



# Flight: Price Prediction

Submitted by:  
Dipak Someshwar

# **ACKNOWLEDGMENT**

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend my sincere thanks to SME. Sapna Verma.

We are highly indebted to Flip Robo technology for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I thank and appreciations also go to our colleague in developing the project and people who have willingly helped us out with their abilities.

Thanks all.

Dipak Someshwar

# INTRODUCTION

- Anyone who has booked a flight ticket knows how unexpectedly the prices vary.
- The cheapest available ticket on a given flight gets more and less expensive over time.
- This usually happens as an attempt to maximize revenue based on -
  1. Time of purchase patterns (making sure last-minute purchases are expensive)
  2. Keeping the flight as full as they want it (raising prices on a flight which is filling up in order to reduce sales and hold back inventory for those expensive last-minute expensive purchases)
- 1. So, you have to work on a project where you collect data of flight fares with other features and work to make a model to predict fares of flights.

## **1. Data Collection Phase:**

In this section scrape the data of flights from different websites (yatra.com, skyscanner.com, official websites of airlines, etc).

The number of columns for data doesn't have limit, it's up to you and your creativity.

Generally, these columns are airline name, date of journey, source, destination, route, departure time, arrival time, duration, total stops and the target variable price.

You can make changes to it, you can add or you can remove some columns, it completely depends on the website from which you are fetching the data.

## **2. Data Analysis :**

After cleaning the data, you have to do some analysis on the data.

Do airfares change frequently?

Do they move in small increments or in large jumps?

Do they tend to go up or down over time? What is the best time to buy so that the consumer can save the most by taking the least risk?

Does price increase as we get near to departure date?

Is Indigo cheaper than Jet Airways?

Are morning flights expensive?

### **3. Model Building :**

After collecting the data, build a machine learning model. Before model building do all data pre-processing steps.

Try different models with different hyper parameters and select the best model.

1. Data Cleaning
2. Exploratory Data Analysis
3. Data Pre-processing
4. Model Building
5. Model Evaluation
6. Selecting the best model

# Analytical Problem Framing

- Import library and load the dataset.

## Import the libraries.

```
In [1]: 1 import numpy as np
        2 import matplotlib.pyplot as plt
        3 import seaborn as sns
        4 import warnings
        5 warnings.filterwarnings('ignore')
```

## Load the dataset.

```
In [2]: 1 import pandas as pd
        2 df = pd.read_excel(r'flight_data_final.xlsx')
        3 df
```

```
Out[2]:
```

	Airline_name	Date_of_Journey	Source	Destination	Departure_time	Arrival_time	Duration	Total_stops	Price
0	Jet Airways	9/05/2022	Kolkata	Banglore	09:35	10:55:00	25h 20m	1 stop	10953
1	SpiceJet	3/04/2022	Kolkata	Banglore	17:10	19:40:00	2h 30m	non-stop	3841
2	IndiGo	1/06/2022	Delhi	Cochin	05:10	10:05:00	4h 55m	1 stop	6842
3	Jet Airways	27/05/2022	Delhi	Cochin	09:00	12:35:00	27h 35m	1 stop	12898
4	Vistara	7/09/2022	Mumbai	Bangalore	17:35	09:00	15h 25m	1 stop	9280
...	...	...	...	...	...	...	...	...	...
1530	SpiceJet	21/03/2022	Banglore	New Delhi	10:20	18:15:00	7h 55m	1 stop	7139
1531	Jet Airways	27/04/2022	Banglore	Delhi	06:00	08:45:00	2h 45m	non-stop	4544
1532	IndiGo	10/11/2022	Pune	Bangalore	22:20	23:55	1h 35m	non-stop	5418
1533	IndiGo	18/05/2022	Delhi	Cochin	08:35	16:10:00	7h 35m	1 stop	6442
1534	Air India	3/06/2022	Delhi	Cochin	03:50	19:15:00	15h 25m	1 stop	8669

1535 rows x 9 columns

```
In [3]: 1 # Get the numbers of rows and columns.
        2 df.shape
```

```
Out[3]: (1535, 9)
```

- Display all column name of dataset.

```
In [4]: 1 # Check column of the dataframe.  
        2 df.columns  
  
Out[4]: Index(['Airline_name', 'Date_of_Journey', 'Source', 'Destination',  
              'Departure_time', 'Arrival_time', 'Duration', 'Total_stops', 'Price'],  
             dtype='object')
```

- Display datatypes and sum of null values.

```
In [5]: 1 # Get the column datatypes.  
        2 df.dtypes
```

```
Out[5]: Airline_name    object  
        Date_of_Journey object  
        Source          object  
        Destination     object  
        Departure_time  object  
        Arrival_time    object  
        Duration        object  
        Total_stops     object  
        Price           int64  
        dtype: object
```

