Configuration Manual

MSc Research Project
Programme Name

Akanksha Shephali
Student ID: X23137720

School of Computing
National College of Ireland

Supervisor: Furqan Rustam

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	AKANKSHA SHEPHALI		
Student ID:	X23137720		
Programme:	MSc in Data Analytics	**Year:**	2024-2025
Module:	MSc Research Project		
Lecturer:	Furqan Rustam		
Submission Due Date:	12/12/2024		
Project Title:	Traffic Violation Arrests using Machine learning models		
Word Count:	**1000**	**Page Count: 11**	

Signature:	AKANKSHA SHEPHALI
Date:	29/01/2025

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

Akanksha Shephali
Student ID: x23137720

1 Introduction

This configuration manual contains a complete guide to setting up, installing, and deploying predicting arrests due to traffic violation projects. It outlines the system specifications, the key steps in the installation process, various data pre-processing techniques, issues on model design, training and testing methods, and framework for various deployment contexts. By reading this manual and following the directions herein, you will also gain an understanding of the most basic concepts of developing, training, and evaluating machine learning and deep learning models for arrest prediction due to traffic violations.

2 System Configuration

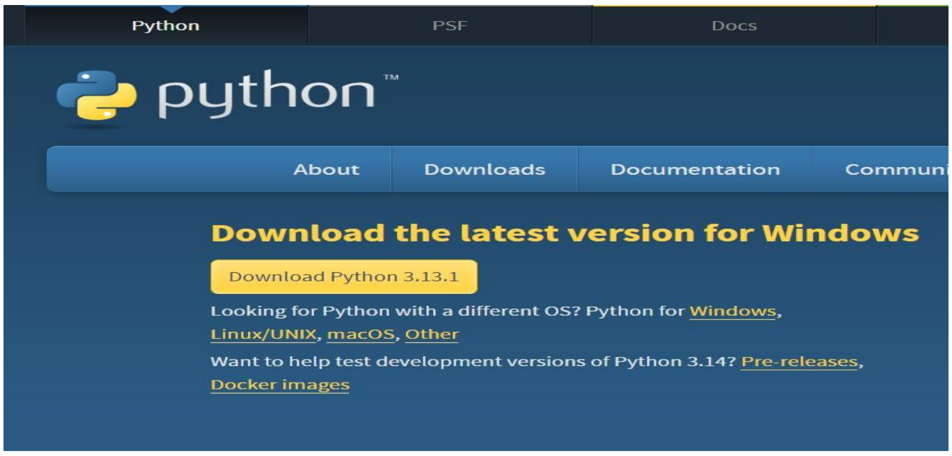
2.1 Hardware Configuration

- OS Used: Windows 11, version 24H2
- Processor Used: 12th GEN INTEL CORE i5
- Hard drive Used: SSD(460GB)
- RAM: 8GB

2.2 Software Configuration

- Python
- Jupiter Notebook

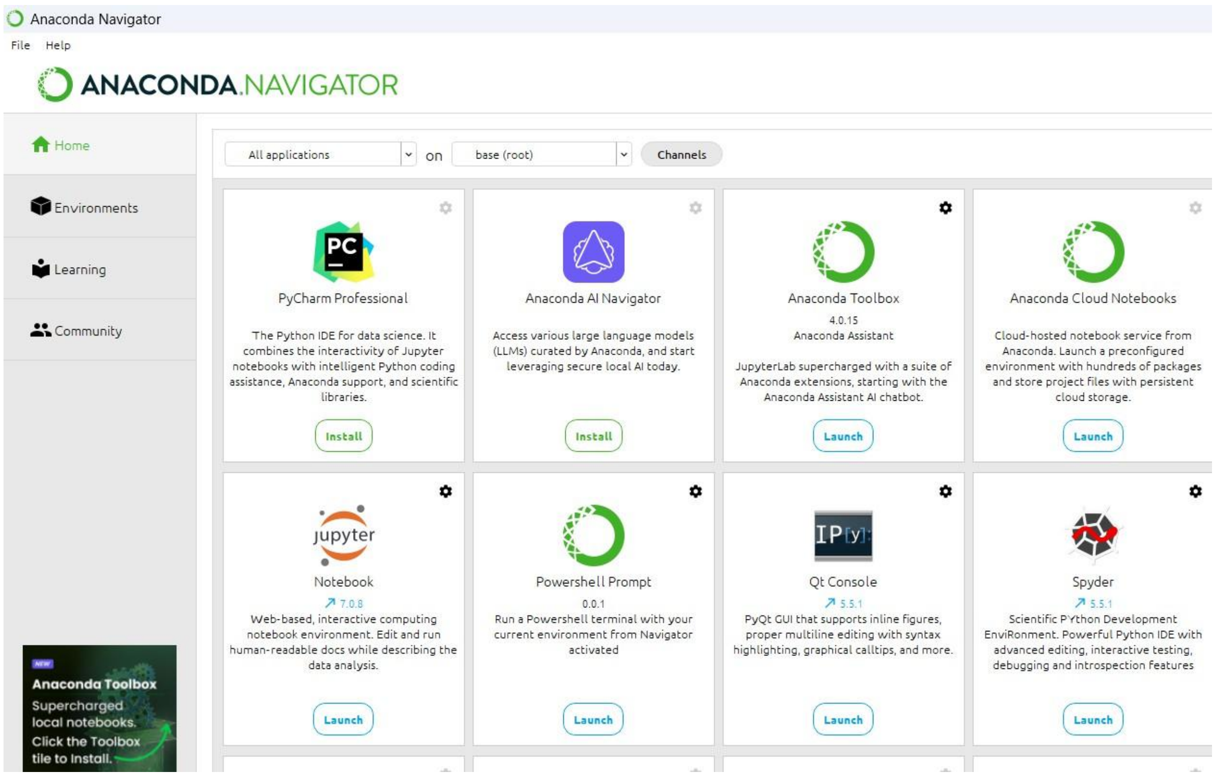
3 Installation and Setting up Environment



Download the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python 3.14? Pre-releases, Docker images



PyCharm Professional

The Python IDE for data science. It combines the interactivity of Jupyter notebooks with intelligent Python coding assistance, Anaconda support, and scientific libraries.

Install

Anaconda AI Navigator

Access various large language models (LLMs) curated by Anaconda, and start leveraging secure local AI today.

Install

Anaconda Toolbox

4.0.15

Anaconda Assistant

JupyterLab supercharged with a suite of Anaconda extensions, starting with the Anaconda Assistant AI chatbot.

Launch

Anaconda Cloud Notebooks

Cloud-hosted notebook service from Anaconda. Launch a preconfigured environment with hundreds of packages and store project files with persistent cloud storage.

