10215501435- -

October 7, 2023

0.1

0.1.1 10215501435

0.2

(rpm)

0.3

rpm	R^+
Road_Octane_Number	$R+$
Compression	R^+
Brake_Horsepower	R^+

0.4

$\alpha = 0.05$

1.
2.
3.
4. 3000 /min 90 100

[74]:
```
import os #

import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from jupyterquiz import display_quiz

import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm
from scipy.stats import f
from scipy.stats import t
```

```python
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn import preprocessing

import warnings
warnings.filterwarnings('ignore')

alpha = 0.05
```

[75]:
```python
os.chdir("/Users/86138/    /Data")
```

[76]:
```python
print('Data 3 is shown as follows: \n', pd.read_csv("Project_3.csv"))
```

```
Data 3 is shown as follows:
       rpm  Road Octane Number  Compression  Brake Horsepower
0    2000                  90          100               225
1    1800                  94           95               212
2    2400                  88          110               229
3    1900                  91           96               222
4    1600                  86          100               219
5    2500                  96          110               278
6    3000                  94           98               246
7    3200                  90          100               237
8    2800                  88          105               233
9    3400                  86           97               224
10   1800                  90          100               223
11   2500                  89          104               230
```

[77]:
```python
Data = pd.read_csv("Project_3.csv")
print(Data.head())
```

```
    rpm  Road Octane Number  Compression  Brake Horsepower
0  2000                  90          100               225
1  1800                  94           95               212
2  2400                  88          110               229
3  1900                  91           96               222
4  1600                  86          100               219
```

[78]:
```python
n = Data.shape[0]
p = Data.shape[1] - 1
print("The number of instances is ", n)
print("The number of features is ", p)
```

```
The number of instances is  12
The number of features is  3
```

1 Task 1:

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon_i, i = 1, 2, \cdots, n$$

$\beta_0, \beta_1, \beta_2, \beta_3$ ϵ_i $E(\epsilon_i) = 0$ $Var(\epsilon_i) = \sigma^2$ n

$$\mathbf{y} = \mathbf{X} \quad +$$

-

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix},$$

- /

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} \end{pmatrix},$$

-

$$= \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix},$$

-

$$= \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}.$$

$$\hat{} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}.$$

Solution:

X_1 X_2 X_3 Y $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 +$ ϵ $\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3$

1.1

[79]:
```
## Method 1: Matrix Calculus
Data1 = sm.add_constant(Data)
Data1_value = Data1.values
X = Data1_value[:,0:(p+1)]
y = Data1_value[:,-1]
beta_hat_1 = np.linalg.inv(X.T @ X) @ (X.T @ y)
# A @ B <=> np.dot(A,B) matrix multiply
print("The estimates of the parameters are \n",
      np.around(beta_hat_1,4))
```

```
The estimates of the parameters are
 [-2.660312e+02  1.070000e-02  3.134800e+00  1.867400e+00]
```

[80]:
```python
## Method 2:  statsmodels  package
Data2 = pd.read_csv("Project_3.csv")

#
Data2 = Data2.rename(columns={'Road Octane Number': 'Road_Octane_Number',␣
 ↪'Brake Horsepower': 'Brake_Horsepower'})

#
print(Data2.head())

model1 = ols("Brake_Horsepower ~ rpm + Road_Octane_Number + Compression",␣
 ↪Data2).fit()
beta_hat_2 = model1.params
#print("The estimates of the parameters are \n",
#       round(model.param(),4))
print("The estimates of the parameters are \n",
      round(beta_hat_2,4))
```

```
    rpm  Road_Octane_Number  Compression  Brake_Horsepower
0  2000                  90          100               225
1  1800                  94           95               212
2  2400                  88          110               229
3  1900                  91           96               222
4  1600                  86          100               219
The estimates of the parameters are
 Intercept             -266.0312
rpm                      0.0107
Road_Octane_Number       3.1348
Compression              1.8674
dtype: float64
```

