In [1]:

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

In [2]:

```
mydata=pd.read_excel("C:Downloads\data.csv.xlsx")
```

In [3]:

```
mydata
```

Out[3]:

	origin	cylinders	displacement	horsepower	weight	acceleration	year	name	Kilomet
0	1	8	307.0	130	3504	12.0	1970	chevrolet chevelle malibu	
1	1	8	350.0	165	3693	11.5	1970	buick skylark 320	
2	1	8	318.0	150	3436	11.0	1970	plymouth satellite	
3	1	8	304.0	150	3433	12.0	1970	amc rebel sst	
4	1	8	302.0	140	3449	10.5	1970	ford torino	
...	...	...	...	...	...	...	...	...	
393	1	4	140.0	86	2790	15.6	1982	ford mustang gl	
394	2	4	97.0	52	2130	24.6	1982	vw pickup	
395	1	4	135.0	84	2295	11.6	1982	dodge rampage	
396	1	4	120.0	79	2625	18.6	1982	ford ranger	
397	1	4	119.0	82	2720	19.4	1982	chevy s-10	

398 rows × 9 columns

In [4]:

```
#Replacing "?" available in the dataset to NA and then we drop NA
mydata1=mydata.replace({'?':np.nan}).dropna()
```

In [5]:

```
mydata1
```

Out[5]:

	origin	cylinders	displacement	horsepower	weight	acceleration	year	name	Kilomet
0	1	8	307.0	130.0	3504	12.0	1970	chevrolet chevelle malibu	
1	1	8	350.0	165.0	3693	11.5	1970	buick skylark 320	
2	1	8	318.0	150.0	3436	11.0	1970	plymouth satellite	
3	1	8	304.0	150.0	3433	12.0	1970	amc rebel sst	
4	1	8	302.0	140.0	3449	10.5	1970	ford torino	
...	...	...	...	...	...	...	...	...	
393	1	4	140.0	86.0	2790	15.6	1982	ford mustang gl	
394	2	4	97.0	52.0	2130	24.6	1982	vw pickup	
395	1	4	135.0	84.0	2295	11.6	1982	dodge rampage	
396	1	4	120.0	79.0	2625	18.6	1982	ford ranger	
397	1	4	119.0	82.0	2720	19.4	1982	chevy s-10	

392 rows × 9 columns

In [6]:

```
#Converting the datatype of "horsepower" from object to float
mydata1['horsepower']=pd.to_numeric(mydata1['horsepower'],downcast="float")
```

In [7]:

```
mydata1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 392 entries, 0 to 397
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   origin               392 non-null    int64
 1   cylinders            392 non-null    int64
 2   displacement         392 non-null    float64
 3   horsepower           392 non-null    float32
 4   weight               392 non-null    int64
 5   acceleration         392 non-null    float64
 6   year                 392 non-null    int64
 7   name                 392 non-null    object
 8   Kilometer_per_liter  392 non-null    float64
dtypes: float32(1), float64(3), int64(4), object(1)
memory usage: 29.1+ KB
```

In [8]:

```
#Dropping the "name" column as it is not required
mydata2=mydata1.drop("name",axis=1)
mydata2
```

Out[8]:

	origin	cylinders	displacement	horsepower	weight	acceleration	year	Kilometer_per_lite
0	1	8	307.0	130.0	3504	12.0	1970	7.65258
1	1	8	350.0	165.0	3693	11.5	1970	6.37715
2	1	8	318.0	150.0	3436	11.0	1970	7.65258
3	1	8	304.0	150.0	3433	12.0	1970	6.80229
4	1	8	302.0	140.0	3449	10.5	1970	7.22744
...	...	...	...	...	...	...	...	..
393	1	4	140.0	86.0	2790	15.6	1982	11.47888
394	2	4	97.0	52.0	2130	24.6	1982	18.70632
395	1	4	135.0	84.0	2295	11.6	1982	13.60459
396	1	4	120.0	79.0	2625	18.6	1982	11.90402
397	1	4	119.0	82.0	2720	19.4	1982	13.17945

392 rows × 8 columns

In [9]:

```
#Dropping the "year" column as it is not required
mydata3=mydata2.drop("year",axis=1)
mydata3
```