Launch

Notebook

7.0.8

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Launch

Powershell Prompt

0.0.1

Run a Powershell terminal with your current environment from Navigator activated

Launch

Qt Console

5.5.1

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Launch

Spyder

5.5.1

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Launch

NEW

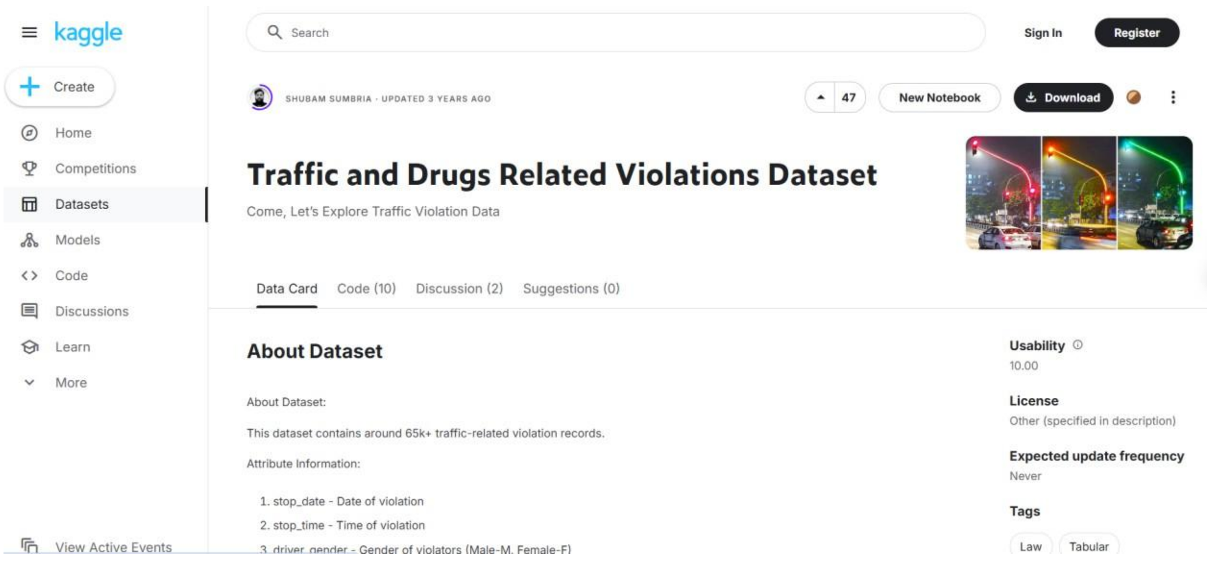
Anaconda Toolbox

Supercharged local notebooks. Click the Toolbox tile to install.

4. Dataset

4.1 Dataset Collection

In order to import the dataset into Jupiter Notebook which is a Python environment, we need to get the dataset ”Traffic and Drugs Related Violations Dataset” (Shubham, 2021) from the website “https://www.kaggle.com/datasets/shubamsumbria/traffic-violations-dataset/data”.



SHUBAM SUMBRIA · UPDATED 3 YEARS AGO

Traffic and Drugs Related Violations Dataset

Come, Let's Explore Traffic Violation Data

About Dataset

About Dataset:

This dataset contains around 65k+ traffic-related violation records.

Attribute Information:

1. stop_date - Date of violation
2. stop_time - Time of violation
3. driver_gender - Gender of violators (Male-M, Female-F)

Usability
10.00

License
Other (specified in description)

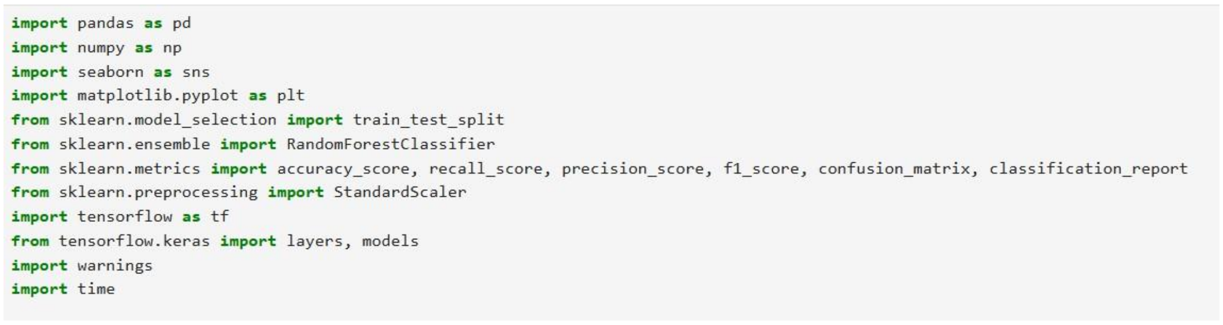
Expected update frequency
Never

Tags
Law Tabular

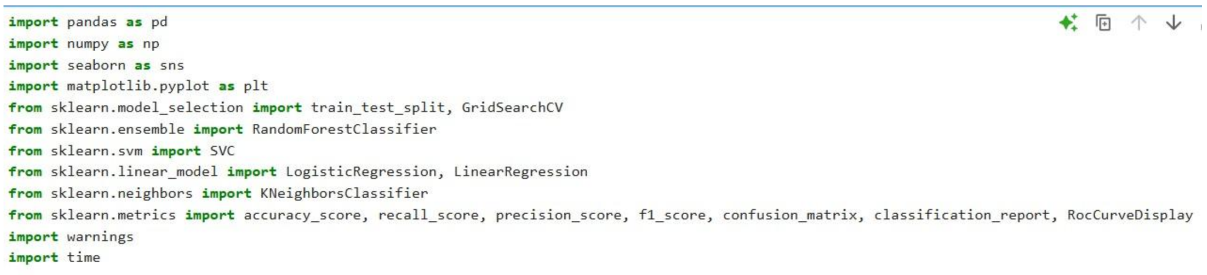
4.2 Importing Python Libraries

- Python is a high-level programming language that contains several libraries, which we need to include while programming as shown in the figure below.

- Thereafter, when we require such libraries based on some requirements that we have then we can go ahead and import them into use in our research.



```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras import layers, models
import warnings
import time
```

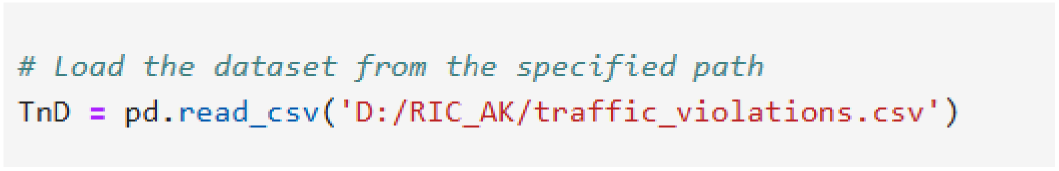


```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, confusion_matrix, classification_report, RocCurveDisplay
import warnings
import time
```

4.3 Data Preprocessing and Analyzing

4.3.1 Data Import

Dataset Path: This dataset is taken from the source D:/RIC_AK/traffic_violations.csv.



```python
# Load the dataset from the specified path
TnD = pd.read_csv('D:/RIC_AK/traffic_violations.csv')
```

4.3.2 Data Cleaning

Several data cleaning steps are applied to ensure the data is ready for modeling:

Step 1: Dropping Irrelevant Columns: Any of the columns that are irrelevant to the analysis are omitted.

Step 2: Handling Missing Data: The na or empty rows are removed to avoid the presence of missing values at the time of model formation.

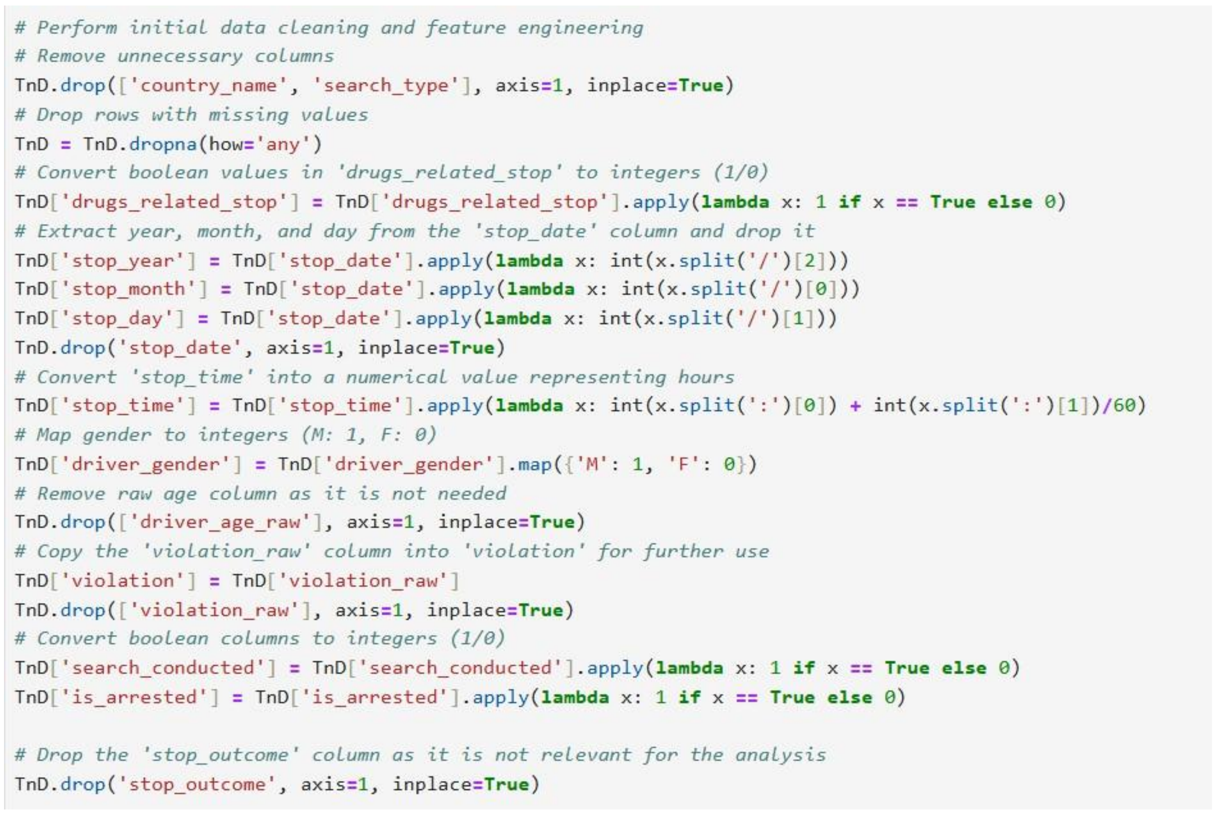
Step 3: Binary Conversion for drugs_related_stop: First the drugs_related_stop column is converted to binary values in which True is 1 and False is 0.

Step 4: Extracting Date Features: Features are extracted from stop_date by making year, month, and day different attributes.

Step 5: Handling stop_time: The stop_time column is then transformed into a numerical form where the value of each stop time is in terms of an hour and a fraction of an hour.

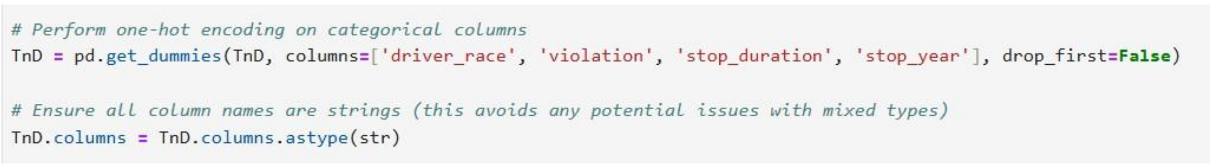
Step 6: Categorical Data Transformation: Driver gender is transformed into binary variables, where M is coded as 1, and, F is coded as 0.

Step 7: Dropping Columns: The driver_age_raw and stop_outcome features are omitted from the analysis as they can be considered irrelevant to the job.



```python
# Perform initial data cleaning and feature engineering
# Remove unnecessary columns
TnD.drop(['country_name', 'search_type'], axis=1, inplace=True)
# Drop rows with missing values
TnD = TnD.dropna(how='any')
# Convert boolean values in 'drugs_related_stop' to integers (1/0)
TnD['drugs_related_stop'] = TnD['drugs_related_stop'].apply(lambda x: 1 if x == True else 0)
# Extract year, month, and day from the 'stop_date' column and drop it
TnD['stop_year'] = TnD['stop_date'].apply(lambda x: int(x.split('/')[2]))
TnD['stop_month'] = TnD['stop_date'].apply(lambda x: int(x.split('/')[0]))
TnD['stop_day'] = TnD['stop_date'].apply(lambda x: int(x.split('/')[1]))
TnD.drop('stop_date', axis=1, inplace=True)
# Convert 'stop_time' into a numerical value representing hours
TnD['stop_time'] = TnD['stop_time'].apply(lambda x: int(x.split(':')[0]) + int(x.split(':')[1])/60)
# Map gender to integers (M: 1, F: 0)
TnD['driver_gender'] = TnD['driver_gender'].map({'M': 1, 'F': 0})
# Remove raw age column as it is not needed
TnD.drop(['driver_age_raw'], axis=1, inplace=True)
# Copy the 'violation_raw' column into 'violation' for further use
TnD['violation'] = TnD['violation_raw']
TnD.drop(['violation_raw'], axis=1, inplace=True)
# Convert boolean columns to integers (1/0)
TnD['search_conducted'] = TnD['search_conducted'].apply(lambda x: 1 if x == True else 0)
TnD['is_arrested'] = TnD['is_arrested'].apply(lambda x: 1 if x == True else 0)

# Drop the 'stop_outcome' column as it is not relevant for the analysis
TnD.drop('stop_outcome', axis=1, inplace=True)
```

Step 8: One-Hot Encoding for Categorical Variables: Categorical variables in driver_race, violation, stop_duration, and stop_year are encoded using the one-hot encoding algorithm.



```python
# Perform one-hot encoding on categorical columns
TnD = pd.get_dummies(TnD, columns=['driver_race', 'violation', 'stop_duration', 'stop_year'], drop_first=False)

# Ensure all column names are strings (this avoids any potential issues with mixed types)
TnD.columns = TnD.columns.astype(str)
```

4.3.3 Split Data

The data is then divided into labels features(X) and label (y) followed by the division into training and testing data (80% training and 20% testing).

5. Modelling

In this section, we use several classification techniques to perform the classification of the target variable is_arrested.

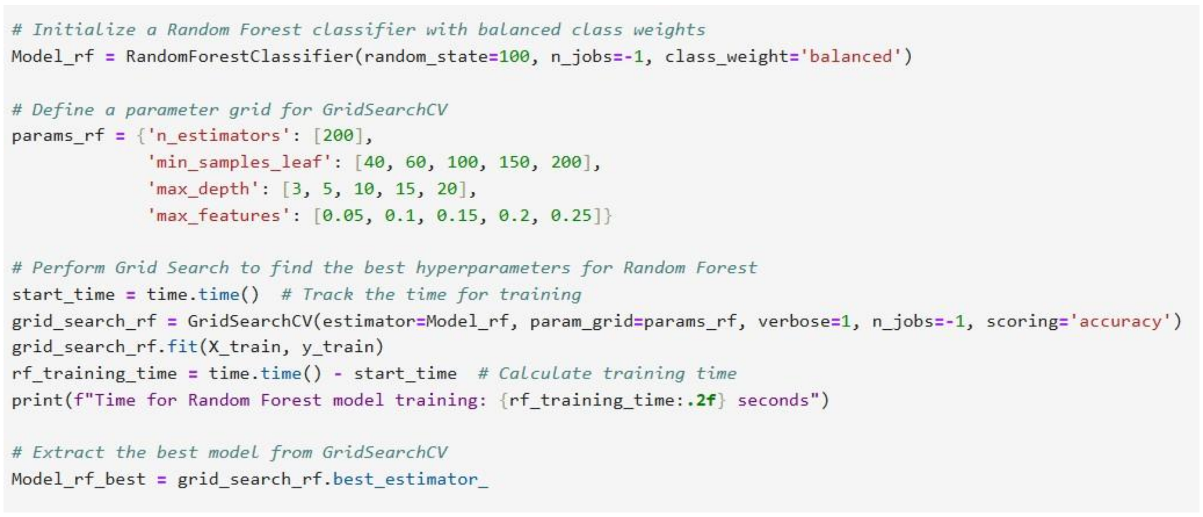
5.1 Random Forest:

The Random Forest model is tuned using Grid Search with the below hyperparameters:

n_estimators: Density of trees in the forest (set to 200 trees).
min_samples_leaf: Minimum samples needed at the terminal nodes are set in (40, 60, 100, 150, 200).
max_depth: Depth of the trees at which they are optimized (values: 3, 5, 10, 15, 20).
max_features: Number of features to consider for splitting: 0.05, 0.1, 0.15, 0.2, 0.25.

To decide the best model, the result of the grid search is used.

By applying the trained model to the testing data as well as the training data possible results are obtained.



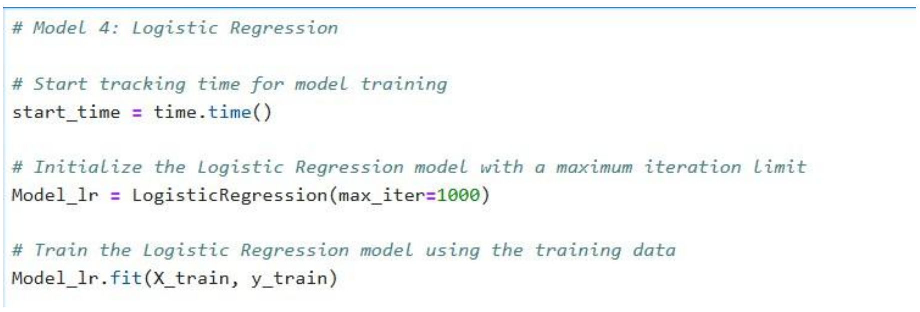
```
# Initialize a Random Forest classifier with balanced class weights
Model_rf = RandomForestClassifier(random_state=100, n_jobs=-1, class_weight='balanced')

# Define a parameter grid for GridSearchCV
params_rf = {'n_estimators': [200],
             'min_samples_leaf': [40, 60, 100, 150, 200],
             'max_depth': [3, 5, 10, 15, 20],
             'max_features': [0.05, 0.1, 0.15, 0.2, 0.25]}

# Perform Grid Search to find the best hyperparameters for Random Forest
start_time = time.time()  # Track the time for training
grid_search_rf = GridSearchCV(estimator=Model_rf, param_grid=params_rf, verbose=1, n_jobs=-1, scoring='accuracy')
grid_search_rf.fit(X_train, y_train)
rf_training_time = time.time() - start_time  # Calculate training time
print(f"Time for Random Forest model training: {rf_training_time:.2f} seconds")

# Extract the best model from GridSearchCV
Model_rf_best = grid_search_rf.best_estimator_
```

5.2 Logistic Regression

The Logistic Regression is another linear model used to categorize the results into two categories. It predicts the likelihood of a categorical dependent variable given one or a set of independent variables.

Hyperparameters used for this model are as below:

max_iter: The maximum number of times the solver is allowed to run in order to look for the best solution.

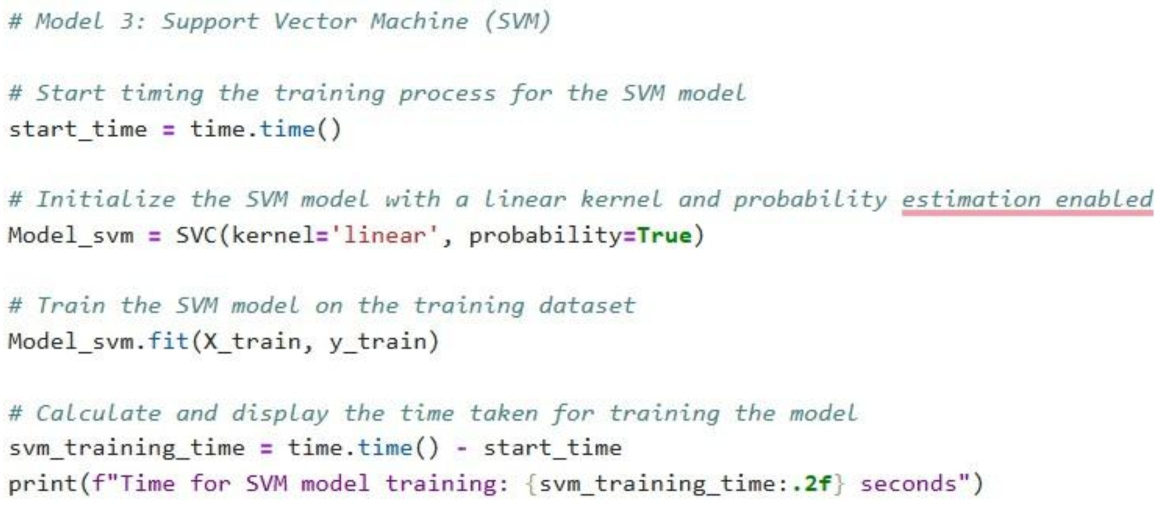


```python
# Model 4: Logistic Regression

# Start tracking time for model training
start_time = time.time()

# Initialize the Logistic Regression model with a maximum iteration limit
Model_lr = LogisticRegression(max_iter=1000)

# Train the Logistic Regression model using the training data
Model_lr.fit(X_train, y_train)
```

5.3 Support Vector Machine (SVM)

Support Vector Machine (SVM) is an elegant classifier that aims at identifying a hyperplane that is closest to the sets of data points belonging to different classes. This is perfect for data whose support extends to high-dimensional spaces.

The hyperparameters used are as follows:
C: Regularization parameter.
kernel: Kernel function type (Linear or Radial etc.,).

