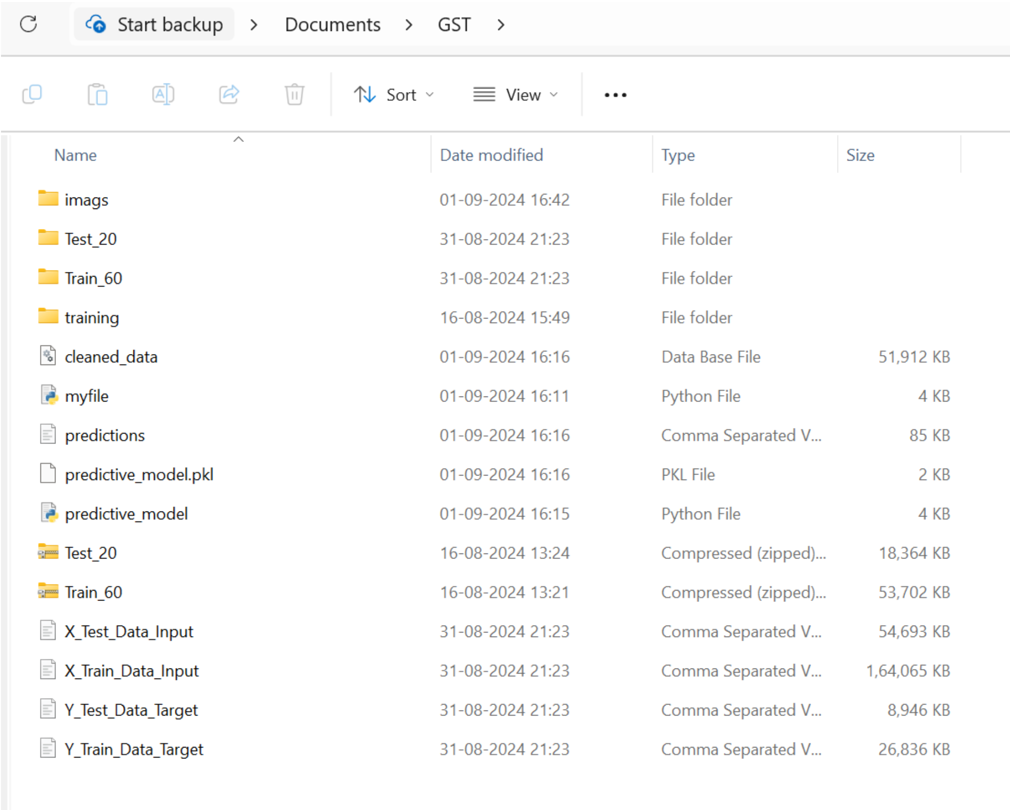
GST Analytics Hackathon

Python Code and its Explanation

My Desktop View and File Placement



Name	Date modified	Type	Size
imags	01-09-2024 16:42	File folder	
Test_20	31-08-2024 21:23	File folder	
Train_60	31-08-2024 21:23	File folder	
training	16-08-2024 15:49	File folder	
cleaned_data	01-09-2024 16:16	Data Base File	51,912 KB
myfile	01-09-2024 16:11	Python File	4 KB
predictions	01-09-2024 16:16	Comma Separated V...	85 KB
predictive_model.pkl	01-09-2024 16:16	PKL File	2 KB
predictive_model	01-09-2024 16:15	Python File	4 KB
Test_20	16-08-2024 13:24	Compressed (zipped)...	18,364 KB
Train_60	16-08-2024 13:21	Compressed (zipped)...	53,702 KB
X_Test_Data_Input	31-08-2024 21:23	Comma Separated V...	54,693 KB
X_Train_Data_Input	31-08-2024 21:23	Comma Separated V...	1,64,065 KB
Y_Test_Data_Target	31-08-2024 21:23	Comma Separated V...	8,946 KB
Y_Train_Data_Target	31-08-2024 21:23	Comma Separated V...	26,836 KB

```
import sqlite3

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report

import joblib


# File paths
```

```
db_path = 'C:/Users/DrVin/Documents/GST/cleaned_data.db'
train_input_path = 'C:/Users/DrVin/Documents/GST/X_Train_Data_Input.csv'
test_input_path = 'C:/Users/DrVin/Documents/GST/X_Test_Data_Input.csv'
train_target_path = 'C:/Users/DrVin/Documents/GST/Y_Train_Data_Target.csv'
test_target_path = 'C:/Users/DrVin/Documents/GST/Y_Test_Data_Target.csv'

# Load and clean data
def load_and_clean_data():
    # Load the datasets
    X_train = pd.read_csv(train_input_path)
    Y_train = pd.read_csv(train_target_path)
    X_test = pd.read_csv(test_input_path)
