What we often do in this use-case

```
High level over-view..


Note :->>

We will solve most of those challenges that we often face in real
world..
we will focus primarily on each & every part of data science life-
cycle..


 Life- Cycle of Data Science Project :
    a) Data collection
    b) Perform Data Cleaning / Data Preparation / Data Pre-processing
    c) Data visuaslisation(EDA)
    d) Perform feature engineering
        I)  Feature encoding
        II) checking outliers & impute it..
        III)Feature selection or feature importance

    e) build machine leaning model & dump it..
    f) Automate ML Pipeline
    g) hypertune ml model..along with cross validation
```

1.. Lets read data !

```
## import necessary packages !

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

Importing dataset

```
Since data is in form of excel file we have to use pandas read_excel
to load the data

train_data = pd.read_excel('Data_Train.xlsx')
```

```
train_data.head(4)
```

```
       Airline Date_of_Journey    Source Destination
Route  \
0       IndiGo      24/03/2019  Banglore   New Delhi              BLR
→ DEL
1    Air India       1/05/2019   Kolkata    Banglore  CCU → IXR → BBI
→ BLR
2  Jet Airways       9/06/2019     Delhi      Cochin  DEL → LKO → BOM
→ COK
3       IndiGo      12/05/2019   Kolkata    Banglore        CCU → NAG
→ BLR

  Dep_Time  Arrival_Time Duration Total_Stops Additional_Info  Price
0    22:20  01:10 22 Mar   2h 50m    non-stop         No info   3897
1    05:50         13:15   7h 25m     2 stops         No info   7662
2    09:25  04:25 10 Jun      19h     2 stops         No info  13882
3    18:05         23:30   5h 25m      1 stop         No info   6218
```

```
train_data.tail(4)
```

```
           Airline Date_of_Journey    Source Destination  \
10679    Air India      27/04/2019   Kolkata    Banglore
10680  Jet Airways      27/04/2019  Banglore       Delhi
10681      Vistara      01/03/2019  Banglore   New Delhi
10682    Air India       9/05/2019     Delhi      Cochin

                       Route Dep_Time Arrival_Time Duration
Total_Stops  \
10679              CCU → BLR    20:45        23:20   2h 35m    non-
stop
10680              BLR → DEL    08:20        11:20       3h    non-
stop
10681              BLR → DEL    11:30        14:10   2h 40m    non-
stop
10682  DEL → GOI → BOM → COK    10:55        19:15   8h 20m     2
stops

      Additional_Info  Price
10679         No info   4145
10680         No info   7229
10681         No info  12648
10682         No info  11753
```

2.. Lets deal with missing values ..

```
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
```

```
'''

10 features belong to object data-type , ie.. in context to Python ,
they belong to string data-type


1 feature belong to int64 nature  , ie
Variations of int are : ('int64','int32','int16') in numpy library..



Int16 is a 16 bit signed integer , it means it can store both positive
& negative values
int16 has has a range of  (2^15 − 1) to -2^15
int16 has a length of 16 bits (2 bytes).. ie Int16 uses 16 bits to
store data


Int32 is a 32 bit signed integer , it means it storesboth positive &
negative values
int32 has has a range of (2³¹ − 1) to  -2^31
int32 has a length of 32 bits (4 bytes),, ie Int32 uses 32 bits to
store data


Int64 is a 64 bit signed integer , it means it can store both positive
& negative values
int64 has has a range of  (2^63 − 1) to -2^63
```

```
int64 has a length of 64 bits (8 bytes) , ie Int64 uses 64 bits to
store data

The only difference is that int64 has max range of storing numbers ,
then comes int32 , then 16 , then int8

That means that Int64’s take up twice as much memory-and doing
operations on them may be a lot slower in some machine architectures.

However, Int64’s can represent numbers much more accurately than
32 bit floats.They also allow much larger numbers to be stored..







The memory usage of a DataFrame (including the index) is shown when
calling the info().
A configuration option, display.memory_usage (see the list of
options), specifies if the DataFrame’s memory usage
 will be displayed when invoking the df.info() method..

memory usage: 918.2+ KB
The + symbol indicates that the true memory usage could be higher,
because pandas does not count the memory used by values in columns
with dtype=object


Passing memory_usage='deep' will enable a more accurate memory usage
report .

'''
```

```
"\n\n10 features belong to object data-type , ie.. in context to
Python , they belong to string data-type\n\n            \n1 feature
belong to int64 nature  , ie \nVariations of int are :
('int64','int32','int16') in numpy library..\n\n\n\nInt16 is a 16 bit
signed integer , it means it can store both positive & negative
values\nint16 has has a range of  (2^15 − 1) to -2^15 \nint16 has a
length of 16 bits (2 bytes).. ie Int16 uses 16 bits to store data\n\n\
nInt32 is a 32 bit signed integer , it means it storesboth positive &
negative values\nint32 has has a range of (2³¹ − 1) to  -2^31\nint32
has a length of 32 bits (4 bytes),, ie Int32 uses 32 bits to store
data\n\n\nInt64 is a 64 bit signed integer , it means it can store
both positive & negative values\nint64 has has a range of  (2^63 − 1)
to -2^63 \nint64 has a length of 64 bits (8 bytes) , ie Int64 uses 64
bits to store data\n            \nThe only difference is that int64
has max range of storing numbers , then comes int32 , then 16 , then
int8\n\nThat means that Int64’s take up twice as much memory-and doing
```

```
\noperations on them may be a lot slower in some machine
architectures.\n\nHowever, Int64’s can represent numbers much more
accurately than \n32 bit floats.They also allow much larger numbers to
be stored..\n\n\n\n\n\n\n\nThe memory usage of a DataFrame (including
the index) is shown when calling the info(). \nA configuration option,
display.memory_usage (see the list of options), specifies if the
DataFrame’s memory usage \n will be displayed when invoking the
df.info() method..\n \nmemory usage: 918.2+ KB \nThe + symbol
indicates that the true memory usage could be higher, \nbecause pandas
does not count the memory used by values in columns with dtype=object\
n\n\nPassing memory_usage='deep' will enable a more accurate memory
usage report .\n\n"
```

```
## After loading it is important to check null/missing values in a
column or a row
## Missing value :  values which occur when no data is recorded for an
observation..

train_data.isnull().sum()

## train_data.isnull().sum(axis=0)
## by-default axis is 0 , ie it computes total missing values column-
wise !
```

```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              1
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        1
Additional_Info    0
Price              0
dtype: int64
```

```
train_data['Total_Stops'].isnull()
```

```
0        False
1        False
2        False
3        False
4        False
         ...
10678    False
10679    False
10680    False
10681    False
```

```
10682    False
Name: Total_Stops, Length: 10683, dtype: bool
```

```
### getting all the rows where we have missing value

train_data[train_data['Total_Stops'].isnull()]
```

```
        Airline Date_of_Journey Source Destination Route Dep_Time  \
9039  Air India       6/05/2019  Delhi      Cochin   NaN    09:45

      Arrival_Time Duration Total_Stops Additional_Info  Price
9039  09:25 07 May  23h 40m         NaN         No info   7480
```

as we have 1 missing value , I can directly drop these

```
train_data.dropna(inplace=True)

train_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
dtype: int64
```

```
train_data.dtypes
```

```
Airline            object
Date_of_Journey    object
Source             object
Destination        object
Route              object
Dep_Time           object
Arrival_Time       object
Duration           object
Total_Stops        object
Additional_Info    object
Price               int64
dtype: object
```

```
### In order to more accurate memory usage , u can leverage 
memory_usage="deep" in info()
train_data.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10682 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10682 non-null  object
 1   Date_of_Journey  10682 non-null  object
 2   Source           10682 non-null  object
 3   Destination      10682 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10682 non-null  object
 6   Arrival_Time     10682 non-null  object
 7   Duration         10682 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10682 non-null  object
 10  Price            10682 non-null  int64 
dtypes: int64(1), object(10)
memory usage: 7.2 MB
```

3.. Lets Perform Data Pre-process & extract Derived attributes from "Date_of_Journey"

```
    lets extract derived attributes from "Date_of_Journey" & fetch day 
, month , year !
```

```
data = train_data.copy()
```

```
data.columns
```

```
Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',
       'Additional_Info', 'Price'],
      dtype='object')
```

```
data.head(2)
```

```
     Airline Date_of_Journey    Source Destination                  
Route  \
0     IndiGo      24/03/2019  Banglore   New Delhi              BLR → 
DEL
1  Air India       1/05/2019   Kolkata    Banglore  CCU → IXR → BBI → 
BLR

  Dep_Time  Arrival_Time Duration Total_Stops Additional_Info  Price
0    22:20  01:10 22 Mar   2h 50m    non-stop         No info   3897
1    05:50         13:15   7h 25m     2 stops         No info   7662
```

```
data.dtypes
```

```
Airline            object
Date_of_Journey    object
Source             object
Destination        object
Route              object
Dep_Time           object
Arrival_Time       object
Duration           object
Total_Stops        object
Additional_Info    object
Price               int64
dtype: object
```

From description we can see that Date_of_Journey is a object data type,

```
 Therefore, we have to convert this datatype into timestamp so as to 
use this column properly for prediction,bcz our
 model will not be able to understand these string values,it just 
understand Time-stamp
For this we require pandas to_datetime to convert object data type to 
datetime dtype.

'''
In date-time , we have 4 data-types in Pandas :
datetime64[ns] or datetime64[ns, tz]  or datetime64[ns, UTC] or 
dtype('<M8[ns]')
     means ‘big-endian’  , < is little-endian
     imagine , data represented a single unsigned 4-byte little-endian 
integer, the dtype string would be <u4..
     (u is type-character code for unsigned integer)

where ,   UTC = Coordinated Universal Time
          ns  = nano second
          tz  = time zone
          M =  M is a character of Data-time , just like int we have i 
for "Integer" ,
```

```
datetime64[ns] is a general dtype, while <M8[ns] is a specific dtype ,
ns is basicaly nano second..
Both are similar , it entirely how your numpy was compiled..

np.dtype('datetime64[ns]') == np.dtype('<M8[ns]')
## True

'''

'\nIn date-time , we have 4 data-types in Pandas :\ndatetime64[ns] or
datetime64[ns, tz]  or datetime64[ns, UTC] or dtype(\'<M8[ns]\')\n
means ‘big-endian’  , < is little-endian\n     imagine , data
represented a single unsigned 4-byte little-endian integer, the dtype
string would be <u4..\n     (u is type-character code for unsigned
integer)\n     \nwhere ,   UTC = Coordinated Universal Time\n
ns  = nano second\n          tz  = time zone\n          M =  M is a
character of Data-time , just like int we have i for "Integer" ,\n\n\
ndatetime64[ns] is a general dtype, while <M8[ns] is a specific
dtype , ns is basicaly nano second..\nBoth are similar , it entirely
how your numpy was compiled..\n\nnp.dtype(\'datetime64[ns]\') ==
np.dtype(\'<M8[ns]\')\n## True\n\n'



def change_into_Datetime(col):
    data[col] = pd.to_datetime(data[col])

import warnings
from warnings import filterwarnings
filterwarnings("ignore")

data.columns

Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',
       'Additional_Info', 'Price'],
      dtype='object')

for feature in ['Dep_Time', 'Arrival_Time' , 'Date_of_Journey']:
    change_into_Datetime(feature)

data.dtypes

Airline                    object
Date_of_Journey    datetime64[ns]
Source                     object
Destination                object
Route                      object
Dep_Time           datetime64[ns]
Arrival_Time       datetime64[ns]
```

```
Duration                          object
Total_Stops                       object
Additional_Info                   object
Price                              int64
dtype: object
```

```python
data["Journey_day"] = data['Date_of_Journey'].dt.day

data["Journey_month"] = data['Date_of_Journey'].dt.month

data["Journey_year"] = data['Date_of_Journey'].dt.year

data.head(3)
```

```
       Airline Date_of_Journey    Source Destination
Route  \
0       IndiGo      2019-03-24  Banglore   New Delhi              BLR
→ DEL
1    Air India      2019-01-05   Kolkata    Banglore  CCU → IXR → BBI
→ BLR
2  Jet Airways      2019-09-06     Delhi      Cochin  DEL → LKO → BOM
→ COK

             Dep_Time        Arrival_Time Duration Total_Stops  \
0 2024-09-08 22:20:00 2024-03-22 01:10:00   2h 50m    non-stop
1 2024-09-08 05:50:00 2024-09-08 13:15:00   7h 25m     2 stops
2 2024-09-08 09:25:00 2024-06-10 04:25:00      19h     2 stops

  Additional_Info  Price  Journey_day  Journey_month  Journey_year
0         No info   3897           24              3          2019
1         No info   7662            5              1          2019
2         No info  13882            6              9          2019
```

4.. Lets try to clean Dep_Time & Arrival_Time & then extract Derived attributes ..

