Configuration Manual

MSc Research Project
Data Analytics

Harsh Gupta
Student ID: x23173815

School of Computing
National College of Ireland

Supervisor: Prof. Jorge Basilio

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Harsh Gupta
Student ID:	x23173815
Programme:	MSc in Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Prof. Jorge Basilio
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	1237
Page Count:	13

Signature:	Harsh Gupta
Date:	25th January 2025

Configuration Manual

Harsh Gupta
x23173815

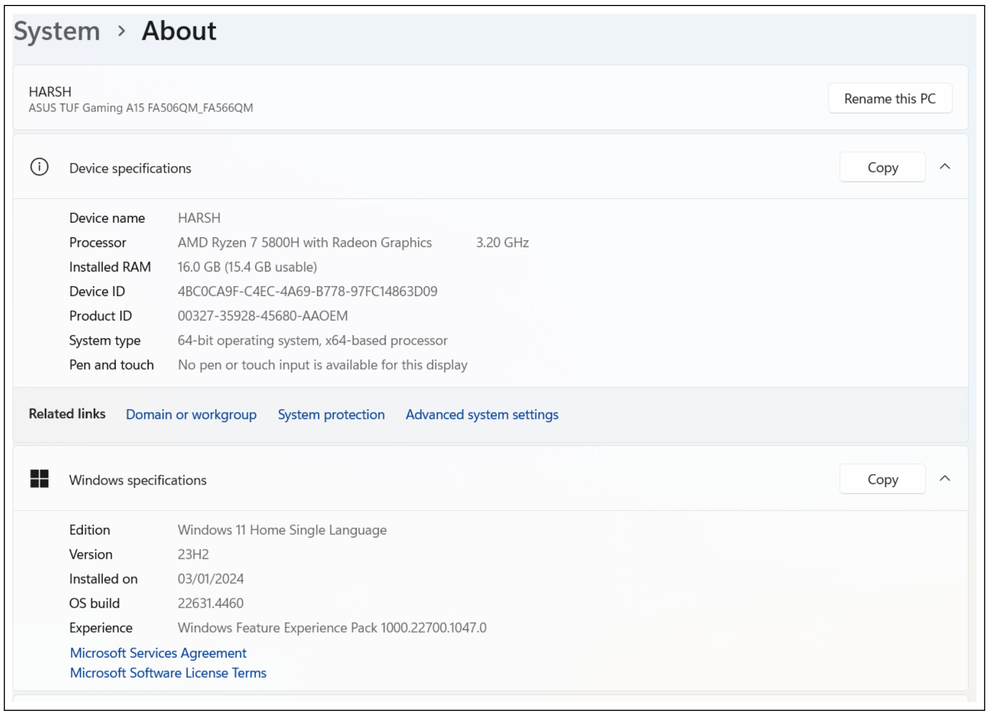
1 Introduction

The purpose of this document is to take you through each steps required to run the project in the system. The document provides all the information from setting up the hardware and software for the research. The document also provides guidance on the research process, including all the steps like data preparation, pre-processing, building of model and evaluate its performance in a detailed manner.

2 Hardware and Software Requirements

2.1 Hardware Configuration

The work in this research is done on a personal laptop. Below, the system configurations are shown in Figure 1 . The system uses AMD Ryzen 7 5800H processor with Radeon Graphics, running at 3.20 GHz. The RAM installed in the system is 16GB RAM and it is a 64-bit Operating System. The Graphic Card used is - NVIDIA GeForce RTX 3060 Laptop GPU.



HARSH
ASUS TUF Gaming A15 FA506QM_FA566QM

Device specifications

Device name	HARSH	
Processor	AMD Ryzen 7 5800H with Radeon Graphics	3.20 GHz
Installed RAM	16.0 GB (15.4 GB usable)	
Device ID	4BC0CA9F-C4EC-4A69-B778-97FC14863D09	
Product ID	00327-35928-45680-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	03/01/2024
OS build	22631.4460
Experience	Windows Feature Experience Pack 1000.22700.1047.0

Figure 1: System Configuration

2.2 Software Configuration

This section of the document provides the software requirements setup that are necessary for the implementation of the research. The following software's must be installed as shown in Table 1 on the system to ensure smooth implementation:

S. No.	Software Requirements
1	Anaconda Navigator
2	Python 3.11.5
3	Jupyter Notebook
4	Google Colaboratory (if cloud-based processing is required)
5	Microsoft Office Suite: Microsoft Word, Microsoft Excel, Microsoft PowerPoint
6	Web Browser: Google Chrome or Microsoft Edge

Table 1: Software Requirements

3 Development Tools for Image Creation

This section outline the tools required and used for creation of images and diagrams during this research. The following tools are utilized:

- **Miro:** `https://miro.com/`
- **App.eraser:** `https://app.eraser.io/workspace/K0tyVB4EZaN8vbsHdbLE?origin=share`

4 Downloading the Jupyter Notebook

4.1 Step 1

The first step involves the process of downloading Anaconda Navigator from the given link:

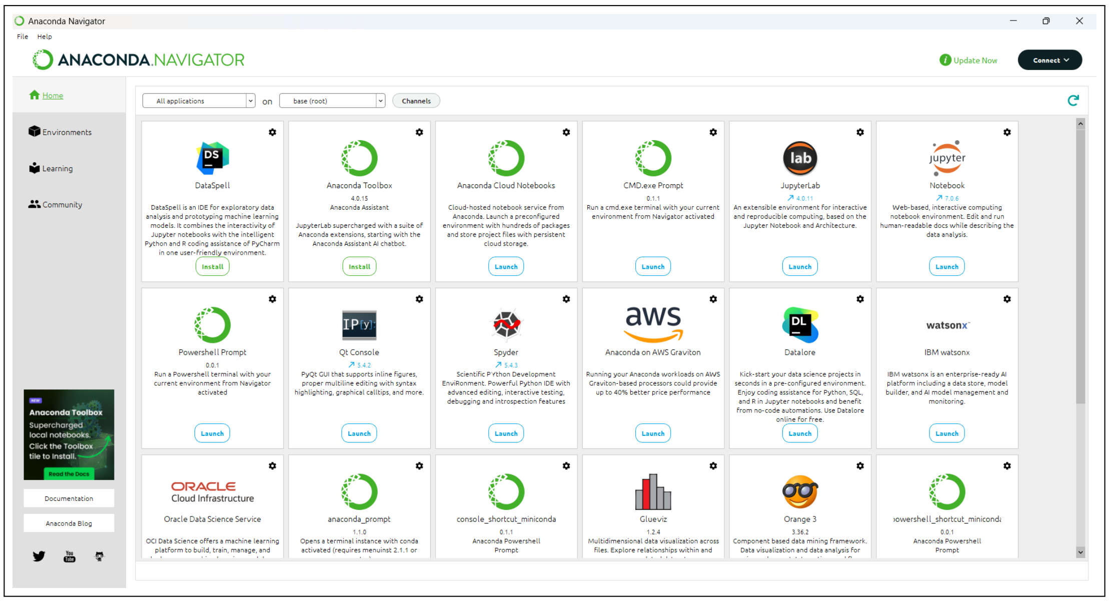
- `https://www.anaconda.com/products/navigator`

Alternatively, users can utilize Google Colaboratory, which can be accessed via the following link:

- `https://colab.google/`

4.2 Step 2

Now using the search window, `'Click on START'`, search for `'Anaconda Navigator'` and then click on the icon and `'Run the Application'`. Once you click on the `'Application'`, the screen will have a display which is shown below in the Figure 2.



DataSpell
DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.
Install

Anaconda Toolbox
4.0.15
Anaconda Assistant
JupyterLab supercharged with a suite of Anaconda extensions, starting with the Anaconda Assistant AI chatbot.
Install

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

CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated
Launch

JupyterLab
4.0.11
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch

Notebook
7.0.6
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
5.4.2
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch

Spyder
5.4.3
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch

Anaconda on AWS Graviton
Running your Anaconda workloads on AWS Graviton-based processors could provide up to 40% better price performance
Launch

Datalore
Kick-start your data science projects in seconds in a pre-configured environment. Enjoy coding assistance for Python, SQL, and R in Jupyter notebooks and benefit from no-code automations. Use Datalore online for free.
Launch

IBM watsonx
IBM watsonx is an enterprise-ready AI platform including a data store, model builder, and AI model management and monitoring.
Launch

Oracle Data Science Service
OCI Data Science offers a machine learning platform to build, train, manage, and

anaconda_prompt
1.1.0
Opens a terminal instance with conda activated (requires menuinst 2.1.1 or

console_shortcut_miniconda
0.1.1
Anaconda Powershell Prompt

Glueviz
1.2.4
Multidimensional data visualization across files. Explore relationships within and

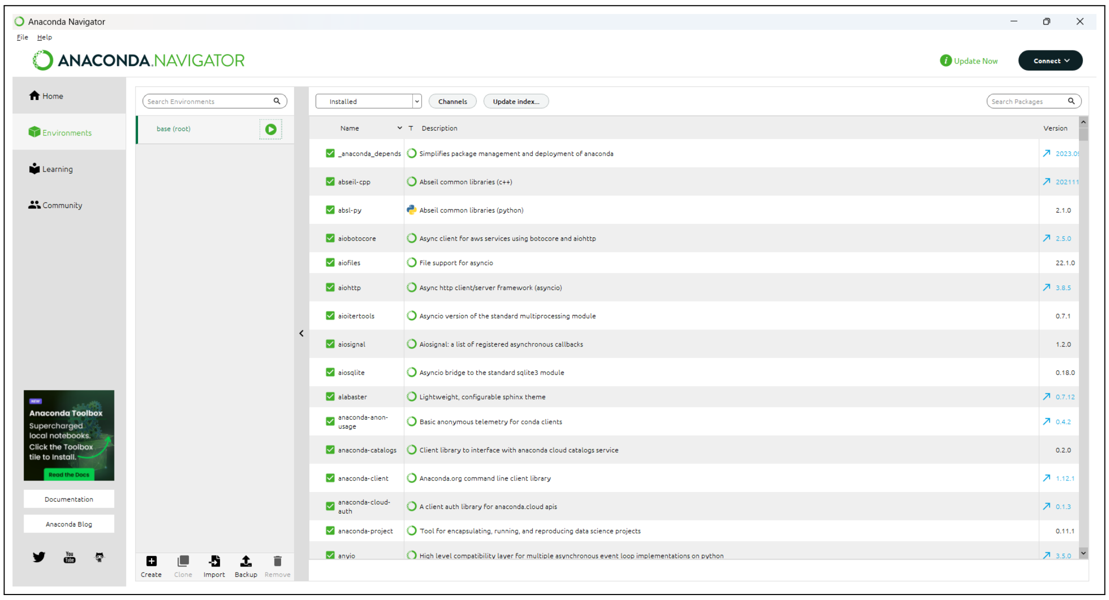
Orange 3
3.36.2
Component based data mining framework. Data visualization and data analysis for

owershell_shortcut_minicond
0.0.1
Anaconda Powershell Prompt

Figure 2: Anaconda Navigator Window

4.3 Step 3

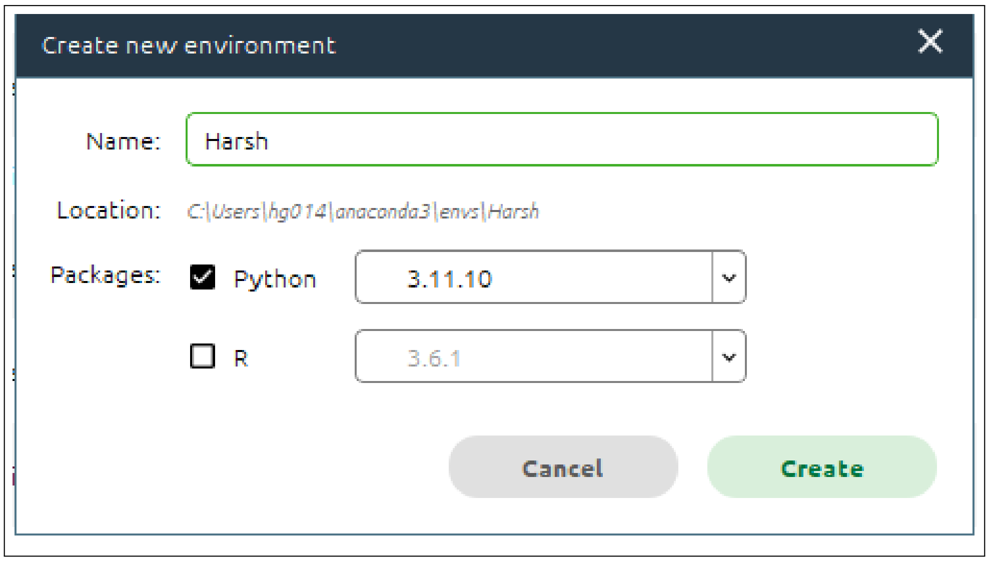
After this, just look for "Jupyter Notebook" and "Download" it on your local machine. Now, after the installation is complete, we can use the search bar or command prompt to open the Jupyter Notebook. The Figure 3 below, shows the creation of python environment in the Anaconda Navigator in which Jupyter Notebook will run.



Name	Description	Version
✓ _anaconda_depends	Simplifies package management and deployment of anaconda	2023.0
✓ abseil-cpp	Abseil common libraries (c++)	202111
✓ absl-py	Abseil common libraries (python)	2.1.0
✓ aiobotocore	Async client for aws services using botocore and aiohttp	2.5.0
✓ aiofiles	File support for asyncio	22.1.0
✓ aiohttp	Async http client/server framework (asyncio)	3.8.5
✓ aioitertools	Asyncio version of the standard multiprocessing module	0.7.1
✓ aiosignal	Aiosignal: a list of registered asynchronous callbacks	1.2.0
✓ aiosqlite	Asyncio bridge to the standard sqlite3 module	0.18.0
✓ alabaster	Lightweight, configurable sphinx theme	0.7.12
✓ anaconda-anon-usage	Basic anonymous telemetry for conda clients	0.4.2
✓ anaconda-catalogs	Client library to interface with anaconda cloud catalogs service	0.2.0
✓ anaconda-client	Anaconda.org command line client library	1.12.1
✓ anaconda-cloud-auth	A client auth library for anaconda.cloud apis	0.1.3
✓ anaconda-project	Tool for encapsulating, running, and reproducing data science projects	0.11.1
✓ anyio	High level compatibility layer for multiple asynchronous event loop implementations on python	3.5.0

Figure 3: Left Tab Environment

Once you click on the environment option, you will see a `'Create Option'`. After you click on the Create Option which is at the very bottom of the screen you will see a pop-up window on the screen. You will have to select the environment name and the python version which is shown in Figure 4 below.


