10: Time Series Analysis

Water Data Analytics | Kateri Salk

Spring 2022

Lesson Objectives

1. Choose appropriate time series analyses for trend detection and forecasting
2. Discuss the influence of seasonality on time series analysis
3. Perform foecasting with ARMA modeling
4. Interpret and communicate results of time series analyses

Session Set Up

```
getwd()
```

```
## [1] "/Users/katerisalk/Box Sync/Courses/Water Data Analytics/Lessons"
```

```
library(tidyverse)
library(lubridate)
library(dataRetrieval)
library(cowplot)
#install.packages("trend")
library(trend)
#install.packages("forecast")
library(forecast)
#install.packages(tseries)
library(tseries)

theme_set(theme_classic())
```

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 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**: see a review of the pros and cons of various methods here
- **t-test (paired or unpaired)**
- **Kruskal-Wallis test**: non-parametric version of t-test
- **ANCOVA**, analysis of covariance

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 ten 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:

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.

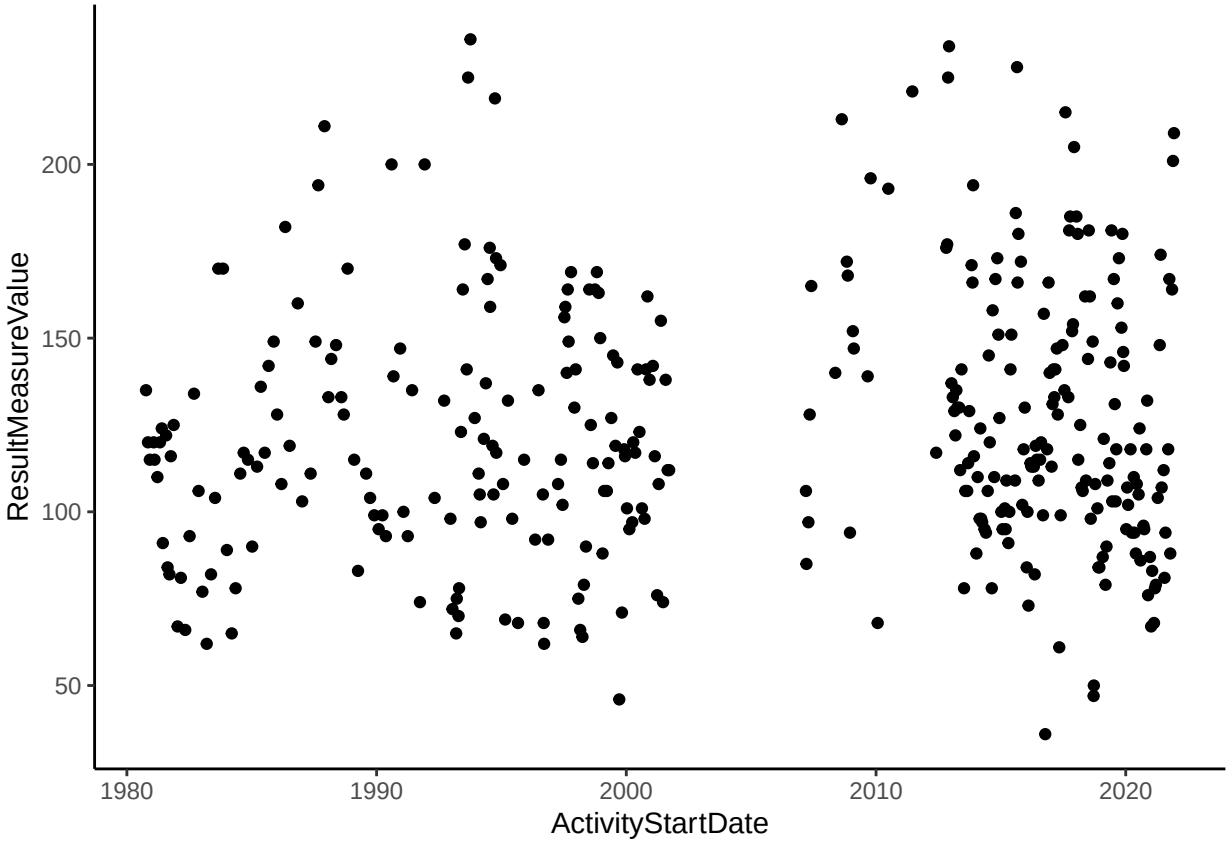
If covariates (another predictor variable) are included in the dataset, additional tests are recommended. A great resource for trend testing for water quality monitoring, which includes guidance on these cases, has been prepared by the Environmental Protection Agency: https://www.epa.gov/sites/production/files/2016-05/documents/tech_notes_6_dec2013_trend.pdf

Trend test example: Discharge and Conductivity in the Neuse River

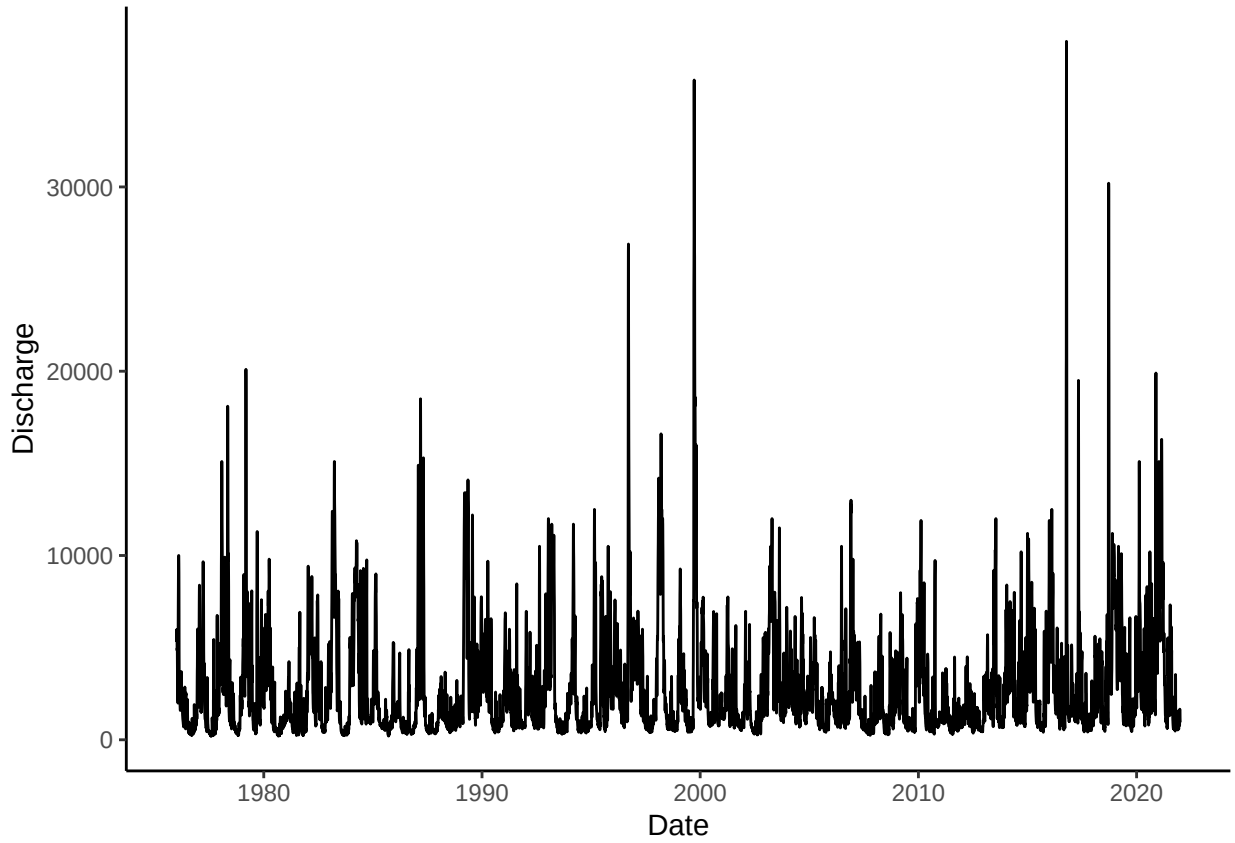
Last week, we examined time series of discharge and conductivity in the Neuse River. We decomposed these time series and found that both had distinct seasonality. Therefore, we will proceed with a **Seasonal Mann-Kendall** test. Recall that for the conductivity data, we also performed a linear interpolation to fill in missing months.

```
NeuseCond <- readWQPqw(siteNumbers = "USGS-02089500", # Neuse River at Kinston, NC
                       parameterCd = "90095",  # Specific conductance, uS/cm
                       startDate = "1976-01-01",
                       endDate = "2021-12-31")
NeuseFlow <- readNWISdv(siteNumbers = "02089500",
                       parameterCd = "00060", # discharge (cfs)
                       startDate = "1976-01-01",
                       endDate = "2021-12-31")
names(NeuseFlow)[4:5] <- c("Discharge", "Approval.Code")


# What do these data look like?
ggplot(NeuseCond, aes(x = ActivityStartDate, y = ResultMeasureValue)) +
  geom_point()
```




```
ggplot(NeuseFlow, aes(x = Date, y = Discharge)) +
  geom_line()
```




```
# create a data frame of months
Months <- data.frame(Date_monthrounded = seq.Date(from = as.Date("1980-10-01"), to = as.Date("2021-12-0

NeuseCond_processed <- NeuseCond %>%
  select(Date = ActivityStartDate,
         Conductivity = ResultMeasureValue) %>%
  mutate(Year = year(Date),
         Month = month(Date),
         Date_monthrounded = floor_date(Date, "month")) %>%
  arrange(Date)

NeuseCond_monthly <- left_join(Months, NeuseCond_processed)
```

```
## Joining, by = "Date_monthrounded"
```

```
# Generate monthly values from October 1980 to December 2021
linearinterpolation <- as.data.frame(approx(NeuseCond_monthly$Conductivity, n = 566, method = "linear"))
NeuseCond_monthly$Conductivity <- linearinterpolation$y

# generate total monthly discharge
NeuseFlow_monthly <- NeuseFlow %>%
  mutate(Year = year(Date),
         Month = month(Date)) %>%
  group_by(Year, Month) %>%
  summarise(Discharge_acftmo = sum(Discharge)*1.98347)
```

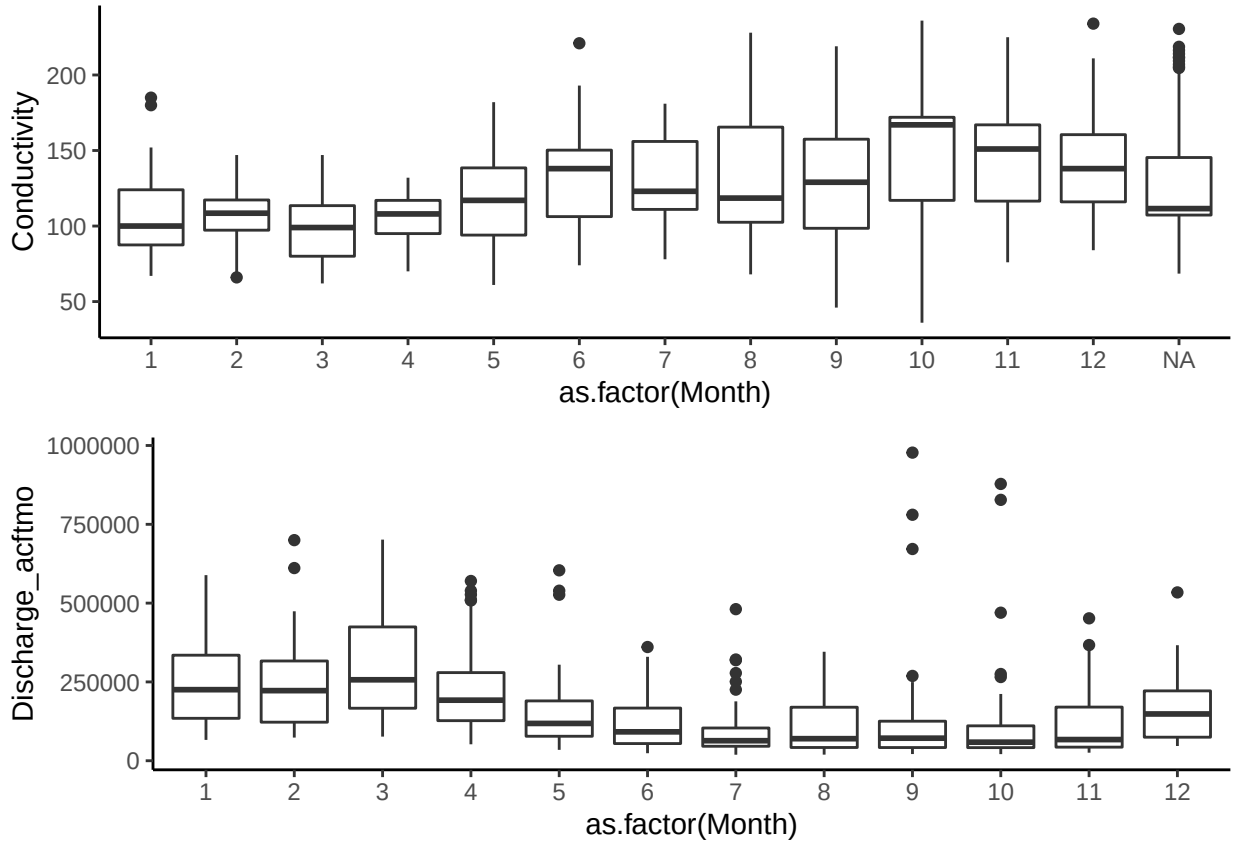
```
## `summarise()` has grouped output by 'Year'. You can override using the `.groups` argument.
```

Let's visualize the seasonality of these two datasets. Add some editing to improve the graphs.

```
Condplot <-
ggplot(NeuseCond_monthly, aes(x = as.factor(Month), y = Conductivity)) +
  geom_boxplot()

Flowplot <-
ggplot(NeuseFlow_monthly, aes(x = as.factor(Month), y = Discharge_acftmo)) +
  geom_boxplot()

plot_grid(Condplot, Flowplot, ncol = 1)
```



Now, let's prepare our datasets for seasonal Mann-Kendall analysis. This test requires a time series rather than a data frame.

```
NeuseCond_timeseries <- ts(NeuseCond_monthly$Conductivity, frequency = 12,
                           start = c(1980, 10, 1), end = c(2021, 12, 1))
# Run SMK test
NeuseCond_trend <- smk.test(NeuseCond_timeseries)

# Inspect results
NeuseCond_trend
```

```
##
##  Seasonal Mann-Kendall trend test (Hirsch-Slack test)
##
## data:  NeuseCond_timeseries
## z = 4.3125, p-value = 1.614e-05
```

```
## alternative hypothesis: true S is not equal to 0
## sample estimates:
##     S  varS
##  1343 96837
```

```
summary(NeuseCond_trend)
```

```
##
##  Seasonal Mann-Kendall trend test (Hirsch-Slack test)
##
## data: NeuseCond_timeseries
## alternative hypothesis: two.sided
##
## Statistics for individual seasons
##
## H0
##                       S   varS    tau      z  Pr(>|z|)
## Season 1:   S = 0  224 7926.7  0.273  2.505 0.0122548  *
## Season 2:   S = 0  247 7923.7  0.302  2.764 0.0057171 **
## Season 3:   S = 0  243 7917.0  0.298  2.720 0.0065324 **
## Season 4:   S = 0  137 7923.7  0.167  1.528 0.1265541
## Season 5:   S = 0  185 7923.7  0.226  2.067 0.0387278  *
## Season 6:   S = 0  192 7922.7  0.235  2.146 0.0318856  *
## Season 7:   S = 0  109 7919.7  0.133  1.214 0.2249061
## Season 8:   S = 0   72 7924.7  0.088  0.798 0.4251211
## Season 9:   S = 0   43 7925.7  0.052  0.472 0.6370901
## Season 10:   S = 0 -74 8508.7 -0.086 -0.791 0.4287148
## Season 11:   S = 0 -52 8511.3 -0.061 -0.553 0.5803975
## Season 12:   S = 0  17 8509.7  0.020  0.173 0.8623011
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

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).

Exercise: run the same test for discharge.

```
NeuseFlow_timeseries <- ts(NeuseFlow_monthly$Discharge_acftmo, frequency = 12,
                           start = c(1976, 1, 1), end = c(2021, 12, 1))
```

What would we conclude based on these findings?

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.

A great tutorial on ARMA models can be found here.

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.

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.

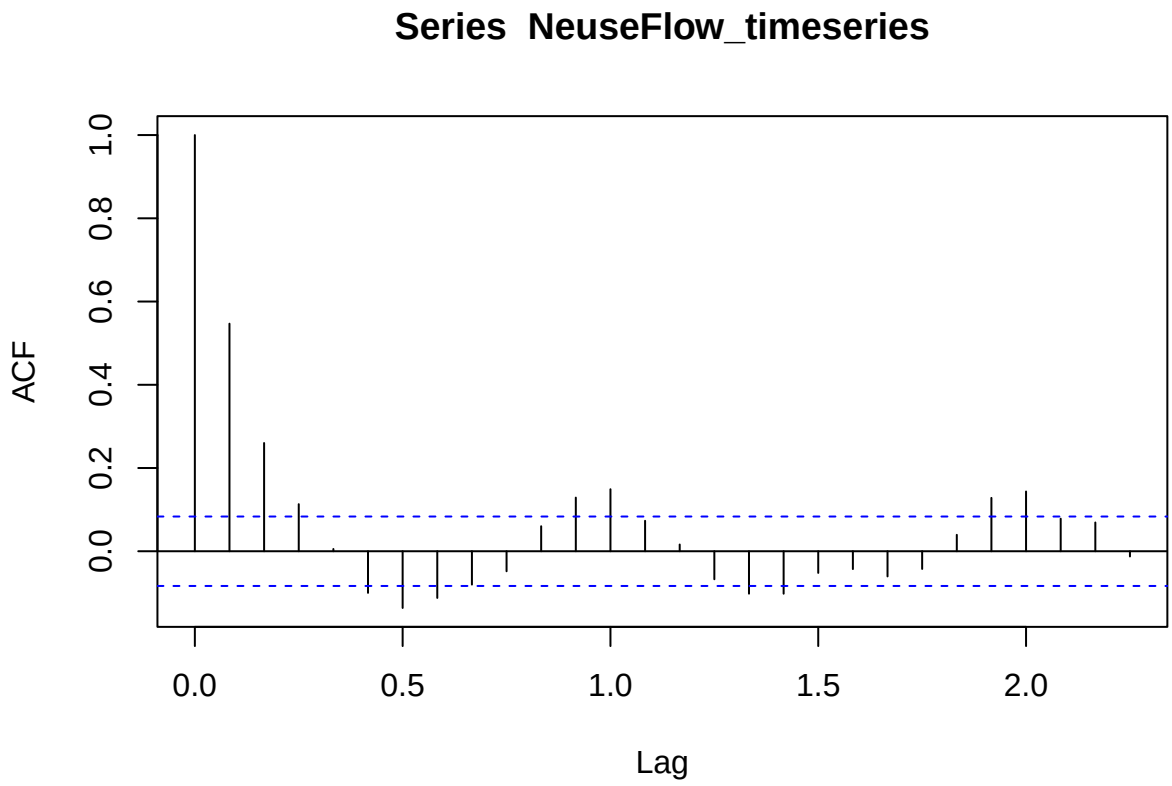
```
adf.test(NeuseFlow_timeseries, alternative = "stationary")
```

```
## Warning in adf.test(NeuseFlow_timeseries, alternative = "stationary"): p-value
## smaller than printed p-value
```

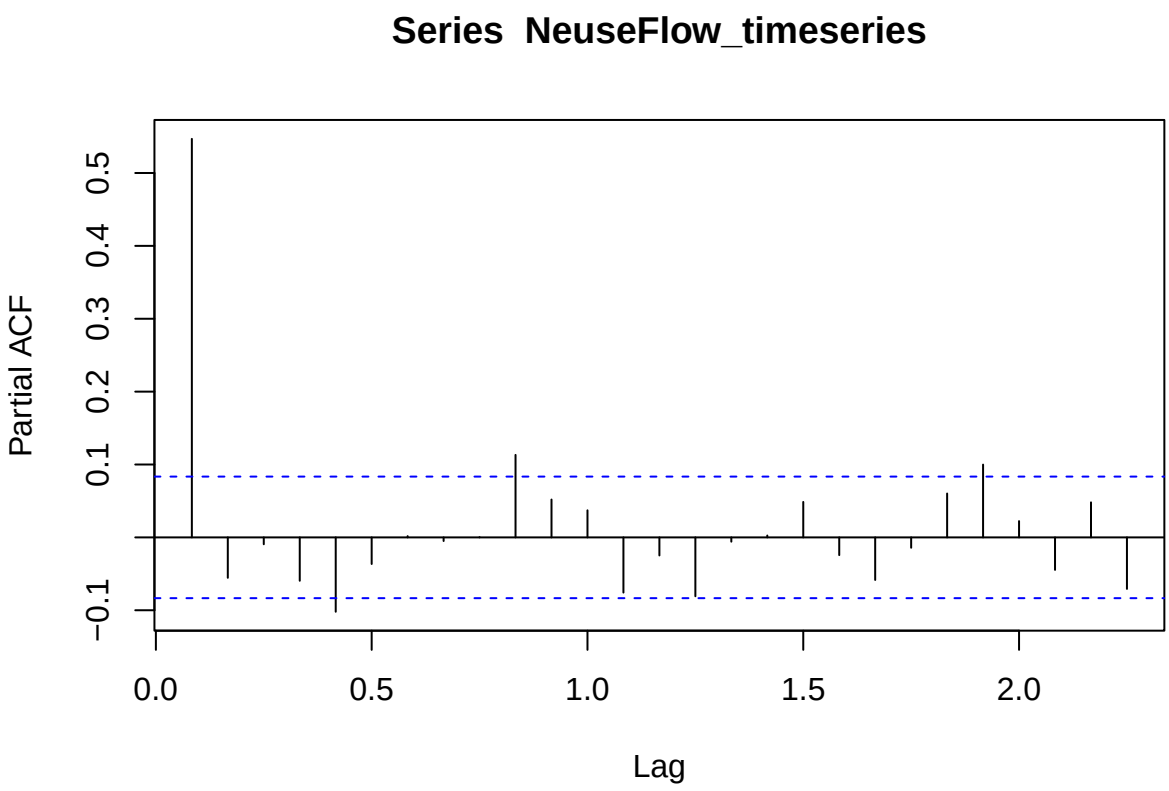
```
##
##  Augmented Dickey-Fuller Test
##
## data:  NeuseFlow_timeseries
## Dickey-Fuller = -7.9351, Lag order = 8, p-value = 0.01
## alternative hypothesis: stationary
```

Let's inspect the ACF and PACF plots. Notice the ACF plot has repeating positive and negative components (indicating seasonality), whereas the PACF decays over time without a clear seasonal structure.

```
acf(NeuseFlow_timeseries)
```



```
pacf(NeuseFlow_timeseries)
```



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, we will 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.

```
# run the arima function and search for best fit
auto.arima(NeuseFlow_timeseries, trace = TRUE)
```

```
##
##  Fitting models using approximations to speed things up...
##
##  ARIMA(2,0,2)(1,0,1)[12] with non-zero mean : Inf
##  ARIMA(0,0,0)            with non-zero mean : 14727.45
##  ARIMA(1,0,0)(1,0,0)[12] with non-zero mean : 14535.78
##  ARIMA(0,0,1)(0,0,1)[12] with non-zero mean : 14560.51
##  ARIMA(0,0,0)            with zero mean     : 15182.95
##  ARIMA(1,0,0)            with non-zero mean : 14533.38
##  ARIMA(1,0,0)(0,0,1)[12] with non-zero mean : 14528.88
##  ARIMA(1,0,0)(1,0,1)[12] with non-zero mean : Inf
##  ARIMA(1,0,0)(0,0,2)[12] with non-zero mean : 14528.01
##  ARIMA(1,0,0)(1,0,2)[12] with non-zero mean : Inf
##  ARIMA(0,0,0)(0,0,2)[12] with non-zero mean : 14715.53
##  ARIMA(2,0,0)(0,0,2)[12] with non-zero mean : 14529.03
##  ARIMA(1,0,1)(0,0,2)[12] with non-zero mean : 14528.74
##  ARIMA(0,0,1)(0,0,2)[12] with non-zero mean : 14557.35
```

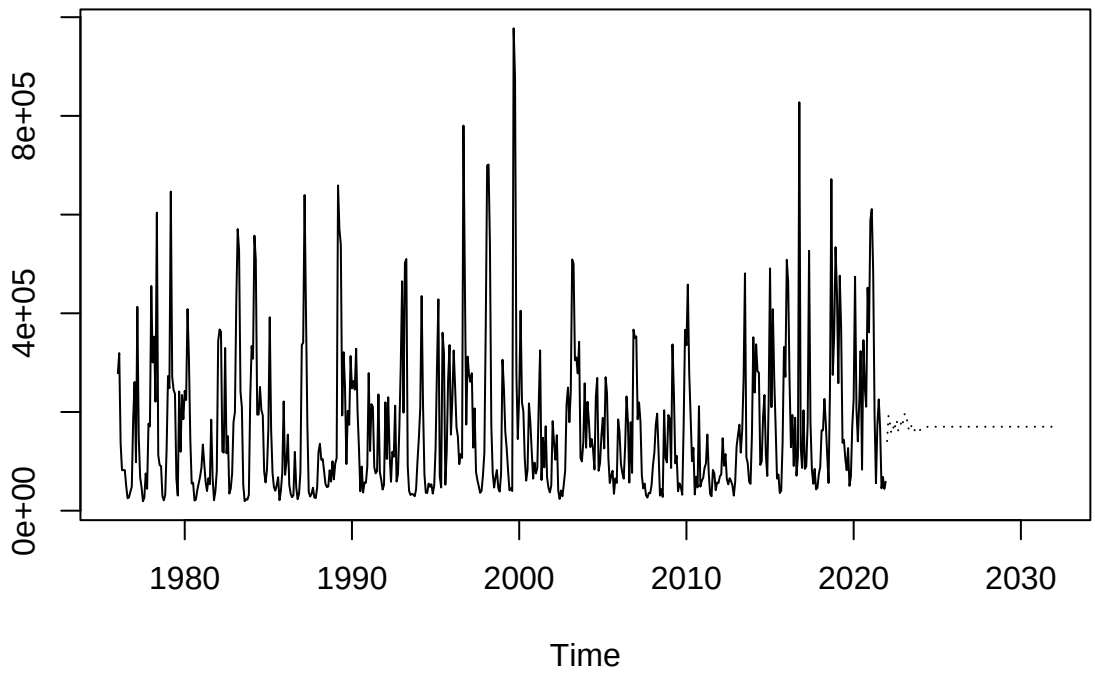
```
##  ARIMA(2,0,1)(0,0,2)[12] with non-zero mean : 14530.21
##  ARIMA(1,0,0)(0,0,2)[12] with zero mean     : 14599.3
##
##  Now re-fitting the best model(s) without approximations...
##
##  ARIMA(1,0,0)(0,0,2)[12] with non-zero mean : 14527.77
##
##  Best model: ARIMA(1,0,0)(0,0,2)[12] with non-zero mean
```

```
## Series: NeuseFlow_timeseries
## ARIMA(1,0,0)(0,0,2)[12] with non-zero mean
##
## Coefficients:
##          ar1    sma1    sma2       mean
##       0.5403  0.0896  0.0744  170111.89
## s.e.  0.0358  0.0438  0.0426   13310.83
##
## sigma^2 = 1.557e+10:  log likelihood = -7258.83
## AIC=14527.66   AICc=14527.77   BIC=14549.23
```

```
# create an object that defines the best fit model
fit <- arima(NeuseFlow_timeseries, c(1,0,0),seasonal = list(order = c(0,0,2), period = 12))

# make a prediction into the future
Neuseprediction <- predict(fit, n.ahead = 10*12)

# plot future predictions
ts.plot(NeuseFlow_timeseries, Neuseprediction$pred, lty = c(1, 3))
```



How do future predictions compare to the past? Why might the forecast be lacking in complexity?