Math 158
Jo Hardin
Simple Linear Regression
R code

1. Data should be either in comma delimited (`sep=","`) or tab delimited (`sep="\t"`) format. Data should live in the **same** directory as the R program (`.RData`). (Or import data using RStudio.)

   ```
   > oldfaith <- read.table("oldfaithful.csv", header=T, sep=",")
   > dim(oldfaith)
   > names(oldfaith)
   ```
   Notice that the variable names are `INTERVAL` and `DURATION`.

   ```
   > attach(oldfaith)
   ```
   When we `attach` we create new variables from the data set. That is, we can now use `INTERVAL` and `DURATION` as their own variables / vectors.

   ```
   > oDUR <- order(DURATION)  # needed for prediction lines
   ```
   The `order` command simply tells us which values are smallest to largest (ordered).

2. It's always a good idea to plot the data to see the visual representation.
   ```
   > plot(DURATION, INTERVAL, pch=19)
   ```

3. Running the model. We use the `lm` command, which takes as its argument `y~x`.
   ```
   > oldfaith.lm <- lm(INTERVAL ~ DURATION)
   ```

   ```
   > summary(oldfaith.lm)
   ```

   ```
   Call:
   lm(formula = INTERVAL ~ DURATION)


   Residuals:
       Min      1Q  Median      3Q     Max
   -14.644  -4.440  -1.088   4.467  15.652


   Coefficients:
               Estimate Std. Error t value Pr(>|t|)
   (Intercept)  33.8282     2.2618   14.96   <2e-16 ***
   DURATION     10.7410     0.6263   17.15   <2e-16 ***
   ---
   Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1


   Residual standard error: 6.683 on 105 degrees of freedom
   Multiple R-squared: 0.7369,     Adjusted R-squared: 0.7344
   F-statistic: 294.1 on 1 and 105 DF,  p-value: < 2.2e-16
   ```

```
> anova(oldfaith.lm)
Analysis of Variance Table

Response: INTERVAL
           Df  Sum Sq Mean Sq F value    Pr(>F)
DURATION    1 13133.0 13133.0  294.08 < 2.2e-16 ***
Residuals 105  4689.0    44.7
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

4. To predict mean values, use the sub-command `interval="confidence"`. To predict future values, use the sub-command `interval="prediction"`.

```
> predict.lm(oldfaith.lm, newdata=data.frame(DURATION=c(2.2,3.8)),
 se.fit=T,interval="confidence")

> predict.lm(oldfaith.lm, newdata=data.frame(DURATION=c(2.2,3.8)),
 se.fit=T,interval="prediction")
```

   To draw prediction lines on the scatterplot, plot `DURATION` versus the mean or predicted upper and lower bounds (in the 2nd and 3rd columns of the output). Try the commands below without the oDUR subsetting, and you should get zig-zag lines. That's because the `lines` plotting command always "connects the dots".

```
> plot(DURATION, INTERVAL, pch=19)
> abline(oldfaith.lm)
> lines(DURATION[oDUR], predict.lm(oldfaith.lm, interval="confidence")[oDUR,2],
    col="red")
> lines(DURATION[oDUR], predict.lm(oldfaith.lm, interval="confidence")[oDUR,3],
    col="red")
> lines(DURATION[oDUR], predict.lm(oldfaith.lm, interval="predict")[oDUR,2],
    col="green")
> lines(DURATION[oDUR], predict.lm(oldfaith.lm, interval="predict")[oDUR,3],
    col="green")
```

5. Residual analysis. Read in the data just like above (again, it is comma delimited).

```
> wineheart <- read.table("wineheart.csv", header=T, sep=",")
> dim(wineheart)
> names(wineheart)
> attach(wineheart)

> wine.lm <- lm(MORTALITY ~ WINE)
```

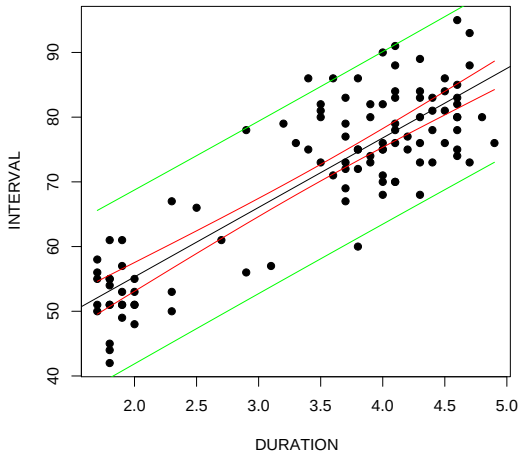

Figure 1: The interior lines represent 95% confidence bounds on the mean values. The exterior lines represent 95% confidence bounds on the future predicted values.

6. Below is the code for various residual plots. Note that

   - `win.graph()` opens a new graphics window.
   - `par(mfrow=c(2,2))` creates a 2x2 array of plots that fill in by row. `par(mfcol=c(2,3))` creates a 2x3 array of plots that fill in by column.
   - `abline` adds a line to the plot of the form "a" "b" (that is, with a slope b and an intercept a). The function can also draw vertical and horizontal lines ("v" and "h").
   - The `log` function is actually the natural log.

