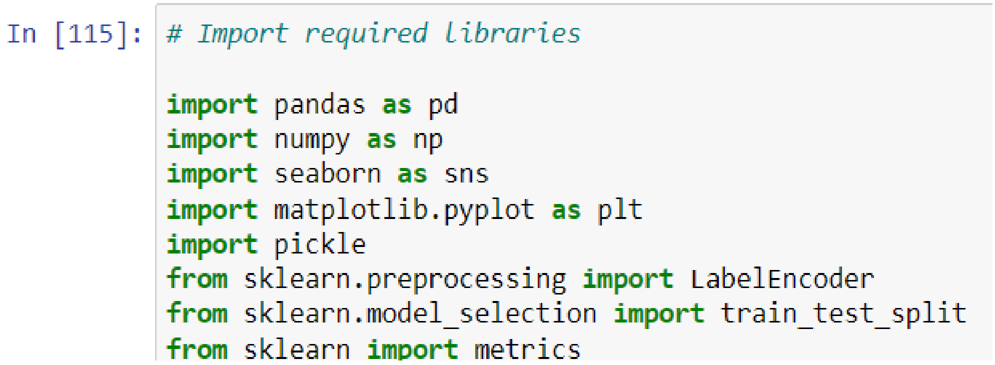
Data Pre-processing

Team ID	PNT2022TMID35762
Project Name	DEMANDEST – AI POWERED FOOD DEMAND FORECASTER

1. Import the required libraries:

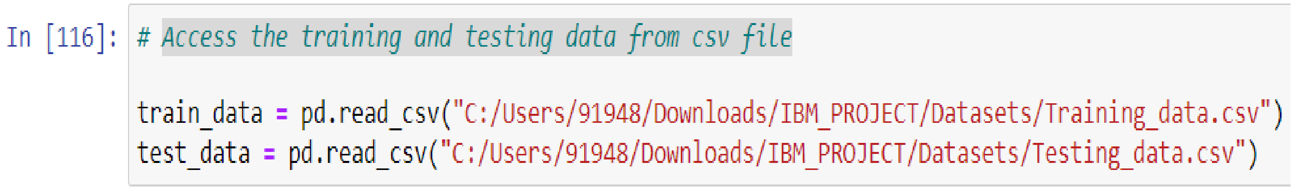
- **pandas -** tabular data can be analyzed and associated data frames can be manipulated.
- **numpy** - mathematical analyses can be performed on it using its multidimensional array object.
- **seaborn -** focuses on statistics visualisation and is utilised when it's necessary to show both the distribution of the data and its summary in visuals.
- **matplotlib -** make visualisations that are interactive, animated, and static.
- **sklearn -** provides a variety of effective tools for statistical modelling, including regression and classification.



```
In [115]: # Import required libraries

          import pandas as pd
          import numpy as np
          import seaborn as sns
          import matplotlib.pyplot as plt
          import pickle
          from sklearn.preprocessing import LabelEncoder
          from sklearn.model_selection import train_test_split
          from sklearn import metrics
```

2. Accessing the data from csv file:

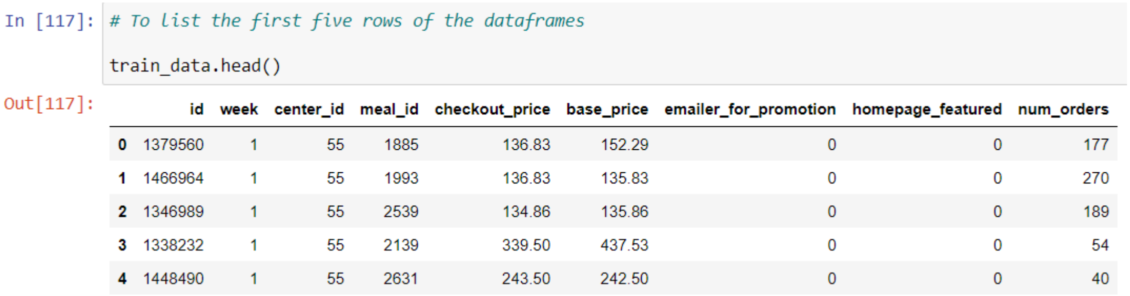
Importing data into the environment is the first step in analysis. For storing tabular data, CSVs are a common option and the quickest method to get started. To open and manipulate a.csv file, use the read.csv() function. It is possible to save and manipulate the contents of the csv file in the variable.



```
In [116]: # Access the training and testing data from csv file

          train_data = pd.read_csv("C:/Users/91948/Downloads/IBM_PROJECT/Datasets/Training_data.csv")
          test_data = pd.read_csv("C:/Users/91948/Downloads/IBM_PROJECT/Datasets/Testing_data.csv")
```

