STATS 401. Applied Statistical Methods II

Chapter 1. Getting started

Welcome!

Objectives: Linear statistical models are the foundation for most of applied statistics. We will develop statistical computation skills (R programming) and mathematical skills (working with matrices and probabilities) while studying data analysis using linear models.

Course information: The syllabus on the course website `ionides.github.io/401f18` has a course outline, comments on pre-requisites, exam times, and other relevant information.

An outline of a data analysis

- A typical data analysis has the following steps:
  1. Obtain data and read it into R
  2. Plot the data
  3. Develop a model
  4. Estimate parameters and test hypotheses of interest
  5. Interpret the results
- All these steps involve statistical computing, either doing it or interpreting results from it.
- The two rising stars in statistical computing are R and Python (`http://r4stats.com/articles/popularity/`). Generally, R is preferred for data analysis, and Python for larger programming projects.

We live in an era of abundant data. Learn R!

Getting started with R

- Some statistical software packages operate by selecting menu options.
- For example, this is the basic way of using R Commander.

Question 1.1. Why should we prefer to use the command line form of R rather than point-and-click menu options?

- Homework 0 involves installing R and RStudio on your laptop.
- The R package `swirl` teaches R in R. See Homework 1.
- R code is given in the notes.
- The course website also has links for the R code extracted from the notes. For example, `ionides.github.io/401f18/01/notes01.R`
- Typing commands and seeing what happens is a good way to learn R.

Case study: Are people healthier in booms or busts?

- Is population health **pro-cyclical** (improving in business cycle booms) or **counter-cyclical** (improving in recessions), or neither?
- We will analyze data on annual death rates to investigate this.
- **Life expectancy at birth** combines instantaneous death rates at all ages and is a basic measure of current population health.
- Life expectancy in 2017 is the average lifespan of a fictitious person whose probability of dying at age 10 matches mortality of 10-year-olds in 2017, and the probability at age 50 matches 50-year-olds in 2017. (`data.oecd.org/healthstat/life-expectancy-at-birth.htm`).
- USA data for 1933–2015 are in the file `life_expectancy.txt` on the class website.
- We could read the data into R directly from a website, but it is good data analysis practice to make your own local copy. Here is a way to do that in R:

```
download.file(destfile="life_expectancy.txt",
  url="https://ionides.github.io/401f18/01/life_expectancy.txt")
```

Inspecting the raw data file

- The raw data can be viewed as a plain text file.

Question 1.2. We work with data in a plain text format. Spreadsheets can be saved to `.txt` or `.csv` format. Why is this good data analysis practice?

- The first lines of the life expectancy data file are:

```
# The United States of America, Life expectancy at birth.
# Downloaded from Human Mortality Database on 30 Oct 2017.
# HMD request that you register at http://www.mortality.org
# if you use these data for research purposes.
  Year       Female    Male      Total
  1933        62.78    59.17     60.88
  1934        62.34    58.34     60.23
```

- Lines marked with # are **comments** that are ignored by R.

Read the data into R and then inspect it

```
L <- read.table(file="life_expectancy.txt",header=TRUE)
```

- The **function** `read.table()` reads data from a file to make a **data matrix** in R.
- The **assignment operator** `<-` gives this matrix the name `L`.
- The **argument** `file="life_expectancy.txt"` names the file.
- `header=TRUE` tells R to treat the first row as column names.
- Let's look at the first three rows of `L` using the function `head()`.

```
head(L,3)
```

```
##   Year Female  Male Total
## 1 1933  62.78 59.17 60.88
## 2 1934  62.34 58.34 60.23
## 3 1935  63.04 58.96 60.89
```

- Check this matches the data file. Reading data correctly can be fiddly.

A comment on the assignment operator

- We read the **assignment operator** `<-` as "gets".
- The assignment above is read as "`L` gets the data in `life_expectancy.txt`".
- We could equivalently have used `=`

```
L = read.table(file="life_expectancy.txt",header=TRUE)
```

- You will see both `<-` and `=` used for assignment.

Question 1.3. `<-` is slightly better coding practice than `=`. Why?

