import libraries and load the dataset

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
cars=pd.read_csv("C:\\Users\\DELL\\Downloads\\DS- Data Sets\\Multi linear regression\\c
cars.head()
```

Out[2]:

	HP	MPG	VOL	SP	WT
0	49	53.700681	89	104.185353	28.762059
1	55	50.013401	92	105.461264	30.466833
2	55	50.013401	92	105.461264	30.193597
3	70	45.696322	92	113.461264	30.632114
4	53	50.504232	92	104.461264	29.889149

In [3]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 81 entries, 0 to 80
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   HP      81 non-null     int64
 1   MPG     81 non-null     float64
 2   VOL     81 non-null     int64
 3   SP      81 non-null     float64
 4   WT      81 non-null     float64
dtypes: float64(3), int64(2)
memory usage: 3.3 KB
```

In [4]:
```
cars.describe()
```

Out[4]:

	HP	MPG	VOL	SP	WT
count	81.000000	81.000000	81.000000	81.000000	81.000000
mean	117.469136	34.422076	98.765432	121.540272	32.412577
std	57.113502	9.131445	22.301497	14.181432	7.492813
min	49.000000	12.101263	50.000000	99.564907	15.712859
25%	84.000000	27.856252	89.000000	113.829145	29.591768
50%	100.000000	35.152727	101.000000	118.208698	32.734518
75%	140.000000	39.531633	113.000000	126.404312	37.392524
max	322.000000	53.700681	160.000000	169.598513	52.997752

In [5]:
```
cars.isnull()
```

Out[5]:

	HP	MPG	VOL	SP	WT
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
76	False	False	False	False	False
77	False	False	False	False	False
78	False	False	False	False	False
79	False	False	False	False	False
80	False	False	False	False	False

81 rows × 5 columns

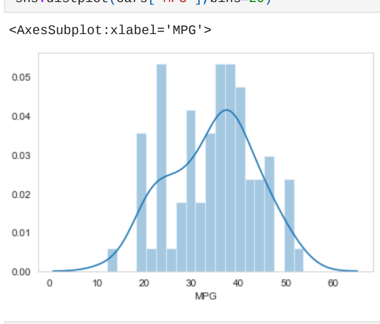
In [6]:
```
cars.corr()
```

Out[6]:

	HP	MPG	VOL	SP	WT
HP	1.000000	-0.725038	0.077459	0.973848	0.076513
MPG	-0.725038	1.000000	-0.529057	-0.687125	-0.526759
VOL	0.077459	-0.529057	1.000000	0.102170	0.999203
SP	0.973848	-0.687125	0.102170	1.000000	0.102439
WT	0.076513	-0.526759	0.999203	0.102439	1.000000

