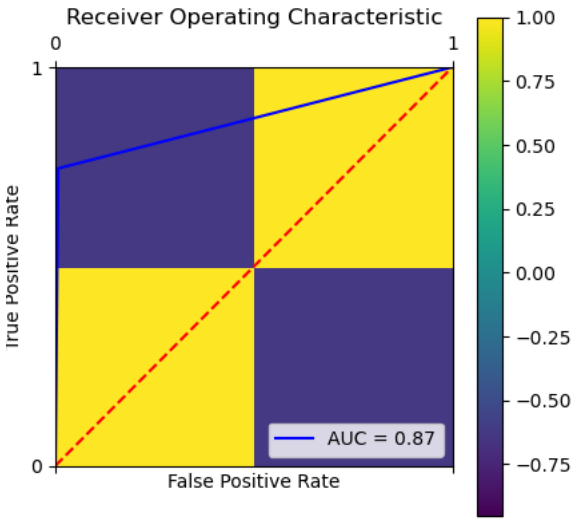
11- Treinamento do modelo

Como verificamos no treinamento do modelo que as melhores colunas foram as V4", "V11", "V16", "V3", "V10", "V18", "V9, já pulamos para a parte de teste da base geral, no treino pudemos averiguar as melhores colunas e o melhor modelo também.

```
X = arquivo[["c"]]
y = arquivo[["CLASS"]]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1, shuffle=True)


modelo = GaussianNB().fit(X_train, y_train)
y_pred = modelo.predict(X_test)

recall_score(y_test, y_pred, average='macro')
accuracy_score(y_test, y_pred)

fpr, tpr, threshold = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label='AUC = %0.2f' % roc_auc)
```

```
plt.legend(loc='lower right')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

confusion_matrix(y_test, y_pred)
```

12- Verificando os resultados finais



Com o resultado da curva ROC e o valor de nossa AUC podemos ver que o modelo realizou bem seu trabalho.

Pela matriz de confusão podemos analisar que:

[70554, 497],
[32, 94]

-> Do total da base tivemos 497 casos em que ocorreram fraude porém o modelo aptou que não eram

-> E apenas 32 casos que não eram fraudes ele marcou que eram

13- Conclusão

Com o estudo pude verificar que o modelo trabalhou bem, e mudando um pouco as colunas e o tipo de modelo possa ter um resultado mais satisfatório, o cliente poderá decidir então qual tipo de modelo ele deseja utilizar de acordo com o objetivo dele.