MULTIPLE LINEAR REGRESSION ON FEMALE EMPLOYMENT

November 26, 2022

[107]:
```
# Basic libraries
import pandas as pd
import numpy as np
#import seaborn as sns
import warnings
#from statsmodels.formula.api import ols
```

[93]:
```
#load data
fem_df= pd.read_csv("C:/Users/LILIAN/Desktop/Linear Regression By Levi/Female␣
↪Employment vs Socioeconimic Factors.csv")
```

1 PERFORM EXPLORATORY DATA ANALYSIS

[94]:
```
fem_df.head(3)
```

[94]:
```
   Year  PerFemEmploy FertilityRate  Ratio_MaletoFemale  PerFemEmployers  \
0  1995         24.30           3.71               28.33              0.1
1  1996         24.57           3.59               28.72              0.1
2  1997         24.82           3.48               29.18              0.1

   Agriculture  Industry  Services  Wage&Salaried  ContrFamWorkers  \
0        84.79      7.66      7.56          18.03            66.80
1        82.28      7.46     10.27          18.38            66.39
2        81.19      7.57     11.24          18.74            65.95

   OwnAccount  Vulnerable
0       15.07       81.87
1       15.14       81.52
2       15.21       81.16
```

[95]:
```
#Drop columns not relevant to the analysis
fem_df= fem_df.drop(columns = ['FertilityRate','Ratio_MaletoFemale',
                              'PerFemEmployers','ContrFamWorkers','OwnAccount',
                             'Vulnerable','Year'], axis=1)
```

[97]:
```python
fem_df.head(3)
```

[97]:

	PerFemEmploy	Agriculture	Industry	Services	Wage&Salaried
0	24.30	84.79	7.66	7.56	18.03
1	24.57	82.28	7.46	10.27	18.38
2	24.82	81.19	7.57	11.24	18.74

[131]:
```python
fem_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   PerFemEmploy   25 non-null     float64
 1   Agriculture    25 non-null     float64
 2   Industry       25 non-null     float64
 3   Services       25 non-null     float64
 4   Wage&Salaried  25 non-null     float64
dtypes: float64(5)
memory usage: 1.1 KB
```

[132]:
```python
fem_df.shape
```

[132]: (25, 5)

[134]:
```python
#the number of duplicates
fem_df.duplicated().sum()
```

[134]: 0

2 PERFORM DESCRIPTIVE STATISTICS

[135]:
```python
print (fem_df.mean())
```

```
PerFemEmploy     27.6808
Agriculture      70.2724
Industry         12.0188
Services         17.7104
Wage&Salaried    21.9652
dtype: float64
```

[137]:
```python
print(fem_df['PerFemEmploy'].corr(fem_df['Agriculture']))
print(fem_df['PerFemEmploy'].corr(fem_df['Industry']))
print(fem_df['PerFemEmploy'].corr(fem_df['Services']))
print(fem_df['PerFemEmploy'].corr(fem_df['Wage&Salaried']))
```

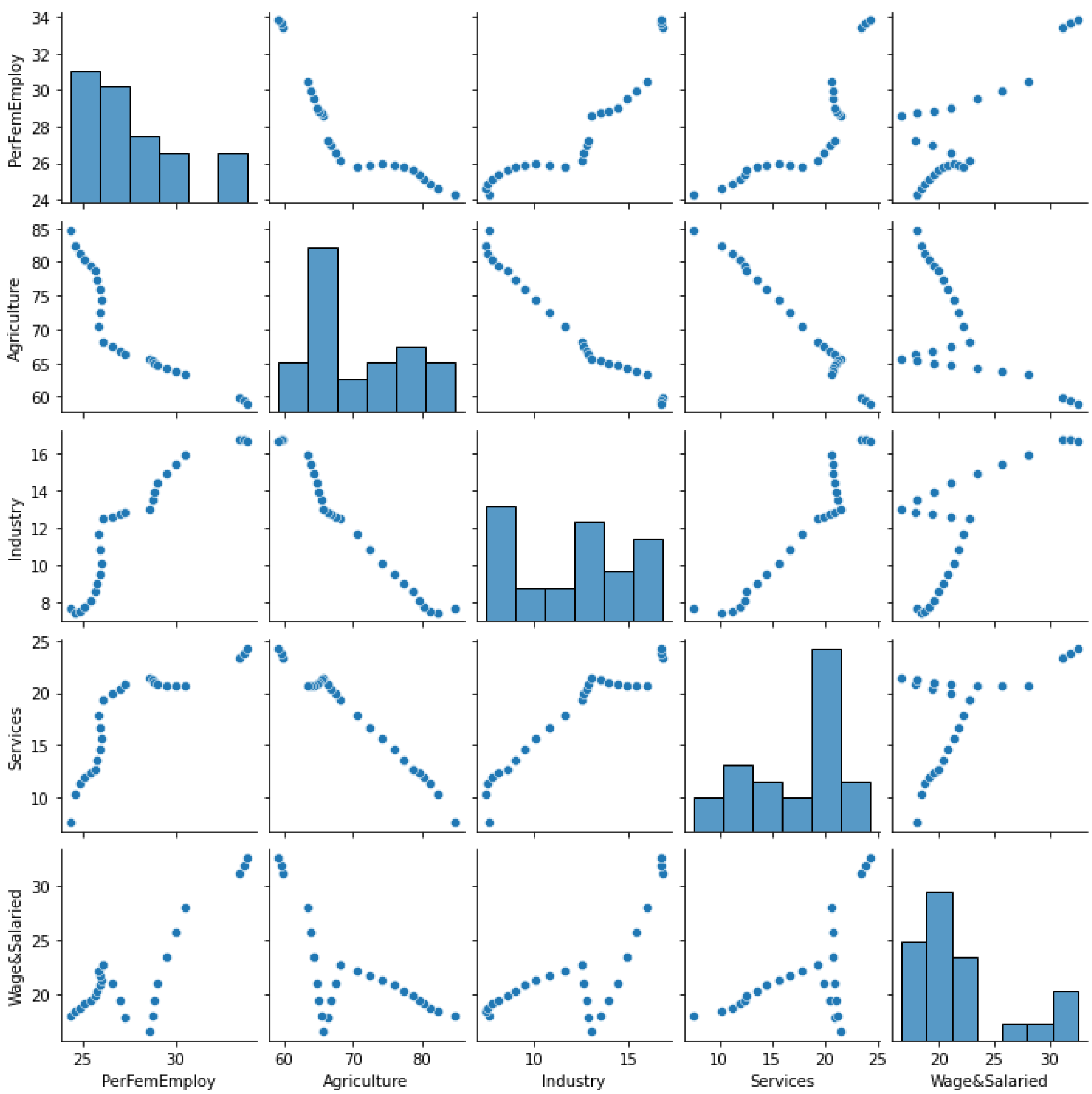
```
-0.8796996404083408
0.9175880178495721
0.8379940328345845
0.8230223960497459
```

[138]:
```python
#checkout for relationships
fem_df.corr().style.background_gradient(cmap ='coolwarm')
```

[138]: <pandas.io.formats.style.Styler at 0x205130aedf0>

[139]:
```python
# Visualizing the relationships between features using pair plots
sns.pairplot(data = fem_df, height = 2)
```

[139]: <seaborn.axisgrid.PairGrid at 0x20513c6e8e0>



```
[ ]: #Separate the features and target,we will give the variables to the X and␣
     ↪Y-axis.
     #X = fem_emp[['Agriculture','Industry','Services','Wage&Salaried']].values
     #y = fem_emp['PerFemEmploy'].values
```

```
[98]: #set up the dependent and the independent variables
      x = pd.DataFrame(fem_df.iloc[:,1:])
      y = pd.DataFrame(fem_df.iloc[:,:1])
```

```
[99]: #view the independent variable
      x
```

[99]:

	Agriculture	Industry	Services	Wage&Salaried
0	84.79	7.66	7.56	18.03
1	82.28	7.46	10.27	18.38
2	81.19	7.57	11.24	18.74
3	80.28	7.77	11.95	19.11
4	79.52	8.12	12.36	19.50
5	78.78	8.65	12.57	19.90
6	77.44	9.01	13.55	20.31
7	75.96	9.51	14.53	20.81
8	74.28	10.11	15.61	21.31
9	72.48	10.83	16.69	21.73
10	70.49	11.65	17.86	22.18
11	68.19	12.50	19.31	22.78
12	67.52	12.62	19.87	21.08
13	66.86	12.75	20.39	19.45
14	66.25	12.89	20.86	17.89
15	65.53	13.05	21.42	16.56
16	65.32	13.49	21.20	17.98
17	65.02	13.94	21.04	19.47
18	64.73	14.42	20.85	21.03
19	64.36	14.90	20.74	23.39
20	63.86	15.41	20.73	25.74
21	63.38	15.94	20.68	28.09
22	59.84	16.78	23.38	31.17
23	59.43	16.74	23.83	31.89
24	59.03	16.70	24.27	32.61

```
[100]: #glance at the dependent variable
       y
```

[100]:

	PerFemEmploy
0	24.30
1	24.57
2	24.82

```
3          25.11
4          25.38
5          25.63
6          25.78
7          25.89
8          25.96
9          25.89
10         25.83
11         26.11
12         26.56
13         27.00
14         27.22
15         28.56
16         28.72
17         28.87
18         28.99
19         29.49
20         29.96
21         30.47
22         33.44
23         33.65
24         33.82
```

[111]:

```
from sklearn.model_selection import train_test_split

#Divide the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

print("training and testing split was successful")
```

```
training and testing split was successful
```

[112]:

```
#checkout the shape of the train and test sets
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(20, 4)
(5, 4)
(20, 1)
(5, 1)
```

[114]:

```
from sklearn.linear_model import LinearRegression

#train the algorithm
regressor = LinearRegression()
```

```
regressor.fit(x_train, y_train)
```

[114]:

```
LinearRegression()
```

[116]:

```
#having a look at the coefficients that the model has chosen
c = pd.DataFrame(regressor.coef_,index=['Co-efficient']).transpose()
d = pd.DataFrame(x.columns, columns=['Attribute'])
```

[117]:

```
#concatenating the dataframes to compare
coeff_df = pd.concat([c,d], axis=1, join ='inner')
coeff_df
```

[117]:

	Co-efficient	Attribute
0	57.542157	Agriculture
1	57.970838	Industry
2	57.671491	Services
3	0.185354	Wage&Salaried

[120]:

```
#comparing the predicted value to the actual value
y_predict = regressor.predict(x_test)
y_predict = pd.DataFrame(y_predict,columns = ['predicted'])
y_predict
```

[120]:

	predicted
0	28.619129
1	27.696211
2	28.234941
3	32.498934
4	32.151210

[121]:

```
y_test
```

[121]:

	PerFemEmploy
12	26.56
14	27.22
11	26.11
24	33.82
22	33.44

[124]:

```
from sklearn import metrics
```

[126]:

```
#to evaluate the algorithm

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predict))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predict))
print('Root Mean squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
 ↪y_predict)))
```

```
Mean Absolute Error: 1.4540272508397032
Mean Squared Error: 2.477671077303806
Root Mean squared Error: 1.5740619674281588
```