**Credit Card Fraud Detection**

**PHASE 4: DEVELOPMENT PART 2**

TOPIC: Building the Credit Card Fraud Detection project by performing Feature Engineering, Model training, Evaluation.



* **FEATURE ENGINEERING:**

Feature engineering is a critical aspect of credit card fraud detection as it involves creating relevant input variables (features) for machine learning models to effectively identify fraudulent transactions. The right set of features can significantly improve the model's accuracy. Here are some feature engineering techniques and features commonly used in credit card fraud detection:

* **Transaction Amount:** Log transformation: Take the logarithm of the transaction amount to account for the long-tailed distribution of transaction amounts.
* **Time of Day:** Extract the hour of the day from the transaction timestamp to capture potential patterns in the time of fraudulent activities.
* **Transaction Frequency:** Calculate the number of transactions made by a user within a certain time window.
* **Geographical Features:** Location-based features like the distance between the user's current location and the transaction location.
* **Aggregated Statistics:** Compute statistics like mean, median, standard deviation, and percentiles of transaction amounts for a user within a specific time frame.
* **Categorical Variables:** Encode categorical variables, such as merchant category codes (MCC), transaction type, and card type, using techniques like one-hot encoding or label encoding.
* **Velocity Filters:** Create features that capture unusual transaction velocity, like the number of transactions in the last hour or the time elapsed since the last transaction.
* **Cardholder Behavior :** Features based on the user's historical behavior, such as the average transaction amount, transaction frequency, and standard deviation of transaction amounts.
* **Anomaly Detection Scores:** Use anomaly detection algorithms like Isolation Forest or One-Class SVM to generate anomaly scores and use these scores as features.
* **Transaction Sequences:** Create sequences of transactions for each user and analyze patterns or deviations from their typical transaction sequences.

**EXPLANATION OF DATASET:**

* Credit card fraud detection is the collective term for the policies, tools, methodologies, and practices that credit card companies and financial institutions take to combat identity fraud and stop fraudulent transactions.
* 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.
* It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data.
* Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'.
* Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset.
* The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning.
* Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.
* The dataset will contain various features or attributes related to each transaction.
* Transaction amount
* Transaction date and time
* Merchant information
* Customer account details
* Location information
* Card type
* Transaction category

**GIVEN DATASET:**

the given dataset for credit card fraud detection :

<https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>

* Data cleaning:

**Data cleaning** is one of the important parts of machine learning. It plays a significant part in building a model. It surely isn’t the fanciest part of machine learning and at the same time, there aren’t any hidden tricks or secrets to uncover. However, the success or failure of a project relies on proper data cleaning. Professional data scientists usually invest a very large portion of their time in this step because of the belief that **“Better data beats fancier algorithms”**.

Trainandtest **:**

Train and Test is a method to measure the accuracy of your model. It is called Train and Test because you split the data and a testing set.

Where 80% is training set and 20% is testing. Where we trained the model using the given dataset.

Example:

train\_df=pd.read\_csv('/kaggle/input/playground-series-s3e4/train.csv')

train\_df=pd.concat([train\_df,original\_df])

train\_df=train\_df.sample(frac=1)

train\_df=train\_df.reset\_index(drop=True)

y\_train=train\_df['Class']

y\_train\_original=original\_df['Class']

train\_df.head()

| id | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | ... | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | NaN | 159699.0 | 1.782149 | -0.540553 | -0.428887 | 1.212215 | -0.561719 | 0.024703 | -0.594129 | 0.190593 | ... | 0.289661 | 0.722221 | 0.042611 | -0.447415 | -0.167928 | -0.558828 | 0.032018 | -0.033765 | 82.00 | 0 |
| 1 | NaN | 160249.0 | 0.333382 | 0.242974 | -1.874050 | -1.384971 | 2.727901 | 3.281084 | 0.120879 | 0.522067 | ... | 0.472621 | 1.318697 | -0.302447 | 0.805499 | -0.238469 | 0.456635 | 0.086509 | 0.158196 | 16.00 | 0 |
| 2 | 17616.0 | 27150.0 | -0.686824 | 0.888828 | 1.160781 | -1.046413 | 0.056778 | -1.110635 | 1.049166 | -0.286959 | ... | -0.327845 | -0.644961 | -0.052440 | 0.416036 | -0.095789 | 0.674044 | 0.247373 | 0.040753 | 32.70 | 0 |
| 3 | 7434.0 | 5498.0 | -0.496816 | 0.575539 | 1.299960 | 0.363437 | 0.908640 | 1.143529 | 0.039475 | 0.483578 | ... | 0.067994 | 0.677545 | -0.444902 | -0.991666 | 0.903282 | -0.032666 | -0.066349 | -0.032767 | 6.99 | 0 |
| 4 | 125868.0 | 67756.0 | 1.160957 | 1.265621 | -1.576473 | 1.472988 | 1.162173 | -1.013532 | 0.658133 | -0.152624 | ... | -0.120365 | -0.204997 | -0.203189 | -0.282172 | 0.785278 | -0.284958 | 0.058399 | 0.085436 | 2.69 | 0 |

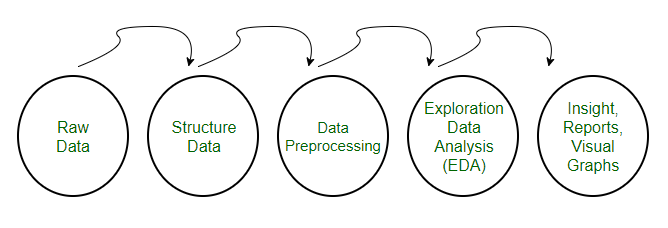
* **MODEL TRAINING:**

Train the selected models on the training data using appropriate performance metrics such as precision, recall, F1-score, and area under the ROC curve (AUC-ROC).

- Use techniques like oversampling or under sampling to address class imbalance issues (fraudulent transactions are often rare compared to legitimate ones).

**1. DATA PREPROCESSING:**

Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.



* **Need of Data Preprocessing**

1. For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.
2. Another aspect is that the data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithm are executed in one data set, and best out of them is chosen.
3. **IMPORT LIBRARIES:**

importnumpyasnp

import pandas aspd

importmatplotlib.pyplotasplt

importseabornassns

fromimblearn.over\_samplingimportSMOTE

from collections importCounter

fromsklearn.model\_selectionimporttrain\_test\_split

fromsklearn.linear\_modelimportLogisticRegression

fromsklearn.metricsimportconfusion\_matrix, classification\_report

from tabulate importtabulate

**2.IMPORT THE DATASET:**

credit\_card=pd.read\_csv("/kaggle/input/creditcardfraud/creditcard.csv")

**3.EXPLORATORY DATA ANALYSIS:**

rows, columns=credit\_card.shape

print( f"Number of Rows: **{**rows**}**")

print(f"Number of Columns: **{**columns**}**")

OUTPUT:

Number of Rows: 284807

Number of Columns: 31

credit\_card.head()

OUTPUT:

|  | 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 |

5ROWS\*31COLUMNS

credit\_card.tail()

OUTPUT:

|  | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |

5 ROWS\*31COLUMNS

credit\_card.info()

OUTPUT:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 284807 entries, 0 to 284806

Data columns (total 31 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Time 284807 non-null float64

1 V1 284807 non-null float64

2 V2 284807 non-null float64

3 V3 284807 non-null float64

4 V4 284807 non-null float64

5 V5 284807 non-null float64

6 V6 284807 non-null float64

7 V7 284807 non-null float64

8 V8 284807 non-null float64

9 V9 284807 non-null float64

10 V10 284807 non-null float64

11 V11 284807 non-null float64

12 V12 284807 non-null float64

13 V13 284807 non-null float64

14 V14 284807 non-null float64

15 V15 284807 non-null float64

16 V16 284807 non-null float64

17 V17 284807 non-null float64

18 V18 284807 non-null float64

19 V19 284807 non-null float64

20 V20 284807 non-null float64

21 V21 284807 non-null float64

22 V22 284807 non-null float64

23 V23 284807 non-null float64

24 V24 284807 non-null float64

25 V25 284807 non-null float64

26 V26 284807 non-null float64

27 V27 284807 non-null float64

28 V28 284807 non-null float64

29 Amount 284807 non-null float64

30 Class 284807 non-null int64

dtypes: float64(30), int64(1)

memory usage: 67.4 MB

credit\_card.isnull()

OUTPUT:

| Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | False | False | False | False |
| 1 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | False | False | False | False |
| 2 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | False | False | False | False |
| 3 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | False | False | False | False |
| 4 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | False | False | False | False |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 284802 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | False | False | False | False |
| 284803 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | False | False | False | False |
| 284804 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | False | False | False | False |
| 284805 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | False | False | False | False |
| 284806 | False | False | False | False | False | False | False | False | False | False | ... | False | False | False | False | False | False | False | False | False | False |

284807rows × 31 columns

credit\_card.duplicated()

OUTPUT:

0 False

1 False

2 False

3 False

4 False

...

284802 False

284803 False

284804 False

284805 False

284806 False

Length: 284807, dtype: bool

credit\_card.duplicated().any()

OUTPUT:

True

credit\_card.shape

OUTPUT:

(284807, 31)

*# Remove duplication*

data=credit\_card.drop\_duplicates()

print(f"Dublicate Transaction:",284807-283726)

OUTPUT:

Dublicate Transaction: 1081

Data

OUTPUT:

|  | 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 |

283726ROWS ×31COLUMNS

data.columns

OUTPUT:

Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',

'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',

'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',

'Class'],

dtype='object')

data.duplicated().any()

OUTPUT:

False

data.shape

OUTPUT:

(283726, 31)

credit\_card['Class'].value\_counts()

OUTPUT:

Class

0 284315

1 492

Name: count, dtype: int64

fraud,legitimate=credit\_card['Class'].value\_counts()

print(f"Fraud Transaction(0): **{**fraud**}**")

print(f"Legitimate Transaction (1): **{**legitimate**}**")

print(f"This is imbalance data")

OUTPUT:

Fraud Transaction(0): 284315

Legitimate Transaction (1): 492

This is imbalnce data

fraud=credit\_card[credit\_card['Class']==0]legitimate=credit\_card[credit\_card['Class']==1]

fraud.value\_counts()

OUTPUT:

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

163152.0 -1.196037 1.585949 2.883976 3.378471 1.511706 3.717077 0.585362 -0.156001 0.122648 4.217934 1.385525 -0.709405 -0.256168 -1.564352 1.693218 -0.785210 -0.228008 -0.412833 0.234834 1.375790 -0.370294 0.524395 -0.355170 -0.869790 -0.133198 0.327804 -0.035702 -0.858197 7.56 0 18

-1.203617 1.574009 2.889277 3.381404 1.538663 3.698747 0.560211 -0.150911 0.124136 4.220998 1.384569 -0.706897 -0.256274 -1.562583 1.692915 -0.787338 -0.226776 -0.412354 0.234322 1.385597 -0.366727 0.522223 -0.357329 -0.870174 -0.134166 0.327019 -0.042648 -0.855262 1.51 0 18

170731.0 2.033492 0.766969 -2.107555 3.631952 1.348594 -0.499907 0.945159 -0.286392 -1.370581 1.653073 -1.600434 -1.510901 -2.143280 1.189850 -0.875588 0.175808 -0.419433 -0.464717 -1.414528 -0.430560 0.241894 0.658545 -0.102644 0.580535 0.643637 0.347240 -0.116618 -0.078601 0.76 0 9

43153.0 -2.086016 2.203265 1.654339 2.941050 -1.683045 0.529728 -1.352162 1.793449 -0.723686 0.600365 -0.982212 -0.551636 -1.337000 0.834403 1.251862 0.033455 1.067978 0.160510 0.213087 0.079002 0.216444 0.567241 -0.035345 0.370201 0.157378 0.440341 0.210230 0.090558 0.76 0 9

68780.0 0.491469 -0.917547 1.675188 3.001976 -1.464269 0.749245 -0.702555 0.384311 0.512069 0.338069 -0.709212 -0.268987 -1.020228 -0.303879 0.313640 0.541331 -0.068891 -0.114026 -1.643937 0.306224 0.370712 0.607597 -0.221228 0.405781 0.080558 0.148798 0.022320 0.086863 281.47 0 5

..

65179.0 1.211936 0.704387 -0.433427 0.888685 0.246402 -1.146268 0.421741 -0.268994 -0.375879 -0.792653 0.912603 0.691485 1.041027 -1.489310 1.063904 0.106578 1.364831 -0.437407 -0.855633 -0.035289 -0.090995 -0.112559 -0.028042 0.321373 0.495229 0.374352 -0.005887 0.045361 0.76 0 1

65180.0 1.115728 -1.075991 0.633478 -0.655357 -1.068740 0.524206 -1.048665 0.367669 -0.460080 0.611242 1.440991 -0.208944 -1.136439 -0.004827 0.076652 0.618923 0.812957 -1.608246 0.188090 0.076327 0.302172 0.742218 -0.071528 -0.277770 0.265391 -0.102726 0.036093 0.011602 80.00 0 1

1.240745 0.776088 -0.313806 0.984619 -0.019049 -1.728700 0.520610 -0.434773 -0.423467 -0.756044 0.917694 0.804933 1.270221 -1.554132 0.769960 0.327372 1.147404 -0.085885 -0.576641 0.005514 -0.102239 -0.168868 -0.035015 0.845781 0.548841 0.331043 -0.020495 0.051391 0.76 0 1

65181.0 -6.968622 1.552543 -1.471983 1.509984 -4.907883 2.324274 -4.959692 -1.745430 -1.017672 -0.596870 -0.618192 2.601800 0.030912 1.676190 -1.985493 -1.380569 2.485633 1.500515 -0.060186 0.574921 -2.901364 0.292703 -0.591841 0.293393 0.166165 -0.132610 0.176943 -0.280242 150.00 0 1

172792.0 -0.533413 -0.189733 0.703337 -0.506271 -0.012546 -0.649617 1.577006 -0.414650 0.486180 -0.915427 -1.040458 -0.031513 -0.188093 -0.084316 0.041333 -0.302620 -0.660377 0.167430 -0.256117 0.382948 0.261057 0.643078 0.376777 0.008797 -0.473649 -0.818267 -0.002415 0.013649 217.00 0 1

Name: count, Length: 283253, dtype: int64

legitimate.value\_counts()

OUTPUT:

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

68207.0 -13.192671 12.785971 -9.906650 3.320337 -4.801176 5.760059 -18.750889 -37.353443 -0.391540 -5.052502 4.406806 -4.610756 -1.909488 -9.072711 -0.226074 -6.211557 -6.248145 -3.149247 0.051576 -3.493050 27.202839 -8.887017 5.303607 -0.639435 0.263203 -0.108877 1.269566 0.939407 1.00 1 6

94362.0 -26.457745 16.497472 -30.177317 8.904157 -17.892600 -1.227904 -31.197329 -11.438920 -9.462573 -22.187089 4.419997 -10.592305 -0.703796 -3.926207 -2.400246 -6.809890 -12.462315 -5.501051 -0.567940 2.812241 -8.755698 3.460893 0.896538 0.254836 -0.738097 -0.966564 -7.263482 -1.324884 1.00 1 4

93860.0 -10.850282 6.727466 -16.760583 8.425832 -10.252697 -4.192171 -14.077086 7.168288 -3.683242 -15.239962 8.030708 -16.060306 0.270530 -14.952981 -0.241095 -11.866731 -15.486990 -5.748652 4.130031 -0.646818 2.541637 0.135535 -1.023967 0.406265 0.106593 -0.026232 -1.464630 -0.411682 78.00 1 2

148053.0 1.261324 2.726800 -5.435019 5.342759 1.447043 -1.442584 -0.898702 0.123062 -2.748496 -3.202436 1.991361 -3.986416 0.577207 -8.485795 -0.794782 -0.666134 -1.372629 -0.104313 -1.466911 0.313332 0.209086 -0.425938 -0.154440 -0.018820 0.632234 0.192922 0.468181 0.280486 1.59 1 2

84204.0 -1.927453 1.827621 -7.019495 5.348303 -2.739188 -2.107219 -5.015848 1.205868 -4.382713 -8.337707 7.190306 -9.424844 -0.223293 -12.875494 -0.071918 -6.299961 -12.719207 -3.740176 0.844060 2.172709 1.376938 -0.792017 -0.771414 -0.379574 0.718717 1.111151 1.277707 0.819081 512.25 1 2

..

