More about R

Chao-Lung Yang, Ph.D.
Department of Industrial Management
National Taiwan University of Science and Technology

Notes

- The following materials are modified based on slides and examples from Statistical Consulting Group ,UCLA Institute for Digital Research & Education
- http://www.ats.ucla.edu/stat/r/seminars/intro.htm

R Packages in this Lecture

- For the purposes of this seminar, we will be using the following packages frequently:
- foreign package to read data files from other stats packages
- xlsx package (requires Java to be installed, same architecture as your R version, also the rJava package and xlsxjars package)
- reshape2 package to easily melt data to long form
- ggplot2 package for elegant data visualization using the *Grammar of Graphics*
- GGally package for scatter plot matrices
- vcd package for visualizing and analyzing categorical data

Installing Packages

- To use packages in R, we must first install them using the install.packages function, which typically downloads the package from CRAN and installs it for use

```
install.packages("xlsx")
install.packages("reshape2")
install.packages("ggplot2")
install.packages("GGally")
install.packages("vcd")
```

Loading Packages

- If we know we will need a particular package for our current R session, we must load it into the R environment using the require function

```
require(foreign)
require(xlsx)
require(reshape2)
require(ggplot2)
require(GGally)
require(vcd)
```

Basic Info On R Session

- To get a description of the version of R and its attached packages used in the current session, we can use the sessionInfo function
- sessionInfo()

R Script

- R code can be entered into the command line directly or saved to a script, which can be run inside a session using the source function
- Commands are separated either by a ; or by a newline.
- R is case sensitive.
- The # character at the beginning of a line signifies a comment, which is not executed.
- Help files for R functions are accessed by preceding the name of the function with ? (e.g. ?require).

R Objects

- R stores both data and output from data analysis (as well as everything else) in *objects*
- Things are assigned to and stored in objects using the <- or = operator
- A list of all objects in the current session can be obtained with ls()

```
# assign the number 3 to object called abc
abc <- 3
# list all objects in current session
ls()
```

ENTERING DATA

Dataset Files

- R works most easily with datasets stored as text files. Typically, values in text files are separated, or delimited, by tabs or spaces:

  gender id race ses schtyp prgtype read write math science socst

  0 70 4 1 1 general 57 52 41 47 57

  1 121 4 2 1 vocati 68 59 53 63 31

- Or by commas (CSV file)

  gender,id,race,ses,schtyp,prgtype,read,write,math,science,socst

  0,70,4,1,1,general,57,52,41,47,57

  1,121,4,2,1,vocati,68,59,53,63,61

Reading in Data 1

- Base R functions read.table and read.csv can read in data stored as text files, delimited by almost anything (notice the sep = option)
- Although we are retrieving files over the internet for this class, these functions are typically used for files saved to disk.
- Note how we are assigning the loaded data to objects.

```
# comma separated values
dat.csv <- read.csv("http://www.ats.ucla.edu/stat/data/hsb2.csv")
# tab separated values
dat.tab <- read.table("http://www.ats.ucla.edu/stat/data/hsb2.txt",
  header=TRUE, sep = "\t")
```

Reading in Data 2

- We can read in datasets from other statistical analysis software using functions found in the foreign package

```
require(foreign)
# SPSS files
dat.spss <- read.spss("http://www.ats.ucla.edu/stat/data/hsb2.sav",
  to.data.frame=TRUE)
# Stata files
dat.dta <- read.dta("http://www.ats.ucla.edu/stat/data/hsb2.dta")
```

Reading in Excel Files

- Datasets are often saved as Excel spreadsheets. Here we utilize the xlsx package and Java to download an Excel dataset.

```
require(xlsx)
# these two steps only needed to read excel files from the internet
f <- tempfile("hsb2", fileext=".xls")

download.file("http://www.ats.ucla.edu/stat/data/hsb2.xls", f, mode="wb")

dat.xls <- read.xlsx(f, sheetIndex=1)
```

Look at Data

- R has ways to look at the dataset at a glance or as a whole

Head

Tail

Colnames

View

```
# first few rows
head(dat.csv)
# last few rows
tail(dat.csv)
# variable names
colnames(dat.csv)
# pop-up view of entire data set (uncomment to run)
View(dat.csv)
```

Data Frames

- Once read in, datasets in R are typically stored as *data frames*, which have a matrix structure. Observations are arranged as rows and variables, either numerical or categorical, are arranged as columns.
- Individual rows, columns, and cells in a data frame can be accessed through many methods of indexing.
- We most commonly use object[row,column] notation.

Exercise 1: Viewing Data

```
# single cell value
dat.csv[2,3]
# omitting row value implies all rows; here all rows in column 3
dat.csv[,3]
# omitting column values implies all columns; here all columns in row 2
dat.csv[2,]
# can also use ranges - rows 2 and 3, columns 2 and 3
dat.csv[2:3, 2:3]
```

Variable Indexing

- We can also access variables directly by using their names, either with object[,"variable"] notation or object$variable notation.

```
# get first 10 rows of variable female using two methods
dat.csv[1:10, "female"]
dat.csv$female[1:10]
```

The c Function

- The c function is widely used to combine values of common type together to form a vector.
- For example, it can be used to access non-sequential rows and columns from a data frame.

```
# get column 1 for rows 1, 3 and 5
dat.csv[c(1,3,5), 1]
# get row 1 values for variables female, prog and socst
dat.csv[1,c("female", "prog", "socst")]
```

Variable Names

- If there were no variable names, or we wanted to change the names, we could use colnames

```
colnames(dat.csv) <- c("ID", "Sex", "Ethnicity", "SES", "SchoolType",
  "Program", "Reading", "Writing", "Math", "Science", "SocialStudies")

# to change one variable name, just use indexing
colnames(dat.csv)[1] <- "ID2"
```

