Predictive Analysis of Food Demand Forecasting

<u>Project Description</u>

Food demand forecasting is crucial for optimizing inventory, reducing waste, and ensuring a steady supply of food products. This project leverages machine learning, specifically the RandomForestClassifier, to predict future food demand based on historical sales data and external factors such as weather, promotions, and holidays.

Scenario : Lead Converted

A **food delivery company** or **grocery store chain** needs to predict future food demand to avoid wastage, reduce shortages, and optimize inventory management. The company operates in multiple locations and sells various perishable food items.

<u>Project Flow:</u>

The company decides to implement a **machine learning model** using **RandomForestClassifier** to:

- Analyze historical sales data.
- Identify demand trends based on external factors (weather, promotions, holidays).
- Accurately predict future food demand for each store or delivery hub

<u>To accomplish this, we have to complete all the activities listed below:</u>

- **Data Collection & Preparation**
    - Collect the dataset
    - Data Preparation
- **Exploratory Data Analysis**
    - Descriptive statistical
    - Visual Analysis
    - Balancing
- **Model Building**
    - Training the model in multiple algorithms
    - Testing the model
- **Performance Testing**
    - Testing model with multiple evaluation metrics
    - Comparing model accuracy
- **Model Deployment**
    - Save the best model
    - Integrate with Web Framework

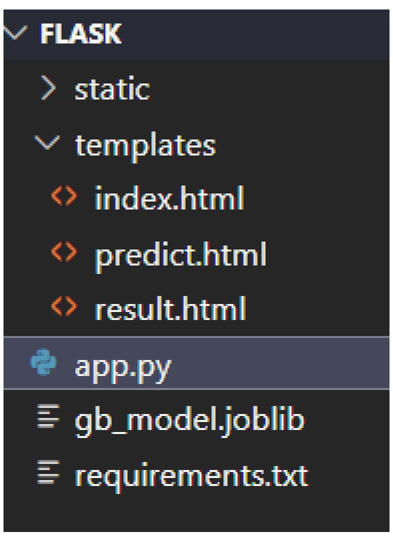
Prior Knowledge:

Before working on this project, you should have a foundational understanding of:

Domain Knowledge:

- **Food Supply Chain & Inventory Management**
- **Factors Affecting Food Demand** (holidays, seasons, promotions, weather)
- **Machine Learning Concepts:**
- **Supervised Learning**: Supervised Learning Guide
- **Classification Algorithms**:
  - **Logistic Regression**: Logistic Regression Guide
  - **K-Nearest Neighbors (KNN)**: KNN Guide
  - **Decision Tree Classifier**: Decision Tree Guide
  - **Random Forest Classifier**: Random Forest Guide
  - **AdaBoost Classifier**: AdaBoost Guide
  - **XGBoost Classifier**: XGBoost Guide
  - **Gradient Boosting Classifier**: Gradient Boosting Guide

Project Structure



- We are building a Flask application that needs HTML pages stored in the Template folder and python script app.py for scripting

- edtech_rf_model.joblib is our saved model. Further, we will use this model for flask integration.

- Training folder contains a model training file.

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.

Milestone 1: 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.

Dataset: LINK

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 several 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

```
# Data Handling & Processing
import numpy as np
import pandas as pd
# Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns
# Preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
# Machine Learning Models
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
# Model Evaluation
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Deployment (if needed)
from flask import Flask, request, jsonify
```

Activity 1.2: Read the Data set

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.

```python
dataframe.head(5)
```

[4] ✓ 0.0s Python

	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders	category	cuisine	city_code
0	1379560	1	55	1885	136.83	152.29	0	0	177	Beverages	Thai	647
1	1466964	1	55	1993	136.83	135.83	0	0	270	Beverages	Thai	647
2	1346989	1	55	2539	134.86	135.86	0	0	189	Beverages	Thai	647
3	1338232	1	55	2139	339.50	437.53	0	0	54	Beverages	Indian	647
4	1448490	1	55	2631	243.50	242.50	0	0	40	Beverages	Indian	647

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 and noise. So, we need to clean the dataset properly in order to fetch good results.

This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling Outliers

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

Activity 2.1: Handling Missing Values

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

Activity 2.1: Handling Missing Values

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

```
dataframe.shape
```

```
(456548, 12)
```

Above Figure Describes the Shape of the Dataset i.e., there are 456548 rows and 12 columns including the Target column as well.

```
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 456548 entries, 0 to 456547
Data columns (total 12 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   week            456548 non-null  int64  
 1   center_id       456548 non-null  int64  
 2   meal_id         456548 non-null  int64  
 3   checkout_price  456548 non-null  float64
 4   base_price      456548 non-null  float64
 5   num_orders      456548 non-null  int64  
 6   category        456548 non-null  object 
 7   cuisine         456548 non-null  object 
 8   city_code       456548 non-null  int64  
 9   region_code     456548 non-null  int64  
 10  center_type     456548 non-null  object 
 11  op_area         456548 non-null  float64
dtypes: float64(3), int64(6), object(3)
memory usage: 41.8+ MB
```

df.info() provides the information about the column's data type and provides the count of non-null values in the column.

1. **Identifying Categorical and Numerical Columns**

There are categorical columns and numerical columns present in the data set so, we store them in cat and num variables

1. **Filling Missing Values for Categorical Columns**

```
for column in cat:
  df[column] = df[column].fillna(df[column].mode()[0])

for column in num:
  df[column] = df[column].fillna(df[column].median())
```

The for loop Iterates over all categorical columns and fills missing values (NaN) with the most frequent value (mode) of that column. mode()[0] extracts the first value in case there are multiple modes. the second for loop iterates over all numerical columns and fills missing values with the median of that column. The median is used because it is less sensitive to outliers compared to the mean

After filling null values, again we will check if there is any column containing null value

```
dataframe.isnull().any()
```

```
week              False
center_id         False
meal_id           False
checkout_price    False
base_price        False
num_orders        False
category          False
cuisine           False
city_code         False
region_code       False
center_type       False
op_area           False
dtype: bool
```

```
dataframe.isnull().sum()
```

```
week             0
center_id        0
meal_id          0
checkout_price   0
base_price       0
num_orders       0
category         0
cuisine          0
city_code        0
region_code      0
center_type      0
op_area          0
dtype: int64
```

Activity 2.2: Handling Categorical Values:

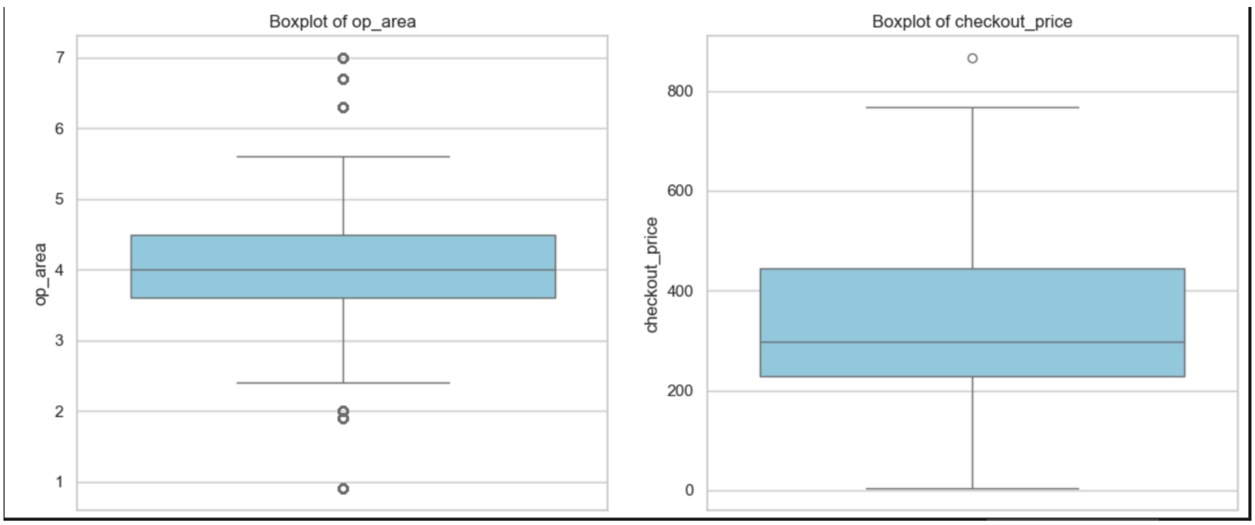
There are categorical column present in the dataset
As we know there are no missing values/ null values present in the dataset. We need to know the number of categories present in the column with their counts.
There are several operations to find different insights using categorical values some of the functions are **value_counts**, **cross_tab(), mode**, **and replacement of values.**

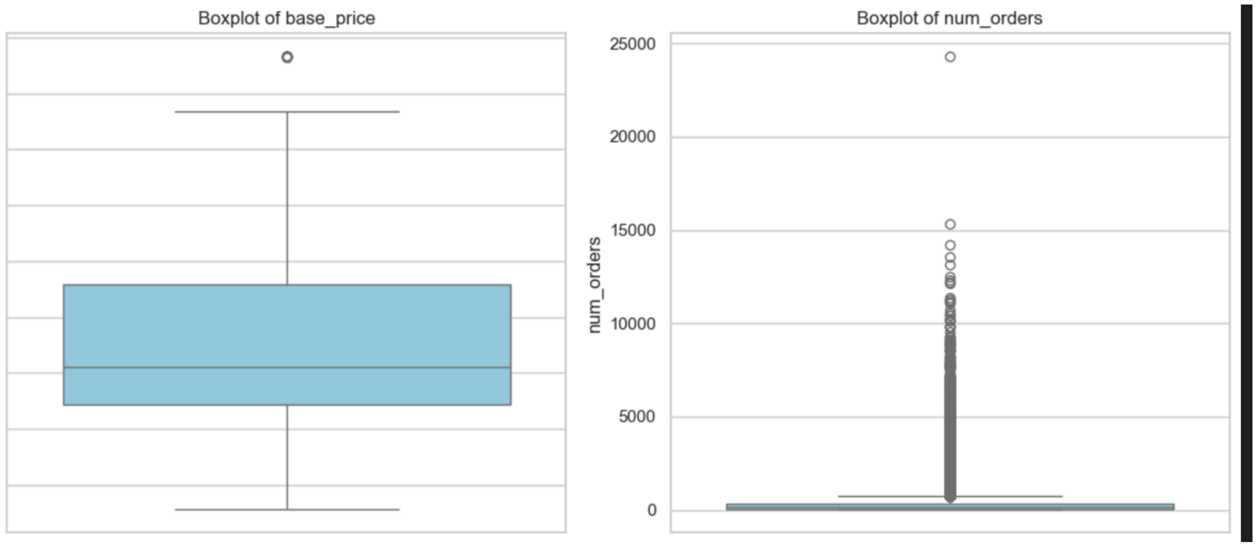
We have multiple categorical columns in the dataset so we will perform label encoding to the columns

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

def encoding(df, columns):
  for column in columns:
    df[column] = le.fit_transform(df[column])
  return df

df = encoding(df, cat)
df
```

Activity 2.3: Treating Outliers:

Outliers are the abnormal data that are away from the range of the distribution of the data of each column in the data. Here we have the box plot to find whether the Outliers are present or not in the data.





Milestone 2: Exploratory Data Analysis

Activity 1: Descriptive Analysis

Descriptive analysis involves examining fundamental characteristics of data using statistical methods. It provides insights into the mean, standard deviation, minimum, maximum, and percentile values of continuous features.

```
dataframe.describe()
```

	week	center_id	meal_id	checkout_price	base_price	num_orders	city_code	region_code	op_area
count	456548.000000	456548.000000	456548.000000	456548.000000	456548.000000	456548.000000	456548.000000	456548.000000	456548.000000
mean	74.768771	82.105796	2024.337458	332.238933	354.156627	261.872760	601.553399	56.614566	4.083590
std	41.524956	45.975046	547.420920	152.939723	160.715914	395.922798	66.195914	17.641306	1.091686
min	1.000000	10.000000	1062.000000	2.970000	55.350000	13.000000	456.000000	23.000000	0.900000
25%	39.000000	43.000000	1558.000000	228.950000	243.500000	54.000000	553.000000	34.000000	3.600000
50%	76.000000	76.000000	1993.000000	296.820000	310.460000	136.000000	596.000000	56.000000	4.000000
75%	111.000000	110.000000	2539.000000	445.230000	458.870000	324.000000	651.000000	77.000000	4.500000
max	145.000000	186.000000	2956.000000	866.270000	866.270000	24299.000000	713.000000	93.000000	7.000000

Activity 2: Visual analysis

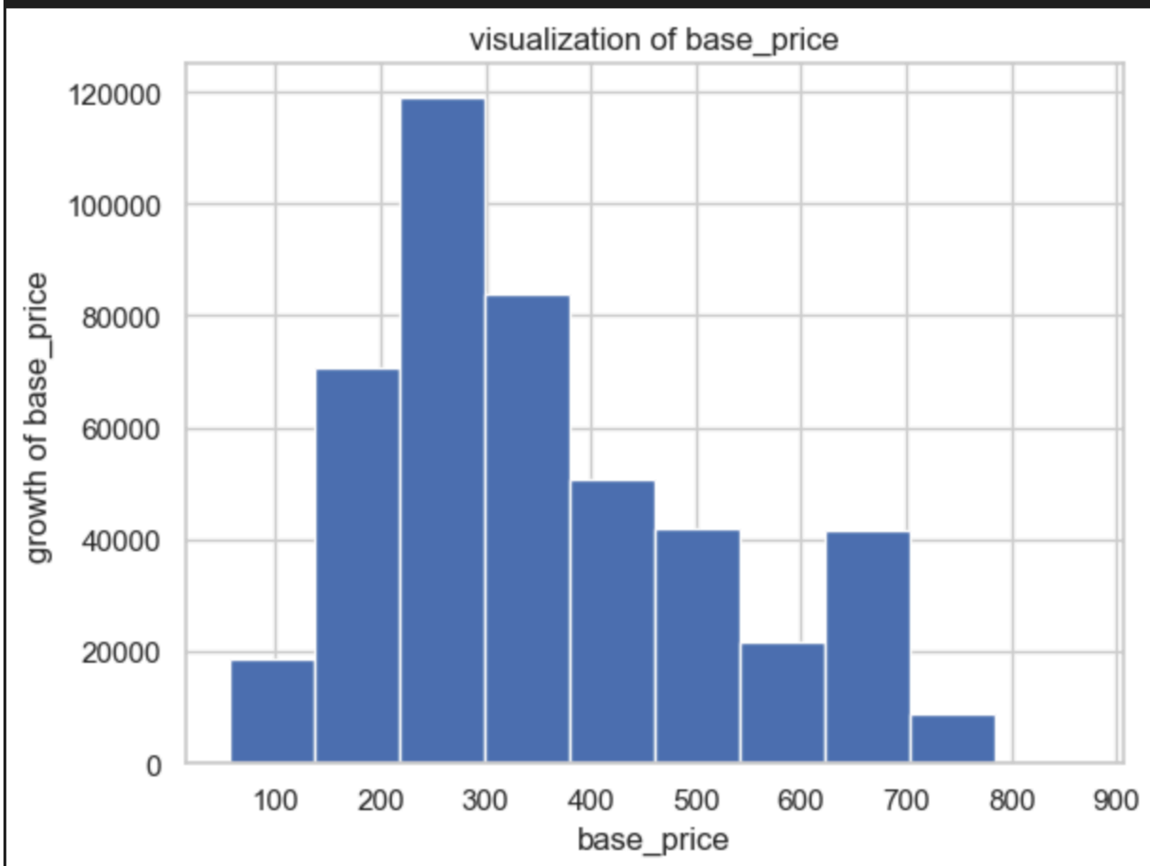
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: univariate analysis

In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed a histogram

```
plt.hist(dataframe['base_price'])
plt.title ('visualization of base_price')
plt.xlabel('base_price')
plt.ylabel('growth of base_price')
```

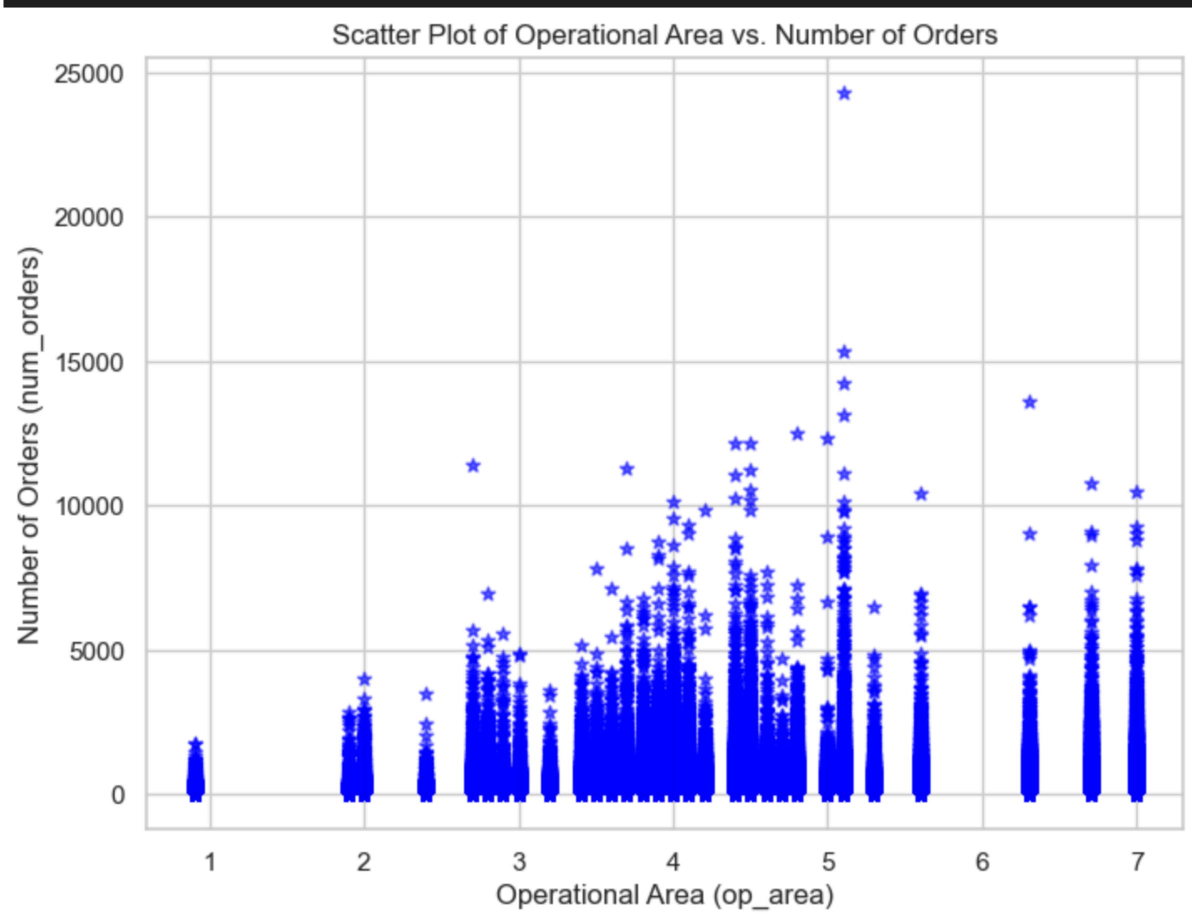
```
ext(0, 0.5, 'growth of base_price')
```



Activity 2.2: Bivariate analysis

Bivariate analysis is a statistical method that involves the analysis of two variables to determine the empirical relationship between them. Here we have Scatter plot for Bivariate Analysis.

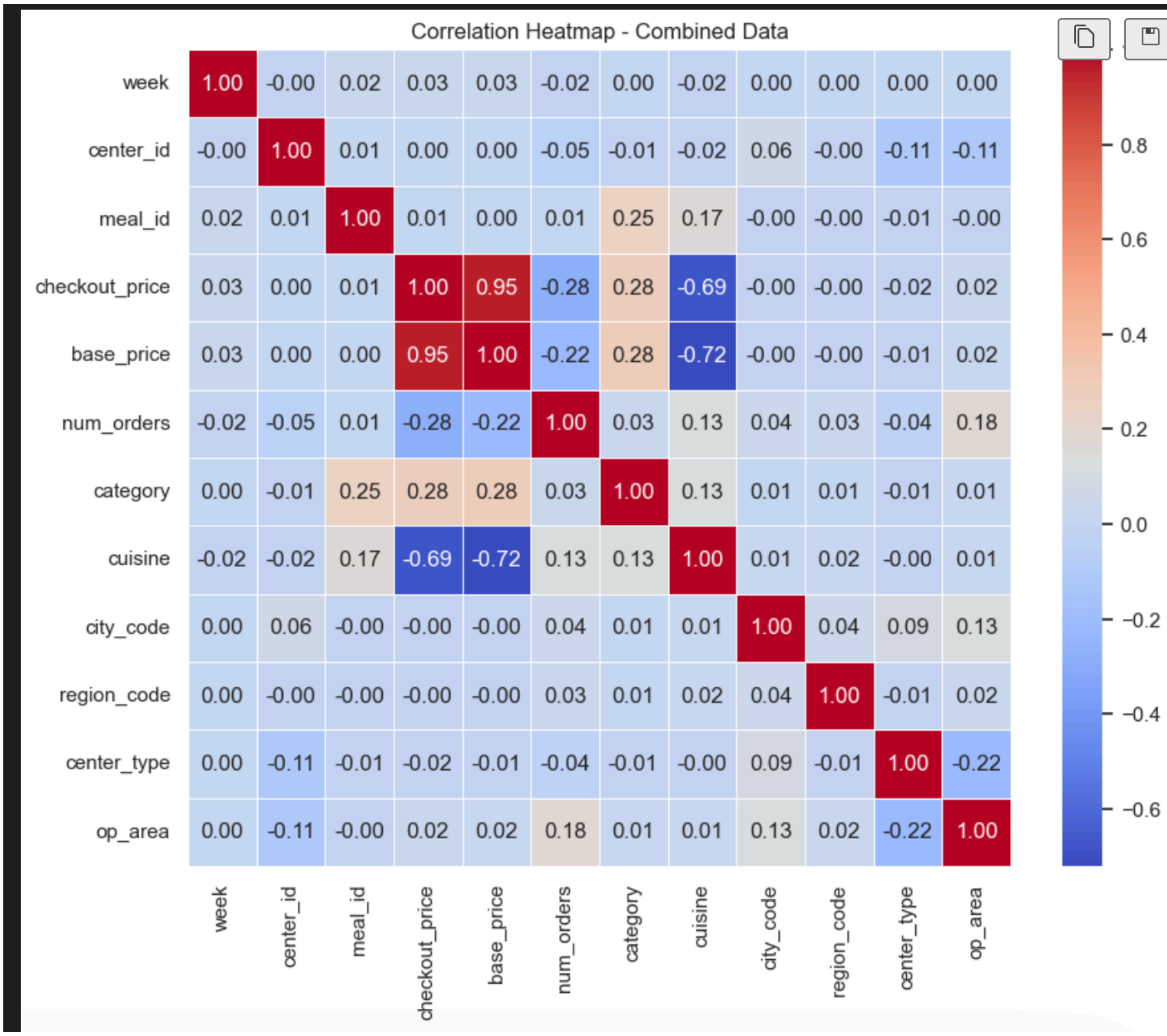
```
# it shops rrelationship between num_orders and op_area
plt.figure(figsize=(8, 6))
plt.scatter(dataframe["op_area"], dataframe["num_orders"], marker='*', color='blue', alpha=0.6)
plt.xlabel("Operational Area (op_area)")
plt.ylabel("Number of Orders (num_orders)")
plt.title("Scatter Plot of Operational Area vs. Number of Orders")
plt.grid(True)
plt.show()
```



Activity 2.3: Multi-variate analysis

Multi-variate analysis is a statistical method that involves the analysis of more than 2 variables to determine the empirical relationship among them. Here we have a heatmap representing the correlation among the variables in the Data.

```python
corr_matrix = dataframe.corr()

# Create heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)

# Title and display
plt.title("Correlation Heatmap - Combined Data")
plt.show()
```



- Total Time Spent on Website and Page Views Per Visit has a strong positive correlation with Converted, indicating that more time spent on the website is associated with higher conversion rates

- Total Visits has a weaker correlation with Converted, implying it is less influential on conversion rates compared to the other variables

Activity 5: 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. "x" represents the whole data columns other than the target column,"y" represents the Target column in the dataset. We need to build the model by giving the training to the model and making the predictions on the test data.so we need to divide the whole dataset into training and testing data.

For splitting training and testing data we are using train_test_split () function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
= df[['Total Time Spent on Website','Tags','Lead Quality', 'Last Notable Activity', 'Lead Origin']]
= df['Converted']

_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Milestone 3: Model Building

Activity 1: Training and testing the models using 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 classification algorithms. The best model is saved based on its performance.

Activity 2.4 : Random-Forest Classifier

Random Forest algorithm is the classification and regressions algorithm initialized and training data is passed to the model and assigned to the variable as model4 with .fit() function. Test data is predicted with model4.predict() function and saved in a new variable. For evaluating the model accuracy is calculated. For the best obtaining of accuracy we use hyper parameter tuning to tune the model with the best hyper parameters using the Grid Search CV by choosing the best params we can able to get the best accuracy this method is known as Hyper parameter Tuning.

```
#train classification model
model= RandomForestClassifier(max_depth=10,random_state=45)
model.fit(X_train, Y_train)

#predictions\
y_pred=model.predict(X_test)


from sklearn.metrics import accuracy_score, classification_report

#Evaluate model
accuracy=accuracy_score(Y_test, y_pred)
print(f"Accuracy:{accuracy:.4f}")

#classification Report
print("\nclassification Report:")
print(classification_report(Y_test, y_pred))
```

✓ 16.5s

```
Accuracy:0.9011
```

We make the predictions on the train and test data using predict function and assigning the predicted values to the y_pred_train and y_pred_test

We are calculating the accuracy scores on how the model is working on the data and we use cross-validations to reduce the Bias and trade condition to the dataset. Actually we are able to divide the dataset based on the chosen test size. If CV=4. then we trained the model with 75% of the data and we tested it with 25% of the data. If CV=5 then we are training the model with 80% of the data and testing the model with 20% of the data.

From the RandomForestClassifier model, we got accuracy as 0.9011

Milestone 4: Performance Testing

Under performance Testing we need to test the model's accuracy with different testing Metrics like Precision, Recall and F1_score. Below are the performance Metrics of final fixed model

```
precision=precision_score(y_train,y_pred_train6)
recall=recall_score(y_train,y_pred_train6)
f1=f1_score(y_train,y_pred_train6)
print("precision",precision)
print("recall",recall)
print("f1_score",f1)
```

```
precision 0.9233308877476155
recall 0.8925531914893617
f1_score 0.9076812116840967
```

By doing Performance Testing, we got precision value as 0.92, recall value as 0.89 and f1_score as 0.90.

Milestone 5: Model Deployment

Activity 1: Save and load 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.

```
import joblib
joblib.dump(model, 'model.joblib')
```

-

Activity 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 3.1: Building Html Pages:

For this project create two HTML files namely

- index.html
- predict.html
- result.html

and save them in the templates folder.

Activity 3.2: Build Python code:

Import the libraries

```
import os
import joblib
import pandas as pd
from flask import Flask, request, render_template
```

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

```
app = Flask(__name__)

# Load the model at the start of the application
model_path = 'gb_model.joblib'

rf_model = joblib.load(model_path) if os.path.exists(model_path) else None


if rf_model is not None:
    print(f"Model loaded successfully from {model_path}.")
else:
    print(f"Model not found at {model_path}.")
```

We render index.html for displaying the web application , similarly we render the predict.html for the user input values of the forms to predict the income. Simultaneously we render the result .html to display the result of the prediction value.

Render Index.html:

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

Here we will be using a declared constructor to route to the HTML page which we have created earlier.
In the above example, '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered.

Render Predict.html:

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

In the predict.html where we provide the user inputs in the form for the prediction of income

Whenever you enter the values from the html page the values can be retrieved using POST and GET Methods.

Retrieving the value from UI:

```python
@app.route('/result', methods=['POST'])
def result():
    if request.method == 'POST':
        try:

            tt_on_website = request.form['ttsw']
            tags = request.form['tags']  # Expecting 'yes' or 'no'
            lead_quality = float(request.form['lead_quality'])  # Expecting float input
            ln_actvty = request.form['ln_actvty']  # Expecting a string like 'low', 'medium', 'high'
            lead_orgin = request.form['lead_orgin']

            input_features = [
                tt_on_website,
                tags,  # Now this is an int
                lead_quality,
                ln_actvty,  # Now this is an int
                lead_orgin
            ]
            print(input_features)

            # Ensure thety correct number of values
            if len(input_features) == 5:
                # Prepare DataFrame for the model
                names = ['Total Time Spent on Website', 'Tags', 'Lead Quality', 'Last Notable Activity',  'Lead Origin']
                data = pd.DataFrame([input_features], columns=names)
                print(data)

                prediction = rf_model.predict(data)
                print(prediction[0])

                if prediction[0]== 0:
                    return render_template('result.html', prediction="Lead not converted")
                else:
                    return render_template('result.html', prediction="Lead converted")
```

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

Main Function:

```
if __name__ == "__main__":
    app.run(debug=True, port=4000)
```

Activity 3.3: Run the web application

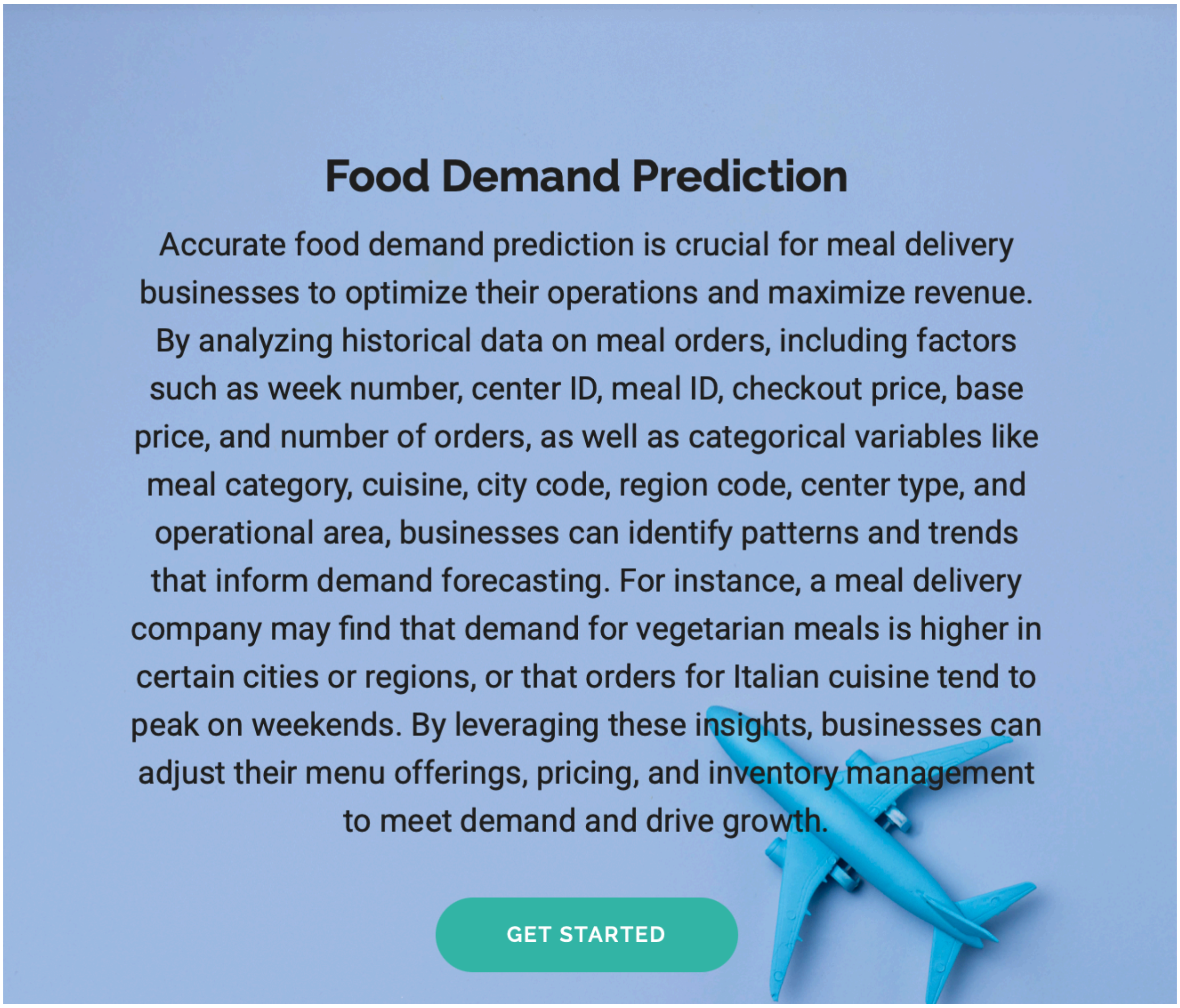
- Open vs code application in the search menu.
- Navigate to the folder where your flask folder of your files exist.
- Click on the view button in the vs code nav bar and click on the terminal option in the dropdown menu.
- Now type “app.py” command
- You will have a link displayed in the terminal as * Running on “http://127.0.0.1:5000” Double click on the link then you will be navigated to the web application.
- Click on the predict button in the nav bar, enter the inputs, click on the predict button, and see the result/prediction in the result.html.

```
Model loaded successfully from /Users/gwilliam/CSV /model.joblib.
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
er instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (fsevents)
Loading model...
Model loaded successfully from /Users/gwilliam/CSV /model.joblib.
 * Debugger is active!
 * Debugger PIN: 108-948-720
```

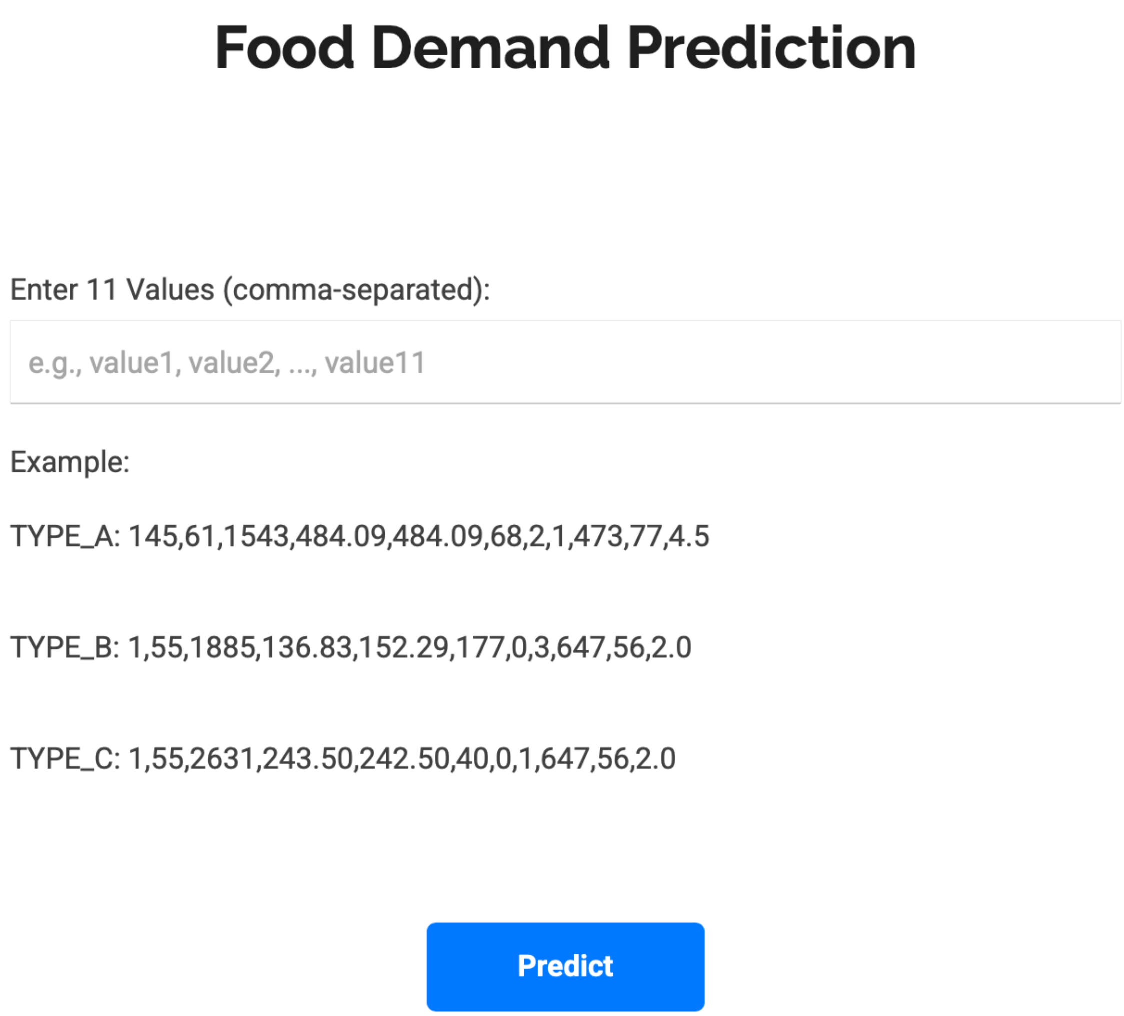
Now, Go the web browser and write the localhost URL (http://127.0.0.1:4000) to get the below results

Results:

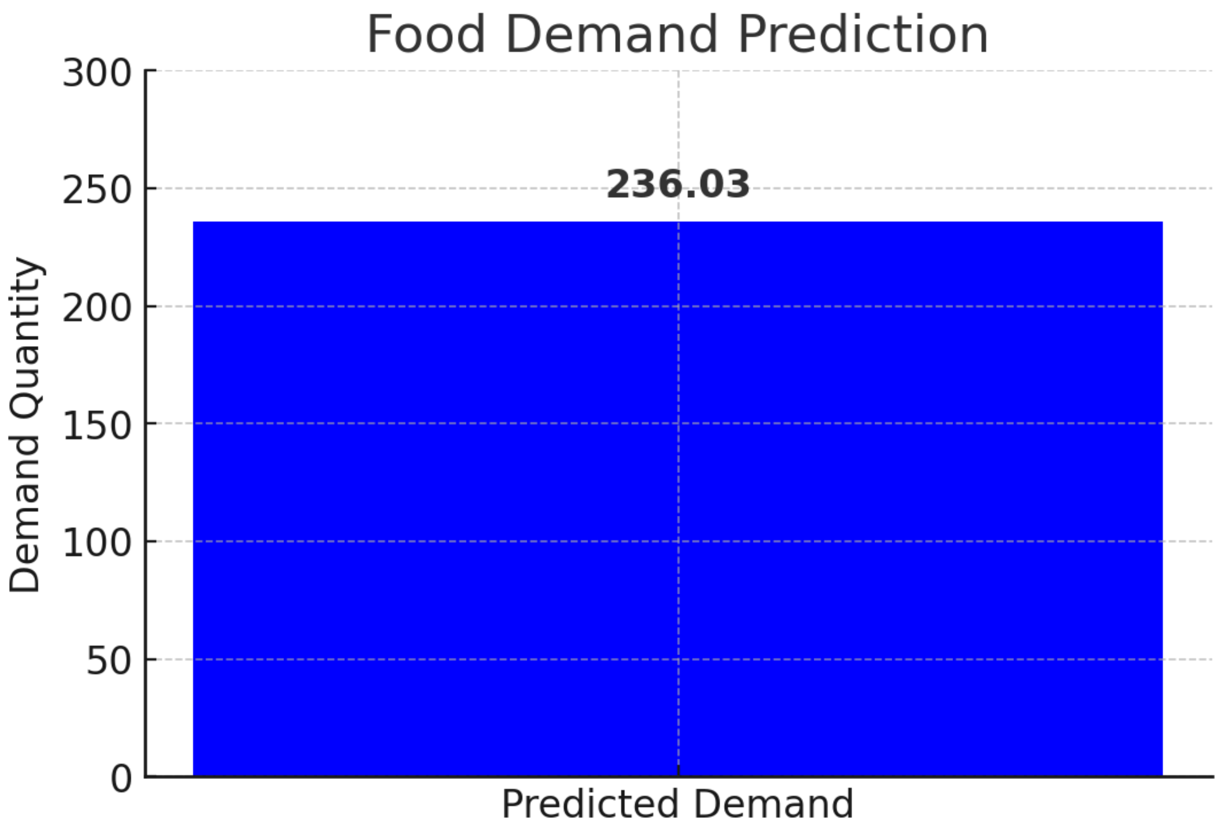
a. Index page (Index.html)



b) Prediction page (Predict1.html) and result page(Result.html)



By providing the inputs for the columns in the prediction page we will get the desired output in the result page.



Here's a bar chart visualizing the predicted food demand (**236.03 units**) based on the given input.