Data Science Project

Credit Card Fraud Detection

FIS01082 - Tópicos Especiais em Engenharia Física

Guilherme Martins Soares

Natália Capra Ferrazzo

Rafael Correa de Lima

Data Analysis and Exploration

O dataset contém transações feitas com cartões de crédito em Setembro de 2013 por cidadões europeus. O dataset apresenta transações que ocorreram em 2 dias, onde ocorreu 492 fraudes do total de 284.807 transações. Este dataset é altamente desbalanceado: a classe positiva (fraudes) representa 0,172% do total de transações.

Há apenas variáveis numéricas resultantes de transformações PCA (Análise de Componentes Principais). Devido a questões de confidencialidade, não foi fornecido as métricas originais ou maiores informações sobre os dados.

- As métricas V1, V2, ... V28 são os componentes principais obtidos com o PCA.
- As unicas métricas que não sofreram transformação PCA é *Time* e *Amount*. A métrica *Time* contém os segundos decorridos entre cada transação e a primeira transação do dataset. A métrica *Amount* é o valor monetário da transação.
- A métrica *Class* é a variável resposta que assume o valor 1 caso seja uma transação fraudulenta, e 0 caso contrário.

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly import tools
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import gc
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from catboost import CatBoostClassifier
from sklearn import svm

pd.set_option('display.max_columns', 100)
```

```
RFC_METRIC = 'gini'  #metric used for RandomForrestClassifier
NUM_ESTIMATORS = 100 #number of estimators used for RandomForrestClassifier
NO_JOBS = 4 #number of parallel jobs used for RandomForrestClassifier


#TRAIN/VALIDATION/TEST SPLIT
#VALIDATION
VALID_SIZE = 0.20 # simple validation using train_test_split
TEST_SIZE = 0.20 # test size using_train_test_split

#CROSS-VALIDATION
NUMBER_KFOLDS = 5 #number of KFolds for cross-validation

RANDOM_STATE = 2018

MAX_ROUNDS = 1000 #lgb iterations
EARLY_STOP = 50 #lgb early stop
OPT_ROUNDS = 1000  #To be adjusted based on best validation rounds
VERBOSE_EVAL = 50 #Print out metric result

IS_LOCAL = False

import os
```

In [4]:
```
data_df = pd.read_csv("creditcard.csv")
```

In [5]:
```
print("Credit Card Fraud Detection data -  rows:",data_df.shape[0]," columns:
```

```
Credit Card Fraud Detection data -  rows: 284807  columns: 31
```

In [6]:
```
data_df.head()
```

Out[6]:

	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	0.0986
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0851
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2476
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3774
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2705

In [7]:
```
data_df.describe()
```

Out[7]:

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.43358

	Time	V1	V2	V3	V4	
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e

In [8]:

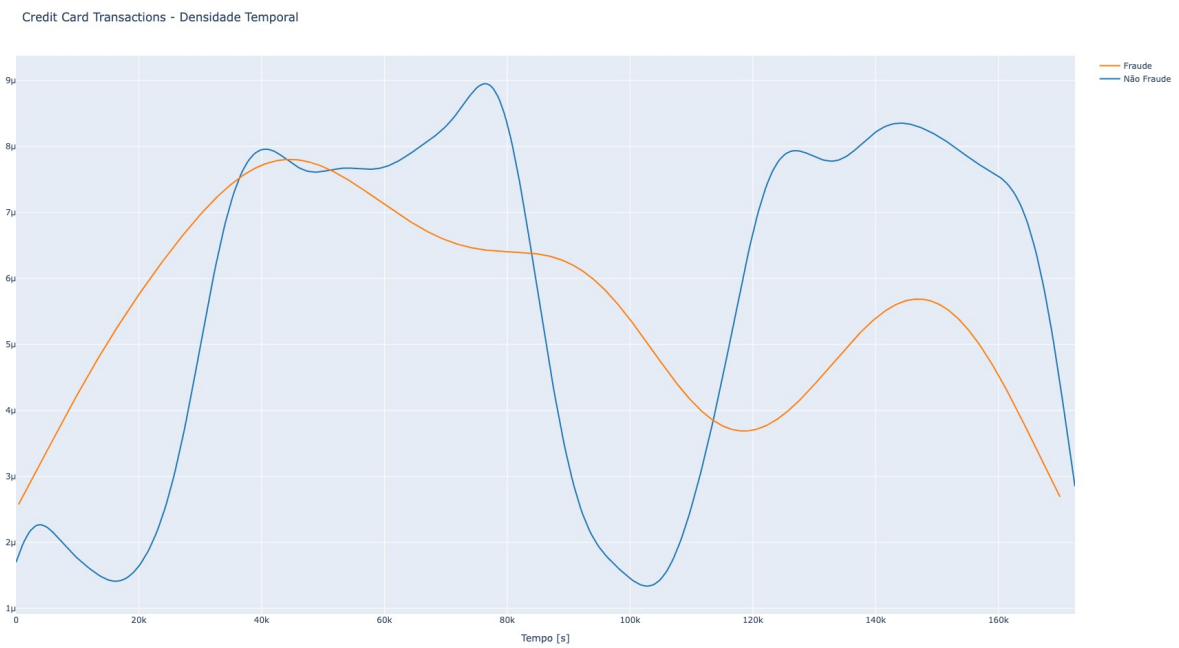
```
total = data_df.isnull().sum().sort_values(ascending = False)
percent = (data_df.isnull().sum()/data_df.isnull().count()*100).sort_values(a
pd.concat([total, percent], axis=1, keys=['Total', 'Percent']).transpose()
```

Out[8]:

	Time	V16	Amount	V28	V27	V26	V25	V24	V23	V22	V21	V20	V19	V18	V
Total	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
Percent	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

In [13]:

```
temp = data_df["Class"].value_counts()
df = pd.DataFrame({'Class': temp.index,'values': temp.values})

trace = go.Bar(
    x = df['Class'],y = df['values'],
    name="Desequilíbrio dos Dados (Não fraude = 0, Fraude = 1)",
    marker=dict(color="Red"),
    text=df['values']
)
data = [trace]
layout = dict(title = 'Desequilíbrio dos Dados (Não fraude = 0, Fraude = 1)',
          xaxis = dict(title = 'Classe', showticklabels=True),
          yaxis = dict(title = 'Número de transações'),
          hovermode = 'closest',width=600
         )
fig = dict(data=data, layout=layout)
iplot(fig, filename='class')
```



Data exploration

In [15]:
```
class_0 = data_df.loc[data_df['Class'] == 0]["Time"]
class_1 = data_df.loc[data_df['Class'] == 1]["Time"]

hist_data = [class_0, class_1]
group_labels = ['Não Fraude', 'Fraude']

fig = ff.create_distplot(hist_data, group_labels, show_hist=False, show_rug=F
fig['layout'].update(title='Credit Card Transactions - Densidade Temporal', x
iplot(fig, filename='dist_only')
```

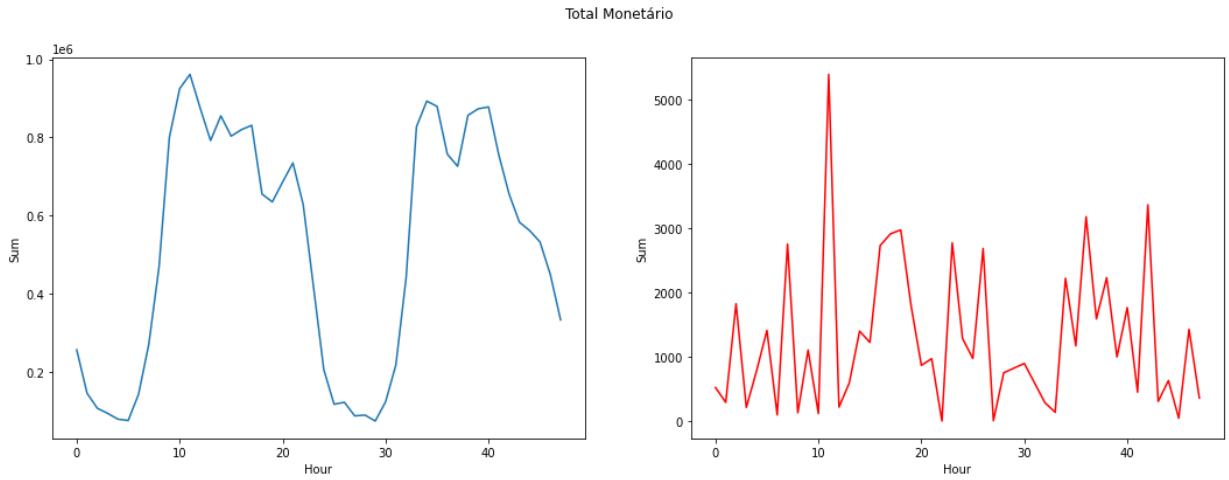
Out[15]:
```
'dist_only.html'
```

In [16]:
```
data_df['Hour'] = data_df['Time'].apply(lambda x: np.floor(x / 3600))

tmp = data_df.groupby(['Hour', 'Class'])['Amount'].aggregate(['min', 'max', '
df = pd.DataFrame(tmp)
df.columns = ['Hour', 'Class', 'Min', 'Max', 'Transactions', 'Sum', 'Mean', '
df.head()
```

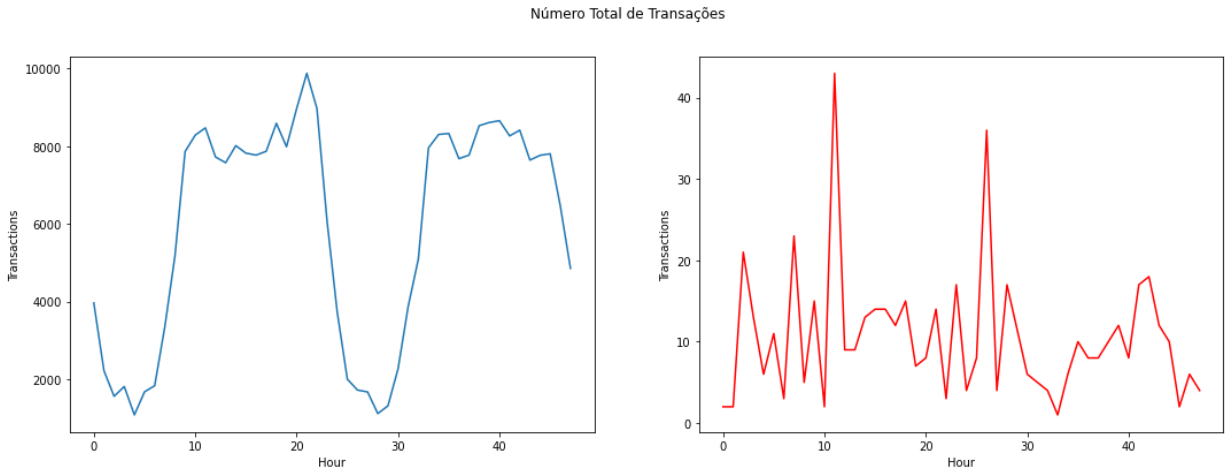
Out[16]:

	Hour	Class	Min	Max	Transactions	Sum	Mean	Median	Var
0	0.0	0	0.0	7712.43	3961	256572.87	64.774772	12.990	45615.821201
1	0.0	1	0.0	529.00	2	529.00	264.500000	264.500	139920.500000
2	1.0	0	0.0	1769.69	2215	145806.76	65.826980	22.820	20053.615770
3	1.0	1	59.0	239.93	2	298.93	149.465000	149.465	16367.832450
4	2.0	0	0.0	4002.88	1555	106989.39	68.803466	17.900	45355.430437

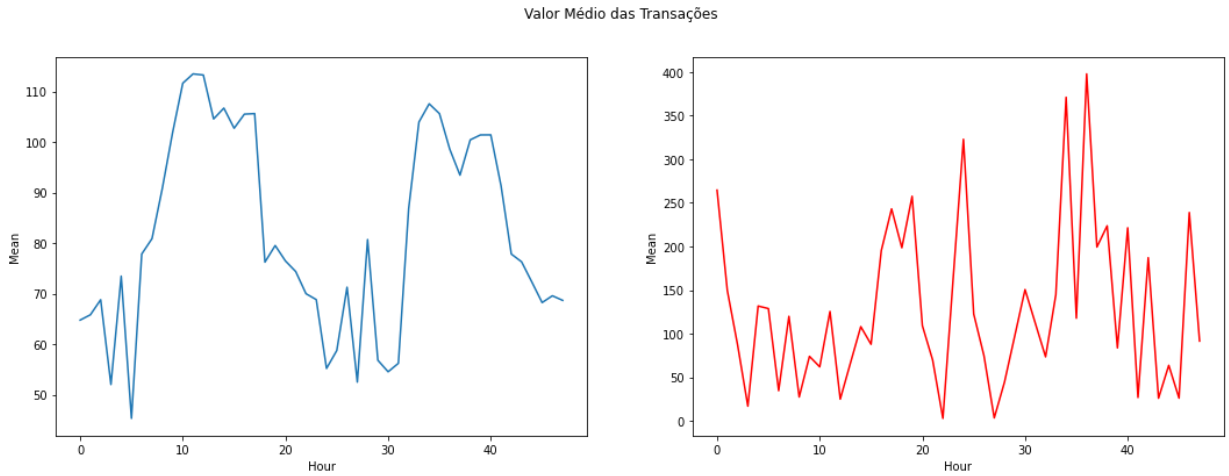
In [20]:
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))
s = sns.lineplot(ax = ax1, x="Hour", y="Sum", data=df.loc[df.Class==0])
s = sns.lineplot(ax = ax2, x="Hour", y="Sum", data=df.loc[df.Class==1], color
plt.suptitle("Total Monetário")
plt.show();
```



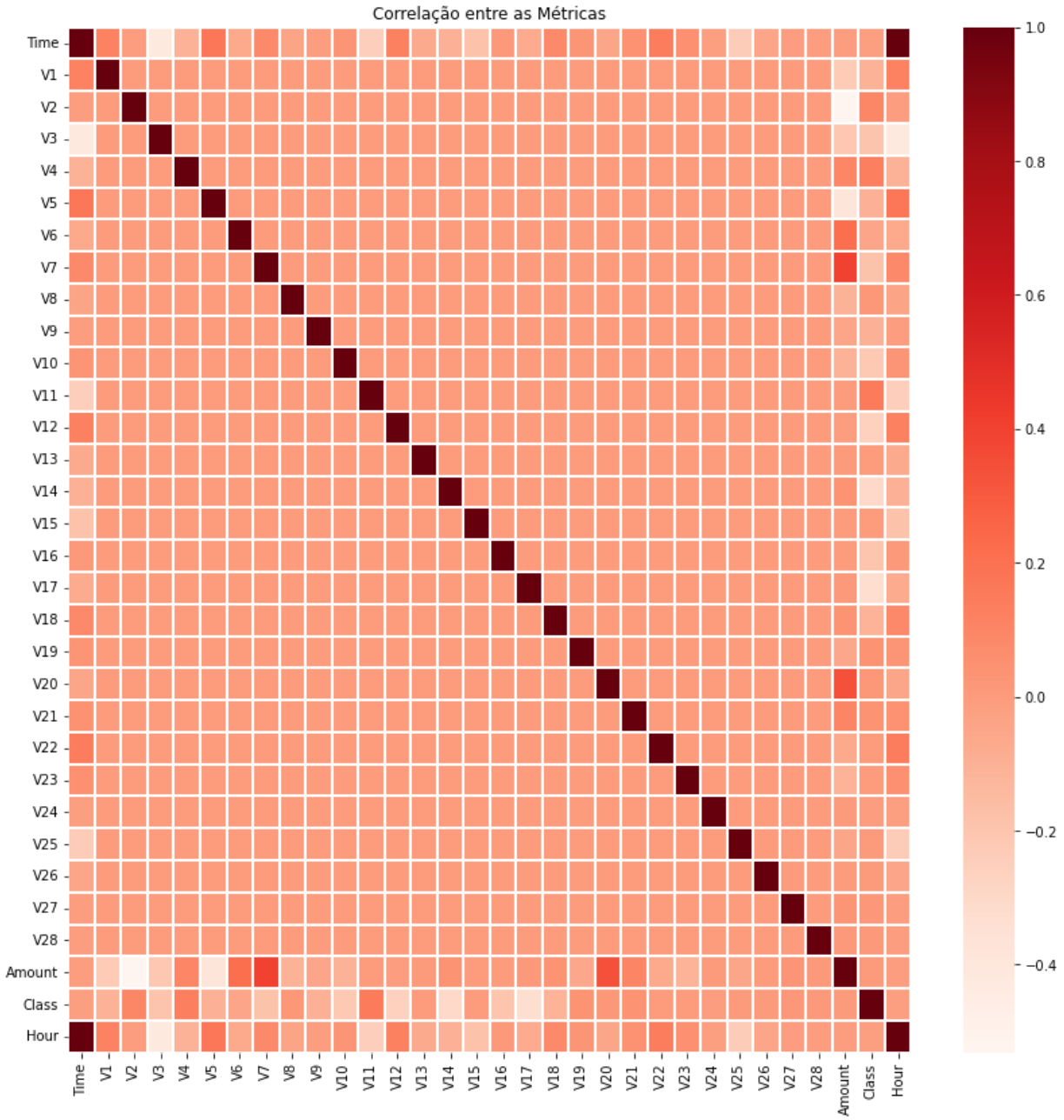
In [21]:
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))
s = sns.lineplot(ax = ax1, x="Hour", y="Transactions", data=df.loc[df.Class==
s = sns.lineplot(ax = ax2, x="Hour", y="Transactions", data=df.loc[df.Class==
plt.suptitle("Número Total de Transações")
plt.show();
```



In [23]:
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))
s = sns.lineplot(ax = ax1, x="Hour", y="Mean", data=df.loc[df.Class==0])
s = sns.lineplot(ax = ax2, x="Hour", y="Mean", data=df.loc[df.Class==1], colo
plt.suptitle("Valor Médio das Transações")
plt.show();
```



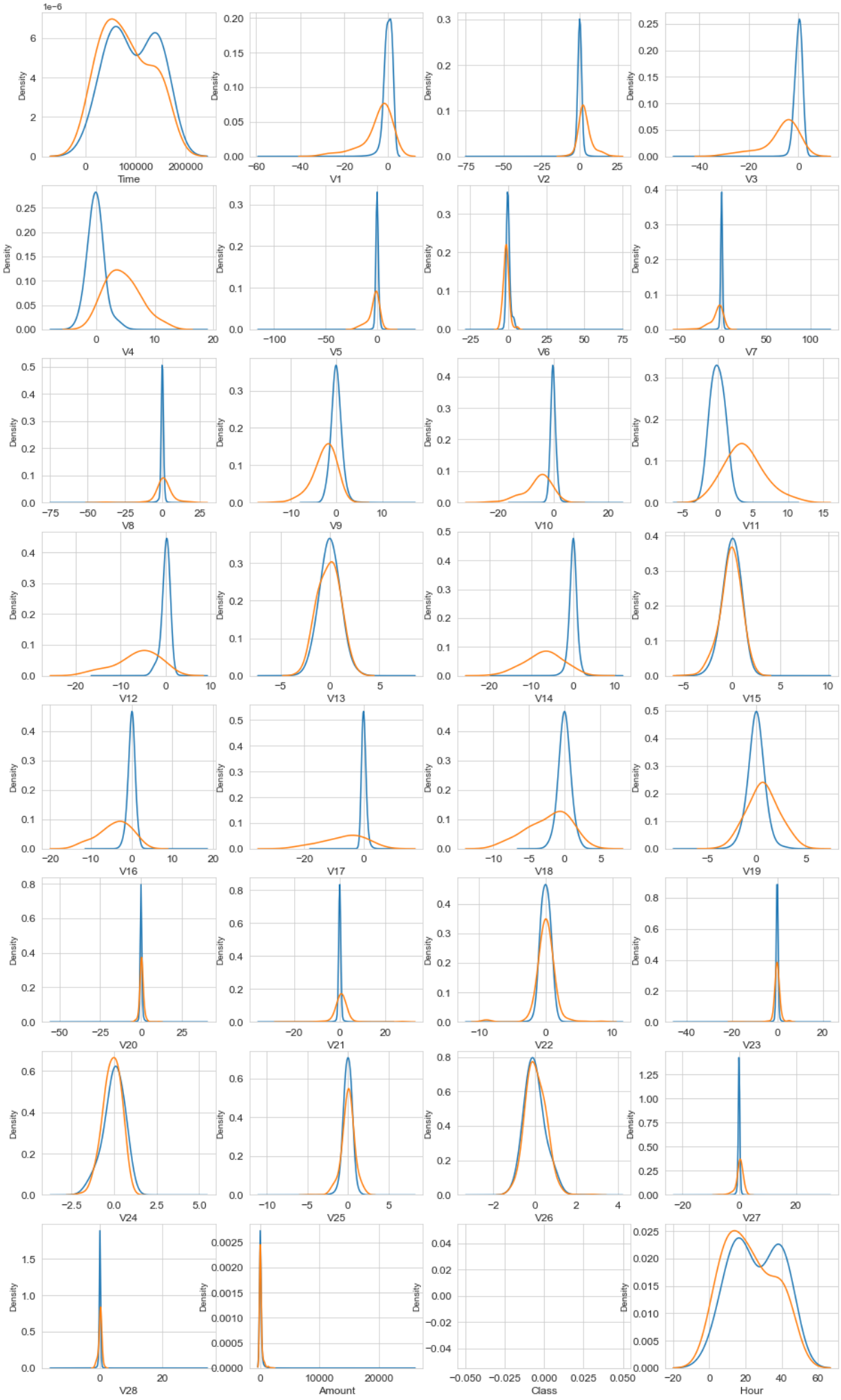
In [26]:
```
plt.figure(figsize = (14,14))
plt.title('Correlação entre as Métricas')
corr = data_df.corr()
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,linewidths
plt.show()
```



