Homework 5Simonsen

April 11, 2024

1 Homework 5

1.0.1 Steven Simonsen

1.0.2 4/7/2024

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)

```
[19]: from statsmodels.datasets.interest_inflation.data import load_pandas
```

```
[20]: df = load_pandas().data
```

```
[21]: df.head()
```

```
[21]:      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
```

```
[22]: #Column Dp represents Delta log gdp deflator, and R represents the nominal 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?

```
[23]: import numpy as np
```

```
[24]: import scipy as sp
      from scipy import stats
```

```
[25]: np.mean(df['Dp'])
```

```
[25]: 0.008397309906542055
```

```
[30]: #Here is an example of the error I got when trying to print 'mean' directly from␣
       ↪within scipy as of 4/10. See addl. explanation below.
      sp.mean(df['Dp'])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~\anaconda3\envs\DSE5002\Lib\site-packages\scipy\__init__.py:150, in␣
 ↪__getattr__(name)
    149 try:
--> 150     return globals()[name]
    151 except KeyError:

KeyError: 'mean'

During handling of the above exception, another exception occurred:

AttributeError                            Traceback (most recent call last)
Cell In[30], line 2
      1 #Here is an example of the error I got when trying to print 'mean'␣
 ↪directly from within scipy as of 4/10. See addl. explanation below.
----> 2 sp.mean(df['Dp'])

File ~\anaconda3\envs\DSE5002\Lib\site-packages\scipy\__init__.py:152, in␣
 ↪__getattr__(name)
    150     return globals()[name]
    151 except KeyError:
--> 152     raise AttributeError(
    153         f"Module 'scipy' has no attribute '{name}'"
    154     )

AttributeError: Module 'scipy' has no attribute 'mean'
```

```
[31]: #Using tmean from stats to show equality
      stats.tmean(df['Dp'])
```

```
[31]: 0.008397309906542055
```

```
[32]: np.mean(df['Dp'])==stats.tmean(df['Dp'])
```

```
[32]: True
```

```
[33]: sp.var(df['Dp'])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~\anaconda3\envs\DSE5002\Lib\site-packages\scipy\__init__.py:150, in
 ↪__getattr__(name)
    149 try:
--> 150     return globals()[name]
    151 except KeyError:

KeyError: 'var'

During handling of the above exception, another exception occurred:

AttributeError                            Traceback (most recent call last)
Cell In[33], line 1
----> 1 sp.var(df['Dp'])

File ~\anaconda3\envs\DSE5002\Lib\site-packages\scipy\__init__.py:152, in
 ↪__getattr__(name)
    150     return globals()[name]
    151 except KeyError:
--> 152     raise AttributeError(
    153         f"Module 'scipy' has no attribute '{name}'"
    154     )

AttributeError: Module 'scipy' has no attribute 'var'
```

```
[34]: np.var(df['Dp'])
```

```
[34]: 0.00035296754186450404
```

```
[35]: #Results are not exactly the same. Need to round in order to make statement true
       ↪as seen below.
      stats.tvar(df['Dp'])
```

```
[35]: 0.0003562974243349239
```

```
[36]: np.round(np.var(df['Dp']),4)==round(stats.tvar(df['Dp']),4)
```

```
[36]: True
```

[37]:
```
#Update 4.10.24 - Although this worked earlier in the week, I'm now receiving␣
↪error messages instead of warnings. To fix this, I read through the scipy
#documentation for the version I'm using (1.11.4). Here, the 'mean' attribute␣
↪does not exist. So instead, I used the tmean to show equality.
#Although I don't get the intended warning anymore, this does resolve the hard␣
↪error message I was receiving.

#I did receive warnings. This means that the scipy functions used are still able␣
↪to be used for now, but will be removed in SciPy version 2.0.0.
#In preparation for this removal, I should use the numpy counterpart functions␣
↪instead since I have proved both to produce identical results.
```

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

[38]:
```
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
results.summary()
```

[38]:

Dep. Variable:	Dp	**R-squared:**	0.018
Model:	OLS	**Adj. R-squared:**	0.009
Method:	Least Squares	**F-statistic:**	1.954
Date:	Thu, 11 Apr 2024	**Prob (F-statistic):**	0.165
Time:	15:37:08	**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

[39]:
```
import statsmodels.formula.api as smfi
model=smfi.quantreg("Dp ~ R", data=df)
results=model.fit(q=0.5)
res2_coefs=results.params
print(results.summary())
```

```
                         QuantReg Regression Results
==============================================================================
Dep. Variable:                     Dp   Pseudo R-squared:              0.02100
Model:                       QuantReg   Bandwidth:                     0.02021
Method:                 Least Squares   Sparsity:                      0.05748
Date:                Thu, 11 Apr 2024   No. Observations:                  107
Time:                        15:37:13   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?

[40]:
```
print(type(res1_coefs))
print(type(res2_coefs))
```

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

[41]:
```
res1_coefs > res2_coefs
#After researching this error, I have found it occurs in Python because Pandas␣
 ↪uses intrinsic data alignment, or indices to perform
#Operations. Because these variables have different indices, I received the␣
 ↪error message.
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[41], line 1
----> 1 res1_coefs > res2_coefs

File ~\anaconda3\envs\DSE5002\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\envs\DSE5002\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\envs\DSE5002\Lib\site-packages\pandas\core\series.py:6105, in␣
 ↪Series._cmp_method(self, other, op)
   6102 res_name = ops.get_op_result_name(self, other)
   6104 if isinstance(other, Series) and not self._indexed_same(other):
-> 6105     raise ValueError("Can only compare identically-labeled Series objects")
   6107 lvalues = self._values
   6108 rvalues = extract_array(other, extract_numpy=True, extract_range=True)

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

[42]:
```
res1_coefs.tolist()
```

[42]:
```
[-0.003126143768981994, 0.15451247409537258]
```

[43]:
```
res2_coefs.tolist()
```

[43]:
```
[-0.005388162484555814, 0.18179963970851887]
```

[44]:
```
res1_coefs.tolist() > res2_coefs.tolist()
```

[44]:
```
True
```

[45]:
```
#After watching the lectures and doing research, OLS estimates the mean and␣
↪quant reg estimates quantiles, where q=0.5 is the median.
#Therefore, the R coefficient effects the median in the quantile regression by 0.
↪1818, and the R coefficient effects the mean in
#the least squares linear regression by 0.1545. I think quantile regression␣
↪would be useful for cases wheree there are significant
#outliers, as the median is less sensitive to outliers in the data as opposed to␣
↪the mean in OLS.
```

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.

The advantages of using Python as a general purpose programming language are vast. For one, ths use of Python can be applied in to a wide range of use-cases. For example, If somebody wanted to build an online application that also contained mobile compatibility, Python would be a great choice. In fact, some of the largest companies within the consumer application space, such as Instagram, are built using Python. Another advantage to Python is the lightweight, compatible framework across operating systems. Python can be utilized using Linux, Windows, and MacOS. In large companies, numerous operating systems are common, and often necessary to perform various tasks in an optimized fashion. Therefore, it is incredibly beneficial that such a large programming language such as Python is cross-platform compatible. This brings me to another advantage to using the language, which is the large user base and following the language has. Python is an open source, and therefore has a large and active community that contributes to its development and provides support(1). For these attractive advantages, along with the versatility (ex: data analysis to application development and machine learning), I think this makes Python a popular choice among Data Scientists and Machine Learning Engineers.

One of the disadvantages to using Python is that it is an interpreted language, which means that it can be slower than compiled languages like C or Java. This can be an issue for performance-intensive tasks(1). Additionally, because of the open source nature of the language, there is a very large number of libraries and packages to use. This can be a good thing, but also a disadvantage because it can lead to bugs and runtime errors if different people are using different versions of the same libararies and packages. Finally, the language is dynamically typed, which means that the types of variables can change at runtime. This can make it more difficult to catch errors and can lead to bugs(1).

Overall, Python is a very popular language because of the flexibility it provides, and vast capability to do many of the functions necessary for data science and machine learning. Some of the biggest organizations in the world use Python for this reason, and in addition to Instagram mentioned above, Google, Yahoo, Samsung, Microsoft, and Facebook use it as well. This also further contributes to the popularity of the language, and increases the demand to resolve bugs and create new packages and libraries. In conclusion, the advantages to using Python far outwiegh the disadvantages, and in my opinion, it will be used widely for many years to come.

Resources: 1. GfG. (2023, November 1). Python language advantages and applications. GeeksforGeeks. https://www.geeksforgeeks.org/python-language-advantages-applications/