Credit Card Fraud Detection Model

by Dajah Vincent

In [44]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
```

Reading the dataset

In [45]:
```
df = pd.read_csv("creditcard.csv")
```

In [46]:
```
df
```

Out[46]:

	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
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739
...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180

284807 rows × 31 columns

Exploratory Data Analysis

In [47]:
```
df.head(10)
```

Out[47]:

	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	...	-0.018
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246

10 rows × 31 columns

In [48]:
```
df.tail(10)
```

Out[48]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
284797	172782.0	-0.241923	0.712247	0.399806	-0.463406	0.244531	-1.343668	0.929369	-0.206210	0.106234
284798	172782.0	0.219529	0.881246	-0.635891	0.960928	-0.152971	-1.014307	0.427126	0.121340	-0.285670
284799	172783.0	-1.775135	-0.004235	1.189786	0.331096	1.196063	5.519980	-1.518185	2.080825	1.159498
284800	172784.0	2.039560	-0.175233	-1.196825	0.234580	-0.008713	-0.726571	0.017050	-0.118228	0.435402
284801	172785.0	0.120316	0.931005	-0.546012	-0.745097	1.130314	-0.235973	0.812722	0.115093	-0.204064
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180

10 rows × 31 columns

Exploratory Data Analysis

In [49]:
```
df.describe()
```

Out[49]:

	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 [50]:
```
#I want to print and see the heading of the columns in the dataframe

print(df.columns)
```

```
Index(['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'],
      dtype='object')
```

In [51]:
```
df.dtypes
```

Out[51]:
```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

In [52]:
```
# I want to check the data type per column

df.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 [53]:
```
# I want to check for missing or NaN values
print(df.isna().sum())
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [54]:
```
#print the number of duplicated values in the dataframe
print(df.duplicated().sum())
```

```
1081
```

In [55]:
```
#I drop the duplicated values from the dataframe

df.drop_duplicates()
```

Out[55]:

	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
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739
...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180

283726 rows × 31 columns

In [71]:
```
df['Class'].value_counts()
```

Out[71]:
```
Class
0    284315
1       492
Name: count, dtype: int64
```

In [84]:
```
legitTrans = df[df.Class == 0];

fraudTrans = df[df.Class == 1]
```

In [85]:
```
legitTrans.shape, fraudTrans.shape
```

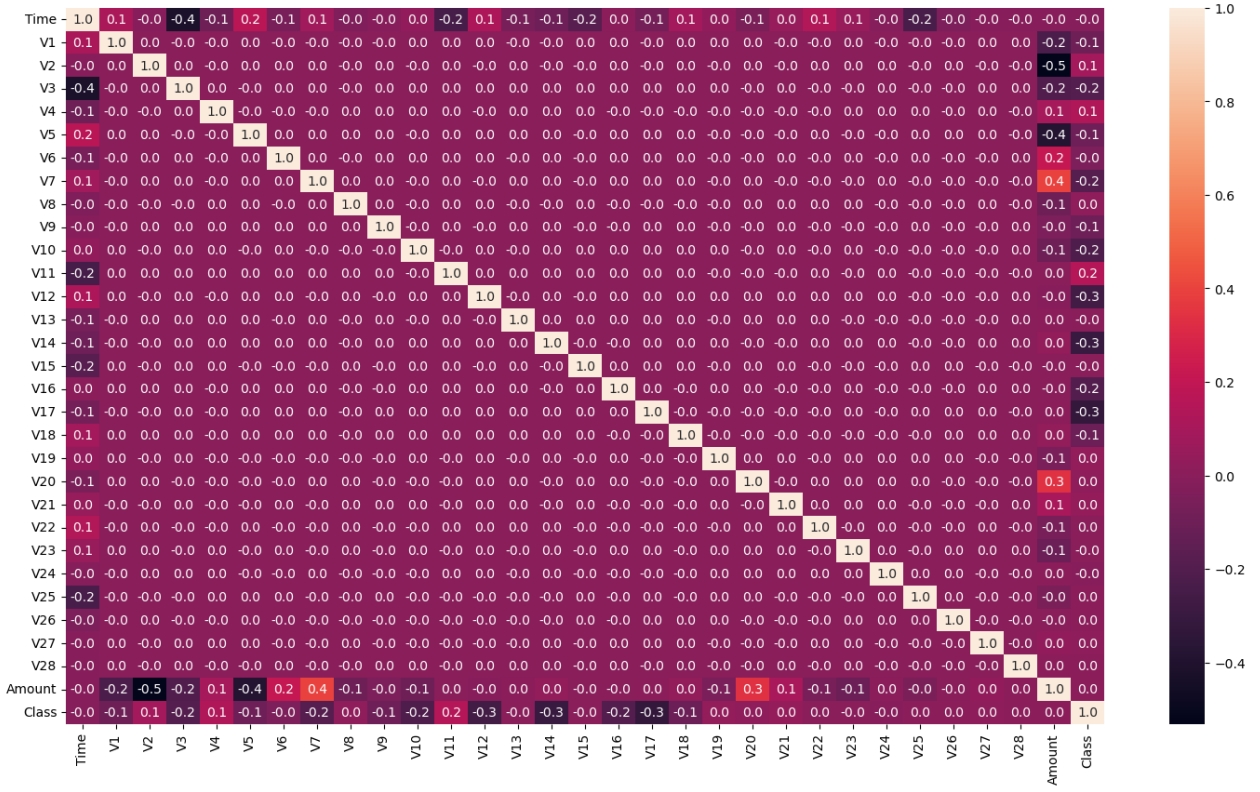
Out[85]:
```
((284315, 31), (492, 31))
```

