Predicting Palate Perfection: Optimizing Food Delivery with Machine Learning

Project Hand Out - Sirige Venkata Mytresh Sai Gowd, VIT - AP University,
Email : mytresh.23bce7908@vitapstudent.ac.in

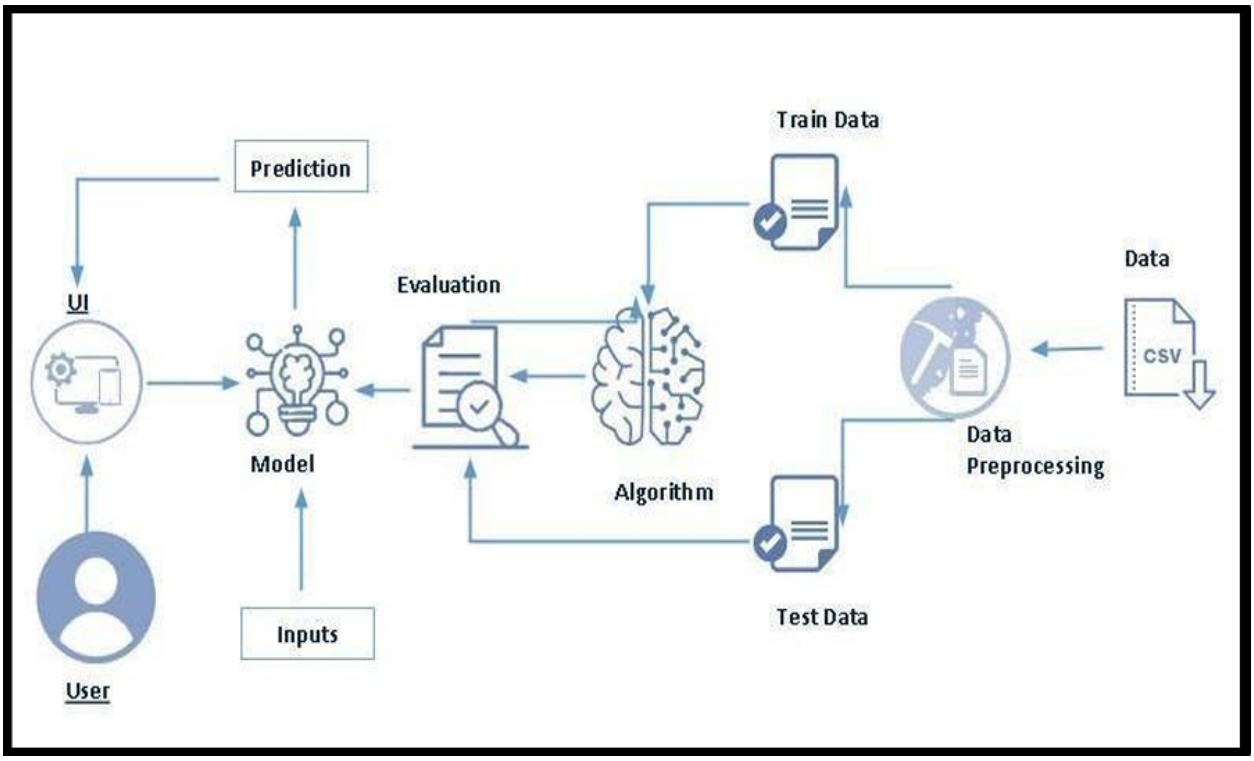
Predicting Palate Perfection: Optimizing Food Delivery with Machine Learning

Now a days, People are instead of going outside all are Ordering home deliveries, So it become a rise to the "Food Delivery Apps" such as "Swiggy, Zomato and Uber Eats" .

While growing it requires the service provided by the company also need to improve in terms of quality and delivery time etc., so here "Food Delivery" time will

plays the key role in terms of "Customers", If the delivery will be done in efficient time it will give the satisfaction to the customers, so the machine learning model will predict the "Time taken for the Delivery" so that the Employees can get the idea regarding the delivery.

Technical Architecture:



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
  - Specify the business problem
  - Business requirements
  - Literature Survey
  - Social or Business Impact.
- Data Collection & Preparation
  - Collect the dataset
  - Data Preparation

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

Prior Knowledge:

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

· ML Concepts

o Supervised learning: https://www.javatpoint.com/supervised-machine-learning

o Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning

· Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification-

algorithm

· Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm

· KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

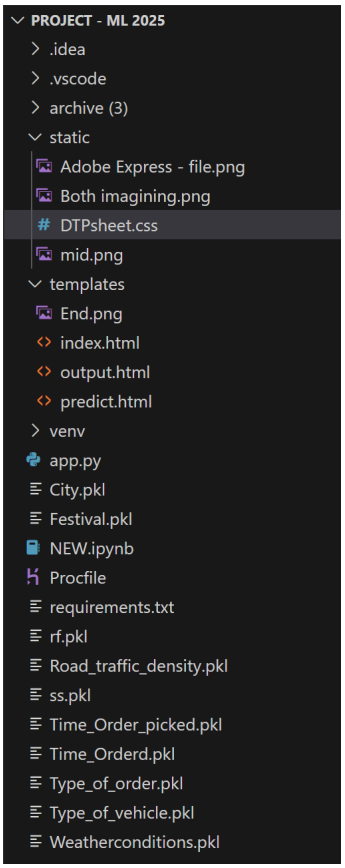
· Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/

· Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-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.
● rf.pkl is our saved model. Further we will use this model for flask integration.
● Data Folder contains the Dataset used
● The Notebook file contains procedure for building th model.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

In today's fast-paced world, food delivery services are increasingly relied upon. However, one of the major issues faced by such platforms is the **uncertainty in delivery time predictions**. Customers expect timely service, and businesses need better planning tools to manage logistics.

Our project, **Delivery Time Prediction using Machine Learning**, aims to accurately estimate the **expected delivery time (in minutes)** based on various real-time and historical factors like:

- Delivery person's age and rating
- Restaurant and delivery location coordinates
- Time of order and pickup
- Weather and traffic conditions
- Vehicle type, order type, and city

By building a robust ML model, we intend to **reduce delay complaints**, **optimize delivery**

Activity 2: Business requirements

A Delivery Time Prediction system may have the following business requirements:

- **Accuracy & Timeliness**: The model must provide delivery time estimates that are as close as possible to reality, considering all available parameters.
- **Scalability**: The system should scale to handle multiple deliveries in different locations under varying conditions.
- **Real-time Operation**: The application should allow input through a simple web interface and respond instantly with predictions.
- **Flexibility to New Data**: The model should be easily updatable as more training data or features become available (like holiday info or rider fatigue levels).
- **User-friendly Interface**: A clean and intuitive UI where managers or users can input delivery conditions and get predictions without technical knowledge.

Activity 3: Literature Survey

- Overview of existing food delivery prediction systems (e.g., Zomato, Swiggy)
- Comparison of ML models used (Random Forest, XGBoost, etc.)
- Studies on how features like traffic and weather impact delivery time
- Limitations in earlier models and how your model addresses those gaps

Activity 4: Social or Business Impact

Social Impact:

- **Enhanced Customer Experience**: Customers are more satisfied when they receive accurate delivery time estimates.
- **Transparency**: Builds trust in food delivery services by reducing false promises or delays.
- **Safety**: Helps avoid over-pressuring delivery persons by optimizing delivery schedules.

Business Impact:

- **Operational Efficiency**: Helps businesses allocate resources more effectively (e.g., how many orders a rider can manage).
- **Data-Driven Planning**: Business owners can use prediction data to analyze peak time performance.
- **Reduction in Complaints**: Predictive systems reduce the gap between promised and actual delivery times.

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/gauravmalik26/food-delivery-dataset?select=train.csv-insurance-claims-data

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.Here we have used visualisation style as fivethirtyeight.

```
[1]: #importing the required libraries, modules and functions

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import warnings
warnings.filterwarnings("ignore")
```

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.

- For checking the null values, df.isna().any( ) 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.

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 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

- For checking the null values, df.isna().any( ) 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.

[4]:
```
#filling the missing values by using mode

m1 = df['multiple_deliveries'].mode()[0]
df['multiple_deliveries'].fillna(m1,inplace = True)

m2 = df['Vehicle_condition'].mode()[0]
df['Vehicle_condition'].fillna(m2,inplace = True)
```

[5]:
```
md1 = df['Delivery_person_Age'].median()
df['Delivery_person_Age'].fillna(md1, inplace = True)

md2 = df['Delivery_person_Ratings'].median()
df['Delivery_person_Ratings'].fillna(md2, inplace = True)

md3 = df['Time_Orderd'].mode()[0]
df['Time_Orderd'].fillna(md3,inplace = True)

md4 = df['Weatherconditions'].mode()[0]
df['Weatherconditions'].fillna(md4, inplace = True)

md5 = df['Road_traffic_density'].mode()[0]
df['Road_traffic_density'].fillna(md5, inplace = True)

md7 = df['Festival'].mode()[0]
df['Festival'].fillna(md7,inplace = True)

md8 = df['City'].mode()[0]
df['City'].fillna(md8,inplace = True)

print(type(df['Delivery_person_Age'][0]))
```

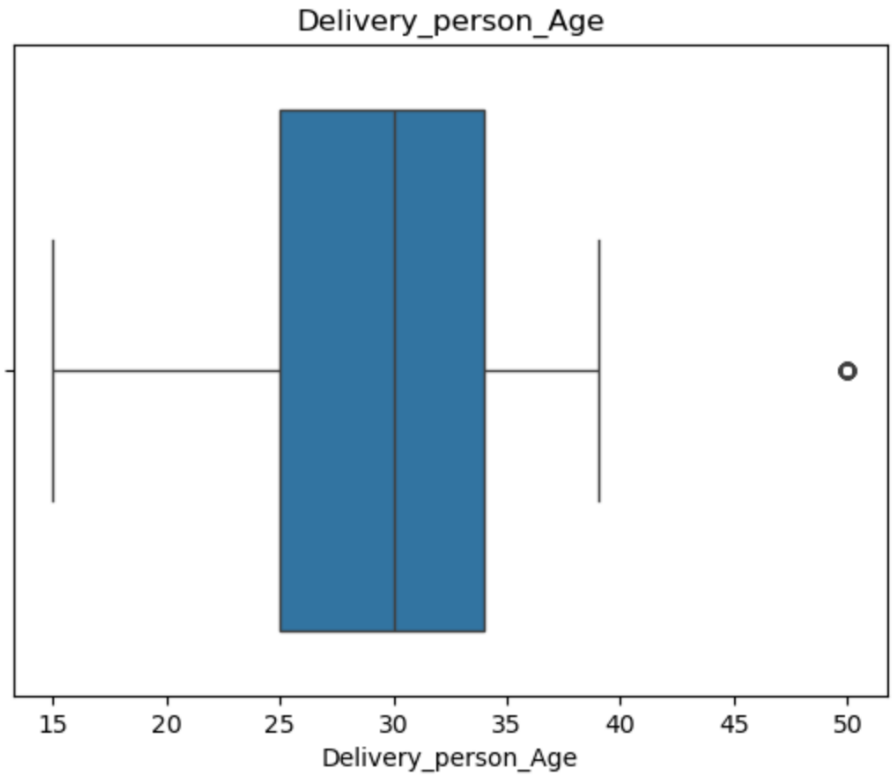
Activity 2.2: Handling Outliers

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

[6]:
```
#modifying Outliers:

#they are saying to use pandas to check the data type

for col in df.columns:
    if pd.api.types.is_numeric_dtype(df[col]):
        sns.boxplot(x = df[col])
        plt.title(col)
        plt.show()
```



From the below diagram, we could visualize that data has outliers. Boxplot from seaborn library is used here.

Activity 2.3: Dropping Unwanted Columns:

Dropping unwanted columns is a common data preprocessing step when working with datasets, especially in data analysis and machine learning projects. This process involves removing specific columns or features from your dataset that are not relevant or necessary for your analysis or modeling. Below is a description of how to drop unwanted columns in a dataset:

```
[ ] df.drop(['ID'],axis=1,inplace=True)
```

```
[ ] df.drop(['Delivery_person_ID'],axis=1,inplace=True)
```

```
[ ] df.drop(['Order_Date'],axis=1,inplace=True)
```

Activity 3: Visualizing And Analyzing The Data

Exploratory Data Analysis (EDA) is a crucial initial step in the data analysis process, where the main goal is to gain insights into the structure, relationships, and patterns within the data.

Activity 3.1: 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.

[8]:
```
#After refining the dataset now its info:

df.shape
```

[8]:
```
(45593, 17)
```

[9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45593 entries, 0 to 45592
Data columns (total 17 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Delivery_person_Age          45593 non-null  float64
 1   Delivery_person_Ratings      45593 non-null  float64
 2   Restaurant_latitude          45593 non-null  float64
 3   Restaurant_longitude         45593 non-null  float64
 4   Delivery_location_latitude   45593 non-null  float64
 5   Delivery_location_longitude  45593 non-null  float64
 6   Time_Orderd                  45593 non-null  object 
 7   Time_Order_picked            45593 non-null  object 
 8   Weatherconditions            45593 non-null  object 
 9   Road_traffic_density         45593 non-null  object 
 10  Vehicle_condition            45593 non-null  int64  
 11  Type_of_order                45593 non-null  object 
 12  Type_of_vehicle              45593 non-null  object 
 13  multiple_deliveries          45593 non-null  float64
 14  Festival                     45593 non-null  object 
 15  City                         45593 non-null  object 
 16  Time_taken(min)              45593 non-null  int64  
dtypes: float64(7), int64(2), object(8)
memory usage: 5.9+ MB
```

[10]:
```
df.describe()
```

[10]:

	Delivery_person_Age	Delivery_person_Ratings	Restaurant_latitude	Restaurant_longitude	Delivery
count	45593.000000	45593.000000	45593.000000	45593.000000	
mean	29.581833	4.636552	17.017729	70.231332	
std	5.686546	0.327906	8.185109	22.883647	
min	15.000000	1.000000	-30.905562	-88.366217	
25%	25.000000	4.600000	12.933284	73.170000	
50%	30.000000	4.700000	18.546947	75.898497	
75%	34.000000	4.800000	22.728163	78.044095	
max	47.500000	6.000000	30.914057	88.433452	

Activity 3.2: 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 3.2.1: Univariate Analysis:

In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as Piechart and countplot. A "histplot" typically refers to a histogram plot in the context of data visualization. A histogram is a graphical representation of the distribution of a dataset. It shows the frequency or count of data points falling into specific intervals or "bins" along the numerical range of the data. Histograms are particularly useful for understanding the underlying data distribution, identifying patterns.

[11]:
```
#Visualising the given data:

#numeric data in histograms
sns.histplot(df.Delivery_person_Age,bins= 20,kde=False,color = 'Red')
plt.title("Delivery Person Age Distribution")
plt.xlabel("Age Group")
plt.show()
```

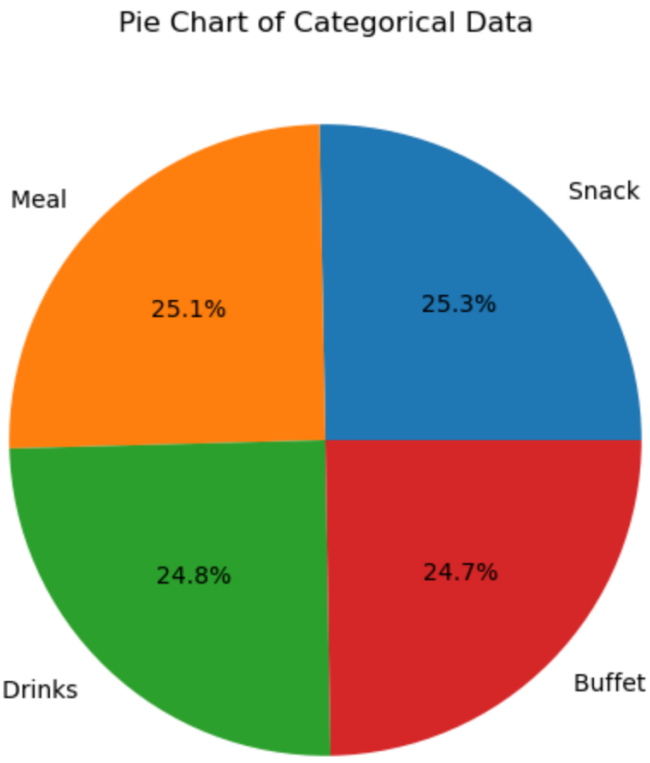
From The Histogram, Most of the People are in the Age group of 20 to 40.

A pie chart is a circular data visualization that is used to represent data in a way that illustrates the relative proportions or percentages of different categories within a whole. Each category is represented as a "slice" of the pie, with the size of each slice proportional to the value or percentage it represents. Pie charts are particularly useful for showing the composition of a whole and making it easy to compare the parts to the whole.

[12]:
```
#Categorical Data in Pie Chart:

frequency = df['Type_of_order'].value_counts()

plt.figure(figsize = (10,6))
plt.pie(frequency, labels = frequency.index, autopct = '%1.1f%%')
plt.title("Pie Chart of Categorical Data")
plt.show()
```



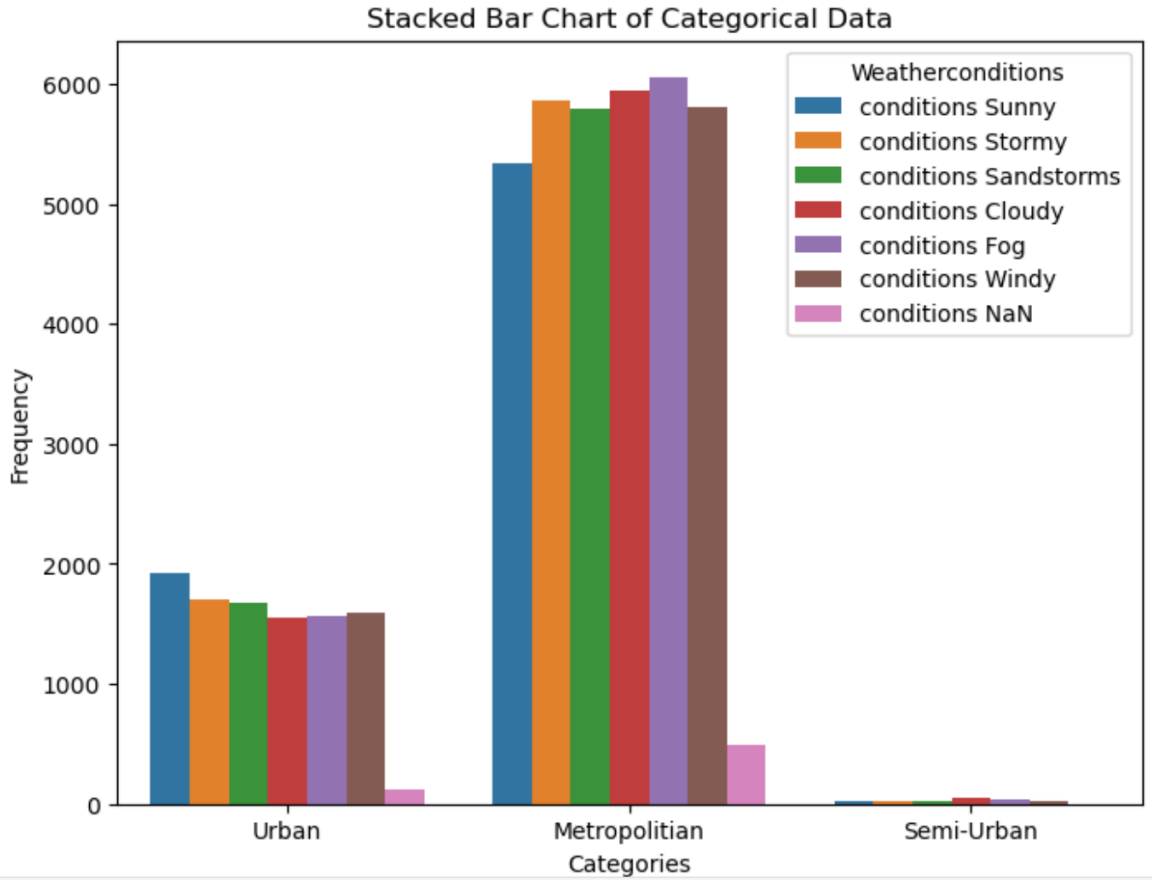
Activity 3.2.2: Bivariate Analysis:

To find the relation between two features we use bivariate analysis. Here we use Stacked Barchart, Scatterplot.
A stacked bar chart, also known as a stacked bar graph, is a data visualization that represents data in the form of rectangular bars.

Unlike a regular bar chart where the bars are side by side, in a stacked bar chart, the bars are stacked on top of each other, and each bar is divided into segments or subcategories.

[13]:
```
#Bivariate analysis Pictorial representation between two columns:

#Create a stacked bar plot
frequency = df['City'].value_counts()
plt.figure(figsize = (8,6))
sns.countplot(data = df, x = 'City', hue = 'Weatherconditions')
plt.xlabel('Categories')
plt.ylabel('Frequency')
plt.title('Stacked Bar Chart of Categorical Data')
plt.legend(title = 'Weatherconditions')
plt.show()
```



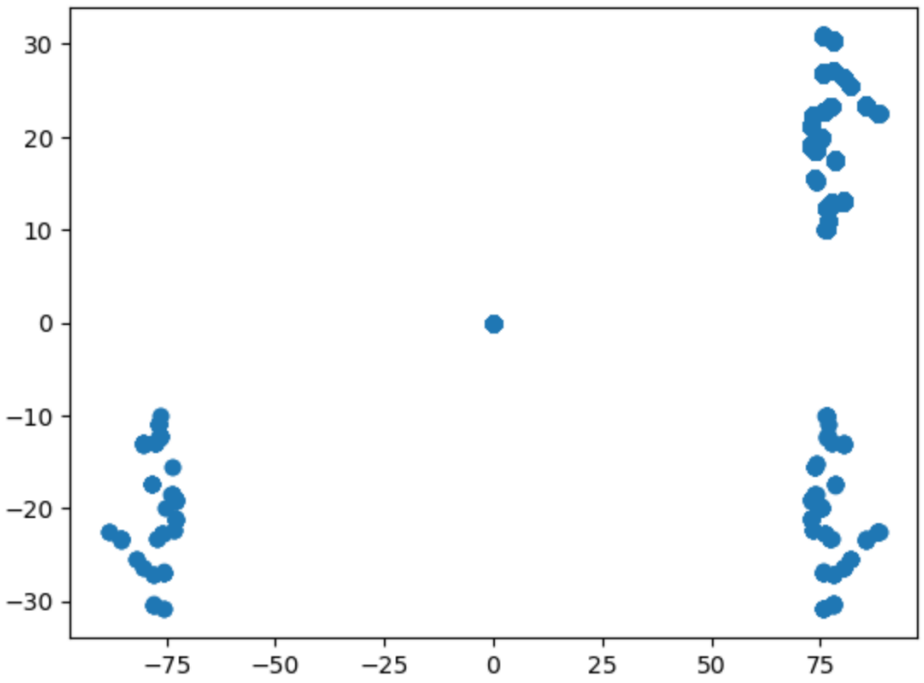
Scatter Plot:

A scatterplot is a type of data visualization used to display the relationship between two continuous variables or to visualize individual data points within a dataset. It is particularly useful for identifying patterns, trends, clusters, and outliers in data. Scatterplots consist of points, each representing a single data

observation, plotted on a Cartesian plane with one variable on the x-axis and another on the y-axis.

[14]:
```
#By using scatterplots we can use to show the clustering:
#Relation between latitude and longitude

plt.scatter(x = df['Restaurant_longitude'],y = df['Restaurant_latitude'])
plt.show()
```

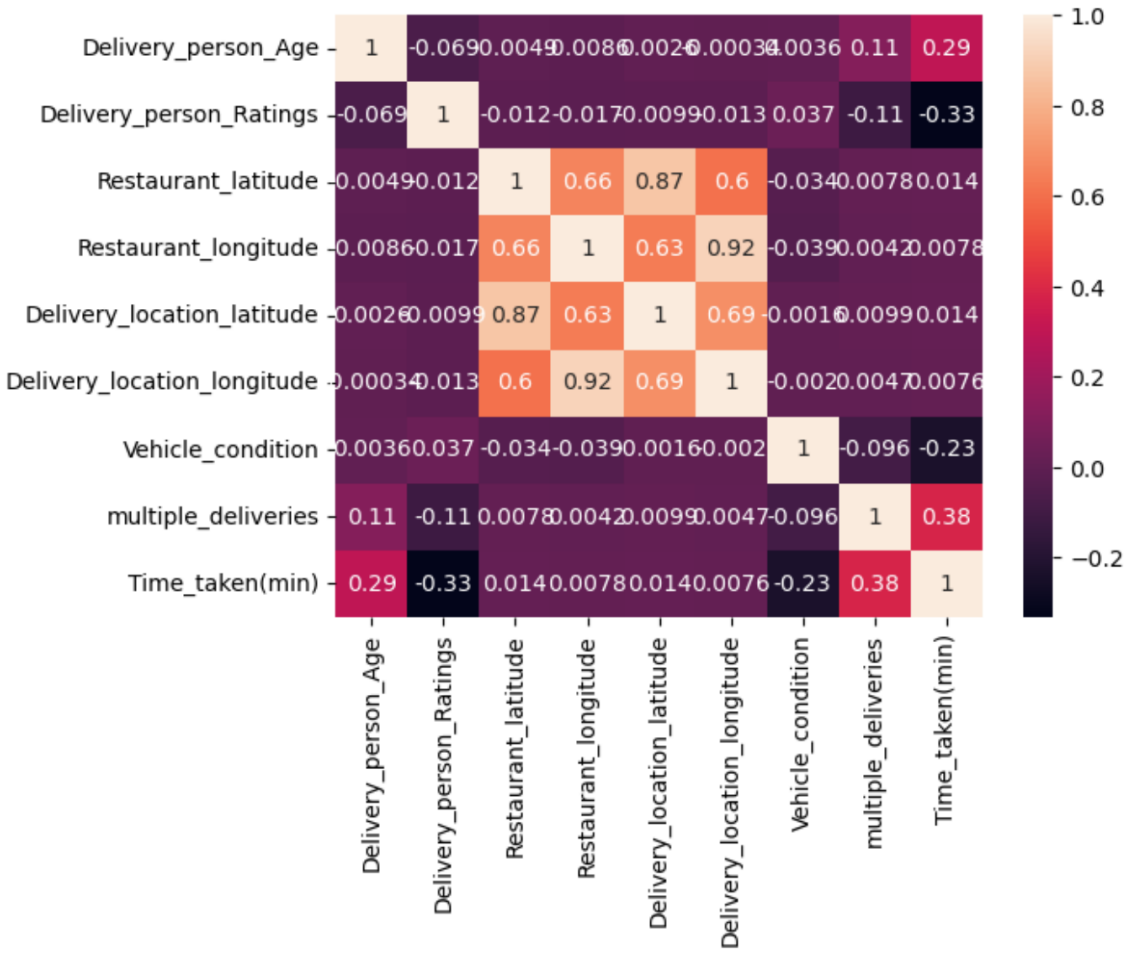


HeatMap:

A heatmap is a data visualization technique used to represent the data values in a matrix or a table using colors. It is particularly useful for displaying the magnitude or intensity of a variable in a two-dimensional format.

```
[15]: sns.heatmap(df.select_dtypes(include='number').corr(), annot=True)
```

```
[15]: <Axes: >
```



Activity 3.3: Feature Engineering:

Feature engineering is a crucial process in data science and machine learning that involves creating new, relevant, and informative features (variables) from the existing data to improve the performance of a machine learning model. Effective feature engineering can have a significant impact on model accuracy and predictive power.

Activity 3.3.1: Column Transformation:

In the Data set we have Restaurant Latitude, Restaurant Longitude, Delivery Location Latitude, Delivery Location Longitude. So We can calculate

the distance between The Restaurant and Delivery Location given their latitudes and longitudes, using the Haversine formula. The Haversine formula calculates the great-circle distance between two points on the surface of a sphere (like the Earth).

```
#Feature Engineering:

#Establishing the new methods to improve the efficiency of model:

#To find the distance using Haversine formula:

import math
#Define the Haversine formula function
def haversine(lat1, lon1, lat2, lon2):
    """Will Predict the distance based on latitudes and longitude"""
    #Convert Latitude and Longitude from degrees to radians

    lat1 = math.radians(lat1)
    lon1 = math.radians(lon1)
    lat2 = math.radians(lat2)
    lon2 = math.radians(lon2)

    #Radius of the Earth in kilometers
    radius = 6371.0

    #Haversine formulae
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = math.sin(dlat / 2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon / 2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    distance = radius * c

    return distance


#Calculate distances for each row
df['distance_km'] = df.apply(lambda row: haversine(row['Restaurant_latitude'], row['Restaurant_longitude'],  row['Delivery_location_latitude'], row['Deli
```

Here I have Created a New Column Named Distance_Km which represents the distance between the delivery Location and Restaurant and dropped the Restaurant Latitude, Restaurant Longitude, Delivery Location Latitude, Delivery Location Longitude.

Activity 3.3.2: Data Transformation:

Data transformation is a fundamental step in data preprocessing and preparation for various data analysis, machine learning, and modeling tasks. It involves modifying the raw data to make it suitable for analysis, model training, or visualization. Data transformation encompasses several techniques and processes, depending on the characteristics of the data and the specific goals of the analysis.

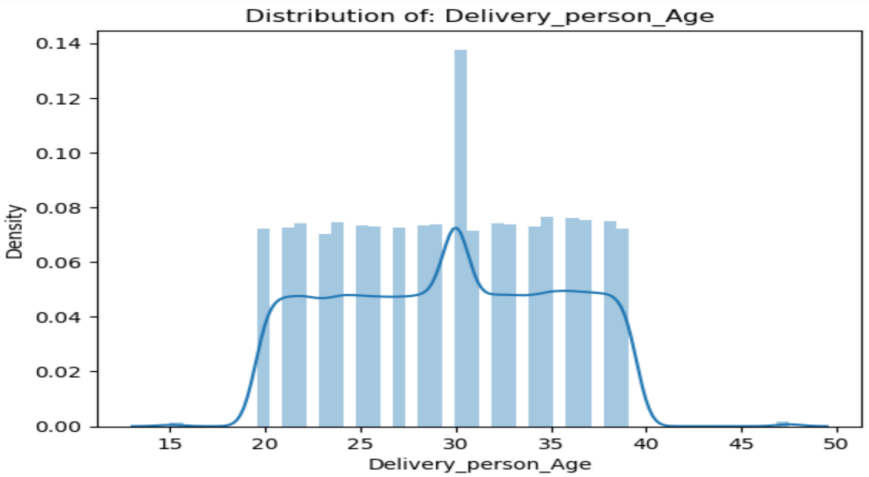
Distplot: A distplot, short for distribution plot, is a data visualization tool that displays the distribution of a univariate dataset. It combines a histogram with a kernel density

estimation (KDE) plot to provide insights into the data's underlying probability distribution. Distplots are particularly useful for understanding the data's central tendency, spread, and shape.

```
[17]: #Visualising the data in the table:

      #in terms of Distplot:

      for col in df.columns:
          if pd.api.types.is_numeric_dtype(df[col]):
              sns.distplot(df[col])
              plt.title("Distribution of: "+col)
              plt.show()
```



So Here Apply the Square root Transformation on The Columns.

Activity 3.3.3: Encoding:

- The categorical Features are can't be passed directly to the Machine Learning Model. So we convert them into Numerical data based on their order. This Technique is called Encoding.
- Here we are importing Label Encoder from the Sklearn Library.
- Here we are applying fit_transform to transform the categorical features to numerical features.

```
import pickle
from sklearn.preprocessing import LabelEncoder

# Create a dictionary to store label encoders for each categorical column
label_encoders = {}

# Identify and initialize label encoders for categorical columns
for col in df.columns:
    if df[col].dtype == 'O':  # Check if the column is of object (string) type
        label_encoders[col] = LabelEncoder()  # Initialize a label encoder for the column

label_encoders
```

```
{'Order_Date': LabelEncoder(),
 'Time_Orderd': LabelEncoder(),
 'Time_Order_picked': LabelEncoder(),
 'Weatherconditions': LabelEncoder(),
 'Road_traffic_density': LabelEncoder(),
 'Type_of_order': LabelEncoder(),
 'Type_of_vehicle': LabelEncoder(),
 'multiple_deliveries': LabelEncoder(),
 'Festival': LabelEncoder(),
 'City': LabelEncoder()}
```

```
[ ] # Apply label encoding to each categorical column
    for col in df.columns:
      if df[col].dtype=='O':
        df[col] = label_encoders[col].fit_transform(df[col])
        filename=f"{col}.pkl"
        pickle.dump(label_encoders[col],open(filename,"wb"))
```

Activity 3.3.4: 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, df is passed with dropping the target variable. And on y target variable is passed.

```
x=df.drop(['Time_taken'],axis=1)
y=df['Time_taken']
y
```

```
0        24
1        33
2        26
3        21
4        30
         ..
45588    32
45589    36
45590    16
45591    26
45592    36
Name: Time_taken, Length: 45593, dtype: int64
```

Activity 3.3.5: Scaling:

- Scaling is a technique used to transform the values of a dataset to a similar scale to improve the performance of machine learning algorithms. Scaling is important because many machine learning algorithms are sensitive to the scale of the input features.
- Here we are using Standard Scaler.
- This scales the data to have a mean of 0 and a standard deviation of 1. The formula is given by: X_scaled = (X - X_mean) / X_std.

Scaling

```
[ ] from sklearn.preprocessing import StandardScaler
    ss=StandardScaler()
    x=ss.fit_transform(x)
```

```
filename="ss.pkl"
pickle.dump(ss,open(filename,"wb"))
```

Activity 3.3.6: Performing Train Test Split:

- train_test_split is a commonly used function in machine learning for splitting a

dataset into two subsets: a training set and a testing (or validation) set. This split allows you to train a machine learning model on one portion of the data and evaluate its performance on another, which helps assess the model's ability to generalize to new, unseen data.

```
[ ]  from sklearn.model_selection import train_test_split
     X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

Activity 4: Model Building:

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 classification algorithms. The best model is saved based on its performance.

Activity 4.1 : 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 three classification algorithms. The best model is saved based on its performance.
Metrics for regression are used to evaluate the performance of machine learning models that are designed for predicting continuous numerical values. These metrics help assess how well a regression model's predictions align with the actual target values. Here are some commonly used regression metrics:

R-Squared (R2) Score:

R-squared measures the proportion of the variance in the target variable that is explained by the model. It ranges from 0 to 1, with higher values indicating a better fit.

Mean Absolute Error(MAE):

MAE measures the average absolute difference between the predicted values and the actual target values. It gives equal weight to all errors, making it less sensitive to outliers.

Mean Squared Error(MSE):

MSE measures the average squared difference between the predicted values and the actual target values. Squaring the errors gives more weight to larger errors, making it sensitive to outliers.

[24]:
```
#Building the Model:

#Regression :  Predicting Values and testing the model performance by using metrics from Regres

from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
def Predictions(model):
    models = model.fit(X_train, y_train)
    y_pred_test =  models.predict(X_test)
    y_pred_train = models.predict(X_train)
    print("R2 Score for Training:",r2_score(y_pred_train,y_train))
    print("\n")
    print("Mean Squared Error for Training:",mean_squared_error(y_pred_train,y_train))
    print("\n")
    print("Mean Absolute Error for Training:",mean_absolute_error(y_pred_train,y_train))
    print("\n")
    print("R2 Score for Testing:",r2_score(y_pred_test,y_test))
    print("\n")
    print("Mean Squared Error for Testing:",mean_squared_error(y_pred_test,y_test))
    print("\n")
    print("Mean Absolute Error for Testing:",mean_absolute_error(y_pred_test,y_test))
    print("\n")
```

Activity 5: Performance Testing & Hyperparameter Tuning:

Hyperparameter tuning is a crucial step in optimizing the performance of machine learning models. Hyperparameters are settings or configurations that are not learned from the data but are set prior to training a model. These parameters can significantly impact a model's performance, and finding the best combination of hyperparameters is often done through a process called hyperparameter tuning.

One popular method for hyperparameter tuning is Randomized Search Cross-Validation (RandomizedSearchCV). RandomizedSearchCV is a more efficient alternative to Grid Search, which exhaustively explores all possible combinations of hyperparameters. Instead, Randomized Search samples a specified number of hyperparameter combinations randomly.

Activity 5.1 : The Best Performing Model:

[25]:
```
#Model - 1:
#Decision Tree Model:
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor()
```

[26]:
```
Predictions(dt)
```

```
R2 Score for Training: 1.0


Mean Squared Error for Training: 0.0


Mean Absolute Error for Training: 0.0


R2 Score for Testing: 0.6549739070004594


Mean Squared Error for Testing: 30.279855247285887


Mean Absolute Error for Testing: 4.16865884417151
```

Milestone 6: Model Deployment:

Activity 6.1: Save the best model:

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance.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.

```
[ ]: #Saving the Best model:(Random Search)

import pickle
filename = "rf.pkl"
pickle.dump(random_search,open(filename,"wb"))
```

Activity 6.2: Integrate with Web Framework:

In this section, we will be building a web application that is integrated to the model we built. A 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 6.2.1: Building HTML Pages:

The index.html code is:

For reference: Index.html code

Activity 6.2.2: Build Python Code:

For reference: app.py code

Activity 6.2.3: 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.

```
C:\Users\mytre\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:440: InconsistentVersionWarning: Trying to unpickle estimator StandardScaler from ver
hen using version 1.7.0. 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(
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

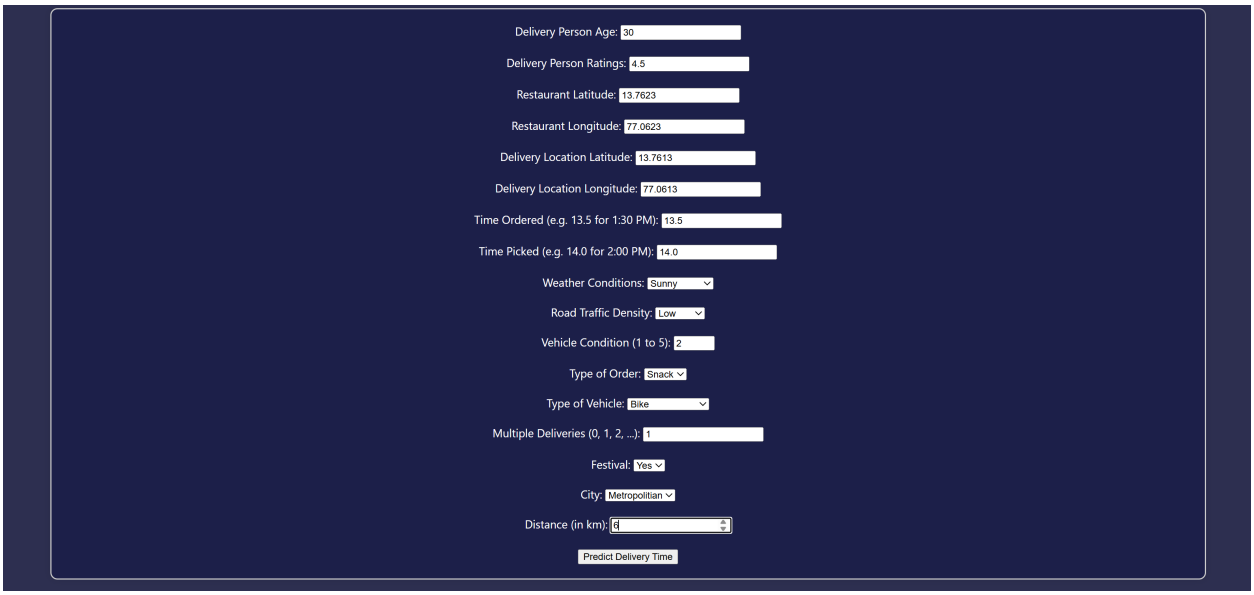
Now, Go the web browser and write the localhost url (http:// 127.0.0.1:5000) to get the below result,

info about 17 Arguments:

"Delivery Time Predictor" can give the accurate time basis on 17 argument Values!

- **Delivery Person Age**: It is one of the factor we can consider while doing delivery!
- **Delivery Person Ratings**: It can resembles his Work in the time of Delivery!
- **Restaurant Latitude and Longitude**: These latitude and longitude of Restaurant can gives the exact location on the map can be useful to predict the accurate time!
- **Location Latitude and Longitude**: These latitude and longitude of Delivery Location can gives the exact location on the map can be useful to predict the accurate time!
- **Ordered and Time Picked**: These two things are also plays a key role to predict the time!
- **Weather Conditions:** Which contains multiple options: "Sunny","Stormy","Windy","Cloudy","Fog" and"Sand Storm"
- **Road Traffic Density:** Which contains multiple options: "Low","Medium","High" and "Jam"
- **Vehicle Conditon:** Here we can represent the condition basis from 0(Very Bad) to 5(Very Good) stage!
- **Type of Order:** Which Contains the options: "Meal","Buffet","Drinks" and "Snack"
- **Type of Vehicle:** Which Contains the three categories: "Scooter","Bike" and "Electric Scooter"
- **Multiple Deliveries:** Which Contains the options: "0"(for single person),"1"(for a couple) and "2"(for a group)
- **Festival:** While Festival it causes the traffic so it contains the options "Yes" and "No"
- **City:** Which Contains "Metropolitian"(eg: Hyderabad), "Urban"(eg : Vijayawada) and "Semi Urban"(eg: Anantapuram)
- **Distance(km):** You need to give the distance(in KM) between Restaurant and Delivery Location!

Filling the Arguments:



The Predicted result is:

Milestone 7: Project Demonstration & Documentation:

Below mentioned deliverables to be submitted along with other deliverables.\

Activity 7.1: Video Demonstration:

Video link

Activity 7.2: Project Documentation:

This is the project document!SI-28190-1751352377