INSTITUTE FOR ADVANCED COMPUTING AND SOFTWARE DEVELOPMENT, AKURDI, PUNE

Credit card fraud Detection using Logistic Regression and XGBoost

PG-DBDA Mar 2024

Submitted By:
Group No: 10

Roll No.	Name.
243530	Navin Chokhat
243551	Sidharth Naik

Mr. Abhijit Nagargoje
Project Guide

Mr. Rohit Puranik
Centre Coordinator

ABSTRACT

The purpose of this project is to detect the fraudulent transactions made by credit cards by using machine learning techniques, to stop fraudsters from the unauthorized usage of customers' accounts. The increase in credit card fraud is overgrowing worldwide, which is the reason actions should be taken to stop fraudsters. Putting a limit on those actions would positively impact the customers as their money would be recovered and retrieved back into their accounts and they wouldn't be charged for items or services that were not purchased by them, which is the project's main goal. Tuning hyperparameters in logistic regression is crucial for improving model performance, preventing overfitting or underfitting, and enhancing generalization to unseen data. It also helps control model complexity, handle class imbalance, and optimize the decision boundary, leading to more accurate and efficient predictions.

Detection of the fraudulent transactions will be made by using the machine learning technique such as Logistic Regression and XG Boost this model will be used on a credit card transaction dataset.

The application of a credit card fraud detection system spans multiple areas, including real-time transaction monitoring, customer notifications, risk management, compliance, and ongoing model improvement. These applications help financial institutions reduce fraud, protect customers, and maintain regulatory compliance, ultimately leading to a safer and more secure payment ecosystem.

ACKNOWLEDGEMENT

I would like to take this opportunity to express my deepest gratitude to God, the Almighty, for blessing us with His grace and guiding our efforts to a successful conclusion. My sincere and heartfelt thanks go to our esteemed guide, Mr. Abhijit Nagargoje, for his invaluable guidance and advice during critical moments, and for steering me in the right direction. I am also profoundly grateful to our respected Centre Coordinator, Mr. Rohit Puranik, for granting us access to the necessary facilities. I extend my appreciation to the other faculty members as well. Finally, I would like to thank my friends and family for their unwavering support and encouragement throughout the course of this work.

Navin Vinodrao Chokhat
(240341225029)
Sidharth Vassudeva Naik
(240341225049)

Table of Contents

1. INTRODUCTION

Credit card fraud has seen a significant rise, reflecting broader trends in global credit card usage. In 2023, credit card fraud was the most reported type of identity theft, accounting for 48% of all reports to the Federal Trade Commission (FTC) in the United States. The FTC received over 416,000 credit card fraud reports that year, and total losses due to fraud exceeded $10 billion, marking a sharp increase from 2022. Additionally, the number of data compromises reached an all-time high, with over 3,200 reported cases in 2023.

Credit card fraud has become a significant challenge for financial institutions and consumers. With the exponential growth in online transactions, fraudsters have developed increasingly sophisticated methods to exploit vulnerabilities in payment systems. As a result, detecting and preventing fraudulent transactions has become a critical area of focus. This document outlines a comprehensive approach to credit card fraud detection using Logistic Regression, a widely used statistical method for binary classification problems.

Building on this foundation, XGBoost, a powerful ensemble learning technique, offers enhanced performance and accuracy by combining multiple decision trees. It addresses the limitations of traditional models like Logistic Regression by capturing complex patterns in data, making it a robust choice for detecting intricate fraud schemes in real time.

1.1 Purpose of the Project

We propose a Machine learning model to detect fraudulent credit card activities in online financial transactions. Analyzing fake transactions manually is impracticable due to the vast amounts of data and its complexity. However, adequately given informative features, could make it possible using Machine Learning. This hypothesis will be explored in the project.

To classify fraudulent and legitimate credit card transactions by supervised learning Algorithms such as Logistic Regression and XG Boost. To help us to get awareness about the fraudulent and without loss of any financially.

1.2 Scope

In this proposed project we designed a protocol or a model to detect fraud activity in credit card transactions. This system is capable of providing most of the essential features required to detect fraudulent and legitimate transactions. As technology changes, it becomes difficult to track the Modeling and pattern of fraudulent transactions. With the rise of machine learning, artificial intelligence, and other relevant fields of information technology, it becomes feasible to automate this process and to save some of the intensive amount of labour that is put into detecting credit card fraud.

Aims & Objectives

The primary goal of this project is to develop a sophisticated credit card fraud detection system capable of accurately identifying fraudulent transactions in real time. The project will involve the extraction and engineering of key features from large-scale financial transaction datasets to enhance the system's ability to detect fraud. Machine learning models, including Logistic Regression and XGBoost, will be meticulously trained and fine-tuned to classify transactions and identify anomalies that may indicate fraudulent activity. These models will undergo rigorous evaluation, with a focus on metrics such as accuracy, recall, and F1-score. The top-performing model will be selected for integration into the final system, ensuring a robust and reliable solution for real-time fraud detection. This system will empower financial institutions to effectively detect and prevent fraudulent transactions, adding a crucial layer of security in today's increasingly digital financial landscape.

Requirements Specification

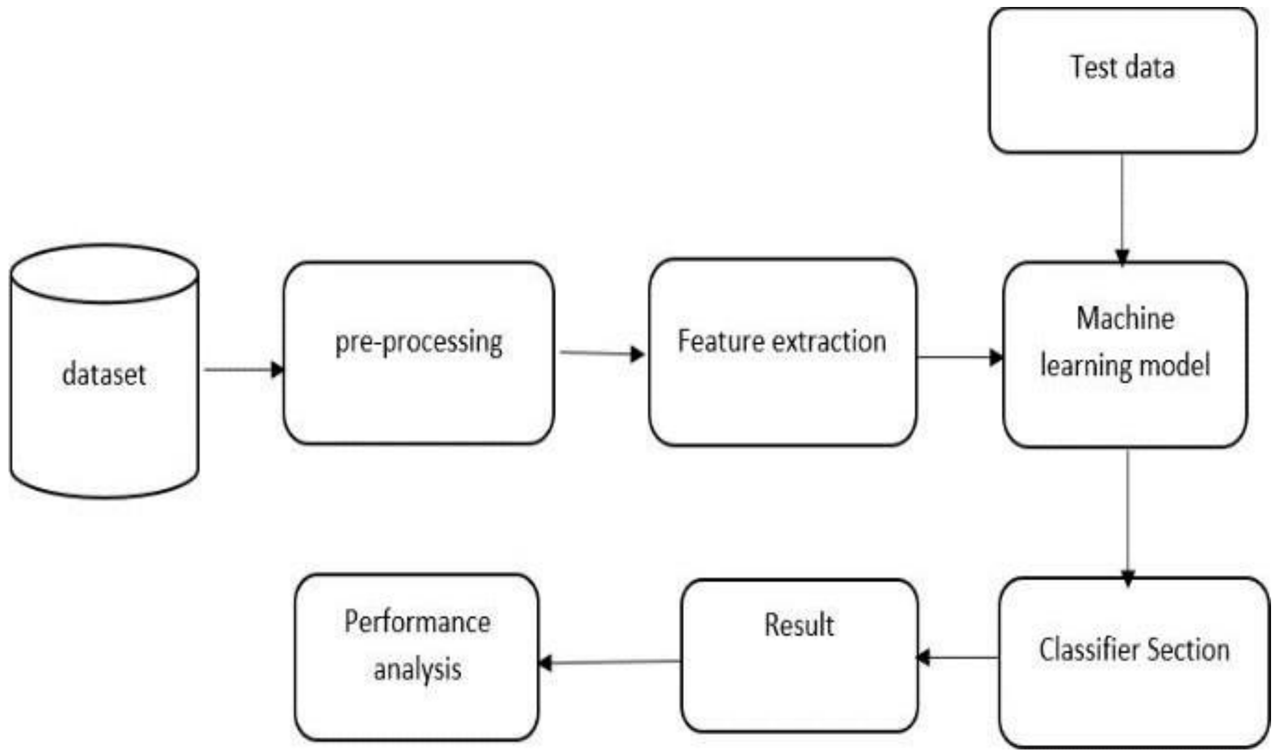
1.1 **Hardware Requirement:**

- 500 GB hard drive (Minimum requirement)
- 8 GB RAM (Minimum requirement)
- PC x64-bit CPU

