Task 05: CREDIT CARD FRAUD DETECTION

#

In [54]:
```
import numpy as np
import pandas as pd
import warnings
import seaborn as sns
from sklearn.metrics import precision_score,recall_score,f1_score
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')
```

In [2]:
```
data=pd.read_csv('creditcard.csv')
```

In [3]:
```
data
```

Out[3]:

	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	0.363
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817
...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486

284807 rows × 31 columns

In [4]:
```
    data.shape
```

Out[4]:
```
(284807, 31)
```

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

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

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.2
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.2
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.1
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.0

5 rows × 31 columns

In [7]:
```
data.tail()
```

Out[7]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.9144
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.5848
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.4324
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.3920
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.4861

5 rows × 31 columns

In [8]:
```
list(data)
```

Out[8]:
```
['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']
```

In [11]:
```
data.iloc[:,1]
```

Out[11]:
```
0         -1.359807
1          1.191857
2         -1.358354
3         -0.966272
4         -1.158233
             ...
284802   -11.881118
284803    -0.732789
284804     1.919565
284805    -0.240440
284806    -0.533413
Name: V1, Length: 284807, dtype: float64
```

In [12]:
```
data.corr()
```

Out[12]:

	Time	V1	V2	V3	V4	V5	V6	
Time	1.000000	1.173963e-01	-1.059333e-02	-4.196182e-01	-1.052602e-01	1.730721e-01	-6.301647e-02	8.471
V1	0.117396	1.000000e+00	4.135835e-16	-1.227819e-15	-9.215150e-16	1.812612e-17	-6.506567e-16	-1.0
V2	-0.010593	4.135835e-16	1.000000e+00	3.243764e-16	-1.121065e-15	5.157519e-16	2.787346e-16	2.055
V3	-0.419618	-1.227819e-15	3.243764e-16	1.000000e+00	4.711293e-16	-6.539009e-17	1.627627e-15	4.895
V4	-0.105260	-9.215150e-16	-1.121065e-15	4.711293e-16	1.000000e+00	-1.719944e-15	-7.491959e-16	-4.1
V5	0.173072	1.812612e-17	5.157519e-16	-6.539009e-17	-1.719944e-15	1.000000e+00	2.408382e-16	2.715
V6	-0.063016	-6.506567e-16	2.787346e-16	1.627627e-15	-7.491959e-16	2.408382e-16	1.000000e+00	1.191
V7	0.084714	-1.005191e-15	2.055934e-16	4.895305e-16	-4.104503e-16	2.715541e-16	1.191668e-16	1.000
V8	-0.036949	-2.433822e-16	-5.377041e-17	-1.268779e-15	5.697192e-16	7.437229e-16	-1.104219e-16	3.344
V9	-0.008660	-1.513678e-16	1.978488e-17	5.568367e-16	6.923247e-16	7.391702e-16	4.131207e-16	1.122
V10	0.030617	7.388135e-17	-3.991394e-16	1.156587e-15	2.232685e-16	-5.202306e-16	5.932243e-17	-7.4
V11	-0.247689	2.125498e-16	1.975426e-16	1.576830e-15	3.459380e-16	7.203963e-16	1.980503e-15	1.425
V12	0.124348	2.053457e-16	-9.568710e-17	6.310231e-16	-5.625518e-16	7.412552e-16	2.375468e-16	-3.5
V13	-0.065902	-2.425603e-17	6.295388e-16	2.807652e-16	1.303306e-16	5.886991e-16	-1.211182e-16	1.266
V14	-0.098757	-5.020280e-16	-1.730566e-16	4.739859e-16	2.282280e-16	6.565143e-16	2.621312e-16	2.607
V15	-0.183453	3.547782e-16	-4.995814e-17	9.068793e-16	1.377649e-16	-8.720275e-16	-1.531188e-15	-1.6
V16	0.011903	7.212815e-17	1.177316e-17	8.299445e-16	-9.614528e-16	2.246261e-15	2.623672e-18	5.869
V17	-0.073297	-3.879840e-16	-2.685296e-16	7.614712e-16	-2.699612e-16	1.281914e-16	2.015618e-16	2.177
V18	0.090438	3.230206e-17	3.284605e-16	1.509897e-16	-5.103644e-16	5.308590e-16	1.223814e-16	7.604
V19	0.028975	1.502024e-16	-7.118719e-18	3.463522e-16	-3.980557e-16	-1.450421e-16	-1.865597e-16	-1.8
V20	-0.050866	4.654551e-16	2.506675e-16	-9.316409e-16	-1.857247e-16	-3.554057e-16	-1.858755e-16	9.379
V21	0.044736	-2.457409e-16	-8.480447e-17	5.706192e-17	-1.949553e-16	-3.920976e-16	5.833316e-17	-2.0
V22	0.144059	-4.290944e-16	1.526333e-16	-1.133902e-15	-6.276051e-17	1.253751e-16	-4.705235e-19	-8.8
V23	0.051142	6.168652e-16	1.634231e-16	-4.983035e-16	9.164206e-17	-8.428683e-18	1.046712e-16	-4.3
V24	-0.016182	-4.425156e-17	1.247925e-17	2.686834e-19	1.584638e-16	-1.149255e-15	-1.071589e-15	7.434

	Time	V1	V2	V3	V4	V5	V6	
V25	-0.233083	-9.605737e-16	-4.478846e-16	-1.104734e-15	6.070716e-16	4.808532e-16	4.562861e-16	-3.0
V26	-0.041407	-1.581290e-17	2.057310e-16	-1.238062e-16	-4.247268e-16	4.319541e-16	-1.357067e-16	-9.6
V27	-0.005135	1.198124e-16	-4.966953e-16	1.045747e-15	3.977061e-17	6.590482e-16	-4.452461e-16	-1.7
V28	-0.009413	2.083082e-15	-5.093836e-16	9.775546e-16	-2.761403e-18	-5.613951e-18	2.594754e-16	-2.7
Amount	-0.010596	-2.277087e-01	-5.314089e-01	-2.108805e-01	9.873167e-02	-3.863563e-01	2.159812e-01	3.973
Class	-0.012323	-1.013473e-01	9.128865e-02	-1.929608e-01	1.334475e-01	-9.497430e-02	-4.364316e-02	-1.8

31 rows × 31 columns

In [13]:
```
data.describe()
```

Out[13]:

	Time	V1	V2	V3	V4	V5	V6
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01

8 rows × 31 columns

In [14]:
```
pd.options.display.max_columns = None
```

In [15]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
data['Amount']=sc.fit_transform(pd.DataFrame(data['Amount']))
```

In [17]:
```
data.shape
```

Out[17]:
```
(284807, 31)
```

In [18]:
```
data = data.drop(['Time'],axis=1)
```

In [19]:
```
data.shape
```

Out[19]:
```
(284807, 30)
```

In [20]:
```
data = data.drop_duplicates()
```

In [21]:
```
data.shape
```

Out[21]:
```
(275663, 30)
```

In [22]:
```
data['Class'].value_counts()
```

Out[22]:
```
0    275190
1       473
Name: Class, dtype: int64
```

In [24]:
```
X = data.drop('Class',axis=1)
y = data['Class']
```

In [25]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,
                                                 random_state=42)
```

In [26]:
```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression()
log.fit(X_train,y_train)
```

Out[26]:
▼ LogisticRegression

LogisticRegression()

In [27]:
```
y_pred1 = log.predict(X_test)
```

In [28]:
```
from sklearn.metrics import accuracy_score
```

In [29]:
```
accuracy_score(y_test,y_pred1)
```

Out[29]:
```
0.9992200678359603
```

In [31]:
```
precision_score(y_test,y_pred1)
```

Out[31]:
```
0.8870967741935484
```

In [32]:
```
recall_score(y_test,y_pred1)
```

Out[32]:
```
0.6043956043956044
```

In [33]:
```
f1_score(y_test,y_pred1)
```

Out[33]:
```
0.718954248366013
```

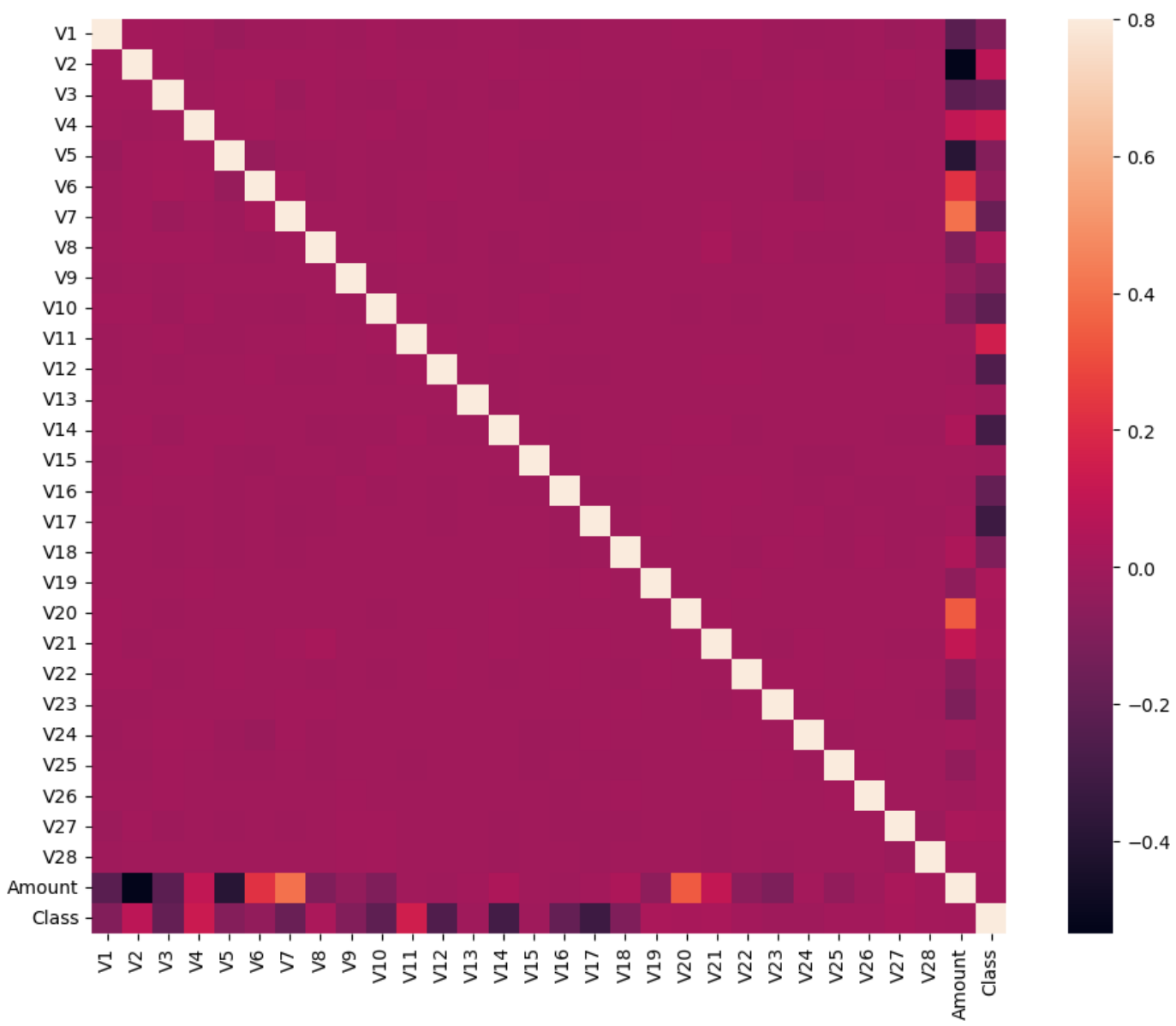
In [34]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train,y_train)
```

Out[34]:
▼ DecisionTreeClassifier

DecisionTreeClassifier()

In [ ]:
```
```

In [57]:
```
# Correlation matrix
corrmat = data.corr()
fig = plt.figure(figsize = (12, 9))
sns.heatmap(corrmat, vmax = .8, square = True)
plt.show()
```



In [47]:
```
# dividing the X and the Y from the dataset
X = data.drop(['Class'], axis = 1)
Y = data["Class"]
print(X.shape)
print(Y.shape)
# getting just the values for the sake of processing
# (its a numpy array with no columns)
xData = X.values
yData = Y.values
```

```
(275663, 29)
(275663,)
```

In [64]:
```
# Using Scikit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
xTrain, xTest, yTrain, yTest = train_test_split(
                xData, yData, test_size = 0.2, random_state = 42)
```

In [68]:
```
X = data.drop('Class',axis=1)
y = data['Class']
```

In [69]:
```
log = LogisticRegression()
log.fit(X_train,y_train)
```

Out[69]:
```
▼ LogisticRegression
LogisticRegression()
```

In [70]:
```
y_pred1 = log.predict(X_test)
```

In [71]:
```
accuracy_score(y_test,y_pred1)
```

Out[71]:
```
0.9992200678359603
```

In [76]:
```
precision_score(y_test,y_pred1)
```

Out[76]:
```
0.8870967741935484
```

In [77]:
```
recall_score(y_test,y_pred1)
```

Out[77]:
```
0.6043956043956044
```

In [78]:
```
f1_score(y_test,y_pred1)
```

Out[78]:
```
0.718954248366013
```

In [79]:
```
dt=DecisionTreeClassifier()
dt.fit(X_train,y_train)
```

Out[79]:
```
▼ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [80]:
```
y_pred2 = dt.predict(X_test)
```

In [81]:
```
accuracy_score(y_test,y_pred2)
```

Out[81]:
```
0.9989661364337148
```

In [82]:
```
precision_score(y_test,y_pred2)
```

Out[82]:
```
0.660377358490566
```

In [83]:
```
recall_score(y_test,y_pred2)
```

Out[83]:
```
0.7692307692307693
```

In [84]:
```
f1_score(y_test,y_pred2)
```

Out[84]:
```
0.7106598984771574
```

In [87]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
rf = RandomForestClassifier()
rf.fit(X_train,y_train)
```

In [90]:
```
y_pred3 = rf.predict(X_test)
```

In [91]:
```
accuracy_score(y_test,y_pred3)
```

Out[91]:
```
0.9994739992382058
```

In [92]:
```
precision_score(y_test,y_pred3)
```

Out[92]:
```
0.9078947368421053
```

In [93]:
```
recall_score(y_test,y_pred3)
```

Out[93]:
```
0.7582417582417582
```

In [94]:
```
f1_score(y_test,y_pred3)
```

Out[94]:
```
0.8263473053892216
```

In [95]:
```
final_data = pd.DataFrame({'Models':['LR','DT','RF'],
              "ACC":[accuracy_score(y_test,y_pred1)*100,
                     accuracy_score(y_test,y_pred2)*100,
                     accuracy_score(y_test,y_pred3)*100
                    ]})
```

In [96]:
```
final_data
```

Out[96]:

	Models	ACC
0	LR	99.922007
1	DT	99.896614
2	RF	99.947400

In [99]:
```
#----------------------------------------------------------THE END-------------------------
```

In [ ]:
```

```