

PROJECT REPORT: ECOMMERCE SHIPPING PREDICTION USING MACHINE LEARNING

1 Introduction

1.1 Project Overview

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.

1.2 Purpose

The purpose of an ecommerce prediction shipping model is to forecast and provide accurate estimates for the delivery time of products purchased by customers through an online store.

2 Literature Survey

2.1 Existing problem

While ecommerce shipping prediction using machine learning has made significant strides, there are still challenges and existing problems that need attention. Here are some common issues faced in this domain:

Dynamic and Unpredictable Factors:

1. External factors such as weather conditions, traffic, or customs delays can impact delivery times. Machine learning models may struggle to account for the dynamic and unpredictable nature of these factors.

Data Quality and Availability:

1. The quality and availability of historical shipping data can vary, affecting the model's ability to generalize accurately. Incomplete or biased data may lead to suboptimal predictions.

Seasonal Variations:

- Seasonal fluctuations in demand, especially during holidays, sales events, or peak shopping seasons, can challenge the accuracy of shipping predictions. Models may need to adapt to such variations.

Complex Logistics Networks:

- Ecommerce businesses often rely on complex logistics networks involving multiple carriers and distribution centers. Coordinating and predicting shipping times across this network can be challenging for machine learning models.

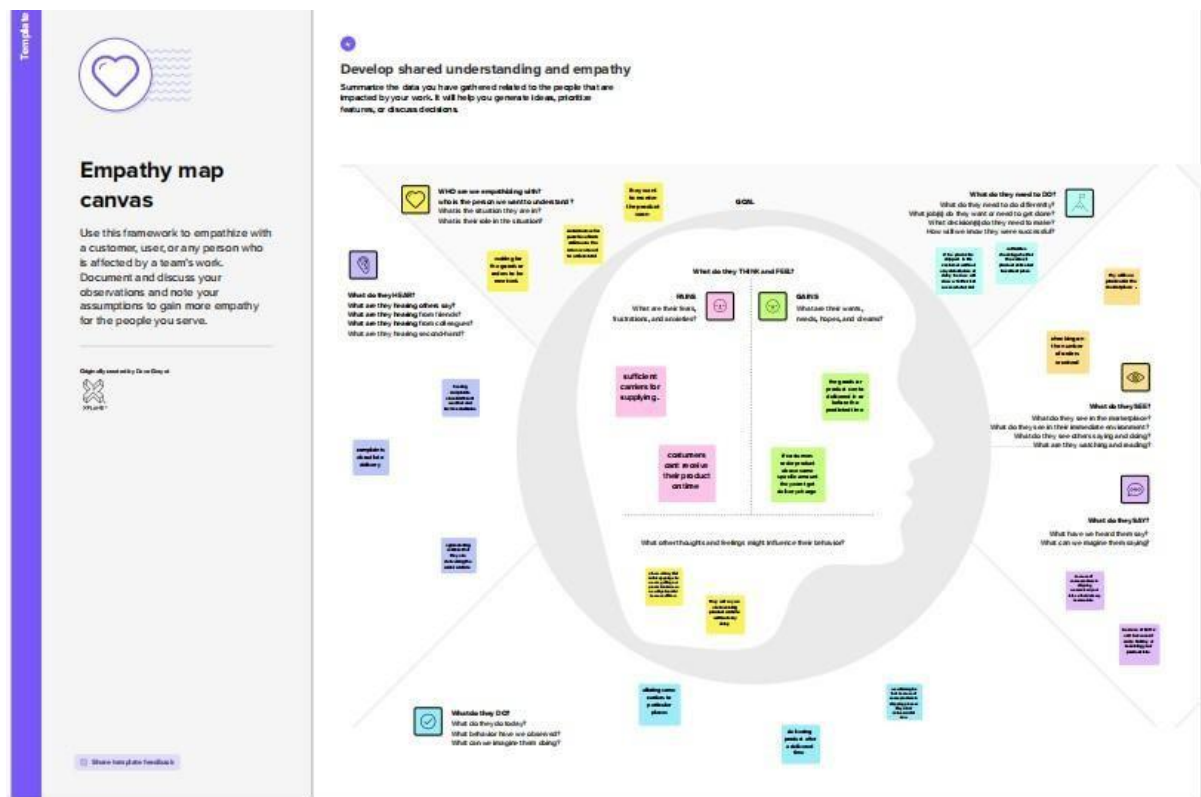
Last-Mile Delivery Challenges:

- The last-mile delivery, which is the final leg of the delivery process to the customer's location, poses specific challenges. Issues such as traffic congestion, address complexities, and failed delivery attempts can impact predictions.

2.2 Problem Statement Definition

Ecommerce Shipping Prediction Using Machine Learning refers to the application of machine learning algorithms and techniques to predict the estimated delivery times or shipping durations for products ordered through an online retail platform. This process involves leveraging historical shipping data, relevant features, and possibly external factors to create a model that can forecast the time it will take for an order to be shipped and delivered to the customer.

The primary goal of Ecommerce Shipping Prediction is to provide accurate and reliable estimates to customers, giving them insight into when they can expect their orders to arrive. This predictive capability enhances the overall customer experience by setting clear expectations, reducing uncertainty, and improving customer satisfaction.



3.2 Ideation and Brainstorming

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

🕒 10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

praveen kumar

Follow a Standard Operating Procedure (SOP) to streamline carrier operations and achieve fastest shipping speed

Optimize delivery routes by selecting the fastest, most cost-effective routes possible

Split your inventory in multiple fulfillment centers across India to reduce transit times dramatically

padmaja mallipudi

Get shipping quotes from multiple carrier companies and choose the best carrier at a reasonable price

Provide Updates Through Notifications

Use Multiple Storage Locations

muthineni himaja

Make Sure You Have a Reliable Carrier Network and Use Visibility Tools for Tracking Shipments

Integrate Your Order and Delivery Management System with Your Logistics Platform

Use Express Package Delivery

Person 4

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

🕒 20 minutes

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

Decrease the shipping distance

Use Multiple Storage Locations

Don't stick with one, consider building multi-carrier strategies

Improve inventory management

Switch from boxes to poly mailers

4

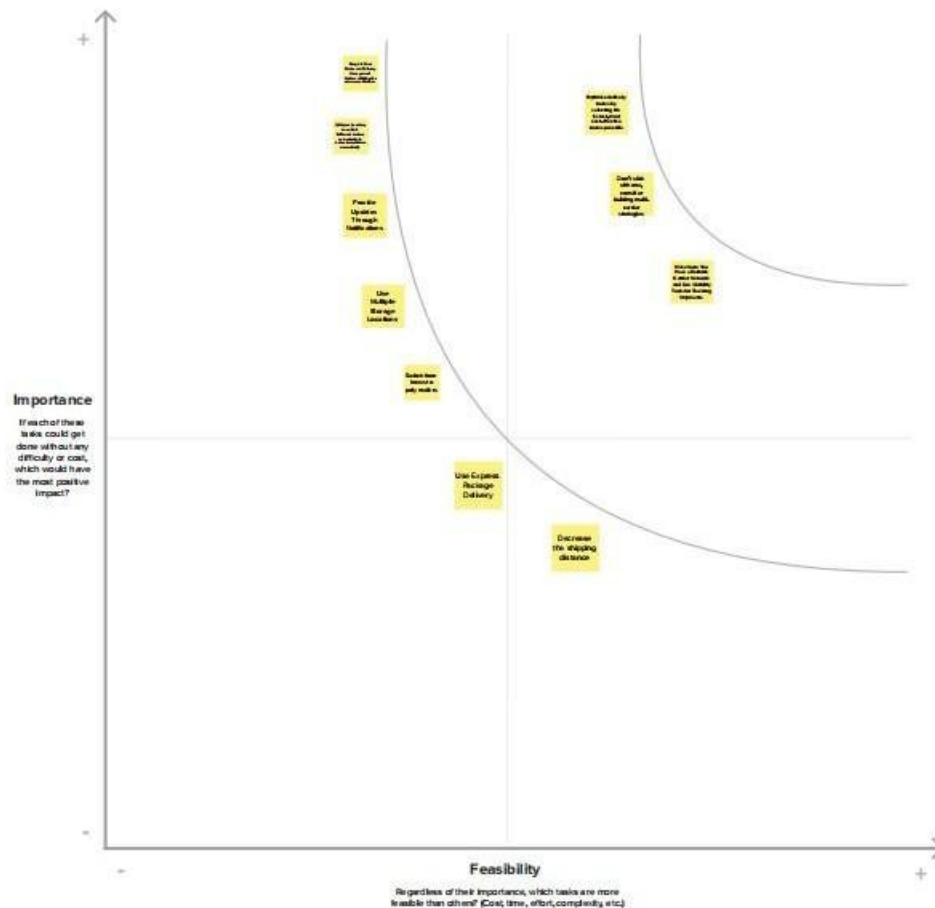
Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

TP

Participants can use their cursor to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the H key on the keyboard.



4 Requirement Analysis

4.1 Functional Requirements

- **Data Ingestion:**

- **Description:** The system should be able to ingest and process relevant data related to past shipping records, order details, geographic information, and any external factors influencing shipping times.

- **Requirements:**

- Ability to connect to and extract data from various sources (order databases, logistics systems, etc.).
- Support for periodic and real-time data updates.

- **Feature Extraction and Selection:**

- **Description:** Identify and extract relevant features from the data that contribute to accurate shipping time predictions.

- **Requirements:**

- Algorithms or methods for feature extraction and selection.
 - Capability to handle both numerical and categorical features.

- **Machine Learning Model Training:**

- **Description:** Train machine learning models using historical data to predict shipping times.

- **Requirements:**

- Selection of appropriate machine learning algorithms (e.g., regression models, decision trees, neural networks).
 - Training pipeline with options for hyperparameter tuning.
 - Validation and evaluation metrics.

- **Real-Time Prediction:**

- **Description:** Provide real-time predictions for shipping durations as new orders are placed.

- **Requirements:**

- Low-latency model inference.
 - Integration with ecommerce platform for seamless real-time prediction.

4.2 Non-Functional Requirements

- **Performance:**

- **Description:** The system should deliver predictions within a reasonable time frame to ensure a responsive user experience.

- **Requirements:**

- Maximum acceptable latency for real-time predictions.
 - Throughput requirements to handle peak loads.

- **Scalability:**

- **Description:** The system should scale horizontally to accommodate an increasing volume of orders and data.

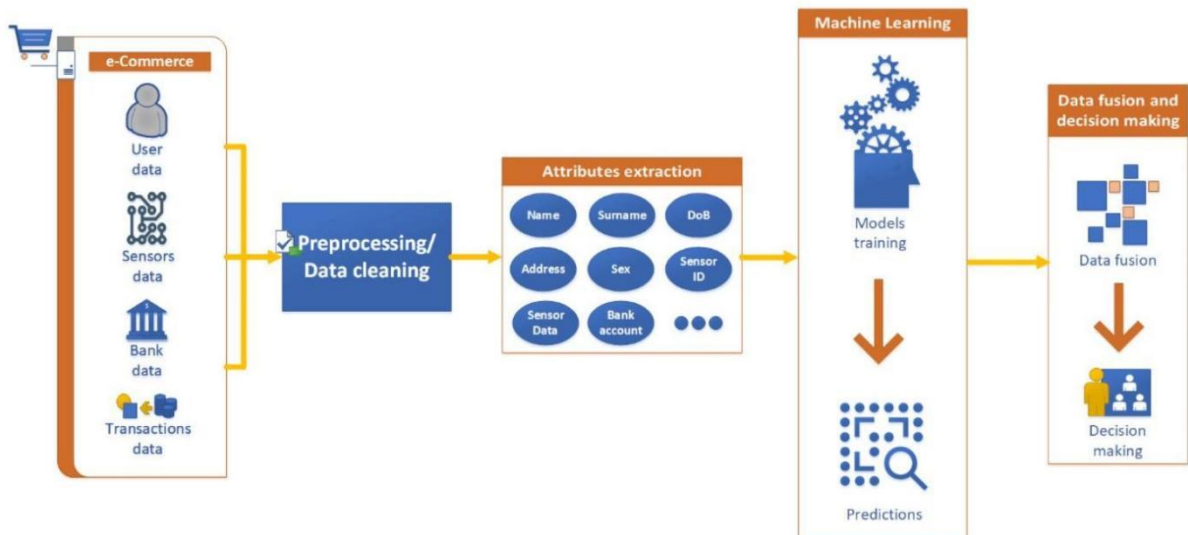
- **Requirements:**

- Ability to add resources dynamically to handle growing workloads.

- Scalable infrastructure that can distribute computations across multiple nodes.
- **Reliability:**
 - **Description:** The system should be reliable, providing accurate predictions consistently.
 - **Requirements:**
 - High availability with minimal downtime.
 - Measures to handle and recover from system failures.
- **Availability:**
 - **Description:** The system should be available for predictions whenever users place orders.
 - **Requirements:**
 - Define acceptable levels of downtime for maintenance or updates.
 - Implement redundancy and failover mechanisms to ensure continuous availability.

5 Project Design

5.1 Data Flow Diagrams and User Stories



User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Order received by the customer	product	USN-1	As a customer, <u>im</u> facing issue in receiving the <u>prodcut</u> .	Issue in the shipping process	High	Sprint-1
		USN-2	As a customer, <u>im</u> receiving my product at late delivery time	Because of delay in shipping process, carriers , weather	High	Sprint-1
		USN-3	As a customer , <u>im</u> receiving my product at a scheduled time.	No issues in any process	Low	Sprint-2
		USN-4	As a customer , <u>im</u> getting product with damage.	Because of some issue in shipping process	Medium	Sprint-1
		USN-5	As a customer , I cant able to track my order.	Because of server problem	High	Sprint-1
	Tracking	USN-6	As a customer , I can track my order with proper <u>imformation</u>	Because of proper development of app	High	Sprint-1
		USN-7	As a customer , I can get notification about my order	because of proper development of app	High	Sprint-1

5.2 Solution Architecture

- **Data Ingestion and Storage:**

- **Component:** Data Ingestion Layer, Data Storage
- **Description:** Ingest relevant data sources, including historical shipping records, order details, and external factors. Store the data in a scalable and fault-tolerant data storage solution, such as a distributed database or data warehouse.

- **Data Processing and Feature Engineering:**

- **Component:** Feature Extraction Module
- **Description:** Process the raw data to extract relevant features that contribute to accurate shipping predictions. Implement feature engineering techniques and transformations to prepare the data for model training.

- **Machine Learning Model Training:**

- **Component:** Model Training Module
- **Description:** Train machine learning models using the pre-processed data. Choose appropriate algorithms (e.g., regression models, decision trees) and employ hyperparameter tuning for optimal model performance.

- **Model Serving and Inference:**

- **Component:** Prediction Service, Model Serving Layer
- **Description:** Deploy trained machine learning models to a prediction service capable of handling real-time inference. This service should provide low-latency responses for predictions and integrate with the ecommerce platform.

6 Project planning and scheduling

6.1 Sprint Planning and Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	product	USN-1	As a customer, im facing issue in receiving the product.	2	High	praveen
Sprint-1		USN-2	As a customer, im receiving my product at late delivery time	1	High	padmaja
Sprint-2		USN-3	As a customer , im receiving my product at a scheduled time.	2	Low	himaja
Sprint-1		USN-4	As a customer , im getting product with damage.	2	Medium	Praveen
Sprint-1	Tracking	USN-5	As a customer , I cant able to track my order.	1	High	padmaja
Sprint-1		USN-6	As a customer , I can track my order with proper information	2	High	himaja
Sprint-1		USN-7	As a customer , I can get notification about my order	1	Medium	himaja

6.2 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2023
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	19	25Oct 2023
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	18	24Oct 2023
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	30Oct 2023

7 Coding and Solutioning

7.1 HTML CODE :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>E-commerce Shipping Prediction</title>
  <!-- Link to your CSS file -->
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <h1>E-commerce Shipping Prediction</h1>
  <form id="predictionForm">
    <label for="warehouseBlock">Warehouse Block:</label>
    <input type="text" id="warehouseBlock" name="warehouseBlock"><br><br>

    <label for="Mode_of_shipment">Mode of shipment:</label>
    <input type="text" id="Mode_of_shipment" name="Mode_of_shipment"><br><br>

    <label for="Customer_care_calls">Customer care calls:</label>
```

```

<input type="text" id="Customer_care_calls" name="Customer_care_calls"> <br> <br>

<label for="Customer_rating">Customer rating:</label>
<input type="text" id="Customer_rating" name="Customer_rating"> <br> <br>

<label for="Cost_of_the_Product">Cost of the Product:</label>
<input type="text" id="Cost_of_the_Product" name="Cost_of_the_Product"> <br> <br>

<label for="Prior_purchases">Prior purchases:</label>
<input type="text" id="Prior_purchases" name="Prior_purchases"> <br> <br>

<label for="Product_importance">Product importance:</label>
<input type="text" id="Product_importance" name="Product_importance"> <br> <br>

<label for="Gender">Gender:</label>
<input type="text" id="Gender" name="Gender"> <br> <br>

<label for="Discount_offered">Discount offered:</label>
<input type="text" id="Discount_offered" name="Discount_offered"> <br> <br>

<label for="Weight_in_gms">Weight_in_gms:</label>
<input type="text" id="Weight_in_gms" name="Weight_in_gms"> <br> <br>

<input type="button" value="Predict" onclick="predictShipping()">
</form>
<div id="predict"> </div>

<script src="script.js"> </script>
</body>
</html>

```

7.2 CSS CODE :

```

body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f5f5f5;
    color: #333;
}

```

```

h1 {
    text-align: center;
    margin-bottom: 20px;
}

```

```

form {
    background-color: #fff;
    padding: 20px;
    border-radius: 5px;
}

```

```

    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    max-width: 400px;
    margin: 0 auto;
}

```

```

label {
    display: block;
    margin-bottom: 8px;
}

```

```

input[type="text"],
input[type="button"] {
    width: calc(100% - 22px);
    padding: 10px;
    margin-bottom: 10px;
    border-radius: 4px;
    border: 1px solid #ccc;
}

```

```

input[type="button"] {
    background-color: #007bff;
    color: #fff;
    cursor: pointer;
    border: none;
}

```

```

input[type="button"]:hover {
    background-color: #0056b3;
}

```

```

#predictionResult {
    margin-top: 20px;
    font-weight: bold;
}

```

7.3 JAVASCRIPT CODE :

```

function predictShipping() {
    var Warehouse_Block = document.getElementById('Warehouse_Block').value;
    var Mode_of_shipment = document.getElementById('Mode_of_shipment').value;
    var Customer_care_calls = document.getElementById('Customer_care_calls').value;
    var Customer_rating = document.getElementById('Customer_rating').value;
    var Cost_of_the_Product = document.getElementById('Cost_of_the_Product').value;
    var Prior_purchases = document.getElementById('Prior_purchases').value;
    var Product_importance = document.getElementById('Product_importance').value;
    var Gender = document.getElementById('Gender').value;
    var Discount_offered = document.getElementById('Discount_offered').value;
    var Weight_in_gms = document.getElementById('Weight_in_gms').value;

    // Get other input values similarly

    // Create a data object to send to the server
}

```

```

var data = {
  warehouseBlock: warehouseBlock,
  Mode_of_shipment: Mode_of_shipment,
  Customer_care_calls: Customer_care_calls,
  Customer_rating: Customer_rating,
  Cost_of_the_Product: Cost_of_the_Product,
  Prior_purchases: Prior_purchases,
  Product_importance: Product_importance,
  Gender: Gender,
  Discount_offered: Discount_offered,
  Weight_in_gms: Weight_in_gms
};

// Send a POST request to the server endpoint '/predict'
fetch('/predict', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(data)
})
.then(response => response.json())
.then(result => {
  // Display prediction result in the HTML
  document.getElementById('predictionResult').innerHTML = "<p>Shipping Prediction: " +
result.prediction + "</p>";
  // Process other parts of the prediction result if needed
})
.catch(error => {
  console.error('Error:', error);
});
}

```

7.4 MODEL SET :

[illegible]

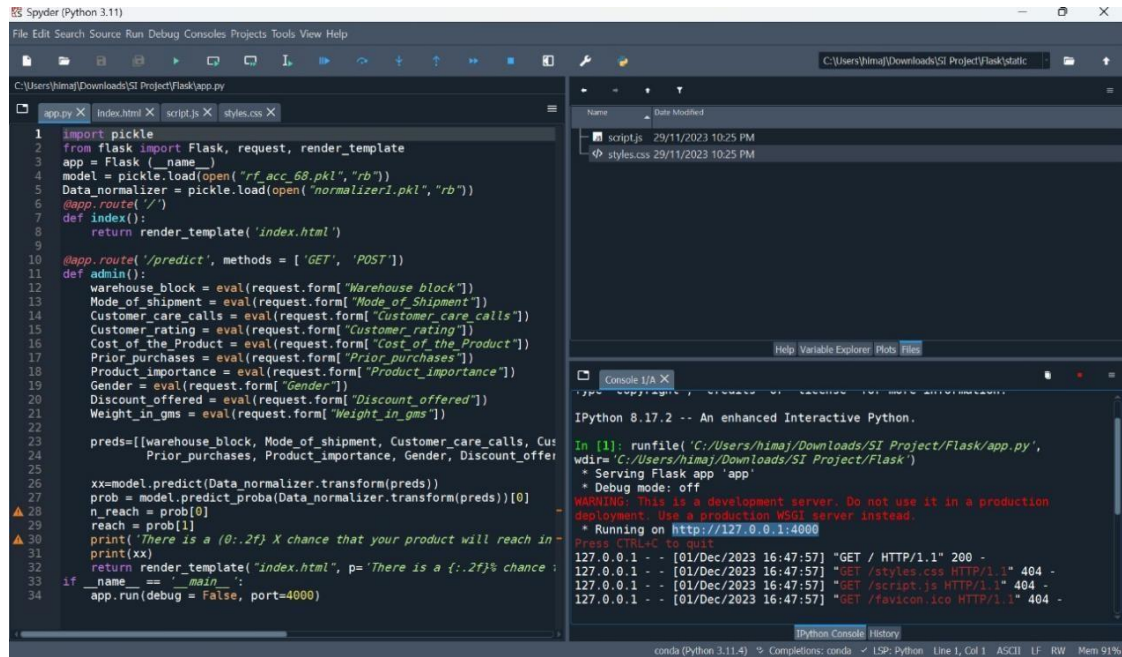
7.5 WEB PAGE :

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:4000'. The page title is 'E-commerce Shipping Prediction'. The main heading is 'E-commerce Shipping Prediction'. Below the heading, there are ten input fields, each with a label and a text box:

- Warehouse Block:
- Mode of shipment:
- Customer care calls:
- Customer rating:
- Cost of the Product:
- Prior purchases:
- Product importance:
- Gender:
- Discount_offered:
- Weight_in_gms:

At the bottom of the form, there is a button labeled 'Predict'.

Run application:



The screenshot shows the Spyder Python IDE interface. The main editor displays a Python script for a Flask application. The script imports necessary libraries, loads a pre-trained model, and defines routes for index, admin, and prediction. The application is running on port 4000. The IPython console shows the execution of the script and the resulting HTTP responses for various requests.

```
1 import pickle
2 from flask import Flask, request, render_template
3 app = Flask(__name__)
4 model = pickle.load(open("rf_acc_68.pkl", "rb"))
5 Data_normalizer = pickle.load(open("normalizer1.pkl", "rb"))
6 @app.route('/')
7 def index():
8     return render_template('index.html')
9
10 @app.route('/predict', methods = ['GET', 'POST'])
11 def admin():
12     warehouse_block = eval(request.form["Warehouse block"])
13     Mode_of_shipment = eval(request.form["Mode of Shipment"])
14     Customer_care_calls = eval(request.form["Customer_care calls"])
15     Customer_rating = eval(request.form["Customer_rating"])
16     Cost_of_the_Product = eval(request.form["Cost of the Product"])
17     Prior_purchases = eval(request.form["Prior_purchases"])
18     Product_importance = eval(request.form["Product_importance"])
19     Gender = eval(request.form["Gender"])
20     Discount_offered = eval(request.form["Discount offered"])
21     Weight_in_gms = eval(request.form["Weight_in_gms"])
22
23     preds=[warehouse_block, Mode_of_shipment, Customer_care_calls, Cus
24             Prior_purchases, Product_importance, Gender, Discount_offer
25
26     xx=model.predict(Data_normalizer.transform(preds))
27     prob = model.predict_proba(Data_normalizer.transform(preds))[0]
28     n_reach = prob[0]
29     reach = prob[1]
30     print('There is a {0:.2f} X chance that your product will reach in-
31           print(xx)
32     return render_template("index.html", p='There is a {:.2f}% chance :
33 if __name__ == '__main__':
34     app.run(debug = False, port=4000)
```

IPython 8.17.2 -- An enhanced Interactive Python.

```
In [1]: runfile('C:/Users/himaj/Downloads/SI Project/Flask/app.py',
* 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
127.0.0.1 - - [01/Dec/2023 16:47:57] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [01/Dec/2023 16:47:57] "GET /styles.css HTTP/1.1" 404 -
127.0.0.1 - - [01/Dec/2023 16:47:57] "GET /script.js HTTP/1.1" 404 -
127.0.0.1 - - [01/Dec/2023 16:47:57] "GET /favicon.ico HTTP/1.1" 404 -
```

7.6 Data Preprocessing :

IMPORT LIBRARIES

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import pickle as pk1
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
```

Read the Dataset

```
[2] data = pd.read_csv(r"Train.csv")
data.head()
```

Python

...

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Prod
0	1	D	Flight	4	2	177	3	
1	2	F	Flight	4	5	216	2	
2	3	A	Flight	2	2	183	4	
3	4	B	Flight	3	3	176	4	
4	5	C	Flight	2	2	184	3	

Data Preparation

1. Handling missing values

```
[3] data.shape
```

Python

...

(10999, 12)

```
[4] data.info
```

Python

...

```
<bound method DataFrame.info of
0      1      D      Flight      4
1      2      F      Flight      4
2      3      A      Flight      2
3      4      B      Flight      3
4      5      C      Flight      2
...    ...    ...    ...    ...
10994  10995      A      Ship      4
10995  10996      B      Ship      4
10996  10997      C      Ship      5
10997  10998      F      Ship      5
10998  10999      D      Ship      2

      Customer_rating  Cost_of_the_Product  Prior_purchases  \
0                2            177            3
1                5            216            2
2                2            183            4
3                3            176            4
4                2            184            3
...            ...            ...            ...
10994            1            252            5
10995            1            232            5
10996            4            242            5
10997            2            223            6
10998            5            155            5
...            ...            ...            ...
10996            0
10997            0
```

```
> data.isnull().sum()
[5]
... 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
```

2. Handling the categorical values

```
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

6]
{'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}}
```

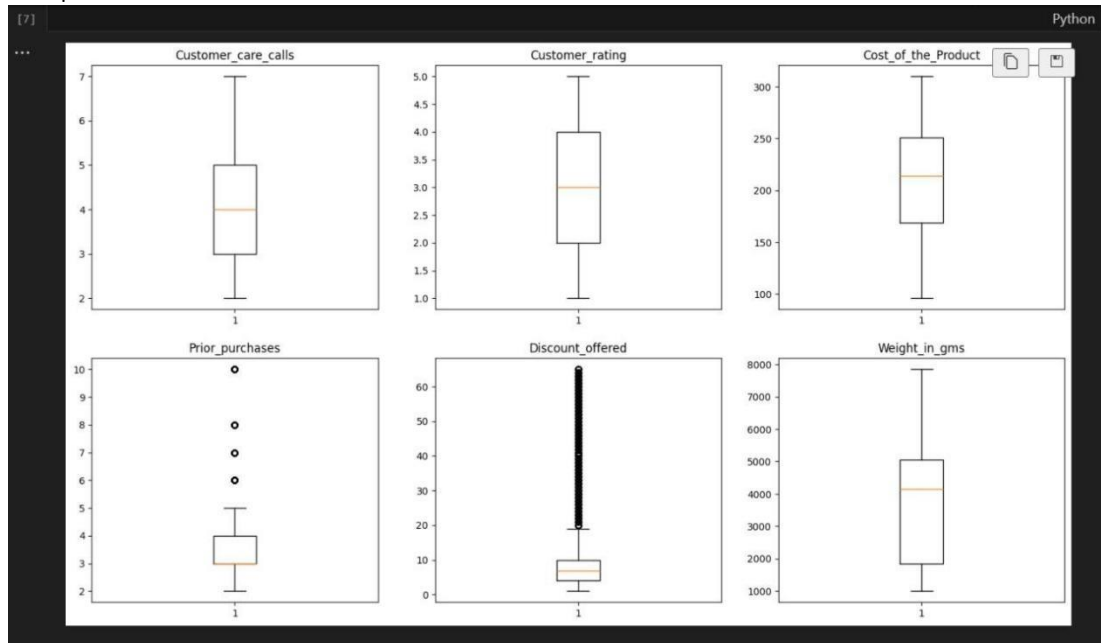
3. Handling the Outliners in the Data

INPUT

```
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'
]).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()

Python
```


Output



Input

```
[8] Python

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'
]).columns:
    if str(data[i].dtype)=='object':
        continue
    print(i)
    check_outliers(data[i])
```

Output

```
[8]
... 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
```

Data splitting

```
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,)

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

(2200,)
```

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

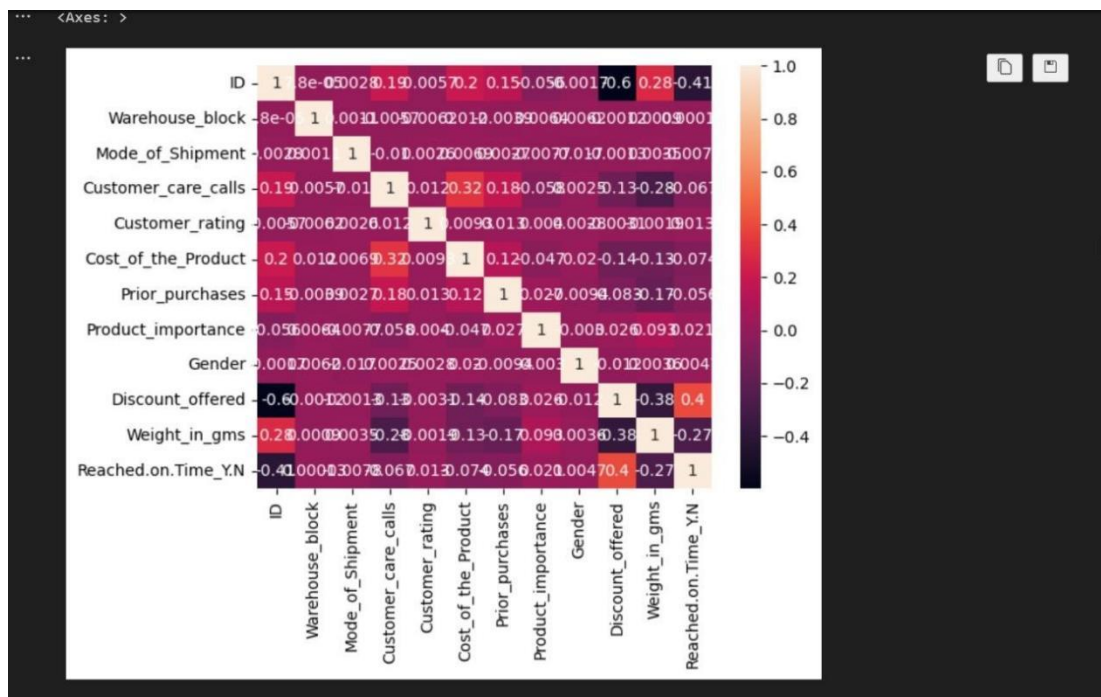
[10] Python

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

Multivariate analysis

```
sns.heatmap(data.corr(),annot=True)
```

[11] Python



Pair plot

```
sns.pairplot(data)
```

[12] Python



Model Building

Creating the functions to train the models

```
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,lcx,rgb,rg,knn,rf,svc
```

[13] Python

NORMALISED DATA :

```
# Normalize data
normalizer = Normalizer()
x_train_normalized = normalizer.fit_transform(x_train)
x_test_normalized = normalizer.transform(x_test)
```

[14] Python

```
lg,lcx,rgb,rg,knn,rf,svc = models_eval_mm(x_train_normalized,y_train,x_test_normalized,y_test)
```

[15] Python

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

--Logistic Regression CV
Train Score: 0.6541652460506876
Test Score: 0.65

--XGBoost
Train Score: 0.9362427548585066
Test Score: 0.6681818181818182

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

--KNN
Train Score: 0.7756563245823389
Test Score: 0.6336363636363637

--Random Forest
Train Score: 1.0
Test Score: 0.6686363636363636

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

Testing the model

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

[16] Python

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

```
[17] lg.predict(x_test_normalized[0].reshape(1,-1))  
... array([1], dtype=int64)
```

Performance Testing and hyperparameter tuning

Compare the model

```
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	65.00	67.89	62.42	74.41
2	XGBoost	66.82	71.03	68.63	73.60
3	Ridge classifier	59.27	74.43	100.00	59.27
4	KNN	63.36	68.27	66.49	70.15
5	Random Forest	66.86	69.64	64.11	76.21
6	Support Vector Classifier	59.27	74.43	100.00	59.27

Comparing the model accuracy before and after hyper parameter tuning

```
XGBOOST

xgb.get_params()

[20] Python

... {'objective': 'binary:logistic',
      'base_score': None,
      'booster': None,
      'callbacks': None,
      'colsample_bylevel': None,
      'colsample_bynode': None,
      'colsample_bytree': None,
      'device': None,
      'early_stopping_rounds': None,
      'enable_categorical': False,
      'eval_metric': None,
      'feature_types': None,
      'gamma': None,
      'grow_policy': None,
      'importance_type': None,
      'interaction_constraints': None,
      'learning_rate': None,
      'max_bin': None,
      'max_cat_threshold': None,
      'max_cat_to_onehot': None,
      'max_delta_step': None,
      'max_depth': None,
      'max_leaves': None,
      'min_child_weight': None,
      'missing': nan,
```

```
RANDOM FOREST

rf.get_params()

[21] Python

... {'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()

[22] Python

... {'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}
```

HYPERPARAMETER 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

[23] Python

... Fitting 5 folds for each of 24 candidates, totalling 120 fits
SVC(C=13, gamma=5, random_state=1234, tol=0.01) {'C': 13, 'gamma': 5, 'kernel': 'rbf', 'tol': 0.01} 0.6676890278567368
```

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

[24] Python

... Fitting 5 folds for each of 54 candidates, totalling 270 fits
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.6, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=2.0, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.5, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
              min_child_weight=10, missing=None, monotone_constraints=None,
              multi_strategy=None, n_estimators=100, n_jobs=None, nthread=3,
              num_parallel_tree=None, ...) {'colsample_bytree': 0.6, 'gamma': 2.0, 'max_depth': 4, 'min_child_weight':
```


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}

```

[25] Python

... Fitting 5 folds for each of 15 candidates, totalling 75 fits
Best Score:0.655869489379296
Best Parameters: {'Cs': 20, 'max_iter': 80}

RANDOM FOREST HYPERPARAMETER OPTIMISATION

```

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf_param_grid = {
    'n_estimators': [200, 300, 500],
    'criterion': ['entropy', 'gini'],
    'max_depth': [7, 8, 60, 80, 100],
    'max_features': ['sqrt', 'log2', 1, 2, None] # Modify this list based on your feature count
}

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:", rf_cv.best_score_)
print("Best Parameters:", rf_cv.best_params_)

```

[26] Python

... Fitting 7 folds for each of 150 candidates, totalling 1050 fits
Best Score: 0.6816683714058415
Best Parameters: {'criterion': 'entropy', 'max_depth': 7, 'max_features': 'sqrt', 'n_estimators': 300}

MODEL DEPLOYMENT :

```
import pickle as pk1

pk1.dump(rf, open('rf_acc_68.pkl', 'wb'))

pk1.dump(Data_normalizer, open('normalizer1.pkl', 'wb'))
```

```
import pickle ...

app = Flask(__name__)
model = pickle.load(open("rf_acc_68.pkl", "rb"))
Data_normalizer = pickle.load(open("normalizer1.pkl", "rb"))

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

APP.py code

```
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("normalizer1.pkl", "rb"))

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

@app.route('/predict', methods=['POST'])
def admin():
    warehouse_block = eval(request.form["warehouseBlock"])
    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]
result_message = 'There is a {:.2f}% chance that your product will reach in
time'.format(reach * 100)
print(result_message)
return render_template("index.html", p=result_message, name='main')

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