



Car: 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

- With the covid 19 impact in the market, we have seen lot of changes in the car market.
- Now some cars are in demand hence making them costly and some are not in demand hence cheaper.
- One of our clients works with small traders, who sell used cars. With the change in market due to covid 19 impact, our client is facing problems with their previous car price valuation machine learning models.
- So, they are looking for new machine learning models from new data. We have to make car price valuation model.
- This project contains two phase:

1. Data Collection Phase:

In this section scrape the data of used cars from websites (Olx, cardekho, Cars24 etc.) and web scraping for this and fetch data for different locations.

The number of columns for data doesn't have limit, it's up to you and your creativity. Generally, these columns are Brand, model, variant, manufacturing year, driven kms, fuel, number of owners, location and at last target variable Price of the car.

2. Model Building Phase:

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'car_data.xlsx')
        3 df
```

Out[2]:

	Brand	Model	Variant	Transmission	Driven_KM	Fuel_type	No. of owners	Manufacture_year	Location	Price
0	Mahindra	XUV	1.2 W6 MT	NaN	9,510 km	Petrol	1st Owner	2021	Ahmedabad	₹10,67,499
1	Maruti	Alto	VXI Manual	Manual	8,339 km	Petrol	1st Owner	2016	Ahmedabad	₹3,37,599
2	Maruti	Ertiga	ZXI Plus SHVS Manual	Manual	21,596 km	Petrol	2nd Owner	2020	Ahmedabad	₹10,39,099
3	Maruti	Swift	LXI Manual	Manual	16,885 km	Petrol	1st Owner	2019	Ahmedabad	₹5,82,299
4	Volkswagen	Polo	Trendline 1.0 L Petrol Manual	Manual	14,469 km	Petrol	1st Owner	2020	Ahmedabad	₹5,82,999
...
5192	Maruti	Alto	LXI Manual	Manual	51,233 km	Petrol	1st Owner	2019	Kochi	₹3,16,399
5193	Maruti	Alto	VXI Manual	Manual	76,693 km	Petrol	1st Owner	2015	Kochi	₹2,56,899
5194	Maruti	Eeco	5 STR WITH AC PLUSHTR Manual	Manual	14,835 km	Petrol	1st Owner	2020	Kochi	₹4,43,999
5195	Hyundai	Grand	SPORTZ (O) 1.2 AT VTVT Automatic	Automatic	23,695 km	Petrol	2nd Owner	2017	Kochi	₹5,26,719
5196	Hyundai	Creta	1.6 SX CRDI Manual	Manual	54,729 km	Diesel	1st Owner	2018	Kochi	₹10,93,999

5197 rows x 10 columns

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

Out[3]: (5197, 10)

- Display all column name of dataset.

```
In [4]: 1 #Check column of the dataframe.
        2 df.columns

Out[4]: Index(['Brand', 'Model', 'Variant', 'Transmission', 'Driven_KM', 'Fuel_type',
              'No. of owners', 'Manufacture_year', 'Location', 'Price'],
              dtype='object')
```

- Display datatypes and sum of null values.

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

```
Out[5]: Brand          object
        Model          object
        Variant        object
        Transmission   object
        Driven_KM       object
        Fuel_type       object
        No. of owners   object
        Manufacture_year int64
        Location        object
        Price           object
        dtype: object
```

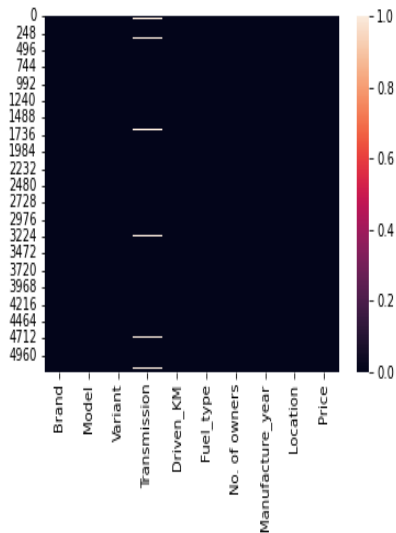
```
In [7]: 1 #Get a count of the empty values for each column.
        2 df.isna().sum()
```

```
Out[7]: Brand          0
        Model          0
        Variant        0
        Transmission   80
        Driven_KM       0
        Fuel_type       0
        No. of owners   0
        Manufacture_year 0
        Location        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.