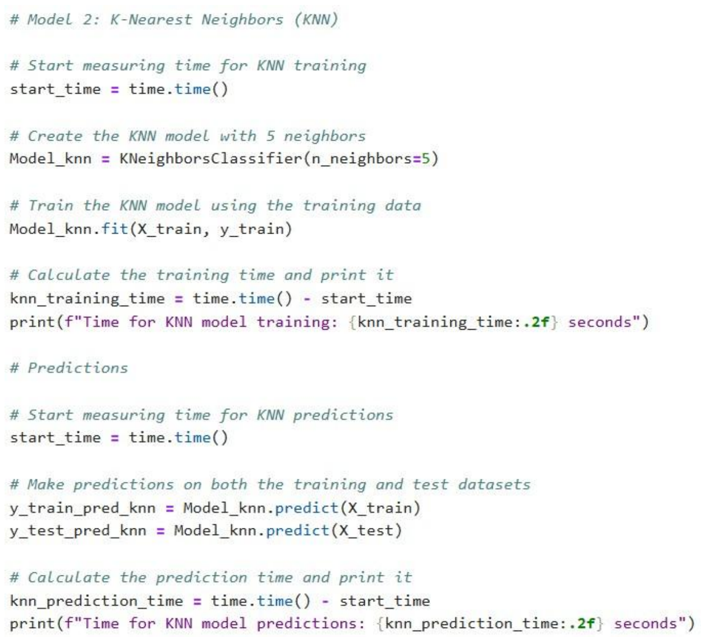


```python
# Model 3: Support Vector Machine (SVM)

# Start timing the training process for the SVM model
start_time = time.time()

# Initialize the SVM model with a linear kernel and probability estimation enabled
Model_svm = SVC(kernel='linear', probability=True)

# Train the SVM model on the training dataset
Model_svm.fit(X_train, y_train)

# Calculate and display the time taken for training the model
svm_training_time = time.time() - start_time
print(f"Time for SVM model training: {svm_training_time:.2f} seconds")
```

5.4 K- Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is another machine learning classification type, which is non-parametric in nature. The method of classification is done with the help of the nearest-neighbor rule which categorizes new data points according to the majority class of the nearest neighbors. Hyperparameter used are as follows:

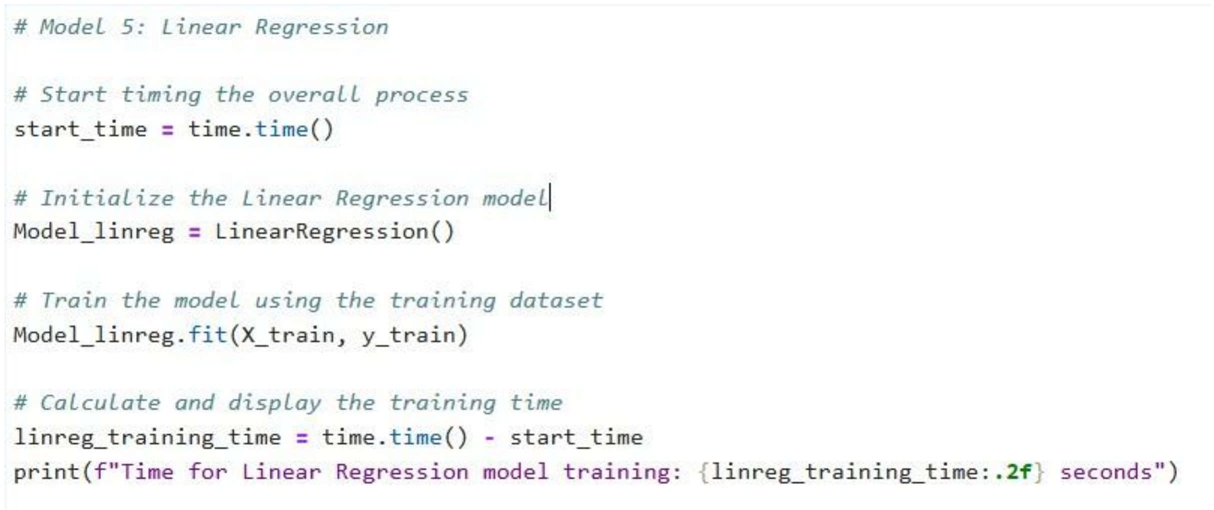
n_neighbors: The number of neighbors for classification or decision making.



```
# Model 2: K-Nearest Neighbors (KNN)

# Start measuring time for KNN training
start_time = time.time()

# Create the KNN model with 5 neighbors
Model_knn = KNeighborsClassifier(n_neighbors=5)

# Train the KNN model using the training data
Model_knn.fit(X_train, y_train)

# Calculate the training time and print it
knn_training_time = time.time() - start_time
print(f"Time for KNN model training: {knn_training_time:.2f} seconds")

# Predictions

# Start measuring time for KNN predictions
start_time = time.time()

# Make predictions on both the training and test datasets
y_train_pred_knn = Model_knn.predict(X_train)
y_test_pred_knn = Model_knn.predict(X_test)

# Calculate the prediction time and print it
knn_prediction_time = time.time() - start_time
print(f"Time for KNN model predictions: {knn_prediction_time:.2f} seconds")
```

5.5 Linear Regression

Linear Regression is used to predict continuous variables although it can be tweaked for binary classification by setting the probability values.

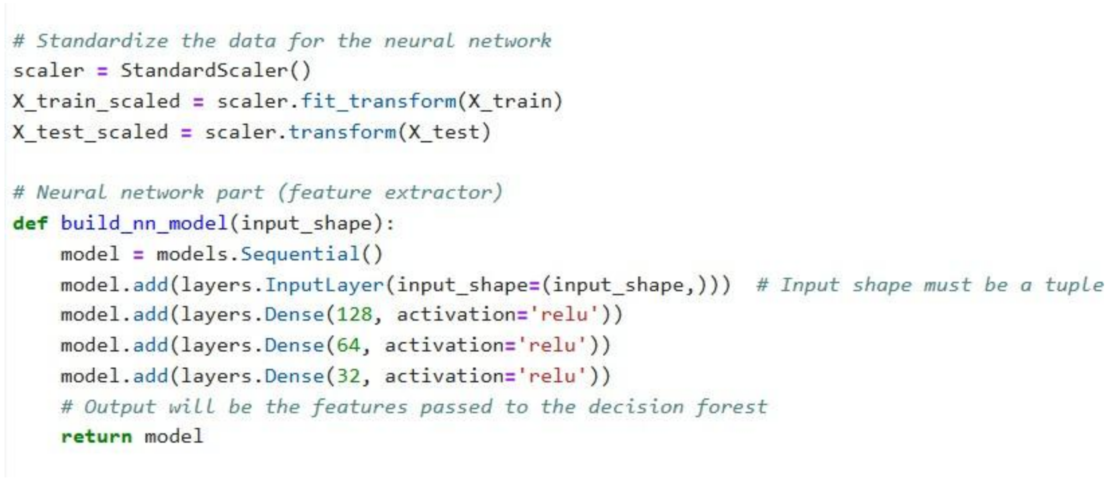


```
# Model 5: Linear Regression

# Start timing the overall process
start_time = time.time()

# Initialize the Linear Regression model
Model_linreg = LinearRegression()

# Train the model using the training dataset
Model_linreg.fit(X_train, y_train)

# Calculate and display the training time
linreg_training_time = time.time() - start_time
print(f"Time for Linear Regression model training: {linreg_training_time:.2f} seconds")
```

5.6 Deep Neural Decision Forest(DNDF)

Deep Neural Decision Forest fuses decision forests and deep learning to obtain the best result. It is sophisticated from the perspective of ensemble learning and includes the application of neural networks as well as decision trees.

Libraries: The implementation of this can be done using TensorFlow or Keras. This, however, is a more advanced technique and needs to be implemented from scratch and not implemented in scikit-learn.

