Introduction to data

Kevin Kirby

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)
library(ggplot2)
```

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.

```r
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?

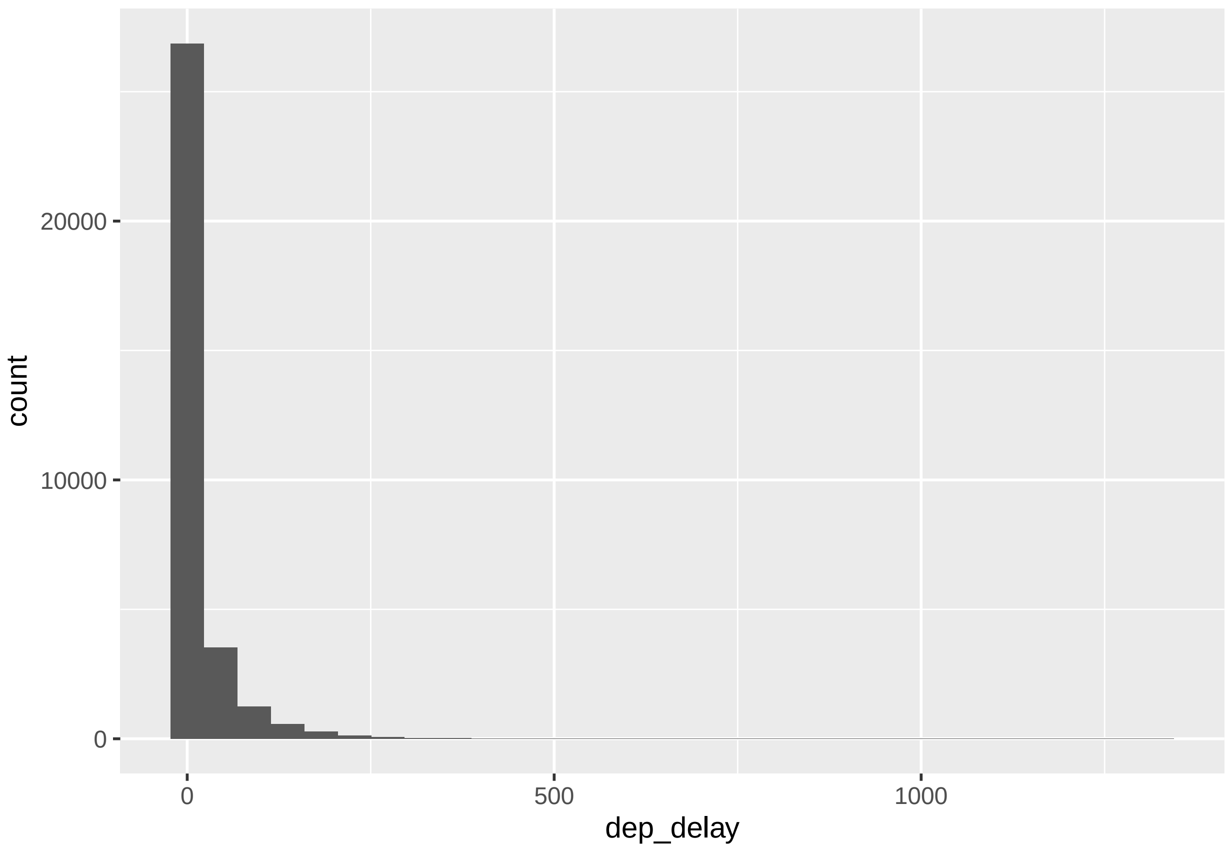
Analysis

Departure delays

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

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

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

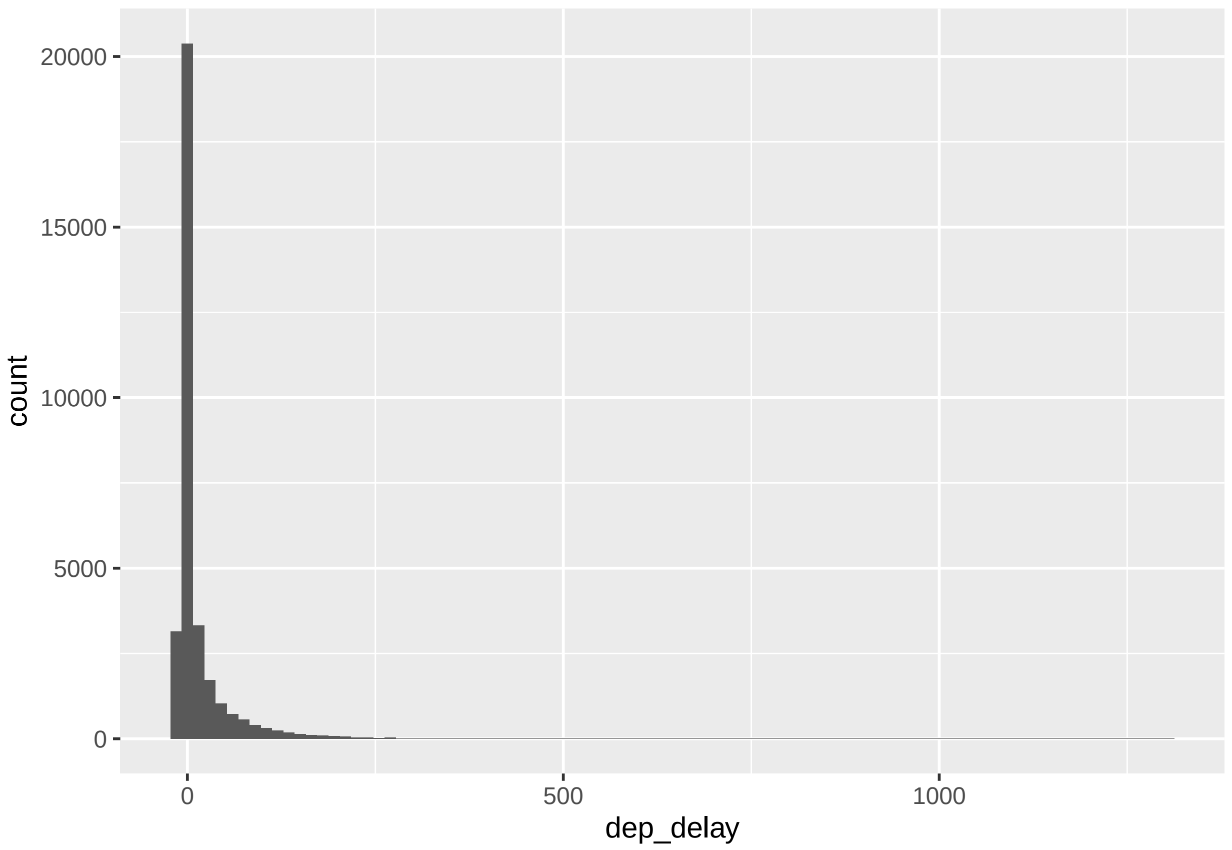


This function says to plot the `dep_delay` variable from the `nycflights` data frame on the x-axis. It also defines a `geom` (short for geometric object), which describes the type of plot you will produce.

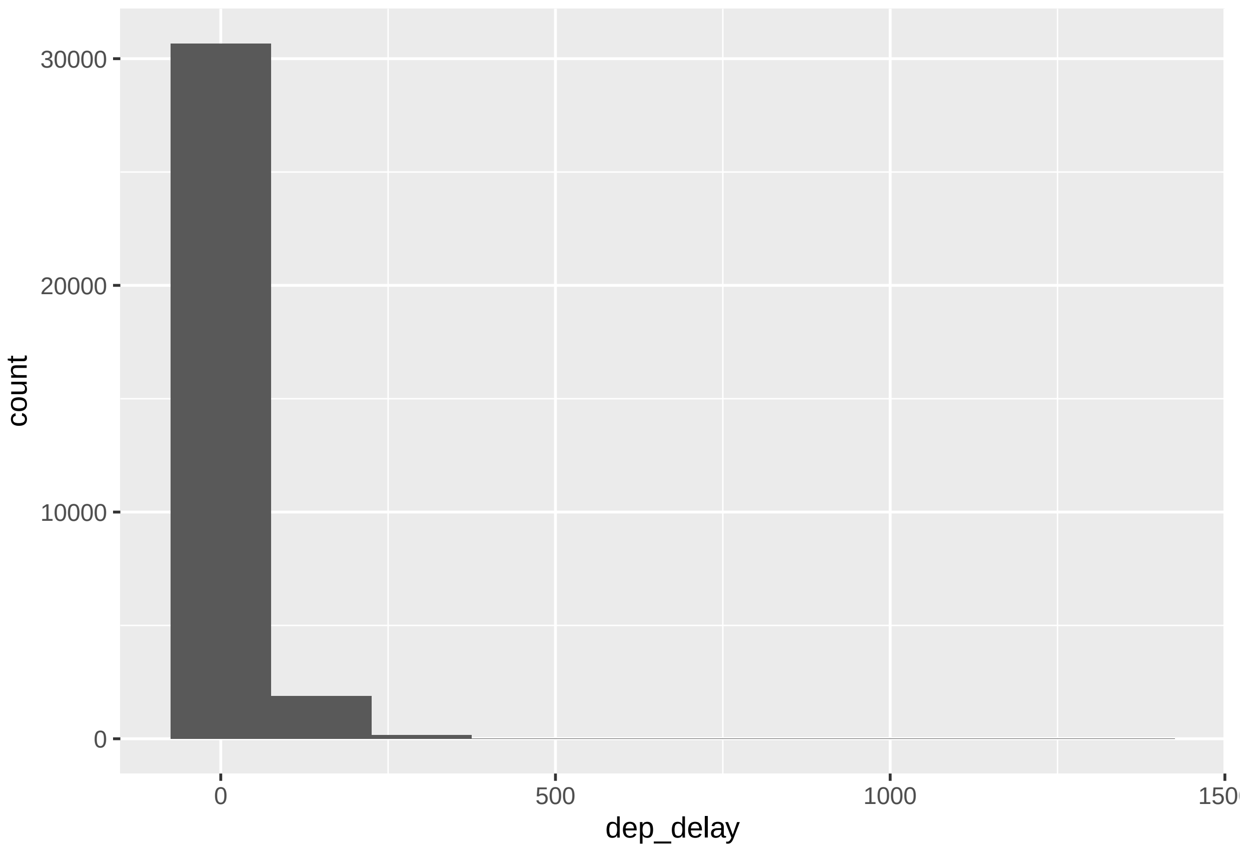
Histograms are generally a very good way to see the shape of a single distribution of numerical data, but that shape can change depending on how the data is split between the different bins. You can easily define the binwidth you want to use:

Kevin Note: I added the 300 one to more a more elongated chart so I could better understand what changes were happening.

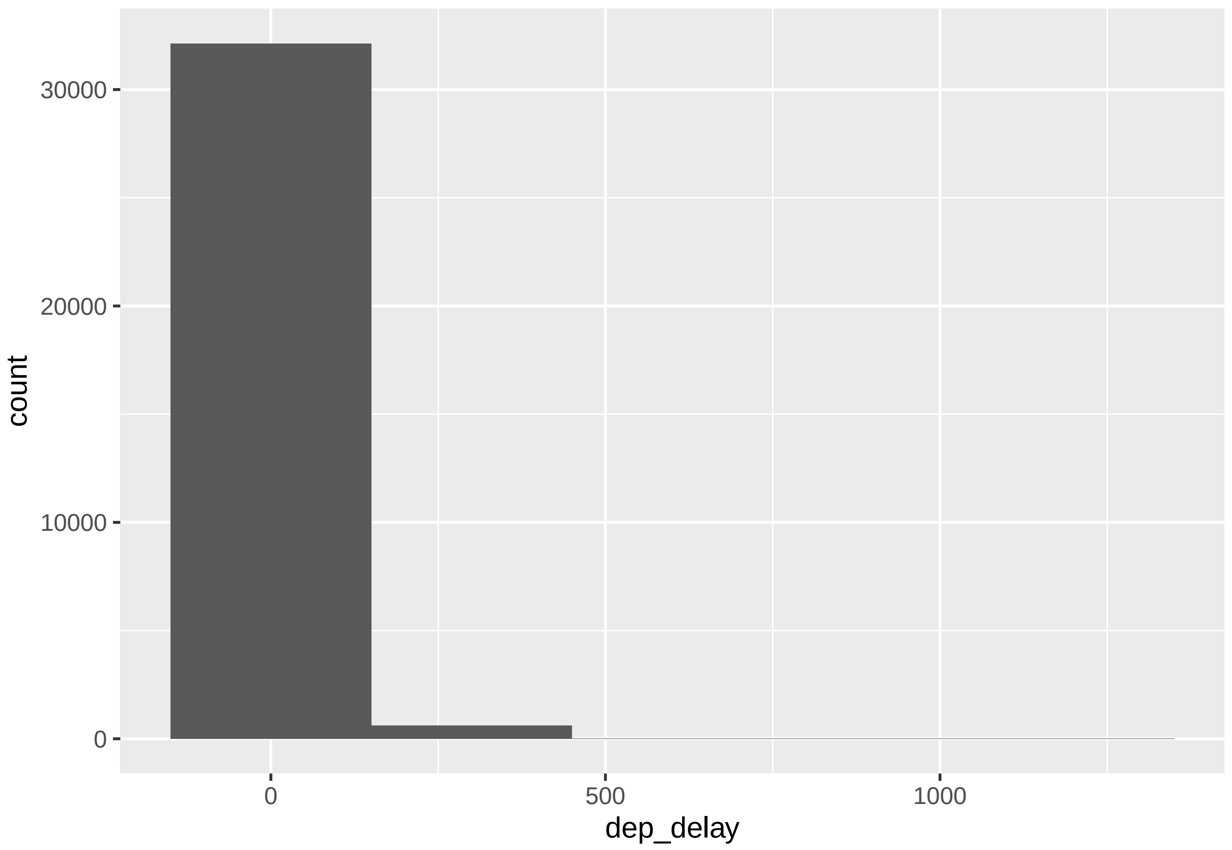
```r
ggplot(data = nycflights, aes(x = dep_delay)) +
  geom_histogram(binwidth = 15)
```



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



```
ggplot(data = nycflights, aes(x = dep_delay)) +
  geom_histogram(binwidth = 300)
```



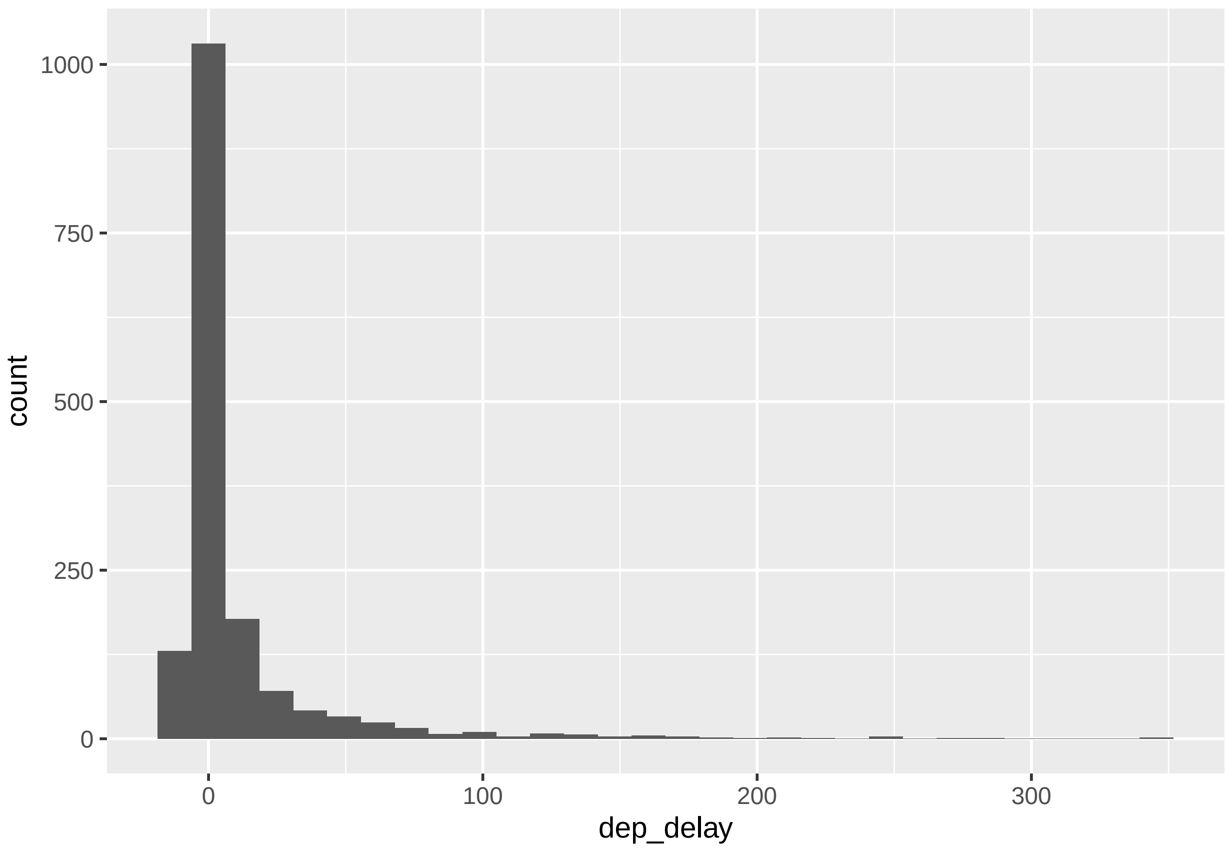
1. Look carefully at these three histograms. How do they compare? Are features revealed in one that are obscured in another?

The first two charts are the best, where no bid width is specific or bins are set to 15. There's a long time and step pattern in the data as the departure delays get longer that are hidden from the 150 and 300 width I added. There's also an interesting shorter bar shown on the 15 minute's x axis below zero, suggesting a small number of early departures.The step pattern on the 150 width really makes me want to unpack the steps and see what's going on.

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()
```

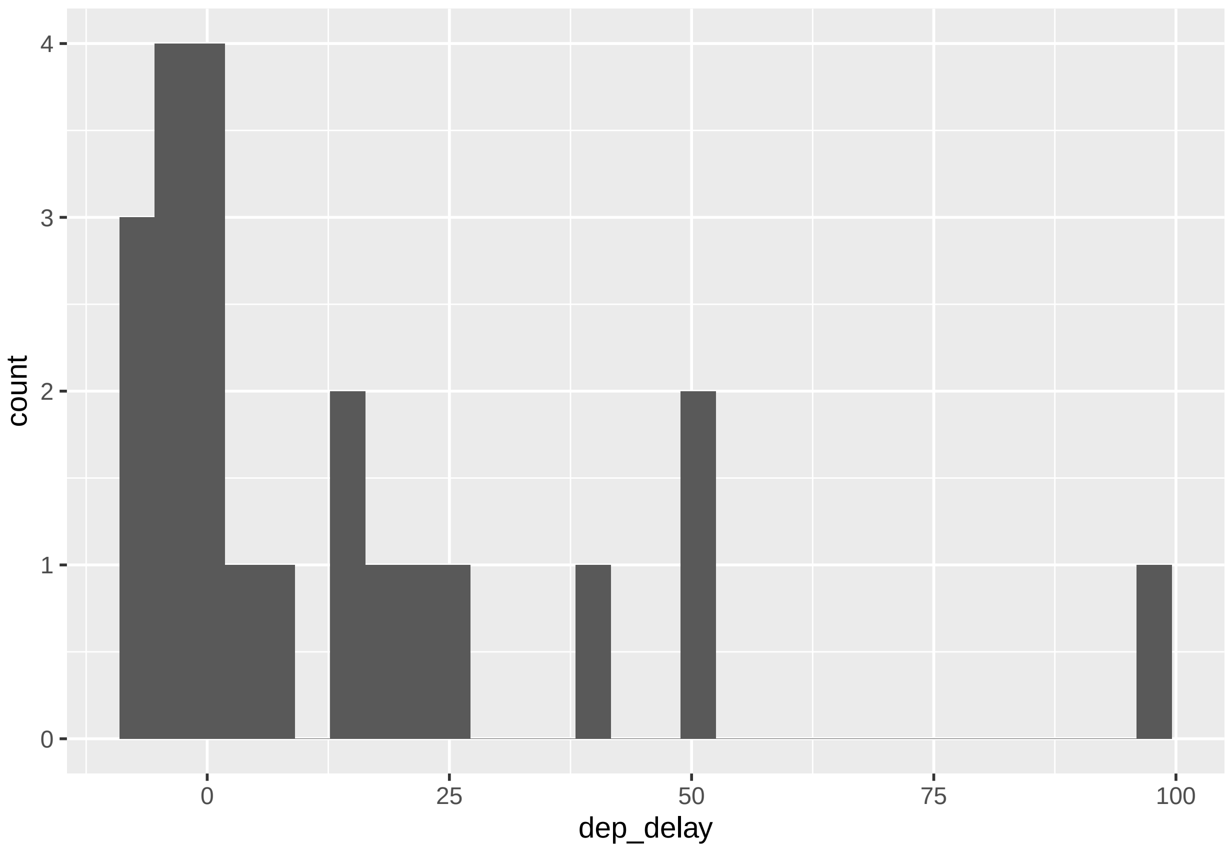
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Kevin Note: I don't like LA so I added a chart for Albuquerque, New Mexico because it reminded me of Breaking Bad, a very good TV show. Additionally, I suspected it would have a bit more chaotic of a chart since there are less flights there.

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

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



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_lax   = mean(dep_delay),
            median_lax = median(dep_delay),
            iqr_lax = IQR(dep_delay),
            n          = n())
```

```
## # A tibble: 1 x 4
##   mean_lax median_lax iqr_lax     n
##      <dbl>      <dbl>   <dbl> <int>
## 1     9.78         -1      11  1583
```

Kevin Note: adding same stats for ABQ since I'm committed to the bit. I also added some other summary stats I wanted to see for both LA and ABQ.

```
abq_flights %>%
  summarise(mean_abq   = mean(dep_delay),
            median_abq = median(dep_delay),
            iqr_aqb = IQR(dep_delay),
            n          = n())
```

```
## # A tibble: 1 x 4
##   mean_abq median_abq iqr_aqb     n
##      <dbl>      <dbl>   <dbl> <int>
## 1     14.0          2    22.8    22
```

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)
```

Kevin Note: as a boy from Maine, I'm going somewhere very warm in February so here's Miami:

```
mia_feb_flights <- nycflights %>%
  filter(dest == "MIA", 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?

This is a copy of above code chunk for convenience of having the answer in one spot. In addition, I added the glimpse function so I could see row count. There are 68 rows in the subset, which means 68 flights meet the criteria.

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

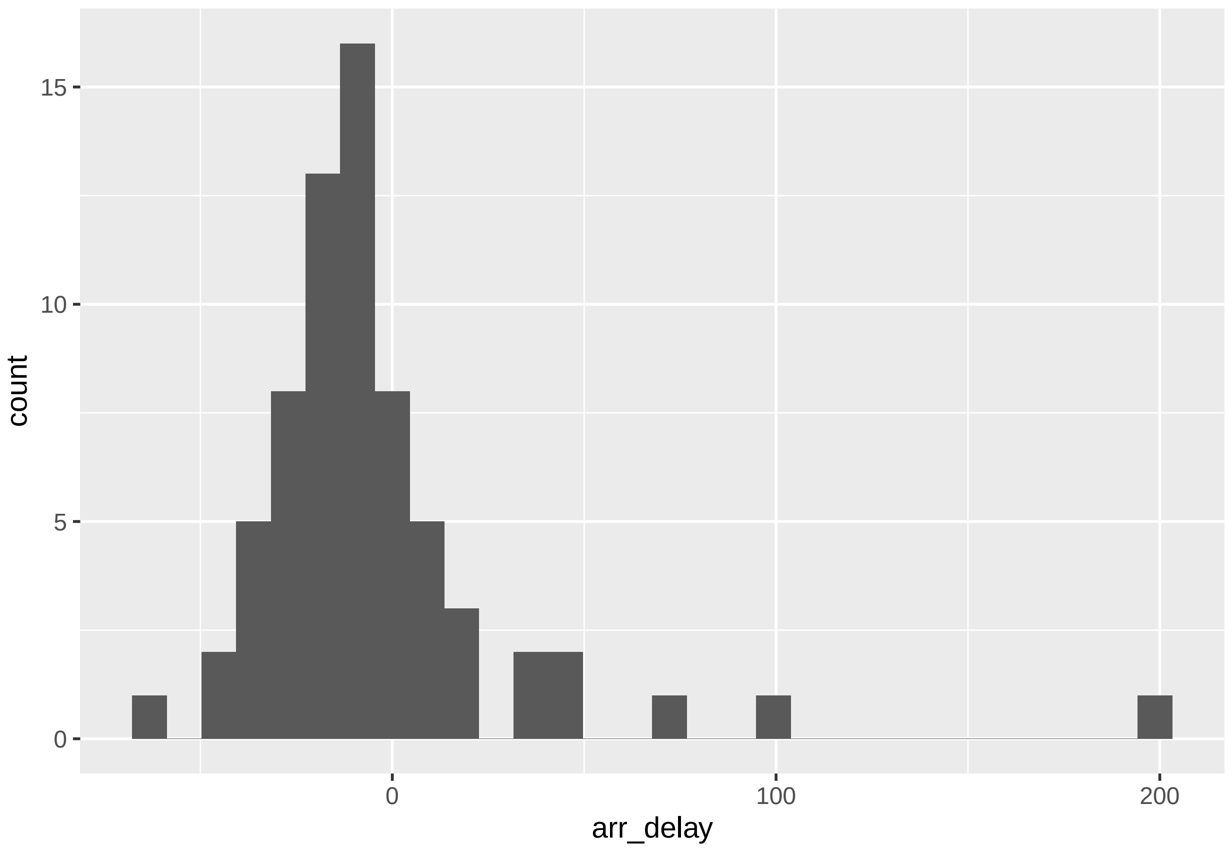
```
## Rows: 68
## Columns: 16
## $ year      <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, ~
## $ month     <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ~
## $ day       <int> 18, 3, 15, 18, 24, 25, 7, 15, 13, 8, 11, 13, 25, 20, 12, 27,~
## $ dep_time  <int> 1527, 613, 955, 1928, 1340, 1415, 1032, 1805, 1056, 656, 191~
## $ dep_delay <dbl> 57, 14, -5, 15, 2, -10, 1, 20, -4, -4, 40, -2, -1, -6, -7, 2~
## $ arr_time  <int> 1903, 1008, 1313, 2239, 1644, 1737, 1352, 2122, 1412, 1039, ~
## $ arr_delay <dbl> 48, 38, -28, -6, -21, -13, -10, 2, -13, -6, 2, -5, -30, -22,~
## $ carrier   <chr> "DL", "UA", "DL", "UA", "UA", "UA", "B6", "AA", "UA", "DL", ~
## $ tailnum   <chr> "N711ZX", "N502UA", "N717TW", "N24212", "N76269", "N532UA", ~
## $ flight    <int> 1322, 691, 1765, 1214, 1111, 394, 641, 177, 642, 1865, 272, ~
## $ origin    <chr> "JFK", "JFK", "JFK", "EWR", "EWR", "JFK", "JFK", "JFK", "JFK~
## $ dest      <chr> "SFO", "SFO", "SFO", "SFO", "SFO", "SFO", "SFO", "SFO", "SFO~
## $ air_time  <dbl> 358, 367, 338, 353, 341, 355, 359, 338, 347, 361, 332, 351, ~
## $ distance  <dbl> 2586, 2586, 2586, 2565, 2565, 2586, 2586, 2586, 2586, 2586, ~
## $ hour      <dbl> 15, 6, 9, 19, 13, 14, 10, 18, 10, 6, 19, 8, 10, 18, 7, 17, 1~
## $ minute    <dbl> 27, 13, 55, 28, 40, 15, 32, 5, 56, 56, 10, 33, 48, 49, 23, 2~
```

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.

First, let's create the histogram:

```r
sfo_feb_flights %>%
  ggplot(aes(x = arr_delay)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Based on this distribution, I would Like to know:

- Median: There's a lot of negative values and I would like to see where the median in. The presence of the outlier at 200 minutes on a small subset like this means the mean will be dragged higher. I'm going to add mean anyway for reference. *Variance: the awkardly distributed tails extending out to the right means there's probably a high variance.* Interquartile Range: I'm expecting the middle 50% to be relatively small, maybe under 30 minutes.

```
sfo_feb_flights %>%
  summarise(mean_sfofeb   = mean(arr_delay),
            median_sfofeb = median(arr_delay),
            iqr_sfofeb = IQR(arr_delay),
            var_sfofeb = var(arr_delay),
            n_sfofed  =   n())
```

```
## # A tibble: 1 x 5
##   mean_sfofeb median_sfofeb iqr_sfofeb var_sfofeb n_sfofed
##         <dbl>         <dbl>      <dbl>      <dbl>    <int>
## 1        -4.5           -11       23.2      1316.       68
```

The distribution is bottom heavy, with a larger than expected number of negative values. Both New York and San Francisco are busy and delayed airports so I would expect more delayed arrivals.However, the airlines have gotten a lot better at padding flight times so, relative to when they officially say they will arrive, they get there early.

The mean is being dragged upwards by the 200 minute delayed flight, which is also causing a high variance. I would describe the overall distribution as normal-ish. I suspect, if there were more data points available, you would see the tail out towards 100 fill out a bit more. That 200 one would remain an outlier and, in future analysis, would need to be handled.

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.

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

First, I'll calculate the required stats and also include a sort by largest mechanism on n_flights so the airline with the most values rises to the top.

```
sfo_feb_flights %>%
  group_by(carrier) %>%
  summarise(median_dd = median(arr_delay),
            iqr_dd = IQR(arr_delay),
            n_flights = n()) %>%
  arrange(desc(n_flights))
```

```
## # A tibble: 5 x 4
##   carrier median_dd iqr_dd n_flights
##   <chr>       <dbl>  <dbl>     <int>
## 1 UA          -10     22          21
## 2 DL          -15     22          19
## 3 VX          -22.5   21.2        12
## 4 AA            5     17.5        10
## 5 B6          -10.5   12.2         6
```

The winner is United Airlines, with 21. This a function of United having a hub in SFO and, therefore, a higher number of flights there than most. In addition, another one of their hubs in EWR and one of their most common routes is EWR to SFO.

Departure delays by month

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

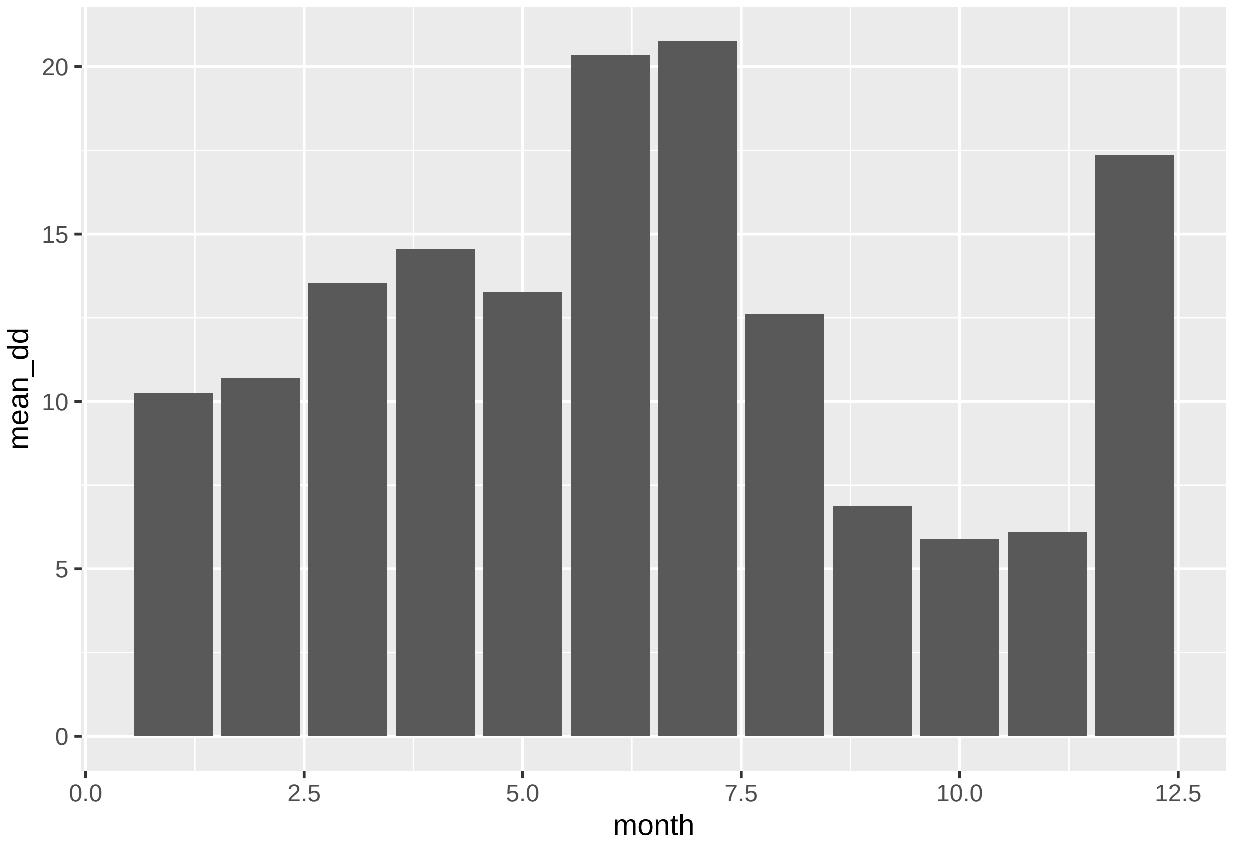
Let's think about how you could answer this question:

- 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

Kevin Note: I added a bar chart that invokes "identity" for stat to allow me to see see the mean data as a regular bar chart.

```
mean_delay_by_month <- nycflights %>%
  group_by(month) %>%
  summarise(mean_dd = mean(dep_delay, na.rm = TRUE)) %>%
  arrange(desc(mean_dd))

ggplot(mean_delay_by_month, aes(x = month, y = mean_dd)) +
  geom_bar(stat = "identity")
```



Kevin Note: the summer months of Junr and July and the holiday month of February being at the top isn't surprising and is more in the "that was predictable" end of the spectrum rather than "huh, that's surprising.

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?

- Mean
  - Pros *This could better represent the chance for an outlier experience since all values would be taken into account
    - It would work best for normally distributed data without an excessive tail. Excessive tails drag means up, which famously happens in housing prices *Cons
    - The mean is really just a medicore metric with limited value by itself and shouldn't usually be the basis for a decision. It can be easily warped by outliers and, if you're traveling, you want something closer to the middle of the dataset

Median * Pros * The median is the middle, which means you have a heavy clutestering of the number of values around it. You would have a higher liklihood, as a percent of number of flights, of having a departure delay near it * Cons * It doesn't adapt to changes in the values of the data until the whole set has moved enough to move the middle

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:

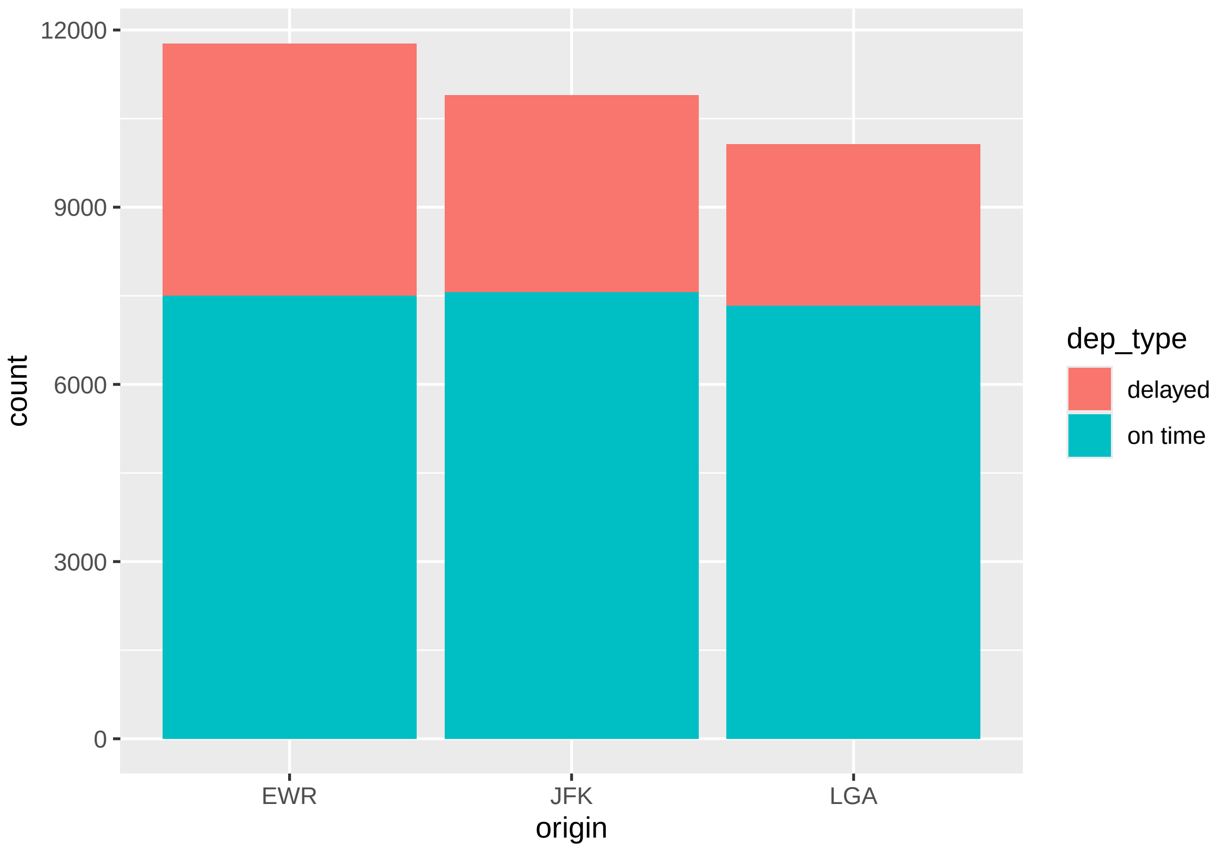
```
nycflights %>%
  group_by(origin) %>%
  summarise(ot_dep_rate = sum(dep_type == "on time") / n()) %>%
  arrange(desc(ot_dep_rate))
```

```
## # 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()
```



LGA is the technically correct answer at 72.8% on time but not the context correct answer. LGA has flight distance restrictions, which means airlines can only have flights that extend as far west as Denver fly out of it. This also means it can't have international flights except to certain spots in Europe. There are a subset of flights that happen at all three and I, as a citizen of NYC, always pick LGA when it's an option for a nonstop flight.

More Practice

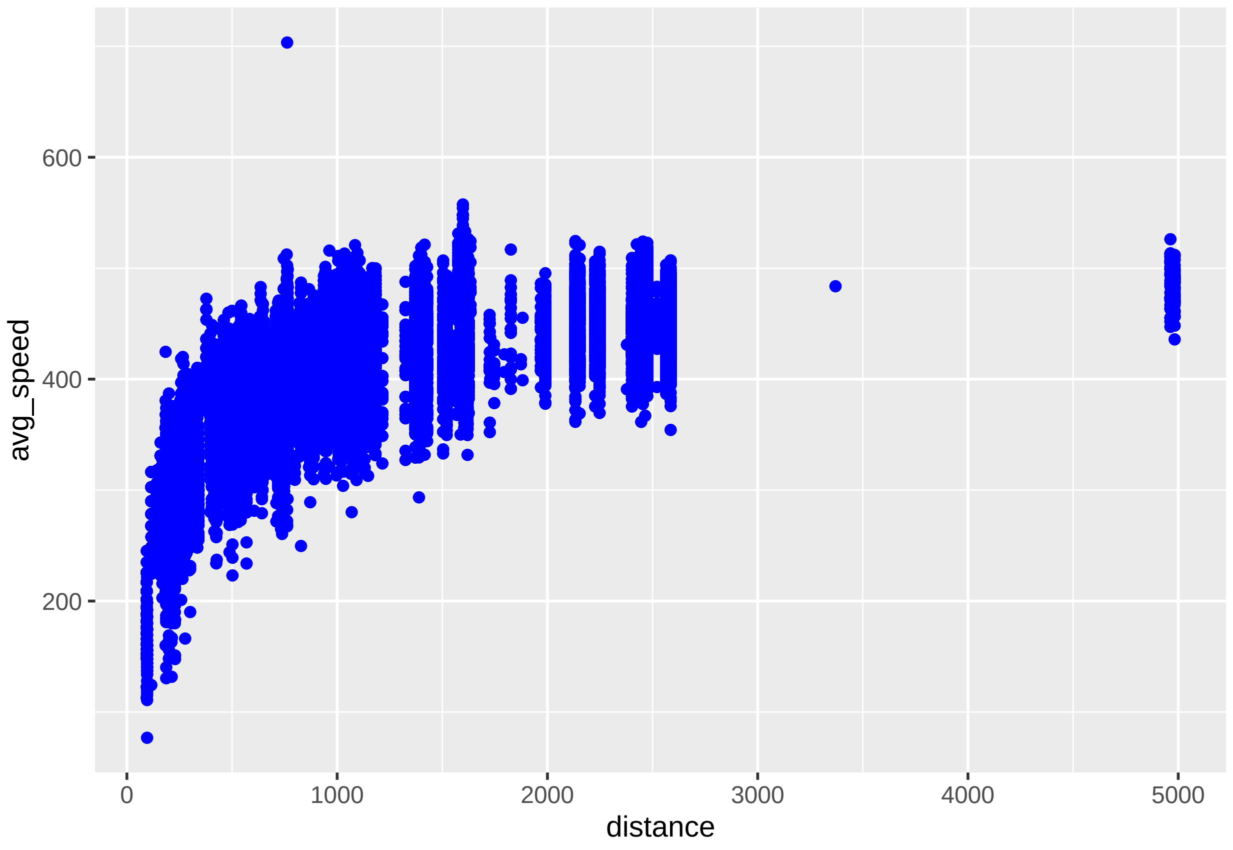
7. Mutate the data frame so that it includes a new variable that contains the average speed, `avg_speed` traveled by the plane for each flight (in mph). **Hint:** Average speed can be calculated as distance divided by number of hours of travel, and note that `air_time` is given in minutes.

Here's how to do the mutation. I added air_time_hours, which just divides air_time hours by 60 to get hours. Then I added avg_speed, which can now be simple division of distance/air_time hours. This provides average distance as a function of flight hours.

```
nycflights <- nycflights %>%
  mutate(air_time_hours = air_time / 60) %>%
  mutate(avg_speed = distance / air_time_hours)
```

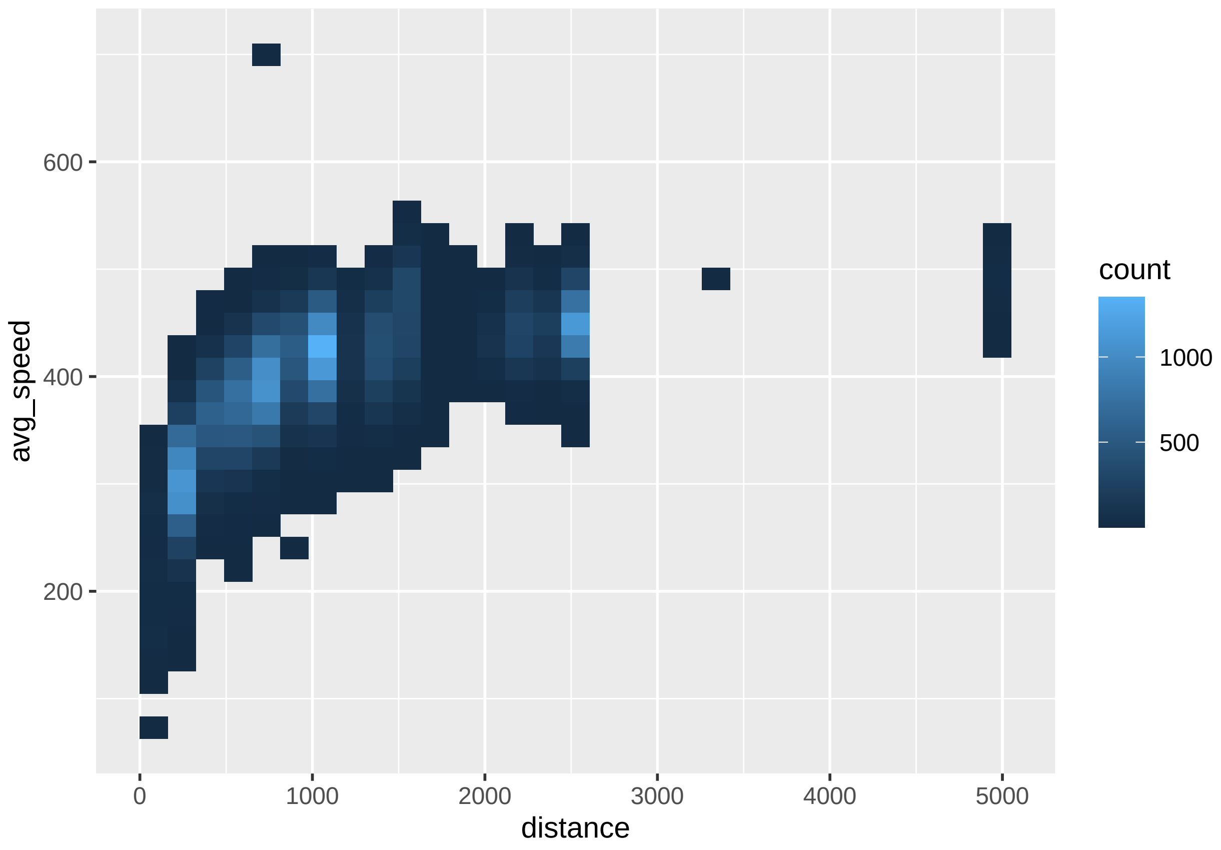
8. Make a scatterplot of `avg_speed` vs. `distance`. Describe the relationship between average speed and distance. **Hint:** Use `geom_point()`.

```
ggplot(nycflights, aes(x = distance, y = avg_speed)) +
  geom_point(color = "blue")
```

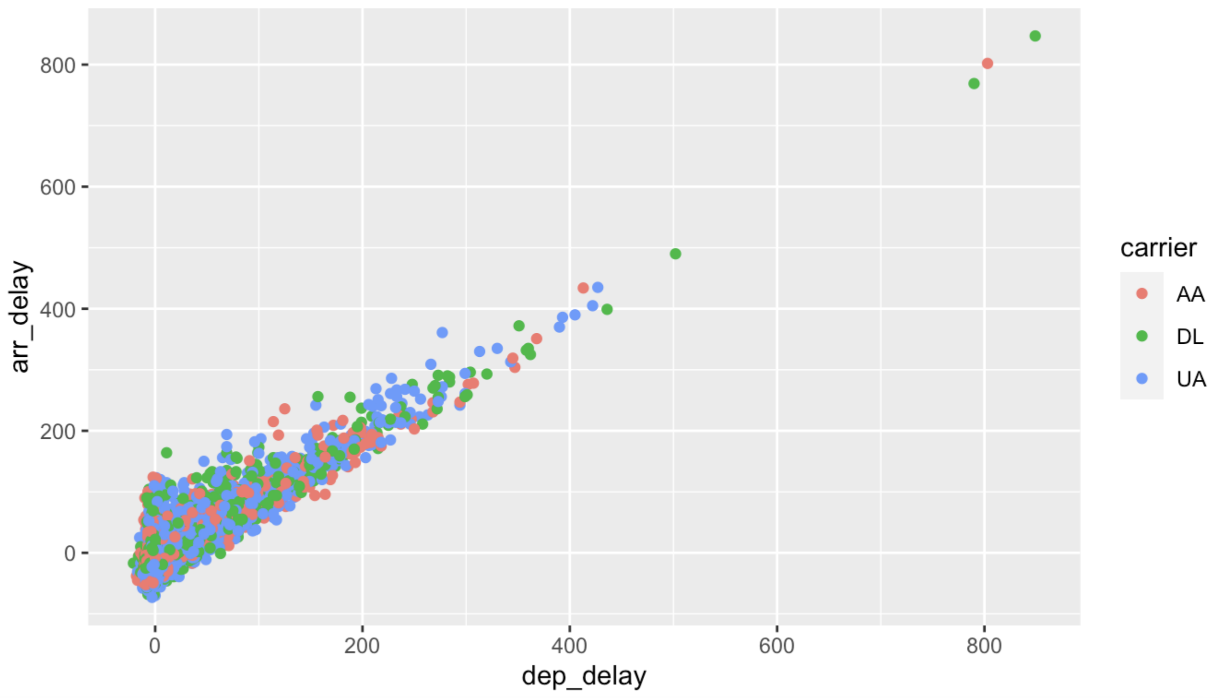


I didn't find that scatter particularly insightful so I tried adding a geom__bid__2d() chart, which is a scatter plot with a heat map built in. Admittedly, it's a bit of an ugly blob that shows what you could guess: the most density of flights is around 500-525 MPH.

```
ggplot(nycflights, aes(x = distance, y = avg_speed)) +
  geom_bin_2d()
```

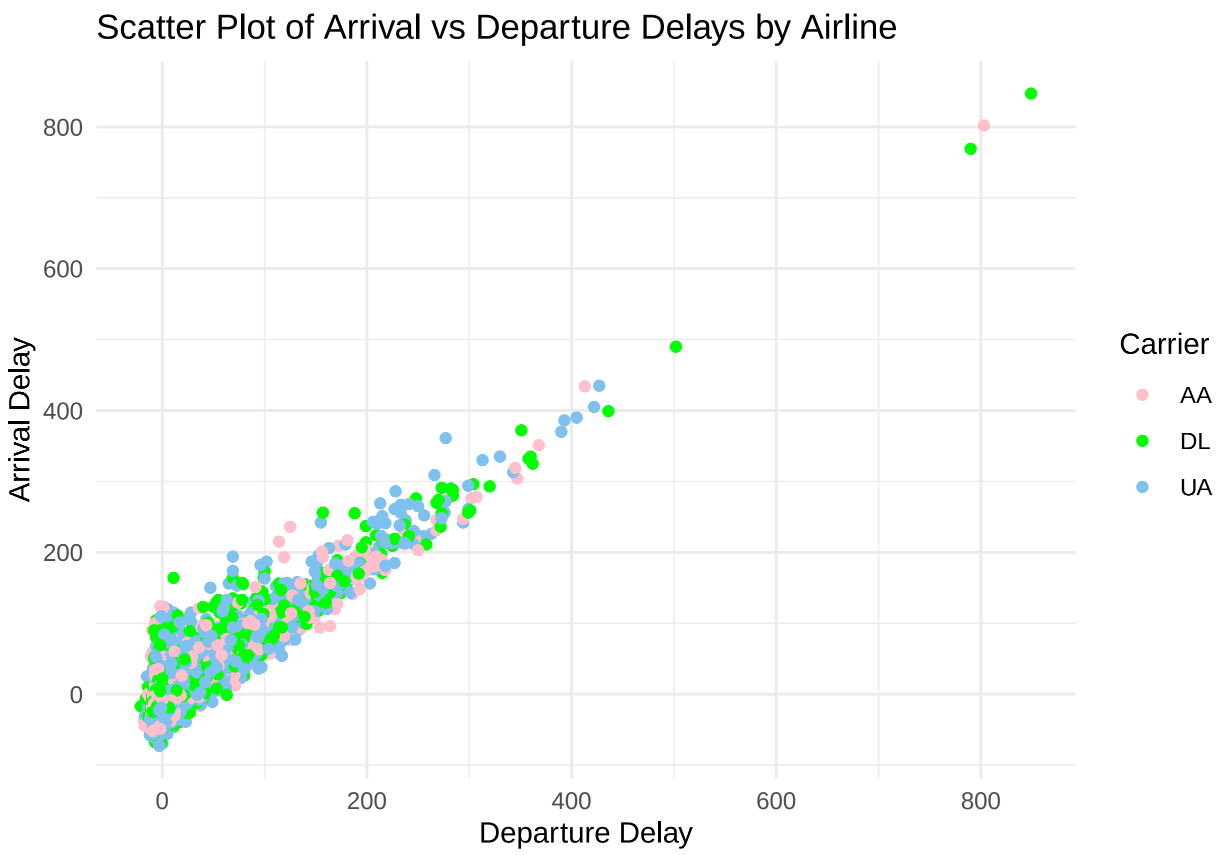


9. Replicate the following plot. **Hint:** The data frame plotted only contains flights from American Airlines, Delta Airlines, and United Airlines, and the points are `colored` by `carrier`. Once you replicate the plot, determine (roughly) what the cutoff point is for departure delays where you can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



```r
scatter_subset_airlines <- nycflights %>%
  filter(carrier %in% c("UA", "AA", "DL"))

ggplot(scatter_subset_airlines, aes(x = dep_delay, y = arr_delay, color = carrier)) +
  geom_point() +
  scale_color_manual(values = c("UA" = "skyblue2", "AA" = "pink", "DL" = "green")) +
  labs(title = "Scatter Plot of Arrival vs Departure Delays by Airline",
       x = "Departure Delay",
       y = "Arrival Delay",
       color = "Carrier") +
  theme_minimal()
```



For the purpose of answering this question, I assumed what really mattered was a flight arriving within five minutes of scheduled arrival. It's sort if like cool story, bro if your flight departs on time and is still an hour late, which a lot of these flights are. Based on the chart, if you're flight departs no more than five minutes late then you can expect your flight to arrive within five minutes of scheduled arrival.

However, using a chart like this to answer this question is suboptimal. Why do "roughly" when you can just produce some numbers to go along with it. Here I make summary stats by airline and then overall, based on a new dataframe that's filtered for arriving withi five minutes.

This shows you the mean, median, minimum, and maximum departure delays you can have by airline and still arrive within five minutes. When combined with the scatter plot, you can understand the numbers on a deeper level and figure out what risk you're willing to take.

"'{r-new-arrival-stats}

five_min_subset <- scatter_subset_airlines %>% filter(arr_delay <= 5)

stats_airline <- five_min_subset %>% group_by(carrier) %>% summarise(mean_dep_delay_a = mean(dep_delay, na.rm = TRUE), median_dep_delay_a = median(dep_delay, na.rm = TRUE), min_dep_delay_a = min(dep_delay, na.rm = TRUE), max_dep_delay_a = max(dep_delay, na.rm = TRUE), n_airline = n())

stats_airline

~~~
As you can see from the overall stats, you really have to have a flight that leaves ahead of schedule to

```{r-overall-delay-stats}
~~~

```
overall_fivemin <- five_min_subset %>%
  summarise(mean_dep_delay_o = mean(dep_delay, na.rm = TRUE),
            median_dep_delay_o = median(dep_delay, na.rm = TRUE),
            min_dep_delay_o = min(dep_delay, na.rm = TRUE),
            max_dep_delay_o = max(dep_delay, na.rm = TRUE),
            n_overall = n())

overall_fivemin
```