In [33]:

```
var = data_df.columns.values

i = 0
t0 = data_df.loc[data_df['Class'] == 0]
t1 = data_df.loc[data_df['Class'] == 1]

sns.set_style('whitegrid')
plt.figure()
fig, ax = plt.subplots(8,4,figsize=(16,28))

for feature in var:
    i += 1
    plt.subplot(8,4,i)
    sns.kdeplot(t0[feature], bw_method=0.5,label="Class = 0", warn_singular=F
    sns.kdeplot(t1[feature], bw_method=0.5,label="Class = 1", warn_singular=F
    plt.xlabel(feature, fontsize=12)
    locs, labels = plt.xticks()
    plt.tick_params(axis='both', which='major', labelsize=12)
plt.show();
```

```
<Figure size 432x288 with 0 Axes>
```


1e−6
Density
Time
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
V28
Amount
Class
Hour

Modelos Preditivos

Para avaliar qual seria o modelo implementado, algumas opções foram testadas:

1 - Regressão Logística

Existem dois tipos de mensuráveis, as variáveis (item sendo medido) e a variável alvo, que é o resultado.

Por exemplo, ao tentar prever se um aluno será aprovado ou reprovado em um teste, as horas estudadas são o recurso, e a variável de resposta terá dois valores - aprovado ou reprovado.

2 - Árvore de Decisão

Tais algoritmos subdividem progressivamente os dados em conjuntos cada vez menores e mais específicos, em termos de seus atributos, até atingirem um tamanho simplificado o bastante para ser rotulado. Para isso é necessário treinar o modelo com dados previamente rotulados, de modo a aplicá-lo a dados novos.

Em geral, uma árvore de decisão é composta por perguntas e respostas booleanas, classificando um indivíduo ou entidade de acordo com o conjunto de respostas obtidas pelo conjunto de perguntas formuladas.

3 - Floresta Aleatória

O Random Forest é um outro algoritmo de aprendizagem supervisionada, ele cria uma combinação (ensemble) de árvores de decisão usando a técnica de bagging que tem como objetivo reduzir a variância das previsões.

No Random Forest teremos diferentes árvores construídas a partir do mesmo conjunto de dados mas de forma aleatória, o modelo irá consultar cada uma dessas árvores e no final fazer uma votação baseado na classe majoritária quando estamos falando de classificação.

Nesta etapa, foi construido estes modelos de classificação com o auxílio da biblioteca Scikit-learn. Embora existam muitos outros modelos que poderiam ser utilizados, esses são os modelos mais populares usados para resolver problemas de classificação.

In [21]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
import seaborn as sns
sns.set()
%matplotlib inline
from termcolor import colored as cl # text customization
import itertools # advanced tools

from sklearn.preprocessing import StandardScaler # data normalization
from sklearn.model_selection import train_test_split # data split
from sklearn.tree import DecisionTreeClassifier # Decision tree algorithm
from sklearn.linear_model import LogisticRegression # Logistic regression alg
from sklearn.ensemble import RandomForestClassifier # Random forest tree algo
```

```
from sklearn.metrics import confusion_matrix # evaluation metric
from sklearn.metrics import accuracy_score # evaluation metric
from sklearn.metrics import f1_score # evaluation metric

df = pd.read_csv('creditcard.csv')
df.drop('Time', axis = 1, inplace = True)

print(df.head())
```

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

         V8        V9       V10  ...       V21       V22       V23       V24
\
0  0.098698  0.363787  0.090794  ... -0.018307  0.277838 -0.110474  0.066928
1  0.085102 -0.255425 -0.166974  ... -0.225775 -0.638672  0.101288 -0.339846
2  0.247676 -1.514654  0.207643  ...  0.247998  0.771679  0.909412 -0.689281
3  0.377436 -1.387024 -0.054952  ... -0.108300  0.005274 -0.190321 -1.175575
4 -0.270533  0.817739  0.753074  ... -0.009431  0.798278 -0.137458  0.141267

        V25       V26       V27       V28  Amount  Class
0  0.128539 -0.189115  0.133558 -0.021053  149.62      0
1  0.167170  0.125895 -0.008983  0.014724    2.69      0
2 -0.327642 -0.139097 -0.055353 -0.059752  378.66      0
3  0.647376 -0.221929  0.062723  0.061458  123.50      0
4 -0.206010  0.502292  0.219422  0.215153   69.99      0

[5 rows x 30 columns]
```

In [13]:

```
class_names = {0:'Not Fraud', 1:'Fraud'}
print(df.Class.value_counts().rename(index = class_names))
```

```
Not Fraud    284315
Fraud           492
Name: Class, dtype: int64
```

Ao analisar as estatísticas, é visto que os valores na métrica *Amout* estão variando bastante quando comparados com o resto das variáveis. Para reduzir sua gama de valores, podemos normalizá-la usando o método 'StandardScaler'.

In [23]:

```
sc = StandardScaler()
amount = df['Amount'].values

df['Amount'] = sc.fit_transform(amount.reshape(-1, 1))

print(cl(df['Amount'].head(10), attrs = ['bold']))
```

```
0    0.244964
1   -0.342475
2    1.160686
3    0.140534
4   -0.073403
5   -0.338556
6   -0.333279
7   -0.190107
8    0.019392
```

```
9   -0.338516
Name: Amount, dtype: float64
```

Seleção de Métricas

Pronto. Agora são definidas as métricas e a variável objetivo.

In [24]:
```
feature_names = df.iloc[:, 1:30].columns
target = df.iloc[:1, 30: ].columns
print(feature_names)
print(target)
```

```
Index(['V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12',
       'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21', 'V22',
       'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount', 'Class'],
      dtype='object')
Index([], dtype='object')
```

In [25]:
```
data_features = df[feature_names]
data_target = df[target]
```

Data Split

Nesse passo, o dataframe é separado em dois: um para treino e outro para teste.

In [27]:
```
X = df.drop('Class', axis = 1).values
y = df['Class'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ra

print(cl('X_train samples : ', attrs = ['bold']), X_train[:1])
print(cl('X_test samples : ', attrs = ['bold']), X_test[0:1])
print(cl('y_train samples : ', attrs = ['bold']), y_train[0:20])
print(cl('y_test samples : ', attrs = ['bold']), y_test[0:20])
```

```
X_train samples :  [[-1.11504743  1.03558276  0.80071244 -1.06039825  0.032621
17  0.85342216
  -0.61424348 -3.23116112  1.53994798 -0.81690879 -1.30559201  0.1081772
  -0.85960958 -0.07193421  0.90665563 -1.72092961  0.79785322 -0.0067594
   1.95677806 -0.64489556  3.02038533 -0.53961798  0.03315649 -0.77494577
   0.10586781 -0.43085348  0.22973694 -0.0705913  -0.30145418]]
X_test samples :  [[-0.32333357  1.05745525 -0.04834115 -0.60720431  1.2598211
5 -0.09176072
   1.1591015  -0.12433461 -0.17463954 -1.64440065 -1.11886302  0.20264731
   1.14596495 -1.80235956 -0.24717793 -0.06094535  0.84660574  0.37945439
   0.84726224  0.18640942 -0.20709827 -0.43389027 -0.26161328 -0.04665061
   0.2115123   0.00829721  0.10849443  0.16113917 -0.19330595]]
y_train samples :  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
y_test samples :  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Modelos

In [28]:
```
# 1. Logistic Regression

lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_yhat = lr.predict(X_test)
```

```
# 2. Decision Tree

tree_model = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
tree_model.fit(X_train, y_train)
tree_yhat = tree_model.predict(X_test)

# 3. Random Forest Tree

rf = RandomForestClassifier(max_depth = 4)
rf.fit(X_train, y_train)
rf_yhat = rf.predict(X_test)
```

Avaliação

In [39]:

```
# 1. Accuracy score

print(cl('Acurácia', attrs = ['bold']))
print(cl('------------------------------------------------------------------------
print(cl('Valor de acurácia do modelo de Regressão Logística: {}'.format(accu
print(cl('------------------------------------------------------------------------
print(cl('Valor de acurácia do modelo de Árvore de Decisão: {}'.format(accura
print(cl('------------------------------------------------------------------------
print(cl('Valor de acurácia do modelo de Floresta Aleatória: {}'.format(accur
print(cl('------------------------------------------------------------------------
```

```
Acurácia
------------------------------------------------------------------------
Valor de acurácia do modelo de Regressão Logística: 0.9991924440855307
------------------------------------------------------------------------
Valor de acurácia do modelo de Árvore de Decisão: 0.9993679997191109
------------------------------------------------------------------------
Valor de acurácia do modelo de Floresta Aleatória: 0.9992977774656788
------------------------------------------------------------------------
```

In [41]:

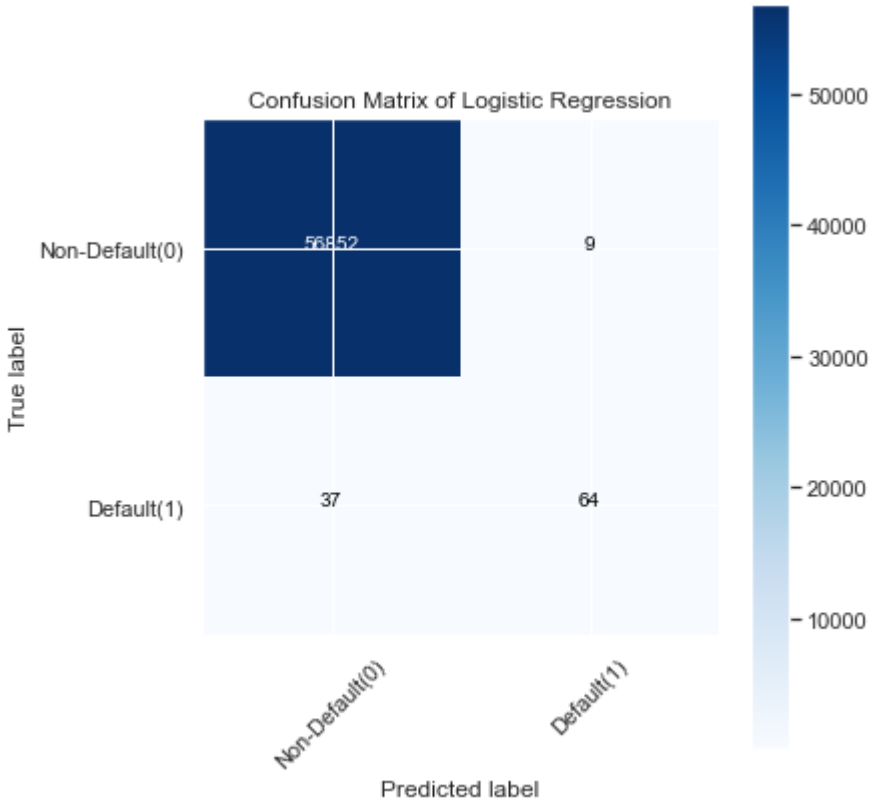
```
# 2. F1 score

print(cl('F1', attrs = ['bold']))
print(cl('------------------------------------------------------------------------
print(cl('F1 do modelo de Regressão Logística: {}'.format(f1_score(y_test, lr
print(cl('------------------------------------------------------------------------
print(cl('F1 do modelo de Árvore de Decisão: {}'.format(f1_score(y_test, tree
print(cl('------------------------------------------------------------------------
print(cl('F1 do modelo de Floresta Aleatória: {}'.format(f1_score(y_test, rf_
print(cl('------------------------------------------------------------------------
```

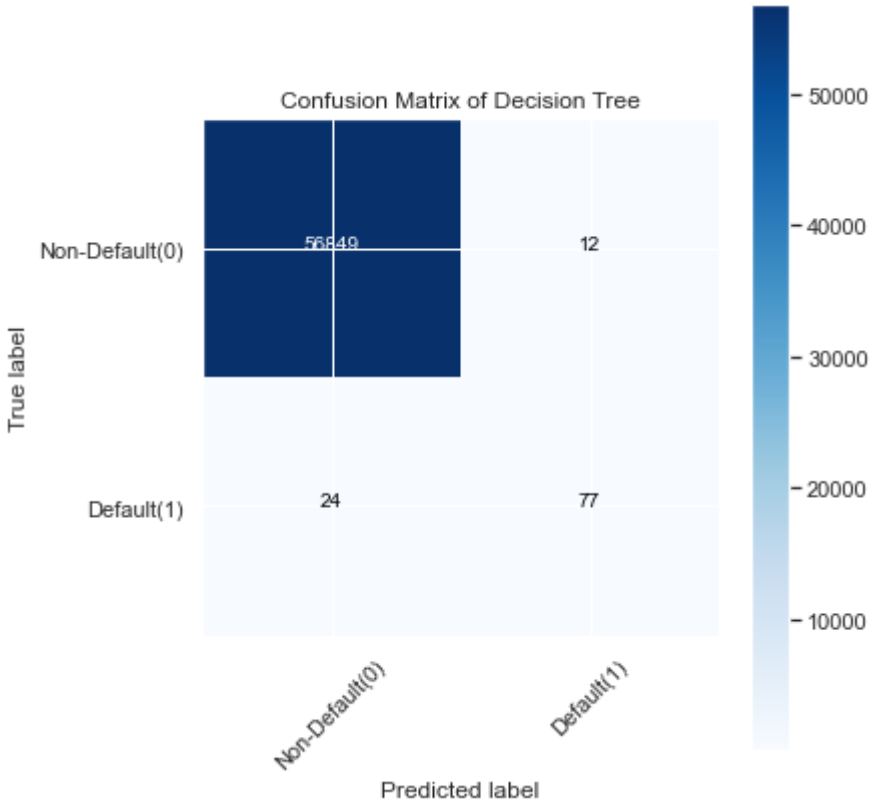
```
F1
------------------------------------------------------------------------
F1 do modelo de Regressão Logística: 0.7356321839080459
------------------------------------------------------------------------
F1 do modelo de Árvore de Decisão: 0.8105263157894738
------------------------------------------------------------------------
F1 do modelo de Floresta Aleatória: 0.7727272727272727
------------------------------------------------------------------------
```

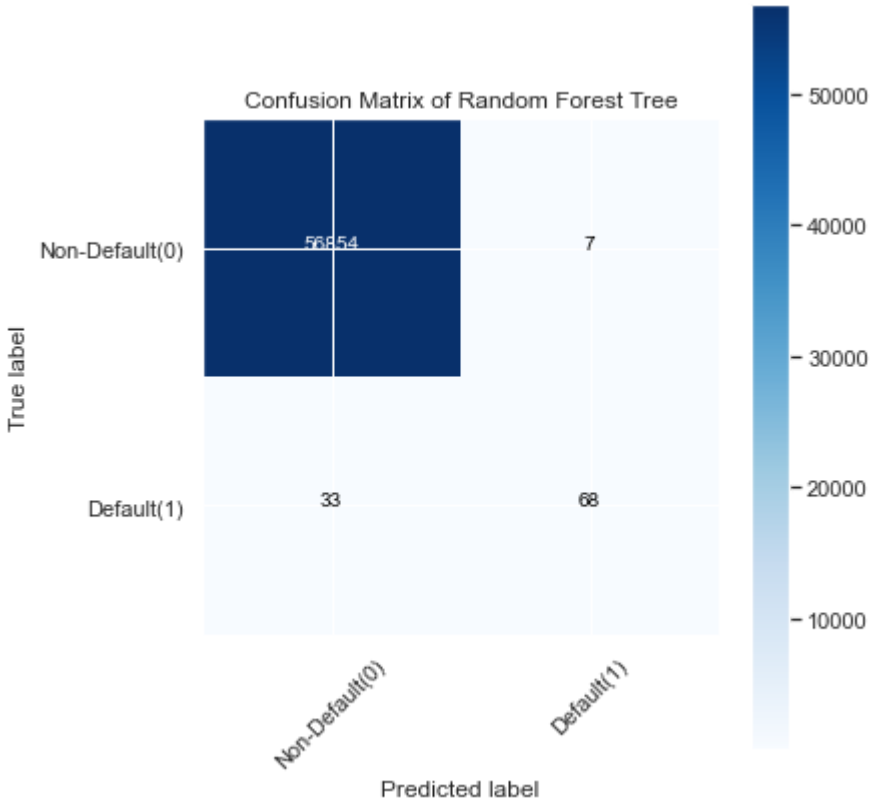
In [44]:

```
# 3. Confusion Matrix

# defining the plot function

def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.c
    title = 'Confusion Matrix of {}'.format(title)
```

```
    if normalize:
        cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment = 'center',
                 color = 'white' if cm[i, j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Compute confusion matrix for the models

tree_matrix = confusion_matrix(y_test, tree_yhat, labels = [0, 1]) # Decision
lr_matrix = confusion_matrix(y_test, lr_yhat, labels = [0, 1]) # Logistic Reg
rf_matrix = confusion_matrix(y_test, rf_yhat, labels = [0, 1]) # Random Fores

# Plot the confusion matrix

plt.rcParams['figure.figsize'] = (6, 6)


# 1. Logistic regression

lr_cm_plot = plot_confusion_matrix(lr_matrix,
                                   classes = ['Non-Default(0)','Default(1)'],
                                   normalize = False, title = 'Logistic Regressi
plt.savefig('lr_cm_plot.png')
plt.show()

# 2. Decision tree

tree_cm_plot = plot_confusion_matrix(tree_matrix,
                                   classes = ['Non-Default(0)','Default(1)'],
                                   normalize = False, title = 'Decision Tree')
plt.savefig('tree_cm_plot.png')
plt.show()

# 3. Random forest tree

rf_cm_plot = plot_confusion_matrix(rf_matrix,
                                   classes = ['Non-Default(0)','Default(1)'],
                                   normalize = False, title = 'Random Forest Tre
plt.savefig('rf_cm_plot.png')
plt.show()
```


Confusion Matrix of Logistic Regression
Non-Default(0)
Default(1)
True label
56852
9
37
64
Non-Default(0)
Default(1)
Predicted label
50000
40000
30000
20000
10000


Confusion Matrix of Decision Tree
Non-Default(0)
Default(1)
True label
56849
12
24
77
Non-Default(0)
Default(1)
Predicted label
50000
40000
30000
20000
10000

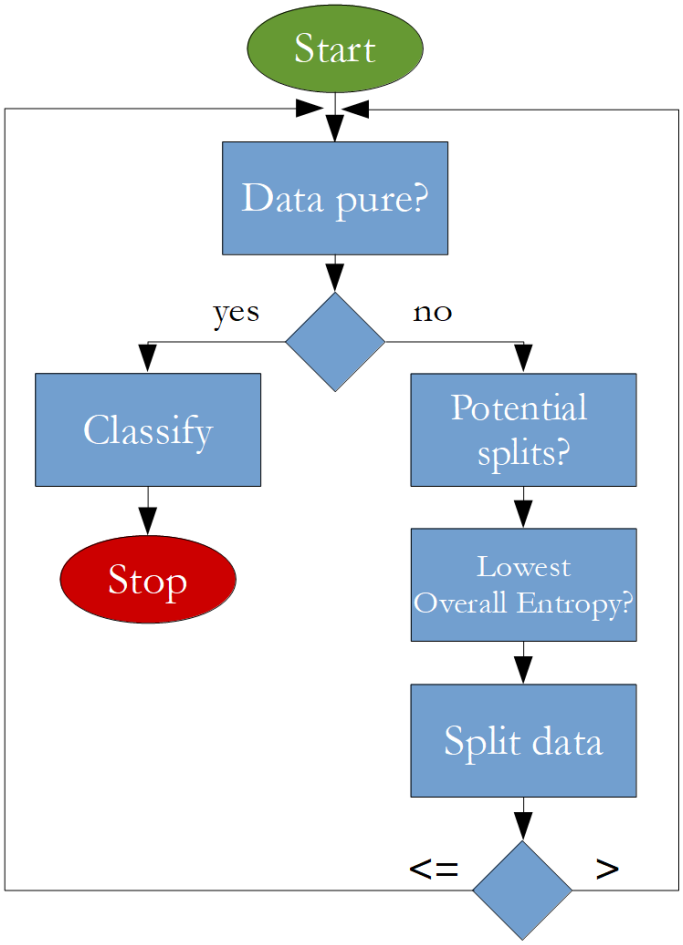

Confusion Matrix of Random Forest Tree
Non-Default(0)
Default(1)
True label
56854
7
33
68
Non-Default(0)
Default(1)
Predicted label
50000
40000
30000
20000
10000

Implementação

O objetivo deste projeto é programar uma Árvore de Decisão de Classificação que pode ser utilizada mediante a seguinte API:

```
df = pd.read_csv("data.csv")

train_df, test_df = train_test_split(df, test_size=0.2) #Step 1
tree = decision_tree_algorithm(train_df, ml_task="classification")
#Step 2
accuracy = calculate_accuracy(test_df, tree) #Step 3
```

O algoritmo que será implementado seguirá o seguinte modelo:



Importação de bibliotecas

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import random
from pprint import pprint
```

In [2]:
```
%matplotlib inline
sns.set_style("darkgrid")
```

Carregamento e Preparo dos Dados

Formato do data set

- A última coluna do data frame deve conter a coluna classificatória e deve se chamar "label".
- Não pode haver valores faltantes no data frame.

In [3]:
```
df = pd.read_csv("creditcard.csv") #Carregando o dataset
df = df.rename(columns={df.columns[-1]: "label"}) #Renomeando última coluna
df = df.drop(["Time"], axis=1) #Excluindo colunas desnecessárias
```

In [5]:
```
print(df.shape)
df.head()
```

```
(284807, 30)
```

Out[5]:

	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	0.
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0

5 rows × 30 columns

In [6]:
```
class_names = {0:'Not Fraud', 1:'Fraud'}
print(df.label.value_counts().rename(index = class_names))
```

```
Not Fraud    284315
Fraud           492
Name: label, dtype: int64
```

Separação de Dados para Teste

Para o *#Step 1* da API, é chamada uma função, *train_test_split* que recebe 2 parâmetros: 1) *parametro 1*: dataframe que se deseja obter a classificação. Nesse caso, *df*. 2) *parametro 2*: *test_size* é o tamanho desejado da amostra que será utilizada do dataframe para testar o algoritmo de classificação. Pode ser o número de linhas ou uma proporção. Nesse caso foi determinado *0,2* - que representa *20%* do dataframe.

A função *train_test_size* executa essa separação e retorna dois dataframes: 1) *train_df*: dataframe que será utilizado no treinamento do algoritmo. 2) *test_df*: dataframe que será utilizado para testar a acurácia do modelo de classificação gerado.

In [7]:
```
def train_test_split(df, test_size):

    if isinstance(test_size, float):
        test_size = round(test_size * len(df)) #Caso o valor de test_size pas

    indices = df.index.tolist() #Passando os indices para uma lista
    test_indices = random.sample(population=indices, k=test_size) #Pegando um
```

```
    test_df = df.loc[test_indices] #Selecionando as linhas
    train_df = df.drop(test_indices) #Retirando as linhas selecionadas do dat

    return train_df, test_df
```

In [8]:
```
random.seed(0) #Definindo uma semente para o gerador random a fim de poder-se
train_df, test_df = train_test_split(df, test_size=0.2)
```

Funções Auxiliares

No decorrer da criação desta API, houve a necessidade de modificar a biblioteca utilizada para a manipulação do dataset. Notou-se que, utilizando um array bidimensional NumPy, o algoritmo ficou cerca de 10 vezes mais rápido. Sendo assim, criou-se uma variável chamada *data* contendo os valores do dataframe *train_df*.

In [9]:
```
data = train_df.values
data[:5]
```

Out[9]:
```
array([[-1.35980713, -0.07278117,  2.53634674,  1.37815522, -0.33832077,
         0.46238778,  0.23959855,  0.0986979 ,  0.36378697,  0.09079417,
        -0.55159953, -0.61780086, -0.99138985, -0.31116935,  1.46817697,
        -0.47040053,  0.20797124,  0.02579058,  0.40399296,  0.2514121 ,
        -0.01830678,  0.27783758, -0.11047391,  0.06692807,  0.12853936,
        -0.18911484,  0.13355838, -0.02105305,  0.24496426,  0.        ],
       [ 1.19185711,  0.26615071,  0.16648011,  0.44815408,  0.06001765,
        -0.08236081, -0.07880298,  0.08510165, -0.25542513, -0.16697441,
         1.61272666,  1.06523531,  0.48909502, -0.1437723 ,  0.63555809,
         0.46391704, -0.11480466, -0.18336127, -0.14578304, -0.06908314,
        -0.22577525, -0.63867195,  0.10128802, -0.33984648,  0.1671704 ,
         0.12589453, -0.0089831 ,  0.01472417, -0.34247454,  0.        ],
       [-1.35835406, -1.34016307,  1.77320934,  0.37977959, -0.50319813,
         1.80049938,  0.79146096,  0.24767579, -1.51465432,  0.20764287,
         0.62450146,  0.06608369,  0.71729273, -0.16594592,  2.34586495,
        -2.89008319,  1.10996938, -0.12135931, -2.2618571 ,  0.52497973,
         0.24799815,  0.7716794 ,  0.90941226, -0.68928096, -0.32764183,
        -0.13909657, -0.05535279, -0.05975184,  1.16068593,  0.        ],
       [-0.96627171, -0.18522601,  1.79299334, -0.86329128, -0.01030888,
         1.24720317,  0.23760894,  0.37743587, -1.38702406, -0.05495192,
        -0.22648726,  0.17822823,  0.50775687, -0.28792375, -0.63141812,
        -1.05964725, -0.68409279,  1.965775  , -1.23262197, -0.20803778,
        -0.10830045,  0.0052736 , -0.19032052, -1.17557533,  0.64737603,
        -0.22192884,  0.06272285,  0.06145763,  0.14053425,  0.        ],
       [-1.15823309,  0.87773675,  1.54871785,  0.40303393, -0.40719338,
         0.09592146,  0.59294075, -0.27053268,  0.81773931,  0.75307443,
        -0.82284288,  0.53819555,  1.34585159, -1.11966983,  0.17512113,
        -0.45144918, -0.23703324, -0.03819479,  0.80348692,  0.40854236,
        -0.0094307 ,  0.79827849, -0.13745808,  0.14126698, -0.20600959,
         0.50229222,  0.21942223,  0.21515315, -0.07340334,  0.        ]])
```

Dados Puros?

Aqui é testada a pureza dos dados em relação a coluna de classificação.

- Caso haja apenas uma classe no dataframe passado à função, ela retorna *True*.
- Caso contrário, retorna *False*.

Essa função é importante para que se possa gerar a classificação dos dados. É aqui que é identificado quantas classes existem no dataframe e quais são elas.

In [10]:

```
def check_purity(data):

    label_column = data[:, -1] #Selecionando a coluna de classificação
    unique_classes = np.unique(label_column) #Selecionando as classes que exi

    if len(unique_classes) == 1:
        return True
    else:
        return False
```

Classificação

Essa função é responsável por identificar a classe que mais aparece no dataframe.

In [11]:

```
def classify_data(data):

    label_column = data[:, -1] #Selecionando a coluna de classificação
    unique_classes, counts_unique_classes = np.unique(label_column, return_co

    index = counts_unique_classes.argmax() #Retornando o index do elemento ma
    classification = unique_classes[index] #Retornando o valor da classe.

    return classification
```

Possíveis clusters

Essa função permite que o algoritmo seja implementado em dataframes em que não se sabe, de antemão, quantas classes existem. Isso permite que este algoritmo seja implementado para outras classificações além da binária. Ela retorna um *array* com os possíveis clusters.

In [12]:

```
def get_potential_splits(data):

    potential_splits = {}
    _, n_columns = data.shape
    for column_index in range(n_columns - 1):           #Excluindo a coluna de
        values = data[:, column_index]
        unique_values = np.unique(values)

        potential_splits[column_index] = unique_values

    return potential_splits
```

Split Data

Esta função separa o data frame em dois: um com valores que estão abaixo de um valor previamente estabelecido e outro com os valores acima.

