CREDIT CARD FRAUD DETECTION

A MACHINE LEARNING PROJECT

Done By,

Srishha.D
(Bachelor Of Computer Science Engineering)

TABLE OF CONTENT

TABLE OF FIGURES

ABSTRACT

This project addresses the critical need for credit card companies to efficiently identify and prevent fraudulent transactions, ensuring that customers are safeguarded against unauthorised charges. The dataset employed comprises credit card transactions conducted by European cardholders in September 2013. Notably, the dataset reveals a highly imbalanced nature. This project actually starts with the exploration of data which involves understanding the data, cleaning the data, data visualisation, preprocessing the data where it will be solving the imbalance in the data, machine learning model building which involves many ML models, cross validation is proceeded for all ML models to find the best and then finally feature importance is found out for the model opted out of cross validation. As the objective of this project Logistic regression, SVC, Decision Tree Classifier models are also built and their inferences are made.The accuracy obtained for SVC and Decision Tree Classifier models are 100% and 99 percent for Logistic Regression.

CHAPTERS

CHAPTER-1

INTRODUCTION

Credit card fraud detection using machine learning is of paramount importance in contemporary financial systems, where electronic transactions have become the norm. With the escalating threat of fraudulent activities and the potential financial losses for both financial institutions and cardholders, the implementation of robust fraud detection systems is critical.

The utilisation of machine learning techniques in credit card fraud detection provides a proactive and adaptive approach to identifying anomalous patterns and potential fraudulent transactions. Traditional rule-based systems may fall short in adapting to the evolving strategies of fraudsters, making machine learning an invaluable tool for staying ahead of sophisticated fraud schemes.

One of the key advantages of machine learning in this context is its ability to analyze vast datasets and discern subtle patterns that may elude rule-based systems. By training models on historical transaction data, machine learning algorithms can learn to recognize normal spending behaviour and promptly identify deviations that may indicate fraudulent activities. This adaptability allows these systems to continuously evolve and improve their accuracy over time.

Moreover, machine learning facilitates the real-time assessment of transactions, enabling swift decision-making to either flag suspicious activities for manual review or automatically block potentially fraudulent transactions. This rapid response is crucial in mitigating the financial impact on both financial institutions and cardholders, reducing the window of vulnerability to fraudulent activities.

CHAPTER-2

SOFTWARE LIBRARIES USED

Pandas

Pandas is a powerful and widely-used Python library for data manipulation and analysis, providing high-performance, easy-to-use data structures and data analysis tools. Central to its capabilities are the DataFrame and Series data structures, which allow users to efficiently organize, clean, and analyze structured data. Pandas simplifies complex data operations with its extensive functionality, offering tools for data cleaning, aggregation, filtering, and transformation. Its seamless integration with other libraries, such as NumPy and Matplotlib, makes it an integral component of the Python data science ecosystem. Whether handling large datasets, exploring data for insights, or preparing data for machine learning tasks, Pandas provides a versatile and intuitive framework that accelerates the data analysis workflow for professionals and enthusiasts alike.

Numpy

NumPy, a fundamental library for numerical computing in Python, plays a pivotal role in scientific and data-intensive applications. Characterized by its efficient and versatile array structures, NumPy enables the manipulation of large, multi-dimensional datasets with unparalleled speed. The library provides an extensive set of mathematical functions, including linear algebra operations, statistical analyses, and random number generation, facilitating complex numerical computations. NumPy's seamless integration with other scientific computing libraries and frameworks, along with its ability to enhance Python's native data structures, makes it an indispensable tool for tasks ranging from data preprocessing and analysis to machine learning and scientific research.

Seaborn

Seaborn is a data visualization library in Python that enhances Matplotlib's capabilities by providing a high-level interface for creating informative and aesthetically pleasing statistical graphics. With its succinct syntax and built-in themes, Seaborn simplifies the creation of complex visualizations, making it particularly useful for exploratory data analysis and presentation-ready graphics. Seaborn excels in generating insightful plots for statistical relationships, distribution comparisons, and multi-plot grids, showcasing its versatility across various types of data. By offering an abstraction layer over Matplotlib, Seaborn enables users to create sophisticated visualizations with minimal code, promoting clarity and interpretability in data-driven narratives. Its integration with Pandas DataFrames and seamless color palette management further solidify Seaborn as a go-to tool for producing compelling and informative visualizations in the Python data science ecosystem.

Matplotlib

Matplotlib stands as a cornerstone in the Python data visualization landscape, providing a comprehensive and flexible framework for creating a wide array of static, animated, and interactive plots. As a 2D plotting library, Matplotlib enables users to generate publication-quality figures with a versatile range of chart types, from line plots and scatter plots to bar charts and histograms. Its syntax and structure serve as a foundation for other visualization libraries and tools in the Python ecosystem. Matplotlib's modular design allows fine-grained control over every aspect of a plot, from axis properties to color schemes, ensuring that users can tailor visualizations to their precise specifications. With its widespread adoption in scientific research, data analysis, and educational contexts, Matplotlib remains a fundamental tool for conveying insights through visual representation in the Python programming language.

Plotly

Plotly is a dynamic and interactive data visualization library in Python that stands out for its capability to produce high-quality, interactive plots and dashboards. With an intuitive syntax, Plotly enables users to create a diverse range of charts, including line plots, bar charts, heatmaps, and 3D visualizations. Its strength lies in its ability to generate interactive plots that respond to user interactions, providing an engaging and exploratory experience. Plotly's compatibility with Jupyter notebooks and its integration with web-based platforms make it a versatile choice for both exploratory data analysis and the development of interactive, web-based applications. Whether used for scientific research, business analytics, or data storytelling, Plotly empowers users to convey complex insights with precision and interactivity, contributing to its popularity in the Python data visualization ecosystem.

Scikit-learn

Scikit-learn, commonly referred to as scikit, is a robust and widely-used machine learning library in Python that provides a comprehensive suite of tools for classical machine learning algorithms. Built on NumPy, SciPy, and Matplotlib, scikit-learn offers user-friendly interfaces for tasks such as classification, regression, clustering, dimensionality reduction, and model selection. Its simplicity, consistency, and extensive documentation make it an accessible resource for both beginners and experienced practitioners in the field of machine learning. Scikit-learn supports a range of supervised and unsupervised learning methods, and its modular design facilitates seamless integration into various data science workflows. With a focus on usability and performance, scikit-learn has become a cornerstone in the Python machine learning ecosystem, contributing to the widespread adoption and success of machine learning applications across diverse industries.