```python
def extract_hour_min(df , col):
    df[col+"_hour"] = df[col].dt.hour
```

```
    df[col+"_minute"] = df[col].dt.minute
    return df.head(3)

data.columns

Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',
       'Additional_Info', 'Price', 'Journey_day', 'Journey_month',
       'Journey_year'],
      dtype='object')

# Departure time is when a plane leaves the gate.

extract_hour_min(data , "Dep_Time")

       Airline Date_of_Journey    Source Destination
Route  \
0       IndiGo      2019-03-24  Banglore   New Delhi              BLR
→ DEL
1    Air India      2019-01-05   Kolkata    Banglore  CCU → IXR → BBI
→ BLR
2  Jet Airways      2019-09-06     Delhi      Cochin  DEL → LKO → BOM
→ COK

             Dep_Time        Arrival_Time Duration Total_Stops  \
0 2024-09-08 22:20:00 2024-03-22 01:10:00   2h 50m    non-stop
1 2024-09-08 05:50:00 2024-09-08 13:15:00   7h 25m     2 stops
2 2024-09-08 09:25:00 2024-06-10 04:25:00      19h     2 stops

  Additional_Info  Price  Journey_day  Journey_month  Journey_year  \
0         No info   3897           24              3          2019
1         No info   7662            5              1          2019
2         No info  13882            6              9          2019

   Dep_Time_hour  Dep_Time_minute
0             22               20
1              5               50
2              9               25

extract_hour_min(data , "Arrival_Time")

       Airline Date_of_Journey    Source Destination
Route  \
0       IndiGo      2019-03-24  Banglore   New Delhi              BLR
→ DEL
1    Air India      2019-01-05   Kolkata    Banglore  CCU → IXR → BBI
→ BLR
2  Jet Airways      2019-09-06     Delhi      Cochin  DEL → LKO → BOM
→ COK

             Dep_Time        Arrival_Time Duration Total_Stops  \
```

```
0 2024-09-08 22:20:00 2024-03-22 01:10:00   2h 50m    non-stop
1 2024-09-08 05:50:00 2024-09-08 13:15:00   7h 25m     2 stops
2 2024-09-08 09:25:00 2024-06-10 04:25:00      19h     2 stops

  Additional_Info  Price  Journey_day  Journey_month  Journey_year  \
0         No info   3897           24              3          2019
1         No info   7662            5              1          2019
2         No info  13882            6              9          2019

   Dep_Time_hour  Dep_Time_minute  Arrival_Time_hour
Arrival_Time_minute
0             22               20                  1
10
1              5               50                 13
15
2              9               25                  4
25

## we have extracted derived attributes from ['Arrival_Time' ,
"Dep_Time"] , so lets drop both these features ..
cols_to_drop = ['Arrival_Time' , "Dep_Time"]

data.drop(cols_to_drop , axis=1 , inplace=True )

data.head(3)

       Airline Date_of_Journey    Source Destination
Route  \
0       IndiGo      2019-03-24  Banglore   New Delhi              BLR
→ DEL
1    Air India      2019-01-05   Kolkata    Banglore  CCU → IXR → BBI
→ BLR
2  Jet Airways      2019-09-06     Delhi      Cochin  DEL → LKO → BOM
→ COK

  Duration Total_Stops Additional_Info  Price  Journey_day
Journey_month  \
0   2h 50m    non-stop         No info   3897           24
3
1   7h 25m     2 stops         No info   7662            5
1
2      19h     2 stops         No info  13882            6
9

   Journey_year  Dep_Time_hour  Dep_Time_minute  Arrival_Time_hour  \
0          2019             22               20                  1
1          2019              5               50                 13
2          2019              9               25                  4

   Arrival_Time_minute
0                   10
```

```
1                    15
2                    25
```

```
data.shape
```

```
(10682, 16)
```

5.. lets analyse when will most of the flights take-off..

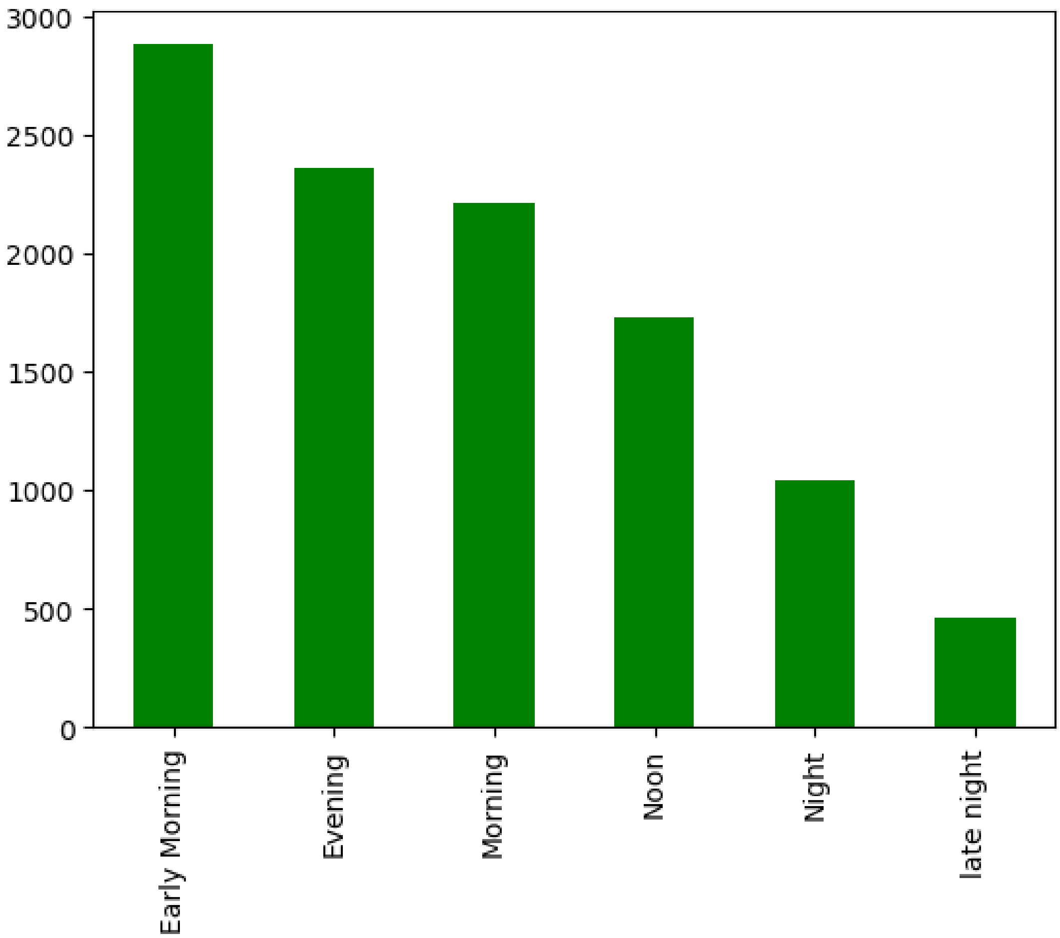
```
data.columns
```

```
Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Duration', 'Total_Stops', 'Additional_Info', 'Price',
'Journey_day',
       'Journey_month', 'Journey_year', 'Dep_Time_hour',
'Dep_Time_minute',
       'Arrival_Time_hour', 'Arrival_Time_minute'],
      dtype='object')
```

```
#### Converting the flight Dep_Time into proper time i.e. mid_night,
morning, afternoon and evening.

def flight_dep_time(x):
    '''
    This function takes the flight Departure time
    and convert into appropriate format.

    '''

    if (x>4) and (x<=8):
        return "Early Morning"

    elif (x>8) and (x<=12):
        return "Morning"

    elif (x>12) and (x<=16):
        return "Noon"

    elif (x>16) and (x<=20):
        return "Evening"

    elif (x>20) and (x<=24):
```

```
        return "Night"

    else:
        return "late night"

data['Dep_Time_hour'].apply(flight_dep_time).value_counts().plot(kind=
"bar" , color="g")
```

```
<AxesSubplot:>
```




```
#### how to make above graph interactive , lets use Cufflinks & plotly
to make it interactive !



##!pip install plotly
##!pip install chart_studio
```

```
Requirement already satisfied: plotly in c:\users\arpit patel\
anaconda3\lib\site-packages (5.11.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\arpit
```

```
patel\anaconda3\lib\site-packages (from plotly) (8.0.1)
Collecting chart_studio
  Downloading chart_studio-1.1.0-py3-none-any.whl (64 kB)
     ---------------------------------------- 64.4/64.4 kB 1.7 MB/s
eta 0:00:00
Requirement already satisfied: plotly in c:\users\arpit patel\
anaconda3\lib\site-packages (from chart_studio) (5.11.0)
Requirement already satisfied: retrying>=1.3.3 in c:\users\arpit
patel\anaconda3\lib\site-packages (from chart_studio) (1.3.4)
Requirement already satisfied: requests in c:\users\arpit patel\
anaconda3\lib\site-packages (from chart_studio) (2.31.0)
Requirement already satisfied: six in c:\users\arpit patel\anaconda3\
lib\site-packages (from chart_studio) (1.16.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\arpit
patel\anaconda3\lib\site-packages (from plotly->chart_studio) (8.0.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\arpit
patel\anaconda3\lib\site-packages (from requests->chart_studio)
(2022.9.14)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\
arpit patel\anaconda3\lib\site-packages (from requests->chart_studio)
(2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\arpit patel\
anaconda3\lib\site-packages (from requests->chart_studio) (3.3)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\arpit
patel\anaconda3\lib\site-packages (from requests->chart_studio)
(1.26.11)
Installing collected packages: chart_studio
Successfully installed chart_studio-1.1.0
```

```python
##!pip install cufflinks
```

```python
## how to use Plotly interactive plots directly with Pandas
dataframes, First u need below set-up !

import plotly
import cufflinks as cf
from cufflinks.offline import go_offline
from plotly.offline import plot , iplot , init_notebook_mode ,
download_plotlyjs
init_notebook_mode(connected=True)
cf.go_offline()

## plot is a command of Matplotlib which is more old-school. It
creates static charts
## iplot is an interactive plot. Plotly takes Python code and makes
beautiful looking JavaScript plots.
```

```
data['Dep_Time_hour'].apply(flight_dep_time).value_counts().iplot(kind
="bar")
```

{"config":{"linkText":"Export to
plot.ly","plotlyServerURL":"https://plot.ly","showLink":true},"data":
[{"marker":{"color":"rgba(255, 153, 51, 0.6)","line":
{"color":"rgba(255, 153, 51,
1.0)","width":1}},"name":"Dep_Time_hour","orientation":"v","text":"","
type":"bar","x":["Early
Morning","Evening","Morning","Noon","Night","late night"],"y":
[2880,2357,2209,1731,1040,465]}],"layout":{"legend":
{"bgcolor":"#F5F6F9","font":
{"color":"#4D5663"}},"paper_bgcolor":"#F5F6F9","plot_bgcolor":"#F5F6F9
","template":{"data":{"bar":[{"error_x":{"color":"#2a3f5f"},"error_y":
{"color":"#2a3f5f"},"marker":{"line":
{"color":"#E5ECF6","width":0.5},"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"bar"}],"barpo
lar":[{"marker":{"line":{"color":"#E5ECF6","width":0.5},"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"barpolar"}],"
carpet":[{"aaxis":
{"endlinecolor":"#2a3f5f","gridcolor":"white","linecolor":"white","min
orgridcolor":"white","startlinecolor":"#2a3f5f"},"baxis":
{"endlinecolor":"#2a3f5f","gridcolor":"white","linecolor":"white","min
orgridcolor":"white","startlinecolor":"#2a3f5f"},"type":"carpet"}],"ch
oropleth":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"choropleth"}],"contour":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"contour"}],"contourcarpet":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"contourcarpet"}],"heatmap":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"heatmap"}],"heatmapgl":[{"colorbar":
{"outlinewidth":0,"ticks":""},"colorscale":[[0,"#0d0887"],
[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],
[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],
[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],
[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"heatmapgl"}],"histogram":[{"marker":{"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"histogram"}],
"histogram2d":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],

```
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"histogram2d"}],"histogram2dcontour":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"histogram2dcontour"}],"mesh3d":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"mesh3d"}],"parcoords":[{"line":
{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"parcoords"}],"pie":
[{"automargin":true,"type":"pie"}],"scatter":[{"fillpattern":
{"fillmode":"overlay","size":10,"solidity":0.2},"type":"scatter"}],"sc
atter3d":[{"line":{"colorbar":{"outlinewidth":0,"ticks":""}},"marker":
{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatter3d"}],"scattercarpet":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattercarpet"}],"scattergeo":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattergeo"}],"scattergl":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattergl"}],"scattermapbox":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattermapbox"}],"scatterpolar"
:[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterpolar"}],"scatterpolargl
":[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterpolargl"}],"scatterterna
ry":[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterternary"}],"surface":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"surface"}],"table":[{"cells":{"fill":
{"color":"#EBF0F8"},"line":{"color":"white"}},"header":{"fill":
{"color":"#C8D4E3"},"line":
{"color":"white"}},"type":"table"}]},"layout":{"annotationdefaults":
{"arrowcolor":"#2a3f5f","arrowhead":0,"arrowwidth":1},"autotypenumbers
":"strict","coloraxis":{"colorbar":
{"outlinewidth":0,"ticks":""}},"colorscale":{"diverging":
[[0,"#8e0152"],[0.1,"#c51b7d"],[0.2,"#de77ae"],[0.3,"#f1b6da"],
[0.4,"#fde0ef"],[0.5,"#f7f7f7"],[0.6,"#e6f5d0"],[0.7,"#b8e186"],
[0.8,"#7fbc41"],[0.9,"#4d9221"],[1,"#276419"]],"sequential":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
```

```
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],[1,"#f0f921"]],"sequentialminus":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],[1,"#f0f921"]]},"colorway":
["#636efa","#EF553B","#00cc96","#ab63fa","#FFA15A","#19d3f3","#FF6692"
,"#B6E880","#FF97FF","#FECB52"],"font":{"color":"#2a3f5f"},"geo":
{"bgcolor":"white","lakecolor":"white","landcolor":"#E5ECF6","showlake
s":true,"showland":true,"subunitcolor":"white"},"hoverlabel":
{"align":"left"},"hovermode":"closest","mapbox":
{"style":"light"},"paper_bgcolor":"white","plot_bgcolor":"#E5ECF6","po
lar":{"angularaxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"bgcolor":"#E5ECF
6","radialaxis":
{"gridcolor":"white","linecolor":"white","ticks":""}},"scene":
{"xaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
,"yaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
,"zaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
},"shapedefaults":{"line":{"color":"#2a3f5f"}},"ternary":{"aaxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"baxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"bgcolor":"#E5ECF
6","caxis":
{"gridcolor":"white","linecolor":"white","ticks":""}},"title":
{"x":5.0e-2},"xaxis":
{"automargin":true,"gridcolor":"white","linecolor":"white","ticks":"",
"title":
{"standoff":15},"zerolinecolor":"white","zerolinewidth":2},"yaxis":
{"automargin":true,"gridcolor":"white","linecolor":"white","ticks":"",
"title":
{"standoff":15},"zerolinecolor":"white","zerolinewidth":2}}},"title":
{"font":{"color":"#4D5663"}},"xaxis":
{"gridcolor":"#E1E5ED","showgrid":true,"tickfont":
{"color":"#4D5663"},"title":{"font":
{"color":"#4D5663"},"text":""},"zerolinecolor":"#E1E5ED"},"yaxis":
{"gridcolor":"#E1E5ED","showgrid":true,"tickfont":
{"color":"#4D5663"},"title":{"font":
{"color":"#4D5663"},"text":""},"zerolinecolor":"#E1E5ED"}}}
```

6.. Pre-process Duration Feature & extract meaningful features from it..

Lets Apply pre-processing on duration column,

```
-->> Once we pre-processed our Duration feature , lets extract
Duration hours and minute from duration..

-->> As my ML model is not able to understand this duration as it
contains string values ,
thats why we have to tell our ML Model that this is hour & this is
minute for each of the row ..

data.head(3)

       Airline Date_of_Journey    Source Destination
Route  \
0       IndiGo      2019-03-24  Banglore   New Delhi              BLR
→ DEL
1    Air India      2019-01-05   Kolkata    Banglore  CCU → IXR → BBI
→ BLR
2  Jet Airways      2019-09-06     Delhi      Cochin  DEL → LKO → BOM
→ COK

  Duration Total_Stops Additional_Info  Price  Journey_day
Journey_month  \
0   2h 50m    non-stop         No info   3897           24
3
1   7h 25m     2 stops         No info   7662            5
1
2      19h     2 stops         No info  13882            6
9

   Journey_year  Dep_Time_hour  Dep_Time_minute  Arrival_Time_hour  \
0          2019             22               20                  1
1          2019              5               50                 13
2          2019              9               25                  4

   Arrival_Time_minute
0                   10
1                   15
2                   25
```

```
def preprocess_duration(x):
    if 'h' not in x:
        x = '0h' + ' ' + x
    elif 'm' not in x:
        x = x + ' ' +'0m'

    return x

data['Duration'] = data['Duration'].apply(preprocess_duration)

data['Duration']

0        2h 50m
1        7h 25m
2        19h 0m
3        5h 25m
4        4h 45m
          ...
10678    2h 30m
10679    2h 35m
10680     3h 0m
10681    2h 40m
10682    8h 20m
Name: Duration, Length: 10682, dtype: object

'''
    Now after pre-processing duration feature , still my ml_model is
not able to understand duration
    bcz it is string data so any how we have to convert it into
numerical(integer of float) values

'''

'\n    Now after pre-processing duration feature , still my ml_model
is not able to understand duration \n    bcz it is string data so any
how we have to convert it into numerical(integer of float) values\n\n'



data['Duration'][0]

'2h 50m'

'2h 50m'.split(' ')

['2h', '50m']

'2h 50m'.split(' ')[0]

'2h'

'2h 50m'.split(' ')[0][0:-1]
```

```
'2'

type('2h 50m'.split(' ')[0][0:-1])

str

int('2h 50m'.split(' ')[0][0:-1])

2

int('2h 50m'.split(' ')[1][0:-1])

50

data['Duration_hours'] = data['Duration'].apply(lambda x :
int(x.split(' ')[0][0:-1]))

data['Duration_mins'] = data['Duration'].apply(lambda x :
int(x.split(' ')[1][0:-1]))

data.head(2)

     Airline Date_of_Journey    Source Destination
Route  \
0     IndiGo      2019-03-24  Banglore   New Delhi              BLR →
DEL
1  Air India      2019-01-05   Kolkata    Banglore  CCU → IXR → BBI →
BLR

  Duration Total_Stops Additional_Info  Price  Journey_day
Journey_month  \
0   2h 50m    non-stop         No info   3897           24
3
1   7h 25m     2 stops         No info   7662            5
1

   Journey_year  Dep_Time_hour  Dep_Time_minute  Arrival_Time_hour  \
0          2019             22               20                  1
1          2019              5               50                 13

   Arrival_Time_minute  Duration_hours  Duration_mins
0                   10               2             50
1                   15               7             25
```

7.. Lets Analyse whether Duration impacts Price or not ?

```
data['Duration'] ## convert duration into total minutes duration ..
```

```
0        2h 50m
1        7h 25m
2        19h 0m
3        5h 25m
4        4h 45m
          ...
10678    2h 30m
10679    2h 35m
10680     3h 0m
10681    2h 40m
10682    8h 20m
Name: Duration, Length: 10682, dtype: object
```

```
2*60
```

```
120
```

```
'2*60'
```

```
'2*60'
```

```
eval('2*60')
```

```
120
```

```
data['Duration_total_mins'] =
data['Duration'].str.replace('h' ,"*60").str.replace(' ' ,
'+').str.replace('m' , "*1").apply(eval)
```

```
data['Duration_total_mins']
```

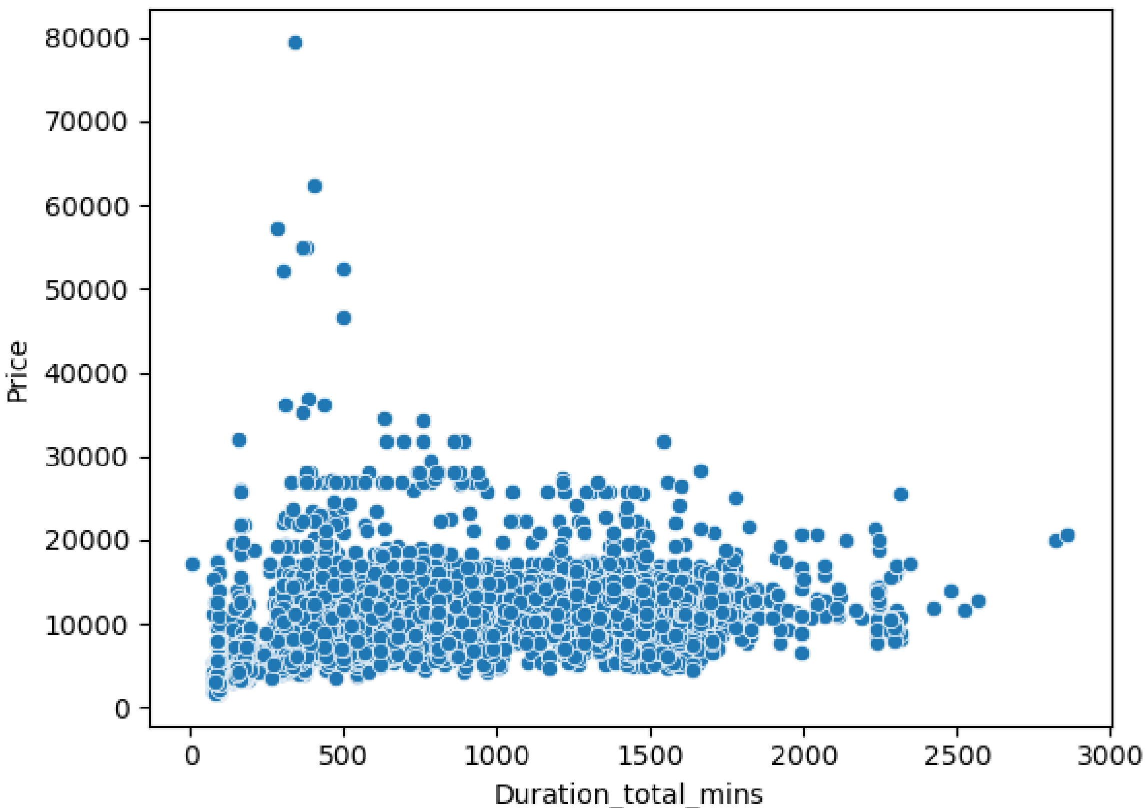
```
0         170
1         445
2        1140
3         325
4         285
         ...
10678     150
10679     155
10680     180
10681     160
10682     500
Name: Duration_total_mins, Length: 10682, dtype: int64
```