1.2 **Software Requirement:**

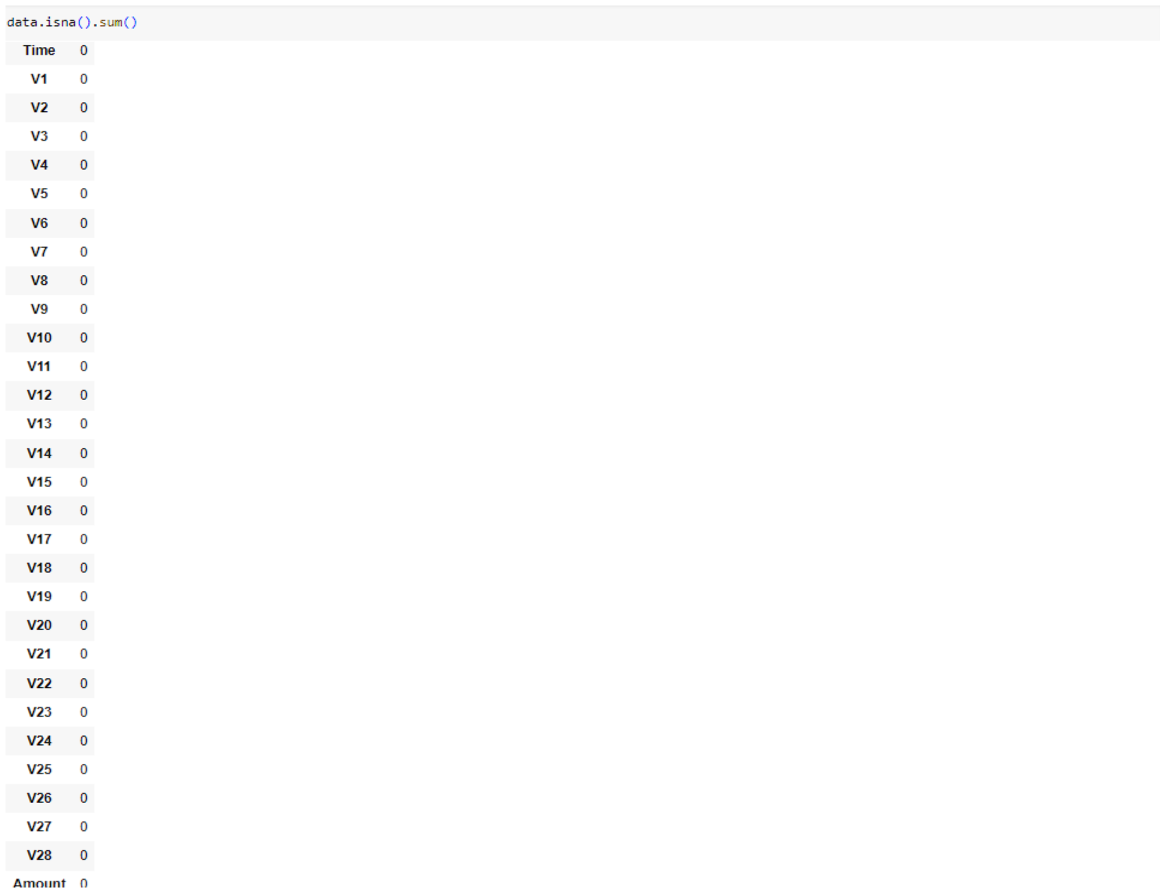
- Windows/Mac/Linux
- Python-3.9.1
- VS Code/Anaconda/Google collab
- Python Extension for VS Code
- Libraries:
  - Numpy 1.18.2
  - Pandas 1.2.1
  - Matplotlib 3.3.3
  - Scikit-learn 0.24.1

Workflow of the Project



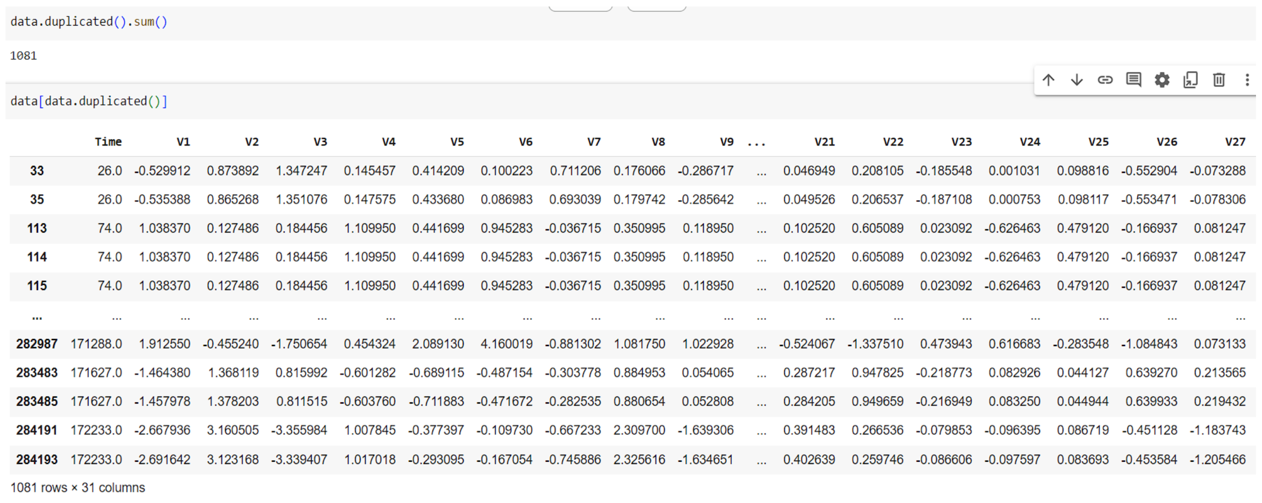
Data Preprocessing

After choosing the most suited dataset the preparation phase begins, the preparation of the dataset includes selecting the wanted attributes or variables, cleaning it by excluding Null rows, deleting duplicated variables, treating outliers if necessary, in addition to transforming data types to the wanted type, data merging can be performed as well where two or more attributes get merged. All those alterations lead to the desired result which is to make the data ready to be modeled.



```
data.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
```



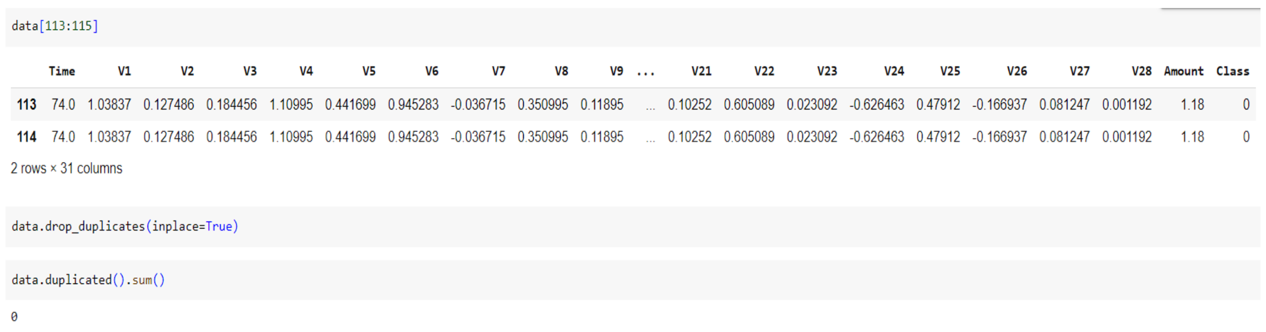
```
data.duplicated().sum()
```

```
1081
```

```
data[data.duplicated()]
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27
33	26.0	-0.529912	0.873892	1.347247	0.145457	0.414209	0.100223	0.711206	0.176066	-0.286717	...	0.046949	0.208105	-0.185548	0.001031	0.098816	-0.552904	-0.073288
35	26.0	-0.535388	0.865268	1.351076	0.147575	0.433680	0.086983	0.693039	0.179742	-0.285642	...	0.049526	0.206537	-0.187108	0.000753	0.098117	-0.553471	-0.078306
113	74.0	1.038370	0.127486	0.184456	1.109950	0.441699	0.945283	-0.036715	0.350995	0.118950	...	0.102520	0.605089	0.023092	-0.626463	0.479120	-0.166937	0.081247
114	74.0	1.038370	0.127486	0.184456	1.109950	0.441699	0.945283	-0.036715	0.350995	0.118950	...	0.102520	0.605089	0.023092	-0.626463	0.479120	-0.166937	0.081247
115	74.0	1.038370	0.127486	0.184456	1.109950	0.441699	0.945283	-0.036715	0.350995	0.118950	...	0.102520	0.605089	0.023092	-0.626463	0.479120	-0.166937	0.081247
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
282987	171288.0	1.912550	-0.455240	-1.750654	0.454324	2.089130	4.160019	-0.881302	1.081750	1.022928	...	-0.524067	-1.337510	0.473943	0.616683	-0.283548	-1.084843	0.073133
283483	171627.0	-1.464380	1.368119	0.815992	-0.601282	-0.689115	-0.487154	-0.303778	0.884953	0.054065	...	0.287217	0.947825	-0.218773	0.082926	0.044127	0.639270	0.213565
283485	171627.0	-1.457978	1.378203	0.811515	-0.603760	-0.711883	-0.471672	-0.282535	0.880654	0.052808	...	0.284205	0.949659	-0.216949	0.083250	0.044944	0.639933	0.219432
284191	172233.0	-2.667936	3.160505	-3.355984	1.007845	-0.377397	-0.109730	-0.667233	2.309700	-1.639306	...	0.391483	0.266536	-0.079853	-0.096395	0.086719	-0.451128	-1.183743
284193	172233.0	-2.691642	3.123168	-3.339407	1.017018	-0.293095	-0.167054	-0.745886	2.325616	-1.634651	...	0.402639	0.259746	-0.086606	-0.097597	0.083693	-0.453584	-1.205466

1081 rows × 31 columns



