Algorithm Selection & Model Evaluation

For dealing with the objective of predicting visitors for Recruit , we had two choices of approaching the problem. One from the angle of a regression problem or the other as a time series prediction. However, upon data exploration, we established the fact that number of restaurants weren't same throughout the data, in fact they jumped from 300 to 800. Thus, since the lag information of 500 odd such restaurants was not available, we reasoned that a timeseries approach would be challenging. Hence, we started with fitting regression models on the dataset.

Model Evaluation

Training vs. Validation vs. Test dataset

One of the major problem that we would want to avoid in any situation while building a model, is to avoid overfitting. If the model is trained on the whole dataset, and is tested on the same dataset, it is expected that the model will not be a generalize meaning it will fare poorly on new unknown scenarios. So for this purpose, we used a validation dataset by separating out our training datasets into 80:20 ratio, 20 being the validation dataset. The overall training dataset is for the period from Jan 2016 to April 2017. The test dataset is the period for which we are going to predict, i.e., from April 23, 2017 to May 31 2017. These are stored in the variables data_train and data_test respectively.

All the models discussed below were fitted on the training dataset and used validation dataset for performance tuning . The final prediction was performed on the test dataset

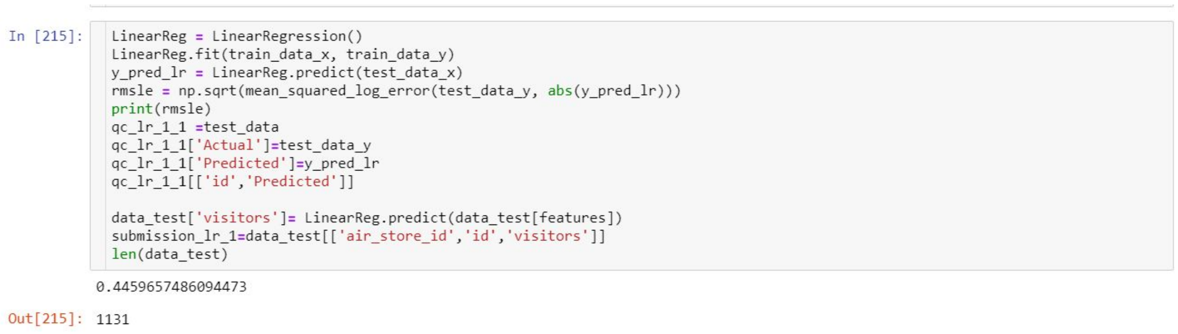
Model Algorithms

Linear Regression

For the start,we approached this problem by selecting a simple algorithm, "Linear Regression". To further address the problem of overfitting, we tried regularization technique by using Lasso and Ridge regression which yielded the below results.

On the validation set, we achieved RMSLE of 0.445 and on the test dataset we obtained a RMSLE of 0.5489.

Below are the code snippets for the same

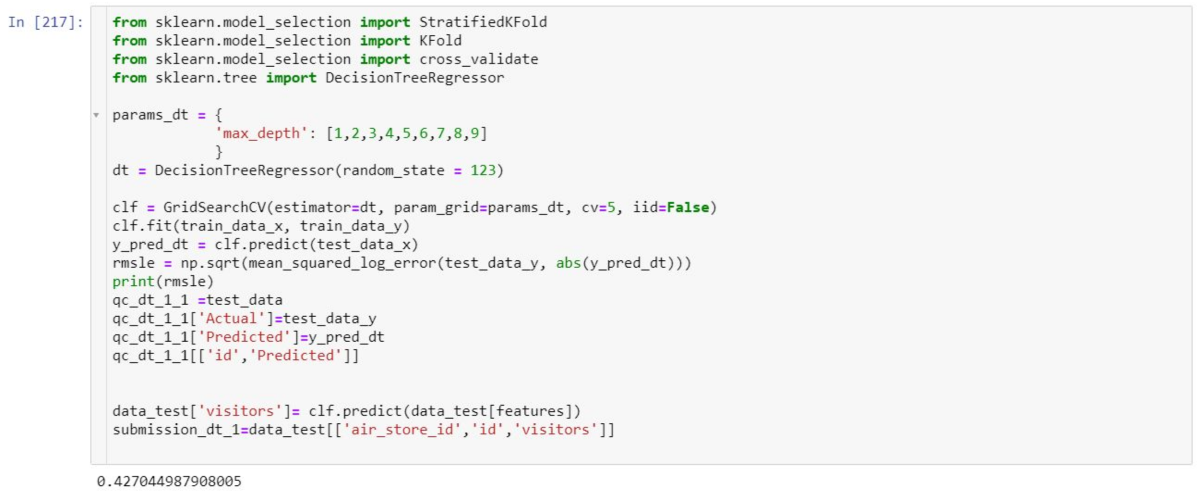


```
In [215]:  LinearReg = LinearRegression()
           LinearReg.fit(train_data_x, train_data_y)
           y_pred_lr = LinearReg.predict(test_data_x)
           rmsle = np.sqrt(mean_squared_log_error(test_data_y, abs(y_pred_lr)))
           print(rmsle)
           qc_lr_1_1 =test_data
           qc_lr_1_1['Actual']=test_data_y
           qc_lr_1_1['Predicted']=y_pred_lr
           qc_lr_1_1[['id','Predicted']]

           data_test['visitors']= LinearReg.predict(data_test[features])
           submission_lr_1=data_test[['air_store_id','id','visitors']]
           len(data_test)

           0.4459657486094473

Out[215]:  1131
```

Decision Tree

In our next iteration, we iterated over decision tree regressor. In addition to fitting this model across the training dataset, we used validation dataset to arrive at the best parameters for the model.

Based on the Grid Search performed, we determined that the best parameters are max_depth at 5



