

Lab 2: Univariate Exploratory Data Analysis

M. Muldoon <mark.muldoon@manchester.ac.uk>

Week of Monday, 30 September 2024

The work for this week involves exploratory data analysis for the two datasets discussed in the lectures. Ideally, you would create a new Python notebook or R script constructed by copying the relevant bits of code from this guode. Unfortunately, it doesn't always work to cut-and-paste from a PDF and so I have provided a notebook and an R script that include the snippets mentioned below. Work through the instructions for the earthquake dataset first, to learn the commands, and then try to apply them to the spinal dataset by yourself.

Supporting materials

- The Jupyter notebook `Week2_Lab.ipynb` is a version of the main lab exercise that you can run cell-by-cell, while the R script `Week2_Lab.R` is an R version of the same thing.
- The Python notebook `earth.ipynb` gives code for the plots seen in lecture for the earthquake data.
- The R script `earth.R` also generates plots from the lectures for the earthquake data.
- The earthquake data is available as `earthquakes_US_14Jul-13Aug_2018.csv`.
- The spinal data is in `vertebral_column_data.txt`, but without column labels. These, along with other useful metadata, are in `vertebral_column_metadata.txt`.
- Other potential sources of data are:
 - UC Irvine Machine Learning Repository:
<https://archive.ics.uci.edu/ml/index.php>
 - Kaggle:
<https://www.kaggle.com/>
 - data.gov.uk – Find open data:
<https://data.gov.uk/>
 - The UK Data Service:
<https://www.ukdataservice.ac.uk/>

Univariate EDA on the earthquake data

A first look at the data

First of all, recall from the lecture that it is worth having an idea where the data comes from, how reliable it is and how much cleaning it needs.

In Python, import all the relevant libraries (see the notebooks for details). Then load the earthquake data with the following snippet:

```
df = pd.read_csv('earthquakes_US_14Jul-13Aug_2018.csv')
x = df.mag.to_numpy()
```

In R, load the libraries `dplyr` and `ggplot2` and then load the data with the snippet:

```
df <- read.table("earthquakes_US_14Jul-13Aug_2018.csv", header=TRUE, sep="," )
x <- df$mag
```

Look at the full dataset `df` and try to understand the meaning of the various columns (in theory the dataset should be accompanied by an explanation of what it contains). Can you spot any missing data?

You do not have to write anything now, but think of what you would write if you had to produce a report about these data. Also, imagine standing up in front of an audience to present an introductory analysis. What would you point out about these data? And what would you say about the methods you are going to use next to study this data? Take inspiration from the slides in the lecture.

Visualisation

No matter which software you use, think about having to explain someone else which visualisation is lossless or lossy, and why. Also, this is often overlooked, especially when you are in a hurry, but ensure all your graphs have **titles**, **labels on the axes** and if need be, **keys** or **legends** to explain plotting conventions.

In Python

Run the command

```
sns.displot(x)
```

and look at the result. In which ways does it differ from the graph produced in the lecture slides? Check the code at `earth.ipynb` to see how to customise your plot and what the different lines of code do.

Explore the way that `displot()` works: there's a helpful [tutorial](#) from Seaborn's developers. Play around with the parameters, for example changing the number of bins using `bins = 8`, or specifying the bin edges, to achieve unequal bin sizes (e.g. `bins = [0, 0.5, 2, 5]`).

One can also use `displot()` to combine a rug plot, histogram and the kernel density plot in a single figure:

```
sns.displot(x, kde=True, rug=True)
```

How would you do a jitter plot in Python? There is a related example in the notebook for Week 1's analysis of the data from Two Truths and a Lie.

Another, more flexible alternative to plot the histogram is to find the x - and y -values of bars with the command:

```
yh, xh = np.histogram(x, n, density=True)
```

where x is the data and n is the number of bins. What does the vector xh contain? How long is it? What does yh contain? How long is it? Why do you think they are not of the same length? To make them of the same length you will have to add an extra value to the yh vector using the function `np.concatenate((vector1, vector2))`. To plot only the “outline” of the histogram (useful if you want to plot two or more histograms on top of each other), use the function `plt.step()`, e.g.:

```
plt.step(xh, np.concatenate(([0.], yh)), 'b', linestyle='dashed')
```

In all these histograms, what does `density=True` do? Try computing `sum(yh)`: why is it not equal to 1? Try `density=False`: what are the values of yh now?

For more flexibility in plotting the kernel density and to be able to choose different kernels we use the following commands:

```
mykdeu = sm.nonparametric.KDEUnivariate(x)
mykdeu.fit(kernel="uni", fft=False)
xu = mykdeu.support
yu = mykdeu.density
plt.plot(xu, yu, 'b')
```

where the first line initializes a univariate kernel density estimator and the second computes the values of the fitted kernel at many points on the x -axis (the option `fft` needs to be `False`, except for in the case of the Gaussian kernel). Other kernel options are `tri` for the triangular and `gau` for the Gaussian one. Here again, helpful tutorials are available [online](#).

Finally, play with different numerical values for the option `bw`, the width of the kernel. For example, try overlaying in different colours the kernel densities obtained for values of `bw` of `[0.02, 0.05, 0.2, 0.5, 2]`. Which values do you think are ‘sensible’?

In R

Run the command

```
ggplot(df, aes(x=mag)) + geom_histogram(aes(y=after_stat(density))) + geom_density()
```

Look at the graph. In which ways does it differ from the graph produced in the lecture slides? Check the code at `earth.R` to understand how to customise your plot and what the different lines of code do.

Explore the way `ggplot` works. For example, a quick search online leads to <https://r4ds.had.co.nz/data-visualisation.html>, which I reached from the “data visualisation” link on <https://ggplot2.tidyverse.org/>. To play around, try changing the number of bins using `bins = 8`, or specifying the bin edges, to achieve unequal bin sizes (e.g. `breaks = c(0, 0.5, 2, 5)`).

One can also separately plot the jitter plot, the histogram and the kernel density with the following commands. Try and see how they work

```
n <- length( df$mag ) # Number of points
ggplot(df, aes(x=mag,y=rep(1,n),stroke=0)) + geom_jitter(width=0,height=1)
ggplot(df, aes(x=mag)) + geom_histogram(aes(y=after_stat(density)))
ggplot(df, aes(x=mag)) + geom_density()
```

How would you do a rug plot in R? You might want to search online for some tips.

To extract the x- and y-values of bars, we can use these commands:

```
n <- 50 # number of bins
h <- hist(x, breaks=n, probability=TRUE)
xh <- h$breaks
yh <- h$density
```

where `x` is the data and `n` is the desired number of bins (however, R changes the number of bins a bit to make the boundaries fall at “nice” numbers). What does the vector `xh` contain? How long is it? What does `yh` contain? How long is it? Why do you think they are not of the same length? To make them on the same length you will have to add an extra value to the `yh` vector using the concatenating function `c(vector1,vector2)`. The histogram can then be plotted as:

```
barplot(c(yh,0), names.arg = xh)
```

In these snippets of code for histograms, what do the arguments `y=after_stat(density)` and `probability=TRUE` do? Try computing `sum(yh)`: why is it not equal to 1? Try `probability=FALSE`: what are the values of `yh` now?

For more flexibility to choose different kernels and different widths, use:

```
ggplot(df, aes(x=mag)) + geom_density(kernel="rectangular", bw=0.1)
```

Other kernel options are `triangular` or `gaussian`. More details can be found online, e.g. at: https://ggplot2.tidyverse.org/reference/geom_density.html.

Finally, play with different numerical values for the option `bw`, the width of the kernel. For example, try overlaying in different colours the kernel densities obtained for values of `bw` of `[0.02,0.05,0.2,0.5,2]`. Which values do you think are ‘sensible’?

Measures of central tendency and higher moments

In Python

One can compute the mean and median with the functions `np.mean(x)` and `np.median(x)`. The most difficult measure of central tendency to compute is the mode because it is estimated from the kernel density estimate:

```
# Use a KDE to estimate the mode
mykde = sns.kdeplot(x,color='k')
xm,ym = mykde.get_lines()[0].get_data()
mo = xm[np.argmax(ym)]
```

Determine what these commands do, then plot a vertical bar for each measure against the kernel density plot.

Next, compute variance (for an unbiased estimate, use `np.var(x, ddof=1)`), as well as the standard deviation, skewness and kurtosis using Python commands. Some of the tools you need are available in `scipy.stats` (note that `moment(x, n)` computes the n -th central moment, but does standardise it). If you know enough python to do so, verify that the computations are correct repeating the them using the basic definitions of these quantities in terms of sums over the data. How would you find the median from the basic definition?

In R

One can compute the mean and median with the functions `mean(x)` and `median(x)` and, as with Python, the most difficult measure of central tendency to compute is the mode because it is estimated from the kernel density, so we first need to extract the values from the plot:

```
p <- ggplot(df, aes(x=mag)) + geom_density()
pb <- ggplot_build(p)
xd <- pb$data[[1]]$x
yd <- pb$data[[1]]$y
mode <- xd[which.max(yd)]
```

Determine what these commands do, then plot a vertical bar for each measure against the kernel density plot.

Finally, compute variance (for the biased estimate, use `moment(x, order=2, central=TRUE)`), the standard deviation, skewness and kurtosis using R commands.

Empirical Cumulative Distribution Function and Survival Function

In Python

Ensure you have imported the ECDF function with:

```
from statsmodels.distributions.empirical_distribution import ECDF
```

Then plot the ECDF with the commands:

```
ecdf=ECDF(x)
plt.step(ecdf.x, ecdf.y)
```

How would you plot the survival function? To explore the tail of the distribution use the option `plt.yscale('log')`.

Finally, find some of the most indicative percentiles, and the interquartile range, and plot them on top of the KDE.

In R

To plot the ECDF, use:

```
ggplot(df, aes(x=mag)) + geom_step(stat="ecdf")
```

To create the survival function, we first need to extract the values from the plot above:

```
e <- ggplot(df, aes(x=mag)) + geom_step(stat="ecdf")
eb <- ggplot_build(e)
xe <- eb$data[[1]]$x
ye <- eb$data[[1]]$y
```

and finally we create a new dataframe out of these data and use the `scale_y_log10()` command to change the y-axis scale to log:

```
ggplot(data=data.frame(x=x, y=1-ye), aes(x=x, y=y)) +  
  geom_step() + scale_y_log10()
```

Finally, find some of the most commonly-reported percentiles, and the interquartile range, and plot them on top of the KDE:

```
q <- quantile(x, probs=c(.25, .5, .75, .95, .99))  
ggplot(df, aes(x=mag)) + geom_step(stat="ecdf") + theme_bw() +  
  labs(x="Magnitude", y="ECDF") + geom_vline(xintercept=q, linetype="dashed")
```

Univariate EDA on the spinal data

Now that you have most of the code ready, repeat a similar analysis with the spinal data (or any other data you prefer from the links above). Focus in particular on the presence of multiple modes. Notice how the number of modes might change depending on the kernel you use and its width.