Configuration Manual

MSc Research Project
MSc in Data Analytics

Pranav Hagavane
Student ID: 22209484

School of Computing
National College of Ireland

Supervisor: Furqan Rustam

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Pranav Hagavane		
Student ID:	22209484		
Programme:	MSc in Data Analytics	**Year:**	2024
Module:	MSc Research Project		
Lecturer:	Furqan Rustam		
Submission Due Date:	16/09/2024		
Project Title:	Evaluating the Impact of Environmental Conditions on Heat Pump Performance Using Machine Learning models		
Word Count:	886 words **Page Count:** 8 pages		

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: Pranav Hagavane

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

Pranav Hagavane
22209484

1 Introduction

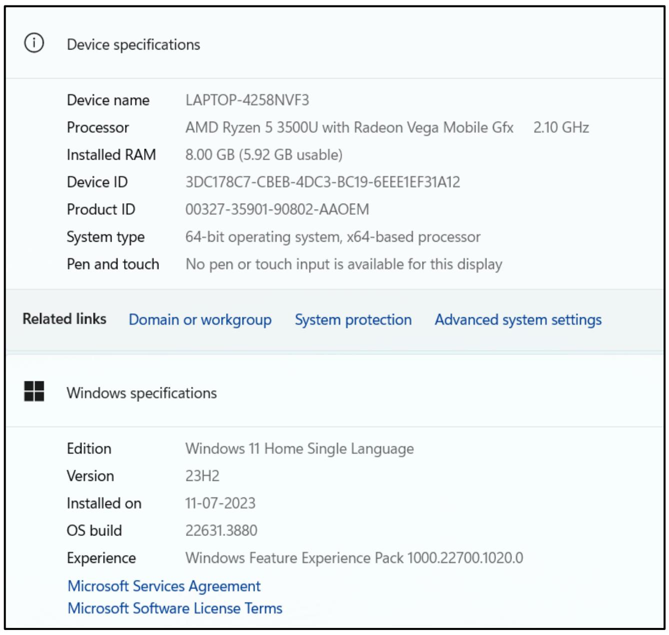
This document provides detailed information about the kind of configuration required to perform the code. This document discusses both the software and hardware requirements to set up the environment.

2 Environment

This section provides information about the software and hardware configuration required to the execute the code.

2.1 Hardware Configuration

Hardware specification used to execute the project is displayed using Figure 1.



Device specifications

Device name	LAPTOP-4258NVF3
Processor	AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz
Installed RAM	8.00 GB (5.92 GB usable)
Device ID	3DC178C7-CBEB-4DC3-BC19-6EEE1EF31A12
Product ID	00327-35901-90802-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Windows specifications

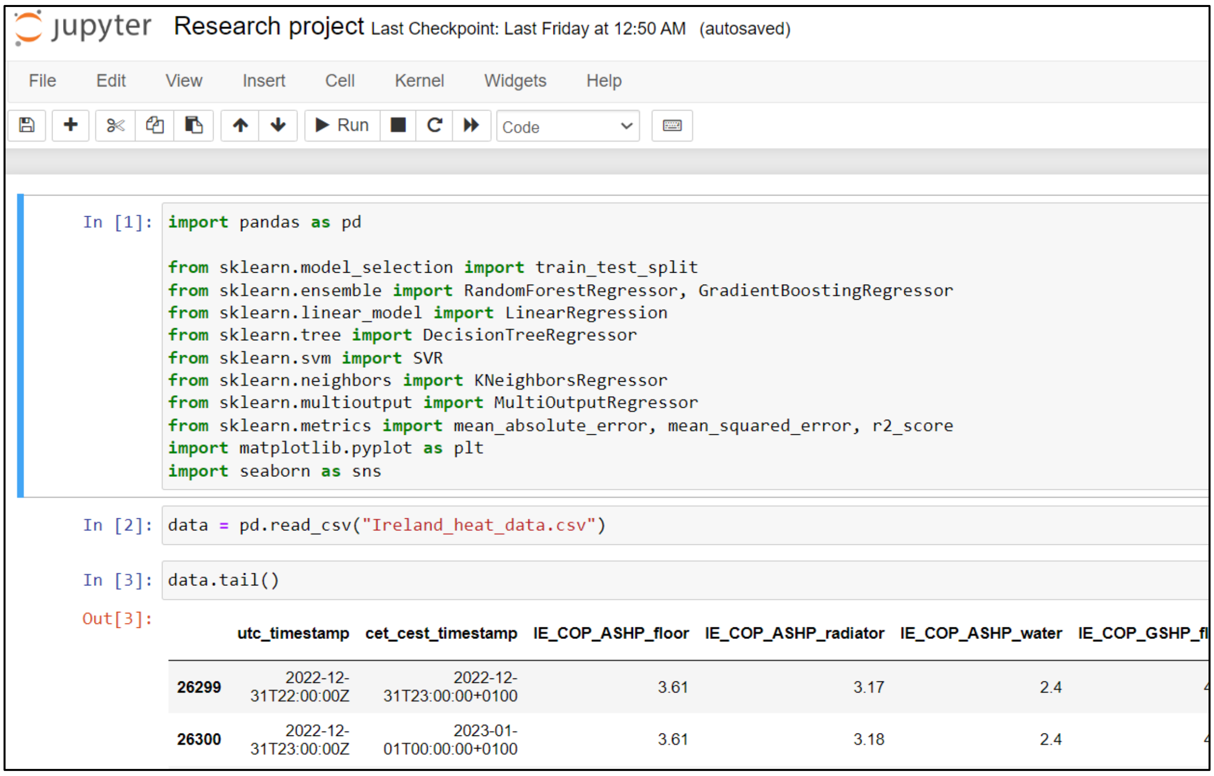
Edition	Windows 11 Home Single Language
Version	23H2
Installed on	11-07-2023
OS build	22631.3880
Experience	Windows Feature Experience Pack 1000.22700.1020.0

Microsoft Services Agreement
Microsoft Software License Terms

Figure 1 Hardware Configuration

2.2 Software Requirements

Python programming language is used to perform the research project. This python code was written using the jupyter notebook environment, which is interactive IDE to perform analysis and modelling. This is performed on python version 3


jupyter Research project Last Checkpoint: Last Friday at 12:50 AM (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

