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]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Load the dataset.

In [2]:
```
import pandas as pd
df = pd.read_excel(r'car_data.xlsx')
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 × 10 columns

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

Out[3]: (5197, 10)

- Display all column name of dataset.

```
In [4]:
#Check column of the dataframe.
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]:
#Get the column datatypes.
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
```

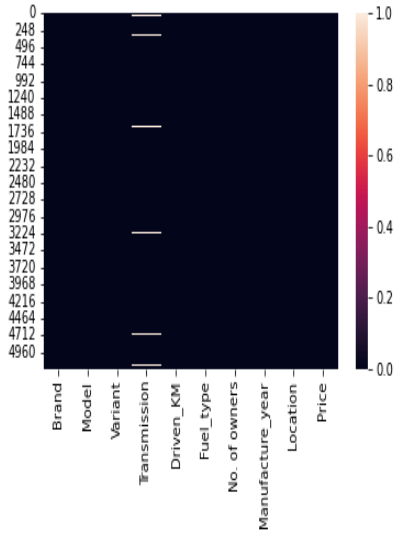
```
In [7]:
#Get a count of the empty values for each column.
df.isna().sum()
```

```
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()
```

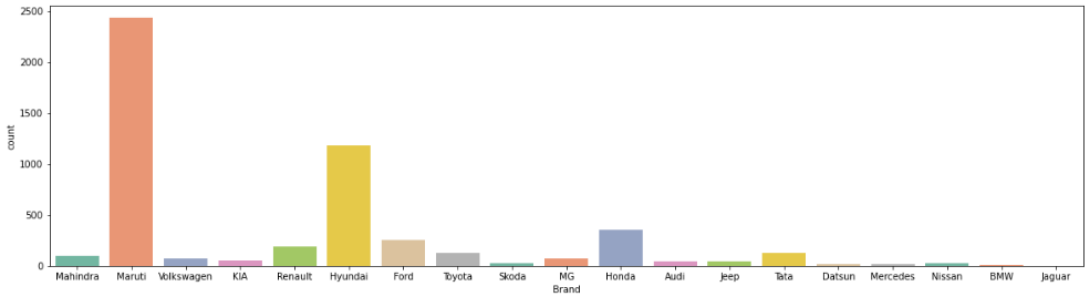
```
Out[17]:
```

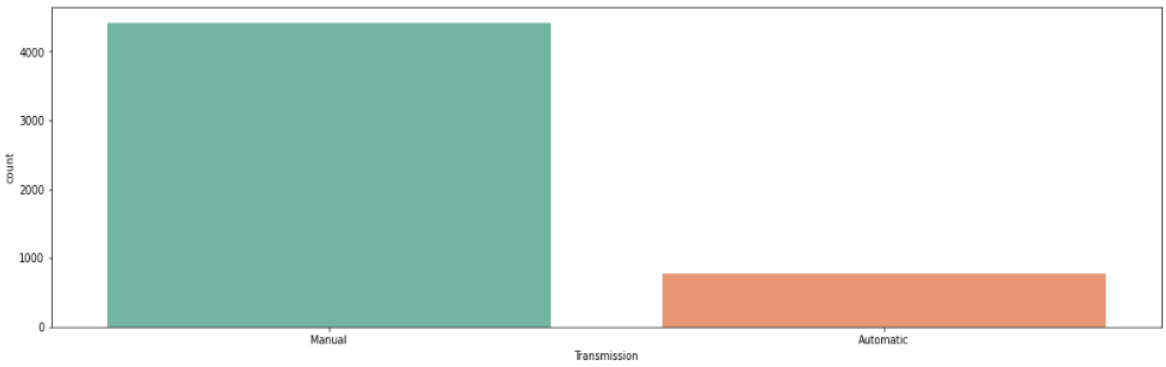
	Driven_KM	Manufacture_year	Price
count	5197.000000	5197.000000	5197.000000
mean	43790.950741	2017.169906	648476.928420
std	31013.177890	2.413196	339002.018013
min	71.000000	2008.000000	143799.000000
25%	21473.000000	2016.000000	420299.000000
50%	38225.000000	2017.000000	554699.000000
75%	60804.000000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

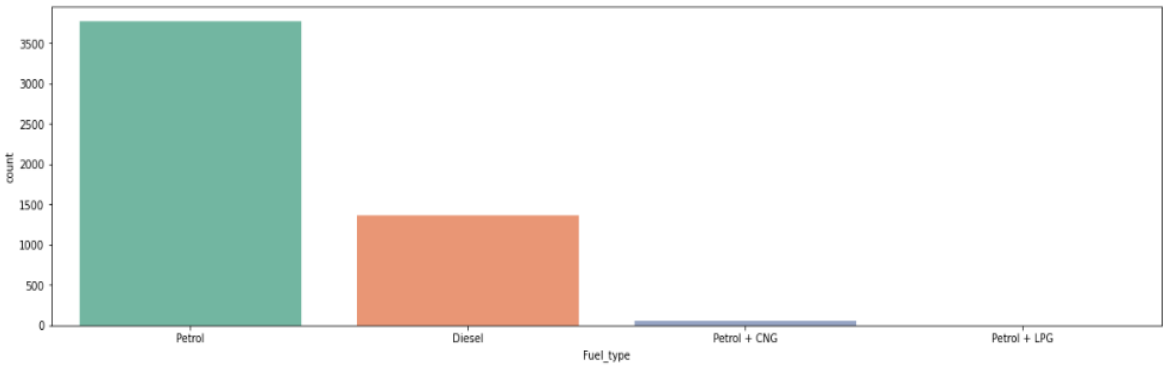
- Display countplot of columns.

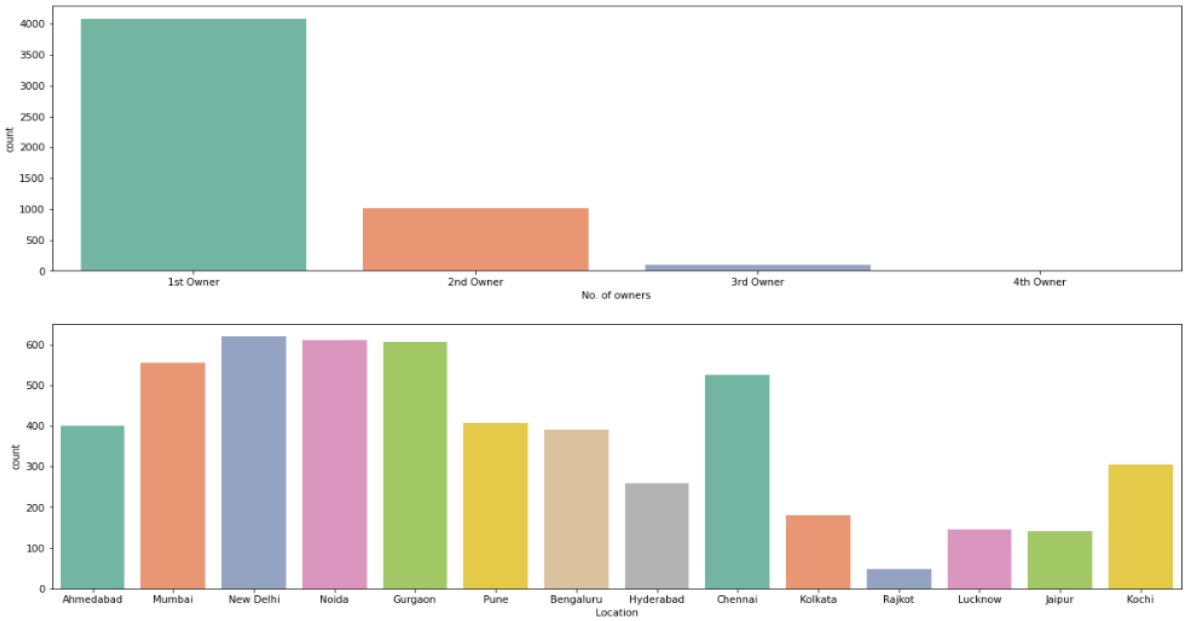
In [19]:
```
#Visualize the number of other independent categorical variables.
fig,ax = plt.subplots(5,1,figsize=(20,30))
sns.countplot(df['Brand'], data=df, palette='Set2',ax=ax[0])
sns.countplot(df['Transmission'], data=df, palette='Set2',ax=ax[1])
sns.countplot(df['Fuel_type'], data=df, palette='Set2',ax=ax[2])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

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





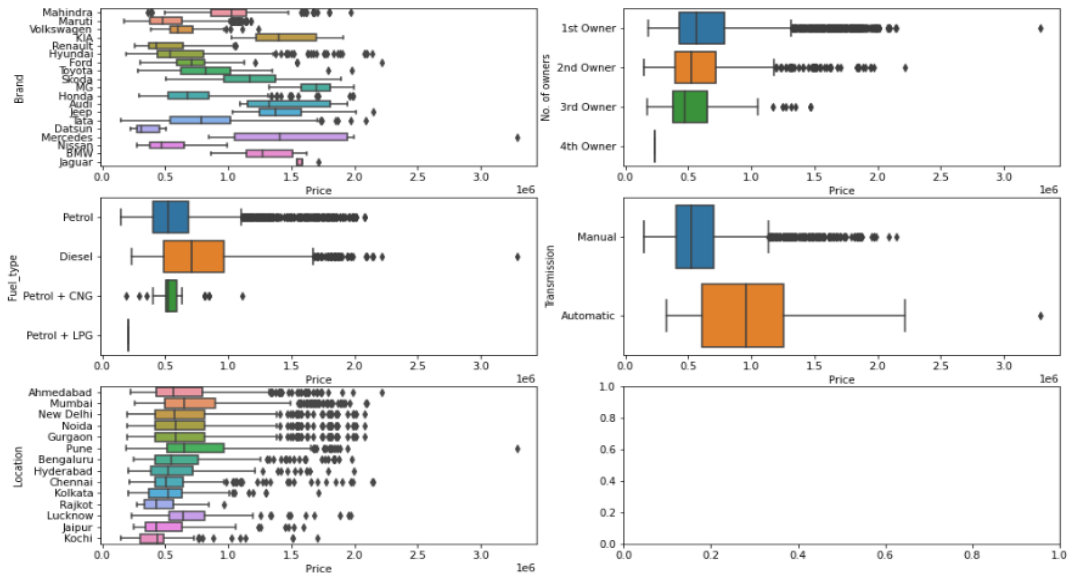




- Display boxplot of columns to compare with price.

```
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'>
```

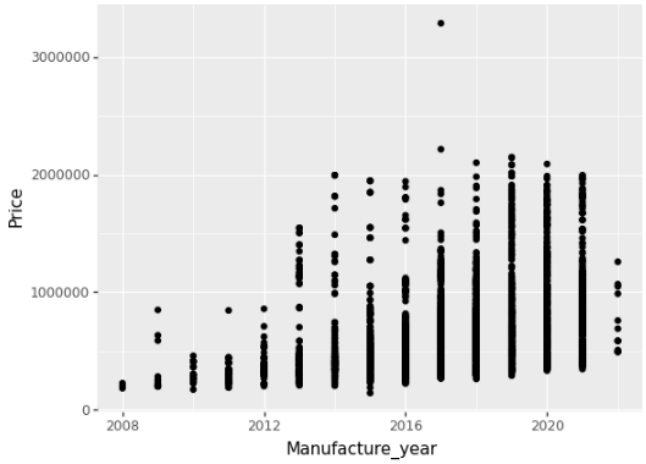


- Display plotline of manuf. year column to compare with price.

In [22]:

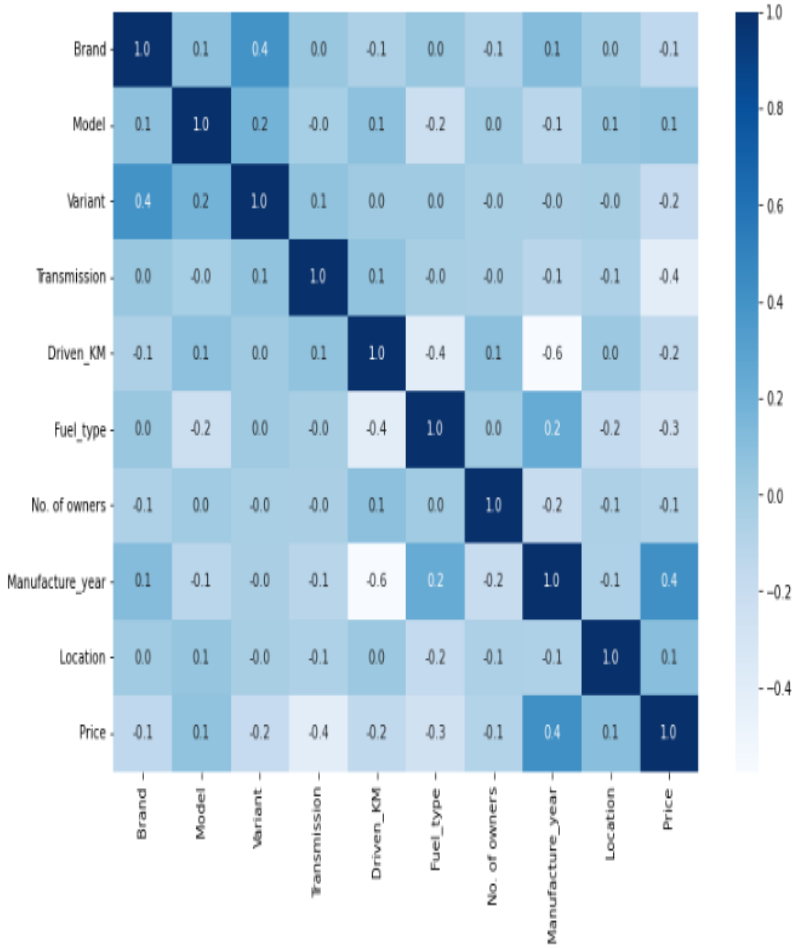
```
from plotnine.data import mpg
from plotnine import ggplot, aes, geom_point

(
    ggplot(df) # What data to use
    + aes(x="Manufacture_year", y="Price") # What variable to use
    + geom_point() # Geometric object to use for drawing
)
```



- Display correlation of columns using heatmap.

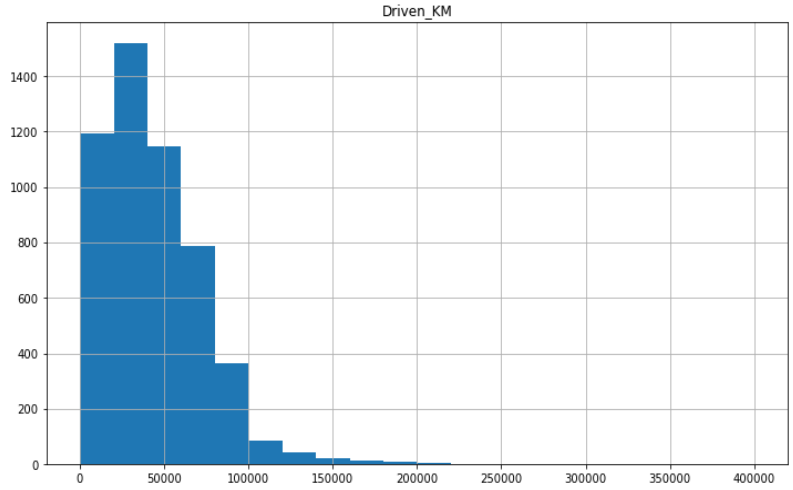
In [41]:
```
# check correlation matrix with heatmap.
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(), annot=True, fmt = '.1f', cmap = 'Blues')
#sns.heatmap(df.corr(), cmap='coolwarm', annot=True, fmt='.1f', linewidths=.1)
plt.show()
```

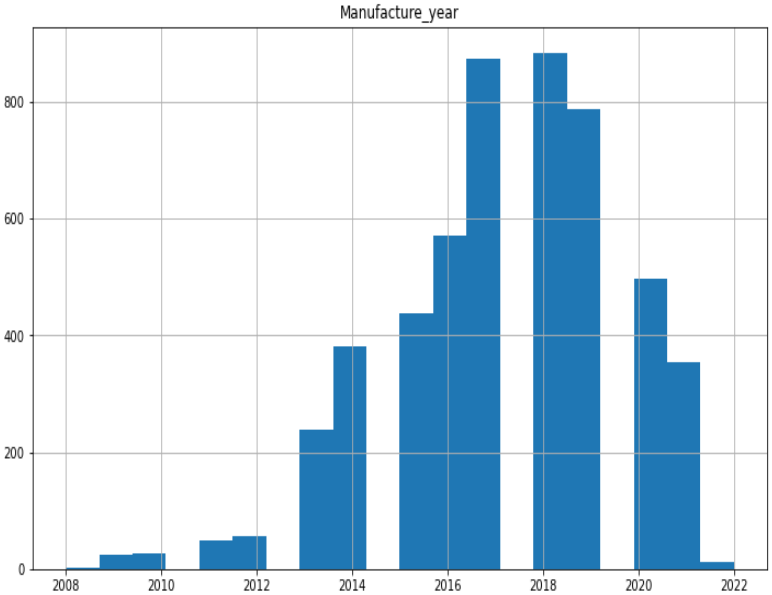


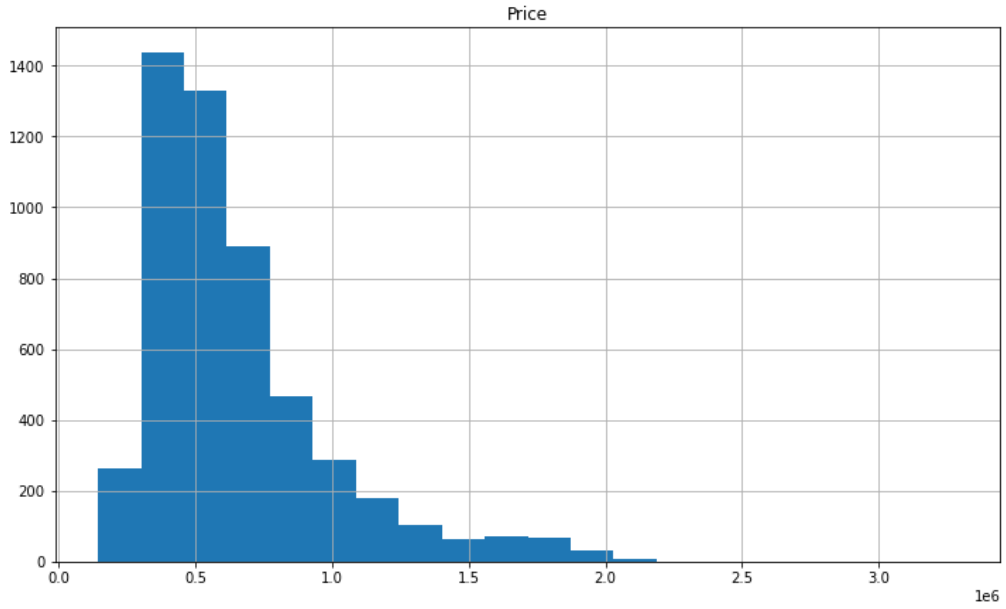
- Display histplot of city pincode column.

```
In [23]:
1 # display histogram of acolumns.
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)
```



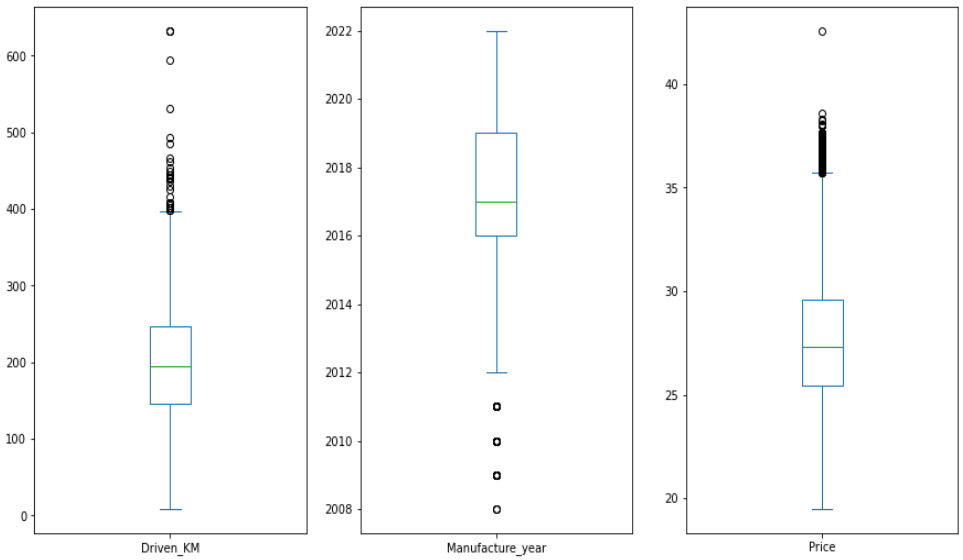




- Display outliers of all columns.

```
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 Scalling the data.

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

In [47]:
```
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]:
```
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]:
```
print(x.shape, y.shape)
```

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

Scaling the data ¶

In [50]:
```
from sklearn.preprocessing import StandardScaler

st = StandardScaler()
x = st.fit_transform(x)
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]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()

from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()

from sklearn.ensemble import AdaBoostRegressor
abr = AdaBoostRegressor()

from sklearn.ensemble import GradientBoostingRegressor
gbr = GradientBoostingRegressor()

from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor()

from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

In [52]:
```
model = [lr,rf,abr,gbr,dtr]
max_r2_score = 0
for r_state in range(0,100):
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=.20,random_state = r_state)
    for i in model:
        i.fit(x_train,y_train)
        pred_test = i.predict(x_test)
        r2_sc = r2_score(y_test,pred_test)
        print("R2 score correspond to random state ",r_state,"is",r2_sc)
        if r2_sc > max_r2_score:
            max_r2_score = r2_sc
            final_state = r_state
            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]:
```
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]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=.20,random_state = 90)
```

Apply best model

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

```
r2 score :-  0.9642235493019548
```

In [56]:
```
# Predict the value:
print("predicted car price:",pred)
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]:
```
# Display MAE, MSE and RMSE:
from sklearn.metrics import mean_squared_error, mean_absolute_error
print('error:')

print('mean absolute error',mean_absolute_error(y_test,pred))
print('mean squared error',mean_squared_error(y_test,pred))

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]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [61]:
```
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# 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(5, 30, num = 6)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

In [62]:
```
parameters = {'n_estimators': n_estimators,
              'max_features': max_features,
              'max_depth': max_depth,
              'min_samples_split': min_samples_split,
              'min_samples_leaf': min_samples_leaf}
```

In [63]:
```
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = parameters, scoring='neg_mean_squared_error', n_iter =
```

In [64]:
```
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]:
```
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]:
```
rf = RandomForestRegressor(n_estimators=600, min_samples_split=5, min_samples_leaf=2, max_features='sqrt', max_depth=30)
rf.fit(x_train,y_train)
rf.score(x_train,y_train)
pred_decision = rf.predict(x_test)

rfs = r2_score(y_test,pred_decision)
print("R2 score:",rfs*100)

rfscore = cross_val_score(rf,x,y,cv=9)
rfc = rfscore.mean()
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.