Create new environment

Name: Harsh

Location: C:\Users\hg014\anaconda3\envs\Harsh

Packages: ✓ Python 3.11.10

☐ R 3.6.1

Figure 4: Creating a New Python Environment

5 Dealing with Zip file and loading it in Jupyter Notebook

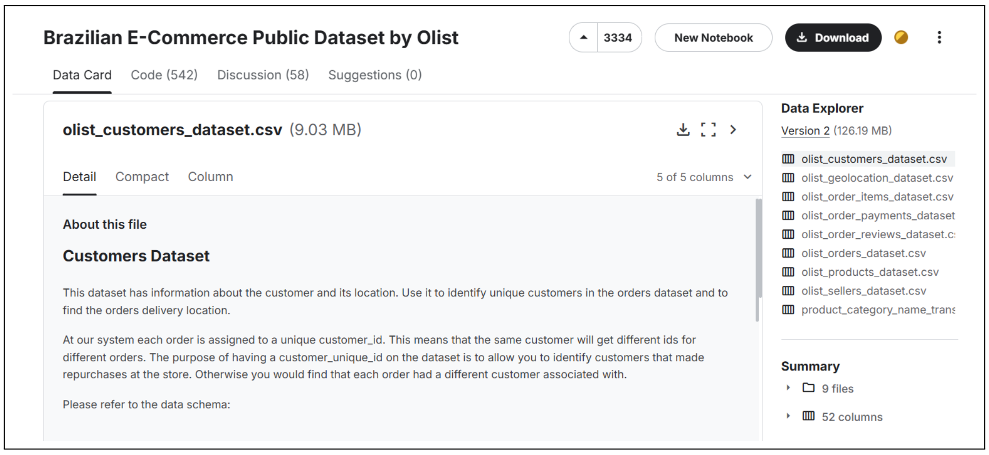
The file will be downloaded in a zip format (i.e. `.zip`). You will have to extract the zip file and upload it to the Jupyter Notebook for the further implementation of research work seen in Figure 5.

Figure 5: Extracting zip folder

6 Methodology and Implementation

6.1 Dataset Collection and Preparation

- **Step 1:** The dataset was collected from one of the famous public dataset repository named *Kaggle*, as shown below in Figure 6.


Brazilian E-Commerce Public Dataset by Olist
3334
New Notebook
Download
Data Card
Code (542)
Discussion (58)
Suggestions (0)
olist_customers_dataset.csv (9.03 MB)
Detail
Compact
Column
5 of 5 columns
Data Explorer
Version 2 (126.19 MB)
olist_customers_dataset.csv
olist_geolocation_dataset.csv
olist_order_items_dataset.csv
olist_order_payments_dataset
olist_order_reviews_dataset.c
olist_orders_dataset.csv
olist_products_dataset.csv
olist_sellers_dataset.csv
product_category_name_trans
Summary
9 files
52 columns

About this file

Customers Dataset

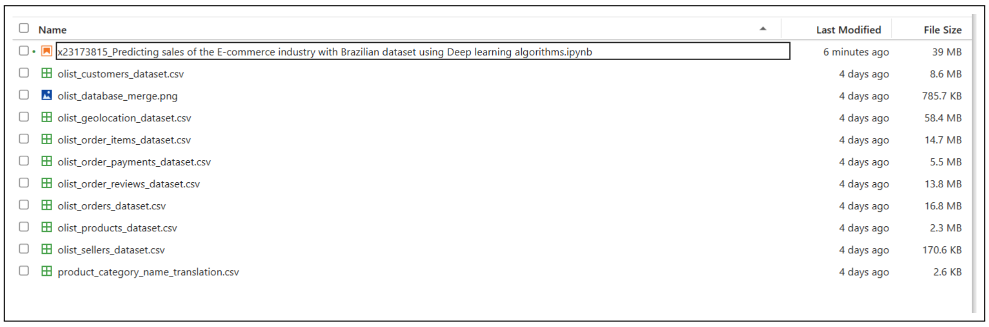
This dataset has information about the customer and its location. Use it to identify unique customers in the orders dataset and to find the orders delivery location.

At our system each order is assigned to a unique customer_id. This means that the same customer will get different ids for different orders. The purpose of having a customer_unique_id on the dataset is to allow you to identify customers that made repurchases at the store. Otherwise you would find that each order had a different customer associated with.

Please refer to the data schema:

Figure 6: Collection of Data

- **Step 2:** When the dataset has been downloaded by clicking on 'Download' button, it will be in the zip format around '45mb'. When the file is extracted, you will see 9 datasets which are:
  - *olist_customers_dataset*
  - *olist_geolocation_dataset*
  - *olist_order_items_dataset*
  - *olist_order_payments_dataset*
  - *olist_order_reviews_dataset*
  - *olist_orders_dataset*
  - *olist_products_dataset*
  - *olist_sellers_dataset*
  - *product_category_name_translation*
- **Step 3:** The next step involves loading these datasets in your local directory or elsewhere you can also use the main zip folder as show in Figure 5 previously and get all the datasets from there and load them.
- **Step 4:** Now in case, if you have extracted the entire zip file you can upload everything to Jupyter Notebook. This process is shown in Figure 7.



Name	Last Modified	File Size
x23173815_Predicting sales of the E-commerce industry with Brazilian dataset using Deep learning algorithms.ipynb	6 minutes ago	39 MB
olist_customers_dataset.csv	4 days ago	8.6 MB
olist_database_merge.png	4 days ago	785.7 KB
olist_geolocation_dataset.csv	4 days ago	58.4 MB
olist_order_items_dataset.csv	4 days ago	14.7 MB
olist_order_payments_dataset.csv	4 days ago	5.5 MB
olist_order_reviews_dataset.csv	4 days ago	13.8 MB
olist_orders_dataset.csv	4 days ago	16.8 MB
olist_products_dataset.csv	4 days ago	2.3 MB
olist_sellers_dataset.csv	4 days ago	170.6 KB
product_category_name_translation.csv	4 days ago	2.6 KB

Figure 7: Uploading to Jupyter Notebook

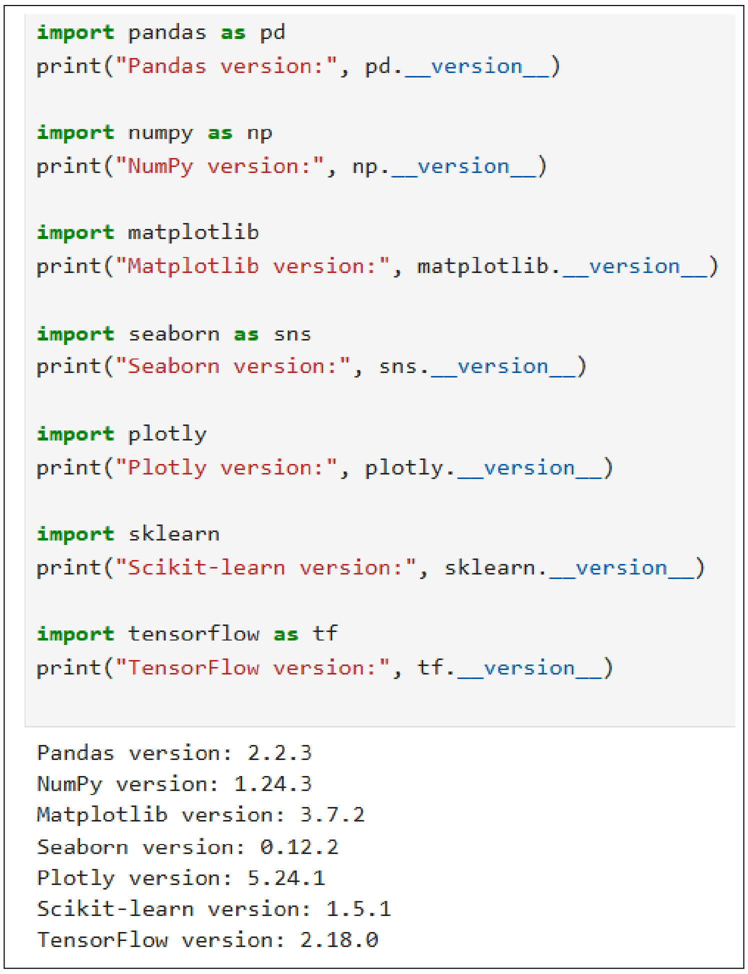
6.2 Downloading and Importing Libraries Required

- **Step 1:** The necessity of installing all the libraries required is a very essential step for ensuring the smooth implementation of the research work. The libraries are required for various tasks like manipulation of data, visualization, building of model and so on. Below, is the list of all the libraries and the version requirements as per our need in Table 2.

