FOOD DEMAND FORECASTING

Data Science With Python Lab Project Report

Bachelor

in

Computer Science

By

KOTYADA SHALINI AND NANDIPATI KUSUMA

S200001

S200317

Rajiv Gandhi University Of Knowledge And Technologies

S.M. Puram , Srikakulam -532410

Andhra Pradesh, India

Abstract

Food Demand Forecasting

Now a days, food delivery market is increasing rapidly. For the success in this food market, the delivery company needs to supply the food ontime to the customers. To make it happen, Food Demand Forecasting is used. We are working on the project of Food Demand Forecasting for a meal delivery company, which has many centers in a city and the outcome of this project is to forecast demand of food for upcoming weeks so that these centers will plan the stock of raw materials accordingly , based on the factors of center id,city-code,region-code,center type,op-area,meal-id,category,cuisine,number of orders etc. We are aiming to get accurate results by using the Machine Learning and Data Science. This dataset is taken from kaggle. Our aim is to create a model that uses our dataset to accurately forecast the demand of food for upcoming weeks. This project can benefit the meal delivery company owner to get the idea of demand of food. In this project,we use some of the python libraries such as pandas,numpy,matplotlib and scikit-learn etc.

Contents

Chapter 1

Introduction

1.1 Introduction to Your Project

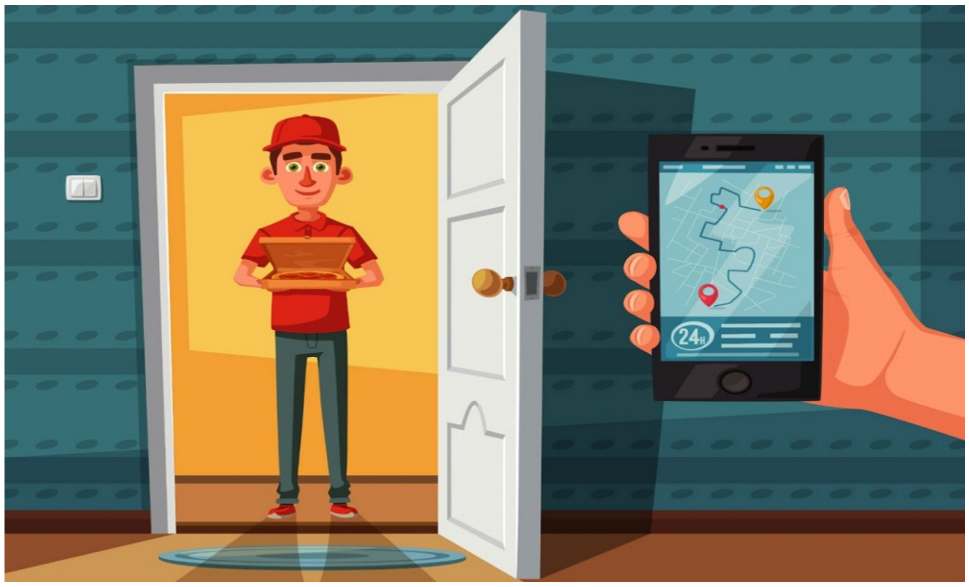


Figure 1.1: Food Delivery

Our project mainly focuses on forecasting food demand for a meal delivery company. They have various fulfillment centers in various cities for dispatching meal orders to their customers. The client wants to help these centers with demand forecasting for upcoming weeks so that these centers will plan the stock of raw materials accordingly. By utilizing various parameters such as meal-id,week,checkout-price,base-price. By employing data pro- 3cessing techniques with pandas and visualizing results or insights through matplotlib and seaborn. We are aim to develop an accurate model.To achieve this,we used some regression techniques linear Regression and Random Forest Regression Techniques.

1.2 Application

Here are the few applications on the project of Food Demand Forecasting.

Food Producers and Manufacturers: They can optimize production schedules, raw material requirement , and resource allocation based on forecasted demand , leading to reduced costs , minimized waste, and improved operational efficiency.

Inventory Management: Accurate demand forecasting enables businesses to maintain optimal inventory levels, reducing carrying costs while ensuring that enough stock is available to meet customer demand. This helps in minimizing wastage.

Menu Planning in Food Service Industry: Restaurants, cafeterias, and catering businesses can use demand forecasting to plan their menus effectively. By predicting which dishes are likely to be popular, they can optimize ingredient purchasing, reduce food waste, and ensure customer satisfaction by offering a diverse and appealing selection.

Production Planning and Resource Allocation: Food manufacturers can use demand forecasts to plan production schedules, allocate resources such as labor and equipment, and optimize production efficiency. This helps in reducing production costs and improving overall operational performance.

1.3 Motivation Towards Your Project

Our motivation towards this project comes from how the restaurants and other food/meprojectal delivery companies managing and supplying food to the customers in an efficient way . Recognizing the demand of food helps both companies and customers. By this,the food supplying process will be easy and smooth. Our primary focus on this was to identify the meal items and their ids and category of food ,which people mostly like. By arranging them in sequential way,we can forecast the estimated demand. The main factors that this project runs on the meal information and information on fulfillment centers. By these factors,we 4can forecast the requirement of raw materials in fulfillment centers.

1.4 Problem Statement

Our Project is Food Demand Forecasting. The project aims to develop a machine learning model that can predict the demand of food for a meal delivery company. The dataset for this project is taken from Kaggle. This project will utilize various features for predicting the demand of food of the company such as meal information ,price,cuisine etc. By analyzing these features,the model should predict accurate food demand using better machine learning model.The dataset is split into training set and test set for model development.

Chapter 2

Approach To Your Project

2.1 Explain About Your Project

Our project, Food Demand Forecasting aims to forecast the demand of food for a meal delivery company. It involves predicting the demand based on number of orders, centre-id, category of food and cuisines. Our goal is to Predict the number of orders for respective meal-id.Machine learning models are then trained on this data to predict future food demand. This predictions can help clients to analyze the food demand and take the appropriate measures to plan the availability of raw materials.

2.2 Data Set

The Dataset ”Food Demand Forecasting” is taken from Kaggle Website. This Dataset contains some CSV files that consists of information about meals, fulfillment centers, number-of-orders. This dataset includes information such as the types of food items, and their classification based on cuisines.The purpose of this dataset is to make analysis and modeling to predict future food demand accurately.

2.3 Prediction technique

1. Data Collection and Preparation

Data Cleaning: Handle missing values, remove duplicates, and correct inconsistencies in the data. Feature Engineering: Create new features that could help improve the model's accuracy, rolling means, and categorical features .

2. Exploratory Data Analysis (EDA)

Statistical Summary: Use functions like describe() to get a summary of the data. Data Visualization: Plot time series graphs, histograms, and boxplots to understand the distribution and trends in the data. Correlation Analysis: Calculate correlation coefficients to identify relationships between different variables.

3. Machine Learning Models

Linear Regression:

A basic approach to model the relationship between the target variable and one or more predictors. Linear Regression works efficiently for the columns having linear relationships.

Decision Trees:

Model that splits data into subsets based on feature values, useful for capturing non-linear relationships. A DecisionTree regression model is created and fitted to the training data that predicts the number of orders for test data.Here we Create and Configure the Decision Tree Regressor with appropriate maximum depth and other required parameters.

Random Forest:

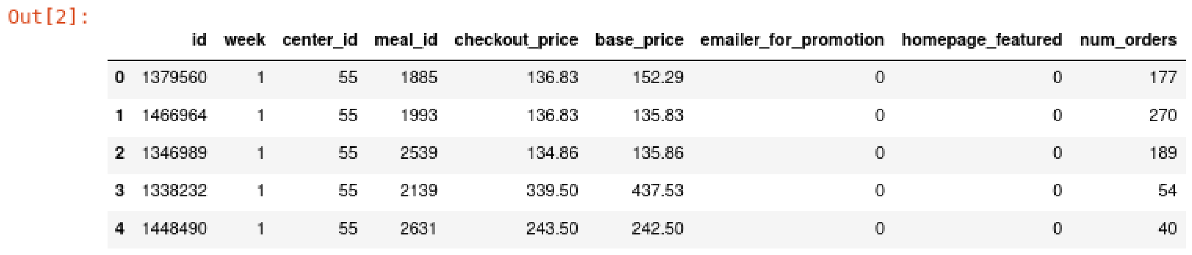
An ensemble method that uses multiple decision trees to improve prediction accuracy and prevent overfitting. A Random Forest regression model is created and fitted to the training data that predicts the number of orders for test data.Here we Create and Configure the Random Forest Regressor with appropriate maximum depth and other required parameters.

2.4 Exploratory Data Analysis (EDA)

Exploratory Data Analysis is the process of analyzing and summarizing datasets to gain insights and understand their main characteristics before applying any machine learning or statistical models.Here's how these functions fit into EDA:

1. head()

```
import pandas as pd

meal=pd.read_csv('meal_info.csv');

train=pd.read_csv('train.csv');

test=pd.read_csv('test.csv');

fc_df=pd.read_csv('fulfilment_center_info.csv')

sub=pd.read_csv('sample_submission.csv')

train.head()
```



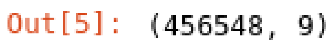
Out[2]:

	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

Figure 2.1: head()

2. shape()

```
train.shape
```

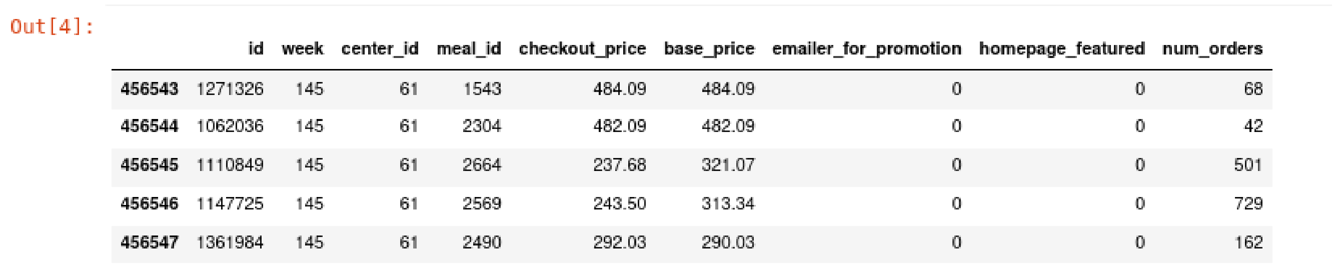


```
Out[5]: (456548, 9)
```

Figure 2.2: shape

3. tail()

```
train.tail()
```



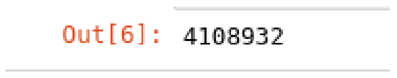
Out[4]:

	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders
456543	1271326	145	61	1543	484.09	484.09	0	0	68
456544	1062036	145	61	2304	482.09	482.09	0	0	42
456545	1110849	145	61	2664	237.68	321.07	0	0	501
456546	1147725	145	61	2569	243.50	313.34	0	0	729
456547	1361984	145	61	2490	292.03	290.03	0	0	162

Figure 2.3: tail()

4. size

```
train.size
```

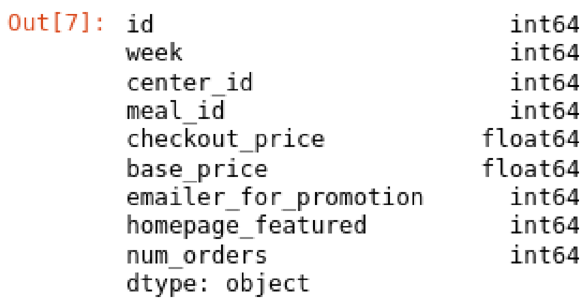


```
Out[6]: 4108932
```

Figure 2.4: size

5. dtypes

```
train.dtypes
```

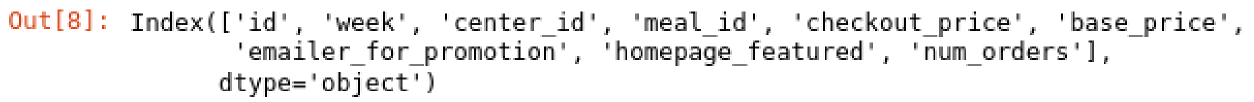


```
Out[7]: id                         int64
        week                       int64
        center_id                  int64
        meal_id                    int64
        checkout_price           float64
        base_price               float64
        emailer_for_promotion      int64
        homepage_featured          int64
        num_orders                 int64
        dtype: object
```