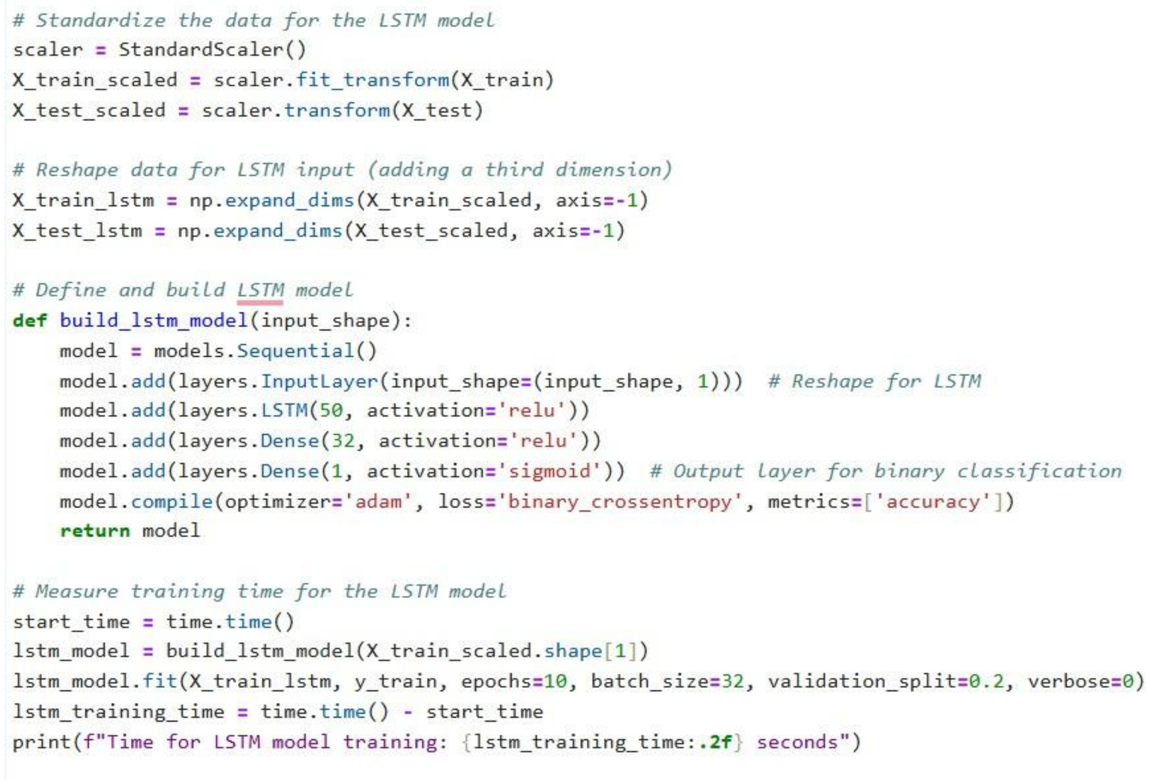


```python
# Standardize the data for the neural network
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Neural network part (feature extractor)
def build_nn_model(input_shape):
    model = models.Sequential()
    model.add(layers.InputLayer(input_shape=(input_shape,)))  # Input shape must be a tuple
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(32, activation='relu'))
    # Output will be the features passed to the decision forest
    return model
```

5.7 LSTM

LSTM is a specific kind of RNN used in input data with the time factor or sequence data. This comes in handy in capturing long term dependencies in the data.

Hyperparameters:
units: The number of neurons in the LSTM layer fixed it in order to make proper dimension of output.



```python
# Standardize the data for the LSTM model
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape data for LSTM input (adding a third dimension)
X_train_lstm = np.expand_dims(X_train_scaled, axis=-1)
X_test_lstm = np.expand_dims(X_test_scaled, axis=-1)

# Define and build LSTM model
def build_lstm_model(input_shape):
    model = models.Sequential()
    model.add(layers.InputLayer(input_shape=(input_shape, 1)))  # Reshape for LSTM
    model.add(layers.LSTM(50, activation='relu'))
    model.add(layers.Dense(32, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))  # Output layer for binary classification
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Measure training time for the LSTM model
start_time = time.time()
lstm_model = build_lstm_model(X_train_scaled.shape[1])
lstm_model.fit(X_train_lstm, y_train, epochs=10, batch_size=32, validation_split=0.2, verbose=0)
lstm_training_time = time.time() - start_time
print(f"Time for LSTM model training: {lstm_training_time:.2f} seconds")
```

5.8 MLP

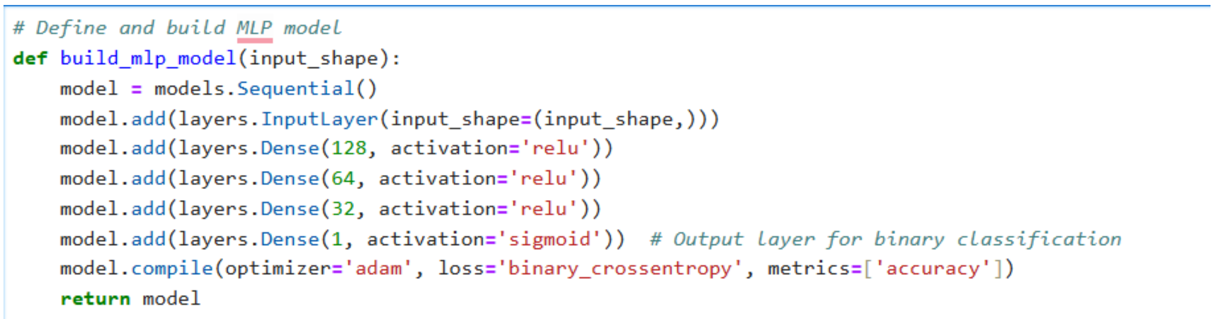
Now MLP is the type of fully connected feedforward neural network with more than one hidden layer, which is most commonly employed in classification problems.

Hyperparameters used in the code:

hidden_layer_sizes: Specifies the quantity of neurons of each hidden layer.

activation: Describes the activation function of the hidden layers, It is a function that is used to put an activator on the hidden layers.

solver: The optimization process to follow; This is known as the algorithm.