Library Name	Installation Command
pandas	`!pip install pandas==2.2.3`
numpy	`!pip install numpy==1.24.3`
matplotlib	`!pip install matplotlib==3.7.2`
seaborn	`!pip install seaborn==0.12.2`
plotly	`!pip install plotly==5.24.1`
scikit-learn	`!pip install scikit-learn==1.5.1`
tensorflow	`!pip install tensorflow==2.18.0`

Table 2: Library Installation Commands

- **Step 2:** Once, all the libraries are installed you can check the version of all libraries using Python commands shown in Figure 8.

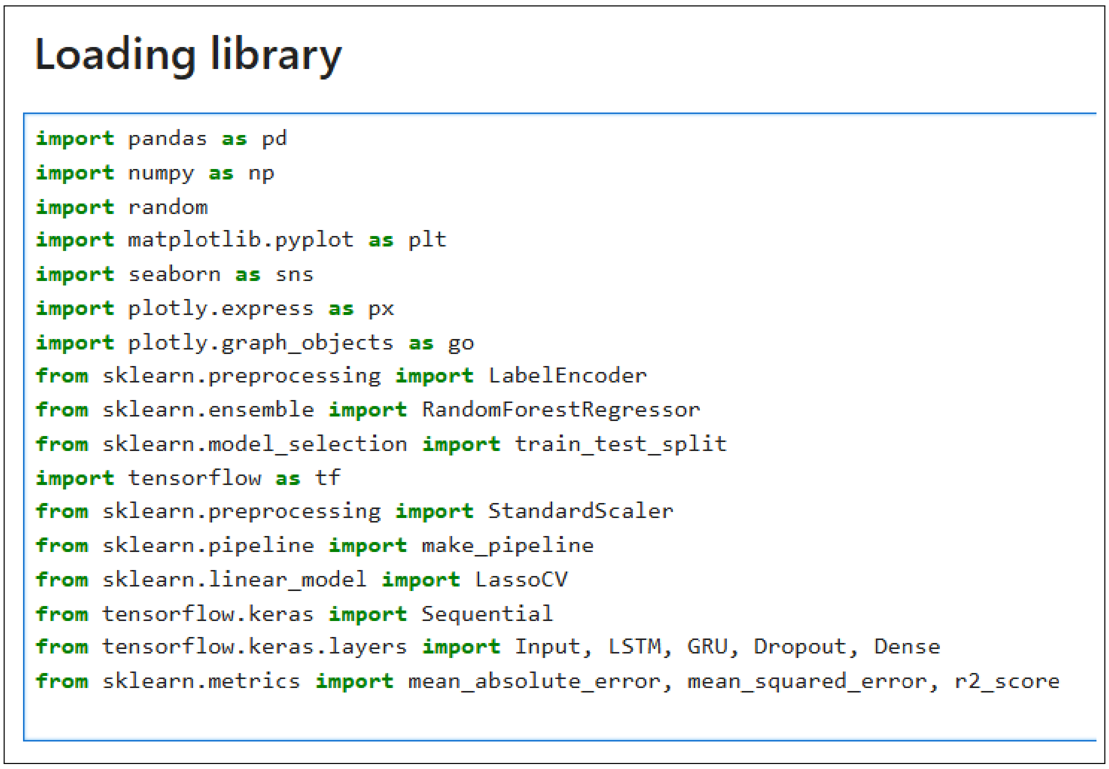


```
import pandas as pd
print("Pandas version:", pd.__version__)

import numpy as np
print("NumPy version:", np.__version__)

import matplotlib
print("Matplotlib version:", matplotlib.__version__)

import seaborn as sns
print("Seaborn version:", sns.__version__)

import plotly
print("Plotly version:", plotly.__version__)

import sklearn
print("Scikit-learn version:", sklearn.__version__)

import tensorflow as tf
print("TensorFlow version:", tf.__version__)
```

```
Pandas version: 2.2.3
NumPy version: 1.24.3
Matplotlib version: 3.7.2
Seaborn version: 0.12.2
Plotly version: 5.24.1
Scikit-learn version: 1.5.1
TensorFlow version: 2.18.0
```

Figure 8: Checking the version details of libraries

- **Step 3:** Once everything is ready and installed, now all the libraries can be imported using the commands shown in the Figure 9.



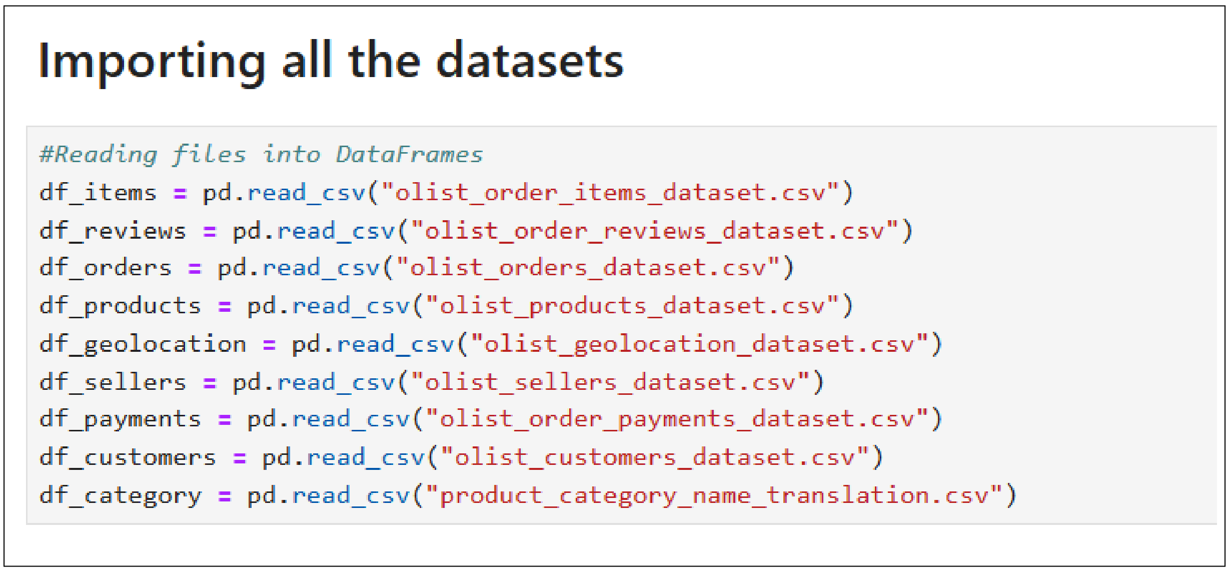
Loading library

```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LassoCV
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Input, LSTM, GRU, Dropout, Dense
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Figure 9: Importing all the libraries

6.3 Accessing the Datasets in Local Machine

The dataset can be accessed using a particular python command. The Figure 10 depicts the way of accessing all the 9 datasets in the local machine.



Importing all the datasets

```
#Reading files into DataFrames
df_items = pd.read_csv("olist_order_items_dataset.csv")
df_reviews = pd.read_csv("olist_order_reviews_dataset.csv")
df_orders = pd.read_csv("olist_orders_dataset.csv")
df_products = pd.read_csv("olist_products_dataset.csv")
df_geolocation = pd.read_csv("olist_geolocation_dataset.csv")
df_sellers = pd.read_csv("olist_sellers_dataset.csv")
df_payments = pd.read_csv("olist_order_payments_dataset.csv")
df_customers = pd.read_csv("olist_customers_dataset.csv")
df_category = pd.read_csv("product_category_name_translation.csv")
```

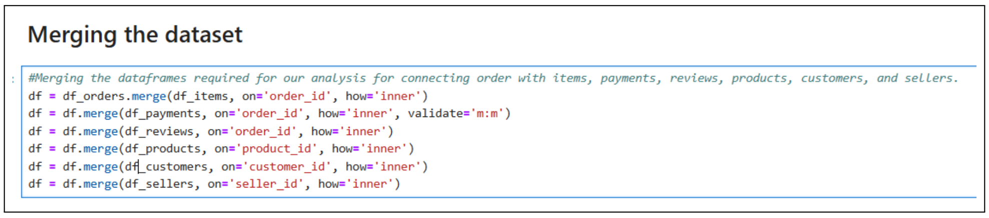
Figure 10: Accessing the datasets

6.4 Merging the Datasets

- **Step 1:** The ‘`Data Schema`’ image shown in the code is added manually. The image was first saved in the local directory named as `olist_database_merge.png`, shown in Figure 7 previously. The cell has to be made Markdown Cell and then need to use this command for the image -

