Configuration Manual for Honeypots and the Use of AI in keeping the IoT Systems Secure

MSc Research Project
Cyber Security

Venkat Goud Goundla
Student ID: x23152397

National College of Ireland

Supervisor: Raza UI Mustafa

National College of Ireland

MSc Project Submission Sheet

Student Name:	………………………Venkat Goud Goundla………………………………………………………		
Student ID:	…………………………………x23152397………………………………………………….……		
Programme:	…………………Cyber Security………………………	**Year:**	………2023-24..
Module:	………………………………MSc Research Project………………………………….………		
Lecturer:	……………………………… Raza ul Mustafa ……………………………………….………		
Submission Due Date:	………………………………16-09-2024…………………………………………………….………		
Project Title:	……Honeypots and the Use of AI in keeping the IoT Systems Secure …		
Word Count:	…………2583…………………	**Page Count:**	…………………17……………….……..………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: …………………Venkat Goud Goundla……………………………………………

Date: ………………………………16-09-2024………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual for HoneyPots

1 System Requirements

This whole project takes into the account three important steps,

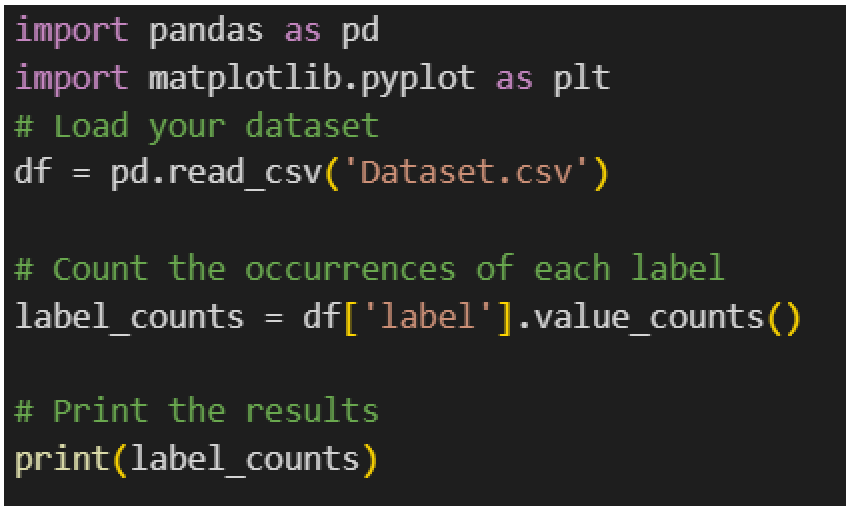
RAM: 16GB DDR2

OS: Windows 11 pro

Processor: i7 9th generation

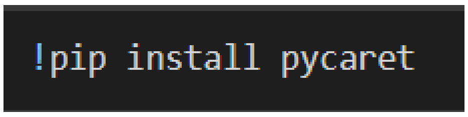
Technology required: Python, Anaconda, Spyder, Streamlit

2 Code execution



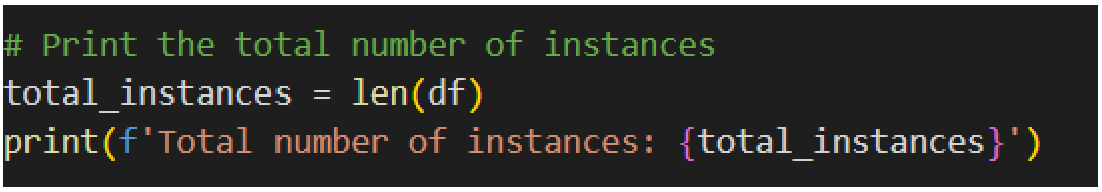
```
import pandas as pd
import matplotlib.pyplot as plt
# Load your dataset
df = pd.read_csv('Dataset.csv')

# Count the occurrences of each label
label_counts = df['label'].value_counts()

# Print the results
print(label_counts)
```

Figure1. This script first brings in pandas and matplotlib and then any dataset of CSV format with the name 'Dataset. csv'. To do this, it employs `value_counts()` to calculate the frequencies of each distinct label in the 'label' column and saves the result to `label_counts`. A script is then used which prints these counts on the console.



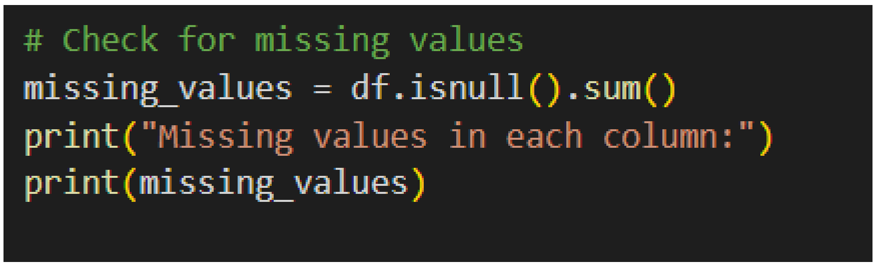
```
!pip install pycaret
```

Figure2. This command creates PyCaret and makes PyCaret usable in your python environment for features jobs.



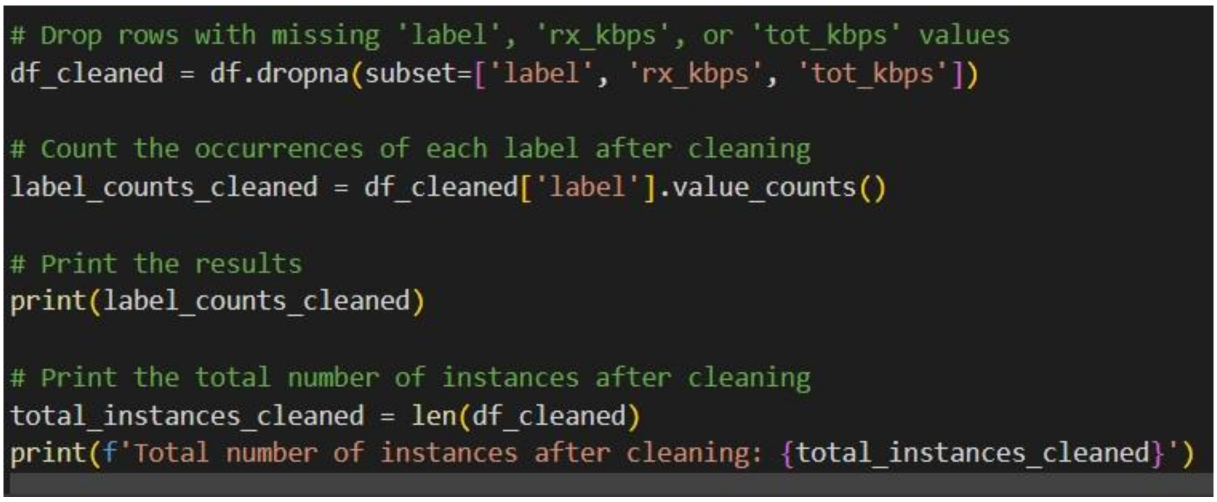
```
# Print the total number of instances
total_instances = len(df)
print(f'Total number of instances: {total_instances}')
```

Figure3. This script calculates the amount of columns, or instances in the frame referred to as `df variable`, then stores the decision within the variable `total_instances`. It then outputs a message with the number of instances in `df`, which gives the count of all entries/records of the DataFrame.



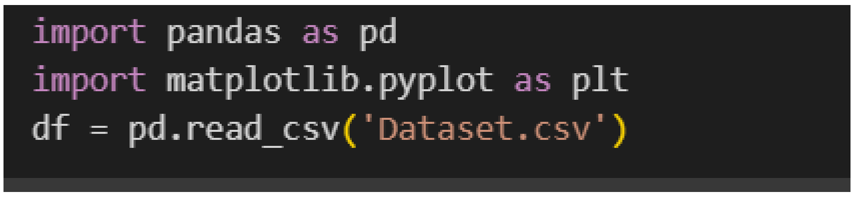
```python
# Check for missing values
missing_values = df.isnull().sum()
print("Missing values in each column:")
print(missing_values)
```

Figure4. This code counts the missing values of all the columns in `df` and prints them to the console.



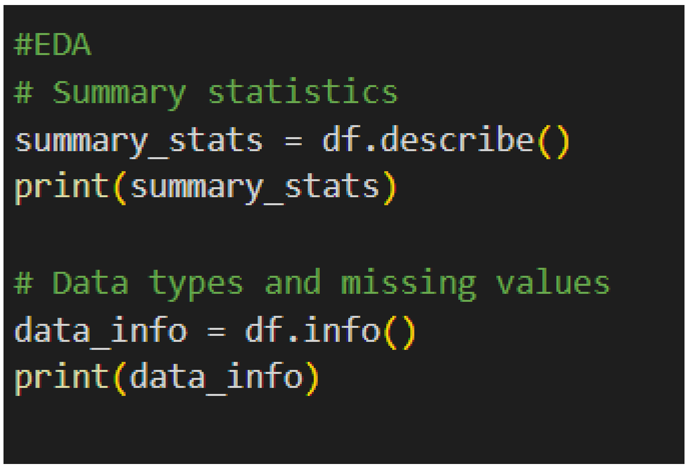
```python
# Drop rows with missing 'label', 'rx_kbps', or 'tot_kbps' values
df_cleaned = df.dropna(subset=['label', 'rx_kbps', 'tot_kbps'])

# Count the occurrences of each label after cleaning
label_counts_cleaned = df_cleaned['label'].value_counts()

# Print the results
print(label_counts_cleaned)

# Print the total number of instances after cleaning
total_instances_cleaned = len(df_cleaned)
print(f'Total number of instances after cleaning: {total_instances_cleaned}')
```

Figure5. This script first drops the rows for which there are missing values in any of the ‘label’, ‘rx_kbps’, or ‘tot_kbps’ columns and then it provides statistics on each of the labels in the clean DataFrame; the statistic includes the total number of rows in the DataFrame after cleaning.



