Find Default (Prediction of Credit Card fraud)

Problem Statement:

A credit card is one of the most used financial products to make online purchases and payments. Though the Credit cards can be a convenient way to manage your finances, they can also be risky. Credit card fraud is the unauthorized use of someone else's credit card or credit card information to make purchases or withdraw cash.

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

We have to build a classification model to predict whether a transaction is fraudulent or not.

Data Set:

1	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
2	0	-1.3598	-0.0728	2.53635	1.37816	-0.3383	0.46239	0.2396	0.0987	0.36379	0.09079	-0.5516	-0.6178	-0.9914	-0.3112	1.46818	-0.4704
3	0	1.19186	0.26615	0.16648	0.44815	0.06002	-0.0824	-0.0788	0.0851	-0.2554	-0.167	1.61273	1.06524	0.4891	-0.1438	0.63556	0.46392
4	1	-1.3584	-1.3402	1.77321	0.37978	-0.5032	1.8005	0.79146	0.24768	-1.5147	0.20764	0.6245	0.06608	0.71729	-0.1659	2.34586	-2.8901
5	1	-0.9663	-0.1852	1.79299	-0.8633	-0.0103	1.2472	0.23761	0.37744	-1.387	-0.055	-0.2265	0.17823	0.50776	-0.2879	-0.6314	-1.0596
6	2	-1.1582	0.87774	1.54872	0.40303	-0.4072	0.09592	0.59294	-0.2705	0.81774	0.75307	-0.8228	0.5382	1.34585	-1.1197	0.17512	-0.4514
7	2	-0.426	0.96052	1.14111	-0.1683	0.42099	-0.0297	0.4762	0.26031	-0.5687	-0.3714	1.34126	0.35989	-0.3581	-0.1371	0.51762	0.40173
8	4	1.22966	0.141	0.04537	1.20261	0.19188	0.27271	-0.0052	0.08121	0.46496	-0.0993	-1.4169	-0.1538	-0.7511	0.16737	0.05014	-0.4436
9	7	-0.6443	1.41796	1.07438	-0.4922	0.94893	0.42812	1.12063	-3.8079	0.61537	1.24938	-0.6195	0.29147	1.75796	-1.3239	0.68613	-0.0761
10	7	-0.8943	0.28616	-0.1132	-0.2715	2.6696	3.72182	0.37015	0.85108	-0.392	-0.4104	-0.7051	-0.1105	-0.2863	0.07436	-0.3288	-0.2101
11	9	-0.3383	1.11959	1.04437	-0.2222	0.49936	-0.2468	0.65158	0.06954	-0.7367	-0.3668	1.01761	0.83639	1.00684	-0.4435	0.15022	0.73945
12	10	1.44904	-1.1763	0.91386	-1.3757	-1.9714	-0.6292	-1.4232	0.04846	-1.7204	1.62666	1.19964	-0.6714	-0.5139	-0.095	0.23093	0.03197
13	10	0.38498	0.61611	-0.8743	-0.094	2.92458	3.31703	0.47045	0.53825	-0.5589	0.30976	-0.2591	-0.3261	-0.09	0.36283	0.9289	-0.1295
14	10	1.25	-1.2216	0.38393	-1.2349	-1.4854	-0.7532	-0.6894	-0.2275	-2.094	1.32373	0.22767	-0.2427	1.20542	-0.3176	0.72567	-0.8156
15	11	1.06937	0.28772	0.82861	2.71252	-0.1784	0.33754	-0.0967	0.11598	-0.2211	0.46023	-0.7737	0.32339	-0.0111	-0.1785	-0.6556	-0.1999
16	12	-2.7919	-0.3278	1.64175	1.76747	-0.1366	0.8076	-0.4229	-1.9071	0.75571	1.15109	0.84456	0.79294	0.37045	-0.735	0.4068	-0.3031
17	12	-0.7524	0.34549	2.05732	-1.4686	-1.1584	-0.0778	-0.6086	0.0036	-0.4362	0.74773	-0.794	-0.7704	1.04763	-1.0666	1.10695	1.66011
18	12	1.10322	-0.0403	1.26733	1.28909	-0.736	0.28807	-0.5861	0.18938	0.78233	-0.268	-0.4503	0.93671	0.70838	-0.4686	0.35457	-0.2466
19	13	-0.4369	0.91897	0.92459	-0.7272	0.91568	-0.1279	0.70764	0.08796	-0.6653	-0.738	0.3241	0.27719	0.25262	-0.2919	-0.1845	1.14317
20	14	-5.4013	-5.4501	1.1863	1.73624	3.04911	-1.7634	-1.5597	0.16084	1.23309	0.34517	0.91723	0.97012	-0.2666	-0.4791	-0.5266	0.472
21	15	1.49294	-1.0293	0.45479	-1.438	-1.5554	-0.721	-1.0807	-0.0531	-1.9787	1.63808	1.07754	-0.632	-0.417	0.05201	-0.043	-0.1664
22	16	0.69488	-1.3618	1.02922	0.83416	-1.1912	1.30911	-0.8786	0.44529	-0.4462	0.56852	1.01915	1.29833	0.42048	-0.3727	-0.808	-2.0446
23	17	0.9625	0.32846	-0.1715	2.1092	1.12957	1.69604	0.10771	0.5215	-1.1913	0.7244	1.69033	0.40677	-0.9364	0.98374	0.71091	-0.6022
24	18	1.16662	0.50212	-0.0673	2.26157	0.4288	0.08947	0.24115	0.13808	-0.9892	0.92217	0.74479	-0.5314	-2.1053	1.12687	0.00308	0.42442
25	18	0.24749	0.27767	1.18547	-0.0926	-1.3144	-0.1501	-0.9464	-1.6179	1.54407	-0.8299	-0.5832	0.52493	-0.4534	0.08139	1.5552	-1.3969
26	22	-1.9465	-0.0449	-0.4056	-1.0131	2.94197	2.95505	-0.0631	0.85555	0.04997	0.57374	-0.0813	-0.2157	0.04416	0.0339	1.19072	0.57884
27	22	-2.0743	-0.1215	1.32202	0.41001	0.2952	-0.9595	0.54399	-0.1046	0.47566	0.14945	-0.8566	-0.1805	-0.6552	-0.2798	-0.2117	-0.3333
28	23	1.17328	0.3535	0.28391	1.13356	-0.1726	-0.9161	0.36902	-0.3273	-0.2467	-0.0461	-0.1434	0.97935	1.49229	0.10142	0.76148	-0.0146
29	23	1.32271	-0.174	0.43456	0.57604	-0.8368	-0.8311	-0.2649	-0.221	-1.0714	0.86856	-0.6415	-0.1113	0.36149	0.17195	0.78217	-1.3559
30	23	-0.4143	0.90544	1.72745	1.47347	0.00744	-0.2003	0.74023	-0.0292	-0.5934	-0.3462	-0.0121	0.7868	0.63595	-0.0863	0.0768	-1.4059
31	23	1.05939	-0.1753	1.26613	1.18611	-0.786	0.57844	-0.7671	0.40105	0.6995	-0.0647	1.04829	1.00562	-0.542	-0.0399	-0.2187	0.00448
32	24	1.23743	0.06104	0.38053	0.76156	-0.3598	-0.4941	0.00649	-0.1339	0.43881	-0.2074	-0.9292	0.52711	0.34868	-0.1525	-0.2184	-0.1916
33	25	1.11401	0.08555	0.4937	1.33576	-0.3002	-0.0108	-0.1188	0.18862	0.20569	0.08226	1.13356	0.6267	-1.4928	0.52079	-0.6746	-0.5291
34	26	-0.5299	0.87389	1.34725	0.14546	0.41421	0.10022	0.71121	0.17607	-0.2867	-0.4847	0.87249	0.85164	-0.5717	0.10097	-1.5198	-0.2844
35	26	-0.5299	0.87389	1.34725	0.14546	0.41421	0.10022	0.71121	0.17607	-0.2867	-0.4847	0.87249	0.85164	-0.5717	0.10097	-1.5198	-0.2844
36	26	-0.5354	0.86527	1.35108	0.14758	0.43368	0.08698	0.69304	0.17974	-0.2856	-0.4825	0.8718	0.85345	-0.5718	0.10225	-1.52	-0.2859
37	26	-0.5354	0.86527	1.35108	0.14758	0.43368	0.08698	0.69304	0.17974	-0.2856	-0.4825	0.8718	0.85345	-0.5718	0.10225	-1.52	-0.2859
38	27	-0.246	0.47327	1.69574	0.26241	-0.0109	-0.6108	0.79394	-0.2473	0.13888	-0.401	-0.8121	-0.1835	-0.6301	-0.2862	-0.337	-0.4281

1	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
2	0.20797	0.02579	0.40399	0.25141	-0.0183	0.27784	-0.1105	0.06693	0.12854	-0.1891	0.13356	-0.0211	149.62	0
3	-0.1148	-0.1834	-0.1458	-0.0691	-0.2258	-0.6387	0.10129	-0.3398	0.16717	0.12589	-0.009	0.01472	2.69	0
4	1.10997	-0.1214	-2.2619	0.52498	0.248	0.77168	0.90941	-0.6893	-0.3276	-0.1391	-0.0554	-0.0598	378.66	0
5	-0.6841	1.96578	-1.2326	-0.208	-0.1083	0.00527	-0.1903	-1.1756	0.64738	-0.2219	0.06272	0.06146	123.5	0
6	-0.237	-0.0382	0.80349	0.40854	-0.0094	0.79828	-0.1375	0.14127	-0.206	0.50229	0.21942	0.21515	69.99	0
7	-0.0581	0.06865	-0.0332	0.08497	-0.2083	-0.5598	-0.0264	-0.3714	-0.2328	0.10591	0.25384	0.08108	3.67	0
8	0.00282	-0.612	-0.0456	-0.2196	-0.1677	-0.2707	-0.1541	-0.7801	0.75014	-0.2572	0.03451	0.00517	4.99	0
9	-1.2221	-0.3582	0.3245	-0.1567	1.94347	-1.0155	0.0575	-0.6497	-0.4153	-0.0516	-1.2069	-1.0853	40.8	0
10	-0.4998	0.11876	0.57033	0.05274	-0.0734	-0.2681	-0.2042	1.01159	0.3732	-0.3842	0.01175	0.1424	93.2	0
11	-0.541	0.47668	0.45177	0.20371	-0.2469	-0.6338	-0.1208	-0.385	-0.0697	0.0942	0.24622	0.08308	3.68	0
12	0.25341	0.85434	-0.2214	-0.3872	-0.0093	0.31389	0.02774	0.50051	0.25137	-0.1295	0.04285	0.01625	7.8	0
13	-0.81	0.35999	0.70766	0.12599	0.04992	0.23842	0.00913	0.99671	-0.7673	-0.4922	0.04247	-0.0543	9.99	0
14	0.87394	-0.8478	-0.6832	-0.1028	-0.2318	-0.4833	0.08467	0.39283	0.16113	-0.355	0.02642	0.04242	121.5	0
15	0.12401	-0.9805	-0.9829	-0.1532	-0.0369	0.07441	-0.0714	0.10474	0.54826	0.10409	0.02149	0.02129	27.5	0
16	-0.1559	0.77827	2.22187	-1.5821	1.15166	0.22218	1.02059	0.02832	-0.2327	-0.2356	-0.1648	-0.0302	58.8	0
17	-0.2793	-0.42	0.43254	0.26345	0.49962	1.35365	-0.2566	-0.0651	-0.0391	-0.0871	-0.181	0.12939	15.99	0
18	-0.0092	-0.5959	-0.5757	-0.1139	-0.0246	0.196	0.0138	0.10376	0.3643	-0.3823	0.09281	0.03705	12.99	0
19	-0.9287	0.68047	0.02544	-0.047	-0.1948	-0.6726	-0.1569	-0.8884	-0.3424	-0.049	0.07969	0.13102	0.89	0
20	-0.7255	0.07508	-0.4069	-2.1968	-0.5036	0.98446	2.45859	0.04212	-0.4816	-0.6213	0.39205	0.94959	46.8	0
21	0.30424	0.55443	0.05423	-0.3879	-0.1776	-0.1751	0.04	0.29581	0.33293	-0.2204	0.0223	0.0076	5	0
22	0.51566	0.62585	-1.3004	-0.1383	-0.2956	-0.572	-0.0509	-0.3042	0.072	-0.4222	0.08655	0.0635	231.71	0
23	0.40248	-1.7372	-2.0276	-0.2693	0.144	0.40249	-0.0485	-1.3719	0.39081	0.19996	0.01637	-0.0146	34.09	0
24	-0.4545	-0.0989	-0.8166	-0.3072	0.0187	-0.062	-0.1039	-0.3704	0.6032	0.10856	-0.0405	-0.0114	2.28	0
25	0.78313	0.43662	2.17781	-0.231	1.65018	0.20045	-0.1854	0.42307	0.82059	-0.2276	0.33663	0.25048	22.75	0
26	-0.9757	0.04406	0.4886	-0.2167	-0.5795	-0.7992	0.8703	0.98342	0.3212	0.14965	0.70752	0.0146	0.89	0
27	0.01075	-0.4885	0.50575	-0.3867	-0.4036	-0.2274	0.74243	0.39853	0.24921	0.2744	0.35997	0.24323	26.43	0
28	-0.5116	-0.3251	-0.3909	0.02788	0.067	0.22781	-0.1505	0.43505	0.72482	-0.3371	0.01637	0.03004	41.88	0
29	-0.2169	1.27177	-1.2406	-0.523	-0.2844	-0.3234	-0.0377	0.34715	0.55964	-0.2802	0.04234	0.02882	16	0
30	0.77559	-0.9429	0.54397	0.09731	0.07724	0.45733	-0.0385	0.64252	-0.1839	-0.2775	0.18269	0.15266	33	0
31	-0.1936	0.04239	-0.2778	-0.178	0.01368	0.21373	0.01446	0.00295	0.29464	-0.3951	0.08146	0.02422	12.99	0
32	-0.1166	-0.6338	0.34842	-0.0664	-0.2457	-0.5309	-0.0443	0.07917	0.50914	0.28886	-0.0227	0.01184	17.28	0
33	0.15826	-0.3988	-0.1457	-0.2738	-0.0532	-0.0048	-0.0315	0.19805	0.56501	-0.3377	0.02906	0.00445	4.45	0
34	-0.3105	-0.4042	-0.8234	-0.2903	0.04695	0.2081	-0.1855	0.00103	0.09882	-0.5529	-0.0733	0.02331	6.14	0
35	-0.3105	-0.4042	-0.8234	-0.2903	0.04695	0.2081	-0.1855	0.00103	0.09882	-0.5529	-0.0733	0.02331	6.14	0
36	-0.3096	-0.4039	-0.8237	-0.2833	0.04953	0.20654	-0.1871	0.00075	0.09812	-0.5535	-0.0783	0.02543	1.77	0
37	-0.3096	-0.4039	-0.8237	-0.2833	0.04953	0.20654	-0.1871	0.00075	0.09812	-0.5535	-0.0783	0.02543	1.77	0
38	-0.0306	-0.5046	0.35518	0.04572	-0.1946	-0.3351	-0.0782	0.39278	-0.0312	0.19804	-0.1753	-0.2036	30.49	0