  ```
  ![image.png](olist_database_merge.png)
  ```

- **Step 2:** All the datasets has to be merged into single dataset which is required for research using the following python commands using *’inner join’* as shown in Figure 11.



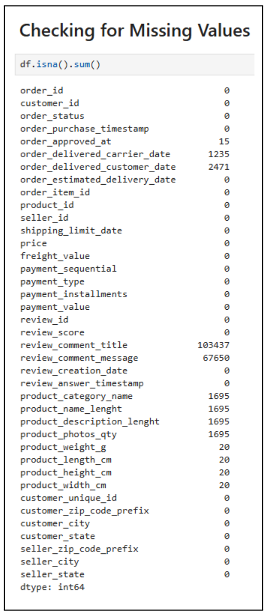
Merging the dataset

```
#Merging the dataframes required for our analysis for connecting order with items, payments, reviews, products, customers, and sellers.
df = df_orders.merge(df_items, on='order_id', how='inner')
df = df.merge(df_payments, on='order_id', how='inner', validate='m:m')
df = df.merge(df_reviews, on='order_id', how='inner')
df = df.merge(df_products, on='product_id', how='inner')
df = df.merge(df_customers, on='customer_id', how='inner')
df = df.merge(df_sellers, on='seller_id', how='inner')
```

Figure 11: Command for Merging the Datasets

6.5 Checking Missing Values and Handling Them

- **Step 1:** The missing values were checked for the merged dataset using `’sum’` function. The Figure 12 shows all the missing values.



Checking for Missing Values

```
df.isna().sum()
```

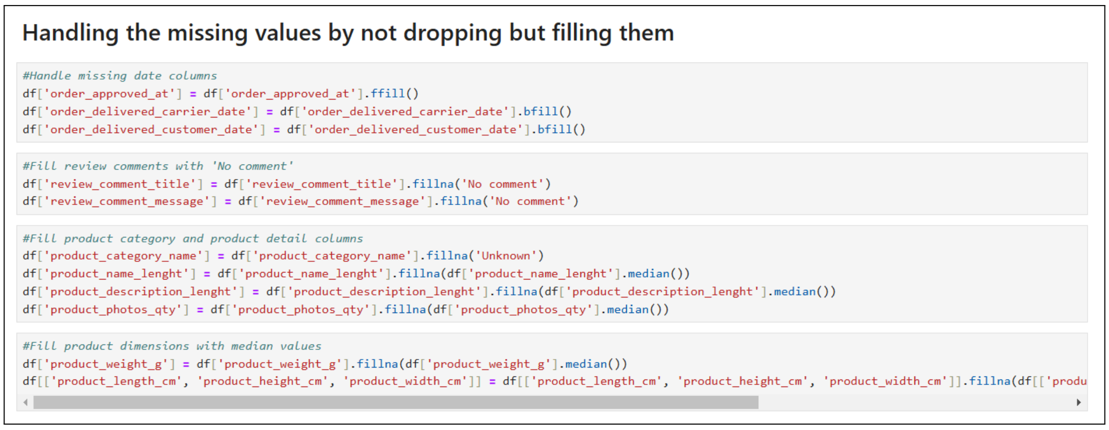
```
order_id                              0
customer_id                           0
order_status                          0
order_purchase_timestamp              0
order_approved_at                    15
order_delivered_carrier_date       1235
order_delivered_customer_date      2471
order_estimated_delivery_date         0
order_item_id                         0
product_id                            0
seller_id                             0
shipping_limit_date                   0
price                                 0
freight_value                         0
payment_sequential                    0
payment_type                          0
payment_installments                  0
payment_value                         0
review_id                             0
review_score                          0
review_comment_title             103437
review_comment_message            67650
review_creation_date                  0
review_answer_timestamp               0
product_category_name              1695
product_name_lenght                1695
product_description_lenght         1695
product_photos_qty                 1695
product_weight_g                     20
product_length_cm                    20
product_height_cm                    20
product_width_cm                     20
customer_unique_id                    0
customer_zip_code_prefix              0
customer_city                         0
customer_state                        0
seller_zip_code_prefix                0
seller_city                           0
seller_state                          0
dtype: int64
```

Figure 12: Checking missing values

- **Step 2:** The missing values were found and handled using various methods like
  - Forward fill

- Backward fill
- Filling values with specific text
- Using the median method
- Other context-specific techniques

The python commands used can be seen in Figure 13.



Handling the missing values by not dropping but filling them

```
#Handle missing date columns
df['order_approved_at'] = df['order_approved_at'].ffill()
df['order_delivered_carrier_date'] = df['order_delivered_carrier_date'].bfill()
df['order_delivered_customer_date'] = df['order_delivered_customer_date'].bfill()
```

```
#Fill review comments with 'No comment'
df['review_comment_title'] = df['review_comment_title'].fillna('No comment')
df['review_comment_message'] = df['review_comment_message'].fillna('No comment')
```

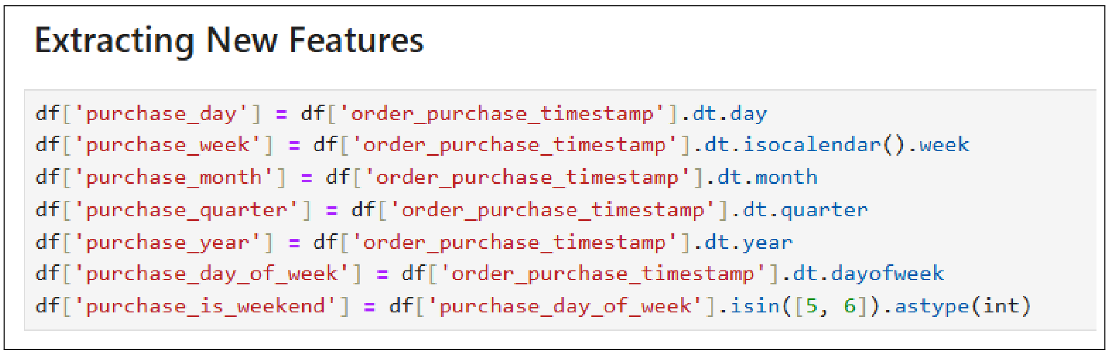
```
#Fill product category and product detail columns
df['product_category_name'] = df['product_category_name'].fillna('Unknown')
df['product_name_lenght'] = df['product_name_lenght'].fillna(df['product_name_lenght'].median())
df['product_description_lenght'] = df['product_description_lenght'].fillna(df['product_description_lenght'].median())
df['product_photos_qty'] = df['product_photos_qty'].fillna(df['product_photos_qty'].median())
```

```
#Fill product dimensions with median values
df['product_weight_g'] = df['product_weight_g'].fillna(df['product_weight_g'].median())
df[['product_length_cm', 'product_height_cm', 'product_width_cm']] = df[['product_length_cm', 'product_height_cm', 'product_width_cm']].fillna(df[['produ
```

Figure 13: Handling missing values

6.6 Feature Engineering and Data Normalization

- **Step 1:** Feature Engineering was carried out and all the new features where extracted that are required for the research work. The Figure 14 shows some of the examples of features extracte for the research work below.

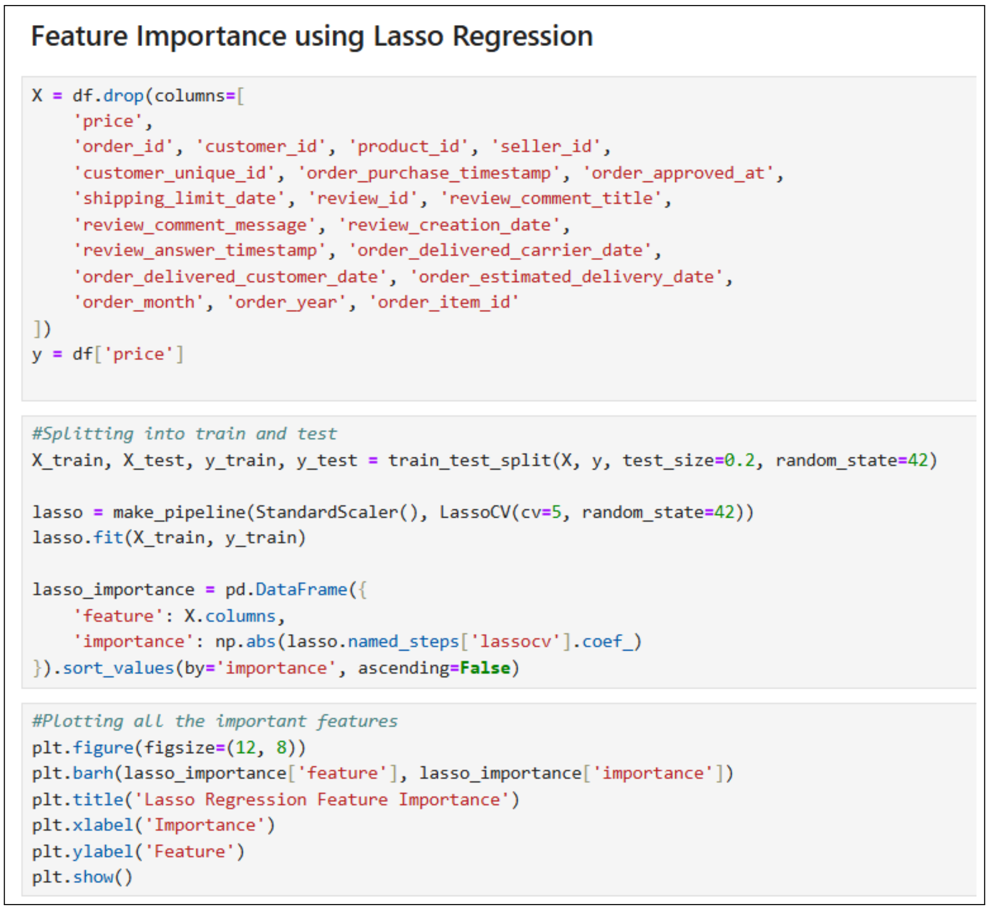


Extracting New Features

```
df['purchase_day'] = df['order_purchase_timestamp'].dt.day
df['purchase_week'] = df['order_purchase_timestamp'].dt.isocalendar().week
df['purchase_month'] = df['order_purchase_timestamp'].dt.month
df['purchase_quarter'] = df['order_purchase_timestamp'].dt.quarter
df['purchase_year'] = df['order_purchase_timestamp'].dt.year
df['purchase_day_of_week'] = df['order_purchase_timestamp'].dt.dayofweek
df['purchase_is_weekend'] = df['purchase_day_of_week'].isin([5, 6]).astype(int)
```

Figure 14: Examples of some new features

- **Step 2:** After this, all the features were encoded using 'Label Encoder'. The selection of Top 15 relevant features is done using `'Lasso Regression'` by creating a pipeline as shown in Figure 15.



Feature Importance using Lasso Regression

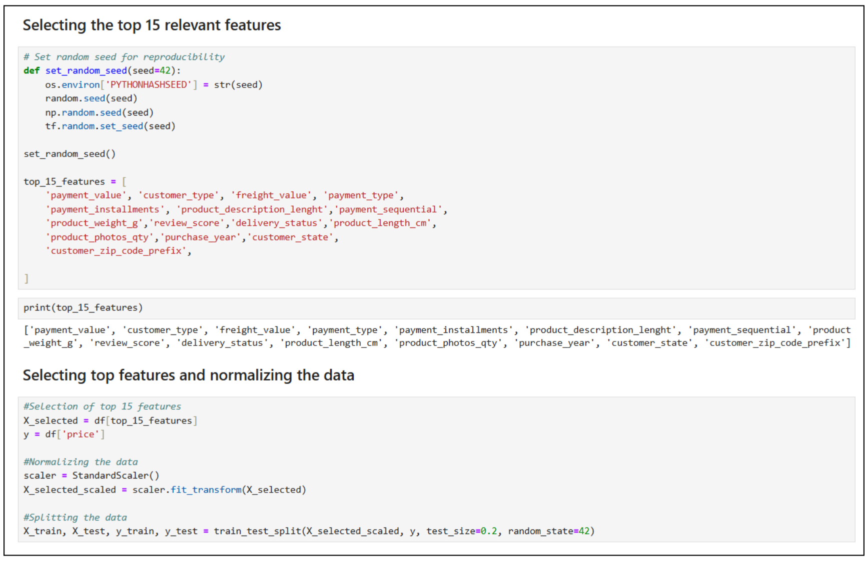
```
X = df.drop(columns=[
    'price',
    'order_id', 'customer_id', 'product_id', 'seller_id',
    'customer_unique_id', 'order_purchase_timestamp', 'order_approved_at',
    'shipping_limit_date', 'review_id', 'review_comment_title',
    'review_comment_message', 'review_creation_date',
    'review_answer_timestamp', 'order_delivered_carrier_date',
    'order_delivered_customer_date', 'order_estimated_delivery_date',
    'order_month', 'order_year', 'order_item_id'
])
y = df['price']
```

```
#Splitting into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

lasso = make_pipeline(StandardScaler(), LassoCV(cv=5, random_state=42))
lasso.fit(X_train, y_train)

lasso_importance = pd.DataFrame({
    'feature': X.columns,
    'importance': np.abs(lasso.named_steps['lassocv'].coef_)
}).sort_values(by='importance', ascending=False)
```

```
#Plotting all the important features
plt.figure(figsize=(12, 8))
plt.barh(lasso_importance['feature'], lasso_importance['importance'])
plt.title('Lasso Regression Feature Importance')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

Figure 15: Feature Importance using Lasso Regression

- **Step 3:** This step involves in selection of Top 15 relevant features and combining them into a single dataframe. After selection of relevant features, the data was then Normalized using `'StandardScaler' technique`. Later, this data is split into training and testing datasets which indicates the dropping of the target variable as shown in Figure 16.



Selecting the top 15 relevant features

```
# Set random seed for reproducibility
def set_random_seed(seed=42):
    os.environ['PYTHONHASHSEED'] = str(seed)
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)

set_random_seed()

top_15_features = [
    'payment_value', 'customer_type', 'freight_value', 'payment_type',
    'payment_installments', 'product_description_lenght','payment_sequential',
    'product_weight_g','review_score','delivery_status','product_length_cm',
    'product_photos_qty','purchase_year','customer_state',
    'customer_zip_code_prefix',

]
```

```
print(top_15_features)
```

```
['payment_value', 'customer_type', 'freight_value', 'payment_type', 'payment_installments', 'product_description_lenght', 'payment_sequential', 'product
_weight_g', 'review_score', 'delivery_status', 'product_length_cm', 'product_photos_qty', 'purchase_year', 'customer_state', 'customer_zip_code_prefix']
```

