Article submission on Flight Price Prediction

Problem Definition:

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 travelers saying that flight ticket prices are so unpredictable. Airlines implement dynamic pricing for their tickets, and base their pricing decisions on demand estimation models. The reason for such a complicated system is that each flight only has a set number of seats to sell, so airlines have to regulate demand. In the case where demand is expected to exceed capacity, the airline may increase prices, to decrease the rate at which seats fill. On the other hand, a seat that goes unsold represents a loss of revenue, and selling that seat for any price above the service cost for a single passenger would have been a more preferable scenario.

Problem Solution:

Building a ML model which is going to predict the close value of flight ticket price based the inputs like particular date, time, route and flight service provider.

For that we are going to use the previous data based on which they are fixing the ticket price

So we are going to feed the data to the machine learning model train it and make it ready to take the inputs and provide the output i.e., the ticket price.

For train the ML we need the various inputs of data like **Airline, Date_of_Journey, Source, Destination, Route, Departure Time, Arrival_Time, Duration, Total_Stops, Additional_Info**.

What is Machine Learning? Why we need to feed it with Data?:

Machine Learning is an idea to learn from examples and experience, without being explicitly programmed. Instead of writing code, you feed data to the generic algorithm, and it builds logic based on the data given. The process of training an ML model involves providing an ML algorithm (that is, Machine Learning Algorithm) with training data to learn from. The regression equations can be built on structured data. The main goal of building the equation is to identify the best fit equation to which a new data will fall under. We can use these ML models to get predictions on new data for which target is unknown.

II. GENRAL STRUCTURE

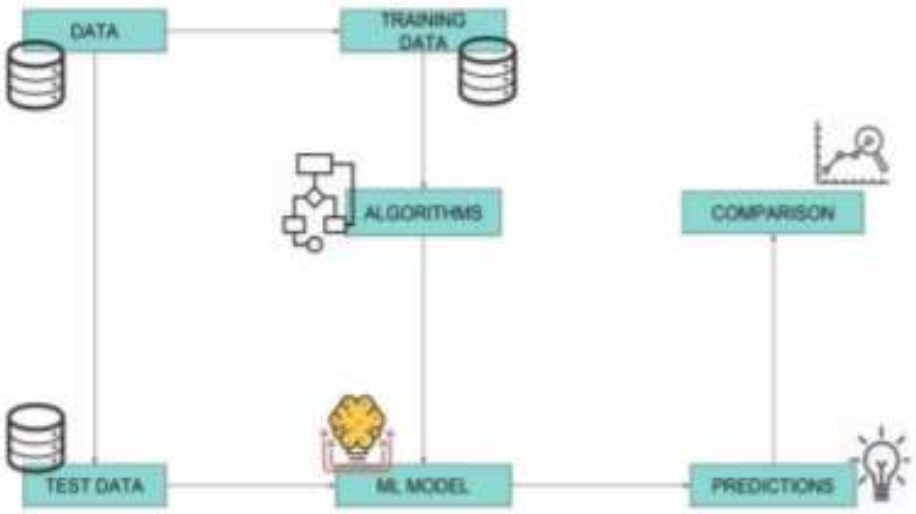


Fig : General Structure of ML Model

The above figure represents the general structure of an ML model. The data are split into Training and Test Data respectively. The Training Data is passed to trough the ML algorithms for enabling the machine to learn and apply it on the test data to predict the solutions. The such predicted solutions can be used for comparisons, calculate the accuracy of the prediction.

Various Machine Learning Steps in Building it:

1.Define required libraries with which the model will optimized.

2.Data Cleansing and Wrangling.

3.Feature Engineering

4.Data pre-processing

5.Feature selection

6.Split the data into training and testing

7.Model selection

8.Model validation

9.Interpret the results

We are going to use the above steps on the one of the data set provided for flight price prediction.

1.Define required libraries with which the model will optimized.

Various Libraries which are going to use in the importing of data, analyzing the data, optimizing it, pre-processing it and to feed it to the machine to learn are import into our canvas of the jupyter note book. Those are as stated below.

