CREDIT CARD FRAUD DETECTION

FINAL PROHECT

DATA SCIENCE

Data science is an interdisciplinary field that encompasses a set of techniques, processes, and methods for extracting valuable insights, knowledge, and patterns from data. It combines elements of statistics, computer science, domain expertise, and data engineering to collect, analyze, and interpret large and complex datasets. The ultimate goal of data science is to use data to inform decision-making, solve problems, and drive improvements in various domains, including business, healthcare, finance, and more. Data scientists use a combination of data analysis, machine learning, data visualization, and domain-specific expertise to uncover meaningful information from data and provide valuable insights for organizations and individuals.

Tools used

1. Python

Python is often used as a support language for software developers, for build control and management, testing, and in many other ways.

2. Python idle

IDLE can be used to execute a single statement and create, modify, and execute Python scripts. IDLE provides a fully-featured text editor to create Python scripts that include features like syntax highlighting, autocompletion, and smart indent.

3. PyCharm

PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers, tightly integrated to create a convenient environment for productive Python, web, and data science development

4. A KAGGLE DATASET

It also known as a Kaggle Kernel is a set of data provided by companies, students, and alum. These data sets allow competitors to work through problems, or use them as a practice simulation.

5. MATLAB

In Data Science, MATLAB is used for simulating neural networks and fuzzy logic. Using the MATLAB graphics library, you can create powerful visualizations. MATLAB is also used in image and signal processing.

6. Excel

Excel is a powerful analytical tool for Data Science. While it has been the traditional tool for data analysis, Excel still packs a punch. Excel comes with various formulae, tables, filters, slicers, etc. You can also create your own custom functions and formulae using Excel. While Excel is not for calculating the huge amount of Data, it is still an ideal choice for creating powerful data visualizations and spreadsheets.

7. Jupyter

Project Jupyter is an open-source tool based on IPython for helping developers in making open-source software and experiences interactive computing. Jupyter supports multiple languages like Julia, Python, and R.

8. Matplotlib

Matplotlib is a plotting and visualization library developed for Python. It is the most popular tool for generating graphs with the analyzed data. It is mainly used for plotting complex graphs using simple lines of code. Using this, one can generate bar plots, histograms, scatterplots etc.

9. Sciket-learn

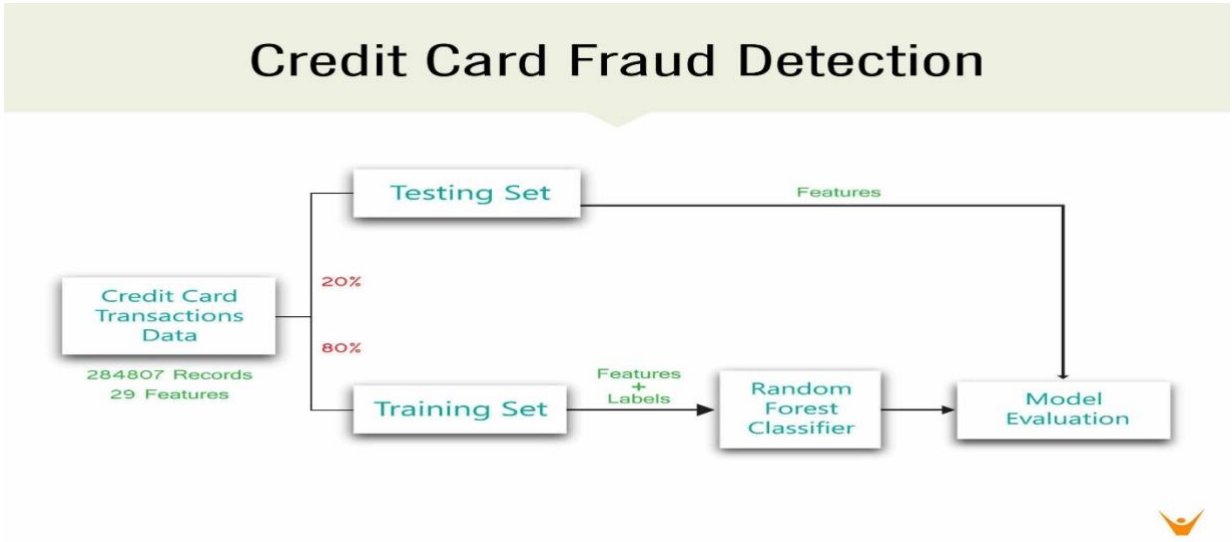
Scikit-learn is a library-based in Python that is used for implementing Machine Learning Algorithms. It is simple and easy to implement a tool that is widely used for analysis and data science. Scikit-learn makes it easy to use complex machine learning algorithms. It is therefore in situations that require rapid prototyping and is also an ideal platform to perform research requiring basic Machine Learning. It makes use of several underlying libraries of Python such as SciPy, Numpy, Matplotlib, etc.

CREDIT CARD FRAUD DETECTION

Credit card fraud is a term that has been coined for unauthorized access of payment cards like credit cards or debit cards to pay for using services or goods. Hackers or fraudsters may obtain the confidential details of the card from unsecured websites. When a fraudster compromises an individual's credit/debit card, everyone involved in the process suffers, right

from the individual whose confidential data has been leaked to the businesses (generally banks) who issue the credit card and the merchant who is finalizing the transaction with purchase. This makes it extremely essential to identify the fraudulent transactions at the

onset. Financial institutions and businesses like e-commerce are taking firm steps to flag the fraudsters entering the system. Various advanced machine learning technologies are at play, assessing every transaction and stemming the fraud users in its nip using behavioral data and transaction patterns. The process of automatically differentiating between fraudulent and genuine users is known as “credit card fraud detection”.



Lost/Stolen cards: People steal credit cards from the mail and use them illegally on behalf of the owner. The process of blocking credit cards that have been stolen and re-issuing them is a hassle for both customers and credit card companies. Some financial institutions keep the credit cards blocked until it is verified that the rightful owner has received the card.

Card Abuse: The customer buys goods and items on the credit card but has no intention to pay back the amount charged by the bank for the same. These customers stop answering the calls as the deadline to settle the dues approaches. Sometimes they even declare bankruptcy—this type of fraud results in losses of millions every year.

Identity Theft: The customers apply illegitimate information, and they might even steal the details of a genuine customer to apply for a credit card and then misuse it. In such cases, even card blocking can not stop the credit card from falling into the wrong hands.

Merchant Abuse: Some merchants show illegal transactions (that never occurred) for money laundering. For performing these illicit transactions, legal information of genuine credit card users is stolen to generate replicas of the cards and use it for illegal work.

Problem Definition:

The problem is to develop a machine learning-based system for real-time credit card fraud detection. The goal is to create a solution that can accurately identify fraudulent transactions while minimizing false positives. This project involves data preprocessing, feature engineering, model selection, training, and evaluation to create a robust fraud detection system

Six Phases of data science:

1.Datasource

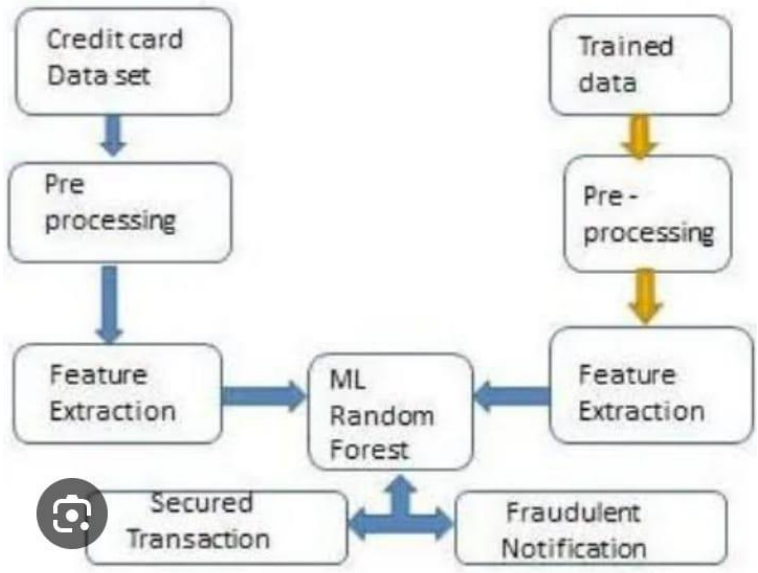
2.Data-Preprocessing

3.Feature Engineering

4.Model Selection

5.Model Training

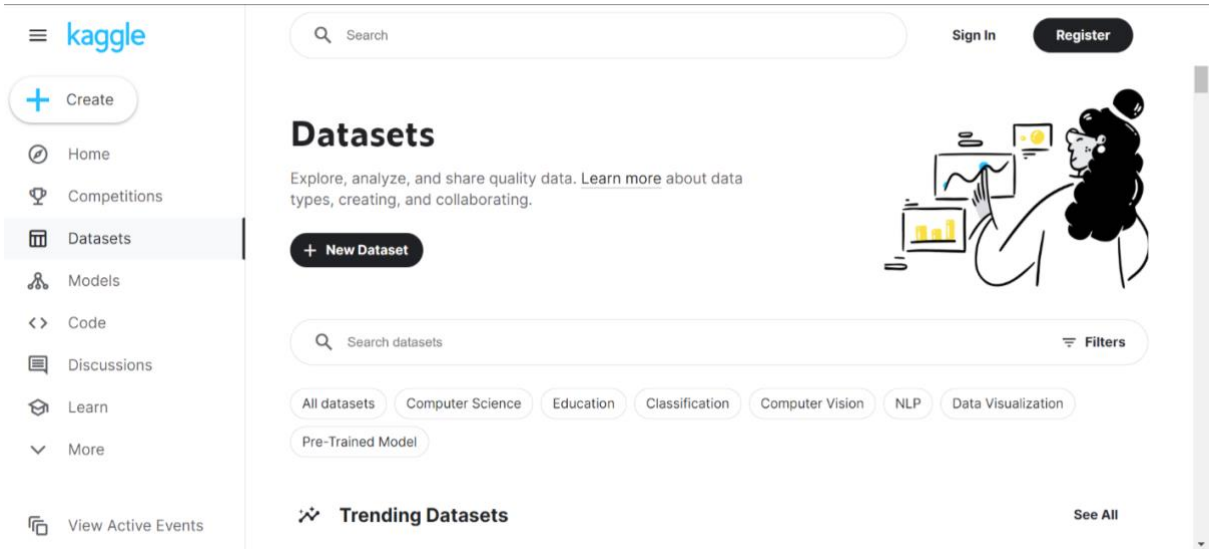
6.Evaluation



1.DATASOURCE:

This dataset is taken from the website name ”**kaggle**” and the link of the website is given below

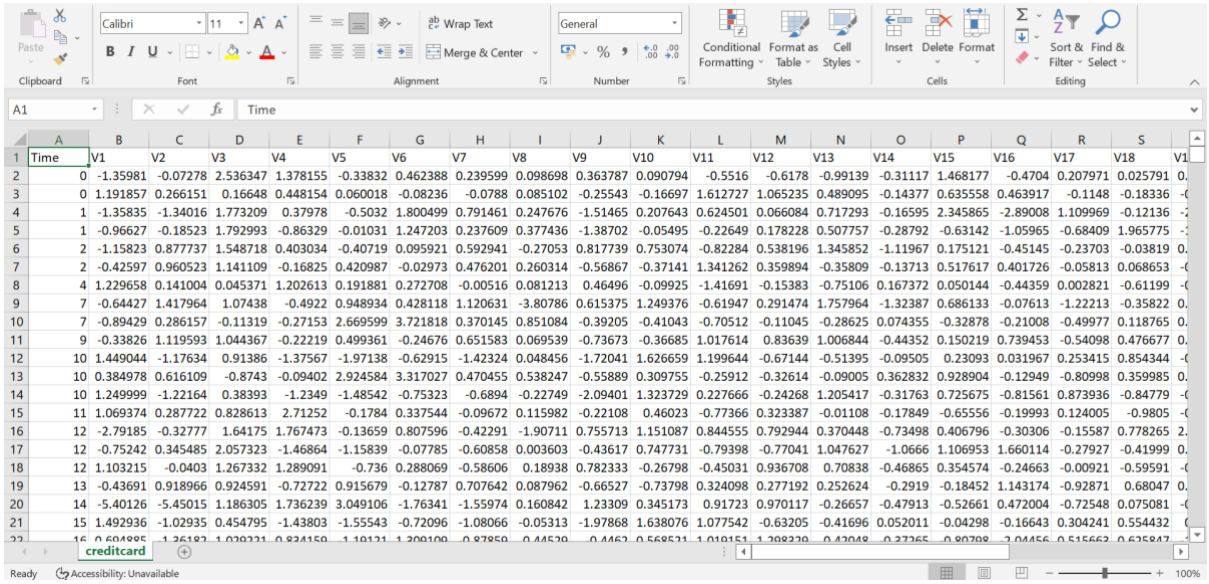
Link: www.kaggle.com/data



In this page search for your desired dataset about the credit card fraud deduction