```
data[113:115]
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
113	74.0	1.03837	0.127486	0.184456	1.10995	0.441699	0.945283	-0.036715	0.350995	0.11895	...	0.10252	0.605089	0.023092	-0.626463	0.47912	-0.166937	0.081247	0.001192	1.18	0
114	74.0	1.03837	0.127486	0.184456	1.10995	0.441699	0.945283	-0.036715	0.350995	0.11895	...	0.10252	0.605089	0.023092	-0.626463	0.47912	-0.166937	0.081247	0.001192	1.18	0

2 rows × 31 columns

```
data.drop_duplicates(inplace=True)
```

```
data.duplicated().sum()
```

```
0
```

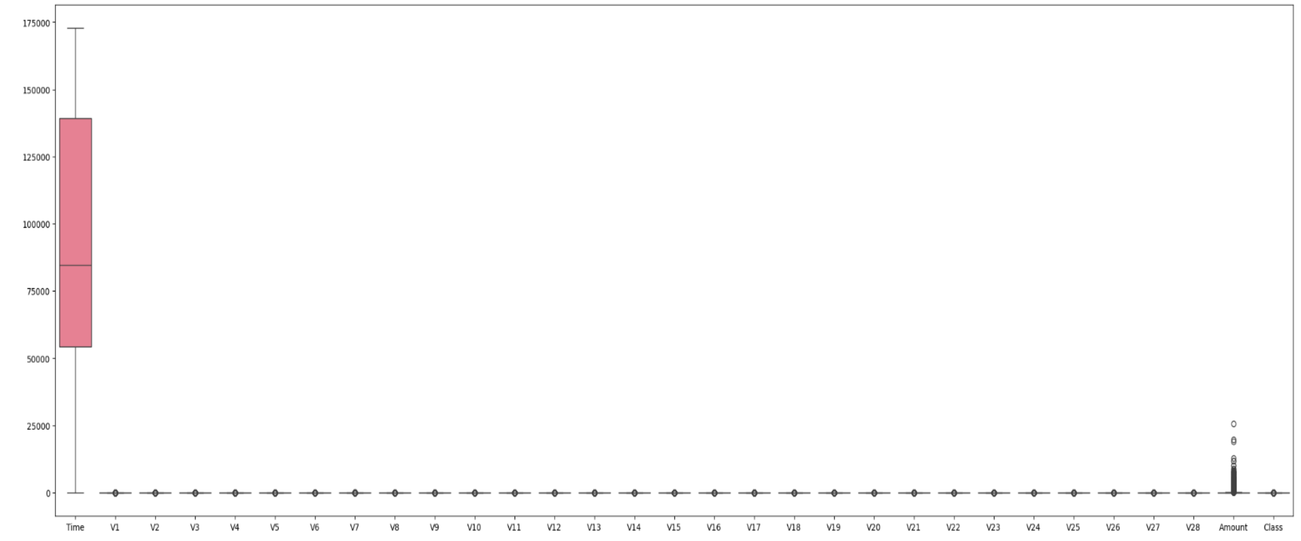
Exploration and Visualization

Exploring the dataset through descriptive statistics and visualizations helps in understanding the distribution of data, identifying outliers, and spotting potential anomalies that could impact the model’s performance.

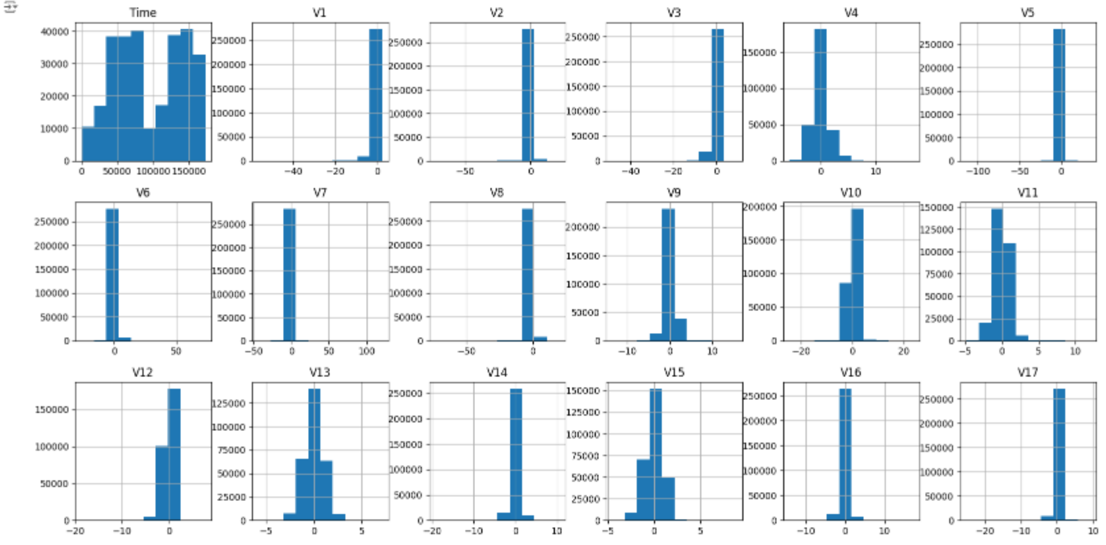
Explore and visualize the dataset to identify any outliers or anomalies.

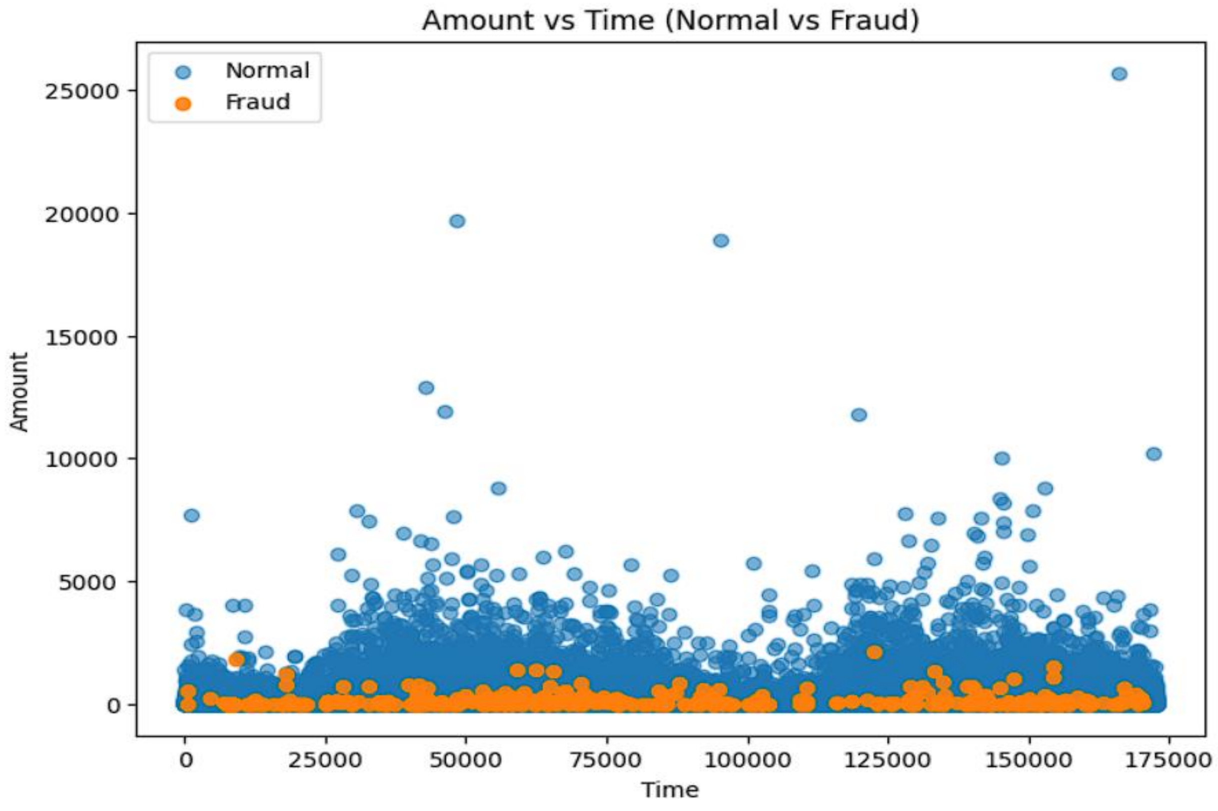
```python
import seaborn as sns
fig = plt.figure(figsize = (30, 10))
sns.boxplot(data)
```

```
<Axes: >
```