```
data.columns
```

```
Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Duration', 'Total_Stops', 'Additional_Info', 'Price',
'Journey_day',
       'Journey_month', 'Journey_year', 'Dep_Time_hour',
'Dep_Time_minute',
       'Arrival_Time_hour', 'Arrival_Time_minute', 'Duration_hours',
       'Duration_mins', 'Duration_total_mins'],
      dtype='object')
```

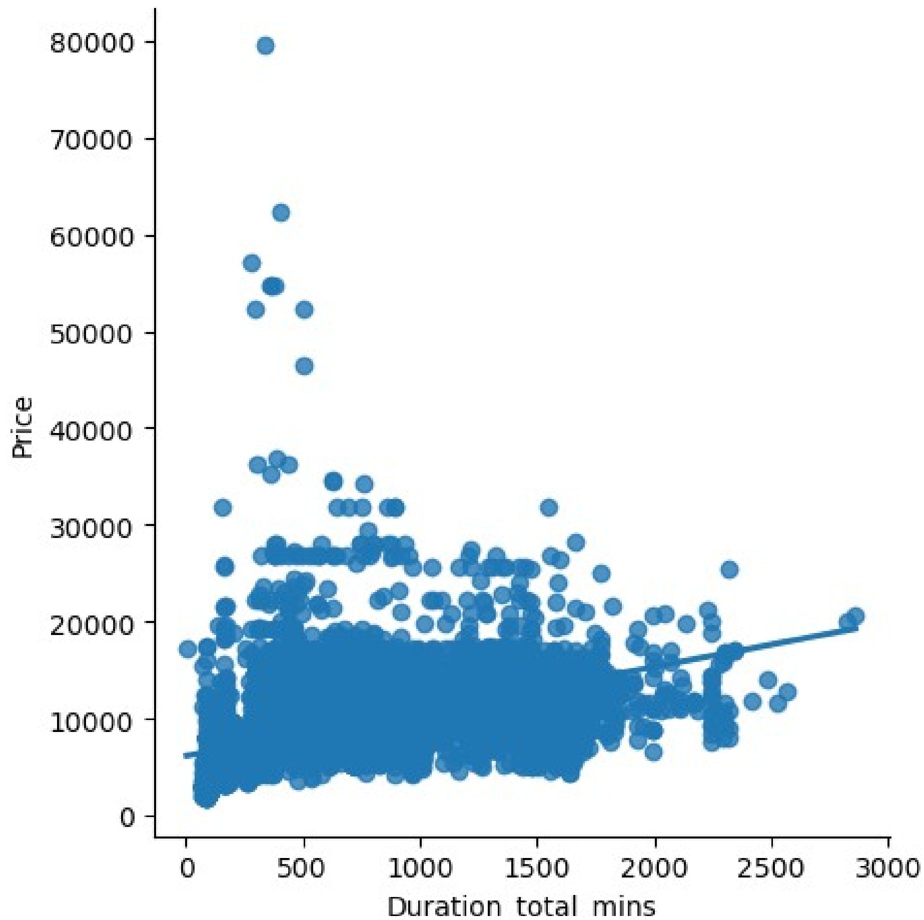
```
sns.scatterplot(x="Duration_total_mins" , y="Price" , data=data)
```

```
<AxesSubplot:xlabel='Duration_total_mins', ylabel='Price'>
```



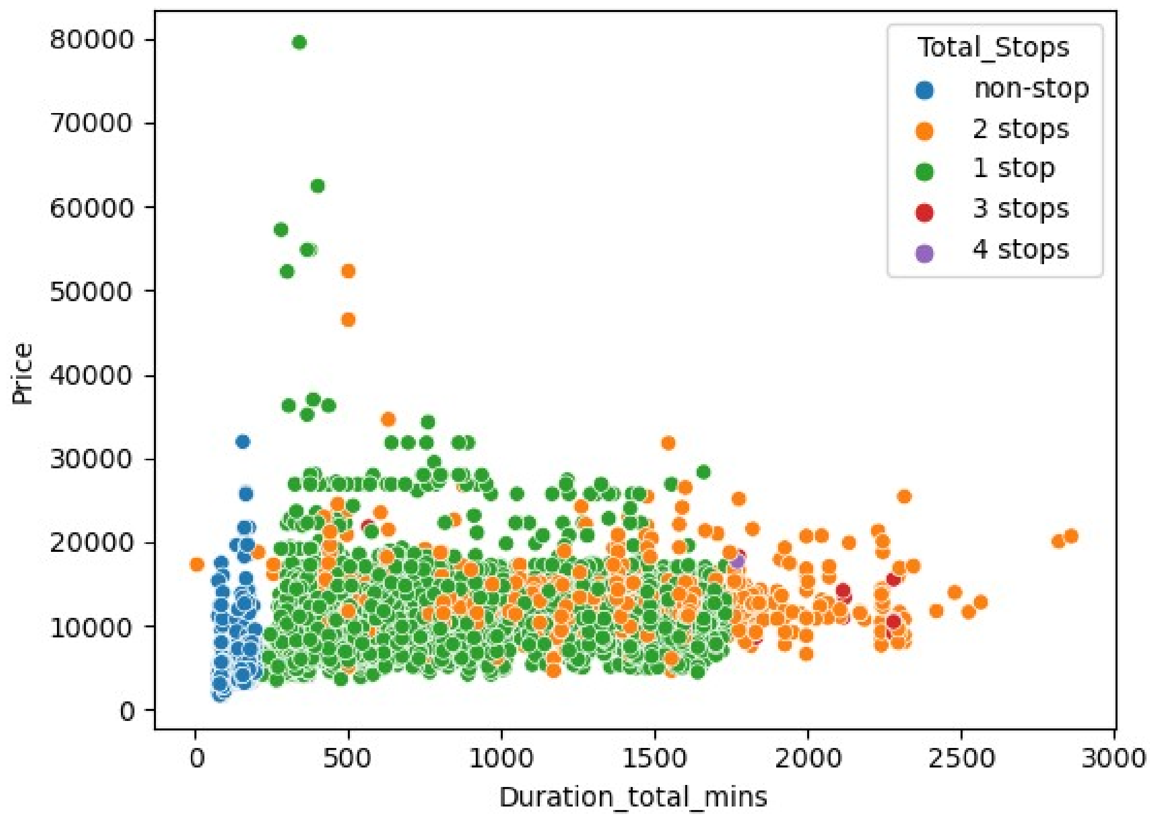

```
sns.lmplot(x="Duration_total_mins" , y="Price" , data=data)

### pretty clear that As the duration of minutes increases Flight
price also increases.
```

```
<seaborn.axisgrid.FacetGrid at 0x1fa69b73e80>
```




```python
### lets understand whether total stops affect price or not !

sns.scatterplot(x="Duration_total_mins" , y="Price" ,
hue="Total_Stops", data=data)
```

```
<AxesSubplot:xlabel='Duration_total_mins', ylabel='Price'>
```




```
'''
Non stops flights take less duration while their fare is also low,
then as the stop increases,
duration also increases and price also increases(in most of the cases)

'''
```

```
'\nNon stops flights take less duration while their fare is also low,
then as the stop increases, \nduration also increases and price also
increases(in most of the cases)\n\n'
```

8.. on which route Jet Airways is extremely used?

```
data['Airline']=='Jet Airways'
```

```
0        False
1        False
```

```
2         True
3        False
4        False
         ...
10678    False
10679    False
10680     True
10681    False
10682    False
Name: Airline, Length: 10682, dtype: bool
```

```
data[data['Airline']=='Jet
Airways'].groupby('Route').size().sort_values(ascending=False)
```

```
Route
CCU → BOM → BLR          930
DEL → BOM → COK          875
BLR → BOM → DEL          385
BLR → DEL                382
CCU → DEL → BLR          300
BOM → HYD                207
DEL → JAI → BOM → COK    207
DEL → AMD → BOM → COK    141
DEL → IDR → BOM → COK     86
DEL → NAG → BOM → COK     61
DEL → ATQ → BOM → COK     38
DEL → COK                 34
DEL → BHO → BOM → COK     29
DEL → BDQ → BOM → COK     28
DEL → LKO → BOM → COK     25
DEL → JDH → BOM → COK     23
CCU → GAU → BLR           22
DEL → MAA → BOM → COK     16
DEL → IXC → BOM → COK     13
BLR → MAA → DEL           10
BLR → BDQ → DEL            8
DEL → UDR → BOM → COK      7
BOM → DEL → HYD            5
CCU → BOM → PNQ → BLR      4
BLR → BOM → JDH → DEL      3
DEL → DED → BOM → COK      2
BOM → BDQ → DEL → HYD      2
DEL → CCU → BOM → COK      1
BOM → VNS → DEL → HYD      1
BOM → UDR → DEL → HYD      1
BOM → JDH → DEL → HYD      1
BOM → IDR → DEL → HYD      1
BOM → DED → DEL → HYD      1
dtype: int64
```

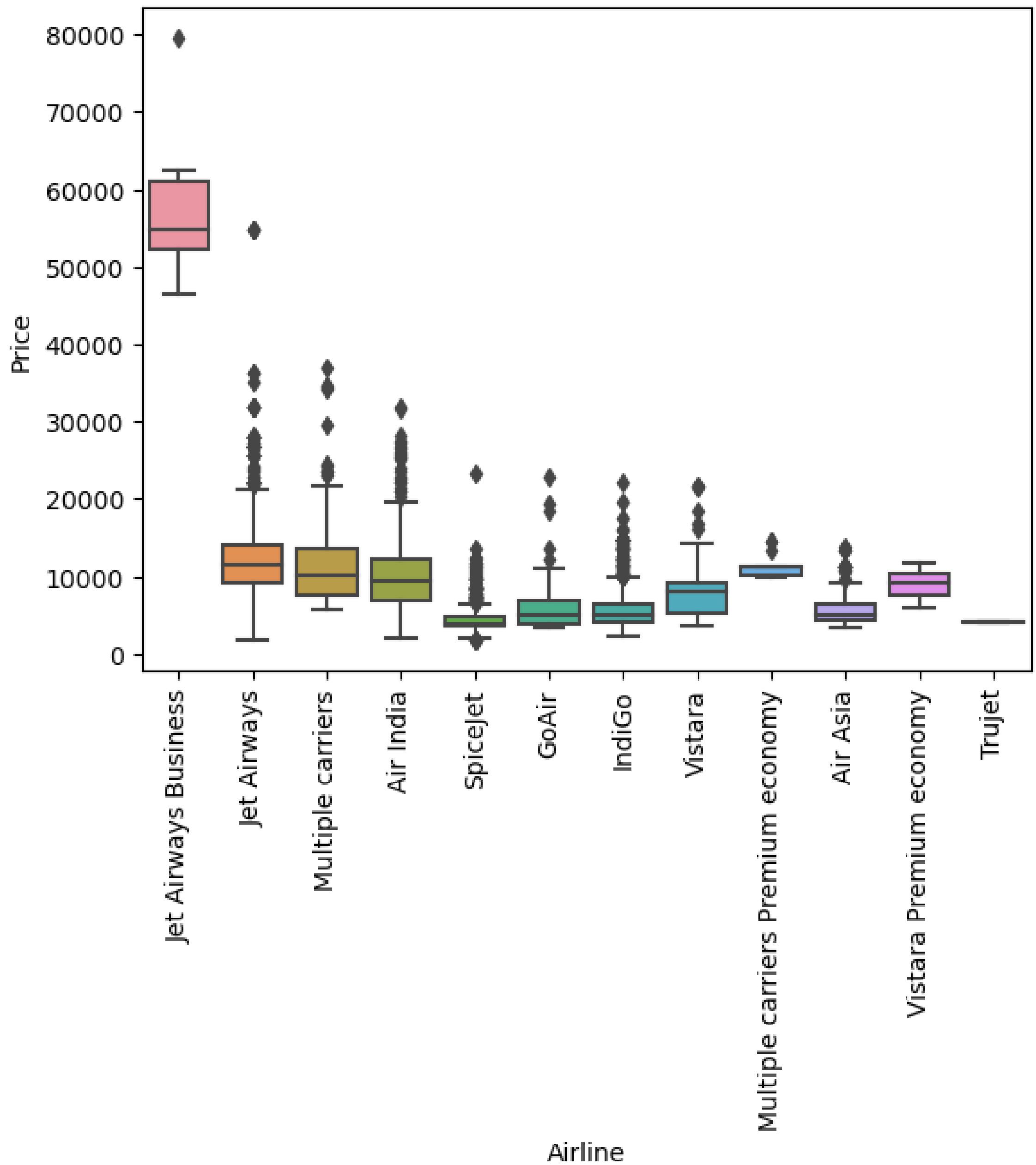
b.. Performing Airline vs Price Analysis..