CHAPTER-3

ALGORITHM USED

1.Understanding the problem and data

2.Data cleaning

3.Exploratory Data Analysis

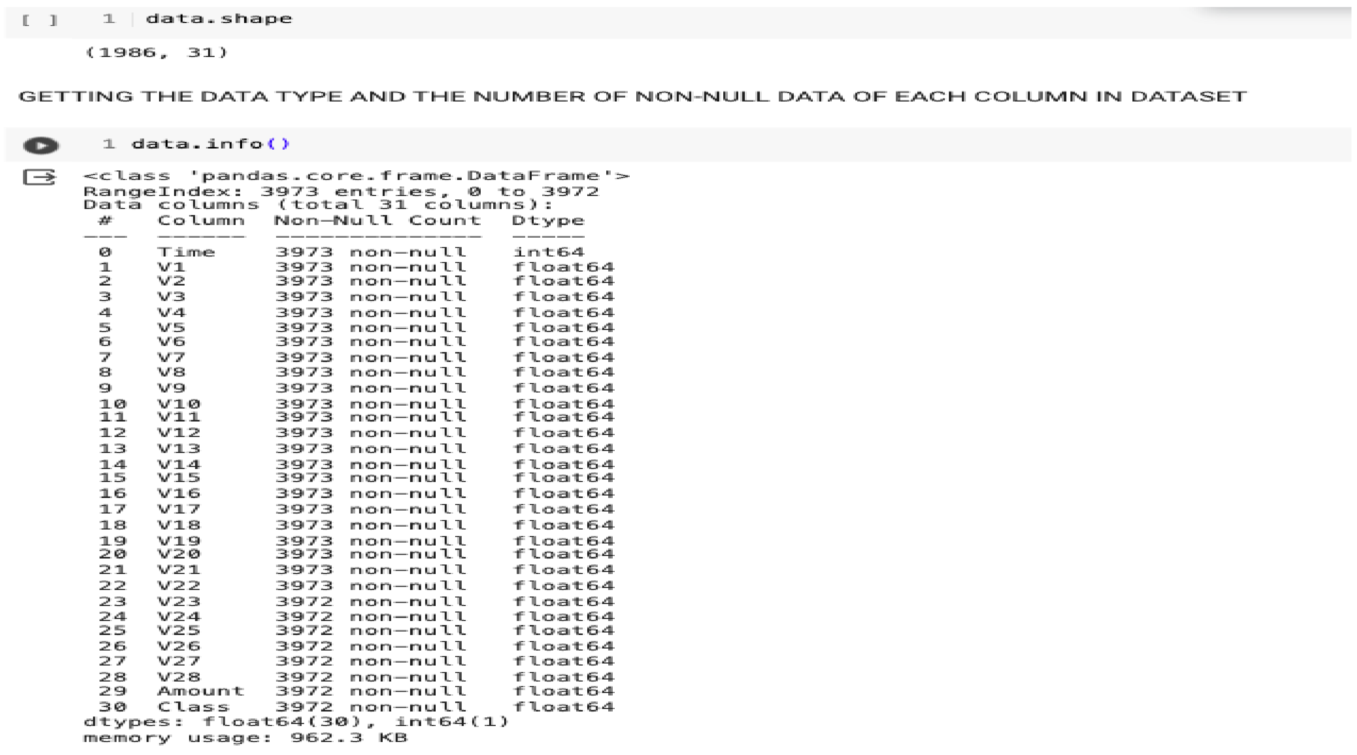
4.Preprocessing the data

5.Machine Learning Model Building

6.Cross Validation

7.Feature importance of best model

1.Understanding the problem and data



```
data.shape
```

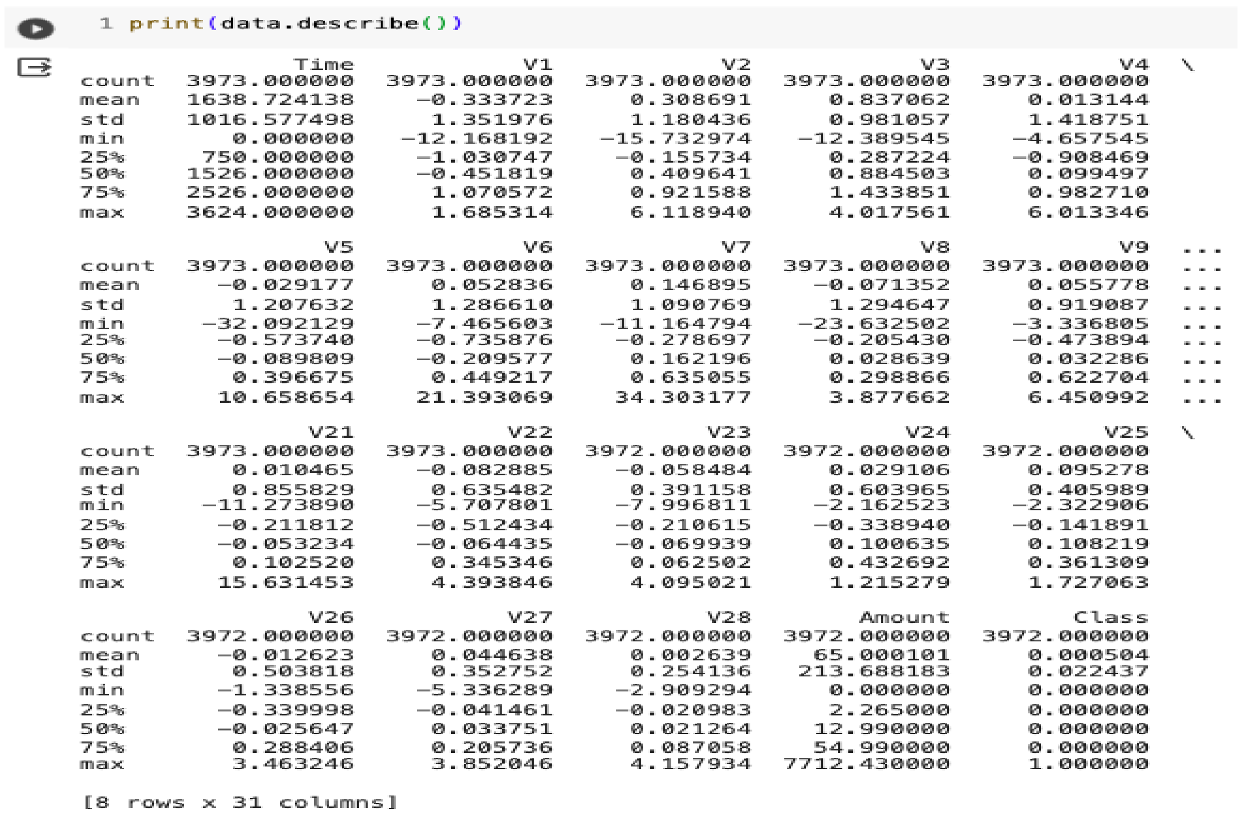
```
(1986, 31)
```

GETTING THE DATA TYPE AND THE NUMBER OF NON-NULL DATA OF EACH COLUMN IN DATASET

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3973 entries, 0 to 3972
Data columns (total 31 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Time     3973 non-null   int64
 1   V1       3973 non-null   float64
 2   V2       3973 non-null   float64
 3   V3       3973 non-null   float64
 4   V4       3973 non-null   float64
 5   V5       3973 non-null   float64
 6   V6       3973 non-null   float64
 7   V7       3973 non-null   float64
 8   V8       3973 non-null   float64
 9   V9       3973 non-null   float64
 10  V10      3973 non-null   float64
 11  V11      3973 non-null   float64
 12  V12      3973 non-null   float64
 13  V13      3973 non-null   float64
 14  V14      3973 non-null   float64
 15  V15      3973 non-null   float64
 16  V16      3973 non-null   float64
 17  V17      3973 non-null   float64
 18  V18      3973 non-null   float64
 19  V19      3973 non-null   float64
 20  V20      3973 non-null   float64
 21  V21      3973 non-null   float64
 22  V22      3973 non-null   float64
 23  V23      3972 non-null   float64
 24  V24      3972 non-null   float64
 25  V25      3972 non-null   float64
 26  V26      3972 non-null   float64
 27  V27      3972 non-null   float64
 28  V28      3972 non-null   float64
 29  Amount   3972 non-null   float64
 30  Class    3972 non-null   float64
dtypes: float64(30), int64(1)
memory usage: 962.3 KB
```

Fig1: Understanding the shape and data type



```python
print(data.describe())
```

```
              Time           V1           V2           V3           V4  \
count  3973.000000  3973.000000  3973.000000  3973.000000  3973.000000
mean   1638.724138    -0.333723     0.308691     0.837062     0.013144
std    1016.577498     1.351976     1.180436     0.981057     1.418751
min       0.000000   -12.168192   -15.732974   -12.389545    -4.657545
25%     750.000000    -1.030747    -0.155734     0.287224    -0.908469
50%    1526.000000    -0.451819     0.409641     0.884503     0.099497
75%    2526.000000     1.070572     0.921588     1.433851     0.982710
max    3624.000000     1.685314     6.118940     4.017561     6.013346

                V5           V6           V7           V8           V9  ...
count  3973.000000  3973.000000  3973.000000  3973.000000  3973.000000  ...
mean     -0.029177     0.052836     0.146895    -0.071352     0.055778  ...
std       1.207632     1.286610     1.090769     1.294647     0.919087  ...
min     -32.092129    -7.465603   -11.164794   -23.632502    -3.336805  ...
25%      -0.573740    -0.735876    -0.278697    -0.205430    -0.473894  ...
50%      -0.089809    -0.209577     0.162196     0.028639     0.032286  ...
75%       0.396675     0.449217     0.635055     0.298866     0.622704  ...
max      10.658654    21.393069    34.303177     3.877662     6.450992  ...

               V21          V22          V23          V24          V25  \
count  3973.000000  3973.000000  3972.000000  3972.000000  3972.000000
mean      0.010465    -0.082885    -0.058484     0.029106     0.095278
std       0.855829     0.635482     0.391158     0.603965     0.405989
min     -11.273890    -5.707801    -7.996811    -2.162523    -2.322906
25%      -0.211812    -0.512434    -0.210615    -0.338940    -0.141891
50%      -0.053234    -0.064435    -0.069939     0.100635     0.108219
75%       0.102520     0.345346     0.062502     0.432692     0.361309
max      15.631453     4.393846     4.095021     1.215279     1.727063

               V26          V27          V28       Amount        Class
count  3972.000000  3972.000000  3972.000000  3972.000000  3972.000000
mean     -0.012623     0.044638     0.002639    65.000101     0.000504
std       0.503818     0.352752     0.254136   213.688183     0.022437
min      -1.338556    -5.336289    -2.909294     0.000000     0.000000
25%      -0.339998    -0.041461    -0.020983     2.265000     0.000000
50%      -0.025647     0.033751     0.021264    12.990000     0.000000
75%       0.288406     0.205736     0.087058    54.990000     0.000000
max       3.463246     3.852046     4.157934  7712.430000     1.000000

[8 rows x 31 columns]
```

Fig2: Description Of Data

The problem trying to solve here is the credit card fraudulent activities. The dataset provided is also heavily imbalanced so our work also aims to solve the imbalance in it and to build a good classification model.Basic information of data is done like the shape, column names, description of data their mean, count, their quartiles and their data types

Data provided consists of 31 columns which can be splitted into 30 features and 1 target

<u>Features</u>

Time:It depicts the time slot of the particular transaction

V1-V28:Anonymous values which can be done for high dimensionality reduction

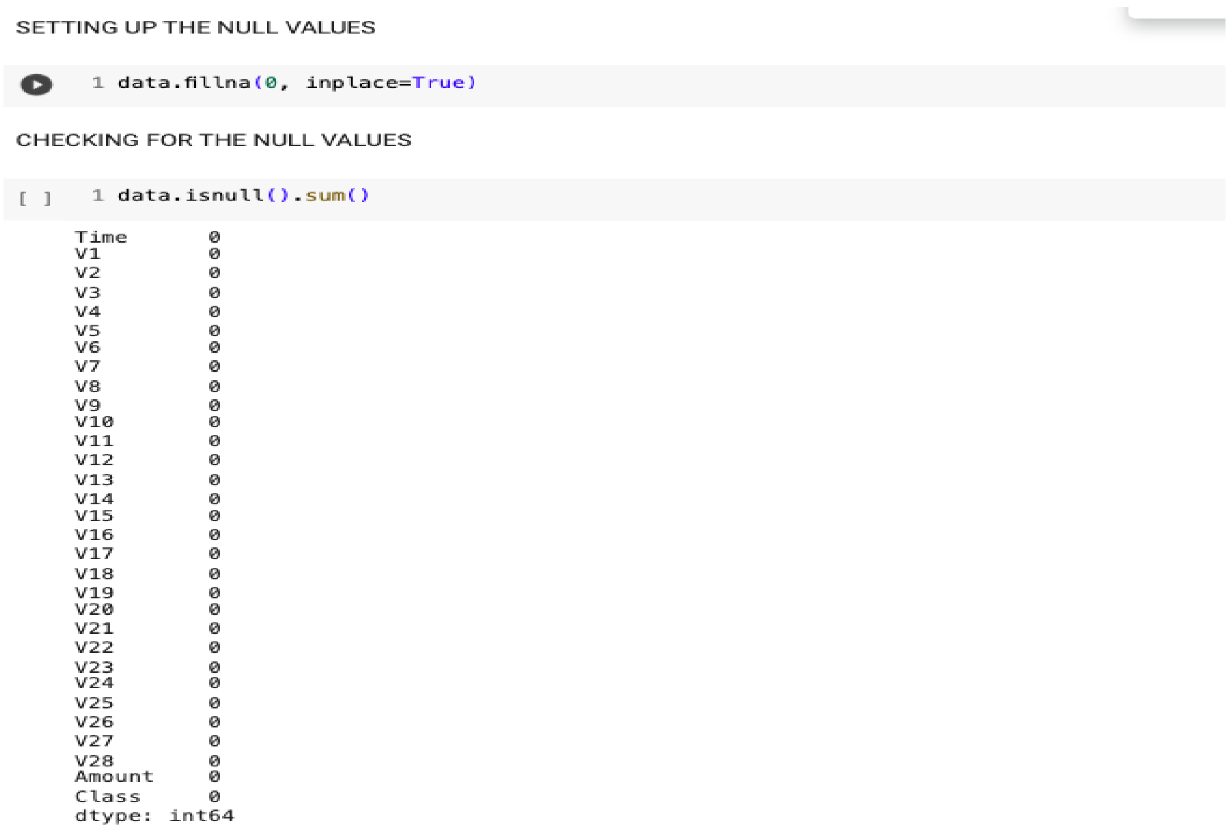
Amount-:Depicts the amount which is transacted

<u>Target</u>

Class:The class which has fraudulent and non-fraudulent classes

2.Data cleaning

The goal of data cleaning is to enhance the quality and reliability of the data, making it suitable for analysis or machine learning applications.Here some null values which were present in the data are removed thereby forming a good model.



SETTING UP THE NULL VALUES

```
data.fillna(0, inplace=True)
```

CHECKING FOR THE NULL VALUES

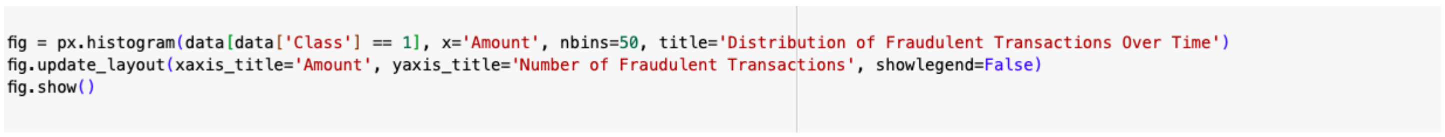
```
data.isnull().sum()
```

```
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
Class     0
dtype: int64
```

Fig3: Setting up the Null values

3.Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in the data analysis process that involves visually and statistically exploring the characteristics of a dataset. EDA helps in understanding the structure, relationships, and patterns within the data, and it plays a significant role in dealing with the dataset in several ways.