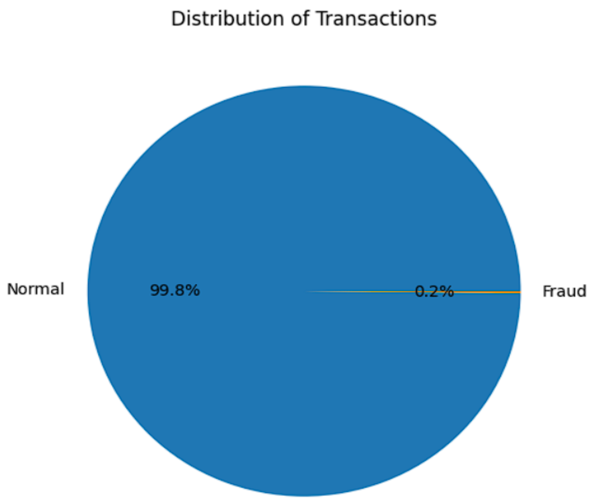
```python
data.hist(figsize=(20,20))
plt.show()
```

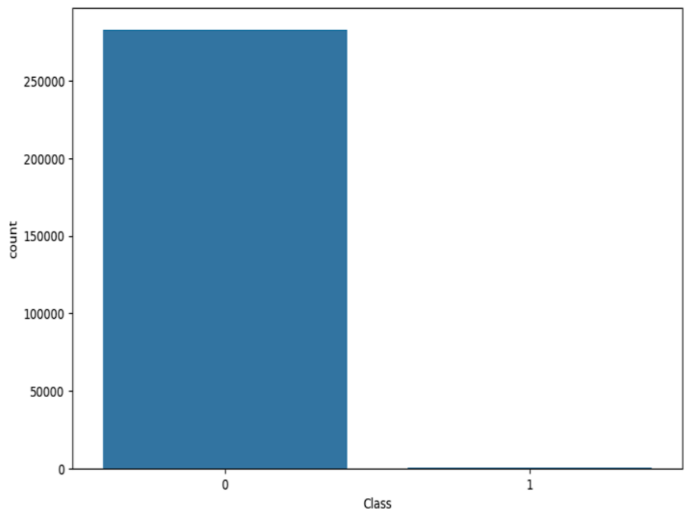




Pie chart for Unbalance Class

Bar chart for Unbalance Class



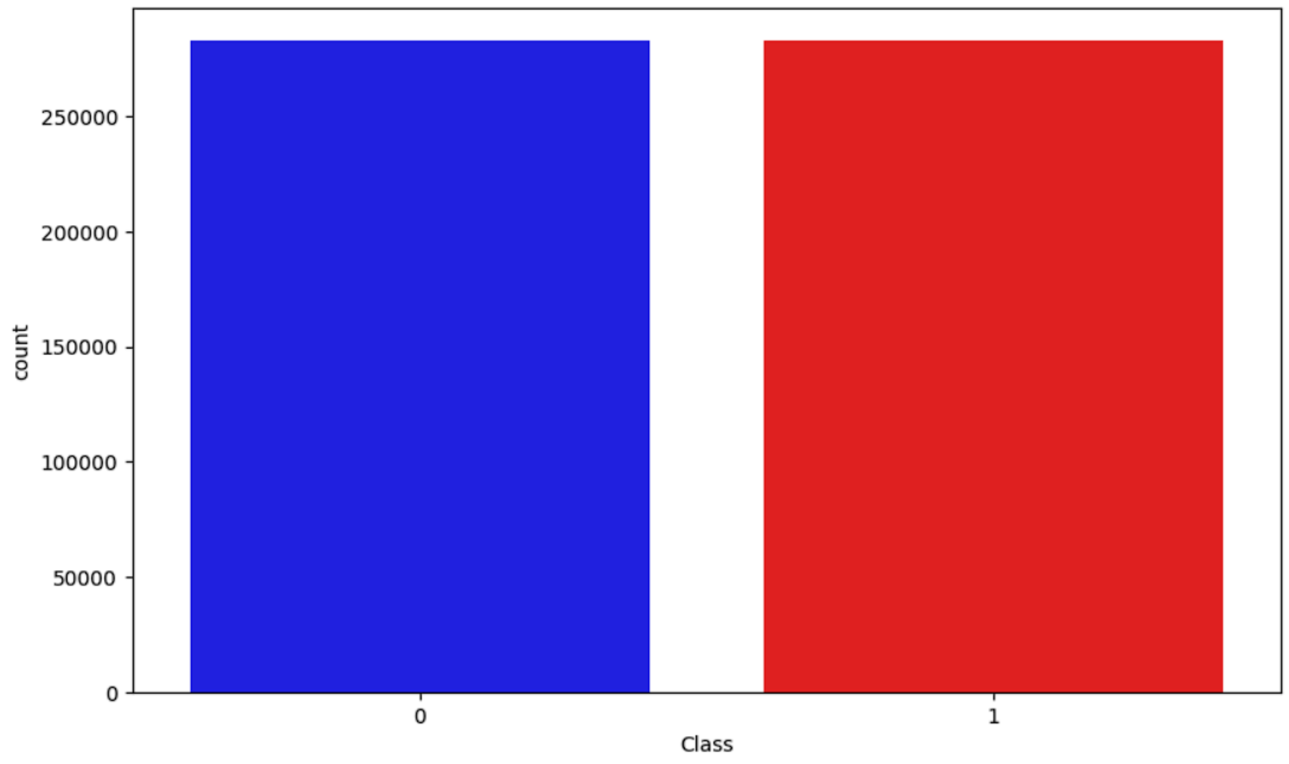


Balancing the Dataset

Credit card fraud detection typically involves imbalanced datasets, where fraudulent transactions are far fewer than legitimate ones. Techniques like undersampling the majority class (non-fraudulent transactions) or oversampling the minority class (fraudulent transactions) or SMOTE can help balance the dataset and improve model performance.

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
x_1, y_1 = smote.fit_resample(x, y)

print(x_1.shape, y_1.shape)
```

```
(566506, 30) (566506,)
```



Feature Selection & Engineering

Selecting relevant features is crucial for model accuracy. Irrelevant features can introduce noise and reduce model performance. This step involves identifying and keeping only those features that contribute meaningfully to predicting fraud.

Feature Engineering: Creating new features that capture underlying patterns in the data can enhance model accuracy. For example, time-based features might capture fraudulent activities at specific times of the day.

```
delta_time = pd.to_timedelta(x_1['Time'], unit='s')
x_1['Time_Day'] = (delta_time.dt.components.days).astype(float)
x_1['Time_Hour'] = (delta_time.dt.components.hours).astype(float)
x_1['Time_Minute'] = (delta_time.dt.components.minutes).astype(float)
x_1.tail(100)
```

By Experimenting with different Time features and their impact on the model's performance below features are dropped

```
x_1.drop(['Time'],axis=1,inplace=True)
x_1.head()
```

Identify relevant features for the model and drop irrelevant features. Normalize the numerical features.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_1['Amount'] = scaler.fit_transform(x_1['Amount'].values.reshape(-1, 1))
```

Model Building

Choosing Logistic Regression: Logistic Regression is selected as the classification algorithm due to its simplicity and effectiveness in binary classification problems like fraud detection. It models the probability that a given input point belongs to a particular class.

Import Necessary Libraries

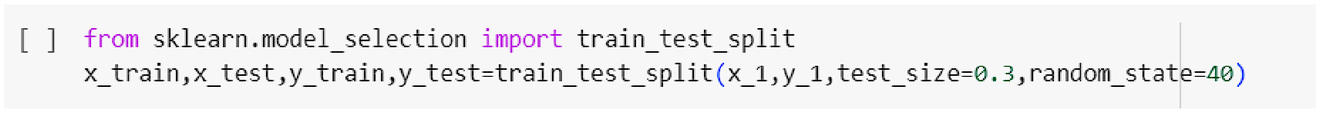
Importing Libraries: Before proceeding, necessary libraries such as LogisticRegression from sklearn.linear_model are imported. This ensures that all the tools required for model building and evaluation are available.

