MSIS 5223: Tutorial 4 – Descriptive Statistics in Python

Instructions

The purpose of this tutorial is to help you become familiar with performing descriptive statistics in Python using Pandas. This is an important step in familiarizing yourself with your data so you can better determine what type of analysis to perform later on. For this example, use both ozone.data.txt and car.test.frame.txt.

Remember, import the libraries Pandas, numpy, and matplotlib.pyplot. Also, if you need to change your working directory, import the os library.

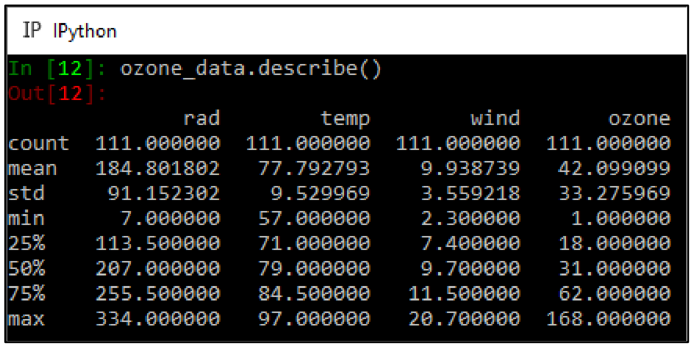
To follow along with the examples used in this document, use the Python file *Data Derivation and Selection*.

1. Summarizing the Data

After obtaining any data, the first thing you should do is familiarize yourself with it. This allows you to make a more informed decision as to the kinds of statistical models you can build with your data. Often, this merely entails looking at the mean, median, spread, shape, and datatypes within your data.

There are many ways to assess the basic descriptive information of the data. Python's Pandas library provides two functions that behave similarly to summary() and str() in R. These two functions are describe() and dtypes. The former provides an overview like summary(), providing basic descriptive information for numeric and categorical data. The attribute, dtypes, only provides the datatype of the columns, unlike str() in R that provides more beyond datatypes.

Load the ozone data and use describe(). The output provides count, mean, standard deviation, minimum value, maximum value, and the values for the 25th, 50th, and 75th quartiles. See the figure to the side.



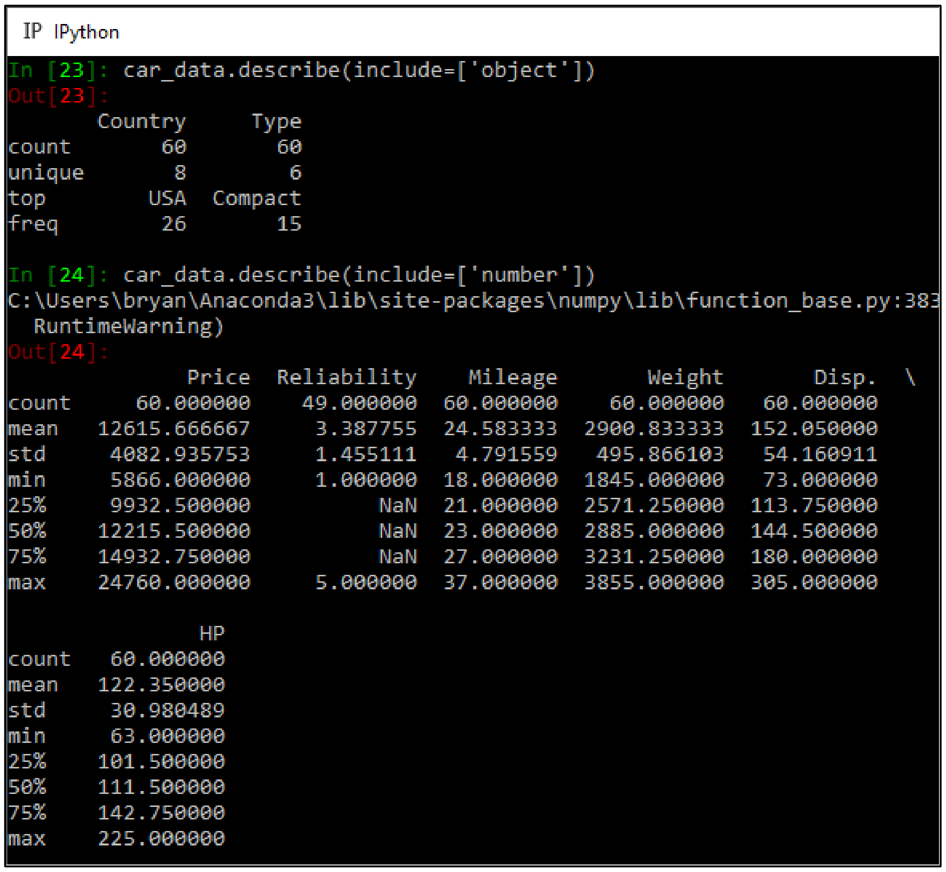
```
In [12]: ozone_data.describe()
Out[12]:
              rad        temp        wind       ozone
count  111.000000  111.000000  111.000000  111.000000
mean   184.801802   77.792793    9.938739   42.099099
std     91.152302    9.529969    3.559218   33.275969
min      7.000000   57.000000    2.300000    1.000000
25%    113.500000   71.000000    7.400000   18.000000
50%    207.000000   79.000000    9.700000   31.000000
75%    255.500000   84.500000   11.500000   62.000000
max    334.000000   97.000000   20.700000  168.000000
```