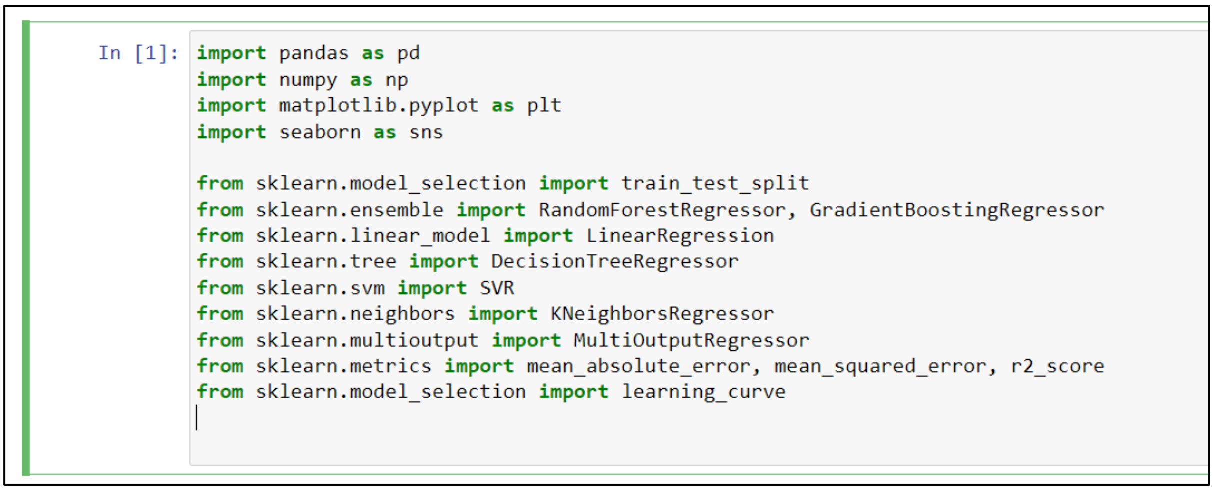
```
In [1]: import pandas as pd

        from sklearn.model_selection import train_test_split
        from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
        from sklearn.linear_model import LinearRegression
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.svm import SVR
        from sklearn.neighbors import KNeighborsRegressor
        from sklearn.multioutput import MultiOutputRegressor
        from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
        import matplotlib.pyplot as plt
        import seaborn as sns

In [2]: data = pd.read_csv("Ireland_heat_data.csv")

In [3]: data.tail()

Out[3]:
```

	utc_timestamp	cet_cest_timestamp	IE_COP_ASHP_floor	IE_COP_ASHP_radiator	IE_COP_ASHP_water	IE_COP_GSHP_fl
26299	2022-12-31T22:00:00Z	2022-12-31T23:00:00+0100	3.61	3.17	2.4	4
26300	2022-12-31T23:00:00Z	2023-01-01T00:00:00+0100	3.61	3.18	2.4	4

Figure 2 Jupyter Notebook Overview

3 Packages Required

Importing the necessary packages is the first step of the research, these packed can be installed using the pip command. Initially some packages such pandas, numpy and matplotlib are required to load the dataset and understand the data performing some operations which may require numpy. Other libraries can be loaded at the later stage of the project.

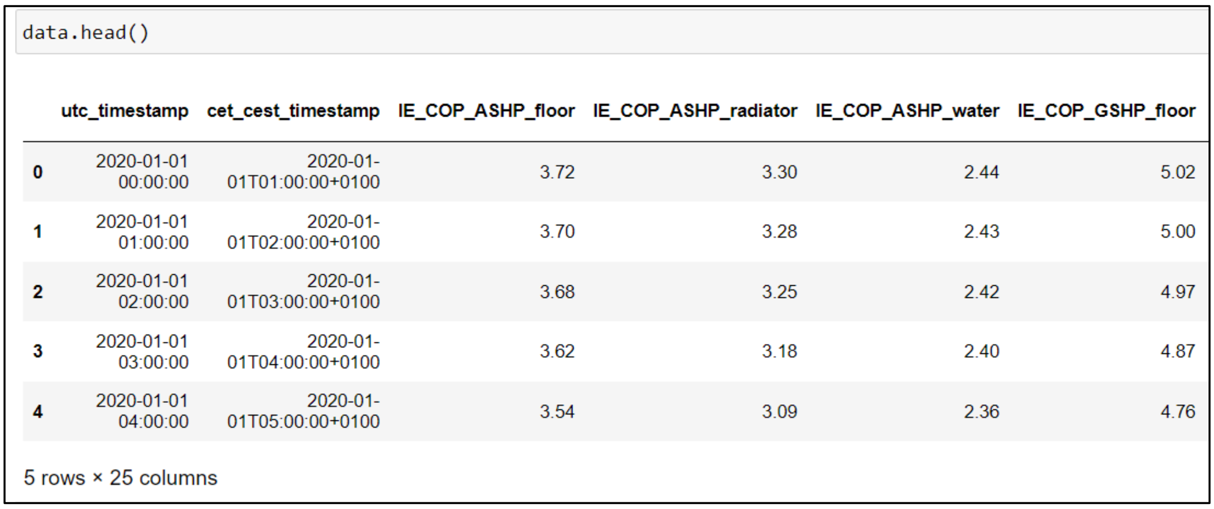


```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns

        from sklearn.model_selection import train_test_split
        from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
        from sklearn.linear_model import LinearRegression
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.svm import SVR
        from sklearn.neighbors import KNeighborsRegressor
        from sklearn.multioutput import MultiOutputRegressor
        from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
        from sklearn.model_selection import learning_curve
```

Figure 3 Importing Libraries

4 Dataset Description

Two datasets are being used in this research project, one dataset contains information about heat pumps[1] and other contains information about weather[2]. Both the datasets are merged using excel based on a common column time stamp.



```
data.head()
```

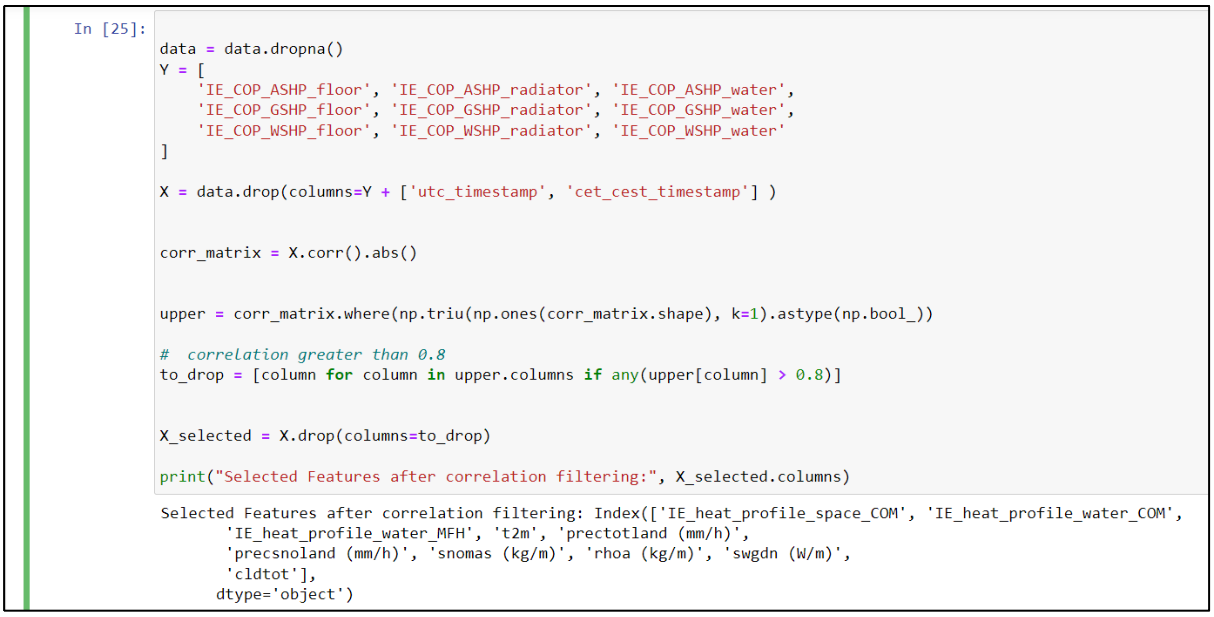
	utc_timestamp	cet_cest_timestamp	IE_COP_ASHP_floor	IE_COP_ASHP_radiator	IE_COP_ASHP_water	IE_COP_GSHP_floor
0	2020-01-01 00:00:00	2020-01-01T01:00:00+0100	3.72	3.30	2.44	5.02
1	2020-01-01 01:00:00	2020-01-01T02:00:00+0100	3.70	3.28	2.43	5.00
2	2020-01-01 02:00:00	2020-01-01T03:00:00+0100	3.68	3.25	2.42	4.97
3	2020-01-01 03:00:00	2020-01-01T04:00:00+0100	3.62	3.18	2.40	4.87
4	2020-01-01 04:00:00	2020-01-01T05:00:00+0100	3.54	3.09	2.36	4.76

5 rows × 25 columns

Figure 4 Dataset Imported

5 Data Preparation

The data is being prepared by deleting all the null values and also performed feature selection afterwards the data is being divided into train and test set to train model.

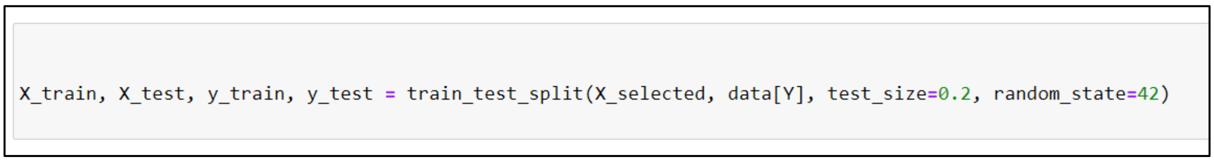


```
In [25]:
data = data.dropna()
Y = [
    'IE_COP_ASHP_floor', 'IE_COP_ASHP_radiator', 'IE_COP_ASHP_water',
    'IE_COP_GSHP_floor', 'IE_COP_GSHP_radiator', 'IE_COP_GSHP_water',
    'IE_COP_WSHP_floor', 'IE_COP_WSHP_radiator', 'IE_COP_WSHP_water'
]

X = data.drop(columns=Y + ['utc_timestamp', 'cet_cest_timestamp'] )


corr_matrix = X.corr().abs()


upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool_))

#  correlation greater than 0.8
to_drop = [column for column in upper.columns if any(upper[column] > 0.8)]


X_selected = X.drop(columns=to_drop)

print("Selected Features after correlation filtering:", X_selected.columns)
```

```
Selected Features after correlation filtering: Index(['IE_heat_profile_space_COM', 'IE_heat_profile_water_COM',
       'IE_heat_profile_water_MFH', 't2m', 'prectotland (mm/h)',
       'precsnoland (mm/h)', 'snomas (kg/m)', 'rhoa (kg/m)', 'swgdn (W/m)',
       'cldtot'],
      dtype='object')
```

Figure 5 Feature Selection

[1] https://data.open-power-system-data.org/when2heat/latest/
[2] https://www.renewables.ninja/#/country



```
X_train, X_test, y_train, y_test = train_test_split(X_selected, data[Y], test_size=0.2, random_state=42)
```

