Project Development Phase
Project Manual

Date	14 November 2023
Team ID	Team- 591756
Project Name	ECOMMERCE SHIPPING PREDICTION USING MACHINE LEARNING
Maximum Marks	15 Marks

ECOMMERCE SHIPPING PREDICTION USING MACHINE LEARNING

Introduction

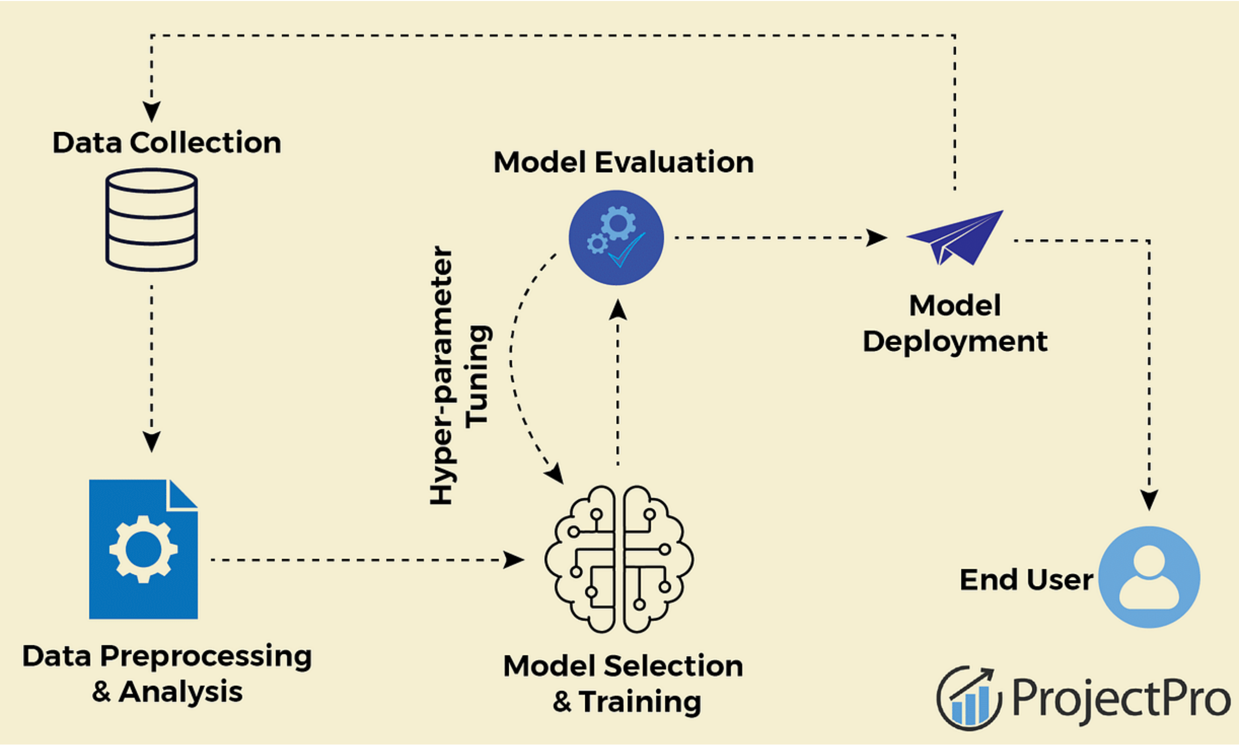
The advent of e-commerce has revolutionized the way businesses operate, offering an unparalleled platform for buying and selling products or services online. One of the critical aspects of e-commerce that directly impacts customer satisfaction and business profitability is shipping. The ability to accurately predict shipping times is crucial for maintaining a competitive edge in the e-commerce industry.

This project, titled "Ecommerce Shipping Prediction Using Machine Learning," aims to develop a predictive model that can accurately estimate shipping times based on various factors such as the product's weight, dimensions, destination, origin, and shipping method. By leveraging machine learning algorithms, we can analyze historical shipping data and identify patterns that can help predict future shipping times.

Accurate shipping predictions can enhance the customer shopping experience by providing them with a reliable delivery timeline. It also allows e-commerce businesses to optimize their logistics and supply chain management, leading to improved operational efficiency and reduced costs.

In the following sections, we will delve into the details of our machine learning model, including data collection and preprocessing, model selection and training, performance evaluation, and potential applications in the e-commerce industry.

Technical Architecture:-



Prerequisites:-

ML Concepts

- Supervised learning: https://www.javatpoint.com/supervised-machine-learning
- Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
- Decisiontree:https://www.javatpoint.com/machine-learning-decision-treclassificationalgorithm
- Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
- KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
- Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-tounderstand-the□math-behind-xgboost/
- Evaluationmetrics:https://www.analyticsvidhya.com/blog/2019/08/11-important-modelevaluation-error-metr ics/
- Flask Basics : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project objectives:-

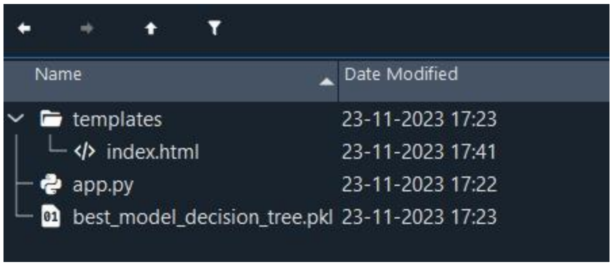
1. **Data Collection and Preprocessing**: Gather historical shipping data from various e-commerce platforms. Clean and preprocess the data to make it suitable for machine learning algorithms.
2. **Feature Selection**: Identify the most relevant features that influence shipping times, such as product weight, dimensions, destination, origin, and shipping method.
3. **Model Development**: Develop a machine learning model that can learn from the historical data and predict shipping times. Experiment with different algorithms to find the one that provides the most accurate predictions.
4. **Model Evaluation**: Evaluate the performance of the machine learning model using appropriate metrics. Fine-tune the model to improve its predictive accuracy.
5. **Integration**: Integrate the predictive model into an e-commerce platform to provide real-time shipping predictions to customers and businesses.
6. **User Experience Enhancement**: By providing accurate shipping predictions, enhance the user experience on the e-commerce platform.
7. **Business Optimization**: Use the predictive model to optimize logistics and supply chain management, leading to improved operational efficiency and reduced costs for e-commerce businesses.

Project Workflow:-

● User interacts with the UI to enter the input.
● Entered input is analysed by the model which is integrated.
● Once model analyses the input the prediction is showcased on the UI To accomplish this, we have to complete all the activities listed below,
● Define Problem / Problem Understanding
> Specify the business problem
○ Business requirements
○ Literature Survey
○ Social or Business Impact.
● Data Collection & Preparation
○ Collect the dataset
○ Data Preparation
● Exploratory Data Analysis
> Descriptive statistical
○ Visual Analysis
● Model Building
> Training the model in multiple algorithms
○ Testing the model
● Performance Testing & Hyperparameter Tuning

> Testing model with multiple evaluation metrics
○ Comparing model accuracy before & after applying hyperparameter tuning
● Model Deployment
> Save the best model
○ Integrate with Web Framework
● Project Demonstration & Documentation
> Record explanation Video for project end to end solution
○ Project Documentation-Step by step project development procedure.

Project Structure:-



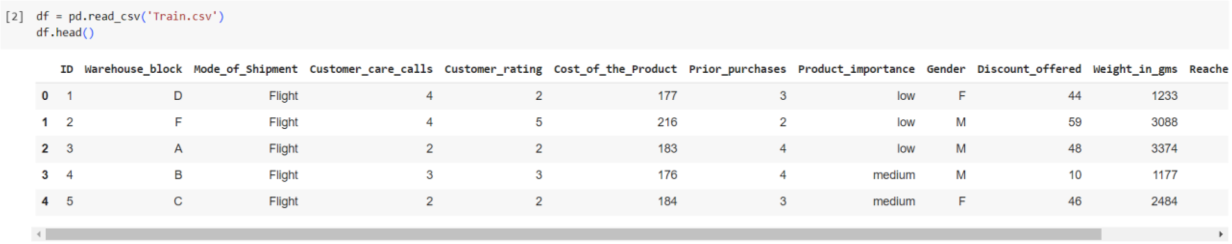
Name	Date Modified
templates	23-11-2023 17:23
index.html	23-11-2023 17:41
app.py	23-11-2023 17:22
best_model_decision_tree.pkl	23-11-2023 17:23

PROJECT DEVELOPMENT:-

DATA COLLECTION:-

The Dataset used in this project is collected from the following link:
https://www.kaggle.com/datasets/prachi13/customer-analytics?select=Train.csv

In this project we have used Train.csv data. This data is downloaded from kaggle.com.

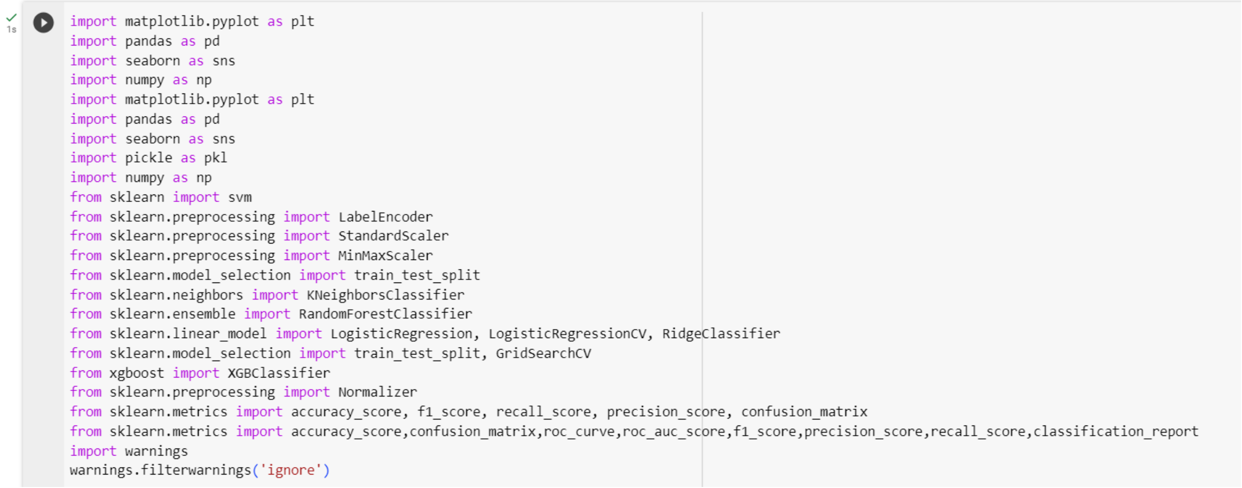


```
[2] df = pd.read_csv('Train.csv')
    df.head()
```

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product_importance	Gender	Discount_offered	Weight_in_gms	Reache
0	1	D	Flight	4	2	177	3	low	F	44	1233	
1	2	F	Flight	4	5	216	2	low	M	59	3088	
2	3	A	Flight	2	2	183	4	low	M	48	3374	
3	4	B	Flight	3	3	176	4	medium	M	10	1177	
4	5	C	Flight	2	2	184	3	medium	F	46	2484	

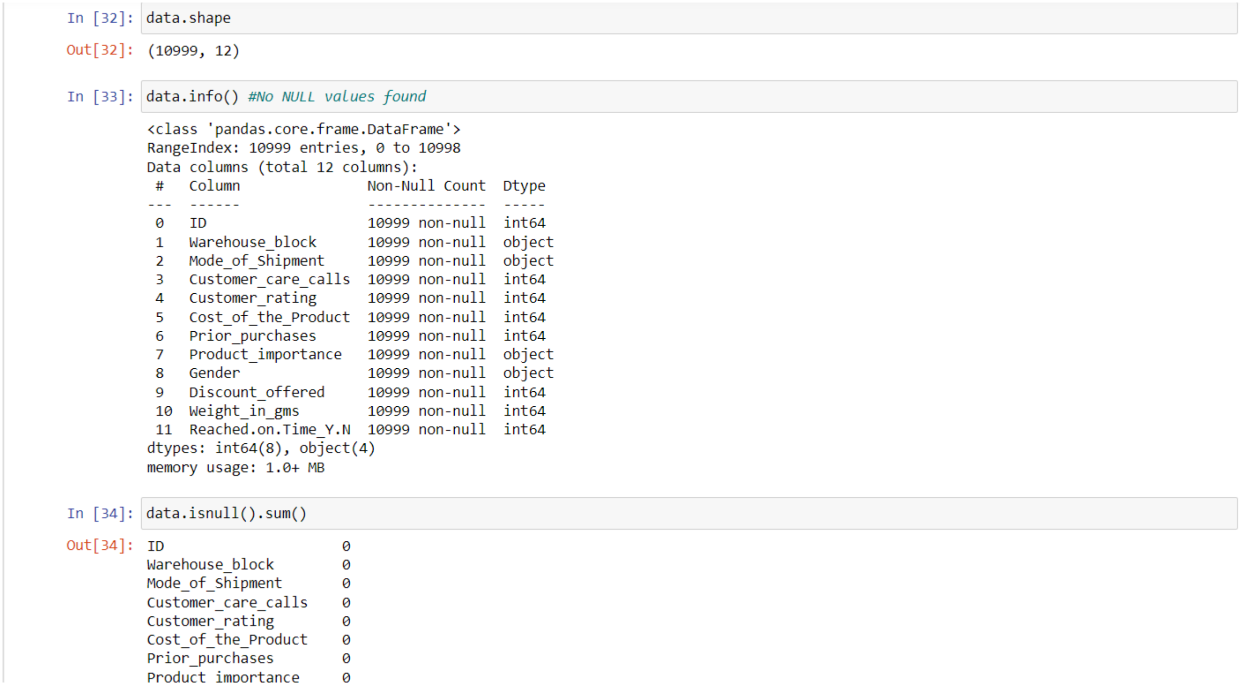
DATA PREPARATION:-

Importing the necessary libraries:-

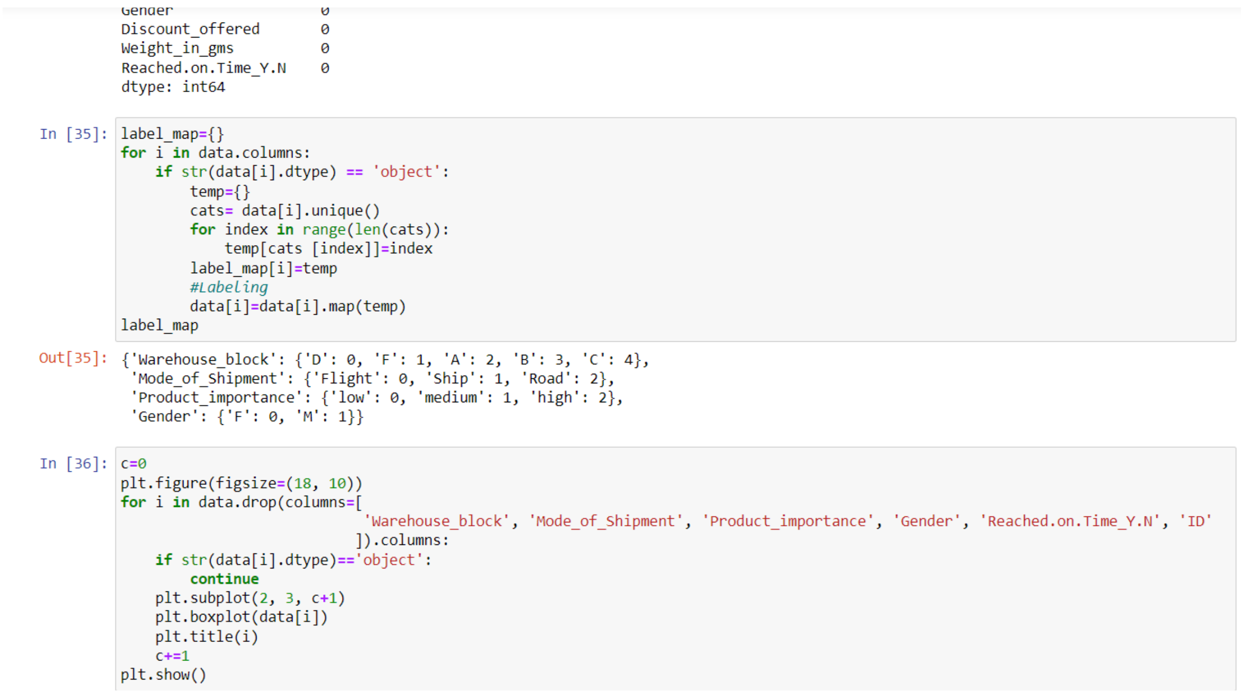


```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import pickle as pkl
import numpy as np
from sklearn import svm
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV, RidgeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from xgboost import XGBClassifier
from sklearn.preprocessing import Normalizer
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score, confusion_matrix
from sklearn.metrics import accuracy_score,confusion_matrix,roc_curve,roc_auc_score,f1_score,precision_score,recall_score,classification_report
import warnings
warnings.filterwarnings('ignore')
```

