The Blog

Flight Price Prediction

By Pallavi Sinha

<u>Contents</u>

The Packages

The libraries which are used in this project are as follows:

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

1. The NumPy Library :

NumPy(Numerical Python) is an open source Python library that's used in almost every field of science and engineering. It's the universal standard for working with numerical data in Python, and it's at the core of the scientific Python and PyData ecosystems. NumPy users include everyone from beginning coders to experienced researchers doing state-of-the-art scientific and industrial research and development. The NumPy Api is used extensively in Pandas, SciPy, Matplotpltlib, scikit-learn, scikit-image , and most other data science and scientific Python packages.

The NumPy library contains multidimensional array and matrix data structures (you'll find more information about this in later sections). It provides ndarray, a homogeneous n-dimensional array object, with methods to efficiently operate on it. NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

2. The Pandas Library :

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in

Python. Additionally, it has the nroader goal of becoming the most powerful and flexible open source data analysis/manipulation tool available in any language.
Pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet.
- Ordered and Unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data(homogeneously typed or heterogeneous) with row and columns labels.
- Any other form of observational / statistical data sets. The data need not be labeled at all to be placed into a pandas data structure

Pandas will help you to explore, clean, and process your data. In pandas, a data table is called a DataFrame.

The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, DataFrame provides everything that R's data.frame provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Here are just a few of the things that pandas does well:

- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data.

- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects.

- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, DataFrame, etc. automatically align the data for you in computations.

- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data.

- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects.

- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets

- Intuitive merging and joining data sets

- Flexible reshaping and pivoting of data sets

- Hierarchical labeling of axes (possible to have multiple labels per tick)

- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast HDF5 format

- Time series-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting, and lagging.

Many of these principles are here to address the shortcomings frequently experienced using other languages / scientific research environments. For data scientists, working with data is typically divided into multiple stages: munging and cleaning data, analyzing / modeling it, then organizing the results of the analysis into a form suitable for plotting or tabular display. pandas is the ideal tool for all of these tasks.

Pandas is fast. Many of the low-level algorithmic bits have been extensively tweaked in Cython code. However, as with anything else generalization usually sacrifices performance. So if you focus on one feature for your application you may be able to create a faster specialized tool.

Pandas is a dependency of statsmodels, making it an important part of the statistical computing ecosystem in Python. Pandas has been used extensively in production in financial applications.

3. The Matplotlib Library:

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

A Python matplotlib script is structured so that a few lines of code are all that is required in most instances to generate a visual data plot. The matplotlib scripting layer overlays two APIs:

- The pyplot API is a hierarchy of Python code objects topped by *matplotlib.pyplot*

- An OO (Object-Oriented) API collection of objects that can be assembled with greater flexibility than pyplot. This API provides direct access to Matplotlib's backend layers.

Matplotlib and Numpy - Numpy is a package for scientific computing. Numpy is a required dependency for matplotlib, which uses numpy functions for numerical data and multi-dimensional arrays.

Matplotlib and Pandas - Pandas is a library used by matplotlib mainly for data manipulation and analysis. Pandas provides an in-memory 2D data table object called a Dataframe. Unlike numpy, pandas is not a required dependency of matplotlib.

EDA – Exploratory Data Analysis

- EDA is applied to investigate the data and summarize the key insights.
- It will give you the basic understanding of your data, it's distribution, null values and much more.
- You can either explore data using graphs or through some python functions.
- There will be two type of analysis. Univariate and Bivariate. In the univariate, you will be analyzing a single attribute. But in the bivariate, you will be analyzing an attribute with the target attribute.
- In the non-graphical approach, you will be using functions such as shape, summary, describe, isnull, info, datatypes and more.
- In the graphical approach, you will be using plots such as scatter, box, bar, density and correlation plots.

