Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Collect The Dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https://www.kaggle.com/code/anshigupta01/flight-price-prediction/data

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Importing The Libraries

Import the necessary libraries as shown in the image. (optional) Here we have used the visualization style as FiveThirtyEight.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoosti
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix
import warnings
import pickle
from scipy import stats
warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')
```

Read The Dataset

Our dataset format might be in .csv, excel files,.txt,.json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of csv file.

```python
data=pd.read_csv("Data_Train.csv")
```

```python
data.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302

Data Preparation

As we have understood how the data is let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much of randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

We have 1 missing value in Route column, and 1 missing value in Total stops column. We will meaningfully replace the missing values going further.

We now start exploring the columns available in our dataset. The first thing we do is to create a list of categorical columns, and check the unique values present in these columns.

```
for i in category:
    print(i, data[i].unique())
```

```
Airline ['IndiGo' 'Air India' 'Jet Airways' 'SpiceJet' 'Multiple carriers' 'GoAir'
 'Vistara' 'Air Asia' 'Vistara Premium economy' 'Jet Airways Business'
 'Multiple carriers Premium economy' 'Trujet']
Source ['Banglore' 'Kolkata' 'Delhi' 'Chennai' 'Mumbai']
Destination ['New Delhi' 'Banglore' 'Cochin' 'Kolkata' 'Delhi' 'Hyderabad']
Additional_Info ['No info' 'In-flight meal not included' 'No check-in baggage included'
 '1 Short layover' 'No Info' '1 Long layover' 'Change airports'
 'Business class' 'Red-eye flight' '2 Long layover']
```

- 1. Airline column has 12 unique values - 'IndiGo' , 'Air India', 'Jet Airways' , 'SpiceJet' , 'Multiple carriers' , 'GoAir', 'Vistara', 'Air Asia', 'Vistara Premium economy' , 'Jet Airways Business', 'Multiple carriers Premium economy', 'Trujet'.
  2. Source column has 5 unique values – ‘Bangalore’, ‘Kolkata’, ‘Chennai’, ‘Delhi’ and ‘Mumbai’.

3. Destination column has 6 unique values - 'New Delhi', 'Banglore', 'Cochin', 'Kolkata', 'Delhi' , 'Hyderabad'.
4. Additional info column has 10 unique values - 'No info', 'In-flight meal not included', 'No check-in baggage included', '1 Short layover' , 'No Info', '1 Long layover', 'Change airports' , 'Business class', 'Red-eye flight' , '2 Long layover'.

We now split the Date column to extract the ‘Date’, ‘Month’ and ‘Year’ values, and store them in new columns in our dataframe.

```
#We now split the Date column to extract the 'Date', 'Month' and 'Year' values, and store them in new columns in our dataframe.

data.Date_of_Journey=data.Date_of_Journey.str.split('/')
```

```
data.Date_of_Journey
```

```
0        [24, 03, 2019]
1         [1, 05, 2019]
2         [9, 06, 2019]
3        [12, 05, 2019]
4        [01, 03, 2019]
              ...
10678     [9, 04, 2019]
10679    [27, 04, 2019]
10680    [27, 04, 2019]
10681    [01, 03, 2019]
10682     [9, 05, 2019]
Name: Date_of_Journey, Length: 10682, dtype: object
```

```
#Treating the data_column


data['Date']=data.Date_of_Journey.str[0]
data['Month']=data.Date_of_Journey.str[1]
data['Year']=data.Date_of_Journey.str[2]
```

- Further, we split the Route column to create multiple columns with cities that the flight travels through. We check the maximum number of stops that a flight has, to confirm what should be the maximum number of cities in the longest route.

```
data.Total_Stops.unique()
```

```
array(['non-stop', '2 stops', '1 stop', '3 stops', '4 stops'],
      dtype=object)
```

-

- Since the maximum number of stops is 4, there should be maximum 6 cities in any particular route. We split the data in route column, and store all the city names in separate columns

```
#Since the maximum number of stops is 4, there should be maximum 6 cities in any particular route. We split the data in route col

data.Route=data.Route.str.split('→')
data.Route
```

```
0                  [BLR ,  DEL]
1      [CCU ,  IXR ,  BBI ,  BLR]
2      [DEL ,  LKO ,  BOM ,  COK]
3             [CCU ,  NAG ,  BLR]
4             [BLR ,  NAG ,  DEL]
                   ...
10678              [CCU ,  BLR]
10679              [CCU ,  BLR]
10680              [BLR ,  DEL]
10681              [BLR ,  DEL]
10682  [DEL ,  GOI ,  BOM ,  COK]
Name: Route, Length: 10682, dtype: object
```

```
data['City1']=data.Route.str[0]
data['City2']=data.Route.str[1]
data['City3']=data.Route.str[2]
data['City4']=data.Route.str[3]
data['City5']=data.Route.str[4]
data['City6']=data.Route.str[5]
```

- In the similar manner, we split the Dep_time column, and create separate columns for departure hours and minutes.

```
#In the similar manner, we split the Dep_time column, and create separate columns for departure hours and minutes -
data.Dep_Time=data.Dep_Time.str.split(':')
```

```
data['Dep_Time_Hour']=data.Dep_Time.str[0]
data['Dep_Time_Mins']=data.Dep_Time.str[1]
```

Further, for the arrival date and arrival time separation, we split the 'Arrival_Time' column, and create 'Arrival_date' column. We also split the time and divide it into ‘Arrival_time_hours’ and ‘Arrival_time_minutes’, similar to what we did with the ‘Dep_time’ column.

- ```
  data.Arrival_Time=data.Arrival_Time.str.split(' ')
  ```

  ```
  data['Arrival_date']=data.Arrival_Time.str[1]
  data['Time_of_Arrival']=data.Arrival_Time.str[0]
  ```

  ```
  data['Time_of_Arrival']=data.Time_of_Arrival.str.split(':')
  ```

  ```
  data['Arrival_Time_Hour']=data.Time_of_Arrival.str[0]
  data['Arrival_Time_Mins']=data.Time_of_Arrival.str[1]
  ```

Next, we divide the ‘Duration’ column to ‘Travel_hours’ and ‘ Travel_mins’

- ```
  #Next, we divide the ‘Duration’ column to ‘Travel_hours’ and ‘ Travel_mins’

  data.Duration=data.Duration.str.split(' ')
  ```

  ```
  data['Travel_Hours']=data.Duration.str[0]
  data['Travel_Hours']=data['Travel_Hours'].str.split('h')
  data['Travel_Hours']=data['Travel_Hours'].str[0]
  data.Travel_Hours=data.Travel_Hours
  data['Travel_Mins']=data.Duration.str[1]

  data.Travel_Mins=data.Travel_Mins.str.split('m')
  data.Travel_Mins=data.Travel_Mins.str[0]
  ```

  ```
  #We also treat the ‘Total_stops’ column, and replace non-stop flights with 0 value and extract the integer part of the ‘Total_Sto
  data.Total_Stops.replace('non_stop',0,inplace=True)
  data.Total_Stops=data.Total_Stops.str.split(' ')
  data.Total_Stops=data.Total_Stops.str[0]
  ```

```
#Next, we divide the 'Duration' column to 'Travel_hours' and ' Travel_mins'

data.Duration=data.Duration.str.split(' ')
```

```
data['Travel_Hours']=data.Duration.str[0]
data['Travel_Hours']=data['Travel_Hours'].str.split('h')
data['Travel_Hours']=data['Travel_Hours'].str[0]
data.Travel_Hours=data.Travel_Hours
data['Travel_Mins']=data.Duration.str[1]

data.Travel_Mins=data.Travel_Mins.str.split('m')
data.Travel_Mins=data.Travel_Mins.str[0]
```

- ```
  #We also treat the 'Total_stops' column, and replace non-stop flights with 0 value and extract the integer part of the 'Total_Sto
  data.Total_Stops.replace('non_stop',0,inplace=True)
  data.Total_Stops=data.Total_Stops.str.split(' ')
  data.Total_Stops=data.Total_Stops.str[0]
  ```

- We also treat the 'Total_stops' column, and replace non-stop flights with 0 value and extract the integer part of the 'Total_Stops' column.

    -

```
#We also treat the 'Total_stops' column, and replace non-stop flights with 0 value and extract the integer part of the 'Total_Sto
data.Total_Stops.replace('non_stop',0,inplace=True)
data.Total_Stops=data.Total_Stops.str.split(' ')
data.Total_Stops=data.Total_Stops.str[0]
```

- We proceed further to the 'Additional_info' column, where we observe that there are 2 categories signifying 'No info', which are divided into 2 categories since 'I' in 'No Info' is capital. We replace 'No Info' by 'No info' to merge it into a single category.

    -
```

```
data.Additional_Info.unique()
```

```
array(['No info', 'In-flight meal not included',
       'No check-in baggage included', '1 Short layover', 'No Info',
       '1 Long layover', 'Change airports', 'Business class',
       'Red-eye flight', '2 Long layover'], dtype=object)
```

- ```
  data.Additional_Info.replace('No Info','No info',inplace=True)
  ```

- We now drop all the columns from which we have extracted the useful information (original columns). We also drop some columns like ‘city4’,'city5' and ‘city6’, since majority of the data in these columns was NaN(null). As a result, we now obtain 20 different columns, which we will be feeding to our ML model. But first, we treat the missing values and explore the contents in the columns and its impact on the flight price, to separate a list of final set of columns.

  -
  ```

```
data.isnull().sum()
```

```
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
City1                0
City2                0
City3             3491
City4             9116
City5            10636
City6            10681
Date                 0
Month                0
Year                 0
Dep_Time_Hour        0
Dep_Time_Mins        0
Arrival_date      6348
Time_of_Arrival      0
Arrival_Time_Hour    0
Arrival_Time_Mins    0
Travel_Hours         0
Travel_Mins       1032
dtype: int64
```

```
#We also drop some columns like ‘city6’ and ‘city5’, since majority of the data in these columns was NaN(null)
data.drop(['City4','City5','City6'],axis=1,inplace=True)
```

```
data.drop(['Date_of_Journey','Route','Dep_Time','Arrival_Time','Duration'],axis=1, inplace=True)
data.drop(['Time_of_Arrival'],axis=1,inplace=True)
```

- After dropping some columns, here we can see the meaningful columns to predict the flight price without the NaN values.

    -

```
#Checking Null Values
data.isnull().sum()
```

```
Airline                 0
Source                  0
Destination             0
Total_Stops             0
Additional_Info         0
Price                   0
City1                   0
City2                   0
City3                3491
Date                    0
Month                   0
Year                    0
Dep_Time_Hour           0
Dep_Time_Mins           0
Arrival_date         6348
Arrival_Time_Hour       0
Arrival_Time_Mins       0
Travel_Hours            0
Travel_Mins          1032
dtype: int64
```

Replacing Missing Values

We further replace ‘NaN’ values in ‘City3’ with ‘None’, since rows where ‘City3’ is missing did not have any stop, just the source and the destination.

We also replace missing values in ‘Arrival_date’ column with values in ‘Date’ column, since the missing values are those values where the flight took off and landed on the same date.

We also replace missing values in ‘Travel_mins’ as 0, since the missing values represent that the travel time was in terms on hours only, and no additional minutes.

```python
#filling City3 as None, the missing values are less
data['City3'].fillna('None',inplace=True)
```

```python
#filling Arrival_Date as Departure_Date
data['Arrival_date'].fillna(data['Date'],inplace=True)
```

```python
#filling Travel_Mins as Zero(0)
data['Travel_Mins'].fillna(0,inplace=True)
```

- Using the above steps, we were successfully able to treat all the missing values from our data. We again check the info in our data and find out that the dataset still has data types for multiple columns as ‘object’, where it should be ‘int’

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 19 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Airline            10682 non-null  object
 1   Source             10682 non-null  object
 2   Destination        10682 non-null  object
 3   Total_Stops        10682 non-null  object
 4   Additional_Info    10682 non-null  object
 5   Price              10682 non-null  int64
 6   City1              10682 non-null  object
 7   City2              10682 non-null  object
 8   City3              10682 non-null  object
 9   Date               10682 non-null  object
 10  Month              10682 non-null  object
 11  Year               10682 non-null  object
 12  Dep_Time_Hour      10682 non-null  object
 13  Dep_Time_Mins      10682 non-null  object
 14  Arrival_date       10682 non-null  object
 15  Arrival_Time_Hour  10682 non-null  object
 16  Arrival_Time_Mins  10682 non-null  object
 17  Travel_Hours       10682 non-null  object
 18  Travel_Mins        10682 non-null  object
dtypes: int64(1), object(18)
memory usage: 1.6+ MB
```

- Hence, we try to change the datatype of the required columns

- ```
  #changing the numerical columns from object to int
  #data.Total_Stops=data.Total_Stops.astype('int64')
  data.Date=data.Date.astype('int64')
  data.Month=data.Month.astype('int64')
  data.Year=data.Year.astype('int64')
  data.Dep_Time_Hour=data.Dep_Time_Hour.astype('int64')
  data.Dep_Time_Hour=data.Dep_Time_Hour.astype('int64')
  data.Dep_Time_Mins=data.Dep_Time_Mins.astype('int64')
  data.Arrival_date=data.Arrival_date.astype("int64")
  data.Arrival_Time_Hour=data.Arrival_Time_Hour.astype('int64')
  data.Arrival_Time_Mins=data.Arrival_Time_Mins.astype('int64')
  #data.Travel_Hours=data.Travel_Hours.astype('int64')
  data.Travel_Mins=data.Travel_Mins.astype('int64')
  ```
- During this step, we face issue converting the 'Travel_hours' column, saying that the column has data as '5m', which is not allowing its conversion to 'int'.

```
data[data['Travel_Hours']=='5m']
```

Price	City1	City2	City3	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins	Travel_Hours	Travel_Mins
17327	BOM	GOI	PNQ	6	3	2019	16	50	6	16	55	5m	0

- The data signifies that the flight time is '5m', which is obviously wrong as the plane cannot fly from BOMBAY->GOA->PUNE->HYDERABAD in 5 mins! (The flight has 'Total_stops' as 2)

```
data.drop(index=6474,inplace=True,axis=0)
```

- We then convert the 'Travel_hours' column to 'int' data type, and the operation happens successfully.
```

We now have a treated dataset with 10682 rows and 17 columns (16 independent and 1 dependent variable).

```
data.Travel_Hours=data.Travel_Hours.astype('int64')
```

```
#Creating list of Different types of columns
categorical=['Airline','Source','Destination','Additional_Info','City1']
numerical=['Total_Stops','Date','Month','Year','Dep_Time_Hour','Dep_Time_Mins','Arrival_date','Arrival_Time_Hour',
           'Arrival_Time_Mins','Travel_Hours','Travel_Mins']
```

We create separate lists of categorical columns and numerical columns for plotting and analyzing the data

Label Encoding

- Label encoding converts the data in machine readable form, but it assigns a unique number (starting from 0) to each class of data. it performs the conversion of categorical data into numeric format.
- In our dataset I have converted these variables 'Airline','Source','Destination','Total_Stops','City1','City2','City3','Additional_Info' into number format. So that it helps the model in better understanding of the dataset and enables the model to learn more complex structures.

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

```
data.Airline=le.fit_transform(data.Airline)
data.Source=le.fit_transform(data.Source)
data.Destination=le.fit_transform(data.Destination)
data.Total_Stops=le.fit_transform(data.Total_Stops)
data.City1	=le.fit_transform(data.City1	)
data.City2=le.fit_transform(data.City2)
data.City3=le.fit_transform(data.City3)
data.Additional_Info=le.fit_transform(data.Additional_Info)
data.head()
```

	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	City2	City3	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arri
0	3	0	5	4	7	3897	0	13	29	24	3	2019	22	20	22	
1	1	3	0	1	7	7662	2	25	1	1	5	2019	5	50	1	
2	4	2	1	1	7	13882	3	32	4	9	6	2019	9	25	10	
3	3	3	0	0	7	6218	2	34	3	12	5	2019	18	5	12	
4	3	0	5	0	7	13302	0	34	8	1	3	2019	16	50	1	

Output Columns

- Initially in our dataset we have 19 features. So, in that some features are not more important to get output (Price).
- So i removed some unrelated features and I selected important features. So, it makes easy to understand. Now we have only 12

Output Columns.

```
data.head()
```

	Airline	Source	Destination	Total_Stops	Additional_Info	Price	City1	City2	City3	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arri
0	3	0	5	4	7	3897	0	13	29	24	3	2019	22	20	22	
1	1	3	0	1	7	7662	2	25	1	1	5	2019	5	50	1	
2	4	2	1	1	7	13882	3	32	4	9	6	2019	9	25	10	
3	3	3	0	0	7	6218	2	34	3	12	5	2019	18	5	12	
4	3	0	5	0	7	13302	0	34	8	1	3	2019	16	50	1	

```
data = data[['Airline','Source','Destination','Date','Month','Year','Dep_Time_Hour','Dep_Time_Mins','Arrival_date','Arrival_Time_
```

```
data.head()
```

	Airline	Source	Destination	Date	Month	Year	Dep_Time_Hour	Dep_Time_Mins	Arrival_date	Arrival_Time_Hour	Arrival_Time_Mins	Price
0	3	0	5	24	3	2019	22	20	22	1	10	3897
1	1	3	0	1	5	2019	5	50	1	13	15	7662
2	4	2	1	9	6	2019	9	25	10	4	25	13882
3	3	3	0	12	5	2019	18	5	12	23	30	6218
4	3	0	5	1	3	2019	16	50	1	21	35	13302