Saving Data

- When we have the dataset as we like it, we can save it to a number of formats, including text, Stata .dta, and Excel .xlsx.
- The function write.dta comes from the foreign package, while write.xlsx comes from the xlsx package.

```
#write.csv(dat.csv, file = "path/to/save/filename.csv")

#write.table(dat.csv, file = "path/to/save/filename.txt", sep = "\t", na=".")

#write.dta(dat.csv, file = "path/to/save/filename.dta")

#write.xlsx(dat.csv, file = "path/to/save/filename.xlsx", sheetName="hsb2")

# save to binary R format (can save multiple datasets and R objects)
#save(dat.csv, dat.dta, dat.spss, dat.txt, file = "path/to/save/filename.RData")
```

EXPLORING DATA

- Now we're going to read some data in and store it in the object, d. We prefer short names for objects that we will use frequently.
- We can now easily explore and get to know these data, which contain a number of school, test, and demographic variables for 200 students.

```
d <- read.csv("http://www.ats.ucla.edu/stat/data/hsb2.csv")
```

Description of Dataset

- Using dim, we get the number of observations(rows) and variables(columns) in d.
- Using str, we get the structure of d, including the class(type) of all variables
- summary is a generic function to summarize many types of R objects, including datasets.

```
dim(d)
str(d)
summary(d)
```

Conditional Summaries 1

- If we want conditional summaries, for example only for those students with high reading scores (read >= 60), we first subset the data, then summarize as usual.
- R permits nested function calls, where the results of one function are passed directly as an argument to another function. Here, subset returns a dataset containing observations where read >= 60. This data subset is then passed to summary to obtain distributions of the variables in the subset.

```
summary(subset(d, read >= 60))
```

Conditional Summaries 2

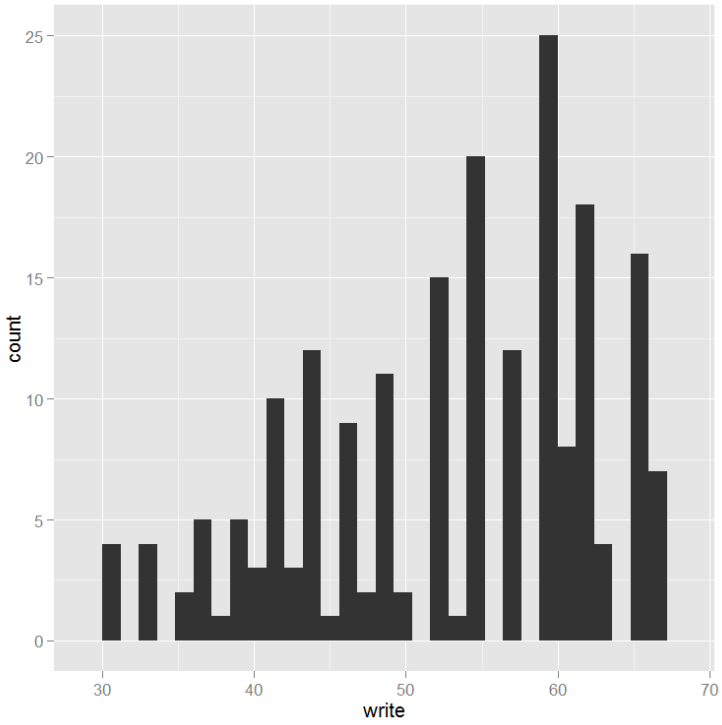
- We can separate the data in other ways, such as by groups. Let's look at the means of the last 5 variables for each type of program, prog.
- Here, we are asking the by function to apply the colMeans function to variables located in columns 7 through 11, and to calculate those means by groups denoted in the variable prog.

```
by(d[, 7:11], d$prog, colMeans)
```

Histograms

- Typically it is easier to inspect variable distributions with graphics. Histograms are often used for continuous variable distributions...

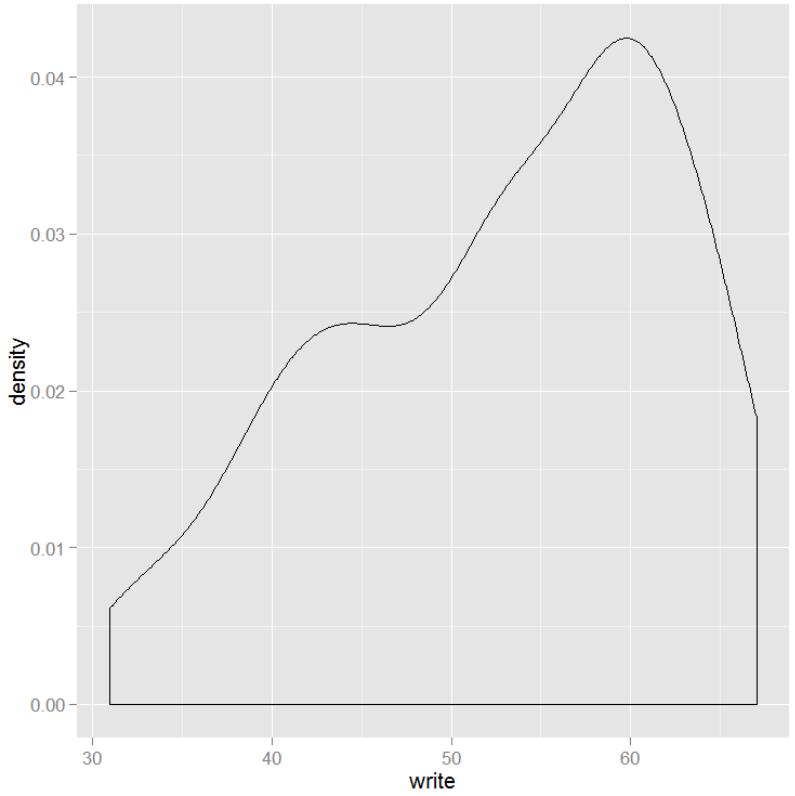
```
ggplot(d, aes(x = write)) + geom_histogram()
```



Density Plots

- Kernel density plots

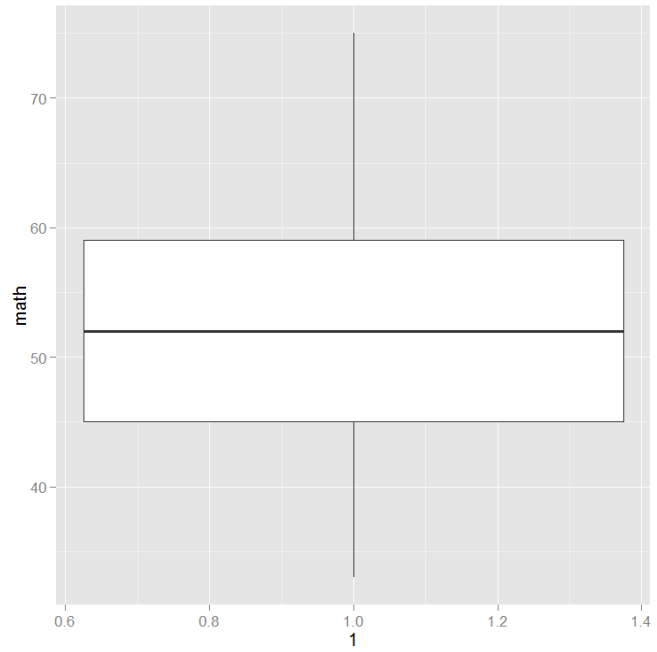
```
ggplot(d, aes(x = write)) + geom_density()
```



Boxplots

- boxplots, which show the median, lower and upper quartiles (or hinges) and the full range

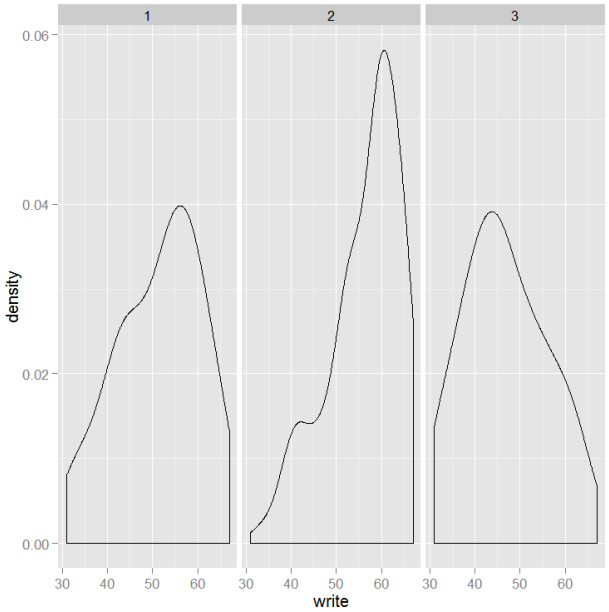
```
ggplot(d, aes(x = 1, y = math)) + geom_boxplot()
```



Conditional Visualization 1

- We can also plot graphs by group to better understand our data. Here we examine the densities of write for each type of program, prog.

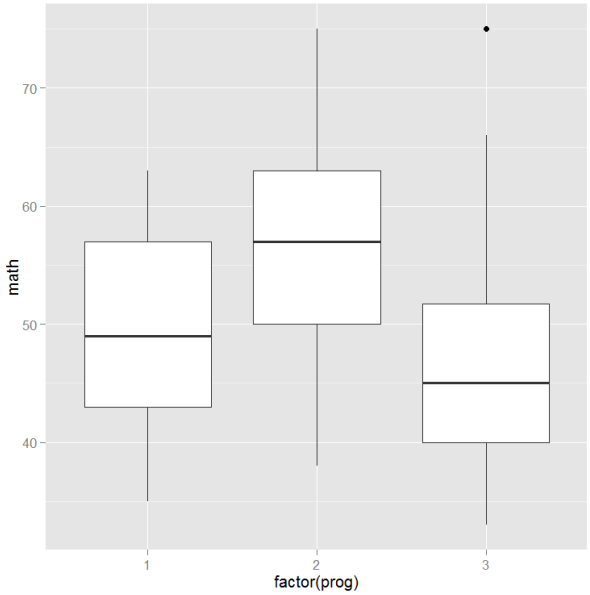
```
# density plots by program type
ggplot(d, aes(x = write)) + geom_density() + facet_wrap(~ prog)
```



Conditional Visualization 2

- We could also view boxplots of math by type of program.
- Here we plot math scores for each group in prog.

```
ggplot(d, aes(x = factor(prog), y = math)) + geom_boxplot()
```