1. Pandas Library, Numpy Library for data download and load it into 2D frame

```
# we will import the libraries at the time of required while we are progressing with our machine building
# as of now we are importing the pandas library for the purpose of getting the data and load it into the DataFrame
import pandas as pd
import numpy as np
```

```
#Loading the data
trdata=pd.read_csv('Train_data.csv',delimiter=' ')
tedata=pd.read_csv('Test_set.csv',delimiter='   ')
```

2. Visual Libraries for graphical analysis.

```
# importing the required libraries
import seaborn as sns
import matplotlib.pyplot as plt
```

3. Encoders from sklearn library preprocessing module for convert the label data to numeric

```
# Before this we need to convert the string data to numeric data
# by using lable encoders
# importing the encoders from the sklearn
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
lb=LabelEncoder()
for i in trs.columns:
    if trs[i].dtype==object:
        trs[i]=lb.fit_transform(trs[i])
```

4. power_transform method from sklearn library preprocessing module to reduce the skewness in data.

```
# importing the required libraries
from scipy.stats import boxcox
from sklearn.preprocessing import power_transform
```

5. scaling methods from sklearn library preprocessing module to reduce the magnitude of the data

```
#importing minmax scaler
from sklearn.preprocessing import MinMaxScaler, StandardScaler
#minamx scaler reduce the value of data in between 0 and 1
```

6.other machine learning libraries to tune the best model, train it and use for prediction.

[53]:
```
# importing the required libraries
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
```

Are the some of the various libraries which are commonly using in the building of the machine learning steps.

EDA (Exploratory Data Analysis):

First we have to look at the data various data types information in the columns

```
data.sample(10)
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
575	Air India	09/03/2019	Banglore	New Delhi	BLR → DEL	21:10	23:55	2h 45m	non-stop	No info	4966
4506	Jet Airways	9/05/2019	Kolkata	Banglore	CCU → BOM → BLR	20:00	04:40 10 May	8h 40m	1 stop	In-flight meal not included	8586
6460	Jet Airways	27/05/2019	Delhi	Cochin	DEL → BOM → COK	20:55	04:25 28 May	7h 30m	1 stop	No info	16079
4272	Air India	3/05/2019	Banglore	Delhi	BLR → DEL	10:00	12:45	2h 45m	non-stop	No info	6121
5397	Jet Airways	6/06/2019	Delhi	Cochin	DEL → BOM → COK	19:15	04:25 07 Jun	9h 10m	1 stop	In-flight meal not included	10262
4523	Jet Airways	12/06/2019	Delhi	Cochin	DEL → BOM → COK	11:30	12:35 13 Jun	25h 5m	1 stop	In-flight meal not included	10262
9501	Air India	21/06/2019	Mumbai	Hyderabad	BOM → HYD	21:05	22:25	1h 20m	non-stop	No info	3100
8526	Jet Airways	18/05/2019	Kolkata	Banglore	CCU → DEL → BLR	20:25	21:05 19 May	24h 40m	1 stop	No info	14151
9641	IndiGo	21/06/2019	Chennai	Kolkata	MAA → CCU	13:15	15:35	2h 20m	non-stop	No info	3597

2.Data Cleansing and Wrangling.

then we have to look into the data for various information like the no of rows, columns, data types, missing values, etc.

```
3]: trdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64 
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

Observation:

```
1.there are 11 fields on the data
2.out of 11 fields 10 are string data type and the target column is int dtype
3.maximum entries in each column are 10683
4. no null values are present in the data
```

The missing values can be replaced with proper suitable values once after checking the skewness, outliers etc.,

Missing values can be filled up with the mean of the variables only when there are no outliers, no skewness. Otherwise the missing data is can be filled up with median for numeric data and in category type the missing values will be filled up with the most occurring one in particular case.