Figure 2.5: dtypes

6. columns

```
train.columns
```

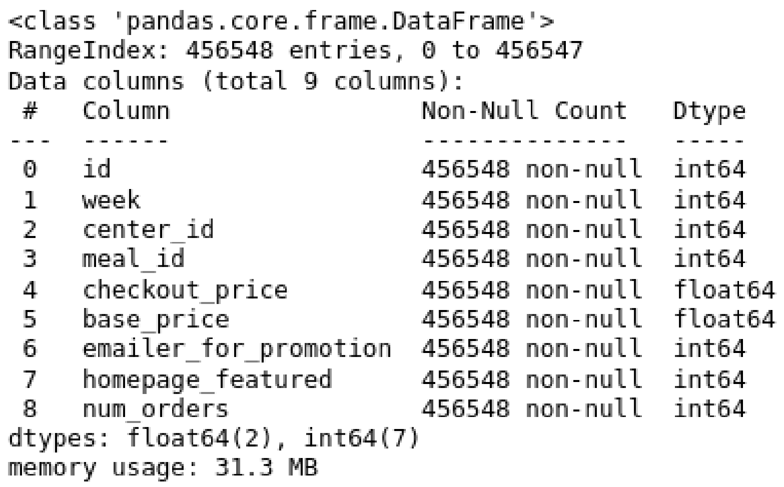


```
Out[8]: Index(['id', 'week', 'center_id', 'meal_id', 'checkout_price', 'base_price',
               'emailer_for_promotion', 'homepage_featured', 'num_orders'],
              dtype='object')
```

Figure 2.6: columns

7. info()

```
train.info()
```

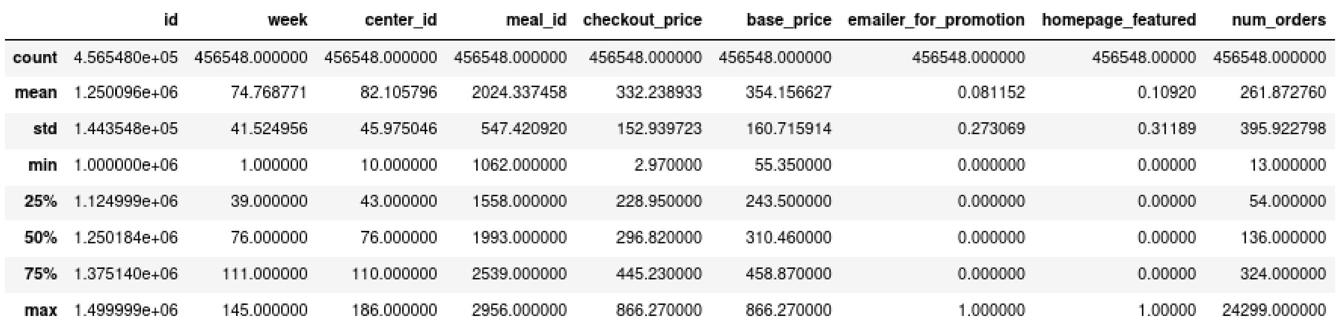


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 456548 entries, 0 to 456547
Data columns (total 9 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   id                     456548 non-null  int64
 1   week                   456548 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
```

Figure 2.7: info()

8. describe()

```
train.describe()
```

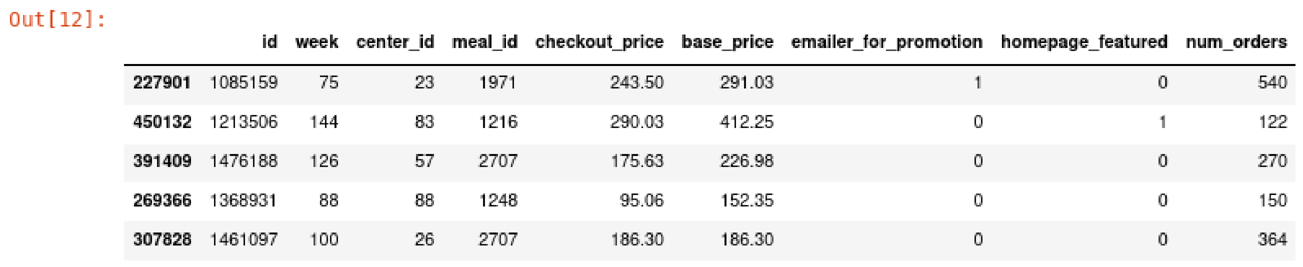


	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders
count	4.565480e+05	456548.000000	456548.000000	456548.000000	456548.000000	456548.000000	456548.000000	456548.00000	456548.000000
mean	1.250096e+06	74.768771	82.105796	2024.337458	332.238933	354.156627	0.081152	0.10920	261.872760
std	1.443548e+05	41.524956	45.975046	547.420920	152.939723	160.715914	0.273069	0.31189	395.922798
min	1.000000e+06	1.000000	10.000000	1062.000000	2.970000	55.350000	0.000000	0.00000	13.000000
25%	1.124999e+06	39.000000	43.000000	1558.000000	228.950000	243.500000	0.000000	0.00000	54.000000
50%	1.250184e+06	76.000000	76.000000	1993.000000	296.820000	310.460000	0.000000	0.00000	136.000000
75%	1.375140e+06	111.000000	110.000000	2539.000000	445.230000	458.870000	0.000000	0.00000	324.000000
max	1.499999e+06	145.000000	186.000000	2956.000000	866.270000	866.270000	1.000000	1.00000	24299.000000

Figure 2.8: describe()

9. sample()

```
train.sample(5)
```



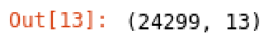
Out[12]:

	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders
227901	1085159	75	23	1971	243.50	291.03	1	0	540
450132	1213506	144	83	1216	290.03	412.25	0	1	122
391409	1476188	126	57	2707	175.63	226.98	0	0	270
269366	1368931	88	88	1248	95.06	152.35	0	0	150
307828	1461097	100	26	2707	186.30	186.30	0	0	364

Figure 2.9: sample()

10. max(),min()

```
train['num_orders'].max(),train['num_orders'].min()
```

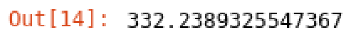


```
Out[13]: (24299, 13)
```

Figure 2.10: min(),max()

11. mean()

```
train['checkout_price'].mean()
```

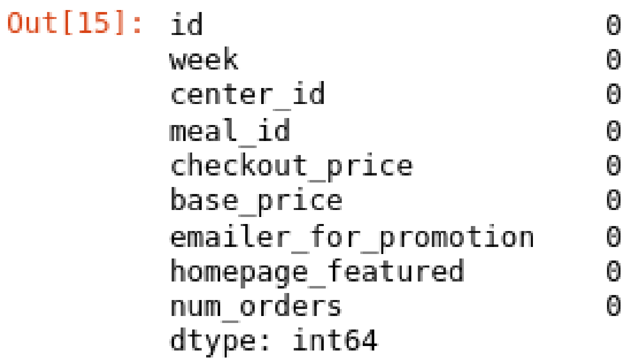


```
Out[14]: 332.2389325547367
```

Figure 2.11: mean()

12. isnull()

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

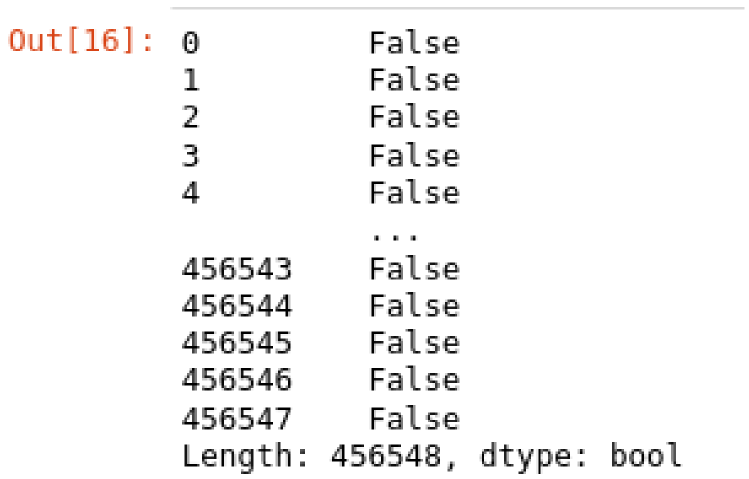


```
Out[15]: id                      0
         week                    0
         center_id               0
         meal_id                 0
         checkout_price          0
         base_price              0
         emailer_for_promotion   0
         homepage_featured       0
         num_orders              0
         dtype: int64
```

Figure 2.12: isnull()

13. duplicated()

```
train.duplicated()
```

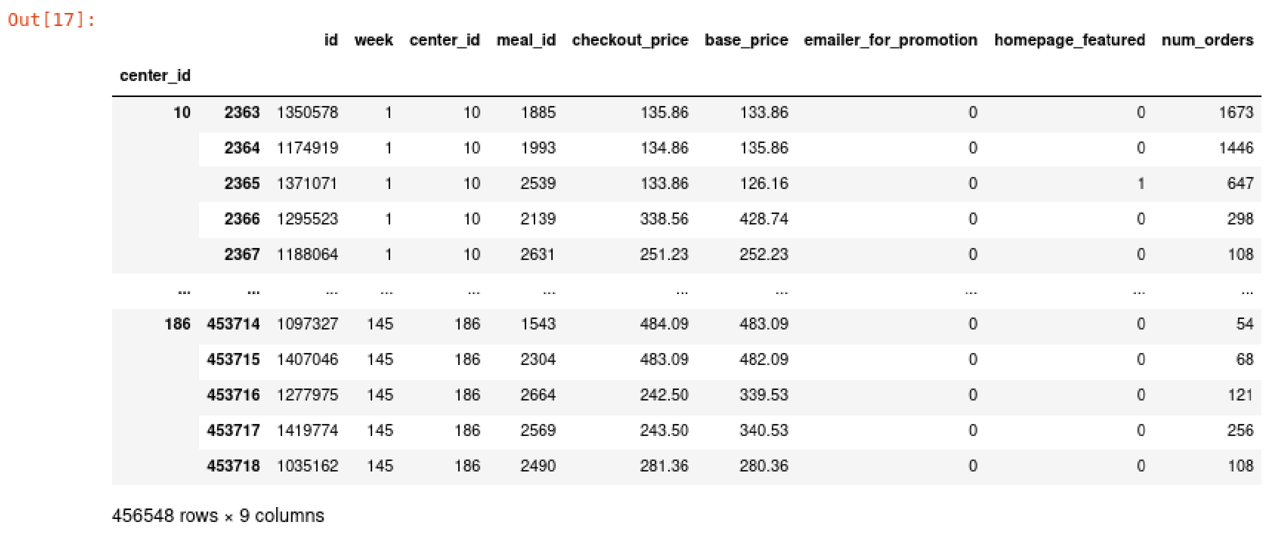


```
Out[16]: 0          False
         1          False
         2          False
         3          False
         4          False
                     ...
         456543     False
         456544     False
         456545     False
         456546     False
         456547     False
         Length: 456548, dtype: bool
```

Figure 2.13: duplicated()

14. groupby()

```
train.groupby(['center_id']).apply(lambda x:x)
```


Out[17]:

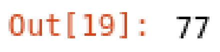
center_id		id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders
10	2363	1350578	1	10	1885	135.86	133.86	0	0	1673
	2364	1174919	1	10	1993	134.86	135.86	0	0	1446
	2365	1371071	1	10	2539	133.86	126.16	0	1	647
	2366	1295523	1	10	2139	338.56	428.74	0	0	298
	2367	1188064	1	10	2631	251.23	252.23	0	0	108
...	...	...	...	...	...	...	...	...	...	...
186	453714	1097327	145	186	1543	484.09	483.09	0	0	54
	453715	1407046	145	186	2304	483.09	482.09	0	0	68
	453716	1277975	145	186	2664	242.50	339.53	0	0	121
	453717	1419774	145	186	2569	243.50	340.53	0	0	256
	453718	1035162	145	186	2490	281.36	280.36	0	0	108

456548 rows × 9 columns

Figure 2.14: groupby()

15. nunique()

```
train['center_id'].nunique()
```

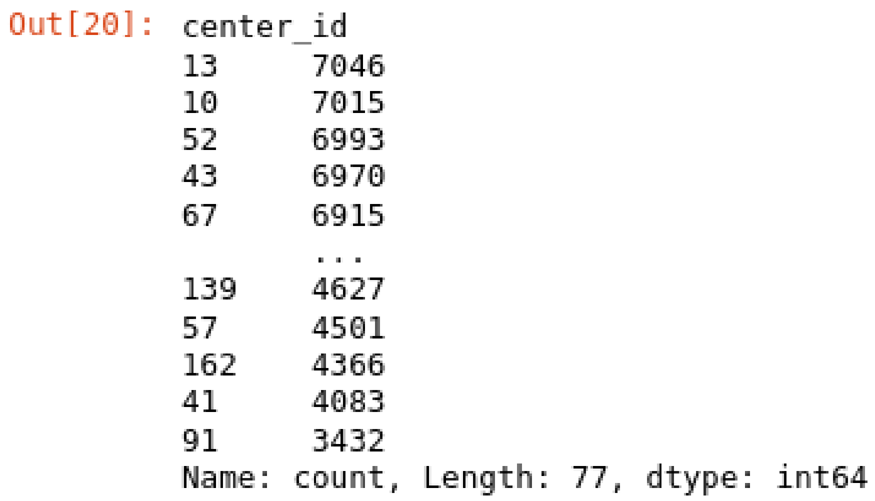


```
Out[19]: 77
```

Figure 2.15: nunique()

16. value_counts()

```
train['center_id'].value_counts()
```



```
Out[20]: center_id
         13     7046
         10     7015
         52     6993
         43     6970
         67     6915
                ...
         139    4627
         57     4501
         162    4366
         41     4083
         91     3432
         Name: count, Length: 77, dtype: int64
```

Figure 2.16: value_counts()

2.5 Graphs

To know about the relationship between the input columns and to distribution of certain columns, we use graphs and Visualization techniques. This makes us clearly visualize the factors and study their effect on the required output.

1.BAR GRAPH

Bargraph helps us to visualize the distribution of a category.It is simple and easier to use bargraph. To know the distribution of orders per each category, we can analyse the demand of each category we use the bargraph.
We use Category column for this graph.
It is shown below.

CODE

```
import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

categories=data['category']

cat_counts={}
```

```
for category in categories:
    cat_counts[category]=cat_counts.get(category,0)+1
category_names=list(cat_counts.keys())
category_values=list(cat_counts.values())
# bar plot
plt.figure(figsize=(8,6))
plt.bar(category_names,category_values,color=["#12436D","#28A197","#801650"
,"#F46A25","#3D3D3D","#A28501" ])
plt.title("Category-wise Distribution of Orders")
plt.xticks(rotation=90)
plt.show()
```

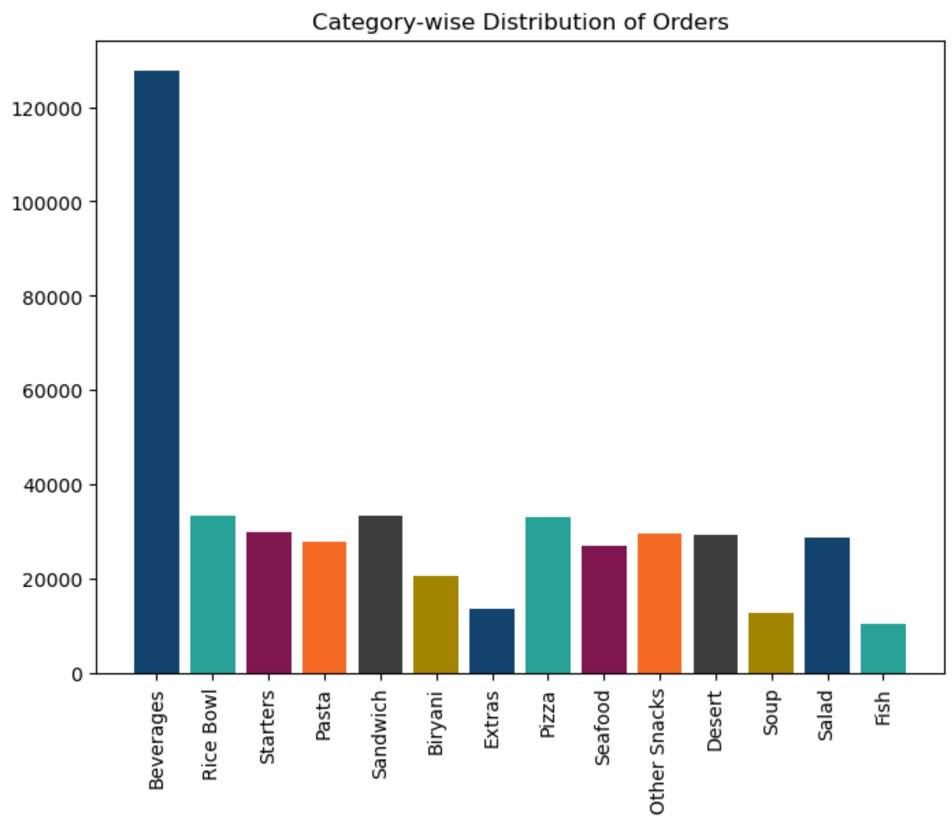


Figure 2.17: Category-wise Distribution of Orders

2.BARH GRAPH

To know the number of orders for each centre based on center id , so that we can analyze and predict the demand of each center individually. Center id's are marked on y-axis and x-axis holds the number of orders as per the scale. This horizontal bargraph is more convinent to view more center id's and their orders.We use Centerid and numorders columns for this horizontal bargraph.

CODE

```
# grouping based on center_id

orders=data.groupby('center_id')['num_orders'].sum()

#plotting

plt.figure(figsize=(10, 12))

orders.plot(kind='barh', color='skyblue')

#setting x-label

plt.xlabel('center_id')

#setting y-label

plt.ylabel('num_orders')

plt.xlim(0,4)

plt.legend()

#using tight_layout for better visualization

plt.tight_layout()

#showing the plot

plt.show()
```

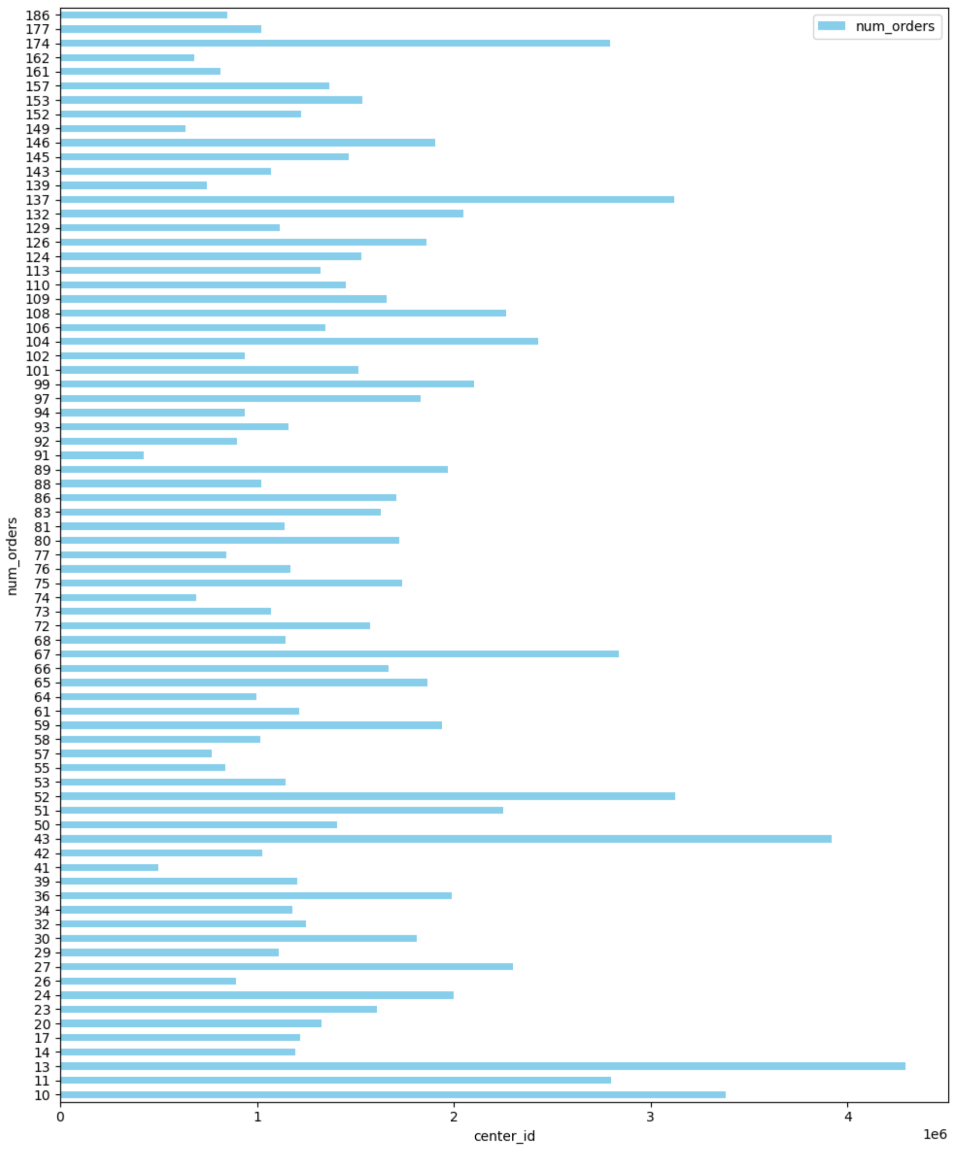


Figure 2.18: Distribution of orders for each center-id

3.GROUPED BARGRAPH

We have different cuisines like Indian,Italian,Thai and Continental in the dataset. To know about which cuisine has highest demand in each category we use the grouped bargraph as shown.

CODE