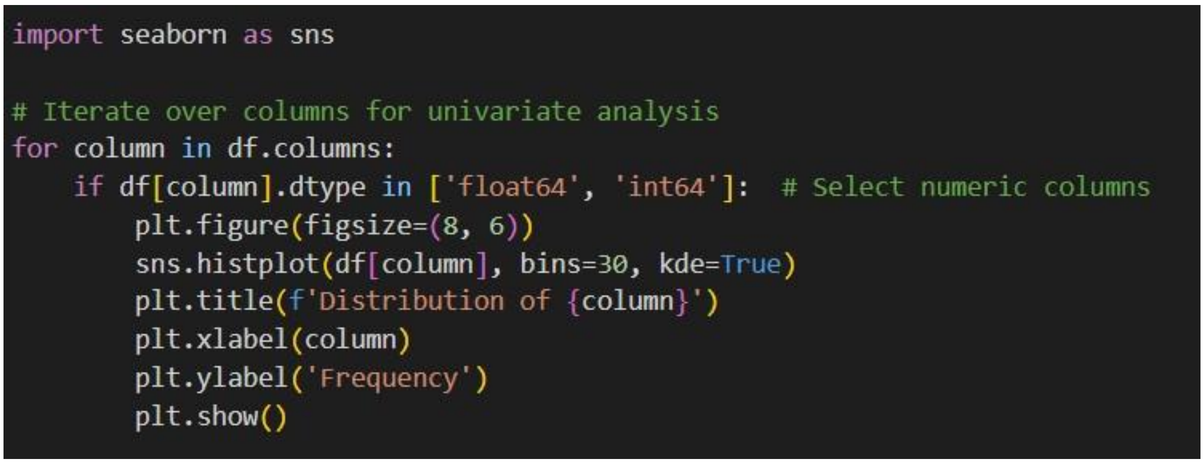
```python
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('Dataset.csv')
```

Figure6. This script incorporates pandas and matplotlib into the data analysis program, and DataFrame is named `df` to store the data read from the 'Dataset. csv' file.



```
#EDA
# Summary statistics
summary_stats = df.describe()
print(summary_stats)

# Data types and missing values
data_info = df.info()
print(data_info)
```

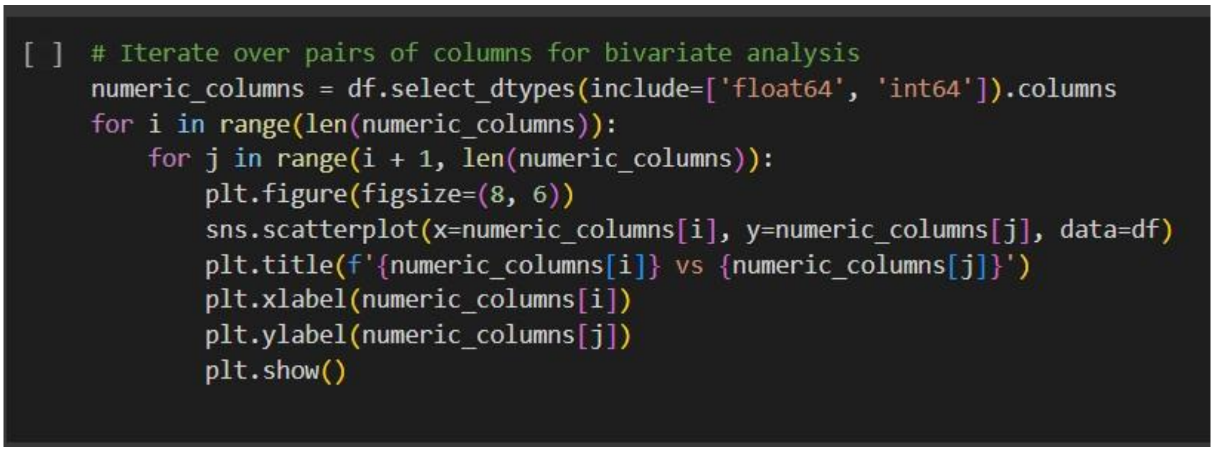
Figure7. The narrative of this script computes the basic statistical measures of the DataFrame named df which are mean, standard deviation, and quantile measurements of the data, and prints these specifications. It also uses `df. The following commands utilize the `info()` function on the data types and missing values in the DataFrame, which after running the commands, is printed out.



```
import seaborn as sns

# Iterate over columns for univariate analysis
for column in df.columns:
    if df[column].dtype in ['float64', 'int64']:  # Select numeric columns
        plt.figure(figsize=(8, 6))
        sns.histplot(df[column], bins=30, kde=True)
        plt.title(f'Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Frequency')
        plt.show()
```

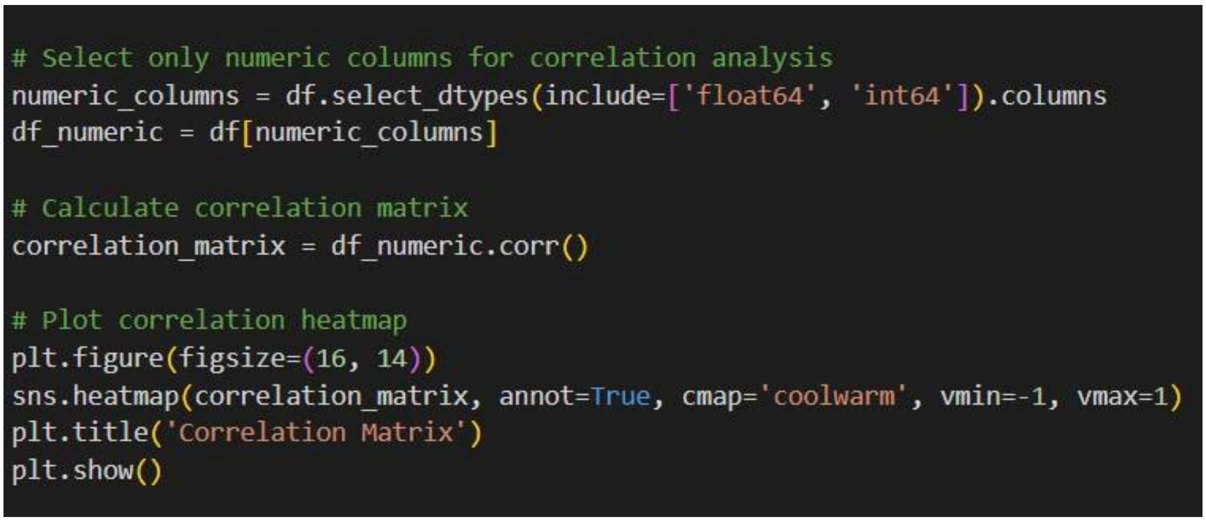
Figure8. This script splits the numeric columns of the DataFrame `df` in the original data to provide equals sub-dataframes to then perform univariate analysis on. It goes through each of the columns and if the data type of the column is numeric, whether it is float64 or int64, it will spit out a histogram as well as a density plot referred to as the Kernel Density Estimate. These plots show the values that are fall under the column top to bottom on the x-axis, the y-axis represent the frequency of these values in the column. The plots are shown

one after the other in regard to each of the numeric columns present in the dashboard..



```
[ ] # Iterate over pairs of columns for bivariate analysis
    numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
    for i in range(len(numeric_columns)):
        for j in range(i + 1, len(numeric_columns)):
            plt.figure(figsize=(8, 6))
            sns.scatterplot(x=numeric_columns[i], y=numeric_columns[j], data=df)
            plt.title(f'{numeric_columns[i]} vs {numeric_columns[j]}')
            plt.xlabel(numeric_columns[i])
            plt.ylabel(numeric_columns[j])
            plt.show()
```

Figure9. This script does the bivariate analysis on the pairs of Numeric columns in DataFrame which is named as `df`. Starting from selecting all numeric columns, it goes through each pair of the columns in turn. For each pair, it draws a scatter plot which helps in understanding how one column is proportional to the other, where the column values are displayed on one axis –X and the other column values on the other axis-Y. For each of the scatter plots, they are presented with individuality to depict the relationship or trend of the numbers in the two numeric columns.



```
# Select only numeric columns for correlation analysis
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
df_numeric = df[numeric_columns]

# Calculate correlation matrix
correlation_matrix = df_numeric.corr()

# Plot correlation heatmap
plt.figure(figsize=(16, 14))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()
```

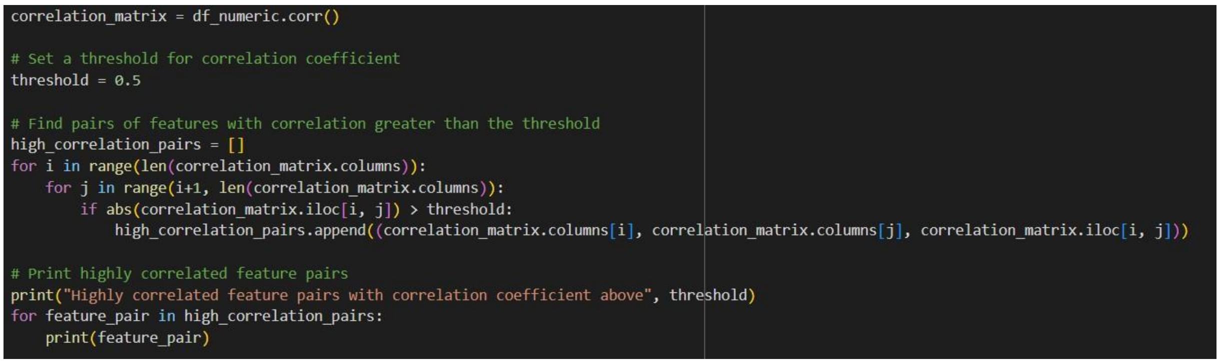
Figure10. This script performs a correlation analysis on the numeric columns in the DataFrame `df`:This script performs a correlation analysis on the numeric columns in the DataFrame `df`:

1. Select Numeric Columns: It then subsets ONLY the numeric columns of `df` to a new DataFrame refersed to as `df_numeric`.

2. Calculate Correlation Matrix: It calculates the correlation matrix for `df_numeric’ that measure the degree of the linear relationship between two numeric columns.

