Machine Learning Engineer Nanodegree

Capstone Project Report

Naga Venkata Harisyam Manda

January 24th, 2018

Recruit Restaurant Visitor Forecasting using Stacking

1. Definition

1.1 Project Overview

Time Series is defined as an ordered sequence of values of a variable at equally spaced time intervals [1]. Time series problems are frequently encountered when monitoring industrial data or the data from a process running in time. Time series analysis comprises methods for analyzing time series data to extract meaningful statistics and other characteristics of the data. Time series forecasting is the use of a model to predict future values based on previously observed values. [2,9]

Time series forecasting is being widely used in many fields:

- Sales Forecasting of a product in an Industry
- Budgetary Analysis of a Firm
- Stock Market Analysis
- Yield Projections in a Chemical Plant
- Process and Quality Control in Automated Manufacturing Plant
- Inventory Studies of a Grocery Store or a restaurant
- Workload Projections on employees
- Damage forecasting of a Vehicle fleet
- Forecasting the average price of gasoline in a city each day

Even though it is being widely used, the implementation of Machine learning techniques in time series forecasting is not very widely popular because of its complexity. The inherent time dependence of the features and the components (trend, seasonality etc.) of time series cannot be easily observed [3,4]. But in the recent years, due to the development of deep learning models, enabled researchers to use different deep networks for performing time series forecasting. From the knowledge I gained from Machine learning nanodegree, I want to extend it a bit further to do time series forecasting. I also do time series forecasting for my Master thesis using the conventional ARIMA models and Holt winter methods, but the conventional models are slow for large datasets. Therefore, I want to use machine learning methods like LSTM's and other supervised learning techniques, for example LightGBM (if the Time series problem is framed as a regression problem) for performing time series forecasting. I would also like to check how efficient the ML models could be when compared to the traditional methods.

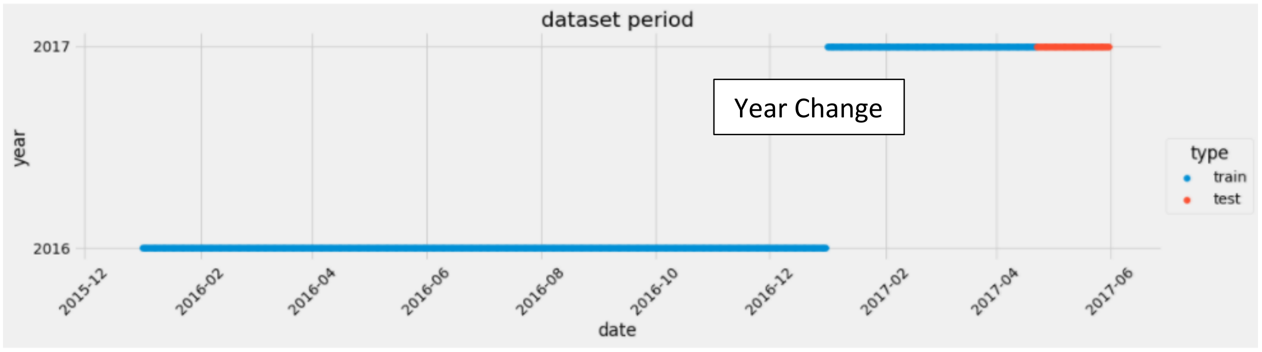
1.2 Project Origin

I had participated in an online competition at Kaggle: https://www.kaggle.com/c/recruit-restaurant-visitor-forecasting. The problem statement is as follows: Running a thriving local restaurant isn't always as charming as first impressions appear. There are often all sorts of unexpected troubles popping up that could hurt business. One common predicament is that restaurants need to know how many customers to expect each day to effectively purchase ingredients and schedule staff members. This forecast isn't easy to make because many unpredictable factors affect restaurant attendance, like weather and local competition. It's even harder for newer restaurants with little historical data.

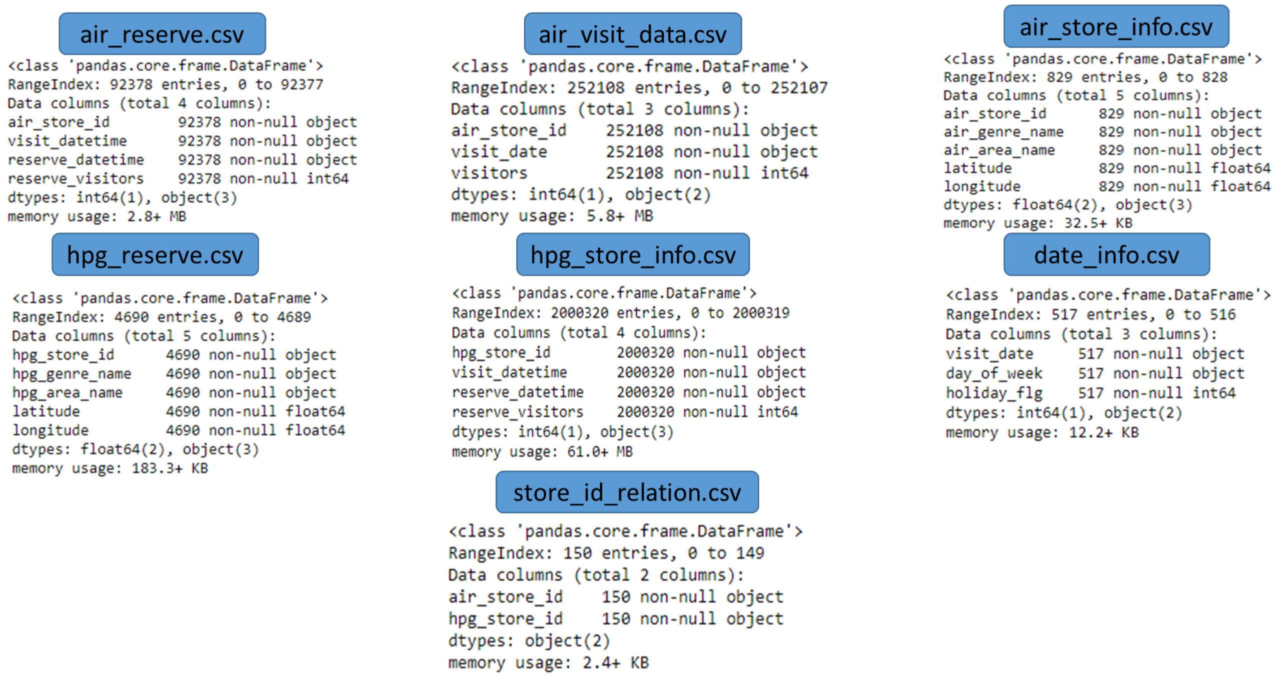
Recruit Holdings has unique access to key datasets that could make automated future customer prediction possible. Specifically, Recruit Holdings owns Hot Pepper Gourmet (a restaurant review service), AirREGI (a restaurant point of sales service), and Restaurant Board (reservation log management software). In this competition, we are challenged to use the reservation and visitation data to predict the total number of visitors to a restaurant for future dates. This information will help the restaurants be much more efficient and allow them to focus on creating an enjoyable dining experience for their customers [10].

1.3 Datasets and Inputs

The datasets can be found here: https://www.kaggle.com/c/recruit-restaurant-visitor-forecasting/data. The target variable in the dataset is the **number of visitors for different restaurants**. The forecasting must be performed for all the restaurants in the submission.csv found in the link above. The training dataset has data from 1st January-2016 to 23rd April 2017, and the test set is from 23rd April 2017 to 31st May 2017.



From the training set: as we have time series data it is suggested to use the **Time Series split** [14] for creating the actual training and validation sets. The date range could be as: **Actual Training set**: from January-2016 to December-2016 & **Validation set:** from January-2017 to mid-April-2017. But the proportions of actual train and validation set might vary for the Timeseries cross-validation (mentioned in the subsequent section). The test set is already predefined in the competition. The actual training and validation sets are not created by the normal random split because if the random samples are picked as for the validation set, we might introduce look ahead bias because here the data is sequentially considered [13]. All the information in the dataset will be used to create new features in the feature engineering step. The brief overview of what each data file contents, and their respective data formats are shown below in a tabular format.



air_reserve.csv

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 92378 entries, 0 to 92377
Data columns (total 4 columns):
air_store_id        92378 non-null object
visit_datetime      92378 non-null object
reserve_datetime    92378 non-null object
reserve_visitors    92378 non-null int64
dtypes: int64(1), object(3)
memory usage: 2.8+ MB
```

air_visit_data.csv

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 252108 entries, 0 to 252107
Data columns (total 3 columns):
air_store_id    252108 non-null object
visit_date      252108 non-null object
visitors        252108 non-null int64
dtypes: int64(1), object(2)
memory usage: 5.8+ MB
```

air_store_info.csv

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 829 entries, 0 to 828
Data columns (total 5 columns):
air_store_id      829 non-null object
air_genre_name    829 non-null object
air_area_name     829 non-null object
latitude          829 non-null float64
longitude         829 non-null float64
dtypes: float64(2), object(3)
memory usage: 32.5+ KB
```

hpg_reserve.csv

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4690 entries, 0 to 4689
Data columns (total 5 columns):
hpg_store_id      4690 non-null object
hpg_genre_name    4690 non-null object
hpg_area_name     4690 non-null object
latitude          4690 non-null float64
longitude         4690 non-null float64
dtypes: float64(2), object(3)
memory usage: 183.3+ KB
```

hpg_store_info.csv

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000320 entries, 0 to 2000319
Data columns (total 4 columns):
hpg_store_id        2000320 non-null object
visit_datetime      2000320 non-null object
reserve_datetime    2000320 non-null object
reserve_visitors    2000320 non-null int64
dtypes: int64(1), object(3)
memory usage: 61.0+ MB
```

date_info.csv

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 517 entries, 0 to 516
Data columns (total 3 columns):
visit_date     517 non-null object
day_of_week    517 non-null object
holiday_flg    517 non-null int64
dtypes: int64(1), object(2)
memory usage: 12.2+ KB
```