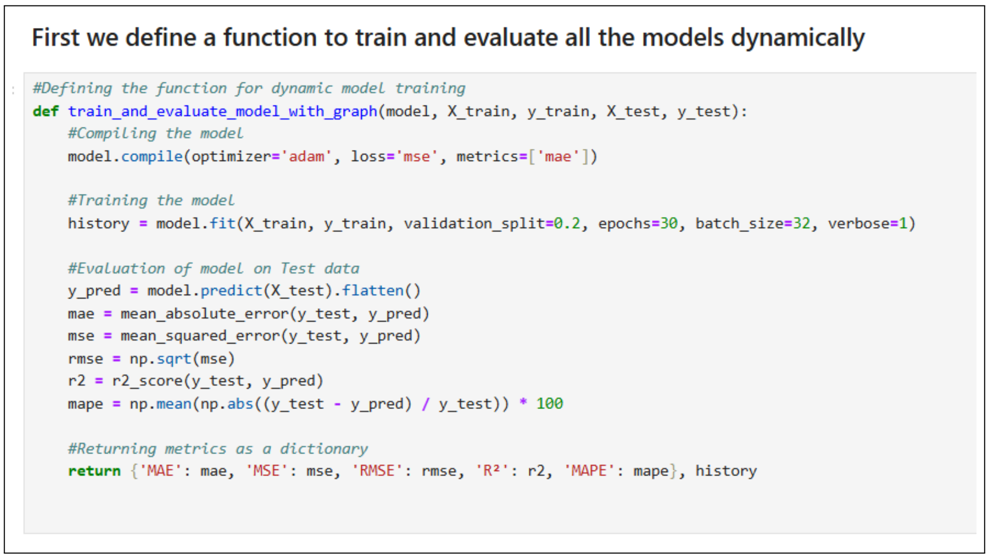
Selecting top features and normalizing the data

```
#Selection of top 15 features
X_selected = df[top_15_features]
y = df['price']

#Normalizing the data
scaler = StandardScaler()
X_selected_scaled = scaler.fit_transform(X_selected)

#Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X_selected_scaled, y, test_size=0.2, random_state=42)
```

Figure 16: Selection of top features and normalizing the data

6.7 Model Building and Training

- **Step 1:** This research focuses on running and building the models dynamically for avoiding the duplication of code again and again. Figure 17 shows the python command for creation of a 'pre-defined function' to run all the three models dynamically.

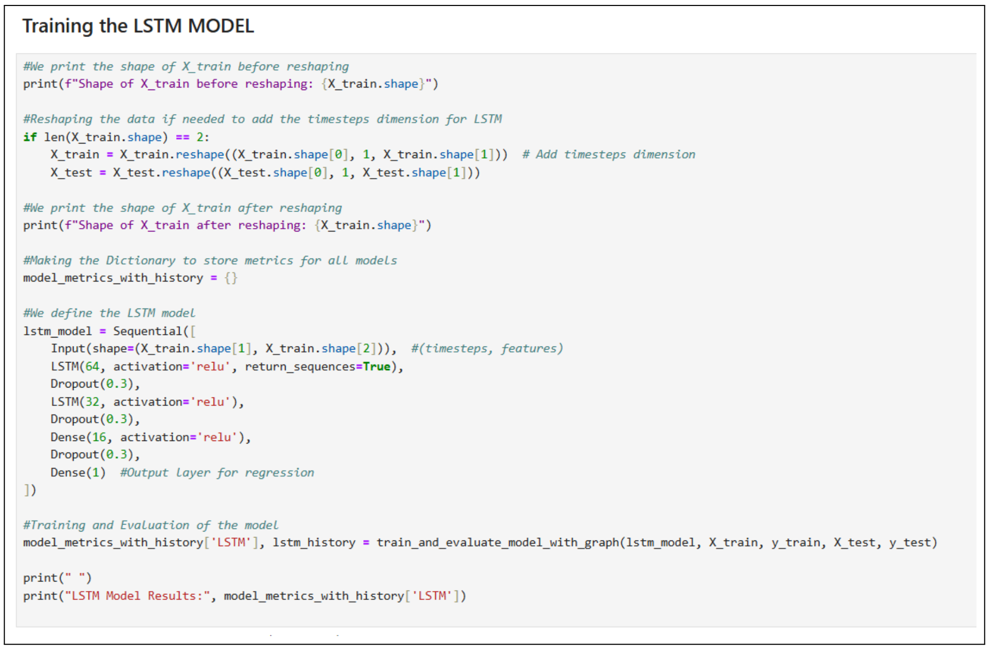


First we define a function to train and evaluate all the models dynamically

```python
#Defining the function for dynamic model training
def train_and_evaluate_model_with_graph(model, X_train, y_train, X_test, y_test):
    #Compiling the model
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])

    #Training the model
    history = model.fit(X_train, y_train, validation_split=0.2, epochs=30, batch_size=32, verbose=1)

    #Evaluation of model on Test data
    y_pred = model.predict(X_test).flatten()
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)
    mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

    #Returning metrics as a dictionary
    return {'MAE': mae, 'MSE': mse, 'RMSE': rmse, 'R²': r2, 'MAPE': mape}, history
```

Figure 17: Creating a function for building the models dynamically

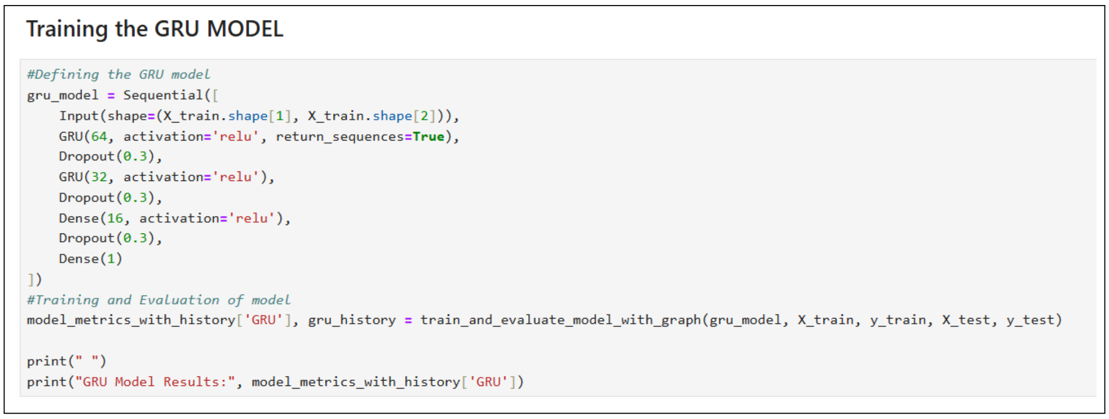
- **Step 2:** The research work has taken three models into consideration. The advance deep learning models implemented are – `LSTM (Long Short-Term Memory) model`, `GRU (Gated Recurrent Unit) model` (Shiri et al.; 2023) and a third model as a `Hybrid model` that is the `combination of (LSTM + GRU)` together. Figures 18, 19, and 20 shows all the python commands used for training and building model respectively below.



Training the LSTM MODEL

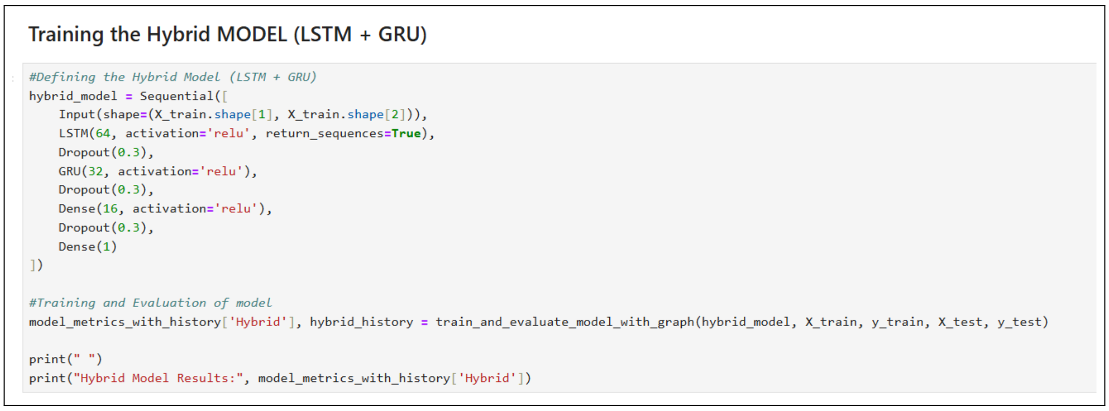
```python
#We print the shape of X_train before reshaping
print(f"Shape of X_train before reshaping: {X_train.shape}")

#Reshaping the data if needed to add the timesteps dimension for LSTM
if len(X_train.shape) == 2:
    X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))  # Add timesteps dimension
    X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

#We print the shape of X_train after reshaping
print(f"Shape of X_train after reshaping: {X_train.shape}")

#Making the Dictionary to store metrics for all models
model_metrics_with_history = {}

#We define the LSTM model
lstm_model = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),  #(timesteps, features)
    LSTM(64, activation='relu', return_sequences=True),
    Dropout(0.3),
    LSTM(32, activation='relu'),
    Dropout(0.3),
    Dense(16, activation='relu'),
    Dropout(0.3),
    Dense(1)  #Output layer for regression
])

#Training and Evaluation of the model
model_metrics_with_history['LSTM'], lstm_history = train_and_evaluate_model_with_graph(lstm_model, X_train, y_train, X_test, y_test)

print(" ")
print("LSTM Model Results:", model_metrics_with_history['LSTM'])
```