DATA PREPROCESSING:-



```
In [32]: data.shape

Out[32]: (10999, 12)

In [33]: data.info() #No NULL values found

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 10999 entries, 0 to 10998
         Data columns (total 12 columns):
          #   Column               Non-Null Count  Dtype
         ---  ------               --------------  -----
          0   ID                   10999 non-null  int64
          1   Warehouse_block      10999 non-null  object
          2   Mode_of_Shipment     10999 non-null  object
          3   Customer_care_calls  10999 non-null  int64
          4   Customer_rating      10999 non-null  int64
          5   Cost_of_the_Product  10999 non-null  int64
          6   Prior_purchases      10999 non-null  int64
          7   Product_importance   10999 non-null  object
          8   Gender               10999 non-null  object
          9   Discount_offered     10999 non-null  int64
          10  Weight_in_gms        10999 non-null  int64
          11  Reached.on.Time_Y.N  10999 non-null  int64
         dtypes: int64(8), object(4)
         memory usage: 1.0+ MB

In [34]: data.isnull().sum()

Out[34]: ID                     0
         Warehouse_block        0
         Mode_of_Shipment       0
         Customer_care_calls    0
         Customer_rating        0
         Cost_of_the_Product    0
         Prior_purchases        0
         Product_importance     0
```



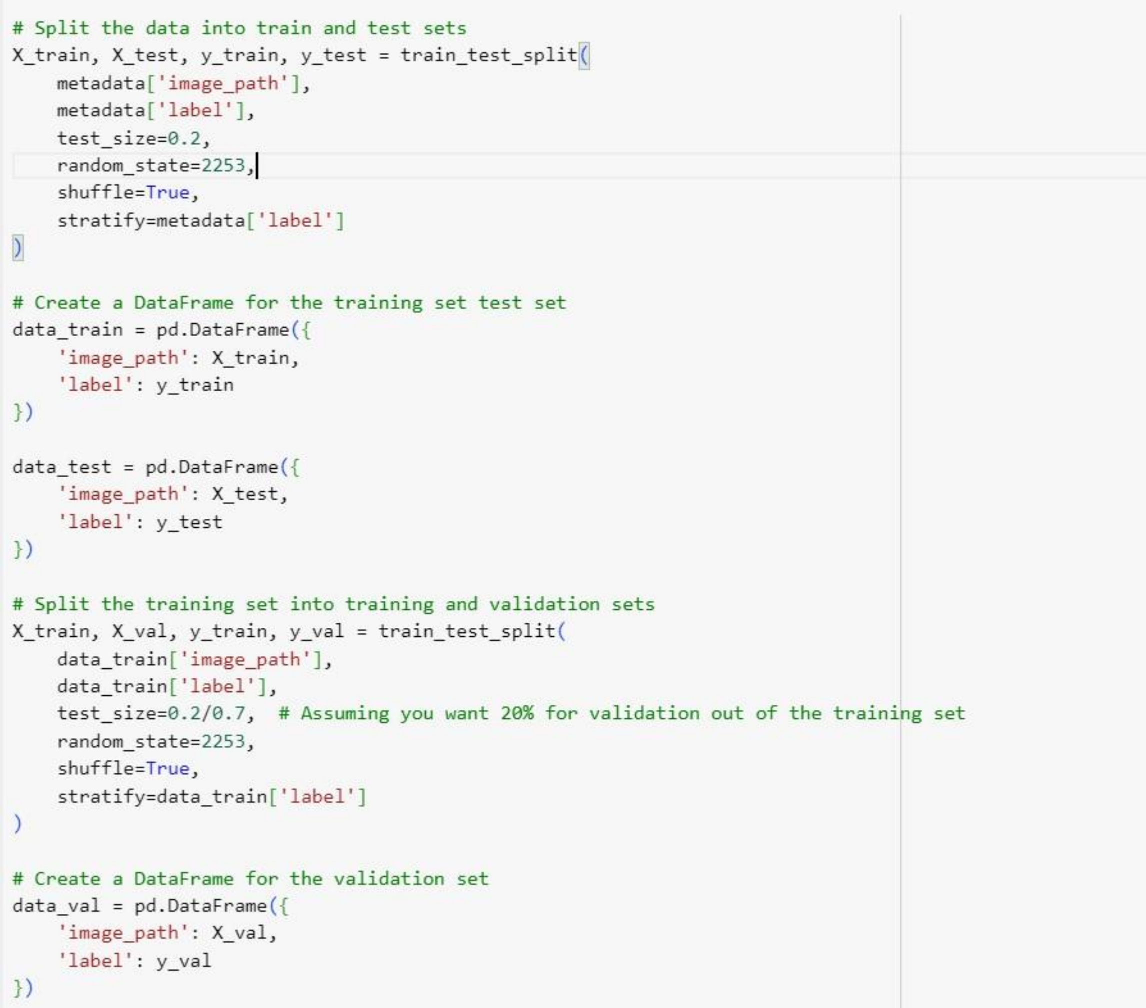
```
Gender                0
Discount_offered      0
Weight_in_gms         0
Reached.on.Time_Y.N   0
dtype: int64
```

In [35]:
```
label_map={}
for i in data.columns:
    if str(data[i].dtype) == 'object':
        temp={}
        cats= data[i].unique()
        for index in range(len(cats)):
            temp[cats [index]]=index
        label_map[i]=temp
        #Labeling
        data[i]=data[i].map(temp)
label_map
```

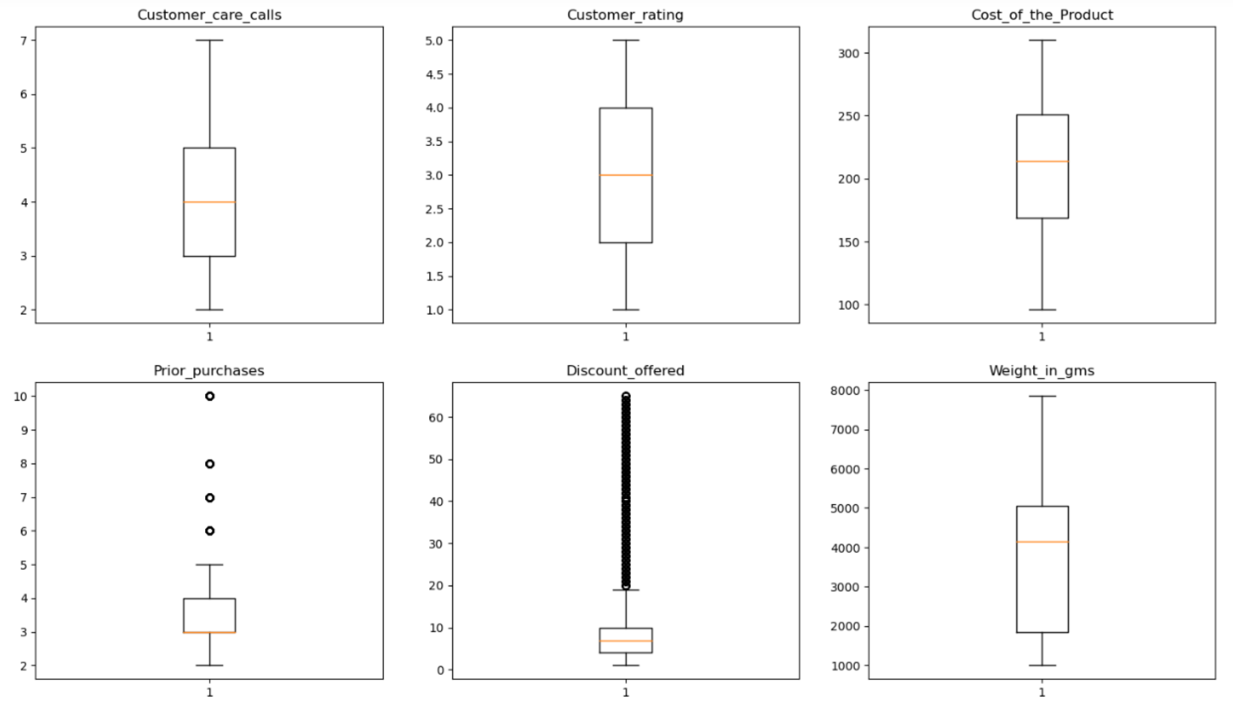
Out[35]:
```
{'Warehouse_block': {'D': 0, 'F': 1, 'A': 2, 'B': 3, 'C': 4},
 'Mode_of_Shipment': {'Flight': 0, 'Ship': 1, 'Road': 2},
 'Product_importance': {'low': 0, 'medium': 1, 'high': 2},
 'Gender': {'F': 0, 'M': 1}}
```

In [36]:
```
c=0
plt.figure(figsize=(18, 10))
for i in data.drop(columns=[
                            'Warehouse_block', 'Mode_of_Shipment', 'Product_importance', 'Gender', 'Reached.on.Time_Y.N', 'ID'
                           ]).columns:
    if str(data[i].dtype)=='object':
        continue
    plt.subplot(2, 3, c+1)
    plt.boxplot(data[i])
    plt.title(i)
    c+=1
plt.show()
```

Splitting the data into train and test sets:-

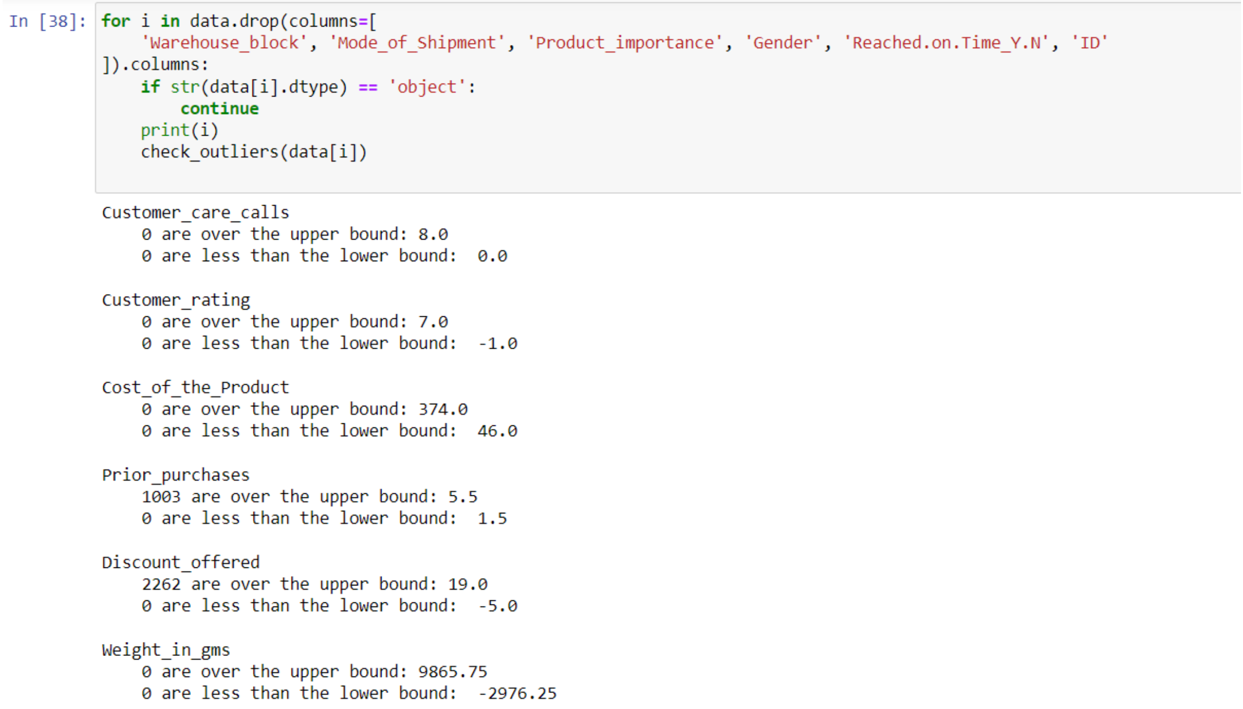


```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    metadata['image_path'],
    metadata['label'],
    test_size=0.2,
    random_state=2253,
    shuffle=True,
    stratify=metadata['label']
)

# Create a DataFrame for the training set test set
data_train = pd.DataFrame({
    'image_path': X_train,
    'label': y_train
})

data_test = pd.DataFrame({
    'image_path': X_test,
    'label': y_test
})

# Split the training set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(
    data_train['image_path'],
    data_train['label'],
    test_size=0.2/0.7,  # Assuming you want 20% for validation out of the training set
    random_state=2253,
    shuffle=True,
    stratify=data_train['label']
)

# Create a DataFrame for the validation set
data_val = pd.DataFrame({
    'image_path': X_val,
    'label': y_val
})
```



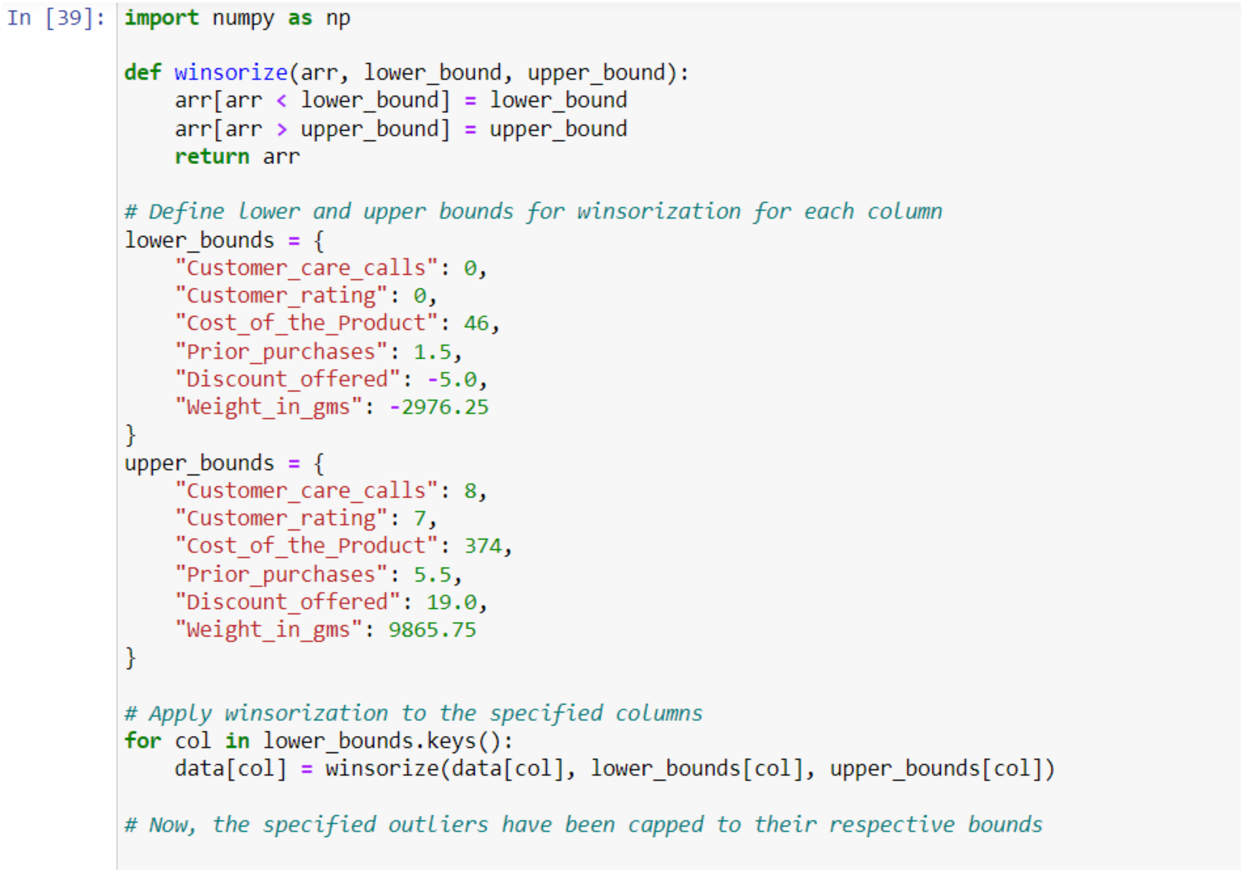
In [37]:
```
def check_outliers(arr):
    Q1= np.percentile (arr, 25, interpolation = 'midpoint')
    Q3= np.percentile (arr, 75, interpolation = 'midpoint')
    IQR = Q3 - Q1


    #Above Upper bound
    upper=Q3+1.5*IQR
    upper_array=np.array(arr>=upper)
    print(' '*3,len (upper_array[upper_array == True]), 'are over the upper bound:',upper)


    #Below Lower bound
    lower= Q1-1.5*IQR
    lower_array=np.array(arr<=lower)
    print(' '*3,len(lower_array[lower_array == True]), 'are less than the lower bound: ', lower, '\n')

for i in data.drop(columns=[
                            'Warehouse_block', 'Mode_of_Shipment', 'Product_importance', 'Gender', 'Reached.on.Time_Y.N', 'ID'
                           ]).columns:
    if str(data[i].dtype)=='object':
        continue
print(i)
check_outliers(data[i])
```

```
Weight_in_gms
    0 are over the upper bound: 9865.75
    0 are less than the lower bound:  -2976.25
```

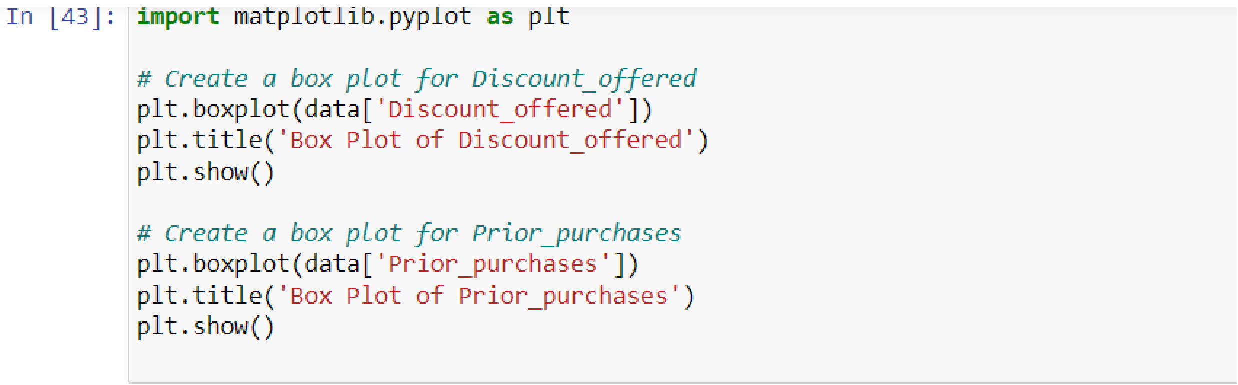


```
In [38]: for i in data.drop(columns=[
             'Warehouse_block', 'Mode_of_Shipment', 'Product_importance', 'Gender', 'Reached.on.Time_Y.N', 'ID'
         ]).columns:
             if str(data[i].dtype) == 'object':
                 continue
             print(i)
             check_outliers(data[i])
```

```
Customer_care_calls
    0 are over the upper bound: 8.0
    0 are less than the lower bound:  0.0

Customer_rating
    0 are over the upper bound: 7.0
    0 are less than the lower bound:  -1.0

Cost_of_the_Product
    0 are over the upper bound: 374.0
    0 are less than the lower bound:  46.0

Prior_purchases
    1003 are over the upper bound: 5.5
    0 are less than the lower bound:  1.5

Discount_offered
    2262 are over the upper bound: 19.0
    0 are less than the lower bound:  -5.0

Weight_in_gms
    0 are over the upper bound: 9865.75
    0 are less than the lower bound:  -2976.25
```

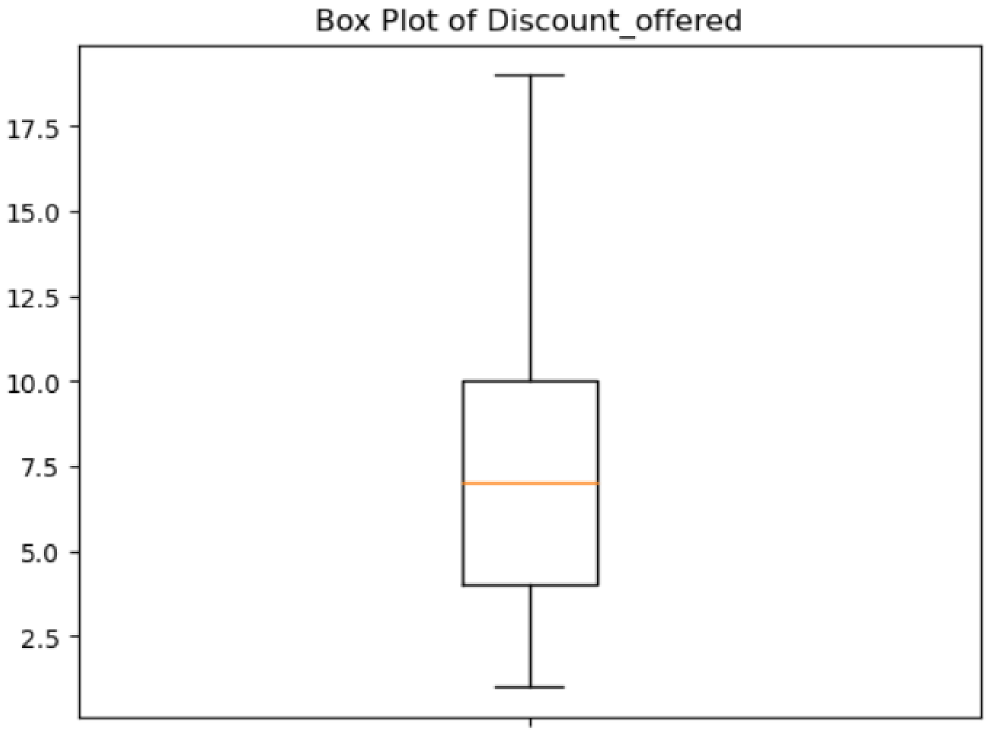


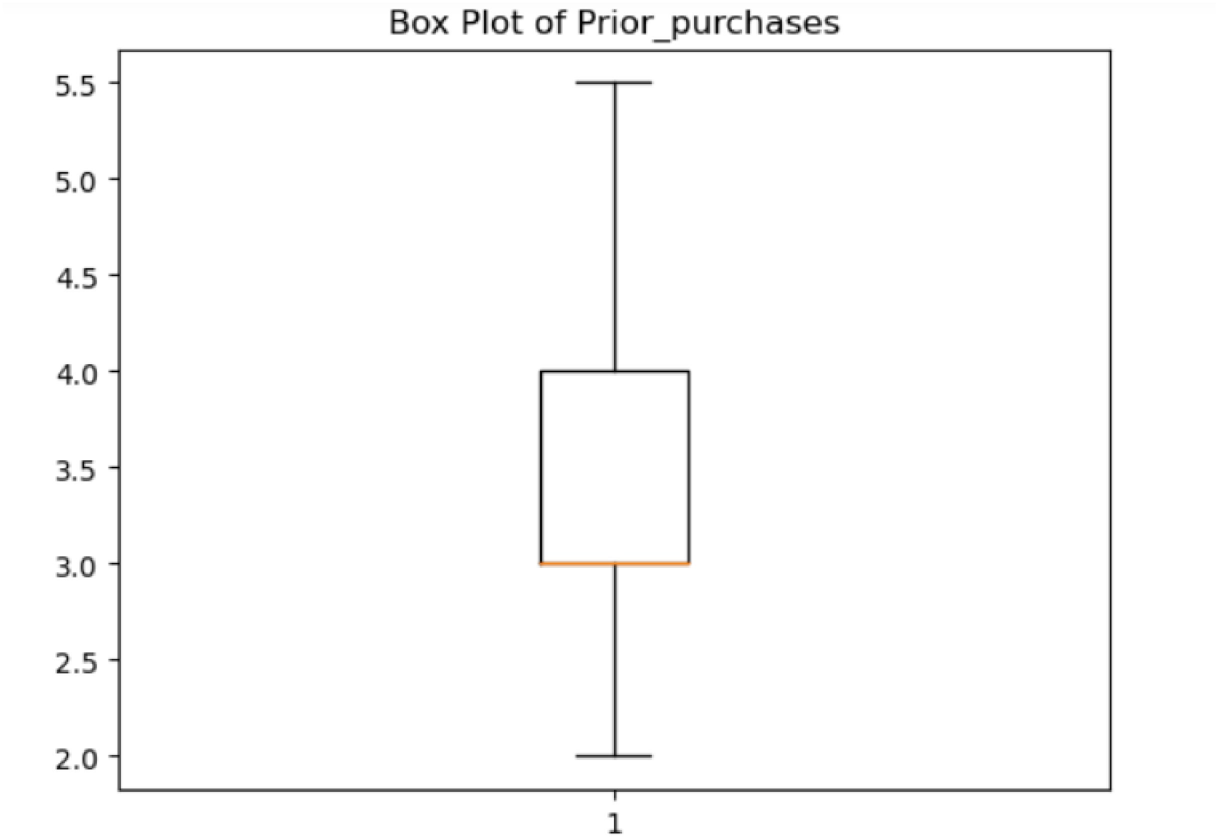
```
In [39]: import numpy as np

         def winsorize(arr, lower_bound, upper_bound):
             arr[arr < lower_bound] = lower_bound
             arr[arr > upper_bound] = upper_bound
             return arr

         # Define lower and upper bounds for winsorization for each column
         lower_bounds = {
             "Customer_care_calls": 0,
             "Customer_rating": 0,
             "Cost_of_the_Product": 46,
             "Prior_purchases": 1.5,
             "Discount_offered": -5.0,
             "Weight_in_gms": -2976.25
         }
         upper_bounds = {
             "Customer_care_calls": 8,
             "Customer_rating": 7,
             "Cost_of_the_Product": 374,
             "Prior_purchases": 5.5,
             "Discount_offered": 19.0,
             "Weight_in_gms": 9865.75
         }

         # Apply winsorization to the specified columns
         for col in lower_bounds.keys():
             data[col] = winsorize(data[col], lower_bounds[col], upper_bounds[col])

         # Now, the specified outliers have been capped to their respective bounds
```



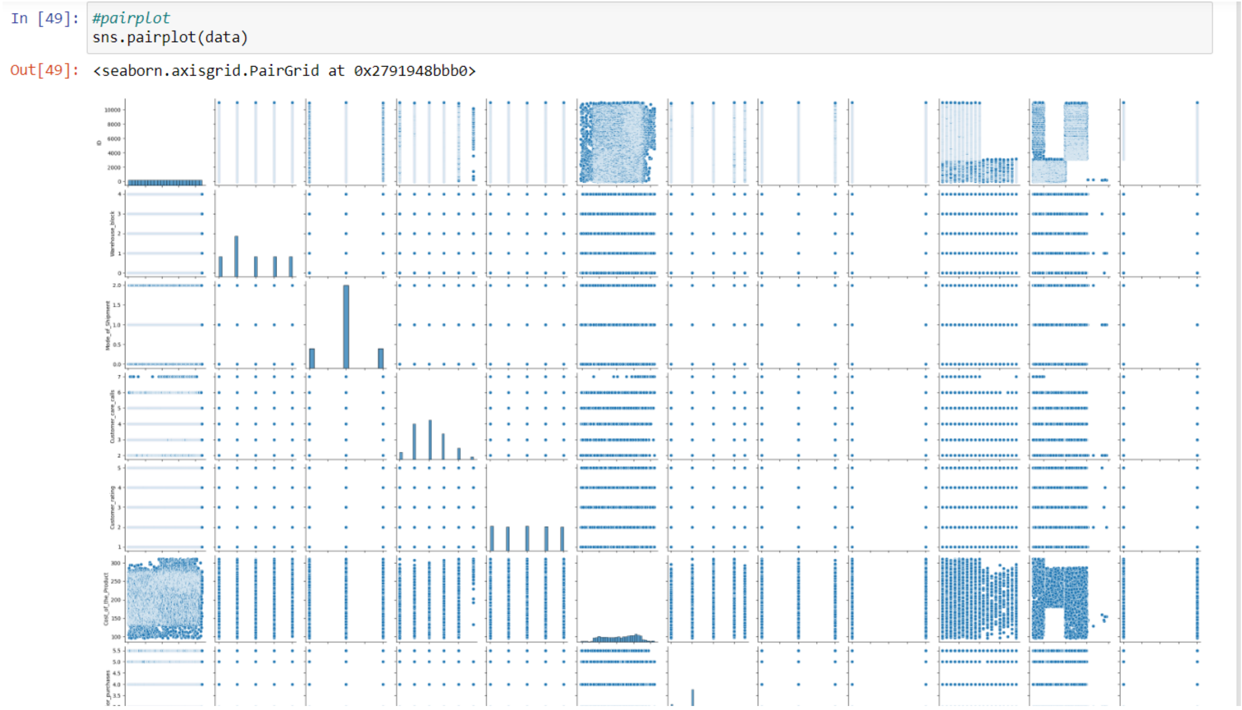
In [43]:

```
import matplotlib.pyplot as plt

# Create a box plot for Discount_offered
plt.boxplot(data['Discount_offered'])
plt.title('Box Plot of Discount_offered')
plt.show()

# Create a box plot for Prior_purchases
plt.boxplot(data['Prior_purchases'])
plt.title('Box Plot of Prior_purchases')
plt.show()
```

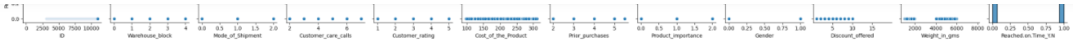




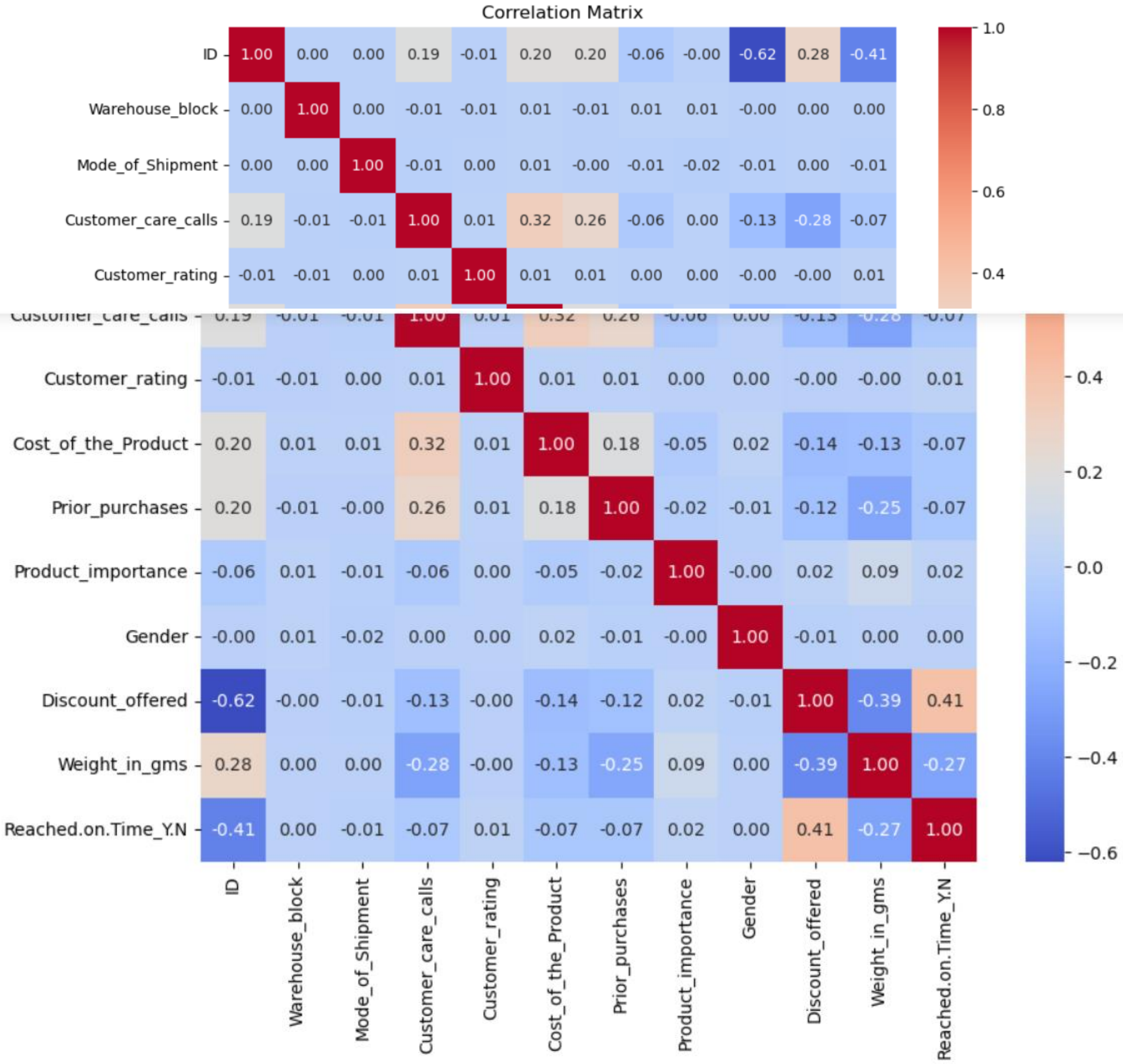
DATA VISUALIZATION:-



```
In [49]: #pairplot
         sns.pairplot(data)
Out[49]: <seaborn.axisgrid.PairGrid at 0x2791948bbb0>
```

