Minor Project - CreditCard

– Nakkina Vinay (B21AI023)
– Geda Durga Vara Praveen (B21CS031)
– Sakam Sai Santhosh (B21AI035)

We should import the dataset to the colab file first.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0.77	0
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	24.79	0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	67.88	0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	10.00	0
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	217.00	0

284807 rows × 31 columns

Next we will use info to find that there is any null values or anything unuseful values.

```
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)
```

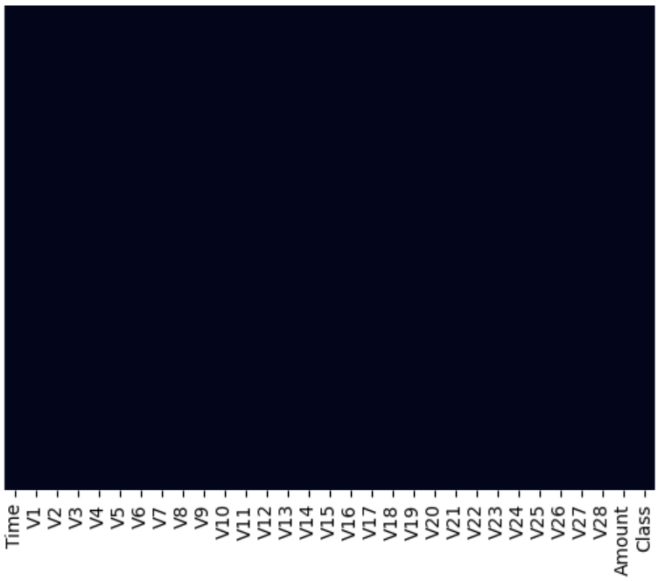
Then we use describe to find mean ,count, std ,min and percentile of features.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	...	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	284807.000000	284807.000000
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-15	...	1.654067e-16	-3.568593e-16	2.578648e-16	4.473266e-15	5.340915e-16	1.683437e-15	-3.660091e-16	-1.227390e-16	88.349619	0.001727
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	...	7.345240e-01	7.257016e-01	6.244603e-01	6.056471e-01	5.212781e-01	4.822270e-01	4.036325e-01	3.300833e-01	250.120109	0.041527
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	...	-3.483038e+01	-1.093314e+01	-4.480774e+01	-2.836627e+00	-1.029540e+01	-2.604551e+00	-2.256568e+01	-1.543008e+01	0.000000	0.000000
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	...	-2.283949e-01	-5.423504e-01	-1.618463e-01	-3.545861e-01	-3.171451e-01	-3.269839e-01	-7.083953e-02	-5.295979e-02	5.600000	0.000000
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	...	-2.945017e-02	6.781943e-03	-1.119293e-02	4.097606e-02	1.659350e-02	-5.213911e-02	1.342146e-03	1.124383e-02	22.000000	0.000000
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	...	1.863772e-01	5.285536e-01	1.476421e-01	4.395266e-01	3.507156e-01	2.409522e-01	9.104512e-02	7.827995e-02	77.165000	0.000000
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	...	2.720284e+01	1.050309e+01	2.252841e+01	4.584549e+00	7.519589e+00	3.517346e+00	3.161220e+01	3.384781e+01	25691.160000	1.000000

8 rows × 31 columns

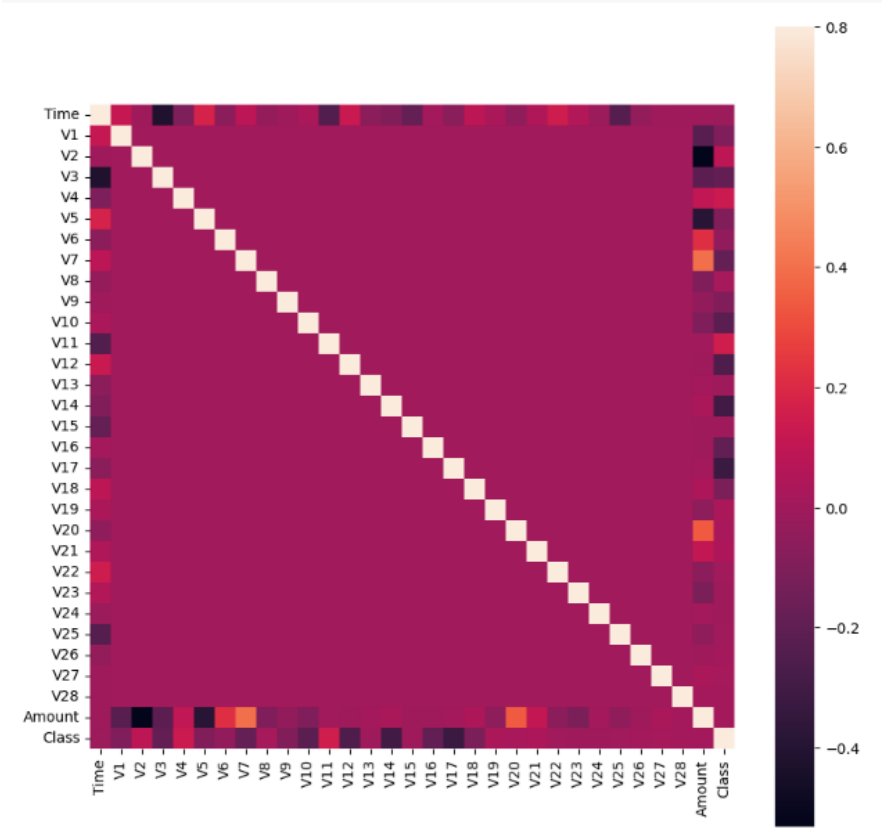
Next we use heat to find the null value using a graph.

```
'Cream Lines in the graph indicates the empty values'
```



Here cream lines indicate empty values. But here there are no cream lines so there are no empty values.

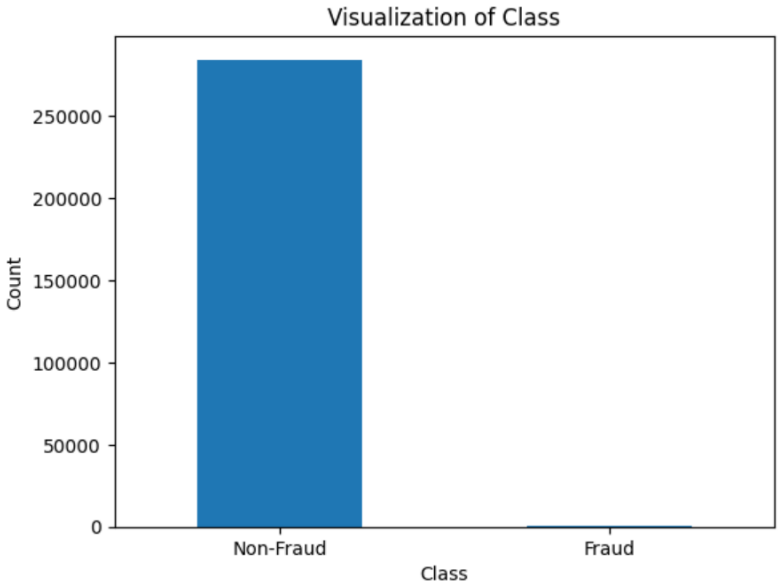
Next we plot the correlation matrix using a heat map.



Next we check fraud and non-fraud transitions by changing the values in respective columns with 0 non-fraud 1 fraud and percentage of fraud by non fraud.

```
Number of Genuine transactions:  284315
Number of Fraud transactions:  492
Percentage of Fraud transactions: 0.1727
```

The database is unbalanced.



Now we find the mean ,count, std ,min and percentile of features of non-fraud and fraud.

```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

non-Fraud

```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

Fraud

Finding the mean of over different classes.

