Data Wrangling

Adam Rich, FCAS

2022-06-09

Session #6: Data Wrangling

Shaping raw data so it may be used to build a model is a key task for any actuary or data scientist. This session will walk through some basic tasks to take information and **transform** it into something useful. We will cover topics like **filtering**, **grouping** and **aggregation** and **combining** multiple sources of data. Examples will use the `tidyverse` packages in R including `dplyr`.

Load **tidyverse**

```
library(tidyverse)
```

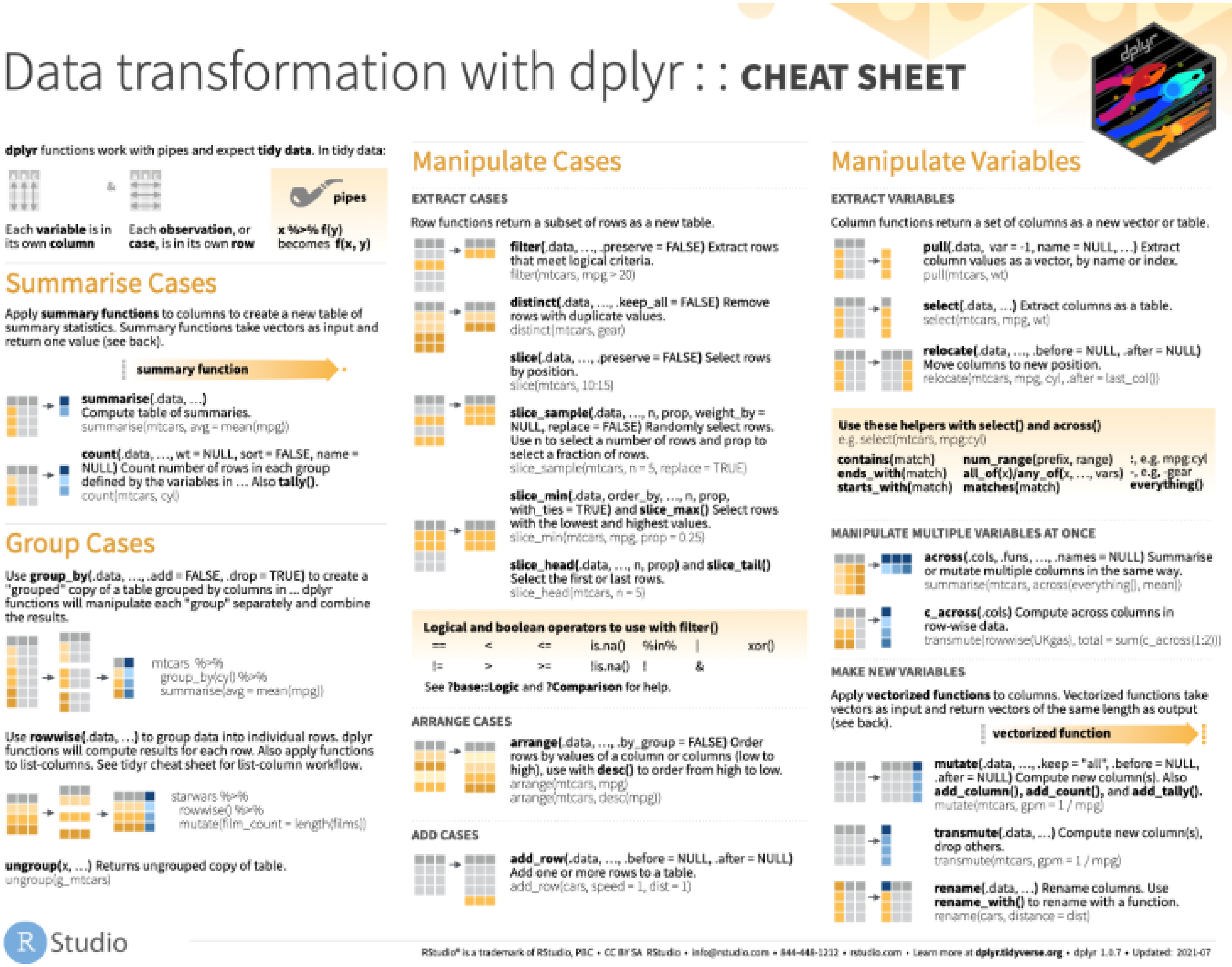
```
-- Attaching packages ---------------- tidyverse 1.3.1 --
v ggplot2 3.3.6     v purrr   0.3.4
v tibble  3.1.7     v dplyr   1.0.9
v tidyr   1.2.0     v stringr 1.4.0
v readr   2.1.2     v forcats 0.5.1
-- Conflicts ------------------- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

Loaded packages

Package	Description
`ggplot2`	A system for 'declaratively' creating graphics
`tibble`	The tibble class is a re-implementation of the data frame
`tidyr`	Tools to help to create tidy data: pivoting, unnesting, rectangling, and imputing
`readr`	Provides a fast and friendly way to read rectangular data
`purrr`	A complete and consistent functional programming toolkit for R
`dplyr`	A fast & consistent tool for working with data frames
`stringr`	A consistent, simple and easy to use set of string functions
`forcats`	Tools for working with factors

Get the `dplyr` cheatsheet!

https://www.rstudio.com/resources/cheatsheets/



Data transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect tidy data. In tidy data:

Each variable is in its own column & Each observation, or case, is in its own row

pipes: x %>% f(y) becomes f(x, y)

Summarise Cases

Apply summary functions to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, ...) Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(.data, ..., wt = NULL, sort = FALSE, name = NULL) Count number of rows in each group defined by the variables in ... Also tally().
count(mtcars, cyl)

Group Cases

Use group_by(.data, ..., .add = FALSE, .drop = TRUE) to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

Use rowwise(.data, ...) to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

starwars %>%
rowwise() %>%
mutate(film_count = length(films))

ungroup(x, ...) Returns ungrouped copy of table.
ungroup(g_mtcars)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, ..., .preserve = FALSE) Extract rows that meet logical criteria.
filter(mtcars, mpg > 20)

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(mtcars, gear)

slice(.data, ..., .preserve = FALSE) Select rows by position.
slice(mtcars, 10:15)

slice_sample(.data, ..., n, prop, weight_by = NULL, replace = FALSE) Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
slice_sample(mtcars, n = 5, replace = TRUE)

slice_min(.data, order_by, ..., n, prop, with_ties = TRUE) and slice_max() Select rows with the lowest and highest values.
slice_min(mtcars, mpg, prop = 0.25)

slice_head(.data, ..., n, prop) and slice_tail() Select the first or last rows.
slice_head(mtcars, n = 5)

Logical and boolean operators to use with filter()

==	<	<=	is.na()	%in%	|	xor()
!=	>	>=	!is.na()	!	&	

See ?base::Logic and ?Comparison for help.

ARRANGE CASES

arrange(.data, ..., .by_group = FALSE) Order rows by values of a column or columns (low to high), use with desc() to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table.
add_row(cars, speed = 1, dist = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1, name = NULL, ...) Extract column values as a vector, by name or index.
pull(mtcars, wt)

select(.data, ...) Extract columns as a table.
select(mtcars, mpg, wt)

relocate(.data, ..., .before = NULL, .after = NULL) Move columns to new position.
relocate(mtcars, mpg, cyl, .after = last_col())

Use these helpers with select() and across()
e.g. select(mtcars, mpg:cyl)

contains(match) num_range(prefix, range) :, e.g. mpg:cyl
ends_with(match) all_of(x)/any_of(x, ..., vars) -, e.g, -gear
starts_with(match) matches(match) everything()

MANIPULATE MULTIPLE VARIABLES AT ONCE

across(.cols, .funs, ..., .names = NULL) Summarise or mutate multiple columns in the same way.
summarise(mtcars, across(everything(), mean))

c_across(.cols) Compute across columns in row-wise data.
transmute(rowwise(UKgas), total = sum(c_across(1:2)))

MAKE NEW VARIABLES

Apply vectorized functions to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, ..., .keep = "all", .before = NULL, .after = NULL) Compute new column(s). Also add_column(), add_count(), and add_tally().
mutate(mtcars, gpm = 1 / mpg)

transmute(.data, ...) Compute new column(s), drop others.
transmute(mtcars, gpm = 1 / mpg)

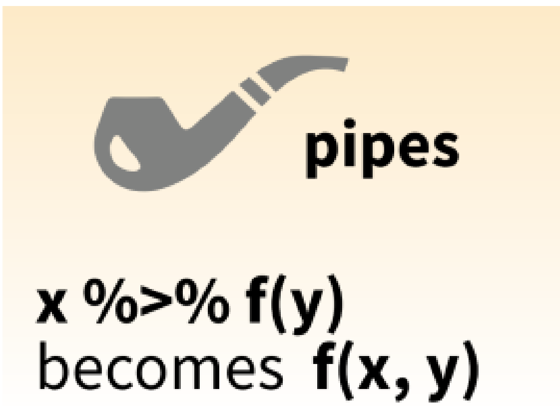
rename(.data, ...) Rename columns. Use rename_with() to rename with a function.
rename(cars, distance = dist)

RStudio

RStudio® is a trademark of RStudio, PBC • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at dplyr.tidyverse.org • dplyr 1.0.7 • Updated: 2021-07

The pipe operator

The `tidyverse` packages, especially `dplyr` rely heavily on an operator that is not included in R by default – the pipe operator, or `%>%`. We use the pipe operator to chain functions that we would otherwise have to nest.



Shortcut key is `CTRL-SHIFT-M`

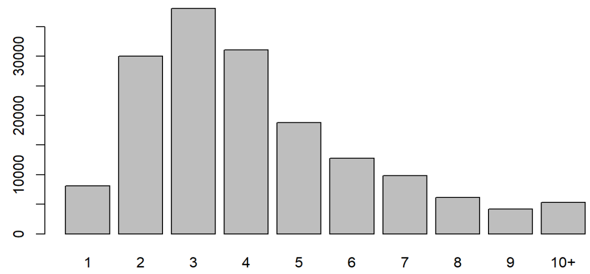
A fun pipe example

What is the distribution of token lengths in a corpus like Bram Stoker's *Dracula*?

