

Lab 1: Examining Your Data – A Picture is Worth a Thousand Words

[Software needed: web access]

It is often useful to plot your data in order to be able to identify outliers, to explore differences between datasets, and to make publication-quality figures. In this lab we'll explore four datasets.

Table 1: Tools covered in this lab (see end of lab for references)

Tool	Notes
Excel/Google Sheets	Excel or Google Sheets can provide an easy way of visualizing your data. We'd quickly run into limitations if we want to make scientific graphs, but for a quick view these tools are handy. We'll introduce some useful keyboard shortcuts to help with navigation too.
RAWGraphs	An "open-source data visualization framework"
R	A go-to statistical programming language for biologists with many packages for working with and plotting data.

Download the *Lab1_scatterplot_set_1.txt* and three similarly named datasets to your computer from the course website (typically, right click > Save link as... works for Windows computers; for Mac, hold down the Control key while clicking to see the local save option. Note that sometimes downloaded files will have an extra extension added to them and you'll need to remove this. Search for how to see filename extensions on the web for your operating system, and then rename the files so that they are called exactly *Lab1_scatterplot_set_1.txt* etc.)

Excel/Google Sheets

1. Go to Google Sheets at sheets.google.com or start up Excel. To view your data file in Excel, do File > Open and then identify the *Lab1_scatterplot_set_1.txt* file you saved on your local computer. Excel will ask you how you wish to open the file: select Delimited, then choose "tab" as the delimiter. In Google Sheets, use the File > Import option and use the defaults. See **Figure 1** for a depiction of the first dataset in Google Sheets.
2. Let's compute the average for the *x_values* and *y_values* columns. Find an empty cell and enter "=average(" and then highlight the cells for which you want to compute the average. There are a number of ways of doing this. Either type "B2:B143" or use your mouse to highlight the cells. Alternatively, click the first cell and then hold the Shift key and highlight the rest of the cells by using the Page Down or down cursor keys. See **Figure 2** on the next page for a list of useful keyboard shortcuts. Enter a closing ")" and press Enter to see the average value for the *x_values* column. Do the same for *y_values*, and then do the same for the three other datasets.

*What are the *x_values* and *y_values* averages for each of the 4 datasets?
What are the standard deviations for them? Hint: use the STDEV function!*

Lab Quiz Question

1

*Answer lab quiz questions while doing lab!

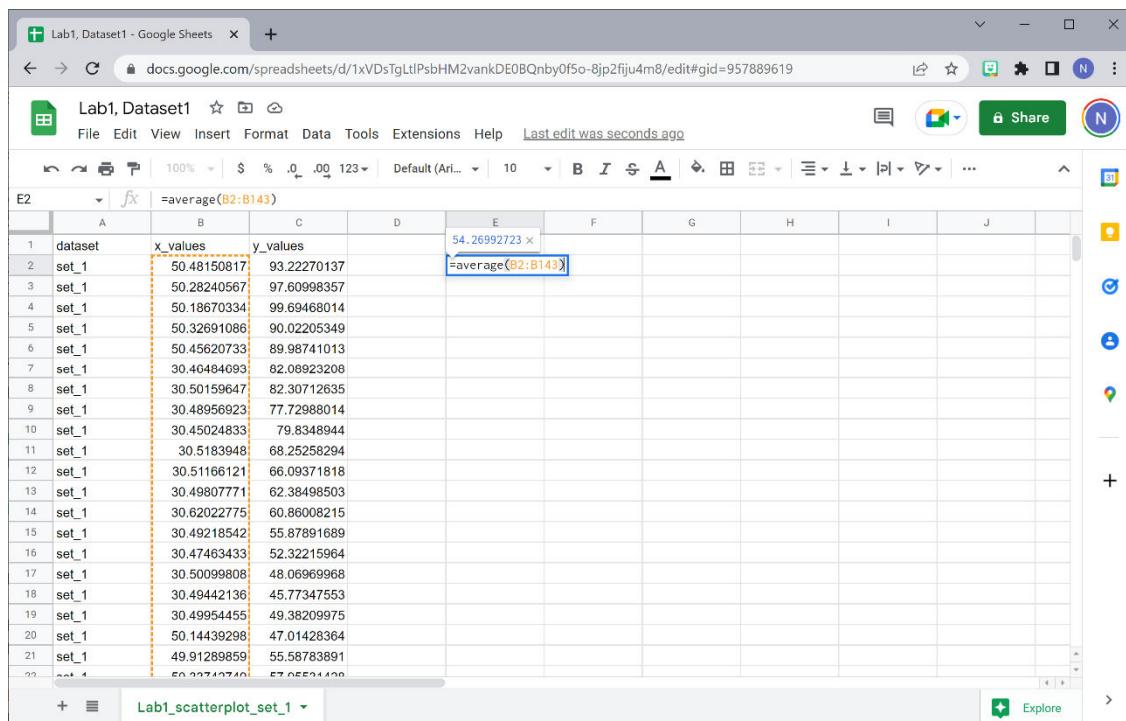




Figure 1. Examining the first dataset in Google Sheets (Excel view is similar). Compute the average for the *x_values* using the `=average()` function, specifying the cells for which to compute the average (here B2:B143).

