HomeWork5Chris

April 8, 2024

Homework 5

Christopher Rodgers

April 8, 2024

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.data import load_pandas
import numpy as np
df = load_pandas().data
print(df.head())

#Dp represents Delta log gdp deflator, R represents normal long term interest␣
 ↪rate.
```

```
     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
```

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
np.mean(df['Dp']) == sp.mean(df['Dp'])
np.var(df['Dp']) == sp.var(df['Dp'])
```

```
#scipy.mean is deprecated and will be removed in SciPy 2.0.0, use numpy.mean␣
 ↪instead going forward
```

```
C:\Users\crodg\AppData\Local\Temp\ipykernel_9784\3511759810.py:4:
DeprecationWarning: scipy.mean is deprecated and will be removed in SciPy 2.0.0,
use numpy.mean instead
  np.mean(df['Dp']) == sp.mean(df['Dp'])
C:\Users\crodg\AppData\Local\Temp\ipykernel_9784\3511759810.py:5:
DeprecationWarning: scipy.var is deprecated and will be removed in SciPy 2.0.0,
use numpy.var instead
  np.var(df['Dp']) == sp.var(df['Dp'])
```

[2]: True

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]:
```
# your code here
import statsmodels.api as sm
y = df['Dp']
x = df['R']
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:                Mon, 08 Apr 2024   Prob (F-statistic):              0.165
Time:                        13:11:34   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.
```

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.QuantR

[4]:
```
# your code here
import statsmodels.formula.api as smf
mod = smf.quantreg("Dp~R", data=df)
res = mod.fit(q=0.5)
res2_coefs = res.params

print(res.summary())
```

```
                         QuantReg Regression Results
==============================================================================
Dep. Variable:                     Dp   Pseudo R-squared:              0.02100
Model:                       QuantReg   Bandwidth:                     0.02021
Method:                 Least Squares   Sparsity:                      0.05748
Date:                Mon, 08 Apr 2024   No. Observations:                  107
Time:                        13:11:57   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
==============================================================================
```

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]:
```
# your code here
print(type(res1_coefs))
print(type(res2_coefs))
```

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

[6]:
```
res1_coefs > res2_coefs
```

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

File ~\anaconda3\Lib\site-packages\pandas\core\ops\common.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 ~\anaconda3\Lib\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 ~\anaconda3\Lib\site-packages\pandas\core\series.py:5798, in Series.
 ↪_cmp_method(self, other, op)
   5795 res_name = ops.get_op_result_name(self, other)
   5797 if isinstance(other, Series) and not self._indexed_same(other):
-> 5798     raise ValueError("Can only compare identically-labeled Series␣
 ↪objects")
   5800 lvalues = self._values
   5801 rvalues = extract_array(other, extract_numpy=True, extract_range=True)

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

cannot be compared using >

[7]:
```
res1_coefs.tolist() > res2_coefs.tolist()
```

[7]: `True`

Ordinary Least Squares (OLS) regression is appropriate when the focus is on estimating the average effect of predictors on the dependent variable, assuming linearity. Effect is of primary interest and relationship between variables is reasonable linear.

Quantile regression is useful in cases where the researcher is interested in understanding how the effects of predictors vary across different quantiles of the dependent variable's distribution.

We have two different coefficient estimates because Ordinary Least Squares (OLS) regression and Quantile Regression (quantreg) employ different approaches to estimating the relationship between variables. OLS focuses on estimating the conditional mean, while quantreg estimates conditional quantiles, leading to distinct coefficient estimates that capture different aspects of the data's distribution.

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.

(1) Python's advantages as a general-purpose programming language include its simplicity and readability, making it accessible for beginners and allowing for rapid development (Sunbul, 2024). Python boasts a vast ecosystem of libraries and frameworks for various purposes, facilitating tasks from web development to scientific computing. Its versatility allows it to be used seamlessly across different platforms and operating systems.

(2) Despite its strengths, Python does have some drawbacks. Its performance can be slower than lower-level languages like C or C++, which may pose challenges for computationally intensive tasks or real-time applications (Hadi J, 2024). Additionally, Python's dynamic typing can lead to potential runtime errors that might not be caught until execution. While Python's simplicity is advantageous for beginners, it may need more advanced features in other languages, limiting its suitability for specific niche applications.

(3) Due to its versatility and ecosystem, Data scientists and machine learning engineers often prefer Python over statistically focused languages like R (Let's Decode, 2023). Python's extensive libraries, such as NumPy, pandas, and scikit-learn, provide powerful tools for data manipulation, analysis, and machine learning. Python's straightforward syntax and readability make prototyping and deploying machine learning models easier. Its integration with other technologies, such as web frameworks like Django and Flask, enables seamless development and deployment of machine learning applications. Overall, Python's simplicity, versatility, and extensive libraries make it a preferred choice for data science and machine learning tasks.

References

Decode, L. (2023, November 8). Why should you choose python over R for data science?. Medium. https://medium.com/@debopamdeycse19/python-vs-r-for-data-science-which-should-i-learn-a236c197c2bd

Hadi J., M. H. (2024, January 11). Python is bad, and here is why?. Medium. https://medium.com/@jiwani.mhadi/python-is-bad-and-here-is-why-3d398a802ad7#:~:text=Python%20is%20an%20interpreted%20language,computing%20or%20resource%2Dintensi

Sunbul. (2024, January 31). Exploring the advantages and disadvantages of python. RedSwitches. https://www.redswitches.com/blog/advantages-and-disadvantages-of-python/

[ ]: