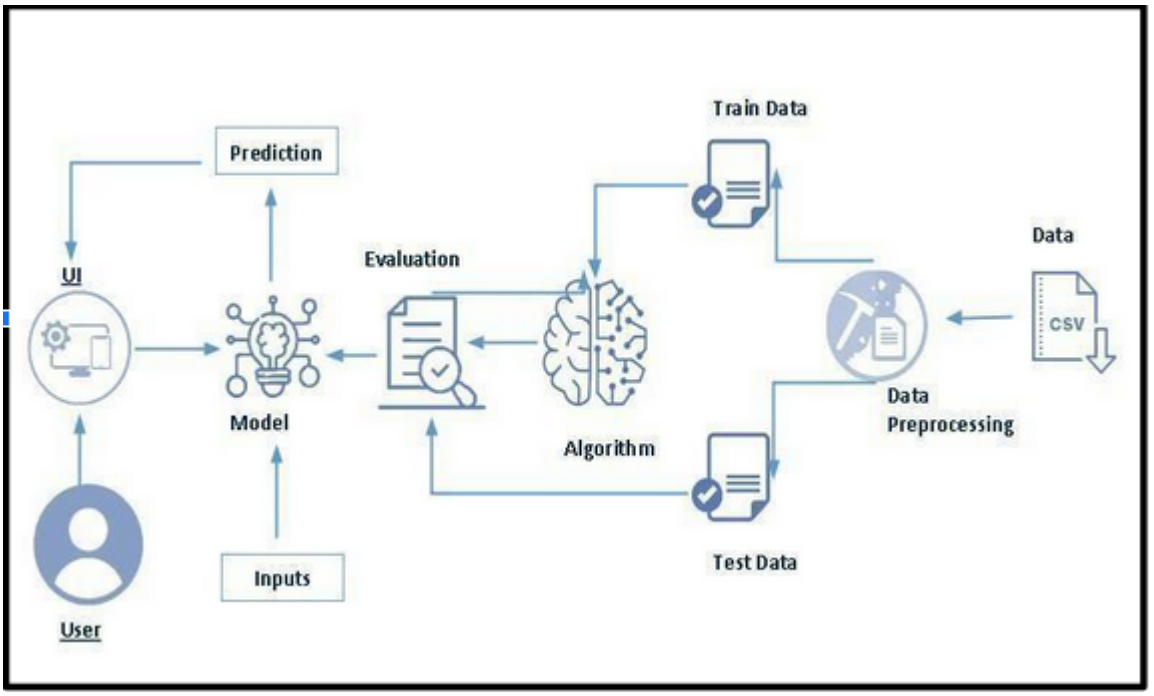
ECOMMERCE SHIPPING PREDICTION USING MACHINE LEARNING

Project Hand-out, Faculty Development Program – NaanMudhalvan

SmartInternz
www.smartinternz.com

Ecommerce Shipping Prediction Using Machine Learning

Ecommerce shipping prediction is the process of estimating the 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. Over All 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:

- 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
    - 0 Specify the business problem
    - Business requirements
    - Literature Survey
    - Social or Business Impact.
- Data Collection & Preparation
    - Collect the dataset
    - Data Preparation
- Exploratory Data Analysis
    - 0 Descriptive statistical
    - Visual Analysis
- Model Building
    - 0 Training the model in multiple algorithms
    - Testing the model
- Performance Testing & Hyperparameter Tuning
    - 0 Testing model with multiple evaluation metrics
    - Comparing model accuracy before & after applying hyperparameter tuning
- Model Deployment
    - 0 Save the best model
    - Integrate with Web Framework
- Project Demonstration & Documentation
    - 0 Record explanation Video for project end to end solution
    - Project Documentation-Step by step project development procedure

Prior Knowledge:

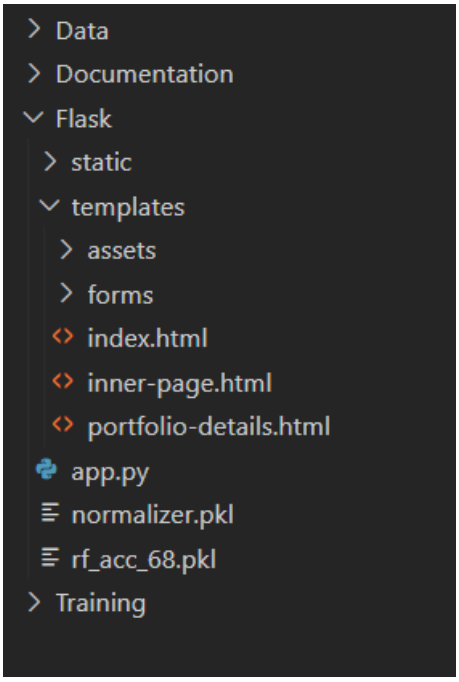
You must have prior knowledge of following topics to complete this project.

ML Concepts

- Supervised learning: https://www.javatpoint.com/supervised-machine-learning
- Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
- Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classificationalgorithm
- 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/
- Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-modelevaluation-error-metrics/
- Flask Basics : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create the Project folder which contains files as shown below



- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- Static folder and assets folder contain the CSS and JavaScript files along with images.
- rf_acc_68.pkl and normalizer.pkl are our saved models. Further we will use these models for flask integration.
- Training folder contains a model training file that are jupyter notebooks.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Refer Project Description

Activity 2: Business requirements

Here are some potential business requirements for an ecommerce product delivery estimation predictor using machine learning:

1. Accurate Delivery Estimates: The primary goal of the system should be to provide accurate delivery estimates to customers. The estimated delivery time should consider factors such as distance, traffic, weather, and any other relevant variables.
2. Real-Time Updates: Customers should be able to receive real-time updates on the status of their delivery, including any delays or changes to the estimated delivery time. The system should be able to adjust the estimated delivery time based on the most up-to-date information available.
3. Integration with Ecommerce Platforms: The system should be able to integrate with popular ecommerce platforms, such as Shopify or Magento, to automatically retrieve order information and provide accurate delivery estimates.
4. Machine Learning Models: The system should use machine learning models to predict delivery times based on historical delivery data and other relevant variables. The machine learning models should be trained and optimized to improve accuracy over time.
5. Scalability: The system should be scalable to handle large volumes of orders and delivery estimates. It should be able to calculate delivery estimates quickly and accurately for a high number of orders at the same time.
6. Reporting and Analytics: The system should provide reporting and analytics capabilities to help ecommerce businesses track delivery performance and identify areas for improvement.
7. Customer Service Integration: The system should be integrated with customer service channels, such as email or chat support, to provide timely updates to customers and handle any delivery-related inquiries or issues.
8. Cost-Effective: The system should be cost-effective to implement and maintain, with reasonable pricing models and minimal upfront costs.