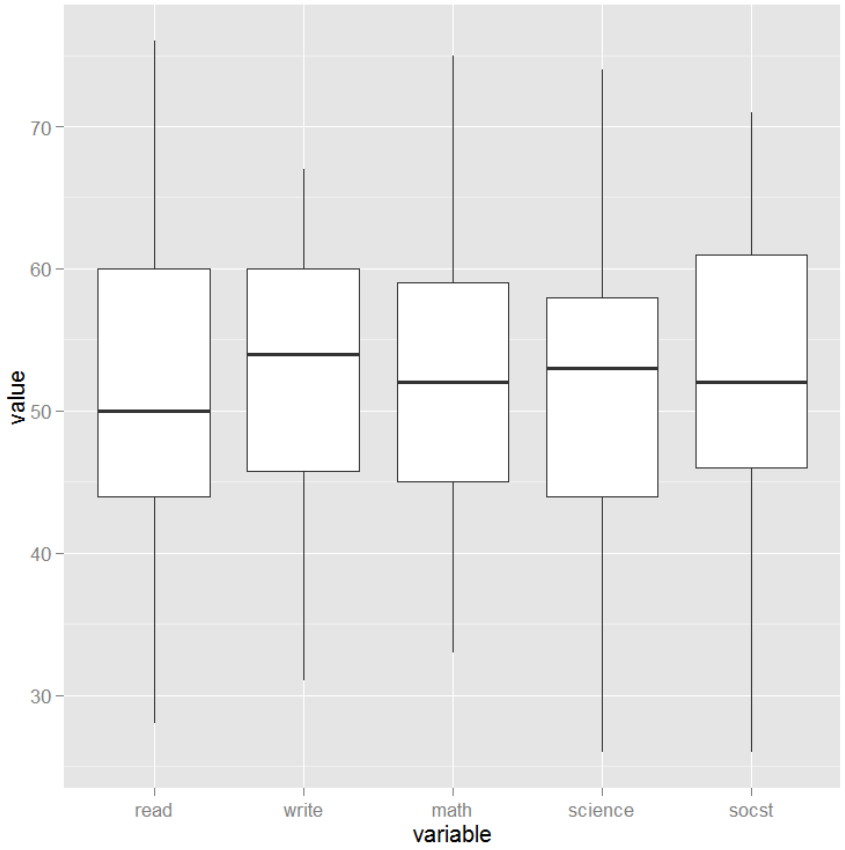


Extended Visualization 2

- Here we demonstrate the flexibility of R by plotting the distributions of all of our continuous variables in a single boxplot.
- The function melt from the reshape2 package stacks the values located in columns 7 through 11 of our dataset on top of one another to form one column called "value", and stacks the variable names associated with each value in a second column called "variable".
- Thus we are asking for a boxplot of "values" grouped by "variable"

Extended Visualization 2 (Con't)

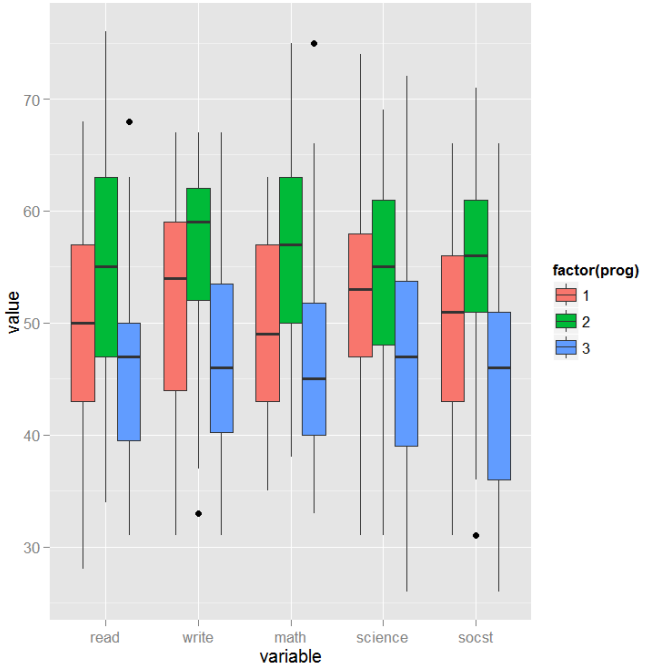
```
ggplot(melt(d[, 7:11]), aes(x = variable, y = value)) + geom_boxplot()
```



Extended Visualization 3

- Finally we can get boxplots of these same variables coloured by program type by specifying a fill argument.

```
ggplot(melt(d[, 6:11], id.vars = "prog"),
       aes(x = variable, y = value, fill = factor(prog))) +
  geom_boxplot()
```



Categorical Data 1

- We can look at the distributions of categorical variables with frequency tables

```
xtabs( ~ female, data = d)
xtabs( ~ race, data = d)
xtabs( ~ prog, data = d)
```

```
> xtabs(~female, data = d)
female
  0   1
 91 109
> xtabs(~race, data = d)
race
  1   2   3   4
 24  11  20 145
> xtabs(~prog, data = d)
prog
  1   2   3
 45 105  50
>
```

Categorical Data 2

- Two-way cross tabs

```
xtabs( ~ ses + schtyp, data = d)
```

```
>
> xtabs( ~ ses + schtyp, data = d)
   schtyp
ses  1  2
  1 45  2
  2 76 19
  3 47 11
>
```

Categorical Data 3

- Three-way cross tabs

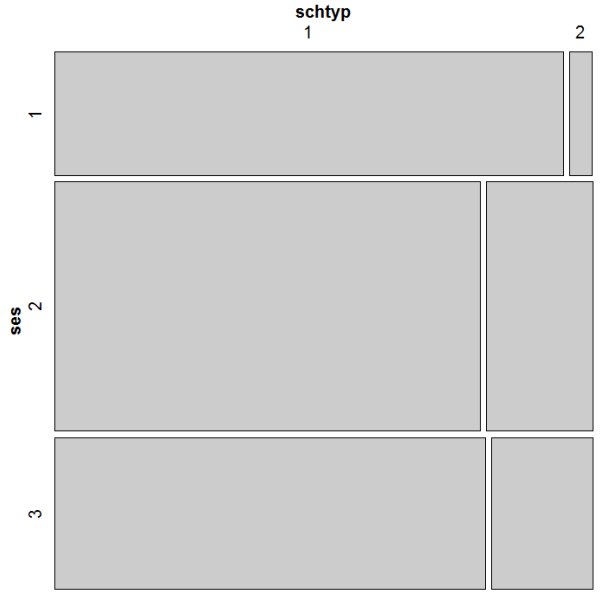
```
(tab3 <- xtabs( ~ ses + prog + schtyp, data = d))
```

```
> (tab3 <- xtabs( ~ ses + prog + schtyp, data = d))
, , schtyp = 1

   prog
ses  1  2  3
  1 14 19 12
  2 17 30 29
  3  8 32  7

, , schtyp = 2

   prog
ses  1  2  3
  1  2  0  0
  2  3 14  2
  3  1 10  0

>
```

