Homework 5

August 3, 2023

1 Homework 5 Solutions

1.0.1 Annelise Lobo

1.0.2 08/02/2023

Answer each question by writing the Python code needed to perform the task. Please only use the libraries requested in each problem.

1.0.3 Problem 1

Load the interest_inflation data from the statsmodels library as a pandas data frame assigned to `df`. Use the function `df.head()` to view the first 5 rows of the data. Notice the first observation is indexed at 0. Unlike R, Python is a 0 based index language which means when you iterate or wish to view the first observation of a data object it will be at the index 0.

What do the columns `Dp` and `R` represent? (You can find this using the documentation)

[1]:
```
# your code here
from statsmodels.datasets.interest_inflation.data import load_pandas
df = load_pandas().data
df.head()

#According to the documentation, Dp represents the delta log gdp deflator
#R represents the nominal long term interest rate
```

[1]:

	year	quarter	Dp	R
0	1972.0	2.0	-0.003133	0.083
1	1972.0	3.0	0.018871	0.083
2	1972.0	4.0	0.024804	0.087
3	1973.0	1.0	0.016278	0.087
4	1973.0	2.0	0.000290	0.102

1.0.4 Problem 2

Import scipy as sp and numpy as np. Using the `mean()` and `var()` function from scipy, validate that both functions equate to their numpy counterparts against the column `Dp`.

By using the scipy library you should receive a warning message. What does the warning message indicate? Which function should you use going forward?

[2]:
```
# your code here
import scipy as sp
import numpy as np

scipy_mean = sp.mean(df["Dp"])
scipy_var = sp.var(df["Dp"])

numpy_mean = np.mean(df["Dp"])
numpy_var = np.var(df["Dp"])

print(scipy_mean == numpy_mean)
print(scipy_var == numpy_var)

#The warning is:
#DeprecationWarning: scipy.mean is deprecated and will be removed in SciPy 2.0.0
#The advice moving forward is to use numpy.mean instead
```

```
True
True
```

```
/var/folders/m3/124kzhfd3x5dqv275f985dyh0000gn/T/ipykernel_23190/892382137.py:5:
DeprecationWarning: scipy.mean is deprecated and will be removed in SciPy 2.0.0,
use numpy.mean instead
  scipy_mean = sp.mean(df["Dp"])
/var/folders/m3/124kzhfd3x5dqv275f985dyh0000gn/T/ipykernel_23190/892382137.py:6:
DeprecationWarning: scipy.var is deprecated and will be removed in SciPy 2.0.0,
use numpy.var instead
  scipy_var = sp.var(df["Dp"])
```

1.0.5 Problem 3

Fit an OLS regression (linear regression) using the statsmodels api where `y = df['Dp']` and `x = df['R']`. By default OLS estimates the theoretical mean of the dependent variable y. Statsmodels.ols does not fit a constant value by default so be sure to add a constant to `x`. Extract the coefficients into a variable named `res1_coefs`. See the documentation for `params`. Finally print the `summary()` of the model.

Documentation: https://www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.OLS.html

[5]:
```
# your code here
import statsmodels.api as sm

y = df['Dp']
x = df['R']
x = sm.add_constant(x)

res1_coefs = sm.OLS(y, x).fit()

print(res1_coefs.params)
```

```
print(res1_coefs.summary())
```

```
const   -0.003126
R        0.154512
dtype: float64
                            OLS Regression Results
==============================================================================
Dep. Variable:                     Dp   R-squared:                       0.018
Model:                            OLS   Adj. R-squared:                  0.009
Method:                 Least Squares   F-statistic:                     1.954
Date:                Thu, 03 Aug 2023   Prob (F-statistic):              0.165
Time:                        19:35:54   Log-Likelihood:                274.44
No. Observations:                 107   AIC:                            -544.9
Df Residuals:                     105   BIC:                            -539.5
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0031      0.008     -0.370      0.712      -0.020       0.014
R              0.1545      0.111      1.398      0.165      -0.065       0.374
==============================================================================
Omnibus:                       11.018   Durbin-Watson:                   2.552
Prob(Omnibus):                  0.004   Jarque-Bera (JB):                3.844
Skew:                          -0.050   Prob(JB):                        0.146
Kurtosis:                       2.077   Cond. No.                         61.2
==============================================================================

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

1.0.6 Probelm 4

Fit a quantile regression model using the statsmodels api using the formula `Dp ~ R`. By default quantreg creates a constant so there is no need to add one to this model. In your `fit()` method be sure to set `q = 0.5` so that we are estimating the theoritical median. Extract the coefficients into a variable named `res2_coefs`. Finally print the `summary()` of the model.

Documentation: https://www.statsmodels.org/dev/generated/statsmodels.regression.quantile_regression.QuantRe

[7]:
```
# your code here
import statsmodels.formula.api as smf

res2_coefs = smf.quantreg('Dp ~ R',
                          df).fit(q =0.5)

print(res2_coefs.params)
```

```
print(res2_coefs.summary())
```

```
Intercept   -0.005388
R            0.181800
dtype: float64
                         QuantReg Regression Results
==============================================================================
Dep. Variable:                     Dp   Pseudo R-squared:               0.02100
Model:                       QuantReg   Bandwidth:                      0.02021
Method:                  Least Squares   Sparsity:                       0.05748
Date:                 Thu, 03 Aug 2023   No. Observations:                  107
Time:                         19:38:33   Df Residuals:                      105
                                         Df Model:                            1
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.0054      0.013     -0.417      0.677      -0.031       0.020
R              0.1818      0.169      1.075      0.285      -0.153       0.517
==============================================================================
```

1.0.7 Problem 5

Part 1: Use the `type()` method to determine the type of `res1_coefs` and `res2_coefs`. Print the type in a Jupyter cell.

Part 2: In the next Jupyter cell show that `res1_coefs > res2_coefs`. What does the error mean? To resolve this error we must convert the data to an unnamed object or change the names of the objects. Since we are not focusing on pandas this week we will simply convert to a different data type.

Part 3: Now, do the same comparision using the `tolist()` function at the end of each object name.

Part 4: We performed two types of linear regression and compared their coefficients. Coefficients are essentially the rate at which x changes the values of y. Do some research on what OLS estimates versus what quantreg estimates and explain why we have two different coefficient estimates. In which cases do you think quantile regression will be useful? What about ordinary least squares regression?

[8]:
```
# your code here

#part 1
print(type(res1_coefs))
print(type(res2_coefs))

#part 2
#res1_coefs > res2_coefs results in a type error
print(str(res1_coefs)>str(res2_coefs))

#part 3
```

```
print(list(str(res1_coefs))>list(str(res2_coefs)))

#The OLS estimates the mean in the coefficient calculations.
#Quantreg uses the median for the coefficient calculations.
#A difference besides the actual numbers is that the mean will be effected by␣
 ↪outliers.
#If being effected by extremes is something important to your data, you should␣
 ↪use OLS.
#Quantreg may be useful for when you do not want extremes. For example if one␣
 ↪point is throwing off your entire dataset,  using quantreg may be more useful.
#OLS may be more useful if there are many outliers contributing to the data
#For example thhe cost of living in a country should include everyone in the␣
 ↪area, regardless of the outliers (OLS is better)
```

```
<class 'statsmodels.regression.linear_model.RegressionResultsWrapper'>
<class 'statsmodels.regression.linear_model.RegressionResultsWrapper'>
True
True
```