3. Plot Correlation Heatmap: To elaborate, it calls upon `seaborn` to create a heatmap with respect to the correlation coefficients along with annotations of the absolute correlation coefficient values. In the heatmap, it uses the ‘coolwarm’ option that uses a color gradient

with -1 as the smallest value, indicating a strong negative correlation and 1 as the largest value, indicating strong positive correlation. The plot is then depicted along the title 'Correlation Matrix'.



```
correlation_matrix = df_numeric.corr()

# Set a threshold for correlation coefficient
threshold = 0.5

# Find pairs of features with correlation greater than the threshold
high_correlation_pairs = []
for i in range(len(correlation_matrix.columns)):
    for j in range(i+1, len(correlation_matrix.columns)):
        if abs(correlation_matrix.iloc[i, j]) > threshold:
            high_correlation_pairs.append((correlation_matrix.columns[i], correlation_matrix.columns[j], correlation_matrix.iloc[i, j]))

# Print highly correlated feature pairs
print("Highly correlated feature pairs with correlation coefficient above", threshold)
for feature_pair in high_correlation_pairs:
    print(feature_pair)
```

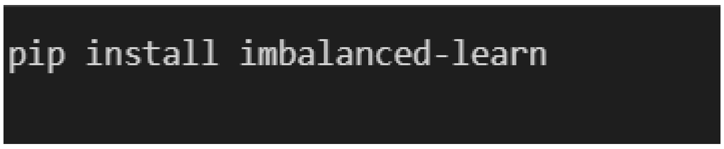
Figure11. This script identifies and lists pairs of numeric features in the DataFrame `df` that exhibit high correlation:

1. Calculate Correlation Matrix: It computes the correlation matrix for the numeric columns in `df_numeric`, showing the pairwise correlation coefficients between features.

2. Set Correlation Threshold: A threshold value of 0.5 is established to filter out significant correlations.

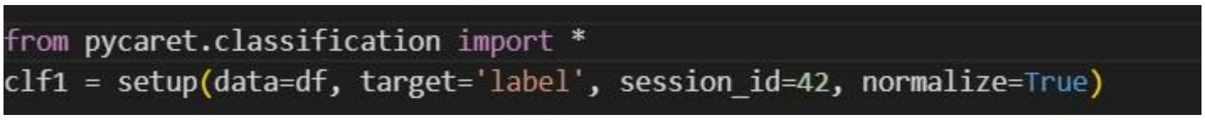
3. Find High Correlation Pairs: The script examines the upper triangle of the correlation matrix (excluding the diagonal), identifying pairs where the absolute value of the correlation coefficient exceeds the threshold. It collects and lists these pairs along with their correlation coefficients.

4. Print Results: It prints out the feature pairs with correlation coefficients above the threshold, highlighting strong relationships between those features.



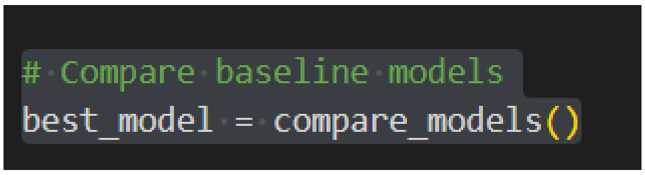
```
pip install imbalanced-learn
```

Figure12. This command installs the imbalanced-learn which is a Python library that provides modules for handling the problems of imbalanced datasets in the machine learning context.



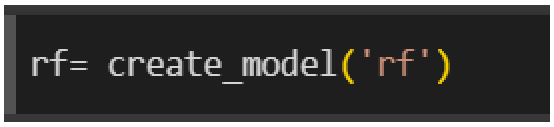
```
from pycaret.classification import *
clf1 = setup(data=df, target='label', session_id=42, normalize=True)
```

Figure13. This script sets up a PyCaret classification session by defining `$df` as the data variable and 'label' as the output variable. It uses normalization and sets a random seed making results reusable with the same data.



```
# Compare baseline models
best_model = compare_models()
```

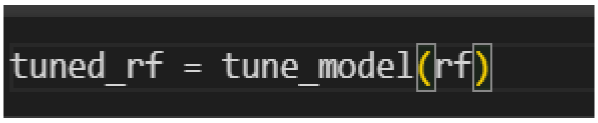
Figure14. This script assesses and puts into perspective multiple baseline classifiers through PyCaret. This function namely `compare_models()` loads and applies several machine learning models on the dataset, sorts the performance of the models based on certain measures, and returns the best performing model as per the set standard measure of performance.



```
rf= create_model('rf')
```

Figure15. This particular script establishes a Random Forest model using PyCaret.

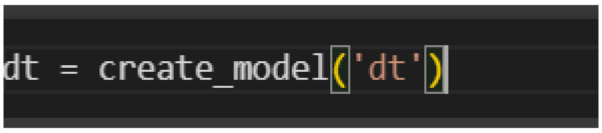
- `create_model('rf')`: To create the model the `create_model` function is called with the 'rf' string which means Random Forest. The following function averages the accuracies estimated by the Random Forest Model by calibrating the parameters to default prior to training and evaluation.



```
tuned_rf = tune_model(rf)
```

Figure16. This script tunes individual hyperparameters of the Random Forest model trained using PyCaret in this script.

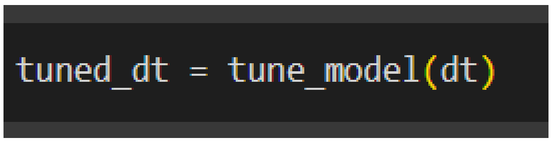
- `tune_model(rf)`: The `tune_model` function fine tunes the parameters of the targeted model which in this case is `rf`. Instead of requiring the user to provide the values, it finds the best range values of the parameter and then provides the enhanced version of the Random Forest model.



```
dt = create_model('dt')
```

Figure17This script opens up a new PyCaret object and builds a Decision Tree model.

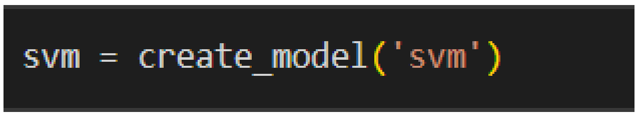
- `create_model('dt')`: By invoking the `create_model` function with the 'dt' string as an argument it prepares Decision Tree model. This function standardizes and initializes the model as to set it up for training and the subsequent assessment on the data.



```
tuned_dt = tune_model(dt)
```

Figure18. This script optimizes the hyperparameters of the Decision Tree model that is stored in the PyCaret environment under the name `dt`.

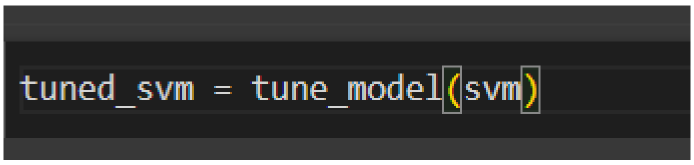
- `tune_model(dt)`: The `tune_model` function refines the set of hyperparameters of the Decision Tree model designated by `dt` to increase its efficiency. It finds out the optimal values of the parameters within the specified range and returns the refined Decision Tree model.



```
svm = create_model('svm')
```

Figure19. This script establishes a Support Vector Machine (SVM) model with the help of PyCaret.

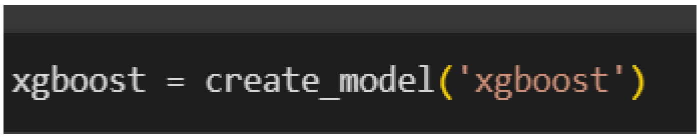
- `create_model('svm')`: Specifically, the `create_model` function is invoked with the 'svm' string and this is an acronym for Support Vector Machine. This function sets up the SVM model with default parameters, to be used for training and testing of the model on the data set.



```
tuned_svm = tune_model(svm)
```

Figure20. This script optimizes the hyperparameters of the Support Vector Machine (SVM) model to be used in PyCaret under `svm `.

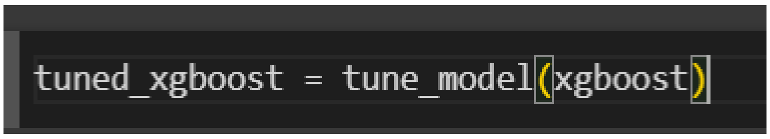
- `tune_model(svm)`: The `tune_model` function adjusts eligible hyperparameters of the stated model here as `svm` to enhance its efficiency. It looks for the best hyperparameter values from a list of potential values and tries to find the best set for the model's configuration in terms of the chosen criteria. The function returns the SVM model after optimization of hyperparameters to one of the highest degrees.



```
xgboost = create_model('xgboost')
```

Figure21. This script optimizes an XGBoost model with the help of the PyCaret library.

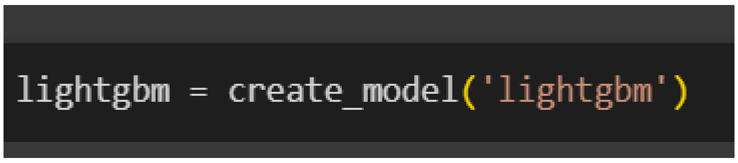
- `create_model('xgboost')`: The `create_model` function is called with the 'xgboost' string, in this case related to the XGBoost algorithm. This function sets all the parameters of the model to their initial state ready for training and evaluation on the data. XGBoost is another boosting algorithm which is commonly used and regarded as highly effective and efficient.



```
tuned_xgboost = tune_model(xgboost)
```

Figure22. This script optimises the hyperparameters of the XGBoost model recognised as `xgboost` with PyCaret.

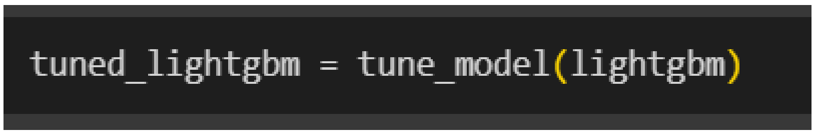
- `tune_model(xgboost)`: The `tune_model` function takes in the XGBoost model and applies the grid search on it to optimize for hyperparameters. It goes through a range of other hyperparameters looking for the value that will boost up the performance of a model. The function returns the best XGBoost model based upon the tuned hyperparameters.



```python
lightgbm = create_model('lightgbm')
```

Figure23. This script builds a LightGBM model with the PyCaret library in python.