Shift plus Page Down	select from a cell to page end
Shift then ►	add adjacent column to selection
Shift then ▼	add adjacent row to selection
Shift plus End then ► [Excel only]	select from a cell to end of row
Shift plus End then ▼ [Excel only]	select from a cell to end of column
Shift plus Space	select the whole row
Ctrl plus Space	select the whole column
Ctrl then - [Excel]	delete selected
Ctrl plus Alt then - [Sheets]	row/column/cell

Figure 2. Useful keyboard shortcuts for navigating in Excel or Google Sheets.

3. Let's plot our data. Move to cell B2 with your cursor keys. Use **Ctrl** plus **Space** to highlight the second column. Use **Ctrl** plus **▶** to highlight the third column. Then go **Insert > Chart** (there are keyboard shortcuts for this, too 😊). In Google Sheets, change the Chart type to Scatter chart . In Excel use the Scatter plot icon .

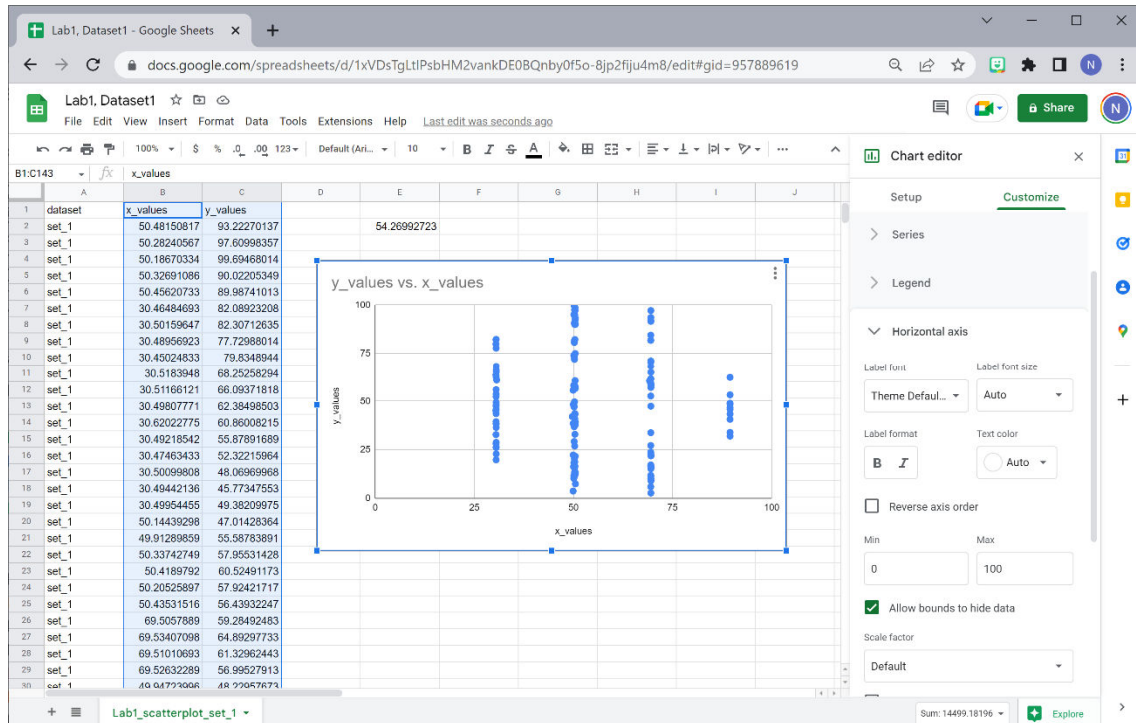


Figure 3. Scatter plot of the first dataset in Google Sheets. The Customize tab has been used to adjust the x axis minimum and maximum values so that they range from 0 to 100. The same adjustment can be carried out by clicking on the x axis values in the chart in Excel.

What does each of the scatterplots look like? You can generate these in Google Sheets or in Excel, or do this in the R section below.

As you can see, it is quite useful to visualize the data using simple charting functions in Excel or Google Sheets. Other features of these tools include conditional formatting of cells to be able to provide heatmap-like background colours to cells to be able to easily identify outliers. However, sometimes we'd like more options for charting. We'll explore a couple of other tools next.

RAWGraphs

RAWGraphs describes itself as “an open-source data visualization framework built with the goal of making the visual representation of complex data easy for everyone... providing a missing link between spreadsheet applications (e.g. Microsoft Excel) and vector graphics editors”.

1. Go to rawgraphs.io and scroll down to see examples of visualizations that are possible in the “Gallery” section.
2. Click “Use it now”, select “Upload your data”, and drag *Lab1_scatterplot_set_1.txt* into the upload section, or select the dataset by browsing to it.
3. In the Choose a chart section (scroll down the page), select “Bubble chart”. This is just like a scatter plot but you can also map data to the colour and size of dots.
4. In the Mapping section (again, scroll down), drag the *x_values* and *y_values* dimensions onto the “X Axis” and “Y Axis” chart slots, as shown in **Figure 4**.

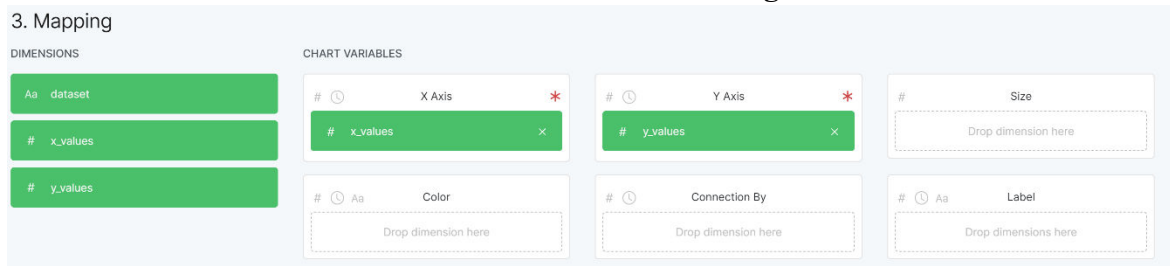


Figure 4. Assigning our *x_values* and *y_values* dimensions to chart variables. Simply drag the appropriate dimension to the desired chart variable slot.

5. In the Customize section, explore some of the options, including “Set X origin to 0” and “Show stroke”, “Max diameter”, and change the colours, as shown in **Figure 5**.

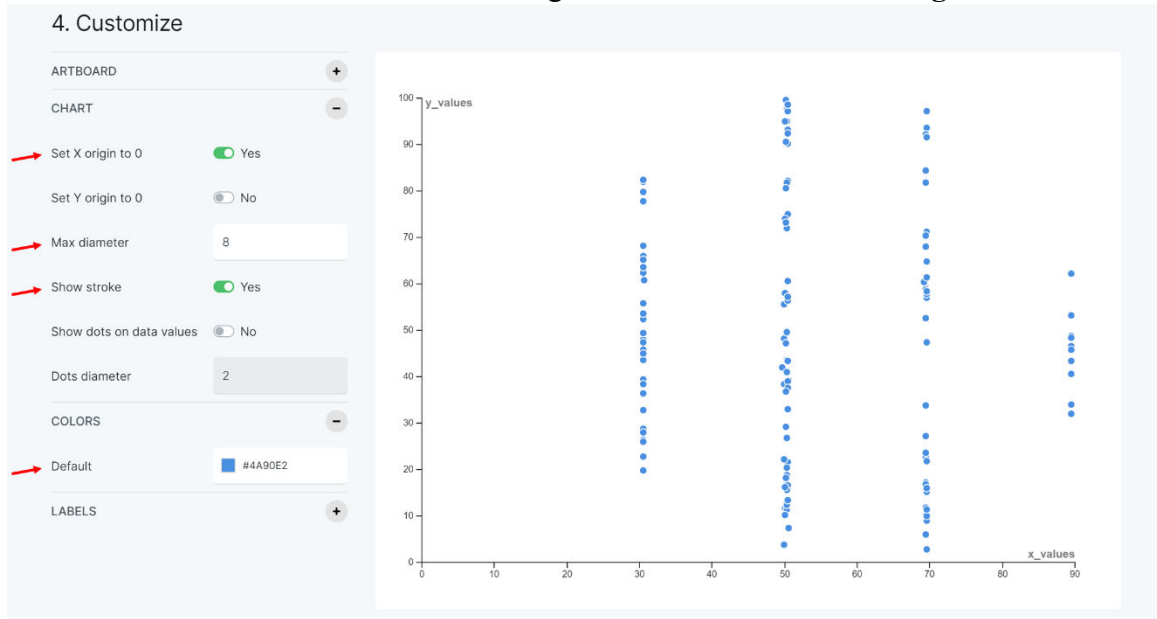


Figure 5. Changing chart options in RAWGraphs.

6. In the first data set, the points are not exactly at the same positions, so they don’t overlap, but it’s difficult to see which regions are the densest. Let’s try another chart that is useful for visualizing distributions and density.

7. Scroll back up to the Choose a chart section and select “Contour plot”. Drag the x and y dimensions onto the chart slots. View the graph by scrolling down to the plot section. Check out the customization options in the Customize section, as in **Figure 6**.