If no clue is getting to fill the missing data the corresponding rows can be dropped from the data set.

same for the testing data

In [24]:

```
# splitting of the Date of Journey to days month and years
tes['Date_of_Journey']=tes['Date_of_Journey'].str.split('/')
tes['Journey_Date']=tes['Date_of_Journey'].str[0]
tes['Journey_Month']=tes['Date_of_Journey'].str[1]
tes['Journey_Year']=tes['Date_of_Journey'].str[2]
tes.drop(['Date_of_Journey'], axis=1, inplace=True)
# splittig of departure time into hours and minutes
tes['Dep_Time']=tes['Dep_Time'].str.split(':')
tes['Dep_Hours']=tes['Dep_Time'].str[0]
tes['Dep_Minute']=tes['Dep_Time'].str[1]
tes.drop(['Dep_Time'],axis=1,inplace=True)
#splitting the route
tes.Route=tes.Route.str.split('→')
tes['city1']=tes['Route'].str[0]
tes['city2']=tes['Route'].str[1]
tes['city3']=tes['Route'].str[2]
tes['city3']=tes['Route'].str[3]
tes['city4']=tes['Route'].str[4]
tes['city5']=tes['Route'].str[5]
tes.drop(['Route'],axis=1,inplace=True)
#splitting the arrival time
tes['Arrival_Time']=tes['Arrival_Time'].str.split(':')
tes['Arrival_Hour']=tes['Arrival_Time'].str[0]
tes['Arrival_Minute']=tes['Arrival_Time'].str[1]
tes.drop(['Arrival_Time'],axis=1,inplace=True)
tes['Arrival_Minute']=tes['Arrival_Minute'].str.split(' ')
tes['Arrival_Minute']=tes['Arrival_Minute'].str[0]
#splitting the Duration into hours and minutes of journey
tes.Duration=tes.Duration.str.split(' ')
tes['Duration_Hours']=tes.Duration.str[0]
tes['Duration_Minutes']=tes.Duration.str[1]
tes.drop(['Duration'],axis=1,inplace=True)
```

After data cleaning is over we just split the columns into required ways for in further use. In the above screen shot the data column has been split into **year, month, day** called wrangling.

In that way the required column can be created and unnecessary columns can be dropped from the data set before going to further analysis of the data.

In this Flight price prediction model we need the particular data particular month, year on which the passengers are used to travel a lot so for that reason only the date column has been split into the other columns and date column has been dropped from the data set.

Statistical and Visual Analysis:

After wrangling we observe the statistical data of the data set and then the visual analysis of the data with univariant and bi variant analysis.

n [8]:
```
#Statistical Analysis
trs.describe(include='all')
```

ut[8]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
count	10683	10683	10683	10683	10682	10683	10683	10683	10682	10683	10683.000000
unique	12	44	5	6	128	222	1343	368	5	10	NaN
top	Jet Airways	18/05/2019	Delhi	Cochin	DEL → BOM → COK	18:55	19:00	2h 50m	1 stop	No info	NaN
freq	3849	504	4537	4537	2376	233	423	550	5625	8345	NaN
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	9087.064121
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4611.359167
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1759.000000
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5277.000000
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	8372.000000
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	12373.000000
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	79512.000000

Observations:

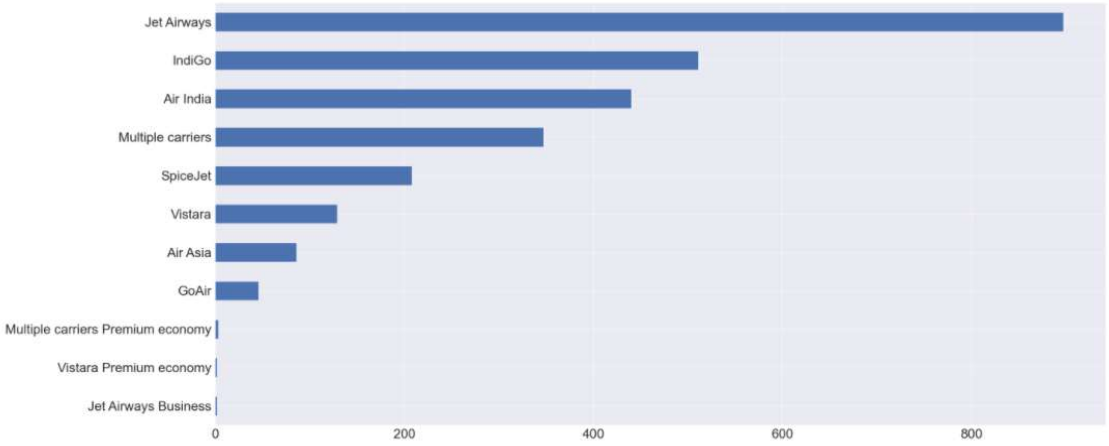
1.In Airline Column there are 12 types of airlines present with JetAirways as leading
2.As per Date of Journey the data is colleted on 44 dates
3.source data is from 5 with delhi as leading for travelling
4.Destination is for 6 types with Cochin as most destination
5.the flights are travelling in 128 Routes in which the most frequent use is del-bom-cok
6.the data colleted contains 222 different times
7.1343 different arrival times data was collected
8.time of travel is of 368 types in which the most passengers travelling 2h 50m.
9.Total stops are of 5 types in which single stop leads with 5625
10.additional info column contains mostly no information in its entries.