```
fig = px.histogram(data[data['Class'] == 1], x='Amount', nbins=50, title='Distribution of Fraudulent Transactions Over Time')
fig.update_layout(xaxis_title='Amount', yaxis_title='Number of Fraudulent Transactions', showlegend=False)
fig.show()
```

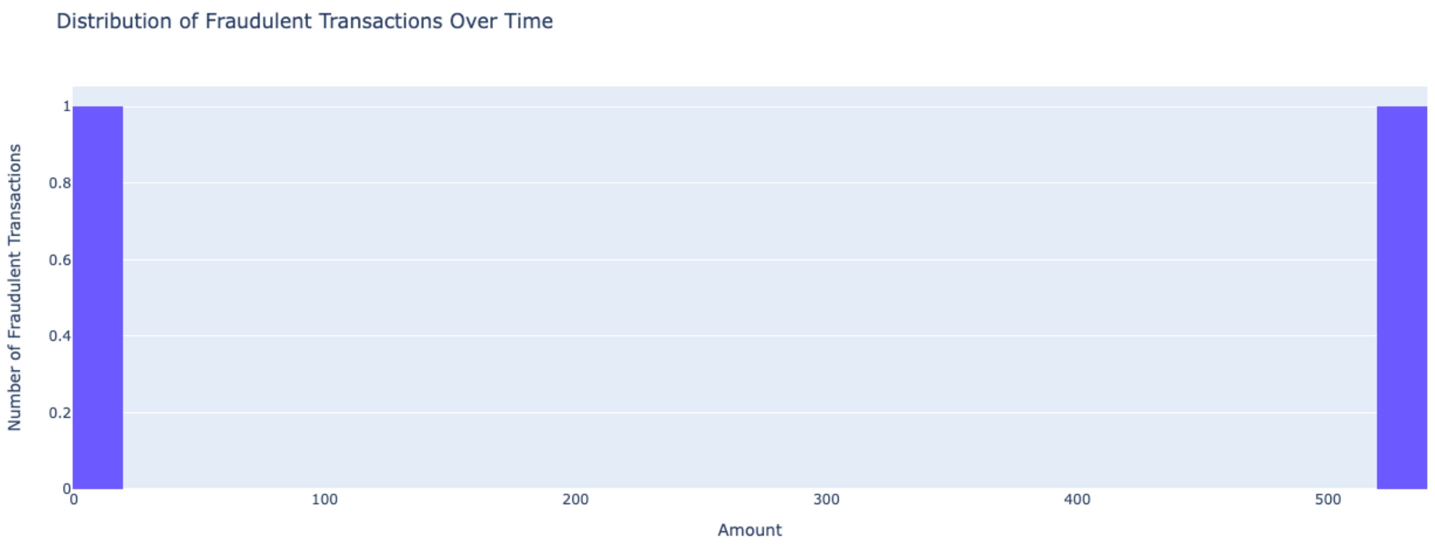
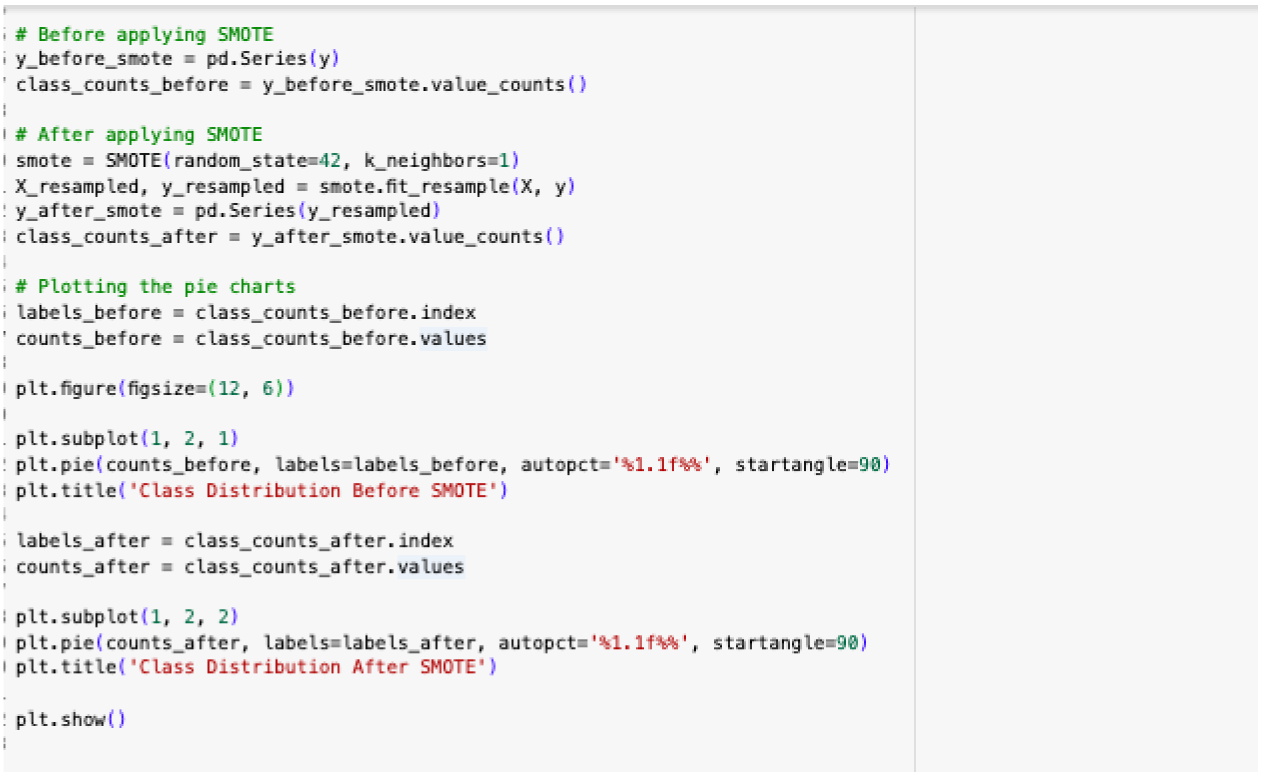


Fig4: Visualisations over data

4.Preprocessing the data

Data preprocessing is a crucial step in preparing raw datasets for analysis and machine learning. It involves a series of operations to clean, transform, and organize data to enhance its quality and usability.



```python
# Before applying SMOTE
y_before_smote = pd.Series(y)
class_counts_before = y_before_smote.value_counts()

# After applying SMOTE
smote = SMOTE(random_state=42, k_neighbors=1)
X_resampled, y_resampled = smote.fit_resample(X, y)
y_after_smote = pd.Series(y_resampled)
class_counts_after = y_after_smote.value_counts()

# Plotting the pie charts
labels_before = class_counts_before.index
counts_before = class_counts_before.values

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.pie(counts_before, labels=labels_before, autopct='%1.1f%%', startangle=90)
plt.title('Class Distribution Before SMOTE')

labels_after = class_counts_after.index
counts_after = class_counts_after.values

plt.subplot(1, 2, 2)
plt.pie(counts_after, labels=labels_after, autopct='%1.1f%%', startangle=90)
plt.title('Class Distribution After SMOTE')