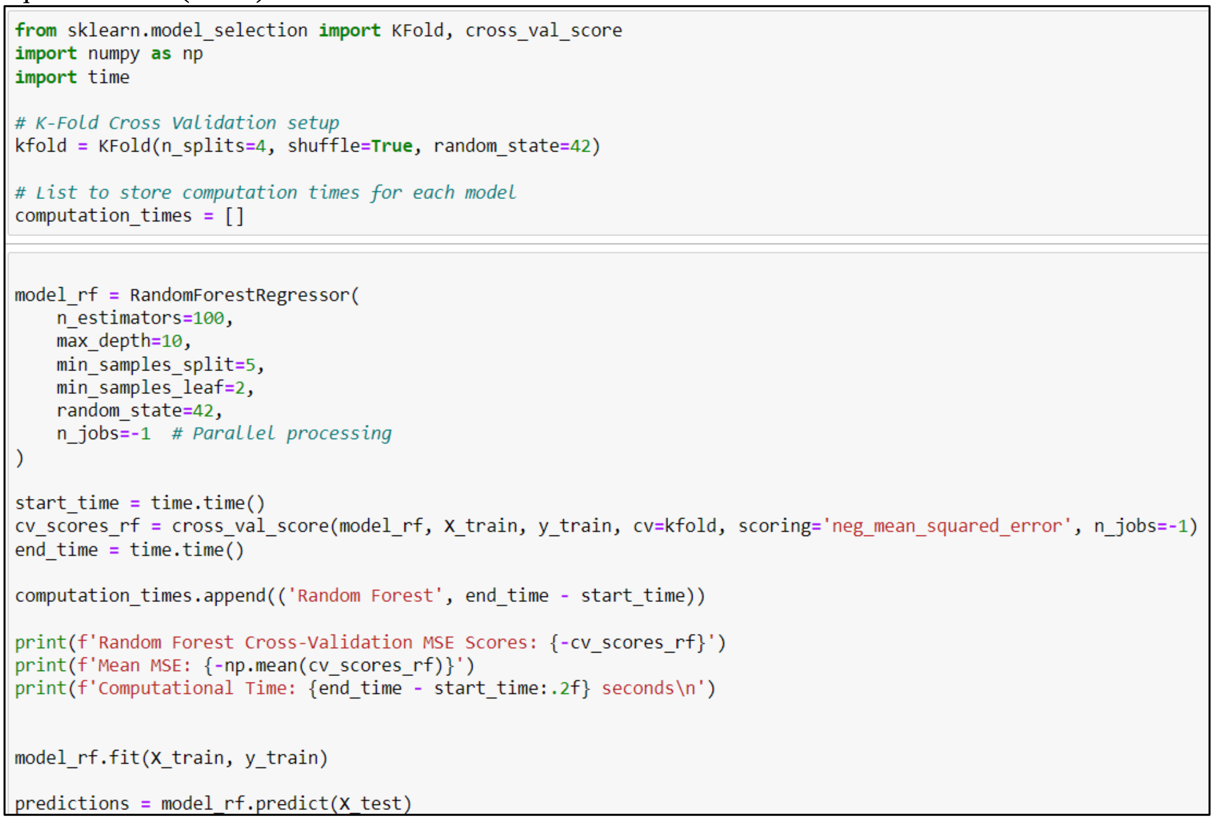
Figure 6 Train and test split

6 Model Preparation

There are six machine learning model and two deep learning models used in this research. The six machine learning model are Random Forest, Gradient Boost, Decision Tree, KNN, Support Vector Regression and Vector Regressor and the two deep learning model are LSTM and MLP.

6.1 Random Forest

A Random Forest model was employed using 100 estimators, with a maximum depth of 10, and minimum samples split and leaf values set to 5 and 2, respectively. K-Fold cross-validation (with 4 splits) was performed to evaluate the performance, using negative mean squared error (MSE).



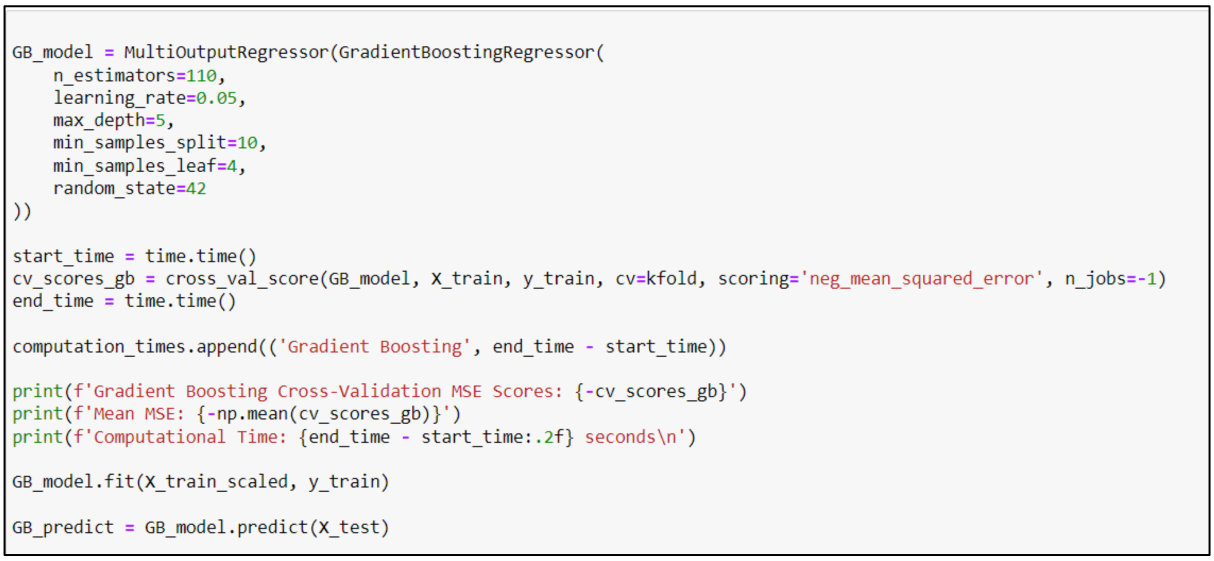
```
from sklearn.model_selection import KFold, cross_val_score
import numpy as np
import time

# K-Fold Cross Validation setup
kfold = KFold(n_splits=4, shuffle=True, random_state=42)

# List to store computation times for each model
computation_times = []
```

```
model_rf = RandomForestRegressor(
    n_estimators=100,
    max_depth=10,
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42,
    n_jobs=-1  # Parallel processing
)

start_time = time.time()
cv_scores_rf = cross_val_score(model_rf, X_train, y_train, cv=kfold, scoring='neg_mean_squared_error', n_jobs=-1)
end_time = time.time()

computation_times.append(('Random Forest', end_time - start_time))

print(f'Random Forest Cross-Validation MSE Scores: {-cv_scores_rf}')
print(f'Mean MSE: {-np.mean(cv_scores_rf)}')
print(f'Computational Time: {end_time - start_time:.2f} seconds\n')


model_rf.fit(X_train, y_train)

predictions = model_rf.predict(X_test)
```

Figure 7 Random Forest

6.2 Gradient Boosting

A Gradient Boosting Regressor was used and wrapped in a MultiOutputRegressor to handle multi-output regression. The model was configured with 110 estimators, a learning rate of 0.05, and a maximum depth of 5, with additional hyperparameters for controlling split and leaf size. K-Fold cross-validation (4 splits) was applied to evaluate the model, using negative MSE, and the computation time was recorded. After evaluation, the model was fitted and used to predict on the test set.