46057.0 -1.309441 1.786495 -1.371070 1.214335 -0.336642 -1.390120 -1.709109 0.667748 -1.699809 -3.843911 2.962599 -3.956045 -1.539232 -4.634631 -0.248403 -2.058551 -5.635494 -0.775271 -0.239310 0.253464 0.533521 -0.022180 -0.299556 -0.226416 0.364360 -0.475102 0.571426 0.293426 1.00 1 1

45541.0 -1.519244 2.308492 -1.503599 2.064101 -1.000845 -1.016897 -2.059731 -0.275166 -1.562206 -2.755797 3.438248 -3.521529 -0.918761 -4.452100 0.499314 -2.907903 -5.248646 -0.936815 1.160120 0.175019 1.307871 0.102826 -0.017746 0.149696 -0.096602 -0.369115 -0.019244 -0.208319 1.00 1 1

45501.0 1.001992 0.047938 -0.349002 1.493958 0.186939 0.190966 -0.001112 0.147140 0.580415 -0.792938 -0.984172 -0.567380 -1.105592 -1.381214 0.405490 0.279890 1.132160 0.092993 -0.298920 0.016004 -0.334417 -1.014315 -0.128427 -0.946242 0.456090 -0.453206 0.046627 0.064698 105.99 1 1

45463.0 -1.476893 2.122314 -1.229470 1.201849 -0.343264 -1.317704 -1.528142 -0.620953 -1.213040 -2.975267 3.532220 -3.682640 -1.154777 -5.165229 -0.240091 -2.404927 -5.671739 -0.994294 -0.289936 0.276893 1.186036 -0.040215 -0.238930 0.110144 0.045418 -0.569232 0.481019 -0.047555 1.00 1 1

170348.0 1.991976 0.158476 -2.583441 0.408670 1.151147 -0.096695 0.223050 -0.068384 0.577829 -0.888722 0.491140 0.728903 0.380428 -1.948883 -0.832498 0.519436 0.903562 1.197315 0.593509 -0.017652 -0.164350 -0.295135 -0.072173 -0.450261 0.313267 -0.289617 0.002988 -0.015309 42.53 1 1

Name: count, Length: 473, dtype: int64

credit\_card.describe()

OUTPUT:

|  | 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 |

8ROWS ×31COLUMNS

X= data .drop('Class', axis=1)Y=data['Class']

### HANDLING IMBALANCE DATA:

#### I handle imbalanced data I handle imbalanced data using Synthetic Minority Oversampling Technique (SMOTE)

SMOTE algorithm works in 4 simple steps:

1. Choose a minority class as the input vector.
2. Find its k nearest neighbors (k\_neighbors is specified as an argument in the SMOTE() function).
3. Choose one of these neighbors and place a synthetic point anywhere on the line joining the point under consideration and its chosen neighbor.
4. Repeat the steps until the data is balanced.

X.shape

OUTPUT:

(283726, 30)

Y.shape

OUTPUT:

(283726,)

*# You can specify a random state for reproducibility*smote=SMOTE(random\_state=42) x\_smote, y\_smote=smote.fit\_resample(X, Y)

y\_smote.value\_counts()

OUTPUT:

Class

0 283253

1 283253

Name: count, dtype: int64

print('Original dataset shape', Counter(Y))print('Resample dataset shape', Counter(y\_smote))

OUTPUT:

Original dataset shape Counter({0: 283253, 1: 473})

Resample dataset shape Counter({0: 283253, 1: 283253})

5.**TRAINING AND TESTING THE DATASET:**

X\_train, X\_test, Y\_train, Y\_test=train\_test\_split(x\_smote, y\_smote, test\_size=0.20, random\_state=42)

1. **DATA VISUALIZATION:**

fraud\_data=data[data['Class']==0]

legitimate\_data=data[data['Class']==1]

plt.figure(figsize=(8,6))

plt.title("Transaction Amount - Fraud vs. Legitimate")

plt.xlabel("Amount ($)")

plt.ylabel("Frequency")

sns.histplot(fraud\_data['Amount'],bins=50,kde=True,color="gold",label="Fraud")

sns.histplot(legitimate\_data['Amount'],bins=50,kde=True,color="blue",label="Legitimate")

plt.legend()

plt.show()

**OUTPUT:**

fraud\_data=data[data['Class']==0]

legitimate\_data=data[data['Class']==1]

plt.figure(figsize=(8,6))

plt.title("Transaction Amount - Fraud vs. Legitimate")

plt.xlabel("Amount ($)")

plt.ylabel("Frequency")

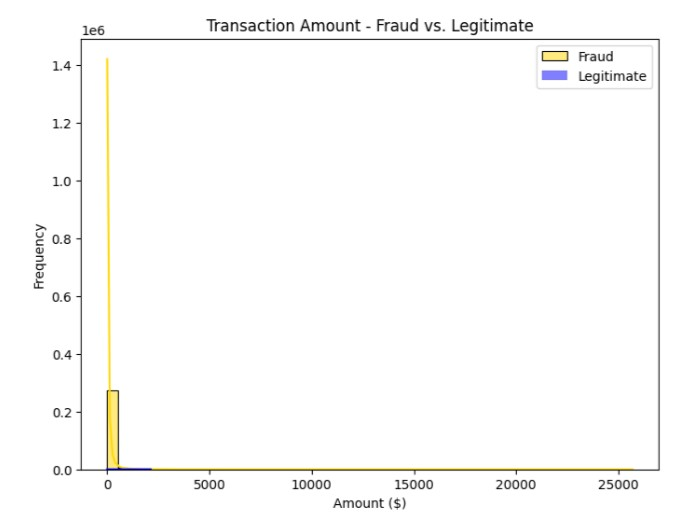
sns.histplot(fraud\_data['Amount'],bins=50,kde=True,color="gold",label="Fraud")

sns.histplot(legitimate\_data['Amount'],bins=50,kde=True,color="blue",label="Legitimate")

plt.legend()

plt.show()