plt.show()
```

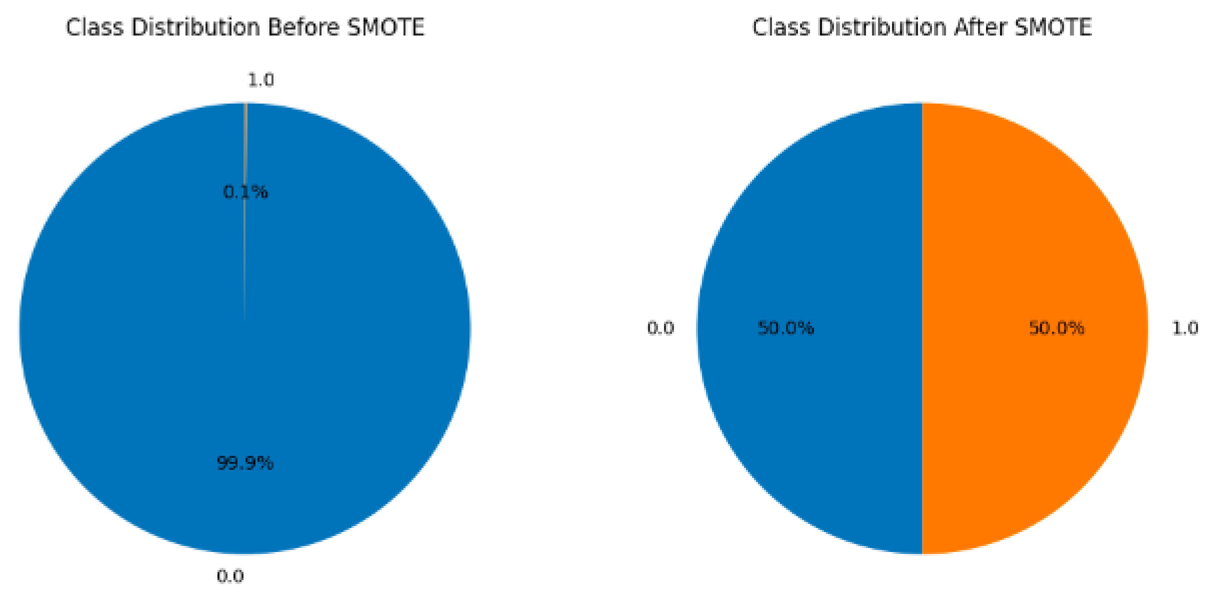
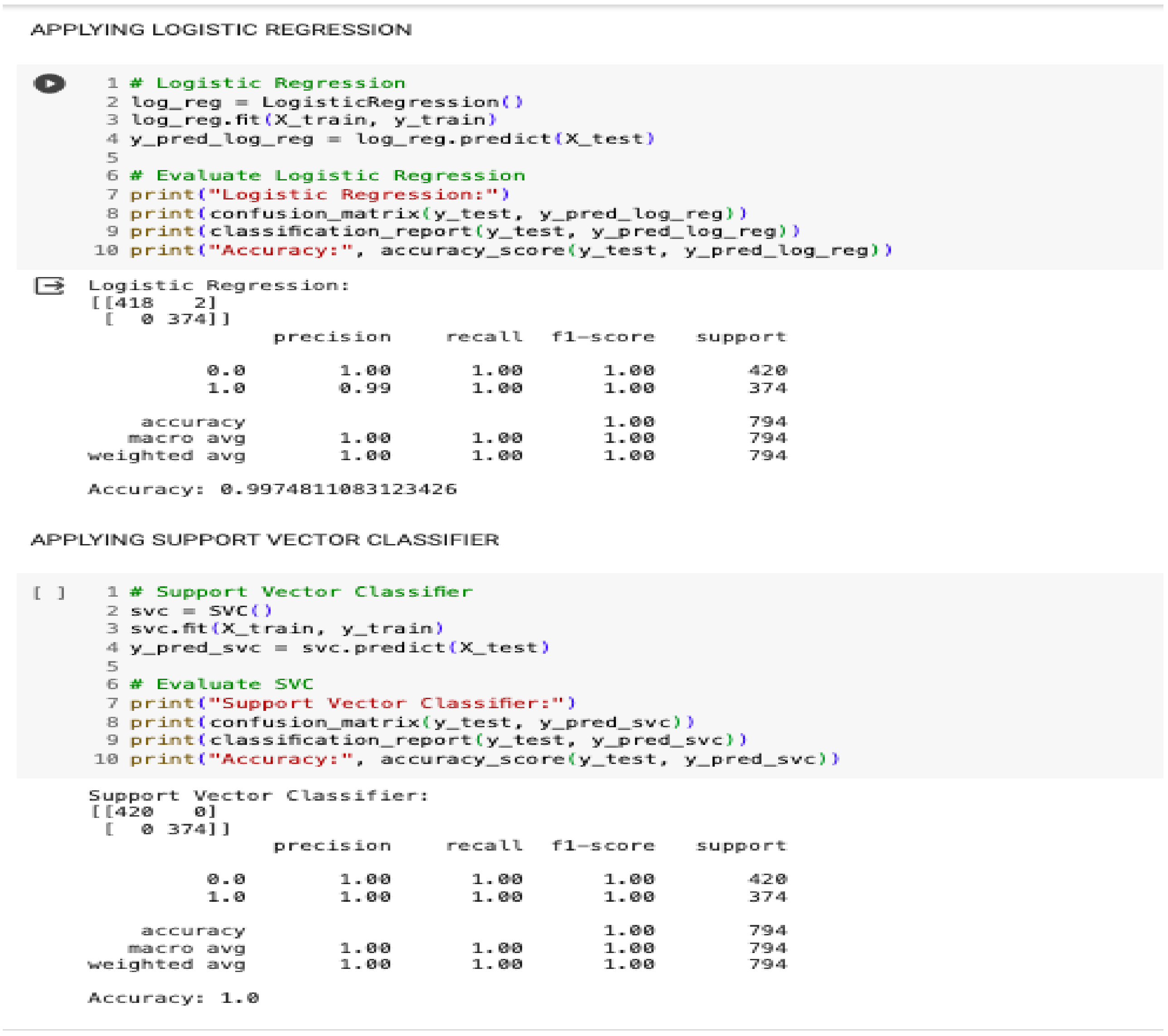


Fig5: Applying SMOTE to remove the imbalance in data

5.Machine Learning Model Building

Machine learning model building involves the process of selecting and training a predictive algorithm on a dataset to make accurate predictions or classifications on new, unseen data. It typically includes steps such as data preprocessing, feature engineering, splitting the dataset into training and testing sets, selecting an appropriate algorithm (e.g., decision trees, support vector machines, or neural networks), training the model on the training set, and evaluating its performance on the testing set. The goal is to create a model that generalizes well to new data and provides meaningful insights or predictions based on the patterns it learned during training.



APPLYING LOGISTIC REGRESSION

```
# Logistic Regression
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred_log_reg = log_reg.predict(X_test)

# Evaluate Logistic Regression
print("Logistic Regression:")
print(confusion_matrix(y_test, y_pred_log_reg))
print(classification_report(y_test, y_pred_log_reg))
print("Accuracy:", accuracy_score(y_test, y_pred_log_reg))
```

```
Logistic Regression:
[[418   2]
 [  0 374]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       420
         1.0       0.99      1.00      1.00       374

    accuracy                           1.00       794
   macro avg       1.00      1.00      1.00       794
weighted avg       1.00      1.00      1.00       794

Accuracy: 0.9974811083123426
```

APPLYING SUPPORT VECTOR CLASSIFIER

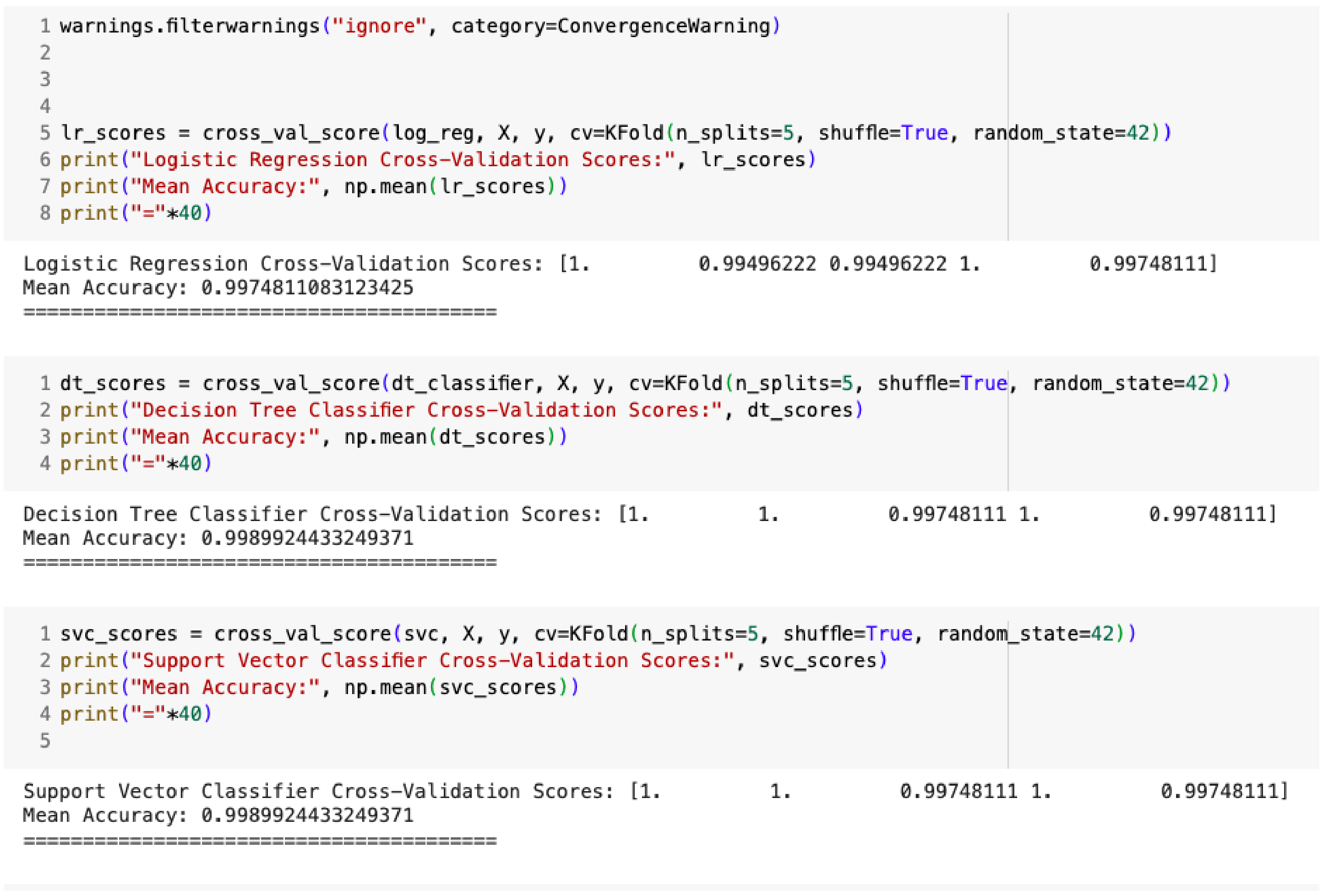
```
# Support Vector Classifier
svc = SVC()
svc.fit(X_train, y_train)
y_pred_svc = svc.predict(X_test)