Activity 3: 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.

5. 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(r"F:\(---------PROJECTS--------)\-----------SMARTINTERNZ\Data\Train.csv")
data.head()
```

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product_importance	Gender	Di
0	1	D	Flight	4	2	177	3	low	F	
1	2	F	Flight	4	5	216	2	low	M	
2	3	A	Flight	2	2	183	4	low	M	
3	4	B	Flight	3	3	176	4	medium	M	
4	5	C	Flight	2	2	184	3	medium	F	

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.

Activity 2.1: 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.

```
data.shape
```

```
(10999, 12)
```

```
data.info() #No null vallues 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
```

- 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().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
```

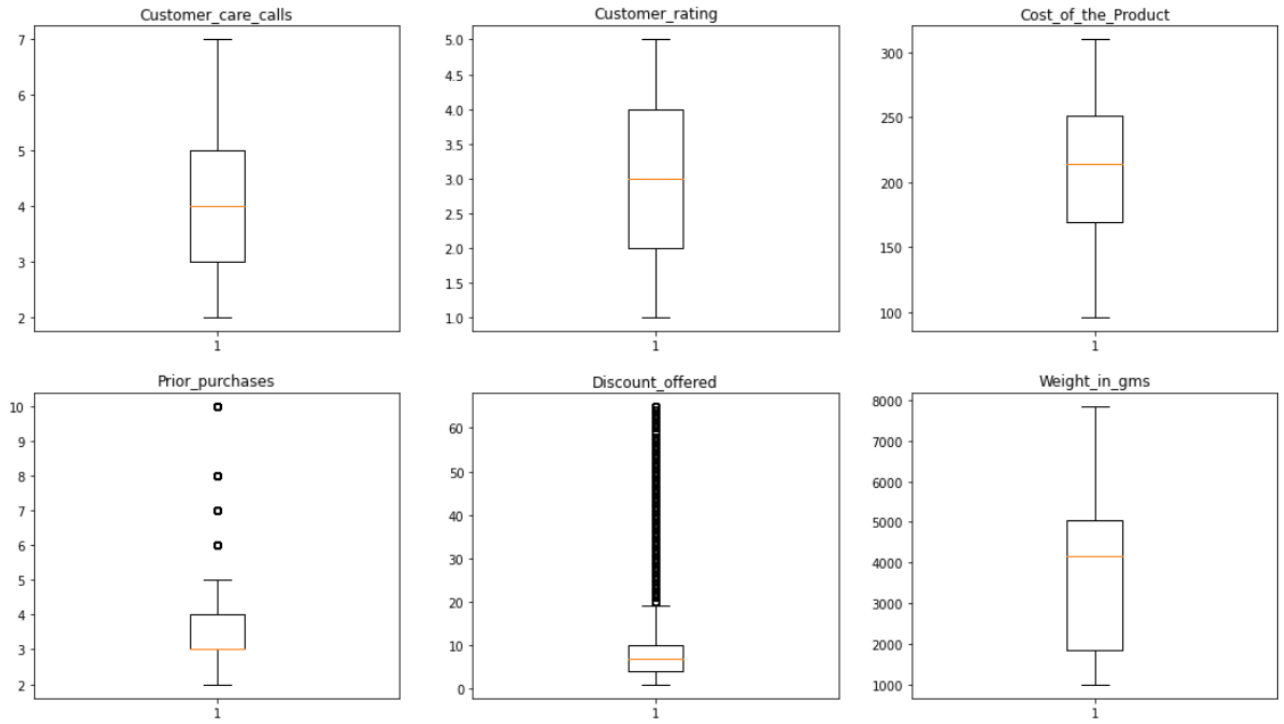
```
{'Warehouse_block': {'D': 0, 'F': 1, 'A': 2, 'B': 3, 'C': 4},
 'Mode_of_Shipment': {'Flight': 0, 'Ship': 1, 'Road': 2},
 'Product_importance': {'low': 0, 'medium': 1, 'high': 2},
 'Gender': {'F': 0, 'M': 1}}
```

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.

```
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 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])
```

```
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
```

- To handle the outliers transformation technique is used. Here L1 transformation is used.

- 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,)
```

- Normalization

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

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.

```
data.describe(include='all')
```

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product_importanc
count	10999.00000	10999.000000	10999.000000	10999.000000	10999.000000	10999.000000	10999.000000	10999.00000
mean	5500.00000	1.833167	0.998454	4.054459	2.990545	210.196836	3.567597	0.60460
std	3175.28214	1.343823	0.567099	1.141490	1.413603	48.063272	1.522860	0.64146
min	1.00000	0.000000	0.000000	2.000000	1.000000	96.000000	2.000000	0.00000
25%	2750.50000	1.000000	1.000000	3.000000	2.000000	169.000000	3.000000	0.00000
50%	5500.00000	1.000000	1.000000	4.000000	3.000000	214.000000	3.000000	1.00000
75%	8249.50000	3.000000	1.000000	5.000000	4.000000	251.000000	4.000000	1.00000
max	10999.00000	4.000000	2.000000	7.000000	5.000000	310.000000	10.000000	2.00000

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.

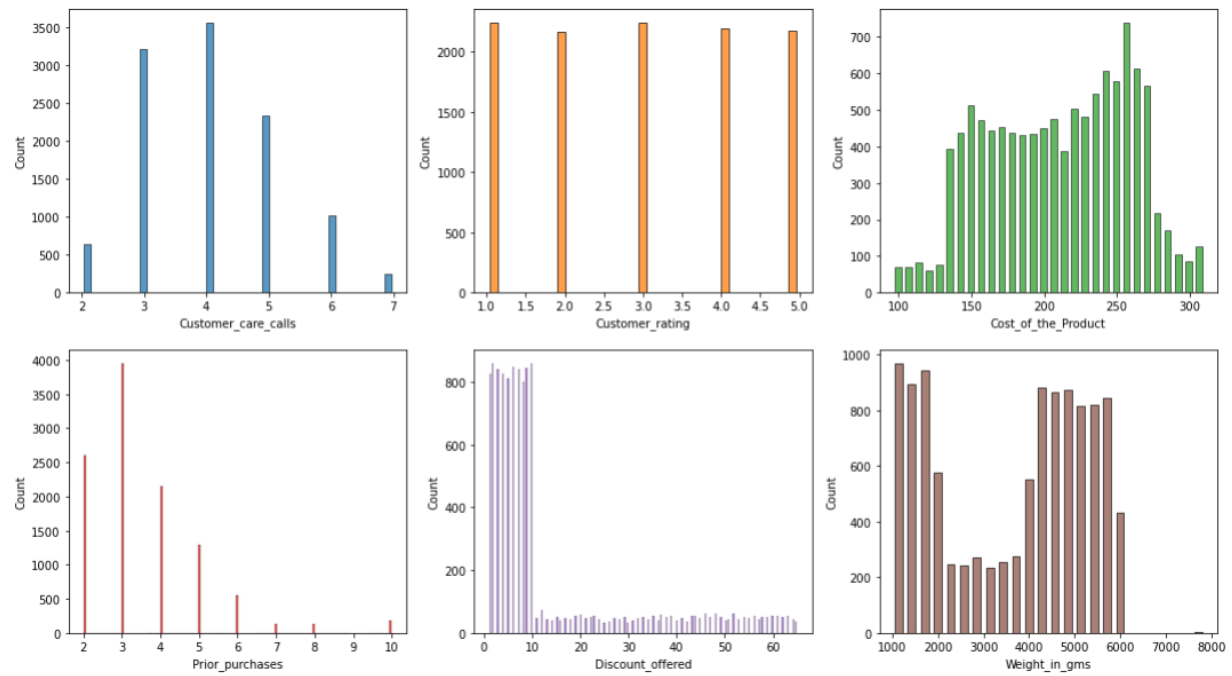
Customer Rating: Uniform distribution.

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

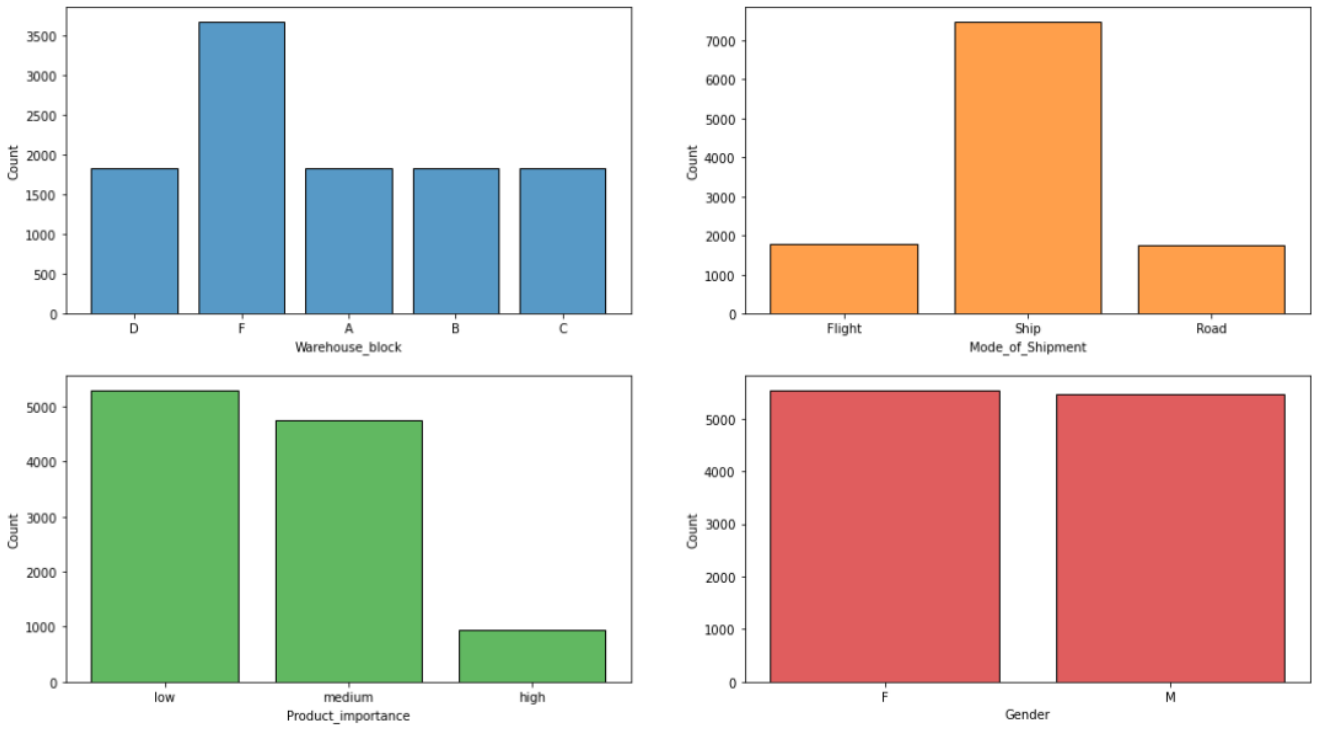
Prior Purchases: Positive skewed normal distribution, mode at 3.

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 graph.



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 predominent (1/2 ratio).
Shipment: Flight and Road have similar observations while Ship is predominent (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,

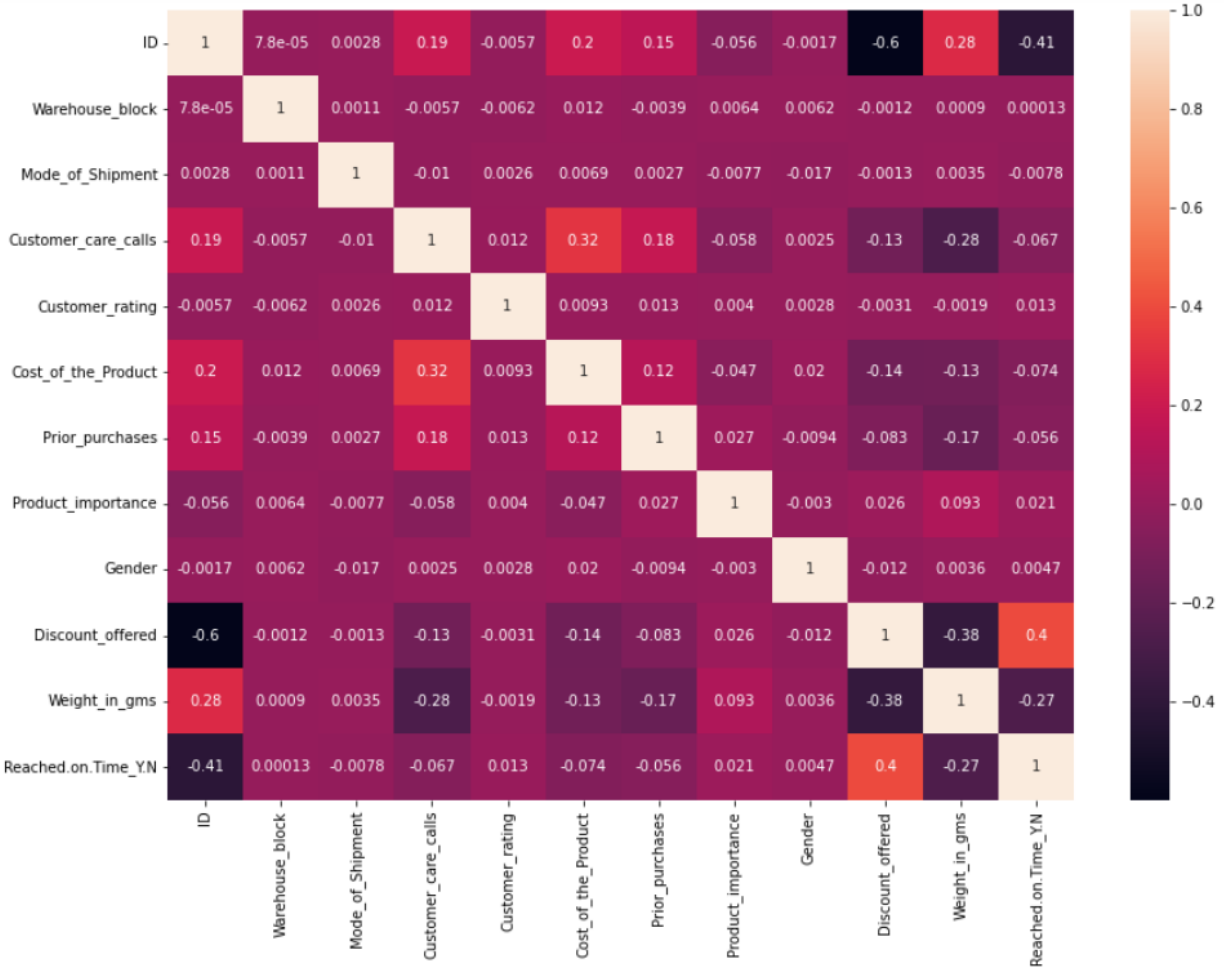
```
Prior_purchases
    1003 are over the upper bound: 5.5
    0 are less than the lower bound: 1.5