**OUTPUT:**



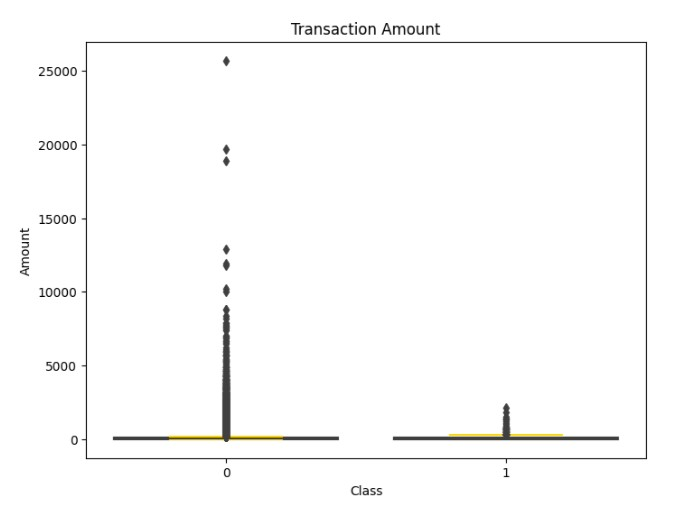
plt.figure(figsize=(8,6))

plt.title("Transaction Amount")

sns.boxplot(x="Class",y="Amount",data=data,boxprops=dict(facecolor="blue"),whiskerprops=dict(color="gold"),capprops=dict(color="gold"))

plt.show()

**OUTPUT:**

****

corr\_imbalanced=credit\_card.corr()

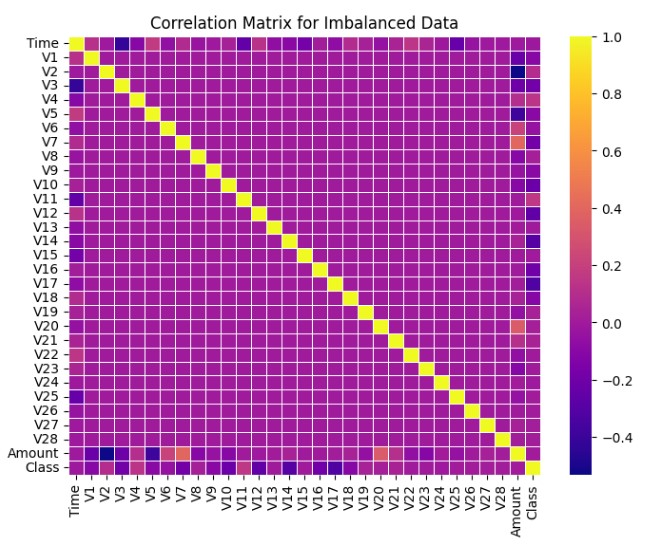
plt.figure(figsize=(8,6))

sns.heatmap(corr\_imbalanced,annot=False,cmap="plasma",linewidths=0.5)

plt.title("Correlation Matrix for Imbalanced Data")

plt.show()

**OUTPUT:**

****

corr\_balanced=x\_smote.corr()

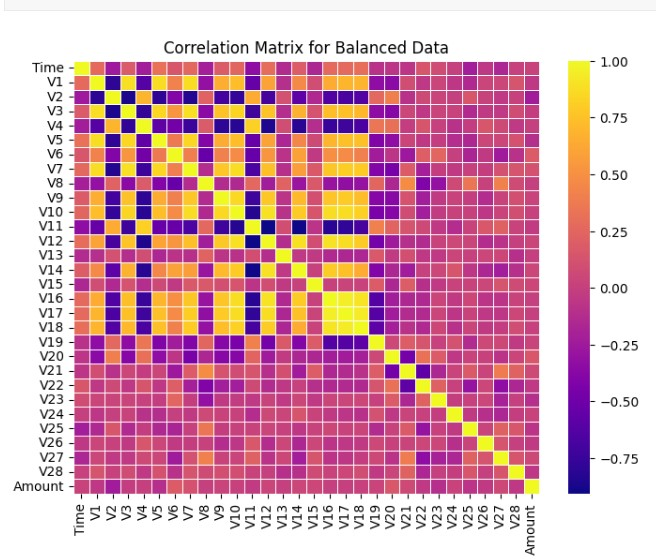
plt.figure(figsize=(8,6))

sns.heatmap(corr\_balanced,annot=False,cmap="plasma",linewidths=0.5)

plt.title("Correlation Matrix for Balanced Data")

plt.show()

**OUTPUT:**

****

custom\_palette=sns.color\_palette(["blue","gold"])

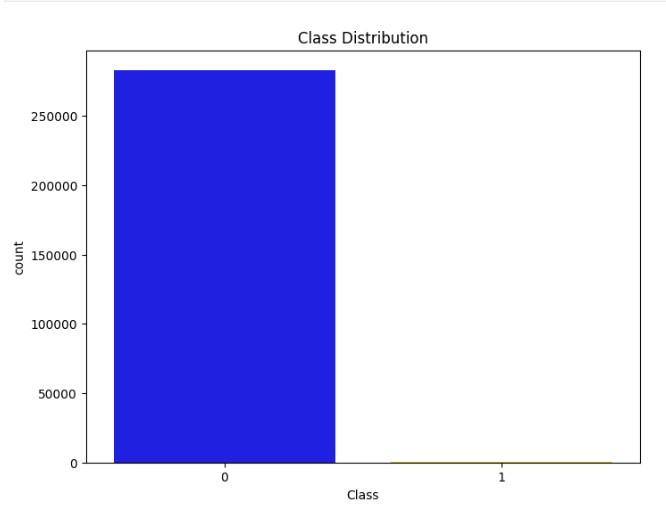
plt.figure(figsize=(8,6))

sns.countplot(x="Class",data=data,palette=custom\_palette)

plt.title("Class Distribution")

plt.show()

**OUTPUT:**

****

custom\_palette=sns.color\_palette(["blue","gold"])

plt.figure(figsize=(8,6))

sns.countplot(x=y\_smote,palette=custom\_palette)

plt.title("Distribution of Classes After Resampling (SMOTE)")

plt.xlabel("Class (0: Fraud, 1: Legitimate)")

plt.ylabel("Count")

plt.show()

**OUTPUT:**

****

1. **BUILDING A REGRESSION MODEL:**

*#Build the Logistic Regression model*

logistic\_model=LogisticRegression(random\_state=42)

logistic\_model.fit(X\_train, Y\_train)

OUTPUT:

/opt/conda/lib/python3.10/site-packages/sklearn/linear\_model/\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

https://scikit-learn.org/stable/modules/preprocessing.html

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression

n\_iter\_i = \_check\_optimize\_result(

LogisticRegression(random\_state=42)

**4.MODEL PREDICTION:**

1. Predict Transactions as fraudulent or genuine by giving Random Sample
2. Predict Transactions as fraudulent or genuine by using my dataset value [23 ,1.322707269, -0.174040833, 0.434555031 ,0.576037652 -0.836758046 -0.831083411 -0.264904961 -0.220981943 -1.071424618 0.868558548 -0.64150629 -0.111315775 0.36148541 0.171945122 0.782166532 -1.35587073 -0.216935153 1.271765385 -1.240621935 -0.522950941 -0.284375572 -0.323357411 -0.037709905 0.347150939 0.559639137 -0.280158166 0.042335258 0.0288223 16 0 ]

Y\_pred\_logistic=logistic\_model.predict(X\_test)

transaction\_data=pd.DataFrame({

'Time': [16],

'V1': [1.322707269],

'V2': [-0.174040833],

'V3': [0.434555031],

'V4': [0.576037652],

'V5': [-0.836758046],

'V6': [-0.831083411],

'V7': [-0.264904961],

'V8': [-0.220981943],

'V9': [-1.071424618],

'V10': [0.868558548],

'V11': [-0.64150629],

'V12': [-0.111315775],

'V13': [0.36148541],

'V14': [0.171945122],

'V15': [0.782166532],

'V16': [-1.35587073],

'V17': [-0.216935153],

'V18': [1.271765385],

'V19': [-1.240621935],

'V20': [-0.522950941],

'V21': [-0.284375572],

'V22': [-0.323357411],

'V23': [-0.037709905],

'V24': [0.347150939],

'V25': [0.559639137],

'V26': [-0.280158166],

'V27': [0.042335258],

'V28': [0.0288223],

'Amount': [0],

'Class': [0] })

new\_predictions=logistic\_model.predict(transaction\_data.drop(columns=['Class']))

ifnew\_predictions[0] ==1:

print("Time: 16**\n**Predict: Fraudulent Transaction")else:

print("Time: 16**\n**Predict: Genuine Transaction")

OUTPUT:

Time: 16

Predict: Fraudulent Transaction

*# Define the minimum and maximum 'Time' values from your training data*

min\_time=data['Time'].min()

max\_time=data['Time'].max()

random\_transactions= []

*# Generate 28 random transactions*

for\_**in**range(28):

new\_transaction= {

'Time': np.random.uniform(min\_time, max\_time),

'Amount': np.random.uniform(0, 500),

'V1': np.random.uniform(-2, 2),

'V2': np.random.uniform(-2, 2),

'V3': np.random.uniform(-2, 2),

'V4': np.random.uniform(-2, 2),

'V5': np.random.uniform(-2, 2),

'V6': np.random.uniform(-2, 2),

'V7': np.random.uniform(-2, 2),

'V8': np.random.uniform(-2, 2),

'V9': np.random.uniform(-2, 2),

'V10': np.random.uniform(-2, 2),

'V11': np.random.uniform(-2, 2),

'V12': np.random.uniform(-2, 2),

'V13': np.random.uniform(-2, 2),

'V14': np.random.uniform(-2, 2),

'V15': np.random.uniform(-2, 2),

'V16': np.random.uniform(-2, 2),

'V17': np.random.uniform(-2, 2),

'V18': np.random.uniform(-2, 2),

'V19': np.random.uniform(-2, 2),

'V20': np.random.uniform(-2, 2),

'V21': np.random.uniform(-2, 2),

'V22': np.random.uniform(-2, 2),

'V23': np.random.uniform(-2, 2),

'V24': np.random.uniform(-2, 2),

'V25': np.random.uniform(-2, 2),

'V26': np.random.uniform(-2, 2),

'V27': np.random.uniform(-2, 2),

'V28': np.random.uniform(-2, 2),

}

random\_transactions.append(new\_transaction)

random\_data=pd.DataFrame(random\_transactions, columns=X\_train.columns)

*# Use the same feature order*

random\_predictions=logistic\_model.predict(random\_data)

fori, prediction**in**enumerate(random\_predictions):

ifprediction==0:

print(f"Transaction**{**i+1**}**: Genuine Transaction")

else:

print(f"Transaction**{**i+1**}**: Fraudulent Transaction")

OUTPUT:

Transaction 1: Genuine Transaction

Transaction 2: Genuine Transaction

Transaction 3: Fraudulent Transaction

Transaction 4: Genuine Transaction

Transaction 5: Genuine Transaction

Transaction 6: Genuine Transaction

Transaction 7: Genuine Transaction

Transaction 8: Genuine Transaction

Transaction 9: Genuine Transaction

Transaction 10: Genuine Transaction

Transaction 11: Genuine Transaction

Transaction 12: Genuine Transaction

Transaction 13: Genuine Transaction

Transaction 14: Genuine Transaction

Transaction 15: Genuine Transaction

Transaction 16: Genuine Transaction

Transaction 17: Fraudulent Transaction

Transaction 18: Genuine Transaction

Transaction 19: Genuine Transaction

Transaction 20: Genuine Transaction

Transaction 21: Fraudulent Transaction

Transaction 22: Genuine Transaction

Transaction 23: Genuine Transaction

Transaction 24: Genuine Transaction

Transaction 25: Genuine Transaction

Transaction 26: Fraudulent Transaction

Transaction 27: Genuine Transaction

Transaction 28: Genuine Transaction

*# Calculate performance metrics for the Logistic Regression model*

confusion\_matrix\_logistic=confusion\_matrix(Y\_test, Y\_pred\_logistic)

classification\_report\_logistic=classification\_report(Y\_test, Y\_pred\_logistic, output\_dict=True)

confusion\_matrix\_df=pd.DataFrame(confusion\_matrix\_logistic,

columns=["Predicted Negative (0)", "Predicted Positive (1)"]

index=["Actual Negative (0)", "Actual Positive (1)"])

classification\_report\_df=pd.DataFrame(classification\_report\_logistic)

print("Logistic Regression Model - Confusion Matrix:")

print(confusion\_matrix\_df)

print("**\n**Logistic Regression Model - Classification Report:")

print(classification\_report\_df)

OUTPUT:

Logistic Regression Model - Confusion Matrix:

Predicted Negative (0) Predicted Positive (1)

Actual Negative (0) 55225 1238

Actual Positive (1) 2120 54719

Logistic Regression Model - Classification Report:

0 1 accuracy macro avg weighted avg

precision 0.963031 0.977876 0.970362 0.970453 0.970478

recall 0.978074 0.962702 0.970362 0.970388 0.970362

f1-score 0.970494 0.970229 0.970362 0.970362 0.970361

support 56463.000000 56839.000000 0.970362 113302.000000 113302.000000

* **EVALUATION:**
* **Accuracy:** Measures the proportion of correctly predicted instances out of the total instances.
* **Precision:** It is a measure of a classifier’s exactness. Low precision indicates a high number of false positives.
* **Recall(Sensitivity):** It is a measure of a classifier’s completeness. Low recall indicates a high number of false negatives.
* **F1-Score:** The weighted average of precision and recall.
* **ROC-AUC:** AUROC represents the likelihood of your model distinguishing observations from two classes.
* **Confusion Matrix:** A table showing correct predictions and types of incorrect predictions.

fromsklearn.metricsimportaccuracy\_score

*# Accuracy score for test data*

print("Training Data Accuracy: ",accuracy\_score(train\_preds, y\_train))

OUTPUT:

Training Data Accuracy: 0.9567979669631512

print("Test Data Accuracy: ", accuracy\_score(test\_preds, y\_test))

OUTPUT:

Test Data Accuracy: 0.934010152284264

fromsklearn.metricsimportaccuracy\_score

*# Accuracy score for test data*

print("Training Data Accuracy: ",accuracy\_score(train\_preds, y\_train))

OUTPUT:

Training Data Accuracy: 0.9567979669631512

print("Test Data Accuracy: ", accuracy\_score(test\_preds, y\_test))

OUTPUT:

Test Data Accuracy: 0.934010152284264

*# Calculate performance metrics for the Logistic Regression model*

confusion\_matrix\_logistic=confusion\_matrix(Y\_test, Y\_pred\_logistic)

classification\_report\_logistic=classification\_report(Y\_test, Y\_pred\_logistic, output\_dict=True)

confusion\_matrix\_df=pd.DataFrame(confusion\_matrix\_logistic,columns=["Predicted Negative (0)", "Predicted Positive (1)"],

index=["Actual Negative (0)", "Actual Positive (1)"])

classification\_report\_df=pd.DataFrame(classification\_report\_logistic)

print("Logistic Regression Model - Confusion Matrix:")

print(confusion\_matrix\_df)

print("**\n**Logistic Regression Model - Classification Report:")

print(classification\_report\_df)

OUTPUT:

Logistic Regression Model - Confusion Matrix:

Predicted Negative (0) Predicted Positive (1)

Actual Negative (0) 55225 1238

Actual Positive (1) 2120 54719

Logistic Regression Model - Classification Report:

0 1 accuracy macro avg weighted avg

precision 0.963031 0.977876 0.970362 0.970453 0.970478recall 0.978074 0.962702 0.970362 0.970388 0.970362

f1-score 0.970494 0.970229 0.970362 0.970362 0.970361

support 56463.000000 56839.000000 0.970362 113302.000000 113302.000000

plt.figure(figsize=(7, 5))

sns.heatmap(confusion\_matrix\_df, annot=True, fmt="d", cmap="Blues")

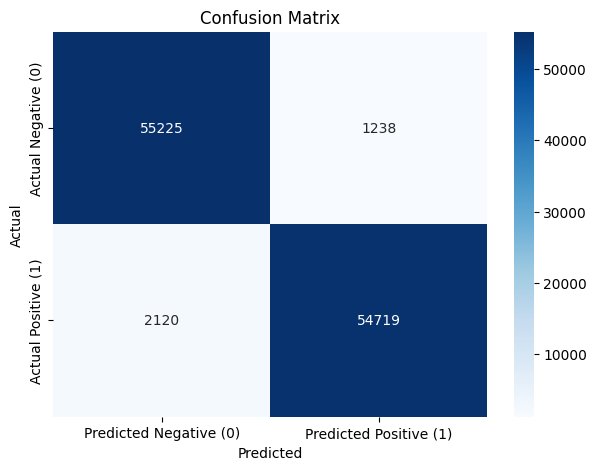
plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

OUTPUT:



*# Calculate ROC-AUC Score and Plot ROC Curve*

fromsklearn.metricsimportroc\_curve, aucy\_scores=logistic\_model.predict\_proba(X\_test)[:, 1]fpr, tpr, thresholds=roc\_curve(Y\_test, y\_scores)roc\_auc=auc(fpr, tpr)

plt.figure(figsize=(7, 5))

plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = **%0.2f**)'%roc\_auc)

plt.plot([0, 1], [0, 1], color='gold', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

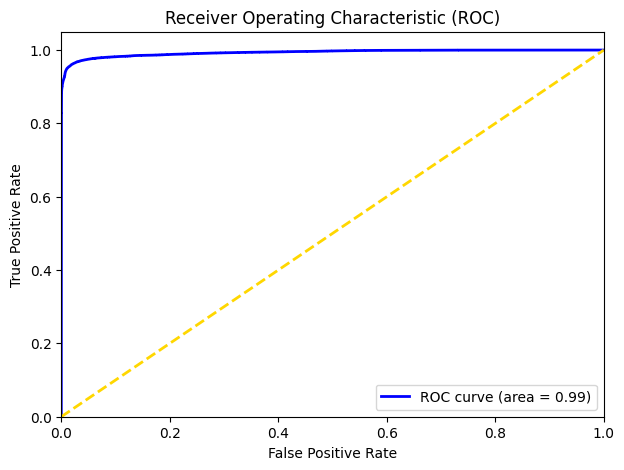
plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC)')

plt.legend(loc="lower right")

plt.show()

OUTPUT:



fraud\_metrics= [classification\_report\_df['0']['precision'], classification\_report\_df['0']['recall'], classification\_report\_df['0']['f1-score']]

legitimate\_metrics= [classification\_report\_df['1']['precision'], classification\_report\_df['1']['recall'], classification\_report\_df['1']['f1-score']]

metrics\_labels= ['Precision', 'Recall', 'F1-Score']

x=range(len(metrics\_labels))

plt.figure(figsize=(8, 6))

plt.bar(x, fraud\_metrics, width=0.4, label='Fraudulent', align='center', color='gold')

plt.bar(x, legitimate\_metrics, width=0.4, label='Genuine', align='edge', color='blue')

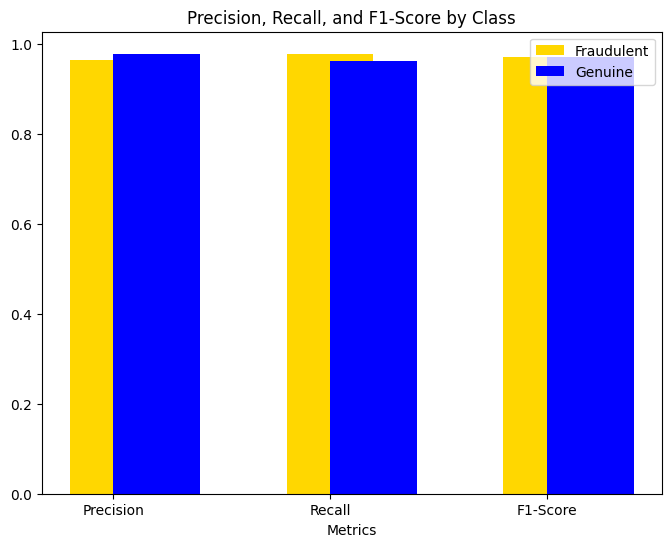
plt.xlabel('Metrics')

plt.xticks(x, metrics\_labels)

plt.title("Precision, Recall, and F1-Score by Class")

plt.legend()plt.show()

OUTPUT:



logreg\_model<- train(form=Class~., data=train,

method="glm", family="binomial",

trControl=kfold\_cv, tuneLength=3)

preds<- predict(logreg\_model, test, type="prob")[,2] *#prob of positive class*preds\_pos<-preds[test[,6]==1] *#preds for true positive class*preds\_neg<-preds[test[,6]==0] *#preds for true negative class*

PRC<-pr.curve(preds\_pos, preds\_neg, curve=TRUE)plot(PRC)

OUTPUT:

Warning message:

“glm.fit: fitted probabilities numerically 0 or 1 occurred”