```
> par(mfrow=c(2,2))
> plot(WINE, MORTALITY, pch=19)
> abline(wine.lm)


> plot(fitted(wine.lm),resid(wine.lm),xlab="fitted",ylab="residuals",pch=19)
> abline(h=0)


> plot(fitted(wine.lm),rstandard(wine.lm), xlab="fitted",
  ylab="standardized resid",pch=19)
> abline(h=0)


> plot(fitted(wine.lm),resid(wine.lm) / summary(wine.lm)$sigma,
 xlab="fitted",ylab="semistudentized resid",pch=19)
> abline(h=0)
```

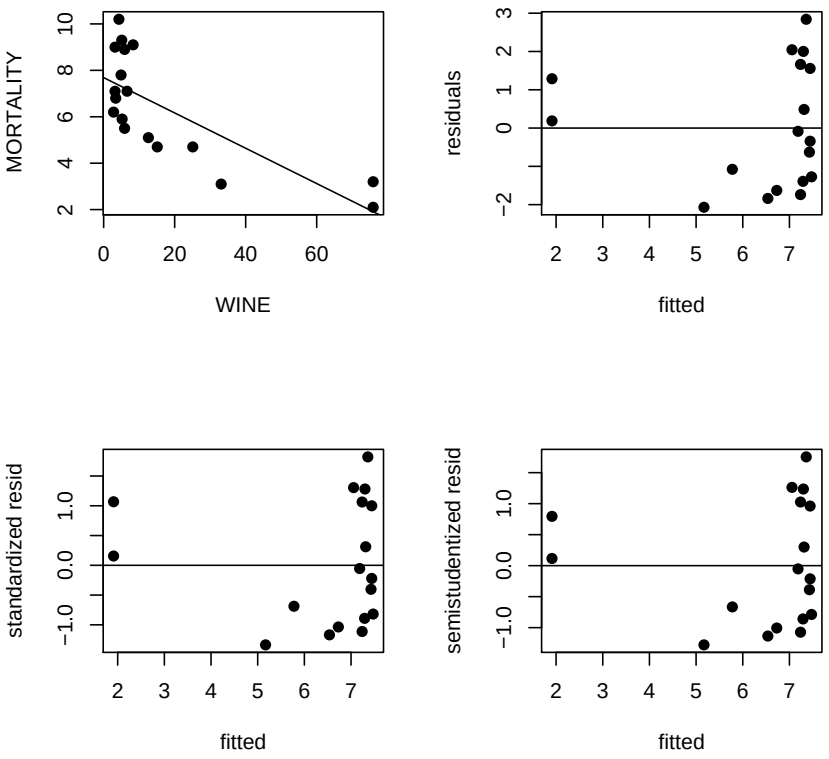

Figure 2: Note that the residuals tell the same story regardless of the scaling.

```
> win.graph()
> par(mfcol=c(2,3))
> wine.lm2 <- lm(MORTALITY ~ log(WINE))

> plot(log(WINE), MORTALITY, pch=19)
> abline(wine.lm2)

> plot(fitted(wine.lm2),rstandard(wine.lm2),
 xlab="fitted",ylab="standardized resid",pch=19)
> abline(h=0)

> wine.lm4 <- lm(log(MORTALITY) ~ WINE)

> plot(WINE, log(MORTALITY), pch=19)
> abline(wine.lm4)

> plot(fitted(wine.lm4),rstandard(wine.lm4),
 xlab="fitted",ylab="standardized resid",pch=19)
> abline(h=0)

> wine.lm3 <- lm(log(MORTALITY) ~ log(WINE))

> plot(log(WINE), log(MORTALITY), pch=19)
> abline(wine.lm3)

> plot(fitted(wine.lm3),rstandard(wine.lm3),
 xlab="fitted",ylab="standardized resid",pch=19)
> abline(h=0)
```

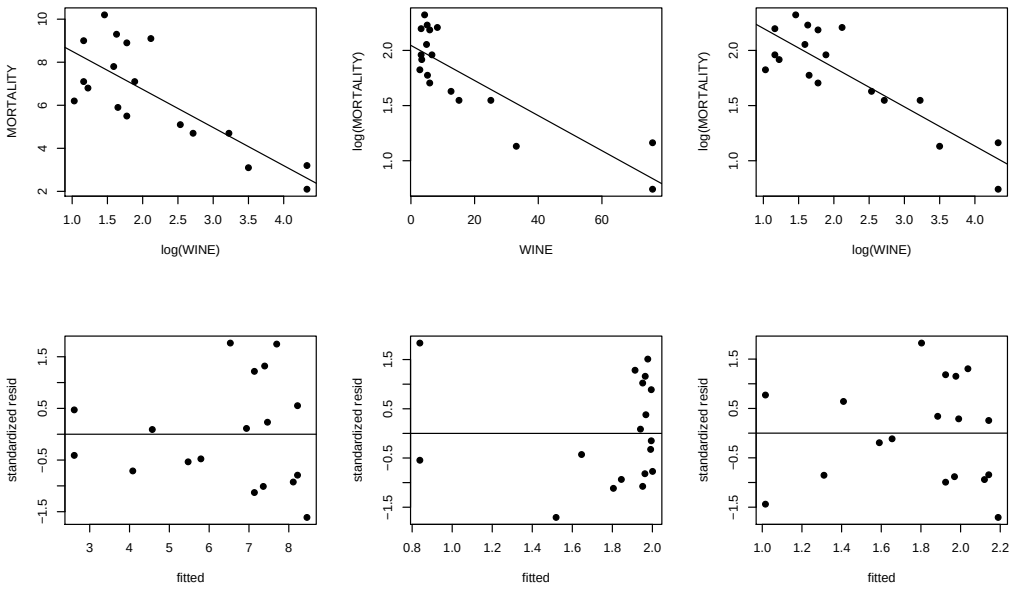

Figure 3: Transforming X (WINE) removes the effect of the extreme value(s). Transforming Y (MORTALITY) creates more constant variance.