Figure 1 Using Describe() in Python

Perform the same operation on the car data. Recall, the car data contains eight columns of data. Unfortunately, the output only presents six columns worth of descriptive data. The reason is, the two columns left out are string/categorical, not numeric. The describe() function in Pandas has additional arguments (and some beyond these):

- describe(include=['object']): focuses just on string data
- describe(include=['categorical']): focuses just on categorical data
- describe(include=['number']): solely looks at numerical data
- describe(include='all'): forces Python to assess both types

Return to the car data and use the include=['object'] argument. Your output should look similar to that found in Figure 2 below. Note the differences of the output.

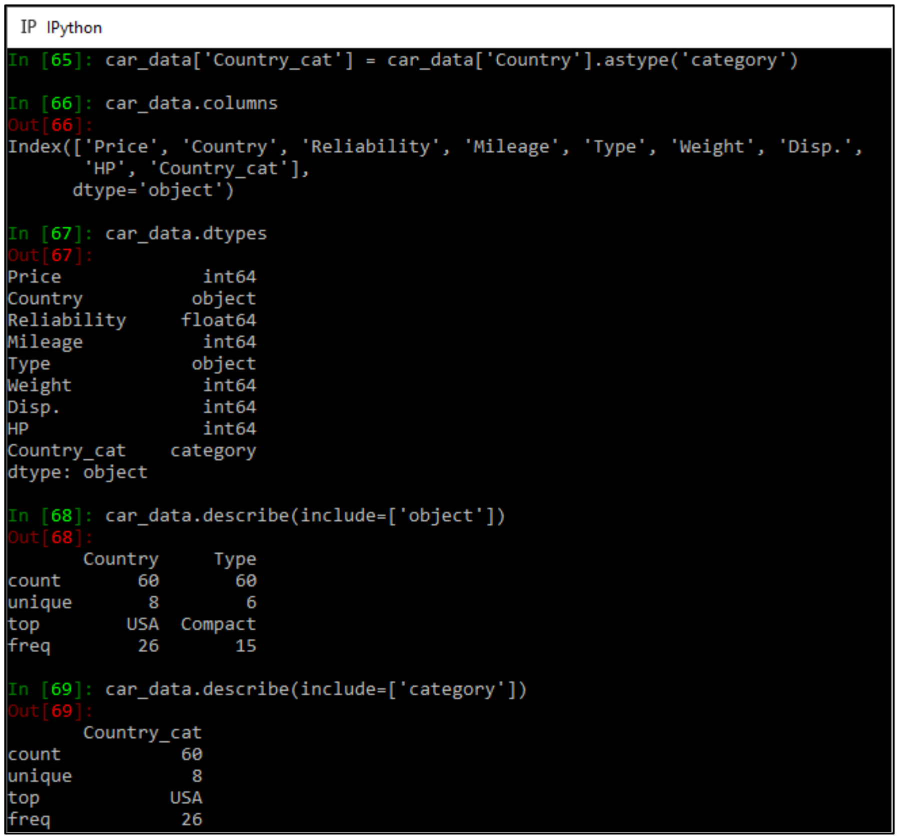


```
IP IPython
In [23]: car_data.describe(include=['object'])
Out[23]:
       Country     Type
count       60       60
unique       8        6
top        USA  Compact
freq        26       15

In [24]: car_data.describe(include=['number'])
C:\Users\bryan\Anaconda3\lib\site-packages\numpy\lib\function_base.py:383
  RuntimeWarning)
Out[24]:
              Price  Reliability    Mileage       Weight       Disp.  \
count     60.000000    49.000000  60.000000    60.000000   60.000000
mean   12615.666667     3.387755  24.583333  2900.833333  152.050000
std     4082.935753     1.455111   4.791559   495.866103   54.160911
min     5866.000000     1.000000  18.000000  1845.000000   73.000000
25%     9932.500000          NaN  21.000000  2571.250000  113.750000
50%    12215.500000          NaN  23.000000  2885.000000  144.500000
75%    14932.750000          NaN  27.000000  3231.250000  180.000000
max    24760.000000     5.000000  37.000000  3855.000000  305.000000

               HP
count   60.000000
mean   122.350000
std     30.980489
min     63.000000
25%    101.500000
50%    111.500000
75%    142.750000
max    225.000000
```