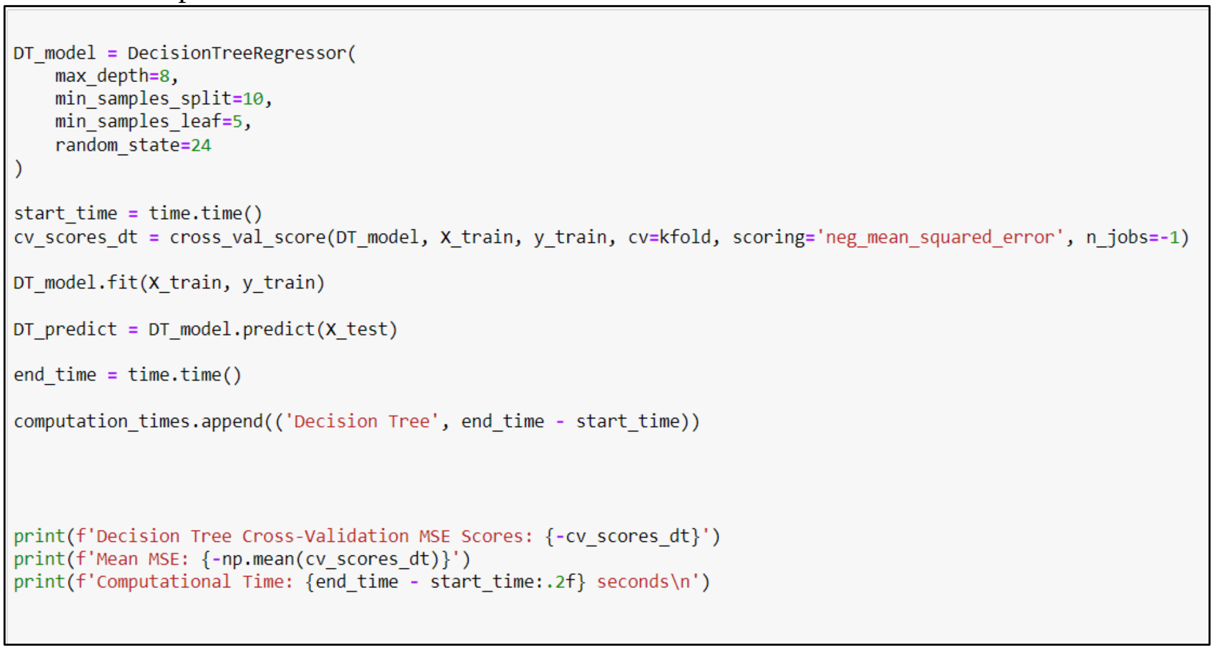


```python
GB_model = MultiOutputRegressor(GradientBoostingRegressor(
    n_estimators=110,
    learning_rate=0.05,
    max_depth=5,
    min_samples_split=10,
    min_samples_leaf=4,
    random_state=42
))

start_time = time.time()
cv_scores_gb = cross_val_score(GB_model, X_train, y_train, cv=kfold, scoring='neg_mean_squared_error', n_jobs=-1)
end_time = time.time()

computation_times.append(('Gradient Boosting', end_time - start_time))

print(f'Gradient Boosting Cross-Validation MSE Scores: {-cv_scores_gb}')
print(f'Mean MSE: {-np.mean(cv_scores_gb)}')
print(f'Computational Time: {end_time - start_time:.2f} seconds\n')

GB_model.fit(X_train_scaled, y_train)

GB_predict = GB_model.predict(X_test)
```

Figure 8 Gradient Boosting

6.3 Decision Tree

A Decision Tree Regressor was configured with a maximum depth of 8, a minimum sample split of 10, and a minimum sample leaf size of 5. K-Fold cross-validation with 4 splits was performed to evaluate the model using negative mean squared error (MSE) as the performance metric. After recording the computational time, the model was trained on the training set and used to make predictions on the test set.

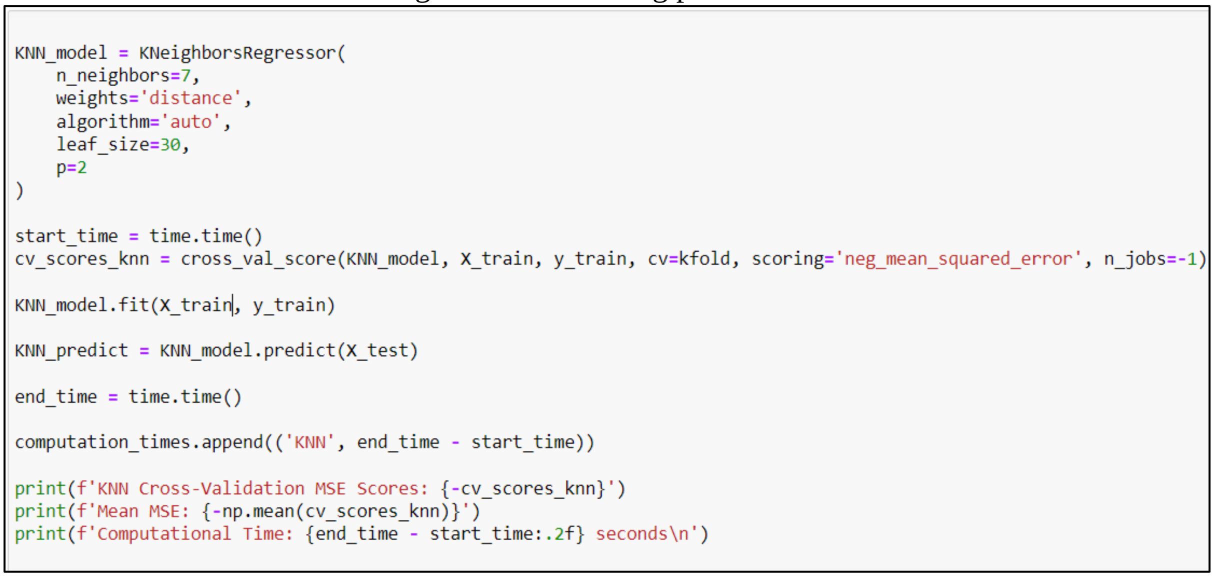


```python
DT_model = DecisionTreeRegressor(
    max_depth=8,
    min_samples_split=10,
    min_samples_leaf=5,
    random_state=24
)

start_time = time.time()
cv_scores_dt = cross_val_score(DT_model, X_train, y_train, cv=kfold, scoring='neg_mean_squared_error', n_jobs=-1)

DT_model.fit(X_train, y_train)

DT_predict = DT_model.predict(X_test)

end_time = time.time()

computation_times.append(('Decision Tree', end_time - start_time))



print(f'Decision Tree Cross-Validation MSE Scores: {-cv_scores_dt}')
print(f'Mean MSE: {-np.mean(cv_scores_dt)}')
print(f'Computational Time: {end_time - start_time:.2f} seconds\n')
```

