Notes for Math 253: Statistical Computing and Machine Learning

Daniel Kaplan

Fall 2016, Macalester College

Contents

Preface

Math 253 and the Macalester statistics curriculum

Math 253, Statistical Computing and Machine Learning, is a course at Macalester College. The course comes early in our curriculum for the statistics major. Currently, that curriculum looks like:

1. Comp 110, Data Computing, a no-prerequisite course in data wrangling and visualization. The main tools in the course are R, dplyr, and ggplot.
2. Math 125, Epidemiology. This course also has no pre-requisites. It's designed to teach quantitative and statistical literacy in the context of public health and decision making. Unlike the other courses in our curriculum, Math 125 is *not* computer intensive. This is an *elective* for the statistics major.
3. Math 155, Introduction to Statistical Modeling. This is our main entry-level statistics course. It also has no pre-requisites, but we encourage students to take a course in our calculus sequence: Applied Multivariate Calculus I, II, and III. The course covers important concepts in modeling: model architectures and fitting using mainly linear and logistic regression, covariation and adjustment, interpretation of model coefficients, inference techniques such as analysis of variance and co-variance and the ways these techniques can inform the decisions needed to build useful models, causality including techniques for making reasonable conclusions about causation from observational data. The course makes very extensive and intensive use of R and the `mosaic` package for R.
4. This course, **Math 253, Statistical Computing and Machine Learning**. Math 253 introduces a broader set of model architectures (e.g. those associated with "machine learning" such as support vector machines) and the trade-offs that make machine learning a human skill rather than a push button mechanism. The main text is Introduction to Statistical Learning in R. Computing in R is used intensively in the course. Math 155 is a pre-requisite. The computing used in that course is the only pre-requisite. A small part of Math 253 is given to basic programming in R, but the large majority is about the mathematical and statistical concepts involved in learning from data and the exercise of a variety of architectures for classification and regression models.

   Math 253 is placed intentionally in the middle of our statistics curriculum to make it accessible to many students who are interested and may have use for the techniques, but whom are not primarily interested in statistics *per se*.
5. Upper level courses including:
   - Math 454: Bayesian statistics
   - Math 453: Biostatistics
   - Math 454: Mathematical statistics
6. Supporting courses including:
   - Math 354: Probability
   - Math 135: Applied Multivariate Calculus I
   - Math 236: Linear algebra
   - Comp 123: Core conceptsin computer science

7. Electives such as
    - Math 432: Mathematical modeling
    - Math/Comp 365: Numerical linear algebra
    - Comp 302: Introduction to database management systems

These notes are written in Bookdown

The document uses an elaboration on R/Markdown, called "Bookdown." I don't yet know if this will be a good way to maintain and distribute class notes.

Some advantages:

- All the notes are in one place.
- Students and other instructors can clone the notes for their own uses.
- People spotting mistakes can contribute corrections via the "pull request" mechanism supported by Bookdown working with GitHub.
- Multiple instructors can contribute to the notes via GitHub. This is the stated purpose behind Bookdown — allowing book-length publications to be authored by many authors.

Disadvantages:

- The notes are *always* a work in progress. Don't be mislead by the polish that RMarkdown and Bookdown give to the notes.

Chapter 1

Introduction

- Subjects
  - Overview of statistical learning.
  - Getting started with R, RStudio, and RMarkdown
- Reading: Chapter 2 of ISL
- Programming basics 1: Names, classes, and objects

1.1 Statistical and Machine Learning

The two terms, "statistical learning" and "machine learning," reflect mainly the artificialities of academic disciplines.

- Statisticians focus on the statistical aspects of the problems
- Computer scientists are interested in "machines", including hardware and software.

"Data Science" is a new term that reflects the reality that both statistical and machine learning are about data. Techniques and concepts from both statistics and computer science are essential.

1.1.1 Example 1: Machine translation of natural languages

Computer scientists took this on.

- Identification of grammatical structures and tagging text.
- Dictionaries of single-word equivalents, common phrases.

Story from early days of machine translation:

- Start with English: "The spirit is willing, but the flesh is weak."
- Translate into Russian
- Translate back into English. Result: "The vodka is good, but the meat is rotten."

Statistical approach:

- Take a large sample of short phrases in language A and their human translation into language B: the dictionary
- Find simple measures of similarity between phrases in language A (e.g. de-stemmed words in common)
- Take new phrase in language A, look up it's closest match in the dictionary phrases in language A. Translation is the corresponding dictionary entry in language B

Where did the sample of phrases come from?

- European Union documents routinely translated into all the member languages. Humans mark correspondence.
- "Mechanical Turk" dispersal of small work tasks.

Result: Google translate.

1.1.2 Example 2: From library catalogs to latent semantic indexing

Early days: computer systems with key words and search systems (as in library catalogs)

Now: dimension reduction (e.g. singular value decomposition), angle between specific documents and what might be called "eigen-documents"

Result: Google search

1.1.3 Computing technique

Each student in the class as a personal repository on GitHub. The instructor is also a contributor to this repository and can see anything in it. Complete instructions for doing this are in the appendix.

1. Set up some communications and security systems (e.g. an RSA key)
2. Clone your repository from GitHub. It is at an address like `github.com/dtkaplan/math253-bobama`.

Day 1 Programming Activity

1.2 Review of Day 1

We discussed what "machine learning" means and saw some examples of situations where machine-learning techniques have been used successfully to solve problems that had at best clumbsy solutions before. (Natural language translation, catalogs of large collections of documents.)

We worked through the process of connecting RStudio to GitHub, so that you can use your personal repository for organizing, backing up, and handing in your work.

The Day-1 programming activity introduced some basic components of R: assignments, strings, vectors, etc.

1.3 Theoretical concepts ISL §2.1

"Data science" lies at the intersection of statistics and computer science.

1.3.1 Statistics concepts

- Sampling variability
- Bias and variance
- Characterization of precision
- Function estimation frameworks, e.g. generalized linear models
- Assumed probability models
- Prior and posterior probabilities (Bayesian framework)

1.3.2 Computing concepts

- Algorithms
- Iteration
- Simulation
- Function estimation frameworks, e.g. classification trees, support vector machines, artificial intelligence techniques
- Kalman filters

1.3.3 Cross fertilization

- Assumed probability models supplanted by simulation
  - Randomization and iteration
  - Cross validation
  - Bootstrapping
- Model interpretibility Rather than an emphasis on the output of a function, interest in what the function has to say about how the world works.

1.4 Many techniques

"Learning" is an attractive word and suggests that "machine learning" is an equivalent for what humans do. Perhaps it is to some extent …

But "modeling" is a more precise term. We will be building models of various aspects of the world based on data.

- Model: A representation for a purpose. Blueprints, dolls, model airplanes.
- Mathematical model: A model built of mathematical stuff
  - polynomials: Math 155
  - functions more generally: e.g. splines, smoothers, …
  - trees
  - geometry of distance: e.g. which exemplar are the inputs closest to?
  - projection onto subspaces
- Statistical model: A mathematical model founded on data.

1.4.1 Unsupervised learning

Wait until the end of the semester.

We will be doing only supervised learning until late in the course.

1.4.2 Supervised learning:

- We have a set of cases, $i = 1, 2, 3, \ldots, n$, called the **training data**.
- For each case, we have an input $\mathbf{X}_i$ consisting potentially of several variables measured on that case.
  - The subscript i in $\mathbf{X}_i$ means that we have one $\mathbf{X}$ for each case. The boldface $\mathbf{X}$ means that the input can be multi-dimensional, that is, consisting of multiple variables.

- For each case, we have an output Y_i.
- We want to learn the overall pattern of the relationship between the inputs $\mathbf{X}$ and the outputs Y, not just for our n training cases, but for potential cases that we have not encountered yet. These as yet unencountered cases are thought of as the **testing data**.

- We are going to represent the pattern with a **function** $\hat{f} : \mathbf{X} \to Y$.

- Sometimes I'll use the word **model** instead of function. A model is a representation for a purpose. A function is a kind of representation. So some models involve functions. That's the kind we'll focus on in this course. I say "model" out of habit, but it's a good habit that reminds us that the **purpose** of the function is important and we should know what that purpose is when we undertake learning.

1.5 Basic dicotomies in machine learning

There are fundamental trade-offs that describe the structure of learning from data. There are also trade-offs that arise between different methods of learning. Finally, there are dicotomies that stem from the different purposes for learning.

These dicotomies provide a kind of road map to tell you where are are and identify where you might want to go.

And, as always, it's important to know why you are doing what you're doing: your purpose.

1.5.1 Purposes for learning:

- Make predictions. Given new inputs, e.g. data about an event, predict what the result of the event will be. e.g. weather forecasting, credit card fraud, success in college, …. In statistics, this is sometimes thought of as "analyzing data from an observational study."
- Anticipate the effects of an intervention that we impose, e.g., giving a patient a drug, changing funding for schools, … Traditionally in statistics, this has been tied exclusively to data from experiments. There is now greater acceptance that experiments are not always possible, and it's important to be able to make reasonable inferences about causation from observational studies.
- Find structure in a mass of data.

1.5.2 Dicotomies

- make predictions vs capture causal mechanism (in this course: common sense. There are also formal techniques to guide causal reasoning.)
- flexibility vs variance (need some tools for this)
- black box vs interpretable models (comparing model architectures)
- reducible vs irreducible error ("bias" vs "residuals")
- regression vs classification (easy!)
- supervised vs unsupervised learning (easy!)

1.5.3 Prediction versus mechanism

Example: Malignancy of cancer from appearance of cells. Works for guiding treatment. Does it matter why malignant cells have the appearance they do?

Story: Mid-1980s. Heart rate variability spectral analysis and holter monitors. (Holters were cassette tape recorders set to record ECG very, very slowly. Spectral analysis breaks down the overal signal into periodic components.) Very large spike at 0.03 Hz seen in people who will soon die.

Could use for prediction, but researchers were also interested in the underlying physiological mechanism. Causal influences. We want to use observations to inform our understanding of what influences what.

Story continued: The very large spike was the "wow and flutter" in the cassette tape mechanism. This had an exact periodicity: a spike in the spectrum. If the person was sick, their heart rate was steady:

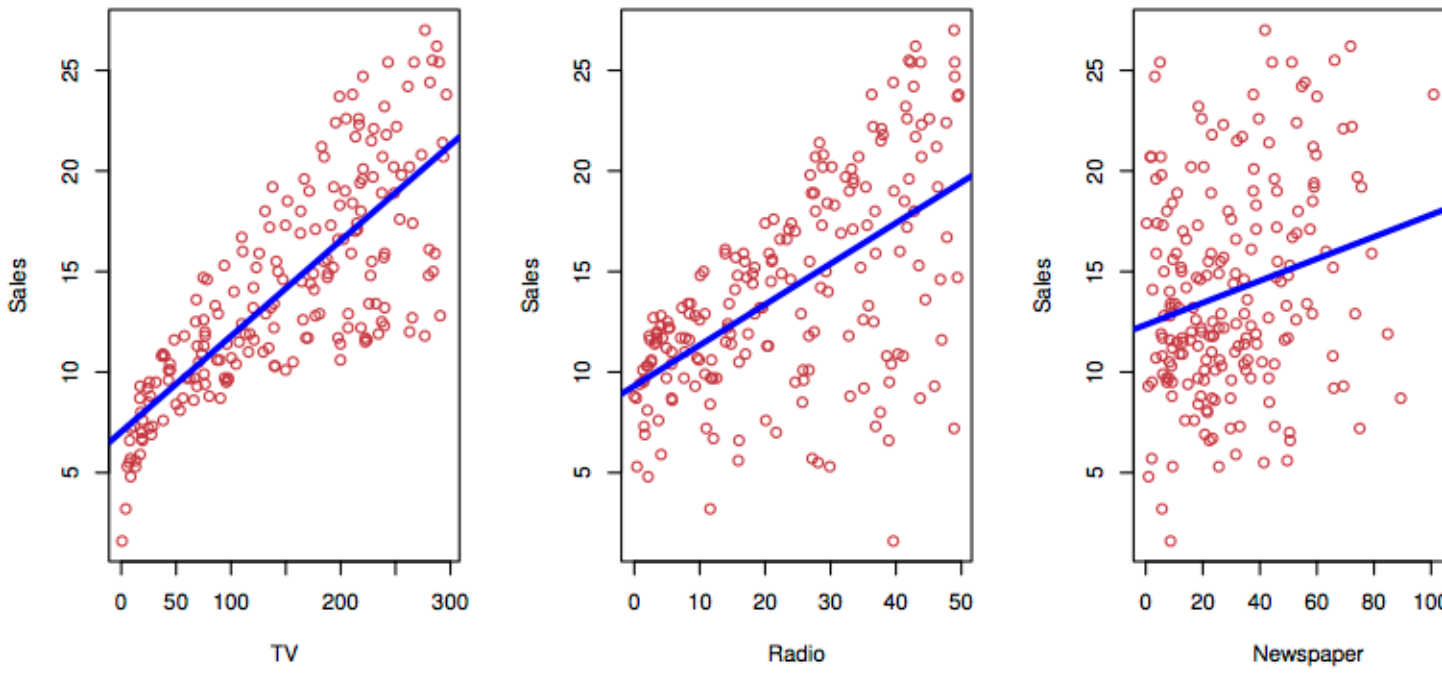


Figure 1.1: Individual fits miss how the explanatory variables interact. ISL Figure 2.1

they had no capacity to vary it as other conditions in the body (arterial compliance, venus tone) called for. Understanding what happens in cardiac illness is, in part, about understanding how the various control systems interact.

1.5.4 Flexibility versus variance

In traditional statistics, this is often tied up with the concept of "degrees of freedom."

Not flexible:

Flexible:

And in multiple variables:

Not flexible:

Flexible:

1.5.5 Black box vs interpretable models

Many learning techniques produce models that are not easily interpreted in terms of the working of the system. Examples: neural networks, random forests, etc. The role of input variables is implicit. Characterizing it requires experimenting on the model. In other learning techniques, the role of the various inputs and their interactions is explicit (e.g. model coefficients).

The reason to use a black-box model is that it can be flexible. So this tradeoff might be called "flexibility vs interpretability."