Figure 2 Describe() for Categorical and Numeric Data

An important note should be provided here. When Pandas imports data into Python as a dataframe, it does not automatically convert columns into categorical datatypes, unlike R. To do so, you must convert the datatype of a column into categorical and then append it to your dataframe. The code presented in the next figure shows the process of taking the *Country* column and converting it into a categorical datatype.

Below, a new column is created called *Country_cat* and added inline to the car_data dataframe. Looking at the listed columns on line 66, the newly created column *Country_cat* is listed at the end. Notice that the datatype of *Country* is still object, while the datatype of *Country_cat* is category. Using the describe() function, both object and categorical datatypes list different columns.



```
In [65]: car_data['Country_cat'] = car_data['Country'].astype('category')

In [66]: car_data.columns
Out[66]:
Index(['Price', 'Country', 'Reliability', 'Mileage', 'Type', 'Weight', 'Disp.',
       'HP', 'Country_cat'],
      dtype='object')

In [67]: car_data.dtypes
Out[67]:
Price             int64
Country          object
Reliability     float64
Mileage           int64
Type             object
Weight            int64
Disp.             int64
HP                int64
Country_cat    category
dtype: object

In [68]: car_data.describe(include=['object'])
Out[68]:
       Country     Type
count       60       60
unique       8        6
top        USA  Compact
freq        26       15

In [69]: car_data.describe(include=['category'])
Out[69]:
       Country_cat
count           60
unique           8
top            USA
freq            26
```

Figure 3 Converting an Object Datatype to Categorical

Many other useful functions exist for categorical data within Pandas. For example, it is possible to add additional categories, remove categories not currently used in the dataset, change the existing categories, and consolidate categories. See the online documentation for more details on how to perform these operations: https://pandas-docs.github.io/pandas-docs-travis/categorical.html.

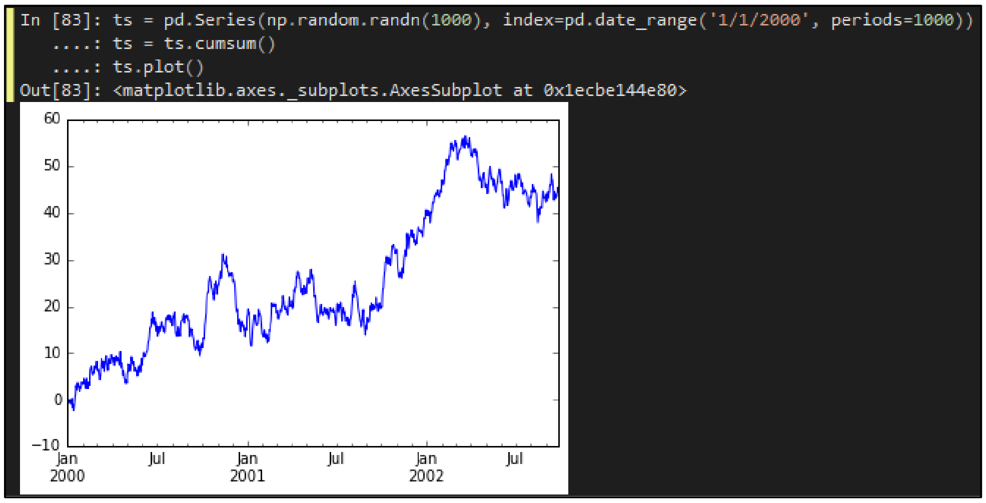
In order to determine the skewness and kurtosis within Python, separate functions must be run. Similar to R, these functions can be run separately for each column or on the entire dataframe. The following table provides a list of the most fundamental descriptive functions within the Pandas library, including those for skewness and kurtosis.