The Data Set contains 284807 rows and 31 columns where to safeguard user identity and secure their confidential data, the dataset provider has used Principal Component Analysis to transform the original numerical features, compressing them into 28 principal components except columns Time, Amount and Class.

The Class column is our target column which is going to be used for finding fraudulent(1) and non-fraudulent(0) transactions.

Data Cleaning:

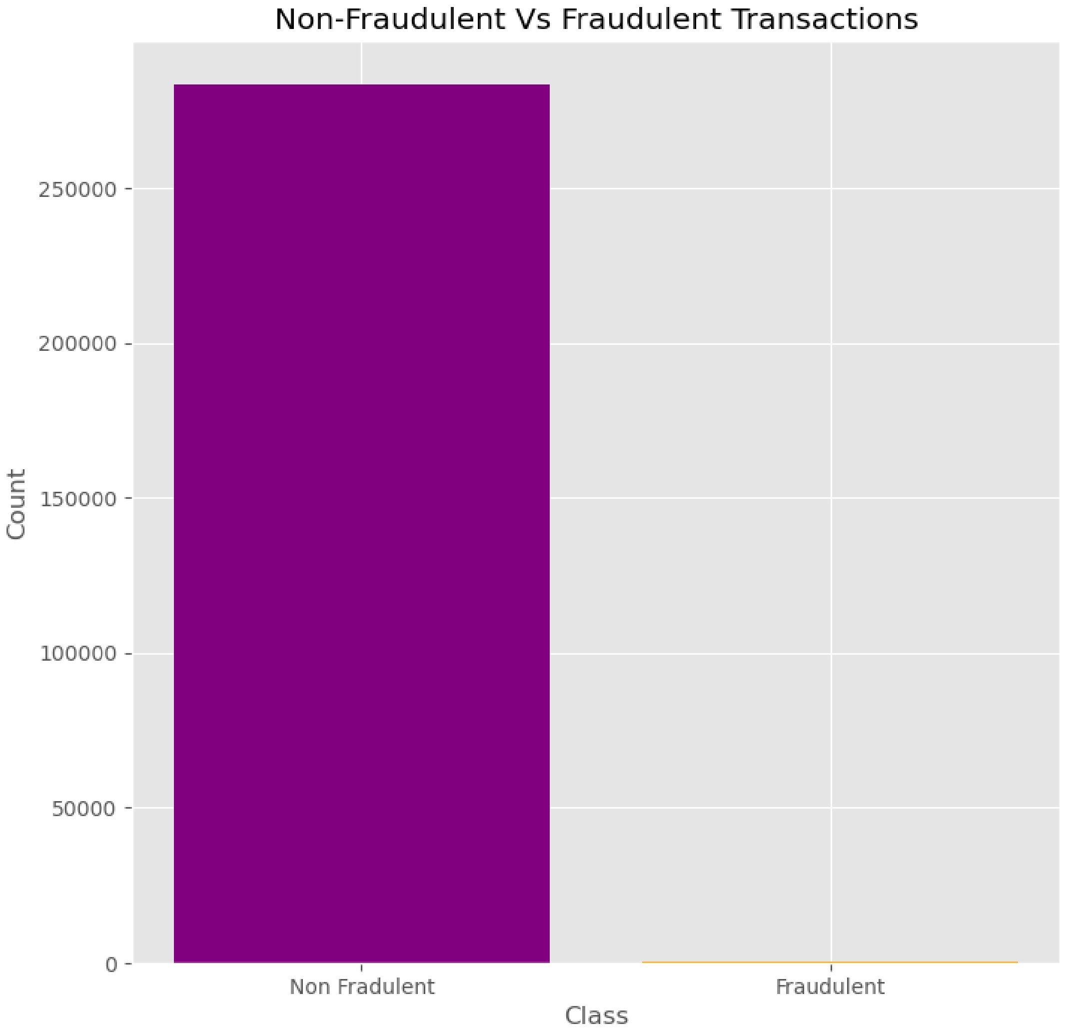
While trying to clean the data from null and duplicate values we found out that there are no null values in data set but it did contain some duplicates.

```
[9]: # check for null values
     df.isna().sum()

[9]: 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
```

```
[11]: # check for duplicate values
      df.duplicated().sum()

[11]: 1081

[12]: df.drop_duplicates(inplace=True)

[13]: # check if duplicates data removed or not
      df.duplicated().sum()

[13]: 0
```

Exploratory Data Analysis:

We tried to find out the number of Fraudulent and non-fraudulent transactions in the data set and plotted the graph showing their relationship.



After getting the the number of Fraudulent and non-fraudulent transactions we tried to understand more about our data set particularly Time and Amount column. We also plotted the graph to know at which time most transactions took place and if outlier data was present in the Amount column.

[21]:

```python
# summary of Time column
df['Time'].describe()
```

[21]:

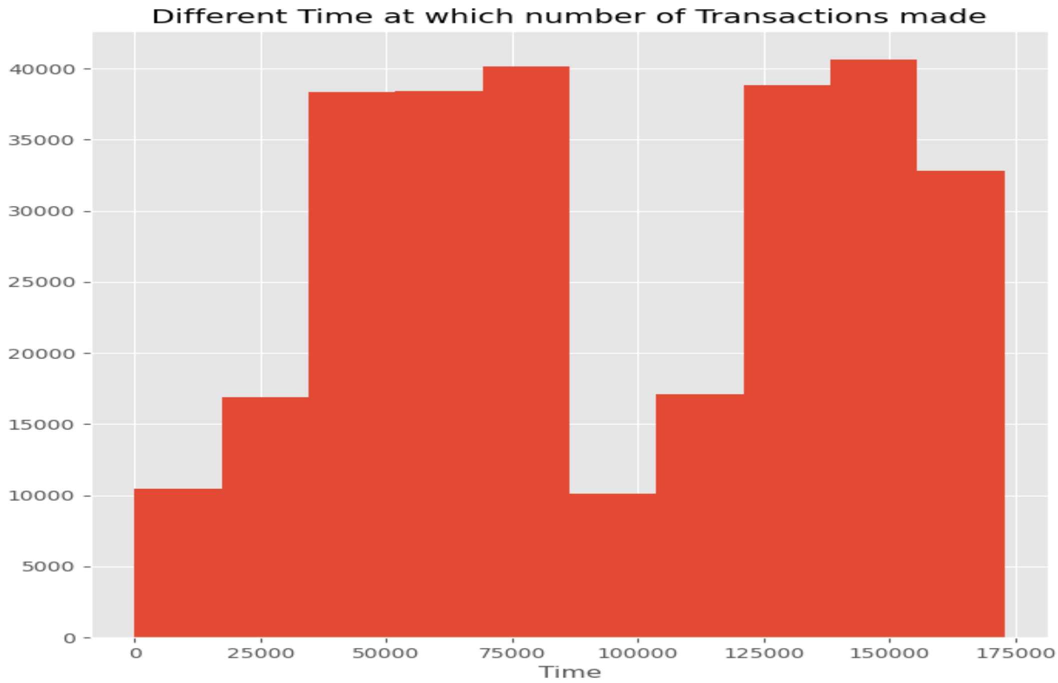
```
count    283726.000000
mean      94811.077600
std       47481.047891
min           0.000000
25%       54204.750000
50%       84692.500000
75%      139298.000000
max      172792.000000
Name: Time, dtype: float64
```

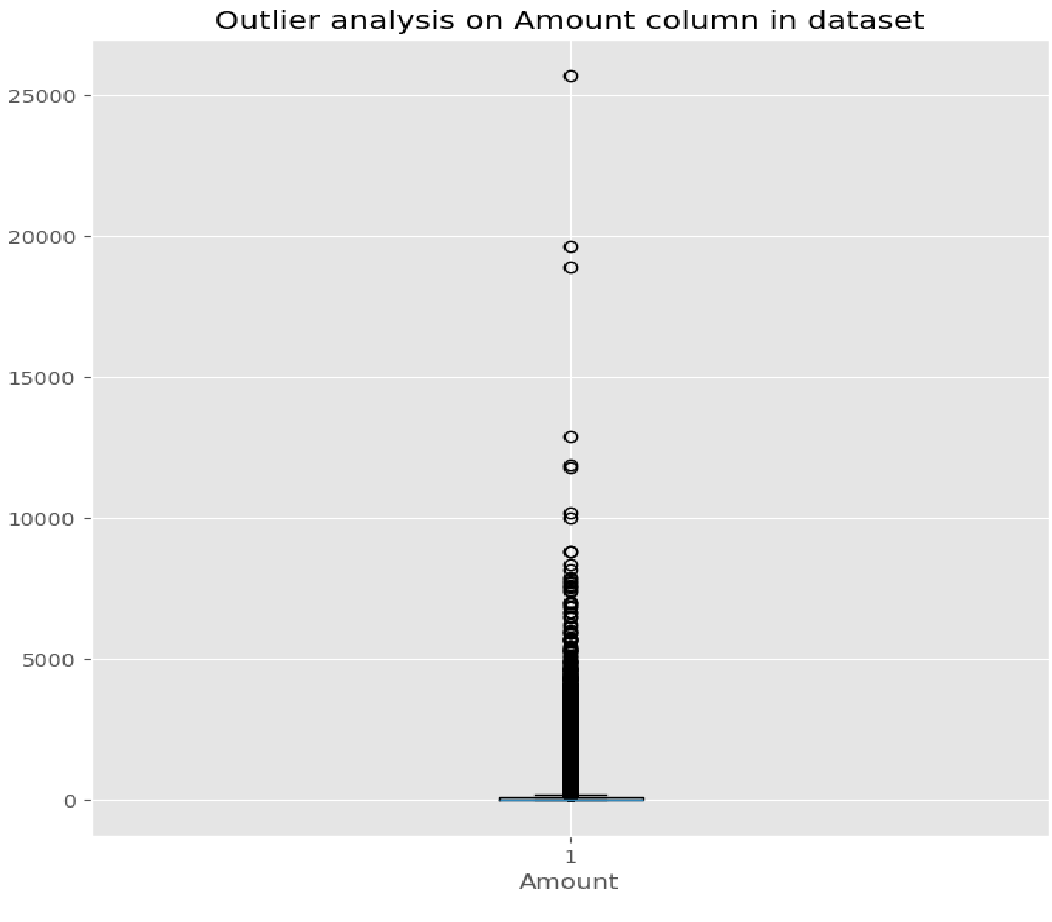
[23]:

```python
# summary of Amount column
df['Amount'].describe()
```

[23]:

```
count    283726.000000
mean         88.472687
std         250.399437
min           0.000000
25%           5.600000
50%          22.000000
75%          77.510000
max       25691.160000
Name: Amount, dtype: float64
```


Different Time at which number of Transactions made
40000
35000
30000
25000
20000
15000
10000
5000
0
0
25000
50000
75000
100000
125000
150000
175000
Time


Outlier analysis on Amount column in dataset
25000
20000
15000
10000
5000
0
1
Amount

We found out the percantage fraudulent transactions in the data set i.e 0.17 %.

```
Number of Non-Fraudulent Transactions:  283253
Number of Fraudulent Transactions:  473
Percentage of Fraudulent Transactions:  0.17
```

Feature Engineering:

Since, we didn't need the Time and Amount column we dropped these 2 columns from our data set and added new column called scaled values.

