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'
```

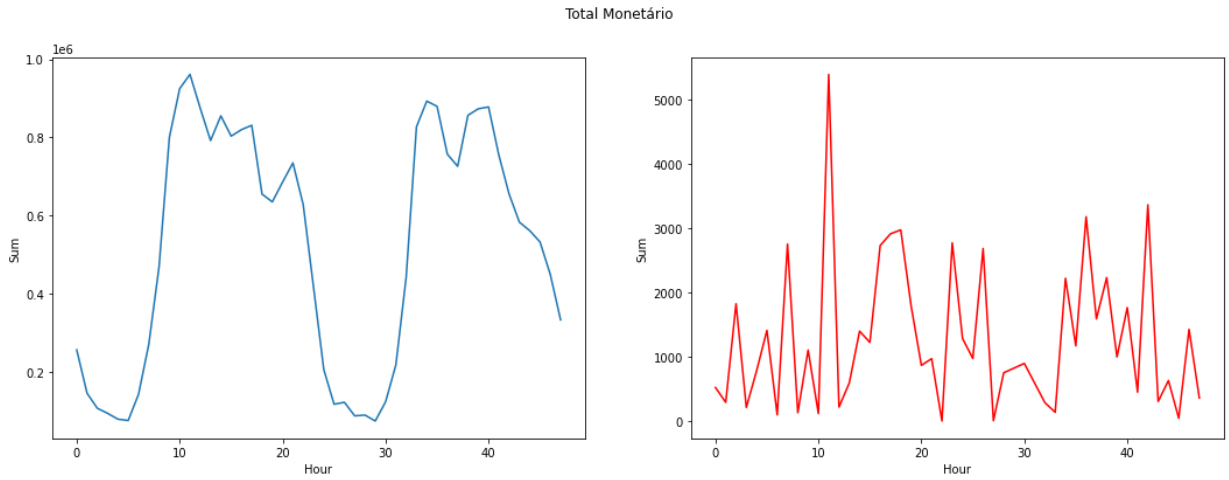
In [16]:
```
data_df['Hour'] = data_df['Time'].apply(lambda x: np.floor(x / 3600))

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()
```

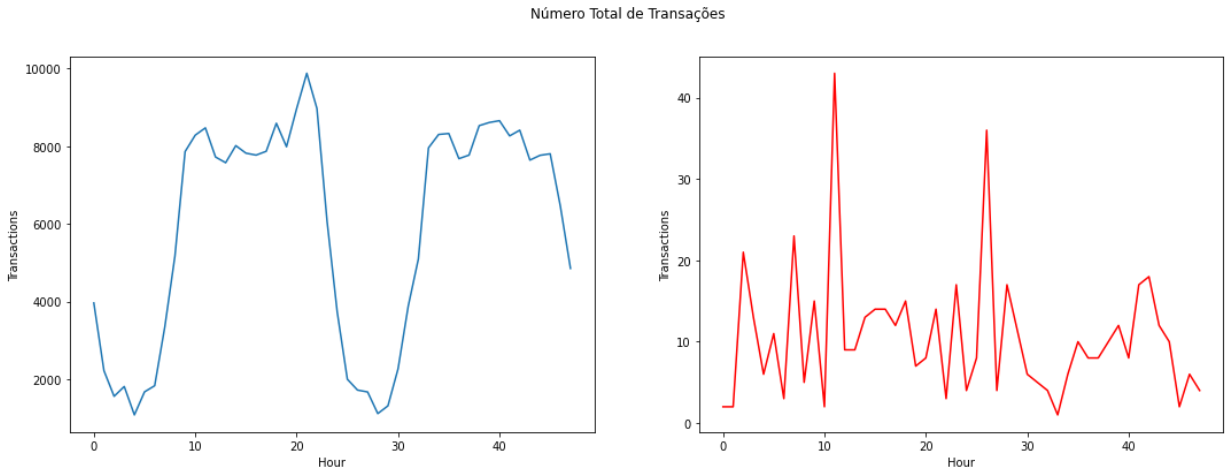
Out[16]:

	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

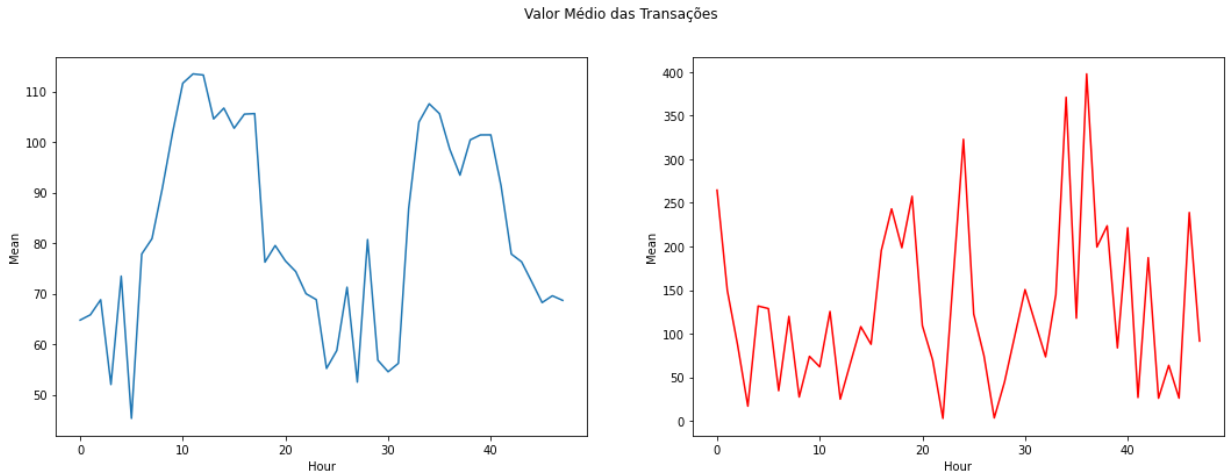