1. Basic information about data – EDA

The df.info() function will give us the basic information about the dataset. For any data, it is good to start by knowing its information. Let's see how it works with our data.

```
In [37]: train_data.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
```

2. Find the Null values

Finding the null values is the most important step in the EDA. As I told many a time, ensuring the quality of data is paramount. So, let's see how we can find the null values.

```
In [39]: train_data.shape
Out[39]: (10683, 11)

In [40]: train_data.dropna(inplace=True) #dropping nan values

In [41]: train_data.shape
Out[41]: (10682, 11)

In [42]: train_data.isnull().sum() #summing up tha no. of null values
Out[42]: Airline            0
         Date_of_Journey    0
         Source             0
         Destination        0
         Route              0
         Dep_Time           0
         Arrival_Time       0
         Duration           0
         Total_Stops        0
         Additional_Info    0
         Price              0
         dtype: int64
```

3. Know the datatypes

Knowing the datatypes which you are exploring is very important and an easy process too. Let's see how it works.

```
In [37]: train_data.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
```

Data Preprocessing

Handling Time & Dates

1. pandas.to_datetime

pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, format=None, exact=True, unit=None, infer_datetime_format=False, origin='unix', cache=True)[source]
Convert argument to datetime.

This function converts a scalar, array-like, Series or DataFrame/dict-like to a pandas datetime object.

Parameters : arg : int, float, str, datetime, list, tuple, 1-d array, Series, DataFrame/dict-like

The object to convert to a datetime. If a DataFrame is provided, the method expects minimally the following columns: "year", "month", "day".

In [43]:
```
train_data['Journey_day'] = pd.to_datetime(train_data.Date_of_Journey, format='%d/%m/%Y').dt.day
```

In [44]:
```
train_data['Journey_month'] = pd.to_datetime(train_data['Date_of_Journey'] , format="%d/%m/%Y").dt.month
```

In [ ]:
```
train_data.drop(["Date_of_Journey"], axis = 1, inplace = True) #dropping dat of journey
```

In [48]:
```
train_data.head()
```

Out[48]:

	Airline	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month
0	IndiGo	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662	1	5
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882	9	6
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218	12	5
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302	1	3

In [53]:
```
#Similar to the date_of_journey we can extract values from dep_time

#extracting hours
train_data["Dep_hour"] = pd.to_datetime(train_data["Dep_Time"]).dt.hour

#extracting minutes
train_data["Dep_min"] = pd.to_datetime(train_data["Dep_Time"]).dt.minute

#now we can drop dep_time as it is of no use
train_data.drop(["Dep_Time"], axis=1,inplace=True)
```

In [54]:
```
train_data.head()
```

Out[54]:

	Airline	Source	Destination	Route	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour	Dep_min
0	IndiGo	Banglore	New Delhi	BLR → DEL	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3	22	20
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	13:15	7h 25m	2 stops	No info	7662	1	5	5	50
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	04:25 10 Jun	19h	2 stops	No info	13882	9	6	9	25
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	23:30	5h 25m	1 stop	No info	6218	12	5	18	5
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	21:35	4h 45m	1 stop	No info	13302	1	3	16	50

In [62]:
```
train_data['Arrival_hour'] = pd.to_datetime(train_data["Arrival_Time"]).dt.hour
train_data['Arrival_min'] = pd.to_datetime(train_data.Arrival_Time).dt.minute

#dropping Arrival_time no use
train_data.drop(["Arrival_Time"],axis=1,inplace=True)
```


In [62]:
```
train_data['Arrival_hour'] = pd.to_datetime(train_data["Arrival_Time"]).dt.hour
train_data['Arrival_min'] = pd.to_datetime(train_data.Arrival_Time).dt.minute

#dropping Arrival_time no use
train_data.drop(["Arrival_Time"],axis=1,inplace=True)
```


In [65]:
```
train_data.head()
```

Out[65]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_min	Arrival_
0	IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	24	3	22	20	10	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	7h 25m	2 stops	No info	7662	1	5	5	50	15	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	19h	2 stops	No info	13882	9	6	9	25	25	
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	5h 25m	1 stop	No info	6218	12	5	18	5	30	
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	4h 45m	1 stop	No info	13302	1	3	16	50	35	

Dynamic Changes

In the above data, “duration” attribute needs to be changed into hour and minutes column. For this we must extract the data by dynamic programming.

```
In [75]: duration=list(train_data.Duration)

         for i in range(len(duration)):
             if len(duration[i].split()) !=2: #check if duration contains only hour or mins
                 if "h" in duration[i]:
                     duration[i]=duration[i].strip()+" 0m"  #adds 0 min
                 else :
                     duration[i]="0h "+duration[i]          #adds 0 hour
         duration_hour = []
         duration_min = []
         for i in range(len(duration)):
             duration_hour.append(int(duration[i].split(sep="h")[0])) #Extracts hours from duration
             duration_min.append(int(duration[i].split(sep="m")[0].split()[-1])) #extracts only min from duration

In [76]: #adding duration_hour list and durtion_min list to train_data dataframe
         train_data["Duration_hours"] = duration_hour
         train_data["Duration_mins"] = duration_min

In [78]: train_data.drop(["Duration"],axis=1,inplace=True)
```

After extracting the data, “duration” attribute is of no use so drop it.

