

Week 4: Data Wrangling

Recommended Reading

- *Modern Data Science with R* Ch. 4: Data Wrangling
- *Modern Data Science with R* Ch. 5: Tidy Data and Iteration
- Wickham, Hadley. (2014). “Tidy Data”. *Journal of Statistical Software* 59(10). Available on BlueLine.
- <https://srvanderplas.github.io/stat-computing-r-python/part-wrangling/00-wrangling.html>

```
library(tidyverse)
```

Data structure and semantics

- Most statistical datasets are tables made up of *rows* and *columns*. A dataset is a collection of *values*: these can be *numbers* (quantitative) or character *strings* (qualitative)

What is Data Wrangling?

Data Wrangling can be defined as the process of cleaning, organizing, and transforming raw data into the desired format for analysts to use for prompt decision making. Also known as data cleaning.

Why do you need this “Data Wrangling” Skill?

- Data wrangling helps to improve data usability as it converts data into a compatible format for the end system.
- It helps to quickly build data flows within an intuitive user interface and easily schedule and automate the data-flow process.

- Integrates various types of information and their sources (like databases, web services, files, etc.)
- Help users to process very large volumes of data easily and easily share data-flow techniques.

Messy Data

Five main ways tables of data tend not to be tidy:

1. Column headers are values, not variable names.
2. Multiple variables are stored in one column.
3. Variables are stored in both rows and columns.
4. Multiple types of observational units are stored in the same table.
5. A single observational unit is stored in multiple tables.

Tidy data

“Tidy” data is a standard way of mapping the meaning of a dataset to its structure.

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

Any other arrangement of the data is called “messy”.

dplyr

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges

Rules of `dplyr`:

- First argument is always a data frame
- Subsequent arguments say what to do with that data frame
- Always returns a data frame

There are some of the primary dplyr verbs, representing distinct data analysis tasks:

`filter()`: Select specified rows of a data frame, produce subsets

`arrange()`: Reorder the rows of a data frame

`select()`: Select particular columns of a data frame

`mutate()`: Add new or change existing columns of the data frame (as functions of existing columns)

`summarise()`: Create collapsed summaries of a data frame

`group_by`: Introduce structure to a data frame

Example: Gapminder

Gapminder is an independent Swedish foundation with no political, religious or economic affiliations. **Gapminder is a fact tank, not a think tank.** Gapminder fights devastating misconceptions about global development. Gapminder produces free teaching resources making the world understandable based on reliable statistics. Gapminder promotes a fact-based worldview everyone can understand. Gapminder collaborates with universities, UN, public agencies and non-governmental organizations.

```
library(dslabs)
data(gapminder)
glimpse(gapminder)
```

Rows: 10,545

Columns: 9

```
$ country      <fct> "Albania", "Algeria", "Angola", "Antigua and Barbuda"~
$ year         <int> 1960, 1960, 1960, 1960, 1960, 1960, 1960, 1960, 1960,~
$ infant_mortality <dbl> 115.40, 148.20, 208.00, NA, 59.87, NA, NA, 20.30, 37.~
$ life_expectancy <dbl> 62.87, 47.50, 35.98, 62.97, 65.39, 66.86, 65.66, 70.8~
$ fertility     <dbl> 6.19, 7.65, 7.32, 4.43, 3.11, 4.55, 4.82, 3.45, 2.70,~
$ population    <dbl> 1636054, 11124892, 5270844, 54681, 20619075, 1867396,~
$ gdp           <dbl> NA, 13828152297, NA, NA, 108322326649, NA, NA, 966778~
$ continent     <fct> Europe, Africa, Africa, Americas, Americas, Asia, Ame~
$ region        <fct> Southern Europe, Northern Africa, Middle Africa, Cari~
```

1. select():

Picks columns from data frame.

```
gapminder %>%  
  select(gdp, region) %>%  
  head()
```

	gdp	region
1	NA	Southern Europe
2	13828152297	Northern Africa
3	NA	Middle Africa
4	NA	Caribbean
5	108322326649	South America
6	NA	Western Asia

```
gapminder %>%  
  select(region, gdp) %>%  
  head()
```

	region	gdp
1	Southern Europe	NA
2	Northern Africa	13828152297
3	Middle Africa	NA
4	Caribbean	NA
5	South America	108322326649
6	Western Asia	NA

2. filter()

Selects every element of each row with the indicated filter value

```
gapminder2000 <- gapminder %>%  
  filter(year == 2000)  
head(gapminder2000)
```

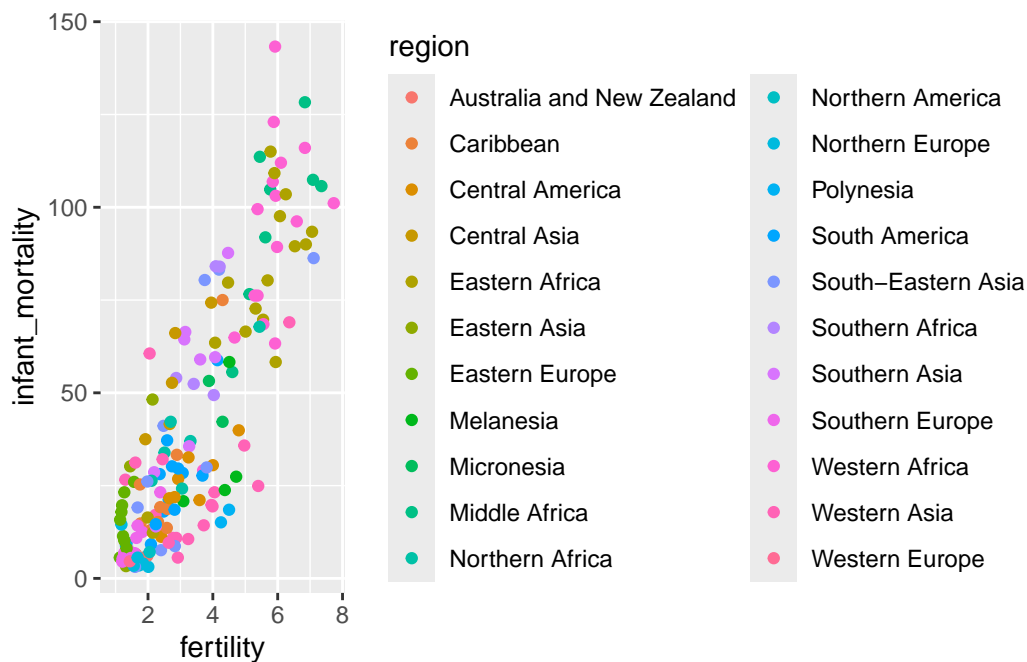
	country	year	infant_mortality	life_expectancy	fertility
1	Albania	2000	23.2	74.7	2.38
2	Algeria	2000	33.9	73.3	2.51

3	Angola	2000	128.3	52.3	6.84
4	Antigua and Barbuda	2000	13.8	73.8	2.32
5	Argentina	2000	18.0	74.2	2.48
6	Armenia	2000	26.6	71.3	1.30

	population	gdp	continent	region
1	3121965	3686649387	Europe	Southern Europe
2	31183658	54790058957	Africa	Northern Africa
3	15058638	9129180361	Africa	Middle Africa
4	77648	802526701	Americas	Caribbean
5	37057453	284203745280	Americas	South America
6	3076098	1911563665	Asia	Western Asia

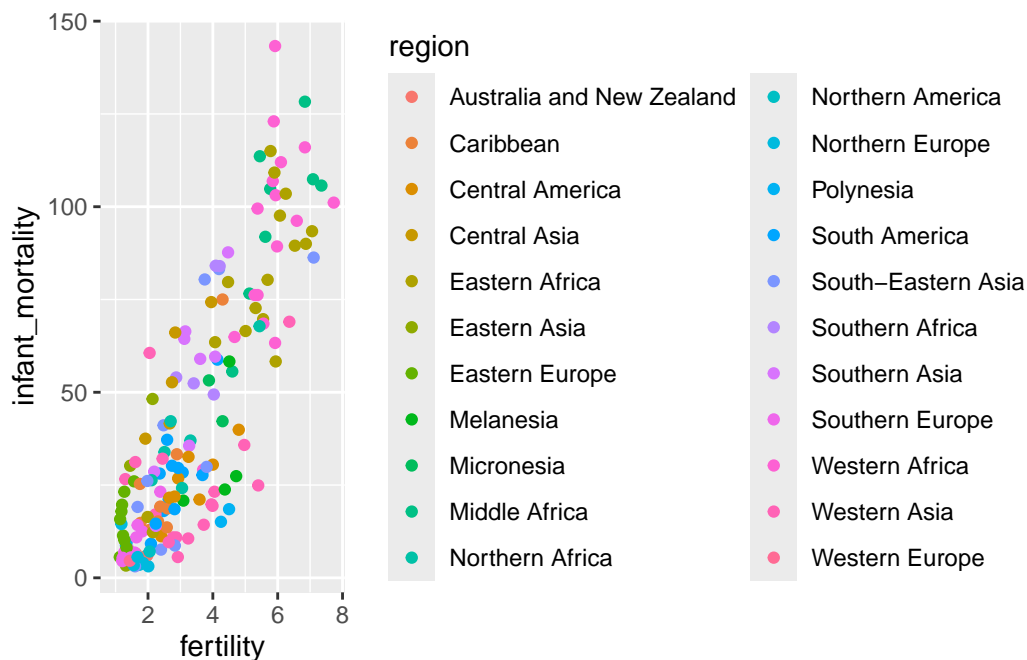
Now, let's take a look at the scatter plot between fertility and infant_mortality for year 2000 only.

```
gapminder2000 %>%
  ggplot(aes(x = fertility, y = infant_mortality)) +
  geom_point(aes(color = region))
```



```
gapminder %>%
  filter(year == 2000) %>%
  ggplot(aes(x = fertility, y = infant_mortality)) +
```

```
geom_point(aes(color = region))
```



3. mutate()

- Change an existing or create a new variable into the data
- create new variables based on manipulations of the old variables
- Great for calculations

Example: We'd like to calculate the gross domestic product per capita. Here are the variables in our data - write an expression to do this calculation.

```
gapminder = gapminder %>% mutate(GDP_pc = gdp/population)
head(gapminder)
```

	country	year	infant_mortality	life_expectancy	fertility
1	Albania	1960	115.40	62.87	6.19
2	Algeria	1960	148.20	47.50	7.65
3	Angola	1960	208.00	35.98	7.32
4	Antigua and Barbuda	1960	NA	62.97	4.43
5	Argentina	1960	59.87	65.39	3.11
6	Armenia	1960	NA	66.86	4.55

	population		gdp	continent	region	GDP_pc
1	1636054		NA	Europe	Southern Europe	NA
2	11124892	13828152297		Africa	Northern Africa	1242.992
3	5270844		NA	Africa	Middle Africa	NA
4	54681		NA	Americas	Caribbean	NA
5	20619075	108322326649		Americas	South America	5253.501
6	1867396		NA	Asia	Western Asia	NA

4. arrange()

A Way to sort/order your data in ascending or descending order

How do we find out which countries have the high GDP per capital? Right now, the data is sorted by country, then year. We could use the `arrange()` command to resort in terms of another variable.

```
gapminder %>%
  select(country, year, GDP_pc, continent) %>%
  arrange(GDP_pc) %>%
  head()
```

	country	year	GDP_pc	continent
1	Liberia	1995	54.88963	Africa
2	Liberia	1996	58.24203	Africa
3	Liberia	1994	59.06140	Africa
4	China	1962	72.36223	Asia
5	Liberia	1993	75.97131	Africa
6	China	1961	78.33912	Asia

```
gapminder %>%
  select(country, year, GDP_pc, continent) %>%
  arrange(desc(GDP_pc)) %>%
  head()
```

	country	year	GDP_pc	continent
1	United Arab Emirates	1980	61340.89	Asia
2	United Arab Emirates	1981	59716.61	Asia
3	United Arab Emirates	1977	58532.69	Asia
4	Luxembourg	2007	57017.14	Europe
5	United Arab Emirates	1975	56279.40	Asia
6	Luxembourg	2008	56218.31	Europe

```
gapminder %>%
  select(country, year, GDP_pc, continent) %>%
  arrange(continent, desc(GDP_pc)) %>%
  head()
```

	country	year	GDP_pc	continent
1	Equatorial Guinea	2011	8527.472	Africa
2	Equatorial Guinea	2009	8519.694	Africa
3	Seychelles	2011	8506.595	Africa
4	Equatorial Guinea	2008	8303.419	Africa
5	Equatorial Guinea	2010	8205.302	Africa
6	Seychelles	2010	8168.815	Africa

5. summarize()

If we want to compare summary statistics, we might use `summarize()`.

```
summary(gapminder$GDP_pc)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
54.89	532.48	1719.03	5583.94	6689.21	61340.89	2972

```
gapminder %>% summarise(avg = mean(GDP_pc),
                        min = min(GDP_pc),
                        max = max(GDP_pc),
                        sd = sd(GDP_pc))
```

	avg	min	max	sd
1	NA	NA	NA	NA

Wait, why are these NAs?

```
gapminder %>% filter(GDP_pc != "NA") %>%
  summarise(avg = mean(GDP_pc),
            min = min(GDP_pc),
            max = max(GDP_pc),
            sd = sd(GDP_pc),
            N = n())
```


	avg	min	max	sd	N
1	5583.936	54.88963	61340.89	8339.741	7573

The `summarize()` function sometimes go with `group_by` function. Instead giving the summary information for the whole data, with a `group_by` function, it provides the summary information by groups.

```
gapminder %>% filter(GDP_pc != "NA") %>%
  group_by(continent) %>%
  summarise(avg = mean(GDP_pc),
            min = min(GDP_pc),
            max = max(GDP_pc),
            sd = sd(GDP_pc),
            N = n())
```

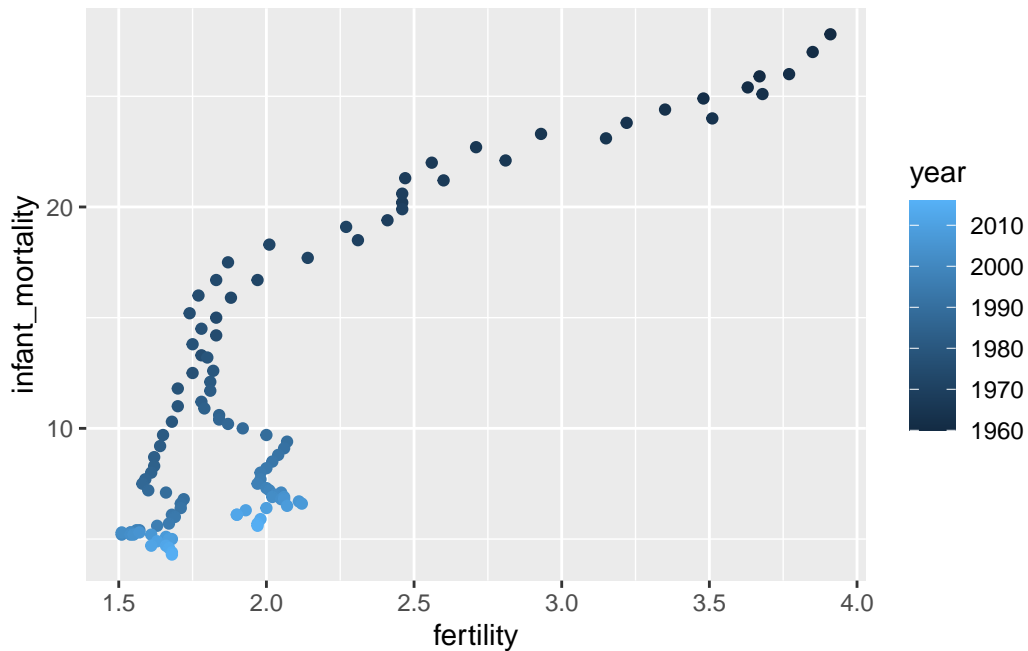
```
# A tibble: 5 x 6
  continent    avg   min    max    sd     N
  <fct>      <dbl> <dbl> <dbl> <dbl> <int>
1 Africa      904.   54.9  8527.  1289.  2270
2 Americas  5426.  370.  38656.  6315.  1703
3 Asia      5979.   72.4 61341.  9547.  1720
4 Europe  12708.  299.  57017. 10506.  1434
5 Oceania   5579.  426.  25439.  6240.   446
```

Try it for yourself

(a). Start with the `gapminder` dataset, filter the data for country United States and Canada, then select fertility, infant mortality and year to be included. Then make a scatterplot of fertility and infant mortality and use color to indicate different years. Note: think about a question, whether the order of filter and select matters?

```
gapminder %>%
  filter(country %in% c("Canada", "United States")) %>%
  select(fertility, infant_mortality, year) %>%
  ggplot(aes(x=fertility, y = infant_mortality, color = year)) +
  geom_point()
```

Warning: Removed 2 rows containing missing values or values outside the scale range (``geom_point()``).



(b). Show the summary statistics (mean, sd, min, max) of GDP_pc for year 2010 for different region.

```
gapminder %>%
  filter(year == 2010) %>%
  group_by(region) %>%
  filter(GDP_pc != "NA") %>%
  summarise(avg = mean(GDP_pc),
            min = min(GDP_pc),
            max = max(GDP_pc),
            sd = sd(GDP_pc),
            N = n())
```

A tibble: 22 x 6

	region <fct>	avg <dbl>	min <dbl>	max <dbl>	sd <dbl>	N <int>
1	Australia and New Zealand	20140.	14875.	25405.	7446.	2
2	Caribbean	7509.	370.	18430.	5765.	11
3	Central America	3462.	915.	5935.	2055.	8
4	Central Asia	1228.	253.	2484.	1003.	5
5	Eastern Africa	1164.	122.	8169.	2331.	15
6	Eastern Asia	21686.	785.	40013.	17569.	6
7	Eastern Europe	4099.	520.	8464.	2797.	10

8	Melanesia	1426.	745.	2219.	627.	4
9	Micronesia	1520.	741.	2299.	1102.	2
10	Middle Africa	2013.	106.	8205.	2816.	8

i 12 more rows

Joining Data

Table joins allow us to combine information stored in different tables, keeping what we need while discarding what we don't

Simple Data Example

```
df1 <- data.frame(
  id = 1:6,
  trt = rep(c("A", "B", "C"),
    rep=c(2,1,3)),
  value = c(5,3,7,1,2,3))
```

```
df1
```

	id	trt	value
1	1	A	5
2	2	B	3
3	3	C	7
4	4	A	1
5	5	B	2
6	6	C	3

```
df2 <- data.frame(
  id=c(4,4,5,5,7,7),
  stress=rep(c(0,1), 3),
  bpm = c(65,125,74,136,48,110))
```

```
df2
```

	id	stress	bpm
1	4	0	65
2	4	1	125
3	5	0	74
4	5	1	136
5	7	0	48
6	7	1	110

left_join()

All elements in the left data set are kept

Mon-matches are filled in by NA

right_join() works symmetric

```
left_join(df1, df2, by="id")
```

	id	trt	value	stress	bpm
1	1	A	5	NA	NA
2	2	B	3	NA	NA
3	3	C	7	NA	NA
4	4	A	1	0	65
5	4	A	1	1	125
6	5	B	2	0	74
7	5	B	2	1	136
8	6	C	3	NA	NA

inner_join()

Only matches from both data sets are kept

```
inner_join(df1, df2, by="id")
```

	id	trt	value	stress	bpm
1	4	A	1	0	65
2	4	A	1	1	125
3	5	B	2	0	74
4	5	B	2	1	136

full_join()

All ids are kept, missings are filled in with NA

```
full_join(df1, df2, by="id")
```

	id	trt	value	stress	bpm
1	1	A	5	NA	NA
2	2	B	3	NA	NA
3	3	C	7	NA	NA
4	4	A	1	0	65
5	4	A	1	1	125
6	5	B	2	0	74
7	5	B	2	1	136
8	6	C	3	NA	NA
9	7	<NA>	NA	0	48
10	7	<NA>	NA	1	110

Traps of joins

Sometimes we unexpectedly cannot match values: missing values, different spelling, ...

Be very aware of things like a trailing or leading space

Join can be along multiple variables, e.g. `by = c("ID", "Date")`

Joining variable(s) can have different names, e.g. `by = c("State" = "Name")`

Always make sure to check dimensions of data before and after a join

Check on missing values; help with that: `anti_join`

anti_join()

Return all rows from x without a match in y

```
anti_join(df1, df2, by="id") # no values for id in df2
```

	id	trt	value
1	1	A	5
2	2	B	3
3	3	C	7
4	6	C	3

```
anti_join(df2, df1, by="id") # no values for id in df1
```

```
id stress bpm
1 7      0 48
2 7      1 110
```

Example: Linking Data: NYC flights

The R package `nycflights13` contains data about all flights that departed one of the three New York City airports (JFK, LGA, and EWR) in 2013. As you can probably imagine, this isn't a small dataset.

```
#install.packages('nycflights13')
library(nycflights13)
data(flights)

names(flights)
```

```
[1] "year"           "month"          "day"            "dep_time"
[5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
[9] "arr_delay"      "carrier"        "flight"         "tailnum"
[13] "origin"         "dest"           "air_time"       "distance"
[17] "hour"           "minute"         "time_hour"
```

```
glimpse(flights)
```

Rows: 336,776

Columns: 19

```
$ year      <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2~
$ month     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ day       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ dep_time  <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, ~
$ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, ~
$ dep_delay <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1~
$ arr_time  <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849,~
$ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851,~
$ arr_delay <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -1~
$ carrier   <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "~
```

```

$ flight      <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 4~
$ tailnum     <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N394~
$ origin      <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA",~
$ dest        <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD",~
$ air_time    <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 1~
$ distance    <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, ~
$ hour        <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6~
$ minute      <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0~
$ time_hour   <dtm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 0~

```

Suppose we want to know more about the airline (`carrier`). In the data set, each carrier is stored using a two-letter code.

```
table(flights$carrier)
```

```

      9E      AA      AS      B6      DL      EV      F9      FL      HA      MQ      OO      UA      US
18460 32729    714 54635 48110 54173    685   3260    342 26397    32 58665 20536
      VX      WN      YV
5162 12275    601

```

- Why use a two-letter code instead of the airline name?
- Can we *link* the airline names to the letter codes?

```
data(airlines)
head(airlines)
```

```

# A tibble: 6 x 2
  carrier name
  <chr>    <chr>
1 9E      Endeavor Air Inc.
2 AA      American Airlines Inc.
3 AS      Alaska Airlines Inc.
4 B6      JetBlue Airways
5 DL      Delta Air Lines Inc.
6 EV      ExpressJet Airlines Inc.

```

Use a common variable, called a *key*, to link the data.

```
inner_join()
```

```
flights_carrier = flights %>%  
  inner_join(airlines, by = c("carrier" = "carrier"))
```

Did it work?

```
glimpse(flights_carrier)
```

Rows: 336,776

Columns: 20

```
$ year      <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2~  
$ month     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~  
$ day       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~  
$ dep_time  <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, ~  
$ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, ~  
$ dep_delay <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1~  
$ arr_time  <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849,~  
$ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851,~  
$ arr_delay <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -1~  
$ carrier   <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "~  
$ flight    <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 4~  
$ tailnum   <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N394~  
$ origin    <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA",~  
$ dest      <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD",~  
$ air_time  <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 1~  
$ distance  <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, ~  
$ hour      <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6~  
$ minute    <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0~  
$ time_hour <dtm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 0~  
$ name      <chr> "United Air Lines Inc.", "United Air Lines Inc.", "Amer~
```

Your Turn: Departure delays on United

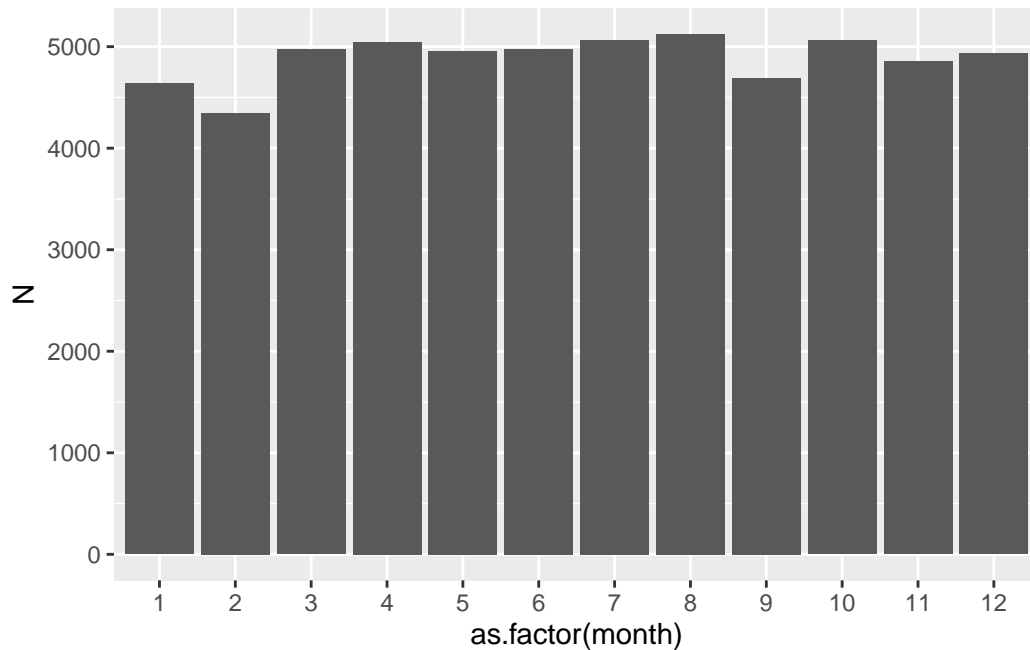
1. Create a new data set, `flights2` that contains the carrier name, year, month, day, departure delay, arrival delay, origin airport, destination airport, and flight number.

```
flights2 <- flights %>%  
  select(carrier, year, month, day, dep_delay, arr_delay, origin, dest, flight)
```


2. Filter the data set to only show United Airlines flights. What month in 2013 did United Airlines have the most flights from the New York Area?

```
flightsUA <- flights2 %>% filter(carrier == "UA")

flightsUA %>%
  group_by(month) %>%
  summarize(N = n()) %>%
  ggplot(aes(x = as.factor(month), y = N)) + geom_col()
```



3. How many unique destinations does United Airlines serve from the New York Area?

```
flightsUA %>%
  group_by(dest) %>%
  summarise(N = n())
```

```
# A tibble: 47 x 2
  dest      N
  <chr> <int>
1 ANC      8
2 ATL     103
3 AUS     670
```

```

4 BDL      8
5 BOS    3342
6 BQN    297
7 BZN     36
8 CHS      1
9 CLE   1890
10 CLT      2
# i 37 more rows

```

4. How many unique destinations does United Airlines run from each of the three area airports?

```

flightsUA %>%
  group_by(origin, dest) %>%
  summarise(N = n())

```

`summarise()` has grouped output by 'origin'. You can override using the `.groups` argument.

```

# A tibble: 53 x 3
# Groups:   origin [3]
  origin dest      N
  <chr>   <chr> <int>
1 EWR    ANC      8
2 EWR    ATL    103
3 EWR    AUS    670
4 EWR    BDL      8
5 EWR    BOS   3342
6 EWR    BQN    297
7 EWR    BZN     36
8 EWR    CHS      1
9 EWR    CLE   1585
10 EWR    CLT      2
# i 43 more rows

```

5. What is the average departure delay of a United Airlines flight leaving any New York area airport?

```

flightsUA %>%
  filter(!is.na(dep_delay)) %>%
  summarize(dep_delay = mean(dep_delay))

```

```
# A tibble: 1 x 1
  dep_delay
  <dbl>
1      12.1
```

6. What is the average departure delay of a United Airlines flight leaving JFK? LGA? EWR?

```
flightsUA %>%
  filter(!is.na(dep_delay)) %>%
  group_by(origin) %>%
  summarize(dep_delay = mean(dep_delay))
```

```
# A tibble: 3 x 2
  origin dep_delay
  <chr>     <dbl>
1 EWR      12.5
2 JFK       7.90
3 LGA      12.1
```