Data Analysis and Visualization

```
In [17]: 1 #summary statistics.
        2 df.describe().style.background_gradient()
```

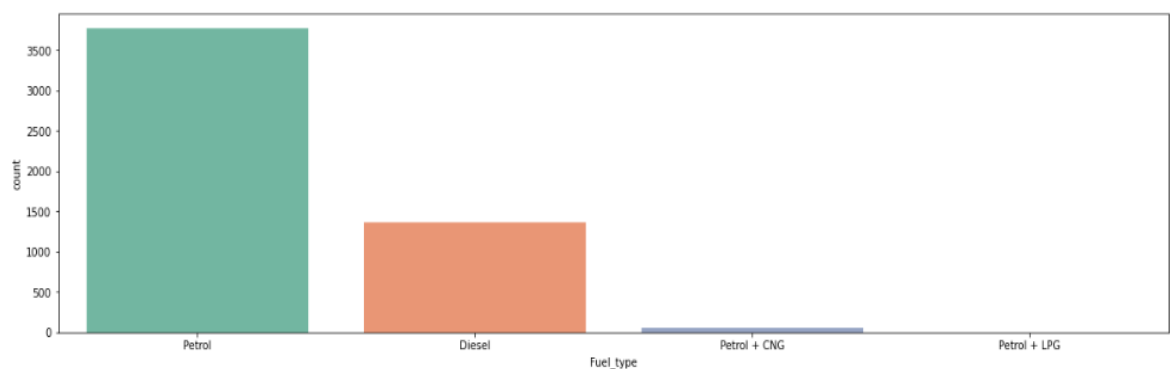
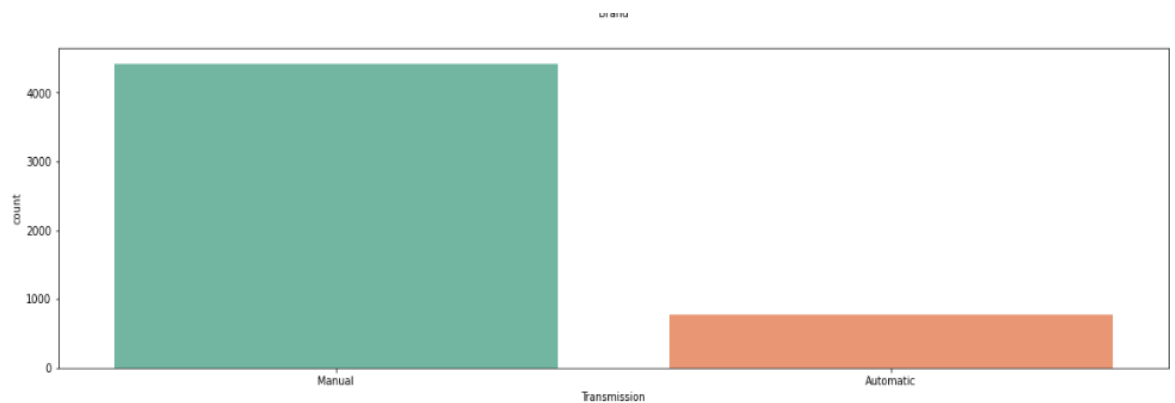
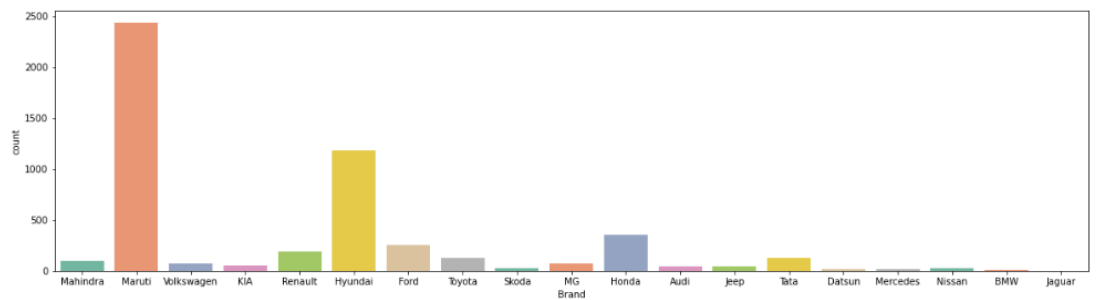
Out[17]:

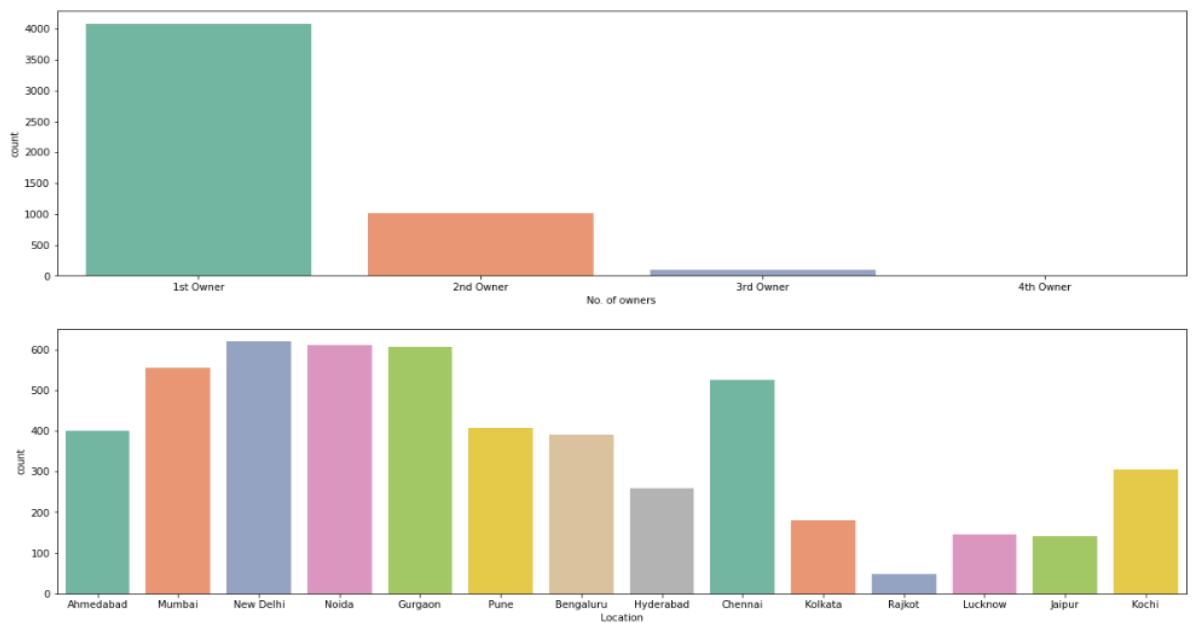
	Driven_KM	Manufacture_year	Price
count	5197.000000	5197.000000	5197.000000
mean	43790.950741	2017.169906	648476.928420
std	31013.177890	2.413196	339002.018013
min	71.000000	2008.000000	143799.000000
25%	21473.000000	2016.000000	420299.000000
50%	38225.000000	2017.000000	554699.000000
75%	60804.000000	2019.000000	765099.000000
max	400055.000000	2022.000000	3287199.000000

- Display countplot of columns.

```
In [19]: 1 #Visualize the number of other independent categorical variables.
2 fig,ax = plt.subplots(5,1,figsize=(20,30))
3 sns.countplot(df['Brand'], data=df, palette='Set2',ax=ax[0])
4 sns.countplot(df['Transmission'], data=df, palette='Set2',ax=ax[1])
5 sns.countplot(df['Fuel_type'], data=df, palette='Set2',ax=ax[2])
6 sns.countplot(df['No. of owners'], data=df, palette='Set2',ax=ax[3])
7 sns.countplot(df['Location'], data=df, palette='Set2',ax=ax[4])
```

Out[19]: <AxesSubplot:xlabel='Location', ylabel='count'>

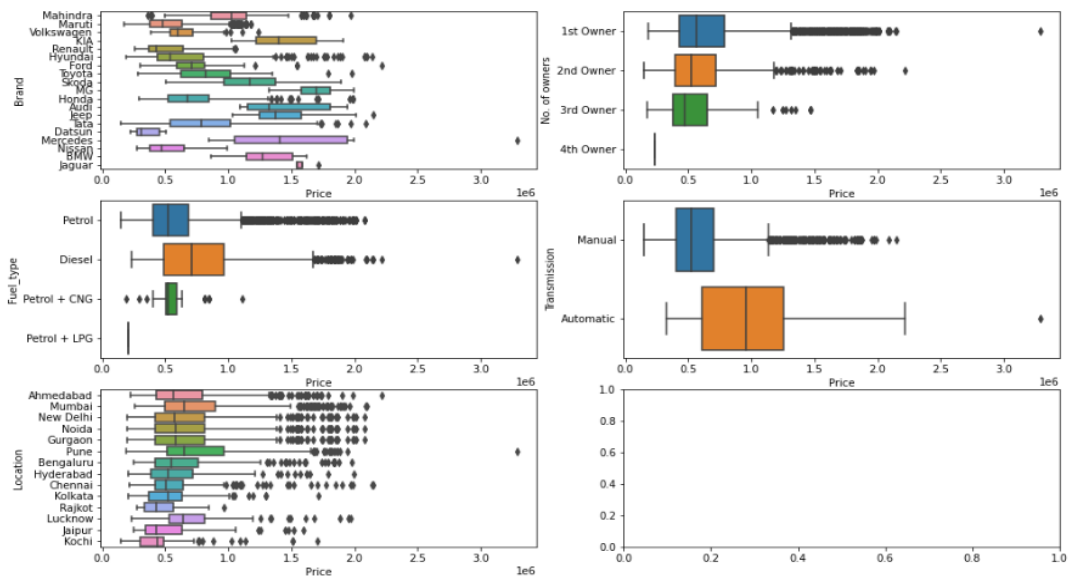




- Display boxplot of columns to compare with price.

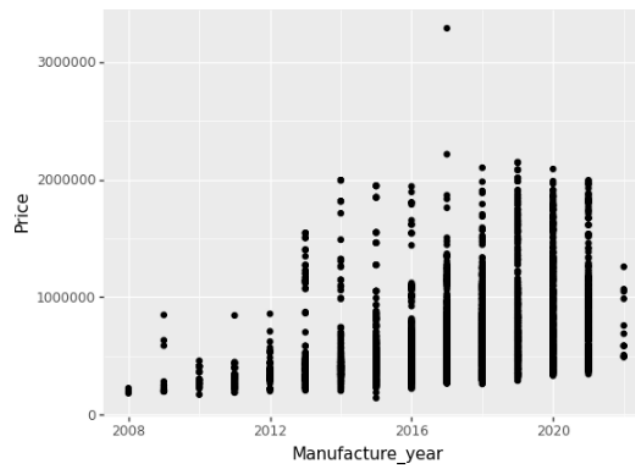
```
In [20]: 1 # display boxplot for compare price.
2 fig,ax = plt.subplots(3,2,figsize=(16,10))
3 sns.boxplot(x='Price',y='Brand',data=df,ax=ax[0][0])
4 sns.boxplot(x='Price',y='No. of owners',data=df,ax=ax[0][1])
5 sns.boxplot(x='Price',y='Fuel type',data=df,ax=ax[1][0])
6 sns.boxplot(x='Price',y='Transmission',data=df,ax=ax[1][1])
7 sns.boxplot(x='Price',y='Location',data=df,ax=ax[2][0])
```