3. Analysing the data:

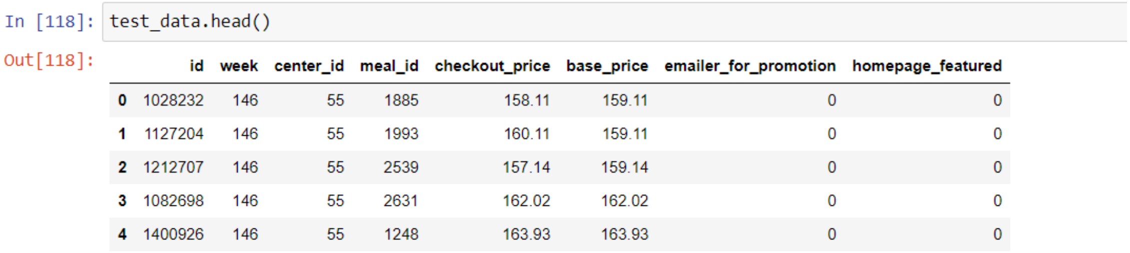
Data analysis is a method of examining data sets to highlight their key features, frequently using visual methods, and is used to decide how to best manipulate data sources to obtain the answers you need. This method makes it simpler for data scientists to find patterns, identify anomalies, test hypotheses, or verify assumptions. For example, info() method is used to get information which contains the number of columns, column labels, column data types, memory usage, range index.



```
In [117]: # To list the first five rows of the dataframes

          train_data.head()
```

Out[117]:

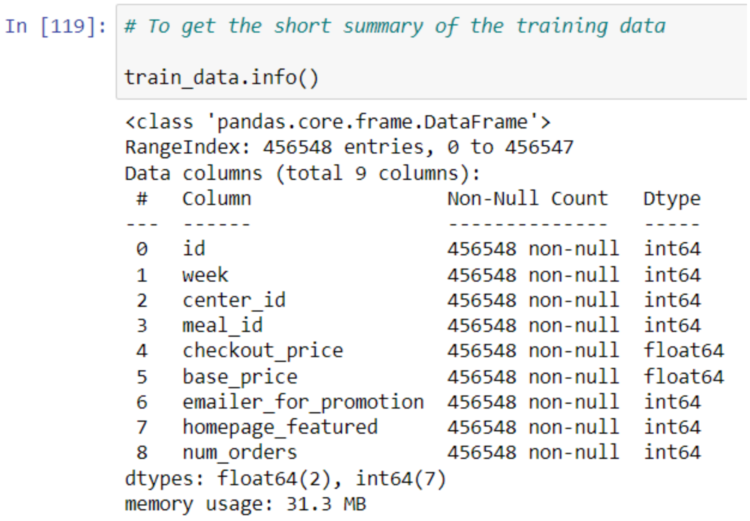
	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders
0	1379560	1	55	1885	136.83	152.29	0	0	177
1	1466964	1	55	1993	136.83	135.83	0	0	270
2	1346989	1	55	2539	134.86	135.86	0	0	189
3	1338232	1	55	2139	339.50	437.53	0	0	54
4	1448490	1	55	2631	243.50	242.50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



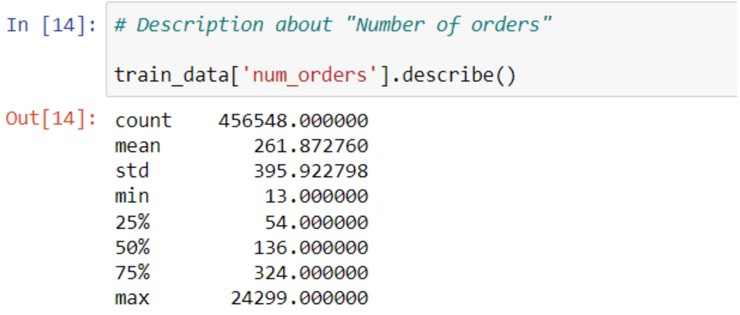
```
In [118]: test_data.head()
```

Out[118]:

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



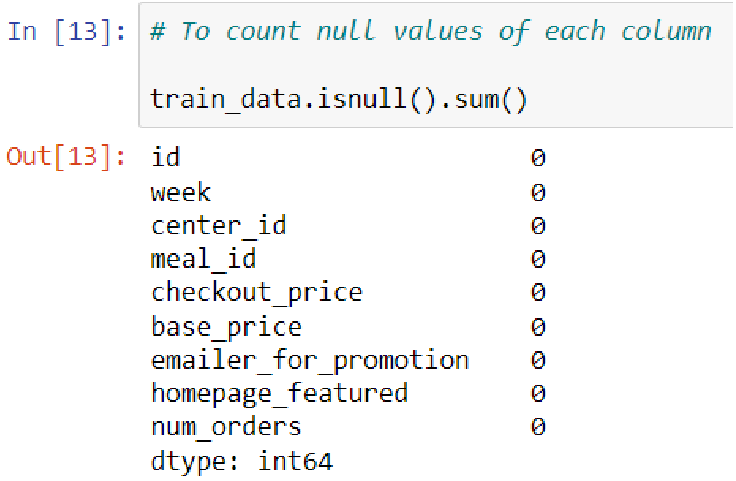
```
In [119]: # To get the short summary of the training data

          train_data.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
```



```
In [14]: # Description about "Number of orders"

         train_data['num_orders'].describe()

Out[14]: count    456548.000000
         mean        261.872760
         std         395.922798
         min          13.000000
         25%          54.000000
         50%         136.000000
         75%         324.000000
         max       24299.000000
```

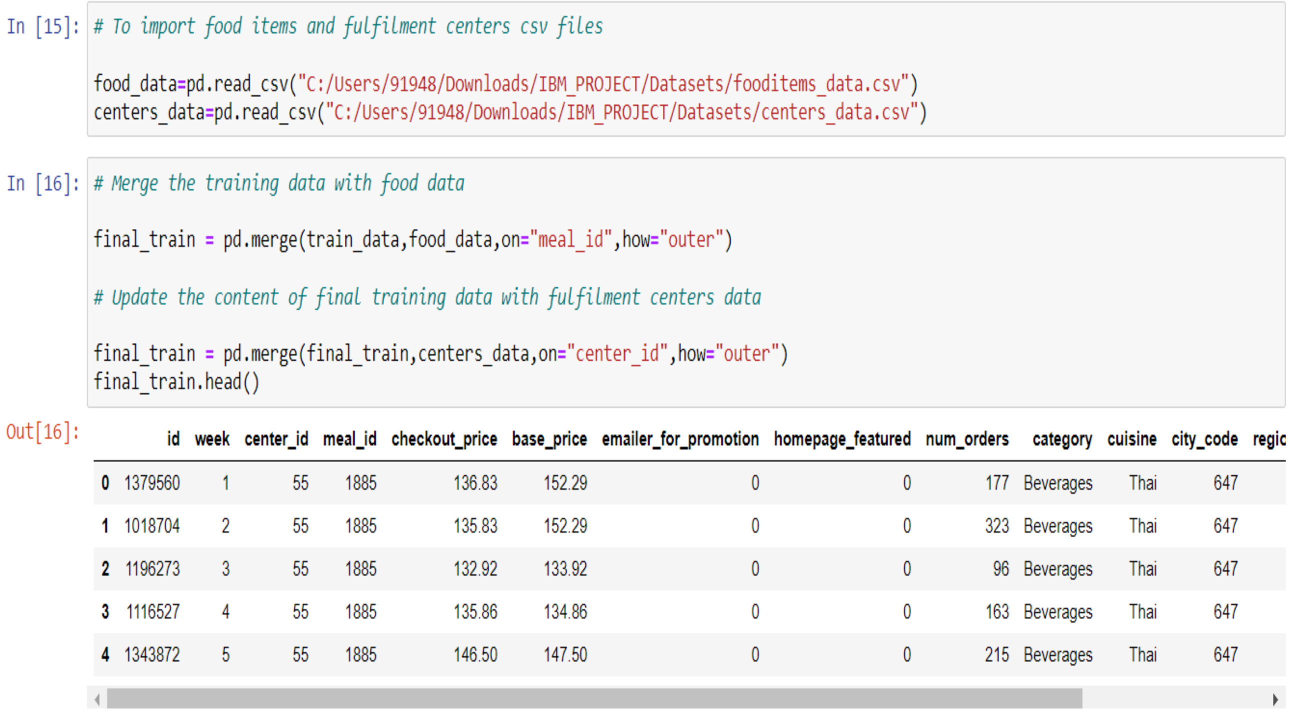
4. Checking for null values:

In real-life circumstances, missing data is a highly serious issue. Many datasets in Data frame occasionally come with blank rows, either because the data was obtained but not included or because it never existed. Use the function isnull() in a pandas Data frame to check for missing values .



```
In [13]: # To count null values of each column

         train_data.isnull().sum()

Out[13]: 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
```

5. Accessing and merging csv files:



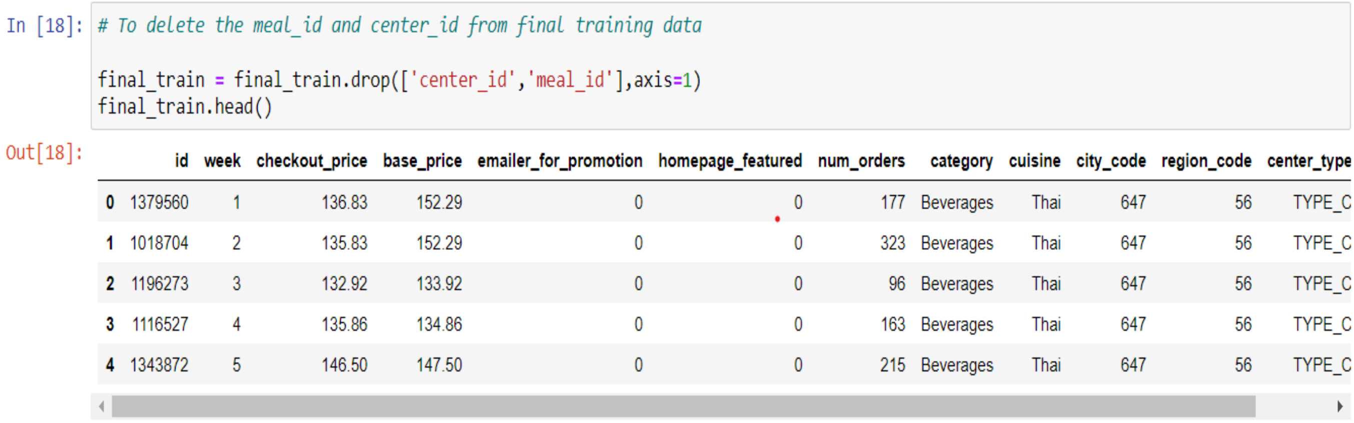
```
In [15]: # To import food items and fulfilment centers csv files

         food_data=pd.read_csv("C:/Users/91948/Downloads/IBM_PROJECT/Datasets/fooditems_data.csv")
         centers_data=pd.read_csv("C:/Users/91948/Downloads/IBM_PROJECT/Datasets/centers_data.csv")
```

```
In [16]: # Merge the training data with food data

         final_train = pd.merge(train_data,food_data,on="meal_id",how="outer")

         # Update the content of final training data with fulfilment centers data

         final_train = pd.merge(final_train,centers_data,on="center_id",how="outer")
         final_train.head()
```

Out[16]:

	id	week	center_id	meal_id	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders	category	cuisine	city_code	regic
0	1379560	1	55	1885	136.83	152.29	0	0	177	Beverages	Thai	647	
1	1018704	2	55	1885	135.83	152.29	0	0	323	Beverages	Thai	647	
2	1196273	3	55	1885	132.92	133.92	0	0	96	Beverages	Thai	647	
3	1116527	4	55	1885	135.86	134.86	0	0	163	Beverages	Thai	647	
4	1343872	5	55	1885	146.50	147.50	0	0	215	Beverages	Thai	647	