Figure 9 Decision Tree

6.4 K – Nearest Neighbour (KNN)

The K-Nearest Neighbors (KNN) Regressor was implemented with 7 neighbors and a distance-based weighting scheme. Cross-validation was performed using K-Fold to assess model performance using negative mean squared error. The computation time was recorded, and the model was trained on the training set before making predictions on the test set.

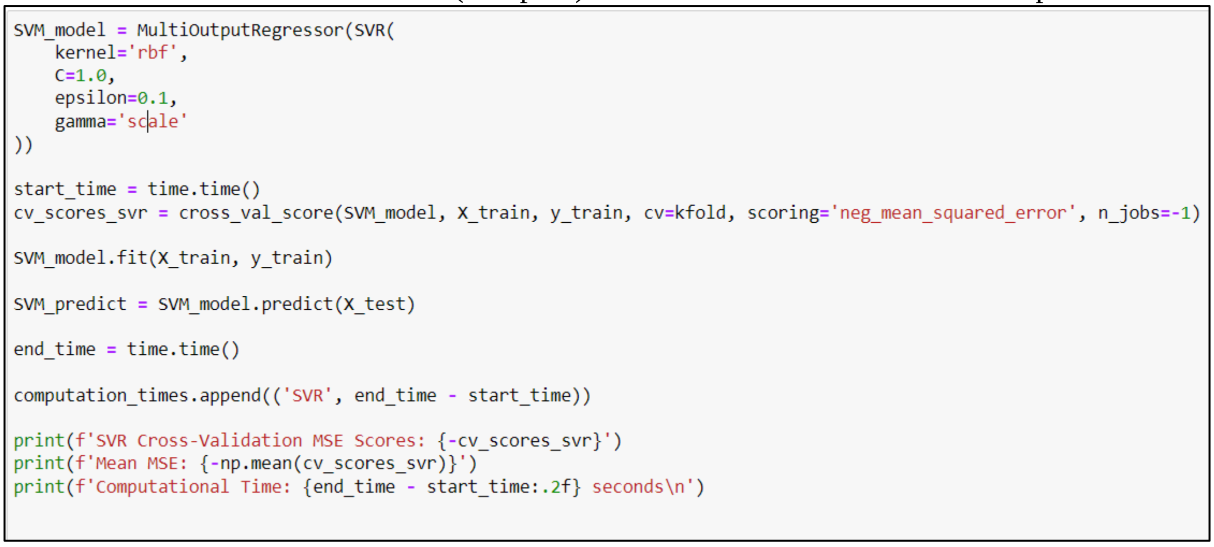


```python
KNN_model = KNeighborsRegressor(
    n_neighbors=7,
    weights='distance',
    algorithm='auto',
    leaf_size=30,
    p=2
)

start_time = time.time()
cv_scores_knn = cross_val_score(KNN_model, X_train, y_train, cv=kfold, scoring='neg_mean_squared_error', n_jobs=-1)

KNN_model.fit(X_train, y_train)

KNN_predict = KNN_model.predict(X_test)

end_time = time.time()

computation_times.append(('KNN', end_time - start_time))

print(f'KNN Cross-Validation MSE Scores: {-cv_scores_knn}')
print(f'Mean MSE: {-np.mean(cv_scores_knn)}')
print(f'Computational Time: {end_time - start_time:.2f} seconds\n')
```

Figure 10 KNN

6.5 Support Vector Regression (SVR)

A Support Vector Regressor (SVR) was employed using a radial basis function (RBF) kernel, with a regularization parameter C=1.0, an epsilon value of 0.1, and the gamma parameter set to 'scale'. The model was wrapped in a MultiOutputRegressor to handle multi-output regression tasks. K-Fold cross-validation (4 splits) was conducted to evaluate performance.