The link for this project:

https://www.kaggle.com/datasets/mig-ulb/creditcardfraud



Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18
0	-1.35981	-0.07278	2.536347	1.378155	-0.33832	0.462388	0.239599	0.098698	0.363787	0.090794	-0.5516	-0.6178	-0.99139	-0.31117	1.468177	-0.4704	0.207971	0.025791
0	1.191857	0.266151	0.16648	0.448154	0.060018	-0.08236	-0.0788	0.085102	-0.25543	-0.16697	1.612727	1.065235	0.489095	-0.14377	0.635558	0.463917	-0.1148	-0.18336
1	-1.35835	-1.34016	1.773209	0.37978	-0.5032	1.800499	0.791461	0.247676	-1.51465	0.207643	0.624501	0.066084	0.717293	-0.16595	2.345865	-2.89008	1.109969	-0.12136
1	-0.96627	-0.18523	1.792993	-0.86329	-0.01031	1.247203	0.237609	0.377436	-1.38702	-0.05495	-0.22649	0.178228	0.507757	-0.28792	-0.63142	-1.05965	-0.68409	1.965775
2	-1.15823	0.877737	1.548718	0.403034	-0.40719	0.095921	0.592941	-0.27053	0.817739	0.753074	-0.82284	0.538196	1.345852	-1.11967	0.175121	-0.45145	-0.23703	-0.03819
2	-0.42597	0.960523	1.141109	-0.16825	0.420987	-0.02973	0.476201	0.260314	-0.56867	-0.37141	1.341262	0.359894	-0.35809	-0.13713	0.517617	0.401726	-0.05813	0.068653
4	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.00516	0.081213	0.46496	-0.09925	-1.41691	-0.15383	-0.75106	0.167372	0.050144	-0.44359	0.002821	-0.61199
7	-0.64427	1.417964	1.07438	-0.4922	0.948934	0.428118	1.120631	-3.80786	0.615375	1.249376	-0.61947	0.291474	1.757964	-1.32387	0.686133	-0.07613	-1.22213	-0.35822
7	-0.89429	0.286157	-0.11319	-0.27153	2.669599	3.721818	0.370145	0.851084	-0.39205	-0.41043	-0.70512	-0.11045	-0.28625	0.074355	-0.32878	-0.21008	-0.49977	0.118765
9	-0.33826	1.119593	1.044367	-0.22219	0.499361	-0.24676	0.651583	0.069539	-0.73673	-0.36685	1.017614	0.83639	1.006844	-0.44352	0.150219	0.739453	-0.54098	0.476677
10	1.449044	-1.17634	0.91386	-1.37567	-1.97138	-0.62915	-1.42324	0.048456	-1.72041	1.626659	1.199644	-0.67144	-0.51395	-0.09505	0.23093	0.031967	0.253415	0.854344
10	0.384978	0.616109	-0.8743	-0.09402	2.924584	3.317027	0.470455	0.538247	-0.55889	0.309755	-0.25912	-0.32614	-0.09005	0.362832	0.928904	-0.12949	-0.80998	0.359985
10	1.249999	-1.22164	0.38393	-1.2349	-1.48542	-0.75323	-0.6894	-0.22749	-2.09401	1.323729	0.227666	-0.24268	1.205417	-0.31763	0.725675	-0.81561	0.873936	-0.84779
11	1.069374	0.287722	0.828613	2.71252	-0.1784	0.337544	-0.09672	0.115982	-0.22108	0.46023	-0.77366	0.323387	-0.01108	-0.17849	-0.65556	-0.19993	0.124005	-0.9805
12	-2.79185	-0.32777	1.64175	1.767473	-0.13659	0.807596	-0.42291	-1.90711	0.755713	1.151087	0.844555	0.792944	0.370448	-0.73498	0.406796	-0.30306	-0.15587	0.778265
12	-0.75242	0.345485	2.057323	-1.46864	-1.15839	-0.07785	-0.60858	0.003603	-0.43617	0.747731	-0.79398	-0.77041	1.047627	-1.0666	1.106953	1.660114	-0.27927	-0.41999
12	1.103215	-0.0403	1.267332	1.289091	-0.736	0.288069	-0.58606	0.18938	0.782333	-0.26798	-0.45031	0.936708	0.70838	-0.46865	0.354574	-0.24663	-0.00921	-0.59591
13	-0.43691	0.918966	0.924591	-0.72722	0.915679	-0.12787	0.707642	0.087962	-0.66527	-0.73798	0.324098	0.277192	0.252624	-0.2919	-0.18452	1.143174	-0.92871	0.68047
14	-5.40126	-5.45015	1.186305	1.736239	3.049106	-1.76341	-1.55974	0.160842	1.23309	0.345173	0.91723	0.970117	-0.26657	-0.47913	-0.52661	0.472004	-0.72548	0.075081
15	1.492936	-1.02935	0.454795	-1.43803	-1.55543	-0.72096	-1.08066	-0.05313	-1.97868	1.638076	1.077542	-0.63205	-0.41696	0.052011	-0.04298	-0.16643	0.304241	0.554432

DETAILS ABOUT DATASET

The dataset consists about the data’s of the credit cards and the time in which the fraud has been taken place with the attributes.

- In the given dataset for Credit card fraud detection,the rows are aligned from V1 to V28 and with amount and class.
- The columns are aligned with integers from 0 to 114.
- The values in dataset are both positive and negative.
- In the amount column, the highest value is 3828 and the least value is 0.75.
- In the class column,the values are almost equal to zero's and one's(Boolean values)
- There are almost 284808 tuples present

2.DATA-PREPROCESSING:

BEGIN BUILDING THE PROJECT BY IMPORTING THE DATASET

importing the libraries

import numpy as np import pandas as pd

import time

import matplotlib.pyplot as plt matplotlib inline

import seaborn as sns

from scipy import stats

from scipy.stats import norm, skew from scipy.special import boxcoxip

from scipy.stats import boxcox_normmax

from sklearn import preprocessing

from sklearn.preprocessing import StandardScaler

import sklearn

from sklearn import metrics

from sklearn.metrics import roc curve, auc, roc auc_score

Fom sklearn.metrics import classification report, confusion matrix from sklearn.metrics

Import average precision score, precision_recall_curve

from sklearn.model selection import train_test_split

from sklearn.model_selection import Stratified Fold

From sklearn.model selection import GridSearchCV, RandomizedSearchCV

from scipy.special import boxcoxip

from scipy.stats import boxcox_normmax

from sklearn import preprocessing

from sklearn.preprocessing import StandardScaler

import sklearn

```
from sklearn import metrics
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import classification_report, confusion matrix
from sklearn.metrics import average precision score, precision_recall_curve
from sklearn.model_selection import train_test_split
from sklearn.model_selection import Stratified Fold from sklearn.model_selection import GridsearchCV, RandomizedSearchCV
from sklearn.linear_model import Ridge, Lasso, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegressionCV from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from xgboost import XGBClassifier
From sklearn.ensemble import AdaBoostClassifier
```

READING THE DATA FROM THE DATASET :

By using Google Colaboratory

Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.

IMPORTING DATASET IN GOOGLE COLAB:

We can use google colaboratory by creating a file namely creditcard.csv and import them.

```
From google.colab import drive drive.mount("/content/gdrive")
Mounted at /content/gdrive
```

Importing dataset in any other softwares:

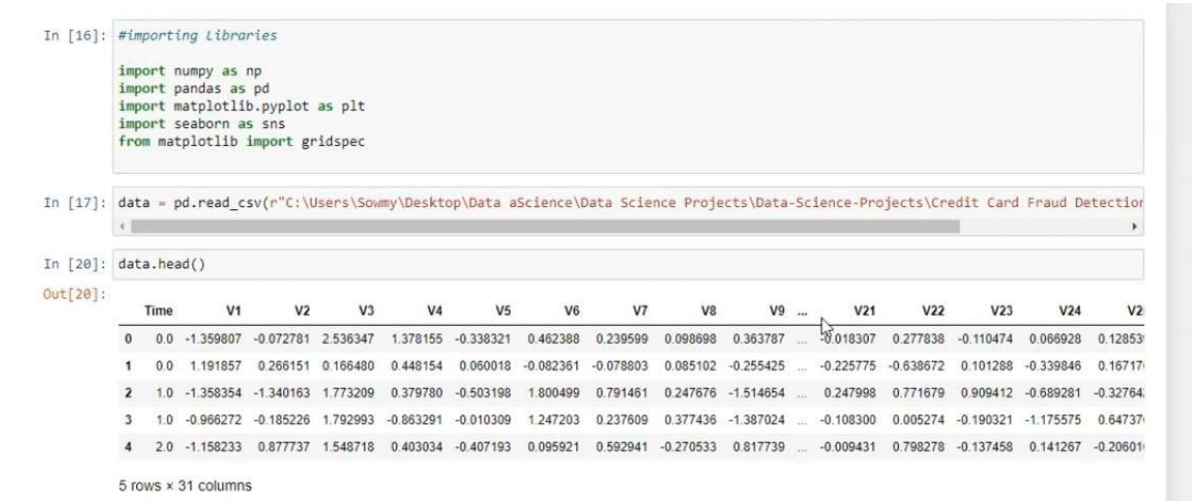
```
Df = pd.read_csv("gdrive/MyDrive/Colab Notebooks/creditcard.csv")
```

DISPLAYING ONLY THE FIRST FIVE DATAS FROM THE DATASET:

Head:

The head() method returns a specified number of rows, string from the top. The head() method returns the first 5 rows if a number is not specified. Note: The column names will also be returned, in addition to the specified rows.

Df.head()



```
In [16]: #importing Libraries

         import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import seaborn as sns
         from matplotlib import gridspec

In [17]: data = pd.read_csv(r"C:\Users\Sowmy\Desktop\Data aScience\Data Science Projects\Data-Science-Projects\Credit Card Fraud Detectior

In [20]: data.head()

Out[20]:
```

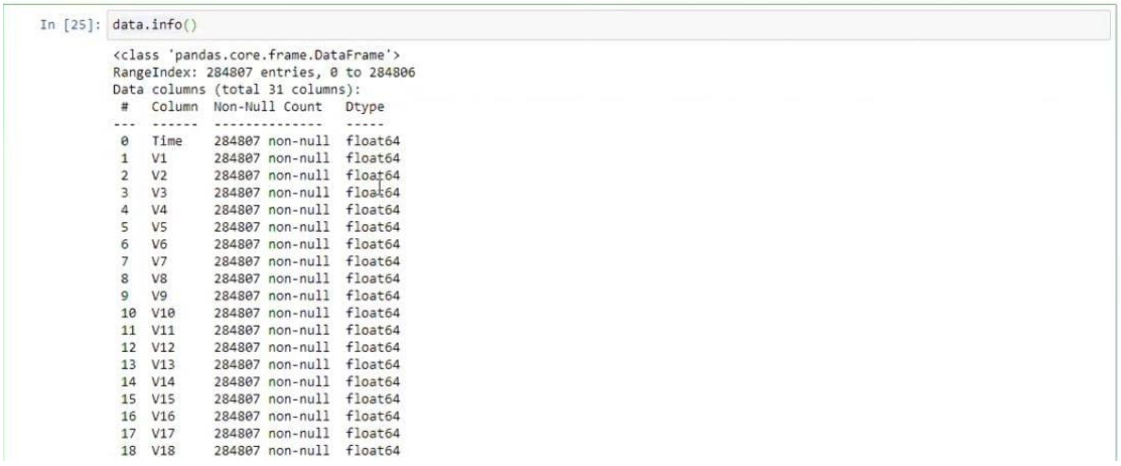
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16717
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	-0.32764
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	0.64737
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	-0.20601

5 rows × 31 columns

DISPLAYING THE INFORMATION ABOUT THE DATASET:

The info() method prints information about the DataFrame. The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values). Note: the info() method actually prints the info.

Df.info()



```
In [25]: 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
```

DISPLAYING THE DESCRIPTION ABOUT THE DATASET:

The describe() method returns description of the data in the DataFrame. If the DataFrame contains numerical data, the description contains these information for each column: count – The number of not-empty values. Mean – The average (mean) value.

```
Df.describe
```

```
In [21]: data.shape
Out[21]: (284807, 31)

In [26]: data.describe()
Out[26]:
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
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
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.040130e-15	-1.698953e-15	-1.893285e-16	-3.147640e-15
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
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
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
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
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
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

8 rows × 31 columns

GETTING THE INDEPENDENT VARIABLE FROM THE DATASET:

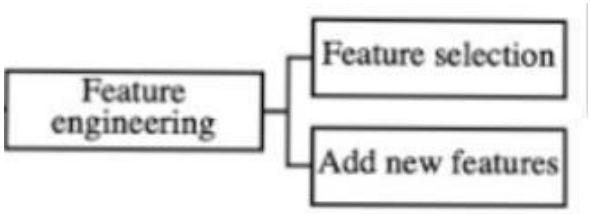
To set the difference between independent and dependent variables we need to consider the class column as y variable

