FOOD DEMAND FORECASTING

Project Report

MS IN BUSINESS ANALYTICS AND PROJECT MANAGEMENT, UNIVERSITY OF CONNECTICUT

OPIM 5512- DATA SCIENCE USING PYTHON

RAMESH SHANKAR

SUMANTH KUMAR GOUD, GOLLA

Contents

1. Introduction:

The aim of this project is to delve into the topic of food demand forecasting using Python. In the ever-changing landscape of the food industry, businesses must accurately anticipate consumer demand for specific products to make well-informed decisions and maintain a competitive edge. Traditional forecasting methods, however, can be labor-intensive, time-consuming, and prone to inaccuracies, leading to inefficiencies in inventory management, increased waste, and reduced profitability.

By harnessing the power of data analysis and machine learning libraries in Python, it is possible to create precise predictive models that can significantly improve inventory control, minimize waste, and enhance overall profitability. These advanced techniques offer a more reliable and efficient approach to forecasting, allowing businesses to adapt to fluctuations in consumer demand more effectively.

Through this project we aim to provide a comprehensive guide on the process of food demand forecasting using Python and machine learning models. By implementing these advanced techniques, businesses can make data-driven decisions, optimize their operations, and achieve long-term success. The following questions were explored through the project:

1. How does food demand vary by food category, cuisine type, city, and center?
2. How effective are promotional email campaigns and homepage featured items in driving food demand?
3. How can restaurants optimize their revenue?
4. How accurately can we predict food demand using the available data, and what factors are most important for making accurate predictions?

Business Relevance of these questions:

Food wastage is a pressing issue faced by restaurants across the globe. In the United States alone, an estimated 22 to 33 billion pounds of food is wasted each year by the restaurant industry. This not only has significant environmental and economic consequences but also highlights the need for more efficient inventory management in the food sector. By obtaining accurate estimates of customer orders, restaurants can better prepare for and manage their inventory, reducing food wastage and boosting overall efficiency. There are several analyses that can be performed to achieve this goal:

- **Demand Forecasting**: This approach aims to accurately predict the quantity of food required, ensuring that the right amount is produced without generating excess waste. By forecasting demand, restaurants can streamline their operations and effectively manage their workforce, further enhancing overall efficiency. Using Python and machine learning libraries, we can create precise predictive models for food demand, allowing for improved inventory control and reduced waste.

- **Menu Optimization**: Analyzing food demand data enables restaurants to identify popular items and those with lower demand. By optimizing their menus, restaurants can focus on offering dishes that attract more customers, increasing sales and customer satisfaction. Simultaneously, removing items that are not in high demand can help minimize food waste and improve the utilization of resources such as raw materials and kitchen staff.
- **Pricing Analysis:** Investigating the relationship between food prices and demand can provide valuable insights for restaurants seeking to optimize their pricing strategies. By understanding how various price points affect customer behavior, restaurants can develop pricing models that generate more revenue while maintaining customer satisfaction. This approach can further contribute to reducing food wastage, as efficient pricing strategies can help balance supply and demand.

Addressing food wastage and enhancing inventory management in the restaurant industry are essential for achieving long-term sustainability and profitability.

2. Literature:

A study by Chen and Huang (2021) used a regression model to predict demand for fresh produce in a retail store based on historical sales data and weather information. Similarly, a study by Lozano et al. (2019) used a regression model to predict demand for a restaurant chain based on historical sales data. A couple of existing projects on Kaggle that tackled the topic of food demand forecasting primarily focused on predicting the demand for the next 10 weeks for a meal delivery company operating in multiple cities. These projects adopted a straightforward approach, consisting of light data exploration, data preprocessing, feature selection, and model building. The models used in these studies included Linear Regression, Decision Tree, and RandomForestRegressor, with R2 score and RMSE employed as performance metrics for evaluating the models.

In contrast, this project goes beyond these existing studies by incorporating a more in-depth approach to data analysis and aiming to explore specific questions that could provide valuable insights for the meal delivery company. One unique aspect of this project is the creation of a new variable called "discount,". Other additions include the utilization of a Gradient Boosting model, and the addition of an evaluation metric, Mean Absolute Percentage Error (MAPE). By addressing these four specific questions, this project can offer a more comprehensive understanding of the variables that influence food demand, enabling the meal delivery company to optimize its operations more effectively.

3. Dataset and Data Description:

The Food Demand Forecasting dataset on Kaggle is a collection of historical food demand data from 52 different outlets of a restaurant chain in India. The dataset contains data for a period of 145 weeks, from January 2017 to March 2020. The dataset consists of four files:

1. Train.csv (contains data for the first 100 weeks)
2. Test.csv (contains data for the next 45 weeks without target variable)
3. Meal_info.csv
4. Center_info.csv

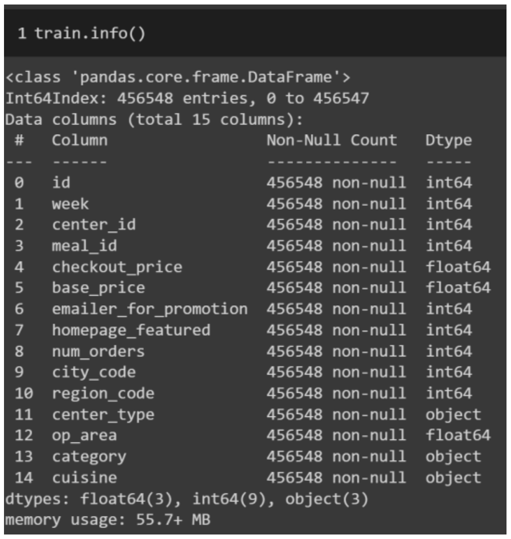
Variable description:

File	Variable	Description
Train.csv and Test.csv	id	Unique identifier for each record in the dataset.
	week	Week number (1-145) when the food demand data was recorded.
	center_id	Unique identifier for each fulfillment center (outlet) where the food is prepared and delivered.
	meal_id	Unique identifier for each meal that is served by the restaurant.
	checkout_price	The price of the meal at the time of order checkout.
	base_price	The base price of the meal, which is the original price before any discounts or promotions.
	emailer_for_promotion	Whether an email was sent to the customers promoting the meal.
	homepage_featured	Whether the meal was featured on the homepage of the restaurant website.
	num_orders (Target variable – only in train.csv)	The target variable, which is the number of food items ordered for a particular meal and center for a given week.
meal_info.csv	meal_id	Unique identifier for each meal that is served by the restaurant.
	category	The broad category to which the meal belongs, such as Beverages, Snacks, or Starters.
	cuisine	The type of cuisine that the meal belongs to, such as Thai, Indian, or Italian.

fulfilment_center_info.csv	center_id	Unique identifier for each fulfillment center (outlet) where the food is prepared and delivered.
	city_code	A unique identifier for the city where the center is located.
	region_code	A unique identifier for the region where the center is located.
	center_type	The type of the fulfillment center, such as TYPE_A, TYPE_B, or TYPE_C.
	op_area	The operational area of the center in square feet.

4. Data Transformations:

There are no null values in the data set:



```
1 train.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 456548 entries, 0 to 456547
Data columns (total 15 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
 9   city_code              456548 non-null  int64
 10  region_code            456548 non-null  int64
 11  center_type            456548 non-null  object
 12  op_area                456548 non-null  float64
 13  category               456548 non-null  object
 14  cuisine                456548 non-null  object
dtypes: float64(3), int64(9), object(3)
memory usage: 55.7+ MB
```

Merging Data Frames:

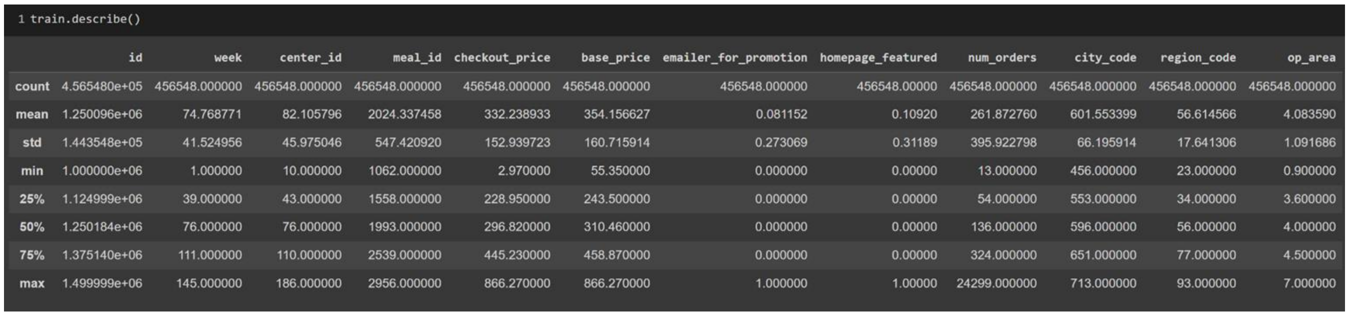
The ‘Merge' function was used to combine the 'itrain' and 'center' data frames using a left join on 'center_id' key. The resulting data frame was temporarily stored in the 'temp' variable. 'meal' data frame with the 'temp' data frame using another left join, this time on the 'meal_id' key. The resulting data frame was assigned to the variable 'train'.