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	...	-0.000644	-0.001235	-0.000024	0.000070	0.000182	-0.000072	-0.000089	-0.000295	-0.000131	88.291022
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014049	-0.040308	-0.105130	0.041449	0.051648	0.170575	0.075667	122.211321

StandardScalar is used to transform the numerical features of a dataset so that they have zero mean and unit variance.
And we drop the ‘Amount’ and ‘Time’ columns because there is no need for that.
We define X and y values.
How we balance the unbalanced data using RandomOverSampler function and we fit X-res and y_res values to balance.
Shape of X = `(284807, 29)` Shape of X_res = `(568630, 29)`
Balancing can lead to an increase or decrease in the number of rows in X, depending on whether samples are added or removed from the dataset to achieve a balanced class distribution. Here the values increased after balancing.

SPLITTING DATA:

Here we split the dataset into a 80:20 ratio, which means Training set percentage is 80% and Testing set percentage is 20%.

RandomForest Classifier :

Random Forest is a popular ensemble learning algorithm used for classification and regression problems. It builds multiple decision trees and combines their predictions to make a final prediction. This randomness helps reduce overfitting, and the algorithm is widely used in machine learning due to its accuracy and robustness.

The RandomForestClassifier is created and trained using the fit() method, then used to predict the class labels for X_test using the predict() method. Four performance metrics, accuracy, precision, recall, and F1 score, are calculated using the corresponding functions, and the results are printed using the print() function.

```
Accuracy for Randomforest Classifier: 0.9999472416158135
Precision for Randomforest Classifier: 0.9998947035906076
Recall for Randomforest Classifier: 1.0
F1 Score for Randomforest Classifier: 0.9999473490233244
```

Confusion Matrix for RandomForestClassifier:

```
Confusion Matrix for Random Forest Classifier:
 [[56744     6]
 [    0 56976]]
```

Decision Tree classifier:

A decision tree is a popular machine learning algorithm used for both classification and regression tasks. It uses a tree-like model of decisions and their possible consequences, with each internal node representing a feature and each leaf node representing a class label or a numerical value. The algorithm is widely used due to its simplicity, interpretability, and ability to handle non-linear relationships.

The DecisionTreeClassifier module is imported from the sklearn.tree module for building the classifier. An instance of the classifier is created and trained using the fit() method on the training data, X_train and y_train. The classifier is then used to predict the class labels for the test data X_test using the predict() method. Four performance metrics, accuracy, precision, recall, and F1 score, are calculated and the results are printed using the print() function. These metrics are used to evaluate the performance of a classifier, with higher values indicating better performance.

```
Accuracy for DecisionTree Classifier: 0.9386595853191003
Precision for DecisionTree Classifier: 0.9672897196261683
Recall for DecisionTree Classifier: 0.9082771693344566
F1 Score for DecisionTree Classifier: 0.936855063543213

Confusion Matrix for Decision Tree Classifier:
 [[56719    31]
 [    0 56976]]
```

SVM (Support Vector Machine) :

SVM (Support Vector Machine) classifier from scikit-learn's SVM module to train a model on the given training data X_train and y_train. It then predicts the class labels for the test data X_test using the predict() method and calculates different performance metrics, including accuracy, precision, recall, and F1 score, using scikit-learn's metrics module. Finally, it prints the values of these metrics for the SVM classifier.

```
Accuracy for SVM Classifier: 0.97407804723634
Precision for SVM Classifier: 0.987987282777557
Recall for SVM Classifier: 0.9599304970513901
F1 Score for SVM Classifier: 0.9737568323036659
```

```
Confusion Matrix for SVM Classifier:
 [[56085   665]
 [ 2283 54693]]
```

LDA Classifier:

LDA is a classification algorithm that seeks a linear combination of features to represent data in a lower-dimensional space while maximizing class separability. In scikit-learn, it can be used to fit a model to training data, predict labels for new data, and evaluate performance with metrics.

LinearDiscriminantAnalysis class from the sklearn.discriminant_analysis module to create an LDA classifier. The model is fitted to the training data using the fit() method, and then used to predict the labels of the test data using the predict() method. Performance of the LDA classifier is evaluated using the accuracy_score(), precision_score(), recall_score(), and f1_score() functions from the sklearn.metrics module.

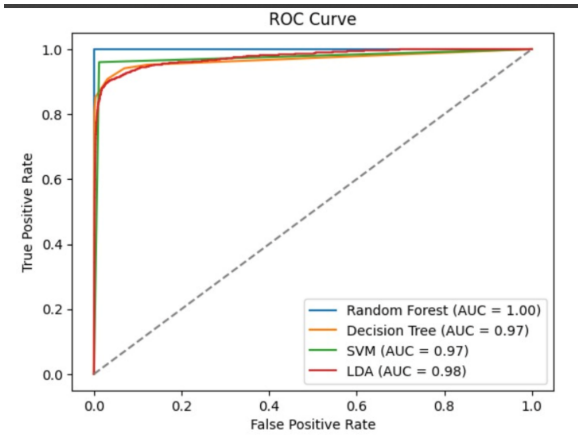
```
Accuracy for LDA Classifier: 0.9191301901060444
Precision for LDA Classifier: 0.9844855908657649
Recall for LDA Classifier: 0.8520078629598428
F1 Score for LDA Classifier: 0.9134685044926377


Confusion Matrix for LDA Classifier:
 [[55985   765]
 [ 8432 48544]]
```

ROC and AUC curve:

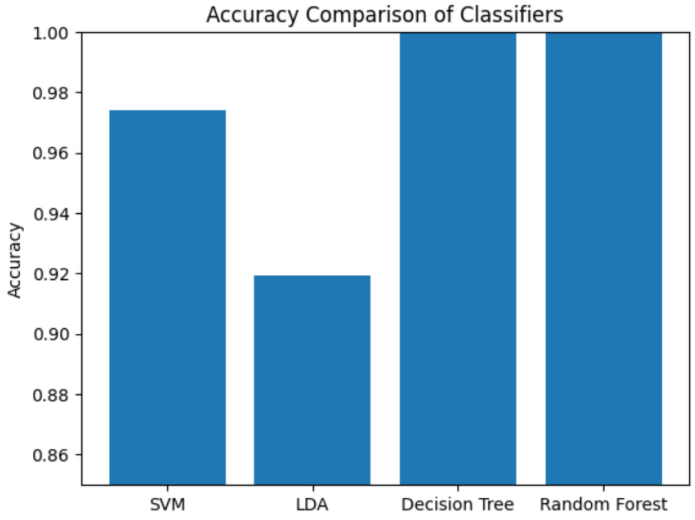
The ROC (Receiver Operating Characteristic) curve is a graphical representation of the trade-off between the true positive rate (TPR) and false positive rate (FPR) of a classifier, as its discrimination threshold is varied. It plots the TPR against FPR at various threshold settings, with higher AUC indicating better performance. The ROC and AUC are widely used evaluation metrics for binary classification models, and are useful in situations where the cost of false positives and false negatives is different.

We plot the ROC curve for

- SVM classifier
- RandomForest classifier
- Decision Tree classifier
- LDA classifier



The accuracies of Decision Tree classifier , RandomForest Classifier, SVM classifier and LDA classifier.




```
Accuracy for Randomforest Classifier: 0.9999560346798445
Accuracy for DecisionTree Classifier: 0.9996488764044944
Accuracy for SVM Classifier: 0.97407804723634
Accuracy for LDA Classifier: 0.9191301901060444
```

K Fold :

Finding the mean of KFold for 5 number of splits and using Decision Tree classifier to train.

```
Mean Accuracy: 0.9992345695475876
```

Training the KFold with LDA classifier

```
Mean Accuracy: 0.9993820375742486
```