Spanish High Speed Train Service Train Ticket Price Prediction

Problem Definition

The Spanish High Speed Train Service (Renfe AVE) provides a convenient way to travel between major cities in Spain. The ticket pricing changes based on demand and time, and there can be significant difference in prices[1]. The objective of the project is to predict the ticket price given the attributes such as date, time, train type, train class, etc.

Datasets and Inputs

The dataset is made available at Kaggle[2]. The datasets include high speed train ticket information from 4/12/2019 through 7/8/2019. The following columns are included in the data set:

- insert_date: date and time when the price was collected and written in the database, scrapping time (UTC)
- origin: origin city (one of the following 5 major Spanish cities: MADRID, SEVILLA, PONFERRADA, BARCELONA, and VALENCIA)
- destination: destination city (see above)
- start_date: train departure time (European Central Time)
- end_date: train arrival time (European Central Time)
- train_type: train service name
- price: price (euros)
- train_class: ticket class, tourist, business, etc.
- fare: ticket fare, round trip, etc.

There are 2.58 million records in the dataset and the size of the csv file is 27.1 MB. The target variable is 'Price' and the rest are predictor variables.

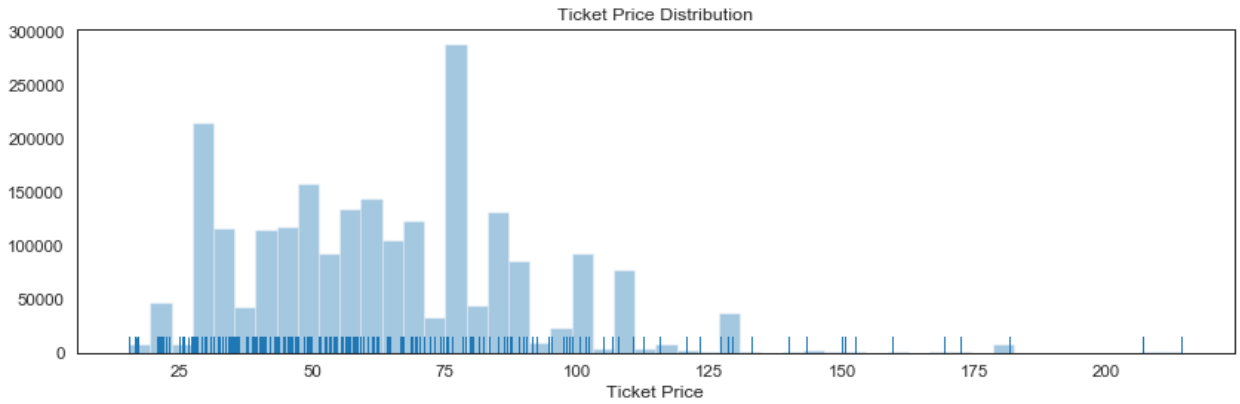
---

[1] A Beginner's Guide to Train Travel in Spain: https://www.seat61.com/Spain-trains.htm

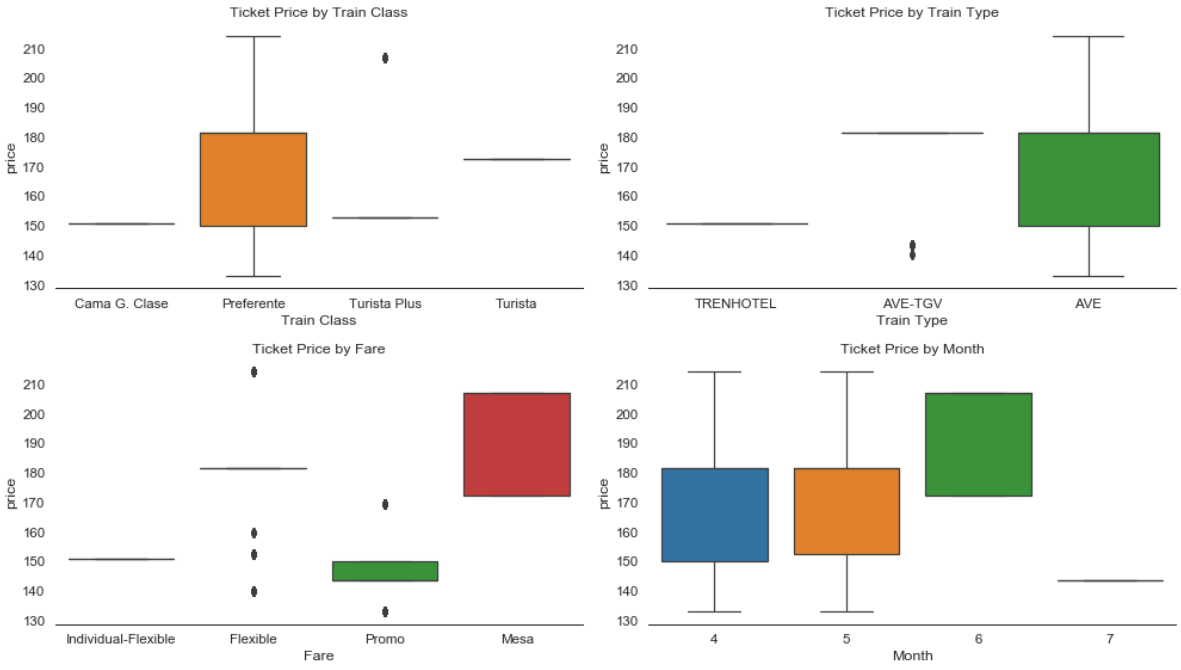
[2] https://www.kaggle.com/thegurus/spanish-high-speed-rail-system-ticket-pricing

