The Elizabeth H.
and James S. McDonnell III
McDonnell Genome Institute
at Washington University

GenViz Module 2: Using R for genomic data visualization and interpretation

Malachi Griffith, Obi Griffith, Zachary Skidmore

Genomic Data Visualization and Interpretation
April 8-12, 2019
Freie Universität Berlin

Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

This is a human-readable summary of (and not a substitute for) the license. Disclaimer.

You are free to:

Share — copy and redistribute the material in any medium or format

Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Learning objectives of the course

- Module 1: Introduction to genomic data visualization and interpretation
- **Module 2: Using R for genomic data visualization and interpretation**
- Module 3: Introduction to GenVisR
- Module 4: Expression profiling, visualization, and interpretation
- Module 5: Variant annotation and interpretation
- Module 6: Q & A, discussion, integrated assignments, and working with your own data

- Tutorials
  - Provide working examples of data visualization and interpretation
  - Self contained, self explanatory, portable

Learning objectives of module 2

- Review basic R usage
- Learn to use R for basic data manipulation
- Learn to create publication quality graphs to display data
- Learn to create interactive graphics

A brief history of R

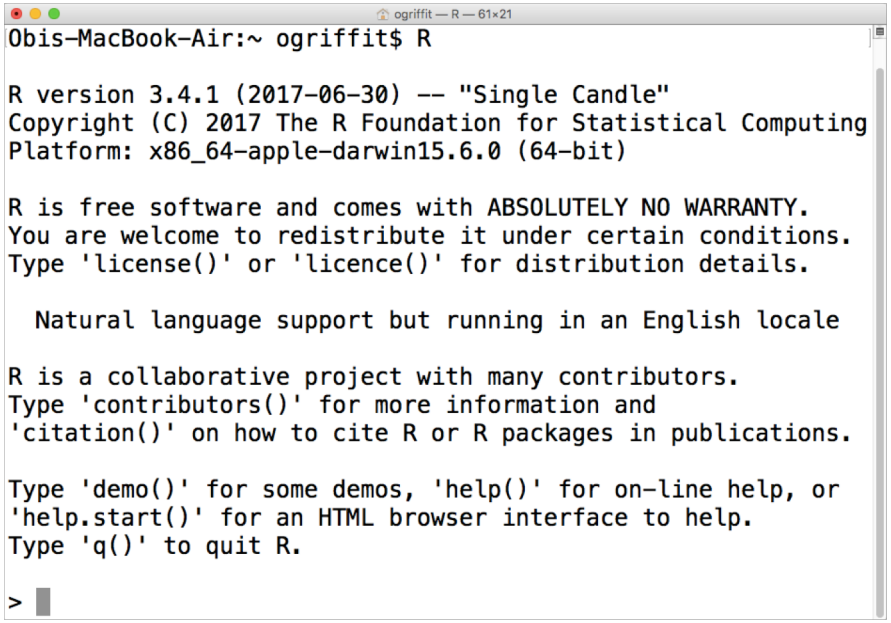
- R is an implementation of the S programming language combined with lexical scoping semantics inspired by Scheme.
- S was created by John Chambers while at Bell Labs
- There are some important differences, but much of the code written for S runs unaltered.
- R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand
- Currently developed by the R Development Core Team, of which Chambers is a member.
- The R project was conceived in 1992, with an initial version released in 1995 and a stable beta version in 2000

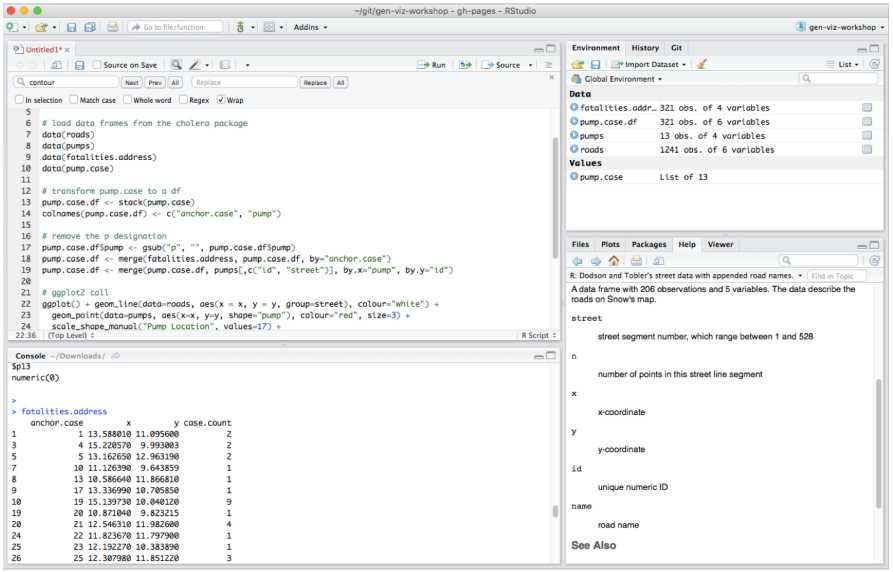
https://en.wikipedia.org/wiki/R_(programming_language)

R is available via command-line or a number of integrated development environments (IDE)

R Project

Rstudio





https://cran.r-project.org/

Open-source, non-profit

https://www.rstudio.com/

Open-source, free + commercial

Installation and versions

- Rstudio installation is very simple
- R installation generally only a little more complicated
- Pre-compiled binaries exist for most operating systems

- Be aware of R versions
  - Occasionally some packages may be version dependent or interdependent
  - Less of an issue these days
  - Simplest to keep R, Rstudio and BioConductor updated to current/latest version
  - Rswitch – allows multiple versions of R/Rstudio to be maintained simultaneously

CRAN and BioConductor

- 11,411 available packages
- All applications
  - ggplot2, cluster, dplyr, reshape2, randomForest, RColorBrewer
- https://cran.r-project.org/

```
install.packages()
```

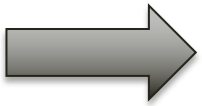
- 1,381 available packages
- Genomic applications
  - AnnotationDBI, GenomicRanges, limma, biomaRt, affy, GEOquery
- https://bioconductor.org/

```
source("https://bioconductor.org/biocLite.R")
biocLite()
```

Getting help: ?, vignette(), and data()

- Type '?' before any function to get a manual style help page in R or Rstudio
- Similarly use vignette() together with a function/package name to get detailed usage vignettes
- Type data() to return a list of available demonstration datasets.

> ?apply



apply {base} R Documentation

Apply Functions Over Array Margins

Description

Returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

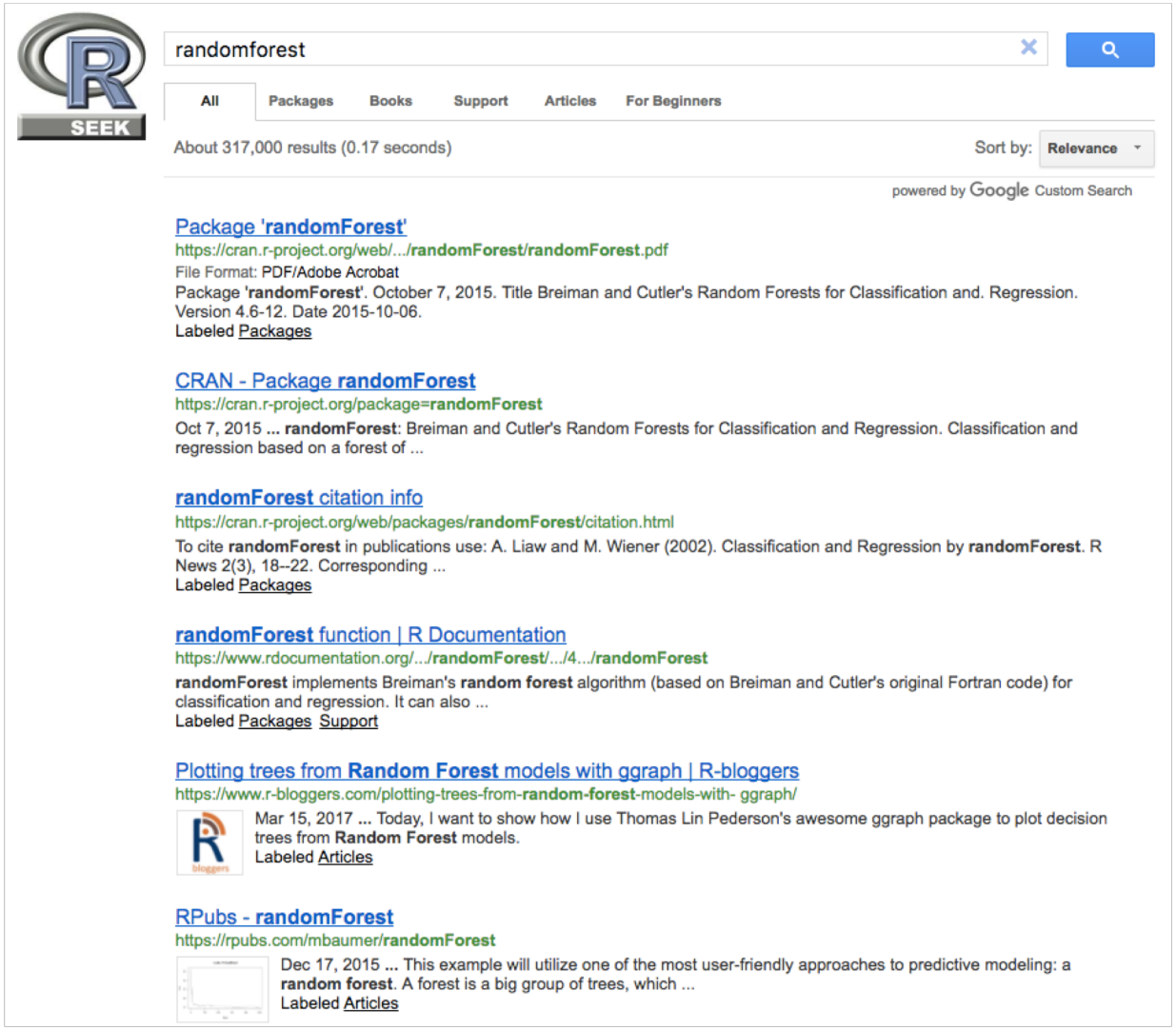
Usage

```
apply(X, MARGIN, FUN, ...)
```

Arguments

X	an array, including a matrix.
MARGIN	a vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where X has named dimnames, it can be a character vector selecting dimension names.
FUN	the function to be applied: see 'Details'. In the case of functions like +, %*%, etc., the function name must be backquoted or quoted.
...	optional arguments to FUN.

Rseek.org to search CRAN, r-bloggers, support.rstudio.com, rpubs.com, etc



Variables, Data Structure (Object) Types and Data types

- As with any programming language, you need to use variables to store information. When you create a variable you reserve some space in memory and keep a record of its location for later retrieval and use.
- The information you wish to store might be characters (e.g., text), integers, boolean (e.g., True/False) etc.
- In contrast to many other programming languages (e.g., C, java, etc), in R variables are not declared as a specific data type.
  - The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable.
- The simplest R-object type is the atomic vector
  - There are six data types for atomic vectors, also termed as six classes of vectors: logical, numerical, integer, complex, character, and raw
- Lists are also vectors but are not atomic vectors, meaning that they can include multiple data types and can be recursive (contain lists of lists)
- The other R-Objects are built upon atomic vectors and include: factors, matrices, arrays, data frames,

Understanding object and data types with class(), typeof() and is.*()

x	class(x)	typeof(x)	is.*(x)
x <- 1.0	numeric	double	is.numeric(x)=TRUE is.double(x)=TRUE
x <- 1L	integer	integer	is.integer(x)=TRUE
x <- “foo”	character	character	is.character(x)=TRUE
x <- TRUE	logical	logical	is.logical(x)=TRUE
x <- charToRaw(“foo”)	raw	raw	is.raw(x)=TRUE
x <- 4 + 4i	complex	complex	is.complex(x)=TRUE
x <- factor(c("foo","bar","bar"))	factor	integer	is.factor(x)=TRUE is.integer(x)=TRUE
x <- matrix(1:4, nrow=2)	matrix	integer	is.matrix(x)=TRUE Is.integer(x)=TRUE
x <- data.frame(x=1:2, y=c(“foo”, “bar”))	data.frame	list	is.data.frame(x)=TRUE is.list(x)=TRUE

Vectors

- Typically defined with c() or extracted from other objects

```
> myVector <- c(“foo”, “bar”, “baz”)

> myVector2 <- c(2,3,5:10,15,20,25,30)
```

- Subsets of vectors can be extracted by index values

```
> myVector[3]
[1] “baz”
```

- Conditional statements can be applied to vectors

```
myVector2 >= 5
[1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE
```

Dataframes

Demo dataframe available in R: “mtcars”

```
> mtcars
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona     21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin       15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28        13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9         27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2     26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa      30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L    15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino      19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora     15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E        21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

- Similar to vectors, you can extract subsets of dataframes using [] but now with two dimensions:
  - e.g., data[ROW,COL]
- You can also extract by row and column name:
  - e.g., mtcars$mpg

Importing and Exporting Data

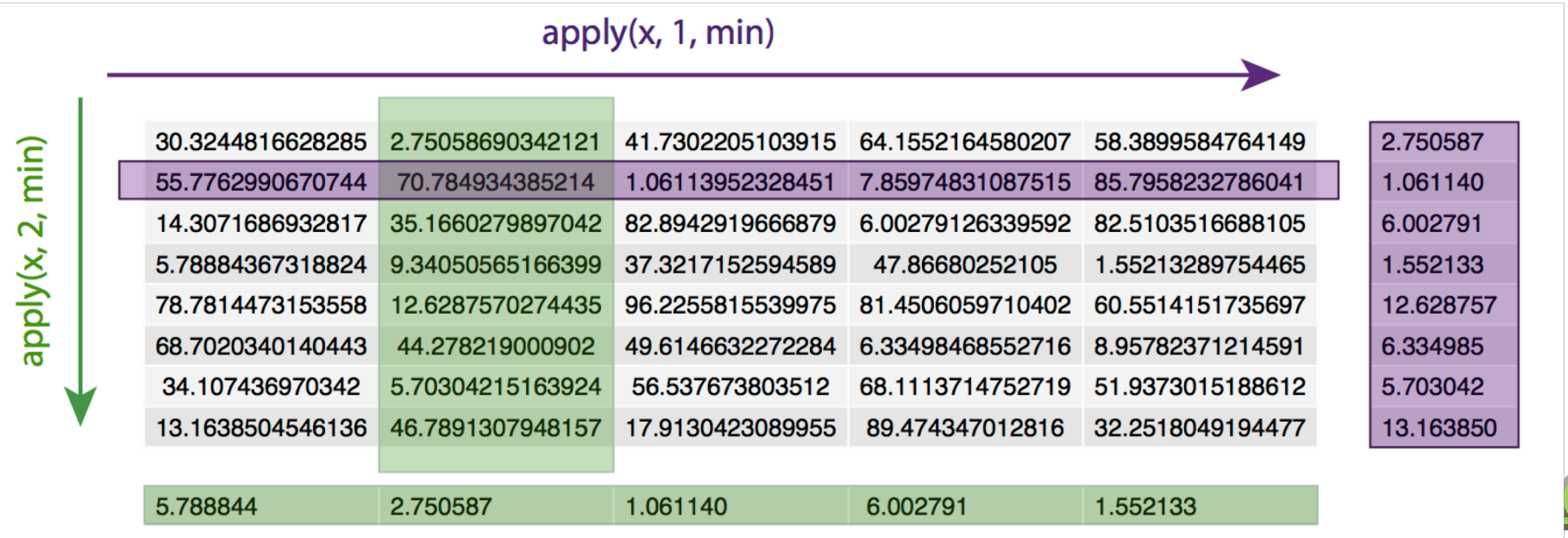
- R functions exist to import most common file formats
  - tsv, csv, excel, xml, json, etc
- read.table() is particularly useful for simple delimited data files

```
data <- read.table(file="my file or URL", header=TRUE, sep="\t",
na.strings = c("NA","N/A","na"), as.is=c(1:27,29:30), row.names=1, …)
```

- Read from file or web URL
- Tell R how you have encoded missing values
- Use as.is to prevent R from converting strings to factors

Apply functions

```
# Create a matrix of 8 rows by 5 columns as above:
x <- matrix(runif(n=40, min=1, max=100), ncol=5)

# Calculate min value for each row with a for loop
for(i in 1:8){
   print(min(x[i,]))
}

# use apply function to do same thing
apply(x, 1, min)
```



Custom functions

- Often necessary when an existing R function doesn't quite do what you need.
- Especially powerful in combination with apply() functions

- Basic structure:

```
myfun <- function(input){
    output <- do_something(input)
    return(output)
}

apply(mydata, 1, myfun)
```

Assignment - the age old “<-” vs “=” debate

- By convention in R, “<-” is used to assign a value to a variable and “=” is used for setting arguments in a function. For R purists, this is the preferred method.
- The “=” symbol will also generally work for assignment.

- Old keyboards had a “<-” key. Now it is an extra key stroke.

- There is a difference in “scope”. Generally you may want values assigned within functions to stay inside the function.
- Let’s look at the simple ‘median’ function to illustrate
  - Usage: median(x, na.rm = FALSE, ...)

```
> data=c(1:10,NA,NA,3:5)
> median(x=data, na.rm=TRUE)
[1] 5
> x
Error: object 'x' not found

> median(x <- data, na.rm=TRUE)
[1] 5
> x
[1]  1  2  3  4  5  6  7  8  9 10 NA NA  3  4  5
```

Assignment - the age old “<-” vs “=” debate

- With “<-” spacing matters. This drives some programmers crazy who think spacing should be a style matter and not change the way a program works

```
> x<-3
> x
[1] 3

> x <- 3
> x
[1] 3

> x < -3
> x
FALSE
```

Assignment - the age old “<-” vs “=” debate

- More weirdness having to do with order of interpretation by R

```
x <- y <- 5   # x and y equal 5
x = y = 5   # x and y equal 5
x = y <- 5   # x and y equal 5
x <- y = 5   # errors!
# Error in (x <- y) = 5 : could not find function "<-<-”
```

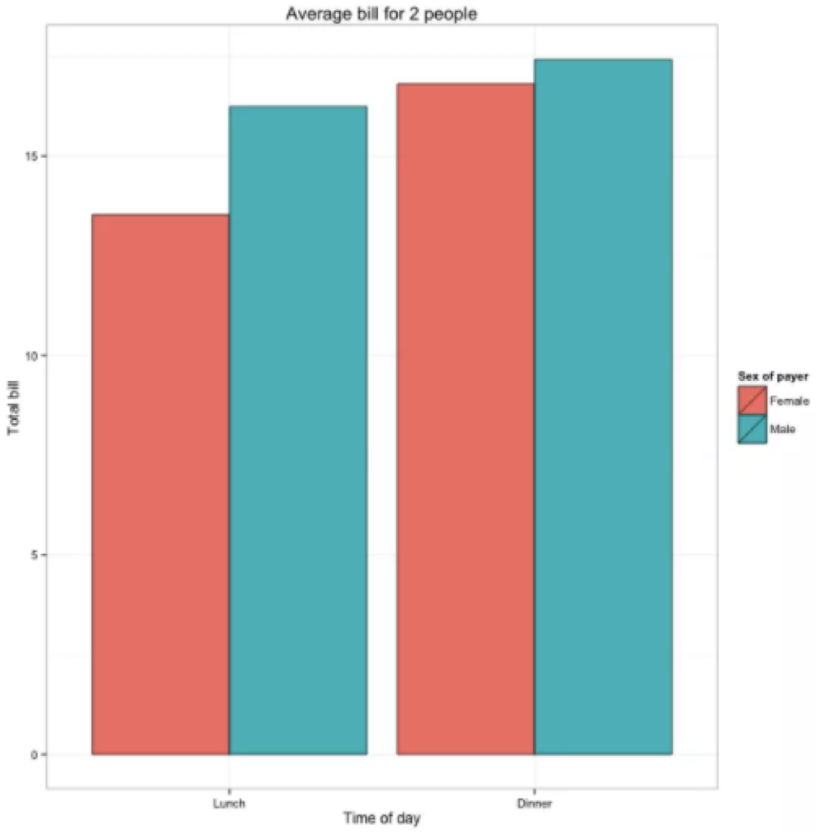
- We could keep going …
- Use “<-” if you want to be a l33t R geek.
- I’ve been writing code for 15 years and use “=” for assignment. Never had a problem.
- formatR package (tidy_* functions) can be used to clean your code for publication

Graphics options in R

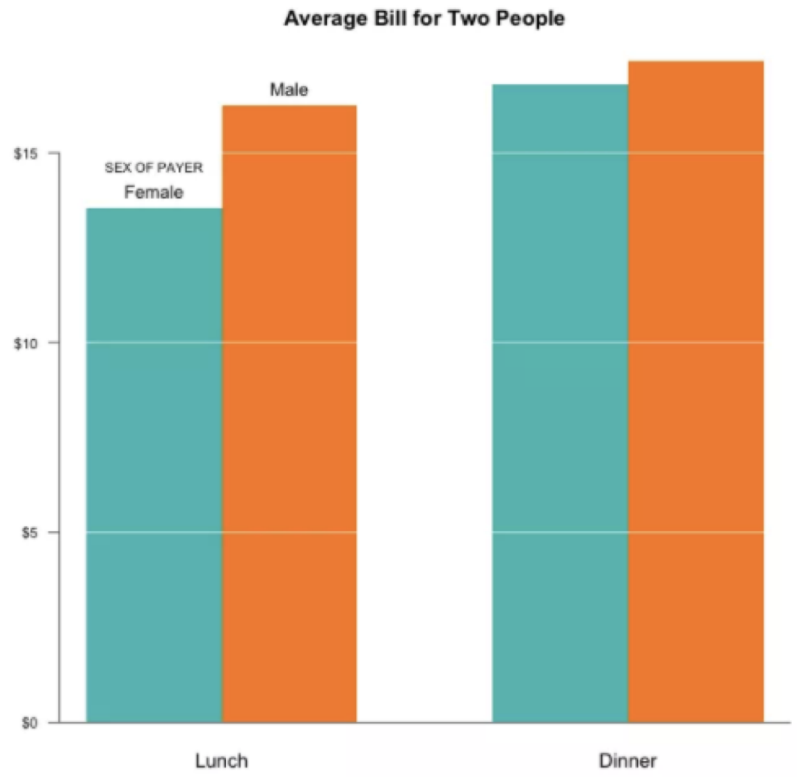
- At least 3 primary graphics options in R
  - base R graphics
    - plot(), par(), etc
  - lattice
  - **ggplot2**

- ggplot2 consistently one of the top two most popular of all R packages

Why use ggplot2? - “prettier” graphics in less lines of code




```
ggplot(data=dat1, aes(x=time, y=total_bill, fill=sex)) +
  geom_bar(colour="black", stat="identity",
           position=position_dodge(),
           size=.3) +
  scale_fill_hue(name="Sex of payer") +
  xlab("Time of day") + ylab("Total bill") +
  ggtitle("Average bill for 2 people") +
  theme_bw()
```




```
par(cex=1.2, cex.axis=1.1)
barplot(dat1mat, beside = TRUE, border=NA, col=mf_col,
    main="Average Bill for Two People", yaxt="n")
axis(2, at=axTicks(2), labels=sprintf("$%s", axTicks(2)),
    las=1, cex.axis=0.8)
grid(NA, NULL, lwd=1, lty=1, col="#ffffff")
abline(0, 0)
text(1.5, dat1mat["Female", "Lunch"], "Female", pos=3)
text(2.5, dat1mat["Male", "Lunch"], "Male", pos=3)
text(1.5, dat1mat["Female", "Lunch"]+0.7, "SEX OF PAYER",
    pos=3, cex=0.75)
```

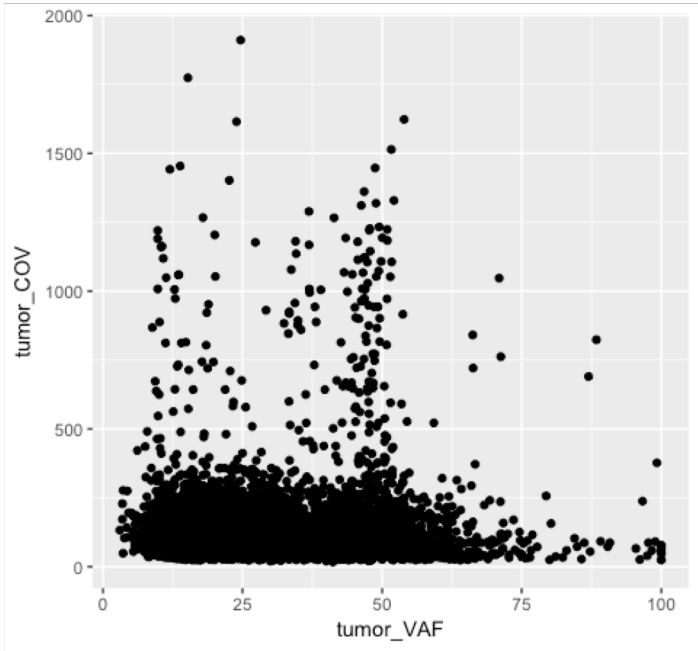
ggplot2 syntax

```
ggplot(data=variantData, aes(x=tumor_VAF, y=tumor_COV)) + geom_point()
```

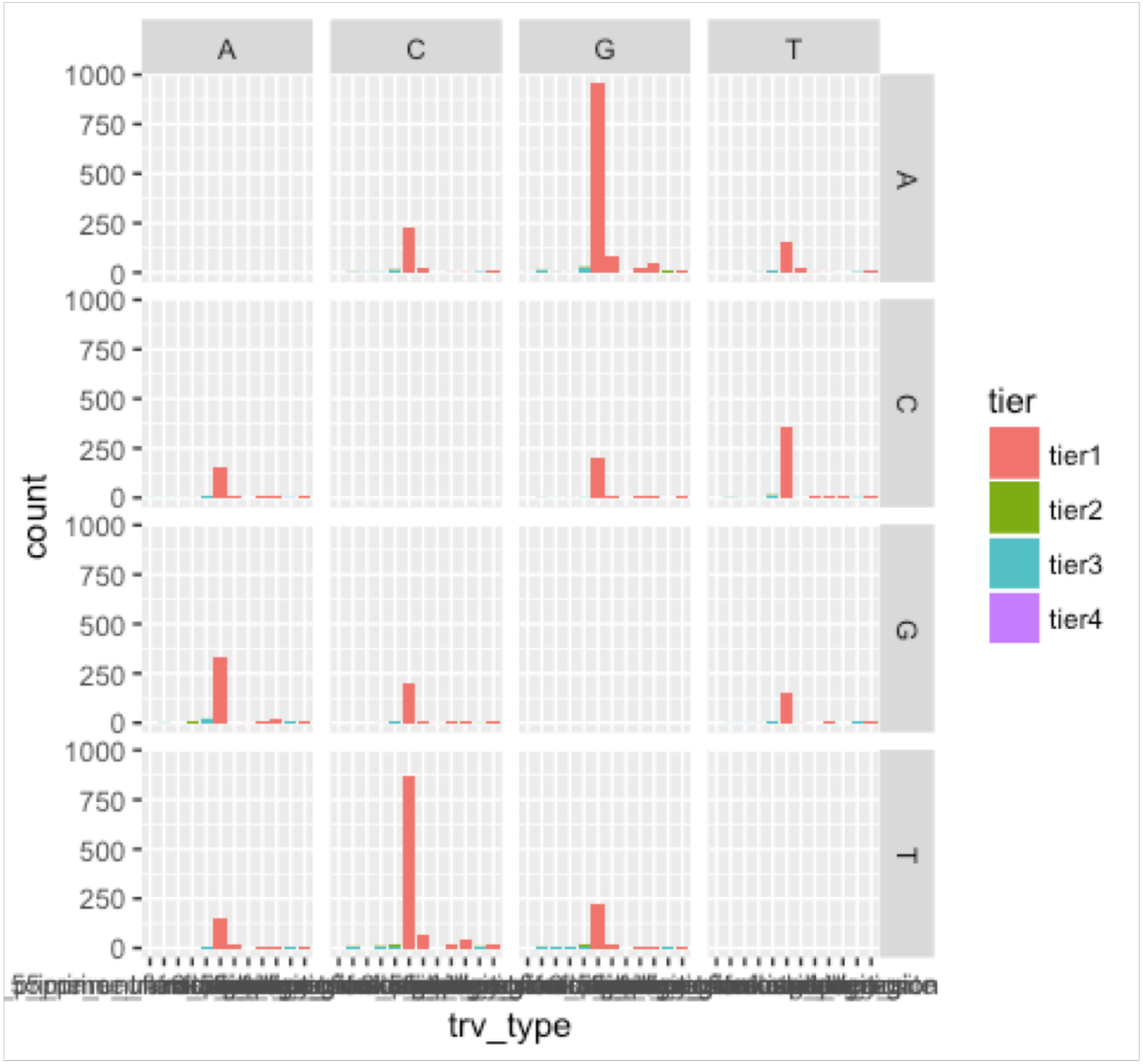
dataframe with data to be plotted

Aesthetic mappings describe how variables in the data are mapped to visual properties (aesthetics) of geometric objects (geoms)

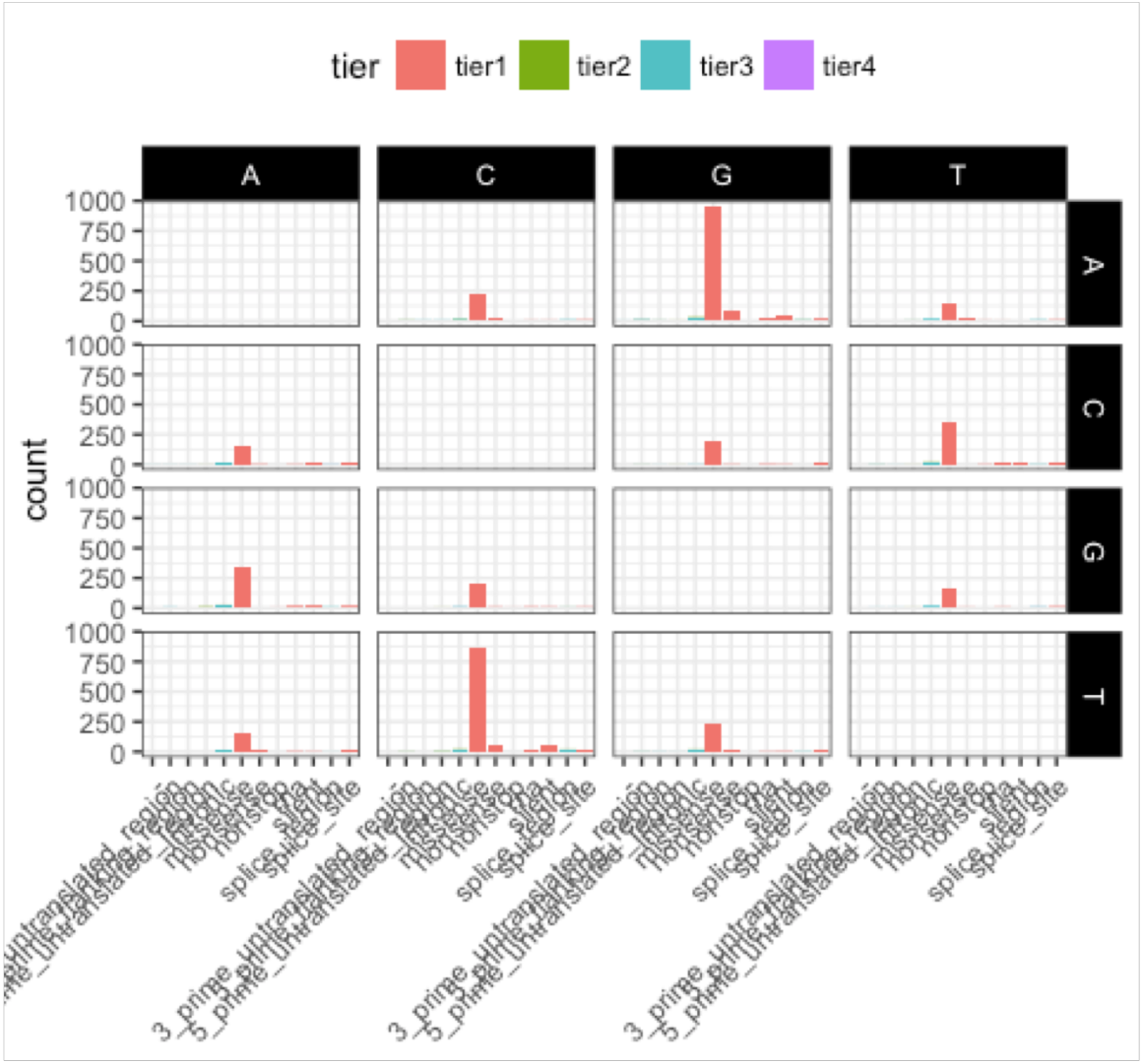
geometric objects specify how data should be plotted



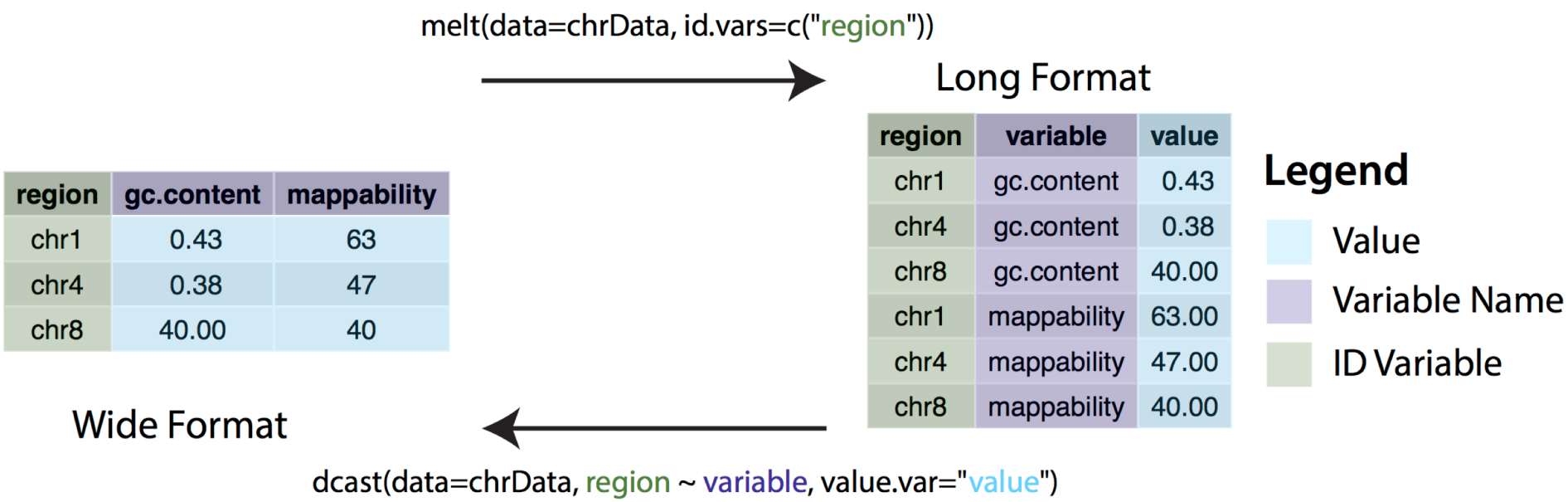
Faceting allows multiple plots in a single page



Themes allow stylistic improvements



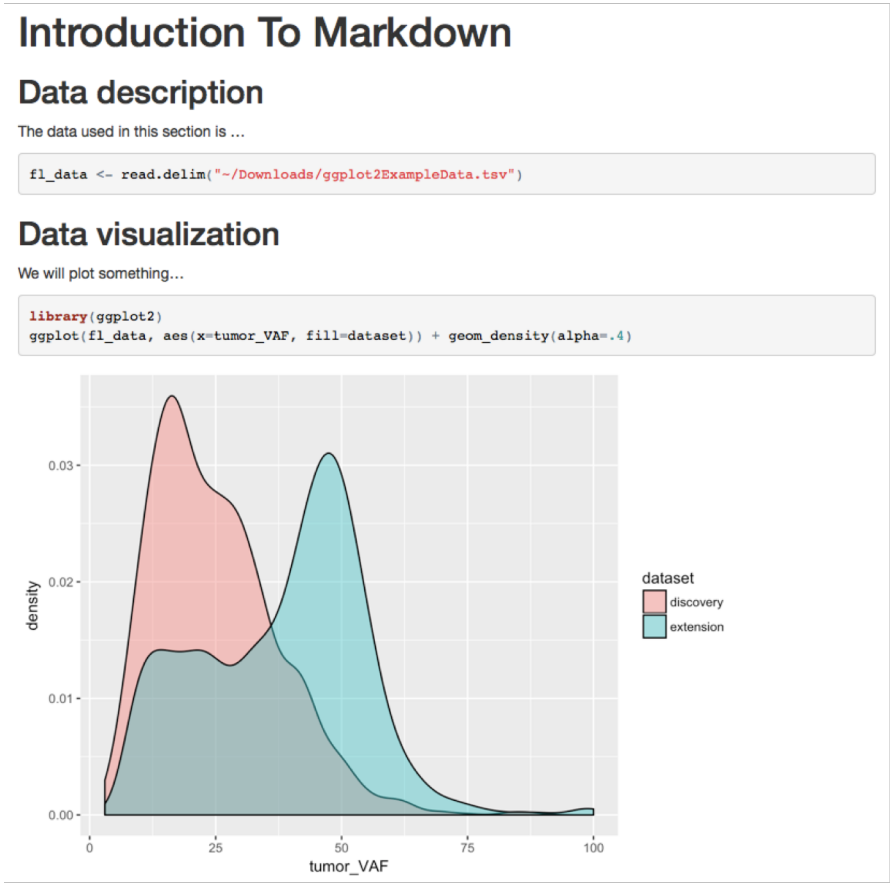
Wide vs long format



Rmarkdown can be used to create publication ready documents presenting all code (analysis), comments, and results

myRprogram.Rmd → knitr → result.html

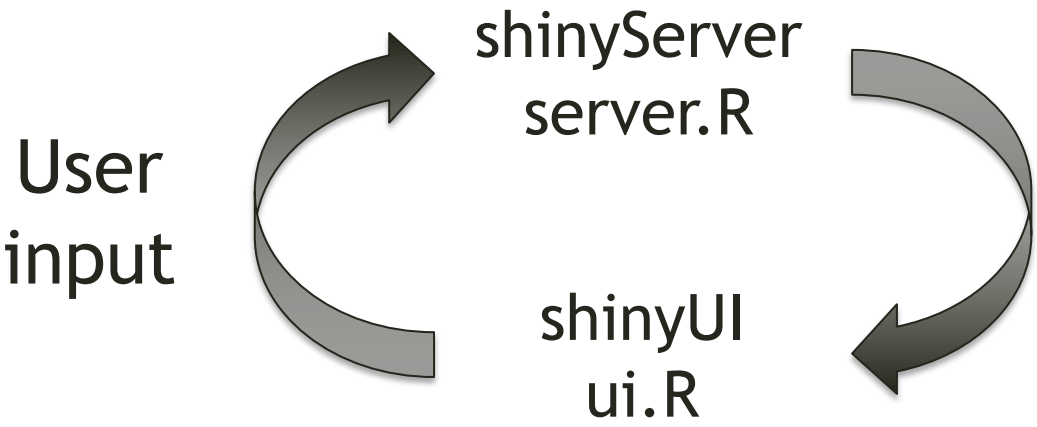
````
---
title: "Introduction To Markdown"
output: html_document
---

  ```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## Data description
The data used in this section is ...
```{r}
fl_data <-
read.delim("~/Downloads/ggplot2ExampleData.tsv")
```

## Data visualization
We will plot something...
```{r}
library(ggplot2)
ggplot(fl_data, aes(x=tumor_VAF, fill=dataset)) +
geom_density(alpha=.4)
```
````



Interactive graphics with R shiny

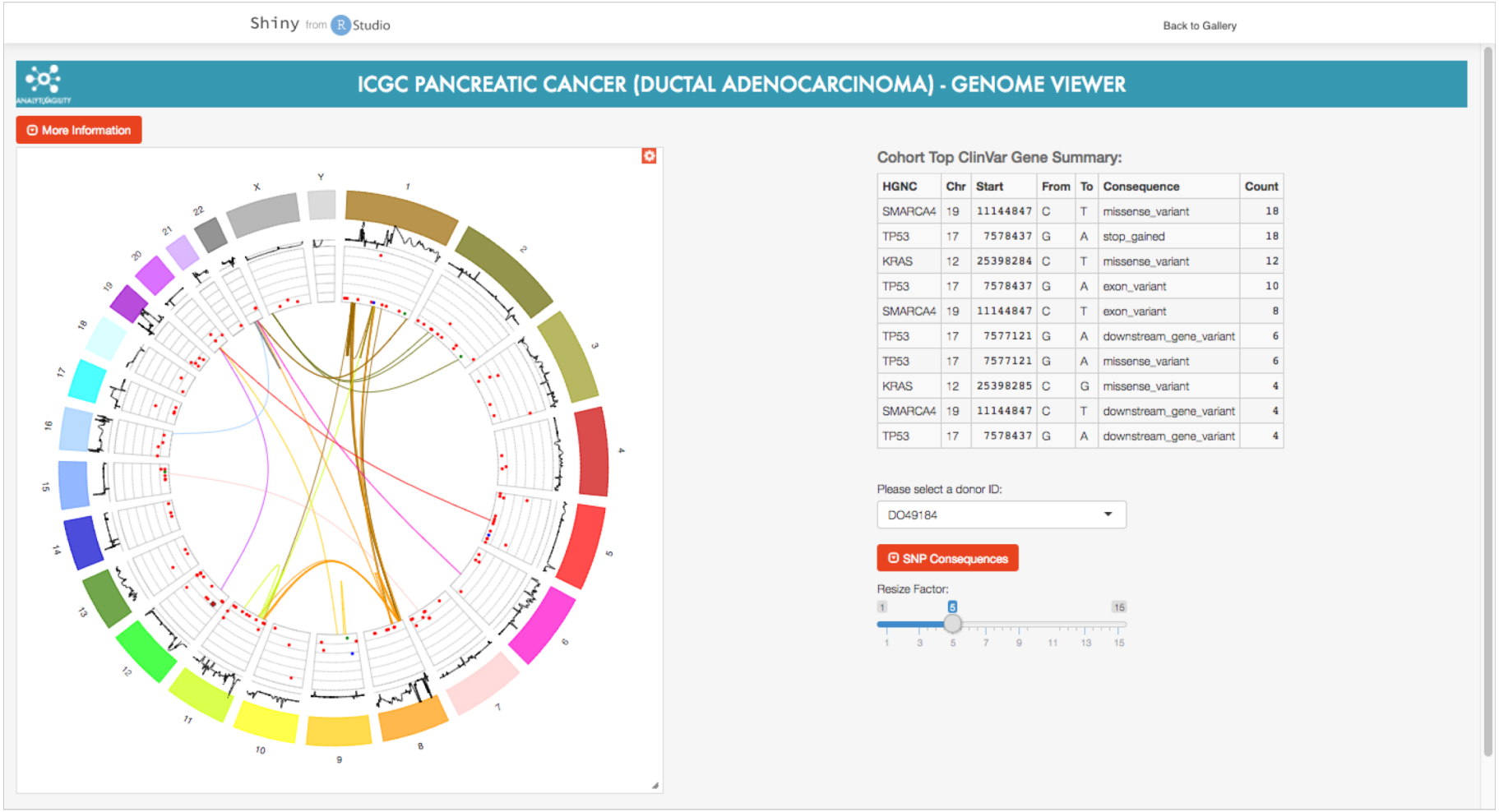
- Optimizing a graphic often requires multiple iterative alterations
- Analysis and interpretation often benefits from active filtering, variable selection, and parameterization
- Interactive graphics allow end-users, especially non-experts, to more effectively explore data
- The R shiny package allows you to quickly and easily create sophisticated web-accessible interactive graphics

Basic organization of a shiny application



Interactive User Interface (UI) = website

Demo of shiny gallery genomics example



https://shiny.rstudio.com/gallery/

Questions?