

Demand forecasting for a store

Rajshree Avinash Adroja

40AIML437/21-1

thrajshree@gmail.com

Under the guidance of Mr. Manoj Kumar Susarla

Diploma in Artificial Intelligence and Machine Learning



University of
Hyderabad

Abstract

The food and beverage industry is one of the most important sectors of every countries economy. Forecasting the demand of meals is rather a challenging task, especially nowadays where integration of online and physical store orders creates an abundance of data that has to be efficiently stored, analyzed, understood and finally, become ready to be acted upon in a very short time frame. This project addresses the demand forecasting of various meals in different meal delivery stores of different regions with the help of various Machine Learning algorithms including a web application where a user can predict demand of meals for upcoming weeks.

Contents

Introduction	3
Chapter I: Literature Survey and Data Sources	
1. Problem statement	5
1.1 Importance of demand forecasting	5
1.2 Business/Real world impact of solving this problem	6
2. Dataset	7
2.1 An overview of dataset	7
2.2 Variables	7
3. Business metric	10
4. Real world challenges and constraints	11
5. Literature survey	12
Chapter II: Exploratory Data Analysis and Feature Extraction	
1. Univariate and Bivariate Feature Analysis	14
2. Multivariate Feature analysis	19
Chapter III: Modeling and Error Analysis	
1. Feature encoding	20
2. Models	20
2.1 Baseline model	20
2.2 KNN regression	21
2.3 Linear regression	23
2.4 Ridge regression	24
2.5 Lasso regression	26
2.6 Elasticnet regression	28
2.7 SGD regression	30
2.8 Decision tree regression	32
2.9 XGBoost regression	34
3. Deep dive into best model	36
Chapter IV: Advanced Modeling and Feature Engineering	
1. Hyper-parameter tuning	40
2. Feature engineering	44
3. Fine tuning	45
Chapter V: Deployment and Productionization	
1. Amazon Web Service and Streamlit	47
2. Throughput and latency	48
3. Limitations	49
Conclusion.....	50
Bibliography.....	51

Introduction

“It is best to read the weather forecast before praying for rain.”~Mark Twain.

Forecasting has always played a crucial role in the human civilization. In simple English forecasting means “anticipation”. We anticipate in every aspect of our life. It is this anticipation that drives us forward. Without hope life would be like a boat without a crew in the middle of the ocean.

Who doesn't want a satisfied job? Who doesn't want successful career? Who doesn't want to own a million dollar business? How would you feel when someone comes and show you how your business would be like after 1 or 3 years from now, that too with a prove? Famous English biologist Charles Darwin has proved “Survival for the fittest”. Therefore in order to survive (or be successful) in this highly competitive world we have to make ourselves fit into it.

In this technology driven era, Internet of things has provided a wide platform to improve in every field of life. Digitization has made so easy and effective to store data in every organization. And instant revolution in the form of Machine Learning, Deep Learning and Artificial Intelligence has provided an effective way to deal with this data and any other forms of problem while using the modern techniques without any hesitation; acting like a cherry on a piece of cake.

Yes of course, Food Industry has also benefited equally like any other industries. In the past few years, instantaneous growing demand has created an alarming situation to restaurants to maintain the quality of food and services which in turn affect the valuable aspect of the restaurant to retain their customers for longer period. However with the utilization of resources on internet and techniques, several restaurants have created enormous amount of profit margin.

The word “Forecasting” is not newly coined; human has been living with it, however with the advent of the said techniques, prediction of any form of business has become so easier and faster like never before in the history. Credits to computers for its never tiring feature. Forecasting means making predictions based on the past and present data.

Forecasting is prediction or an estimation of an actual value in a future time period or for another situation. It is a technique that uses historical data as inputs to make informed estimates that are predictive in determining the direction of the future trends. Businesses use forecasting to determine how to allocate their budgets and resources or plan for anticipated expenses for an upcoming period of time.

Demand forecasting is, in essence, developing the best possible understanding of future demand. It means analyzing the impact of range of variables that affect demand - from historical demand patterns to internal business decisions or even external factors - to increase the accuracy of prediction.

Demand forecasts can be developed on different levels of granularity- monthly, weekly, daily, or even hourly – to support different planning processes and business decisions, but highly granular forecast are always extremely valuable. The benefits of granular forecast are obvious when it comes to fresh products whose short shelf-lives sometimes call for intra-day forecast at the product-location level to prevent spoilage.

Chapter I:

Literature Survey and Data Acquisition

1. Problem statement:

The problem is about a meal delivery company (client) that operates in multiple cities. They have various fulfillment centers in those cities for dispatching meal orders to their customers.

The client wants us to help these centers with demand forecasting for upcoming weeks so that these centers will plan the stock of raw materials accordingly.

The replenishment of the majority of raw materials is done on weekly basis and since the raw material is perishable, procurement planning is of utmost importance.

Secondly, staffing of the centers is also one area wherein accurate demand forecasts are really helpful.

Given the following information, the task is to predict the demand for the next 10 weeks (Weeks: 146-155) for the center-meal combinations in the test set:

- Historical data of demand for a product-center combination (Weeks: 1 to 145)
- Product (Meal) features such as category, sub-category, current price, and discount
- Information for fulfillment centers like center area, city information, etc.

1.1 Importance of demand forecasting:

Have the right products in stock: Efficient inventory management ensures business owner has the right products on hand exactly when your customers want them, but not so much that they go bad or become irrelevant.

By demand forecasting, one can prepare for supply chain and inventory for any expected peaks, as well as prevent excessive inventory that hurts the cash flow and increases storage costs.

Demand forecasting also helps in reduction of inventory, or food waste in the restaurant industry, especially when it comes to the perishable food with a low shelf-life.

Assess the risk of launching new products: The business owner accounts for the costs of manufacturers, suppliers, storage, and marketing of a new product. This can be a massive

gamble. Demand forecasting minimizes the risk that comes with sourcing a product that has never sold before.

Make smart staffing decisions: Having the right number of employees staffed to support customers shopping is a win for everyone. But just like missing the mark with product inventory, staffing mistakes can be costly. Considering historical sales data, demand forecasting helps a business plan staff schedule and thus can adjust throughout the day and cut shifts short if the store has scheduled more staff than necessary.

Run better marketing campaigns: Demand forecasting helps a business in understanding customer demand which in turn makes marketing efforts more intentional and cost-effective.

Budget accurately and maintain positive cash flow: With data-backed sales forecasts, the business owner can shift their budget as needed instead of jumping into panic mode when there is dramatically dipped in revenue.

1.2 Business/Real world impact of solving this problem:

Demand forecasting is critical to the future of any business as it helps reduce risks and make the right call on many fronts.

From omnichannel giants to the smallest brick and mortar boutiques, all retailers rely on demand forecasts to predict their best estimate of how much they will sell. Modern demand forecasting is a sophisticated statistical analysis that takes into account numerous variables to optimize that prediction.

While some retailers still rely on spreadsheets and manual calculations, such high-powered statistical analysis is best executed by specialized software designed to process enormous, retail-scale data sets. The perk of this technique is it transparently shows users what data is being used to build forecasts and how the forecasts are being calculated. Modern demand forecasting software automates difficult and time-consuming decisions, using machine learning to optimize predictions.

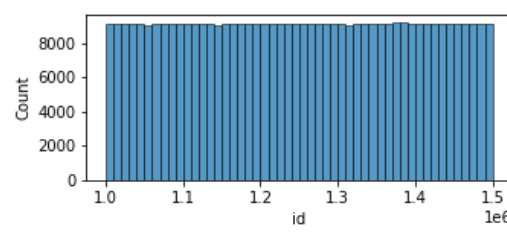
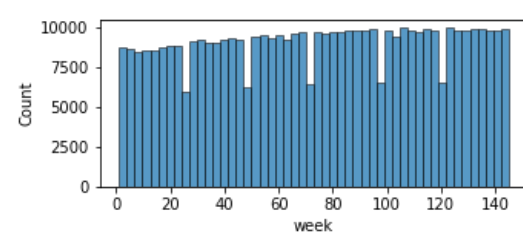
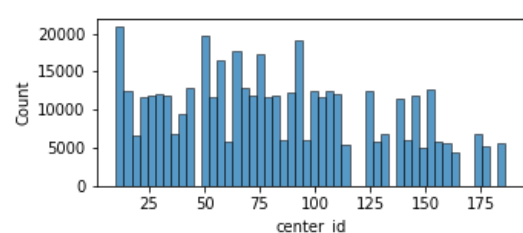
Though it will never be 100% accurate, forecasting demand can help improve production lead times, increase operational efficiencies, save money, launch new products, and provide a better customer experience overall.

2. Dataset:

2.1 An overview of dataset:

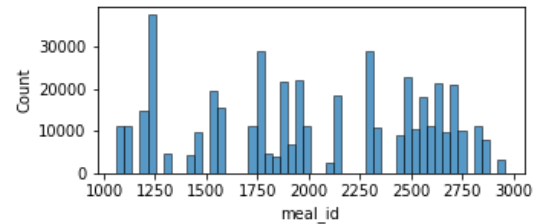
Source	Kaggle
Number of variables	15
Number of observations	456548
Missing cells	0
Duplicate rows	0
Total size of memory	55.7 MB
Numeric variables	10
Categorical variables	5

2.2 Variables:

<p>“id” - Real number</p> <p>Level: Nominal Distinct: 456548 Minimum: 1000000 Maximum: 1499999 Mean: 1250096.306</p>	
<p>“week” – Real number</p> <p>Level: Nominal Distinct: 145 Minimum: 1 Maximum: 145 Mean: 74.7687713</p>	
<p>“center_id” – Real number</p> <p>Level: Nominal Distinct: 77 Minimum: 10 Maximum: 186 Mean: 82.1057961</p>	

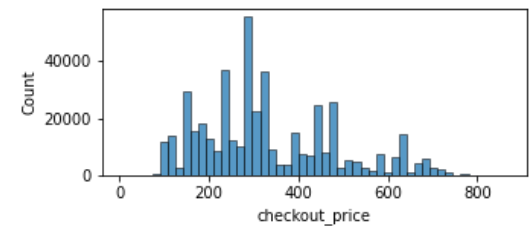
“meal_id” – Real number

Level: Nominal
Distinct: 51
Minimum: 1060
Maximum: 2956
Mean: 2024.337458



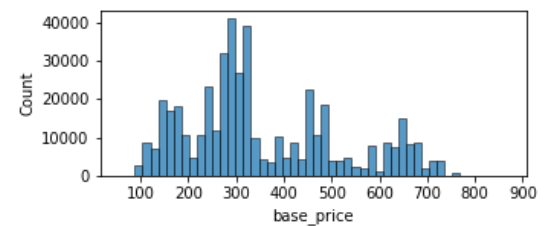
“checkout_price”- Real number

Level: Nominal
Distinct: 1992
Minimum: 2.97
Maximum: 866.27
Mean: 332.2389326



“base_price” – Real number

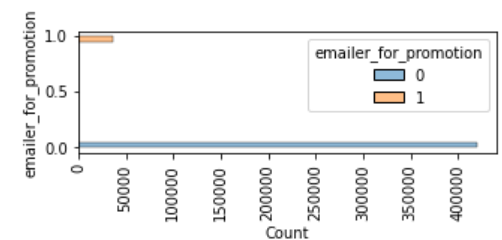
Level: Nominal
Distinct: 1907
Minimum: 55.35
Maximum: 866.27
Mean: 354.1566275



“emailer_for_promotion” – Categorical

Level: Nominal
Distinct: 2

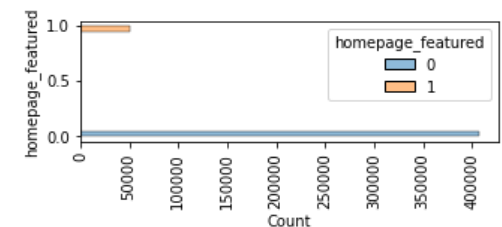
Value	Count	Frequency
0	419498	91.9%
1	37050	8.1%



“homepage_featured” – Categorical

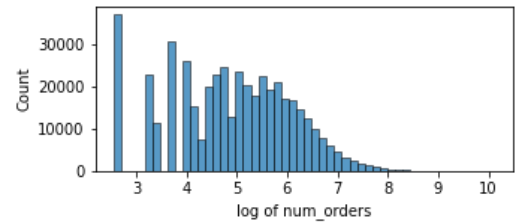
Level: Nominal
Distinct: 2

Value	Count	Frequency
0	406693	89.1%
1	49855	10.9%



“num_orders” – Real number

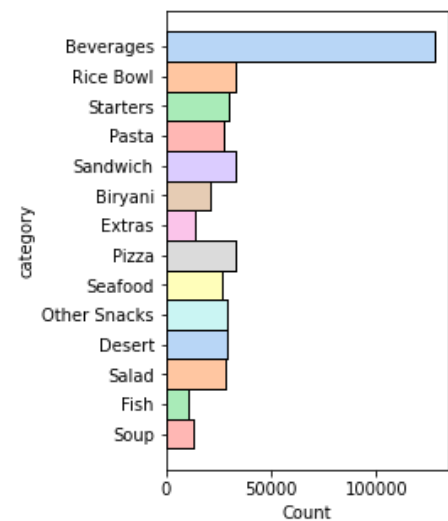
Level: Nominal
Distinct: 1250
Minimum: 13
Maximum: 24299
Mean: 261.8727604