store_id_relation.csv

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 2 columns):
air_store_id    150 non-null object
hpg_store_id    150 non-null object
dtypes: object(2)
memory usage: 2.4+ KB
```

Figure 1-1: Contents of the data files and null-counts of each feature

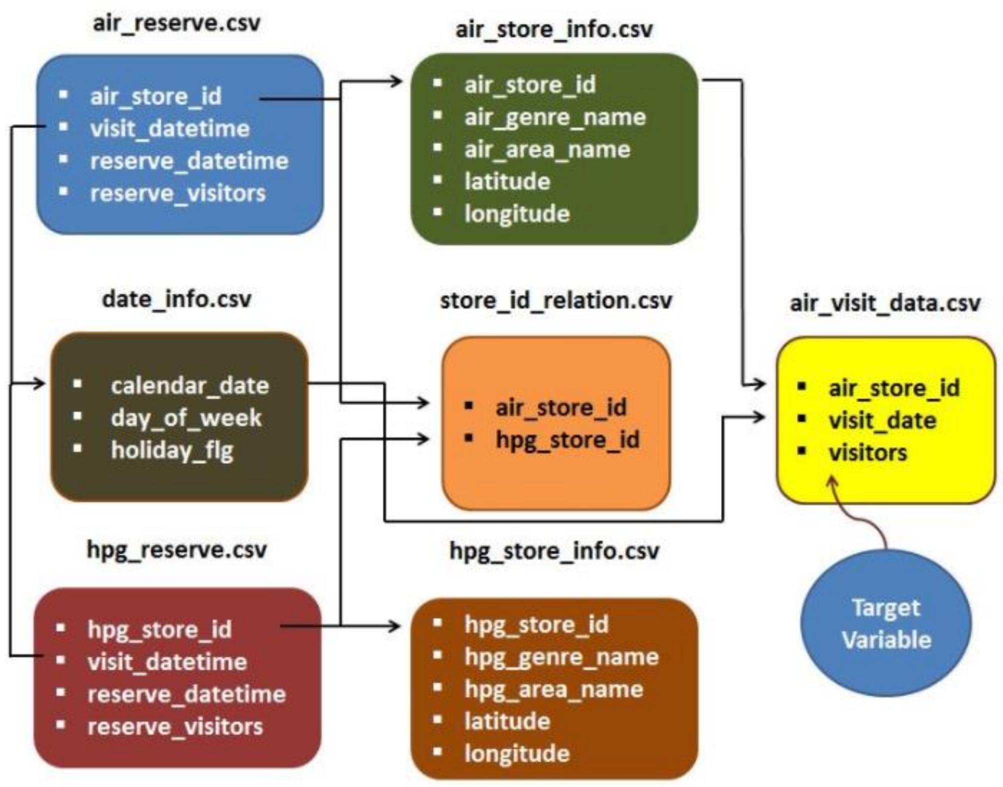


Figure 1-2: File relations [11]

All the files are perfectly formatted with no null values in any feature of the data file. The data file descriptions and the feature descriptions are in the following table.

S.No	File Name	File Contents (features)	File Description
1	air_reserve.csv	• **air_store_id** - the restaurant's id in the airREGI 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	This file contains reservations made in the airREGI system. The reserve_datetime indicates the time when the reservation was created by user, whereas the visit_datetime is the time in the future where the visit will occur.
2	air_visit_data.csv	• **air_store_id** - the restaurant's id in the airREGI system • **visit_date** - time series information • **visitors** - the number of visitors to the restaurant on a particular date. **This is the target variable for the problem**	This file contains historical visit data for all the customers in the airREGI restaurant system.
3	air_store_info.csv	• **air_store_id** - the restaurant's id in the airREGI system • **air_genre_name** - airREGI restaurant genre / theme • **air_area_name** - air-restaurant's city + district + area • **latitude** – coordinate for the restaurant • **longitude** – coordinate for the restaurant	This file contains information about air restaurants like the genre and it's location in a particular with latitude and longitude co-ordinates
4	hpg_store_info.csv	• **hpg_store_id** - the restaurant's id in the HPG system • **hpg_genre_name** - HPG restaurant genre / theme • **hpg_area_name** – restaurant city+ district +area Location • **latitude** - coordinate for the restaurant • **longitude** - coordinate for the restaurant	This file contains information about various HPG restaurants like the genre and it's location in a particular with latitude and longitude co-ordinates.
5	hpg_reserve.csv	• **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	This file contains reservations made in the HPG system.
6	store_id_relation.csv	• hpg_store_id • air_store_id	This file allows to join the restaurants that are both existent in the airREGI and HPG system.
7	date_info.csv	• **calendar_date** – basically the visit date • day_of_week • **holiday_flg** – indicates a public holiday in Japan	This file gives basic information about the calendar dates in the dataset and also public holidays in Japan
8	sample_submission.csv	• **id** - the id is formed by concatenating the **air_store_id** and **visit_date** with an underscore • **visitors**- the number of visitors to be forecasted for the AirREGI restaurant on a particular date	This file shows a submission in the correct format, including the days for which must be forecasted. Basically the test-set for the problem

Figure 1-3: Data files and feature descriptions

The data files are to be merged for creating the data-sets for the AIR and HPG systems. To merge the air store information and air restaurant reservation information with its corresponding AirREGI visitors' data, the data files are to be merged on the air_store_id. Similarly, for the HPG restaurants the files are to be merged on the hpg_store_id. For generating a training set, the data files are to be merged with the use of store-id-relation file which has the link between HPG restaurants and the AirREGI restaurants. The date info file will help in noticing the public holidays in the data set and can be merged on the visit_date column. To show the complete file relations for better intuition, a pictorial representation is presented above in fig. 1-2.

2. Problem Statement

The data provided contains time series information for the number of visitors entering an Air-restaurant each day and the target is to forecast the number of visitors for each restaurant for 39 days (from 23rd April to 31st May 2017). This problem is also multivariate time-series forecasting problem because of the inherent dependence on time and it also contain important features related to the restaurants like genre, area where a restaurant is located (fig. 1-3) and the global weather conditions suitable for the visit.

Generally, conventional models like ARIMA are used to perform the forecast for a single time series problem. In this case 829 restaurants time series data with multiple features are present in the database, which implies that there are 829-time series problems for which the forecast must be performed. Since each restaurant performs differently when compared to others the ARIMA models must be tuned manually for each time series problem and this usually takes up lot of time. Even though there are automatic algorithms which perform the tuning, because of the linear nature of ARIMA models, they might not generalize very well when large number of dependent features (like genre, weather, etc.) are present. Hence, I would like to use supervised machine learning models which respect the nature of time series (sequence dependent) for obtaining the visitor forecasts for the test period of 39 days.

To achieve very good results, my idea was to use stacking of various baseline ML models (frequently used in kaggle competitions to achieve higher LB scores). The baseline models which I will train initially are: LSTM's, XGBoost, LightGBM, Random Forests, K-nearest neighbors, Multi-layer perceptron and Gradient Boosting Machine. The trained and cross-validated models will be stacked by using a meta-learner which would give out the final predictions for the number of visitors on a date for a restaurant. I would also investigate different possible stacking procedures mentioned in [7, 8]. My goal was to get better public and private leaderboard scores (which would increase my ranking).

3. Evaluation Metrics

The evaluation metrics used in the competition is **RMSLE**. Definition of RMSLE is: **R**oot **M**ean **S**quared **L**ogarithmic **E**rror, which is calculated as:

$$\sqrt{\frac{1}{n} * \sum_{i=1}^{n} (\ln(p_i + 1) - \ln(a_i + 1))^2}$$

Where:

n is the total number of observations

p_i is the prediction of visitors

a_i is the actual number of visitors

Lower the RMSLE value, better the forecast is determined by the model. RMSLE basically measures the ratio of the predicted and actual number of visitors. As this can be treated as extrapolated regression problem, metrics like MAE or RMSE can be used but RMSLE provides a much finer view on the Leaderboard scores. This finer control of RMSLE when compared to RMSE is because of the penalization capability. For example (obtained from [17])

- **Case-a:** p_i=600 and a_i =1000 (under-estimated) $\rightarrow$ RMSE = 20 and RMSLE = 0.5108
- **Case-b:** p_i=1400 and a_i =1000 (over-estimated) $\rightarrow$ RMSE = 20 and RMSLE = 0.3365

RMSE treats both the cases a and b equally whereas the RMSLE penalizes the under-estimate more. Hence the choice of RMSLE is better.

4. Analysis

For this project I have written 4 notebooks, which will give complete insight into the procedure for solving the problem. The four notebooks are:

- **1_recruit_restaurant_EDA** – for the exploratory data analysis of all the input files
- **2_recruit_restaurant_arima_final** – for the time series analysis using SARIMAX model (benchmark model)
- **3_recruit_restaurant_TSCV_or_kfold** – for the basic insights into the problem and exploring different cross validation strategies
- **4_recruit_restaurant_final** – for the baseline models with cross-validation and the stacked ensemble

In the following section, the inferences made from the EDA notebook will be presented.

4.1 Exploratory Data Analysis

The first step was to obtain data from the competition and extracting them out of the *.rar files. The initial exploration was carried out on the individual files and their respective statistics and the feature details are show in fig. 1-1 and fig. 1-3.

General Information: There were 829 air-restaurants in the air-visitors database and only 821 air-restaurants will be used for the predictions (because the sample_submission file had only 821 restaurants). The HPG files are merged to the air-restaurant databases by the help of store-id-relation file.

About the AIR-Restaurants: Extensive data exploration is performed to analyze the input files and their combinations (see fig. 1 above). Some important graphs and observations are shown here:

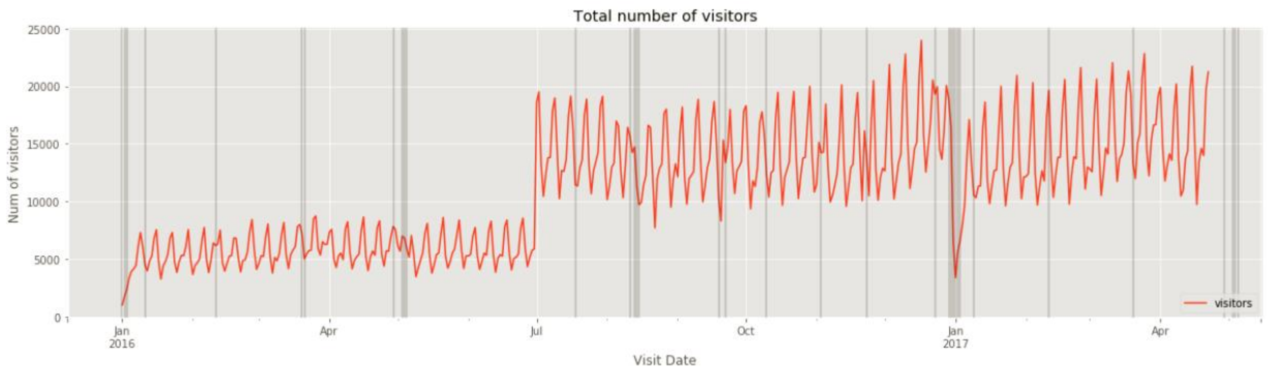


Fig. 4-1: Total number of AIR_restaurant visits with respect to holidays in Japan

There is a sudden increase of number of visitors in July and this is because of addition of new restaurants to the AIR visits database [appendix fig. 8-1]. The golden week is on the same days in the year 2016 and in the year 2017.

Golden Week in 2016 and 2017 - 29th April to 5th May. These dates are very import in the cross-validation strategies. The number of reserve visitors are zero from June to October in the Air-Reservation system may be due to some error in the database (see fig. 4-2 below). The number of reservations had a spike in last week of December 2016 mainly due to Christmas week. On an overall perspective there is a weekly patter in reservations and visits.

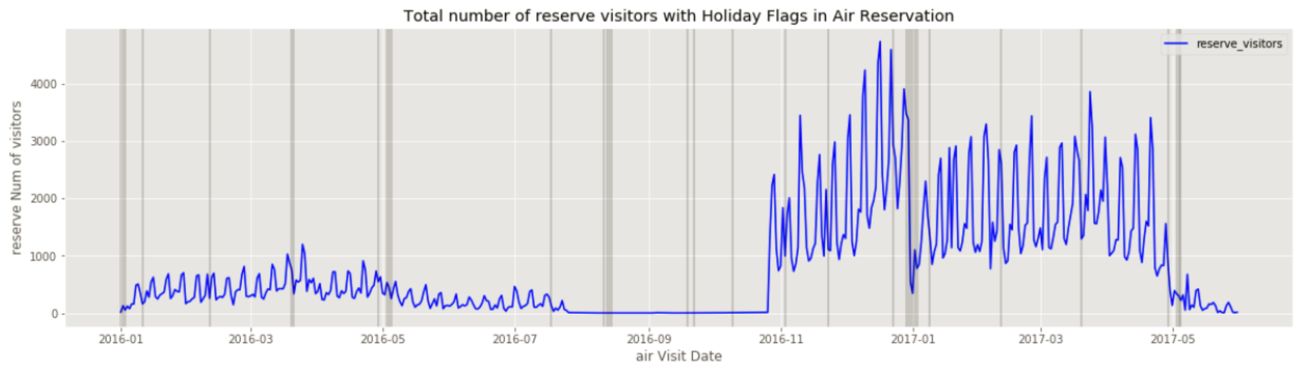


Fig. 4-2: Air Reservations with respect to holidays in Japan

Comparing the visitors from the air reservations and the total visitors in the air visits gives insight that the reservation services provide on an average of 25% of the total clientele for the AIR-Restaurants.

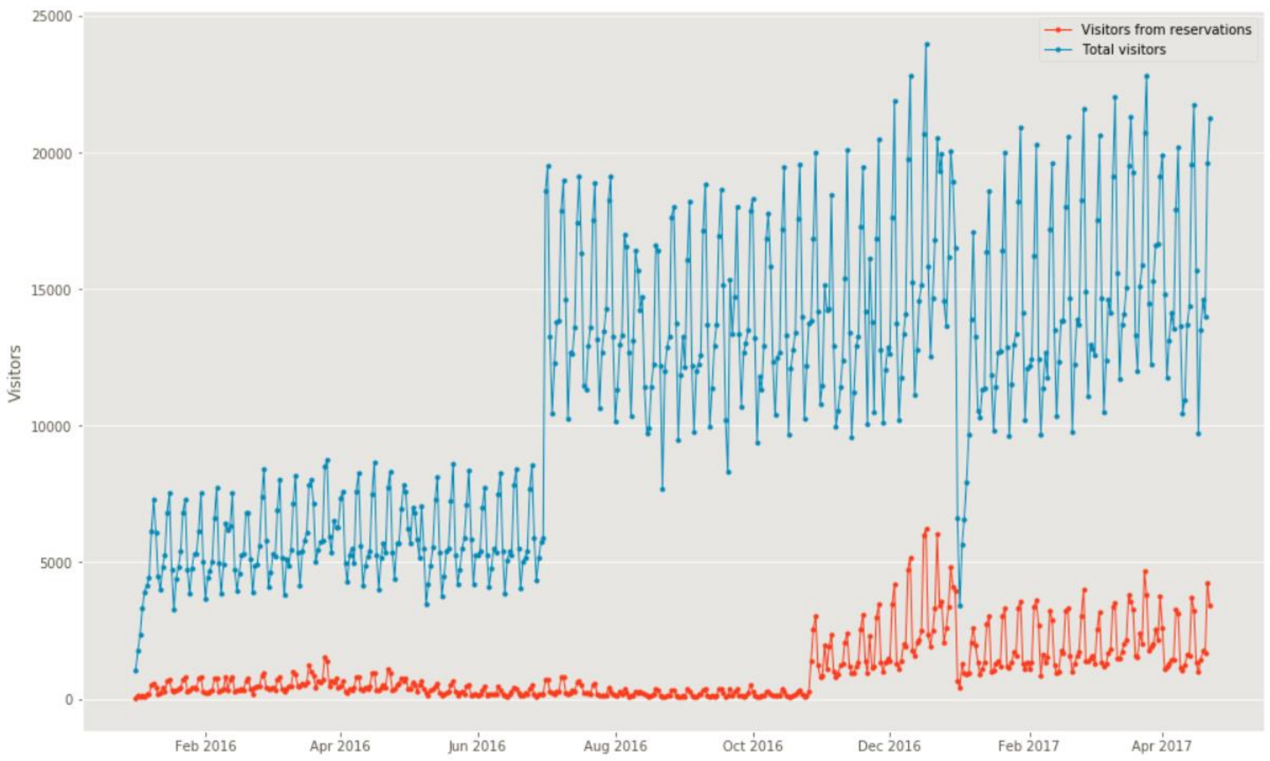


Fig. 4-3: Air-Restaurants Total visitors vs. Visitors from the reservations

Highest number of air-restaurants are present in Tokyo area and the Fukuoka area [fig. 4-4]. The most popular genre in AIR database is **Izakaya**. There are too many genres, hence I had to group them into smaller number of genres in the data-preprocessing step. Other EDA plots for the AIR-restaurants are available in the notebook.

About the HPG-Restaurants: The HPG database do not monitor the visitors but rather they monitor only the reservations. The total reservation in HPG database with respect to the holidays is shown in fig. 4-3

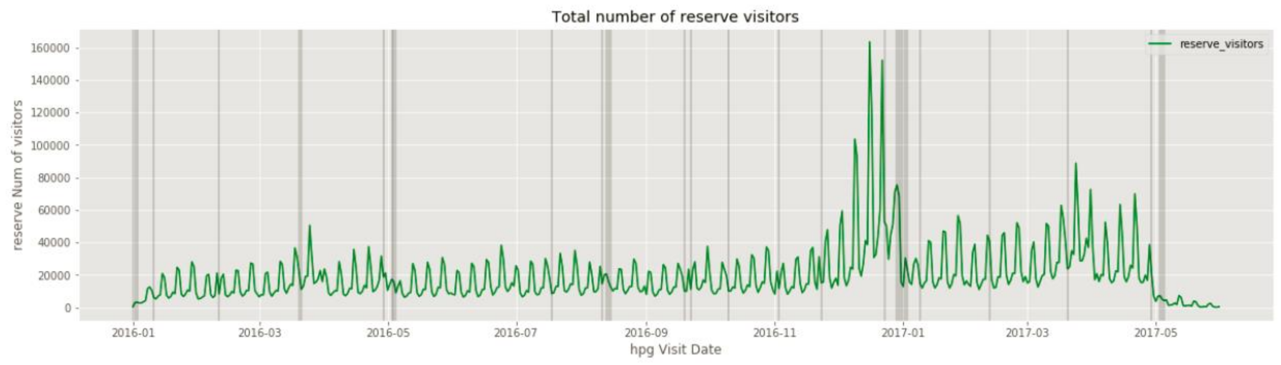


Fig. 4-4: HPG reservations with respect to holidays

The HPG reserves also show a weekly pattern which is similarly found in the air-reservations. From fig. 4-2, it can be observed that the number of reservations in HPG were more in number when compared to the air-reservations. Highest number of HPG restaurants are found in Tokyo area [fig. 4-4] and the most popular genre is the **Japanese style.**

Another interesting observation is that the number of reservations in AIR-reserve database are more in the last few hours before the visit but in the HPG-reserve there is a patter for long term reservations. This detail could be used to filter the reservation data when including these features in the final data-frame. Other EDA plots for the HPG-restaurants are available in the notebook. I am not including the plots because there are too many plots and the aspect ratio is very big. (Notebook-1 for all the plots)

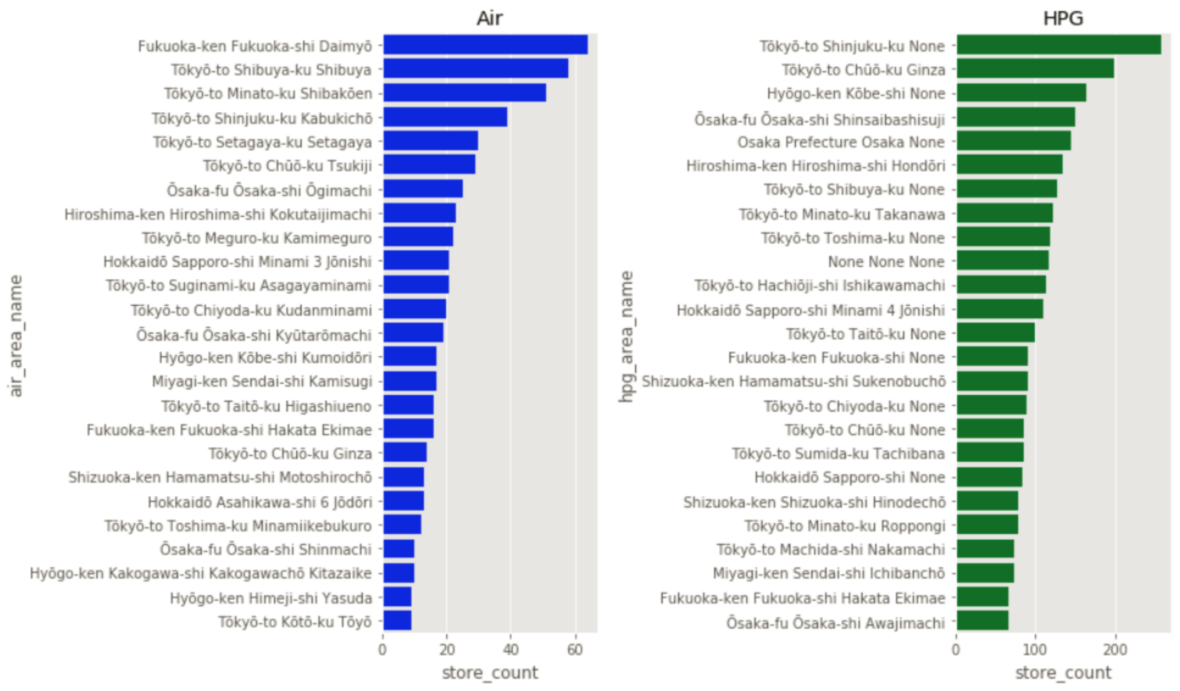


Fig. 4-5: Number of AIR-restaurants and HPG-restaurants wrt. Area

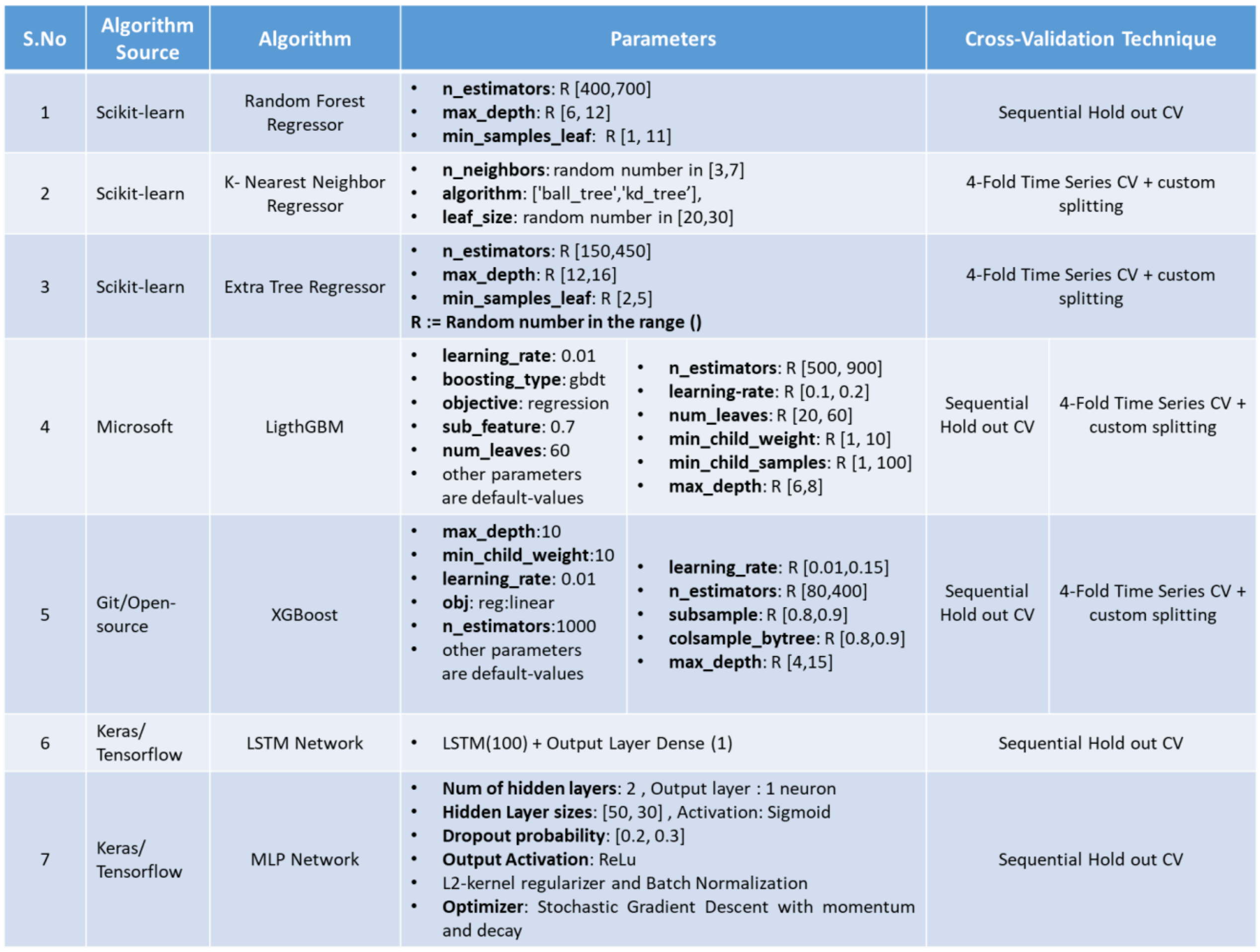
4.2 Algorithms and Techniques

At the first step after the data exploration, the data files were merged based on the relations shown in fig. 1.2. The merged table had been all the information available related to the reservations, air-visitors, genres and area of every restaurant in the database. There were missing values in the final merged data frame, which were because of the holidays or because the HPG reservation system has more data than the AIR system. Hence, treating the missing values was an important step because dropping the rows with missing values since they were not present in air-database could basically eliminate some crucial information about the restaurants. Therefore, the missing values are filled with an extreme value (an outlier), this could create trouble in the learning phase, but it would be reduced with the time-series cross-validation techniques.

For the cross-validation strategies, I was not very sure initially which one to implement, i.e. either a K-fold Time series cross validation, a Sequential hold-out or a K-Fold cross-validation. I was sure that the conventional K-fold CV technique would not work in this problem, because this problem was sequence dependent. To enable comparison, I had created models with all the cross-validation techniques. Even though the cross-validation improves the scores of various algorithms, without proper features explaining the underlying phenomenon between the outputs and inputs, the models would still suffer in performance. Hence, feature engineering had been performed. Initially I have extracted all the temporal and spatial information from the 'visit_date' (for e.g. extracting month, week, weekend, day, etc.) and the 'air_area_name' (for e.g. extracting city name, district name, locality name etc.) respectively. I had also performed some advanced feature engineering and they will be discussed in the methodology section. This kind of feature engineering is performed as this time-series problem is posed as regression problem, the dataset should contain all the features which can affect the number of visitors for a restaurant based on the time aspect.

To create a stacked ensemble for this problem, I had to train various baseline models present in scikit-learn library along with the libraries like: XGBoost ensemble method, Light GBM ensemble method and Long Short-Term Memory (LSTM) networks, MLP Regressor. The basic rule for making a stacked-ensemble shine is to create diversity in the baseline models. For creating diversity, I had to use Random Grid Search along with a cross-validation technique.

The reasons behind choosing the random Forests algorithm as one of the baseline algorithm were: it is an ensemble algorithm which can capture non-linear effects in the data very well and it runs in parallel mode, saving lot of time. Another baseline algorithm was K- Nearest neighbor Regressor, which is a simpler technique and was chosen because it requires less computing time and uses Parzen windows for modeling when compared to random forests which uses decision trees (this is a stark contrast which would help in diversity of the base learners). The extra-trees Regressor generally implements a meta-estimator that fits randomized decision trees on various subsamples of the data and it is a slight modification to the random-forests and I think this could create some diversity. I had also used XGBoost instead of the conventional Gradient Boosting Machine algorithm because of XGBoost is more efficient and faster. I had also added LightGBM to my baseline models because it uses decision trees where the tree growth is based on number of leaves which is different when compared to XGBoost where the growth and splitting is based on the depth of the tree. This method could make LightGBM over-fit but proper cross-validation and specifying the max_depth parameter could increase performance and reduce over-fitting. I also added LSTM and MLP's to my baseline models, which would increase diversity. I chose basic parameters for LSTM and for the MLP I carefully chose the parameters and layers. All the algorithms and their parameters which were used are presented as a summary-table in fig. 4-5. As mentioned earlier to create diversity in the baseline models, I had used different parameter ranges for the models. The parameter range is represented with letter 'R' following the limits, for example, the n-estimators parameter which will be chosen randomly by the Random-Grid search function is shown as 'R [400,700]' in the figure below.



S.No	Algorithm Source	Algorithm	Parameters		Cross-Validation Technique	
1	Scikit-learn	Random Forest Regressor	• **n_estimators**: R [400,700] • **max_depth**: R [6, 12] • **min_samples_leaf**: R [1, 11]		Sequential Hold out CV	
2	Scikit-learn	K- Nearest Neighbor Regressor	• **n_neighbors**: random number in [3,7] • **algorithm**: ['ball_tree','kd_tree'], • **leaf_size**: random number in [20,30]		4-Fold Time Series CV + custom splitting	
3	Scikit-learn	Extra Tree Regressor	• **n_estimators**: R [150,450] • **max_depth**: R [12,16] • **min_samples_leaf**: R [2,5] **R := Random number in the range ()**		4-Fold Time Series CV + custom splitting	
4	Microsoft	LigthGBM	• **learning_rate**: 0.01 • **boosting_type**: gbdt • **objective**: regression • **sub_feature**: 0.7 • **num_leaves**: 60 • other parameters are default-values	• **n_estimators**: R [500, 900] • **learning-rate**: R [0.1, 0.2] • **num_leaves**: R [20, 60] • **min_child_weight**: R [1, 10] • **min_child_samples**: R [1, 100] • **max_depth**: R [6,8]	Sequential Hold out CV	4-Fold Time Series CV + custom splitting
5	Git/Open-source	XGBoost	• **max_depth**:10 • **min_child_weight**:10 • **learning_rate**: 0.01 • **obj**: reg:linear • **n_estimators**:1000 • other parameters are default-values	• **learning_rate**: R [0.01,0.15] • **n_estimators**: R [80,400] • **subsample**: R [0.8,0.9] • **colsample_bytree**: R [0.8,0.9] • **max_depth**: R [4,15]	Sequential Hold out CV	4-Fold Time Series CV + custom splitting
6	Keras/ Tensorflow	LSTM Network	• LSTM(100) + Output Layer Dense (1)		Sequential Hold out CV	
7	Keras/ Tensorflow	MLP Network	• **Num of hidden layers**: 2 , Output layer : 1 neuron • **Hidden Layer sizes**: [50, 30] , Activation: Sigmoid • **Dropout probability**: [0.2, 0.3] • **Output Activation**: ReLu • L2-kernel regularizer and Batch Normalization • **Optimizer**: Stochastic Gradient Descent with momentum and decay		Sequential Hold out CV	

Fig. 4-5: Number of AIR-restaurants and HPG-restaurants wrt. Area

4.3 SARIMAX Benchmark model

The benchmark which I used for the project is the SARIMAX or Seasonal ARIMA with exogenous variables model from the statsmodel library. Initially, I thought of using random Forests, but the previous reviewer suggested may be the benchmark model should be a linear model, therefore I changed it to ARIMA model. The idea was to analyze the 829 restaurants as separate time-series problems. To show the SARIMAX model development, I will use an example restaurant and the process explained below is applied for all the restaurants with no variation in parameters because this is a benchmark and a naïve model. For example, for the restaurant- air_ba937bf13d40fb24, if the air visits are plotted over the time-period, there are missing values because of the holidays or because there is a failure in the recording system or the recording system was not in place.

The fix for the gaps in the data is to resample the data. After resampling the time series plot for the restaurant in fig. 8-5 is shown below. There are spikes because of the linear interpolation between the data. This was just a naïve idea to fill up the gaps.

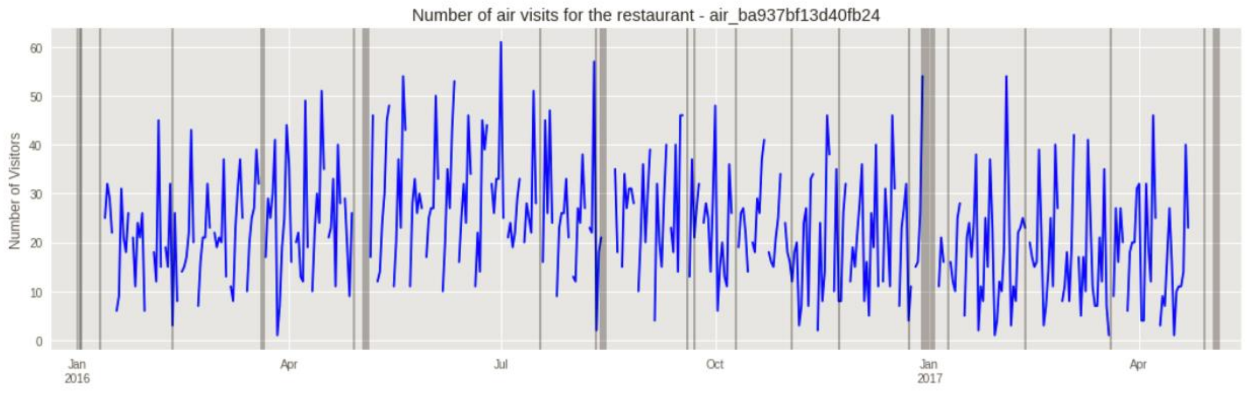


Fig. 4-6: Time series plot for a restaurant- air_ba937bf13d40fb24 (gaps in the data)

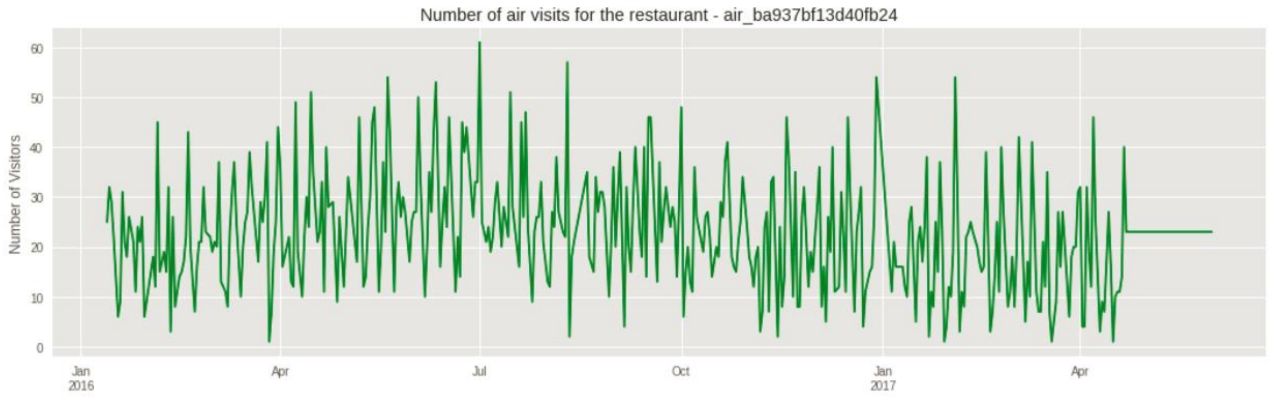
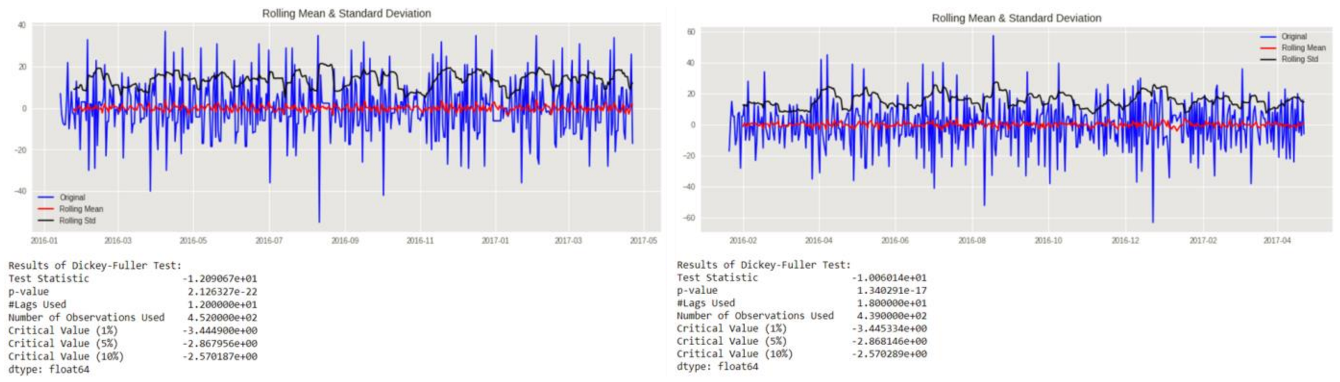


Fig. 4-7: Resampled Time series plot for restaurant- air_ba937bf13d40fb24

Now I tested the time-series stationarity using the Dickey-fuller test for the restaurant- air_ba937bf13d40fb24 and found out that the 1st-differencing and 7-day seasonal difference are yielding stationary time series. The stats of the test are shown below:



```
Results of Dickey-Fuller Test:
Test Statistic                -1.209067e+01
p-value                        2.126327e-22
#Lags Used                     1.200000e+01
Number of Observations Used    4.520000e+02
Critical Value (1%)           -3.444900e+00
Critical Value (5%)           -2.867956e+00
Critical Value (10%)          -2.570187e+00
dtype: float64
```

```
Results of Dickey-Fuller Test:
Test Statistic                -1.006014e+01
p-value                        1.340291e-17
#Lags Used                     1.800000e+01
Number of Observations Used    4.390000e+02
Critical Value (1%)           -3.445334e+00
Critical Value (5%)           -2.868146e+00
Critical Value (10%)          -2.570289e+00
dtype: float64
```

Fig. 4-8: 1st difference (left) and seasonal-7-day difference (right)

I have used SARIMAX (p,d,q)(P,D,Q,m) model with exogenous variables, these variables are developed by doing some feature engineering. To determine the parameters of the SARIMAX model the following Autocorrelation and partial-autocorrelation plots are plotted:

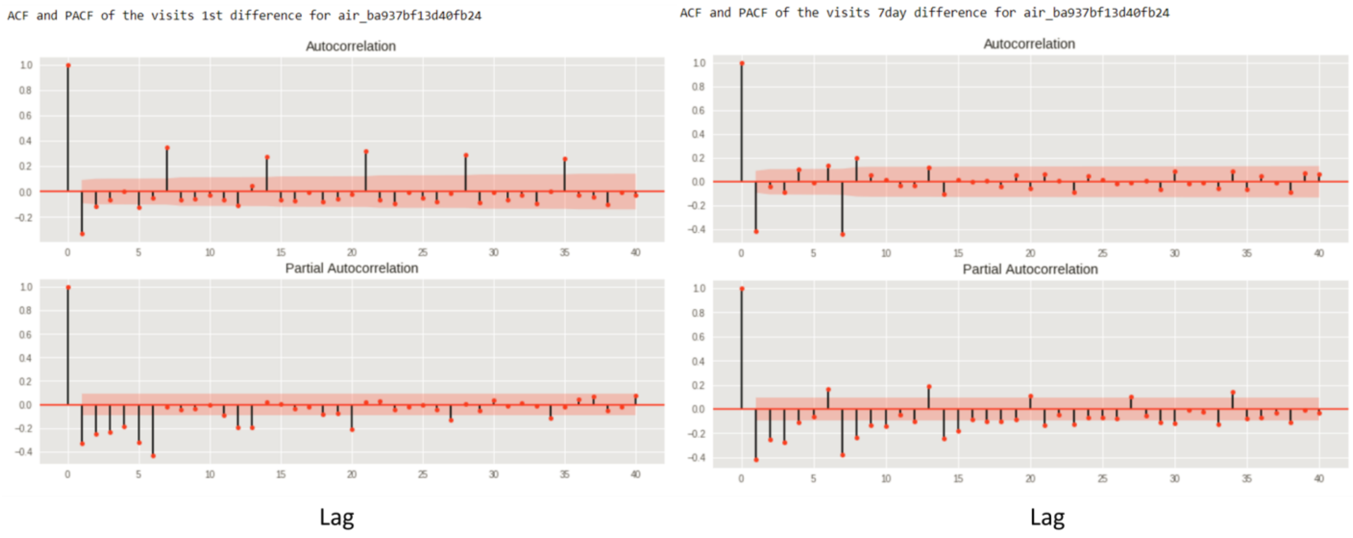


Fig. 4-9: ACF and PACF plots for 1st differenced TS (left) and seasonal-7-day differenced TS (right)

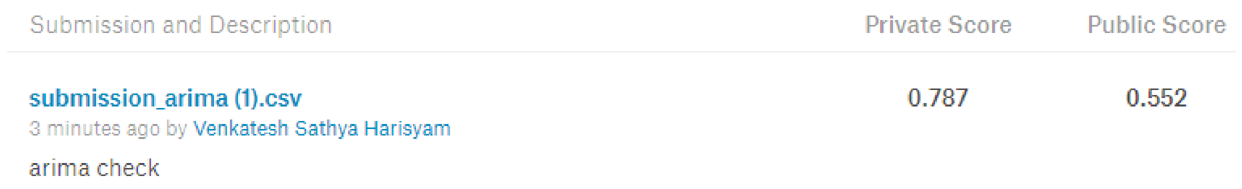
SARIMAX model has the following parameters: (p, d, q)(P, D, Q, m) where the seasonal terms are indicated in uppercase letters and the 'm' is the number of periods in the season.

After making the time series stationary, I plotted the ACF and PACF plots for the 1st differenced series and the seasonal 7-day differenced series. From the above ACF and PACF **plots I made the following conclusions using the 7-day differenced plot (right)** and **I used this reference [15]:**

- From the ACF plot after lag=2 the values are small and enters the 95% confidence interval and at lag=1 the values are negative which means the series slightly over-differentiated and the number of MA terms will be the lag =2 because the values go to 95% intervals => **q=2**
- From the ACF plot the seasonal difference has a negative lag at lag=7 => **Q=1**
- From the PACF plot at lag=2 the values go beyond 95% confidence intervals => **p=2** or could be p=7 but to make it simple I just go with p=2
- Choosing **D=1** instead of d=1 => using only the seasonal difference

The parameters chosen for the SARIMAX model are – **(2,0,2)(0,1,1,7)**. As this is a naïve benchmark model I will be not tuning the parameters and I will apply these parameters to all the restaurants. I have also used exogenous variables by extracting the 'day_of_week' from the 'visit_date' and the holiday_flag (refer notebook-2). I used a loop to basically run the SARIMAX model for the 829 restaurants and to perform the forecast for the 39 days test period. I have also observed that the SARIMAX model with exogenous variable is forecasting better than the model without the exogenous variables.

The inputs for SARIMAX model are the 'visit_date', along with the exogenous variables: 'day_of_week', 'holiday_flag' (these variables show weekly pattern) and the target variable is 'visitors'. The result of my SARIMAX model is:



Submission and Description	Private Score	Public Score
submission_arima (1).csv 3 minutes ago by Venkatesh Sathya Harisyam arima check	0.787	0.552

The low score from the SARIMAX model could be because the parameters are different for different restaurants. The interpolation of the time series might have introduced noise in the data. The data for few restaurants is not available before July, hence they are overfitting because of less amount of data.

5. Methodology

5.1 Data Preprocessing

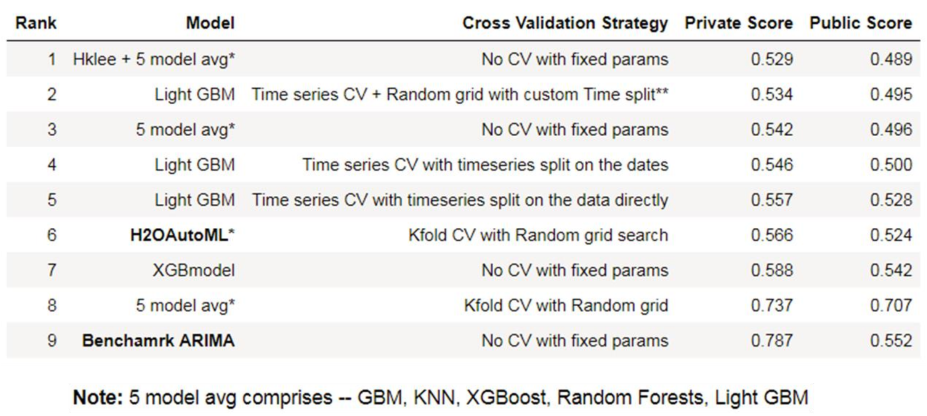
Initially, I had different questions in model building and to perform feature engineering. From the ARIMA-notebook, I was able to understand that the restaurants do not have enough data. Hence, the time series problem must be framed as a Regression problem with time series features. Therefore, to get more insights into the problem, I used some of the public kernels available at the competition webpage to help in some basic feature engineering. I have listed all the necessary references for the public kernels in the notebook-3. The main important questions which I had were:

- How to construct the Regression problem with time series features?
- Cross validation Strategy?

In notebook-3, I have extracted different temporal features from the visit datetime to basically give time series effect. I explored a bit on the spatial information of the restaurants, and I also found out that there are too many genres for the restaurants. Hence, I had combined the genres into 5 different categories. **Initial feature engineering in notebook-3 is as follows:**

- From the air_reserve_data extracting → difference between visit_date and reservation date → this will help in understanding the effect of short term and long term air-reservations
- From the hpg_reserve_data extracting → difference between visit_date and reservation date → this will help in understanding the effect of short term and long term air-reservations
- From the air_visit_data extracting →month, day_of_year, day_in_month, is_month_end, etc. → helps in giving the regression problem a sequence effect
- Calculating the visitor statistics like min_visitors, max_visitors, mean_visitors, visitors_count based on each restaurant wrt. day of the week
- Combining genres into 5 groups →1) Bar or Club 2) European 3) Japanese 4) Other 5) Asian (excluding Japanese)
- Extracting the City, district and local area from the 'air_area_name' for e.g: 'Tōkyō-to Chiyoda-ku Kudanminami' is extracted as → City = Tōkyō-to, district = Chiyoda-ku and local_area = Kudanminami → this will help in giving more spatial information about the restaurants.

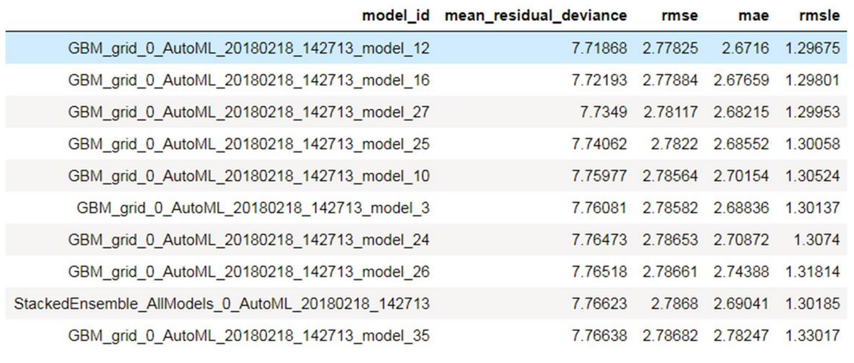
After these basic feature engineering steps, I have used time series cross-validation + random grid Search with custom splitting rather than the direct Timeseries split CV from sklearn and also used K-Fold cross validation for comparison. The k-fold CV seems to yield very bad results when compared to the time-series CV, this is because the k-fold CV was not able to capture the seasonal effect. The regression problems seem to have better results than the individual time series problem. I created a preliminary Leaderboard depicted in fig. 5-1 below.



Rank	Model	Cross Validation Strategy	Private Score	Public Score
1	Hklee + 5 model avg*	No CV with fixed params	0.529	0.489
2	Light GBM	Time series CV + Random grid with custom Time split**	0.534	0.495
3	5 model avg*	No CV with fixed params	0.542	0.496
4	Light GBM	Time series CV with timeseries split on the dates	0.546	0.500
5	Light GBM	Time series CV with timeseries split on the data directly	0.557	0.528
6	**H2OAutoML***	Kfold CV with Random grid search	0.566	0.524
7	XGBmodel	No CV with fixed params	0.588	0.542
8	5 model avg*	Kfold CV with Random grid	0.737	0.707
9	**Benchamrk ARIMA**	No CV with fixed params	0.787	0.552

Note: 5 model avg comprises -- GBM, KNN, XGBoost, Random Forests, Light GBM

H2O AutoML Leaderboard – GBM: rank -1 and Stacked Ensemble: rank-9



model_id	mean_residual_deviance	rmse	mae	rmsle
GBM_grid_0_AutoML_20180218_142713_model_12	7.71868	2.77825	2.6716	1.29675
GBM_grid_0_AutoML_20180218_142713_model_16	7.72193	2.77884	2.67659	1.29801
GBM_grid_0_AutoML_20180218_142713_model_27	7.7349	2.78117	2.68215	1.29953
GBM_grid_0_AutoML_20180218_142713_model_25	7.74062	2.7822	2.68552	1.30058
GBM_grid_0_AutoML_20180218_142713_model_10	7.75977	2.78564	2.70154	1.30524
GBM_grid_0_AutoML_20180218_142713_model_3	7.76081	2.78582	2.68836	1.30137
GBM_grid_0_AutoML_20180218_142713_model_24	7.76473	2.78653	2.70872	1.3074
GBM_grid_0_AutoML_20180218_142713_model_26	7.76518	2.78661	2.74388	1.31814
StackedEnsemble_AllModels_0_AutoML_20180218_142713	7.76623	2.7868	2.69041	1.30185
GBM_grid_0_AutoML_20180218_142713_model_35	7.76638	2.78682	2.78247	1.33017

Fig. 5-1: Preliminary Leaderboard with H2O AutoML leaderboard

By doing these basic feature engineering steps mentioned above for the supervised ML algorithms, their forecasting power has become better when compared to the SARIMAX benchmark model. To show the performance of the supervised ML when compared to SARIMAX the restaurant – air_ba937bf13d40fb24 is considered: (forecast (shown in red and green) comparison between SARIMAX and 5 model_avg model – fig. 5-1).

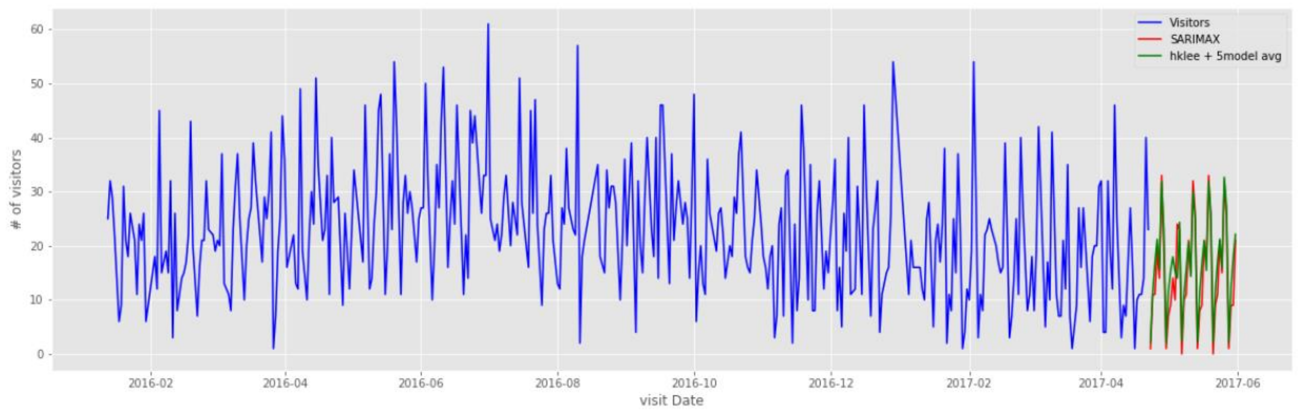


Fig. 5-2: Forecast for air_ba937bf13d40fb24 by SARIMAX model vs. 5 model avg. model

From the notebook-3, I understood that Timeseries CV with custom splitting should be used since it outperformed many other models in the leaderboard. The custom splitting is basically pertaining to the cut dates which must be used for splitting the dataset. There are two important phenomena, which must be carefully observed: 1st - the Golden week in 2016 and the 2nd being the Christmas and New Year's Eve in the end of 2016. Therefore, I must carefully set the cut dates (see below) to observe these above-mentioned effects and I will be using this strategy in the final notebook.

Time series cross-validation strategy:

- **Fold -1:** Training Data: till 2016-04-22 (1 week before golden week), validation data: from 2016-04-22 to 2016-05-12, Hold out: remaining training data
- **Fold -2:** Training Data: till 31st Aug 2016, validation data: 31st Aug to 30th Sept 2016, hold out: remaining training data
- **Fold -3:** Training Data: till 18th Dec 2016, validation data: 18th Dec to 15th Jan 2017, hold out: remaining training data
- **Fold -4:** Training Data: till 23rd Mar 2017, validation data: 23rd Mar to 23rd April 2017, hold out: nothing

I also think that the stacking is not working here, because in the H2O AutoML leaderboard, it can be observed that the stacked ensemble is at rank -9 /10. But this does not exactly prove anything because the cross-validation strategy is K-fold rather than the time series split.

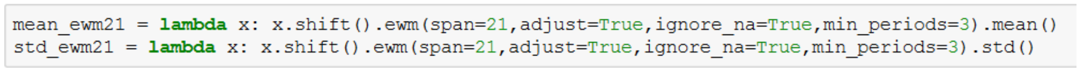
Inference from the RMSLE scores: On a general level the RMSLE values are also not very good and there is a huge difference in the scores between the public LB and private leaderboard. The public leaderboard is scored based on the 16% of the test data and the private LB is scored on the 84% of the test data. From the preliminary leaderboard it can be observed that the models are overfitting the first 16% of the test data. This should be minimized by implementing the time series cross-validation and adding features like rolling averages which will provide time series effect to the regression problem.

5.2 Advanced Feature Engineering, Model Implementation and Stacking

In this section all the inferences and all the feature engineering made in the previous sections will be implemented with some more additional feature engineering. The important features and steps which I took in the final notebook are outlined here:

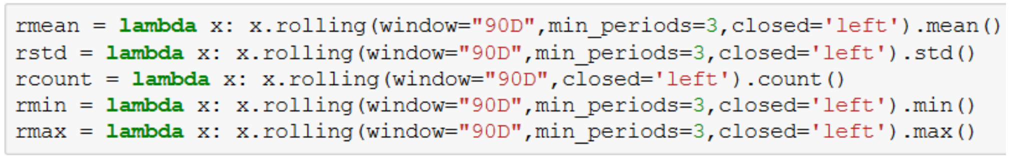
- **Feature related to golden week**: the reason for these feature columns is basically giving special status to the week before and after the golden week as there is surge in visitors to the restaurants
  Feature **- 'before_golden_week'** = 22nd April to 28th April-2016 and 22nd April to 28th April-2017 this feature has value = 1 and has value=0 for other days
  Feature **- 'after_golden_week'** = 9th May to 15th May-2016 and 9th May to 15th May-2017 this feature has value = 1 and has value=0 for other days
- **Combining the genres into 10 different groups** rather than 5 groups
  Grouping genres into: 1) Bar or Club 2) Western 3) Italian/French 4) Japanese
  5) International Cuisine 6) Creative Cuisine 7) Asian (excluding Japanese) 8) Korean food 9) Okonomiyaki/Monja/Teppanyaki 10) Dining and Cafe

- Splitting the areas into 3 different features → accessing the city, district and locality (from section 5.1)
- **Including the weather data** → the reason to add weather data is because, if the weather is bad at a particular location then the visitors for restaurants in that location will be effected. Since, there was seasonal pattern in the visitors to a restaurant adding this information to the dataset will improve the LB scores. Features added are: 'Global average temperature', 'Global average precipitation', 'Global average hours of sunlight'
- **Removing the outliers** → in the EDA notebook I have noticed that few restaurants have very high number of reservations, may be this could be because of some errors in the measurements. These outliers have to be removed. The values more than 2*standard-deviation are removed.
- **Filtering reservations** → removing reservations less than 5 days (Long term reservations more important)
- Adding rolling averages (the most important part)
    - Weighted rolling average of the visitors with lag =7 for seasonality → for the weighted rolling mean, in the **training set** I used a 1-week lagged rolling mean.
      For the test set, I have used the 1-week lagged rolling mean for the first week. For the weeks beyond, I used 2-weeks lagged rolling mean, 3-weeks lagged rolling mean and so on as this lagged rolling mean feature. Just a naive idea, I don't think it is perfect
    - Exponential weighted rolling average (ewm) of the visitors with lag = 21 → here, I used the ewm function in the pandas to get the average:

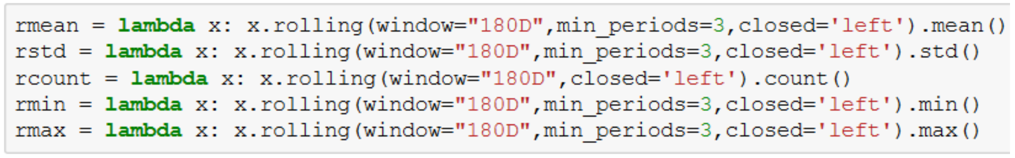


```
mean_ewm21 = lambda x: x.shift().ewm(span=21,adjust=True,ignore_na=True,min_periods=3).mean()
std_ewm21 = lambda x: x.shift().ewm(span=21,adjust=True,ignore_na=True,min_periods=3).std()
```

      I applied the above lambda function to create features: 1) ewm of standard_deviation and mean on the number of visitors when grouped by air_store_id and day_of_week
      2) ewm of standard_deviation and mean on the number of visitors when grouped by air_store_id
      3) ewm of standard_deviation and mean on the number of visitors when grouped by air_store_id and local area
    - Rolling averages of visitors with window width of 90 days and 180 days → this will help in capturing the weather seasonal effects. I have used the pandas rolling function and created the following lambda functions



```
rmean = lambda x: x.rolling(window="90D",min_periods=3,closed='left').mean()
rstd = lambda x: x.rolling(window="90D",min_periods=3,closed='left').std()
rcount = lambda x: x.rolling(window="90D",closed='left').count()
rmin = lambda x: x.rolling(window="90D",min_periods=3,closed='left').min()
rmax = lambda x: x.rolling(window="90D",min_periods=3,closed='left').max()
```



```
rmean = lambda x: x.rolling(window="180D",min_periods=3,closed='left').mean()
rstd = lambda x: x.rolling(window="180D",min_periods=3,closed='left').std()
rcount = lambda x: x.rolling(window="180D",closed='left').count()
rmin = lambda x: x.rolling(window="180D",min_periods=3,closed='left').min()
rmax = lambda x: x.rolling(window="180D",min_periods=3,closed='left').max()
```

      These lambda functions are applied to create features: (total dataset: train + test)
      1) Rolling averages (for 90 and 180 days) for visitors when total dataset is grouped by air_store_id
      2) Rolling averages (for 90 and 180 days) for visitors when total dataset is grouped by air_store_id and day of week

3) Rolling averages (for 90 and 180 days) for visitors when the total datset grouped by air_store_id and air_genre_name

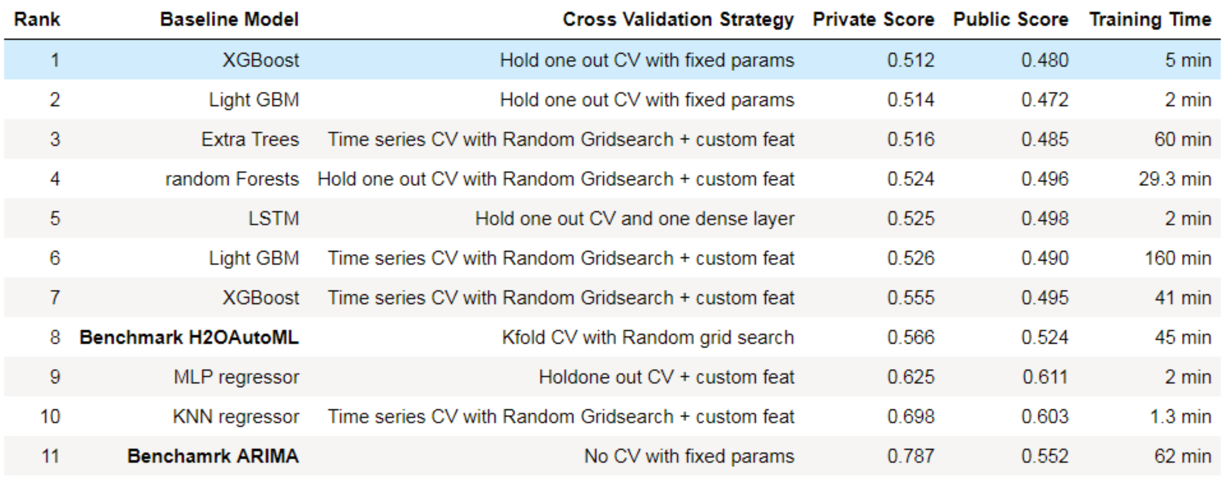
- When using the rolling averages with 90 days sliding window, the values before march-1st should be removed because the values will not be accurate because there is no historic data for the year 2015. If this data is not removed from the training set the models might over-fit because of the incorrect values. The sliding widths are approximately considered. These could be optimized.

By using these features, I had a very rich feature set which can be used with various baseline models mentioned in fig. 4-5. To create diversity of the baseline models I had to drops some of the features, and perform different cross-validation strategy. The models which were mentioned in fig. 4-5 are implemented as shown in the following table:

S.No	Algorithm	Feature Columns		Cross-Validation Technique	
1	Random Forest Regressor	Dropped columns include: • Weather related features • All features related to Rolling Averages for 90 and 180 days		Sequential Hold out CV	
2	K- Nearest Neighbor Regressor	Dropped columns include: • All features related to Exponential Weighted Averages		4-Fold Time Series CV + custom splitting	
3	Extra Tree Regressor	Full feature set is used		4-Fold Time Series CV + custom splitting	
4	LigthGBM	Full feature set is used	Dropped columns include: • Weather related features • All features related to Rolling Averages for 90 and 180 days	Sequential Hold out CV	4-Fold Time Series CV + custom splitting
5	XGBoost	Full feature set is used	Dropped columns include: • All features related to Exponential-Weighted Averages	Sequential Hold out CV	4-Fold Time Series CV + custom splitting
6	LSTM Network	Full feature set is used		Sequential Hold out CV	
7	MLP Network	Full feature set is used		Sequential Hold out CV	

Fig. 5-3: RMSE for the LGB model with Sequential hold out CV

After training all the models with inputs as mentioned in the fig. 5-3, I had obtained a baseline leaderboard as follows:



Rank	Baseline Model	Cross Validation Strategy	Private Score	Public Score	Training Time
1	XGBoost	Hold one out CV with fixed params	0.512	0.480	5 min
2	Light GBM	Hold one out CV with fixed params	0.514	0.472	2 min
3	Extra Trees	Time series CV with Random Gridsearch + custom feat	0.516	0.485	60 min
4	random Forests	Hold one out CV with Random Gridsearch + custom feat	0.524	0.496	29.3 min
5	LSTM	Hold one out CV and one dense layer	0.525	0.498	2 min
6	Light GBM	Time series CV with Random Gridsearch + custom feat	0.526	0.490	160 min
7	XGBoost	Time series CV with Random Gridsearch + custom feat	0.555	0.495	41 min
8	**Benchmark H2OAutoML**	Kfold CV with Random grid search	0.566	0.524	45 min
9	MLP regressor	Holdone out CV + custom feat	0.625	0.611	2 min
10	KNN regressor	Time series CV with Random Gridsearch + custom feat	0.698	0.603	1.3 min
11	**Benchamrk ARIMA**	No CV with fixed params	0.787	0.552	62 min

Fig. 5-4: Baseline Leaderboard

Now once the baseline models were trained and cross validated, I chose the first 6 models along with the 5 model average, present in the baseline-leaderboard (fig. 5-4) and the preliminary-leaderboard (fig. 5-1) respectively, as the base-line models for the stacked ensemble. I had collected the predictions for the training and testing sets into different data frames for stacking with a meta-learner. To choose a meta-learner I had to check the correlation between the base-line models. Hence, the following heat-map is created to visualize the correlations between the base line models.

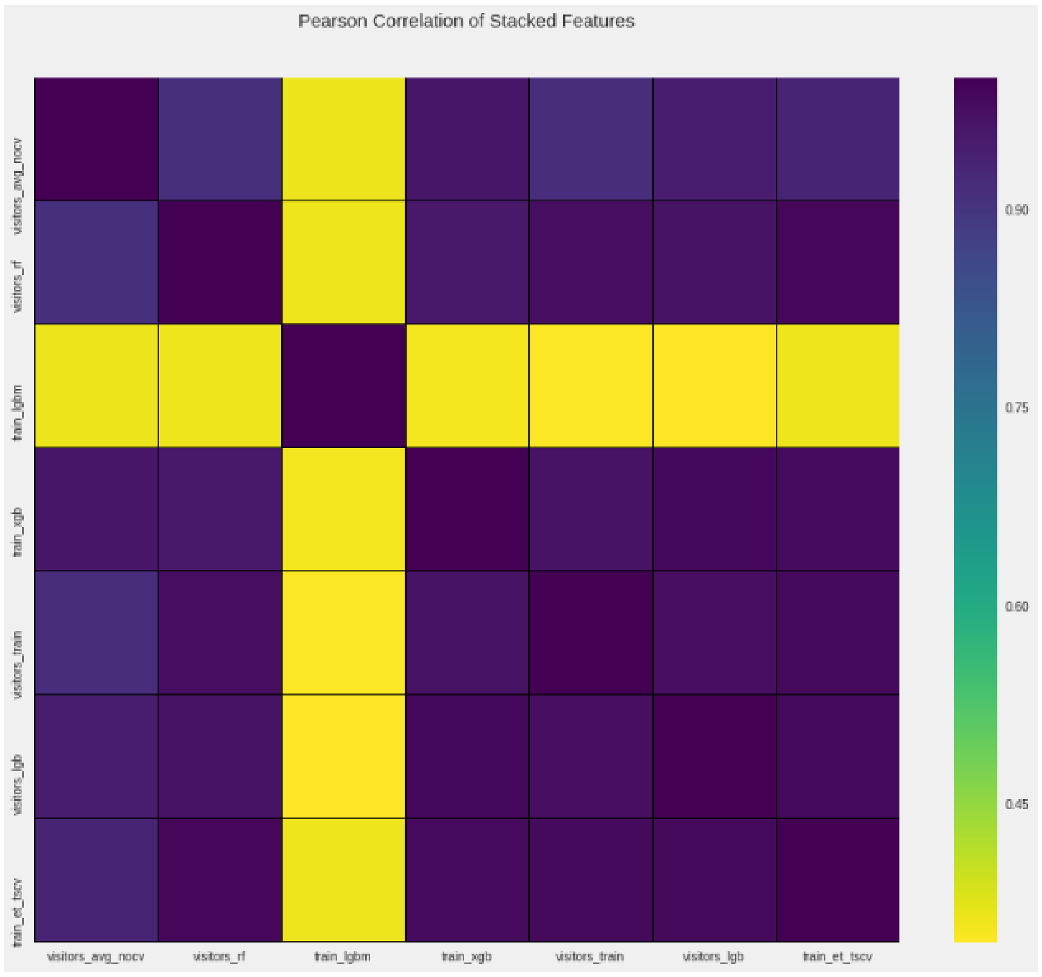
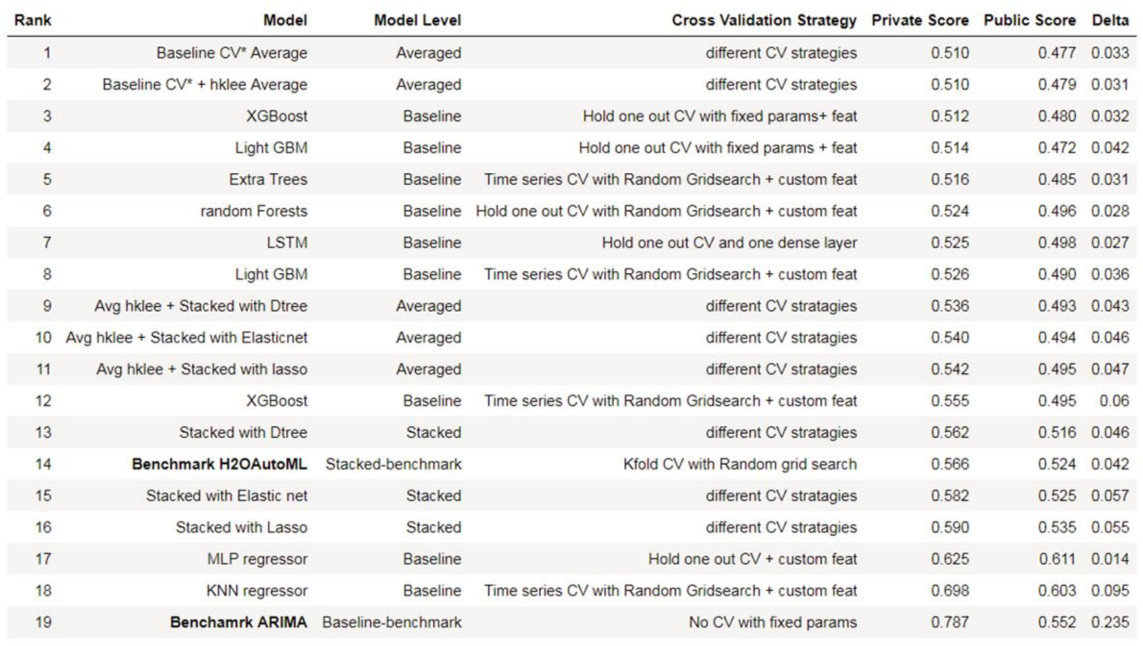


Fig. 5-5: Baseline Heat-map

From the fig. 5-4 it can be understood that the models are correlated except for the 5 model avg and the LGBM model built by sequential hold-out. Although, I have created lot of variation by providing different features for the algorithms and also by performing different cross-validation techniques, the models were still correlated. Since, the baseline models are well correlated, I had to use simple **meta-learners** and I chose: linear-models like **LASSO with CV, Elastic-Net with CV and Decision trees**. I had used all the meta-learners mentioned here with time-series CV (sequential hold-out) and also performed weighted model averaging for the baseline models and the final stacked models because they were giving better Leaderboard scores. All the results are summarized in the final-leaderboard below. The delta in fig. 5-6 shows the difference between the private and the public score, this also gives an idea about how well the cross-validation is performed.



Rank	Model	Model Level	Cross Validation Strategy	Private Score	Public Score	Delta
1	Baseline CV* Average	Averaged	different CV strategies	0.510	0.477	0.033
2	Baseline CV* + hklee Average	Averaged	different CV strategies	0.510	0.479	0.031
3	XGBoost	Baseline	Hold one out CV with fixed params+ feat	0.512	0.480	0.032
4	Light GBM	Baseline	Hold one out CV with fixed params + feat	0.514	0.472	0.042
5	Extra Trees	Baseline	Time series CV with Random Gridsearch + custom feat	0.516	0.485	0.031
6	random Forests	Baseline	Hold one out CV with Random Gridsearch + custom feat	0.524	0.496	0.028
7	LSTM	Baseline	Hold one out CV and one dense layer	0.525	0.498	0.027
8	Light GBM	Baseline	Time series CV with Random Gridsearch + custom feat	0.526	0.490	0.036
9	Avg hklee + Stacked with Dtree	Averaged	different CV stratagies	0.536	0.493	0.043
10	Avg hklee + Stacked with Elasticnet	Averaged	different CV stratagies	0.540	0.494	0.046
11	Avg hklee + Stacked with lasso	Averaged	different CV stratagies	0.542	0.495	0.047
12	XGBoost	Baseline	Time series CV with Random Gridsearch + custom feat	0.555	0.495	0.06
13	Stacked with Dtree	Stacked	different CV stratagies	0.562	0.516	0.046
14	**Benchmark H2OAutoML**	Stacked-benchmark	Kfold CV with Random grid search	0.566	0.524	0.042
15	Stacked with Elastic net	Stacked	different CV stratagies	0.582	0.525	0.057
16	Stacked with Lasso	Stacked	different CV stratagies	0.590	0.535	0.055
17	MLP regressor	Baseline	Hold one out CV + custom feat	0.625	0.611	0.014
18	KNN regressor	Baseline	Time series CV with Random Gridsearch + custom feat	0.698	0.603	0.095
19	**Benchamrk ARIMA**	Baseline-benchmark	No CV with fixed params	0.787	0.552	0.235

Note: Baseline CV consists of the first 6 models in the Baseline leaderboard
Delta := Private Score – Public Score

Fig. 5-6-: Final Leaderboard

Now from fig. 5-6 the results of the baseline models: XGBoost (3rd rank) and LightGBM (4th rank) will be discussed in the next section, because in the average models, which are the first and 2nd in the leaderboard larger weights were given to these single models when compared to the remaining baseline models.

5.3 Results

After following all the feature engineering steps mentioned in section 5.1, I created a Light GBM model with Sequential hold out CV technique (because it gave a better result than Timeseries CV). The data set is split as follows: **Training data** till - 12th March 2017 and **hold out validation set** from 12th March 2017 to 23rd April 2017. The RMSE training and validation curve for the model is shown in figure. 5-6 below.

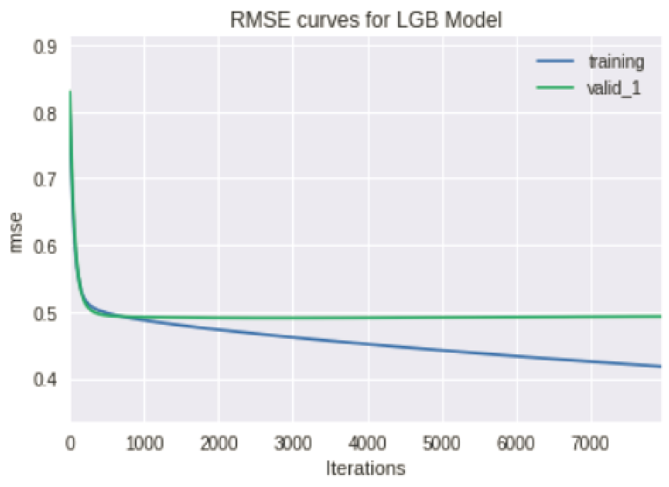


Fig. 5-7: RMSE for the LGB model with Sequential hold out CV

The parameters for the LGB model can be found in fig.4-5. From fig. 5-7, it can be inferred that the LGBM model is converging and I have used 'early_stopping' parameter in the Light GBM training and it had the best iteration at 2976. From the graph we can observe that beyond this iteration the difference between the RMSE scores of the validation and training seems to increase. Hence, the parameters of the LGB model at this iteration are used for forecasting.

To understand whether the LGB model is performing well or not, the forecast for the restaurants should be checked with the benchmark. The output for the air_ba937bf13d40fb24 is shown as the example for the forecast comparison with SARIMAX.

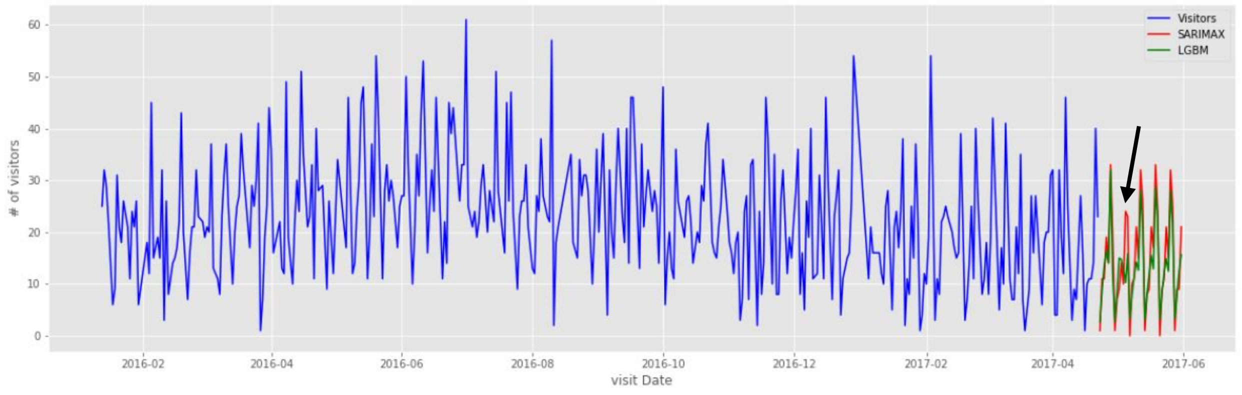


Fig. 5-8: Forecasts for air_ba937bf13d40fb24 by LGBM vs. SARIMAX

From fig. 5-8, it can be observed that the number of visits are controlled on the second peak and on an overall level the values are lower than SARIMAX model, I think this is because of the additional feature engineering and cross-validation in the LGBM model which facilitates for more relations between the inputs and the target variables.

I had also implemented the XGBoost model with the same sequential hold out CV as mentioned above for the LGBM and the RMSE curve is as follows: the parameters for the XGB model in fig.4-5

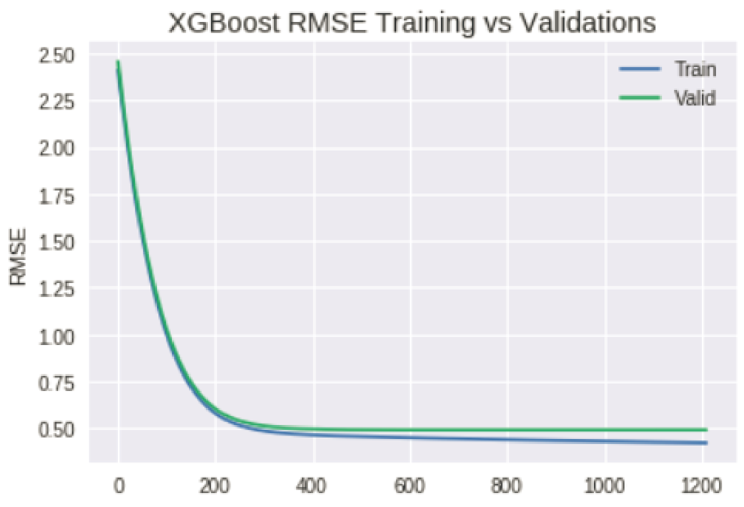


Fig. 5-9: RMSE Curve for XGBoost model with Sequential hold out CV

From fig. 5-9, it can be inferred that the XGBoost model is converging and I have also used 'early_stopping' parameter here for the training and it had the best iteration at 711. From the RMSE graph we can observe that beyond this iteration the difference between the RMSE scores of the validation and training seems to increase. To understand whether the XGB model is performing well or not, the forecast for the restaurants should be checked with the benchmark. The output for the air_ba937bf13d40fb24 is shown as the example for the forecast comparison with SARIMAX.

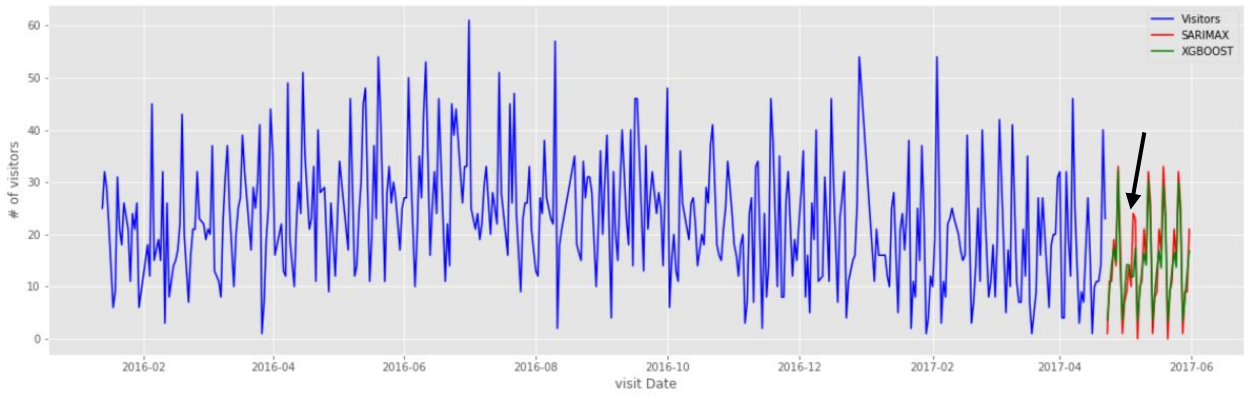


Fig. 5-10: Forecast for restaurant air_ba937bf13d40fb24 by SARIMAX model vs tuned XGBOOST

In fig. 5-10, the forecasts for XGB model are much more controlled over the 2nd peak when compared to the LGB model (hard to visualize- need to zoom in on the black arrowed portion). It can also be observed from the leaderboard scores in fig. 5-6 that XGBoost is better than LGB model. If we check the delta scores for the LGB model (0.042) and the XGB model (0.032), it can be understood that the LGB model is slightly overfitting the first 16% of the data (Public LB test-data).

I think the XGB model or the LGB model with sequential hold-out data set, are working very well for the restaurants with data from January 2016 and whereas due to the shortage of data for other restaurants (fig. 5-11, fig. 5-12 is shown for air_8e4360a64dbd4c50 restaurant where the training data is from July 1st) their RMSLE scores are not decreasing any further because of over-fitting for these restaurants. The LGB and XGBoost models are almost retracing the SARIMAX predictions in the fig. 5-11 and fig. 5-12 for the restaurants with lesser data and this would be the reason for overfitting.

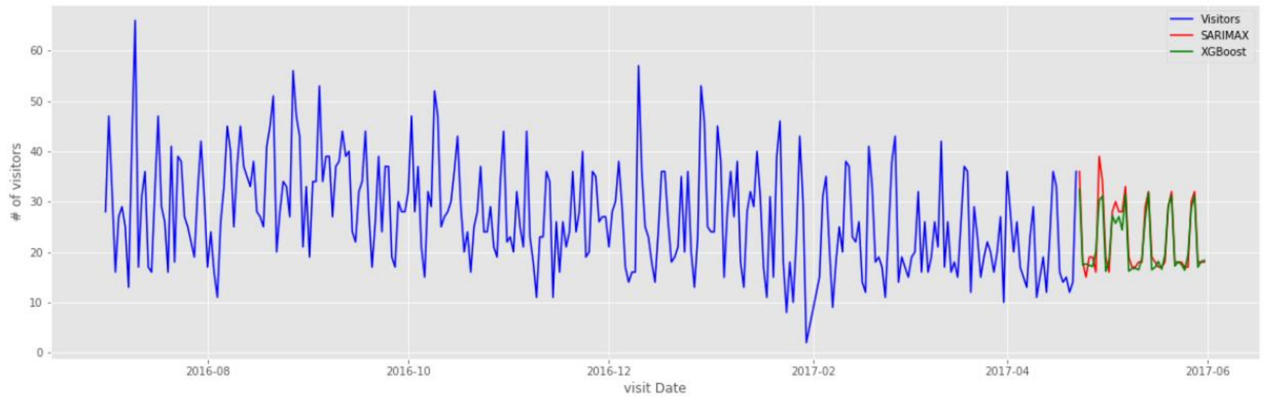


Fig. 5-11: Forecast for restaurant air_8e4360a64dbd4c50 by SARIMAX model vs tuned XGBOOST

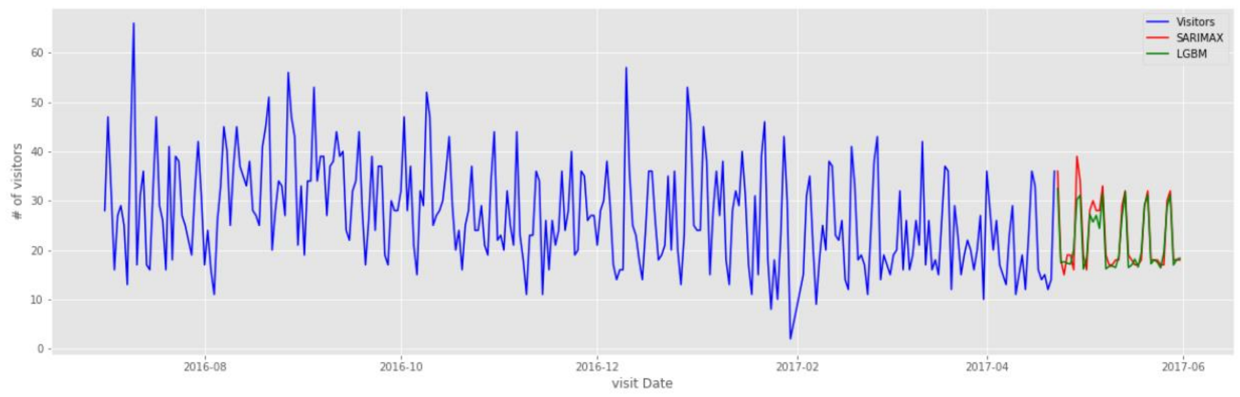


Fig. 5-12: Forecast for restaurant air_8e4360a64dbd4c50 by SARIMAX model vs tuned LGB model

Finally, I would say that with the current feature engineering techniques used here and provided the missing data for the restaurants, both these models would do well in the real-world setting. I could say this because with my current XGB model's RMSLE score, I would be in the top 20 positions in the competition, this shows that the feature engineering and the CV technique used by me are doing good job.

6 Conclusions and Outlook

In this project, the task was to forecast the number of visitors for different restaurants in the AIR-database. The time period for the forecast is 39 days and the data was given from 1st January 2016 to 23rd April 2017. The data given was in different files and all the information had to be merged based on the store-relation id. After performing the merging of various data files, there were lot of missing values because of holidays where the recording system does not record any data and also there was excess information in the HPG database, which covers more restaurants and genres than the AIR-database. The main problem was with respect to the dealing of the missing information for the restaurants and also related to excess information in the HPG reservation data. For the restaurants which were not in the AIR-database, I had removed them completely and while for the remaining missing data I had filled the data samples with an extreme value of -1.