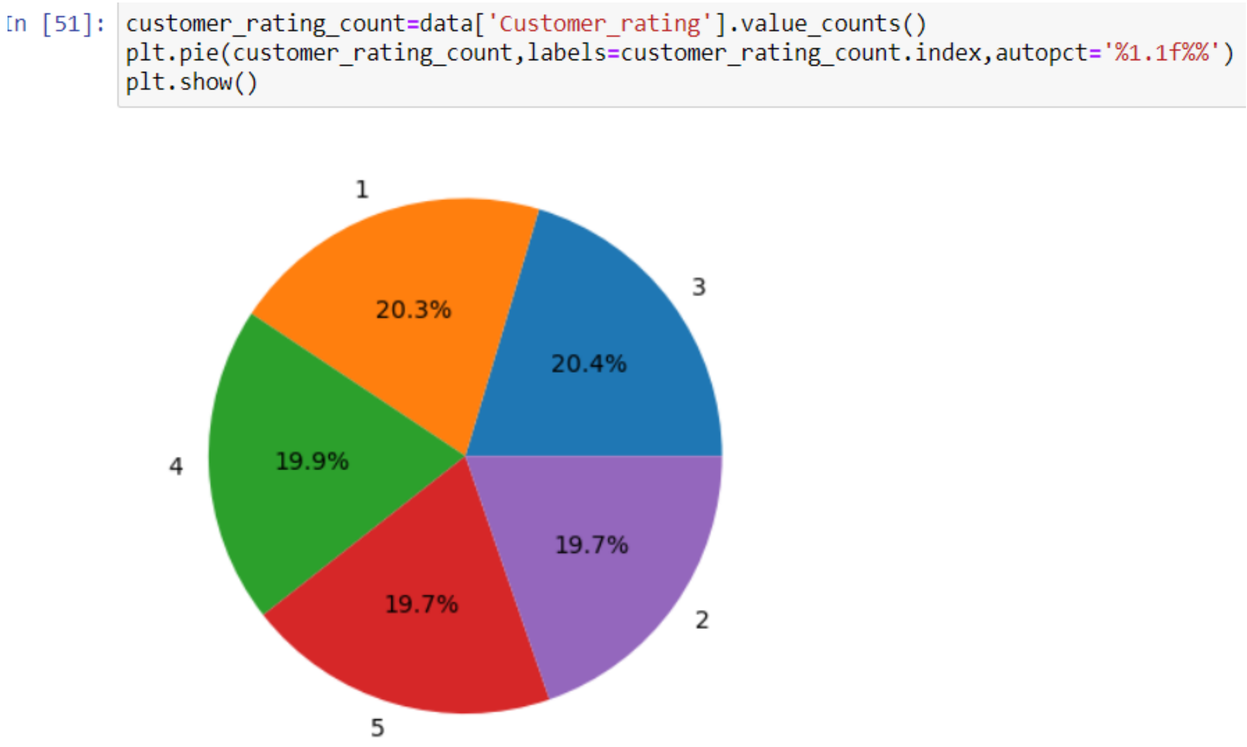


```
In [50]: import seaborn as sns
         import matplotlib.pyplot as plt
         import pandas as pd

         # Assuming you have a DataFrame named 'data' with your dataset

         # Compute the correlation matrix
         corr = data.corr()

         # Plotting the correlation matrix using a heatmap
         plt.figure(figsize=(10, 8))
         sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
         plt.title('Correlation Matrix')
         plt.show()
```

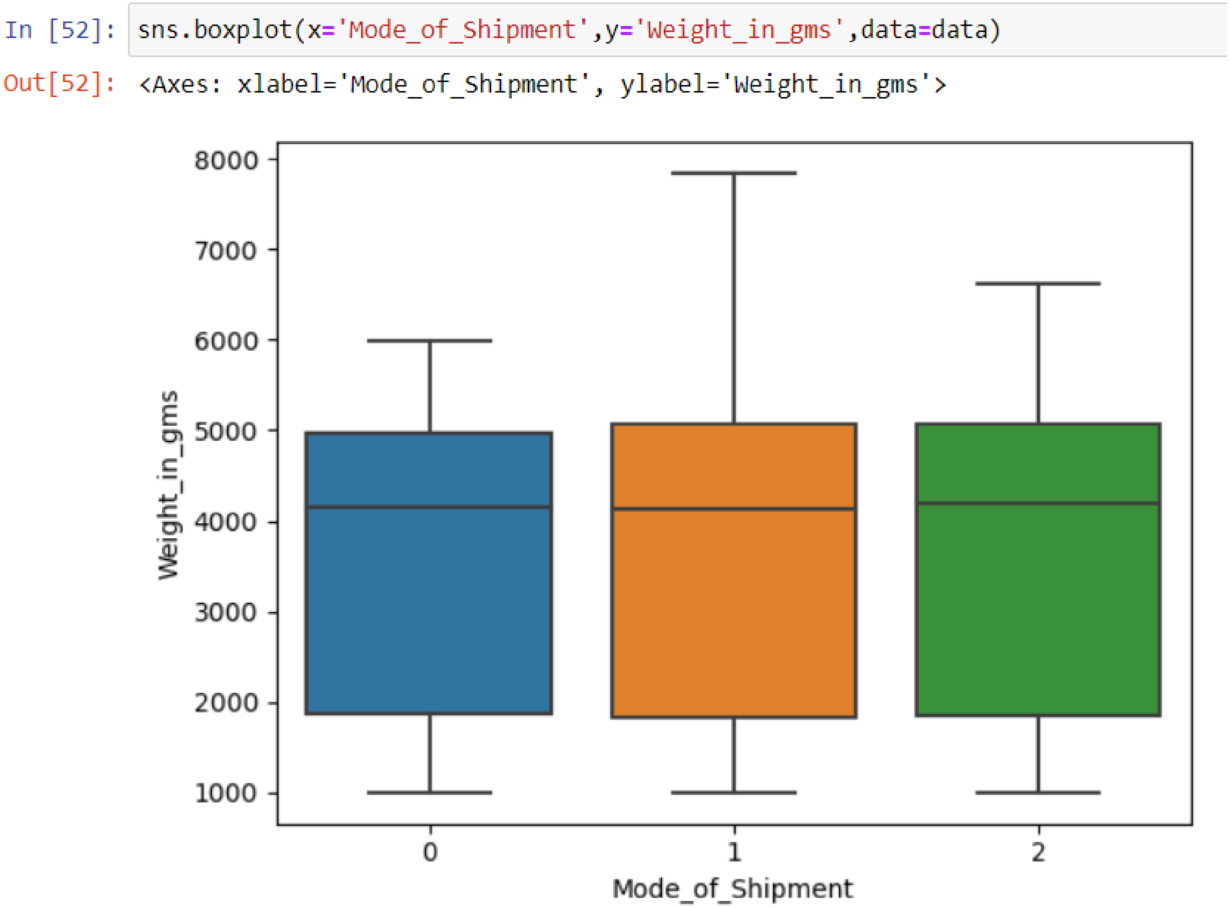


Correlation Matrix

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product_importance	Gender	Discount_offered	Weight_in_gms	Reached.on.Time_Y.N
ID	1.00	0.00	0.00	0.19	-0.01	0.20	0.20	-0.06	-0.00	-0.62	0.28	-0.41
Warehouse_block	0.00	1.00	0.00	-0.01	-0.01	0.01	-0.01	0.01	0.01	-0.00	0.00	0.00
Mode_of_Shipment	0.00	0.00	1.00	-0.01	0.00	0.01	-0.00	-0.01	-0.02	-0.01	0.00	-0.01
Customer_care_calls	0.19	-0.01	-0.01	1.00	0.01	0.32	0.26	-0.06	0.00	-0.13	-0.28	-0.07
Customer_rating	-0.01	-0.01	0.00	0.01	1.00	0.01	0.01	0.00	0.00	-0.00	-0.00	0.01
Cost_of_the_Product	0.20	0.01	0.01	0.32	0.01	1.00	0.18	-0.05	0.02	-0.14	-0.13	-0.07
Prior_purchases	0.20	-0.01	-0.00	0.26	0.01	0.18	1.00	-0.02	-0.01	-0.12	-0.25	-0.07
Product_importance	-0.06	0.01	-0.01	-0.06	0.00	-0.05	-0.02	1.00	-0.00	0.02	0.09	0.02
Gender	-0.00	0.01	-0.02	0.00	0.00	0.02	-0.01	-0.00	1.00	-0.01	0.00	0.00
Discount_offered	-0.62	-0.00	-0.01	-0.13	-0.00	-0.14	-0.12	0.02	-0.01	1.00	-0.39	0.41
Weight_in_gms	0.28	0.00	0.00	-0.28	-0.00	-0.13	-0.25	0.09	0.00	-0.39	1.00	-0.27
Reached.on.Time_Y.N	-0.41	0.00	-0.01	-0.07	0.01	-0.07	-0.07	0.02	0.00	0.41	-0.27	1.00

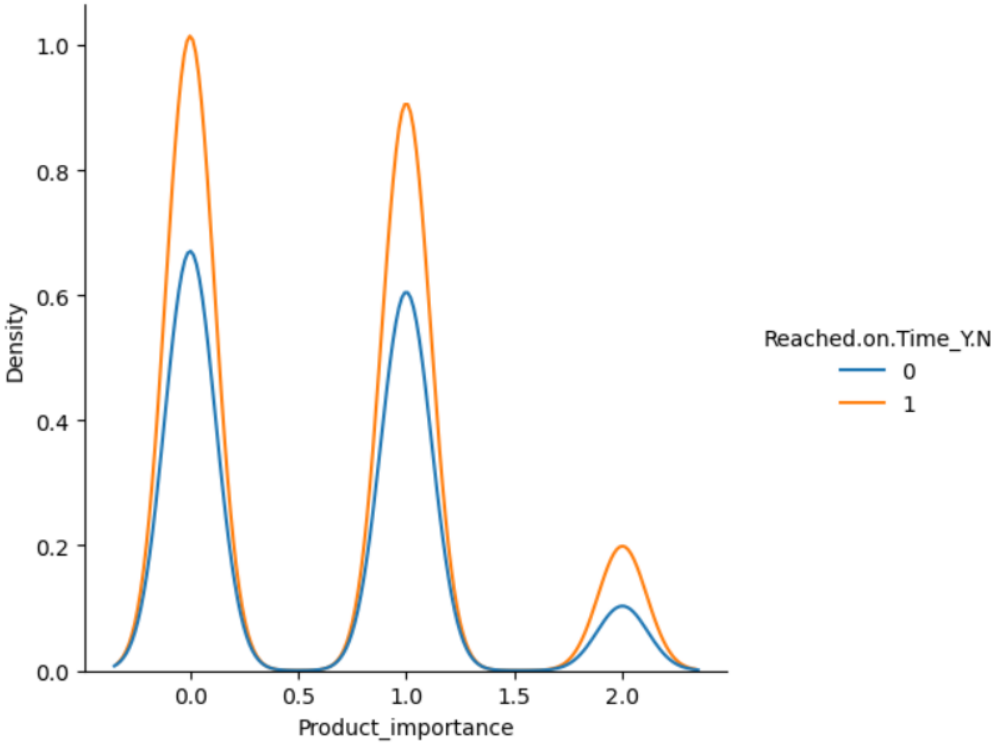


```
In [51]: customer_rating_count=data['Customer_rating'].value_counts()
         plt.pie(customer_rating_count,labels=customer_rating_count.index,autopct='%1.1f%%')
         plt.show()
```



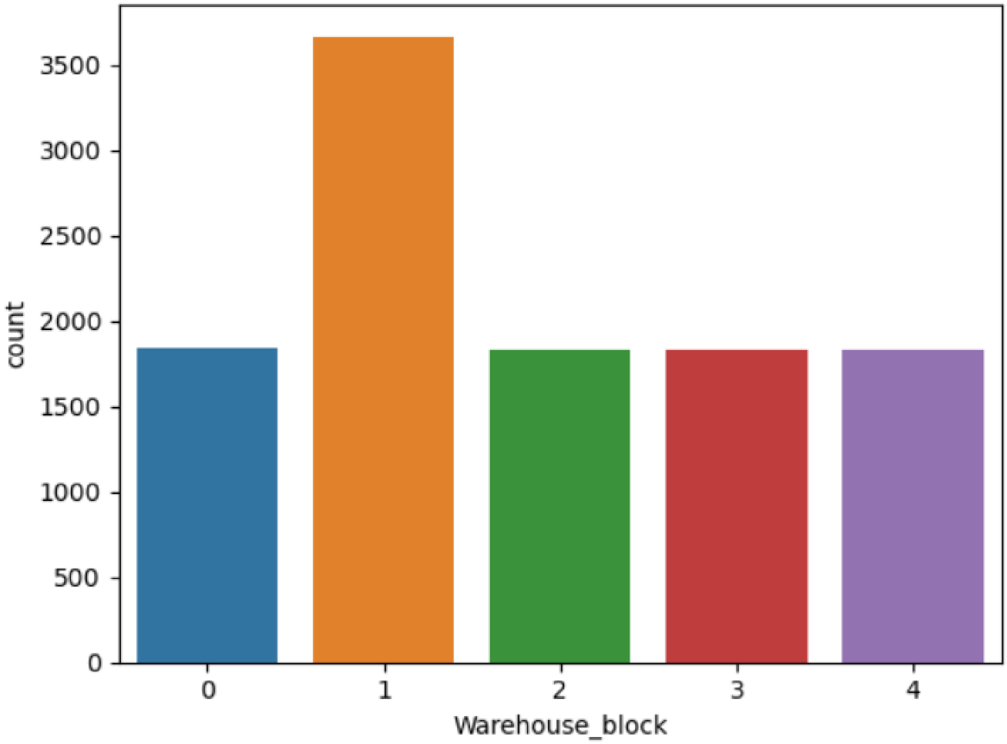
```
In [52]: sns.boxplot(x='Mode_of_Shipment',y='Weight_in_gms',data=data)

Out[52]: <Axes: xlabel='Mode_of_Shipment', ylabel='Weight_in_gms'>
```