“category” – Categorical

Level: Nominal
Distinct: 14

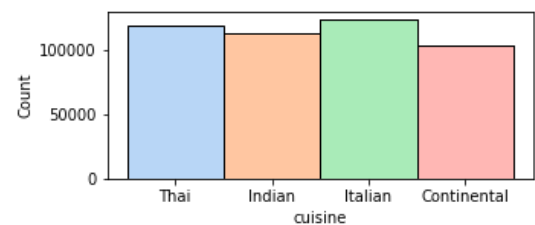
Value	Count	Frequency
Beverage	127890	28%
Rice Bowl	33408	7.3%
Sandwich	33291	7.3%
Pizza	33138	7.3%
Starters	29941	6.6%
O. Snacks	29941	6.4%
Desert	29294	6.4%
Salad	28559	6.3%
Pasta	27694	6.1%
Seafood	26916	6.1%
Biryani	20614	4%
Extras	13562	2%
Soup	12675	2%
Fish	10187	2%



“cuisine” – Categorical

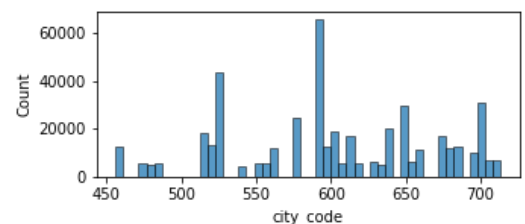
Level: Nominal
Distinct: 14

Value	Count	Frequency
Italian	122925	26.9%
Thai	118216	25.9%
Indian	112612	24.7%
Continental	102795	22.5%



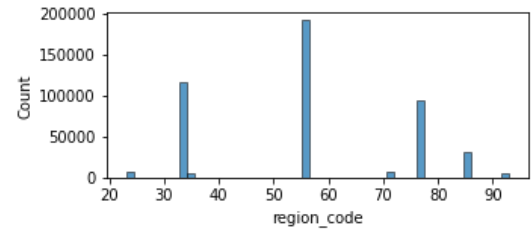
“city_code” – Real number

Level: Nominal
Distinct: 51
Minimum: 456
Maximum: 713
Mean: 601.5533985



“region_code” – Real number

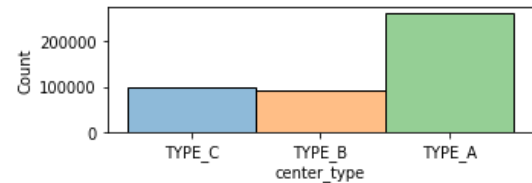
Level: Nominal
Distinct: 8
Minimum: 23
Maximum: 93
Mean: 56.61456627



“center_type” – Categorical

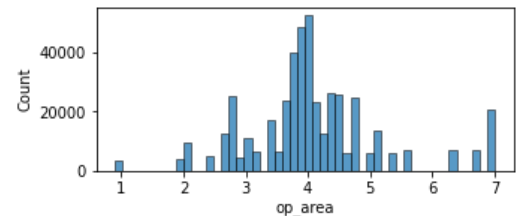
Level: Nominal
Distinct: 8

Value	Count	Frequency
TYPE_A	262881	57.6%
TYPE_B	99593	21.8%
TYPE_C	94074	20.6%



“op_area” – Real number

Level: Nominal
Distinct: 30
Minimum: 0.9
Maximum: 7
Mean: 4.083589896



3. Business Metric:

A business metric is a quantifiable measure that is used to track and assess the status of a specific business process. The metric to be used in this project is RMSE (Root Mean Square Value). It is the square root of the average squared error and its formula is

$$RMSE = \sqrt{\frac{1}{n} \sum e_t^2}$$

where, error, $e = (y_i - \hat{y}_i)$, actual value = y_i , predicted value = \hat{y}_i

The Root Mean Square Error represents the standard deviation of the error (residual error). It is always positive, and a lower value indicates better performance. The ideal value would be 0 but it is never achieved.

RMSE is it penalizes large errors. It results in the same unit as the outcome variables which is smoothly differentiable and makes it easier to perform mathematical operations. RMSE is highly sensitive to outliers hence it should not be used prior to removal of outliers.

RMSE is the most commonly used metric in modern ML and Data science problems, for example: in meteorology, bioinformatics, economics, hydrogeology, imaging science, computational neuroscience, submission for the Netflix prize, simulation of energy consumption of buildings.

And to see how well a model is for a dataset, Coefficient of determination or R-squared (R^2), is being used in this project. R^2 lies between 0.0 to 1.0. A value of 1.0 indicates a perfect fit, and is thus a highly reliable model for future forecasts, while a value of 0.0 would indicate that the calculation fails to accurately model the data at all.

But a value of 0.20, for example, suggests that 20% of the dependent variable is predicted by the independent variable, while a value of 0.50 suggests that 50% of the dependent variable is predicted by the independent variable, and so forth. It is also known as “Goodness of fit” and its formula is

$$R^2 = 1 - \frac{SSR}{SST} = 1 - \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2}$$

Where, Sum Square of Residuals, $SSR = (\hat{y}_i - \bar{y})^2$,

Sum Square of Total, $SST = (y_i - \bar{y})^2$,

actual value = y_i , average value \bar{y} , predicted value = \hat{y}_i

4. Real world challenges and constraints:

Seasonality: It implies there are periods of the year when the demand for a product becomes higher than other times, and that it's a pattern that repeats every year.

The drivers of seasonality include factors like:

- Time: hour, day of the week, week, month
- Holidays: public, festivals and school holidays (like 15th of August or Diwali)
- Weather: temperature, rainfall, snowfall
- Events: hockey, cricket, wedding or festival season)

Competitors: When a competitive force comes into play — whether it's a direct competitor or a new kind of solution that forces our customers to choose between us or them — demand forecast will be skewed. So an agile demand forecasting model can help us respond quickly without any surprise.

Advertisements, Promotions and discounts: Demand would vary based on the advertisements and promotions either on web page or any public media or social media. Seasonality discounts can greatly affects the demand.

Types of goods: Demand forecasting will be very different for different products and services — from perishable goods that expire quickly to subscription boxes that come at the same time each month.

It's important to know the lifetime value of customers (the total purchases they buy from across channels over time), our average order value (how much they're spending each time), and the combinations of products ordered to improve demand forecasting.

Location type: Where the retail store is located plays a great role in demand forecasting. It may be in a street or shopping malls. The average age of customers in a particular location may also affect the demand of products.

The geography of where our customers reside and where we manufacture and ship orders from can greatly impact inventory forecasting and the speed at which we can fulfill customer orders.

World events: The year 2020 and the pandemic flipped retail and ecommerce on their respective heads. No one could have predicted the outcomes.

People changed their shopping habits in many ways, including how often they visit stores and where they go to find new products.

Customer spending on luxury items took a dip, but the athleisure market saw massive growth.

We can't predict a global pandemic and similar world event, but we can commit to understanding how they impact our customers.

5. Literature survey:

There are no “one-size-fits-all” forecasting algorithms. Often, demand forecasting features consist of several machine learning approaches. The choice of machine learning models depends on several factors, such as business goal, data type, data amount and quality, forecasting period, etc.

Here are some of the existing machines learning methodologies:

- 1) **Time-series approach:** This involves processed data points that occur over a specific time that are used to predict the future. Time series is a sequence of data points taken at successive, equally-spaced points in time. The major components to analyze are: trends, seasonality, irregularity, cyclicity.

The analysis algorithm involves the use of historical data to forecast future demand. That historical data includes trends, cyclical fluctuations, seasonality, and behavior patterns.

Below are the most applicable time series models:

- a) **ARIMA** (auto-regressive integrated moving average) models aim to describe the auto-correlations in the time series data. When planning short-term forecasts, ARIMA can make

accurate predictions. By providing forecasted values for user-specified periods, it clearly shows results for demand, sales, planning, and production.

- b) **SARIMA** (Seasonal Autoregressive Integrated Moving Average) models are the extension of the ARIMA model that supports uni-variate time series data involving backshifts of the seasonal period.
 - c) Exponential Smoothing models generate forecasts by using weighted averages of past observations to predict new values. The essence of these models is in combining Error, Trend, and Seasonal components into a smooth calculation.
- 2) **Linear regression approach:** It is a statistical method for predicting future values from past values. It can help determine underlying trends and deal with cases involving overstated prices.

This regression type allows to:

- Predict trends and future values through data point estimates.
 - Forecast impacts of changes and identify the strength of the effects by analyzing dependent and independent variables.
- 3) **Featured Engineering:** Feature engineering is the use of domain knowledge data and the creation of features that make machine learning models predict more accurately. It enables a deeper understanding of data and more valuable insights.

Since feature engineering is creating new features according to business goals, this approach is applicable in any situation where standard methods fail to add value.

- 4) **Decision tree:** Decision tree builds regression or classification models in the form of trees. It breaks down the Random forest is the more advanced approach that makes multiple decision trees and merges them together. By taking an average of all individual decision tree estimates, the random forest model results in more reliable forecasts.

Random forest can be used for both classification and regression tasks, but it also has limitations. The model may be too slow for real-time predictions when analyzing a large number of trees.

Chapter II: Exploratory Data Analysis and Feature Extraction

1. Univariate and Bivariate Feature Analysis:

Histograms, distribution plots, scatter plots, line plots, boxplots are some of the features analysis used to understand insights of the data. These are the most powerful strategies to come up with beautiful insights of the data, especially when it comes with the relation between two variables.

However, it fails to give complete information on which variable has the most important feature to give highest accurate prediction.

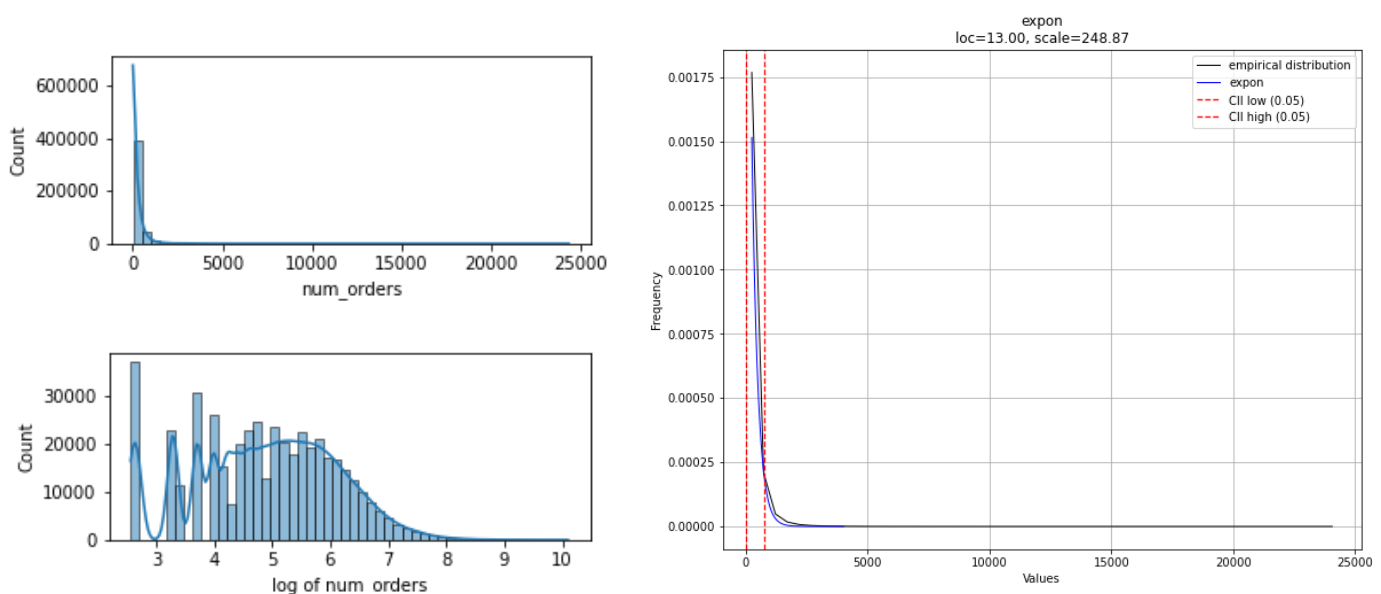
We plotted some of these to know more about the behavior of our data.

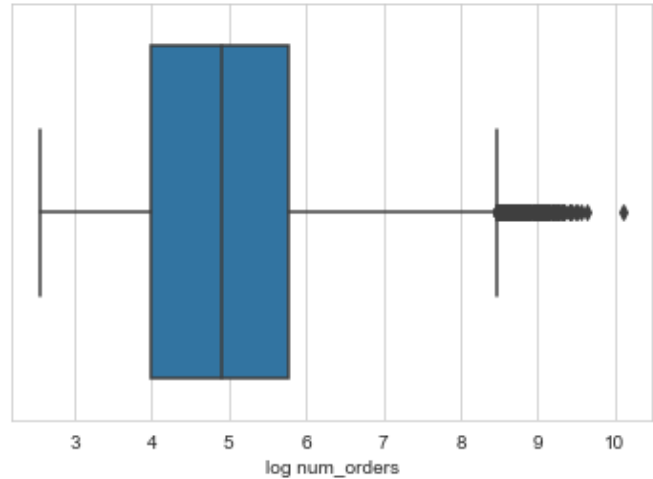
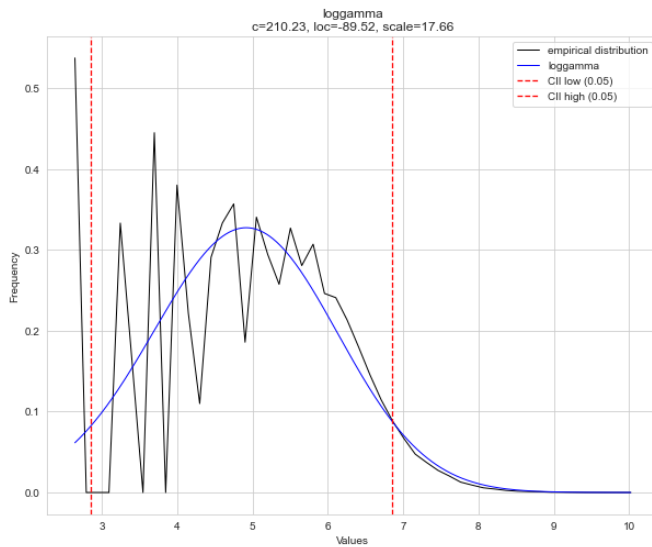
The pattern of number of orders follows exponential distribution and can be seen from the graph below along with its histogram of “log of num_orders” with its kernel density estimation (kde).