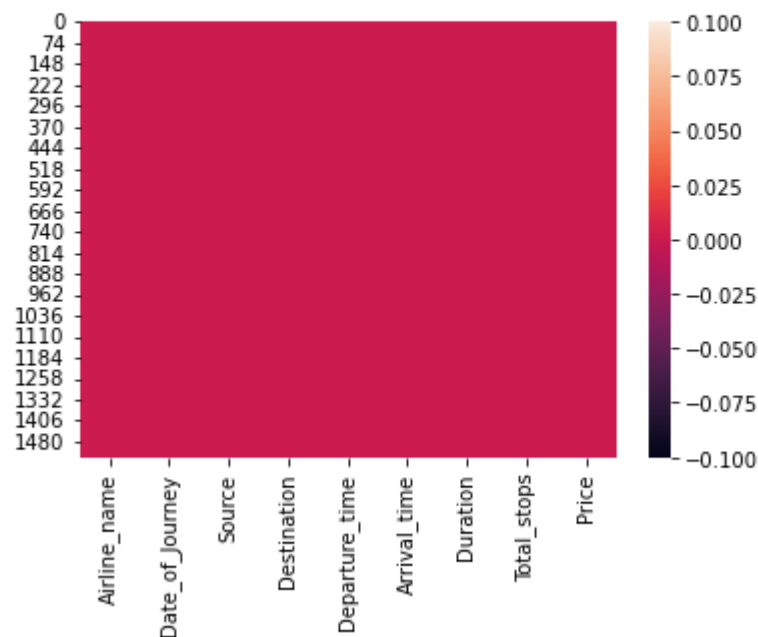
```
In [7]: 1 # Get a count of the empty values for each column.  
        2 df.isna().sum()
```

```
Out[7]: Airline_name    0  
        Date_of_Journey 0  
        Source          0  
        Destination     0  
        Departure_time  0  
        Arrival_time    0  
        Duration        0  
        Total_stops     0  
        Price           0  
        dtype: int64
```

- Display null values of columns using heatmap.

```
In [9]: 1 #Checking for null values using heatmap.
        2 sns.heatmap(df.isnull())
```

Out[9]: <AxesSubplot:>



- Display statistical summary.

```
In [16]: 1 # Summary statistics:
        2 df.describe().style.background_gradient()
```

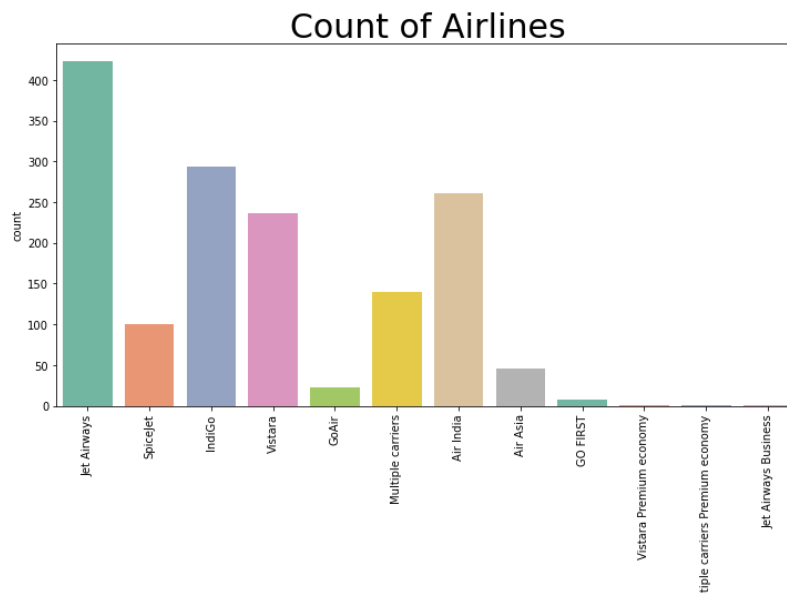
Out[16]:

	Price
count	1533.000000
mean	9229.936725
std	4485.641009
min	1965.000000
25%	5418.000000
50%	9211.000000
75%	11789.000000
max	52229.000000



