Capstone Project

Vidhya

Machine Learning Engineer Nanodegree

Recruit Restaurant Visitor Forecasting

Project Overview:

The most important factor for the success of a restaurant is its customers. Restaurants try to create the best dining experience for their customers while keeping the operational costs low. Being able to predict the number of customers a restaurant can expect on a day can help it to effectively plan on scheduling staff members, purchasing supplies and allows it to focus on creating an enjoyable dining experience for their customer. Several factors affect the number of visitors to a restaurant, the day of the week, is it a holiday, the type of cuisine the restaurant serves, and the price range of the restaurant are some of the factors. These factors when used along with historical visitor information can predict the future visitors and can help restaurants improve their business.

Restaurants can increase or decrease the number of staff and optimize supply stock based on the volume and thus reducing the cost of operations. Restaurants can also create customized special menus on high volume days to create a better dining experience.

The dataset used for building a model to predict future visitors to a restaurant is a time-series dataset. The model, given the date, should forecast the number of visitors a restaurant will receive on that date. The dataset contains historical visitor data for about 800+ restaurants in Japan which are clients of

- Hot Pepper Gourmet (hpg): like Yelp, here users can search restaurants and also make a reservation online
- AirREGI / Restaurant Board (air): like Square, a reservation control and cash register system

Problem Statement:

The goal of this project is to improve business by predicting 'no of visitors a restaurant will receive in future. The input data is in the form of a time-series, where the time/date is the input and the number of visitors is the output. This time series problem can be approached as supervised learning problem by reframing the time series dataset into supervised regression problem dataset. This reframing can be done by replacing time with previous time step's visitor count. This conversion still maintains the order by replacing time with average no of customers of previous year/month based on the data. This conversion creates the flexibility to apply suites of linear and non-linear supervised machine learning algorithm.

Metrics:

In this project, the output is a continuous value. Then this problem would come under supervised regression problem. The most commonly used metrics to measure the performance of regression problem is Root Mean Square Error

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

It's the square root of the average of squared differences between prediction and actual observation. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable even if count is less.

Analysis

Data Exploration

The dataset for this problem are in csv format. Each file is prefaced with the source (either air or hpg) to indicate its origin. Each restaurant has a unique air_store_id and hpg_store_id.

air_visit_data.csv - This file contain air_Store_id, visit_date and no of visitors to those restaurants. This file contains training examples

air_store_info.csv: This file contains information about select air restaurants. Column names and contents are self-explanatory. air_store_id, air_genre_name, air_area_name , latitude longitude latitude and longitude of the area to which the store belongs

air_reserve.csv: This file contains reservations made in the air system. It includes following fields air_store_id - the restaurant's id in the air system visit_datetime - the time of the reservation visitor will occur reserve_datetime - the time the reservation was made reserve_visitors - the number of visitors for that reservation

hpg_store_info.csv: This file contains information about select hpg restaurants. Column names and contents are self-explanatory. hpg_store_id hpg_genre_name hpg_area_name latitude longitude Note: latitude and longitude are the latitude and longitude of the area to which the store belongs

hpg_reserve.csv: This file contains reservations made in the hpg system. hpg_store_id - the restaurant's id in the hpg system visit_datetime - the time of the reservation reserve_datetime - the time the reservation was made reserve_visitors - the number of visitors for that reservation

store_id_relation.csv: This file allows you to join select restaurants that have both the air and hpg system. hpg_store_id air_store_id air_visit_data.csv This file contains historical visit data for the air restaurants.

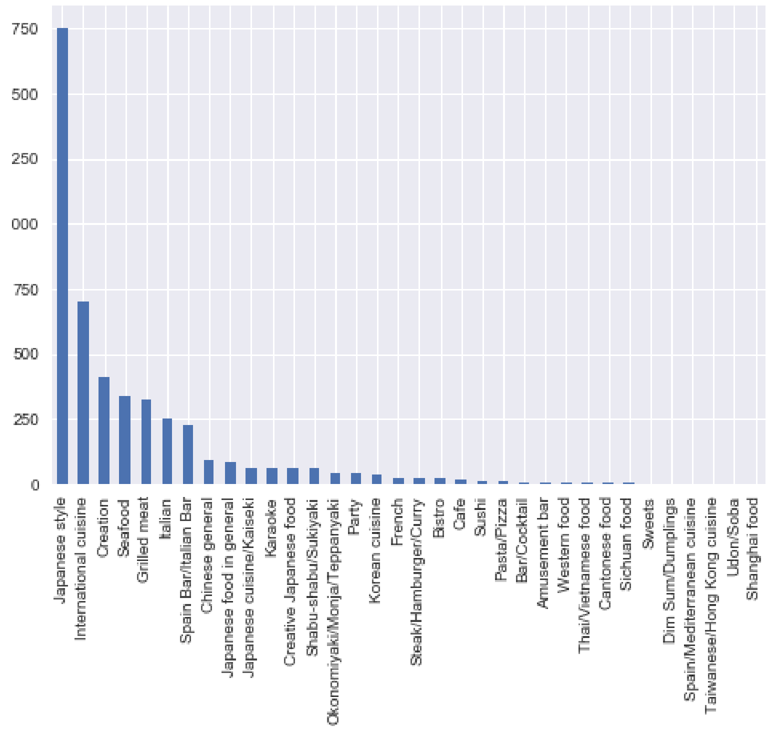
sample_submission.csv: This file shows a submission in the correct format, including the days for which you must forecast. This is the testing file. id - the id is formed by concatenating the air_store_id and visit_date with an underscore visitor- the number of visitors forecasted for the store and date combination

date_info.csv: This file gives basic information about the calendar dates in the dataset. calendar_date, day_of_week are self-explanatory. holiday_flg - is the day a holiday in Japan

The training data covers the dates from 2016 until April 2017. The test set covers the last week of April and May of 2017.

Data Visualization:

Total 800+ restaurants. Different genre restaurants. Majority of restaurant serves Japanese style food that make sense because these restaurants from Japan.



Count of visitors is the target variable. To understand the distribution of no of visitors during holidays and working days, I found summary statistics for both option and summary below shows that average no of visitors on holidays are greater than working days.

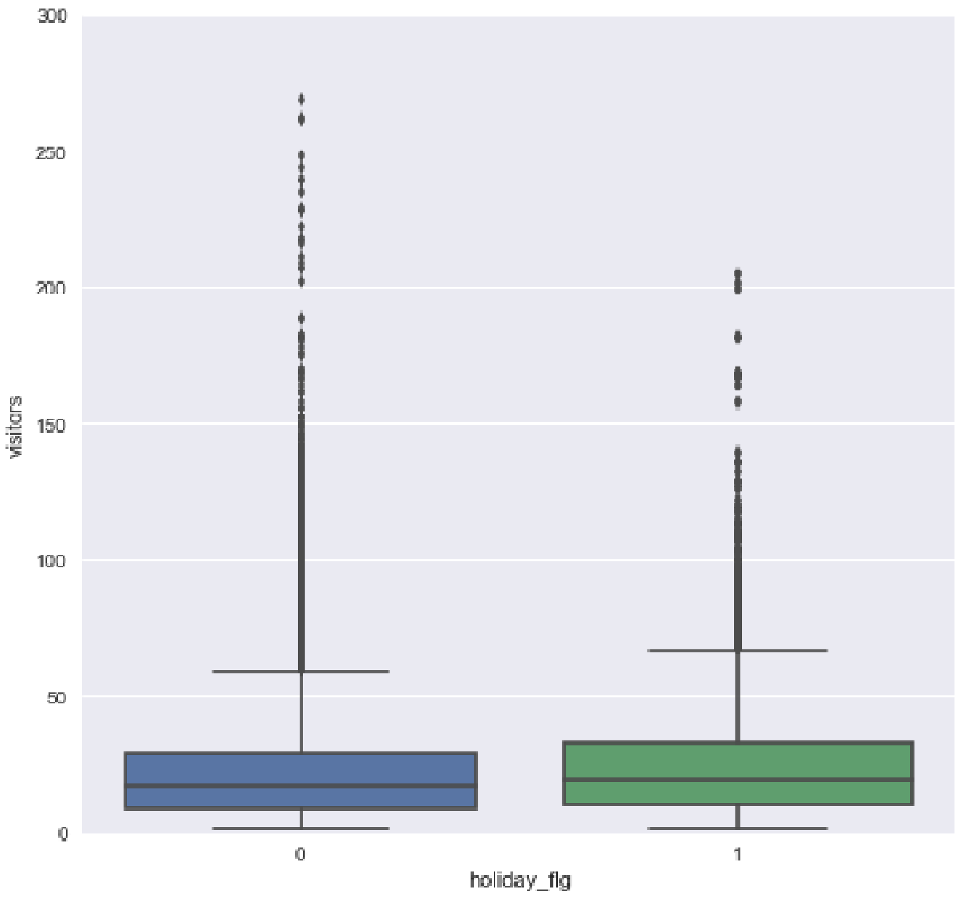
Summary statistic of no of visitors during working days.

```
count    239333.000000
mean         20.828064
std          16.624503
min           1.000000
25%           9.000000
50%          17.000000
75%          29.000000
max         877.000000
Name: visitors, dtype: float64
```

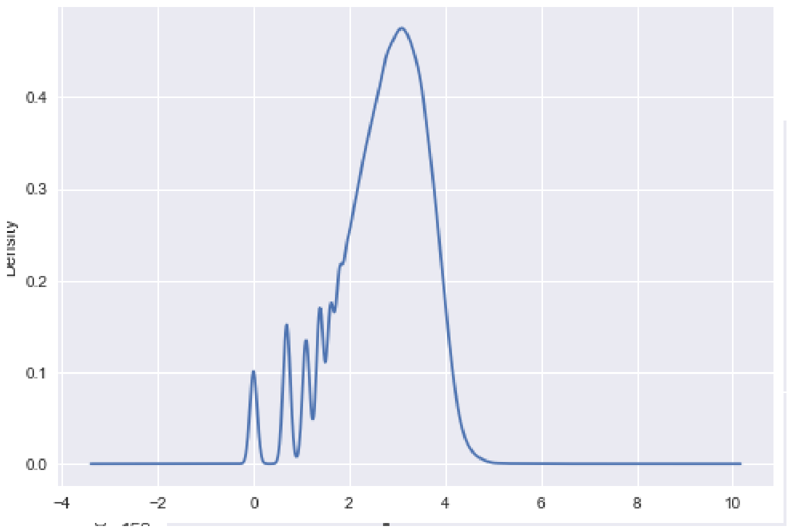
Summary statistic of no of visitors during holidays.

```
count    12775.000000
mean        23.703327
std         18.863804
min          1.000000
25%         10.000000
50%         19.000000
75%         33.000000
max        205.000000
Name: visitors, dtype: float64
```

In all percentile and average for no of visitors on holidays are higher but max no of visitors is high during working day. This shows the presence of extreme values. Because some restaurant might receive high no of customers during working day, maybe this restaurant serves daily food for working people at affordable range



The plot below shows no of visitors of median values and percentiles are high during holidays. It also shows the presence of extreme values. So instead of treating them as outliers, let's try a log transformation to nullify their effect.



After taking log transformation the distribution no of visitors looks normal distribution.

Algorithm and techniques.

1.Datatype conversion

- Given dataset is timeseries dataset, finding the datatype of time column and if its type is string then convert the string into pandas datetime format.
- Restaurants listed in the dataset are clients of The Hot Pepper Gourmet and Air REGI Through them clients can reserve table for different restaurants. The data also include reserve and visit date information.
- Finding difference between reserve_datetime and visit_datetime of ar and hr data frame on each row and grouby based on restaurant id and visit date to find the total no of reserve visitors for each restaurant.

2.Perform Feature engineering:

- Extracting day of week, year, month, date part of time stamp from visit_date of training and testing data frames.
- Convert the categorical variables like genre_names, area_names into numeric variable using encoder from Sklearn library.
- To match the features in both training and testing dataset, air_store_id in testing data frame is concatenation of id and visit date, extracting visit_date and id from that air_store_id column of testing data frame. **3.Data preparation for training and testing:**

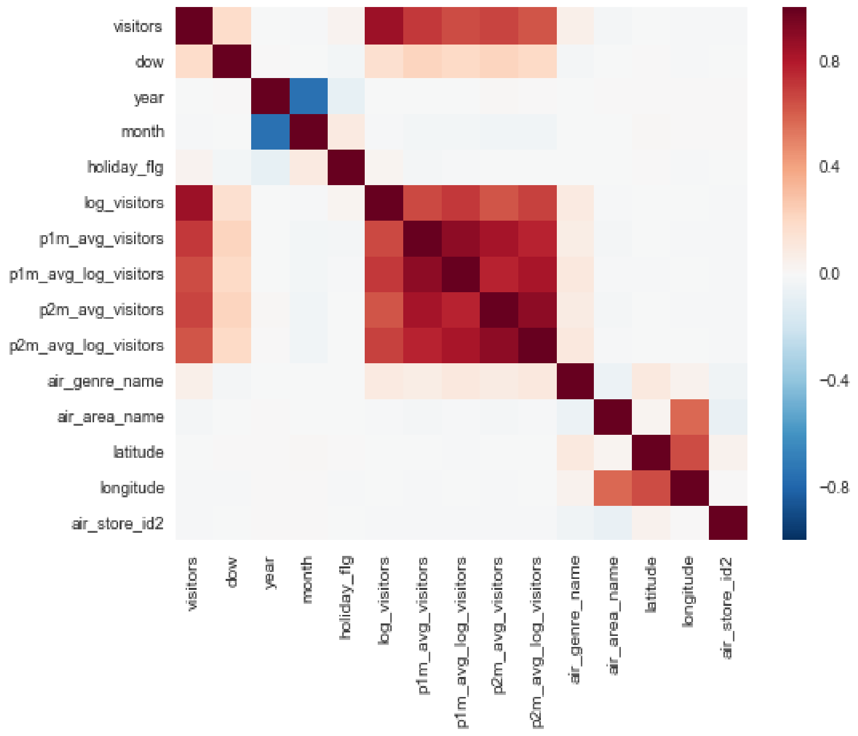
- Merging training and testing data frame with holiday data frame using left join on column visit_date.
- Because model must learn the increase/decrease in no of visitors and its relationship with holiday_flag, which tells weather that particular day is holiday or not.
- No of visitors to restaurant also influenced by location of the restaurant and cuisines of the restaurant. So, information related to location and cuisines are added to training and test set.

4.Conversion of time series data set into supervised machine learning:

- Time series problem can be framed as supervised learning problem. This will open a lot of linear and nonlinear supervised machine learning algorithm. This can be done using the previous time steps as input variable and next time step as output variable. The number of previous time steps is called the window width or size of the lag. This method referred as sliding window method for time series data.

Reason for the conversion:

- When you are training the model to forecast no of visitors, input will be the timestamp because the model has to forecast on particular time stamp what will be the counts of visitors. But when we see the correlation between timestamp and no of visitors which are weekly correlated. And if we train the model with data/time as input , the output produced will not be accurate.
- To overcome this problem, instead of giving timestamp as input will replace with previous year/previous month average no of visitor (depending on the problem) as input which is strongly correlated with current no of visitors.



If we train previous time step data as input, output produced will be better accurate and model will be robust by capturing the trend and seasonality in the dataset.

The plot shown above is the correlation matrix, the dark color represents strong correlation. Visitors is the target variable. The input variables strongly correlated with visitor variables are

- p1m_avg_visitors: Previous month's day of week’s average no of visitors.
- p1m_avg_log_visitors: Previous month day of week average no of log transformed visitors p2m_avg_visitors: Two months ago, day of week average no of visitors.
- p2m_avg_log_visitors: Two months ago, day of week average no of log transformed visitors.
- Dow and genre_name

The above mentioned four input variables are computed in the process of converting time series into supervised regression problem.

Framing Time series to supervised learning problem

- The trend and seasonality pattern in the data set can be captured in conversion by including previous year's day of the week average no of visitors. But the training data given for this problem does not have enough data to include previous year day of week average, so I have taken previous two months day of week average no of visitors

5.Machine learning Algorithm:

- Once the time series dataset is converted into supervised regression problem. Now suites of linear and nonlinear regression Algorithm can be applied to this dataset.
- I have chosen following Regression Algorithm
    - Stochastic Gradient Descent regressor
    - Support vector regressor:
    - KNearestNeighbor regressor
    - Random Forest Regressor
- I have implemented using Sklearn libraries.
- Using helper function split_to_kfold, divide the data into training and validation and then using Sklearn libraries, each model is implemented and trained using the fit method and y-hat values are predicted and compared with actual value.
- Compute the root mean square error of predicted and actual values for every iteration and find the average RMSE.

Benchmark model:

- Bench mark model: Zero Rule Algorithm
- To know whether the predictions for above mentioned algorithms are good or not, we need to use baseline prediction algorithm.
- A baseline prediction algorithm provides a set of predictions that I evaluate as I would any predictions for the problem, such as RMSE.

- The scores from these algorithms provide the required point of comparison when evaluating all other machine learning algorithms on my problem.
- Once established, you can comment on how much better a given algorithm is as compared to the naive baseline algorithm, providing context on just how good a given method actually is.
- The two most commonly used baseline algorithms are:
    - Random Prediction Algorithm.
    - Zero Rule Algorithm.
- I have chosen zero Rule Algorithm as my Bench mark model.
- The Zero Rule Algorithm is a better baseline than the random algorithm.
- It uses more information about a given problem to create one rule in order to make predictions.
- Regression problems require the prediction of a real value.
- A good default prediction for real values is to predict the central tendency. This could be the mean or the median.
- A good default is to use the mean (also called the average) of the output value observed in the training data.
- This is likely to have a lower error than random prediction which will return any observed output value.
- Below is a function to do that named zero_rule_algorithm_regression(). It works by calculating the mean value for the observed output values.
- https://machinelearningmastery.com/implement-baseline-machine-learning-algorithms-scratch-python/

Methodology

Data Preprocessing:

- As a result of feature engineering and data set preparation, new feature has been extracted from the input features and relevant features have been merged to training and testing dataset from other data frames.
- Before applying machine learning model, the data should be numeric. So, convert categorical variables into numeric using encoder from Sklearn libraries.
- Feature scaling is done, data are scaled to a fixed range - usually 0 to 1 using Min max scaler
- The cost of having this bounded range is that we will end up with smaller standard deviations, which can suppress the effect of outliers.
- Algorithm like KNearest neighbor which uses distance to find the similarity, it is better to scale all the features so that feature which has high range value should not dominate to find similarity. After scaling all the features value ranges from 0 to 1 then applying KNN on this dataset will find the similarity feature for target variable would be highly relevant features.

Implementation

conversion of dataset into Supervised learning problem:

- The no of visitors on holidays and working day have huge difference for different restaurants, to include the factor in training dataset. Filtering the train dataset which include the rows of no of visitors on working day and the result is stored in temp_no_hold and compute previous month day of week average excluding holidays and add that to training data.
- Now the training data have daily visitor information+ previous month average visitors +holiday flag along with other relevant information. When this data is trained the model see the pattern that visitors count, previous moth count and holiday flag and find the relationship with count and holiday flag to predict the future visitors' count.

```
temp_no_hold=temp[temp.holiday_flg==0]
```

```
air_store_id  visit_date  visitors  dow  year  month  day_of_week
\
0  air_ba937bf13d40fb24  2016-01-13        25    2  2016      1
6
1  air_ba937bf13d40fb24  2016-01-14        32    3  2016      1
4
2  air_ba937bf13d40fb24  2016-01-15        29    4  2016      1
0
3  air_ba937bf13d40fb24  2016-01-16        22    5  2016      1
2
4  air_ba937bf13d40fb24  2016-01-18         6    0  2016      1
1

   holiday_flg  log_visitors
0            0      3.218876
1            0      3.465736
2            0      3.367296
3            0      3.091042
4            0      1.791759
```

- This time series to supervised learning dataset conversion is done using groupby in pandas and helper function.
- groupby: Used to compute every month's day of week average no of visitor for each restaurant on temp data frame.
- create two copies of stores data frame and stored as stores_NH and sotres_NH_1 data frame.
- Helper function(compute_shift(data)): This function used to increment the values in month column of stores_NH data frame by 1. After incrementing 1, the values in month column

incremented from 2 to 13. But month ranges from 1 to 12. values of 13 in month column should be changed to 1 and values of year in that row should also incremented by one. It refers to first month in next year.

- compute_shift_1(data): This function used to increment the values in month column of stores_NH_1 data frame by 2. After incrementing by 2, the values in month column incremented from 3 to 14. But month ranges from 1 to 12. values of month greater than 12 is replaced with (current value of month - 12) and values of year in that row should also incremented by one.
- After incrementing Month and year values in stores_NH data frame, then merging stores_NH and temp data frame based on 'air_store_id','year','month','dow' using left join, previous month day of week average visitors will be joined to current month .ie when month 7 is incremented to 8 when we perform left join with training data based on 'air_store_id','year','month','dow',8th month day of week average is merged to 8th month dow of each restaurant in training data but that 8 month average month visitor value is actually average value of 7th month. After merging in each row, you have visitors on that day plus previous month that day of week visitors average.
- After incrementing Month and year values in stores_NH_1 data frame, then merging stores_NH_1 and temp dataframe based on 'air_store_id','year','month','dow', two month ago day of week average visitors will be joined to current month .ie when month 7 is incremented to 9 when we perform left join with training data based on 'air_store_id','year','month','dow',9th month day of week average is merged to 9th month dow of each restaurant in training data but that 9 month average month visitor value is actually average value of 7th month. After merging in each row, you have visitors on that day plus month before previous month's day of week visitors average visitor.
- The output of above operation is stored in train dataframe. Now this dataframe holds daily no of visitors information along with previous month day of week average visitors and two months ago day of week average visitors.
- Again stores_NH and stores_NH_1 are merged with test data frame. The result of this operation is stored in test. Now this data frame has previous month's and month before previous month's day of week average no of visitors. But daily no of visitor is zero which must be predicted.
- The merging operation mentioned above is done using left join so that training and test dataframe information is retained when the values for row in training and testing is not found in other dataframe we are merging with. These values are filled with NAN value.
- Now when training the model, input which is previous month day of week average no of visitors of current month and output will be no of visitors of current month. These are strongly correlated.
- hence the conversion.

The output of `train.info()` will give information about training data after this conversion

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 187058 entries, 0 to 187057
```

```
Data columns (total 17 columns):
air_store_id            187058 non-null object
visit_date              187058 non-null object
visitors                187058 non-null int64
dow                     187058 non-null int64
year                    187058 non-null int64
month                   187058 non-null int64
holiday_flg             187058 non-null int64
log_visitors            187058 non-null float64
p1m_avg_visitors        187058 non-null float64
p1m_avg_log_visitors    187058 non-null float64
p2m_avg_visitors        187058 non-null float64
p2m_avg_log_visitors    187058 non-null float64
air_genre_name          187058 non-null int64
air_area_name           187058 non-null int64
latitude                187058 non-null float64
longitude               187058 non-null float64
air_store_id2           187058 non-null int64
dtypes: float64(7), int64(8), object(2)
memory usage: 25.7+ MB
```

The output of `test.info()` will give information about training data after this conversion

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 32019 entries, 0 to 32018
Data columns (total 17 columns):
id                      32019 non-null object
visitors                32019 non-null int64
visit_date              32019 non-null object
air_store_id            32019 non-null object
dow                     32019 non-null int64
year                    32019 non-null int64
month                   32019 non-null int64
p1m_avg_visitors        29759 non-null float64
p1m_avg_log_visitors    29759 non-null float64
p2m_avg_visitors        29839 non-null float64
p2m_avg_log_visitors    29839 non-null float64
air_genre_name          32019 non-null int64
air_area_name           32019 non-null int64
latitude                32019 non-null float64
longitude               32019 non-null float64
day_of_week             32019 non-null int64
holiday_flg             32019 non-null int64
dtypes: float64(6), int64(8), object(3)
```

Handling Missing values:

In testing dataset ,

- p1m_avg_visitors, p1m_avg_log_visitors, p2m_avg_visitors, p2m_avg_log_visitors. These columns in testing data set have NAN values, which is filled by following ways

- Compute yearly, day of week average no of visitors of temp_no_holiday dataframe and result is stored in avgdow_yearly dataframe.
- 2.Then merge this dataframe with test dataframe. Then fill the missing values in above columns from avg_visitors and log_avg_visitors columns of avgdow_yearly dataframe.
- After merging operation, the above four columns have NAN values but has reduced from previous count.
- To fill those missing values, compute the day of week month average no of visitors for restaurants in same area and serves same cuisine using groupby in pandas.
- The result of above operation is stored as cuisines dataframe. Since the average is computed monthly we will use compute_shift and compute_shift_1 function to increment month and year values. And the result is stored as cusines_p1m and cusines_p2m dataframes.
- Then merges these dataframes with test dataframe and fill the missing value from column in test dataframe with columns from cusines_p1m and cusines_p2m dataframes.
- After filling the NAN values, columns included in test dataframe because of merging is dropped.
- This merging operation reduces NAN and remaining NAN is replaced with negative 1.

The result of above steps the output of testing dataset is

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 32019 entries, 0 to 32018
Data columns (total 17 columns):
id                        32019 non-null object
visitors                  32019 non-null int64
visit_date                32019 non-null object
air_store_id              32019 non-null object
dow                       32019 non-null int64
year                      32019 non-null int64
month                     32019 non-null int64
p1m_avg_visitors          31900 non-null float64
p1m_avg_log_visitors      31900 non-null float64
p2m_avg_visitors          31899 non-null float64
p2m_avg_log_visitors      31899 non-null float64
air_genre_name            32019 non-null int64
air_area_name             32019 non-null int64
latitude                  32019 non-null float64
longitude                 32019 non-null float64
day_of_week               32019 non-null int64
holiday_flg               32019 non-null int64
dtypes: float64(6), int64(8), object(3)
memory usage: 4.4+ MB
```

Dividing training dataset for training and validating the model:

- Dividing the training dataset into training (X_train, y_train) and testing(X_test and y_test).

- Using K-fold cross validation, divide the training dataset into k equal folds. Then k-1 folds are chosen randomly for training and leaving one-fold for validating the regression models.
- Repeats this process for k iteration.
- Then the models performance is measured using performance metrics (Root mean square error) and takes average of the metrics and choose the model which has less value of Root mean square error will be chosen to evaluate the model using testing set.

Implementation

Using Sklearn libraries, training dataset is divided into training and test set and validation dataset as well

```
X_train, X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=1)
```

K-fold is implemented using `split_to_kfold(X_train,y_train,nfolds,model)`

Algorithm and its implementation:

Bench mark model: Zero Rule Algorithm

Implementation:

This algorithm is implemented by taking average of target variable i.e.. count of visitors and it uses that average as the predicted value for unseen data.

```
RMSE of benchmark model: 0.132471734388
```

Stochastic Gradient Descent:

- This Algorithm finds a best fit line for the data using linear function y=mx+b.
- A standard approach to solving this type of problem is to define an error function (also called a cost function) that measures how “good” a given line is. This function will take in a (m,b) pair and return an error value based on how well the line fits our data. To compute this error for a given line, we’ll iterate through each (x,y) point in our data set and sum the square distances between each point’s y value and the candidate line’s y value (computed at mx + b). It’s conventional to square this distance to ensure that it is positive and to make our error function differentiable.
- Lines that fit our data better (where better is defined by our error function) will result in lower error values. If we minimize this function, we will get the best line for our data
- To minimize the error, run gradient descent on the error function. To compute it, we will need to differentiate our error function. Since our function is defined by parameters (m and b), we will need to compute a partial derivative for each.
- We can initialize our search to start at any pair of m and b values (i.e., any line) and let the gradient descent algorithm march downhill on our error function towards the best line. Each iteration will update m and b to a line that yields slightly lower error than the previous iteration. The direction to move in for each iteration is calculated using the two partial derivatives.
- The learning Rate variable controls how large of a step we take downhill during each iteration. If we take too large of a step, we may step over the minimum. However, if we take small steps, it will require many iterations to arrive at the minimum.

- This hyper parameter need to be tuned to find out best value for this dataset.
- To avoid overfitting l2 regularization is used, which will increase the penalty of error function if the magnitude of fitting parameters and magnitude of lambda are high. The regularization parameter lambda is a control on your fitting parameters, which is again hyperparameter need to be tuned.

Reason for SGD as final model:

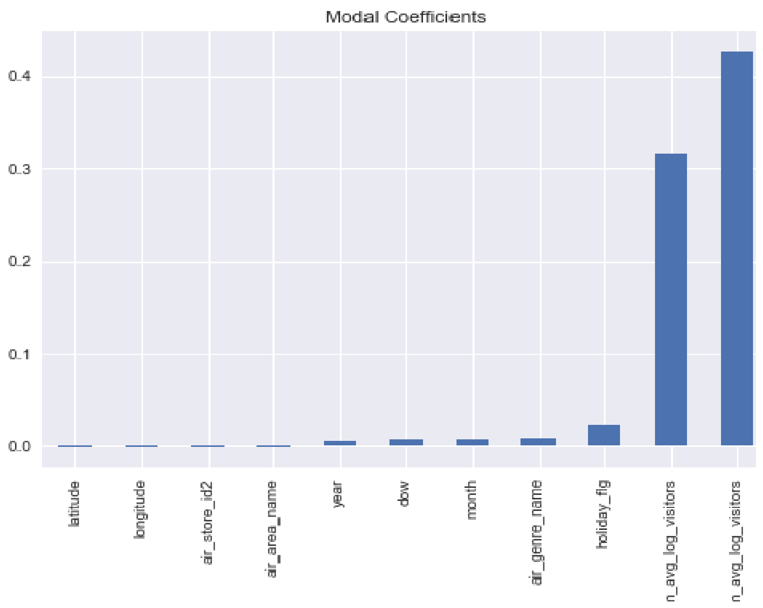
- The major advantage of SGD is its efficiency, which is basically linear in the number of training examples. If X is a matrix of size (n, p) training has a cost of O(k n \bar p), where k is the number of iterations (epochs) and \bar p is the average number of non-zero attributes per sample.
- Since it is faster than other algorithms, which helps in tuning the hyper parameters using grid search cross validation to find out best combination of hyperparameters.
- It uses gradient descent to optimize the algorithm. The performance can be improved by training the data on different epochs, on every epochs the error is descending to lower value. This parameter is tuned using grid search to find best number of iteration for this dataset so that error value descent to lower values. To avoid overfitting L2 regularization is used.

Implementation:

SGD is implemented using Sklearn library and the model is trained and validated using default values of hyper parameter.

Performance of the model is measured using RMSE metrics and the values of RMSE for 5 fold cross validations are