Once I had obtained an initial data-frame with all the information, I did some basic EDA for the Benchmark SARIMAX model for obtaining the model parameters. Then, I had collected a sub data-frame which consists of 'air_store_id', 'visitors', 'holiday_flag' and 'day_of_week' and passed this data-frame into a function which extracted a particular restaurant's information from the sub data-frame and executed the SARIMAX model individually with exogenous variables ('holiday_flag' and 'day_of_week') and forecasted the test-period. This function was executed in a loop for the 829 restaurants and finally all the results were collected for the test_set. From the LB scores, I had observed that the SARIMAX model is overfitting because of insufficient information provided to the model. I was not able to include many exogenous variables because the execution-time of the SARIMAX model takes longer and sometimes it had failed to converge. To avoid convergence problems, I chose lesser number of exogenous variables.

From the benchmark model, I had observed that the restaurants which have lesser visitor information are over-fitting. Hence, the next idea was to pose this time-series problem as a supervised regression problem with time-series features so that the over-fitting problem would be reduced. To get some basic idea on how to perform the conversion to a supervised problem, I had to refer some public kernels in the kaggle website. Using the knowledge gained from the public kernels, I had performed some basic feature-engineering by extracting temporal information from the visit date and also exploring spatial information related to the area of the restaurants. The main problem I faced here was related to the number of genres. There were too many genre categories in the AIR-database, therefore I had grouped them into five categories. After completing the basic feature engineering steps, I had explored on different validation strategies and also constructed different models to check if the supervised problem is better forecasting than the SARIMAX. Finally, I observed that the Time-series CV was performing well compared to the K-fold CV. All the results of the models constructed were displayed in the preliminary leaderboard. I also had an indication that the H2OAutoML stacked ensemble is not performing well. (All the results were reported in notebook-3)

Using the inferences gained from the notebook-3, I had applied some additional feature-engineering techniques to improve the LB scores of the models. I had basically used rolling averages, exponential weighted averages and the rolling weighted averages for the statistics (like mean, min, max and std) based on the number visitors grouped by features like air_store_id and day_of_week. Construction of these features is very difficult because they require lot of historic data. Since, the data at the starting of the datasets will have incorrect values for the rolling averages, I had to remove these rows while training. I had also checked for outliers in the number of visitors and removed them from the dataset before training. After following all these steps, I had observed that the rolling average features had great impact on the LB scores of the models. In fact the single LGB and XGB models with sequential hold-out CV have achieved scores which put my position in the top-20 in the competition leaderboard.

Using the feature engineering techniques (section 5.2), I have constructed various baseline models using different combination of features in the training set. Finally, I created a baseline Leaderboard from which I had picked the first 6 models along with 5-model-avg model from the preliminary leaderboard for the stacked ensemble. After performing various iterations with the stacking of the base-learners using various meta-learners, I had observed that stacking did not work as planned, this was also an indication from the H2OAutoML. I think the stacking could have worked if the base learners are much more diverse and their individual RMSLE scores should also be lower. I also think that 7 models are probably not enough to create a stacking ensemble. There can also be another reason which is related to the data, I think the data is insufficient for few restaurants because they have only data starting from July-2016. The running time is not too large for all the baseline models except for the LGB model, hence it was not an important issue at all. The baseline models and the benchmark models were executed in an hour. The faster execution times could be because of the PC in which I ran all the models. It had 16 cores, hence running time was not an issue.

On contrary to my initial idea of stacking with a meta-learner instead I was able to achieve higher scores with weighted model averaging. I am clearly surprised with the averaging and maybe I should also consider this as one type of stacking method but with constant-linear weighting. Finally, I conclude saying that the single baseline models (LGBM and XGBoost) when implemented with the Sequential Hold-out CV and weighted model averaging are performing better than the stacked ensemble in this problem.

I think the baseline models could still be improved, using the following ideas:

- Implementing Bayesian optimization to tune the hyper-parameters for some base learners to create diversity in the stack [16]
- Implementing GRU cell – another model to create diversity in the stack
- Optimizing the window sizes for the rolling averages so the delta reduces
- More careful optimization for the restaurants with less data

7 References

[1] http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc41.htm

[2] Imdadullah. "Time Series Analysis". Basic Statistics - http://itfeature.com/time-series-analysis-and-forecasting/time-series-analysis-forecasting

[3] Manisha Gahirwal, Vijayalakshmi M. - Inter Time Series Sales Forecasting,

https://arxiv.org/ftp/arxiv/papers/1303/1303.0117.pdf

[4] https://machinelearningmastery.com/time-series-forecasting/

[5] https://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/

[6] Christoph Bergmeira, Rob J Hyndmanb, Bonsoo Koob: A Note on the Validity of Cross-Validation for Evaluating Autoregressive Time Series Prediction

https://robjhyndman.com/papers/cv-wp.pdf

[7] https://github.com/h2oai/h2o-tutorials/tree/master/h2o-world-2017

[8] https://mlwave.com/kaggle-ensembling-guide/

[9] http://home.ubalt.edu/ntsbarsh/Business-stat/stat-data/Forecast.htm#rgintroduction

[10] https://www.kaggle.com/c/recruit-restaurant-visitor-forecasting

[11] https://www.kaggle.com/jeru666/rrv-forecasting

[12] https://stats.stackexchange.com/questions/14099/using-k-fold-cross-validation-for-time-series-model-selection

[13] https://robjhyndman.com/hyndsight/tscv/

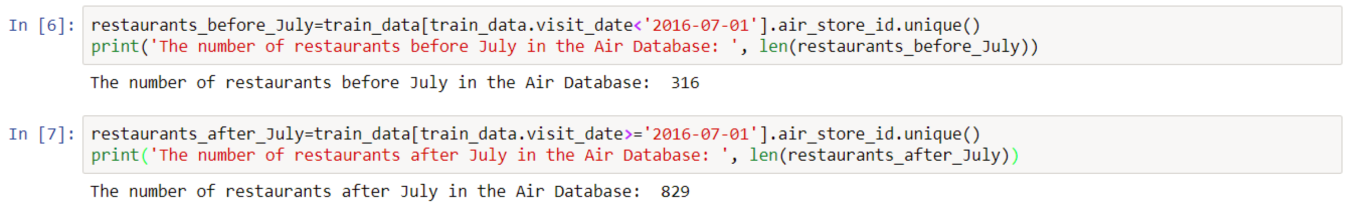
[14] https://www.datasciencecentral.com/profiles/blogs/avoiding-look-ahead-bias-in-time-series-modelling-1

[15] http://people.duke.edu/~rnau/arimrule.htm

[16] https://arimo.com/data-science/2016/bayesian-optimization-hyperparameter-tuning/

[17] https://www.quora.com/What-is-the-difference-between-an-RMSE-and-RMSLE-logarithmic-error-and-does-a-high-RMSE-imply-low-RMSLE

8 Appendix



```
In [6]: restaurants_before_July=train_data[train_data.visit_date<'2016-07-01'].air_store_id.unique()
        print('The number of restaurants before July in the Air Database: ', len(restaurants_before_July))

        The number of restaurants before July in the Air Database:  316

In [7]: restaurants_after_July=train_data[train_data.visit_date>='2016-07-01'].air_store_id.unique()
        print('The number of restaurants after July in the Air Database: ', len(restaurants_after_July))

        The number of restaurants after July in the Air Database:  829
```

Fig. 8-1: Number of restaurants before and after July