

Handout 2: Introduction to **R**

Dave Armstrong
University of Wisconsin – Milwaukee
Department of Political Science

e: armstrod@uwm.edu

w: www.quantoid.net/ICPSR.php

Contents

1	Some basic statistical routines	2
2	Filtering with Logical Expressions and Sorting	3
2.1	Sorting	5
3	Working with Real Data	6
3.1	Tables and Cross-Tabulations	6
3.2	Linear Models	8
3.2.1	Adjusting the base category	10
3.2.2	Predict after lm	11
3.2.3	Linear Hypothesis Tests	14

1 Some basic statistical routines

Just to get you thinking about how we might extend our use of **R** over the next few days, let's talk about summaries, means and variances for the variables in `stata.dat`. We just saw above how to extract a single variable using the `$`, now how would we take the mean of variable `x1`? Well, there is a function in **R** called `mean()` which calculates the mean of a variable.

```
  x1  x2 x3
1   1 none yes
2   2 none no
3   3 some no
4   4 none yes
5   3 none no
6   4 none yes
7   1 some yes
8   2 some yes
9   5 some no
10  6 none no
```

```
  x1  x2 x3
1   1 none yes
2   2 none no
3  NA some no
4   4 <NA> yes
5   3 none no
6   4 none .
7   1 some yes
8   2 some yes
9   5 some no
10  6 none no
```

```
[1] yes no no yes no . yes yes no no
Levels: . no yes
```

```
  x1  x2 x3 x3fac
1   1 none yes yes
2   2 none no no
3  NA some no no
4   4 <NA> yes yes
5   3 none no no
6   4 none . <NA>
7   1 some yes yes
8   2 some yes yes
9   5 some no no
10  6 none no no
```

```

      x1  x2  x3 x3fac
1     1 none  yes  yes
2     2 none  no   no
3    NA some  no   no
4     4 <NA> yes  yes
5     3 none  no   no
6     4 none <NA> <NA>
7     1 some  yes  yes
8     2 some  yes  yes
9     5 some  no   no
10    6 none  no   no

```

```
> mean(stata.dat$x1)
```

```
[1] 3.1
```

The command for calculating the variance is `var()`:

```
> var(stata.dat$x1)
```

```
[1] 2.766667
```

We can also summarize the entire dataset using the `summary()` command:

```
> summary(stata.dat)
```

```

      x1      x2      x3
Min.   :1.0  none:6  Length:10
1st Qu.:2.0  some:4  Class  :character
Median :3.0                      Mode   :character
Mean   :3.1
3rd Qu.:4.0
Max.   :6.0

```

Notice here, we get some quantiles and the mean for numeric variables, frequencies for factors and no information about the specific cell values for character strings.

2 Filtering with Logical Expressions and Sorting

At the end of the previous lecture we talked a bit about filtering with logical expressions, though it wasn't in the notes. I thought for the sake of completeness that I would reproduce some of that here. A logical expression is one that evaluates to either **TRUE** (the condition is met) or **FALSE** (the condition is not met). There are a few operators you need to know (which are the same as the operators in Stata or SPSS).

EQUALITY `==` (two equal signs) is the symbol for logical equality. `A == B` evaluates to **TRUE** if A is equivalent to B and evaluates to **FALSE** otherwise.

INEQUALITY `!=` is the command for inequality. `A != B` evaluates to **TRUE** when A is not equivalent to B.

AND `&` is the conjunction operator. `A & B` would evaluate to **TRUE** if both A and B were met. It would evaluate to **FALSE** if either A and/or B were not met.

OR `|` (the pipe character) is the logical or operator. `A | B` would evaluate to **TRUE** if either A and/or B is met and would evaluate to **FALSE** only if neither A nor B were met.

NOT `!` (the exclamation point) is the character for logical negation. `!(A & B)` is the mirror image of `(A & B)` such that the latter evaluates to **TRUE** when the former evaluates to **FALSE**.

