Configuration Manual

MSc Research Project
Data Analytics

Avni Rathi
Student ID: x22182918

School of Computing
National College of Ireland

Supervisor: Naushad Alam

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Avni Rathi		
Student ID:	x22182918		
Programme:	MSc. in Data Analytics	**Year:**	2023-2024
Module:	MSc Research Project		
Lecturer:	Naushad Alam		
Submission Due Date:	12/08/2024		
Project Title:	Optimising Supply Chain Performance with Machine Learning for Predicting Late Deliveries		
Word Count:	1540	**Page Count:**	7

Signature: Avni Rathi

Date: 12/08/2024

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.	□

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

Configuration Manual

Avni Rathi
Student ID: x22182918

1 Introduction

This Configuration Manual describes the steps for setting up, configuring, and running the project. The purpose is to ensure a smooth process creating the project environment and proceeding with development stages. The manual includes all the details of project file, system specifications, software used, and installation steps. It begins by explaining the project files details, followed by system specifications with hardware and software requirements. The software section details the tools and libraries used, along with downloading and installation processes suggested. Finally, the manual will present key development tasks such as importing libraries, loading data, data processing, and creating machine learning models to provide a clear step-by-step configuration guide.

2 Project Files Details

The purpose of the project is to employ a machine learning to predict supply chain outcomes, primarily focusing on late deliveries and sales forecasting. The core files necessary for the completion of the project are "code.ipynb" being the primary one. This jupyter notebook file includes the full code step by step for data analysis, model training, and evaluation. The second file is "DataCoSupplyChainDataset.csv" which represents the dataset in the form of a CSV file holding historical records of supply chain flows and sales. Therefore, both these files are essential for the project as it enable the development and application of machine learning approaches to supply chain issues.

3 System Specification

In order to have an understanding of hardware used for this research project is an Apple MacBook Air laptop with MacOS operating system and 8GB RAM as shown in Table 1.

Model Name:	MacBook Air
Model Identifier:	M1, 2020
Chip:	Apple M1
Memory:	8 GB
MacOS	Sonoma 14.5

Table 1: Hardware Specification

4 Software Used

4.1 Programming Language and Environment

This project used Python as the main programming language. The python version which is used in this project is Python 3.9.19 within the Anaconda Jupyter Notebook environment and is preferred to use this version or newer because it ensures the highest level of compatibility with modern libraries and tools. In this project, the anaconda version 23.5.2 and jupyter notebook version is 6.4.12 is used which gives a versatile platform that support flawless performance of Python programs. Also, the recommended setup make sure that the code runs without any errors.

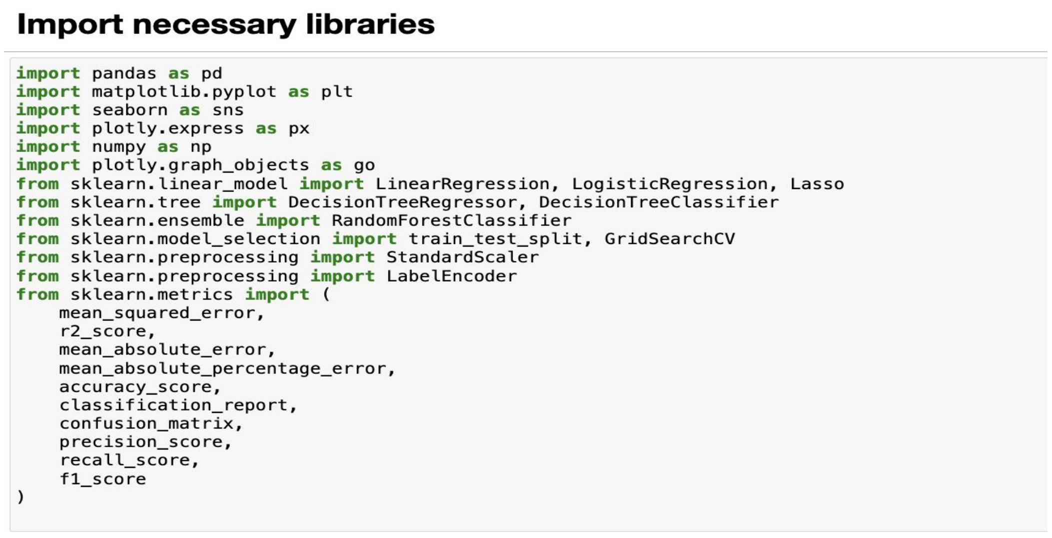
4.2 Programming Libraries and Frameworks

The project encompasses various key libraries and frameworks that are integral for data manipulation and visualization, as well as machine learning. More specifically, pandas are used for data manipulation and analysis, followed by numpy, which supports numerical operations. Matplotlib and Seaborn are adopted for data visualization and statistical analysis purposes, respectively. In terms of machine learning algorithms for modeling and evaluation, scikit-learn is used.

Pip is a package manager used for facilitating the installation and management of the required Python libraries and their dependencies for a project. It is the built-in tool in Python that is fit for dealing with Python packages. It enables the users to install libraries such as pandas, numpy, matplotlib, seaborn, and scikit-learn. Installing of packages and libararies is very easy in juypter notebook. One of the examples for installing a package using pip is running pip install package_name in the terminal or command prompt.

5 Project Development

5.1 Importing Library



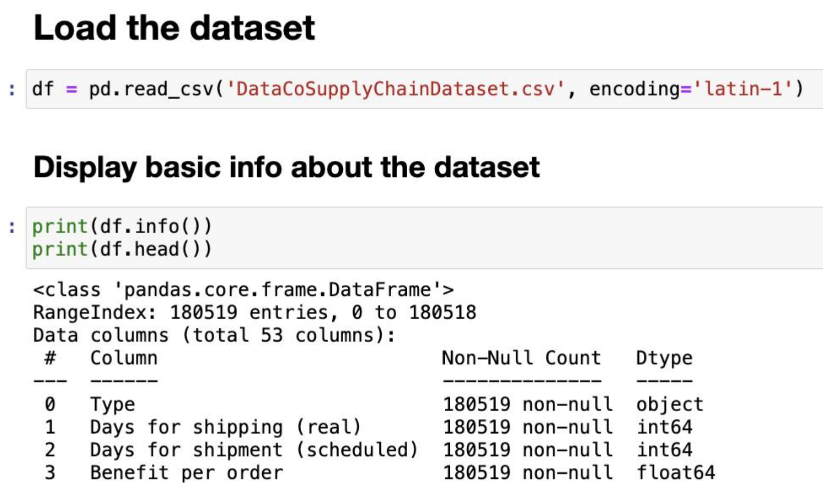
Import necessary libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import numpy as np
import plotly.graph_objects as go
from sklearn.linear_model import LinearRegression, LogisticRegression, Lasso
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import (
    mean_squared_error,
    r2_score,
    mean_absolute_error,
    mean_absolute_percentage_error,
    accuracy_score,
    classification_report,
    confusion_matrix,
    precision_score,
    recall_score,
    f1_score
)
```

Figure 1: Import Libraries

In the project, various Python libraries are imported to facilitate data manipulation, visualization, and machine-learning model implementation. The Panda's library is utilized for efficient data handling and manipulation, while matplotlib and Seaborn are employed for creating insightful visualizations. For advanced visualization, plotly.graph_objects is included. For machine learning, the scikit-learn library provides a range of algorithms and tools. train_test_split is used to divide the dataset into training and testing sets. Classification models are implemented using RandomForestClassifier, LogisticRegression, and DecisionTreeClassifier. DecisionTreeRegressor is used for regression tasks. The library also includes various metrics to evaluate model performance, such as accuracy_score, confusion_matrix, precision_score, recall_score, f1_score, mean_absolute_error, mean_squared_error, and r2_score. LabelEncoder is employed for encoding categorical variables into numerical values.

5.2 Importing Files



Load the dataset

```
df = pd.read_csv('DataCoSupplyChainDataset.csv', encoding='latin-1')
```

Display basic info about the dataset

