All,

The purpose of AIC is to penalize models with more parameters to find a model which best explains the variation in your response using the least number of predictors, as whenever you add predictors to a model the R² increases. AICc is an AIC value corrected for small sample sizes, which penalizes additional predictors more than AIC. This reduces the chances of overfitting when you have small sample size or when you have a lot of predictors. Basically, you should always you AICc, as if you have enough samples to compensate for the number of predictors it will give you the same value as AIC anyway.

To begin you will want to install R, and R-Studio. R-Studio is an invaluable graphical user interface for R that will make working with R much easier.

You probably already have R installed, but if not:

http://cran.us.r-project.org/

R-Studio Here:

https://www.rstudio.com/ide/download/

Next download and install these R packages. If you installed R studio you click on the "Packages" tab in the lower right hand window, click on "Install Packages", and then type the package names "AICcmodavg" and "MuMIn".

Alternatively, here are the command line functions for installation.

AICcmodavg

install.packages("AICcmodavg")

MuMIn

install.packages("MuMIn")

Next step is to load the packages. For R-Studio, simply find the packages in the “Packages” tab from before. You will see both packages in the window. Simply click on the small box in front of both packages to load it. The code will be input automatically, and you will see a check mark in the box letting you know the package is loaded.

Again, , here are the command line functions for loading the packages.

library("AICcmodavg")

library("MuMIn")

These two packages serve different functions. AICcmodavg allows you to generate models a priori and then rank them. MuMIn is for data mining. You create a global model and MuMIn runs every possible iteration, but excludes interactions. Examples of input models include simple linear models using the function “lm”, or generalized linear models using “glm”. “glm” is a little bit more complex, as it lets you define different distributions. An example would be species richness data. Richness is count data, so you want to define the family as Poisson. For purpose of introduction, I’ve included a data set with a number of predictors and species richness as a response. However, the species richness data I’ve included is a mean value, so is not an integer and we don’t have to worry about distributions. We also won’t worry about normality, or homoscedasticity.

The libraries are loaded, so load the data I’ve provided.

1) In R-Studio: in the upper right window with “Workspace” and “History” tabs, activate the “Workspace” tab; click on “Import Dataset” and find the “data.csv” file I sent

2) A window will pop up make sure that “Yes” is highlighted in the “Heading” field, so the columns will be named. The default data fram name will be the name of your .csv, but you can change it here. This is useful if you have very long descriptive .csv names, but want a shorter name to make it faster to type in R. The other categories should default to “Seperator=comma”, “Decimal=period”, and “Quote=double quote (“)”. These fields come in handy when you are working with data from other sources. For instance, if the data is tab delimited you can change “Seperator=tab”.

3) Now the data is loaded and you will see it in the upper left window. When working with multiple matrices, each will have a tab. This is super handy, as you can see the names of columns. R is super picky about capitalization, so this makes it easy to check your names. If you look at the data, you will see all the fields. “sprichness” will be our response, and there are a number of climate, flow, and water quality variables to use as predictors.

\*Important Note

When working with data you have two main options when reading it.

1) You can use “attach(dataframe)” to make each column a defined variable.

2) You can call out the data frame to be used in the models you generate with the “data= dataframe” at the end of the model. We will use this approach.

**MuMIn**

Next we’ll look at the data mining approach using “MuMIn”. This package can be super slow with a lot of predictors, as combinations grow exponentially with additional ones. To start we’ll do a simple linear model and apply the “dredge” function from “MuMIn”.

1) First generate a simple Global linear model with species richness as the response and secchi (secchi), total yearly rainfall (TPCPcm), max flow (qmax), mean flow (qmean), dissolved oxygen (Domg), ammonia (NH3), fecal coliform count (FecCol), total phosphorus (Phos), and total phenolic compounds (phenugl).

lms<-lm(sprichness~secchi+TPCPcm+qmax+qmean+Domg+NH3+FecCol+Phos+phenugl, data=sp)

#This creates a file for your linear model called “lms”, which will be visible in the upper right window under “Values”. You can call results using “lms”, and “summary(lms)”

2) Now we will apply the “dredge” function to the model to run all possible combinations. I’ve copied this from R, because R is super picky about the font of quotations.

dlms<-dredge(lms, extra=c("R^2", "adjR^2"))

#This creates a file with all possible iteratations. I included the extra command, as dredge doesn’t give R^2 or adjR^2 by default. You need R^2 to select best models from competing models based on deltaAICc, and when reporting results.

3)You can call the results of dredge by just typing the name into a command line: “dlms”. You can see with that many models and variables it’s impossible to see each row fully. We will export the results of dredge to a csv file so we can view them.

\*Note: R will export everything to wherever you’ve set your working directory. Go to R-Studio “Tools” pull down, “Set Working Directory”, “Choose Directory”, and select where you want the output to go. Alternatively, the command line is “setwd(“c:\dir\dir”)”.

write.table(dlms, file="dredgelm.csv", sep=",", col.names=NA)

#This writes comma delimited csv named “dredgeglm.csv” to your working directory. The “col.names=NA” command just tells it to use the names of the columns dredge assigned.

4) Open dredgeglm.csv in Excel. By default dredge is sorted by the AICc value. The delta value is simply the difference between AICc values of the model and the model with the lowest AICc. You can use delta and R^2 to select the best model. If multiple models have a delta of ≤ 2 use the one with the best R^2 combined with your biological knowledge. \*Note: Models with a delta between 2-3 can also be considered if they have at least one additional predictor, due to the way AIC penalizes predictors.

**AICmodavg**

Now we will use a priori competing models. This method is slightly more complex coding wise than MuMIn. Few things to keep in mind; you want a global model for comparison, it’s easier to code interactions, and base models off biological responses. Additionally, if you plan on using model weight to look at the importance of individual predictors, you need to use them equally or correct for the number of models they are included in by dividing each variables weight by the number of models it’s in.

1) First we need to tell R we are creating a list we will be adding each model to.

richness<-list()

2) Next we generate models to add to the list. I’ve created a few competing models. These are simple 1 variable models, a max flow fecal coliform interaction to account for sewage overflow, an overall stress model, and a global model.

richness[[1]]<-lm(sprichness~secchi,data=sp)

richness[[2]]<-lm(sprichness~FecCol,data=sp)

richness[[3]]<-lm(sprichness~qmax,data=sp)

richness[[4]]<-lm(sprichness~NH3,data=sp)

richness[[5]]<-lm(sprichness~phenugl,data=sp)

richness[[6]]<-lm(sprichness~Domg,data=sp)

richness[[7]]<-lm(sprichness~qmax\*FecCol,data=sp)

richness[[8]]<-lm(sprichness~secchi+Domg+NH3+phenugl,data=sp)

richness[[9]]<-lm(sprichness~secchi+Domg+NH3+phenugl+qmax\*FecCol,data=sp)

#This tells R the order in which to add models to “richness” while generating the models.

3) Next we create another list defining the model names. This is important to do in the same order you generated the models.

Modnames<-c("Secchi","Sewage", "High Flow", "Ammonia", "Phenols", "Oxygen", "CSO", "Stress", "Global")

4) Next we use “aictab” to rank the models using AICc

richnessaic<-aictab(richness, modnames=Modnames, sort=T)

5) Now write the table to a csv.

write.table(richnessaic, file="Competing.csv", sep=",", col.names=NA)

6) Open the Competing.csv. Notice that this doesn’t give R^2 or adjR^2. This package is more for model averaging, so it doesn’t do R^2. There are a couple ways to deal with this. First method is to create a new list and extract the values from summaries using:

summary(richness[[1]])$r.squared, and summary(richness[[1]])$adj.r.squared.

Honestly though, this method is a HUGE pain. When you use a priori models you probably won’t have to many, so the easiest way is to simply get a summary for each model and manually type them into the Competing.csv.