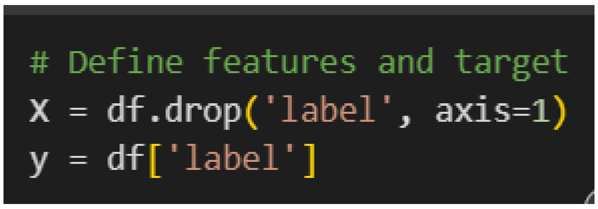
- `create_model('lightgbm')`: The `create_model` function is called with that 'lightgbm' string which is actually represents the LightGBM algorithm. This function sets the defaults of the hyperparameters to the model that is ready for learning and testing on the dataset. LightGBM is a gradient boosting framework used for solving problems efficiently, and it shows the best performance on big data.



```python
tuned_lightgbm = tune_model(lightgbm)
```

Figure24. This script tunes the LightGBM Model parameters using PyCaret specifically the $lightgbm model's hyperparameters.

- `tune_model(lightgbm)`: The `tune_model` function fine tunes various attributes of the said LightGBM model with different possible values assigned to them. It optimizes the model's parameters with regards to the evaluation criteria so as to improve the model's efficiency. On the third line of the final function, the LightGBM model with the best tuned hyperparameters is returned.

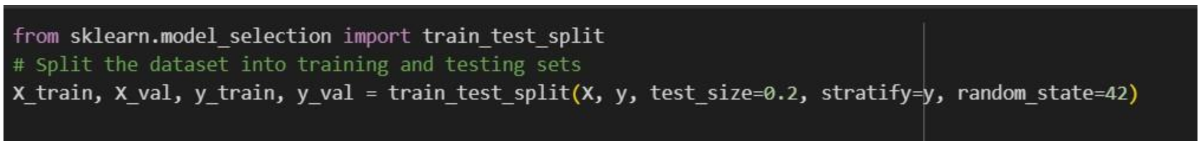


```python
# Define features and target
X = df.drop('label', axis=1)
y = df['label']
```

Figure25. This script prepares the feature matrix and target vector for a machine learning model:This script prepares the feature matrix and target vector for a machine learning model:

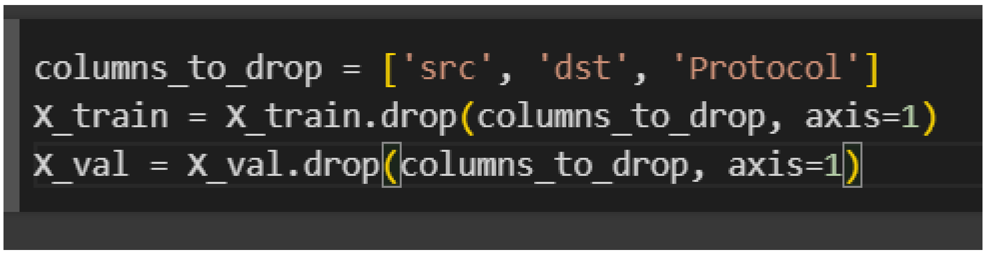
- `X = df. drop('label', axis=1)`: This line generates `X`, a DataFrame with column names same as that of the input DataFrame `df` excluding the 'label' column. The `drop` method is applied for removal of the 'label' variable, which is the outcome or the dependent variable.

- `y = df['label']`: This line produces `y`, a Series, possessing only the 'label' column from the DataFrame `df`. This column contains the data that the model will forecast; it serves as the dependent variable.



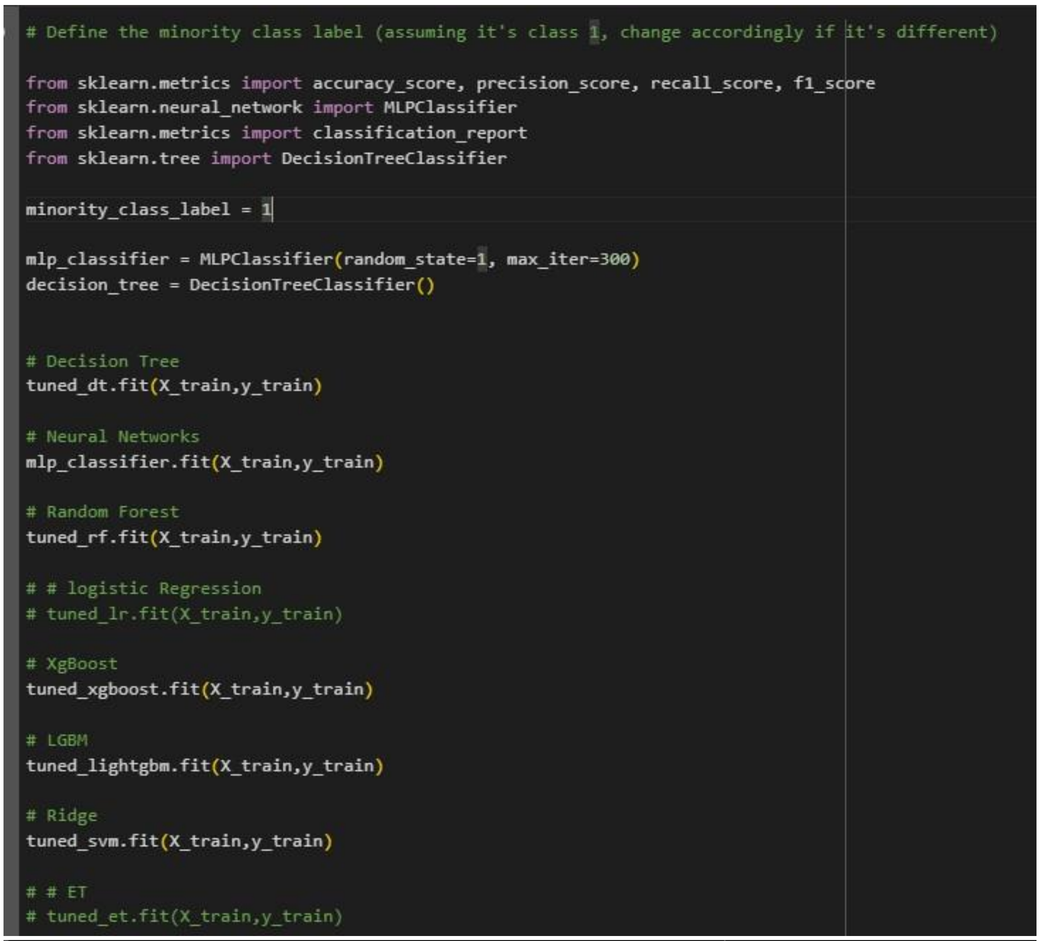
```python
from sklearn.model_selection import train_test_split
# Split the dataset into training and testing sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

Figure26.This code splits the dataset into training and testing sets:

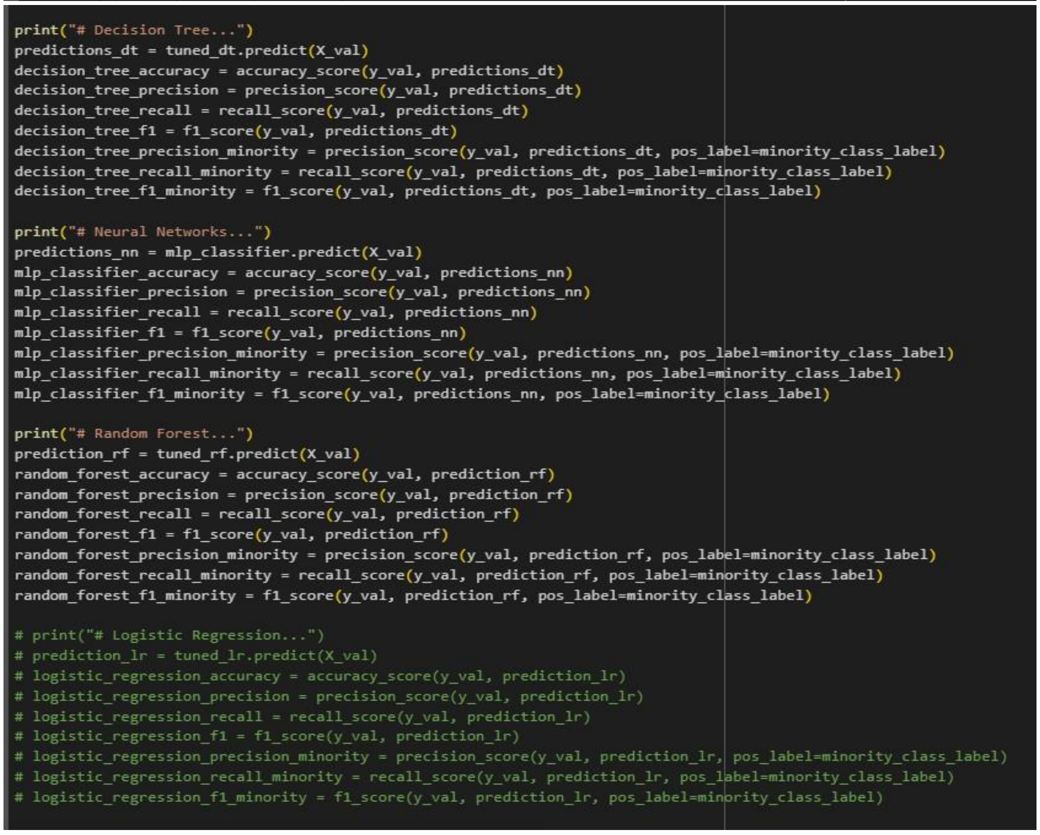


```
columns_to_drop = ['src', 'dst', 'Protocol']
X_train = X_train.drop(columns_to_drop, axis=1)
X_val = X_val.drop(columns_to_drop, axis=1)
```

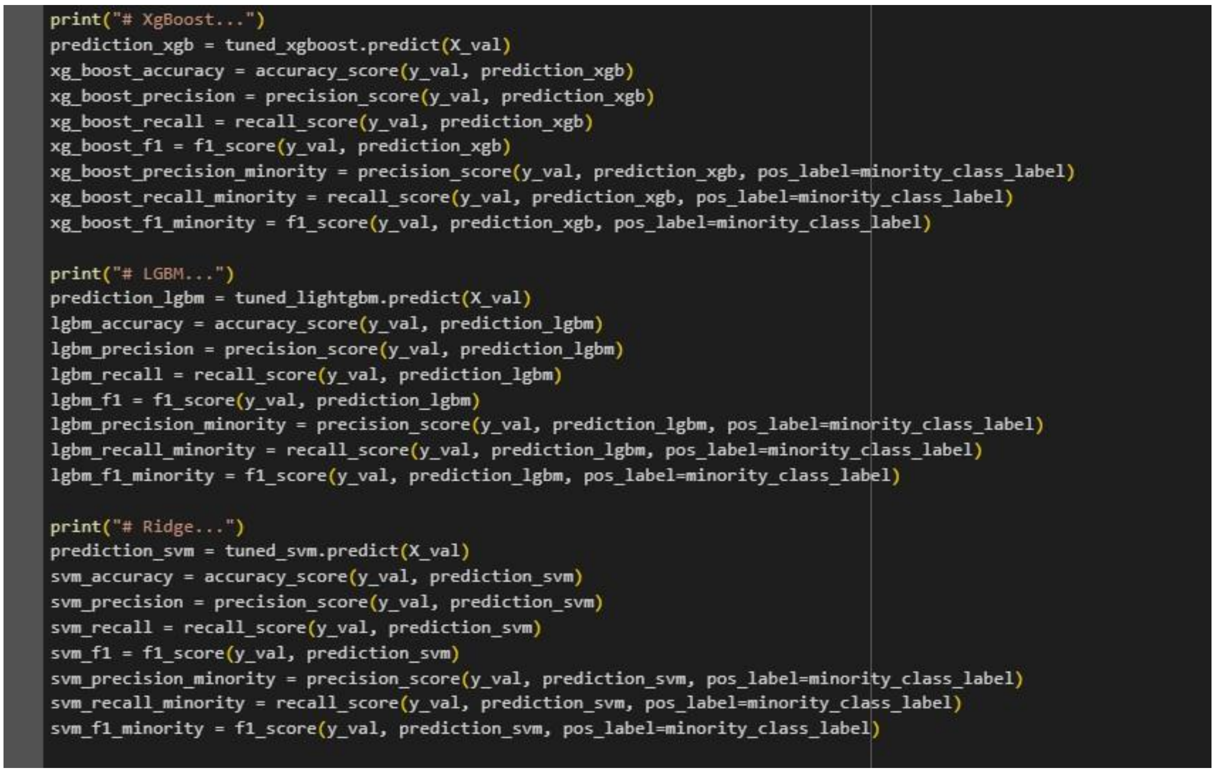
Figure27. This script removes the specified columns ('src', 'dst', 'Protocol') from both the training and validation feature matrices, `X_train` and `X_val`, to prepare the data for modeling.



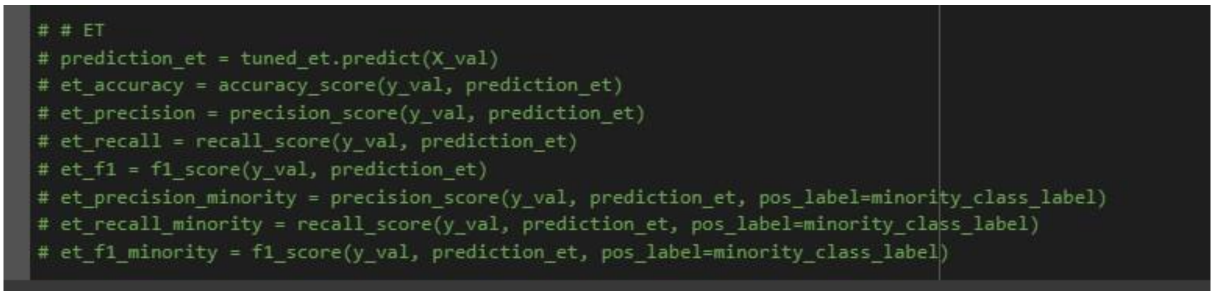
```python
# Define the minority class label (assuming it's class 1, change accordingly if it's different)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier

minority_class_label = 1

mlp_classifier = MLPClassifier(random_state=1, max_iter=300)
decision_tree = DecisionTreeClassifier()


# Decision Tree
tuned_dt.fit(X_train,y_train)

# Neural Networks
mlp_classifier.fit(X_train,y_train)

# Random Forest
tuned_rf.fit(X_train,y_train)

# # logistic Regression
# tuned_lr.fit(X_train,y_train)

# XgBoost
tuned_xgboost.fit(X_train,y_train)

# LGBM
tuned_lightgbm.fit(X_train,y_train)

# Ridge
tuned_svm.fit(X_train,y_train)

# # ET
# tuned_et.fit(X_train,y_train)
```



```python
print("# Decision Tree...")
predictions_dt = tuned_dt.predict(X_val)
decision_tree_accuracy = accuracy_score(y_val, predictions_dt)
decision_tree_precision = precision_score(y_val, predictions_dt)
decision_tree_recall = recall_score(y_val, predictions_dt)
decision_tree_f1 = f1_score(y_val, predictions_dt)
decision_tree_precision_minority = precision_score(y_val, predictions_dt, pos_label=minority_class_label)
decision_tree_recall_minority = recall_score(y_val, predictions_dt, pos_label=minority_class_label)
decision_tree_f1_minority = f1_score(y_val, predictions_dt, pos_label=minority_class_label)

print("# Neural Networks...")
predictions_nn = mlp_classifier.predict(X_val)
mlp_classifier_accuracy = accuracy_score(y_val, predictions_nn)
mlp_classifier_precision = precision_score(y_val, predictions_nn)
mlp_classifier_recall = recall_score(y_val, predictions_nn)
mlp_classifier_f1 = f1_score(y_val, predictions_nn)
mlp_classifier_precision_minority = precision_score(y_val, predictions_nn, pos_label=minority_class_label)
mlp_classifier_recall_minority = recall_score(y_val, predictions_nn, pos_label=minority_class_label)
mlp_classifier_f1_minority = f1_score(y_val, predictions_nn, pos_label=minority_class_label)

print("# Random Forest...")
prediction_rf = tuned_rf.predict(X_val)
random_forest_accuracy = accuracy_score(y_val, prediction_rf)
random_forest_precision = precision_score(y_val, prediction_rf)
random_forest_recall = recall_score(y_val, prediction_rf)
random_forest_f1 = f1_score(y_val, prediction_rf)
random_forest_precision_minority = precision_score(y_val, prediction_rf, pos_label=minority_class_label)
random_forest_recall_minority = recall_score(y_val, prediction_rf, pos_label=minority_class_label)
random_forest_f1_minority = f1_score(y_val, prediction_rf, pos_label=minority_class_label)

# print("# Logistic Regression...")
# prediction_lr = tuned_lr.predict(X_val)
# logistic_regression_accuracy = accuracy_score(y_val, prediction_lr)
# logistic_regression_precision = precision_score(y_val, prediction_lr)
# logistic_regression_recall = recall_score(y_val, prediction_lr)
# logistic_regression_f1 = f1_score(y_val, prediction_lr)
# logistic_regression_precision_minority = precision_score(y_val, prediction_lr, pos_label=minority_class_label)
# logistic_regression_recall_minority = recall_score(y_val, prediction_lr, pos_label=minority_class_label)
# logistic_regression_f1_minority = f1_score(y_val, prediction_lr, pos_label=minority_class_label)
```

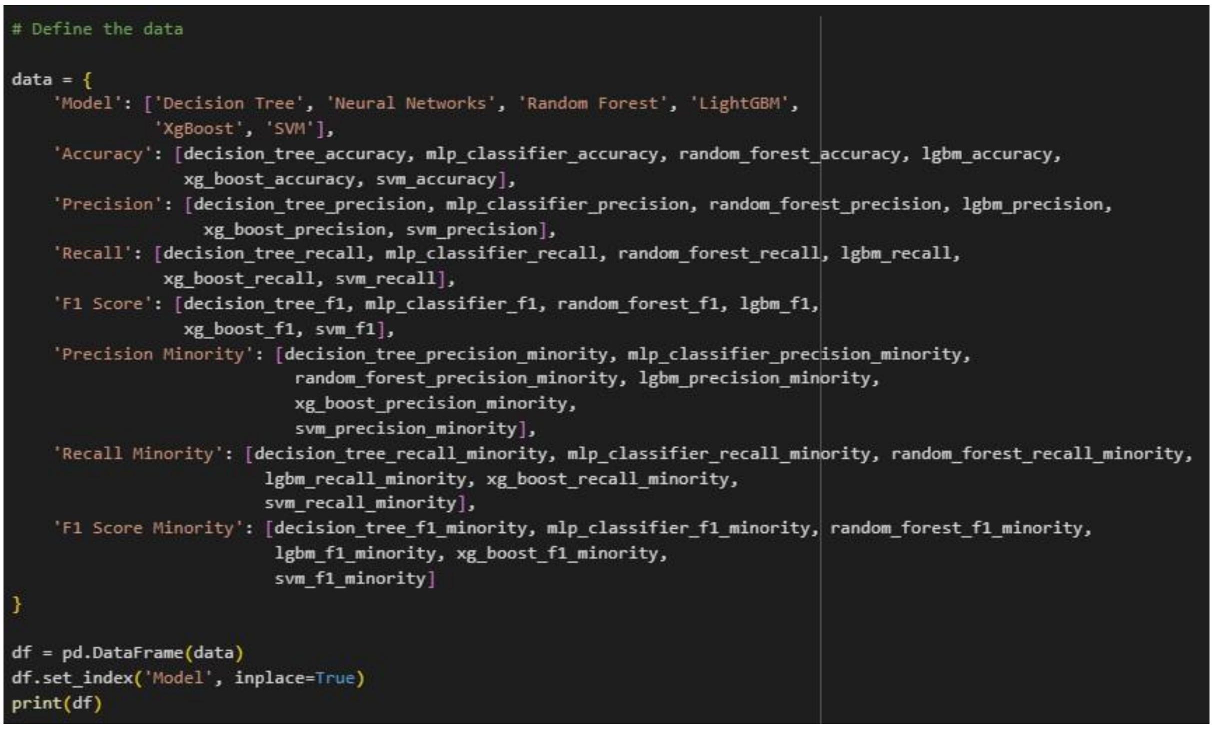