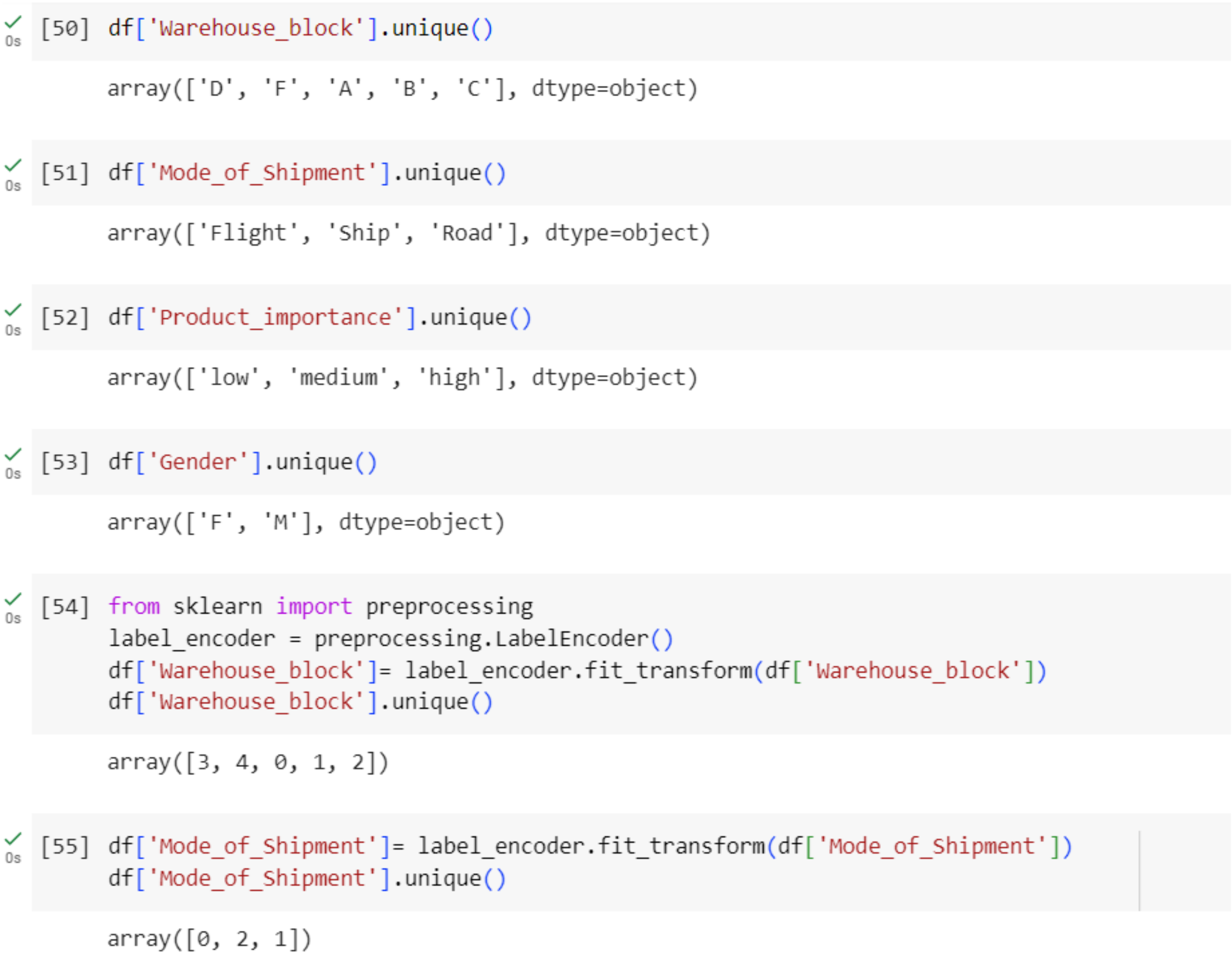
```
In [53]: sns.displot(data,hue='Reached.on.Time_Y.N', x='Product_importance', kind='kde')

Out[53]: <seaborn.axisgrid.FacetGrid at 0x279199c58a0>
```



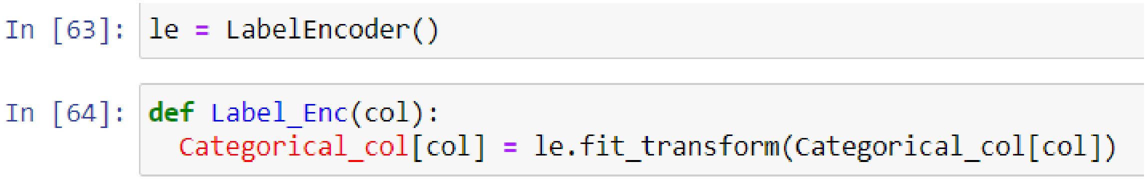
```
In [58]: #Warehouse_block countplot
         sns.countplot(x='Warehouse_block',data=data)
```

```
Out[58]: <Axes: xlabel='Warehouse_block', ylabel='count'>
```





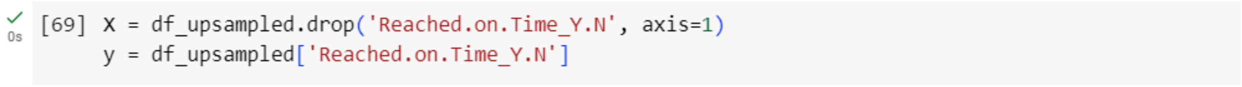
```
[50] df['Warehouse_block'].unique()

     array(['D', 'F', 'A', 'B', 'C'], dtype=object)

[51] df['Mode_of_Shipment'].unique()

     array(['Flight', 'Ship', 'Road'], dtype=object)

[52] df['Product_importance'].unique()

     array(['low', 'medium', 'high'], dtype=object)

[53] df['Gender'].unique()

     array(['F', 'M'], dtype=object)

[54] from sklearn import preprocessing
     label_encoder = preprocessing.LabelEncoder()
     df['Warehouse_block']= label_encoder.fit_transform(df['Warehouse_block'])
     df['Warehouse_block'].unique()

     array([3, 4, 0, 1, 2])

[55] df['Mode_of_Shipment']= label_encoder.fit_transform(df['Mode_of_Shipment'])
     df['Mode_of_Shipment'].unique()

     array([0, 2, 1])
```



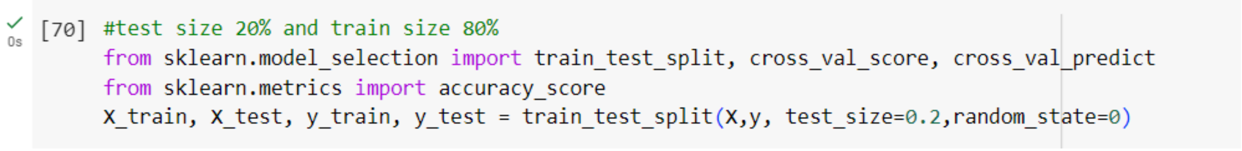
```
In [63]: le = LabelEncoder()

In [64]: def Label_Enc(col):
           Categorical_col[col] = le.fit_transform(Categorical_col[col])
```

DATA AUGMENTATION:-

Applying data augmentation to train, test, validation data:-

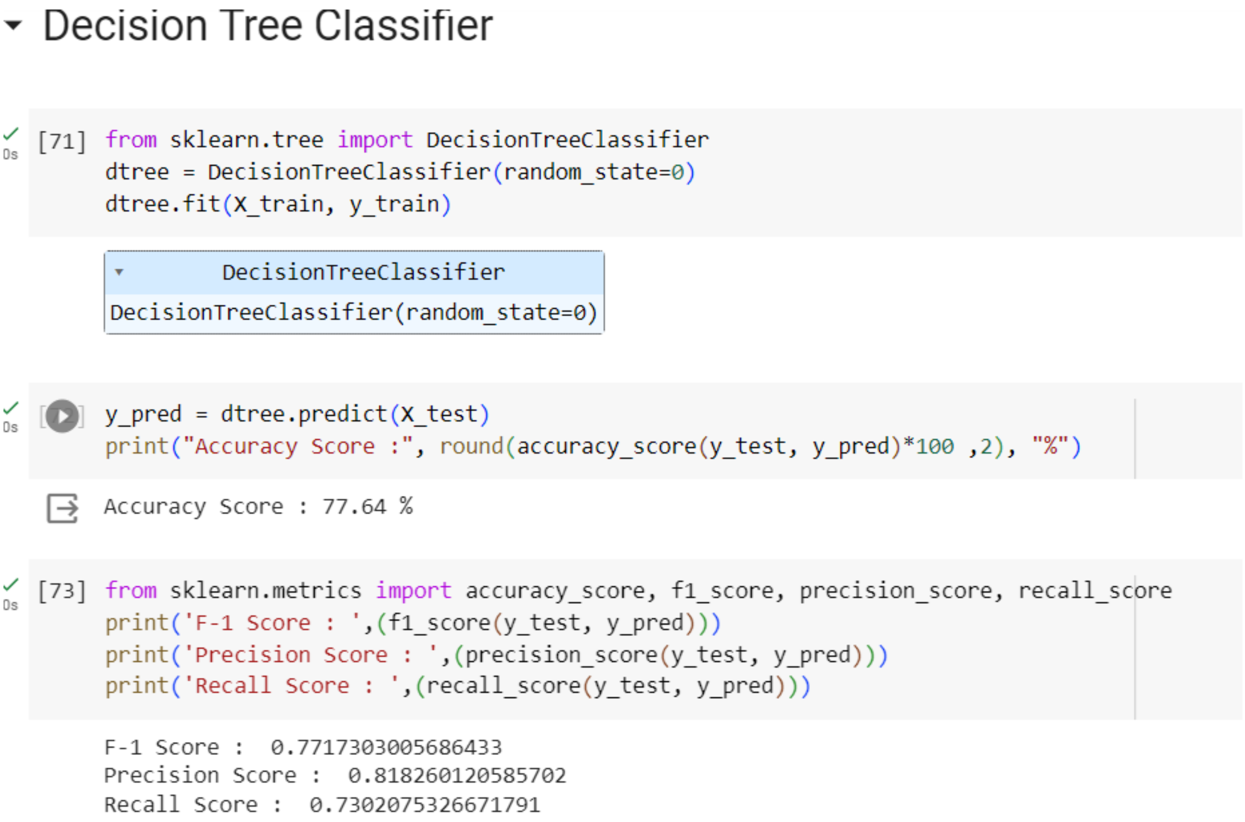


```
[69] X = df_upsampled.drop('Reached.on.Time_Y.N', axis=1)
     y = df_upsampled['Reached.on.Time_Y.N']
```



```
[70] #test size 20% and train size 80%
     from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict
     from sklearn.metrics import accuracy_score
     X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=0)
```

Machine Learning Model Building



Decision Tree Classifier

```
[71] from sklearn.tree import DecisionTreeClassifier
     dtree = DecisionTreeClassifier(random_state=0)
     dtree.fit(X_train, y_train)
```

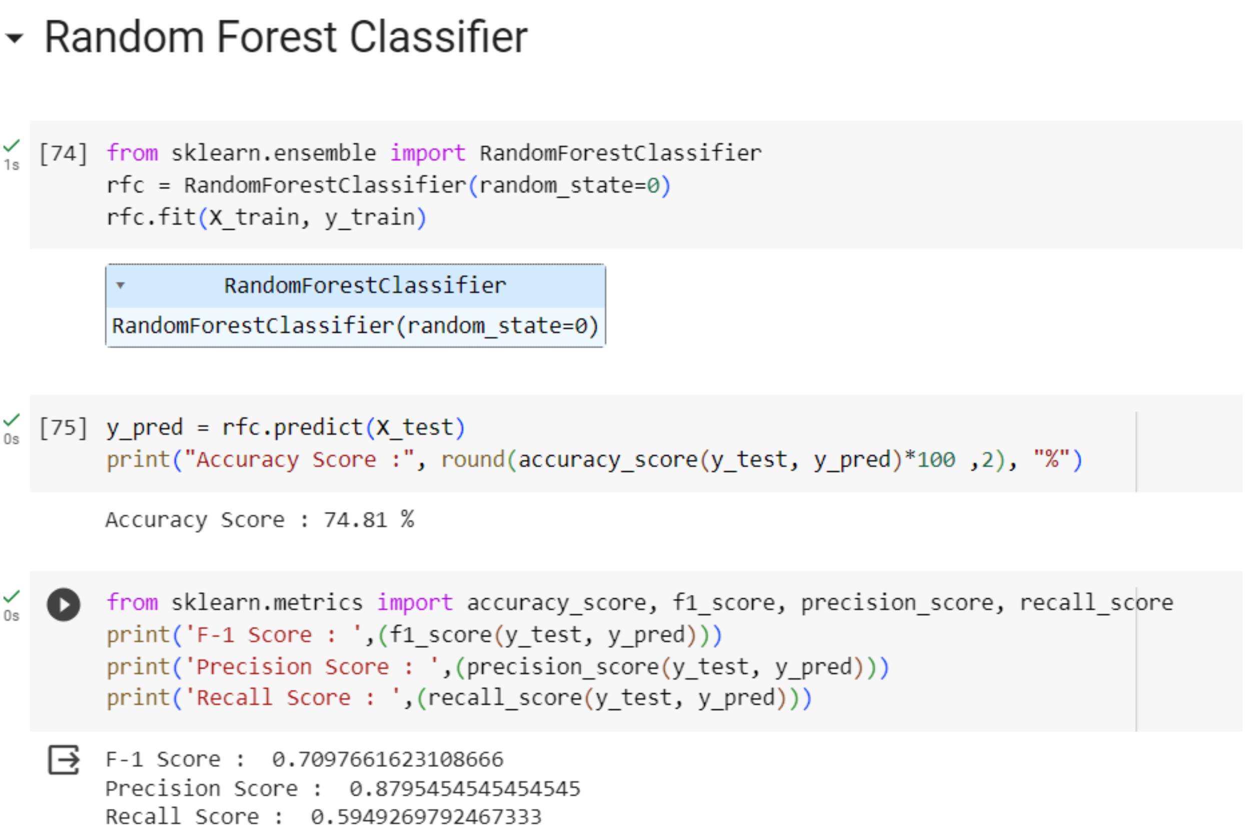
```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

```
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 77.64 %
```

```
[73] from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
     print('F-1 Score : ',(f1_score(y_test, y_pred)))
     print('Precision Score : ',(precision_score(y_test, y_pred)))
     print('Recall Score : ',(recall_score(y_test, y_pred)))
```

```
F-1 Score :  0.7717303005686433
Precision Score :  0.818260120585702
Recall Score :  0.7302075326671791
```



Random Forest Classifier

```
[74] from sklearn.ensemble import RandomForestClassifier
     rfc = RandomForestClassifier(random_state=0)
     rfc.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=0)
```

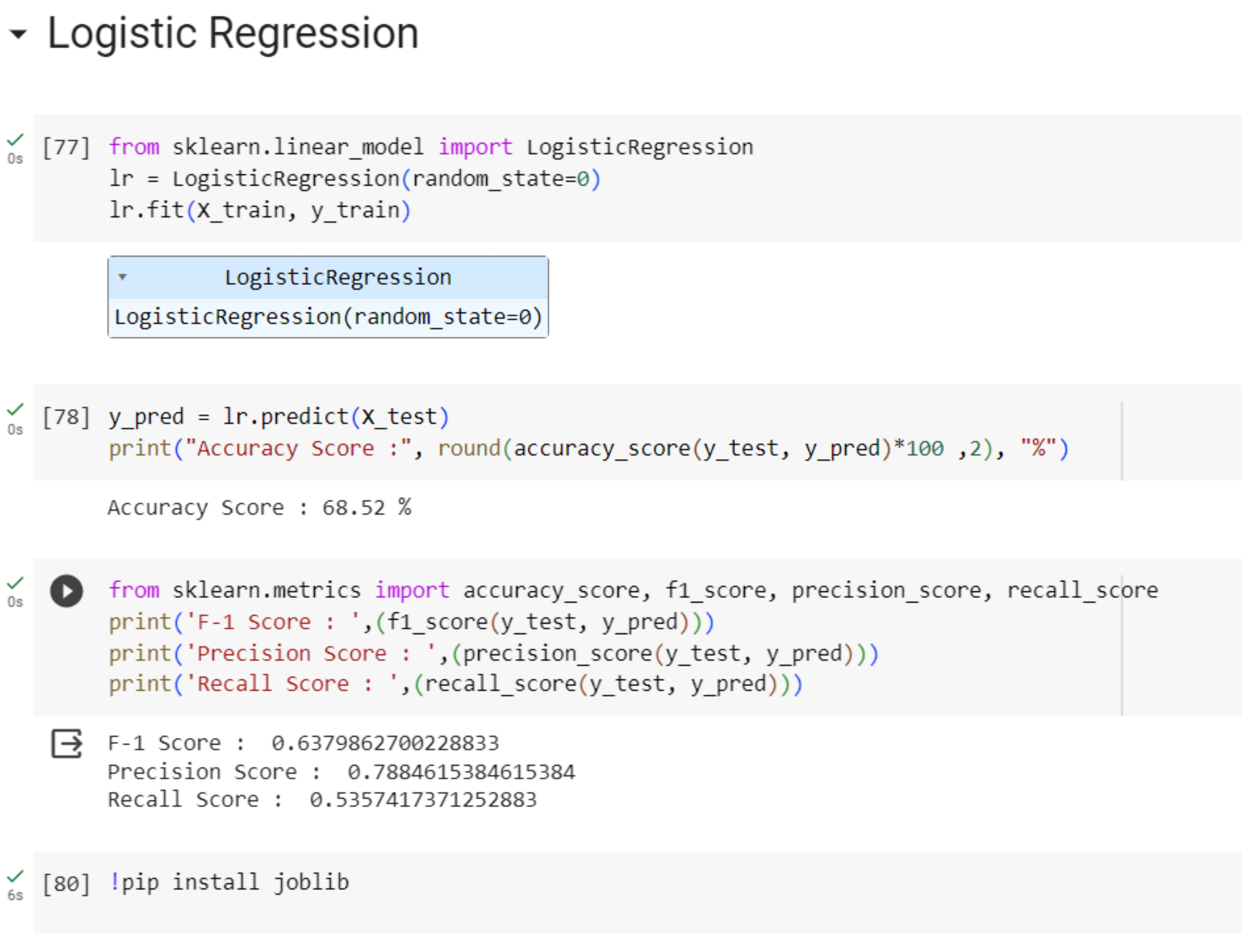
```
[75] y_pred = rfc.predict(X_test)
     print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 74.81 %
```

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
print('F-1 Score : ',(f1_score(y_test, y_pred)))
print('Precision Score : ',(precision_score(y_test, y_pred)))
print('Recall Score : ',(recall_score(y_test, y_pred)))
```

```
F-1 Score :  0.7097661623108666
Precision Score :  0.8795454545454545
Recall Score :  0.5949269792467333
```

