

Visualizing High Dimensional Data in R

MakML Lectures 2-3

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Rutgers University – New Jersey Medical School
w.evan.johnson@rutgers.edu

2024-10-31

MakML Lecutes

We will be having the following MakML lectures:

- ① Data Science, Infectious Disease Biomarkers (Mon 1100, 1200)
- ② Introduction to R, Dimension Reduction (Tues 1500)
- ③ Visualizing High Dimensional Data in R (Wed 1500)
- ④ Machine Learning: Support Vector Machines (Thu 0800)
- ⑤ Machine Learning: Random Forests (Thu 0900)
- ⑥ Machine Learning: Neural Networks/Deep Learning (Thu 1500)

Visualizing High Dimensional Data in R

This is the first lecture for our Programming, Plotting, and Machine Learning Workshop at Makerere University in October 2024. Hopefully they are helpful!

Things you should know about this course:

- Lots of diverse material and new concepts will be covered in this course
 - Programming and Machine Learning is **NOT** a spectator sport! You need to practice the skills you learn over and over again!
- Communication: if you have questions or concerns, please email me:
`w.evan.johnson@rutgers.edu`
- Materials for the course:
 - All materials for this course will be posted on the course GitHub page:
`https://github.com/wevanjohnson/2024_10_makML`

R Tutorials (GitHub and YouTube)

This first section of this lecture is an abbreviated version of Dr. Johnson's online R tutorial on GitHub:

https://github.com/wewanjohnson/2024_04_R_tutorial

R Tutorials (GitHub and YouTube)

Lecture	Topics
Lecture 1	Installing R, RStudio, and R packages
Lecture 2	Introduction to R/RStudio
Lecture 3	R basics, Part 1
Lecture 4	R basics, Part 2
Lecture 5	R basics, Part 3
Lecture 6	Programming Basics
Lecture 7	R Markdown
Lecture 8	Input/output data, Data structures
Lecture 9	The tidyverse
Lecture 10	Visualization with ggplot2, Part 1
Lecture 11	Visualization with ggplot2, Part 2
Lecture 12	Visualization with ggplot2, Part 3
Lecture 13	Creating R Packages
Lecture 14	Shiny Programming, Part 1
Lecture 15	Shiny Programming, Part 2

Important installations

You will need to install the following software tools:

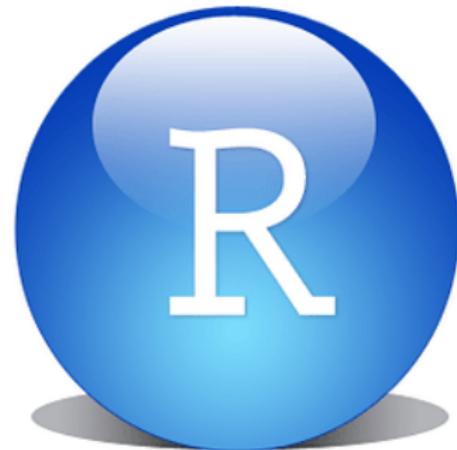
- R
- R Studio

And then the following packages in R:

```
install.packages(c("tidyverse", "umap"))
```

R and Rstudio

R is a language for statistical computing and graphics. **RStudio** is an interactive desktop environment (IDE), but it is not R, nor does it include R when you download and install it. Therefore, to use RStudio, we first need to install R.



Installing R (Windows and Mac)

You can download R from the Comprehensive R Archive Network (CRAN)¹.
Search for CRAN on your browser:

A screenshot of a Google search results page for "CRAN". The search bar at the top shows the query "CRAN". Below the search bar, there are several search results. The first result is titled "The Comprehensive R Archive Network" and includes a link to <https://cran.r-project.org/>. The second result is titled "Contributed Packages" and includes a link to [Contributed Packages. Available Packages. Currently, the CRAN ...](#). The third result is titled "CRAN and" and includes a link to [Download and Install R. Precompiled binary ...](#). The fourth result is titled "Cran - Wikipedia" and includes a link to <https://en.wikipedia.org/wiki/Cran>.

Google Search

Secure | https://www.google.com/search?rlz=1C1CHBF_enUS789US789&ei=9g20WuOsPKejwSEv5-IBA&q=CRAN&oq=CRAN&gs_l=psy-ab.3..0i131i67k1j...

CRAN

All Images Videos Books Maps More Settings Tools

About 17,700,000 results (0.50 seconds)

[The Comprehensive R Archive Network](https://cran.r-project.org/)
https://cran.r-project.org/

What are R and CRAN? R is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the R project ...

[Contributed Packages](#)
Contributed Packages. Available Packages. Currently, the CRAN ...

[CRAN and](#)
Download and Install R. Precompiled binary ...

[More results from r-project.org »](#)

[Cran - Wikipedia](#)
https://en.wikipedia.org/wiki/Cran

Cran may refer to: Calorie restriction with adequate nutrition, a dietary regimen; C-RAN, a proposed architecture for future cellular telecommunication networks; CRAN (R programming language), a package archive network for the R programming language; Cran (unit), a measurement of uncleaned laundry. Cranberry, a fruit.

Installing R (Windows and Mac)

Once on the CRAN page, select the version for your operating system: Linux, Mac OS X, or Windows. Here we show screenshots for Windows, but the process is similar for the other platforms. When they differ, we will also show screenshots for Mac OS X.

The screenshot shows a web browser displaying the CRAN website at <https://cran.r-project.org>. The main navigation bar includes links for 'CRAN', 'Mirrors', 'What's new?', 'Task Views', 'Search', 'About R', 'R Homepage', and 'The R Journal'. On the left, there is a large R logo. The central content area is titled 'The Comprehensive R Archive Network' and features a section titled 'Download and Install R'. It states: 'Precompiled binary distributions of the base system and contributed packages, Windows and Mac users most likely want one of these versions of R:' followed by a list of download links: 'Download R for Linux', 'Download R for (Mac) OS X', and 'Download R for Windows'. A mouse cursor is hovering over the 'Download R for Windows' link. Below this, a note says: 'R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.' Another section titled 'Source Code for all Platforms' notes: 'Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!' It lists several links related to source code and releases, such as 'The latest release (2018-03-15, Someone to Lean On) [R-3.4.4.tar.gz](#)', 'Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release)', and 'Daily snapshots of current patched and development versions are [available here](#)'. The bottom of the page includes footer links for 'Documentation', 'Manuals', 'FAQs', and 'Contributed' packages.

Installing RStudio (Windows and Mac)

To install RStudio, start by searching for “RStudio” on your browser:

The screenshot shows a Google search results page for "rstudio". The search bar contains "rstudio". Below it, there are several search results:

- RStudio Desktop**
Posit https://posit.co › download › rstudio-desktop ...
RStudio Desktop. Used by millions of people weekly, the RStudio integrated development environment (IDE) is a set of tools built to help you be more productive ...
R Packages · Enterprise · Cheatsheets · Resources
- Download RStudio | The Popular Open-Source IDE from ...**
Posit https://posit.co › products › open-source › rstudio ...
It includes a console, syntax-highlighting editor that supports direct code execution, and tools for plotting, history, debugging, and workspace management.
- Posit | The Open-Source Data Science Company**
The best data science is open source. Posit is committed to creating incredible open-source tools for individuals, teams, and enterprises.
RStudio Desktop · Download RStudio · RStudio IDE · RStudio Server
- Download RStudio**
Posit https://posit.co › downloads ...
RStudio IDE. The most popular coding environment for R, built with love by Posit. Used by

RStudio
Computer program

RStudio is an integrated development environment for R, a programming language for statistical computing and graphics. It is available in two formats: RStudio Desktop is a regular desktop application while RStudio Server runs on a remote server and allows accessing RStudio using a web browser. [Wikipedia](#)

Programming languages: Java, C++, JavaScript
Developer: Posit, Joseph J. Allaire
Inventor: JJ Allaire [douie](#)
Initial release: 28 February 2011; 13 years ago
License: GNU Affero General Public License v3
Operating system: Ubuntu, Fedora, Red Hat Linux, openSUSE, macOS, Windows NT
Platform: IA-32, x86-64; Qt

People also search for

W. Evan Johnson, Ph.D. Professor, Division of Visualizing High Dimensional Data in R

Installing RStudio (Mac)

You should find the Posit/RStudio website as shown above. Once there, click on “Download RStudio Desktop for Mac OS 12+” below the *2: Install RStudio* header.