Handling Categorical Data

1. **Identifying Categorical Data: Nominal, Ordinal and Continuous**

Categorical features can only take on a limited, and usually fixed, number of possible values. For example, if a dataset is about information related to users, then you will typically find features like country, gender, age group, etc. Alternatively, if the data you're working with is related to products, you will find features like product type, manufacturer, seller and so on. These are all categorical features in your dataset. These features are typically stored as text values which represent various traits of the observations. For example, gender is described as Male (M) or Female (F), product type could be described as electronics, apparels, food etc. Note that these type of features where the categories are only labeled without any order of precedence are called nominal features. Features which have some order associated with them are called ordinal features. For example, a feature like economic status, with three categories: low, medium and high, which have an order associated with them. There are also continuous features. These are numeric variables that have an infinite number of values between any two values. A continuous variable can be numeric or a date/time. Regardless of what the value is used for, the challenge is determining how to use this data in the analysis because of the following constraints:

Categorical features may have a very large number of levels, known as high cardinality, (for example, cities or URLs), where most of the levels appear in a relatively small number of instances.

Many machine learning models, such as regression or SVM, are algebraic. This means that their input must be numerical. To use these models, categories must be transformed into numbers first, before you can apply the learning algorithm on them.

While some ML packages or libraries might transform categorical data to numeric automatically based on some default embedding method, many other ML packages don't support such inputs.

For the machine, categorical data doesn't contain the same context or information that humans can easily associate and understand. For example, when looking at a feature called City with three cities New York, New Jersey and New Delhi, humans can infer that New York is closely related to New Jersey as they are from same country, while New York and New Delhi are much different. But for the model, New York, New Jersey and New Delhi, are just three different levels (possible values) of the same feature City. If you don't specify the additional contextual information, it will be impossible for the model to differentiate between highly different levels.

You therefore are faced with the challenge of figuring out how to turn these text values into numerical values for further processing and unmask lots of interesting information which these features might hide. Typically, any standard work-flow in feature engineering involves some form of transformation of these categorical values into numeric labels and then applying some encoding scheme on these values.

1. “Airline” is nominal categorical data so OneHotEncoding will be performed.

In [83]:
```
#As Airline is nominal categorical data we will perform OneHotEncoding
Airline = train_data[["Airline"]]
Airline = pd.get_dummies(Airline,drop_first=True)
Airline.head()
```

Out[83]:

	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Jet Airways Business	Airline_Multiple carriers	Airline_Multiple carriers Premium economy	Airline_SpiceJet	Airline_Trujet	Airline_Vistara	Airline_Vista Premiu econo
0	0	0	1	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	
2	0	0	0	1	0	0	0	0	0	0	
3	0	0	1	0	0	0	0	0	0	0	
4	0	0	1	0	0	0	0	0	0	0	

2. “Source” is nominal categorical data so OneHotEncoding will be performed.

In [90]:
```
#Source is nominal data therefore OneHotEncoding
Source = train_data[["Source"]]
Source = pd.get_dummies(Source,drop_first=True)
Source.head()
```

Out[90]:

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	0

3. "Destination" is nominal categorical therefore we will perform OneHotEncoding with the help of get_dummies() method from pandas library.

In [91]:
```
#Destination is nominal data therefore OneHotEncoding
Destination = train_data[["Destination"]]
Destination = pd.get_dummies(Destination,drop_first=True)
Destination.head()
```

Out[91]:

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destination_New Delhi
0	0	0	0	0	1
1	0	0	0	0	0
2	1	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	1

4. "Total_Stops" is ordinal categorical data therefore we will perform LabelEncoder, in this case we will assign the values of "Total_Stops" column with corresponding keys.

