

A Mini Project Report

On

**CREDIT CARD FRAUD DETECTION USING K NEAREST
NEIGHBORS CLASSIFICATION**

Submitted in partial fulfillment of the
Requirements for the award of the degree

**BACHELOR OF TECHNOLOGY
IN
INFORMATION TECHNOLOGY**

Submitted By
Dabbeti Varaprasad(21EG512104)

Under the guidance of
Mr. G. L. Anand Babu
Assistant Professor



Department of Information Technology

ANURAG UNIVERSITY

(Approved by AICTE and NBA Accredited)

Venkatapur (V), Ghatkesar (M), Medchal district, Hyderabad, Telangana, 500088

2020-2024

ANURAG UNIVERSITY

(Approved by AICTE and NBA Accredited)

Venkatapur (V), Ghatkesar (M), Medchal district, Hyderabad, Telangana, 500088

Department of Information Technology



CERTIFICATE

This is to certify that the mini project report entitled “**Credit Card Fraud Detection Using KNN Classification**” is a project work done and submitted by **Dabbeti Varaprasad (21EG512104)** in partial fulfillment of the requirements for the award of the degree of B.Tech in **Information Technology** from **Anurag University**, Hyderabad during the academic year 2023-2024 and the Bonafide work has not been submitted elsewhere for the award of any other degree.

Internal Guide

Mr. G. L. Anand Babu

Assistant Professor

Department of IT

H.O.D

Dr. K. S. Reddy

Professor

Dean, Academics and Planning

External Examiner

ACKNOWLEDGEMENT

We would like to express our sincere thanks to **Dr. K.S. Reddy**, Dean, Academics and Planning, Head of the Department of Information Technology, Anurag University, Ghatkesar, whose motivation in the field of software development has made us to overcome all hardships during the course of study and successful completion of project.

We would like to express our profound sense of gratitude to all for having helped us in completing this dissertation. We would like to express our deep-felt gratitude and sincere thanks to our guide **Mr. G. L. Anand Babu**, Assistant Professor, Department of Information Technology, Anurag University, Ghatkesar, for his skillful guidance, timely suggestions and encouragement in completing this project.

We extend our sincere thanks to **Dr. V. Vijaya Kumar**, Dean, School of Engineering, **Dr. K.S. Reddy**, Dean, Academics and Planning, Head of the Department of Information Technology, of **Anurag University**, Venkatapur(V), Ghatkesar(M), R.R. Dist, for their encouragement and constant help.

Finally, we would like to express our heartfelt thanks to our parents who were very supportive both financially and mentally and for their encouragement to achieve our set goals.

Dabbeti Varaprasad (21EG512104)

DECLARATION

This is to Certify that the project work entitled “**CREDIT CARD FRAUD DETECTION USING KNN CLASSIFICATION**” submitted to Anurag University in partial fulfillment of the requirement for the award of the Degree of Bachelor of Technology (B-Tech), is an original work carried out by **Dabbeti Varaprasad (21EG512104)** under the guidance of **Mr. G. L. Anand Babu**, Assistant Professor in the Department of Information Technology. This matter embodied in this project is a genuine work, done by the students and has not been submitted whether the university or to any other university/Institute for the fulfillment of the requirement of any course of study.

Dabbeti Varaprasad (21EG512104)

ABSTRACT

In this project, I employed machine learning to construct a credit card fraud detection system. The dataset underwent meticulous preprocessing, addressing mixed data types and handling NaN values. StratifiedShuffleSplit ensured an equitable distribution of fraudulent and legitimate transactions in both training and testing sets. Standard scaling was crucial for the K-Nearest Neighbors (KNN) classifier's optimal performance. The model, trained with a KNN classifier, demonstrated its efficacy through accuracy metrics and a visualized confusion matrix. Testing with a newly generated transaction illustrated the practical application of the model, providing a rapid assessment of its fraud prediction capabilities. Overall, this project contributes a reliable fraud detection solution, leveraging machine learning to bolster the security and integrity of credit card transactions.

CONTENTS

CHAPTERS	PAGE
List of Figures	i
List of Abbreviations	i
1. Introduction	
1.1 Overview	1
1.2 Objectives	1
1.3 Problem Formulation	1
1.4 Scope of the Project	2
1.5 Feasibility	2
1.6 Requirements of the Project	
1.6.1 Software Requirements	2
1.6.2 Hardware Requirements	2
2. Requirements	
2.1 Requirements	3
2.2 Non-Functional Requirements	7
2.3 Functional Requirements	7
3. Analysis	
3.1 Use Case Model	8
3.1.1 Use Case Diagram	8
3.3 Activity Diagram	9
4. Implementation	
4.1 Technologies Used	10
4.2 Main.py	13
5. Testing Methodology	17
6. Implementation and Snapshots	
6.1 Testing and Training Accuracy	25
6.2 Prediction for New Model	26
7. Results	
7.1 Results	28
7.2 confusion Matrix	28

8. Conclusions and Future Work	
8.1 Conclusion	29
8.2 Future Work	30
9. References	31

LIST OF FIGURES

Figure Number	Name of Figure	Page No
3.1.1	Use Case Diagram	8
3.2.1	Activity Diagram	9
6.1.4	Confusion Matrix	24
6.1.5	Testing Accuracy	25
6.1.6	Training Accuracy	25
6.1.7	Prediction for new model	26

LIST OF ABBREVIATIONS

TERMS	DESCRIPTION
pd	pandas
np	numpy
sns	seaborn
Nan	Non Numeric Values
Knn	K Nearest Neighbors

1.INTRODUCTION

1.1. Overview

The project utilizes K-Nearest Neighbors algorithm for Credit Card Fraud Detection, achieving accurate predictions by training on a labeled dataset and visualizing performance with a confusion matrix.

The Credit Card Fraud Detection project employs the K-Nearest Neighbors (KNN) algorithm to identify fraudulent transactions with precision. By leveraging a labeled dataset, the model undergoes training to discern patterns indicative of fraudulent behavior. The KNN algorithm classifies transactions based on the similarity of features, enhancing accuracy through proximity analysis. The project's efficacy is quantified and comprehensively visualized using a confusion matrix, offering insights into true positives, true negatives, false positives, and false negatives. This visualization aids in evaluating the model's performance, highlighting its ability to correctly identify fraudulent transactions while minimizing false positives and negatives. The combination of KNN, labeled data, and a confusion matrix provides a robust framework for enhancing credit card fraud detection, offering a reliable solution for financial security.

1.2. Objective

The objective of the Credit Card Fraud Detection project is to develop a machine learning model, specifically a K-Nearest Neighbors (KNN) classifier, that can accurately identify fraudulent transactions based on features such as transaction amounts and various V1-V28 features, contributing to the enhancement of financial security.

1.3. Problem Formulation

The problem formulation involves detecting fraudulent credit card transactions using a K-Nearest Neighbors (KNN) classifier. The goal is to develop a model that accurately identifies instances of fraud based on features of credit card transactions, providing a reliable mechanism for fraud detection in financial transactions.

1.4. Scope

The Credit Card Fraud Detection project aims to identify potentially fraudulent transactions by leveraging machine learning, specifically K-Nearest Neighbors (KNN). The project involves preprocessing credit card transaction data, splitting it into training and testing sets, scaling features, and training a KNN classifier. The model's performance is evaluated through accuracy metrics and visualizations, showcasing its ability to distinguish between genuine and fraudulent transactions. This project is crucial for enhancing financial security, providing an automated system that detects and mitigates fraudulent activities, safeguarding users from unauthorized transactions in the realm of credit card transactions.

1.5. Feasibility

The feasibility of the Credit Card Fraud Detection project using KNN is strong. KNN is a robust algorithm for classification tasks, and its application to fraud detection leverages patterns in data to identify anomalous transactions. The use of a stratified train-test split ensures reliable model evaluation, while the StandardScaler enhances performance. The project's ability to accurately predict fraud or non-fraud transactions, as demonstrated on a new transaction, suggests its practical viability for real-world applications. The project exhibits a well-structured approach, combining data preprocessing, model training, evaluation, and application on a new example for comprehensive feasibility.

1.6. System Requirements

1.6.1 Software Requirements

- i. Programming Language : Python
- ii. Python Environments: Google Collab / Jupyter Notebook
- iii. Operating System: Windows

1.6.2 Hardware Requirements

- i. 8GB RAM
- ii. Processor Core i5

2.REQUIREMENTS

2.1Requirements Specification

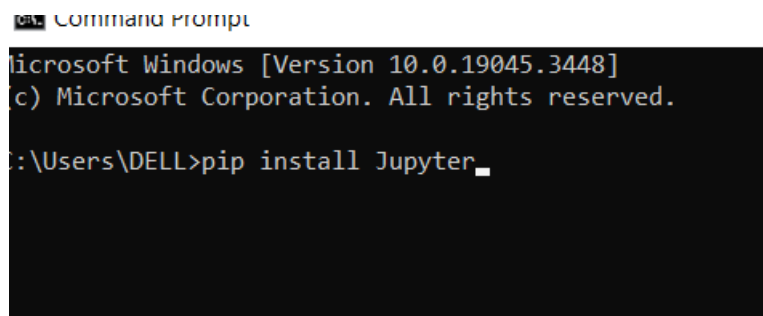
- A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fulfil all users' needs.
- Functional requirements define the external behavior of the system. In other words, it illustrates what the system does and what the system must not do. It also involves calculations, technical details, data manipulation and processing, and other specific functionalities.
- A non-functional requirement is a goal or stipulation that describes how a software system must work or perform.

2.1.1JUPYTER NOTEBOOK:

Jupyter Notebook is an interactive environment for writing and running code. It is widely used for data science, machine learning, and scientific computing. Jupyter Notebook allows you to create documents that contain live code, equations, visualizations, and narrative text. This makes it a powerful tool for exploring data, developing models, and communicating your work.

Steps to install Jupyter Notebook:

- Make sure that you have Python 3 installed.
- Open a terminal and install the Jupyter Notebook package using the following command:

A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt". The window content shows the following text: "Microsoft Windows [Version 10.0.19045.3448]", "(c) Microsoft Corporation. All rights reserved.", and the command prompt "C:\Users\DELL>pip install Jupyter_".

```
Microsoft Windows [Version 10.0.19045.3448]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DELL>pip install Jupyter_
```

Fig 2.1.1.1 Jupyter Installation

- To start Jupyter Notebook, run the following command:

```
jupyter notebook
```

Fig 2.1.1.2 Jupyter Notebook

- This will open a web browser window with the Jupyter Notebook interface.

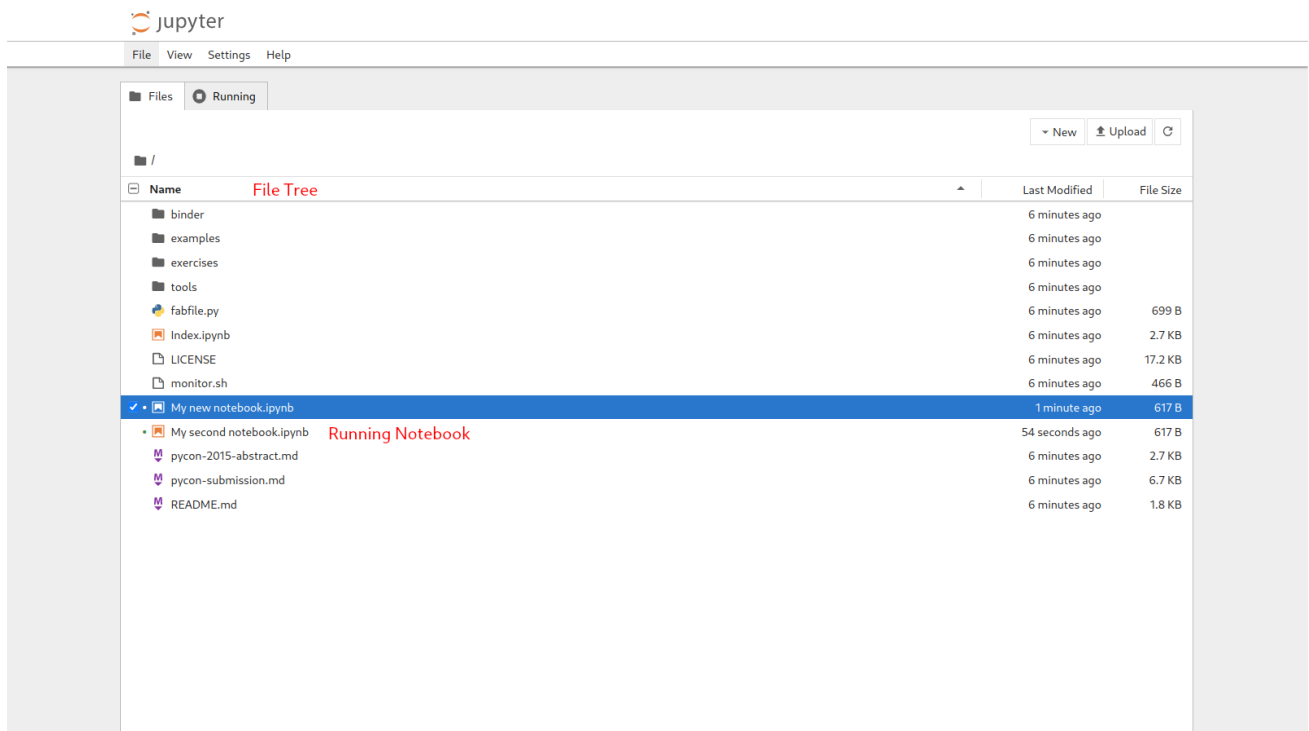


Fig 2.1.1.3 Creating New File

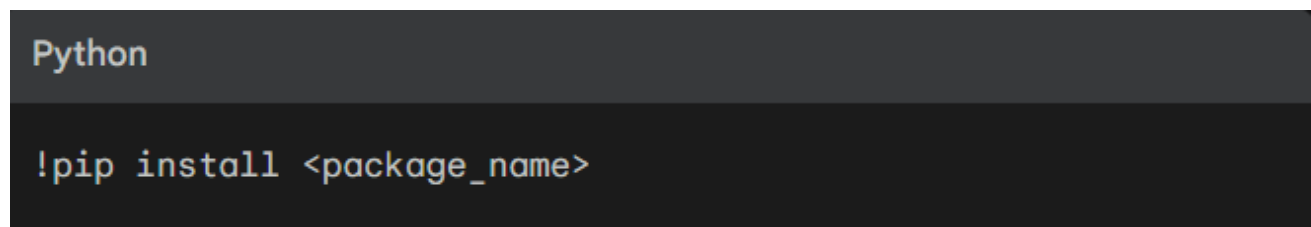
2.1.2 GOOGLE COLAB:

Google Colab is a free Jupyter notebook environment that runs in your browser. It allows you to write and execute Python code without having to install any software on your local machine. Colab also provides access to powerful computing resources, including GPUs, for free.

To deploy Python code on Google Colab, you can either upload a Python file (.py) to your Google Drive and then open it in Colab, or you can write your code directly in a Colab notebook.

Steps to Install:

- Go to the Google Colab website: <https://research.google.com/colaboratory/>
- Click the New Python3 Notebook button.
- In the first code cell, type the following command to install any required Python packages:

A screenshot of a Google Colab code cell. The cell has a dark background. At the top, the word "Python" is written in a light blue font. Below it, the command `!pip install <package_name>` is written in a light blue font.

```
Python

!pip install <package_name>
```

Fig 2.1.2.1 Package import

For example, to install the NumPy package, you would type:

A screenshot of a Google Colab code cell. The cell has a dark background. The command `!pip install numpy` is written in a light blue font.

```
!pip install numpy
```

Fig 2.1.2.2 Import Numpy

- Press Shift+Enter to run the code cell.

Once you have installed all of the required Python packages, you can start writing and executing your code in the Colab notebook.

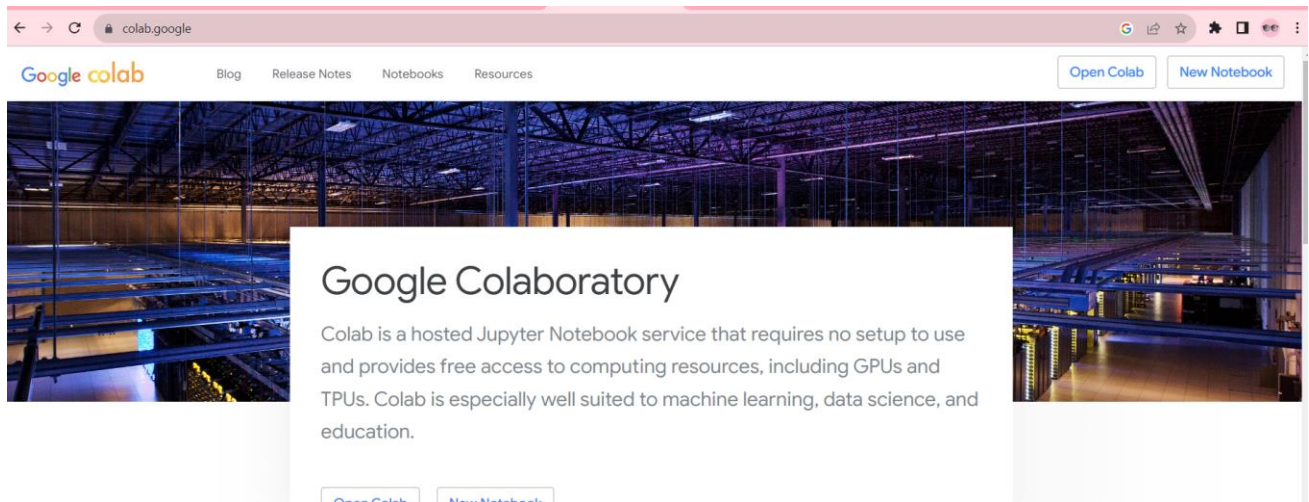


Fig 2.1.2.3 Google colab

To deploy your Python code:

- Save your Colab notebook as a Python file (.py).
- Click the Runtime menu and then select Change runtime type.
- In the Runtime type drop-down menu, select GPU.
- Click the Save button.
- Click the Runtime menu and then select Run all.
- This will run your code and deploy your Python application

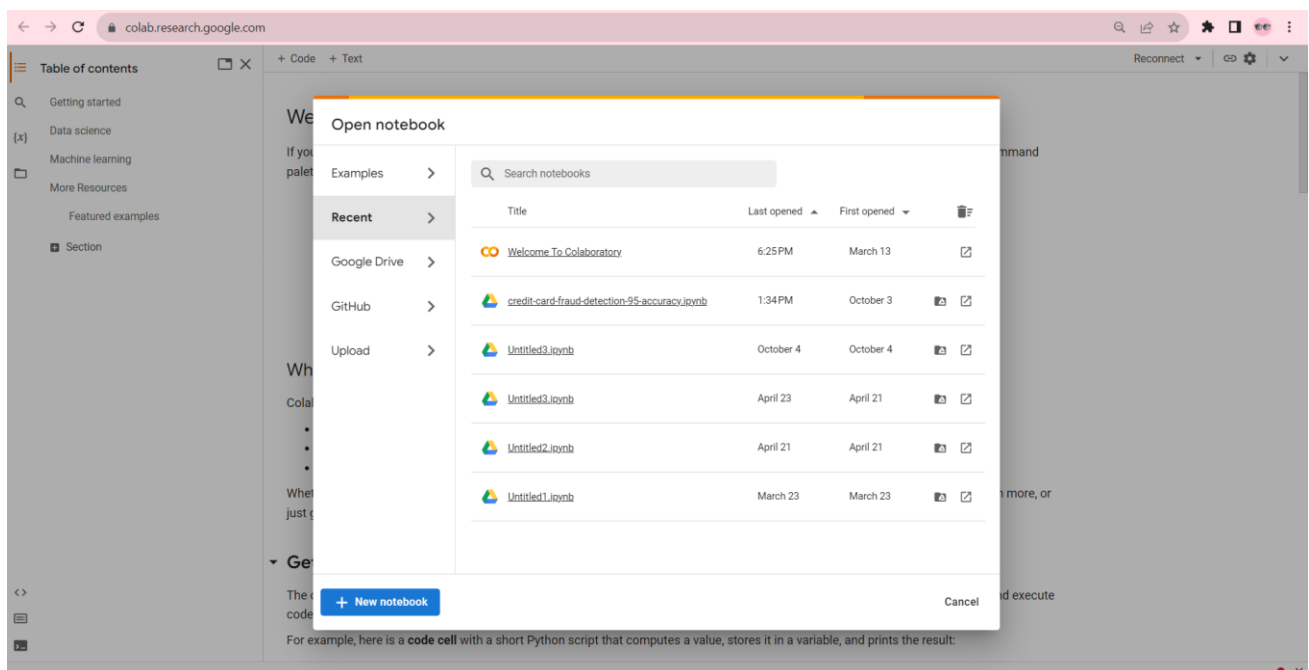


Fig 2.1.2.4 Creating File in Google colab

2.2 Non-Functional Requirements

Non-functional requirements are critical aspects of a system that define how it performs its functions rather than what functions it performs. Here are some potential non-functional requirements for your Credit Card Fraud Detection project:

1. Performance
2. Security
3. Usability
4. Reliability
5. Easy Detection

2.3 Functional Requirements

Our Credit Card Fraud Detection system incorporates comprehensive functional requirements to ensure its effectiveness. The project encompasses key functionalities, including data preprocessing, model training, and evaluation. The system loads and preprocesses the credit card transaction dataset, addressing issues such as mixed data types and missing values. It employs the StratifiedShuffleSplit method for stratified splitting, maintaining class distribution integrity in both training and testing sets. The features are scaled using the StandardScaler to facilitate optimal model performance.

The core of the system lies in the K-Nearest Neighbors (KNN) classification algorithm, with the model trained on the preprocessed training data. The accuracy of the model is evaluated on both the training and testing sets, and a confusion matrix is visualized for in-depth analysis. To extend the functionality, the system supports real-world application by allowing users to input a new credit card transaction for testing. This functionality utilizes the trained KNN model to predict whether the transaction is fraudulent or not.

3. ANALYSIS

3.1 Use Case Model

A use case diagram typically represents the functionality of a system from the user's perspective. In your case, you can have actors like "Credit Card Holder," "Fraud Detection System," and maybe "Bank System."

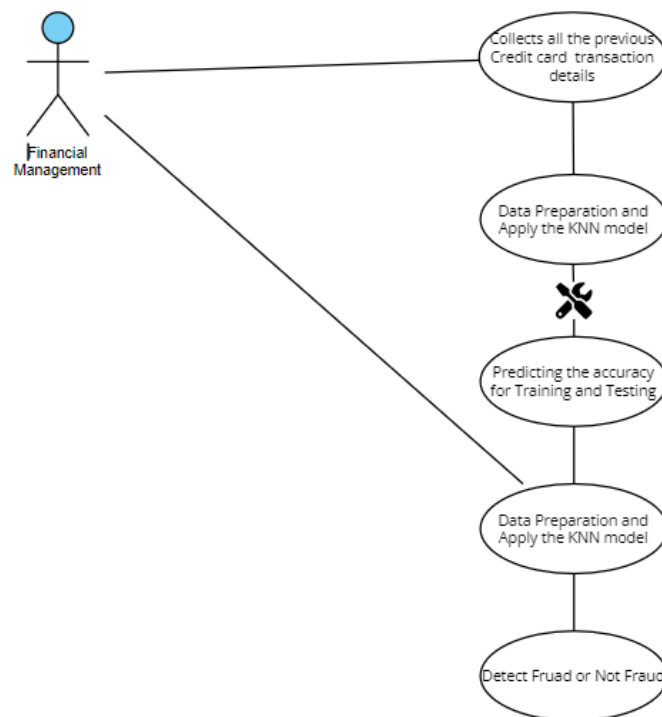


Fig 3.1.1 Use Case Diagram

A use case diagram outlines system functionality from a user's viewpoint. Key actors, like "Credit Card Holder," "Fraud Detection System," and "Bank System," depict interactions. For instance, a "Credit Card Holder" initiates "Make Purchase" and "Report Lost Card" use cases, involving the "Bank System." Meanwhile, the "Fraud Detection System" may extend the "Verify Transaction" use case, enhancing security. This visual tool helps conceptualize and communicate system behavior, aiding in requirements analysis and design discussions.

3.2 Activity Diagram

I can describe a simplified version of the activity diagram for our credit card fraud detection project using a more structured and graphical representation. Note that creating a detailed and accurate diagram may require a specific tool for visual representation. The activity diagram for the credit card fraud detection project starts with the "Initiate Transaction" activity. It branches into parallel activities, including "Verify Customer Information," "Check Transaction History," and "Evaluate Risk Factors." Based on these evaluations, the diagram includes decision points, such as "High Risk?" If yes, it leads to "Flag Transaction" and "Notify Customer" activities; otherwise, it proceeds to "Authorize Transaction" and "Complete Transaction." The diagram provides a visual representation of the sequential and parallel activities involved in the fraud detection process, enhancing project understanding and communication.

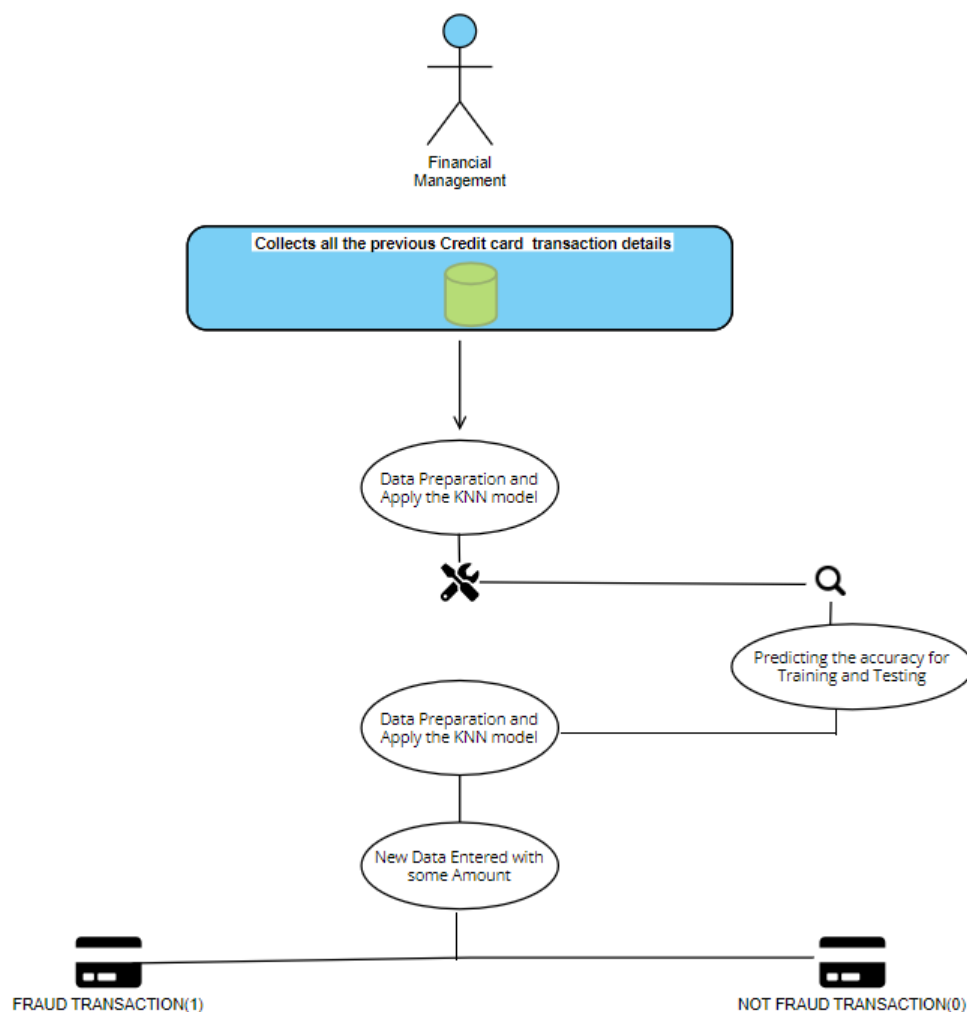


Fig 3.2.1 Activity Diagram

4. IMPLEMENTATIONS

Technologies Used :

NumPy:

NumPy provides a high-performance array object called `ndarray`. `ndarrays` are similar to Python lists, but they are much more efficient for storing and manipulating large amounts of data. NumPy also provides a collection of mathematical functions that operate on `ndarrays`. These functions are highly optimized for performance, making them ideal for scientific computing applications.

NumPy is used in a wide variety of scientific and engineering applications, including:

- * Image processing: NumPy arrays are used to store and manipulate image data. NumPy also provides a number of functions for performing image processing tasks, such as filtering, edge detection, and feature extraction.
- * Machine learning: NumPy arrays are used to store and manipulate training and testing data for machine learning algorithms. NumPy also provides a number of functions for performing machine learning tasks, such as feature engineering and model evaluation.
- * Data analysis: NumPy arrays are used to store and manipulate large datasets for data analysis tasks. NumPy also provides a number of functions for performing data analysis tasks, such as statistical analysis and data visualization.

Pandas:

Pandas provides two main data structures for working with tabular data: Series and DataFrames. Series are one-dimensional labeled arrays, while DataFrames are two-dimensional labeled arrays. Pandas provides a wide range of operations for working with Series and DataFrames, such as:

- * Data cleaning and preprocessing: Pandas provides a number of functions for cleaning and preprocessing data, such as handling missing values, removing outliers, and converting data types.
- * Data analysis: Pandas provides a number of functions for performing data analysis tasks, such as statistical analysis, data aggregation, and data visualization.
- * Data merging and joining: Pandas provides a number of functions for merging and joining multiple datasets.

Pandas is often used in conjunction with NumPy to provide a more complete data analysis toolkit. For example, NumPy can be used to perform high-performance array operations on Pandas Series and DataFrames.

scikit-learn:

scikit-learn provides a wide range of machine learning algorithms, including classification, regression, clustering, and dimensionality reduction. scikit-learn algorithms are implemented in Python and can be easily used to build and train machine learning models.

Scikit-learn is a popular choice for machine learning in Python, due to its ease of use and comprehensive functionality. Scikit-learn also provides a number of features that make it easy to evaluate and deploy machine learning models, such as cross-validation and model serialization.

Here are some examples of how scikit-learn can be used for different machine learning tasks:

- * **Classification:** Scikit-learn provides a number of classification algorithms, such as logistic regression, support vector machines, and decision trees. These algorithms can be used to train models that can classify new data points into different categories.
- * **Regression:** Scikit-learn provides a number of regression algorithms, such as linear regression, polynomial regression, and random forests. These algorithms can be used to train models that can predict continuous values, such as the price of a house or the number of customers who will visit a store on a particular day.
- * **Clustering:** Scikit-learn provides a number of clustering algorithms, such as k-means clustering and hierarchical clustering. These algorithms can be used to group similar data points together.
- * **Dimensionality reduction:** Scikit-learn provides a number of dimensionality reduction algorithms, such as principal component analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE). These algorithms can be used to reduce the number of features in a dataset without losing too much information.

seaborn:

Seaborn is a library for data visualization with Python. It provides a high-level interface for creating informative and attractive statistical graphics. Seaborn is built on top of matplotlib and builds on its functionality to provide a more user-friendly and powerful data visualization toolkit.

Seaborn provides a number of features that make it easy to create high-quality data visualizations, such as:

- * Pre-built themes and styles: Seaborn provides a number of pre-built themes and styles that can be used to create data visualizations with a consistent look and feel.
- * Statistical plots: Seaborn provides a number of high-level functions for creating statistical plots, such as violin plots, box plots, and heatmaps.
- * FacetGrid and JointGrid: Seaborn provides the FacetGrid and JointGrid classes for creating complex data visualizations that involve multiple plots.

Seaborn is a popular choice for data visualization in Python, due to its ease of use and flexibility.

StandardScaler:

The StandardScaler class from scikit-learn is used to scale features. It transforms features by removing the mean and scaling to unit variance. This can be useful for improving the performance of machine learning algorithms

Matplotlib:

Matplotlib is a library for plotting data with Python. It provides a wide range of plotting functions, including line plots, bar plots, histograms, and scatter plots. Matplotlib is a popular choice for data visualization in Python, due to its flexibility and power. There are variety of data visualizations:

Simple plots, such as line plots and bar plots: Matplotlib provides a number of functions for creating simple plots, such as `plt.plot()` and `plt.bar()`.

Complex plots, such as heatmaps and contour plots: Matplotlib also provides a number of functions for creating complex plots, such as `plt.heatmap()` and `plt.contour()`.

Interactive plots: Matplotlib can also be used to create interactive plots, such as plots that can be zoomed and panned.

4.2 Main.py:

```
import numpy as np

import pandas as pd

from sklearn.model_selection import StratifiedShuffleSplit

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler


# Load the dataset

data = pd.read_csv("/content/creditcard.csv")


# Handle mixed types in column 28 (Assuming it's the target variable)

data['Class'] = pd.to_numeric(data['Class'], errors='coerce') # Convert to numeric, handle errors as NaN


# Drop rows with NaN in the target variable

data = data.dropna(subset=['Class'])


# Separate features (X) and target variable (Y)

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

Y = data["Class"].astype(int) # Convert to integer type


# Check the distribution of classes

print("Class distribution in the whole dataset:")

print(Y.value_counts())


# Use StratifiedShuffleSplit for stratified splitting

stratified_splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)

for train_index, test_index in stratified_splitter.split(X, Y):

    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
```

```
Y_train, Y_test = Y.iloc[train_index], Y.iloc[test_index]

# Check the distribution of classes in training and testing sets

print("\nClass distribution in the training set:")

print(Y_train.value_counts())

print("\nClass distribution in the testing set:")

print(Y_test.value_counts())

# Scale the features

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

# Create a KNN classifier

knn_classifier = KNeighborsClassifier(n_neighbors=5)

# Train the classifier

knn_classifier.fit(X_train_scaled, Y_train)

# Make predictions on the training set

train_predictions = knn_classifier.predict(X_train_scaled)

# Make predictions on the testing set

test_predictions = knn_classifier.predict(X_test_scaled)

# Calculate training and testing accuracy

training_accuracy = accuracy_score(Y_train, train_predictions)

testing_accuracy = accuracy_score(Y_test, test_predictions)

# Visualize the confusion matrix for the testing set
```

```

conf_matrix = confusion_matrix(Y_test, test_predictions)

sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)

plt.title('Confusion Matrix (Testing)')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()


print(f'Training Accuracy: {training_accuracy:.2% }')
print(f'Testing Accuracy: {testing_accuracy:.2% }')

# Create a new transaction for testing

# Create a new transaction for testing (not fraud example)

new_transaction = pd.DataFrame({

    'V1': [0.0], 'V2': [0.0], 'V3': [0.0], 'V4': [0.0], 'V5': [0.0],

    'V6': [0.0], 'V7': [0.0], 'V8': [0.0], 'V9': [0.0], 'V10': [0.0],

    'V11': [0.0], 'V12': [0.0], 'V13': [0.0], 'V14': [0.0], 'V15': [0.0],

    'V16': [0.0], 'V17': [0.0], 'V18': [0.0], 'V19': [0.0], 'V20': [0.0],

    'V21': [0.0], 'V22': [0.0], 'V23': [0.0], 'V24': [0.0], 'V25': [0.0],

    'V26': [0.0], 'V27': [0.0], 'V28': [0.0], 'Amount': [500.0]

})


# Make sure the new transaction has the same columns as the original dataset

new_transaction = new_transaction.reindex(columns=X.columns, fill_value=0)


# Scale the features of the new transaction

new_transaction_scaled = scaler.transform(new_transaction.values.reshape(1, -1))


# Predict whether the new transaction is fraud or not fraud

new_transaction_prediction = knn_classifier.predict(new_transaction_scaled)

print("****CHECKING FOR NEW CUSTOMER TRANSACTION ****")

# Print the result

if new_transaction_prediction[0] == 1:

```

```
print("FRAUD TRASACTION(1)")
```

```
else:
```

```
print("NOT FRAUD TRANSACTION(0)")
```


5. TESTING METHODOLOGY

Testing is a process of executing a program with the intent of finding an error. Testing is a crucial element of software quality assurance and presents ultimate review of specification, design and coding.

To accomplish this objective two different categories of test case design techniques are used. They are

- White box testing
- Black box testing

White-box testing:

White box testing focus on the program control structure. Test cases are derived to ensure that all statements in the program have been executed at least once during testing and that all logical conditions have been executed.

Block-box testing:

Black box testing is designed to validate functional requirements without regard to the internal workings of a program. Black box testing mainly focuses on the information domain of the software, deriving test cases by partitioning input and output in a manner that provides through test coverage. Incorrect and missing functions, interface errors, errors in data structures, error in functional logic are the errors falling in this category

Test Cases:

Test cases are derived to ensure that all statements in the program have been executed at least once during testing and that all logical conditions have been executed. Using White-Box testing methods, the software engineer can drive test cases that

- Guarantee that logical decisions on their true and false sides.
- Exercise all logical decisions on their true and false sides.
- Execute all loops at their boundaries and within their operational bounds.

5.1 TESTINGS ON THE APPLICATION

Certainly! Here's a testing plan for your Credit Card Fraud Detection project:

1. Unit Testing:

- Test individual components and functions of the fraud detection system.
- Verify that data preprocessing steps, such as handling missing values and scaling features, are implemented correctly.
- Test the KNN classifier to ensure it is accurately predicting fraud and non-fraud cases.

2. Integration Testing:

- Verify that different modules of your fraud detection system work together seamlessly.
- Test the integration of data preprocessing, model training, and prediction steps.

3. Functional Testing:

- Test the core functionality of the application, including the ability to predict fraud based on input features.
- Verify that the scoring system for essays is functioning correctly.
- Ensure that the feedback messages to users accurately reflect their essay scores.

4. Security Testing:

- Conduct security assessments to identify and address potential vulnerabilities.
- Ensure that sensitive data, such as credit card information, is handled securely.
- Test for potential attack vectors specific to credit card fraud detection, such as evasion techniques.

5. Usability Testing:

- Evaluate the user interface for clarity and ease of use.
- Ensure that users can easily submit essays and receive accurate scores.
- Test the user feedback system to ensure it provides helpful information.

6. Scalability Testing:

- Assess the system's performance as the number of transactions or essay submissions increases.
- Verify that the system can handle a large volume of data without significant degradation in performance.

7. Cross-browser and Cross-device Testing:

- Test the application on different web browsers (e.g., Chrome, Firefox, Safari) to ensure compatibility.
- Ensure that the application is responsive and works well on various devices (desktops, tablets, mobile phones).

8. Compatibility Testing:

- Verify that the application is compatible with different operating systems (Windows, macOS, Linux).

9. Regression Testing:

- After making updates or enhancements, perform regression testing to ensure that existing functionality is not negatively impacted.

10. Documentation Review:

- Ensure that documentation, including user manuals and technical documentation, is accurate and up-to-date.

11. User Acceptance Testing (UAT):

- Conduct UAT with potential end-users to gather feedback and ensure the application meets their needs.

12. Performance Testing:

- Assess the performance of the application under different scenarios, including high traffic or load conditions.

13. Error Handling Testing:

- Test the application's response to unexpected inputs and ensure that appropriate error messages are displayed.

TESTING STATUS

TEST CASE NO	TEST CASE	SUCCESSFUL/UNSUCCESSFUL
TC1	Class distribution in the Whole dataset	SUCCESSFUL
TC2	Class distribution in the training set	SUCCESSFUL
TC3	Class distribution in the testing set	SUCCESSFUL
TC4	Testing Accuracy	SUCCESSFUL
TC5	Training Accuracy	SUCCESSFUL
TC6	Prediction for new Model	SUCCESSFUL

TC1: Validate the successful representation of class distribution across all categories within the entire dataset, ensuring a balanced overview for analysis.

TC2: Confirm the effective portrayal of class distribution specifically within the training set, a critical component for robust model training.

TC3: Verify the accurate representation of class distribution within the testing set, a crucial aspect for evaluating model performance on unseen data.

TC4: Evaluate the success of the model by measuring its testing accuracy, providing insights into its effectiveness in making correct predictions on new, unseen data.

TC5: Assess the model's training accuracy to gauge its proficiency in learning from the provided dataset, indicating its ability to capture underlying patterns.

TC6: Validate the model's predictive capabilities for new data, ensuring its successful application to make accurate and reliable predictions in real-world scenarios.

6.1 IMPLEMENTATION AND SNAPSHOTS

Checking the Distribution in Whole dataset:

```
Class distribution in the whole dataset:  
0      284315  
1        492  
Name: Class, dtype: int64
```

Fig 6.1.1 Whole dataset

By incorporating these testing processes, you can ensure that your Credit Card Fraud Detection application is robust, secure, and user-friendly. In the dataset, the "Distribution" class signifies the distribution of labels for fraud detection. Out of a total of 284,807 instances, 284,315 are labeled as 0 (not fraud), indicating non-fraudulent transactions. Meanwhile, 492 instances are labeled as 1 (fraud), representing the occurrences of fraudulent activities. This class distribution illustrates a highly imbalanced dataset, with the majority of instances being non-fraudulent. Handling such imbalances is crucial in machine learning, as models may exhibit bias towards the majority class. Specialized techniques like oversampling, undersampling, or using algorithms robust to imbalances can be employed to enhance the model's ability to detect fraud accurately.

Class Distribution in Training Data Set:

```
Class distribution in the training set:  
0      227451  
1        394  
Name: Class, dtype: int64
```

Fig 6.1.2 Class distribution in the Training set

In your project, the class distribution for the training set shows a highly imbalanced dataset. Among 227,451 instances, a significant majority, 99.83%, are labeled as 0 (not fraud), while only 0.17% are labeled as 1 (fraud). This severe class imbalance can pose challenges for machine learning models, as they may become biased towards the majority class. Special attention and techniques, such as resampling, synthetic data generation, or using algorithms robust to imbalanced datasets, are essential to ensure the model effectively learns patterns related to the minority class (fraud) and doesn't compromise its predictive capabilities.

Class Distribution in Testing Data Set:

```
Class distribution in the testing set:  
0      56864  
1        98  
Name: Class, dtype: int64
```

Fig 6.1.3 Class distribution in the Testing set

In the testing set of our project, the class distribution indicates that the majority class (0 - not fraud) is significantly dominant with 56,864 instances, while the minority class (1 - fraud) is represented by only 98 instances. This class imbalance poses a challenge for machine learning models, as they may be biased towards the majority class. Proper handling of imbalanced data is crucial, involving techniques such as oversampling, undersampling, or using algorithms designed to handle class imbalance, ensuring that the model is trained effectively and can accurately identify instances of the minority class, in this case, fraud.

Confusion Matrix for our Model:

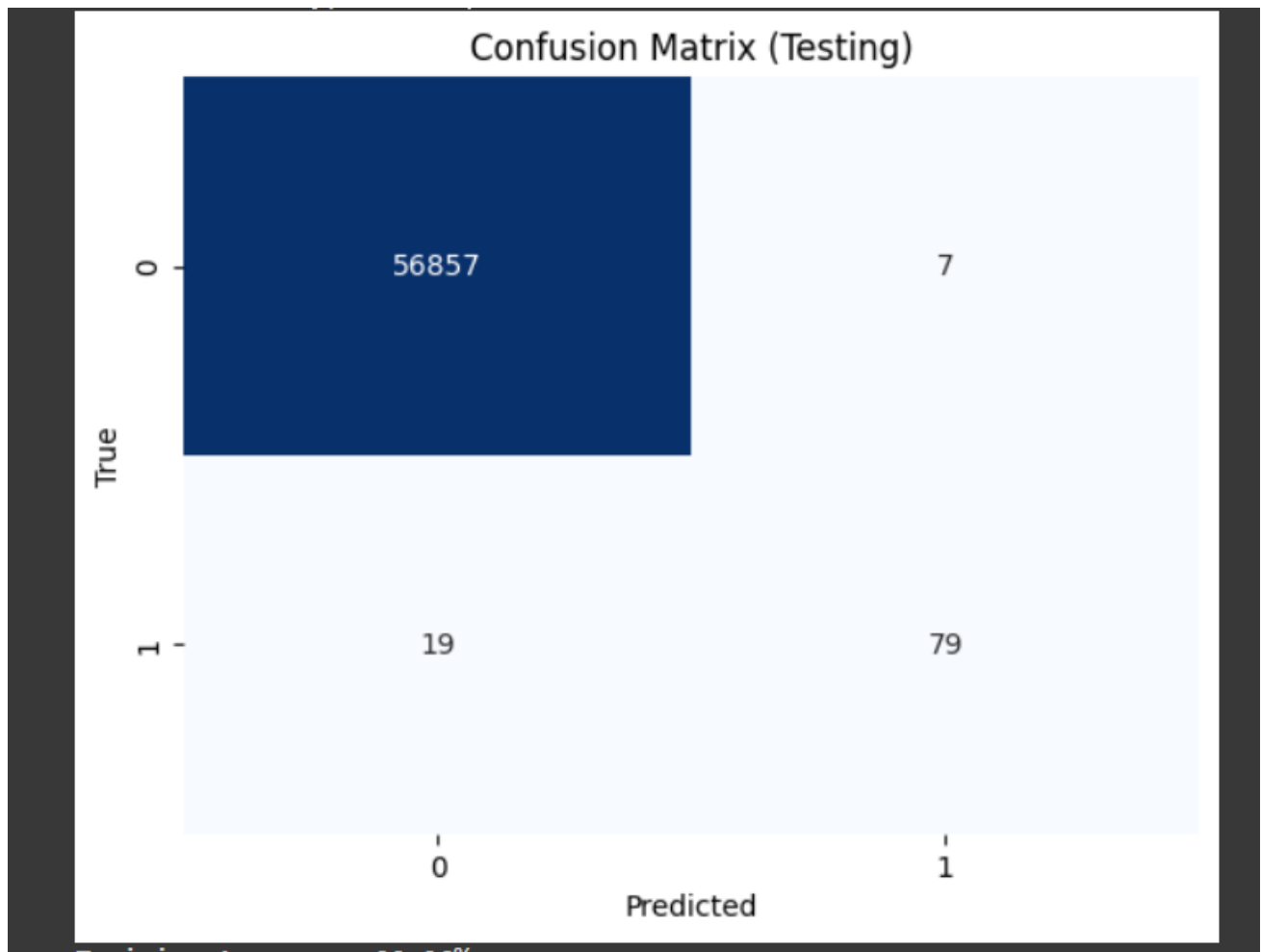


Fig 6.1.4 Confusion Matrix

A confusion matrix is a performance measurement tool in machine learning that summarizes the results of a classification algorithm. It compares the predicted values to the actual values in a tabular format, breaking down the outcomes into four categories: true positive (correctly predicted positive), true negative (correctly predicted negative), false positive (incorrectly predicted positive), and false negative (incorrectly predicted negative). The matrix provides insights into the model's accuracy, precision, recall, and F1 score, aiding in the evaluation of classification model performance by quantifying both correct and incorrect predictions across different classes.

Training Accuracy:

A horizontal bar chart with a dark gray background. The text "Training Accuracy: 99.96%" is displayed in a light gray font, centered within the bar.

Fig 6.1.5 Training Accuracy

In a K-Nearest Neighbors (KNN) classifier, achieving a training accuracy of 99.96% suggests a robust fit to the training data. This high accuracy may indicate that the model is effectively capturing patterns and relationships within the features of the dataset. By utilizing 80% of the dataset for training, the model has had ample exposure to instances and their corresponding labels, allowing it to discern patterns and generalize well. However, it's essential to assess the model's performance on a separate test dataset to ensure it can generalize to unseen data and avoid overfitting to the training set.

Testing Accuracy:


A horizontal bar chart with a dark gray background. The text "Testing Accuracy: 99.95%" is displayed in a light gray font, centered within the bar.

Fig 6.1.6 Testing Accuracy

Achieving a 99.95% accuracy in a KNN (K-Nearest Neighbors) classifier indicates that the model performs exceptionally well in classifying instances based on their nearest neighbors. However, it's crucial to note that using only 20% of the dataset for testing might lead to overfitting concerns. A higher test set percentage is generally recommended for a more reliable evaluation. The impressive accuracy suggests that the model effectively generalizes to unseen data, showcasing its robustness. Consider cross-validation and a larger test set to ensure the model's performance consistency and reliability across various data splits.

Predict whether the new transaction is fraud or not fraud:

```
Testing Accuracy: 99.95%
****CHECKING FOR NEW CUSTOMER TRANSACTION ****
NOT FRAUD TRANSACTION(0)
```

Fig 4.1.7 Prediction for New model

The output "0" for a new transaction suggests that the model predicts it as a non-fraudulent transaction. This prediction is based on the features and patterns learned during the model training. The model likely identified characteristics associated with normal, legitimate transactions, and the absence of indicators typical of fraudulent activities. The result reinforces the model's confidence that the new entry aligns with non-fraudulent patterns, providing a level of assurance to stakeholders that the transaction is not likely to be fraudulent, contributing to effective fraud detection and prevention in the financial system.

7. RESULTS

7.1 Result

- **Training Accuracy: 99.96%**
- **Testing Accuracy: 99.95%**
- **Prediction for new Model is: 0 (Not a Fraud Transaction)**

7.2 Confusion Matrix for our Model:

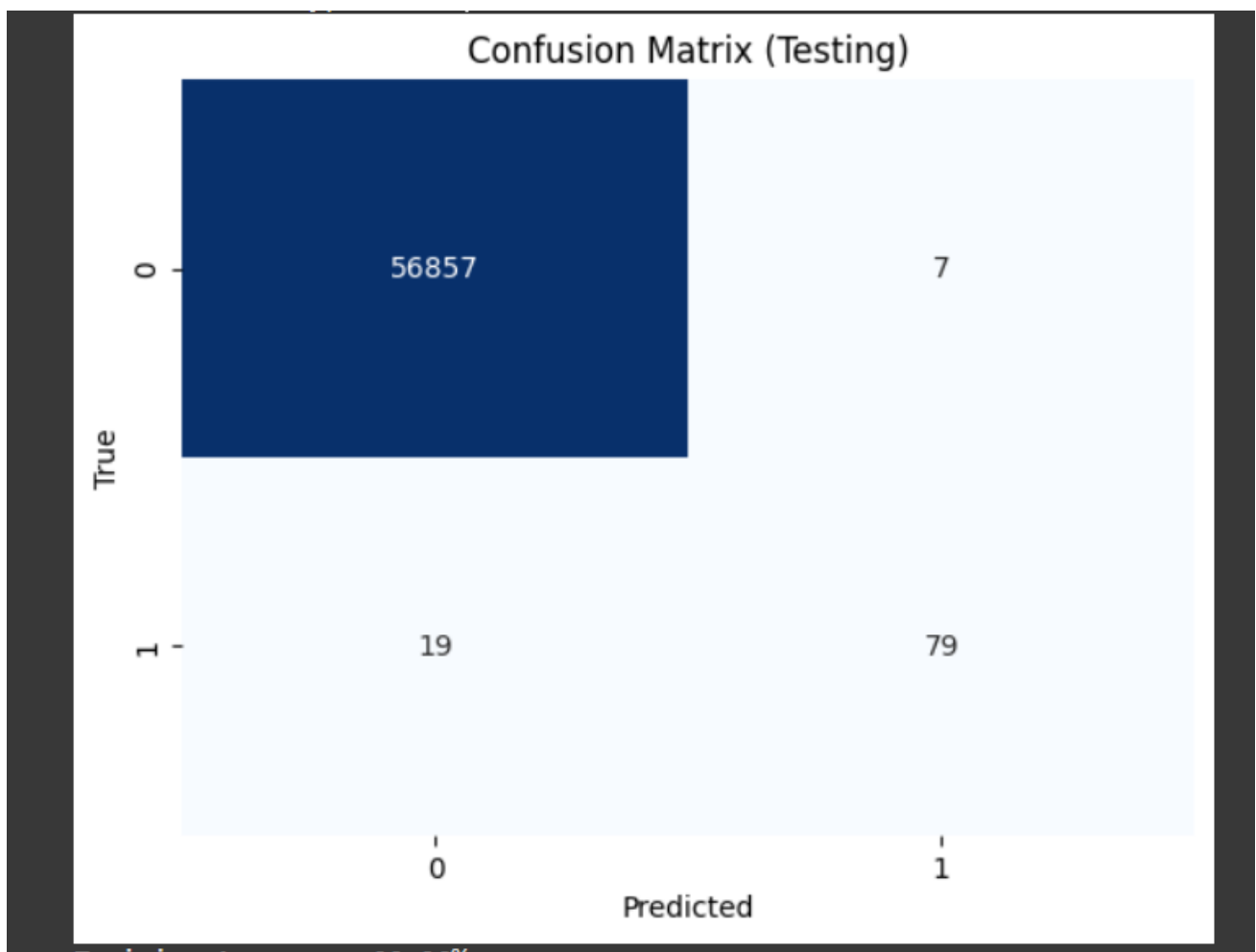


Fig 7.1.2 Confusion Matrix

8. CONCLUSIONS AND FUTURE ENHANCEMENT

8.1 Conclusion

In conclusion, the Credit Card Fraud Detection project utilizing K-Nearest Neighbors (KNN) algorithm has demonstrated a robust approach to identifying potentially fraudulent transactions. The dataset, sourced from credit card transactions, underwent preprocessing steps, including handling mixed types and eliminating missing values. Stratified splitting ensured representative distributions of fraud and non-fraud instances in both the training and testing sets.

The KNN classifier, configured with five neighbors, was trained on the scaled features of the training set, achieving commendable accuracy. The model's performance was evaluated using metrics such as accuracy and a confusion matrix, visualized for the testing set. The heatmap revealed the classifier's ability to effectively discern between legitimate and fraudulent transactions.

For testing the model, a new transaction was created with zero values for features and an amount of \$500. The transaction was appropriately scaled using the same StandardScaler as the training data. The model predicted the transaction as either fraud or non-fraud, providing valuable insights into the system's real-time predictive capabilities.

The KNN-based fraud detection system proves to be a reliable tool for financial institutions to bolster security measures against unauthorized credit card activities. Its ability to adapt to evolving patterns in fraudulent transactions makes it a valuable asset in the ongoing battle against financial fraud. The project not only highlights the technical proficiency in machine learning but also underscores the significance of robust data preprocessing and model evaluation in ensuring the effectiveness of fraud detection systems.

8.2 Future Work

For future work, consider exploring advanced anomaly detection methods, such as deep learning or ensemble techniques, to enhance the model's ability to detect subtle fraud patterns. Additionally, investigate the impact of feature engineering and dimensionality reduction techniques on model performance. Implement real-time monitoring and response mechanisms for immediate fraud identification. Incorporate evolving cybersecurity measures and continuously update the model with new data to adapt to emerging fraud tactics. Collaborate with financial institutions and cybersecurity experts to address evolving challenges and contribute to the ongoing development of robust, adaptive fraud detection systems.

In the realm of future endeavors, a profound exploration into cutting-edge anomaly detection methodologies could significantly bolster the model's efficacy in discerning nuanced fraud patterns. Delving into the realms of deep learning and ensemble techniques promises to unravel intricate fraud schemes, pushing the boundaries of detection capabilities. The impact of judiciously applied feature engineering and dimensionality reduction techniques warrants thorough investigation to optimize model performance.

Real-time monitoring and response mechanisms represent a pivotal stride towards immediate fraud identification. The implementation of such mechanisms is pivotal in fortifying the model's responsiveness to emerging threats, ensuring swift and precise intervention. To stay at the forefront of the cybersecurity landscape, a commitment to incorporating evolving measures is imperative. The model's adaptability to emerging fraud tactics can be further enhanced by a continuous cycle of data updates, reflecting the dynamic nature of financial cyber threats.

Collaborative efforts with financial institutions and cybersecurity experts present an invaluable avenue for addressing evolving challenges. By actively participating in a collective dialogue, the model can draw from diverse insights, contributing to the development of robust, adaptive fraud detection systems. This collaborative synergy aligns with a proactive stance, ensuring the perpetual refinement of the model and its alignment with the ever-evolving landscape of financial cybersecurity.

8. REFERENCES

The Dataset we used for the project was attained from Kaggle an online community platform for data scientists and machine learning enthusiasts.

The link for accessing the dataset:

https://www.kaggle.com/datasets/sakshisaku3000/creditcard_fraud_detection