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

What this means is that there are 1003 and 2263 outliers for Prior_purchases and Discount_offered variables of the dataset.

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_staten shuffle.

```
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 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:Creating 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.

```
def models_eval_mm(x_train,y_train,x_test,y_test):
    lg = LogisticRegression(random_state=1234)
    lg.fit(x_train,y_train)
    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)
    print('--Logistic Regression CV')
    print('Train Score:',lcv.score(x_train,y_train))
    print('Test Score:',lcv.score(x_test,y_test))
    print()

    print('--XGBoost')
    xgb = XGBClassifier(random_state=1234)
    xgb.fit(x_train,y_train)
    print('Train Score:',xgb.score(x_train,y_train))
    print('Test Score:',xgb.score(x_test,y_test))
    print()

    print('--Ridge Classifier')
    rg = RidgeClassifier(random_state=1234)
    rg.fit(x_train,y_train)
    print('Train Score:',rg.score(x_train,y_train))
    print('Test Score:',rg.score(x_test,y_test))
    print()

    print('--KNN')
    knn = KNeighborsClassifier()
    knn.fit(x_train,y_train)
    print('Train Score:',knn.score(x_train,y_train))
    print('Test Score:',knn.score(x_test,y_test))
    print()

    print('--Random Forest')
    rf = RandomForestClassifier(random_state=1234)
    rf.fit(x_train,y_train)
    print('Train Score:',rf.score(x_train,y_train))
    print('Test Score:',rf.score(x_test,y_test))
    print()

    print('--SVM classifier')
    svc = svm.SVC(random_state=1234)
    svc.fit(x_train,y_train)
    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
```

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_mm(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: 0.6362086600750085
Test Score: 0.6327272727272727

--XGBoost
Train Score: 0.9498806682577565
Test Score: 0.6559090909090909

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

--KNN
Train Score: 0.7797476986021139
Test Score: 0.6327272727272727

--Random Forest
Train Score: 1.0
Test Score: 0.6718181818181819

--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.

```
model.predict(x_test_normalized[0].reshape(1,-1))
```

```
array([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. .

```
def eval(name,model):
    y_pred = 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) * 100))
    result.append("{:.2f}".format(recall_score(y_test, y_pred) * 100))
    result.append("{:.2f}".format(precision_score(y_test, y_pred) * 100))
    return result
```

```
model_list = {
    'logistic regression':lg,
    'logistic regression CV':lcv,
    'XGBoost':xgb,
    'Ridge classifier':rg,
    'KNN':knn,
    'Random Forest':rf,
    'Support Vector Classifier':svc
}
model_eval_info = []

for i in model_list.keys():
    model_eval_info.append(eval(i,model_list[i]))
model_eval_info = pd.DataFrame(model_eval_info,columns=['Name','Accuracy','f1_score','Recall','Precision'])
model_eval_info.to_csv('model_eval.csv')
model_eval_info
```

	Name	Accuracy	f1_score	Recall	Precision
0	logistic regression	59.27	74.43	100.00	59.27
1	logistic regression CV	63.27	66.78	62.27	71.99
2	XGBoost	65.59	69.63	66.56	73.00
3	Ridge classifier	59.27	74.43	100.00	59.27
4	KNN	63.27	68.39	67.02	69.81
5	Random Forest	67.18	70.36	65.72	75.71
6	Support Vector Classifier	59.27	74.43	100.00	59.27

After calling the function, the results of models are displayed as output. From the trained models random forest is performing well

Activity 2: Comparing model accuracy before & after applying hyperparameter tuning (Hyperparameter tuning is optional. For this project it is not required.)