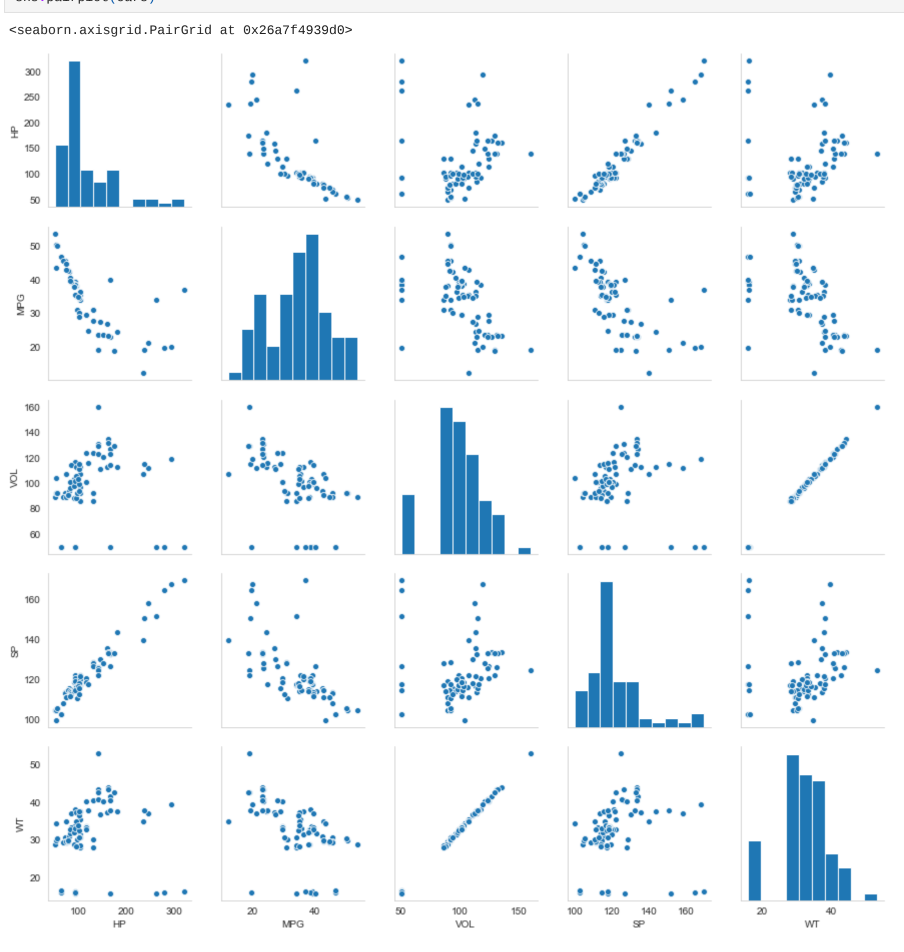
In [7]:
```
# set the background style of the plot
#MPG is dependent variable
sns.set_style('whitegrid',{'axes.grid' : False})
sns.distplot(cars['MPG'],bins=20)
```

Out[7]: <AxesSubplot:xlabel='MPG'>



In [8]:
```
#pairplot to check linearity of dependent and independent variables
sns.set_style('whitegrid',{'axes.grid' : False})
sns.pairplot(cars)
```

Out[8]: <seaborn.axisgrid.PairGrid at 0x26a7f4939d0>



create model and fit it

In [9]:
```
import statsmodels.formula.api as smf
model=smf.ols('MPG~HP+VOL+SP+WT',data=cars).fit()
```

In [10]:
```
#Coefficients
model.params
```

Out[10]:
```
Intercept    30.677336
HP           -0.205444
VOL          -0.336051
SP            0.395627
WT            0.400574
dtype: float64
```

In [11]:
```
#t and p-Values
print(model.tvalues, '\n', model.pvalues)
```

```
Intercept    2.058841
HP          -5.238735
VOL         -0.590970
SP           2.499880
WT           0.236541
dtype: float64
 Intercept    0.042936
HP           0.000001
VOL          0.556294
SP           0.014579
WT           0.813649
dtype: float64
```

In [12]:
```
#R squared values
(model.rsquared,model.rsquared_adj)
```

Out[12]: (0.7705372737359844, 0.7584602881431415)

Calculating VIF

In [13]:
```
rsq_hp = smf.ols('HP~WT+VOL+SP',data=cars).fit().rsquared
vif_hp = 1/(1-rsq_hp) # 16.33

rsq_wt = smf.ols('WT~HP+VOL+SP',data=cars).fit().rsquared
vif_wt = 1/(1-rsq_wt) # 564.98

rsq_vol = smf.ols('VOL~WT+SP+HP',data=cars).fit().rsquared
vif_vol = 1/(1-rsq_vol) #  564.84

rsq_sp = smf.ols('SP~WT+VOL+HP',data=cars).fit().rsquared
vif_sp = 1/(1-rsq_sp) #  16.35

# Storing vif values in a data frame
d1 = {'Variables':['Hp','WT','VOL','SP'],'VIF':[vif_hp,vif_wt,vif_vol,vif_sp]}
Vif_frame = pd.DataFrame(d1)
Vif_frame
```