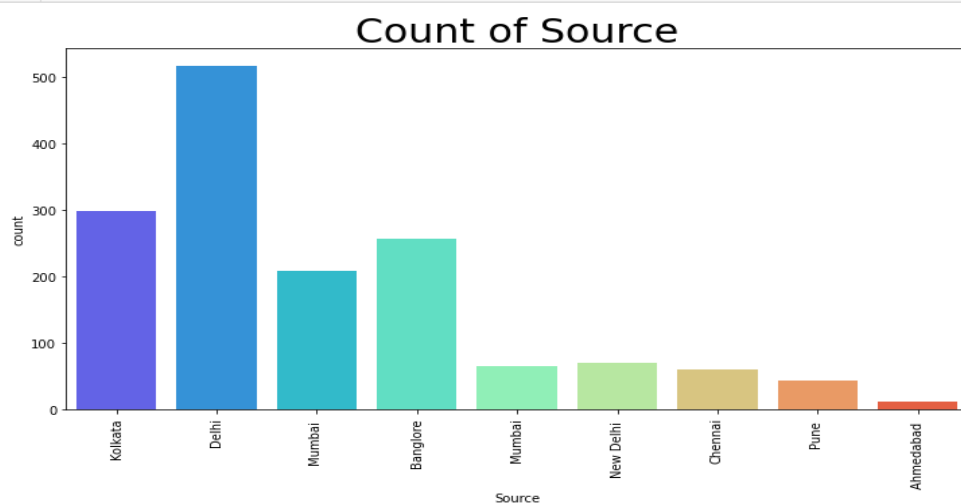
- Display count of Airline column.

```
In [17]: 1 # Here plotting Count of Airlines :
2 plt.figure(figsize=(12,6))
3 sns.countplot(df['Airline_name'], palette='Set2')
4 plt.title('Count of Airlines', size=30)
5 plt.xticks(rotation=90)
6 plt.show()
```



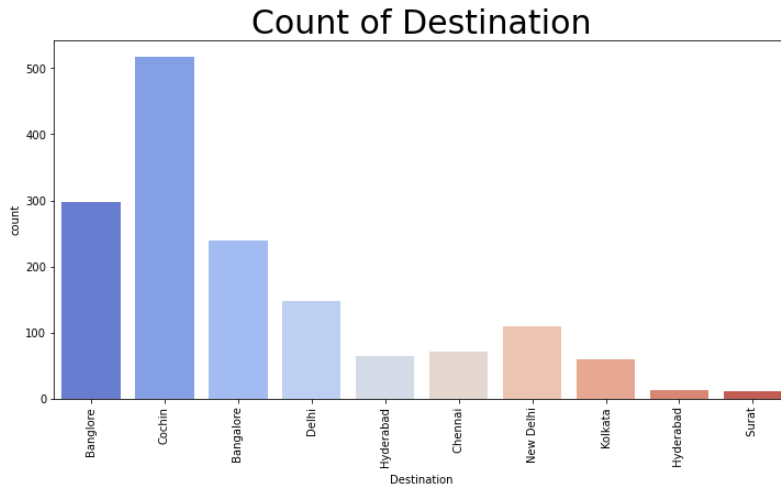
- Display count of Source column.

```
In [19]: 1 # Here plotting Count of Source :
2 plt.figure(figsize=(12,6))
3 sns.countplot(df['Source'], palette='rainbow')
4 plt.title('Count of Source', size=30)
5 plt.xticks(rotation=90)
6 plt.show()
```



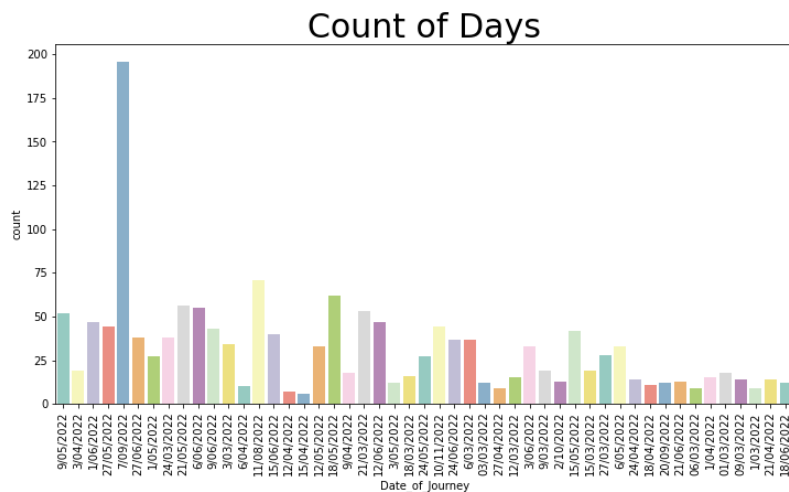
- Display count of Destination column.

```
In [20]: 1 # Here plotting Count of Source :
2 plt.figure(figsize=(12,6))
3 sns.countplot(df['Destination'], palette='coolwarm')
4 plt.title('Count of Destination', size=30)
5 plt.xticks(rotation=90)
6 plt.show()
```