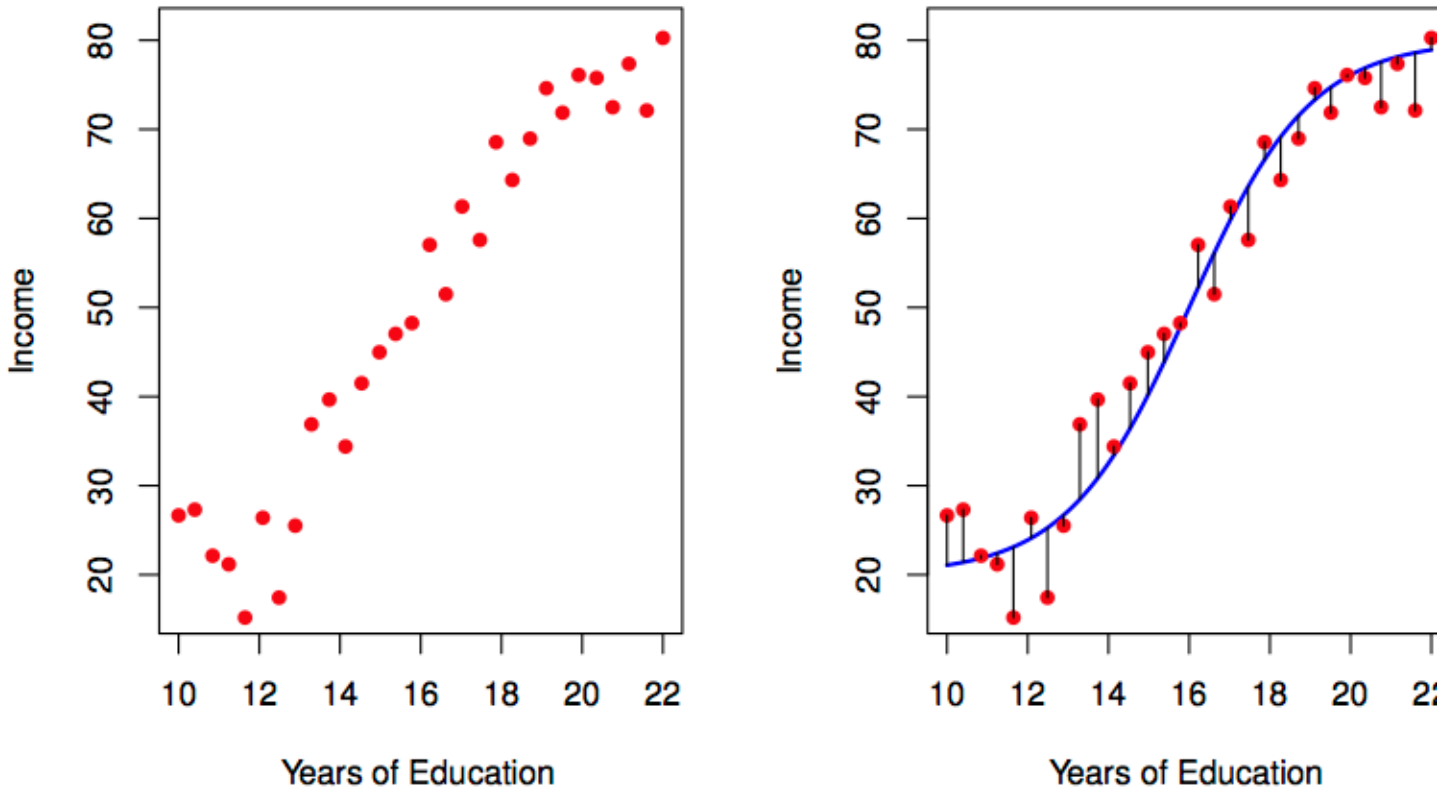


Figure 1.2: Such detailed patterns are more closely associated with physical science data than with social/economic data. ISL Figure 2.2

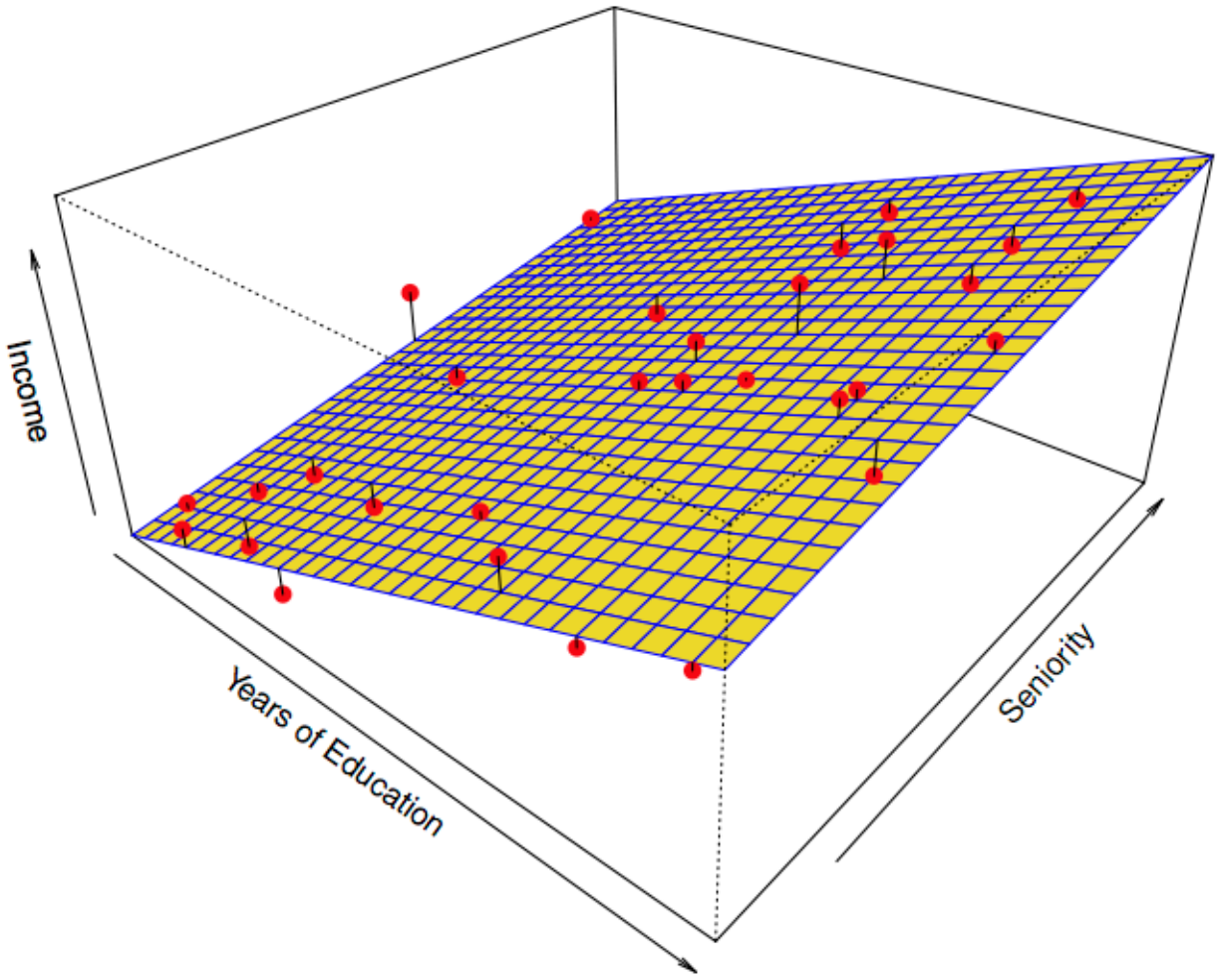


Figure 1.3: ISL Figure 2.4

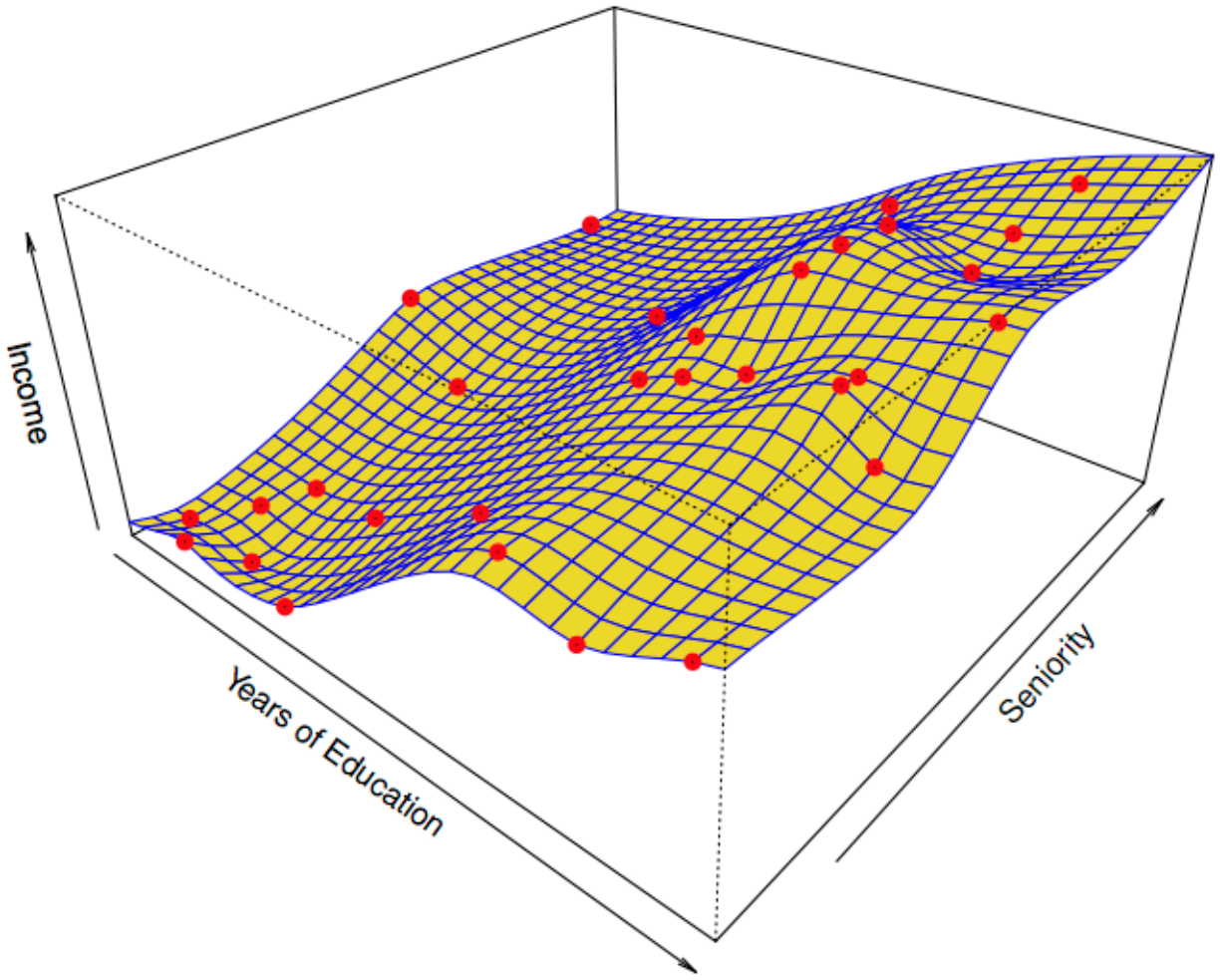


Figure 1.4: ISL Figure 2.6

A quick characterization of several model architectures (which they call "statistical learning methods")

1.5.6 Reducible versus irreducible error

How good can we make a model? How do we describe how good it is?

What does this mean? (from p. 19)

$$\begin{aligned} E(Y-\hat{Y})^2 &= E[f(X)+\epsilon-\hat{f}(X)]^2 \\ &= \underbrace{[f(X)-\hat{f}(X)]^2}_{Reducible} + \underbrace{Var(\epsilon)}_{Irreducible} \end{aligned}$$

Notation:

- X — the inputs that determine the output Y.
- Y — the output, that is, the quantity we want to predict
- $\hat{Y}$ — our prediction
  - hat means estimated, no hat means "real" (whatever that might mean)
- $E(Y-\hat{Y})^2$ — the mean of the square difference between our prediction and the "real" value. E means "expectation value."
- $f(X)$ — what Y would be, ideally, for a given X
- $\hat{f}(X)$ — our estimate of $f(X)$
- ϵ — but Y is subject to other, "random" influences. ϵ represents these. ϵ is a misleading notation because it may not be at all small in practice. But ϵ is alway centered on zero (by definition).
- $|f(X)-\hat{f}(X)|$ — the magnitude of the difference between the "real" $f()$ and our estimate. This can be made small by
  1. collecting more data
  2. using a more flexible model
  3. expanding the set of inputs considered
- $Var(\epsilon)$ — the "variance" of ϵ. This is the mean square of ϵ, that is, $E(\epsilon^2)$.

1.5.7 Regression versus classification

Regression: quantitative response (value, probability, count, …)

Classification: categorical response with more than two categories. (When there are just two categories, regression (e.g. logistic regression) does the job.)

1.5.8 Supervised versus unsupervised

- Demographics groups in marketing.
- Poverty vs middle-class
- Political beliefs … left vs right?

1.6 Programming Activity 1

Using R/Markdown

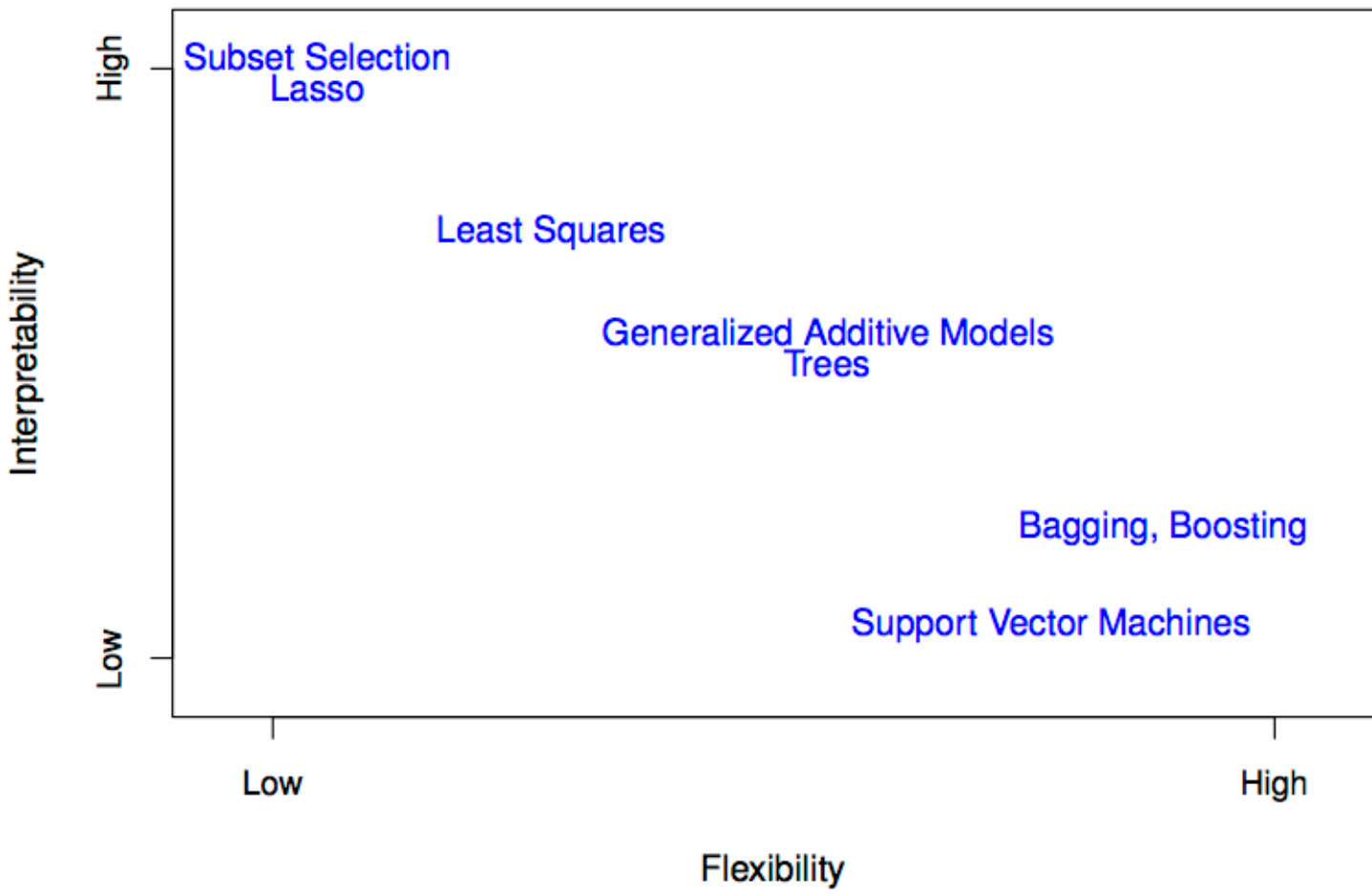


Figure 1.5: ISL Figure 2.7

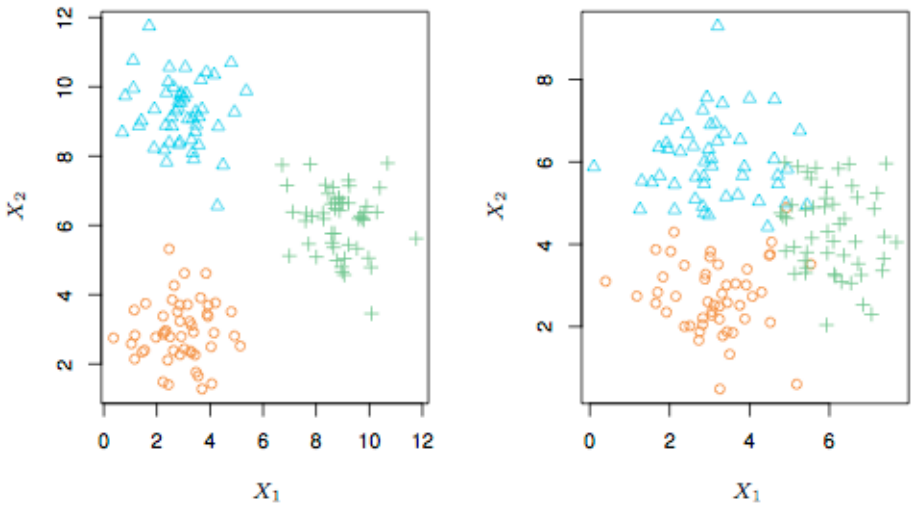


Figure 1.6: ISL Figure 2.8

1.7 Review of Day 2

1.7.1 Trade-offs/Dicotomies

- Regression vs classification
  - Different kinds of functions. A classifier has output as a categorical level. A regression has output as a number.
  - Many classifiers are arranged to produce as output a set of numbers: the probability of each of the possible levels of the categorical output. When there are just two such levels, only one probability is needed. (The other is simply $(1 - p)$.) So for two-level classifiers, there's not necessarily a distinction between regression and classification. Thus, "logistic regression."
- Supervised vs unsupervised learning
  - In supervised learning, with have an output (response variable) Y which we want to generate from input $\mathbf{X}$. We train a function $\hat{f} : \mathbf{X} \to Y$
  - In unsupervised learning, there is no identified response variable. Instead of modeling the response as a function of $\mathbf{X}$, we look for patterns within $\mathbf{X}$.
- Prediction vs causal mechanism
  - Two different kinds of purpose. There may well be different kinds of functions best suited to each purpose.
- Accuracy (flexibility) vs interpretability
  We always want models to be accurate. Whether we need to be able to interpret the model depends on our overall purpose.
- Reducible error vs irreducible error
  It's good to know how accurate our models can get. That gives a goal for trying out different types of models to know when we don't need to keep searching.

1.8 A Classifier example

A classification setting: Blood cell counts.

Build a machine which takes a small blood sample and examines and classifies individual white blood cells.

The classification is to be based on two measured inputs, shown on the x- and y-axes.

Training data has been developed where the cell was classified "by hand." In medicine, this is sometimes called the *gold standard*. The gold standard is sometimes not very accurate. Here, each cell is one dot. The color is the type of the cell: granulocytes, lymphocytes, monocytes, …

1.9 Programming Activity 2

Some basics with data

1.10 Day 3 theory: accuracy, precision, and bias

1.10.1 Figure 2.10

In constructing a theory, it's good to have a system you can play with where you know exactly what is going on: e.g. a simulation.

The dark blue line in the left panel is a function the authors created for use in a simulation:

The dots are data the textbook authors generated from evaluating the function at a few dozen values of x and adding noise to each result.

The difference between the dots' vertical position and the function value is the *residual*, which they are calling the *error*. The mean square error MSE is

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

- Take this notation apart. What's n? What's i?
- Suppose that $f(x)$ were constant. In that situation, what kind of statistical quantity does this resemble?
- In actual practice, we don't know $f(x)$. (Indeed, it's a matter of philosophy whether this is an $f(x)$ — it's a kind of Platonic form.) Here we know $f(x)$ because we are playing a game: running a simulation.

Looking again at the left panel in Figure 2.9, you can see three different functions that they have fitted to the data. It's not important right now, but you might as well know what these model architectures are:

1. Linear regression line (orange)
2. Smoothing splines (green and light blue). A smoothing spline is a functional form with a parameter: the *smoothness*. The green function is less smooth than the light blue function.
3. That smoothness measure can also be applied to the linear regression form

Each of these three functions were fitted to the data. Another word for fitted is *trained*. As such, we use the term *training error* for the difference between the data points and the fitted functions. Also, because the functions are not the Platonic $f(x)$, they are written $\hat{f}(x)$.

For each of the functions, the training MSE is

$$\text{Training MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

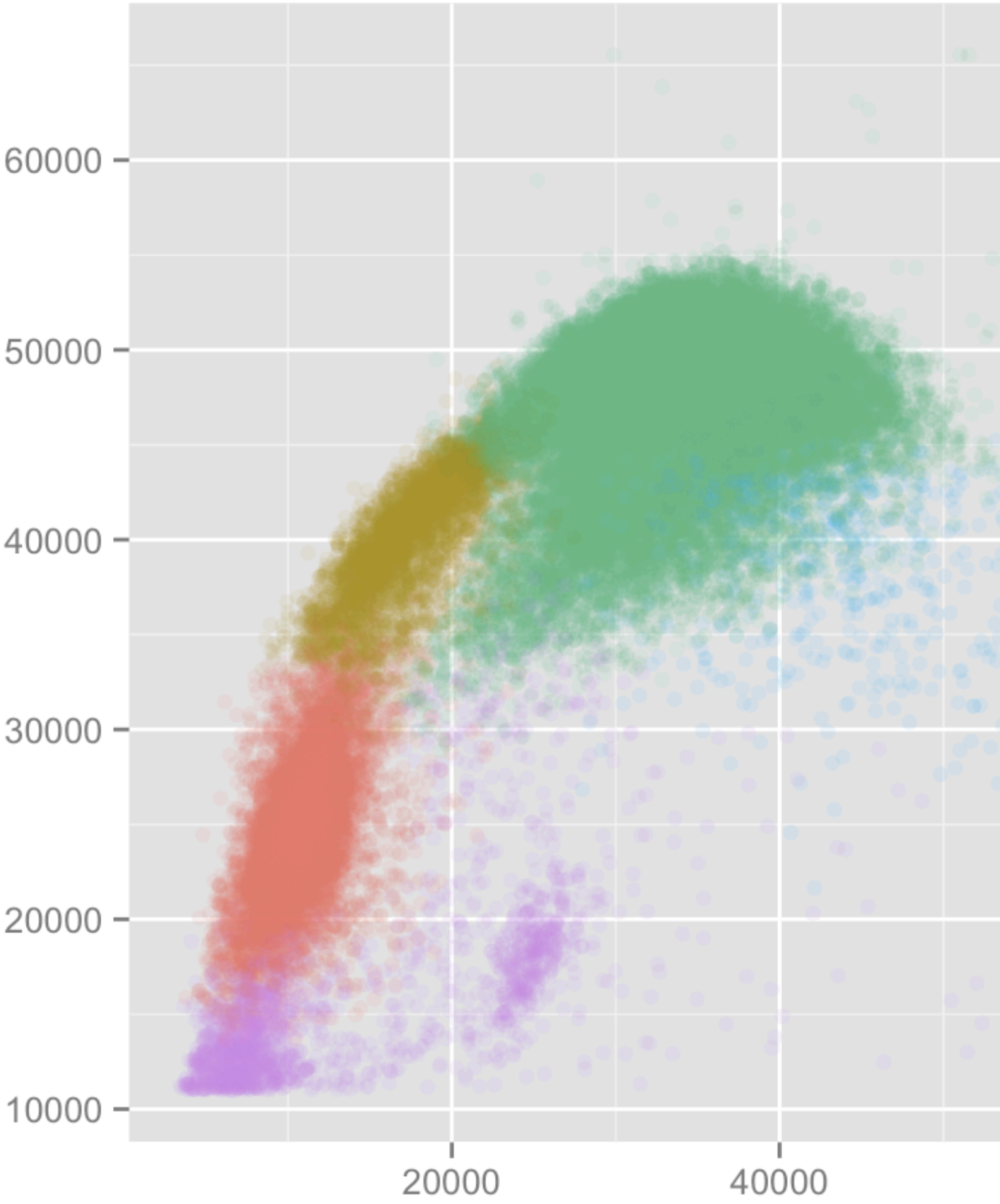


Figure 1.7: Blood cell classification

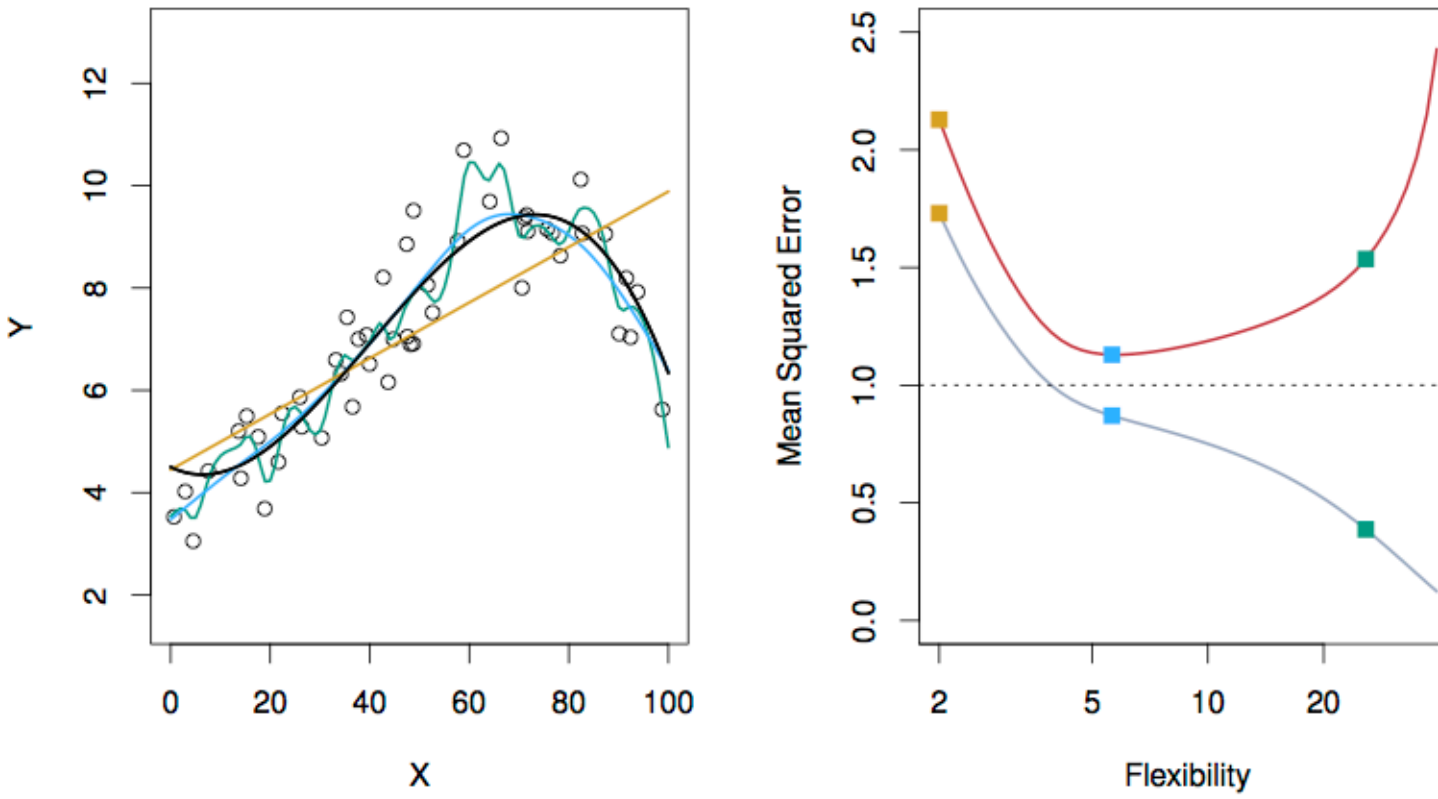


Figure 1.8: ISL Figure 2.9

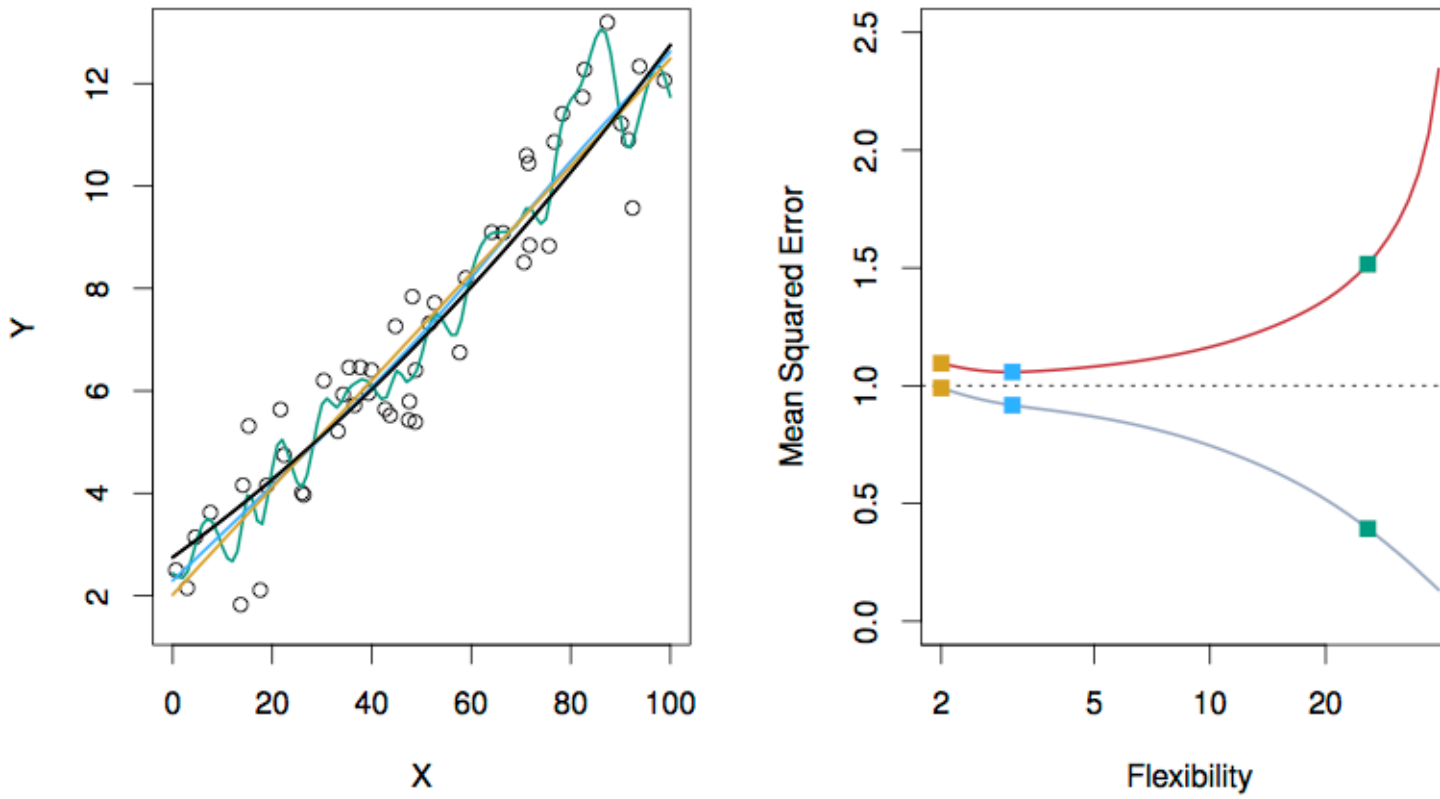


Figure 1.9: ISL Figure 2.10

Right panel of the graph is something completely different: both the axes are different than in the left panel.

- x-axis: the smoothness of the functions. This is labelled *flexibility*.
- The three x positions correspond to the smoothness of the three models. This is measured as the effective *number of parameters* of the function.
  Why does the straight-line function have a smoothness of 2?
- y-axis: the MSE
  - The dots connected by the gray curve show the *training* MSE of the three models.
  - The dots connected by the orange curve show the *testing* MSE of the three models.
  - The continuous curves were constructed by calculating the MSE for many more values of smoothness than shown in the left panel.
- How did they measure the *training* MSE?

1.10.2 Another example: A smoother simulated $f(x)$.

- What's different between the right panel of 2.9 and that of 2.10?

1.10.3 What's the "best" of these models?

When examining training MSE, the more flexible model has the smaller MSE. This answer is pre-ordained, regardless of the actual shape of the Platonic $f(x)$.

In traditional regression, we use ANOVA or *adjusted$ R^2 to help avoid this inevitability that more complicated models will be closer to the training data. Both of those traditional methods* inflate* the estimate of the MSE by taking into account the "degrees of freedom," df, in the model and how that compares to the number of cases n in the training dataset. The inflation looks like

$$\frac{n}{n - \text{df}}$$

So when $\text{df} \to n$, we inflate the MSE quite a lot.

Another approach to this is to use *testing* MSE rather than training MSE. So pick the model with flexibility at the bottom of the U-shaped testing MSE curve.

1.10.4 Why is testing MSE U-shaped?

- Bias: how far $\hat{f}(x)$ is from $f(x)$
- Variance: how much $\hat{f}$ would vary among different randomly selected possible training samples.

In traditional regression, we get at the variance by using confidence intervals on parameters. The broader the confidence interval, the higher the variation from random sample to random sample. These confidence intervals come from normal theory or from bootstrapping. Bootstrapping is a simulation of the variation in model fit due to training data.

Bias decreases with higher flexibility.

Variance tends to increase with higher flexibility.

Irreducible error is constant.

1.10.5 Measuring the variance of independent sources of variation

Simulation: Make and edit a file `Day-03.Rmd`.

1.10.5.1 Explore

Add three different sources of variation. The width of the individual sources is measured by the standard deviation `sd=`.

```
n <- 1000
sd( rnorm(1000, sd=3) + rnorm(1000, sd=1) + rnorm(1000, sd=2) )
```

```
## [1] 3.96915
```

- Divide into small groups and
  - construct a theory about how the variation in the individual components relates to the variation in the whole.
  - test whether your theory works for other random distributions, e.g. `rexp()`

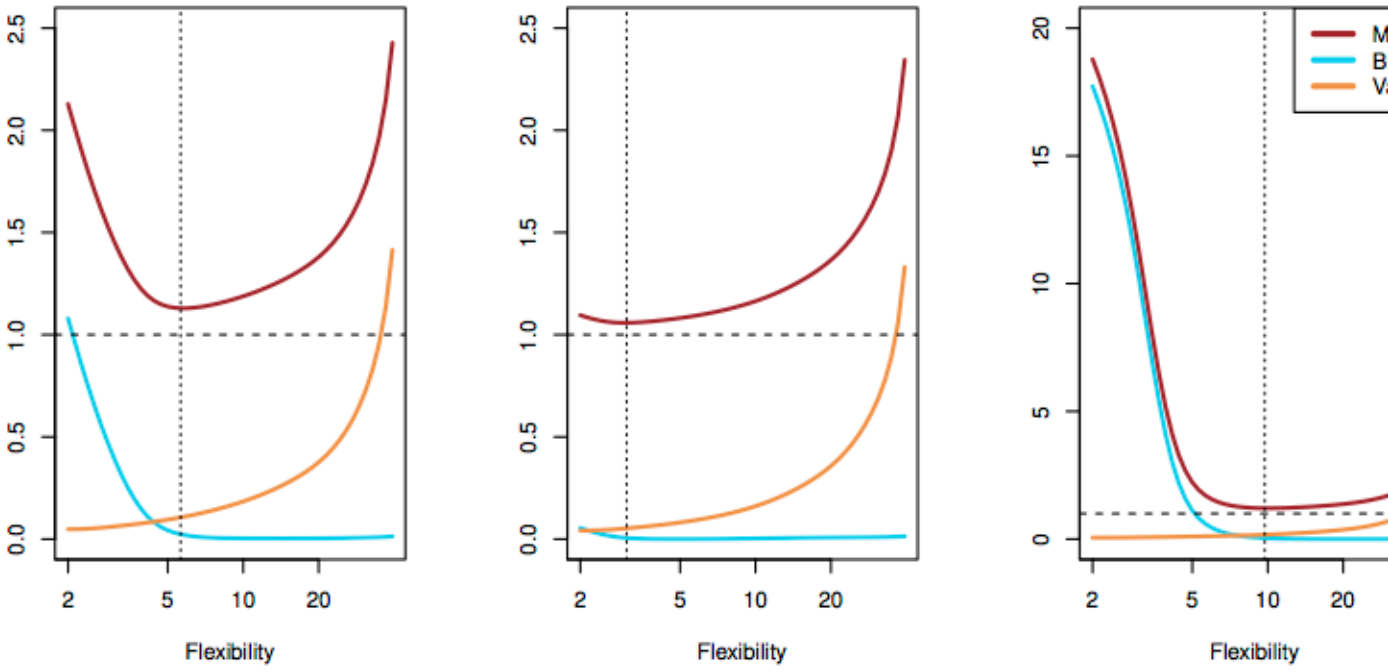


Figure 1.10: ISL Figure 2.12

1.10.5.2 Result (Don't read until you've drawn your own conclusions!)

The variance of the sum of independent random variables is the sum of the variances of the individual random variables.

1.10.6 Equation 2.7

$$E(y - \hat{f}(x))^2 = \text{Var}(\hat{f}(x)) + [\text{Bias}(\hat{f}(x))]^2 + \text{Var}(\epsilon)$$

Breaks down the total "error" into three independent sources of variation:

1. How y_i differs from $f(x_i)$. This is the irreducible noise: ϵ
2. How $\hat{f}(x_i)$ (if fitted to the testing data) differs from $f(x_i)$. This is the *bias*.
3. How the particular $\hat{f}(x_i)$ fitted to the training data differs from the $\hat{f}(x_i)$ that would be the best fit to the testing data.

$$\underbrace{E(y - \hat{f}(x))^2}_{\text{Total error}} = \underbrace{\text{Var}(\hat{f}(x))}_{\text{source 3.}} + \underbrace{[\text{Bias}(\hat{f}(x))]^2}_{\text{source 2.}} + \underbrace{\text{Var}(\epsilon)}_{\text{source 1.}}$$

1.11 Programming Basics I: Names, classes, and objects {progbasics1}

1.11.1 Names

Composed of letters, numbers, `_` and `.`.
- Don't use `.` — it's a bad habit. But plenty of people do. - Can't lead with a number. - Capitalization counts. - Unquoted (… almost always)

1.11.2 Objects

Information (bits) in a particular format.
- Different formats for different purposes. - The format is the `class()` or `mode()`. `mode` is more basic than `class`.

Assignment: Give a name to an object

1.11.3 Vectors

1-dimensional homogeneous collections of numbers, character strings, booleans/logicals, etc.

1.11.3.1 Must Know!

There are some basic types of vectors. The most common are:

1. **numeric**, e.g. `3`, `3.14159`, `6.023e26`,`6.626196e-34`
2. **character**, e.g. `"hello"`, "'When in the course of human events …"
3. **logical** (or "booleans"). The only allowable values: `TRUE` and `FALSE`

Much of the software you'll use in this course will work with an alternative to character strings called "factors."

4. **factor**, an encoded representation of levels of a categorical variable.

Some operations you will use when dealing with categorical variables are `as.character()` and (for older software) `as.factor()`.

Vectors are "1-dimensional" collections. That is, you need only one index to refer to a specific element.

```
my_vector <- c("apple", "berry", "cherry")
my_vector[2]
```

```
## [1] "berry"
```

```
my_vector[c(3, 1, 2, 2)]
```

```
## [1] "cherry" "apple"  "berry"  "berry"
```

```
my_vector[4] <- "durian"
```

Boolean indexing

```
my_vector[my_vector > "b"]
```

```
## [1] "berry"  "cherry" "durian"
```

If you want, you can convert the boolean style to a number style with `which()`.

Other important functions:

- `length()`, to say how many elements there are in the vector. The length can be zero.
- Arithmetic operations, e.g. `sum()`, `mean()`, `max()`, `cumsum()`, and other functions (e.g. `sqrt()`, `log()`, `sin()`), …
- Logical operations, that is, operations that transform vectors into a logical vector. Examples:
  - Comparison: `>`, `==`, `>=`, `!=`, `<`, `<=`
  - Boolean operations: `!`, `|`, `&`. (Note, these are single characters.)
- Categorical operations, e.g. `table()`

```
funny <- ceiling(length(my_vector)*runif(10))
my_vector[funny]
```

```
##  [1] "berry"  "cherry" "berry"  "apple"  "berry"  "apple"  "cherry"
##  [8] "berry"  "apple"  "berry"
```

```
length(my_vector)
```

```
## [1] 4
```

```
dim(my_vector)
```

```
## NULL
```

1.11.4 Matrices

- 2-dimensional homogeneous collections of numbers or of character strings. These collections are called *matrices*
  - 2-dimensional means you need two indices to refer to a specific element.
  - `dim()`
  - Operations, e.g., `t()`, `colSums()`, `rowSums()`, `%*%`, …
  - Index matrices with `[ , ]`. You need to give two

1.11.5 Lists

1-dimensional heterogeneous collections

- create with `list()`

```
my_list <- list(magic = 1:10, greeting = "hello", is_wednesday = FALSE)
my_list[c("greeting", "magic", "magic")]
```

```
## $greeting
## [1] "hello"
##
## $magic
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $magic
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
my_list$magic
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

- index with `[[ ]]` or `[ ]`. The first is for an individual item which you want in the form of that item itself. The second is to create a subset of the list which will be a list itself.
- you can name items in lists and refer to them by name. `[["name"]]` To set names of list items, use named arguments to `list()` or use the `names()` function on the left-hand side of the `<-` operation.

(The left-hand side of `<-` is called an "assignable." These can be names, but they can be other things as well such as indexed arrays, `names()`, etc.)

- Data frames

A list of vectors.

- Each component in a given vector must be the same kind of thing as the other components. (Special cases: `NA` for missing data. Two more special things for numerical data.`NaN`, & `Inf`)
- Important functions: `nrow()`, `names()`, `$`.

1.11.6 Functions

Take inputs and produce an output (and maybe a side-effect). - Make them with `function(){ }`, a special form. (It's not actually a function!) - Try this

```
class(sin)
class(3)
class("a string")
class(function)
```

```
## Error: <text>:4:15: unexpected ')'
## 3: class("a string")
## 4: class(function)
##                   ^
```

1.12 Programming Activity 3

Indexing on data: training and testing data sets

1.13 Review of Day 3

- $f(\mathbf{X})$ versus $\hat{f}(\mathrm{X})$: Platonic idea versus what we get out of the training data. Quip: "The hat means there's a person underneath the model."
- Mean Square Error — like the standard deviation of residuals
- Training vs testing data
- Smoothness, a.k.a. flexibility, model degrees of freedom
  - More flexibility → better training MSE
- Components of MSE
  1. Irreducible random noise: ϵ
  2. Bias: $f(\mathbf{X}) - \hat{f}(\mathbf{X})$
     - Caused by too much smoothness
     - Caused by omitting a relevant variable
     - Caused by including an irrelevant variable
  3. $Var(\hat{f}(\mathbf{X}))$ — how much $\hat{f}$ varies from one possible training set to another.
     - Increased by too many degrees of freedom: *overfitting*
     - Increased by collinearity and multi-collinearity.
     - Increased by large ϵ
     - Decreased by large n

1.14 Start Thursday 15 Sept.

Programming Basics I

Indexing on data: training and testing data sets

Chapter 2

Linear Regression

2.1 Day 4 Preview

- The linear model (e.g. what `lm()` does)
- A variety of questions relevant to different purposes, e.g.
  - how good will a prediction be?
  - what's the strength of an effect?
  - is there synergy between different factors?

2.1.1 ISL book's statement on why to study linear regression

> "Though it may seem somewhat dull compared to some of the more modern statistical learning approaches described … later …, linear regression is still a useful and widely used statistical learning method. Moreover, it serves as a good jumping-off point for newer approaches…. Consequently, the importance of having a good understanding of linear regression before studying more complex learning methods cannot be overstated."

Concepts from linear regression:

- Compact representation of model form: polynomial coefficients.
- Much of inference (confidence intervals, hypothesis tests) can be expressed in terms of a polynomial coefficient.
- "Size" of model quantifiable as an integer: number of coefficients: degrees of freedom.
- Highly efficient estimation (when doing least squares)

2.2 Small data

The regression techniques were developed in an era of small data, such as that that might be written in a lab notebook or field journal. As a result:

1. Emphasis on very simple descriptions, such as means, differences between means, simple regression.
2. Theoretical concern with details of distributions, such as the t-distribution.
   - the difference between z- and t-distributions are of no consequence for moderate DF and higher.
3. No division into training and testing data. Data are too valuable to test! (Ironic, given the importance of replicability in the theory of the scientific method.)

As a consequence of (3), there's a great deal of concern about *assumptions*, e.g.

- linearity of $f(\mathbf{X})$

- structure of ϵ: IID — Independent and Identically Distributed
  - uncorrelated between cases
  - each is a draw from the same distribution.

2.3 Programming basics: Linear Models

Syntactic element: **formulas**. Formulas provide a way of using variables "symbolically." This is useful, for instance, in depicting the desired relationship among variables. Two forms:

- `y ~ x` two-sided
- `~ x` one-sided
- NOT ALLOWED, `y ~`

Important functions:

- `lm()`, `predict()` , `anova()`, `summary()`.
- For later: `solve()`, `model.matrix()`.
- From 155: `coef()`, `fitted()`, `resid()`

Training with `lm()`: Specify formula `Y ~ X1 + X2` ...

```
data(College, package = "ISLR")
mod <- lm(Outstate ~ Enroll + Accept + perc.alumni, data = College )
coef(mod)
```

```
## (Intercept)      Enroll      Accept perc.alumni
## 6387.368606   -2.888077    1.101019  179.528731
```

What kind of thing is `mod`?

Model output with `predict()`

```
predict(mod,
        newdata = data.frame(Enroll = 100, Accept = 1000, perc.alumni = 25))
```

```
##       1
## 11687.8
```

What kind of thing is the output of `predict()`?

```
predict(mod, interval = "confidence",
        newdata = data.frame(Enroll = 100, Accept = 1000, perc.alumni = 25))
```

```
##       fit      lwr      upr
## 1 11687.8 11383.57 11992.03
```

```
predict(mod, interval = "prediction",
        newdata = data.frame(Enroll = 100, Accept = 1000, perc.alumni = 25))
```

```
##       fit      lwr      upr
## 1 11687.8 5548.956 17826.64
```

Why is the "confidence interval" so much narrower than the "prediction interval?"

Inference with `anova()` and `summary()`. This is all "in-sample" inference, not cross-validated.

```
M <- model.matrix(~ Enroll + Accept * perc.alumni, data = College)
qr.solve(M, College$Outstate)
```

```
##        (Intercept)             Enroll             Accept
##      7024.34843865        -3.00377409         0.78401053
##        perc.alumni Accept:perc.alumni
```

```
##       147.31268325         0.02011475
```

For the people who have had linear algebra, why doesn't this work?

```
solve(M, College$Outstate)
```

```
## Error in solve.default(M, College$Outstate): 'a' (777 x 5) must be square
```

Indexing on data: training and testing data sets

2.4 Review of Day 4, Sept 15, 2016

- Discussed the linear regression architecture and how it relates to machine learning.
  - linear regression is designed to work even in "small data" situations where
    a. cross-validation is not appropriate
    b. the number of model degrees of freedom may be almost as large as n
  - provides a ready definition of the "size" of a model: the number of coefficients.
- Introduced the main software used in linear regression, `lm()` and `predict()`
- Programming basics: indexing of vectors, matrices and data frames.

2.5 Regression and Interpretability

Regression models are generally constructed for the sake of interpretability:

- Global linearity
- Coefficients are indication of effect size. The coefficients have physical units.
- Term by term indication of statistical significance

An example on `College` data from `ISLR` package

```
data(College, package="ISLR")
College$Yield <- with(College, Enroll/Accept)
mod1 <- lm(Yield ~ Outstate + Grad.Rate + Top25perc, data = College)
mosaic::rsquared(mod1)
```

```
## [1] 0.2170221
```

```
mod2 <- lm(Yield ~ . - Grad.Rate, data = College)
mosaic::rsquared(mod2)
```

```
## [1] 0.5004599
```

- What variables matter?
- How good are the predictions?
- How strong are the effects?

2.6 Toward an automated regression process

In machine learning, we ask the computer to identify patterns in the data.

- In "traditional" regression (which is still very important), we specify the explanatory terms and the computer finds the "best" model with those terms: least squares.
- In machine learning, we want the computer to figure out which terms, of all the possibilities, will lead to the "best" model.

2.7 Selecting model terms

The regression techniques

- Traditional regression:
  - Our knowledge of the system being studied.
  - Heirarchical principal
    * main effects, then
    * interaction.
- Machine learning
  - Look at all combinations of variables?
  - Activity 1:
    * Write a statement that will pull 2 random variables from a data frame *and* the explanatory variable.
    * Use `Yield ~ .` as the formula.

```
explanatory_vars <- names(College)[-19]

my_vars <- sample(explanatory_vars, size = 12)
new_data <- College[ , c(my_vars, "Yield")]
mod <- lm(Yield ~ ., data = new_data)
mosaic::rsquared(mod)
```

```
## [1] 0.3367808
```

```
mod
```

```
##
## Call:
## lm(formula = Yield ~ ., data = new_data)
##
## Coefficients:
## (Intercept)    S.F.Ratio    Top10perc         Apps     Terminal
##   6.474e-01    2.078e-03    3.182e-03   -2.297e-05   -5.685e-04
##         PhD    Top25perc       Enroll        Books   PrivateYes
##  -1.161e-03   -1.323e-03    8.917e-05    5.326e-05   -4.106e-02
## F.Undergrad  perc.alumni   Room.Board
##  -1.273e-06   -5.822e-04   -3.005e-05
```

```
2^18
```

```
## [1] 262144
```

- Activity 2: * How many combinations are there of k explanatory variables? Calculate this for $k = 5, 10, 15, 20$. How many are there in the `College`? * What about with interactions?
- How long does it take to fit a model?

```
system.time( do(1000) * lm(Yield ~ ., data=College))
```

```
##    user  system elapsed
##   5.953   0.052   6.026
```

```
256*7/3600
```

```
## [1] 0.4977778
```

```
names(College)
```

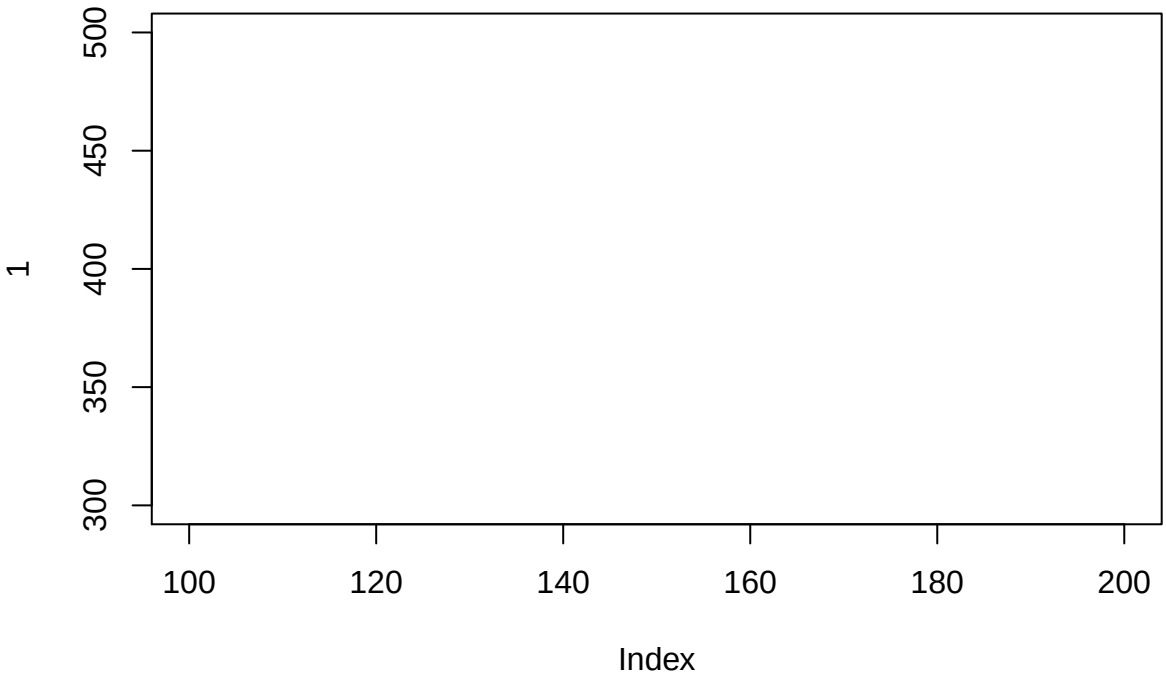
```
##  [1] "Private"     "Apps"        "Accept"      "Enroll"      "Top10perc"
##  [6] "Top25perc"   "F.Undergrad" "P.Undergrad" "Outstate"    "Room.Board"
```

```
## [11] "Books"       "Personal"    "PhD"          "Terminal"    "S.F.Ratio"
## [16] "perc.alumni" "Expend"      "Grad.Rate"    "Yield"
```

With k explanatory variables, 2^k possibilities, not even including interactions. Including first-order interactions, it's $2^k + 2^{k(k-1)/2}$. Calculate this for $k=3. - Increase in R^2? Problem: R^2 will always go up as we add a new term. - Some other measure that takes into account how much R^2 should go up.

2.8 Programming basics: Graphics

```
plot(1, type = "n", xlim = c(100,200), ylim = c(300,500))
```



Basic functions:

1. Create a frame: `plot()`. Blank frame: `plot( , type="n")`
   - set axis limits,
2. Dots: `points(x, y)`, `pch=20`
3. Lines: `lines(x, y)` — with `NA` for line breaks
4. Polygons: `polygon(x, y)` — like lines but connects first to last.
   - fill
5. Color, size, … `rgb(r, g, b, alpha)`, "tomato"

2.9 In-class programming activity

Programming Activity 4: Drawing a histogram.

2.10 Day 5 Summary

2.10.1 Linear regression

- Discussed "interpretability" of linear models, e.g. meaning of coefficients, confidence intervals, R^2, etc.
  - which variables are "important" via ANOVA and mean sum of squares
- Discussed metrics to compare models
  - R^2 – not fair, since "bigger" models are always better
  - Punishment: Two criteria for judging
    * R^2
    * How big the model is.
    * These two are somehow combined together into "adjusted R^2." We'll say more about that today.
  - Cross-validation. Judge each model on its "out of sample" prediction performance.

2.10.2 Coefficients as quantities

Coefficients in linear models are not just numbers, they are physical quantities with dimensions and units.

- Dimensions are always (dim of response)/(dim of this term)
- The model doesn't depend on the units of these quantities. The units only set the magnitude to the numerical part of the coefficient, but as a quantity a coefficient is the same thing regardless of units.
- Conversion from one unit to another by multiplying by 1, but expressed in different units, e.g. 60 seconds per minute, 2.2 pounds per kilogram.

2.10.3 Graphics basics

1. API for graphics: `plot()`, `points()`, `lines()`, `polygon()`, `text()`, …

2.11 K-nearest neighbors

K-nearest neighbors is a simple, general kind of function-building method. But some problems:

- Interpretability: but you can always take partial derivatives.
- When you have prediction (aka "explanatory") variables in dollars and in miles, how do you calculate the distance between points? What are the dimensions of distance?
  - Dimensionality refers to the physical feature, e.g. time, distance, area, volume, money, charge, luminance, mass, …
  - Units are the ways in which dimensions are measured, e.g., cups, gallons, liters … all refer to volume
    * Give some examples of units for each of the dimensions.
    * Some everyday quantities are dimensionless, e.g. pure numbers. Give some examples: … (angles, percent, fractions, … but not ratios in general.)
  - Regression fixes units automatically, since the coefficients themselves have dimensionality. They will adjust automatically to changes in units, so the model is the same regardless of whether we use miles, km, parsecs, …
  - In KNN, to avoid dependence on units, need to do some standardization by dividing by something in the same units, e.g. sd.
- Curse of dimensionality. Let's create 1000 randomly placed points in the unit square:

```
rpts <- matrix(runif(2*1000), ncol=2)
```

What's the distribution of distances from a single random point to the 1000 others:

```
our_point <- runif(2)
```

The distance between our point and each of the others

```
tmp <- matrix(our_point, ncol=2, nrow=1000, byrow=TRUE)
delta <- sqrt(rowSums((rpts - tmp)^2))
```

- How far away is a typical point?
- Write a function that takes the matrix of points and the "our point" and finds the distance from our point to each and every one of the points in the matrix.
- How far away is a typical point in 1-dimensional space?
- In 10-dimensional space?
- In 100-dimensional space?

2.12 In-class programming activity

Day 5 activity

Drawing a histogram.

2.13 Day 6 Summary

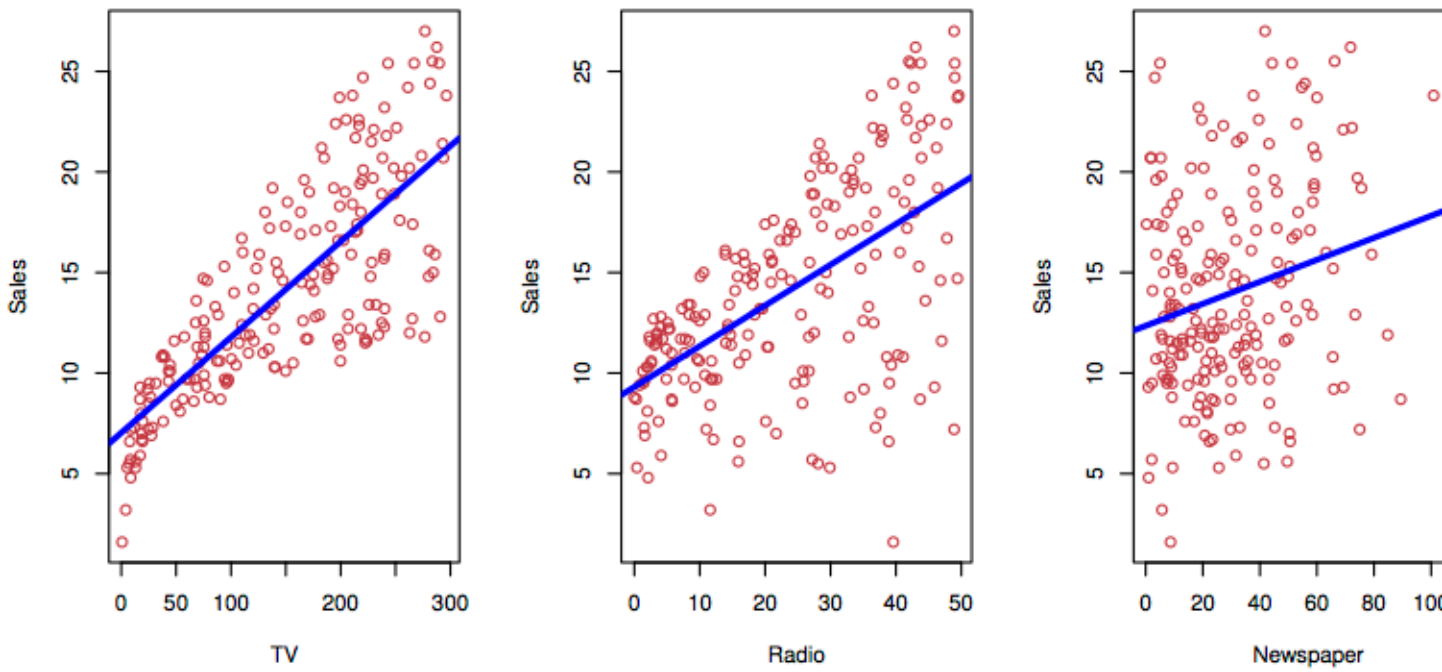
- R^2
  - var(fitted) / var(response)
  - partitioning of variance:
    - var(fitted) + var(resid) = var(response)
    - same with sum of squares: SS(fitted) + SS(resid) = SS(response)
- Adjusted R^2
  - R^2 vs p picture
  - Derive a formula from the picture: we've got $p+1$ df to get from $R^2 = 0$ to our observed R^2, so $n-(p+1)$ df left for the residuals. Rate of increase due to junk is $(1-R^2)/(n-p-1)$. Projecting back p terms gives an adjustment of $(1-R^2)/\frac{p}{n-p-1}$. Subtract this from R^2.
  - Wikipedia gives two formulas:
    - $AdjR^2 = 1-(1-R^2)\frac{n-1}{n-p-1}$ – this projects back $n-1$ terms from 1.
    - $AdjR^2 = R^2-(1-R^2)\frac{p}{n-p-1}$ — projects back p terms from R^2.
- Adjusted R^2
- Whole model ANOVA.
- ANOVA on model parts

2.14 Measuring Accuracy of the Model

- R^2 = Var(fitted)/Var(response)
- Adjusted R^2 - takes into account estimate of average increase in R^2 per junk degree of freedom
- Residual Standard Error - Sqrt of average square error per residual degree of freedom. The sqrt of the mean square for residuals in ANOVA.

2.15 Bias of the model

You need to know the "truth" to calculate the bias. We don't.



- Perhaps effect of TV goes as sqrt(money) as media get saturated?
- Perhaps there is a synergy that wasn't included in the model?

2.15.1 Theory of whole-model ANOVA.

Standard measure: $\frac{\text{Explained amount}}{\text{Unexplained amount}}$

Examples:

- Standard error of mean: $\frac{\hat{\mu}}{\sigma/n}$ – note the n.
- t statistic on difference between two means: $\frac{\hat{\mu}_1 - \hat{\mu}_2}{\sigma/(n-1)}$
- F statistic: $\frac{SS/df1}{SSR/df2}$
  - df1 is the number of degrees of freedom involved by the model or model term under consideration.
  - df2 is $n - (p - 1)$ where p is the total degrees of freedom in the model. (I called this m in the Math 155 book.) The intercept is what the -1 is about: the intercept *can never* account for case-to-case variation.

Trade-off between eating variance and consuming degrees of freedom.

2.16 Forward, backward and mixed selection

Use the `College` model to demonstrate each of the approaches by hand. Start with `pairs()` or write an `lapply()` for the correlation with `Yield`?

Create a whole bunch of model terms

- "main" effects
- "interaction" effects
- nonlinear transformations: powers, logs, sqrt, steps, ...
- categorical variables

Result: a set of k vectors that we're interested to use in our model.

Considerations:

- not all of the k vectors may pull their weight
- two or more vectors may overlap in how they eat up variance

Algorithmic approaches:

- Try all combinations, pick the best one.
  - computationally expensive/impossible 2^k possibilities
  - what's the sensitivity of the process to the choice of training data?
- "Greedy" approaches

2.17 Programming Basics: Functions

1. Syntax of functions:

```
name <- function(arg1, arg2, ...) {
  body of function. Can use arg1, arg2, etc.
}
```

- typically you will return a value. The value calculated by the last command line in the body is what's returned. Or you can use `return()` at any point in the function.
- Often functions are designed to produce "side effects", e.g. graphics.

- Scope: what happens in functions stays in functions.

1. Create a plotting frame: `plot()`
   - Write a function that makes this more convenient to use. What features would you like.

```
blank_frame <- function(xlim, ylim) {

}
```

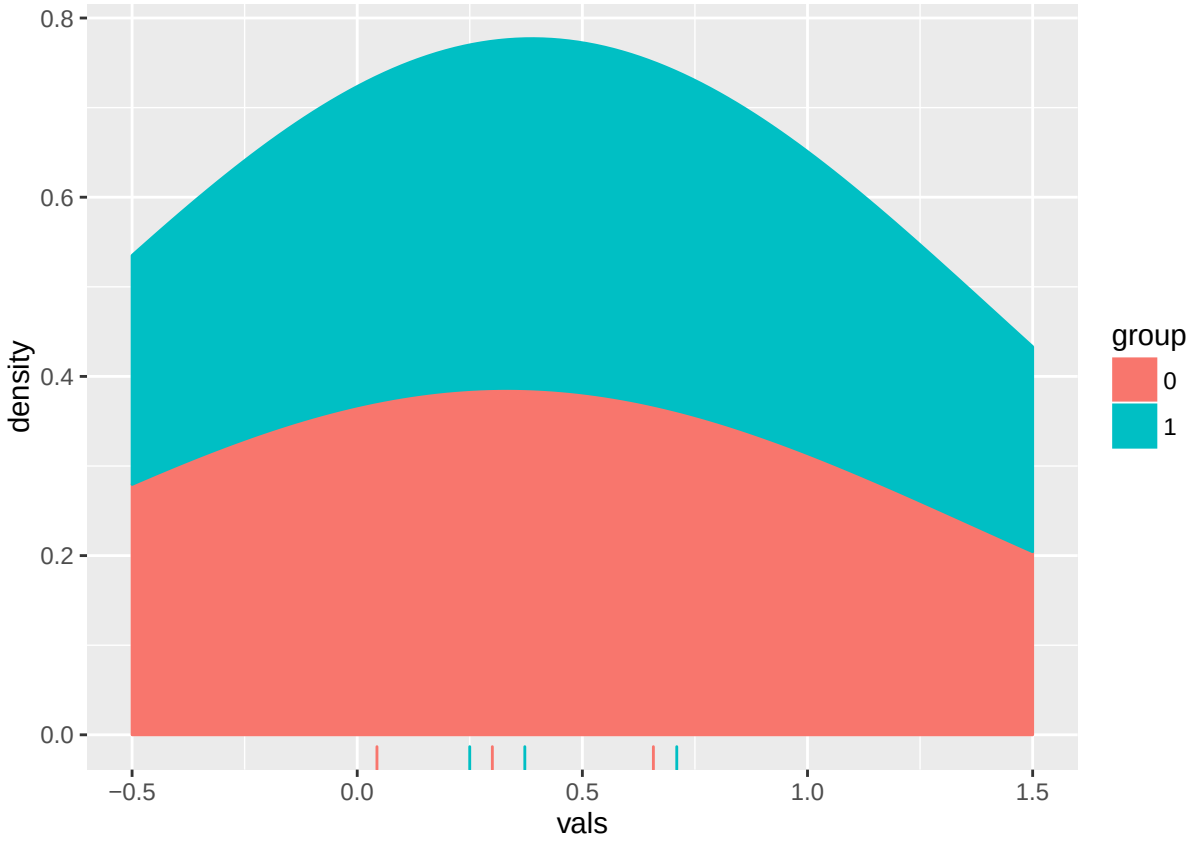
2. Write a function to draw a circle.
   - What do you want the interface to look like? What arguments are essential? What options are nice to have?

2.18 In-class programming activity

Histogram and density functions

```
set.seed(101)
n = 20
X <- data.frame(vals = runif(n),
                group = as.character((1:n) %% 2))
ggplot(head(X, 6), aes(x = vals)) +
  geom_density(bw = 1, position = "stack", aes(color = group, fill = group)) +
  geom_rug(aes(color = group)) +
```

```
# facet_grid( . ~ group) +
 xlim(-0.5, 1.5)
```



Day 6 activity

2.19 Review of Day 7

- We finished reviewing adjusted R^2 and ANOVA.
- Started talking about linear algebra.

We only got through the first few elements in our review of linear algebra. Let's go through them again

2.20 Using predict() to calculate precision

- confidence intervals
- prediction intervals

2.21 Conclusion

This wraps up our look at linear regression. Main points:

- model output is a linear combination of the inputs.
- `lm()` finds the "best" linear combination.
- rich theory relating to precision of coefficients and the residuals.

- traditional ways of applying that theory: F tests and t tests.

Chapter 3

Foundations

The topics in this section — linear algebra, Bayes' rule, and likelihood — underlie many of the machine-learning techniques we will be studying later in the course. Bayes' rule is a way to *flip* conditional probabilities. Among other things it allows you to interpret data in the light of previous knowledge and belief (which to be fancy, we can call "theory"). Likelihood is a unifying principle for using data to estimate model parameters and is fundamental in statistical theory. It's also an essential part of Bayes' rule. And linear algebra is used throughout statistics and machine learning. Among other things, it's at work behind the motivation and calculations of regression.

3.1 Linear Algebra

The idea here is not to teach you linear algebra, but to expose you to some of the terminology and operations of linear algebra, so that when you see it again later in the course you'll have a good start.

- A vector — a column of numbers. The *dimension* is the count of numbers in the vector.
- A *space*: the set of all possible vectors of a given dimension.
- *Scalar multiplication*
- *Vector addition*: walk the first vector, then the second.
- *Linear combination*: do scalar multiplication on each vector, then add.
- A *matrix* — a collection of vectors (all the same dimension).
- *Dot product*: a basic calculations on vectors:
  - the length (via Pythagorus)
  - the angle between two vectors
  - orthogonality: when two vectors are perpendicular, their dot product is zero.
- Matrix operation: *Linear combination*.
  - Take a linear combination of the vectors in a matrix. Analogous to taking a trip. Result: a vector representing the end-point of the trip.
- The *subspace* spanned by the matrix: the set of all possible points you can get to with a linear combination.
- Matrix operation: *Orthogonalization* — Find perpendicular vectors that span the same subspace as a matrix. Example, draw the picture for two vectors $\vec{a}$ and $\vec{b}$.
- Matrix operation: *Projection*
  - Given a matrix M and a vector V, find the closest point in the subspace of M to the vector V. How? Orthogonalize matrix M, then for each vector in orthogonalized M, subtract out the part of V aligned with that vector.
- Matrix operation: *inversion* — the inverse operation to linear combination.
  - given an end-point in the space spanned by M, figure out a linear combination that will get you there.

- Vector to vector operation: Outer product. col vector × row vector.
  - Can generalize to operations other than ×.

For linear algebra folks: Projection is the Q part of QR decomposition. R is the solve part. - In economics, they write things in an older style: Solve $Mb = y$ for b. But M may not be square, so no regular inverse. - Pre-multiply by M^T to get $M^T Mb = M^T y$ - Invert to get $b = (M^T M)^{-1} M^T y$. The matrix $(M^T M)^{-1} M^T$ is called the pseudo inverse.

3.2 Arithmetic of linear algebra operations

1. Addition comes for free. Confirm this.
2. Scalar multiplication comes for free. Confirm this.
3. Write a function for the dot product.

```
vdot <- function(v, w) {
  sum(v * w)
}
```

1. Write a function for the vector length.

```
vlength <- function(v) {
  sqrt(vdot(v, v))
}
```

1. Write a function for the cosine of the angle between two vectors.

```
vangle <- function(v, w, degrees = FALSE) {
   theta <- acos(vdot(v, w) / (vlength(v) * vlength(w)))

   if (degrees) theta * (180 / pi )
   else theta
}
```

1. Write a function to project vector $\vec{a}$ onto $\vec{b}$. Subtracting the result from $\vec{a}$ will give the component of $\vec{a}$ orthogonal to $\vec{b}$. So we can decompose $\vec{a}$ into two components relative to $\vec{b}$. Show that the supposedly ortogonal component is really orthogonal to b — that is, the dot product is 0.

```
vproject <- function(v, onto) { # the red thing
  onto * vlength(v) * cos(vangle(v, onto)) / vlength(onto)
}
vresid <- function(v, onto) {
  v - vproject(v, onto)
}
```

1. Generalization: Write a function to orthogonalize a matrix M.
2. Generalization: Write a function to calculate the projection of V onto M.

```
vdot <- function(a, b) {
  sum(a * b)
}
vlen <- function(a) sqrt(vdot(a, a))
vcos <- function(a, b) {
  vdot(a, b) / (vlen(a) * vlen(b))
}
vangle <- function(a, b, degrees = FALSE) {
  res <- vcos(a, b)
  if (degrees) res <- res * 180 / pi
```

```
  res
}
vproj <- function(a, onto = b) {
  vlen(a) * vcos(a, onto) * onto
}
```

3.3 The geometry of fitting

- Data tables: cases and variables.
- Case space (the rows of the matrix) and variable space (the columns).
- A quantitative variable is a vector.
- A categorical variable can be encoded as a set of "dummy" vectors.
- Response variable and explanatory variable
- The linear projection problem: find the point spanned by the explanatory variables that's closest to the response. That linear combination is the best-fitting model.
  - One explanatory and the response
  - Two explanatory on board and the response on the board (perfect, but meaningless fit)
  - Two explanatory in three-space and the response (residual likely)

3.4 Precision of the coefficients

$$\text{standard error of B coef.} = |\text{residuals}| \frac{1}{|\text{B}|} \frac{1}{\sin(\theta)} \frac{1}{\sqrt{n}} \sqrt{\frac{n}{n-m}}$$

- m — degrees of freedom in model
- θ — angle between this model vector and the space spanned by the others
- B — this model vector
- residuals — the residual vector

3.5 Likelihood and Bayes

We accept that our models won't produce an $\hat{f}(x)$ that always matches y. There is the *irreducible error* ϵ, in addition to variance and bias.

- Variance: a measure of how far off our $\hat{f}()$ is from that we would have been able to construct with an infinite amount of data: $\hat{f}_\infty()$.
- Bias: a measure of how far off $\hat{f}_\infty()$ is from $f()$.

We're using *mean square error* or *sum of square errors* as a measure of how far $\hat{f}(x_i)$ is from the actual result y_i.

Now we're going to look at the difference in terms of probabilities: what would be the probability of any particular $\hat{y}_i$ given our $\hat{f}(x_i)$.

Let's quantify probability.

3.6 Summary of Day 8

We finished up our brief introduction to linear algebra and started discussing probability. I suggested the rather broad definition of a probability as a number between zero and one.

3.7 Day 9 Announcements

Make sure you've accepted the invitation to the discussion group.

Reading for Thursday: "What is Bayesian statistics and why everything else is wrong"

3.7.1 What's a probability?

- Chances of something happening
- Frequentist: Number of "favorable events" / number of events
- Bayesian. Number between 0 and 1.

$ p(rain | Sept 29, Libra, Thursday)$

- densities
- cumulative — this is really what probability refers to.
- discrete events
- joint events
- conditional events
- relating joint and conditional: p(A & X) = p(A | X) p(X) = p(X | A) p(A)
- Bayes rule p(A | X) = p(X | A) p(A) / p(X)

3.8 Conditional probability

The probability of an event in a *given* state of the world. That state of the world might have been set up by another event having occurred.

3.9 Inverting conditional probabilities

What we want is p(state of world|observations). I'll write this $p(\theta|\mathcal{O})$

Tree with cancer (benign or malignant) and cell shape (round, elongated, ruffled)

SPACE FOR THE TREE

SEE PAPER NOTES. (remember to transcribe them here)

, e.g. observe ruffled, what is the chance that the tumor is malignant.

Of the 10000 people in the study,
* 7000 had benign tumors of which 10% or 700 had ruffled cells * 3000 had malignant tumors of whom 60% or 1800 had ruffled cells

So, of the 2500 people with ruffled cells, 1800 had malignant tumors. $p(\theta|\mathcal{O})$

3.10 Exponential probability density

What's the time between random events, e.g. 500-year storms or earthquakes in a region that has a big one roughly every 100 years?

Earthquake warning in Southern California, late Sept. 2016

> But over the last week, anxieties were particularly heightened, and the natural denial that is part of living in earthquake country was harder to pull off.

> A swarm of seismic activity at the Salton Sea that began a week ago prompted scientists to say there was an elevated risk for a big San Andreas fault earthquake. By Monday [Oct 3, 2016], that risk had lessened.
>
> But the impact of that warning was still being felt. For some, it meant checking quake safety lists. Others looked at preparing for the Big One, such as bolting bookshelves to walls, installing safety latches on kitchen cabinets and strapping down televisions.

Why has the risk gotten smaller? How much smaller?

3.10.1 Meanwhile, further north ...

From *The Really Big One, an article in the New Yorker about discoveries in the last few decades that established a high risk in the Pacific Northwest for an earthquake of magnitude 9.

> We now know that the Pacific Northwest has experienced forty-one subduction-zone earthquakes in the past ten thousand years. If you divide ten thousand by forty-one, you get two hundred and forty-three, which is Cascadia's recurrence interval: the average amount of time that elapses between earthquakes. That timespan is dangerous both because it is too long—long enough for us to unwittingly build an entire civilization on top of our continent's worst fault line—and because it is not long enough. Counting from the earthquake of 1700, we are now three hundred and fifteen years into a two-hundred-and-forty-three-year cycle.
>
> It is possible to quibble with that number. Recurrence intervals are averages, and averages are tricky: ten is the average of nine and eleven, but also of eighteen and two. It is not possible, however, to dispute the scale of the problem.

The last paragraph …

> All day long, just out of sight, the ocean rises up and collapses, spilling foamy overlapping ovals onto the shore. Eighty miles farther out, ten thousand feet below the surface of the sea, the hand of a geological clock is somewhere in its slow sweep. All across the region, seismologists are looking at their watches, wondering how long we have, and what we will do, before geological time catches up to our own.

Have students propose distributions and justify them.

3.11 California earthquake warning, reprise

The Salton Sea earthquake happens. Our prior on large λ immediately surges, so there is a significant probability of a quake in the next hours. But as more time goes by, that probability goes down.

- We're interested in λ in the exponential distribution $\lambda \exp(-lambdat)$. This has a cumulative $1 - \exp(-\lambda t)$.
- Observation: Earthquake hasn't occurred after D days. Likelihood is 1 minus the cumulative, or $\exp(-\lambda D)$.
- Prior: a mix of the conventional (very small lambda) and some small probability of very high lambda.

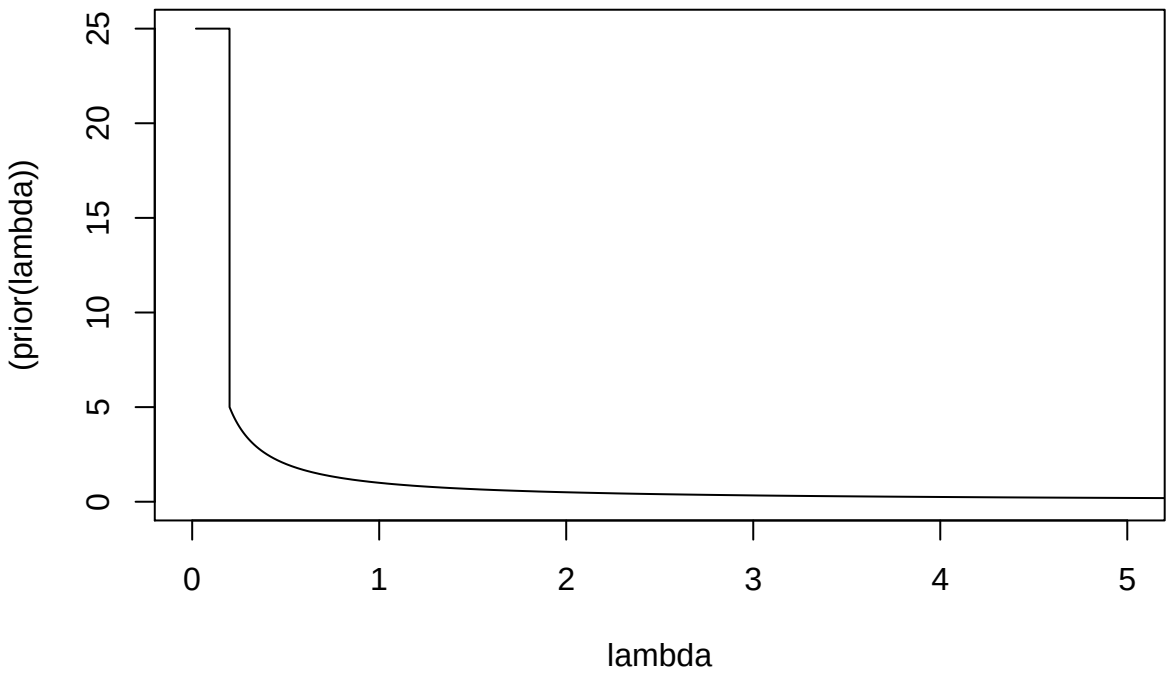
Plot out the posterior for different values of D:

- D = 0.1 two hours after the quake.
- D = 1 a day after the quake
- D = 3 three days after the quake

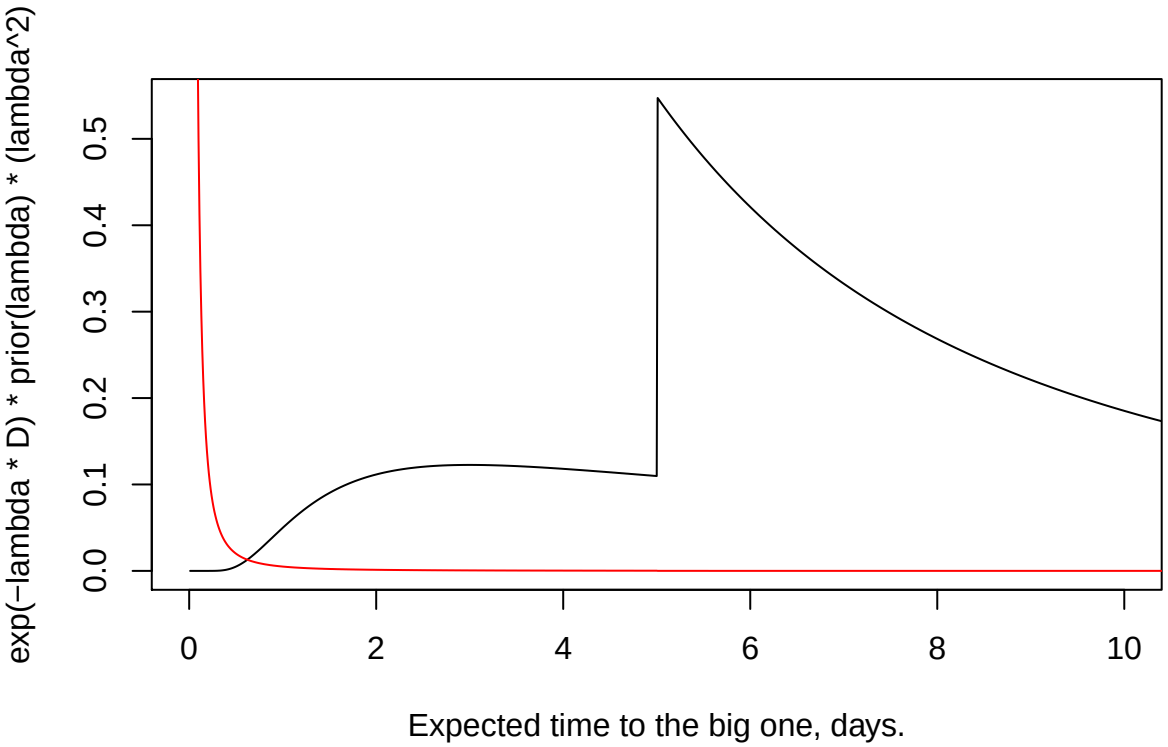
The area to the right of 5 days (in expected time to the next quake) is the conventional model.

Plot this out as a function of $1/\lambda$, so we need to adjust the density by $|df/d\lambda| = |d\frac{1}{\lambda}/d\lambda| = \lambda^2$

```
D <- 3
lambda <- 100/(1:5000)
# prior: proportional to lambda:
#  small lambda unlikely, so short time to next earthquake
prior <- function(lambda) (ifelse(lambda < .2, 25, 1/lambda))
plot(lambda, ( prior(lambda)), type = "l", xlim = c(0,5))
```




```
plot(1/lambda, exp( - lambda * D) * prior(lambda) * (lambda^2),
     type = "l", xlab = "Expected time to the big one, days.",
     xlim = c(0, 10))
lines(1/lambda, lambda*.005/prior(lambda), col = "red")
```



For small D, the "urgency" part of the prior overwhelms the likelihood. As D gets bigger, we revert to the standard model.

3.12 The Price is Right!

The Price is Right is a game show in which contestants compete to guess the price of a prize. The winner is the person whose guess is closest to the actual price considering just those contestants who guesses a price less than or equal to the actual price.

Strategy:

1. First person to guess: an honest guess, hedged on the low side.
2. Second person: bias guess to be far from the first person's guess.
3. Third person:
4. Fourth person: Zero, or just above one of the other guesses.

Play this game. Call down 4 contestants. What's the price of this yacht?

Now, suppose rather than being a strategic game biased toward the last guesser, we wanted to evaluate political prognosticators. The winner should be the person who makes the best prediction rather than the best guess.

Game: Predict the number of electoral college votes for Donald Trump.

Game: Predict the results of the Ukrainian Parliament's vote of no confidence in Prime Minister Arseniy Yatsenyuk. How many votes for no confidence were there.[1]

Play this game asking people to draw the probability distribution of their prediction.

- Suppose you know something about the contestants.
  - David Moore from International Studies
  - Gary Krueger from Economics

[1] Actual result for you to compare your prediction to: one-hundred ninety-four out of three-hundred thirty-nine.

Figure 3.1:

  - Sybill Trelawney from Divination Science
  - Jesse Ventura from Political Science
- You've been asked to assign a probability to each contestant. You'll use this probability to weight each of their future predictions.

Have the contestants keep their identity secret at first.

Draw a density on the board. Give them a vertical scale for density, insisting that each of their densities has area one.

1. *The Likelihood Game*: Who won? How to evaluate the predictions?
2. *The Bayesian Game*:
   - The contestants reveal their identity
   - What's your posterior probability on each of them.

3.13 From likelihood to Bayes

Multiply likelihood by prior probability. Normalize so that total probability is 1.

3.14 Choosing models using maximum likelihood

- We model the error as random, with a probability distribution we choose. Often this distribution has parameters.
- To find the error, we need to make an assumption of what the parameters of the deterministic model are.
  - Make that assumption.
  - Make a similar assumption for the parameters of the probability distribution.
  - Find the errors.
  - Calculate the probability of those errors given the probability distribution that we choose. That's the likelihood for the assumed parameters.
- Repeat in order to modify the assumptions to increase the likelihood.

Straight line model:

Gaussian errors:

$$f(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

What happens when you take the log … why it's sum of squares.

Question: What about minimizing the absolute value of the residuals, rather than the square? - Corresponds to a two-sided exponential distribution like $\frac{\lambda}{2}\exp(-\lambda|x|)$

3.15 Day 9 Preview

Read "What is Bayesian Statistics and why everything else is wrong."

Emphasize the choice of what detail of the sampling model to use. Just this school in isolation? This school as the max of 1000 schools?

Appendices

Connecting RStudio to your GitHub repository

3.16 Setting up your Math 253 repository

First, we have to set up communications between your RStudio system and GitHub. You need only do this once on each RStudio system. But if you want to work on, say, a new computer or new RStudio server, you'll need to do it again.

1. Make sure you have a GitHub account. You'll need two pieces of information about your account:
   a. Your GitHub ID. For the instructions, I'll call in `brosenberg`, but remember to use your own ID
   b. The email address you gave when you set up your account. I'll use `brosenberg@macalester.edu` for the example.
2. Start up RStudio and do the following:
   a. In the console, give these commands:

```
system('git config --global user.name "Brian MacAlistair"')
system('git config --global user.email "brosenberg@macalester.edu"')
system('git config --global --list')
```

   b. Select the menu item Tools/Global Options/Git-SVN. You will see a button to "Create RSA key …". If the box above that button is empty, press the button. Otherwise continue on ….
   c. Click "View public key" and copy the resulting displayed text to your clipboard.
3. Go to your GitHub account. It will have an address like `github.com/brosenberg`.
   a. Press the "Edit Profile" button, then select "SHS and GPG keys."
   b. Press the "New SSH key" button.
   c. Two text boxes will appear. In the smaller one, insert some description of your RStudio system, e.g. `rstudio.macalester.edu` or `my own laptop`. In the larger one, paste the text you copied to your keyboard in step 2c.
4. Almost done … A GitHub repository has been set up for you. It will has a URL in this form: <github.com/dtkaplan/math253-brosenberg>. If that doesn't work, ask for help. There might have been some mistake or delay in setting up your repository, and only `dtkaplan` can fix things.
   a. Direct your browser to the address of your repository.
   b. On the right side of the page, there is a green button: "Clone or download". Press it. A small dialog panel will appear. It should say, "Clone with SSH". (If not, press the small "Use SSH" link to the right.)
   c. Click on the small clipboard icon. This will copy an address to your clipboard. The address starts with `git@github.com`.
5. Go back to your RStudio system. Select File/New Project/Version Control/GIT. A dialog box will appear.
   a. Paste the address from 4c into the topmost text-entry widget in the dialog box.
   b. Press "Create project." Some stuff will happen. Ultimately, you should see RStudio restart and the name of your repository (e.g. `math253-brosenberg`) will be on the upper right corner of the RStudio window. (If you are using the server version of RStudio, you might have to refresh your

browser to see the change.)

3.17 Using your repository

You will be modifying files that are already in your repository, for instance `01-Programming.Rmd`.

1. Make sure that the project displayed in your RStudio is the right one, e.g. `math253-brosenberg`
2. Open, edit, debug, and revise the files you are working on in the normal way. You should knit the file to HTML frequently. This helps you spot problems early.
3. You'll mainly be working with `.Rmd` files. When you're satisfied with things, knit the file to produce HTML one final time.
4. Ready to "hand in" your work? Go to the "Git" tab in RStudio. You should see at least two files listed: the file you edited and the corresponding HTML file.
   a. "Stage" the files by checking the little boxes.
   b. "Commit" the files by pressing the "commit" button. You will be asked to write a message. Choose something short and informative, e.g. "Day 1 project submission." A dialog box will appear with some text indicating what was in the commit.
   c. Press the "Pull" arrow.
   d. Press the "Push" arrow.

Assuming that no errors appear, you are all done!

Well … actually you're never "all done." You will often want to revise your work. Follow the same steps as above.

It is **not cheating** to revise your work, even after you hand it in. I encourage you to do this, be it the next day or several weeks after you handed it in. Git keeps a detailed history of all your commits, so I can always find any version of the file that I need.

3.18 Why are we doing this?

We speak of "handing in" assignments, projects, and other work for a course. But just as we no longer "dial" a telephone, there is no longer a hand involved in handing in. With course support systems like Moodle, you "hand in" by uploading to a server a copy of the file containing your work. You have a working copy of your work and, when you're finished with your work, you upload a final copy.

Making a copy has implications that can get in the way of carrying out your work. The problem is that there is usually nothing in the copy that indicates unambiguously where it comes from. If you want to revise the document, should you be revising the working document or the final copy?

The word "final" suggests the traditional approach to this question. Once the work is done and the document is finalized, you no longer make any changes.

It turns out that "final" does not accord well with the way that people work in collaboration. For all the time that people work with a document, there is a continuing process of revision and refinement. You may think something is final, but in a collaboration that's not entirely your call; you may find out that it isn't yet finished. Insofar as your document contains web links to other documents, the situation becomes even more complex, since those other documents may change.

An extreme instance of this lack of finality appears in software development. Software is complex and bugs are apt to be discovered even after the "final" release. Any piece software written by you or your team relies on and communicates with other software. Changes in that other software can imply a need to revise your own software. We've become familiar with the idea of versions of software: R 3.3.2, Word 2016, etc.

There are advantages in thinking about documents in the same way we think about software.[2] Or, put another way, there are advantages to thinking about working with documents using the same tools that computer programmers use to manage their own complex collaborations with other programmers and with other software.

That's what we're going to do. There are two reasons: (1) it provides a superior way of managing communications and (2) it is the way things are heading in general, so the skills and concepts you develop will help you in your future work and career.

To outline the components of the process …

1. You are going to be writting documents using the .Rmd (R/Markdown) format. You'll use this even if there happens not to be any R content in the document.
2. Each document that you create as part of your work for this course will be located in an RStudio "project." A project is a means of grouping together related files.
3. The project and the files within it will be located on your own computer. This might be your own laptop or a server.
4. Your project will be cloned to storage on other computers. In particular, there will be a clone on the instructor's computer and another clone on a server in the cloud.
5. The cloud server is maintained by GitHub.com. This is not the only such server, but it is the one we are going to use.
6. Both GitHub.com and your computer (and any other computers the project is cloned on) communicate with a system called "git." The git system provides facilities for keeping a thorough record of the changes you make in files contained in your project. You control how fine-grained you want this record to be. The means for doing this is to take a snapshot of the current state of your project. Such a snapshot is called a "commit." You have control of which of the documents will have changes recorded in the history; directing git to include the changes to a particular file (or even its creation) is called "staging" the file.
7. At times of your own choice, you can synchronize/update the clone of the web server to the most recent commit of your project. This is called "pushing" the commit.
8. Also at times of your choosing, you can synchronize/update your own clone of the project to the one on the GitHub server. This is called "pulling" a commit.
9. Pulling and pushing are the means by which you collaborate with others on the project. Just as you will push your changes to the server, others will be pushing their own changes. You pull to synchronize your clone of the project with the changes that have been pushed by others. In a typical work session,
    a. you will **commit** the current state of your project,
    b. then **pull** from the server,
    c. make the revisions you want in your clone.
    d. commit as often as you like within a session. Committing provides you with a marker. If something went wrong and you want to undo your revisions, it's very easy to do so to the point where you last made a commit. Some people commit every paragraph. You decide. It takes very little time and costs nothing. If your file is intended to be translated to HTML, make sure to do this and to stage and commit the resulting version of the HTML file.
    e. When you are done with that session's revisions, you — wait for it! — once again **pull** from the server. This lets you capture any changes that others made while you were revising your own clone of the project. It's important because, sometimes, your changes will conflict with those made by others and pulling gives git a chance to bring any such conflicts to your attention. (Git will not let you push until all such conflicts have been resolved.)
    f. Finally, you **push** to the server, so that the clone on the server is synchronized to the changes you have made.
    g. At the instant after your push, you are now ready to continue at step (c). If you recommence work after enough time has elapsed that someone else might have pushed their own clone to the server, you should instead continue at step (a). You don't need to be anxious about this; git will

[2] Documents on a computer are always software: instructions to some display system about what image to paint on the computer screen or paper. To paraphrase the artist Magritte, "Ceçi n'est pas un document." What you write is the software to create a document, just as Magritte's famous painting of a pipe is a painting, not a pipe.

identify any conflicts that might have arisen due to another collaborator's commits after step (f).

It's up to you to decide how often you should push. Pushing serves two purposes:

- It "backs up" your work to the cloud, so you won't lose anything done up to the point of the push.
- It allows your collaborators to work with your revisions.

You will "hand in" your work by

a. Staging any files involved in the work.
b. Committing the staged files. You might want to identify the commit with a message like "Handing in Assignment 2."
c. Pushing to the server.

You're work will be marked as "handed in" the moment you commit. But the instructor will only be able to see the files you've handed in *after* you push to the cloud server. If you decide to revise your work, simply revise, stage, commit and push again. The instructor will see the new version and will also have access to the earlier versions along with the time stamp made when they were committed. When in doubt about a deadline, push your work. You can always revise and push again if you discover that you have additional time until the deadline. What's more, you can revise and push again *after* the deadline. Your instructor will be able to sort out which is the version to use for grading purposes. It is *not cheating* to revise after the deadline, because the instructor always knows which versions were submitted before the deadline. Indeed, I encourage you to revise your work even after it's been submitted. That way you will have a copy that reflects your current understanding.

By the way, the web site for this course is managed in exactly the same way. The only difference is that whatever clone the server has is made available to web browsers.

Vocabulary: You should be able to give an accurate definition of each of these words, and say how they are connected to one another:

- git, GitHub, clone, stage, commit, push, pull, conflict

DT Kaplan, Tue Oct 4 12:58:39 2016

Instructions for the publishing system: Bookdown

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter **??**. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter **??**.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 3.2. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 3.1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

You can write citations, too. For example, we are using the **bookdown** package (Xie, 2016) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015).

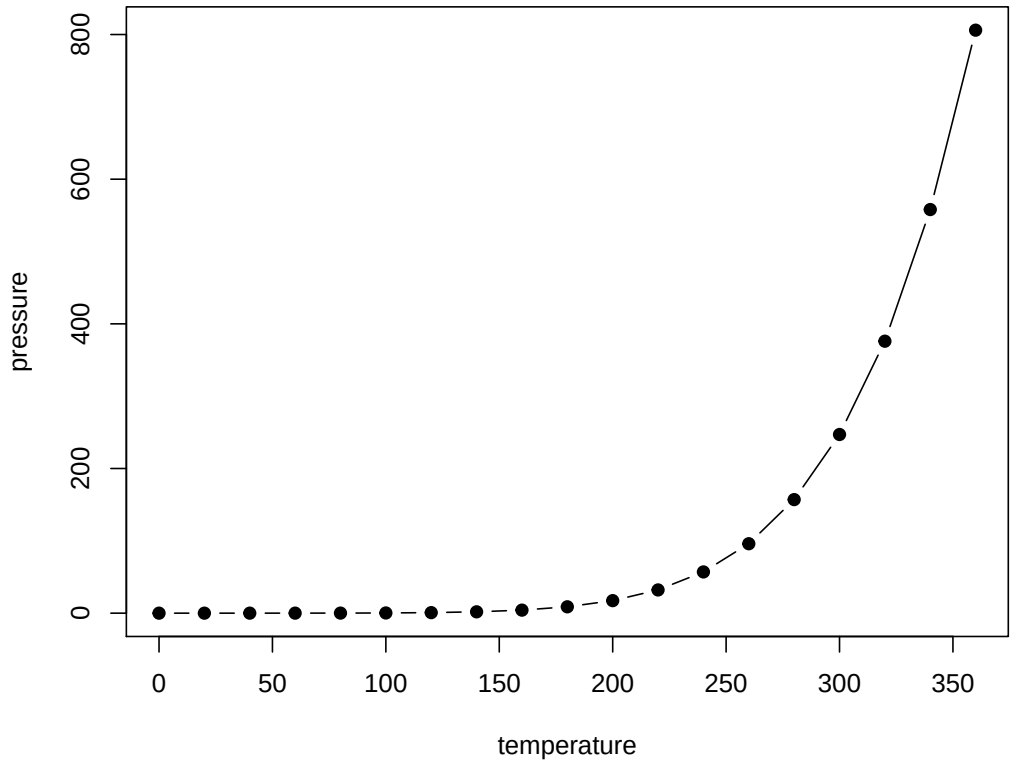


Figure 3.2: Here is a nice figure!

Table 3.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

Bibliography

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2016). *bookdown: Authoring Books with R Markdown*. R package version 0.1.6.