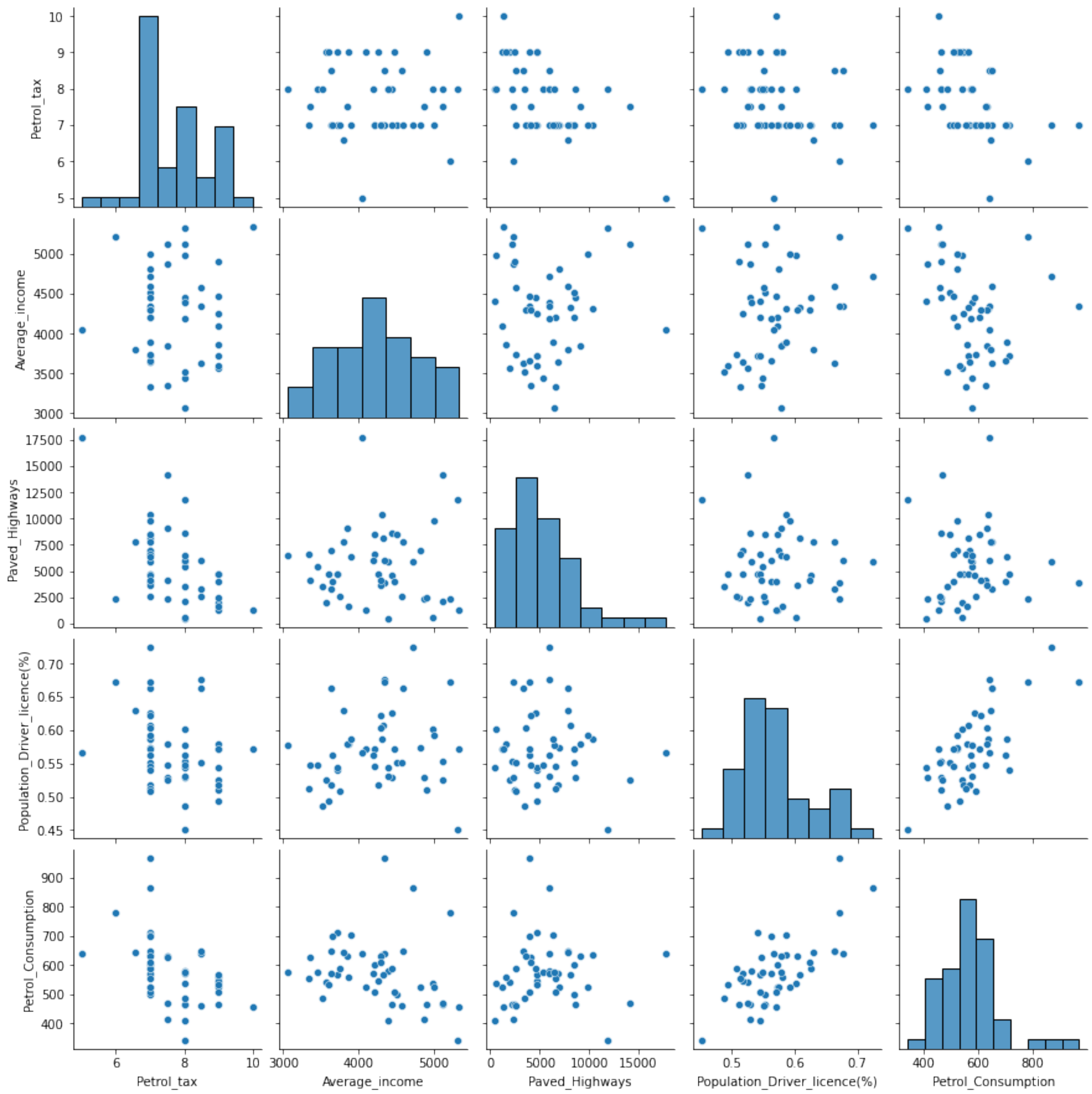
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#%matplotlib inline
from scipy.stats import norm

dataset=pd.read_csv("petrol_consumption.csv")

dataset.head()

   Petrol_tax  Average_income  ...  Population_Driver_licence(%)  
Petrol_Consumption
0         9.0            3571  ...                         0.525  
541
1         9.0            4092  ...                         0.572  
524
2         9.0            3865  ...                         0.580  
561
3         7.5            4870  ...                         0.529  
414
4         8.0            4399  ...                         0.544  
410

[5 rows x 5 columns]

dataset.describe()

       Petrol_tax  ...  Petrol_Consumption
count   48.000000  ...           48.000000
mean     7.668333  ...          576.770833
std      0.950770  ...          111.885816
min      5.000000  ...          344.000000
25%      7.000000  ...          509.500000
50%      7.500000  ...          568.500000
75%      8.125000  ...          632.750000
max     10.000000  ...          968.000000

[8 rows x 5 columns]

dataset.corr()

                              Petrol_tax  ...  Petrol_Consumption
Petrol_tax                      1.000000  ...           -0.451280
Average_income                  0.012665  ...           -0.244862
Paved_Highways                 -0.522130  ...            0.019042
Population_Driver_licence(%)   -0.288037  ...            0.698965
Petrol_Consumption             -0.451280  ...            1.000000