When using these with variables, the conditions for character strings should be specified with characters. With numeric variables, the conditions should be specified using numbers. With factors, either the numerical value or the label can be used. A few examples will help to illuminate things here.

```
> stata.dat$x3 == "yes"
```

```
[1] TRUE FALSE FALSE TRUE FALSE TRUE TRUE TRUE FALSE FALSE
```

```
> stata.dat$x2 == "none"
```

```
[1] TRUE TRUE FALSE TRUE TRUE TRUE FALSE FALSE FALSE TRUE
```

```
> stata.dat$x2 == 1
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
> stata.dat$x1 == 2
```

```
[1] FALSE TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE
```

As I suggested yesterday, you can use a logical expression to subset a matrix and you will only see the observations where the conditional statement evaluates to **TRUE**. Let's use this to subset our dataset.

```
> stata.dat[stata.dat$x1 == 1 & stata.dat$x2 == "none", ]
```

```
  x1  x2  x3
1  1 none yes
```

2.1 Sorting

Sorting is something that is a bit less intuitive. Here, we have to leverage the fact that we can re-arrange the rows of a matrix by simply mixing up the row indices.

```
> stata.dat
```

```
   x1  x2 x3
1   1 none yes
2   2 none no
3   3 some no
4   4 none yes
5   3 none no
6   4 none yes
7   1 some yes
8   2 some yes
9   5 some no
10  6 none no
```

```
> stata.dat[c(10, 9, 8, 7, 6, 5, 4, 3, 2, 1), ]
```

```
   x1  x2 x3
10  6 none no
9   5 some no
8   2 some yes
7   1 some yes
6   4 none yes
5   3 none no
4   4 none yes
3   3 some no
2   2 none no
1   1 none yes
```

There is a command in **R** called `order` which will tell you the ordering of observations based on the value of some variable. This gives us the order from smallest to largest of the observations based on the variable `x1`.

```
> order(stata.dat$x1)
```

```
[1] 1 7 2 8 3 5 4 6 9 10
```

We can use this to sort the dataset as follows:

```
> stata.dat_reorder <- stata.dat[order(stata.dat$x1), ]
> stata.dat_reorder
```

```

      x1  x2  x3
1     1  none yes
7     1  some yes
2     2  none  no
8     2  some yes
3     3  some  no
5     3  none  no
4     4  none yes
6     4  none yes
9     5  some  no
10    6  none  no

```

Now, there are two datasets in our workspace, one ordered on `x1` and one original one. Both contain exactly the same information, but sorted a different way.

3 Working with Real Data

Now, I thought we would work with some “real” data. For these exercises, we’re going to use the `Mroz` data from the `car` package. These are data from the Panel Study of Income Dynamics (PSID) dealing with female labor market participation. Looking at the help file for `Mroz`, you can see the variable names and descriptions:

```
lfp labor-force participation; a factor with levels: 'no'; 'yes'.
```

```
k5 number of children 5 years old or younger.
```

```
k618 number of children 6 to 18 years old.
```

```
age in years.
```

```
wc wife's college attendance; a factor with levels: 'no'; 'yes'.
```

```
hc husband's college attendance; a factor with levels: 'no';
'yes'.
```

```
lwg log expected wage rate; for women in the labor force, the
actual wage rate; for women not in the labor force, an
imputed value based on the regression of 'lwg' on the other
variables.
```

```
inc family income exclusive of wife's income.
```

3.1 Tables and Cross-Tabulations

Just as any other statistical language, **R** provides you with a relatively easy way of making tables and cross-tabulations. There is a command called `table` that makes either one- or

two-way tables. If you simply want to see the marginal distribution of one variable (for example, `wc`), you can do the following:

```
> table(Mroz$wc)
```

```
no yes  
541 212
```

If we wanted to know what the proportion was at each level, we could do the following:

```
> sum(table(Mroz$wc))
```

```
[1] 753
```

```
> table(Mroz$wc)/753
```

```
no yes  
0.7184595 0.2815405
```

Or, we could simply cut out the middle-man and do the following:

```
> table(Mroz$wc)/sum(table(Mroz$wc))
```

```
no yes  
0.7184595 0.2815405
```