And for training we copied the data into 2 different variables X and Y.

where X contained PCA components + Scaled amount and Y contained Class.

Model Training:

We splitted the credit card data into 70-30 using train_test_split() where our parameters were:-

- **X : Feature Matrix**
- **Y : Target Variable**

We set the test_size to 0.3, meaning 30% of the data is allocated for the test set.

```
Shape of the training dataset train_X:  (198608, 29)
Shape of the testing dataset test_X:  (85118, 29)
```

Model Selection:

We selected the Decision Tree and Random Forest algorithms for our model.

- **A Decision Tree is a supervised learning algorithm that splits the dataset into subsets based on feature values, creating a tree-like structure of decisions.**

- **Random Forest is an ensemble learning method that combines multiple decision trees to improve accuracy and prevent overfitting.**

```
[50]: # Decision Tree
decision_tree = DecisionTreeClassifier()

# Random Forest
random_forest = RandomForestClassifier(n_estimators=100)
```

We found the score for each of the algorithm for finding which is better for our data.

```
Decision Tree:   99.92
Random Forest:   99.95
```

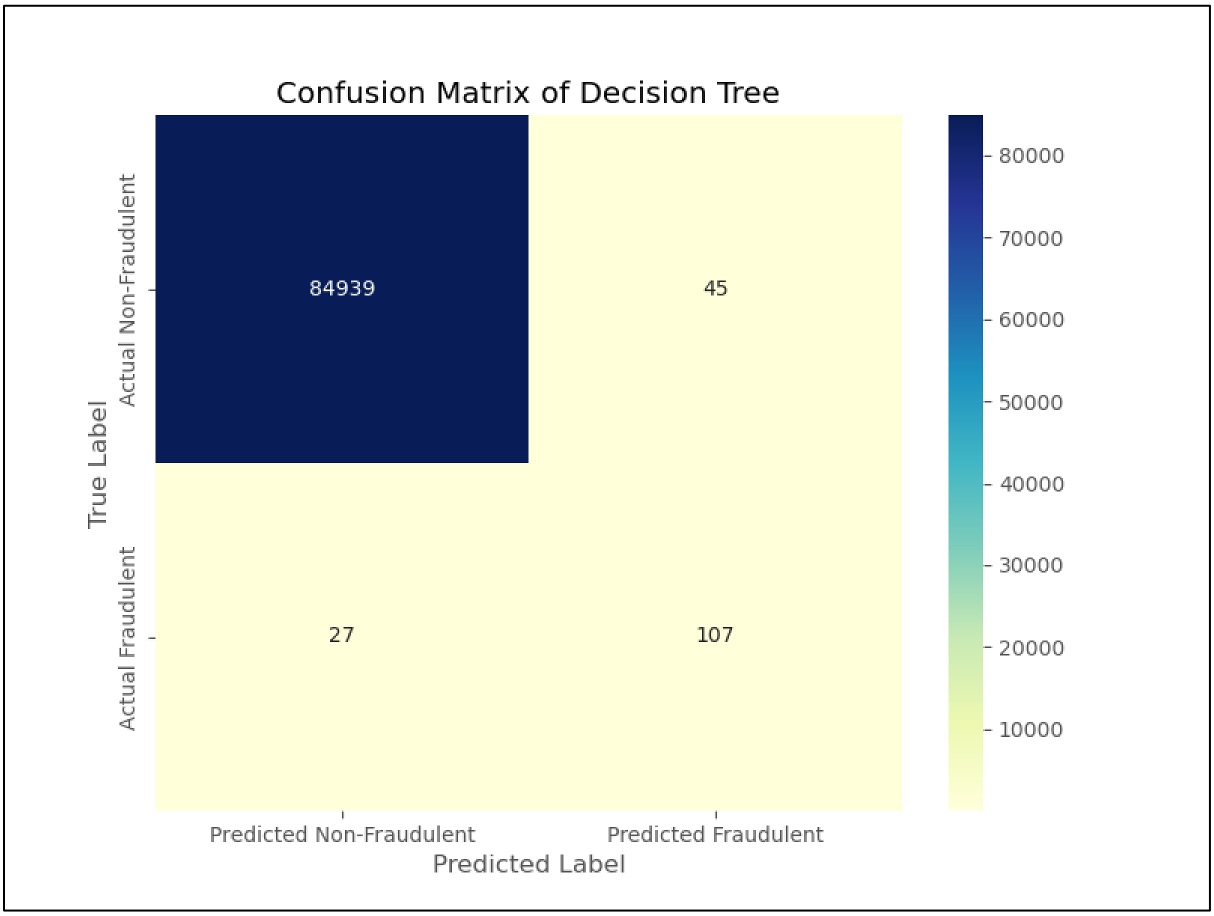
Model Validation:

We tried to validate and test our models based on various criterias such as accuracy_score , precision_score , recall_score , F1_score and confusion matix.