Out[9]:

	origin	cylinders	displacement	horsepower	weight	acceleration	Kilometer_per_liter
0	1	8	307.0	130.0	3504	12.0	7.652587
1	1	8	350.0	165.0	3693	11.5	6.377156
2	1	8	318.0	150.0	3436	11.0	7.652587
3	1	8	304.0	150.0	3433	12.0	6.802299
4	1	8	302.0	140.0	3449	10.5	7.227443
...	...	...	...	...	...	...	...
393	1	4	140.0	86.0	2790	15.6	11.478880
394	2	4	97.0	52.0	2130	24.6	18.706323
395	1	4	135.0	84.0	2295	11.6	13.604599
396	1	4	120.0	79.0	2625	18.6	11.904024
397	1	4	119.0	82.0	2720	19.4	13.179455

392 rows × 7 columns

In [10]:

```
#Now we calculate the mean, std, min, max and quantiles by using describe function.
mydata3.describe()
```

Out[10]:

	origin	cylinders	displacement	horsepower	weight	acceleration	Kilometer_
count	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	39
mean	1.576531	5.471939	194.411990	104.469391	2977.584184	15.541327	
std	0.805518	1.705783	104.644004	38.491138	849.402560	2.758864	
min	1.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	
25%	1.000000	4.000000	105.000000	75.000000	2225.250000	13.775000	
50%	1.000000	4.000000	151.000000	93.500000	2803.500000	15.500000	
75%	2.000000	8.000000	275.750000	126.000000	3614.750000	17.025000	1
max	3.000000	8.000000	455.000000	230.000000	5140.000000	24.800000	1

In [11]:

```
#We use null function to calculate the null values present in the dataset.
mydata3.isnull().sum()
```

Out[11]:

```
origin               0
cylinders            0
displacement         0
horsepower           0
weight               0
acceleration         0
Kilometer_per_liter  0
dtype: int64
```

In [12]:

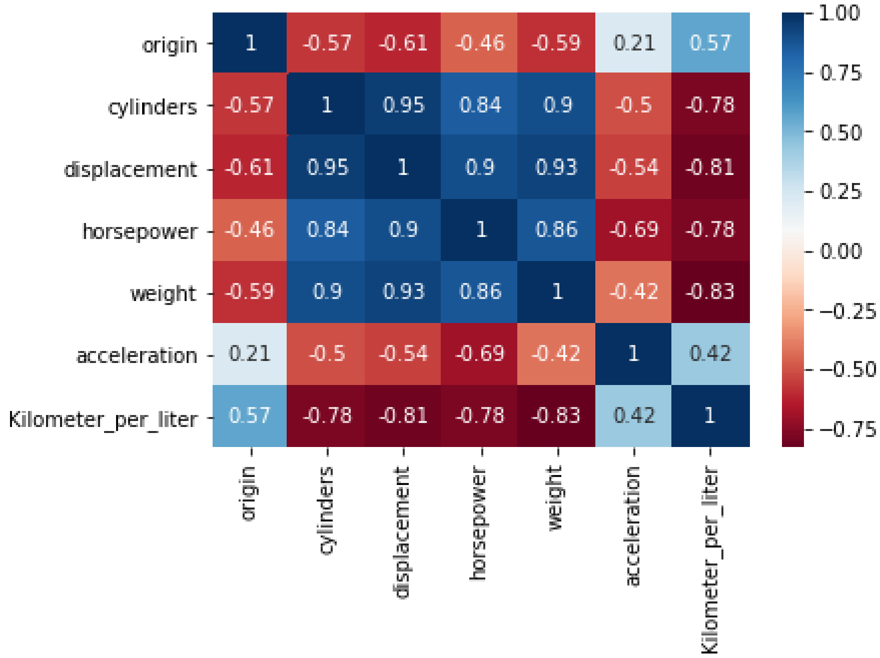
```
#Now we calculate the correlation of our dataset.
mydata3_corr=mydata3.corr()
mydata3_corr
```

Out[12]:

	origin	cylinders	displacement	horsepower	weight	acceleration	Kil
origin	1.000000	-0.568932	-0.614535	-0.455171	-0.585005	0.212746	
cylinders	-0.568932	1.000000	0.950823	0.842983	0.897527	-0.504683	
displacement	-0.614535	0.950823	1.000000	0.897257	0.932994	-0.543800	
horsepower	-0.455171	0.842983	0.897257	1.000000	0.864538	-0.689196	
weight	-0.585005	0.897527	0.932994	0.864538	1.000000	-0.416839	
acceleration	0.212746	-0.504683	-0.543800	-0.689196	-0.416839	1.000000	
Kilometer_per_liter	0.565209	-0.777618	-0.805127	-0.778427	-0.832244	0.423329	

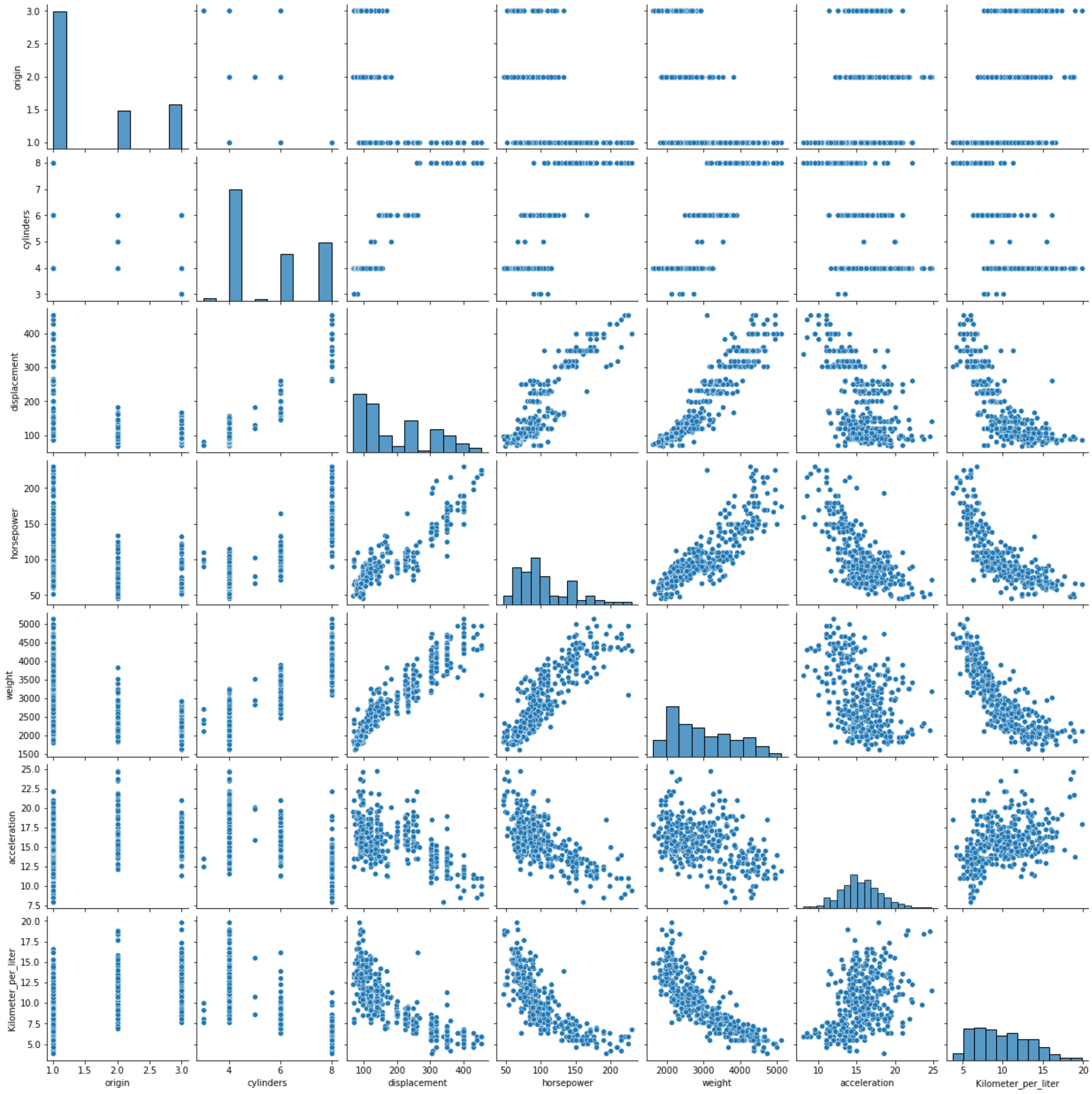
In [13]:

```
#We use heatmap to demonstrate the correlation pattern.
sns.heatmap(mydata3_corr,annot=True,cmap='RdBu');
```



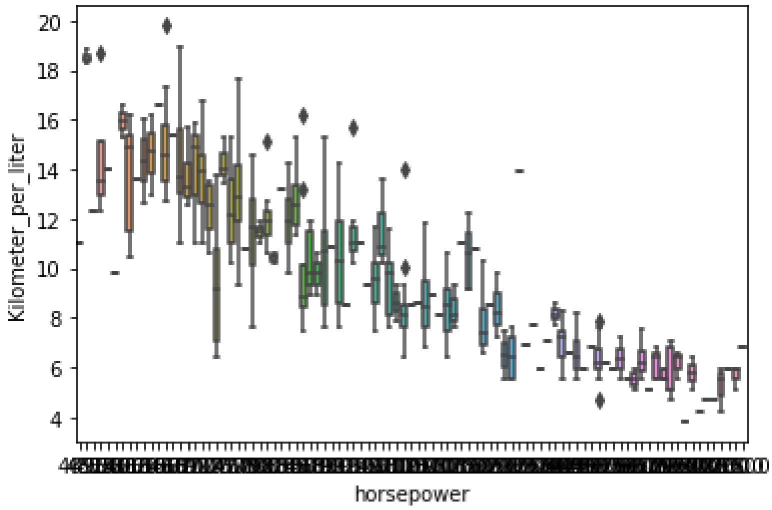
In [14]:

```
#We use pairplot to compare all the columns with eachother.
sns.pairplot(mydata3);
```



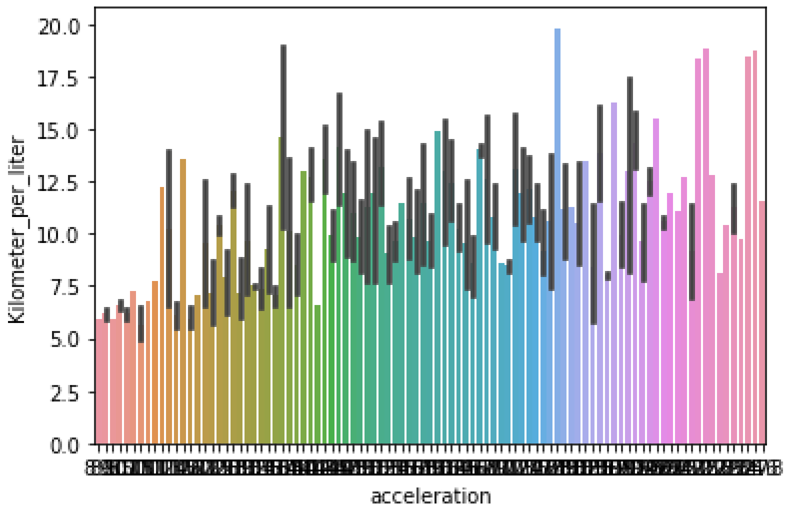
In [15]:

```
#Here by using boxplot we can compare two columns specifically.
sns.boxplot(x="horsepower",y="Kilometer_per_liter",data=mydata3);
```



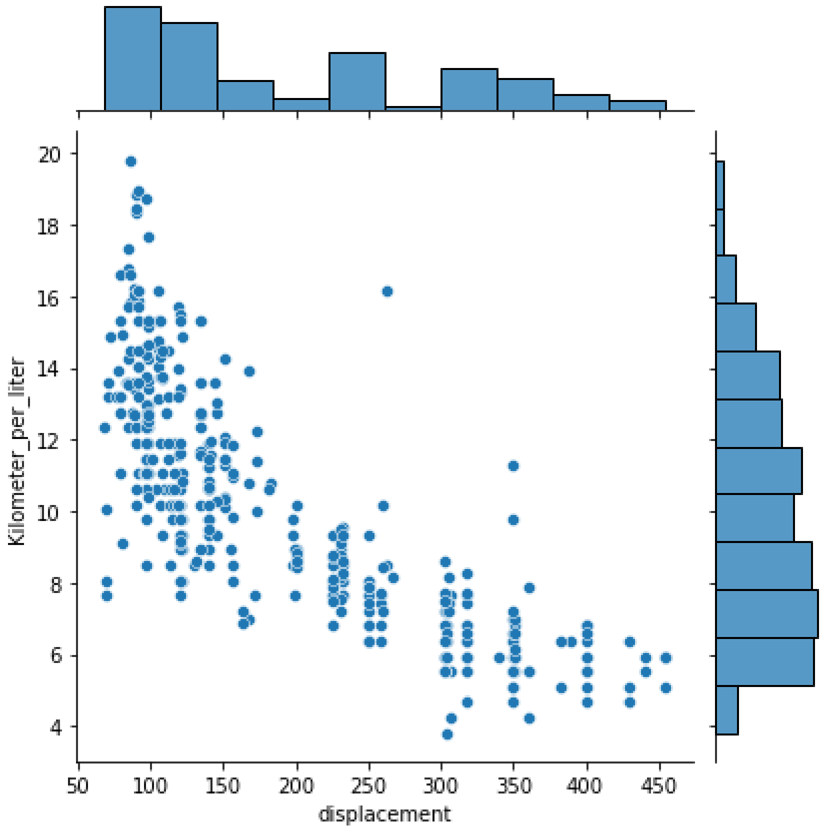
In [16]:

```
#Similarly like the boxplot, We can use barplot as well to compare 2 different columns.
sns.barplot(x="acceleration",y="Kilometer_per_liter",data=mydata3);
```



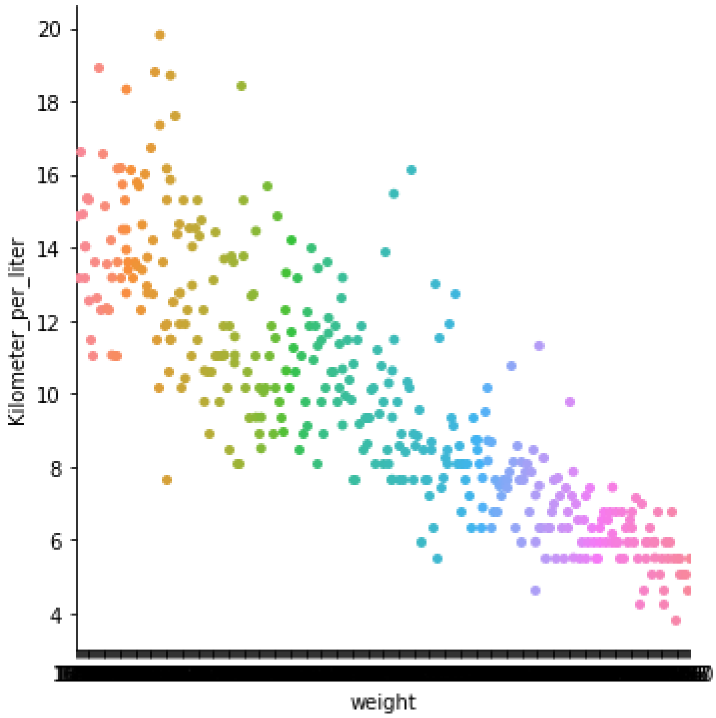
In [17]:

```
#We can also use joinplot to compare 2 different columns.
sns.jointplot(x="displacement",y="Kilometer_per_liter",data=mydata3);
```



In [18]:

```
#And finally we have used catplot to categorize 2 columns which is similar to previous grap
sns.catplot(x="weight",y="Kilometer_per_liter",data=mydata3);
```



In [19]:

```
#Now we separate dependent and independent values.
y_dep=mydata3.Kilometer_per_liter
y_dep
```

Out[19]:

```
0       7.652587
1       6.377156
2       7.652587
3       6.802299
4       7.227443
          ...
393    11.478880
394    18.706323
395    13.604599
396    11.904024
397    13.179455
Name: Kilometer_per_liter, Length: 392, dtype: float64
```

In [20]:

```
#Here we drop the dependent values to filter out all the independent values
x_ind=mydata3.drop("Kilometer_per_liter",axis=1)
x_ind
```

Out[20]:

	origin	cylinders	displacement	horsepower	weight	acceleration
0	1	8	307.0	130.0	3504	12.0
1	1	8	350.0	165.0	3693	11.5
2	1	8	318.0	150.0	3436	11.0
3	1	8	304.0	150.0	3433	12.0
4	1	8	302.0	140.0	3449	10.5
...	...	...	...	...	...	...
393	1	4	140.0	86.0	2790	15.6
394	2	4	97.0	52.0	2130	24.6
395	1	4	135.0	84.0	2295	11.6
396	1	4	120.0	79.0	2625	18.6
397	1	4	119.0	82.0	2720	19.4