```
Df['class'].value_count()
```

Where, Value_count() defines about the value_counts() function returns a Series that contain counts of unique values. It returns an object that will be in descending order so that its first element will be the most frequently-occurred element. By default, it excludes NA values.

3. FEATURE ENGINEERING :

Feature engineering refers to manipulation — addition, deletion, combination, mutation — of your data set to improve model training, leading to better performance and greater accuracy. Effective feature engineering is based on sound knowledge of the business problem and the available data sources.



SPLIT TRAIN AND TEST DATA :

Splitting the dataset into x and y

Y=df['class']

X=df.drop(['class'],axis=1)

Checking some rows of x

x.head()

Checking some rows of y

y. head()

Splitting The Dataset Using Train Test Split :

X_train,x_test,y_train,y_test=train_test_split(x,y,random_state=100,test_size=0.20)

Preserve x_test and y_ test to evaluate

Checking The Spread Of Data Post Split :

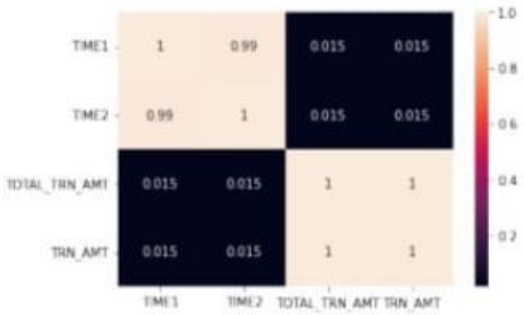
Print(np sum(y))

Print(np sum(y_train))

Print(np sum(y_test))

A. Feature selection

The dataset contains non-numeric values. As shown in the Table 1, there are two columns for time and two columns for the amount of the transaction. To investigate whether these columns have the same values or not we perform the heat map for them in Figure 1.



CONFUSION MATRIX :

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes.

The following 4 are the basic terminology which will help us in determining the metrics we are looking for.

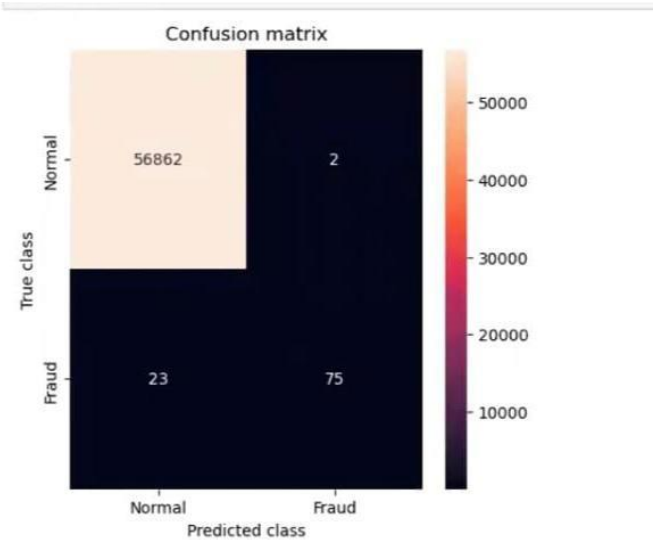
True Positives (TP): when the actual value is Positive and predicted is also Positive.

True Negatives (TN): when the actual value is Negative and prediction is also Negative.

False Positives (FP): When the actual is negative but prediction is Positive. Also known as the Type 1 error

False Negatives (FN): When the actual is Positive but the prediction is Negative. Also known as the Type 2 error.

```
Confusion_matrix_dt=confusion_matrix(test_Y,predictions_dt.round())
Print("Confusion Matrix – Decision Tree")
Print(confusion_matrix_dt)
Plot_confusion_matrix(confusion_matrix_dt, classes=[0, 1], title= "Confusion Matrix – Decision Tree"
```



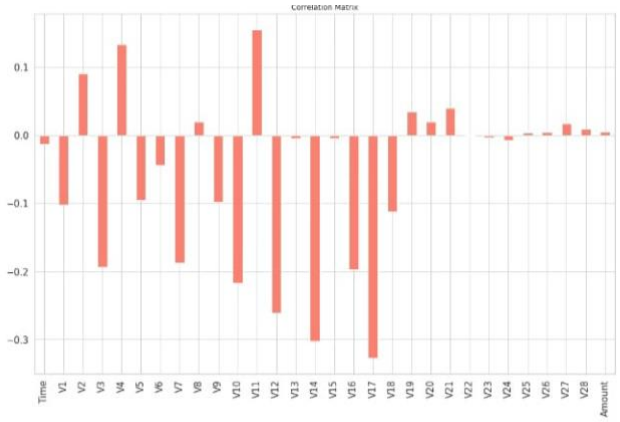
HANDLING THE MISSING DATA :

Missing values can affect the performance of the anomaly Detection algorithm.

Therefore, it is essential to check whether there are anyMissing values in the dataset and take appropriate action.

Check if there are any missing values in the dataset

```
Print(df.isnull().sum()
```



Accuracy:

Accuracy is a metric that generally describes how the model performs across all classes. It is useful when all classes are of equal importance. It is calculated as the ratio between the number of correct predictions to the total number of predictions

CODE FOR ACCURACY OF A CLASSIFER

Importing All necessary libraries

from sklearn.metrics import accuracy_score

Calculating the accuracy of classifier

```
print(Accuracy of the classifier is: {accuracy_score(y_test, predictions)}")
```

IMPLEMENTATION OF ACCURACY METRICS

The Accuracy score is calculated by dividing the number of correct predictions by the total prediction number.

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

4.MODEL SELECTION

Model selection in machine learning is the process of selecting the best algorithm and model architecture for a specific job or dataset. It involves assessing and contrasting various models to identify the one that best fits the data and produces the best results.

The process of model selection is crucial in machine learning as it helps to avoid overfitting, which occurs when a model is too complex and fits the training data too well, but fails to

generalize to new data. Model selection also helps to improve the interpretability of the model, making it easier to explain to stakeholders

ALGORITHMS USED IN CREDIT CARD FRAUD DEDUCTION

1.Decision Tree

2.Predictive Analytics and Algorithms

3.Clustering Techniques

4.K-Nearest Neighbour Algorithm

5.Neural Networks

6.Naive Bayes Classifiers

7.Support Vector Machines (SVMs)

8.Ensemble Stacking

9. XGB Classifier

10. Gradient Boosting Classifier

Train_test_split data:

```
st.header('Train and test split')


    # Putting feature variables into X
    X = df.drop(['Class'], axis=1)


    # Putting target variable to y
    y = df['Class']
    # Splitting data into train and test set 80:20
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, test_size = 0.2,
random_state = 42)


    st.write('X_train: ', X_train.shape)
    st.write('y_train: ', y_train.shape)
    st.write('X_test: ', X_test.shape)
    st.write('y_test: ', y_test.shape)
    # --- TRAIN AND TEST SPLIT ---
```

1.Logestic regression:

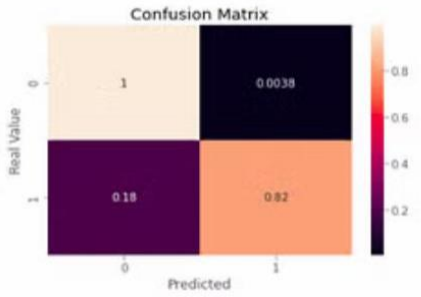
Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