The screenshot shows a Safari browser window on a Mac OS X desktop. The address bar displays 'posit.co'. The main content area is a landing page for RStudio Desktop. At the top, it says 'Grow your data science skills at posit::conf(2024)' and 'August 12th-14th in Seattle'. Below this is a navigation bar with links for 'posit', 'PRODUCTS', 'SOLUTIONS', 'LEARN & SUPPORT', 'EXPLORE MORE', and 'PRICING'. A search bar is on the right. The main heading is 'DOWNLOAD RStudio Desktop'. Subtext explains that it's used by millions weekly for R and Python development. It offers 'Posit Cloud for free' and a 'book a call with us' option. Two large numbered sections are at the bottom: '1: Install R' and '2: Install RStudio'. The '2: Install RStudio' section has a blue button labeled 'DOWNLOAD RSTUDIO DESKTOP FOR MACOS 12+'. Below it, a note states that the version is supported on macOS 12 and higher, with a link to download a previous version.

1: Install R

RStudio requires R 3.3.0+. Choose a version of R that matches your computer's operating system.

DOWNLOAD AND INSTALL R

2: Install RStudio

DOWNLOAD RSTUDIO DESKTOP FOR MACOS 12+

This version of RStudio is only supported on macOS 12 and higher. For earlier macOS environments, please [download a previous version](#).

More on R and Rstudio

See more detailed instructions at in Lecture 1 at:

https://github.com/wewanjohnson/2024_04_R_tutorial

Why R?

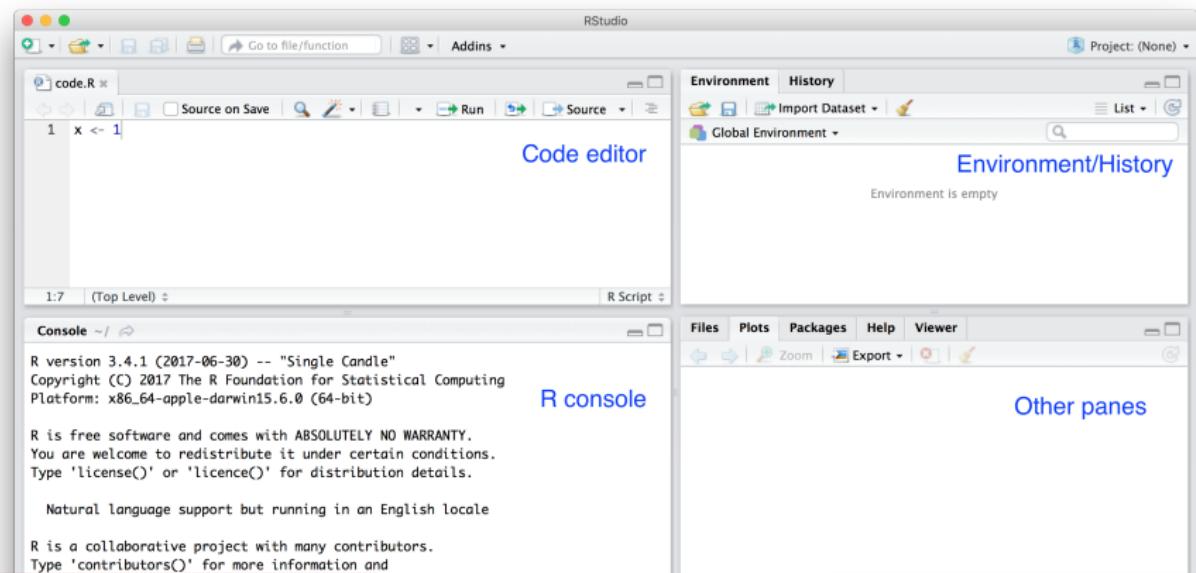
R is not a programming language for software development like C or Java. It was created by statisticians as an environment for data analysis. A history of R is summarized here: A Brief History of S.



The **interactivity** of R (more later), is an indispensable feature in data science because, as you will learn, the ability to quickly explore data is a necessity for success in this field.

RStudio

One of the great advantages of R over point-and-click analysis software is that you can save your work as scripts. You can edit and save these scripts using a text editor. We will use the interactive *Integrated Development Environment* (IDE) RStudio.



Installing R packages

The functionality provided by a fresh install of R is only a small fraction of what is possible. In fact, we refer to what you get after your first install as **base R**. The extra functionality comes from add-ons available from developers.

There are currently hundreds of these available from CRAN and many others shared via other repositories such as GitHub. However, because not everybody needs all available functionality, R instead makes different components available via **packages**.

Installing R packages

R makes it very easy to install packages from within R. For example, to install the **ggplot2** package, which we use to plot our data, you would type:

```
install.packages("ggplot2")
```

We can install more than one package at once by feeding a character vector to this function:

```
install.packages(c("tidyverse", "umap"))
```

Motivating Example

The TBnanostring.rds dataset contains gene expression measurements in the blood for 107 TB-related genes for 179 patients with either active tuberculosis infection (TB) or latent TB infection (LTBI) from one of Dr. Johnson's publications.

JOURNAL ARTICLE

Development and Validation of a Parsimonious Tuberculosis Gene Signature Using the digital NanoString nCounter Platform

Vaishnavi Kaipilyawar, Yue Zhao, Xutao Wang, Noyal M Joseph, Selby Knudsen, Senbagavalli Prakash Babu, Muthuraj Muthiah, Natasha S Hochberg, Sonali Sarkar, Charles R Horsburgh, Jr ... Show more

Author Notes



Volume 75, Issue 6
15 September 2022

Clinical Infectious Diseases, Volume 75, Issue 6, 15 September 2022, Pages 1022–1030,
<https://doi.org/10.1093/cid/ciac010>

Published: 07 January 2022 Article history ▾

 PDF  Split View  Cite  Permissions  Share ▾

Article Contents

Abstract

METHODS

RESULTS

Abstract

Background

Data Frames

A large proportion of data analysis challenges start with data stored in a data frame. For example, we stored the data for our motivating example in a data frame. You can access this dataset by loading `TBNanostring.rds` object in R:

```
TBnanostring <- readRDS("TBnanostring.rds")
```

Data Frames

In RStudio we can view the data with the `View` function:

```
View(TBnanostring)
```

You will notice that the TB status is found in the first column of the data frame, followed by the genes in the subsequent columns. The rows represent each individual patient.