[81]:
```python
## Method 3:  scikit-learn package
model2 = linear_model.LinearRegression()
X_without_intercept = X[:,1:4]
model2.fit(X_without_intercept, y)
beta_hat_3 = np.append(np.array(model2.intercept_),model2.coef_)
print("The estimates of the parameters are \n",
      np.around(beta_hat_3,4))
```

```
The estimates of the parameters are
 [-2.660312e+02  1.070000e-02  3.134800e+00  1.867400e+00]
```

1.2

[82]:
```
import scipy.stats as stats
import math

x = pd.read_csv('Project_3.csv')
x.insert(0, 'intercept', np.ones(len(x)))
data = x.values * 1
df = pd.DataFrame(data, columns = ['intercept', 'P1', 'P2', 'P3', 'F'])
print(df)

# Do the multiple linear regression
model = ols('F ~ P1 + P2 + P3', df).fit()
beta = model.params
print('   : \n', round(beta, 4))
X = data[:, 0 : p + 1]
Y = data[:, -1]
Y_hat = model.fittedvalues
model.summary()
```

```
    intercept      P1    P2     P3      F
0         1.0  2000.0  90.0  100.0  225.0
1         1.0  1800.0  94.0   95.0  212.0
2         1.0  2400.0  88.0  110.0  229.0
3         1.0  1900.0  91.0   96.0  222.0
4         1.0  1600.0  86.0  100.0  219.0
5         1.0  2500.0  96.0  110.0  278.0
6         1.0  3000.0  94.0   98.0  246.0
7         1.0  3200.0  90.0  100.0  237.0
8         1.0  2800.0  88.0  105.0  233.0
9         1.0  3400.0  86.0   97.0  224.0
10        1.0  1800.0  90.0  100.0  223.0
11        1.0  2500.0  89.0  104.0  230.0
   :
 Intercept   -266.0312
P1             0.0107
P2             3.1348
P3             1.8674
dtype: float64
```

[82]:

Dep. Variable:	F	**R-squared:**	0.807
Model:	OLS	**Adj. R-squared:**	0.734
Method:	Least Squares	**F-statistic:**	11.12
Date:	Sat, 07 Oct 2023	**Prob (F-statistic):**	0.00317
Time:	15:22:12	**Log-Likelihood:**	-40.708
No. Observations:	12	**AIC:**	89.42
Df Residuals:	8	**BIC:**	91.36
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-266.0312	92.674	-2.871	0.021	-479.737	-52.325
P1	0.0107	0.004	2.390	0.044	0.000	0.021
P2	3.1348	0.844	3.712	0.006	1.188	5.082
P3	1.8674	0.535	3.494	0.008	0.635	3.100

Omnibus:	0.392	**Durbin-Watson:**	1.043
Prob(Omnibus):	0.822	**Jarque-Bera (JB):**	0.230
Skew:	-0.282	**Prob(JB):**	0.891
Kurtosis:	2.625	**Cond. No.**	9.03e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.03e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

1.3

[83]:
```
print('Y_hat = %.2f + (%.2f * X1) + (%.2f * X2) + (%.2f * X3)' % (beta[0],
↪beta[1], beta[2], beta[3]))
```

```
Y_hat = -266.03 + (0.01 * X1) + (3.13 * X2) + (1.87 * X3)
```

2 Task 2:

2.1

2.1.1

[84]:
```
model1.summary()
```

[84]:

Dep. Variable:	Brake_Horsepower	**R-squared:**	0.807
Model:	OLS	**Adj. R-squared:**	0.734
Method:	Least Squares	**F-statistic:**	11.12
Date:	Sat, 07 Oct 2023	**Prob (F-statistic):**	0.00317
Time:	15:22:14	**Log-Likelihood:**	-40.708
No. Observations:	12	**AIC:**	89.42
Df Residuals:	8	**BIC:**	91.36
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
Intercept	-266.0312	92.674	-2.871	0.021	-479.737	-52.325
rpm	0.0107	0.004	2.390	0.044	0.000	0.021
Road_Octane_Number	3.1348	0.844	3.712	0.006	1.188	5.082
Compression	1.8674	0.535	3.494	0.008	0.635	3.100

Omnibus:	0.392	**Durbin-Watson:**	1.043
Prob(Omnibus):	0.822	**Jarque-Bera (JB):**	0.230
Skew:	-0.282	**Prob(JB):**	0.891
Kurtosis:	2.625	**Cond. No.**	9.03e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.03e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

[85]:
```
##
Data1 = sm.add_constant(Data)
Data1_value = Data1.values
X = Data1_value[:,0:(p+1)]
y = Data1_value[:,-1]
X_center = preprocessing.scale(X_without_intercept, with_mean = True,␣
 ↪with_std=False)
y_center = preprocessing.scale(y, with_mean = True, with_std=False)
# with_mean = True (default), with_std = True (default)

# print(X_center)

print("The sample means of centered features are ", np.around(np.
 ↪mean(X_center,axis=0),4))
print("The sample mean of centered response is ", np.around(np.
 ↪mean(y_center,axis=0),4))
```

```
The sample means of centered features are  [-0. -0.  0.]
The sample mean of centered response is  0.0
```

[86]:
```
model3 = linear_model.LinearRegression()
model3.fit(X_center, y_center)
beta_hat_4 = np.append(np.array(model3.intercept_),model3.coef_)
```

```
print("The estimates of the parameters are \n",
          np.around(beta_hat_4,4))
```

```
The estimates of the parameters are
 [0.     0.0107 3.1348 1.8674]
```

2.1.2

[87]:
```
X = data[:, 0 : p + 1]
Y = data[:, -1]
#
X_mean = []
for k in range(p + 1):
    X_mean.append(np.mean(data[:, k]))  #    x
Y_mean = np.mean(data[:, -1])  #    y

#
X_cent = X - X_mean
Y_cent = Y - Y_mean

# Do the multiple linear regression
df = pd.DataFrame(X_cent, columns = ['intercept_cent', 'P1_cent', 'P2_cent',␣
 ↪'P3_cent'])
df['F_cent'] = Y_cent
model_cent = ols('F_cent ~ P1_cent + P2_cent + P3_cent', df).fit()
beta_cent = model_cent.params
print('   : \n', round(beta_cent, 4))
Y_hat_cent = model_cent.fittedvalues
model_cent.summary()
```

```
   :
 Intercept    0.0000
P1_cent      0.0107
P2_cent      3.1348
P3_cent      1.8674
dtype: float64
```

[87]:

Dep. Variable:	F_cent	**R-squared:**	0.807
Model:	OLS	**Adj. R-squared:**	0.734
Method:	Least Squares	**F-statistic:**	11.12
Date:	Sat, 07 Oct 2023	**Prob (F-statistic):**	0.00317
Time:	15:22:16	**Log-Likelihood:**	-40.708
No. Observations:	12	**AIC:**	89.42
Df Residuals:	8	**BIC:**	91.36
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P> |t|	[0.025	0.975]
Intercept	1.11e-16	2.544	4.36e-17	1.000	-5.866	5.866
P1_cent	0.0107	0.004	2.390	0.044	0.000	0.021
P2_cent	3.1348	0.844	3.712	0.006	1.188	5.082
P3_cent	1.8674	0.535	3.494	0.008	0.635	3.100

Omnibus:	0.392	**Durbin-Watson:**	1.043
Prob(Omnibus):	0.822	**Jarque-Bera (JB):**	0.230
Skew:	-0.282	**Prob(JB):**	0.891
Kurtosis:	2.625	**Cond. No.**	574.

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

2.1.3

[88]:
```
print('Y_hat_cent = %.2f + (%.2f * X1_cent) + (%.2f * X2_cent) + (%.2f *␣
↪X3_cent)' % (beta_cent[0], beta_cent[1], beta_cent[2], beta_cent[3]))
```

```
Y_hat_cent = 0.00 + (0.01 * X1_cent) + (3.13 * X2_cent) + (1.87 * X3_cent)
```

2.1.4 0

0 X Y 0

2.2

[89]:
```
X = data[:, 0 : p + 1]
Y = data[:, -1]
#
X_mean = []
for k in range(p + 1):
    X_mean.append(np.mean(data[:, k]))  #    x
Y_mean = np.mean(data[:, -1])  #    y

#
X_std = (X - X_mean) / np.std(X, axis=0)
Y_std = (Y - Y_mean) / np.std(Y)

# Do the multiple linear regression
df = pd.DataFrame(X_std, columns=['intercept_std', 'P1_std', 'P2_std',␣
↪'P3_std'])
df['F_std'] = Y_std
model_std = ols('F_std ~ P1_std + P2_std + P3_std', df).fit()
beta_std = model_std.params
print('   : \n', round(beta_std, 4))
```

```
Y_hat_std = model_std.fittedvalues
model_std.summary()
```

```
  :
 Intercept    0.0000
P1_std       0.3757
P2_std       0.5793
P3_std       0.5477
dtype: float64
```

[89]:

Dep. Variable:	F_std	**R-squared:**	0.807
Model:	OLS	**Adj. R-squared:**	0.734
Method:	Least Squares	**F-statistic:**	11.12
Date:	Sat, 07 Oct 2023	**Prob (F-statistic):**	0.00317
Time:	15:22:18	**Log-Likelihood:**	-7.1718
No. Observations:	12	**AIC:**	22.34
Df Residuals:	8	**BIC:**	24.28
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	1.735e-17	0.156	1.12e-16	1.000	-0.359	0.359
P1_std	0.3757	0.157	2.390	0.044	0.013	0.738
P2_std	0.5793	0.156	3.712	0.006	0.219	0.939
P3_std	0.5477	0.157	3.494	0.008	0.186	0.909

Omnibus:	0.392	**Durbin-Watson:**	1.043
Prob(Omnibus):	0.822	**Jarque-Bera (JB):**	0.230
Skew:	-0.282	**Prob(JB):**	0.891
Kurtosis:	2.625	**Cond. No.**	1.16

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

2.2.1

[90]:

```
print('Y_hat_std = %.2f + (%.2f * X1_std) + (%.2f * X2_std) + (%.2f * X3_std)'␣
 ↪% (beta_std[0], beta_std[1], beta_std[2], beta_std[3]))
```

```
Y_hat_std = 0.00 + (0.38 * X1_std) + (0.58 * X2_std) + (0.55 * X3_std)
```

2.2.2

Y

3 Task 3:

3.0.1 1

[91]:
```
model1.summary()
```

[91]:

Dep. Variable:	Brake_Horsepower	**R-squared:**	0.807
Model:	OLS	**Adj. R-squared:**	0.734
Method:	Least Squares	**F-statistic:**	11.12
Date:	Sat, 07 Oct 2023	**Prob (F-statistic):**	0.00317
Time:	15:22:21	**Log-Likelihood:**	-40.708
No. Observations:	12	**AIC:**	89.42
Df Residuals:	8	**BIC:**	91.36
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-266.0312	92.674	-2.871	0.021	-479.737	-52.325
rpm	0.0107	0.004	2.390	0.044	0.000	0.021
Road_Octane_Number	3.1348	0.844	3.712	0.006	1.188	5.082
Compression	1.8674	0.535	3.494	0.008	0.635	3.100

Omnibus:	0.392	**Durbin-Watson:**	1.043
Prob(Omnibus):	0.822	**Jarque-Bera (JB):**	0.230
Skew:	-0.282	**Prob(JB):**	0.891
Kurtosis:	2.625	**Cond. No.**	9.03e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.03e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

- F 11.12 p 0.00317 - t $2.390 3.712 $3.494 p$ $0.05

3.0.2

[92]:
```
#
SSE = sum((Y - Y_hat) ** 2)
SST = sum((Y - Y_mean) ** 2)
SSR = sum((Y_hat - Y_mean) ** 2)

sigma2 = SSE / (n - p - 1)  #
sigma = np.sqrt(sigma2)  #

c = np.dot(X.T, X)
C = np.linalg.inv(c)  #
# print(C)
```

——**F** :

$H_0 : \beta_1 = \beta_2 = \beta_3 = 0$ vs $H_1 : \exists \beta_i \neq 0, i = 1, 2, 3$

[93]:
```
#  F0
F0 = (SSR / p) / (SSE / (n - p - 1))
# F0 = model.fvalue
print('F0: %.2f' % F0)
F = round(f.ppf(0.95, dfn = p, dfd = n - p - 1), 2)

#  1
pVal1 = f.sf(F0, p, n - p - 1)
# pVal1 = model.f_pvalue
print('pVal1: %.5f' % pVal1)
if pVal1 < alpha:
    print('\nSince p-value < 0.05, reject H0.')
else:
    print('\nAccept H0.')

#  2
if F0 > F:
    print('Since F0 > F(0.95, 3, 8) = %.2f, reject H0.' % F)
else:
    print('Accept H0.')
```

```
F0: 11.12
pVal1: 0.00317

Since p-value < 0.05, reject H0.
Since F0 > F(0.95, 3, 8) = 4.07, reject H0.
```

F

——t :

$H_{0j} : \beta_j = 0$ vs $H_{1j} : \beta_j \neq 0, j = 1, 2, 3$

[94]:
```
# t
t0 = []
for i in range(p + 1):
    t0.append(beta[i] / (np.sqrt(C[i][i] * sigma2)))  #  t
# t0 = model.tvalues
print('t0  ', np.round(t0, 4))
tVal = t.ppf(1 - alpha / 2, n - p - 1)
print('t   %.4f' % tVal)
pVal2 = []
for i in range(p + 1):
    P = t.sf(abs(t0[i]), n - p - 1)
    pVal2.append(P)  #    p
# pVal2 = model.pvalues / 2
print('P  ', np.round(pVal2, 4))

print('\n')
```

```
# 1
for i in range(p):
    if pVal2[i + 1] < alpha:
        print ('Since p%d-value < 0.05, reject H0%d.' % (i + 1, i + 1))
    else:
        print('Accept H0%d.' % (i + 1))
print('\n')

# 2
for i in range(p):
    if abs(t0[i + 1]) > tVal:
        print('Since t0%d > t(0.975, 8) = %.4f, reject H0%d' % (i + 1, tVal, i + 1))
    else:
        print('Accept H0%d.' % (i + 1))
```

```
t0   [-2.8706  2.3896  3.7123  3.4936]
t    2.3060
P   [0.0104 0.0219 0.003  0.0041]


Since p1-value < 0.05, reject H01.
Since p2-value < 0.05, reject H02.
Since p3-value < 0.05, reject H03.


Since t01 > t(0.975, 8) = 2.3060, reject H01
Since t02 > t(0.975, 8) = 2.3060, reject H02
Since t03 > t(0.975, 8) = 2.3060, reject H03
```

t

:

$$R^2 = \frac{SS_R}{SS_T} = 1 - \frac{SS_E}{SS_T}$$

R^2 [0,1]

1. R^2 1,
2. R^2 0,

$$R_a^2 = 1 - \frac{n-1}{n-m-1}(1-R^2)$$

R^2 $adjusted R^2$

```
[95]: #
R2 = SSR / SST
print('   %.4f' % R2)

#
R2c = 1 - (SSE/(n-p-1)) / (SST/(n-1))
print('   Ra %.4f' % R2c)
```

```
 0.8065
  Ra 0.7340
```

1 X_1,X_2,X_3 Y

3.0.3 2

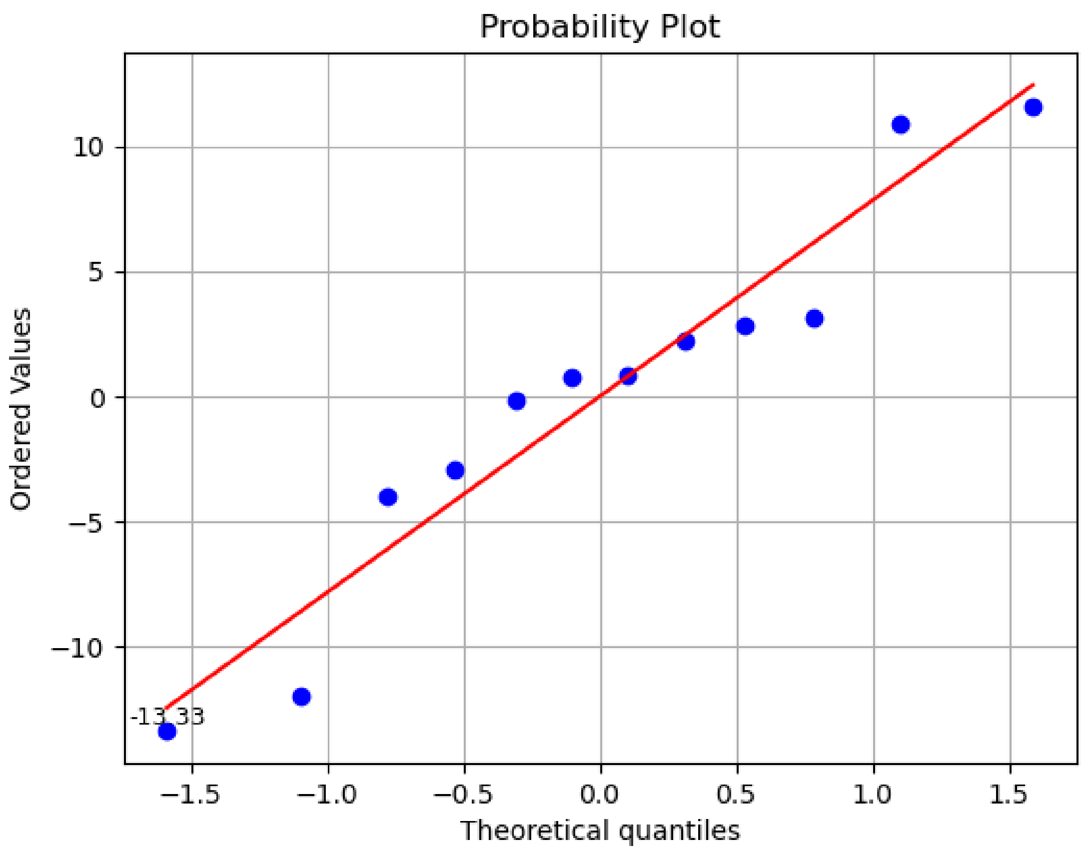
3.0.4 Brake_Horsepower

```
[96]: #
data_res = data * 1.0  # 1.0
for i in range(n):
    data_res[:, p + 1] = Y - Y_hat
df = pd.DataFrame(data_res, columns = ['intercept', 'P1', 'P2', 'P3', 'F_res'])
res = data_res[:, p + 1]
# res = model.resid
print(df.head())
```

```
   intercept      P1    P2     P3      F_res
0        1.0  2000.0  90.0  100.0   0.731289
1        1.0  1800.0  94.0   95.0 -13.328247
2        1.0  2400.0  88.0  110.0 -11.958476
3        1.0  1900.0  91.0   96.0   3.137442
4        1.0  1600.0  86.0  100.0  11.555798
```

3.0.5

```
[97]: #
osm, osr = stats.probplot(res, dist = 'norm', plot = plt)
x = osm[0][0]
y = osm[1][0]
plt.text(x, y, '%.2f' % float(y), ha='center', va= 'bottom', fontsize=9)
plt.grid()
plt.show()
```



[98]:
```
#
MSE = SSE / (n - p - 1)
# MSE = model.mse_resid
d = np.abs(y) / np.sqrt(MSE)
if d < 3:
    print(' ', round(y, 2), '   .')
else:
    print(' ', round(y, 2), '   ')
```

```
-13.33    .
```

3.0.6

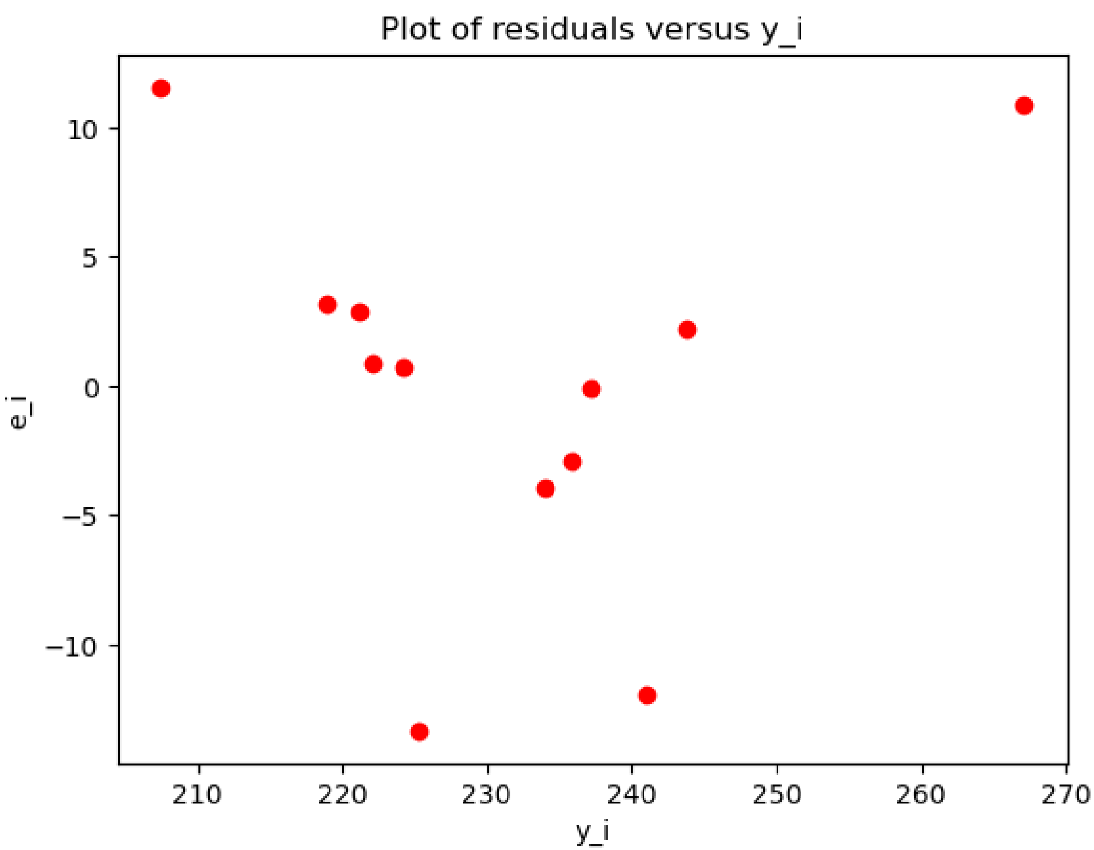
“ ” “ ” 0

:

0
X

```
[99]: #
      plt.scatter(Y_hat, res, c = 'red')
      plt.title('Plot of residuals versus y_i')
      plt.xlabel('y_i')
      plt.ylabel('e_i')
```

```
[99]: Text(0, 0.5, 'e_i')
```



dummy

ANOVA - x y

“ ”

4 Task 4: 3000 /min 90 100

Remark3:

$$(\hat{y}_f - t_{\alpha/2} * \sqrt{(1 + \frac{1}{n} + \frac{x_f - \bar{x}}{\sum_{i=1}^{n}(x_i - \bar{x})^2})\sigma^2}, \hat{y}_f + t_{\alpha/2} * \sqrt{(1 + \frac{1}{n} + \frac{x_f - \bar{x}}{\sum_{i=1}^{n}(x_i - \bar{x})^2})\sigma^2}$$

$$(\hat{y}_f - t_{\alpha/2} * \sqrt{(\frac{1}{n} + \frac{x_f - \bar{x}}{\sum_{i=1}^{n}(x_i - \bar{x})^2})\sigma^2}, \hat{y}_f + t_{\alpha/2} * \sqrt{(\frac{1}{n} + \frac{x_f - \bar{x}}{\sum_{i=1}^{n}(x_i - \bar{x})^2})\sigma^2}$$

4.0.1

[100]:
```
#   x_0  E(y_0)
def confidence_interval(x0):
    x0 = np.array(x0)
    Y0 = np.dot(x0.T, beta)
    delta0 = tVal * sigma * np.sqrt(x0.T @ C @ x0)
    Y0_int = [Y0 - delta0, Y0 + delta0]
    print(delta0)
    return Y0_int

x0 = [1]
for i in range(p):
    x0.append(int(input()))
print('  x = ', x0, ', E(y_0)     ', np.round(confidence_interval(x0), 4))
print('  = ',np.round(confidence_interval(x0), 4)[1]-np.
 ↪round(confidence_interval(x0), 4)[0])
```

```
3000
90
100
8.736173458784997
  x =  [1, 3000, 90, 100] , E(y_0)     [226.2457 243.7181]
8.736173458784997
8.736173458784997
  =  17.472399999999993
```

4.0.2

[101]:
```
#   x_0  y_0
def confidence_interval(x0):
    x0 = np.array(x0)
    Y0 = np.dot(x0.T, beta)
    delta1 = tVal * sigma * np.sqrt(1 + x0.T @ C @ x0)
```

```
    Y0_int = [Y0 - delta1, Y0 + delta1]
    return Y0_int

x0_ = [1]
for i in range(p):
    x0_.append(int(input()))
print('   x = ', x0_, ', y_0     ', np.round(confidence_interval(x0_), 4))
print('   = ',np.round(confidence_interval(x0), 4)[1]-np.
 ↪round(confidence_interval(x0), 4)[0])
```

```
3000
90
100
  x =  [1, 3000, 90, 100] , y_0     [212.8622 257.1016]
  =  44.23940000000002
```

44.23>17.47

4.0.3

1

[102]:
```
X = data[:, 0 : p + 1]
Y = data[:, -1]
#
X_mean = []
for k in range(p + 1):
    X_mean.append(np.mean(data[:, k]))  #    x
Y_mean = np.mean(data[:, -1])  #    y

#
X_std = (X - X_mean) / np.std(X)
Y_std = (Y - Y_mean) / np.std(Y)

# Do the multiple linear regression
df = pd.DataFrame(X_std, columns=['intercept_std', 'P1_std', 'P2_std',␣
 ↪'P3_std'])
df['F_std'] = Y_std
model_std = ols('F_std ~ P1_std + P2_std + P3_std', df).fit()
beta_std = model_std.params
print('   : \n', round(beta_std, 4))
Y_hat_std = model_std.fittedvalues
model_std.summary()
```

```
   :
 Intercept      0.0000
P1_std         0.6913
P2_std       202.2802
P3_std       120.4986
```

```
dtype: float64
```

[102]:

Dep. Variable:	F_std	**R-squared:**	0.807
Model:	OLS	**Adj. R-squared:**	0.734
Method:	Least Squares	**F-statistic:**	11.12
Date:	Sat, 07 Oct 2023	**Prob (F-statistic):**	0.00317
Time:	15:22:49	**Log-Likelihood:**	-7.1718
No. Observations:	12	**AIC:**	22.34
Df Residuals:	8	**BIC:**	24.28
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
Intercept	1.735e-17	0.156	1.12e-16	1.000	-0.359	0.359
P1_std	0.6913	0.289	2.390	0.044	0.024	1.358
P2_std	202.2802	54.489	3.712	0.006	76.628	327.932
P3_std	120.4986	34.491	3.494	0.008	40.961	200.036

Omnibus:	0.392	**Durbin-Watson:**	1.043
Prob(Omnibus):	0.822	**Jarque-Bera (JB):**	0.230
Skew:	-0.282	**Prob(JB):**	0.891
Kurtosis:	2.625	**Cond. No.**	350.

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

X axis=0 axis=0 np.std(X) X

axis=0 NumPy X_std

axis=0

2 np.array() “@” ’*‘ ’np.dot()’ ’@’ ‘@’

[ ]:

```

```