[5 rows x 5 columns]

dataset.shape
```

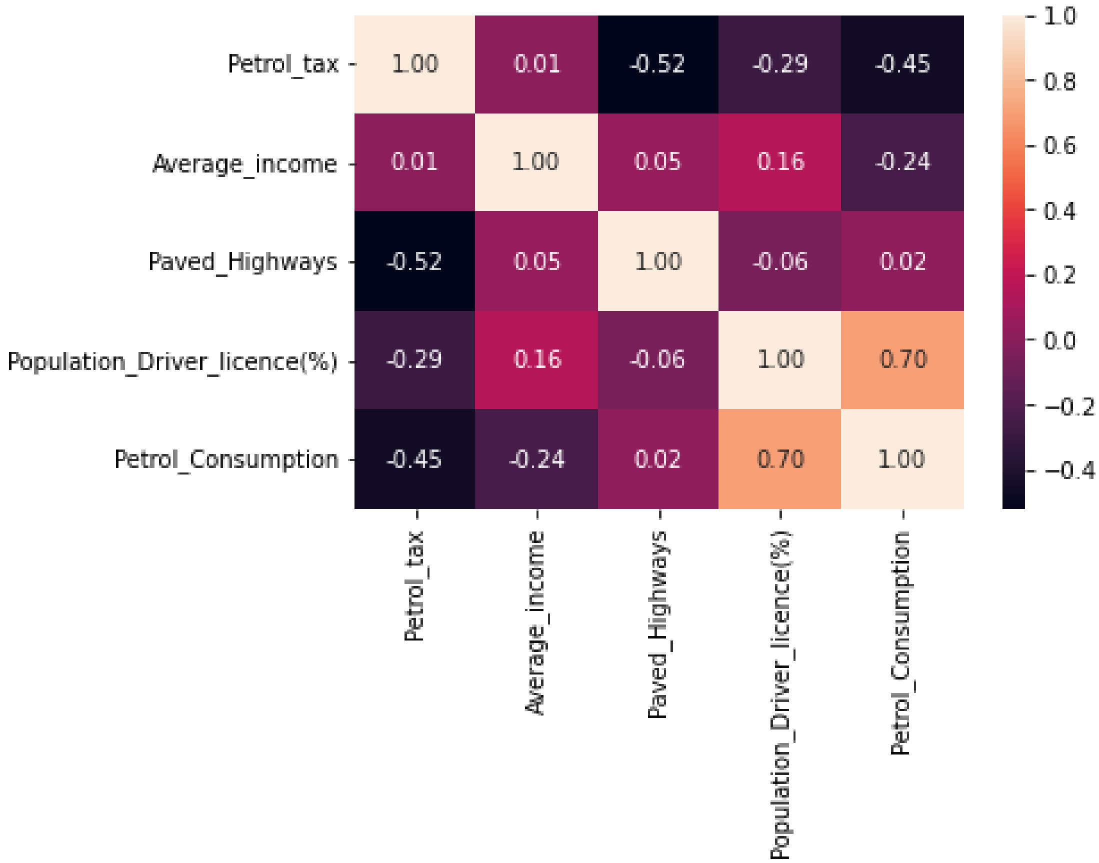
```
(48, 5)
```

```python
sns.pairplot(dataset)
```

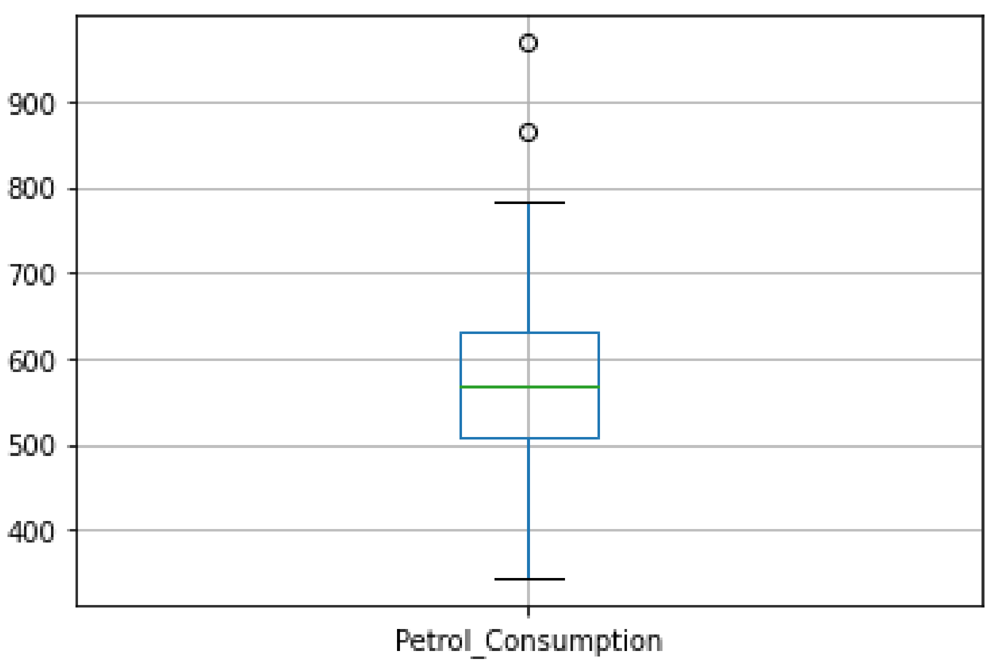
```
<seaborn.axisgrid.PairGrid at 0x7f4abcc7fc90>
```




