Statistical Software Camp: Introduction to R

Day 2
Descriptive Statistics and Graphing

January 27, 2009

1 Descriptive Statistics and Graphing for Univariate Data

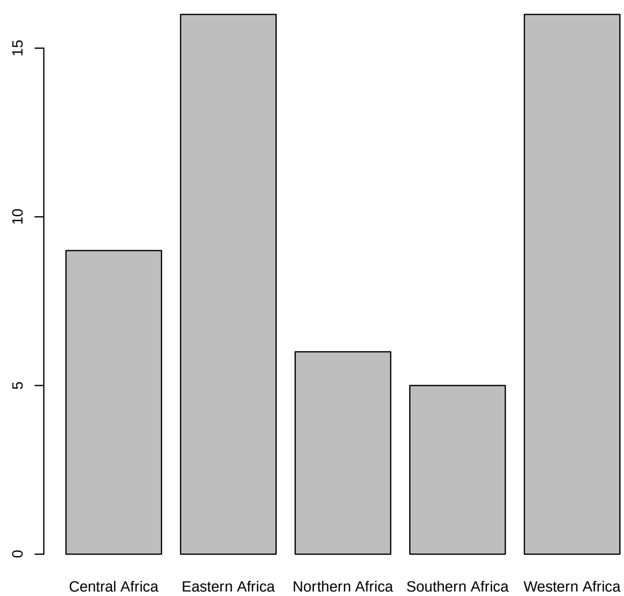
1.1 Simple Graphs for Univariate Data

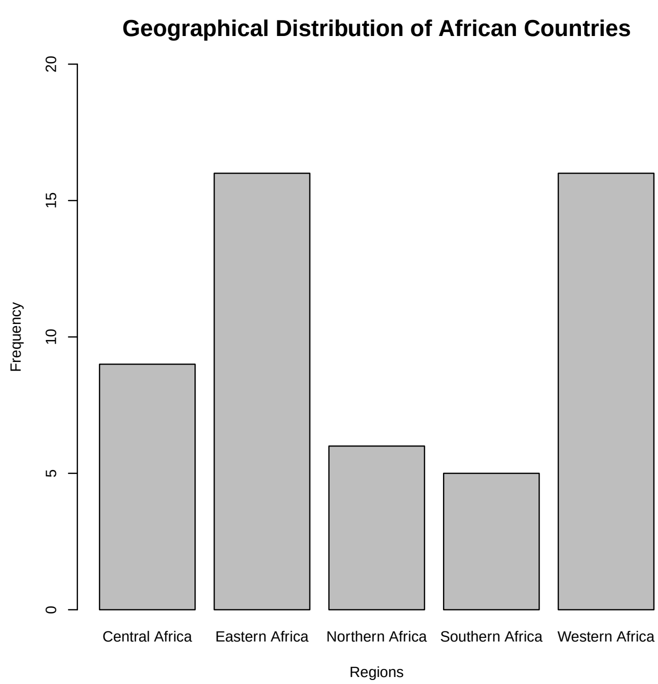
- Great graphics: Easy to understand the "story" without much explanation or eye-balling
- Bad graphics:
  - Inefficient (leaving out easy-to-add information)
  - Potentially misleading (leaving the reader with the wrong impression)
  - Too complicated (taking too much time to understand)
- The function `barplot()` will produce a **barplot** figure
- The function `pie()` will produce a **pie chart** figure
- Specify **arguments** within each function to tweak the graphs to our exact specifications
- To learn the arguments for a particular command, use the help function (e.g., `?barplot` or `help("pie")`)

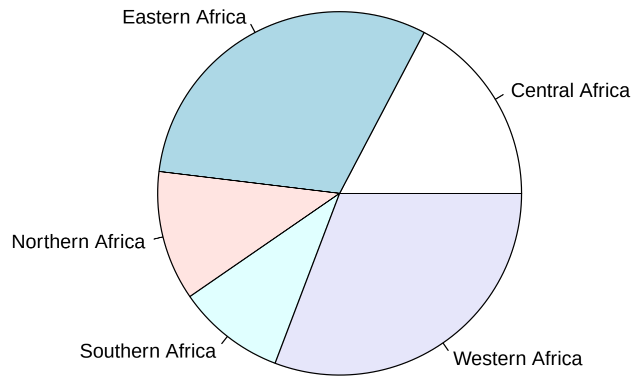
`main =`	Title to put on the graphic.
`xlab =`	Label for the x-axis. Similarly for `ylab`.
`xlim =`	Specify the x-limits, as in `xlim = c(0,10)`, for the interval $[0, 10]$. Similar argument for the y-axis is `ylim`.
`type =`	Type of plot to make. Use `"p"` for points (the default), `"l"` for lines, and `"h"` for vertical lines.
`pch =`	The style of point that is plotted. This can be a number or a single character. Numbers between 0 and 25 give different symbols. The command `plot(0:25,pch = 0:25)` will show those possible.
`lty =`	When lines are plotted, specifies the type of line to be drawn. e.g., `"solid"`, `"dashed"`, `"dotted"` (See `?par` for full details.)
`lwd =`	The thickness of lines. Numbers bigger than 1 increase the default.
`col =`	Specifies the color to use for the points or lines, e.g., `"blue"`, `"red"`, `"yellow"`.

Table 1: Useful arguments for `plot()` and other graphic functions. From Verzani (2005), Table 3.7 (p. 86).

```
> load("Africa.RData")
> x <- table(Africa$Region) # make a table for a factor variable
> x

 Central Africa  Eastern Africa Northern Africa Southern Africa  Western Africa
              9              16               6               5              16

> barplot(x) # Bar graph; takes in a numeric vector
> barplot(x, xlab="Regions", ylab="Frequency", ylim=c(0,20),
+     main="Geographical Distribution of African Countries")  # Adds labels
> pie(x) # Pie chart; takes in a numeric vector
```







1.2 Printing and Saving Graphs

There are a few ways to print and save the graphs you create in **R**.

- In the window of your graph (if you are a Mac user, make sure your graphic window rather than the R console is selected), you can click File: Save as: PDF... or File: Print....
- You can also right-click on a figure in **R** and copy the image (if you are a Mac user, you need to highlight the graph and type Apple+C to copy it). Then paste that image into Microsoft Word or any other document.
- You can also do it via a command by using `pdf()` before your plotting commands.

  ```
  > pdf(file="pie.pdf", height=3, width=5) # height and width are in inches
  ```

  After your plotting commands, you need to type

  ```
  > dev.off()
  ```

- A variety of options available through `par()`; e.g., `par(cex = X)` where `X` is a magnification factor for text. Numbers bigger than 1 increase the font size. see `?par` for more.
- Setting `mfrow=c(X, Y)` or `mfcol=c(X, Y)` in `par()` will allow you to place multiple ($X \times Y$) plots in one graph

1.3 Summary Statistics for Univariate Data

- For a `numeric` object, we have `mean()` (mean), `median()` (median), `min()` (minimum), `max()` (maximum), `var` (variance), `sd()` (standard deviation)
- The function `summary()` will provide the mean, median, minimum, maximum, and quartiles of a `numeric` object and a table for a `factor` object (you can also use `table()` for this)

- Weighted mean, $\sum_{i=1}^{n} w_i x_i / \sum_{i=1}^{n} w_i$, can be computed using `weighted.mean()`

```
> mean(Africa$GDP.pc)  # Simple mean

[1] 4616.115

> Africa$pop <- Africa$GDP / Africa$GDP.pc  # Population of each country
> weighted.mean(Africa$GDP.pc, Africa$pop) # Weighted mean

[1] 3329.112

> median(Africa$GDP.pc)

[1] 2162.5

> summary(Africa$GDP.pc)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    500    1366    2162    4616    5569   23290

> var(Africa$GDP.pc)  # Variance of GDP per capita

[1] 30692211

> sd(Africa$GDP.pc)  # the standard deviation

[1] 5540.055
```

- The function `quantile(X, P)` provides the sample quantiles of a numeric object `X` for each element of `P`
- The function `IQR()` returns the interquartile range.

```
> quantile(Africa$HDI)

    0%    25%    50%    75%   100%
0.3310 0.4335 0.5125 0.6430 0.8430

> quantile(Africa$HDI, c(0.1,0.25,0.50,0.75,0.9))  # Reports specified quantiles

   10%    25%    50%    75%    90%
0.3804 0.4335 0.5125 0.6430 0.7305

> IQR(Africa$HDI)  #Inter-Quartile Range

[1] 0.2095
```

- `tapply(X, INDEX, FUN)` applies the function `FUN` to `X` for each of the groups defined by `INDEX`
- Replace `FUN` with `mean, median, sd`, etc. to generate desired quantity.

```
> tapply(Africa$HDI, Africa$Region, mean)  # Calculates mean HDI for each region

 Central Africa  Eastern Africa Northern Africa Southern Africa  Western Africa
      0.5202222       0.5170000       0.6995000       0.6148000       0.4570000
```

1.4 More Graphics for Univariate Data

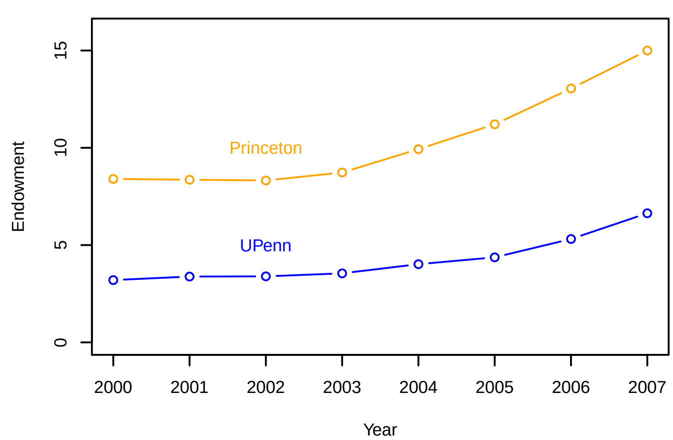
1.4.1 Trend Plots

- The function `plot(X, Y)` will produce the **trend plot** where `X` is a vector for time and `Y` is a corresponding vector of values. You can set `type = "l"` or `type = "b"`.
- You can add lines, points, and texts to the graph too:

`lines()`	Add a plot-line to a currently open figure. e.g. `lines(x,y)` where `x` and `y` are vectors of x- and y-coordinates.
`abline()`	Add a straight line. e.g. `abline(h=`τ`)` to place a horizontal line at height τ. e.g. `abline(v=`τ`)` to place a vertical line at point τ. e.g. `abline(a=`τ`, b=`λ`)` to place a line with intercept τ and slope λ.
`points()`	Add points. e.g. `points(x,y)` to place dots with `x` and `y` as vectors of x- and y-coordinates. e.g. `points(x,y, line=TRUE)` to connect the dots as a line.
`text()`	Add additional text to the plot. e.g. `text(x,y,"my text")` to write ”my text” centered at coordinates x,y.

Table 2: Additional commands to append to an open graphic figure.

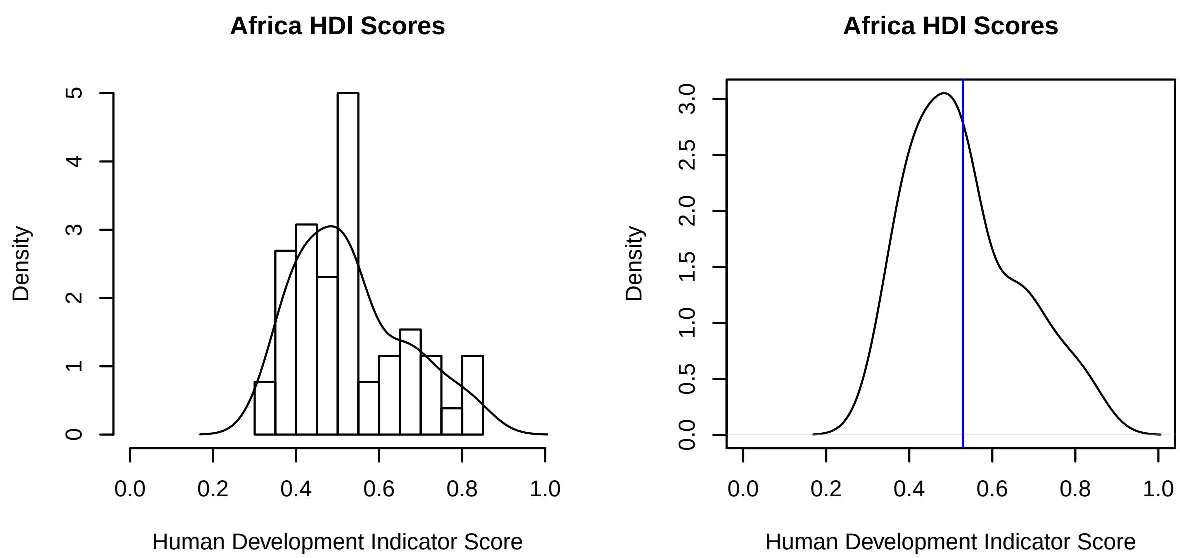
```
> Year <- 2000:2007
> # Princeton endowment 2000-2007
> Endowment <- c(8.398, 8.359, 8.320, 8.730, 9.928, 11.207, 13.045, 15.000)
> # UPenn endowment
> Endowment.penn <- c(3.201,3.382,3.393,3.547,4.019,4.370,5.313,6.635)
> par(cex = 0.6) # Use smaller font to look nicer in this handout
> plot(Year, Endowment, type = "b", ylim = c(0,16), col = "orange")
> lines(Year, Endowment.penn, type = "b", col = "blue") # Add the UPenn trend line
> text(2002, 10, "Princeton", col = "orange") # Added to the plot
> text(2002, 5, "UPenn", col = "blue")
```



1.4.2 Histograms

- The function `hist(X, freq = FALSE)` will produce a **histogram**; the argument `breaks` will set the number of bins
- Setting `freq = TRUE` in `hist()` will produce a **frequency plot** rather than a histogram
- The function `density()` will calculate the density of a `numeric` object and can be used to draw smoothed histogram via `plot(density(x))` or `lines(density(x))`

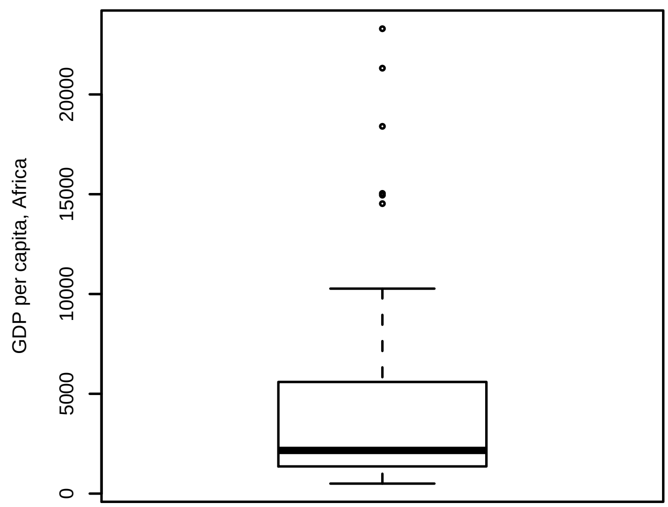
```
> par(mfrow=c(1,2), cex = 0.65) # placing multiple plots in one graph
> hist(Africa$HDI, xlim = c(0, 1), freq = FALSE, main = "Africa HDI Scores",
+      breaks = 10, xlab = "Human Development Indicator Score")
> lines(density(Africa$HDI))  # Added to the histogram
> plot(density(Africa$HDI), xlim = c(0, 1),  #looks roughly normal
+     xlab = "Human Development Indicator Score", main = "Africa HDI Scores")
> avg <- mean(Africa$HDI)
> abline(v = avg, col = "blue")  # Adds a vertical line at avg
```



1.4.3 Boxplots

- The function `boxplot()` will produce a **boxplot** figure

```
> par(cex = 0.5)
> boxplot(Africa$GDP.pc, ylab = "GDP per capita, Africa")
```



2 Descriptive Statistics and Graphing for Multivariate Data

2.1 Two-Way Tables

- The function `table(X, Y)` will create a two-way table using two variables `X` and `Y`.

```
> admit <- read.csv("admit.csv", header=T) # Data saved off of Blackboard
> table(admit$female)

 0  1
71 35

> table(admit$score)

 1  2  3  4  5
23 24  2 37 20

> gre.scores <- table(admit$female, admit$score)
> gre.scores # Look at the data again

     1  2  3  4  5
  0 16 16  0 27 12
  1  7  8  2 10  8
```

- The function `prop.table()` will convert a table to a table with proportions.

```
> prop.table(gre.scores)
```

```
           1          2          3          4          5
0 0.15094340 0.15094340 0.00000000 0.25471698 0.11320755
1 0.06603774 0.07547170 0.01886792 0.09433962 0.07547170
```

- The function `addmargins()` will append the sums for both rows and columns onto our table

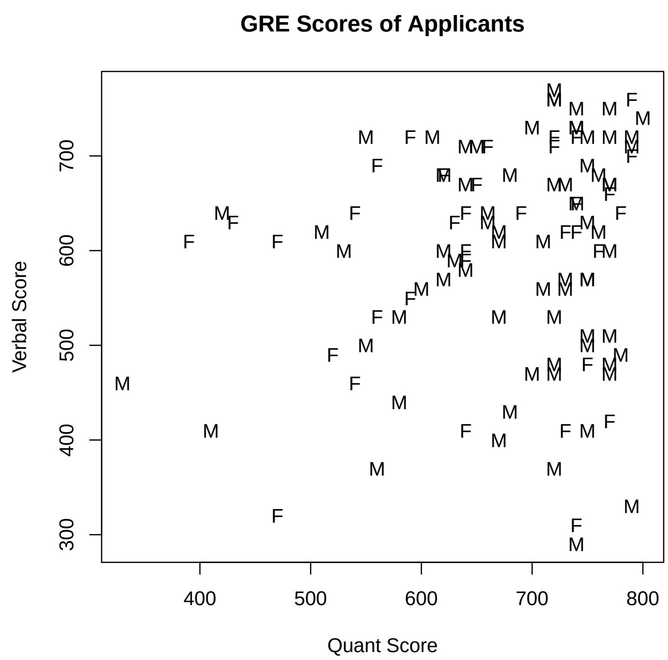
```
> addmargins(gre.scores) # Append the sums for both rows and columns onto our table
```

```
      1   2   3   4   5 Sum
0    16  16   0  27  12  71
1     7   8   2  10   8  35
Sum  23  24   2  37  20 106
```

2.2 Graphing Multivariate Data

- The function `plot(x, y, ...)` will create a simple scatterplot, in which the vector `x` is plotted against `y`

```
> admit$gender <- ifelse(admit$female==1,"F","M") # Creates a new gender variable
> plot(admit$gre.quant, admit$gre.verbal, pch = admit$gender,
+     xlab="Quant Score", ylab="Verbal Score", main="GRE Scores of Applicants")
```

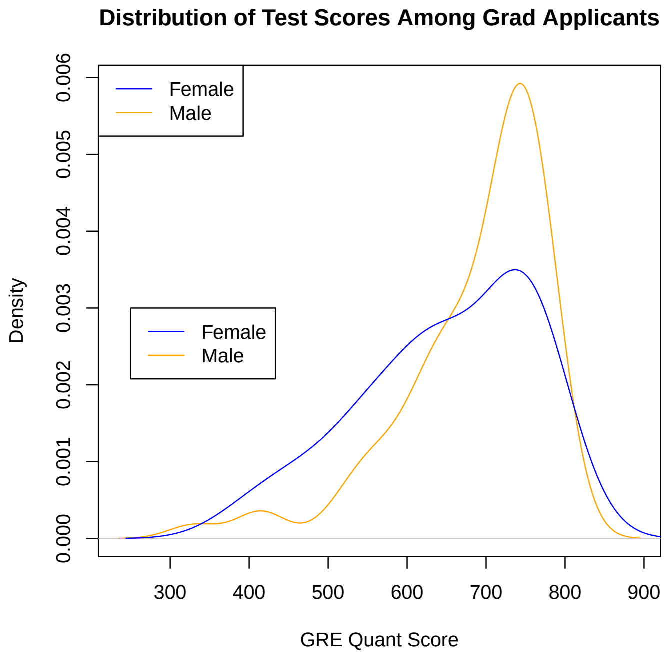


- The function `legend(X, Y, Z)` will add a legend to an existing plot where `X` is the x-coordinate, `Y` is the y-coordinate, and `Z` is a vector of text.

- The `X,Y` argument can be replaced with a keyword indicating location, such as `"topleft"`, `"bottomright"`, etc.

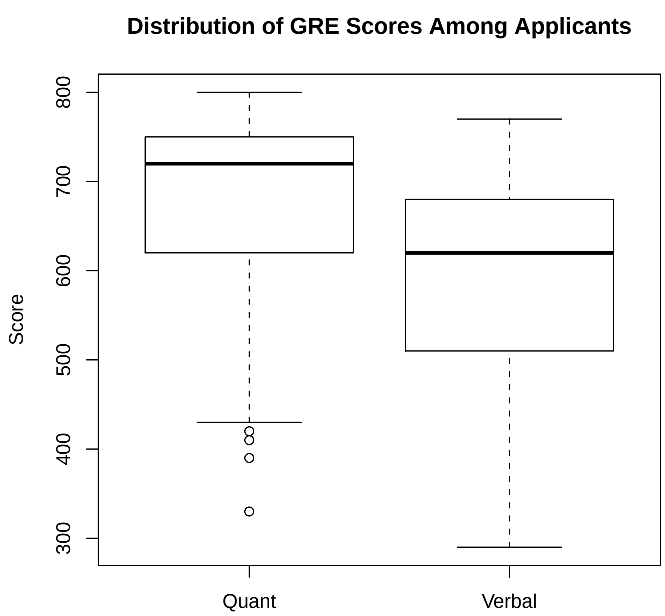
```
> plot(density(admit$gre.quant[admit$female==0]),
+     xlab="GRE Quant Score", ylab="Density",
```

```
+     main="Distribution of Test Scores Among Grad Applicants", col="orange")
> lines(density(admit$gre.quant[admit$female==1]), col="blue")
> legend("topleft", c("Female","Male"), lty = c(1,1), col = c("blue","orange"))
> legend(250, 0.003, c("Female","Male"), lty = c(1,1), col = c("blue","orange"))
```



- The function `boxplot(a, b, ...)` will create a side-by-side boxplot for the variables `a` and `b`

```
> # Side-by-side Boxplots
> boxplot(admit$gre.quant, admit$gre.verbal, names=c("Quant", "Verbal"),
+     ylab="Score", main="Distribution of GRE Scores Among Applicants")
```



2.3 Correlation

- The function `cor(X, Y)` takes in two vectors (`X` and `Y`) and returns their correlation

```
> cor(admit$gre.verbal, admit$gre.quant)

[1] 0.1599913
```

2.4 Linear Regression

- The function `lm(Y ~ X, data = Z)` regresses a variable `Y` on a variable `X` taken from the data frame `Z`.

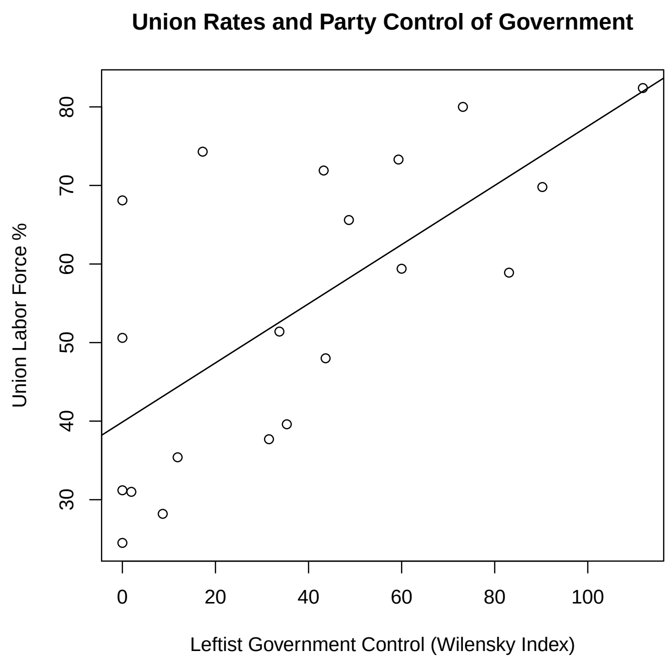
```
> # load in the union dataset, downloaded from Blackboard
> union <- read.csv("union.csv", header=T)
> # union: percentage of workers who belong to a union
> # left: extent to which parties of the left have controlled government
> # size: size of the labor force
> # concen: measure of economic concentration in top-4 industries
>
> fit.1 <- lm(union ~ left, data=union)  # Our linear regression
```

- Applying `summary()` to the regression output will produce a summary.
- Applying the function `coef()` to your linear model will output just the coefficient estimates for your regression.

```
> summary(fit.1)

Call:
lm(formula = union ~ left, data = union)

Residuals:
    Min      1Q  Median      3Q     Max
-15.384 -10.269  -3.558  10.808  28.216

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 39.88406    4.81269   8.287 1.48e-07 ***
left         0.37639    0.09619   3.913  0.00102 **
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

Residual standard error: 14.16 on 18 degrees of freedom
Multiple R-squared: 0.4597,	Adjusted R-squared: 0.4296
F-statistic: 15.31 on 1 and 18 DF,  p-value: 0.001019

> coef(fit.1)

(Intercept)        left
 39.8840609   0.3763868
```

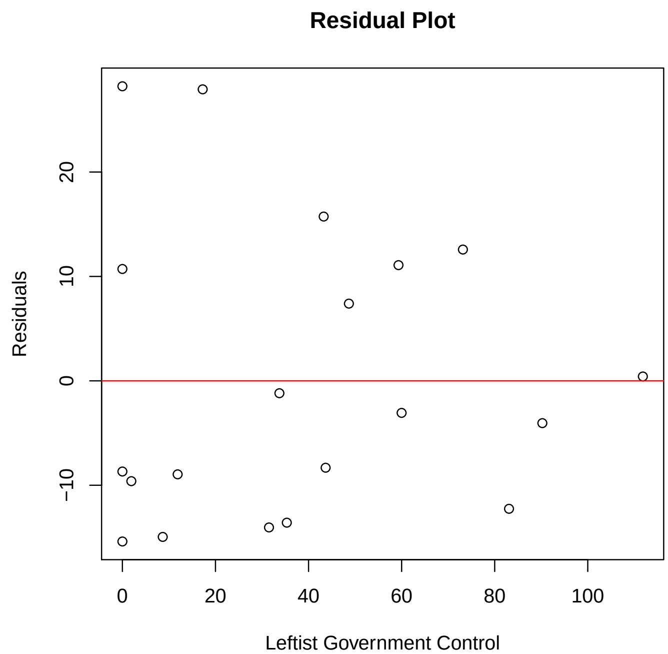
- You can add the fitted line to the scatter plot through `abline()`.

```
> plot(union$left, union$union,
+     xlab="Leftist Government Control (Wilensky Index)",
+     ylab="Union Labor Force %",
+     main="Union Rates and Party Control of Government")
>     # Creates a scatterplot
> abline(fit.1)  # Adds the best-fit line to our plot
> abline(coef(fit.1)) # Equivalent command
```

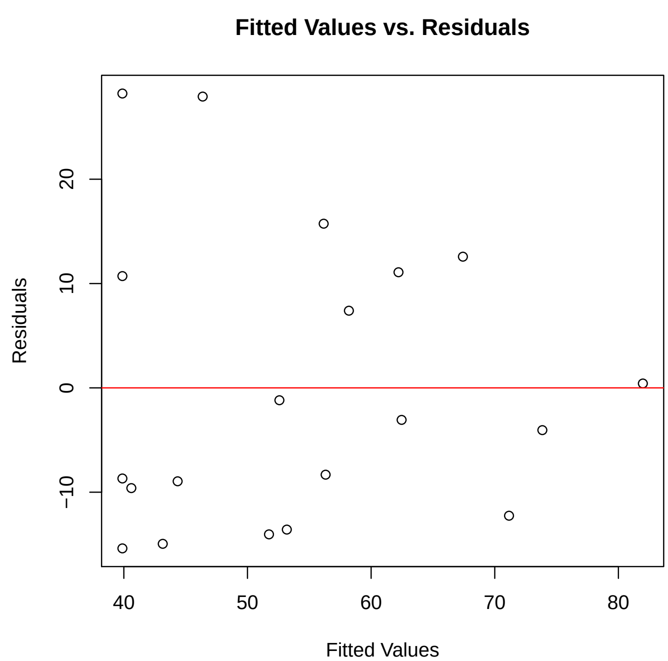


- The function `resid()` yields the residuals from a linear regression.
- The function `fitted()` yields the fitted values from a linear regression.
- These functions can be used to create residual plots.

```
> resid(fit.1)

         1          2          3          4          5          6          7
  0.420834  12.575713  27.923266  11.084907  15.737208  -4.049210  28.215939
         8          9         10         11         12         13         14
  7.397191  -3.067272 -12.254280  -1.183353  10.715939  -8.320875 -13.581808
        15         16         17         18         19         20
-14.040247  -8.951773  -8.684061  -9.606724 -14.947335 -15.384061

> plot(union$left, resid(fit.1), xlab="Leftist Government Control",
+     ylab="Residuals", main = "Residual Plot") # Plots residuals against X
> abline(h=0, col="red") # Adds a zero line
```




```
> plot(fitted(fit.1), resid(fit.1), xlab="Fitted Values",
+      ylab="Residuals", main = "Fitted Values vs. Residuals")
>      # Plots residuals against y hat
> abline(h=0, col="red") # Adds a zero line
```



3 Getting Ready for Simulation

3.1 Loop

- The function `for(i in X)` will create a loop in your programming code where `i` is a counter and `X` is a vector for the counter. That is, the following syntax,

  ```
  for (i in X) {
    blah1...
    blah2...
    ...
  }
  ```

  will execute the code chunk, `blah1... blah2... ...`, the same number of times as the length of `X` vector while setting the counter `i` to each element of `X`. You can have as many commands and lines in a loop as you like.

- Braces (`{}`) are used to denote the beginning and end of your loops. If your code chunk only contains one line, you can get away without using the braces. That is,

  ```
  for (i in X)
    blah1...
  ```

  works though it is generally a good idea to keep the braces.

- The function `rep(X,Y)` will create a vector of length Y with each item equal to X.

- The function `print()` will print a formatted object.

- The function `cat()` will concatenate (i.e. paste) a set of texts and/or objects together (each should be separated by a comma) and then print the information to the **R** console.

  ```
  > for (i in 1:3){ #notice the open brace
  +   print(i)
  + } #and closing brace for our for() loop

  [1] 1
  [1] 2
  [1] 3

  > x <- c("hey", "Hey", "HEY")
  > for (i in x){
  +   print(i)
  + }

  [1] "hey"
  [1] "Hey"
  [1] "HEY"
  ```

```
> for (j in 3:5){
+   x <- j*2
+   cat(j, "times 2 is equal to", x, "\n") #\n changes a line
+ }

3 times 2 is equal to 6
4 times 2 is equal to 8
5 times 2 is equal to 10

> Z <- rep(NA, 10) # Create an empty vector to hold our answer in
> for (j in 1:10){
+   Z[j] <- j*2 # Store the value from each loop into the vector
+ }
> Z

 [1]  2  4  6  8 10 12 14 16 18 20
```

3.2 Conditional Statements

- The following syntax

```
if (X) {
  blah1...
  blah2...
  ...
}
```

will execute the code chunk, `blah1...` `blah2...` if the condition `X` is met. If the condition is not met, then it will not execute that code chunk.

- You can have as many lines in the code chunk as you like. Similar to a loop, if you only have one line in the code chunk, you can omit the braces though it is generally a good idea to have them for the sake of clarity. It is also a good idea to indent the code chunk so that the code is easy to read.

```
> if (3>4) 3*12 #No action takes place because condition isn't met
> if (5>4) 3*12 #Condition met and R proceeds with computation

[1] 36

> # you can use if() within a loop
> x <- c(1, 5, 4, 2, 3)
> y <- 0
> for (i in 1:length(x)) {
+   if (x[i] > 2) {
+     y <- y + x[i]
+   }
+ }
> y
```

```
[1] 12
```

```
> ## this is the same as
> sum(x[x > 2])
```

```
[1] 12
```

- The following syntax

  ```
  if (X) {
    blah1...
    blah2...
    ...
  } else {
    blah3...
    blah4...
    ...
  }
  ```

  will execute the code chunk, `blah1...` `blah2...` `...`, if the condition `X` is met. Otherwise, the code chunk, `blah3...` `blah4...` `...`, will be executed.

- You can nest multiple conditional statements. For example,

  ```
  if (X) {
    blah1...
    blah2...
    ...
  } else if (Y) {
    blah3...
    blah4...
    ...
  } else if (Z) {
    blah5...
    ...
  } else {
    blah6...
    ...
  }
  ```

  ```
  > #Add in an else statement
  > if (3 > 4){
  +     x <- 3*12
  + } else {
  +     x <- 3*20
  + }
  > x
  ```

  ```
  [1] 60
  ```

```
> #use if, else if, and else
> if (3 > 4){
+   a <- "Skip Election"
+ } else if (4 > 3) {
+   a <- "Obama Wins"
+ } else {
+   a <- "McCain Wins"
+ }
> a

[1] "Obama Wins"
```

3.3 More Functions for Simulation

- The function `sample(X, Y, replace = TRUE, prob = P)` will let you sample `Y` units from a vector `X` with or without replacement (`replace = TRUE` or `replace = FALSE`) using a vector of probability `P` (the default is equal probability).

```
> ## Randomly draw 8 samples from Z, with replacement
> sample(Z, 8, replace = TRUE)

[1]  2 16 16 20 16 18 20 20

> ## Randomly draw 8 samples from Z, without replacement
> sample(Z, 8, replace = FALSE)

[1] 16 12 14  2 20 18  8  6

> ## Randomly draw 8 samples from Z, with replacement
> sample(Z, 8, replace = TRUE, prob = Z/sum(Z))

[1] 16 18 20 18 10 16 16 10
```

- The function `unique()` will return the unique elements of a vector or a dataframe.

```
> Y <- c(1,0,1,2,0,2,3)
> unique(Y) #display only the unique elements from the vector

[1] 1 0 2 3
```