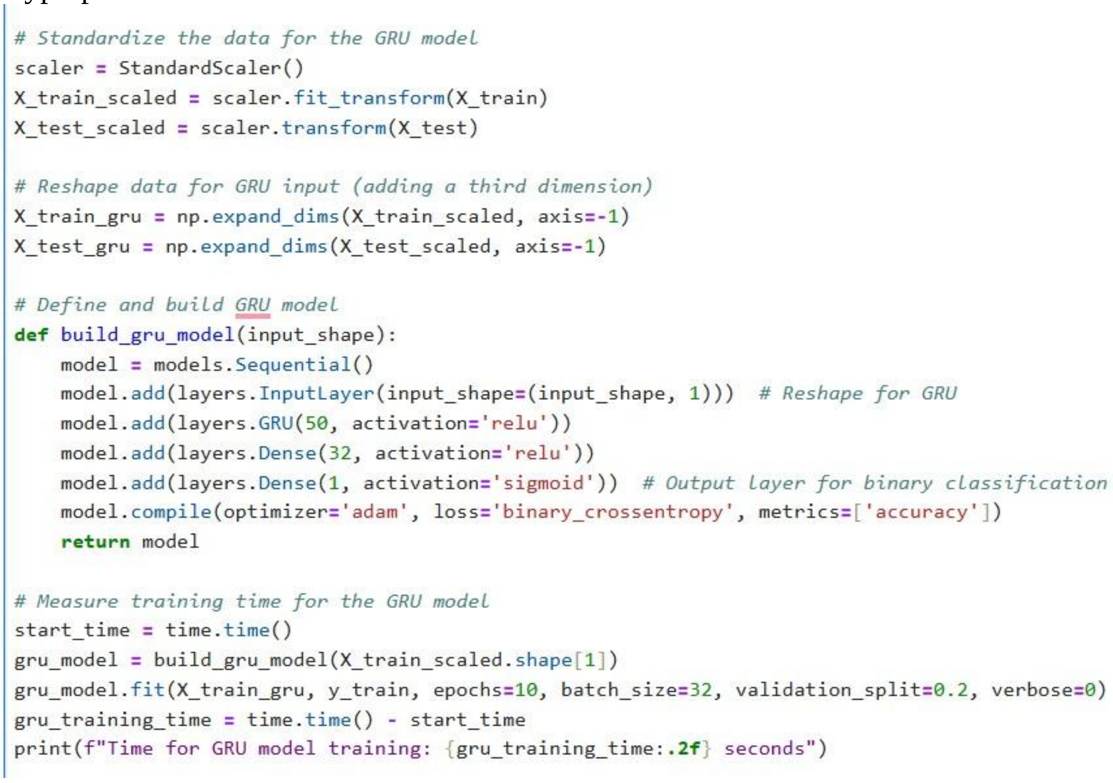


```python
# Define and build MLP model
def build_mlp_model(input_shape):
    model = models.Sequential()
    model.add(layers.InputLayer(input_shape=(input_shape,)))
    model.add(layers.Dense(128, activation='relu'))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(32, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))  # Output layer for binary classification
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

5.9 GRU

GRU is also an RNN based network; however, it’s simpler than LSTM but in most sequence types of problem, it is better or at least as good as the latter.

Hyperparameters used in the code below:

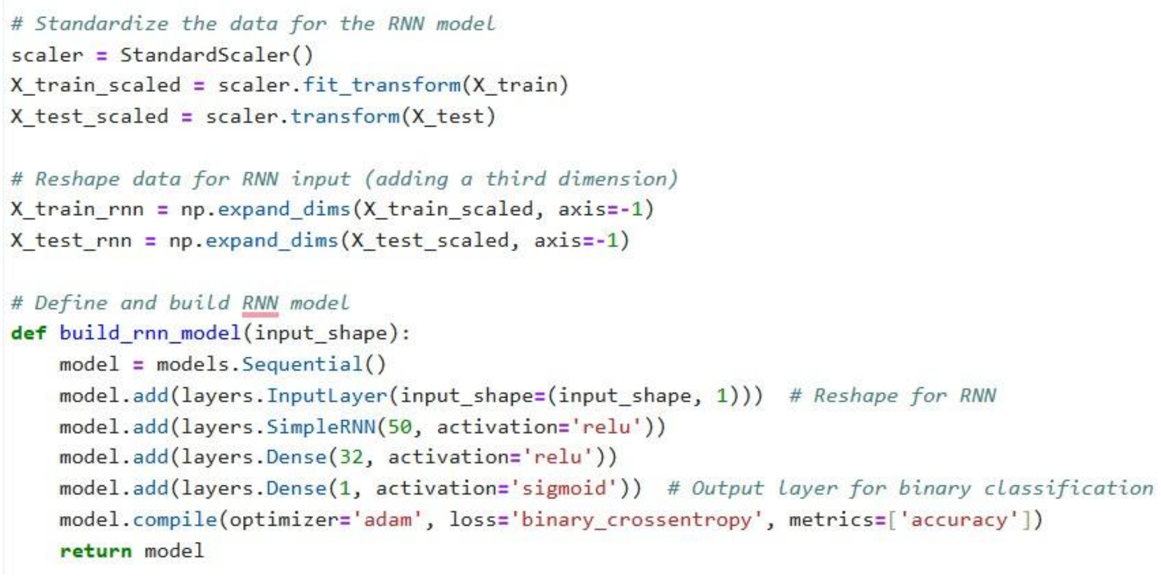


```python
# Standardize the data for the GRU model
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape data for GRU input (adding a third dimension)
X_train_gru = np.expand_dims(X_train_scaled, axis=-1)
X_test_gru = np.expand_dims(X_test_scaled, axis=-1)

# Define and build GRU model
def build_gru_model(input_shape):
    model = models.Sequential()
    model.add(layers.InputLayer(input_shape=(input_shape, 1)))  # Reshape for GRU
    model.add(layers.GRU(50, activation='relu'))
    model.add(layers.Dense(32, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))  # Output layer for binary classification
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Measure training time for the GRU model
start_time = time.time()
gru_model = build_gru_model(X_train_scaled.shape[1])
gru_model.fit(X_train_gru, y_train, epochs=10, batch_size=32, validation_split=0.2, verbose=0)
gru_training_time = time.time() - start_time
print(f"Time for GRU model training: {gru_training_time:.2f} seconds")
```

5.10 RNN

Recurrent Neural Networks (RNNs) are actors or machines whose performance is to use temporal data since the current output depends on previous inputs (text, time series data).

Hyperparameters used in the code below:
units: The number of neurons that is to be contained in the RNN layer of this network.

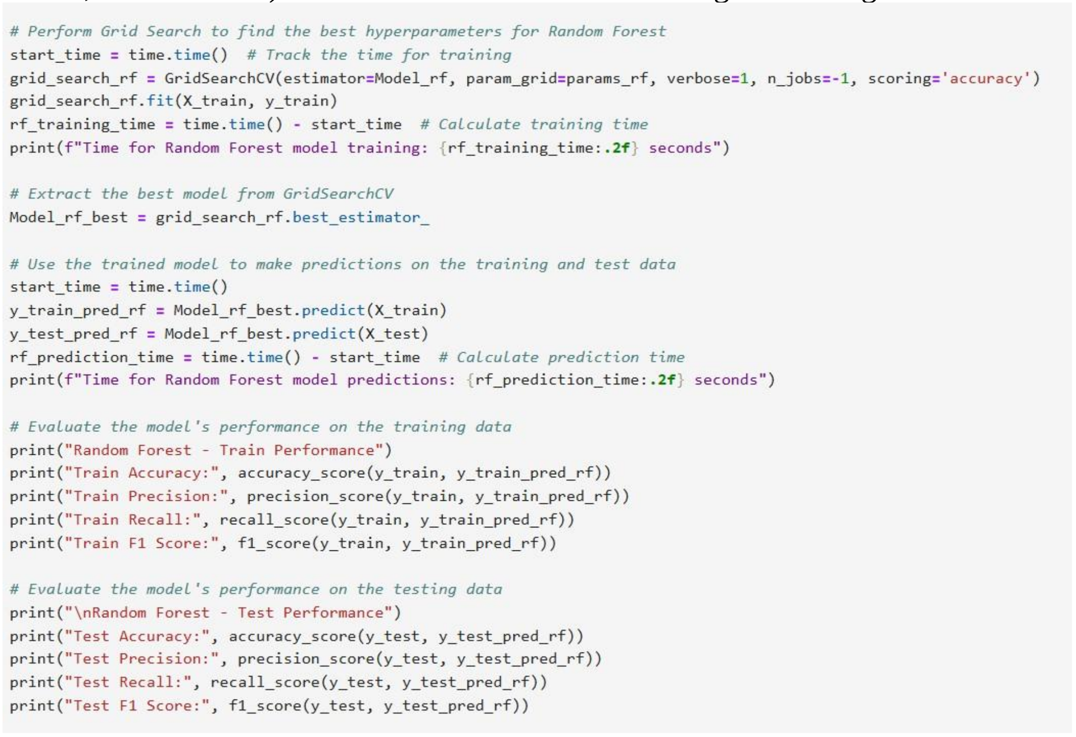