Log is taken to handle the wide range of data (13 to 24299). It is multimodal as it has more than one peak point. It is right skewed with a value of 6.929966065.

The higher number of orders is concentrated in between 1 to 6. The plots below depict the difference on the histogram of num_orders with and without log.

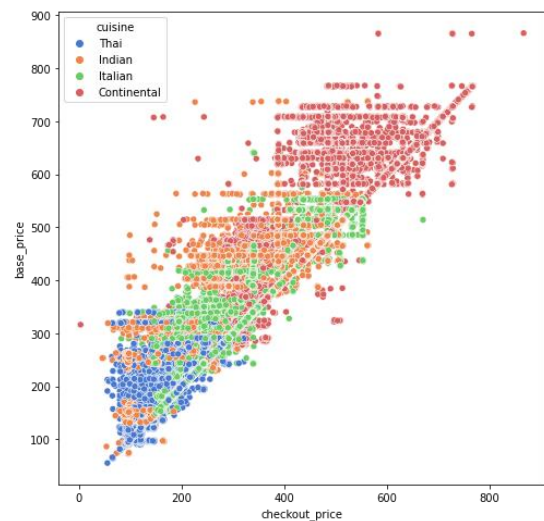
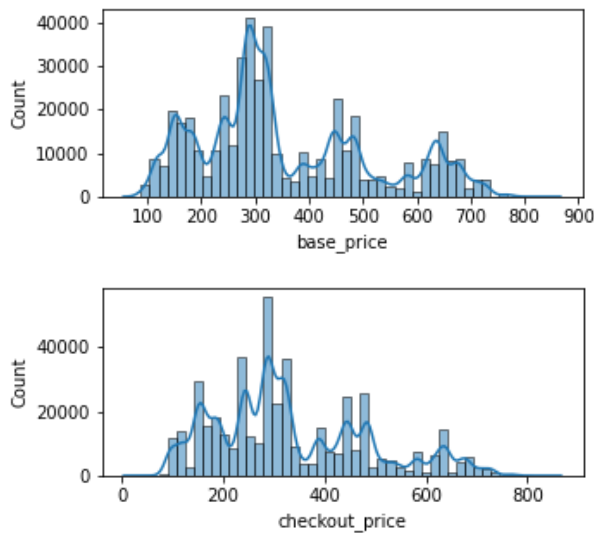
The log of num_orders follows log gamma distribution and from the boxplot of the log of num_orders, it is seen that there are some outliers that needs to be removed.



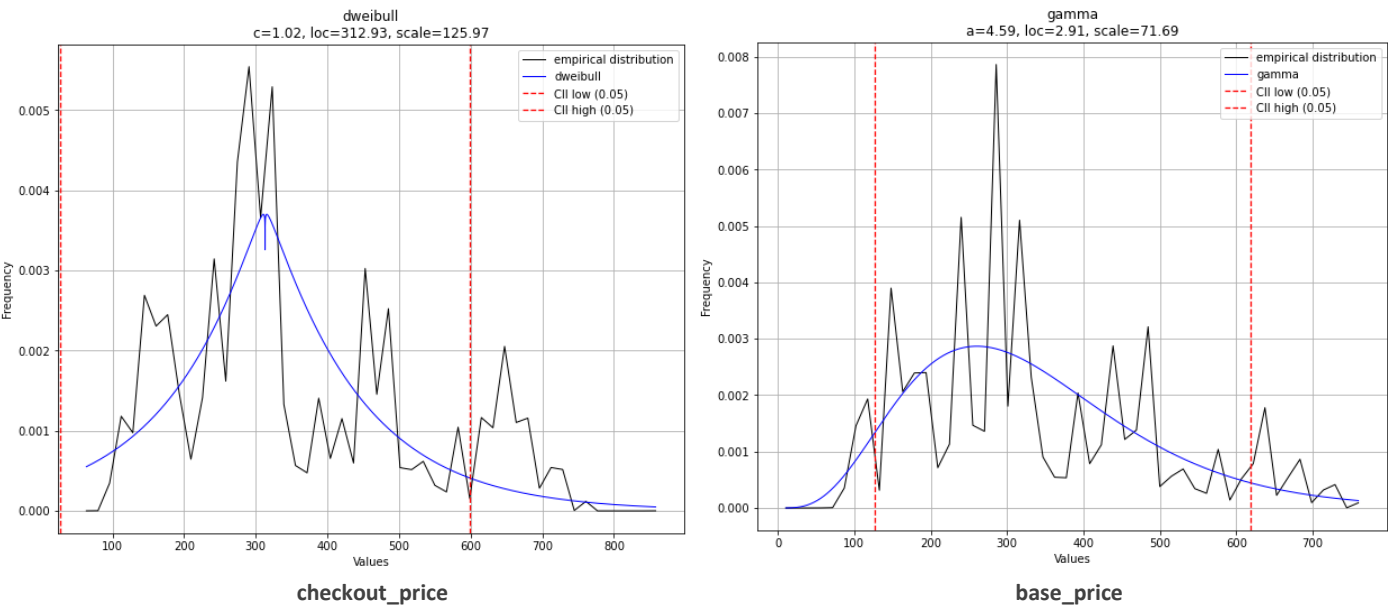


From the histogram of base_price and checkout_price below, it is found that they both are in the range of 100 to 800 with 300 as the highly concentrated range.

As the scatter plot between base_price and checkout_price depicts they have linear relation where majority of the base_price are on the higher side although there are also some points with higher checkout_price than its base_price. This means that there is more number of discounts.

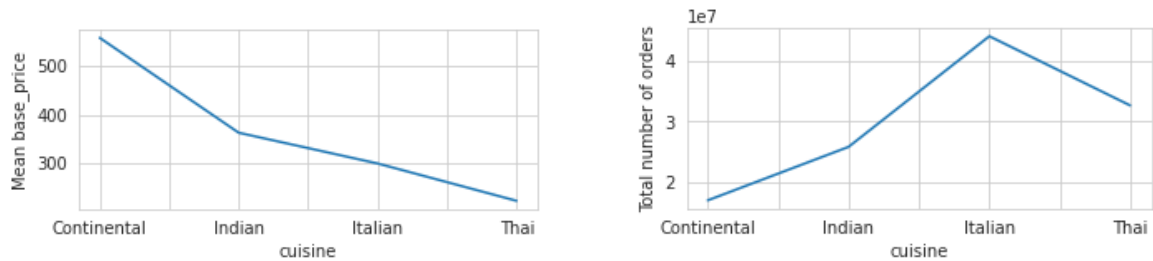


The graphs below show the distribution of checkout_price and base_price. Checkout_price follows Dweibull distribution while base_price follows Gamma distribution.

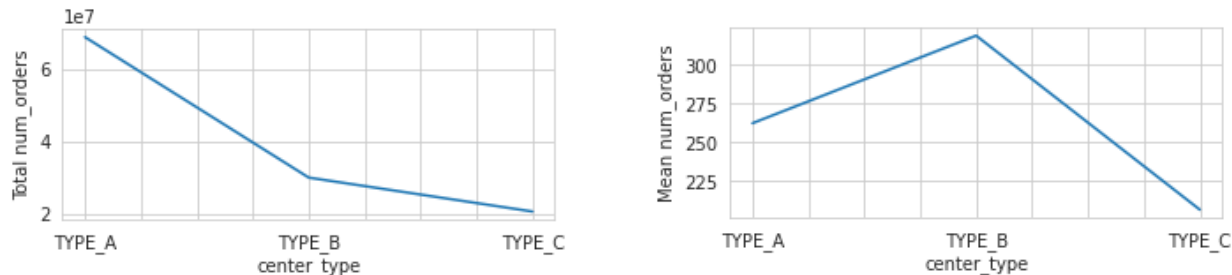


The cuisine type “Continental” has higher base_price then other 3 cuisines, “Thai”, “Indian” and Italian” are at a regional range.

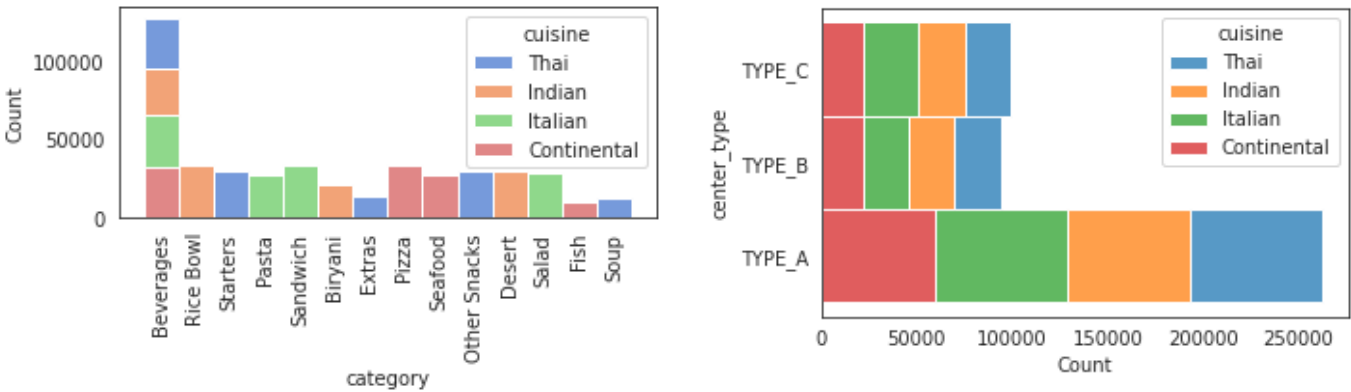
And from the line plots below it is clear that cuisines with higher_price have less number of orders. This applies to variable “category” as well.



Although total number of orders is high in center_type A, average number of orders is highest in center_type B. Center_type A has more number of centers.

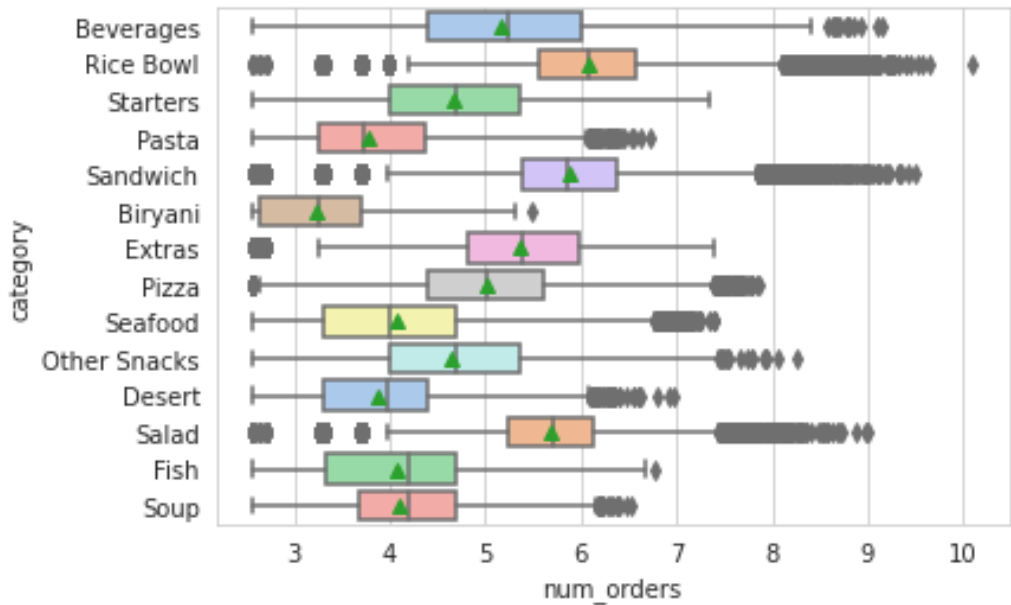


There are 14 meals and 4 cuisines in the given dataset. From the below histogram of variable “category”, “Beverages” can be seen in all the “cuisine”, consequently having the highest number of orders compare to other meals.

