CANDIDATE’S DECLARATION

I hereby certify that the work which is being presented in this Minor thesis titled **“AMAZON DELIVERY DATA ANALYSIS”** in fulfilment of the requirement for the degree of Master of Technology in Computer Engineering (specialization in Networking) and submitted to “**SATYUG DARSHAN INSTITUTE OF ENGINEERING AND TECHNOLOGY**”, is an authentic record of my own work carried out under the supervision of **Mr. Ankit Mishra.**

The work contained in this thesis has not been submitted to any other University or Institute for the award of any other degree or diploma by me.

Vishnu Garg	Yash
BCA(DS-23/057)	BCA(DS-23/058)

ACKNOWLEDGEMENT

It is with deep sense of gratitude and reverence that I express my sincere thanks to my supervisor **Mr.Ankit Mishra** for her guidance, encouragement, help and useful suggestions throughout. Her untiring and painstaking efforts, methodical approach and individual help made it possible for me to complete this work in time. I consider myself very fortunate for having been associated with the scholar like her. Her affection, guidance and scientific approach served a veritable incentive for completion of this work.

I shall ever remain indebted to the faculty members of **Satyug Darshan Institute of Engineering and Technology, Faridabad, CAREERERA** team and all my classmates for their cooperation, kindness and general help extended to me during the completion of this work.

Although it is not possible to name individually, I cannot forget my well-wishers at Satyug Darshan Institute of Engineering and Technology, Faridabad and outsiders for their persistent support and cooperation.

This acknowledgement will remain incomplete if I fail to express my deep sense of obligation to my parents and God for their consistent blessings and encouragement

(Vishnu Garg)
(Yash)

CERTIFICATE

ABOUT TRAINING

As each and every sector of the market is growing, data is building up day by day, we need to keep the record of the data which can be helpful for the analytics and evaluation. Now we don’t have data in gigabyte or terabyte but in zetta byte and petabyte and this data cannot be handled with the day By day software such as Excel or MATLAB. Therefore, in this report we will be dealing with large data sets with the high-level programming language ‘Python’.

The main goal of this training is to aggregate and analyse the data collected from the different data sources available on the internet like Kaggle etc., This project mainly focuses on the usage of the python programming language and Data Analysis. This language has not only it’s application in the field of just analysing the data and represent it graphically.

LIST OF ABBRIVATIONS

- pd for pandas
- np for NumPy
- plt for pyplot
- sns for seaborn
- df for data frame
- int for integer
- len for length
- str for string
- bool for Boolean
- arr for array
- loc for location
- info for information
- col for columns
- hist for histogram
- sqrt for square root

LIST OF FIGURES

Figures	Page no.
1.2.1 Arithmetic operator	10
1.2.2 Assignment operator	11
1.2.3 Logical operator	12
1.2.4 Comparison operator	12
1.2.5 Membership operator	13
1.2.6 if else statement	14
1.2.7 Elif statement	16
1.3.1 Matplotlib basic example	18
1.3.2 Pandas basic example	20
1.3.3 NumPy basic example	21

CHAPTER 1: INTRODUCTION TO DATA SCIENCE

1.1 Introduction to topic

- **Optimizing Operational Efficiency**: This report investigates the performance metrics of Amazon's delivery services, exploring how data analysis can enhance operational efficiency.

- **Addressing Multifaceted Challenges**: With an extensive global network, Amazon's delivery system faces complex challenges. This analysis aims to uncover insights that streamline processes and enhance delivery reliability.

- **Enhancing the Customer Experience**: By analyzing delivery data, this report seeks to improve the delivery experience for customers, ensuring timely arrivals and meeting expectations.

- **Data-Driven Strategies**: Leveraging data-driven approaches, this report examines trends and patterns to inform strategic decisions, aiming for continuous improvement in Amazon's delivery operations.

1.2 Motivation

- **Optimizing Delivery Routes**: Analyzing data can reveal patterns that enable the optimization of delivery routes, leading to reduced delivery times and costs while enhancing service reliability.

- **Enhancing Customer Satisfaction**: Insights gained from data analysis help ensure on-time deliveries, accurate tracking, and proactive customer communication, ultimately improving overall customer satisfaction.

- **Operational Efficiency**: By identifying bottlenecks and inefficiencies in the delivery process, Amazon can streamline operations, allocate resources more effectively, and enhance fleet management, thereby improving operational efficiency.

- **Competitive Advantage**: Leveraging data to enhance delivery logistics provides Amazon with a competitive edge by offering faster and more reliable service compared to competitors..

1.3 Objective of training

- To give participants the ability to create and refine predictive models that precisely categorize, regress, and cluster data in order to address practical issues.

- ❖ To give participants the know-how to prepare data for machine learning algorithms by preprocessing, transforming, and visualizing it.

- ❖ To teach participants how to choose, apply, and assess machine learning algorithms to handle challenging issues in computer vision, natural language processing, and recommender systems, among other fields.

- ❖ Giving participants the tools they need to implement and incorporate machine learning models into bigger systems while maintaining maintainability, scalability, and dependability.

CHAPTER 2: PYTHON FOR DATA SCIENCE

1.1. Introduction to Python

"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics". This language consist of mainly data structures which make it very easy for the data scientists to analyse the data very effectively. It does not only help in forecasting and analysis it also helps in connecting the two different languages. Two best features of this programming language is that it does not have any compilation step as compared to the other programming language in which compilation is done before the program is being executed and other one is the reuse of the code, it consist of modules and packages due to which we can use the previously written code anywhere in between the program whenever is required. There are multiple languages for example R, Java, SQL, MATLAB available in market which can be used to analyse and evaluate the data, but due to some outstanding features python is the most famous language used in the field of data science.

Python is mostly used and easy among all other programming languages.

1.2 Operators, Conditional Statements …..

OPERATORS - Operators are the symbols in python that are used to perform Arithmetic or logical operations. Following are the different types of operators in python.

Arithmetic operators - Arithmetic operators carry out mathematical operations and they are mostly used with the numeric values.

Arithmetic operators		
Operator	Name	Example
+	Addition	A+B
-	Subtraction	A-B
*	Multiplication	A*B
/	Division	A/B
%	Modulus	A%B
**	Exponentiation	A**B
//	Quotient	A//B

Fig. 1.2.1: Arithmetic

operators A and B are the numeric value

Assignment operators - As the name decides this operators are used for assigning the values to the variables.

ASSIGNMENTOPERATORS		
Operator	Example	mayalsobe written
=	a=6	a=6
+=	a+=3	a=a+3
-=	a-=4	a=a-4
*=	a*=5	a=a* 5
/=	a /= 6	a =a / 6
%=	a%=7	a=a%7
//=	a//=8	a=a// 8
**=	a**=9	a=a** 9
&=	a&= 1	a=a&1

Fig. 1.2.2: Assignment Operators

Here a is any value and number of operations are performed on this value.

Logical operators - These operators are used to join conditional statements

Logical Operators		
Operator	Description	Example
and	If both statements are true it Returns true	x <5 **and**x <10
or	If any of the two statement Is true it returns true	x <4 **or**x <8
not	If the result is true it reverses the result and gives false	**not**(x<4 **and**x <8)

Fig.1.2.3:Logical Operators

Here a is any value provided by us and on which multiple operations can be performed.

Comparison operators - These operators are used to compare two different values.

Comparison operators		
Operator	Name	Example
==	Equal	a==b
!=	Not equal	a!=b
>	Greatert han	a>b
<	less than	a<b
>=	Greater than Equal to	a>=b
<=	Less than equal to	a<=b

Fig. 1.2.4: Comparison operators

Here a and b are two different values and these values are compared.

Membership operators - These operators are used to check membership of a particular value. It is used to check whether a specific value is present in the object or not.

Membershipoperators		
Operator	Description	Example
in	It returns a True if the value is present inside the object	A **in** b
not in	It returns a True if the value is not present inside the object	A **not in** b

Fig.1.2.5:Membershipoperators

Condition statements

If elsestatements

“Themost common type of statement is the if statement. ifstatement consistof a block which is called as clause”, it is the block after if statement, it executed the statement if the condition is true.The statement is omitted if the conditionis False. then the statement in the else part is printed

If statement consist of following-

• **If keyword itself**
• **Condition which may be True or False**
• **Colon**
• **If clause or a block of code** Below is the figure shows how If and else statements are used with description inside it.

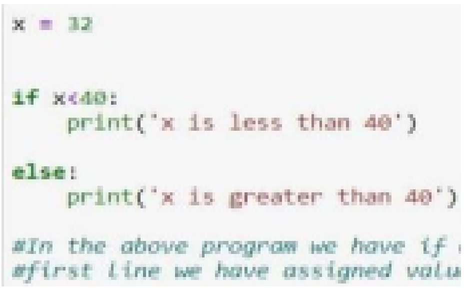


Figure 1.2.6 : if else statement

elif statements

In this statement only one statement is executed, There are many cases in which there is only one possibility to execute. ”The elif statement is an else if statement that always follows an if or another elif statement”[8]. The elif statement provides another condition that is checked only if any of the previous conditions were False. In code, an elif statement always consists of the following:. The only difference between if else and elif statement is that in elif statement we have the condition where as in else statement we do not have any condition.

elIf statement consist of following -

- **elIf keyword itself**
- **Condition which may be True or False**
- **Colon**
- **elIf clause or a block of code**

Below is the figure shows how elIf statement is used with description inside it.

```
var = 't'

if var == 'a':
    print('this is the vowel a')
elif var == 'e':
    print('this is the vowel e')
elif var == 'i':
    print('this is the vowel i')
elif var == 'o':
    print('this is the vowel o')
elif var == 'u':
    print('this is the vowel u')
else:
    print('The value in variable var is constant')
```

Figure 1.2.7: elif example

1.3 Understanding Standard Libraries Pandas, Numpy…..

Libraries in Python

Python library is vast. There are built in functions in the library which are written in C lan- guage. This library provide access to system functionality such as file input output and that is not accessible to Python programmers. This modules and library provide solution to the many problems in programming.

Following are some Python libraries.

Matplotlib
Pandas
Numpy

Matplotlib

”Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy”[11]. MATLAB provides an application that is used in graphical user interface tool kits. Another such library is pylab which is almost same as MATLAB.

It is a library for 2D graphics, it finds its application in web application servers, graphical user interface toolkit and shell. Below is the example of a basic plot in python.

```
import matplotlib.pyplot as plt
#we have imported matplotlib library first
#we have imported the pyplot module from matplotlib library
#we have give name 'plt' instead of using whole function name
plt.plot([1,2,3],[1,3,4])
#we used plot function to plot a graph
#we have take simple list in plot function
plt.xlabel('x label') #This function is used to name x axis
plt.ylabel('y label') #This function is used to name y axis
plt.title('basic plot') #This function used for title of grapho
plt.show() #This function show the graph
```

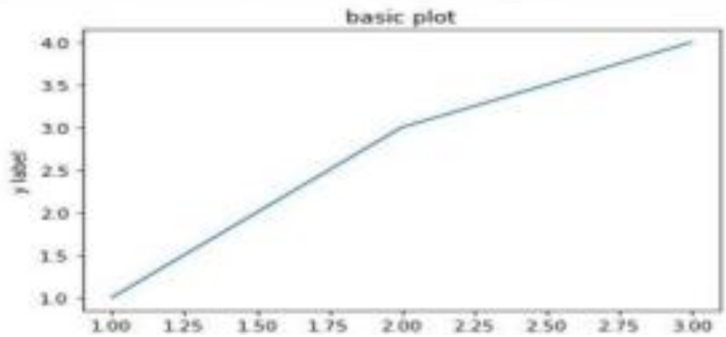


Figure 1.3.1: Matplotlib basic example

Pandas

Pandas is also a library or a data analysis tool in python which is written in python program- ming language. It is mostly used for data analysis and data manipulation. It is also used for data structures and time series.

We can see the application of python in many fields such as - Economics, Recommendation Sys- tems - Spotify, Netflix and Amazon, Stock Prediction, Neuro science, Statistics, Advertising, Analytics, Natural Language Processing. Data can be analyzed in pandas in two ways -

Data frames - In this data is two dimensional and consist of multiple series. Data is always represented in rectangular table.

Series - In this data is one dimensional and consist of single list with index.

```
#SERIES
import pandas as pd
#We are first importing the Librabry
#and keeping its name as 'pd' for our convenience
odd_numbers = pd.Series([3,9,13,15])
#we have imported series array from pandas
odd_numbers
```

```
0     3
1     9
2    13
3    15
dtype: int64
```

```
#DATAFRAME - IT HAS TWO OR MORE ARRAYS IN IT
info_stu = {'students':['john','mike','harry','robert'],
      'age':[28,26,29,25],
      'country':['spain','rome','holand','russia']}
#here we have defind our 3 differnet arrays

#then we have imported 'DataFrame' from pandas

info =pd.DataFrame(info_stu)

info

#0,1,2,3 are the index number of different rows
```

	students	age	country
0	john	28	spain

Figure 1.3.2: series and data frame in pandas

NumPy

”NumPy is a library for the Python programming language, adding support for large, multi- dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays”. The previous similar programming of NumPy is Numeric, and this language was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. [12] It is an opensource library and free of cost.

```
import numpy as np  #we have imported numpy library

data = np.arange(60)
#arange is the range of the array function
#we called arange function from numpy libraby
data.shape = (10,6)
#there will be 10 rows and 6 columns in array
#now we have given the number of rows and columns
print(data) #print the data
print(len(data)) #gives length of array
print(data.ndim) #gives dimension of array
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]
```

Figure 1.3.3: NumPy basic example

CHAPTER 4: APPROACH USED (REQUIRED TOOLS)

- **Decision Tree:** A decision tree is a type of supervised machine learning used to categorize or make predictions based on how a previous set of questions were answered.
- **KNN Algorithm:** K-NN algorithm stores all the available data and classifies a new data point based on the similarity.
- **Linear Regression :** A machine learning technique called linear regression uses a linear equation to express the linear relationship between one or more input data and a continuous output variable to predict the latter.
- **Ridge And Lasso:** While Lasso (Least Absolute Shrinkage and Selection Operator) Regression adds a penalty term to the cost function to reduce overfitting by setting some coefficients to zero, thereby performing feature selection, Ridge Regression adds a penalty term to the cost function to reduce overfitting by shrinking coefficients towards zero.
- **RandomForest Regressor:** Random Forest Regressor in Python is a supervised learning algorithm that combines multiple decision trees to predict a continuous output variable. With the aid of methods like fit, predict, and score, it can be applied to regression tasks.
- **Gradient Boosting Regressor** : is a supervised learning algorithm that combines multiple weak models to create a strong predictive model, iteratively training each tree to correct the errors of the previous tree

REQUIRED TOOLS:

For application development, the following Software Requirements are:
Operating System: Windows 11

Language: python

Tools: JUPYTER notebook or COLAB, Microsoft Excel (Optional).
Technologies used: python.

CHAPTER 5: RESULTS

This Notebook will cover -

1. Exploratory Data Analysis
2. Data Modelling and Evaluation

Import Laibraries

```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, Lasso

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

import warnings
warnings.filterwarnings('ignore')
print('Done')
```

Done

Import Data

```python
df = pd.read_csv('amazon_delivery.csv')
df.head()
```

	Order_ID	Agent_Age	Agent_Rating	Store_Latitude	Store_Longitude	Drop_Latitude	Drop_Longitude	Order_Date	Order_Time	Pickup_Time	Weather
0	ialx566343618	37	4.9	22.745049	75.892471	22.765049	75.912471	19-03-2022	11:30:00	11:45:00	Sunny
1	akqg208421122	34	4.5	12.913041	77.683237	13.043041	77.813237	25-03-2022	19:45:00	19:50:00	Stormy
2	njpu434582536	23	4.4	12.914264	77.678400	12.924264	77.688400	19-03-2022	08:30:00	08:45:00	Sandstorms
3	rjto796129700	38	4.7	11.003669	76.976494	11.053669	77.026494	05-04-2022	18:00:00	18:10:00	Sunny
4	zguw716275638	32	4.6	12.972793	80.249982	13.012793	80.289982	26-03-2022	13:30:00	13:45:00	Cloudy

Traffic	Vehicle	Area	Delivery_Time	Category
High	motorcycle	Urban	120	Clothing
Jam	scooter	Metropolitian	165	Electronics
Low	motorcycle	Urban	130	Sports
Medium	motorcycle	Metropolitian	105	Cosmetics
High	scooter	Metropolitian	150	Toys

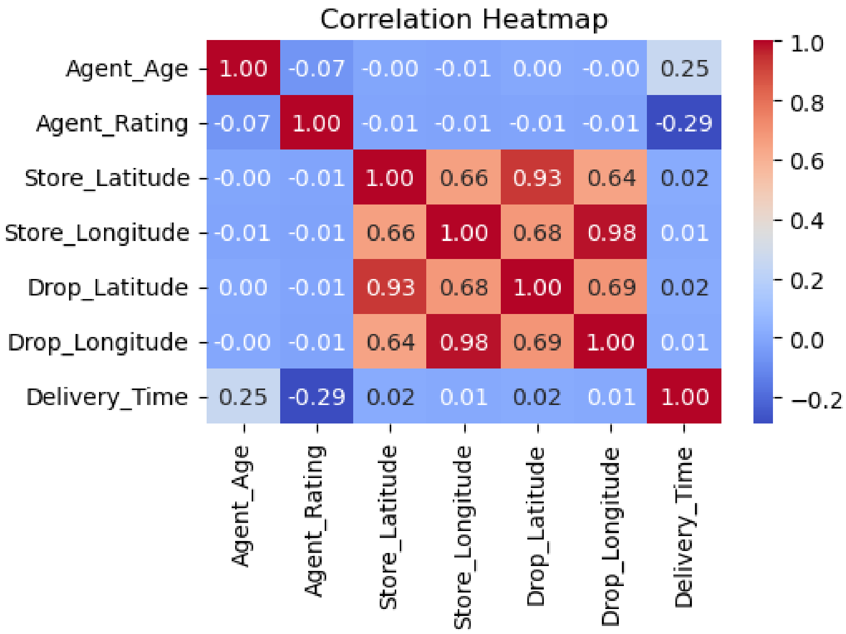
Data Overview

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43739 entries, 0 to 43738
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Order_ID         43739 non-null  object 
 1   Agent_Age        43739 non-null  int64  
 2   Agent_Rating     43685 non-null  float64
 3   Store_Latitude   43739 non-null  float64
 4   Store_Longitude  43739 non-null  float64
 5   Drop_Latitude    43739 non-null  float64
 6   Drop_Longitude   43739 non-null  float64
 7   Order_Date       43739 non-null  object 
 8   Order_Time       43739 non-null  object 
 9   Pickup_Time      43739 non-null  object 
 10  Weather          43648 non-null  object 
 11  Traffic          43739 non-null  object 
 12  Vehicle          43739 non-null  object 
 13  Area             43739 non-null  object 
 14  Delivery_Time    43739 non-null  int64  
 15  Category         43739 non-null  object 
dtypes: float64(5), int64(2), object(9)
memory usage: 5.3+ MB
```

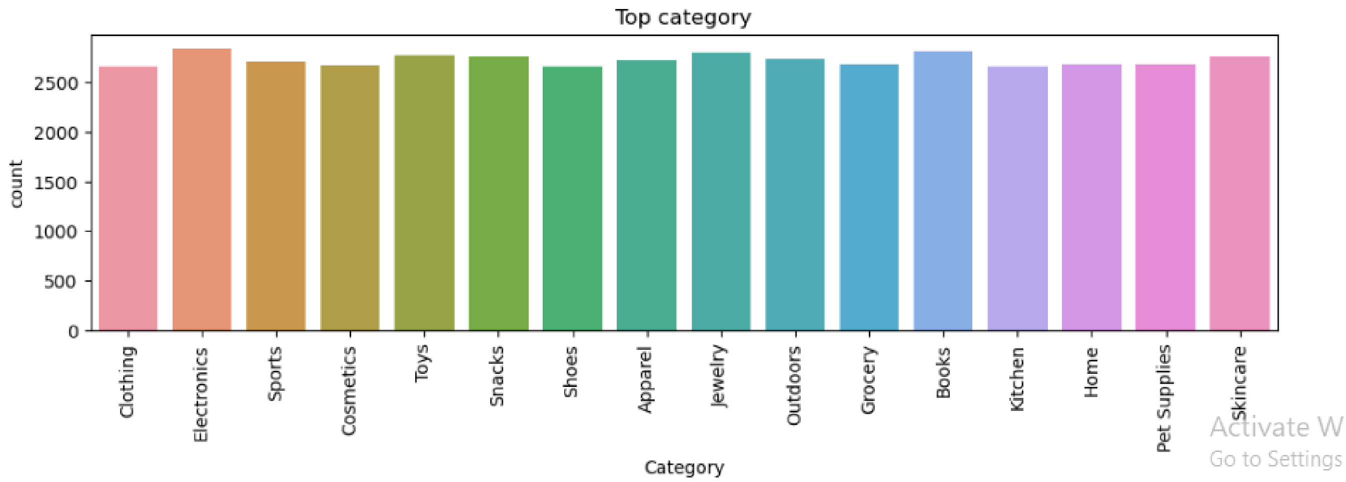
Correlation Heatmap

```python
# Correlation heatmap
numeric_df = df.select_dtypes(include=[np.number])
plt.figure(figsize=(8, 5))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



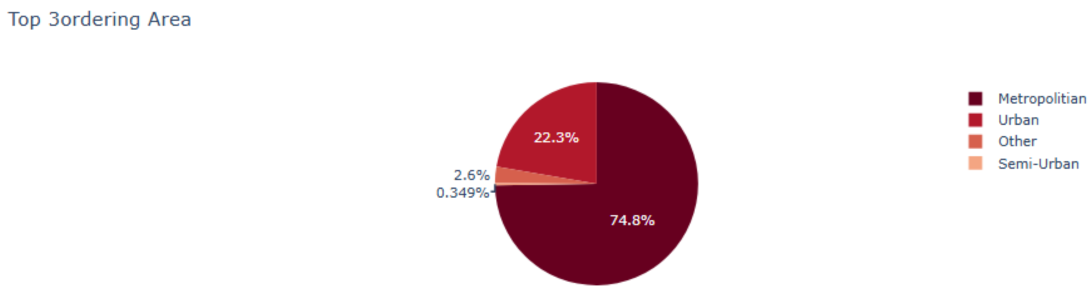
Top Categories

```
#Top Categories
plt.figure(figsize=(12, 3))
sns.countplot(x="Category",data=df)
plt.title('Top category')
plt.xlabel('Category')
plt.ylabel('count')
plt.xticks(rotation=90)
plt.show()
```



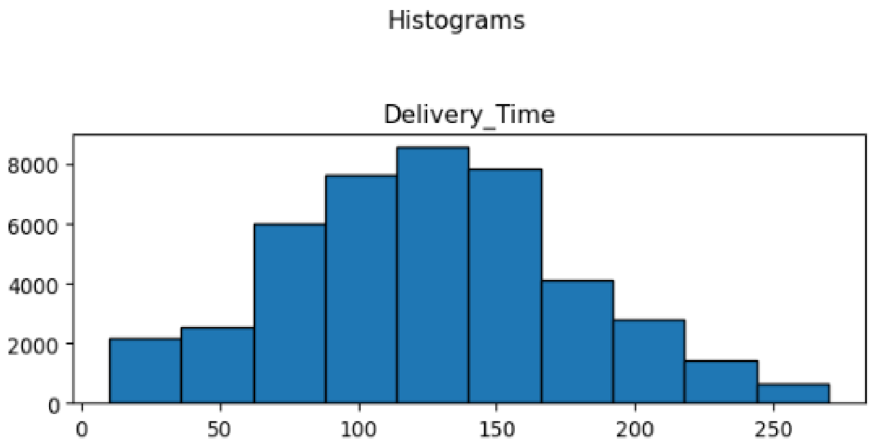
Top 3 Ordering Area

```
import plotly.express as px
top_Area = df.groupby('Area').size().reset_index().rename(columns={0: 'Total'}).sort_values('Total', ascending=False).head()
fig = px.pie(top_Area, values='Total', names='Area', color_discrete_sequence=px.colors.sequential.RdBu, title='Top 3ordering Area')
fig.show()
```

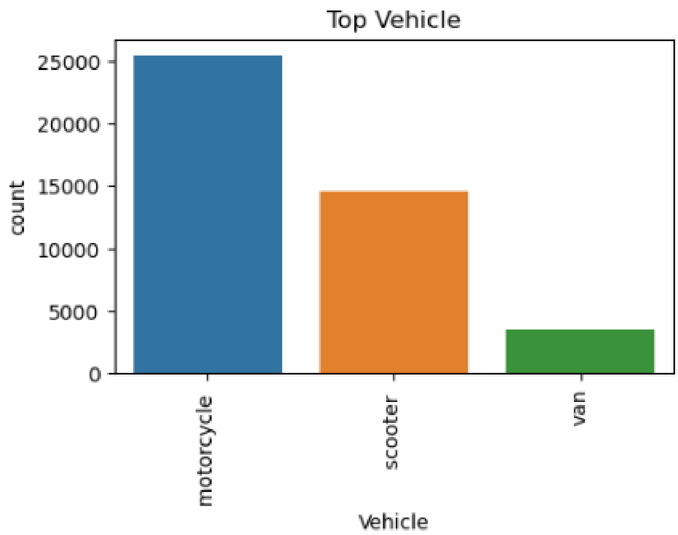


Delivery Time And Vehicle Graph

```
df.hist(column=['Delivery_Time'],figsize=(7,8),layout=(3,1),grid=False,edgecolor='black')
plt.suptitle('Histograms')
plt.show()
```




```
plt.figure(figsize=(5, 3))
sns.countplot(x="Vehicle",data=df)
plt.title('Top Vehicle')
plt.xlabel('Vehicle')
plt.ylabel('count')
plt.xticks(rotation=90)
plt.show()
```



Data Cleaning

```
import seaborn as sns
```

```
# Set the earth's radius (in kilometers)
R = 6371

# Convert degrees to radians
def deg_to_rad(degrees):
    return degrees * (np.pi/180)

# Function to calculate the distance between two points using the haversine formula
def distcalculate(lat1, lon1, lat2, lon2):
    d_lat = deg_to_rad(lat2-lat1)
    d_lon = deg_to_rad(lon2-lon1)
    a = np.sin(d_lat/2)**2 + np.cos(deg_to_rad(lat1)) * np.cos(deg_to_rad(lat2)) * np.sin(d_lon/2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
    return R * c

# Calculate the distance between each pair of points
df['Distance'] = np.nan

for i in range(len(df)):
    df.loc[i, 'Distance'] = distcalculate(df.loc[i, 'Store_Latitude'],
                                    df.loc[i, 'Store_Longitude'],
                                    df.loc[i, 'Drop_Latitude'],
                                    df.loc[i, 'Drop_Longitude'])
```

```
df['Agent_Age'].unique()
```

```
array([37, 34, 23, 38, 32, 22, 33, 35, 36, 21, 24, 29, 25, 31, 27, 26, 20,
       28, 39, 30, 15, 50], dtype=int64)
```

```
df['Agent_Rating'].unique()
```

```
array([4.9, 4.5, 4.4, 4.7, 4.6, 4.8, 4.2, 4.3, 4. , 4.1, 5. , 3.5, 3.8,
       nan, 3.9, 3.7, 2.6, 2.5, 3.6, 3.1, 2.7, 1. , 3.2, 3.3, 6. , 3.4,
       2.8, 2.9, 3. ])
```

```
import pandas as pd

# Specify the allowed rating values
allowed_ratings =[4.9, 4.5, 4.4, 4.7, 4.6, 4.8, 4.2, 4.3, 4.0, 4.1, 5.0, 3.5, 3.8, 3.9, 3.7, 2.6, 2.5, 3.6, 3.1, 2.7, 1.0, 3.2, 3.3, 3.4, 2.8, 2.9, 3.0]

# Filter the DataFrame to keep only the allowed rating values
df = df[df['Agent_Rating'].isin(allowed_ratings)]
```

```
df['Agent_Rating'].unique()
```

```
array([4.9, 4.5, 4.4, 4.7, 4.6, 4.8, 4.2, 4.3, 4. , 4.1, 5. , 3.5, 3.8,
       3.9, 3.7, 2.6, 2.5, 3.6, 3.1, 2.7, 1. , 3.2, 3.3, 3.4, 2.8, 2.9,
       3. ])
```

```
print(df['Weather'].unique())
print(df['Area'].unique())
print(df['Traffic'].unique())
print(df['Vehicle'].unique())
print(df['Category'].unique())
```

```
['Sunny' 'Stormy' 'Sandstorms' 'Cloudy' 'Fog' 'Windy' nan]
['Urban ' 'Metropolitian ' 'Semi-Urban ' 'Other']
['High ' 'Jam ' 'Low ' 'Medium ' 'NaN ']
['motorcycle ' 'scooter ' 'van' 'bicycle ']
['Clothing' 'Electronics' 'Sports' 'Cosmetics' 'Toys' 'Snacks' 'Shoes'
 'Apparel' 'Jewelry' 'Outdoors' 'Grocery' 'Books' 'Kitchen' 'Home'
 'Pet Supplies' 'Skincare']
```

```
import pandas as pd

# Specify the allowed rating values
allowed_traffic = ['High ','Jam ','Low ','Medium ']

df = df[df['Traffic'].isin(allowed_traffic)]
```

```
#find null value
df.isnull().sum()
```

```
Order_ID           0
Agent_Age          0
Agent_Rating       0
Store_Latitude     0
Store_Longitude    0
Drop_Latitude      0
Drop_Longitude     0
Order_Date         0
Order_Time         0
Pickup_Time        0
Weather            0
Traffic            0
Vehicle            0
Area               0
Delivery_Time      0
Category           0
Distance           0
dtype: int64
```

Trying different models with different features

Linear Regression

```
df['Agent_Rating'].fillna(df['Agent_Rating'].mode()[0], inplace=True)
df['Weather'].fillna(df['Weather'].mode()[0], inplace=True)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
```

```
X = df[['Agent_Age','Agent_Rating','Distance']]
y=df['Delivery_Time']
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

```
LR = LinearRegression()
```

```
LR.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
predictions = LR.predict(X_test)
```

```
print(mean_squared_error(y_test,predictions))
```

```
2263.3657626532604
```

```
import math
math.sqrt(2263.365762653255)
```

```
47.57484380061857
```

```
r2 = r2_score(y_test, predictions)
print(f"R-squared: {r2:.2f}")
```

```
R-squared: 0.14
```

Mapping to enhance model and checking the linear accuracy

```
print(df['Weather'].unique())
print(df['Area'].unique())
print(df['Traffic'].unique())
print(df['Vehicle'].unique())
print(df['Category'].unique())
```

```
['Sunny' 'Stormy' 'Sandstorms' 'Cloudy' 'Fog' 'Windy']
['Urban ' 'Metropolitian ' 'Semi-Urban ' 'Other']
['High ' 'Jam ' 'Low ' 'Medium ']
['motorcycle ' 'scooter ' 'van']
['Clothing' 'Electronics' 'Sports' 'Cosmetics' 'Toys' 'Snacks' 'Shoes'
 'Apparel' 'Jewelry' 'Outdoors' 'Grocery' 'Books' 'Kitchen' 'Home'
 'Pet Supplies' 'Skincare']
```

Mapping Weather column

```
weather_unique = df['Weather'].unique()

# Create a mapping dictionary to replace the unique values
weather_mapping = {weather: i for i, weather in enumerate(weather_unique)}

# Replace the unique values with the mapped values
df['Weather_encoded'] = df['Weather'].map(weather_mapping)

# Now you can use the encoded weather column as a feature in your linear regression model
X = df[['Weather_encoded']]
y = df['Delivery_Time']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print(math.sqrt(mean_squared_error(y_test,predictions)))
```

```
50.75275823292387
```

Mapping Area Column

```
area_unique = df['Area'].unique()

# Create a mapping dictionary to replace the unique values
area_mapping = {area: i for i, area in enumerate(area_unique)}

# Replace the unique values with the mapped values
df['Area_encoded'] = df['Area'].map(area_mapping)

# Now you can use the encoded area column as a feature in your linear regression model
X = df[['Area_encoded']]
y = df['Delivery_Time']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print(math.sqrt(mean_squared_error(y_test,predictions)))
```

```
50.98928151196163
```

Mapping Traffic Column

```
traffic_unique = df['Traffic'].unique()

# Create a mapping dictionary to replace the unique values
traffic_mapping = {traffic: i for i, traffic in enumerate(traffic_unique)}

# Replace the unique values with the mapped values
df['Traffic_encoded'] = df['Traffic'].map(traffic_mapping)

# Now you can use the encoded traffic column as a feature in your linear regression model
X = df[['Traffic_encoded']]
y = df['Delivery_Time']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print(math.sqrt(mean_squared_error(y_test,predictions)))
```

```
50.73385289812991
```

Mapping Vehicle Column

```
vehicle_unique = df['Vehicle'].unique()

# Create a mapping dictionary to replace the unique values
vehicle_mapping = {vehicle: i for i, vehicle in enumerate(vehicle_unique)}

# Replace the unique values with the mapped values
df['Vehicle_encoded'] = df['Vehicle'].map(vehicle_mapping)

# Now you can use the encoded traffic column as a feature in your linear regression model
X = df[['Vehicle_encoded']]
y = df['Delivery_Time']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print(math.sqrt(mean_squared_error(y_test,predictions)))
```

```
50.805125843238606
```

Mapping Category Column

```
category_unique = df['Category'].unique()

# Create a mapping dictionary to replace the unique values
category_mapping = {category: i for i, category in enumerate(category_unique)}

# Replace the unique values with the mapped values
df['Category_encoded'] = df['Category'].map(category_mapping)

# Now you can use the encoded category column as a feature in your linear regression model
X = df[['Category_encoded']]
y = df['Delivery_Time']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print(math.sqrt(mean_squared_error(y_test,predictions)))
```

```
51.214578978412334
```

Check Mapping Column info

```
#checking data after mapping
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 43594 entries, 0 to 43738
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Order_ID          43594 non-null  object 
 1   Agent_Age         43594 non-null  int64  
 2   Agent_Rating      43594 non-null  float64
 3   Store_Latitude    43594 non-null  float64
 4   Store_Longitude   43594 non-null  float64
 5   Drop_Latitude     43594 non-null  float64
 6   Drop_Longitude    43594 non-null  float64
 7   Order_Date        43594 non-null  object 
 8   Order_Time        43594 non-null  object 
 9   Pickup_Time       43594 non-null  object 
 10  Weather           43594 non-null  object 
 11  Traffic           43594 non-null  object 
 12  Vehicle           43594 non-null  object 
 13  Area              43594 non-null  object 
 14  Delivery_Time     43594 non-null  int64  
 15  Category          43594 non-null  object 
 16  Distance          43594 non-null  float64
 17  Weather_encoded   43594 non-null  int64  
 18  Area_encoded      43594 non-null  int64  
 19  Traffic_encoded   43594 non-null  int64  
 20  Vehicle_encoded   43594 non-null  int64  
 21  Category_encoded  43594 non-null  int64  
dtypes: float64(6), int64(7), object(9)
memory usage: 7.6+ MB
```

Some more trial models

Linear Regression Model

```
from sklearn.preprocessing import StandardScaler

# Feature variables
X = df[['Agent_Age', 'Agent_Rating', 'Distance',
        'Weather_encoded', 'Area_encoded', 'Traffic_encoded',
        'Vehicle_encoded', 'Category_encoded']]

# Target variable
y = df['Delivery_Time']

# 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=42)

# Standardize the feature variables
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the linear regression model
model = LinearRegression()
model.fit(X_train_scaled, y_train)
```

```
# Make predictions on the test set
y_pred = model.predict(X_test_scaled)

# Evaluate the model
mae_lr = mean_absolute_error(y_test, y_pred)
mse_lr = mean_squared_error(y_test, y_pred)
rmse_lr = np.sqrt(mse_lr)
r2_lr = r2_score(y_test, y_pred)
accuracy_lr = 100 - rmse_lr

print(f'RMSE : {rmse_lr:.2f}')
print(f'MAE : {mae_lr:.2f}')
print(f'MSE : {mse_lr:.2f}')
print(f'RMSE : { rmse_lr :.2f}')
print(f'R2 : {r2_lr:.2f}')
print(f'Accuracy : {accuracy_lr:.2f}%')
```

```
RMSE : 45.77
MAE : 35.31
MSE : 2094.79
RMSE : 45.77
R2 : 0.20
Accuracy : 54.23%
```

Decision Tree Regressor Model

```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mae_dt = mean_absolute_error(y_test, y_pred)
mse_dt = mean_squared_error(y_test, y_pred)
rmse_dt = np.sqrt(mse_dt)
r2_dt = r2_score(y_test, y_pred)
accuracy_dt = 100 - rmse_dt

print(f'RMSE : {rmse_dt:.2f}')
print(f'MAE : {mae_dt:.2f}')
print(f'MSE : {mse_dt:.2f}')
print(f'RMSE : { rmse_dt :.2f}')
print(f'R2 : {r2_dt:.2f}')
print(f'Accuracy : {accuracy_dt:.2f}%')
```

```
RMSE : 31.32
MAE : 23.16
MSE : 980.70
RMSE : 31.32
R2 : 0.63
Accuracy : 68.68%
```

Ridge Regression Model ¶

```
from sklearn.linear_model import Ridge

model = Ridge(alpha=1.0)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mae_rg = mean_absolute_error(y_test, y_pred)
mse_rg = mean_squared_error(y_test, y_pred)
rmse_rg = np.sqrt(mse_rg)
r2_rg = r2_score(y_test, y_pred)
accuracy_rg = 100 - rmse_rg

print(f'RMSE : {rmse_rg:.2f}')
print(f'MAE : {mae_rg:.2f}')
print(f'MSE : {mse_rg:.2f}')
print(f'RMSE : { rmse_rg :.2f}')
print(f'R2 : {r2_rg:.2f}')
print(f'Accuracy : {accuracy_rg:.2f}%')
```

```
RMSE : 45.77
MAE : 35.31
MSE : 2094.79
RMSE : 45.77
R2 : 0.20
Accuracy : 54.23%
```

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor

# Train the Random Forest Ridge Regression Model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mae_rf = mean_absolute_error(y_test, y_pred)
mse_rf = mean_squared_error(y_test, y_pred)
rmse_rf = np.sqrt(mse_rf)
r2_rf = r2_score(y_test, y_pred)
accuracy_rf = 100 - rmse_rf

print(f'RMSE : {rmse_rf:.2f}')
print(f'MAE : {mae_rf:.2f}')
print(f'MSE : {mse_rf:.2f}')
print(f'RMSE : { rmse_rf :.2f}')
print(f'R2 : {r2_rf:.2f}')
print(f'Accuracy : {accuracy_rf:.2f}%')
```

```
RMSE : 23.28
MAE : 17.90
MSE : 541.91
RMSE : 23.28
R2 : 0.79
Accuracy : 76.72%
```

Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor

# Train the Gradient Boosting Regression Model
model = GradientBoostingRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mae_gb = mean_absolute_error(y_test, y_pred)
mse_gb = mean_squared_error(y_test, y_pred)
rmse_gb = np.sqrt(mse_gb)
r2_gb = r2_score(y_test, y_pred)
accuracy_gb = 100 - rmse_gb

print(f'RMSE : {rmse_gb:.2f}')
print(f'MAE : {mae_gb:.2f}')
print(f'MSE : {mse_gb:.2f}')
print(f'RMSE : { rmse_gb :.2f}')
print(f'R2 : {r2_gb:.2f}')
print(f'Accuracy : {accuracy_gb:.2f}%')
```

```
RMSE : 25.06
MAE : 19.77
MSE : 628.17
RMSE : 25.06
R2 : 0.76
Accuracy : 74.94%
```

K Neighbours Regressor

```
from sklearn.neighbors import KNeighborsRegressor

# Train the K Neighbour Ridge Regression Model
model = KNeighborsRegressor(n_neighbors=5)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mae_knn = mean_absolute_error(y_test, y_pred)
mse_knn = mean_squared_error(y_test, y_pred)
rmse_knn = np.sqrt(mse_knn)
r2_knn = r2_score(y_test, y_pred)
accuracy_knn = 100 - rmse_knn

print(f'RMSE : {rmse_knn:.2f}')
print(f'MAE : {mae_knn:.2f}')
print(f'MSE : {mse_knn:.2f}')
print(f'RMSE : { rmse_knn :.2f}')
print(f'R2 : {r2_knn:.2f}')
print(f'Accuracy : {accuracy_knn:.2f}%')
```

```
RMSE : 35.27
MAE : 27.38
MSE : 1244.27
RMSE : 35.27
R2 : 0.53
Accuracy : 64.73%
```

Finding Best Model

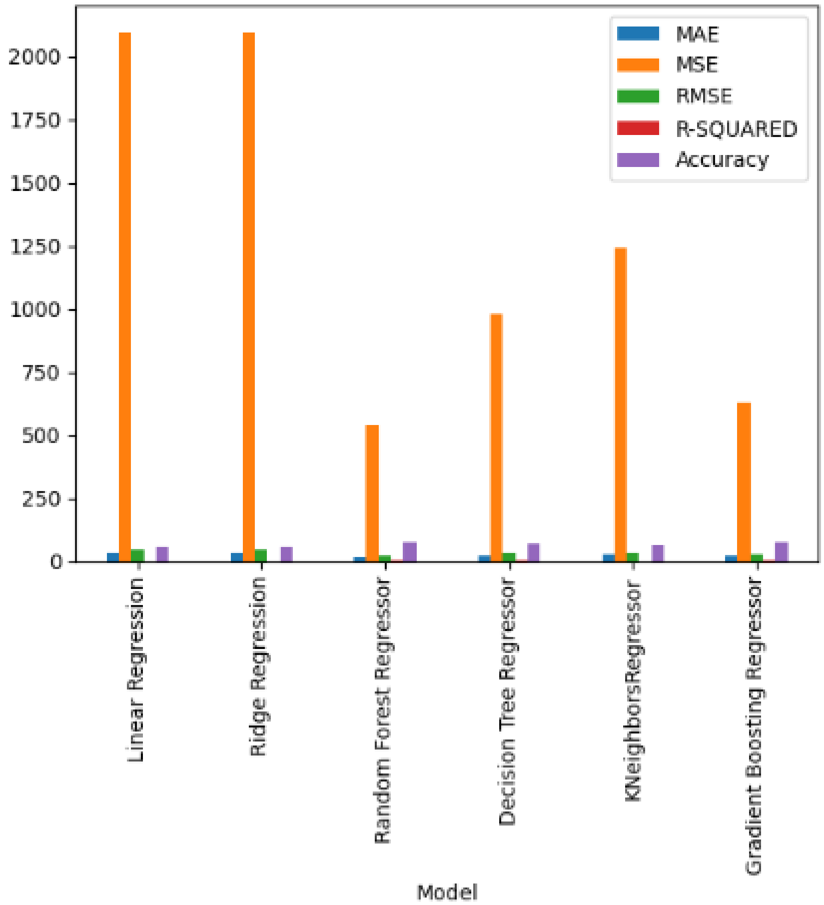
```
model_info = {
    'Model': ['Linear Regression','Ridge Regression', 'Random Forest Regressor', 'Decision Tree Regressor',
              'KNeighborsRegressor','Gradient Boosting Regressor'],
    'MAE': [mae_lr, mae_rg, mae_rf, mae_dt,mae_knn,mae_gb],
    'MSE': [mse_lr, mse_rg, mse_rf, mse_dt,mse_knn,mse_gb],
    'RMSE': [rmse_lr, rmse_rg, rmse_rf, rmse_dt,rmse_knn,rmse_gb],
    'R-SQUARED': [r2_lr, r2_rg, r2_rf, r2_dt,r2_knn,r2_gb],
    'Accuracy':[accuracy_lr, accuracy_rg, accuracy_rf, accuracy_dt,accuracy_knn,accuracy_gb],
}

performance = pd.DataFrame(model_info)
performance
```

	Model	MAE	MSE	RMSE	R-SQUARED	Accuracy
0	Linear Regression	35.312763	2094.788560	45.768860	0.203448	54.231140
1	Ridge Regression	35.313002	2094.789143	45.768867	0.203448	54.231133
2	Random Forest Regressor	17.898698	541.913945	23.279045	0.793935	76.720955
3	Decision Tree Regressor	23.160053	980.700568	31.316139	0.627085	68.683861
4	KNeighborsRegressor	27.381351	1244.274619	35.274277	0.526860	64.725723
5	Gradient Boosting Regressor	19.770975	628.166018	25.063240	0.761137	74.936760

```python
#Check Performace of the models
performance[performance['R-SQUARED'] == performance['R-SQUARED'].max()]
```

	Model	MAE	MSE	RMSE	R-SQUARED	Accuracy
2	Random Forest Regressor	17.898698	541.913945	23.279045	0.793935	76.720955

```python
#find best model by graph
performance.plot(kind='bar', x= 'Model')
plt.show()
```



Final Model

```python
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
accuracy = 100 - rmse

print(f"RMSE: {rmse:.2f}")
print(f"Accuracy: {accuracy:.2f}%")
```

```
RMSE: 23.28
Accuracy: 76.72%
```

```python
#download best model
import joblib

joblib.dump(model, 'project_best_model.pkl')
```

```
['project_best_model.pkl']
```

```python
# use model for prediction
best_model = joblib.load('project_best_model.pkl')

new_data = np.array([[7.2574, 52.0, 8.288136, 1.073446, 496.0, 2.802260, 37.85, -122.24]])

scaled_new_data = scaler.transform(new_data)


delivery_time = best_model.predict(scaled_new_data)

print('Predicted Delivery Time:', delivery_time)
```

```
Predicted Delivery Time: [77.75]
```

SUMMARY & CONCLUSIONS

- **Route Optimization**: The ML model optimizes delivery routes by considering factors such as traffic patterns, weather conditions, and delivery schedules. This minimizes delivery times and reduces operational costs.

- **Real-Time Adaptability**: The model adjusts dynamically to real-time data inputs, ensuring that deliveries are adjusted promptly in response to changing circumstances like traffic congestion or unexpected events.

- **Enhanced Customer Experience**: By predicting delivery times accurately and offering real-time updates, Amazon improves the overall customer experience. Customers benefit from reliable and timely deliveries, leading to higher satisfaction and loyalty.

REFERENCES & BIBLIOGRAPHY

www.simplilearn
www.kaggle.com
www.androidheadline
www.wikipedia.com
www.geeksofgeeks.com
www.javatpoint.com
www.edubca.com

GitHub Link for the Project File

https://github.com/Vishnugarg897/Projects-Report