What is the Tidyverse

The tidyverse is a collection of libraries for R, created by the same people, with a unified vision.

They are *extremely* useful for loading, transforming, exploring, and visualizing datasets.

Ok, what are libraries?

Libraries are packages.

In R, most public packages live in the CRAN repository.

Packages are installed with a package manager.

R comes with a built-in package manager.

Packages in R

R's default package manager installs all packages *globally* on your computer. This makes working in teams difficult:

- What if you have a different version of some library than I do?
- What if I want one version of xyz-package for one project, and a different version for another?
- If I upgrade xyz-package to use the latest features in one project, will my old project stop working?

Reproducible Environments

Some solutions:

- Use Docker (containerized environments, for example, with Jupyter Notebooks).
- Use Packrat/Jetpack (R libraries for reproducible environments).

Dataframe

Before we enter the tidyverse, we should review the dataframe:

A dataframe is a list, where each element in the list is a vector of equal length.

Dataframe

Dataframes have columns and rows.

```
iris
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width     Sp
1            5.1         3.5          1.4         0.2      s
2            4.9         3.0          1.4         0.2      s
3            4.7         3.2          1.3         0.2      s
4            4.6         3.1          1.5         0.2      s
5            5.0         3.6          1.4         0.2      s
6            5.4         3.9          1.7         0.4      s
7            4.6         3.4          1.4         0.3      s
...
```

Dataframe

Just like a list, subsetting is done with []:

```
> iris[c("Sepal.Width", "Species")]

    Sepal.Width    Species
1           3.5     setosa
2           3.0     setosa
3           3.2     setosa
4           3.1     setosa
5           3.6     setosa
6           3.9     setosa
```

Dataframe

```
iris[1:5,]
  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
```

Dataframe

And columns are access with $ or [[]], which returns a vector:

```
> iris$Species

  [1] setosa     setosa     setosa     setosa     setosa
  [7] setosa     setosa     setosa     setosa     setosa
 [13] setosa     setosa     setosa     setosa     setosa
 [19] setosa     setosa     setosa     setosa     setosa
 [25] setosa     setosa     setosa     setosa     setosa
 [31] setosa     setosa     setosa     setosa     setosa
 ...
```

Tibbles

A tibble is a lot like a dataframe:

```
> library(tibble)
> tibble::as_tibble(iris)

# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Specie
          <dbl>       <dbl>        <dbl>       <dbl>  <fct
 1          5.1         3.5          1.4         0.2  setos
 2          4.9         3.0          1.4         0.2  setos
 3          4.7         3.2          1.3         0.2  setos
 4          4.6         3.1          1.5         0.2  setos
 5          5.0         3.6          1.4         0.2  setos
 6          5.4         3.9          1.7         0.4  setos
 7          4.6         3.4          1.4         0.3  setos
 8          5.0         3.4          1.5         0.2  setos
 9          4.4         2.9          1.4         0.2  setos
10          4.9         3.1          1.5         0.1  setos
```

Tibbles

But:

- Tibbles do not print all the rows!
- Tibbles print out the type of each column.
- Tibbles do not coerce types (dataframes will often coerce strings into factors, for example. bConstructing data as a tibble will avoid that headache).
- Subsetting a tibble (with []) will always return a tibble. Never a vector.

Tibbles

It's nicer to work with tibbles.

Readr

Reading csv's can be done with:

```
read.csv('path/to/myfile.csv')
```

But the Tidyverse gives us:

```
library(readr)
read_csv('path/to/myfile.csv')
```

Which will both tell us what it's doing, and return a tibble.

Dplyr and Tidyr

Now that we've read our data, we will need to reformat it.

Dplyr and Tidyr are libraries you will use *extensively* for that.

Dplyr

Dplyr consists of a bunch of functions:

```
filter()
arrange()
select()
rename()
mutate()
group_by()
summarise()
```

Dplyr

Each function in the Dplyr library returns a dataframe (or tibble)

They *do not* mutate the original variable.

Dplyr

Because of this, you must either save the output of the functions to a variable, or chain the functions together with the "pipe" operator:

```
iris %>%
    filter(Sepal.Width > 3.1) %>%
    group_by(Species) %>%
    summarise()
```

Dplyr

Dplyr does one thing very differently from other libraries in R. It allows you to write variable names that don't exist in your current environment, as long as they are column names on your dataframe!

```
filter(iris, Sepal.Width > 3.1)
```

Here, Sepal.Width does not exist, but we are *not* putting it in quotes. It is not a string, it is a variable reference!