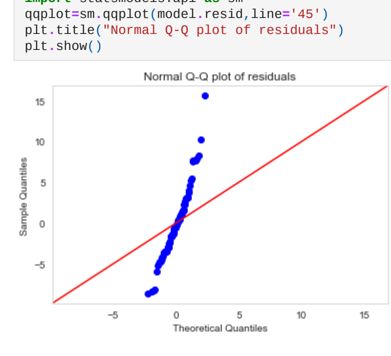
Out[13]:

	Variables	VIF
0	Hp	19.926589
1	WT	639.533818
2	VOL	638.806084
3	SP	20.007639

Module Validation

Test for Normality of Residuals (Q-Q Plot)

In [18]:
```
#Q-Q plot
import statsmodels.api as sm
qqplot=sm.qqplot(model.resid,line='45')
plt.title("Normal Q-Q plot of residuals")
plt.show()
```

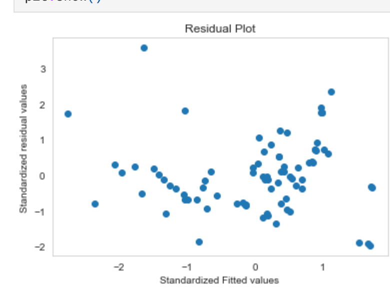


Residual Plot for Homoscedasticity

In [19]:
```
def get_standardized_values( vals ):
    return (vals - vals.mean())/vals.std()
```

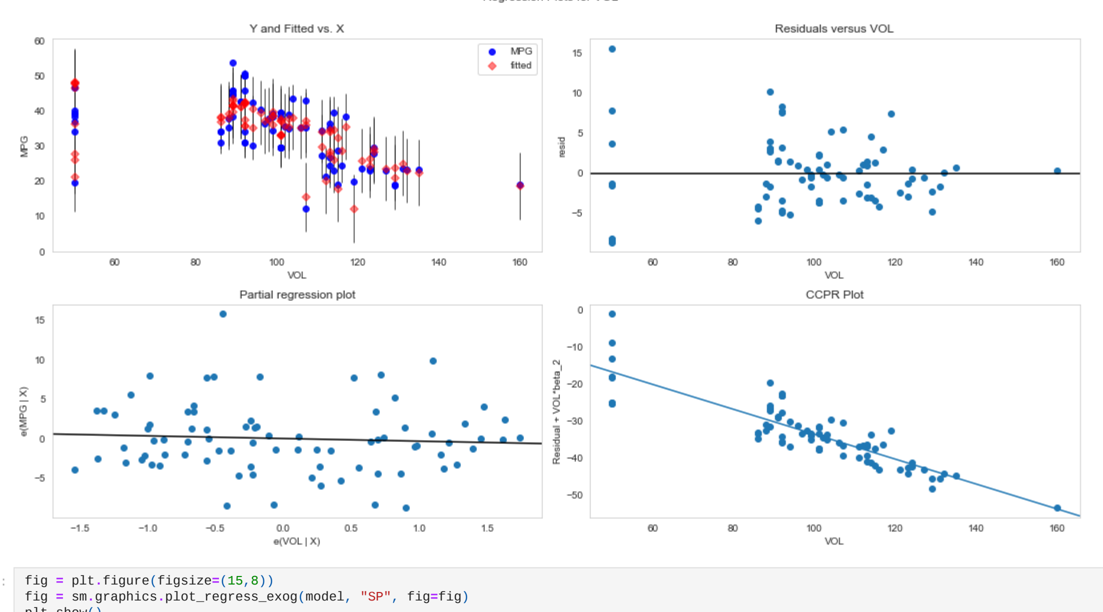
In [20]:
```
plt.scatter(get_standardized_values(model.fittedvalues),
            get_standardized_values(model.resid))

plt.title('Residual Plot')
plt.xlabel('Standardized Fitted values')
plt.ylabel('Standardized residual values')
plt.show()
```