If you want two-way cross-tabs, you can get those either with the `table` command, or with more information from the `CrossTable` command from the `gmodels` package.

```
> library(gmodels)
```

```
> table(Mroz$wc, Mroz$hc)
```

```
no yes  
no 417 124  
yes 41 171
```

```
> CrossTable(Mroz$wc, Mroz$hc, chisq = T)
```

```
Cell Contents  
|-----|  
| N |  
| Chi-square contribution |  
| N / Row Total |  
| N / Col Total |  
| N / Table Total |  
|-----|
```

Total Observations in Table: 753

Mroz\$wc	Mroz\$hc		Row Total
	no	yes	
no	417	124	541
	23.505	36.492	
	0.771	0.229	0.718
	0.910	0.420	
	0.554	0.165	
yes	41	171	212
	59.982	93.125	
	0.193	0.807	0.282
	0.090	0.580	
	0.054	0.227	
Column Total	458	295	753
	0.608	0.392	

Statistics for All Table Factors

Pearson's Chi-squared test

Chi^2 = 213.1042 d.f. = 1 p = 2.888253e-48

Pearson's Chi-squared test with Yates' continuity correction

Chi^2 = 210.688 d.f. = 1 p = 9.722333e-48

3.2 Linear Models

There are tons of linear models presentation and diagnostic tools in **R**. We will first look at how to estimate a linear model using the Duncan data from the `car` package. These are OD Duncan's data on occupational prestige.

```
type Type of occupation. A factor with the following levels:
      'prof', professional and managerial; 'wc', white-collar;
      'bc', blue-collar.

income Percent of males in occupation earning $3500 or more in
      1950.

education Percent of males in occupation in 1950 who were
      high-school graduates.

prestige Percent of raters in NORC study rating occupation as
      excellent or good in prestige.
```

This will give me a chance to show how factors work in the linear model context.

At the heart of the modeling functions in **R** is the *formula*. Particularly the dependent variable is given first then a tilde `~` and the independent variables are then given separated by `+`. For example: `prestige ~ income + type` is a formula. Now, we have to tell **R** in what context it should evaluate that formula. For our purposes today, we'll be using the `lm` function. This will estimate an OLS regression (unless otherwise indicated with weights).

```
> lm(prestige ~ income + type, data = Duncan)
```

Call:

```
lm(formula = prestige ~ income + type, data = Duncan)
```

Coefficients:

(Intercept)	income	typeprof	typewc
6.7039	0.6758	33.1557	-4.2772

```
> mod <- lm(prestige ~ income + type, data = Duncan)
> summary(mod)
```

Call:

```
lm(formula = prestige ~ income + type, data = Duncan)
```

Residuals:

Min	1Q	Median	3Q	Max
-23.243	-6.841	-0.544	4.295	32.949

Coefficients:

Estimate	Std. Error	t value	Pr(> t)
----------	------------	---------	----------

```

(Intercept) 6.70386    3.22408    2.079    0.0439 *
income      0.67579    0.09377    7.207 8.43e-09 ***
typeprof   33.15567    4.83190    6.862 2.58e-08 ***
typewc     -4.27720    5.54974   -0.771 0.4453
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

```

Residual standard error: 10.68 on 41 degrees of freedom
Multiple R-squared: 0.893, Adjusted R-squared: 0.8852
F-statistic: 114 on 3 and 41 DF, p-value: < 2.2e-16

```

We have saved our model object as `mod`. If we want to see what pieces of information are in the little box labeled `mod`, we can simply type the following:

```

> names(mod)

 [1] "coefficients" "residuals"    "effects"      "rank"
 [5] "fitted.values" "assign"       "qr"          "df.residual"
 [9] "contrasts"    "xlevels"     "call"        "terms"
[13] "model"

```

3.2.1 Adjusting the base category

It is relatively easy to adjust the base category of the factor here. We simply need to manipulate the variable's contrasts. There are many different types of these, but the one we most usually think of are *treatment contrasts*. Treatment contrasts make dummy variables for all but one level of the categorical variable, leaving one out as the base category. The first level is the one that is left out by default. We can see what the dummy variables will look like by typing:

```

> contrasts(Duncan$type)

      prof wc
bc      0  0
prof    1  0
wc      0  1

```

We can modify these by re-specifying the contrasts:

```

> contrasts(Duncan$type) <- contr.treatment(3, base = 2)
> contrasts(Duncan$type)

      1 3
bc    1 0
prof  0 0
wc    0 1

> lm(prestige ~ type, data = Duncan)

```

```
Call:
lm(formula = prestige ~ type, data = Duncan)
```

```
Coefficients:
(Intercept)      type1      type3
      80.44      -57.68      -43.78
```

Another way of accomplishing the same goal is to use the simpler, but less flexible `relevel` command. This command takes arguments - the variable and the new base level. For example

```
> data(Duncan)
> Duncan$type <- relevel(Duncan$type, "prof")
> lm(prestige ~ income + type, data = Duncan)
```

```
Call:
lm(formula = prestige ~ income + type, data = Duncan)
```

```
Coefficients:
(Intercept)      income      typebc      typewc
      39.8595      0.6758     -33.1557     -37.4329
```

If you wanted deviation or effects coding, you could chose the `contr.sum` contrasts:

```
> contrasts(Duncan$type) <- "contr.sum"
> lm(prestige ~ type, data = Duncan)
```

```
Call:
lm(formula = prestige ~ type, data = Duncan)
```

```
Coefficients:
(Intercept)      type1      type2
      46.62      33.82     -23.86
```

Typing `?contrasts` will give you the help file on the different types of contrasts available in R.

3.2.2 Predict after lm

There are a number of different ways you can get fitted values after you've estimated a linear model. If you only want to see the fitted values for each observation, you can use

```
> data(Duncan)
> Duncan$type <- relevel(Duncan$type, "prof")
> mod <- lm(prestige ~ income + type, data = Duncan)
> summary(mod)
```

Call:

```
lm(formula = prestige ~ income + type, data = Duncan)
```

Residuals:

Min	1Q	Median	3Q	Max
-23.243	-6.841	-0.544	4.295	32.949

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	39.85953	6.16834	6.462	9.53e-08 ***
income	0.67579	0.09377	7.207	8.43e-09 ***
typebc	-33.15567	4.83190	-6.862	2.58e-08 ***
typewc	-37.43286	5.11026	-7.325	5.75e-09 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.68 on 41 degrees of freedom

Multiple R-squared: 0.893, Adjusted R-squared: 0.8852

F-statistic: 114 on 3 and 41 DF, p-value: < 2.2e-16

> mod\$fitted

accountant	pilot	architect	author
81.75848	88.51638	90.54374	77.02795
chemist	minister	professor	dentist
83.11006	54.05111	83.11006	93.92269
reporter	engineer	undertaker	lawyer
47.70456	88.51638	68.24269	91.21953
physician	welfare.worker	teacher	conductor
91.21953	67.56690	72.29743	53.78667
contractor	factory.owner	store.manager	banker
75.67637	80.40690	68.24269	92.57111
bookkeeper	mail.carrier	insurance.agent	store.clerk
22.02456	34.86456	39.59509	22.02456
carpenter	electrician	RR.engineer	machinist
20.89544	38.46597	61.44281	31.03228
auto.repairman	plumber	gas.stn.attendant	coal.miner
21.57123	36.43860	16.84070	11.43438
streetcar.motorman	taxi.driver	truck.driver	machine.operator
35.08702	12.78596	20.89544	20.89544
barber	bartender	shoe.shiner	cook
17.51649	17.51649	12.78596	16.16491
soda.clerk	watchman	janitor	policeman
14.81333	18.19228	11.43438	29.68070
waiter			
12.11017			