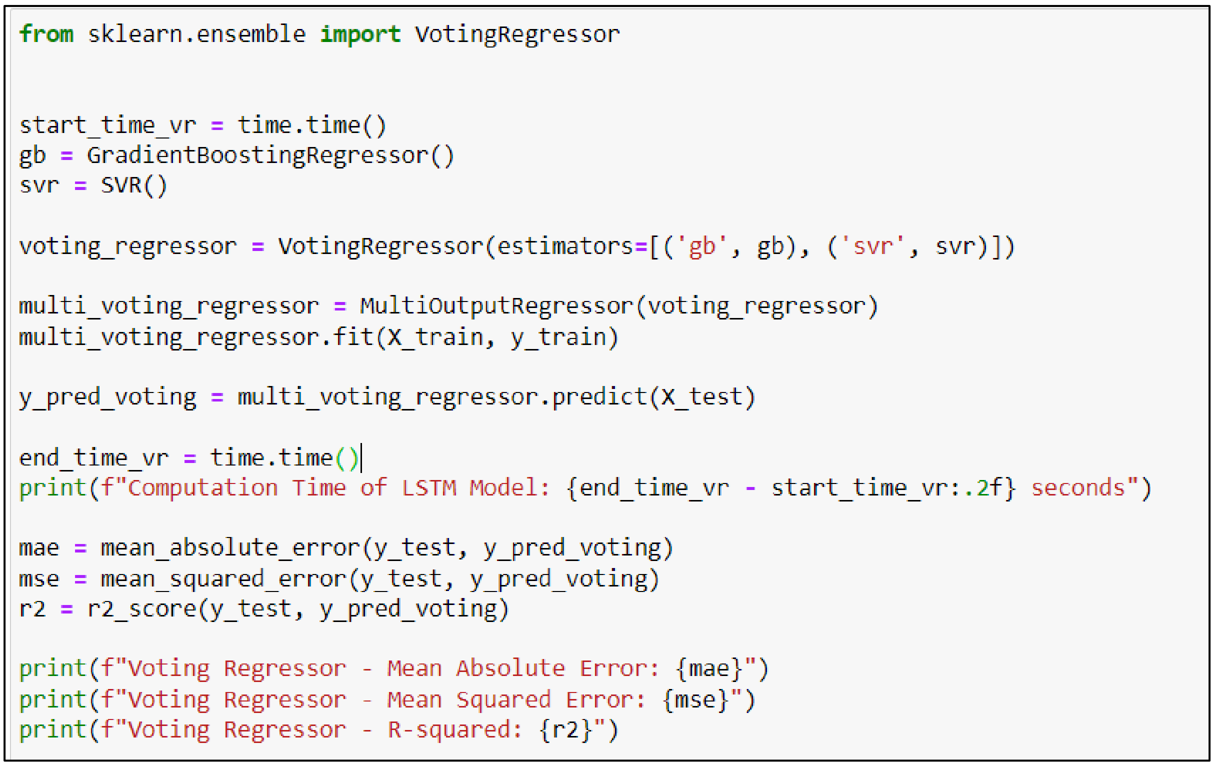


```python
SVM_model = MultiOutputRegressor(SVR(
    kernel='rbf',
    C=1.0,
    epsilon=0.1,
    gamma='scale'
))

start_time = time.time()
cv_scores_svr = cross_val_score(SVM_model, X_train, y_train, cv=kfold, scoring='neg_mean_squared_error', n_jobs=-1)

SVM_model.fit(X_train, y_train)

SVM_predict = SVM_model.predict(X_test)

end_time = time.time()

computation_times.append(('SVR', end_time - start_time))

print(f'SVR Cross-Validation MSE Scores: {-cv_scores_svr}')
print(f'Mean MSE: {-np.mean(cv_scores_svr)}')
print(f'Computational Time: {end_time - start_time:.2f} seconds\n')
```

Figure 11 SVR

6.6 Voting Regressor

A Voting Regressor was built by combining two base models Gradient Boosting Regressor and a Support Vector Regressor (SVR). This ensemble approach aggregates the predictions from both models to improve overall performance. The model was wrapped in a MultiOutputRegressor to handle multi-output regression tasks. The Voting Regressor was trained and evaluated using the test set using the evaluation metrics.

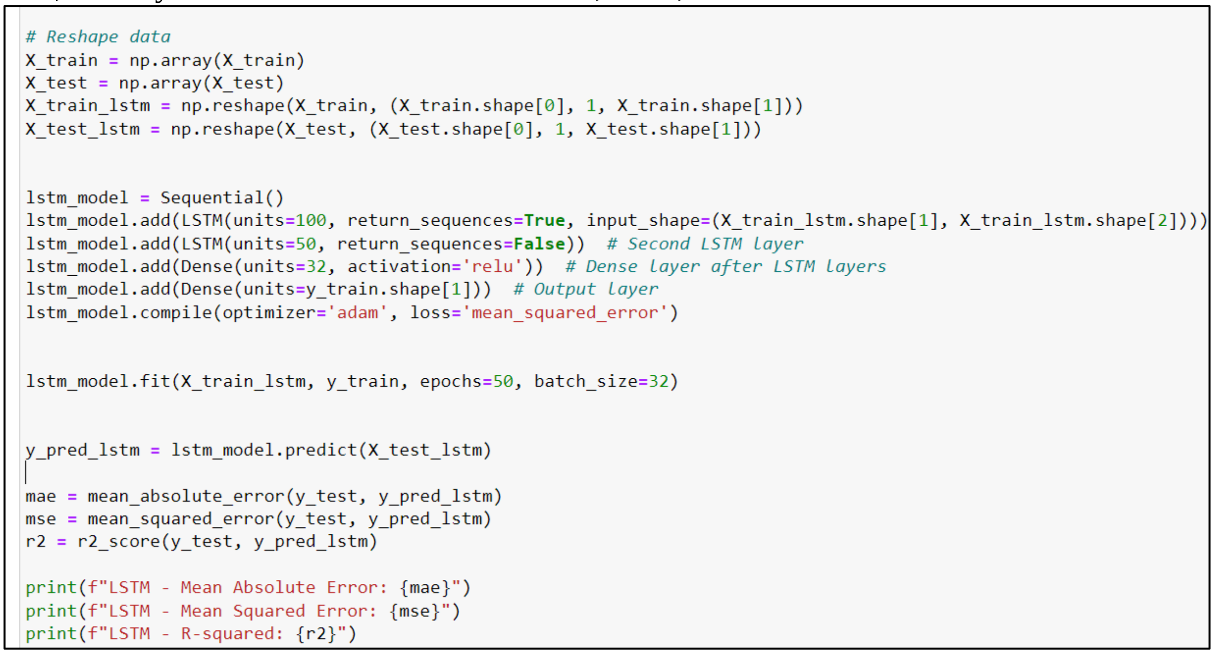


```python
from sklearn.ensemble import VotingRegressor


start_time_vr = time.time()
gb = GradientBoostingRegressor()
svr = SVR()

voting_regressor = VotingRegressor(estimators=[('gb', gb), ('svr', svr)])

multi_voting_regressor = MultiOutputRegressor(voting_regressor)
multi_voting_regressor.fit(X_train, y_train)

y_pred_voting = multi_voting_regressor.predict(X_test)

end_time_vr = time.time()
print(f"Computation Time of LSTM Model: {end_time_vr - start_time_vr:.2f} seconds")

mae = mean_absolute_error(y_test, y_pred_voting)
mse = mean_squared_error(y_test, y_pred_voting)
r2 = r2_score(y_test, y_pred_voting)

print(f"Voting Regressor - Mean Absolute Error: {mae}")
print(f"Voting Regressor - Mean Squared Error: {mse}")
print(f"Voting Regressor - R-squared: {r2}")
```

Figure 12 Voting Regressor

6.7 Long Short-Term Memory (LSTM)

An LSTM neural network was implemented for the time-series prediction task, consisting of two LSTM layers with 100 and 50 units, respectively. A Dense layer with 32 units and ReLU activation was added, followed by an output layer to predict the target variables. The model was trained for 50 epochs with a batch size of 32, using the Adam optimizer and mean squared error as the loss function. After training, predictions were made on the reshaped test data, and key evaluation metrics such as MAE, MSE, and R² were calculated.

