Introduction to R, Day 2: “Into the Tidyverse”

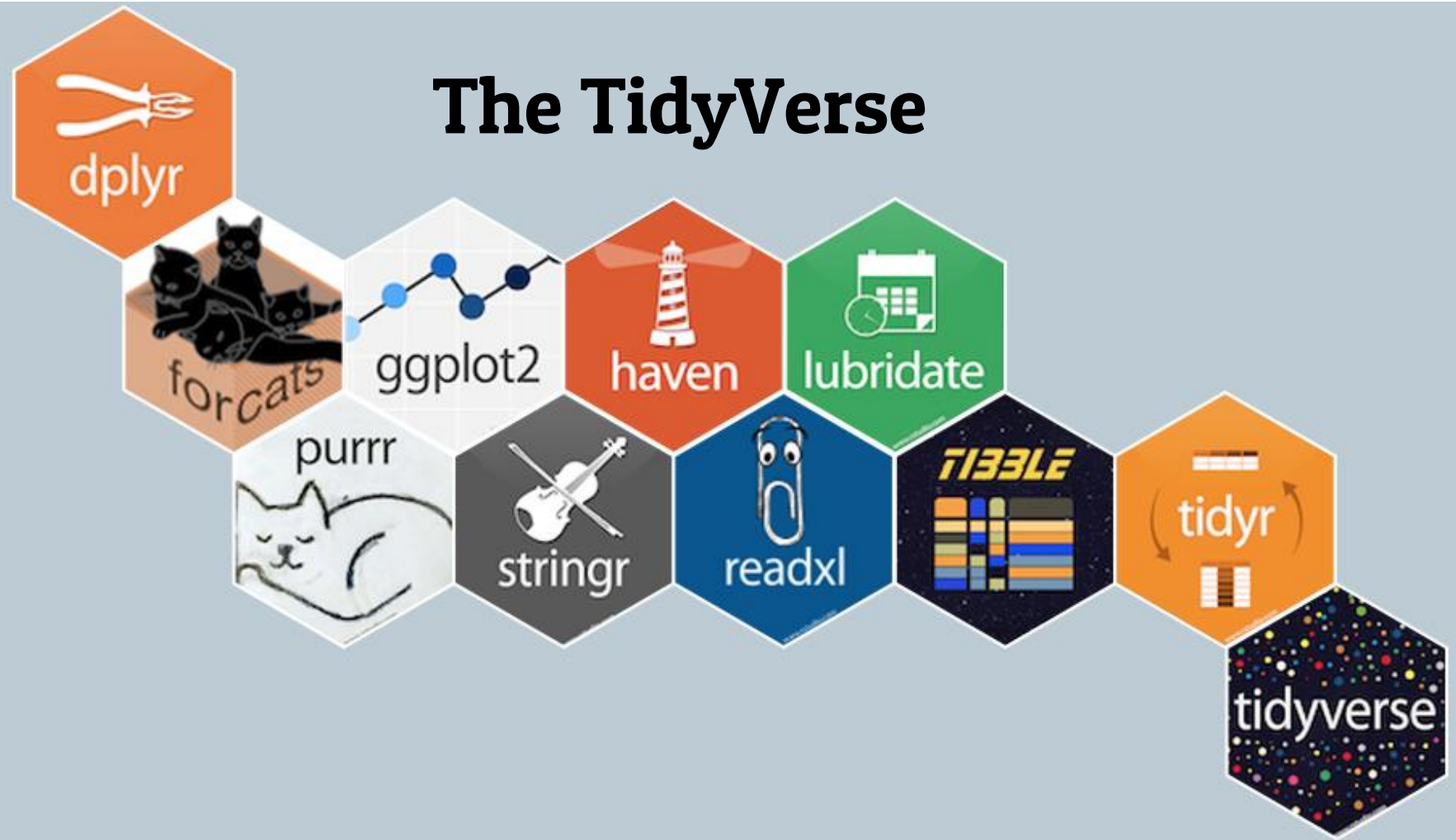
Michael Chimenti and Diana Kolbe
Dec 5 & 6 2019
Iowa Institute of Human Genetics

Slides adapted from HPC Bio at Univ. of Illinois:
https://wiki.illinois.edu/wiki/pages/viewpage.action?pageId=705021292

Learning objectives Day 2:

1. Import and work with an example dataset
2. Understand basic operations of the “tidyverse” and tibbles
3. Understand basic plotting functions of ggplot2
4. Learn how to access and work through vignettes for CRAN and Bioconductor packages
5. Be able to describe the **differences** between Tidyverse R and base R, between CRAN and Bioconductor

The TidyVerse



Why are we learning ‘tidyverse’ now?

- With tidyverse tools you can get started doing useful transformations with data immediately
- Avoid the steep learning curve of base R syntax
- Learn to think like a data scientist

Tidy data principles

1) Variables make up columns
2) Observations make up rows
3) Values go into cells

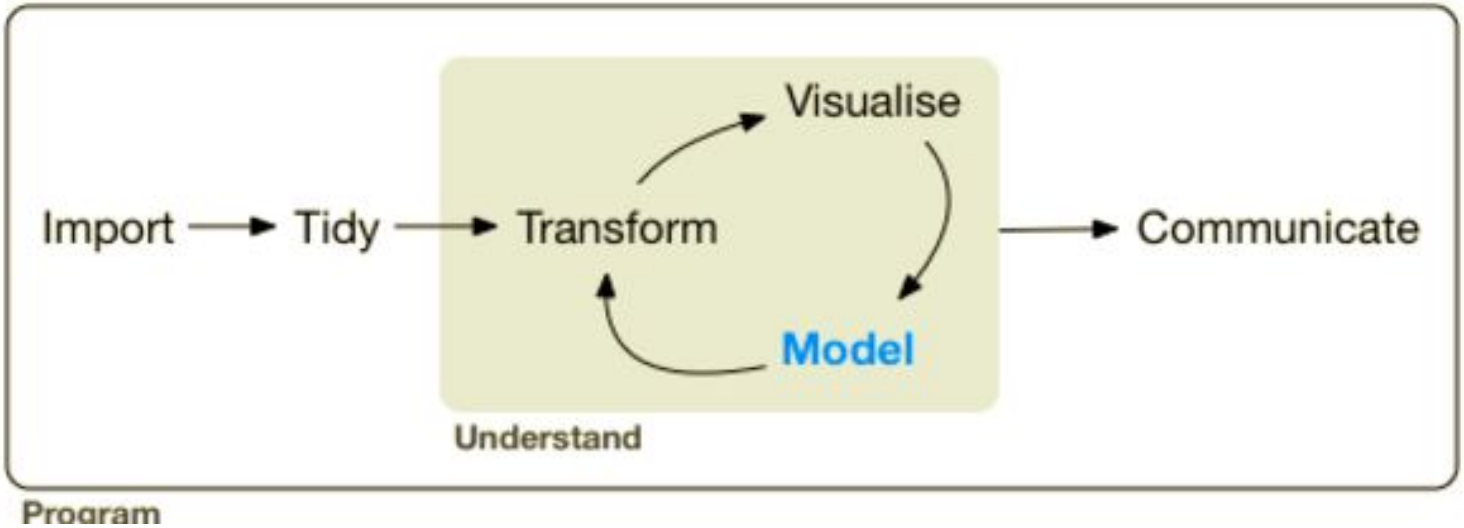
Example ‘tidy’ dataframe (actually a tibble...more on that later)

```
> iris_tib
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
 3          4.7         3.2          1.3         0.2 setosa
 4          4.6         3.1          1.5         0.2 setosa
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
# … with 140 more rows
```

Variables in columns

Observations in rows

Model for tidy data science



Importing data into R

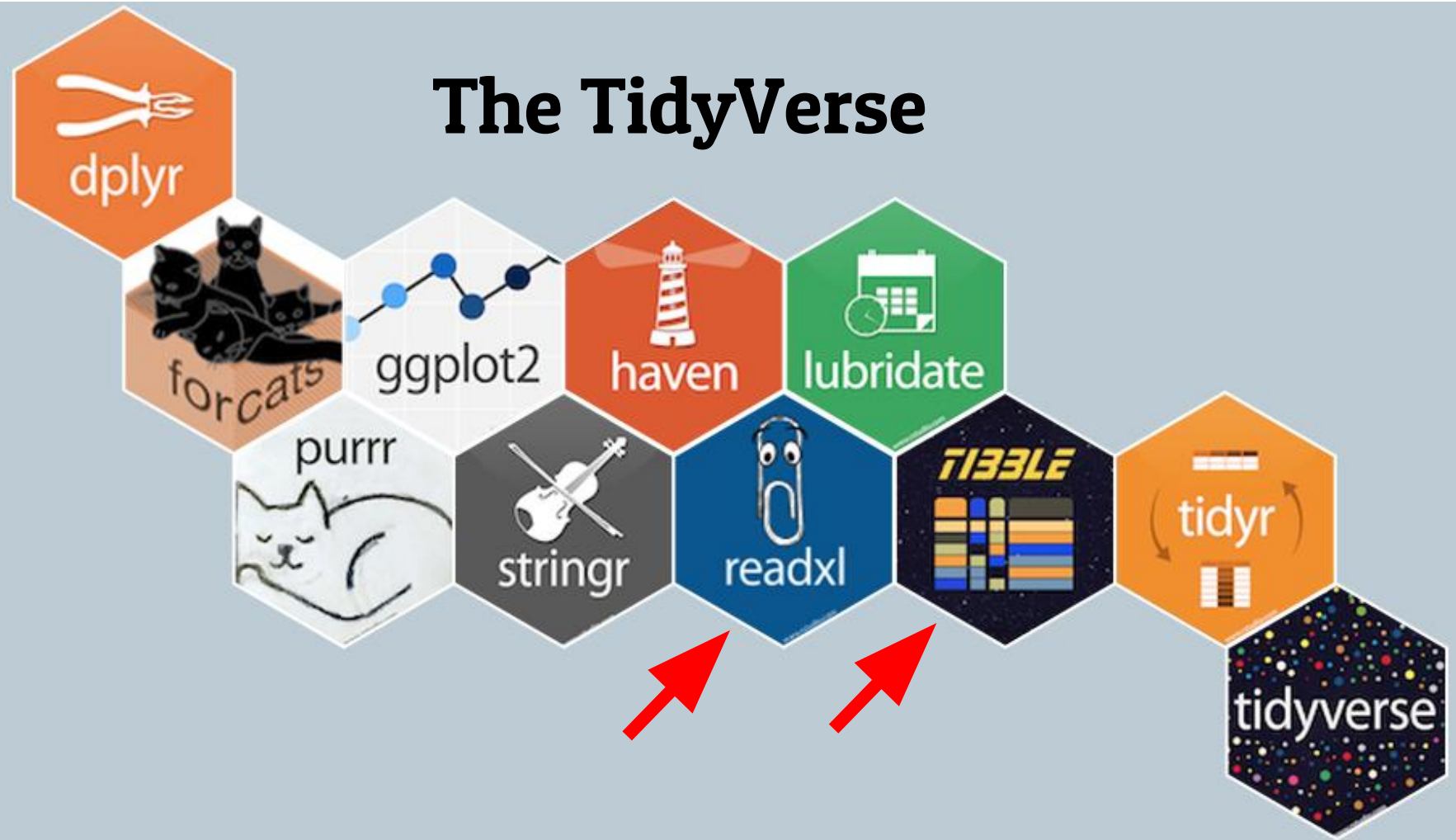
Use *read.csv()* or *read.table()* to import your spreadsheets from comma- or tab-separated text files

```
> read.csv(file = "~/my_files/my_table.csv", header = TRUE, sep = ",")
```

The tidyverse also contains *read_excel()* which can read excel files:

```
> read_excel('my_excel_sheet.xlsx')
```

The TidyVerse



Available datasets within R to play with

Type:

> data()

```
Data sets in package 'datasets':

AirPassengers         Monthly Airline Passenger Numbers
                      1949-1960
BJsales               Sales Data with Leading Indicator
BJsales.lead (BJsales)
                      Sales Data with Leading Indicator
BOD                   Biochemical Oxygen Demand
CO2                   Carbon Dioxide Uptake in Grass Plants
ChickWeight           Weight versus age of chicks on different
                      diets
DNase                 Elisa assay of DNase
EuStockMarkets        Daily Closing Prices of Major European
                      Stock Indices, 1991-1998
Formaldehyde          Determination of Formaldehyde
HairEyeColor          Hair and Eye Color of Statistics Students
Harman23.cor          Harman Example 2.3
Harman74.cor          Harman Example 7.4
Indometh              Pharmacokinetics of Indomethacin
InsectSprays          Effectiveness of Insect Sprays
JohnsonJohnson        Quarterly Earnings per Johnson & Johnson
                      Share
LakeHuron             Level of Lake Huron 1875-1972
LifeCycleSavings      Intercountry Life-Cycle Savings Data
Loblolly              Growth of Loblolly pine trees
Nile                  Flow of the River Nile
Orange                Growth of Orange Trees
OrchardSprays         Potency of Orchard Sprays
PlantGrowth           Results from an Experiment on Plant
                      Growth
```

Today we will use an internal R dataset

Type:

> data("iris")

Then,

> head(iris)

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Type:

> str(iris)

```
> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1
1 1 1 ...
```

Dataframe (base R) to tibble (tidyR)

Type:

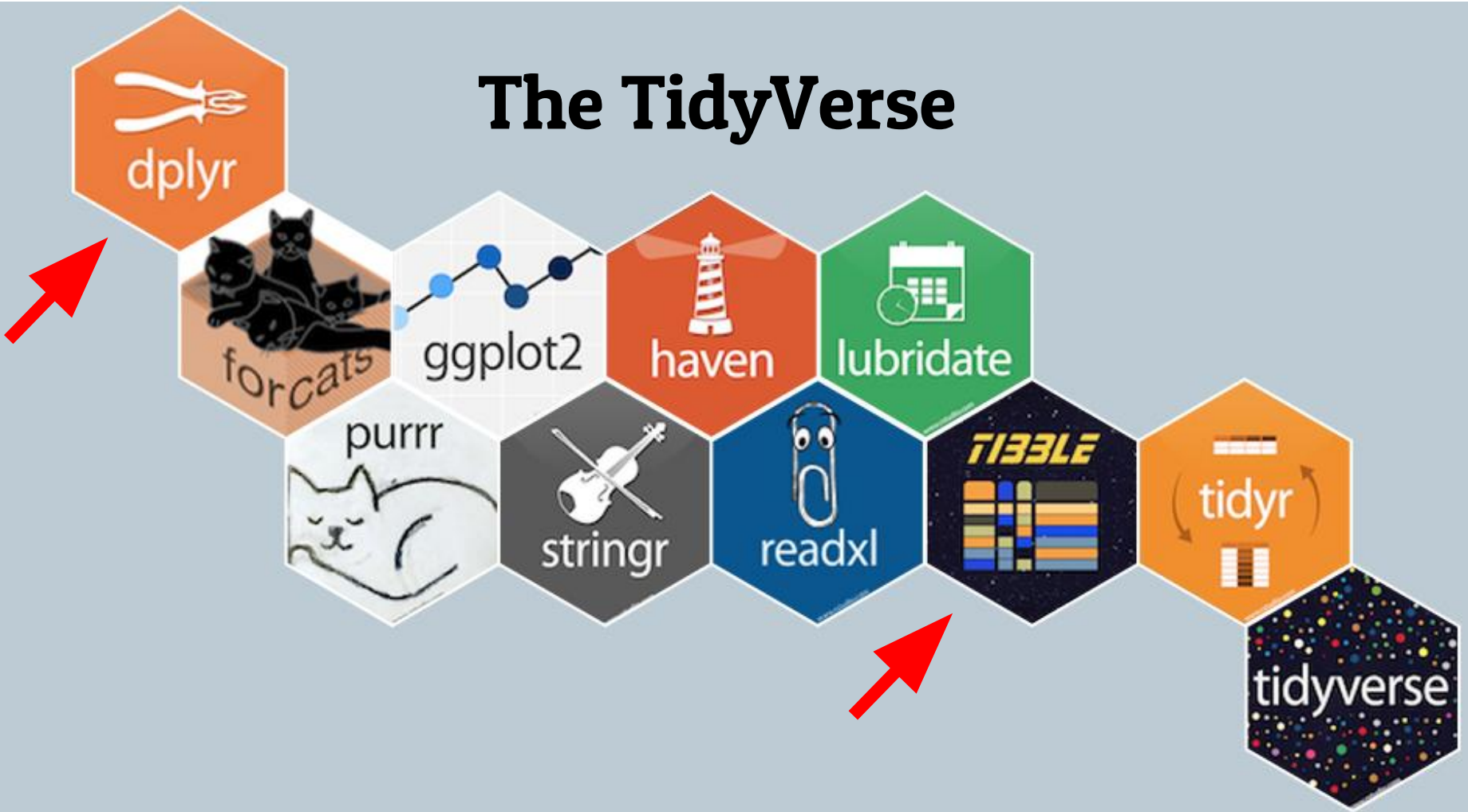
```
> library(tidyverse)
> iris_tib <- as_tibble(iris)
```

Then:

```
> iris
> iris_tib
```

Notice the difference in the output?

The TidyVerse



A tibble is a special tidyverse dataframe

Structure

Column names

Data type

```
> iris_tib
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
 3          4.7         3.2          1.3         0.2 setosa
 4          4.6         3.1          1.5         0.2 setosa
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
# … with 140 more rows
```

Type:

> summary(iris_tib)

```
> summary(iris_tib)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
       Species
 setosa    :50
 versicolor:50
 virginica :50
```

Summary is a very useful “Base R” function

First ‘dplyr’ operation: select rows with *filter()*

Type:

> filter(iris_tib, Species == ”virginica”)

```
# A tibble: 50 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          6.3         3.3          6           2.5 virginica
 2          5.8         2.7          5.1         1.9 virginica
 3          7.1         3            5.9         2.1 virginica
 4          6.3         2.9          5.6         1.8 virginica
 5          6.5         3            5.8         2.2 virginica
 6          7.6         3            6.6         2.1 virginica
 7          4.9         2.5          4.5         1.7 virginica
 8          7.3         2.9          6.3         1.8 virginica
 9          6.7         2.5          5.8         1.8 virginica
10          7.2         3.6          6.1         2.5 virginica
# … with 40 more rows
```

Assign the results to a new variable

Type:

```
> iris_tib_vir <- filter(iris_tib, Species == "virginica")
> iris_tib_vir
```

```
# A tibble: 50 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          6.3         3.3          6           2.5 virginica
 2          5.8         2.7          5.1         1.9 virginica
 3          7.1         3            5.9         2.1 virginica
 4          6.3         2.9          5.6         1.8 virginica
 5          6.5         3            5.8         2.2 virginica
 6          7.6         3            6.6         2.1 virginica
 7          4.9         2.5          4.5         1.7 virginica
 8          7.3         2.9          6.3         1.8 virginica
 9          6.7         2.5          5.8         1.8 virginica
10          7.2         3.6          6.1         2.5 virginica
# … with 40 more rows
```

‘Arrange’ a tibble to sort on values

Type:

> arrange(iris_tib, Sepal.Length)

> arrange(iris_tib, desc(Petal.Width))

```
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          6.3         3.3          6           2.5 virginica
 2          7.2         3.6          6.1         2.5 virginica
 3          6.7         3.3          5.7         2.5 virginica
 4          5.8         2.8          5.1         2.4 virginica
 5          6.3         3.4          5.6         2.4 virginica
 6          6.7         3.1          5.6         2.4 virginica
 7          6.4         3.2          5.3         2.3 virginica
 8          7.7         2.6          6.9         2.3 virginica
 9          6.9         3.2          5.7         2.3 virginica
10          7.7         3            6.1         2.3 virginica
# … with 140 more rows
```

dplyr ‘select()’ to subset and rename columns

```
> select(iris_tib, Species)

> select(iris_tib, -Species)

> select(iris_tib, c(Species, Petal.Length))

> select(iris_tib, Sp=Species, PL=Petal.Length)
```

```
# A tibble: 150 x 2
   Sp        PL
   <fct>  <dbl>
 1 setosa   1.4
 2 setosa   1.4
 3 setosa   1.3
```

Add new columns with dplyr 'mutate()'

> mutate(iris_tib, Petal.Length.Mean = mean(Petal.Length))

```
# A tibble: 150 x 6
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Petal.Length.Mean
          <dbl>       <dbl>        <dbl>       <dbl> <fct>               <dbl>
 1          5.1         3.5          1.4         0.2 setosa               3.76
 2          4.9         3            1.4         0.2 setosa               3.76
 3          4.7         3.2          1.3         0.2 setosa               3.76
 4          4.6         3.1          1.5         0.2 setosa               3.76
 5          5           3.6          1.4         0.2 setosa               3.76
 6          5.4         3.9          1.7         0.4 setosa               3.76
 7          4.6         3.4          1.4         0.3 setosa               3.76
 8          5           3.4          1.5         0.2 setosa               3.76
 9          4.4         2.9          1.4         0.2 setosa               3.76
10          4.9         3.1          1.5         0.1 setosa               3.76
# … with 140 more rows
```

Add new columns with dplyr ‘mutate()’

> mutate(iris_tib, Sepal.Area = Sepal.Width * Sepal.Length)

```
# A tibble: 150 x 6
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Area
          <dbl>       <dbl>        <dbl>       <dbl> <fct>        <dbl>
 1          5.1         3.5          1.4         0.2 setosa        17.8
 2          4.9         3            1.4         0.2 setosa        14.7
 3          4.7         3.2          1.3         0.2 setosa        15.0
 4          4.6         3.1          1.5         0.2 setosa        14.3
 5          5           3.6          1.4         0.2 setosa        18
 6          5.4         3.9          1.7         0.4 setosa        21.1
 7          4.6         3.4          1.4         0.3 setosa        15.6
 8          5           3.4          1.5         0.2 setosa        17
 9          4.4         2.9          1.4         0.2 setosa        12.8
10          4.9         3.1          1.5         0.1 setosa        15.2
# … with 140 more rows
```

The “split-apply-combine” data science paradigm

SPLIT **APPLY** **COMBINE**

name	age	sex
John	13	Male
Mary	15	Female
Alice	14	Female
Peter	13	Male
Roger	14	Male
Phyllis	13	Female

.(sex)

name	age	sex
John	13	*Male*
Peter	13	*Male*
Roger	14	*Male*

name	age	sex
Mary	15	*Female*
Alice	14	*Female*
Phyllis	13	*Female*

.(age)

name	age	sex
John	*13*	Male
Peter	*13*	Male
Phyllis	*13*	Female

name	age	sex
Alice	*14*	Female
Roger	*14*	Male

name	age	sex
Mary	*15*	Female

Calculate groupwise summary stats

New object w/ Summarized Groupwise Stats

Figure 4: Two examples of splitting up a data frame by variables. If the data frame was split up by both sex and age, there would only be one subset with more than one row: 13-year-old males.

Introducing the dplyr “pipe” operator

In order to “split-apply-combine” we need a tool to chain operations together:

```
log(x) %>% plot()
```

This operator is called a ‘pipe’, and basically says do the thing on the left and pass the result to the thing on the right…

Here are four reasons why you should be using pipes in R:

- You'll structure the sequence of your data operations from left to right, as opposed to from inside and out;
- You'll avoid nested function calls;
- You'll minimize the need for local variables and function definitions; And
- You'll make it easy to add steps anywhere in the sequence of operations.

https://www.datacamp.com/community/tutorials/pipe-r-tutorial

So let's first group a tibble with 'group_by':

Type:

```
> iris_gr <- group_by(iris, Species)
```

Read this as: group 'iris' by species, assign result to "iris_gr"

What type of object is "iris_gr"?

Let's 'split-apply-combine' with a pipe:

Type:

```
> iris_gr_mean <- group_by(iris, Species) %>%
mutate(meanwidth=mean(Petal.Width))
```

Read this as: group 'iris' by species, pass groups to mutate, calculate groupwise mean petal width, assign result to "iris_gr_mean"

> iris_gr_mean

```
# A tibble: 150 x 6
# Groups:   Species [3]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species meanwidth
          <dbl>       <dbl>        <dbl>       <dbl> <fct>       <dbl>
 1          5.1         3.5          1.4         0.2 setosa      0.246
 2          4.9         3            1.4         0.2 setosa      0.246
 3          4.7         3.2          1.3         0.2 setosa      0.246
 4          4.6         3.1          1.5         0.2 setosa      0.246
 5          5           3.6          1.4         0.2 setosa      0.246
 6          5.4         3.9          1.7         0.4 setosa      0.246
 7          4.6         3.4          1.4         0.3 setosa      0.246
 8          5           3.4          1.5         0.2 setosa      0.246
 9          4.4         2.9          1.4         0.2 setosa      0.246
10          4.9         3.1          1.5         0.1 setosa      0.246
# … with 140 more rows
```

Let's count the rows in each group

Type:

```
> summarize(iris_gr_mean, count = n())
```

```
# A tibble: 3 x 2
  Species    count
  <fct>      <int>
1 setosa        50
2 versicolor    50
3 virginica     50
>
```

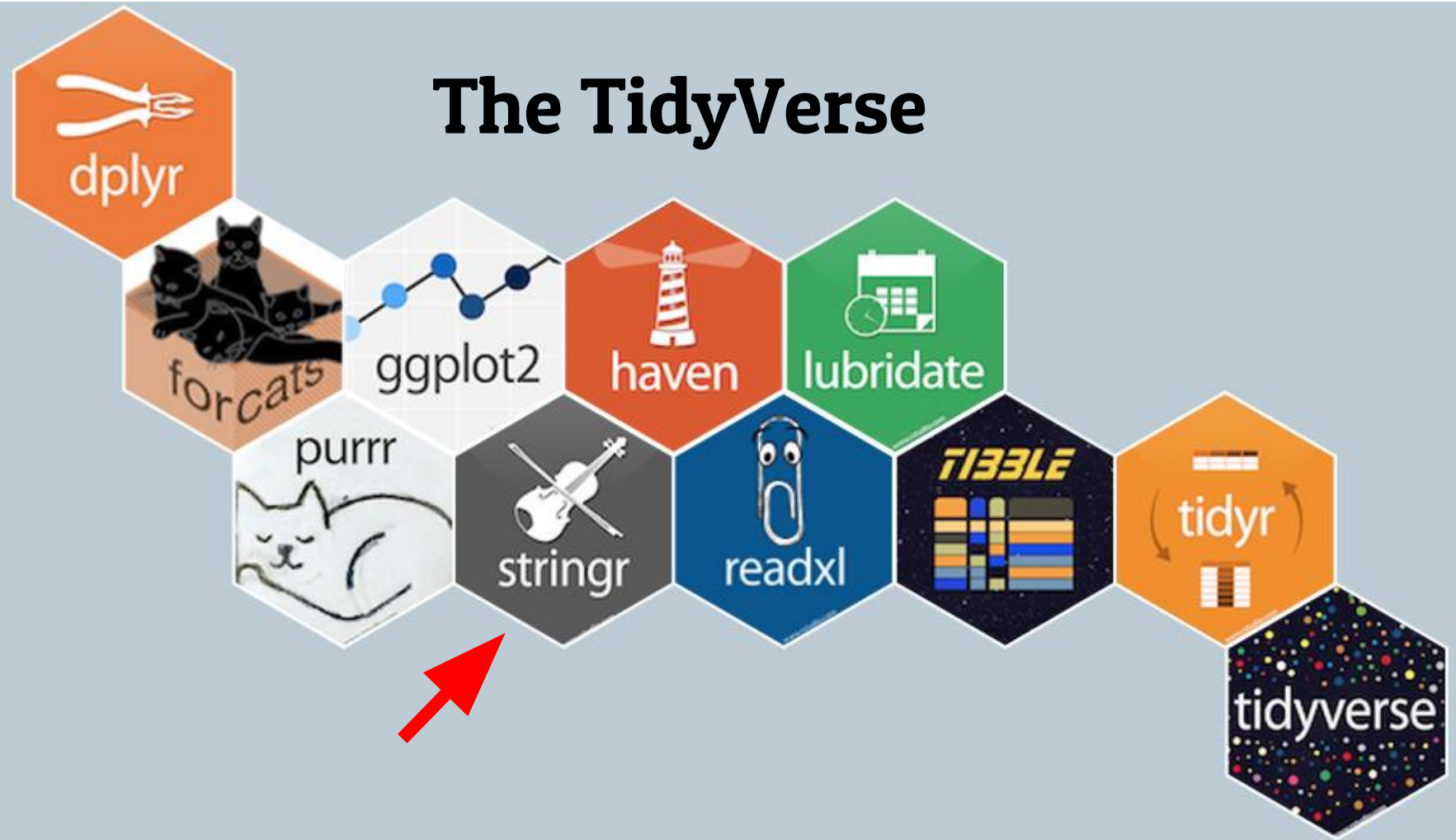
Now we calculate other groupwise values:

Type:
> summarize(iris_gr_mean, max_Sep_Len = max(Sepal.Length))

```
# A tibble: 3 x 2
  Species    max_Sep_Len
  <fct>            <dbl>
1 setosa             5.8
2 versicolor         7
3 virginica          7.9
>
```

What happens if you try this on ‘iris_tib’ (*i.e.*, not grouped)?

The TidyVerse



Dealing with strings with *stringr*

Type:

```
> str_to_upper(iris_tib$Species)
```

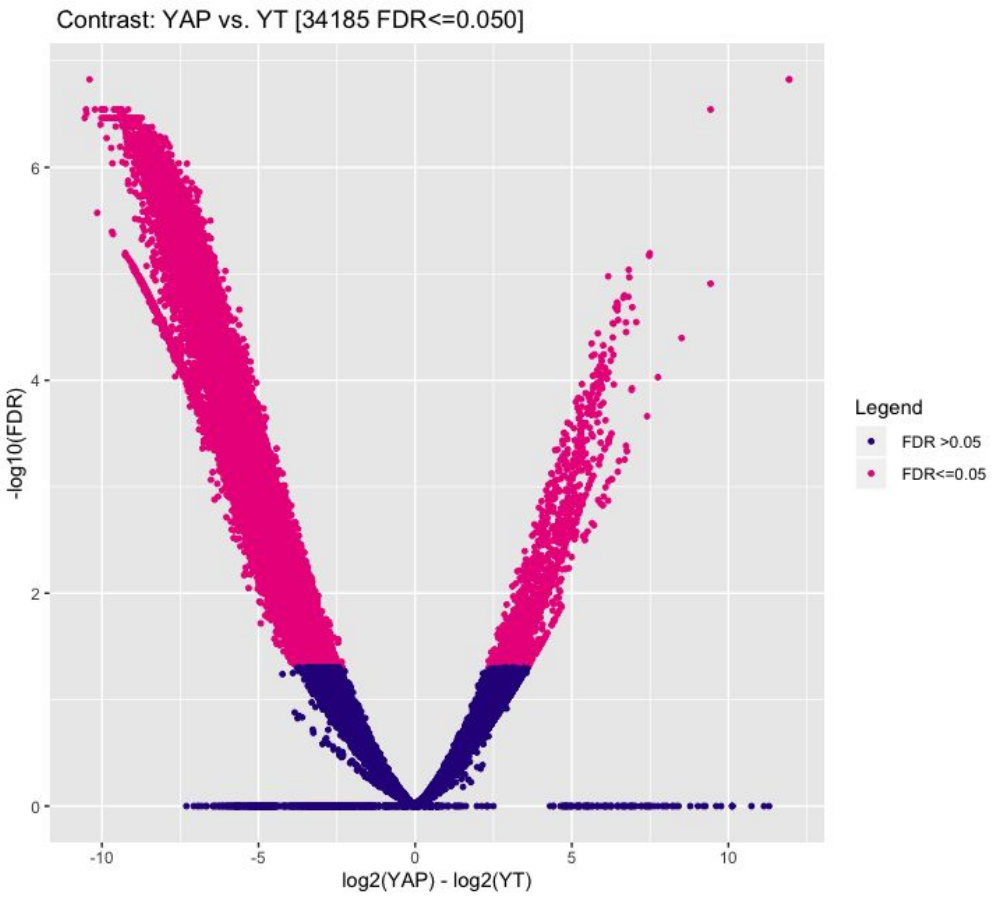
Think like a data scientist:

```
> iris_tib_UP <- iris_tib %>% mutate(Species =
str_to_upper(Species))
```

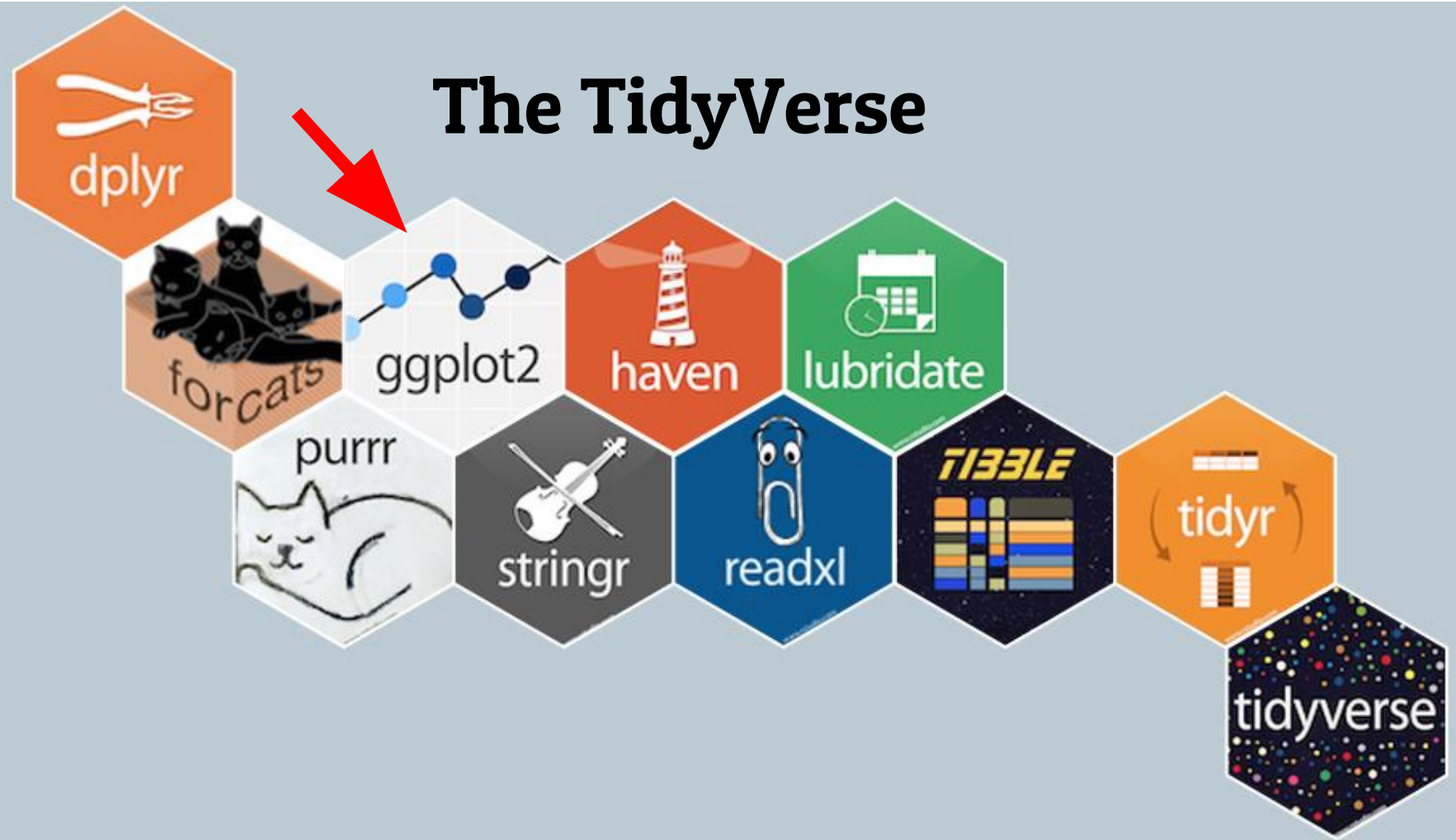
What does the new tibble look like?

We’ve learned to manipulate tibbles…

Now let’s plot some data…



The TidyVerse



‘ggplot2’ basics

Data + aesthetic + geom + options

ggplot2 operates on tibbles and dataframes

Data = a tibble or dataframe
Aesthetic = x and y columns for plotting
Geom = the type of plot
Options = other options like axis limits, legends

ggplot2 builds plots like dplyr manipulates data:

```
New_data <- my_df %>% mutate() %>% group_by()

Myplot <- ggplot(data) + aes() + geom() + theme()
```

The '+' is analogous to a pipe operator

How does this work with our tibble, ‘iris_tib’?

An example:

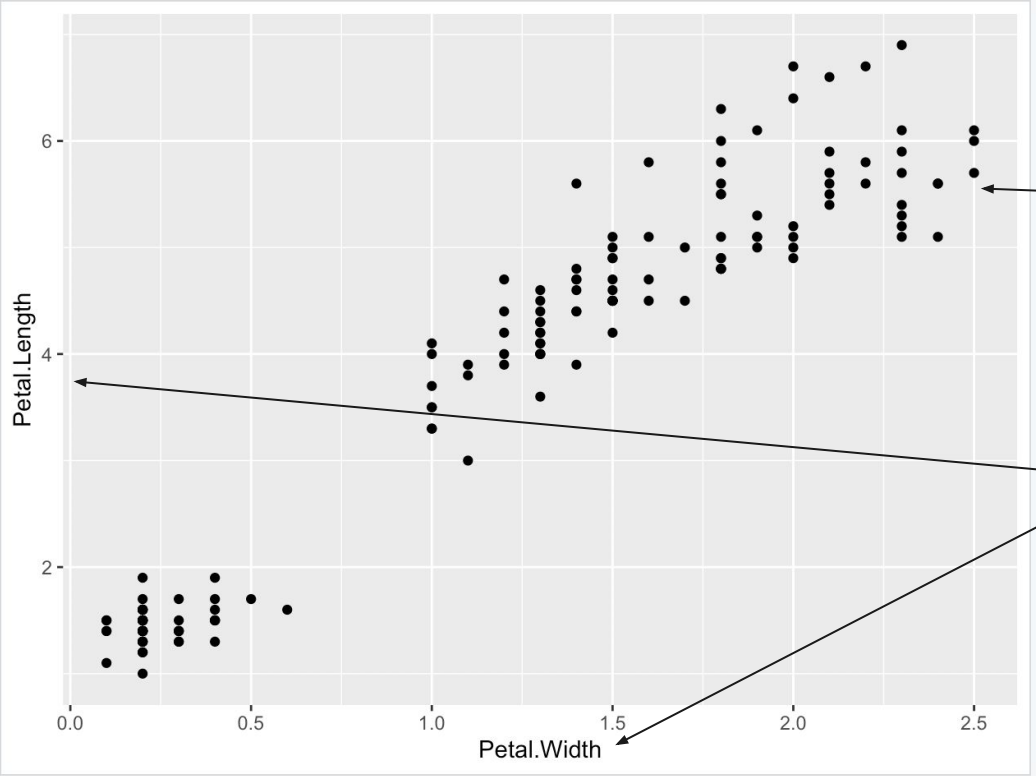
```
> library(‘ggplot2’)

> myplot <- ggplot(iris_tib, aes(x = Petal.Width, y = Petal.Length)
> myplot
```

Notice that no data is plotted yet...now add a scatter point “geom”:

```
> myplot <- myplot + geom_point()
> myplot
```

The result:



geom_point()

aes(x,y)

Building up plots

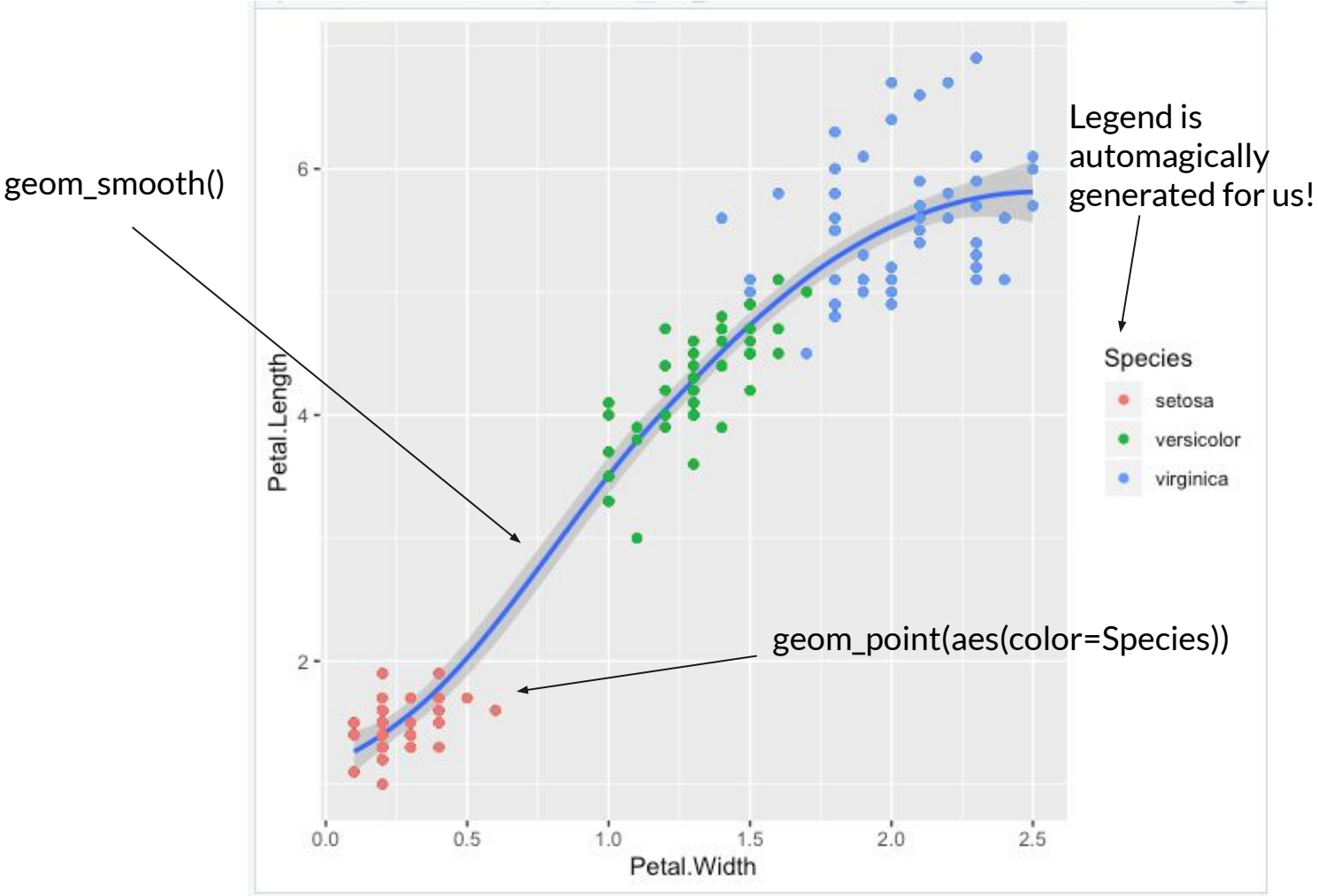
Now let's add a trendline:

```
> myplot <- myplot + geom_smooth()
> myplot
```

Let's add some colors:

```
> myplot <- myplot + geom_point(aes(color=Species))
```

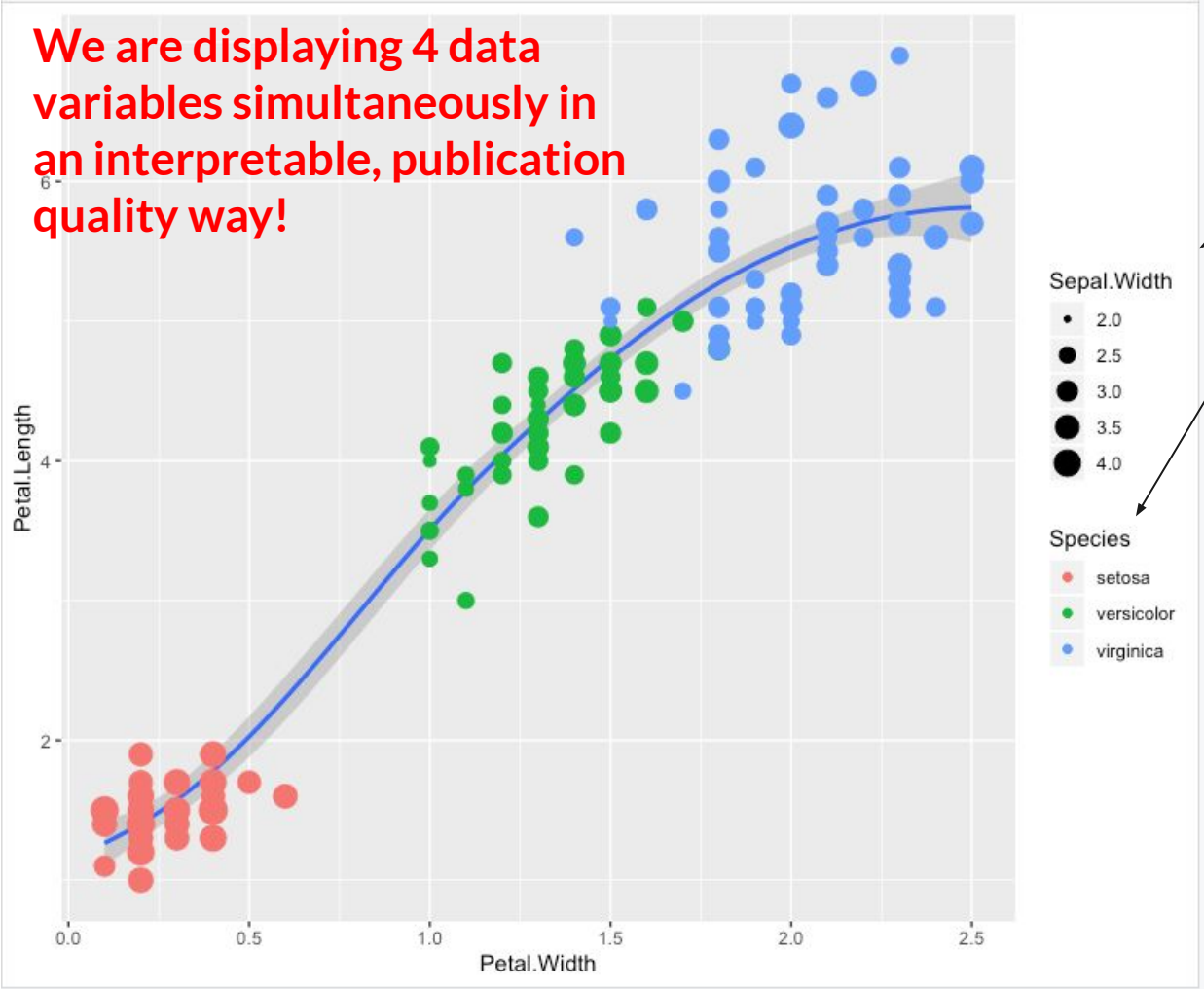
What have we done?


geom_smooth()
Legend is automagically generated for us!
geom_point(aes(color=Species))
Petal.Length
Petal.Width
6
4
2
0.0
0.5
1.0
1.5
2.0
2.5
Species
setosa
versicolor
virginica

Building up plots

```
> myplot <- ggplot(iris_tib, aes(x = Petal.Width, y = Petal.Length))
> myplot <- myplot + geom_point(aes(color=Species, size =
Sepal.Width)) + geom_smooth()
```

Now what have we done?

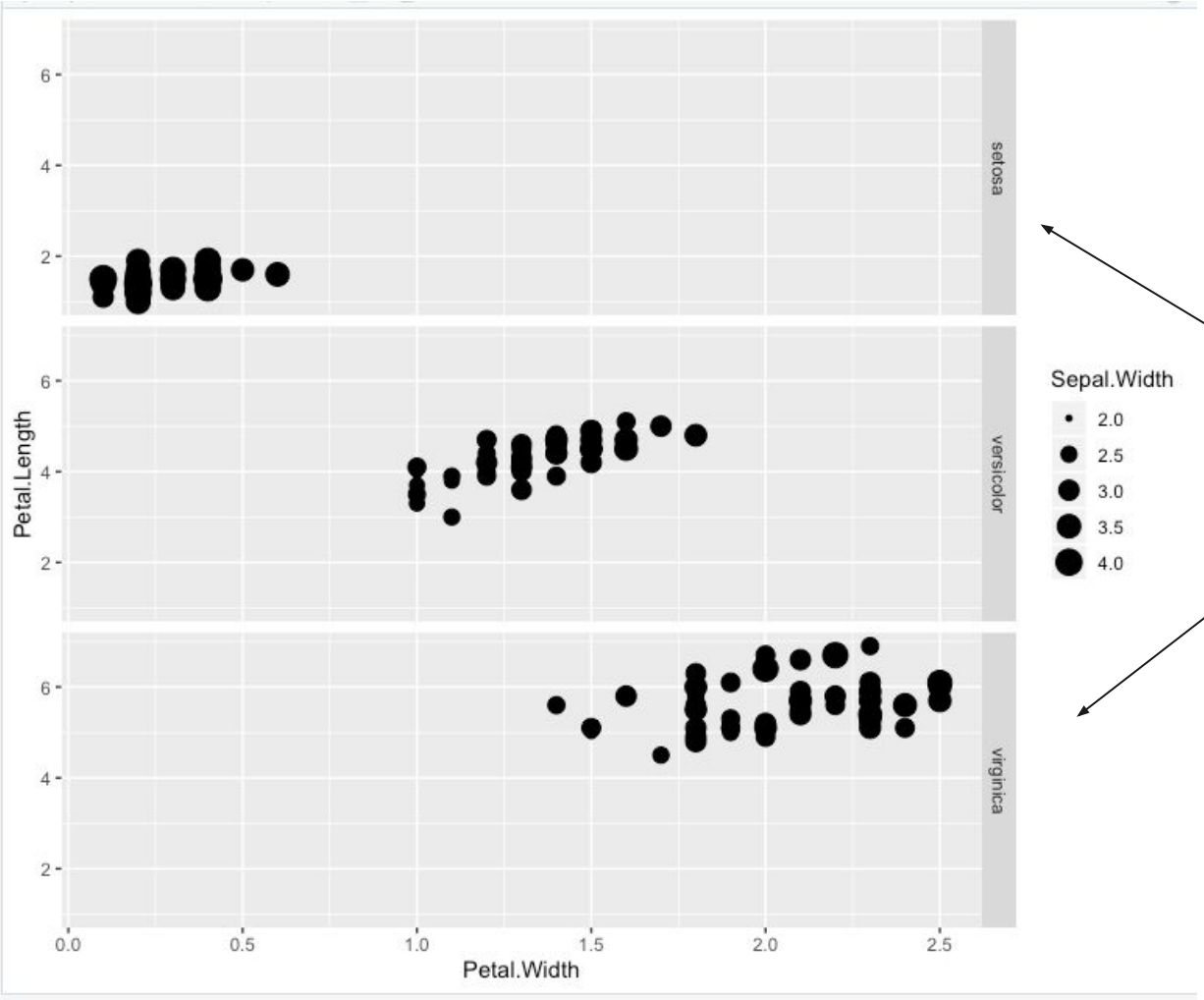


Two legends now

Too many variables? Facet them out!

```
> myplot <- ggplot(iris_tib, aes(x = Petal.Width, y = Petal.Length))
> myplot <- myplot + geom_point(aes(size = Sepal.Width)) +
facet_grid("Species")
```

A Facet Grid builds multiple plots from factors in your dataframe



"Facet" plots on species

What is Bioconductor?

www.bioconductor.org

- "… open source, open development software project to provide tools for the analysis and comprehension of **high-throughput genomic data**"
- Primarily based on R language (functions can be in other languages), and run in R environment
- Current release consists of 1741 **software packages** (sets of functions) for specific tasks
- Also maintains 948 **annotation packages** for many commercial arrays and model organisms plus 371 **experiment data packages** and 27 **workflow packages**

More background on Bioconductor

http://bioconductor.org/about/

- Overseen by a core team, mostly located at the Roswell Park Cancer Institute in Buffalo, NY*
  - Provide infrastructure and access to packages
  - Include metadata, annotation and data sets
  - Develop/extend a common software platform to provide **interoperability between packages**
  - Provide documentation and training
- But majority of software packages contributed by users
  - Any package that is related to genomic data and passes BioC's checks is accepted
  - BioC enforces more rigorous standards than CRAN

Reference manuals vs. vignettes

Reference manual: list of all functions in a package with explanations of their arguments (e.g., all help pages together alphabetically in a pdf)

Vignette: Explanation of how to use the functions in a typical analysis in start-to-finish order

Both CRAN and BioC require reference manuals, but only BioC (mostly) requires vignettes!

Navigating Bioconductor

Navigate your browser to http://bioconductor.org/packages/release/BiocViews.html

- BiocViews – allows partitioning of packages by categories

Take a few minutes to investigate the different software packages. Can you find the names of 2 or more packages that might be useful for your research?

How many packages are linked to ChIPSeq (as of Nov 2019)?

A. 30

B. 62

C. 89

D. 184

Navigating Bioconductor's annotation data

Go to: http://bioconductor.org/packages/release/BiocViews.html

- Annotation packages also partitioned by categories
- Under PackageType:
  - BSgenome - genome sequences
  - OrgDb - gene annotation packages for species

Does your research organism have any packages in BioC?

A. Yes

B. No

Where to find other “non-model” organisms

See software packages:

1. AnnotationForge - build your own org.*.db package
2. AnnotationHub - access to resources for > 1000 "less-model organisms" from the following databases:

Ensembl, EncodeDCC, UCSC, Inparanoid8, NHLBI, ChEA, Pazar, NIH Pathway Interaction Database, RefNet, Haemcode, GEO, BroadInstitute, dbSNP, ftp://ftp.ncbi.nlm.nih.gov/gene/DATA , PRIDE, Gencode

Highly recommend doing AnnotationHub's How To vignette

Bioconductor package install method

- Used to have biocLite() for R < 3.5.0
- Now have submitted BiocManager package to CRAN:
  - install.packages("BiocManager")
- Then can install *any* package via:
  - BiocManager::install("limma")
- BioC's version also automatically checks for package updates for any installed package
- *Please do not update packages if you already loaded any via* library()

How to access vignettes and example code

> browseVignettes(package = “Biostrings”)

Takes you a local HTML webpage for this package:

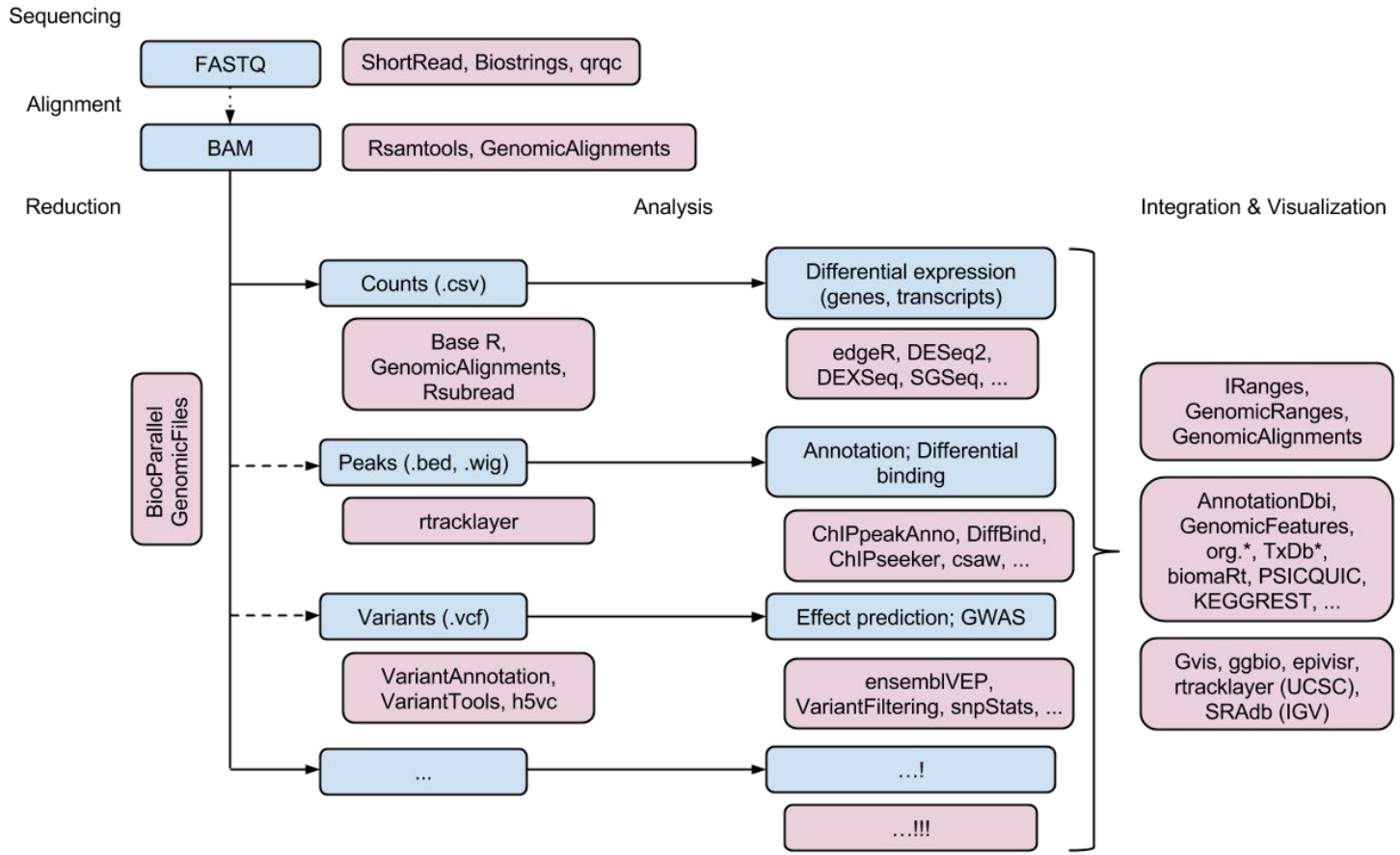
Vignettes found by “browseVignettes("Biostrings")”

Vignettes in package Biostrings

- A short presentation of the basic classes defined in Biostrings 2 - PDF source R code
- Biostrings Quick Overview - PDF source
- Handling probe sequence information - PDF source R code
- Multiple Alignments - PDF source R code
- Pairwise Sequence Alignments - PDF source R code

Try this now ^^^

Typical tasks and Bioconductor/R packages



From:
http://www.bioconductor.org/help/course-materials/2015/CSAMA2015/lab/L1.2-bioc-intro-morgan.html

Additional Help

- BioC provides some example workflows for analyzing different types of genomic data.
  - Pick a workflow that is of the most interest to you. Write down some of the packages they suggest using.
- BioC runs various training courses and also make the training materials available on the web.
  - Search for CSAMA 2019
  - BioC2019 materials are also very useful
- Community-supplied resources and tutorials
- F1000 Research Bioconductor Channel https://f1000research.com/channels/bioconductor

Base R Swirl lessons

type :

> library(‘swirl’)

>swirl()

Work lessons 7 through 10