In [94]:
```
train_data["Total_Stops"].value_counts()
```

Out[94]:
```
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

In [105]:
```
#As this case of Ordinal Categorical type we perform LabelEncoder
#Here Values are assigned with corresponding keys

train_data = train_data.replace({"non-stop":0,"1 stop":1,"2 stops":2,"3 stops":3,"4 stops":4})
```

As all the categorical data is now encoded, we can concat the respective values to our dataset (train_data) .

In [106]:
```
#concat dataframe -> train_data + Airline + Source + Destination
data_train = pd.concat([train_data,Airline,Source,Destination],axis=1)
```

In [107]:
```
data_train.head()
```

Out[107]:

Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Jet Airways Business	Airline_Multiple carriers	Airline_Multiple carriers Premium economy	Airline_SpiceJet	Airline_Trujet	Airline_Vistara	Airline_Vistara Premium economy
0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0

In [108]:
```
data_train.drop(["Airline","Source","Destination"],axis=1,inplace=True)
```

```
In [106]: #concat dataframe -> train_data + Airline + Source + Destination
          data_train = pd.concat([train_data,Airline,Source,Destination],axis=1)

In [107]: data_train.head()
Out[107]:
```

rce_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destination_New Delhi
0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1

Also, as the old "Airline" , "Source" and "Destination" columns are no more needed, we will drop it.

After the data preprocessing, our dataset i.e. train_data is ready to be used and appear as follows:

```
In [44]: data_train.head()
Out[44]:
```

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	Airline_Air India	Airline_GoAir
0	0	3897	24	3	22	20	1	10	2	50	0	0
1	2	7662	1	5	5	50	13	15	7	25	1	0
2	2	13882	9	6	9	25	4	25	19	0	0	0
3	1	6218	12	5	18	5	23	30	5	25	0	0
4	1	13302	1	3	16	50	21	35	4	45	0	0

```
In [44]: data_train.head()
Out[44]:
```

Airline_IndiGo	Airline_Jet Airways	Airline_Jet Airways Business	Airline_Multiple carriers	Airline_Multiple carriers Premium economy	Airline_SpiceJet	Airline_Trujet	Airline_Vistara	Airline_Vistara Premium economy	Source_Chennai	Source_D
1	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	
0	1	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	

```
In [44]: data_train.head()
```

Out[44]:

rce_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destination_New Delhi
0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1

All the above data-preprocessing methods will now be applied to the test_set in the very same way .

Test Set Operations

```
In [116]: test_data = pd.read_excel(r"C:\Users\Asus\Downloads\Datasets\Flight_Ticket_Participant_Datasets\Test_set.xlsx")
test_data
```

Out[116]:

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info
0	Jet Airways	6/06/2019	Delhi	Cochin	DEL → BOM → COK	17:30	04:25 07 Jun	10h 55m	1 stop	No info
1	IndiGo	12/05/2019	Kolkata	Banglore	CCU → MAA → BLR	06:20	10:20	4h	1 stop	No info
2	Jet Airways	21/05/2019	Delhi	Cochin	DEL → BOM → COK	19:15	19:00 22 May	23h 45m	1 stop	In-flight meal not included
3	Multiple carriers	21/05/2019	Delhi	Cochin	DEL → BOM → COK	08:00	21:00	13h	1 stop	No info
4	Air Asia	24/06/2019	Banglore	Delhi	BLR → DEL	23:55	02:45 25 Jun	2h 50m	non-stop	No info
...	...	...	...	...	...	...	...	...	...	...
2666	Air India	6/06/2019	Kolkata	Banglore	CCU → DEL → BLR	20:30	20:25 07 Jun	23h 55m	1 stop	No info
2667	IndiGo	27/03/2019	Kolkata	Banglore	CCU → BLR	14:20	16:55	2h 35m	non-stop	No info
2668	Jet Airways	6/03/2019	Delhi	Cochin	DEL → BOM → COK	21:50	04:25 07 Mar	6h 35m	1 stop	No info
2669	Air India	6/03/2019	Delhi	Cochin	DEL → BOM → COK	04:00	19:15	15h 15m	1 stop	No info
2670	Multiple carriers	15/06/2019	Delhi	Cochin	DEL → BOM → COK	04:55	19:15	14h 20m	1 stop	No info

2671 rows × 10 columns

In [47]:
```python
test_data['Journey_day'] = pd.to_datetime(test_data.Date_of_Journey, format='%d/%m/%Y').dt.day
test_data['Journey_month'] = pd.to_datetime(test_data['Date_of_Journey'] , format="%d/%m/%Y").dt.month
test_data.drop(["Date_of_Journey"], axis = 1, inplace = True) #dropping dat of journey