```
    ie find price distribution & 5-point summary of each Airline..
```

```
data.columns
```

```
Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Duration', 'Total_Stops', 'Additional_Info', 'Price',
'Journey_day',
       'Journey_month', 'Journey_year', 'Dep_Time_hour',
'Dep_Time_minute',
       'Arrival_Time_hour', 'Arrival_Time_minute', 'Duration_hours',
       'Duration_mins', 'Duration_total_mins'],
      dtype='object')
```

```
sns.boxplot(y='Price' , x='Airline' , data=data.sort_values('Price' ,
ascending=False))
plt.xticks(rotation="vertical")
plt.show()
```




```
'''

Conclusion--> From graph we can see that Jet Airways Business have the
highest Price.,
              Apart from the first Airline almost all are having
similar median

'''
```

9.. Applying one-hot Encoding on data..

```
data.head(2)
```

```
     Airline Date_of_Journey    Source Destination
Route  \
0     IndiGo      2019-03-24  Banglore   New Delhi              BLR →
DEL
1  Air India      2019-01-05   Kolkata    Banglore  CCU → IXR → BBI →
BLR

  Duration Total_Stops Additional_Info  Price  Journey_day
Journey_month  \
0   2h 50m    non-stop         No info   3897           24
3
1   7h 25m     2 stops         No info   7662            5
1

   Journey_year  Dep_Time_hour  Dep_Time_minute  Arrival_Time_hour  \
0          2019             22               20                  1
1          2019              5               50                 13

   Arrival_Time_minute  Duration_hours  Duration_mins
Duration_total_mins
0                   10               2             50
170
1                   15               7             25
445
```

```
'''

Categorical data refers to a data type that can be stored into
groups/categories/labels
Examples of categorical variables are  age group, educational
level,blood type etc..


Numerical data refers to the data that is in the form of numbers,
Examples of numerical data are height, weight, age etc..

Numerical data has two categories: discrete data and continuous data


Discrete data : It basically takes countable numbers like 1, 2, 3, 4,
5, and so on.
                In case of infinity, these numbers will keep going
on...
```

```
                age of a fly : 8 , 9 day etc..

Continuous data : which is continuous in nature
                  amount of sugar , 11.2 kg  , temp of a city  , your
bank balance !

For example, salary levels and performance classifications are
discrete variables,
whereas height and weight are continuous variables.

'''

cat_col = [col for col in data.columns if data[col].dtype=="object"]

num_col = [col for col in data.columns if data[col].dtype!="object"]
```

Handling Categorical Data

```
We are using 2 basic Encoding Techniques to convert Categorical data
into some numerical format
Nominal data --> data are not in any order --> OneHotEncoder is used
in this case
Ordinal data --> data are in order -->       LabelEncoder is used in
this case

But in real-world , it is not necessary that u have to always One-hot
or label ,
hence we will discuss more interesting approaches in upcoming sessions
to do this !

cat_col

['Airline',
 'Source',
 'Destination',
 'Route',
 'Duration',
 'Total_Stops',
 'Additional_Info']



### Applying One-hot from scratch :

data['Source'].unique()

array(['Banglore', 'Kolkata', 'Delhi', 'Chennai', 'Mumbai'],
dtype=object)

data['Source'].apply(lambda x : 1 if x=='Banglore' else 0)
```

```
0        1
1        0
2        0
3        0
4        1
        ..
10678    0
10679    0
10680    1
10681    1
10682    0
Name: Source, Length: 10682, dtype: int64
```

```python
for sub_category in data['Source'].unique():
    data['Source_'+sub_category] = data['Source'].apply(lambda x : 1
if x==sub_category else 0)

data.head(3)
```

```
       Airline Date_of_Journey    Source Destination
Route  \
0       IndiGo      2019-03-24  Banglore   New Delhi              BLR
→ DEL
1    Air India      2019-01-05   Kolkata    Banglore  CCU → IXR → BBI
→ BLR
2  Jet Airways      2019-09-06     Delhi      Cochin  DEL → LKO → BOM
→ COK

  Duration Total_Stops Additional_Info  Price  Journey_day  ...  \
0   2h 50m    non-stop         No info   3897           24  ...
1   7h 25m     2 stops         No info   7662            5  ...
2   19h 0m     2 stops         No info  13882            6  ...

   Arrival_Time_hour  Arrival_Time_minute  Duration_hours
Duration_mins  \
0                  1                   10               2
50
1                 13                   15               7
25
2                  4                   25              19
0

   Duration_total_mins  Source_Banglore  Source_Kolkata  Source_Delhi
\
0                  170                1               0             0

1                  445                0               1             0

2                 1140                0               0             1
```

```
   Source_Chennai  Source_Mumbai
0               0              0
1               0              0
2               0              0

[3 rows x 24 columns]
```

10.. Lets Perform target guided encoding on Data

```
ofcourse we can use One-hot , but if we have more sub-categories , it
creates curse of dimensionality
lets use Target Guided Mean Encoding in such case to get rid of curse
of dimensionality..
```

```
'''

Now on 2 features , Airline & Destination , we can apply on-hot as
there is no such order
but total_stops is my ordinal data , it makes no sense if we apply on-
hot on top of this..
similarly if we have any feature which have more categories , it is
not good to apply one-hot as it will create
curse of dimensionality issue , which leads to usage of more resources
of your pc..

So we can think for appplying mean Encoding or better techniques like
Target Guided Ordinal Encoding !


'''
```

```
cat_col
```

```
['Airline',
 'Source',
 'Destination',
 'Route',
 'Duration',
 'Total_Stops',
 'Additional_Info']
```

```
data.head(2)
```

```
     Airline Date_of_Journey    Source Destination
Route  \
0     IndiGo      2019-03-24  Banglore   New Delhi              BLR →
DEL
1  Air India      2019-01-05   Kolkata    Banglore  CCU → IXR → BBI →
BLR

  Duration Total_Stops Additional_Info  Price  Journey_day  ...  \
0   2h 50m    non-stop         No info   3897           24  ...
1   7h 25m     2 stops         No info   7662            5  ...

   Arrival_Time_hour  Arrival_Time_minute  Duration_hours
Duration_mins  \
0                  1                   10               2
50
1                 13                   15               7
25

   Duration_total_mins  Source_Banglore  Source_Kolkata  Source_Delhi
\
0                  170                1               0             0

1                  445                0               1             0


   Source_Chennai  Source_Mumbai
0               0              0
1               0              0

[2 rows x 24 columns]
```

```
data['Airline'].nunique()
```

```
12
```

```
data.groupby(['Airline'])['Price'].mean().sort_values()
```

```
Airline
Trujet                                      4140.000000
SpiceJet                                    4338.284841
Air Asia                                    5590.260188
IndiGo                                      5673.682903
GoAir                                       5861.056701
Vistara                                     7796.348643
Vistara Premium economy                     8962.333333
Air India                                   9612.427756
```

```
Multiple carriers                          10902.678094
Multiple carriers Premium economy          11418.846154
Jet Airways                                11643.923357
Jet Airways Business                       58358.666667
Name: Price, dtype: float64
```

```
airlines = data.groupby(['Airline'])
['Price'].mean().sort_values().index

airlines
```

```
Index(['Trujet', 'SpiceJet', 'Air Asia', 'IndiGo', 'GoAir', 'Vistara',
       'Vistara Premium economy', 'Air India', 'Multiple carriers',
       'Multiple carriers Premium economy', 'Jet Airways',
       'Jet Airways Business'],
      dtype='object', name='Airline')
```

```
dict_airlines = {key:index for index , key in enumerate(airlines , 0)}

dict_airlines
```

```
{'Trujet': 0,
 'SpiceJet': 1,
 'Air Asia': 2,
 'IndiGo': 3,
 'GoAir': 4,
 'Vistara': 5,
 'Vistara Premium economy': 6,
 'Air India': 7,
 'Multiple carriers': 8,
 'Multiple carriers Premium economy': 9,
 'Jet Airways': 10,
 'Jet Airways Business': 11}
```

```
data['Airline'] = data['Airline'].map(dict_airlines)

data['Airline']
```

```
0         3
1         7
2        10
3         3
4         3
         ..
10678     2
10679     7
10680    10
10681     5
10682     7
Name: Airline, Length: 10682, dtype: int64
```

```
data.head(3)
```

```
   Airline Date_of_Journey    Source Destination
Route  \
0        3      2019-03-24  Banglore   New Delhi              BLR →
DEL
1        7      2019-01-05   Kolkata    Banglore  CCU → IXR → BBI →
BLR
2       10      2019-09-06     Delhi      Cochin  DEL → LKO → BOM →
COK

  Duration Total_Stops Additional_Info  Price  Journey_day  ...  \
0   2h 50m    non-stop         No info   3897           24  ...
1   7h 25m     2 stops         No info   7662            5  ...
2   19h 0m     2 stops         No info  13882            6  ...

   Arrival_Time_hour  Arrival_Time_minute  Duration_hours
Duration_mins  \
0                  1                   10               2
50
1                 13                   15               7
25
2                  4                   25              19
0

   Duration_total_mins  Source_Banglore  Source_Kolkata  Source_Delhi
\
0                  170                1               0             0

1                  445                0               1             0

2                 1140                0               0             1


   Source_Chennai  Source_Mumbai
0               0              0
1               0              0
2               0              0

[3 rows x 24 columns]
```

```
### now lets perform Target Guided Mean encoding on 'Destination' ..

data['Destination'].unique()
```

```
array(['New Delhi', 'Banglore', 'Cochin', 'Kolkata', 'Delhi',
'Hyderabad'],
      dtype=object)
```

```python
'''

till now,Delhi has only one Airport which is IGI & its second Airport
is yet to build in Greater Noida (Jewar)
which is neighbouring part of Delhi so we will consider New Delhi &
Delhi as same

but in future , these conditions may change..


