Homework 5

April 23, 2023

1 Homework 5

1.0.1 Sue Susman MEd, BSN, RN

1.0.2 April 23, 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]:
```
from statsmodels.datasets.interest_inflation import load_pandas
df = load_pandas().data
print(df.head())
```

```
     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
```

What do the columns Dp and R represent? (You can find this using the documentation)
year - 1972q2 - 1998q4 quarter - 1-4 Dp - Delta log gdp deflator R - nominal long term interest rate

Dp: Inflation rate, measured as the percentage increase in the consumer price index (CPI) from one year to the next. R: Interest rate, measured as the percentage yield on 3-month US Treasury bills, which are short-term debt securities issued by the US government.

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]:
```
import numpy as np
import scipy as sp
from statsmodels.datasets import interest_inflation

# Load data into a pandas dataframe
df = interest_inflation.load_pandas().data

# Extract the 'Dp' column as a numpy array
Dp = df['Dp'].to_numpy()

# Calculate the mean and variance using numpy
np_mean = np.mean(Dp)
np_var = np.var(Dp)

# Calculate the mean and variance using scipy
sp_mean = sp.mean(Dp)
sp_var = sp.var(Dp)

# Check if the mean and variance values are equivalent
print(np.isclose(np_mean, sp_mean))  # True
print(np.isclose(np_var, sp_var))  # True
```

```
True
True
```

```
/var/folders/7w/vf_3wx5d6kq9bm2666hl7yy80000gn/T/ipykernel_18172/2175973929.py:1
6: DeprecationWarning: scipy.mean is deprecated and will be removed in SciPy
2.0.0, use numpy.mean instead
  sp_mean = sp.mean(Dp)
/var/folders/7w/vf_3wx5d6kq9bm2666hl7yy80000gn/T/ipykernel_18172/2175973929.py:1
7: DeprecationWarning: scipy.var is deprecated and will be removed in SciPy
2.0.0, use numpy.var instead
  sp_var = sp.var(Dp)
```

1. What does the warning message indicate? The warning message indicates that the mean() and var() functions are deprecated in Scipy and will be removed in a future version.

2. Which function should you use going forward? Therefore, going forward it is recommended to use the numpy functions np.mean() and np.var() to calculate the mean and variance of the Dp column, respectively.

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

[3]:
```
import statsmodels.api as sm

# Define the dependent variable (y) and the independent variable (x)
y = df['Dp']
x = df['R']

# Add a constant term to the independent variable array
x = sm.add_constant(x)

# Fit the linear regression model
model = sm.OLS(y, x).fit()

# Extract the coefficients
res1_coefs = model.params

# Print the model summary
print(model.summary())
```

```
                            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:                Sun, 23 Apr 2023   Prob (F-statistic):              0.165
Time:                        20:01:54   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 Problem 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.QuantR

[4]:
```
import statsmodels.api as sm

# Define the dependent variable (y) and the independent variable (x)
y = df['Dp']
x = df['R']

# Fit the quantile regression model for the 0.5th quantile (i.e., the median)
q_model = sm.QuantReg(y, x).fit(q=0.5)

# Extract the coefficients
res2_coefs = q_model.params

# Print the model summary
print(q_model.summary())
```

```
                         QuantReg Regression Results
==============================================================================
Dep. Variable:                     Dp   Pseudo R-squared:               0.01709
Model:                       QuantReg   Bandwidth:                      0.02021
Method:                 Least Squares   Sparsity:                       0.05759
Date:                Sun, 23 Apr 2023   No. Observations:                  107
Time:                        20:01:54   Df Residuals:                      106
                                        Df Model:                            1
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
R              0.1104      0.036      3.029      0.003       0.038       0.183
==============================================================================
```

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?

[5]:
```
print(type(res1_coefs))
print(type(res2_coefs))
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

[6]:
```
print(res1_coefs > res2_coefs)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[6], line 1
----> 1 print(res1_coefs > res2_coefs)

File ~/opt/anaconda3/envs/DSE5002/lib/python3.11/site-packages/pandas/core/ops/
 ↪common.py:81, in _unpack_zerodim_and_defer.<locals>.new_method(self, other)
     77             return NotImplemented
     79 other = item_from_zerodim(other)
---> 81 return method(self, other)

File ~/opt/anaconda3/envs/DSE5002/lib/python3.11/site-packages/pandas/core/
 ↪arraylike.py:56, in OpsMixin.__gt__(self, other)
     54 @unpack_zerodim_and_defer("__gt__")
     55 def __gt__(self, other):
---> 56     return self._cmp_method(other, operator.gt)

File ~/opt/anaconda3/envs/DSE5002/lib/python3.11/site-packages/pandas/core/series.
 ↪py:6091, in Series._cmp_method(self, other, op)
   6088 res_name = ops.get_op_result_name(self, other)
   6090 if isinstance(other, Series) and not self._indexed_same(other):
-> 6091     raise ValueError("Can only compare identically-labeled Series objects")
   6093 lvalues = self._values
   6094 rvalues = extract_array(other, extract_numpy=True, extract_range=True)

ValueError: Can only compare identically-labeled Series objects
```

The ValueError indicates that we cannot compare the two Series objects res1_coefs and res2_coefs because they have different labels. To compare them, we need to first ensure that they have the same labels. We can do this by resetting the index of both Series and then comparing them. Here's how we can modify the code to compare the two Series:

[ ]:
```
# assuming res1_coefs and res2_coefs are Pandas dataframes
res1_coefs = res1_coefs.values.tolist()
```

```
res2_coefs = res2_coefs.values.tolist()

# comparing the two lists element-wise
print(res1_coefs > res2_coefs)
```

The reason why OLS and quantile regression can produce different coefficient estimates is because they are estimating different aspects of the relationship between the predictor and response variables and make different assumptions about the underlying distribution of the data.

The choice between using quantile regression versus ordinary least squares regression depends on the research question and the nature of the relationship between the dependent variable and independent variable(s).