Figure 18: Training the LSTM model


Training the GRU MODEL

```
#Defining the GRU model
gru_model = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),
    GRU(64, activation='relu', return_sequences=True),
    Dropout(0.3),
    GRU(32, activation='relu'),
    Dropout(0.3),
    Dense(16, activation='relu'),
    Dropout(0.3),
    Dense(1)
])
#Training and Evaluation of model
model_metrics_with_history['GRU'], gru_history = train_and_evaluate_model_with_graph(gru_model, X_train, y_train, X_test, y_test)

print(" ")
print("GRU Model Results:", model_metrics_with_history['GRU'])
```

Figure 19: Training the GRU model

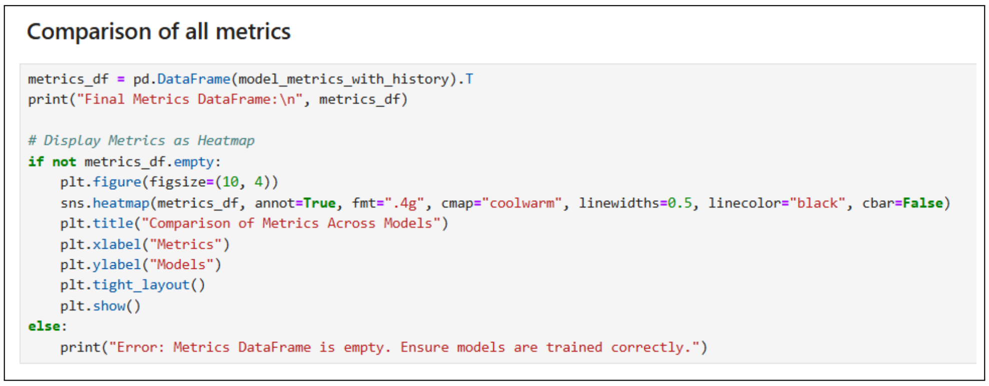

Training the Hybrid MODEL (LSTM + GRU)

```
#Defining the Hybrid Model (LSTM + GRU)
hybrid_model = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2])),
    LSTM(64, activation='relu', return_sequences=True),
    Dropout(0.3),
    GRU(32, activation='relu'),
    Dropout(0.3),
    Dense(16, activation='relu'),
    Dropout(0.3),
    Dense(1)
])

#Training and Evaluation of model
model_metrics_with_history['Hybrid'], hybrid_history = train_and_evaluate_model_with_graph(hybrid_model, X_train, y_train, X_test, y_test)

print(" ")
print("Hybrid Model Results:", model_metrics_with_history['Hybrid'])
```

Figure 20: Training the Hybrid model (LSTM + GRU)

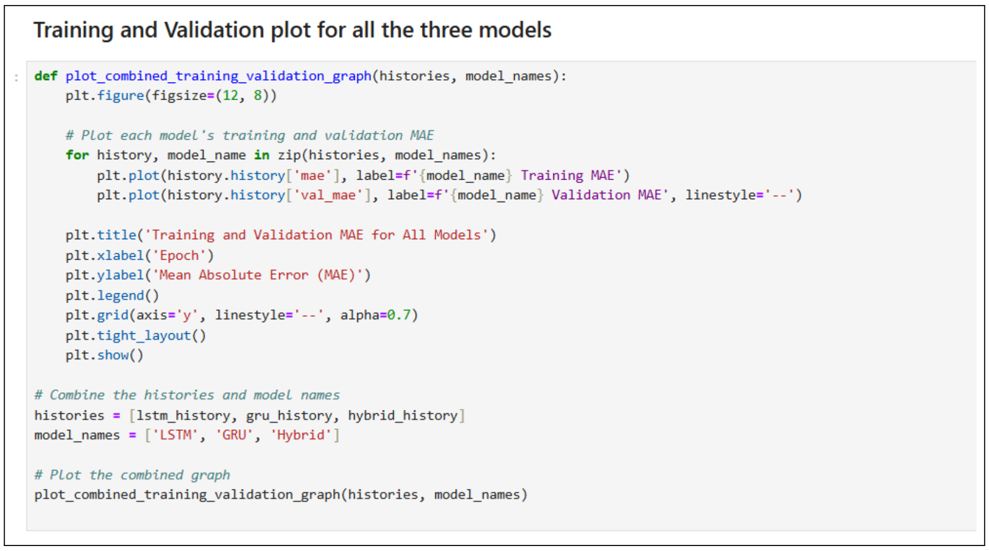
7 Model Evaluation

All the three models were evaluated. The models were evaluated based on **MAE** (Mean Absolute Error), **MSE** (Mean Squared Error), **RMSE** (Root Mean Squared Error), **R²** (Coefficient of Determination) (Chicco et al.; 2021), and **MAPE** (Mean Absolute Percentage Error) Botchkarev (2018). In addition, the comparison plots of all values of evaluation metrics on all the three models were shown by different graphs. Figures 21 and 22 depicts, all the Python commands required for doing the evaluation of all models.


Comparison of all metrics

```
metrics_df = pd.DataFrame(model_metrics_with_history).T
print("Final Metrics DataFrame:\n", metrics_df)

# Display Metrics as Heatmap
if not metrics_df.empty:
    plt.figure(figsize=(10, 4))
    sns.heatmap(metrics_df, annot=True, fmt=".4g", cmap="coolwarm", linewidths=0.5, linecolor="black", cbar=False)
    plt.title("Comparison of Metrics Across Models")
    plt.xlabel("Metrics")
    plt.ylabel("Models")
    plt.tight_layout()
    plt.show()
else:
    print("Error: Metrics DataFrame is empty. Ensure models are trained correctly.")
```

Figure 21: Comparison of all metrics


Training and Validation plot for all the three models

```
def plot_combined_training_validation_graph(histories, model_names):
    plt.figure(figsize=(12, 8))

    # Plot each model's training and validation MAE
    for history, model_name in zip(histories, model_names):
        plt.plot(history.history['mae'], label=f'{model_name} Training MAE')
        plt.plot(history.history['val_mae'], label=f'{model_name} Validation MAE', linestyle='--')

    plt.title('Training and Validation MAE for All Models')
    plt.xlabel('Epoch')
    plt.ylabel('Mean Absolute Error (MAE)')
    plt.legend()
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()

# Combine the histories and model names
histories = [lstm_history, gru_history, hybrid_history]
model_names = ['LSTM', 'GRU', 'Hybrid']

# Plot the combined graph
plot_combined_training_validation_graph(histories, model_names)
```

Figure 22: Training and Validation for all the three models

References

Botchkarev, A. (2018). Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology, *arXiv preprint arXiv:1809.03006* .

Chicco, D., Warrens, M. J. and Jurman, G. (2021). The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation, *Peerj computer science* **7**: e623.

Shiri, F. M., Perumal, T., Mustapha, N. and Mohamed, R. (2023). A comprehensive overview and comparative analysis on deep learning models: Cnn, rnn, lstm, gru, *arXiv preprint arXiv:2305.17473* .