We also plotted the heatmaps for each of the model's confusion matix.

Evaluation of Decision Tree Model:	Evaluation of Random Forest Model:
Accuracy: 0.9992	Accuracy: 0.9995
Precision: 0.7039	Precision: 0.9519
recall_score: 0.7985	recall_score: 0.7388
F1-Score: 0.7483	F1-Score: 0.8319

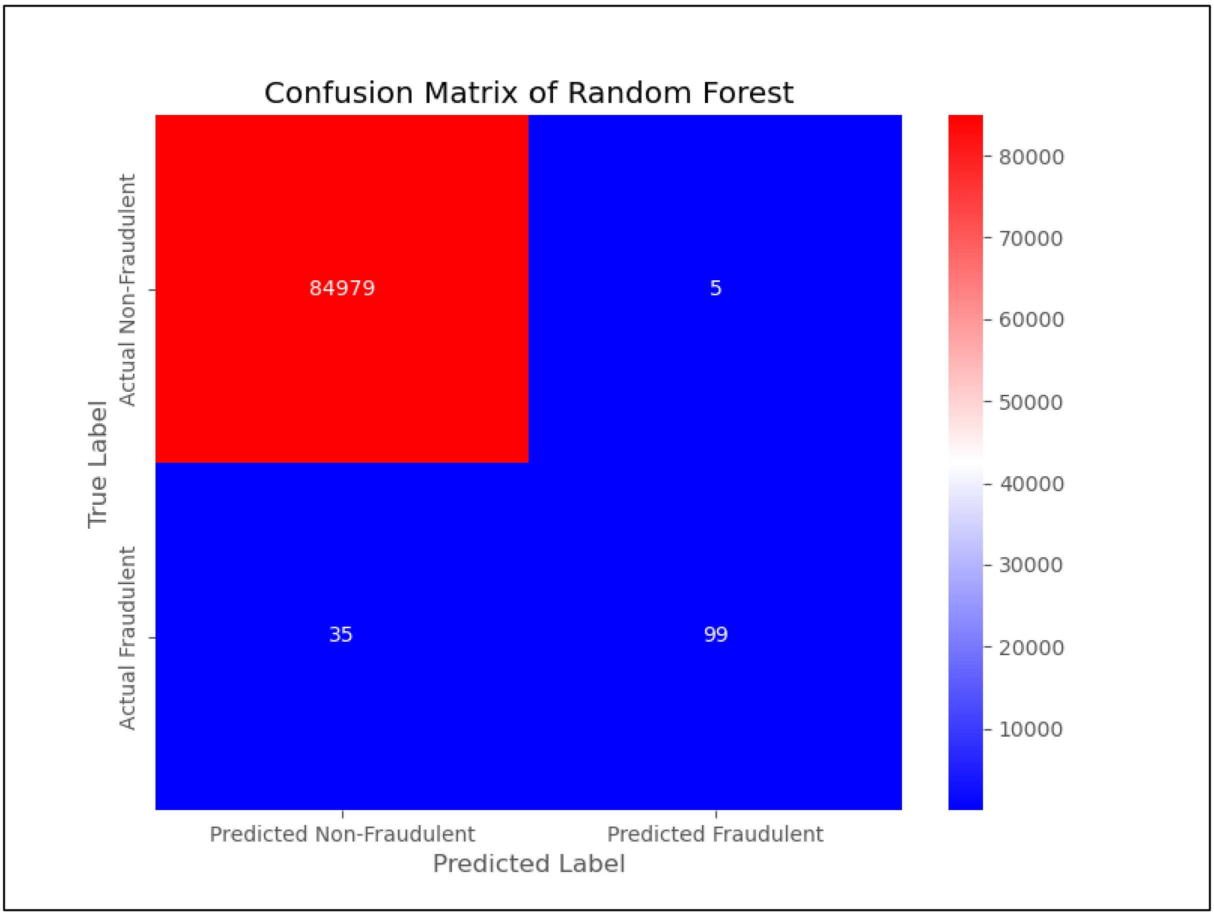
Decision Tree:



From the confusion matrix heatmap of Decision Tree we get to know:

1. **The model correctly identified 107 fraudulent transactions.**
2. **The model incorrectly identified 27 transactions as non-fraudulent.**
3. **The model correctly identified 84939 non-fraudulent transactions.**
4. **The model incorrectly identified 45 non-fraudulent transactions as fraudulent.**

Random Forest:



From the confusion matrix heatmap of Decision Tree we get to know:

1. **The model correctly identified 99 fraudulent transactions.**
2. **The model incorrectly identified 35 transactions as non-fraudulent.**
3. **The model correctly identified 84979 non-fraudulent transactions.**
4. **The model incorrectly identified 5 non-fraudulent transactions as fraudulent.**

Dealing with Imbalanced data:

This data set is highly imbalanced. The data should be balanced using the appropriate methods before moving onto model building.

The class imbalance problem can be solved by various techniques. We will use one such technique called Oversampling .
The method through which oversampling is possible is called SMOT (Synthetic Minority Oversampling Technique or SMOTE).

```
print("Resampled shape of X: ",X_resampled.shape)
print("Resampled shape of Y: ",Y_resampled.shape)
```

```
Resampled shape of X:  (566506, 29)
Resampled shape of Y:  (566506,)
```

We used Random forest algorithm on the resampled data as we found out that random forest is better than the Decision tree.
The score of Random forest on resampled data was:

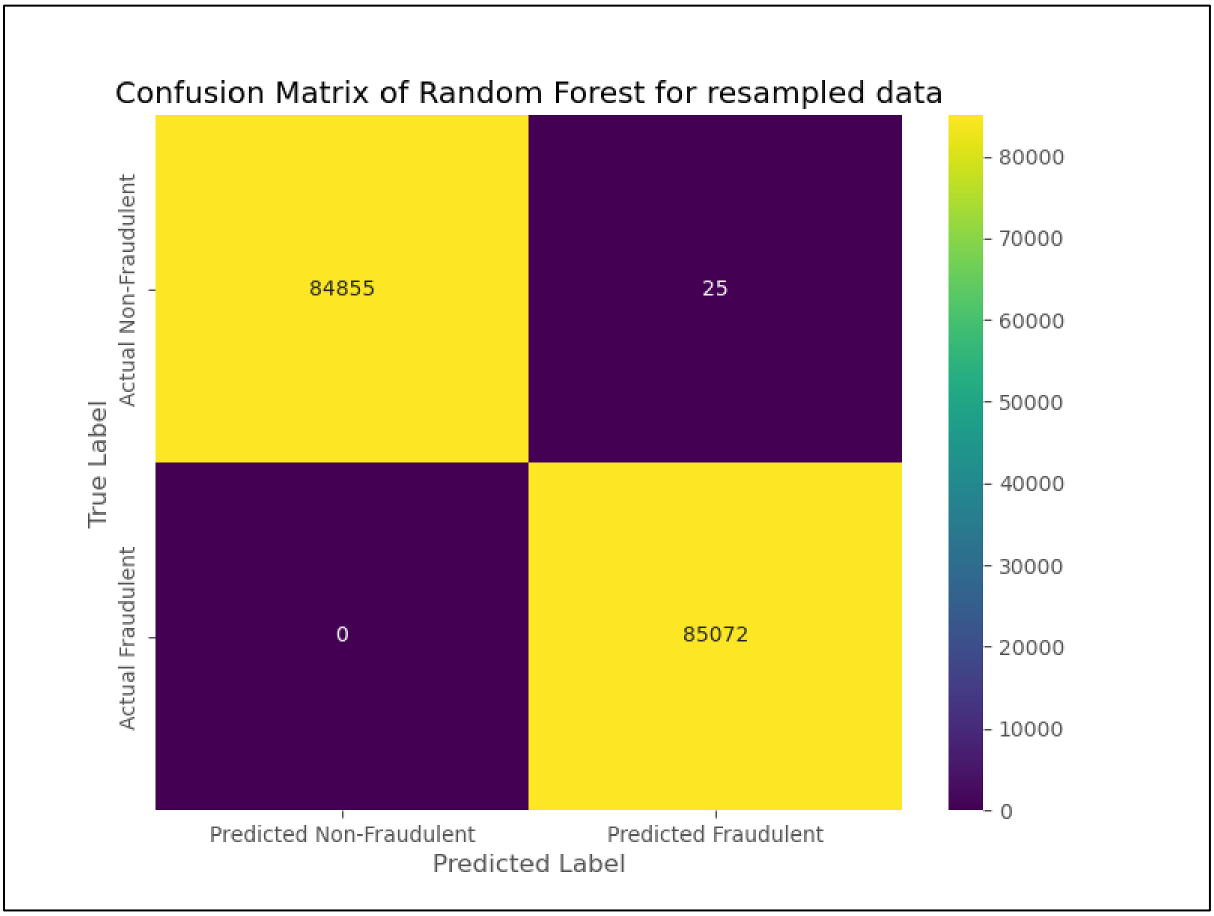
```
predictions_resampled = rf_resampled.predict(test_X)
random_forest_score_resampled = rf_resampled.score(test_X, test_Y) * 100
print(random_forest_score_resampled)
```

```
99.98528996422519
```

Then we validated the resampled Random forest model based on various criterias such as accuracy_score , precision_score , recall_score , F1_score and confusion_matrix.

We also plotted the heatmaps for each of the model's confusion matix.

```
Evaluation of Random Forest Model for resampled data:
Accuracy: 0.999853
Precision: 0.999706
recall_score: 1.0
F1-Score: 0.999853
```



From the confusion matrix heatmap of Decision Tree we get to know:

1. **The model correctly identified 85072 fraudulent transactions.**
2. **The model incorrectly identified 0 transactions as non-fraudulent.**
3. **The model correctly identified 84885 non-fraudulent transactions.**
4. **The model incorrectly identified 25 non-fraudulent transactions as fraudulent.**

Model Deployment:

To prepare for model deployment as part of future plans, we will use the pickle library to save both the dataframe and the model.

[198]:

```
import pickle
pickle.dump(df,open('df.pkl','wb'))
pickle.dump(rf_resampled,open('rf_resampled.pkl','wb'))
```