In [13]:

```
def split_data(data, split_column, split_value):

    split_column_values = data[:, split_column]

    type_of_feature = FEATURE_TYPES[split_column]
    if type_of_feature == "continuous":
        data_below = data[split_column_values <= split_value]
        data_above = data[split_column_values >  split_value]
```

```
    # feature is categorical
    else:
        data_below = data[split_column_values == split_value]
        data_above = data[split_column_values != split_value]

    return data_below, data_above
```

Menor Entropia Absoluta?

In [14]:
```
def calculate_entropy(data):

    label_column = data[:, -1]
    _, counts = np.unique(label_column, return_counts=True)

    probabilities = counts / counts.sum()
    entropy = sum(probabilities * -np.log2(probabilities))

    return entropy
```

In [15]:
```
def calculate_overall_entropy(data_below, data_above):

    n = len(data_below) + len(data_above)
    p_data_below = len(data_below) / n
    p_data_above = len(data_above) / n

    overall_entropy =  (p_data_below * calculate_entropy(data_below)
                      + p_data_above * calculate_entropy(data_above))

    return overall_entropy
```

In [16]:
```
def determine_best_split(data, potential_splits):

    overall_entropy = 9999
    for column_index in potential_splits:
        for value in potential_splits[column_index]:
            data_below, data_above = split_data(data, split_column=column_ind
            current_overall_entropy = calculate_overall_entropy(data_below, d

            if current_overall_entropy <= overall_entropy:
                overall_entropy = current_overall_entropy
                best_split_column = column_index
                best_split_value = value

    return best_split_column, best_split_value
```

Decision Tree Algorithm

Representação da Árvore de Decisão

In [17]:
```
sub_tree = {"question": ["yes_answer",
                         "no_answer"]}
```

Determinação do Tipo de Métrica

In [18]:
```
def determine_type_of_feature(df):

    feature_types = []
    n_unique_values_treshold = 15
    for feature in df.columns:
        if feature != "label":
            unique_values = df[feature].unique()
            example_value = unique_values[0]

            if (isinstance(example_value, str)) or (len(unique_values) <= n_u
                feature_types.append("categorical")
            else:
                feature_types.append("continuous")

    return feature_types
```

Algoritmo

In [19]:
```
def decision_tree_algorithm(df, counter=0, min_samples=2, max_depth=5):

    # data preparations
    if counter == 0:
        global COLUMN_HEADERS, FEATURE_TYPES
        COLUMN_HEADERS = df.columns
        FEATURE_TYPES = determine_type_of_feature(df)
        data = df.values
    else:
        data = df


    # base cases
    if (check_purity(data)) or (len(data) < min_samples) or (counter == max_d
        classification = classify_data(data)

        return classification


    # recursive part
    else:
        counter += 1

        # helper functions
        potential_splits = get_potential_splits(data)
        split_column, split_value = determine_best_split(data, potential_spli
        data_below, data_above = split_data(data, split_column, split_value)

        # check for empty data
        if len(data_below) == 0 or len(data_above) == 0:
            classification = classify_data(data)
            return classification

        # determine question
        feature_name = COLUMN_HEADERS[split_column]
        type_of_feature = FEATURE_TYPES[split_column]
        if type_of_feature == "continuous":
            question = "{} <= {}".format(feature_name, split_value)

        # feature is categorical
        else:
            question = "{} = {}".format(feature_name, split_value)

        # instantiate sub-tree
```

```
        sub_tree = {question: []}

        # find answers (recursion)
        yes_answer = decision_tree_algorithm(data_below, counter, min_samples
        no_answer = decision_tree_algorithm(data_above, counter, min_samples,

        # If the answers are the same, then there is no point in asking the q
        # This could happen when the data is classified even though it is not
        # yet (min_samples or max_depth base case).
        if yes_answer == no_answer:
            sub_tree = yes_answer
        else:
            sub_tree[question].append(yes_answer)
            sub_tree[question].append(no_answer)

        return sub_tree
```

In [31]:

```
tree = decision_tree_algorithm(train_df, max_depth=3)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/var/folders/np/sdvcmn9166l6z2_x7dl6z9m40000gn/T/ipykernel_88758/2824870231.py
in <module>
----> 1 tree = decision_tree_algorithm(train_df, max_depth=3)

/var/folders/np/sdvcmn9166l6z2_x7dl6z9m40000gn/T/ipykernel_88758/1509369994.py
in decision_tree_algorithm(df, counter, min_samples, max_depth)
     24         # helper functions
     25         potential_splits = get_potential_splits(data)
---> 26         split_column, split_value = determine_best_split(data, potenti
al_splits)
     27         data_below, data_above = split_data(data, split_column, split_
value)
     28

/var/folders/np/sdvcmn9166l6z2_x7dl6z9m40000gn/T/ipykernel_88758/2364581489.py
in determine_best_split(data, potential_splits)
      4     for column_index in potential_splits:
      5         for value in potential_splits[column_index]:
----> 6             data_below, data_above = split_data(data, split_column=col
umn_index, split_value=value)
      7             current_overall_entropy = calculate_overall_entropy(data_b
elow, data_above)
      8

/var/folders/np/sdvcmn9166l6z2_x7dl6z9m40000gn/T/ipykernel_88758/120946100.py
 in split_data(data, split_column, split_value)
      5     type_of_feature = FEATURE_TYPES[split_column]
      6     if type_of_feature == "continuous":
----> 7         data_below = data[split_column_values <= split_value]
      8         data_above = data[split_column_values >  split_value]
      9

KeyboardInterrupt:
```

Matriz de Confusão

In [22]:

```
from sklearn.metrics import confusion_matrix
class_names = ['Não fraude', 'Fraude']
matrix = confusion_matrix(y_test, prediction)
# Create pandas dataframe
```

```
dataframe = pd.DataFrame(matrix, index=class_names, columns=class_names)
# Create heatmap
sns.heatmap(dataframe, annot=True, cbar=None, cmap="Blues", fmt = 'g')
plt.title("Matriz de Confusão"), plt.tight_layout()
plt.ylabel("Classe Real"), plt.xlabel("Classe Predita")
plt.show()
```

Calculo de Acurácia

In [23]:
```
def calculate_accuracy(df, tree):

    df["classification"] = df.apply(classify_example, axis=1, args=(tree,))
    df["classification_correct"] = df["classification"] == df["label"]

    accuracy = df["classification_correct"].mean()

    return accuracy
```

In [30]:
```
accuracy = calculate_accuracy(test_df, tree)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/var/folders/np/sdvcmn9166l6z2_x7dl6z9m40000gn/T/ipykernel_88758/3156084455.py
in <module>
----> 1 accuracy = calculate_accuracy(test_df, tree)
      2 print(cl('Valor de acurácia do modelo de Árvore de Decisão: {}'.format
(accuracy), attrs = ['bold'], color='green'))

NameError: name 'tree' is not defined
```

In [29]:
```
from termcolor import colored as cl # text customization

print(cl('Valor de acurácia do modelo de Árvore de Decisão: {}'.format(accura
```

Valor de acurácia do modelo de Árvore de Decisão: 0.9992275552122467