In [108]:
```
print(round(legitTrans.Amount.describe(), 2))
print(round(fraudTrans.Amount.describe(), 2))
```

```
count    284315.00
mean         88.29
std         250.11
min           0.00
25%           5.65
50%          22.00
75%          77.05
max       25691.16
Name: Amount, dtype: float64
count     492.00
mean      122.21
std       256.68
min         0.00
25%         1.00
50%         9.25
75%       105.89
max      2125.87
Name: Amount, dtype: float64
```

Building Visuals

In [80]:
```
#Craete a heatmap with labels
plt.figure(figsize = (18, 10))
HM = sb.heatmap(df.corr(), annot = True, fmt = '.1f')
plt.show()
```



In [ ]:

In [81]:
```
#importing the model building libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [104]:
```
x = df.drop(columns = "Class", axis = 1)
y = df['Class']

print(x)
```

```
            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  ...       V20       V21  \
0       0.462388  0.239599  0.098698  0.363787  ...  0.251412 -0.018307
1      -0.082361 -0.078803  0.085102 -0.255425  ... -0.069083 -0.225775
2       1.800499  0.791461  0.247676 -1.514654  ...  0.524980  0.247998
3       1.247203  0.237609  0.377436 -1.387024  ... -0.208038 -0.108300
4       0.095921  0.592941 -0.270533  0.817739  ...  0.408542 -0.009431
...          ...       ...       ...       ...  ...       ...       ...
284802 -2.606837 -4.918215  7.305334  1.914428  ...  1.475829  0.213454
284803  1.058415  0.024330  0.294869  0.584800  ...  0.059616  0.214205
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.001396  0.232045
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.127434  0.265245
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.382948  0.261057

             V22       V23       V24       V25       V26       V27       V28  \
0       0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
1      -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2       0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
3       0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
4       0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153
...          ...       ...       ...       ...       ...       ...       ...
284802  0.111864  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731
284803  0.924384  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527
284804  0.578229 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561
284805  0.800049 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533
284806  0.643078  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649

        Amount
0       149.62
1         2.69
2       378.66
3       123.50
4        69.99
...        ...
284802    0.77
284803   24.79
284804   67.88
284805   10.00
284806  217.00

[284807 rows x 30 columns]
```

In [105]:
```
print(y)
```

```
0         0
1         0
2         0
3         0
4         0
         ..
284802    0
284803    0
284804    0
284805    0
284806    0
Name: Class, Length: 284807, dtype: int64
```

Split the dataset into training and testing dataset

In [120]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 5
```

In [121]:
```
x.shape, x_train.shape, x_test.shape
```

Out[121]:
```
((284807, 30), (213605, 30), (71202, 30))
```

In [122]:
```
mymodel = LogisticRegression()
```

In [123]:
```
#model
mymodel.fit(x_train, y_train)
```

```
C:\Users\DajahV01\AppData\Local\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.
py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/st
able/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://
scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[123]:

▾ LogisticRegression

LogisticRegression()

Model Evaluation

In [132]:
```
x_train_prediction = mymodel.predict(x_train)
training_data_accuracy = accuracy_score(x_train_prediction, y_train)
print("The model's accuracy is :", round(training_data_accuracy*100, 2),"%")
```

```
The model's accuracy is : 99.89 %
```

In [ ]: