Data Wrangling with dplyr

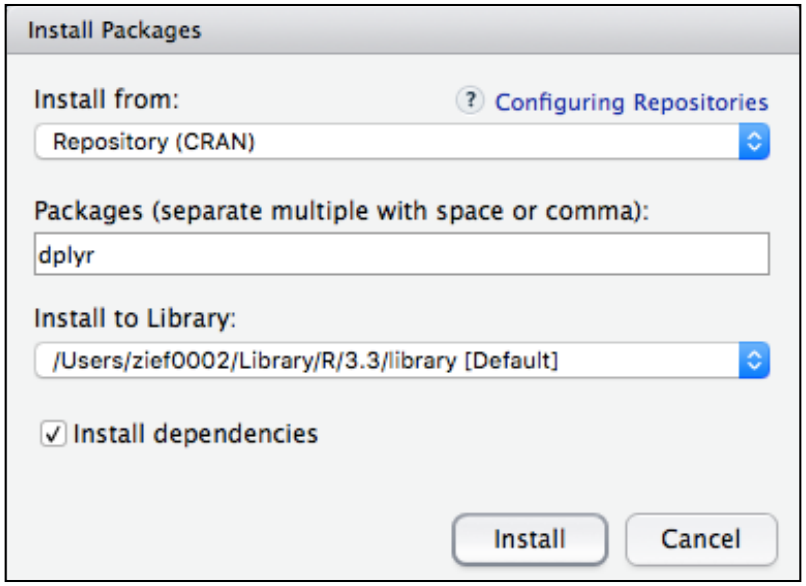
Andrew Zieffler

```
# Load the city data
> city = read.csv("~/Google Drive/andy/epsy-8251/data/riverside.csv")

> head(city)

  education income seniority gender male       party
1         8  37449         7   male    1    Democrat
2         8  26430         9 female    0 Independent
3        10  47034        14   male    1    Democrat
4        10  34182        16 female    0 Independent
5        10  25479         1 female    0  Republican
6        12  46488        11 female    0    Democrat
```

Install and load the **dplyr** Package

Using the RStudio GUI...

- Click the **Packages** tab.
- Click **Install Packages**.
- Enter *dplyr* in the text box.
- Click **Install**.

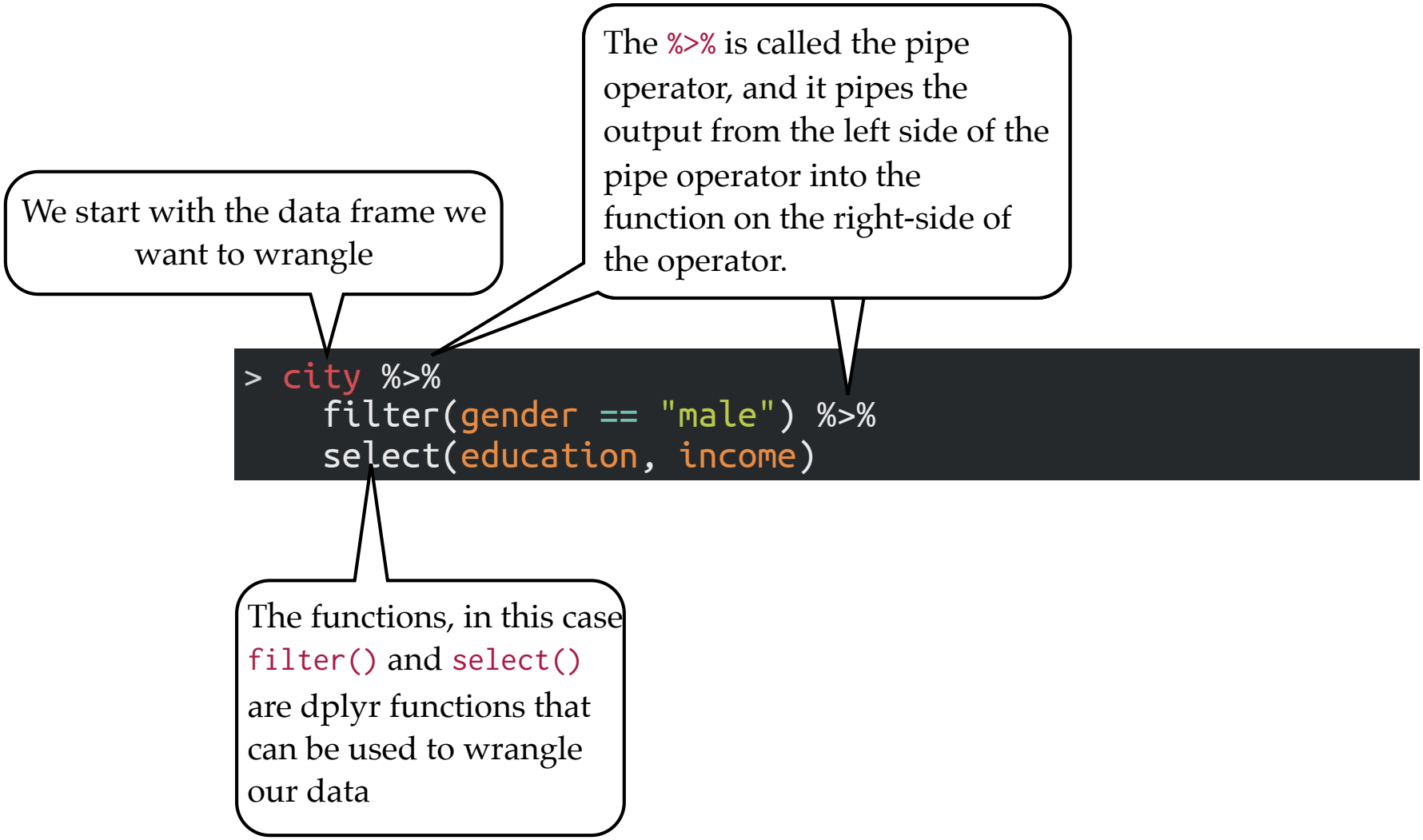


...or directly from the R command line...

```
> install.packages("dplyr", dependencies = TRUE)
```

```
# After installing the package, load the dplyr library
> library(dplyr)
```

Understanding the Basic Syntax



Common dplyr Functions

Here are some common operations that we want to use with data to "wrangle" it into shape and the corresponding **dplyr** functions.

- Select a subset of rows from a data frame. Use the `filter()` function.
- Select a subset of columns from a data frame. Use the `select()` function.
- Add new columns that are functions of existing columns. Use the `mutate()` function.
- Sort and re-order data in a data frame. Use the `arrange()` function.
- Compute summaries of a data frame. Use the `summarize()` function.
- Group the data to carry out computations for each group. Use the `group_by()` function.

Select a Subset of Rows

To select a subset of rows, we will use the `filter()` function. The argument(s) for this function are expressions that filter the data frame.

```
> city %>% filter(gender == "male")
```

The == operator says "is equal to". Remember a single = is the assignment operator

Here we are selecting only the rows where the gender variable **is equal to** (==) the string “male”.

```
> city %>% filter(gender == "male")


   education income seniority gender male       party
1          8  37449         7   male    1    Democrat
2         10  47034        14   male    1    Democrat
3         12  37656        14   male    1    Democrat
4         12  50265        24   male    1    Democrat
5         14  49968        18   male    1 Independent
6         14  64926        26   male    1 Independent
7         16  55782         6   male    1    Democrat
8         16  63471        10   male    1    Democrat
9         19  65142        16   male    1  Republican
10        20  61629         5   male    1  Republican
11        20  82726        23   male    1  Republican
12        21  73542        16   male    1  Republican
13        22  70044        14   male    1 Independent
14        24  79227        27   male    1    Democrat
```

Note the output is just printed to the screen. If you want to keep the filtered data or operate on it further, you need to write the output into an object.

```
> males = city %>% filter(gender == "male")


> head(males)

  education income seniority gender male       party
1         8  37449         7   male    1    Democrat
2        10  47034        14   male    1    Democrat
3        12  37656        14   male    1    Democrat
4        12  50265        24   male    1    Democrat
5        14  49968        18   male    1 Independent
6        14  64926        26   male    1 Independent


> mean(males$income)

[1] 59919
```

Your Turn

Find the average annual pay for females.

```
> high_school = city %>% filter(education <= 12)

> mean(high_school $income)

[1] 39718
```

You can filter on multiple attributes by adding additional arguments.

```
> males_high_school = city %>% filter(gender == "male", education <= 12)

> males_high_school

  education income seniority gender male    party
1         8  37449         7   male    1 Democrat
2        10  47034        14   male    1 Democrat
3        12  37656        14   male    1 Democrat
4        12  50265        24   male    1 Democrat


> mean(males_high_school $income)

[1] 43101
```

Linking Expressions: AND and OR

When we include multiple expressions in the `filter()` function, they are linked using the AND (&) operator. This means that both expressions have to evaluate as TRUE to be included.

```
> city %>% filter(gender == "male" & education <= 12)
```

We can also `filter()` using the OR (|) operator. This means that if either expression evaluates as TRUE it is included.

```
> city %>% filter(gender == "male" | education <= 12)
```

```
> males_OR_high_school = city %>% filter(gender == "male" | education <= 12)

> males_OR_high_school
```

	education	income	seniority	gender	male	party
1	8	37449	7	male	1	Democrat
2	8	26430	9	female	0	Independent
3	10	47034	14	male	1	Democrat
4	10	34182	16	female	0	Independent
5	10	25479	1	female	0	Republican
6	12	46488	11	female	0	Democrat
7	12	37656	14	male	1	Democrat
8	12	50265	24	male	1	Democrat
9	12	52480	16	female	0	Independent
10	14	49968	18	male	1	Independent
11	14	64926	26	male	1	Independent
12	16	55782	6	male	1	Democrat
13	16	63471	10	male	1	Democrat
14	19	65142	16	male	1	Republican
15	20	61629	5	male	1	Republican
16	20	82726	23	male	1	Republican
17	21	73542	16	male	1	Republican
18	22	70044	14	male	1	Independent
19	24	79227	27	male	1	Democrat

Here to be selected the employee needs to be male…OR..have an education level less than or equal to 12 years.

Your Turn

Select only the female Democrats.

Select a Subset of Columns

To select a subset of columns, we will use the `select()` function. The argument(s) for this function are column names of the data frame that you want to select.

```
> city2 = city %>% select(education, income, gender)

> head(city2)

  education income gender
1         8  37449   male
2         8  26430 female
3        10  47034   male
4        10  34182 female
5        10  25479 female
6        12  46488 female
```

You can rename a column by naming it in the `select()` function. Here we rename *education* to *edu*

```
> city2 = city %>% select(edu = education, income, gender)

> head(city2)

  Edu income gender
1   8  37449   male
2   8  26430 female
3  10  47034   male
4  10  34182 female
5  10  25479 female
6  12  46488 female
```

There are a number of helper functions you can use within `select()`. For example, `starts_with()`, `ends_with()`, and `contains()`. These let you quickly match larger blocks of variables that meet some criterion.

```
> city2 = city %>% select(ends_with("e"))

> head(city2)


  income male
1  37449    1
2  26430    0
3  47034    1
4  34182    0
5  25479    0
6  46488    0
```

Create New Columns

To create new columns, we will use the `mutate()` function.

```
> city3 = city %>%
    mutate(income2 = income / 1000)

> head(city3)

  education income seniority gender male       party income2
1         8  37449         7   male    1    Democrat    37.4
2         8  26430         9 female    0 Independent    26.4
3        10  47034        14   male    1    Democrat    47.0
4        10  34182        16 female    0 Independent    34.2
5        10  25479         1 female    0  Republican    25.5
6        12  46488        11 female    0    Democrat    46.5
```

Create multiple columns by including each new column as an argument in the `mutate()` function.

```
> city3 = city %>%
    mutate(
       income2 = income / 1000,
       educ_after_8 = education - 8
    )

> head(city3)

  education income seniority gender male       party income2 educ_after_8
1         8  37449         7   male    1    Democrat    37.4            0
2         8  26430         9 female    0 Independent    26.4            0
3        10  47034        14   male    1    Democrat    47.0            2
4        10  34182        16 female    0 Independent    34.2            2
5        10  25479         1 female    0  Republican    25.5            2
6        12  46488        11 female    0    Democrat    46.5            4
```

Arrange

The `arrange()` function sorts the data within a data frame. The data is ordered based on the column name provided in the argument(s).

```
> city4 = city %>% arrange(income)

> city4


  education income seniority gender male       party
1        10  25479         1 female    0  Republican
2         8  26430         9 female    0 Independent
3        14  32631         5 female    0 Independent
4        10  34182        16 female    0 Independent
5        15  37302         8 female    0    Democrat
6         8  37449         7   male    1    Democrat

27        19  65142        16   male    1  Republican
28        22  70044        14   male    1 Independent
29        21  71202        26 female    0    Democrat
30        21  73542        16   male    1  Republican
31        24  79227        27   male    1    Democrat
32        20  82726        23   male    1  Republican
```

Multiple arguments sort first by the first argument, and then by the subsequent arguments.

```
> city4 = city %>% arrange(gender, income)

> head(city4)

   education income seniority gender male       party
1         10  25479         1 female    0  Republican
2          8  26430         9 female    0 Independent
3         14  32631         5 female    0 Independent
4         10  34182        16 female    0 Independent
5         15  37302         8 female    0    Democrat
6         16  38586        11 female    0 Independent

27        14  64926        26   male    1 Independent
28        19  65142        16   male    1  Republican
29        22  70044        14   male    1 Independent
30        21  73542        16   male    1  Republican
31        24  79227        27   male    1    Democrat
32        20  82726        23   male    1  Republican
```

Use the `desc()` function to order a column in descending order:

```
> city4 = city %>% arrange(gender, desc(income))

> head(city4)

   education income seniority gender male       party
1         21  71202        26 female    0    Democrat
2         18  62466        16 female    0  Republican
3         17  60068        10 female    0 Independent
4         16  59499        20 female    0 Independent
5         20  56343        22 female    0 Independent
6         22  56322         8 female    0 Independent

27        16  55782         6   male    1    Democrat
28        12  50265        24   male    1    Democrat
29        14  49968        18   male    1 Independent
30        10  47034        14   male    1    Democrat
31        12  37656        14   male    1    Democrat
32         8  37449         7   male    1    Democrat
```

Summarizing

The `summarize()` function is used to compute summaries of data. It collapses a data frame to a single row.

```
> mySummaries = city %>% summarize( M = mean(income) )

> mySummaries

      M
1 53742
```

The output is a single row data frame with a column called *M*.

Multiple summaries can be computed by providing more than one argument to the `summarize()` function. The output is still a single row data frame, but now there will be multiple columns, one for each summary computation.

```
> mySummaries = city %>%
    summarize(
      M = mean(income),
      SD = sd(income)
    )

> mySummaries

      M    SD
1 53742 14553
```

The `group_by()` function groups the data by a specified variable. By itself, it does nothing, but it is powerful when the grouped output is chained to other functions, such as `summarize()`.

```
> mySummaries = city %>%
    group_by(gender) %>%
    summarize(
      M = mean(income),
      SD = sd(income)
    )

> mySummaries

# A tibble: 2 × 3
  gender     M    SD
  <fctr> <dbl> <dbl>
1 female 48938 13265
2   male 59919 14210
```

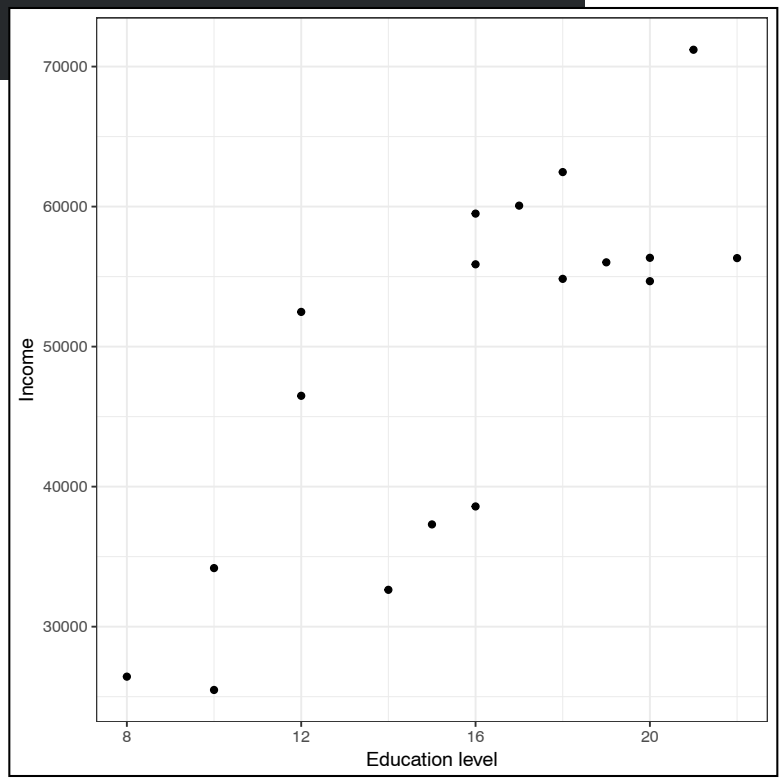
The output from summarize now has multiple rows, one for each level of the grouping variable. It computes the two summaries for each of those groups and outputs this in a data frame. It also includes a column for the grouping level.

We can combine dplyr output with ggplot. For example, what if you wanted to plot the relationship between income and education level for females? You could filter the data to get the females, and plot them.

```
# Filter the females
> females = city %>%
    filter(gender == "female")


# Load the ggplot2 library
> library(ggplot2)


# Plot
> ggplot(data = females, aes(x = education, y = income)) +
    geom_point() +
    theme_bw() +
    xlab("Education level") +
    ylab("Income")
```

We can also pipe dplyr output DIRECTLY into ggplot. The `data=.` (dot) tells ggplot to use the dataset that was just piped into it.

```
city %>%
  filter(gender == "female") %>%
  ggplot(data = ., aes(x = education, y = income)) +
    geom_point() +
    theme_bw() +
    xlab("Education level") +
    ylab("Income")
```



dplyr Resources

- **dplyr Cheatsheet**: A one-page (front and back) cheatsheet of dplyr syntax with pictures https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

- **Introduction to dplyr:** Web-based tutorial with examples https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html

- **tidy data paper:** A paper that outlines how to tidy/clean data for analysis. http://vita.had.co.nz/papers/tidy-data.html

`#protip`**:** Use Google to find out how to do just about anything with dplyr.