```
dracula <- readr::read_file(
  'http://gutenberg.org/cache/epub/345/pg345.txt')
dracula_words <- strsplit(dracula, split = '\\s+')
```

Chain of functions

```
words <- unlist(dracula_words)
words_lower <- tolower(words)
nchar_words <- nchar(words_lower)
nchar_words10 <- pmin(nchar_words, 10)
table_words10 <- table(nchar_words10)
barplot(table_words10, names.arg = c(1:9, '10+'))
```



That's a lot of new variables.

Combine as one

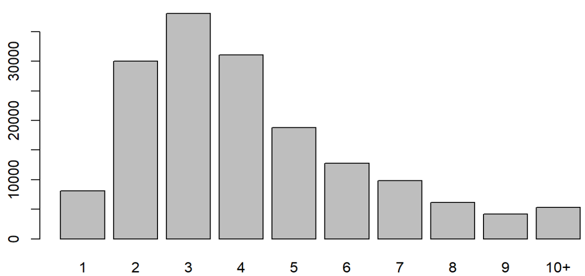
```
unlist(dracula_words)
tolower(unlist(dracula_words))
nchar(tolower(unlist(dracula_words)))
pmin(nchar(tolower(unlist(dracula_words))), 10)
table(pmin(nchar(tolower(unlist(dracula_words))), 10))

# Fully nested!
barplot(table(pmin(
  nchar(tolower(unlist(dracula_words))), 10)),
  names.arg = c(1:9, '10+'))
```

That's hard to read.

Pipe it!

```
dracula_words %>%
  unlist() %>%
  tolower() %>%
  nchar() %>%
  pmin(10) %>%
  table() %>%
  barplot(names.arg = c(1:9, '10+'))
```



Data for today's examples

```r
install.packages(
  c('xts', 'sp'),
  type = 'binary')

install.packages(
  'CASdatasets',
  repos = 'http://cas.uqam.ca/pub/')

library(CASdatasets)
```

`CAS` here stands for *Computational Actuarial Science with R*, by several authors and edited by Arthur Charpentier.

Brazilian Vehicle Insurance Data

```
data(brvehins1a, package = 'CASdatasets')
brvehins1a <- as_tibble(brvehins1a)
```

When printed to the console, tibbles generate output that is more user friendly than data frames when the data set is large.

```
print(brvehins1a)
```

```
## # A tibble: 393,071 x 23
##   Gender DrivAge VehYear VehModel       VehGroup Area  State
##   <fct>  <fct>     <int> <fct>          <fct>    <fct> <fct>
## 1 Female >55        1997 Gm - Chevrole~ Gm Chev~ Inte~ Rio ~
## 2 Female 36-45      2010 Gm - Chevrole~ Gm Chev~ Mara~ Mara~
## 3 Female 18-25      2008 Vw - Volkswag~ Vw Volk~ Mato~ Mato~
## 4 Male   >55        2004 Harley-davids~ Harley-~ Met.~ Rio ~
## 5 Male   36-45      2009 Volvo - Fh 44~ Volvo C~ Ribe~ Sao ~
## # ... with 393,066 more rows, and 16 more variables:
## #   StateAb <fct>, ExposTotal <dbl>, ExposFireRob <dbl>,
## #   PremTotal <dbl>, PremFireRob <dbl>, SumInsAvg <dbl>,
## #   ClaimNbRob <dbl>, ClaimNbPartColl <dbl>,
## #   ClaimNbTotColl <dbl>, ClaimNbFire <dbl>,
## #   ClaimNbOther <dbl>, ClaimAmountRob <dbl>,
## #   ClaimAmountPartColl <dbl>, ...
```

Select columns using `select`

```
brvehins1a %>%
  select(ExposTotal, PremTotal)
```

```
## # A tibble: 393,071 x 2
##   ExposTotal PremTotal
##        <dbl>     <dbl>
## 1       1.01      743.
## 2       3        5026.
## 3       1.01      916.
## 4       1.45     1602.
## 5       4.55    53031.
## # ... with 393,066 more rows
```

Anti-select operator

```r
brvehins1a %>%
  select(-Gender, -ClaimNbFire)
```

You can only remove columns that actually exist.

```r
brvehins1a %>%
  select(-DriverAge)
```

```
## Error in `select()`:
## ! Can't subset columns that don't exist.
## x Column `DriverAge` doesn't exist.
```

Extract vectors using `pull`

`select` gives you a tibble, even if you only get one column of data. If you want to get a single vector instead of a one-column tibble, use `pull`.

```
brvehins1a %>%
  pull(PremTotal)
```

If you pass two columns to `pull` you get a named vector. That may or may not make any sense depending on the data you have.

```
brvehins1a %>%
  pull(PremTotal, Gender) %>%
  head
```

```
##   Female   Female   Female     Male     Male     Male
##   742.75  5025.68   916.26  1601.68 53031.35   660.59
```

Helper functions for `select`

- `contains`
- `ends_with`
- `starts_with`
- `matches`
- `num_range`
- `all_of`
- `any_of`
- `everything`

Helper function examples

```
brvehins1a %>%
  select(contains('Veh'))

brvehins1a %>%
  select(ends_with('Total'))

brvehins1a %>%
  select(starts_with('claim'))

brvehins1a %>%
  select(matches('claim.*coll'))
```

Extract rows using `filter`

Any function that returns a vector of `TRUE` and `FALSE` values works.

- `==`, `<`, `>`, `<=`, etc.
- `is.na`
- `%in%`
- `stringr::str_detect` and `stringr::str_starts`
- And many more

Simple examples

```
brvehins1a %>%
  filter(Gender == 'Male')
```

```
## # A tibble: 171,089 x 23
##   Gender DrivAge VehYear VehModel       VehGroup Area  State
##   <fct>  <fct>     <int> <fct>          <fct>    <fct> <fct>
## 1 Male   >55        2004 Harley-davids~ Harley-~ Met.~ Rio ~
## 2 Male   36-45      2009 Volvo - Fh 44~ Volvo C~ Ribe~ Sao ~
## 3 Male   26-35      1998 Vw - Volkswag~ Vw Volk~ Met.~ Rio ~
## 4 Male   18-25      1999 Gm - Chevrole~ Gm Chev~ Gran~ Sao ~
## 5 Male   36-45      2010 Volvo - C30 2~ Volvo -~ Tria~ Mina~
## # ... with 171,084 more rows, and 16 more variables:
## #   StateAb <fct>, ExposTotal <dbl>, ExposFireRob <dbl>,
## #   PremTotal <dbl>, PremFireRob <dbl>, SumInsAvg <dbl>,
## #   ClaimNbRob <dbl>, ClaimNbPartColl <dbl>,
## #   ClaimNbTotColl <dbl>, ClaimNbFire <dbl>,
## #   ClaimNbOther <dbl>, ClaimAmountRob <dbl>,
## #   ClaimAmountPartColl <dbl>, ...
```

```
brvehins1a %>%
  filter(
    PremTotal > 250e3,
    VehYear == 2011,
    Gender != 'Corporate')
```

Using **stringr** functions with **filter**

```
brvehins1a %>%
  filter(
    PremTotal > 250e3,
    VehYear == 2011,
    !str_starts(Gender, 'C|c'))
```

```
## # A tibble: 12 x 23
##   Gender DrivAge VehYear VehModel       VehGroup Area  State
##   <fct>  <fct>     <int> <fct>          <fct>    <fct> <fct>
## 1 Female 26-35      2011 Ford - Ka 1.0~ Ford Ka~ Met.~ Sao ~
## 2 Female 26-35      2011 Outros         Outros   Goias Goias
## 3 Female 46-55      2011 Outros         Outros   Met.~ Sao ~
## 4 Male   >55        2011 Outros         Outros   Pern~ Pern~
## 5 Male   36-45      2011 Vw - Volkswag~ Vw Volk~ Ribe~ Sao ~
## # ... with 7 more rows, and 16 more variables:
## #   StateAb <fct>, ExposTotal <dbl>, ExposFireRob <dbl>,
## #   PremTotal <dbl>, PremFireRob <dbl>, SumInsAvg <dbl>,
## #   ClaimNbRob <dbl>, ClaimNbPartColl <dbl>,
## #   ClaimNbTotColl <dbl>, ClaimNbFire <dbl>,
## #   ClaimNbOther <dbl>, ClaimAmountRob <dbl>,
## #   ClaimAmountPartColl <dbl>, ...
```

Remove incomplete rows with **drop_na**

Our data frame has 393,071 rows, but 83,484 observations have `NA` values in at least one column. You can drop those rows quickly with `drop_na`. It *may not* be the right thing to do for your analysis, but the power is yours.

```
brvehins1a %>%
  drop_na()
```

If you want to only look at certain columns, you can pass any “tidy select” commands to `drop_na`, too.

```
brvehins1a %>%
  drop_na(Gender, DrivAge, contains('total'))
```

Extract rows using `slice`

```
brvehins1a %>%
  select(ExposTotal, PremTotal) %>%
  slice(10:11)
```

```
## # A tibble: 2 x 2
##   ExposTotal PremTotal
##        <dbl>     <dbl>
## 1       0.93      589.
## 2      10.4     12908.
```

Adding computed colums using `mutate`

```
brvehins1a %>%
  select(ExposTotal, PremTotal) %>%
  mutate(
    PremPerExpos = PremTotal / ExposTotal,
    PremPerExpos_e3 = PremPerExpos / 1000)
```

```
## # A tibble: 393,071 x 4
##   ExposTotal PremTotal PremPerExpos PremPerExpos_e3
##        <dbl>     <dbl>        <dbl>           <dbl>
## 1       1.01      743.         735.           0.735
## 2       3        5026.        1675.           1.68
## 3       1.01      916.         907.           0.907
## 4       1.45     1602.        1105.           1.10
## 5       4.55    53031.       11655.          11.7
## # ... with 393,066 more rows
```

Rename columns using `rename`

```
brvehins1a %>%
  select(DrivAge, State) %>%
  rename(
    DriverAge = DrivAge,
    Province = State) %>%
  filter(Province == 'Sao Paulo')
```

```
## # A tibble: 91,983 x 2
##   DriverAge Province
##   <fct>     <fct>
## 1 36-45     Sao Paulo
## 2 18-25     Sao Paulo
## 3 <NA>      Sao Paulo
## 4 36-45     Sao Paulo
## 5 <NA>      Sao Paulo
## # ... with 91,978 more rows
```

Summary data using `group_by` and `summarize`

```
brvehins1a %>%
  group_by(State) %>%
  summarize(
    PremiumPerState = sum(PremTotal),
    ExposuresPerState = sum(ExposTotal)) %>%
  mutate(
    AvgPremPerExposure = PremiumPerState / ExposuresPerState)
```

```
## # A tibble: 28 x 4
##   State    PremiumPerState ExposuresPerSta~ AvgPremPerExpos~
##   <fct>              <dbl>            <dbl>            <dbl>
## 1 Acre            1899093.            1120.            1696.
## 2 Alagoas        10498583.            8473.            1239.
## 3 Amapa           1044331.             626.            1670.
## 4 Amazonas        6387090.            4737.            1348.
## 5 Bahia          56739968.           44326.            1280.
## # ... with 23 more rows
```

Summarized counts using **n**

```
brvehins1a %>%
  group_by(State) %>%
  summarize(
    StateCount = n()) %>%
  filter()
```

```
## # A tibble: 28 x 2
##   State    StateCount
##   <fct>         <int>
## 1 Acre           1408
## 2 Alagoas        4988
## 3 Amapa           980
## 4 Amazonas       3848
## 5 Bahia         11520
## # ... with 23 more rows
```

Another grouping example

```
brvehins1a %>%
  group_by(Gender) %>%
  summarize(
    Premium = sum(PremTotal),
    Exposures = sum(ExposTotal))
```

```
## # A tibble: 4 x 3
##   Gender       Premium Exposures
##   <fct>          <dbl>     <dbl>
## 1 Corporate 269541892.   149551.
## 2 Female    496045301.   505905.
## 3 Male      699937786.   597197.
## 4 <NA>       14466031.     7938.
```

Dealing with `NA` using `coalesce`

You can use `coalesce` anywhere, even outside of a `dplyr` pipe, to convert `NA` values to a default *in place* without having to use a more verbose option like `ifelse`.

```
brvehins1a %>%
  group_by(Gender) %>%
  summarize(
    Premium = sum(PremTotal),
    Exposures = sum(ExposTotal)) %>%
  mutate(Gender = coalesce(Gender, 'UNKNOWN'))
```

```
## # A tibble: 4 x 3
##   Gender       Premium Exposures
##   <chr>          <dbl>     <dbl>
## 1 Corporate 269541892.   149551.
## 2 Female    496045301.   505905.
## 3 Male      699937786.   597197.
## 4 UNKNOWN    14466031.     7938.
```

Sort using **arrange**

Sort by a column in ascending or descending order. The following is descending. Use `arrange('Exposures')` to get the default ascending order.

```
brvehins1a %>%
  group_by(Gender) %>%
  summarize(
    Premium = sum(PremTotal),
    Exposures = sum(ExposTotal)) %>%
  mutate(Gender = coalesce(Gender, 'UNKNOWN')) %>%
  arrange(desc(Exposures))
```

```
## # A tibble: 4 x 3
##   Gender       Premium Exposures
##   <chr>          <dbl>     <dbl>
## 1 Male      699937786.   597197.
## 2 Female    496045301.   505905.
## 3 Corporate 269541892.   149551.
## 4 UNKNOWN    14466031.     7938.
```

Tidy version of `rbind` with `bind_rows`

The `brvehins1a` data in `CASdatasets` is actually just one-fifth of the full data set. We can get the full data in one object using `bind_rows`.

```
data(
  brvehins1b,
  brvehins1c,
  brvehins1d,
  brvehins1e,
  package = 'CASdatasets')

brvehins1 <- brvehins1a %>%
  bind_rows(brvehins1b, brvehins1c, brvehins1d, brvehins1e)

dim(brvehins1)
```

```
## [1] 1965355      23
```

Combining data

There are several functions for combining data in the `tidyverse`. If you've used SQL, many of these will be familiar.

- `inner_join`
- `left_join`
- `right_join`
- `full_join`
- `anti_join`
- `semi_join`

Population of States in Brazil

Let’s say we wanted to do a crude market penetration analysis and we have a second data set with population by State.

```
pop <- read.csv('brazil-states.csv') %>% as_tibble
print(pop)
```

```
## # A tibble: 27 x 2
##   State    PopThousands
##   <chr>           <int>
## 1 Acre              888
## 2 Alagoas          3334
## 3 Amapa             838
## 4 Amazonas         4147
## 5 Bahia           14897
## # ... with 22 more rows
```

```
sum(pop$PopThousands) / 1000
```

```
## [1] 210.017
```

inner_join

An inner join will return records only when there is a match between rows in both data sets. Joins will match automatically on columns with the same names.

```
count_by_state <- brvehins1 %>%
  group_by(State) %>%
  summarize(Count = n())

count_by_state %>%
  inner_join(pop)
```

```
## # A tibble: 26 x 3
##   State    Count PopThousands
##   <chr>    <int>        <int>
## 1 Acre      6822          888
## 2 Alagoas  24859         3334
## 3 Amapa     5318          838
## 4 Amazonas 20015         4147
## 5 Bahia    58159        14897
## # ... with 21 more rows
```

Specify join parameters

However, you can explicitly state the join-by column(s). This is safer sometimes. Also, it is the only way to do a join if the columns you want to join on have different names.

```
count_by_state %>%
  inner_join(pop, by = 'State')
```

```
## # A tibble: 26 x 3
##   State    Count PopThousands
##   <chr>    <int>        <int>
## 1 Acre      6822          888
## 2 Alagoas  24859         3334
## 3 Amapa     5318          838
## 4 Amazonas 20015         4147
## 5 Bahia    58159        14897
## # ... with 21 more rows
```

Join with different column names

```
count_by_prov <- brvehins1 %>%
  group_by(State) %>%
  summarize(Count = n()) %>%
  rename(Province = State)

count_by_prov %>%
  inner_join(pop, by = c('Province' = 'State'))
```

```
## # A tibble: 26 x 3
##   Province Count PopThousands
##   <chr>    <int>        <int>
## 1 Acre      6822          888
## 2 Alagoas  24859         3334
## 3 Amapa     5318          838
## 4 Amazonas 20015         4147
## 5 Bahia    58159        14897
## # ... with 21 more rows
```

anti_join

This is not a common join in SQL, but it can be really helpful. An anti-join tells what rows are in the left-hand data set but not in the right.

```
count_by_state %>%
  anti_join(pop)
```

```
## # A tibble: 2 x 2
##   State      Count
##   <fct>      <int>
## 1 Sao Paulo 457773
## 2 <NA>          13
```

```
pop %>%
  anti_join(count_by_state)
```

```
## # A tibble: 1 x 2
##   State     PopThousands
##   <chr>            <int>
## 1 São Paulo        45926
```

Left and right joins

Left joins are more common, but both act in a similar way. A left join returns at least one row in the result for every row that exists in the left-hand table, even if there isn't a match.

```
count_by_prov %>%
  left_join(pop, by = c('Province' = 'State')) %>%
  filter(str_starts(Province, 'S'))
```

```
## # A tibble: 3 x 3
##   Province        Count PopThousands
##   <chr>           <int>        <int>
## 1 Santa Catarina 177588         7158
## 2 Sao Paulo      457773           NA
## 3 Sergipe         21825         2303
```

```
count_by_prov %>%
  right_join(pop, by = c('Province' = 'State')) %>%
  filter(str_starts(Province, 'S'))
```

```
## # A tibble: 3 x 3
##   Province        Count PopThousands
##   <chr>           <int>        <int>
## 1 Santa Catarina 177588         7158
## 2 Sergipe         21825         2303
## 3 São Paulo          NA        45926
```

Full joins

Full joins are a mix of right and left. This is sometimes called an outer join in SQL.

```
count_by_state %>%
  full_join(pop, by = 'State') %>%
  filter(str_starts(State, 'S'))
```

```
## # A tibble: 4 x 3
##   State           Count PopThousands
##   <chr>           <int>        <int>
## 1 Santa Catarina 177588         7158
## 2 Sao Paulo      457773           NA
## 3 Sergipe         21825         2303
## 4 São Paulo          NA        45926
```

Pivoting data

Sometimes you want to turn columns into rows or groups of rows into columns. This is called pivoting.

```
tall_data <- brvehins1 %>%
  group_by(Gender, State) %>%
  summarize(PremTotal = sum(PremTotal)) %>%
  drop_na()

tall_data
```

```
## # A tibble: 81 x 3
## # Groups:   Gender [3]
##   Gender    State    PremTotal
##   <fct>     <fct>        <dbl>
## 1 Corporate Acre      1857874.
## 2 Corporate Alagoas   6370776.
## 3 Corporate Amapa     1568820.
## 4 Corporate Amazonas  8298949.
## 5 Corporate Bahia    46443173.
## # ... with 76 more rows
```

Wide data

```
wide_data <- tall_data %>%
  pivot_wider(
    id_cols = 'State',
    names_from = 'Gender',
    values_from = 'PremTotal') %>%
  mutate(
    TotalPrem = Corporate + Female + Male,
    CorporateProp = Corporate / TotalPrem) %>%
  arrange(desc(CorporateProp))

wide_data
```

```
## # A tibble: 27 x 6
##   State      Corporate Female   Male TotalPrem CorporateProp
##   <fct>          <dbl>  <dbl>  <dbl>     <dbl>         <dbl>
## 1 Rondonia      9.83e5 1.10e6 1.32e6    3.40e6         0.289
## 2 Amapa         1.57e6 1.81e6 2.37e6    5.74e6         0.273
## 3 Amazonas      8.30e6 1.06e7 1.30e7    3.20e7         0.260
## 4 Minas Ger~    1.53e8 1.84e8 2.64e8    6.01e8         0.254
## 5 Santa Cat~    7.80e7 1.03e8 1.45e8    3.26e8         0.239
## # ... with 22 more rows
```