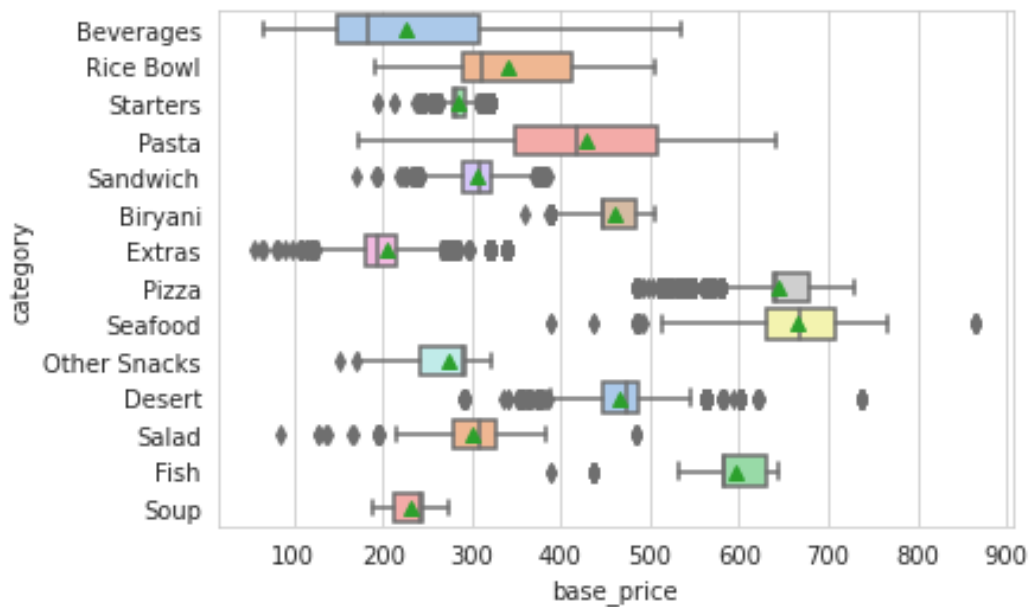


There are 3 center types given in the dataset. From the above histogram of variable “center_type” color encoded with “cuisine”, it is seen that there is equal distributions of all cuisines in all types of center.

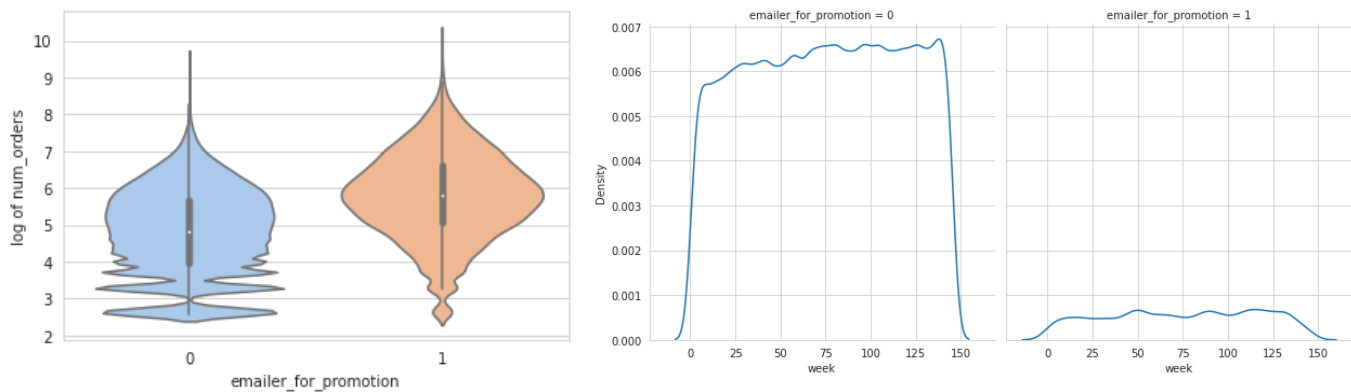
From the below boxplot of variable “category” with the number of orders, it is found that “Rice Bowl”, “Sandwich” and “Salad” has higher outliers and mean number of orders compare to other meals, while “Biryani” has the lowest mean order value.



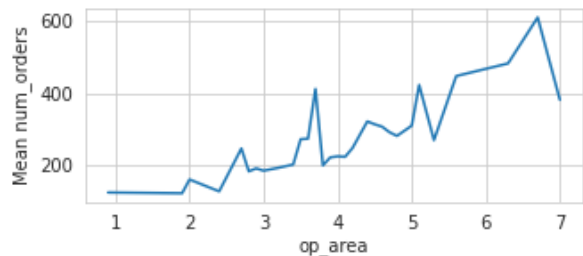
“Extras”, “Beverages” and “Soup” has lower base_price while “Pizza”, “Sandwich” and “Fish” are on the higher range as shown in the boxplot below.



From the distribution and violin plots below, it can be seen that “emailer_for_promotion” with value “1” has a smooth distribution curve compare to the one with value “0”, even though it was lasted for very less number of weeks. Likewise, variable “homepage_featured” with value “1” has same effect on the number of orders.



2.11. The number of meal orders is roughly increasing with the op_area as shown is the graph below.

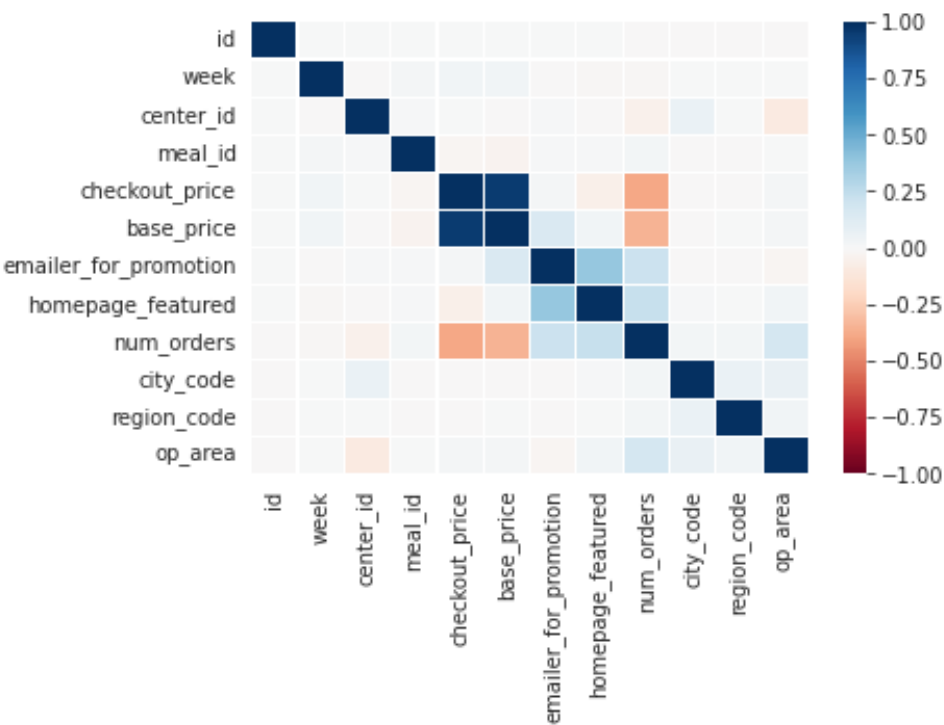


2. Multivariate Feature analysis:

Finding correlation coefficients is one of the most effective multivariate analyses. Pearson correlation is the most common one which gives measure of how numerical variables are correlated to each other. So as Spearman and Kendall coefficients.

However, Spearman correlation coefficient is more robust to outliers and handles sparse matrix more effectively. Below is a heatmap plot for spearman coefficient. It gives values between -1 to 1. 1 being linearly related, -1 being inversely related while 0 being no relations.

Checkout_price and base_price are closely related and inversely related to the num_orders. Emailer_for_promotion and homepage_featured are also closely related and to num_orders as well. Op_area is also somehow related to num_orders.



Chapter III:

Modeling and Error Analysis

1. Feature encoding:

Before we proceed to the modeling part, we went through some preprocessing methods to make the data readable to machine. One hot encoding which is one of the simplest techniques to handle categorical variables is being used. The technique encodes categorical variables into binary digits making input variables more understandable to machines. All the categorical variables are converted to each one column making sparsely high dimensional dataset.

“category”, “cuisine” and “center_type” columns are one hot encoded and “category”, “cuisine” and “center_type” columns are removed as they will not be needed anymore.

As it is a time series problem, whole dataframe is arranged in ascending order of “week” column and splits into train and test data with TimeSeriesSplit (n_splits=5). To begin with, we tried different Machine algorithms to check which algorithm fits the best to our dataset.

Here we have taken data for only 4 weeks because some of the algorithm might take forever to run the whole 456548 data. We have also done gridsearch hyper-parameter tuning to find the best hyper-parameter on each algorithm. After find the best algorithm we would train our whole data on it.

2. Models:

2.1 Baseline model:

As a simple baseline model, mean value of y_train is assumed to be predicted value. RMSE value and R2_score is being calculated as shown below. RMSE turns out to be 360.93 and R2_score with negative value. This shows that the baseline model is quite a bad model.

```
1 mean_train = np.mean(y_train)
2 y_baseline = np.ones(y_test.shape)*mean_train
3 rmse = np.sqrt(mean_squared_error(y_test,y_baseline))
4 score = r2_score(y_test,y_baseline)
5 print("RMSE: %.3f"%rmse)
6 print(" R2:  %0.3f"%score)
```

```
RMSE: 360.930
R2: -0.003
```

2.2 KNN regression:

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

Here, distance measured is minkowski by giving more weight to the closer neighbors of the query points we tried for different k values (1 to 20). 16 came out to be the best k value with RMSE value of 289.829. Codes below shows how we have implemented kNN algorithm and plotted RMSE vs. K graph for.

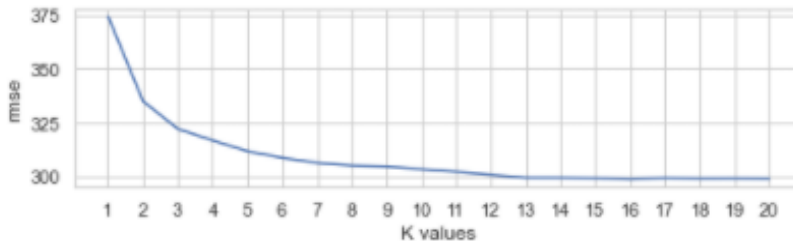
```
1 # Initialising range of values to be use as hyperparameter k
2 s = time.time()
3 # Initialising variable a and b
4 a, b = 1, 21
5 # Initialising range of k values to be calculated
6 k_value = [i for i in range(a,b)]
7 # Initialising List to collect all the metrices calculated for every k
8 rmse, r2 = [], []
9 # For every each value of k ranging from 1-20, train, predict and calculate metrices
10 for k in k_value:
11     # Instantiating knn regression function
12     knn = KNeighborsRegressor(n_neighbors=k, weights="distance", p=1)
13     knn.fit(x_train, y_train)
14     # Predict using knn regressor
15     y_knn = knn.predict(x_test)
16     accu = round(knn.score(x_test,y_test),3)
17     r2.append(accu)
18     # Calculating error by calling calculate_error function
19     err = calculate_error(y_test,y_knn)
20     rmse.append(err)
21 # Storing all the above results to a dataframe
22 result = pd.DataFrame(list(zip(k_value, rmse, r2)))
23 # Finding index of the maximum value of r2
24 # Maximum value of r2 indicates highest accuracy which gives best k
25 idx = int(result[[2]].idxmax())
26 # Best value of k and its corresponding performance metrices are stored separately
27 best_k, knn_rmse, knn_r2 = result.loc[idx].values.tolist()
28 print("Best k:", best_k)
29 print("RMSE:", knn_rmse)
30 print("R2:", knn_r2)
31 e = time.time()
32 print(round(e-s,2),"sec")
```

```
Best k: 16.0
RMSE: 298.829
R2: 0.039
21.05 sec
```

```

1 # Plotting the rmse values against k values
2 # best k value would show the lowest rmse value
3 plt.figure(figsize=(8,2)) # Setting size of graph
4 sb.set_theme(style="whitegrid") # Setting theme of graph
5 plt.plot(k_value, rmse) # Plotting k_values and rmse on x and y axes respectively
6 plt.xticks(ticks=k_value) # Setting ticks on x axis to k_value
7 plt.xlabel("K values") # Labelling x axis
8 plt.ylabel("rmse") # Labelling y axis
9 plt.show()

```



Advantages	Disadvantages
<ol style="list-style-type: none"> 1) Instance based learning hence no explicit training period. 2) Very simple to understand, easy to implement and no assumptions needed as it is a non-parametric algorithm. 3) Can be used for both classification and regression problem. 	<ol style="list-style-type: none"> 1) Efficiency and speed of algorithm declines very fast as data grows. 2) Difficult to calculate distances for high dimensions. 3) Very sensitive to missing values and outliers,
Applications	
<ol style="list-style-type: none"> 1) K-NN is one of the most popular algorithms for text categorization or text mining. 2) K-NN method in agriculture includes climate forecasting and estimating soil water parameters. 3) Other applications of KNN in finance includes: <ol style="list-style-type: none"> a. Forecasting stock market: Predict the price of a stock, on the basis of company performance measures and economic data. b. Currency exchange rate c. Bank bankruptcies d. Understanding and managing financial risk e. Trading futures f. Credit rating g. Loan management h. Bank customer profiling i. Money laundering analyses 4) Applications of K_NN in medical sectors includes: <ol style="list-style-type: none"> a. Predict whether a patient, hospitalized due to a heart attack, will have a second heart attack. The prediction is to be based on demographic, diet and clinical measurements for that patient. b. Estimate the amount of glucose in the blood of a diabetic person, from the infrared 	

absorption spectrum of that person's blood.