'''

data['Destination'].replace('New Delhi' , 'Delhi' , inplace=True)

data['Destination'].unique()
```

```
array(['Delhi', 'Banglore', 'Cochin', 'Kolkata', 'Hyderabad'],
      dtype=object)
```

```python
dest = data.groupby(['Destination'])
['Price'].mean().sort_values().index

dest
```

```
Index(['Kolkata', 'Hyderabad', 'Delhi', 'Banglore', 'Cochin'],
dtype='object', name='Destination')
```

```python
dict_dest = {key:index for index , key in enumerate(dest , 0)}

dict_dest
```

```
{'Kolkata': 0, 'Hyderabad': 1, 'Delhi': 2, 'Banglore': 3, 'Cochin': 4}
```

```python
data['Destination'] = data['Destination'].map(dict_dest)

data['Destination']
```

```
0        2
1        3
2        4
3        3
4        2
        ..
10678    3
10679    3
10680    2
10681    2
10682    4
Name: Destination, Length: 10682, dtype: int64
```

```python
data.head(3)
```

```
   Airline Date_of_Journey    Source  Destination
Route  \
0        3      2019-03-24  Banglore            2              BLR →
DEL
1        7      2019-01-05   Kolkata            3  CCU → IXR → BBI →
BLR
2       10      2019-09-06     Delhi            4  DEL → LKO → BOM →
COK

  Duration Total_Stops Additional_Info  Price  Journey_day  ...  \
0   2h 50m    non-stop         No info   3897           24  ...
1   7h 25m     2 stops         No info   7662            5  ...
2   19h 0m     2 stops         No info  13882            6  ...

   Arrival_Time_hour  Arrival_Time_minute  Duration_hours
Duration_mins  \
0                  1                   10               2
50
1                 13                   15               7
25
2                  4                   25              19
0

   Duration_total_mins  Source_Banglore  Source_Kolkata  Source_Delhi
\
0                  170                1               0             0

1                  445                0               1             0

2                 1140                0               0             1


   Source_Chennai  Source_Mumbai
0               0              0
1               0              0
2               0              0

[3 rows x 24 columns]
```

11.. Perform Label(Manual) Encoding on Data

```
data.head(3)
```

```
   Airline Date_of_Journey    Source  Destination
Route  \
0        3      2019-03-24  Banglore            2              BLR →
DEL
1        7      2019-01-05   Kolkata            3  CCU → IXR → BBI →
BLR
2       10      2019-09-06     Delhi            4  DEL → LKO → BOM →
COK

  Duration Total_Stops Additional_Info  Price  Journey_day  ...  \
0   2h 50m    non-stop         No info   3897           24  ...
1   7h 25m     2 stops         No info   7662            5  ...
2   19h 0m     2 stops         No info  13882            6  ...

   Arrival_Time_hour  Arrival_Time_minute  Duration_hours
Duration_mins  \
0                  1                   10               2
50
1                 13                   15               7
25
2                  4                   25              19
0

   Duration_total_mins  Source_Banglore  Source_Kolkata  Source_Delhi
\
0                  170                1               0             0

1                  445                0               1             0

2                 1140                0               0             1


   Source_Chennai  Source_Mumbai
0               0              0
1               0              0
2               0              0

[3 rows x 24 columns]
```

```
data['Total_Stops']
```

```
0        non-stop
1         2 stops
2         2 stops
3          1 stop
4          1 stop
           ...
10678    non-stop
10679    non-stop
10680    non-stop
```

```
10681    non-stop
10682     2 stops
Name: Total_Stops, Length: 10682, dtype: object

data['Total_Stops'].unique()

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

# As this is case of Ordinal Categorical type we perform Label
encoding from scratch !
# Here Values are assigned with corresponding key

stop = {'non-stop':0, '2 stops':2, '1 stop':1, '3 stops':3, '4
stops':4}

data['Total_Stops'] = data['Total_Stops'].map(stop)

data['Total_Stops']

0        0
1        2
2        2
3        1
4        1
        ..
10678    0
10679    0
10680    0
10681    0
10682    2
Name: Total_Stops, Length: 10682, dtype: int64
```

b.. Remove Un-necessary features

```
data.head(1)

   Airline Date_of_Journey    Source  Destination      Route Duration
\
0        3      2019-03-24  Banglore            2  BLR → DEL   2h 50m


   Total_Stops Additional_Info  Price  Journey_day  ...
Arrival_Time_hour  \
0            0         No info   3897           24  ...
1

   Arrival_Time_minute  Duration_hours  Duration_mins
```

```
Duration_total_mins  \
0                   10               2             50
170

   Source_Banglore  Source_Kolkata  Source_Delhi  Source_Chennai  \
0                1               0             0               0

   Source_Mumbai
0              0

[1 rows x 24 columns]
```

```
data.columns
```

```
Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Duration', 'Total_Stops', 'Additional_Info', 'Price',
'Journey_day',
       'Journey_month', 'Journey_year', 'Dep_Time_hour',
'Dep_Time_minute',
       'Arrival_Time_hour', 'Arrival_Time_minute', 'Duration_hours',
       'Duration_mins', 'Duration_total_mins', 'Source_Banglore',
       'Source_Kolkata', 'Source_Delhi', 'Source_Chennai',
'Source_Mumbai'],
      dtype='object')
```

```
data['Additional_Info'].value_counts()/len(data)*100

# Additional_Info contains almost 80% no_info,so we can drop this
column
```

```
No info                         78.112713
In-flight meal not included     18.554578
No check-in baggage included     2.995694
1 Long layover                   0.177869
Change airports                  0.065531
Business class                   0.037446
No Info                          0.028085
1 Short layover                  0.009362
Red-eye flight                   0.009362
2 Long layover                   0.009362
Name: Additional_Info, dtype: float64
```

```
data.head(4)
```

```
   Airline Date_of_Journey    Source  Destination
Route  \
0        3      2019-03-24  Banglore            2              BLR →
DEL
1        7      2019-01-05   Kolkata            3  CCU → IXR → BBI →
BLR
2       10      2019-09-06     Delhi            4  DEL → LKO → BOM →
```

```
COK
3         3      2019-12-05   Kolkata           3        CCU → NAG →
BLR

  Duration  Total_Stops Additional_Info  Price  Journey_day  ...  \
0   2h 50m            0         No info   3897           24  ...
1   7h 25m            2         No info   7662            5  ...
2   19h 0m            2         No info  13882            6  ...
3   5h 25m            1         No info   6218            5  ...

   Arrival_Time_hour  Arrival_Time_minute  Duration_hours
Duration_mins  \
0                  1                   10               2
50
1                 13                   15               7
25
2                  4                   25              19
0
3                 23                   30               5
25

   Duration_total_mins  Source_Banglore  Source_Kolkata  Source_Delhi
\
0                  170                1               0             0

1                  445                0               1             0

2                 1140                0               0             1

3                  325                0               1             0


   Source_Chennai  Source_Mumbai
0               0              0
1               0              0
2               0              0
3               0              0

[4 rows x 24 columns]
```

```
data.columns
```

```
Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Duration', 'Total_Stops', 'Additional_Info', 'Price',
'Journey_day',
       'Journey_month', 'Journey_year', 'Dep_Time_hour',
'Dep_Time_minute',
       'Arrival_Time_hour', 'Arrival_Time_minute', 'Duration_hours',
       'Duration_mins', 'Duration_total_mins', 'Source_Banglore',
```

```
       'Source_Kolkata', 'Source_Delhi', 'Source_Chennai',
'Source_Mumbai'],
      dtype='object')

data['Journey_year'].unique()

array([2019], dtype=int64)

'''

lets drop Date_of_Journey as well as we have already extracted
"Journey_hour" , "jpuney_month" , Journey_day"..
Additional_Info contains almost 80% no_info , so we can drop this
column ..
lets drop Duration_total_mins as we have already extracted
"Duration_hours" & "Duration_mins"
Lets drop "Source" feature as well as we have already perform feature
encoding on this Feature
lets drop Journey_year as well , as it has constant values throughtout
dataframe which is 2019..

'''

data.drop(columns=['Date_of_Journey' , 'Additional_Info' ,
'Duration_total_mins' , 'Source' , 'Journey_year'] , axis=1 ,
inplace=True)



data.columns

Index(['Airline', 'Destination', 'Route', 'Duration', 'Total_Stops',
'Price',
       'Journey_day', 'Journey_month', 'Dep_Time_hour',
'Dep_Time_minute',
       'Arrival_Time_hour', 'Arrival_Time_minute', 'Duration_hours',
       'Duration_mins', 'Source_Banglore', 'Source_Kolkata',
'Source_Delhi',
       'Source_Chennai', 'Source_Mumbai'],
      dtype='object')

data.head(4)

   Airline  Destination                  Route Duration  Total_Stops
Price  \
0        3            2              BLR → DEL   2h 50m            0
3897
1        7            3  CCU → IXR → BBI → BLR   7h 25m            2
7662
2       10            4  DEL → LKO → BOM → COK   19h 0m            2
13882
3        3            3        CCU → NAG → BLR   5h 25m            1
```

```
6218

   Journey_day  Journey_month  Dep_Time_hour  Dep_Time_minute  \
0           24              3             22               20   
1            5              1              5               50   
2            6              9              9               25   
3            5             12             18                5   

   Arrival_Time_hour  Arrival_Time_minute  Duration_hours
Duration_mins  \
0                  1                   10               2
50  
1                 13                   15               7
25  
2                  4                   25              19
0  
3                 23                   30               5
25  

   Source_Banglore  Source_Kolkata  Source_Delhi  Source_Chennai  \
0                1               0             0               0   
1                0               1             0               0   
2                0               0             1               0   
3                0               1             0               0   

   Source_Mumbai  
0              0  
1              0  
2              0  
3              0  
```

```
data.drop(columns=['Route'] , axis=1 , inplace=True)

## we can drop Route as well bcz Route is directly related to Total
stops & considering 2 same features doesnt make sense while building
ML model..

data.head(3)
```

```
   Airline  Destination Duration  Total_Stops  Price  Journey_day  \
0        3            2   2h 50m            0   3897           24   
1        7            3   7h 25m            2   7662            5   
2       10            4   19h 0m            2  13882            6   

   Journey_month  Dep_Time_hour  Dep_Time_minute  Arrival_Time_hour  \
0              3             22               20                  1   
1              1              5               50                 13   
2              9              9               25                  4   