A similar process was performed for the 'itest' data frame to create a new 'test' data frame. The 'test' data frame is created by merging the 'itest' and 'center' data frames on the 'center_id' key, followed by merging the resulting data frame with the 'meal' data frame on the 'meal_id' key.

By combining the three data frames into a single data frame, consolidated dataset was created that included information on the meal, center, and week attributes, which was used for training and evaluating machine learning models for food demand forecasting.

Data type modification:

The columns ‘center_id', 'meal_id', 'category', 'cuisine', 'city_code', 'region_code', and 'center_type' were changed from ‘Int64’ to ‘category’ type as they represented data with a limited number of unique values. This data type conversion was performed using the 'astype()' function in Pandas, which converts the data type of a column to a specified data type.



```
1 train.describe()
```

```
                 id           week      center_id        meal_id  checkout_price     base_price  emailer_for_promotion  homepage_featured     num_orders      city_code    region_code        op_area
count  4.565480e+05  456548.000000  456548.000000  456548.000000   456548.000000  456548.000000          456548.000000       456548.00000  456548.000000  456548.000000  456548.000000  456548.000000
mean   1.250096e+06      74.768771      82.105796    2024.337458      332.238933     354.156627               0.081152            0.10920     261.872760     601.553399      56.614566       4.083590
std    1.443548e+05      41.524956      45.975046     547.420920      152.939723     160.715914               0.273069            0.31189     395.922798      66.195914      17.641306       1.091686
min    1.000000e+06       1.000000      10.000000    1062.000000        2.970000      55.350000               0.000000            0.00000      13.000000     456.000000      23.000000       0.900000
25%    1.124999e+06      39.000000      43.000000    1558.000000      228.950000     243.500000               0.000000            0.00000      54.000000     553.000000      34.000000       3.600000
50%    1.250184e+06      76.000000      76.000000    1993.000000      296.820000     310.460000               0.000000            0.00000     136.000000     596.000000      56.000000       4.000000
75%    1.375140e+06     111.000000     110.000000    2539.000000      445.230000     458.870000               0.000000            0.00000     324.000000     651.000000      77.000000       4.500000
max    1.499999e+06     145.000000     186.000000    2956.000000      866.270000     866.270000               1.000000            1.00000   24299.000000     713.000000      93.000000       7.000000
```

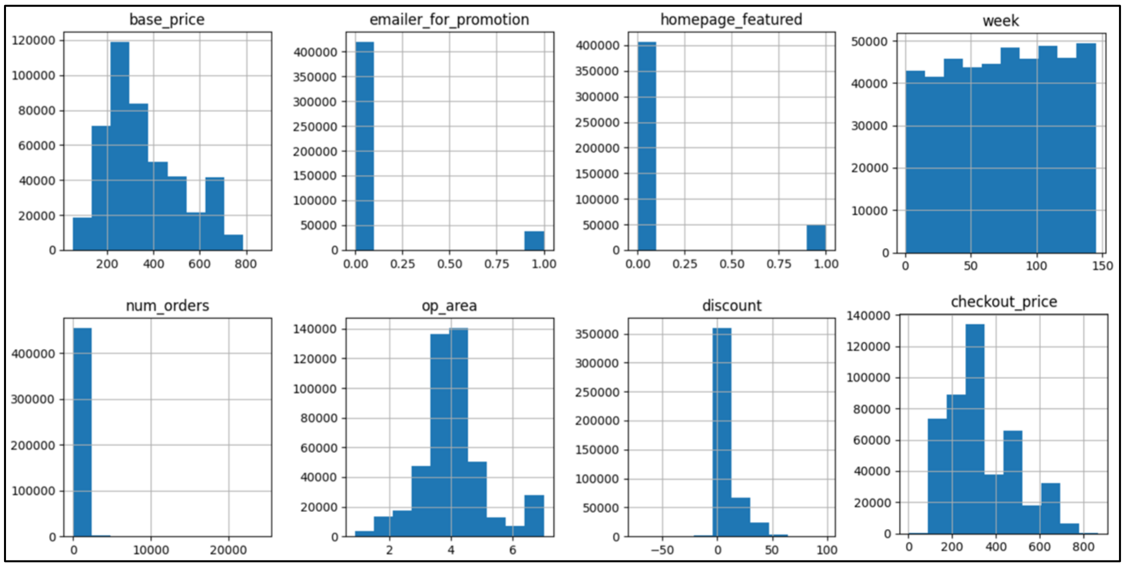
New variable introduction:

A new column ‘discount’ was added in both 'train' and 'test' dataframes. It is based on the percentage difference between the base price and the checkout price.

5. Exploratory Data Analysis:

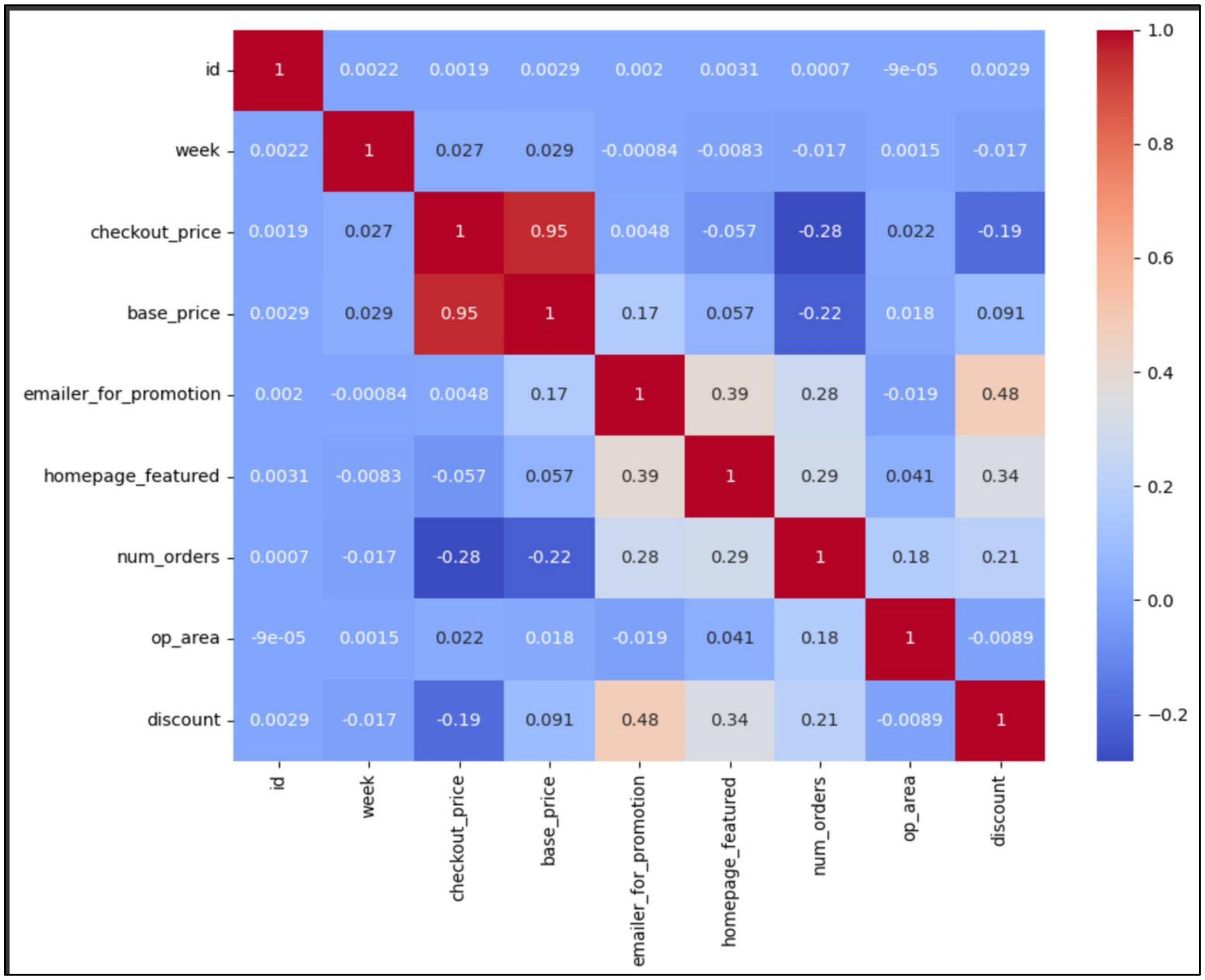
Histogram:

A histogram plot for all variables was plotted to understand the distribution of the data.



