

# Introduction to the Tidyverse

Nandan Rao

## 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.

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.

Dataframes have columns and rows.

```
> iris
```

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

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

```
> iris[1:5,]  
  Sepal.Length Sepal.Width Petal.Length  
1           5.1           3.5           1.4  
2           4.9           3.0           1.4  
3           4.7           3.2           1.3  
4           4.6           3.1           1.5  
5           5.0           3.6           1.4
```

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

```
> iris$Species
```

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

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
```

```
      <dbl>         <dbl>         <dbl>
```

```
1         5.1         3.5         1.4
```

```
2         4.9         3.0         1.4
```

```
3         4.7         3.2         1.3
```

```
4         4.6         3.1         1.5
```

```
5         5.0         3.6         1.4
```

```
# ... with 145 more rows
```



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. Constructing data as a tibble will avoid that headache).
- Subsetting a tibble (with `[]`) will always return a tibble. Never a vector.

It's nicer to work with tibbles.

Reading csv's can be done in base R 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.

The Tidyverse also provides a library for reading excel files. We will use that to load some data:

```
> library(read_excel)
> read_excel('./bls_data/oesm10in4/nat3d_M2010_d1.xls')
```

```
# A tibble: 19,289 x 24
```

	NAICS	NAICS_TITLE	OCC_CODE	OCC_TITLE
	<chr>	<chr>	<chr>	<chr>
1	1130...	Forestry a...	00-0000	Industry...
2	1130...	Forestry a...	11-0000	Manageme...
3	1130...	Forestry a...	11-1021	General ...
4	1130...	Forestry a...	13-0000	Business...
5	1130...	Forestry a...	13-1023	Purchasi...

We've loaded a dataset with information on professions in the US, downloaded from the US Bureau of Labor Statistics.

The first column are the NAICS codes. These classify the industries to which the professions belong.

Let's try and get a list of the top 15 most used NAICS codes in the BLS data we loaded via excel.

Dplyr can help us with that. Dplyr is a tidyverse library used for transforming our data. It consists of a bunch of functions, including:

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

Let's try to "group" our data by the NAICS code:

```
group_by(dat, NAICS)
```

Notice that we are not putting NAICS in quotes. We are treating it as though it were a variable, even though it is not.

Dplyr does something a little funky, but it's similar to how formulas work in R. You write variable names that don't exist in your current environment, as long as they are column names on your dataframe.

```
> group_by(dat, NAICS)

# A tibble: 19,289 x 24
# Groups:   NAICS [88]
  NAICS NAICS_TITLE OCC_CODE OCC_TITLE
  <chr> <chr>         <chr>    <chr>
1 1130... Forestry a... 00-0000 Industry...
2 1130... Forestry a... 11-0000 Manageme...
...
```

Notice that grouping doesn't really do anything that we can see. The only visible difference is an extra line that informs us about the groups.



Grouping is a way to split our dataframe into many “subsets”.  
Each subset is a small dataframe (or tibble)!

What do we want to do with each of those small dataframes?

Very often, it can be helpful to look at one of the small dataframes to see what we can do to it.

Tidyr provides a helpful function that can be used for this: `nest()`

```
> nest(group_by(dat, NAICS))

# A tibble: 88 x 2
  NAICS  data
  <chr>  <list>
1 113000 <tibble [56 × 23]>
2 115000 <tibble [123 × 23]>
3 211000 <tibble [169 × 23]>
4 212000 <tibble [176 × 23]>
...

```

As the name suggests, we've created a big tibble which "nested" the small tibbles inside of it!

Let's use the `$` to access the column, which will return a list of all the small tibbles.

We then use `[[1]]` to extract the first element of the list.

```
nest(group_by(dat, NAICS))$data[[1]]
```

```
# A tibble: 56 x 23
```

```
  NAICS_TITLE OCC_CODE OCC_TITLE GROUP
  <chr>        <chr>    <chr>    <chr>
1 Forestry a... 00-0000 Industry... total
2 Forestry a... 11-0000 Manageme... major
...
```

What would `[1]` have returned?

Notice that our notation is a bit nasty.

All the functions in Dplyr and Tidyr are “pure” functions, which is to say that they do not change anything in the global environment.

As such, they *do not* mutate the original variable/dataframe/tibble.

So we need to capture their return value, and do something with it. Sometimes, this means storing it in a variable. But often, we just want to send it on to another function.

Dplyr comes with bundled with the magrittr package, which gives us the following handy infix operator:

`%>%`

An infix operator is just a fancy term for a function that is used the same way one uses + or - in arithmetic operations:

The pipe operator allows us to pass the return value of one function into the first argument of the next function:

```
dat %>%  
  group_by(NAICS) %>%  
  nest() %>%  
  .$data %>%  
  .[[1]]
```

The `.` references the value that has been passed (in this case, a tibble and then a list).

But ok, what we wanted to do with each small dataframe was to measure the size of it.

It makes sense to output this information into another dataframe: one column can be the NAICS code, the other can be the size of the subset that includes that NAICS code.

`summarize()` is a function from `dplyr` that can be used either on a dataframe (tibble) or a grouped dataframe.

If used on a single dataframe, it collapses it into a single row.

A “summary”, if you will.

If used on a grouped dataframe, it collapses each group into a single row.



`summarize()` can be used with a host of other dply functions:

```
mean()  
median()  
min()  
max()  
n()  
...
```

`n()` returns the number of rows of a dataframe (group)

```
dat %>%
  group_by(NAICS) %>%
  nest() %>%
  .$data %>%
  .[[1]] %>%
  summarize(size = n())
```

```
# A tibble: 1 x 1
  size
<int>
1    56
```

Note that `summarize()` requires all arguments to be named!

As mentioned, `summarize()` also works on grouped dataframes:

```
dat %>%
  group_by(NAICS) %>%
  summarize(size = n())
```

```
# A tibble: 88 x 2
```

	NAICS	size
	<chr>	<int>
1	113000	56
2	115000	123
3	211000	169
...		

We're close. Now we just need to sort the dataframe by size, and get the top 15:

```
dat %>%  
  group_by(NAICS) %>%  
  summarize(size = n()) %>%  
  arrange(desc(size)) %>%  
  head(15) %>%  
  pull(NAICS)
```

```
[1] "999000" "561000" "611000" "541000" "551000"  
... 
```

Voila!

In addition to `nest()` and `unnest()`, Tidy contains two other important functions:

`spread()`

`gather()`

`gather()` is used to turn data from “wide” to “long” (tidy) format.

`spread()` goes from “long” to “wide.”

In tidy format, each column is a variable and each variable only belongs to one column.

This is a very important concept!

## Reading about tidy data

Hadley Wickham gives a great discussion on tidy data:

<http://r4ds.had.co.nz/tidy.html>