Function	Description
count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of values
mad()	Mean absolute deviation
median()	Arithmetic median of values
min()	Minimum
max()	Maximum
mode()	Mode
abs()	Absolute Value
prod()	Product of values
std()	Sample standard deviation
var()	Unbiased variance
sem()	Standard error of the mean
skew()	Sample skewness (3rd moment)
kurt()	Sample kurtosis (4th moment)
quantile()	Sample quantile (value at %)

Table 4 Additional Functions in Pandas

2. Using Plots

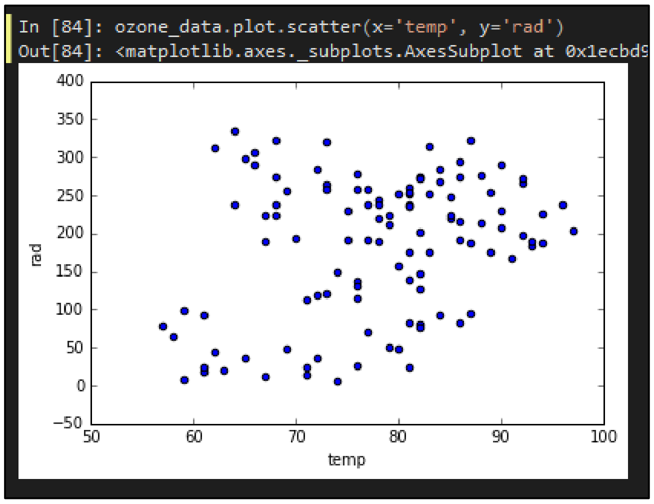
Often, numbers by themselves are not intuitive. Human brains are designed to interpret visual objects more readily than numerical data. Thus, it is important to create basic plots to assess your data in addition to looking at numbers. This includes simple scatter plots, box plots, or histograms. Below is an example of a simple plot for time-ordered data using the function plot():



```
In [83]: ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=1000))
   ....: ts = ts.cumsum()
   ....: ts.plot()
Out[83]: <matplotlib.axes._subplots.AxesSubplot at 0x1ecbe144e80>
```

Figure 5 Plot for Time-Ordered Data

The scatter plot is a popular plot used to assess data prior to modeling. The scatter plot is simple enough to use within Python. For this example, variables from the ozone dataset will be used. The variable *radiation*, or *rad*, is the *y*; the variable *temperature*, or *temp*, is the *x*. See Figure 6 for specific code. This allows you do assess the relationship between variables prior to modeling. In the plot below, a weak positive relationship exists between *radiation* and *temperature*.



```
In [84]: ozone_data.plot.scatter(x='temp', y='rad')
Out[84]: <matplotlib.axes._subplots.AxesSubplot at 0x1ecbd9
```

Figure 6 Scatter Plot of Ozone Data

Another type of plot that you should use is the boxplot. While you can use the function `plot(x, y)` to obtain the boxplot for categorical data, a different function is needed for continuous data. Using the ozone data as an example, typing in `ozone_data.boxplot()` results in a boxplot for each variable:

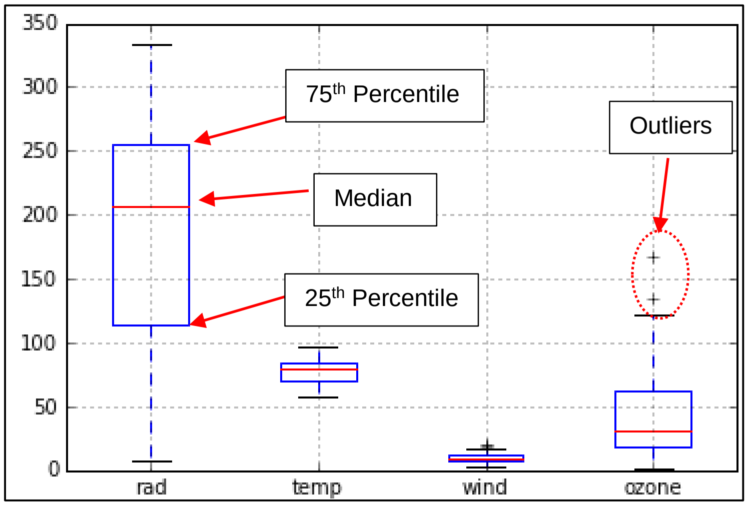
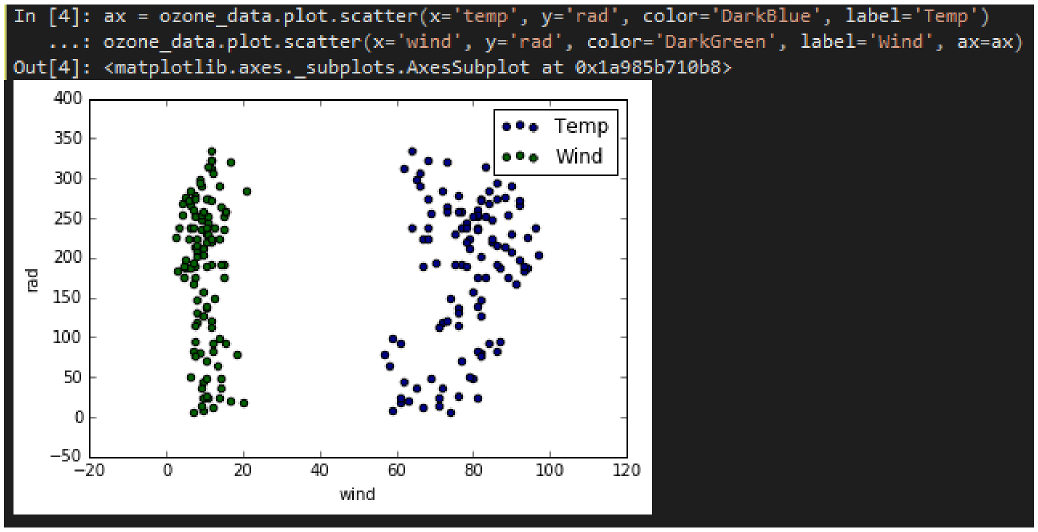


Figure 7 Box Plot for Ozone Data

The bold, horizontal line in the middle of each box is the median value for each variable. The top of the box is the 75th percentile while the bottom of the box is the 25th percentile. The dashed lines are the whiskers, leading from the 25th or 75th percentiles to the minimum or maximum data point, respectively. Sometimes, however, that horizontal line does not refer to the minimum or maximum value; they can represent 1.5 times the interquartile range of the data (approximately 2 standard deviations).

For both *wind* and *ozone* there are data points above the horizontal bar. These are considered outliers in the data. These are data points that are either greater than 1.5 times the interquartile range or lower than 1.5 times the interquartile range.

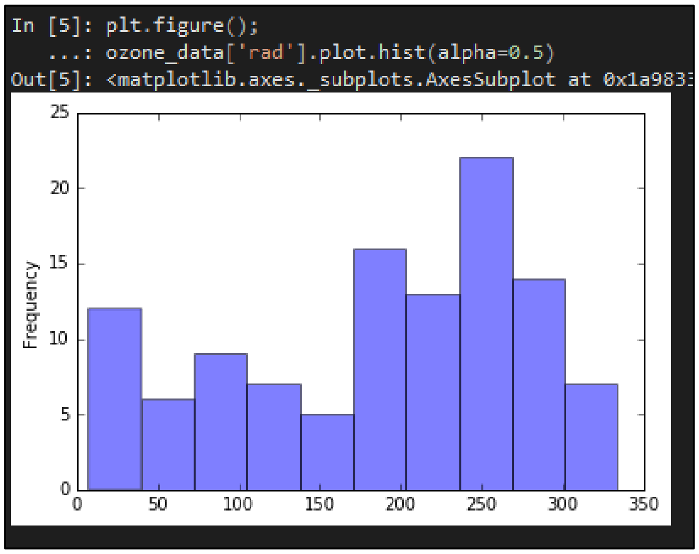
If your data set contains more than one or two variables, it may take too much time to create individual scatter plots. The below code illustrates how to plot multiple columns using a single axes:



```
In [4]: ax = ozone_data.plot.scatter(x='temp', y='rad', color='DarkBlue', label='Temp')
   ...: ozone_data.plot.scatter(x='wind', y='rad', color='DarkGreen', label='Wind', ax=ax)
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x1a985b710b8>
```

Figure 8 Multiple Columns in Scatter Plot

Unlike R, this does not provide a very clean assessment of the variables. Notice the x-axis is labeled *wind*, one of the columns of interest. The variable *temperature*, is forced onto *wind*'s scale, which doesn't provide the best assessment. The recommendation for Python is to create individual scatter plots for each column of data.



```
In [5]: plt.figure();
   ...: ozone_data['rad'].plot.hist(alpha=0.5)
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x1a983
```

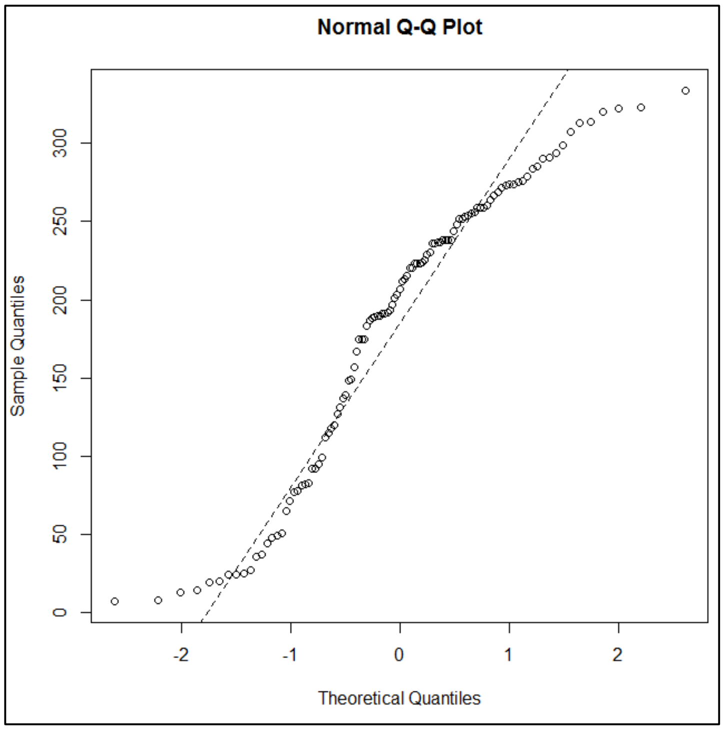
Figure 9 Histogram of Radiation

Histograms are excellent for viewing the spread of your data. A histogram provides the frequency in which certain data points appear. This also lets you eyeball the skewness and kurtosis of your data. The histogram on the previous page uses the variable *rad*.

3. Assessing Normality

Testing for normality is an important first step of familiarizing yourself with your data. In many ways, this will help determine what kinds of analysis you should perform. If non-normality is an issue, you may need to use a non-parametric test. Some statistical techniques, such as linear regression, require the data to exhibit a normal distribution.

Two simple tests exist for assessing normality: 1) Quantile-quantile plot and 2) Shapiro-Wilk test. The quantile-quantile plot, or QQ plot, is a more subjective assessment that relies on the statistician’s eye. On the next page is a QQ plot. The straight, dashed line represents a normal distribution; the circles represent the data points of your variable. Notice the slight S-shape of the data. The tail ends appear above and slightly below the left and right ends of the normal distribution, respectively.



It should be noted that this plot is for a single variable. If you have several, say 12, then you would have to create 12 separate QQ plots. To perform this function in Python for the *temperature* variable, use the following code. Note, the sts is a reference to the library scipy.stats. Prior to running the code for the QQ plot, you would run import scipy.stats as sts.

```
sts.probplot(ozone_data.temp, dist="norm", plot=plt)
```

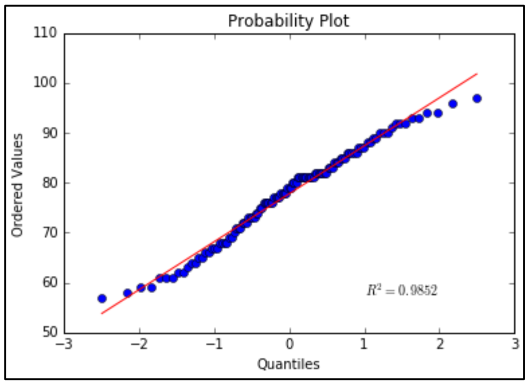
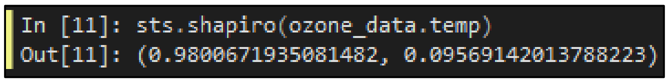


Figure 10 QQ Plot for Temperature

In addition to the QQ plot, you can use the Shapiro-Wilk test. This is considered a more objective assessment and provides a p-value. Normality results in a non-significant result of the test. Importantly, the significance depends on the alpha level you choose. I always take a more conservative approach and require an alpha of 0.05. This function also comes from the library scipy.stats.



```
In [11]: sts.shapiro(ozone_data.temp)
Out[11]: (0.9800671935081482, 0.09569142013788223)
```

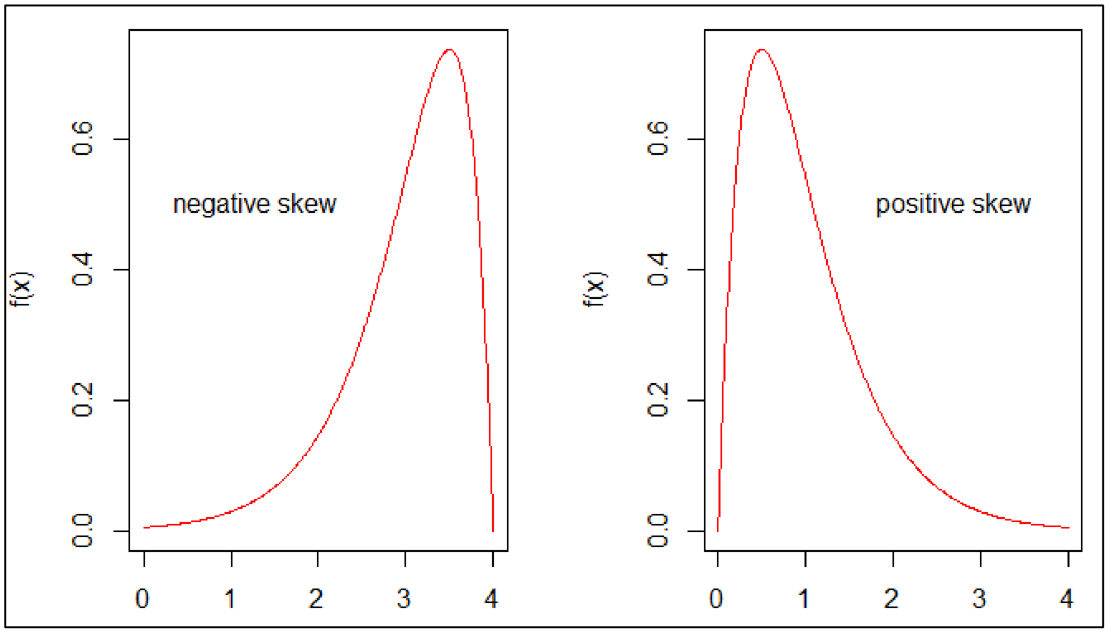
Figure 11 Shapiro-Wilk Test in Python

Within the output, the first value of 0.98 is the test statistic; the second value, 0.0956, is the p-value. Since the test here resulted in a non-significant result, the conclusion is *temperature* is normally distributed. This concurs with the above QQ plot.

4. Skewness and Kurtosis

After assessing the normality of your data, you need to understand the distribution in more detail. This requires you to look at the skewness and the kurtosis. While a histogram can help you to visually assess the skewness, it is better to use more objective criteria.

Skewness deals with the extent to which the end tails of the data are drawn out. Michael Crawley provides an excellent illustration in his book of what this may look like (page 286 1st edition, page 350 2nd edition), which I have included here. Note, that a negative skew is an indication of a skew to the left (see the tail end?) and a positive skew indicates a skew to the right. The positive and negative is in reference to the high point of that hump.

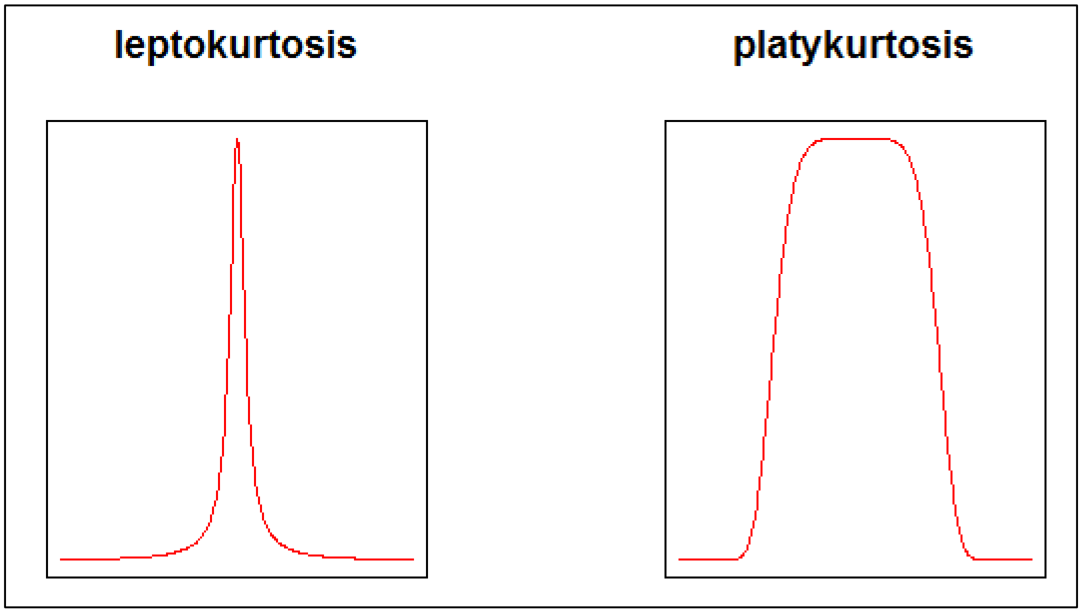


To calculate the skewness value for a single variable like *wind*, use the following function in Python:

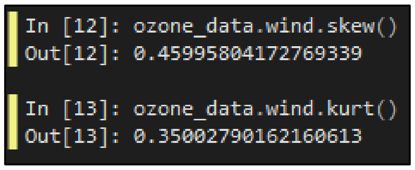
```
ozone_data.wind.skew()
```

This is a univariate test. This can result in a value that is negative (left skew), 0 (no skew), or positive (right skew). What a skew value does not indicate is how significant the skew value is. In other words, is it really worth worrying about? If it is, then a transformation, such as a square-root transformation, may be needed.

Kurtosis is another measure of normality that deals with the peak of a distribution. Two main types of kurtosis are encountered: leptokurtosis, which is very pointy, and platykurtosis, which is flat. See the images below:



Extending the example from above using the variable *wind*, simply type in the code as shown below:



```
In [12]: ozone_data.wind.skew()
Out[12]: 0.45995804172769339

In [13]: ozone_data.wind.kurt()
Out[13]: 0.35002790162160613
```

Figure 12 Skewness and Kurtosis for Wind