From the above graphs it can be observed that there are some outliers in the target variable ‘num_orders.

Correlation Matrix:



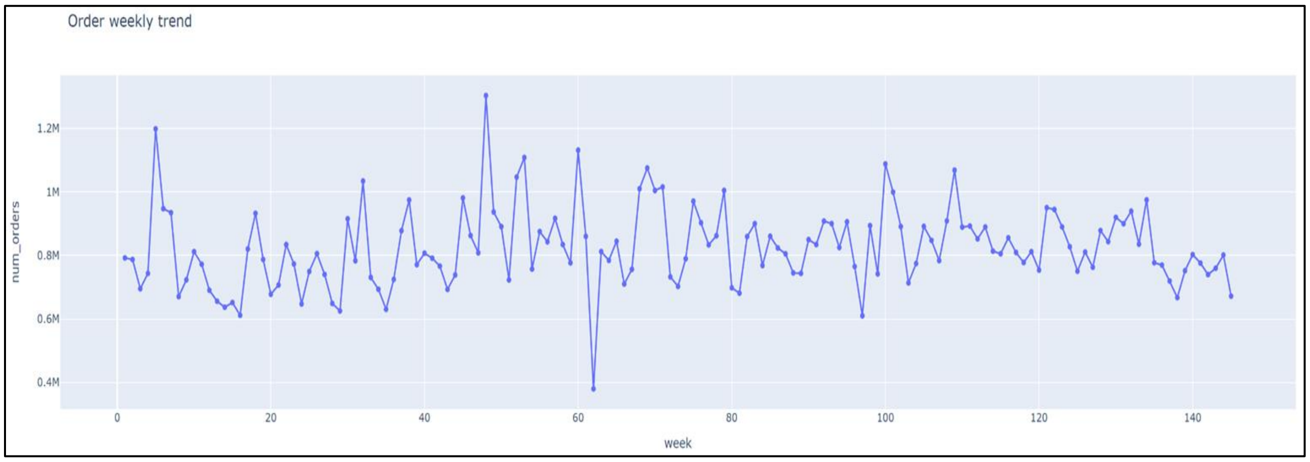
We have checked for the correlation among the variables and observed that checkout_price and base_price have a positive correlation, meaning that when one goes up, the other also tends to go up. Therefore, to avoid multicollinearity in the regression analysis, one of them needs to be removed. Since base_price has less correlation with the target variable, it was removed.

The 'id' column was removed from the analysis as it is only used to differentiate distinct rows and does not provide any valuable information for predicting the target variable.

The variables that could function as the best predictors were homepage_featured, emailer_for_promotion, and checkout_price. These variables are likely to have a significant impact on the target variable and can be used to build a regression model for demand forecasting.

Trend and Seasonality:

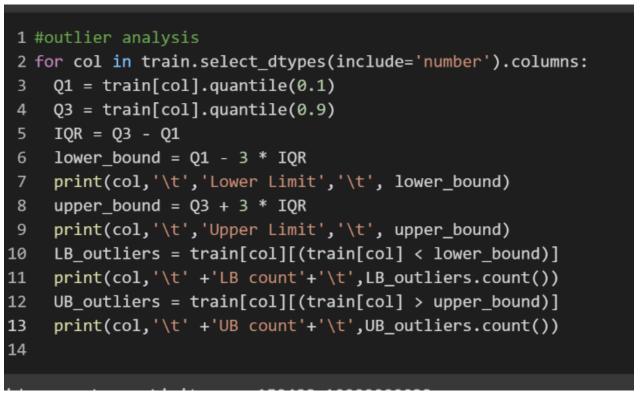
Since there are multiple orders per particular time period in the data, we were not able to perform 'Time Series Forecasting System' method for predicting the future orders. To understand the trend and seasonality in the data, the total number of orders were plotted against corresponding week number.



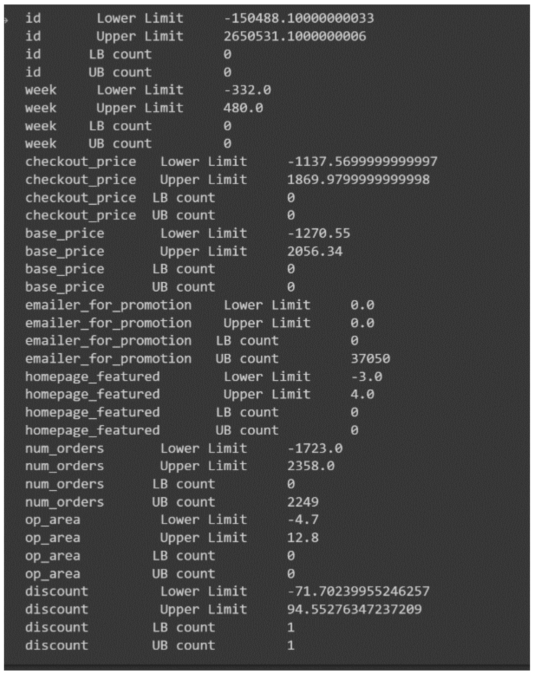
The graph suggests that there is no trend or seasonality in the data.

Outlier Analysis:

A 'for' loop was created for each variable to calculate the first and third quartiles (Q1 and Q3), the interquartile range (IQR), and the lower and upper bounds for outliers, defined as values that are more than 3 times the IQR below Q1 or above Q3. It then counts the number of outliers that fall below the lower bound (LB count) or above the upper bound (UB count) for each column.



```
#outlier analysis
for col in train.select_dtypes(include='number').columns:
  Q1 = train[col].quantile(0.1)
  Q3 = train[col].quantile(0.9)
  IQR = Q3 - Q1
  lower_bound = Q1 - 3 * IQR
  print(col,'\t','Lower Limit','\t', lower_bound)
  upper_bound = Q3 + 3 * IQR
  print(col,'\t','Upper Limit','\t', upper_bound)
  LB_outliers = train[col][(train[col] < lower_bound)]
  print(col,'\t' +'LB count'+'\t',LB_outliers.count())
  UB_outliers = train[col][(train[col] > upper_bound)]
  print(col,'\t' +'UB count'+'\t',UB_outliers.count())

```



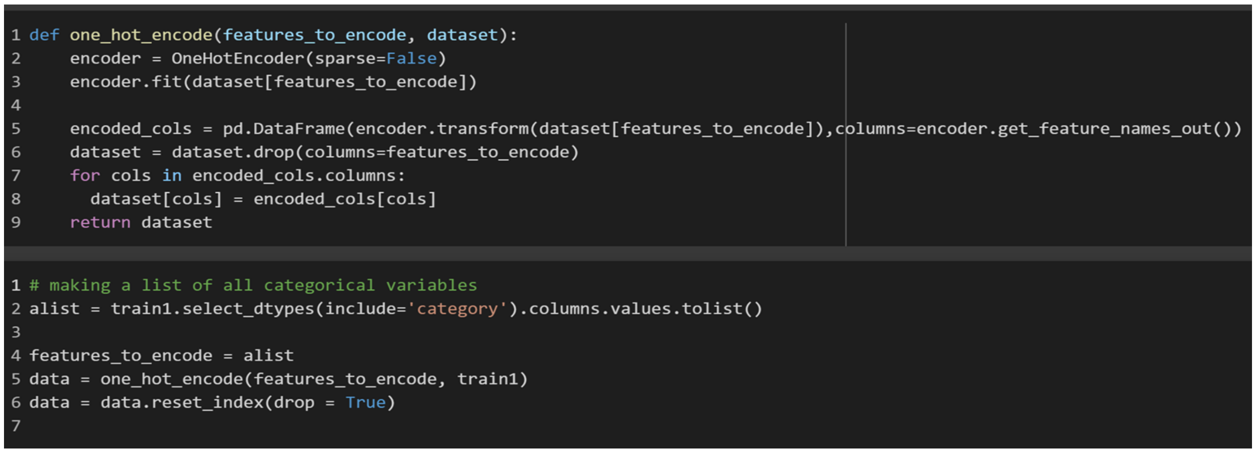
```
id       Lower Limit     -150488.10000000033
id       Upper Limit     2650531.1000000006
id      LB count         0
id      UB count         0
week     Lower Limit     -332.0
week     Upper Limit     480.0
week    LB count         0
week    UB count         0
checkout_price   Lower Limit     -1137.5699999999997
checkout_price   Upper Limit     1869.9799999999998
checkout_price  LB count         0
checkout_price  UB count         0
base_price       Lower Limit     -1270.55
base_price       Upper Limit     2056.34
base_price      LB count         0
base_price      UB count         0
emailer_for_promotion    Lower Limit     0.0
emailer_for_promotion    Upper Limit     0.0
emailer_for_promotion   LB count         0
emailer_for_promotion   UB count         37050
homepage_featured        Lower Limit     -3.0
homepage_featured        Upper Limit     4.0
homepage_featured       LB count         0
homepage_featured       UB count         0
num_orders       Lower Limit     -1723.0
num_orders       Upper Limit     2358.0
num_orders      LB count         0
num_orders      UB count         2249
op_area          Lower Limit     -4.7
op_area          Upper Limit     12.8
op_area         LB count         0
op_area         UB count         0
discount         Lower Limit     -71.70239955246257
discount         Upper Limit     94.55276347237209
discount        LB count         1
discount        UB count         1
```

The output shows that there are significant outliers in 'num_orders,' which was also evident in the histogram plot. These outliers in the target variable were removed by filtering out the rows where the value of "num_orders" is greater than 2358.

6. Predictive Modelling:

Data preparation:

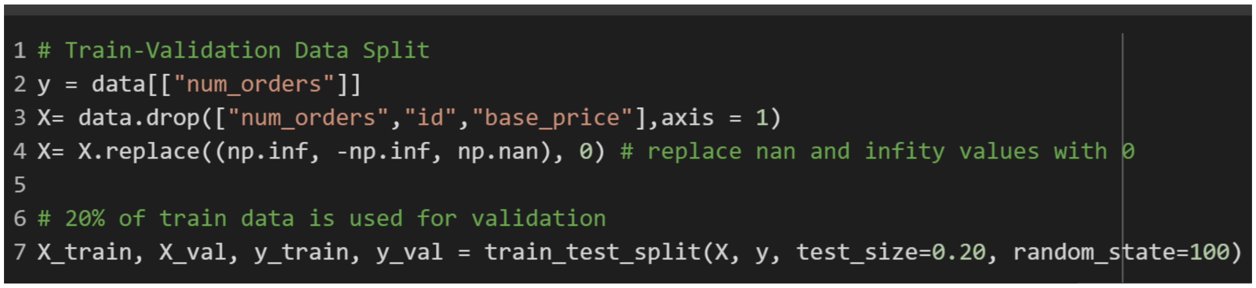
A new function 'one_hot_encode' was defined for converting the categorical variables into dummy variables. This process converts categorical variables into a binary representation, where each unique category becomes a binary column. The function first creates an instance of OneHotEncoder and fits it to the given dataset's categorical features. Then, the function applies the transformation on the features and creates new columns for each unique category. Finally, the original categorical columns are dropped, and the new binary columns are added to the dataset. The function returns the updated dataset as output.



```python
def one_hot_encode(features_to_encode, dataset):
    encoder = OneHotEncoder(sparse=False)
    encoder.fit(dataset[features_to_encode])

    encoded_cols = pd.DataFrame(encoder.transform(dataset[features_to_encode]),columns=encoder.get_feature_names_out())
    dataset = dataset.drop(columns=features_to_encode)
    for cols in encoded_cols.columns:
      dataset[cols] = encoded_cols[cols]
    return dataset
```

```python
# making a list of all categorical variables
alist = train1.select_dtypes(include='category').columns.values.tolist()

features_to_encode = alist
data = one_hot_encode(features_to_encode, train1)
data = data.reset_index(drop = True)

```

All required variables except target variable were assigned to 'X' and target variable to 'Y.' all the null values were replaced with 0. The entire dataset was split into two parts - 80% of the data was used as training data, and 20% of the data was used as validation data, with a random state=100 to ensure reproducibility of results. The training data was used to train the model, while the validation data was used to evaluate the performance of the trained model and tune the hyperparameters to improve the model's performance.



```python
# Train-Validation Data Split
y = data[["num_orders"]]
X= data.drop(["num_orders","id","base_price"],axis = 1)
X= X.replace((np.inf, -np.inf, np.nan), 0) # replace nan and infity values with 0

# 20% of train data is used for validation
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.20, random_state=100)
```

Predictive Models:

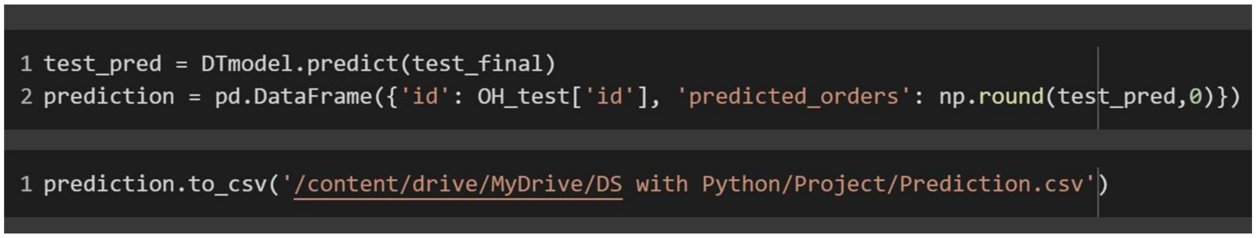
Models used and evaluation metrics are summarized in the below table. (Code used for models available in appendix)

Parameter	**Linear Regression**	**Decision Tree**	**Random Forest**	**Gradient Boosting**
R2_Score	0.24	0.57	0.38	0.55
RMSE	263.29	198.63	236.74	201.00
MAPE	2.26	1.08	1.83	0.85

Based on the evaluation metrics, Decision Tree model performs better than the other models. It has a higher R2 score of 0.57, which means that it explains more variance in the target variable compared to the other models. Additionally, it has the lowest RMSE value of 198.63, which indicates that the predicted values are closer to the actual values compared to the other models.

While the Gradient Boosting model has a higher R2 score of 0.55 and a lower MAPE value of 0.85, the Decision Tree model is still preferred due to its lower RMSE value, which is a more important metric for this specific problem.

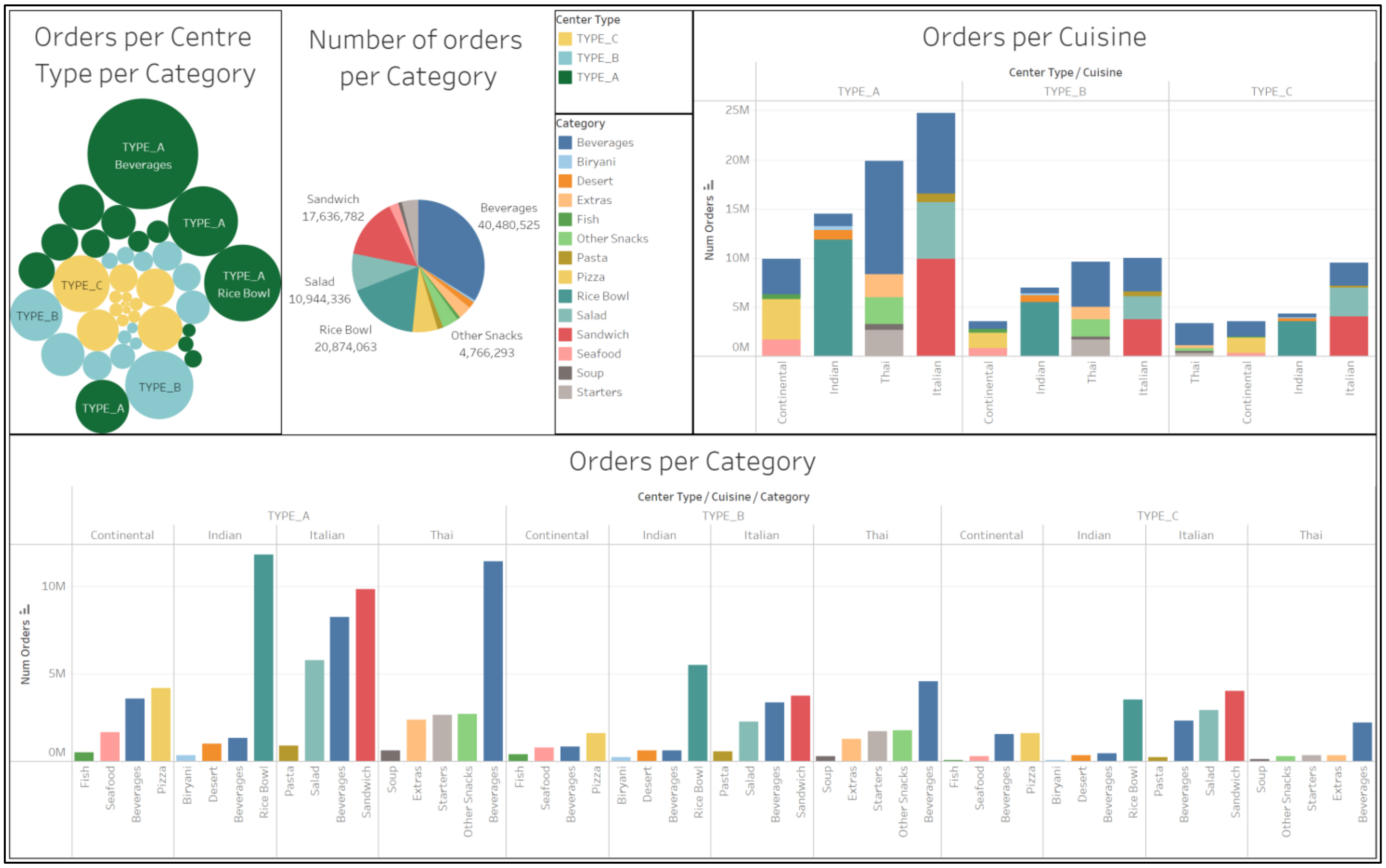
Decision tree model was used to predict the number of orders from the weeks in test dataframe.



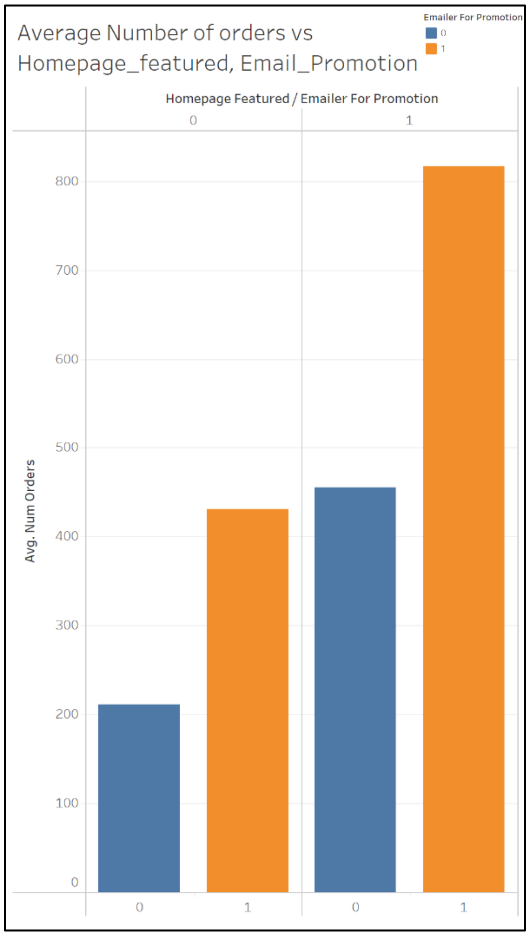
```
test_pred = DTmodel.predict(test_final)
prediction = pd.DataFrame({'id': OH_test['id'], 'predicted_orders': np.round(test_pred,0)})
```

```
prediction.to_csv('/content/drive/MyDrive/DS with Python/Project/Prediction.csv')
```

7. Findings:



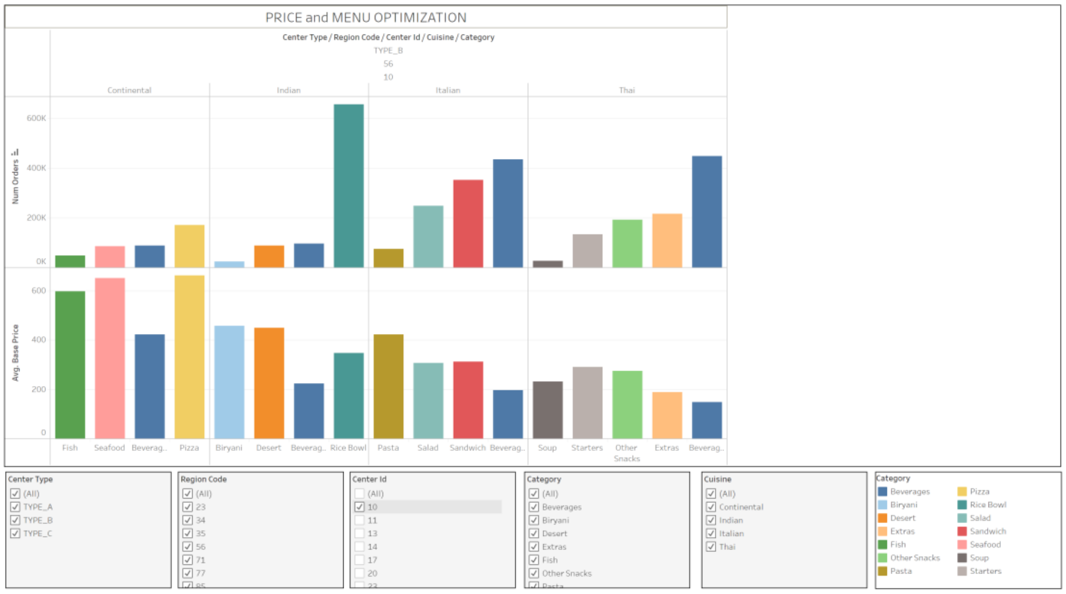
- Beverages are the most ordered item across all centers.
- Type A has the highest number of orders. It can also be a factor due to more data available in the Type-A center when compared to other centers.
- Most ordered food category per center:
    - Type- A: Rice bowl in Indian Cuisine
    - Type - B: Rice bowl in Indian Cuisine
    - Type - C: Sandwich in Italian Cuisine



- Items featured on the homepage received more orders compared to those that were not featured in homepage.
- Email promotions have doubled the average orders.