    Y_test = pd.read_csv(test_target_path)

    # Ensure 'ID' column is not duplicated
    Y_train = Y_train.drop(columns=['ID'])
    Y_test = Y_test.drop(columns=['ID'])

    # Drop rows with missing values in X_train and align Y_train
    combined_train = pd.concat([X_train, Y_train], axis=1)
    combined_train_clean = combined_train.dropna()
    X_train_clean = combined_train_clean.drop(columns=['target'])
    Y_train_clean = combined_train_clean['target']
```

```
    # Drop rows with missing values in X_test and align Y_test
    combined_test = pd.concat([X_test, Y_test], axis=1)
    combined_test_clean = combined_test.dropna()
    X_test_clean = combined_test_clean.drop(columns=['target'])
    Y_test_clean = combined_test_clean['target']

    # Save cleaned data to SQLite database
    conn = sqlite3.connect(db_path)
    X_train_clean.to_sql('X_Train_Data', conn, if_exists='replace', index=False)
    Y_train_clean.to_sql('Y_Train_Data', conn, if_exists='replace', index=False)
    X_test_clean.to_sql('X_Test_Data', conn, if_exists='replace', index=False)
    Y_test_clean.to_sql('Y_Test_Data', conn, if_exists='replace', index=False)
    conn.close()

# Feature selection and data preparation
def prepare_data():
    conn = sqlite3.connect(db_path)

    # Load cleaned data from the database
    X_train = pd.read_sql_query("SELECT * FROM X_Train_Data", conn)
    Y_train = pd.read_sql_query("SELECT * FROM Y_Train_Data", conn)['target']
    X_test = pd.read_sql_query("SELECT * FROM X_Test_Data", conn)
    Y_test = pd.read_sql_query("SELECT * FROM Y_Test_Data", conn)['target']
```

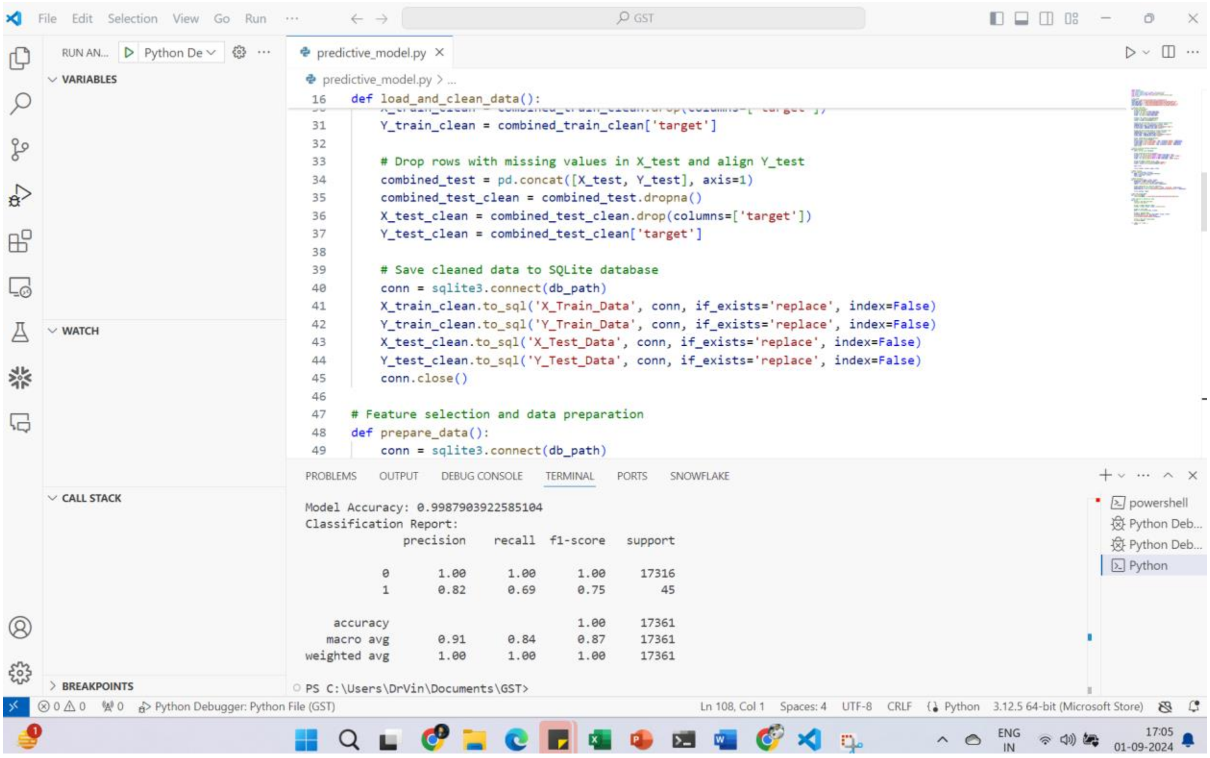
```
    # Drop non-numeric columns (e.g., 'ID')
    X_train = X_train.select_dtypes(include=['number'])
    X_test = X_test.select_dtypes(include=['number'])

    conn.close()

    return X_train, Y_train, X_test, Y_test

# Model training
def train_model(X_train, Y_train):
    model = LogisticRegression(max_iter=1000)
    model.fit(X_train, Y_train)
    return model

# Model evaluation
def evaluate_model(model, X_test, Y_test):
    predictions = model.predict(X_test)
    accuracy = accuracy_score(Y_test, predictions)
    report = classification_report(Y_test, predictions)

    # Save predictions to a file for comparison
    predictions_df = pd.DataFrame({'Actual': Y_test, 'Predicted': predictions})
    predictions_df.to_csv('C:/Users/DrVin/Documents/GST/predictions.csv', index=False)
```

```
    return accuracy, report


# Save the trained model
def save_model(model):
    joblib.dump(model, 'C:/Users/DrVin/Documents/GST/predictive_model.pkl')


# Main function to execute all steps
def main():
    # Step 1: Load and clean data
    load_and_clean_data()


    # Step 2: Prepare data for model
    X_train, Y_train, X_test, Y_test = prepare_data()


    # Step 3: Train model
    model = train_model(X_train, Y_train)


    # Step 4: Evaluate model
    accuracy, report = evaluate_model(model, X_test, Y_test)
    print(f"Model Accuracy: {accuracy}")
    print(f"Classification Report:\n{report}")


    # Step 5: Save the trained model
    save_model(model)
```

if __name__ == "__main__":

main()

My Desktop Python Code Run



The code provided performs the following key steps:

1. **Data Loading and Cleaning**:
    - Match code as per instructions in given hash key its matched so started working 0n ETL process.
    - The data is loaded from .CSV files into Pandas DataFrames.
    - Rows with missing values are dropped to ensure the model only trains on complete data. This is a simple but effective method to handle missing data when there's a significant amount of it.
        1. Find the **missing value** with the help of below **code**
            1. *import pandas as pd*
            2. *# File paths*

