



# ELEMENTS OF DATA SCIENCE AND STATISTICAL LEARNING

SPRING 2017

Week 4

# OUTLINE

- Multiple regression
- Algae dataset example
- Resampling
  - Cross-validation
  - Bootstrap
- Bias-variance decomposition
- K-nearest neighbors regression

# MULTIPLE LINEAR REGRESSION

- The framework of simple linear regression can be easily generalized to the case of multiple predictor variables

$$Y \sim X_1 + X_2 + X_3 + \cdots + X_k + \varepsilon$$

Or in traditional “mathematical” notation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \cdots + \beta_k X_k + \varepsilon$$

- The coefficients are calculated from the data using closed form matrix expressions (which we are not discussing here) → still a very efficient computational model

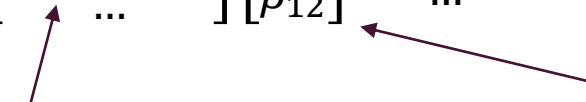
# POLYNOMIAL REGRESSION

- Often times still referred to as “linear regression”
- A variant of multiple linear regression when we manually introduce non-linear functions of the *predictor variables*, for instance:

$$Y \sim X_1 + X_1^2 + \varepsilon, \text{ or}$$

$$Y = \beta_0 + \beta_{11}X_1 + \beta_{12}X_1^2 + \varepsilon$$

- Important: multiple/polynomial regressions are still *linear models* in statistics. Indeed, the random variables, or combinations/transformations thereof are just *numbers* as far as model fitting is concerned! We *measure* some data values and we solve/optimize a system of (**linear!**) equations for the regression coefficients!!

$$\begin{bmatrix} y_1 \\ y_2 \\ \dots \end{bmatrix} = \begin{bmatrix} \beta_0 + \beta_{11}x_1 + \beta_{12}x_1^2 \\ \beta_0 + \beta_{11}x_2 + \beta_{12}x_2^2 \\ \dots \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \dots \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_{11} \\ \beta_{12} \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \dots \end{bmatrix}$$


Design matrix. Fixed numbers!

“Variables” we are solving for  
(model coefficients!)

# INTERACTION

- After we allowed multiple predictions and non-linear transformations (with respect to predictor variables), there is of course no reason not to consider *interactions* between explanatory variables
- The effect of variable  $X_1$  on the outcome  $Y$  depends on the value of other variable,  $X_2$ :

$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_{12} X_1 X_2 + \varepsilon$  can be rewritten as:

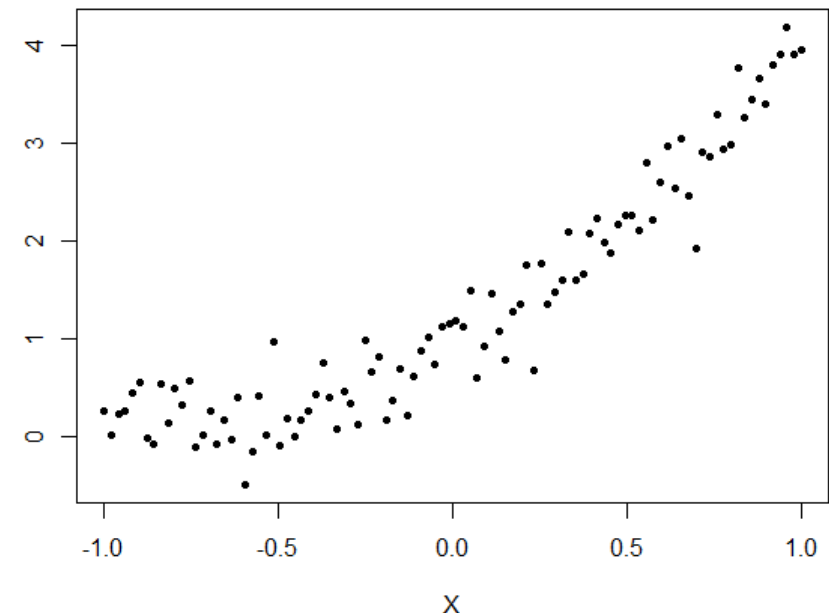
$Y = \beta_0 + (\beta_1 + \beta_2 X_2) X_1 + \varepsilon$  (just like it was said above: effect of  $X_1$  depends on  $X_2$  !)

- Synergy between variables can be more important in many cases than individual higher-order degrees!
- But in general, the more complex dependence we are trying to uncover, the more data we need!

# MODEL SHOULD BE ADEQUATE TO THE DATA

- The “linear model” turns out to be a pretty rich and flexible framework. How do we choose which variables/transformations to use, in which combination?
  - No easy answer. Or there is one: we do not know
  - The only important thing to remember is “*the data has the ultimate say*”. What this means is that even if we have a pretty good idea of what dependency we are looking for, the small amount of data available and/or noise too strong might prevent us from being able to fit the “correct” model. Or rather, the “correct” model is the one that describes the data at hand, even if we know from some physical principles that the true dependence has additional terms.
- Let us illustrate this by a simulation
  - A dataset with *built-in* second-order dependency and some weak noise:

```
> X=seq(-1,1,length=100)
> Y=1+2*X+X^2+rnorm(length(X),sd=0.3)
> plot(X,Y,pch=19,cex=0.7)
```



# TOY MODEL, CONTINUED

- Let us compare two models :  $Y \sim X$  and  $Y \sim X + X^2$ :

```
> summary(lm(Y~X))

...
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.32721    0.04811   27.59  <2e-16 ***
X            1.91657    0.08251   23.23  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4811 on 98 degrees of freedom
Multiple R-squared:  0.8463,    Adjusted R-squared:  0.8447
F-statistic: 539.6 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
> summary(lm(Y~X+I(X^2)))

...
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.89968    0.04326   20.80  <2e-16 ***
X            1.91657    0.04946   38.75  <2e-16 ***
I(X^2)       1.25719    0.09483   13.26  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2884 on 97 degrees of freedom
Multiple R-squared:  0.9453,    Adjusted R-squared:  0.9442
F-statistic: 838.8 on 2 and 97 DF,  p-value: < 2.2e-16
```

- The second order term improves the fit a lot, *is* significant *and* we get a reasonable estimate for its coefficient.

# TOY MODEL, CONTINUED II

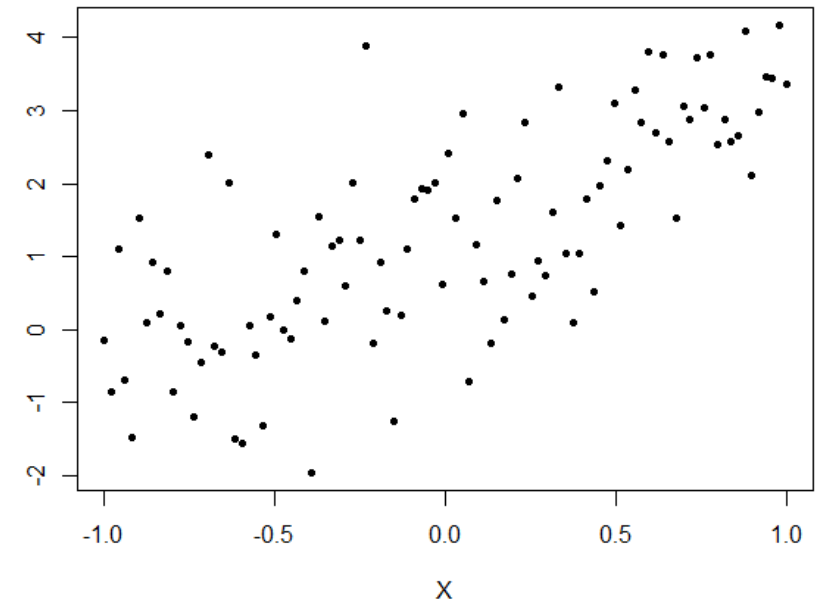
- Let us now introduce stronger noise into the same dependency:  $Y = 1 + 2 \cdot X + X^2 + \text{rnorm}(\text{length}(X), \text{sd}=1)$

```
> summary(lm(Y~X))
...
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.2909     0.1035   12.47  <2e-16 ***
X             1.8683     0.1775   10.52  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.035 on 98 degrees of freedom
Multiple R-squared:  0.5305,    Adjusted R-squared:  0.5257
F-statistic: 110.7 on 1 and 98 DF,  p-value: < 2.2e-16
```

```
> summary(lm(Y~X+I(X^2)))
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.0573     0.1528   6.918 4.91e-10 ***
X             1.8683     0.1747  10.694 < 2e-16 ***
I(X^2)        0.6868     0.3350   2.050  0.043 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.019 on 97 degrees of freedom
Multiple R-squared:  0.55, Adjusted R-squared:  0.5408
F-statistic: 59.29 on 2 and 97 DF,  p-value: < 2.2e-16
```



- Fit is not very good *and* it barely improved!
- Coefficient is marginally significant
- Determined with larger error



# OUTLINE

- Multiple regression
- Algae dataset example
- Resampling
  - Cross-validation
  - Bootstrap
- Bias-variance decomposition
- K-nearest neighbors regression

## MORE REALISTIC EXAMPLE

- In the following, we will try to illustrate multiple regression and related concepts on a real dataset
- We will follow up by introducing model cross-validation

# ALGAE DATASET

- More data to play with: <https://kdd.ics.uci.edu/>

“The archive is intended to serve as a permanent repository of publicly-accessible data sets for research in KDD and data mining. It complements the original [UCI Machine Learning Archive](#), which typically focuses on smaller classification-oriented data sets.”

- Computational Intelligence and Learning (COIL) competition, 1999: <https://kdd.ics.uci.edu/databases/coil/coil.data.html>

“This data comes from a water quality study where samples were taken from sites on different European rivers of a period of approximately one year. These samples were analyzed for various chemical substances including: nitrogen in the form of nitrates, nitrites and ammonia, phosphate, pH, oxygen, chloride. In parallel, algae samples were collected to determine the algae population distributions.

The competition involved the prediction of algal frequency distributions on the basis of the measured concentrations of the chemical substances and the global information concerning the season when the sample was taken, the river size and its flow velocity”

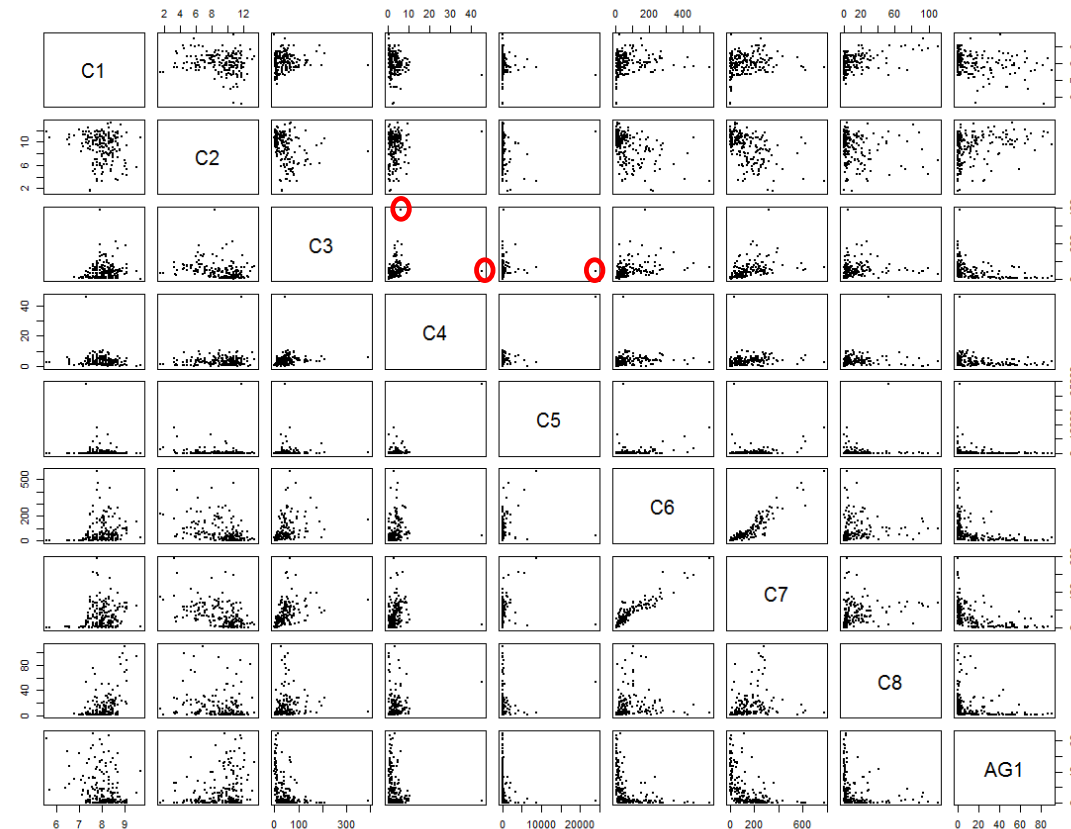
- 200 samples (observations), 18 variables: categorical: season, river size, river flow; continuous: concentrations of 8 chemicals, frequencies of 7 algae species. Goal: predict all 7 algae frequencies.

```
> algae=read.table("coil.analysis.data.txt", header=F, sep=" ", row.names=NULL, na.strings="XXXXXXX")
> colnames(algae)=c("season", "size", "velocity", paste("C", 1:8, sep=""), paste("AG", 1:7, sep=""))
> algae[1:3,]
```

	season	size	velocity	C1	C2	C3	C4	C5	C6	C7	C8	AG1	AG2	AG3	AG4	AG5	AG6	AG7
1	winter	small	medium	8.00	9.8	60.80	6.238	578.000	105.000	170.000	50.0	0.0	0.0	0.0	0.0	34.2	8.3	0.0
2	spring	small	medium	8.35	8.0	57.75	1.288	370.000	428.750	558.750	1.3	1.4	7.6	4.8	1.9	6.7	0.0	2.1
3	autumn	small	medium	8.10	11.4	40.02	5.330	346.667	125.667	187.057	15.6	3.3	53.6	1.9	0.0	0.0	0.0	9.7

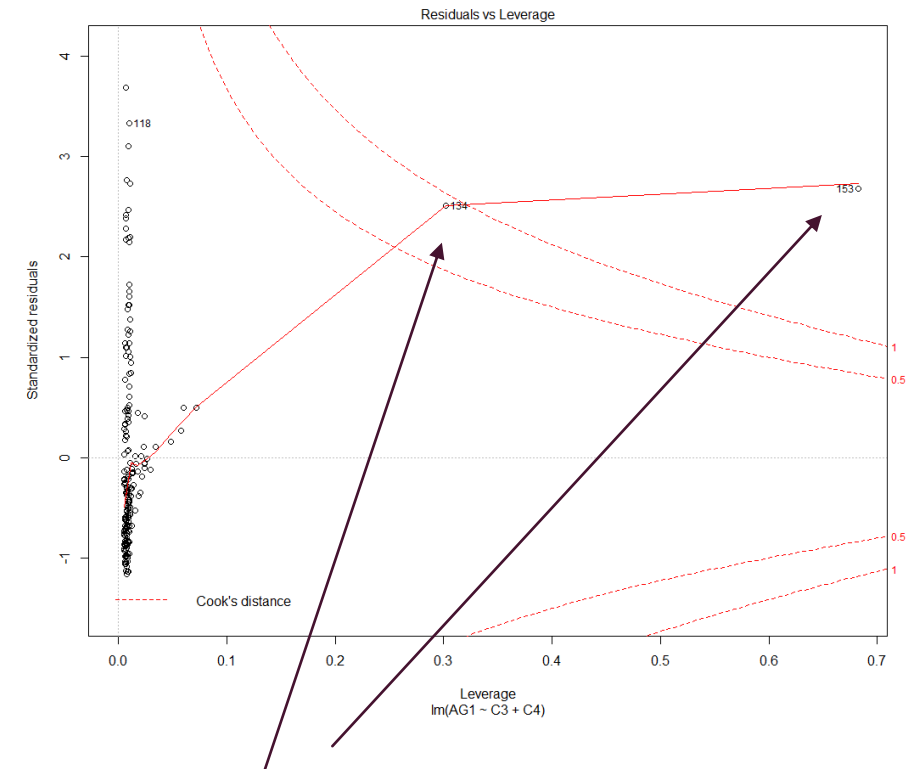
# LOOKING AT THE DATA: DEPENDENCIES

- `pairs(algae[,4:12],cex=0.2,pch=19)` to see pairwise scatterplots:
- Draw scatterplot with `plot(algae[,c(i,j)])` followed by `identify(algae[,c(i,j)])` to identify outliers
- Two samples: 134, 153



# REMOVING OUTLIERS

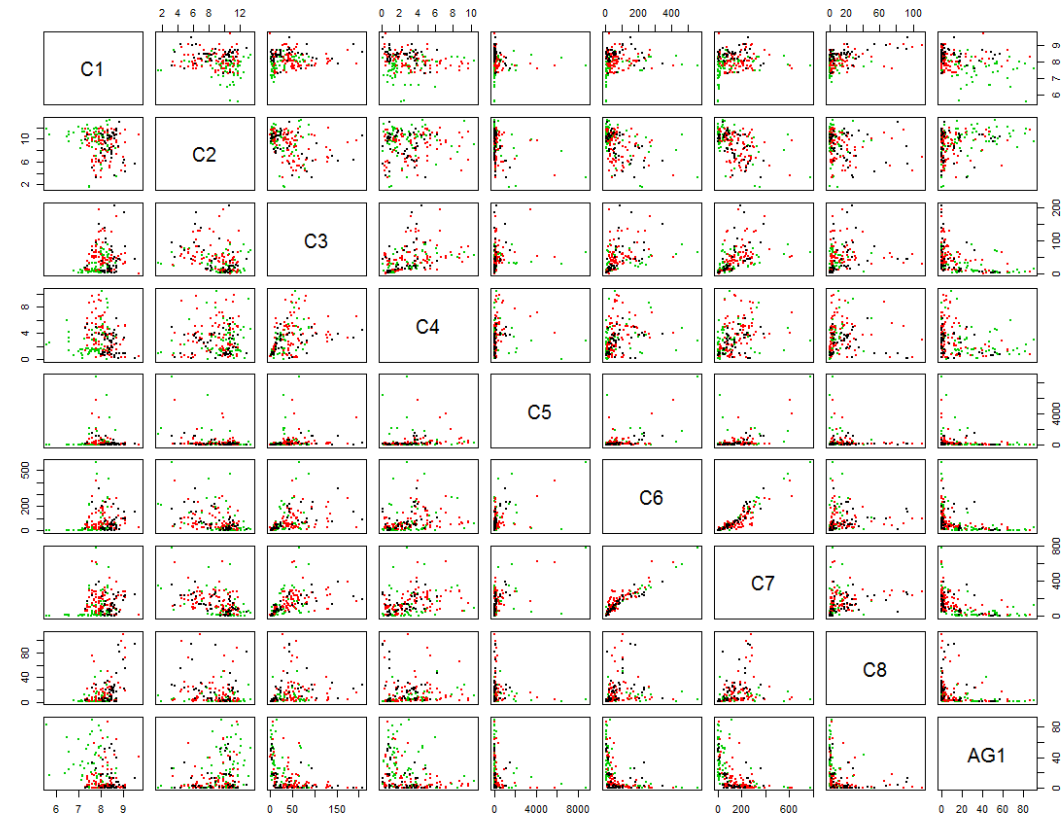
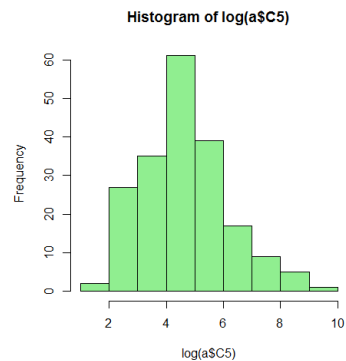
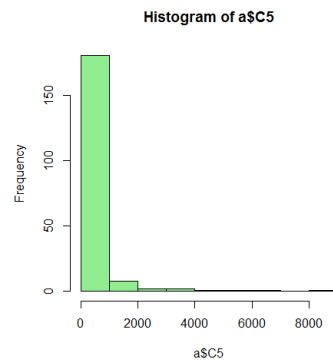
- Quick fit with `lm(AG1~C3+C4, data=algae)` confirms that the outliers have very large effect (by residuals vs leverage diagnostic plot)
- These points might be experimental/clerical errors or true measurements that are highly unusual
- One can still argue that even if the data points are “real”, they represent
  - (a) very small percentage of cases (~1%)
  - (b) potentially, completely different situation/regime (for which we do not have enough data)
- Hence one can argue that we cannot adequately fit that separate regime anyway, while by keeping these points we allow them to exert unfairly strong effect on the overall fit
- Finding a good fit (if we can!) that works for 98% of cases does not sound like a bad result!



```
# discard outliers:  
a = algae[-c(134,153),]
```

# DEPENDENCIES II

- Repeat the scatterplot with outliers removed
- color by factors: the strongest effect that can be identified visually: river size (note how green='small' tends to have distinct location)
- Note problematic distribution of C5
  - Huge range, very non-uniform distribution
  - Confirm: `hist(a$C5)` ; `hist(log(a$C5))`



# VARIABLE TRANSFORMATION

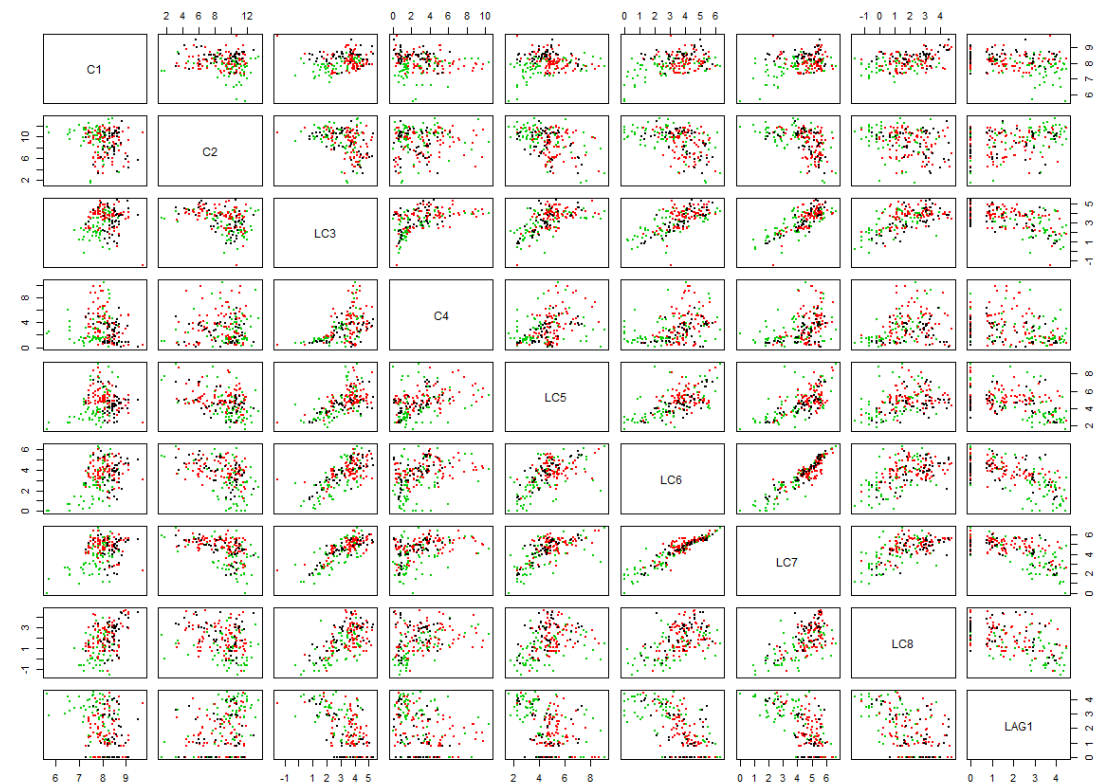
- By examining the variables and their distributions closer, we discover that most have in fact log-normal distributions (similar to C5), including outcomes (plot histograms of the variables and their logs and observe!)
- Outcomes are trickier as many values are 0 (can't take log!): *Regularized log*:  $\log(x+r)$  (e.g.  $r=1$ )

```
a$LC3=log(a$C3)
a$LC5=log(a$C5)
a$LC6=log(a$C6)
a$LC7=log(a$C7)
a$LC8=log(a$C8)
for (col in
paste("AG",1:7,sep="")) {
  a[[paste("L",col,sep="")]] =
    log(a[[col]]+1)
}
```

Variable X is distributed Log-normally if  $\log X$  exhibits normal distribution.

Multiplicative random processes:

$X = Z_1 * Z_2 * \dots * Z_N \rightarrow \log X = \log Z_1 + \dots + \log Z_N$  (normally distributed by CLT!)



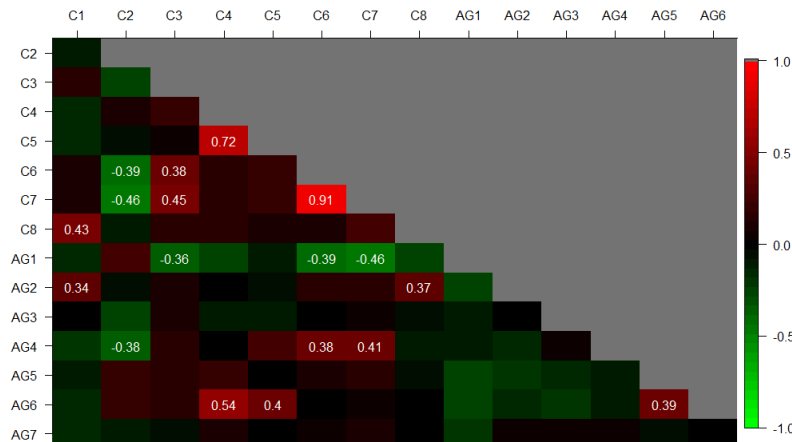
# CORRELATIONS

- Let us compare correlations in original and transformed variables

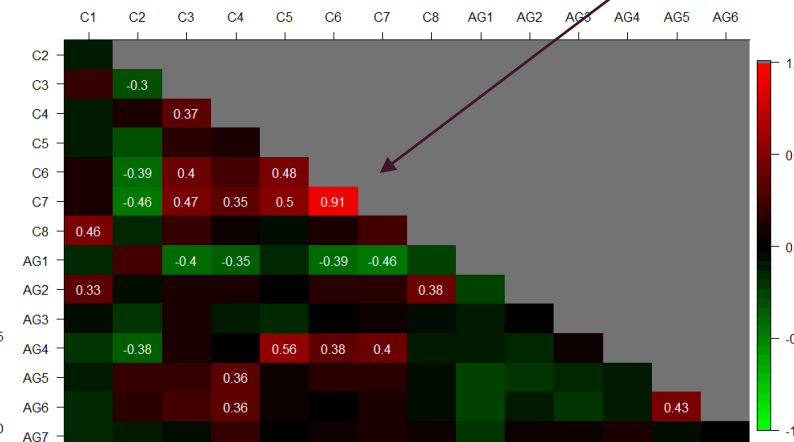
```
# use one of the following two commands to define cr (1st is original, 2nd is transformed)
cr=cor(a[4:18],use="complete")
cr=cor(a[,c("C1","C2","LC3","C4","LC5","LC6","LC7","LC8",paste("LAG",1:7,sep=""))],use="complete")
# the code we define in next slide will use this (it will ignore all 1's and paint them grey as below):
for ( i in 2:(nrow(cr)) ) {
  for ( j in 1:(i-1) ) cr[i,j]=1
}
```

Will not use C7!!

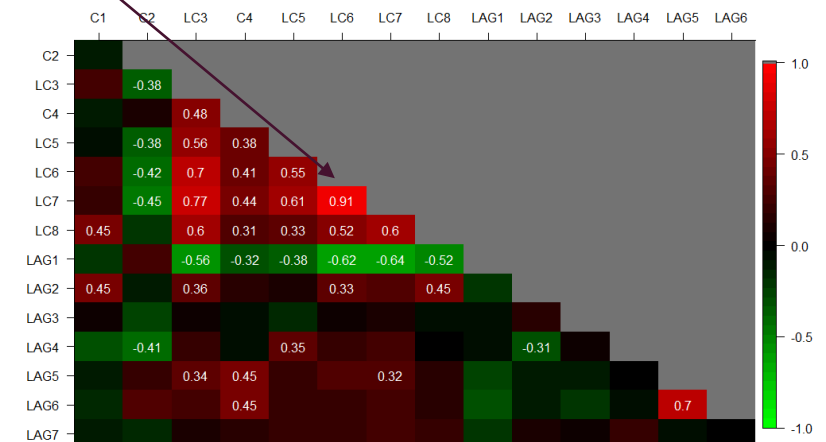
Full dataset, original



Outliers removed, original



Outliers removed, transformed





# DIGRESSION: IMAGE PLOTS

- The code below generates heatmaps of correlation matrices shown in the previous slide
  - Value 1 indicates “unused cell, paint with grey” (you can change it to use purely geometrical conditions instead)
  - You may want to wrap it as a function

```
library(fields) # need this for image.plot()
# leave margin on the right, will need it later to draw legend:
oldpar=par(oma=c(0,0,0,4))
# define color breaks. Values in each bin (between breaks) will
# be painted with corresponding color (as set below). If we
# don't set breaks manually here, they will be determined
# automatically from the actual data range: not good as we want
# to ensure that black is right at 0:
col.breaks=seq(-1,0.99,length=40)
nbins=length(col.breaks)-1 # how many bins we got for values
# generate palette (1 color corresponds to each data bin):
colors = colorRampPalette(c("green","black","red"))(nbins)
# add an extra color for masked values (we mask >= 1.0)
# and also add corresponding extra bin boundary:
colors = c(colors,"grey45")
col.breaks=c(col.breaks,1.01)

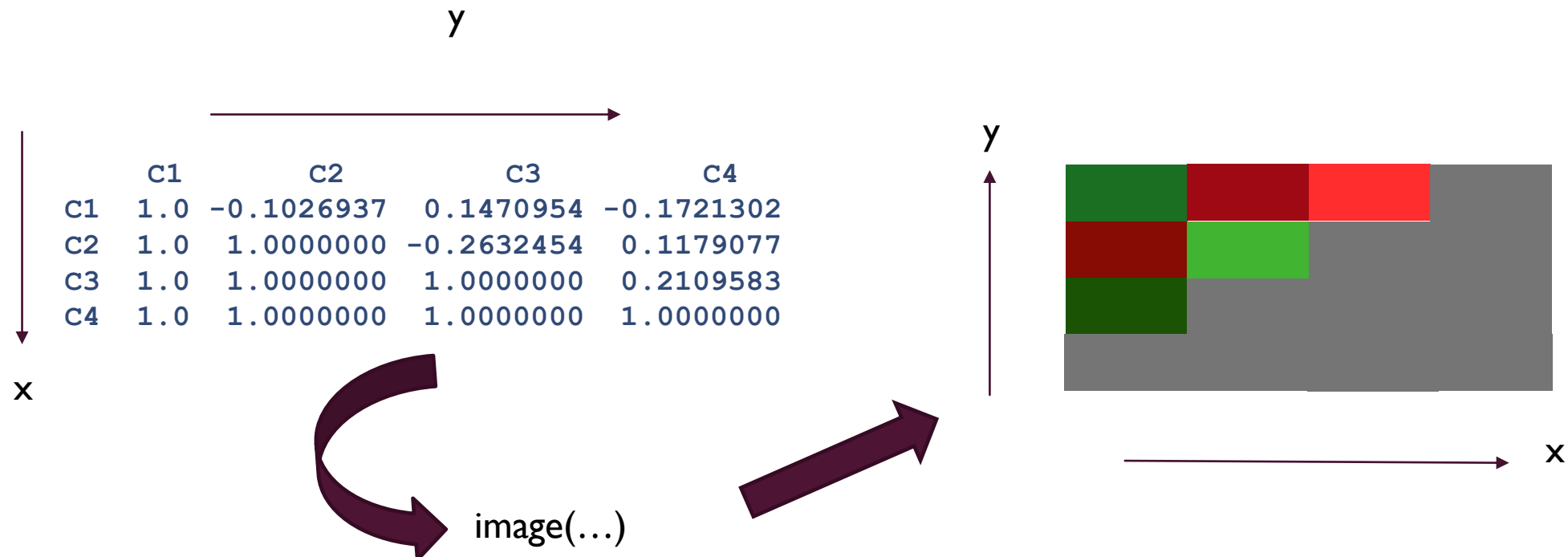
N=nrow(cr)
```

```
image(1:(N-1),1:(N-1),
      cr[-N,N:2],axes=F,col=colors,
      breaks=col.breaks,xlab="",ylab="")
axis(3,at=1:N,labels=rownames(cr))
axis(2,at=1:N,labels=colnames(cr)[N:1],las=2)
for( i in 1:(N-1) ) { # x coord
  for( j in 1:(N-i) ) { # y coord
    val=cr[i,N-j+1]
    if ( i != N-j+1 && abs(val) > 0.3 ) {
      text(i,j,round(val,2),col="white")
    }
  }
}

par(oma=c(0,0,0,1)) # change margin for legend
image.plot(cr,legend.only = T,col=colors,
           breaks = col.breaks)
par(oldpar)
```

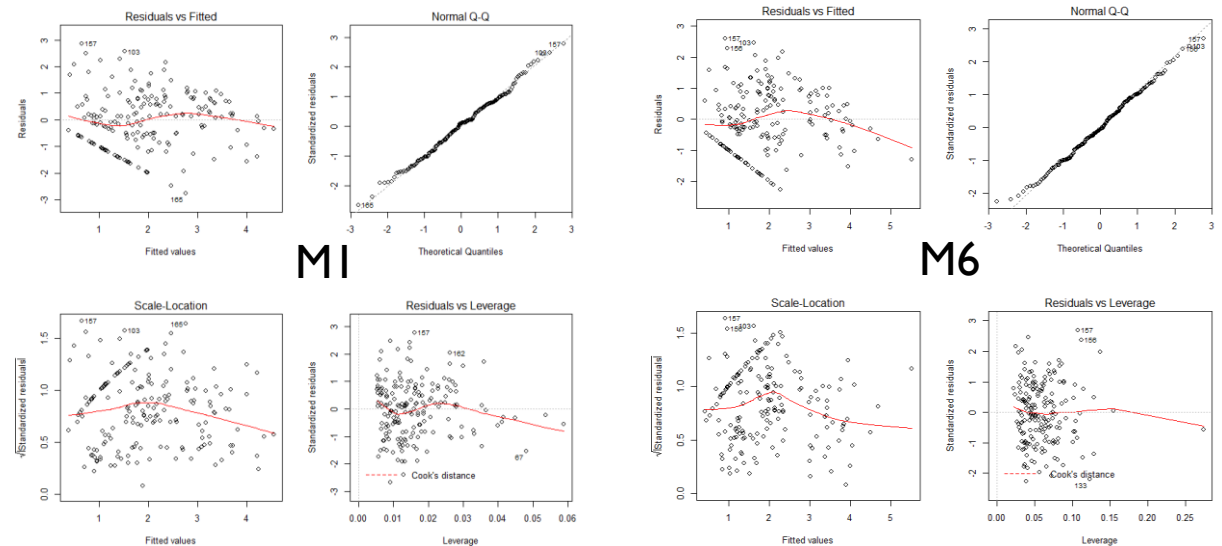
# GEOMETRY OF IMAGE PLOT

- In the previous slide, the only purpose of strange indexing into the `cr` matrix is to account for the way `image()` interprets “x” and “y” coordinates. We also specified explicit integer coordinates along the dimensions of the data matrix - only to have a more convenient way to place the text labels later. If we did not specify `x, y` vectors in `imageplot()` and passed to it just the data matrix, the coordinates would be automatically assumed to be an equally spaced grid in  $[0, 1] \times [0, 1]$



# BUILD A MODEL!

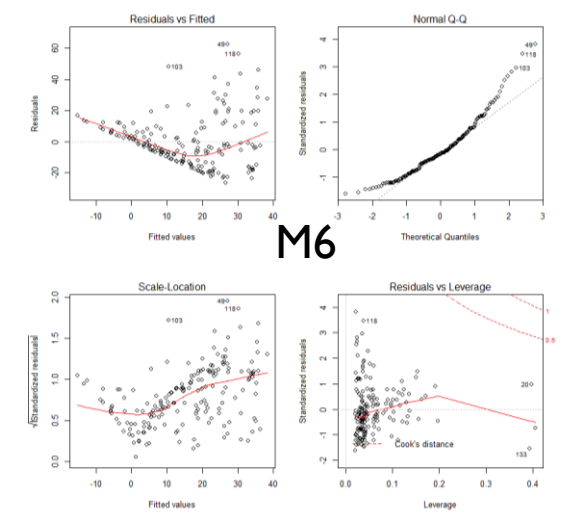
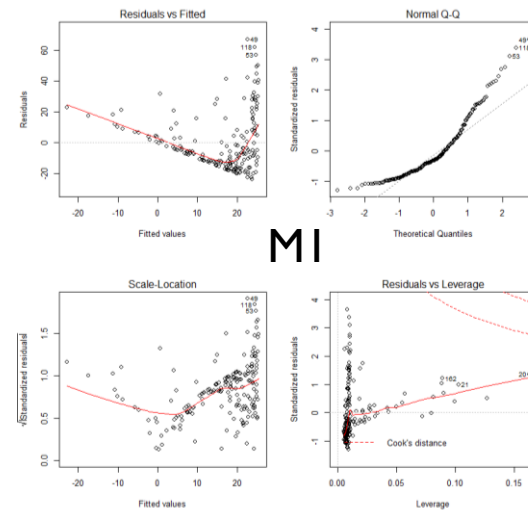
- Too many variables – what should we choose?
- Let us try to build few different models
- In a wide range of parameter choices the models perform very similarly!
- All models result in very similar  $R^2$  (46-49%) and RSE (next slides)



```
M1=lm(LAG1~LC6+LC8, data=a)
M2=lm(LAG1~LC3+C4+LC5+LC6+LC8, data=a)
M3=lm(LAG1~LC6+LC8+LC6:velocity+velocity, data=a)
M4=lm(LAG1~LC5+LC6+LC8+LC6:velocity+velocity+LC5:LC8, data=a)
M5=lm(LAG1~C1+C2+LC3+C4+LC5+LC6+LC8+LC5:LC8, data=a)
M6=lm(LAG1~C1+C2+LC3+C4+LC5+LC6+LC8+LC5:LC8+size, data=a)
```

# COMPARE WITH UNTRANSFORMED DATA

- It is instructive to see how the same variables would perform when used on original scale
- The results further suggest (confirm?) that we indeed should have use transformed data



```
M1.orig=lm(AG1~C6+C8,data=a)
M2.orig=lm(AG1~C3+C4+C5+C6+C8,data=a)
M3.orig=lm(AG1~C6+C8+C6:velocity+velocity,data=a)
M4.orig=lm(AG1~C5+C6+C8+C6:velocity+velocity+C5:C8,data=a)
M5.orig=lm(AG1~C1+C2+C3+C4+C5+C6+C8+C5:C8,data=a)
M6.orig=lm(AG1~C1+C2+C3+C4+C5+C6+C8+C5:C8+size,data=a)
```

# COMPARING DIFFERENT MODELS

- How do we compare models? Which one is “better”?
- We can look at the accuracy of the *fit*
  - Check  $R^2$ : *adjusted  $R^2$*  is more appropriate in multiple linear regression! – tries to adjust for the fact that  $R^2$  *always* increases as we add variables, no matter significant/important or not.
  - Check RSE, check F statistics (variance explained)

```
sapply(list(M1=M1,M2=M2,M3=M3,M4=M4,M5=M5,M6=M6),
       function(x) { unlist(summary(x)[c("r.squared","adj.r.squared","sigma")]) })
```

	M1	M2	M3	M4	M5	M6
r.squared	0.4584886	0.4670565	0.4985102	0.5195828	0.4829294	0.4934891
adj.r.squared	0.4525704	0.4521698	0.4817005	0.4978691	0.4591560	0.4640408
sigma	1.0390756	1.0408755	1.0110518	0.9951567	1.0280586	1.0234054

```
sapply(list(M1=M1.orig,M2=M2.orig,M3=M3.orig,M4=M4.orig,M5=M5.orig,M6=M6.orig),
       function(x) { unlist(summary(x)[c("r.squared","adj.r.squared","sigma")]) })
```

	M1	M2	M3	M4	M5	M6
r.squared	0.2138023	0.3059402	0.2870436	0.2893107	0.3391252	0.3712317
adj.r.squared	0.2052100	0.2865530	0.2631456	0.2571891	0.3087402	0.3346754
sigma	18.4940002	17.5472460	17.8071938	17.8790231	16.9244282	16.6039016

# AIC/BIC CRITERIA

- Information criteria: idea similar to model selection approaches we are going to be studying later: models with too many variables are *penalized*
- This approach is principled and based on rigorous statistical considerations; with somewhat different assumptions, one can estimate the “divergence” between the predicted distribution  $\hat{P}(Y)$  and the true distribution  $P(Y)$

$$AIC = -2\log L(\theta|\text{data}) + 2k$$

$$BIC = -2\log L(\theta|\text{data}) + k\ln n,$$

$k$  =# variables,  $n$ =# observations

- As it is seen from the definitions above, the larger the score, the worse the model
- Don't read too much into small differences in AIC/BIC!
- The models at hand are largely similar! Scores would start deteriorating quickly at larger  $k$

```
> AIC (M1 ,M2 ,M3 ,M4 ,M5 ,M6)
      df      AIC
M1    4 547.0800
M2    7 547.7308
M3    8 540.7987
M4   10 536.8140
M5   10 540.2307
M6   12 540.4547
Warning message:
... models are not all fitted to the same number of observations
> BIC (M1 ,M2 ,M3 ,M4 ,M5 ,M6)
      df      BIC
M1    4 559.9830
M2    7 570.2733
M3    8 566.6047
M4   10 569.0715
M5   10 572.3255
M6   12 578.9685
Warning message:
... models are not all fitted to the same number of observations
```

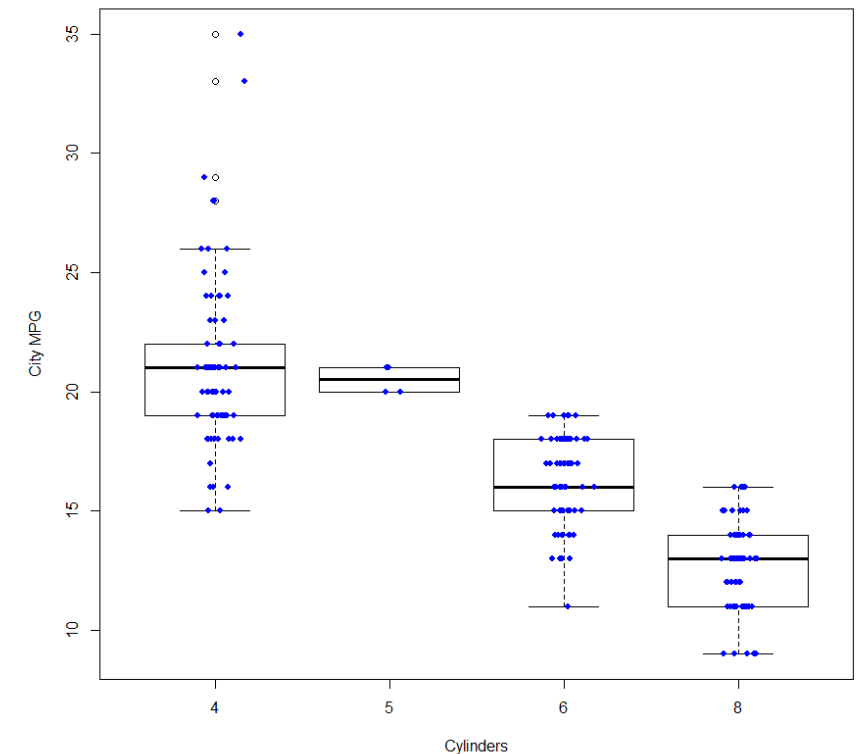
# A NOTE ON THE MODEL AND DATA SIZES

- Note that ideally we are looking to model joint probability of the data  $P(Y, X_1, \dots, X_p)$  or at least the conditional  $P(Y|X_1, \dots, X_p)$ .
- In the algae dataset, there are 3 categorical variables: season (4 values), river size (3 values), river flow velocity (3 values).
  - If we consider all combinations of these 3 factors alone, it's  $4*3*3=36$  different “bins”.
  - In order to adequately quantify distribution of  $Y$  in each of those bins (or prove that different bins have no effect on  $Y$ , e.g.  $P(Y|\text{season, size, flow})=P(Y)$ ), we need sufficient number of observations in each bin.
  - But we have only 200 observations: less than 6 per bin, considering *only* the categorical variables (and we'd want to quantify how  $Y$  depends on C1-C8 in each bin!)
- We *almost never* have enough data points and thus we *have* to resort to all kinds of approximations
  - We may never be able to extract the “true” dependency
  - Given the (amount of) data there is only so much we can glean from them!
  - We might need to use a model (e.g. a linear one) if only to summarize limited data as best as we can, without necessarily believing that the true dependence is exactly as we model it (we might not have enough data to tell the difference!)
  - But being smart about the data (e.g. transformations) and the models (which variables to choose? Use linear model vs something else? Can we use domain knowledge to our advantage?) definitely helps tremendously

# CASE IN QUESTION: WITH LOTS OF DATA, DO WE NEED A MODEL?

- Let us consider dataset `mpg` as an example
  - Let us look at city mpg vs number of cylinders
  - First, consider number of cylinders as a categorical variable. There are different types of cars. Some of them are “4cyl”, some of them are “6cyl”, etc
  - Number of cylinders is nothing but a *categorical label*
  - We do have plenty of observations in each category. So, we can estimate the mean mpg in each category (except for “5 cyl”) with very high accuracy.
  - Thus in principle we can make very good prediction in each category (again, except for 5cyl) without resorting to any kind of a model!
  - Why would we need a model at all?

```
boxplot(cty~cyl,data=mpg,xlab="Cylinders",ylab="City MPG")
points(as.numeric(as.factor(mpg$cyl))+rnorm(nrow(mpg),sd=0.05),
       mpg$cty,col='blue',pch=19,cex=0.8)
```





# A MODEL CAN BETTER EXPLAIN THE DATA

- At the very least, a model can *explain* and *summarize* the data
- Useful for inference (note how in the example below the linear model does very good job predicting for 4, 6 and 8
  - Better understanding the world around us
  - Formulating hypotheses

```
plot(mpg$cyl+rnorm(nrow(mpg), sd=0.05), mpg$cty, col='blue', pch=19, cex=0.8)
abline(lm(cty~cyl, data=mpg), col='red')
lm(cty~cyl, data=mpg)
```

Call:

```
lm(formula = cty ~ cyl, data = mpg)
```

Coefficients:

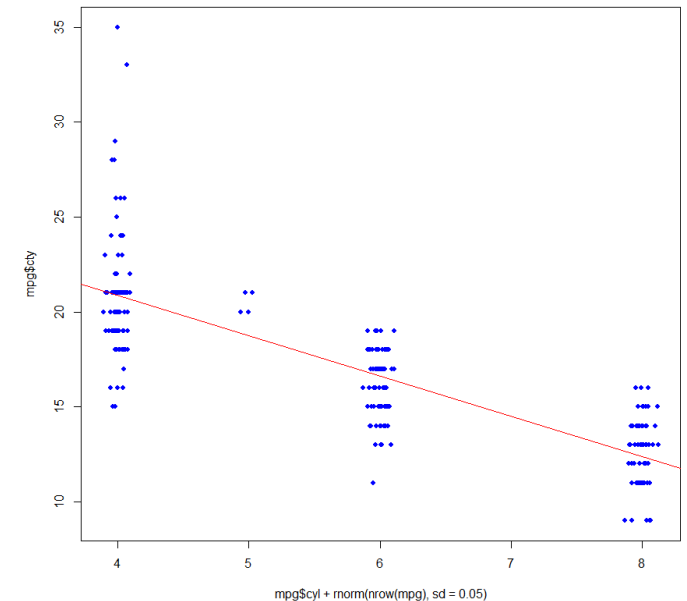
(Intercept)	cyl
29.390	-2.128

```
sapply(4:8, function(x) { mean(mpg$cty[mpg$cyl==x]) })
```

```
[1] 21.01235 20.50000 16.21519      NaN 12.57143
```

```
predict(lm(cty~cyl, data=mpg), newdata=data.frame(cyl=4:8))
```

1	2	3	4	5
20.87850	18.75052	16.62253	14.49455	12.36656

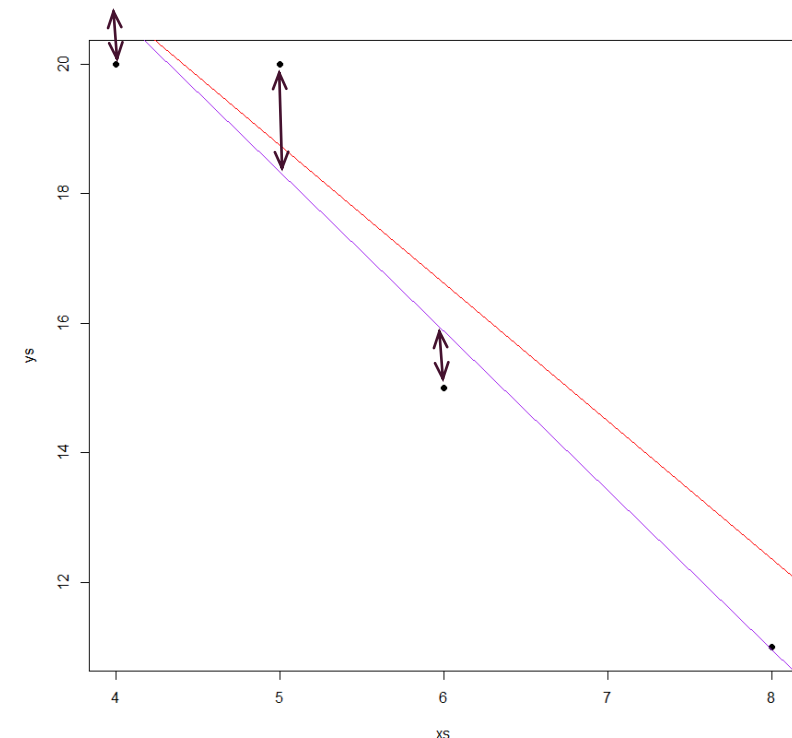


# DEPENDENCE IS A STRONG ASSUMPTION

- Let us sample just one car for each # of cyl from the observed distributions
- The best “prediction” we can make, per each category taken separately is just the (single!) observed value; and we do not even know anything about the error we should expect
- However, if we use a (linear) model, we (1) estimate error from pooled observations (2) have a good chance of making a better prediction (note how the fitted line (purple) is closer to the very accurate fit from the previous slide (red) than individual points

```
xs=c(4:6,8)
ys=apply(c(4:6,8),function(x) { sample(mpg$cty[mpg$cyl==x],size=1)})
plot(xs,ys,pch=19)
abline(lm(cty~cyl,data=mpg),col='red')
abline(lm(ys~xs),col="purple")
summary(lm(ys~xs))
...
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  30.6286     2.8987   10.566  0.00884 **
xs           -2.4571     0.4882   -5.033  0.03729 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.444 on 2 degrees of freedom
Multiple R-squared:  0.9268, Adjusted R-squared:  0.8902
sapply(4:8,function(x) { sd(mpg$cty[mpg$cyl==x]) })
[1] 3.4981918 0.5773503 1.7735296      NA 1.8063610
```



# OUTLINE

- Multiple regression
- Algae dataset example
- Resampling
  - Cross-validation
  - Bootstrap
- Bias-variance decomposition
- K-nearest neighbors regression

# CROSS-VALIDATION

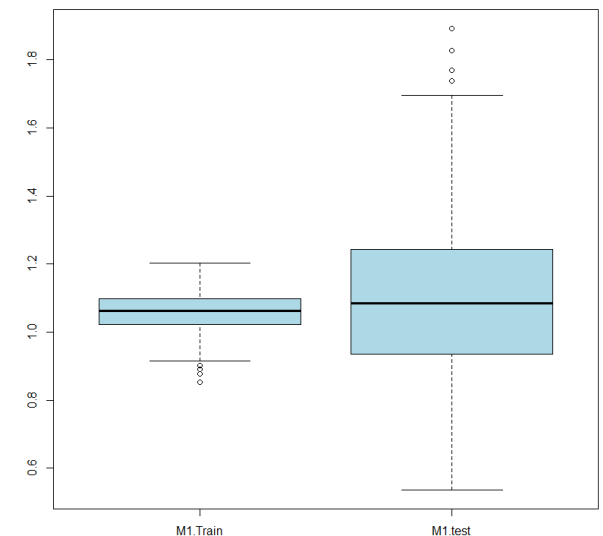
- We can always improve fit by adding more variables to the model
- But as the result we will eventually start overfitting (=fitting particular realization of the noise)
- Solution: apply the model to a previously unseen test set, calculate MSE to characterize prediction accuracy
  - Test set is rarely available.
  - But we can put aside part of the training data!
- We can split the data in many possible ways, however
  - Putting aside too much data will leave us with little data to train the model on. We will likely get a poor fit (except for relatively rare cases when we have *so many* data points that we can afford setting aside e.g. half of them)
  - Another extreme: **leave-one-out** cross-validation: set aside just one data point. Fit the rest of the data, calculate prediction error on the single test point... Now our fit does not suffer, but we don't have enough data to test on! Solution: repeat *many* times (up to dataset size  $N$ ), calculate MSE across multiple retrials. Can be computationally demanding.
  - Leave-two-out, leave-3-out, ...
  - Middle ground: set aside a small(-ish) chunk of data (e.g.  $1/4$ ,  $1/5$ ,  $1/10$ , etc): if we train, e.g., on  $4/5$  of the data, test on  $1/5$ , repeat 5 times – this is **5X cross-validation**

# CROSS-VALIDATION: EXAMPLE

- Let us try K-X cross-validation of our models. Let us write a script that reruns K-X cross-validation N times and generates distributions of train (fit accuracy) and test (prediction accuracy) MSEs.

```
N.xval=5
mse.test=numeric()
mse.train = numeric()
for ( iter in 1:100 ) {
  residuals=numeric()
  # randomy assign groups 1-Nxval to observations:
  grps= sample( (1:nrow(a)) %% N.xval+1 )
  for ( i in 1:N.xval ) { # for each group:
    data.test = a[grps==i,] #set it as test
    data.train = a[grps != i,] # set the rest as train
    M=lm(LAG1~LC6+LC8, data=data.train,na.action="na.exclude")
    M.predicted=predict(M,newdata=data.test)
    r=M.predicted-data.test$LAG1
    mse.test = c( mse.test, sum(r^2,na.rm=T)/sum(!is.na(r)) )
    mse.train = c( mse.train,
      sum(M$residuals^2,na.rm=T)/sum(!is.na(M$residuals)) )
  }
}

boxplot(list(M1.Train=mse.train,M1.test=mse.test),col='lightblue')
```



# FORMULA AND MODEL OBJECTS

- It would be helpful to have an `xval()` function defined that can take a formula and data and perform cross-validation. Otherwise we'd need to copy-paste the code from the previous slide, just changing the model every time
- Alternatively, since model object remembers how the model was build, maybe we can pass models as parameters too?

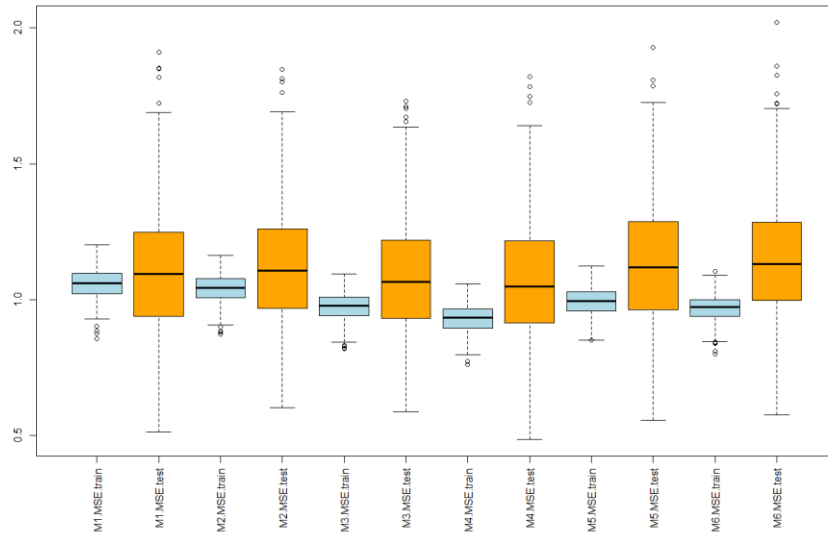
■ Let's investigate:

```
> mode (AG1~LC6+LC8)
[1] "call"
> class (AG1~LC6+LC8)
[1] "formula"
> names (M1)
[1] "coefficients" "residuals" "effects" "rank" "fitted.values" "assign" "qr"
[8] "df.residual" "na.action" "xlevels" "call" "terms" "model"
> M1$call
lm(formula = LAG1 ~ LC6 + LC8, data = a)
> as.list(M1$call)
[[1]]
lm
 $formula
LAG1 ~ LC6 + LC8
 $data
a
```

# CODE REUSE

- Now we can write our function:

```
lmse= c( xval(M1,a), xval(M2,a),  
        xval(M3,a), xval(M4,a),  
        xval(M5,a), xval(M6,a))  
boxplot(lmse,col=c('lightblue',  
                  'orange'),las=3)
```

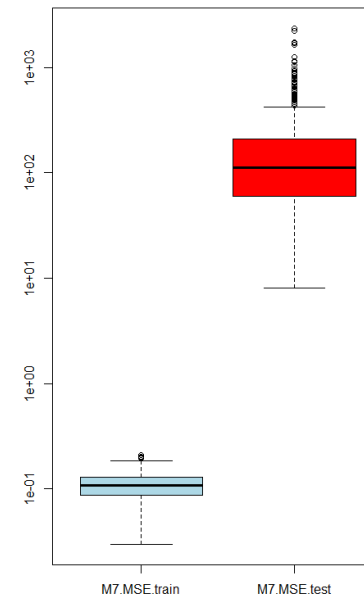


```
xval = function(x,data,N.xval=5,N.iter=100,prefix=substitute(x)) {  
  if ( class(x) == "formula" ) f = x  
  else f = as.list(x$call)$formula  
  mse.test=numeric(N.iter*N.xval)  
  mse.train = numeric(N.iter*N.xval)  
  k=1  
  for ( iter in 1:N.iter ) {  
    grps= sample( (1:nrow(data)) %% N.xval+1 )  
    for ( i in 1:N.xval ) { # for each group:  
      data.test = data[grps==i,] #set it as test  
      data.train = data[grps != i,] # set the rest as train  
      M=lm(f, data=data.train,na.action="na.exclude")  
      M.predicted=predict(M,newdata=data.test)  
      r=M.predicted-data.test$LAG1  
      mse.test[k] = sum(r^2,na.rm=T)/sum(!is.na(r))  
      mse.train[k] = sum(M$residuals^2,na.rm=T)/sum(!is.na(M$residuals))  
      k=k+1  
    }  
  }  
  l = list(MSE.train=mse.train,MSE.test=mse.test)  
  if ( ! is.null(prefix) && prefix != "" ) {  
    names(l) = paste(prefix,c(".MSE.train",".MSE.test"),sep="")  
  }  
  return( l )  
}
```

# ARE ALL MODELS THE SAME?

- Across a few models we have considered, the performance is quite comparable. What is wrong?
- In fact, the models were chosen very carefully
  - There is large degree of correlation between the predictor variables (as we observed), so it's not too surprising that a few different combinations of predictors result in comparable fits (note that all of them are far from perfect,  $R^2 \sim 50\%$ )
  - See what happens if we choose some really large model
  - In the model shown below,  $R^2$  is  $\sim 84\%$  !! (but adjusted  $R^2$  is still at  $\sim 48\%$ , which already indicates a problem!)

```
M7=lm(LAG1~C1*C2*LC3*C4*LC5*LC6*LC8,data=a)  
boxplot(xval(M7,a),col=c('lightblue','red'),log="y")
```





# BOOTSTRAPPING

- An alternative approach to cross-validation is offered by method known as ‘Bootstrapping’.
  - General idea is very similar: we want to resample from the actual data, train on one part and test on what was set aside
  - It is critical for bootstrap that resampling is performed *with replacement*.
  - This allows to better adjust for the effect of outliers in the data; in particular, bootstrap is often used for numerical estimation of confidence intervals.
- Let us illustrate bootstrapping with a trivial example:
  - We are going to determine the standard error of the sample mean (which is directly related to the confidence interval)
  - In order to achieve that, we will simulate a random sample – that’s the “observed data” available to us (nothing else is available as it is usually the case in real life)
  - We will resample with replacement from the data we have. See next slide

# BOOTSTRAPPING THE SAMPLE MEAN

```
# random sample from normal; this is our "measured data":
> x <- rnorm(100)
> v <- numeric(10000) # reserve the vector of length 10K
> for ( n in 1:10000 ) { # resamplings: bootstrap 10K times
  # new sample x1: resampling with replacement; the size of the sample,
  # and thus its statistical power, is going to be the same, but due
  # to replacement a few randomly chosen points from the original data
  # will be dropped and replaced by additional copies of other points
  x1 <- sample(x,replace=T)
  v[n] <- mean(x1) # calculate the mean of bootstrapped sample
}
> sd(v) # standard deviation of our resampled mean estimates (SEM)
[1] 0.09651252
> sd(x)/10
[1] 0.09663202
```

# BOOTSTRAPPING STATISTICAL MODELS

- In the latest example, by resampling from the *SAME* sample (no other data is available to us!), with replacement, we reproduce correct SEM with 0.1% accuracy!
  - Lots of information about how samples fluctuate from one experiment to another is already hidden in the sample we hold – if we sample with replacement
- In the framework of fitting statistical models, the approach is very similar:
  - Resample (*with replacement!*) new dataset  $D_{boot}$  from the training dataset  $D_{train}$ .
  - With each resampling, some training data points from  $D_{train}$  are going to be left unused (because of replacement), these will be used as  $D_{test}$ , the test set
  - Fit the model on  $D_{boot}$ , calculate MSE on  $D_{test}$
  - Repeat bootstrap resampling many times
  - Enjoy the expected distribution of MSE on the test set (i.e. true predictive accuracy of the model)

# OUTLINE

- Multiple regression
- Algae dataset example
- Resampling
  - Cross-validation
  - Bootstrap
- Bias-variance decomposition
- K-nearest neighbors regression

# BIAS-VARIANCE DECOMPOSITION / TRADEOFF

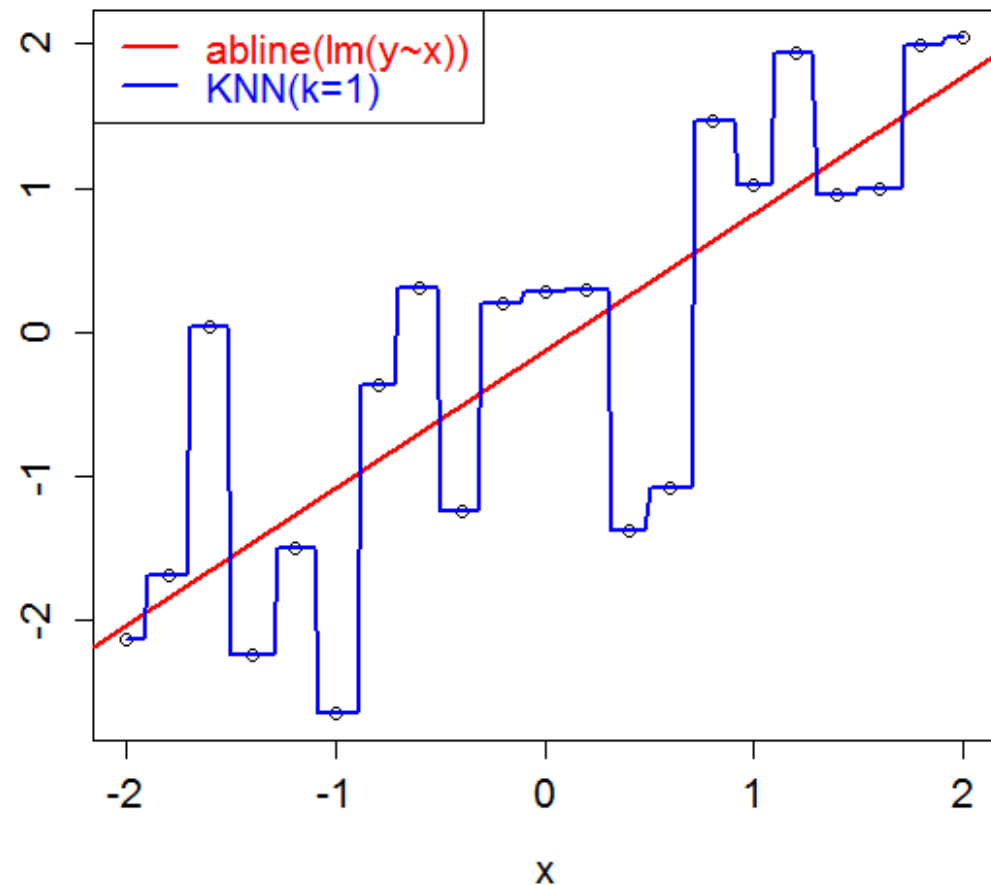
- Bias-variance decomposition (cf. Eq.2.7 in ISLR) taking some shortcuts for simplicity (see ESL for more details):
$$E[(Y - \hat{Y})^2] = E[\{f(X) + \varepsilon - E(\hat{Y}) + E(\hat{Y}) - \hat{Y}\}^2] = E[(E(\hat{Y}) - \hat{Y})^2] + E[(f(X) - E(\hat{Y}))^2] + \text{Var}(\varepsilon) = \text{Var}(\hat{Y}) + [\text{Bias}(\hat{Y})]^2 + \text{Var}(\varepsilon)$$
- $E[(E(\hat{Y}) - \hat{Y})^2]$  – variance: variability of estimated response over multiple training sets
- $E[(f(X) - E(\hat{Y}))^2]$  – bias: systematic difference between true dependency  $f(X)$  and average estimated outcome  $E(\hat{Y})$
- $\text{Var}(\varepsilon)$  – irreducible error due to stochastic nature of  $Y$
- High variance here equates to large variability of estimated outcome across training sets
  - Typical for higher complexity models (highly non-linear, many parameters, etc.)
- High bias implies large systematic differences between true dependency and the model
  - E.g. linear model fit to non-linear data, or, setting it to a constant (average outcome in the training data)
- Typical tuning of a statistical learning approach amounts to finding optimal (in context specific sense) bias-variance tradeoff
  - I.e. finding model complexity that minimizes summary contribution of bias and variance to test error

# OUTLINE

- Multiple regression
- Algae dataset example
- Resampling
  - Cross-validation
  - Bootstrap
- Bias-variance decomposition
- K-nearest neighbors regression

# NON-PARAMETRIC APPROACHES: KNN

- Linear model represents parametric approach
  - assumes linear dependency between outcome and predictors governed by corresponding parameter estimates
- Great if linear form is close to true  $f(x)$ , but what if not?
  - Can we just "use data" to "make predictions"? Which data? How much? What does it mean to "make predictions"?
- KNN (K nearest neighbors; not K-means!) provides one set of answers to these questions and exemplifies non-parametric approach:
  - for a given point  $X_0$ , predicted outcome  $Y_0$  is the average of K outcomes from K points closest to  $X_0$  in the space of  $X$
- Also, excellent illustrations and discussion in ISLR Ch.3.5



# CODE FOR THESE TWO FIGURES

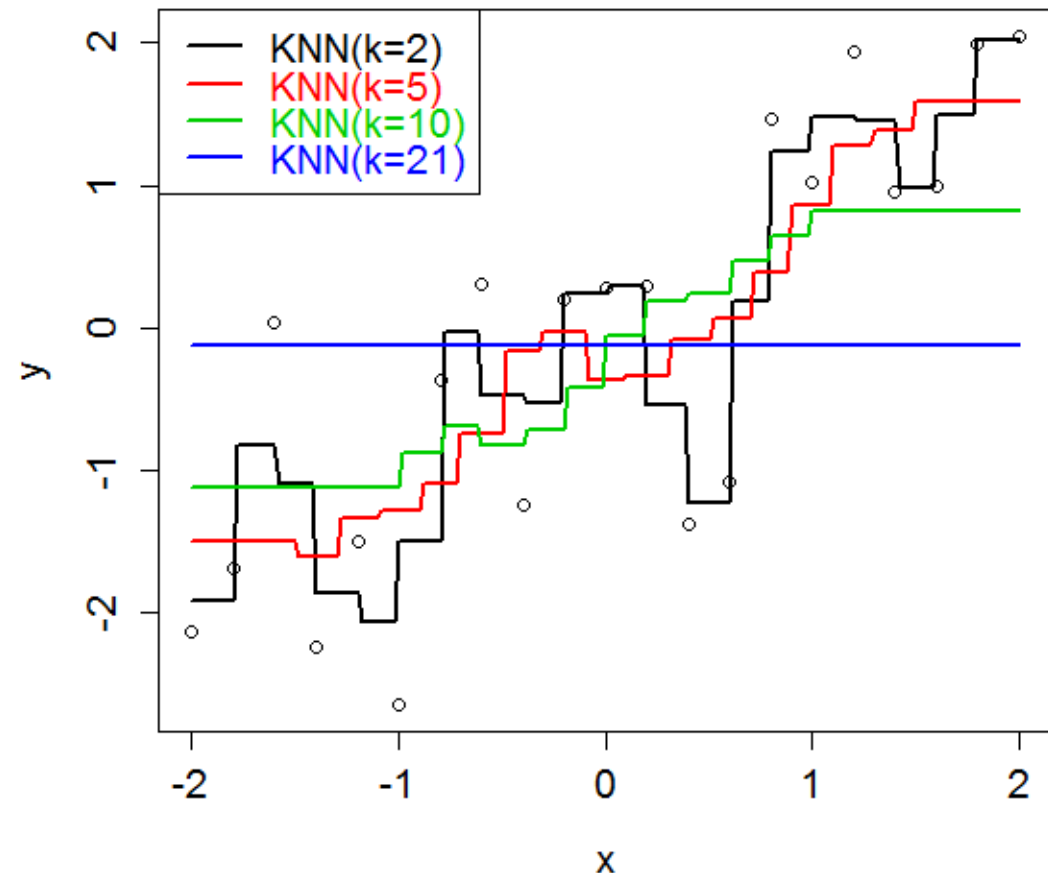
```
library(FNN)
x <- (-10:10)/5
y <- x+rnorm(length(x))
# previous figure:
plot(x,y)
abline(lm(y~x),lwd=2,col="red")
xNew <- (-100:100)/50
knn1RegRes <- knn.reg(as.matrix(x),as.matrix(xNew),y,k=1)
points(xNew,knn1RegRes$pred,lwd=2,col="blue",type="l")
legend("topleft",c("abline(lm(y~x))","KNN(k=1)"),col=c("red","blue"),text.col=c("red","blue"),lwd=2)
# next figure:
plot(x,y)
tmpKvals <- c(2,5,10,length(x))
for ( iTmp in 1:length(tmpKvals) ) {
  knnRegTmp <- knn.reg(matrix(x),matrix(xNew),y,k=tmpKvals[iTmp])
  points(xNew,knnRegTmp$pred,lwd=2,col=iTmp,type="l")
}
legend("topleft",paste0("KNN(k=",tmpKvals,")"),col=1:length(tmpKvals),text.col=1:length(tmpKvals),lwd=2)
```

There is R library for that!

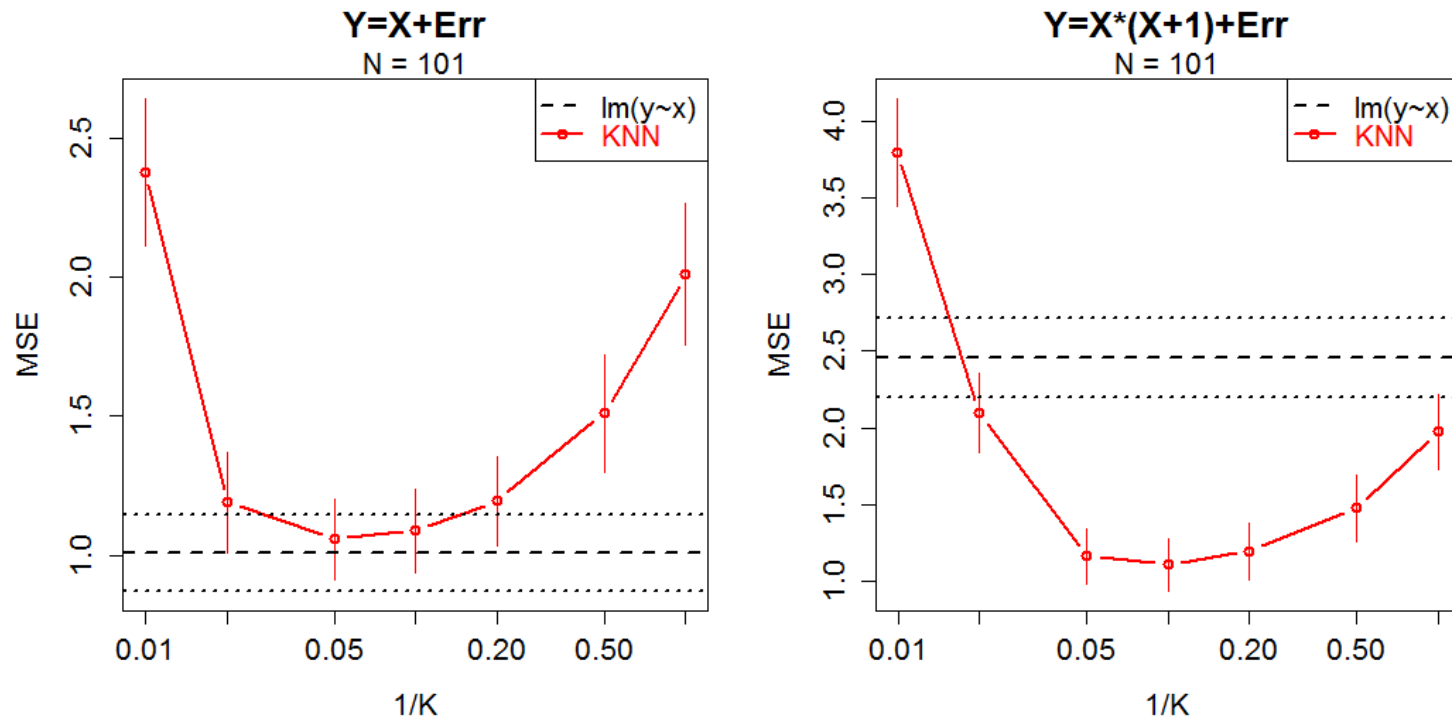


# KNN (CONTINUED)

- $K$  – number of nearest neighbors used to estimate outcome – governs model variability and bias
- Low  $K$  means estimated outcome follow very closely observed data
  - High model complexity: low bias, high variance
- High  $K$  (cannot be higher than training sample size though) means low variance and high bias
  - Low model complexity
- For fixed  $X$  in our simple model -  $Y=X+\varepsilon$ ,  $\varepsilon \sim N(0,1)$  – for  $K=n$ ,  $\text{Var}(\hat{Y})=1/n$ , while for  $K=1$ ,  $\text{Var}(\hat{Y})=1$
- Changing  $K$  from 1 to  $n$  provides obvious and intuitive means to varying model complexity and bias-variance tradeoff
- When does it make a difference?



# KNN: BIAS VARIANCE TRADEOFF



- Lowest MSE is achieved (in these examples) at intermediate level of model complexity ( $1 < K < N$ )
- When linear model adequately represents true  $Y=f(X)$  relationship it is competitive with KNN
- When linear model is inadequate, KNN can achieve lower MSE (not always, depends on dimensionality, see ISLR)

# CODE FOR PREVIOUS PLOTS

```
old.par <- par(mfrow=c(1,2),ps=16)
for ( iPow in 1:2 ) {
  x <- (-50:50)/25
  ymean <- x
  if ( iPow == 2 ) {
    ymean <- ymean + x*x
  }
  lmMSEtmp <- numeric()
  tmpKvals <- c(1,2,5,10,20,50,length(x))
  knnMSEtmp <- matrix(NA,nrow=length(tmpKvals),ncol=100)
  for ( iSim in 1:ncol(knnMSEtmp) ) {
    y <- ymean+rnorm(length(x))
    lmTmp <- lm(y~x)
    ytest <- ymean+rnorm(length(x))
    ypredlm <- predict(lmTmp)
    lmMSEtmp[iSim] <- mean((ypredlm - ytest)^2)
    for ( iKtmp in 1:length(tmpKvals) ) {
      knnTmp <- knn.reg(matrix(x),matrix(x),y,k=tmpKvals[iKtmp])
      knnMSEtmp[iKtmp,iSim] <- mean((knnTmp$pred-ytest)^2)
    }
  }
}
```

```
knnMSEaveTmp <- apply(knnMSEtmp,1,mean)
knnMSEsdTmp <- apply(knnMSEtmp,1,sd)
xTmpLim <- range(1/tmpKvals)
yTmpLim <- range(c(mean(lmMSEtmp)+c(-1,1)*sd(lmMSEtmp),knnMSEaveTmp-
knnMSEsdTmp, knnMSEaveTmp+knnMSEsdTmp))
plot(xTmpLim, yTmpLim, type="n", log="x", xlab="1/K", ylab="MSE",
main=c("Y=X+Err","Y=X*(X+1)+Err")[iPow])
points(1/tmpKvals,knnMSEaveTmp,type="b",col="red",lwd=2)
for ( iKtmp in 1:length(tmpKvals) ) {
  points(c(1,1)/tmpKvals[iKtmp], knnMSEaveTmp[iKtmp]+c(-
1,1)*knnMSEsdTmp[iKtmp], type="l", col="red")
}
abline(h=mean(lmMSEtmp)+c(-1,0,1)*sd(lmMSEtmp),lty=c(3,2,3),lwd=2)
mtext(paste("N =",length(x)))

legend("topright",c("lm(y~x)","KNN"),col=1:2,text.col=1:2,lwd=2,lty=c(2,1),pch
=c(NA,1))
}
par(old.par)
```

# SUMMARY

- Multiple linear regression: polynomial terms, multiple predictor variables, interactions
- Example: algae dataset
  - Outliers
  - Data distribution and variable transformations
  - R: pretty correlation plots
  - Fitting many multiple regression models (with interactions too)
  - Assessing the (comparative) quality of multiple models:  $R^2$ , significance, and importantly:
- K-fold Cross-validation
- Bootstrap (you will learn more and practice in your homework assignment!)
- Bias-variance tradeoff: something's gotta give
- K-nearest neighbors: non-parametric approach with built-in variance-tradeoff dial
- Philosophical statement :What data do tell us and what they don't: go with the data