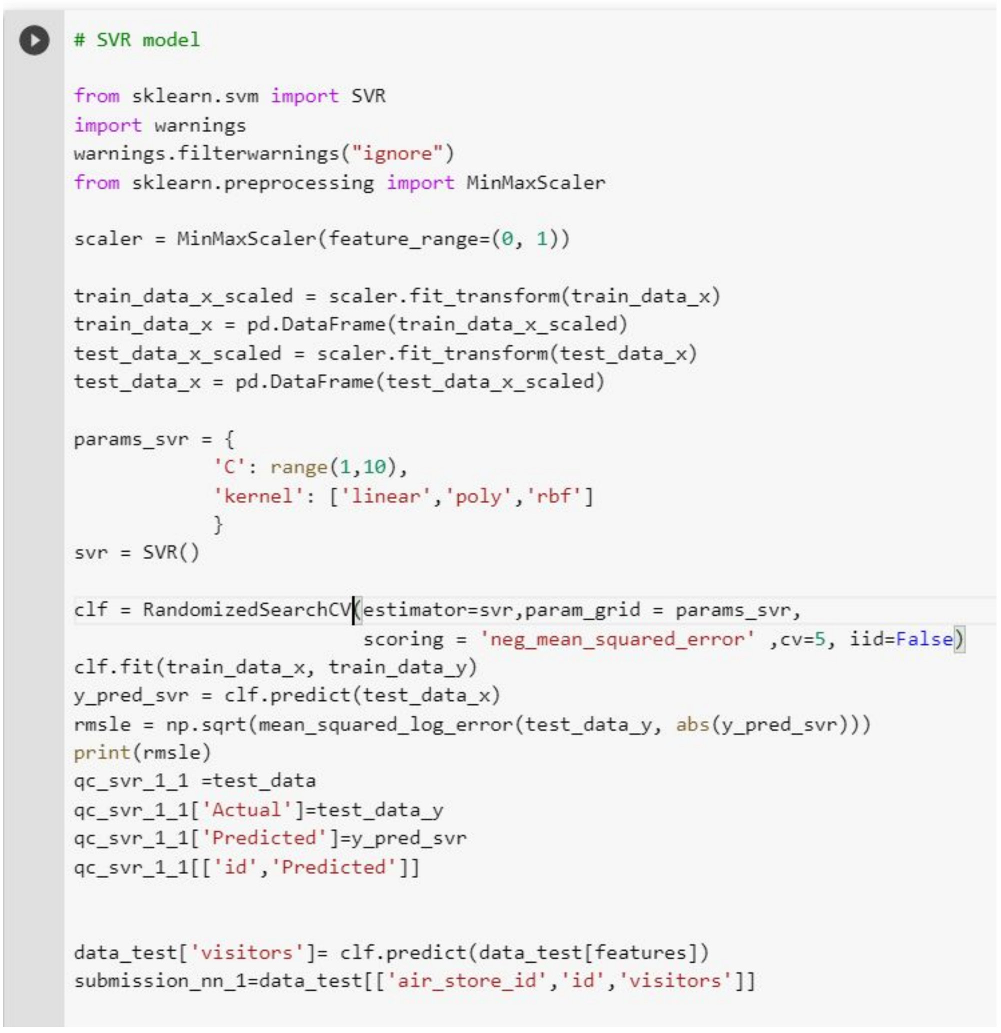
```
In [217]:  from sklearn.model_selection import StratifiedKFold
           from sklearn.model_selection import KFold
           from sklearn.model_selection import cross_validate
           from sklearn.tree import DecisionTreeRegressor

           params_dt = {
                       'max_depth': [1,2,3,4,5,6,7,8,9]
                       }
           dt = DecisionTreeRegressor(random_state = 123)

           clf = GridSearchCV(estimator=dt, param_grid=params_dt, cv=5, iid=False)
           clf.fit(train_data_x, train_data_y)
           y_pred_dt = clf.predict(test_data_x)
           rmsle = np.sqrt(mean_squared_log_error(test_data_y, abs(y_pred_dt)))
           print(rmsle)
           qc_dt_1_1 =test_data
           qc_dt_1_1['Actual']=test_data_y
           qc_dt_1_1['Predicted']=y_pred_dt
           qc_dt_1_1[['id','Predicted']]


           data_test['visitors']= clf.predict(data_test[features])
           submission_dt_1=data_test[['air_store_id','id','visitors']]

           0.427044987908005
```

Decision Tree Regression has performed slightly better than the linear model with RMSLE on validation dataset at 0.427 and on the test dataset, it achieved an RMSLE of 0.5483(marginally better than the linear model)

Support Vector Regression

After trying the basic linear models, the next model in line was Support Vector Regression(SVR). To identify the right parameters for the model we threw a RandomisedSearchCV to find the best combination of attributes. Based on the hyper tuning of parameters, we arrived at the following as the best parameters {To be filled}. Due to it's computational requirement we ran the below code in Google Collab to leverage GPU resource

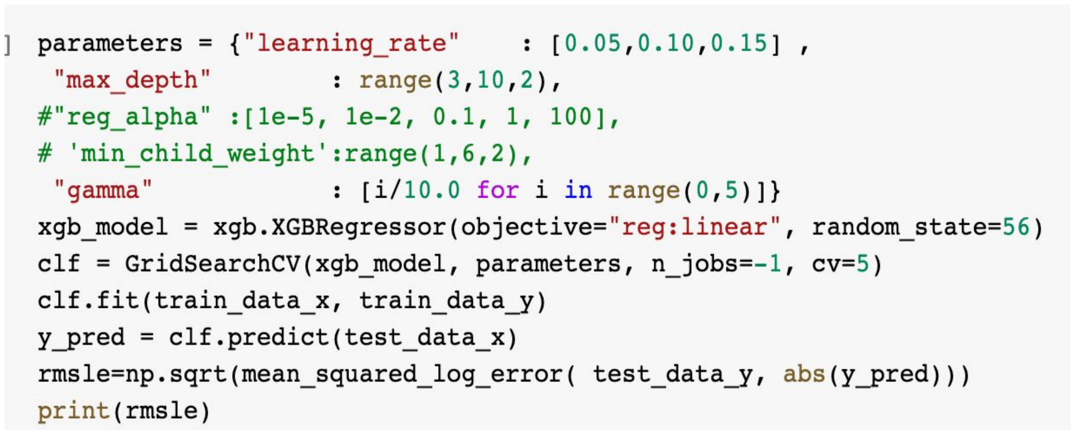
This model yielded even better results in comparison to the other two simple linear models, with an overall RMSLE on the test dataset of 0.5472.



```
# SVR model

from sklearn.svm import SVR
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))

train_data_x_scaled = scaler.fit_transform(train_data_x)
train_data_x = pd.DataFrame(train_data_x_scaled)
test_data_x_scaled = scaler.fit_transform(test_data_x)
test_data_x = pd.DataFrame(test_data_x_scaled)

params_svr = {
            'C': range(1,10),
            'kernel': ['linear','poly','rbf']
            }
svr = SVR()

clf = RandomizedSearchCV(estimator=svr,param_grid = params_svr,
                         scoring = 'neg_mean_squared_error' ,cv=5, iid=False)
clf.fit(train_data_x, train_data_y)
y_pred_svr = clf.predict(test_data_x)
rmsle = np.sqrt(mean_squared_log_error(test_data_y, abs(y_pred_svr)))
print(rmsle)
qc_svr_1_1 =test_data
qc_svr_1_1['Actual']=test_data_y
qc_svr_1_1['Predicted']=y_pred_svr
qc_svr_1_1[['id','Predicted']]


data_test['visitors']= clf.predict(data_test[features])
submission_nn_1=data_test[['air_store_id','id','visitors']]
```

XGBoost

Post trying our linear models, we finally iterated the dataset with an ensemble model namely XGBoost Regression. Fitting this model on the training dataset, we used GridSearchCV to fine tune the hyperparameters for this model. The best values for the hyper parameters Learning Rate, Max Depth, Gamma are 0.1, 7, 0.3 respectively.



```
] parameters = {"learning_rate"    : [0.05,0.10,0.15] ,
   "max_depth"        : range(3,10,2),
  #"reg_alpha" :[1e-5, 1e-2, 0.1, 1, 100],
  # 'min_child_weight':range(1,6,2),
   "gamma"            : [i/10.0 for i in range(0,5)]}
  xgb_model = xgb.XGBRegressor(objective="reg:linear", random_state=56)
  clf = GridSearchCV(xgb_model, parameters, n_jobs=-1, cv=5)
  clf.fit(train_data_x, train_data_y)
  y_pred = clf.predict(test_data_x)
  rmsle=np.sqrt(mean_squared_log_error( test_data_y, abs(y_pred)))
  print(rmsle)
```

XGBoost model on the above mentioned hyperparameters yielded on average an RMSLE of 0.5343 on the validation dataset and the predictions on the test dataset resulted in 0.5280 RMSLE.

Business Insights

The final solution of predicting the traffic at the day level at each restaurants will provide a close estimate for Recruit Restaurants at a granular level, helping them to better anticipate the demands. Major advantage for Recruit Restaurants results in mainly in terms of optimization of resources and inventory management. One such area is decision related to staffing decisions. Based on the predicted visitors, they can classify based on peak / non-peak days which in turn helps them to avoid under / over staffing and help improve customer experience. Furthermore, demand forecasting can help the restaurant make decisions on store redesigning with an ultimate objective of achieving higher customer satisfaction as now the stores can anticipate peaks and for instance tackle queues efficiently.