- c. Identify the risk factors for prostate cancer, based on clinical and demographic variables.

1.3 Linear regression:

Linear regression model is an algorithm based on supervised learning. Linear regression performs the task to predict a dependent variable value (y) based on given independent variables (x_1 through x_p). So, this regression technique finds out a linear relationship between x (input) and y (output). Hence, the name is Linear Regression.

Linear regression tries to fit the data into a straight line, by minimizing the variance in the form of sum of squares of all the errors. The formula of linear regression is:

$$y = X\beta + \varepsilon$$

Where y is the predicted or expected value of the dependent variable, X is $N \times K$ matrix independent or predictor variables, ε is the value of y when all of the independent variables are equal to zero i.e. error, and β is the estimated regression coefficients which is determined by ordinary least squares which is given by the formula

$$\beta = \arg \min_b \sum_{i=1}^N (y_i - x_i b)^2$$

Linear regression assumes that the data follows a normal distribution and fits a straight line which is not the case in our problem; hence linear regression gives RMSE error of 222.157.

```
1 s = time.time()
2 # Instantiating Linear_regression
3 lin_regressor = LinearRegression()
4 # Training with Linear regression
5 lin_regressor.fit(x_train,y_train)
6 # Predicting
7 y_lin = lin_regressor.predict(x_test)
8 # Calculating error by calling calculate_error function
9 lin_rmse = calculate_error(y_test, y_lin)
10 # Getting the r2 squared value rounding upto 3 values
11 lin_r2 = round(lin_regressor.score(x_test,y_test),3)
12 print("RMSE:",lin_rmse)
13 print("R2:",lin_r2)
14 e = time.time()
15 print("Time taken:",round(e-s,2),"sec")
```

```
RMSE: 222.157
R2: 0.469
Time taken: 0.03 sec
```


Advantages	Disadvantages
1) Easy to implement, interpret and very efficient to train. 2) Performs well when the data is linearly separable. 3) Prone to overfitting but it can be easily avoided using techniques like dimensionality reduction, regularization (L1 & L2) and cross validation.	1) Real world data are rarely linearly separable. 2) Fails to fit complex dataset as it is prone to noise and overfitting. 3) Outliers needs to be analyzed and removed as it is prone to outliers.
Applications	
1) Businesses frequently use linear regression to comprehend the connection between advertising spending and revenue. 2) It can be used in the medical field to understand the relationships between drug dosage and blood pressure of the patients. 3) Agriculture scientists frequently use linear regression to see the impact of rainfall and fertilizer on the number of fruits/vegetables yielded.	

We also tried the linear regression with different regularizations.

1.4 Ridge regression:

Ridge regression is also a linear regression with L2 regularisation while minimizing errors as given below:

$$\beta = \arg \min_b \sum_{i=1}^N (y_i - x_i b^2) + \alpha \sum_{k=1}^K b_k^2$$

L2 regularisation = $\alpha \sum_{k=1}^K b_k^2$, where, λ is a positive constant.

If α is 0, ridge regression behaves like linear regression, and α is large the model under-fits. It is important to carefully choose the value of α to avoid over-fitting.

With the best alpha of 0.003 ridge algorithm came up with RMSE value of 246.282.

```

1 s = time.time()
2 # Instantiating ridge function
3 ridge_regressor = Ridge(normalize=True)
4 # Initialising sets of parameters as dictionary
5 rid_params = {"alpha":list(np.arange(0,1,0.001))}
6 # Instantiating gridsearchcv with ridge regressor and the above parameters
7 rid_grid = GridSearchCV(ridge_regressor, param_grid=rid_params)
8 # Fitting x_train and y_train into gridsearchcv
9 rid_grid.fit(x_train,y_train)
10 print("Best alpha:",rid_grid.best_params_["alpha"])
11 print("RMSE:",calculate_error(y_test,y_ridge))
12 print("R2 score:",round(rid_grid.score(x_test,y_test),3))
13 print("Time taken:",round(time.time()-s,2),"sec")

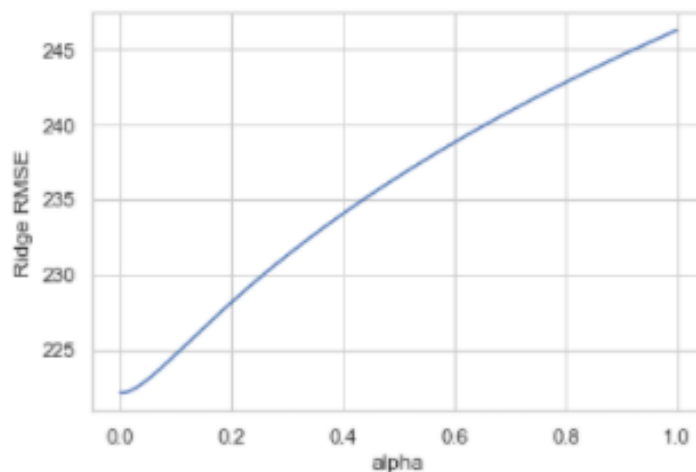
```

Best alpha: 0.003
 RMSE: 246.282
 R2 score: 0.469
 Time taken: 26.17 sec

```

1 s = time.time()
2 # Initialising sets of parameters as dictionary
3 rid_params = list(np.arange(0,1,0.001))
4 rid_errors,rid_scores=[],[]
5 # Getting the best parameter alpha
6 for i in rid_params:
7     rid_reg = Ridge(alpha=i,normalize=True)
8     rid_reg.fit(x_train,y_train)
9     y_ridge = rid_reg.predict(x_test)
10    sco = round(rid_reg.score(x_test,y_test),3)
11    rid_scores.append(sco)
12    error = calculate_error(y_test,y_ridge)
13    rid_errors.append(error)
14 plt.plot(rid_params,rid_errors)
15 plt.xlabel("alpha")
16 plt.ylabel("Ridge RMSE")
17 plt.show()
18 e = time.time()
19 print("Time taken:",round(e-s,2),"sec")

```



Time taken: 4.77 sec

Advantages	Disadvantages
1) Prevents overfitting. 2) Trades variance for bias 3) Prevents multicollinearity and model complexity by shrinking the parameters.	1) Increase bias 2) Need to select perfect α 3) Model interpretability is low
Applications:	
1) Ridge regression was developed as a possible solution to the imprecision of least square estimators when linear regression models have some multicollinear (highly correlated) independent variables—by creating a ridge regression estimator (RR). 2) It has been used in many fields including econometrics, chemistry, and engineering.	

1.5 Lasso regression:

Lasso regression is another linear regression with L1 regularisation:

$$\beta = \arg \min_b \sum_{i=1}^N (y_i - x_i b^2) + \alpha \sum_{k=1}^K |b_k|$$

L1 regularisation = $\alpha \sum_{k=1}^K |b_k|$, where, α is a positive constant. If α is 0 lasso behaves like linear regression, if very high it leads to under-fitting.

With the best alpha of 0.01 lasso algorithm came up with RMSE value of 222.156.

```

1 s = time.time()
2 # Instantiating Lasso function
3 lasso_regressor = Lasso(max_iter=1500)
4 # Initialising sets of parameters as dictionary
5 las_params = {"alpha":list(np.arange(0.01,0.2,0.001))}
6 # Instantiating gridsearchcv with Lasso regressor and the above parameters
7 las_grid = GridSearchCV(lasso_regressor, param_grid=las_params)
8 # Fitting x_train and y_train into gridsearchcv
9 las_grid.fit(x_train,y_train)
10 print("Best alpha:",las_grid.best_params_["alpha"])
11 print("RMSE:",calculate_error(y_test, y_lasso))
12 print("R2:",round(las_grid.score(x_test,y_test),3))
13 print("Time taken:",round(time.time()-s,2),"sec")

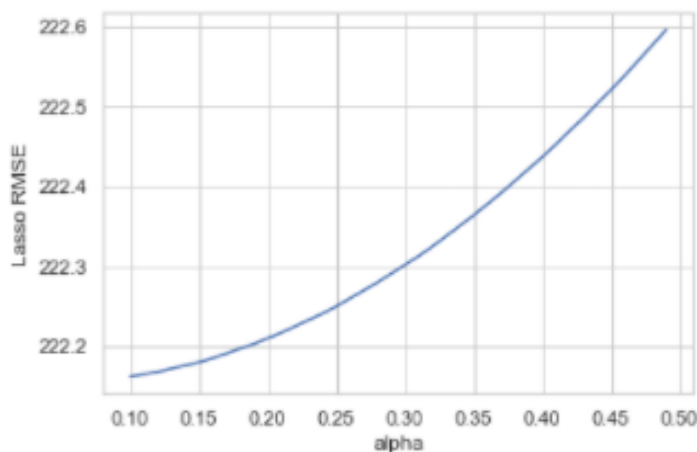
```

Best alpha: 0.01
 RMSE: 222.156
 R2: 0.469
 Time taken: 66.14 sec

```

1 s = time.time()
2 # Initialising sets of parameters as dictionary
3 las_params = list(np.arange(0.1,0.5,0.01))
4 las_errors,las_scores=[],[]
5 # Getting the best parameter alpha
6 for i in las_params:
7     las_reg = Lasso(alpha=i)
8     las_reg.fit(x_train,y_train)
9     y_lasso = las_reg.predict(x_test)
10    sco = round(las_reg.score(x_test,y_test),3)
11    las_scores.append(sco)
12    error = calculate_error(y_test,y_lasso)
13    las_errors.append(error)
14 plt.plot(las_params,las_errors)
15 plt.xlabel("alpha")
16 plt.ylabel("Lasso RMSE")
17 plt.show()
18 e = time.time()
19 print("Time taken:",round(e-s,2),"sec")

```



Time taken: 2.49 sec

Advantages	Disadvantages
1) Select features by shrinking coefficient towards zero. 2) Avoids overfitting.	1) Highly biased. 2) Arbitrarily selects only one feature from a group correlated features.
Applications:	
1) Lasso was introduced in order to improve the prediction accuracy and interpretability of regression models by eliminating unwanted parameters.	

1.6 ElasticNet regression:

Elasticnet regression tries to get the best solution by using both L1 and L2 regularisation.

$$\beta = \arg \min_b \sum_{i=1}^N (y_i - x_i b)^2 + \alpha \rho \sum_{k=1}^K |b_k| + \frac{\alpha(1-\rho)}{2} \sum_{k=1}^K b_k^2$$

Keeping l1_ratio, $\rho = 0.5$, so that it regularizes both L1 and L2 equally, with the best value of alpha= 0.01, RMSE value is 223.406

```

1 s = time.time()
2 # Instantiating elastic_net function
3 elasticnet = ElasticNet(tol=0.001,max_iter=20000)
4 # Initialising sets of parameters as dictionary
5 ela_params = {"alpha":list(np.arange(0.01,0.5,0.01))}
6 # Instantiating gridsearchcv with elasticnet and the above parameters
7 ela_grid = GridSearchCV(elasticnet, param_grid=ela_params)
8 # Fitting x_train and y_train into gridsearchcv
9 ela_grid.fit(x_train,y_train)
10 print("Best alpha:",ela_grid.best_params_["alpha"])
11 print("RMSE:",calculate_error(y_test, y_elastic))
12 print("R2:",round(ela_grid.score(x_test,y_test),3))
13 print("Time taken:",round(time.time()-s,2),"sec")

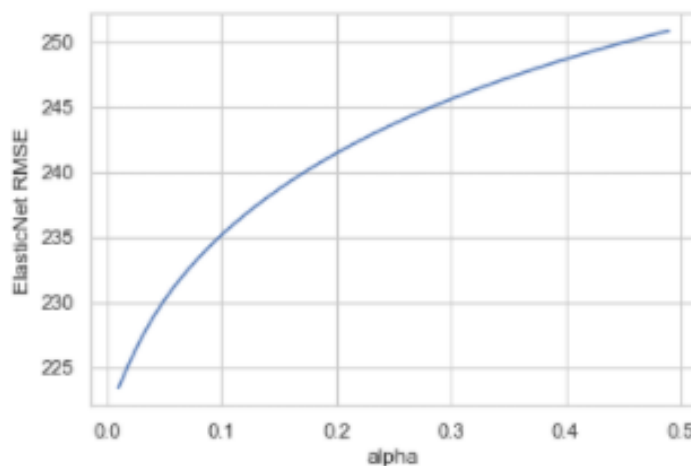
```

Best alpha: 0.01
 RMSE: 223.406
 R2: 0.463
 Time taken: 10.71 sec

```

1 s = time.time()
2 ela_params = list(np.arange(0.01,0.5,0.01))
3 ela_errors,ela_scores=[],[]
4 for i in ela_params:
5     elasticnet = ElasticNet(alpha=i,tol=1e-3,max_iter=2e4)
6     elasticnet.fit(x_train,y_train)
7     y_elastic = elasticnet.predict(x_test)
8     sco = round(elasticnet.score(x_test,y_test),6)
9     ela_scores.append(sco)
10    error = calculate_error(y_test,y_elastic)
11    ela_errors.append(error)
12    #print(round(i,6)," ",sco," ",error)
13 #print(ela_scores)
14 plt.plot(ela_params,ela_errors)
15 plt.xlabel("alpha")
16 plt.ylabel("ElasticNet RMSE")
17 plt.show()
18 e = time.time()
19 print("Time taken:",round(e-s),"sec")

```



Time taken: 3 sec

Advantages	Disadvantages
1) Doesn't have any problem when there are multiple features that are correlated with one another.	1) Computationally more expensive than Ridge and Lasso.
Applications	
1) Application of ElasticNet method includes: <ul style="list-style-type: none"> a. Support Vector Machine b. Metric learning c. Portfolio optimizer d. Cancer prognosis 	

1.7 SGD regression:

Stochastic gradient descent is an optimization algorithm often used in machine learning applications to find the model parameters that correspond to the best fit between predicted and actual outputs. It's an inexact but powerful technique.

