Introduction to data modeling

ENS-215

26-Feb-2019

First let's load in the packages we'll use today.

```
library(tidyverse)
library(lubridate)
library(moderndive)
```

Time-series analysis (wrap-up)

Last class we saw some approaches to visualizing and interpreting time-series data. I'd like to introduce you all to one additional method in time-series visualization and analysis.

Cycle plots

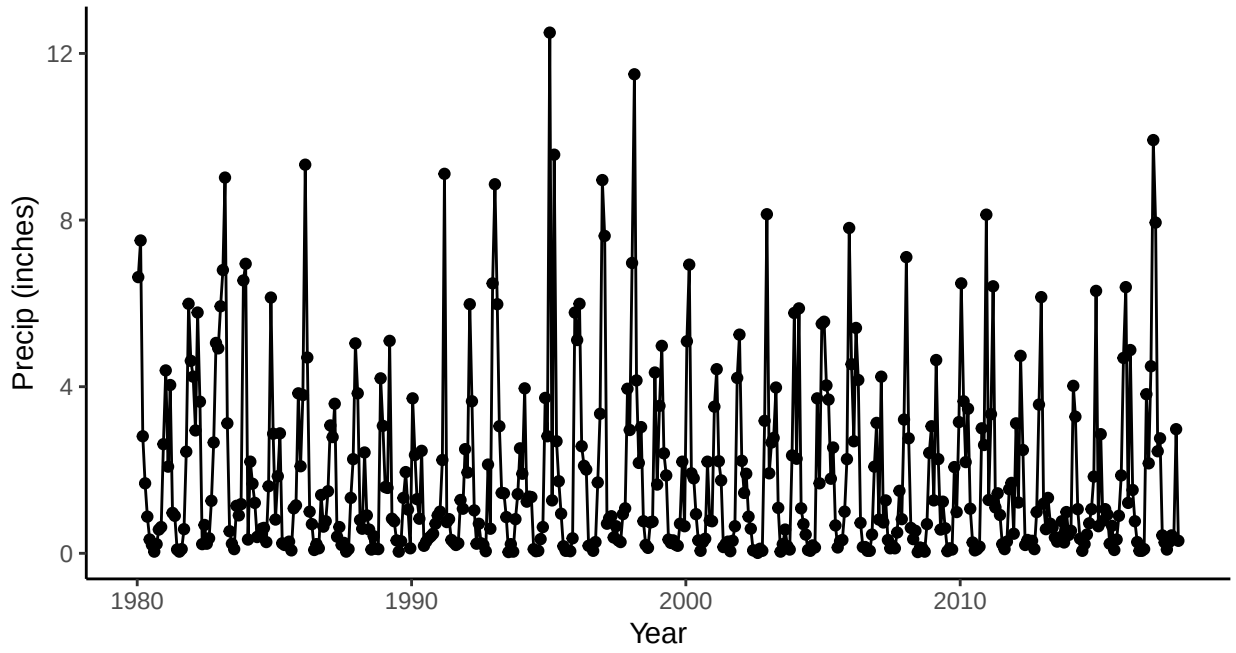
Another really useful way to examine data that may have both a seasonal and long-term trend is through the use of **cycle plots**. The best way to understand cycle plots is through an example. Let's consider precipitation in California. There might be a seasonal pattern (there is as you will see shortly) and there could also be long-term trends in the data. For instance it might be the case that some months have been getting wetter (or drier) over time. Seeing both the seasonal pattern and the trends (in particular trends that apply only to some parts of the year) would be very difficult to see in a traditional time-series plot.

Let's look at a standard time-series plot to highlight the point.

First load in the NOAA precipitation data that we've used in the past.

```
precip_data <- read_csv("https://stahlm.github.io/ENS_215/Data/NOAA_State_Precip_LabData.csv")
```

```
precip_data %>%
  filter(Year >= 1980,
         state_cd == "CA") %>%
  mutate(date = ymd(paste(Year,Month,15)) ) %>%
  ggplot(aes(x = date, y = Precip_inches)) +
  geom_line() +
  geom_point() +
  theme_classic() +
  labs(x = "Year",
       y = "Precip (inches)")
```

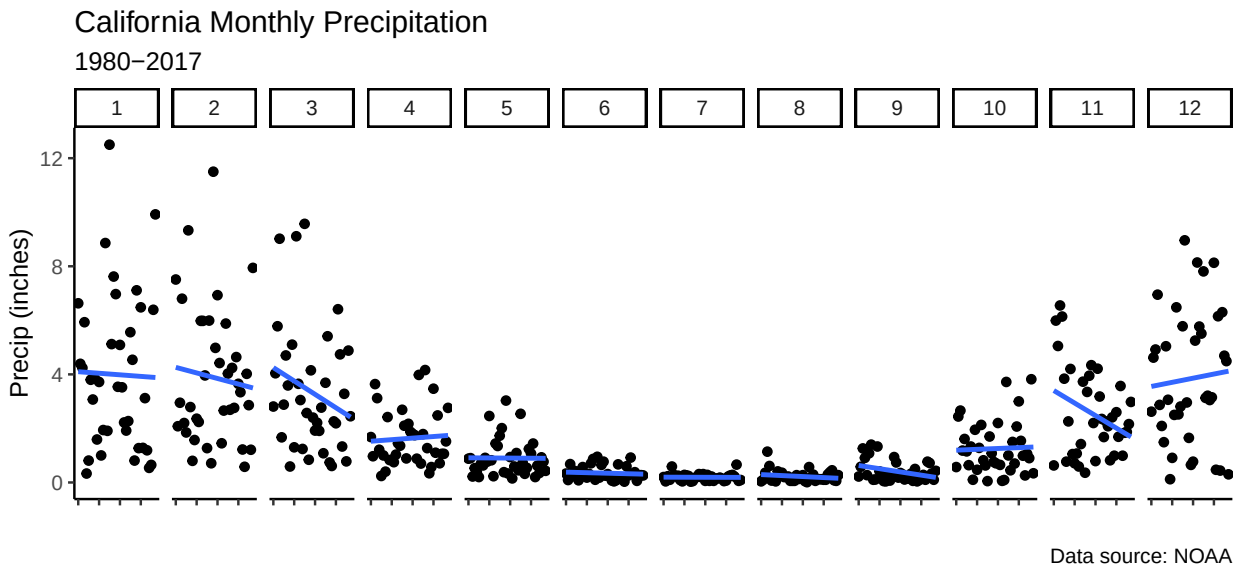


We can see the monthly precipitation for California from 1980-2017, however it is very difficult to see what the seasonal patterns are (e.g. wet vs. dry seasons) and also if there have been long-term trends in a given month (e.g. is February getting wetter?). A cycle plot allows us to answer these types of questions.

In a cycle plot we will plot a time-series of each months data (e.g. All of the January data ordered from earliest to most recent year, all of the Feb data ordered from earliest to most recent year,...).

Below is a cycle plot of the California monthly precipitation data from 1980 onward. Thus the first panel (labeled "1") has the monthly precipitation for all of the Januaries on record, with the left-most point being Jan 1980, the next point Jan 1981,..., the last point Jan 2017.

```
precip_data %>%
  filter(Year >= 1980,
         state_cd == "CA") %>%
  ggplot(aes(x = Year, y = Precip_inches)) +
  geom_point() +
  geom_smooth(se = F, method = "lm") +
  facet_wrap(~ Month, ncol = 12) +
  theme_classic() +
  theme(axis.text.x = element_blank()) +
  labs(x = "",
       y = "Precip (inches)",
       title = "California Monthly Precipitation",
       subtitle = "1980-2017",
       caption = "Data source: NOAA")
```



Look at the above figure.

- Can you identify any seasonal patterns (i.e. wet season vs. dry season)
- Can you identify any trends in particular months (i.e. are some months getting drier)
- Once you've done that, then create a cycle plot for the Schoharie Creek streamflow data. Use this figure to identify season patterns (if they exist) and trends in particular months (if they exist). Think about the implications of these results and discuss with your neighbor.

The Schoharie Creek streamflow data is here

```
flow <- read_csv("https://stahlm.github.io/ENS_215/Data/USGS_flow_01351500.csv") %>%
  drop_na() %>%
  filter(Year >= 1940 & Year <= 2016) %>%  # select years 1940 through 2016
  mutate(date = make_date(Year, Month, Day)) # create a Date column that has the dates as an R date obj
```

Data modeling: Basic regression

As scientists and engineers (or anyone else who deals with data) we seek to better understand the processes recorded by our data. Throughout the term, we have been exploring, summarizing, and visualizing data to help gain insight into the topic of interest and to understand the relationship between variables in our datasets.

Now we are at the point where we can advance into data modeling. The ModernDive text (*excerpt below*), does a great job at describing the concept of data modeling:

> The fundamental premise of data modeling is to make explicit the relationship between:
>
> - an outcome variable y, also called a dependent variable and
> - an explanatory/predictor variable x, also called an independent variable or covariate.
>
> Another way to state this is using mathematical terminology: we will model the outcome variable y as a function of the explanatory/predictor variable x. Why do we have two different labels, explanatory and predictor, for the variable x? That's because roughly speaking data modeling can be used for two purposes:

1. **Modeling for prediction:** When we model for prediction we are interested in predicting the outcome of y based on the information contained in the predictor variable(s). Here we care less about understanding the fundamental relationship between x and y and more about making predictions of y based on

information from x. For example, imagine you are a public health official in Bangladesh and you would like to be able to predict arsenic concentrations in wells based on the depth of the well – this will help you identify at risk wells. Here you don't really care about the fundamental reasons why arsenic might vary with depth, though you care very much about making accurate predictions of groundwater arsenic based on the depth of the well.

2. **Modeling for explanation:** When modeling for explanation you want to describe and understand the conceptual/fundamental implications of the relationship between the outcome variable y and the explanatory variable x. Imagine the scenario in Bangladesh, however in this situation you are a hydrologist and would like to understand the scientific reasons why arsenic concentrations depend on depth. Here you would like to interpret the significance of the relationship and what it might imply with regards to the physical/chemical processes driving arsenic release.

There are many techniques and approaches to data modeling, though in today's lecture we will focus first on **linear regression** and then move onto non-parametric regression.

Linear regression

Linear regression is one of the most commonly-used data models and is relatively easy to understand. We will focus on *simple linear regression* which is when we have a single numeric explanatory variable x that we will use to explain/predict the outcome of the numeric dependent variable y.

A linear model is of the following form:

$y = \beta_0 + \beta_1 * x$

Where β_0 represents the intercept and β_1 is the slope.

A linear regression finds the slope and intercept parameters that generate the best predictions (*i.e.* result in the minimum possible difference between the observed values of y and the predicted values of $\hat{y}$).

We'll use the BGS Bangladesh groundwater chemistry data to test out linear regressions.

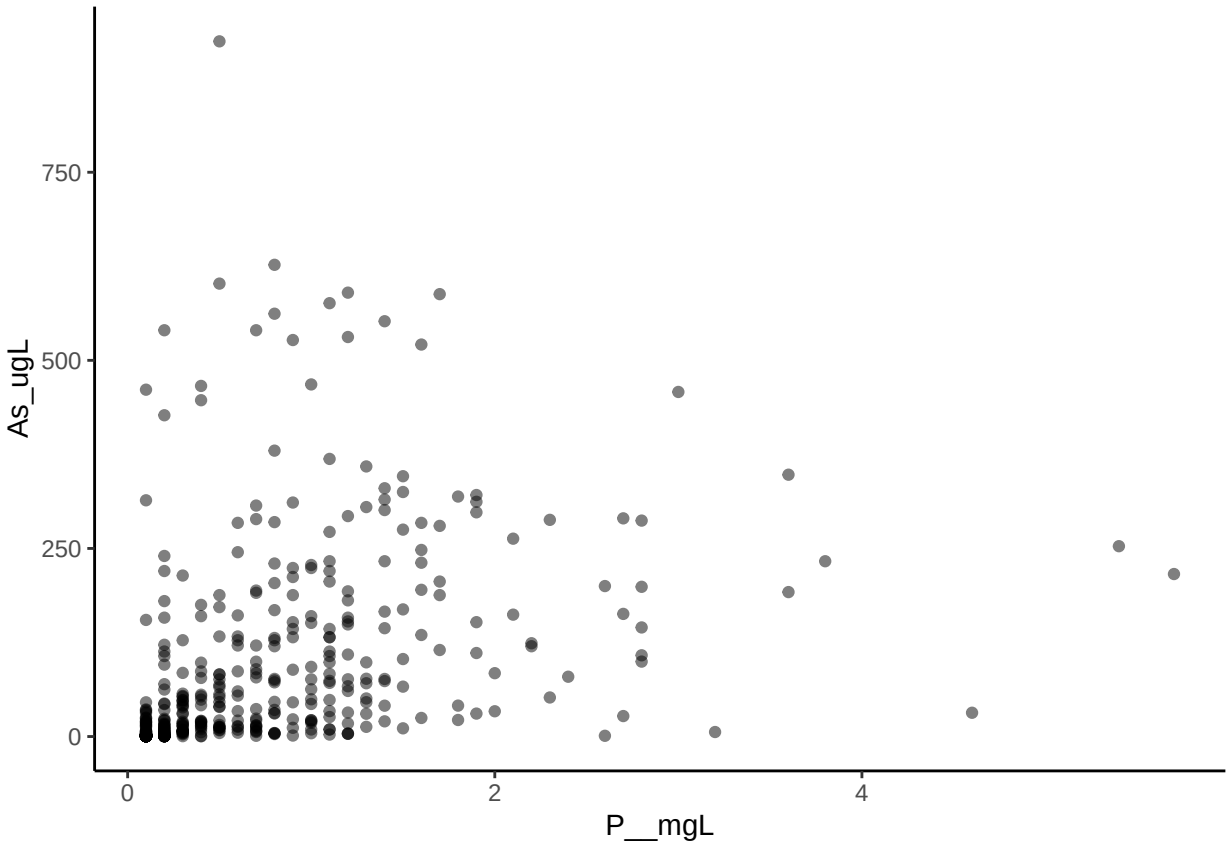
Now let's load in the BGS Bangladesh groundwater chemistry data to use in the analysis to follow.

```
bangladesh_gw <- read_csv("https://stahlm.github.io/ENS_215/Data/NationalSurveyData_DPHE_BGS_LabData.cs
  drop_na()
```

Right now you are a hydrologist and you are interested in the understanding how phosphate PO_4 concentrations affect arsenic concentrations in the groundwaters of Bangladesh. You know theory suggests that phosphate can desorb (knock-off of solid minerals and put into the water) arsenic from the solid aquifer material and thus lead to increases in the concentration of arsenic in the groundwater. You would like to conduct a regression to better understand if there is in fact a relationship between phosphate and arsenic concentrations in the groundwater and what this might imply scientifically.

Let's first generate a scatter plot to see how the two variables relate to one another. Let's focus on just shallow wells from the Dhaka division in our analysis.

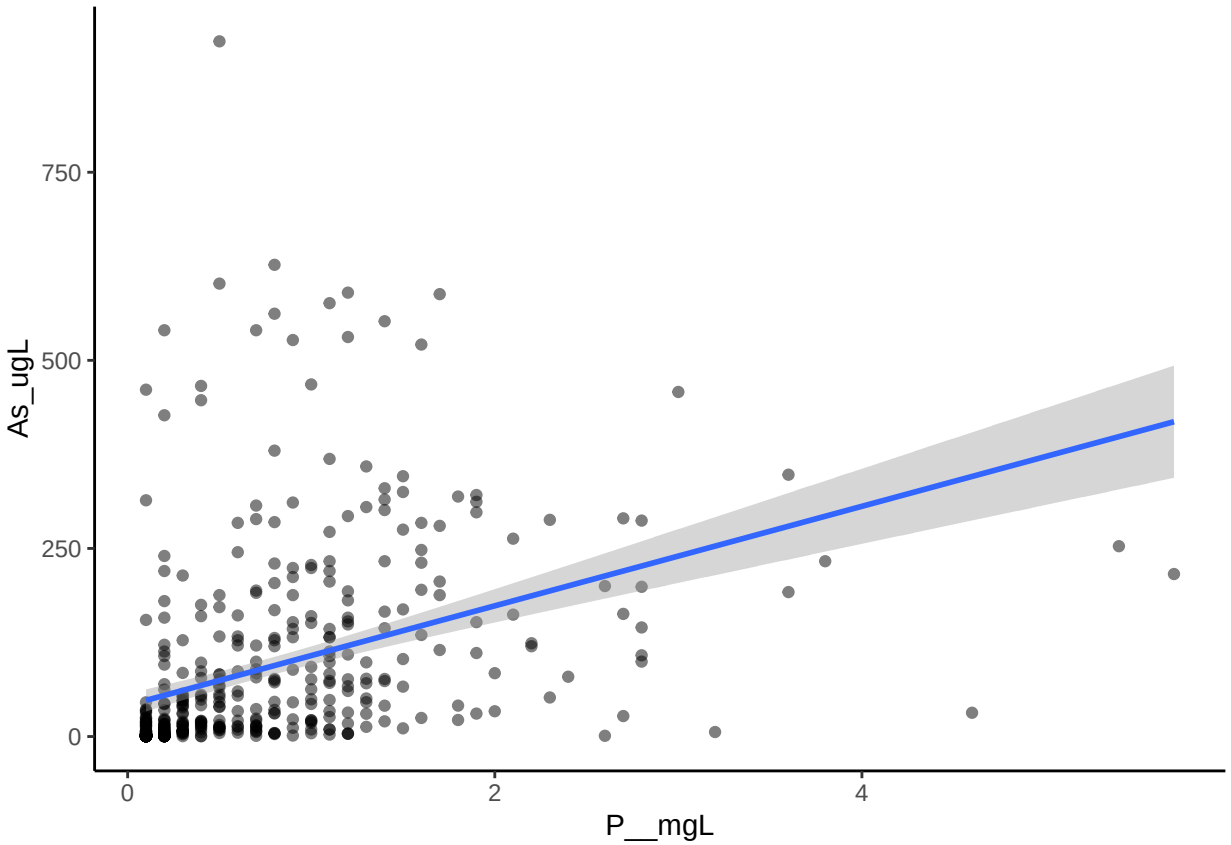
```
Dhaka_gw <- bangladesh_gw %>%
  filter(DIVISION == "Dhaka", WELL_DEPTH_m < 40)

Dhaka_gw %>%
  ggplot(aes(x = P__mgL, y = As_ugL)) +
  geom_point(alpha = 0.5) +
  theme_classic()
```



So it looks like there might be a relationship between the two variables. Let's fit a regression line to the data and plot the fitted line on our graphic.

To plot a regression line we can use the `geom_smooth()` function and specify that we want the fitted line to be a **linear model**. We specify this by setting `method = "lm"` in the `geom_smooth()` function call.

```
Dhaka_gw %>%
  ggplot(aes(x = P__mgL, y = As_ugL)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm") +
  theme_classic()
```



You can see that there is a positive relationship between phosphate and arsenic in the groundwater. Thus, higher concentrations of phosphate generally indicate higher concentrations of arsenic. As hydrologists we are interested in understanding the fundamental relationship between phosphate and arsenic – in this case the positive relationship (i.e. the positive slope of the linear fit) is consistent with geochemical theory.

Note that our linear model is indicated by the blue line. The gray band around the line is the *standard error*, which is a measure of the uncertianty of the model fit. You can remove the standard error by specifying `se = FALSE` in the `geom_smooth()` function call.

Now that we've created a graphic showing the linear regression fit to our data, we would like to actually know the slope and intercept of our model fit. To get this information we can use the `lm()` function to perform a linear regression. The `lm()` function does the exact same thing that `geom_smooth(method = "lm")` does, except the `lm()` function allows us to save the model output as opposed to simply plotting the results.

Using `lm()` to perform a linear regression is relatively straightforward – the sytax is as follows:

```
lm(data = your_dataframe, formula = y ~ x)
```

Where, `y` is the dependent variable, `x` is the explanatory/predictive variable and `your_dataframe` is the data frame that contains your `x` and `y` data.

Let's compute the linear regression to determine the relationship between arsenic and phosphorus. In mathematical terms we are solving for the slope (β_1) and intercept (β_0) of the following equation:

$Arsenic = \beta_0 + \beta_1 * phosphorus$

```
lm(data = Dhaka_gw, formula = As_ugL ~ P__mgL)
```

```
##
## Call:
```

```
## lm(formula = As_ugL ~ P__mgL, data = Dhaka_gw)
##
## Coefficients:
## (Intercept)       P__mgL
##       41.44        66.14
```

You can see from out linear regression that the intercept $\beta_0 = 41.44$ and the slope $\beta_1 = 66.14$

Thus the equation is:

$Arsenic = 41.44 + 66.14 * phosphorus$

This means that for every increase of 1 mg/L phosphorus, there is a 66.14 ug/L increase in arsenic. As hydrologists this is useful from a *modeling for explanation* perspective as it can help provide insight into the fundamental understanding of arsenic mobilization. The linear model is also useful from a *modeling for prediction* perspective as it can help us to predict arsenic concentrations if we know the phosphorus concentration and it can also help us to predict how arsenic concentrations will change with any changes in phosphorus.

Exercises

- Examine the relationship between several other pairs of variables in the `bangladesh_gw` dataset.
  - Create scatter plots and add a linear fit to the graphic. In some cases you may find it helpful to perform a linear fit to the log of both variables.
  - Examine the slope and intercept of your fit and think about and discuss what they imply.

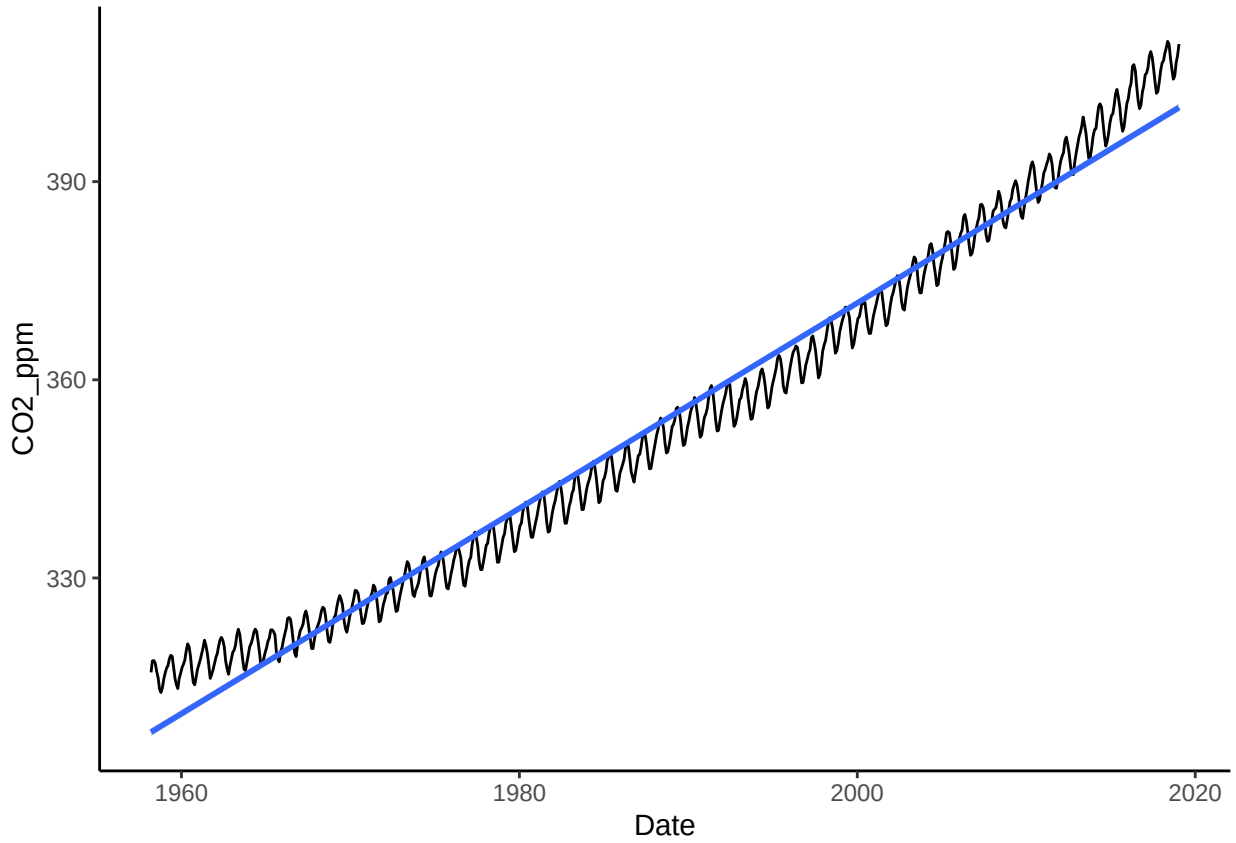
Regression – continued

Let's load in some data of atmospheric CO_2 concentrations that have been recorded at Mauna Loa, Hawaii since the late-1950s. The dataset has monthly average concentrations.

```
mauna_loa <- read_csv("https://stahlm.github.io/ENS_215/Data/Mauna_Loa_CO2.csv", skip = 2) %>%
  mutate(Date = decimal_date(make_date(Year, Month, 15)))  # add a column that has the date information
```

Let's plot the data and add a linear fit to the data. This will help us to characterize the change in CO_2 concentrations with time.

```
mauna_loa %>%
  ggplot(aes(x = Date, y = CO2_ppm)) +
  geom_line() +
  geom_smooth(method = "lm") +
  theme_classic()
```



- What do you observe with respect to atmospheric CO_2 concentrations over time?
- Does the rate of change of CO_2 with time look constant? If not, does the rate of change appear to be increasing or decreasing? What does this imply with respect to using a linear model?

So we've seen how to generate a linear fit to the data using the `lm()` function as well as the `geom_smooth()` function, which allows us to easily plot the fitted model. Now let's learn how to get a bit more information about our fitted model using a few functions from the `moderndive` package. First, we'll fit a linear model to the Mauna Loa CO_2 data and save the fitted model to an R data object.

```
lm_mauna_loa <- lm(CO2_ppm ~ Date, data = mauna_loa)
```

Mathematically, our model is represented as:

$CO_2 = \beta_0 + \beta_1 * Date$

The slope β_0 tells us how the concentration of CO_2 (in ppm) in the atmosphere changes with time. Since the `Date` variable is in units if years, a slope of 2 would tell us that the CO_2 concentration increases by 2 ppm each year.

To get the coefficients β_0 and β_1 along with information about how confident (mathematically speaking) we are in these estimates, we can use the `get_regression_table()` function from the `moderndive` package.

```
get_regression_table(lm_mauna_loa, digits = 5)
```

```
## # A tibble: 2 x 7
##   term      estimate std_error statistic p_value lower_ci upper_ci
##   <chr>        <dbl>     <dbl>     <dbl>   <dbl>    <dbl>    <dbl>
## 1 intercept -2737.    17.4         -158.       0 -2771.   -2703.
## 2 Date          1.55   0.00873      178.       0     1.54     1.57
```

The table output by `get_regression_table()` gives us the intercept and slope estimates from the fitted linear model. We see that the slope is 1.55 – meaning that the linear model predicts a 1.55 ppm change in CO_2 concentration each year. The other columns in the table provide information about the mathematical confidence of these slope and intercept estimates. We don't have sufficient time in this class to dive into the meaning of the the *p-value*, but generally speaking it tells us how confident we are that the computed estimate is different from zero – a smaller *p-value* means we are more confident in this fact. Typically a p-value < 0.05 is considered to be a "good" indication that the estimate differs from zero. The `lower_ci` and `upper_ci` are confidence intervals on the estimate – loosely speaking they provide a set of reasonable bounds on the "true" value of the parameter.

The `get_regression_summaries()`function from the `moderndive` package provides summary statistics that tell us how well our modeled (fitted) values match the observations. While we don't have time in this class to cover the details of these statistics, they are nonetheless worth being aware of. The *r-squared* statistic is one that you will commonly hear reference to – it tells us how well our explanatory variable (in this example the year) explains the dependent variable (in this example CO_2). An r-squared of 1 means the explanatory variable perfectly explains the variablity in the dependent variable and a value of 0 means that the explanatory variable has no explanatory power.

```
get_regression_summaries(lm_mauna_loa)
```

```
## # A tibble: 1 x 8
##   r_squared adj_r_squared   mse  rmse sigma statistic p_value    df
##       <dbl>         <dbl> <dbl> <dbl> <dbl>     <dbl>   <dbl> <dbl>
## 1     0.978         0.977  17.2  4.14  4.15    31709.       0     2
```

Residuals

Once you've fit a model to data you should examine the residuals. A residual is the difference between an observation and the model predicted value. Conceptually, when looking at a graph that has both the observed and modeled data plotted, the distance between a given point and the model estimated value (i.e. a vertical line from the observed point to the modeled line) is a points residual.

Below is a graphic showing a linear model fit to teaching score data, where the explanatory variable is the instructors beauty score. The residual for a single observation is depicted as the vertical distance between the observation (red circle) and the linear model. You can find this example in your ModernDive text.

Let's explore the residuals for our linear model model fit to the Mauna Loa data. We can use the `get_regression_points()` function from the `moderndive` package to obtain a data frame with both the observed and modeled results, along with the residuals.

```
lm_mauna_loa_points <- get_regression_points(lm_mauna_loa)

head(lm_mauna_loa_points)
```

```
## # A tibble: 6 x 5
##      ID CO2_ppm  Date CO2_ppm_hat residual
##   <int>   <dbl> <dbl>       <dbl>    <dbl>
## 1     1    316. 1958.        307.     9.05
## 2     2    317. 1958.        307.    10.7
## 3     3    318. 1958.        307.    10.6
## 4     4    317. 1958.        307.    10.1
## 5     5    316. 1959.        307.     8.68
## 6     6    315. 1959.        307.     7.62
```

You can see that `lm_mauna_loa_points` has `Date` and observed CO~2 (`CO2_ppm`) as well as the estimated CO_2 from the linear model (`CO2_ppm_hat`) and the residuals (`residual`).

Now, that we have a data frame with the residuals, let's create a **residual-dependence plot**. A **residual-**

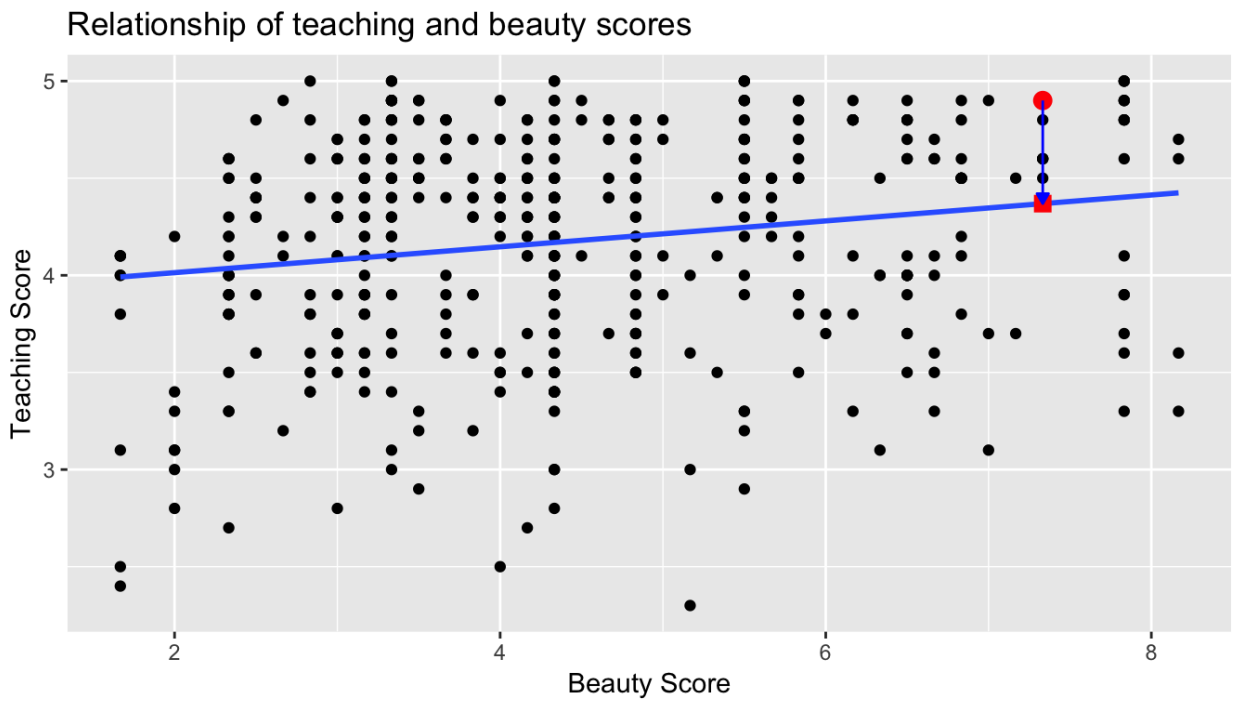
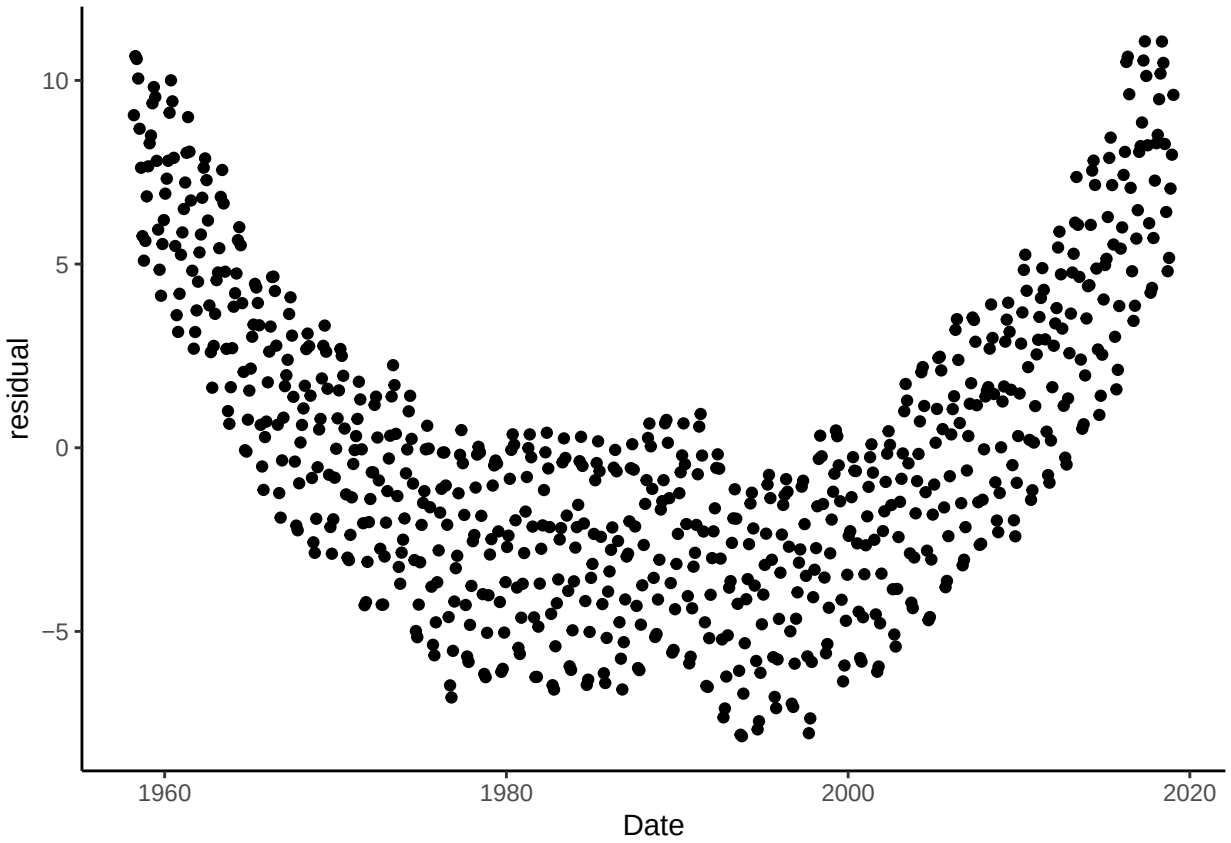


Figure 1: Image source: *ModernDive*

dependence plot displays the residuals as a function of our explanatory variable, which in this case is the `Date`. This type of plot is useful as it helps us to identify any patterns in the residuals.

```
lm_mauna_loa_points %>%
  ggplot(aes(x = Date, y = residual)) +
  geom_point() +
  theme_classic()
```



A good model fit should have small residuals, which are centered and symmetrically distributed around zero. You can see from the above plot that there is a clear "U-shaped" pattern to the residuals from our linear model fit to the Mauna Loa data. In particular the linear model under-predicts CO_2 concentrations (*i.e.* positive residual) for early times (~ pre-1970) and late times (~ post 2000) and over-predicts from approximately 1970-2000.

The fact that the residuals show a dependence on our explanatory variable (Date) indicates that the model we have chosen is not able to fully capture the features of the data. You likely already suspected this just by looking at our original plot of the observed and modeled data. You probably observed that while the linear model did a good job at capturing the general trend of increasing CO_2 concentrations, it wasn't able to capture the curved (*i.e.* non-linear) nature of the data.

To better represent the Mauna Loa data we will need to rely something other than a linear model.

Non-parameteric regression

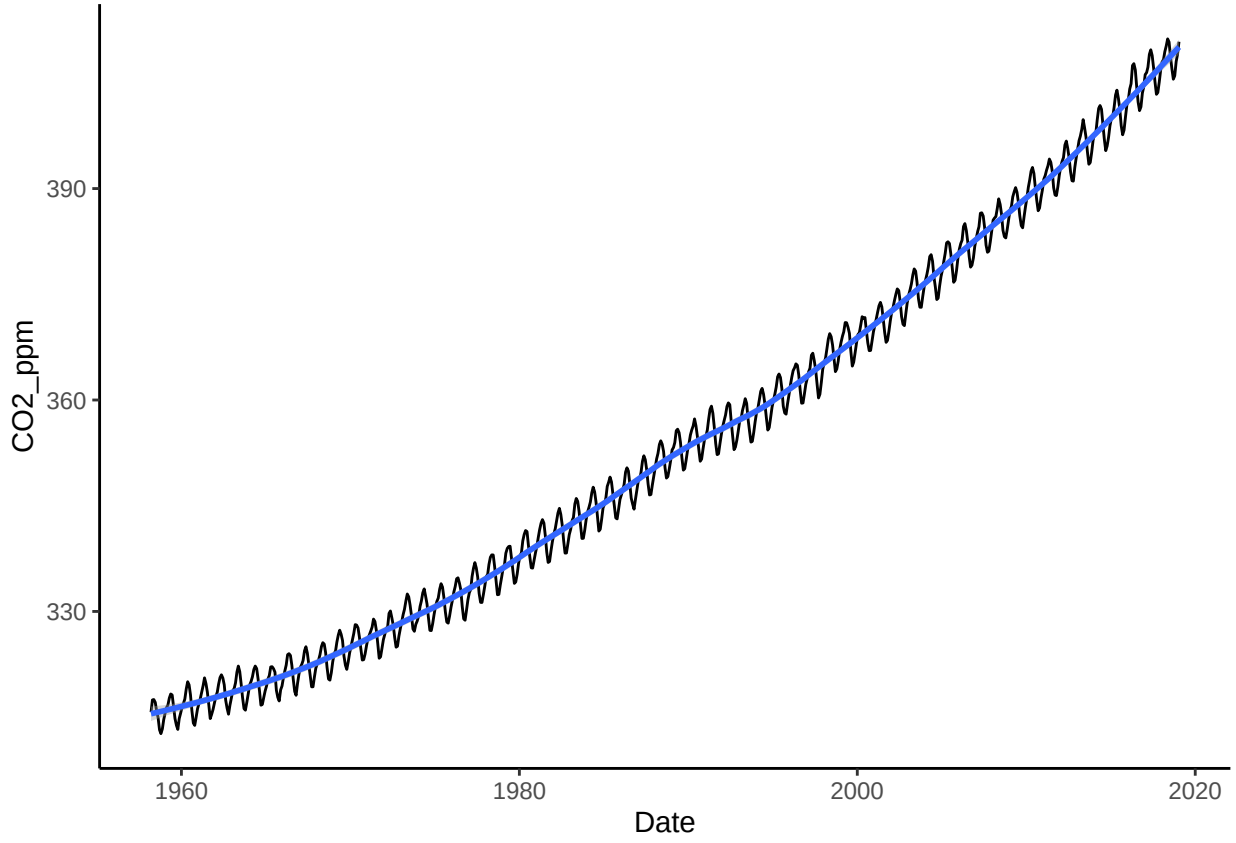
There are many other types of models that we could choose from other than a linear model. Here we are going to apply a **non-parametric** model. Non-parametric models do not impose a structure on the data. Consider a linear fit, which is a **parametric model** – this type of model imposes a function of the form $y = \beta_0 + \beta_1 * x$, where there is a finite set of fixed parameters (e.g. β_0 and β_1). A **non-parametric** model does not take a predetermined form and there is not a fixed set of parameters that describes the model.

LOESS is a commonly used non-parametric regression model. LOESS (locally estimated scatterplotr smoothing) fits a regression across small segments of the data and links these fits together to create a smooth curve. A LOESS curve is an excellent method for teasing out structure and features in a dataset – especially when there is not a known parametric equation to describe the data.

To plot a LOESS regression we can use the `geom_smooth()` function and specify `method = loess`. You can see in the code below that we've specified `span = 1/4`. This tells the LOESS smoothing algorithm to consider

1/4 (25%) of the points at each of the local regressions. Thus, the regression computed at each point along the x-axis will consider the a span of data (around that local point) that covers 25% of the all of the points in the dataset. The larger the span the smoother the line. A smaller span value will capture more local features and will result in a wigglier line. Since the span used below is relatively large, our LOESS fit will capture long-term trends but will smooth out the seasonal oscillations.

```
mauna_loa %>%
  ggplot(aes(x = Date, y = CO2_ppm)) +
  geom_line() +
  geom_smooth(method = "loess", span = 1/4) +
  theme_classic()
```



You can see that the LOESS fit appears to capture overall trend of the data much better the the linear model. Let's now examine the residuals from our LOESS model to examine how it performs (and to compare it's performance to the linear model we fit earlier).

We can compute a LOESS estimate using the `loess()` function and then save the results to a data object.

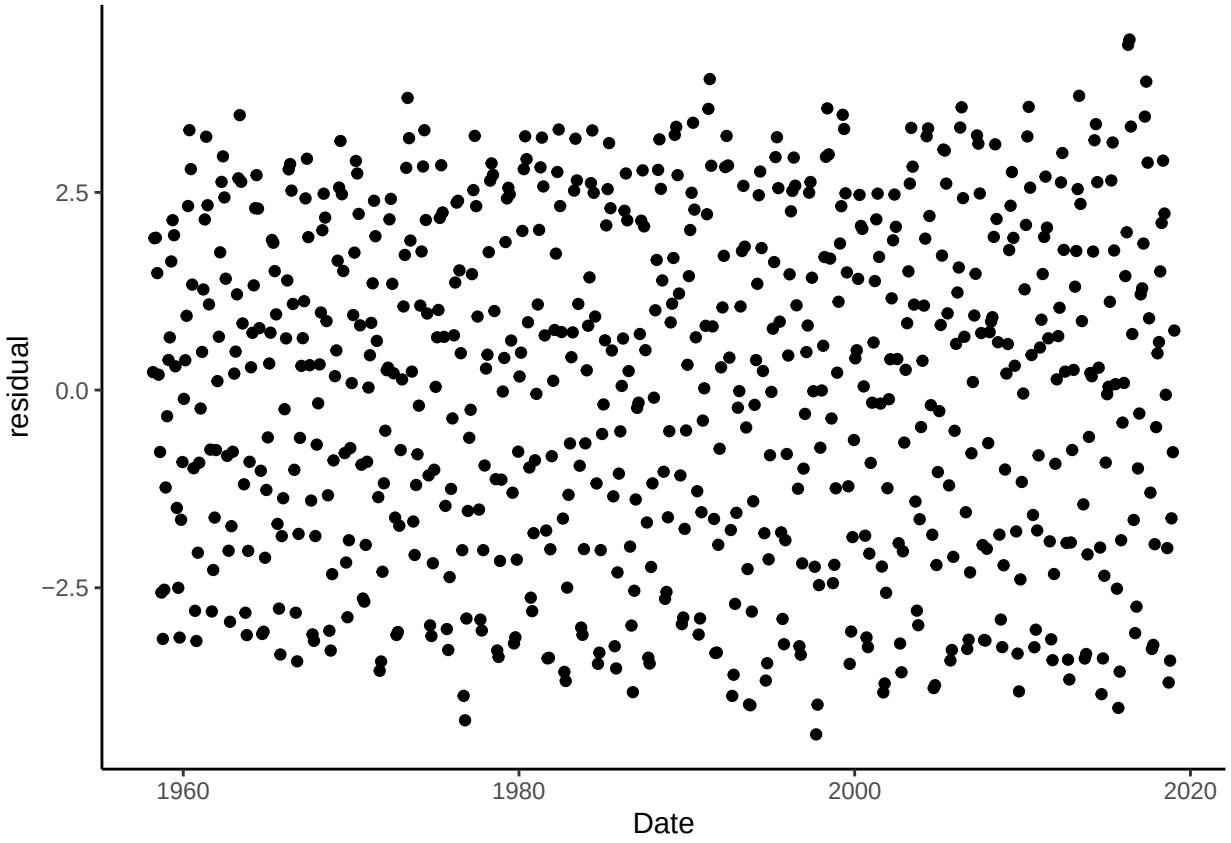
```
loess_mauna_loa <- loess(CO2_ppm ~ Date, data = mauna_loa, span = 1/4)
```

Next, let's save the residuals at each point to a data frame `loess_mauna_loa_residuals` and then plot the residuals as a function of the `Date`.

```
loess_mauna_loa_residuals <- bind_cols(as.tibble(loess_mauna_loa$x), as.tibble(loess_mauna_loa$residuals
  rename(residual = value)
```

```
## Warning: `as.tibble()` is deprecated, use `as_tibble()` (but mind the new semantics).
## This warning is displayed once per session.
```

```
loess_mauna_loa_residuals %>%
  ggplot(aes(x = Date, y = residual)) +
  geom_point() +
  theme_classic()
```

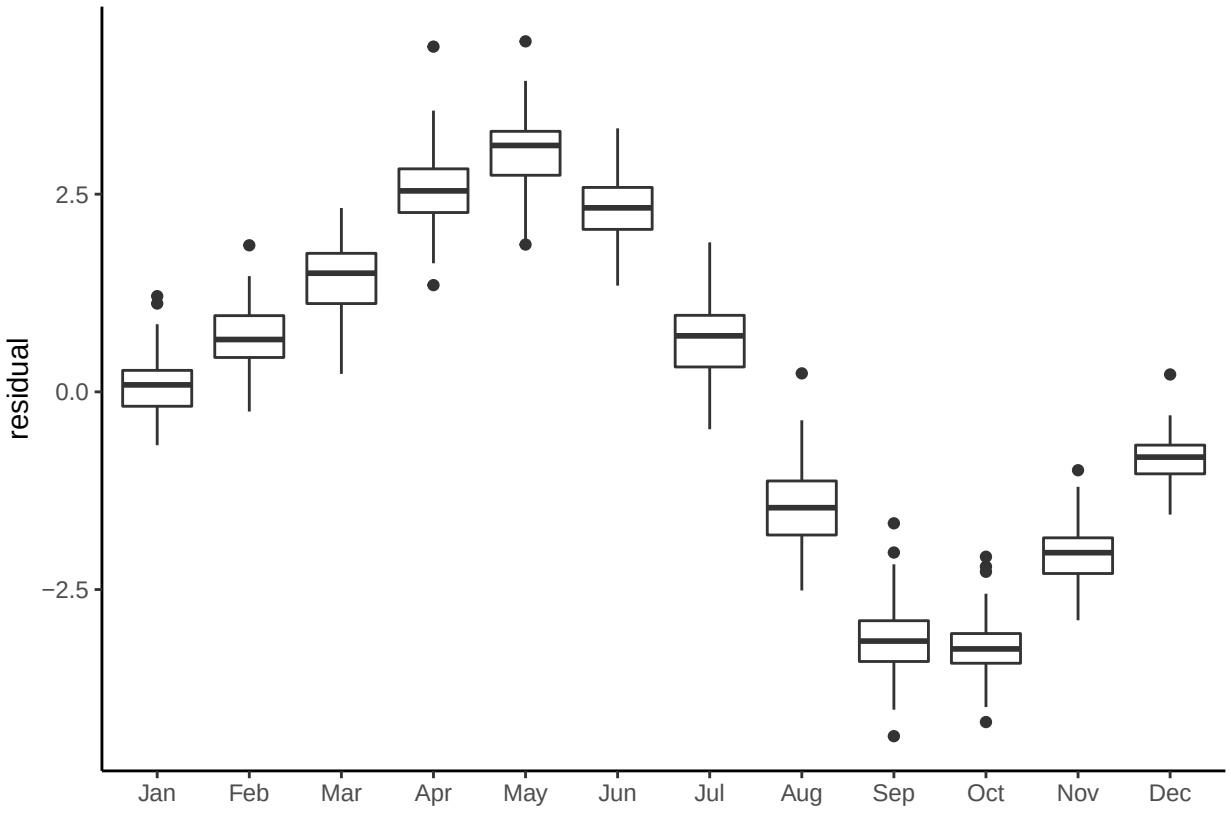


Unlike with our linear model we can see that the residuals from the LOESS fit are centered about zero and do not display any dependence on the explanatory variable `Date`. This is an excellent sign and indicates that are model performs well across the whole range of x-values (Dates).

Since our LOESS fit has a relatively large span (25%), the local regressions at each point considered points within about a 15 yr span (± 7.5). Therefore, the LOESS fit captured the long-term trends but smoothed out the seasonal trends. Since the LOESS fit modeled the long-term trend, but did not model the seasonal oscillations, the residuals should reflect the season component in the Mauna Loa CO_2 data.

This is actually very useful as examining the residuals of the LOESS fit should allow us to characterize the seasonal variations (i.e. variations within a given year) of atmospheric CO_2. Let's create a boxplot of the monthly residuals.

```
loess_mauna_loa_residuals %>%
  ggplot(aes(x = factor(month(date_decimal(Date), label = TRUE)), y = residual)) +
  geom_boxplot() +
  labs(x = "") +
  theme_classic()
```



From the boxplot of the monthly residuals we have clearly characterized the seasonal patterns in the data. We can see that each year CO_2 peaks around May and reaches a minimum around Sept-Oct. This peak in CO_2 is due to decreased primary production during the winter months (in the northern hemisphere) which allows CO_2 to accumulate in the atmosphere. At the onset of spring, increased primary production helps to draw down CO_2 and this drawdown continues until the onset of fall in Sept-Oct.

Extra section: Combining different figures into a single graphic

If you finish the above work in time you can proceed on. If not you can work go over this section after class or in lab.

Oftentimes we have several different (but related) figures that we would like to combine into a single graphic. For instance you may want to have a time-series of streamflow data and a boxplot of that same data shown in a single combined graphic. There is a very helpful package called `patchwork` that allows you to easily do this. This will likely be very helpful as you put together your final projects.

Let's first install the package. To do this, type the following code to your console and run it.

```
devtools::install_github("thomasp85/patchwork")
```

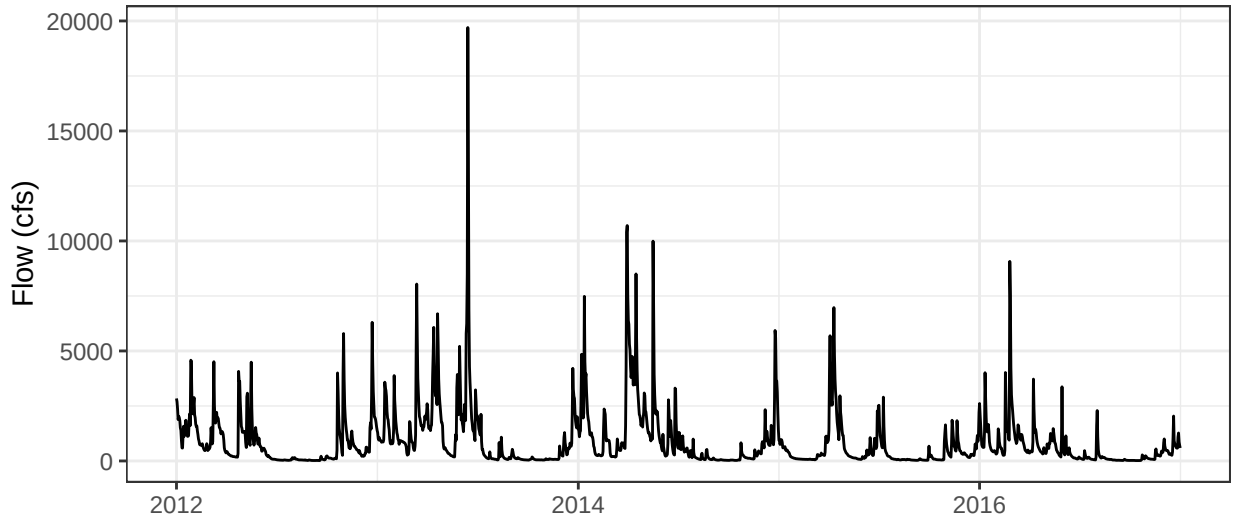
Once the package is installed let's load it in

```
library(patchwork)
```

Now let's create three individual graphics that we will then combine using the functionality from the `patchwork` package.

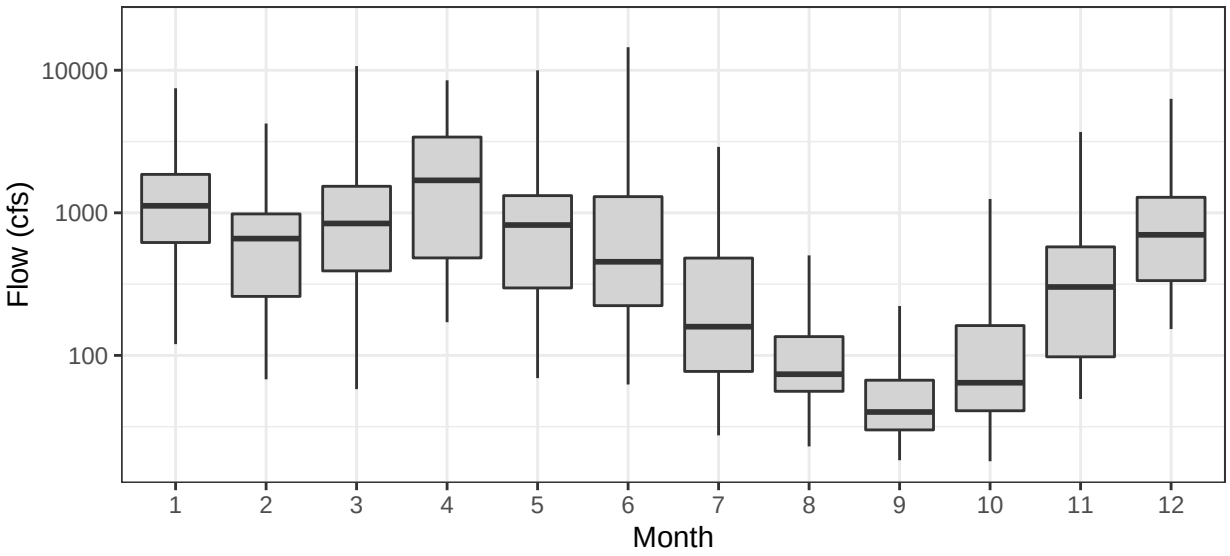
Now let's create a time-series of the daily flows (selecting only 2012 onward) in Schoharie Creek.

```r
fig_stream_1 <- flow %>%
  filter(Year >= 2012) %>%
  ggplot(aes(x = date, y = flow_cfs)) +
  geom_line() +
  theme_bw() +
  labs(x = "",
       y = "Flow (cfs)")

fig_stream_1
```



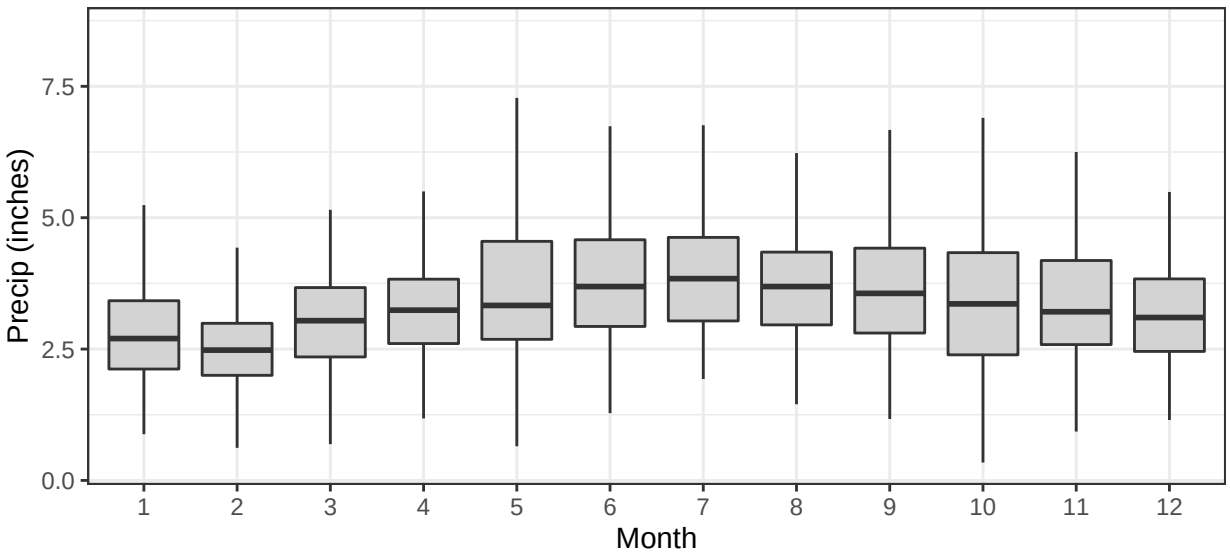
Next let's create a boxplot of daily flow for each month.

```r
fig_stream_2 <- flow %>%
  filter(Year >= 2012) %>%
  ggplot(aes(x = as.factor(Month), y = flow_cfs)) +
  geom_boxplot(outlier.shape = NA, fill = "lightgrey") +
  scale_y_log10() +
  theme_bw() +
  labs(x = "Month",
       y = "Flow (cfs)")

fig_stream_2
```



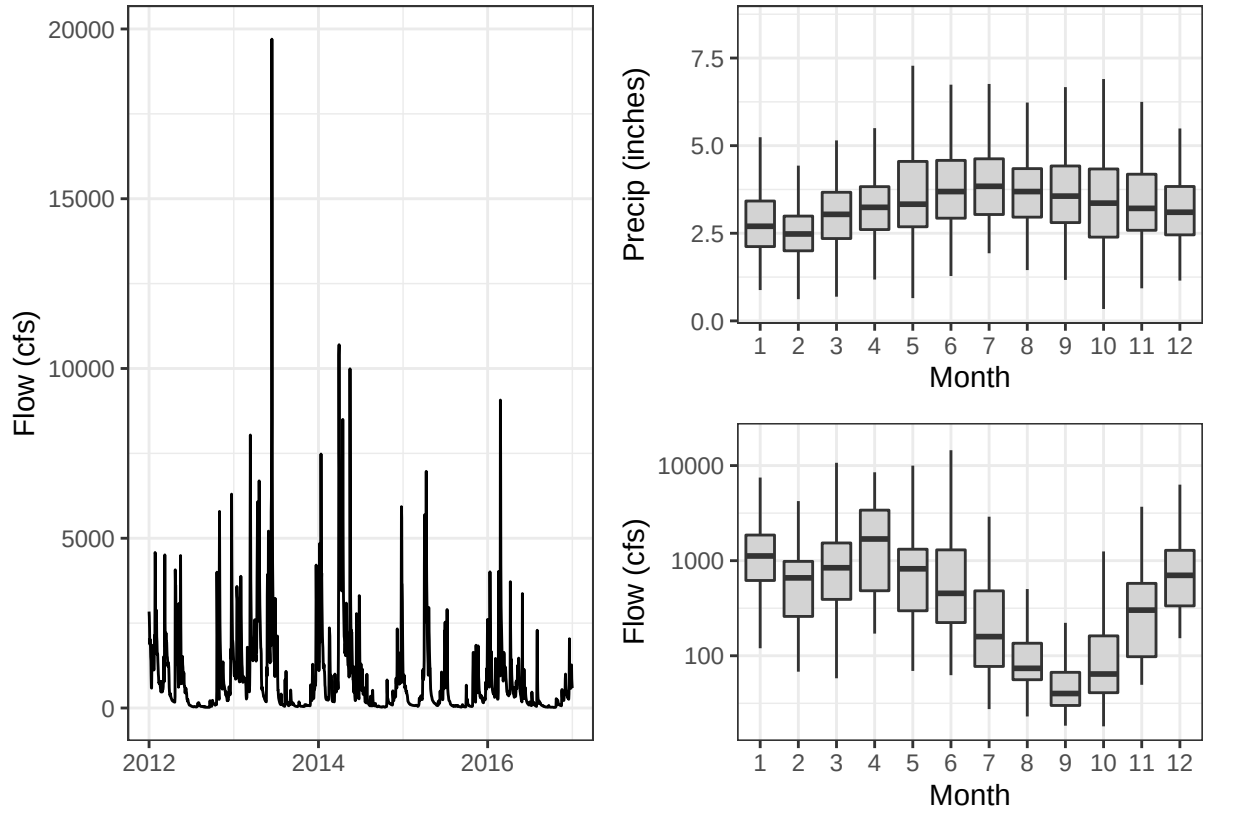
Finally let's create a boxplot of monthly precipitation for New York state. You can imagine that precipitation amount will influence streamflow and thus you might want to show this data along with the streamflow data.

```
fig_precip <- precip_data %>%
  filter(state_cd == "NY") %>%
  ggplot(aes(x = as.factor(Month), y = Precip_inches)) +
  geom_boxplot(outlier.shape = NA, fill = "lightgrey") +
  labs(x = "Month",
       y = "Precip (inches)") +
  theme_bw()

fig_precip
```



Now let's combine these figures. To add graphics side-by-side you use the `+` symbol. To stack graphics on top of one another you use the `/` symbol. Note that you can also use the `|` to place graphic side by side. You can control the order of precedence of operations by using parentheses. As you can see below we have a panel with `fig_stream_1` on the left and on the right we have `fig_precip` stacked on top of `fig_stream_2`.

```
fig_stream_1 | (fig_precip / fig_stream_2)
```



You can also add annotation to your combined graphic using the `plot_annotation()` function. The `tag_levels` argument allows us to specify how each of the sub-figures is labeled. In this case we specify that capital letters (A for the 1st figure, B for the 2nd,...) are used.

```
( fig_stream_1 + (fig_precip / fig_stream_2) ) +
  plot_annotation(title = 'Schoharie Creek Hydrology',
                  tag_levels = 'A')
```

Schoharie Creek Hydrology