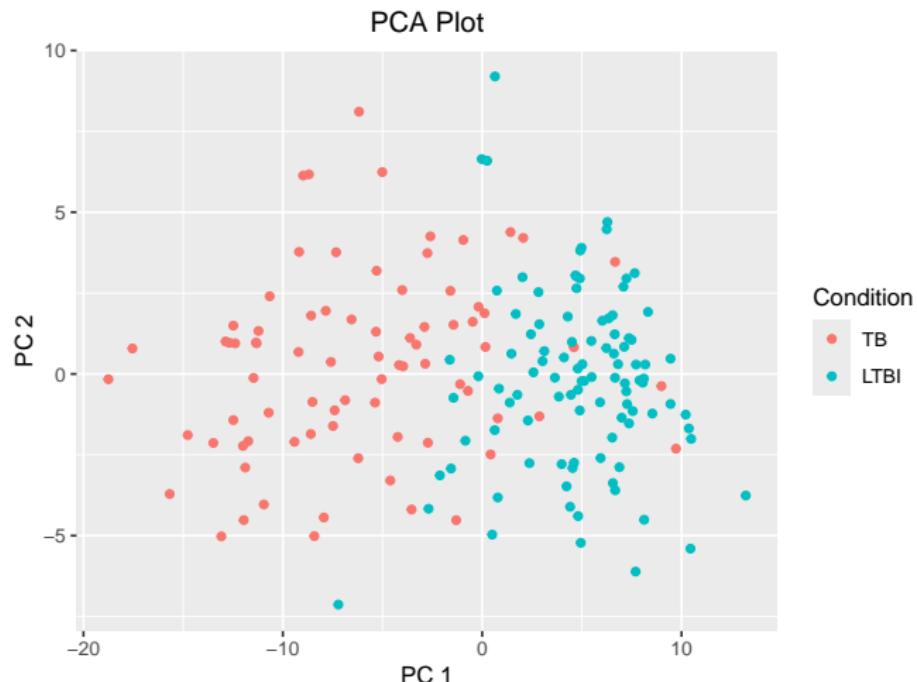
Data Visualization with R

Exploratory data visualization is perhaps the greatest strength of R. One can quickly go from idea to data to plot with a unique balance of flexibility and ease.

But first lets consider a motivating example in TB research!

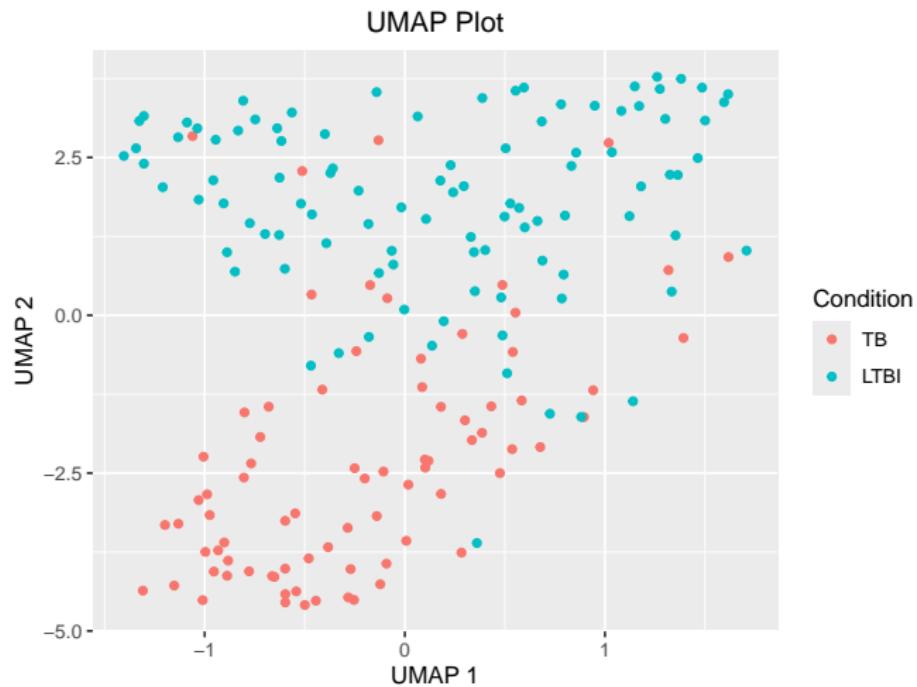
Motivating Example: PCA

Here is a PCA dimension reduction of the TB Nanostring dataset. The points are colored based on TB status.



Motivating Example: UMAP

Here is a UMAP dimension reduction of the TB Nanostring dataset.



Dimension reduction

A typical machine learning challenge will include a large number of predictors, which makes visualization somewhat challenging. We have shown methods for visualizing univariate and paired data, but plots that reveal relationships between many variables are more complicated in higher dimensions.

Dimension reduction

Here we describe powerful techniques useful for exploratory data analysis, among other things, generally referred to as **dimension reduction**.

The general idea is to reduce the dimension of the dataset while preserving important characteristics, such as the distance between features or observations.

The technique behind it all, the singular value decomposition, is also useful in other contexts. Principal component analysis (PCA) is the approach we will be showing first. Before applying PCA to high-dimensional datasets, we will motivate the ideas behind with a simple example.

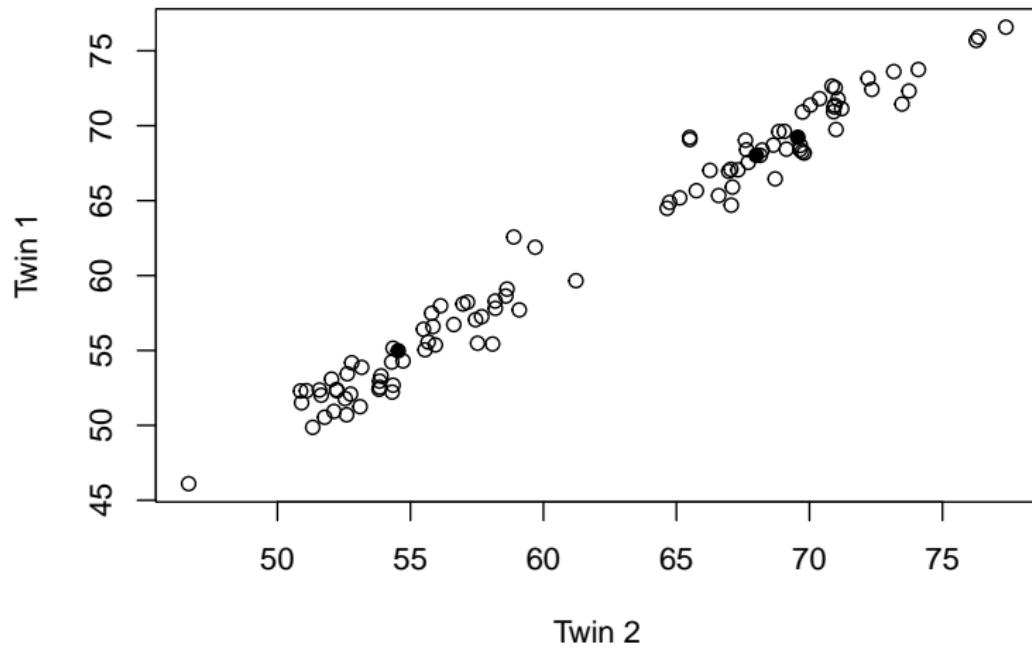
Dimension reduction: twin heights

We consider a simulated example with twin heights (children and adults):

```
set.seed(1988)
library(MASS)
n <- 100
Sigma <- matrix(c(9, 9 * 0.9, 9 * 0.92, 9 * 1), 2, 2)
x <- rbind(mvrnorm(n / 2, c(69, 69), Sigma),
           mvrnorm(n / 2, c(55, 55), Sigma))
```

Dimension reduction: twin heights

A scatterplot reveals that the correlation is high and there are two groups of twins: adults and children:



Dimension reduction: twin heights

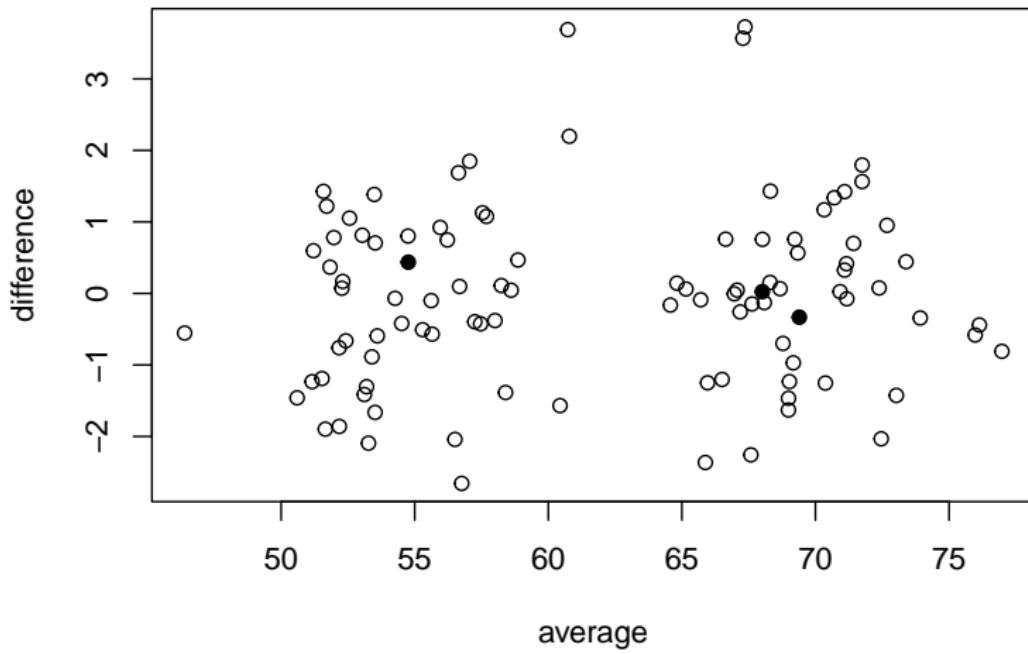
Now, can we pick a one-dimensional summary that makes this approximation even better?

If we look back at the previous scatterplot and visualize a line between any pair of points, the length of this line is the distance between the two points. These lines tend to go along the direction of the diagonal. Notice that if we instead plot the difference versus the average:

```
z <- cbind(average=(x[,2] + x[,1])/2,  
           difference=x[,2] - x[,1])
```

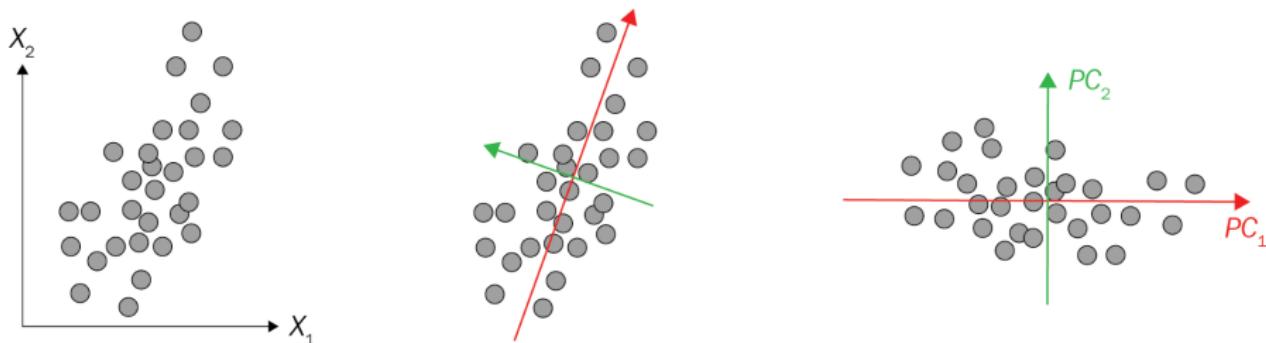
Dimension reduction: twin heights

We can see how the distance between points is mostly explained by the first dimension: the average.



Principal component analysis (PCA)

Dimension reduction can often be described as applying a transformation that *moves* or *rotates* the information, then keeping just these few informative columns, thus reducing the dimension of the vectors contained in the rows.



Data in feature space

→ Find principal components

→ Data in **principal c**omponents space

Principal component analysis

The **first principal component (PC)** of a matrix X is the linear orthogonal transformation of X that maximizes the variability. The function `prcomp` provides this info:

```
pca <- prcomp(x)
```

```
pca
```

```
## Standard deviations (1, ..., p=2):  
## [1] 11.3574818 0.8811441  
##  
## Rotation (n x k) = (2 x 2):  
## PC1 PC2  
## [1,] -0.7022354 0.7119448  
## [2,] -0.7119448 -0.7022354
```

Note that the first PC is (almost) the same as the mean we used earlier and the second is (almost) the difference!

Non-linear transformations: UMAP

Check out the following links:

- <https://pair-code.github.io/understanding-umap/>
- <https://pair-code.github.io/understanding-umap/supplement.html>

Non-linear transformations: UMAP

The intuitions behind the core principles are actually quite simple: UMAP essentially constructs a weighted graph from the high dimensional data, with edge strength representing how “close” a given point is to another, then projects this graph down to a lower dimensionality. The advanced mathematics (topology) gives UMAP a solid footing with which to handle the challenges of doing this in high dimensions with real data.

PCA on the Nanostring data

```
pca_out <- prcomp(TBnanostring[, -1])  
  
## make a dataframe with the results  
pca_reduction <- as.data.frame(pca_out$x)  
pca_reduction$Condition <- as.factor(TBnanostring$TB_Status)  
  
## Explained variation  
round(pca_out$sdev^2/sum(pca_out$sdev^2), 3)  
  
## [1] 0.468 0.073 0.058 0.037 0.033 0.024 0.021 0.018 0.016  
## [13] 0.011 0.010 0.009 0.009 0.008 0.008 0.008 0.007 0.007  
## [25] 0.006 0.006 0.005 0.005 0.005 0.004 0.004 0.004 0.004  
## [37] 0.003 0.003 0.003 0.003 0.002 0.002 0.002 0.002 0.002  
## [49] 0.002 0.002 0.002 0.002 0.001 0.001 0.001 0.001 0.001  
## [61] 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001 0.001  
## [73] 0.001 0.001 0.001 0.001 0.000 0.000 0.000 0.000 0.000  
## [85] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
```

UMAP on the Nanostring data

```
set.seed(0)
library(umap)
umap_out <- umap(TBnanostring[, -1])

## make a dataframe with the results
umap_reduction <- as.data.frame(umap_out$layout)
umap_reduction$Class <- as.factor(TBnanostring$TB_Status)

# View(umap_reduction)
```

Data Visualization with ggplot2

We will be creating plots using the ggplot² package.

```
library(tidyverse)
library(ggplot2) ## loaded with the tidyverse
```

Many other approaches are available for creating plots in R. In fact, the plotting capabilities that come with a basic installation of R are already quite powerful. There are also other packages for creating graphics such as **grid** and **lattice**.

We chose to use ggplot2 because it breaks plots into components in a way that permits beginners to create relatively complex and aesthetically pleasing plots using syntax that is intuitive and comparatively easy to remember.

²<https://ggplot2.tidyverse.org/>

Data Visualization with ggplot2

One reason ggplot2 is generally more intuitive for beginners is that it uses a **grammar of graphics**³, the *gg* in ggplot2.

This is analogous to the way learning grammar can help a beginner construct hundreds of different sentences by learning just a handful of verbs, nouns and adjectives without having to memorize each specific sentence. Similarly, by learning a handful of ggplot2 building blocks and its grammar, you will be able to create hundreds of different plots.

³<http://www.springer.com/us/book/9780387245447>

Data Visualization with ggplot2

One limitation is that ggplot2 is designed to work exclusively with data tables in tidy format (where rows are observations and columns are variables). However, most datasets that beginners work with can be converted into this format.

Data Visualization with ggplot2

To use ggplot2 you will have to learn several functions and arguments. These are hard to memorize, so we highly recommend you have the ggplot2 cheat sheet handy.

You can get a copy with an internet search for “ggplot2 cheat sheet” or by clicking here:

<https://statsandr.com/blog/files/ggplot2-cheatsheet.pdf>

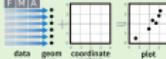
Data Visualization with ggplot2

Data Visualization with ggplot2 Cheat Sheet

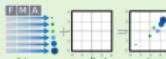


Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data set**, a set of **geoms**—visual marks that represent data points, and a **coordinate system**.



To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.



Build a graph with **plot()** or **ggplot()**

ggplot(`data`, `aes(x = cty, y = hwy, color = cyl, data = mpg, geom = "point")`)
Creates a simple plot with given data, geom, and mappings. Supplies many useful defaults.

ggplot(`data = mpg, aes(x = cty, y = hwy)`)

Begins a plot that you finish by adding layers to. No defaults, but provides more control than **plot()**.

ggplot(`data = mpg, aes(x = hwy, y = cty) + geom_point(aes(x = est.col ~ cyl) + geom_rect(bx_min = mpg$cty ~ mpg$hwy) + geom_rect(bx_min = mpg$cty ~ mpg$hwy) + scale_x_continuous("arcs") + theme_bw())`)
Add a new layer to a plot with a **geom_*** or **stat_***() function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

last_plot()
Retrieves the last plot.

Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

a <- ggplot(mpg, aes(cty, hwy))
a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(~density..), stat = "bin")
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(~density..))
x, y, alpha, color, fill, shape, size
c + geom_dotplot()
x, y, alpha, color, fill
d + geom_freqpoly()
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(~density..))
e + geom_histogram(binwidth = 5)
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(~density..))

Discrete

b <- ggplot(mpg, aes(cyl))
b + geom_bar()
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

c <- ggplot(mpg, aes(long, lat))

c + geom_polygon(aes(group = group))
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

d + geom_path(lineend = "butt",
linejoin = "round", linemtire = 1)
x, y, alpha, color, linetype, size
d + geom_rect(bxmin = "date",
bymin = "unemploy" - 900,
ymax = "unemploy" + 900)
x, y, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

e + geom_segment(aes(
xend = long + delta_long,
yend = lat + delta_lat))
x, y, end, alpha, color, linetype, size
e + geom_rect(aes(xmin = long, ymin = lat,

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))
f + geom_blank()
f + geom_jitter()
x, y, alpha, color, fill, shape, size
f + geom_point()
x, y, alpha, color, fill, shape, size
f + geom_quantile()
x, y, alpha, color, linetype, size, weight
f + geom_rug(sides = "bl")
alpha, color, linetype, size
f + geom_smooth(model = lm)
x, y, alpha, color, fill, linetype, size, weight
f + geom_text(aes(label = cty))
x, y, label, alpha, angle, color, family, fontface,
hjust, lineheight, size, vjust

Continuous Bivariate Distribution

j <- ggplot(movies, aes(year, rating))
j + geom_bin2d(binwidth = c(5, 0.5))
xmax, xmin, ymax, ymin, alpha, color, fill,
linetype, size, weight
j + geom_density2d()
x, y, alpha, colour, linetype, size
j + geom_hex()
x, y, alpha, colour, fill, size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))
j + geom_area()
x, y, alpha, color, fill, linetype, size
j + geom_line()
x, y, alpha, color, linetype, size
j + geom_step(direction = "hv")
x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4.5, se = 1.2)
k <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))
k + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, linetype,
size
k + geom_boxplot()
lower, middle, upper, x, ymax, ymin, alpha,
color, fill, linetype, shape, size, weight
k + geom_dotplot(binaxis = "y",
stackdir = "center")
x, y, alpha, color, fill
k + geom_violin(scale = "area")
x, y, alpha, color, fill, linetype, size, weight

Maps

data <- data.frame(murder = USArrests\$Murder,
state = tolower(rownames(USArrests)))
map <- tmap_mode("spatial")
l <- ggplot(data, aes(fill = murder))
l + geom_map(aes(map_id = state), map = map) +
expand_limits(x = maplon, y = maplat)
map_id, alpha, color, fill, linetype, size

Three Variables

sealsSz <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

m + geom_raster(aes(fill = z), hjust = 0.5,
vjust = -0.5, interpolate = FALSE)

Data Visualization with ggplot2

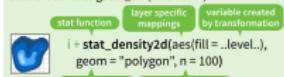
Stats - An alternative way to build a layer

Some plots visualize a **transformation** of the original data set. Use a **stat** to choose a common transformation to visualize, e.g. `a + geom_bar(stat = "bin")`



Each stat creates additional variables to map aesthetics to. These variables use a common `.name..` syntax.

stat functions and geom functions both combine a stat with a geom to make a layer, i.e. `stat_bin(geom = "bar")` does the same as `geom_bar(stat = "bin")`



`a + stat_bin(binwidth = 1, n = 10)` 1D distributions
`x, y,count., .n., .density., .density..`
`a + stat_bindot(binwidth = 1, binx = 1)`
`x, y,count., .n., .density..`
`a + stat_density(adjust = 1, kernel = "gaussian")`
`x, y,count., .density..`

`f + stat_bin2d(binx = 30, drop = TRUE)` 2D distributions
`x, y,count., .density..`
`f + stat_binnedot(binx = 30)`
`x, y,fill., .density..`
`f + stat_density2d(contour = TRUE, n = 100)`
`x, y, color, size | .level..`

`m + stat_contour(binx = 2)` 3 Variables
`x, y,count., .density..`
`m + stat_spoke(binradius = 2, angle = 2)`
`angle, radius, x, yend, y, yend.., .xend., yend.., yend..`
`m + stat_summary_hex(binx = 2, bins = 30, fun = mean)`
`x, y, z, fill || .value..`
`m + stat_summary2d(binx = 2), bins = 30, fun = mean`
`x, y, z, fill || .value..`

`g + stat_boxplot(coef = 1.5)` Comparisons
`x, y,lower., .middle., .upper., .outliers..`
`g + stat_identity(binwidth = 1, kernel = "gaussian", scale = "area")`
`x, y | .density.., .scaled., .count.,width., .width..`

`f + stat_ecdf(n = 40)` Functions
`x, y | ...`
`f + stat_quantile(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq")`
`x, y | .quantile.,y..`
`f + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95)`
`x, y |se., .ymin., .ymax..`

`ggplot(a, stat_function(fun = ...))` General Purpose
`fun = dnorm, n = 101, args = list(x0=0.5)`
`x | ...`
`f + stat_identity()`
`ggplot(a, stat_qq(examples = 1:100, distribution = qt, obsnames = list(t0:100)))`

Scales

Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale.



General Purpose scales

Use with any aesthetic:
`alpha, color, fill, linetype, shape, size`

`scale_*`(continuous) - map cont values to visual values
`scale_*`(discrete) - map discrete values to visual values

`scale_*`(identity) - use data values as visual values
`scale_*`(manual) = `c(l)` - map discrete values to manually chosen visual values

X and Y location scales

Use with x or y aesthetics (see x here)

`scale_x_date(labels = date, format = "%m/%d"),`
`breaks = date_breaks("2 weeks"))` - treat x values as dates. See `strptime` for label formats.

`scale_x_datetime()` - treat x values as date times. Use same arguments as `scale_x_date()`.

`scale_x_log10()` - Plot x on log10 scale
`scale_x_reverse()` - Reverse direction of x axis
`scale_x_sqrt()` - Plot x on square root scale

Color and fill II scales

Discrete
Continous



Shape scales

`o <- f + geom_point(aes(..., shape = ...))`
`f + scale_shape(palette = "Set1")`
`f + scale_shape_manual(values = ...)`
`Shape values shown in chart on side`

Coordinate Systems

`r + b + geom_bar()`

`r + coord_cartesian(xlim = c(0, 5), ylim, ylim)`

The default cartesian coordinate system

`r + coord_fixed(ratio = 1/2)`

Cartesian coordinates with fixed aspect ratio between x and y units

`r + coord_flip()`

Flipped Cartesian coordinates

`r + coord_polar(theta = "x", direction = 1)`

theta, start, direction

Polar coordinates

`r + coord_trans(trans = "sqrt")`

trans, trans, limx, limy

Transformed cartesian coordinates. Set extras and strains to the name of a window function.

`z + coord_map(projection = "ortho", orientation = c(-11, -74, 0))`

projection, orientation, xlim, ylim

Map projections from the mapproj package

(mercator (default), aezqualeara, lagrange, etc.)

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

`t + facet_grid(~ f)`

facet into columns based on f1

`t + facet_grid(~ year -)`

facet into rows based on year

`t + facet_grid(~ year - f1)`

facet into both rows and columns

`t + facet_wrap(~ f)`

wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

`t + facet_grid(~ x, scales = "free")`

x and y axis limits adjust to individual facets

- `*.free_x*` - x axis limits adjust

- `*.free_y*` - y axis limits adjust

Set **labeler** to adjust facet labels

`t + facet_grid(~ f1, labeler = label_both)`

`f1:0 f1:1 f1:e f1:p f1:r`

`t + facet_grid(~ f1, labeler = label_bquote(alpha ^ .(i)))`

`t + facet_grid(~ f1, labeler = label_parsed)`

`c d e p r`

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

`s <- ggplot(mpg, aes(fit, fill = drv))`

`s + geom_bar(position = "dodge")`

Arrange elements side by side

`s + geom_bar(position = "fill")`

Stack elements on top of one another, normalize height

`s + geom_bar(position = "stack")`

Stack elements on top of one another

`f + geom_point(position = "jitter")`

Add random noise to X and Y position of each element to avoid overlapping

Each position adjustment can be recast as a function with manual width and height arguments

`s + geom_bar(position = position_dodge(width = 1))`

Themes

`theme_bw()`
White background with grid lines

`theme_classic()`
White background no gridlines

`theme_grey()`
Grey background

`theme_minimal()`

Labels

`t + ggtitle("New Plot Title")`

Add a main title above the plot

`t + xlab("New X label")`

Change the label on the X axis

`t + ylab("New Y label")`

Change the label on the Y axis

`t + labs(title = "New title", x = "New x", y = "New y")`

All of the above

Legends

`t + theme(legend.position = "bottom")`

Place legend at "bottom", "top", "left", or "right"

`t + guides(color = "none")`

Set legend type for each aesthetic: colorbar, legend, or none (no legend)

`t + scale_fill_discrete(name = "Title", labels = c("A", "B", "C"))`

Set legend title and labels with a scale function,

Zooming

`Without clipping (preferred)`

`t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))`

Allows efficient zooming on a subset of data points

The Components of a Graph

The first step in learning `ggplot2` is to be able to break a graph apart into components. The main three components to note are:

- **Data:** The US murders data table is being summarized.
- **Geometry:** The plot above is a scatterplot. Other possible geometries are barplot, histogram, smooth densities, qqplot, and boxplot.
- **Aesthetic mapping:** The plot uses several visual cues to represent the information provided by the dataset. The two most important cues in this plot are the point positions on the x-axis and y-axis. Each point represents a different observation, and we *map* data about these observations to visual cues. Color is another visual cue that we map to region.

Layers of a plot

In ggplot2 we create graphs by adding **layers**. Layers can define geometries, compute summary statistics, define what scales to use, or even change styles.

To add layers, we use the symbol `+`. In general, like this:

`data %>% ggplot() + layer 1 (geom) + ... + layer N`

ggplot objects

The first step in creating a ggplot2 graph is to define a ggplot object. We do this with the function `ggplot`, which initializes the graph.

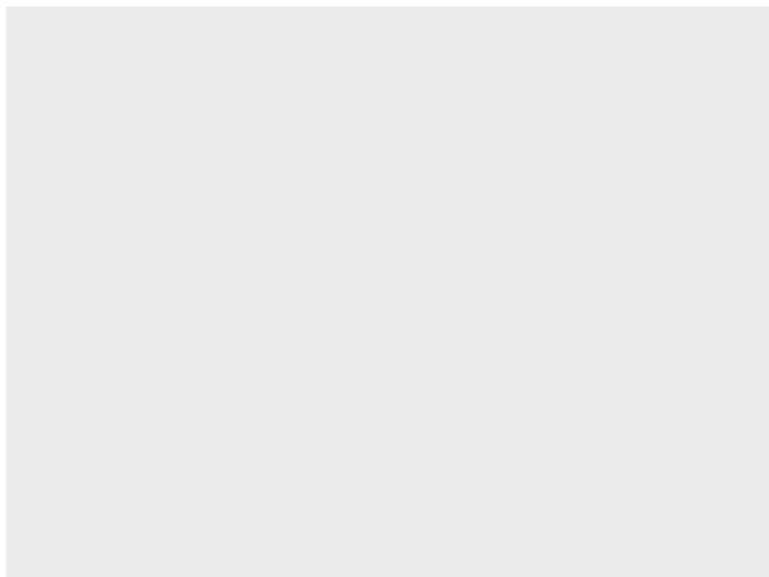
```
ggplot(data = pca_reduction)
```

We can also pipe the data in as the first argument, and save it in an object. So this line of code is equivalent to the previous one:

```
p <- pca_reduction %>% ggplot()
```

ggplot objects

It renders a plot, in this case a blank slate since no geometry has been defined. The only style choice we see is a grey background.



Geometries

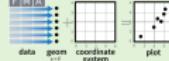
Usually, the first added layer defines the geometry. We want to make a scatterplot. What geometry do we use? Taking a quick look at the cheat sheet, we see that the function used to create plots with this geometry is `geom_point`.

Data Visualization with ggplot2 Cheat Sheet

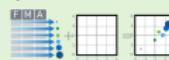


Basics

`ggplot2` is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data set**, a set of **geoms**—visual marks that represent data points, and a **coordinate system**:



To display data values, map variables in the data set to aesthetic properties of the geom like `size`, `color`, and `x` and `y` locations.



Build a graph with `plot()` or `ggplot()`

Aesthetic mappings
`plot(x = cty, y = hwy, color = cyl, data = mpg, geom = "point")`
Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

```
a <- ggplot(mpg, aes(hwy))  
a + geom_area(stat = "bin")  
x, y, alpha, color, fill, linetype, size  
b + geom_area(aes(y = density(.), stat = "bin")  
x, y, alpha, color, fill, linetype, size, weight  
b + geom_density(kernelf = "gaussian")  
x, y, alpha, color, fill, linetype, size, weight  
b + geom_density(aes(y = ..count..))  
a + geom_dotplot()  
x, y, alpha, color, fill  
a + geom_freqpoly()  
x, y, alpha, color, linetype, size  
b + geom_freqpoly(aes(y = ..density..))  
a + geom_histogram(binwidth = 5)  
x, y, alpha, color, fill, linetype, size, weight  
b + geom_histogram(aes(y = ..density..))
```

Discrete

```
b <- ggplot(mpg, aes(flf))  
b + geom_bar()  
x, alpha, color, fill, linetype, size, weight
```

Graphical Primitives

```
c <- ggplot(map, aes(long, lat))  
c + geom_polygon(aes(group = group))  
x, y, alpha, color, fill, linetype, size
```

Two Variables

Continuous X, Continuous Y

```
f + geom_blank()  
f + geom_jitter()  
x, y, alpha, color, fill, shape, size  
f + geom_point()  
x, y, alpha, color, fill, shape, size  
f + geom_quantile()  
x, y, alpha, color, linetype, size, weight  
f + geom_rug(sides = "bl")  
alpha, color, linetype, size  
f + geom_smooth(model = lm)  
x, y, alpha, color, fill, linetype, size, weight  
C f + geom_text(aes(label = cyl))  
x, y, label, alpha, color, angle, family, fontface,  
hjust, lineheight, size, vjust
```

AB

Continuous Bivariate Distribution

```
i + geom_bin2d(binwidth = c(5, 0.5))  
xmax, xmin, ymax, ymin, alpha, color, fill,  
linetype, size, weight  
i + geom_density2d()  
x, y, alpha, colour, linetype, size  
i + geom_hex()  
x, y, alpha, colour, fill, size
```

Continuous Function

```
j + ggplot(economics, aes(date, unemploy))  
j + geom_area()  
x, y, alpha, color, fill, linetype, size  
j + geom_line()  
x, y, alpha, color, linetype, size  
j + geom_step(direction = "hv")  
x, y, alpha, color, linetype, size
```

Visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
```

```
k <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))
```

```
k + geom_crossbar(fatten = 2)  
x, y, ymax, ymin, alpha, color, fill, linetype,  
size  
k + geom_errorbar()  
x, ymax, ymin, alpha, color, linetype, size,  
width (also geom_errorbarh())  
k + geom_linerange()  
x, ymin, ymax, alpha, color, linetype, size
```

Geometries

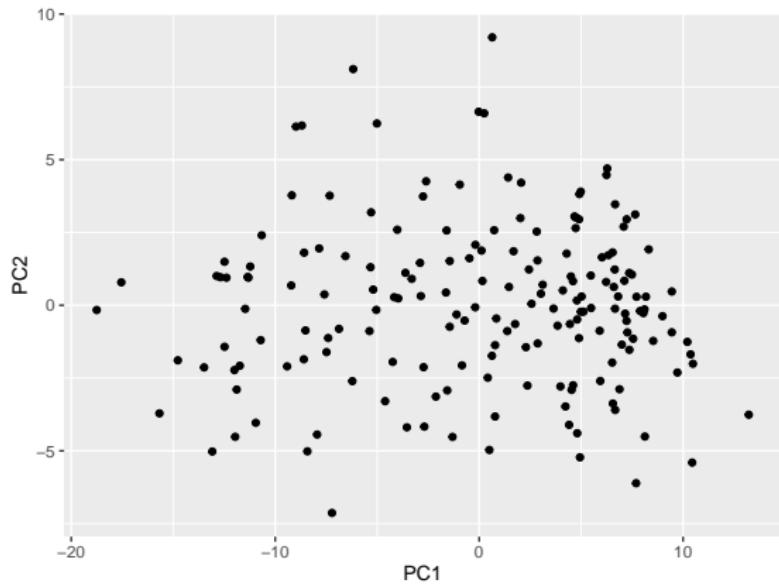
Geometry function names follow the pattern: `geom_X` where X is the name of the geometry. Some examples include `geom_point`, `geom_bar`, and `geom_histogram`.

For `geom_point` to run properly we need to provide data and a mapping (aesthetic). In this case, two aesthetic arguments are required: `x` and `y`.

Geometries: geom_point

For example, to make a scatter plot for the PCA components:

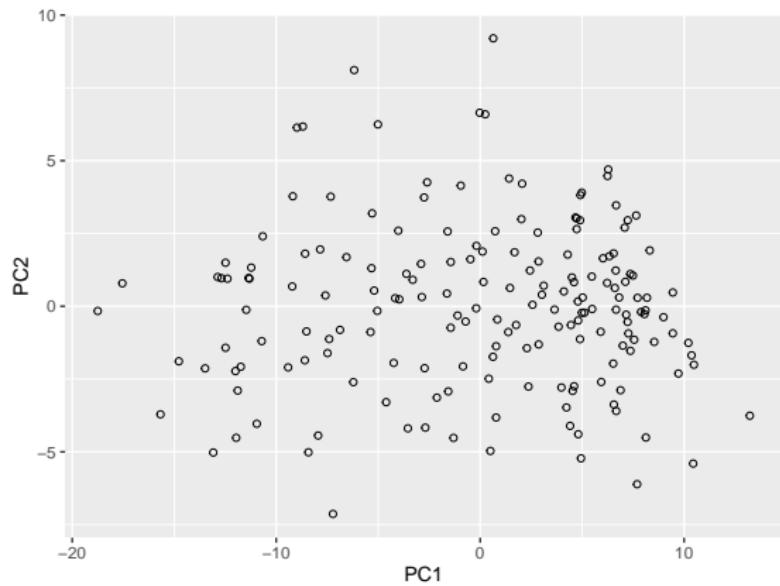
```
p + geom_point(aes(x=PC1, y=PC2))
```



Geometries: additional options

We can change the shape or color of the points

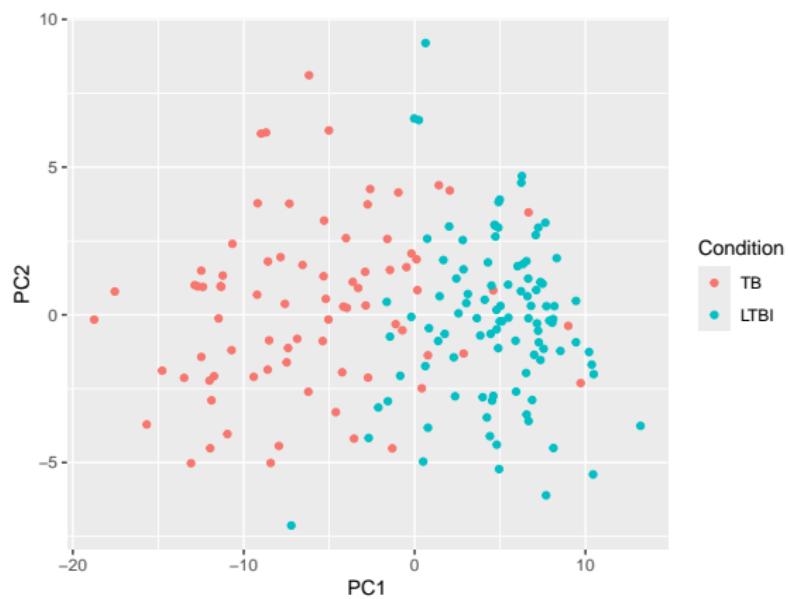
```
p + geom_point(aes(x=PC1, y=PC2), shape=1)
```



Geometries: additional aesthetics

But if we want to change color or shape based on the data, we need to add an aesthetic:

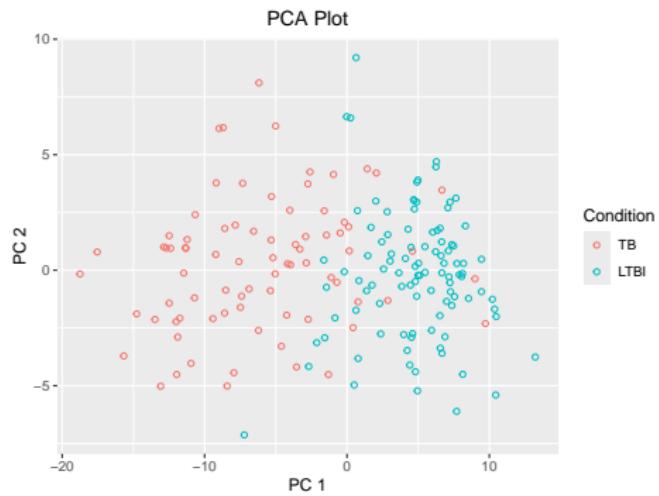
```
p + geom_point(aes(x=PC1, y=PC2, color=Condition))
```



Adding Layers

Now we can add layers that change the labels, title, etc:

```
p + geom_point(aes(x=PC1, y=PC2, color=Condition), shape=1) +
  xlab("PC 1") + ylab("PC 2") + ggtitle("PCA Plot") +
  theme(plot.title = element_text(hjust = 0.5))
```



Final results: UMAP

Here is the final code for the UMAP plot:

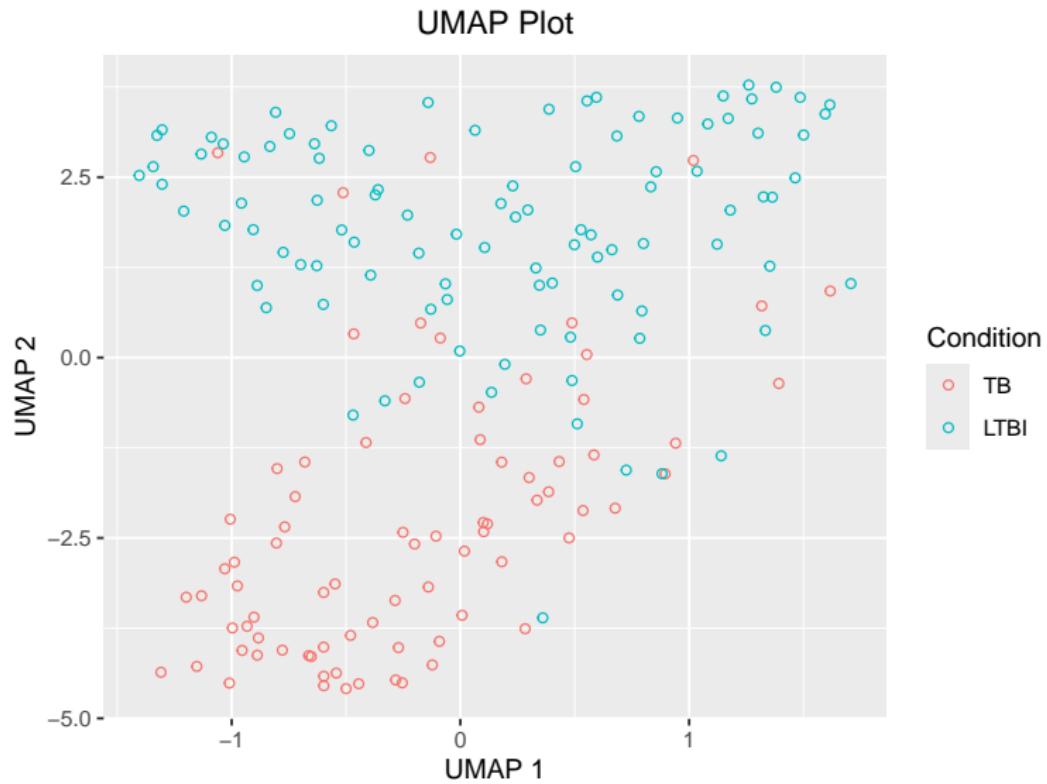
```
## read in data
TBnanostring <- readRDS("TBnanostring.rds")

## Apply UMAP reduction
set.seed(0)
library(umap)
umap_out <- umap(TBnanostring[, -1])

## Make dataframe for plotting in tidy format
umap_reduction <- as.data.frame(umap_out$layout)
umap_reduction$Condition <- as.factor(TBnanostring$TB_Status)

## Plot results with ggplot
umap_reduction %>% ggplot() +
  geom_point(aes(x=V1, y=V2, color=Condition), shape=1) +
  xlab("UMAP 1") + ylab("UMAP 2") + ggtitle("UMAP Plot") +
  theme(plot.title = element_text(hjust = 0.5))
```

Final results: UMAP



Extra: Use ChatGPT

I gave ChatGPT the following prompt:

“write R code to upload the TBnanostring.rds gene expression dataset, where the first column is the TB status, the other columns are gene expression values for each sample, apply PCA to the data, and make a ggplot of the first two principal components, colored by TB_status”

And it worked! (Almost!)

Session info

```
sessionInfo()
```

```
## R version 4.4.0 (2024-04-24)
## Platform: aarch64-apple-darwin20
## Running under: macOS Sonoma 14.2.1
##
## Matrix products: default
## BLAS:    /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
## LAPACK:  /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  LAPACK version 3
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Africa/Kampala
## tzcode source: internal
##
## attached base packages:
## [1] stats      graphics   grDevices utils      datasets   methods    base
##
## other attached packages:
## [1] MASS_7.3-61     umap_0.2.10.0   lubridate_1.9.3 forcats_1.0.0
## [5] stringr_1.5.1   dplyr_1.1.4    purrr_1.0.2    readr_2.1.5
## [9] tidyverse_2.0.0  tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
## [1] utf8_1.2.4       generics_0.1.3   stringi_1.8.4   lattice_0.22-6
## [5] hms_1.1.3        digest_0.6.37    magrittr_2.0.3   evaluate_1.0.0
## [9] grid_4.4.0       timechange_0.3.0 fastmap_1.2.0   jsonlite_1.8.9
## [13] Matrix_1.7-0     RSpectra_0.16-2  tinytex_0.53    fansi_1.0.6
## [17] scales_1.3.0     cli_3.6.3       rlang_1.1.4    munsell_0.5.1
## [21] withr_3.0.1      yaml_2.3.10    tools_4.4.0    tzdb_0.4.0
## [25] colorspace_2.1-1 reticulate_1.39.0 png_0.1-8    vctrs_0.6.5
```