Stochastic Gradient Descent (SGD) regressor basically implements a plain SGD learning routine supporting various loss functions and penalties to fit linear regression models.

SGD with default parameters: loss="squared_error" and eta0=0.01, with the best alpha=0.23, RMSE result with a large value and R2 with negative value.

```

1 s = time.time()
2 # Instantiating SGD_regression function
3 sgd_regressor = SGDRegressor()
4 # Initialising sets of parameters as dictionary
5 sgd_params = {"alpha":list(np.arange(0.01,1,0.01))}
6 # Instantiating gridsearchcv with sgd_regressor and the above parameters
7 sgd_grid = GridSearchCV(sgd_regressor, param_grid=sgd_params)
8 # Fitting x_train and y_train into gridsearchcv
9 sgd_grid.fit(x_train,y_train)
10 print("Best alpha:",sgd_grid.best_params_["alpha"])
11 print("RMSE:",calculate_error(y_test, y_sgd))
12 print("R2:",round(sgd_grid.score(x_test,y_test),3))
13 print("Time taken:",round(time.time()-s,2),"sec")

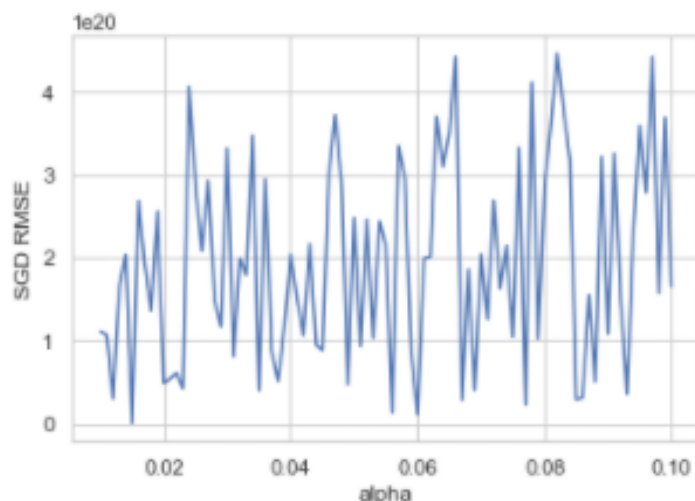
```

Best alpha: 0.23
 RMSE: 6.1861234326198714e+19
 R2: -2.5763897362568915e+35
 Time taken: 99.28 sec

```

1 s = time.time()
2 sgd_params = list(np.arange(0.01,.1,0.001))
3 sgd_errors,sgd_scores=[],[]
4 for i in sgd_params:
5     sgd_reg = SGDRegressor(alpha=i)
6     sgd_reg.fit(x_train,y_train)
7     y_sgd = sgd_reg.predict(x_test)
8     sco = round(sgd_reg.score(x_train,y_train),3)
9     sgd_scores.append(sco)
10    error = calculate_error(y_test,y_sgd)
11    sgd_errors.append(error)
12 plt.plot(sgd_params,sgd_errors)
13 plt.xlabel("alpha")
14 plt.ylabel("SGD RMSE")
15 plt.show()
16 print("Time taken:",round(time.time()-s,2),"sec")

```



Time taken: 26.75 sec

Advantages	Disadvantages
2) Doesn't have any problem when there are multiple features that are correlated with one another.	2) Computationally more expensive than Ridge and Lasso.
Applications	
2) Application of ElasticNet method includes: <ul style="list-style-type: none"> e. Support Vector Machine f. Metric learning g. Portfolio optimizer h. Cancer prognosis 	

1.8 Decision Tree regression:

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

With decision tree regression we could see that $RMSE=159.866$ with criterion="mse".

```

1 s = time.time()
2 # Instantiating DT_regression function
3 dt_regressor = DecisionTreeRegressor(splitter="best",random_state=2)
4 # Initialising sets of parameters as dictionary
5 dt_params = {"criterion":["mse","friedman_mse","mae","poisson"]}
6 # Instantiating gridsearchcv with sdg_regressor and the above parameters
7 dt_grid = GridSearchCV(dt_regressor,param_grid=dt_params)
8 # Fitting x_train and y_train into gridsearchcv
9 dt_grid.fit(x_train,y_train)
10 print("Best parameters:",dt_grid.best_params_["criterion"])
11 print("RMSE:",calculate_error(y_test, y_dt))
12 print("R2:",dt_grid.score(x_test,y_test))
13 print("Time taken:",round(time.time()-s,2),"sec")

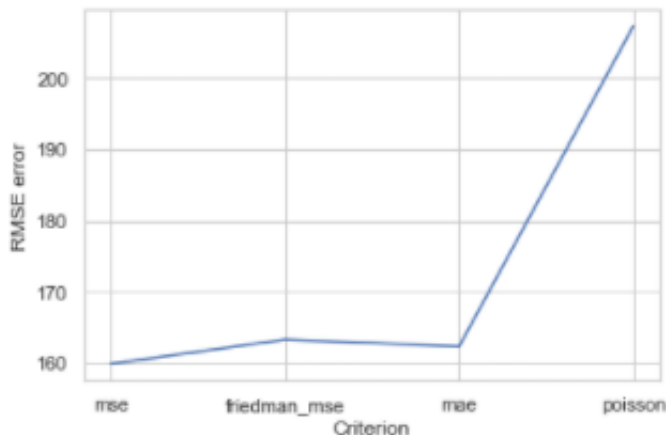
```

Best parameters: mse
RMSE: 159.866
R2: 0.7249691405527896
Time taken: 15.49 sec

```

1 s = time.time()
2 # Instantiating DT_regression function
3 criterion = ["mse","friedman_mse","mae","poisson"]
4 dt_errors,dt_scores=[],[]
5 for cri in criterion:
6     dt_regressor = DecisionTreeRegressor(criterion=cri,splitter="best",random_state=2)
7     dt_regressor.fit(x_train,y_train)
8     y_dt = dt_regressor.predict(x_test)
9     sco = dt_regressor.score(x_test,y_test)
10    dt_scores.append(sco)
11    error = calculate_error(y_test,y_dt)
12    dt_errors.append(error)
13 plt.plot(criterion,dt_errors)
14 plt.xlabel("Criterion")
15 plt.ylabel("RMSE error")
16 plt.show()
17 print("Time taken:",round(time.time()-s,2),"sec")

```



Time taken: 5.34 sec

Advantages	Disadvantages
<ol style="list-style-type: none"> 1) Simple to understand and to interpret. Trees can be visualized. 2) Requires little data preparation. 3) Able to handle both numerical and categorical data. 4) Non-parametric method which means no assumption needed. 	<ol style="list-style-type: none"> 1) Overfitting is one of the practical difficulties. 2) Not compatible for continuous numerical data. 3) Small variations in the data might result in a completely different tree being generated.
Applications	
<ol style="list-style-type: none"> 1) Marketing: Decision trees can help in audience segmentation and support businesses in producing better-targeted advertisements that have higher conversion rates. 2) Retention of Customers: Companies use decision trees for customer retention through analyzing their behaviors and releasing new offers or products to suit those behaviors. 3) Diagnosis of Diseases and Ailments: Decision trees can help physicians and medical professionals in identifying patients that are at a higher risk of developing serious (or preventable) conditions such as diabetes or dementia. 4) Detection of Frauds: Companies can prevent fraud by using decision trees to identify fraudulent behavior beforehand. It can save companies a lot of resources, including time and money. 	

1.9 XGBoost regression:

XGBoost (eXtreme Gradient Boosting), an ensemble learning method which offers a systematic solution to combine the predictive power of multiple learners, is used for better performance and execution speed.

XGBoost with best learning of 0.311, RMSE value is 121.009

```

1 s = time.time()
2 # Instantiating XGB_regression function
3 xgb_regressor = xg.XGBRegressor(objective ='reg:linear', verbosity = 0, silent=True, random_state=42)
4 # Initialising sets of parameters as dictionary
5 xgb_params = {"learning_rate":list(np.arange(0.001,0.5,0.01))}
6 # Instantiating gridsearchcv with sdg_regressor and the above parameters
7 xgb_grid = GridSearchCV(xgb_regressor, param_grid=xgb_params, cv=10)
8 # Fitting x_train and y_train into gridsearchcv
9 xgb_grid.fit(x_train, y_train)
10 print("Best learning rate:",round(xgb_grid.best_params_["learning_rate"],3))
11 print("RMSE:",calculate_error(y_test, y_xgb))
12 print("R2:",round(xgb_grid.score(x_test,y_test),3))
13 print("Time taken:",round(time.time()-s,2),"sec")

```

Best learning rate: 0.311

RMSE: 121.009

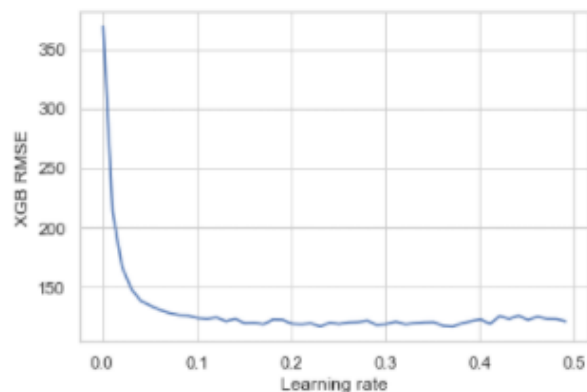
R2: 0.843

Time taken: 665.98 sec

```

1 s = time.time()
2 xgb_errors,xgb_scores=[],[]
3 xgb_params = list(np.arange(0.001,0.5,0.01))
4 for i in xgb_params:
5     xgb_regressor = xg.XGBRegressor(learning_rate=i, objective ='reg:linear',
6                                     verbosity = 0, silent=True, random_state=42)
7     xgb_regressor.fit(x_train,y_train)
8     y_xgb = xgb_regressor.predict(x_test)
9     sco = round(xgb_regressor.score(x_test,y_test),3)
10    xgb_scores.append(sco)
11    error = calculate_error(y_test,y_xgb)
12    xgb_errors.append(error)
13 plt.plot(xgb_params,xgb_errors)
14 plt.xlabel("Learning rate")
15 plt.ylabel("XGB RMSE")
16 plt.show()
17 print("Time taken:",round(time.time()-s,2),"sec")

```



Time taken: 82.47 sec

Comparing all the error values of all the algorithms, we could see that XGBoost performs best with RMSE value of 121.009. we assume that XGBoost can perform well with our whole dataset, so we go forward with this powerful algorithm.

	Model	RMSE value	R2 value
1	Baseline	360.930	-0.003
2	KNN regression	298.829	0.039
3	Linear regression	222.157	0.469
4	Ridge regression	246.282	0.469
5	Lasso regression	222.156	0.469
6	Elasticnet regression	223.406	0.463
7	Support Vector regression	6.19e+19	-2.57e+3
8	Decision Tree regression	159.866	0.724
9	eXtreme Gradient Boosting	121.009	0.843

3. Deep dive into best model:

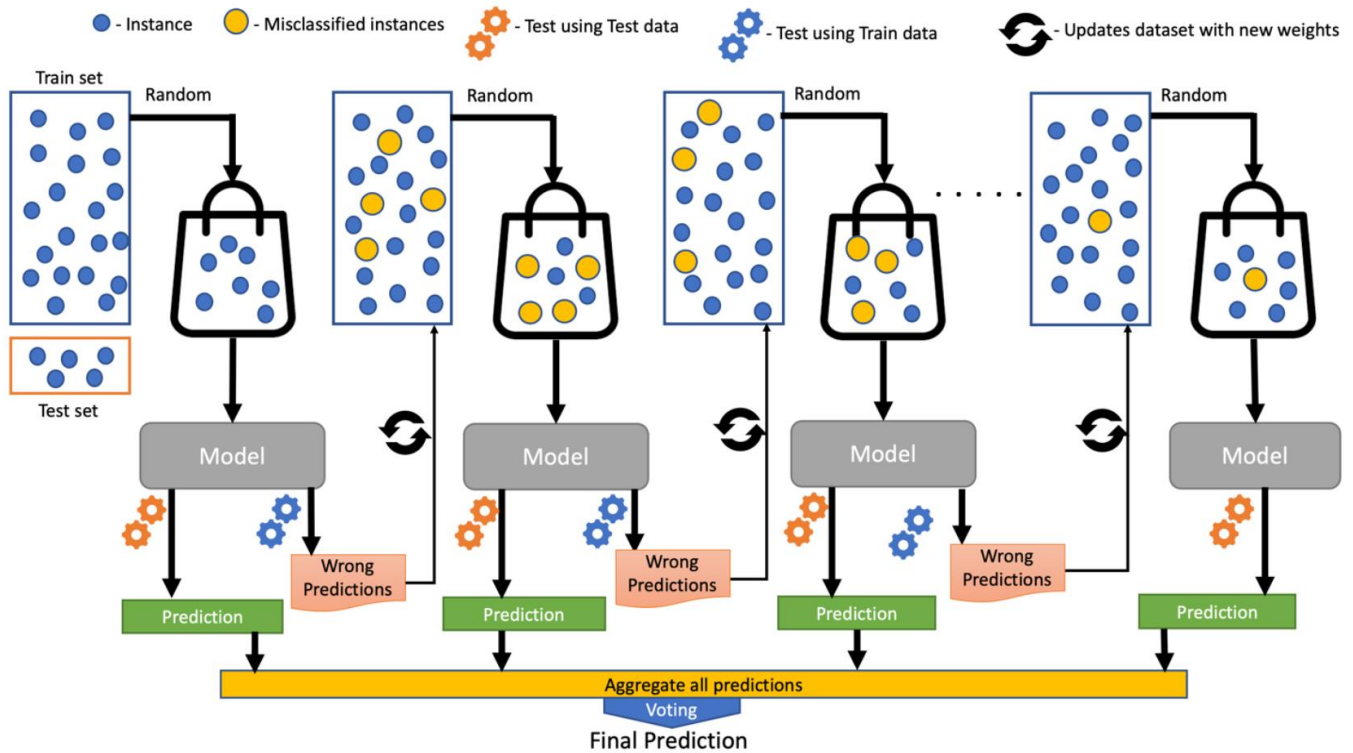
XGBoost is one of the most popular machine learning algorithms these days, regardless of the type of prediction task at hand; regression or classification. XGBoost is an ensemble learning method. From the above performance of different models, we see that it may not be sufficient to rely upon the results of just one machine learning model and XGBoost performs the best of all models. Ensemble learning offers a systematic solution to combine the predictive power of multiple learners. The resultant is a single model which gives the aggregated output from several models.

In boosting, the trees are built sequentially such that each subsequent tree aims to reduce the errors of the previous tree. Each tree learns from its predecessors and updates the residual errors. Hence, the tree that grows next in the sequence will learn from an updated version of the residuals.

The base learners in boosting are weak learners in which the bias is high, and the predictive power is just a tad better than random guessing. Each of these weak learners contributes some vital information for prediction, enabling the boosting technique to produce a strong learner by effectively combining these weak learners. The final strong learner brings down both the bias and the variance.

The beauty of this powerful algorithm lies in its scalability, which drives fast learning through parallel and distributed computing and offers efficient memory usage. The technique is much like a neural network.

The diagram below shows how XGBoost works.



The boosting ensemble technique consists of three simple steps:

An initial model F_0 is defined to predict the target variable y . This model will be associated with a residual $(y - F_0)$

A new model h_1 is fit to the residuals from the previous step

Now, F_0 and h_1 are combined to give F_1 , the boosted version of F_0 . The mean squared error from F_1 will be lower than that from F_0 :

$$F_1(x) < -F_0(x) + h_1(x)$$

To improve the performance of F_1 , we could model after the residuals of F_1 and create a new model F_2 :

$$F_2(x) < -F_1(x) + h_2(x)$$

This can be done for ' m ' iterations, until residuals have been minimized as much as possible:

$$F_m(x) < -F_{m-1}(x) + h_m(x)$$

Here, the additive learners do not disturb the functions created in the previous steps. Instead, they impart information of their own to bring down the errors.

Having a large number of trees might lead to overfitting. So, it is necessary to carefully choose the stopping criteria for boosting.

Gradient boosting helps in predicting the optimal gradient for the additive model, unlike classical gradient descent techniques which reduce error in the output at each boosting round.

The following steps are involved in gradient boosting:

$F_0(x)$ with which we initialize the boosting algorithm is to be defined:

$$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^n L(y_i, \gamma)$$

The gradient of the loss function is computed iteratively:

$$r_{im} = -\alpha \left[\frac{\partial(L(y_i, F(x)))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)}$$

Where, α is the learning rate.

Each $h_m(x)$ is fit on the gradient obtained at each step. The multiplicative factor γ_m for each terminal node is derived and the boosted model $F_m(x)$ is defined:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

In XGBoost, complexity of tree f_m is defined as

$$\Omega(f_m) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T \omega_j^2$$

Here γ is the penalization term on the number of terminal nodes, T is the number of leaves, λ is regularization term. ω_j is weight of the leaves.

Similarity score is calculated as

$$\text{similarity score} = \frac{(\text{sum of residuals})^2}{\text{no. of residual} + \lambda}$$

After calculating the similarity scores each node, best model can be picked by calculating the gain of each tree. This can be done by the following equation

$$Gain = Left_{sim} + Right_{sim} - Root_{sim}$$

$Left_{sim}$ = similarity score left node,

$Right_{sim}$ = similarity score right node, and

$Root_{sim}$ = similarity score of the root node

Pruning is done by removing the branches if the gain is smaller than γ .

Training with XGBoost takes $O(tdx \log n)$, where t is the number of trees, d is the height of the trees, and x is the number of data points in the training data. Prediction for a new sample takes $O(td)$

Table below shows some advantages and disadvantages of XGBoost along with its applications.

Advantages	Disadvantages
<ol style="list-style-type: none"> 1) Inbuilt L1 and L2 regularization prevents it from overfitting. 2) It can handle sparse and weighted data. 3) Inbuilt capability to handle missing values. 4) Better accuracy than single decision tree models 5) Better prediction performance as it use boosting ensemble learning algorithm. 	<ol style="list-style-type: none"> 1) Training time is pretty high for large dataset. 2) More likely to overfit than bagging.
Applications	
<ol style="list-style-type: none"> 1) XGBoost is intensively used in business intelligence modeling because of its ability to process scattered data efficiently, resulting in overall excellent model performance. 2) By optimizing CPU memory use with parallel computing, XGBoost becomes the appropriate model for extensive data models, classification, and feature recognition data. 	

Chapter IV:

Advanced Modeling and Feature Engineering

In the earlier section we were using data of only 4 weeks. After knowing XGBoost performs best in those data we go forward training our whole data with XGBoost algorithm.

1. Hyperparameter tuning:

To begin with, we run our whole dataset with default parameters. Default parameters looks like:

```
{'max_depth': 6,
  'min_child_weight': 1,
  'eta': 0.3,
  'subsample': 1,
  'colsample_bytree': 1,
  'eval_metric': 'rmse',
  'objective': 'reg:squarederror'}
```

Running the model with the default hyperparameters, we see that minimum RMSE value converges unto 183.77 in 33 rounds. Below snippet of output shows how it converges.

```
[23] Test-rmse:190.34909
[24] Test-rmse:188.46718
[25] Test-rmse:187.97455
[26] Test-rmse:187.35979
[27] Test-rmse:186.69882
[28] Test-rmse:186.07811
[29] Test-rmse:185.19665
[30] Test-rmse:184.92970
[31] Test-rmse:184.03519
[32] Test-rmse:183.76558
[33] Test-rmse:184.96480
[34] Test-rmse:185.00926
[35] Test-rmse:185.00470
[36] Test-rmse:184.83537
[37] Test-rmse:184.24490
[38] Test-rmse:184.06963
[39] Test-rmse:190.84912
[40] Test-rmse:190.71968
[41] Test-rmse:190.70969
Best RMSE with default parameters: 183.77 with 33 rounds
Time taken: 7.8748619556427 sec
```

With the help of gridsearch, tuning the value of max_depth and min_child_weight from 6 to 9 and 5 to 6 respectively, we see that minimum RMSE value is 133.69 when max_depth=7 and min_child_weight=6. Then we update these values to our parameter. Codes are given below.

```

1 s=time.time()
2 # Forming parameters for gridsearch
3 gridsearch_params = [(max_depth, min_child_weight)
4                       for max_depth in range(6,9)
5                       for min_child_weight in range(5,8)]
6 # Defining initial best params and MAE
7 min_rmse = float("Inf")
8 best_params = None
9 # Iterating each value gridsearch params
10 for max_depth, min_child_weight in gridsearch_params:
11     print("CV with max_depth={}, min_child_weight={}".format(max_depth,min_child_weight))
12     # Updating parameters
13     params['max_depth'],params['min_child_weight']=max_depth,min_child_weight
14     # Running CV
15     cv_results = xgb.cv(params,dtrain,num_boost_round=num_boost_round,seed=42,nfold=5,
16                        metrics='rmse',early_stopping_rounds=10)
17     # Updating best RMSE and boosting rounds
18     mean_rmse = cv_results['test-rmse-mean'].min()
19     boost_rounds = cv_results['test-rmse-mean'].argmin()
20     print("\tRMSE {} for {} rounds".format(mean_rmse, boost_rounds))
21     # Updating min_rmse and best_params
22     if mean_rmse < min_rmse:
23         min_rmse = mean_rmse
24         best_params = (max_depth,min_child_weight)
25 print("Best params: {}, {}, RMSE: {}".format(best_params[0], best_params[1], min_rmse))
26 print("Time taken:",round(time.time()-s),"sec")

```

```

CV with max_depth=6, min_child_weight=5
  RMSE 135.2981322 for 998 rounds
CV with max_depth=6, min_child_weight=6
  RMSE 135.4826172 for 824 rounds
CV with max_depth=6, min_child_weight=7
  RMSE 136.07359639999999 for 997 rounds
CV with max_depth=7, min_child_weight=5
  RMSE 134.5116884 for 785 rounds
CV with max_depth=7, min_child_weight=6
  RMSE 133.6908294 for 784 rounds
CV with max_depth=7, min_child_weight=7
  RMSE 135.72905559999998 for 485 rounds
CV with max_depth=8, min_child_weight=5
  RMSE 135.8453614 for 395 rounds
CV with max_depth=8, min_child_weight=6
  RMSE 135.050038 for 448 rounds
CV with max_depth=8, min_child_weight=7
  RMSE 135.30952440000002 for 343 rounds
Best params: 7, 6, RMSE: 133.6908294
Time taken: 4369 sec

```

Then we again tune the value of sub_sample and colsample from 9 to 11 and 6 to 11 respectively. From the output of below codes we could see that minimum RMSE value converges to 132.61 when sub_sample=1.0 and colsample=0.6.

```

1 s=time.time()
2 gridsearch_params = [(subsample, colsample)
3                       for subsample in [i/10. for i in range(9,11)]
4                       for colsample in [i/10. for i in range(6,11)]]
5 min_rmse = float("Inf")
6 best_params = None
7 # Iterating from the largest value and to the smallest
8 for subsample, colsample in reversed(gridsearch_params):
9     print("CV with subsample={}, colsample={}".format(subsample, colsample))
10    # We update our parameters
11    params['subsample'] = subsample
12    params['colsample_bytree'] = colsample
13    # Run CV
14    cv_results = xgb.cv(params,dtrain,num_boost_round=num_boost_round,seed=42,nfold=5,
15                       metrics={'rmse'},early_stopping_rounds=10)
16    # Update best score
17    mean_rmse = cv_results['test-rmse-mean'].min()
18    boost_rounds = cv_results['test-rmse-mean'].argmin()
19    print("\tRMSE {} for {} rounds".format(mean_rmse, boost_rounds))
20    if mean_rmse < min_rmse:
21        min_rmse = mean_rmse
22        best_params = (subsample,colsample)
23 print("Best params: {}, {}, RMSE: {}".format(best_params[0], best_params[1], min_rmse))
24 print("Time taken:",round(time.time()-s),"sec")

```

```

CV with subsample=1.0, colsample=1.0
    RMSE 133.6908294 for 784 rounds
CV with subsample=1.0, colsample=0.9
    RMSE 135.0003188 for 723 rounds
CV with subsample=1.0, colsample=0.8
    RMSE 133.8671446 for 857 rounds
CV with subsample=1.0, colsample=0.7
    RMSE 133.3833678 for 823 rounds
CV with subsample=1.0, colsample=0.6
    RMSE 132.6128082 for 964 rounds
CV with subsample=0.9, colsample=1.0
    RMSE 136.0344422 for 548 rounds
CV with subsample=0.9, colsample=0.9
    RMSE 134.58330099999998 for 934 rounds
CV with subsample=0.9, colsample=0.8
    RMSE 134.5635894 for 718 rounds
CV with subsample=0.9, colsample=0.7
    RMSE 136.70336 for 623 rounds
CV with subsample=0.9, colsample=0.6
    RMSE 135.207083 for 760 rounds
Best params: 1.0, 0.6, RMSE: 132.6128082
Time taken: 5377 sec

```

Last but not the least, we checked RMSE values for every values of “eta” = [0.3, 0.2, 0.1, 0.01, 0.05]. “eta” with value 0.1 gives minimum RMSE of 130.27.

```

1 s=time.time()
2 %time
3 min_rmse = float("Inf")
4 best_params = None
5 # Iterating each values
6 for eta in [.3, .2, .1, 0.01, .05]:
7     print("CV with eta={}".format(eta))
8     # Updating parameters
9     params['eta'] = eta
10    # Running CV
11    %time cv_results = xgb.cv(params,dtrain,num_boost_round=num_boost_round,seed=42,nfold=5,\
12                             metrics=['rmse'],early_stopping_rounds=10)
13    # Updating best score with its boost_rounds
14    mean_rmse = cv_results['test-rmse-mean'].min()
15    boost_rounds = cv_results['test-rmse-mean'].argmin()
16    print("\tRMSE {} for {} rounds\n".format(mean_rmse, boost_rounds))
17    # Checking whether mean_rmse is Less than min_rmse
18    if mean_rmse < min_rmse:
19        min_rmse = mean_rmse
20        best_params = eta
21 print("Best params: {}, RMSE: {}".format(best_params, min_rmse))
22 print("Time taken:",round(time.time()-s),"sec")

```

```

Wall time: 0 ns
CV with eta=0.3
Wall time: 9min 21s
    RMSE 132.6128082 for 964 rounds

CV with eta=0.2
Wall time: 8min 4s
    RMSE 132.56892720000002 for 838 rounds

CV with eta=0.1
Wall time: 9min 31s
    RMSE 133.5884828 for 998 rounds

CV with eta=0.01
Wall time: 9min 58s
    RMSE 160.86943060000002 for 998 rounds

CV with eta=0.05
Wall time: 9min 38s
    RMSE 139.5577792 for 998 rounds

Best params: 0.2, RMSE: 132.56892720000002
Time taken: 2794 sec

```

Finally the updated parameters looks like:

```

{'max_depth': 7,
 'min_child_weight': 6,
 'eta': 0.2,
 'subsample': 1.0,
 'colsample_bytree': 0.6,
 'eval_metric': 'rmse',
 'objective': 'reg:squarederror'}

```

Now we retrain our model with the updated parameters and see RMSE value.