In [20]:
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))
s = sns.lineplot(ax = ax1, x="Hour", y="Sum", data=df.loc[df.Class==0])
s = sns.lineplot(ax = ax2, x="Hour", y="Sum", data=df.loc[df.Class==1], color
plt.suptitle("Total Monetário")
plt.show();
```



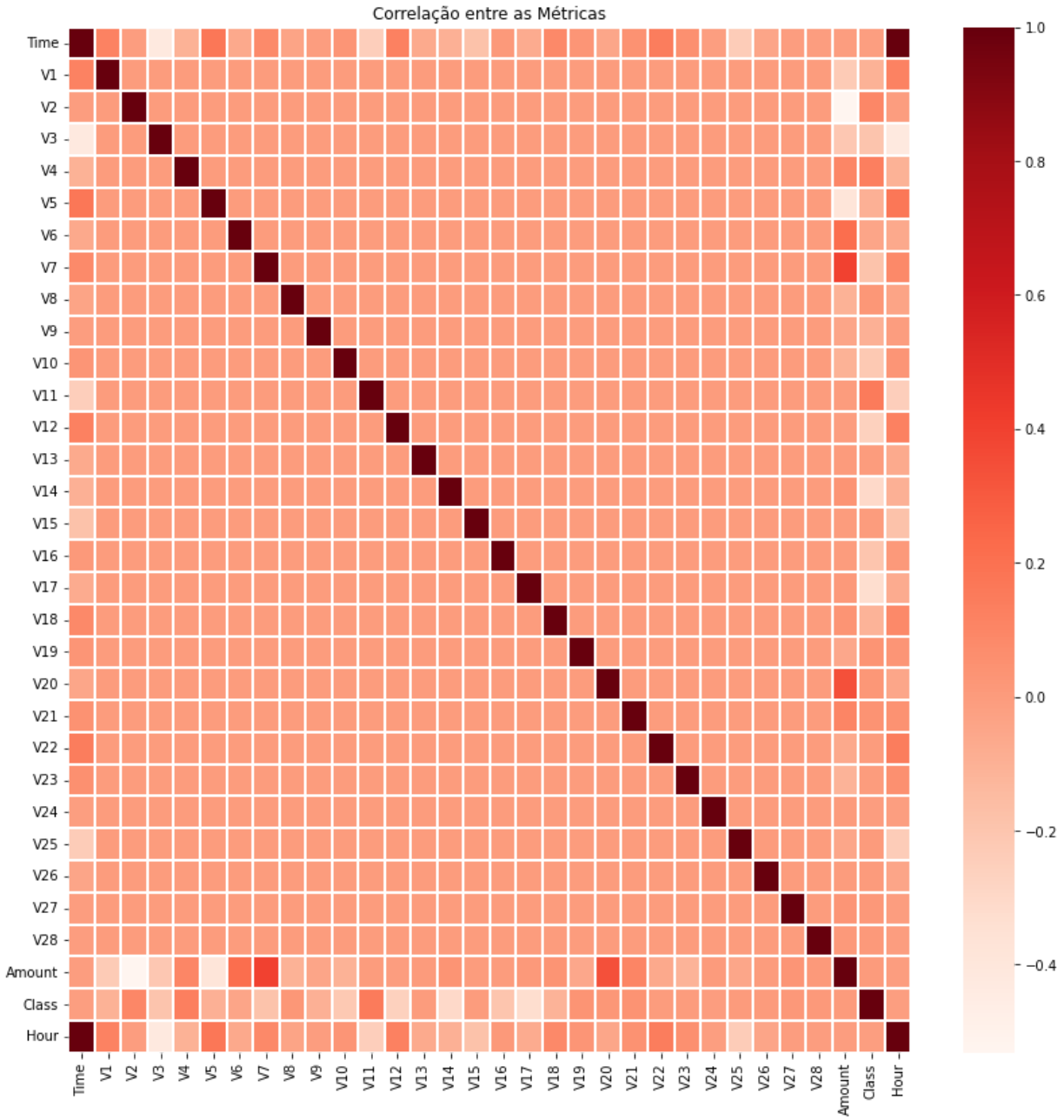
In [21]:
```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))