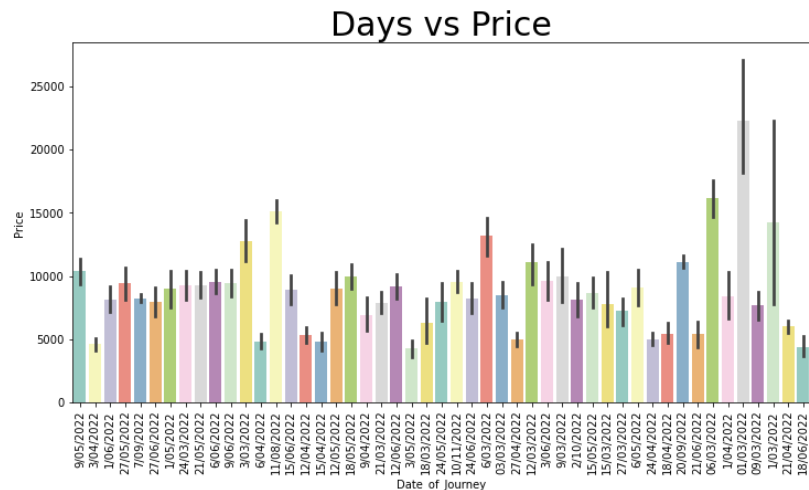
- Display count of Days.

```
In [21]: 1 # Here plotting Count of Days :
2 plt.figure(figsize=(12,6))
3 sns.countplot(df['Date_of_Journey'], palette='Set3')
4 plt.title('Count of Days', size=30)
5 plt.xticks(rotation=90)
6 plt.show()
```



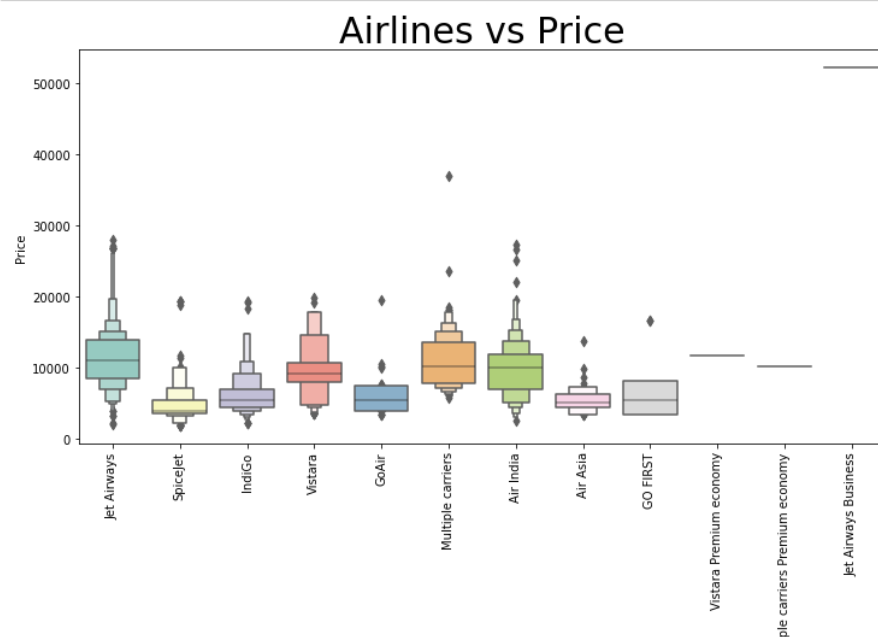
- Display Days vs Price.

```
In [22]: 1 # Here Plotting days vs price plot:
2 plt.figure(figsize=(12,6))
3 sns.barplot(df['Date_of_Journey'], df['Price'], palette='Set3')
4 plt.title('Days vs Price', size=30)
5 plt.xticks(rotation=90)
6 plt.show()
```



- Display Airline vs Price.

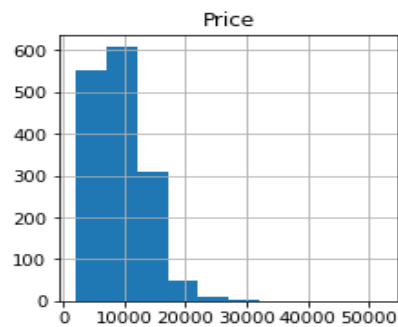
```
In [23]: 1 # Here Plotting Price vs Airline plot:
2 plt.figure(figsize=(12,6))
3 sns.boxplot(df['Airline_name'], df['Price'], palette='Set3')
4 plt.title('Airlines vs Price', size=30)
5 plt.xticks(rotation=90)
6 plt.show()
```