```
print(df.info())
print(df.head())
```

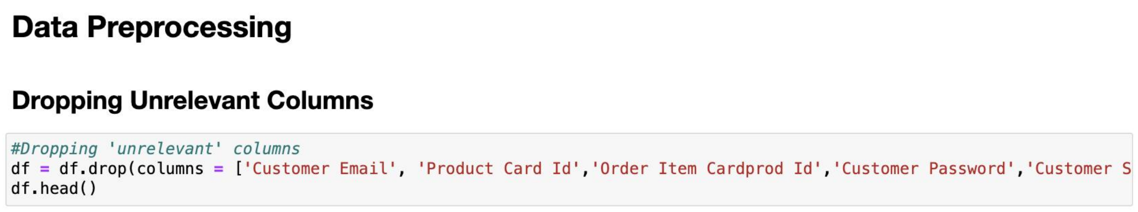
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180519 entries, 0 to 180518
Data columns (total 53 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   Type                           180519 non-null  object
 1   Days for shipping (real)       180519 non-null  int64
 2   Days for shipment (scheduled)  180519 non-null  int64
 3   Benefit per order              180519 non-null  float64
```

Figure 2: Loading the Dataset

In order to import the dataset in jupyter notebook shown in Figure 2, first ensure that the dataset file, DataCoSupplyChainDataset.csv, is uploaded to the jupyter notebook or located in the appropriate directory. Begin by loading the dataset using the pandas library, which is used for data manipulation and analysis. The dataset can be accessed directly from the nootbook if uploaded, or from a specific directory path if placed in a folder. Ensure that the file path is correctly specified based on its location to successfully load the dataset.

5.3 Data Cleaning and Processing

The data cleaning code can be found under the Data Cleaning step in the code.ipynb file. For data quality, the set of data is cleared out of all missing values and duplicates by using df.isna().sum() which shows the missing values and df.drop_duplicates(inplace=True) which removes all repetitive rows respectively. To handle missing values, some empty columns were removed and the columns that had few number of missing values were also removed. In order to ensure that the new actions happen, the data set is examined. Thus, the command df.info() is run to remember the data and be aware of its characteristics and the types of data in each entrance. Also, to visualize the first few data points, the function df. head() is run.



Data Preprocessing

Dropping Unrelevant Columns

```
#Dropping 'unrelevant' columns
df = df.drop(columns = ['Customer Email', 'Product Card Id','Order Item Cardprod Id','Customer Password','Customer S
df.head()
```

Figure 3: Dropping Irrelevant Columns

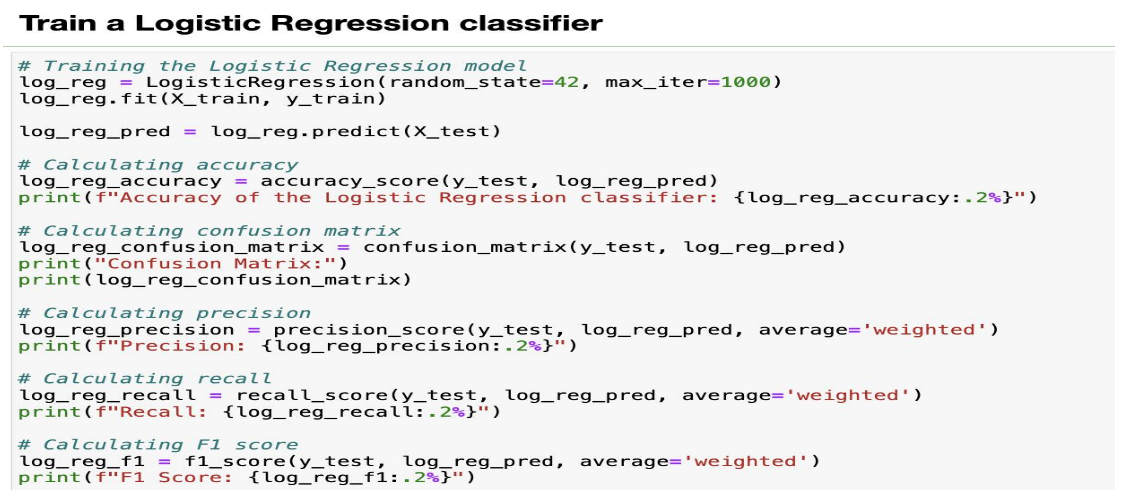
The code for feature engineering can be found under the Feature Engineering section in the code.ipnyb file. The dataset is preprocessed to prepare it for modeling. First, the relevant columns are selected and stored in late_df. Categorical variables are converted into dummy variables using pd.get_dummies(), with the specified nominal columns encoded and the first category dropped to avoid multicollinearity. Ordinal columns, if any, would be encoded using LabelEncoder, though none are included in this example. The features (X) and target variable (y) are then separated. The dataset is split into training and testing sets using train_test_split(), with 70% of the data allocated to training and 30% to testing, ensuring a random seed for reproducibility.