Out[20]: <AxesSubplot: xlabel='Price', ylabel='Location'>



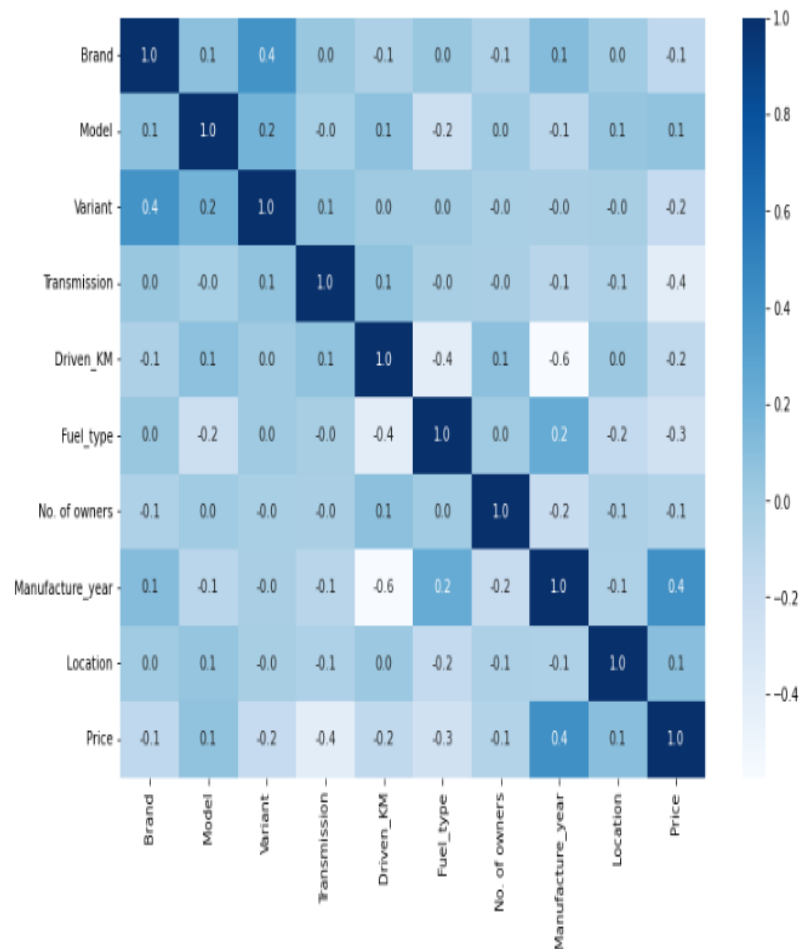
- Display plotline of manuf. year column to compare with price.

```
In [22]: 1 from plotnine.data import mpg
          2 from plotnine import ggplot, aes, geom_point
          3
          4 (
          5     ggplot(df) # what data to use
          6     + aes(x="Manufacture_year", y="Price") # what variable to use
          7     + geom_point() # Geometric object to use for drawing
          8 )
```



- Display correlation of columns using heatmap.

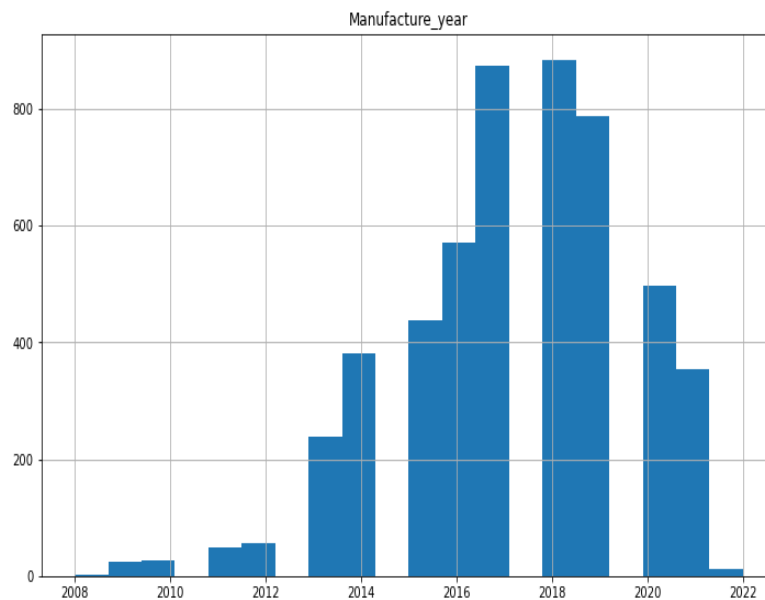
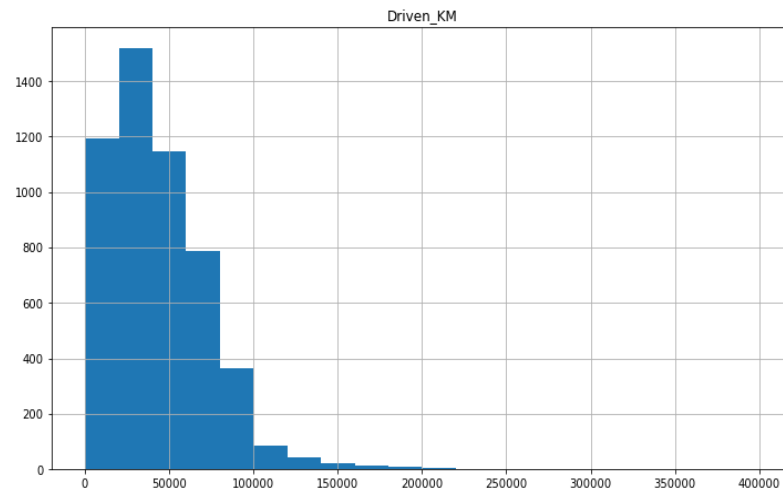
```
In [41]: 1 # check correlation matrix with heatmap.
2 plt.figure(figsize=(12,8))
3 sns.heatmap(df.corr(), annot=True, fmt = '.1f', cmap = 'Blues')
4 #sns.heatmap(df.corr(), cmap='coolwarm', annot=True, fmt='.1f', linewidths=.1)
5 plt.show()
```

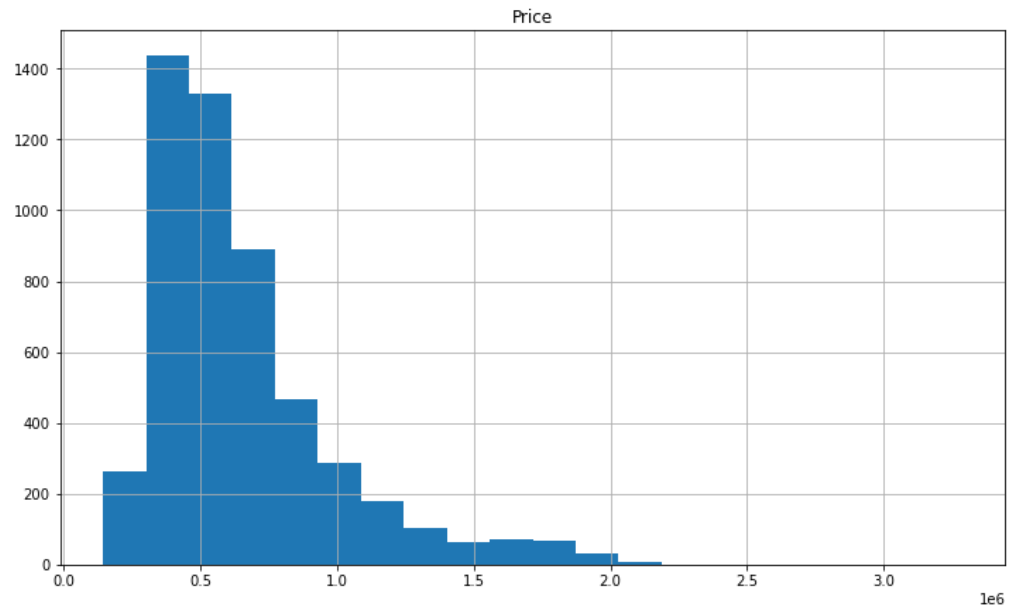


- Display histplot of city pincode column.

```
In [23]: 1 # display histogram of a columns.  
2 df.hist(figsize=(12,25), layout=(3,1), sharex=False, bins=20)
```

```
Out[23]: array([[<AxesSubplot:title={center:'Driven_KM'}>],  
                [<AxesSubplot:title={center:'Manufacture_year'}>],  
                [<AxesSubplot:title={center:'Price'}>]], dtype=object)
```

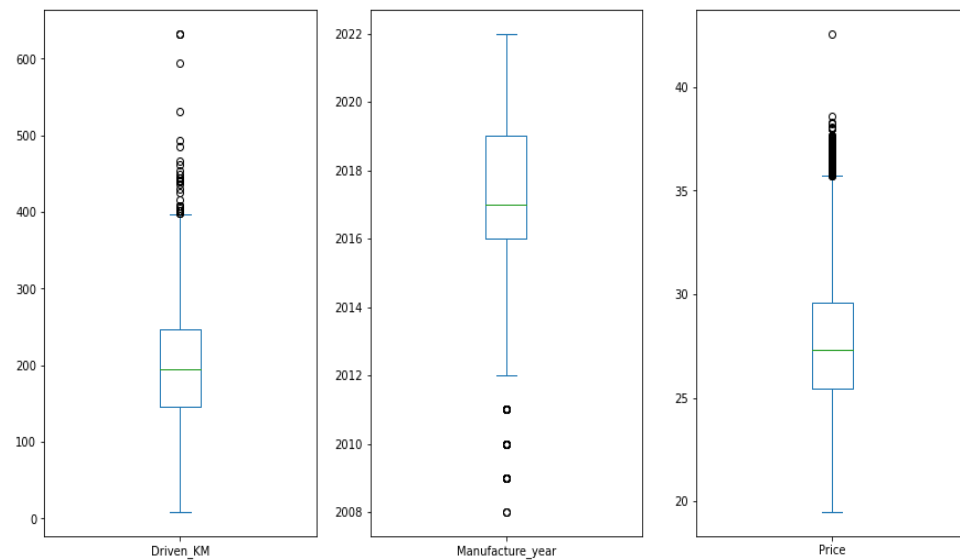




- Display outliers of all columns.

```
In [33]: 1 # checking outliers with boxplot.
          2 df.plot(kind='box', subplots=True, layout=(1,3), figsize=(15,8))
```

```
Out[33]: Driven_KM      AxesSubplot(0.125,0.125;0.227941x0.755)
          Manufacture_year AxesSubplot(0.398529,0.125;0.227941x0.755)
          Price           AxesSubplot(0.672059,0.125;0.227941x0.755)
          dtype: object
```



- Data Pre-processing and Scaling the data.

```
In [46]: 1 x = df.drop('Price',axis=1)
        2 y = df['Price']
```

```
In [47]: 1 x.head()
```

```
Out[47]:
```

	Brand	Model	Variant	Transmission	Driven_KM	Fuel_type	No. of owners	Manufacture_year	Location
0	10	95	63	1	97.519229	1	0	2021	0
1	11	8	545	1	91.318125	1	0	2016	0
2	11	31	651	1	146.955776	1	1	2020	0
3	11	75	349	1	129.942295	1	0	2019	0
4	18	60	500	1	120.287156	1	0	2020	0

```
In [48]: 1 y.head()
```

```
Out[48]: 0    32.143404
        1    24.104620
        2    31.927450
        3    27.623997
        4    27.632295
        Name: Price, dtype: float64
```

```
In [49]: 1 print(x.shape, y.shape)
```

```
(5132, 9) (5132,)
```

Scaling the data ¶

```
In [50]: 1 from sklearn.preprocessing import StandardScaler
        2
        3 st = StandardScaler()
        4 x = st.fit_transform(x)
        5 x
```

```
Out[50]: array([[ 0.25599352,  1.68925997, -1.65671813, ..., -0.4988746 ,
        1.61647718, -1.53744764],
        [ 0.51153889, -1.24634569,  0.93931325, ..., -0.4988746 ,
        -0.52325996, -1.53744764],
        [ 0.51153889, -0.47026603,  1.51022472, ...,  1.63999296,
        1.18852975, -1.53744764],
        ...,
        [ 0.51153889, -0.60523641, -1.00501731, ..., -0.4988746 ,
        1.18852975, -0.07032579],
        [-1.02173332, -0.20032528,  0.38455966, ...,  1.63999296,
        -0.09531253, -0.07032579],
        [-1.02173332, -0.74020678, -1.31201687, ..., -0.4988746 ,
        0.3326349 , -0.07032579]])
```

- Run and evaluate selected models.

Finding best random_state

```
In [51]: 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 [52]: 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 0 is 0.4807862529115282
R2 score correspond to random state 0 is 0.9455381681637013
R2 score correspond to random state 0 is 0.6181095072485208
R2 score correspond to random state 0 is 0.8853137721920655
R2 score correspond to random state 0 is 0.9047316471625844
R2 score correspond to random state 1 is 0.44706740411152446
R2 score correspond to random state 1 is 0.9424326724099487
R2 score correspond to random state 1 is 0.5935615530844223
R2 score correspond to random state 1 is 0.8651738098326522
R2 score correspond to random state 1 is 0.8941644930762143
R2 score correspond to random state 2 is 0.4829853081477473
R2 score correspond to random state 2 is 0.9505640141805456
R2 score correspond to random state 2 is 0.628458826646328
R2 score correspond to random state 2 is 0.8768670966488437
R2 score correspond to random state 2 is 0.9093284809734428
R2 score correspond to random state 3 is 0.4420868858361279
R2 score correspond to random state 3 is 0.9534774693550198
R2 score correspond to random state 3 is 0.6378161202616932
R2 score correspond to random state 3 is 0.8756007430474653
```

```
In [53]: 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 90 is 0.9638785765449112 and model is RandomForestRegressor()
```

Creating train-test split

```
In [54]: 1 x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=.20,random_state = 90)
```

Apply best model

```
In [55]: 1 rf.fit(x_train,y_train)
2 pred = rf.predict(x_test)
3 print("r2 score :- ",r2_score(y_test,pred))
```

r2 score :- 0.9642235493019548

```
In [56]: 1 # Predict the value:
2 print("predicted car price:",pred)
3 print("actual car price",y_test)
```

predicted car price: [25.99206867 29.60509919 27.01303367 ... 24.42923861 28.94813261
23.27003821]
actual car price 2448 25.452737
1583 29.717317
3321 28.234410
570 30.667215
4067 27.474539
...
992 33.837358
3072 29.382087
374 24.242660
929 28.504847
3875 23.338827
Name: Price, Length: 1027, dtype: float64

- Checking MAE, MSE and RMSE:

Check MAE, MSE and RMSE

```
In [57]: 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 0.4120222169118902
mean squared error 0.394599340273081
Root mean squared error 0.6281714258648519

- Hypertuning of the Model.

```
In [60]: 1 from sklearn.model_selection import RandomizedSearchCV

In [61]: 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 # Number of features to consider at every split
6 max_features = ['auto', 'sqrt']
7 # Maximum number of levels in tree
8 max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
9 # Minimum number of samples required to split a node
10 min_samples_split = [2, 5, 10, 15, 100]
11 # Minimum number of samples required at each leaf node
12 min_samples_leaf = [1, 2, 5, 10]

In [62]: 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 [63]: 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 [64]: 1 rf_random.fit(x_train,y_train)
```

```
Out[64]: RandomizedSearchCV(cv=9, estimator=RandomForestRegressor(), n_jobs=1,
param_distributions={'max_depth': [5, 10, 15, 20, 25, 30],
'max_features': ['auto', 'sqrt'],
'min_samples_leaf': [1, 2, 5, 10],
'min_samples_split': [2, 5, 10, 15, 100],
'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200]},
random_state=90, scoring='neg_mean_squared_error',
verbose=2)
```

```
In [65]: 1 rf_random.best_params_

Out[65]: {'n_estimators': 600,
'min_samples_split': 5,
'min_samples_leaf': 2,
'max_features': 'sqrt',
'max_depth': 30}
```

```
In [66]: 1 rf = RandomForestRegressor(n_estimators=600, min_samples_split=5, min_samples_leaf=2, 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=9)
10 rfc = rfscore.mean()
11 print("Cross Val Score is",rfc*100)
```

R2 score: 94.7143407769353
Cross Val Score is 91.70908177901437

- 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.