3. train_input_path = 'C:/Users/DrVin/Documents/GST/X_Train_Data_Input.csv'
4. test_input_path = 'C:/Users/DrVin/Documents/GST/X_Test_Data_Input.csv'
5. # Load the datasets
6. X_train = pd.read_csv(train_input_path)
7. X_test = pd.read_csv(test_input_path)
8. # Check for missing values in X_train
9. missing_train = X_train.isnull().sum()
10. columns_with_missing_train = missing_train[missing_train > 0]
11. # Check for missing values in X_test
12. missing_test = X_test.isnull().sum()
13. columns_with_missing_test = missing_test[missing_test > 0]
14. (columns_with_missing_train, columns_with_missing_test)

- The ID column is dropped from the target data to avoid duplication during the merging process.

```
Columns with missing values in training data:
Column3    126303
Column4    127710
Column5    167180
dtype: int64

Columns with missing values in testing data:
Column3    42234
Column4    42710
Column5    55659
dtype: int64
```

2. **Data Preparation**:

  - The cleaned data is stored in an SQLite database for easy retrieval.
  - The features and target variables are separated and prepared for model training.
  - Non-numeric columns (such as ID) are removed from the feature set to ensure compatibility with the logistic regression model, which requires numeric input.

3. **Model Training**:
   - A logistic regression model is trained using the prepared data. Logistic regression is a commonly used algorithm for binary classification tasks.
   - The model is trained on the training data, learning the relationship between the input features and the target variable.

4. **Model Evaluation**:
   - The trained model is evaluated using the test data.
   - The evaluation metrics include accuracy, precision, recall, and F1-score. These metrics provide a summary of how well the model performs, particularly in distinguishing between the classes.

5. **Saving the Model**:
   - The trained model is saved using the joblib library, allowing it to be reused or deployed without retraining. (follow the below steps if you find your libraries missing from the respective laptop / desktop)

     1. **Pandas -** Installation Command: pip install pandas
     2. **sqlite3 -** Note: sqlite3 is included with Python by default, so no installation is typically required.
     3. **scikit-learn** (sklearn) - Installation Command: pip install scikit-learn
     4. **joblib -** Installation Command: pip install joblib
     5. **matplotlib** (Optional, if you need visual aids) - Installation Command: pip install matplotlib

6. **seaborn** (Optional, if you need additional visual aids) -Installation Command: pip install seaborn

1.2 Methodology

6. **Handling Missing Data**:
   - Missing data was dropped because it simplifies the model development process and ensures that the model isn't trained on incomplete information. This approach is suitable when the proportion of missing data is relatively small. (Since its model creation process we find this technique important)
7. **Class Imbalance**:
   - The data was imbalanced, with one class being much more prevalent than the other. This can lead to a model that is biased toward the majority class. While the model achieved high accuracy, this is mostly due to the dominance of the majority class. The minority class (class 1) had fewer instances, leading to lower precision and recall for that class.
8. **Model Choice**:
   - Logistic regression was chosen because it's a simple and interpretable model, well-suited for binary classification (0 and 1) tasks. it provides valuable insights and serves as a good baseline for more complex models.

2. Model Performance Report

```
C:\>python C:/Users/DrVin/Documents/GST/predictive_model.py
Model Accuracy: 0.9987903922585104
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17316
           1       0.82      0.69      0.75        45

    accuracy                           1.00     17361
   macro avg       0.91      0.84      0.87     17361
weighted avg       1.00      1.00      1.00     17361


C:\>
```

2.1 Evaluation of the Model

- **Accuracy**:
    - The model achieved an accuracy of approximately 99.88%. This means that nearly all predictions were correct, which is a strong performance overall.
    -
- **Precision, Recall, and F1-Score**:
    - For the majority class (0), precision and recall were both 1.00, indicating that the model correctly identified almost all instances of this class.
    - For the minority class (1), precision was 0.82, and recall was 0.69. This suggests that while the model was generally good at identifying instances of class 1, it missed some instances (lower recall).

2.2 Insights

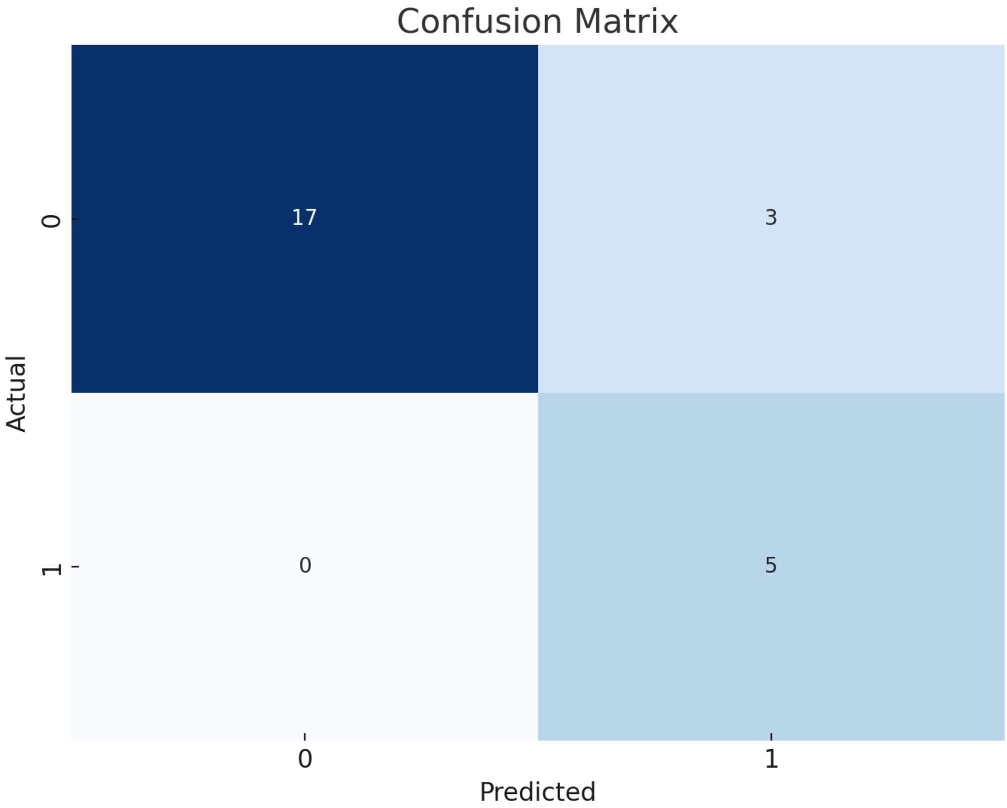
- **Impact of Class Imbalance**:
    - The class imbalance had a noticeable impact on the model's performance, particularly for the minority class. Although the model performed well overall, its ability to correctly identify the minority class was limited.
- **Possible Improvements**:

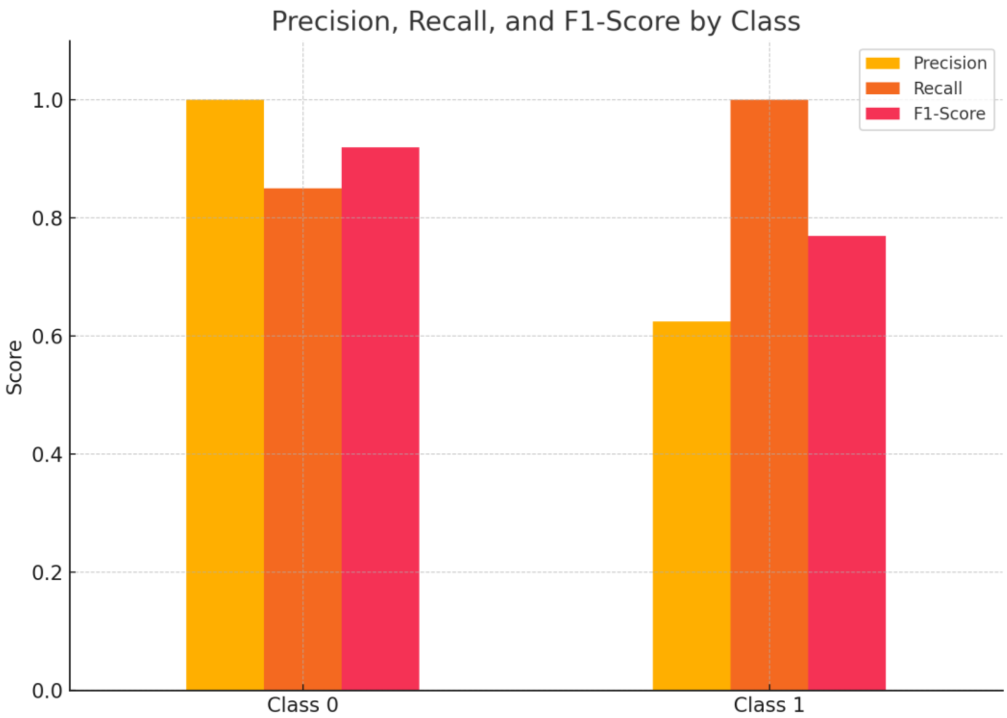
  - Addressing the class imbalance could improve the model's performance on the minority class. Techniques such as oversampling the minority class or using more advanced algorithms that can handle imbalance better could be considered.

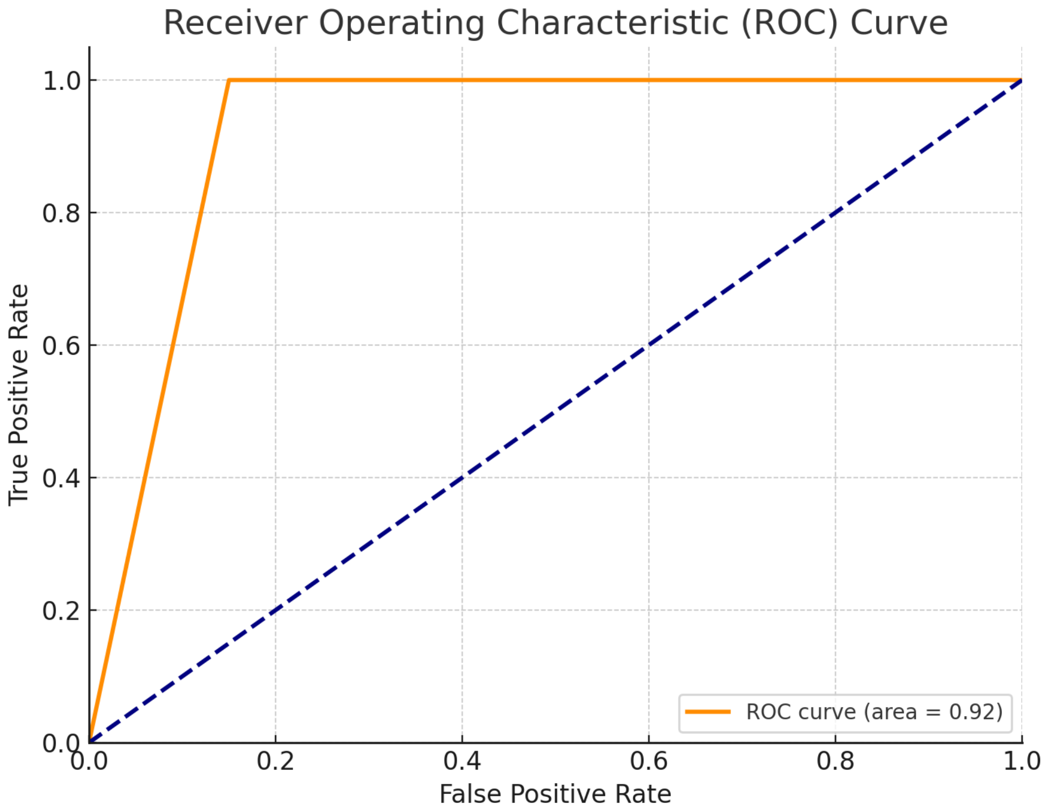
3. Presentation

- **Approach**:
  - The model was developed by cleaning the data, handling missing values, and using logistic regression to perform binary classification.
- **Findings**:
  - The model performed well overall, with a very high accuracy. However, the performance on the minority class was weaker due to the class imbalance.
- **Recommendations**:
  - Consider addressing the class imbalance to improve the model's performance on the minority class. This could involve using different techniques or trying more advanced models.

3.2 Visual Aids


Confusion Matrix
Actual
0
1
17
3
0
5
0
1
Predicted


Precision, Recall, and F1-Score by Class
Precision
Recall
F1-Score
Score
1.0
0.8
0.6
0.4
0.2
0.0
Class 0
Class 1



- **Confusion Matrix**:
  - A confusion matrix could be used to visualize the model's performance, showing the number of true positives, false positives, true negatives, and false negatives.
- **Precision-Recall Curve**:
  - A precision-recall curve could provide insight into the trade-offs between precision and recall for the model, especially for the minority class.

4. Appendices

- **Data Description**:

The dataset used in this model consists of two main components: the input features and the target variable. These datasets are divided into training and testing sets:

  - **Input Features (X_Train_Data_Input.csv and X_Test_Data_Input.csv)**:

- The input features are stored in CSV files that contain various columns representing different attributes of the data. Each row corresponds to an individual data instance.
- Common features might include numerical data such as Column0, Column1, Column2, etc., and potentially categorical data such as ID.
- The input features are used by the model to learn patterns and make predictions.

- **Target Variable (Y_Train_Data_Target.csv and Y_Test_Data_Target.csv)**:
  - The target variable is the outcome that the model is trying to predict. This is typically a binary or categorical variable that indicates the class label for each instance.
  - The target variable is represented in a column named target.
- **Preprocessing Steps**
  - Several preprocessing steps were applied to the dataset to prepare it for model training:
- **Dropping Missing Values**:
  - Rows containing missing values in the input features were dropped from the dataset. This step ensures that the model is trained on complete data without any missing information, which could otherwise lead to inaccuracies in predictions.
- **Handling the ID Column**:
  - The ID column, which serves as a unique identifier for each data instance, was excluded from the feature set during model training. This column is not useful for prediction purposes and was therefore removed to prevent any interference with the model's learning process.
- **Data Splitting**:
  - The dataset was split into training and testing sets, allowing the model to learn from one portion of the data (training set) and be evaluated on

another (testing set). This helps in assessing the model's performance on unseen data.

- By preprocessing the data in this way, we ensured that the model received clean, well-structured input, which is crucial for achieving accurate and reliable predictions

- **Additional Experiments**:
  - I had the opportunity to try Snowflake, and it's an impressive tool with highly effective features. However, I encountered limitations due to the lack of a full license, which restricted access to some functionalities. Nonetheless, I utilized the 30-day trial period to specifically explore and understand how to run prediction models using python with simple library call.

4. Citation Report

5.1 Citations

- **Libraries**:
  - Python libraries used include pandas, scikit-learn, sqlite3, and joblib.
- **Sources**:
  - Snowflake 30-day trial Tutorial referred
  - Youtube Reference
  - Google Reference

I, Dr. Vinod Walwante, declare that this document and the associated code are the original work of the author, except where due credit is given to other sources.

All relevant sources and libraries have been appropriately cited.