```
# --- START Logistic regression ---
    st.header('Logistic Regression')
    # --- START Training the Logistic Regression Model on the Training set ---
    st.subheader('Training the Logistic Regression Model on the Training set')
    LR_model = LogisticRegression(random_state = 0)
    LR_model.fit(X_train, y_train)
    y_train_pred = LR_model.predict(X_train)
    y_test_pred = LR_model.predict(X_test)
    acc1 = accuracy_score(y_test, y_test_pred)

  # Train Score
    st.write('Recall score: %0.4f'% recall_score(y_train, y_train_pred))
    st.write('Precision score: %0.4f'% precision_score(y_train, y_train_pred))
    st.write('F1-Score: %0.4f'% f1_score(y_train, y_train_pred))
    st.write('Accuracy score: %0.4f'% accuracy_score(y_train, y_train_pred))
    st.write('AUC: %0.4f' % roc_auc_score(y_train, y_train_pred))

    # Train Predictions
    st.pyplot(visualize_confusion_matrix(y_train, y_train_pred))
    st.pyplot(ROC_AUC(y_train, y_train_pred))
    # --- END Training the Logistic Regression Model on the Training set ---
    # --- START Training the Logistic Regression Model on the Testing set ---
    st.subheader('Training the Logistic Regression Model on the Testing set')
    # Test score
    st.write('Recall score: %0.4f'% recall_score(y_test, y_test_pred))
    st.write('Precision score: %0.4f'% precision_score(y_test, y_test_pred))
    st.write('F1-Score: %0.4f'% f1_score(y_test, y_test_pred))
    st.write('Accuracy score: %0.4f'% accuracy_score(y_test, y_test_pred))
```

```
st.write('AUC: %0.4f' % roc_auc_score(y_test, y_test_pred))
```

```
# Test Predictions
st.pyplot(visualize_confusion_matrix(y_test, y_test_pred))
st.pyplot(ROC_AUC(y_test, y_test_pred))
# --- END Training the Logistic Regression Model on the Testing set ---
# Result
st.header('Results')
st.subheader('Training set')
st.text('- Recall score: 0.6397\n- Precision score: 0.8688\n- F1-Score: 0.7368\n- Accuracy score: 0.9992\n- AUC: 0.8198'
st.subheader('Testing set')
st.text('- Recall score: 0.5556\n- Precision score: 0.9091\n- F1-Score: 0.6897\n- Accuracy score: 0.9992\n- AUC: 0.7777')
# --- END Logistic regression ---
```

	precision	recall	f1-score	support
0.0	0.96	1.00	0.98	520
1.0	0.98	0.82	0.89	103
accuracy			0.97	623
macro avg	0.97	0.91	0.93	623
weighted avg	0.97	0.97	0.97	623



2.NAÏVE BAYES MODEL:

```
# --- START Training the Naive Bayes Model on the Training set ---
st.subheader('Training the Naive Bayes Model on the Training set')
```

```
NB_model = GaussianNB()
NB_model.fit(X_train, y_train)
y_train_pred = NB_model.predict(X_train)
y_test_pred = NB_model.predict(X_test)
acc2 = accuracy_score(y_test, y_test_pred)

# Train Score
st.write('Recall score: %0.4f'% recall_score(y_train, y_train_pred))
st.write('Precision score: %0.4f'% precision_score(y_train, y_train_pred))
st.write('F1-Score: %0.4f'% f1_score(y_train, y_train_pred))
st.write('Accuracy score: %0.4f'% accuracy_score(y_train, y_train_pred))
st.write('AUC: %0.4f' % roc_auc_score(y_train, y_train_pred))

# Train Predictions
st.pyplot(visualize_confusion_matrix(y_train, y_train_pred))
st.pyplot(ROC_AUC(y_train, y_train_pred))
# --- END Training the Naive Bayes Model on the Training set ---


# --- START Training the Naive Bayes Model on the Testing set ---
st.subheader('Training the Naive Bayes Model on the Testing set')

# Test score
st.write('Recall score: %0.4f'% recall_score(y_test, y_test_pred))
st.write('Precision score: %0.4f'% precision_score(y_test, y_test_pred))
st.write('F1-Score: %0.4f'% f1_score(y_test, y_test_pred))
st.write('Accuracy score: %0.4f'% accuracy_score(y_test, y_test_pred))
st.write('AUC: %0.4f' % roc_auc_score(y_test, y_test_pred))

# Test Predictions
st.pyplot(visualize_confusion_matrix(y_test, y_test_pred))
```

st.pyplot(ROC_AUC(y_test, y_test_pred))

 # --- END Training the Naive Bayes Model on the Testing set ---

 # Result

st.header('Results')

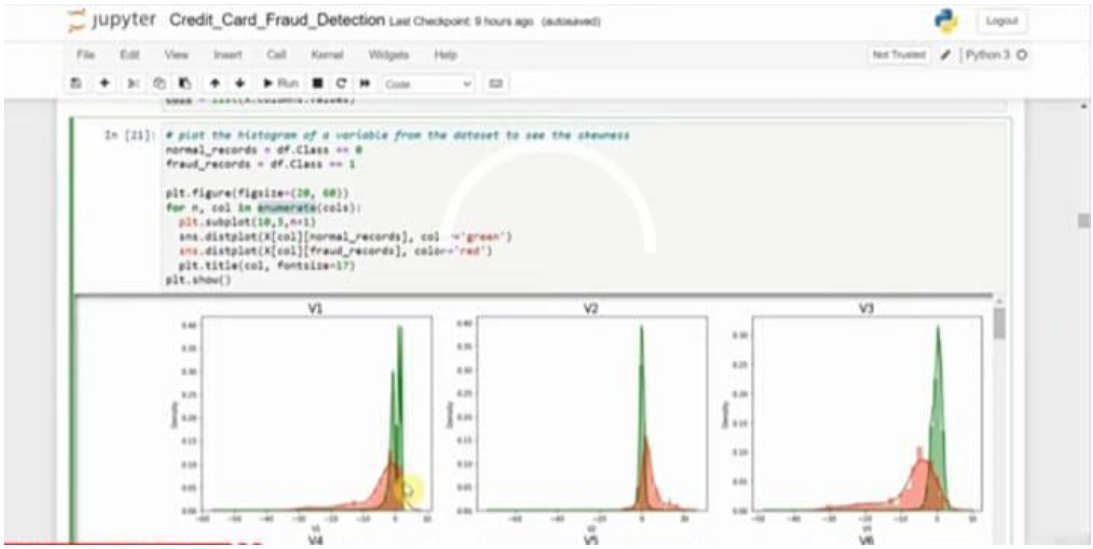
st.subheader('Training set')

St. Text('- Recall score: 0.8277\n- Precision score: 0.0604\n- F1-Score: 0.1125\n- Accuracy score: 0.9780\n- AUC: 0.9030')

st.subheader('Testing set')

st.text('- Recall score: 0.7778\n- Precision score: 0.0523\n- F1-Score: 0.0980\n- Accuracy score: 0.9773\n- AUC: 0.8777')

 # --- END Naive Bayes ---



3.Decision tree

 # --- START Decision tree ---

st.header('Decision tree')

 # --- START Training the Decision tree Model on the Training set ---

st.subheader('Training the Decision tree Model on the Training set')

DTR_model = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)

DTR_model.fit(X_train, y_train)

y_train_pred = DTR_model.predict(X_train)

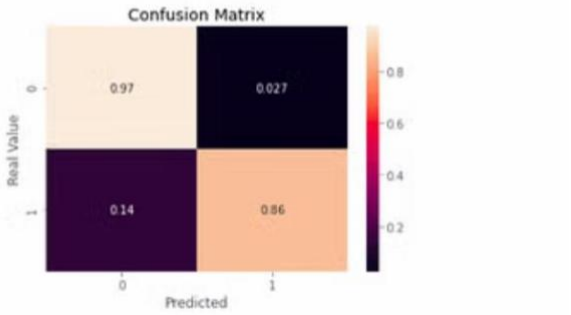
y_test_pred = DTR_model.predict(X_test)

acc4 = accuracy_score(y_test, y_test_pred)

```
# Train Score
st.write('Recall score: %0.4f'% recall_score(y_train, y_train_pred))
st.write('Precision score: %0.4f'% precision_score(y_train, y_train_pred))
st.write('F1-Score: %0.4f'% f1_score(y_train, y_train_pred))
st.write('Accuracy score: %0.4f'% accuracy_score(y_train, y_train_pred))
st.write('AUC: %0.4f' % roc_auc_score(y_train, y_train_pred))
st.pyplot(visualize_confusion_matrix(y_train, y_train_pred))
st.pyplot(ROC_AUC(y_train, y_train_pred))
# --- END Training the Decision tree Model on the Training set ---


# --- START Training the Decision tree Model on the Testing set ---
st.subheader('Training the Decision tree Model on the Testing set')
st.write('Recall score: %0.4f'% recall_score(y_test, y_test_pred))
st.write('Precision score: %0.4f'% precision_score(y_test, y_test_pred))
st.write('F1-Score: %0.4f'% f1_score(y_test, y_test_pred))
st.write('Accuracy score: %0.4f'% accuracy_score(y_test, y_test_pred))
st.write('AUC: %0.4f' % roc_auc_score(y_test, y_test_pred))
st.pyplot(visualize_confusion_matrix(y_test, y_test_pred))
st.pyplot(ROC_AUC(y_test, y_test_pred))
# --- END Training the Decision tree Model on the Testing set ---


# Result
st.header('Results')
st.subheader('Training set')
st.text('- Recall score: 1.0000\n- Precision score: 1.0000\n- F1-Score: 1.0000\n- Accuracy score: 1.0000\n- AUC: 1.0000')
st.subheader('Testing set')
st.text('- Recall score: 0.6889\n- Precision score: 0.7561\n- F1-Score: 0.7209\n- Accuracy score: 0.9992\n- AUC: 0.8443')
```

 # --- END Decision tree ---

	precision	recall	f1-score	support
0.0	0.97	0.97	0.97	520
1.0	0.86	0.86	0.86	103
accuracy			0.96	623
macro avg	0.92	0.92	0.92	623
weighted avg	0.96	0.96	0.96	623



 # Logistic Regression

```
y_pred_logistic = LR_model.predict_proba(X_test)[:,1]
logistic_fpr, logistic_tpr, threshold = roc_curve(y_test, y_pred_logistic)
auc_logistic = auc(logistic_fpr, logistic_tpr)
```

 # Naive Bayes

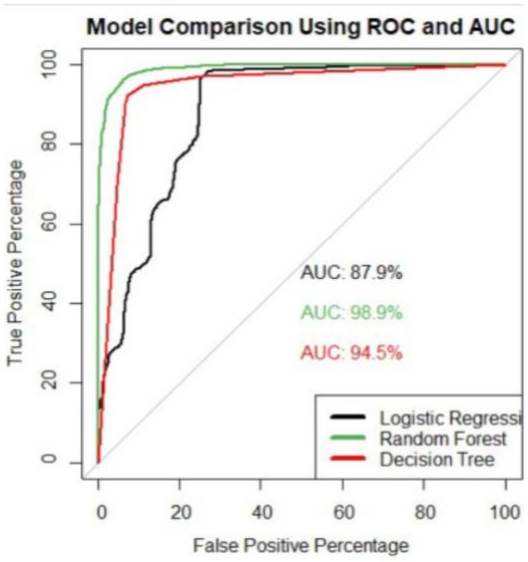
```
y_pred_nb = NB_model.predict_proba(X_test)[:,1]
nb_fpr, nb_tpr, threshold = roc_curve(y_test, y_pred_nb)
auc_nb = auc(nb_fpr, nb_tpr)
```

 # Decision Tree

```
y_pred_dtr = DTR_model.predict_proba(X_test)[:,1]
dtr_fpr, dtr_tpr, threshold = roc_curve(y_test, y_pred_dtr)
auc_dtr = auc(dtr_fpr, dtr_tpr)
```

graph plotting

```
def plottingGraphResultCompare():
    plt.figure(figsize=(10, 8), dpi=100)
    plt.plot([0, 1], [0, 1], 'k--')

    # Logistic Regression
    plt.plot(logistic_fpr, logistic_tpr, label='Logistic Regression (auc = %0.4f)' % auc_logistic)

    # Naive Bayes
    plt.plot(nb_fpr, nb_tpr, label='Naive Bayes (auc = %0.4f)' % auc_nb)

    # Decision Tree
    plt.plot(dtr_fpr, dtr_tpr, label='Decision Tree (auc = %0.4f)' % auc_dtr)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='best')
    plt.show()
```



verification of legitimate and fraudulent transactions

```
st.pyplot(plottingGraphResultCompare())
if st.sidebar.checkbox('Manual transaction verification'):
```

```
    # separate legitimate and fraudulent transactions
    legit = df[df.Class == 0]
    fraud = df[df.Class == 1]

    # undersample legitimate transactions to balance the classes
    legit_sample = legit.sample(n=len(fraud), random_state=2)
    data = pd.concat([legit_sample, fraud], axis=0)

    # split data into training and testing sets
    X = data.drop(columns="Class", axis=1)
    y = data["Class"]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=2)

    # train logistic regression model
    model = LogisticRegression()
    model.fit(X_train, y_train)

    # evaluate model performance
    train_acc = accuracy_score(model.predict(X_train), y_train)
    test_acc = accuracy_score(model.predict(X_test), y_test)

    # create Streamlit app
    st.title("Manual transaction verification")
    st.write("Enter the following features to check if the transaction is legitimate or
fraudulent:")