- Display Histogram of Price.

```
In [24]: 1 # display histogram:
2 df.hist(figsize=(12,12), layout=(3,3), sharex=False)

Out[24]: array([[<AxesSubplot:title={ 'center': 'Price' }>, <AxesSubplot:>,
<AxesSubplot:>],
[<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>],
[<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```



- Perform Feature Engineering.

#### Feature Engineering

```
In [30]: 1 # Here converting the hours in minutes:
2 df['Duration'] = df['Duration'].str.replace("h", '*60').str.replace(' ', '+').str.replace('m','*1').apply(eval)
```

```
In [32]: 1 # Here we are organizing the format of the date of journey in our dataset for better:
2 df["Journey_day"] = df['Date_of_Journey'].str.split('/').str[0].astype(int)
3 df["Journey_month"] = df['Date_of_Journey'].str.split('/').str[1].astype(int)
4 df.drop(["Date_of_Journey"], axis = 1, inplace = True)
```

```
In [34]: 1 # Converting departure time into hours and minutes:
2 df["Dep_hour"] = pd.to_datetime(df["Departure_time"]).dt.hour
3 df["Dep_min"] = pd.to_datetime(df["Departure_time"]).dt.minute
4 df.drop(["Departure_time"], axis = 1, inplace = True)
```

```
In [35]: 1 # Here converting the arrival time into hours and minutes:
2 df["Arrival_hour"] = pd.to_datetime(df.Arrival_time).dt.hour
3 df["Arrival_min"] = pd.to_datetime(df.Arrival_time).dt.minute
4 df.drop(["Arrival_time"], axis = 1, inplace = True)
```

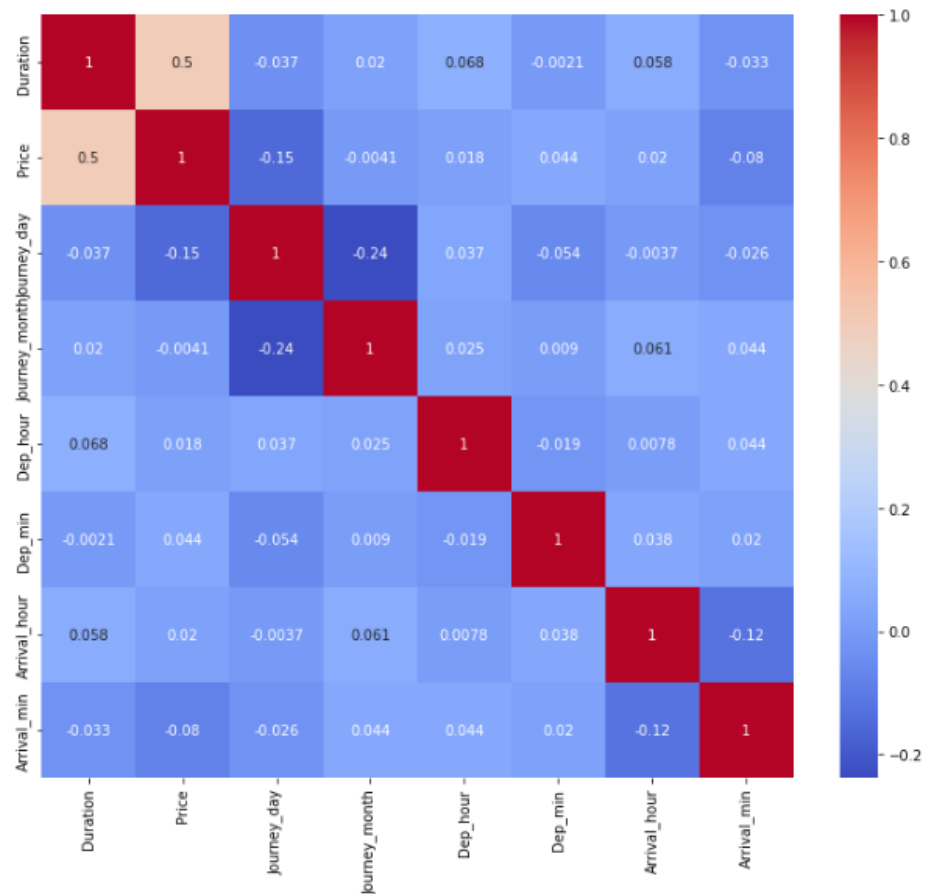
```
In [36]: 1 # Showing dataframe after performing feature engineering.
2 df.head()
```