Multiple columns have been added based on the date columns to improve the data's predictive performance further. The order_year, order_month, order_day, shipping_year, and shipping_month, as well as the shipping_day, are taken directly from the order date and shipping date. These columns are introduced by temporal patterns and the way seasonality may allow people to predict whether orders placed at certain times of the year or days of the week will be late. These columns are removed as an irrelevant column and presents a much cleaner dataset that is designed to predict delivery time ranges.

5.4 Modelling

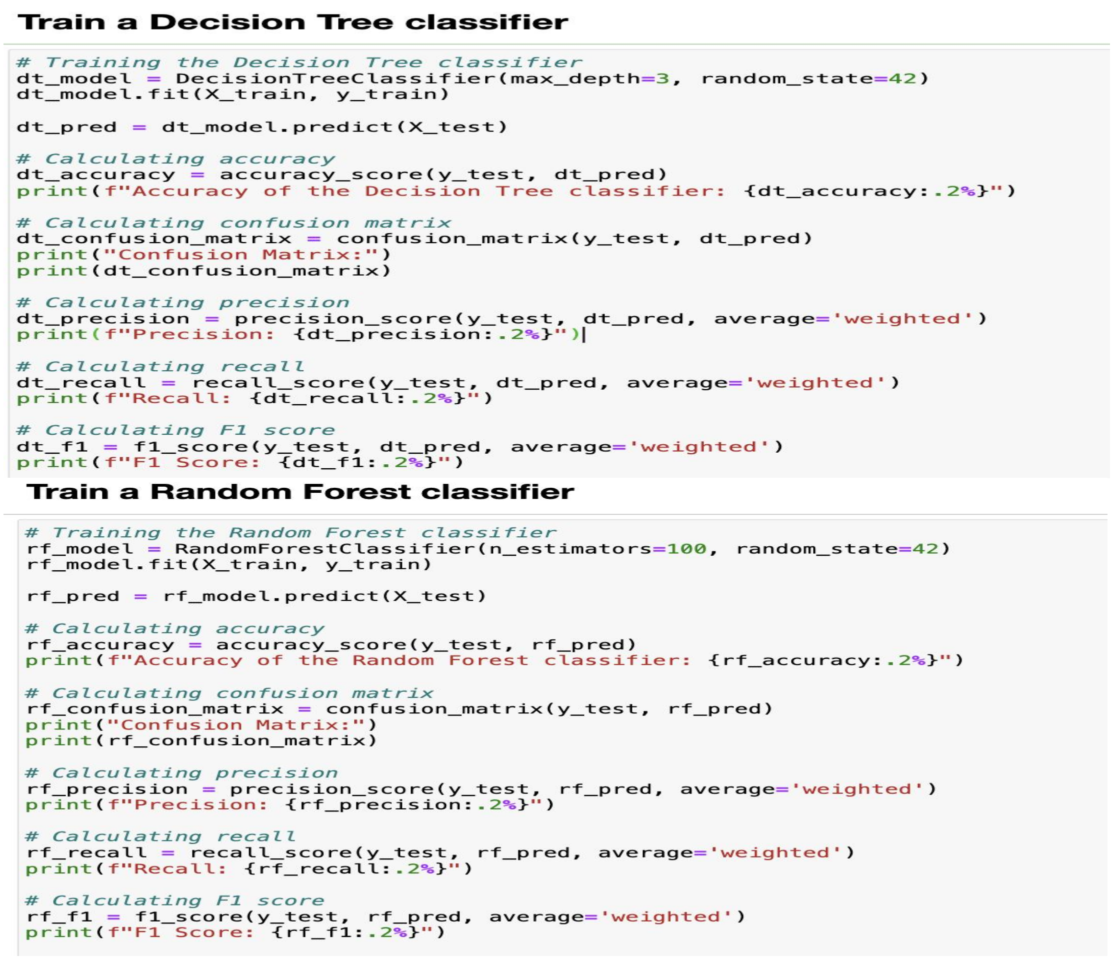
5.4.1 Initial Model building for Research Question 1