8.Recommendations for business

- Menu and Price Optimization:



Center Type
- [✓] (All)
- [✓] TYPE_A
- [✓] TYPE_B
- [✓] TYPE_C

Region Code
- [✓] (All)
- [✓] 23
- [✓] 34
- [✓] 35
- [✓] 56
- [✓] 71
- [✓] 77
- [✓] 85

Center Id
- [ ] (All)
- [✓] 10
- [ ] 11
- [ ] 13
- [ ] 14
- [ ] 17
- [ ] 20
- [ ] 23

Category
- [✓] (All)
- [✓] Beverages
- [✓] Biryani
- [✓] Desert
- [✓] Extras
- [✓] Fish
- [✓] Other Snacks
- [✓] Pasta

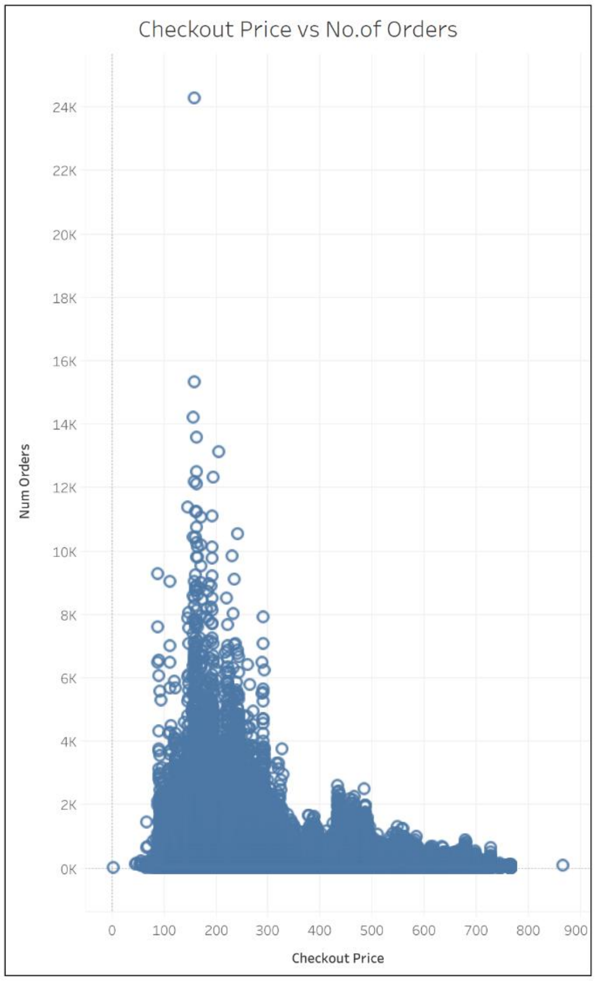
Cuisine
- [✓] (All)
- [✓] Continental
- [✓] Indian
- [✓] Italian
- [✓] Thai

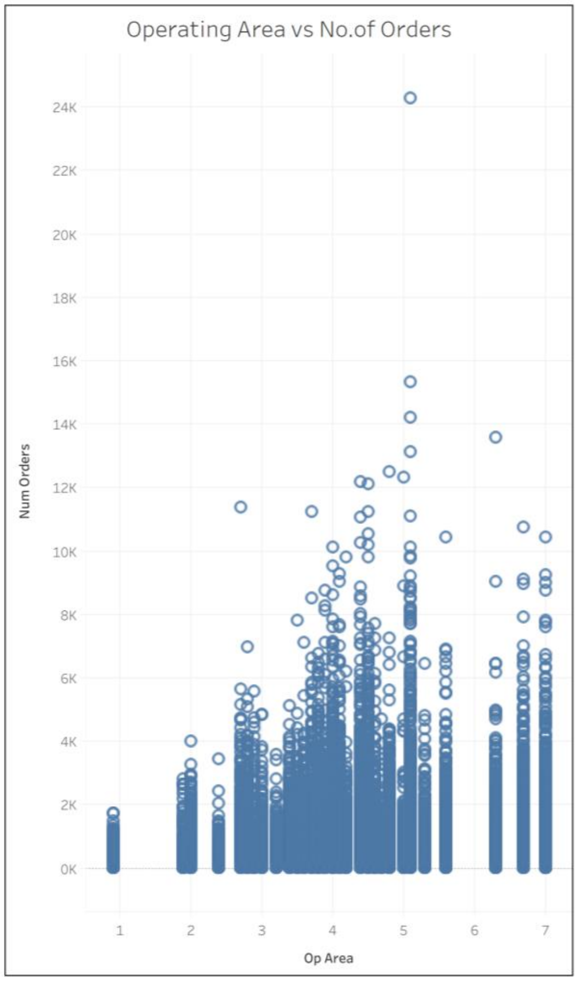
Category

🟦 Beverages	🟨 Pizza
🟦 Biryani	🟩 Rice Bowl
🟧 Desert	🟩 Salad
🟧 Extras	🟥 Sandwich
🟩 Fish	🟥 Seafood
🟩 Other Snacks	⬛ Soup
🟨 Pasta	⬜ Starters

From the above visualization, business owners can identify the gap between the orders vs base prices and regulate the prices wherever it is feasible to gain better revenue and profits.

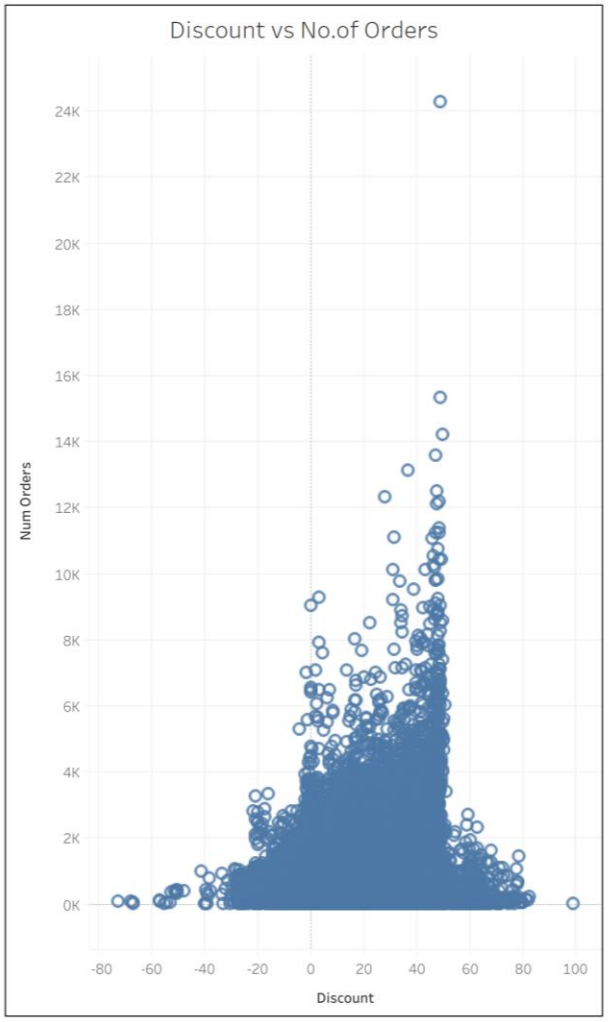
- The checkout_price vs no. of orders graph shows data which matches our intuition that lower the price, higher the number of orders.





- The operating area vs no. of orders graph shows that there are very few orders in regions with a low operating area. Business owners should focus on these areas to boost the orders by offering discounts and coupons.

- Below graph shows that offering discounts beyond 50% does not have any positive impact like improving the number of orders, while discounts from 0% to 50% and number of orders are directly related. So, business owners should not provide higher discounts.



References

1. Dataset from Kaggle:

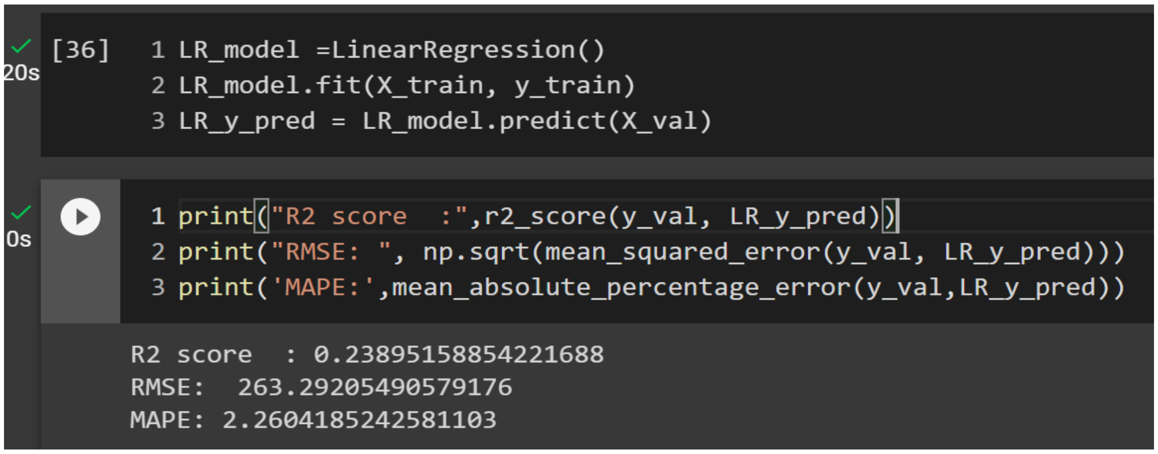
https://www.kaggle.com/datasets/kannanaikkal/food-demand-forecasting?select=meal_info.csv