```
# Standardize the data for the RNN model
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape data for RNN input (adding a third dimension)
X_train_rnn = np.expand_dims(X_train_scaled, axis=-1)
X_test_rnn = np.expand_dims(X_test_scaled, axis=-1)

# Define and build RNN model
def build_rnn_model(input_shape):
    model = models.Sequential()
    model.add(layers.InputLayer(input_shape=(input_shape, 1)))  # Reshape for RNN
    model.add(layers.SimpleRNN(50, activation='relu'))
    model.add(layers.Dense(32, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))  # Output layer for binary classification
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

6. Results

6.1 Performance Metrics

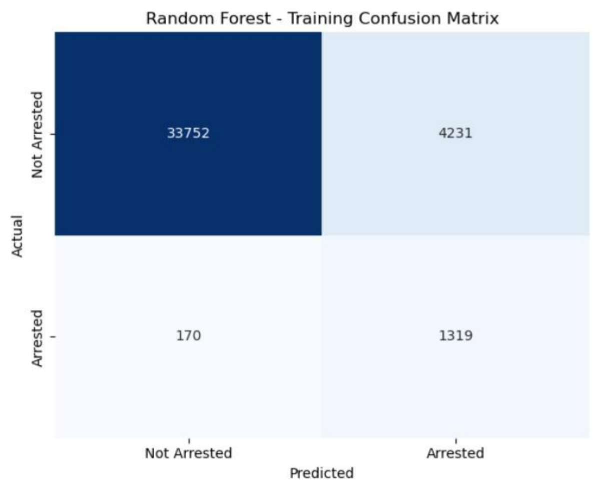
After training and predicting with the models, the performance metrics (Accuracy, Precision, Recall, and F1 Score) are calculated for both the training and testing sets:

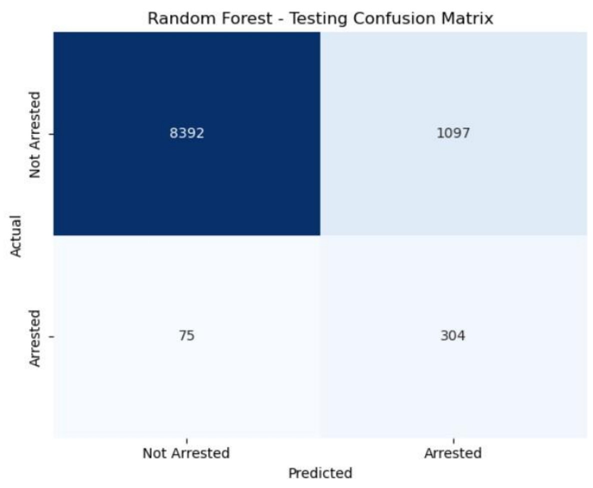


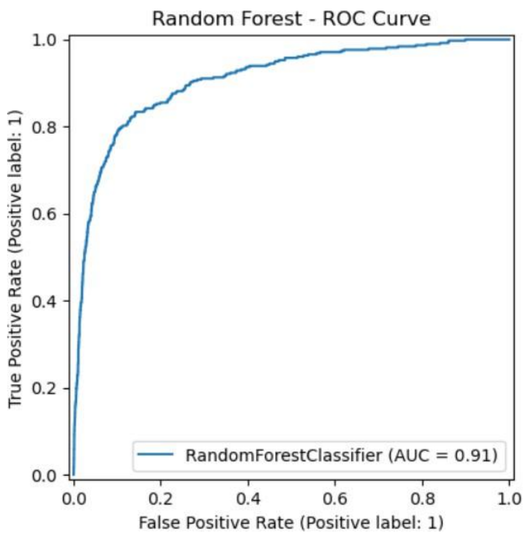
```
# Perform Grid Search to find the best hyperparameters for Random Forest
start_time = time.time()  # Track the time for training
grid_search_rf = GridSearchCV(estimator=Model_rf, param_grid=params_rf, verbose=1, n_jobs=-1, scoring='accuracy')
grid_search_rf.fit(X_train, y_train)
rf_training_time = time.time() - start_time  # Calculate training time
print(f"Time for Random Forest model training: {rf_training_time:.2f} seconds")

# Extract the best model from GridSearchCV
Model_rf_best = grid_search_rf.best_estimator_

# Use the trained model to make predictions on the training and test data
start_time = time.time()
y_train_pred_rf = Model_rf_best.predict(X_train)
y_test_pred_rf = Model_rf_best.predict(X_test)
rf_prediction_time = time.time() - start_time  # Calculate prediction time
print(f"Time for Random Forest model predictions: {rf_prediction_time:.2f} seconds")

# Evaluate the model's performance on the training data
print("Random Forest - Train Performance")
print("Train Accuracy:", accuracy_score(y_train, y_train_pred_rf))
print("Train Precision:", precision_score(y_train, y_train_pred_rf))
print("Train Recall:", recall_score(y_train, y_train_pred_rf))
print("Train F1 Score:", f1_score(y_train, y_train_pred_rf))

# Evaluate the model's performance on the testing data
print("\nRandom Forest - Test Performance")
print("Test Accuracy:", accuracy_score(y_test, y_test_pred_rf))
print("Test Precision:", precision_score(y_test, y_test_pred_rf))
print("Test Recall:", recall_score(y_test, y_test_pred_rf))
print("Test F1 Score:", f1_score(y_test, y_test_pred_rf))
```

6.2 Confusion Matrix & ROC Curve

A confusion matrix and Receiver Operating Characteristics curve plotting is done on individual models to summarize their performance.







6.3 Training and Prediction Time

The time taken for both training and predictions for the models is printed:

```
# Use the trained model to make predictions on the training and test data
start_time = time.time()
y_train_pred_rf = Model_rf_best.predict(X_train)
y_test_pred_rf = Model_rf_best.predict(X_test)
rf_prediction_time = time.time() - start_time  # Calculate prediction time
print(f"Time for Random Forest model predictions: {rf_prediction_time:.2f} seconds")
```

```
Time for Random Forest model training: 266.38 seconds
Time for Random Forest model predictions: 0.31 seconds
```

6.4 Final Computation Time

The total time for the entire computation process is printed at the end of the script:

```
# Calculate and print the total computation time
overall_end_time = time.time()
overall_time = overall_end_time - overall_start_time
print(f"\nTotal computation time: {overall_time:.2f} seconds")
```

```
Total computation time: 267.98 seconds
```

References

AIML.com (2023). *Compare the different Sequence models (RNN, LSTM, GRU, and Transformers)*. [online] AIML.com. Available at: https://aiml.com/compare-the-different-sequence-models-rnn-lstm-gru-and-transformers/.

Yan, Y., Guo, L., Li, J., Yu, Z., Sun, S., Xu, T., Zhao, H. and Guo, L. (2024). Hybrid GRU–Random Forest Model for Accurate Atmospheric Duct Detection with Incomplete Sounding Data. *Remote Sensing*, [online] 16(22), pp.4308–4308. doi:https://doi.org/10.3390/rs16224308.

Grieve, P. (2023). *A simple way to understand machine learning vs deep learning - Zendesk*. [online] Zendesk. Available at: https://www.zendesk.com/blog/machine-learning-and-deep-learning/.