```
Mean squared error using kfold [0.09025091174936287, 0.09141443765820627,
0.0909727998830593, 0.08977863624349908, 0.0911687397178775]
Mean value 0.0907171050504
And the coefficient value of mapping functions are
```



KNearestNeighbor:

- It is a non-parametric model, which means it does not make any assumption about data distributions.
- KNN is also a lazy algorithm (as opposed to an eager algorithm).
- It means is that it does not use the training data points to do any generalization. In other words, there is no explicit training phase, or it is very minimal. This also means that the training phase is pretty fast.
- Lack of generalization means that KNN keeps all the training data.
- the training data is needed during the testing phase. KNN stores the entire training dataset which it uses as its representation. KNN does not learn any model.
- KNN Algorithm is based on feature similarity. KNN makes predictions just-in-time by calculating the similarity between an input sample and each training instance.
- The algorithm can be summarized as:
- A positive integer k is specified, along with a new sample. We select the k entries in our database which are closest to the new sample, using distance function like Euclidean distance, Manhattan distance.
- After finding k closest entries, take average of its output value and use that average as predicted value for that new sample.

Reason for Choosing the algorithm:

- The closest k sample for new sample would be highly relevant ie. The nearest data for no of visitors on specific day would be previous days no of visitors, and its previous month day of week average which are more appropriate for predicting the current day visitor count. The no of

Neighbors is hyperparameter and it is given by user while training the parameter. The right number will make the model less biased and variance.

Pros:

- Insensitive to outliers — accuracy can be affected from noise or irrelevant features
- No assumptions about data — useful, for example, for nonlinear data
- Simple algorithm — to explain and understand/interpret

Cons:

- Computationally expensive — because the algorithm must compute the distance between new sample and each training sample to find the closest training sample's output. So Prediction stage might be slow (with big N)
- High memory requirement -Stores all (or almost all) of the training data
- Sensitive to irrelevant features and the scale of the data. To avoid that feature scaling to be done.

Implementation:

- KNN is implemented using Sklearn library and the model is trained and validated using 8 nearest neighbors hyper parameter.
- Performance of the model is measured using RMSE metrics and the values of RMSE for 5-fold cross validations are
- Hyper parameters: In kNN is trained using following values for hyperparameters,
  - N-neighbors – 8, after computing distance between new data and all training data, it sort list five smallest distance and consider those eight data points, it computes average of eight data points visitor's count and use that as predicted value for that new data. And the process repeats for all testing data.
  - I implement with default value for n-neighbors but the RMSE was higher than N-neighbors-8

Output:

```
o Mean squared error using 5fold [0.09176238525640379, 0.093010070128
  14008, 0.09187554072548273, 0.09096208920455963, 0.09258394140104
  208]
o Mean of rmse of KNN using 5fold 0.0920388053431
```

SVM regression:

- 1.SVM regression works by mapping the data x into a high-dimensional feature space F via a nonlinear mapping (Kernel tricks), and to do linear regression in this space.

- Finding best hyper plane with large margin which is - epsilon insensitive zone ie trying to find the best fit so that points are closer to plane and within the zone.
- SVM regression performs linear regression in the high-dimension feature space using epsilon -insensitive loss and, at the same time, tries to reduce model complexity by minimizing ||w||2. This can be described by introducing (non-negative) slack variables, to measure the deviation of training samples outside -insensitive zone epsilon.
- It is well known that SVM generalization performance (estimation accuracy) depends on a good setting of meta-parameters parameters C, and the kernel parameters.
- Parameter C - determines the tradeoff between the model complexity (flatness) and the degree to which deviations larger than are tolerated in optimization formulation for example, if C is too large (infinity), then the objective is to minimize the empirical risk only, without regard to model complexity part in the optimization formulation.
- Parameter epsilon - controls the width of the epsilon-insensitive zone, used to fit the training data. The value of can affect the number of support vectors used to construct the regression function. The bigger, the fewer support vectors are selected. On the other hand, bigger -values results in more 'flat' estimates. Hence, both C and -values affect model complexity (but in a different way).

Pros:

- Kernel based framework is very powerful and flexible.
- SVM work well with small training examples.

Cons:

- Computational complexity: During training time, it has to do matrix computation (inner product) to find similarity between data points to find the support vectors by using implicit function to convert data points to into high dimensional feature space.
- Learning can take very long time for large scale problem. http://kernelsvm.tripod.com/#

Implementation:

- SVM is implemented using Sklearn library and the model is trained and validated using Default values of hyper parameters
  - kernel='rbf' : kernel function uses implicit mapping function to map input to high dimensional space and compute the similarity. If K is large, then it means two input vectors in feature space is similar.
  - epsilon=0.01 : It specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value.

- C =1 - *C* is the box constraint, a positive numeric value that controls the penalty imposed on observations that lie outside the epsilon margin (ε) and helps to prevent overfitting (regularization).

- Performance of the model is measured using RMSE metrics and the values of RMSE for SVR cross validations are

```
Mean squared error using SVR [0.0897648365416425, 0.0910014682407444
8, 0.09045240480246867, 0.08950414992633948, 0.09076090617375704]

Mean of rmse of SVM 0.090296753137
```

Random Forest:

- Random forest builds multiple CART model with different sample and different initial fe atures. For instance, it will take a random sample of 100 observation and 5 randomly cho sen initial features to build a CART model. It will repeat the process (say) 10 times and th en make a final prediction on each observation. Final prediction is a function of each pred iction. This final prediction can simply be the mean of each prediction.
- Each tree is built by randomly chosen features and observation, use attribute selection stat istic like Gini index or information gain to compute the values for each attribute by rando mly choosing split value to convert into category if it is real value. Then sort the values a nd the attribute which has highest value (in case of information gain) will be selected as r oot for the tree and split the training data into subset. Then the attribute which has next hi ghest value will be child node and process repeats until leaf node has reached.
- No of trees to build is hyperparameter which can be tuned.

Implementation:

- Using ensemble class of Sklearn library built 10 decision trees by randomly selecting features and observations.
- Performance of the model is measured using RMSE metrics and the values of RMSE for RF using 5-fold cross validations are

```
Mean squared error using 5fold CV [0.09660743075493328, 0.0975276344
6247971, 0.09721947045096051, 0.09650572518620514, 0.097461137798146
64]
 Mean of rmse of Random Forest 0.0970642797305
```

Comparison:

After evaluating performance of all the models, the above four models perform better than benchmark model. Among these models, Stochastic gradient descent and support vector regression model have low error value compared to other models. So, I have decided SGD regressor to be my final model for this problem because of its efficiency and speed.

Refinement:

- Using Grid search cross validation, tuning the hyperparameter of the model
    - Penalty to avoid overfitting

- alpha which controls the magnitude of the coefficient of the features
- learning rate
- no of epochs.
- `parameters={'penalty':('l1','l2'),'alpha':[0.001, 0.01, 0.1, 1],'n_iter':[5,10,15],'eta0':[0.01,0.1,1.0]}`
- After training model with different values of hyper parameter, the grid search cv helps to find the best combination of the hyperparameters which has less loss on the validation data.

Output:

Result of `grid_search.best_params_`

```
{'alpha': 0.001, 'eta0': 0.01, 'n_iter': 15, 'penalty': 'l2'}
```

Evaluating the performance of Stochastic gradient descent regressor with test data:

- After finding the best values of hyperparameter, train the Stochastic Gradient descent with below values of hyper parameter
  {'alpha': 0.001, 'eta0': 0.01, 'n_iter': 15, 'penalty': 'l2'}
  with whole training data and predict the values of test data. Then measure the evaluation metrics ie Root mean square error.

- RMSE of refined SGD model is 0.0909269637974

Justification:

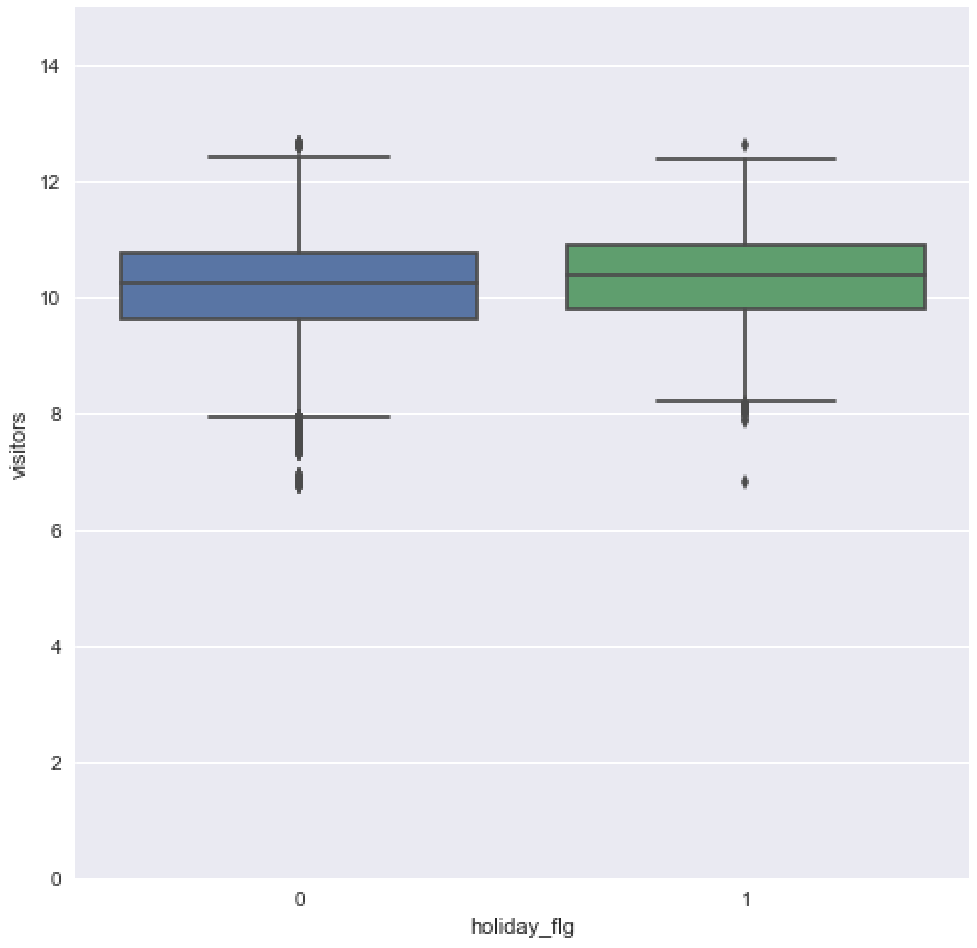
- The performance of model has improved after tuning the hyper parameters. Because the model perform well at 15 iteration. At 15th iteration the model finds best fit line that fit our data better (where better is defined by our error function) will result in lower error values.
- Initially the model finds mapping function with random values of weights then compute the difference between actual and predicted.
- To better approximate the actual value, error function is used which is defined by the parameters. Need to adjust the parameters to get better result which is done using gradient descent.
- Each iteration it updates the value of parameters, it takes 15 iterations to minimize the error.
- Alpha is lambda in regularization, if the value is small then that will reduce the magnitude of weight which in turn reduce the loss.
- Learning rate: 0.01 , it chose the values which is reasonable because it should not be too small like 0.001 which takes lot of iteration to descent and it won’t reach global minima or too big like 1 or 10., then it will overshoot at each iteration again reaching global minima it takes long iterations.
- line. Each iteration will update m and b to a line that yields slightly lower error than the previous iteration. The direction to move in for each iteration is calculated using the two partial derivatives.

Conclusion

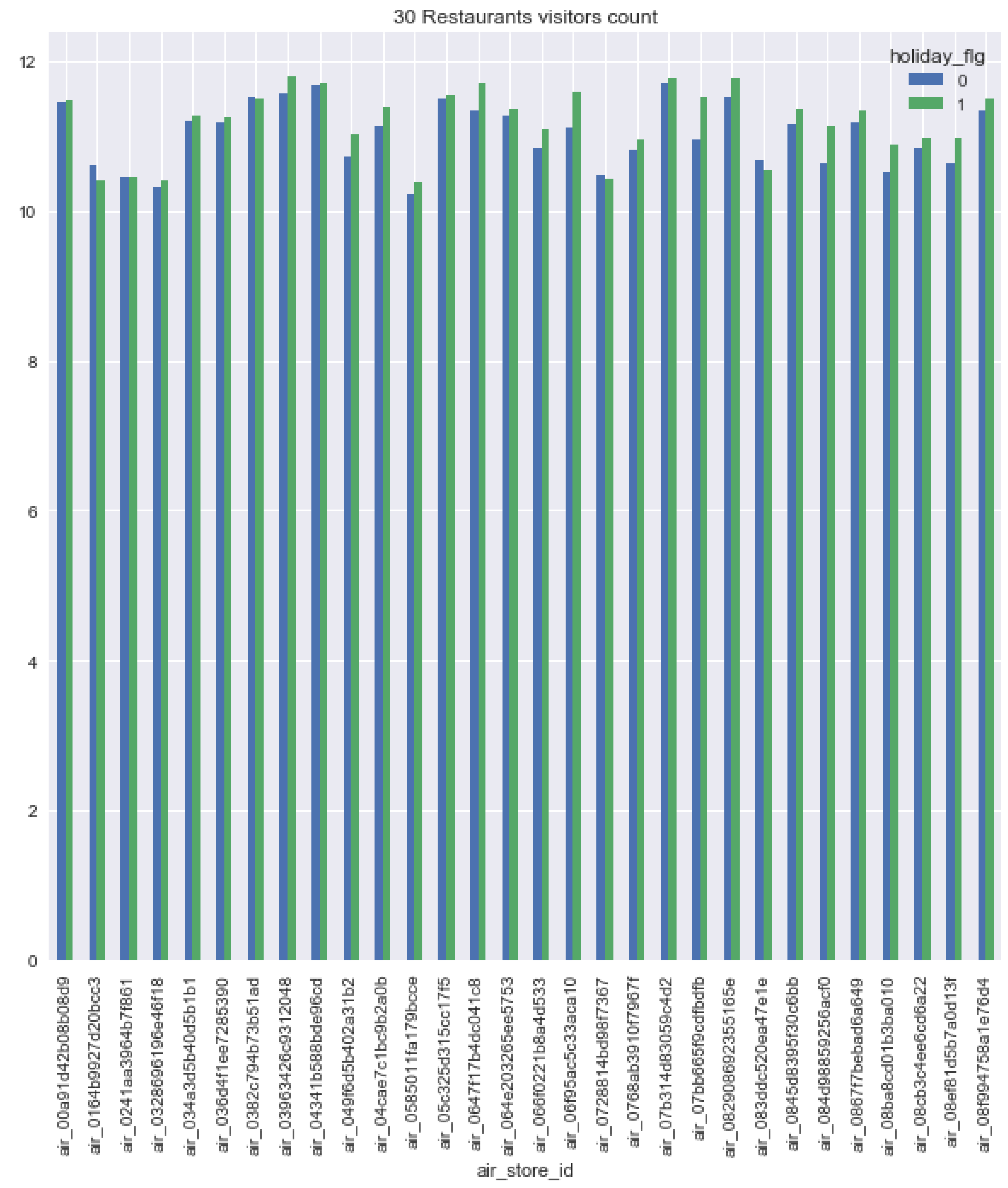
Forecasting the unseen data with refined SGD Regressor:

1. To predict the no of visitors of unseen data then model should be trained with entire training data, so that the model will have enough data to find better parameters of mapping function which then generalize well for unseen data.

Free Form Visualization:



The box plot shows that model captures the relationship between visitors count and holiday flag and it shows that during holiday the average visitors are high than working day .



Reflection

The process used for this project can be summarized as follows:

1. Understanding the problem and gaining knowledge of the problem domain
2. Downloading the data and understanding the dataset

3. Performing feature engineering and preprocessing the data
4. Converting timeseries to supervised learning problem
5. Handling the missing values
6. A bench mark model – naïve regressor was created for this problem
7. Implementing supervised machine learning models
8. Validate and evaluate the performance of model using Kfold cross validation and evaluation metrics RMSE
9. Tuning the hyperparameters of Stochastic gradient descent regressor model and find the optimal parameters. And train the refined model with training data and measure the performance.
10. When the performance of the model has improved, then train the model with the entire training data and the model is ready for the forecast.

I found step 4, 5 and 6 are challenging for me. To do step 4 and 6 made myself familiarize with domain knowledge and dataset. To do step 5, this article helped to approach timeseries as supervised learning problem.

During the process of converting timeseries into supervised learning, it would be more relevant if I use previous year day of week monthly average but data for previous year was not available so I initially used yearly day of week average but the model didn't give good prediction, figuring out previous time step and its implementation was challenging to me.

https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/

Given dataset is not enough to find previous year visitor count, so I have taken previous month as previous timestep to convert the timeseries to supervised learning.

Improvement:

- To achieve optimal experience and better performance I would train the data with SVM with capable hardware, Since SVM performed well with default values of hyper parameter. But it is slower than the bench mark model. To tune those hyper parameters, I may need better hardware to perform fast.
- The dataset didn't include weather feature which also influence no of visitors. I would download weather forecast of 2016 and 17 which tells about that day is rainy, windy, sunny and use that data to predict the no of visitors.
- In future if I am able to get large amount of data for this problem, I may apply deep learning like LSTM Recurrent neural network which perform well for time series dataset.