	Train	Test
RMSE:	125.626	172.927
R2:	0.884	0.687

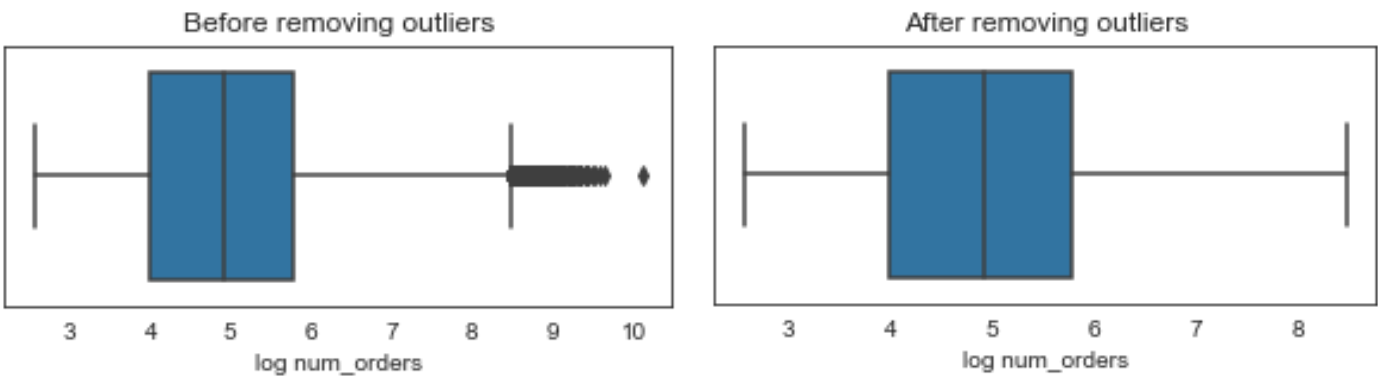
With proper hyper-parameter tuning we could decrease the error from 183.77 to 172.24, which is a good progress. Now to further decrease the error we went through some feature engineering.

2. Feature engineering:

We began with removing outliers from the “num_orders”. Quartile1 (*q1*), quartile2 (*q2*) and Inter Quartile Region (*iqr*) on log of “num_orders” are being calculated and the points which are less than $q1 - 1.5 * iqr$, and points which are greater than $q2 + 1.5 * iqr$ are being removed.

Log is being taken on the “num_orders” as it is exponentially distributed which is right skewed hence there are more outliers on the right side of the graph which we have already seen from the boxplot. This may lead to removal of non-outlier points from the dataset so it is necessary to take log to convert the data into normal like distribution before removal of outliers.

This is how the boxplot of log of “num_orders” looks before and after removing outliers.



Outliers for the feature “base_price” and “checkout_price” are also being removed in the same way. Finally we have 456241 data points after removing 306 outliers.

Feature named “discount_precentage” is formed and discount% is calculated for each data point with the formula

$$discount\% = \frac{checkout_price - base_price}{base_price} \%$$

Columns “checkout_price” and “base_price” are of no use now. Hence are removed. Column “id” containing unique values of each order would not give much value to our model so it is also being removed. Columns “city_code”, and “region_code” are also removed.

Now we are left with 456241 rows and 29 columns of data.

3. Fine tuning:

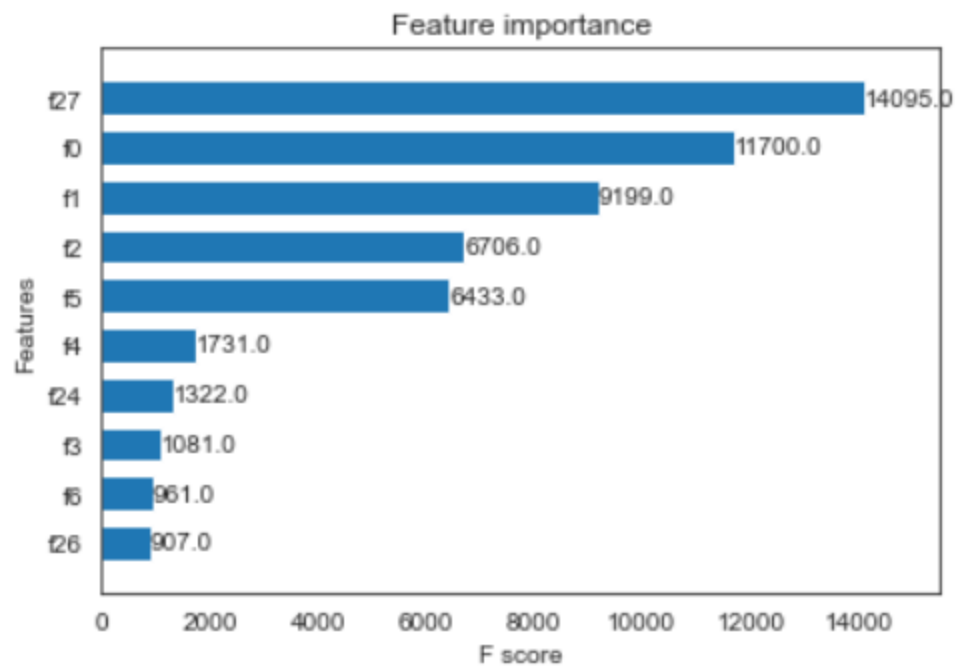
After the completion of feature engineering we fine tune the XGBoost model to get the best parameters for these newly engineered features. Fine tuning is done the same way we did hyper-parameter tuning but with newly formed dataset. Below shows are the parameters we got after fine tuning.

```
{'max_depth': 10,  
'min_child_weight': 7,  
'eta': 0.1,  
'subsample': 1,  
'colsample_bytree': 0.7,  
'eval_metric': 'rmse',  
'objective': 'reg:squarederror'}
```

We finally retrain the model with the above parameters and got RMSE= , which is reduced by %. Thus by giving some time we increase our models performance by just hyper-parameter tuning. The model is saved as a pickle file for deployment and Productionization.

	Train	Test
RMSE:	108.388	146.267
R2:	0.909	0.804

The following shows feature importance graph for the first 10 features.



Chapter V:

Deployment and Productionization

Deployment is the method of integrating a machine learning model into an existing production environment to make practical business decisions based on data. It is one of the last stages in the machine learning life cycle and can be one of the most cumbersome.

Model deployment is one of the most difficult processes of gaining value from machine learning. It requires a strong coordination between data scientists, IT teams, software developers, and business professionals to ensure the model works reliably in the organization's production environment. This presents a major challenge because there is often a discrepancy between the programming language in which a machine learning model is written and the languages our production system can understand, and re-coding the model can extend the project timeline by weeks or months.

In order to get the most value out of machine learning models, it is important to seamlessly deploy them into production so a business can start using them to make practical decisions

1. Amazon Web Service and Streamlit:

An application of the system developed with is built with Streamlit and is deployed on [Amazon Web Service \(AWS\)](#). AWS is a subsidiary of Amazon providing on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis. It was launched in 2006 when no computing has cloud computing business model and now 70% of the whole cloud computing market is residing on AWS.

Companies like Netflix, Kellogg's, General Electric, Adobe, Airbnb and Amazon are wholly dependent on AWS.

To deploy a system on this platform one has to sign up with AWS and create an EC2 instance, then connect with local host through "ssh" commands.

After successfully connecting with the remote host on AWS, data from the local host can be transferred to remote host by various methods like "scp" commands, [PuTTY](#) or [WinSCP](#).

Python notebook of the system is converted into pickle file, then transfer to the remote host along with other required files of the system. Pickle converts a python object into a byte stream to store it in a file or database, maintain program state across sessions, or transport data over the network. Below codes shows how a Python object named "best_model" is dumped or saved as "pro99_model.pkl" file:


```
import pickle
pickle.dump(best_model, open("pro99_model.pkl", "wb"))
```

In the AWS remote server, apart from libraries like Numpy, Pandas and XGBoost, to deploy the system, Pickle and Streamlit libraries are installed with “pip” command.

[Pickle](#) files are unpickled and are ready to use. Pickle can serialize arbitrary Python object with minimum codes. However, it is not universal as it is only designed for Python. Pickle is slow and insecure against erroneous and maliciously constructed data; therefore it is advised not to unpickled data received from an untrustworthy or unauthorized source. Pickle file is loaded with the following code:

```
model = pickle.load(open("pro99_model.pkl", "rb"))
```

[Streamlit](#) is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science. The perk of Streamlit is that we can code on python language. It is the best platform for a new bee who wants to develop an application or showcase their working machine learning models without learning any new web application languages. The application’s python file is also loaded to AWS EC2 instance and now the ready to on the remote server. Application on Streamlit is run with the following command:

```
streamlit run pro99_app.py
```

The application would start running on a web browser and takes inputs from users. Inputs in the form of variables – week numbers, center ID, meal ID, checkout price, base price, meal category, cuisine, center type, and operational area. Week number ranges from 146 to 155 that is the prediction will be for the next 10 weeks. Center ID is a list of 4 digit integer numbers assigned to each of the centers of all regions. Meal ID is also list of a Application python file preprocess the input variables and feed them into the model. The system outputs the predicted number of orders for a meal in a week for a particular center with an accuracy of 80.4% in just 998 micro seconds.

2. Throughput and Latency:

Latency refers to the time taken to process one unit of data provided only one unit of data is processed at a time. The latency of the system is $9.491910478003254e-07$ sec.

And the throughput is 1053528 . The following snippet of codes shows the latency and throughput of the system:

```
1 # Calculating throughput
2 # Throughput is the number of data points executed in one second
3 print("Number of data points executed in one second: {}".format(int(test_data.shape[0]/t)))
4 # Calculating Latency
5 # Latency is the time taken to execute one data point
6 print("Time taken to execute one data point: {} sec".format(t/test_data.shape[0]))
```

Number of data points executed in one second: 1053528

Time taken to execute one data point: 9.491910478003254e-07 sec

In Machine learning throughput is a measurement to determine the performance of various models for a specific application. It refers to the number of data unit processed in one unit of time.

3. Limitations:

The system predicts with 80.4% accuracy, it can be improved with more feature engineering with the help of domain experts.

As there are many hyperparameters in XGBoost, tuning all of them takes hours. And it could go wrong if not done carefully.

Visual design of the web application can be improved for better User Experience (UX).

Conclusion:

Demand prediction plays a crucial role in planning operations for restaurant's management. Dataset loaded from Kaggle is analyzed, understood and fitted into various machine learning algorithms, and discussed advantages and disadvantages for each one of them. It is almost universally agreed in the forecasting literature that no single method is best in every situation; however we extensively fine tuned XGBoost method to yield possible minimum errors to give best predictions. Throughout this whole process we understood that the meal demand highly depends on the discount given on the meal, type of the meal and the location of the store; with promotions on web pages and emails make enhancement on it.

Bibliography

- 1) <https://www.wikipedia.org/>
- 2) https://www.kaggle.com/arashnic/food-demand?select=foodDemand_train
- 3) <https://towardsdatascience.com/how-to-machine-learning-driven-demand-forecasting-5d2fba237c19>
- 4) <https://towardsdatascience.com/forecast-kpi-rmse-mae-mape-bias-cdc5703d242d>
- 5) <https://www.shopify.in/retail/demand-forecasting>
- 6) <https://www.shipbob.com/blog/demand-forecasting/>
- 7) <https://retalon.com/blog/forecasting-demand>
- 8) <https://www.relexsolutions.com/resources/demand-forecasting/>
- 9) <https://mobidev.biz/blog/machine-learning-methods-demand-forecasting-retail>
- 10) <https://blog.minitab.com/en/adventures-in-statistics-2/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit>
- 11) <https://www.aionlinecourse.com/tutorial/machine-learning/evaluating-regression-models-performance>
- 12) <https://www.geeksforgeeks.org>
- 13) <https://www.analyticsvidhya.com/blog/2021/05/know-the-best-evaluation-metrics-for-your-regression-model/>
- 14) <https://akhilendra.com/evaluation-metrics-regression-mae-mse-rmse-rmsle/>
- 15) <https://www.statisticshowto .com/probability-and-statistics/regression-analysis>
- 16) https://xgboost.readthedocs.io/en/stable/python/python_api.html
- 17) <https://www.youtube.com/watch?v=OtD8wVaFm6E>
- 18) <https://dzone.com/articles/xgboost-a-deep-dive-into-boosting>
- 19) <https://xgboost.readthedocs.io/en/latest/tutorials/model.html>