```python
sns.heatmap(dataset.corr(),annot=True,fmt=".2f")
plt.show()
```




```
dataset.boxplot(column="Petrol_Consumption")
plt.show()
```



```
X=dataset[["Petrol_tax",   "Average_income",    "Paved_Highways",
"Population_Driver_licence(%)"]]
Y=dataset["Petrol_Consumption"]
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=13)
```

```
y_train
```

```
24    460
45    510
15    635
36    640
17    714
40    587
5     457
3     414
7     467
33    628
12    525
8     464
42    632
47    524
1     524
28    574
30    571
22    464
43    591
31    554
21    540
19    640
32    577
39    968
11    471
9     498
46    610
13    508
37    704
2     561
26    577
35    644
25    566
34    487
38    648
16    603
10    580
18    865
Name: Petrol_Consumption, dtype: int64
```

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression(fit_intercept=True)
regressor.fit(x_train,y_train)
```

```
LinearRegression()
```

```
coef_df = pd.DataFrame(regressor.coef_, X.columns,
columns=['Coefficient'])
coef_df
```

```
                              Coefficient
Petrol_tax                     -40.214615
Average_income                  -0.062788
Paved_Highways                  -0.003358
Population_Driver_licence(%)  1426.621220
```

```
regressor.intercept_
```

```
353.4972912944651
```

```
y_pred = regressor.predict(x_train)
y_pred
```

```
array([501.75318134, 511.88861177, 602.15179804, 646.19789721,
       593.05651563, 670.13661061, 426.06105487, 492.89684077,
       491.66997703, 607.63044299, 565.1525867 , 478.44064337,
       650.05140945, 571.02464073, 546.46620998, 551.94598344,
       559.24740516, 404.87135005, 552.80863328, 572.43466715,
       575.71134779, 684.74937036, 642.13830662, 744.7560876 ,
       431.37205803, 547.6213926 , 677.33050397, 563.2567191 ,
       641.86736818, 571.00372769, 578.94422998, 721.20103277,
       518.07488253, 493.29133418, 718.29319589, 595.36363237,
       492.24238485, 788.89596594])
```

```
regressor.coef_
```

```
array([-4.02146155e+01, -6.27881489e-02, -3.35821983e-03,
1.42662122e+03])
```

```
import sklearn.preprocessing as pp
scaler = pp.MinMaxScaler()
x_train_scaled = scaler.fit_transform(x_train)
```

```
x_train_scaled
```

```
array([[0.7       , 0.66301009, 0.11740396, 0.27004219],
       [0.8       , 0.62000878, 0.19441211, 0.35443038],
       [0.4       , 0.55068012, 0.56682189, 0.41772152],
       [0.        , 0.43089074, 1.        , 0.33333333],
       [0.4       , 0.28740676, 0.23998836, 0.22362869],
       [0.4       , 0.60816147, 0.23498254, 0.58649789],
       [1.        , 1.        , 0.04254948, 0.35443038],
       [0.5       , 0.79289162, 0.10180442, 0.17721519],
```

```
       [0.6       , 0.90522159, 0.08940629, 0.27848101],
       [0.5       , 0.12900395, 0.2048312 , 0.25316456],
       [0.4       , 0.76963581, 0.36833527, 0.36708861],
       [0.6       , 0.6072839 , 0.46420256, 0.17721519],
       [0.4       , 0.54278192, 0.17654249, 0.48945148],
       [0.4       , 0.85081176, 0.53504075, 0.44725738],
       [0.8       , 0.45151382, 0.03771828, 0.35864979],
       [0.6       , 0.49363756, 0.31274738, 0.32067511],
       [0.4       , 0.25318122, 0.36688009, 0.13080169],
       [0.8       , 0.80473892, 0.10750873, 0.10126582],
       [0.4       , 0.29925406, 0.1169383 , 0.08860759],
       [0.4       , 0.11847301, 0.34877765, 0.10970464],
       [0.6       , 0.84247477, 0.        , 0.48523207],
       [0.7       , 0.56077227, 0.31478463, 0.80168776],
       [0.6       , 0.        , 0.34470314, 0.38396624],
       [0.4       , 0.56252742, 0.19225844, 0.78059072],
       [0.5       , 0.90522159, 0.79068685, 0.16033755],
       [0.4       , 0.63580518, 0.46012806, 0.2742616 ],
       [0.4       , 0.54102677, 0.20261932, 0.57383966],
       [0.4       , 0.50197455, 0.34796275, 0.24472574],
       [0.4       , 0.36594998, 0.33661234, 0.41772152],
       [0.8       , 0.35190873, 0.0572759 , 0.39240506],
       [0.6       , 0.16893374, 0.27922002, 0.25738397],
       [0.316     , 0.32426503, 0.4209546 , 0.59915612],
       [0.8       , 0.28872312, 0.24121071, 0.24050633],
       [0.6       , 0.20403686, 0.16839348, 0.        ],
       [0.7       , 0.25098728, 0.15552969, 0.74261603],
       [0.4       , 0.50153576, 0.46018626, 0.35864979],
       [0.6       , 0.58271172, 0.31065192, 0.1814346 ],
       [0.4       , 0.72531812, 0.30925495, 1.        ]])
```

```
regressor_scaled = LinearRegression(fit_intercept=True)
regressor_scaled.fit(x_train_scaled,y_train)
```

```
LinearRegression()
```

```
coef_df_scaled=pd.DataFrame(regressor_scaled.coef_, X.columns,
columns=['Scaled_Coefficients'])
coef_df_scaled
```