392 rows × 6 columns

Checking p_value and R_Squared value

In [21]:

```
import statsmodels.api as sm
```

In [22]:

```
model=sm.OLS(y_dep,x_ind)  #OLS means Ordinary Least SQuare
```

In [23]:

```
my_fit=model.fit()
```

In [24]:

```
my_fit.summary()
```

Out[24]:

OLS Regression Results

Dep. Variable:	Kilometer_per_liter	**R-squared (uncentered):**	0.954
Model:	OLS	**Adj. R-squared (uncentered):**	0.953
Method:	Least Squares	**F-statistic:**	1326.
Date:	Wed, 25 Aug 2021	**Prob (F-statistic):**	4.64e-254
Time:	19:54:08	**Log-Likelihood:**	-875.83
No. Observations:	392	**AIC:**	1764.
Df Residuals:	386	**BIC:**	1787.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
origin	1.2389	0.184	6.741	0.000	0.878	1.600
cylinders	0.7222	0.207	3.486	0.001	0.315	1.130
displacement	-0.0125	0.005	-2.533	0.012	-0.022	-0.003
horsepower	0.0352	0.008	4.548	0.000	0.020	0.050
weight	-0.0025	0.000	-5.643	0.000	-0.003	-0.002
acceleration	0.6479	0.041	15.702	0.000	0.567	0.729

Omnibus:	17.389	**Durbin-Watson:**	1.132
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	21.608
Skew:	0.405	**Prob(JB):**	2.03e-05
Kurtosis:	3.816	**Cond. No.**	5.89e+03

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 5.89e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [ ]:

```
#Conclusion :-
# R-Squared value is greater than 0.5 i.e. 0.954, Therefore the dataset is stable.
# P-values are less than 0.05. Therefore, all the variables are statistically significant.
```

Machine Learning

In [25]:

```
import sklearn #Scikit Library used for performing Machine Learning in Python.
```

In [26]:

```
from sklearn import model_selection
from sklearn.model_selection import train_test_split
```

In [27]:

```
x_train,x_test,y_train,y_test=train_test_split(x_ind,y_dep,train_size=0.8,random_state=1)
# x_train = 80% of x independent values
# x_test = 20% of x independent values
# y_train = 80% of y dependent values
# y_test = 20% of y dependent values
```

In [28]:

```
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
```

In [29]:

```
#Create a model
model=LinearRegression()
```

In [30]:

```
#Then we fit the model.
model.fit(x_train,y_train)
```

Out[30]:

```
LinearRegression()
```

In [31]:

```
#We calculate the machine predicted values.
y_pred=model.predict(x_test)
```

In [32]:

```
y_pred
```

Out[32]:

```
array([11.84999284, 13.00748369, 12.69198052, 10.16054164, 13.53359842,
       11.71838874, 12.50362867,  5.97177772, 12.40575286, 12.88272377,
        8.75525876, 12.36354569,  7.21451224, 13.56477432, 10.87968507,
        7.64836895, 11.12196563, 12.80917311,  4.29415956,  9.77448791,
       12.20592564,  8.21412026,  7.56271109,  5.65412394,  4.79535918,
        6.61835603, 13.47663107,  8.33010537,  9.27149279, 11.0898943 ,
        7.64715482, 10.39338981,  5.37455235,  9.62709968,  8.66000877,
        6.18499364,  8.20703932,  8.49184554, 13.56680576, 12.75703209,
        4.99869913,  5.21975065, 10.56245595,  9.80036802, 10.00123846,
        8.04112071,  4.62045055, 13.59130868,  9.11142812,  3.66535164,
        7.00078968,  9.39089576, 10.48187735, 11.24033864, 13.27501299,
        9.54249098,  9.83650567, 10.70298983, 10.65183325, 14.08718089,
       10.00362091, 11.53318538, 13.6774748 ,  8.52745023,  9.69936458,
        9.4352686 , 10.05142939,  6.89475022, 12.88007462,  3.92318037,
       12.11724342,  7.70844148,  6.92028422, 12.25216255, 11.3334102 ,
       12.07275168,  6.72769602,  5.63734172,  5.56535967])
```

In [56]:

```
#To calculate the score of Linear Regression.
model.score(x_test,y_test)
```

Out[56]:

```
0.7233881200171334
```

In [33]:

```
#We then compare the Actual values and Machine predicted values.
f_comp=pd.DataFrame({"Actual":y_test, "Machine_predicted":y_pred})
```

In [34]:

```
f_comp
```

Out[34]:

	Actual	Machine_predicted
82	9.778305	11.849993
167	12.329168	13.007484
356	13.774656	12.691981
120	8.077730	10.160542
385	16.155461	13.533598
...	...	...
23	11.053736	11.333410
295	15.177630	12.072752
13	5.952012	6.727696
91	5.526868	5.637342
62	5.526868	5.565360

79 rows × 2 columns

Residuals

In [35]:

```
#Residuals is used to calculate the error between machine predicted value and actual values
Res = y_pred-y_test
Res
```

Out[35]:

```
82      2.071688
167     0.678316
356    -1.082676
120     2.082811
385    -2.621862
          ...
23      0.279674
295    -3.104879
13      0.775684
91      0.110474
62      0.038491
Name: Kilometer_per_liter, Length: 79, dtype: float64
```

In [36]:

```
from sklearn.metrics import mean_squared_error as ms
```

In [37]:

```
mean_sqr=ms(y_test,y_pred)
mean_sqr
```

Out[37]:

```
3.4658929751135883
```

In [38]:

```
#RMSE
root_mean_sqr=np.sqrt(mean_sqr)
root_mean_sqr
```

Out[38]:

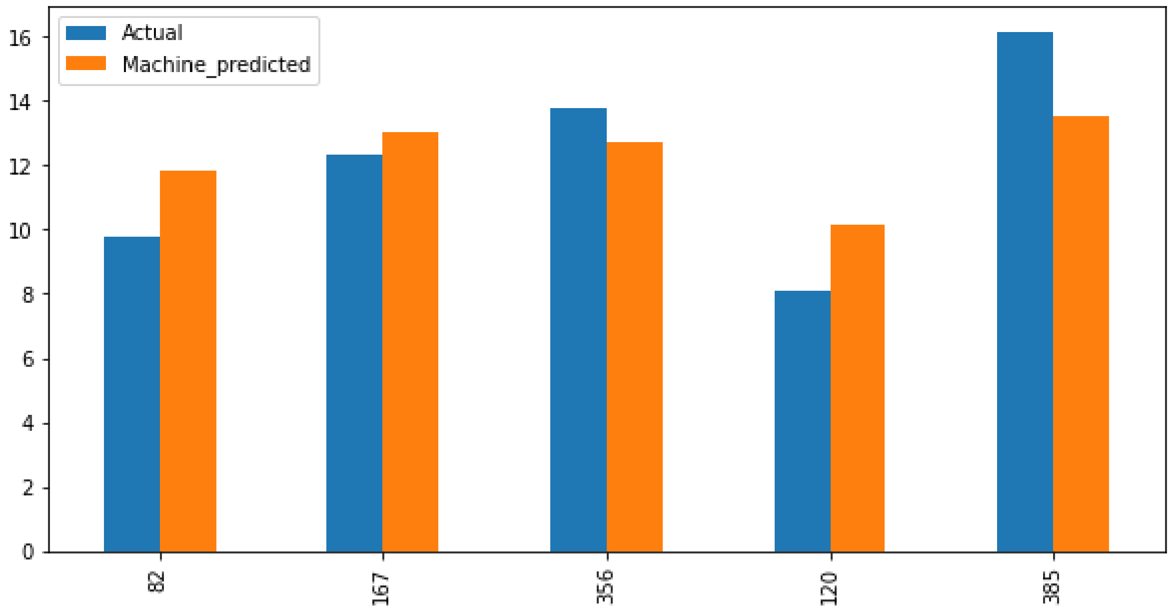
```
1.8616908913978143
```

In [39]:

```
#Comparison Graph between Actual and Machine Predicted values
comp_g=f_comp.head()
```

In [40]:

```
comp_g.plot(kind="bar", figsize=(10,5));
```



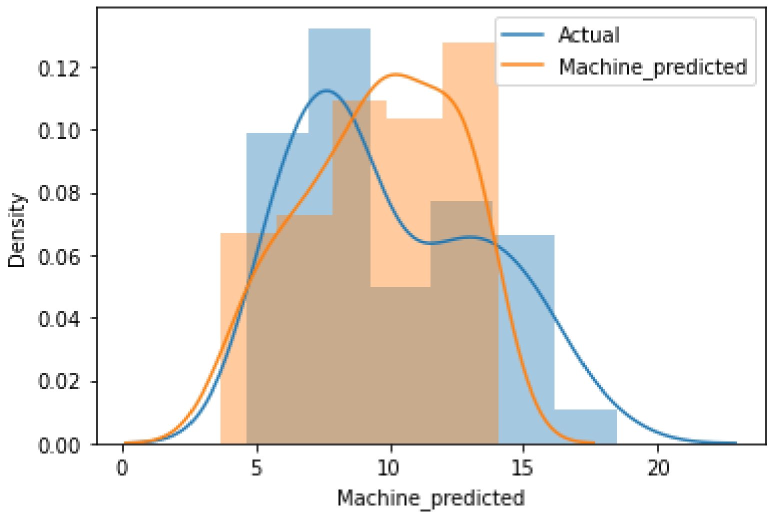
In [41]:

```
sns.distplot(f_comp["Actual"])
sns.distplot(f_comp["Machine_predicted"])
plt.legend(["Actual","Machine_predicted"])
```

```
C:\Users\aneef\anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\aneef\anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a fu
ture version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

Out[41]:

```
<matplotlib.legend.Legend at 0x22494fff700>
```



In [42]:

```
model.intercept_
```

Out[42]:

```
16.886328884015484
```

In [43]:

```
model.coef_
```

Out[43]:

```
array([ 0.64807509, -0.08358926,  0.00210625, -0.0229945 , -0.00199078,
        0.02209666])
```

Stochastic Gradient Descent

In [44]:

```
from sklearn.model_selection import train_test_split
```

In [45]:

```
X_train,X_test,Y_train,Y_test=train_test_split(x_ind,y_dep,train_size=0.8,random_state=1)
```

In [46]:

```
from sklearn.preprocessing import StandardScaler
```

In [47]:

```
norm=StandardScaler()
```

In [48]:

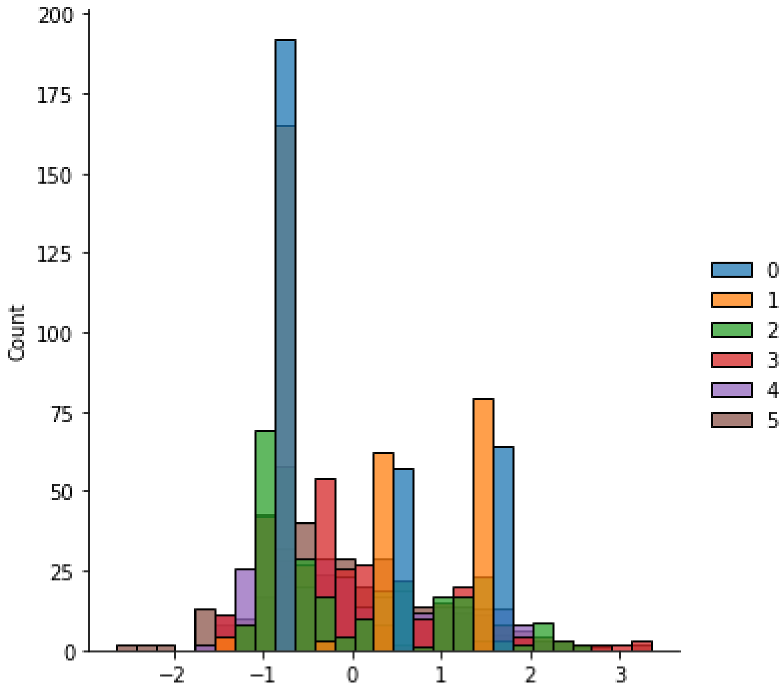
```
X_train=norm.fit_transform(X_train)
X_test=norm.fit_transform(X_test)
```

In [49]:

```
import seaborn as sns
```

In [50]:

```
sns.displot(X_train); # checking normal plot
```



In [51]:

```
from sklearn.linear_model import SGDRegressor
```

In [52]:

```
model1=SGDRegressor()
```

In [53]:

```
model1.fit(X_train,Y_train)
```

Out[53]:

```
SGDRegressor()
```

In [54]:

```
ypred=model1.predict(X_test)
```

In [55]:

```
#To calculate the score of SGDR.
model1.score(X_test,Y_test)
```

Out[55]:

```
0.7336026093987825
```