```
> fitted(mod)
```

accountant	pilot	architect	author
81.75848	88.51638	90.54374	77.02795
chemist	minister	professor	dentist
83.11006	54.05111	83.11006	93.92269
reporter	engineer	undertaker	lawyer
47.70456	88.51638	68.24269	91.21953
physician	welfare.worker	teacher	conductor
91.21953	67.56690	72.29743	53.78667
contractor	factory.owner	store.manager	banker
75.67637	80.40690	68.24269	92.57111
bookkeeper	mail.carrier	insurance.agent	store.clerk
22.02456	34.86456	39.59509	22.02456
carpenter	electrician	RR.engineer	machinist
20.89544	38.46597	61.44281	31.03228
auto.repairman	plumber	gas.stn.attendant	coal.miner
21.57123	36.43860	16.84070	11.43438
streetcar.motorman	taxi.driver	truck.driver	machine.operator
35.08702	12.78596	20.89544	20.89544
barber	bartender	shoe.shiner	cook
17.51649	17.51649	12.78596	16.16491
soda.clerk	watchman	janitor	policeman
14.81333	18.19228	11.43438	29.68070
waiter			
12.11017			

Both of the above commands will produce the same output - a predicted value for each observation. We could also get fitted values using the `predict` command which will also calculate standard errors or confidence bounds.

```
> pred <- predict(mod)
> pred.se <- predict(mod, se.fit = T)
> pred.mean.ci <- predict(mod, interval = "confidence")
> pred.ind.ci <- predict(mod, interval = "prediction")
```

Notice, that when the original data are used, the fourth option indicates that these confidence intervals are for future predictions. They are not meant to say something interesting about the observed data, rather about future or hypothetical cases.

It is also possible to make out-of-sample predictions. To do this, you need to make a new data frame that has the same variables that are in your model, with values at which you want to get predictions. Let's say that above, we wanted to get predictions for a single observation that had an income value of 50 and had "blue collar" as the type. We could do the following:

```
> newdat <- data.frame(income = 50, type = "bc")
> predict(mod, newdat, interval = "confidence")
```

```

      fit      lwr      upr
1 40.49334 33.64969 47.33698

```

If we wanted to get predictions for “blue collar” occupations with incomes of 40, 50 and 60, we could do that as follows:

```

> newdat <- data.frame(income = c(40, 50, 60), type = c("bc", "bc",
+ "bc"))
> predict(mod, newdat, interval = "confidence")

```

```

      fit      lwr      upr
1 33.73544 28.11384 39.35704
2 40.49334 33.64969 47.33698
3 47.25123 38.93011 55.57236

```

3.2.3 Linear Hypothesis Tests

You can use the `linearHypothesis` command in R to test any hypothesis you want about any linear combination of parameters (this is akin to `lincom` in Stata). For example, let’s say that we wanted to test the hypothesis that $\beta_{\text{income}} = 1$, we could do:

```

> linearHypothesis(mod, "income = 1")

```

Linear hypothesis test

Hypothesis:

income = 1

Model 1: restricted model

Model 2: prestige ~ income + type

```

      Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1         42 6038.3
2         41 4675.2  1    1363.1 11.954 0.001284 **

```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

If you wanted to test whether $\beta_{bc} = \beta_{wc}$, you could do:

```

> linearHypothesis(mod, "typebc=typewc")

```

Linear hypothesis test

Hypothesis:

typebc - typewc = 0

Model 1: restricted model

Model 2: prestige ~ income + type

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	42	4742.9				
2	41	4675.2	1	67.731	0.594	0.4453

If we wanted to test whether both were simultaneously zero, rather than just the same, we could use:

```
> linearHypothesis(mod, c("typebc=0", "typewc=0"))
```

Linear hypothesis test

Hypothesis:

typebc = 0

typewc = 0

Model 1: restricted model

Model 2: prestige ~ income + type

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	43	13022.8				
2	41	4675.2	2	8347.6	36.603	7.575e-10 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Or, we could use Anova to accomplish the same goal:

```
> Anova(mod)
```

Anova Table (Type II tests)

Response: prestige

	Sum Sq	Df	F value	Pr(>F)
income	5922.4	1	51.938	8.428e-09 ***
type	8347.6	2	36.603	7.575e-10 ***
Residuals	4675.2	41		

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1