Tutorial 0b: Introduction to the Tidyverse

Introducing The Tidyverse

Tidyverse

Caveat

Learning these packages is **not necessary for performing data analysis in R**. The R language has existed for 25 years, and while popular, the *tidyverse* is a relatively new addition to the R ecosystem of packages. Many statisticians, data scientists and other scientists are happy (and highly skilled!) performing data analysis in R without using the *tidyverse* packages. Most tasks that can be accomplished using packages in the *tidyverse* can be similarly completed using base R. However, in many cases (in particular, data manipulation), it can be much **easier with the tools in the *tidyverse***. For this and many other reasons (including excellent online documentation and a large user community), we hope that you'll give the *tidyverse* a try!

Tidyverse

Scope

If you're still interested and reading, **great!**

This tutorial will go over some basics about programming in R using packages in the *tidyverse*. The *tidyverse* is a family of related R packages developed to streamline data science in R. If you've ever used the `ggplot2` package to create plots, you've already experienced part of the *tidyverse*! The core *tidyverse* packages include `ggplot2`, `dplyr`, `tidyr`, `readr`, `purrr`, `tibble`, `stringr`, and `forcats`.

Tidyverse

Scope

Phew! That's a lot of packages!

Unfortunately, we don't have time to cover all of them. Instead, we'll give a light introduction to a couple of the packages that will be helpful for working with large datasets:

- `dplyr` : for data manipulation,
- `tidyr` : for "tidy"ing data.

We will assume that you've had some exposure to the powerful plotting capabilities of the `ggplot2` package through other resources.

Installing the Tidyverse

To get started, we will need to install the *tidyverse* family of packages.

```
1 install.packages("tidyverse")
```

If the packages are installed without any errors, we can load them as usual.

```
1 library(tidyverse)
```

Notice that the core tidyverse packages are listed under `Attaching packages` and loaded all at once. How wonderful!

Example Dataset

To demonstrate the basic usage of these packages, we also import the raw and summarized pharmacological datasets that we'll be analyzing today.

```
pharmacoData <- readRDS(file.path("..", "data", "rawPharmacoData.rds"))
str(pharmacoData)
```

```
'data.frame':   43427 obs. of  6 variables:
 $ cellLine     : chr  "22RV1" "22RV1" "22RV1" "22RV1" ...
 $ drug         : chr  "17-AAG" "17-AAG" "17-AAG" "17-AAG" ...
 $ doseID       : chr  "doses1" "doses2" "doses3" "doses4" ...
 $ concentration: num  0.0025 0.008 0.025 0.08 0.25 0.8 2.53 8 0.0025 0.008 ...
 $ viability    : num  94.1 86 99.9 85 62 ...
 $ study        : chr  "CCLE" "CCLE" "CCLE" "CCLE" ...
```

```
summarizedData <- readRDS(file.path("..", "data", "summarizedPharmacoData.rds"))
str(summarizedData)
```

```
'data.frame':   2557 obs. of  6 variables:
 $ cellLine : chr  "22RV1" "5637" "639-V" "697" ...
 $ drug     : chr  "Nilotinib" "Nilotinib" "Nilotinib" "Nilotinib" ...
 $ ic50_CCLE: num  8 7.48 8 1.91 8 ...
 $ auc_CCLE : num  0 0.00726 0.07101 0.15734 0 ...
 $ ic50_GDSC: num  155.27 219.93 92.18 3.06 19.63 ...
 $ auc_GDSC : num  0.00394 0.00362 0.00762 0.06927 0.02876 ...
```

The Pipe %>%

The "pipe" symbol (%>%) is a commonly used feature of the *tidyverse*. The %>% symbol can seem confusing and intimidating at first. However, once you understand the basic idea, it can become addicting!

The %>% symbol is placed between a **value on the left** and a **function on the right**. The %>% simply takes the value to the left and passes it to the function on the right as the first argument. It acts as a "pipe". That's it!

$$\text{value } \%>\% \text{ function} \quad \Longleftrightarrow \quad \text{function(value)}$$

The Pipe %>%

$$\text{value } \%>\% \text{ function} \iff \text{function(value)}$$

Suppose we have a variable, `x`.

```
1 x <- 9
```

The following are *the exact same*.

```
1 sqrt(x)
```

```
[1] 3
```

```
1 x %>% sqrt()
```

```
[1] 3
```

The Pipe %>%

$$\text{value } \%>\% \text{ function} \quad \Longleftrightarrow \quad \text{function(value)}$$

As a slightly more complex example, the following calls to `ggplot` are also equivalent.

```
gp1 <- pharmacoData %>%
    ggplot(aes(x = concentration, y = viability))

gp2 <- ggplot(pharmacoData,
              aes(x = concentration, y = viability))
```

That's it! We'll continue to use %>% throughout this tutorial to show how useful it can be for chaining various data manipulation steps during an analysis.

The dplyr Package

The `dplyr` Package

Most of this tutorial will be spent describing several functions in the `dplyr` package for working with tabular data. Remember, the best way to get a feel for these functions is to **use them to answer questions about your data**. We have included several examples for using these `dplyr` functions with the `pharmacoData` dataset.

There are many more functions in the `dplyr` package that we won't have time to cover here. More details on all of the useful functions defined in the package can be found on the `dplyr` reference page.

Subsetting

First, let's take a look at subsetting the data. To subset *rows* in a table based on values in a column, use the `filter` function.

The following examples filter the data on a single drug and a singe cell line, respectively.

```
nilotinibData <- filter(pharmacoData, drug == "Nilotinib")
head(nilotinibData)
```

```
  cellLine      drug doseID concentration viability study
1    22RV1 Nilotinib doses1        0.0025    109.98  CCLE
2    22RV1 Nilotinib doses2        0.0080    107.66  CCLE
3    22RV1 Nilotinib doses3        0.0250     97.80  CCLE
4    22RV1 Nilotinib doses4        0.0800    115.10  CCLE
5    22RV1 Nilotinib doses5        0.2500    129.50  CCLE
6    22RV1 Nilotinib doses6        0.8000    121.20  CCLE
```

```
cl639vData <- filter(pharmacoData, cellLine == "639-V")
head(cl639vData)
```

```
  cellLine   drug doseID concentration viability study
1    639-V 17-AAG doses1        0.0025      90.0  CCLE
2    639-V 17-AAG doses2        0.0080      86.0  CCLE
3    639-V 17-AAG doses3        0.0250      98.8  CCLE
4    639-V 17-AAG doses4        0.0800      77.0  CCLE
5    639-V 17-AAG doses5        0.2500      26.0  CCLE
6    639-V 17-AAG doses6        0.8000      13.0  CCLE
```

Subsetting with %>%

Redo the first example using the Pipe, %>%:

```
nilotinibData <- filter(pharmacoData, drug == "Nilotinib")
head(nilotinibData)
```

```
  cellLine      drug doseID concentration viability study
1    22RV1 Nilotinib doses1        0.0025    109.98  CCLE
2    22RV1 Nilotinib doses2        0.0080    107.66  CCLE
3    22RV1 Nilotinib doses3        0.0250     97.80  CCLE
4    22RV1 Nilotinib doses4        0.0800    115.10  CCLE
5    22RV1 Nilotinib doses5        0.2500    129.50  CCLE
6    22RV1 Nilotinib doses6        0.8000    121.20  CCLE
```

```
nilotinibData2 <- pharmacoData %>%
    filter(drug == "Nilotinib")
head(nilotinibData2)
```

```
  cellLine      drug doseID concentration viability study
1    22RV1 Nilotinib doses1        0.0025    109.98  CCLE
2    22RV1 Nilotinib doses2        0.0080    107.66  CCLE
3    22RV1 Nilotinib doses3        0.0250     97.80  CCLE
4    22RV1 Nilotinib doses4        0.0800    115.10  CCLE
5    22RV1 Nilotinib doses5        0.2500    129.50  CCLE
6    22RV1 Nilotinib doses6        0.8000    121.20  CCLE
```

Subsetting

We can also combine multiple filters.

```
n6Data <- pharmacoData %>%
    filter(drug == "Nilotinib", cellLine == "639-V")
head(n6Data)
```

```
  cellLine      drug doseID concentration viability study
1    639-V Nilotinib doses1        0.0025    106.81  CCLE
2    639-V Nilotinib doses2        0.0080     85.00  CCLE
3    639-V Nilotinib doses3        0.0250     94.90  CCLE
4    639-V Nilotinib doses4        0.0800     95.50  CCLE
5    639-V Nilotinib doses5        0.2500    102.62  CCLE
6    639-V Nilotinib doses6        0.8000    103.57  CCLE
```

Subsetting

The `distinct` function is a quick way to just take the unique rows in a table. The function can be called with zero or more columns specified. If any columns are specified, only unique rows for those columns will be returned.

The following returns the unique cell line and drug combinations in our data.

```
cldData <- pharmacoData %>%
    distinct(cellLine, drug)
head(cldData)
```

```
  cellLine       drug
1    22RV1     17-AAG
2    22RV1    AZD6244
3    22RV1  Nilotinib
4    22RV1   Nutlin-3
5    22RV1 PD-0325901
6    22RV1 PD-0332991
```

```
dim(cldData)
```

```
[1] 2557    2
```

Subsetting

To subset *columns*, use the `select` function. The following example returns a smaller table with just the `cellLine` and `drug` columns.

```
subdat <- pharmacoData %>%
    select(cellLine, drug)
head(subdat)
```

```
  cellLine   drug
1    22RV1 17-AAG
2    22RV1 17-AAG
3    22RV1 17-AAG
4    22RV1 17-AAG
5    22RV1 17-AAG
6    22RV1 17-AAG
```

Modifying

Now that we know how to subset columns, what about **adding** columns? This can be done with the `mutate` function. Suppose instead of concentrations, we want to look at the data with *log2* concentrations. We can add a new column to the table with the following call.

```
pharmacoData %>%
    mutate(logConcentration = log2(concentration)) %>%
    head()
```

```
  cellLine   drug doseID concentration viability study logConcentration
1    22RV1 17-AAG doses1        0.0025    94.100  CCLE       -8.6438562
2    22RV1 17-AAG doses2        0.0080    86.000  CCLE       -6.9657843
3    22RV1 17-AAG doses3        0.0250    99.932  CCLE       -5.3219281
4    22RV1 17-AAG doses4        0.0800    85.000  CCLE       -3.6438562
5    22RV1 17-AAG doses5        0.2500    62.000  CCLE       -2.0000000
6    22RV1 17-AAG doses6        0.8000    29.000  CCLE       -0.3219281
```

Simple enough! Notice that the new column is added as "`logConcentration`", as specified in the call to `mutate`. What would have happened if we had set the new column to "`concetration`" (the name of an existing column)? Give it a try!

Modifying

Remember, if you want to keep the new columns, you'll have to assign the modified table to a variable.

```
pharmacoData <- pharmacoData %>%
    mutate(logConcentration = log2(concentration))
head(pharmacoData)
```

```
  cellLine   drug doseID concentration viability study logConcentration
1    22RV1 17-AAG doses1        0.0025    94.100  CCLE       -8.6438562
2    22RV1 17-AAG doses2        0.0080    86.000  CCLE       -6.9657843
3    22RV1 17-AAG doses3        0.0250    99.932  CCLE       -5.3219281
4    22RV1 17-AAG doses4        0.0800    85.000  CCLE       -3.6438562
5    22RV1 17-AAG doses5        0.2500    62.000  CCLE       -2.0000000
6    22RV1 17-AAG doses6        0.8000    29.000  CCLE       -0.3219281
```

Summarizing

Another useful set of functions in the `dplyr` package allow for aggregating across the rows of a table. Suppose we want to compute some summary measures of the viability scores.

```
pharmacoData %>%
    summarize(
        minViability = min(viability),
        maxViability = max(viability),
        avgViability = mean(viability)
    )
```

```
  minViability maxViability avgViability
1          -20     319.4919     88.12281
```

Great!

Summarizing

For the simple case of counting the occurrences of the unique values in a column, use `count`. The following example counts the number of rows in the table corresponding to each study.

```
count(pharmacoData, study)
```

```
  study     n
1  CCLE 20414
2  GDSC 23013
```

Interesting! It looks like we have slightly more data from the GDSC study.

Grouping

Summarization of the entire table is great, but often we want to summarize by **groups**. For example, instead of just computing the minimum, maximum and average viability across *all* viability measures, what about computing these values for the CCLE and GDSC studies separately?

To do this, `dplyr` includes the `group_by` function. All we have to do is "group" by `study` before calling `summarize` as we did above.

```
pharmacoData %>%
    group_by(study) %>%
    summarize(
        minViability = min(viability),
        maxViability = max(viability),
        avgViability = mean(viability)
    )
```

```
# A tibble: 2 × 4
  study minViability maxViability avgViability
  <chr>        <dbl>        <dbl>        <dbl>
1 CCLE           -20          201          85.9
2 GDSC           -14.2        319.         90.1
```

Amazing, right?

Grouping

Tip: Always remember to `upgroup` your data after you're finished performing operations on the groups. Forgetting that your data is still "grouped" can cause major headaches while performing data analysis! If you're not sure if the data is grouped, just `ungroup`! (There's no harm in calling `ungroup` too often.)

```
pharmacoData %>%
    group_by(cellLine, drug, study) %>%
    mutate(viability = viability / max(viability) * 100) %>%
    ungroup()
```

```
# A tibble: 43,427 × 7
   cellLine drug    doseID concentration viability study logConcentration
   <chr>    <chr>   <chr>          <dbl>     <dbl> <chr>            <dbl>
 1 22RV1    17-AAG  doses1        0.0025      94.2 CCLE             -8.64
 2 22RV1    17-AAG  doses2        0.008       86.1 CCLE             -6.97
 3 22RV1    17-AAG  doses3        0.025      100   CCLE             -5.32
 4 22RV1    17-AAG  doses4        0.08        85.1 CCLE             -3.64
 5 22RV1    17-AAG  doses5        0.25        62.0 CCLE             -2
 6 22RV1    17-AAG  doses6        0.8         29.0 CCLE             -0.322
 7 22RV1    17-AAG  doses7        2.53        26.0 CCLE              1.34
 8 22RV1    17-AAG  doses8        8           20.0 CCLE              3
 9 22RV1    AZD6244 doses1        0.0025     100   CCLE             -8.64
10 22RV1    AZD6244 doses2        0.008       80.3 CCLE             -6.97
# ℹ 43,417 more rows
```

Can you figure out what we're doing in the code above?

Joining

Finally, the `dplyr` package includes several functions for combining multiple tables. These functions are incredibly useful for combining multiple tables with partially overlapping data. For example, what if we want to combine the raw and summarized pharmacological datasets?

Notice that both datasets include columns with `cellLine` and `drug` information.

```
1 head(pharmacoData)
```

```
  cellLine   drug doseID concentration viability study logConcentration
1    22RV1 17-AAG doses1        0.0025    94.100  CCLE       -8.6438562
2    22RV1 17-AAG doses2        0.0080    86.000  CCLE       -6.9657843
3    22RV1 17-AAG doses3        0.0250    99.932  CCLE       -5.3219281
4    22RV1 17-AAG doses4        0.0800    85.000  CCLE       -3.6438562
5    22RV1 17-AAG doses5        0.2500    62.000  CCLE       -2.0000000
6    22RV1 17-AAG doses6        0.8000    29.000  CCLE       -0.3219281
```

```
1 head(summarizedData)
```

```
  cellLine      drug ic50_CCLE  auc_CCLE  ic50_GDSC auc_GDSC
1    22RV1 Nilotinib  8.000000 0.0000000 155.269917 0.003935
2     5637 Nilotinib  7.475355 0.0072625 219.934550 0.003616
3    639-V Nilotinib  8.000000 0.0710125  92.177125 0.007622
4      697 Nilotinib  1.910434 0.1573375   3.063552 0.069265
5    769-P Nilotinib  8.000000 0.0000000  19.633514 0.028758
6    786-0 Nilotinib  8.000000 0.0750125 137.066882 0.005482
```

Joining

```
1 fullData <- full_join(pharmacoData, summarizedData, by = c("cellLine", "drug"))
2 head(fullData)
```

```
  cellLine   drug doseID concentration viability study logConcentration
1    22RV1 17-AAG doses1        0.0025    94.100  CCLE       -8.6438562
2    22RV1 17-AAG doses2        0.0080    86.000  CCLE       -6.9657843
3    22RV1 17-AAG doses3        0.0250    99.932  CCLE       -5.3219281
4    22RV1 17-AAG doses4        0.0800    85.000  CCLE       -3.6438562
5    22RV1 17-AAG doses5        0.2500    62.000  CCLE       -2.0000000
6    22RV1 17-AAG doses6        0.8000    29.000  CCLE       -0.3219281
  ic50_CCLE auc_CCLE ic50_GDSC auc_GDSC
1 0.3297017  0.37246  3.491684 0.067091
2 0.3297017  0.37246  3.491684 0.067091
3 0.3297017  0.37246  3.491684 0.067091
4 0.3297017  0.37246  3.491684 0.067091
5 0.3297017  0.37246  3.491684 0.067091
6 0.3297017  0.37246  3.491684 0.067091
```

Notice that we now have a single table with the columns from both tables. There are several other functions for merging tables, including `left_join`, `inner_join`, and `anti_join`. To learn more about how these differ, take a look at the documentation page.

The `tidyr` Package

The `tidyr` Package

While the `dplyr` package includes many functions for manipulating data, we would have a hard time using these tools if our data is not in the correct "form". For example, what if we want to compare the viability scores in `pharmacoData` for two drugs in the CCLE study? To do this, we would want the two drugs to be in separate columns, so that we can compare them side-by-side. No amount of subsetting or mutating the table will get us there. We need to fundamentally *transform the shape* of our data.

The `tidyr` Package

This is where the `tidyr` package comes in. The `tidyr` package includes several functions to help with arranging and rearranging our data. We will highlight the two most important functionxs for this task:

- `pivot_wider`: to spread values in a single column to multiple columns, making the table **wider**,
- `pivot_longer`: to gather values in multiple columns to a single column, making the table **longer**.

Again, there are many more functions in the `tidyr` package that we won't have time to cover here. More details can be found on the `tidyr` reference page.

Don't worry if it takes some time for these ideas to start making sense! At first, transforming data with `tidyr` can feel like mental yoga.

Pivoting Wider

To demonstrate what it means to take a data set and `pivot_wider`, let's consider the example described above. Suppose we would like to compare the viability scores for two drugs in the CCLE study, `lapatinib` and `paclitaxel`, across cell lines and concentrations.

First, using the `dplyr` functions from above, we'll subset the data.

```
subdat <- pharmacoData %>%
    filter(study == "CCLE",
           drug %in% c("lapatinib", "paclitaxel")) %>%
    select(cellLine, drug, concentration, viability)
head(subdat)
```

```
  cellLine      drug concentration viability
1      697 lapatinib        0.0025     89.00
2      697 lapatinib        0.0080    104.52
3      697 lapatinib        0.0250    117.90
4      697 lapatinib        0.0800    140.10
5      697 lapatinib        0.2500     96.00
6      697 lapatinib        0.8000     83.00
```

Pivoting Wider

Next, we will use the `pivot_wider` function to take the viability scores for the two drugs into separate columns. How do we do this? We would like to take the values in the `drug` column and turn these into our new columns. We then want to fill these columns with values from the `viability` column. To do this, we simply specify `drug` as the "`names_from=`" and `viability` as the "`values_from=`" parameters to the `pivot_wider` function.

```
1 subdat_wide <- pivot_wider(subdat,
2                            names_from = drug,
3                            values_from = viability)
4 head(subdat_wide)
```

```
# A tibble: 6 × 4
  cellLine concentration lapatinib paclitaxel
  <chr>            <dbl>     <dbl>      <dbl>
1 697             0.0025        89         12
2 697             0.008        105.         3
3 697             0.025        118.         3
4 697             0.08         140.         2
5 697             0.25          96          1
6 697             0.8           83          1
```

Great! Notice that the data in the other columns (`cellLine` and `concentration`) are still there. When populating the `lapatinib` and `paclitaxel` columns, the

Pivoting Longer

Next, let's use the `summarizedData` to demonstrate how `pivot_longer` works.

```
1 head(summarizedData)
```

```
  cellLine      drug ic50_CCLE  auc_CCLE  ic50_GDSC auc_GDSC
1    22RV1 Nilotinib  8.000000 0.0000000 155.269917 0.003935
2     5637 Nilotinib  7.475355 0.0072625 219.934550 0.003616
3    639-V Nilotinib  8.000000 0.0710125  92.177125 0.007622
4      697 Nilotinib  1.910434 0.1573375   3.063552 0.069265
5    769-P Nilotinib  8.000000 0.0000000  19.633514 0.028758
6    786-0 Nilotinib  8.000000 0.0750125 137.066882 0.005482
```

Pivoting Longer

Suppose we now want to organize all of the IC50 and AUC values stored in the separate `ic50_CCLE`, `auc_CCLE`, `ic50_GDSC` and `auc_GDSC` columns into a single column of "metric values". Essentially, we would like to reverse the "widening" procedure that we carried out above (turn a short wide table into a long skinny table). To do this, we call `pivot_longer`, specifying the columns we want to bring together (here, `ic50_CCLE`, `auc_CCLE`, `ic50_GDSC` and `auc_GDSC`), along with new column names for the two columns that will contain the former column names and the values (we'll just call these `metric` and `value`.

```
summarizedDataLong <- pivot_longer(summarizedData,
                                   c(ic50_CCLE, auc_CCLE, ic50_GDSC, auc_GDSC),
                                   names_to = "metric",
                                   values_to = "value")
head(summarizedDataLong)
```

```
# A tibble: 6 × 4
  cellLine drug      metric        value
  <chr>    <chr>     <chr>         <dbl>
1 22RV1    Nilotinib ic50_CCLE   8
2 22RV1    Nilotinib auc_CCLE    0
3 22RV1    Nilotinib ic50_GDSC 155.
4 22RV1    Nilotinib auc_GDSC    0.00394
5 5637     Nilotinib ic50_CCLE   7.48
6 5637     Nilotinib auc_CCLE    0.00726
```

Pivoting Longer

Notice that the former column names (`ic50_CCLE`, `auc_CCLE`, `ic50_GDSC` and `auc_GDSC`) are now in the `metric` column, and the values are in the `value` column. Alternatively, since the number of columns we would like to exclude is smaller, we can specify these columns with the a "minus".

```
## alternatively
summarizedDataLong <- pivot_longer(summarizedData,
                                   -c(cellLine, drug),
                                   names_to = "metric",
                                   values_to = "value")
head(summarizedDataLong)
```

```
# A tibble: 6 × 4
  cellLine drug      metric        value
  <chr>    <chr>     <chr>         <dbl>
1 22RV1    Nilotinib ic50_CCLE   8
2 22RV1    Nilotinib auc_CCLE    0
3 22RV1    Nilotinib ic50_GDSC 155.
4 22RV1    Nilotinib auc_GDSC    0.00394
5 5637     Nilotinib ic50_CCLE   7.48
6 5637     Nilotinib auc_CCLE    0.00726
```

The result is the same!

Now, we have a new column of the names (`metric`) and a new column (`value`) with the original entries of those columns.

Pivoting Longer

The `pivot_longer` and `pivot_wider` functions are opposites. Therefore, we can undo the `pivot_longer` operation above by calling `pivot_wider`.

```
summarizedDataUndo <- pivot_wider(summarizedDataLong,
                                  names_from = metric,
                                  values_from = value)
head(summarizedDataUndo)
```

```
# A tibble: 6 × 6
  cellLine drug      ic50_CCLE auc_CCLE ic50_GDSC auc_GDSC
  <chr>    <chr>         <dbl>    <dbl>     <dbl>    <dbl>
1 22RV1    Nilotinib      8     0          155.    0.00394
2 5637     Nilotinib      7.48  0.00726    220.    0.00362
3 639-V    Nilotinib      8     0.0710      92.2   0.00762
4 697      Nilotinib      1.91  0.157        3.06  0.0693
5 769-P    Nilotinib      8     0           19.6   0.0288
6 786-0    Nilotinib      8     0.0750     137.    0.00548
```

We are back to our original dataset!

References

For a complete book on how to do data science using R and the *tidyverse*, we highly recommend **R for Data Science**, available for free online, by Garrett Grolemund and Hadley Wickham.

More practically, the accompanying websites for the tidyverse packages are absolutely amazing. These sites are a great resource for trying to understand how these packages work. Also, when in doubt ask the internet.