

**Ain Shams University**

**Faculty of Computer and Information Science**

**Scientific Computing department**

**Ain shams university**

**Faculty of computer and information science**

**Bioinformatics department**

**Project Title**

**Airline Ticket Price Prediction**

**PHASE 2**

|  |  |  |
| --- | --- | --- |
| **Name** | **ID** | **Section** |
| **Nour Mohamed Hussein Kamaly** | **20191700701** | **5** |
| **Nourhan Abdel-Karim Khalaf Abdel-Hafez** | **20191700716** | **5** |
| **Ahmed Mohammed Samy Mohammed** | **20191700065** | **1** |
| **Mohammed Nour-Elden Abbas Ismael** | **20191700583** | **4** |
| **Abdul-Rahman Sayed Ali Mohammed** | **20191700339** | **3** |

**Under the supervision of**

**Dr. Dina Khattab**

**Scientific Computing Department,**

**Faculty of computer and Information Science**

**Ain Shams University**

**Analysis phase**

Basic analysis questions that are very useful in the understanding of data:

1. What is the airline that has the max number of ‘very expensive’ ticket category?

Visitara.

Chart

Description automatically generated

1. What is the airline that has the max number of ‘cheap’ ticket category?

Indigo.

Chart, bar chart

Description automatically generated

1. What is the airline that has the max number of ‘expensive’ ticket category?

Air India.

Chart

Description automatically generated

1. What is the airline that has the max number of ‘moderate’ ticket category?

Vistara.

Chart, bar chart

Description automatically generated

1. What is the day that has max number of ‘very expensive’ ticket in each airplane?

Air India : Sunday.

Vistara : Monday.

Table

Description automatically generated

1. What is the day that has max number of ‘cheap’ ticket in each airplane?

Air India: Monday

Air Asia: Tuesday

GO FIRST: Tuesday

Indigo: Tuesday

SpiceJet: Saturday

StarAir: Saturday

Trujet: Friday

Table

Description automatically generatedTable

Description automatically generated

1. What is the day that has max num of ‘moderate’ ticket in each airplane?

Air India: Sunday

Air Asia: Friday

GO FIRST: Thursday

Indigo: Saturday

SpiceJet: Thursday

StarAir: Saturday, thusrsday

Vistara: Wednesday

Table

Description automatically generatedTable

Description automatically generated

1. What is the frequency of each category in each day of airplanes?

Chart, bar chart

Description automatically generated

1. What is the frequency of each category in each month of airplanes?

Chart, bar chart

Description automatically generated

**Preprocessing phase**

1. Preprocessing techniques on features:

* Date :
* Format\_dates() : some dates had ‘/’ , others had ‘-’, the character needed to be unified, to be able to split with it later on, ‘-’ was chosen to replace all backslashes ‘/’. This is implemented by iterating over all the date column and replacing ‘-’ with ‘/’ using .replace() function.
* Extract\_day\_month () : extracting the month and the day of the month by iterating over the formatted date column and splitting by the dash ‘-’, adding the first item of the list to flight day feature, and adding the second item of the list to the flight month feature. Based on the month, the format of the formatted date is changed (could have been done by using pandas datetime but it swapped the month with the day so I did it manually).
* Extract\_weekday () : extracting weekday by converting the formatted date to datetime using pandas then calling function .day\_name() and filling up week day of flight feature with the result.
* Route :
* Split\_route() : the route is originally a dictionary, but it is stored as a string object in the data frame, by using the abstract syntax library’s function literal evaluation, this function returns an object of the datatype it finds in the string, so by converting it to a dictionary, I can access the source and the destination easily and put these values in new columns.
* Stop:
* Split\_num\_of\_stops() :the number of stops can be known from the first few characters, so by slicing the string up to a specific character (0 -> 7 for non-stop, 0 - > 5 for 1-stop, else it’s more than 2 stops ) we can use the integer value of number of stops to represent this feature.
* Find\_where\_is\_the\_stop() : if the number of stops is 1, an extra piece of information may be provided, which is where the stop was, this string would look like 1-stop\n\t\t\t\t\t Mumbai\n\t\t\t\t, so to get the city alone I can split the whole string by the space first, then split the second item of the returned list by the endline, and my desired city would be the first element of the returned list if the length of the list exceeds 1 (if the city information is provided).
* Price:
* Fix\_price\_format(): the price is stored in the data frame as a string object due to the presence of a comma “50,000”, so since this a string, I splitted this string by the said comma, concatenated the results of splitting, and returned the integer value.
* Time taken:
* Calculate\_time\_taken() : the time taken is stored as the number of hours and the number of minutes the flight took in a string object (example : “10h 30m”) so we can split by the space, obtain the hours and the minutes separately, and since we chose to use the hours only, to make use of the minutes, if the number of the minutes exceeded 40 (our chosen threshold ) we add 1 to the hours, there are 2 corner cases : there may not be any minutes (“7h”), and the number of minutes may be added to the hour (“1.03h m”), and both cases are handled in the code.
* Arrival and departure time:
* Categorize\_time(): there is a 100% dependency between arrival time, departure time and time taken, and this dependency can be expressed by: time taken = arrival time - departure time.

Since using the hours themselves in arrival time and departure time will cause redundancy during training, we can extract the time of the day the flight departed and arrived as some times maybe cheaper or more expensive, this is done by categorizing the time intervals to : early morning, morning, afternoon, evening.

1. Preprocessing techniques on the dataset:

* Feature balance on airline feature
* Class imbalance may occur in labels (if it’s a classification problem) or may happen if a certain category appears more than the other categories in a given feature.
* In airline feature, 5 airlines have from (0% -> 7 %) of the data, this would cause the model to overlook these categories, especially when airline Vistara alone is present in 42% of the dataset, to create some sort of feature balance and decrease the number of categories, the 5 airlines with very small percentage are gathered in one new category called “Other\_airline”.

1. Features engineered / extracted

* Flight day / flight month: these were extracted from the date by using pandas datetime as they might have important weights contributing to the label.
* Weekday of flight: representing days of the week, extracted from date by using pandas datetime.
* Distance between the 2 countries: the distance in kilometers between source and destination, it is then normalized to values between 0 ->1 by dividing the distance by the greatest distance found.

1. Features used / discarded

* Hypothesis testing using p value was used in feature selection, our null hypothesis is that the model is learning from the feature, so if the p value exceeded 0.05 then my null hypothesis failed, and the model is not making use of the feature.
* Features used are (p value < 0.05): type, flight month, number of stops, distance between 2 countries, airline, source, destination, Saturday, Sunday, Thursday, Tuesday, departure time, and arrival time.
* Features discarded were: flight day, number of hours taken, one stop in, Friday, Monday, Wednesday.
* The above features were discarded based on hypothesis testing, but these features were discarded as they were in the wrong format (they were fixed and given a new name): date, time taken, stop, route, price.
* Ch code was dropped as there was a 100% dependency between it and the airline as the ch code is a code for the airline, so using both would cause redundancy.
* Num code was dropped as there were so many value counts each had a low frequency between the observations.

1. Sizes of training and testing sets

* As this is a time series data, choosing a random train test would lead to data leakage as we can’t let the model train on new data and test on old or shuffled data so:

The train set consists of the first 80% of the sorted dataset (sorted by date using quicksort)

The test set consists of the last 20% of the sorted dataset (newer dates)

**Modeling Phase**

**- We use 8 models for this project:**

**Text

Description automatically generated**

**General Notes:**

The hyperparameters in the models effect on the accuracy of the model so it should be picked carefully.

**AdaBoost Classifier:**

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

**The results of the model on the dataset:**

Text

Description automatically generated with medium confidence

**GB Classifier:**

GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage n\_classes\_ regression trees are fit on the negative gradient of the loss function, e.g. binary or multiclass log loss. Binary classification is a special case where only a single regression tree is induced.

**The results of the model on the dataset:**

A screenshot of a computer

Description automatically generated with low confidence

**Bagging Classifier:**

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

**The results of the model on the dataset:**

A screenshot of a computer

Description automatically generated with low confidence

**RF Classifier:**

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max\_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

**The results of the model on the dataset:**

A screenshot of a computer

Description automatically generated with low confidence

**XGB Classifier:**

The XGBoost or Extreme Gradient Boosting algorithm is a decision tree based machine learning algorithm which uses a process called boosting to help improve performance. Since it’s introduction, it’s become of one of the most effective machine learning algorithms and regularly produces results that outperform most other algorithms, such as logistic regression, the random forest model and regular decision trees.

**The results of the model on the dataset:**

A screenshot of a computer

Description automatically generated with low confidence

**DT Classifier:**

re a non-parametric supervised learning method used for [classification](https://scikit-learn.org/stable/modules/tree.html#tree-classification) and [regression](https://scikit-learn.org/stable/modules/tree.html#tree-regression). The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation.

**The results of the model on the dataset:**

A screenshot of a computer

Description automatically generated with low confidence

**HGB Classifier:**

This estimator is much faster than GradientBoostingRegressor for big datasets

This estimator has native support for missing values (NaNs). During training, the tree grower learns at each split point whether samples with missing values should go to the left or right child, based on the potential gain. When predicting, samples with missing values are assigned to the left or right child consequently. If no missing values were encountered for a given feature during training, then samples with missing values are mapped to whichever child has the most samples.

**The results of the model on the dataset:**

A screenshot of a computer

Description automatically generated with low confidence

**HGB Classifier:**

This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

**The results of the model on the dataset:**

A screenshot of a computer

Description automatically generated with medium confidence