Session 4: Data wrangling

Natalie Nelson, PhD, Biological & Agricultural Engineering, NCSU

02/09/2018

1 Data

Data analyzed in this session are from the Department of the Interior (US Geological Survey and Bureau of Reclamation) and include flow (cfs) measurements from three locations along the Colorado River: Lake Powell, Lees Ferry, and the US-Mexico border (Figure 1). The Lake Powell data also include water depth measurements (ft). Observations were collected over different periods for each of the three sites, with the earliest dating back to October 1921.

The Colorado River provides water to one in eight Americans and supports one-seventh of the nation's crops *(Unplugging the Colorado River, NYT)*, making it one of the most important water bodies in the US. The Colorado River's flows are regulated by two major dams: the Hoover Dam (construction completed in 1936) and Glen Canyon Dam (construction completed in 1963). Lake Mead and Lake Powell are the reservoirs of the Hoover and Glen Canyon Dams, respectively. The Colorado River crosses the US-Mexico border and discharges into the Pacific Ocean. Lees Ferry is located in the reach of the Colorado River just downstream of the Glen Canyon Dam.

Figure 1: Major dams and reservoirs of the Lower Colorado River. Map adapted from the New York Times.

---

2 Questions

In today's problem session, we will analyze hydrologic data from the Colorado River system to investigate the following questions (deliverables required for each question are listed as bullets):

1. How have natural flows along the Colorado River changed over time? For the purpose of this assignment, consider the Lake Powell inflow, Lees Ferry flows, and border flows to be natural.

- Multi-panel plot of flow vs. date for Lake Powell, Lees Ferry, Border (each panel should correspond to a different location); plot should be 1 column; vertical lines should be added to mark the completion of the Hoover and Glen Canyon Dams; panels should be sorted geographically.

2. How have statistical properties of flows measured at Lees Ferry been impacted by the construction of the Glen Canyon Dam?

- Quantile plots of flow pre-/post-construction of the Glen Canyon Dam; one single-panel plot.
- Boxplots organized by month for pre-/post-construction; one multi-panel plot (one panel for pre-, one for post-).
- Numerical summary including mean, standard deviation, median, IQR, median absolute deviation of flow for pre- and post-construction conditions. Summary statistics should be organized in a table.

3. How do statistical properties differ between the inflows and outflows of Lake Powell?

- Scatterplot of outflow vs. inflow, points colored based on year; colors should not be the default dark-to-light blue gradient.
- Quantile plots of inflows and outflows; one single-panel plot; vertical line at the median.
- Plot of inflow vs. date and outflow vs. date; one multi-panel plot.

For each question, provide the practical interpretation of the results. What are the real-world implications of the patterns you've uncovered in the data?

3 Analysis

Setting up your R script

- New packages (`install.packages("")`): `forcats`
- Packages to load: `ggplot2`, `dplyr`, `readr`, `tidyr`, and `forcats`

```r
# Clear workspace
rm(list=ls(all=TRUE))

# Load packages
library(ggplot2)
library(dplyr)
library(tidyr)
library(readr)
library(forcats)

# Set working directory
#setwd("your-path-name") # Note: this line should not be commented out in your code!
```

Read in your data

The data are saved in three .csv files:

- Lake Powell: "1-LakePowell-up-GlenCanyon.csv"
- Lees Ferry: "2-LeesFerry-down-GlenCanyon.csv"
- US-MX border: "3-Border.csv"

```r
p <- read_csv("1-LakePowell-up-GlenCanyon.csv")
```

```
## Parsed with column specification:
## cols(
##   date = col_date(format = ""),
##   year = col_integer(),
##   month = col_integer(),
##   depth = col_double(),
##   inflow = col_double(),
##   outflow = col_double()
## )
```

```r
f <- read_csv("2-LeesFerry-down-GlenCanyon.csv")
```

```
## Parsed with column specification:
## cols(
##   date = col_date(format = ""),
##   year = col_integer(),
##   month = col_integer(),
##   flow = col_integer()
## )
```

```r
b <- read_csv("3-Border.csv")
```

```
## Parsed with column specification:
## cols(
##   date = col_date(format = ""),
##   year = col_integer(),
##   month = col_integer(),
##   flow = col_double()
## )
```

Check the packaging and the top and bottom of your data

As always, our next step is to inspect our data.

```r
str(p)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    18534 obs. of  6 variables:
##  $ date   : Date, format:
```

```
## Warning in format.POSIXlt(as.POSIXlt(x), ...): unknown timezone 'zone/tz/
## 2018c.1.0/zoneinfo/America/New_York'
```

```
## "1963-12-28" "1963-12-29" ...
##  $ year   : int  1963 1963 1963 1963 1964 1964 1964 1964 1964 1964 ...
##  $ month  : int  12 12 12 12 1 1 1 1 1 1 ...
##  $ depth  : num  3409 3409 3409 3410 3410 ...
##  $ inflow : num  4739 4416 4587 4546 4532 ...
##  $ outflow: num  980 980 980 980 980 980 980 980 980 980 ...
##  - attr(*, "spec")=List of 2
##   ..$ cols   :List of 6
##   .. ..$ date   :List of 1
##   .. .. ..$ format: chr ""
##   .. .. ..- attr(*, "class")= chr  "collector_date" "collector"
##   .. ..$ year   : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
```

```
##   .. ..$ month  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ depth  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ inflow : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ outflow: list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   ..$ default: list()
##   .. ..- attr(*, "class")= chr  "collector_guess" "collector"
##   ..- attr(*, "class")= chr "col_spec"
```

Based on seeing "Classes 'tbl_df', 'tbl' and 'data.frame' " in the upper left corner of our output, we know that the data were loaded as a *tibble*. With tibbles, you can also get a lot of valuable information simply by looking at the top of your data. Although you can use `head()` to view the top of a tibble, you can also just enter your tibble's object name to see the first 10 rows printed.

```
p
```

```
## # A tibble: 18,534 x 6
##          date  year month  depth  inflow outflow
##        <date> <int> <int>  <dbl>   <dbl>   <dbl>
##  1 1963-12-28  1963    12 3409.0 4738.93     980
##  2 1963-12-29  1963    12 3409.2 4415.93     980
##  3 1963-12-30  1963    12 3409.4 4586.51     980
##  4 1963-12-31  1963    12 3409.6 4545.84     980
##  5 1964-01-01  1964     1 3409.8 4532.17     980
##  6 1964-01-02  1964     1 3410.0 4415.76     980
##  7 1964-01-03  1964     1 3410.2 4501.51     980
##  8 1964-01-04  1964     1 3410.4 4540.38     980
##  9 1964-01-05  1964     1 3410.6 4453.21     980
## 10 1964-01-06  1964     1 3410.8 4551.07     980
## # ... with 18,524 more rows
```

```
tail(p)
```

```
## # A tibble: 6 x 6
##         date  year month   depth  inflow  outflow
##       <date> <int> <int>   <dbl>   <dbl>    <dbl>
## 1 2014-09-19  2014     9 3605.73 9075.13 10407.05
## 2 2014-09-20  2014     9 3605.70 8564.40  9628.62
## 3 2014-09-21  2014     9 3605.66 8079.66  9144.08
## 4 2014-09-22  2014     9 3605.58 6915.73 10389.46
## 5 2014-09-23  2014     9 3605.51 7836.96 10773.67
## 6 2014-09-24  2014     9 3605.46 8798.77 10664.73
```

From this output, we can see that we have 6 variables in the Lake Powell (`p`) tibble: one of type "date", two of type "integer", and three of type "double". The only new data type in this case is "date".

```
f
```

```
## # A tibble: 33,962 x 4
##          date  year month  flow
##        <date> <int> <int> <int>
##  1 1921-10-01  1921    10  7120
##  2 1921-10-02  1921    10 11800
##  3 1921-10-03  1921    10  7830
##  4 1921-10-04  1921    10  7470
```

```
##  5 1921-10-05  1921    10  6780
##  6 1921-10-06  1921    10  6950
##  7 1921-10-07  1921    10  6950
##  8 1921-10-08  1921    10  7120
##  9 1921-10-09  1921    10  6780
## 10 1921-10-10  1921    10  6780
## # ... with 33,952 more rows
```

```r
tail(f)
```

```
## # A tibble: 6 x 4
##         date  year month  flow
##       <date> <int> <int> <int>
## 1 2014-09-19  2014     9 10600
## 2 2014-09-20  2014     9  9950
## 3 2014-09-21  2014     9  9430
## 4 2014-09-22  2014     9 10600
## 5 2014-09-23  2014     9 11000
## 6 2014-09-24  2014     9 11000
```

```r
b
```

```
## # A tibble: 22,188 x 4
##          date  year month     flow
##        <date> <int> <int>    <dbl>
##  1 1950-01-01  1950     1 13278.31
##  2 1950-01-02  1950     1 13454.89
##  3 1950-01-03  1950     1 13207.69
##  4 1950-01-04  1950     1 11936.36
##  5 1950-01-05  1950     1 13878.66
##  6 1950-01-06  1950     1 14196.50
##  7 1950-01-07  1950     1 14408.38
##  8 1950-01-08  1950     1 14902.79
##  9 1950-01-09  1950     1 14479.01
## 10 1950-01-10  1950     1 13949.29
## # ... with 22,178 more rows
```

```r
tail(b)
```

```
## # A tibble: 6 x 4
##         date  year month  flow
##       <date> <int> <int> <dbl>
## 1 2010-09-25  2010     9     0
## 2 2010-09-26  2010     9     0
## 3 2010-09-27  2010     9     0
## 4 2010-09-28  2010     9     0
## 5 2010-09-29  2010     9     0
## 6 2010-09-30  2010     9     0
```

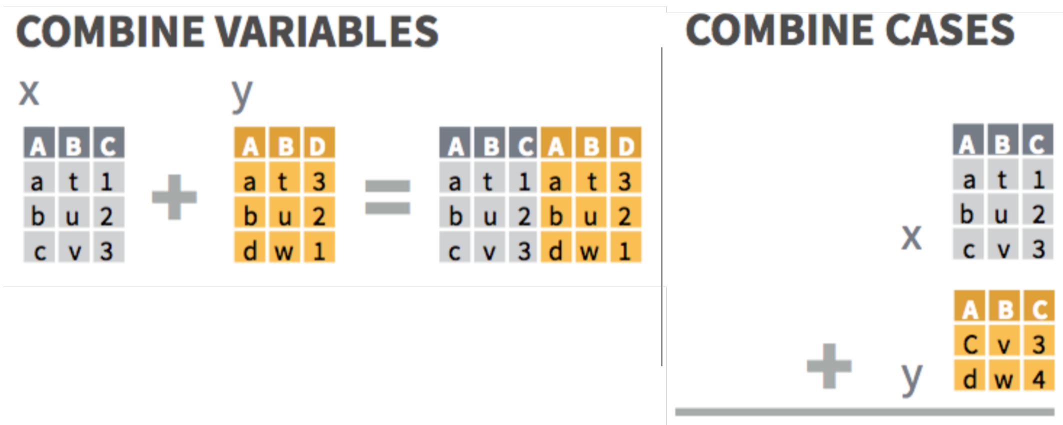
Here, we see that the Lees Ferry (`f`) and US-MX border (`b`) tibbles are almost identical in structure, except flow is of type "integer" in f, and "double" in b. Since both integer and double types are *numeric*, there is no issue.

Question 1: How have natural flows along the Colorado River changed over time?

Multi-panel plot of flow vs. date for Lake Powell, Lees Ferry, Border (each panel should correspond to a different location); plot should be 1 column; vertical lines should be added to mark the completion of the

Hoover and Glen Canyon Dams; panels should be sorted geographically.

In order to construct a multi-panel plot of flow vs. date for Lake Powell, Lees Ferry, and the border, we'll need to merge our datasets. Per our **Data Transformation** cheat sheet, we know that we can combine datasets either by row ("combining cases") or column ("combining variables").



For this application, we want to combine cases; we aren't adding any new types of variables to the data, we're simply combining observations of the same variable types. To do this we'll use `bind_rows()`, which stacks tibbles/dataframes into a single tibble/dataframe. Additionally, with `bind_rows()`, you can specify that an "ID" column of your naming be added to the output table. We will opt to create an ID column in which we store the location names associated with each tibble that we're merging, and we'll call this ID column "site".

```
flows <- bind_rows("Lake Powell" = p, "Lees Ferry" = f, "Border" = b, .id = "site")
flows
```

```
## # A tibble: 74,684 x 8
##           site       date  year month  depth  inflow outflow  flow
##          <chr>     <date> <int> <int>  <dbl>   <dbl>   <dbl> <dbl>
##  1 Lake Powell 1963-12-28  1963    12 3409.0 4738.93     980    NA
##  2 Lake Powell 1963-12-29  1963    12 3409.2 4415.93     980    NA
##  3 Lake Powell 1963-12-30  1963    12 3409.4 4586.51     980    NA
##  4 Lake Powell 1963-12-31  1963    12 3409.6 4545.84     980    NA
##  5 Lake Powell 1964-01-01  1964     1 3409.8 4532.17     980    NA
##  6 Lake Powell 1964-01-02  1964     1 3410.0 4415.76     980    NA
##  7 Lake Powell 1964-01-03  1964     1 3410.2 4501.51     980    NA
##  8 Lake Powell 1964-01-04  1964     1 3410.4 4540.38     980    NA
##  9 Lake Powell 1964-01-05  1964     1 3410.6 4453.21     980    NA
## 10 Lake Powell 1964-01-06  1964     1 3410.8 4551.07     980    NA
## # ... with 74,674 more rows
```

```
tail(flows)
```

```
## # A tibble: 6 x 8
##     site       date  year month depth inflow outflow  flow
##    <chr>     <date> <int> <int> <dbl>  <dbl>   <dbl> <dbl>
## 1 Border 2010-09-25  2010     9    NA     NA      NA     0
## 2 Border 2010-09-26  2010     9    NA     NA      NA     0
## 3 Border 2010-09-27  2010     9    NA     NA      NA     0
## 4 Border 2010-09-28  2010     9    NA     NA      NA     0
## 5 Border 2010-09-29  2010     9    NA     NA      NA     0
## 6 Border 2010-09-30  2010     9    NA     NA      NA     0
```

Looking at the top of our data, we can see that inflow, outflow, and flow each have their own columns. We

want all of our flows to be in the same column - in particular, we want our *inflow* and *flow* data to be merged in a single column since these are the data that we'll be plotting in our multi-panel plot. The `bind_rows()` function merges data based on the column names. In this case, `b` and `f` would have merged correctly since these tibbles had the exact same column names. However, `p` has different column names.
To get around this issue, we can create a subset of `p` and rename *inflow* to *flow* so that it correctly merges with `b` and `f`. To do this, we'll try using pipes! We will first pipe (`%>%`) our data into `select()`. We use `select()` when we want to subset our tibble based on *columns*. See your Data Transformation cheat sheet (front page under "Manipulate Variables"). We will use the select function to choose the columns that we'd like to include in our subset. Next, we'll pipe the output of select into `rename()`, which is a function used to rename tibble/dataframe columns.

```
p.flow <- p %>% select(date, year, month, inflow) %>% rename(flow = inflow)
p.flow
```

```
## # A tibble: 18,534 x 4
##          date  year month    flow
##        <date> <int> <int>   <dbl>
##  1 1963-12-28  1963    12 4738.93
##  2 1963-12-29  1963    12 4415.93
##  3 1963-12-30  1963    12 4586.51
##  4 1963-12-31  1963    12 4545.84
##  5 1964-01-01  1964     1 4532.17
##  6 1964-01-02  1964     1 4415.76
##  7 1964-01-03  1964     1 4501.51
##  8 1964-01-04  1964     1 4540.38
##  9 1964-01-05  1964     1 4453.21
## 10 1964-01-06  1964     1 4551.07
## # ... with 18,524 more rows
```

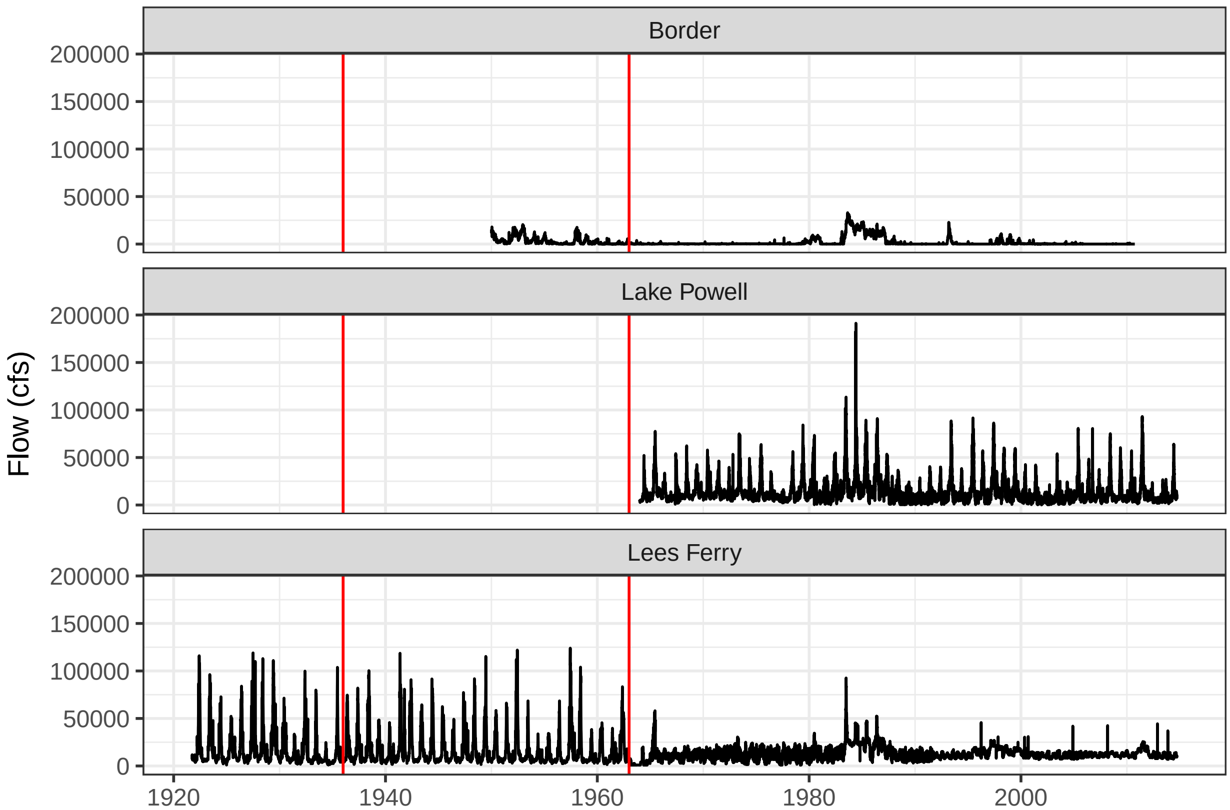
Did the trick! Now, we can go ahead and try combining our `p`, `f`, and `b` tibbles/dataframes again.

```
flows <- bind_rows("Lake Powell"=p.flow, "Lees Ferry"=f, "Border"=b, .id="site")
flows
```

```
## # A tibble: 74,684 x 5
##           site       date  year month    flow
##          <chr>     <date> <int> <int>   <dbl>
##  1 Lake Powell 1963-12-28  1963    12 4738.93
##  2 Lake Powell 1963-12-29  1963    12 4415.93
##  3 Lake Powell 1963-12-30  1963    12 4586.51
##  4 Lake Powell 1963-12-31  1963    12 4545.84
##  5 Lake Powell 1964-01-01  1964     1 4532.17
##  6 Lake Powell 1964-01-02  1964     1 4415.76
##  7 Lake Powell 1964-01-03  1964     1 4501.51
##  8 Lake Powell 1964-01-04  1964     1 4540.38
##  9 Lake Powell 1964-01-05  1964     1 4453.21
## 10 Lake Powell 1964-01-06  1964     1 4551.07
## # ... with 74,674 more rows
```

We can now go ahead and visualize our flows vs. dates. Per the specified deliverable, we'll be creating a 1-column multi-panel plot that includes vertical lines to mark the dates when the Hoover and Glen Canyon Dams were constructed. Each panel should include one of the three sites. We will divide the data into panels using `facet_wrap()` and specify that the panels should be in one column with the `ncol = 1` argument. `geom_vline()` is used to add a verticle line. Annoyingly, `geom_vline()` requires that data be of type `numeric`. Therefore, we have to double-specify that the data are dates, and then that the data are numeric. We will learn more about working with dates next week, so don't worry about this for now.

```r
ggplot(flows, aes(x = date, y = flow))+
  geom_line()+
  geom_vline(xintercept = as.numeric(as.Date("1936-01-01")), color = "red")+ # Hoover
  geom_vline(xintercept = as.numeric(as.Date("1963-01-01")), color = "red")+ # Glen Canyon Dam
  theme_bw()+
  ylab("Flow (cfs)")+
  xlab("")+
  facet_wrap(~site, ncol=1)
```

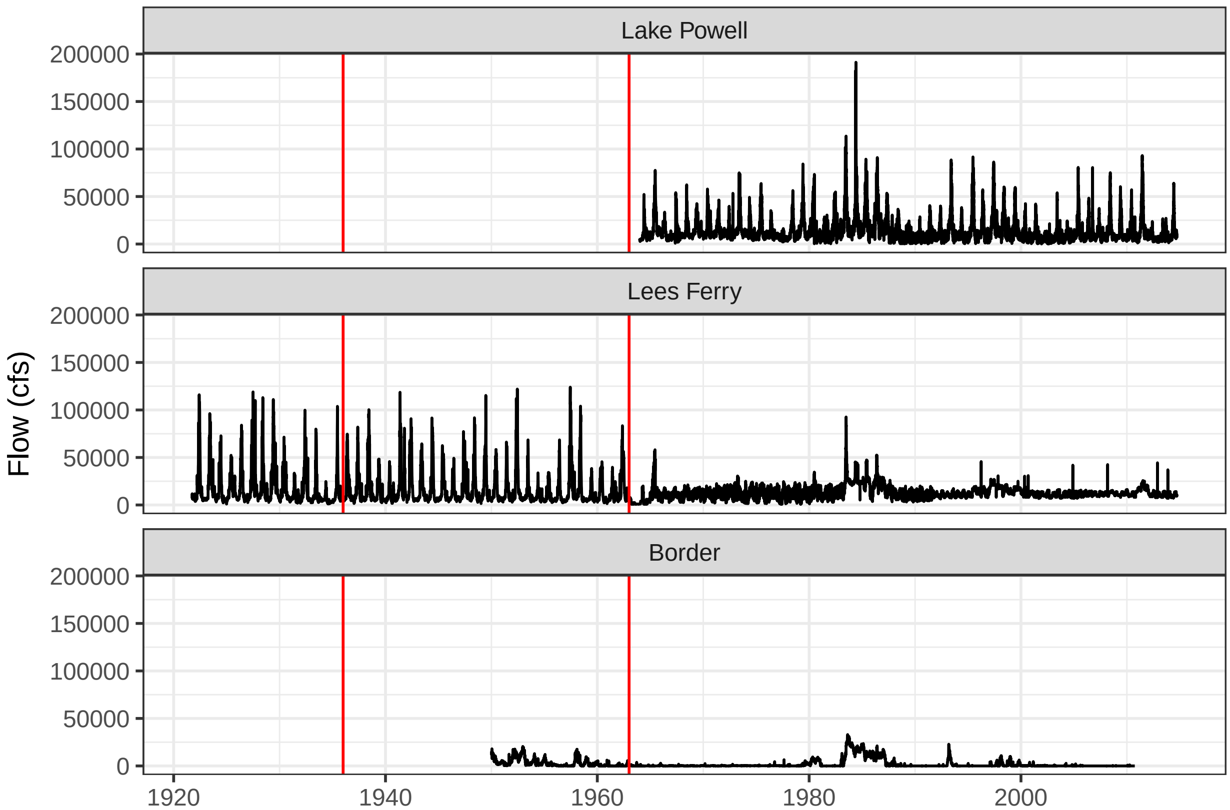


The deliverable stipulates that the data need to be sorted geographically. This means having the panels in the same order as they occur in the Colorado River. The panels should be ordered as Lake Powell, Lees Ferry, border. Presently, they're ordered as border, Lake Powell, Lees Ferry. This is because character variables are always ordered alphabetically by default. To change the order, we have to specify that the order should be changed. This is done with `fct_relevel()`, which is an abbreviated way of saying *factor re-level*. With this function, we simply list the levels in the order we desire.

```r
flows$site <- fct_relevel(flows$site, "Lake Powell", "Lees Ferry", "Border")
```

We can now try re-plotting our data.

```r
ggplot(flows, aes(x = date, y = flow))+
  geom_line()+
  geom_vline(xintercept = as.numeric(as.Date("1936-01-01")), color = "red")+ # Hoover
  geom_vline(xintercept = as.numeric(as.Date("1963-01-01")), color = "red")+ # Glen Canyon Dam
  theme_bw()+
  ylab("Flow (cfs)")+
  xlab("")+
  facet_wrap(~site, ncol=1)
```



Question 2: How have statistical properties of flows measured at Lees Ferry been impacted by the construction of the Glen Canyon Dam?

(1) Quantile plots of flow pre-/post-construction of the Glen Canyon Dam, one single-panel plot; (2) Boxplots organized by month for pre-/post-construction; one multi-panel plot (one panel for pre-, one for post-); (3) Table of summary statistics including mean, standard deviation, median, IQR, median absolute deviation of flow for pre- and post-construction conditions.

We will first create quantile plots of our Lees Ferry flow observations pre- and post-construction. To do this, we have to calculate the quantiles of the pre-construction data (pre-1963), and post-construction data (1963-). The first step in doing this will be to create two subsets: `year < 1963` and `year > 1962`. Within each of these subsets, we will then create a new column in the tibble that includes our quantile values. To create a new column in a tibble, we can use the `mutate()` function. See the front page of your Data Transformation cheat sheet under "Manipulate Variables" > "Make New Variables".

On the back of our Data Transformation cheat sheet, we have a list of Vector Functions that can be used with `mutate()`. One noteworthy function is `cume_dist()`, which calculates the proportion of all values <= to a given observation. This is the equivalent of calculating the corresponding quantiles for each observation in the dataset, so this is the function we will use.

We can pipe the subset output into the mutate function in order to simplify the code:

```
pre <- f %>% subset(year < 1963) %>% mutate(q = cume_dist(flow))
pre
```

```
## # A tibble: 15,067 x 5
##          date  year month  flow         q
##        <date> <int> <int> <int>     <dbl>
##  1 1921-10-01  1921    10  7120 0.4292825
##  2 1921-10-02  1921    10 11800 0.6520210
##  3 1921-10-03  1921    10  7830 0.4914714
```

```
##   4 1921-10-04  1921    10  7470 0.4620694
##   5 1921-10-05  1921    10  6780 0.3935754
##   6 1921-10-06  1921    10  6950 0.4115617
##   7 1921-10-07  1921    10  6950 0.4115617
##   8 1921-10-08  1921    10  7120 0.4292825
##   9 1921-10-09  1921    10  6780 0.3935754
##  10 1921-10-10  1921    10  6780 0.3935754
## # ... with 15,057 more rows
```

```r
post <- f %>% subset(year > 1962) %>% mutate(q = cume_dist(flow))
post
```

```
## # A tibble: 18,895 x 5
##          date  year month  flow          q
##        <date> <int> <int> <int>      <dbl>
##   1 1963-01-01  1963     1  2250 0.02556232
##   2 1963-01-02  1963     1  2400 0.02625033
##   3 1963-01-03  1963     1  2600 0.02942577
##   4 1963-01-04  1963     1  2990 0.03217782
##   5 1963-01-05  1963     1  3180 0.03344800
##   6 1963-01-06  1963     1  3500 0.03540619
##   7 1963-01-07  1963     1  3940 0.03879333
##   8 1963-01-08  1963     1  4280 0.04530299
##   9 1963-01-09  1963     1  4520 0.04704948
##  10 1963-01-10  1963     1  4700 0.04916645
## # ... with 18,885 more rows
```

Now, let's merge these two subsets with `bind_rows()` to recreate a new tibble. We'll call this new tibble `f.pp` for "f pre post".

```r
f.pp <- bind_rows("pre" = pre, "post" = post, .id = "id")
f.pp
```

```
## # A tibble: 33,962 x 6
##        id       date  year month  flow         q
##     <chr>     <date> <int> <int> <int>     <dbl>
##   1   pre 1921-10-01  1921    10  7120 0.4292825
##   2   pre 1921-10-02  1921    10 11800 0.6520210
##   3   pre 1921-10-03  1921    10  7830 0.4914714
##   4   pre 1921-10-04  1921    10  7470 0.4620694
##   5   pre 1921-10-05  1921    10  6780 0.3935754
##   6   pre 1921-10-06  1921    10  6950 0.4115617
##   7   pre 1921-10-07  1921    10  6950 0.4115617
##   8   pre 1921-10-08  1921    10  7120 0.4292825
##   9   pre 1921-10-09  1921    10  6780 0.3935754
##  10   pre 1921-10-10  1921    10  6780 0.3935754
## # ... with 33,952 more rows
```

```r
tail(f.pp)
```

```
## # A tibble: 6 x 6
##       id       date  year month  flow         q
##    <chr>     <date> <int> <int> <int>     <dbl>
## 1  post 2014-09-19  2014     9 10600 0.3779307
## 2  post 2014-09-20  2014     9  9950 0.2870601
## 3  post 2014-09-21  2014     9  9430 0.2438740
## 4  post 2014-09-22  2014     9 10600 0.3779307
```

```
## 5  post 2014-09-23  2014     9 11000 0.4130193
## 6  post 2014-09-24  2014     9 11000 0.4130193
```

```r
# We can also check to see that the number of rows matches properly
nrow(f.pp)
```

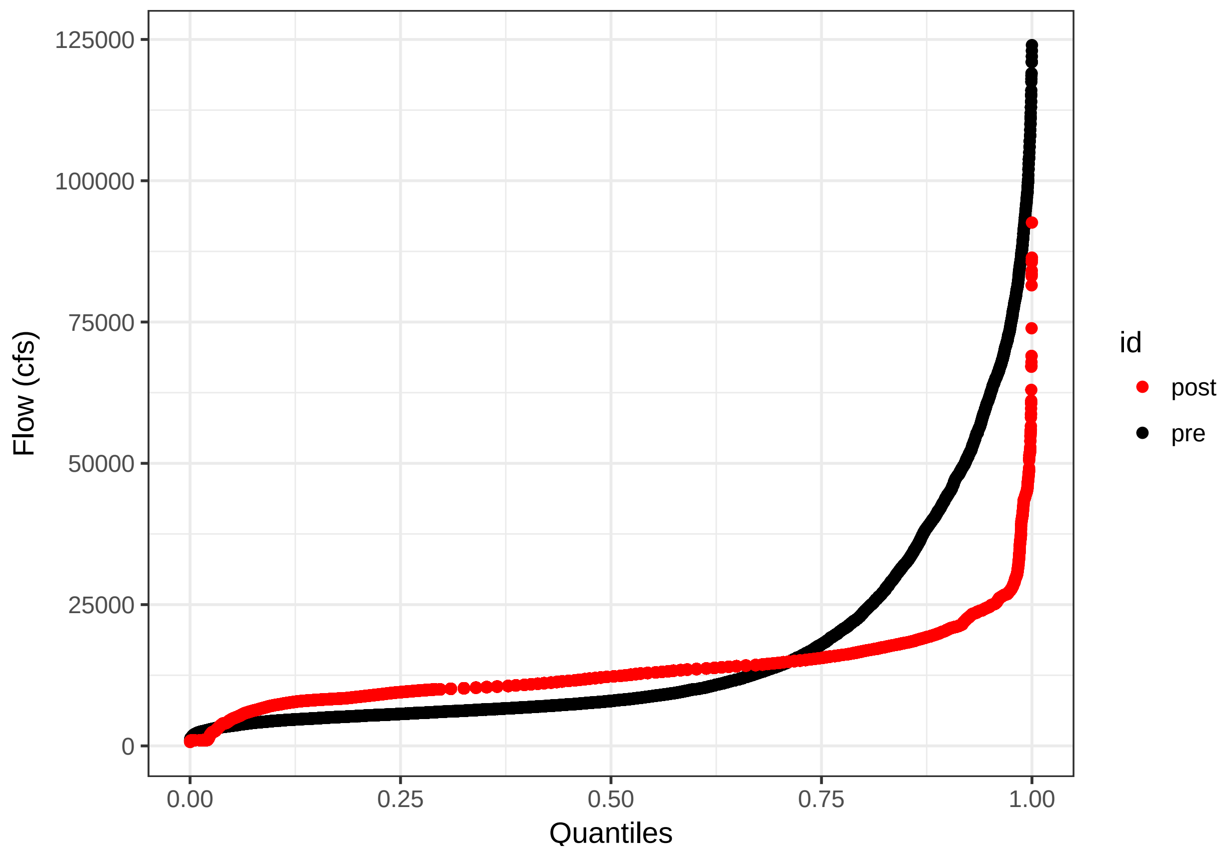
```
## [1] 33962
```

```r
nrow(pre)+nrow(post)
```

```
## [1] 33962
```

We can now create our quantile plots by plotting flow vs. q. Since we only want a single-panel plot, we will not use facet_wrap. Instead, we have to specify that we have two separate groups (pre- and post-) in the *id* column.

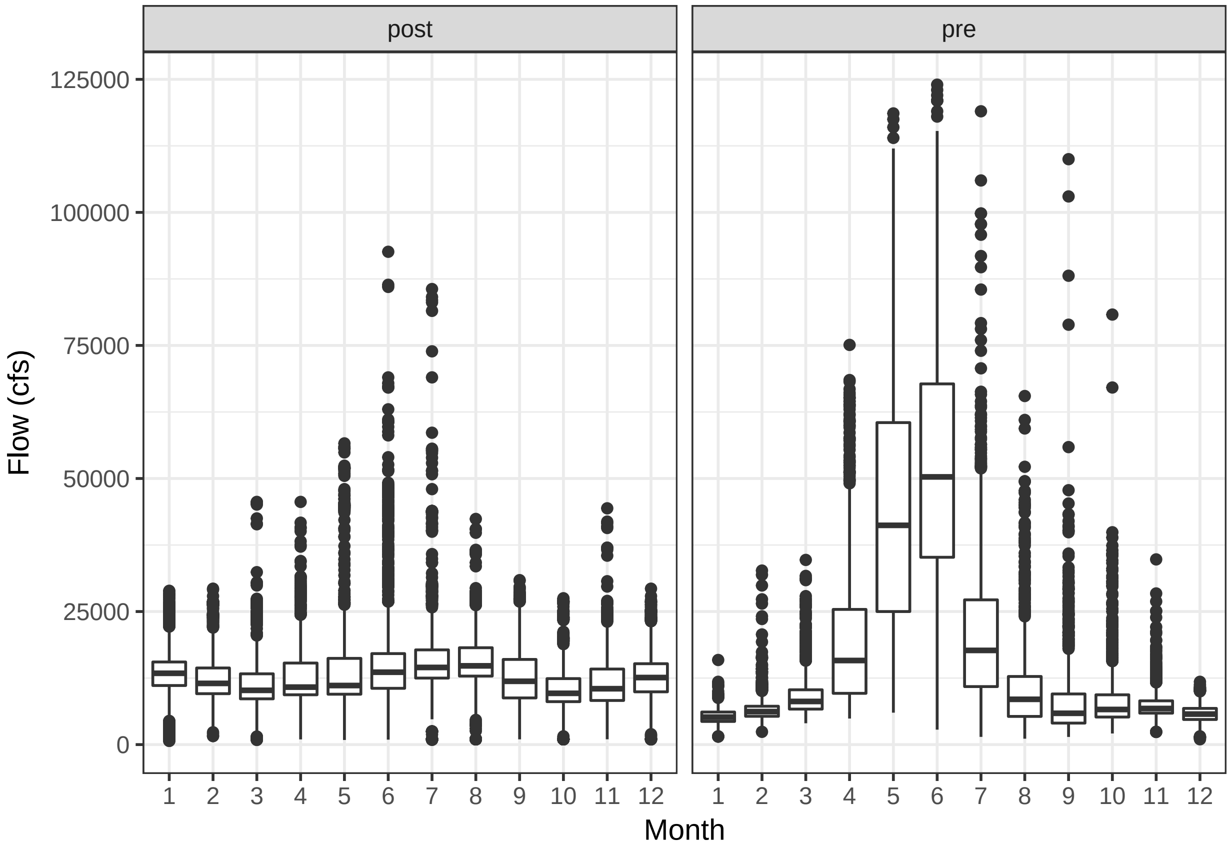
```r
ggplot(f.pp, aes(x = q, y = flow, group = id))+
  geom_point(aes(color = id))+
  scale_color_manual(values = c("red", "black"))+ # Note that this is how you change the colors!
  theme_bw()+
  ylab("Flow (cfs)")+
  xlab("Quantiles")
```



Next, let's create monthly boxplots for pre- and post-construction conditions, with pre- and post- in separate panels (since a multi-panel plot was requested). Note that we have to include `factor(month)` as the argument for x since month is of type "integer", but boxplots require discrete (= character or factor) x variables.

```r
ggplot(f.pp, aes(x = factor(month), y = flow))+
  geom_boxplot()+
  theme_bw()+
```

```
  ylab("Flow (cfs)")+
  xlab("Month")+
  facet_wrap(~id)
```



The last deliverable to produce for Question 2 is a table of summary statistics including mean, standard deviation, median, IQR, median absolute deviation of flow for pre- and post-construction conditions. From our Data Transformation cheat sheet, we can see that data can be summarized with the `summarise()` function (on the front page of the Cheat Sheet under "Summarise Cases"). On the back of the Cheat Sheet under "Summary Functions", we have a list of functions that can be used in `summarise()`. We're specifically interested in those listed under "Location" and "Spread" (remember "measures of location and spread" from previous lectures?).

```
f.summary <- f.pp %>% summarise(mean = mean(flow), sd = sd(flow), median = median(flow), iqr = IQR(flow)
f.summary
```

```
## # A tibble: 1 x 5
##       mean       sd median   iqr     mad
##      <dbl>    <dbl>  <dbl> <dbl>   <dbl>
## 1 14737.51 13887.81  10900  8790 6226.92
```

This didn't work because we didn't specify that the summary should be prepared for pre- and post-construction conditions. This can be done using the `group_by()` function. See the front page of your Data Transformation cheat sheet under "Group Cases" for more details.

```
f.summary <- f.pp %>% group_by(id) %>% summarise(mean = mean(flow), sd = sd(flow), median = median(flow)
f.summary
```

```
## # A tibble: 2 x 6
##      id     mean        sd median   iqr      mad
```

```
##    <chr>    <dbl>     <dbl>  <int> <dbl>    <dbl>
## 1  post 13287.84  6830.761  12300  6080 4447.800
## 2   pre 16555.50 19243.303   7950 12340 4729.494
```

If we want to save these outputs so we can easily open them in a text editor or Excel, we can write `f.summary` to a .csv file.

```
write_csv(f.summary,"summary-lees-ferry-pre-post.csv")
```

Question 3: How do statistical properties differ between the inflows and outflows of Lake Powell?

Complete this question on your own. **Hint:** after creating your scatterplot, you will need to `gather()` your inflow and outflow columns to move on to creating the quantile plots. We used `gather()` in the last problem session, and it's also outlined in your Data Import cheat sheet under "Reshape Data" (on back).