MODEL EVALUATION:-



Logistic Regression

```
[77] from sklearn.linear_model import LogisticRegression
     lr = LogisticRegression(random_state=0)
     lr.fit(X_train, y_train)
```

```
         LogisticRegression
LogisticRegression(random_state=0)
```

```
[78] y_pred = lr.predict(X_test)
     print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

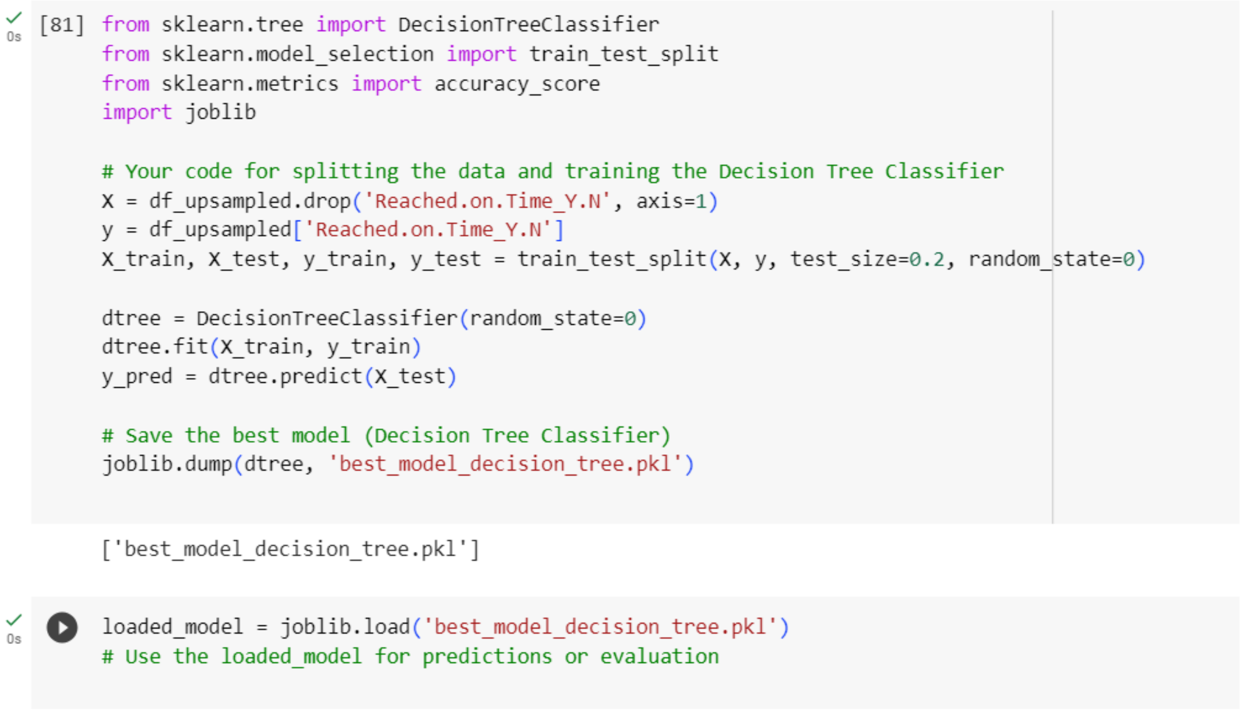
```
Accuracy Score : 68.52 %
```

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
print('F-1 Score : ',(f1_score(y_test, y_pred)))
print('Precision Score : ',(precision_score(y_test, y_pred)))
print('Recall Score : ',(recall_score(y_test, y_pred)))
```

```
F-1 Score :  0.6379862700228833
Precision Score :  0.7884615384615384
Recall Score :  0.5357417371252883
```

```
[80] !pip install joblib
```

LOAD AND TEST THE MODEL:-



[81]

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import joblib

# Your code for splitting the data and training the Decision Tree Classifier
X = df_upsampled.drop('Reached.on.Time_Y.N', axis=1)
y = df_upsampled['Reached.on.Time_Y.N']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

dtree = DecisionTreeClassifier(random_state=0)
dtree.fit(X_train, y_train)
y_pred = dtree.predict(X_test)

# Save the best model (Decision Tree Classifier)
joblib.dump(dtree, 'best_model_decision_tree.pkl')
```

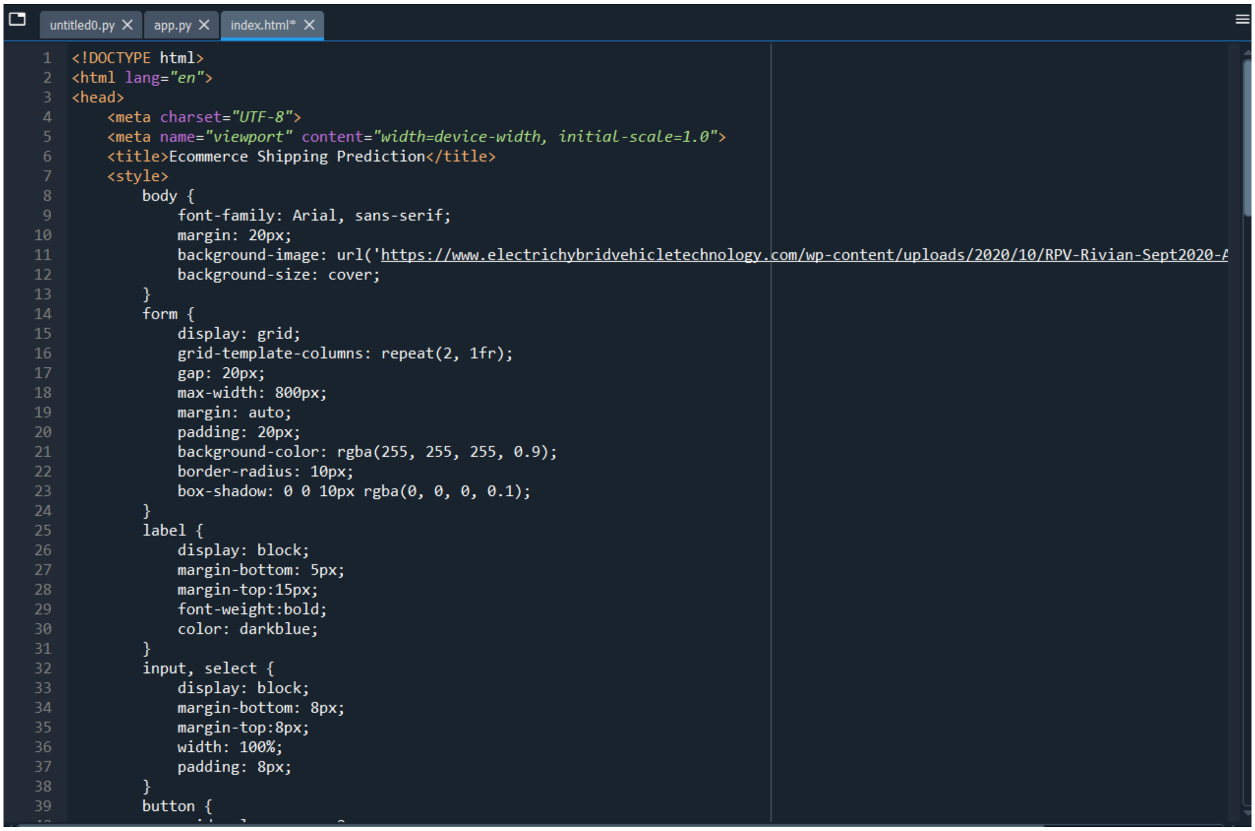
```
['best_model_decision_tree.pkl']
```

```
loaded_model = joblib.load('best_model_decision_tree.pkl')
# Use the loaded_model for predictions or evaluation
```

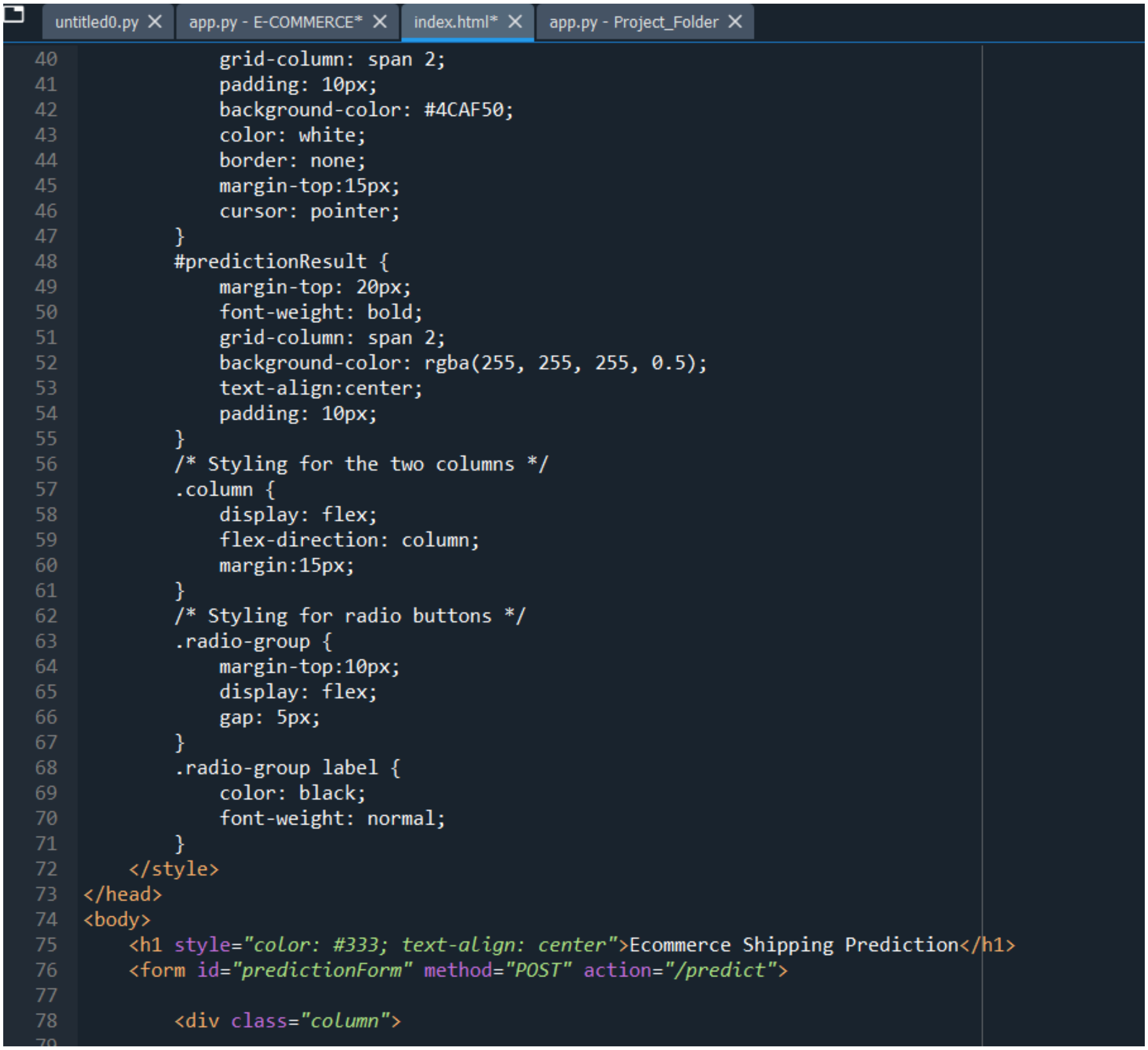
<u>**APPLICATION BUILDING:-**</u>

1. Creating an HTML Page.
2. Adding styles to it using css.
3. Adding actions to it using javascript.
4. Creating App.py python script for web application that uses the model for image classification predictions.
5. Executing these files using Spyder IDE.

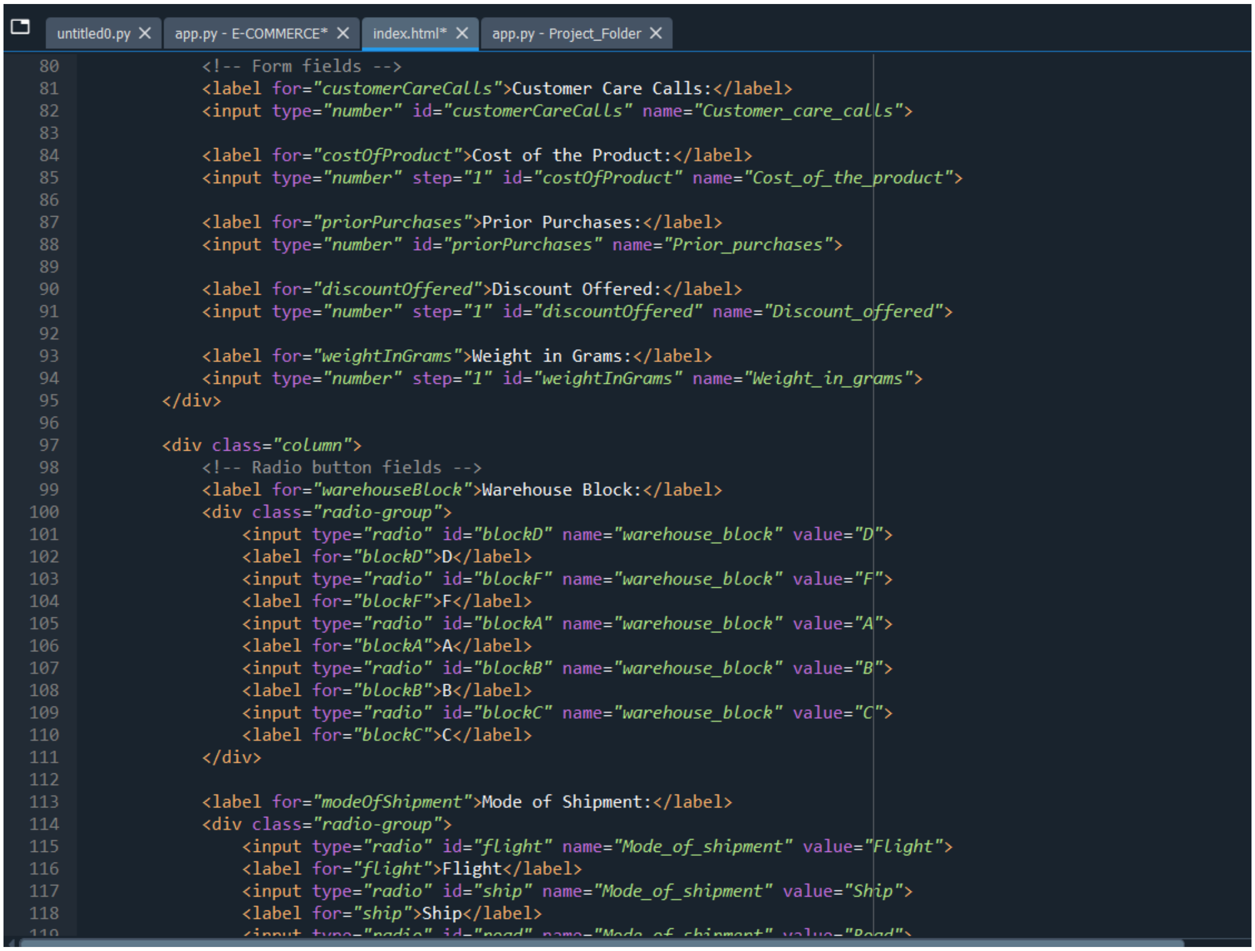
HTML CODE:-



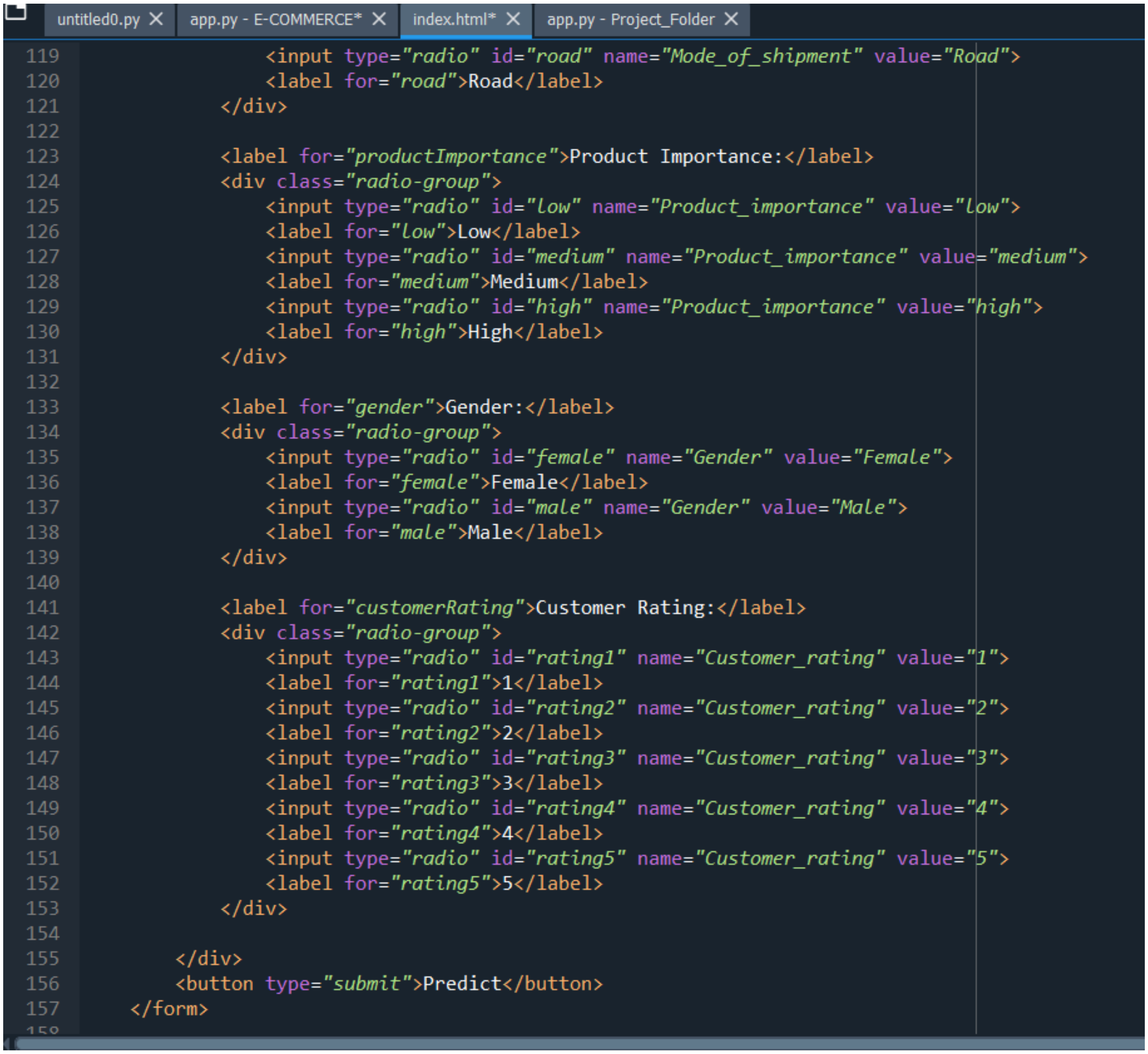
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ecommerce Shipping Prediction</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
            background-image: url('https://www.electrichybridvehicletechnology.com/wp-content/uploads/2020/10/RPV-Rivian-Sept2020-A
            background-size: cover;
        }
        form {
            display: grid;
            grid-template-columns: repeat(2, 1fr);
            gap: 20px;
            max-width: 800px;
            margin: auto;
            padding: 20px;
            background-color: rgba(255, 255, 255, 0.9);
            border-radius: 10px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        label {
            display: block;
            margin-bottom: 5px;
            margin-top:15px;
            font-weight:bold;
            color: darkblue;
        }
        input, select {
            display: block;
            margin-bottom: 8px;
            margin-top:8px;
            width: 100%;
            padding: 8px;
        }
        button {
```



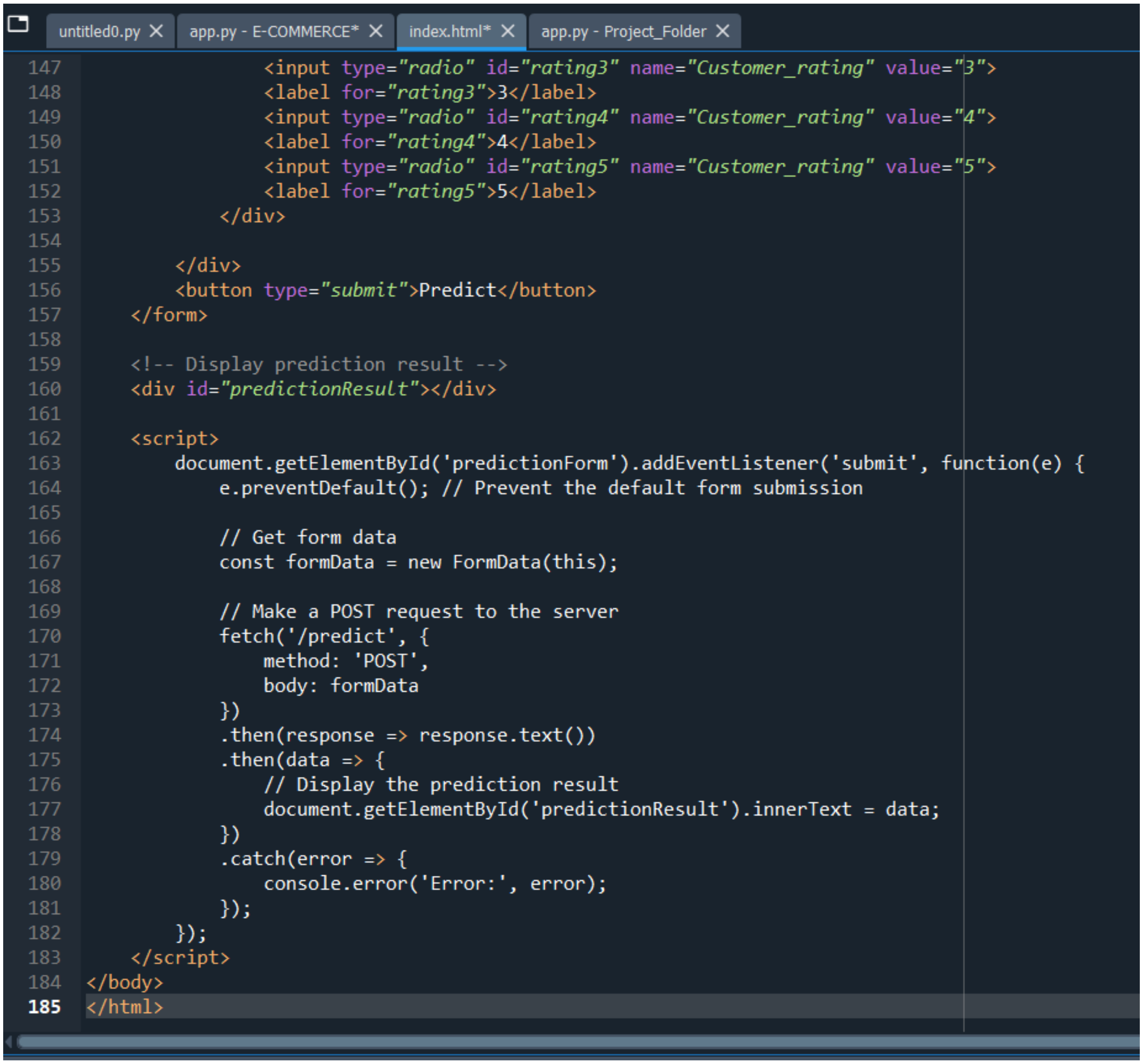
```
            grid-column: span 2;
            padding: 10px;
            background-color: #4CAF50;
            color: white;
            border: none;
            margin-top:15px;
            cursor: pointer;
        }
        #predictionResult {
            margin-top: 20px;
            font-weight: bold;
            grid-column: span 2;
            background-color: rgba(255, 255, 255, 0.5);
            text-align:center;
            padding: 10px;
        }
        /* Styling for the two columns */
        .column {
            display: flex;
            flex-direction: column;
            margin:15px;
        }
        /* Styling for radio buttons */
        .radio-group {
            margin-top:10px;
            display: flex;
            gap: 5px;
        }
        .radio-group label {
            color: black;
            font-weight: normal;
        }
    </style>
</head>
<body>
    <h1 style="color: #333; text-align: center">Ecommerce Shipping Prediction</h1>
    <form id="predictionForm" method="POST" action="/predict">

        <div class="column">
```



```
            <!-- Form fields -->
            <label for="customerCareCalls">Customer Care Calls:</label>
            <input type="number" id="customerCareCalls" name="Customer_care_calls">

            <label for="costOfProduct">Cost of the Product:</label>
            <input type="number" step="1" id="costOfProduct" name="Cost_of_the_product">

            <label for="priorPurchases">Prior Purchases:</label>
            <input type="number" id="priorPurchases" name="Prior_purchases">

            <label for="discountOffered">Discount Offered:</label>
            <input type="number" step="1" id="discountOffered" name="Discount_offered">

            <label for="weightInGrams">Weight in Grams:</label>
            <input type="number" step="1" id="weightInGrams" name="Weight_in_grams">
        </div>

        <div class="column">
            <!-- Radio button fields -->
            <label for="warehouseBlock">Warehouse Block:</label>
            <div class="radio-group">
                <input type="radio" id="blockD" name="warehouse_block" value="D">
                <label for="blockD">D</label>
                <input type="radio" id="blockF" name="warehouse_block" value="F">
                <label for="blockF">F</label>
                <input type="radio" id="blockA" name="warehouse_block" value="A">
                <label for="blockA">A</label>
                <input type="radio" id="blockB" name="warehouse_block" value="B">
                <label for="blockB">B</label>
                <input type="radio" id="blockC" name="warehouse_block" value="C">
                <label for="blockC">C</label>
            </div>

            <label for="modeOfShipment">Mode of Shipment:</label>
            <div class="radio-group">
                <input type="radio" id="flight" name="Mode_of_shipment" value="Flight">
                <label for="flight">Flight</label>
                <input type="radio" id="ship" name="Mode_of_shipment" value="Ship">
                <label for="ship">Ship</label>
                <input type="radio" id="road" name="Mode_of_shipment" value="Road">
```

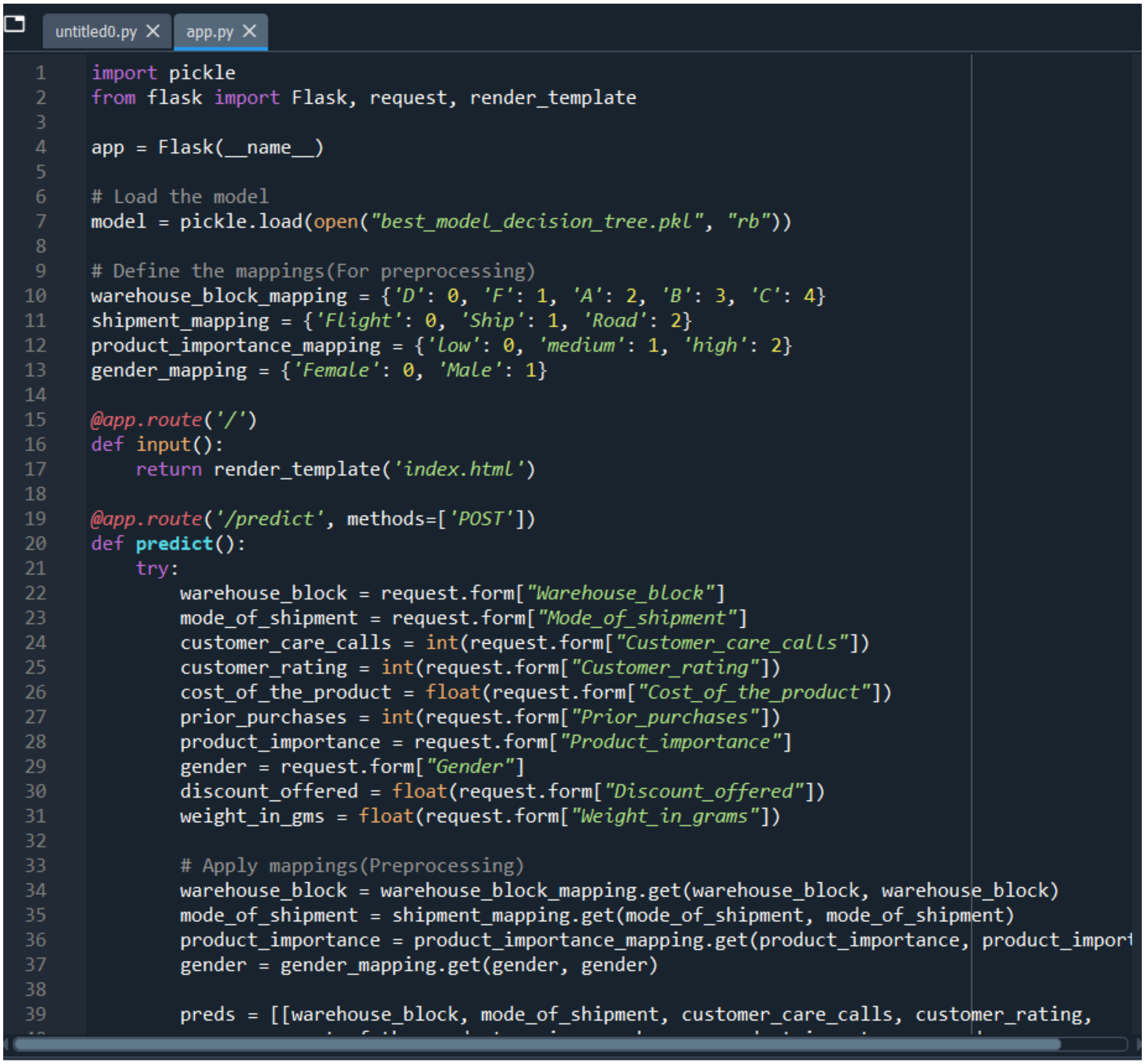


```
                <input type="radio" id="road" name="Mode_of_shipment" value="Road">
                <label for="road">Road</label>
            </div>

            <label for="productImportance">Product Importance:</label>
            <div class="radio-group">
                <input type="radio" id="low" name="Product_importance" value="low">
                <label for="low">Low</label>
                <input type="radio" id="medium" name="Product_importance" value="medium">
                <label for="medium">Medium</label>
                <input type="radio" id="high" name="Product_importance" value="high">
                <label for="high">High</label>
            </div>

            <label for="gender">Gender:</label>
            <div class="radio-group">
                <input type="radio" id="female" name="Gender" value="Female">
                <label for="female">Female</label>
                <input type="radio" id="male" name="Gender" value="Male">
                <label for="male">Male</label>
            </div>

            <label for="customerRating">Customer Rating:</label>
            <div class="radio-group">
                <input type="radio" id="rating1" name="Customer_rating" value="1">
                <label for="rating1">1</label>
                <input type="radio" id="rating2" name="Customer_rating" value="2">
                <label for="rating2">2</label>
                <input type="radio" id="rating3" name="Customer_rating" value="3">
                <label for="rating3">3</label>
                <input type="radio" id="rating4" name="Customer_rating" value="4">
                <label for="rating4">4</label>
                <input type="radio" id="rating5" name="Customer_rating" value="5">
                <label for="rating5">5</label>
            </div>

        </div>
        <button type="submit">Predict</button>
    </form>
```

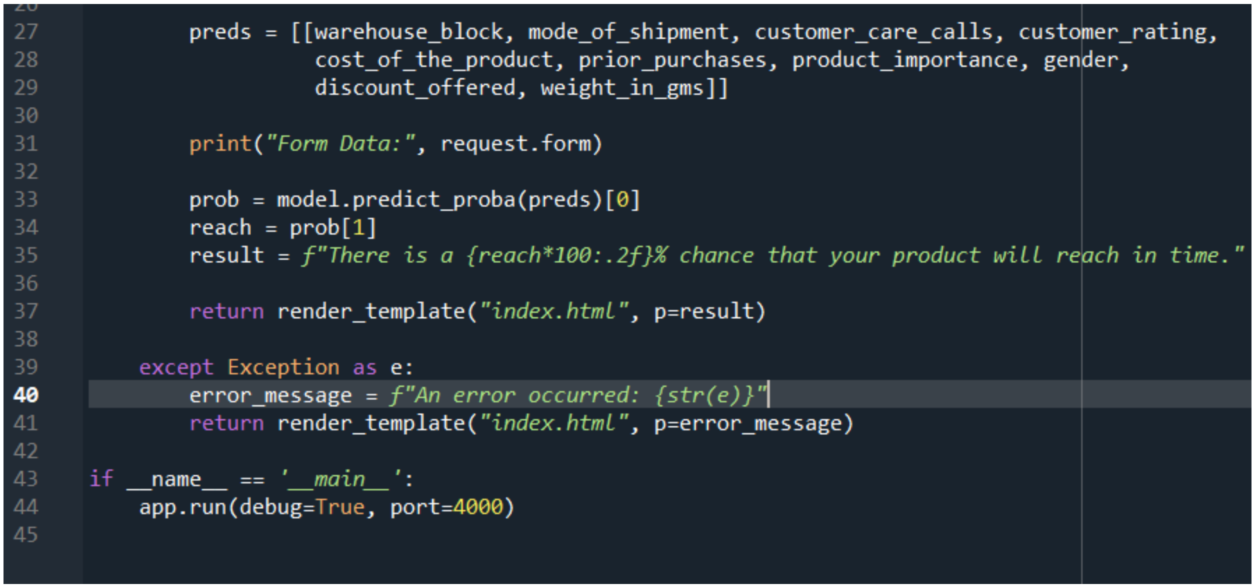


```
                <input type="radio" id="rating3" name="Customer_rating" value="3">
                <label for="rating3">3</label>
                <input type="radio" id="rating4" name="Customer_rating" value="4">
                <label for="rating4">4</label>
                <input type="radio" id="rating5" name="Customer_rating" value="5">
                <label for="rating5">5</label>
            </div>

        </div>
        <button type="submit">Predict</button>
    </form>

    <!-- Display prediction result -->
    <div id="predictionResult"></div>

    <script>
        document.getElementById('predictionForm').addEventListener('submit', function(e) {
            e.preventDefault(); // Prevent the default form submission

            // Get form data
            const formData = new FormData(this);

            // Make a POST request to the server
            fetch('/predict', {
                method: 'POST',
                body: formData
            })
            .then(response => response.text())
            .then(data => {
                // Display the prediction result
                document.getElementById('predictionResult').innerText = data;
            })
            .catch(error => {
                console.error('Error:', error);
            });
        });
    </script>
</body>
</html>
```

App.py CODE:-

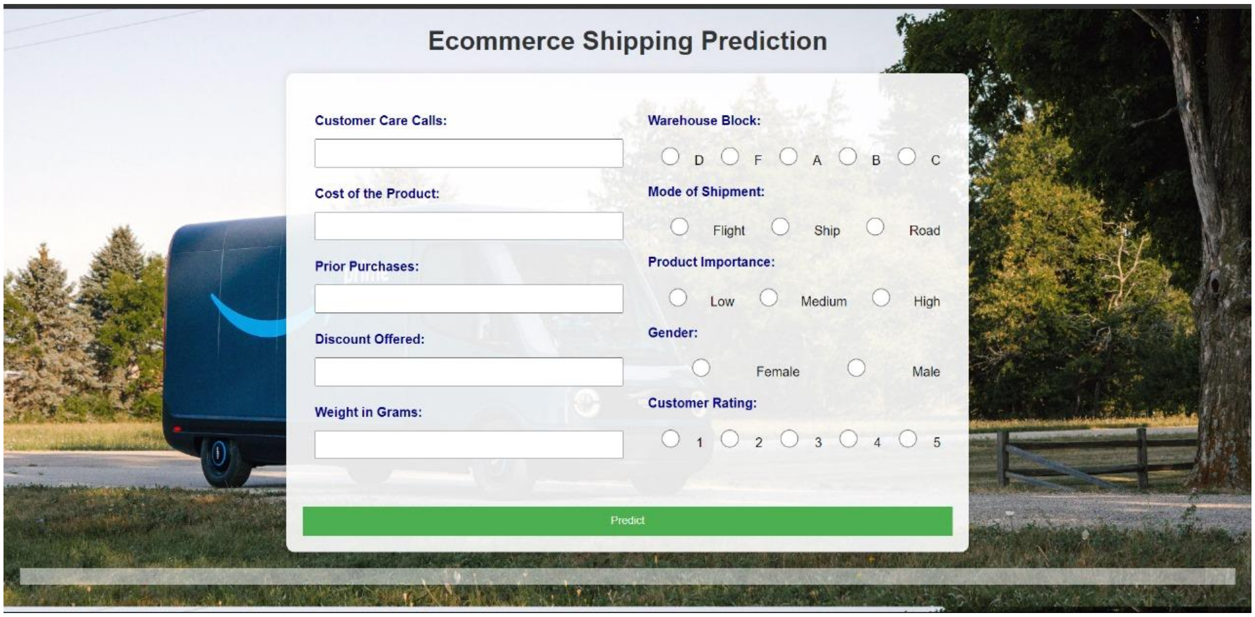


```python
import pickle
from flask import Flask, request, render_template

app = Flask(__name__)

# Load the model
model = pickle.load(open("best_model_decision_tree.pkl", "rb"))

# Define the mappings(For preprocessing)
warehouse_block_mapping = {'D': 0, 'F': 1, 'A': 2, 'B': 3, 'C': 4}
shipment_mapping = {'Flight': 0, 'Ship': 1, 'Road': 2}
product_importance_mapping = {'low': 0, 'medium': 1, 'high': 2}
gender_mapping = {'Female': 0, 'Male': 1}

@app.route('/')
def input():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    try:
        warehouse_block = request.form["Warehouse_block"]
        mode_of_shipment = request.form["Mode_of_shipment"]
        customer_care_calls = int(request.form["Customer_care_calls"])
        customer_rating = int(request.form["Customer_rating"])
        cost_of_the_product = float(request.form["Cost_of_the_product"])
        prior_purchases = int(request.form["Prior_purchases"])
        product_importance = request.form["Product_importance"]
        gender = request.form["Gender"]
        discount_offered = float(request.form["Discount_offered"])
        weight_in_gms = float(request.form["Weight_in_grams"])

        # Apply mappings(Preprocessing)
        warehouse_block = warehouse_block_mapping.get(warehouse_block, warehouse_block)
        mode_of_shipment = shipment_mapping.get(mode_of_shipment, mode_of_shipment)
        product_importance = product_importance_mapping.get(product_importance, product_import
        gender = gender_mapping.get(gender, gender)

        preds = [[warehouse_block, mode_of_shipment, customer_care_calls, customer_rating,
```



```python
        preds = [[warehouse_block, mode_of_shipment, customer_care_calls, customer_rating,
                  cost_of_the_product, prior_purchases, product_importance, gender,
                  discount_offered, weight_in_gms]]

        print("Form Data:", request.form)

        prob = model.predict_proba(preds)[0]
        reach = prob[1]
        result = f"There is a {reach*100:.2f}% chance that your product will reach in time."

        return render_template("index.html", p=result)

    except Exception as e:
        error_message = f"An error occurred: {str(e)}"
        return render_template("index.html", p=error_message)

if __name__ == '__main__':
    app.run(debug=True, port=4000)
```

WEB APPLICATION:-



Ecommerce Shipping Prediction

Customer Care Calls:

Cost of the Product:

Prior Purchases:

Discount Offered:

Weight in Grams:

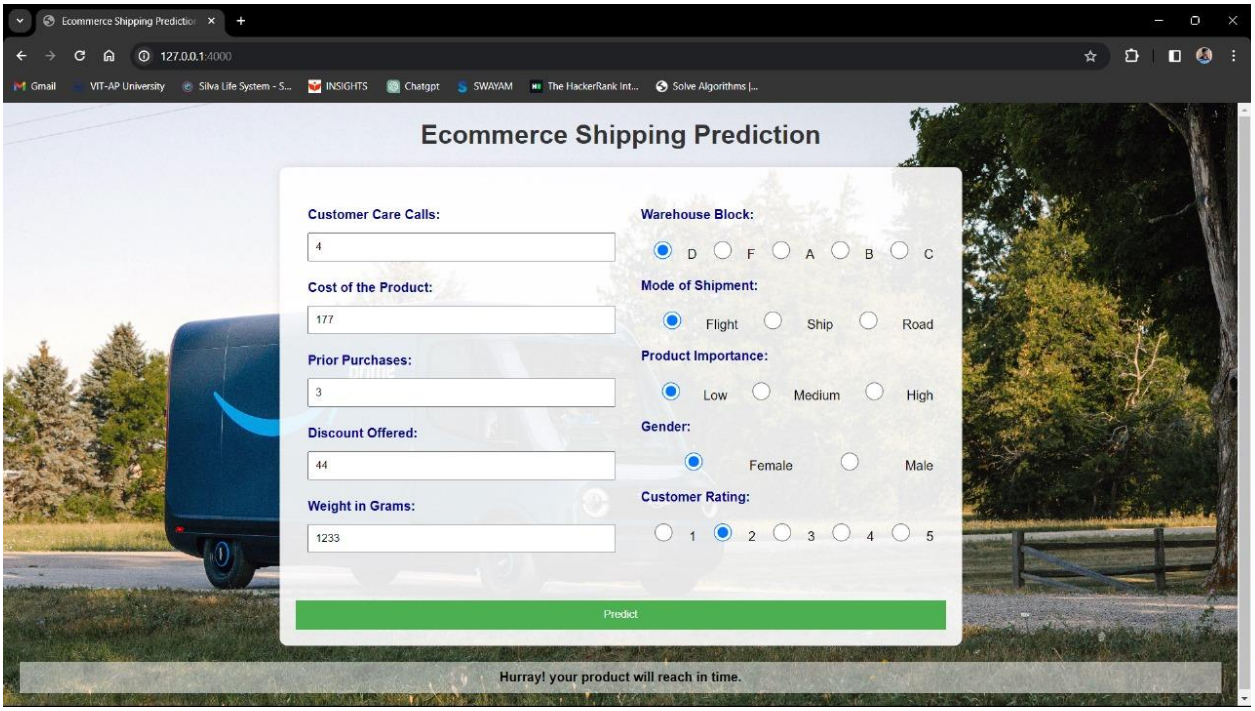
Warehouse Block: ○ D ○ F ○ A ○ B ○ C

Mode of Shipment: ○ Flight ○ Ship ○ Road

Product Importance: ○ Low ○ Medium ○ High

Gender: ○ Female ○ Male

Customer Rating: ○ 1 ○ 2 ○ 3 ○ 4 ○ 5



Ecommerce Shipping Prediction

Customer Care Calls: 4

Cost of the Product: 177

Prior Purchases: 3

Discount Offered: 44

Weight in Grams: 1233

Warehouse Block: ● D ○ F ○ A ○ B ○ C

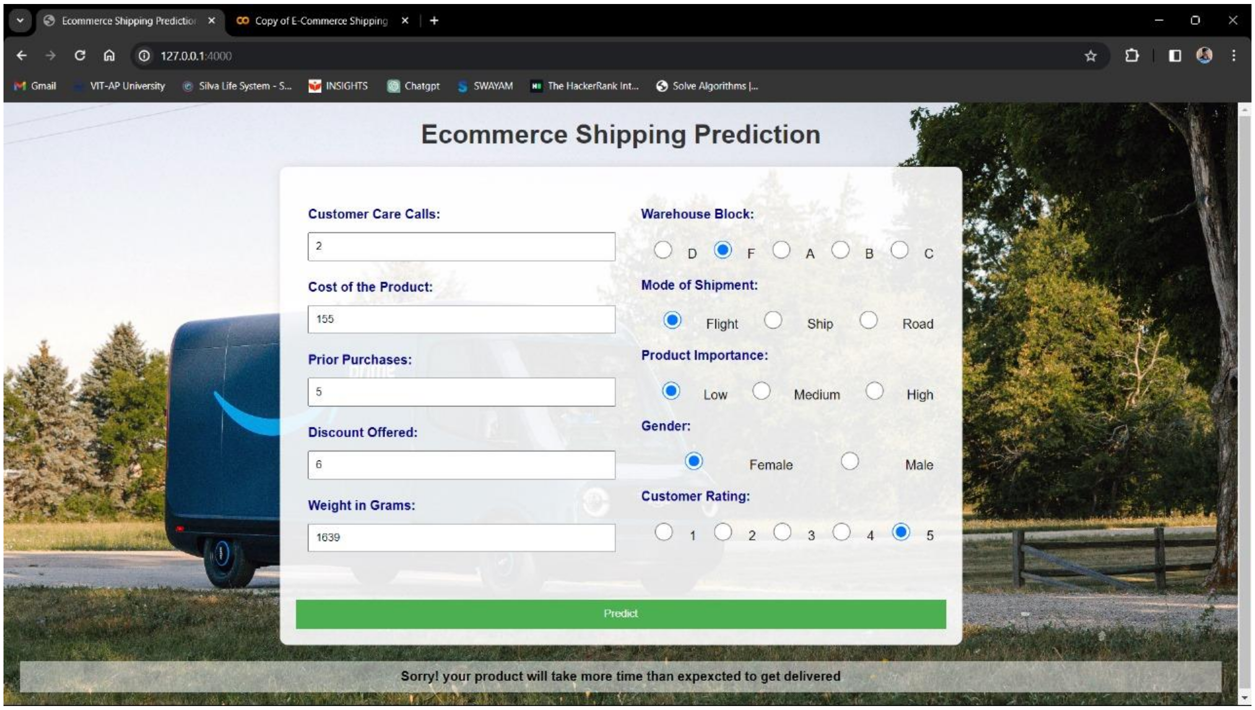
Mode of Shipment: ● Flight ○ Ship ○ Road

Product Importance: ● Low ○ Medium ○ High

Gender: ● Female ○ Male

Customer Rating: ○ 1 ● 2 ○ 3 ○ 4 ○ 5

Hurray! your product will reach in time.



Ecommerce Shipping Prediction

Customer Care Calls: 2

Cost of the Product: 155

Prior Purchases: 5

Discount Offered: 6

Weight in Grams: 1639

Warehouse Block: ○ D ● F ○ A ○ B ○ C

Mode of Shipment: ● Flight ○ Ship ○ Road

Product Importance: ● Low ○ Medium ○ High

Gender: ● Female ○ Male

Customer Rating: ○ 1 ○ 2 ○ 3 ○ 4 ● 5

Sorry! your product will take more time than expexcted to get delivered