Chandonnet.Module05Lab01

November 26, 2022

1 Homework 5

1.0.1 Ray Chandonnet

1.0.2 November 26, 2022

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)

[8]:

```python
# your code here
from statsmodels.datasets.interest_inflation.data import load_pandas
df = load_pandas().data
df.head()

# Dp is the Delta log GDP deflator
# R is the nominal long term interest rate
```

[8]:

```
     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?

[9]:

```python
import scipy as sp
import numpy as np
mean_from_scipy = sp.mean(df['Dp'])
mean_from_numpy =np.mean(df['Dp'])
equivalent1 = mean_from_numpy == mean_from_scipy
equivalent1
var_from_scipy = sp.var(df['Dp'])
var_from_numpy =np.var(df['Dp'])
equivalent2 = var_from_numpy == var_from_scipy
equivalent2

# The error message when you calculate mean with scipy warns of "deprecation". ␣
 ↪Deprecation means that a function
# is likely to be discontinued in later releases.  So since the deprecation␣
 ↪warning comes from using scipy, we should
# use the mean function from numpy going forward
```

```
/var/folders/vv/qsdw41b97s35x1dgcrhkvxfm0000gn/T/ipykernel_23292/2009063642.py:3
: DeprecationWarning: scipy.mean is deprecated and will be removed in SciPy
2.0.0, use numpy.mean instead
  mean_from_scipy = sp.mean(df['Dp'])
/var/folders/vv/qsdw41b97s35x1dgcrhkvxfm0000gn/T/ipykernel_23292/2009063642.py:7
: DeprecationWarning: scipy.var is deprecated and will be removed in SciPy
2.0.0, use numpy.var instead
  var_from_scipy = sp.var(df['Dp'])
```

[9]:

```
True
```

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

[10]:

```python
import statsmodels.api as sm
x = df['R']
y = df['Dp']
x= sm.add_constant(x)
result1 = sm.OLS(y,x).fit()
res1_coefs=result1.params
print(result1.summary())
print(res1_coefs)
```

```
                            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:                Sat, 26 Nov 2022   Prob (F-statistic):              0.165
Time:                        14:10:53   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.
const   -0.003126
R        0.154512
dtype: float64
```

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.QuantRe

[11]:

```
import statsmodels.formula.api as smf
result2 = smf.quantreg("Dp ~ R",df).fit(q=0.5)
res2_coefs =result2.params
print(result2.summary())
print(res2_coefs)
```

```
                         QuantReg Regression Results
==============================================================================
Dep. Variable:                     Dp   Pseudo R-squared:              0.02100
Model:                       QuantReg   Bandwidth:                     0.02021
Method:                 Least Squares   Sparsity:                      0.05748
```

```
Date:                  Sat, 26 Nov 2022   No. Observations:                  107
Time:                          14:10:55   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
==============================================================================
Intercept   -0.005388
R            0.181800
dtype: float64
```

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?

[12]:
```
type1 = type(res1_coefs)
type2 = type(res2_coefs)
print(type1)
print(type2)
test_greater1 = res1_coefs > res2_coefs
print(test_greater1)
# This generates an error, since comparison operations cannot be performed on␣
 ↪data type "class", which has multiple
# pieces of data with their values and attributes.  So we convert these to list␣
 ↪types to make a comparison
```

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/var/folders/vv/qsdw41b97s35x1dgcrhkvxfm0000gn/T/ipykernel_23292/1570535550.py in␣
 ↪<module>
      3 print(type1)
      4 print(type2)
----> 5 test_greater1 = res1_coefs > res2_coefs
      6 print(test_greater1)
      7 # This generates an error, since comparison operations cannot be performed␣
 ↪on data type "class", which has multiple

~/opt/anaconda3/envs/DSE5002/lib/python3.9/site-packages/pandas/core/ops/common.py␣
 ↪in new_method(self, other)
     70         other = item_from_zerodim(other)
     71 
---> 72         return method(self, other)
     73 
     74     return new_method

~/opt/anaconda3/envs/DSE5002/lib/python3.9/site-packages/pandas/core/arraylike.py␣
 ↪in __gt__(self, other)
     56     @unpack_zerodim_and_defer("__gt__")
     57     def __gt__(self, other):
---> 58         return self._cmp_method(other, operator.gt)
     59 
     60     @unpack_zerodim_and_defer("__ge__")

~/opt/anaconda3/envs/DSE5002/lib/python3.9/site-packages/pandas/core/series.py in␣
 ↪_cmp_method(self, other, op)
   6235 
   6236         if isinstance(other, Series) and not self._indexed_same(other):
-> 6237             raise ValueError("Can only compare identically-labeled Series␣
 ↪objects")
   6238 
   6239         lvalues = self._values

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

[14]:

```
list1 = res1_coefs.tolist()
list2 = res2_coefs.tolist()
test_greater2 = list1 > list2
print(test_greater2)
# The Ordinary Least Squares ("OLS") method estimates the dependent variable as␣
 ↪the mean (average) result across
# given independent variables   The Quantile method estimates the dependent␣
 ↪variable as the median result across
# given independent variables.  Mean vs. median is an important statistical␣
 ↪distinction.  When your data is pretty
# normally distributed, mean and median are quite similar and you can use either␣
 ↪method - though OLS is likely preferred
```

```
# However, when you have outlier data that can skew the results of a mean calculation, the use of median is preferable
# which translates into using a Quantile regression method.
```

```
True
```

[ ]:

```
```

[ ]:

```
```