Exploratory Analysis

After loading the data into Jupyter Notebook, we'll talk a look at the ticket price distribution.



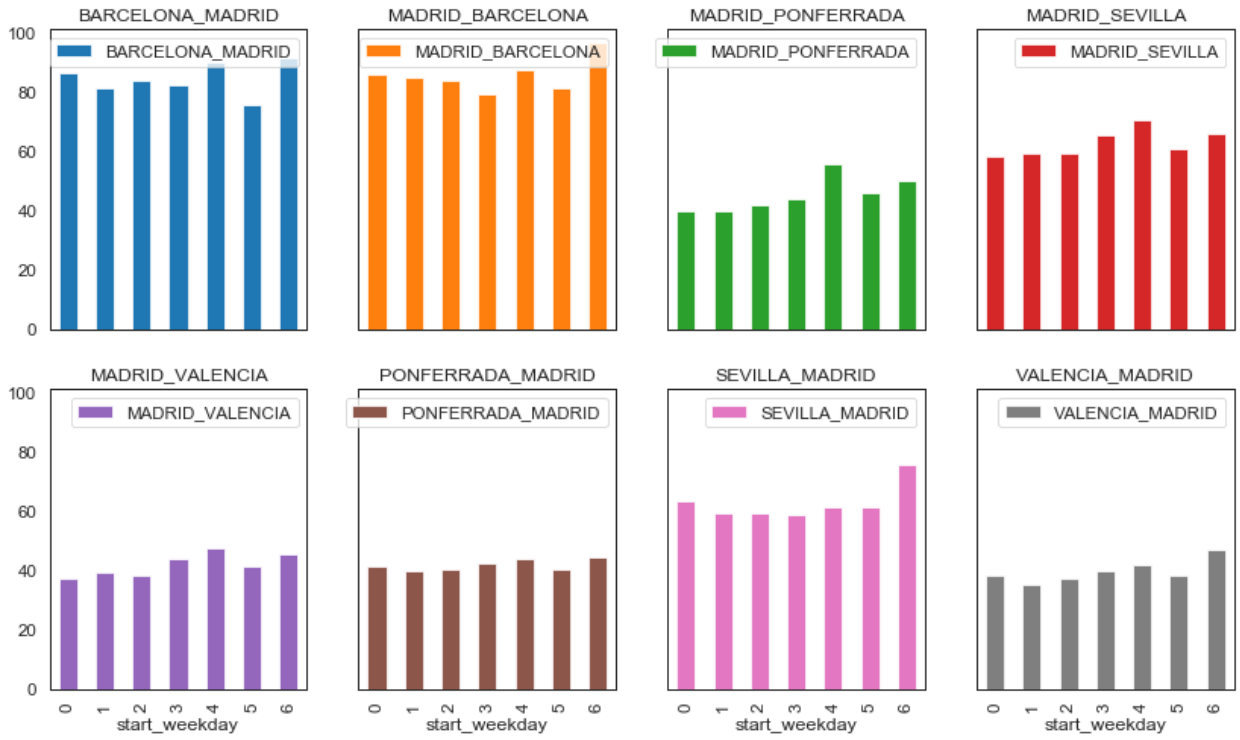
It looks like even the average ticket price is $63.58, the price can go as high as over $200. I was wondering if train type, class, month have something to do with the high price. So I selected only prices over 130 and created some box plots:



It looks like the high price tickets are mostly "Preferente" (First Class), AVE (the fastest of the three train types), and Mesa (seats by a table sold in pairs). Also, most of the high price tickets were issued in June while very few in July.

Note: The high prices are not outliers. They are valid data points and will be used for modeling.

Now, we'll look at all the records in the dataset. Here are the price distributions by Origin-Destination. The ticket prices for travel between Madrid and Barcelona are much higher than the rest even though the travel time is not much longer:



By comparison, here are the travel distances from Google (Note the distance between Madrid and Barcelona is longer than the other routes):

Madrid - Serville: 532.7 km

Madrid - Barcelona: 624.7 km

Madrid - Ponferrada: 389.7 km

Madrid - Valencia: 355 km

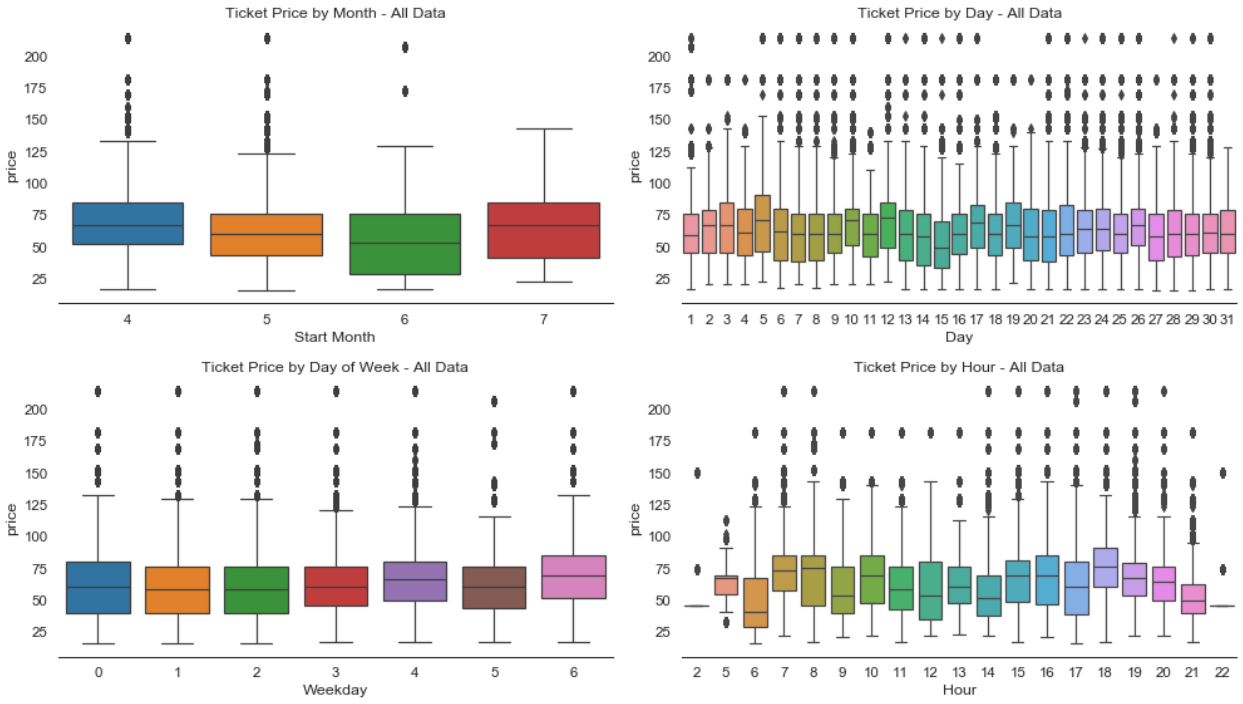
Madrid and Barcelona are major tourism hotspots and the demand is high. Also, there are more high-speed trains between them (high-speed more likely to demand high price).

The higher prices for tickets between Madrid and Barcelona might be due to a combination of distance, train type, high demand, and high travel class. The travel time (see below) shows that even the distance between Madrid and Barcelona is long, the travel time is short.

The average travel time (by train):

Origin - Destination	Hours
BARCELONA_MADRID	2.66
MADRID_BARCELONA	2.56
MADRID_PONFERRADA	4.33
MADRID_SEVILLA	2.16
MADRID_VALENCIA	2.52
PONFERRADA_MADRID	4.59
SEVILLA_MADRID	2.28
VALENCIA_MADRID	2.67

Here are the distribution of ticket prices by Month, Day, Weekday, and Hour for all records.



The box-plots above are squeezed together so it's hard to tell how much the prices vary in different categories. A one-way anova test confirms that all the variables (Month, Day, Weekday, Hour) are significant in predicting the ticket price. For example

```
>> get_f_p('start_month', [4,5,6,7])
F Stat is: 18809.292953094857, P Value is: 0.0.
```

Implementation

Now with a good understanding of the data, we're ready to implement the model in AWS SageMaker. We will perform feature engineering first.

Feature Engineering and Data Preparation

The feature engineering process includes the following steps:

1. Remove the 'row id' and 'insert date' columns as they don't add value to the analysis;
2. Drop rows with missing values. There are 310,681 records with missing 'price' information. Although I could fill those with inferred values (such as average price), I decided to remove those records as I have enough records to build the model.
3. Calculate the 'start_month', 'start_day', 'start_weekday', and 'start_hour' columns from the 'start_date' column. I don't need the 'year' since the entire data is in 2019.
4. The 'start_date' and 'end_date' columns will be dropped since I will use just the components for modeling and 'end_date' is highly correlated with 'start_date'.
5. Use one-hot encoding to encode the categorical variables. They are: 'origin_destination', 'train_type', 'train_class', 'fare', 'start_month', 'start_day', 'start_weekday', 'start_hour'.
6. After the dummy variables are created, the categorical variables will be dropped.
7. The remaining data will be split into 3 partitions: training (4/9), validation (2/9), test (1/3). The data are saved in CSV files

Hyper Parameter Tuning

After much research I created the following process to select the hyper parameters:

1. Build an XGBoost model using SKlearn for hyper parameter tuning. Focus on the following hyper parameters as we have a small number of features:

    Learning Rate (step size used in updating the weights for the features)

Max_depth (no. of layers in each tree)

Min_child_weight (minimum no. of instances in a leaf)

2. Hyper parameter tuning is tricky as the combinations are endless. Based on prior experience, I ran the SageMaker XGBoost model with the following configuration and the model gave a very low RMSE of $5.17

   Learning Rate=0.1

   Max_depth=5

   Min_child_weight=6

3. I decided to use the Grid Search Cross Validation function (GridSearchCV) in SKLearn to find the best parameters from the following options

```
params = {'min_child_weight':[5,7,9],
          'max_depth': [7,8,10],
          'learning_rate': [0.2, 0.9],
          'subsample':[0.5],
          'n_estimators': [500]
         }
```

4. Since the purpose of the search was to find best hyper parameters, not fitting the model, I created a stratified sample (416k rows) from the original dataset (2.27 million rows) for this purpose.

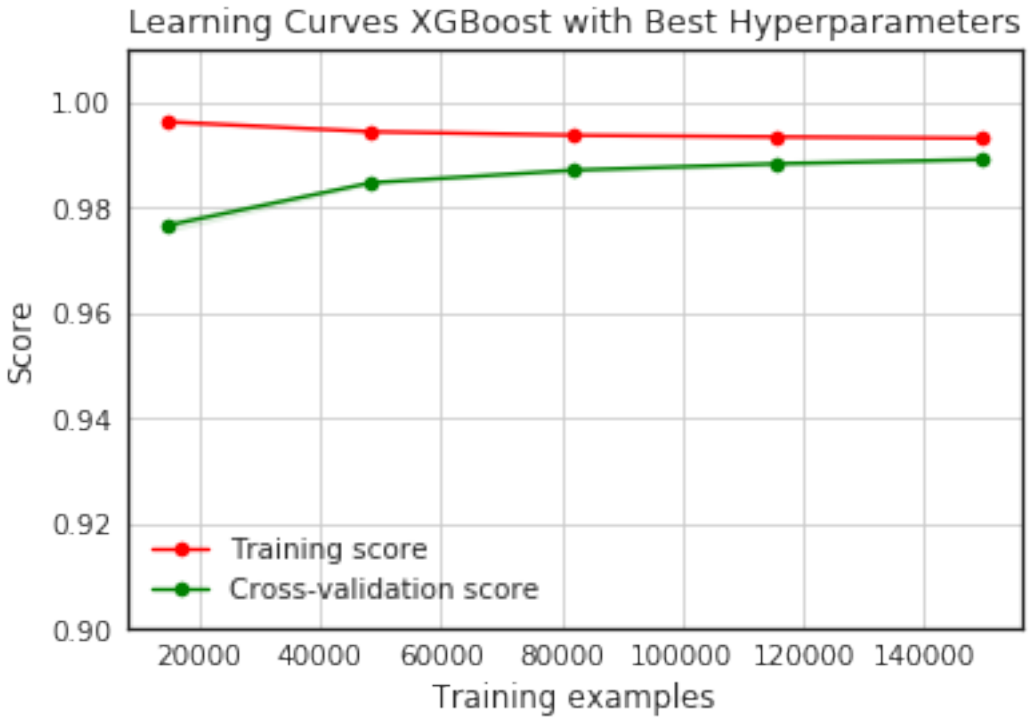
5. It still took 3 hours to finish the search (3x3x2x3=54 models). And here are the best hyper parameters

   Learning Rate=0.2

   Max_depth=10

   Min_child_weight=5

6. A Learning Curve plot is created below to check the training/validation scores as training examples increase. The plot shows that as more training data is used, the validation score keeps going up. After 140,000 the training and cross-validation scores are very close and flatten out. The explained variance ratio is over 98% for both so the accuracy is high.



Modeling

The modeling process is carried out in the following steps:

1. Load the CSV files (Train, Validation, Test data) to AWS S3.
2. Use AWS SageMaker, get an EC2 instance with XGBoost configuration.
3. Build the linear regression model using the XGBoost algorithm. Set parameters using previously decided best choices. Use the Train data to train the model, and the Validation data to reduce overfitting. Set the model to run 1,500 epochs.
4. Once the model is trained, create a SageMaker transformer to make predictions on the test data. The output will be saved on S3.
5. Move the prediction results back to Jupyter Notebook for model evaluation.

Algorithms for Modeling

Two Major algorithms are used for modeling. Here is a brief explanation on how they work.

- XGBoost - XGBoost stands for e**X**treme **G**radient **B**oosting. It has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. It is an implementation of Gradient Boosting, which improves decision tree models by using information from the residuals or errors of prior models. If offers parameter regularization (to reduce overfitting), parallelization of tree construction, and distributed computing for high speed and flexibility.

- Light Gradient Boosting Models (LGBM) is yet another implementation of the Gradient Boosting Decision Tree algorithm, released by Microsoft. Like XGBoost, it uses histogram-based methods to find the best splits fast, and in addition, it uses leaf-wise strategy to grow the tree. It uses one-side sample and feature bundling to reduce training time.

Both methods are used for this project. They achieved similar accuracy, measured by RMSE, and LGBM model is much faster than XGBoost (training time 1:10).

Benchmark Model

The average ticket price in the test dataset is $63.38. Without knowing anything about the data, I would guess a ticket price to be $63.38. This is my base model.

If I plug in the $63.38 to calculate the RMSE (Root Mean Squared Error) for the entire test dataset, I will get the Base RMSE of $25.75. My objective is to reduce the RMSE through modeling and training.

Results Evaluation

As explained before, since the prediction is a continuous numeric value, I will use RMSE (Root Mean Squared Error) to evaluate the model. Here is how RMSE is calculated

$$RMSErrors = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

Where ŷ is the prediction, and y is the true label for the test data.

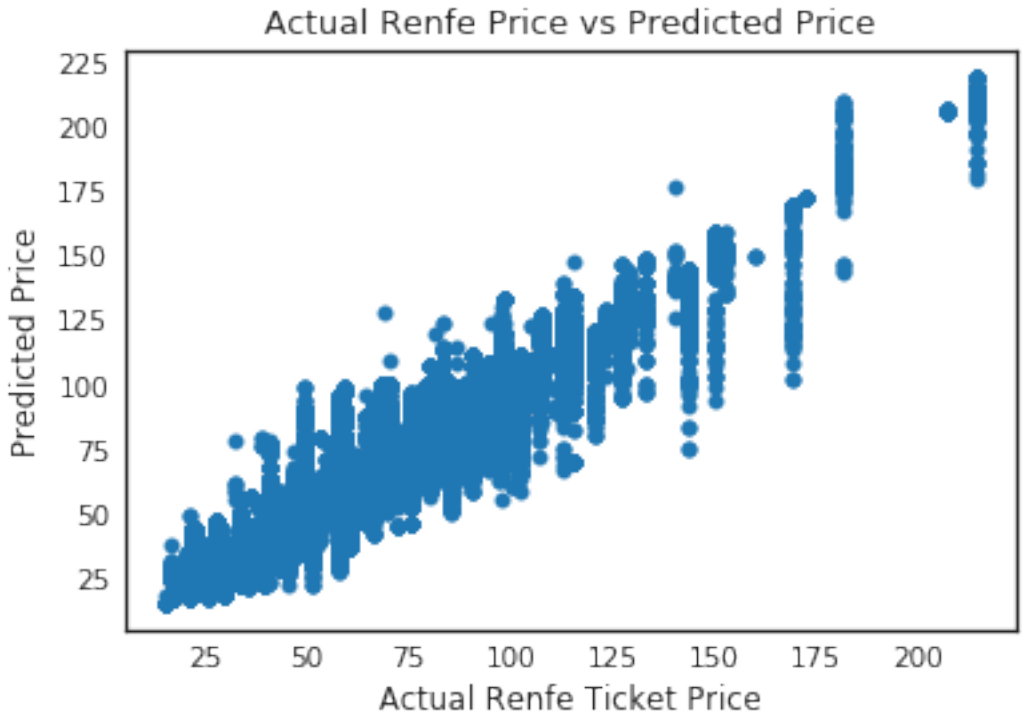
A detailed explanation can be found on Wikipedia[3].

AWS SageMaker XGBoost model generates a respectable RMSE of $4.07, which is less than 10% of the average ticket price of $63.58. However it took long time to train the model. In this case, it took 45 minutes to run 1,500 epochs.

```
rmse = np.sqrt(mean_squared_error(Y_test, Y_pred))
print("RMSE: %f" % (rmse))
RMSE: 4.078219
```

And here is the scatterplot of predicted prices vs. actual prices.

[3] https://en.wikipedia.org/wiki/Root-mean-square_deviation



Comparison with LGBMRegressor

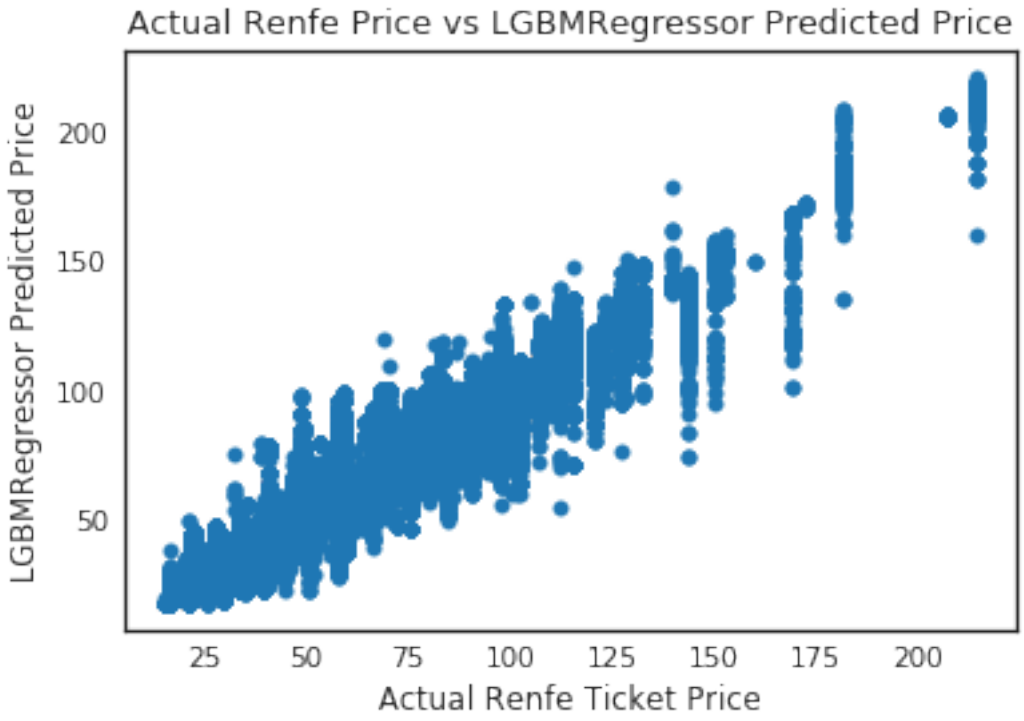
I came across this article[4] about LGBMRegressor. According to the article, its implementation is similar to XGBoost but the training time is much faster. So I decided to give it a try.

I followed the same process in the "Hyper parameter tuning" section to find the best hyper parameters. Here are the results after the grid search

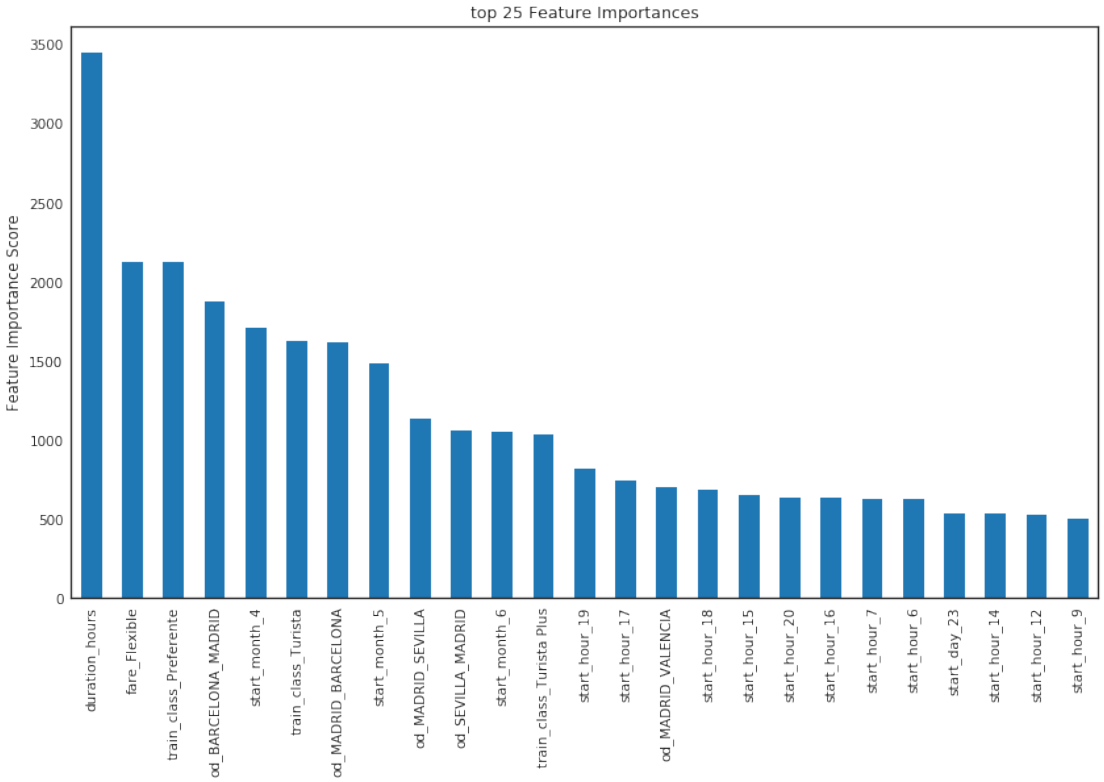
```
LGBM Best Estimator {'learning_rate': 0.9, 'max_depth': 10,
'min_child_weight': 5, 'n_estimators': 500, 'subsample': 0.5}
0.987145
```

The training time for the LGBM model was <5 minutes, compared to a good 45 minutes with the AWS SageMaker. And the RMSE is $4.08, which is very close to the XGBoost model. Here is the scatterplot of predicted prices vs. actual prices.

[4] https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/



Here is the variable importance chart by LGBMRegressor.



The feature importance chart according to the LGBM model above shows that the trip duration is the most important feature relevant to the ticket price, which makes sense. Other interesting points are:

- Flexible tickets command higher prices;
- Tickets between Madrid and Barcelona are generally more expensive;
- First class (Preferente) and Tourist (Turista) class tickets are more expensive;
- Tickets sold in April vary a lot in prices and are likely to be more expensive.