Understand Logistic Regression

Overview of the Algorithm: Logistic Regression estimates the probability of a binary outcome (fraud or no fraud) by fitting data to a logistic curve. It is crucial to understand its parameters, such as the regularization term, for effective model configuration.

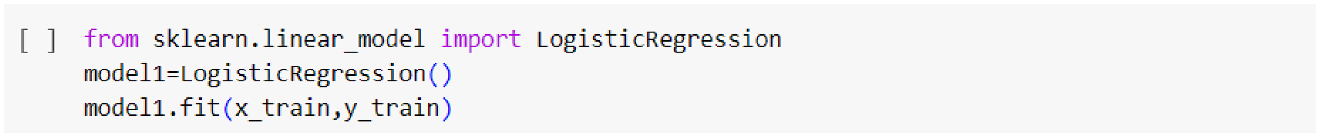
Model Training

Training the Model: The training process involves using the fit method to learn from the training data. This step is critical as it allows the model to understand patterns in the data that distinguish fraudulent from non-fraudulent transactions.

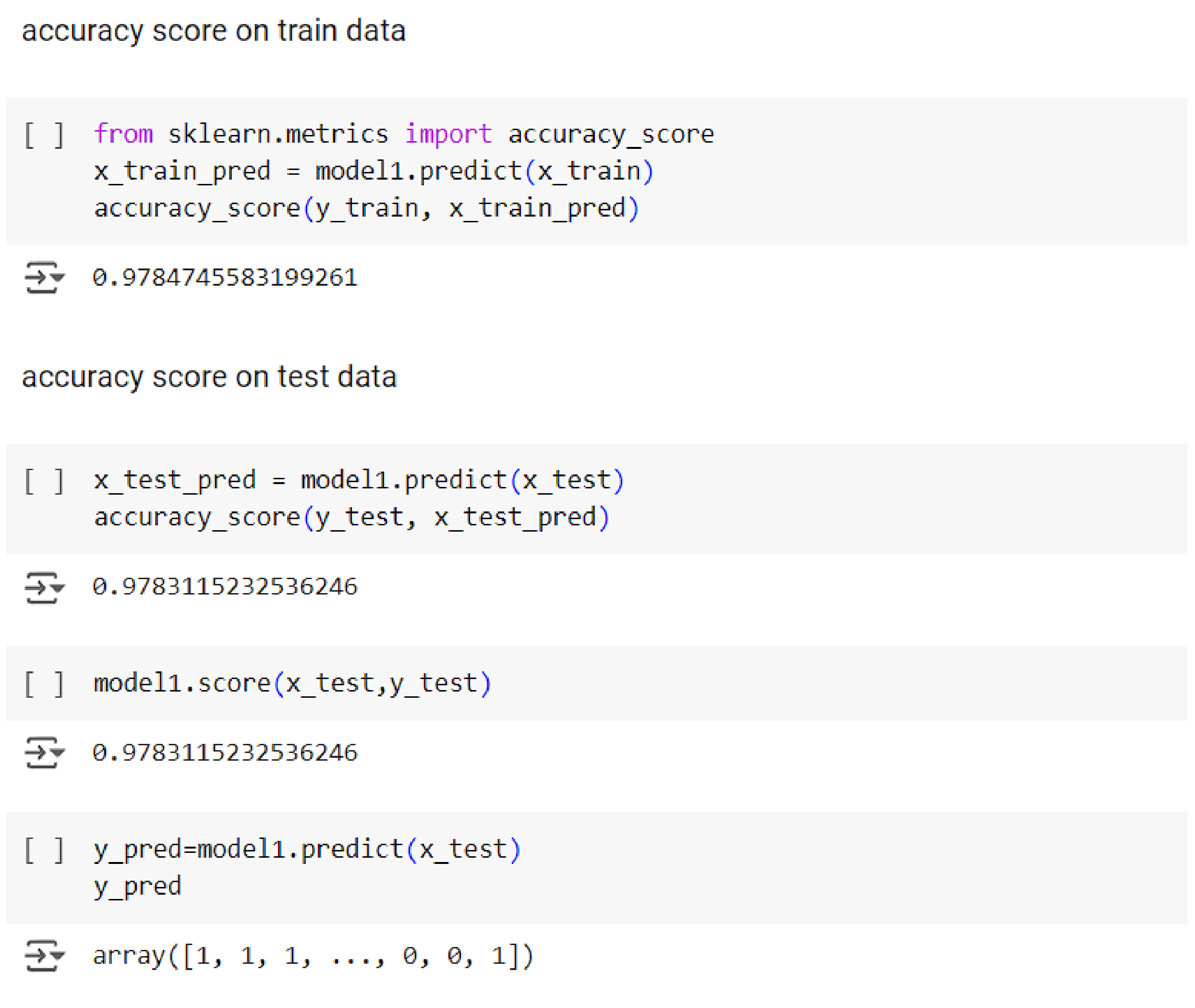


```
[ ] from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test=train_test_split(x_1,y_1,test_size=0.3,random_state=40)
```

LOGISTIC REGRESSION



```
[ ] from sklearn.linear_model import LogisticRegression
    model1=LogisticRegression()
    model1.fit(x_train,y_train)
```



accuracy score on train data

```
[ ] from sklearn.metrics import accuracy_score
    x_train_pred = model1.predict(x_train)
    accuracy_score(y_train, x_train_pred)
```

```
0.9784745583199261
```

accuracy score on test data

```
[ ] x_test_pred = model1.predict(x_test)
    accuracy_score(y_test, x_test_pred)
```

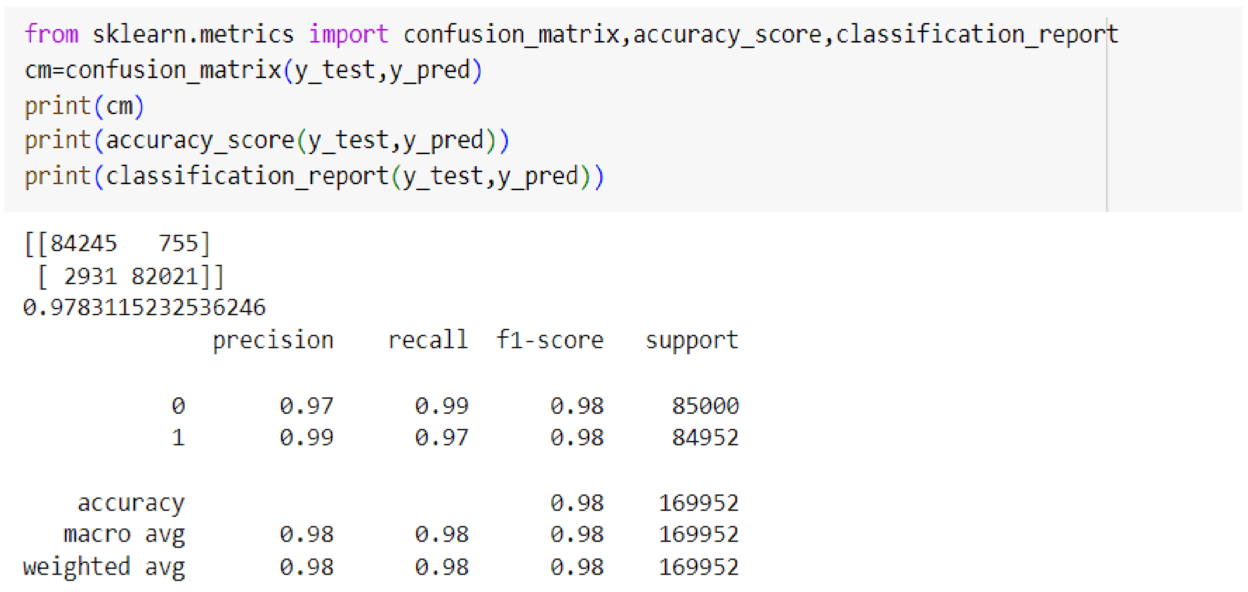
```
0.9783115232536246
```

```
[ ] model1.score(x_test,y_test)
```

```
0.9783115232536246
```

```
[ ] y_pred=model1.predict(x_test)
    y_pred
```

```
array([1, 1, 1, ..., 0, 0, 1])
```

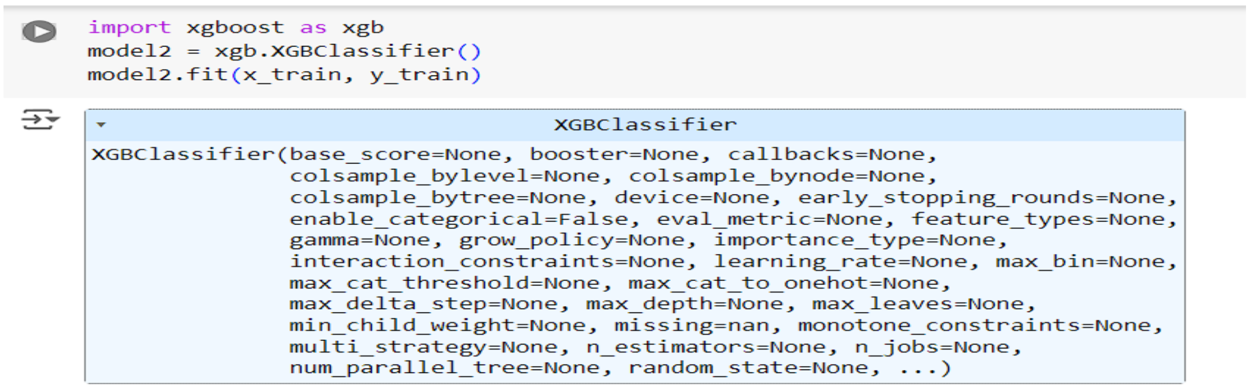


```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
cm=confusion_matrix(y_test,y_pred)
print(cm)
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[84245   755]
 [ 2931 82021]]
0.9783115232536246
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     85000
           1       0.99      0.97      0.98     84952

    accuracy                           0.98    169952
   macro avg       0.98      0.98      0.98    169952
weighted avg       0.98      0.98      0.98    169952
```

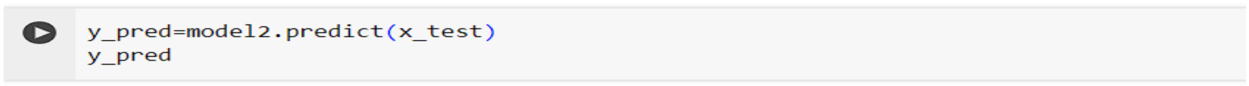
XGBoost

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithm s under the Gradient Boosting framework.

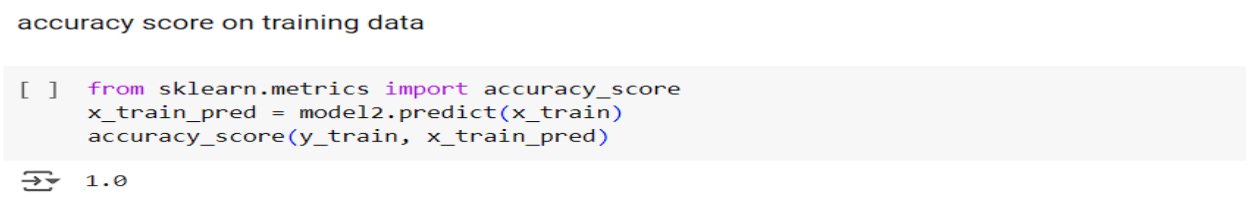


```
import xgboost as xgb
model2 = xgb.XGBClassifier()
model2.fit(x_train, y_train)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```



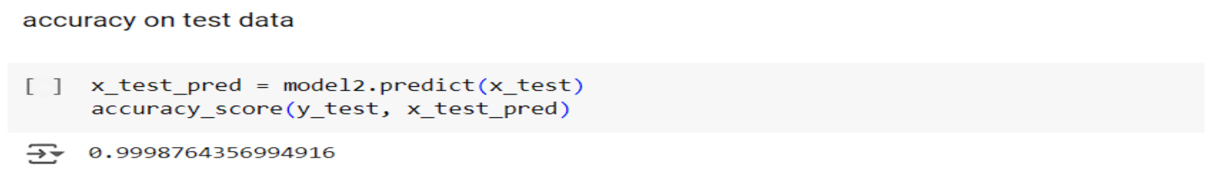
```
y_pred=model2.predict(x_test)
y_pred
```



accuracy score on training data

```
from sklearn.metrics import accuracy_score
x_train_pred = model2.predict(x_train)
accuracy_score(y_train, x_train_pred)
```

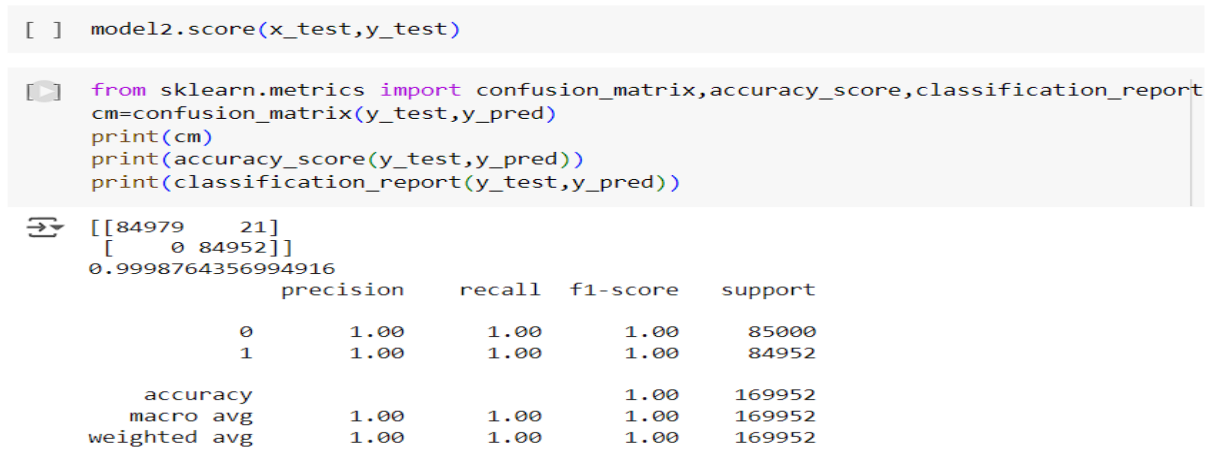
```
1.0
```



accuracy on test data

```
x_test_pred = model2.predict(x_test)
accuracy_score(y_test, x_test_pred)
```

```
0.9998764356994916
```



```
model2.score(x_test,y_test)
```

```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
cm=confusion_matrix(y_test,y_pred)
print(cm)
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[84979    21]
 [    0 84952]]
0.9998764356994916
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85000
           1       1.00      1.00      1.00     84952

    accuracy                           1.00    169952
   macro avg       1.00      1.00      1.00    169952
weighted avg       1.00      1.00      1.00    169952
```

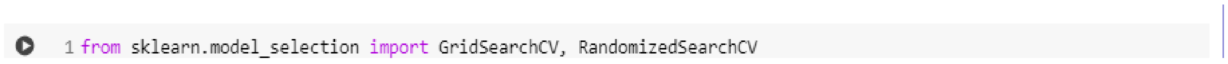
Model Tuning

Hyperparameter Tuning

Using **GridSearchCV or RandomizedSearchCV:** These techniques automate the process of hyperparameter tuning by testing different combinations of hyperparameters and selecting the best-performing model.

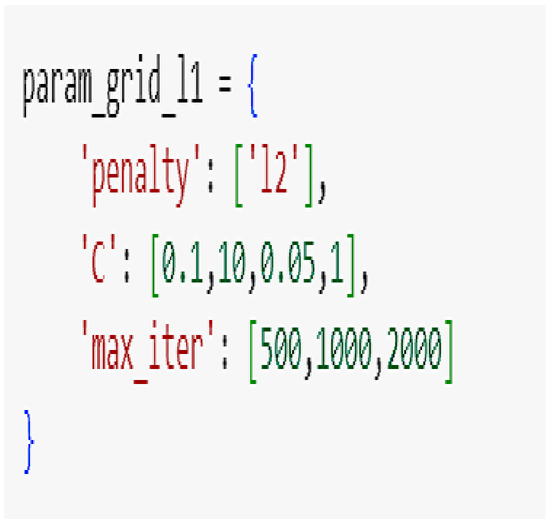
Import Necessary Libraries

Importing Libraries: Libraries such as GridSearchCV and RandomizedSearch CV from sklearn.model_selection are imported to facilitate hyperparameter tuning.

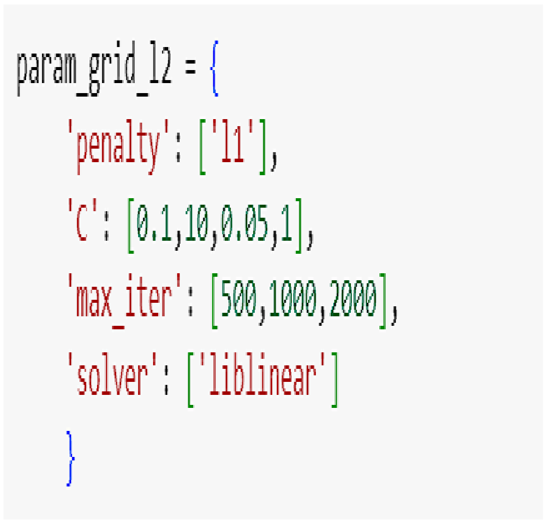


```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

Creating a Hyperparameter Grid: This involves defining the hyperparameter s to be tuned, such as the regularization strength for Logistic Regression, and their respective ranges.



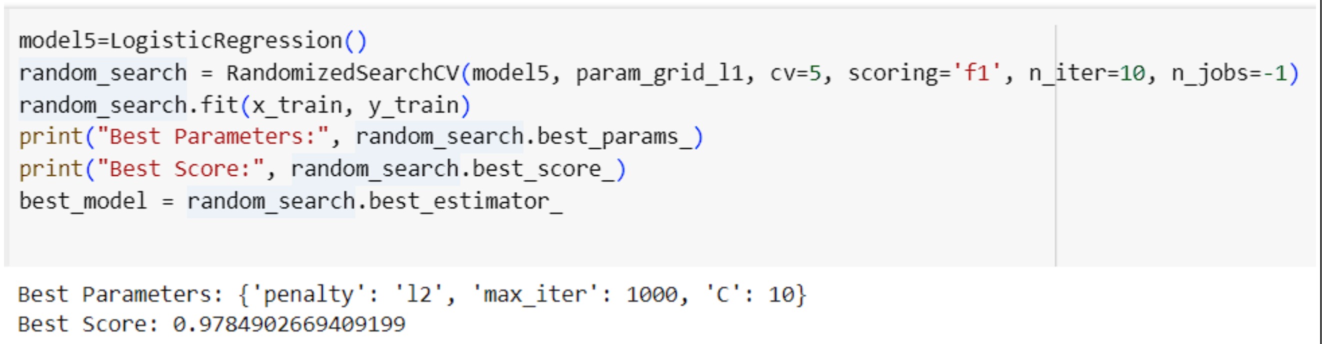
```
param_grid_l1 = {
    'penalty': ['l2'],
    'C': [0.1,10,0.05,1],
    'max_iter': [500,1000,2000]
}
```



```
param_grid_l2 = {
    'penalty': ['l1'],
    'C': [0.1,10,0.05,1],
    'max_iter': [500,1000,2000],
    'solver': ['liblinear']
    }
```

RandomizedSearchCV

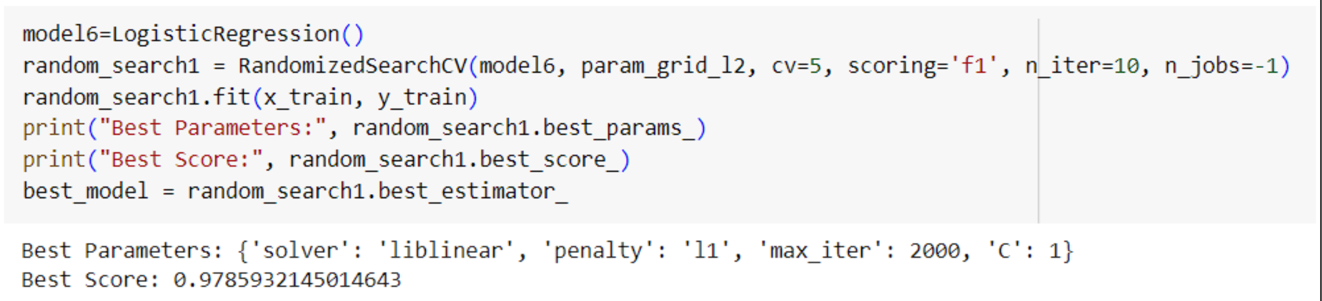
RandomizedSearchCV – Penalty:L2



```
model5=LogisticRegression()
random_search = RandomizedSearchCV(model5, param_grid_l1, cv=5, scoring='f1', n_iter=10, n_jobs=-1)
random_search.fit(x_train, y_train)
print("Best Parameters:", random_search.best_params_)
print("Best Score:", random_search.best_score_)
best_model = random_search.best_estimator_
```

```
Best Parameters: {'penalty': 'l2', 'max_iter': 1000, 'C': 10}
Best Score: 0.9784902669409199
```

RandomizedSearchCV – Penalty:L1

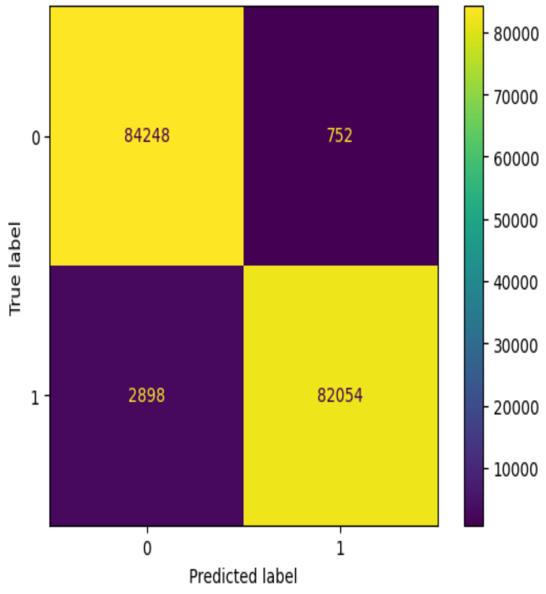


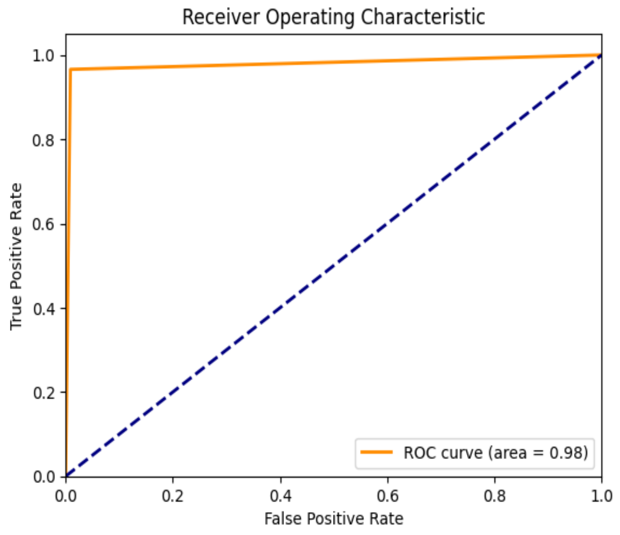
```
model6=LogisticRegression()
random_search1 = RandomizedSearchCV(model6, param_grid_l2, cv=5, scoring='f1', n_iter=10, n_jobs=-1)
random_search1.fit(x_train, y_train)
print("Best Parameters:", random_search1.best_params_)
print("Best Score:", random_search1.best_score_)
best_model = random_search1.best_estimator_
```

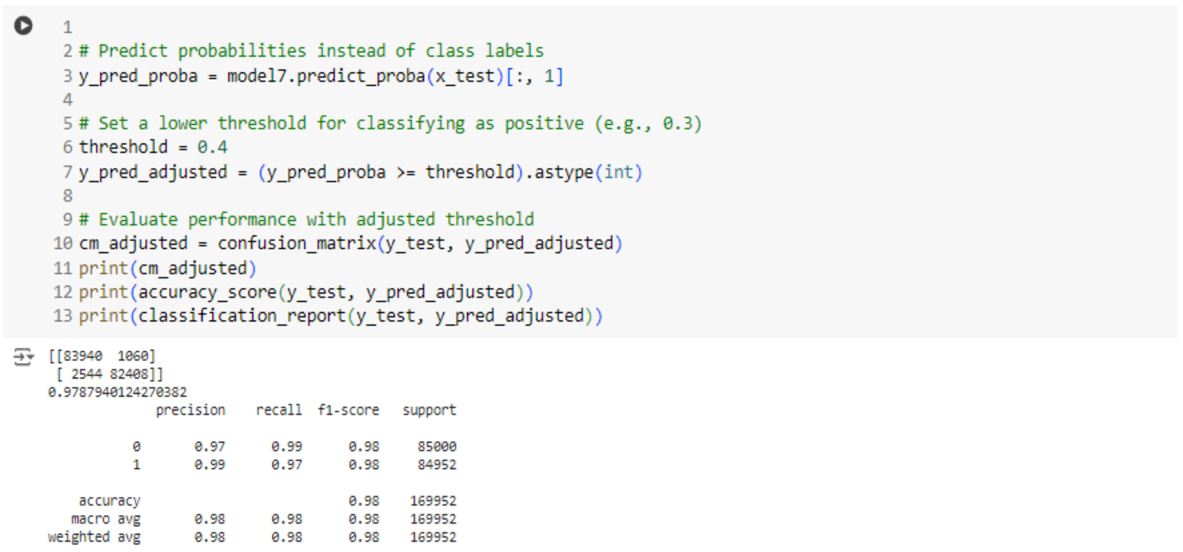
```
Best Parameters: {'solver': 'liblinear', 'penalty': 'l1', 'max_iter': 2000, 'C': 1}
Best Score: 0.9785932145014643
```

Model Evaluation

Logistic Regression : Confusion Matrix and ROC-AUC curve for Logistic Regression.



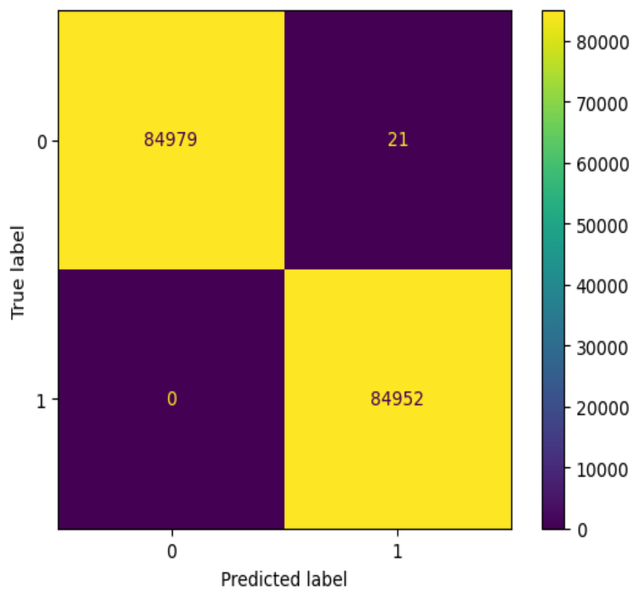


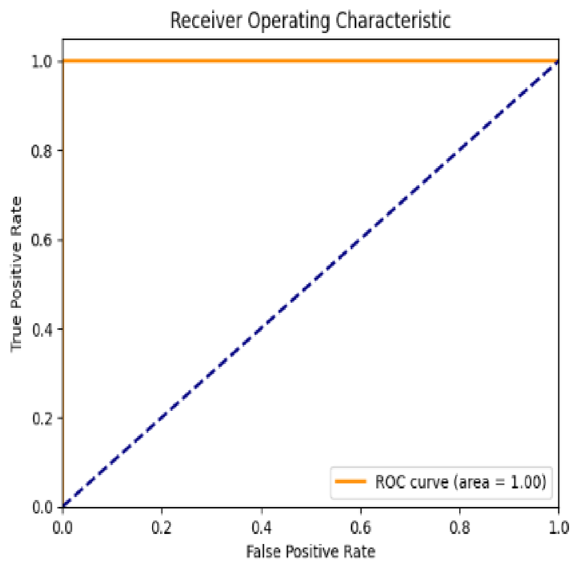


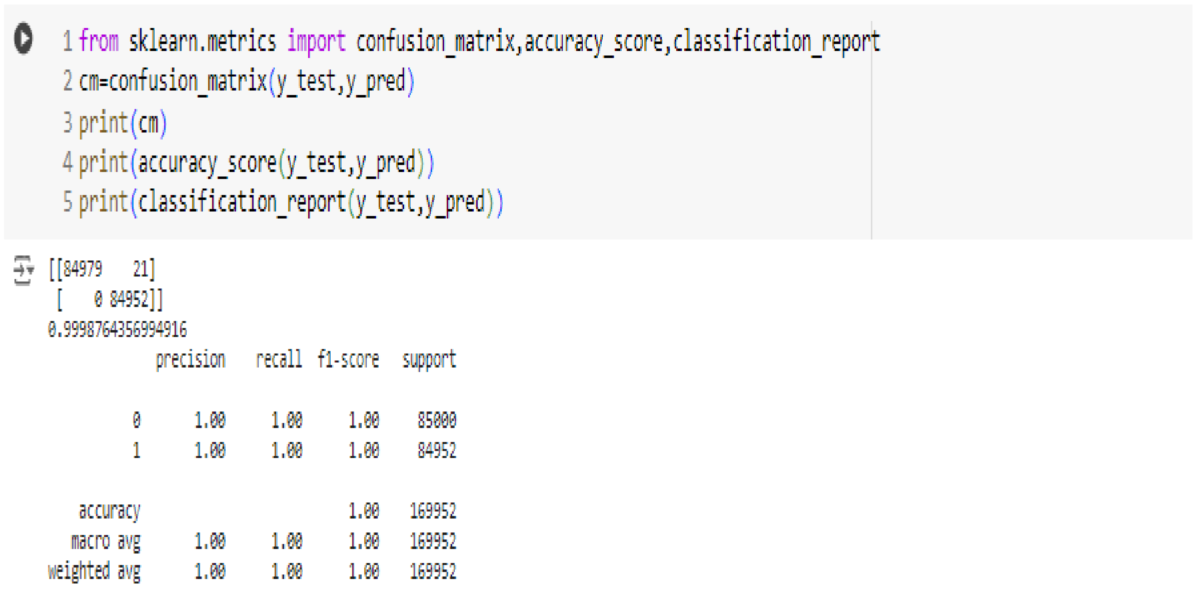
```

# Predict probabilities instead of class labels
y_pred_proba = model7.predict_proba(x_test)[:, 1]

# Set a lower threshold for classifying as positive (e.g., 0.3)
threshold = 0.4
y_pred_adjusted = (y_pred_proba >= threshold).astype(int)

# Evaluate performance with adjusted threshold
cm_adjusted = confusion_matrix(y_test, y_pred_adjusted)
print(cm_adjusted)
print(accuracy_score(y_test, y_pred_adjusted))
print(classification_report(y_test, y_pred_adjusted))
```

```
[[83940  1060]
 [ 2544 82408]]
0.9787940124270382
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     85000
           1       0.99      0.97      0.98     84952

    accuracy                           0.98    169952
   macro avg       0.98      0.98      0.98    169952
weighted avg       0.98      0.98      0.98    169952
```

XGBoost: Confusion Matrix and ROC-AUC curve for XGBoost.







```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
cm=confusion_matrix(y_test,y_pred)
print(cm)
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[84979    21]
 [    0 84952]]
0.9998764356994916
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85000
           1       1.00      1.00      1.00     84952

    accuracy                           1.00    169952
   macro avg       1.00      1.00      1.00    169952
weighted avg       1.00      1.00      1.00    169952
```

Conclusion

From the above model it was observed that

1. While using Smote Oversampling Technique with Logistic Regression model is giving recall 0.97 for fraud class and 0.99 for normal class with accuracy 0.9791.
2. After hyperparameter tuning model is giving recall 0.97 for fraud class and 0.99 for normal class with accuracy 0.9793, which is slightly better.
3. On the other side after using XGBoost model is giving recall and f1 score for both normal and fraud class was 1 with accuracy 0.9998.
4. It is concluded that while both Logistic Regression and XGBoost models are commonly used for Credit Card Fraud Detection, the XGBoost model outperforms Logistic Regression in terms of accuracy, recall, f1score and handling complex patterns in the data, making it the more effective choice for this application.

Future Scope

There is always space for improvement in a project. Owing to the sensitivity of this project, different techniques can be integrated as components and their results combined to increase the quality of the final output. This model can be improved even further by adding more algorithms. These algorithms' result, however, must be from the same structure as most of the others. The modules are simple to add once that criteria are met, as seen in the script. It is beneficial. As a result, we can employ a range of ways to address this issue in the project's ongoing growth.

References

1]. **Aleskerov, Emin & Freisleben, Bernd & Rao, Bharat. (1997). CARDWATCH:** A Logistic Regression with hyperparameter tunning along with GridSearchCV. IEEE/IAFE Conference on Computational Intelligence for Financial Engineering, Proceedings (CIFEr). 220 - 226. 10.1109/CIFER.1997.618940.

2]. **Samaneh Sorournejad, Zahra Zojaji, Reza Ebrahimi Atani, Amir Hassan Monadjemi,** A Survey of credit card fraud detection techniques: Data and techniques-oriented perspective.

3]. **Yashvi Jain, Namrata Tiwari, Shripriya Dubey, Sarika Jain,** A Comparative Analysis of Various Credit Card Fraud Detection Techniques, Blue Eyes Intelligence Engineering and Sciences Publications 2019