2021

Flight price prediction report

Submitted by:

SHARUKH ANSARI

Acknowledgement

I'm extremely grateful to SME Khushboo Garg who give me an opportunity to work on this project and provide all the data required and information about the dataset which is crucial for completing this project and guided us in every possible manner. I'd also like to show my gratitude to live Data trained support who help me with this project and guided me in taking care of the skewness.

Objective and Background knowledge

The objective of this project is to predict flight prices given the various parameters. Flight ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, it will be a different story. We might have often heard travellers saying that flight ticket prices are so unpredictable. As data scientists, we are gone prove that given the right data anything can be predicted.

Anyone who has booked a flight ticket knows how unexpectedly the prices vary. Airlines use using sophisticated quasi-academic tactics known as "revenue management" or "yield management". The cheapest available ticket for a given date gets more or less expensive over time. This usually happens as an attempt to maximize revenue based on -

1. Time of purchase patterns (making sure last-minute purchases are expensive)

2. Keeping the flight as full as they want it (raising prices on a flight which is filling up in order to reduce sales and hold back inventory for those expensive last-minute expensive purchases) So, if we could inform the travellers with the optimal time to buy their flight tickets based on the historic data and also show them various trends in the airline industry we could help them save money on their travels. This would be a practical implementation of a data analysis, statistics and machine learning techniques to solve a daily problem faced by travellers.

The objectives of the project can broadly be laid down by the following questions -

1. Flight Trends

   Do airfares change frequently? Do they move in small increments or in large jumps? Do they tend to go up or down over time?

2. Best Time To Buy

   What is the best time to buy so that the consumer can save the most by taking the least risk? So

should a passenger wait to buy his ticket, or should he buy as early as possible?

3. Verifying Myths

   Does price increase as we get near to departure date? Is Indigo cheaper than Jet Airways? Are morning flights expensive?

Review of literature

Predicting flight price is really one hack of job because there is no particular pattern which they follow. A lot of researches have been done on Artificial Intelligence and Machine learning is a part of that. This has helped many researchers focusing on flight price problem to solve using machine learning. Here we are using supervised machine learning to predict the price of the ticket. Some work has been done for determining optimal purchase timing for airline tickets. Our work is especially inspired by [Etzioni et al., 2003].

Motivation

Flight price prediction is really one of the tough tasks because it changes very frequently with day or any particular occasion. So, I have decided to build a model which can be used to predict the price of any airlines at any particular given date or time. So that people can decide which day they can travel with less fare

Data Extraction

1. Automated Script to Collect Historical Data
   For any prediction/classification problem, we need historical data to work with. In this project, flight prices for each route needs to be collected on a daily basis. Manually collecting data daily is not efficient and thus a python script was run on a remote server which collected prices daily at specific time.

2. Cleaning & Preparing Data
   After we have the data, we need to clean & prepare the data according to the model's requirements. In any machine learning problem, this is the step that is the most important and the most time consuming. We used various statistical techniques & logics and implemented them using built-in Python packages.

3. Analysing & Building Models
   Data preparation is followed by analysing the data, uncovering hidden trends and then applying various predictive & classification models on the training set. These included Random Forest, Logistic Regression, Gradient Boosting and combination of these models to increase the accuracy. Further statistical models and trend analyzer model have been built to increase the accuracy of the ML algorithms for this task.

4. Merging Models & Accuracy Calculation
   Having built various models, we have to test the models on our testing set and calculate the savings or loss done on each query put by the user. A statistic of the over Savings, Loss and the mean saving per transaction are the measures used to calculate the Accuracy of the model implemented.

   Here we decided to build a Python script that extracts the required values from a website and stores it as a CSV file. We decided to scrape travel service providers website using selenium.

Such scrapping returns numerous variables for each flight returned and we had to decide the parameters that might be needed for the flight prediction algorithm. Not all are required and thus we selected the following -

- Origin City
- Destination City

- Departure Date
- Departure Time
- Arrival Time
- Total Fare
- Airway Carrier Name
- Duration
- Stops

Mathematical Analysis

In this dataset we have Outliers and skewness issues and to solve that issue we have used Interquartile method and PowerTransformer to take care skewness.

Interquartile method: Interquartile method is used to detect outliers and remove it form dataset using interquartile range.

```
q1=df.quantile(0.25)
q3 = df.quantile(0.75)
iqr = q3 - q1

upper_limit = q3["duration_hours"] + (1.5*iqr["duration_hours"])
index = np.where(df["duration_hours"] > upper_limit)
```

```
df.drop(df.index[index],axis=0,inplace=True)
```

```
print("Data loss is",((14697-df.shape[0])/14697)*100,"Percent")
```

```
Data loss is 0.8301013812342655 Percent
```

Power Transformation method: Power transformation method is used to take care of the skewness present in the dataset and it is one of the best method to treat skewness because of the robustness

```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer()
df[["duration_hours","Arrival_hour","stops"]] = pt.fit_transform(df[["duration_hours","Arrival_hour","stops"]])
```

Data Sources and their formats

For this project we have extracted the data from flight booking site **paytm** from that site we have extracted the selected features which is need to predict the price of a week and stored it in **CSV** format.

Data Preprocessing Done

After scrapping all the data from **paytm** we stored it in local system.

Checking Basic things about data like null values and info

1. We have seen that there is only one columns whose data type is numerical and rest of them is either categorical or datetime.

   So we need to make all the data numerical because machine only understand the numerical data but before start preprocessing we need to check basic info and statistics of the data

```
df.head()
```

	Unnamed: 0	Name	Source	destination	date_of_journey	fare	duration	arrival_time	departure_time	stops
0	0	Go First	BLR-Bengaluru	PNQ-Pune	2021-11-28	5177	1h 20m	19:00	17:40	Non Stop
1	1	SpiceJet	BLR-Bengaluru	PNQ-Pune	2021-11-28	5177	1h 25m	11:10	09:45	Non Stop
2	2	Go First	BLR-Bengaluru	PNQ-Pune	2021-11-28	5177	1h 30m	08:30	07:00	Non Stop
3	3	Go First	BLR-Bengaluru	PNQ-Pune	2021-11-28	5177	1h 30m	18:15	16:45	Non Stop
4	4	IndiGo	BLR-Bengaluru	PNQ-Pune	2021-11-28	5178	1h 35m	10:40	09:05	Non Stop

As we can see in source and destination column that source and destination also containing the code of that city so we need to split it.

```
# Separating code and Airport City Name
df["Source"] = df["Source"].str.split("-",1).str[-1]
df["destination"] = df["destination"].str.split("-",1).str[-1]
```

Checking null values:

```
df.isna().sum()
```

```
Unnamed: 0         0
Name               0
Source             0
destination        0
date_of_journey    0
fare               0
duration           0
arrival_time       0
departure_time     0
stops              0
dtype: int64
```

Basic Statistics and info:

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	14697.0	7348.000000	4242.802788	0.0	3674.0	7348.0	11022.0	14696.0
fare	14697.0	11853.693135	5201.382071	3132.0	7624.0	11448.0	15154.0	51916.0

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14697 entries, 0 to 14696
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Unnamed: 0       14697 non-null  int64 
 1   Name             14697 non-null  object
 2   Source           14697 non-null  object
 3   destination      14697 non-null  object
 4   date_of_journey  14697 non-null  object
 5   fare             14697 non-null  int64 
 6   duration         14697 non-null  object
 7   arrival_time     14697 non-null  object
 8   departure_time   14697 non-null  object
 9   stops            14697 non-null  object
dtypes: int64(2), object(8)
memory usage: 1.1+ MB
```

Separating day, month, year from date:

We need to use feature engineering to separate the day, month, and year from datetime.

```
df["date_of_journey"] = pd.to_datetime(df["date_of_journey"])
df["Day"] =df["date_of_journey"].dt.day
df["Month"] =df["date_of_journey"].dt.month
df["Year"] =df["date_of_journey"].dt.year
```

Splitting Hour and min from Arrival time, Departure time and Duration:

It is very important to know whether this features are important for model and if they are then we need to separate hour and minute and make separate columns for them using feature engineering

```
# separating the hours and minute from Duration and making two new features
duration = list(df["duration"])

for i in range(0,len(duration)):
    if len(duration[i].split()) !=2:
        if "h" in duration[i]:
            duration[i] = duration[i].strip() +" 0m"
        else:
            duration[i] = "0h " +duration[i]

duration_hours = []
duration_min = []

for i in range(len(duration)):
    duration_hours.append(int(duration[i].split(sep = "h")[0]))
    duration_min.append(int(duration[i].split(sep = "m")[0].split()[-1]))



df["duration_hours"] = duration_hours
df["duration_min"] = duration_min
```

```
df["arrival_time"] = pd.to_datetime(df["arrival_time"])
df['Arrival_hour'] = df["arrival_time"].dt.hour
df['Arrival_min'] = df["arrival_time"].dt.minute
```

```
df["departure_time"] = pd.to_datetime(df["departure_time"])
df['departure_hour'] = df["departure_time"].dt.hour
df['departure_min'] = df["departure_time"].dt.minute
```

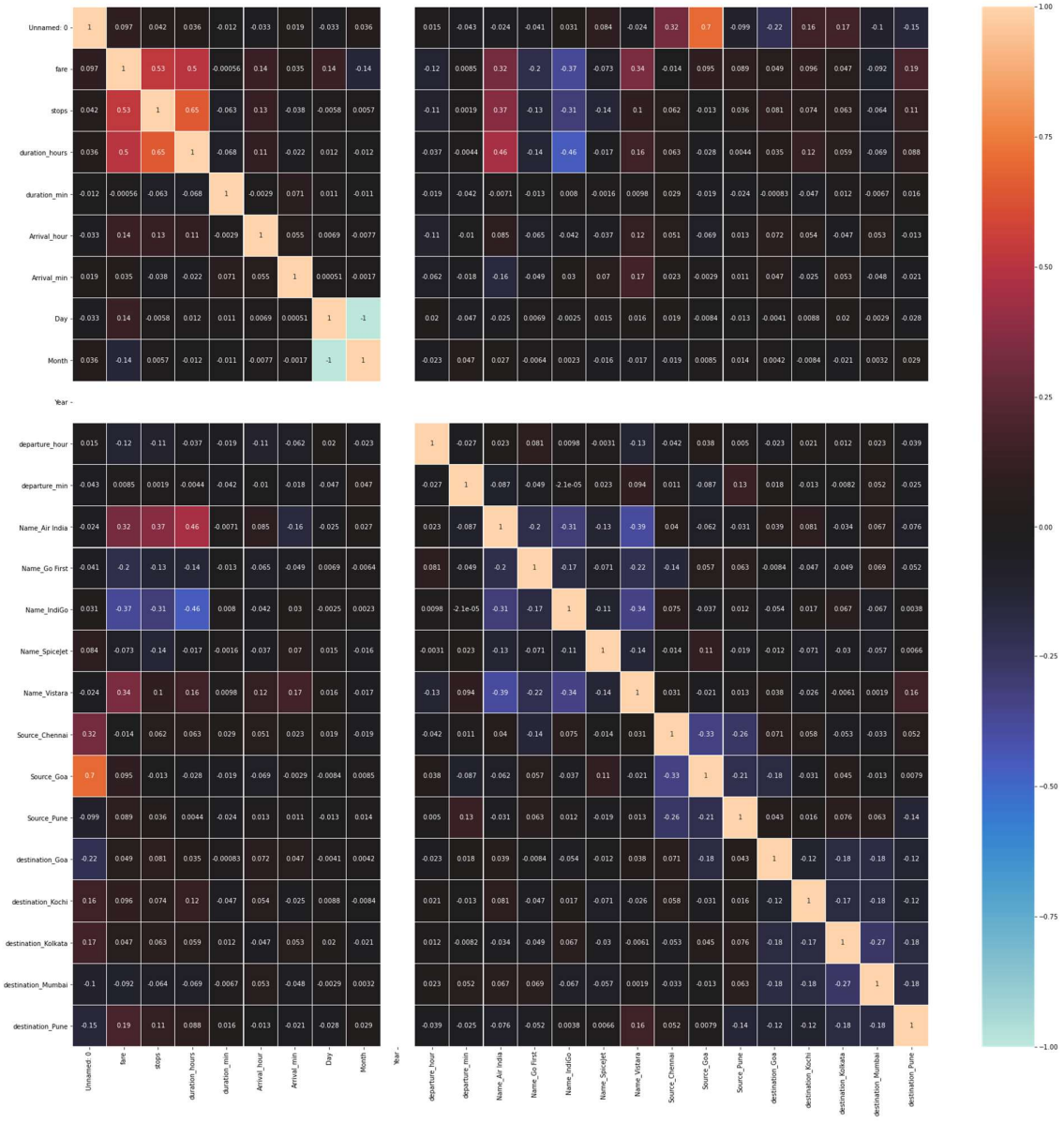
For duration I have built a loop which will split the data into hours and minutes and where minutes is not present in will append zero there and same will be done with hour if hour is missing. And for Arrival time and departure time I have used pandas build in function.

Encoding data using dummies encoder: The get_dummies() function is used to convert categorical variable into dummy/indicator variables.

```
df = pd.get_dummies(df.drop("week_day",axis=1),drop_first=True)

pd.set_option("display.max_columns",None)
```

Multivariate Analysis:



Here we can that Day and month has negative Multicollinearity issues but we know that in month we have only November and December data where as the day change frequently.

There no strong correlation of any column with the target column.

- Data cleaning:

In machine learning we have to assume that data we are getting is normally distributed or may be closer to normally distributed if it is not than we have to make it look like Gaussian distributed (closer to normal) by taking care of Outliers and skewness in the data. Outliers can affect a regression model by pulling our estimated regression line further away from the true population regression line. So, we remove or replace those observations from our data. We have to remove this Outliers from the dataset. Basic approach to make data look normal is using z score as we know 99.7% data lies between -3 to +3 and if data loss is greater than 10%, we can reduce it by adjusting the threshold value. Or we can use interquartile method to remove outliers. Here I have used the interquartile method to remove outliers from the continues types of columns and kept the outliers present in discrete or categorical columns.

```
q1=df.quantile(0.25)
q3 = df.quantile(0.75)
iqr = q3 - q1

upper_limit = q3["duration_hours"] + (1.5*iqr["duration_hours"])
index = np.where(df["duration_hours"] > upper_limit)
```

```
df.drop(df.index[index],axis=0,inplace=True)
```

```
print("Data loss is",((14697-df.shape[0])/14697)*100,"Percent")
```

And applying Power Transformer method to remove skewness from the continuous columns

```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer()
df[["duration_hours","Arrival_hour","stops"]] = pt.fit_transform(df[["duration_hours","Arrival_hour","stops"]])
```

Hardware requirement: -

- Minimum core i5 or higher
- Minimum 8gb of ram
- Sufficient disk space

Software and tools required:

- Python is widely used in scientific and numeric computing:
- SciPy is a collection of packages for mathematics, science, and engineering.
- Pandas is a data analysis and modelling library.

Modules or library required for project data analysis and visualization:

- Pandas for data analysis and import
- NumPy to perform Mathematical operation
- Seaborn and matplotlib to data visualization
- Scikit-learn: All the models, metrics and feature selection etc are present inside of that module. We import from this library according to our need

Modelling: -

- Suppose Our data is normally distributed and we have start checking the general statistics and trying to make the data normally distributed or close to normal distributed by using varies technique.

- Convert all the categorical columns into numerical as we know our models only understand the numerical data

-
- First we need to split the data into dependent and independent variables.