Visualizing Cat Data (VCD) 1

- The area of each cell in a mosaic plot corresponds to the frequency from the crosstabs. mosaic comes from the vcd package

```
(tab2 <- xtabs( ~ ses + schtyp, data = d))
set.seed(10)
(testtab2 <- coindep_test(tab2, n = 5000))



# simple mosaic plot
mosaic(tab2)
```



Correlations 1

- As a last step in our data exploration, we would like some quick looks at bivariate (pairwise) relationships in our data. Correlation matrices provide quick summaries of these pairwise relationships.
- If there are no missing data, we can use the cor function with default arguments. Otherwise, we could use the use argument to get listwise or pairwise deletion

Correlations 2

- Here we are requesting all pairwise correlations among variables in columns 7 through 11

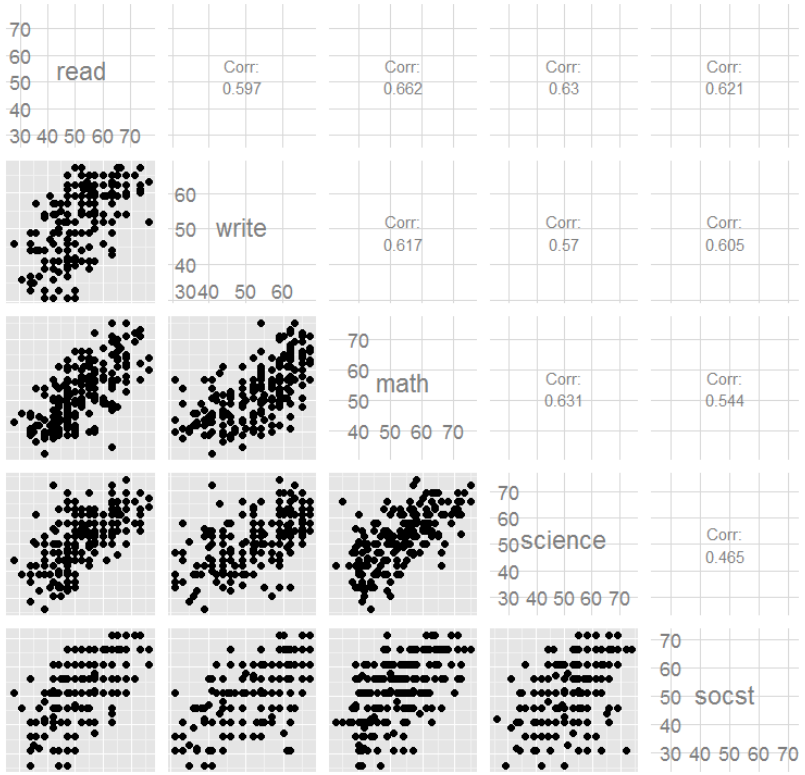
```
cor(d[, 7:11])
```

```
> cor(d[, 7:11])
             read     write      math   science     socst
read    1.0000000 0.5967765 0.6622801 0.6301579 0.6214843
write   0.5967765 1.0000000 0.6174493 0.5704416 0.6047932
math    0.6622801 0.6174493 1.0000000 0.6307332 0.5444803
science 0.6301579 0.5704416 0.6307332 1.0000000 0.4651060
socst   0.6214843 0.6047932 0.5444803 0.4651060 1.0000000
```

Visual Summaries, Continuous Variables

- We can inspect univariate and bivariate relationships using a scatter plot matrix

```
ggpairs(d[, 7:11])
```



MODIFYING DATA

Modifying Data

- This section demonstrates reordering and modifying your data.
- Let's begin by reading in our dataset again and storing it in object d

```
# read data in and store in an easy to use name to save typing
d <- read.csv("http://www.ats.ucla.edu/stat/data/hsb2.csv")
```

Sorting

- We can sort data using the order function.
- Here we are requesting that order sort by id, and then by female. The function order returns the indices of our observations in sorted order, which, when applied to the unsorted dataset as row indices, will return a sorted dataset

```
d <- d[order(d$id, d$female), ]
head(d)
```

Labeling Categorical Variables

- For categorical variables, we can change them from numeric variables to *factors*, a special class that R uses for categorical variables, which also allows for value labeling.
- Here, using the factor function, we convert all categorical variables to factors and label their values.
- To spare us from unnecessary typing, we use the within function to let R know all that all conversions to factors and labeling should occur within the dataset d

Exercise: Factor

```
str(d)
d <- within(d, {
  id <- factor(id)
  female <- factor(female, levels = 0:1, labels = c("male", "female"))
  race <- factor(race, levels = 1:4, labels = c("Hispanic", "Asian", "African American", "White"))
  schtyp <- factor(schtyp, levels = 1:2, labels = c("public", "private"))
  prog <- factor(prog, levels = 1:3, labels = c("general", "academic", "vocational"))
})
```

Scoring & Recoding

- Often we need to create variables from other variables. For example, we may want to sum individual test items to form a total score. Or, we may want to convert a continuous scale into several categories, such as letter grades.
- We can sum variables using rowSums. Related functions are: rowMeans, colSums, colMeans.
- Here, we request that sums involving missing values, represented by NA in R, be missing themselves by specifying na.rm=FALSE. This means: do NOT remove NA values.
- We use the cut function to recode continuous ranges into categories.

```r
d$total <- rowSums(d[, 7:10], na.rm=FALSE)
# recode by breaking into categories
d$grade <- cut(d$total,
  breaks = c(0, 140, 180, 210, 234, 300),
  labels = c("F", "D", "C", "B", "A"))
# view results
summary(d[, c("total", "grade")])
```

```
> # recode by breaking into categories
> d$grade <- cut(d$total,
+   breaks = c(0, 140, 180, 210, 234, 300),
+   labels = c("F", "D", "C", "B", "A"))
> # view results
> summary(d[, c("total", "grade")])
     total       grade
 Min.   :139.0   F: 1
 1st Qu.:180.0   D:51
 Median :210.0   C:50
 Mean   :209.5   B:49
 3rd Qu.:234.0   A:49
 Max.   :277.0
> 
```

Standardize and Average

- We can get standardized (Z) scores with the scale function, and get average read scores for each level of ses using the ave function, which itself calls mean

```
d <- within(d, {
  zread <- scale(read)
  readmean <- ave(read, ses, FUN = mean)
})

head(d[, c("read", "zread", "readmean")])
```

```
> d <- within(d, {
+   zread <- scale(read)
+   readmean <- ave(read, ses, FUN = mean)
+ })
>
> head(d[, c("read", "zread", "readmean")])
  read      zread readmean
1   57  0.4652326 48.27660
2   68  1.5380959 51.57895
3   44 -0.8026968 56.50000
4   63  1.0504307 56.50000
5   47 -0.5100977 51.57895
6   44 -0.8026968 51.57895
>
```

Scoring

- We can also take the mean of a set of variables, ignoring missing values, which is useful for creating composite scores

```
d$rowmean <- rowMeans(d[, 7:10], na.rm=TRUE)
```

```
> d$rowmean <- rowMeans(d[, 7:10], na.rm=TRUE
+
+
+ )
> head(d)
   id female race ses schtyp prog read write math science socst rowmean
1  70      0    4   1      1    1   57    52   41      47    57   49.25
2 121      1    4   2      1    3   68    59   53      63    61   60.75
3  86      0    4   3      1    1   44    33   54      58    31   47.25
4 141      0    4   3      1    3   63    44   47      53    56   51.75
5 172      0    4   2      1    2   47    52   57      53    61   52.25
6 113      0    4   2      1    2   44    52   51      63    61   52.50
>
>
> |
```

MANAGING DATA

Subsetting Observation

- Suppose we would like to subset our dataset into 2 datsets, one for each gender. Here we use the subset function to split the dataset into 2 subsets, and we store each subset in a new object

```
dfemale <- subset(d, female == "female")
dmale <- subset(d, female == "male")
```

```
> dboth <- rbind(dfemale, dmale)
> dim(dfemale)
[1] 109  11
> dim(dmale)
[1] 91 11
> dim(dboth)
[1] 200  11
>
```

Subsetting Variables

- Often, datasets come with many more variable than we want. We can also use subset to keep only the variables we need.
- Note the use of the select argument to subset by variable rather than by observation

```
# note that select is special, so we do not need to quote the variable names
duse <- subset(d, select = c(id, female, read, write))
# note the - preceding c(female... , which means drop these variables
ddropped <- subset(d, select = - c(female, read, write))
```

Adding Observation (Appending)

- If we were given separate files for males and females but wanted to stack them (add observations) we can append the datasets together row-wise with rbind

```
dboth <- rbind(dfemale, dmale)
dim(dfemale)
dim(dmale)
dim(dboth)
```

Merging Data

- If instead we were given separate files describing the same students, but using different variables, we could merge datasets to combine both sets of variables into 1 dataset, using the merge function. Note that we do not need to use the same variable as the id in both datasets. We could have said, by.x = "id.x", by.y = "id.y".

```
dall <- merge(duse, ddropped, by = "id", all = TRUE)
dim(duse)
dim(ddropped)
dim(dall)
```

ANALYZING DATA

Analyzing Categorical Data 1

- We often analyze relationships between categorical variables using chi square tests. chisq.test can use raw data, or you can give it a frequency table. We do the latter.

```
(tab <- xtabs(~ ses + schtyp, data = d))
chisq.test(tab)
```

```
> (tab <- xtabs(~ ses + schtyp, data = d))
   schtyp
ses public private
  1     45       2
  2     76      19
  3     47      11
> chisq.test(tab)

        Pearson's Chi-squared test

data:  tab
X-squared = 6.3342, df = 2, p-value = 0.04213

>
```

T-tests 1

- t.test performs t-tests, used to compare pairs of means. Here we show a one sample t-test comparing the mean of write to a mean of 50, and a paired samples t-test comparing the means of write and read.

```
t.test(d$write, mu = 50)
with(d, t.test(write, read, paired = TRUE))
```

```
> t.test(d$write, mu = 50)

        One Sample t-test

data:  d$write
t = 4.1403, df = 199, p-value = 5.121e-05
alternative hypothesis: true mean is not equal to 50
95 percent confidence interval:
 51.45332 54.09668
sample estimates:
mean of x
   52.775

> with(d, t.test(write, read, paired = TRUE))

        Paired t-test

data:  write and read
t = 0.8673, df = 199, p-value = 0.3868
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.6941424  1.7841424
sample estimates:
mean of the differences
                 0.545
```

T-tests 2

- Perhaps we would like to compare the means of write between males and females. Here are independent samples t-tests with equal variance assumed and not assumed.

```
t.test(write ~ female, data = d, var.equal=TRUE)
t.test(write ~ female, data = d)
```

```
> t.test(write ~ female, data = d, var.equal=TRUE)

        Two Sample t-test

data:  write by female
t = -3.7341, df = 198, p-value = 0.0002463
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -7.441835 -2.298059
sample estimates:
  mean in group male mean in group female
            50.12088             54.99083

> t.test(write ~ female, data = d)

        Welch Two Sample t-test

data:  write by female
t = -3.6564, df = 169.707, p-value = 0.0003409
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -7.499159 -2.240734
sample estimates:
  mean in group male mean in group female
            50.12088             54.99083
```

ANOVA and Regression

- ANOVAs and ordinary least-squares regression are just linear models, so we use lm for both. You should typically store the results of an estimation command in an object, as it often contains a wealth of information.
- The object storing the results of lm can then be supplied to anova for an ANOVA table partitioning the variance sequentially, or to summary for a table of regression parameters and measures of fit.
- If you would like ANOVAs using other types of sums of squares, get the car package from CRAN.

```
> m <- lm(write ~ prog * female, data = d)
> anova(m)
Analysis of Variance Table

Response: write
             Df  Sum Sq Mean Sq F value    Pr(>F)
prog          2  3175.7 1587.85 23.2511 8.876e-10 ***
female        1  1128.7 1128.70 16.5278 6.963e-05 ***
prog:female   2   326.0  162.98  2.3865   0.09464 .
Residuals   194 13248.5   68.29
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> summary(m)

Call:
lm(formula = write ~ prog * female, data = d)

Residuals:
    Min      1Q  Median      3Q     Max
-21.617  -5.143   1.037   6.123  21.174

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                   49.143      1.803  27.251  < 2e-16 ***
progacademic                   5.474      2.169   2.524  0.01241 *
progvocational                -7.317      2.494  -2.933  0.00376 **
femalefemale                   4.107      2.469   1.663  0.09787 .
progacademic:femalefemale     -1.138      2.954  -0.385  0.70052
progvocational:femalefemale    5.030      3.405   1.477  0.14129
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.264 on 194 degrees of freedom
Multiple R-squared: 0.259,      Adjusted R-squared: 0.2399
F-statistic: 13.56 on 5 and 194 DF,  p-value: 2.382e-11

>
```

Regression Continued

- We can easily update a model, for instance by adding a continuous predictor, say read, using update.

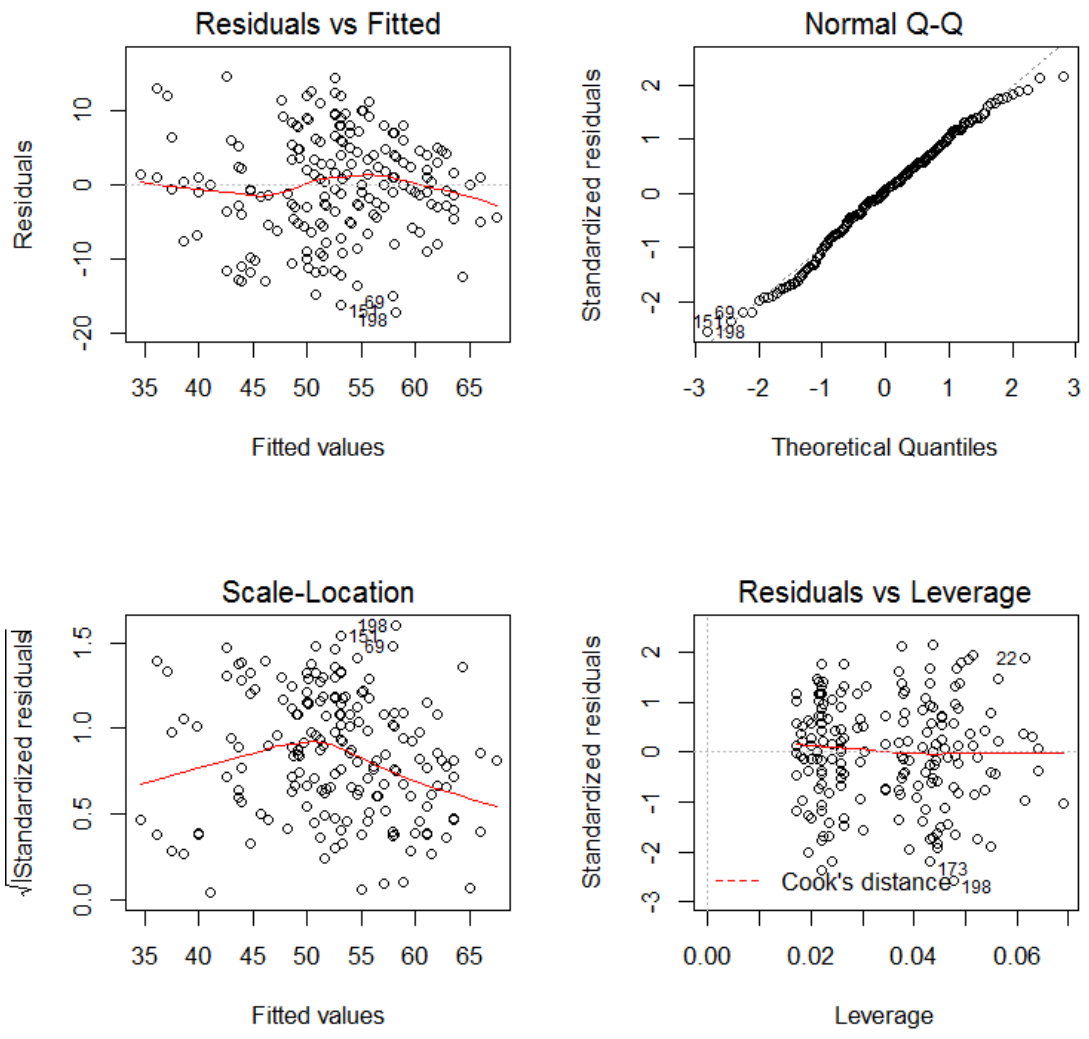
```
summary(m2 <- update(m, . ~ . + read))
```

```
  error in evaluating the argument 'x' in selecting a method for functio$
> summary(m2 <- update(m, . ~ . + read))

Call:
lm(formula = write ~ prog + female + read + prog:female, data = d)

Residuals:
     Min       1Q   Median       3Q      Max
-17.2189  -4.4671   0.4257   4.6084  14.5070

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)                22.94056    3.18807   7.196 1.34e-11 ***
progacademic                3.82925    1.81474   2.110  0.03614 *
progvocational             -3.70443    2.11274  -1.753  0.08113 .
femalefemale                7.07316    2.08061   3.400  0.00082 ***
read                        0.49483    0.05311   9.317  < 2e-16 ***
progacademic:femalefemale  -4.00124    2.47907  -1.614  0.10816
progvocational:femalefemale 1.56171    2.85980   0.546  0.58563
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.881 on 193 degrees of freedom
Multiple R-squared: 0.4889,     Adjusted R-squared: 0.473
F-statistic: 30.77 on 6 and 193 DF,  p-value: < 2.2e-16
```

Regression Diagnostics 1

- Using the plot function on a object storing the results of lm will produce regression diagnostic plots. Here we are arranging them in a 2x2 square using par.

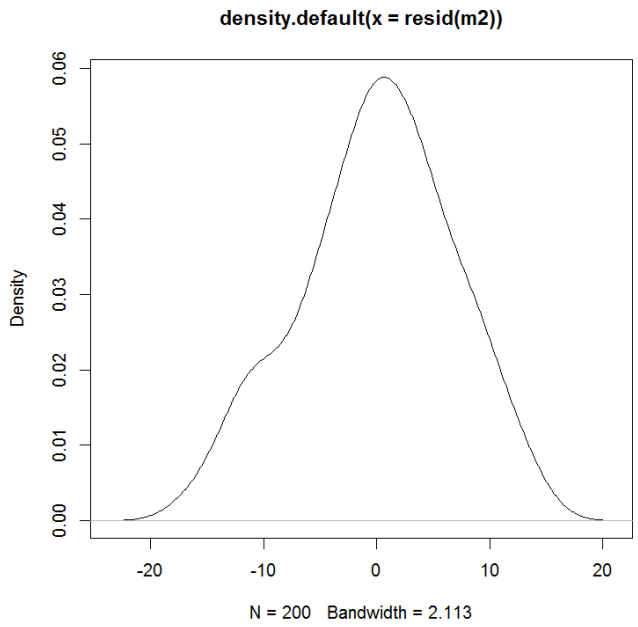
```
par(mfrow = c(2, 2))
plot(m2)
```


Residuals vs Fitted
Residuals
Fitted values
Normal Q-Q
Standardized residuals
Theoretical Quantiles
Scale-Location
√|Standardized residuals|
Fitted values
Residuals vs Leverage
Standardized residuals
Cook's distance
Leverage

Regression Diagnostics 1

- We can get specific diagnostic plots, such as a density plot of the residuals (assumed normally distributed for inference).

```
par(mfrow = c(1, 1))
plot(density(resid(m2)))
```



Regression 3

- Interactions are easy to add to models using *. This notation also automatically creates the lower order terms.

```
summary(m3 <- lm(write ~ prog * read, data = d))
```

```
> summary(m3 <- lm(write ~ prog * read, data = d))

Call:
lm(formula = write ~ prog * read, data = d)

Residuals:
     Min       1Q   Median       3Q      Max
-17.5949  -4.7734   0.5573   5.3749  18.4051

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)        27.3371966  6.1786867   4.424 1.61e-05 ***
progacademic        2.8270764  7.5613555   0.374  0.70890
progvocational     -2.8858077  8.3682061  -0.345  0.73058
read                0.4822806  0.1221409   3.949  0.00011 ***
progacademic:read  -0.0176797  0.1441267  -0.123  0.90250
progvocational:read 0.0005898  0.1712196   0.003  0.99725
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 7.482 on 194 degrees of freedom
Multiple R-squared: 0.3926,     Adjusted R-squared: 0.3769
F-statistic: 25.08 on 5 and 194 DF,  p-value: < 2.2e-16
```

Regression 4

- To get the multi-degree of freedom test for the interaction, we can compare a model with and without the interaction.

```
m3b <- update(m3, . ~ . - prog:read)
anova(m3b, m3)
```

```
> 
> m3b <- update(m3, . ~ . - prog:read)
> anova(m3b, m3)
Analysis of Variance Table

Model 1: write ~ prog + read
Model 2: write ~ prog * read
  Res.Df   RSS Df Sum of Sq      F Pr(>F)
1    196 10861
2    194 10860  2    1.3736 0.0123 0.9878
```

Estimated Means 1

- We can get the estimated (predicted) cell means using predict.
- First, using expand.grid we create a new dataset containing all possible crossings of predictor values for which we would like to predict the outcome. We then use the model coefficients to predict our cell means, and store them back in the new dataset.

```
newdat <- with(d, expand.grid(prog = levels(prog), female = levels(female)))
(newdat <- cbind(newdat, predict(m, newdat, se=TRUE)))
```

```
> newdat <- with(d, expand.grid(prog = levels(prog), female = levels(fem$
> (newdat <- cbind(newdat, predict(m, newdat, se=TRUE)))
        prog female      fit   se.fit  df residual.scale
1    general   male 49.14286 1.803321 194       8.263856
2   academic   male 54.61702 1.205407 194       8.263856
3 vocational   male 41.82609 1.723133 194       8.263856
4    general female 53.25000 1.686852 194       8.263856
5   academic female 57.58621 1.085097 194       8.263856
6 vocational female 50.96296 1.590380 194       8.263856
>
>
```

Estimated Means 2

- Plots of predicted values can be nice.

```
ggplot(newdat, aes(x = prog, y = fit, colour = female)) +
  geom_errorbar(aes(ymin = fit - se.fit, ymax = fit + se.fit), width=.25) +
  geom_point(size=3)
```