

FOOD DEMAND FORECASTING FOR FOOD DELIVERY COMPANY

Project Description:

Food Demand Forecasting for a Food Delivery Company using IBM Cloud involves leveraging advanced data analytics and machine learning algorithms to predict the future demand for various food items. By analyzing historical data such as order volumes, time of day, day of the week, weather conditions, and special events, the system can forecast the demand accurately. This helps the food delivery company optimize its operations by efficiently allocating resources, reducing wastage, and ensuring timely delivery to customers.

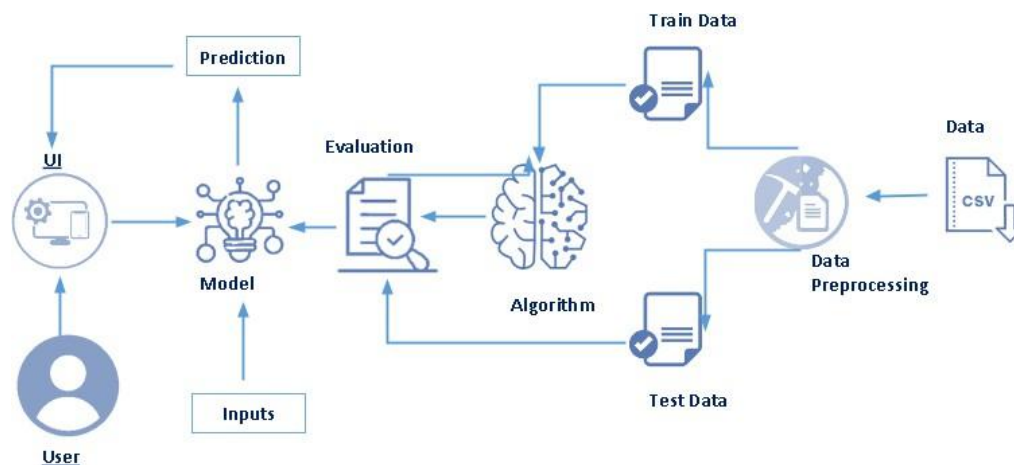
Project Scenario:

Scenario 1: During a heat wave, the system predicts a surge in demand for cold beverages and salads, prompting the company to stock up on these items and adjust delivery routes to meet the increased demand.

Scenario 2: On a rainy weekend, the forecast indicates a higher demand for comfort foods like pizza and soup. The company prepares by increasing kitchen staff and ingredient orders to fulfill orders promptly.

Scenario 3: Ahead of a major sporting event, the system predicts a spike in orders for party platters and snacks. The company adjusts its marketing strategy to promote these items and ensures sufficient inventory to meet the anticipated demand, enhancing customer satisfaction and maximizing revenue.

Technical Architecture :



Pre-Requisites

To complete this project, you must require following software's, concepts and packages.

Pre-Requisites:

To complete this project, you must require following software's, concepts and packages.

In Order To Develop This Project, We Need To Install Following Software's/Packages

To develop this project, we need to install the following software's/packages:

Anaconda Navigator:

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning-related applications. It can be installed on Windows, Linux, and macOS. Conda is an open-source, cross-platform, package management system. Anaconda comes with so many very nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter Notebook and Spyder

To Build Machine Learning Models You Must Require The Following Packages

Numpy:

It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations

Pandas:

It is an open-source numerical Python library. It is mainly used for data manipulation.

Scikit-learn:

It is a free machine-learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbors, and it also supports Python numerical and scientific libraries like NumPy and SciPy

Matplotlib and Seaborn:

Matplotlib is mainly deployed for basic plotting. Visualization using Matplotlib generally consists of bars, pies, lines, scatter plots, and so on. Seaborn: Seaborn, on the other hand, provides a variety of visualization patterns. It uses less syntax and has easily interesting default themes.

Flask:

Web framework used for building Web applications

If you are using Anaconda Navigator, follow the below steps to download the required packages:

Open anaconda prompt.

Type “pip install jupyter notebook” and click enter.

Type “pip install spyder” and click enter.

Type “pip install numpy” and click enter.

Type “pip install pandas” and click enter.

Type “pip install matplotlib” and click enter.

Type “pip install seaborn” and click enter.

Type “pip install sklearn” and click enter.

Type “pip install Flask” and click enter.

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

Project Objectives

By the end of this project:

- You'll be able to understand the problem to classify if it is a regression or a classification kind of problem.
- You will be able to know how to pre-process/clean the data using different data pre-processing techniques.
- You will be able to analyze or get insights into data through visualization.
- Applying different algorithms according to the dataset and based on visualization.
- You will be able to know how to find the accuracy of the model.
- You will be able to know how to build a web application using the Flask framework.

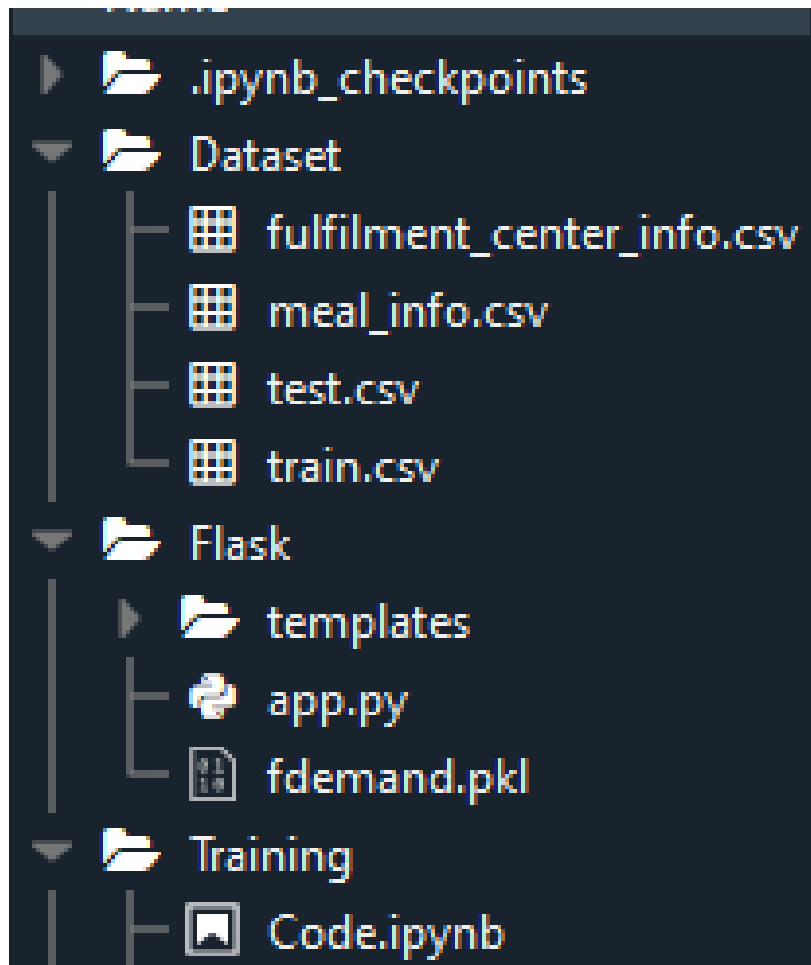
Project Flow

Project Work Flow:

- The user interacts with the UI (User Interface) to upload the input features.
- Uploaded features/input is analyzed by the model which is integrated.
- Once the model analyses the uploaded inputs, the prediction is showcased on the UI.
- To accomplish this, we have to complete all the activities and tasks listed below

Project Structure

Create a Project folder which contains files as shown below



- We have three folders dataset, Flask, and training.
- A Python file called app.py for server-side scripting.
- We need the model that is saved and the saved model in this content is in fdemand.pkl
- Templates folder which contains home.html and upload.html files.
- Static folder which contains css folder which contains styles.css.

The above project can be downloaded from

<https://github.com/Guided-Projects/Food-Demand-Forecasting>

Dataset Collection

ML depends heavily on data, without data, it is impossible for an “AI” to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training data set. It is the actual data set used to train the model for performing various actions.

Collect The Dataset Or Create The Dataset:

Our base data consists of four csv files containing information about test data, train data and other required information.

- train.csv: Contains information like id, week, center id, meal id, checkout price, base price, emailer for promotion, homepage featured, number of orders. This file is used for training.

Variable	Definition
id	Unique ID
week	Week No
center_id	Unique ID for fulfillment center
meal_id	Unique ID for Meal
checkout_price	Final price including discount, taxes & delivery charges
base_price	Base price of the meal
emailer_for_promotion	Emailer sent for promotion of meal
homepage_featured	Meal featured at homepage
num_orders	(Target) Orders Count

- test.csv: Contains information like id, week, center id, meal id, checkout price, base price, emailer for promotion, homepage featured. This file is used for testing.

Variable	Definition
id	Unique ID
week	Week No
center_id	Unique ID for fulfillment center
meal_id	Unique ID for Meal
checkout_price	Final price including discount, taxes & delivery charges
base_price	Base price of the meal
emailer_for_promotion	Emailer sent for promotion of meal
homepage_featured	Meal featured at homepage

- fulfillment_center_info.csv: Contains information of each fulfillment center.

Variable	Definition
center_id	Unique ID for fulfillment center
city_code	Unique code for city
region_code	Unique code for region
center_type	Anonymized center type
op_area	Area of operation (in km ²)

- meal_info.csv: Contains information of each meal being served.

Variable	Definition
meal_id	Unique ID for the meal
category	Type of meal (beverages/snacks/soups,...)
cuisine	Meal cuisine (Indian/Italian/...)

- You can collect dataset from different open sources like kaggle.com, data.gov, UCI machine learning repository etc.
- Here's the link to download dataset.

Data Pre-Processing

Data Pre-processing includes the following main tasks

- Import the Libraries. Reading the dataset.
- Exploratory Data Analysis
- Checking for Null Values.
- Reading and merging .csv files
- Dropping the columns
- Label Encoding
- Data Visualization.
- Splitting the Dataset into Dependent and Independent variable.
- Splitting Data into Train and Test.

Importing The Libraries

The first step is usually importing the libraries that will be needed in the program.

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

Pandas: It is a Python library mainly used for data manipulation.

NumPy: This Python library is used for numerical analysis.

Matplotlib and Seaborn: Both are the data visualization libraries used for plotting graphs which will help us understand the data.

Pickle: to serialize your machine learning algorithms and save the serialized format to a file.

Reading The Dataset

- You might have your data in .csv files, .excel files .tsv files, or something else. But the goal is the same in all cases. If you want to analyse that data using pandas, the first step will be to read it into a data structure that's compatible with pandas.
- Let's load a .csv data file into pandas. There is a function for it, called read_csv(). We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program).
- names on Windows tend to have backslashes in them. But we want them to mean actual backslashes, not special characters.

```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

Exploratory Data Analysis

Exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods, and used to determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions.

`head()` : To check the first five rows of the dataset, we have a function called `head()`.

```
test.head()
```

	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured
0	1028232	146	55	1885	158.11	159.11	0	0
1	1127204	146	55	1993	160.11	159.11	0	0
2	1212707	146	55	2539	157.14	159.14	0	0
3	1082698	146	55	2631	162.02	162.02	0	0
4	1400926	146	55	1248	163.93	163.93	0	0

This `head()` function returns the first 5 rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

```
train.head()
```

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

- Understanding Data Type and Summary of Features
- How the information is stored in a DataFrame or Python object affects what we can do with it and the outputs of calculations as well. There are two main types of data those are numeric and text data types.
- Numeric data types include integers and floats.
- A text data type is known as Strings in Python or Objects in Pandas. Strings can contain numbers and/or characters.
- For example, a string might be a word, a sentence, or several sentences.
- Will see how our dataset is, by using the `info()` method.


```
train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 456548 entries, 0 to 456547
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype
---  -
0   id                    456548 non-null  int64
1   week                 456548 non-null  int64
2   center_id            456548 non-null  int64
3   meal_id              456548 non-null  int64
4   checkout_price       456548 non-null  float64
5   base_price           456548 non-null  float64
6   emailer_for_promotion 456548 non-null  int64
7   homepage_featured    456548 non-null  int64
8   num_orders           456548 non-null  int64
dtypes: float64(2), int64(7)
memory usage: 31.3 MB
```

describe(): functions are used to compute values like count, mean, standard deviation and IQR(Inter Quantile Ranges) and give a summary of numeric type data.

```
train['num_orders'].describe()

count    456548.000000
mean      261.872760
std       395.922798
min       13.000000
25%       54.000000
50%      136.000000
75%      324.000000
max      24299.000000
Name: num_orders, dtype: float64
```

Checking For Null Values

1. After loading it is important to check the complete information of data as it can indicate many of the hidden information such as null values in a column or a row
2. Check whether any null values are there or not. if it is present then the following can be done,

- a. Imputing data using Imputation method in sklearn
- b. Filling NaN values with mean, median, and mode using fillna() method.

We will be using isnull().sum() method to see which total number of missing values.

```
train.isnull().sum()

id                0
week              0
center_id         0
meal_id           0
checkout_price    0
base_price        0
emailer_for_promotion 0
homepage_featured 0
num_orders        0
dtype: int64
```

Reading And Merging .Csv Files

```
meal_info = pd.read_csv("meal_info.csv")
center_info = pd.read_csv("fulfilment_center_info.csv")
```

Merging train.csv and meal_info.csv datasets by using common key id

We notice that the meal_id column in train.csv is similar to the meal_id in the meal_info.csv dataset. Let us merge these two datasets, train.csv and meal_info.csv using the common key meal_id and name the table as trainfinal.

Merging trainfinal.csv and center_info.csv dataset by using common key id

We notice that the center_id column in trainfinal.csv is similar to the center_id in the center_info.csv dataset. Let us merge these two datasets, trainfinal.csv and center_info.csv using the common key center_id and store it back in trainfinal.

Display the first five rows of the trainfinal using head().

```
infinal = pd.merge(train, meal_info, on="meal_id", how="outer")
infinal = pd.merge(trainfinal, center_info, on="center_id", how="outer")
infinal.head()
```

	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_order
1379560	1	55	1885	136.83	152.29	0	0	177	
1018704	2	55	1885	135.83	152.29	0	0	323	
1196273	3	55	1885	132.92	133.92	0	0	96	
1116527	4	55	1885	135.86	134.86	0	0	163	
1343872	5	55	1885	146.50	147.50	0	0	215	

homepage_featured	num_orders	category	cuisine	city_code	region_code	center_type	op_area
0	177	Beverages	Thai	647	56	TYPE_C	2.0
0	323	Beverages	Thai	647	56	TYPE_C	2.0
0	96	Beverages	Thai	647	56	TYPE_C	2.0
0	163	Beverages	Thai	647	56	TYPE_C	2.0
0	215	Beverages	Thai	647	56	TYPE_C	2.0

Dropping Columns

Let's drop columns "center_id" and "meal_id" as they are not required for the further process. Display the changes of trainfinal table using head().

```
trainfinal = trainfinal.drop(['center_id', 'meal_id'], axis=1)
trainfinal.head()
```

	id	week	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders	category	cuisine	city_code
0	1379560	1	136.83	152.29	0	0	177	Beverages	Thai	647
1	1018704	2	135.83	152.29	0	0	323	Beverages	Thai	647
2	1196273	3	132.92	133.92	0	0	96	Beverages	Thai	647
3	1116527	4	135.86	134.86	0	0	163	Beverages	Thai	647
4	1343872	5	146.50	147.50	0	0	215	Beverages	Thai	647

category	cuisine	city_code	region_code	center_type	op_area
Beverages	Thai	647	56	TYPE_C	2.0
Beverages	Thai	647	56	TYPE_C	2.0
Beverages	Thai	647	56	TYPE_C	2.0
Beverages	Thai	647	56	TYPE_C	2.0
Beverages	Thai	647	56	TYPE_C	2.0

Display the list of columns present in trainfinal table and store it in variable "cols".

```
cols = trainfinal.columns.tolist()
print(cols)
```

```
['id', 'week', 'checkout_price', 'base_price', 'emailer_for_promotion', 'homepage_featured', 'num_orders', 'category', 'cuisine', 'city_code', 'region_code', 'center_type', 'op_area']
```

Rearrange the columns by slicing the columns of "cols" and print "cols"

```
cols = cols[:2] + cols[9:] + cols[7:9] + cols[2:7]
print(cols)
```

```
['id', 'week', 'city_code', 'region_code', 'center_type', 'op_area', 'category', 'cuisine', 'checkout_price', 'base_price', 'emailer_for_promotion', 'homepage_featured', 'num_orders']
```

Store the changes of columns in trainfinal and display the datatypes of trainfinal using trainfinal.dtypes. Here, we can see that, we not only have numerical data but we also have object data.

```
trainfinal.dtypes
```

id	int64
week	int64
city_code	int64
region_code	int64
center_type	object
op_area	float64
category	object
cuisine	object
checkout_price	float64
base_price	float64
emailer_for_promotion	int64
homepage_featured	int64
num_orders	int64

Label Encoding

Typically, any structured dataset includes multiple columns with a combination of numerical as well as categorical variables. A machine can only understand the numbers. It cannot understand the text. That's essentially the case with Machine Learning algorithms too.

Reference Link:

https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/?utm_source=blog&utm_medium=one-hot-encoding-vs-label-encoding-using-scikit-learn

We need to convert each text category to numbers for the machine to process those using mathematical equations.

How should we handle categorical variables? There are multiple ways to handle this, but we will see one of them is Label Encoding.

Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

Let's see how to implement label encoding in Python using the scikit-learn library.

As we have to convert only the text class category columns, we first select it then we will implement Label Encoding to it.

```
from sklearn.preprocessing import LabelEncoder

lb1 = LabelEncoder()
trainfinal['center_type'] = lb1.fit_transform(trainfinal['center_type'])

lb2 = LabelEncoder()
trainfinal['category'] = lb1.fit_transform(trainfinal['category'])

lb3 = LabelEncoder()
trainfinal['cuisine'] = lb1.fit_transform(trainfinal['cuisine'])
```

In the above code we have selected text class categorical columns and performed label encoding.

```
final.head()
```

region_code	center_type	op_area	category	cuisine	checkout_price	base_price	emailer_for_promotion	home
56	2	2.0	0	3	136.83	152.29	0	0
56	2	2.0	0	3	135.83	152.29	0	0
56	2	2.0	0	3	132.92	133.92	0	0
56	2	2.0	0	3	135.86	134.86	0	0
56	2	2.0	0	3	146.50	147.50	0	0

We notice the output of the above code, after performing label encoding, alphabetical classes- “Center Type, Category and City code are converted to numeric values.

```
trainfinal.shape
```

```
(456548, 13)
```

Finally, display the number of rows and columns of trainfinal using shape()

Data Visualization

Data visualization is where a given data set is presented in a graphical format. It helps the detection of patterns, trends, and correlations that might go undetected in text-based data.

Understanding your data and the relationship present within it is just as important as any algorithm used to train your machine-learning model. Even the most sophisticated machine learning models will perform poorly on data that wasn't visualized and understood properly.

To visualize the dataset we need libraries called Matplotlib and Seaborn.

The matplotlib library is a Python 2D plotting library that allows you to generate plots, scatter plots, histograms, bar charts etc.

Let's visualize our data using Matplotlib and seaborn library.

Before diving into the code, let's look at some of the basic properties we will be using when plotting.

xlabel: Set the label for the x-axis.

ylabel: Set the label for the y-axis.

title: Set a title for the axes.

Legend: Place a legend on the axes.

Let's visualize our data using Matplotlib and seaborn library.

Univariate Analysis

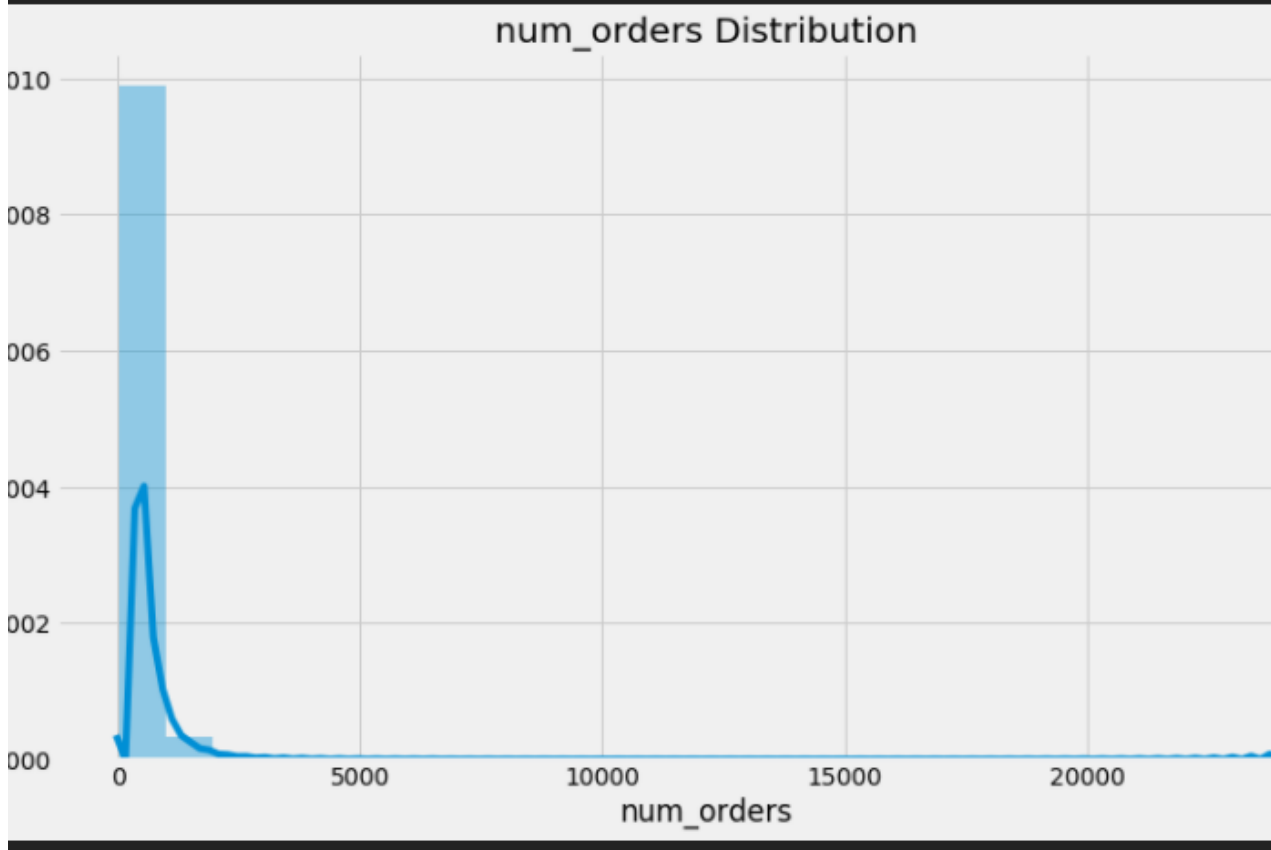
Univariate analysis is the simplest form of data analysis where the data being analyzed contains only one variable.

Bivariate Analysis

It involves the analysis of two variables (often denoted as X, Y), for the purpose of determining the empirical relationship between them.

```
plt.style.use('fivethirtyeight')
plt.figure(figsize=(12,7))
sns.distplot(trainfinal.num_orders, bins = 25)
plt.xlabel("num_orders")
plt.ylabel("Number of Buyers")
plt.title("num_orders Distribution")
```

```
5, 1.0, 'num_orders Distribution')
```



Drop the column “id” and find the correlation between the columns.

```
trainfinal2 = trainfinal.drop(['id'], axis=1)
correlation = trainfinal2.corr(method='pearson')
columns = correlation.nlargest(8, 'num_orders').index
columns

Index(['num_orders', 'homepage_featured', 'emailer_for_promotion', 'op_area',
      'cuisine', 'city_code', 'region_code', 'category'],
      dtype='object')
```

Correlation is a statistical relationship between two variables and it could be positive, meaning both variables move in the same direction, or negative, meaning that when one variable's value increases, the other variables' values decrease.

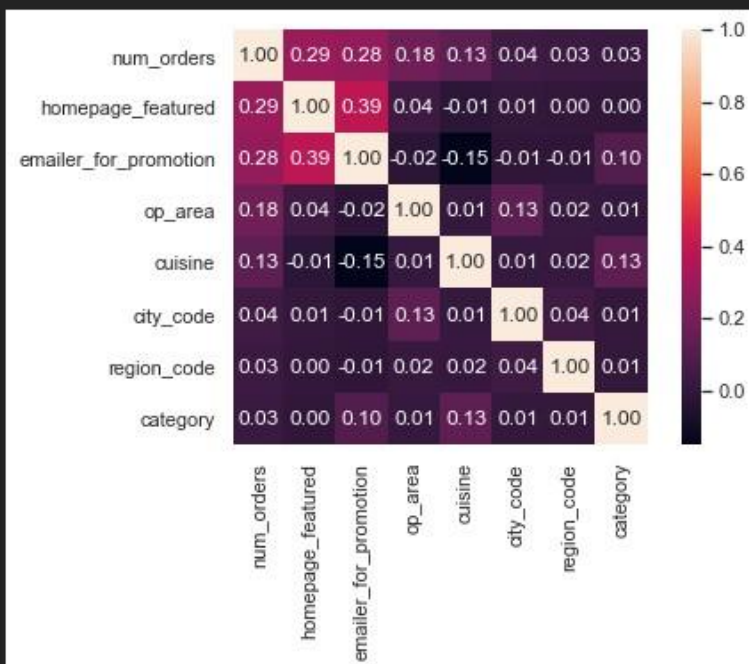
With the help of seaborn heatmap we will be plotting the heatmap and for finding the correlation between variable we have `corr()` available.


```

correlation_map = np.corrcoef(trainfinal2[columns].values.T)
sns.set(font_scale=1.0)
heatmap = sns.heatmap(correlation_map, cbar=True, annot=True, square=True, fmt='.2f',
                      yticklabels=columns.values, xticklabels=columns.values)

plt.show()

```



Splitting The Dataset Into Dependent And Independent Variable

In machine learning, the concept of dependent variables (y) and independent variables(x) is important to understand. Here, the Dependent variable is nothing but output in the dataset and the independent variable is all inputs in the dataset.

With this in mind, we need to split our dataset into the matrix of independent variables and the vector or dependent variable. Mathematically, a Vector is defined as a matrix that has just one column.

Let's split our dataset into independent and dependent variables.

1. The independent variable in the dataset would be considered as 'x' and the 'homepage_featured', 'emailer_for_promotion', 'op_area', 'cuisine', 'city_code', 'region_code', and 'category' columns would be considered as the independent variable.

2. The dependent variable in the dataset would be considered as 'y' and the 'num_orders' column is considered as the dependent variable.

Now we will split the data into independent and dependent variables,

```
features = columns.drop(['num_orders'])
trainfinal3 = trainfinal[features]
X = trainfinal3.values
y = trainfinal['num_orders'].values
```

```
trainfinal3.head()
```

	homepage_featured	emailer_for_promotion	op_area	cuisine	city_code	region_code	category
0	0	0	2.0	3	647	56	0
1	0	0	2.0	3	647	56	0
2	0	0	2.0	3	647	56	0
3	0	0	2.0	3	647	56	0
4	0	0	2.0	3	647	56	0

Split The Dataset Into Train Set And Test Set

When you are working on a model and you want to train it, you obviously have a dataset.

However, after training, we have to test the model on some test datasets. For this, you will a dataset that is different from the training set you used earlier. However, it might not always be possible to have so much data during the development phase. In such cases, the solution is to split the dataset into two sets, one for training and the other for testing.

But the question is, how do you split the data? You can't possibly manually split the dataset into two sets. And you also have to make sure you randomly split the data. To help us with this task, the Scikit library provides a tool, called the Model Selection library. There is a class in the library which is, 'train_test_split.' Using this we can easily split the dataset into the training and the testing datasets in various proportions.

The train-test split is a technique for evaluating the performance of a machine-learning algorithm.

Train Dataset: Used to fit the machine learning model.

Test Dataset: Used to evaluate the fit machine learning model.

In general, you can allocate 80% of the dataset to training set and the remaining 20% to test set. We will create 4 sets— X_train (training part of the matrix of features), X_val (test part of the matrix of features), Y_train (training part of the dependent variables associated with the X train sets, and therefore also the same indices), Y_val (test part of the dependent variables associated with the X_val sets, and therefore also the same indices).

There are a few other parameters that we need to understand before we use the class:

test_size — this parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.5 as the value, the dataset will be split 50% as the test dataset

train_size — you have to specify this parameter only if you're not specifying the test_size. This is the same as test_size, but instead you tell the class what percent of the dataset you want to split as the training set.

Now split our dataset into train set and test using train_test_split class from scikit learn library.

Model Building

- Predictive modeling is a mathematical approach to create a statistical model to forecast future behavior based on input test data.
- Steps involved in predictive modeling:
- Algorithm Selection:
 - When we have the structured dataset, and we want to estimate the continuous or categorical outcome then we use supervised machine learning methodologies like regression and classification techniques. When we have unstructured data and want to predict the clusters of items to which a particular input test sample belongs, we use unsupervised algorithms. An actual data scientist applies multiple algorithms to get a more accurate model.
- Train Model:
 - After assigning the algorithm and getting the data handy, we train our model using the input data applying the preferred algorithm. It is an action to determine the correspondence between independent variables, and the prediction targets.
- Model Prediction:
 - We make predictions by giving the input test data to the trained model. We measure the accuracy by using a cross-validation strategy or ROC curve which performs well to derive model output for test data.
- Model building includes the following main tasks
 - Train and test model algorithms
 - Evaluation of Model
 - Save the model
 - Predicting the output using the model

Train And Test Model Algorithms

There are several Machine learning algorithms to be used depending on the data you are going to process such as images, sound, text, and numerical values. The algorithms that you can choose according to the objective that you might have it may be Classification algorithms or Regression algorithms.

Example: 1. Linear Regression.

2. Lasso Regression.

3. ELasticNet Regression / Classification.

4. Decision Tree Regression / Classification.

5. KNeighbors Regressor

6. Gradient Boosting Regressor

7. XGB Regressor

You will need to train the datasets to run smoothly and see an incremental improvement in the prediction rate. Import these libraries using:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
```

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable.

Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters).

Elastic Net is an extension of linear regression that adds regularization penalties to the loss function during training.

A decision tree is a supervised machine learning model used to predict a target by learning decision rules from features.

KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the same neighborhood.

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

Model Evaluation

We're going to use `x_train` and `y_train` obtained above in `train_test_split` section to train our regression model. We're using the `fit` method and passing the parameters as shown below.

Finally, we need to check to see how well our model is performing on the test data.

Regression Evaluation Metrics:

RMSE: Root Mean Square Error

RMSE is the square root of the averaged squared difference between the target value and the value predicted by the model. It is preferred more in some cases because the errors are first squared before averaging which poses a high penalty on large errors. This implies that RMSE is useful when large errors are undesired.

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (Predicted_i - Actual_i)^2}{N}}$$

For testing the model we use the below method,

```
XG = XGBRegressor()
XG.fit(X_train, y_train)
y_pred = XG.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSLE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))

RMSLE: 70.71297763309899
```

```
LR = LinearRegression()
LR.fit(X_train, y_train)
y_pred = LR.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSLE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))

RMSLE: 129.06868252180055
```

```
L = Lasso()
L.fit(X_train, y_train)
y_pred = L.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSLE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))
```

RMSLE: 128.65858871245018

```
EN = ElasticNet()
EN.fit(X_train, y_train)
y_pred = EN.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSLE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))
```

RMSLE: 130.97849267827252

```
KNN = KNeighborsRegressor()
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSLE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))
```

RMSLE: 67.18569592088637

```
GB = GradientBoostingRegressor()
GB.fit(X_train, y_train)
y_pred = GB.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSLE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))
```

RMSLE: 99.31565587258103

Save The Model

After building the model we have to save the model. Here, we are using decision tree model as we have got best RMSE value compared to others.

Pickle is used for serializing and de-serializing Python object structures, also called marshalling or flattening. Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network. Later on, this character stream can then be retrieved and de-serialized back to a Python object.

This is done by the below code

```
import pickle
pickle.dump(DT,open('fdemand.pkl','wb'))
```

Here, DT is our decision tree model saving as fdemand.pkl file. Wb is the write binary in bytes.

Predicting The Output Using The Model

Here, we are creating X_test which we are using to test the model to predict the number of orders by giving input to the model build.

```
testfinal = pd.merge(test, meal_info, on="meal_id", how="outer")
testfinal = pd.merge(testfinal, center_info, on="center_id", how="outer")
testfinal = testfinal.drop(['meal_id', 'center_id'], axis=1)

tcols = testfinal.columns.tolist()
tcols = tcols[:2] + tcols[8:] + tcols[6:8] + tcols[2:6]
testfinal = testfinal[tcols]

lb1 = LabelEncoder()
testfinal['center_type'] = lb1.fit_transform(testfinal['center_type'])

lb2 = LabelEncoder()
testfinal['category'] = lb1.fit_transform(testfinal['category'])

lb3 = LabelEncoder()
testfinal['cuisine'] = lb1.fit_transform(testfinal['cuisine'])

X_test = testfinal[features].values
```

```
pred = DT.predict(X_test)
pred[pred<0] = 0
submit = pd.DataFrame({
    'id' : testfinal['id'],
    'num_orders' : pred
})
```

Here, DT is our decision tree model saving as fdemand.pkl file. Wb is the write binary in bytes.

```
submit.to_csv("submission.csv", index=False)
```

```
submit.describe()
```

	id	num_orders
count	3.257300e+04	32573.000000
mean	1.248476e+06	262.795966
std	1.441580e+05	364.258059
min	1.000085e+06	15.530303
25%	1.123969e+06	64.403756
50%	1.247296e+06	148.947368
75%	1.372971e+06	324.777344
max	1.499996e+06	5323.888889

Application Building

Application Building involves following steps

1. Create an HTML file
2. Build a Python Code
3. Run the app

Create An HTML File

We use HTML to create the front-end part of the web page.

Here, we created 2 html pages- home.html, upload.html.

home.html displays the home page.

upload.html accepts the values from the user and displays the prediction.

For more information regarding HTML refer to the link below

We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.

Build Python Code

Let us build the flask file 'app.py' which is a web framework written in Python for server-side scripting. Let's see the step-by-step procedure for building the backend application.

App starts running when the "__name__" constructor is called in the main.

render_template is used to return html file.

The "GET" method is used to take input from the user.

The "POST" method is used to display the output to the user.

Importing Libraries

Libraries required for the app to run are to be imported.

Creating our Flask app and loading the model

Now after all the libraries are imported, we will be creating our flask app. and loading our model into our flask app.

Routing to the HTML Page:

@app.route is used to route the application where it should route.

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

Here, "home.html" is rendered when the home button is clicked on the UI, and

"upload.html" is rendered when predict button is clicked.

```
# import the necessary packages
import pandas as pd
import numpy as np
import pickle
import os
from flask import Flask, request, render_template
```

Firstly, we render the home.html template and from there, we navigate to our prediction page which is upload.html. We enter input values here and these values are sent to the loaded model and the resultant output is displayed on upload.html.

```
@app.route('/', methods=['GET'])
def index():
    return render_template('home.html')
@app.route('/home', methods=['GET'])
def about():
    return render_template('home.html')
@app.route('/pred', methods=['GET'])
def page():
    return render_template('upload.html')
```

Main Function

This is used to run the application in a local host.

```
@app.route('/predict', methods=['GET', 'POST'])
def predict():
    print("[INFO] Loading model...")
    model = pickle.loads(open('fdemand.pkl', "rb").read())
    input_features = [float(x) for x in request.form.values()]
    features_value = [np.array(input_features)]
    print(features_value)

    features_name = ['homepage_featured', 'emailer_for_promotion', 'op_area', 'cuisine',
                    'city_code', 'region_code', 'category']
    prediction = model.predict(features_value)
    output=prediction[0]
    print(output)
    return render_template('upload.html', prediction_text=output)
```

The local host runs on port number 8000.

Run The App

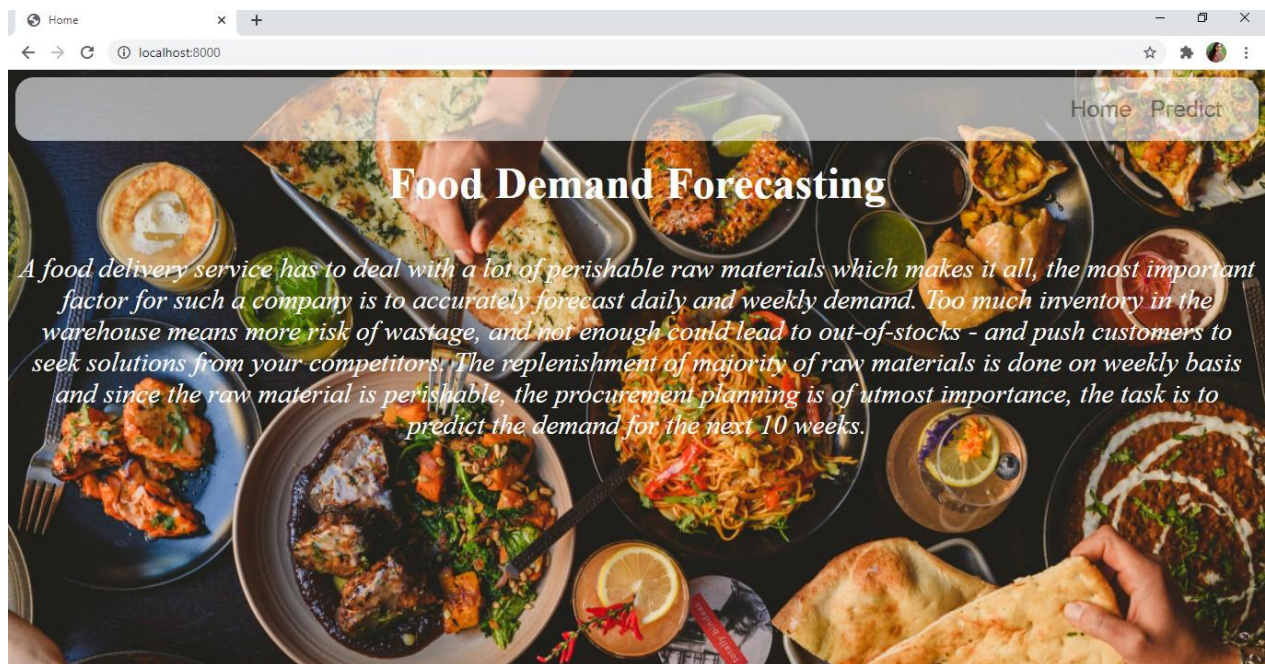
Run the application from anaconda prompt

- Open new anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- It will show the local host where your app is running on http://127.0.0.1:8000/
- Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.
- Enter the values, click on the predict button and see the result/prediction on the web page.

```
(base) C:\Users\rincy\anaconda3\Food Forecasting\Flask>python app.py
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:8000/ (Press CTRL+C to quit)
```

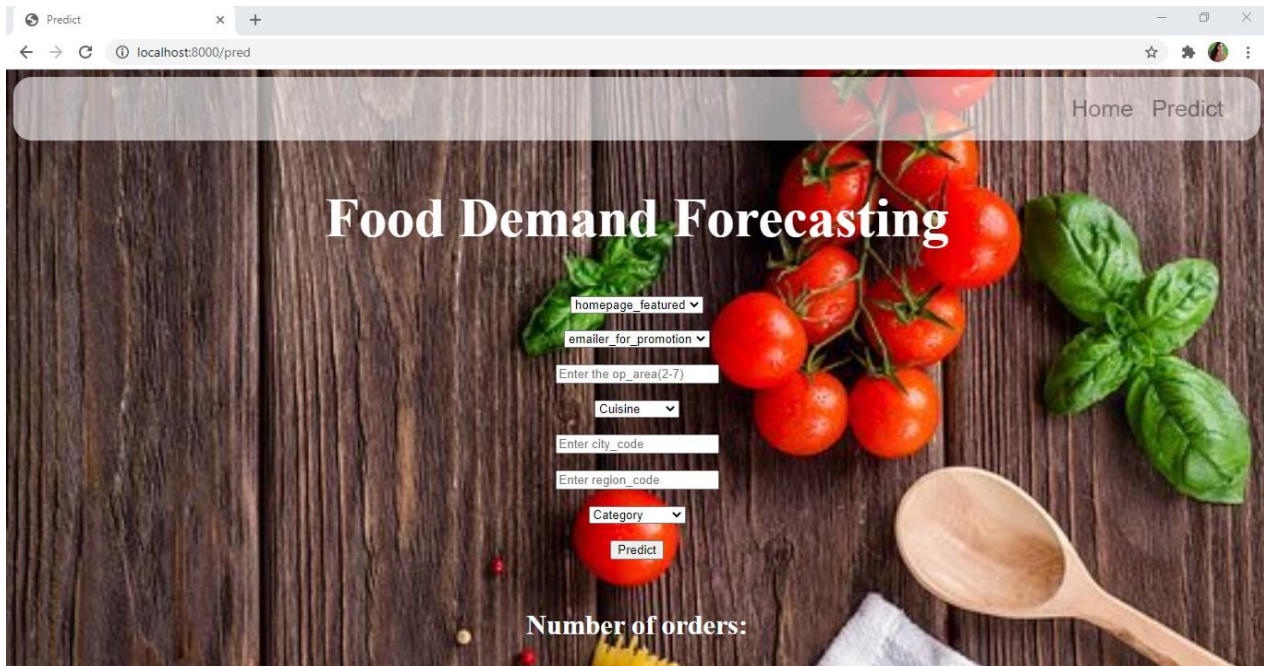
Showcasing the output on UI

The home page is displayed when the home button is clicked.



The predict page is displayed when the predict button is clicked.

Enter input values to predict a number of orders.



Predict

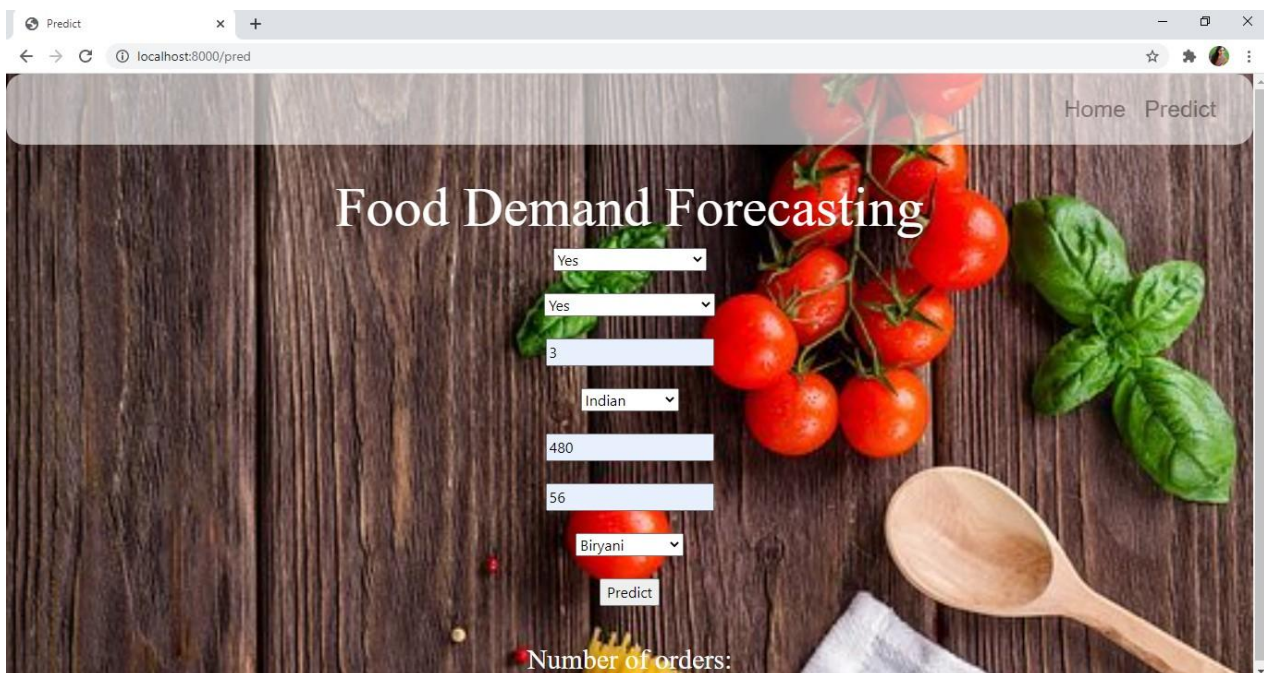
localhost:8000/pred

Home Predict

Food Demand Forecasting

homepage_featured ▾
emailer_for_promotion ▾
Enter the op_area(2-7)
Cuisine ▾
Enter city_code
Enter region_code
Category ▾
Predict

Number of orders:



Predict

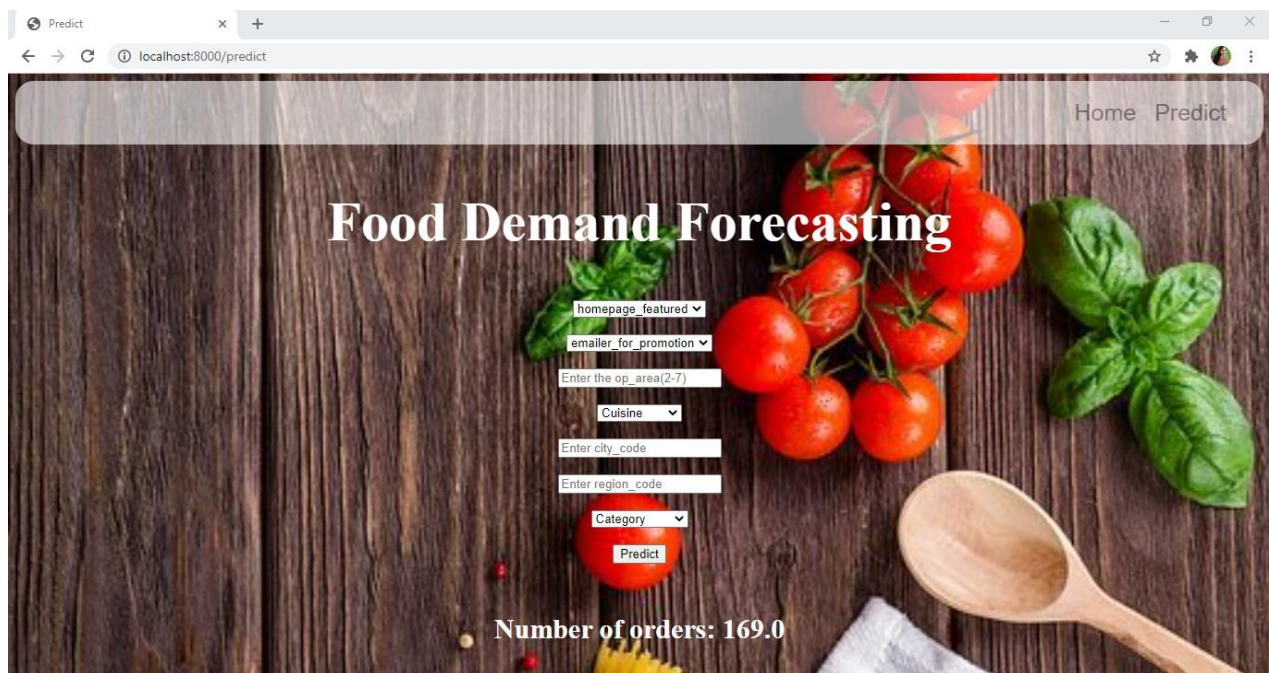
localhost:8000/pred

Home Predict

Food Demand Forecasting

Yes ▾
Yes ▾
3
Indian ▾
480
56
Biryani ▾
Predict

Number of orders:



Finally, the prediction for the given input features is shown.

Train The Model On IBM

In this milestone, you will learn how to build a Machine Learning Model and deploy it on the IBM Cloud.