```
m_cuisine=data.groupby(['category','cuisine']).size().unstack(fill_value=0)

fig, ax = plt.subplots(figsize=(12, 11))

m_cuisine.plot(kind="bar",ax=ax,width=1)
```

```
plt.xlabel('category')

plt.ylabel('count')

plt.title('Count of cuisines in each category')

plt.show()
```

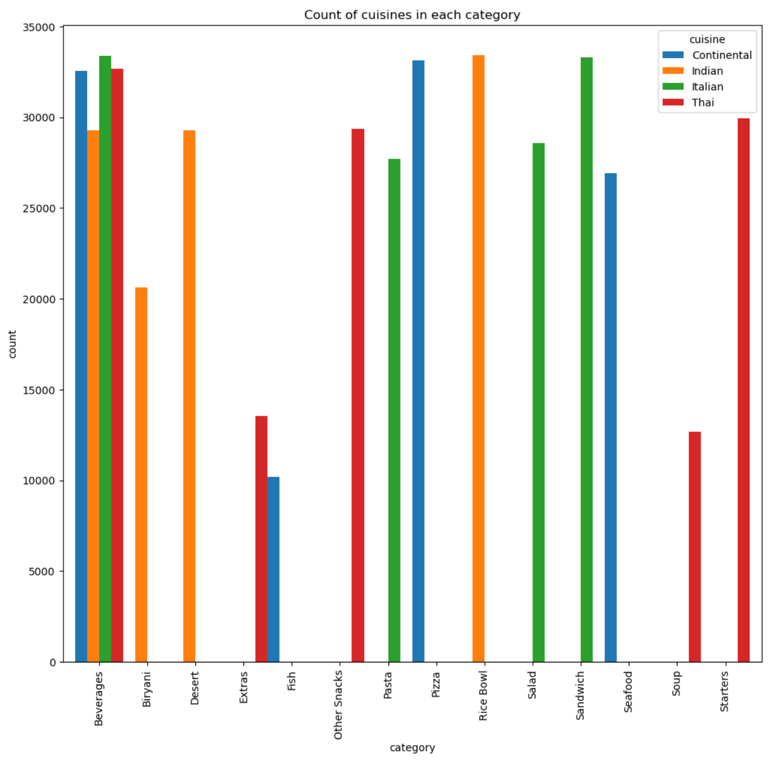


Figure 2.19: Count of cuisines in each category

2.6 Visualization

We use the following visualization techniques to understand the relationships better.We plotted the appropriate and relatable plots on the columns of the dataset ”Food Demand Forecasting”.

These plots makes our task of knowing the effecting factors of the output easier.Thus we can build an accurate model accordingly.

The plotted visulaizations are shown below.

4. PIE CHART

Pie chart helps us to depict the distributions. To know the percentage distribution of different categories we use the below piechart.

We use the column 'category' from the dataset Food Demand Forecasting.

The pie chart is as shown below.

CODE

```
# counting the values for each category

category_counts=data['category'].value_counts()

print(category_counts)

#listing labels

labels=category_counts.index.tolist()

# listing sizes

sizes=category_counts.values.tolist()

#plotting

plt.figure(figsize=(8,8))

# pie

plt.pie(sizes,labels=labels,autopct='%1.1f%%',startangle=40)

plt.title("Distribution of categories")

# showing the plot

plt.show()
```

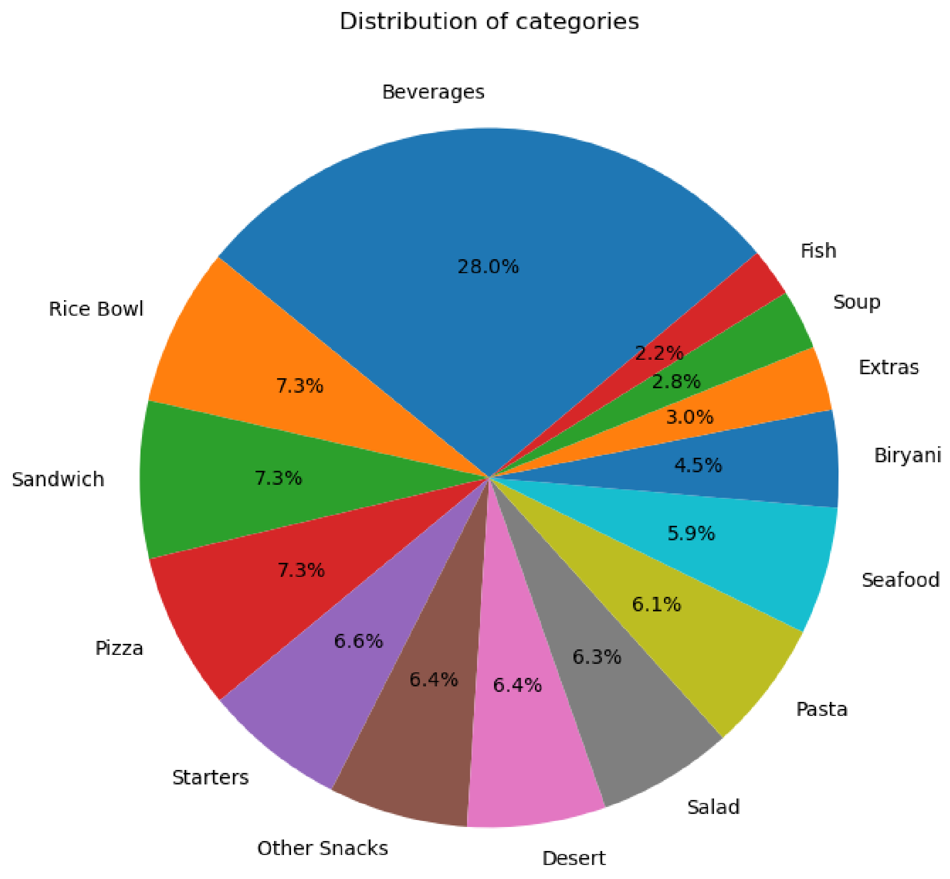


Figure 2.20: Category distribution piechart

5. PIE CHART

Cusinies are the other category in the dataset.To know the percentage distribution of different cusinies, we use the below piechart . We use the column 'cuisine' of the dataset Food Demand Forecasting.

CODE

```
# counting values of each cuisine

cuisine_counts=data['cuisine'].value_counts()

print(cuisine_counts)

labels=cuisine_counts.index.tolist()

sizes=cuisine_counts.values.tolist()

#plotting
```

```
plt.figure(figsize=(8,8))

plt.legend()

plt.pie(sizes,labels=labels,autopct='%1.1f%%',startangle=40)

plt.title("Distribution of cuisines")

plt.show()
```

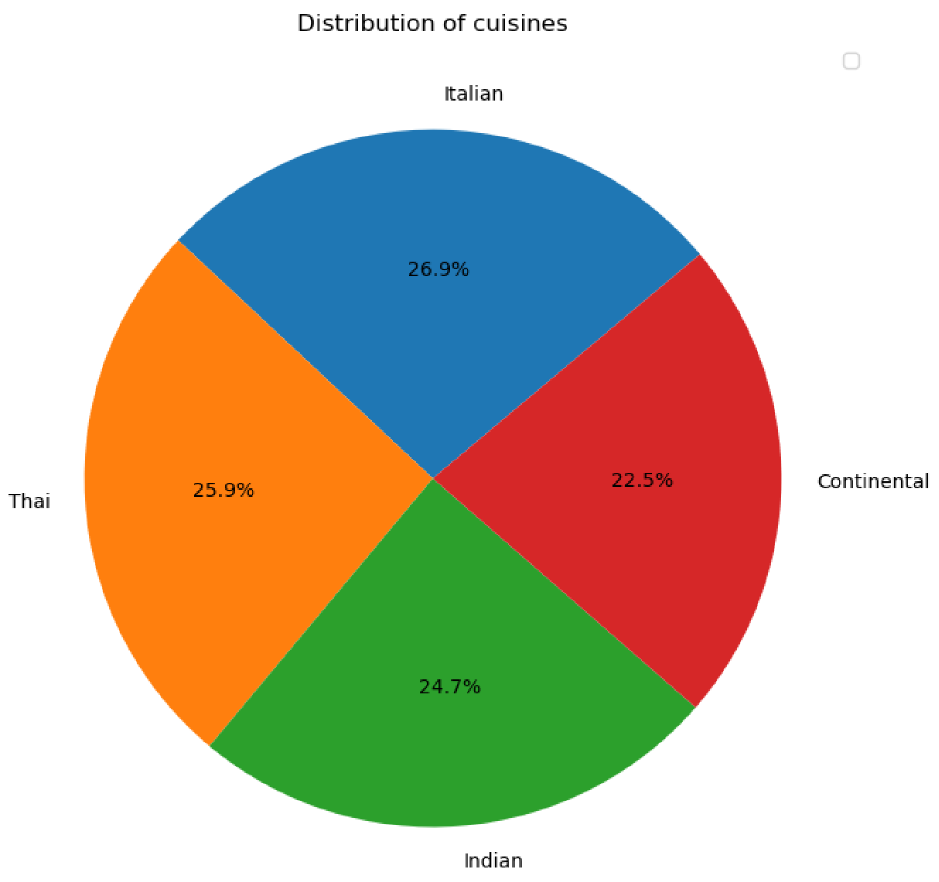


Figure 2.21: Cuisine distribution piechart

6. SCATTER PLOT

The below Scatter plot depicts relation between checkout price and number of orders. Using this we will be able to know how checkout price influence number of orders

CODE

```
x=data['checkout_price']

y=data['num_orders']
```

```
plt.scatter(x,y,color="brown")

plt.xlabel('checkout_price')

plt.ylabel('num_orders')

plt.show()
```

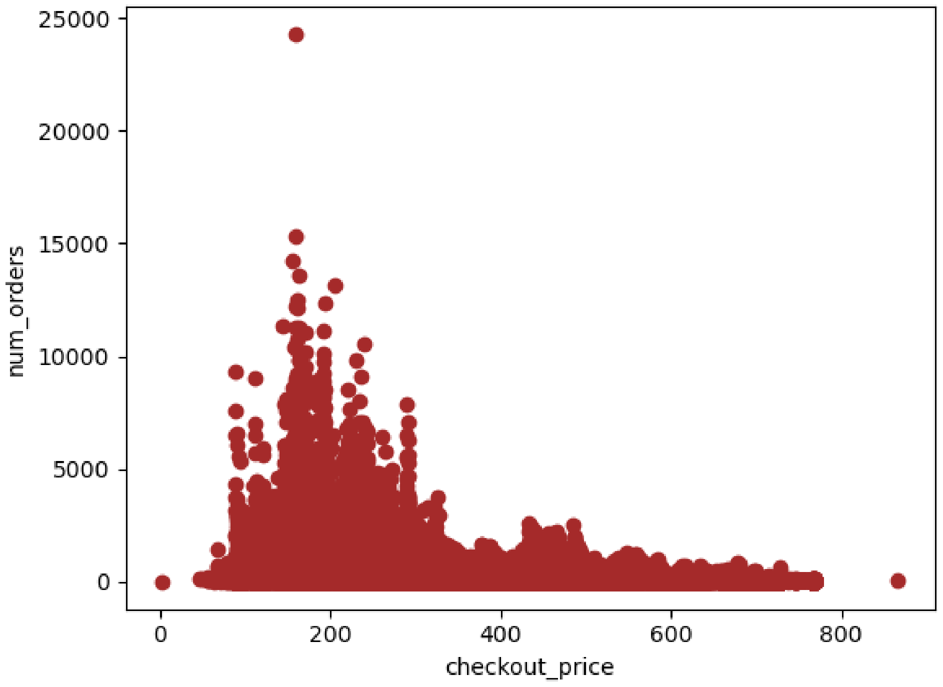


Figure 2.22: Relationship between checkout_price and num_orders

7. LINE PLOT

The below lineplot depicts 'Trend of orders in different weeks', so that we may know how the trend of orders would be in coming weeks. We used 'week' and 'num orders' columns of the our dataset.

CODE

```
# grouping data by week

from matplotlib import ticker

week_orders=data.groupby('week')['num_orders'].mean()

# plotting
```

```
plt.figure(figsize=(12,6))

plt.plot(week_orders.index,week_orders,color="green")

plt.xlabel('week')

plt.ylabel('Number of Orders')

plt.title("Trend of Average Orders per week")

tick_locator=ticker.MultipleLocator(base=10)

plt.gca().xaxis.set_major_locator(tick_locator)

plt.tight_layout()

plt.show()
```

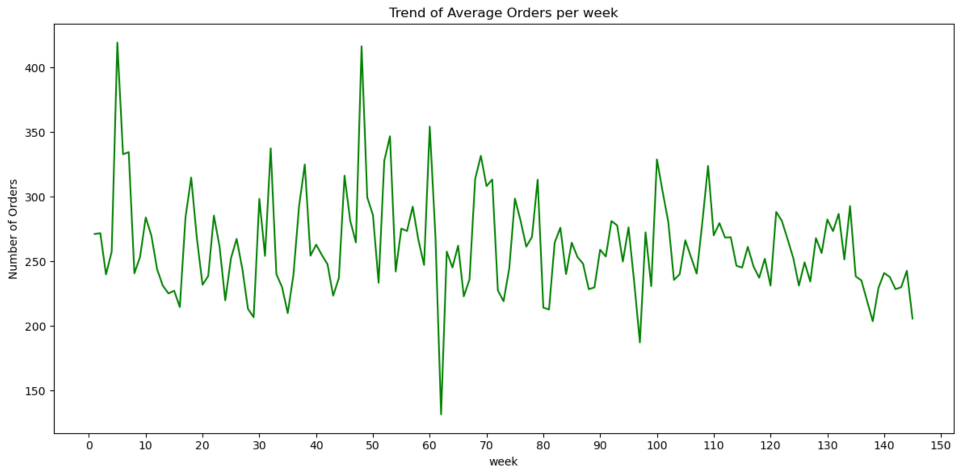


Figure 2.23: Trend of Average Orders per week

8. BOXPLOT

Boxplot helps us to understand the distribution in a detailed way. It additionally provides us the median and we can also check for the Outliers of our data. We used 'checkout price' and 'num orders' columns of the our dataset 'Food Demand Forecasting'.
Using this boxplot, we can analyse at which range of checkout prices, maximum sales are concentrated for each category. We can figure out Outliers in checkout prices.

CODE

```
plt.figure(figsize=(20,10))
```

```
data.boxplot('checkout_price','category',color="")

plt.xlabel('category')

plt.ylabel('checkout_price')

plt.xticks(rotation=90)

plt.show()
```

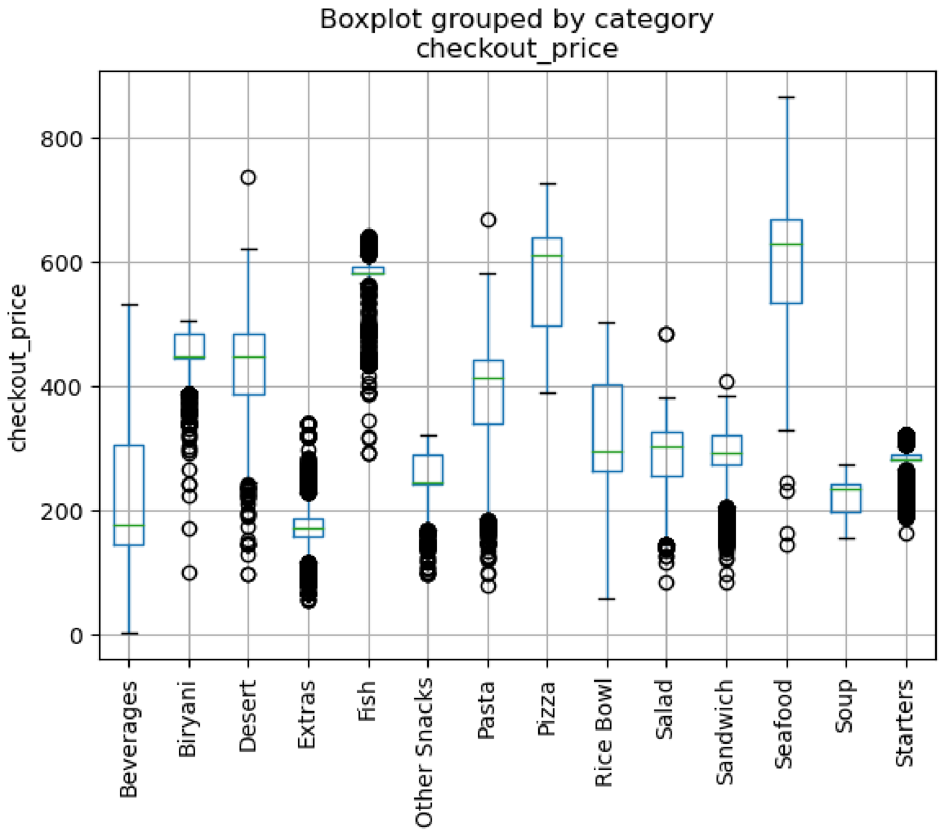


Figure 2.24: Trend of orders based on checkout_price

9. HISTOGRAM

Histograms provide a visual representation of the frequency distribution of an attribute, showing how often each value or range of values occurs. To know the 'Distribution of number of orders', we plotted the histogram shown below. We used 'num orders' column of the our dataset 'Food Demand Forecasting' to plot the histogram.
Using this histogram, we can get the distribution of number of orders.

CODE

```
# histogram for num_orders
num_orders = data['num_orders']
plt.figure(figsize=(10,6))
plt.hist(num_orders, bins=100,color="lightcoral")  # Adjust the number
of bins as needed    , edgecolor='black'
plt.xlabel('Number of Orders')
plt.xlim(0,3000)
# plt.title("Trend of Average Orders per week")
plt.legend()
plt.ylabel('Frequency')
plt.title('Distribution of Number of Orders')
plt.show()
```

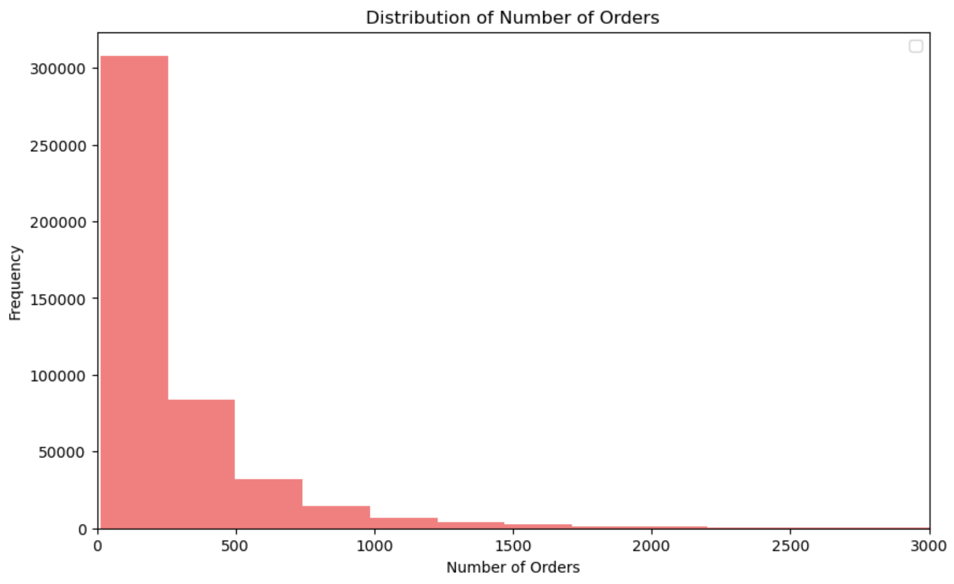


Figure 2.25: Trend of orders in different weeks

10. LINE PLOT

Using the below lineplot we can analyze, how promotion effects on sales that is number of orders. We used the columns ”emailer for promotion” and ”num orders” from the dataset.

CODE

```
promoapplied=[1]
```

```
promotionsapplieddf=train.loc[train['emailer_for_promotion'].isin(promoapplied)]

df = promotionsapplieddf

fig = px.line(df, x="week", y="num_orders", title='Promotions effect on
sales')

fig.show()
```

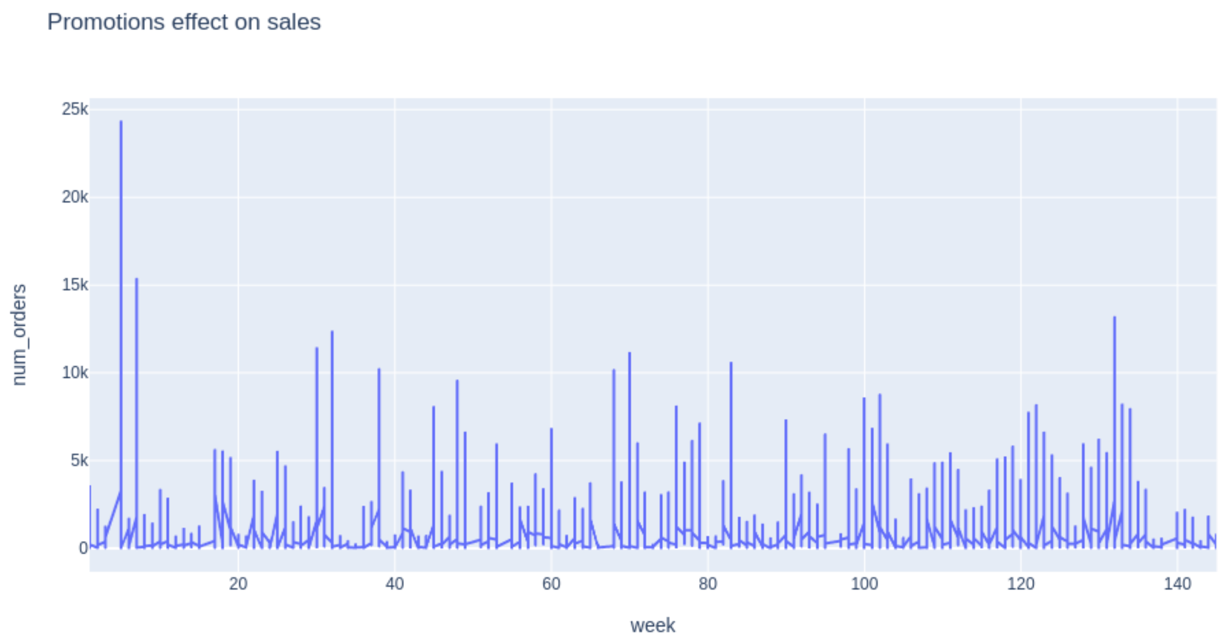


Figure 2.26: Promotions effect on sales

11. LINE PLOT

Using the below lineplot we can analyze,how homepage features effects on sales that is number of orders. We used the columns ”homepage features” and ”num orders” from the dataset.

CODE

```
# homepage features

homepagepushad=[1]

homepagepushaddf=train.loc[train['homepage_featured'].isin(homepagepushad)]

df = homepagepushaddf
```

```
fig = px.line(df, x="week", y="num_orders", title='Homepage push ad meal
sales')
# showing plot
fig.show()
```

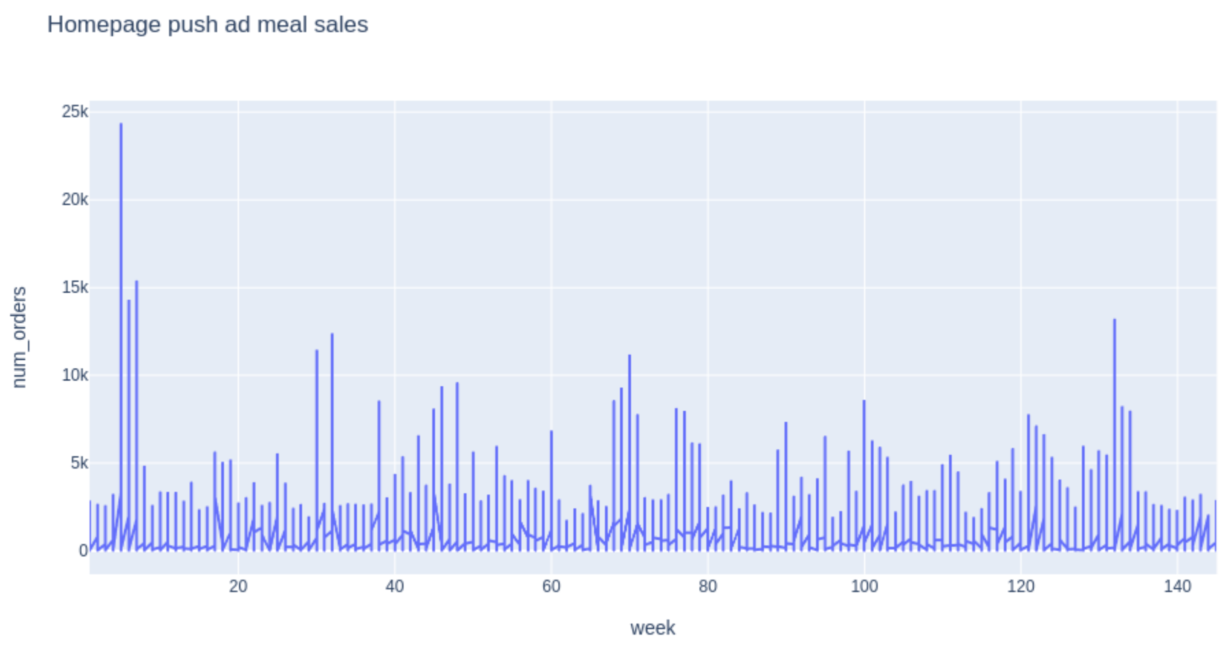


Figure 2.27: Homepage push ad meal sales

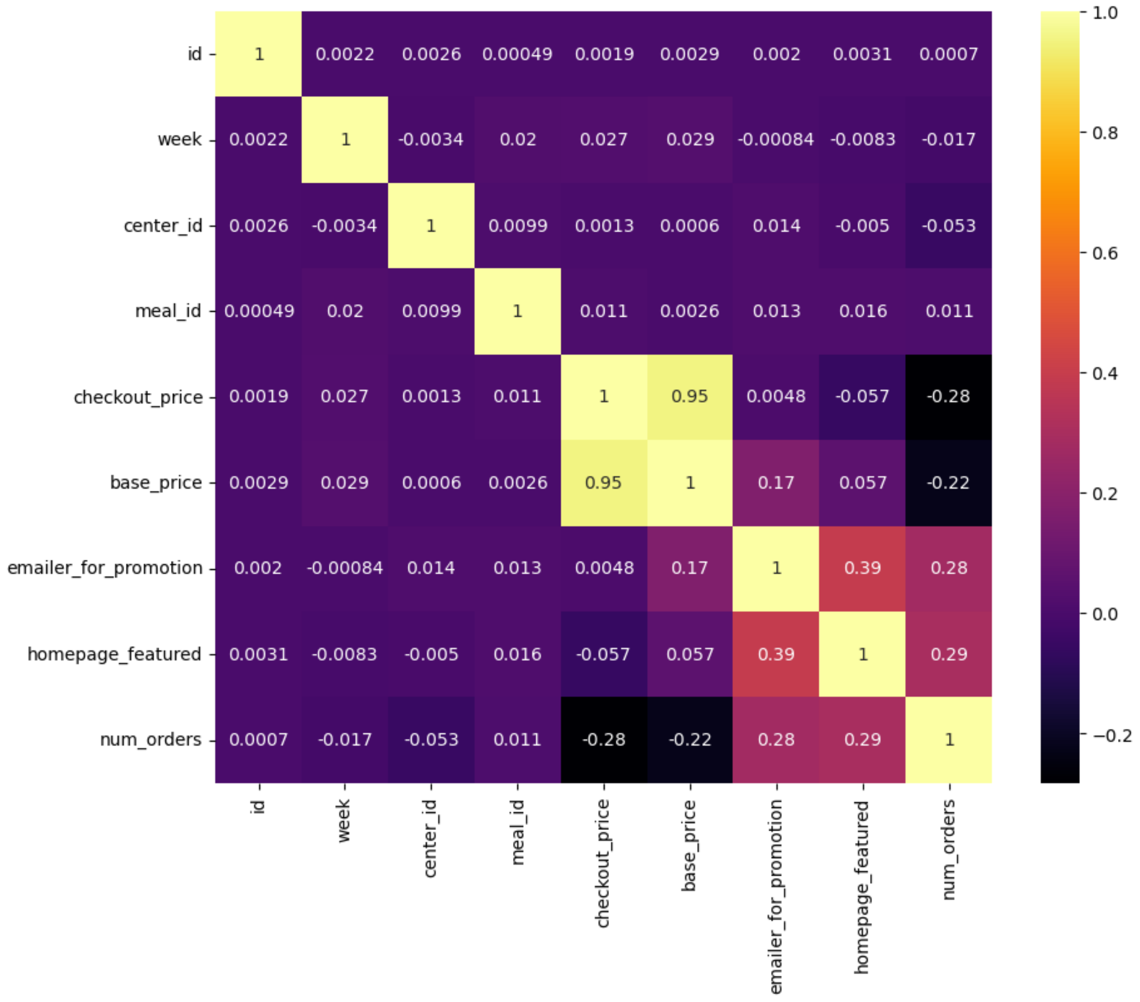
12. HEATMAP

Heatmaps are powerful data visualization tools that can convey a wealth of information through the use of color to represent data values in a matrix or table.
Helps us visualizing the correlation between different variables in a dataset.In the dataset 'Food Demand Forecaasting' containing various indicators, a heatmap can help identify which indicators are strongly correlated as shown.

CODE

```
import seaborn as sns
# correlation matrix
co_mat=train.corr()
```

```
plt.figure(figsize=(10, 8))

sns.heatmap(co_mat, annot=True, cmap='inferno')

# id and week has less correlation
```



	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders
id	1	0.0022	0.0026	0.00049	0.0019	0.0029	0.002	0.0031	0.0007
week	0.0022	1	-0.0034	0.02	0.027	0.029	-0.00084	-0.0083	-0.017
center_id	0.0026	-0.0034	1	0.0099	0.0013	0.0006	0.014	-0.005	-0.053
meal_id	0.00049	0.02	0.0099	1	0.011	0.0026	0.013	0.016	0.011
checkout_price	0.0019	0.027	0.0013	0.011	1	0.95	0.0048	-0.057	-0.28
base_price	0.0029	0.029	0.0006	0.0026	0.95	1	0.17	0.057	-0.22
emailer_for_promotion	0.002	-0.00084	0.014	0.013	0.0048	0.17	1	0.39	0.28
homepage_featured	0.0031	-0.0083	-0.005	0.016	-0.057	0.057	0.39	1	0.29
num_orders	0.0007	-0.017	-0.053	0.011	-0.28	-0.22	0.28	0.29	1

Figure 2.28: Heatmap of correlation matrix

Chapter 3

Code

3.1 Explain Your Code With Outputs

Forecasting 'Food Demand' is a critical task for businesses in the food industry, including restaurants and food manufacturers. In this context, regression models are powerful tools that can predict future food demand based on historical data and various influencing factors.Regression models are statistical techniques used to understand the relationship between a dependent variable (in this case, food demand) and one or more independent variables (such as checkout price,base price,category,week etc).

We are using the following Machine Learning algorithms to fit the best model.

1.Linear Regression

2.Decision Tree Regression

3.Random Forest Regression

4.Ada Boost Regression

3.2 Feature allocation

```
X=traind.drop(["num_orders"],axis=1)

y=traind["num_orders"]
```

3.3 Splitting the dataset into train and test sets

```python
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test= train_test_split(X, y, test_size= 0.25,

random_state=0)
```

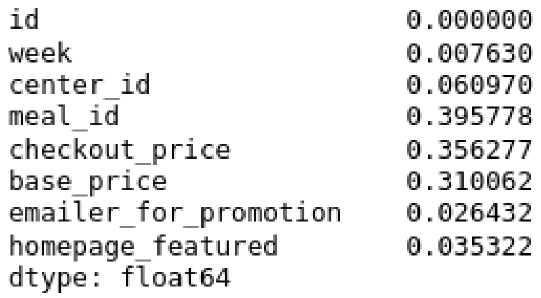
MUTUAL INFO REGRESSION:

The mutual_info_regression function from scikit-learn is used to estimate mutual information for a regression problem. This function helps in identifying the dependency between each feature and the target variable. High mutual information scores indicate that the feature is informative about the target variable.Better practice to do on train and test split data

CODE

```python
from sklearn.feature_selection import mutual_info_regression

# Mutual information

mutual_info = mutual_info_regression(x_train,y_train)

mutual_info_series = pd.Series(mutual_info, index=X.columns)

print(mutual_info_series)
```

The output is shown below



```
id                      0.000000
week                    0.007630
center_id               0.060970
meal_id                 0.395778
checkout_price          0.356277
base_price              0.310062
emailer_for_promotion   0.026432
homepage_featured       0.035322
dtype: float64
```

Figure 3.1: mutual_info_regression

```python
mutual_info_series.plot(kind='bar')
```

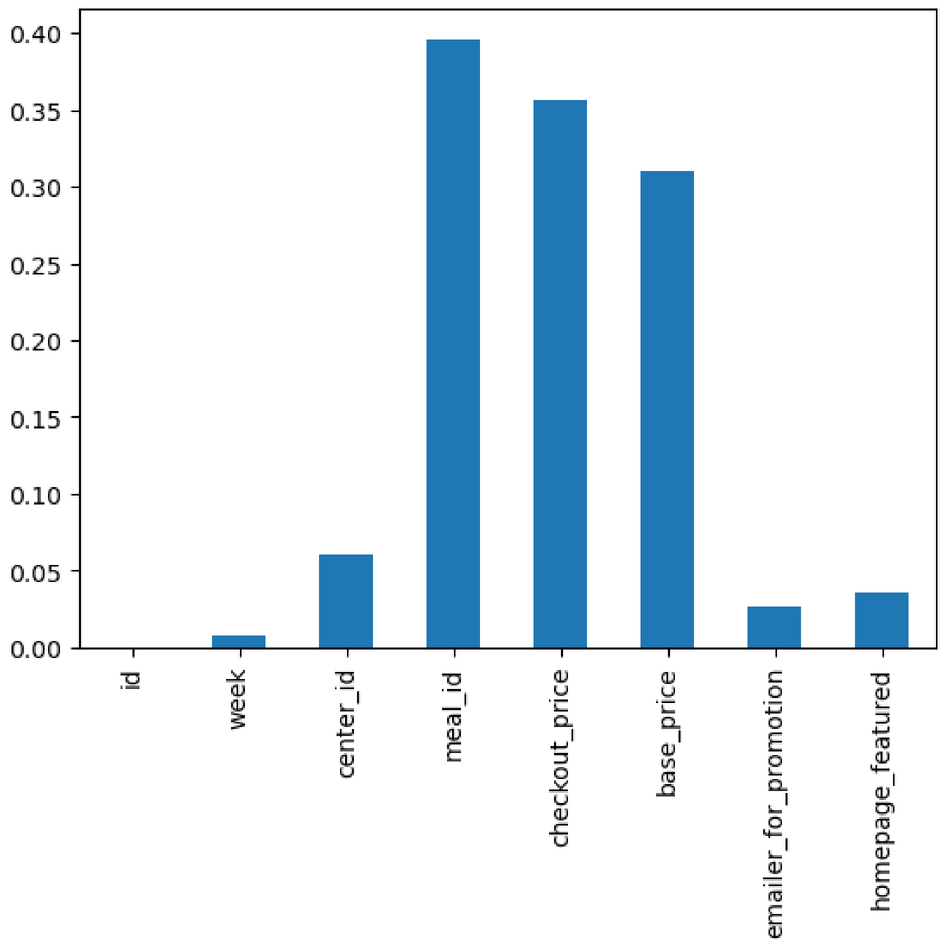


Figure 3.2: mutual_info_regression

We can remove the low importance values because it doesn't have any importance in model building and it contributes nothing to the prediction of demand(i.e number of orders).Here we can remove 'id' and 'week' columns.