6. Drop columns from the dataset:



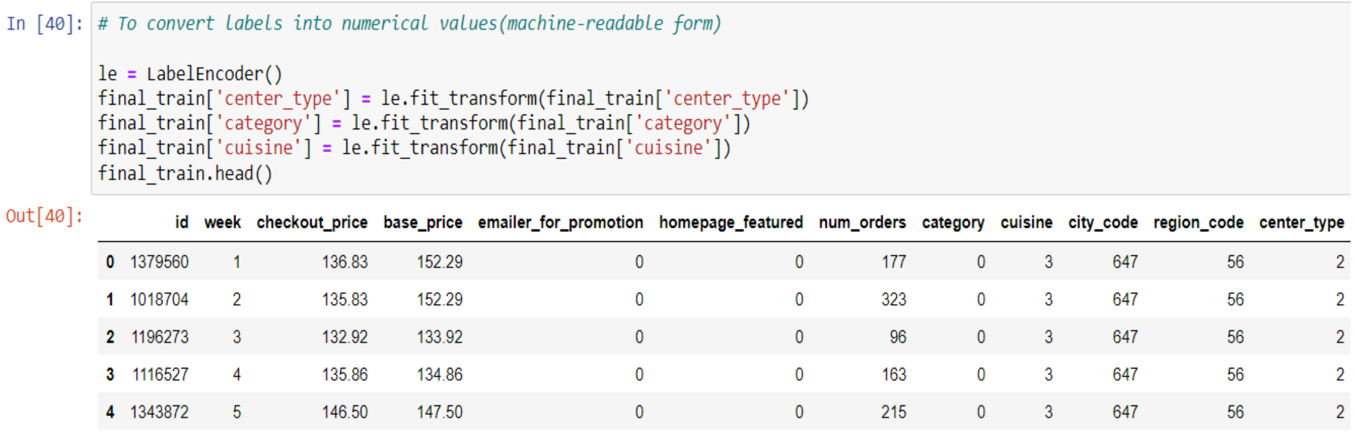
```
In [18]: # To delete the meal_id and center_id from final training data

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

Out[18]:

	id	week	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders	category	cuisine	city_code	region_code	center_type
0	1379560	1	136.83	152.29	0	0	177	Beverages	Thai	647	56	TYPE_C
1	1018704	2	135.83	152.29	0	0	323	Beverages	Thai	647	56	TYPE_C
2	1196273	3	132.92	133.92	0	0	96	Beverages	Thai	647	56	TYPE_C
3	1116527	4	135.86	134.86	0	0	163	Beverages	Thai	647	56	TYPE_C
4	1343872	5	146.50	147.50	0	0	215	Beverages	Thai	647	56	TYPE_C

7. Label Encoding:

Label encoding is the process of transforming labels into a numeric form by sklearn library so that they may be read by machines. The operation of those labels can then be better determined by machine learning techniques. It is a significant supervised learning pre-processing step for the structured dataset.

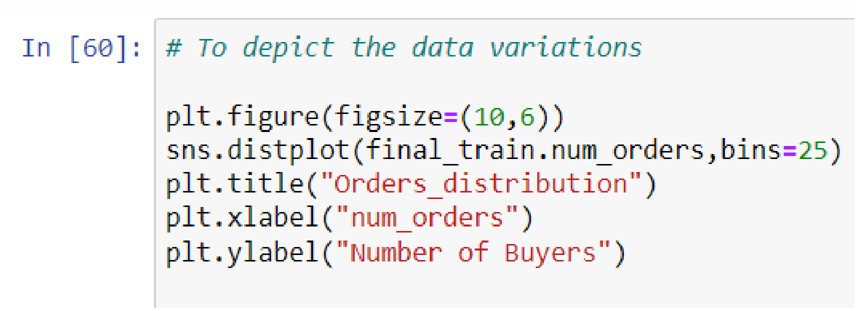


```
In [40]: # To convert labels into numerical values(machine-readable form)

         le = LabelEncoder()
         final_train['center_type'] = le.fit_transform(final_train['center_type'])
         final_train['category'] = le.fit_transform(final_train['category'])
         final_train['cuisine'] = le.fit_transform(final_train['cuisine'])
         final_train.head()
```

Out[40]:

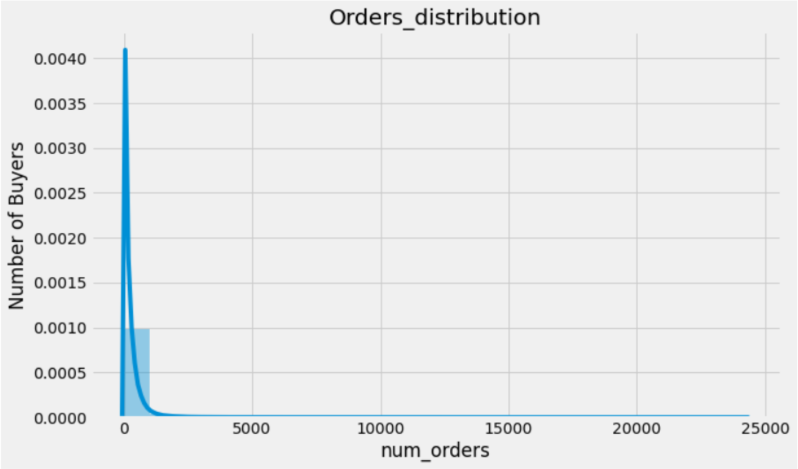
	id	week	checkout_price	base_price	emailer_for_promotion	homepage_featured	num_orders	category	cuisine	city_code	region_code	center_type
0	1379560	1	136.83	152.29	0	0	177	0	3	647	56	2
1	1018704	2	135.83	152.29	0	0	323	0	3	647	56	2
2	1196273	3	132.92	133.92	0	0	96	0	3	647	56	2
3	1116527	4	135.86	134.86	0	0	163	0	3	647	56	2
4	1343872	5	146.50	147.50	0	0	215	0	3	647	56	2

8. Data Visualization:



```
In [60]: # To depict the data variations

         plt.figure(figsize=(10,6))
         sns.distplot(final_train.num_orders,bins=25)
         plt.title("Orders_distribution")
         plt.xlabel("num_orders")
         plt.ylabel("Number of Buyers")
```

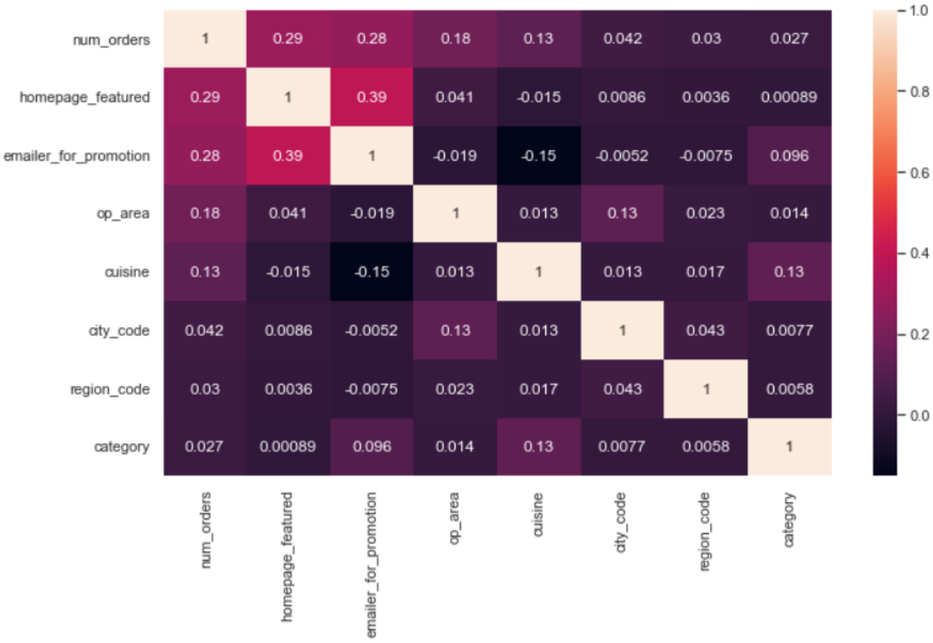
```
Out[60]: Text(0, 0.5, 'Number of Buyers')
```



```
In [62]: final_train2 = final_train.drop(['id'],axis=1)

         # To find the correlation of columns

         correlation = final_train2.corr(method='pearson')
         cols = correlation.nlargest(8,'num_orders').index
         cols
```

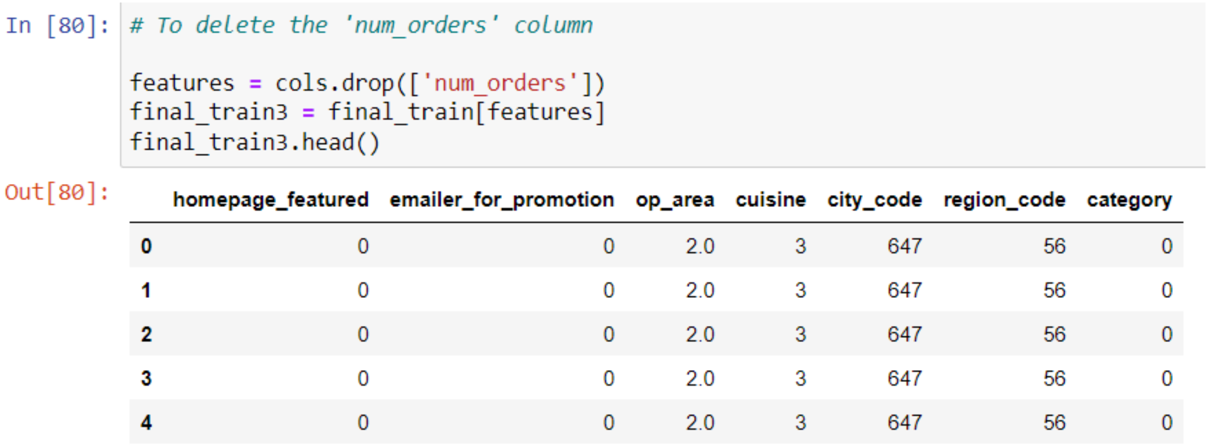
```
Out[62]: Index(['num_orders', 'homepage_featured', 'emailer_for_promotion', 'op_area',
                'cuisine', 'city_code', 'region_code', 'category'],
               dtype='object')
```

```
In [74]: correlation_mat = np.corrcoef(final_train2[cols].values.T)

         # To visualize how well features correlate with each other

         plt.figure(figsize=(10,6))
         Heat_map = sns.heatmap(correlation_mat,annot=True,yticklabels=cols.values,xticklabels=cols.values)
         plt.show()
```



9. Splitting the dataset into dependent and independent variables:

In the dataset, "X" would be the independent variable, and the columns "homepage featured," "emailer for promotion," "op area," "cuisine," "city code," "region code," and "category" would be the independent variables. In the dataset, "Y" would be regarded as the dependent variable, and the "num_orders" column would be regarded as the dependent variable.



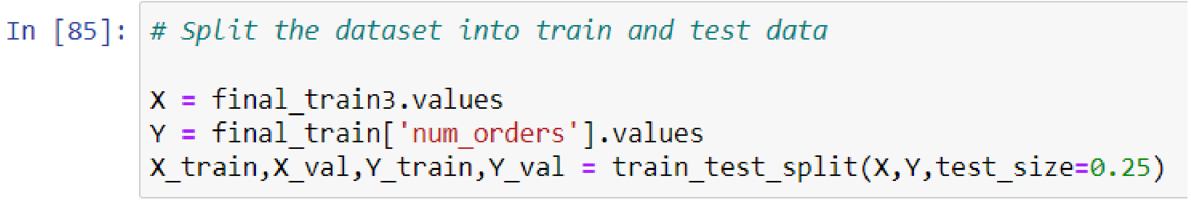
```
In [80]: # To delete the 'num_orders' column

         features = cols.drop(['num_orders'])
         final_train3 = final_train[features]
         final_train3.head()
```

Out[80]:

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

10. Split the dataset into training and testing data:

There must be a dataset available while working on a model and attempting to train it. The model must be tested on a test dataset after training, though. A dataset that differs from the training set you previously used will be needed for this. It might not always be practical, though, to have so much data available during the development stage. Divide the dataset into two sets, one for training and the other for testing, as a solution in such circumstances.



```
In [85]: # Split the dataset into train and test data

         X = final_train3.values
         Y = final_train['num_orders'].values
         X_train,X_val,Y_train,Y_val = train_test_split(X,Y,test_size=0.25)
```