```
print("# XgBoost...")
prediction_xgb = tuned_xgboost.predict(X_val)
xg_boost_accuracy = accuracy_score(y_val, prediction_xgb)
xg_boost_precision = precision_score(y_val, prediction_xgb)
xg_boost_recall = recall_score(y_val, prediction_xgb)
xg_boost_f1 = f1_score(y_val, prediction_xgb)
xg_boost_precision_minority = precision_score(y_val, prediction_xgb, pos_label=minority_class_label)
xg_boost_recall_minority = recall_score(y_val, prediction_xgb, pos_label=minority_class_label)
xg_boost_f1_minority = f1_score(y_val, prediction_xgb, pos_label=minority_class_label)

print("# LGBM...")
prediction_lgbm = tuned_lightgbm.predict(X_val)
lgbm_accuracy = accuracy_score(y_val, prediction_lgbm)
lgbm_precision = precision_score(y_val, prediction_lgbm)
lgbm_recall = recall_score(y_val, prediction_lgbm)
lgbm_f1 = f1_score(y_val, prediction_lgbm)
lgbm_precision_minority = precision_score(y_val, prediction_lgbm, pos_label=minority_class_label)
lgbm_recall_minority = recall_score(y_val, prediction_lgbm, pos_label=minority_class_label)
lgbm_f1_minority = f1_score(y_val, prediction_lgbm, pos_label=minority_class_label)

print("# Ridge...")
prediction_svm = tuned_svm.predict(X_val)
svm_accuracy = accuracy_score(y_val, prediction_svm)
svm_precision = precision_score(y_val, prediction_svm)
svm_recall = recall_score(y_val, prediction_svm)
svm_f1 = f1_score(y_val, prediction_svm)
svm_precision_minority = precision_score(y_val, prediction_svm, pos_label=minority_class_label)
svm_recall_minority = recall_score(y_val, prediction_svm, pos_label=minority_class_label)
svm_f1_minority = f1_score(y_val, prediction_svm, pos_label=minority_class_label)
```



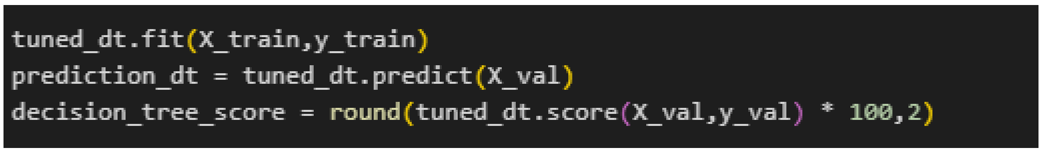
```
# # ET
# prediction_et = tuned_et.predict(X_val)
# et_accuracy = accuracy_score(y_val, prediction_et)
# et_precision = precision_score(y_val, prediction_et)
# et_recall = recall_score(y_val, prediction_et)
# et_f1 = f1_score(y_val, prediction_et)
# et_precision_minority = precision_score(y_val, prediction_et, pos_label=minority_class_label)
# et_recall_minority = recall_score(y_val, prediction_et, pos_label=minority_class_label)
# et_f1_minority = f1_score(y_val, prediction_et, pos_label=minority_class_label)
```

Figure28.This code trains and evaluates various machine learning models (Decision Tree, Neural Networks, Random Forest, XGBoost, LightGBM, and SVM) on the validation set. It calculates and prints performance metrics (accuracy, precision, recall, and F1 score) for each model, including metrics for the minority class. Some models (Logistic Regression and Extra Trees) are prepared but not executed.



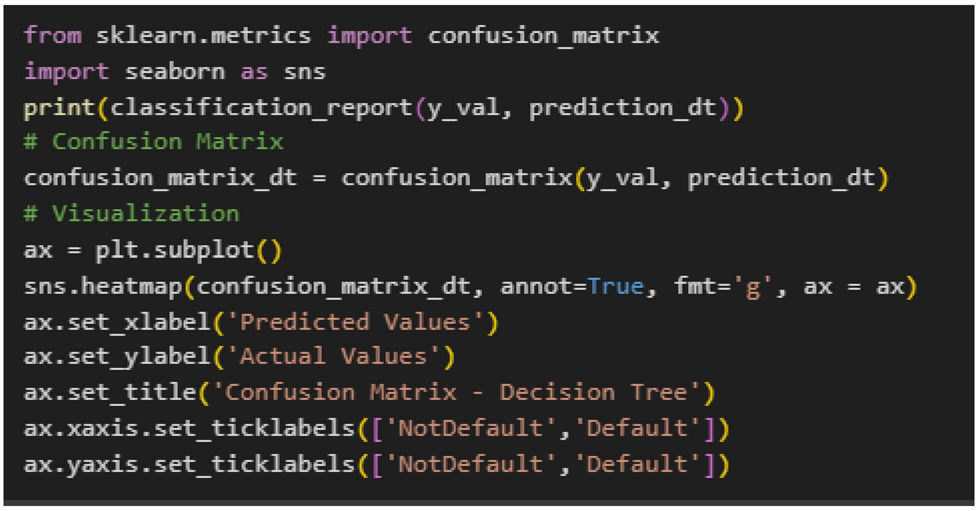
```
# Define the data

data = {
    'Model': ['Decision Tree', 'Neural Networks', 'Random Forest', 'LightGBM',
              'XgBoost', 'SVM'],
    'Accuracy': [decision_tree_accuracy, mlp_classifier_accuracy, random_forest_accuracy, lgbm_accuracy,
                 xg_boost_accuracy, svm_accuracy],
    'Precision': [decision_tree_precision, mlp_classifier_precision, random_forest_precision, lgbm_precision,
                  xg_boost_precision, svm_precision],
    'Recall': [decision_tree_recall, mlp_classifier_recall, random_forest_recall, lgbm_recall,
               xg_boost_recall, svm_recall],
    'F1 Score': [decision_tree_f1, mlp_classifier_f1, random_forest_f1, lgbm_f1,
                 xg_boost_f1, svm_f1],
    'Precision Minority': [decision_tree_precision_minority, mlp_classifier_precision_minority,
                           random_forest_precision_minority, lgbm_precision_minority,
                           xg_boost_precision_minority,
                           svm_precision_minority],
    'Recall Minority': [decision_tree_recall_minority, mlp_classifier_recall_minority, random_forest_recall_minority,
                        lgbm_recall_minority, xg_boost_recall_minority,
                        svm_recall_minority],
    'F1 Score Minority': [decision_tree_f1_minority, mlp_classifier_f1_minority, random_forest_f1_minority,
                          lgbm_f1_minority, xg_boost_f1_minority,
                          svm_f1_minority]
}

df = pd.DataFrame(data)
df.set_index('Model', inplace=True)
print(df)
```

Figure29. This script defines a DataFrame that is used to condense and print performance measurements (Accuracy, Precision, Recall, F1 Score) of the machine learning models. The table provides the metrics of each model, where the model names are used as the index.



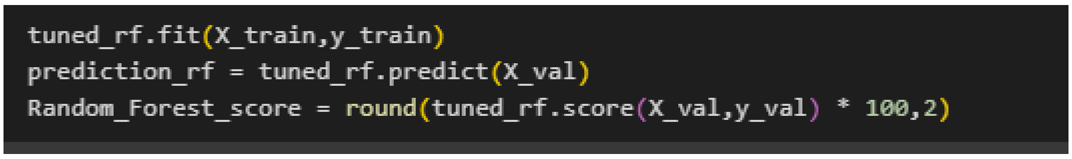
```
tuned_dt.fit(X_train,y_train)
prediction_dt = tuned_dt.predict(X_val)
decision_tree_score = round(tuned_dt.score(X_val,y_val) * 100,2)
```

Figure30. The following script builds and tunes a Decision Tree model, predicts the outcome in the validation set and measures its efficiency in percentage terms.



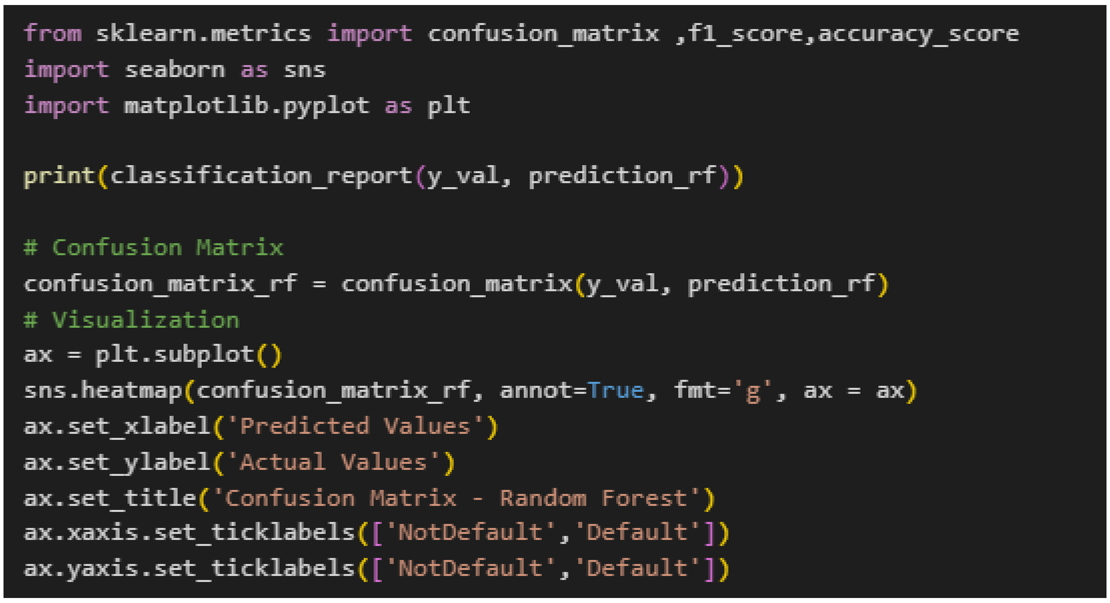
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
print(classification_report(y_val, prediction_dt))
# Confusion Matrix
confusion_matrix_dt = confusion_matrix(y_val, prediction_dt)
# Visualization
ax = plt.subplot()
sns.heatmap(confusion_matrix_dt, annot=True, fmt='g', ax = ax)
ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values')
ax.set_title('Confusion Matrix - Decision Tree')
ax.xaxis.set_ticklabels(['NotDefault','Default'])
ax.yaxis.set_ticklabels(['NotDefault','Default'])
```

Figure31. The following script displays a classification report on the Decision Tree model and generates the confusion matrix of this model. Confusion matrix is plotted as the heatmap with the annotation of the classes for the 'NotDefault' and 'Default' classes – this shows the Predicted value against the actual value.



```
tuned_rf.fit(X_train,y_train)
prediction_rf = tuned_rf.predict(X_val)
Random_Forest_score = round(tuned_rf.score(X_val,y_val) * 100,2)
```

Figure32. This script trains a tuned Random Forest model, makes predictions on the validation set, and calculates its accuracy as a percentage.



```
from sklearn.metrics import confusion_matrix ,f1_score,accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

print(classification_report(y_val, prediction_rf))

# Confusion Matrix
confusion_matrix_rf = confusion_matrix(y_val, prediction_rf)
# Visualization
ax = plt.subplot()
sns.heatmap(confusion_matrix_rf, annot=True, fmt='g', ax = ax)
ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values')
ax.set_title('Confusion Matrix - Random Forest')
ax.xaxis.set_ticklabels(['NotDefault','Default'])
ax.yaxis.set_ticklabels(['NotDefault','Default'])
```

Figure33. This script evaluates and visualizes the performance of the tuned Random Forest model:This script evaluates and visualizes the performance of the tuned Random Forest model:

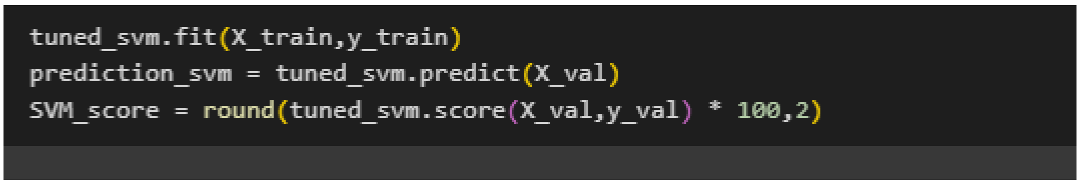
1. Print Classification Report: Includes precision, recall, and F1 score of Random Forest model set as the number of relevant documents.

2. Compute Confusion Matrix: Estimates the confusion matrix which indicates true positive, true negative, false positive, and false negative.

3. Visualize Confusion Matrix:

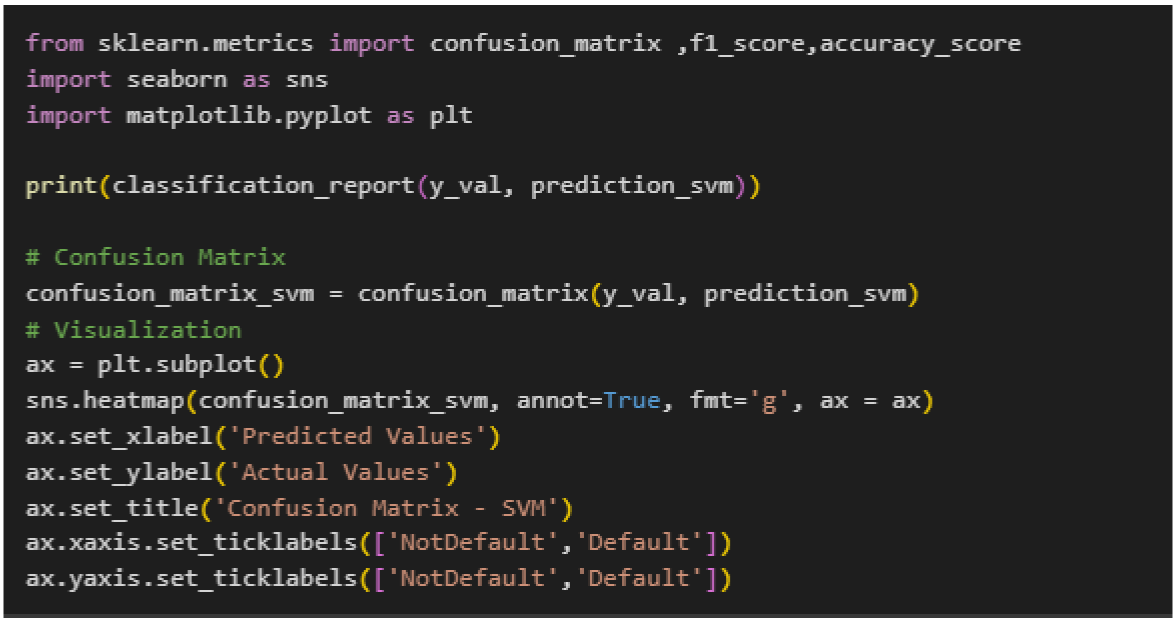
- `sns. heatmap()`: Plots the confusion matrix as a heatmap where you also see the numerical values.

- Labels and Title: Also, axis labels and title of the heatmap are added and tick labels are set as ‘NotDefault’ and ‘Default’ for class labels.



```
tuned_svm.fit(X_train,y_train)
prediction_svm = tuned_svm.predict(X_val)
SVM_score = round(tuned_svm.score(X_val,y_val) * 100,2)
```

Figure34. This script trains the tuned SVM model, makes predictions on the validation set, and calculates its accuracy as a percentage..



```python
from sklearn.metrics import confusion_matrix ,f1_score,accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

print(classification_report(y_val, prediction_svm))

# Confusion Matrix
confusion_matrix_svm = confusion_matrix(y_val, prediction_svm)
# Visualization
ax = plt.subplot()
sns.heatmap(confusion_matrix_svm, annot=True, fmt='g', ax = ax)
ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values')
ax.set_title('Confusion Matrix - SVM')
ax.xaxis.set_ticklabels(['NotDefault','Default'])
ax.yaxis.set_ticklabels(['NotDefault','Default'])
```

Figure35. This script evaluates and visualizes the performance of the tuned SVM model:

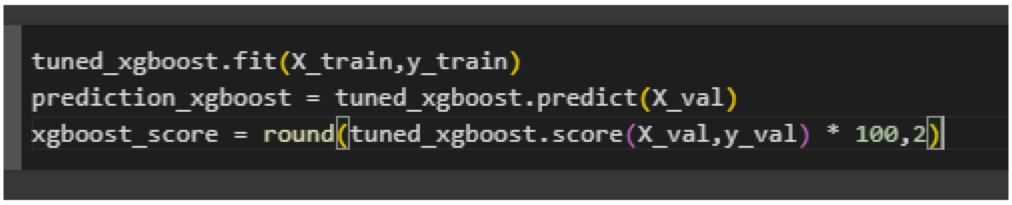
1. Print Classification Report: Outputs precision, recall, F1 score, and other metrics for the SVM model.

2. Compute Confusion Matrix: Calculates the confusion matrix, showing counts of true and false predictions.

3. Visualize Confusion Matrix:

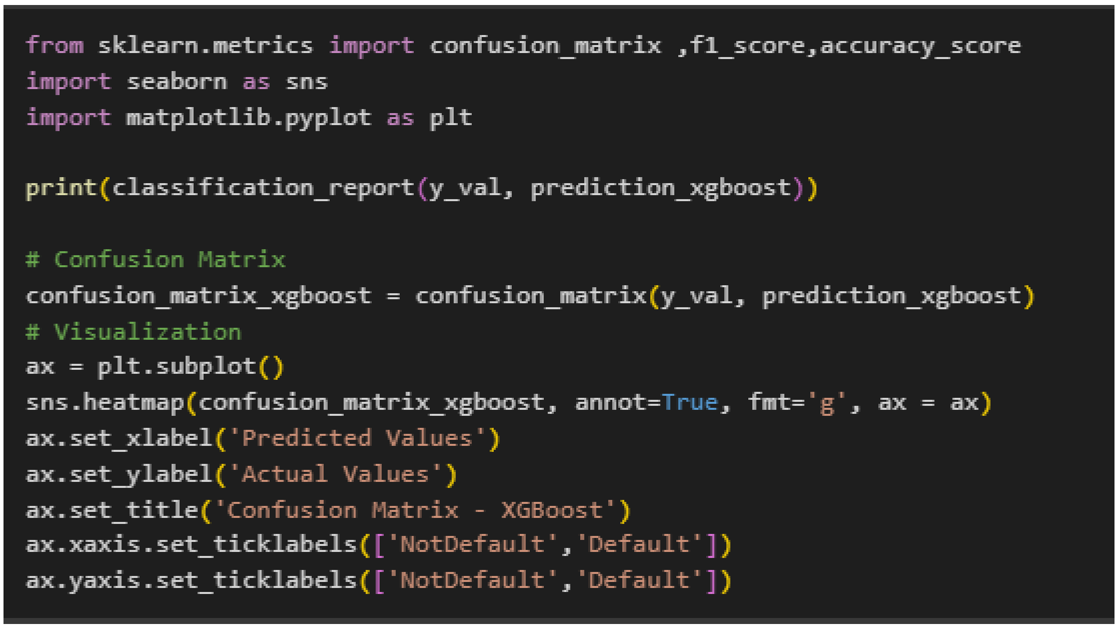
   - `sns.heatmap()`: Creates a heatmap of the confusion matrix with annotations.

   - Labels and Title: Adds axis labels and a title to the heatmap, with tick labels set to 'NotDefault' and 'Default' to indicate the classes.



```python
tuned_xgboost.fit(X_train,y_train)
prediction_xgboost = tuned_xgboost.predict(X_val)
xgboost_score = round(tuned_xgboost.score(X_val,y_val) * 100,2)
```

Figure36. This script trains the tuned XGBoost model, makes predictions on the validation set, and calculates its accuracy as a percentage.



```python
from sklearn.metrics import confusion_matrix ,f1_score,accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

print(classification_report(y_val, prediction_xgboost))

# Confusion Matrix
confusion_matrix_xgboost = confusion_matrix(y_val, prediction_xgboost)
# Visualization
ax = plt.subplot()
sns.heatmap(confusion_matrix_xgboost, annot=True, fmt='g', ax = ax)
ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values')
ax.set_title('Confusion Matrix - XGBoost')
ax.xaxis.set_ticklabels(['NotDefault','Default'])
ax.yaxis.set_ticklabels(['NotDefault','Default'])
```

Figure37. This script evaluates and visualizes the performance of the tuned XGBoost model:This script evaluates and visualizes the performance of the tuned XGBoost model:

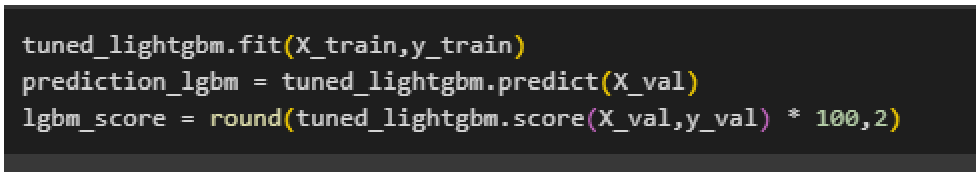
1. Print Classification Report: Accuracy, Recall, F-Value for the XGBoost model and other metrics.

2. Compute Confusion Matrix: Computes the confusion matrix that represents the absolute number of true and false on/off predictions.

3. Visualize Confusion Matrix:

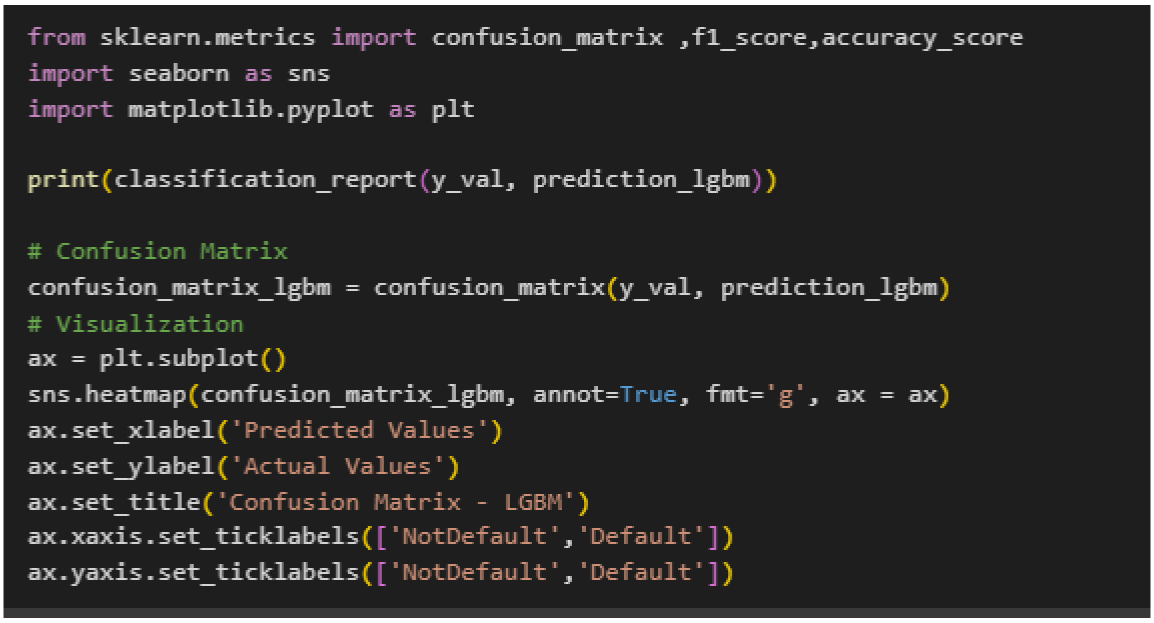
- `sns. heatmap()`: Builds a heatmap of the confusion matrix with the count included in the annotation.

- Labels and Title: Provides axis annotation and a caption to the created heatmap; makes the tick labels equal to 'NotDefault' and 'Default'.



```python
tuned_lightgbm.fit(X_train,y_train)
prediction_lgbm = tuned_lightgbm.predict(X_val)
lgbm_score = round(tuned_lightgbm.score(X_val,y_val) * 100,2)
```

Figure38. This script uses the tuned LightGBM model to train on the whole data, to make prediction for the unseen validation dataset and calculate its accuracy in percentage.



```python
from sklearn.metrics import confusion_matrix ,f1_score,accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

print(classification_report(y_val, prediction_lgbm))

# Confusion Matrix
confusion_matrix_lgbm = confusion_matrix(y_val, prediction_lgbm)
# Visualization
ax = plt.subplot()
sns.heatmap(confusion_matrix_lgbm, annot=True, fmt='g', ax = ax)
ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values')
ax.set_title('Confusion Matrix - LGBM')
ax.xaxis.set_ticklabels(['NotDefault','Default'])
ax.yaxis.set_ticklabels(['NotDefault','Default'])
```

Figure39. This script evaluates and visualizes the performance of the tuned LightGBM model:This script evaluates and visualizes the performance of the tuned LightGBM model:

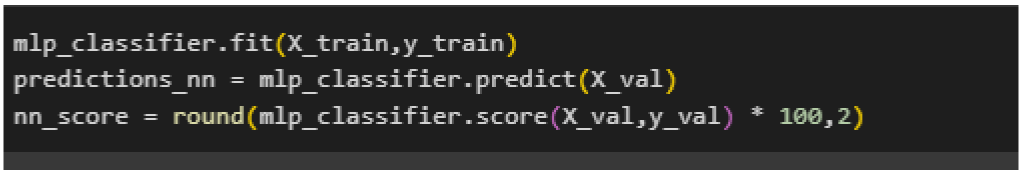
1. Print Classification Report: I/Os metrics such as precision, recall, the F1 score and other characteristics of the model LightGBM.

2. Compute Confusion Matrix: It computes the confusion matrix needed to illustrate the true and false results.

3. Visualize Confusion Matrix:

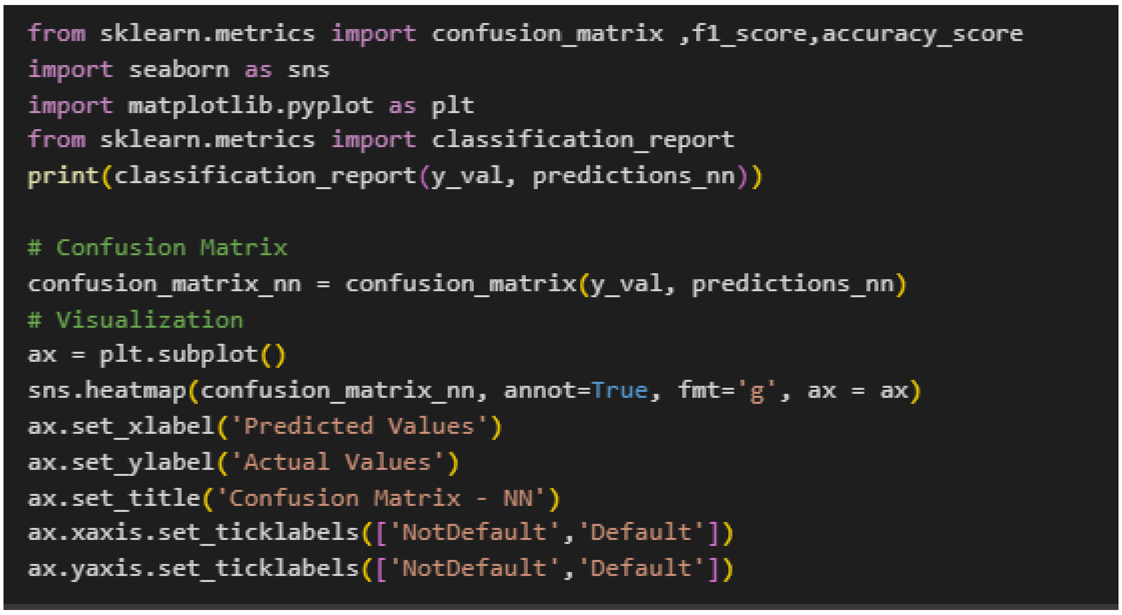
- `sns. heatmap()`: Creates the heat map of the confusion matrix and labels it.

- Labels and Title: Plots axis labels and title to the heatmap, tick labels to be used are ‘NotDefault’ and ‘Default’ for the classes.



```python
mlp_classifier.fit(X_train,y_train)
predictions_nn = mlp_classifier.predict(X_val)
nn_score = round(mlp_classifier.score(X_val,y_val) * 100,2)
```

Figure40.This script fine-tunes the neural network model of MLPClassifier, evaluates them on the validation set and prints out the accuracy in percentage.



```
from sklearn.metrics import confusion_matrix ,f1_score,accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
print(classification_report(y_val, predictions_nn))

# Confusion Matrix
confusion_matrix_nn = confusion_matrix(y_val, predictions_nn)
# Visualization
ax = plt.subplot()
sns.heatmap(confusion_matrix_nn, annot=True, fmt='g', ax = ax)
ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values')
ax.set_title('Confusion Matrix - NN')
ax.xaxis.set_ticklabels(['NotDefault','Default'])
ax.yaxis.set_ticklabels(['NotDefault','Default'])
```

Figure41. This script evaluates and visualizes the performance of the neural network model (`mlp_classifier`):This script evaluates and visualizes the performance of the neural network model (`mlp_classifier`):

1. Print Classification Report: accuracy, precision, recall, f1-score and other details of the particular neural network model used in the solution.

2. Compute Confusion Matrix: Returns the confusion matrix with the number of correct and incorrect predictions.

3. Visualize Confusion Matrix:

- `sns. heatmap()`: Creates figures and subplots of the confusion matrix and borders them with the heatmap and annotations.

- Labels and Title: Appends axis labels and a title to the heatmap to include the tick labels as 'NotDefault' and 'Default' which are the classes.

3 Steps to Run and execute the codes

Step 1: Initially the user has to go to the google drive which contain the codes and data.

Step 2: Run the colab file.

Step 3: Access authentication for the drive must be made.

Step 4: Now record the application by start running it in the Anaconda Prompt.

Step 5: Run the code python -m streamlit run app. py