#Similar to the date_of_journey we can extract values from dep_time

#extracting hours
test_data["Dep_hour"] = pd.to_datetime(test_data["Dep_Time"]).dt.hour

#extracting minutes
test_data["Dep_min"] = pd.to_datetime(test_data["Dep_Time"]).dt.minute

#now we can drop dep_time as it is of no use
test_data.drop(["Dep_Time"], axis=1,inplace=True)

test_data['Arrival_hour'] = pd.to_datetime(test_data["Arrival_Time"]).dt.hour
test_data['Arrival_min'] = pd.to_datetime(test_data.Arrival_Time).dt.minute

#dropping Arrival_time no use
test_data.drop(["Arrival_Time"],axis=1,inplace=True)

test_data.head()
```

Out[47]:

	Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min
0	Jet Airways	Delhi	Cochin	DEL → BOM → COK	10h 55m	1 stop	No info	6	6	17	30	4	25
1	IndiGo	Kolkata	Banglore	CCU → MAA → BLR	4h	1 stop	No info	12	5	6	20	10	20
2	Jet Airways	Delhi	Cochin	DEL → BOM → COK	23h 45m	1 stop	In-flight meal not included	21	5	19	15	19	0
3	Multiple carriers	Delhi	Cochin	DEL → BOM	13h	1 stop	No info	21	5	8	0	21	0

In [119]:
```python
duration=list(test_data.Duration)

for i in range(len(duration)):
    if len(duration[i].split()) !=2: #check if duration contains only hour or mins
        if "h" in duration[i]:
            duration[i]=duration[i].strip()+" 0m"   #adds 0 min
        else :
            duration[i]="0h "+duration[i]          #adds 0 hour
duration_hour = []
duration_min = []
for i in range(len(duration)):
    duration_hour.append(int(duration[i].split(sep="h")[0])) #Extracts hours from duration
    duration_min.append(int(duration[i].split(sep="m")[0].split()[-1])) #extracts only min from duration
#adding duration_hour list and durtion_min list to train_data dataframe
test_data["Duration_hours"] = duration_hour
test_data["Duration_mins"] = duration_min
```

In [131]:
```python
#As Airline is nominal categorical data we will perform OneHotEncoding
Airline = test_data[["Airline"]]
Airline = pd.get_dummies(Airline,drop_first=True)
Airline.head()

#Source is nominal data therefore OneHotEncoding
Source = test_data[["Source"]]
Source = pd.get_dummies(Source,drop_first=True)
Source.head()

#Destination is nominal data therefore OneHotEncoding
Destination = test_data[["Destination"]]
Destination = pd.get_dummies(Destination,drop_first=True)
Destination.head()

#As Total_stops is of Ordinal Categorical type we perform LabelEncoder
#Here Values are assigned with corresponding keys

test_data = test_data.replace({"non-stop":0,"1 stop":1,"2 stops":2,"3 stops":3,"4 stops":4})
#concat dataframe -> test_data + Airline + Source + Destination
data_test = pd.concat([test_data,Airline,Source,Destination],axis=1)
```

After all the data preprocessing methods, the test_set will have something look like following, with 28 columns/attributes.

