

**Food Demand Forecasting For Food  
Delivery Company**

**INTEGRATED DEVELOPMENT ENVIRONMENT LAB**

**PROJECT**

**K. KAUSHIK DATTA**

**B.Tech Final year**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **NAME** | **PAGE NO** |
| **1** | **ABSTRACT** | **2** |
| **2** | **INTRODUCTION** | **3** |
| **3** | **SOFTWARE REQUIREMENTS AND SPECIFICATIONS** | **4** |
| **4** | **METHODOLOGY** | **5-7** |
| **5** | **DATA ANALYSIS** | **8-11** |
| **6** | **ALGORITHMS IMPLEMENTED** | **12** |
| **7** | **EVALUATION METRICS** | **13** |
| **8** | **WEB APPLICATIONS** | **14** |
| **9** | **CODE AND OUTPUT** | **14-42** |

**ABSTRACT**

* A food delivery service has to deal with a lot of perishable raw materials which makes it all, the most important factor for such a company is to accurately forecast daily and weekly demand.
* Too much inventory in the warehouse means more risk of wastage, and not enough could lead to out-of-stocks - and push customers to seek solutions from your competitors.
* The replenishment of the majority of raw materials is done on a weekly basis and since the raw material is perishable, the procurement planning is of utmost importance, the task is to predict the demand for the next 10 weeks.

**INTRODUCTION**

Demand forecasting is a key component to every growing online business.Without proper demand forecasting processes in place, it can be nearly impossible to have the right amount of stock on hand at any given time. A food delivery service has to deal with a lot of perishable raw materials which makes it all the more important for such a company to accurately forecast daily and weekly demand.Too much inventory in the warehouse means more risk of wastage, and not enough could lead to out-of-stocks — and push customers to seek solutions from your competitors.The client is a meal delivery company which operates in multiple cities. They have various fulfillment centers in these cities for dispatching meal orders to their customers. The client wants to forecast the demand in these centers for upcoming weeks so that these centers can plan the stock of raw materials accordingly.The replenishment of majority of raw materials is done on a weekly basis and since the raw material is perishable, the procurement planning is of utmost importance. Secondly, staffing of the centers is also one area wherein accurate demand forecasts are really helpful.

**SOFTWARE REQUIREMENTS AND SPECIFICATIONS**

**IDE Used-Jupyter Notebook:**

The Jupyter Notebook is a living online notebook, letting faculty and students weave together computational information (code, data, statistics) with narrative, multimedia, and graphs.

Faculty can use it to set up interactive textbooks, full of explanations and examples which students can test out right from their browsers.

Students can use it to explain their reasoning, show their work, and draw connections between their classwork and the world outside.

Scientists, journalists, and researchers can use it to open up their data, share the stories behind their computations, and enable future collaboration and innovation.

# Programming language-Python

* As there are a number of programming languages available, we still choose Python because it is easy to learn, friendly to use, readable and maintainable.
* Python is an interpreted, high-level and general purpose programming language.
* It is a multi-paradigm programming language, many of its features supports

object-oriented programming and structured programming, functional

programming and aspect-oriented programming.

* Rather than having all of its functionality built into its core, Python was designed to be highly extensible.

# Web Based Application

A web based application is an application software that runs on a web server, unlike computer

based software programs that are stored locally on the operating system of the device.

Web applications are accessed by the user through a web browser with an active internet connection. Web applications can be designed for a wide variety of uses and can be used by anyone, from an organization to individual users for numerous reasons.

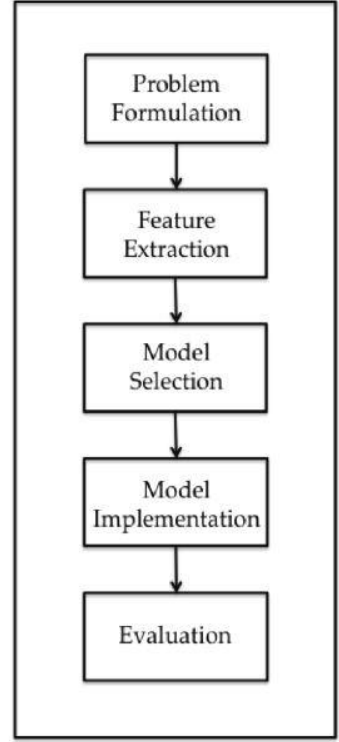
We used Flask here to connect the frontend and backend code. Flask is a web framework used for smaller applications.