```
Out[36]:
```

	Airline_name	Source	Destination	Duration	Total_stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min
0	Jet Airways	Kolkata	Banglore	1520	1 stop	10953	9	5	9	35	10	55
1	SpiceJet	Kolkata	Banglore	150	non-stop	3841	3	4	17	10	19	40
2	IndiGo	Delhi	Cochin	295	1 stop	6842	1	6	5	10	10	5
3	Jet Airways	Delhi	Cochin	1655	1 stop	12898	27	5	9	0	12	35
4	Vistara	Mumbai	Bangalore	925	1 stop	9280	7	9	17	35	9	0

- Display correlation of columns using heatmap.

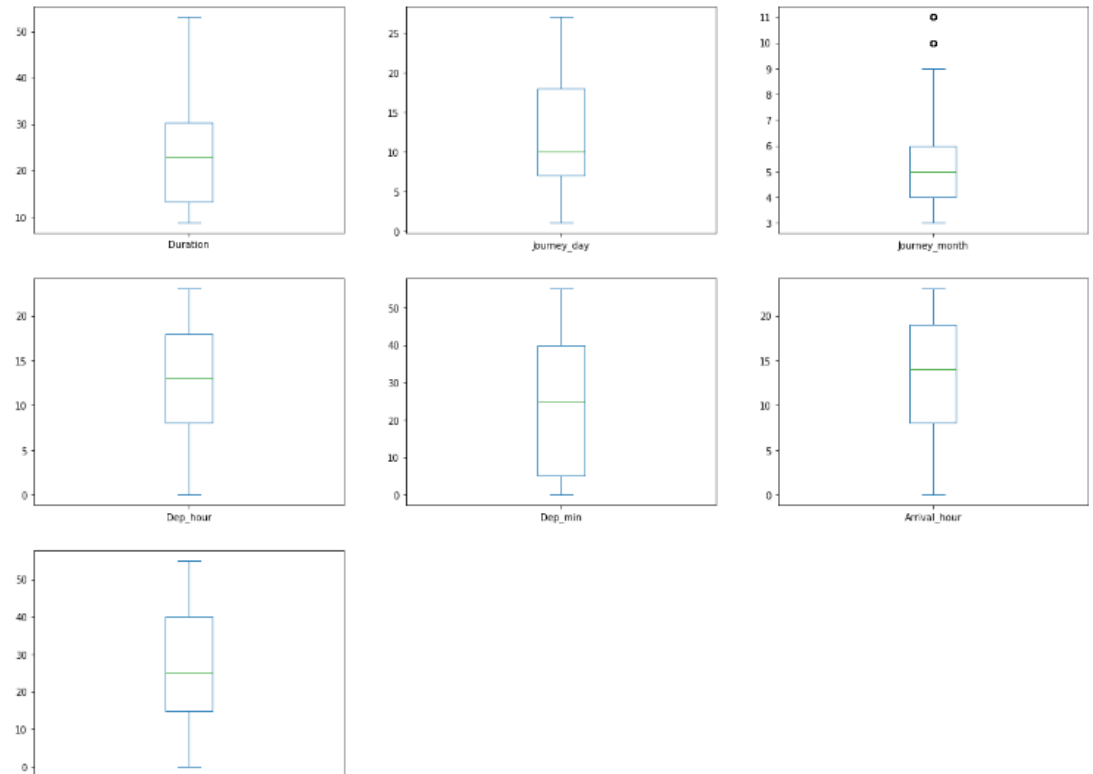
```
In [38]: 1 plt.figure(figsize = (12,10))
2          sns.heatmap(df.corr(), annot = True, cmap = "coolwarm")
3          plt.show()
```



- Display outliers of all columns.

```
In [48]: 1 # Checking outliers with boxplot:
          2 numerical_data.plot(kind='box', subplots=True, layout=(3,3), figsize=(20,15))
```

```
Out[48]: Duration      AxesSubplot(0.125,0.657941;0.227941x0.222059)
Journey_day  AxesSubplot(0.398529,0.657941;0.227941x0.222059)
Journey_month AxesSubplot(0.672059,0.657941;0.227941x0.222059)
Dep_hour     AxesSubplot(0.125,0.391471;0.227941x0.222059)
Dep_min      AxesSubplot(0.398529,0.391471;0.227941x0.222059)
Arrival_hour AxesSubplot(0.672059,0.391471;0.227941x0.222059)
Arrival_min  AxesSubplot(0.125,0.125;0.227941x0.222059)
dtype: object
```



- Data Pre-processing and Scaling the data.

#### Data Preprocessing (split the data into independent 'x' and dependent 'y' datasets)

```
In [52]: 1 # Here Concatenating both Categorical Data and Numerical Data:
2 x = pd.concat([categorical_data, numerical_data], axis=1)
3 y = df['Price']
```

```
In [54]: 1 x.head()
```

```
Out[54]:
```

	Airline_name	Source	Destination	Total_stops	Duration	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	
0		5	4	4	0	38.987177	9	5	9	35	10	55
1		9	4	4	3	12.247449	3	4	17	10	19	40
2		4	3	5	0	17.175564	1	6	5	10	10	5
3		5	3	5	0	40.681691	27	5	9	0	12	35
4		10	6	0	0	30.413813	7	9	17	35	9	0

```
In [55]: 1 y.head()
```

```
Out[55]: 0    10953
1     3841
2     6842
3    12898
4     9280
Name: Price, dtype: int64
```

```
In [56]: 1 print(x.shape, y.shape)
```

```
(1533, 11) (1533,)
```

#### Scaling the data.

```
In [57]: 1 from sklearn.preprocessing import StandardScaler
2
3 st = StandardScaler()
4 x = st.fit_transform(x)
5 x
```

```
Out[57]: array([[ -0.05204319,  0.20716433, -0.15155037, ...,  0.54456632,
-0.50335041,  1.72537747],
[ 1.27766042,  0.20716433, -0.15155037, ..., -0.78265175,
 0.81543531,  0.84125368],
[ -0.3844691 , -0.33111118,  0.24558931, ..., -0.78265175,
-0.50335041, -1.22170182],
...,
[ -0.3844691 ,  2.36026881, -1.74010908, ..., -0.25176452,
 1.4015623 ,  1.72537747],
[ -0.3844691 , -0.33111118 ,  0.24558931, ...,  0.54456632,
 0.37584007, -0.92699389],
[ -1.38174681, -0.33111118 ,  0.24558931, ...,  1.34089716,
 0.81543531, -0.63228596]])
```

- Run and evaluate selected models.

#### Finding best random\_state

```
In [58]: 1 from sklearn.linear_model import LinearRegression
2 lr = LinearRegression()
3
4 from sklearn.ensemble import RandomForestRegressor
5 rf = RandomForestRegressor()
6
7 from sklearn.ensemble import AdaBoostRegressor
8 abr = AdaBoostRegressor()
9
10 from sklearn.ensemble import GradientBoostingRegressor
11 gbr = GradientBoostingRegressor()
12
13 from sklearn.tree import DecisionTreeRegressor
14 dtr = DecisionTreeRegressor()
15
16 from sklearn.metrics import r2_score
17 from sklearn.model_selection import train_test_split
```

```
In [59]: 1 model = [lr,rf,abr,gbr,dtr]
2 max_r2_score = 0
3 for r_state in range(0,100):
4     x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=.20,random_state = r_state)
5     for i in model:
6         i.fit(x_train,y_train)
7         pred_test = i.predict(x_test)
8         r2_sc = r2_score(y_test,pred_test)
9         print("R2 score correspond to random state ",r_state,"is",r2_sc)
10        if r2_sc > max_r2_score:
11            max_r2_score = r2_sc
12            final_state = r_state
13            final_model = i
```

```
R2 score correspond to random state 9 is 0.6196406794640958
R2 score correspond to random state 9 is 0.3822601727419733
R2 score correspond to random state 10 is 0.44376074740412497
R2 score correspond to random state 10 is 0.7877210630853698
R2 score correspond to random state 10 is 0.4924833375519503
R2 score correspond to random state 10 is 0.7700066152629447
R2 score correspond to random state 10 is 0.5325174298757969
R2 score correspond to random state 11 is 0.46852978946636037
R2 score correspond to random state 11 is 0.7388002704805672
R2 score correspond to random state 11 is 0.5397650556851779
R2 score correspond to random state 11 is 0.7145815649700709
R2 score correspond to random state 11 is 0.43000474467977345
R2 score correspond to random state 12 is 0.39447416751259723
R2 score correspond to random state 12 is 0.8061866381796754
R2 score correspond to random state 12 is 0.5736027845522467
R2 score correspond to random state 12 is 0.709091996036882
R2 score correspond to random state 12 is 0.5410057640727091
R2 score correspond to random state 13 is 0.4178068600662399
R2 score correspond to random state 13 is 0.7345934403372618
R2 score correspond to random state 13 is 0.5151725353435027
```

```
In [60]: 1 print("max R2 score correspond to random state ",final_state,"is",max_r2_score,"and model is",final_model)

max R2 score correspond to random state 79 is 0.8307051042076956 and model is RandomForestRegressor()
```



## Creating train-test split:

```
In [61]: 1 x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=.20,random_state = 79)
```

## Apply best model:

```
In [62]: 1 rf.fit(x_train,y_train)
2 pred = rf.predict(x_test)
3 print("r2 score :- ",r2_score(y_test,pred))
```

r2 score :- 0.8297931346192062