	Scaled_Coefficients
Petrol_tax	-201.073077
Average_income	-143.094191
Paved_Highways	-57.694217
Population_Driver_licence(%)	338.109229

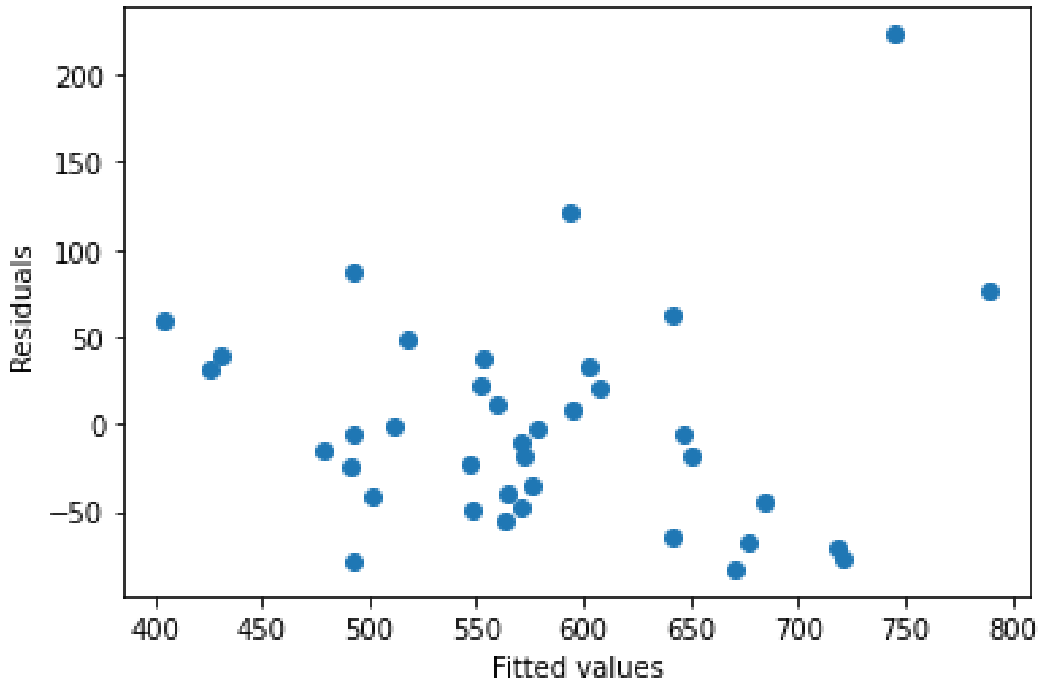
```
df = pd.DataFrame({'Actual':y_train, 'Predicted':y_pred})
df
```

	Actual	Predicted
24	460	501.753181

```
45     510  511.888612
15     635  602.151798
36     640  646.197897
17     714  593.056516
40     587  670.136611
5      457  426.061055
3      414  492.896841
7      467  491.669977
33     628  607.630443
12     525  565.152587
8      464  478.440643
42     632  650.051409
47     524  571.024641
1      524  546.466210
28     574  551.945983
30     571  559.247405
22     464  404.871350
43     591  552.808633
31     554  572.434667
21     540  575.711348
19     640  684.749370
32     577  642.138307
39     968  744.756088
11     471  431.372058
9      498  547.621393
46     610  677.330504
13     508  563.256719
37     704  641.867368
2      561  571.003728
26     577  578.944230
35     644  721.201033
25     566  518.074883
34     487  493.291334
38     648  718.293196
16     603  595.363632
10     580  492.242385
18     865  788.895966
```

```
#Validating OLS assumptions
plt.scatter(y_pred, (y_train-y_pred))
plt.xlabel('Fitted values')
plt.ylabel('Residuals')
```

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



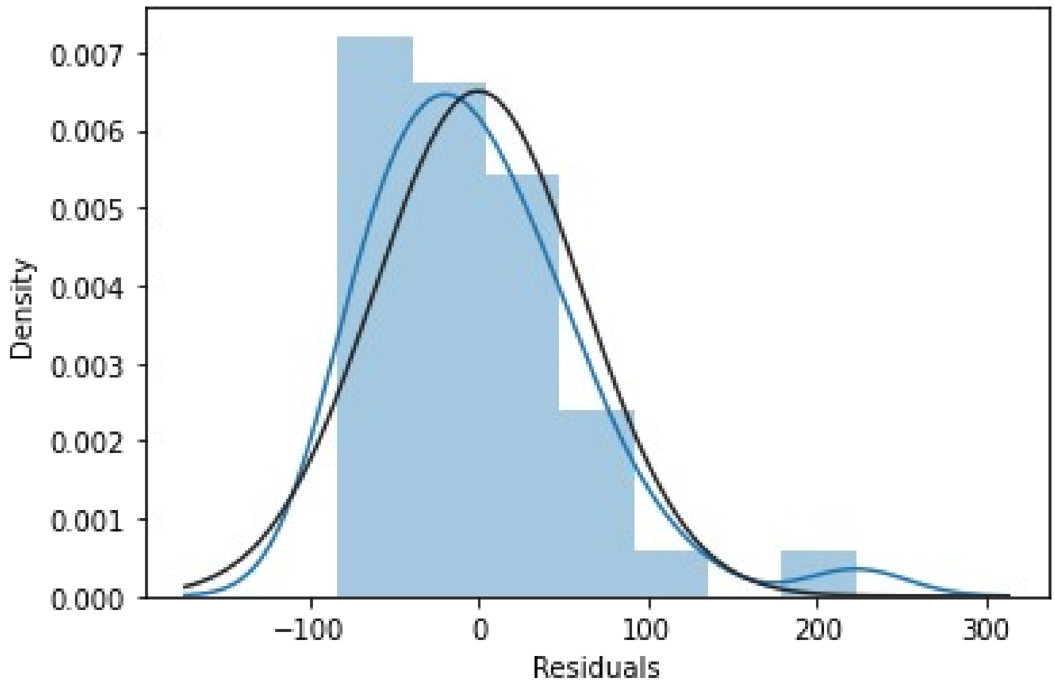

```
(y_train-y_pred).mean()
```

```
-5.3851659973397064e-14
```

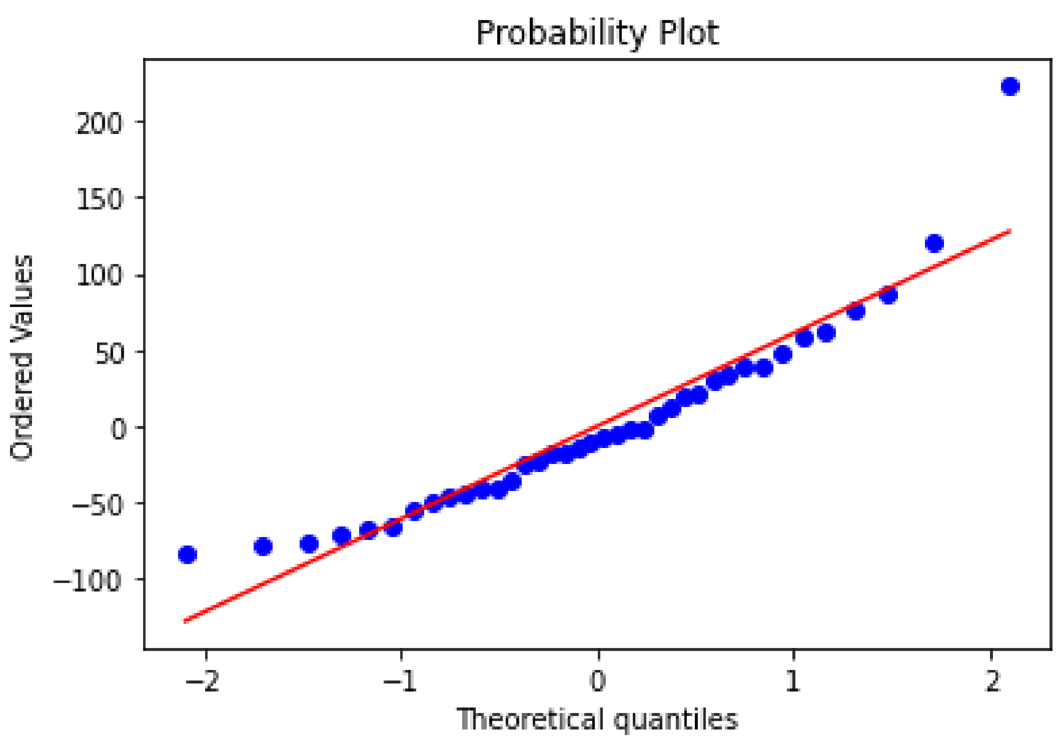
```
sns.distplot(y_train-y_pred, fit=norm)
plt.xlabel('Residuals')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed
in a future 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)
```

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




```
from scipy import stats
stats.probplot(y_train-y_pred, plot=plt)
plt.show()
```



```
import statsmodels.api as sm
x_endog = sm.add_constant(x_train)
x_endog1 = sm.add_constant(x_test)
```

```
const                           38
Petrol_tax                      38
Average_income                  38
Paved_Highways                  38
Population_Driver_licence(%)    38
dtype: int64
```

```
res = sm.OLS(y_train, x_endog)
res.fit()
```

```
<statsmodels.regression.linear_model.RegressionResultsWrapper at
0x7f4a9c5d8d10>
```

```
res.fit().summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results

======================================================================
========
Dep. Variable:     Petrol_Consumption   R-squared:
0.672
Model:                            OLS   Adj. R-squared:
0.633
Method:                 Least Squares   F-statistic:
16.94
Date:                Mon, 06 Dec 2021   Prob (F-statistic):
1.22e-07
Time:                        09:00:41   Log-Likelihood:
-210.36
No. Observations:                  38   AIC:
430.7
Df Residuals:                      33   BIC:
438.9
Df Model:                           4

Covariance Type:            nonrobust

======================================================================
==========================
                                   coef    std err          t      P>|
t|      [0.025      0.975]
----------------------------------------------------------------------
--------------------------
const                          353.4973    196.097      1.803
0.081     -45.465     752.460
```

```
Petrol_tax                    -40.2146     15.405     -2.611
0.013     -71.556      -8.873
Average_income                 -0.0628      0.020     -3.172
0.003      -0.103      -0.023
Paved_Highways                 -0.0034      0.004     -0.828
0.414      -0.012       0.005
Population_Driver_licence(%)  1426.6212    216.537      6.588
0.000     986.074    1867.169
======================================================================
========
Omnibus:                      17.059   Durbin-Watson:
2.041
Prob(Omnibus):                 0.000   Jarque-Bera (JB):
22.210
Skew:                          1.321   Prob(JB):
1.50e-05
Kurtosis:                      5.654   Cond. No.
1.91e+05
======================================================================
========

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 1.91e+05. This might indicate that
there are
strong multicollinearity or other numerical problems.
"""
```

```
from sklearn import metrics
print("Mean Absolute Error for Training
Data:",metrics.mean_absolute_error(y_train,y_pred))
print("Mean Squared Error for Training
Data:",metrics.mean_squared_error(y_train,y_pred))
print("Root Mean Squared Error for Training
Data:",np.sqrt(metrics.mean_squared_error(y_train,y_pred)))
```

```
Mean Absolute Error for Training Data: 46.35023352889107
Mean Squared Error for Training Data: 3766.3675675584573
Root Mean Squared Error for Training Data: 61.37073869164732
```

```
y_pred1 = regressor.predict(x_test)
```

```
print("Mean Absolute Error for Testing
Data:",metrics.mean_absolute_error(y_test,y_pred1))
print("Mean Squared Error for Testing
Data:",metrics.mean_squared_error(y_test,y_pred1))
print("Root Mean Squared Error for Testing
Data:",np.sqrt(metrics.mean_squared_error(y_test,y_pred1)))
```

```
Mean Absolute Error for Testing Data: 64.21137215974832
Mean Squared Error for Testing Data: 4960.283075034098
Root Mean Squared Error for Testing Data: 70.42927711565765
```

```
y_pred2=res.fit().predict(x_endog)

print("Mean Absolute Error for Training
Data:",metrics.mean_absolute_error(y_train,y_pred2))
print("Mean Squared Error for Training
Data:",metrics.mean_squared_error(y_train,y_pred2))
print("Root Mean Squared Error for Training
Data:",np.sqrt(metrics.mean_squared_error(y_train,y_pred2)))
```

```
Mean Absolute Error for Training Data: 46.350233528891046
Mean Squared Error for Training Data: 3766.3675675584573
Root Mean Squared Error for Training Data: 61.37073869164732
```

```
y_pred3=res.fit().predict(x_endog1)

print("Mean Absolute Error for Testing
Data:",metrics.mean_absolute_error(y_test,y_pred3))
print("Mean Squared Error for Testing
Data:",metrics.mean_squared_error(y_test,y_pred3))
print("Root Mean Squared Error for Testing
Data:",np.sqrt(metrics.mean_squared_error(y_test,y_pred3)))
```

```
Mean Absolute Error for Testing Data: 64.2113721597514
Mean Squared Error for Testing Data: 4960.283075034507
Root Mean Squared Error for Testing Data: 70.42927711566055
```

```
def mean_aboslute_percentage_error(y_true, y_pred):
  return np.mean(np.abs((y_true-y_pred)/y_true)*100)

print("Mean Absolute Percentage Error For Training
Data:",mean_aboslute_percentage_error(y_train, y_pred))
```

```
Mean Absolute Percentage Error For Training Data: 7.719487194763885
```

```
print("Mean Absolute Percentage Error For Testing
Data:",mean_aboslute_percentage_error(y_test, y_pred1))
```

```
Mean Absolute Percentage Error For Testing Data: 12.194036441833633
```

```
X_improved = dataset[["Petrol_tax",   "Average_income",
"Population_Driver_licence(%)"]]
Y_improved = dataset["Petrol_Consumption"]

x_train_improved, x_test_improved, y_train_improved, y_test_improved =
train_test_split(X_improved, Y_improved, test_size=0.2,
random_state=13)

regressor_improved = LinearRegression(fit_intercept=True)
regressor_improved.fit(x_train_improved, y_train_improved)
```

```python
coef_df_improved = pd.DataFrame(regressor_improved.coef_,
X_improved.columns, columns=['Improved Coefficients'])
```

```python
regressor_improved.intercept_
```

```
264.6773040332913
```

```python
coef_df_improved
```

	Improved Coefficients
Petrol_tax	-31.952461
Average_income	-0.064837
Population_Driver_licence(%)	1453.791163

```python
regressor_improved.coef_
```

```
array([-3.19524609e+01, -6.48369022e-02,  1.45379116e+03])
```

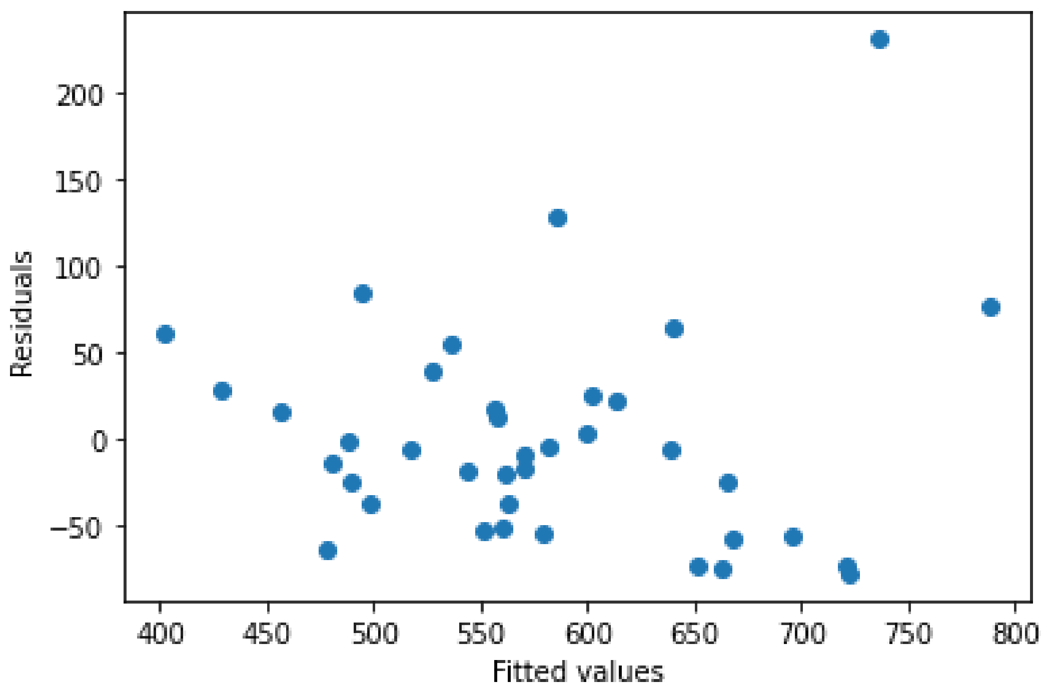
```python
y_pred_improved = regressor_improved.predict(x_train_improved)
```

```python
y_pred_improved
```

```
array([497.55632583, 517.00993503, 612.96595492, 665.49552787,
       584.99370278, 662.62396724, 428.90871674, 478.33365813,
       480.65016859, 602.60013215, 563.16684675, 489.7834373 ,
       638.84746893, 578.79405192, 543.36109665, 556.00509452,
       558.06757557, 402.48612942, 536.72178921, 570.70354874,
       561.15761259, 695.84101056, 650.75347698, 736.24139856,
       455.9202465 , 550.95869635, 668.18263979, 560.55741339,
       640.26229076, 569.70940276, 582.17753474, 722.35485007,
       526.70943482, 488.30932163, 721.26278726, 599.87461169,
       494.86809499, 787.78404829])
```

```python
plt.scatter(y_pred_improved, (y_train_improved-y_pred_improved))
plt.xlabel("Fitted values")
plt.ylabel("Residuals")
```

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



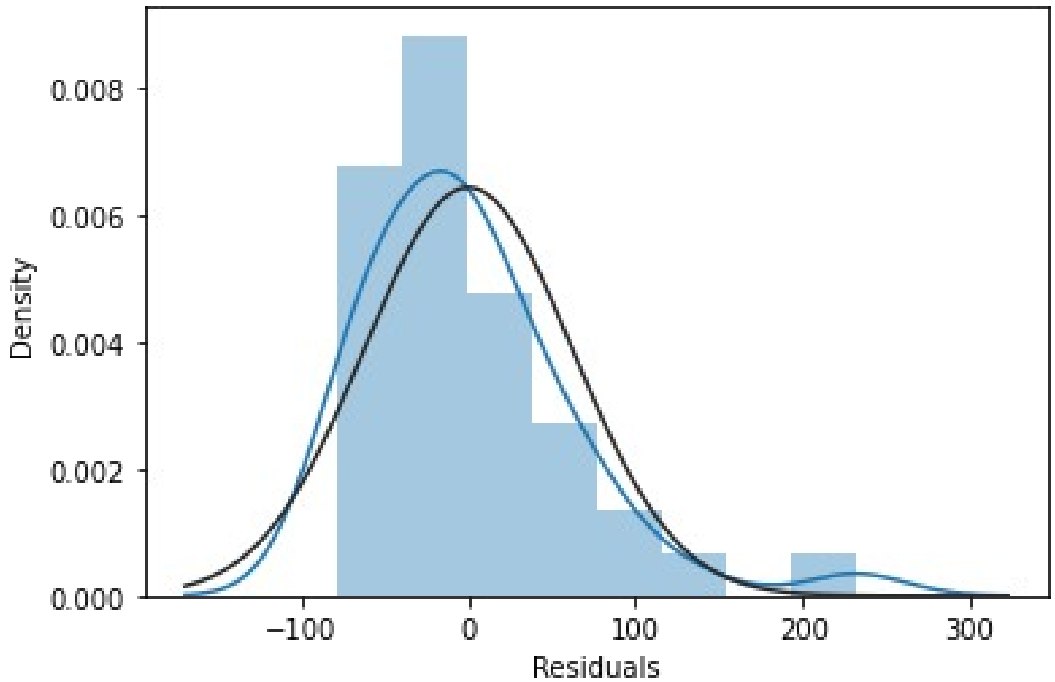

```
(y_train_improved-y_pred_improved).mean()
```

```
-1.5108382381425288e-13
```

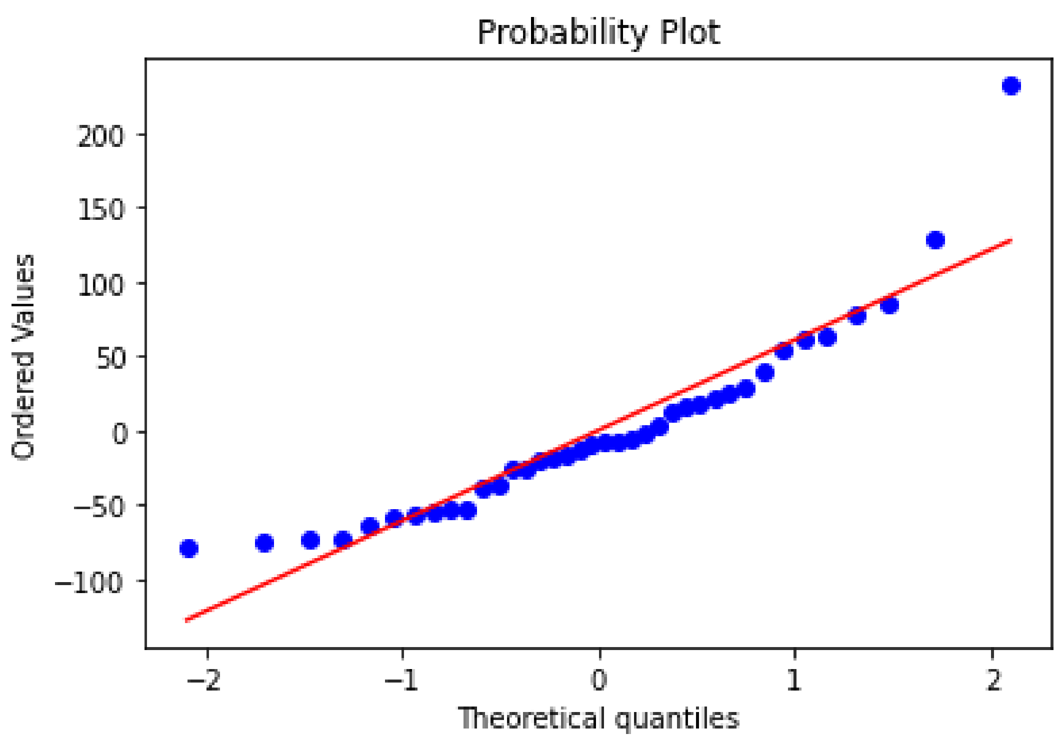
```
sns.distplot(y_train_improved-y_pred_improved, fit=norm)
plt.xlabel("Residuals")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed
in a future 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)
```

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




```
stats.probplot(y_train_improved-y_pred_improved, plot=plt)
plt.show()
```




```
import statsmodels.api as sm
x_endog_improved = sm.add_constant(x_train_improved)
x_endog_improved1 = sm.add_constant(x_test_improved)
```

```
res_improved = sm.OLS(y_train_improved, x_endog_improved)
res_improved.fit()
```

```
<statsmodels.regression.linear_model.RegressionResultsWrapper at
0x7f4a96877950>
```

```
res_improved.fit().summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results

======================================================================
========
Dep. Variable:     Petrol_Consumption   R-squared:
0.666
Model:                            OLS   Adj. R-squared:
0.636
Method:                 Least Squares   F-statistic:
22.56
Date:                Mon, 06 Dec 2021   Prob (F-statistic):
3.21e-08
Time:                        10:59:06   Log-Likelihood:
-210.75
No. Observations:                  38   AIC:
429.5
Df Residuals:                      34   BIC:
436.1
Df Model:                           3

Covariance Type:            nonrobust

======================================================================
==========================
                                   coef    std err          t      P>|
t|      [0.025      0.975]
----------------------------------------------------------------------
--------------------------
const                          264.6773    163.403      1.620
0.115     -67.397     596.752
Petrol_tax                     -31.9525     11.683     -2.735
0.010     -55.695      -8.210
Average_income                  -0.0648      0.020     -3.317
0.002      -0.105      -0.025
Population_Driver_licence(%)  1453.7912    213.045      6.824
0.000    1020.832    1886.751
======================================================================
========
Omnibus:                       20.317   Durbin-Watson:
2.150
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
30.370
Skew:                           1.487   Prob(JB):
2.54e-07
Kurtosis:                       6.214   Cond. No.
1.01e+05
======================================================================
========

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 1.01e+05. This might indicate that
there are
strong multicollinearity or other numerical problems.
"""
```

```
print("Improved Mean Absolute Error for Training
Data:",metrics.mean_absolute_error(y_train_improved,y_pred_improved))
print("Improved Mean Squared Error for Training
Data:",metrics.mean_squared_error(y_train_improved,y_pred_improved))
print("Improved Root Mean Squared Error for Training
Data:",np.sqrt(metrics.mean_squared_error(y_train_improved,y_pred_impr
oved)))
```

```
Improved Mean Absolute Error for Training Data: 45.61004100464847
Improved Mean Squared Error for Training Data: 3844.631677401122
Improved Root Mean Squared Error for Training Data: 62.0050939633279
```

```
y_pred_improved2 = regressor_improved.predict(x_test_improved)
print("Improved Mean Absolute Error for Testing
Data:",metrics.mean_absolute_error(y_test_improved,y_pred_improved2))
print("Improved Mean Squared Error for Testing
Data:",metrics.mean_squared_error(y_test_improved,y_pred_improved2))
print("Improved Root Mean Squared Error for Testing
Data:",np.sqrt(metrics.mean_squared_error(y_test_improved,y_pred_impro
ved2)))
```

```
Improved Mean Absolute Error for Testing Data: 62.552004344693614
Improved Mean Squared Error for Testing Data: 4734.841302969787
Improved Root Mean Squared Error for Testing Data: 68.81018313425555
```

```
print("Improved Mean Absolute Percentage Error For Training
Data:",mean_aboslute_percentage_error(y_train_improved,
y_pred_improved))
```

```
Improved Mean Absolute Percentage Error For Training Data:
7.497718961673175
```

```
print("Improved Mean Absolute Percentage Error For Testing
Data:",mean_aboslute_percentage_error(y_test_improved,
y_pred_improved2))
```

```
Improved Mean Absolute Percentage Error For Testing Data:
11.437027978173756
```

```
X_improved2 = dataset[[]]
```