We will perform hyper parameter tuning on models that are better performing which are SVC, Random Forest, XGBoost and Logistic Regression CV. First we'll discover the existing hyperparameters of these models as below.

XGboost

```
xgb._get_params
```

```
<bound method _BaseComposition._get_params of Pipeline(steps=[('standardscaler', StandardScaler()),
                ('xgbclassifier',
                 XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
                               colsample_bylevel=1, colsample_bynode=1,
                               colsample_bytree=1, early_stopping_rounds=None,
                               enable_categorical=False, eval_metric=None,
                               gamma=0, gpu_id=-1, grow_policy='depthwise',
                               importance_type=None, interaction_constraints='',
                               learning_rate=0.300000012, max_bin=256,
                               max_cat_to_onehot=4, max_delta_step=0,
                               max_depth=6, max_leaves=0, min_child_weight=1,
                               missing=nan, monotone_constraints='()',
                               n_estimators=100, n_jobs=0, num_parallel_tree=1,
                               predictor='auto', random_state=0, reg_alpha=0,
                               reg_lambda=1, ...))])>
```

Random Forest

```
rf.get_params()
```

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 1234,
 'verbose': 0,
 'warm_start': False}
```

Logistic Regression CV

```
lcv.get_params()
```

```
{'Cs': 10,
 'class_weight': None,
 'cv': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1.0,
 'l1_ratios': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': 1234,
 'refit': True,
 'scoring': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0}
```

To perform hyper parameter tuning use the following code as shown below.

Hyperparamter optimisation for SVM

```
svc = svm.SVC(random_state=1234)
params = {
    'kernel' : ['poly', 'rbf'],
            'C': [ 10, 13],
             'gamma': [4,5],
    'tol':[1e-1,1e-2,1e-3]
        }
fitmodel = GridSearchCV(svc, param_grid=params, cv=5, refit=True, scoring="accuracy", n_jobs=-1, verbose=3)
fitmodel.fit(x_train_normalized, y_train)
print(fitmodel.best_estimator_, fitmodel.best_params_, fitmodel.best_score_)

# SVC(C=10, gamma=1, random_state=1234) {'C': 10, 'gamma': 1, 'kernel': 'rbf'} 0.6657575326890279
# SVC(C=6, gamma=2, random_state=1234) {'C': 6, 'gamma': 2, 'kernel': 'rbf'} 0.6659845470050132
```

Hyperparameter optimisation for XGboost

```
params = {
        'min_child_weight': [10,20],
        'gamma': [1.5, 2.0, 2.5],
        'colsample_bytree': [0.6, 0.8, 0.9],
        'max_depth': [4,5,6]
        }
xgb = XGBClassifier(learning_rate=0.5, n_estimators=100, objective='binary:logistic', nthread=3)
fitmodel = GridSearchCV(xgb, param_grid=params, cv=5, refit=True, scoring="accuracy", n_jobs=-1, verbose=3)
fitmodel.fit(x_train_normalized, y_train)
print(fitmodel.best_estimator_, fitmodel.best_params_, fitmodel.best_score_)

# {'colsample_bytree': 0.8, 'gamma': 2.0, 'max_depth': 4, 'min_child_weight': 10} 0.6608713628611298
```

Logistic regression hyperparameter optimisation

```
# Plug in appropriate max_depth and random_state parameters
lg = LogisticRegressionCV(n_jobs=-1,random_state= 1234)
lg_param_grid = {
    'Cs': [6,8,10,15,20],
     'max_iter': [60,80,100]
}
lg_cv= GridSearchCV(lg,lg_param_grid,cv=5,scoring="accuracy", n_jobs=-1, verbose=3)
lg_cv.fit(x_train_normalized,y_train)

print("Best Score:" + str(lg_cv.best_score_))
print("Best Parameters: " + str(lg_cv.best_params_))

#Best Score:0.633821644529433
#Best Parameters: {'Cs': 10, 'max_iter': 60}
```

Random Forest Hyperparameter optimisation

```
# Plug in appropriate max_depth and random_state parameters
rf = RandomForestClassifier()
rf_param_grid = {
    'n_estimators': [200,300,500],
    'criterion': ['entropy','gini'],
    'max_depth': [7,8,60,80,100],
    'max_features': ['auto', 'sqrt', 'log2']
}
rf_cv= GridSearchCV(rf,rf_param_grid,cv=7,scoring="accuracy", n_jobs=-1, verbose=3)
rf_cv.fit(x_train_normalized,y_train)

print("Best Score:" + str(rf_cv.best_score_))
print("Best Parameters: " + str(rf_cv.best_params_))

#Best Score:0.6849642004773271
#Best Parameters: {'criterion': 'entropy', 'max_depth': 7, 'max_features': 'auto', 'n_estimators': 200}
```

Model evaluation before hyper parameter tuning

	Name	Accuracy	f1_score	Recall	Precision
0	logistic regression	59.27	74.43	100.00	59.27
1	logistic regression CV	63.27	66.78	62.27	71.99
2	XGBoost	65.59	69.63	66.56	73.00
3	Ridge classifier	59.27	74.43	100.00	59.27
4	KNN	63.27	68.39	67.02	69.81
5	Random Forest	67.18	70.36	65.72	75.71
6	Support Vector Classifier	59.27	74.43	100.00	59.27

Model evaluation after hyper parameter tuning

	Name	Accuracy	f1_score	Recall	Precision
0	logistic regression	59.27	74.43	100.00	59.27
1	logistic regression CV	63.27	66.78	62.27	71.99
2	XGBoost	65.27	68.84	64.72	73.52
3	Ridge classifier	59.27	74.43	100.00	59.27
4	KNN	63.27	68.39	67.02	69.81
5	Random Forest	67.55	66.38	54.06	85.98
6	Support Vector Classifier	67.05	69.06	62.04	77.86

Milestone 6: Model Deployment

Activity 1: 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_68.pkl', 'wb'))
```

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

Activity 2: 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 uses where he has to enter the values for predictions. The enter 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

Activity 2.1: Building Html Pages:

For this project create two HTML files namely

- index.html
- inner-page.html
- portfolio-details.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!

Activity 2.2: Build Python code:

Import the libraries

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

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__)
model = pickle.load(open("rf_acc_68.pkl","rb"))
Data_normalizer = pickle.load(open("normalizer.pkl","rb"))
```

Render HTML page:

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

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```
import pickle
from flask import Flask , request, render_template
app = Flask(__name__)
model = pickle.load(open("rf_acc_68.pkl","rb"))
Data_normalizer = pickle.load(open("normalizer.pkl","rb"))
@app.route('/')
def indput():
    return render_template('index.html')

@app.route('/predict',methods = ['GET','POST'])
def admin():
    Warehouse_block=eval(request.form["Warehouse_block"])
    Mode_of_Shipment=eval(request.form["Mode_of_Shipment"])
    Customer_care_calls=eval(request.form["Customer_care_calls"])
    Customer_rating=eval(request.form["Customer_rating"])
    Cost_of_the_Product = eval(request.form["Cost_of_the_Product"])
    Prior_purchases = eval(request.form["Prior_purchases"])
    Product_importance = eval(request.form["Product_importance"])
    Gender = eval(request.form["Gender"])
    Discount_offered = eval(request.form["Discount_offered"])
    Weight_in_gms = eval(request.form["Weight_in_gms"])

    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]]
    xx=model.predict(Data_normalizer.transform(preds))
    prob=model.predict_proba(Data_normalizer.transform(preds))[0]
    n_reach = prob[0]
    reach = prob[1]
    print('There is a {0:.2f} % chance that your product will reach in time'.format(reach*100))
    print(xx)
    return render_template("index.html",p='There is a {0:.2f} % chance that your product will reach in time'.format(reach*100)
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:

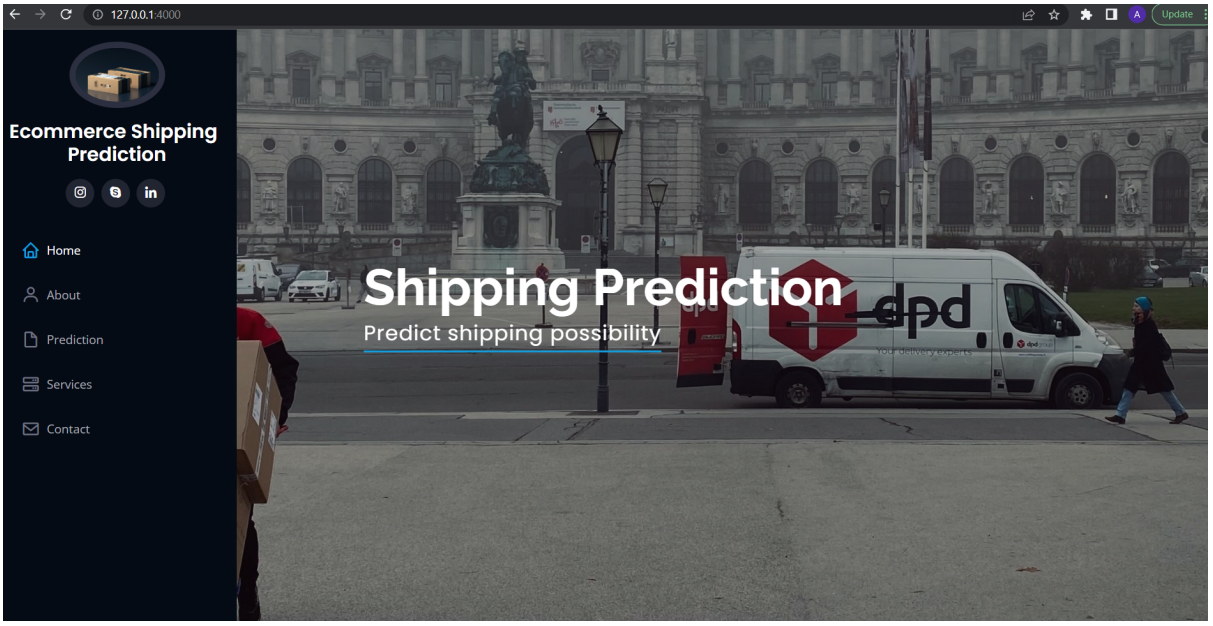
```
if __name__ == '__main__':
    app.run(debug = False, port=4000)
```

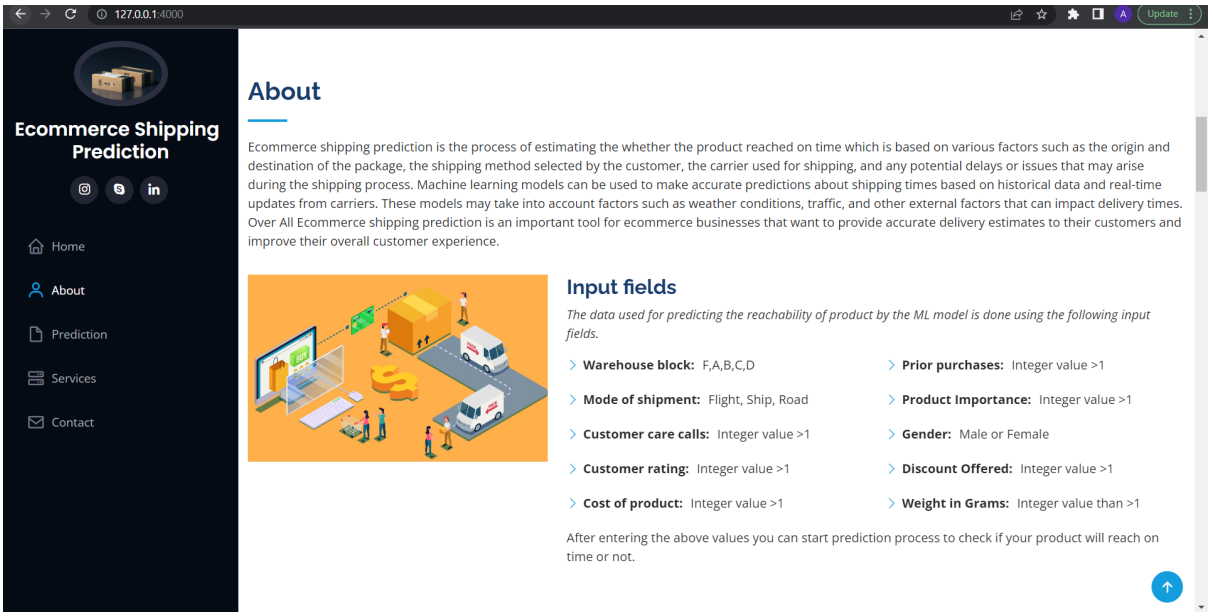
Activity 2.3: Run the web application

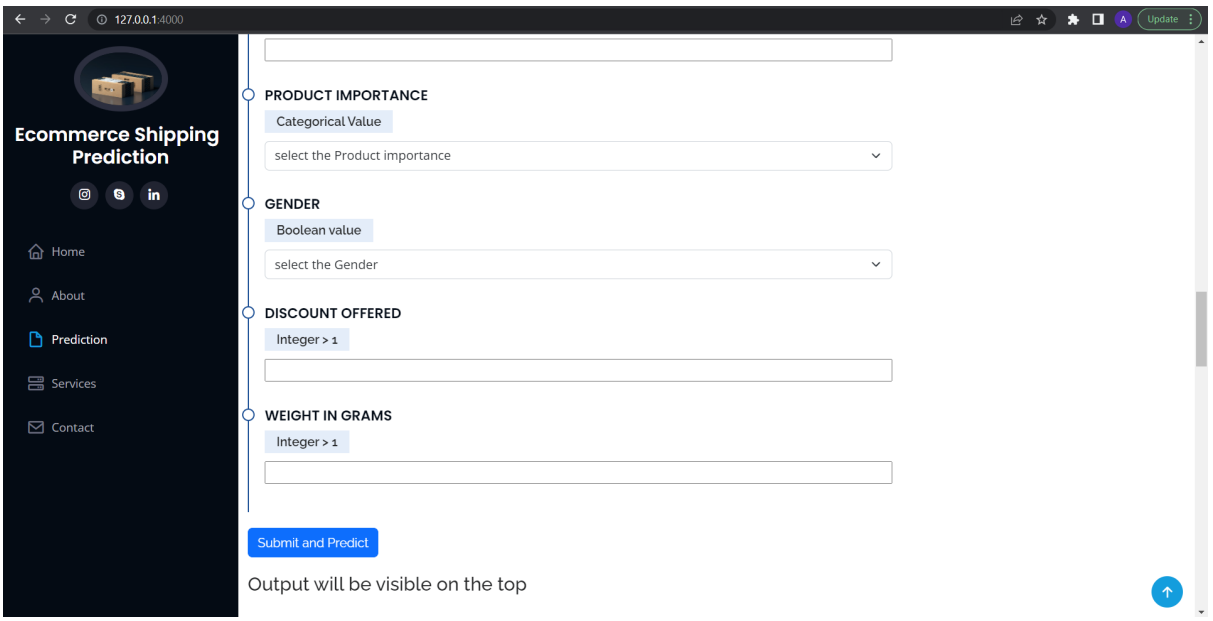
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

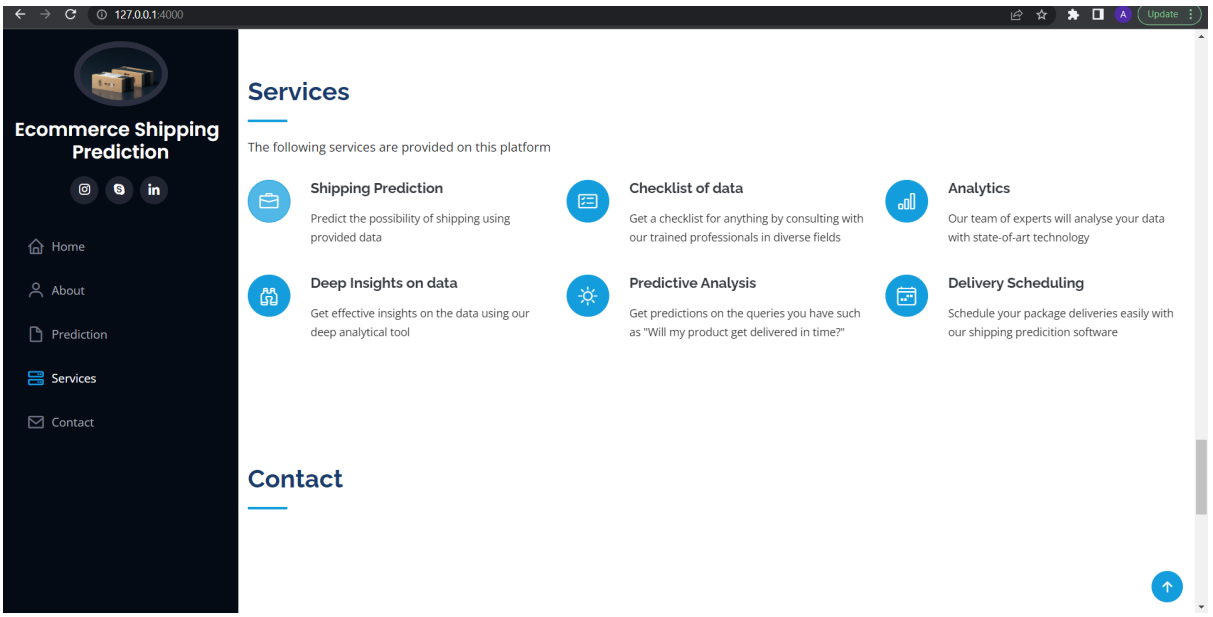
```
F:\(---------PROJECTS--------)\------------SMARTINTERNZ\Flask>python app.py
C:\Users\ashwi\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:288: UserWarning: Trying to unp
ickle estimator DecisionTreeClassifier from version 1.1.2 when using version 1.2.0. This might lead 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\ashwi\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:288: UserWarning: Trying to unp
ickle estimator RandomForestClassifier from version 1.1.2 when using version 1.2.0. 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\ashwi\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:288: UserWarning: Trying to unp
ickle estimator Normalizer from version 1.1.2 when using version 1.2.0. This might lead to breaking code or invalid resu
lts. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
 * Serving Flask app 'app'
 * Debug mode: off
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
```

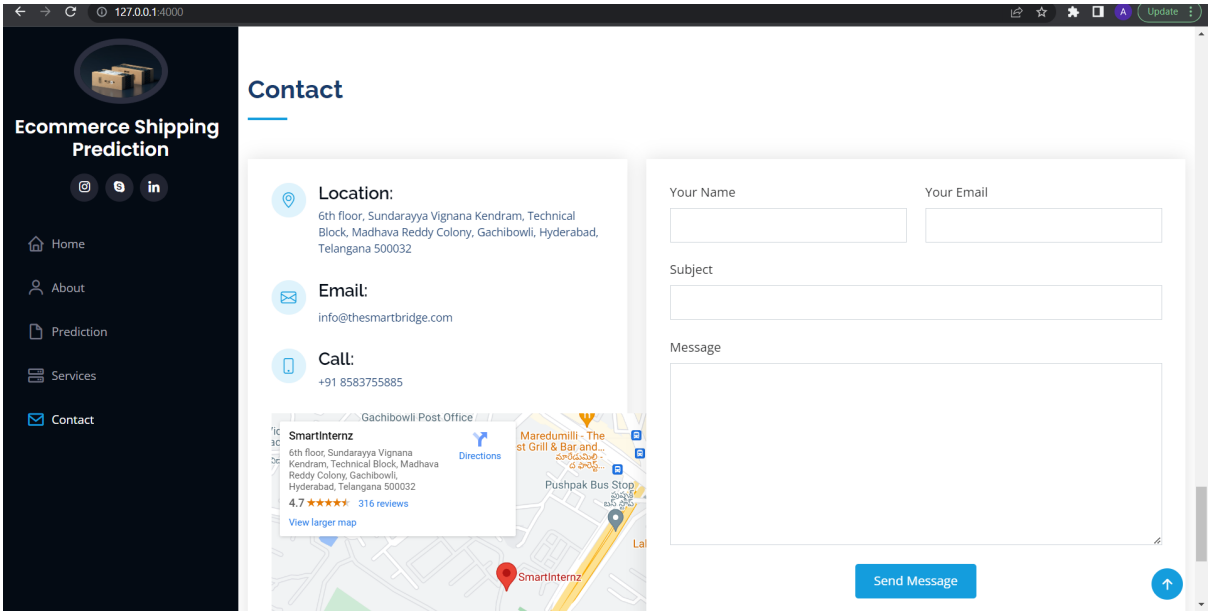
Now,Go the web browser and write the localhost url (http://127.0.0.1:4000) to get the below result


127.0.0.1:4000
Update
Ecommerce Shipping Prediction
Home
About
Prediction
Services
Contact
Shipping Prediction
Predict shipping possibility

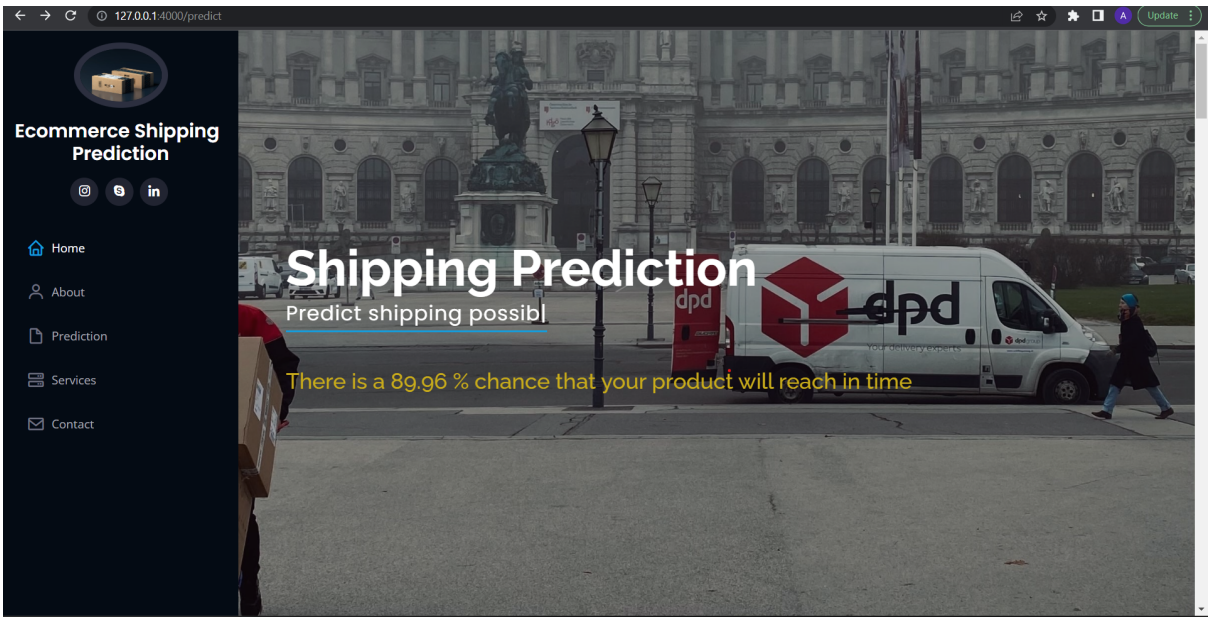

127.0.0.1:4000
Update
Ecommerce Shipping Prediction
Home
About
Prediction
Services
Contact
About
Ecommerce shipping prediction is the process of estimating the whether the product reached on time which is based on various factors such as the origin and destination of the package, the shipping method 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. Over All 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.
Input fields
The data used for predicting the reachability of product by the ML model is done using the following input fields.
Warehouse block: F,A,B,C,D
Mode of shipment: Flight, Ship, Road
Customer care calls: Integer value >1
Customer rating: Integer value >1
Cost of product: Integer value >1
Prior purchases: Integer value >1
Product Importance: Integer value >1
Gender: Male or Female
Discount Offered: Integer value >1
Weight in Grams: Integer value than >1
After entering the above values you can start prediction process to check if your product will reach on time or not.


127.0.0.1:4000
Update
Ecommerce Shipping Prediction
Home
About
Prediction
Services
Contact
PRODUCT IMPORTANCE
Categorical Value
select the Product importance
GENDER
Boolean value
select the Gender
DISCOUNT OFFERED
Integer > 1
WEIGHT IN GRAMS
Integer > 1
Submit and Predict
Output will be visible on the top





Upon entering the data in input fields and clicking on submit we'll get the prediction as shown below.



Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables.

Activity 1: - Record explanation Video for project end to end solution.

Activity 2: - Project Documentation-Step by step project development procedure.

Create document as per the template provided.