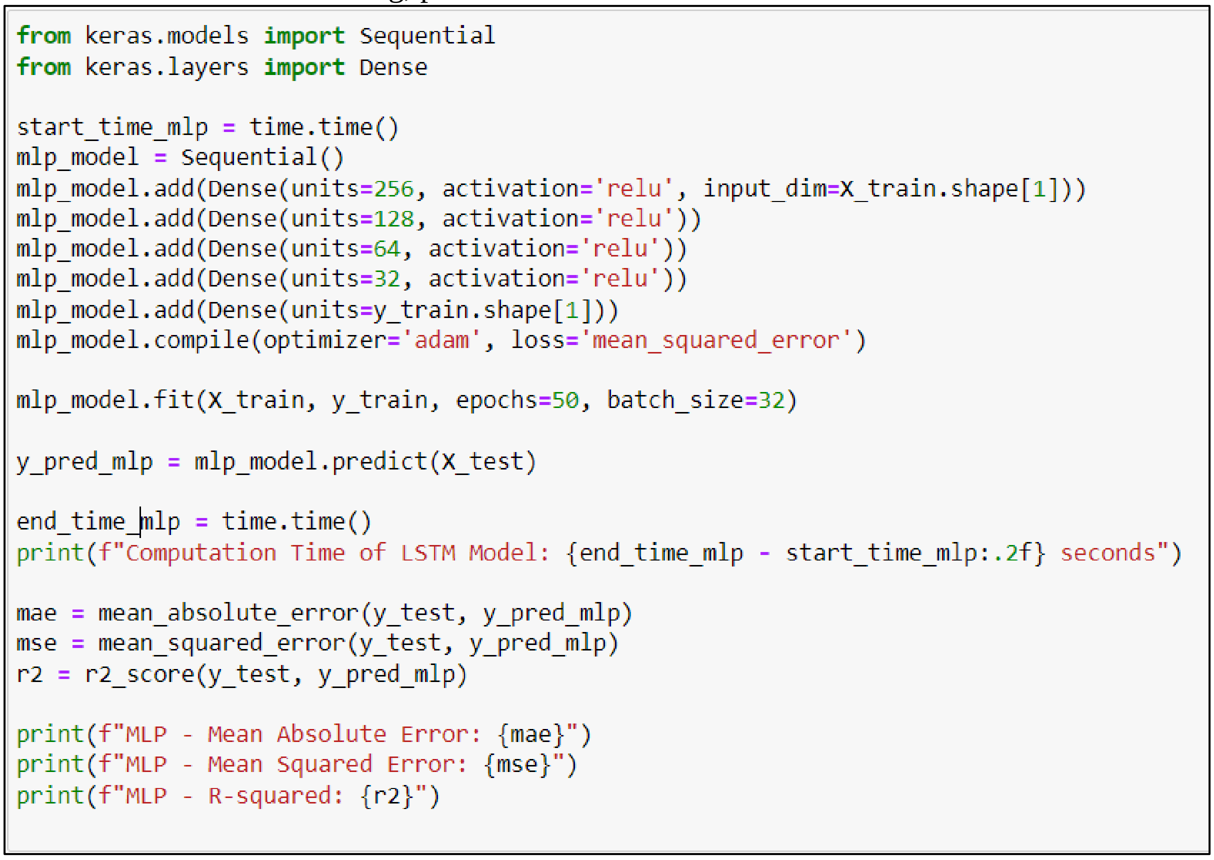


```python
# Reshape data
X_train = np.array(X_train)
X_test = np.array(X_test)
X_train_lstm = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test_lstm = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))


lstm_model = Sequential()
lstm_model.add(LSTM(units=100, return_sequences=True, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
lstm_model.add(LSTM(units=50, return_sequences=False))  # Second LSTM layer
lstm_model.add(Dense(units=32, activation='relu'))  # Dense layer after LSTM layers
lstm_model.add(Dense(units=y_train.shape[1]))  # Output layer
lstm_model.compile(optimizer='adam', loss='mean_squared_error')


lstm_model.fit(X_train_lstm, y_train, epochs=50, batch_size=32)


y_pred_lstm = lstm_model.predict(X_test_lstm)

mae = mean_absolute_error(y_test, y_pred_lstm)
mse = mean_squared_error(y_test, y_pred_lstm)
r2 = r2_score(y_test, y_pred_lstm)

print(f"LSTM - Mean Absolute Error: {mae}")
print(f"LSTM - Mean Squared Error: {mse}")
print(f"LSTM - R-squared: {r2}")
```

Figure 13 LSTM

6.8 Multi-Layer Perceptron (MLP)

A Multi-Layer Perceptron (MLP) model was constructed using a series of dense layers with ReLU activation. The model architecture included layers with 256, 128, 64, and 32 units, followed by an output layer corresponding to the target dimensions. The model was trained for 50 epochs with a batch size of 32, using the Adam optimizer and mean squared error as the loss function. After training, predictions were made on the test set.



```python
from keras.models import Sequential
from keras.layers import Dense

start_time_mlp = time.time()
mlp_model = Sequential()
mlp_model.add(Dense(units=256, activation='relu', input_dim=X_train.shape[1]))
mlp_model.add(Dense(units=128, activation='relu'))
mlp_model.add(Dense(units=64, activation='relu'))
mlp_model.add(Dense(units=32, activation='relu'))
mlp_model.add(Dense(units=y_train.shape[1]))
mlp_model.compile(optimizer='adam', loss='mean_squared_error')

mlp_model.fit(X_train, y_train, epochs=50, batch_size=32)

y_pred_mlp = mlp_model.predict(X_test)

end_time_mlp = time.time()
print(f"Computation Time of LSTM Model: {end_time_mlp - start_time_mlp:.2f} seconds")

mae = mean_absolute_error(y_test, y_pred_mlp)
mse = mean_squared_error(y_test, y_pred_mlp)
r2 = r2_score(y_test, y_pred_mlp)

print(f"MLP - Mean Absolute Error: {mae}")
print(f"MLP - Mean Squared Error: {mse}")
print(f"MLP - R-squared: {r2}")
```

Figure 14 MLP