2. Tableau Dashboard link:

https://public.tableau.com/app/profile/sumanth.kumar.goud.golla/viz/Book1_16811779920910/Dashboard3?publish=yes

Appendix

- Code for Linear Regression model:

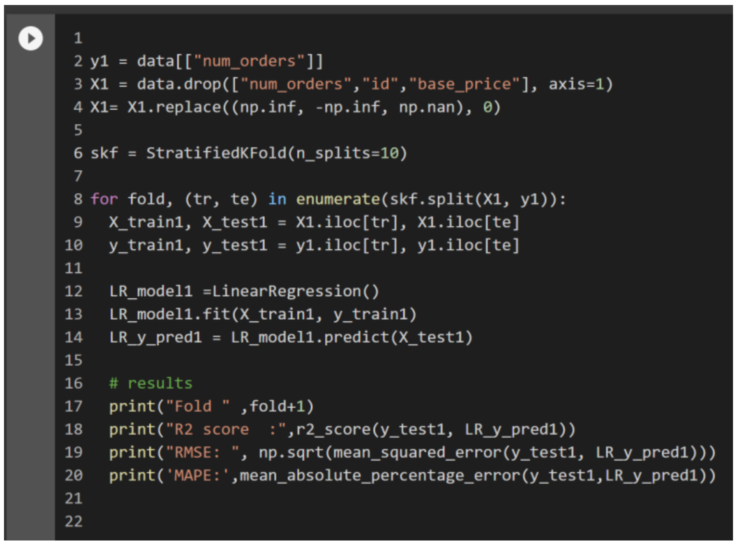


```
[36]
LR_model =LinearRegression()
LR_model.fit(X_train, y_train)
LR_y_pred = LR_model.predict(X_val)
```

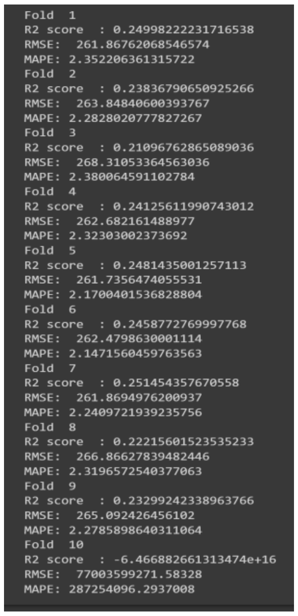
```
print("R2 score  :",r2_score(y_val, LR_y_pred))
print("RMSE: ", np.sqrt(mean_squared_error(y_val, LR_y_pred)))
print('MAPE:',mean_absolute_percentage_error(y_val,LR_y_pred))
```

```
R2 score  : 0.23895158854221688
RMSE:  263.29205490579176
MAPE: 2.2604185242581103
```

- Cross-Validation in Linear Regression model (no difference between 10 combinations)

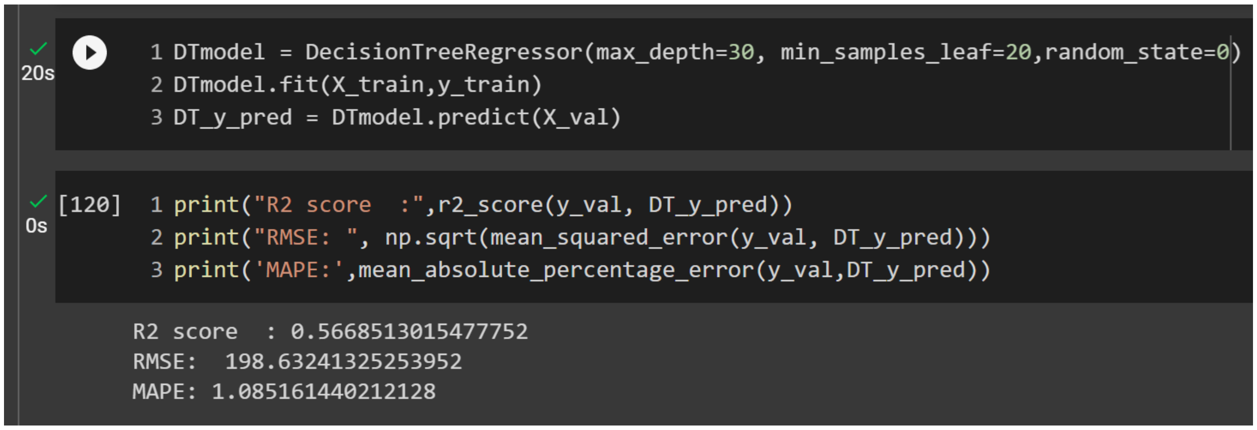


```

y1 = data[["num_orders"]]
X1 = data.drop(["num_orders","id","base_price"], axis=1)
X1= X1.replace((np.inf, -np.inf, np.nan), 0)

skf = StratifiedKFold(n_splits=10)

for fold, (tr, te) in enumerate(skf.split(X1, y1)):
  X_train1, X_test1 = X1.iloc[tr], X1.iloc[te]
  y_train1, y_test1 = y1.iloc[tr], y1.iloc[te]

  LR_model1 =LinearRegression()
  LR_model1.fit(X_train1, y_train1)
  LR_y_pred1 = LR_model1.predict(X_test1)

  # results
  print("Fold " ,fold+1)
  print("R2 score  :",r2_score(y_test1, LR_y_pred1))
  print("RMSE: ", np.sqrt(mean_squared_error(y_test1, LR_y_pred1)))
  print('MAPE:',mean_absolute_percentage_error(y_test1,LR_y_pred1))
```



```
Fold  1
R2 score  : 0.24998222231716538
RMSE:  261.86762068546574
MAPE: 2.352206361315722
Fold  2
R2 score  : 0.23836790650925266
RMSE:  263.84840600393767
MAPE: 2.2828020777827267
Fold  3
R2 score  : 0.21096762865089036
RMSE:  268.31053364563036
MAPE: 2.380064591102784
Fold  4
R2 score  : 0.24125611990743012
RMSE:  262.682161488977
MAPE: 2.32303002373692
Fold  5
R2 score  : 0.2481435001257113
RMSE:  261.7356474055531
MAPE: 2.1700401536828804
Fold  6
R2 score  : 0.2458772769997768
RMSE:  262.4798630001114
MAPE: 2.1471560459763563
Fold  7
R2 score  : 0.251454357670558
RMSE:  261.8694976200937
MAPE: 2.2409721939235756
Fold  8
R2 score  : 0.22215601523535233
RMSE:  266.86627839482446
MAPE: 2.3196572540377063
Fold  9
R2 score  : 0.23299242338963766
RMSE:  265.09242456102
MAPE: 2.2785898640311064
Fold  10
R2 score  : -6.466882661313474e+16
RMSE:  77003599271.58328
MAPE: 287254096.2937008
```

- Code for Decision Tree model:

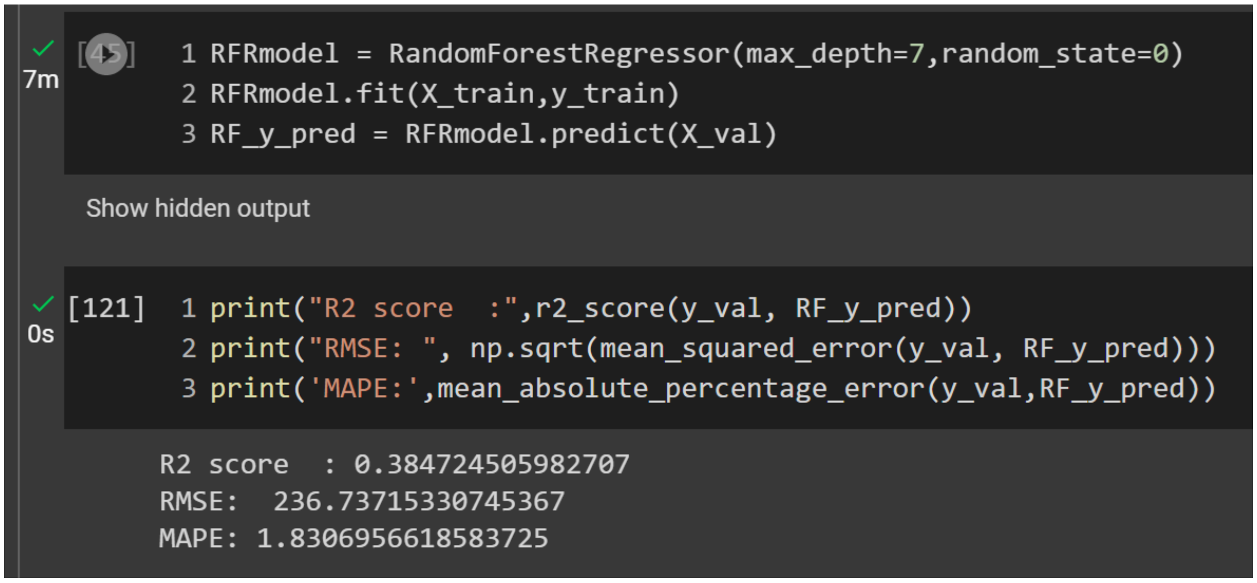


```
DTmodel = DecisionTreeRegressor(max_depth=30, min_samples_leaf=20,random_state=0)
DTmodel.fit(X_train,y_train)
DT_y_pred = DTmodel.predict(X_val)
```

```
print("R2 score  :",r2_score(y_val, DT_y_pred))
print("RMSE: ", np.sqrt(mean_squared_error(y_val, DT_y_pred)))
print('MAPE:',mean_absolute_percentage_error(y_val,DT_y_pred))
```

```
R2 score  : 0.5668513015477752
RMSE:  198.63241325253952
MAPE: 1.085161440212128
```

- Code for Random Forest Regression model:



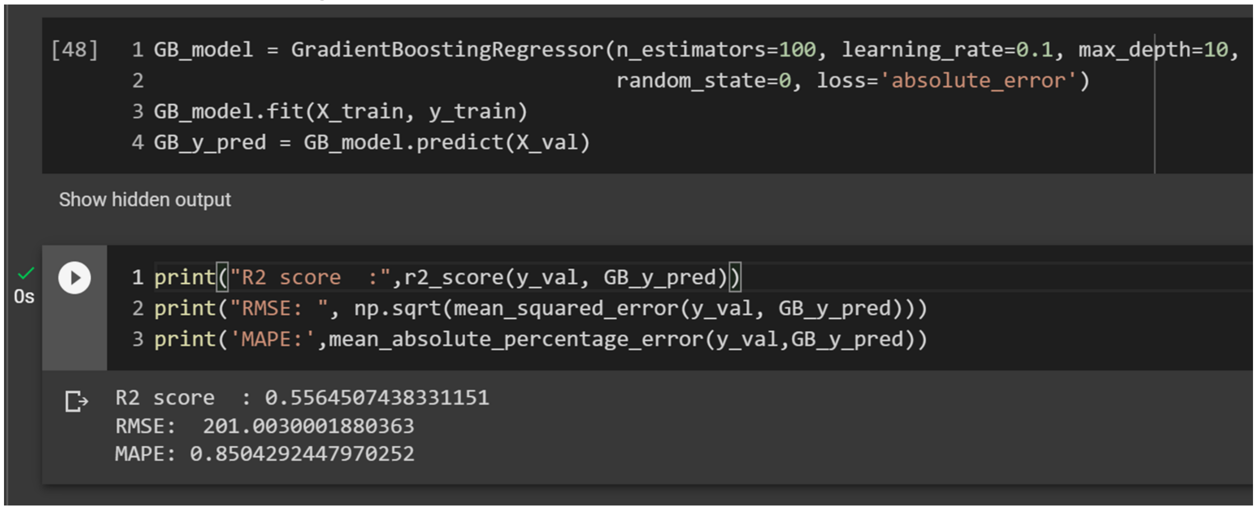
```
RFRmodel = RandomForestRegressor(max_depth=7,random_state=0)
RFRmodel.fit(X_train,y_train)
RF_y_pred = RFRmodel.predict(X_val)
```

Show hidden output

```
print("R2 score  :",r2_score(y_val, RF_y_pred))
print("RMSE: ", np.sqrt(mean_squared_error(y_val, RF_y_pred)))
print('MAPE:',mean_absolute_percentage_error(y_val,RF_y_pred))
```

```
R2 score  : 0.384724505982707
RMSE:  236.73715330745367
MAPE: 1.8306956618583725
```

- Code for Gradient Boosting model:



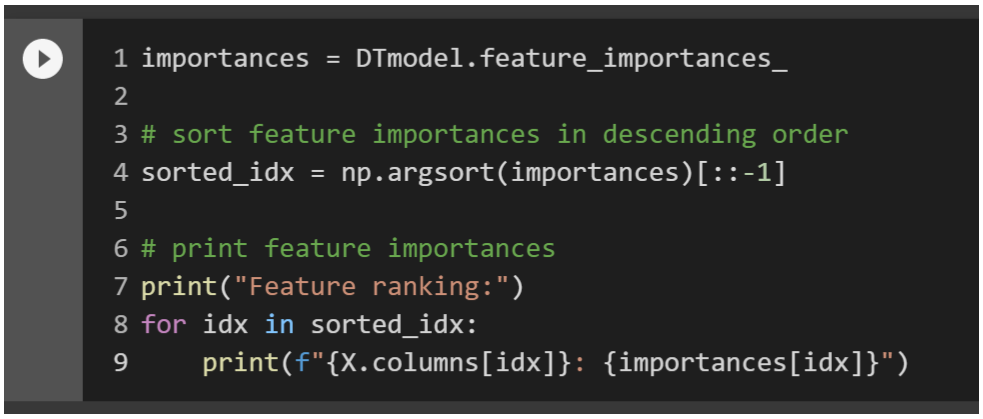
```
GB_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=10,
                                     random_state=0, loss='absolute_error')
GB_model.fit(X_train, y_train)
GB_y_pred = GB_model.predict(X_val)
```

Show hidden output

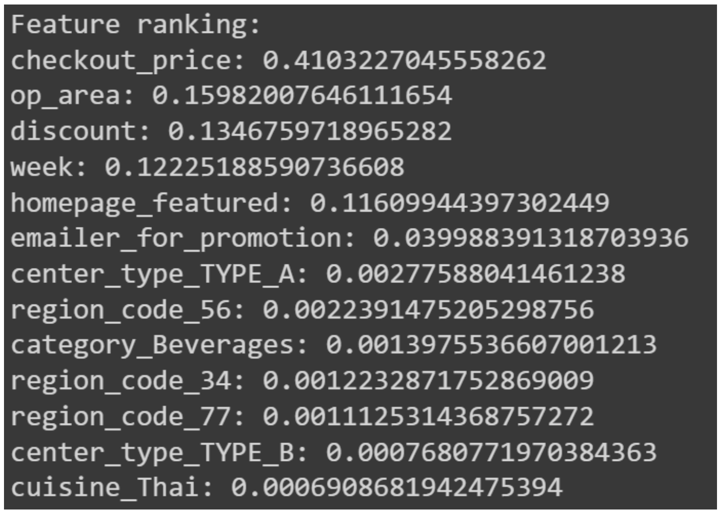
```
print("R2 score  :",r2_score(y_val, GB_y_pred))
print("RMSE: ", np.sqrt(mean_squared_error(y_val, GB_y_pred)))
print('MAPE:',mean_absolute_percentage_error(y_val,GB_y_pred))
```

```
R2 score  : 0.5564507438331151
RMSE:  201.0030001880363
MAPE: 0.8504292447970252
```

- Important Variables/ features in DT model:



```
importances = DTmodel.feature_importances_

# sort feature importances in descending order
sorted_idx = np.argsort(importances)[::-1]

# print feature importances
print("Feature ranking:")
for idx in sorted_idx:
    print(f"{X.columns[idx]}: {importances[idx]}")
```



```
Feature ranking:
checkout_price: 0.4103227045558262
op_area: 0.15982007646111654
discount: 0.1346759718965282
week: 0.12225188590736608
homepage_featured: 0.11609944397302449
emailer_for_promotion: 0.039988391318703936
center_type_TYPE_A: 0.00277588041461238
region_code_56: 0.0022391475205298756
category_Beverages: 0.0013975536607001213
region_code_34: 0.0012232871752869009
region_code_77: 0.0011125314368757272
center_type_TYPE_B: 0.0007680771970384363
cuisine_Thai: 0.0006908681942475394
```