Workshop 13: Data Analysis in Python – Numerical and graphical summaries

Michael Kelleher

1 Introduction

Data summaries are a great first pass at identifying properties of a data set. Serious consideration is needed however, if numerical or graphical summaries are to be presented to others, not to misrepresent the shape of a data set. [1] This workshop will cover several different numerical and graphical summaries for one-dimensional data.

[1] **Questions** to ask before visualising data

1. Who is my audience?
2. What piece of the story does this data tell?
3. What is the best way to communicate that?
4. How will my audience perceive this visualisation?

2 Robustness and Resistance

Classical statistical techniques are based on strict assumptions about nature of data. These were made out of necessity for calculations to be performed by hand. These techniques can still be useful, but the required assumptions should be verified before use.

Ideally, statistical measures should be **robust** and **resistant**. **Robustness** means that a measure is not sensitive to particular assumptions about nature of data, while **resistance** means a measure is not sensitive to a small number of outliers. [2]

[2] **Note:** It is not always possible to use robust and resistant measures due to complexity of a particular data set or if direct comparison with other techniques is needed.

The **median** is a robust estimator of the centre of a data set as no assumptions need to be made about the shape of data before use. `Numpy` has a built-in method for calculating the median: `numpy.median()`. The arithmetic mean is non-robust (it assumes data are Gaussian), but is useful when this condition is met or approximated.

Given a small set of numbers: `[10, 12, 14, 16, 18]`, its sample average is 14, and its median is 14. Say the final number is transposed to 81, giving us a new set: `[10, 12, 14, 16, 81]`, its median is unchanged at 14, though its sample average changes to 26.6, heavily influenced by the outlier.

3 Numerical Summary Measures

A numerical summary measure reduces a large set of data to a single (or a few) numbers. They are a way to describe and compare data (under certain conditions). Examples include the mean, median, and variance.

3.1 *Quartiles*

QUARTILES ARE A SUMMARY MEASURE defined by sorted data. Given some set of sorted data $\{x_1, x_2, ..., x_N\}$, where $x_{(i)}$ is the i^{th} smallest value, the quartile q_p is the data point where p% of the data lies below it. These can also be called "percentiles", or "fractiles".

The 50^{th} percentile (or 0.5 quartile) is called the median, and means 50% of data fall above and below this point. If there are an odd number of points, this is trivial (e.g. $q_{0.5}$ of $[1,3,4,5,6]$ is 4). However, for an even number of points, it becomes the average of the two middle values (e.g.$q_{0.5}$ of $[1,3,4,5]$ is 3.5)

The formal definition is as follows[3]:

$$q_{(0.5)} = \begin{cases} x_{([n+1]/2)} & \text{if n odd,} \\ \dfrac{x_{(n/2)} + x_{([n/2]+1)}}{2} & \text{if n even} \end{cases}$$

Where $x_{(i)}$ is as above, the i^{th} smallest value of x, and this pattern holds for other quartiles[4].

[3] Note: the indexing here is mathematical, thus starting from 1, to transfer this to a zero-indexed programming language (e.g. Python?), simply subtract one, i.e. $x_{0.5} = x_{([[n+1]/2]-1)}$ for n odd.

[4] **Quartiles** can be computed in `numpy` using
`numpy.percentile(x, q)`
where x is an array of data, and q is the desired percentile (e.g. q=50 is the median)

3.2 *Location*

THE LOCATION OF A DATA SET describes its centre, depending on the shape of a data set, different methods may yield slightly or vastly different results. The classical measure of location is the **sample mean** [5], it is simple and easy to calculate.

The **median** has been covered above, it is resistant and robust, measuring the middle of a data set in numpy: `numpy.median(x)`.

The **trimean** [6] is not as well known, but is resistant and robust as it depends on quartiles.

The **trimmed mean** [7] is also resistant, by trimming off the extremes of data set.

