GEOG 6000 Lab 03 Modeling I

Simon Brewer

September 4, 2017

This lab covers two things:

- Further plotting in R
- How to build basic models in R and how to perform some simple diagnostics.

We will be using the following files for these examples:

- A simple dataset with two variables: *regrex1.csv*
- A dataset of death rates in Virginia according to age, sex and social groups *VADeaths.csv*
- Simulations of annual global temperature from the fourth IPCC report *ipccScenario_1900_2100.csv*
- A DEM of Maunga Whau volcano *volcanodem.txt*

Things to remember:

- The following font conventions are used:
  - Normal font = text, comments etc
  - `Courier font` = R commands
  - *Italic font* = file names
- R is case-sensitive for variables and functions. Iris is not the same as iris or IRIS.
- `#` indicates comments - no need to type these
- **Remember to change your R working directory to where your files are stored**

With all the examples given, it is important to not just type in the code, but to try changing the parameters and re-running the functions multiple times to get an idea of their effect. Help on the parameters can be obtained by typing `help(functionname)` or `?functionname`.

Other plot types

Some more advanced plotting examples are given here to give further demonstration of plotting multiple data series, including series with different scales, plotting matrices as raster images or as contours. Only a basic description of each plot is given, so use the `help()` function to look at any functions that you are not familar with. With any time remaining, start working through these examples and don't hesitate to ask for help as you go.

Bar plots

This makes a simple barplot using a dataset of Virginia death rates by age and urban/rural split. The colors for the bars are generated using `heat.colors()` which creates a color palette. Look at the help for `barplot()` to see options to arrange the bars differently.

```
VADeaths <- read.csv("VADeaths.csv", row.names=1)
mycol <- heat.colors(5)
barplot(as.matrix(VADeaths), beside=T, legend = rownames(VADeaths),
    col = mycol)
```

Dot charts

We have already used dot charts to visualize a small set of observations. These also adapt to having two different categories, as with the Virginia dataset. We need to convert the dataframe to a matrix, then can plot it. By transposing the matrix in the second example, we can change the hierarchy of the groupings.

```r
VADeaths.m <- as.matrix(VADeaths)
dotchart(VADeaths.m)
dotchart(t(VADeaths.m))
```

Plotting multiple series with points

In this example, we will plot death rate by age class but with a different series for each combination of sex and urban/rural. We start by generating a set of age class midpoints, then add the series with different colors and symbols, and finally add a legend explaining the points. Note that the plot does not adjust as extra data series are added, so we need to manually set the limits of the y-axis in the first `plot()` function.

```r
cmp <- c(52,57,62,67,72)
plot(cmp, VADeaths$Rural.Male, pch=1, col=1, ylim=c(0,70),
    xlab='Age Class', ylab='Mortality', main='Virginia Death Rates')
points(cmp, VADeaths$Rural.Female, pch=2, col=2)
points(cmp, VADeaths$Urban.Male, pch=3, col=3)
points(cmp, VADeaths$Urban.Female, pch=4, col=4)
legend("topleft",legend=c("Rural Male","Rural Female","Urban Male","Urban Female"),
    col=c(1,2,3,4), pch=c(1,2,3,4))
```

Plotting multiple series with lines

The file *ipccScenario_1900_2100.csv* contains the results of simulations of annual global temperature from 1900 to 2100 under a range of socio-economic scenarios (commit, b1, a1b, a2). The values are deviations from temperature in the year 2000. Each scenario has a median value and values representing the range of possible values (high/low). We will make a plot showing the four median simulated temperatures.

```r
ipcc <- read.csv("ipccScenario_1900_2100.csv")
plot(ipcc$yrs, ipcc$commitMed, type='l',lwd=2,col='orange',
    ylim=c(-1.0,3.5), main='IPCC Scenarios', xlab='Years',ylab='Global Temp.')
lines(ipcc$yrs, ipcc$b1Med, lwd=2, col='blue')
lines(ipcc$yrs, ipcc$a1bMed, lwd=2, col='green')
lines(ipcc$yrs, ipcc$a2Med, lwd=2, col='red')
## Add reference line at zero (h for horizontal)
abline(h=0, lty=2)
legend("topleft",legend=c("Commit","B1","A1B","A2"),
  lty=1, lwd=2, col=c('orange','blue','green','red'))
```

Plotting polygons

Here, we use a polygon to plot the range of one of the scenarios of temperature change, and then overlay the median value as a line. We need to create the full set of coordinates for the polygon, which may be obtained by plotting in the order of the series for the low range, then reversing the order of the high range.

```r
plot(ipcc$yrs, ipcc$commitMed, type='n', ylim=c(-1.5,1.0),
    main='Commit Scenario', xlab='Years',ylab='Global Temp.')
polygon(c(ipcc$yrs,rev(ipcc$yrs)),c(ipcc$commitLo,rev(ipcc$commitHi)), col='orange')
lines(ipcc$yrs, ipcc$commitMed, lwd=2, col='black')
```

Polygons can also be filled with shading lines, rather than a block color.

```r
## Same thing but with shading lines in polygon, rather than fill
plot(ipcc$yrs, ipcc$commitMed, type='n', ylim=c(-1.5,1.0))
polygon(c(ipcc$yrs,rev(ipcc$yrs)),c(ipcc$commitLo,rev(ipcc$commitHi)),
    col='orange', density=20, angle=210)
lines(ipcc$yrs, ipcc$commitMed, lwd=2, col='orange')
```

This allows you to overplot several polygons and see where they overlap. If you're feeling up to the challenge, try adding polygons showing the range of the other three scenarios (changing the angle of the shading lines will help here).

Plotting images

R has a series of functions for plotting matrices, in a way similar to plotting raster images. This will use a small text file with elevation information for the Maunga Whau volcano. Start by reading this, then creating the coordinates (x and y) and a matrix `z`, with the elevation multiplied by 2 to exaggerate the relief. Note that we will look at better functions for plotting spatial data later in the semester

```r
volcano <- read.table("volcanodem.txt")
## Convert to matrix
volcano <- as.matrix(volcano)
## Create coordinates (10m resolution)
z <- 2 * volcano        # Exaggerate the relief
x <- 10 * (1:nrow(z))   # 10 meter spacing (S to N)
y <- 10 * (1:ncol(z))   # 10 meter spacing (E to W)
```

- Plot image

```r
image(x,y,z)
```

- Plot image with better color scale

```r
image(x,y,z,col=terrain.colors(100))
```

- Contour plot

```r
contour(x,y,z,nlevel=20)
```

- Perspective plot (`theta` controls the horizontal view angle, `phi` the vertical angle)

```r
persp(x,y,z,theta=210, phi=15, scale=FALSE)
```

- Shaded perspective plot (`ltheta` controls the light angle, `shade` the diffusion of the lighting)

```r
persp(x,y,z,theta=130, phi=30, scale=FALSE,
      col='green3', ltheta=-120, shade=0.75,
      border=NA, box=FALSE)
```

Graphic output

As a reminder, you can output any figure to a file, rather than on screen as follows:

```r
pdf('volcano.pdf')
image(x,y,z,col=terrain.colors(100), main="Maunga Whau DEM")
dev.off()
```

Alternatively, you can copy-paste directly into Word:

- Mac OSX: select graphic window, then copy-paste from Edit menu. You can also save directly from the file menu if the graphics window has been selected
- Windows: copy-paste directly from the graphic window. You can also save directly as a png, pdf, etc from here

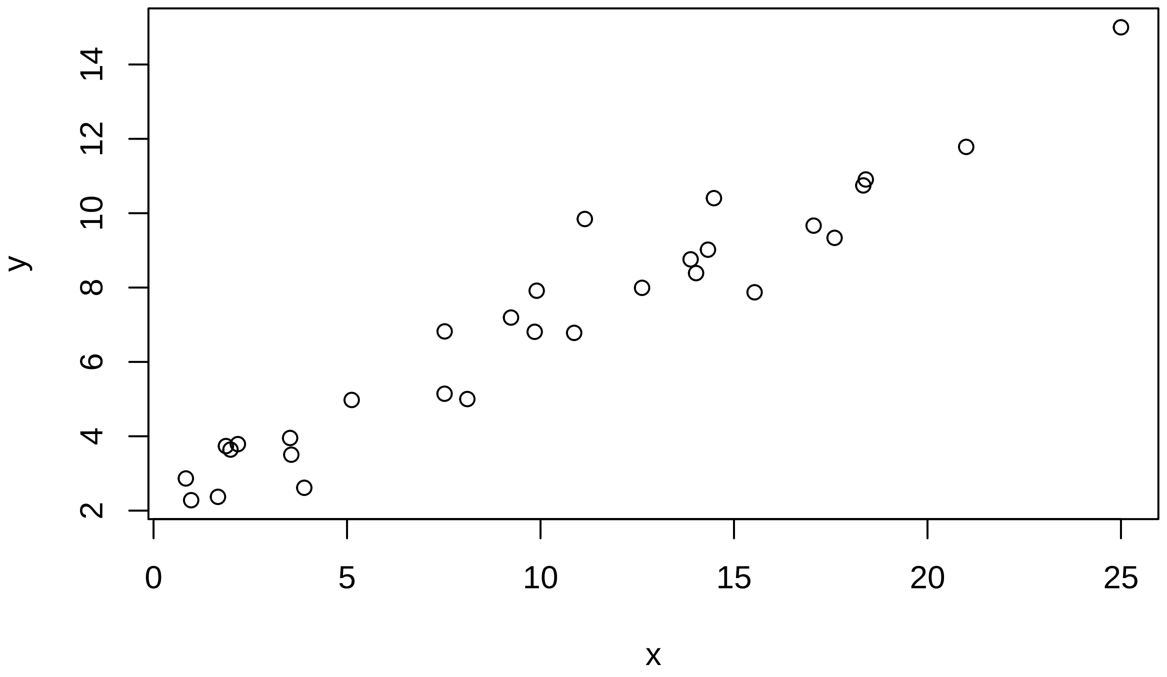
Simple Linear Regression

For this exercise, we will use the file *regrex.csv*, which contains two variables, x and y. Before starting the model, read in the dataset, calculate summary statistics and make a simple scatter plot. Note that we use the formula syntax (∼) to relate the dependent and independent variables.

```
regrex <- read.csv("regrex.csv")
summary(regrex)
```

```
##        y                x
##  Min.   : 2.281   Min.   : 0.8347
##  1st Qu.: 3.830   1st Qu.: 3.6437
##  Median : 7.006   Median : 9.8750
##  Mean   : 6.971   Mean   :10.0678
##  3rd Qu.: 9.257   3rd Qu.:14.4428
##  Max.   :15.000   Max.   :25.0000
```

```
plot(y ~ x, data=regrex)
```



And we can test for correlation between the two variables as follows:

```
cor.test(regrex$x, regrex$y)
```

```
##
##  Pearson's product-moment correlation
##
## data:  regrex$x and regrex$y
## t = 19.194, df = 28, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.9250459 0.9829233
## sample estimates:
##       cor
## 0.9640353
```

Model building

The scatterplot and correlation show an obvious linear relationship, so we can now go ahead and construct a simple model:

```
ex1.lm <- lm(y ~ x, data=regrex)
```

To look at the output of the model, simply type the name of the object. This will show the call (what you asked for in the model) and the cofficients for the slope and intercept.

```
ex1.lm
```

```
##
## Call:
## lm(formula = y ~ x, data = regrex)
##
## Coefficients:
## (Intercept)            x
##      2.2481       0.4691
```

To just obtain the coefficients of the model, type:

```
coef(ex1.lm)
```

Model diagnostics

We'll now go through the steps of running model diagnostics

ANOVA with linear models

The first diagnostic to run is an ANOVA on the model fit. This uses the ratio of explained/unexplained variance to see if more variance is captured by the model, than is left in the residuals. This ratio is based on the mean sum of squares explained by the model and the remaining mean sum of squares in the residuals. We will use the `anova()` to calculate the ANOVA, as the output is slightly easier to work with than the `aov()` fucntion we previously used:

```
anova(ex1.lm)
```

```
## Analysis of Variance Table
##
## Response: y
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## x          1 283.947 283.947   368.4 < 2.2e-16 ***
## Residuals 28  21.581   0.771
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Look through the output and try to find the following values:

- The sum of squares in the model and the residuals
- The mean sum of squares
- The degrees of freedom
- The F-statistic
- The p-value

The `summary()` function

Other model diagnostics can be obtained by using the `summary()` function. This adapts to the model object and produces a fairly long list of output:

```
summary(ex1.lm)
```

```
##
## Call:
```

```
## lm(formula = y ~ x, data = regrex)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.66121 -0.53286 -0.02869  0.50436  2.36786
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.24814    0.29365   7.656 2.44e-08 ***
## x            0.46906    0.02444  19.194  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8779 on 28 degrees of freedom
## Multiple R-squared:  0.9294, Adjusted R-squared:  0.9268
## F-statistic: 368.4 on 1 and 28 DF,  p-value: < 2.2e-16
```

Look for the following:

- A summary description of the residuals (look to see that the median is close to zero, and the 1st and 3rd quantiles are approximately equal)
- The standard error and significance of the coefficients (based on a t-test)
- The amount of variance explained (R^2)

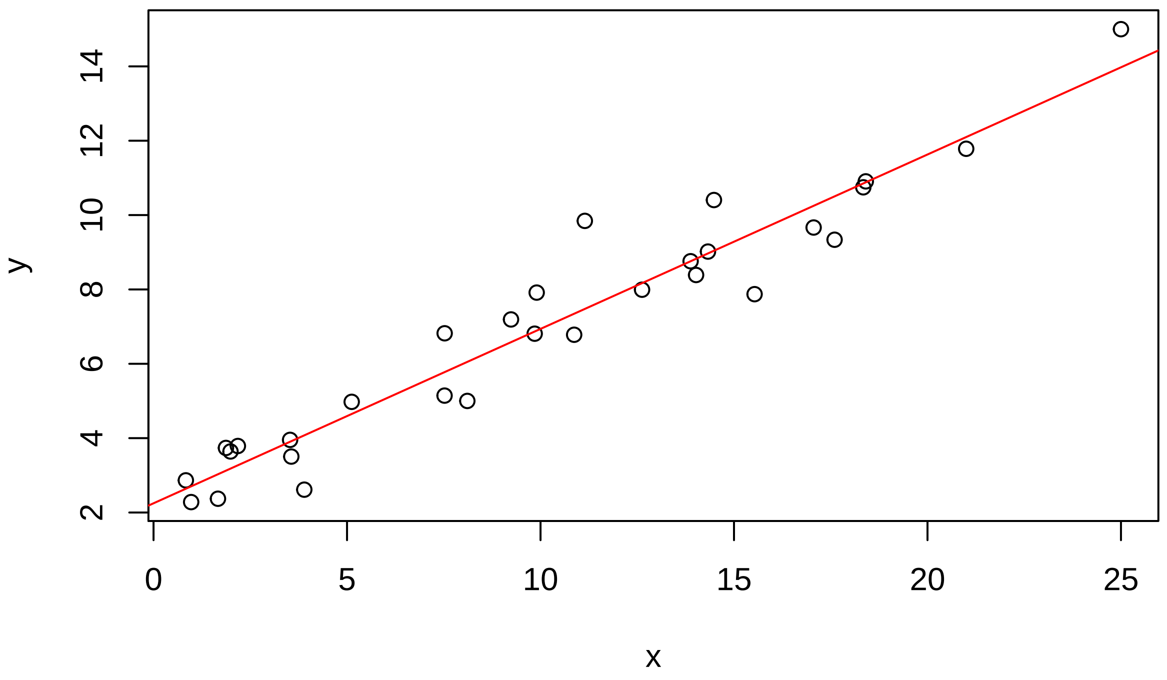
Note that the results of the F-test from the ANOVA are given on the last line.

- Do you think this is a good model?

Linearity check

A first and simple check on the model is simply to plot the model line through the observations. The function `abline()` allows you to add straight lines to a plot; if it is used with a model object, then it will plot the model line (this only really works for bivariate models). You can also use it to plot horizontal and vertical lines on a plot. Note the `col` parameter that allows you to change the color of the line. This can be used with most plotting commands to distinguish different series on the same plot.

```r
plot(y ~ x, data=regrex)
abline(ex1.lm, col="red")
```



As we have built a linear model, we expect that the data show a linear relationship. If the data points are scattered randomly above and below the line, this supports a linear relationship.

Model standard errors

Standard errors for the model fit can be generated using the `predict()` function, with the parameter `int="confidence"`. Note that these are the model errors, giving a confidence interval within which the model most likely falls. The range of this confidence interval is set using the parameter `level`, here set to 0.95 or 95% CI. The output of the `predict()` function will be a three column matrix with predicted value (`fit`) and lower (`lwr`) and upper (`upr`) intervals.

```
predict(ex1.lm, level = 0.95, interval = "conf")
```

To add confidence intervals to the model plot, we can make predictions for a new set of values of 'x'. To do this, we need to create a new dataframe, with a variable that has the same name as the variable used in the model ('x'). For our plot, we create values between -1 and 30 using the `seq()` function, then use this in the `predict()` function as the new data.

```
newx <- data.frame(x=-1:30)
newy <- predict(ex1.lm, newdata=newx, level = 0.95, interval = "conf")
str(newy)
```

We can now add these to the plot of the model line to show the confidence intervals using the `lines()` function, which allows multiple lines to added. As well as using `col` here to change color, we use the `lty` parameter to added dashed lines to an existing figure.

```
plot(y ~ x, data=regrex, pch=16, main='Model confidence intervals')
lines(newx$x, newy[,"fit"], col=2)
lines(newx$x, newy[,"lwr"], col=3, lty=2)
lines(newx$x, newy[,"upr"], col=3, lty=2)
```

Residual plots

Next, we will make a series of plots with the model residuals. The assumption behind the model is that these should be normally distributed and independent. So we will look for both normality and patterns in the residuals. Examples of patterns are clusters of negative or positive residuals when plotted.

Model residuals can be extracted using the `residuals()` function and the model object, allowing us to use thhese in tests and plots.

```r
ex1.res = residuals(ex1.lm)
```

Start by making a histogram. This should be approximately symmetrical and resemble a normal distribution.

```r
hist(ex1.res)
```

Plot out the residuals to look for biases in the model fit. Here, we use the `plot()` function with the model object created earlier. This has a parameter `which` that chooses the type of plot to make. The first option is a plot with the fitted values on the x-axis and the residuals on the y-axis (`which=1`). This also fits a trend line (red), which should be roughly horizontal in unbiased residuals. Do the residuals show any trends or biases?

```r
plot(ex1.lm, which=1)
```

Next, we'll make a QQ-plot of the residual, and add the QQ-line (`which=2`). This plots each residual as its quantile position in the dataset (y-axis) against the expected position of that value from a theoretical normal distribution with the same mean and s.d. A straight line is added which is the expected match if the data are normally distributed. If the points fall more or less on this line, the residual are likely normally distributed.

```r
plot(ex1.lm, which=2)
```

Finally, we'll make a plot of Cook's distance for each observation. This is a measure of the leverage of each point, or its importance in the estimating the model, and is based on how isolated a given observation is in the dataset. The vertical lines represent the distance (D), and the bigger the distance the greater the influence.

```r
plot(ex1.lm, which=4)
```

Shapiro-Wilk test

One of the inference tests that we looked at tests for the assumption of normality in a set of data. This can be useful if you have any doubts about the residuals of your model. We will run this here, even though the visual examination does not suggest any evidence for non-normality

```r
shapiro.test(ex1.res)
```

The null hypothesis is that the data *are* normally distributed. Here we get a high p-value, and no evidence to reject the null.

Prediction with a simple model

Now use the model you have created to predict new values of y for values of x using the `predict()` function. As before, we need to make a new dataframe containing a variable(s) with the same name(s) as the original data. Predictions for new values of 'x' can then be made with the `predict()` function.

```r
newx <- data.frame(x=1:25)
pred.int <- predict(ex1.lm, int="prediction", newdata=newx)
```

By including the parameter `int='prediction'`, R calculates the 95% prediction confidence intervals (note that these are not the same as the model confidence intervals calculated earlier). The output of the `predict()` function is a three column matrix with predicted value and lower and upper bounds, and these can added to the plot using the `lines()` function.

```r
plot(y ~ x, data=regrex)
lines(newx$x, pred.int[,1])
lines(newx$x, pred.int[,2], lty=2, col=2)
lines(newx$x, pred.int[,3], lty=2, col=2)
```

- Try plotting 99% confidence intervals by changing the `level` parameter in the `predict()` function

Exercises

1. The file *statedata.csv* contains state data for the U.S., published by the U.S. Department of Commerce in 1977.

- Make a scatterplot of murder rate (explanatory) and life expectancy (response)
- Build a linear model between these two variables, using the `lm()` function. Give the code you used in this model and the null and alternate hypotheses that the model is testing
- Use the `summary()` function to report the coefficients of the fit and their significance as p-values
- Use the `anova()` function to examine the goodness-of-fit of the mode, report the F-statistic and it's significance
- Add the model fit line to the scatterplot
- Produce a plot of residuals against fitted values and comment on any bias you see
- Use the `predict()` function and this model to estimate the life expectancy if the murder rate dropped to zero. Use the parameter `int='pred'` to obtain 95% confidence intervals about this prediction
- Do you think that murder has a direct effect on general life expectancy? (Not just on that of the victims…)

Files used in lab

Simple regression data set: *regrex1.csv*

Column header	Variable
y	Dependent variable
x	Independent variable

Virginia mortality dataset: *VADeaths.csv*

Column header	Variable
X	Site elevation
Age class	Age classes (5yr bins)
Rural.Male	Rural male death rates per 1000
Rural.Female	Rural female death rates per 1000
Urban.Male	Urban male death rates per 1000
Urban.Female	Urban female death rates per 1000

IPCC AR4 temperature simulations *ipccScenario_1900_2100.csv*

Column header	Variable
yrs	Year
commitLo	Commit scenario low estimate
commitMed	Commit scenario median estimate
commitHi	Commit scenario high estimate
b1Lo	B1 scenario low estimate
b1Med	B1 scenario median estimate
b1Hi	B1 scenario high estimate
a1bLo	A1B scenario low estimate
a1bMed	A1B scenario median estimate
a1bHi	A1B scenario high estimate
a2Lo	A2 scenario low estimate
a2Med	A2 scenario median estimate
a2Hi	A2 scenario high estimate

IPCC CO2 concentration scenarios: *tar-isam_CO2.csv*

Column header	Variable
Year	Year
Cols 2-12	CO2 concentration under different emission scenarios

Maunga Whau DEM: *volcanodem.txt*

Flat ASCII format, with each value representing elevation in a single 10x10 m cell

US State data set: *statedata.csv*

Column header	Variable
State	Dependent variable
Population	Population estimate (1975)
Income	Per capita income (1974)
Illiteracy	Illiteracy (%age, 1970)
Life.Exp	Life expectancy in years (1969-71)
Murder	Murder rate per 100K population (1976)
HS.Grad	Percent high-school graduates (1970)
Frost	Number of days < 32F in largest city (1931-60)
Area	Area of state (square miles)

R code covered in lab

R Command	Purpose
Convert object types	
`barplot`	Barplot using vector of heights
`dotplot`	Simple index plot
`abline`	Add reference line to figure
`points`	Add points to figure
`lines`	Add lines to figure
`polygon`	Add filled polygon to figure
`image`	Heatmap of matrix
`contour`	Contour plot of matrix
`persp`	3D wireframe plot
`lm(y ~ x)`	Model y as a function of x. Note that the '$\sim$' dictates the nature of the relationhsip. Dependent variable(s) on the left hand side, independent on the right hand side
`summary(model)`	Provides summary of model fit, including coefficients, R^2, F-statistic, etc
`coef`	Extract coefficients of model
`residuals`	Extract residuals of model
`fitted`	Extract predicted y values for observations
`abline`	Plot model line (only really works for bivariate regression)
`predict`	Predict for new values of x. NB: requires a data frame with a variable having the same name as the original x.