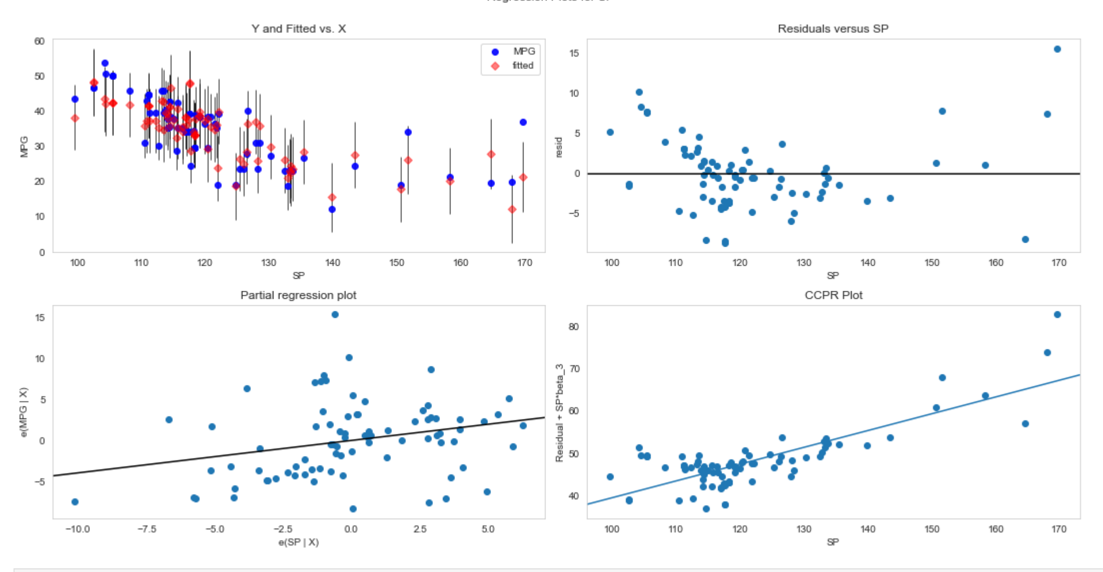


Residual Vs Regressors

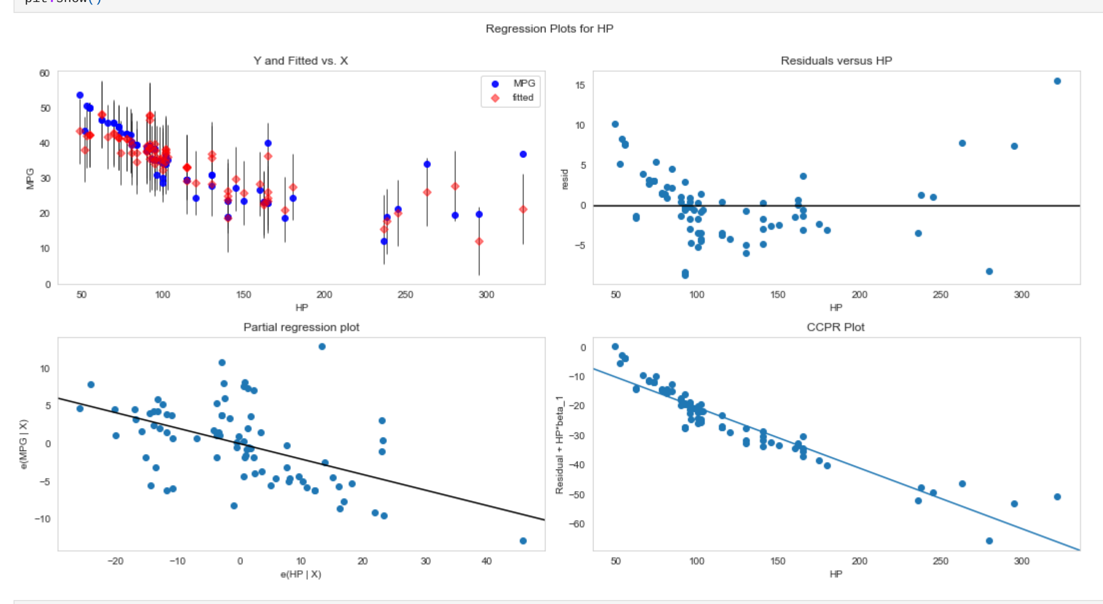
In [21]:
```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "VOL", fig=fig)
plt.show()
```