s = sns.lineplot(ax = ax1, x="Hour", y="Transactions", data=df.loc[df.Class==
s = sns.lineplot(ax = ax2, x="Hour", y="Transactions", data=df.loc[df.Class==
plt.suptitle("Número Total de Transações")
plt.show();
```



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]:

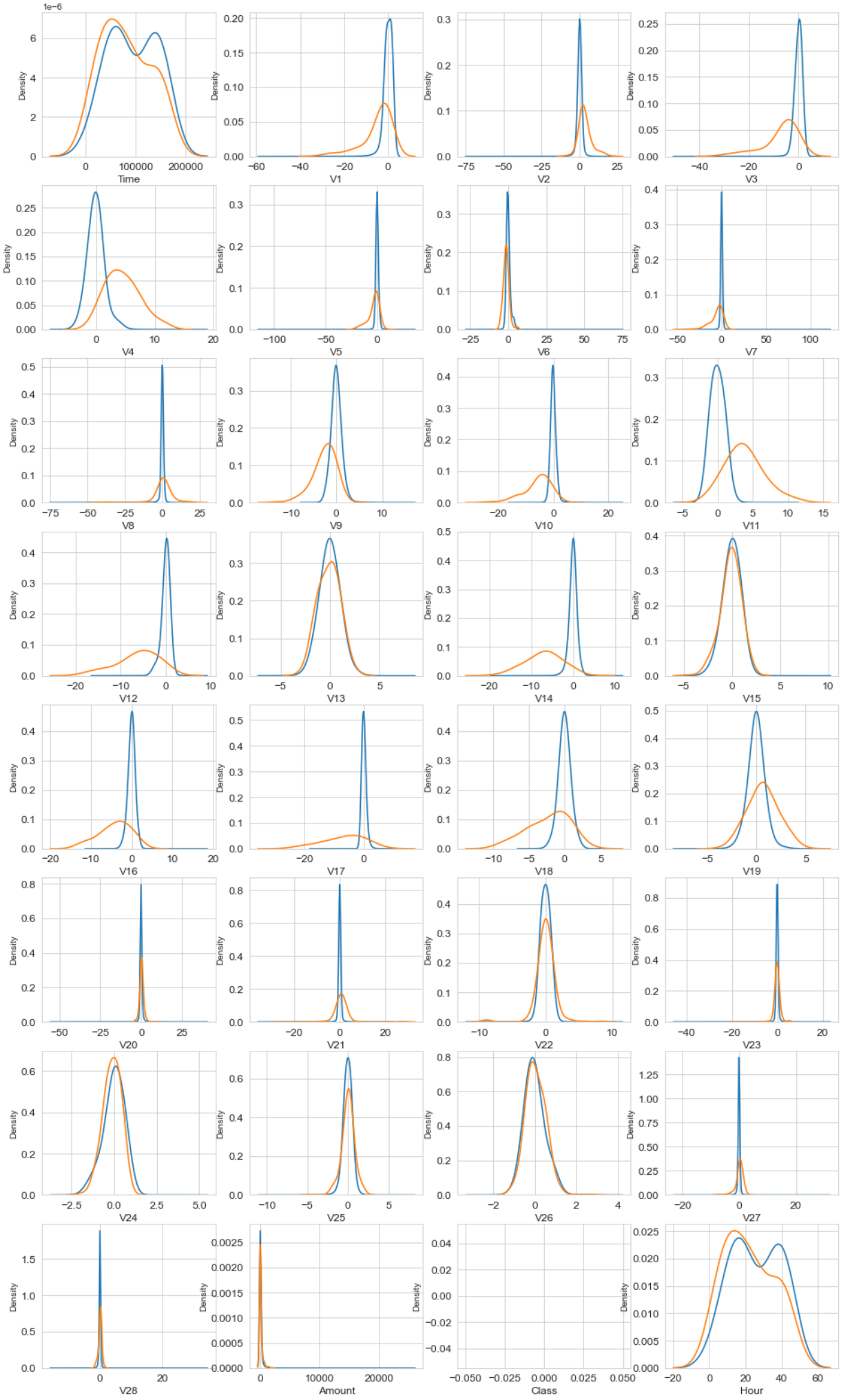
```
var = data_df.columns.values

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