Linear Regression – Class Assessment

Dataset:Fish

In [1]:

```
'''
Name- Mayur V Kolki (PGA14)


'''
```

Out[1]:

```
'\nName- Mayur V Kolki (PGA14)\n\n\n'
```

In [2]:

```
#import libraries
import pandas as pd
import numpy as np
```

In [3]:

```
#cross- validation

from sklearn.model_selection import train_test_split ,KFold
```

In [4]:

```
#OLS library for linear regression
import statsmodels.api as sm
```

In [5]:

```
from sklearn.preprocessing import LabelEncoder
```

In [6]:

```
#visualisation
import pylab
import matplotlib.pyplot as plt
import seaborn as sns
```

In [7]:

```
from sklearn.metrics import mean_squared_error
```

In [8]:

```
import scipy.stats as stats
```

In [9]:

```
#VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [10]:

```
#feature selection for regression [specific]
from sklearn.feature_selection import  f_regression
```

In [ ]:

```

```

In [11]:

```
#read the data
path ="C:/Users/mayur/Desktop/datascience DELL/pythonstorage/dataset_ML/LinearRegressionusingPython/Linear Regression using Python/Fish_dataset.csv"
fish=pd.read_csv(path)
```

In [12]:

```
fish.head()
```

Out[12]:

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340

In [13]:

```
fish.tail()
# Here we found that the data is arrange in a order so need to shuffle the data for better underst
anding.
```

Out[13]:

	Species	Weight	Length1	Length2	Length3	Height	Width
154	Smelt	12.2	11.5	12.2	13.4	2.0904	1.3936
155	Smelt	13.4	11.7	12.4	13.5	2.4300	1.2690
156	Smelt	12.2	12.1	13.0	13.8	2.2770	1.2558
157	Smelt	19.7	13.2	14.3	15.2	2.8728	2.0672
158	Smelt	19.9	13.8	15.0	16.2	2.9322	1.8792

In [14]:

```
# We have to shuffle the data
fish = fish.sample(frac=1)
```

In [15]:

```
fish.head(15)
```

Out[15]:

	Species	Weight	Length1	Length2	Length3	Height	Width
65	Parkki	150.0	18.4	20.0	22.4	8.8928	3.2928
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
61	Parkki	55.0	13.5	14.7	16.5	6.8475	2.3265
74	Perch	40.0	13.8	15.0	16.0	3.8240	2.4320
18	Bream	610.0	30.9	33.5	38.6	15.6330	5.1338
93	Perch	145.0	20.7	22.7	24.2	5.9532	3.6300
92	Perch	150.0	20.5	22.5	24.0	6.7920	3.6240

	Species	Weight	Length1	Length2	Length3	Height	Width
9	Bream	500.0	28.5	30.7	36.2	14.2266	4.9594
116	Perch	900.0	36.5	39.0	41.4	11.1366	7.4934
54	Roach	390.0	29.5	31.7	35.0	9.4850	5.3550
89	Perch	135.0	20.0	22.0	23.5	5.8750	3.5250
121	Perch	1015.0	37.0	40.0	42.4	12.3808	7.4624
5	Bream	450.0	26.8	29.7	34.7	13.6024	4.9274
8	Bream	450.0	27.6	30.0	35.1	14.0049	4.8438
90	Perch	110.0	20.0	22.0	23.5	5.5225	3.9950

In [16]:

```
fish.columns
```

Out[16]:

```
Index(['Species', 'Weight', 'Length1', 'Length2', 'Length3', 'Height',
       'Width'],
      dtype='object')
```

In [17]:

```
#data summary
fish.describe(include="all")
```

Out[17]:

	Species	Weight	Length1	Length2	Length3	Height	Width
count	159	159.000000	159.000000	159.000000	159.000000	159.000000	159.000000
unique	7	NaN	NaN	NaN	NaN	NaN	NaN
top	Perch	NaN	NaN	NaN	NaN	NaN	NaN
freq	56	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	398.326415	26.247170	28.415723	31.227044	8.970994	4.417486
std	NaN	357.978317	9.996441	10.716328	11.610246	4.286208	1.685804
min	NaN	0.000000	7.500000	8.400000	8.800000	1.728400	1.047600
25%	NaN	120.000000	19.050000	21.000000	23.150000	5.944800	3.385650
50%	NaN	273.000000	25.200000	27.300000	29.400000	7.786000	4.248500
75%	NaN	650.000000	32.700000	35.500000	39.650000	12.365900	5.584500
max	NaN	1650.000000	59.000000	63.400000	68.000000	18.957000	8.142000

In [18]:

```
fish.dtypes
```

Out[18]:

```
Species      object
Weight      float64
Length1     float64
Length2     float64
Length3     float64
Height      float64
Width       float64
dtype: object
```

In [19]:

```
# Now Check the nulls and zeros in data set.
fish.isnull().sum()
```

Out[19]:

```
Species    0
Weight     0
Length1    0
Length2    0
Length3    0
Height     0
Width      0
dtype: int64
```

In [20]:

```
fish[fish==0].count()
# Here we found that there is a zero in a Weight feature (and it is a invalid zero we have to impu
te it)
```

Out[20]:

```
Species    0
Weight     1
Length1    0
Length2    0
Length3    0
Height     0
Width      0
dtype: int64
```

In [21]:

```
print(fish.loc[fish['Weight']== 0])
```

```
   Species  Weight  Length1  Length2  Length3  Height   Width
40   Roach     0.0     19.0     20.5     22.8  6.4752  3.3516
```

In [22]:

```
col = ['Species' , 'Weight']
fish[col][fish.Weight==0]
#Here we found that the Roach is a Species where wieght is a zero now can impute the
#mean wieght of Roach Species.
```

Out[22]:

	Species	Weight
40	Roach	0.0

In [23]:

```
fish[col][fish.Species=='Roach'].mean()
# Here we found that the mean of wieght is 152.05 when Species is Roach.
```

Out[23]:

```
Weight    152.05
dtype: float64
```

In [24]:

```
# Now store the mean in a object and then impute it.
mean_imp = fish[col][fish.Species=='Roach'].mean()
mean_imp
```

Out[24]:

```
Weight    152.05
dtype: float64
```

In [25]:

```
fish[fish.Weight==0] = fish[fish.Weight==0].replace(0,152.05)
```

In [26]:

```
#check for 0
fish[fish==0].count()
```

Out[26]:

```
Species    0
Weight     0
Length1    0
Length2    0
Length3    0
Height     0
Width      0
dtype: int64
```

In [27]:

```
#pd.set_option("display.max.rows",None)
```

In [28]:

```
print(fish.loc[40])
```

```
Species     Roach
Weight     152.05
Length1        19
Length2      20.5
Length3      22.8
Height     6.4752
Width      3.3516
Name: 40, dtype: object
```

In [29]:

```
fish.shape
```

Out[29]:

```
(159, 7)
```

In [30]:

```
 fish.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 159 entries, 65 to 50
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Species  159 non-null    object 
 1   Weight   159 non-null    float64
 2   Length1  159 non-null    float64
 3   Length2  159 non-null    float64
 4   Length3  159 non-null    float64
 5   Height   159 non-null    float64
 6   Width    159 non-null    float64
dtypes: float64(6), object(1)
memory usage: 14.9+ KB
```

In [31]:

```
fish.describe(include='all')
```

Out[31]:

	Species	Weight	Length1	Length2	Length3	Height	Width
count	159	159.000000	159.000000	159.000000	159.000000	159.000000	159.000000
unique	7	NaN	NaN	NaN	NaN	NaN	NaN
top	Perch	NaN	NaN	NaN	NaN	NaN	NaN
freq	56	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	399.282704	26.247170	28.415723	31.227044	8.970994	4.417486
std	NaN	357.109544	9.996441	10.716328	11.610246	4.286208	1.685804
min	NaN	5.900000	7.500000	8.400000	8.800000	1.728400	1.047600
25%	NaN	122.500000	19.050000	21.000000	23.150000	5.944800	3.385650
50%	NaN	273.000000	25.200000	27.300000	29.400000	7.786000	4.248500
75%	NaN	650.000000	32.700000	35.500000	39.650000	12.365900	5.584500
max	NaN	1650.000000	59.000000	63.400000	68.000000	18.957000	8.142000

In [32]:

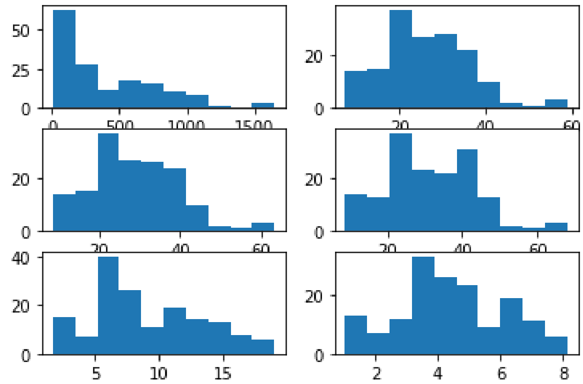
```
fish.dtypes
#we see that there are no more missing values
#"Species"  class we have to use one hot encoding before building the model.
```

Out[32]:

```
Species      object
Weight      float64
Length1     float64
Length2     float64
Length3     float64
Height      float64
Width       float64
dtype: object
```

In [33]:

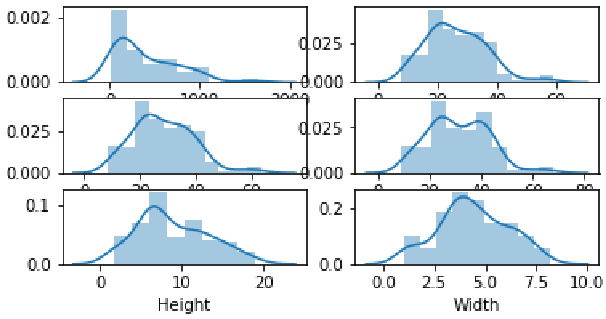
```
# Save coloumn names in a attribute
cols = list(fish.columns)
cols.remove('Species')
# Distribution Plot
nrow = 3 ; ncol=2 ; npos=1
fig= plt.figure()
for c in cols:
 fig.add_subplot(nrow,ncol,npos)
 plt.hist(fish[c])
 npos+=1
```



In [34]:

```
# Dist Plot
nrow = 4 ; ncol=2 ; npos=1
fig= plt.figure()
for c in cols:
 fig.add_subplot(nrow,ncol,npos)
 sns.distplot(fish[c])
```

```
npos+=1
```



Agistino-Person test for normality

H0: normal distribution H1: not a normal distribution

In [35]:

```
from scipy.stats import normaltest

#create a K-v  pair to store column names and its corresponding distribution type (Normal/NOt norm
al)

aptest = {}

for c in cols:
    tstat,pval = normaltest(fish[c])
    if pval< 0.05:
        aptest[c] = "not Normal Test"
    else:
        aptest[c] = "Normal"
```

In [36]:

```
aptest # Here we found that Weight , length3 , and width are only normally distributed others are
not.
```

Out[36]:

```
{'Weight': 'not Normal Test',
 'Length1': 'not Normal Test',
 'Length2': 'not Normal Test',
 'Length3': 'Normal',
 'Height': 'not Normal Test',
 'Width': 'Normal'}
```

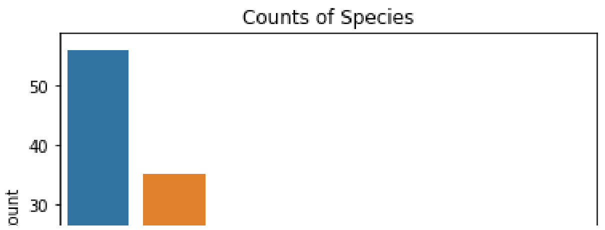
Q.1 Plot a bar chart showing count of individual species?

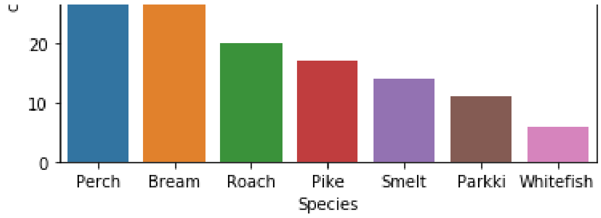
In [37]:

```
#Q.1 Plot a bar chart showing count of individual species?
sns.countplot(x='Species' , data= fish , order = fish['Species'].value_counts().index)
plt.title('Counts of Species')
```

Out[37]:

```
Text(0.5, 1.0, 'Counts of Species')
```



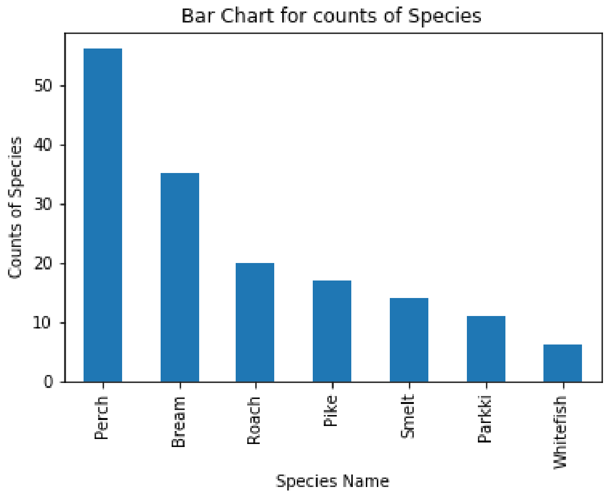


In [38]:

```
fish['Species'].value_counts().plot(kind='bar')
plt.xlabel('Species Name')
plt.ylabel('Counts of Species')
plt.title('Bar Chart for counts of Species')
```

Out[38]:

```
Text(0.5, 1.0, 'Bar Chart for counts of Species')
```

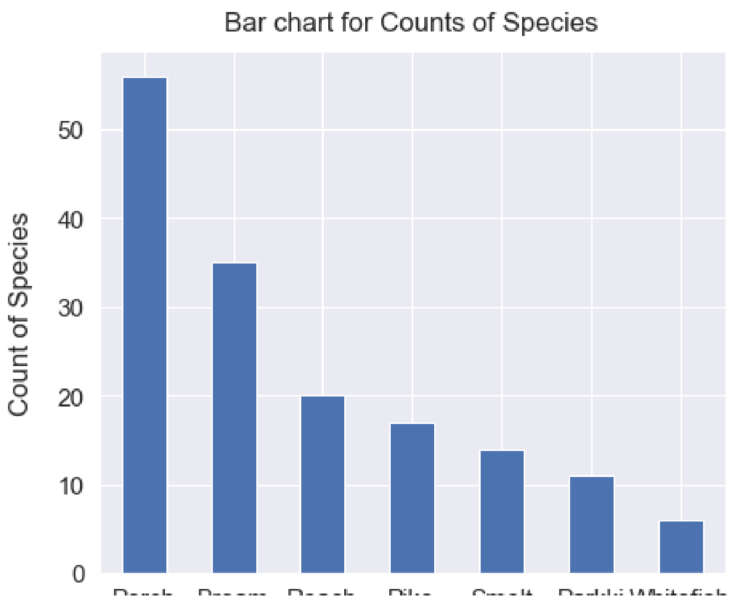


In [39]:

```
sns.set(font_scale=1.4)
fish['Species'].value_counts().plot(kind='bar', figsize=(7, 6), rot=0)
plt.xlabel("Species", labelpad=14)
plt.ylabel("Count of Species", labelpad=14)
plt.title("Bar chart for Counts of Species", y=1.02)
```

Out[39]:

```
Text(0.5, 1.02, 'Bar chart for Counts of Species')
```

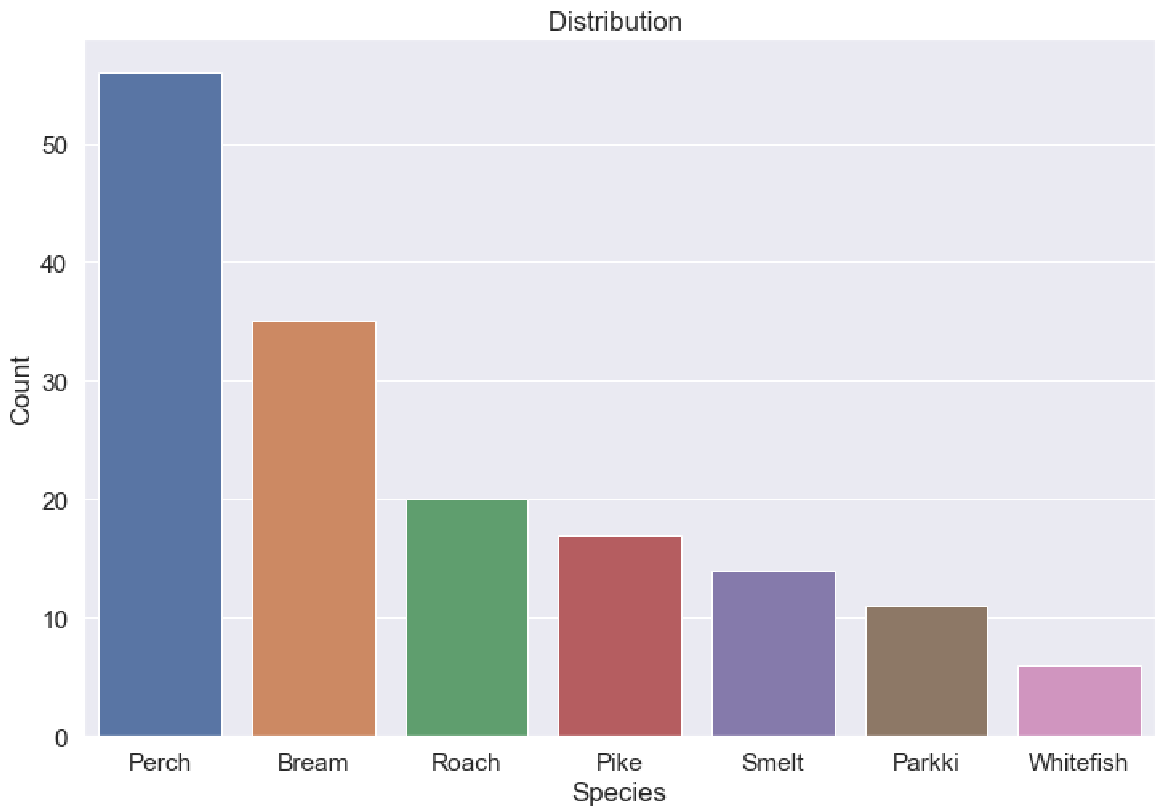


In [40]:

```
plt.figure(figsize=(12,8))
ax = sns.countplot(x="Species", data=fish , order = fish['Species'].value_counts().index)
plt.title('Distribution ')
plt.xlabel('Species')
plt.ylabel('Count')
```

Out[40]:

```
Text(0, 0.5, 'Count')
```



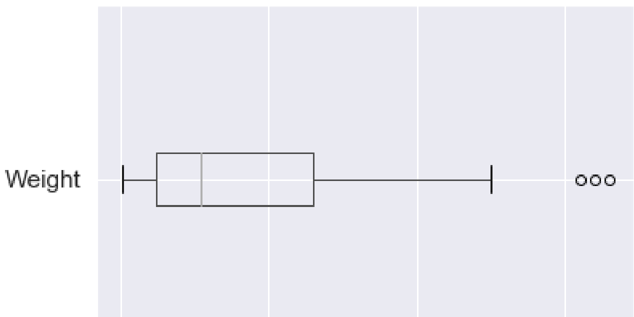
Q.2) Identify outliers and remove if any?

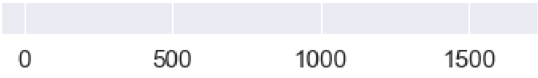
In [41]:

```
# Q.2) Identify outliers and remove if any?
# We have many method to find out outliers
# Lets Visualize the outlier with the help of boxplot.
#outliers
fish.boxplot('Weight',vert=False) # Here we found that there are 3 outlier lets check others also.
```

Out[41]:

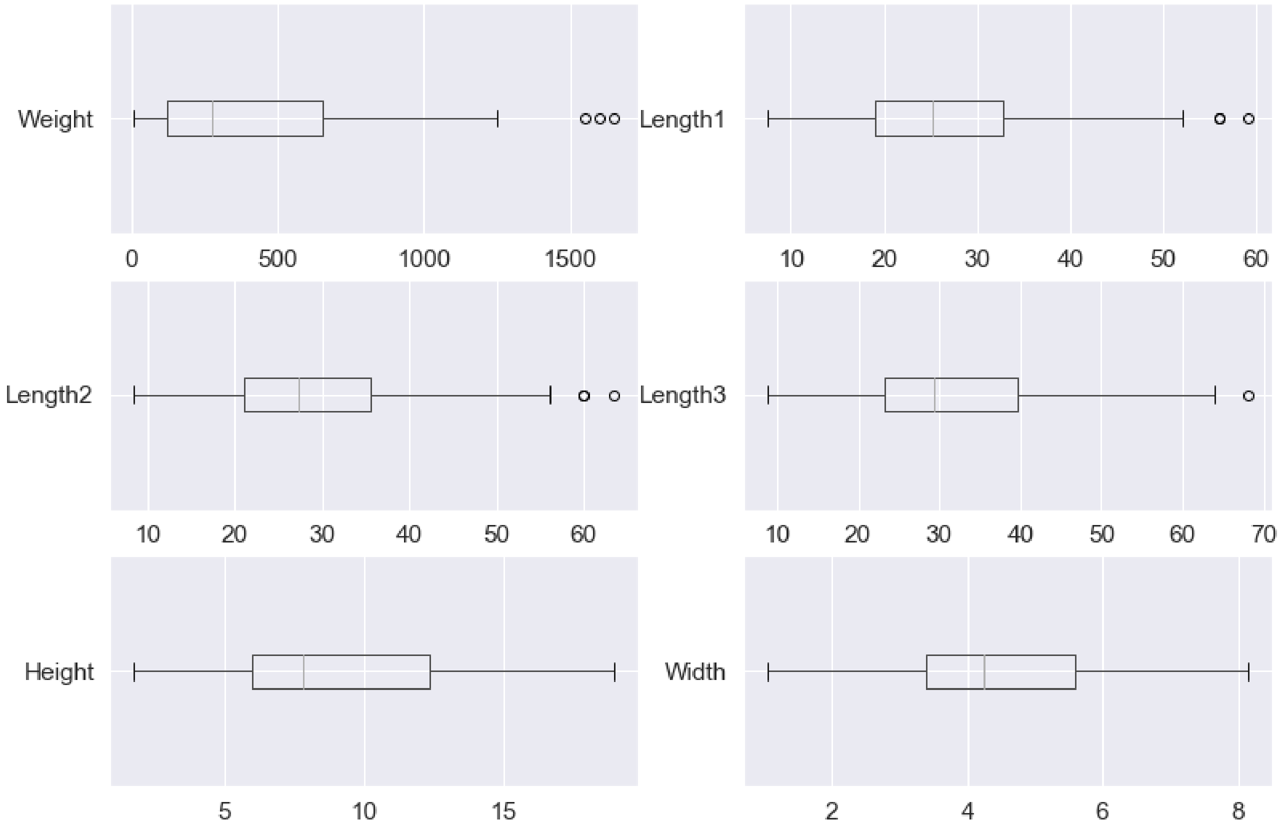
```
<matplotlib.axes._subplots.AxesSubplot at 0x2b5af6f2e88>
```





In [42]:

```
# Now check outlier in a loop
nrow = 3;ncol=2;npos=1
fig = plt.figure(figsize =(13,9))
for c in cols:
    fig.add_subplot(nrow,ncol,npos)
    fish.boxplot(c , vert=False)
    npos+=1
```



here we found that "Weight" ,"length1" ,"length2" and "length3" have some outliers we have to fix it.

In [43]:

```
# lets use IQR Method to find out outlier.
Q1 = fish.quantile(0.25)
Q3 = fish.quantile(0.75)
IQR = Q3 - Q1
print(IQR) # Here we got the IQR Of each coloumn so lets use it for next opration.
```

```
Weight      527.50000
Length1      13.65000
Length2      14.50000
Length3      16.50000
Height        6.42110
Width         2.19885
dtype: float64
```

In [44]:

```
# Lets find out outlier in datasets.
outlier = (fish < (Q1 - 1.5 * IQR)) | (fish > (Q3 + 1.5 * IQR))
```

In [45]:

```
outlier[outlier.Weight==True] # Here we found that there is a outlier in Wieght at index of 142,143,144.
```

Out[45]:

	Height	Length1	Length2	Length3	Species	Weight	Width
144	False	True	True	True	False	True	False
142	False	True	True	False	False	True	False
143	False	True	True	False	False	True	False

In [46]:

```
outlier[outlier.Length1==True] # Here we found that there is a outlier in Length1 at index of 142,143,144.
```

Out[46]:

	Height	Length1	Length2	Length3	Species	Weight	Width
144	False	True	True	True	False	True	False
142	False	True	True	False	False	True	False
143	False	True	True	False	False	True	False

In [47]:

```
outlier[outlier.Height==True] # No any outliers
```

Out[47]:

Height	Length1	Length2	Length3	Species	Weight	Width

In [48]:

```
outlier[outlier.Length2==True] # Here we found that there is a outlier in Length2 at index of 142,143,144.
```

Out[48]:

	Height	Length1	Length2	Length3	Species	Weight	Width
144	False	True	True	True	False	True	False
142	False	True	True	False	False	True	False
143	False	True	True	False	False	True	False

In [49]:

```
outlier[outlier.Length3==True] # Here we found that there is a outlier in Length3 at index of 144.
```

Out[49]:

	Height	Length1	Length2	Length3	Species	Weight	Width
144	False	True	True	True	False	True	False

In [50]:

```
outlier[outlier.Width==True] # No any outlier
```

Out[50]:

Height	Length1	Length2	Length3	Species	Weight	Width

Here we have to decide that we want to impute it or remove it .

give in description -->Identify outliers and remove if any

In [51]:

```
# calculate the outlier cutoff
cut_off = IQR * 1.5
lower = Q1 - cut_off
upper = Q3 + cut_off
```

In [52]:

```
lower
```

Out[52]:

```
Weight    -668.750000
Length1     -1.425000
Length2     -0.750000
Length3     -1.600000
Height      -3.686850
Width        0.087375
dtype: float64
```

In [53]:

```
upper
```

Out[53]:

```
Weight     1441.250000
Length1      53.175000
Length2      57.250000
Length3      64.400000
Height       21.997550
Width         8.882775
dtype: float64
```

In [54]:

```
cols
```

Out[54]:

```
['Weight', 'Length1', 'Length2', 'Length3', 'Height', 'Width']
```

In [55]:

```
# lets remove the outliers from the data
new_fish = fish[~((fish < (Q1 - 1.5 * IQR)) | (fish > (Q3 + 1.5 * IQR))).any(axis=1)]
```

In [56]:

```
#lets Verify the data
new_fish.shape # here we see that the outlier is removed from the data set and we we remaining dat
a.
```

Out[56]:

```
(156, 7)
```

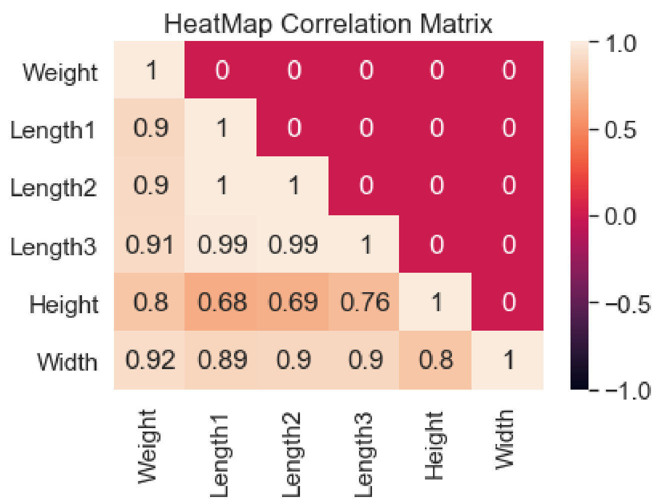
In [57]:

```
# we have to check the multicolinearity in data.
```

```
# Checking for Multicolinearity.
# Correlation Matrix take only the lower triangle
cor = new_fish[cols].corr()
cor = np.tril(cor)
sns.heatmap(cor , xticklabels = cols , yticklabels = cols ,
 vmin = -1 , vmax = 1 , annot=True , square = False)
plt.title('HeatMap Correlation Matrix')
```

Out[57]:

```
Text(0.5, 1, 'HeatMap Correlation Matrix')
```



here we found that our Y variable is highly corelated with X-variables

which is good for us, but length3 is highly corelated with length2 , length1 and width .

So , we can drop the Length3 .

In [58]:

```
#split the columns into nc and fc
fc=new_fish.select_dtypes(include='object').columns.values

nc=new_fish.select_dtypes(exclude='object').columns.values
```

In [59]:

```
fc
```

Out[59]:

```
array(['Species'], dtype=object)
```

In [60]:

```
nc
```

Out[60]:

```
array(['Weight', 'Length1', 'Length2', 'Length3', 'Height', 'Width'],
      dtype=object)
```

In [61]:

```
# We have to use one hot encoding for handle the factor variable coz it required numeric data for linear regression model.
dummy=pd.get_dummies(new_fish.Species,drop_first=True,prefix='Species')
new_fish= new_fish.join(dummy)
```

In [62]:

```
new_fish.head(10)
```

Out[62]:

	Species	Weight	Length1	Length2	Length3	Height	Width	Species_Parkki	Species_Perch	Species_Pike	Species_Roach	Spe
65	Parkki	150.0	18.4	20.0	22.4	8.8928	3.2928	1	0	0	0	
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056	0	0	0	0	
61	Parkki	55.0	13.5	14.7	16.5	6.8475	2.3265	1	0	0	0	
74	Perch	40.0	13.8	15.0	16.0	3.8240	2.4320	0	1	0	0	
18	Bream	610.0	30.9	33.5	38.6	15.6330	5.1338	0	0	0	0	
93	Perch	145.0	20.7	22.7	24.2	5.9532	3.6300	0	1	0	0	
92	Perch	150.0	20.5	22.5	24.0	6.7920	3.6240	0	1	0	0	
9	Bream	500.0	28.5	30.7	36.2	14.2266	4.9594	0	0	0	0	
116	Perch	900.0	36.5	39.0	41.4	11.1366	7.4934	0	1	0	0	
54	Roach	390.0	29.5	31.7	35.0	9.4850	5.3550	0	0	0	1	

In [63]:

```
# Now drop the original column from dataset
new_fish = new_fish.drop('Species' , axis = 1)
```

In [64]:

```
# Verify the result
new_fish
```

Out[64]:

	Weight	Length1	Length2	Length3	Height	Width	Species_Parkki	Species_Perch	Species_Pike	Species_Roach	Species_Sme
65	150.0	18.4	20.0	22.4	8.8928	3.2928	1	0	0	0	
1	290.0	24.0	26.3	31.2	12.4800	4.3056	0	0	0	0	
61	55.0	13.5	14.7	16.5	6.8475	2.3265	1	0	0	0	
74	40.0	13.8	15.0	16.0	3.8240	2.4320	0	1	0	0	
18	610.0	30.9	33.5	38.6	15.6330	5.1338	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	
72	5.9	7.5	8.4	8.8	2.1120	1.4080	0	1	0	0	
150	8.7	10.8	11.3	12.6	1.9782	1.2852	0	0	0	0	
46	140.0	21.0	22.5	25.0	6.5500	3.3250	0	0	0	1	
69	200.0	21.2	23.0	25.8	10.3458	3.6636	1	0	0	0	
50	200.0	22.1	23.5	26.8	7.3968	4.1272	0	0	0	1	

156 rows × 12 columns

In [65]:

```
# Check the data type of dataset to verify that the dataset have all the numeric variable
new_fish.dtypes
```

Out[65]:

```
Weight            float64
Length1           float64
Length2           float64
Length3           float64
Height            float64
Width             float64
Species_Parkki      uint8
Species_Perch       uint8
Species_Pike        uint8
```

```
Species_Roach         uint8
Species_Smelt         uint8
Species_Whitefish     uint8
dtype: object
```

In [66]:

```
# CHeck some stat.
new_fish.describe(include = 'all')
```

Out[66]:

	Weight	Length1	Length2	Length3	Height	Width	Species_Parkki	Species_Perch	Species_Pike	Specie
count	156.000000	156.000000	156.000000	156.000000	156.000000	156.000000	156.000000	156.000000	156.000000	15
mean	376.191987	25.655769	27.786538	30.571154	8.951128	4.375719	0.070513	0.358974	0.089744	
std	318.625672	9.119630	9.792651	10.695359	4.324325	1.672188	0.256834	0.481245	0.286735	
min	5.900000	7.500000	8.400000	8.800000	1.728400	1.047600	0.000000	0.000000	0.000000	
25%	120.000000	19.000000	21.000000	23.025000	5.931675	3.369600	0.000000	0.000000	0.000000	
50%	271.000000	25.000000	26.750000	29.250000	7.647800	4.243300	0.000000	0.000000	0.000000	
75%	612.500000	32.125000	35.000000	39.425000	12.378550	5.424375	0.000000	1.000000	0.000000	
max	1250.000000	52.000000	56.000000	59.700000	18.957000	8.142000	1.000000	1.000000	1.000000	

In [ ]:

Q.3 Build a regression model and print regression equation?

In [67]:

```
# Now Split the data int train and test
trainx ,testx , trainy , testy = train_test_split(new_fish.drop('Weight' , axis=1),new_fish["Weight
"] , test_size=0.15)
```

In [68]:

```
print("trainx={},trainy={},testx={},testy ={}".format(trainx.shape,trainy.shape,testx.shape,testy.s
hape))
```

```
trainx=(132, 11),trainy=(132,),testx=(24, 11),testy =(24,)
```

Build the Regression Model by using OLS

In [69]:

```
trainx
```

Out[69]:

	Length1	Length2	Length3	Height	Width	Species_Parkki	Species_Perch	Species_Pike	Species_Roach	Species_Smelt	Speci
50	22.1	23.5	26.8	7.3968	4.1272	0	0	0	1	0	
46	21.0	22.5	25.0	6.5500	3.3250	0	0	0	1	0	
94	21.0	23.0	24.5	5.2185	3.6260	0	1	0	0	0	
104	25.4	27.5	28.9	7.0516	4.3350	0	1	0	0	0	
13	29.5	32.0	37.3	13.9129	5.0728	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	
35	12.9	14.1	16.2	4.1472	2.2680	0	0	0	1	0	
49	22.0	23.4	26.7	6.9153	3.6312	0	0	0	1	0	
81	18.2	20.0	21.0	5.0820	2.7720	0	1	0	0	0	
152	11.3	11.8	13.1	2.2139	1.1659	0	0	0	0	1	

	Length1	Length2	Length3	Height	Width	Species_Parkki	Species_Perch	Species_Pike	Species_Roach	Species_Smelt	Speci
92	20.5	22.5	24.0	6.7920	3.6240	0	1	0	0	0	

132 rows × 11 columns

In [70]:

```
trainy
```

Out[70]:

```
50     200.0
46     140.0
94     150.0
104    265.0
13     340.0
       ...
35      40.0
49     161.0
81      85.0
152      9.9
92     150.0
Name: Weight, Length: 132, dtype: float64
```

In [ ]:

```

```

In [71]:

```
m1 = sm.OLS(trainy , trainx).fit()
```

In [72]:

```
m1.summary()
```

Out[72]:

OLS Regression Results

Dep. Variable:	Weight	**R-squared (uncentered):**	0.971
Model:	OLS	**Adj. R-squared (uncentered):**	0.968
Method:	Least Squares	**F-statistic:**	365.8
Date:	Wed, 24 Mar 2021	**Prob (F-statistic):**	2.98e-87
Time:	23:02:47	**Log-Likelihood:**	-773.57
No. Observations:	132	**AIC:**	1569.
Df Residuals:	121	**BIC:**	1601.
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Length1	40.3074	36.663	1.099	0.274	-32.277	112.892
Length2	137.8173	44.353	3.107	0.002	50.010	225.625
Length3	-157.5933	22.077	-7.138	0.000	-201.301	-113.885
Height	16.9911	14.661	1.159	0.249	-12.035	46.017
Width	114.6926	22.304	5.142	0.000	70.536	158.849
Species_Parkki	-333.9652	30.963	-10.786	0.000	-395.264	-272.666
Species_Perch	-519.3576	45.901	-11.315	0.000	-610.231	-428.484
Species_Pike	-399.8925	125.230	-3.193	0.002	-647.817	-151.968
Species_Roach	-336.3474	41.828	-8.041	0.000	-419.157	-253.538
Species_Smelt	-229.4549	42.504	-5.398	0.000	-313.603	-145.306

Species_Whitefish	-319.6180	57.453	-5.563	0.000	-433.361	-205.875

Omnibus:	12.237	**Durbin-Watson:**	1.860
Prob(Omnibus):	0.002	**Jarque-Bera (JB):**	16.048
Skew:	0.532	**Prob(JB):**	0.000327
Kurtosis:	4.336	**Cond. No.**	976.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [73]:

```
# Add a constant term in train x and testx
# this will ensure that the model summary has the intercept term displayed.
trainx = sm.add_constant(trainx)
testx = sm.add_constant(testx)
```

In [74]:

```
m2 = sm.OLS(trainy,trainx).fit()
```

In [75]:

```
m2.summary()
```

Out[75]:

OLS Regression Results

Dep. Variable:	Weight	**R-squared:**	0.946
Model:	OLS	**Adj. R-squared:**	0.941
Method:	Least Squares	**F-statistic:**	192.3
Date:	Wed, 24 Mar 2021	**Prob (F-statistic):**	1.13e-70
Time:	23:02:47	**Log-Likelihood:**	-755.53
No. Observations:	132	**AIC:**	1535.
Df Residuals:	120	**BIC:**	1570.
Df Model:	11		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-697.8289	113.605	-6.143	0.000	-922.759	-472.899
Length1	-4.0034	32.912	-0.122	0.903	-69.167	61.160
Length2	50.9421	41.341	1.232	0.220	-30.910	132.795
Length3	-32.5057	28.082	-1.158	0.249	-88.106	23.095
Height	44.8251	13.617	3.292	0.001	17.864	71.786
Width	57.7089	21.626	2.669	0.009	14.891	100.526
Species_Parkki	44.4055	67.303	0.660	0.511	-88.850	177.661
Species_Perch	93.2345	107.527	0.867	0.388	-119.662	306.130
Species_Pike	121.9798	138.739	0.879	0.381	-152.714	396.674
Species_Roach	105.5654	80.733	1.308	0.194	-54.281	265.412
Species_Smelt	388.5245	107.273	3.622	0.000	176.132	600.917
Species_Whitefish	115.2613	86.859	1.327	0.187	-56.713	287.235

Omnibus:	35.576	**Durbin-Watson:**	1.926
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	71.888

Skew:	1.153	**Prob(JB):**	2.45e-16
Kurtosis:	5.785	**Cond. No.**	1.95e+03

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.95e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

Now here we found that there is a drop in a R sqr. after adding constant.

we can Improve our

model but it s not neccesary now.

We can use only significance variables.

we can remove multicolinearity.

We can use only VIF Function.

We can do boxcox transformation

We can do Log and minmax transformation.

But question is only for building a single model.

In [76]:

```
#Lets Check for Assumption..
#1) mean of residual should be zero
print(m1.resid.mean())
```

```
-2.4708688137832624
```

In [77]:

```
#Lets Check for Assumption..
#1) mean of residual should be zero
print(m2.resid.mean())
```

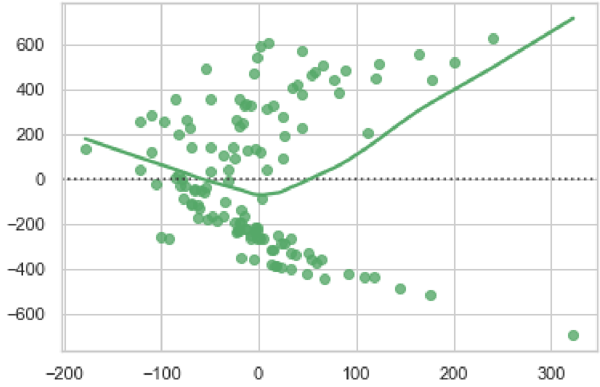
```
2.0247238229454854e-12
```

In [78]:

```
#ii) Residual have constant variance(homoscidasticity) # Lowess-> locally wieghted scatterplotsmotting.
# Plot the graph
yhat = m2.predict(trainx)
sns.set(style='whitegrid')
sns.residplot(m2.resid,yhat, color='g' , lowess=True) ## BAsed On graph we found that the model is homoscedasticity.
```

Out[78]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b5ae465b48>
```



In [79]:

```
# Bruesch - pagan test for finding the hetro and homo.
import statsmodels.stats.api as sms
# H0-> Homoscedasticity
# H1-> Hetrocedasticity
# return value of breushch test
# lagrange _ Multiplier , pvalue , fscore , fp-value
pval = sms.het_breuschpagan(m1.resid , m1.model.exog)[1]
```

In [80]:

```
if pval<0.05:
 print('Reject H0 , Model is Hetroscidasticity')
else:
 print('FTR H0 , Model is homoscedasticity')
```

```
Reject H0 , Model is Hetroscidasticity
```

In [81]:

```
# With Constant
pval = sms.het_breuschpagan(m2.resid , m2.model.exog)[1]
pval
```

Out[81]:

```
0.7085084851091408
```

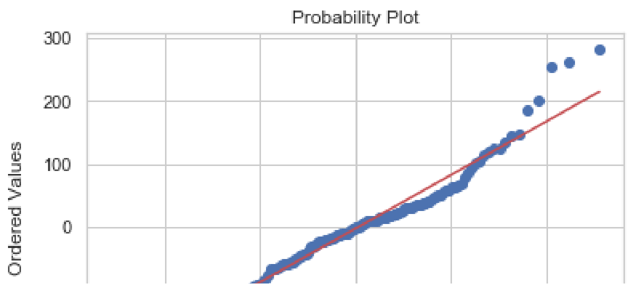
In [82]:

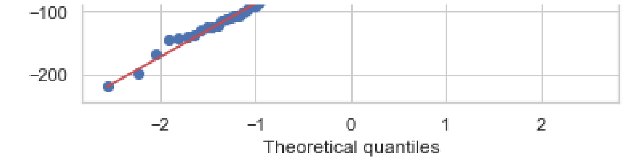
```
if pval<0.05:
 print('Reject H0 , Model is Hetroscidasticity')
else:
 print('FTR H0 , Model is homoscedasticity') # Model is homo when we add constant value.
```

```
FTR H0 , Model is homoscedasticity
```

In [83]:

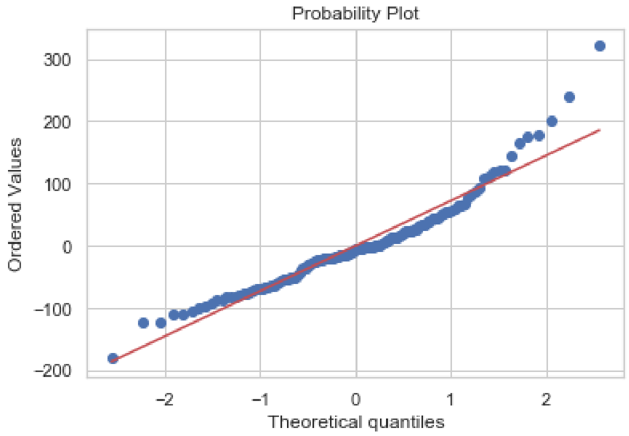
```
# 3) Check error is normally distributed or not.
stats.probplot(m1.resid , dist = 'norm' , plot=pylab)
pylab.show() # Not normally distributed
```





In [84]:

```
# 3) Check error is normally distributed or not. # with constant add.
stats.probplot(m2.resid , dist = 'norm' , plot=pylab)
pylab.show() # Not normally distributed
```



In [85]:

```
# K-Fold Cross Validation. we can not give the dataframe , we have to give array.
folds = 5
cv_mse = []
x = trainx.values
y = trainy.values
from sklearn.model_selection import KFold
kf = KFold(folds)
kf.get_n_splits(x)
for train_index , test_index in kf.split(x):
 print('train = ', train_index)
 print('test = ' , test_index)
 print('\n')

for train_index , test_index in kf.split(x):
 cv_trainx , cv_testx = x[train_index] , x[test_index]
 cv_trainy , cv_testy = y[train_index] , y[test_index]
 # Build the model
 m = sm.OLS(cv_trainy , cv_trainx).fit()
 p = m.predict(cv_testx)
 # Store the mse in the list of each model
 cv_mse.append(np.round(mean_squared_error(cv_testy , p),3))
```

```
train =  [ 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]
test =  [ 0  1  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]


train =  [  0   1   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  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]
test =  [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]
```

```
train =  [  0   1   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
  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]
test =  [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]


train =  [  0   1   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  79 106 107 108 109 110 111 112 113 114 115
 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131]
test =  [ 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]


train =  [  0   1   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]
test =  [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]
```

In [86]:

```
cv_mse
```

Out[86]:

```
[11009.135, 7184.885, 6344.48, 8499.78, 5099.217]
```

In [87]:

```
# Mean MSE_of K-FOLD CV
np.mean(cv_mse)
```

Out[87]:

```
7627.499399999999
```

In [88]:

```
# Predict on the test data
p1 = round(m2.predict(testx),1)
p1
```

Out[88]:

```
23      685.3
20      624.5
154       4.7
89      139.0
105     345.2
132     492.2
149     -17.6
124     912.2
17      579.6
130     334.1
76       -7.0
90      150.3
29      899.0
73     -161.0
146     -36.2
26      722.2
133     424.8
```

```
43      120.8
1       339.6
51      233.2
38       56.9
57      403.8
139     680.2
131     405.1
dtype: float64
```

In [89]:

```
# MSE of model 1
mse1 = round(mean_squared_error(testy , p1),3)
```

In [90]:

```
mse1
```

Out[90]:

```
5775.443
```

RMSE

In [91]:

```
import math

math.sqrt(mse1)
```

Out[91]:

```
75.996335437967
```

In [92]:

```
# Know campare the train and test error
print('Training MSE = {} , Testing MSE = {}' . format(np.mean(cv_mse) , mse1))
```

```
Training MSE = 7627.499399999999 , Testing MSE = 5775.443
```

In [93]:

```
#.5 Compare real and predicted weights and give a conclusion statement based on it?
df = pd.DataFrame({'Actual':testy , "Predicted":p1})
```

In [94]:

```
df
```

Out[94]:

	Actual	Predicted
23	680.0	685.3
20	575.0	624.5
154	12.2	4.7
89	135.0	139.0
105	250.0	345.2
132	430.0	492.2
149	9.8	-17.6
124	1000.0	912.2
17	700.0	579.6

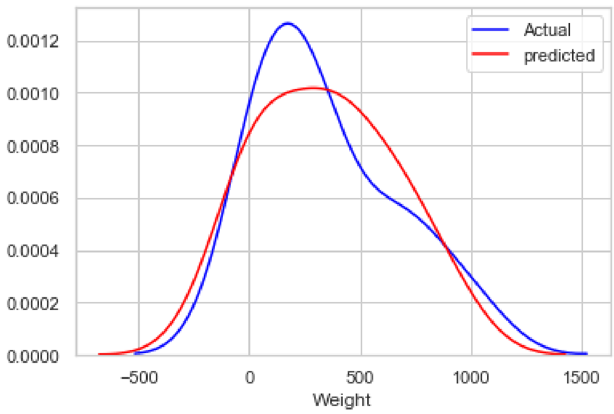
	Actual	Predicted
130	300.0	334.1
76	70.0	-7.0
90	110.0	150.3
29	1000.0	899.0
73	32.0	-161.0
146	7.5	-36.2
26	720.0	722.2
133	345.0	424.8
43	150.0	120.8
1	290.0	339.6
51	180.0	233.2
38	87.0	56.9
57	306.0	403.8
139	770.0	680.2
131	300.0	405.1

In [95]:

```
#plot the actual and predicted values
ax1=sns.distplot(testy,hist=False,color='blue',label ='Actual')
sns.distplot(p1,hist=False,color='red',label='predicted',ax=ax1)
```

Out[95]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b5ae420548>
```



In [96]:

```
#other considerations
#VIF (Variance Inflation Factor)
vif=pd.DataFrame()

vif['inflation']=[variance_inflation_factor(trainx.values,i)
for i in range (trainx.shape[1])]

vif['features'] =list(trainx.columns)

vif
```

Out[96]:

	inflation	features
0	282.422492	const
1	1921.023020	Length1
2	3488.043045	Length2
3	1914.515067	Length3

	inflation	features
4	71.831631	Height
5	28.748061	Width
6	7.571946	Species_Parkki
7	59.535474	Species_Perch
8	26.761004	Species_Pike
9	16.003276	Species_Roach
10	19.235876	Species_Smelt
11	6.016680	Species_Whitefish

In [97]:

```
new_fish
```

Out[97]:

	Weight	Length1	Length2	Length3	Height	Width	Species_Parkki	Species_Perch	Species_Pike	Species_Roach	Species_Sme
65	150.0	18.4	20.0	22.4	8.8928	3.2928	1	0	0	0	
1	290.0	24.0	26.3	31.2	12.4800	4.3056	0	0	0	0	
61	55.0	13.5	14.7	16.5	6.8475	2.3265	1	0	0	0	
74	40.0	13.8	15.0	16.0	3.8240	2.4320	0	1	0	0	
18	610.0	30.9	33.5	38.6	15.6330	5.1338	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	
72	5.9	7.5	8.4	8.8	2.1120	1.4080	0	1	0	0	
150	8.7	10.8	11.3	12.6	1.9782	1.2852	0	0	0	0	
46	140.0	21.0	22.5	25.0	6.5500	3.3250	0	0	0	1	
69	200.0	21.2	23.0	25.8	10.3458	3.6636	1	0	0	0	
50	200.0	22.1	23.5	26.8	7.3968	4.1272	0	0	0	1	

156 rows × 12 columns

Now we build the model by removing the higher multicolinear column "Length3"

In [98]:

```
new_fish1 = new_fish.drop("Length3",axis=1)
new_fish1
```

Out[98]:

	Weight	Length1	Length2	Height	Width	Species_Parkki	Species_Perch	Species_Pike	Species_Roach	Species_Smelt	Specie
65	150.0	18.4	20.0	8.8928	3.2928	1	0	0	0	0	
1	290.0	24.0	26.3	12.4800	4.3056	0	0	0	0	0	
61	55.0	13.5	14.7	6.8475	2.3265	1	0	0	0	0	
74	40.0	13.8	15.0	3.8240	2.4320	0	1	0	0	0	
18	610.0	30.9	33.5	15.6330	5.1338	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	
72	5.9	7.5	8.4	2.1120	1.4080	0	1	0	0	0	
150	8.7	10.8	11.3	1.9782	1.2852	0	0	0	0	1	
46	140.0	21.0	22.5	6.5500	3.3250	0	0	0	1	0	
69	200.0	21.2	23.0	10.3458	3.6636	1	0	0	0	0	
50	200.0	22.1	23.5	7.3968	4.1272	0	0	0	1	0	

156 rows × 11 columns

model 3

In [99]:

```
trainx,testx,trainy,testy =
train_test_split(new_fish1.drop('Weight',axis=1),new_fish1['Weight'],test_size =0.3)
```

In [100]:

```
print("trainx={},trainy={},testx={},testy ={}".format(trainx.shape,trainy.shape,testx.shape,testy.s
hape))
```

```
trainx=(109, 10),trainy=(109,),testx=(47, 10),testy =(47,)
```

In [101]:

```
m3= sm.OLS(trainy,trainx).fit()
```

In [102]:

```
m3.summary()
```

Out[102]:

OLS Regression Results

Dep. Variable:	Weight	**R-squared (uncentered):**	0.958
Model:	OLS	**Adj. R-squared (uncentered):**	0.954
Method:	Least Squares	**F-statistic:**	228.5
Date:	Wed, 24 Mar 2021	**Prob (F-statistic):**	1.03e-63
Time:	23:02:48	**Log-Likelihood:**	-656.43
No. Observations:	109	**AIC:**	1333.
Df Residuals:	99	**BIC:**	1360.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Length1	78.4650	52.247	1.502	0.136	-25.204	182.134
Length2	-51.1209	50.780	-1.007	0.317	-151.879	49.638
Height	-62.8298	13.124	-4.787	0.000	-88.870	-36.789
Width	163.9820	28.997	5.655	0.000	106.446	221.518
Species_Parkki	-234.7933	41.592	-5.645	0.000	-317.321	-152.266
Species_Perch	-498.8491	61.170	-8.155	0.000	-620.224	-377.474
Species_Pike	-713.5241	146.745	-4.862	0.000	-1004.698	-422.350
Species_Roach	-496.5510	50.818	-9.771	0.000	-597.385	-395.718
Species_Smelt	-346.0252	47.703	-7.254	0.000	-440.678	-251.372
Species_Whitefish	-439.2632	72.307	-6.075	0.000	-582.736	-295.791

Omnibus:	13.260	**Durbin-Watson:**	2.177
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	23.544
Skew:	0.494	**Prob(JB):**	7.72e-06
Kurtosis:	5.052	**Cond. No.**	719.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [103]:

```
# Add a constant term in train x and testx
# this will ensure that the model summary has the intercept term displayed.
trainx = sm.add_constant(trainx)
testx = sm.add_constant(testx)
```

In [104]:

```
m3=  sm.OLS(trainy,trainx).fit()
```

In [105]:

```
m3.summary()
```

Out[105]:

OLS Regression Results

Dep. Variable:	Weight	**R-squared:**	0.952
Model:	OLS	**Adj. R-squared:**	0.947
Method:	Least Squares	**F-statistic:**	194.9
Date:	Wed, 24 Mar 2021	**Prob (F-statistic):**	5.12e-60
Time:	23:02:48	**Log-Likelihood:**	-615.55
No. Observations:	109	**AIC:**	1253.
Df Residuals:	98	**BIC:**	1283.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-838.4959	80.126	-10.465	0.000	-997.502	-679.489
Length1	-26.5989	37.458	-0.710	0.479	-100.933	47.735
Length2	42.5473	36.198	1.175	0.243	-29.287	114.382
Height	39.8974	13.362	2.986	0.004	13.382	66.413
Width	44.9955	23.031	1.954	0.054	-0.708	90.699
Species_Parkki	110.4175	43.744	2.524	0.013	23.610	197.225
Species_Perch	187.0429	77.981	2.399	0.018	32.293	341.793
Species_Pike	139.9371	130.095	1.076	0.285	-118.233	398.107
Species_Roach	168.1519	72.571	2.317	0.023	24.137	312.167
Species_Smelt	488.5445	86.289	5.662	0.000	317.307	659.782
Species_Whitefish	119.6908	73.125	1.637	0.105	-25.423	264.805

Omnibus:	11.839	**Durbin-Watson:**	1.895
Prob(Omnibus):	0.003	**Jarque-Bera (JB):**	12.849
Skew:	0.684	**Prob(JB):**	0.00162
Kurtosis:	3.978	**Cond. No.**	1.22e+03

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.22e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [106]:

```
p2=m3.predict(testx)
print(p2)
```

```
126    999.297692
74    -118.312062
150    -19.682158
71     383.841733
49     179.376455
66     136.132293
25     729.901927
121    896.014700
132    470.750348
39      94.687007
106    352.142773
18     619.645746
88     154.991204
131    394.576241
21     652.652874
130    354.936686
77      41.312929
2      358.438577
113    702.467315
23     697.820015
100    265.185728
85     155.164203
60     777.434913
149    -17.447729
52     354.008448
29     898.130463
105    339.008106
98     250.162118
41     105.466630
101    283.523071
153     -2.988999
72    -345.930715
96     242.995693
30     871.993225
125    944.389097
110    661.879485
112    740.132508
120    859.367526
61     -83.839124
44     144.728878
63       7.167483
134    530.690154
91     157.285995
37      17.200785
50     222.499668
47     152.059025
135    518.678712
dtype: float64
```

In [107]:

```
#Lets Check for Assumption..
#1) mean of residual should be zero
print(m3.resid.mean())
```

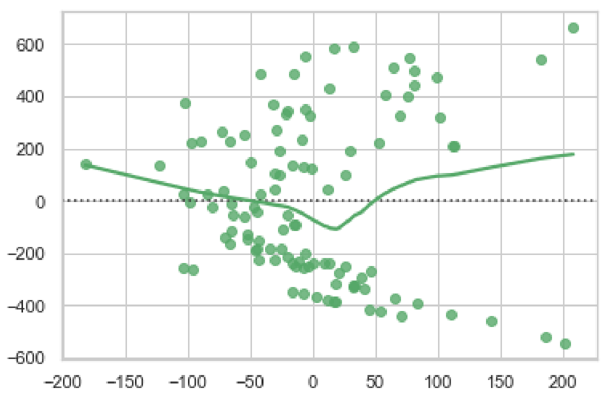
```
-1.6977408266179861e-12
```

In [108]:

```
#ii) Residual have constant variance(homoscidasticity) # Lowess-> locally wieghted scatterplotsmotting.
# Plot the graph
yhat = m3.predict(trainx)
sns.set(style='whitegrid')
sns.residplot(m3.resid,yhat, color='g' , lowess=True) ## BAsed On graph we found that the model is homoscedasticity.
```

Out[108]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b5afd9e348>
```



In [109]:

```
# Bruesch - pagan test for finding the hetro and homo.
import statsmodels.stats.api as sms
# H0-> Homoscedasticity
# H1-> Hetrocedasticity
# return value of breushch test
# lagrange _ Multiplier , pvalue , fscore , fp-value
pval = sms.het_breuschpagan(m3.resid , m3.model.exog)[1]
```

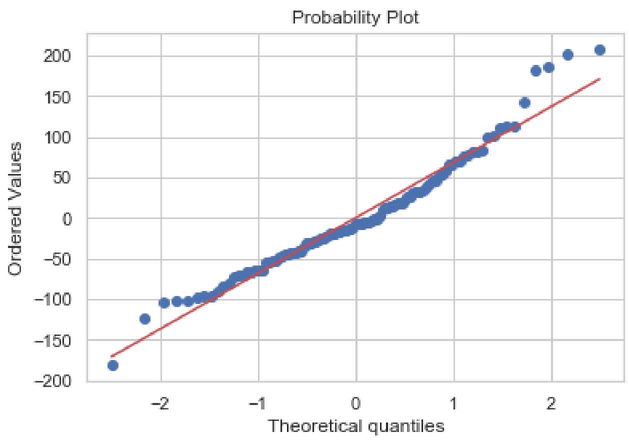
In [110]:

```
if pval<0.05:
 print('Reject H0 , Model is Hetroscidasticity')
else:
 print('FTR H0 , Model is homoscedasticity')
```

```
FTR H0 , Model is homoscedasticity
```

In [111]:

```
# 3) Check error is normally distributed or not.
stats.probplot(m3.resid , dist = 'norm' , plot=pylab)
pylab.show() # Not normally distributed
```



In [112]:

```
# MSE of model 3
mse2 = round(mean_squared_error(testy , p2),3)
mse2
```

Out[112]:

```
7821.936
```

RMSE

In [113]:

```
import math

math.sqrt(mse2)
```

Out[113]:

```
88.44170961712578
```

In [114]:

```
# Know campare the train and test error
print('Training MSE = {} , Testing MSE1 = {},  MSE2 ={}' . format(np.mean(cv_mse) , mse1 , mse2))
```

```
Training MSE = 7627.499399999999 , Testing MSE1 = 5775.443,  MSE2 =7821.936
```

In [115]:

```
#.5 Compare real and predicted weights and give a conclusion statement based on it?
df = pd.DataFrame({'Actual':testy ,"Predicted2":p2})
df
```

Out[115]:

	Actual	Predicted2
126	1000.0	999.297692
74	40.0	-118.312062
150	8.7	-19.682158
71	300.0	383.841733
49	161.0	179.376455
66	140.0	136.132293
25	725.0	729.901927
121	1015.0	896.014700
132	430.0	470.750348
39	120.0	94.687007
106	250.0	352.142773
18	610.0	619.645746
88	130.0	154.991204
131	300.0	394.576241
21	685.0	652.652874
130	300.0	354.936686
77	100.0	41.312929
2	340.0	358.438577
113	700.0	702.467315
23	680.0	697.820015
100	197.0	265.185728
85	130.0	155.164203
60	1000.0	777.434913
149	9.8	-17.447729
52	290.0	354.008448
29	1000.0	898.130463
105	250.0	339.008106
98	188.0	250.162118

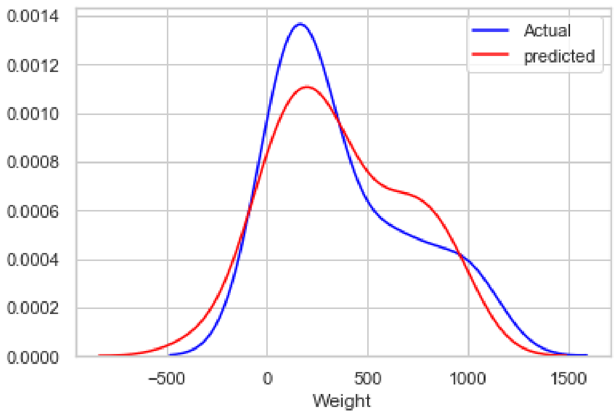
	Actual	Predicted2
41	110.0	105.466656
101	218.0	283.523071
153	9.8	-2.988999
72	5.9	-345.930715
96	225.0	242.995693
30	920.0	871.993225
125	1100.0	944.389097
110	556.0	661.879485
112	685.0	740.132508
120	900.0	859.367526
61	55.0	-83.839124
44	145.0	144.728878
63	90.0	7.167483
134	456.0	530.690154
91	130.0	157.285995
37	78.0	17.200785
50	200.0	222.499668
47	160.0	152.059025
135	510.0	518.678712

In [116]:

```
#plot the actual and predicted values
ax1=sns.distplot(testy,hist=False,color='blue',label ='Actual')
sns.distplot(p2,hist=False,color='red',label='predicted',ax=ax1)
```

Out[116]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b5acdc6b48>
```



we will now build the model on the basis of VIF

In [117]:

```
vif
```

Out[117]:

	inflation	features
0	282.422492	const
1	1921.023020	Length1
2	3488.043045	Length2

	inflation	features
3	1914.515067	Length3
4	71.831631	Height
5	28.748061	Width
6	7.571946	Species_Parkki
7	59.535474	Species_Perch
8	26.761004	Species_Pike
9	16.003276	Species_Roach
10	19.235876	Species_Smelt
11	6.016680	Species_Whitefish

In [118]:

```
new_fish2 = new_fish.drop(["Species_Whitefish","Species_Parkki"],axis=1)
new_fish2
```

Out[118]:

	Weight	Length1	Length2	Length3	Height	Width	Species_Perch	Species_Pike	Species_Roach	Species_Smelt
65	150.0	18.4	20.0	22.4	8.8928	3.2928	0	0	0	0
1	290.0	24.0	26.3	31.2	12.4800	4.3056	0	0	0	0
61	55.0	13.5	14.7	16.5	6.8475	2.3265	0	0	0	0
74	40.0	13.8	15.0	16.0	3.8240	2.4320	1	0	0	0
18	610.0	30.9	33.5	38.6	15.6330	5.1338	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
72	5.9	7.5	8.4	8.8	2.1120	1.4080	1	0	0	0
150	8.7	10.8	11.3	12.6	1.9782	1.2852	0	0	0	1
46	140.0	21.0	22.5	25.0	6.5500	3.3250	0	0	1	0
69	200.0	21.2	23.0	25.8	10.3458	3.6636	0	0	0	0
50	200.0	22.1	23.5	26.8	7.3968	4.1272	0	0	1	0

156 rows × 10 columns

model 4

In [119]:

```
trainx,testx,trainy,testy =
train_test_split(new_fish2.drop('Weight',axis=1),new_fish2['Weight'],test_size =0.3)
```

In [120]:

```
print("trainx={},trainy={},testx={},testy ={}".format(trainx.shape,trainy.shape,testx.shape,testy.s
hape))
```

```
trainx=(109, 9),trainy=(109,),testx=(47, 9),testy =(47,)
```

In [121]:

```
m4= sm.OLS(trainy,trainx).fit()
```

In [122]:

```
m4.summary()
```

Out[122]:

OLS Regression Results

Dep. Variable:	Weight	**R-squared (uncentered):**	0.946
Model:	OLS	**Adj. R-squared (uncentered):**	0.941
Method:	Least Squares	**F-statistic:**	195.8
Date:	Wed, 24 Mar 2021	**Prob (F-statistic):**	2.30e-59
Time:	23:02:49	**Log-Likelihood:**	-673.51
No. Observations:	109	**AIC:**	1365.
Df Residuals:	100	**BIC:**	1389.
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Length1	153.5219	52.295	2.936	0.004	49.771	257.273
Length2	-55.3684	63.051	-0.878	0.382	-180.459	69.722
Length3	-93.3276	29.073	-3.210	0.002	-151.008	-35.648
Height	27.3085	15.963	1.711	0.090	-4.361	58.978
Width	173.6550	38.759	4.480	0.000	96.759	250.551
Species_Perch	-293.5614	50.613	-5.800	0.000	-393.976	-193.147
Species_Pike	-25.6713	140.955	-0.182	0.856	-305.321	253.979
Species_Roach	-263.3364	50.274	-5.238	0.000	-363.079	-163.593
Species_Smelt	-125.4550	59.390	-2.112	0.037	-243.283	-7.627

Omnibus:	3.332	**Durbin-Watson:**	1.979
Prob(Omnibus):	0.189	**Jarque-Bera (JB):**	3.632
Skew:	-0.031	**Prob(JB):**	0.163
Kurtosis:	3.892	**Cond. No.**	704.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [123]:

```
# Add a constant term in train x and testx
# this will ensure that the model summary has the intercept term displayed.
trainx = sm.add_constant(trainx)
testx = sm.add_constant(testx)
```

In [124]:

```
m4=  sm.OLS(trainy,trainx).fit()
```

In [125]:

```
m4.summary()
```

Out[125]:

OLS Regression Results

Dep. Variable:	Weight	**R-squared:**	0.943
Model:	OLS	**Adj. R-squared:**	0.938
Method:	Least Squares	**F-statistic:**	181.4
Date:	Wed, 24 Mar 2021	**Prob (F-statistic):**	2.09e-57
Time:	23:02:49	**Log-Likelihood:**	-626.80
No. Observations:	109	**AIC:**	1274.

Df Residuals:	99	**BIC:**	1301.
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-575.8104	49.695	-11.587	0.000	-674.416	-477.204
Length1	16.0016	36.239	0.442	0.660	-55.905	87.908
Length2	40.1596	42.098	0.954	0.342	-43.373	123.692
Length3	-40.4673	19.575	-2.067	0.041	-79.308	-1.626
Height	38.4539	10.496	3.664	0.000	17.628	59.280
Width	65.5676	27.038	2.425	0.017	11.918	119.217
Species_Perch	18.7353	42.716	0.439	0.662	-66.023	103.493
Species_Pike	-5.6021	92.308	-0.061	0.952	-188.762	177.557
Species_Roach	21.8487	41.102	0.532	0.596	-59.707	103.404
Species_Smelt	290.5813	52.928	5.490	0.000	185.560	395.602

Omnibus:	25.116	**Durbin-Watson:**	1.981
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	40.741
Skew:	1.029	**Prob(JB):**	1.42e-09
Kurtosis:	5.176	**Cond. No.**	728.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [126]:

```
p3=m4.predict(testx)
print(p3)
```

```
90      169.789939
64      119.341325
58      592.696641
117     739.708048
71      393.581946
50      213.946029
151       8.658942
156      41.915175
100     284.667311
66      164.991402
12      497.346642
1       364.057023
107     403.494259
157     131.182514
118     883.552496
47      170.841493
129     253.716741
153       5.001130
83      124.207631
121     890.934761
5       487.699063
36        8.234879
24      692.267300
97      222.813134
43      125.811601
124     913.539092
31      857.778958
18      639.707130
98      264.310026
3       435.658840
135     478.984768
92      202.128499
59      706.025658
70      361.219901
146     -21.068889
```

```
42      91.874627
96      251.425898
44      161.946080
0       308.031077
80       69.837405
141     952.621672
102     438.001986
148      12.263160
111     812.050342
62      -23.989873
10      541.795402
48      202.384263
dtype: float64
```

In [127]:

```
#Lets Check for Assumption..
#1) mean of residual should be zero
print(m4.resid.mean())
```

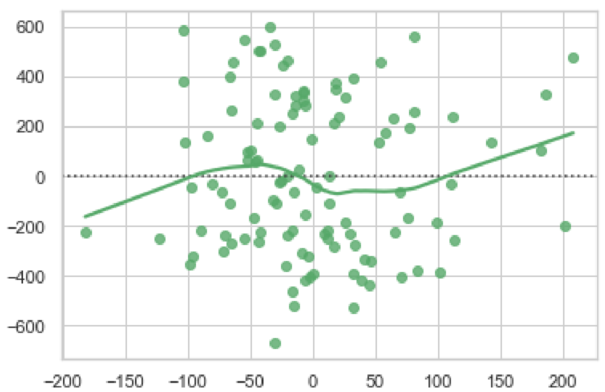
```
-1.190322050915544e-12
```

In [128]:

```
#ii) Residual have constant variance(homoscidasticity) # Lowess-> locally wieghted scatterplotsmotting.
# Plot the graph
yhat = m4.predict(trainx)
sns.set(style='whitegrid')
sns.residplot(m3.resid,yhat, color='g' , lowess=True) ## BAsed On graph we found that the model is homoscedasticity.
```

Out[128]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b5ae06dd48>
```



In [129]:

```
# Bruesch - pagan test for finding the hetro and homo.
import statsmodels.stats.api as sms
# H0-> Homoscedasticity
# H1-> Hetrocedasticity
# return value of breushch test
# lagrange _ Multiplier , pvalue , fscore , fp-value
pval = sms.het_breuschpagan(m4.resid , m4.model.exog)[1]
```

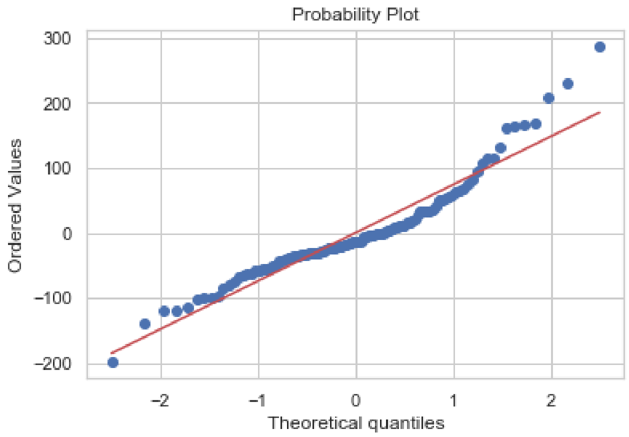
In [130]:

```
if pval<0.05:
    print('Reject H0 , Model is Hetroscidasticity')
else:
    print('FTR H0 , Model is homoscedasticity')
```

```
FTR H0 , Model is homoscedasticity
```

In [131]:

```
# 3) Check error is normally distributed or not.
stats.probplot(m4.resid , dist = 'norm' , plot=pylab)
pylab.show() # Not normally distributed
```



In [132]:

```
# MSE of model 1
mse3 = round(mean_squared_error(testy , p3),3)
mse3
```

Out[132]:

```
5788.822
```

RMSE

In [133]:

```
import math

math.sqrt(mse3)
```

Out[133]:

```
76.08430850050489
```

In [134]:

```
# Know campare the train and test error
print('Training MSE = {} , Testing MSE1 = {},  MSE2 ={}, MSE3 ={}' . format(np.mean(cv_mse) , mse1
, mse2, mse3))
```

```
Training MSE = 7627.499399999999 , Testing MSE1 = 5775.443,  MSE2 =7821.936, MSE3 =5788.822
```

In [135]:

```
#.5 Compare real and predicted weights and give a conclusion statement based on it?
df = pd.DataFrame({'Actual':testy ,"Predicted2":p3})
df
```

Out[135]:

	Actual	Predicted2
90	110.0	169.789939
64	120.0	119.341325
58	540.0	592.696641

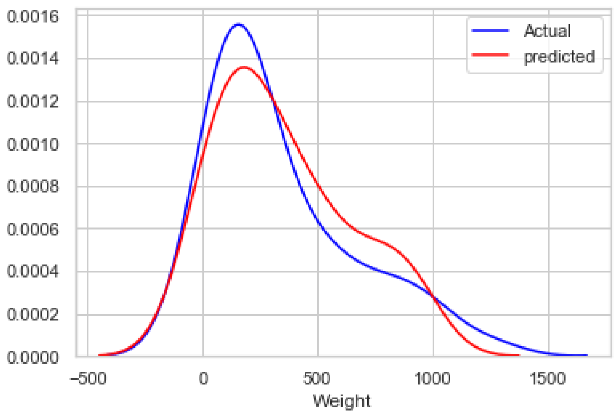
	Actual	Predicted2
117	650.0	739.708048
71	300.0	393.581946
50	200.0	213.946029
151	10.0	8.658942
156	12.2	41.915175
100	197.0	284.667311
66	140.0	164.991402
12	500.0	497.346642
1	290.0	364.057023
107	300.0	403.494259
157	19.7	131.182514
118	820.0	883.552496
47	160.0	170.841493
129	300.0	253.716741
153	9.8	5.001130
83	115.0	124.207631
121	1015.0	890.934761
5	450.0	487.699063
36	69.0	8.234879
24	700.0	692.267300
97	145.0	222.813134
43	150.0	125.811601
124	1000.0	913.539092
31	955.0	857.778958
18	610.0	639.707130
98	188.0	264.310026
3	363.0	435.658840
135	510.0	478.984768
92	150.0	202.128499
59	800.0	706.025658
70	273.0	361.219901
146	7.5	-21.068889
42	120.0	91.874627
96	225.0	251.425898
44	145.0	161.946080
0	242.0	308.031077
80	85.0	69.837405
141	1250.0	952.621672
102	300.0	438.001986
148	9.7	12.263160
111	840.0	812.050342
62	60.0	-23.989873
10	475.0	541.795402
48	169.0	202.384263

In [136]:

```
#plot the actual and predicted values
ax1=sns.distplot(testy,hist=False,color='blue',label ='Actual')
sns.distplot(p3,hist=False,color='red',label='predicted',ax=ax1)
```

Out[136]:

<matplotlib.axes._subplots.AxesSubplot at 0x2b5afbb3988>



so here we conclude that we have built 3 models

1st with all the data -----> #MSE1 = 5775.443

2nd with the removing multicolinearity -----> #MSE2 =7821.936

3rd on the bases of VIF -----> #MSE3 =5788.822