```
In [133]: data_test.drop(["Duration"],axis=1,inplace=True)

In [135]: data_test.drop(["Airline","Source","Destination"],axis=1,inplace=True)

In [137]: data_test.shape
Out[137]: (2671, 28)
```

```
In [57]: data_test.head()
Out[57]:
```

Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways
6	6	17	30	4	25	10	55	0	0	0	1
12	5	6	20	10	20	4	0	0	0	1	0
21	5	19	15	19	0	23	45	0	0	0	1
21	5	8	0	21	0	13	0	0	0	0	0
24	6	23	55	2	45	2	50	0	0	0	0

```
In [57]: data_test.head()
Out[57]:
```

Airline_Jet Airways Business	Airline_Multiple carriers	Airline_Multiple carriers Premium economy	Airline_SpiceJet	Airline_Vistara	Airline_Vistara Premium economy	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai	De
0	0	0	0	0	0	0	1	0	0	
0	0	0	0	0	0	0	0	1	0	
0	0	0	0	0	0	0	1	0	0	
0	1	0	0	0	0	0	1	0	0	
0	0	0	0	0	0	0	0	0	0	

In [57]: `data_test.head()`

Out[57]:

rce_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destination_New Delhi
0	1	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0
0	1	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0

Feature Selection

Feature selection is a method of filtering out the important features as all the features present in the dataset are not equally important. There are some features that have no effect on the output. So we can skip them. As our motive is to reduce the data before feeding it to the training model. So feature selection is performed before training machine learning models. The models with less number of features have higher explainability, it is easier to implement machine learning models with reduced features. Feature selection removes data redundancy. Feature selection is performed after feature engineering.

Feature Selection : Finding out the best feature which will contribute and have good relation with target variable. Following are some of the feature selection methods –

1. heatmap
2. feature_importance_
3. SelectKBest

In the following code we will see how we take all independent variables except the dependent variable "Price"

In [142]:
```
data_train.columns
```

Out[142]:
```
Index(['Total_Stops', 'Price', 'Journey_day', 'Journey_month', 'Dep_hour',
       'Dep_min', 'Arrival_min', 'Arrival_hour', 'Duration_hours',
       'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
       'Airline_Jet Airways', 'Airline_Jet Airways Business',
       'Airline_Multiple carriers',
       'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
       'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
       'Destination_Kolkata', 'Destination_New Delhi'],
      dtype='object')
```

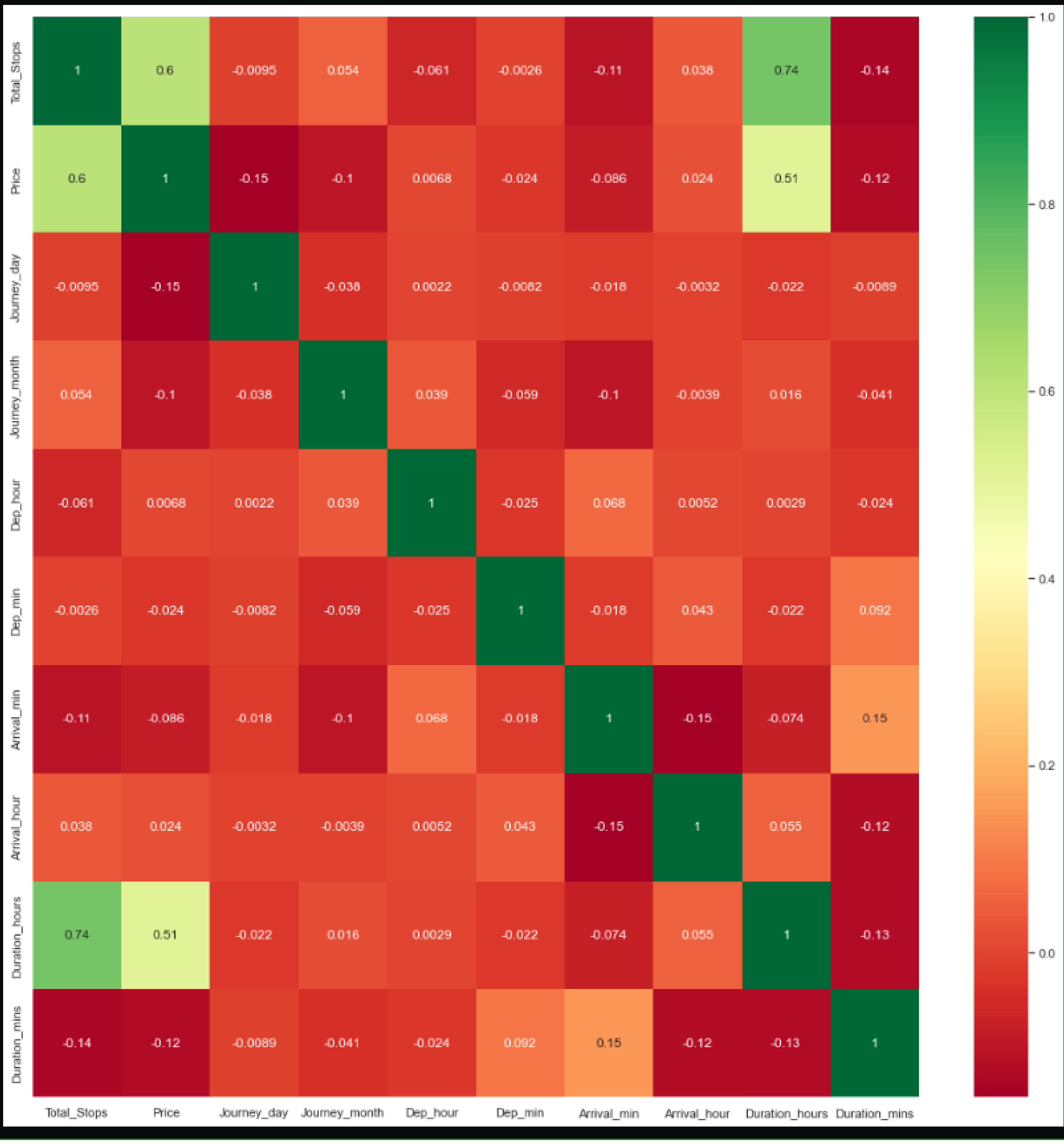
In [144]:
```
#taking all columns except the dependent var Price
X = data_train.loc[:,['Total_Stops', 'Journey_day', 'Journey_month', 'Dep_hour',
       'Dep_min', 'Arrival_min', 'Arrival_hour', 'Duration_hours',
       'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
       'Airline_Jet Airways', 'Airline_Jet Airways Business',
       'Airline_Multiple carriers',
       'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
       'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
       'Destination_Kolkata', 'Destination_New Delhi']]
X.head()
```

Out[144]:

	Total_Stops	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_min	Arrival_hour	Duration_hours	Duration_mins	Airline_Air India	Airline_GoAir	Airline_
0	0	24	3	22	20	10	1	2	50	0	0	
1	2	1	5	5	50	15	13	7	25	1	0	
2	2	9	6	9	25	25	4	19	0	0	0	
3	1	12	5	18	5	30	23	5	25	0	0	
4	1	1	3	16	50	35	21	4	45	0	0	

In [147]:

```python
#Finds correlation between Independent and Dependent attributes
plt.figure (figsize=(18,18))
sns.heatmap(train_data.corr(),annot=True,cmap="RdYlGn")
plt.show()
```

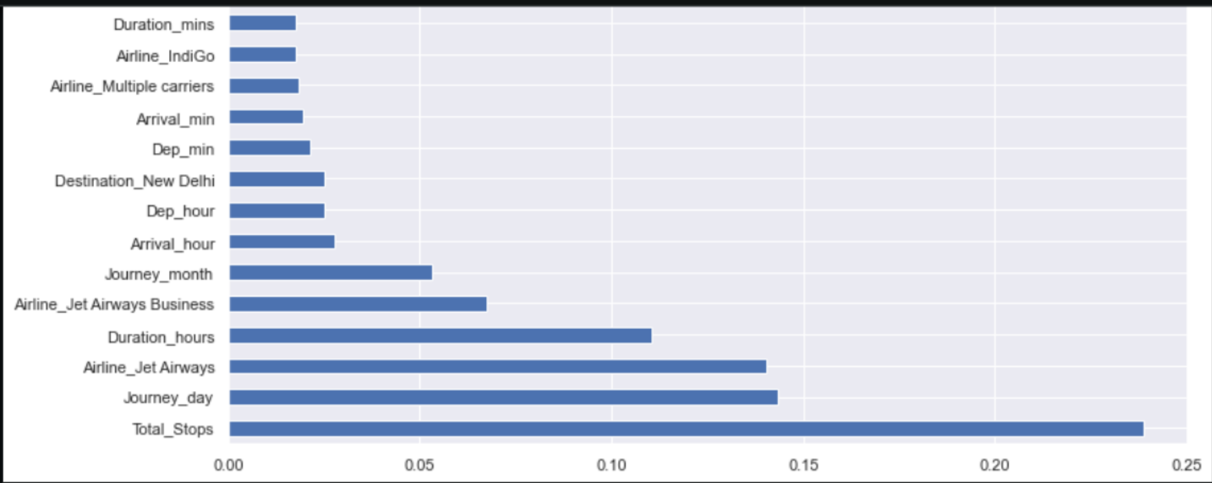


Feature Selection Scores :

```
In [157]: #Important feature using ExtraTreesRegressor
          from sklearn.ensemble import ExtraTreesRegressor
          selection = ExtraTreesRegressor()
          selection.fit(X,y)

          ExtraTreesRegressor()

In [156]: print(selection.feature_importances_)

          [2.22211967e-01 1.44730327e-01 5.30447942e-02 2.44517114e-02
           2.13446981e-02 1.94142004e-02 2.76557166e-02 1.14956862e-01
           1.73121551e-02 9.32436960e-03 1.65682276e-03 1.56006839e-02
           1.47652893e-01 6.74203974e-02 2.09898907e-02 9.17275572e-04
           2.68509228e-03 1.41691318e-04 5.11211340e-03 8.05575314e-05
           4.30035713e-04 1.13581867e-02 3.27321386e-03 7.87880841e-03
           8.04346528e-03 2.01232745e-02 6.79499000e-03 5.07283934e-04
           2.48865225e-02]
```

For a better visualization , we can plot a graph with respect to scores:

```
In [158]: #plot graphof feature importances for better visualization
          plt.figure(figsize=(12,8))
          feat_importances = pd.Series(selection.feature_importances_,index=X.columns)
          feat_importances.nlargest(20).plot(kind='barh')
          plt.show()
```



Prediction Scores

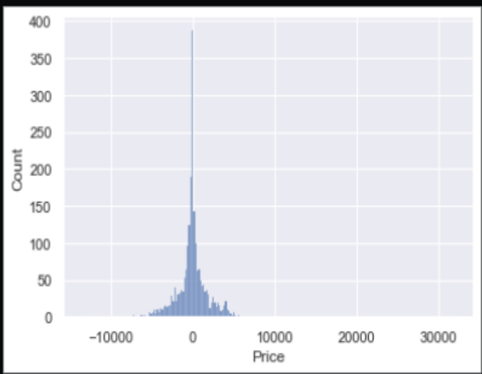
Fitting Model using Random Forest:

- Split dataset into train and test set in order to prediction wrt X_test
- If needed do scaling of data -> Scaling is not done in Random Forest
- Import Model
- Fit the data
- Predict with respect to X_test
- In Regression check RSME score
- Plot graph

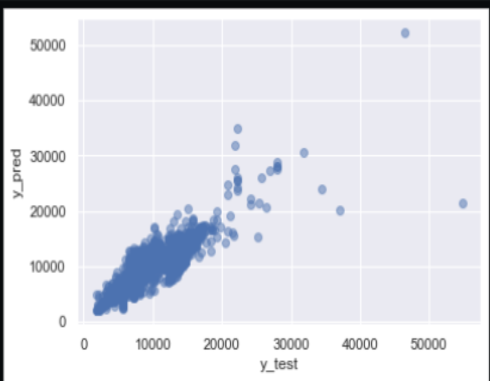
```
In [160]: from sklearn.model_selection import train_test_split
          X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state = 1)

In [161]: from sklearn.ensemble import RandomForestRegressor
          reg_rf = RandomForestRegressor()
          reg_rf.fit(X_train,y_train)

Out[161]: RandomForestRegressor()

In [162]: y_pred = reg_rf.predict(X_test)

In [163]: reg_rf.score(X_train,y_train)

Out[163]: 0.9541723122391247

In [164]: reg_rf.score(X_test,y_test)

Out[164]: 0.8068784353372984
```

Data visualization with help of Seaborn library function -> histplot()

```
In [72]: sns.histplot(y_test-y_pred)
         plt.show()
```




```
In [168]: plt.scatter(y_test,y_pred,alpha=0.5)
          plt.xlabel("y_test")
          plt.ylabel("y_pred")
          plt.show()
```



Conclusion

This project helped me learning the libraries and the respective functions they offer in order to build a machine learning model.

Thank You.