Homework 5

August 4, 2023

1 Homework 5

1.0.1 Adeline Casali

1.0.2 8/3/2023

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)

[26]:
```
import pandas as pd

from statsmodels.datasets.interest_inflation import load_pandas
df = load_pandas().data

df.head()

# According to the documentation, Dp represents the Delta log gdp deflator,
# and R represents the nominal long term interest rate.
```

[26]:

	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?

[27]:
```
import scipy as sp
import numpy as np

dp_values = df["Dp"]

# Using scipy
mean_scipy = sp.mean(dp_values)
variance_scipy = sp.var(dp_values)

# Using numpy
mean_numpy = np.mean(dp_values)
variance_numpy = np.var(dp_values)

# Comparing the results
print("Mean (Scipy):", mean_scipy)
print("Mean (Numpy):", mean_numpy)

print("Variance (Scipy):", variance_scipy)
print("Variance (Numpy):", variance_numpy)

mean_scipy == mean_numpy
variance_scipy == variance_numpy

# The warning indicates that scipy.mean is being decomissioned in the next␣
 ↪version of SciPy and numpy.mean should
# be used in its place.
```

```
Mean (Scipy): 0.008397309906542055
Mean (Numpy): 0.008397309906542055
Variance (Scipy): 0.00035296754186450404
Variance (Numpy): 0.00035296754186450404
```

```
/var/folders/rl/bylb2hfn6bj44jtcpcgs931c0000gn/T/ipykernel_14519/1991367081.py:7
: DeprecationWarning: scipy.mean is deprecated and will be removed in SciPy
2.0.0, use numpy.mean instead
  mean_scipy = sp.mean(dp_values)
/var/folders/rl/bylb2hfn6bj44jtcpcgs931c0000gn/T/ipykernel_14519/1991367081.py:8
: DeprecationWarning: scipy.var is deprecated and will be removed in SciPy
2.0.0, use numpy.var instead
  variance_scipy = sp.var(dp_values)
```

[27]:
```
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

[28]:
```
import statsmodels.api as sm

y = df['Dp']
x = df['R']

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

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

# Extract the coefficients
res1_coefs = result.params

# Print the summary
print(result.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:                Fri, 04 Aug 2023   Prob (F-statistic):              0.165
Time:                        14:43:43   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

[29]:
```
import statsmodels.formula.api as smf

data = df[['Dp', 'R']]

# Fit the quantile regression model
quant_model = smf.quantreg('Dp ~ R', data)
result = quant_model.fit(q=0.5)

# Extract the coefficients
res2_coefs = result.params

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

```
                         QuantReg Regression Results
==============================================================================
Dep. Variable:                     Dp   Pseudo R-squared:               0.02100
Model:                       QuantReg   Bandwidth:                      0.02021
Method:                 Least Squares   Sparsity:                       0.05748
Date:                Fri, 04 Aug 2023   No. Observations:                  107
Time:                        14:43:48   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?

[30]:
```
# Part 1
print("Type of res1_coefs:", type(res1_coefs))
print("Type of res2_coefs:", type(res2_coefs))

# Part 2
print(res1_coefs > res2_coefs)
# There is an error becuase they are pandas series object, which cannot be␣
 ↪directly compared with operators like >

# Part 3
print(res1_coefs.tolist() > res2_coefs.tolist())

# Part 4
# OLS works to provide the best fitting line by minimizing the sum of squared␣
 ↪residuals. On the other hand,
# quantile regression uses the median and the sum of absolute deviations to␣
 ↪determine the line of best fit.
# Because of this, OLS distributions work best with normally distributed data␣
 ↪and quantile regression deals well
# with non-normally distributed data and is robust to outliers.
```

```
Type of res1_coefs: <class 'pandas.core.series.Series'>
Type of res2_coefs: <class 'pandas.core.series.Series'>
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[30], line 6
      3 print("Type of res2_coefs:", type(res2_coefs))
      5 # Part 2
----> 6 print(res1_coefs > res2_coefs)
      7 # There is an error becuase they are pandas series object, which cannot b␣
 ↪directly compared with operators like >
      8
      9 # Part 3
     10 print(res1_coefs.tolist() > res2_coefs.tolist())
```

```
File ~/anaconda3/envs/DSE5002/lib/python3.10/site-packages/pandas/core/ops/commor
 ↪py:72, in _unpack_zerodim_and_defer.<locals>.new_method(self, other)
     68             return NotImplemented
     70 other = item_from_zerodim(other)
---> 72 return method(self, other)

File ~/anaconda3/envs/DSE5002/lib/python3.10/site-packages/pandas/core/arraylike.
 ↪py:58, in OpsMixin.__gt__(self, other)
     56 @unpack_zerodim_and_defer("__gt__")
     57 def __gt__(self, other):
---> 58     return self._cmp_method(other, operator.gt)

File ~/anaconda3/envs/DSE5002/lib/python3.10/site-packages/pandas/core/series.py:
 ↪6237, in Series._cmp_method(self, other, op)
   6234 res_name = ops.get_op_result_name(self, other)
   6236 if isinstance(other, Series) and not self._indexed_same(other):
-> 6237     raise ValueError("Can only compare identically-labeled Series objects')
   6239 lvalues = self._values
   6240 rvalues = extract_array(other, extract_numpy=True, extract_range=True)

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

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.

There are many advantages of using Python for data science and machine learning over more statistically focused languages such as R. Python is a general programming language and highly versatile, with a vast array of valuable libraries. It is suitable for web development, scientific computing, data analysis, machine learning, and more. It also has clear and readable syntax, making it easy to learn and simple to write clean code. In addition to this, because it is so popular, it has a community of developers worldwide that provide updates, documentation, and resources for troubleshooting.

On the other hand, there are some disadvantages to Python. It could be better for mobile development, has a high memory consumption, and is insecure regarding database access. It is also considered relatively slow compared to Java or C, as it is a dynamically typed language. Furthermore, it doesn't contain as robust statistical modeling software as a more statistics-focused language like R.

Overall, Python is popular due to all the advantages listed above, its community, ease of use, and integration capabilities. It is considered an essential tool in data scientists' toolboxes. In addition, integrating Jupyter Notebooks with Python allows for easy collaboration, sharing of code, and interactive computing. These factors have all led to Python's success and rapid growth in popularity.

Sources: Advantages and disadvantages of python. Tagline Infotech. (2023, July 17). https://taglineinfotech.com/advantages-and-disadvantages-of-python/#Disadvantages_Of_Python_Programming GeeksforGeeks. (2021, July 29). Disadvantages of python. GeeksforGeeks. https://www.geeksforgeeks.org/disadvantages-of-python/ IBM Cloud Team. (2021, March 23). Python vs. R: What's the difference? IBM Blog. https://www.ibm.com/blog/python-vs-r/ Nagpal, A., & Gabrani, G. (2019). Python for data analytics, scientific and technical applications. 2019 Amity International Conference on Artificial Intelligence (AICAI). https://doi.org/10.1109/aicai.2019.8701341