All the data I got here is classification type and I am going to change it into the numeric data by using the encoding techniques.

From the below univariant analysis we are going to see which airline is highly in transporting the people, which places they are preferring for travel most from which place they are travelling like that information.

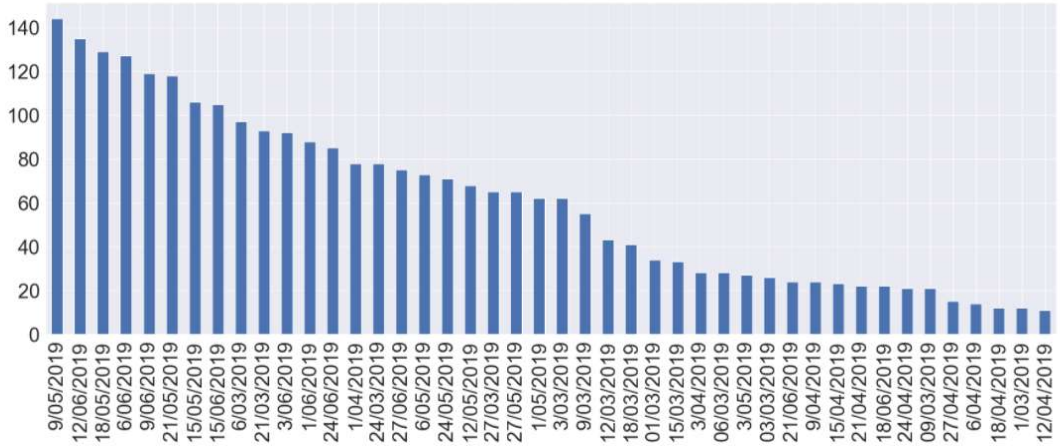
In [14]:
```
sns.set_theme(style='darkgrid',palette='deep',font_scale=3, color_codes=True)
plt.figure(figsize=(40,20))
#test data
tes.Airline.value_counts().sort_values(ascending=True).plot.barh(rot=0)
```

Out[14]: <AxesSubplot:>



it is clear that the people using the jet airways mostly follows Indigo,Air India

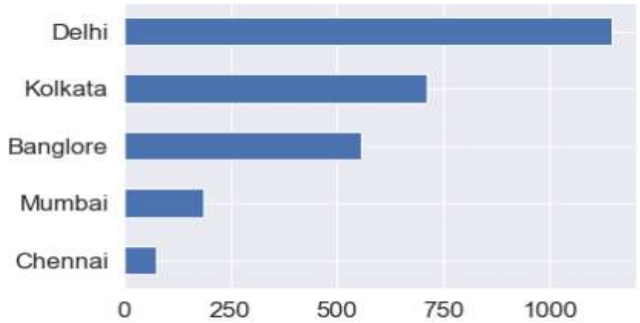
Out[16]: <AxesSubplot:>



the journey on 18/05/2019 was compared to of all the date was high

In [18]:
```
#test data
tes.Source.value_counts().sort_values(ascending=True).plot.barh(rot=0)
```

Out[18]: <AxesSubplot:>



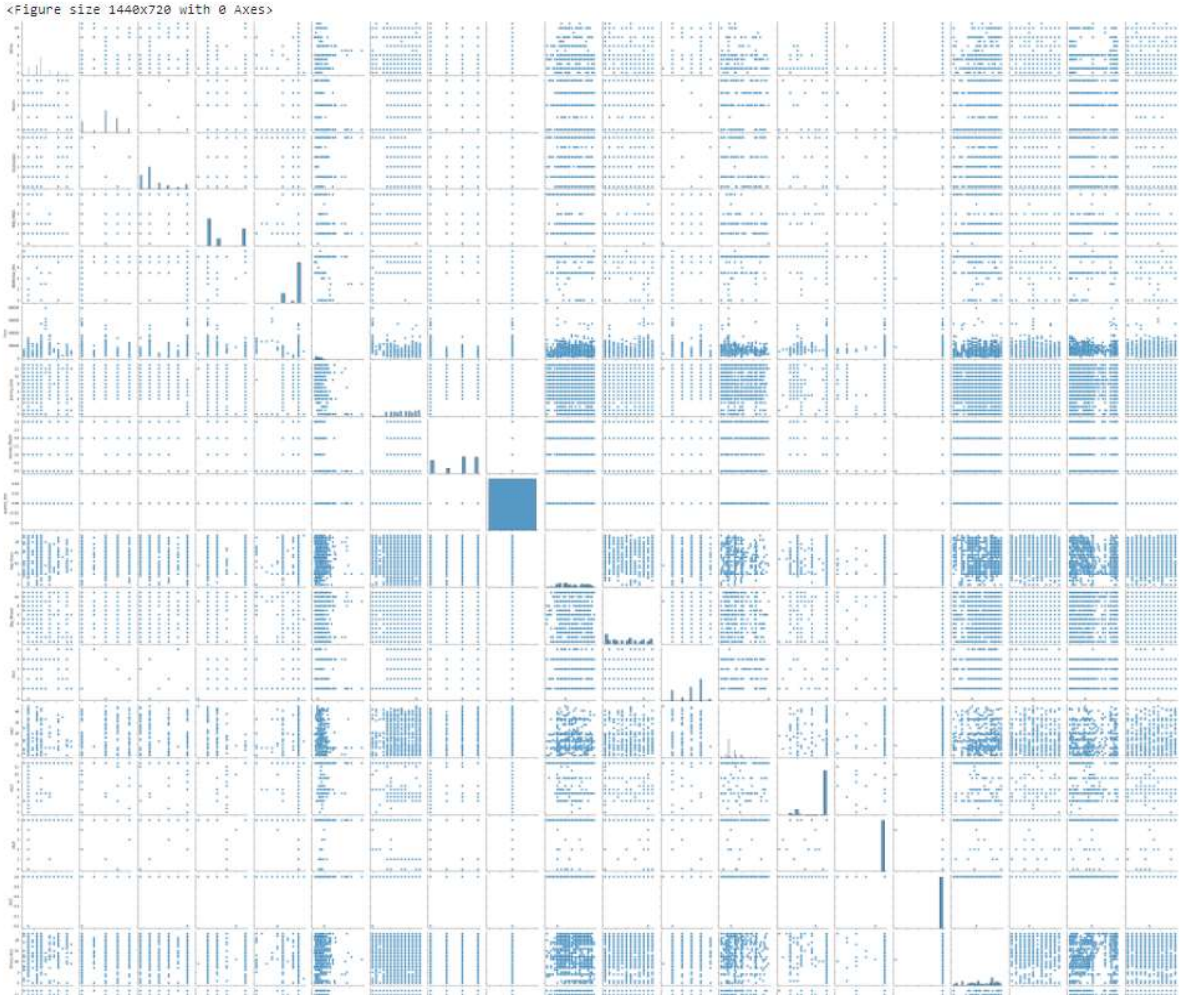
From both the data grphs we can see that the Delhi is preferred place of bording for journey

Bivariant Analysis

```
plt.figure(figsize=(20,10))
sns.pairplot(trs)
```

```
<seaborn.axisgrid.PairGrid at 0x1c4b2ddb700>
```

```
<Figure size 1440x720 with 0 Axes>
```



1.few airlines runs the flights in particular dates only remaining all are running flights in all days

