Introduction to data

Some define statistics as the field that focuses on turning information into knowledge. The first step in that process is to summarize and describe the raw information – the data. In this lab we explore flights, specifically a random sample of domestic flights that departed from the three major New York City airports in 2013. We will generate simple graphical and numerical summaries of data on these flights and explore delay times. Since this is a large data set, along the way you'll also learn the indispensable skills of data processing and subsetting.

Getting started

Load packages

In this lab, we will explore and visualize the data using the **tidyverse** suite of packages. The data can be found in the companion package for OpenIntro labs, **openintro**.

Let's load the packages.

```
library(tidyverse)
library(openintro)
```

The data

The Bureau of Transportation Statistics (BTS) is a statistical agency that is a part of the Research and Innovative Technology Administration (RITA). As its name implies, BTS collects and makes transportation data available, such as the flights data we will be working with in this lab.

First, we'll view the `nycflights` data frame. Type the following in your console to load the data:

```
data(nycflights)
```

The data set `nycflights` that shows up in your workspace is a *data matrix*, with each row representing an *observation* and each column representing a *variable*. R calls this data format a **data frame**, which is a term that will be used throughout the labs. For this data set, each *observation* is a single flight.

To view the names of the variables, type the command

```
names(nycflights)
```

```
##  [1] "year"      "month"     "day"       "dep_time"  "dep_delay" "arr_time"
##  [7] "arr_delay" "carrier"   "tailnum"   "flight"    "origin"    "dest"
## [13] "air_time"  "distance"  "hour"      "minute"
```

This returns the names of the variables in this data frame. The **codebook** (description of the variables) can be accessed by pulling up the help file:

```
?nycflights
```

One of the variables refers to the carrier (i.e. airline) of the flight, which is coded according to the following system.

- `carrier`: Two letter carrier abbreviation.
    - `9E`: Endeavor Air Inc.
    - `AA`: American Airlines Inc.

    - AS: Alaska Airlines Inc.
    - B6: JetBlue Airways
    - DL: Delta Air Lines Inc.
    - EV: ExpressJet Airlines Inc.
    - F9: Frontier Airlines Inc.
    - FL: AirTran Airways Corporation
    - HA: Hawaiian Airlines Inc.
    - MQ: Envoy Air
    - OO: SkyWest Airlines Inc.
    - UA: United Air Lines Inc.
    - US: US Airways Inc.
    - VX: Virgin America
    - WN: Southwest Airlines Co.
    - YV: Mesa Airlines Inc.

Remember that you can use `glimpse` to take a quick peek at your data to understand its contents better.

```
glimpse(nycflights)
```

```
## Rows: 32,735
## Columns: 16
## $ year      <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, ~
## $ month     <int> 6, 5, 12, 5, 7, 1, 12, 8, 9, 4, 6, 11, 4, 3, 10, 1, 2, 8, 10~
## $ day       <int> 30, 7, 8, 14, 21, 1, 9, 13, 26, 30, 17, 22, 26, 25, 21, 23, ~
## $ dep_time  <int> 940, 1657, 859, 1841, 1102, 1817, 1259, 1920, 725, 1323, 940~
## $ dep_delay <dbl> 15, -3, -1, -4, -3, -3, 14, 85, -10, 62, 5, 5, -2, 115, -4, ~
## $ arr_time  <int> 1216, 2104, 1238, 2122, 1230, 2008, 1617, 2032, 1027, 1549, ~
## $ arr_delay <dbl> -4, 10, 11, -34, -8, 3, 22, 71, -8, 60, -4, -2, 22, 91, -6, ~
## $ carrier   <chr> "VX", "DL", "DL", "DL", "9E", "AA", "WN", "B6", "AA", "EV", ~
## $ tailnum   <chr> "N626VA", "N3760C", "N712TW", "N914DL", "N823AY", "N3AXAA", ~
## $ flight    <int> 407, 329, 422, 2391, 3652, 353, 1428, 1407, 2279, 4162, 20, ~
## $ origin    <chr> "JFK", "JFK", "JFK", "JFK", "LGA", "LGA", "EWR", "JFK", "LGA~
## $ dest      <chr> "LAX", "SJU", "LAX", "TPA", "ORF", "ORD", "HOU", "IAD", "MIA~
## $ air_time  <dbl> 313, 216, 376, 135, 50, 138, 240, 48, 148, 110, 50, 161, 87,~
## $ distance  <dbl> 2475, 1598, 2475, 1005, 296, 733, 1411, 228, 1096, 820, 264,~
## $ hour      <dbl> 9, 16, 8, 18, 11, 18, 12, 19, 7, 13, 9, 13, 8, 20, 12, 20, 6~
## $ minute    <dbl> 40, 57, 59, 41, 2, 17, 59, 20, 25, 23, 40, 20, 9, 54, 17, 24~
```

The `nycflights` data frame is a massive trove of information. Let's think about some questions we might want to answer with these data:

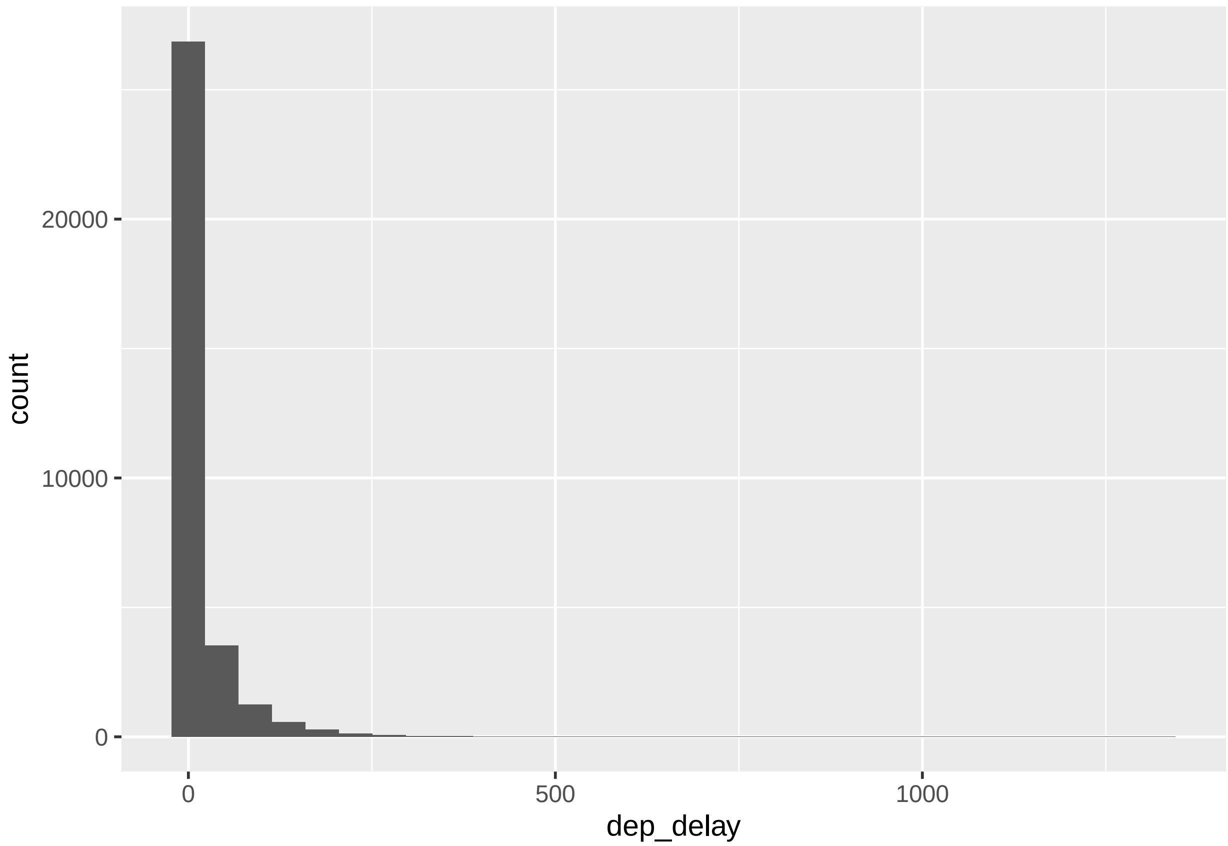
- How delayed were flights that were headed to Los Angeles?
- How do departure delays vary by month?
- Which of the three major NYC airports has the best on time percentage for departing flights?

Analysis

Departure delays

Let's start by examing the distribution of departure delays of all flights with a histogram.

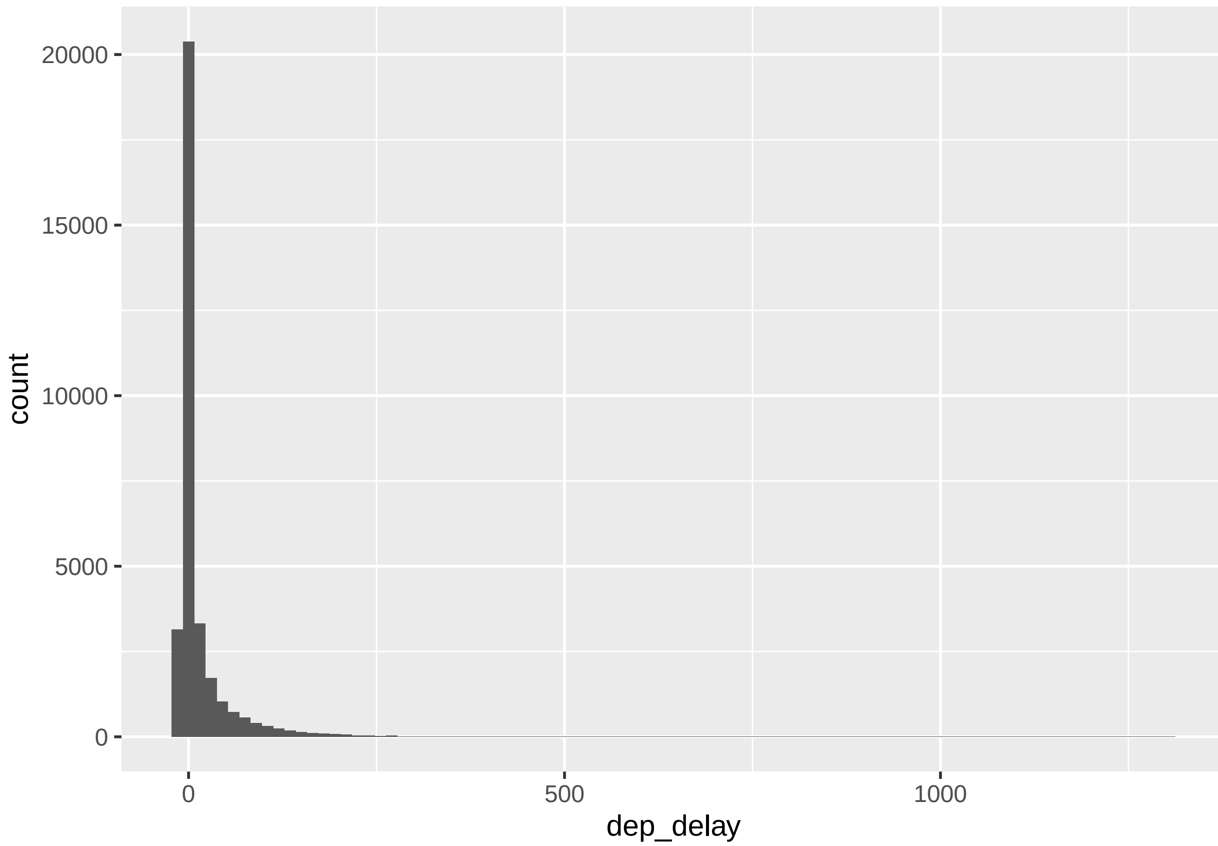
```
ggplot(data = nycflights, aes(x = dep_delay)) +
  geom_histogram()
```



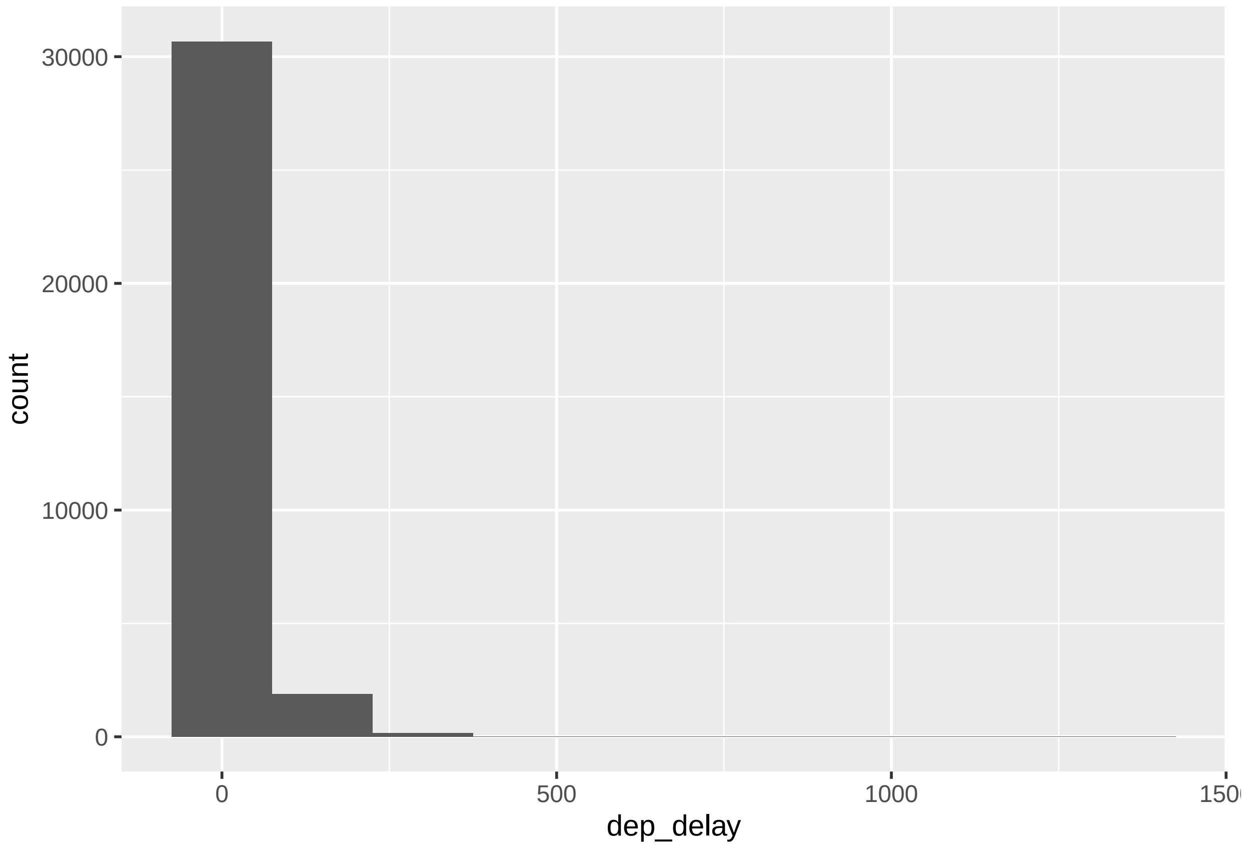
This function says to plot the `dep_delay` variable from the `nycflights` data frame on the x-axis. It also defines a `geom` (short for geometric object), which describes the type of plot you will produce.

Histograms are generally a very good way to see the shape of a single distribution of numerical data, but that shape can change depending on how the data is split between the different bins. You can easily define the binwidth you want to use:

```
ggplot(data = nycflights, aes(x = dep_delay)) +
  geom_histogram(binwidth = 15)
```




```
ggplot(data = nycflights, aes(x = dep_delay)) +
  geom_histogram(binwidth = 150)
```

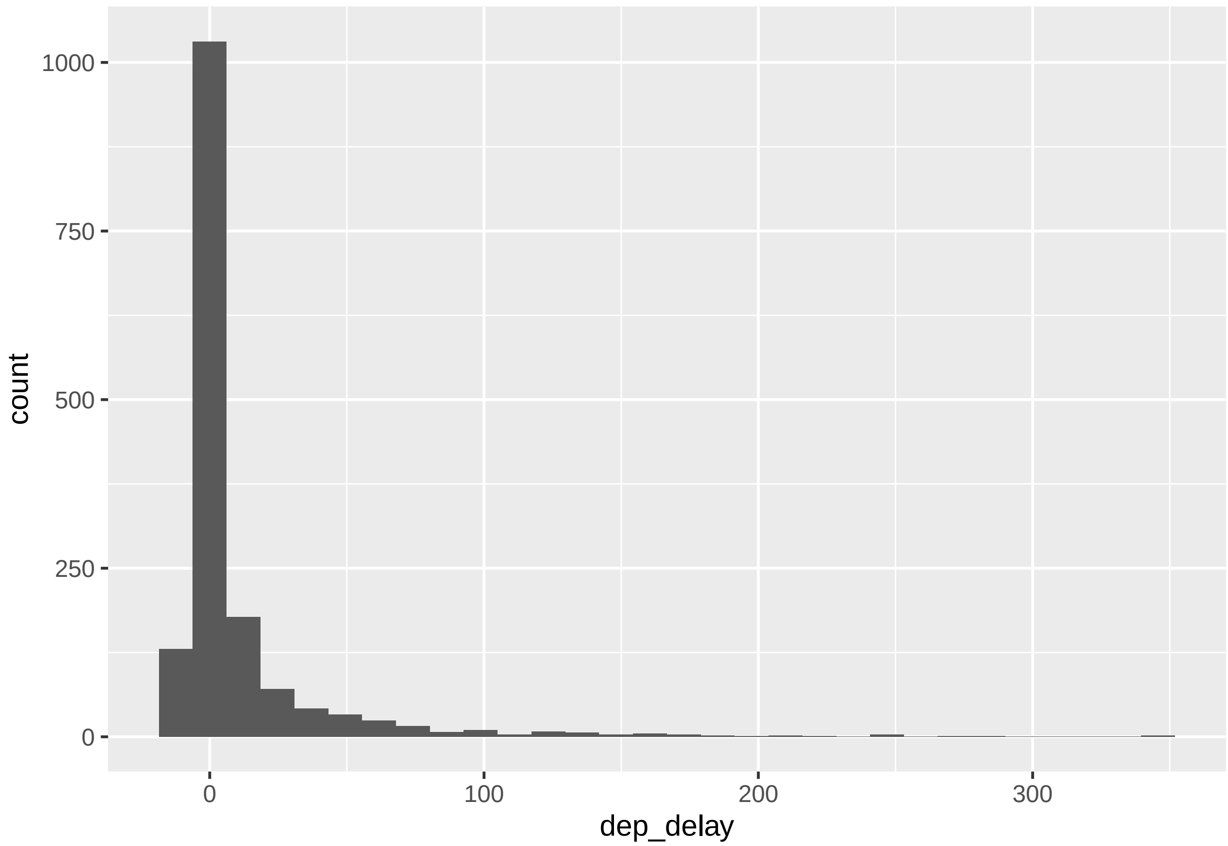


1. Look carefully at these three histograms. How do they compare? Are features revealed in one that are obscured in another?

- These 3 histograms have similar shapes in that the distribution is skewed right with a peak (mode) close to a departure delay of 0. The bin width of each histogram is different, and a smaller bin width (i.e., higher number of bins) shows more granularity in the distribution of departure delay times for instance, the when the `binwidth` is set to 15, we see a strong peak in the histogram at 0. This is harder to see in the other two histograms because departure delays around (but not equal to) 0 are included in that same bin. Smaller bin size would be akin to more ticks on the axis of a dot or boxplot.

If you want to visualize only on delays of flights headed to Los Angeles, you need to first `filter` the data for flights with that destination (`dest == "LAX"`) and then make a histogram of the departure delays of only those flights.

```
lax_flights <- nycflights %>%
  filter(dest == "LAX")
ggplot(data = lax_flights, aes(x = dep_delay)) +
  geom_histogram()
```



Let's decipher these two commands (OK, so it might look like four lines, but the first two physical lines of code are actually part of the same command. It's common to add a break to a new line after `%>%` to help readability).

- Command 1: Take the `nycflights` data frame, `filter` for flights headed to LAX, and save the result as a new data frame called `lax_flights`.
  - `==` means "if it's equal to".
  - `LAX` is in quotation marks since it is a character string.
- Command 2: Basically the same `ggplot` call from earlier for making a histogram, except that it uses the smaller data frame for flights headed to LAX instead of all flights.

Logical operators: Filtering for certain observations (e.g. flights from a particular airport) is often of interest in data frames where we might want to examine observations with certain characteristics separately from the rest of the data. To do so, you can use the `filter` function and a series of **logical operators**. The most commonly used logical operators for data analysis are as follows:

- `==` means "equal to"
- `!=` means "not equal to"
- `>` or `<` means "greater than" or "less than"
- `>=` or `<=` means "greater than or equal to" or "less than or equal to"

You can also obtain numerical summaries for these flights:

```
lax_flights %>%
  summarise(mean_dd   = mean(dep_delay),
            median_dd = median(dep_delay),
            n         = n())
```

```
## # A tibble: 1 x 3
##   mean_dd median_dd     n
```

```
##     <dbl>     <dbl> <int>
## 1    9.78        -1  1583
```

Note that in the `summarise` function you created a list of three different numerical summaries that you were interested in. The names of these elements are user defined, like `mean_dd`, `median_dd`, `n`, and you can customize these names as you like (just don't use spaces in your names). Calculating these summary statistics also requires that you know the function calls. Note that `n()` reports the sample size.

Summary statistics: Some useful function calls for summary statistics for a single numerical variable are as follows:

- `mean`
- `median`
- `sd`
- `var`
- `IQR`
- `min`
- `max`

Note that each of these functions takes a single vector as an argument and returns a single value.

You can also filter based on multiple criteria. Suppose you are interested in flights headed to San Francisco (SFO) in February:

```
sfo_feb_flights <- nycflights %>%
  filter(dest == "SFO", month == 2)
```

Note that you can separate the conditions using commas if you want flights that are both headed to SFO **and** in February. If you are interested in either flights headed to SFO **or** in February, you can use the `|` instead of the comma.

2. Create a new data frame that includes flights headed to SFO in February, and save this data frame as `sfo_feb_flights`. How many flights meet these criteria?

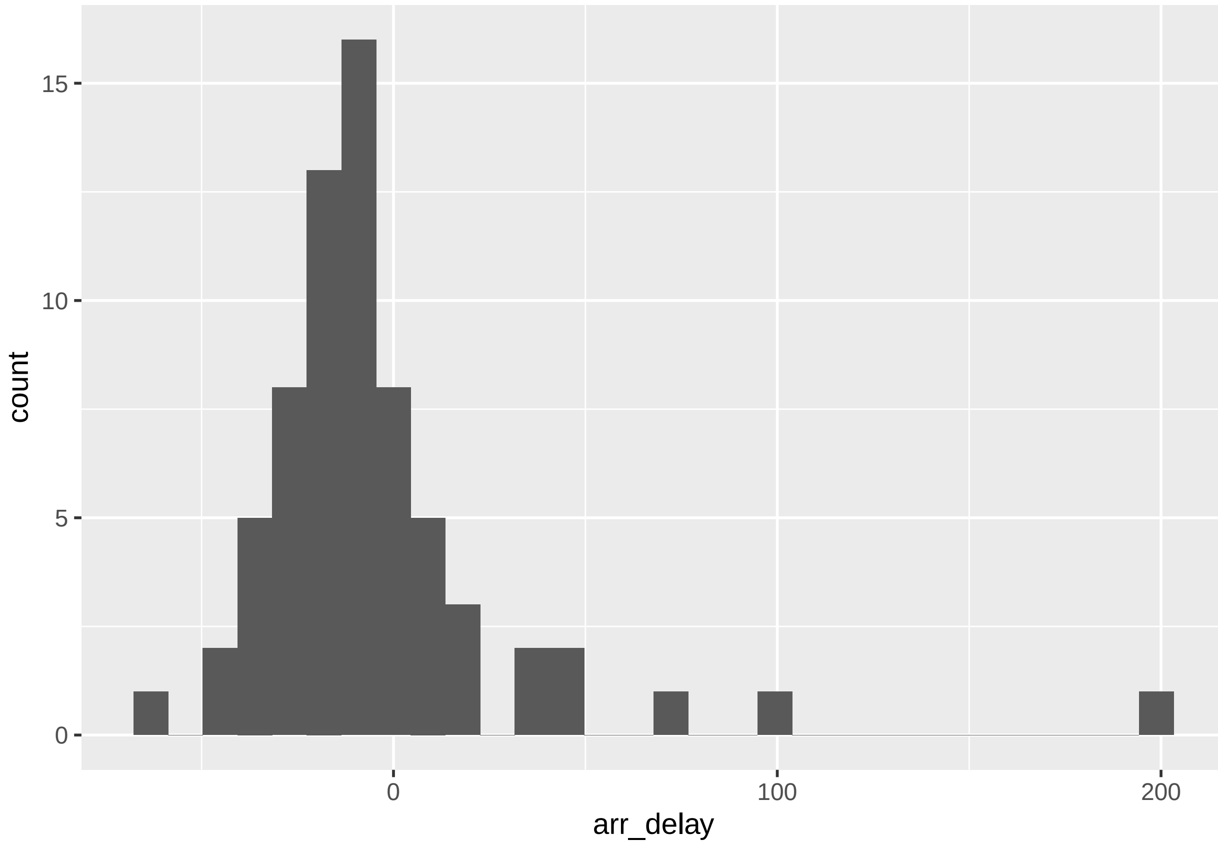
```
# Using the comma to serve as an "AND" operator on our filtering.
sfo_feb_flights <- nycflights %>%
  filter(dest == "SFO", month == 2)

sfo_feb_flights
```

```
## # A tibble: 68 x 16
##     year month   day dep_time dep_delay arr_time arr_de~1 carrier tailnum flight
##    <int> <int> <int>    <int>     <dbl>    <int>    <dbl> <chr>   <chr>    <int>
##  1  2013     2    18     1527        57     1903       48 DL      N711ZX    1322
##  2  2013     2     3      613        14     1008       38 UA      N502UA     691
##  3  2013     2    15      955        -5     1313      -28 DL      N717TW    1765
##  4  2013     2    18     1928        15     2239       -6 UA      N24212    1214
##  5  2013     2    24     1340         2     1644      -21 UA      N76269    1111
##  6  2013     2    25     1415       -10     1737      -13 UA      N532UA     394
##  7  2013     2     7     1032         1     1352      -10 B6      N627JB     641
##  8  2013     2    15     1805        20     2122        2 AA      N335AA     177
##  9  2013     2    13     1056        -4     1412      -13 UA      N532UA     642
## 10  2013     2     8      656        -4     1039       -6 DL      N710TW    1865
## # ... with 58 more rows, 6 more variables: origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, and abbreviated
## #   variable name 1: arr_delay
```

3. Describe the distribution of the **arrival** delays of these flights using a histogram and appropriate summary statistics. **Hint:** The summary statistics you use should depend on the shape of the

distribution.

```
# Arrival delays into SFO
ggplot(sfo_feb_flights, aes(x=arr_delay)) + geom_histogram()
```




```
sfo_feb_flights %>%
  summarise(median_ad = median(arr_delay),
            iqr_ad = IQR(arr_delay),
            n_flights = n())
```

```
## # A tibble: 1 x 3
##   median_ad iqr_ad n_flights
##       <dbl>  <dbl>     <int>
## 1       -11   23.2        68
```

```
sfo_feb_flights
```

```
## # A tibble: 68 x 16
##     year month   day dep_time dep_delay arr_time arr_de~1 carrier tailnum flight
##    <int> <int> <int>    <int>     <dbl>    <int>    <dbl> <chr>   <chr>    <int>
##  1  2013     2    18     1527        57     1903       48 DL      N711ZX    1322
##  2  2013     2     3      613        14     1008       38 UA      N502UA     691
##  3  2013     2    15      955        -5     1313      -28 DL      N717TW    1765
##  4  2013     2    18     1928        15     2239       -6 UA      N24212    1214
##  5  2013     2    24     1340         2     1644      -21 UA      N76269    1111
##  6  2013     2    25     1415       -10     1737      -13 UA      N532UA     394
##  7  2013     2     7     1032         1     1352      -10 B6      N627JB     641
##  8  2013     2    15     1805        20     2122        2 AA      N335AA     177
##  9  2013     2    13     1056        -4     1412      -13 UA      N532UA     642
```

```
## 10  2013     2     8      656        -4     1039       -6 DL      N710TW    1865
## # ... with 58 more rows, 6 more variables: origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, and abbreviated
## #   variable name 1: arr_delay
```

Looking at the histogram of arrival delays into SFO, we see the peak (mode) of the distributrion slightly under 0 minutes. This makes sense for arrival delays if we take those values to be a "negative" delay (i.e., a plane landing early instead of late). Since the distribution is skewed and there's an outlier value at `arr_delay` ~ 100min, we should use median and IQR as summary stats instead of mean/std dev.

Another useful technique is quickly calculating summary statistics for various groups in your data frame. For example, we can modify the above command using the `group_by` function to get the same summary stats for each origin airport:

```
sfo_feb_flights %>%
  group_by(origin) %>%
  summarise(median_dd = median(dep_delay), iqr_dd = IQR(dep_delay), n_flights = n())
```

```
## # A tibble: 2 x 4
##   origin median_dd iqr_dd n_flights
##   <chr>      <dbl>  <dbl>     <int>
## 1 EWR          0.5   5.75         8
## 2 JFK         -2.5  15.2         60
```

Here, we first grouped the data by `origin` and then calculated the summary statistics.

1. Calculate the median and interquartile range for `arr_delay`s of flights in in the `sfo_feb_flights` data frame, grouped by carrier. Which carrier has the most variable arrival delays?

```
# Using arr_delay field instead of dep_delay
sfo_feb_flights %>%
  group_by(origin) %>%
  summarise(median_ad = median(arr_delay), iqr_ad = IQR(arr_delay), n_flights = n())
```

```
## # A tibble: 2 x 4
##   origin median_ad iqr_ad n_flights
##   <chr>      <dbl>  <dbl>     <int>
## 1 EWR        -15.5   17.5         8
## 2 JFK        -10.5   22.8        60
```

Departure delays by month

Which month would you expect to have the highest average delay departing from an NYC airport?

Let's think about how you could answer this question:

- First, calculate monthly averages for departure delays. With the new language you are learning, you could
  - `group_by` months, then
  - `summarise` mean departure delays.
- Then, you could to `arrange` these average delays in `desc`ending order

```
nycflights %>%
  group_by(month) %>%
  summarise(mean_dd = mean(dep_delay)) %>%
  arrange(desc(mean_dd))
```

```
## # A tibble: 12 x 2
##    month mean_dd
```

```
##    <int>   <dbl>
##   1     7   20.8
##   2     6   20.4
##   3    12   17.4
##   4     4   14.6
##   5     3   13.5
##   6     5   13.3
##   7     8   12.6
##   8     2   10.7
##   9     1   10.2
##  10     9    6.87
##  11    11    6.10
##  12    10    5.88
```

5. Suppose you really dislike departure delays and you want to schedule your travel in a month that minimizes your potential departure delay leaving NYC. One option is to choose the month with the lowest mean departure delay. Another option is to choose the month with the lowest median departure delay. What are the pros and cons of these two choices?

We'll want to group our flights dataframe by month and then calculate the mean and media departure delays for each month. It appears that October (month 10) has the lowest mean departure delay, but is tied with September for median departure delay (-3 minutes). The median of a distribution is less resistant to outlier values, so one very long delay could change the mean departure delay much more than the median. Looking at the histograms above, those are also skewed, meaning the mean and median won't necessarily be equal. These two choices do give a good sense of how much a traveler could expect to be delayed on a given flight.

```
nycflights %>%
  group_by(month) %>%
  summarise(mean_dep_delay_by_month = mean(dep_delay), median_dep_delay_by_month = median(dep_delay)) %
  arrange(desc(mean_dep_delay_by_month))
```

```
## # A tibble: 12 x 3
##    month mean_dep_delay_by_month median_dep_delay_by_month
##    <int>                   <dbl>                     <dbl>
##  1     7                   20.8                          0
##  2     6                   20.4                          0
##  3    12                   17.4                          1
##  4     4                   14.6                         -2
##  5     3                   13.5                         -1
##  6     5                   13.3                         -1
##  7     8                   12.6                         -1
##  8     2                   10.7                         -2
##  9     1                   10.2                         -2
## 10     9                    6.87                        -3
## 11    11                    6.10                        -2
## 12    10                    5.88                        -3
```

On time departure rate for NYC airports

Suppose you will be flying out of NYC and want to know which of the three major NYC airports has the best on time departure rate of departing flights. Also supposed that for you, a flight that is delayed for less than 5 minutes is basically "on time."" You consider any flight delayed for 5 minutes of more to be "delayed".

In order to determine which airport has the best on time departure rate, you can

- first classify each flight as "on time" or "delayed",
- then group flights by origin airport,

- then calculate on time departure rates for each origin airport,
- and finally arrange the airports in descending order for on time departure percentage.

Let's start with classifying each flight as "on time" or "delayed" by creating a new variable with the `mutate` function.

```
nycflights <- nycflights %>%
  mutate(dep_type = ifelse(dep_delay < 5, "on time", "delayed"))
```

The first argument in the `mutate` function is the name of the new variable we want to create, in this case `dep_type`. Then if `dep_delay < 5`, we classify the flight as `"on time"` and `"delayed"` if not, i.e. if the flight is delayed for 5 or more minutes.

Note that we are also overwriting the `nycflights` data frame with the new version of this data frame that includes the new `dep_type` variable.

We can handle all of the remaining steps in one code chunk:

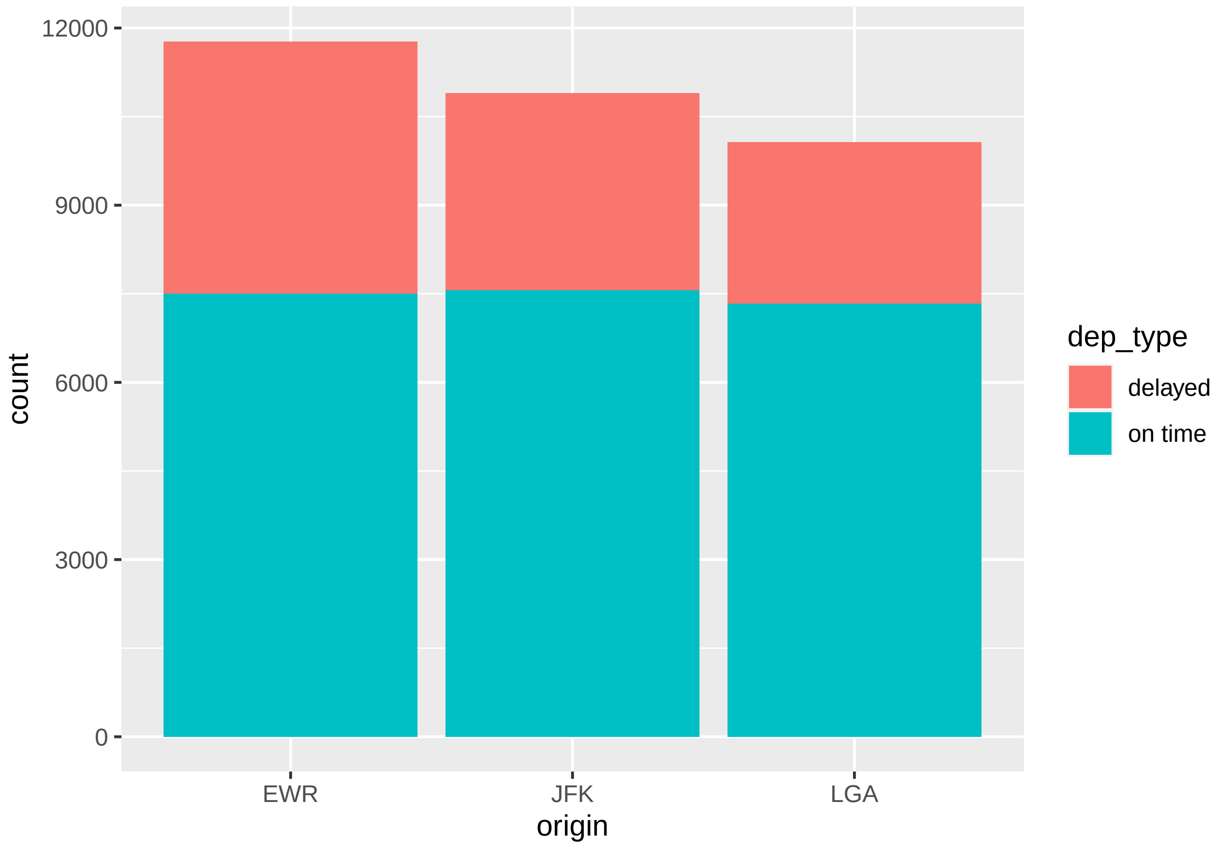
```
nycflights %>%
  group_by(origin) %>%
  summarise(ot_dep_rate = sum(dep_type == "on time") / n()) %>%
  arrange(desc(ot_dep_rate))
```

```
## # A tibble: 3 x 2
##   origin ot_dep_rate
##   <chr>        <dbl>
## 1 LGA          0.728
## 2 JFK          0.694
## 3 EWR          0.637
```

6. If you were selecting an airport simply based on on time departure percentage, which NYC airport would you choose to fly out of?

You can also visualize the distribution of on on time departure rate across the three airports using a segmented bar plot.

```
ggplot(data = nycflights, aes(x = origin, fill = dep_type)) +
  geom_bar()
```



Because the 3 airports have different total numbers of flights departing, we'll want to calculate percentage (delay rate). This can be calculated

```
# Could also calculate the delay rate as 1 - on-time rate, since these are mutually exclusive
nycflights %>%
  group_by(origin) %>%
  summarise(delayed_dep_rate = sum(dep_type == "delayed") / n()) %>%
  arrange(desc(delayed_dep_rate))
```

```
## # A tibble: 3 x 2
##   origin delayed_dep_rate
##   <chr>             <dbl>
## 1 EWR               0.363
## 2 JFK               0.306
## 3 LGA               0.272
```

LaGuardia (LGA) has the highest rate of on-time departures (72.8% of departing flights on-time), so we'll want to choose there if we want the best chance of leaving on time * * *

More Practice

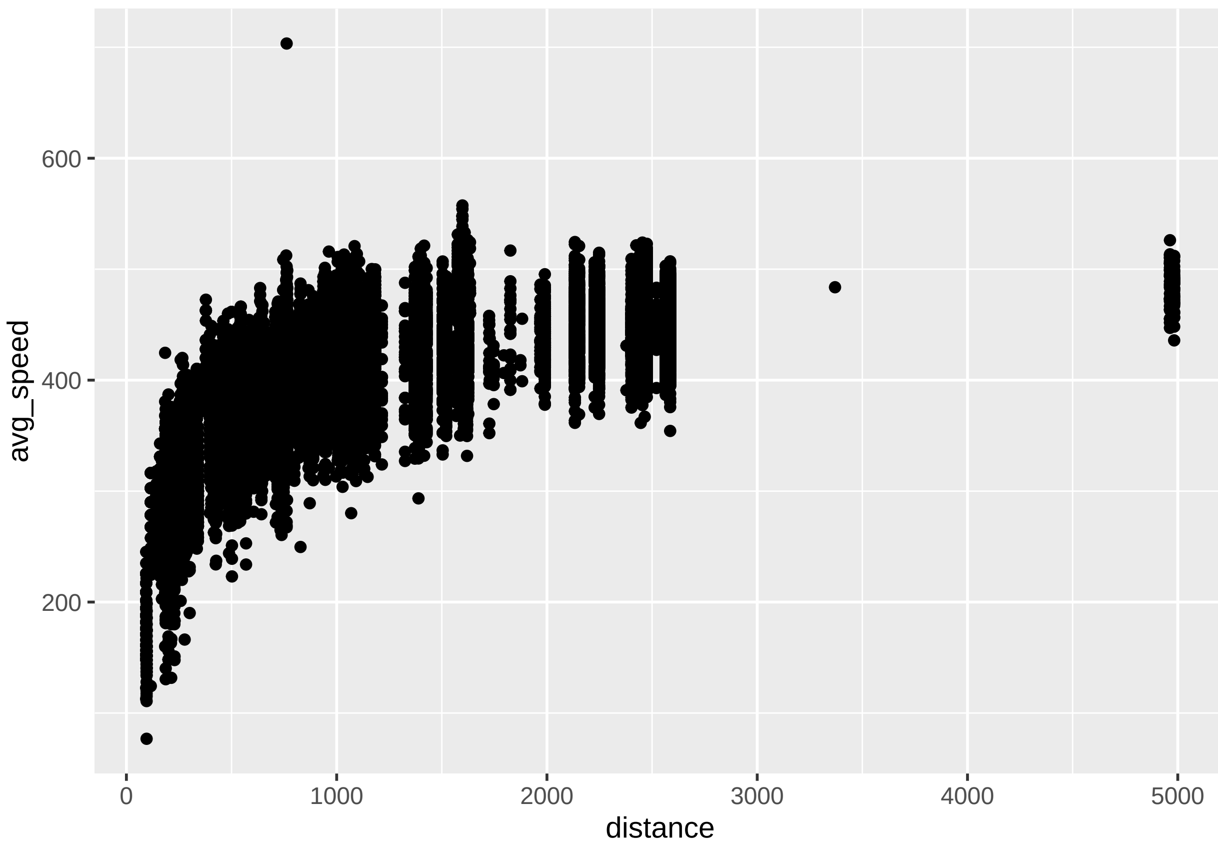
7. Mutate the data frame so that it includes a new variable that contains the average speed, `avg_speed` traveled by the plane for each flight (in mph). **Hint:** Average speed can be calculated as distance divided by number of hours of travel, and note that `air_time` is given in minutes.

```
df <- nycflights %>%
  mutate(avg_speed = distance / (air_time / 60)) # converting travel time to hours

head(df$avg_speed, 10)
```

```
## [1] 474.4409 443.8889 394.9468 446.6667 355.2000 318.6957 352.7500 285.0000
## [9] 444.3243 447.2727
```

8. Make a scatterplot of `avg_speed` vs. `distance`. Describe the relationship between average speed and distance. **Hint:** Use `geom_point()`.

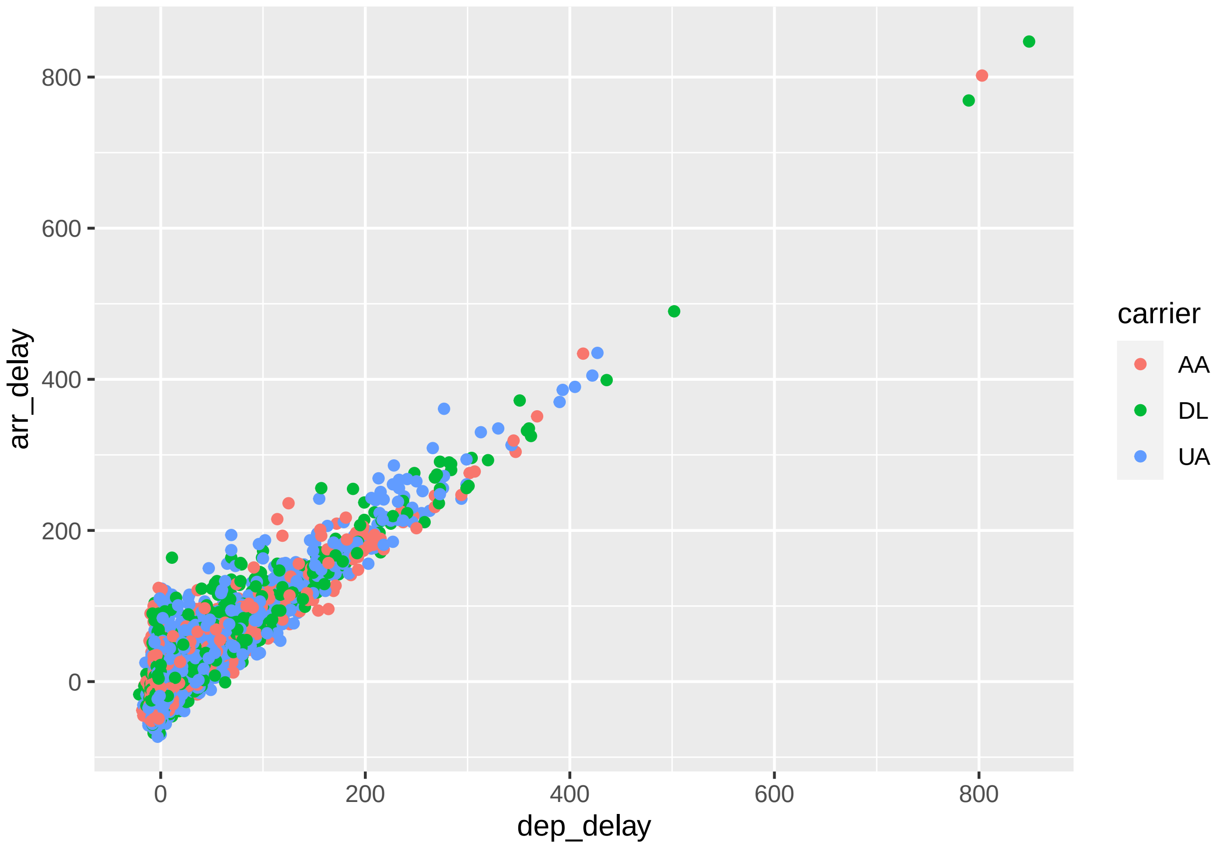
```
ggplot(df, aes(x=distance, y=avg_speed)) + geom_point()
```



As the distance of a flight increases, the average speed of the flight also increases. This relationship occurs on a steeper incline for shorter flights (smaller `distance`) and levels off as flights become longer. There is some variance in average speed between distances of ~1000 mi to ~2000 mi, with some flights travelling at speeds between ~350 mph and~550 mph. Interestingly, there is a dearth of flights between 3000 mi and 5000 mi.

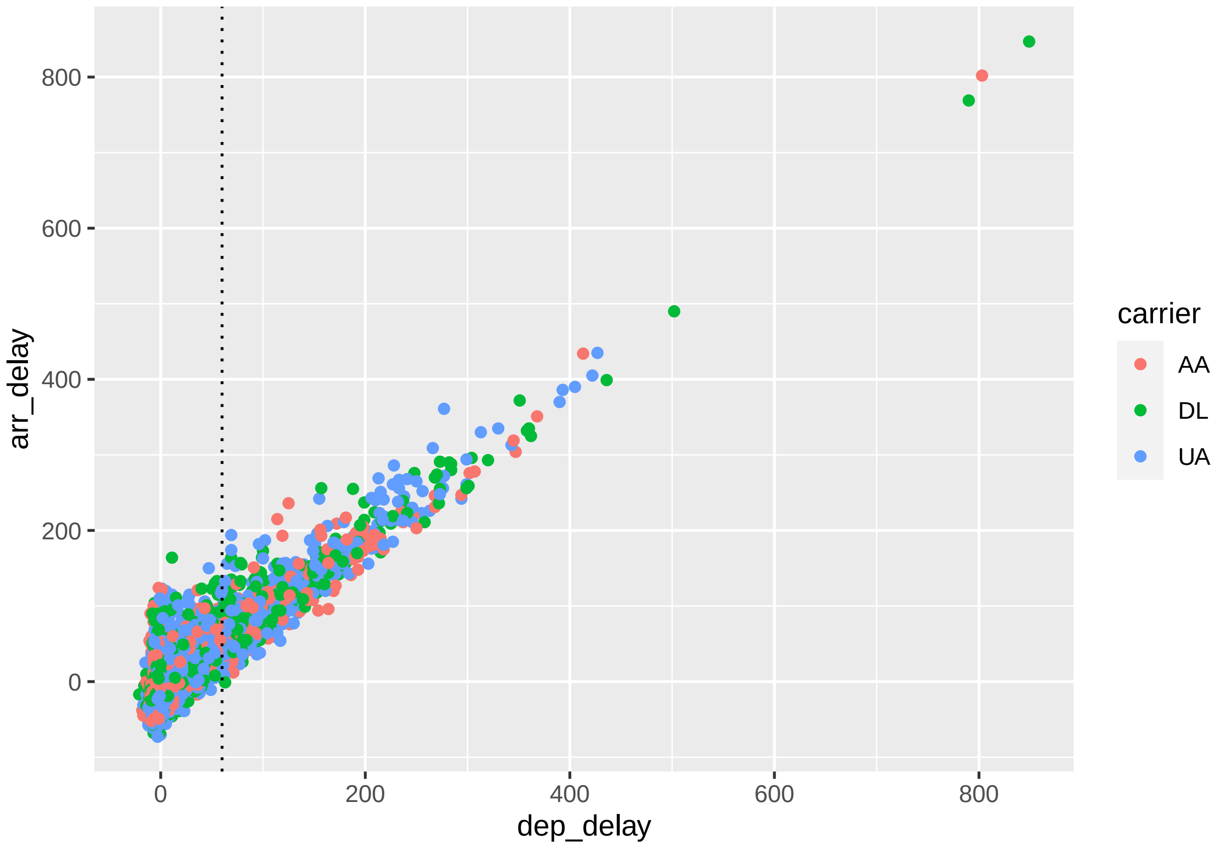
9. Replicate the following plot. **Hint:** The data frame plotted only contains flights from American Airlines, Delta Airlines, and United Airlines, and the points are `colored` by `carrier`. Once you replicate the plot, determine (roughly) what the cutoff point is for departure delays where you can still expect to get to your destination on time.

```
# Filtering our flights df to just the airlines we want
desired_airlines <- nycflights %>%
                        filter(carrier == "AA" | carrier == "DL" | carrier == "UA")

ggplot(desired_airlines, aes(x=dep_delay, y=arr_delay, color=carrier)) + geom_point()
```



It looks like at a `dep_delay` > ~60 minutes you won't be able to arrive to your destination on time. If we draw a vertical line on this plot at ‘$dep_{d}elay$‘ $= 60$ we can see this cutoff more clearly.

```
sp <- ggplot(desired_airlines, aes(x=dep_delay, y=arr_delay, color=carrier)) + geom_point()

sp + geom_vline(xintercept = 60, linetype="dotted")
```


800
600
400
200
0
arr_delay
0
200
400
600
800
dep_delay
carrier
AA
DL
UA