```
X1=train.drop(['num_orders','id','week'],axis=1)

y1=train['num_orders']
```

3.3.1 Splitting the new dataset

```
from sklearn.model_selection import train_test_split

x1_train,x1_test,y1_train,y1_test=train_test_split(X1,y1,test_size=0.25,random_state=4
```

3.4 Buliding Model

Since our food demand forecasting project comes under regression tasks,we are importing different regression techniques to perform the regression task and determines the best model which prediction is accurate by doing Evaluation Metrics.

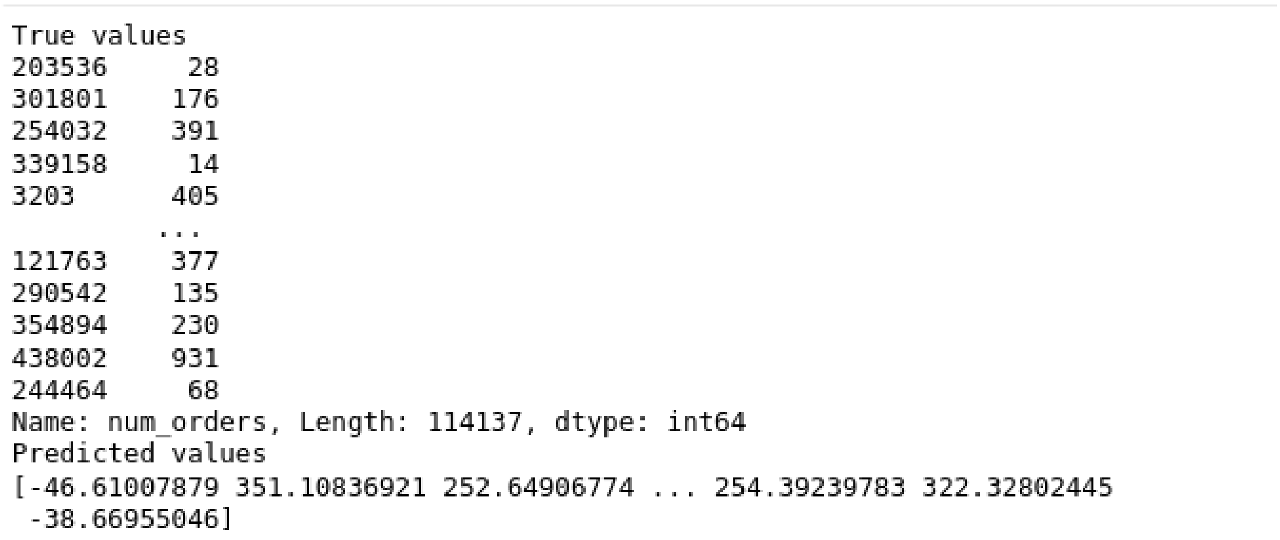
3.4.1 Linear Regression

```
from sklearn.linear_model import LinearRegression

lr_model=LinearRegression()

#fit

lr_model.fit(x1_train,y1_train)
```

```
▾ LinearRegression
LinearRegression()
```

Figure 3.3: LinearRegression

```
# Making predictions

lr_pred = lr_model.predict(x1_test)

# true values

print("True values")

print(y1_test)

# predicted values

print("Predicted values")

print(lr_pred)
```

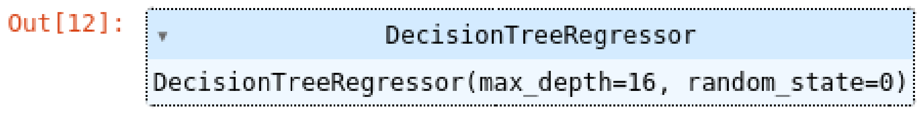


```
True values
203536     28
301801    176
254032    391
339158     14
3203      405
         ...
121763    377
290542    135
354894    230
438002    931
244464     68
Name: num_orders, Length: 114137, dtype: int64
Predicted values
[-46.61007879 351.10836921 252.64906774 ... 254.39239783 322.32802445
 -38.66955046]
```

Figure 3.4: LinearRegression

3.4.2 Decision Tree Regression

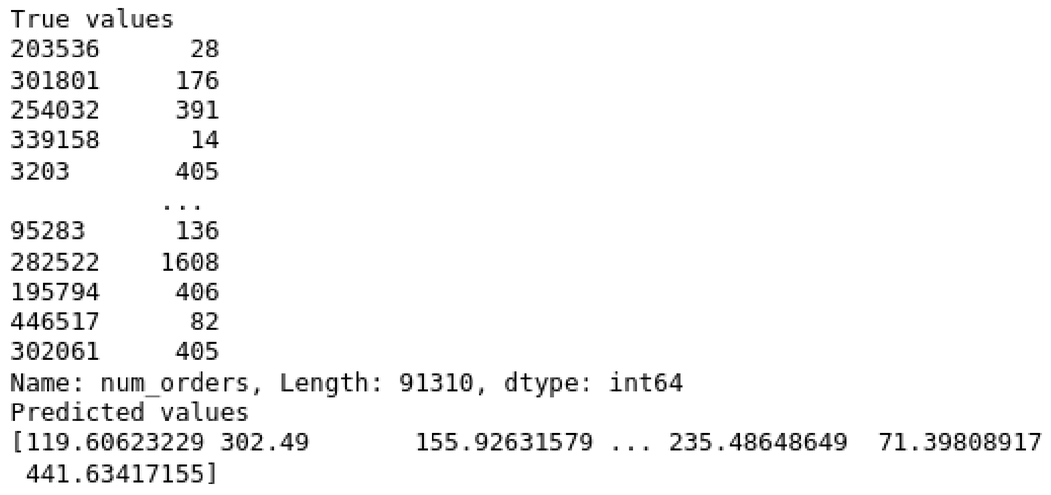
```
from sklearn.tree import DecisionTreeRegressor

X2_train,X2_test,y2_train,y2_test = train_test_split(X1,y1,test_size=0.2,

random_state=42)

dtr_model = DecisionTreeRegressor(max_depth=16,random_state=0)

dtr_model.fit(X2_train,y2_train)

dtr_pred = dtr_model.predict(X2_test)
```



```
Out[12]:  DecisionTreeRegressor
          DecisionTreeRegressor(max_depth=16, random_state=0)
```

Figure 3.5: Decision Tree Regression

```
dtr_pred = dtr_model.predict(X2_test)

print(dtr_pred)

print(y2_test)
```

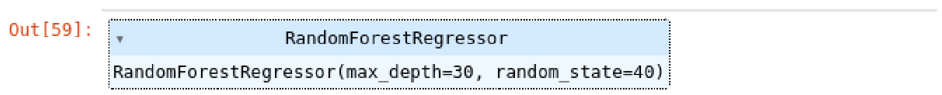


```
True values
203536      28
301801     176
254032     391
339158      14
3203       405
           ...
95283      136
282522    1608
195794     406
446517      82
302061     405
Name: num_orders, Length: 91310, dtype: int64
Predicted values
[119.60623229 302.49       155.92631579 ... 235.48648649  71.39808917
 441.63417155]
```

Figure 3.6: Decision Tree Regression

3.4.3 Random Forest Regression

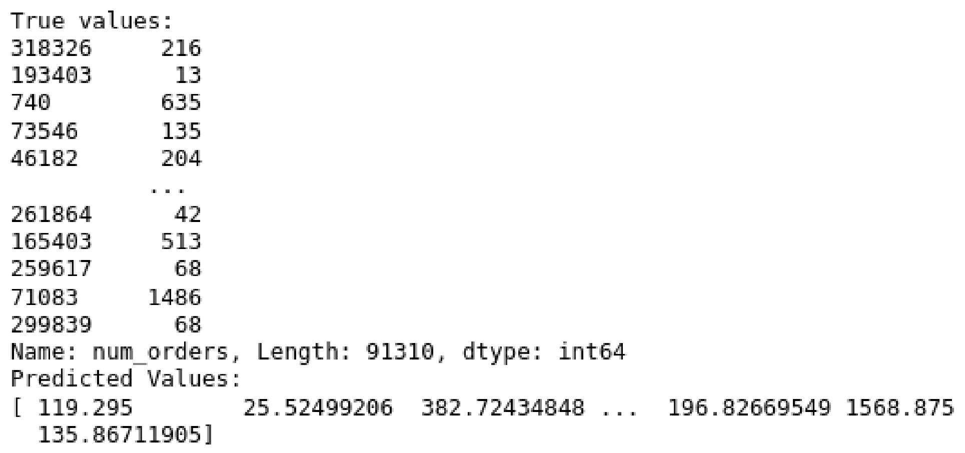
```
from sklearn.ensemble import RandomForestClassifier

X3_train,X3_test,y3_train,y3_test= train_test_split(X1,y1,test_size=0.2,

random_state=40)

rf_model = RandomForestClassifier(max_depth=16, random_state=0)

rf_model.fit(X3_train,y3_train)
```



```
Out[59]:
RandomForestRegressor
RandomForestRegressor(max_depth=30, random_state=40)
```

Figure 3.7: Random Forest Regression

```
rf_pred = rf_model.predict(X3_test)

print("True values:")

print(y3_test)

print("Predicted Values:")

print(rf_pred)
```

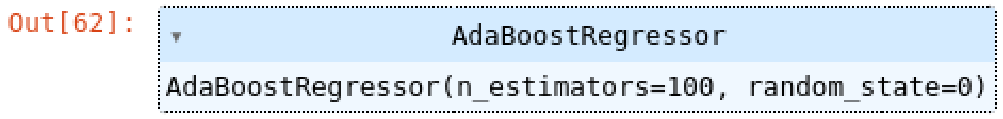


```
True values:
318326     216
193403      13
740        635
73546      135
46182      204
          ...
261864      42
165403     513
259617      68
71083     1486
299839      68
Name: num_orders, Length: 91310, dtype: int64
Predicted Values:
[ 119.295        25.52499206  382.72434848 ...  196.82669549 1568.875
  135.86711905]
```

Figure 3.8: Random Forest Regression Predictions

3.4.4 Ada Boost Regression

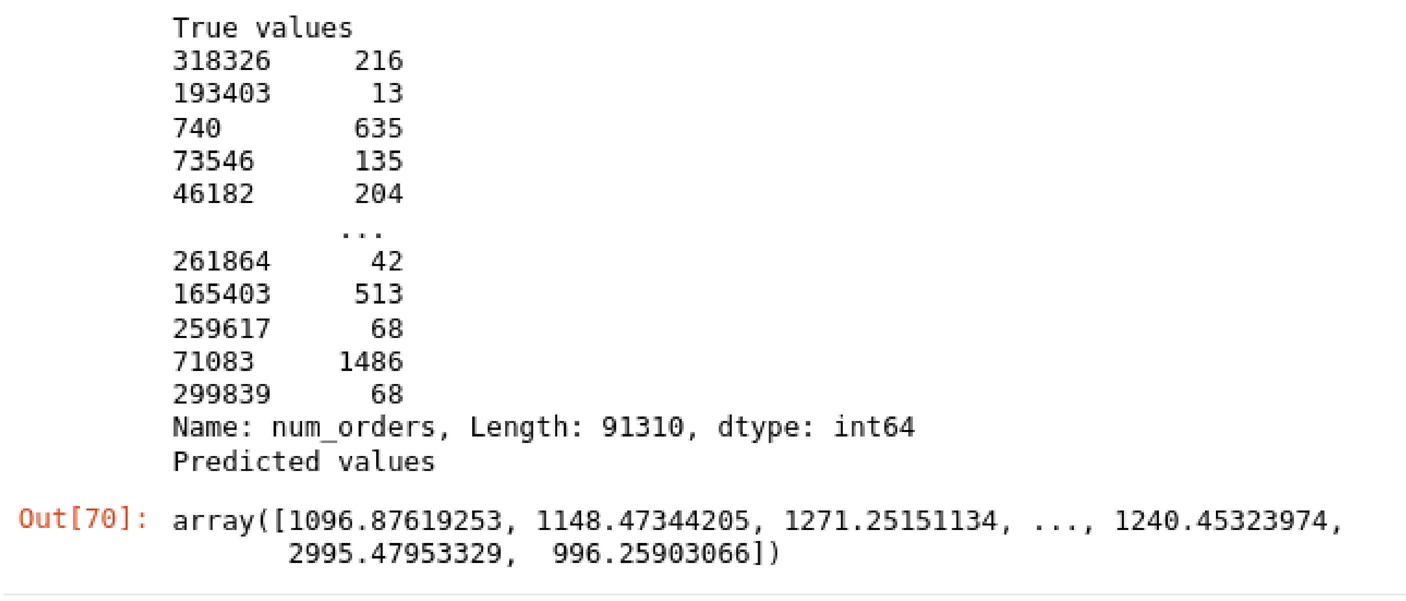
```
from sklearn.ensemble import AdaBoostRegressor