After data preprocessing, the model building step was applied and three classification model were applied to predict risk of late delivery in this research paper.



Train a Logistic Regression classifier

```
# Training the Logistic Regression model
log_reg = LogisticRegression(random_state=42, max_iter=1000)
log_reg.fit(X_train, y_train)

log_reg_pred = log_reg.predict(X_test)

# Calculating accuracy
log_reg_accuracy = accuracy_score(y_test, log_reg_pred)
print(f"Accuracy of the Logistic Regression classifier: {log_reg_accuracy:.2%}")

# Calculating confusion matrix
log_reg_confusion_matrix = confusion_matrix(y_test, log_reg_pred)
print("Confusion Matrix:")
print(log_reg_confusion_matrix)

# Calculating precision
log_reg_precision = precision_score(y_test, log_reg_pred, average='weighted')
print(f"Precision: {log_reg_precision:.2%}")

# Calculating recall
log_reg_recall = recall_score(y_test, log_reg_pred, average='weighted')
print(f"Recall: {log_reg_recall:.2%}")

# Calculating F1 score
log_reg_f1 = f1_score(y_test, log_reg_pred, average='weighted')
print(f"F1 Score: {log_reg_f1:.2%}")
```



Train a Decision Tree classifier

```
# Training the Decision Tree classifier
dt_model = DecisionTreeClassifier(max_depth=3, random_state=42)
dt_model.fit(X_train, y_train)

dt_pred = dt_model.predict(X_test)

# Calculating accuracy
dt_accuracy = accuracy_score(y_test, dt_pred)
print(f"Accuracy of the Decision Tree classifier: {dt_accuracy:.2%}")

# Calculating confusion matrix
dt_confusion_matrix = confusion_matrix(y_test, dt_pred)
print("Confusion Matrix:")
print(dt_confusion_matrix)

# Calculating precision
dt_precision = precision_score(y_test, dt_pred, average='weighted')
print(f"Precision: {dt_precision:.2%}")

# Calculating recall
dt_recall = recall_score(y_test, dt_pred, average='weighted')
print(f"Recall: {dt_recall:.2%}")

# Calculating F1 score
dt_f1 = f1_score(y_test, dt_pred, average='weighted')
print(f"F1 Score: {dt_f1:.2%}")
```

Train a Random Forest classifier

```
# Training the Random Forest classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

rf_pred = rf_model.predict(X_test)

# Calculating accuracy
rf_accuracy = accuracy_score(y_test, rf_pred)
print(f"Accuracy of the Random Forest classifier: {rf_accuracy:.2%}")

# Calculating confusion matrix
rf_confusion_matrix = confusion_matrix(y_test, rf_pred)
print("Confusion Matrix:")
print(rf_confusion_matrix)

# Calculating precision
rf_precision = precision_score(y_test, rf_pred, average='weighted')
print(f"Precision: {rf_precision:.2%}")

# Calculating recall
rf_recall = recall_score(y_test, rf_pred, average='weighted')
print(f"Recall: {rf_recall:.2%}")

# Calculating F1 score
rf_f1 = f1_score(y_test, rf_pred, average='weighted')
print(f"F1 Score: {rf_f1:.2%}")
```

Figure 4: Applying Random Forest, Decision Tree and Logistic Regression

5.4.2 Model with Hyperparameter Tuning

The hyperparameter tuning is performed for every machine learning model using GridSearchCV. It can be found in the code.ipynb file after every initial model implementation. The specific parameter is set for each model, and cross-validation is performed to find the best combination that works. The Decision Tree's requirements for leaf nodes, split criteria, and depth are all optimized. The processes of regularization and solver selection are the main areas of the logistic regression model. After the hyperparameter tuning, every model's performance improved.

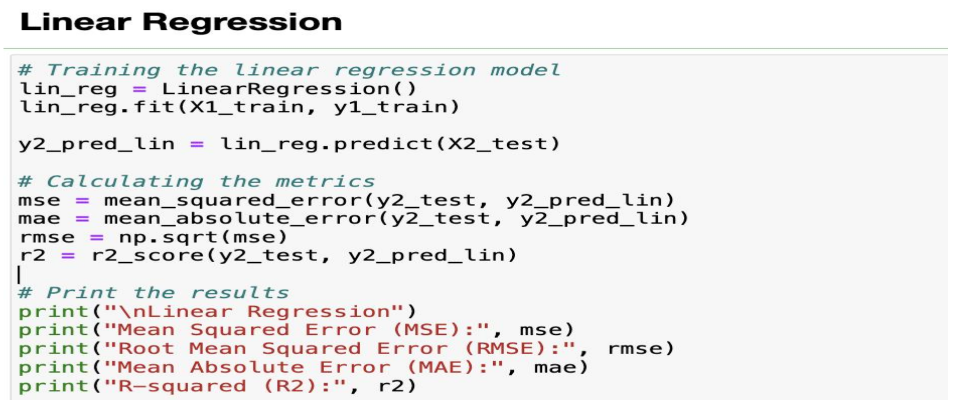
After applying feature importance analysis, the results showed that features have different impacts dependent on the classifier. The most impactful feature in the random forest classifier is “Days for shipping (real)”. Other influential features include “Days for shipment (scheduled)” and “Shipping Mode_Standard Class” recorded at 0.157 and 0.114, respectively. Other significant features are “Shipping Mode_Second Class” and “Benefit per order.” Features having a major impact on the decision tree classifier are “Days for shipping (real)”, and “Days for shipment (scheduled)” In the logistic regression classifier, the most significant feature is “Days for shipment (scheduled)”. Days for shipping (real)” feature is recorded as positive. The rest of the influential features in the logistic regression classifier are “Shipping Mode_Second Class”, and “Shipping Mode_Same Day”.

5.4.3 Feature Importance

This section is present in the code.ipynb file after applying each model in the code.It has been applied individually after every model. The results showed that features have different impacts dependent on the classifier. The most impactful feature in the random forest classifier is “Days for shipping (real)”. Other influential features include “Days for shipment (scheduled)” and “Shipping Mode_Standard Class” recorded at 0.157 and 0.114, respectively. Other significant features are “Shipping Mode_Second Class” and “Benefit per order.” Features having a major impact on the decision tree classifier are “Days for shipping (real)”, and “Days for shipment (scheduled)” In the logistic regression classifier, the most significant feature is “Days for shipment (scheduled)”. Days for shipping (real)” feature is recorded as positive. The rest of the influential features in the logistic regression classifier are “Shipping Mode_Second Class”, and “Shipping Mode_Same Day”.

