7: Time Series

Environmental Data Analytics | Adapted by John Fay and Luana Lima | Developed by Kateri Salk

Fall 2022

Objectives

1. Discuss the purpose and application of time series analysis for environmental data
2. Explore the components of times series: trend, seasonal, random
3. Learn how to perform stationarity test

Set up

Today we will work with two datasets. The USGS dataset on discharge at the Eno River and a new dataset we haven't explored yet on wind speed. The data file is available at "./Data/Raw/Wind_Speed_PortArthurTX.csv". It contains average wind speed in monthly time steps (elevation = 5 meters). The data is available from NOAA National Centers for Environmental Information (NCEI) [here][https://www.ncdc.noaa.gov/cdo-web/datasets#GSOM].

```r
library(tidyverse)
library(lubridate)
#install.packages("trend")
library(trend)
#install.packages("zoo")
library(zoo)
#install.packages("Kendall")
library(Kendall)
#install.packages("tseries")
library(tseries)
library(formatR)

knitr::opts_chunk$set(tidy.opts=list(width.cutoff=45), tidy=TRUE)

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

#Read Eno river data
EnoDischarge <- read.csv("./Data/Processed/USGS_Site02085000_Flow_Processed.csv",
                         stringsAsFactors = TRUE)
EnoDischarge$datetime <- as.Date(EnoDischarge$datetime, format = "%Y-%m-%d")

#Read wind speed data
wind_data <- read.csv(file="./Data/Raw/Wind_Speed_PortArthurTX.csv",header=TRUE,
                      stringsAsFactors = TRUE)
wind_data$DATE <- ym(wind_data$DATE)
```

Time Series Analysis overview

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:

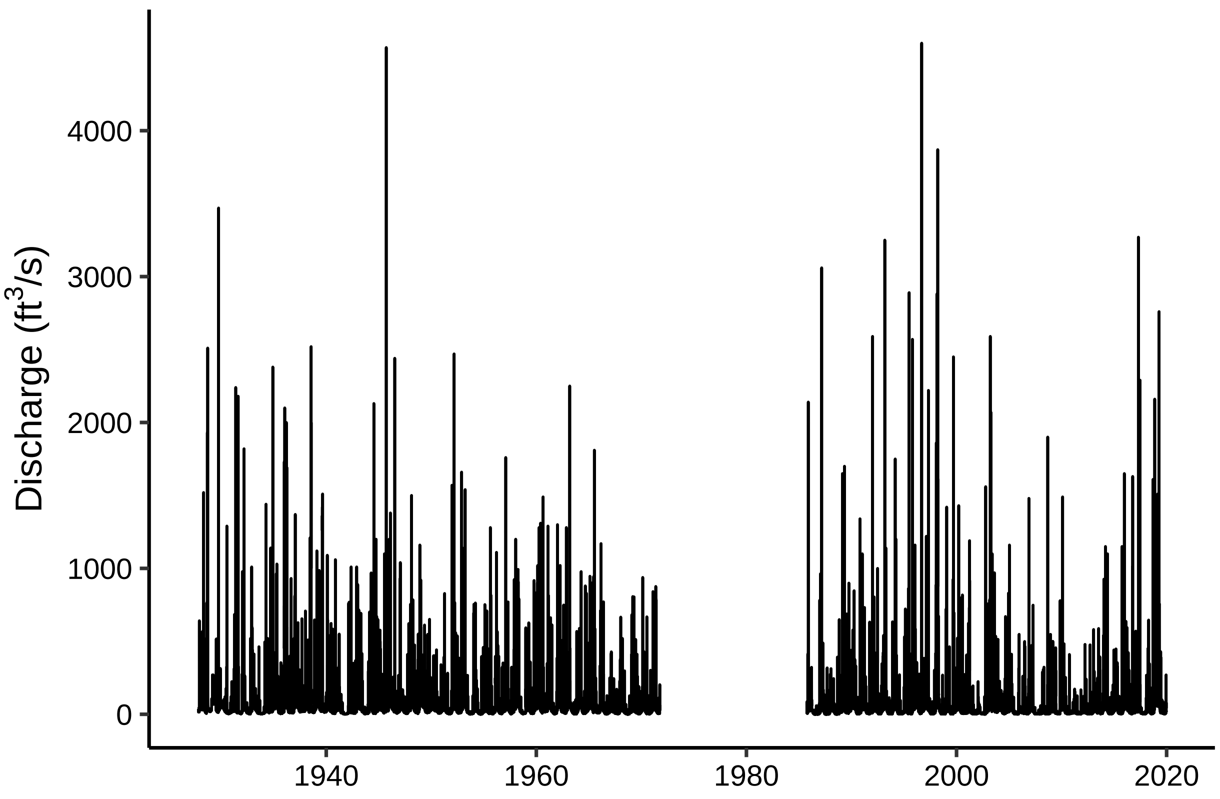
- **Autocorrelation**: Data points are not independent from one another (i.e., the measurement at a given time point is dependent on previous time point(s))
- **Data gaps**: Data are not collected at regular intervals, necessitating *interpolation* between measurements.
- **Seasonality**: seasonal patterns 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

Handling data gaps and missing data. Example: 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.

```r
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.

```r
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.

Example: Eno discharge

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.

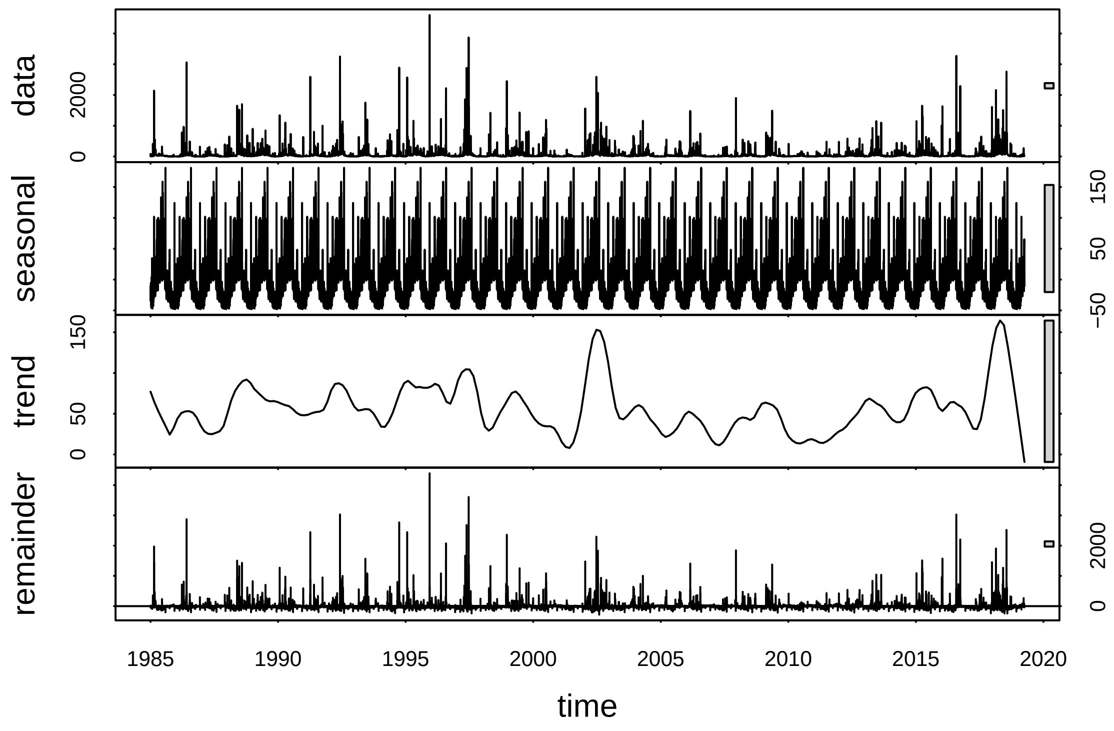
We first need to turn the discharge data into a time series object in R. This is done using the `ts` function. Notice we can only specify one column of data and need to specify the period at which the data are sampled. The resulting time series object cannot be viewed like a regular data frame.

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.

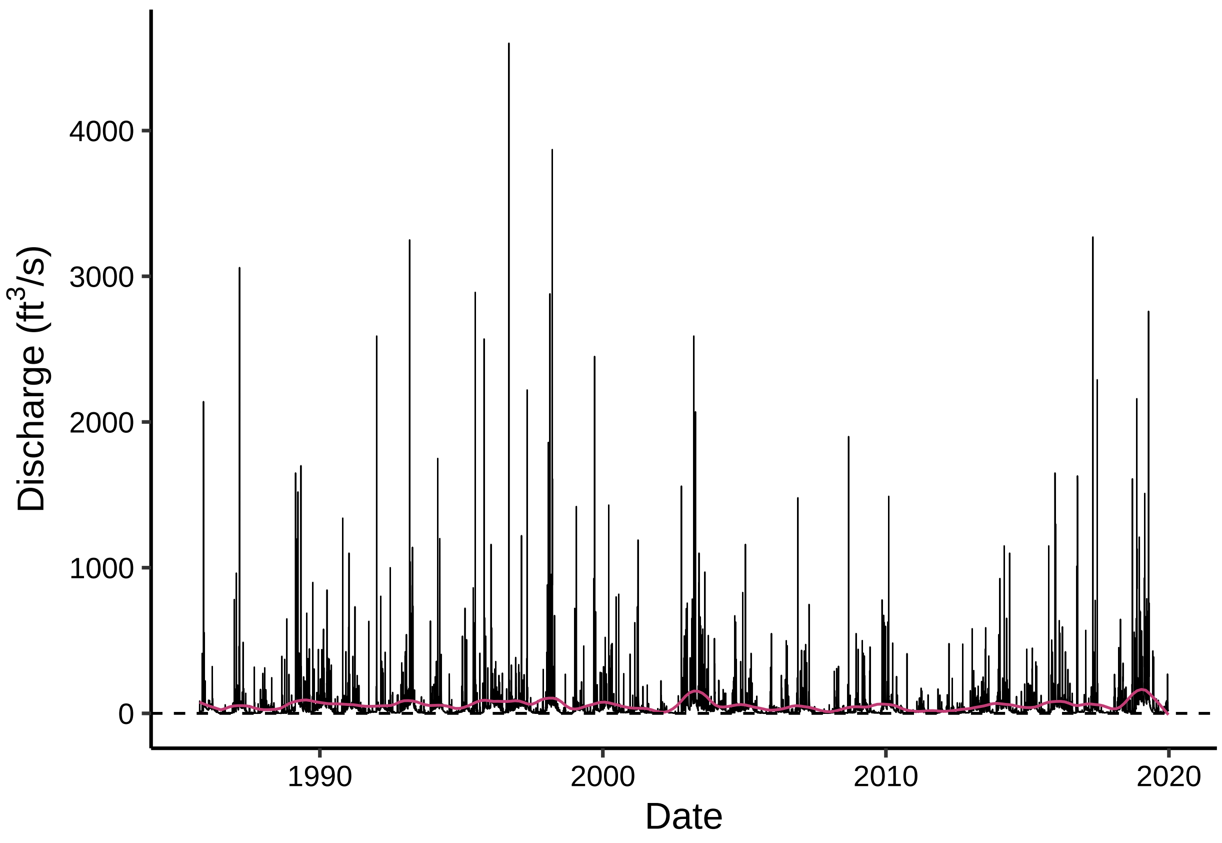
```
EnoDischarge.late_ts <- ts(EnoDischarge.late$discharge.mean,
    start = c(1985, 1), frequency = 365)
```

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.

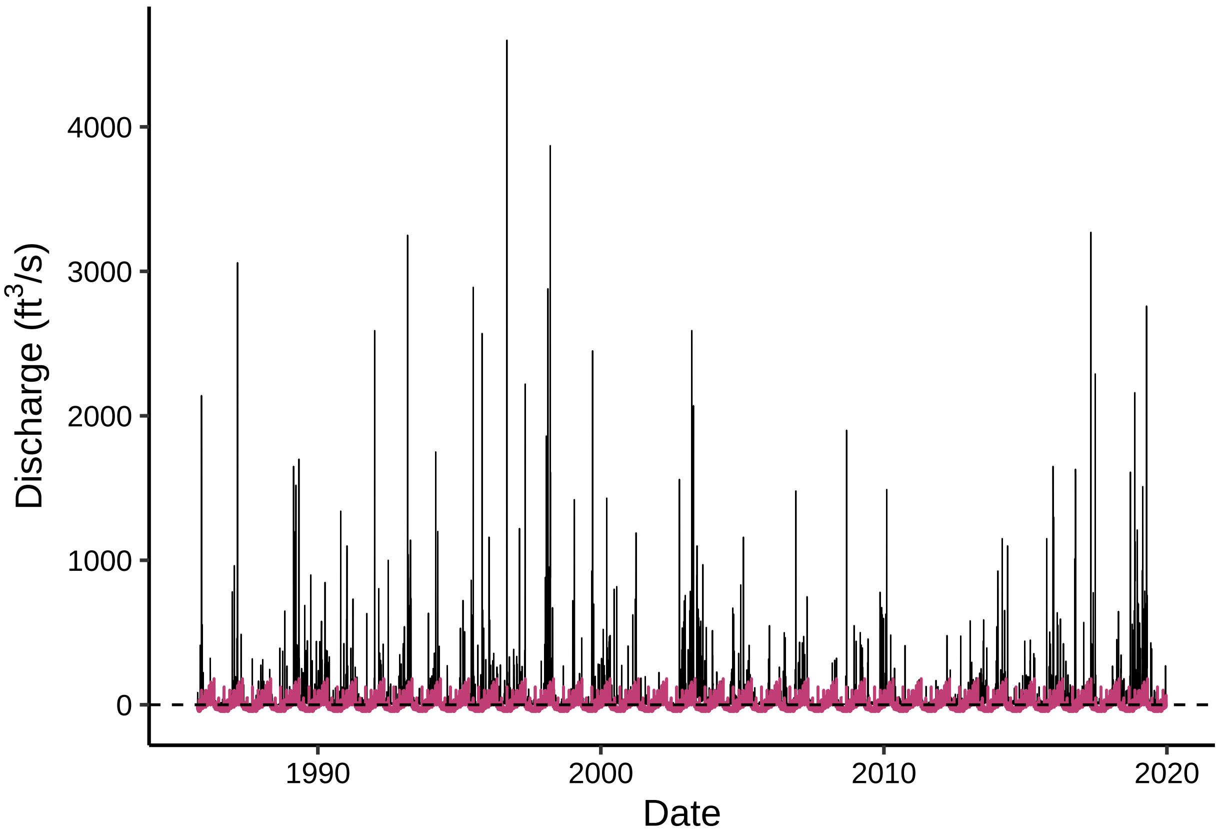
```
`?`(stl)
# Generate the decomposition
EnoDischarge.late_Decomposed <- stl(EnoDischarge.late_ts,
    s.window = "periodic")

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



```
# 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)"))
```



```
# 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/deterministic** or **stochastic**. Monotonic trends are a gradual shift over time that is consistent in direction, for example in response to land use change.

A third type of trend we haven't talked about is the **step** trend, also knows as a level shift. Step trends are a distinct shift at a given time point, for example in response to a policy being enacted.

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. A common method to replace missing values is **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 using the wind speed dataset.

```
head(wind_data)
```

```
##       STATION                                      NAME       DATE AWND
## 1 USW00012917 PORT ARTHUR SE TX REGIONAL AIRPORT, TX US 1984-01-01  4.3
## 2 USW00012917 PORT ARTHUR SE TX REGIONAL AIRPORT, TX US 1984-02-01  5.0
## 3 USW00012917 PORT ARTHUR SE TX REGIONAL AIRPORT, TX US 1984-03-01  5.1
## 4 USW00012917 PORT ARTHUR SE TX REGIONAL AIRPORT, TX US 1984-04-01  5.5
## 5 USW00012917 PORT ARTHUR SE TX REGIONAL AIRPORT, TX US 1984-05-01  4.5
## 6 USW00012917 PORT ARTHUR SE TX REGIONAL AIRPORT, TX US 1984-06-01  3.9
```

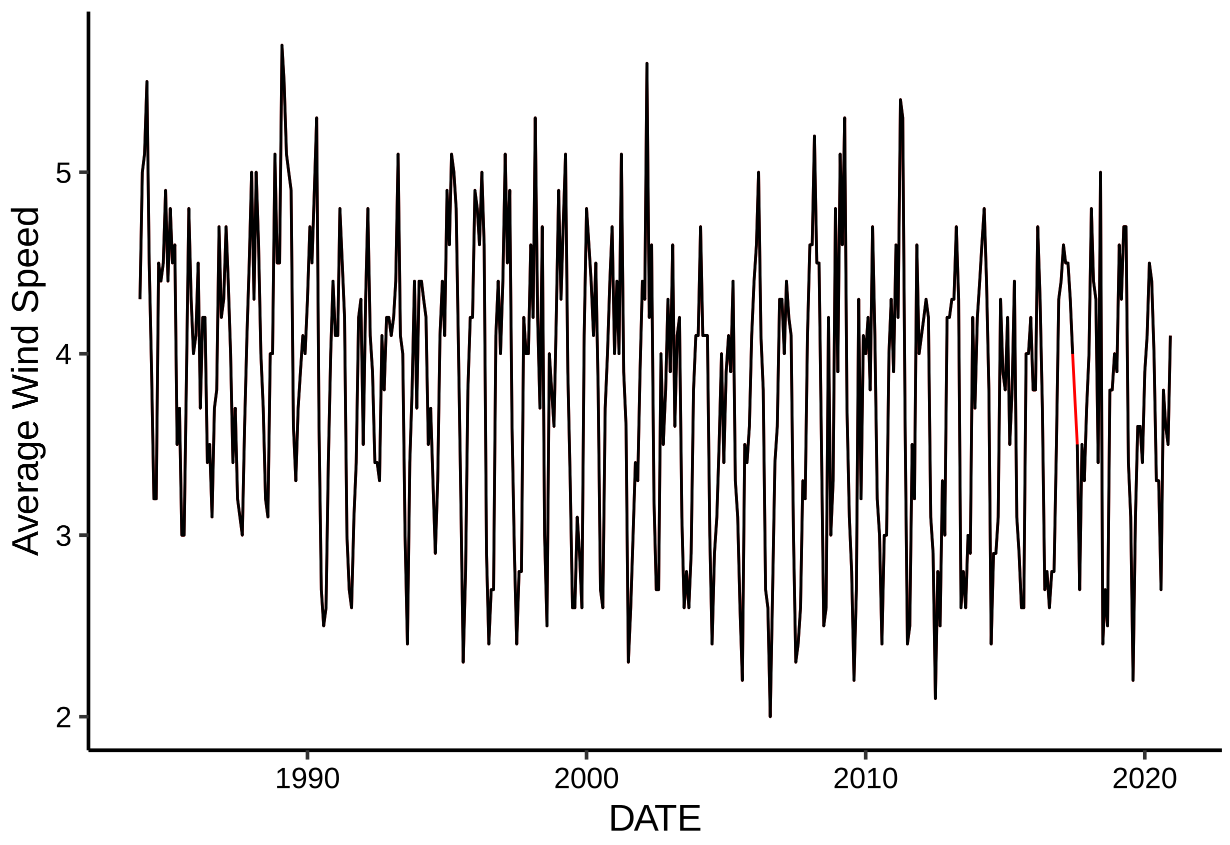
```
summary(wind_data$AWND)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   2.000   3.200   4.000   3.819   4.400   5.700       1
```

```
# Adding new column with no missing obs,
# just for illustration purpose In real
# applications you will simply replace NAs
wind_data_clean <- wind_data %>%
    mutate(AWND.clean = zoo::na.approx(AWND))

summary(wind_data_clean$AWND.clean)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   3.200   4.000   3.819   4.400   5.700
```

```
# Note the NA is gone

ggplot(wind_data_clean) + geom_line(aes(x = DATE,
    y = AWND.clean), color = "red") + geom_line(aes(x = DATE,
    y = AWND), color = "black") + ylab("Average Wind Speed")
```



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, missing data allowed. Function: `MannKendall()` (package: Kendall)
- **Seasonal Mann-Kendall**: seasonality, non-parametric `SeasonalMannKendall` (package: Kendall)
- **Spearman Rho**: no seasonality, non-parametric, missing data allowed. Function: `cor.test(method="spearman")` (package: stats)

Specific test for stochastic trend analysis:

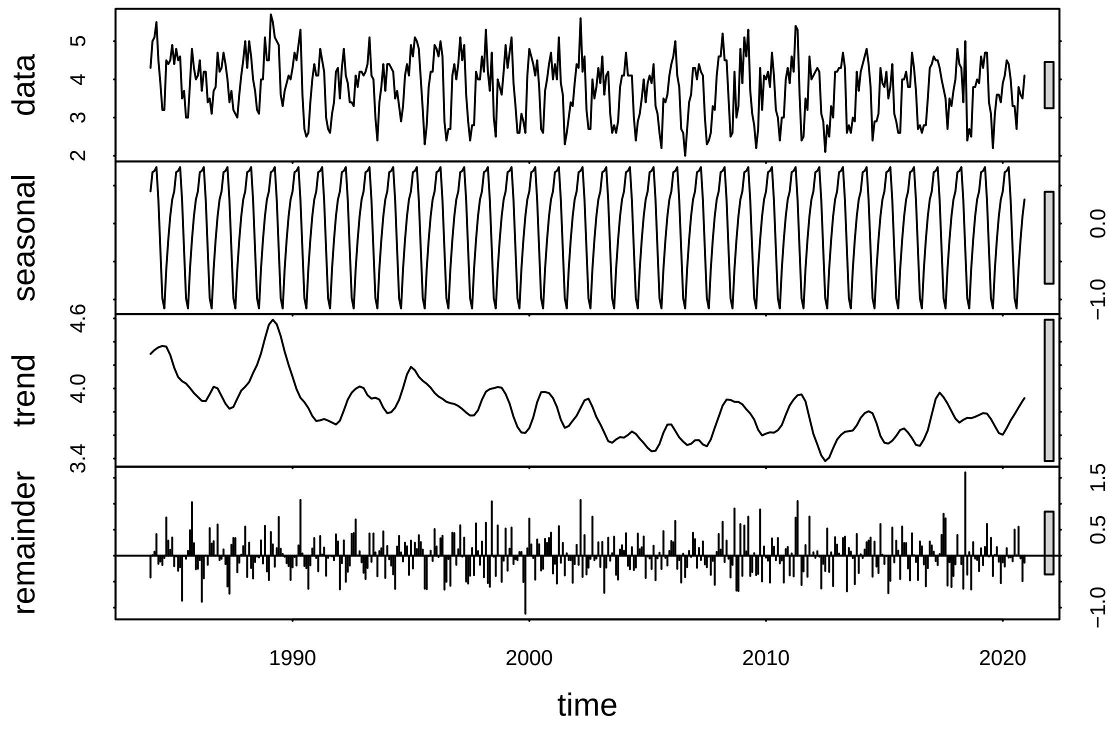
- **Augmented Dickey Fuller**: no seasonality, non-parametric, missing data not allowed. Function: `adf.test()` (package: tseries)

Example: monotonic trend analysis Let's refer to our wind speed data. We already performed interpolation, but we still need to create our time series object and decompose the series to find out which stationarity test we can apply.

Note that wind speed has a seasonal cycle. We might be interested in knowing how (if) speed 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.

```
# Generate time series (trend test needs ts,
# not data.frame)
f_month <- month(first(wind_data_clean$DATE))
f_year <- year(first(wind_data_clean$DATE))
```

```
wind_data_ts <- ts(wind_data_clean$AWND.clean,
    start = c(f_year, f_month), frequency = 12)

# decompose
wind_data_decomp <- stl(wind_data_ts, s.window = "periodic")
plot(wind_data_decomp)
```



```
# Run SMK test
wind_data_trend1 <- Kendall::SeasonalMannKendall(wind_data_ts)

# Inspect results
wind_data_trend1
```

```
## tau = -0.229, 2-sided pvalue =1.7751e-11
```

```
summary(wind_data_trend1)
```

```
## Score =  -1770 , Var(Score) = 69305.33
## denominator =  7725.182
## tau = -0.229, 2-sided pvalue =1.7751e-11
```

```
wind_data_trend2 <- trend::smk.test(wind_data_ts)
# Inspect results
wind_data_trend2
```

```
##
##  Seasonal Mann-Kendall trend test (Hirsch-Slack test)
##
```

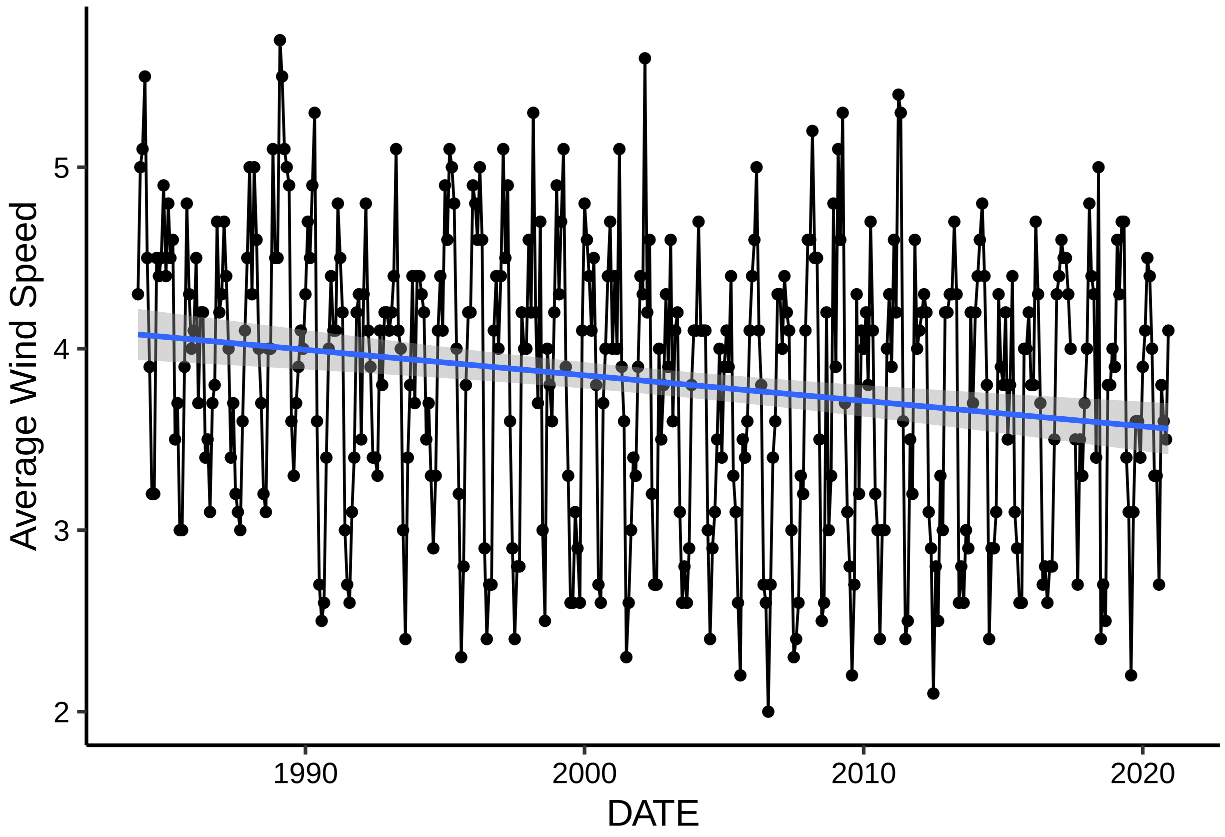
```
## data:  wind_data_ts
## z = -6.7196, p-value = 1.822e-11
## alternative hypothesis: true S is not equal to 0
## sample estimates:
##        S     varS
## -1770.00 69305.33
```

```
summary(wind_data_trend2)
```

```
##
##  Seasonal Mann-Kendall trend test (Hirsch-Slack test)
##
## data: wind_data_ts
## alternative hypothesis: two.sided
##
## Statistics for individual seasons
##
## H0
##                         S   varS    tau      z   Pr(>|z|)
## Season 1:   S = 0  -203 5753.7 -0.318 -2.663 0.00774363  **
## Season 2:   S = 0  -129 5731.7 -0.203 -1.691 0.09089192   .
## Season 3:   S = 0  -125 5796.3 -0.193 -1.629 0.10337344
## Season 4:   S = 0   -42 5780.0 -0.065 -0.539 0.58968885
## Season 5:   S = 0    -9 5815.7 -0.014 -0.105 0.91645233
## Season 6:   S = 0  -139 5806.3 -0.214 -1.811 0.07013462   .
## Season 7:   S = 0  -148 5777.3 -0.230 -1.934 0.05311469   .
## Season 8:   S = 0  -147 5749.0 -0.230 -1.926 0.05415953   .
## Season 9:   S = 0  -199 5798.3 -0.307 -2.600 0.00931583  **
## Season 10:   S = 0 -216 5783.3 -0.335 -2.827 0.00469638  **
## Season 11:   S = 0 -258 5792.0 -0.399 -3.377 0.00073306 ***
## Season 12:   S = 0 -155 5721.7 -0.245 -2.036 0.04175898   *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Visualization
wind_data_plot <- ggplot(wind_data, aes(x = DATE,
    y = AWND)) + geom_point() + geom_line() +
    ylab("Average Wind Speed") + geom_smooth(method = lm)
print(wind_data_plot)
```

```
## `geom_smooth()` using formula 'y ~ x'

## Warning: Removed 1 rows containing non-finite values (stat_smooth).

## Warning: Removed 1 rows containing missing values (geom_point).
```



What would we conclude based on these findings?

Answer:

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. 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.