```
# Splitting dataset for model building
x = df.drop("fare",axis=1)
y= df["fare"]
```

- Normalizing the features so that our model did not get biased toward a particular feature

```
scale = StandardScaler()
x_scaled = scale.fit_transform(x)
```

- Splitting the data we get after scaling in to training and testing so that we can also evaluate the performance if the model before passing the actual testing data.

```
x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.30,random_state=27)
```

Model Building:-

Once the data is clean and we have gained insights about the dataset, we can apply an appropriate machine learning model that fits our dataset. We have selected four algorithms to predict the dependent variable in our dataset. The four algorithms are Gradient Boosting Regression, Linear Regression, Random Forest Regression and Bagging Regressor and for the confidence of the model or evaluating the performance of the model we are using r2 score and Cross validation score to select the best model. As we know the difference between the r2 score and cross validation score is minimum for the best model and on that basis, we are going to select Best model.

- We can't depend upon the single model that's why we are building multiple models

1. GradientBoosting Regression

```
gbc = GradientBoostingRegressor(n_estimators=500,min_samples_split=3)
gbc.fit(x_train,y_train)
```

```
GradientBoostingRegressor(min_samples_split=3, n_estimators=500)
```

```
y_pred = gbc.predict(x_test)
```

```
r = r2_score(y_test,y_pred)
c = cross_val_score(gbc,x_scaled,y,cv=kf).mean()
```

```
r2.append(r)
cross.append(c)
diff.append(r-c)
```

```
print(" R2 Score is :",r,"\n","=-"*60,"\n Cross validation score :",c,"\n","=-"*60,"\n Training score :",gbc.score(x_train,y_trai
```

```
 R2 Score is : 0.8312881730527241
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
 Cross validation score : 0.8207891976801441
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
 Training score : 0.8526124633123078
```

After instantiating model into an object we used .fit method to pass training data to model for learning and understanding. After .fit we use .predict to predict the test data to know the accuracy and cross_validation_score

2. RandomForestRegressor: -

```
rf = RandomForestRegressor(n_estimators=200,min_samples_split=4)
rf.fit(x_train,y_train)
```

```
RandomForestRegressor(min_samples_split=4, n_estimators=200)
```

```
y_pred = rf.predict(x_test)
```

```
r = r2_score(y_test,y_pred)
c = cross_val_score(rf,x_scaled,y,cv=kf).mean()
```

```
r2.append(r)
cross.append(c)
diff.append(r-c)
```

```
print(" R2 Score is :",r,"\n","=-"*60,"\n Cross validation score :",c,"\n","=-"*60,"\n Training score :",rf.score(x_train,y_trair
```

```
 R2 Score is : 0.8890972882948567
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
 Cross validation score : 0.8822154251302654
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
 Training score : 0.9720610222048879
```

3. XGBRegressor: -

```
xgb = XGBRegressor(max_depth=8,learning_rate=0.1)
xgb.fit(x_train,y_train)
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.1, max_delta_step=0,
             max_depth=8, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=8,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

```
y_pred = xgb.predict(x_test)
```

```
r = r2_score(y_test,y_pred)
c = cross_val_score(xgb,x_scaled,y,cv=kf).mean()
```

```
r2.append(r)
cross.append(c)
diff.append(r-c)
```

```
print(" R2 Score is :",r,"\n","=-"*60,"\n Cross validation score :",c,"\n","=-"*60,"\n Training score :",xgb.score(x_train,y_trai
```

```
 R2 Score is : 0.8898413894040613
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
 Cross validation score : 0.8847309870270884
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
 Training score : 0.9489926200637169
```

4. Bagging Regressor

```
bag_reg = BaggingRegressor(n_estimators=200)
bag_reg.fit(x_train,y_train)
```

```
BaggingRegressor(n_estimators=200)
```

```
y_pred = bag_reg.predict(x_test)
```

```
r = r2_score(y_test,y_pred)
c = cross_val_score(bag_reg,x_scaled,y,cv=kf).mean()
```

```
r2.append(r)
cross.append(c)
diff.append(r-c)
```

```
print(" R2 Score is :",r,"\n","=-"*60,"\n Cross validation score :",c,"\n","=-"*60,"\n Training score :",bag_reg.score(x_train,y_
```

```
 R2 Score is : 0.8902490795441946
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
 Cross validation score : 0.8840197710746004
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
 Training score : 0.9814552973198455
```

5. Decision Tree Regressor:

```
dt = DecisionTreeRegressor(max_depth=10,min_samples_split=4)
dt.fit(x_train,y_train)
```

```
DecisionTreeRegressor(max_depth=10, min_samples_split=4)
```

```
y_pred = dt.predict(x_test)
```

```
r = r2_score(y_test,y_pred)
c = cross_val_score(dt,x_scaled,y,cv=kf).mean()
```

```
r2.append(r)
cross.append(c)
diff.append(r-c)
```

```
print(" R2 Score is :",r,"\n","=-"*60,"\n Cross validation score :",c,"\n","=-"*60,"\n Training score :",dt.score(x_train,y_trair
```

```
 R2 Score is : 0.7542625729621902
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
 Cross validation score : 0.7412408030968738
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
 Training score : 0.8175052263449207
```

Above you can see the snapshots of the model which I have used to predict the dataset.

Metrics used to select the model: -

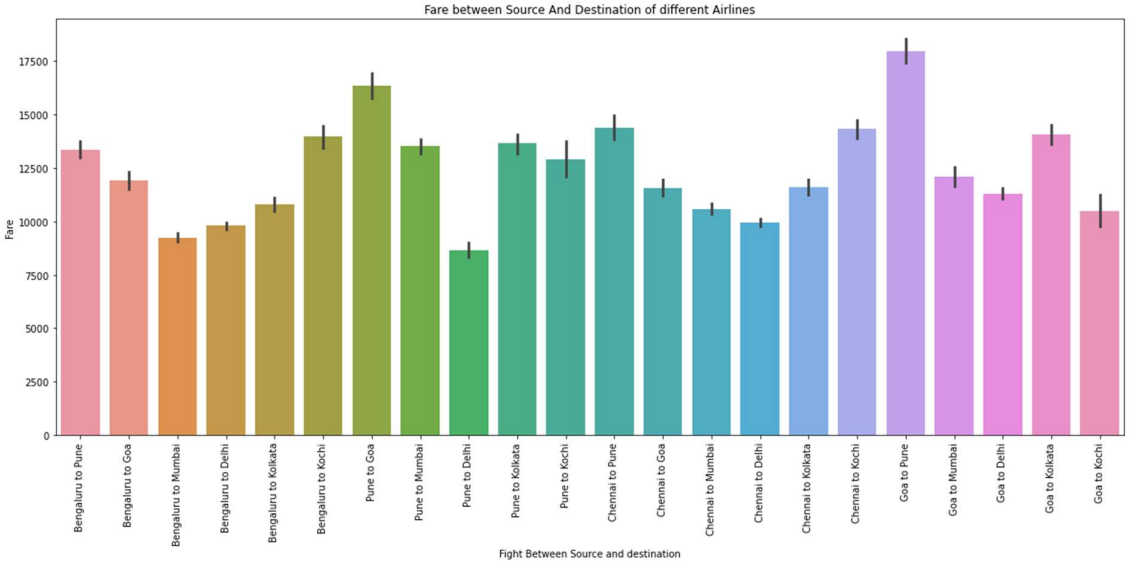
R2 (coefficient of determination) regression score function: - **R-squared (R2)** is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model.

KFold Cross validation score: - KFold Cross-validation is a statistical method used to estimate the skill of machine learning models. That k-fold cross validation is a procedure used to estimate the skill of the model on new data.

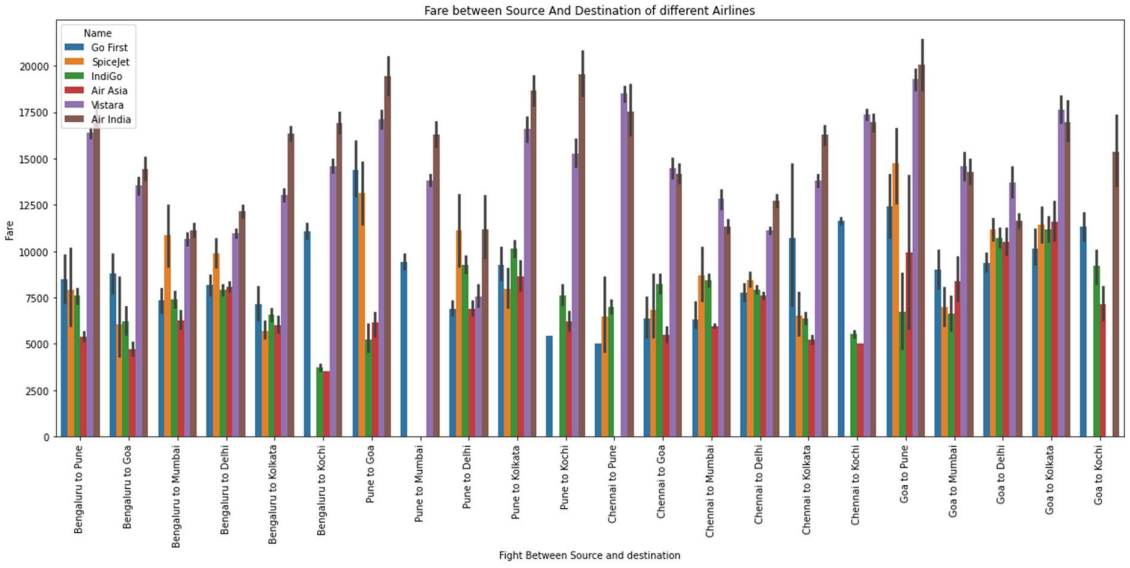
Here we are selecting the model on R2 score and cross; validation score. For those model who have least the difference between the R2 score and cross validation score that model is best.

Visualisation: -

Data visualization is the discipline of trying to understand data by placing it in a visual context so that patterns, trends and correlations that might not otherwise be detected can be exposed.

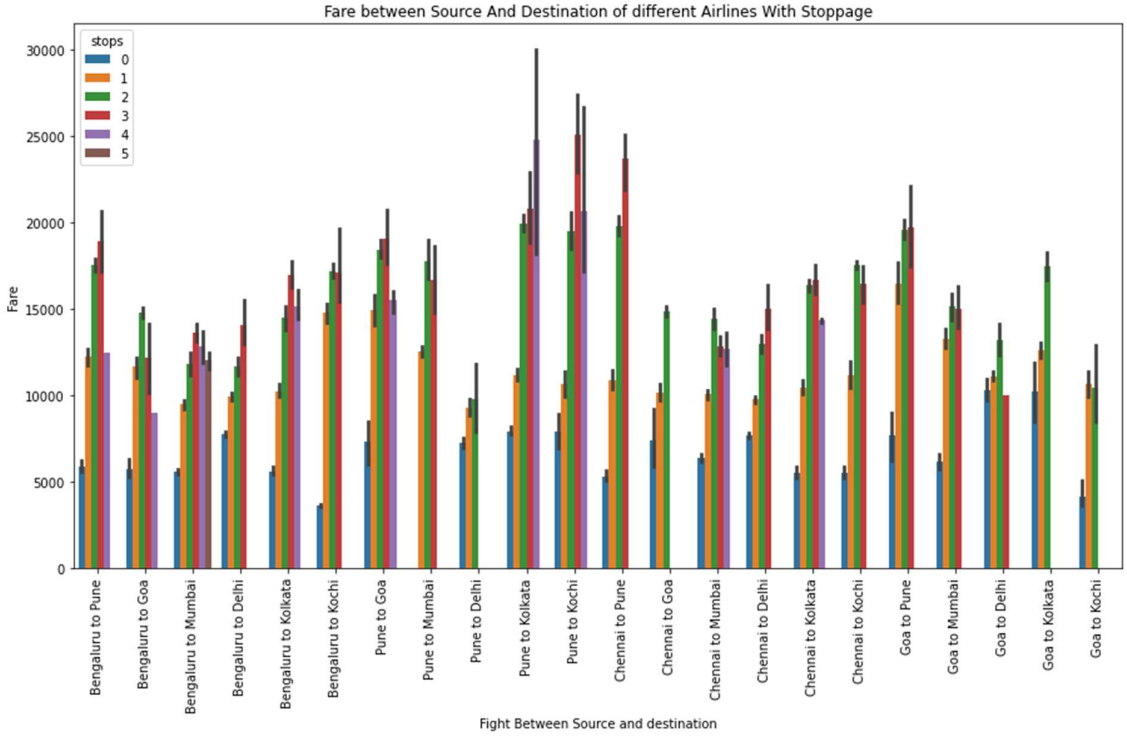


Above graph is showing the fare between the source and destination

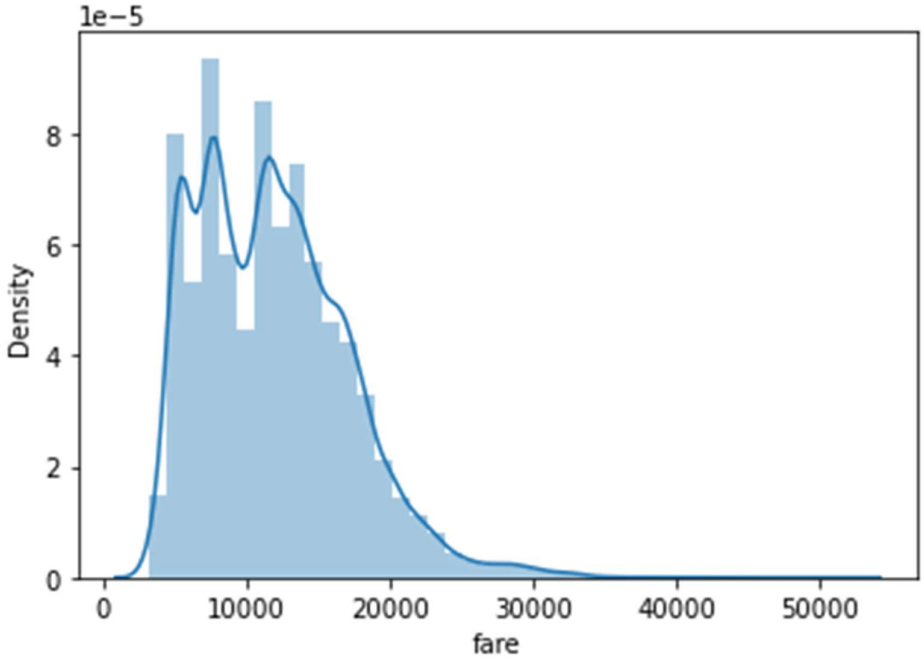


Above graph is showing the fare between source and destination by airlines name

Fare between source and destination with no. of stops. As we can see that as the no. of stops increases till 3 the fare is also increases but the flight which is having more than 3 stops their fare start declining

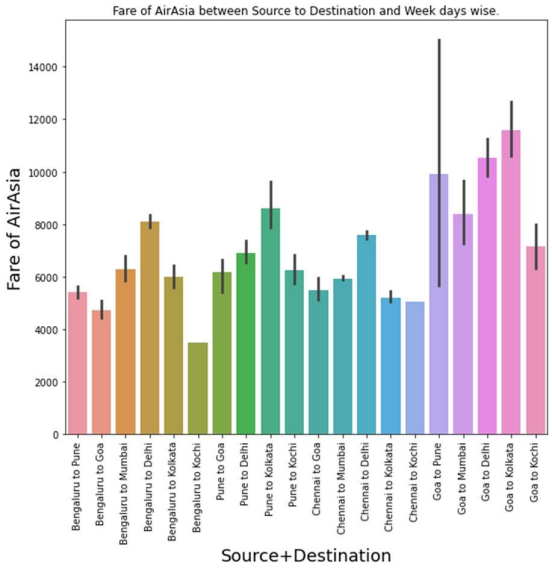


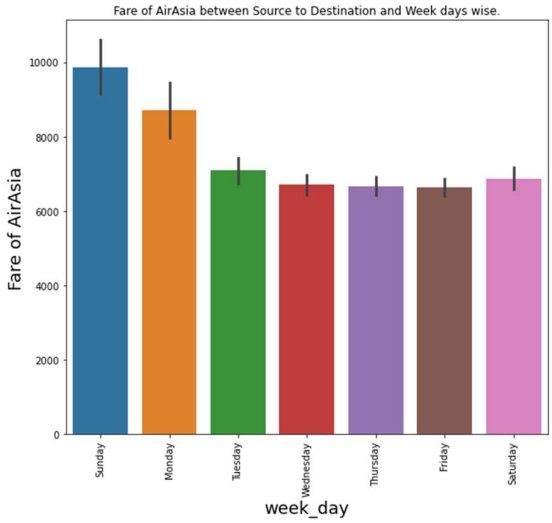
Ticket price distribution:

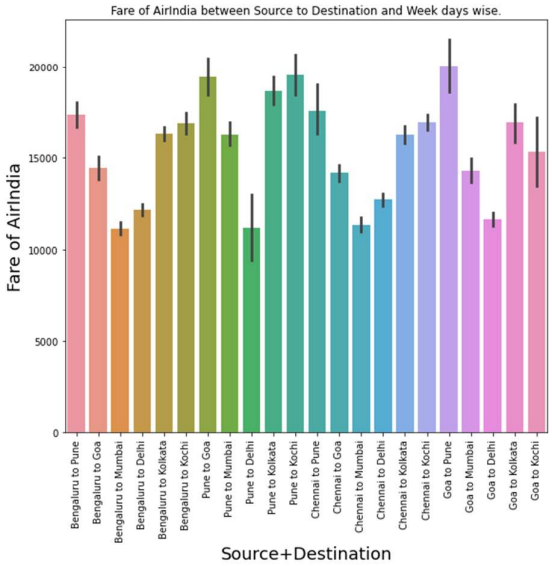


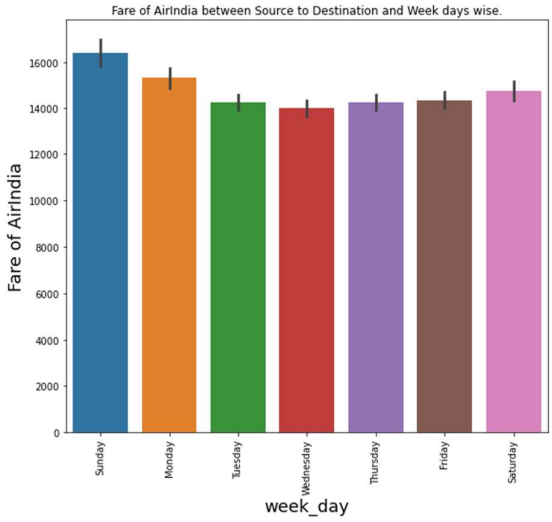
There is skewness in the fare column

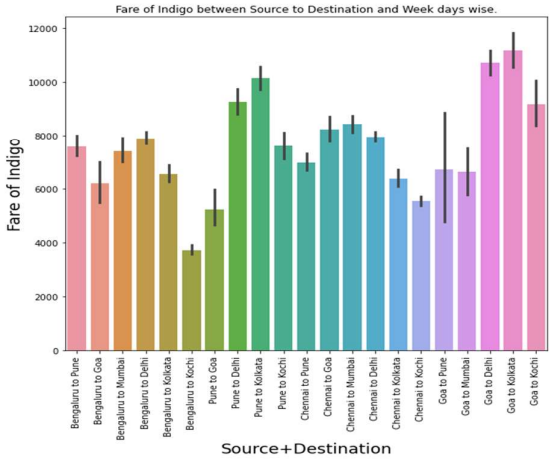
Below graph shows the Fare of particular Airlines from source and destination and day wise

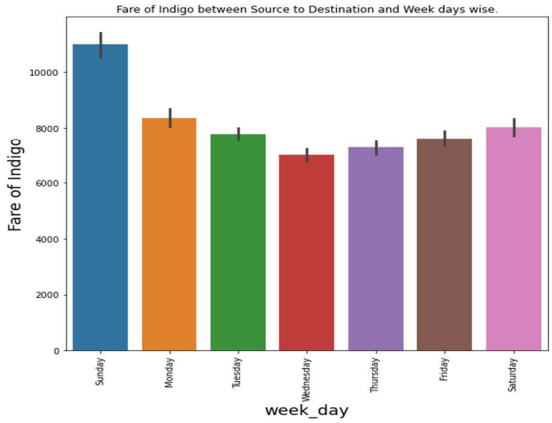


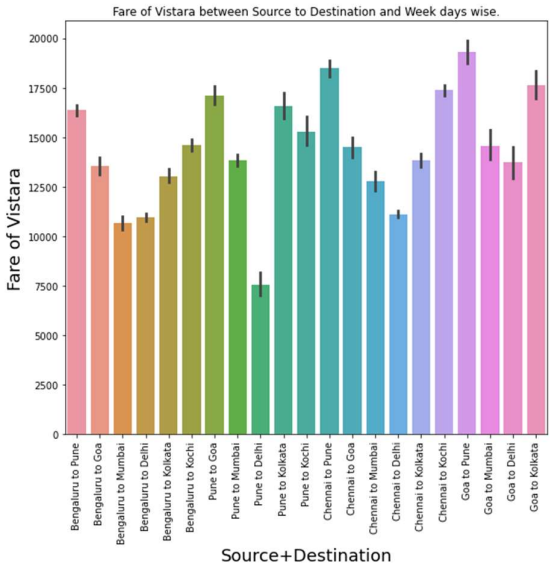


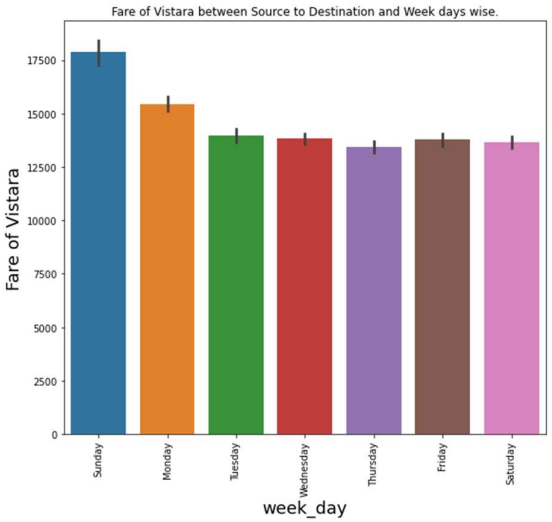


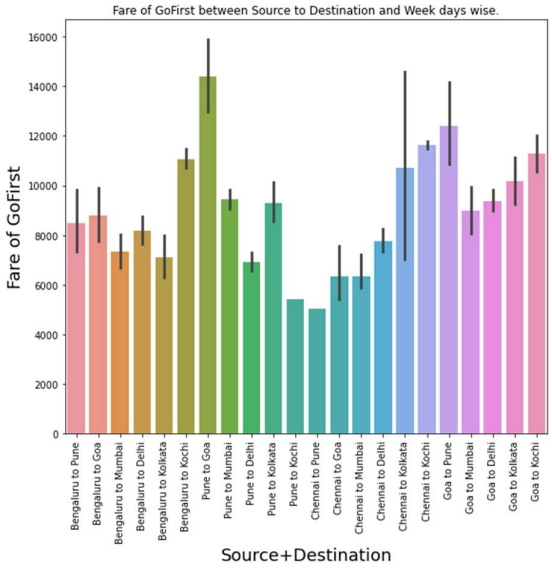


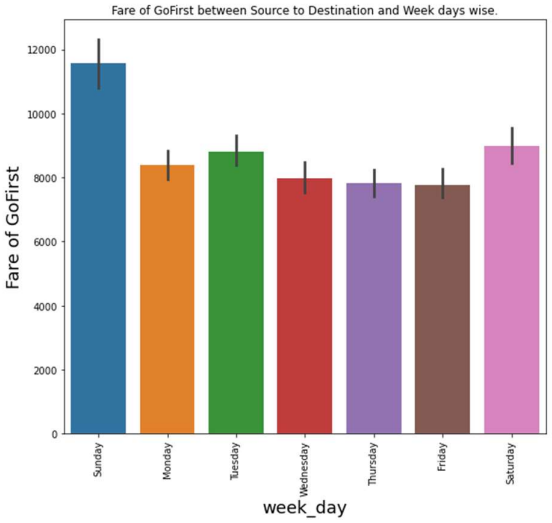




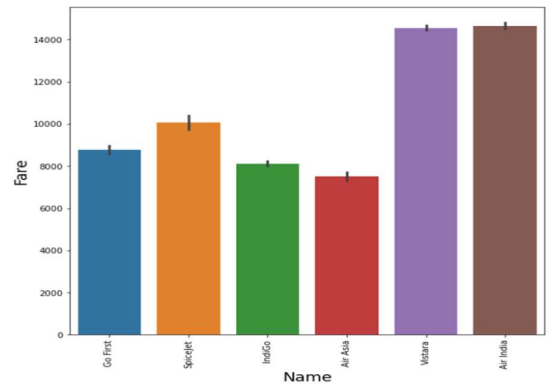


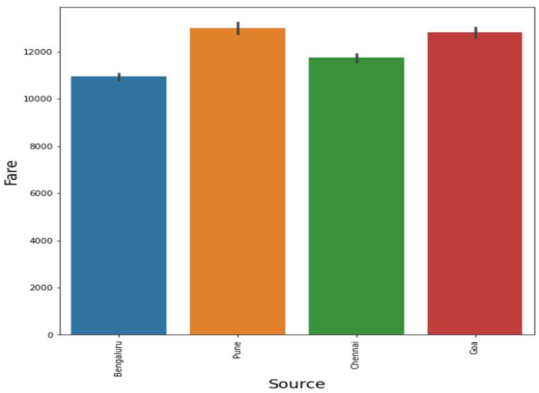


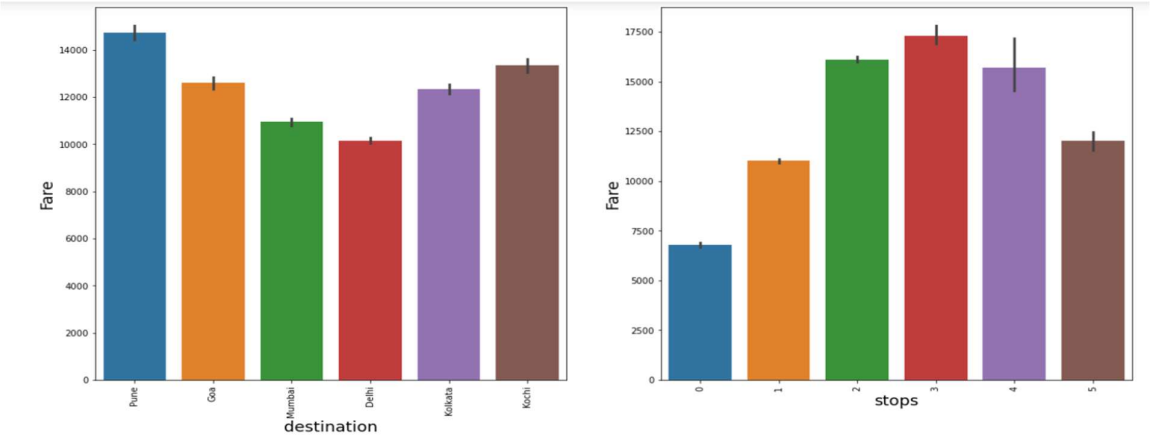




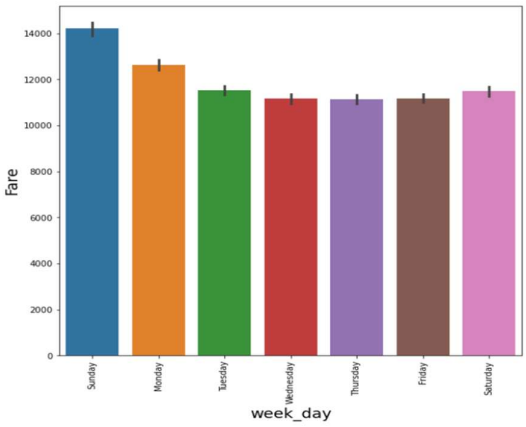
From Above all the Graph of airlines wise. Prices of ticket between Pune and Goa is Higher as comapred to other source and destination. And if we see days wise data we can see that the fare is higher On **Sunday, Monday and Saturday** and Flight fare is less on **Wednesday** as compare to other day

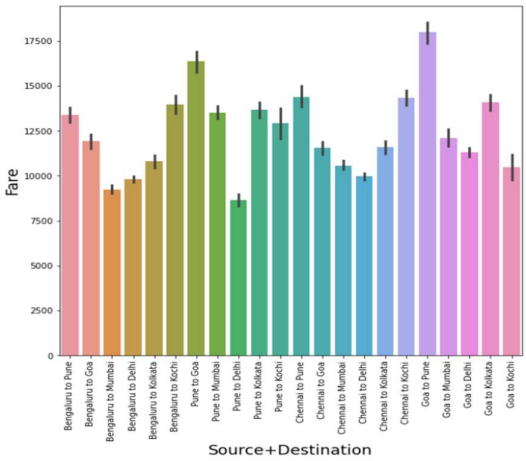


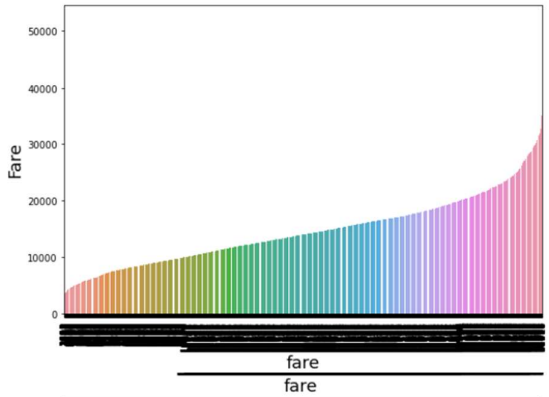


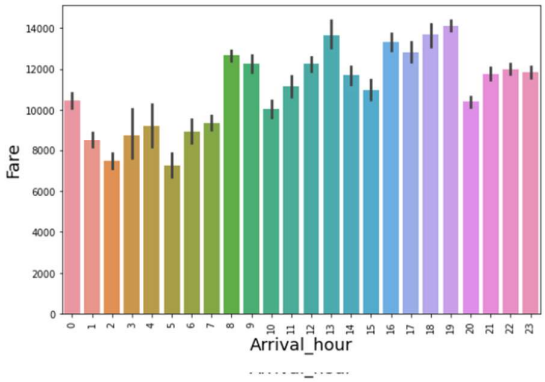

Fare
destination
Pune
Goa
Mumbai
Delhi
Kolkata
Kochi

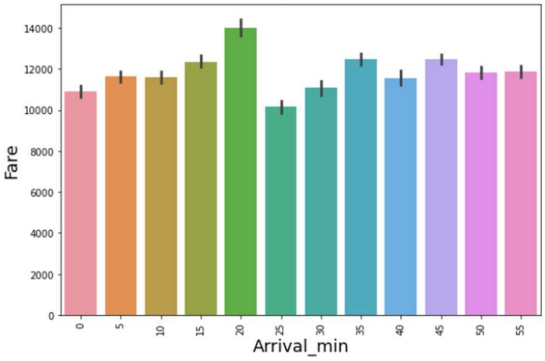
Fare
stops

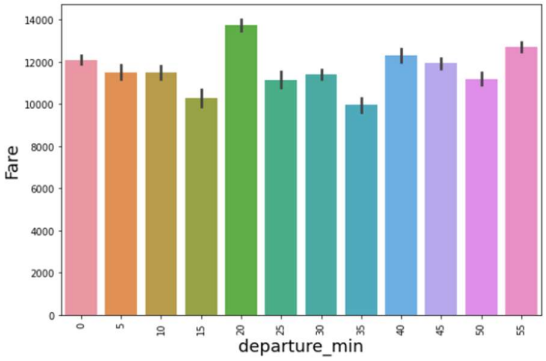

Fare
week_day
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday

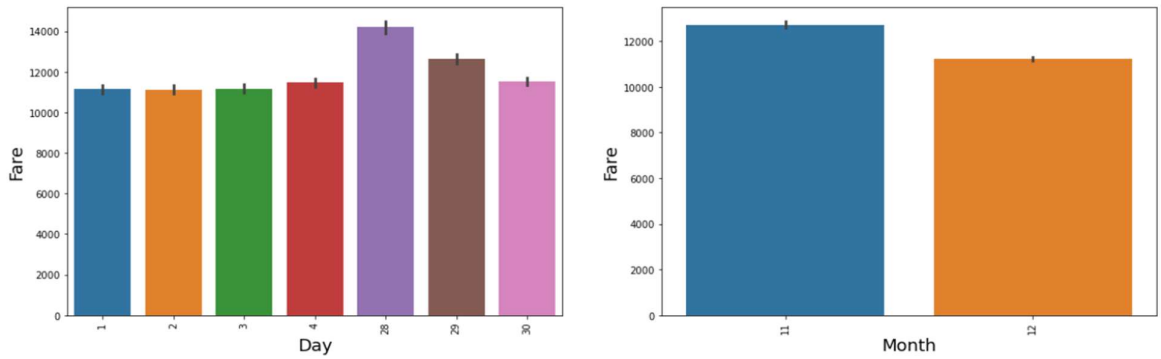

Fare
Source+Destination


Fare
fare

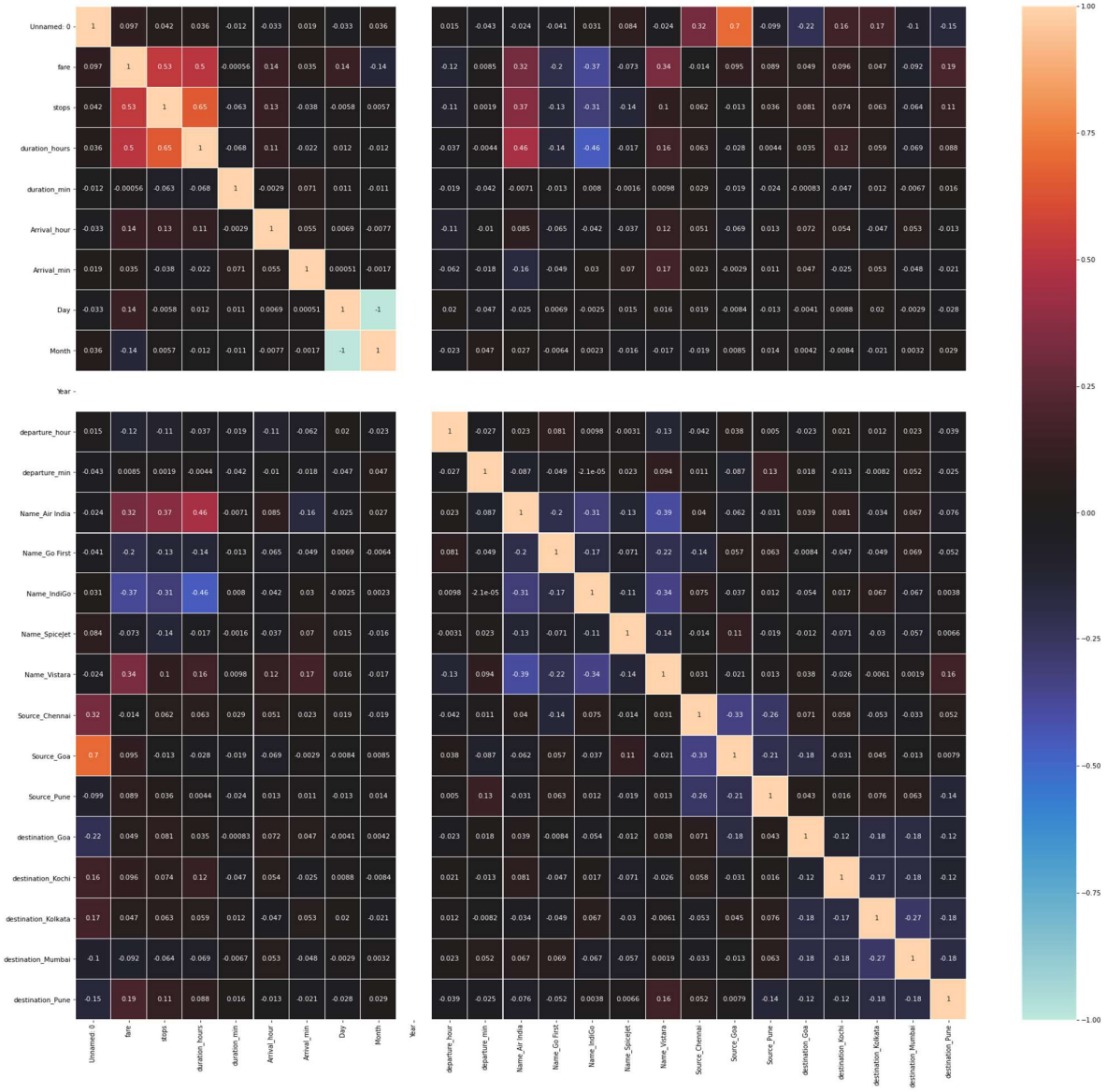

Fare
Arrival_hour


Fare
Arrival_min

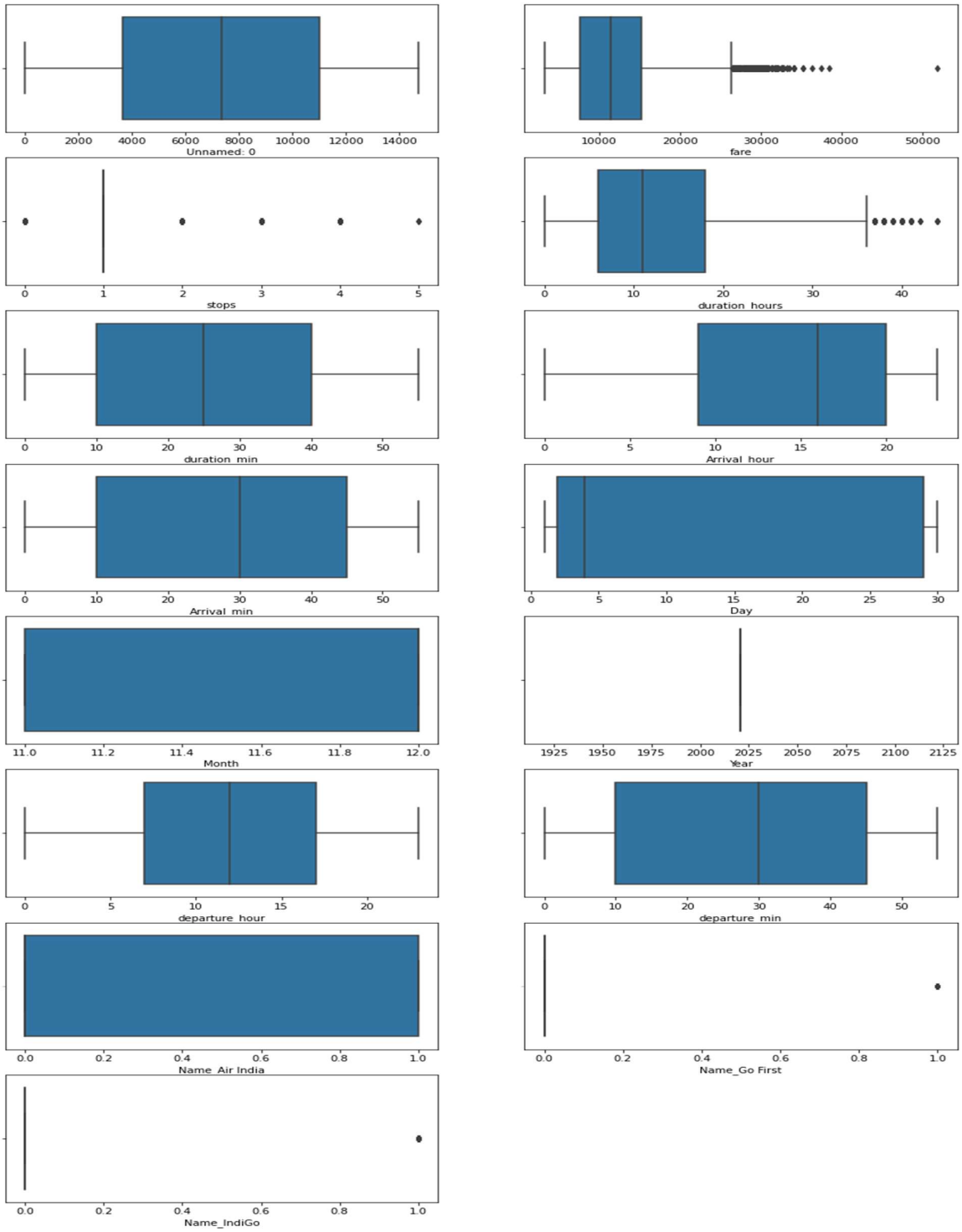

Fare
departure_min



If we see the Day graph we can see that the fare of the airlines of the very next day or following 2-3 days is higher and when we move far from the current date the price of the ticket is decreases. Or we can say that if we book ticket prior one or two month before the date_of_journey the ticket fair will cost us less expensive as compared to urgent booking.



It seems that Day and Month has negative multicollinearity. **Year** has no correlation with the target Variable



As we can see Only Fare, Stops and duration_hours has outliers. Fare is our target column whereas stops is discrete columns

Interpretation of the Results

- Airfare varies according to the day of the week of travel. It is higher for weekends and Monday and slightly lower for the other days.

- Booking on urgent basis cost too much as compared to book a ticket prior a month from the data of journey

- We can see that Pune to Goa fare is high of all the airlines as compared to other destination

- The fare goes up with the no. of stoppage till 3 stops. After 3 stop the price start declining.

Conclusion

- Flight prices almost always remain constant or increase between the major cities.
- Routes with data collected over the longer duration of time tend to facilitate with much more accurate predictions in the model and thus lead to higher average savings

Future Work

- More routes can be added and the same analysis can be expanded to major airports and travel routes in India.
- The analysis can be done by increasing the data points and increasing the historical data used. That will train the model better giving better accuracies and more savings.
- More rules can be added in the Rule based learning based on our understanding of the industry, also incorporating the offer periods given by the airlines.
- Developing a more user-friendly interface for various routes giving more flexibility to the users.

References

- O. Etzioni, R. Tuchinda, C. A. Knoblock, and A. Yates. To buy or not to buy: mining airfare data to minimize ticket purchase price.
- Manolis Papadakis. Predicting Airfare Prices.
- Groves and Gini, 2011. A Regression Model For Predicting Optimal Purchase Timing For Airline Tickets.
- Modeling of United States Airline Fares – Using the Official Airline Guide (OAG) and Airline Origin and Destination Survey (DB1B), Krishna Rama-Murthy, 2006.
- B. S. Everitt: The Cambridge Dictionary of Statistics, Cambridge University Press, Cambridge (3rd edition, 2006). ISBN 0-521-69027-7.
- Bishop: Pattern Recognition and Machine Learning, Springer, ISBN 0-387-31073-8.
- E. Bachis and C. A. Piga. Low-cost airlines and online price dispersion. International Journal of Industrial Organization, In Press, Corrected Proof, 2011.
- P. P. Belobaba. Airline yield management. an overview of seat inventory control

Transportation Science, 21(2):63, 1987.

- Y. Levin, J. McGill, and M. Nediak. Dynamic pricing in the presence of strategic consumers and oligopolistic competition. Management Science, 55(1):32–46, 2009.