Using matrix indexing to identify entries in the dataset

- Each entry in a data matrix can be identified by its row and column index.
- We can use **matrix indexing** to read and manipulate entries of `L`.

```
L[2,3]
```

```
## [1] 58.34
```

- `L[i,j]` is the row `i` column `j` entry of `L`.
- Most datasets have this matrix form, also called **rectangular** data.
- Each row is a measured object (a person, a car, an experiment, a country) generically called a **unit**.
- The columns lists **variables** measured for each unit.

Question 1.4. For the life expectancy data, what are the units? What are the variables?

Selecting rows or columns of a matrix

- Above, we used the function `head()` to see the first 3 rows of `L`.
- We could have used matrix indexing.

```
L[1:3,]

##   Year Female  Male Total
## 1 1933  62.78 59.17 60.88
## 2 1934  62.34 58.34 60.23
## 3 1935  63.04 58.96 60.89
```

- `1:3` is the sequence `1,2,3`.
- The blank space after the comma in `L[1:3,]` requests all the columns for the specified rows.

Vectors in R

- A **numeric vector** in R is a sequence of numbers.
- One way to make a vector is to use the **concatenation function**, `c()`.

```
v <- c(3,1,4,1,5,9)
v
## [1] 3 1 4 1 5 9
```

- `c(1,2,3,4,5,6)` can be written as `1:6`
- Elements of a vector can be extracted using `[]`.

```
v[3]
## [1] 4
```

Question 1.5. What do you think is the value of `v[c(2,6)]` and `v[3:5]`?

Matrices and their dimensions

- Mathematically, we write $\mathbb{L} = \begin{bmatrix} \ell_{11} & \ell_{12} & \dots & \ell_{1p} \\ \ell_{21} & \ell_{22} & \dots & \ell_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \dots & \ell_{np} \end{bmatrix}$.
- We say $\mathbb{L}$ is a matrix with **dimension** $n \times p$.
- It is common to use n for the number of rows, corresponding to the number of measured units.
- We will see that p is common notation for the number of variables.
- To get the dimension in R, we use the `dim()` function.

```
dim(L)
```

```
## [1] 83  4
```

- We say that `L` is a matrix with `n`=83 rows and `p`=4 columns.

Extracting rows and columns from a matrix

- In R, a single row or column of a matrix is a vector.
- For example, we can extract the first 5 entries for total life expectancy, which is the fourth column of `L`, as follows.

```
L[1:5,4]
```

```
## [1] 60.88 60.23 60.89 60.35 61.05
```

- We can also use row and column names:

```
L[1:5,"Total"]
```

```
## [1] 60.88 60.23 60.89 60.35 61.05
```

- We need `""` so that R knows `Total` is text not a variable name.

Question 1.6. What is going on in this peculiar example?

```
Total <- "Female"
L[1:5,Total]
```

```
## [1] 62.78 62.34 63.04 62.60 63.37
```

Vectors in R are not matrices

- A vector in R has a `length` but not a `dim`.
- We have already found the dimension of the data matrix `L`,

```
dim(L)
```

```
## [1] 83  4
```

- Let's set `y` to be the fourth column of the data and see its dimension:

```
y <- L[,4]
dim(y)
```

```
## NULL
```

- However, `y` does have a `length`, equal to the number of rows:

```
length(y)
```

```
## [1] 83
```

Addition and multiplication of vectors in R

- Let's see what happens when we add two vectors:

```
u <- c(3,1,4)
v <- c(1,5,9)
u+v
```

```
## [1]  4  6 13
```

- Vectors are added **elementwise**.
- They also multiply elementwise:

```
u*v
```

```
## [1]  3  5 36
```

- In R, many functions apply elementwise. This is useful for data analysis: we often want to do the same operation to each data point.

Addition and multiplication of a vector with a scalar

- A single number is called a **scalar** or a **constant**.
- Let's see what happens when we add and muliply with a scalar.

```
u <- c(3,1,4)
a <- 5
u+a
```

```
## [1] 8 6 9
```

- The constant is added to each element of the vector.
- The same rule applies for multiplying a vector and a scalar:

```
u*a
```

```
## [1] 15  5 20
```

- The rules in R aim to be convenient for data analysis. This is often the case, but we still have to learn what R does and check it is doing what we want.

Calculating the increase in life expectancy

- As above, set `y` to be a vector of life expectancy at birth each year, for men and women combined:

```
y <- L[,"Total"]
```

- To obtain the increase in life expectancy each year over the previous year,

```
g <- y[2:length(y)] - y[1:(length(y)-1)]
```

Question 1.7. Write out the expression for the elements `g[1]` and `g[2]` in terms of elements of `y`. This should convince you whether our calculation is correct.

NA and NULL values in R

- Increase in life expectancy is not available for the first year it is measured.

```
length(y)
## [1] 83
length(g)
## [1] 82
```

- To make include this missing value and make the data vectors equal length, let's set the first increase to NA,

```
g <- c(NA,g)
g[1:8]
## [1]    NA -0.65  0.66 -0.54  0.70  1.34  0.68  0.16
```

- Now we've seen two of R's special non-numeric values. NULL means "doesn't exist". NA means "not available" or "missing". Data matrices can have NA entries but not NULL. R tries to treat missing data appropriately.

Logical vectors in R

- Elements of **logical vectors** are `TRUE` or `FALSE`.
- One way to get a logical vector in R is to test for equality or inequality using >, < or ==.
- To find years with decreasing life expectancy:

```
g[1:6]
## [1]    NA -0.65  0.66 -0.54  0.70  1.34
L_down <- g<0
L_down[1:6]
## [1]    NA  TRUE FALSE  TRUE FALSE FALSE
```

- As we might expect, > is calculated elementwise.
- We can index using a logical vector:

```
year <- L[,"Year"]
years_down <- year[L_down]
years_down[1:6]
## [1]   NA 1934 1936 1943 1957 1960
```

Character vectors in R

- Elements of **character vectors** are strings of letters and symbols.
- Qualitative data are often represented as character strings:

```
L_qualitative <- ifelse(g<0,"decreased","increased")
L_qualitative[1:6]
```

```
## [1] NA           "decreased" "increased" "decreased"
## [5] "increased" "increased"
```

Question 1.8. Guess how `ifelse()` works. Is is elementwise?

- You can check the R help on `ifelse()` by typing

```
?ifelse
```

- It takes some practice to learn to read R help effectively!

Checking the class of an R object

- The `class` function tells us what data type R is working with

```
class(g)
## [1] "numeric"
class(L_down)
## [1] "logical"
class(L_qualitative)
## [1] "character"
```

- It is often useful to look at the class of a new R object you have made, to see if it is what you think it is!

R data structures: dataframes and matrices

- A matrix in R must have all entries of the same class. The mathematics of fitting a linear statistical model will require numeric data.
- For example, to convert data to a numeric representation for statistical analysis, `L_down` or `L_qualitative` could be coded using `0` for `FALSE` (or `"increased"`) and `1` for `TRUE` (or `"decreased"`).
- A dataframe in R may have different classes in each column. Data are usually stored in dataframes, e.g., `read.table()` generates a dataframe.

```
class(L)
```

```
## [1] "data.frame"
```

```
L_matrix <- as.matrix(L)
class(L_matrix)
```

```
## [1] "matrix"
```

- For many purposes, dataframes and matrices behave the same.
- R has many ways of working with data like the Innuit have many words for snow (`wikipedia:Eskimo_words_for_snow`).

Getting help with R

- Learning a new computing language is frustrating.
- Try to debug your code for a reasonable time and then start working up the resources until the problem is solved.
- The following order is a guide:

1. The R help, e.g., type `?ifelse` for information on the syntax of `ifelse`.
2. The internet, e.g., google "R ifelse".
3. Classmates.
4. Piazza.
5. Office hours, start-and-end of class, lab.
6. Email to instructor and/or GSI. For email help, please construct and email a simple example demonstrating the issue.

Subsetting matrices in R

- Matrices can be subsetted using logical vectors to pick out subsets of rows and columns.
- Each row or column is included if the logical vector is `TRUE` and excluded if `FALSE`.
- Rows and columns can also be selected using row and column names:

```
colnames(L)
## [1] "Year"   "Female" "Male"   "Total"
rownames(L)[1:8]
## [1] "1" "2" "3" "4" "5" "6" "7" "8"
```

Question 1.9. What is computed below. Can you find any interpretation?

```
L[g<0,"Year"]

##  [1]   NA 1934 1936 1943 1957 1960 1962 1963 1966 1968 1980
## [12] 1985 1988 1993 2015
```

Building a matrix in R

- The basic way to build a matrix is the `matrix()` function.

```
A <- matrix(1:6,nrow=2)
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

- We told R to construct a matrix with elements `1,2,3,4,5,6`.
- `matrix()` fills the matrix by columns.
- R figured out `A` should have 3 columns since we gave `matrix()` 6 numbers and asked for 2 rows.
- If you don't give R enough numbers, it **recycles** the ones you gave it.

```
matrix(c(1,2,3),nrow=2,ncol=3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    2
## [2,]    2    1    3
```

Exercise on using `matrix()` and the recycling rule

Question 1.10. Guess the ouput of

```
matrix(c(0,1),nrow=3,ncol=2,byrow=TRUE)
```

- You can try variations on this exercise. Guess what you think R should produce, see what it does produce, and train your intuition.

Gluing together vectors and matrices

- We've used `c()` to combine numbers into vectors. It also combines vectors into longer vectors.

```
u <- c(1,2) ; v <- c(3,4) ; c(u,v)
```

```
## [1] 1 2 3 4
```

- We can build a matrix by binding together vectors either as rows or columns.

```
B <- rbind(u,v) ; C <- cbind(u,v)
B
```

```
##   [,1] [,2]
## u    1    2
## v    3    4
```

```
C
```

```
##      u v
## [1,] 1 3
## [2,] 2 4
```

Question 1.11. Guess what `cbind(B,C)` and `rbind(B,C)` produce.

Unemployment data for our health economics case study

- The **business cycle** (`Wikipedia:Business_cycle`) consists of alternating periods of higher and lower economic activity.
- Unemployment is **counter-cyclical**: it is higher when economic activity is lower. We'll use Bureau of Labor Statistics unemployment data.

```
download.file(destfile="unemployment.csv",
  url="https://ionides.github.io/401f18/01/unemployment.csv")
```

```
# From http://data.bls.gov/timeseries/LNU04000000
# Percent unemployment, age 16+, not seasonally adjusted
Year,Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
1948,4.0,4.7,4.5,4.0,3.4,3.9,3.9,3.6,3.4,2.9,3.3,3.6
1949,5.0,5.8,5.6,5.4,5.7,6.4,7.0,6.3,5.9,6.1,5.7,6.0
```

```
U <- read.table(file="unemployment.csv",sep=",",header=TRUE)
U[1:2,]
```

```
##   Year Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1 1948   4 4.7 4.5 4.0 3.4 3.9 3.9 3.6 3.4 2.9 3.3 3.6
## 2 1949   5 5.8 5.6 5.4 5.7 6.4 7.0 6.3 5.9 6.1 5.7 6.0
```

Averaging columns in R using apply()

- We want annual average unemployment. For each row, we must average columns 2:13.
- The apply() function is useful for carrying out manipulations on rows and columns of matrices.

```
u <- apply(U[,2:13],1,mean)
u[1:6]
```

```
## [1] 3.766667 5.908333 5.325000 3.333333 3.033333 2.925000
```

- apply() carries out an operation (here, taking the mean) on rows or columns of matrices.
- The middle argument 1 to apply() asks for the function mean() to be applied to each row.
- Setting 2 would give the average over rows for each column.
- Mnemonic: apply(U,1,...) gives a vector of length dim(U)[1], and apply(U,2,...) gives a vector of length dim(U)[2].

Checking the mnemonic for apply(,1,) and apply(,2,)

```
dim(U)
```

```
## [1] 68 13
```

```
length(apply(U,1,mean))
```

```
## [1] 68
```

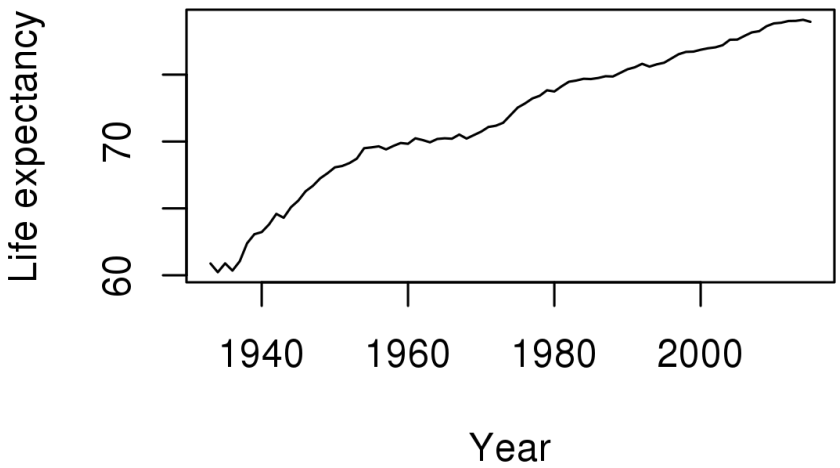
```
length(apply(U,2,mean))
```

```
## [1] 13
```

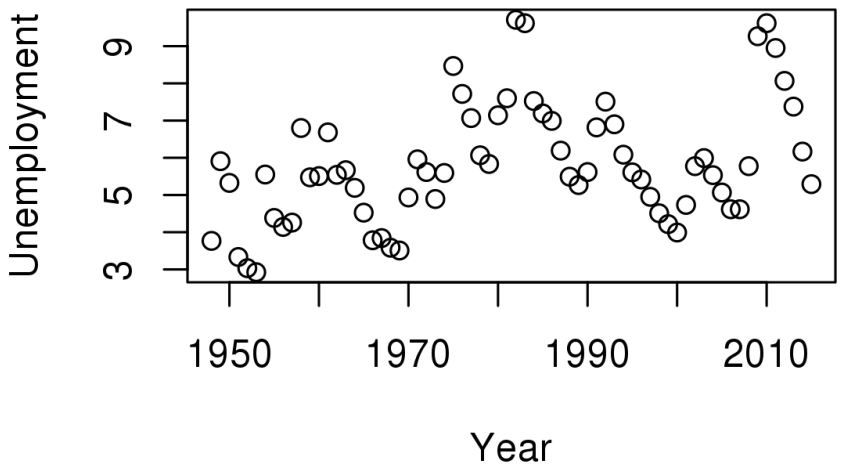
Question 1.12. (a) How many rows are in the monthly unemployment matrix `U`? (b) How many columns does `U` have? (c) Which `apply()` function call takes the average across all the columns in each row of `U`?

Plotting the data

```
plot(L$Year,y,type="line",
  xlab="Year",
  ylab="Life expectancy")
```




```
plot(U$Year,u,
  xlab="Year",
  ylab="Unemployment")
```



- A basic rule of applied statistics is to plot the data.
- Carefully designed plots can reveal secrets in the data: (i) label axes; (ii) lines or points or both; (iii) any other creative ideas?
- In this code, `U$Year` is equivalent to `U[,"Year"]` or `U[,1]`.
- In R, `$` extracts a component of a **list**.
- The list class can collect together objects of any other class.

Lists in R

- Let's learn lists by example

```
my_list <- list(apples=c("red","green"),oranges=c(6,12))
class(my_list)
## [1] "list"
class(my_list$apples)
## [1] "character"
class(my_list$oranges)
## [1] "numeric"
```

- We can index a component of a list using double brackets, `[[.]]`

```
my_list[[1]]
## [1] "red"   "green"
```

- A dataframe is actually a list of vectors. This lets each column have a different class, so we can store both qualitative and quantitative data.
- Each column in a dataframe must have the same length, so R lets us index it like a matrix.

Detrending life expectancy

- Life expectancy has been generally increasing with time. We say it has an **increasing trend**.
- We'll investigate if it is above or below trend during economic booms.
- Subtracting an estimate of the trend from each data point is called **detrending**. A basic way to do this is to fit a linear trend that fits the data best, by finding the line mimizing the sum of squares of distances to the data.
- In this course, we're going to study linear models and their statistical properties in considerable detail.
- Let's first look at a **least squares** fitted line computed using the `lm()` function in R.

Fitting a linear model using `lm()` using formula notation

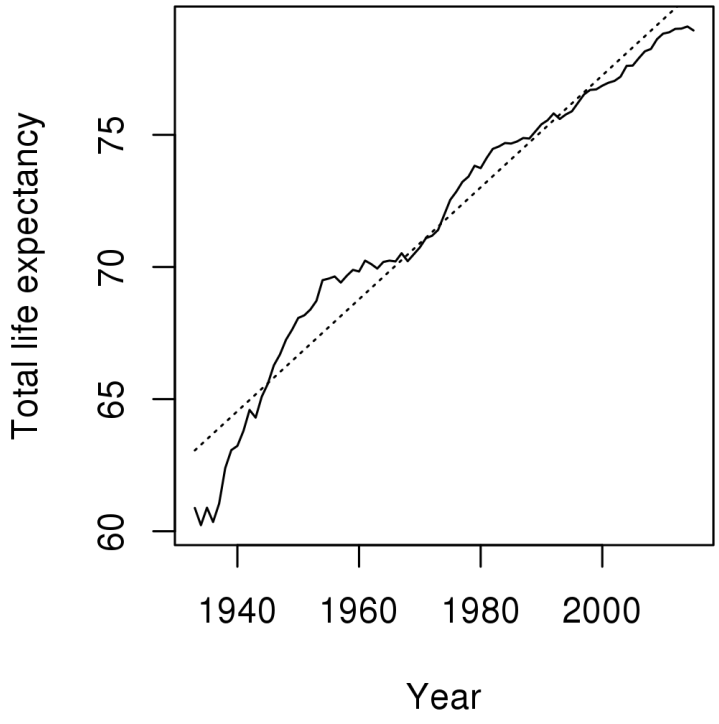
- The `lm()` function has a **formula notation** using a tilde (~).

```
L_fit <- lm(Total~Year,data=L)
```

- The formula `Total~Year` is read as *Total depends on Year*.
- Writing `data=L` tells `lm()` to look for the linear model variables in the column names of the dataframe `L`.
- `lm(L$Total~L$Year)` and `lm(y=L$Total,x=L$Year)` are equivalent.

```
plot(Total~Year,data=L,
  type="l",
  ylab="Total life expectancy")
lines(L$Year,L_fit$fitted.values,
  lty="dotted")
```

- We can use formula notation in `plot()` just like we did in `lm()`.



Exploring the output of `lm`

- We call `L_fit` a **fitted model object** since it is an R object that was created by fitting a model, in this case a linear model fitted using `lm`.
- Let's check the class of this new R object

```
class(L_fit)
```

```
## [1] "lm"
```

- We see that `lm` is both the name of the function to fit a linear model and the class of the resulting fitted model object.
- Now, let's see what the fitted model object contains:

```
names(L_fit)
```

```
##  [1] "coefficients"  "residuals"     "effects"
##  [4] "rank"          "fitted.values" "assign"
##  [7] "qr"            "df.residual"   "xlevels"
## [10] "call"          "terms"         "model"
```

- `L_fit` is a **list** with all the things R thinks we might want to know about the fitted linear model. Thus, in the plot above, `L_fit$fitted.values` gives points on the fitted line.

Computer software notation vs math notation

- Computers compute things. That's what they do. It seems obvious.
- A computer function takes numbers in and spits numbers out. It can't know whether the analysis is correct, or reasonable, or useful for some purpose, or complete nonsense. Artificial intelligence is not (yet) good at applied statistics!
- **For describing statistical assumptions, understanding the behavior of statistical tests, and defining statistical models, mathematics is a more appropriate language than computer code.**
- If all is well, the math helps us understand the computing and vice versa.
- We have seen one example: matrices and vectors are simultaneously (i) mathematical objects, with certain mathematical rules and definitions; (ii) R objects which follow a set of rules inspired by the mathematics.
- How do we mathematically write down the statistical linear model that we fitted using `lm()`?

A linear model – the sample version

- Suppose our data are $y_1, y_2, \ldots, y_n$ and on each unit i we have p explanatory variables $x_{i1}, x_{i2}, \ldots, x_{ip}$. A linear model is

$$\text{(LM1)} \qquad y_i = b_1 x_{i1} + b_2 x_{i2} + \cdots + b_p x_{ip} + e_i \quad \text{for } i = 1, 2, \ldots, n$$

- This is a model for a particular sample $y_1, \ldots, y_n$. A basic task of statistics is to generalize from a sample to a population. We'll do that later.
- The **residual error** terms $e_1, \ldots, e_n$ in equation (LM1) include everything about the data $y_1, y_2, \ldots, y_n$ that cannot be explained by a **linear combination** of the explanatory variables.
- Using **summation notation** we can write the linear model for this sample in a more compact way,

$$\text{(LM2)} \qquad y_i = \sum_{j=1}^{p} x_{ij} b_j + e_i \qquad \text{for } i = 1, 2, \ldots, n$$

- We'll review summation notation in due course.

Applying the linear model to detrend life expectancy

- When we did `L_fit<-lm(Total~Year,data=L)` earlier, we fitted the linear model (LM1) with y_i being the total life expectancy for the ith year in the dataset and x_{i1} being the corresponding year.
- To fit a linear trend, we also want an **intercept**, which we can write by setting $x_{i2} = 1$ for each year i.
- In this special case, with $p = 2$ variables and $x_{i2} = 1$, the model (LM1) is called the **simple linear regression** model,

$$\text{(SLR1)} \qquad y_i = b_1 x_{i1} + b_2 + e_i \quad \text{for } i = 1, 2, \ldots, n$$

- Here, b_2 is the intercept for the **fitted line** $y_i = b_1 x_{i1} + b_2$ when we ignore the residual errors $e_1, \ldots, e_n$.
- In `L_fit<-lm(Total~Year,data=L)`, we gave R the task of finding the values of the **coefficients** b_1 and b_2 which minimize the **residual sum of squares**, $\sum_{i=1}^{n} e_i^2$.
- We didn't have to tell R we wanted an intercept. By default, it assumed we did. In this case it was right.

Is unemployment rate assosciated with mortality?

- Now, we'll fit another linear model to see if the detrended life expectancy can be explained by the level of economic activity, quantified by the unemployment rate.
- We have seen that `residuals` is one of the components of an `lm` object, by looking at `names(L_fit)`.
- **Residual** is a more polite name than **residual error**. That is appropriate here, since the "error" e_i is exactly the deviation from trend that we are interested in. Interpretation of e_i depends on the particular situation.
- First, let's set up the variables for the regression. Since we detrended life expectancy, we should also detrend unemployment. Then, we use `%in%` to find the years when these two datasets match and `subset()` to select these years from `L_detrended`.

```
L_detrended <- L_fit$residuals
U_detrended <- lm(u~U$Year)$residuals
L_detrended <- subset(L_detrended,L$Year %in% U$Year)
```

A linear model linking mortality and unemployment

```
lm1 <- lm(L_detrended~U_detrended)
coef(lm1)

## (Intercept) U_detrended
##   0.2899928   0.1313673
```

- We have obtained a positive coefficient for the sample linear model. Higher unemployment seems to be associated with higher life expectancy. This may be surprising. Is the result **statistically significant**? What happens if we use a different explanatory variable instead of unemployment? Or if we use more than one explanatory variable? Are there any violations of statistical assumptions that might invalidate this analysis? Is it reasonable to make a causal interpretation (that economic cycles cause fluctuations in life expectancy) or must we limit ourselves to a claim that these variables are associated?
- Giving informed answers to statistical questions such as these is a primary goal of the course.