   Arrival_Time_minute  Duration_hours  Duration_mins  Source_Banglore
\
```

```
0                    10                 2              50                 1

1                    15                 7              25                 0

2                    25                19               0                 0


   Source_Kolkata  Source_Delhi  Source_Chennai  Source_Mumbai
0               0             0               0              0
1               1             0               0              0
2               0             1               0              0
```

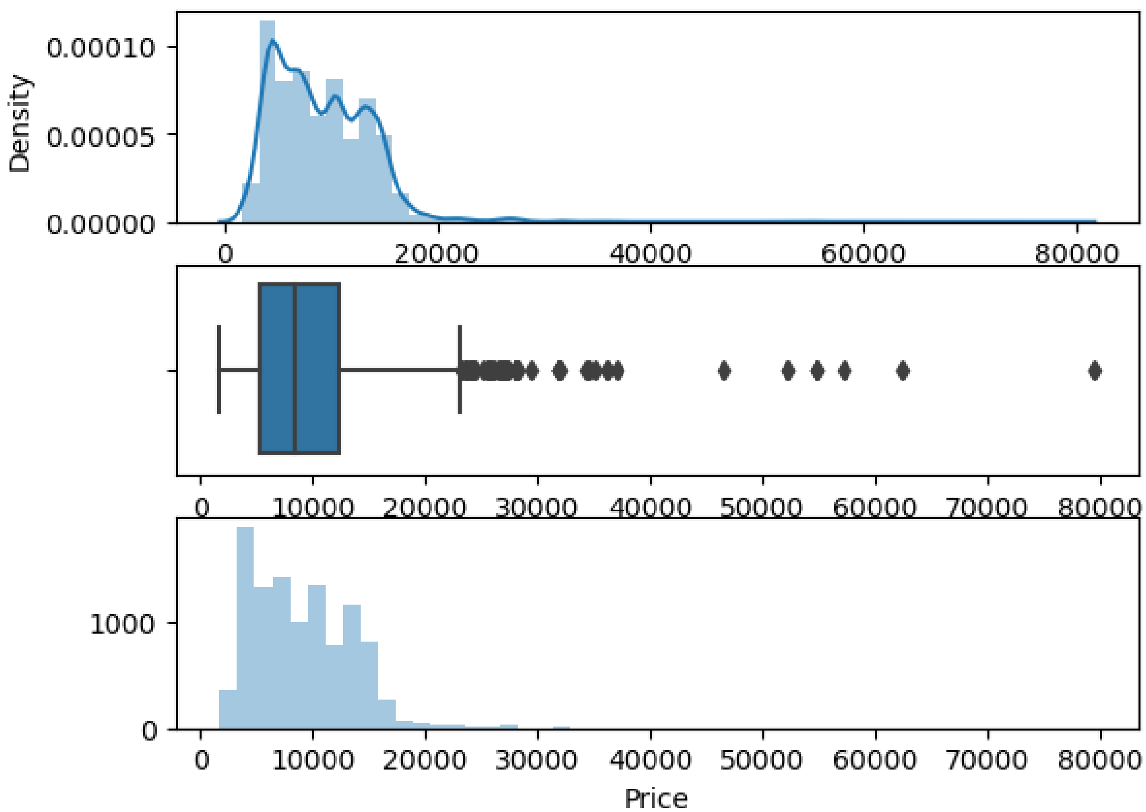
```
data.drop(columns=['Duration'] , axis=1 , inplace=True)

## we can drop "Duration" feature as we have extracted "Duration hour"
& "Duration Minute"..

data.head(3)
```

```
   Airline  Destination  Total_Stops  Price  Journey_day
Journey_month  \
0        3            2            0   3897           24
3
1        7            3            2   7662            5
1
2       10            4            2  13882            6
9

   Dep_Time_hour  Dep_Time_minute  Arrival_Time_hour
Arrival_Time_minute  \
0             22               20                  1
10
1              5               50                 13
15
2              9               25                  4
25

   Duration_hours  Duration_mins  Source_Banglore  Source_Kolkata  \
0               2             50                1               0
1               7             25                0               1
2              19              0                0               0

   Source_Delhi  Source_Chennai  Source_Mumbai
0             0               0              0
1             0               0              0
2             1               0              0
```

12.. Lets Perform outlier detection !

Here the list of data visualization plots to spot the outliers.

```
1. Box and whisker plot (box plot).
2. Scatter plot.
3. Histogram.
4. Distribution Plot.

def plot(df, col):
    fig , (ax1 , ax2 , ax3) = plt.subplots(3,1)

    sns.distplot(df[col] , ax=ax1)
    sns.boxplot(df[col] , ax=ax2)
    sns.distplot(df[col] , ax=ax3 , kde=False)

plot(data , 'Price')
```



```
    If Features Are Skewed We Use the below Technique which is IQR
    Data which are greater than IQR +1.5 IQR and data which are below
than IQR - 1.5 IQR are my outliers
    where ,  IQR = 75th%ile data - 25th%ile data

     & IQR +- 1.5 IQR  will be changed depending upon the domain ie it
could be sometimes IQR +- 3IQR
```

```python
q1 = data['Price'].quantile(0.25)
q3 = data['Price'].quantile(0.75)

iqr = q3- q1

maximum = q3 + 1.5*iqr
minimum = q1 - 1.5*iqr

print(maximum)
```

```
23017.0
```

```python
print(minimum)
```

```
-5367.0
```

```python
print([price for price in data['Price'] if price> maximum or
price<minimum])
```

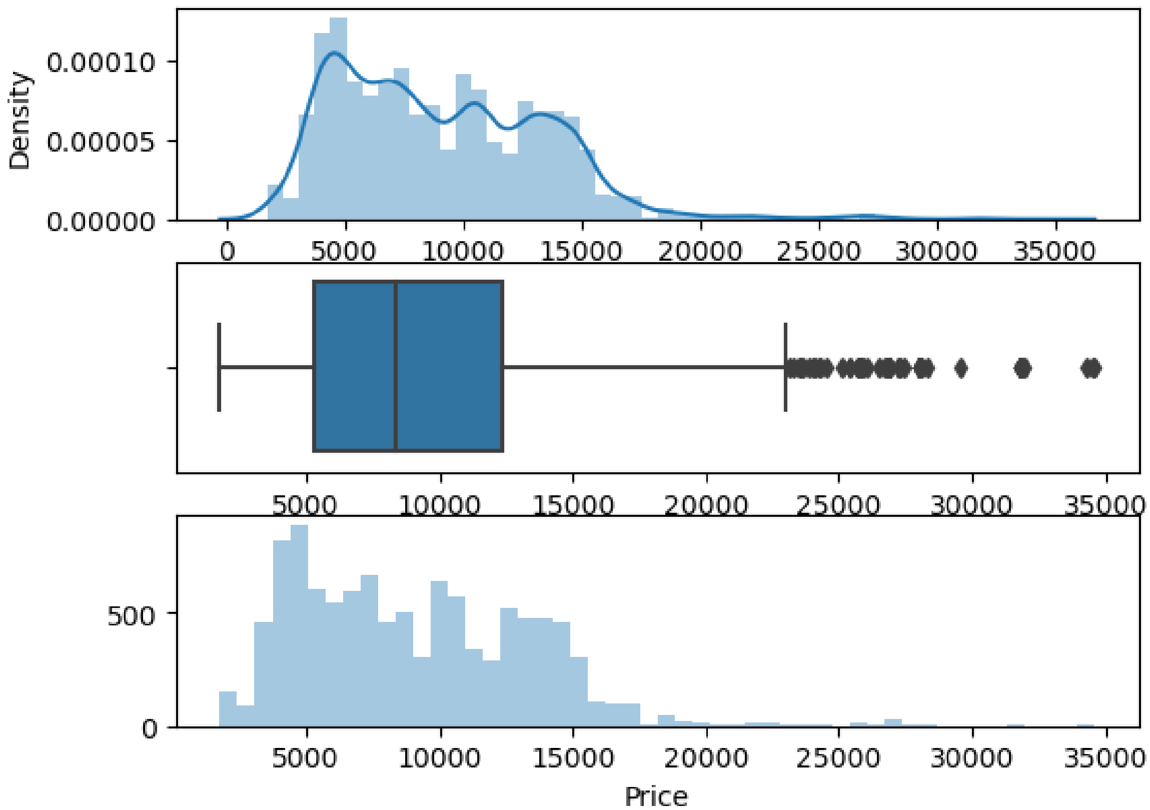
```
[27430, 36983, 26890, 26890, 25139, 27210, 52229, 26743, 26890, 25735,
27992, 26890, 26890, 23583, 26890, 23533, 24115, 25735, 54826, 31783,
27992, 26890, 26890, 25430, 36235, 27210, 26890, 25735, 54826, 26890,
35185, 79512, 28097, 27992, 26890, 25735, 26092, 31825, 25913, 25735,
27992, 31825, 23267, 62427, 54826, 31825, 25430, 26890, 36235, 23843,
26890, 25735, 28322, 25735, 25735, 31825, 26890, 27992, 34273, 46490,
29528, 26890, 26890, 26890, 34503, 26890, 27992, 26890, 26890, 23170,
24528, 26890, 27992, 25735, 34608, 25703, 26890, 23528, 31825, 27282,
25735, 27992, 52285, 24017, 31945, 26890, 24318, 23677, 27992, 24210,
57209, 26890, 31825, 26480]
```

```python
len([price for price in data['Price'] if price> maximum or
price<minimum])
```

```
94
```

b.. How to deal with Outlier

```
### wherever I have price >35K just replace replace it with median of
Price

data['Price'] = np.where(data['Price']>=35000 , data['Price'].median()
, data['Price'])

plot(data , 'Price')
```




```

```

13.. Lets Perform feature selection

```
'''
    : Feature Selection
    Finding out the best feature which will contribute and have good
relation with target variable.
```

```
    Q-> Why to apply Feature Selection?
    To select important features ie to get rid of curse of
dimensionality ie..or to get rid of duplicate features

'''

X = data.drop(['Price'] , axis=1)

y = data['Price']

from sklearn.feature_selection import mutual_info_regression

imp = mutual_info_regression(X , y)

'''
Estimate mutual information for a continuous target variable.

Mutual information between two random variables is a non-negative
value, which measures the dependency between the variables.
If It is equal to zero it means two random variables are independent,
and higher
values mean higher dependency.

'''

imp
```

```
array([0.97817067, 1.00276815, 0.78910531, 0.18819494, 0.24499169,
       0.33867287, 0.26424611, 0.40629992, 0.35573934, 0.46581843,
       0.344347  , 0.39219058, 0.46201314, 0.52335437, 0.1418939 ,
       0.19824146])
```

```
imp_df = pd.DataFrame(imp , index=X.columns)

imp_df.columns = ['importance']

imp_df
```

	importance
Airline	0.978171
Destination	1.002768
Total_Stops	0.789105
Journey_day	0.188195
Journey_month	0.244992
Dep_Time_hour	0.338673
Dep_Time_minute	0.264246
Arrival_Time_hour	0.406300
Arrival_Time_minute	0.355739
Duration_hours	0.465818
Duration_mins	0.344347
Source_Banglore	0.392191

```
Source_Kolkata        0.462013
Source_Delhi          0.523354
Source_Chennai        0.141894
Source_Mumbai         0.198241
```

```
imp_df.sort_values(by='importance' , ascending=False)
```

	importance
Destination	1.002768
Airline	0.978171
Total_Stops	0.789105
Source_Delhi	0.523354
Duration_hours	0.465818
Source_Kolkata	0.462013
Arrival_Time_hour	0.406300
Source_Banglore	0.392191
Arrival_Time_minute	0.355739
Duration_mins	0.344347
Dep_Time_hour	0.338673
Dep_Time_minute	0.264246
Journey_month	0.244992
Source_Mumbai	0.198241
Journey_day	0.188195
Source_Chennai	0.141894

14.. Lets Build ML model

split dataset into train & test

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.25, random_state=42)
```

what we often do in modelling:

```
a..Initially ,lets build basic random model.
b..then later-on , we will try to improve this model using some
parameters..
c..Then we will try to improve it..
d..Then we will hyper-tune my model to get optimal value of parameters
in order to achieve optimal value of params..
```

```
from sklearn.ensemble import RandomForestRegressor

ml_model = RandomForestRegressor()

ml_model.fit(X_train , y_train)

RandomForestRegressor()


y_pred = ml_model.predict(X_test)

y_pred

array([16744.87,  6291.59,  8840.03, ...,  3528.6 ,  6461.49,
6785.11])


from sklearn import metrics

metrics.r2_score(y_test , y_pred)

0.8061777476681846
```

b.. Lets Save model

lets try to dump ml model using pickle or joblib..

```
advantage of dumping--
imagine in future we have new data & lets say we have to predict price
on this huge data

then to do prediction on this new data , we can use this pre-trained
model what we have dumped..

!pip install pickle

import pickle

# open a file, where you want to store the data
file = open(r'Z:\Flight_Price\Datasets/rf_random.pkl' , 'wb')

# dump information to that file
pickle.dump(ml_model , file)


model = open(r'Z:\Flight_Price\Datasets/rf_random.pkl' , 'rb')
```

```
forest = pickle.load(model)

y_pred2 = forest.predict(X_test)

metrics.r2_score(y_test , y_pred2)

0.8061777476681846
```

15.. How to automate ml pipeline & How to define your Evaluation metric..

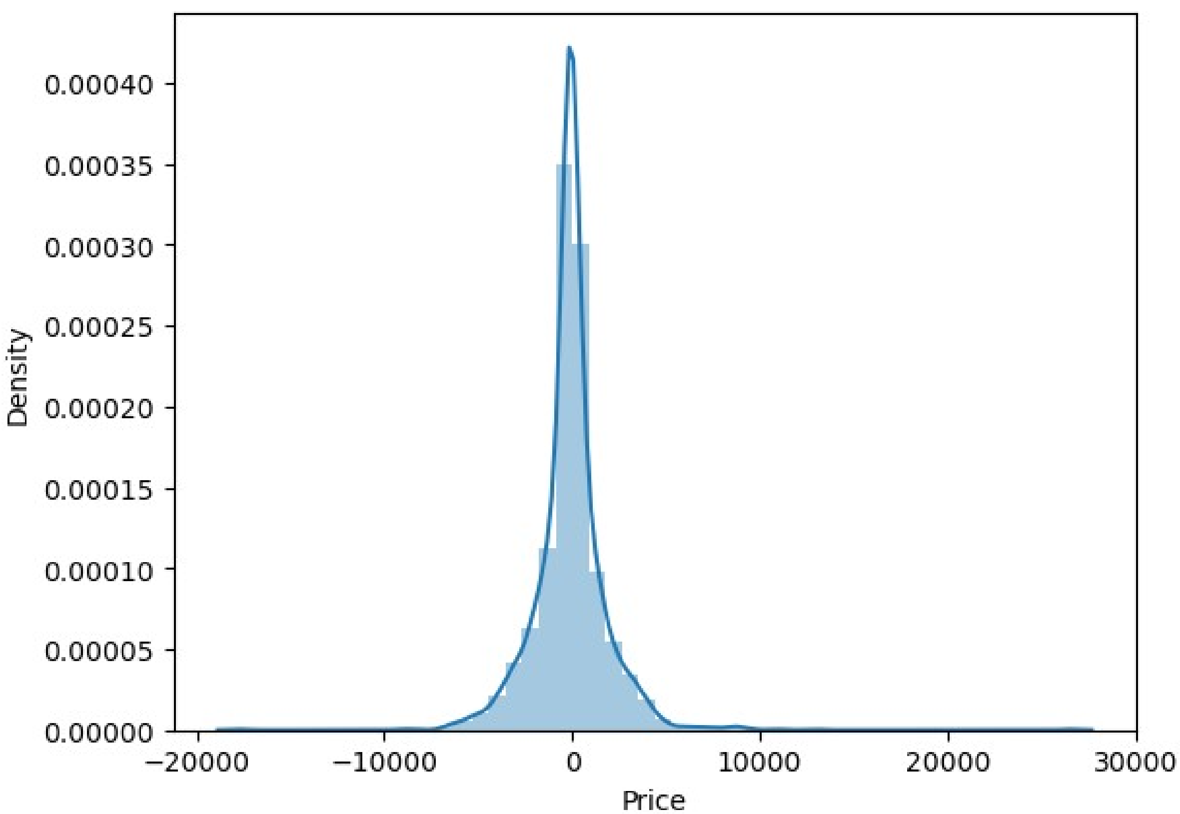
a.. how to make our own metric...

```
def mape(y_true , y_pred):
    y_true , y_pred = np.array(y_true) , np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

mape(y_test , y_pred)

13.247647518053313
```

b.. How to automate ml pipeline !

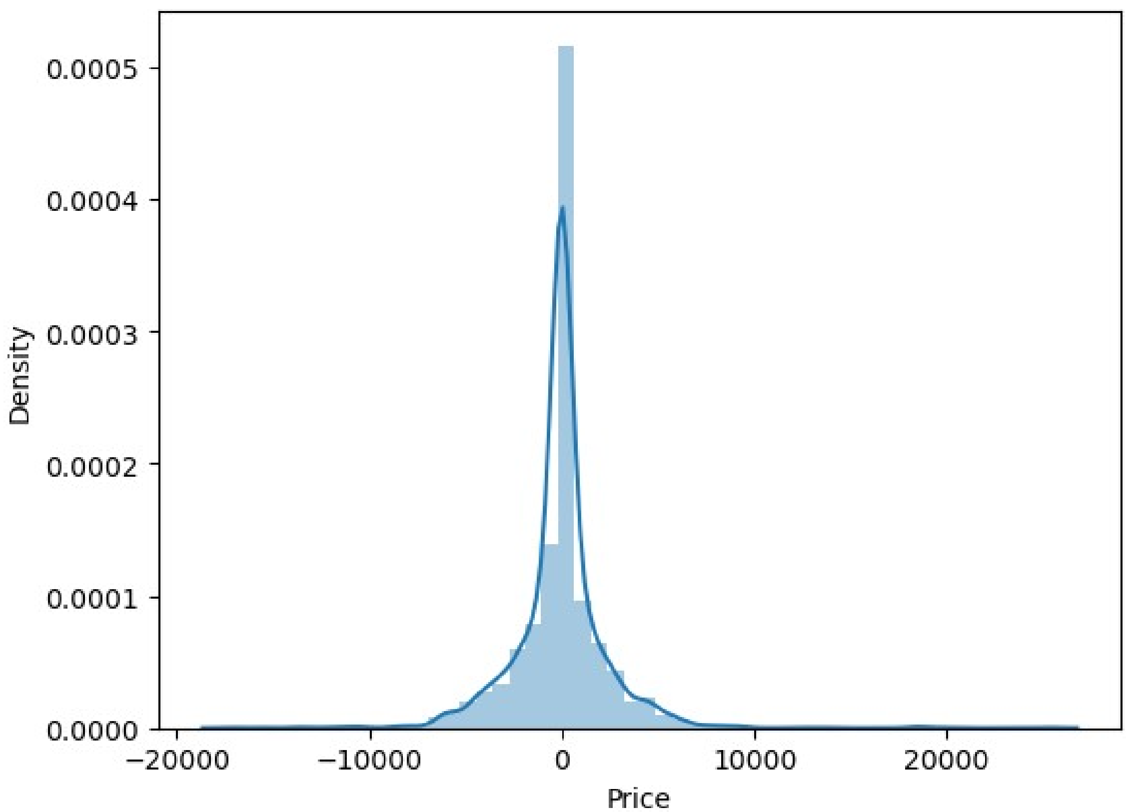
```
'''

    Lets automate all the stuffs..
    let say ,I will just pass ml algo & i get several results like--

    Training score, predictions, r2_score, mse, mae, rmse,
mape,distribution of error


'''

from sklearn import metrics

def predict(ml_model):
    model = ml_model.fit(X_train , y_train)
    print('Training score : {}'.format(model.score(X_train ,
y_train)))
    y_predection = model.predict(X_test)
```

```
    print('predictions are : {}'.format(y_predection))
    print('\n')
    r2_score = metrics.r2_score(y_test , y_predection)
    print('r2 score : {}'.format(r2_score))
    print('MAE : {}'.format(metrics.mean_absolute_error(y_test ,
y_predection)))
    print('MSE : {}'.format(metrics.mean_squared_error(y_test ,
y_predection)))
    print('RMSE : {}'.format(np.sqrt(metrics.mean_squared_error(y_test
, y_predection))))
    print('MAPE : {}'.format(mape(y_test , y_predection)))
    sns.distplot(y_test - y_predection)

predict(RandomForestRegressor())

Training score : 0.9512447050359809
predictions are : [16753.62  6414.15  8879.07 ...  3527.02  6268.23
6908.65]


r2 score : 0.8081762295335504
MAE : 1186.4927309208845
MSE : 3734348.3697542027
RMSE : 1932.4462139356435
MAPE : 13.304873080407178
```



```python
from sklearn.tree import DecisionTreeRegressor

predict(DecisionTreeRegressor())
```

```
Training score : 0.966591628243878
predictions are : [16840.  6976.  8610. ...  3419.  5797.  6818.]


r2 score : 0.6996399141630966
MAE : 1368.7372394858357
MSE : 5847289.90654707
RMSE : 2418.1170167192217
MAPE : 15.205975817573686
```



16.. how to hypertune ml model

```
## how to select which ML algo we should apply for
## ans is use Multiple Algos,then go for Hyper-parameter
Optimization,then for Cross Validation then go for various metrics
## & based on domain expertise knowledge Then I can say ya this model
perfoms best
```

Hyperparameter Tuning or Hyperparameter Optimization

```
1.Choose following method for hyperparameter tuning
    a.RandomizedSearchCV --> Fast way to Hypertune model
    b.GridSearchCV--> Slower way to hypertune my model
2.Choose ML algo that u have to hypertune
2.Assign hyperparameters in form of dictionary or create hyper-
parameter space
3.define searching &  apply searching on Training data or  Fit the CV
model
4.Check best parameters and best score
```

```
from sklearn.model_selection import RandomizedSearchCV
```

```
### initialise your estimator
reg_rf = RandomForestRegressor()
```

```
np.linspace(start =100 , stop=1200 , num=6)
```

```
array([ 100.,  320.,  540.,  760.,  980., 1200.])
```

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start =100 , stop=1200 ,
num=6)]

# Number of features to consider at every split
max_features = ["auto", "sqrt"]

# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(start =5 , stop=30 , num=4)]

# Minimum number of samples required to split a node
min_samples_split = [5,10,15,100]

# Create the random grid or hyper-parameter space

random_grid = {
    'n_estimators' : n_estimators ,
    'max_features' : max_features ,
    'max_depth' : max_depth ,
    'min_samples_split' : min_samples_split
}
```

```
random_grid
```

```
{'n_estimators': [100, 320, 540, 760, 980, 1200],
 'max_features': ['auto', 'sqrt'],
 'max_depth': [5, 13, 21, 30],
 'min_samples_split': [5, 10, 15, 100]}
```

```
## Define searching

# Random search of parameters, using 3 fold cross validation
# search across 576 different combinations


rf_random = RandomizedSearchCV(estimator=reg_rf ,
param_distributions=random_grid , cv=3 , n_jobs=-1 , verbose=2)

rf_random.fit(X_train , y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_jobs=-1,
                   param_distributions={'max_depth': [5, 13, 21, 30],
                                        'max_features': ['auto',
'sqrt'],
                                        'min_samples_split': [5, 10,
15, 100],
                                        'n_estimators': [100, 320,
540, 760,
                                                         980, 1200]},
                   verbose=2)
```

```
rf_random.best_params_
```

```
{'n_estimators': 760,
 'min_samples_split': 5,
 'max_features': 'auto',
 'max_depth': 13}
```

```
#### In your case , may be your parameters may vary a little bit ,
thats not a major issue..

rf_random.best_estimator_
```

```
RandomForestRegressor(max_depth=13, min_samples_split=5,
n_estimators=760)
```

```
rf_random.best_score_
```

```
0.821484460770345
```