    # create input fields for user to enter feature values
    input_df = st.text_input('Input All features')
    input_df_lst = input_df.split(',')
    # create a button to submit input and get prediction
```

```
submit = st.button("Submit")

if submit:
    # get input feature values
    features = np.array(input_df_lst, dtype=np.float64)
    # make prediction
    prediction = model.predict(features.reshape(1,-1))
    # display result
    if prediction[0] == 0:
        st.write("Legitimate transaction")
    else:
        st.write("Fraudulent transaction")
```

5. Model Evaluation:

Model evaluation is the process of figuring out how well the model performs at guessing something.

This evaluation is usually handled with a test dataset. What we do is is we take some proportion of the total data, set it aside so it never gets involved with the model training and then we pass that data through the model once we've trained it.

We have the known outcomes of that data that's been set aside. We take the independent variables from that test dataset, and feed it through the model. The model should then spit out a bunch of guesses for the dependent variable.

We can then compare the guesses against the known dependent variable values. Depending on how the actual outcomes and the guessed outcomes match up, we can come up with all sorts of measures of model performance.

Accuracy

Accuracy is a simple and common measure for whether or not predictions were correct compared to known outcomes.Oftentimes though, it's not sufficient.

Let's say we know that 90% of the population does not snore. We could easily nail 90% accuracy on average without a model by simply saying everyone does not snore. Full stop. We'll get it wrong 10% of the time.

Using confusion matrix:

A confusion matrix is a way to compare known outcomes and guessed outcomes. If it's a binary classifier where there are two possible guesses (True/False, Yes/No, etc.), then the matrix ends up looking like a 2x2 grid. One axis represents the true outcome, and another axis represents the guessed or predicted outcome.

Usually each box of the confusion matrix gets a count, percentage, or proportion reflecting the outcomes of the different predictions from the model.

Precision

Precision is the ratio of the true positives over the sum of the true positive and false positive.You can think of this as of all the things predicted to be true, how many were actually true?

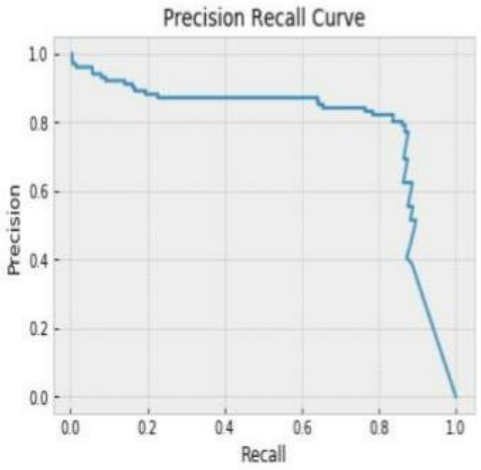
It's a way of assessing how reliable the true guesses are.

Recall

Recall is the ratio of the true positive divided by the sum of the true positive and the false positive, or _everything that is true.

Think of recall as how many of the true cases were actually found?

Recall is especially important where the cost of missing a true case is high. For instance, recall is especially important in disease diagnosis, where failing to catch a disease can have huge negative impacts on a patient.



F1 Score

This is a unified measure that combines both the precision and recall.A higher F1 score is generally deemed better.This is a handy metric to rely on when comparing multiple different models

cross validation:

Stratifiedkfold

In machine learning, When we want to train our ML model we split our entire dataset into training_set and test_set using train_test_split() class present in sklearn. Then we train our model on training_set and test our model on test_set. The problems that we are going to face in this method are:
Whenever we change the random_state parameter present in train_test_split(), We get different accuracy for different random_state and hence we can't exactly point out the accuracy for our model.
The train_test_split() splits the dataset into training_test and test_set by random sampling. But stratified sampling is performed.

Repeated K-fold Cross Validation

Repeated K-fold is the most preferred Cross-Validation technique for both classification and regression Machine Learning models. Shuffling and random sampling of the data set multiple times is the core procedure of repeated K-fold algorithm and it results in making a robust model as it covers the maximum training and testing operations.
Steps involved in the repeated K-fold cross-validation:

1. Split the data set into K subsets randomly
2. For each one of the developed subsets of data points

- Treat that subset as the validation set
- Use all the rest subsets for training purposes
- Training of the model and evaluate it on the validation set or test set
- Calculate prediction error.

By validating all the algorithms mentioned above.we have created a table which includes methodology,model, accuracy,roc_value, threshold value we can say that among these models logistic regression with L2 regularization is more accurate and precise and this model shows and perfect for our project credit card fraud deductio In machine learning, When we want to train our ML model we split our entire dataset into training_set and test_set using train_test_split() class present in sklearn. Then we train our model on training_set and test our model on test_set. The problems that we are going to face in this method are:
Whenever we change the random_state parameter present in train_test_split(),

We get different accuracy for different random_state and hence we can't exactly point out the accuracy for our model.

The train_test_split() splits the dataset into training_test and test_set by random sampling. But stratified sampling is performed.

6.Conclusion:

Credit card fraud detection is a challenging task that involves recognizing fraudulent credit card transactions so that customers of credit card companies are not charged for items they did not purchase. Enormous data is processed every day, and the model build must be fast enough to respond to the scam in time. Imbalanced data, where most transactions (99.8%) are not fraudulent, makes it really hard for detecting the fraudulent ones. Misclassified data can be another major issue, as not every fraudulent transaction is caught and reported. Adaptive techniques used against the model by scammers can also pose a challenge.

Various machine learning techniques have been proposed to detect credit card fraud, including Hidden Markov Model, Decision Trees, Logistic Regression, Support Vector Machines (SVM), Genetic algorithm, Neural Networks, Random Forests, Ensemble Stacking , XGB Classifier , Bayesian Belief Network 2

By validating all the algorithms mentioned above.we have created a table which includes methodology,model, accuracy,roc_value, threshold value we can say that among these models logistic regression with L2 regularization is more accurate and precise and this model shows and perfect for our project credit card fraud deduction.

Model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	Training Time (Second)
Ensemble Stacking (Poor)	0.934959	0.968504	0.911111	0.963964	0.938931	8.42
Extra Trees Classifier	0.906504	1.000000	0.82963	1.000000	0.906883	8.34
Decision Tree Classifier	0.886179	0.879433	0.918519	0.846847	0.898551	0.19
Gaussian NB	0.914634	0.983051	0.859259	0.981982	0.916996	0.05
Ensemble Stacking (Strong)	0.930894	0.968254	0.903704	0.963964	0.934866	21.71
Random Forest Classifier	0.922764	0.991525	0.866667	0.990991	0.924901	3.06
MLP Classifier	0.934959	0.96124	0.918519	0.954955	0.939394	11.86
XGB Classifier	0.922764	0.946154	0.911111	0.936937	0.928302	1.37
Gradient Boosting Classifier	0.918699	0.952756	0.896296	0.945946	0.923664	2.1