# Evaluate SVC
print("Support Vector Classifier:")
print(confusion_matrix(y_test, y_pred_svc))
print(classification_report(y_test, y_pred_svc))
print("Accuracy:", accuracy_score(y_test, y_pred_svc))
```

```
Support Vector Classifier:
[[420   0]
 [  0 374]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       420
         1.0       1.00      1.00      1.00       374

    accuracy                           1.00       794
   macro avg       1.00      1.00      1.00       794
weighted avg       1.00      1.00      1.00       794

Accuracy: 1.0
```

Fig6: Machine Learning Model Building

6.Cross Validation

Cross-validation in data science is a technique used to assess the performance and generalizability of a machine learning model. It involves dividing the dataset into multiple subsets, training the model on some of these subsets, and evaluating its performance on the remaining subsets. The most common type of cross-validation is k-fold cross-validation, where the data is split into k folds or partitions. The model is trained k times, each time using k-1 folds for training and the remaining fold for validation. This process is repeated until each fold has been used as the validation data exactly once. Cross-validation helps in obtaining a more robust estimate of a model's performance, reducing the impact of randomness in the data split, and identifying potential issues like overfitting or underfitting. It is a crucial step in assessing a model's ability to generalize well to new, unseen data.



```
warnings.filterwarnings("ignore", category=ConvergenceWarning)



lr_scores = cross_val_score(log_reg, X, y, cv=KFold(n_splits=5, shuffle=True, random_state=42))
print("Logistic Regression Cross-Validation Scores:", lr_scores)
print("Mean Accuracy:", np.mean(lr_scores))
print("="*40)
```

```
Logistic Regression Cross-Validation Scores: [1.         0.99496222 0.99496222 1.         0.99748111]
Mean Accuracy: 0.9974811083123425
========================================
```

```
dt_scores = cross_val_score(dt_classifier, X, y, cv=KFold(n_splits=5, shuffle=True, random_state=42))
print("Decision Tree Classifier Cross-Validation Scores:", dt_scores)
print("Mean Accuracy:", np.mean(dt_scores))
print("="*40)
```

```
Decision Tree Classifier Cross-Validation Scores: [1.         1.         0.99748111 1.         0.99748111]
Mean Accuracy: 0.9989924433249371
========================================
```

```
svc_scores = cross_val_score(svc, X, y, cv=KFold(n_splits=5, shuffle=True, random_state=42))
print("Support Vector Classifier Cross-Validation Scores:", svc_scores)
print("Mean Accuracy:", np.mean(svc_scores))
print("="*40)

```

