Marc Riley DSC 680 Credit Card Fraud Detection

In [1]:
```
# import libraries used for EDA
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, precision_re
```

In [2]:
```
%config Completer.use_jedi = False
```

In [3]:
```
# Load Data
df = pd.read_csv('creditcard.csv')
```

In [4]:
```
#examine the data
df.head()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.3
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8

5 rows × 31 columns

In [5]:
```
df.describe
```

Out[5]:
```
<bound method NDFrame.describe of             Time         V1         V2
V3        V4        V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
...          ...        ...        ...       ...       ...       ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  ...       V21       V22  \
0       0.462388  0.239599  0.098698  0.363787  ... -0.018307  0.277838
1      -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775 -0.638672
2       1.800499  0.791461  0.247676 -1.514654  ...  0.247998  0.771679
3       1.247203  0.237609  0.377436 -1.387024  ... -0.108300  0.005274
4       0.095921  0.592941 -0.270533  0.817739  ... -0.009431  0.798278
...          ...       ...       ...       ...  ...       ...       ...
284802 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  ...  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057  0.643078

             V23       V24       V25       V26       V27       V28  Amount  \
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
...          ...       ...       ...       ...       ...       ...     ...
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

        Class
0           0
1           0
2           0
3           0
4           0
...       ...
284802      0
284803      0
284804      0
284805      0
284806      0

[284807 rows x 31 columns]>
```

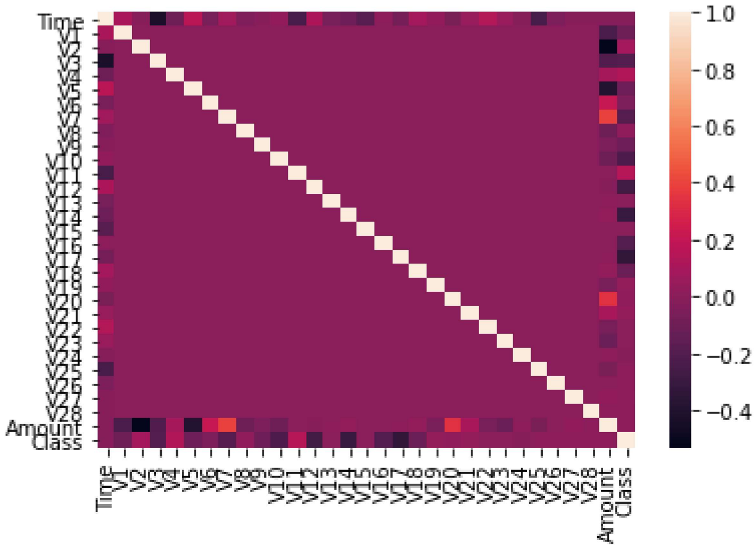
In [6]:

```
# find missing values
for c in df.columns:
    miss = df[c].isnull().sum()
    if miss >0:
        print("{} has {} missing values".format(c,miss))
    else:
        print("{} column has no missing values!".format(c))
```

```
Time column has no missing values!
V1 column has no missing values!
V2 column has no missing values!
V3 column has no missing values!
V4 column has no missing values!
V5 column has no missing values!
V6 column has no missing values!
V7 column has no missing values!
V8 column has no missing values!
V9 column has no missing values!
V10 column has no missing values!
V11 column has no missing values!
V12 column has no missing values!
V13 column has no missing values!
V14 column has no missing values!
V15 column has no missing values!
V16 column has no missing values!
V17 column has no missing values!
V18 column has no missing values!
V19 column has no missing values!
V20 column has no missing values!
V21 column has no missing values!
V22 column has no missing values!
V23 column has no missing values!
V24 column has no missing values!
V25 column has no missing values!
V26 column has no missing values!
V27 column has no missing values!
V28 column has no missing values!
Amount column has no missing values!
Class column has no missing values!
```

In [7]:
```
#Correlation test
corr = df.corr()
sns.heatmap(corr,
            xticklabels = corr.columns.values,
            yticklabels = corr.columns.values)
```

Out[7]: <AxesSubplot:>



In [8]:
```
#find how many actve fraud classes are in the data
Fraud = df[df["Class"] == 1]
Normal = df[df["Class"] == 0]
print(Fraud.shape)
print(Normal.shape)
#very unbalanced data
```

```
(492, 31)
(284315, 31)
```

In [8]:
```
#find duplicates
df.duplicated().sum()
```

Out[8]: 1081

In [9]:
```
#drop duplicates
df.drop_duplicates(inplace = True)
```

In [9]:
```
#decided to leave duplicates in due to the fact they seem to be tied to fraud
Fraud = df[df["Class"] == 1]
Normal = df[df["Class"] == 0]
print(Fraud.shape)
print(Normal.shape)
```

```
(492, 31)
(284315, 31)
```

In [16]:
```
#Create Plots
plt.figure(figsize = (20,12))
j=1;
for i in range(1,13):
    sns.distplot(Fraud["V"+str(i)],hist = False,color = 'red',label = "Fraud")
    sns.distplot(Normal["V"+str(i)],hist = False, color = 'gray',label = "Normal'
    plt.legend(fontsize = "medium",loc = "best")
    j = j+1
```

```
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `kdeplot` (an axes-level function for ker
nel density plots).
  warnings.warn(msg, FutureWarning)
C:\Users\Daffy\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `kdeplot` (an axes-level function for ker
nel density plots).
  warnings.warn(msg, FutureWarning)
C:\Users\Daffy\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level fu
nction with similar flexibility) or `kdeplot` (an axes-level function for ker
nel density plots).
  warnings.warn(msg, FutureWarning)
C:\Users\Daffy\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a futu
```

In [10]:
```
#split the target varibale and create x y
x = df.drop("Class",axis=1)
Y = df['Class']
x_train, x_test, y_train, y_test = train_test_split(x, Y, test_size=0.2, stratify
```

In [11]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [27]:
```
#gradient boosting

model = GradientBoostingClassifier(
    max_features='auto',
    min_samples_leaf=1,
    n_estimators=50,
    learning_rate=0.5,
    max_depth=5,
    random_state=1,
    )
model.fit(X_train,y_train)
```

Out[27]:
```
GradientBoostingClassifier(learning_rate=0.5, max_depth=5, max_features='auto',
                           n_estimators=10, random_state=1)
```

In [28]:
```
results = model.predict(X_test)
print(classification_report(results,y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56912
           1       0.46      0.90      0.61        50

    accuracy                           1.00     56962
   macro avg       0.73      0.95      0.80     56962
weighted avg       1.00      1.00      1.00     56962
```

In [29]:
```
#Over_Sampling
#Load smote
from imblearn.over_sampling import SMOTE

OverSample = SMOTE(random_state=142)

X_os, y_os = Oversample.fit_resample(X_train, y_train)
```

In [34]:
```
X_train_over, X_test_over, y_train_over, y_test_over = train_test_split(X_os, y_c
```

In [36]:
```
model = GradientBoostingClassifier(
    max_features='auto',
    min_samples_leaf=1,
    n_estimators=50,
    learning_rate=0.5,
    max_depth=5,
    random_state=3,
    )
mdoel.fit(X_train_over,y_train_oer)
```

Out[36]:
```
GradientBoostingClassifier(learning_rate=0.5, max_depth=5, max_features='auto',
                           n_estimators=10, random_state=3)
```

In [37]:
```
results = xgbus.predict(X_test_os)
print(classification_report(results,y_test_over))
```

```
              precision    recall  f1-score   support

           0       0.94      0.97      0.96     41606
           1       0.97      0.95      0.96     43690

    accuracy                           0.96     85296
   macro avg       0.96      0.96      0.96     85296
weighted avg       0.96      0.96      0.96     85296
```