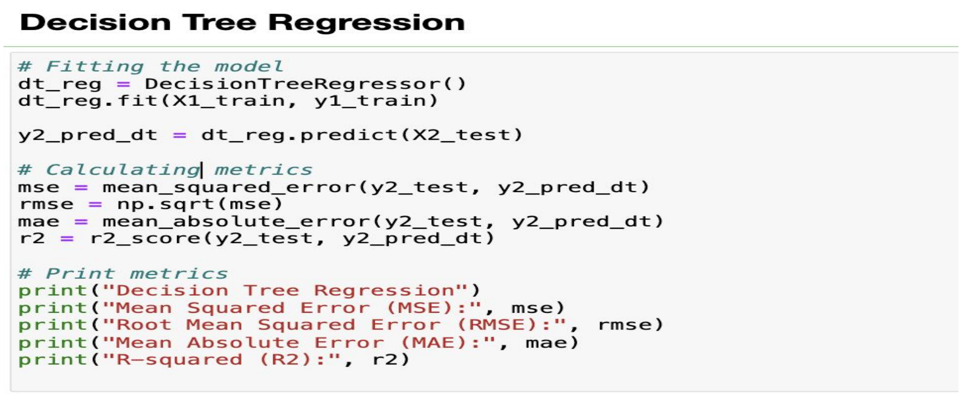
5.4.4 Model building for Research Question 2

To predict the sales, multiple regression models were employed in this research study. The linear regression, lasso regression, and decision tree regression models were applied, and after the model, feature importance analysis was performed on each model to identify the most influential features from the dataset.



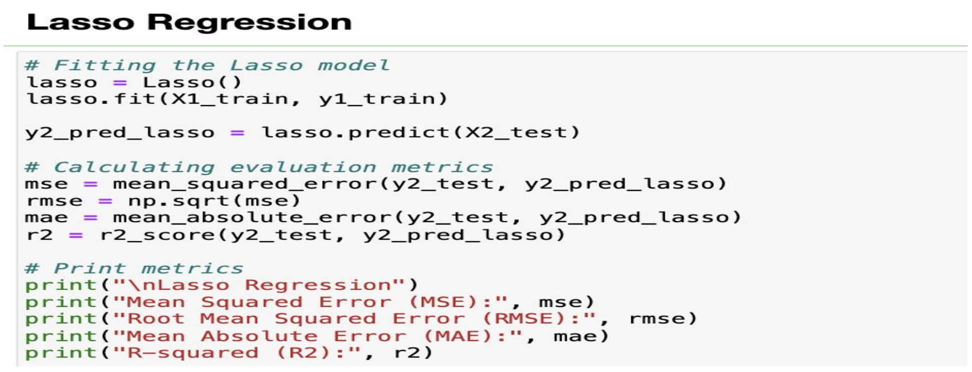
Linear Regression

```
# Training the linear regression model
lin_reg = LinearRegression()
lin_reg.fit(X1_train, y1_train)

y2_pred_lin = lin_reg.predict(X2_test)

# Calculating the metrics
mse = mean_squared_error(y2_test, y2_pred_lin)
mae = mean_absolute_error(y2_test, y2_pred_lin)
rmse = np.sqrt(mse)
r2 = r2_score(y2_test, y2_pred_lin)

# Print the results
print("\nLinear Regression")
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R2):", r2)
```



Decision Tree Regression

```
# Fitting the model
dt_reg = DecisionTreeRegressor()
dt_reg.fit(X1_train, y1_train)

y2_pred_dt = dt_reg.predict(X2_test)

# Calculating metrics
mse = mean_squared_error(y2_test, y2_pred_dt)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y2_test, y2_pred_dt)
r2 = r2_score(y2_test, y2_pred_dt)

# Print metrics
print("Decision Tree Regression")
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R2):", r2)
```


Lasso Regression

```
# Fitting the Lasso model
lasso = Lasso()
lasso.fit(X1_train, y1_train)

y2_pred_lasso = lasso.predict(X2_test)

# Calculating evaluation metrics
mse = mean_squared_error(y2_test, y2_pred_lasso)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y2_test, y2_pred_lasso)
r2 = r2_score(y2_test, y2_pred_lasso)

# Print metrics
print("\nLasso Regression")
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R2):", r2)
```

Figure 5: Applying Linear Regression, Decision Tree Regression and Lasso Regression

References

Feizabadi, J., 2022. Machine learning demand forecasting and supply chain performance. International Journal of Logistics Research and Applications, 25(2), pp.119-142.