[5]
$$\overline{x} = \frac{1}{N} \sum_{i=1}^{i=N} x_i$$
`numpy.mean(x)`

[6]
$$TM = \frac{q_{0.25} + 2\,q_{0.50} + q_{0.75}}{4}$$

[7]
$$\overline{x}_\alpha = \frac{1}{n-2k} \sum_{i=k+1}^{n-k} x_{(i)}$$
where α denotes the proportion of data excluded

3.3 *Spread*

THE SPREAD OF A DATA SET represents how near to the centre location the bulk of the data set lies. The classical measure of spread, analogous to the sample mean, is the **sample standard deviation** [8] which again is simple and easy to calculate.

The **inter-quartile range** [9] depends on quartiles and is resistant and robust.

Another, possibly unfamiliar, measure is the **median absolute deviation** [10] similar to standard deviation, but for median. This may be costly to compute for large data sets, but is resistant.

The **trimmed variance** [11] is analogous to the trimmed mean, and also resistant.

[8]
$$\overline{x} = \sqrt{\frac{1}{N-1} \sum_{i=1}^{i=N} (x_i - \overline{x})^2}$$
`numpy.std(x)`

[9] IQR = $q_{0.75} - q_{0.25}$
`scipy.stats.iqr(x)`

[10] MAD = median$|x_i - q_{0.5}|$

[11]
$$\overline{s}^2_\alpha = \frac{1}{n-2k} \sum_{i=k+1}^{n-k} (x_{(i)} - \overline{x}_\alpha)^2$$
where α denotes the proportion of data excluded

4 Graphical Data Summaries

4.1 Box Plots

THE BOX PLOT IS A SIMPLE GRAPHICAL TOOL for visualising data quickly. The `matplotlib.pyplot.boxplot()` routine creates a boxplot that consists of a box that spans the IQR (from $q_{0.75}$ to $q_{0.25}$), with a line marking the median, whiskers that extend to $q_{0.75} + 1.5\,IQR$ and $q_{0.25} - 1.5\,IQR$, with a $+$ symbol marking each outlier beyond this range.

The advantages of the boxplot, also called box-and-whisker plot, is that it enables side-by-side data set comparison, and has a large data density, meaning a large amount of information is contained in a small area. The disadvantage of this plot is that some information or internal variability can be disguised.

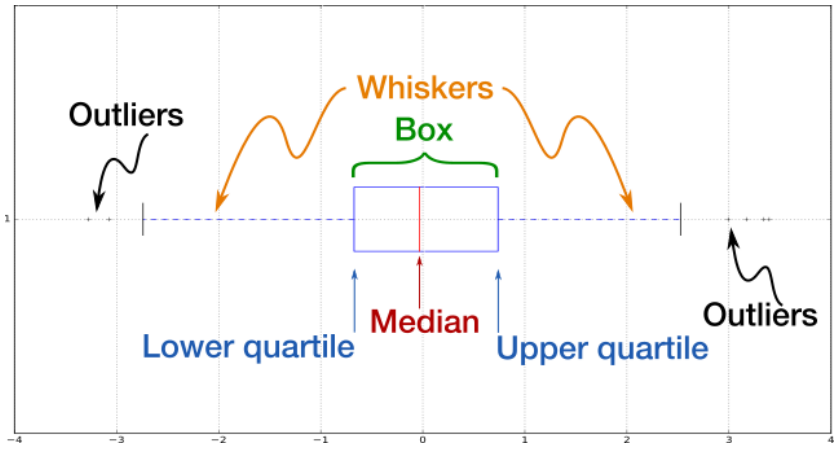

Figure 1: Box and whisker plot schematic

4.2 Histograms

A HISTOGRAM IS A GRAPHICAL REPRESENTATION of a one-dimensional data set, where the range of data is divided into intervals called *bins*. The value axis is then determined by the count of data points falling within each interval. In other words, the rectangle's first dimension is the width of the interval, and its second is the count within that interval.

Figures 2 and 3 are histograms of the same data set, however, they tell different stories. In Figure 2, the bin width is set at 1.48°C, while in Figure 3, the bin width is set using Equation 1. This equation gives the "optimal" bin width for a data set, using the inter-quartile range, IQR, the number of points n, and a constant $c \in [2.0, 2.6]$, where 2.6 is used for normally (Gaussian) distributed data, and smaller values

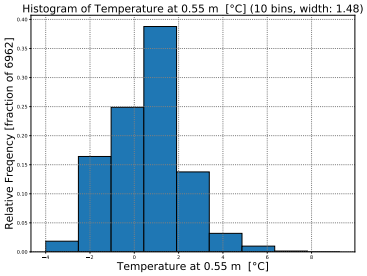

Figure 2: Histogram with 10 bins

are used for multi-modal or skewed data. In this case with $c = 2.0$, $h \approx .25$°C. This bin width calculation is only a guideline, not a hard and fast rule.

$$h \approx \frac{c\,IQR}{n^{1/3}} \tag{1}$$

The `matplotlib.pyplot.hist()` can be used to plot histograms in Python, `plt.hist(data, 10)` will plot a histogram of the array `data` into 10 bins. This second argument can also be a list or array, as: `plt.hist(data, [-2, -1, 0, 1, 2])` if direct specification of the bins is desired.

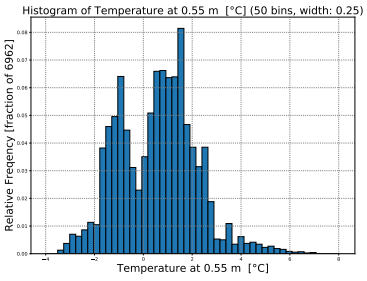

Figure 3: Histogram with 50 bins

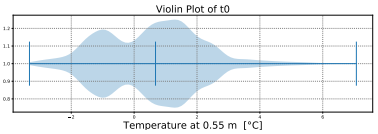

Figure 4: Violin plot of temperature data

4.3 *Violin Plot*

A COMBINATION OF THE HISTOGRAM AND BOX PLOT IS CALLED THE VIOLIN PLOT. It facilitates direct comparison of data sets, and has the ability to highlight a data set's variability. Again, it is as simple as using `matplotlib.pyplot.violinplot()`, with multiple options as well. Figure 4 shows the same data as in the histograms of figures 2 and 3. The middle line shows the median, and the outer lines show the range.

5 *Exercises*

- **Exercise 0: Load Data**:
  Begin your exercises by loading the dataset found on ELE called `Exeter_sfc_2016.csv` using pandas. This can be done using `sfc_data = pd.read_csv('Exeter_sfc_2016.csv')`. Have a look at its contents (recall how to examine the first few rows of a `DataFrame`). What columns are in the dataset, what is the frequency of the observations (how many rows are in this dataset)?

- **Exercise 1: Numerical Summary**

  1. Create two functions, one called `array_location`, and another called `array_spread`. Each should take a column from a pandas DataFrame as the input then:
     [12]
     - "`array_location`" should return a tuple of the mean, median, and trimean
     - "`array_spread`" should return a tuple of the standard deviation, IQR and MAD

     Recall: Functions must to go at the top of your program, with code that uses them below.

[12] **Hint:** Some of these measures are built in to `numpy` or `scipy`, and are indicated in the margins above, while others are not. For the measures that are not, *you should write a function for each one*.

2. Use your `array_location` and `array_spread` functions on the `Exeter_sfc_2016.csv` to summarise the daily maximum temperature and daily mean humidity (i.e. the fields `MaxT` and `MeanHumidity`).

- **Exercise 2: Plotting Data**
  Note: All plots should have titles, x-axis, and y-axis labels with units where needed.

  1. Create a function that takes a column of a pandas DataFrame and produces a histogram with ideal bin widths. Title the plot with the number of bins, and the bin width.
     Use this with the fields `MaxT` and `Precipitation` (the daily maximum temperature and daily precipitation respectively).
  2. Produce a figure with two subplots[13]: one with box plots of three temperature variables (`MaxT, MinT, MeanT`) on the same axis, and one with three relative humidity variables (`MaxHumidity, MeanHumidity, MinHumidity`) on the same axis.
  3. As in step (b), but with violin plots, be sure that the median is included.

[13] **Hint:** Explore the `subplot` function to create multiple panels on one figure