X4_train,X4_test,y4_train,y4_test= train_test_split(X1,y1,test_size=0.2,

random_state=40)

regr = AdaBoostRegressor(random_state=0, n_estimators=100)

regr.fit(X4_train,y4_train)
```



```
Out[62]:  AdaBoostRegressor
          AdaBoostRegressor(n_estimators=100, random_state=0)
```

Figure 3.9: Ada Boost Regression

```
print("True values")

print(y4_test)

print("Predicted values")

predboost=regr.predict(X4_test)

predboost
```



```
True values
318326     216
193403      13
740        635
73546      135
46182      204
          ...
261864      42
165403     513
259617      68
71083     1486
299839      68
Name: num_orders, Length: 91310, dtype: int64
Predicted values
```

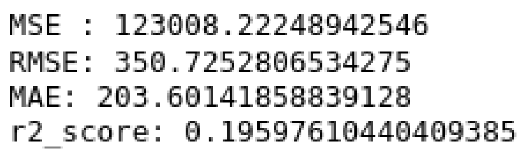
```
Out[70]: array([1096.87619253, 1148.47344205, 1271.25151134, ..., 1240.45323974,
                2995.47953329,  996.25903066])
```

Figure 3.10: Ada Boost Regression Predictions

3.5 Evaluation Metrics

3.5.1 Linear Regression

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score,accuracy

print("MSE :",mean_squared_error(y1_test, lr_pred))

print("RMSE:" ,np.sqrt(mean_squared_error(y1_test, lr_pred)))

print("MAE:",mean_absolute_error(y1_test, lr_pred))

print("r2_score:",r2_score(y1_test, lr_pred))
```

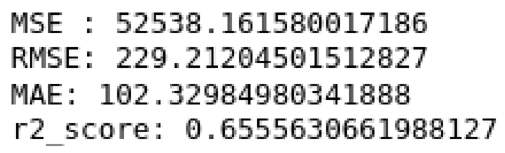


```
MSE : 123008.22248942546
RMSE: 350.7252806534275
MAE: 203.60141858839128
r2_score: 0.19597610440409385
```

Figure 3.11: Linear Regression Predictions

3.5.2 Decision Tree Regression

```
print("MSE :",mean_squared_error(y2_test, dtr_pred))

print("RMSE:" ,np.sqrt(mean_squared_error(y2_test, dtr_pred)))
print("MAE:",mean_absolute_error(y2_test, dtr_pred))

print("r2_score:",r2_score(y2_test, dtr_pred))
```

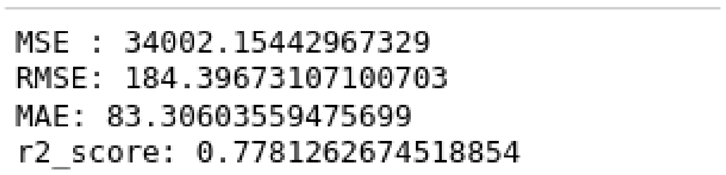


```
MSE : 52538.161580017186
RMSE: 229.21204501512827
MAE: 102.32984980341888
r2_score: 0.6555630661988127
```

Figure 3.12: Decision Tree Regression Predictions

3.5.3 Random Forest Regression

```
print("MSE :",mean_squared_error(y3_test, rf_pred))

print("RMSE:" ,np.sqrt(mean_squared_error(y3_test, rf_pred)))

print("MAE:",mean_absolute_error(y3_test, rf_pred))

print("r2_score:",r2_score(y3_test, rf_pred))
```

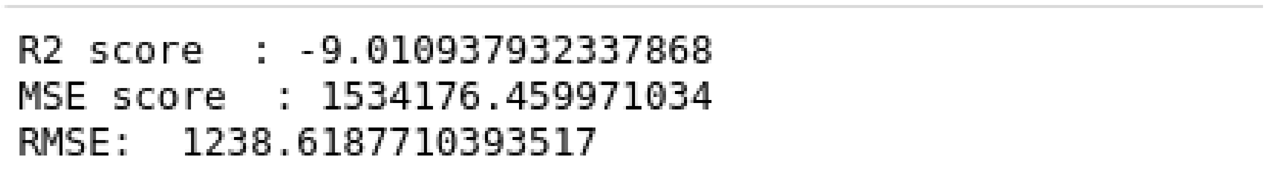


```
MSE : 34002.15442967329
RMSE: 184.39673107100703
MAE: 83.30603559475699
r2_score: 0.7781262674518854
```

Figure 3.13: Random Forest Regression Predictions

3.5.4 Ada Boost Regression

```
print("R2 score  :",r2_score(y4_test,predboost))

print("MSE score  :",mean_squared_error(y4_test,predboost))

print("RMSE: ",np.sqrt(mean_squared_error(y4_test,predboost)))
```



```
R2 score  : -9.010937932337868
MSE score  : 1534176.459971034
RMSE:  1238.6187710393517
```

Figure 3.14: Ada Boost Regression Predictions

These are the regression techniques used in building an efficient model to predict the accurate outputs.

Chapter 4

Conclusion and Future Work

In our DSP (Data Science Project) on food demand forecasting, we are able to predict the demand for various food items using historical data sourced from Kaggle. This project involved a thorough exploration of the dataset, feature engineering, model training, and evaluation to identify most suitable machine learning model for accurate forecasting. Here is the visual representation of models evaluation.

```
results = {'Model': ['LinearRegression','DecisionTreeRegression',
           'RandomForestRegression','AdaBoostRegression'],
           'MeanSquaredError': [123008.22248942546,52538.161580017186,
           34002.15442967329,1534176.45],
           'R2score': [0.19,0.65, 0.77,-9.010937932337868]}
results_df = pd.DataFrame(results)
results_df
```

Performance Chart

Performance Chart helps us to visualize the performance of the models. We use them as shown below.

Plotting the Mean_square_error of the models

```python
# Plot performance chart
plt.figure(figsize=(10, 6))
plt.bar(results_df['Model'], results_df['MeanSquaredError'], color='orange',
label='MSE')
plt.xlabel('Performance')
plt.title('Performance Comparison of ML Algorithms')
plt.legend()
plt.show()
```

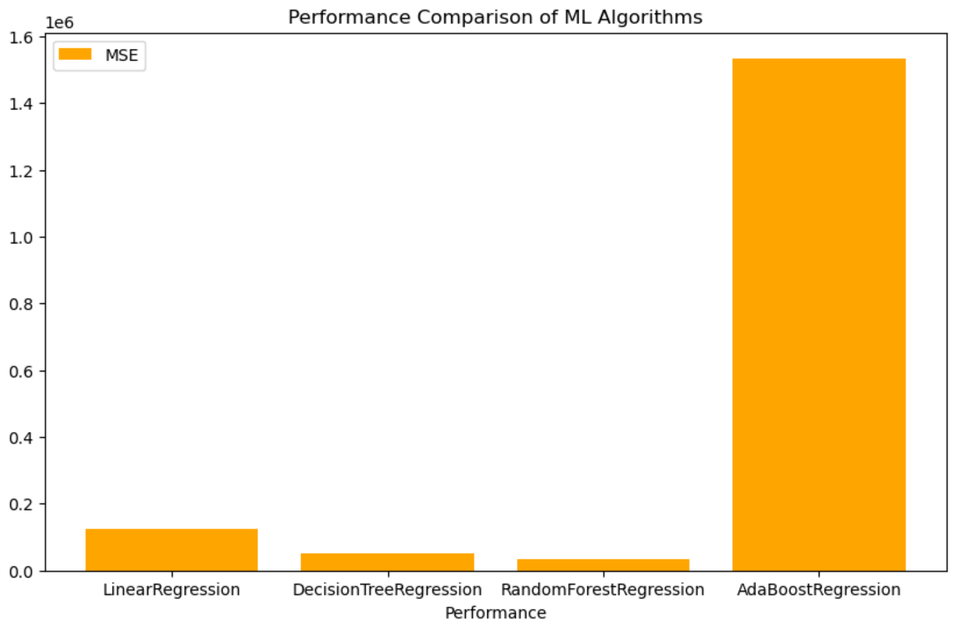


Figure 4.1: Mean_square_error of models

Mean_square_error of Random Forest Regression seems to lesser value.

Plotting the R2_score of the models

```python
# Plot performance chart
plt.figure(figsize=(10, 6))
plt.bar(results_df['Model'], results_df['R2score'], color='skyblue',
label='R2score')
plt.xlabel('Performance')
plt.title('Performance Comparison of ML Algorithms')

plt.ylim(0,1)

plt.legend()

plt.show()
```

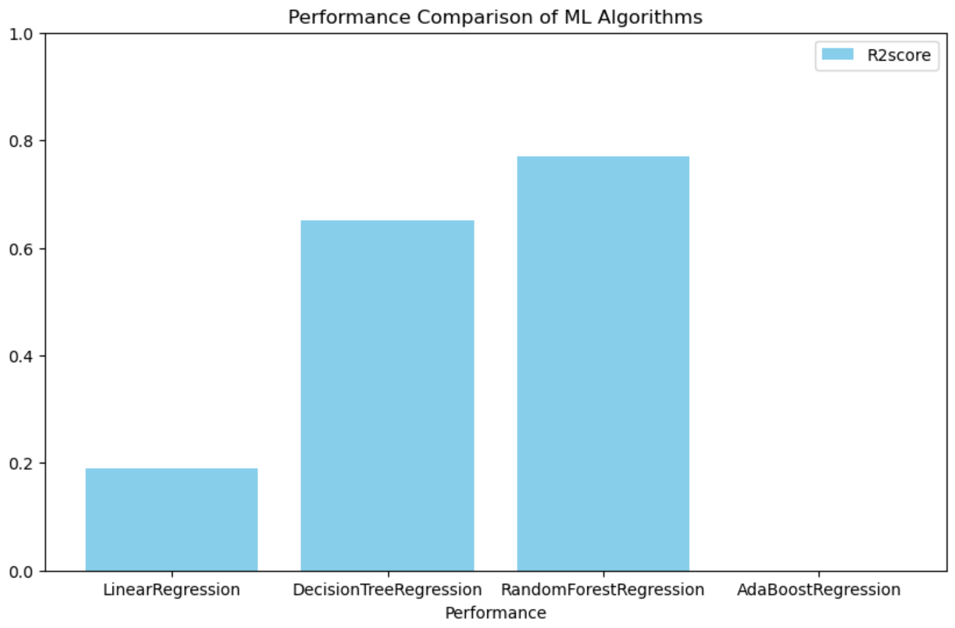


Figure 4.2: R2_score of models

Among all the models evaluated, the Random Forest Regressor demonstrated superior performance. The final evaluation of the Random Forest Regressor on the testing set yielded the following metrics:

MSE : 34002.15442967329

RMSE: 184.39673107100703

MAE: 83.30603559475699

r2_score: 0.7781262674518854

The Random Forest Regressor achieved the highest R2_score and the lowest MSE and RMSE among all models tested, indicating its robustness and accuracy in predicting food demand. These metrics validated the Random Forest Regressor as a reliable and effective model for our food demand forecasting project because of its Handling Non-Linear Relationships,feature importance,model's tree structure and Performance Metrics.

The Random Forest Regressor proved to be the most suitable model for our food demand forecasting project, providing accurate predictions and valuable insights into the factors driving demand.

In summary, the project successfully demonstrated the application of machine learning techniques to solve real-world demand forecasting problems, showcasing the potential of Random Forest Regressor in achieving reliable and actionable predictions.