**METHODOLOGY**

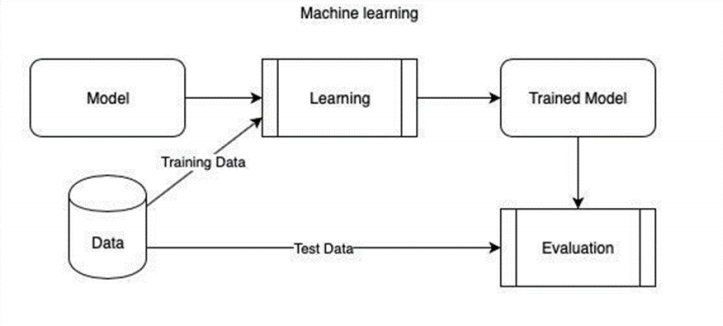
We developed web applications by using Python language which is an interpreted and high-level programming language and using Machine Learning algorithms. For writing the code we used the Jupyter Notebook environment of the Anaconda distributions and the Spyder IDE, which is an integrated scientific programming in the python language. For creating a user interface for the prediction and taking the required factors/parameters, we used Flask (which is a micro web framework written in Python. It is classified as a micro-framework because it does not require particular tools or libraries), and a scripting language to create a webpage i.e. HTML by creating the templates to use in the functions of the Flask.

**Technologies/Tools Used:**

* Jupyter Notebook Environment
* Spyder IDE
* Machine Learning Algorithms
* Python( numpy, pandas, matplotlib, seaborn, sklearn)
* HTML,CSS
* Flask



**DESIGN**

****

**STEP 1-DATA ACQUISITION:**

Data must be acquired for the implementation of our project. We acquire the dataset required from Kaggle.

**STEP 2-DATA PREPARATION:**

The dataset acquired is not in the proper format. So, we must make certain reformations to obtain the dataset according to our project. We drop the attributes which we do not require and import the final dataset. We remove any null or missing values in the dataset.

**STEP 3-DATA TRANSFORMATION:**

Here we are going to transform the data accordingly. We have to use label encoding to convert the string values to numerical values and then use scaling to bring all the values in the same range. We should split the dataset into independent variables and dependent variables. Once this is completed, we are going to split into a training set and testing set.

**STEP 4-TRAIN DATA:**

75% of the data is training data that is used to train the model.

**STEP 5-TEST DATA:**

25% of the data is testing data that is used to test the model.

**STEP 6-MODEL BUILDING:**

We are going to build the model which gives output and the required predictions.

**STEP 7-MODEL EVALUATION:**

We are going to use r2-score, RMSE (Root Mean Squared Error) to evaluate the accuracy of our model.

**STEP 8- APPLICATION BUILDING:**

We dumped the ML code onto the local system using pickle package. For creating a website, HTML and CSS code is written and saved in the folder where the dumped pkl file is saved. For connecting the front end website and the ML code we wrote a code using Flask in Spyder IDE.

**DATA ANALYSIS**

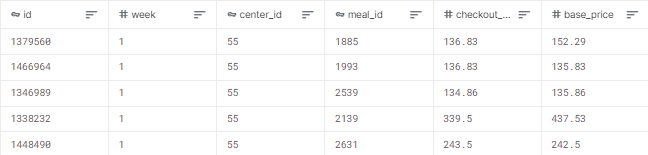
**Source of Data:**

Kaggle is an online community for descriptive analysis and predictive modelling. It collects a variety of research fields dataset from data analytic practitioners. Data scientists compete to build the best model for both descriptive and predictive analytic. It however allows an individual to access their dataset in order to create models and also work with other data scientists to solve various real-world analytics problems. The input dataset used in developing this model has been downloaded from Kaggle.

**Structure of Dataset:**

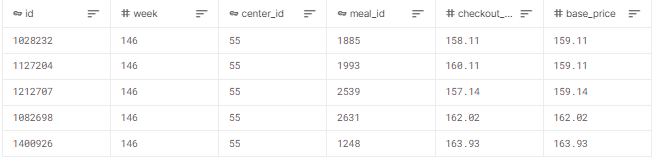
The dataset contains twelve 5 csv files representing the different features, where a particular set of columns are used to predict the Food Demand. The columns in the dataset are:

1. **train.csv:**

****

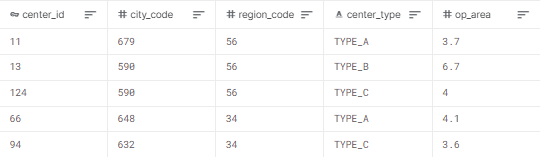
****

1. **test.csv**

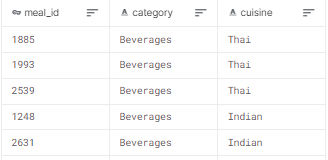
****

****

1. **fulfilment\_center\_info.csv**

****

1. **meal\_info.csv**

****

1. **sample\_submission.csv**

****

**Data Preprocessing**

Once we have read the dataset, we have to perform data pre-processing. Coming to this we use four libraries Pandas, NumPy, matplotlib, seaborn. We load the given dataset and check for any null values in the dataset. If there are any null values then fill them using mean if it’s a continuous value and with mode it it’s a categorical value. If there are any unwanted columns or rows we drop them which in this case are the year, state, station code and location columns.

**Calculation of num\_orders Index:**

After completing the pre-processing by eliminating/replacing null values and drop non-required columns, we normalize each column individually based on the type of parameter it is and scale it uniformly in a particular range. After normalizing the given parameters, we use it to calculate the num\_orders through one of the standard techniques/formulae. We add the num\_orders column to the dataset and calculate the food demand for each row. This obtained num\_orders column will be the dependent column of our data.

**Correlation Analysis:**

A correlation analysis was employed in this study to examine if explanatory variables share the same linear relationship with the outcome variable in order to detect duplications of variables in the dataset. Among other things, highly correlations between variables were observed in the dataset. The Pearson correlation coefficient r, takes a range of values between +1 to -1. A value of 0 indicates that there is no relationship between the two variables. A value less than zero indicates a negative relationship and a value greater than zero connotes a positive association: that is as one unit of variable increases, so does the value of the other variable.

**Partition of Data:**

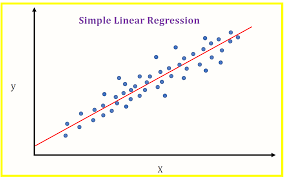
After that we separate the data into dependent and independent variables. Now the dataset is partitioned into two parts for training and testing purposes: 75% of the entire dataset for training the selected models and 25% for testing purposes. Most importantly, the respective training and validation dataset were randomly sampled to circumvent sampling bias.

**Algorithms Implemented**

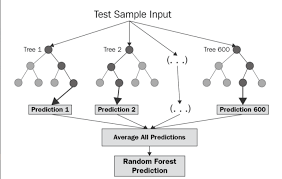
In this project the output to be predicted is continuous value, so we are using supervised learning algorithms in particular Regression Algorithms. Linear Regression, Multiple Linear Regression, Random Forest, Lasso Regression, Support Vector Machines etc are the regression algorithms.

For building a Machine Learning model to predict the required output value we have used two different algorithms.

1. Linear Regression: It is a statistical technique that uses several explanatory variables to predict the outcome of a response variable.



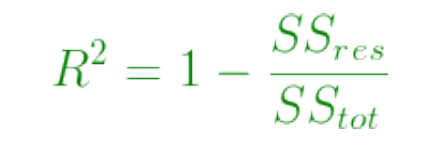
1. Random Forest Regressor: A random forest is a meta-estimator (i.e. it combines the result of multiple predictions) which aggregates many decision trees. This algorithm works efficiently even when the correlation between the variables is low. Using this Random Forest Regressor the obtained r2 score was 0.66 (66% accurate approx.)



**Evaluation metrics**

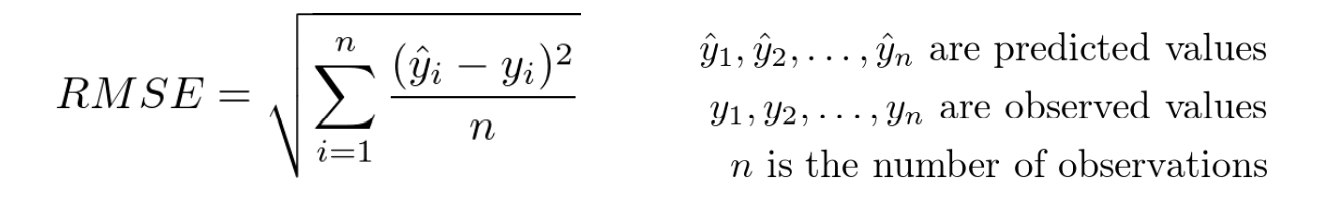
The evaluation metrics that we have used in the project are :

1. **R2 Score**: R-squared is a statistical measure that represents the goodness of fit of a regression model. The ideal value for r-square is 1. The closer the value of r-square to 1, the better is the model fitted.



Where SSres is the residual sum of squares and SStot is the total sum of squares.

1. **RMSE**: Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are. It is also a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.



**Pickle Package:**

After calculating evaluation metrics, the built model is to be dumped into a file using pickle package. This has to be saved in the local system where html and other python files are saved.

pickle.dump() is used to store the object data to the file. This function takes 3 arguments. The first argument is the object that you want to store. The second argument is the file object you get by opening the desired file in write-binary (wb) mode. And the third argument is the key-value argument.

Then a pkl extension file is generated on the jupyter notebook home page which is to be saved to the local system for easy accessing.

**WEB APPLICATION**

For creating the website HTML and CSS are used to write the code.

HTML(Hypertext Markup Language) and CSS (Cascading Style Sheets) are two of the core technologies for building Web pages. HTML provides the structure of the page, CSS the (visual and aural) layout, for a variety of devices. Along with graphics and scripting, HTML and CSS are the basis of building Web pages and Web Applications.

Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications. Flask framework is used to load the dumped file and the html page connected which is imported from flask package. Other modules like render\_template and request are also imported along with flask.

When the created flask code is executed on Spyder IDE, then an IP address is generated with port 5000 which open the website. In this we can give the input values that are taken from the dataset and the predicted output is displayed on the website.

**CODE AND OUTPUT**

**ML Code:**

**Importing Libraries:**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

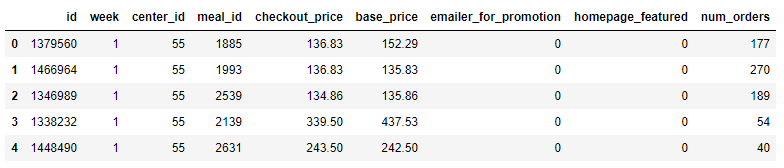
import seaborn as sns

**Loading Data:**

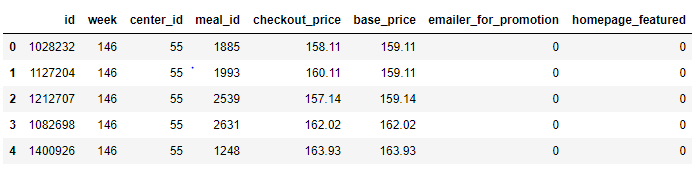
train = pd.read\_csv("train.csv")

test = pd.read\_csv("test.csv")

train.head()

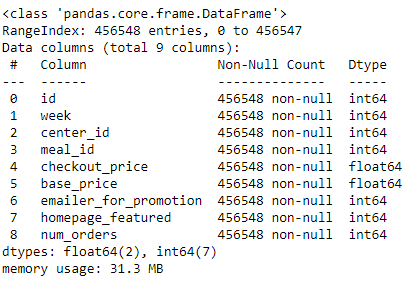


test.head()

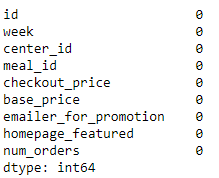


## Checking for null values

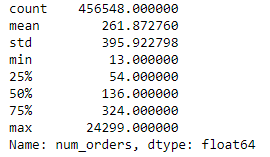
train.info()



train.isnull().sum()



train['num\_orders'].describe()



meal\_info = pd.read\_csv("meal\_info.csv")

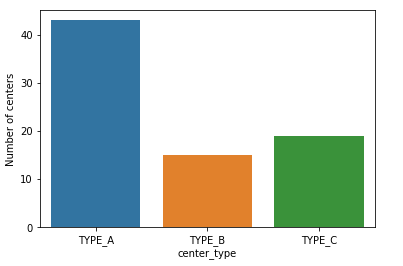
center\_info = pd.read\_csv("fulfilment\_center\_info.csv")

# Data Visualization

ax = sns.countplot(center\_info['center\_type'])

ax.set(ylabel='Number of centers')

plt.show()



print("Total Number of cities: ", center\_info['city\_code'].nunique())

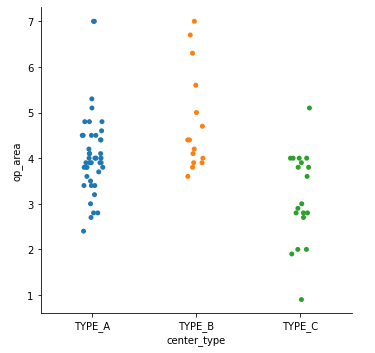
print("Total number of regions: ", center\_info['region\_code'].nunique())

Total Number of cities: 51

Total number of regions: 8

sns.catplot(x = 'center\_type', y = 'op\_area', data=center\_info)

plt.show()

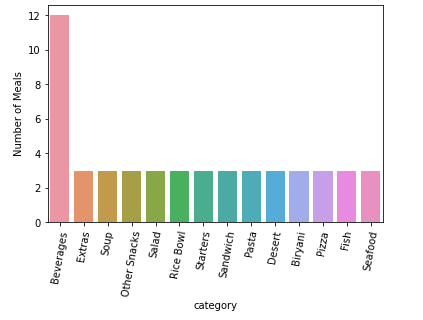


ax = sns.countplot(meal\_info['category'])

ax.set(ylabel= "Number of Meals")

plt.xticks(rotation=80)

plt.show()



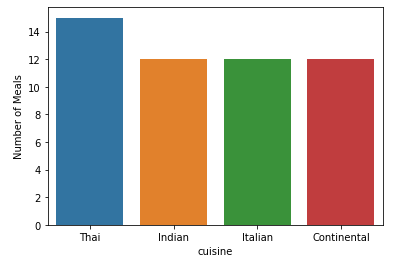
print("Total number of different types of meal: ", meal\_info['meal\_id'].nunique())

Total number of different types of meal: 51

ax = sns.countplot(meal\_info['cuisine'])

ax.set(ylabel= "Number of Meals")

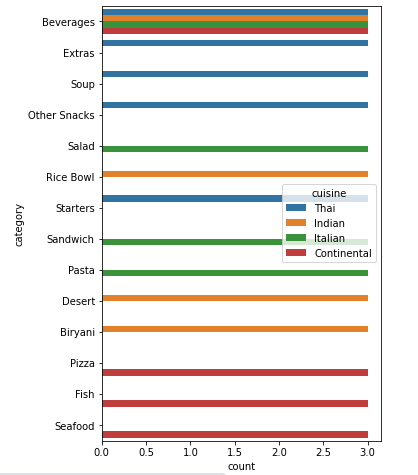
plt.show()



fig, ax = plt.subplots(figsize=(5,8))

sns.countplot(y = meal\_info['category'], hue=meal\_info['cuisine'], ax=ax)

plt.show()



plt.bar(train['emailer\_for\_promotion'].value\_counts().index, train['emailer\_for\_promotion'].value\_counts(), width=0.5, bottom=None, align='center', data=train, color='r')

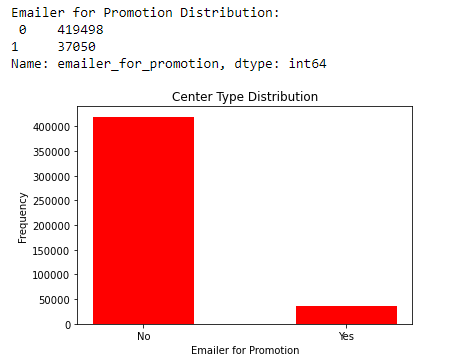
plt.title('Center Type Distribution')

plt.xticks(np.arange(2), ('No', 'Yes'))

plt.xlabel('Emailer for Promotion')

plt.ylabel('Frequency')

print('Emailer for Promotion Distribution:\n',train['emailer\_for\_promotion'].value\_counts())



plt.bar(train['homepage\_featured'].value\_counts().index, train['homepage\_featured'].value\_counts(), width=0.5, bottom=None, align='center', data=train, color='g')

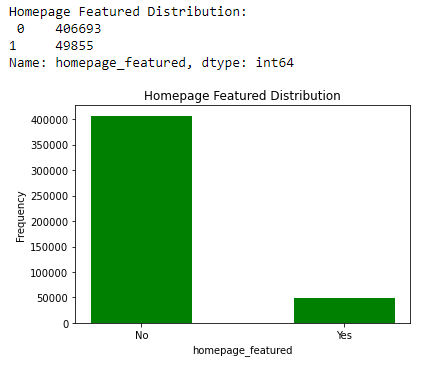
plt.title('Homepage Featured Distribution')

plt.xticks(np.arange(2), ('No', 'Yes'))

plt.xlabel('homepage\_featured')

plt.ylabel('Frequency')

print('Homepage Featured Distribution:\n',train['homepage\_featured'].value\_counts())



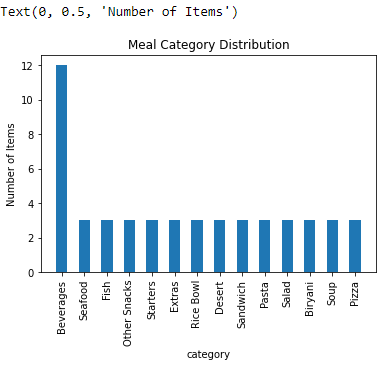
plt.bar(meal\_info['category'].value\_counts().index, meal\_info['category'].value\_counts(), width=0.5, bottom=None, align='center', data=meal\_info)

plt.title('Meal Category Distribution')

plt.xticks(rotation='vertical')

plt.xlabel('category')

plt.ylabel('Number of Items')

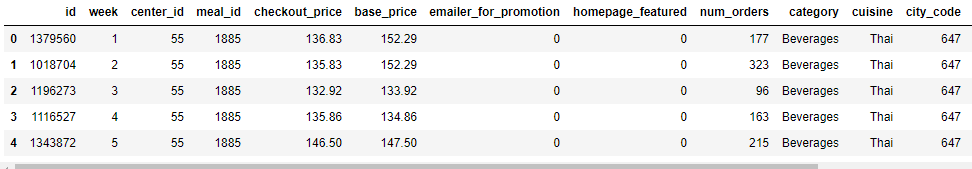


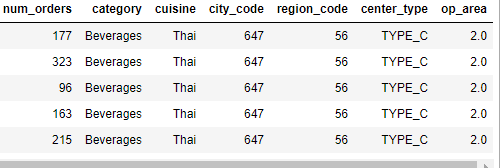
## Merging all csv files into the train table

trainfinal = pd.merge(train, meal\_info, on="meal\_id", how="outer")

trainfinal = pd.merge(trainfinal, center\_info, on="center\_id", how="outer")

trainfinal.head()





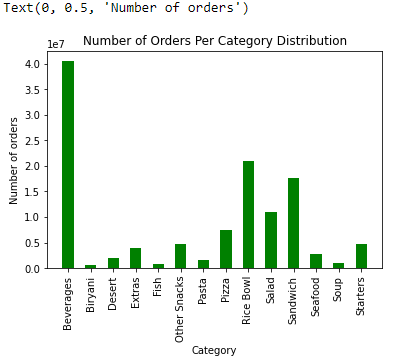
plt.bar(meal\_info.groupby( [ "category"] ).sum().index,trainfinal.groupby( [ "category"] )['num\_orders'].sum(), width=0.5, bottom=None, align='center', data=trainfinal, color='g')

plt.title('Number of Orders Per Category Distribution')

plt.xticks(rotation='vertical')

plt.xlabel('Category')

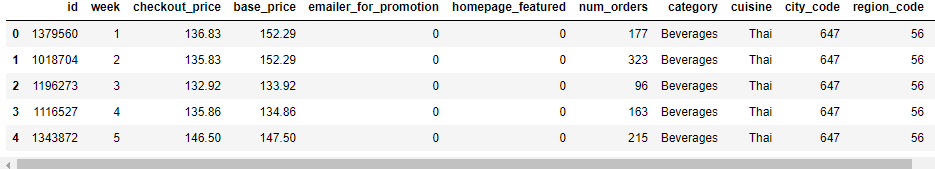
plt.ylabel('Number of orders')

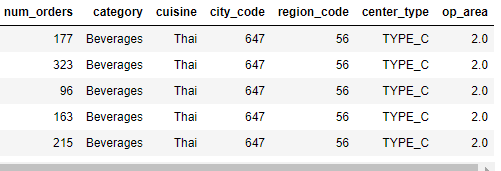


## Now we drop the columns center\_id and meal\_id

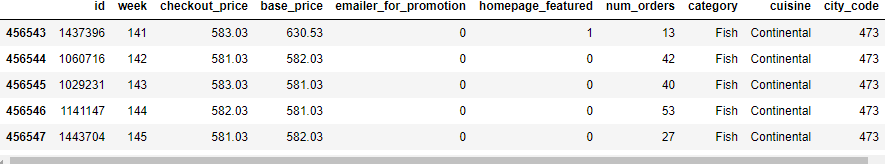
trainfinal = trainfinal.drop(['center\_id', 'meal\_id'], axis=1)

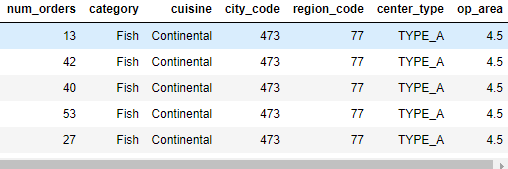
trainfinal.head()





trainfinal.tail()





## We will create a list of all the columns of trainfinal

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']

## We interchange the position of the table so that the details and meal and center will be after the week column

cols = cols[:2] + cols[9:] + cols[7:9] + cols[2:7]

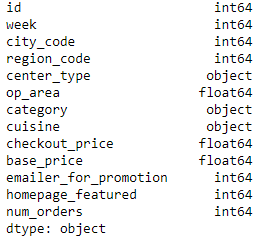
print(cols)

['id', 'week', 'city\_code', 'region\_code', 'center\_type', 'op\_area', 'category', 'cuisine', 'checkout\_price', 'base\_price', 'emailer\_for\_promotion', 'homepage\_featured', 'num\_orders']

## We pass these list as columns to the trainfinal dataset

trainfinal = trainfinal[cols]

trainfinal.dtypes



## Next we use LabelEncoder to convert the text in the center\_type, category, cuisine to integers

from sklearn.preprocessing import LabelEncoder

lb1 = LabelEncoder()

trainfinal['center\_type'] = lb1.fit\_transform(trainfinal['center\_type'])

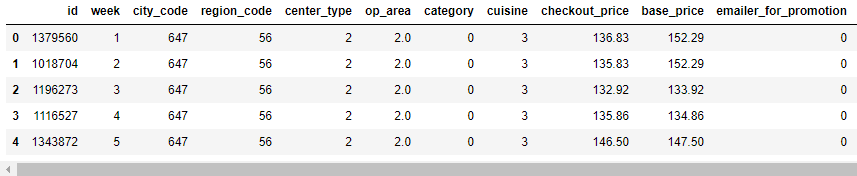
lb2 = LabelEncoder()

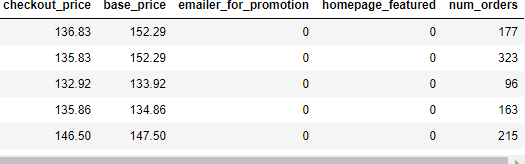
trainfinal['category'] = lb1.fit\_transform(trainfinal['category'])

lb3 = LabelEncoder()

trainfinal['cuisine'] = lb1.fit\_transform(trainfinal['cuisine'])

trainfinal.head()





## Analyzing the num\_orders column of trainfinal set

trainfinal.shape

(456548, 13)

plt.style.use('fivethirtyeight')

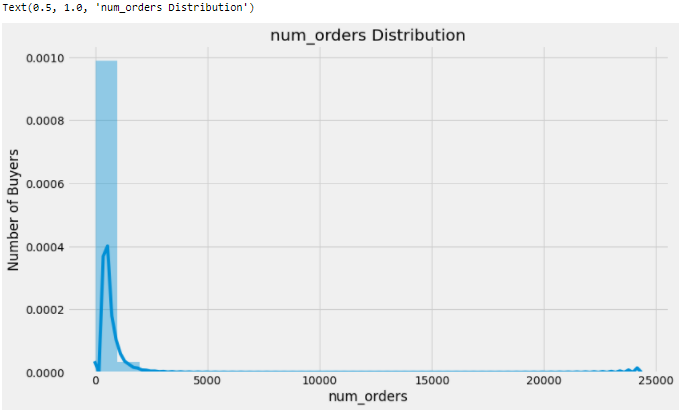
plt.figure(figsize=(12,7))

sns.distplot(trainfinal.num\_orders, bins = 25)

plt.xlabel("num\_orders")

plt.ylabel("Number of Buyers")

plt.title("num\_orders Distribution")



## Now we take the reciprocal of the num\_order column and check the result

def reciprocal(x):

y = 1/x

return y

hehe = reciprocal(trainfinal.num\_orders)

import math

def log(x):

y = math.log(x, 10)

return y

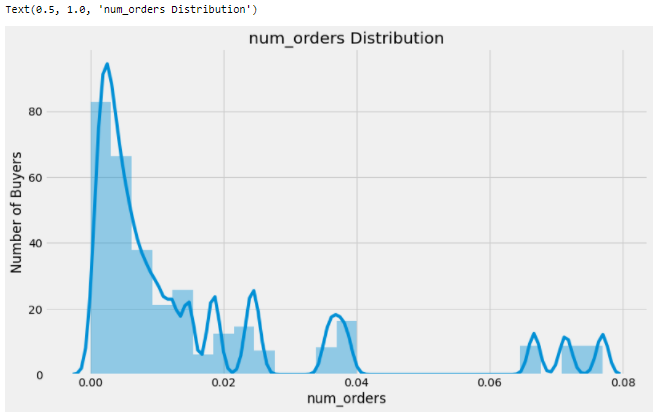
plt.figure(figsize=(12,7))

sns.distplot(hehe, bins = 25)

plt.xlabel("num\_orders")

plt.ylabel("Number of Buyers")

plt.title("num\_orders Distribution")



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')

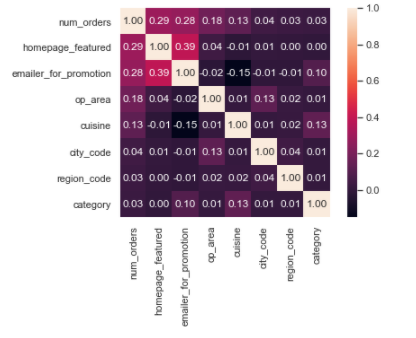
## We plot the heat map

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 data into training set and test set

features = columns.drop(['num\_orders'])

trainfinal3 = trainfinal[features]

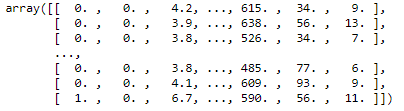
X = trainfinal3.values

y = trainfinal['num\_orders'].values

from sklearn.model\_selection import train\_test\_split

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.25)

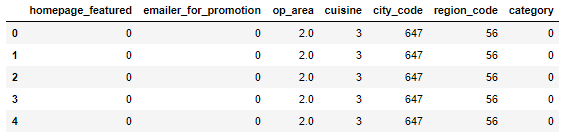
X\_train



y\_train



trainfinal3.head()



from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

sc.fit\_transform(X\_train)

sc.transform(X\_val)

from joblib import dump

dump(sc,"scalar.save")



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

## Linear Regression Model

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)))



## Lasso Model

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)))



## Elastic Net Model

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)))



## Decision Tree Model

DT = DecisionTreeRegressor()

DT.fit(X\_train, y\_train)

y\_pred = DT.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)))



## Random Forest

from sklearn.ensemble import RandomForestRegressor

rf=RandomForestRegressor(n\_estimators=10,random\_state=0,n\_jobs=-1)

rf.fit(X\_train,y\_train)



y\_pred = rf.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)))



from sklearn.metrics import r2\_score

r2\_score(y\_val,y\_pred)



import pickle

pickle.dump(rf,open('random.pkl','wb'))

## KNN Model

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)))



## Gradient Boost Model

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)))



## 

## 

## We merge the meal\_id and center\_id columns to the testfinal set and assign list of all columns to tcols

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()

print(tcols)



## We now rearrage the order of all the columns and perform Label Encoding to get numerical values

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()

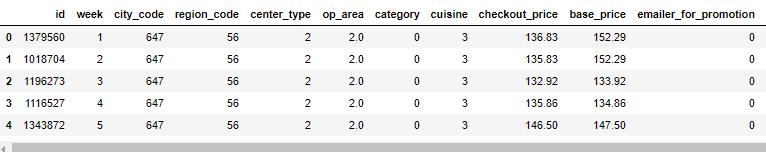
testfinal['cuisine'] = lb1.fit\_transform(testfinal['cuisine'])

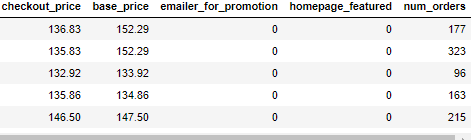
testfinal.head()



# This is the trainfinal set

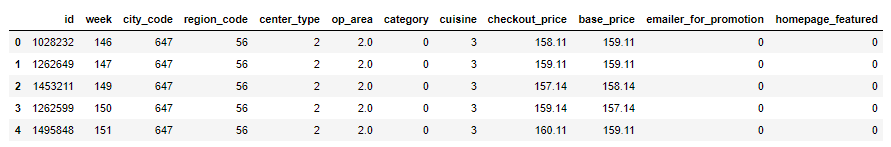
trainfinal.head()





## This is the testfinal set

testfinal.head()



X\_test = testfinal[features].values

features



## Now we create the dataframe which has id and num\_orders and store in the variable submit

pred = DT.predict(X\_test)

pred[pred<0] = 0

submit = pd.DataFrame({

'id' : testfinal['id'],

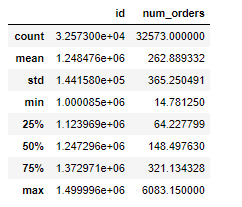
'num\_orders' : pred

})

## At last we convert the dataframe to a csv type

submit.to\_csv("submission.csv", index=False)

## This is the data for the customer id and the number of orders which are predicted



**HTML CODE**

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1">

<style>

body{

font-family: Calibri, Helvetica, sans-serif;

background-color: red;

}

.container {

padding: 50px;

background-color: #bfff00;

}

input[type=text], input[type=password], textarea {

width: 100%;

padding: 15px;

margin: 5px 0 22px 0;

display: inline-block;

border: none;

background: #f1f1f1;

}

input[type=text]:focus, input[type=password]:focus {

background-color: orange;

outline: none;

}

div {

padding: 10px 0;

}

hr {

border: 1px solid #f1f1f1;

margin-bottom: 25px;

}

.registerbtn {

background-color: orange;

color: white;

padding: 16px 20px;

margin: 8px 0;

border: none;

cursor: pointer;

width: 100%;

opacity: 0.9;

}

.registerbtn:hover {

opacity: 1;

}

</style>

</head>

<body>

<form action="http://localhost:5000/predict" method="POST">

<div class="container">

<center> <h1> Food Demand Forecasting For Food Delivery Company</h1> </center>

<hr>

<label> homepage\_featured </label>

<input type="text" name="do" placeholder= "homepage\_featured" size="15" required />

<label> emailer\_for\_promotion </label>

<input type="text" name="ph" placeholder= "emailer\_for\_promotion" size="15" required />

<label> op\_area </label>

<input type="text" name="co" placeholder= "op\_area" size="15" required />

<label> cuisine </label>

<input type="text" name="bod" placeholder= "cuisine" size="15" required />

<label> city\_code </label>

<input type="text" name="na" placeholder= "city\_code" size="15" required />

<label> region\_code </label>

<input type="text" name="tc" placeholder= "region\_code" size="15" required />

<label> category </label>

<input type="text" name="tc1" placeholder= "category" size="15" required />

<div>

<button type="submit" class="registerbtn">Predict Number of orders </button>

<br>

<br>

<label>The Predicted Number of orders is:</label>

<input type="text" placeholder= "Press Predict Number of orders" size="15" required, value={{wqi}} >

</form>

</body>

</html>

**FLASK CODE:**

import numpy as np

from flask import Flask, request, render\_template

import pickle

model = pickle.load(open('random.pkl','rb'))

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return render\_template('index.html')

@app.route('/predict',methods=['POST'])

def predict():

float\_features = []

float\_features.append(float(request.form['do']))

float\_features.append(float(request.form['ph']))

float\_features.append(float(request.form['co']))

float\_features.append(float(request.form['bod']))

float\_features.append(float(request.form['na']))

float\_features.append(float(request.form['tc']))

float\_features.append(float(request.form['tc1']))

final\_features = [np.array(float\_features)]

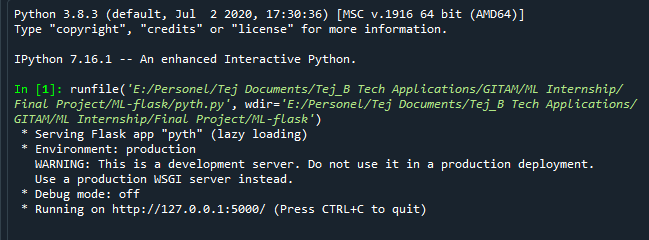
prediction = model.predict(final\_features)

output = np.round\_(prediction[[0]], 2)

return render\_template('index.html', wqi=output[0])

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=False)



**OUTPUT**



