AurianaAnderson.module05lab015

December 1, 2024

1 Homework 5

1.0.1 Auriana Anderson

1.0.2 November 24, 2024

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)

[63]:
```
from statsmodels.datasets.interest_inflation.data 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
```

The columns Dp represents:Delta log gdp deflator aka how much pricing levels are changing between periods (Log scale) The column R represents: normal long term interest rate

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?

[39]:
```
import scipy as sp
import numpy as np

dp_numpy_mean = np.mean(df['Dp'])
dp_numpy_var = np.var(df['Dp'])

print(dp_numpy_mean)
print(dp_numpy_var)

dp_scipy_mean = sp.stats.tmean(df['Dp'])
dp_scipy_var = sp.stats.tvar(df['Dp'])

print(dp_scipy_mean)
print(dp_scipy_var)
```

```
0.008397309906542055
0.00035296754186450404
0.008397309906542055
0.0003562974243349239
```

I tried this and could not get a warning to come up. I tried sp.mean and sp.var and it completely throws an error, The only way i could get this work is to use sp.stats for both. After a few google searches I found that the worning message is supposed to be that scipy has a deprecated mean(), meaning that its available for use but no longer recommened and will eventually be removed from newer versions.

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

[54]:
```
import statsmodels.api as sm
X = df['R']
Y = df['Dp']

X = sm.add_constant(X)
model = sm.OLS(Y,X)
results = model.fit()

res1_coefs = results.params

print(results.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, 24 Nov 2024   Prob (F-statistic):              0.165
Time:                        09:40:02   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

[56]:
```
import statsmodels.api as sm

X = df['R']
Y = df['Dp']

model = sm.QuantReg(Y, X)
results = model.fit(q = 0.5)

res2_coefs = results.params

print(results.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, 24 Nov 2024   No. Observations:                    107
Time:                         09:42:10   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?

[57]:
```
# Part 1: determine the type of `res1_coefs` and `res2_coefs`

print(type(res1_coefs))
print(type(res2_coefs))
```

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

[58]:
```
#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.

res1_coefs > res2_coefs
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[58], line 5
```

```
      1 #Part 2: In the next Jupyter cell show that `res1_coefs > res2_coefs`.␣
 ↪What does the error mean?
      2 #To resolve this error we must convert the data to an unnamed object or␣
 ↪change the names of the objects.
      3 #Since we are not focusing on pandas this week we will simply convert to a␣
 ↪different data type.
----> 5 res1_coefs > res2_coefs

File␣
 ↪~\AppData\Local\anaconda3\envs\DSE5002_Intro_to_R_python\Lib\site-packages\pandas\core\ops\c
 ↪py:76, in _unpack_zerodim_and_defer.<locals>.new_method(self, other)
     72             return NotImplemented
     74 other = item_from_zerodim(other)
---> 76 return method(self, other)

File␣
 ↪~\AppData\Local\anaconda3\envs\DSE5002_Intro_to_R_python\Lib\site-packages\pandas\core\array
 ↪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␣
 ↪~\AppData\Local\anaconda3\envs\DSE5002_Intro_to_R_python\Lib\site-packages\pandas\core\serie
 ↪py:6114, in Series._cmp_method(self, other, op)
   6111 res_name = ops.get_op_result_name(self, other)
   6113 if isinstance(other, Series) and not self._indexed_same(other):
-> 6114     raise ValueError("Can only compare identically-labeled Series objects")
   6116 lvalues = self._values
   6117 rvalues = extract_array(other, extract_numpy=True, extract_range=True)

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

The error is that it says it can only compare identalically-labeled series objects and since the OLS and QuantReg use different indicies, its having a hard time

[62]:
```
#Part 3: Now, do the same comparision using the `tolist()` function at the end␣
 ↪of each object name.

res1_coefs.tolist() > res2_coefs.tolist()
```

[62]:
```
False
```

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?

Ordinary least squares regression: This is used for linear regression analysis. It is trying to find an average relationship between x and y, y being the dependent variable. Its essentially trying to find a line of best fit within the data and minimize square differences. This is best to use when the data isn't heavily skewed and doesn't have many extreme outliers. Also when there is a clear established relationship between x and y.

quantile regression: Uses specific points in data to try and define the relationship between the two, this instead tries to minimize overall differences. I think this is the best when there are some heavy ouliers in the data that can skew results.

1.0.8 Problem 6

What are the advantages of using Python as a general purpose programming language? What are the disadvantages? Why do you think data scientists and machine learning engineers prefer Python over other statistically focused languages like R? Your answer should a paragraph for: (1) advantages, (2) disadvantages, and (3) why its popular. Please cite each source used in your answer.

Advantages:

Python is versatile and general purpose program. It has a rich and extensive ecosystem utilizing third party programs, adiing on to its functionality. This is one of my favorite parts about it, is that you can integrate it with different programs. Another point that I continually saw was that it has such a wide varitey of libraries and since it is well established, it also has a huge support system and community. This makes doing things in python easier and you have a lot of support from its community if you need help resolving issues. Another great thing about it, is that the language is fairly simple and easy to lear, therefore making it user friendly.

Source: https://www.geeksforgeeks.org/python-language-advantages-applications/

Disadvantages: One disadvanatage that I saw was that once code starts to get more complex, python can use a lot of memory. This can slow down processes in a work place. Another thing that I saw was that it lacks strictness, meaning that sometimes it can be too flexibile causing code to be created which can be hard to interpret. Variables don't necessarily have to be declared which can make it hard for someone coming in on a project. One last important point I saw was that It has a Global Interpreter Lock which limits one task to execute python code at a time. This means you can have multiple threads running in your python program but one can only be actively running. I found this interesting because most computers these days are multi-cored but you wouldn't necessarily be able to multi-task with python regardless.

Source: https://www.geeksforgeeks.org/python-language-advantages-applications/

Why its popular:

Python is popular because of its simplicity. I also saw numerous times that you can be more efficient with python because you can do more with less with pythons syntax. This cuts down on time you might have with other "tedious" languages where you may have to follow a stricter syntax. Again, as pointed out earlier, its well established, it has extensive libraries, and an extensive community. This means there are a lot of resources on it and if you need help, you are likely able to get help quickly with one of the many forums out there. It was also mentioned that it is widely used in academia and some students have started learning it as early as elementary school.

source: https://github.blog/developer-skills/programming-languages-and-frameworks/why-python-keeps-growing-explained/