14: Time Series

Environmental Data Analytics | Kateri Salk

Spring 2020

Objectives

1. Discuss the purpose and application of time series analysis for environmental data
2. Choose appropriate time series analyses for trend detection
3. Address the influence of seasonality on time series analysis
4. Interpret and communicate results of time series analyses

Set up

```
getwd()
```

```
## [1] "/Users/ks501/Documents/GithubRepos/Environmental_Data_Analytics_2020"
```

```
library(tidyverse)
library(lubridate)
#install.packages("trend")
library(trend)
#install.packages("zoo")
library(zoo)

# Set theme
mytheme <- theme_classic(base_size = 14) +
  theme(axis.text = element_text(color = "black"),
        legend.position = "top")
theme_set(mytheme)

EnoDischarge <- read.csv("./Data/Processed/USGS_Site02085000_Flow_Processed.csv")
EnoDischarge$datetime <- as.Date(EnoDischarge$datetime, format = "%Y-%m-%d")

NCAir <- read.csv("./Data/Processed/EPAair_O3_PM25_NC1819_Processed.csv")
NCAir$Date <- as.Date(NCAir$Date, format = "%Y-%m-%d")
```

Time Series Analysis

Time series are a special class of dataset, where a response variable is tracked over time. The frequency of measurement and the timespan of the dataset can vary widely. At its most simple, a time series model includes an explanatory time component and a response variable. Mixed models can include additional explanatory variables (check out the `nlme` and `lme4` R packages). We will cover a few simple applications of time series analysis in these lessons, with references for how to take analyses further.

Opportunities

Analysis of time series presents several opportunities. For environmental data, some of the most common questions we can answer with time series modeling are:

- Has there been an increasing or decreasing **trend** in the response variable over time?
- Can we **forecast** conditions in the future?

Challenges

Time series datasets come with several caveats, which need to be addressed in order to effectively model the system. A few common challenges that arise (and can occur together within a single dataset) are:

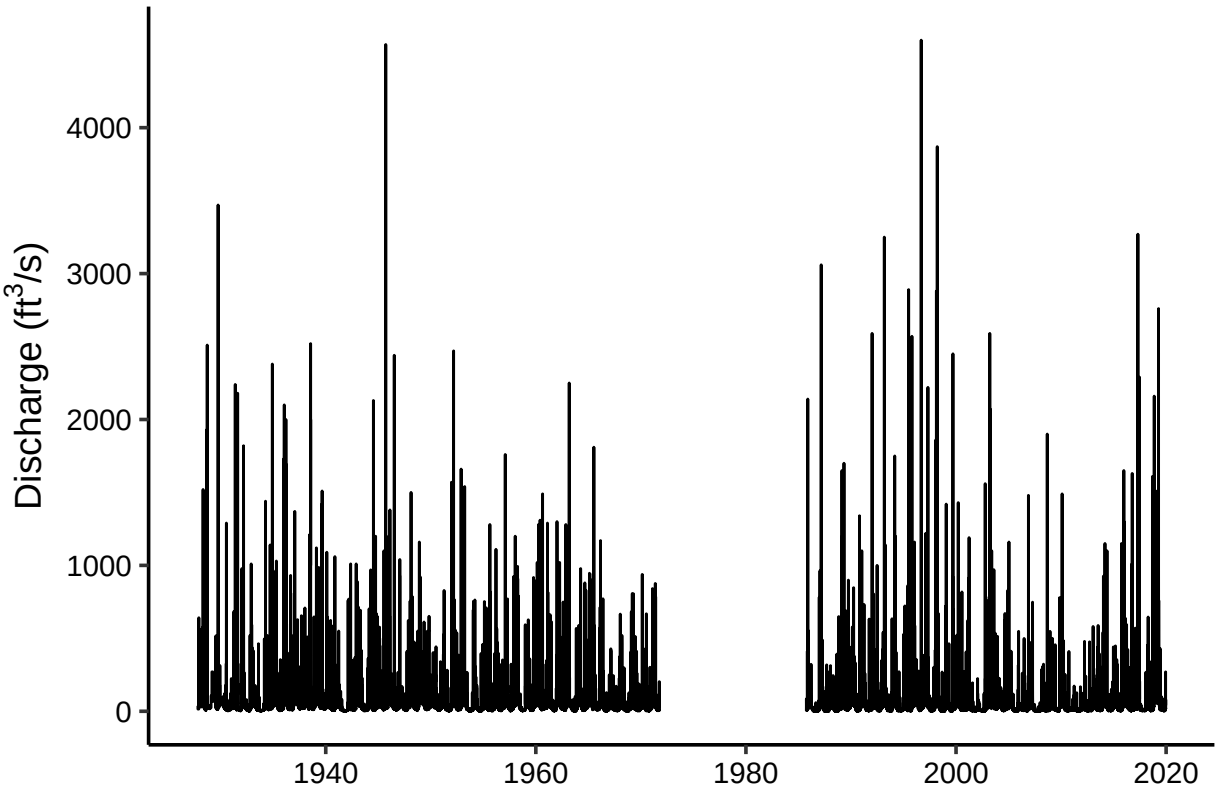
- **Autocorrelation**: Data points are not independent from one another (i.e., the measurement at a given time point is dependent on previous time point(s))
- **Data gaps**: Data are not collected at regular intervals, necessitating *interpolation* between measurements.
- **Seasonality**: Cyclic patterns in variables occur at regular intervals, impeding clear interpretation of a monotonic (unidirectional) trend.
- **Heteroscedasticity**: The variance of the time series is not constant over time
- **Covariance**: the covariance of the time series is not constant over time

Example dataset: Eno River Discharge

River discharge is measured daily at the Eno River gage station. Since we are working with one location measured over time, this will make a great example dataset for time series analysis.

Let's look at what the dataset contains for mean daily discharge.

```
ggplot(EnoDischarge, aes(x = datetime, y = discharge.mean)) +
  geom_line() +
  labs(x = "", y = expression("Discharge (ft"^3*"/s)"))
```



Notice there are missing data from 1971 to 1985. Gaps this large are generally an issue for time series analysis, as we don't have a continuous record of data or a good way to characterize any variability that happened over those years. We will illustrate a few workarounds to address these issues.

Let's start by removing the NAs and splitting the dataset into the early and late years.

```
EnoDischarge.complete <- EnoDischarge %>%
  drop_na(discharge.mean)

EnoDischarge.early <- EnoDischarge.complete %>%
  filter(datetime < as.Date("1985-01-01"))

EnoDischarge.late <- EnoDischarge.complete %>%
  filter(datetime > as.Date("1985-01-01"))
```

Decomposing a time series dataset

A given time series can be made up of several component series:

1. A **seasonal** component, which repeats over a fixed known period (e.g., seasons of the year, months, days of the week, hour of the day)
2. A **trend** component, which quantifies the upward or downward progression over time. The trend component of a time series does not have to be monotonic.
3. An **error** or **random** component, which makes up the remainder of the time series after other components have been accounted for. This component reflects the noise in the dataset.
4. (optional) A **cyclical** component, which repeats over periods greater than the seasonal component. A good example of this is El Niño Southern Oscillation (ENSO) cycles, which occur over a period of 2-8

years.

We will decompose the EnoDischarge.late data frame for illustrative purposes today. It is possible to run time series analysis on detrended data by subtracting the trend component from the data. However, detrending must be done carefully, as many environmental data are bounded by zero but are not treated as such in a decomposition. If you plan to use decomposition to detrend your data, please consult time series analysis guides before proceeding.

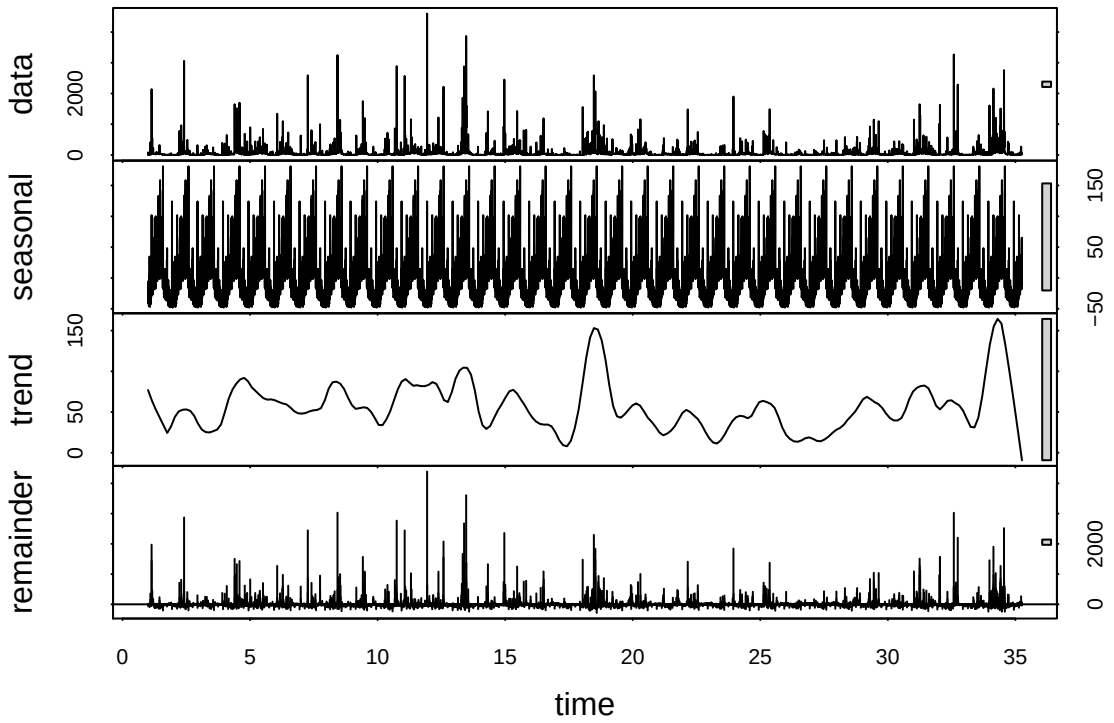
We first need to turn the discharge data into a time series object in R. This is done using the `ts` function. Notice we can only specify one column of data and need to specify the period at which the data are sampled. The resulting time series object cannot be viewed like a regular data frame.

Note: time series objects must be equispaced. In our case, we have daily data with no NAs in the data frame, so we don't need to worry about this. We will cover how to address data that are not equispaced later in the lesson.

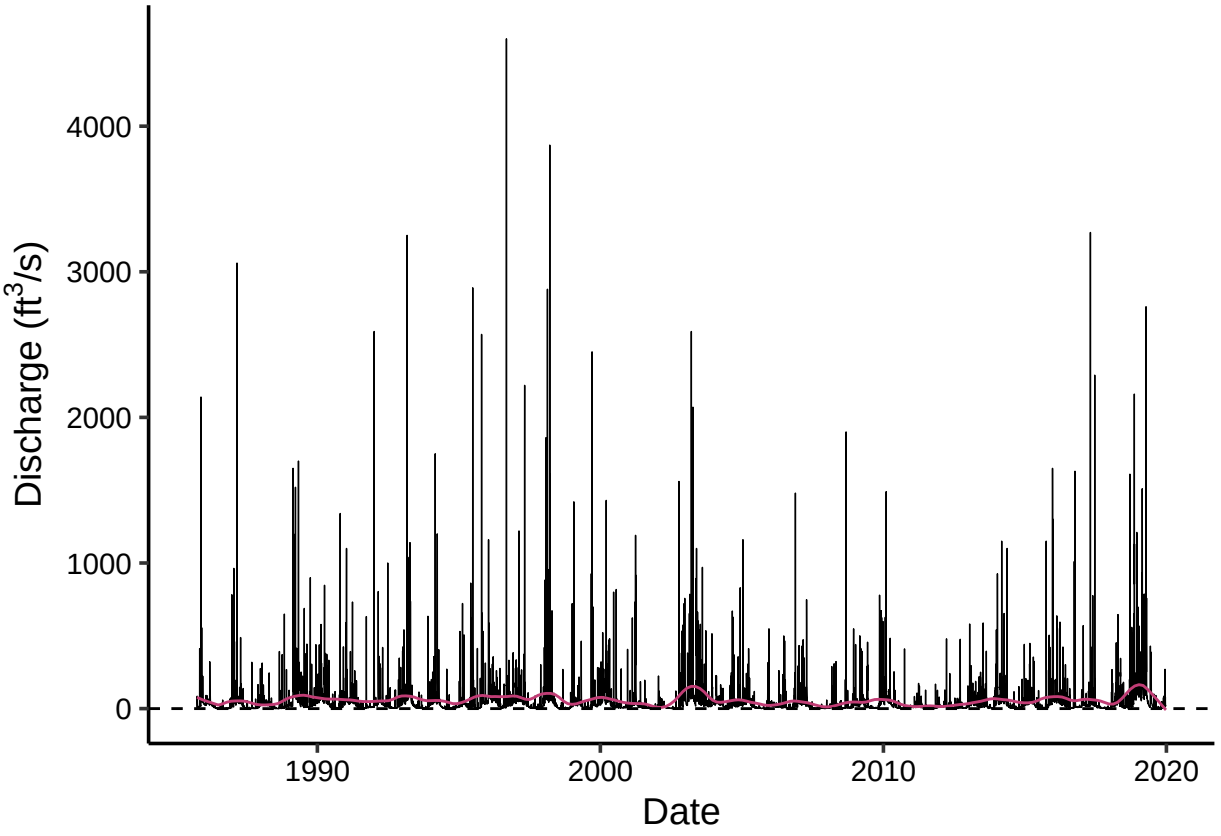
```
EnoDischarge.late_ts <- ts(EnoDischarge.late[[8]], frequency = 365)
```

The `stl function` decomposes the time series object into its component parts. We must specify that the window for seasonal extraction is either "periodic" or a specific number of at least 7. The decomposition proceeds through a loess (locally estimated scatterplot smoothing) function.

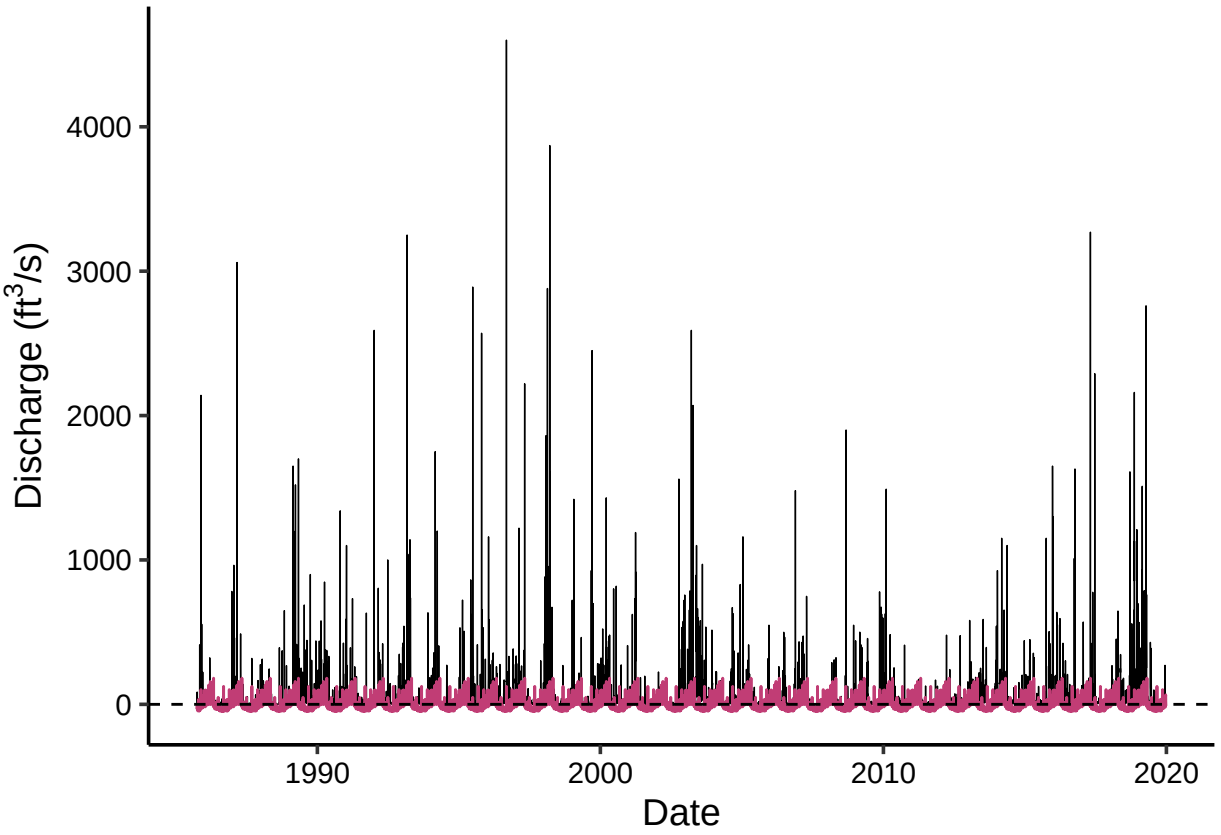
```
?stl
# Generate the decomposition
EnoDischarge.late_Decomposed <- stl(EnoDischarge.late_ts, s.window = "periodic")

# Visualize the decomposed series.
plot(EnoDischarge.late_Decomposed)
```



```r
# We can extract the components and turn them into data frames
EnoDischarge.late_Components <- as.data.frame(EnoDischarge.late_Decomposed$time.series[,1:3])
EnoDischarge.late_Components <- mutate(EnoDischarge.late_Components,
                      Observed = EnoDischarge.late$discharge.mean,
                      Date = EnoDischarge.late$datetime)

# Visualize how the trend maps onto the data
ggplot(EnoDischarge.late_Components) +
  geom_line(aes(y = Observed, x = Date),  size = 0.25) +
  geom_line(aes(y = trend, x = Date), color = "#c13d75ff") +
  geom_hline(yintercept = 0, lty = 2) +
  ylab(expression("Discharge (ft"^3*"/s)"))
```




```r
# Visualize how the seasonal cycle maps onto the data
ggplot(EnoDischarge.late_Components) +
  geom_line(aes(y = Observed, x = Date),  size = 0.25) +
  geom_line(aes(y = seasonal, x = Date), color = "#c13d75ff") +
  geom_hline(yintercept = 0, lty = 2) +
  ylab(expression("Discharge (ft"^3*"/s)"))
```



Note that the decomposition can yield negative values when we apply a seasonal adjustment or a trend adjustment to the data. The decomposition is not constrained by a lower bound of zero as discharge is in real life. Make sure to interpret with caution!

Trend analysis

Two types of trends may be present in our time series dataset: **monotonic** or **step**. Monotonic trends are a gradual shift over time that is consistent in direction, for example in response to land use change. Step trends are a distinct shift at a given time point, for example in response to a policy being enacted.

Step trend analysis

Step trend analysis works well for upstream/downstream and before/after study design. We will not delve into each of these methods during class, but specific tests are listed below for future reference.

Note: ALWAYS look into the assumptions of a given test to ensure it matches with your data and with your research question.

- **Change point detection**, e.g., `pettitt.test` (package: trend), `breakpoints` (package: strucchange), `chngpt.test` (package: chngpt), multiple tests in package: changepoint
- **t-test (paired or unpaired)**
- **Kruskal-Wallis test**: non-parametric version of t-test
- **ANCOVA**, analysis of covariance

Example: step trend analysis

Let's say we wanted to know whether discharge was higher in the early period or the late period. Perhaps there was a change in the methodology of streamflow measurement between the two periods that caused a differene in the magnitude of measured discharge?

```
EnoDischarge.early.subsample <- sample_n(EnoDischarge.early, 5000)
EnoDischarge.late.subsample <- sample_n(EnoDischarge.late, 5000)

shapiro.test(EnoDischarge.early.subsample$discharge.mean)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  EnoDischarge.early.subsample$discharge.mean
## W = 0.3301, p-value < 2.2e-16
```

```
shapiro.test(EnoDischarge.late.subsample$discharge.mean)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  EnoDischarge.late.subsample$discharge.mean
## W = 0.27196, p-value < 2.2e-16
```

```
var.test(EnoDischarge.early$discharge.mean, EnoDischarge.late$discharge.mean)
```

```
##
##  F test to compare two variances
##
## data:  EnoDischarge.early$discharge.mean and EnoDischarge.late$discharge.mean
## F = 0.80003, num df = 16076, denom df = 12504, p-value < 2.2e-16
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.7739912 0.8268842
## sample estimates:
## ratio of variances
##          0.8000284
```

```
wilcox.test(EnoDischarge.early$discharge.mean, EnoDischarge.late$discharge.mean)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  EnoDischarge.early$discharge.mean and EnoDischarge.late$discharge.mean
## W = 1.17e+08, p-value < 2.2e-16
## alternative hypothesis: true location shift is not equal to 0
```

How might you interpret the results of this test, and how might you represent them graphically?

Monotonic trend analysis

In general, detecting a monotonic trend requires a long sequence of data with few gaps. If we are working with monthly data, a time series of at least five years is recommended. Gaps can be accounted for, but a gap that makes up more than 1/3 of the sampling period is generally considered the threshold for considering a gap to be too long (a step trend analysis might be better in this situation).

Adjusting the data may be necessary to fulfill the assumptions of a trend test. These adjustments include **aggregation**, **subsampling**, and **interpolation**. What do each of these mean, and why might we want to use them?

aggregation:

subsampling:

interpolation:

Common interpolation methods:

- **Piecewise constant**: also known as a "nearest neighbor" approach. Any missing data are assumed to be equal to the measurement made nearest to that date (could be earlier or later).
- **Linear**: could be thought of as a "connect the dots" approach. Any missing data are assumed to fall between the previous and next measurement, with a straight line drawn between the known points determining the values of the interpolated data on any given date.
- **Spline**: similar to a linear interpolation except that a quadratic function is used to interpolate rather than drawing a straight line.

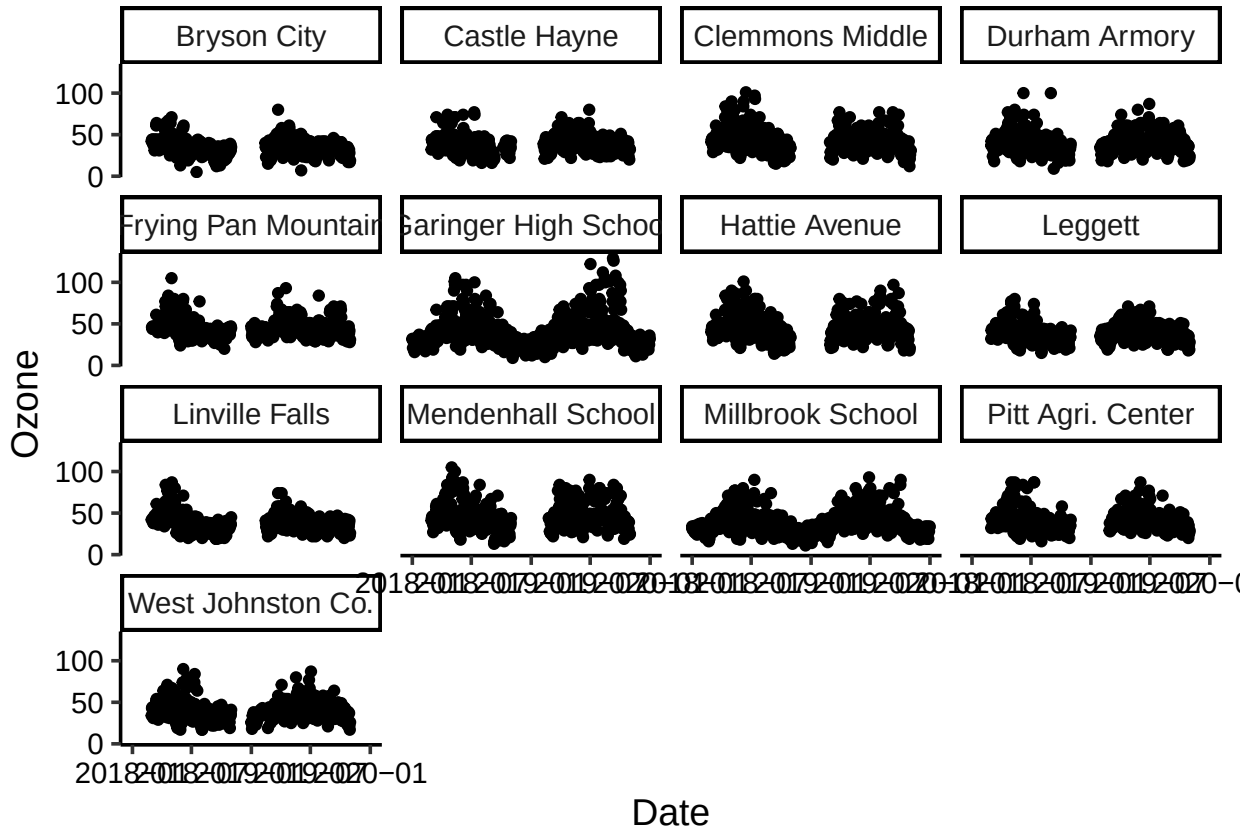
Example: interpolation

The Eno River discharge data doesn't have any short periods of missing data, so interpolation would not be a good choice for that dataset. We will illustrate a linear interpolation of the NC Air quality dataset below.

In this case, several sites have a lot of missing data, and several sites monitor most days with few missing data points.

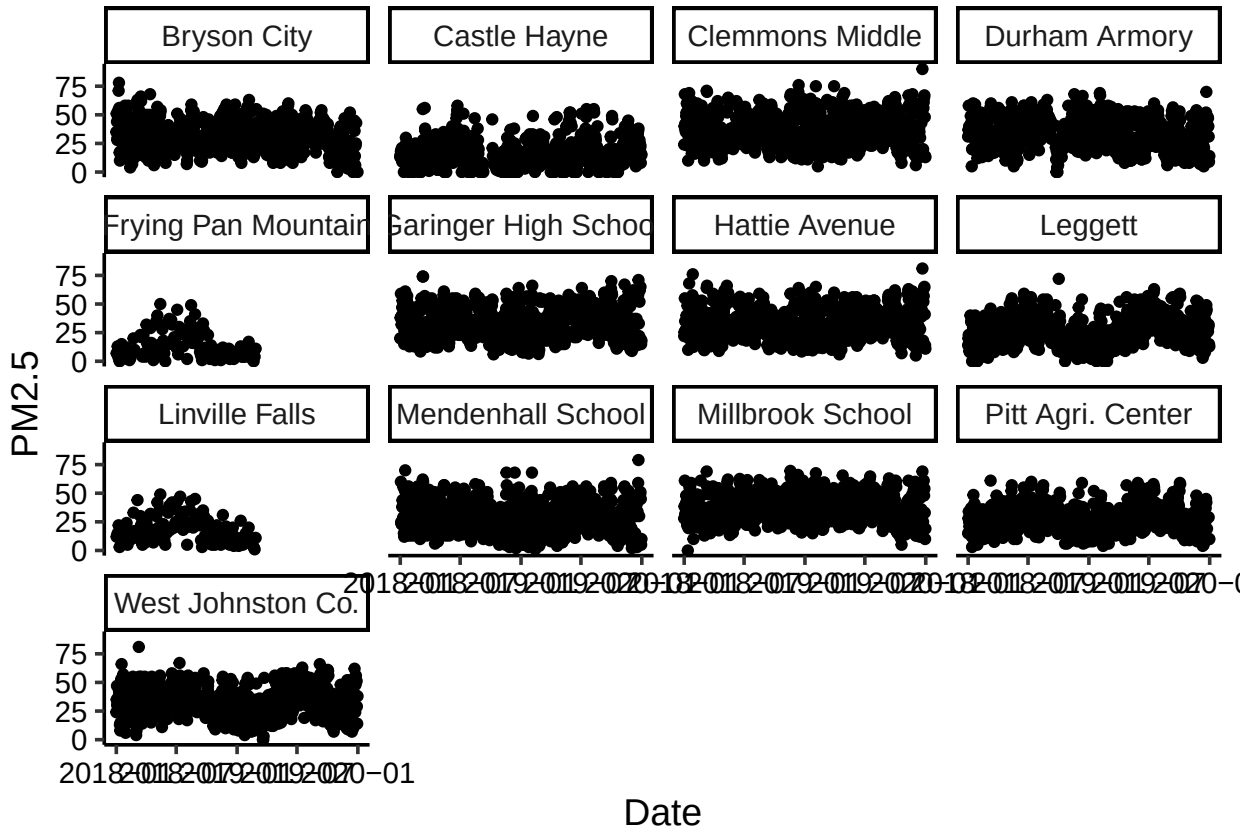
```
NCOzone <-
ggplot(NCAir, aes(x = Date, y = Ozone)) +
  geom_point() +
  facet_wrap(vars(Site.Name))
print(NCOzone)
```

```
## Warning: Removed 2146 rows containing missing values (geom_point).
```




```
NCPM2.5 <-
ggplot(NCAir, aes(x = Date, y = PM2.5)) +
  geom_point() +
  facet_wrap(vars(Site.Name))
print(NCPM2.5)
```

```
## Warning: Removed 1054 rows containing missing values (geom_point).
```

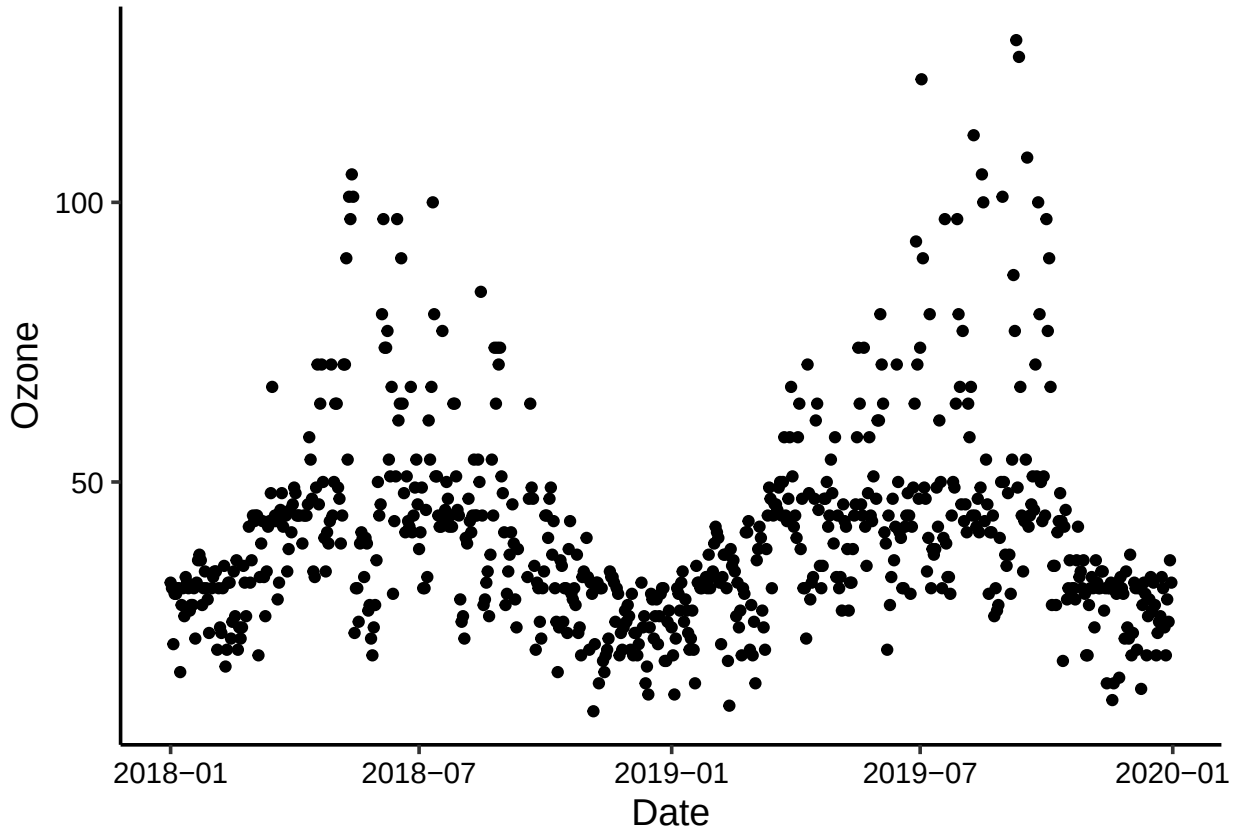



```
summary(NCAir$Site.Name)
```

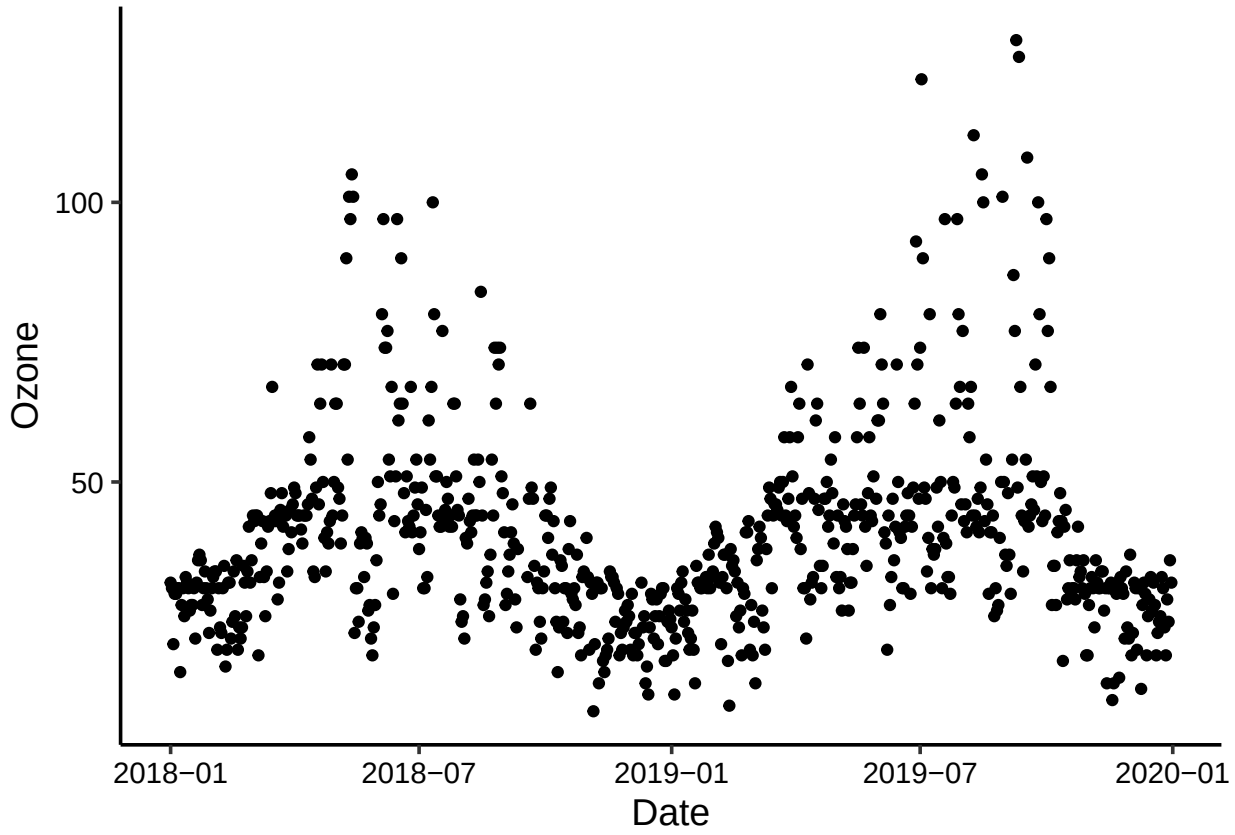
```
##          Bryson City         Castle Hayne      Clemmons Middle
##                  724                  677                  730
##        Durham Armory  Frying Pan Mountain Garinger High School
##                  722                  556                  722
##        Hattie Avenue              Leggett       Linville Falls
##                  730                  717                  545
##    Mendenhall School     Millbrook School    Pitt Agri. Center
##                  716                  724                  697
##    West Johnston Co.
##                  716
```

```
NCAir.Garinger <- NCAir %>%
  filter(Site.Name == "Garinger High School")

GaringerOzone <-
ggplot(NCAir.Garinger, aes(x = Date, y = Ozone)) +
  geom_point()
print(GaringerOzone)
```

```
## Warning: Removed 5 rows containing missing values (geom_point).
```




```
# na.approx function fills in NAs with a linear interpolation
# Spline interpolation can also be specified as an alternative
# Piecewise constant interpolation can be done with na.aggregate
NCAir.Garinger$Ozone <- na.approx(NCAir.Garinger$Ozone)
NCAir.Garinger$PM2.5 <- na.approx(NCAir.Garinger$PM2.5)

GaringerOzone.interpolated <-
ggplot(NCAir.Garinger, aes(x = Date, y = Ozone)) +
  geom_point()
print(GaringerOzone.interpolated)
```



Monotonic trend analysis, continued

Specific tests for monotonic trend analysis are listed below, with assumptions and tips:

- **linear regression**: no seasonality, fits the assumptions of a parametric test. Function: `lm`
- **Mann-Kendall**: no seasonality, non-parametric, no temporal autocorrelation, missing data allowed. Function: `mk.test` (package: trend)
- **modified Mann-Kendall**: no seasonality, non-parametric, accounts for temporal autocorrelation, missing data allowed. Function: `mmky` and `mmkh` (package: modifiedmk)
- **Seasonal Mann-Kendall**: seasonality, non-parametric, no temporal autocorelation, identical distribution. Function: `smk.test` (package: trend)

The packages trend, Kendall, and modifiedmk also include other modifications to monotonic trend tests. Look into the documentation for these packages if you are applying a special case.

If covariates (another predictor variable) are included in the dataset, additional tests are recommended. A great resource for trend testing for water quality monitoring, which includes guidance on these cases, has been prepared by the Environmental Protection Agency: https://www.epa.gov/sites/production/files/2016-05/documents/tech_notes_6_dec2013_trend.pdf. This would likely be useful for other types of environmental data too.

Example: monotonic trend analysis

Remember that we noticed in the decomposition that the Eno River discharge data has a seasonal cycle (despite high random variability). We might be interested in knowing how (if) discharge has changed over

the course of measurement while incorporating the seasonal component. In this case, we will use a Seasonal Mann-Kendall test to figure out whether a monotonic trend exists. We will use the late dataset again.

The Seasonal Mann-Kendall assumes no temporal autocorrelation, but we know that daily data is prone to temporal autocorrelation. In this case, we may want to collapse our data down into monthly data so that we can (1) reduce temporal autocorrelation and (2) break down the potential seasonal trend into more interpretable components.

We will calculate the mean monthly discharge for this dataset, rather than calculating the total monthly discharge or subsampling a given day in each month. Why did we make this decision?

```
EnoDischarge.late.monthly <- EnoDischarge.late %>%
  mutate(Year = year(datetime),
         Month = month(datetime)) %>%
  group_by(Year, Month) %>%
  summarise(Discharge = mean(discharge.mean))

EnoDischarge.late.monthly$Date <- as.Date(paste(EnoDischarge.late.monthly$Year,
                                                EnoDischarge.late.monthly$Month,
                                                1, sep="-"),
                                          format = "%Y-%m-%d")

# Generate time series (smk.test needs ts, not data.frame)
EnoDischarge.late.monthly.ts <- ts(EnoDischarge.late.monthly$Discharge, frequency = 12,
                        start = c(1985, 10, 1), end = c(2019, 12, 1))
# Run SMK test
EnoDischarge.late.trend <- smk.test(EnoDischarge.late.monthly.ts)

# Inspect results
EnoDischarge.late.trend
```

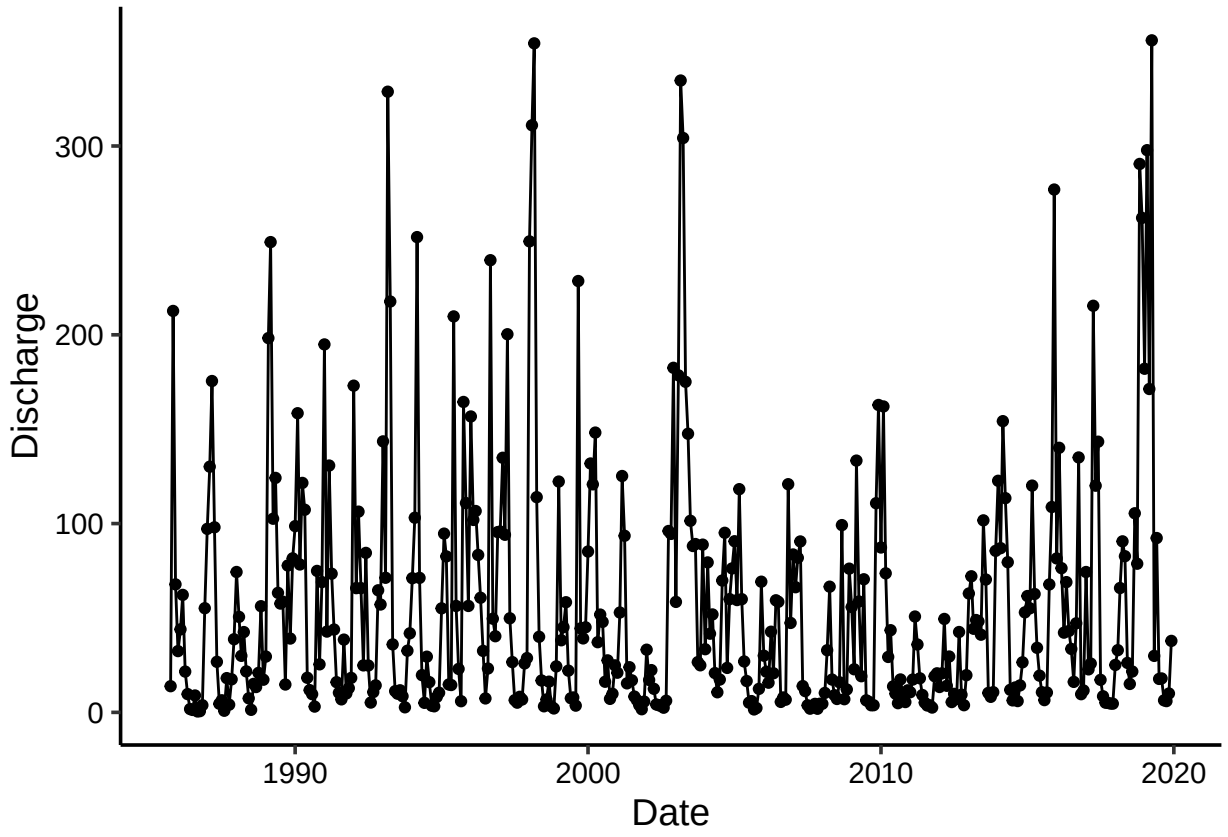
```
##
##  Seasonal Mann-Kendall trend test (Hirsch-Slack test)
##
## data:  EnoDischarge.late.monthly.ts
## z = -0.13967, p-value = 0.8889
## alternative hypothesis: true S is not equal to 0
## sample estimates:
##     S  varS
##   -34 55828
```

```
summary(EnoDischarge.late.trend)
```

```
##
##  Seasonal Mann-Kendall trend test (Hirsch-Slack test)
##
## data: EnoDischarge.late.monthly.ts
## alternative hypothesis: two.sided
##
## Statistics for individual seasons
##
## H0
##                       S   varS    tau      z Pr(>|z|)
## Season 1:   S = 0  -91 4550.3 -0.162 -1.334  0.18214
## Season 2:   S = 0  -69 4550.3 -0.123 -1.008  0.31342
## Season 3:   S = 0  -85 4550.3 -0.152 -1.245  0.21304
```

```
## Season 4:   S = 0  -21 4550.3 -0.037 -0.296  0.76686
## Season 5:   S = 0   59 4550.3  0.105  0.860  0.38989
## Season 6:   S = 0  101 4550.3  0.180  1.482  0.13822
## Season 7:   S = 0   71 4550.3  0.127  1.038  0.29940
## Season 8:   S = 0   29 4550.3  0.052  0.415  0.67808
## Season 9:   S = 0   21 4550.3  0.037  0.296  0.76686
## Season 10:   S = 0  15 4958.3  0.025  0.199  0.84240
## Season 11:   S = 0 -63 4958.3 -0.106 -0.880  0.37859
## Season 12:   S = 0  -1 4958.3 -0.002  0.000  1.00000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
EnoDischarge.monthly <-
ggplot(EnoDischarge.late.monthly, aes(x = Date, y = Discharge)) +
  geom_point() +
  geom_line()
print(EnoDischarge.monthly)
```



What would we conclude based on these findings?

If a significant trend was present, we could compute a **Sen's Slope** to quantify that trend (`sens.slope` function in the trend package).

Forecasting with Autoregressive and Moving Average Models (ARMA)

We might be interested in characterizing a time series in order to understand what happened in the past and to effectively forecast into the future. Two common models that can approximate time series are **autoregressive** and **moving average** models. To classify these models, we use the **ACF (autocorrelation function)** and the **PACF (partial autocorrelation function)**, which correspond to the autocorrelation of a series and the correlation of the residuals, respectively.

Autoregressive models operate under the framework that a given measurements is correlated with previous measurements. For example, an AR1 formulation dictates that a measurement is dependent on the previous measurement, and the value can be predicted by quantifying the lag.

Moving average models operate under the framework that the covariance between a measurement and the previous measurement is zero. While AR models use past forecast *values* to predict future values, MA models use past forecast *errors* to predict future values.

Here are some great resources for examining ACF and PACF lags under different formulations of AR and MA models. https://nwfsc-timeseries.github.io/atsa-labs/sec-tslab-autoregressive-ar-models.html https://nwfsc-timeseries.github.io/atsa-labs/sec-tslab-moving-average-ma-models.html

ARMA models require stationary data. This means that there is no monotonic trend over time and there is also equal variance and covariance across the time series. The function `adf.test` will determine whether our data are stationary. The null hypothesis is that the data are not stationary, so we infer that the data are stationary if the p-value is < 0.05.

While some processes might be easy to identify, it is often complicated to predict the order of AR and MA processes when the operate in the same dataset. To get around this issue, it is often necessary to run multiple potential formulations of the model and see which one results in the most parsimonious fit using AIC. The function `auto.arima` does this automatically.