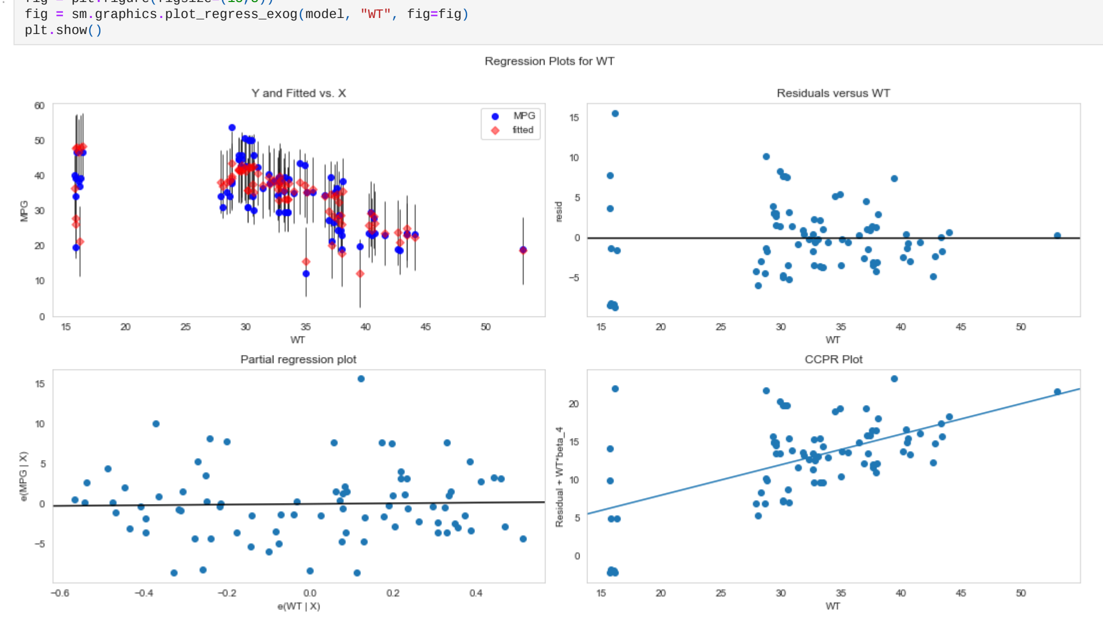
In [22]:
```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "SP", fig=fig)
plt.show()
```



In [23]:
```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "HP", fig=fig)
plt.show()
```



In [24]:
```
fig = plt.figure(figsize=(15,8))
fig = sm.graphics.plot_regress_exog(model, "WT", fig=fig)
plt.show()
```



Model Deletion Diagnostics

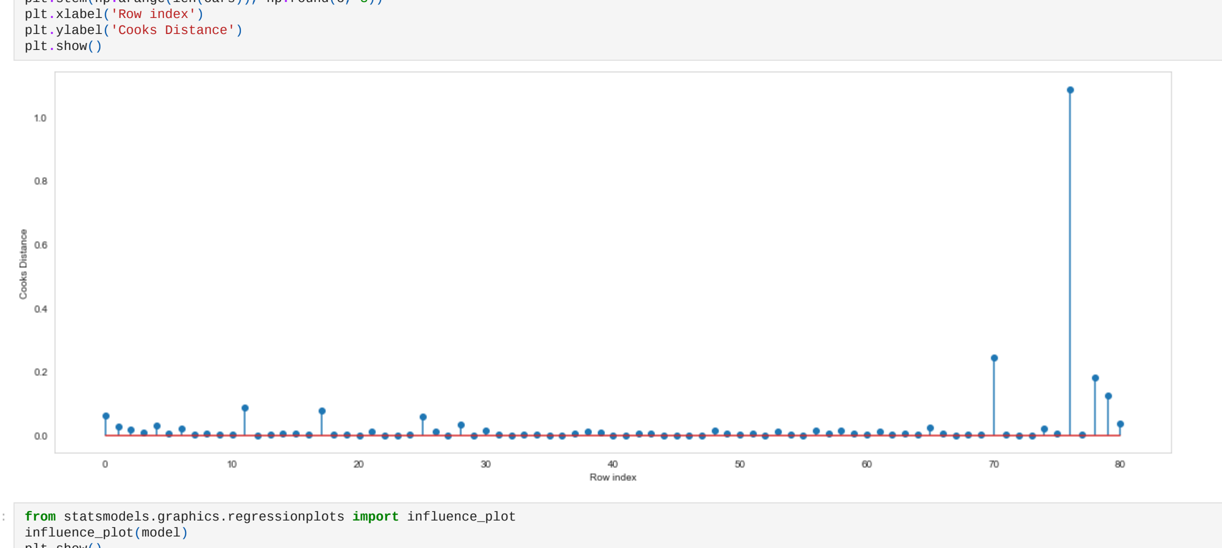
1]INFLUENCE

2]Cook's Distance

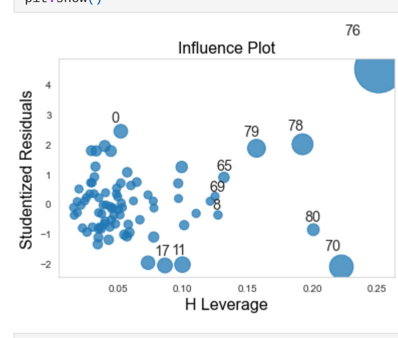
3] High Influence Points

In [25]:
```
model_influence = model.get_influence()
(c, _) = model_influence.cooks_distance
```

In [26]:
```
#Plot the influencers values using stem plot
fig = plt.subplots(figsize=(20, 7))
plt.stem(np.arange(len(cars)), np.round(c, 3))
plt.xlabel('Row index')
plt.ylabel('Cooks Distance')
plt.show()
```



In [27]:
```
from statsmodels.graphics.regressionplots import influence_plot
influence_plot(model)
plt.show()
```



In [28]:
```
k = cars.shape[1]
n = cars.shape[0]
leverage_cutoff = 3*((k + 1)/n)
```

From the above plot, it is evident that data point 70 and 76 are the influencers¶

In [48]:
```
cars1=cars.drop(cars.index[[70,76,80,78,79]],axis=0)
cars1
```

Out[48]:

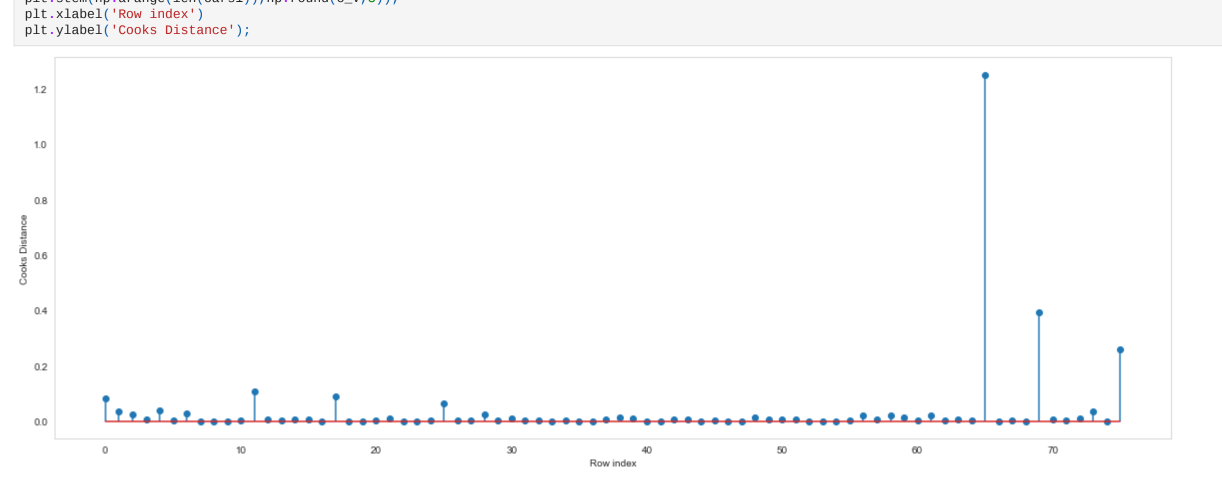
	HP	MPG	VOL	SP	WT
0	49	53.700681	89	104.185353	28.762059
1	55	50.013401	92	105.461264	30.466833
2	55	50.013401	92	105.461264	30.193597
3	70	45.696322	92	113.461264	30.632114
4	53	50.504232	92	104.461264	29.889149
...	...	...	...	...	...
72	162	23.203569	132	133.140074	43.353123
73	140	19.086341	160	124.715241	52.997752
74	140	19.086341	129	121.864163	42.618698
75	175	18.762837	129	132.864163	42.778219
77	238	19.197888	115	150.576579	37.923113

76 rows × 5 columns

In [49]:
```
#again build new model
model1=smf.ols('MPG~HP+VOL+SP+WT',data=cars1).fit()
```

In [50]:
```
#Again check for influencers
model_influence_V = model1.get_influence()
(c_V, _) = model_influence_V.cooks_distance
```

In [51]:
```
fig= plt.subplots(figsize=(20,7))
plt.stem(np.arange(len(cars1)),np.round(c_V,3));
plt.xlabel('Row index')
plt.ylabel('Cooks Distance');
```



In [52]:
```
#check for accuracy
(model1.rsquared,model1.aic)
```

Out[52]: (0.8569558504981126, 406.0655455898309)

Predicting for new data

In [54]:
```
#New data for prediction
new_data=pd.DataFrame({'HP':53,"VOL":92,"SP":104,"WT":29},index=[1])
```

In [55]:
```
model1.predict(new_data)
```

Out[55]:
```
1    43.736808
dtype: float64
```

In [56]:
```
model1.predict(cars1.iloc[0:5,])
```

Out[56]:
```
0    45.496455
1    44.169166
2    44.115832
3    43.867092
4    44.133189
dtype: float64
```

In [58]:
```
pred_y = model1.predict(cars1)
pred_y
```

Out[58]:
```
0     45.496455
1     44.169166
2     44.115832
3     43.867092
4     44.133189
        ...
72    22.145207
73    20.545911
74    23.310018
75    18.857466
77    11.615921
Length: 76, dtype: float64
```

In [59]:
```
model.summary()
```

Out[59]:

OLS Regression Results

Dep. Variable:	MPG	R-squared:	0.771
Model:	OLS	Adj. R-squared:	0.758
Method:	Least Squares	F-statistic:	63.80
Date:	Sat, 22 May 2021	Prob (F-statistic):	1.54e-23
Time:	22:17:55	Log-Likelihood:	-233.96
No. Observations:	81	AIC:	477.9
Df Residuals:	76	BIC:	489.9
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	30.6773	14.900	2.059	0.043	1.001	60.354
HP	-0.2054	0.039	-5.239	0.000	-0.284	-0.127
VOL	-0.3361	0.569	-0.591	0.556	-1.469	0.796
SP	0.3956	0.158	2.500	0.015	0.080	0.711
WT	0.4006	1.693	0.237	0.814	-2.972	3.773

Omnibus:	10.780	Durbin-Watson:	1.403
Prob(Omnibus):	0.005	Jarque-Bera (JB):	11.722
Skew:	0.707	Prob(JB):	0.00285
Kurtosis:	4.215	Cond. No.	6.09e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.09e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [ ]: