

Team ID : SWTID1720196555

Team Leader : PATHAKUNTLA NARENDAR REDDY

Team member : MITTA UDAY SRINADH

Team member : Gummadi Jyotshna

Team member : Mallipeddi Harshitha

Ecommerce Shipping Prediction Using Machine Learning

Category: Machine Learning Skills Required: Python, Machine

Learning, Numpy, Pandas, Matplotlib, Seaborn, HTML, Python-Flask Project Description:

Ecommerce shipping prediction is the process of estimating whether the product reached on time. which is based on various factors such as the origin and destination of the package, the shipping method selected by the customer, the carrier used for shipping, and any potential delays or issues that may arise during the shipping process. Machine learning models can be used to make accurate predictions about shipping times based on historical data and real-time updates from carriers. These models may take into account factors such as weather conditions, traffic, and other external factors that can impact delivery times. Overall Ecommerce shipping prediction is an important tool for ecommerce businesses that want to provide accurate delivery estimates to their customers and improve their overall customer experience.

Project Flow:

1. User interacts with the UI to enter the input.
2. Entered input is analysed by the model which is integrated.
3. Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

4. Define Problem / Problem Understanding










a. Specify the business problem

- Business requirements
- Literature Survey
- Social or Business

Impact.

- Data Collection & Preparation

- Collect the dataset
- Data Preparation
- 5. Exploratory Data Analysis
 - a. Descriptive statistical
- Visual Analysis
- 6. Model Building
 - a. Training the model in multiple algorithms
- Testing the model
- 7. Performance Testing & Hyperparameter Tuning
 - a. Testing model with multiple evaluation metrics
- Comparing model accuracy before & after applying hyperparameter tuning
- 8. Model Deployment
 - a. Save the best model
- Integrate with Web Framework
- 9. Project Demonstration & Documentation
 - a. Record explanation Video for project end to end solution
- Project Documentation-Step by step project development procedure

Name	Date Modified
>  .ipynb_checkpoints	7/9/2024 3:48 PM
>  static	7/9/2024 4:25 PM
>  templates	7/10/2024 1:45 PM
 app.py	7/12/2024 6:35 PM
 Ecommerce Shipping Prediction.ipynb	7/9/2024 3:54 PM
 normalizer.pkl	7/9/2024 8:44 PM
 result.html	7/10/2024 11:30 AM
 rf_acc_67.pkl	7/9/2024 8:44 PM
 Train.csv	7/5/2024 10:02 PM

Literature Survey (Student Will Write)

A literature survey for a Ecommerce Shipping Prediction project would involve researching and reviewing existing studies, articles, and other publications on the topic of Ecommerce Shipping Prediction. The survey would aim to gather information on current systems, their strengths and weaknesses, and any gaps in knowledge that the project could address. The literature survey would also look at the methods and techniques used in previous projects, and any relevant data or findings that could inform the design and implementation of the current project.

Activity 4: Social or Business Impact.

Social Impacts:

1. Improved Customer Experience: Providing accurate delivery estimates can help improve the overall customer experience by reducing uncertainty and increasing transparency.
2. Reduced Environmental Impact: By accurately estimating delivery times, businesses can optimize their logistics and reduce the number of unnecessary trips and emissions from delivery vehicles.
3. Reduced Stress for Delivery Workers: By optimizing delivery routes and times, businesses can reduce the workload and stress on delivery workers, leading to a better work environment and potentially reducing turnover.

Business Impacts:

1. **Increased Sales:** Accurate delivery estimates can help increase customer confidence and reduce cart abandonment rates, leading to increased sales and revenue.
2. **Improved Operational Efficiency:** By optimizing delivery routes and times, businesses can reduce costs associated with transportation, labor, and inventory management.
3. **Competitive Advantage:** Implementing a delivery estimation predictor using machine learning can provide a competitive advantage over other ecommerce businesses that do not offer accurate and transparent delivery estimates.
4. **Better Data-Driven Decision Making:** By analyzing delivery data and performance metrics, businesses can make data-driven decisions to optimize their logistics and improve their overall delivery performance.

Brand Loyalty: Providing accurate delivery estimates and real-time updates can help build brand loyalty by increasing trust and confidence in the business.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the dataset.

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: <https://www.kaggle.com/datasets/prachi13/customer-analytics?select=Train.csv>

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image.

```
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.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
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called `read_csv()` to read the dataset. As a parameter we have to give the directory of the csv file.

```
data = pd.read_csv('Train.csv')
```

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

Handling missing values

Handling categorical data

Handling Outliers

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Handling missing values

Let's find the shape of our dataset first. To find the shape of our data, the df.shape method is used. To find the data type, df.info() function is used

```
# Display the shape of the dataset  
data.shape
```

```
(10999, 12)
```

```
data.info()
```

```
<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
```

For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
data.isnull().any()
```

ID	False
Warehouse_block	False
Mode_of_Shipment	False
Customer_care_calls	False
Customer_rating	False
Cost_of_the_Product	False
Prior_purchases	False
Product_importance	False
Gender	False
Discount_offered	False
Weight_in_gms	False
Reached.on.Time_Y.N	False
dtype:	bool

```
data.isnull().sum()
```

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

Activity 2.2: Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using manual encoding with the help of list comprehension.

In our project, categorical features are

- Warehouse_block
- Mode_of_shipment

- Product_importance
- Gender.

With list comprehension encoding is done.

```
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
```

```
{'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}}
```

Activity 2.3: Handling Outliers in Data

With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound of numerical features with some mathematical formula.

From the below diagram, we could visualize that Discount_offered, Prior_purchases features have outliers. Boxplot from matplotlib library is used here.

```
import matplotlib.pyplot as plt

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

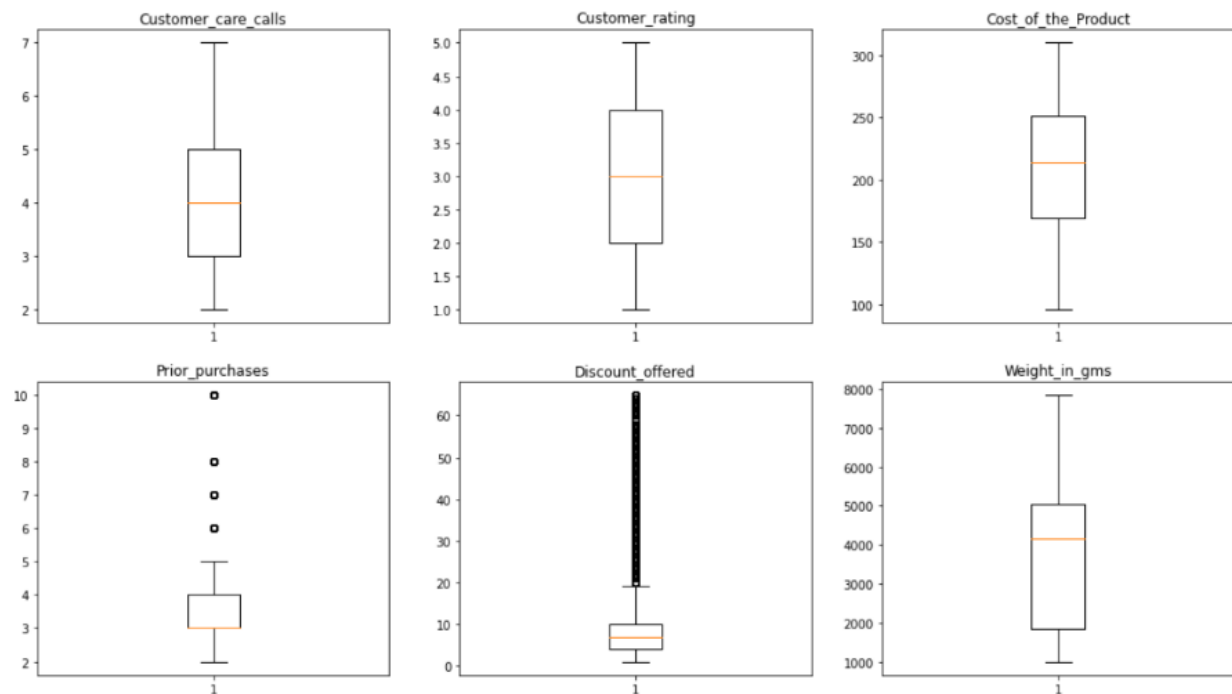
plt.show()
```



```

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()

```



To find upper bound we have to multiply IQR (Interquartile range) with 1.5 and add it with 3rd quantile. To find lower bound instead of adding, subtract it with 1st quantile. Take image attached below as your reference.

```

def find_bounds(arr):
    Q1 = np.percentile(arr, 25)
    Q3 = np.percentile(arr, 75)
    IQR = Q3 - Q1

    upper = Q3 + 1.5 * IQR
    lower = Q1 - 1.5 * IQR

    return lower, upper

# Finding bounds for Discount_offered and Prior_purchases
discount_bounds = find_bounds(data['Discount_offered'])
prior_purchases_bounds = find_bounds(data['Prior_purchases'])

print(f"Discount Offered - Lower Bound: {discount_bounds[0]}, Upper Bound: {discount_bounds[1]}")
print(f"Prior Purchases - Lower Bound: {prior_purchases_bounds[0]}, Upper Bound: {prior_purchases_bounds[1]}")

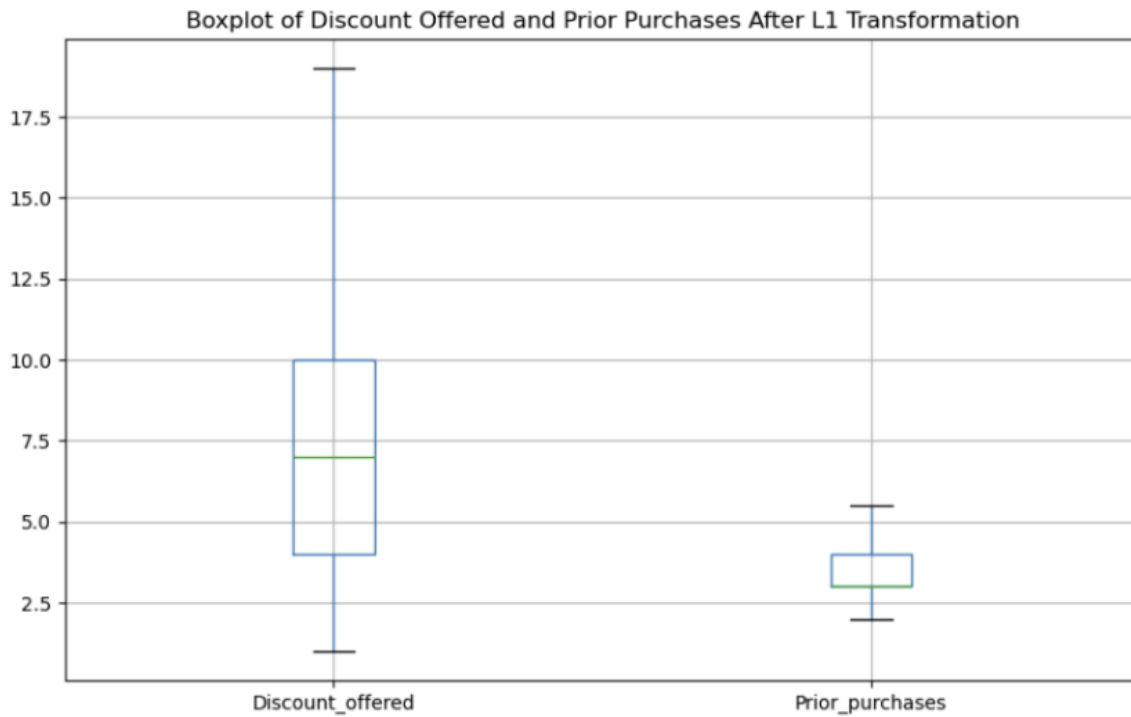
```

Discount Offered - Lower Bound: -5.0, Upper Bound: 19.0
 Prior Purchases - Lower Bound: 1.5, Upper Bound: 5.5

```
def l1_transformation(arr, lower, upper):
    return np.clip(arr, lower, upper)

data['Discount_offered'] = l1_transformation(data['Discount_offered'], discount_bounds[0], discount_bounds[1])
data['Prior_purchases'] = l1_transformation(data['Prior_purchases'], prior_purchases_bounds[0], prior_purchases_bounds[1])

# Verify transformation
plt.figure(figsize=(10, 6))
data.boxplot(column=['Discount_offered', 'Prior_purchases'])
plt.title('Boxplot of Discount Offered and Prior Purchases After L1 Transformation')
plt.show()
```



1. Data splitting

The data was split into train and test variables as shown below using the `train_test_split()` method of scikitlearn module with a `split_size` of 0.20 and a `random_state = 1234`.

```
x_train, x_test, y_train, y_test = train_test_split(
    data.drop(columns=['ID', 'Reached.on.Time_Y.N']),
    data['Reached.on.Time_Y.N'],
    random_state=1234, test_size = 0.20,
    shuffle=True
)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(8799, 10)
(2200, 10)
(8799,)
(2200,)
```

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product_importance	Gender
count	10999.00000	10999.000000	10999.000000	10999.000000	10999.000000	10999.000000	10999.000000	10999.000000	10999.000000
mean	5500.00000	1.833167	0.998454	4.054459	2.990545	210.196836	3.421629	0.604600	0.495861
std	3175.28214	1.343823	0.567099	1.141490	1.413603	48.063272	1.136903	0.641464	0.500000
min	1.00000	0.000000	0.000000	2.000000	1.000000	96.000000	2.000000	0.000000	0.000000
25%	2750.50000	1.000000	1.000000	3.000000	2.000000	169.000000	3.000000	0.000000	0.000000
50%	5500.00000	1.000000	1.000000	4.000000	3.000000	214.000000	3.000000	1.000000	0.000000
75%	8249.50000	3.000000	1.000000	5.000000	4.000000	251.000000	4.000000	1.000000	1.000000
max	10999.00000	4.000000	2.000000	7.000000	5.000000	310.000000	5.500000	2.000000	1.000000

Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to

explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as histplot and countplot.

Seaborn package provides a wonderful function histplot. With the help of histplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.

From the plot we came to know,

Customer Care Calls: Slight positive skewed normal distribution with mode at 4.

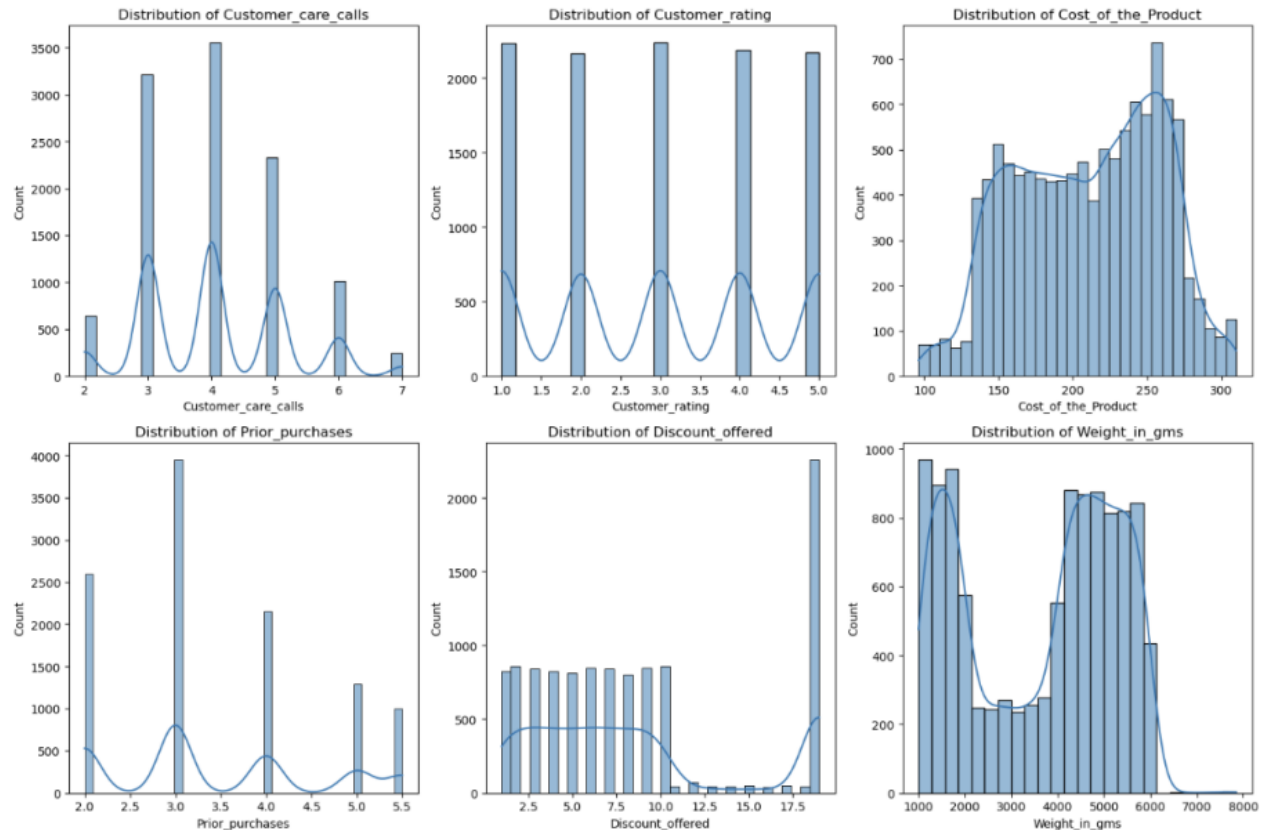
Customer Rating: Uniform distribution.

Cost of the product: 2 picks: smallest around 150, highest around 250.

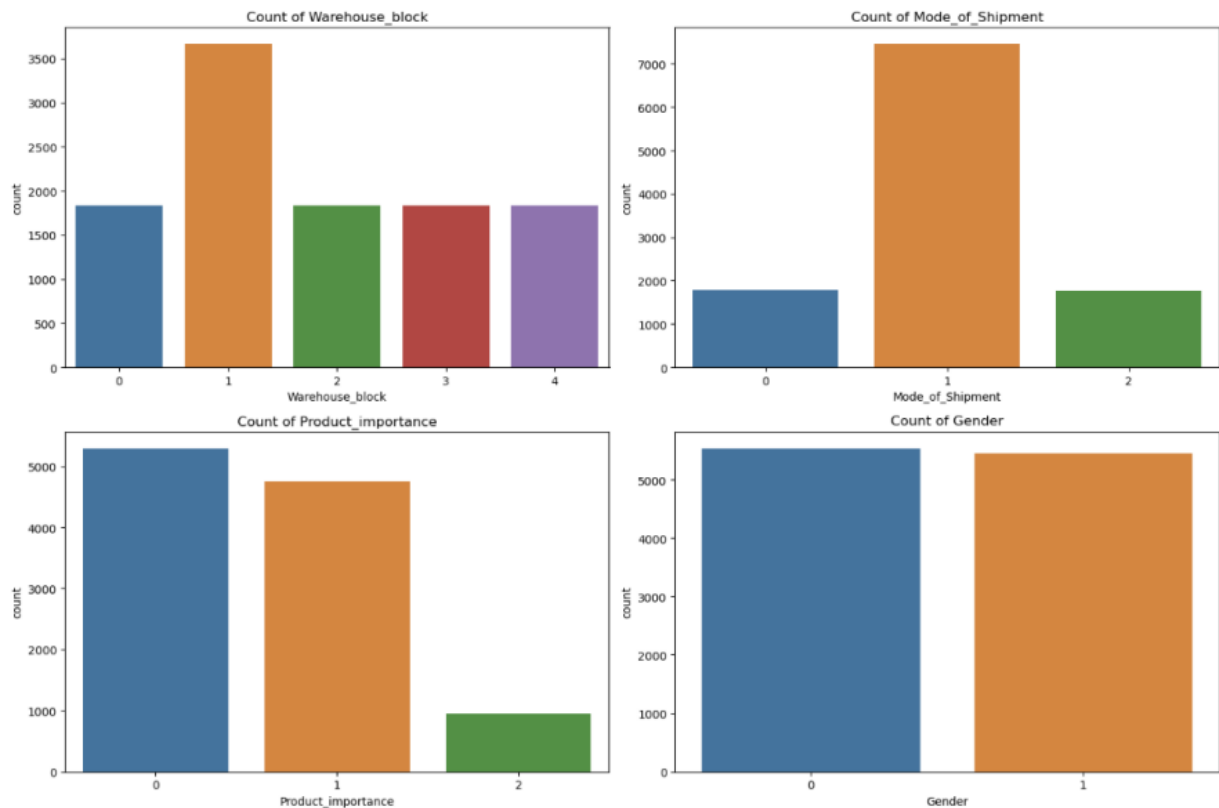
Prior Purchases: Positive skewed normal distribution, mode at 3.

Discount offered: Separated into 2 uniform distributions: 0 to 10 is predominant and then small amount from 10 to 65.

Weight: 3 zones: high from 1000 to 2000 and from 4000 to 6000. Low from 2000 to 4000.



In our dataset we have some categorical features. With the countplot function, we are going to count the unique category in those features. We have created a dummy data frame with categorical features. With for loop and subplot we have plotted this below



Activity 2.2: Bivariate analysis

To find the relation between two features we use bivariate analysis. Here we are visualizing the relationship between drug & BP, drug & sex and drug & cholesterol.

Countplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

From the below plot you can understand that drugA and drugB is not preferred to low and normal BP patients. DrugC is preferred only to low BP patients.

By third graph we can understand, drugC is not preferred to normal cholesterol patients.

Warehouse: Blocks A, B, C, D are equilibrated while block F is predominant (1/2 ratio).

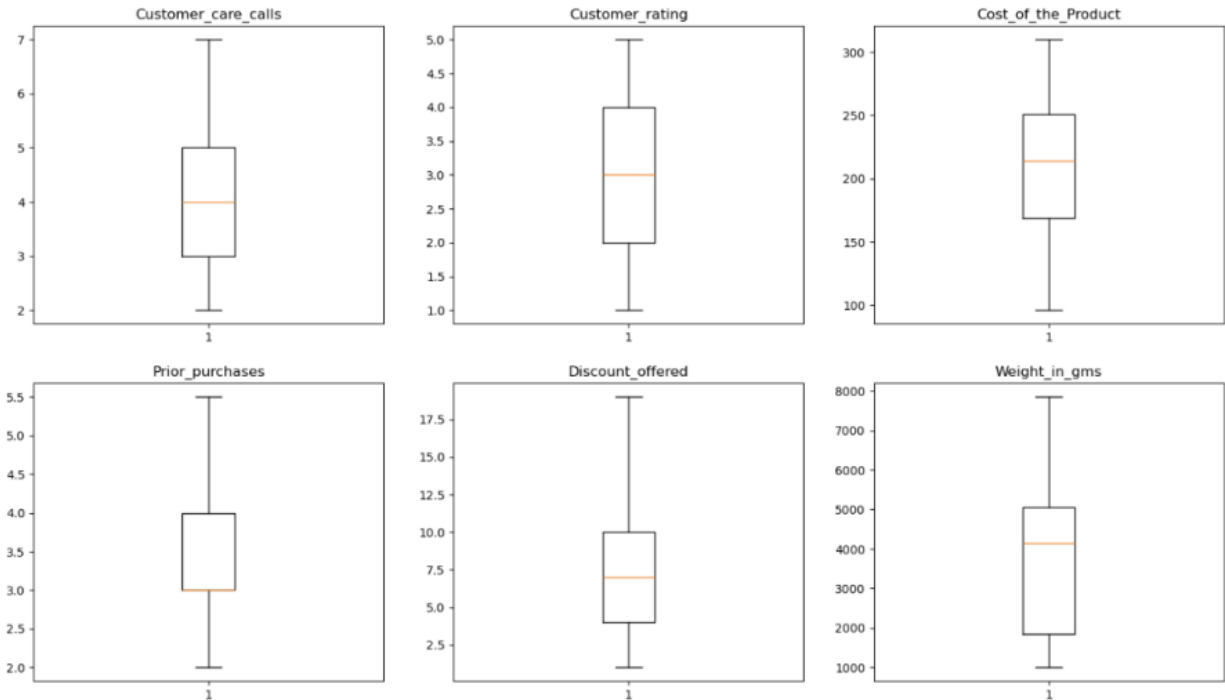
Shipment: Flight and Road have similar observations while Ship is predominant (1/4 ratio).

Importance: There is a majority of low and medium importances and a minority of high importances.

Genders: Both classes are balanced.

From the plots, we observe that very less number of orders are considered as important.

The most common mode of shipment is by Ship and products are packaged from warehouse F. Both genders order the products in a balanced way with orders by Females being slightly higher.



Outliers were found for 2 features (Discount_offered,Weight_in_gms) as visualized above using box plots. To be specific using IQR (inter quartile range) it was observed that, What this means is that there are 1003 and 2263 outliers for Prior_purchases and Discount_offered variables of the dataset.

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

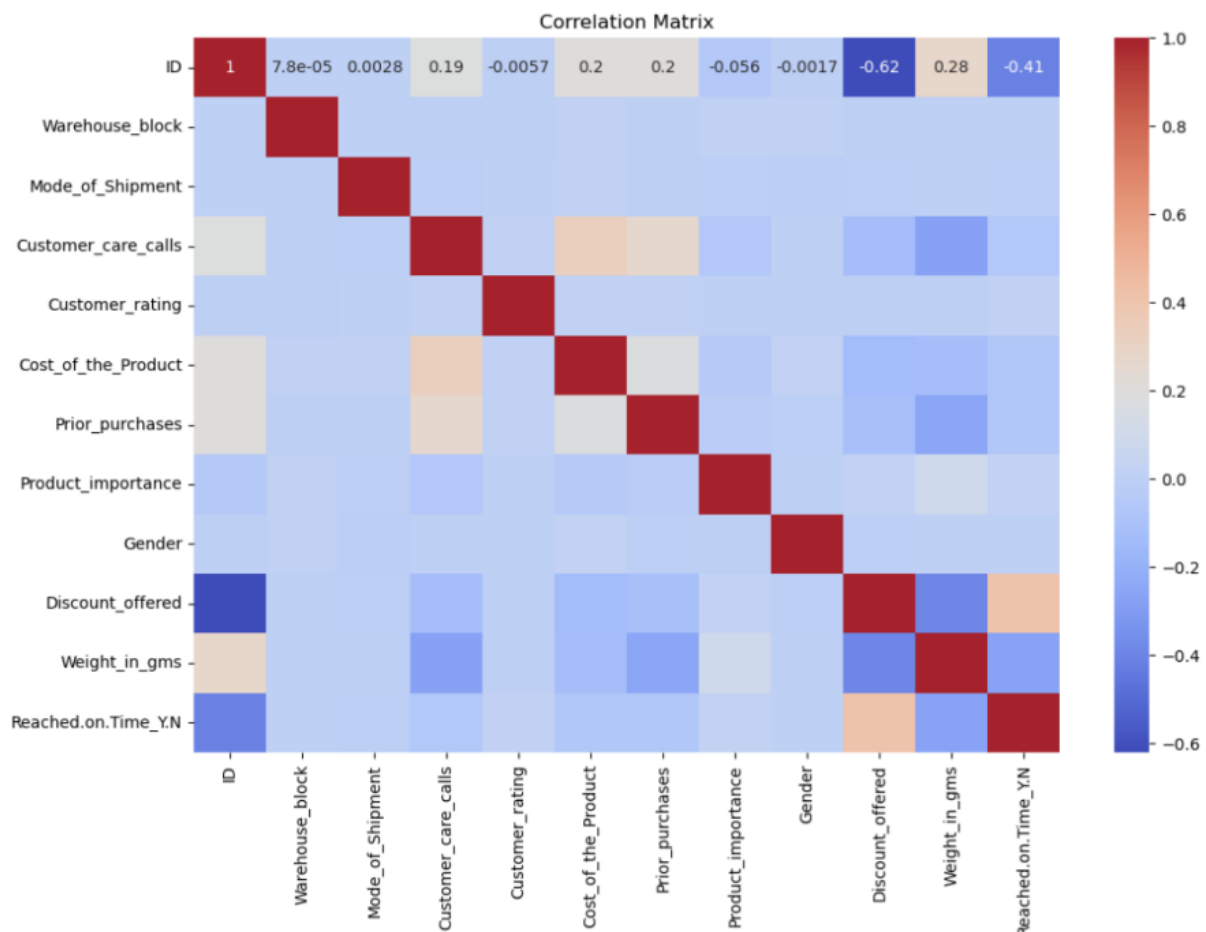
0 are less than the lower bound: -5.0

Activity 2.3: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used heatmap from seaborn package.

From the below image, we came to a conclusion that the product discount is the feature that most highly correlates to if a product is delivered on time and Number of calls and product cost are also highly correlated

among other variables.



Splitting data into train and test

Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the data set.

Here x and y variables are created. On x variable, data is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using `train_test_split()` function from sklearn. As parameters, we are passing x, y, test_size, random_state shuffle.


```
from sklearn.model_selection import train_test_split

X = data.drop(columns=['ID', 'Reached.on.Time_Y.N'])
Y = data['Reached.on.Time_Y.N']
|
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=1234, shuffle=True)
```

```
X_train.shape
```

```
(8799, 10)
```

```
X_test.shape
```

```
(2200, 10)
```

```
Y_train.shape
```

```
(8799,)
```

```
Y_test.shape
```

```
(2200,)
```

Normalization

The data will be normalized using L1 regularisation that will be applied on `x_train` and `x_test` variables separately.

```

from sklearn.preprocessing import Normalizer

# Initialize the normalizer
normalizer = Normalizer(norm='l1')

# Fit and transform the training data
X_train_normalized = normalizer.fit_transform(X_train)
X_test_normalized = normalizer.transform(X_test)

# Convert normalized data back to a DataFrame for ease of use
X_train_normalized = pd.DataFrame(X_train_normalized, columns=X_train.columns)
X_test_normalized = pd.DataFrame(X_test_normalized, columns=X_test.columns)

# Verify the normalization
print(X_train_normalized.describe())
print(X_test_normalized.describe())

```

	Warehouse_block	Mode_of_Shipment	Customer_care_calls \
count	8799.000000	8799.000000	8799.000000
mean	0.000613	0.000332	0.001401
std	0.000619	0.000283	0.001018
min	0.000000	0.000000	0.000249
25%	0.000189	0.000174	0.000685
50%	0.000472	0.000229	0.000938
75%	0.000782	0.000473	0.001904
max	0.003431	0.001699	0.005452

	Customer_rating	Cost_of_the_Product	Prior_purchases \
count	8799.000000	8799.000000	8799.000000
mean	0.000991	0.070023	0.001189
std	0.000775	0.044056	0.000889
min	0.000160	0.016150	0.000319
25%	0.000465	0.038016	0.000543
50%	0.000765	0.051544	0.000807
75%	0.001218	0.092064	0.001650
max	0.004322	0.227820	0.004624

Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying seven classification algorithms. The best model is saved based on its

performance.

Activity 1.1: Writing function to train the models

A function named `models_eval_mm` is created and train, test data are passed as parameters. In the function, logistic regression, logistic regression cv, XGBclassifier, RidgeClassifier, KNN classifier, Random forest classifier and SVC algorithms are initialised and training data is passed to the model with `.fit()` function. Test data is predicted with `predict()` function and saved in a new variable. For evaluating the model, train and test scores are used.

```
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV, RidgeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier

def models_eval_m(X_train, Y_train, X_test, Y_test):
    lg = LogisticRegression(random_state=1234)
    lg.fit(X_train, Y_train)
    Y_pred = lg.predict(X_test)
    print(' -- Logistic Regression')
    print('Train Score: ', lg.score(X_train, Y_train))
    print('Test Score: ', lg.score(X_test, Y_test))
    print()

    lcv = LogisticRegressionCV(random_state=1234)
    lcv.fit(X_train, Y_train)
    Y_pred = lcv.predict(X_test)
    print(' -- Logistic Regression CV')
    print('Train Score: ', lcv.score(X_train, Y_train))
    print('Test Score: ', lcv.score(X_test, Y_test))
    print()

    xgb = XGBClassifier(random_state=1234)
    xgb.fit(X_train, Y_train)
    Y_pred = xgb.predict(X_test)
    print(' -- XGBoost')
    print('Train Score: ', xgb.score(X_train, Y_train))
    print('Test Score: ', xgb.score(X_test, Y_test))
    print()

    rg = RidgeClassifier(random_state=1234)
    rg.fit(X_train, Y_train)
    Y_pred = rg.predict(X_test)
    print(' -- Ridge Classifier')
    print('Train Score: ', rg.score(X_train, Y_train))
    print('Test Score: ', rg.score(X_test, Y_test))
    print()
```

```

knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
print(' -- KNN')
print('Train Score: ', knn.score(X_train, Y_train))
print('Test Score: ', knn.score(X_test, Y_test))
print()

rf = RandomForestClassifier(random_state=1234)
rf.fit(X_train, Y_train)
Y_pred = rf.predict(X_test)
print(' -- Random Forest')
print('Train Score: ', rf.score(X_train, Y_train))
print('Test Score: ', rf.score(X_test, Y_test))
print()

svc = SVC(random_state=1234)
svc.fit(X_train, Y_train)
Y_pred = svc.predict(X_test)
print(' -- SVM classifier')
print('Train Score: ', svc.score(X_train, Y_train))
print('Test Score: ', svc.score(X_test, Y_test))
print()

return lg, lcv, xgb, rg, knn, rf, svc

# Example usage:
# Replace 'x_train', 'y_train', 'x_test', 'y_test' with your actual data variables
# models_eval_m(x_train, y_train, x_test, y_test)

```

Activity 1.2: Calling the function

The function is called by passing the train, test variables. The models are returned and stored in variables as shown below. Clearly, we can see that the models are not performing well on the data. So, we'll optimise the hyperparameters of models using GridsearchCV.

```
lg, lcv, xgb, rg, knn, rf, svc = models_eval_m(X_train_normalized, Y_train, X_test_normalized, Y_test)
```

```
-- Logistic Regression  
Train Score: 0.5976815547221275  
Test Score: 0.5927272727272728
```

```
-- Logistic Regression CV  
Train Score: 0.6329128310035231  
Test Score: 0.6345454545454545
```

```
-- XGBoost  
Train Score: 0.9496533697011024  
Test Score: 0.649090909090909
```

```
-- Ridge Classifier  
Train Score: 0.5976815547221275  
Test Score: 0.5927272727272728
```

```
-- KNN  
Train Score: 0.7787248550971702  
Test Score: 0.6331818181818182
```

```
-- Random Forest  
Train Score: 1.0  
Test Score: 0.6668181818181819
```

```
-- SVM classifier  
Train Score: 0.5976815547221275  
Test Score: 0.5927272727272728
```

Activity 2: Testing the model

Here we have tested with Random forest algorithm. You can test with all algorithm. With the help of predict() function.

```
lg_preds = lg.predict(X_test)
```

```
lg_preds
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Milestone 5: Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-

score.

Activity 1.1: Compare the model

For comparing the above four models, the eval() function is defined. After calling the function, the results of models are displayed as output. From the trained models random forest is performing well

```
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV, RidgeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
import warnings
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score

warnings.filterwarnings("ignore", category=UserWarning, module="sklearn.metrics")
```

```
model_dict = {
    'LogisticRegression': (LogisticRegression(), {'C': [0.1, 1, 10]}),
    'LogisticRegressionCV': (LogisticRegressionCV(), {'Cs': [1, 10, 100]}),
    'XGBoost': (XGBClassifier(), {'n_estimators': [50, 100], 'learning_rate': [0.01, 0.1]}),
    'Ridge Classifier': (RidgeClassifier(), {'alpha': [0.1, 1, 10]}),
    'Knn': (KNeighborsClassifier(), {'n_neighbors': [3, 5, 7]}),
    'Random Forest': (RandomForestClassifier(), {'n_estimators': [50, 100], 'max_depth': [None, 10, 20]}),
    'Support Vector Classifier': (SVC(), {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']})
}
```

```
def eval_model(name, model, param_grid):
    grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
    grid_search.fit(X_train_normalized, Y_train)
    best_model = grid_search.best_estimator_
    Y_pred = best_model.predict(X_test_normalized)
    result = []
    result.append(name)
    result.append("{: .2f}".format(accuracy_score(Y_test, Y_pred) * 100))
    result.append("{: .2f}".format(f1_score(Y_test, Y_pred, average='weighted') * 100))
    result.append("{: .2f}".format(recall_score(Y_test, Y_pred, average='weighted') * 100))
    result.append("{: .2f}".format(precision_score(Y_test, Y_pred, average='weighted') * 100))
    return result
```

```

model_eval_info = []

for name, (model, param_grid) in model_dict.items():
    model_eval_info.append(eval_model(name, model, param_grid))

model_eval_info_df = pd.DataFrame(model_eval_info, columns=['Name', 'Accuracy', 'f1_score', 'Recall', 'Precision'])
model_eval_info_df.to_csv('model_eval.csv', index=False)

print(model_eval_info_df)

```

	Name	Accuracy	f1_score	Recall	Precision
0	LogisticRegression	59.27	44.12	59.27	35.13
1	LogisticRegressionCV	63.45	63.61	63.45	63.84
2	XGBoost	67.68	67.74	67.68	72.06
3	Ridge Classifier	60.95	52.94	60.95	60.20
4	Knn	64.09	64.32	64.09	64.80
5	Random Forest	68.00	67.94	68.00	73.25
6	Support Vector Classifier	65.86	66.12	65.86	66.83

Save The Best Model

Saving the best model and normalizer after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
import pickle as pkl
```

```
pkl.dump(rf, open('rf_acc_67.pkl', 'wb'))
```

```
pkl.dump(normalizer, open('normalizer.pkl', 'wb'))
```

Integrate With Web Framework

In this section, we will be building a web application that is integrated to the model we built. An UI is provided for the users where he has to enter the values for predictions. The entered values are given to the saved model and prediction is showcased on the UI. This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the web application

Building Html Pages:

For this project create two HTML files namely

- index.html
- result.html

and save them in the templates folder.

It is not necessary to follow the exact format as above so feel free to use whatever templates or format you like. Be creative!

Build Python code:

Import the libraries

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

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (`__name__`) as argument.

```
app = Flask(__name__, static_url_path='', static_folder='.')

model = pickle.load(open("rf_acc_67.pkl", "rb"))
data_normalizer = pickle.load(open("normalizer.pkl", "rb"))
```

Render HTML page:

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

@app.route('/result.html')
def result():
    return app.send_static_file('result.html')
```



```

@app.route('/predict', methods=['POST'])
def predict():
    try:
        data = request.get_json()

        Warehouse_block = int(data["Warehouse_block"])
        Mode_of_Shipment = int(data["Mode_of_Shipment"])
        Customer_care_calls = int(data["Customer_care_calls"])
        Customer_rating = int(data["Customer_rating"])
        Cost_of_the_Product = int(data["Cost_of_the_Product"])
        Prior_purchases = int(data["Prior_purchases"])
        Product_importance = int(data["Product_importance"])
        Gender = int(data["Gender"])
        Discount_offered = int(data["Discount_offered"])
        Weight_in_gms = int(data["Weight_in_gms"])

```

```

        preds = [[Warehouse_block, Mode_of_Shipment, Customer_care_calls, Cu
                    Prior_purchases, Product_importance, Gender, Discount_offe
        normalized_preds = data_normalizer.transform(preds)

        prediction = model.predict(normalized_preds)
        probability = model.predict_proba(normalized_preds)[0][1]

        return jsonify({
            'prediction': int(prediction[0]),
            'probability': round(probability * 100, 2)
        })
    except Exception as e:
        return jsonify({'error': str(e)})

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

```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

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

Run the web application

Open anaconda prompt from the start menu

Navigate to the folder where your python script is.

Now type “python app.py” command

Navigate to the localhost where you can view your web page.

Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
Python 3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023, 18:05:47) [MSC v.1916 64 bit (AMD64)]  
Type "copyright", "credits" or "license()" for more  
  
Python 8.20.0 -- An enhanced Interactive Python.  
  
In [1]: runfile('C:/Users/Narendar Reddy/Desktop/project/app.py', wdir='C:/Users/Narendar Reddy/Desktop/project')  
* Serving Flask app 'app'  
* Debug mode: off  
C:\Users\Narendar Reddy\anaconda3\Lib\site-packages\sklearn\base.py:318: UserWarning: Trying to unpickle estimator  
DecisionTreeClassifier from version 1.1.2 when using version 1.2.2. This might lead to breaking code or invalid results. Use at  
your own risk. For more info please refer to:  
https://scikit-learn.org/stable/model\_persistence.html#security-maintainability-limitations  
warnings.warn(  
C:\Users\Narendar Reddy\anaconda3\Lib\site-packages\sklearn\base.py:318: UserWarning: Trying to unpickle estimator  
RandomForestClassifier from version 1.1.2 when using version 1.2.2. This might lead to breaking code or invalid results. Use at  
your own risk. For more info please refer to:  
https://scikit-learn.org/stable/model\_persistence.html#security-maintainability-limitations  
warnings.warn(  
C:\Users\Narendar Reddy\anaconda3\Lib\site-packages\sklearn\base.py:318: UserWarning: Trying to unpickle estimator Normalizer  
from version 1.1.2 when using version 1.2.2. This might lead to breaking code or invalid results. Use at your own risk. For mor  
info please refer to:  
https://scikit-learn.org/stable/model\_persistence.html#security-maintainability-limitations  
warnings.warn(  
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
* Running on http://127.0.0.1:4000  
Press CTRL+C to quit
```

Name	Date Modified
> .ipynb_checkpoints	7/9/2024 3:48 PM
> static	7/9/2024 4:25 PM
> templates	7/10/2024 1:45 PM
app.py	7/12/2024 6:35 PM
Ecommerce Shipping Prediction.ipynb	7/9/2024 3:54 PM
normalizer.pkl	7/9/2024 8:44 PM
result.html	7/10/2024 11:30 AM
rf_acc_67.pkl	7/9/2024 8:44 PM
Train.csv	7/5/2024 10:02 PM

XPRESS ECOM

Home

Shop

Categories

Cart

Contact

Login

Warehouse Block

A

Mode of Shipment

Flight

Customer Care Calls

Customer Rating

Cost of the Product

Prior Purchases

Product Importance

low

Gender

Male

Discount Offered

Weight in gms

Submit and Proceed



Prediction Result

There is a 91.17% chance that your product will reach in time.