2.not all airlines are operating in all the provided sources

3.majority of air lines covering all the routes

4.all airlines are operating all the times in a day

5.non stop flights are more compared to 5 stop flights

6.all airlines having all range of prices

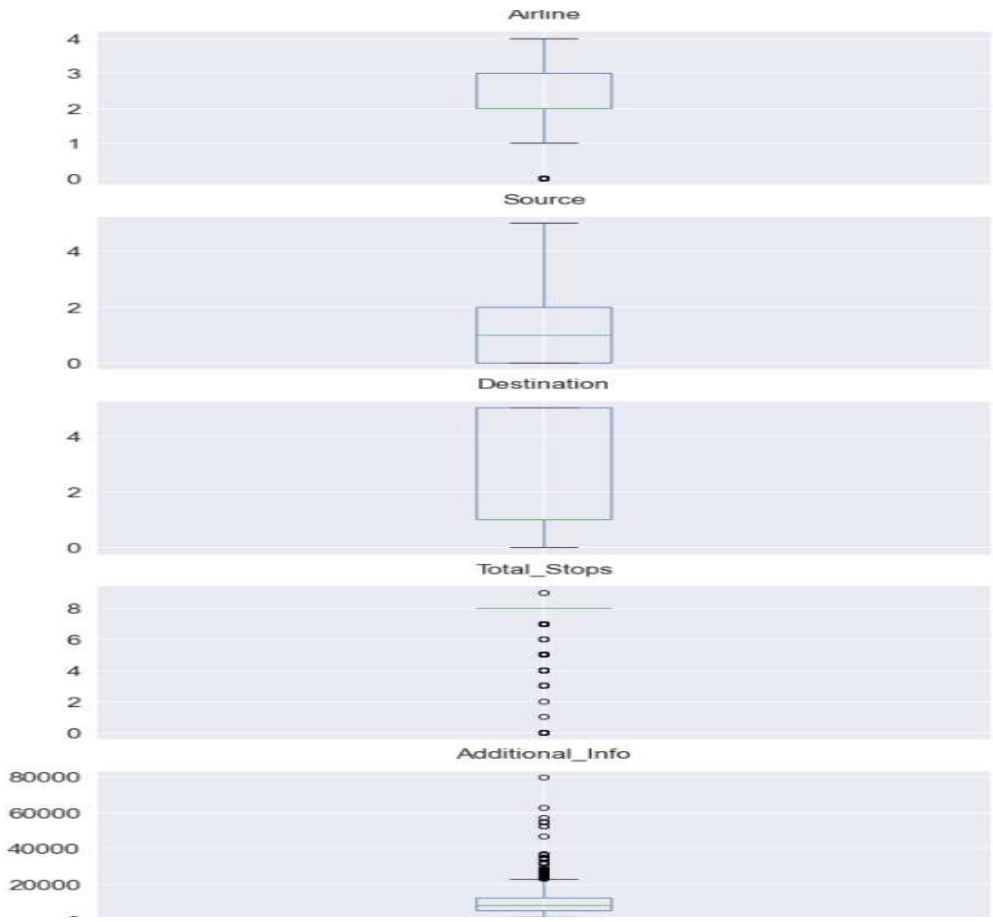
7. there is no direct or indirect relation between duration and no of stops more stops having less duration and more stops is for more duration.

8.price and duration has no relation directly or indirectly, all duration of travel have all range of ticket price

check for outliers:

we always use the box plot to check the outliers.

```
In [36]: sns.set_theme(style='darkgrid',palette='deep',font_scale=1.5, color_codes=True)
         trs.plot(kind='box',subplots=True, figsize=(10,80),layout=(20,1))
```



There are few outlier in the data but I am not going to remove them as the provided data is less and it is the data collected in real time of the general flight ticket booking events from the passengers.

If the data is having more outliers we use the zscore method to remove them. Just like below.

In [71]:
```
#cheking for outliers
#for that import the zscore from scipy library and numpy library
from scipy.stats import zscore
import numpy as np
```

In [72]:
```
z=np.abs(zscore(data.iloc[:,:-1]))
```

In [73]:
```
throshold=3
np.where(z>3)
```

Out[73]:
```
(array([ 16, 101], dtype=int64), array([3, 3], dtype=int64))
```

Only 2 values are obtained as outliers we should filter them to make the data as outliers free

In [76]:
```
data=data[(z<=3).all(axis=1)]
```

Checking for Skewness:

Skewness is the leaning of the data towards the left side or right side in the from the mean.

This can be check and rectify it if the skew values are >+/-0.5

In [38]:
```
trs.skew()
```

Out[38]:
```
Airline             0.731095
Source             -0.424054
Destination         1.244169
Total_Stops         0.631543
Additional_Info    -1.779838
Price               1.812552
Journey_Date       -0.307459
Journey_Month      -0.387493
Journey_Year        0.000000
Dep_Hours           0.113073
Dep_Minute          0.167029
city1              -0.618762
city2               1.426582
city3              -2.082590
city4             -16.012886
city5            -103.358599
Arrival_Hour       -0.369988
Arrival_Minute      0.110945
Duration_Hours      0.338322
Duration_Minutes    0.004741
dtype: float64
```

There are few columns in the data with high skew, we are going to reduce it by using the boxcox method or log transformation.

In [44]:
```
# city 4 and 5 have high skewness we are going to drop them
# Airline, Destination, Total Stops, Additional info, city 1,city 2 city 3 have skewness we will reduce it
tes.drop(['city4','city5'],axis=1,inplace=True)
```

In [45]:
```
# reduce skewness in test data
l=['Airline','Destination','Total_Stops','Additional_Info','city1','city2','city3']
for i in l:
    d1=np.array(tes[i])
    tes[i]=power_transform(d1.reshape(-1,1))
```

With the above method the skewness was reduced to limit of +/-0.5

Scaling:

After this we are going to reduce the magnitude of the variables in each field so that units of the values can be nullified and the magnitude decrease, by do so its greatly reduce the ML training time compared to unscaled data.

There are 2 methods which are commonly used for scaling

i.MinMaxScaler: fits the data between 0 and 1

ii.Standardscaler: fits the data by making the mean as 0

Scaling

In [49]:
```
#importing minmax scaler
from sklearn.preprocessing import MinMaxScaler, StandardScaler
#minamx scaler reduce the value of data in between 0 and 1
```

In [50]:
```
#training data scaling
scale=StandardScaler()
for i in x:
    d=np.array(x[i])
    scale.fit(d.reshape(-1,1))
    scale.transform(d.reshape(-1,1))
    x[i]=d
```

In [51]:
```
# testing data scaling
scale=StandardScaler()
for i in tes:
    d=np.array(tes[i])
    scale.fit(d.reshape(-1,1))
    scale.transform(d.reshape(-1,1))
    tes[i]=d
```

In [52]:
```
x.head()
```

Out[52]:

Additional_Info	Journey_Date	Journey_Month	Journey_Year	Dep_Hours	Dep_Minute	city1	city2	city3	Arrival_Hour	Arrival_Minute	Duration_Hours	Duration_Mi
0.524703	9	0	0	22	4	-1.554021	0.357673	0.414182	1	2	21	
0.524703	4	2	0	5	10	-0.157586	1.516072	-2.426307	13	3	41	
0.524703	13	3	0	9	5	0.831274	2.015656	-2.421196	4	5	9	
0.524703	5	2	0	18	1	-0.157586	2.143911	0.414182	23	6	38	
0.524703	0	0	0	16	10	-1.554021	2.143911	0.414182	21	7	37	

We used the standard scaler method to scale the data.

After scaling the data has been modified and reduced to its possible minimum values.

We are going to feature selection after done with data preprocessing.

Feature Selection:

For feature selection we are going to use the Hyper Parameter Tuning, this method can use the provided parameters to its model and select the best model out of the given models to test.

Hyper parameter tuning:

Hyper parameter tuning is nothing but the finding the correct parameters of a model so the model can perform at it highest performance state.

[53]:
```
# importing the required libraries
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
```

[54]:
```
# for that we are going to import
from sklearn.model_selection import GridSearchCV
```

The required libraries for the model and grid search cv can be imported to the note book.

Here we are going to test the regression model Liner, randomforest, adaboost and decision tree regressors to test in the grid search cv for the best performer.

In [258…

```
rf, ab,dt,lr=RandomForestRegressor(), AdaBoostRegressor(), DecisionTreeRegressor(), LinearRegression()
#creating parameters
model_par={'DecisionTreeRegressor':{'model':dt,'param':{'criterion' : ["mse", "friedman_mse", "mae", "poisson"],'splitter':['best','random']}},
           'RandomForestRegressor':{'model':rf,'param':{'n_estimators':[100,120,150],'criterion':['mse', 'mae'],'max_features':['auto','sqrt','log2'
           'AdaBoostRegressor':{'model':ab,'param':{'loss':['linear', 'square', 'exponential']}},
           'LinearRegression':{'model':lr,'param':{'fit_intercept':[True],'normalize':[True]}}
          }
scores=[]
for model_name, mp in model_par.items():
    model_selection=GridSearchCV(estimator=mp['model'],param_grid=mp['param'],cv=5,return_train_score=False)
    model_selection.fit(x,y)
    scores.append({'model':model_name,'best_score':model_selection.best_score_,'best_params':model_selection.best_params_})
scores
```

Out[258…

```
[{'model': 'DecisionTreeRegressor',
  'best_score': 0.9333952449669198,
  'best_params': {'criterion': 'mae', 'splitter': 'random'}},
 {'model': 'RandomForestRegressor',
  'best_score': 0.9527985342566975,
  'best_params': {'criterion': 'mse',
   'max_features': 'auto',
   'n_estimators': 100}},
 {'model': 'AdaBoostRegressor',
  'best_score': 0.27280495115240455,
  'best_params': {'loss': 'linear'}},
 {'model': 'LinearRegression',
  'best_score': 0.1090680513574348,
  'best_params': {'fit_intercept': True, 'normalize': True}}]
```

So as per the tuning the best model for this data is RandomForestRegressor with 95.27% accuracy and parameters of

'criterion': 'mse'

'max_features': 'auto'

'n_estimators': 100

Model Selection:

From the above grid search cv we can clearly see that the random forest regressor performs best for this data set So we can use RandomForestRegressor().

In [73]:

```
rf=RandomForestRegressor(criterion='mae',max_features='auto',n_estimators=100)
r=DecisionTreeRegressor(criterion='mae',splitter='random')
```

Model Tuning:

We are going to tune the model to get the maximum efficiency from it.

In [ ]:

```
I=[]
score=[]
ma=[]
ms=[]
r2=[]
cv=[]

for i in range(87,90):
        xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=i)
        rf.fit(xtrain,ytrain)
        tr=rf.score(xtrain,ytrain)
        te=rf.score(xtest,ytest)
        pre=rf.predict(xtest)
        mar=mean_absolute_error(ytest,pre)
        mse=np.sqrt(mean_squared_error(ytest,pre))
        CV=cross_val_score(rf,x,y,cv=4).mean()*100
        r=r2_score(ytest,pre)
        I.append(i)
        score.append(te)
        ma.append(mar)
        ms.append(mse)
        r2.append(r)
        cv.append(CV)
values=pd.DataFrame({'accuracy':score,'r2_score':r2,'r_state':I,'error':ms,'CV_score':cv})
values.sort_values(by='accuracy', ascending=False, inplace=True,ignore_index=True)
print('at random state',round(values.loc[0][2],0),'the model having the highest accuracy of',round(values.loc[0][0]*100,2))
print('Mean Square Error Value',values.loc[0][3],'CV_Score',values.loc[0][4])
```

model validation

In [91]:

```
s=pd.read_csv('fsub.csv')
sub=pd.DataFrame(s)
```

In [137…

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.5,random_state=88)
rf.fit(xtrain,ytrain)
tr=rf.score(xtrain,ytrain)
pre=rf.predict(xtest)
mar=mean_absolute_error(ytest,pre)
mse=np.sqrt(mean_squared_error(ytest,pre))
CV=cross_val_score(rf,x,y,cv=4).mean()*100
print('accuracy score {0} mean absolute error {1} and cv score of {2}'.format(tr,mar,CV))
```

```
accuracy score 0.9951292677870024 mean absolute error 925.3490265818045 and cv score of 75.77170651106935
```

The model is 99.5% efficient in predicting the flight ticket price.

Model Saving:

The model can be saved by importing the pickle library

In [161…

```
### Model saving
import pickle
filename='flightprice.pkl'
pickle.dump(rf,open(filename,'wb'))
```