```
Support Vector Classifier Cross-Validation Scores: [1.         1.         0.99748111 1.         0.99748111]
Mean Accuracy: 0.9989924433249371
========================================
```

Fig7:Checking Cross Validation Scores

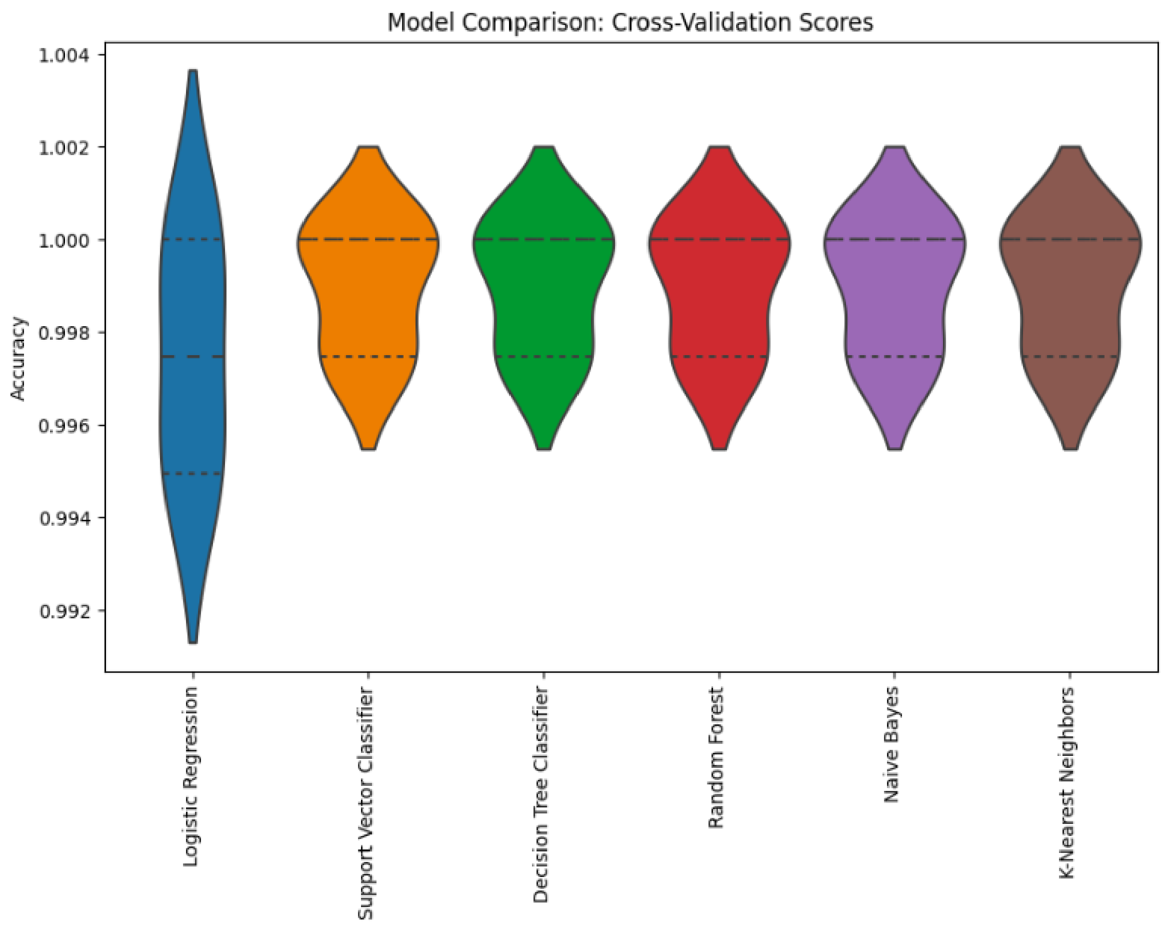
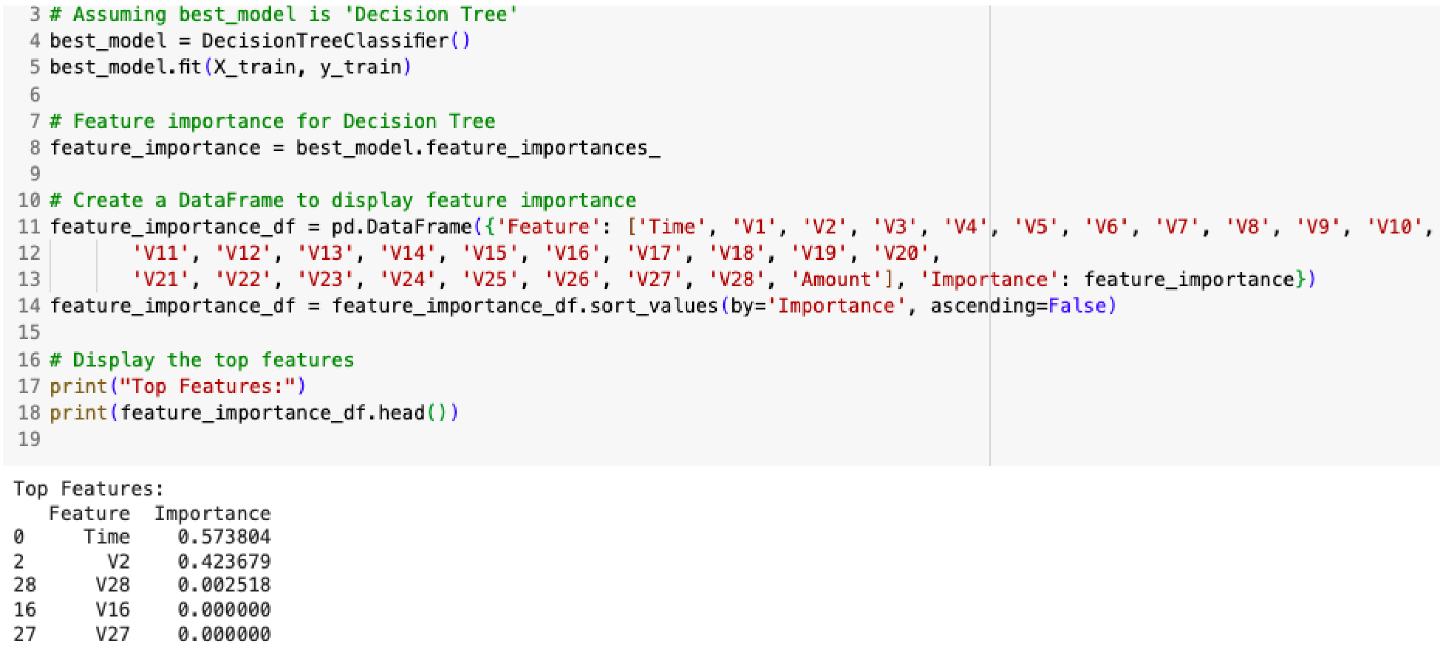


Fig8:Comparison Of Cross Validation Scores

7.Feature importance of best model

Feature importance in the context of machine learning refers to the assessment of the contribution of each feature (input variable) in a model's predictive performance. After training the best model, which could be a decision tree, random forest, or other models, you can evaluate the importance of each feature. Feature importance is typically determined by analyzing how much each feature contributes to reducing the model's prediction error.



```python
# Assuming best_model is 'Decision Tree'
best_model = DecisionTreeClassifier()
best_model.fit(X_train, y_train)

# Feature importance for Decision Tree
feature_importance = best_model.feature_importances_

# Create a DataFrame to display feature importance
feature_importance_df = pd.DataFrame({'Feature': ['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'], 'Importance': feature_importance})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Display the top features
print("Top Features:")
print(feature_importance_df.head())

```

```
Top Features:
   Feature  Importance
0     Time    0.573804
2       V2    0.423679
28     V28    0.002518
16     V16    0.000000
27     V27    0.000000
```

Fig9:Feature Importance Of the model

CHAPTER-4

INFERENCE OF MACHINE LEARNING MODELS

1.LOGISTIC REGRESSION

2.SUPPORT VECTOR CLASSIFIER

3.DECISION TREE CLASSIFIER

1.LOGISTIC REGRESSION

Logistic Regression is a linear model used for binary classification problems, where the outcome variable is categorical with two classes. Despite its name, it's a classification algorithm rather than a regression one. Logistic Regression uses the logistic function (sigmoid) to map the output to a probability between 0 and 1. If the probability is above a certain threshold (usually 0.5), the input is classified as one class; otherwise, it's classified as the other.

- Simple and interpretable.
- Assumes a linear relationship between input features and the log-odds of the output.
- Regularization terms (L1 or L2) can be applied to prevent overfitting.
- Widely used for its simplicity and efficiency in binary classification tasks.

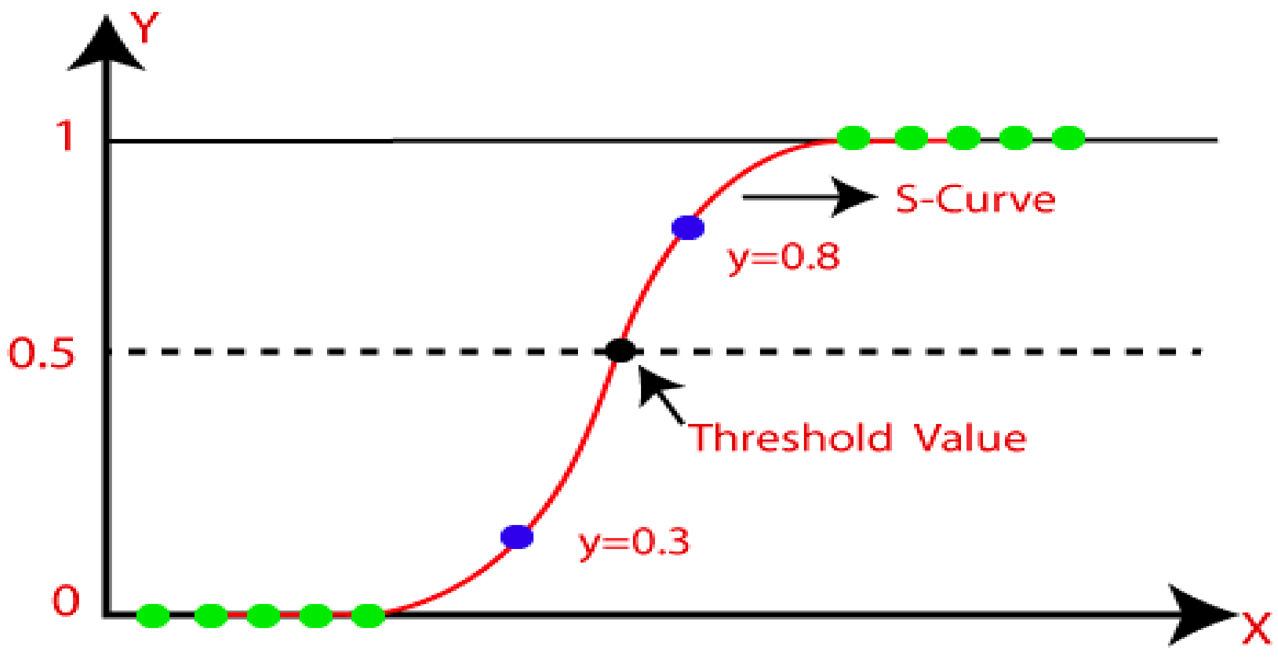


Fig10:Logistic Regression

2.SUPPORT VECTOR CLASSIFIER

Support Vector Classifier (SVC), also known as Support Vector Machine (SVM) in the context of classification, is a powerful supervised learning algorithm used for both binary and multi-class classification tasks. Here's a detailed explanation of the Support Vector Classifier:

SVC aims to find a hyperplane in a high-dimensional space that best separates the data into different classes. The goal is to maximize the margin, which is the distance between the hyperplane and the nearest data points of each class.

SVM can efficiently handle non-linear decision boundaries by using the kernel trick. This involves mapping the input features into a higher-dimensional space, where a linear separation is performed. Common kernel functions include linear, polynomial, and radial basis function (RBF or Gaussian) kernels.In real-world scenarios where data might not be perfectly separable, SVC introduces the concept of a soft margin.

The decision function in SVC assigns a new data point to a particular class based on its position with respect to the hyperplane. The sign of the decision function indicates the predicted class.SVC is evaluated based on metrics like accuracy, precision, recall, and F1 score. Fine-tuning involves selecting appropriate kernel functions, adjusting the regularization parameter (C), and optimizing other hyperparameters.Support Vector Classifier is widely used in various applications, including image classification, text classification, and bioinformatics, due to its effectiveness in handling complex decision boundaries and its ability to generalize well to unseen data.

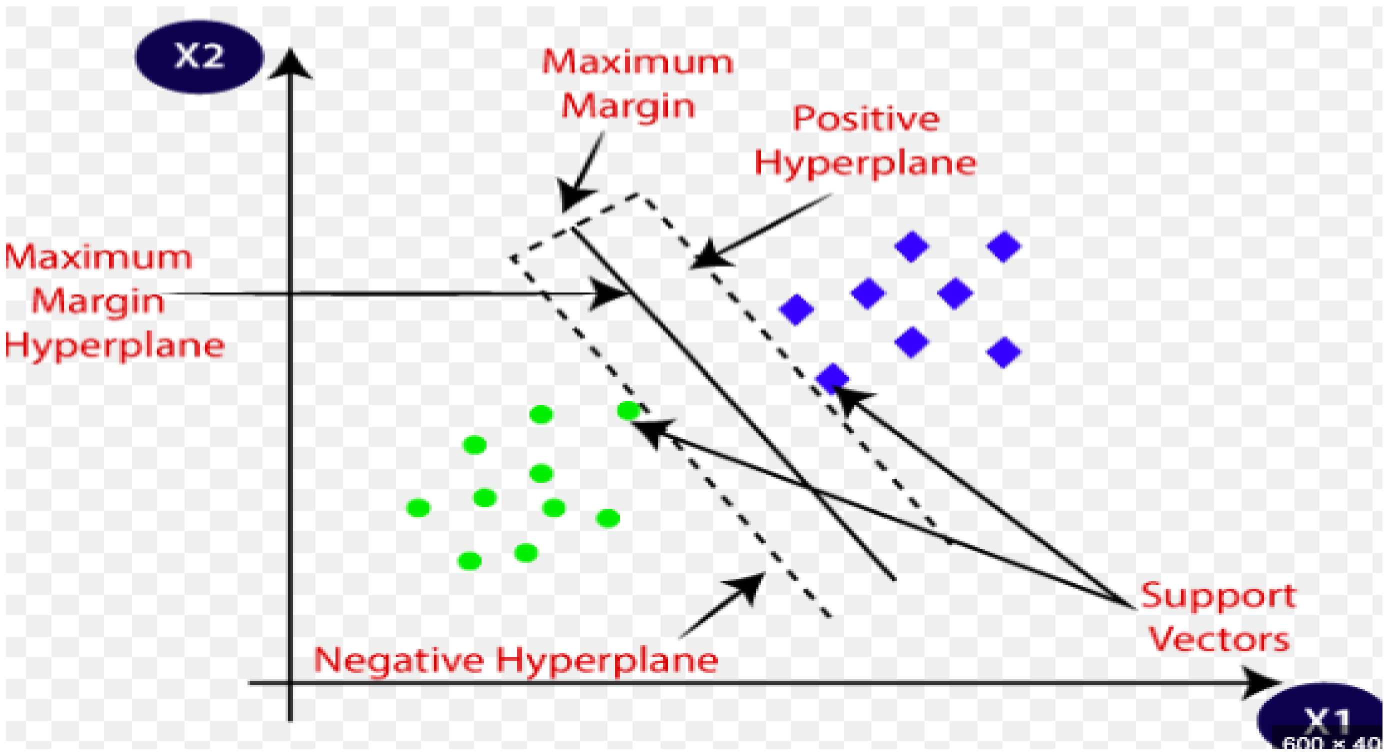


Fig11:Support Vector Classifier

3.DECISION TREE CLASSIFIER

A Decision Tree Classifier is a versatile and interpretable machine learning algorithm used for both classification and regression tasks. It creates a tree-like structure where each internal node represents a decision based on a feature, each branch represents the outcome of the decision, and each leaf node represents the final predicted class label (for classification) or a continuous value (for regression)

A Decision Tree is a hierarchical structure composed of nodes. The top node is called the root, internal nodes represent decisions based on features, branches represent the outcomes of decisions, and leaf nodes represent the final predictions.

The tree is built through a process called recursive binary splitting. At each node, the algorithm selects the feature and the split point that maximizes the information gain or Gini impurity (for classification) or minimizes the mean squared error (for regression).Information gain measures the reduction in entropy or Gini impurity achieved by splitting a dataset. A split that leads to more homogeneity in the classes results in higher information gain.For regression, mean squared error is minimized at each split to reduce the variance in predicted values.

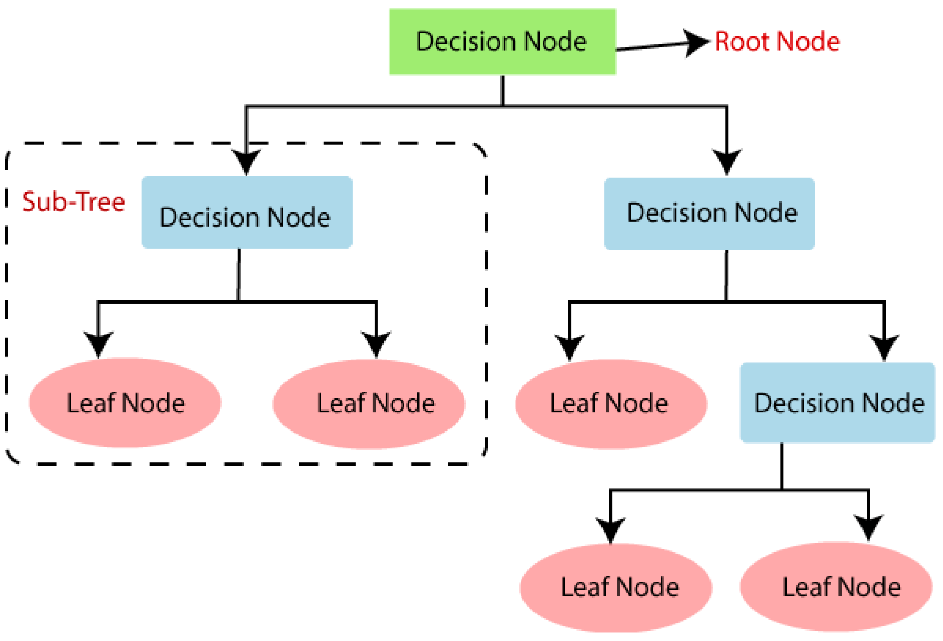


Fig12:Decision Tree Classifier

CHAPTER-5

CONCLUSION

In this project the dataset undergoes some visualisations in the perspective of understanding it and then some preprocessing techniques in order to remove the imbalance as it is the one of the objective of this project .Machine learning models are built in order to find out the accuracies and how the data reacts to each model.Finally, some evaluations were done over the model built, cross validation scores were compared and the important features found out for the models.