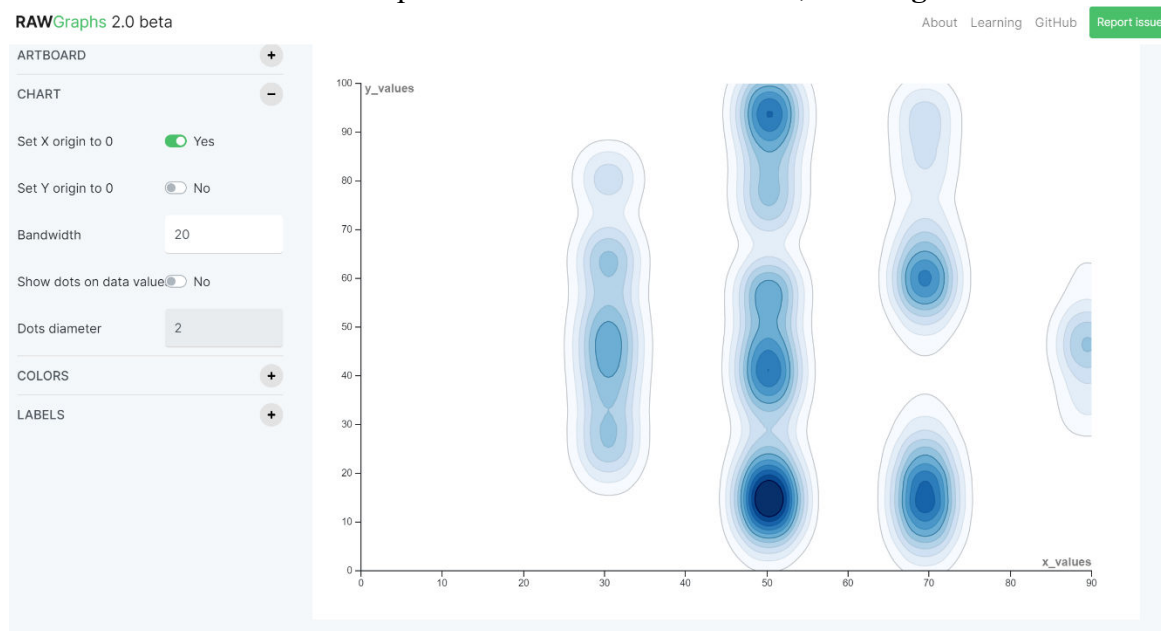


Figure 6. Generating a contour plot in RAWGraphs.

Where is the greatest number of x - y points located? Are you able to figure out how to adjust the x axis so that the maximum value is 100?

Lab Quiz
Question 2

8. For the final step, save your chart as an SVG image that you can later use in a presentation or edit with an SVG image editing program such as Figma, Adobe Illustrator, or the open-source program Inkscape, from inkscape.org. Use the Export section to do so.

R

R is a statistical programming environment and possesses many methods for creating graphs and charts. Although R is complex and can be intimidating, this exercise will serve as a superficial introduction to how we can recapitulate what we’ve done in a spreadsheet program and with RAWGraphs with a few keystrokes. Another more detailed tutorial is referenced at the end of this lab.

You can use your own installation of R as per the **Where to get it** section, or use Coursera’s built-in R Notebooks. R is similar to Matlab or SAS. It is dynamically typed so if you have experience with either MATLAB, SAS, or programming in general, then you should feel pretty comfortable fairly quickly in R. The most straightforward way to do this lab is to use an R Notebook running on Coursera’s Jupyter Hub. An R Notebook instance permits you to use a nice web interface to interact with R. Click on [Open Lab](#) to start up your own Jupyter Hub instance in the section of the

Coursera platform where you found this lab. Your work may not be saved if you close your browser but you can save your workspace at any time by typing `save.image("Lab1.RData")` at the prompt. You can download this file and upload it to continue at a later point.

To instantiate an R Notebook in Jupyter Hub, use the **New** > Notebook: R to fire one up, as shown in **Figure 7**. Or use the two pre-loaded R Notebooks. The “Lab 1 R (code).ipynb” R Notebook has all the code shown on the following pages. The “Lab 1 R (sandbox)” is a blank R Notebook where you can copy or type the code shown on the following pages. We’ve also provided the datasets (you can upload these on your own instance, see below). Another option is to use an RStudio interface.

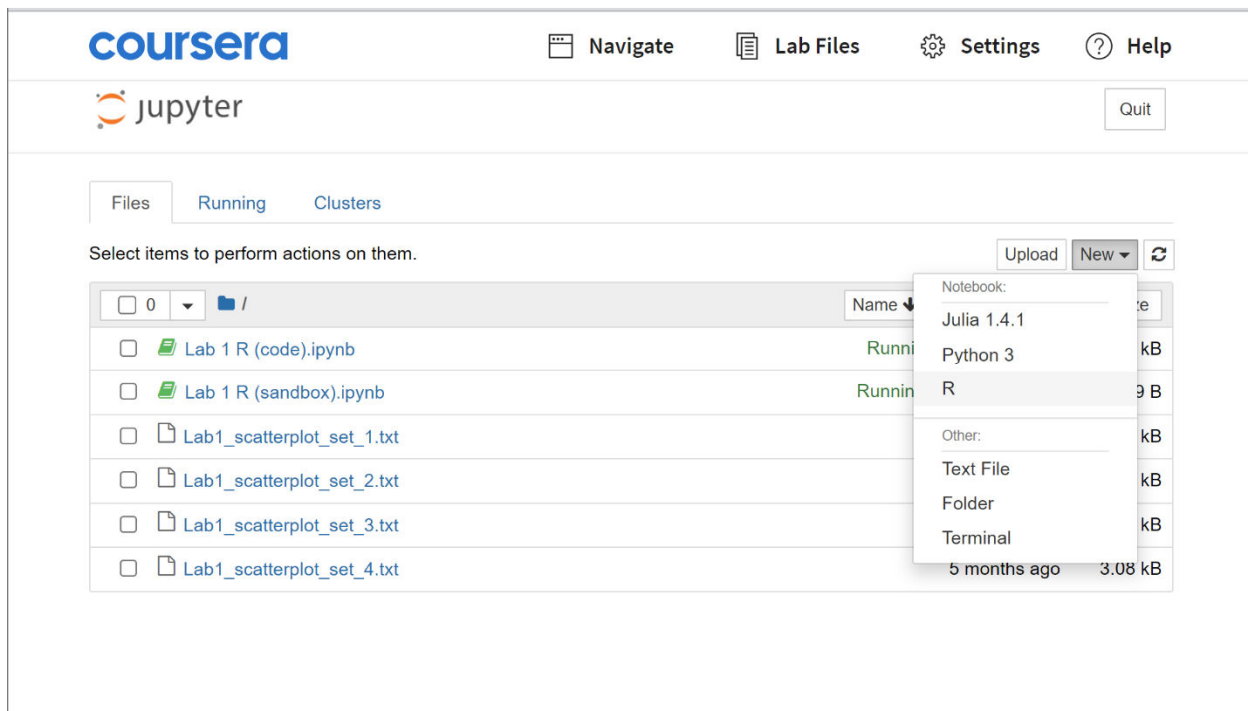


Figure 7. Creating an R Notebook in Jupyter Hub...or use the two pre-loaded .ipynb R Notebooks by clicking on them.

You’ll see an R Notebook where you can enter commands. Click the **▶** character beside a cell to run the commands in that cell. You can add new cells by clicking the **+** icon. See **Figure 8** for a depiction of a new R Notebook, renamed to “Lab 1” (click on “Untitled” to rename it), with a command entered into a code cell.

If you don’t have any programming experience, then here are three things about R:

a) Syntax

Like any programming language, the correct use of syntax (also known as language grammar) is important. This means that every character, variable name, method, parentheses, comma, and sometimes even space(s) can have importance to the code. Often our toughest enemy can be typos, as these can create a wide variety of error messages. So, keep a careful eye on how you reproduce

the following steps (you can actually just copy-paste the code if you wish)! If you want to learn more about R as a data science language, you can peruse the "R for Data Science" e-book by Hadley Wickham found here: <https://r4ds.had.co.nz/introduction.html>.

b) Variable assignment

Variable assignment is analogous to the algebraic assignment of variables, e.g. $x = 3$. However, the assignment of complex data structures to variables is possible and variable names can be of any length.

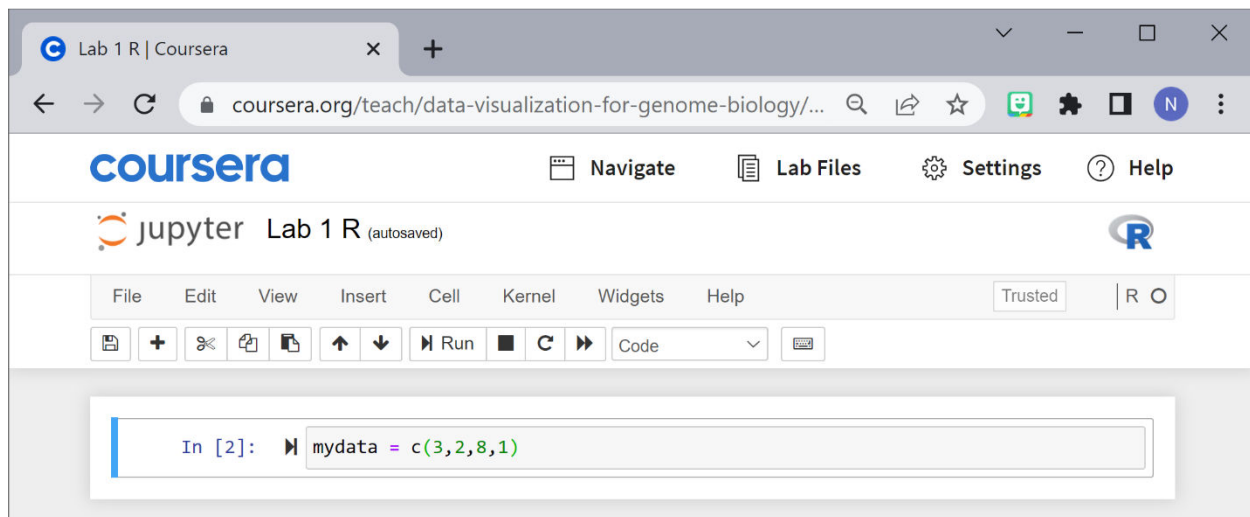


Figure 8. Typing a command into a code cell.

Try entering the following commands into the code cell, as denoted above. Or use the R Console in the left pane of RStudio (don't forget to hit Enter after each **bolded** command if using RStudio):

```
mydata = c(3,2,8,1)
```

```
mydata
```

```
3 2 8 1 #outputs from running a cell with ▶ (or by hitting Enter in RStudio)
      comments will be in regular weight font
```

Here we created a variable '**mydata**' and assigned a vector of 4 numbers to it. A vector in mathematical terms is simply a series of numbers. As you may have gathered, the typing of the variable name and hitting enter returns the variables contents. But what about the `c()`?

c) Methods

Methods are predefined programs that take data and parameters (placed within parentheses), perform some calculation or operation on those data (within the context of the provided parameters) and return a result. This result is usually assigned to a variable but doesn't have to be.


Here `c()` is the combine method. It combines comma delimited numbers into a vector of numbers and returns the vector, which we assigned to the variable `'mydata'`. Type `'?c'` within R to get a detailed description of the `c()` method.

With this background information in mind, we're all set to create some graphs in R!

1. First, we'll load some packages that will help us with our goal of creating a scatterplot. See supplemental material if you want more information on these...**tidyverse** is a widely used package in data science. Not all of these packages will be used today.


```
# Packages to help tidy our data
library (tidyverse)
# Packages for the graphical analysis section
library (repr)
library (viridis)
# packages used for working with/formatting dates in R
library (lubridate)
library (zoo)
```

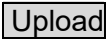

You will receive several messages back about these packages having been attached successfully, with a few warnings...you can ignore these.

2. Now let's check out what our working directory is. This is where data will be loaded from. Create a new code cell by clicking on the  icon. Then type or copy the following.

```
getwd ()
```

```
'/home/jovyan/work' #if using Coursera's Jupyter Hub
```

To get the above output you can again click the  beside the code cell but a handy shortcut to execute the code cell and to add a new code cell below that is to hold the Alt key (Option key on a Mac) then hit the Enter key.

3. We'll have to upload some data from our computer if you're using your own instance – if you haven't done so already, download the *Lab1_scatterplot_set_1.txt* and three other datasets from the course website to your computer first, and then click the  button in the top right of your Jupyter Hub to upload the files (hold “Ctrl” to select multiple files). Click the blue  buttons beside each one to finish the process of uploading.

If you're using the Coursera R Notebook instance, you will have seen that we have already provided these files, as shown in **Figure 7**.

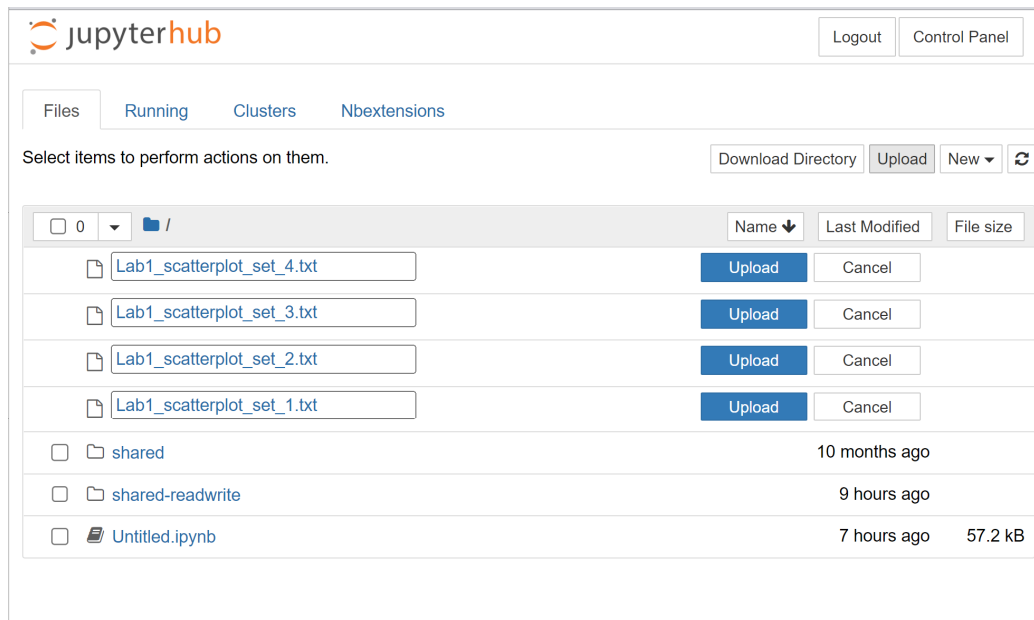


Figure 9. Uploading data to our R Notebook...only required if you're using your own R Notebook instance.

- OK, now we can create a plot! First, load your data.

```
# Initialize a plot with our data
plot_1.df <- read_tsv("Lab1_scatterplot_set_1.txt")
```

```
Rows: 142 Columns: 3
Column specification
```

```
Delimiter: "\t"
chr (1): dataset
dbl (2): x_values, y_values
...
```

- We can examine the data frame structure using the **str** command.

```
# Take a quick look at the structure of the data
str(plot_1.df)
```

```
$ dataset: chr [1:142] "set_1" "set_1" "set_1" "set_1" ...
$ x : num [1:142] 50.5 50.3 50.2 50.3 50.5 ...
$ y : num [1:142] 93.2 97.6 99.7 90 90 ...
- attr(*, "spec")=
.. cols(
.. dataset = col_character(),
.. x_values = col_double(),
.. y_values = col_double()
.. )
```

6. We will use a powerful R package called ggplot to generate a scatter plot based on the data we put into our data frame.

```
# Instantiate ggplot object
plot_1.plot <- ggplot(plot_1.df)
```

There are many different kinds of graphs available. Here's a selection:

`geom_point()` for scatter plots
`geom_line()` for line graphs
`geom_boxplot()` for boxplots
`geom_violin()` for violin plots
`geom_bar()` for bar graphs
`geom_histogram()` for histograms

7. With the commands shown below, we'll specify some parameters for our plot, and then plot it, as shown in **Figure 10**. The numbered arrows show where you can add some additional parameters in Step 8.

```
# Update the aesthetics with axis and colour information, then
# add a scatter graph!
plot_1.plot <- plot_1.plot +
aes(x = x_values, y = y_values, colour = dataset) +
1 → geom_point() +
  theme(text = element_text(size = 20)) + # set text size
  guides(colour = guide_legend(title="Set 1")) + # Legend title
  xlab("x value") + # Set the x-axis label
  ylab("y value") + # Set the y-axis label
  xlim(0,100) + # specify the minimum and maximum x axis
  values
2 → ylim(0,100) # ditto for the y axis

#display our plot
plot_1.plot
```

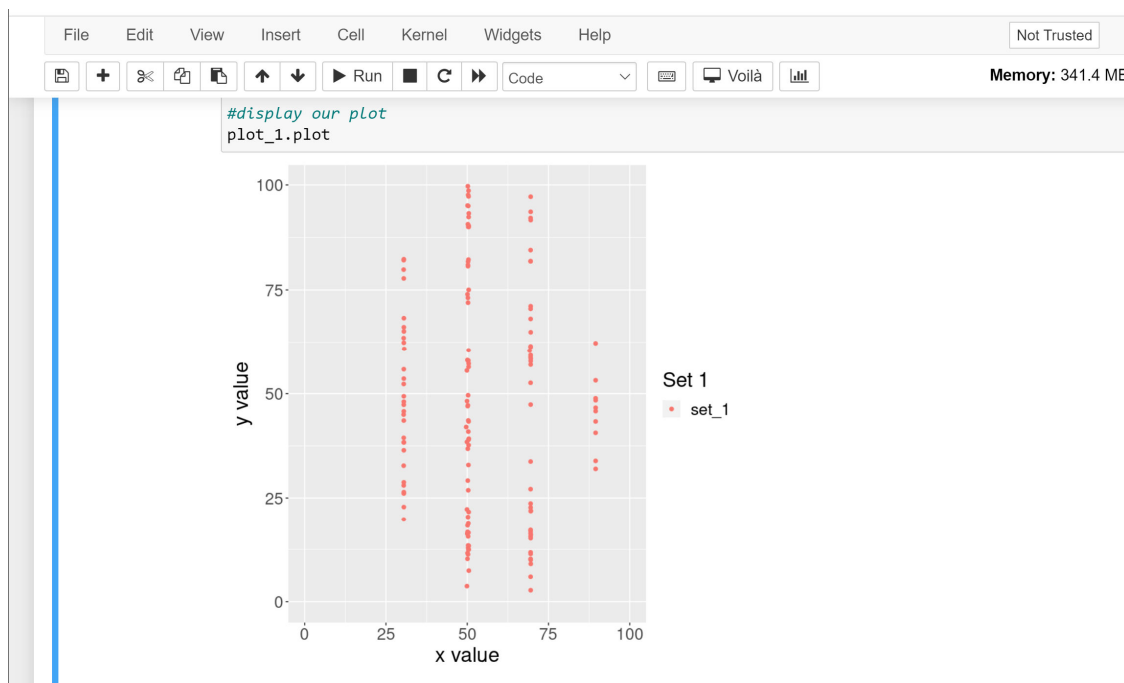


Figure 10. Scatter plot of the first dataset with ggplot, in R Notebook.

There are lots of ways to change the aesthetics of the graph. Check out this useful RStudio cheat sheet: <https://raw.githubusercontent.com/rstudio/cheatsheets/main/data-visualization.pdf>. Try choosing a different theme, shapes, or colours, as provided on page 2 of the cheat sheet...see some variants below.

8. View the resulting images for all four graphs. Simply repeat the code in Steps 4 – 7 but substitute the different file names. It would also be good to create new plot names, so `plot_1.df <- read_tsv("Lab1_scatterplot_set_1.txt")` would become `plot_2.df <- read_tsv("Lab1_scatterplot_set_2.txt")`. Make sure you change reference from `plot_1` to `plot_2` in the new code, so `plot_2.plot <- ggplot(plot_2.df)` etc. Last, update the label by changing `title="Set 1"`.

Try adding the following at the indicated locations in Step 7 for plots 3 and 4, respectively. Rerun Steps 4-6 if you're just updating the code, before rerunning Step 7.

- 1 → `theme_bw() +` # check out themes at <https://ggplot2-book.org/polishing.html>
- # use a different palette, see <https://ggplot2-book.org/scale-colour.html>
- 2 → `plot_3.plot <- plot_3.plot + scale_colour_brewer(palette = "Pastel2")` #colour_brewer palettes are colourblind-safe!
- or
- 2 → `plot_4.plot <- plot_4.plot + scale_colour_manual(values= c("blue1", "blue2", "blue3", "blue4"))`
we only use the first colour ("blue1") here, as there is only one dataset

If you haven't done so in the first part of the lab, comment on the four plots...

Lab Quiz
Question 3

9. Last, let's save our plot. You will be able to find the image in your root directory on Jupyter Hub.

```
# Save the plot we've generated to the root directory of the files.  
ggsave(plot = plot_1.plot, filename = "Set_1.png", scale=2,  
device = "png", units = c("cm"))
```

Saving 33.9 x 33.9 cm image

End of Lab!

Lab 1 Objectives

By the end of Lab 1 (comprising the lab including its boxes, and the lectures), you should:

- understand why context is important for data visualization;
- be familiar with the terminology (grammar) used for describing parts of charts;
- know common chart types, and those that are used in biology;
- be able to create simple scatter plots in Excel or Google Sheets;
- be able to appreciate the power (and limitations) of RAWGraphs for creating complex data visualizations;
- be able to use R and ggplot to graph some data.

Do not hesitate to check with the Forums for this course on Coursera if you do not understand any of the above after reading the relevant material.

Where to get it: You can run your R visualizations in an R Notebook available within Coursera's Jupyter Hub, as described in this lab. If you want to run R on your own computer, note that this lab was developed with R version 3.6.3, and with a version of dplyr (1.1.0) more recent than that available in Coursera's base instance of Tidyverse. We've also installed these packages: RColorBrewer, umap, UpSetR, and viridis. You can download older versions of R here: <https://cran.r-project.org/bin/windows/base/old/>. Then start R and use the following command to add the packages we're using: `install.packages(c('tidyverse', 'RColorBrewer', 'umap', 'UpSetR', 'viridis', 'dplyr',).`

Additional Resources

The four datasets that we used for this lab are a subset of 13 datasets generated by Matejka and Fitzmaurice in 2017 (see reference below). They all have the same statistical properties and are a follow-up to Francis Anscombe's 1973 "quartet" of four datasets that result in very different plots but almost identical statistical properties.

Justin Matejka and George Fitzmaurice. 2017. Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics through Simulated Annealing. Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. Association for Computing Machinery, New York, NY, USA, 1290-1294. DOI: <https://doi.org/10.1145/3025453.3025912>.

Francis Anscombe (1973) Graphs in Statistical Analysis. The American Statistician, 27:1, 17-21, DOI: <http://dx.doi.org/10.1080/00031305.1973.10478966>.

RStudio's ggplot cheat sheet: <https://raw.githubusercontent.com/rstudio/cheatsheets/main/data-visualization.pdf>

"R for Data Science" e-book by Hadley Wickham at <https://r4ds.had.co.nz/introduction.html>

Tips for creating better figures for biology: Rougier NP, Droettboom M, Bourne PE (2014) Ten Simple Rules for Better Figures. PLoS Comp Biol 10(9): e1003833. <https://doi.org/10.1371/journal.pcbi.1003833>

Data visualization style guide mentioned on Slide 21 of the Data Visualization Literacy lecture, compiled by Jamie Waese: [Google Docs link](#)