```
In [63]: 1 # Predict the value:
2 print("predicted ticket price:",pred)
3 print("actual ticket price",y_test)
```

predicted ticket price: [ 3180.86            11339.54            13469.22            5187.15			
12701.6	7005.8	8448.695	14926.715
2646.035	10070.	10492.43	14472.525
4537.64	13219.72	6786.39	8555.87
4462.9825	6542.58	16606.37	7713.72866667
5887.95	11988.77	11428.82	9268.68
3846.22	11391.27	10078.43	4627.54666667
9282.75	3821.58	11603.04714286	11066.12833333
8357.63	6246.35	12048.24	3866.035
13268.71	4445.75	9446.03	9280.
8074.42	7543.49	10697.11	12118.7
16700.04	7618.94	11386.1	12011.775
13075.5275	11767.53	4083.69	6830.98
4426.09666667	4721.95	6553.71	12033.425
10242.44166667	4703.68	10128.16	12111.16166667
12271.13	13064.66	12244.82	9083.58166667
16895.73	6238.31	9596.8025	7911.72
4789.22	7558.76	12473.825	26440.47
10070.	4818.16	3881.24	9264.415
10319.865	4003.66	9280.	8953.49
4550.72	12229.85	10264.85	8291.75
5264.38	16749.48	4643.425	13714.39
7318.43	5211.4	9166.39	12738.16

- Checking MAE, MSE and RMSE:

## Check MAE, MSE and RMSE:

```
In [64]: 1 # Display MAE, MSE and RMSE:
2 from sklearn.metrics import mean_squared_error, mean_absolute_error
3 print('error:')
4
5 print('mean absolute error',mean_absolute_error(y_test,pred))
6 print('mean squared error',mean_squared_error(y_test,pred))
7
8 print('Root mean squared error',np.sqrt(mean_squared_error(y_test,pred)))
```

error:

mean absolute error 1240.9198928959206

mean squared error 3241536.9721175395

Root mean squared error 1800.426886079393

- Hypertuning of the Model.

### Hypertuning of the model:

```
In [67]: 1 from sklearn.model_selection import RandomizedSearchCV
```

```
In [68]: 1 #Randomized Search CV
2
3 # Number of trees in random forest
4 n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
5
6 # Number of features to consider at every split
7 max_features = ['auto', 'sqrt']
8
9 # Maximum number of levels in tree
10 max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
11
12 # Minimum number of samples required to split a node
13 min_samples_split = [2, 5, 10, 15, 100]
14
15 # Minimum number of samples required at each leaf node
16 min_samples_leaf = [1, 2, 5, 10]
```

```
In [69]: 1 parameters = {'n_estimators': n_estimators,
2               'max_features': max_features,
3               'max_depth': max_depth,
4               'min_samples_split': min_samples_split,
5               'min_samples_leaf': min_samples_leaf}
```

```
In [70]: 1 rf_random = RandomizedSearchCV(estimator = rf, param_distributions = parameters, scoring='neg_mean_squared_error', n_iter =
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```
In [71]: 1 rf_random.fit(x_train,y_train)

Fitting 4 folds for each of 10 candidates, totalling 40 fits
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time= 0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time= 0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time= 0.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time= 0.6s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time= 0.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time= 0.3s
```

```
In [72]: 1 rf_random.best_params_
```

```
Out[72]: {'n_estimators': 300,
          'min_samples_split': 2,
          'min_samples_leaf': 1,
          'max_features': 'sqrt',
          'max_depth': 30}
```

```
In [73]: 1 rf = RandomForestRegressor(n_estimators=300, min_samples_split=2, min_samples_leaf=1, max_features='sqrt', max_depth=30)
2 rf.fit(x_train,y_train)
3 rf.score(x_train,y_train)
4 pred_decision = rf.predict(x_test)
5
6 rfs = r2_score(y_test,pred_decision)
7 print("R2 score:",rfs*100)
8
9 rfscore = cross_val_score(rf,x,y,cv=4)
10 rfc = rfscore.mean()
11 print("Cross Val Score is",rfc*100)
```

R2 score: 81.46995701280109  
Cross Val Score is 71.72954936874227

- Hardware and Software Requirements and Tools Used

➤ **Language :-** Python

➤ **Tool:-** Jupyter

➤ **OS:-** Windows 10

➤ **RAM:-** 8gb

## CONCLUSION:

- This Kernel investigates different models for car price prediction.
- Different types of Machine Learning methods including LinearRegression, RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor and DecisionTreeRegressor in machine learning are compared and analysed for optimal solutions.
- Even though all of those methods achieved desirable results, different models have their own pros and cons.
- The RandomForestRegressor is probably the best one and has been selected for this problem.
- Finally, the RandomForestRegressor is the best choice when parameterization is the top priority.