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

2024-10-29

MakML Lecutures

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/wevanjohnson/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:

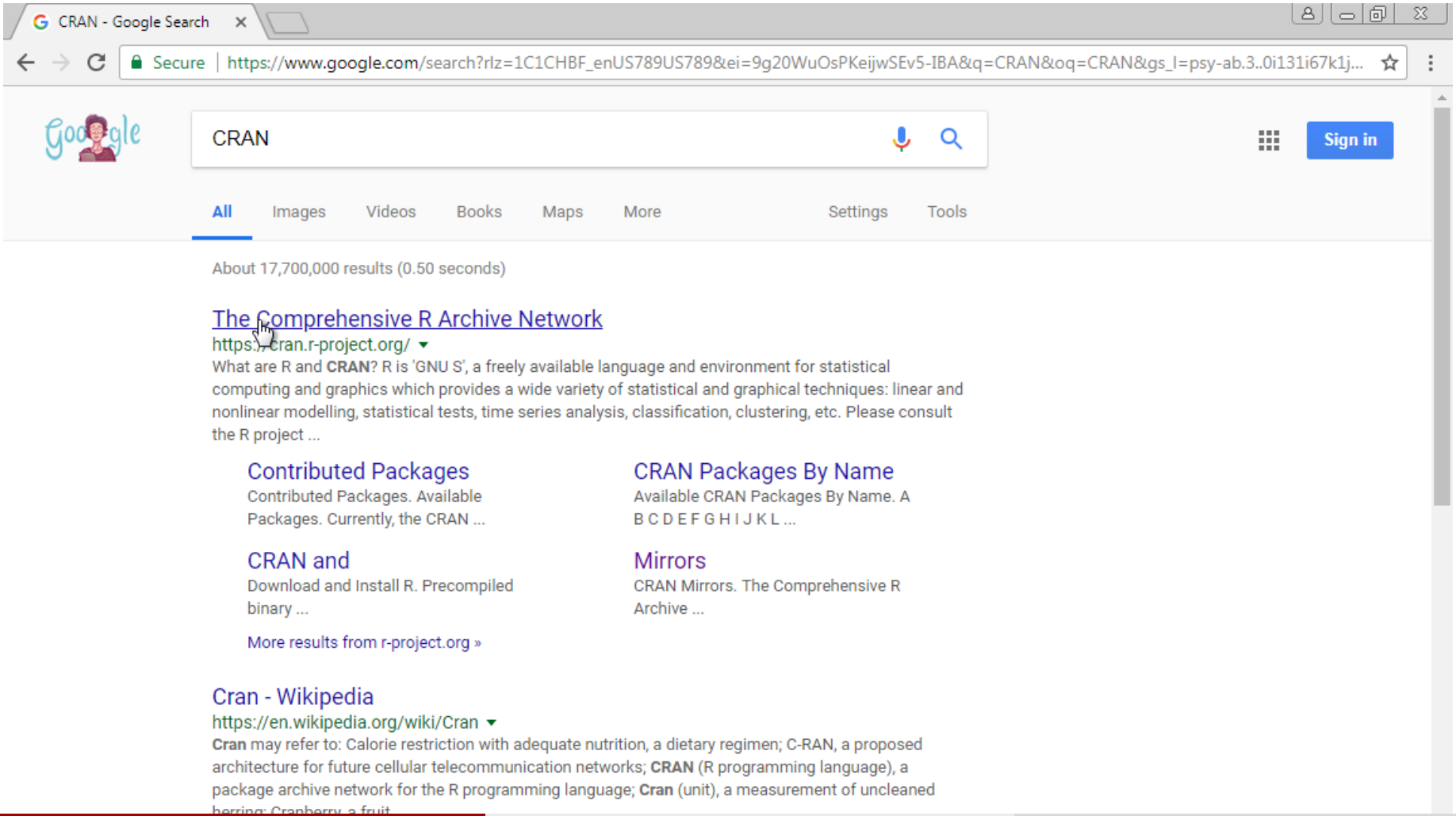
```
install.packages(c("tidyverse", "umap"))
```

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.

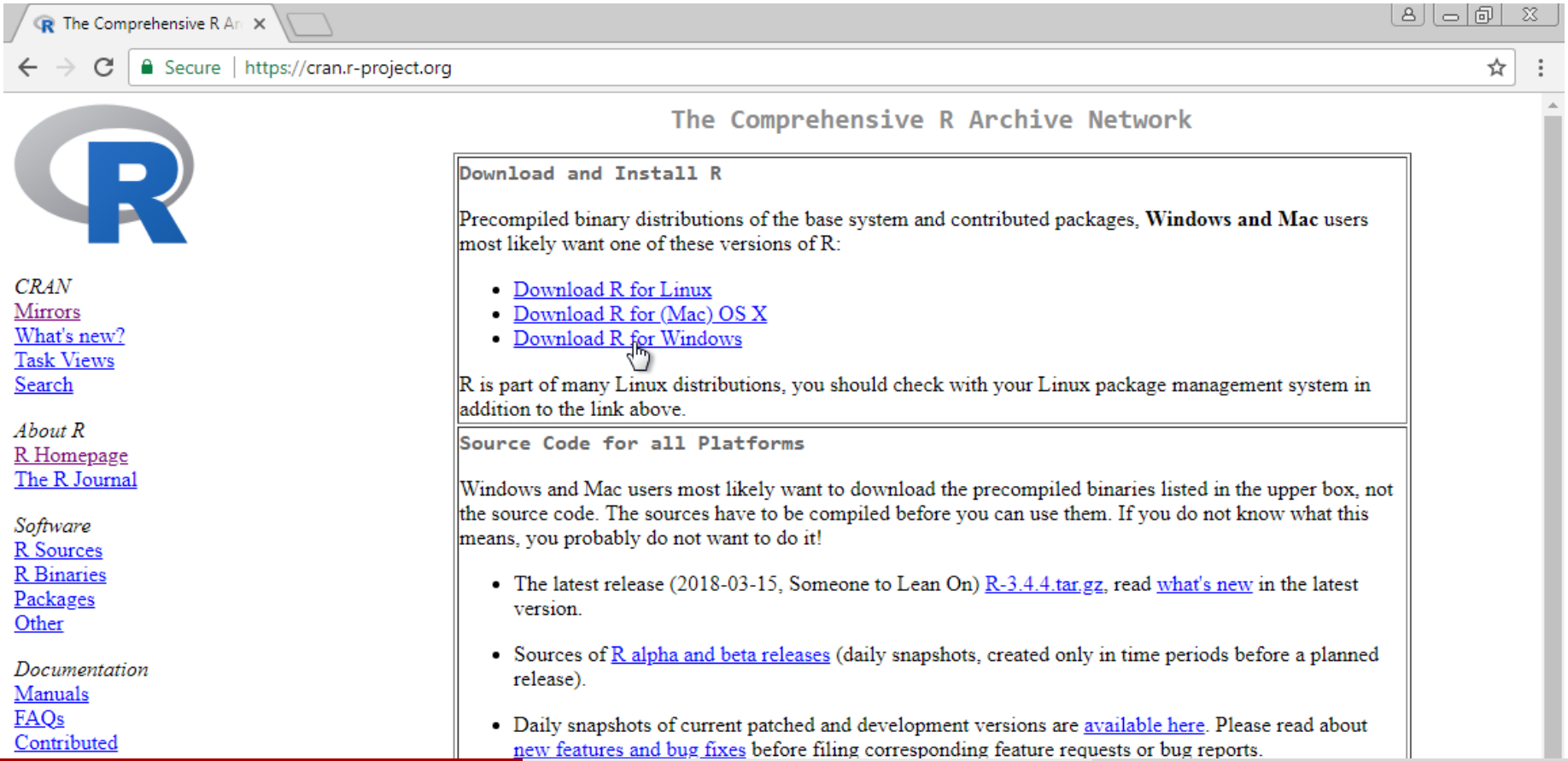
Installing R (Windows and Mac)

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



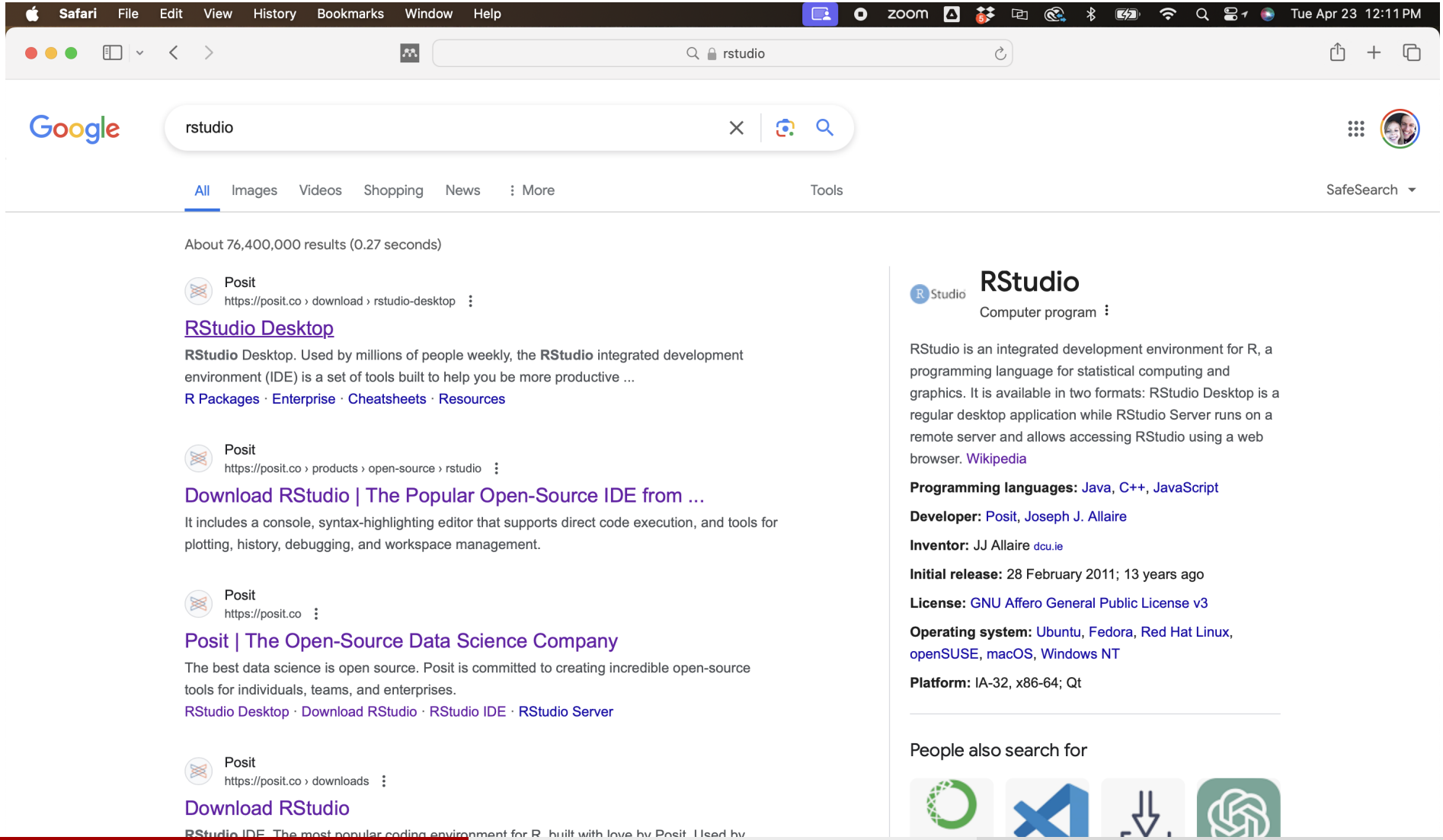
Installing R (Windows and Mac)

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.



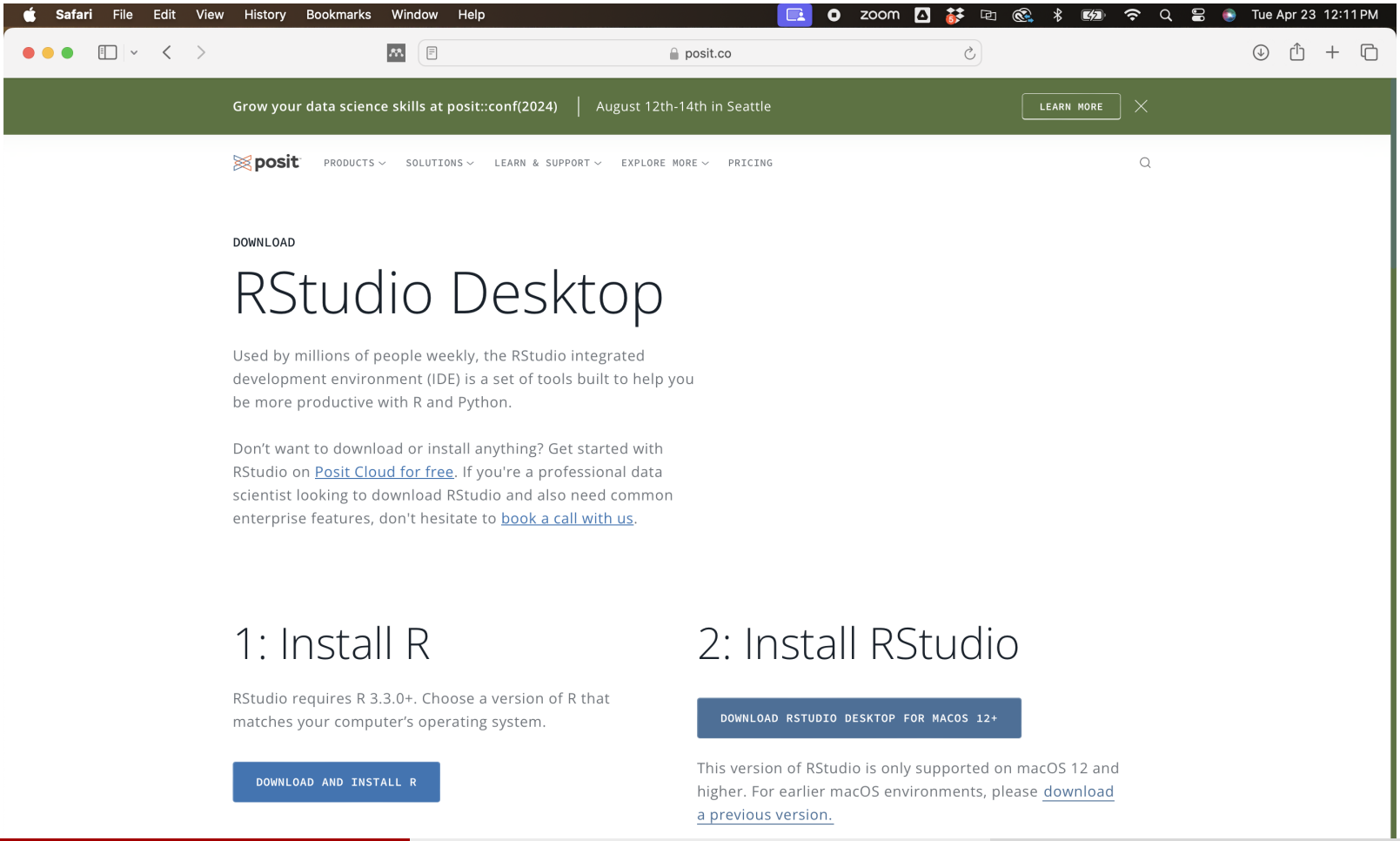
Installing RStudio (Windows and Mac)

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



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.



More on R and Rstudio

See more detailed instructions at in Lecture 1 at:
https://github.com/wevanjohnson/2024_04_R_tutorial

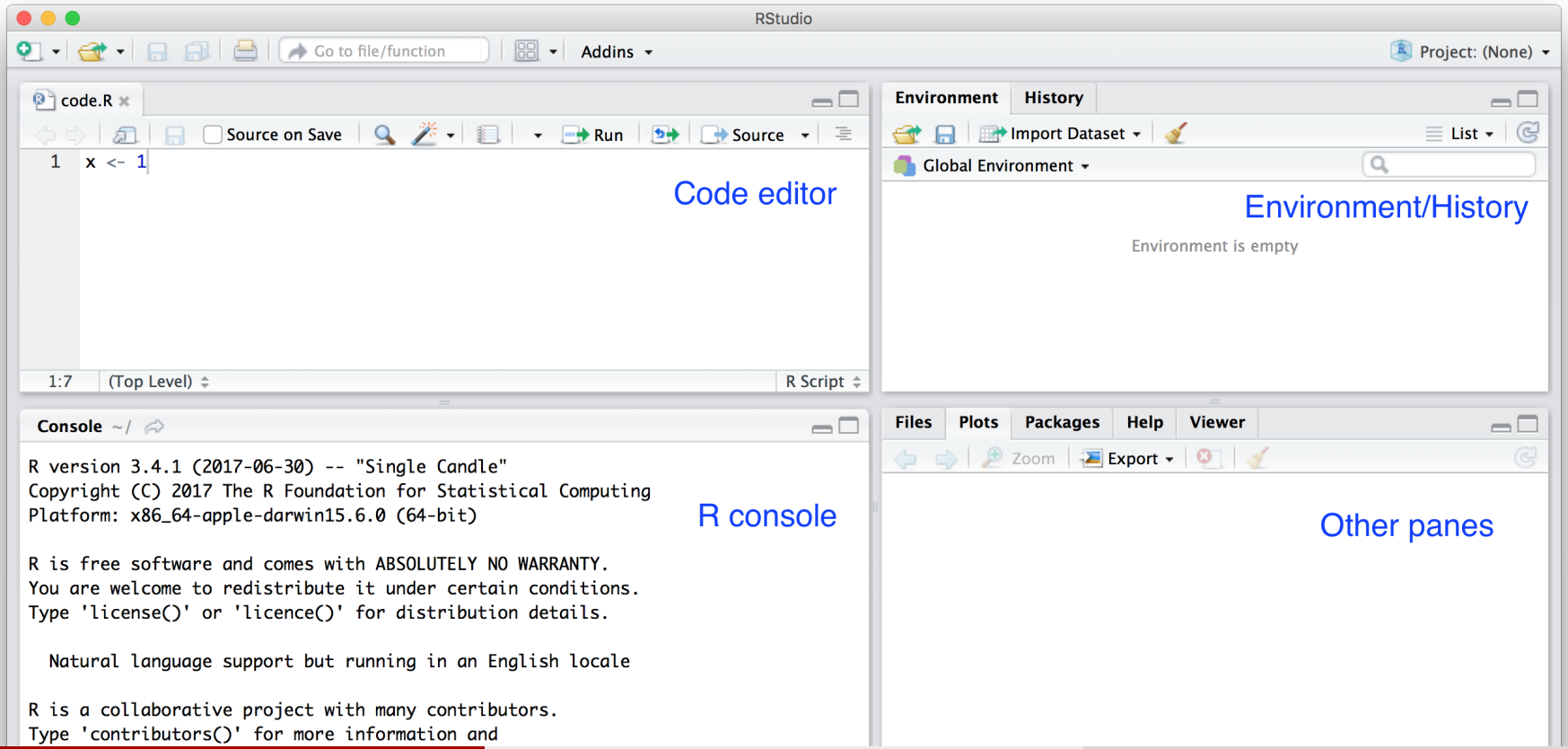
Why R?

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.

Volume 75, Issue 6
15 September 2022

Article Contents

Abstract

METHODS

RESULTS

JOURNAL ARTICLE

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

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

Author Notes

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 ▾

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.

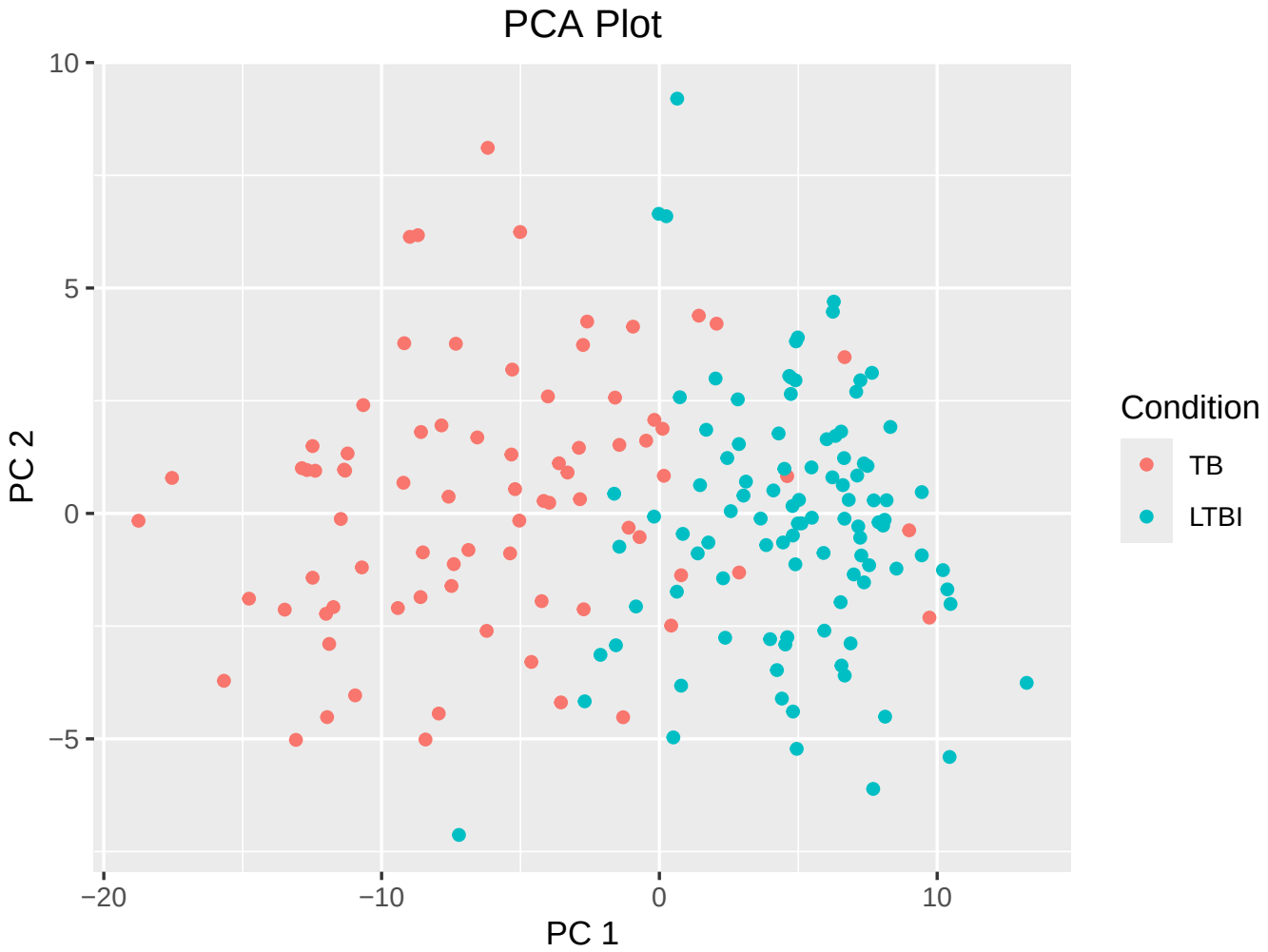
Data Visualization with R

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!

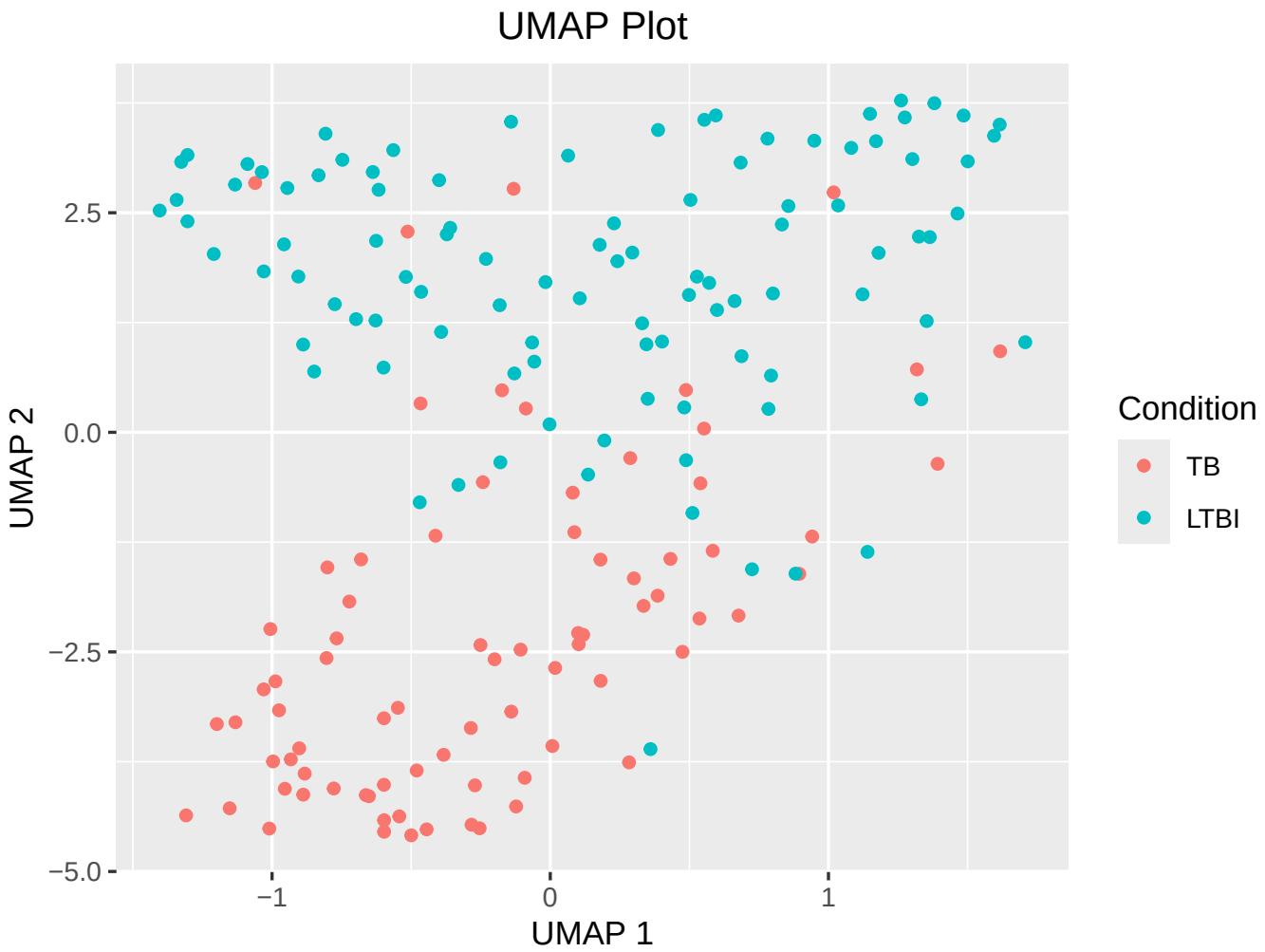
Motivating Example: PCA

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.

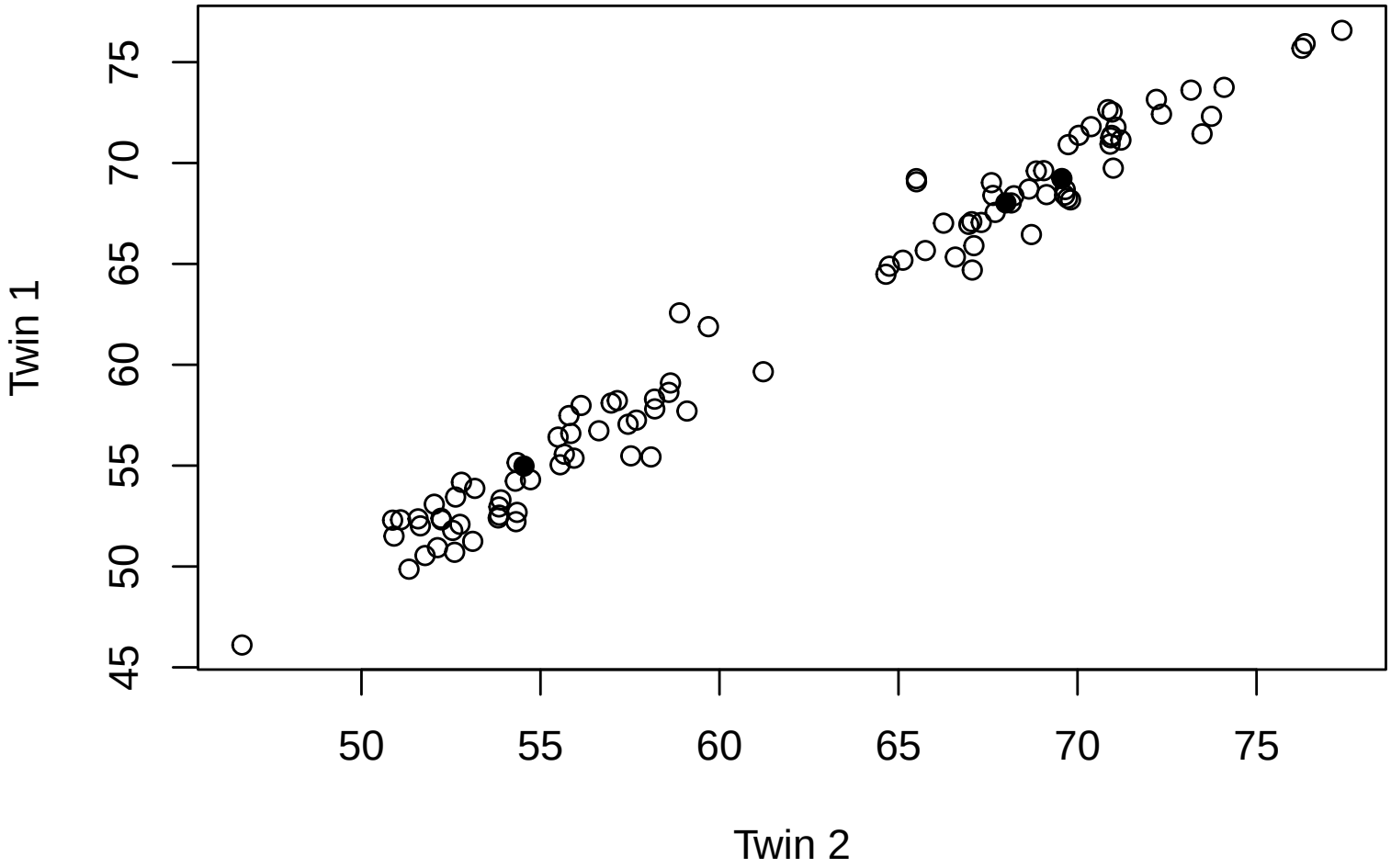
Dimension reduction: twin heights

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

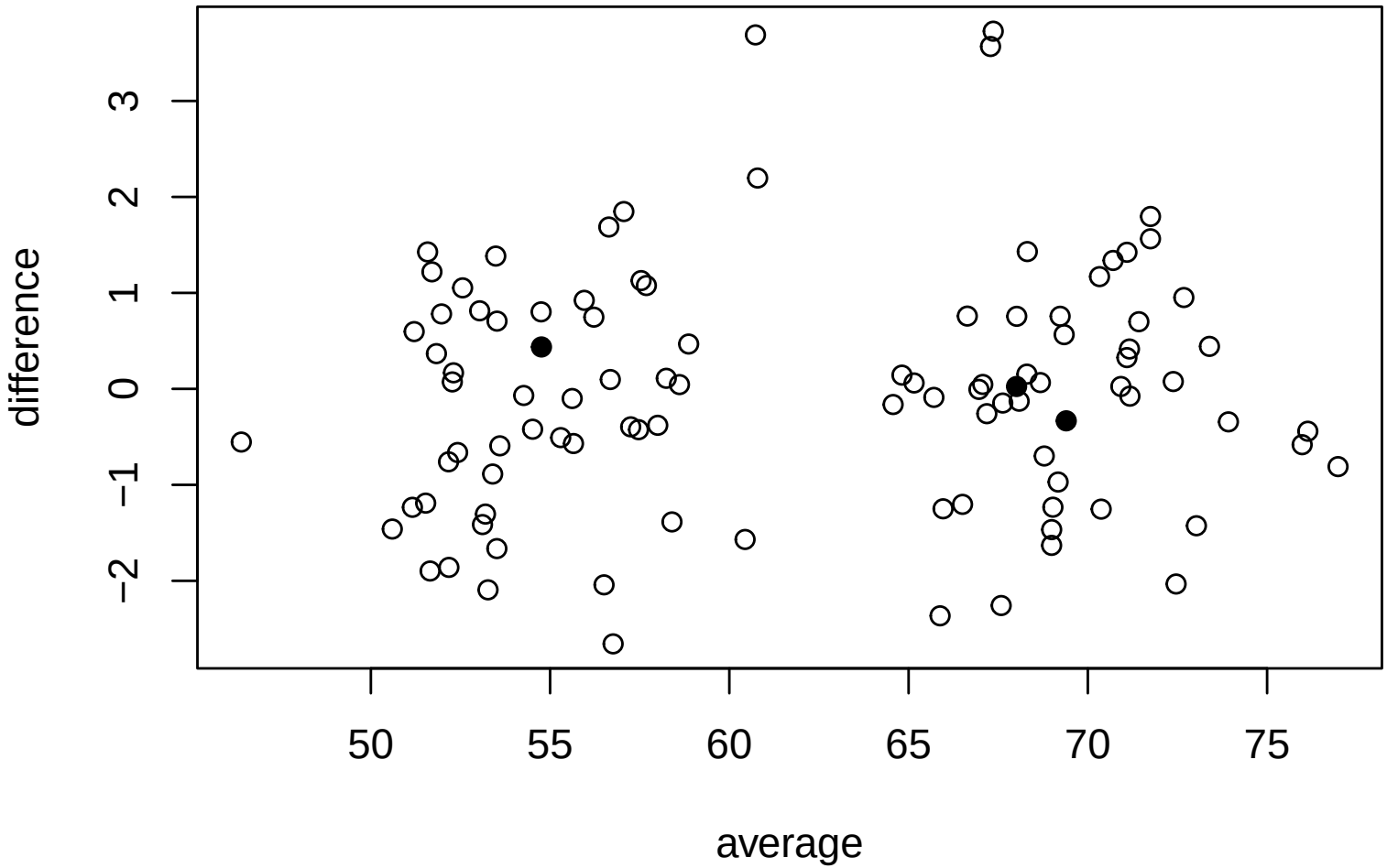
Now, can we pick a one-dimensional summary that makes this approximation even better?

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])
```

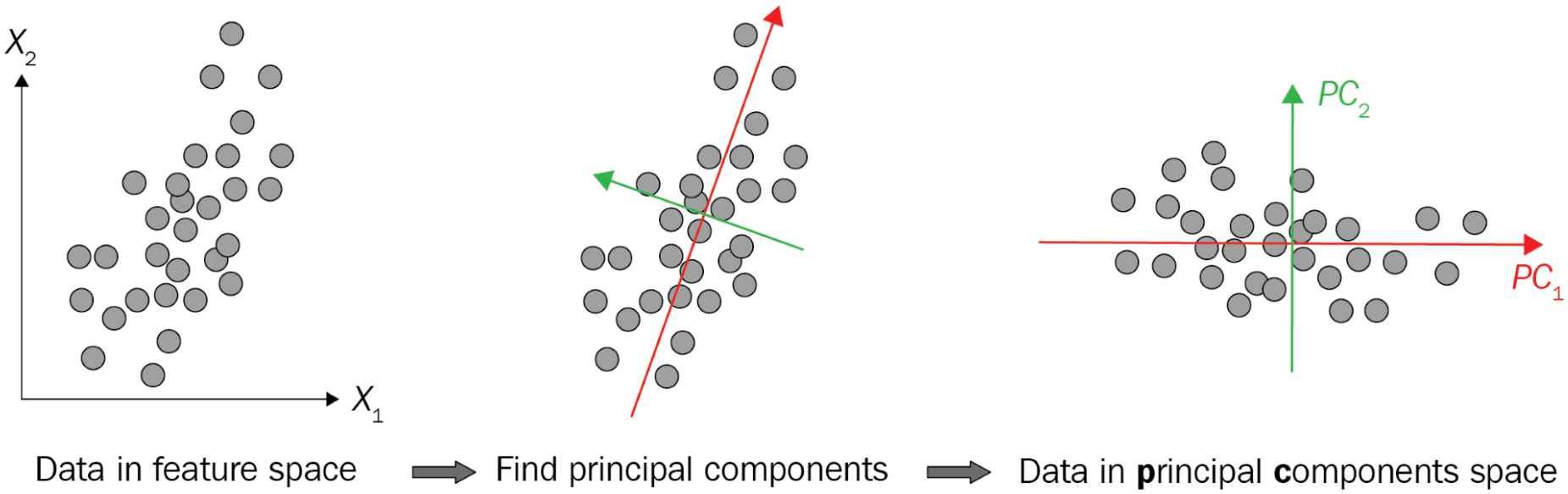
Dimension reduction: twin heights

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.



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)

# View(pca_reduction)
```

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[2] 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 acan 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 clinking here:

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

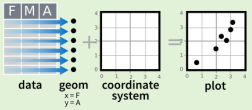
Data Visualization with ggplot2

Data Visualization with ggplot2 Cheat Sheet

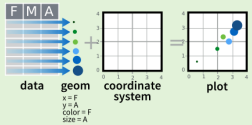
RStudio

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 **qplot()** or **ggplot()**

aesthetic mappings | data | geom

qplot(x = cty, y = hwy, color = cyl, data = mpg, geom = "point")

Creates a complete 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 qplot().

data

```
ggplot(mpg, aes(hwy, cty)) +
  geom_point(aes(color = cyl)) +
  geom_smooth(method = "lm") +
  coord_cartesian() +
  scale_color_gradient() +
  theme_bw()
```

add layers, elements with +

layer = geom + default stat + layer specific mappings

additional elements

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()

Returns 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(hwy))

a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

a + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

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(fl))

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

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

d + geom_path(lineend="butt", linejoin="round', linemitre=1**)**
x, y, alpha, color, linetype, size

d + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)**)**
x, ymax, ymin, 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, xend, y, yend, alpha, color, linetype, size

e + geom_rect(aes(xmin = long, ymin = lat, xmax= long + delta_long

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

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + geom_bar(stat = "identity")
x, y, alpha, color, fill, linetype, size, weight

g + geom_boxplot()
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + geom_dotplot(binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill

g + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

h + geom_jitter()
x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

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

k + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

l + geom_map(aes(map_id = state), map = map**) + expand_limits(**x = map$long, y = map$lat**)**
map_id, alpha, color, fill, linetype, size

Three Variables

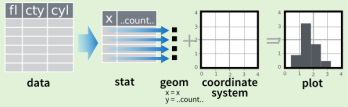
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

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")**

stat function | layer specific mappings | variable created by transformation

i + stat_density2d(aes(fill = ..level..), geom = "polygon", n = 100)

geom for layer | parameters for stat

1D distributions

a + stat_bin(binwidth = 1, origin = 10**)**
x, y | ..count.., ..ncount.., ..density.., ..ndensity..
a + stat_bindot(binwidth = 1, binaxis = "x"**)**
x, y, | ..count.., ..ncount..
a + stat_density(adjust = 1, kernel = "gaussian"**)**
x, y, | ..count.., ..density.., ..scaled..

2D distributions

f + stat_bin2d(bins = 30, drop = TRUE**)**
x, y, fill | ..count.., ..density..
f + stat_binhex(bins = 30**)**
x, y, fill | ..count.., ..density..
f + stat_density2d(contour = TRUE, n = 100**)**
x, y, color, size | ..level..

3 Variables

m + stat_contour(aes(z = z)**)**
x, y, z, order | ..level..
m+ stat_spoke(aes(radius= z, angle = z)**)**
angle, radius, x, xend, y, yend | ..x.., ..xend.., ..y.., ..yend..
m + stat_summary_hex(aes(z = z), bins = 30, fun = mean**)**
x, y, z, fill | ..value..
m + stat_summary2d(aes(z = z), bins = 30, fun = mean**)**
x, y, z, fill | ..value..

Comparisons

g + stat_boxplot(coef = 1.5**)**
x, y | ..lower.., ..middle.., ..upper.., ..outliers..
g + stat_ydensity(adjust = 1, kernel = "gaussian", scale = "area"**)**
x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

Functions

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

General Purpose

ggplot() + stat_function(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5)**)**
x | ..y..
f + stat_identity()
ggplot() + stat_qq(aes(sample=1:100), distribution = qt, dparams = list(df=5)**)**

Scales

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

n <- b + geom_bar(aes(fill = fl)**)**
n

scale_ | aesthetic to adjust | prepackaged scale to use | scale specific arguments

n + scale_fill_manual(values = c("skyblue", "royalblue", "blue", "navy"), limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"), name = "fuel", labels = c("D", "E", "P", "R")**)**

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

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(values = c()**)** - map discrete values to manually chosen visual values

X and Y location scales

Use with x or y aesthetics (x shown 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 f ll scales

Discrete

n <- b + geom_bar(aes(fill = fl)**)**

n + scale_fill_brewer(palette = "Blues"**)**
For palette choices:
library(RColorBrewer)
display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red"**)**

Continuous

o <- a + geom_dotplot(aes(fill = ..x..)**)**

o + scale_fill_gradient(low = "red", high = "yellow"**)**

o + scale_fill_gradient2(low = "red", hight = "blue", mid = "white", midpoint = 25**)**

o + scale_fill_gradientn(colours = terrain.colors(6)**)**
Also: rainbow(), heat.colors(), topo.colors(), cm.colors(), RColorBrewer::brewer.pal()

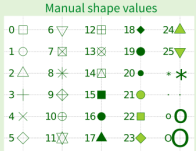
Shape scales

p <- f + geom_point(aes(shape = fl)**)**

p + scale_shape(solid = FALSE**)**

p + scale_shape_manual(values = c(3:7)**)**
Shape values shown in chart on right

Manual shape values



Coordinate Systems

r <- b + geom_bar()

r + coord_cartesian(xlim = c(0, 5)**)**
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2**)**
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1 **)**
theta, start, direction
Polar coordinates

r + coord_trans(ytrans = "sqrt"**)**
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set extras and strains to the name of a window function.

z + coord_map(projection = "ortho", orientation=c(41, -74, 0)**)**
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Faceting

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

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(. ~ fl)
facet into columns based on fl

t + facet_grid(year ~ .)
facet into rows based on year

t + facet_grid(year ~ fl)
facet into both rows and columns

t + facet_wrap(~ fl)
wrap facets into a rectangular layout

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

t + facet_grid(y ~ 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 **labeller** to adjust facet labels

t + facet_grid(. ~ fl, labeller = label_both)

f : c	f : d	f : e	f : p	f : r

t + facet_grid(. ~ fl, labeller = label_bquote(alpha ^ .(x)**))**

↵c	↵d	↵e	↵p	↵f

t + facet_grid(. ~ fl, labeller = 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(fl, 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 overplotting

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

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

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

Use scale functions to update legend labels

Legends

t + theme(legend.position = "bottom"**)**
Place legend at "bottom", "top", "lef ", 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.

Themes

r + theme_bw()
White background with grid lines

r + theme_classic()
White background no gridlines

r + theme_grey()

r + theme_minimal()

Zooming

Without clipping (preferred)
t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20)**)**

With clipping (removes unseen 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:

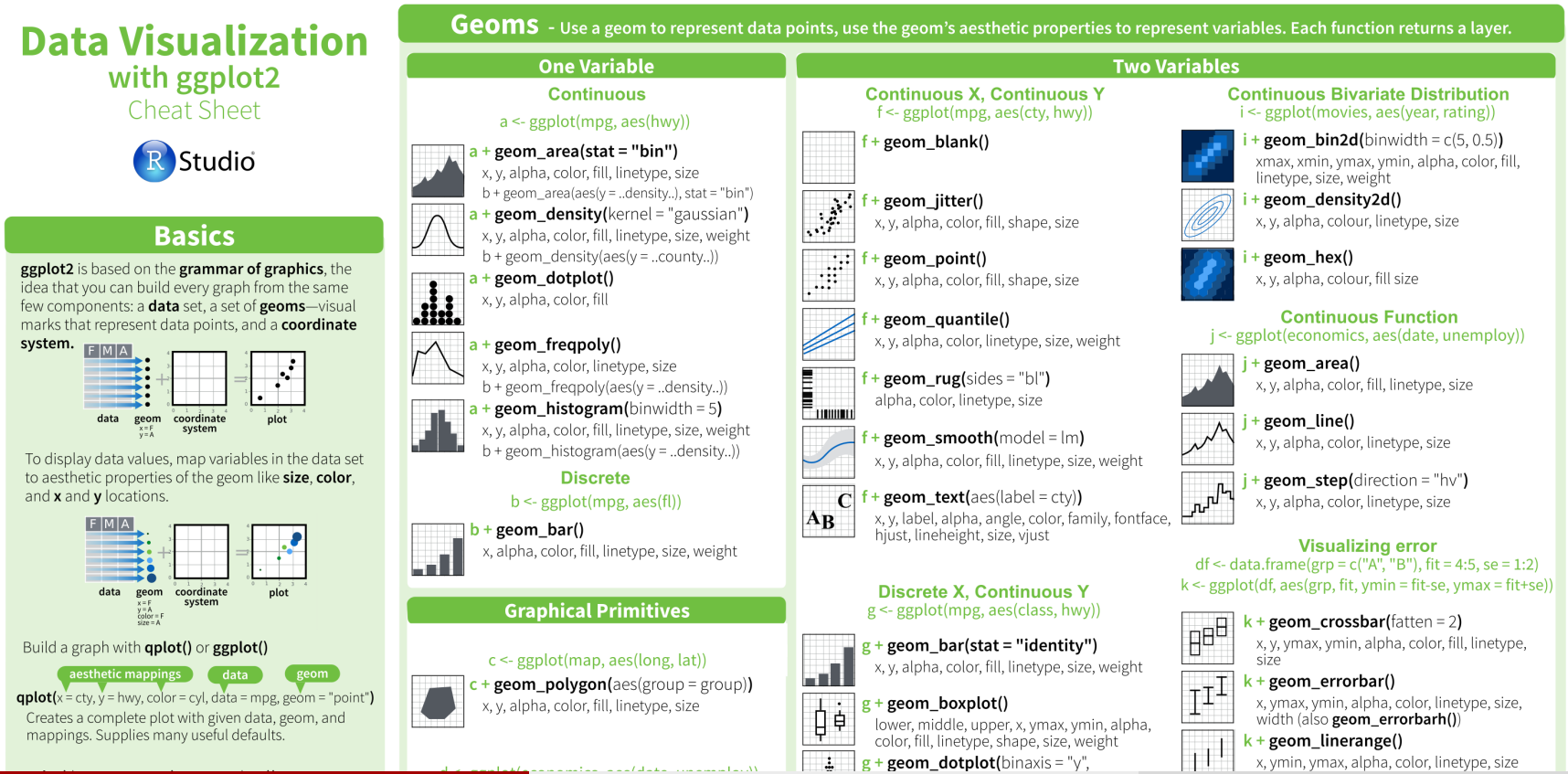
```
p <- pca_reduction %>% ggplot()
```

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

RStudio

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.**

data · geom · coordinate system · plot

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 **qplot()** or **ggplot()**

aesthetic mappings · data · geom

qplot(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")

a + **geom_density(**kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

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(fl))

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 <- 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

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + **geom_bar(stat = "identity")**
x, y, alpha, color, fill, linetype, size, weight

g + **geom_boxplot()**
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + **geom_dotplot(**binaxis = "y",

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

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

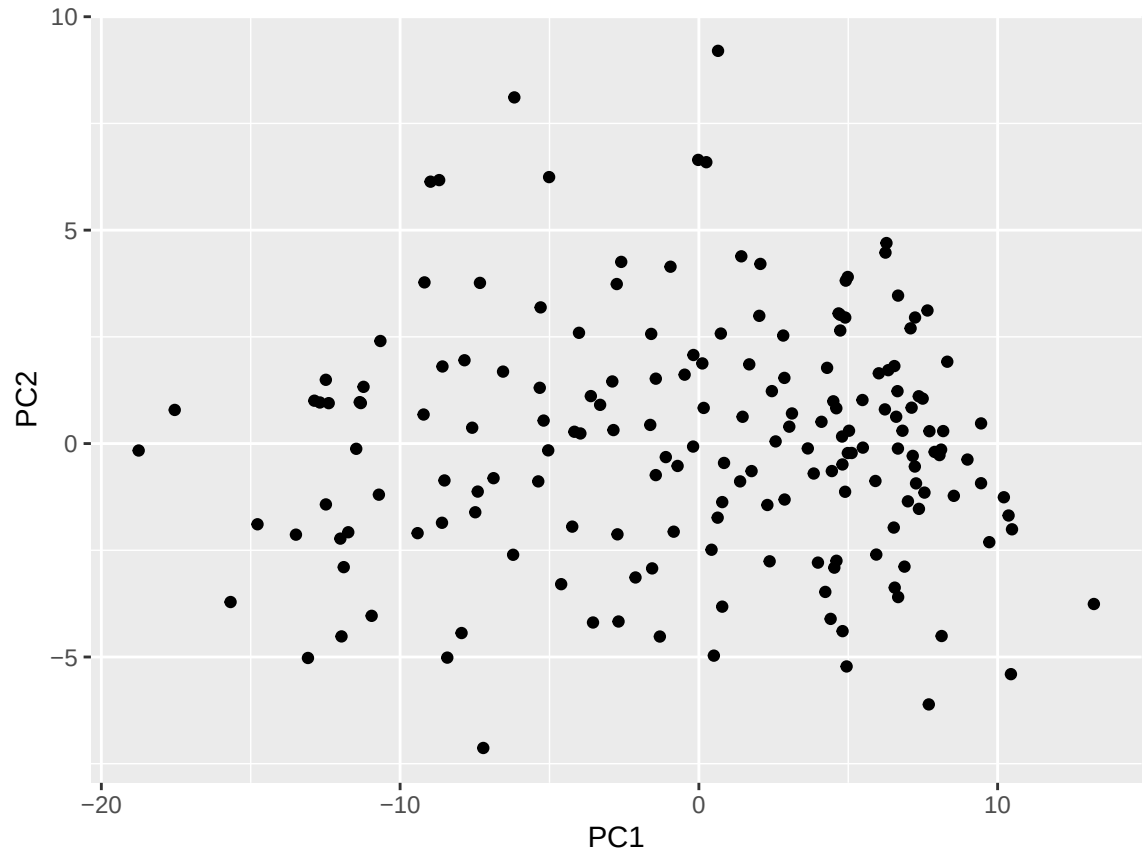
Geometry function names follow the pattern: `geom_X` where X is the name of the geometry. Some examples include `geom_point`, `geom_bar`, and `geom_histogram`.

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:

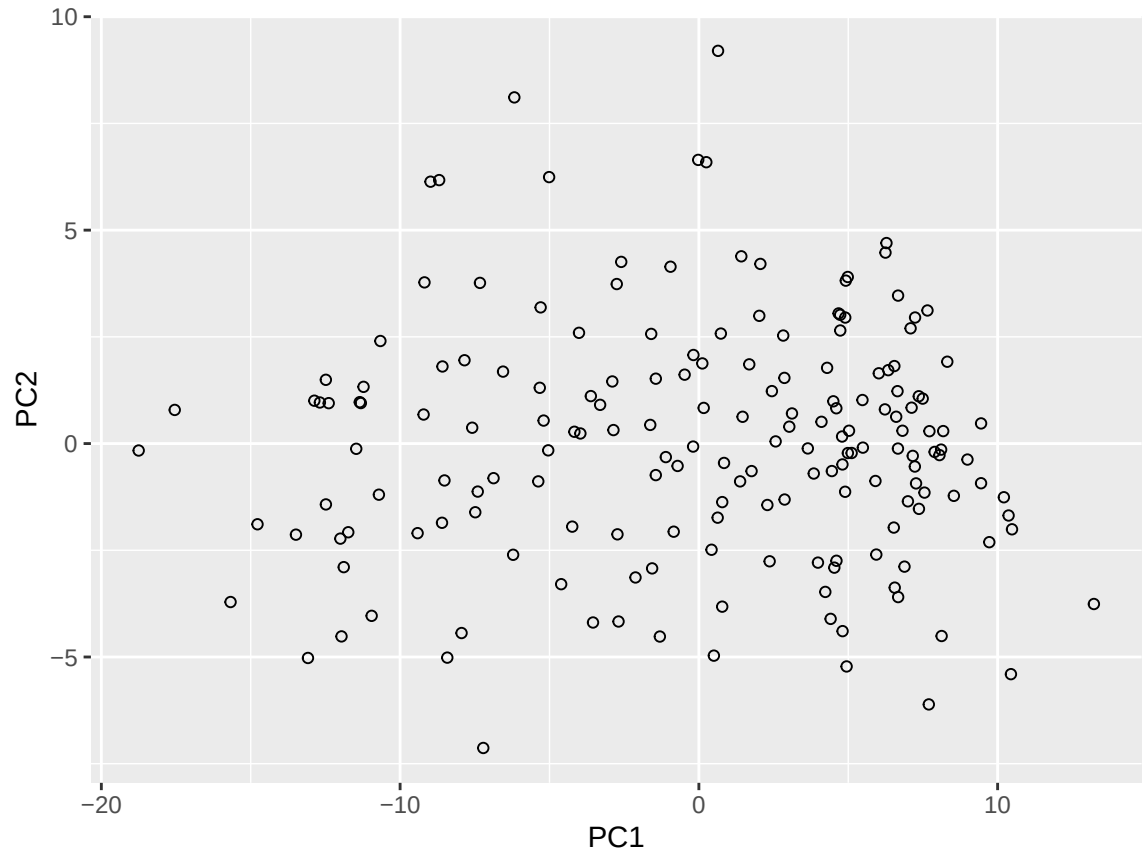
```
p + geom_point(aes(x=PC1, y=PC2))
```



Geometries: additional options

We can change the shape or color of the points

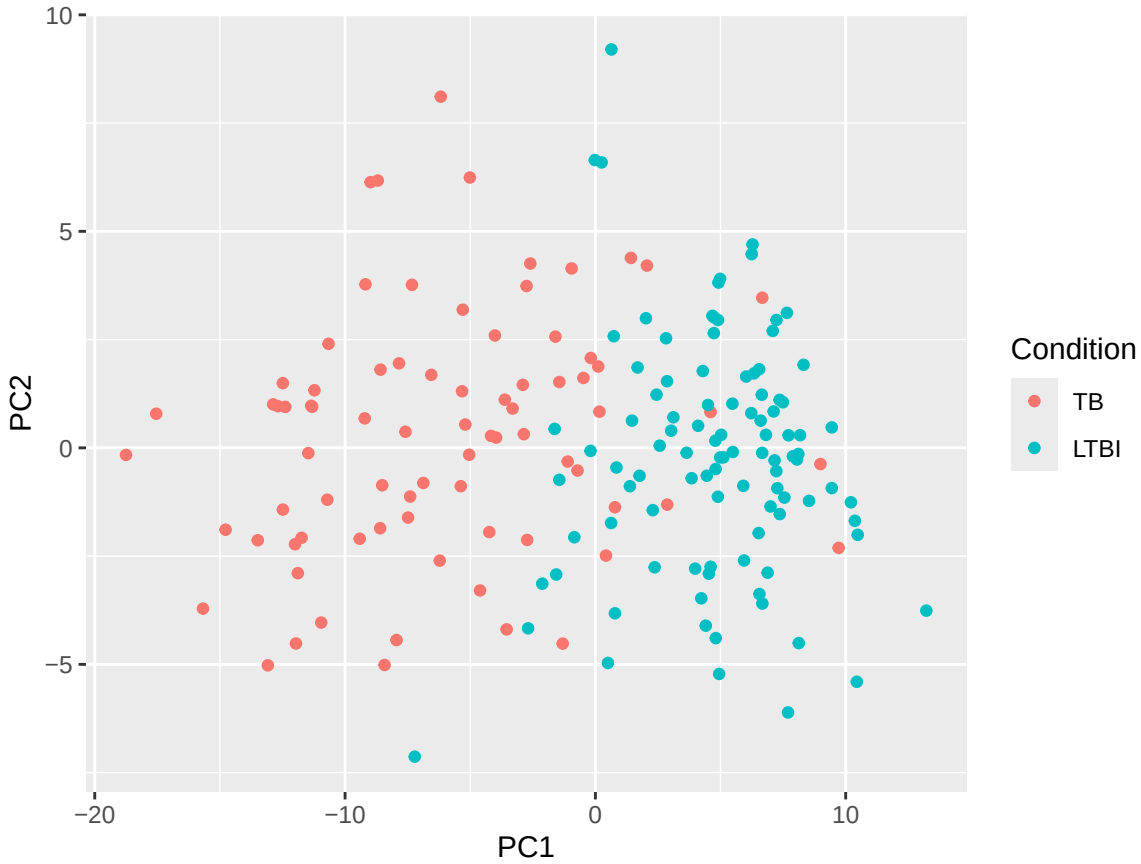
```
p + geom_point(aes(x=PC1, y=PC2), shape=1)
```



Geometries: additional aesthetics

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

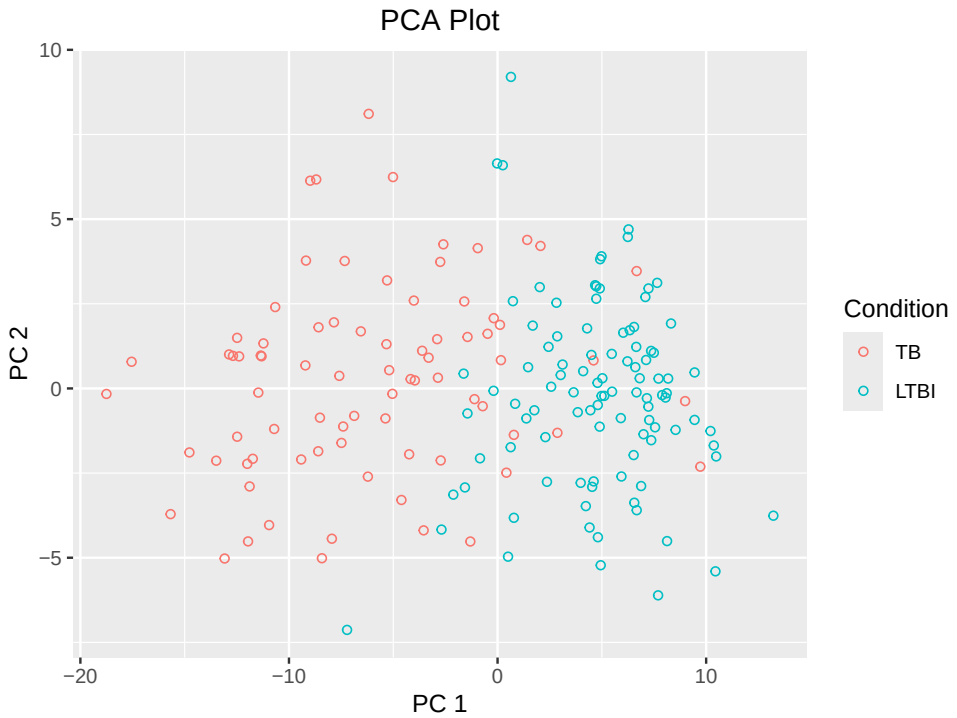
```
p + geom_point(aes(x=PC1, y=PC2, color=Condition))
```



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:

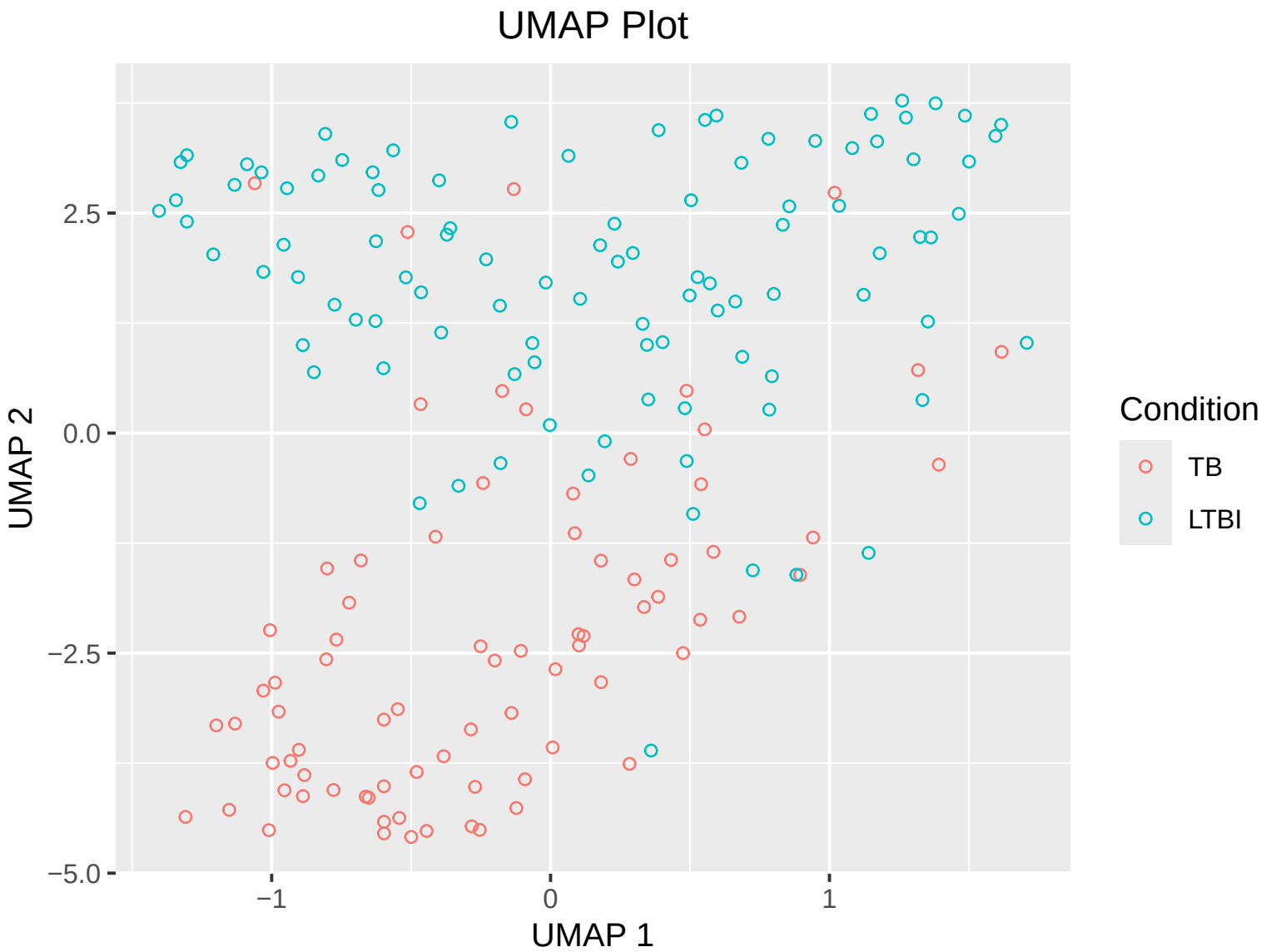
```
## read in data
TBnanostring <- readRDS("TBnanostring.rds")

## 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 ggpplot
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] tidyr_1.3.1     tibble_3.2.1    ggplot2_3.5.1   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
```