

Example R session

A few miscellaneous notes:

1. R is case sensitive, but insensitive to white space.
2. Start variable names with a letter, digits and periods can also be used.
3. Use the up arrow key to repeat previous commands. These can then be edited before hitting enter.
4. If you hit enter before the end of a command, e.g., R is expecting a “)”, then R will start the next line with a “+” prompt allowing you to finish the command.
5. All file path names need to use the forward slash / (as in Unix) not the windows backward slash .

To see what is in your workspace:

```
ls()
```

```
## character(0)
```

The following command clears the workspace.

```
rm(list=ls())  
ls()
```

```
## character(0)
```

R as calculator:

```
2+3
```

```
## [1] 5
```

```
2+3*4
```

```
## [1] 14
```

The assignment operator is “<-“. That is, a less than sign followed by a hyphen.

```
x <- 2  
y <- 3  
z <- 4  
x+y*z
```

```
## [1] 14
```

```
result <- x+y*z  
result
```

```
## [1] 14
```

Everything given a name is an object and is stored in the workspace.

```
ls()
```

```
## [1] "result" "x"      "y"      "z"
```

Vectors

Creating vectors, the : and c() function

```
x <- 1:5  
x
```

```
## [1] 1 2 3 4 5
```

```
y <- 11:15  
y
```

```
## [1] 11 12 13 14 15
```

```
z <- c(10,6,3,5,1)
z
```

```
## [1] 10 6 3 5 1
```

Indexing vectors

```
y[2]
```

```
## [1] 12
```

```
z[4]
```

```
## [1] 5
```

```
y[2:4]
```

```
## [1] 12 13 14
```

```
z[c(5,1,2)]
```

```
## [1] 1 10 6
```

Creating vectors, the seq() function

```
help(seq)
a <- seq(10,20)
a
```

```
## [1] 10 11 12 13 14 15 16 17 18 19 20
```

```
a <- seq(20,10)
a
```

```
## [1] 20 19 18 17 16 15 14 13 12 11 10
```

```
b <- seq(from=17,to=30,by=2)
b
```

```
## [1] 17 19 21 23 25 27 29
```

```
b <- seq(17,30,2)
b
```

```
## [1] 17 19 21 23 25 27 29
```

```
b <- seq(by=2,to=30,from=17)
b
```

```
## [1] 17 19 21 23 25 27 29
```

```
c <- seq(2,1,-.1)
c
```

```
## [1] 2.0 1.9 1.8 1.7 1.6 1.5 1.4 1.3 1.2 1.1 1.0
```

Functions that work on vectors

```
vec <- c(10,5,1,12,4)
vec
```

```
## [1] 10 5 1 12 4
```

```

vec^2
## [1] 100 25 1 144 16
vec + vec
## [1] 20 10 2 24 8
sum(vec)
## [1] 32
10 + 5 + 1 + 12 + 4
## [1] 32
mean(vec)
## [1] 6.4
32/5
## [1] 6.4
sum(vec)/length(vec)
## [1] 6.4
vec2 <- c(vec,10,10,4)
vec2
## [1] 10 5 1 12 4 10 10 4
table(vec2)
## vec2
## 1 4 5 10 12
## 1 2 1 3 1

```

Counting things in vectors

Comparison operators (==, <, >, <=, >=, !=),

Boolean operators and, or, not (|, &, !)

```

vec1 <- c(10, 5, 1, 12, 4)
vec1
## [1] 10 5 1 12 4
vec2 <- c(10, 5, 1, 12, 4, 10, 10, 4)
vec2==1
## [1] FALSE FALSE TRUE FALSE FALSE FALSE FALSE
sum(vec2 == 1)
## [1] 1
vec2 == 10
## [1] TRUE FALSE FALSE FALSE FALSE TRUE TRUE FALSE

```

```

sum(vec2 == 10)

## [1] 3
sum(vec2 < 10)

## [1] 4
sum(vec2 <= 10)

## [1] 7
vec2

## [1] 10  5  1 12  4 10 10  4
vec2 == 10 | vec2 == 1

## [1]  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE
sum(vec2 == 10 | vec2 == 1)

## [1] 4
vec2 > 1 & vec2 < 12

## [1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
sum(vec2 > 1 & vec2 < 12)

## [1] 6
sum(vec2 != 10)

## [1] 5

```

Creating two dimensional arrays (matrices) by using the `matrix()` function

```

help(matrix)
x <- matrix(1:12,nrow=3)
x

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12

dim(x)

## [1] 3 4

length(x)

## [1] 12

y <- matrix(1:12,nrow=4)
y

##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12

```

```
dim(y)
```

```
## [1] 4 3
```

```
z <- matrix(1:12,nrow=3,byrow=T)
```

```
temp <- c(5,10,12,24,42,60,63,72)
```

```
w <- matrix(temp,ncol=2)
```

```
w
```

```
##      [,1] [,2]
```

```
## [1,]    5  42
```

```
## [2,]   10  60
```

```
## [3,]   12  63
```

```
## [4,]   24  72
```

Indexing to extract values from an array

```
w
```

```
##      [,1] [,2]
```

```
## [1,]    5  42
```

```
## [2,]   10  60
```

```
## [3,]   12  63
```

```
## [4,]   24  72
```

```
w[4,2]
```

```
## [1] 72
```

```
w[1,2]
```

```
## [1] 42
```

```
z
```

```
##      [,1] [,2] [,3] [,4]
```

```
## [1,]    1    2    3    4
```

```
## [2,]    5    6    7    8
```

```
## [3,]    9   10   11   12
```

```
z[3,3]
```

```
## [1] 11
```

```
z[1:2,1:2]
```

```
##      [,1] [,2]
```

```
## [1,]    1    2
```

```
## [2,]    5    6
```

Indexing to get an entire row or column of an array

```
z
```

```
##      [,1] [,2] [,3] [,4]
```

```
## [1,]    1    2    3    4
```

```
## [2,]    5    6    7    8
```

```
## [3,]    9   10   11   12
```

```

z[1, ]
## [1] 1 2 3 4
z[ ,1]
## [1] 1 5 9
w
##      [,1] [,2]
## [1,]    5  42
## [2,]   10  60
## [3,]   12  63
## [4,]   24  72
w[ ,2]
## [1] 42 60 63 72

```

Reading in data from a text file using the read.table function.

Note I'm reading this as a local file in the same directory as my R session (.RData file). A full path name will also work, but you need to use front slashes (as in Unix) not back slashes (as in Windows). R can also read web address path names.

Open the file in a text editor: the variable names, "x1", "x2", and "y", are in the first line of the data file: hence "header=TRUE" below.

```

help(read.table)
example.data <- read.table(file="example.dat", header=TRUE)

```

head() and tail() show the first and last six cases

```

head(example.data)
##      x1      x2      y
## 1 20.42405 0.3015210 36.172660
## 2 32.74049 1.3469176 28.415784
## 3 24.94999 1.1793272 28.376674
## 4 27.72610 0.9360980 -4.515564
## 5 13.54863 2.0794730 12.169991
## 6 20.98872 0.9834769 27.996572

tail(example.data)
##      x1      x2      y
## 95 15.55136 0.1174105 13.759553
## 96 24.08216 1.4166113 16.811138
## 97 21.98282 1.3674975 25.881897
## 98 11.77869 1.4063972 28.399718
## 99 10.40391 1.8212314  4.083947
## 100 22.86390 1.2976605 23.642835

```

The following shows the first 10 cases in the data file. "[1:10,]" means rows 1 through 10 and all of the columns. "[4,3]" would refer to the value in the 4th row, 3rd column. "[,1:2]" all of the rows and the first two columns (x1 and x2). In order not to have all 100 cases displayed for this last example, I used the head() function to show only the first six.

```
example.data[1:10, ]
```

```
##           x1           x2           y
## 1  20.42405  0.3015210  36.172660
## 2  32.74049  1.3469176  28.415784
## 3  24.94999  1.1793272  28.376674
## 4  27.72610  0.9360980  -4.515564
## 5  13.54863  2.0794730  12.169991
## 6  20.98872  0.9834769  27.996572
## 7  15.17568  0.9979689  23.448322
## 8  32.34730  1.0945375  23.722077
## 9  24.18020  0.2266746  17.164225
## 10 17.83226  0.6170786  18.524674
```

```
example.data[4,3]
```

```
## [1] -4.515564
```

```
head( example.data[ ,1:2] )
```

```
##           x1           x2
## 1  20.42405  0.3015210
## 2  32.74049  1.3469176
## 3  24.94999  1.1793272
## 4  27.72610  0.9360980
## 5  13.54863  2.0794730
## 6  20.98872  0.9834769
```

Data frames (data.frame)

Data frames are a special kind of list that allows for both numeric and character variables as you would find in the usual flat tabular data file. The result of the `read.table()` function is a `data.frame`.

```
is.array(example.data)
```

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

```
is.list(example.data)
```

```
## [1] TRUE
```

```
is.data.frame(example.data)
```

```
## [1] TRUE
```

```
dim(example.data)
```

```
## [1] 100    3
```

```
nrow(example.data)
```

```
## [1] 100
```

```
ncol(example.data)
```

```
## [1] 3
```

```
names(example.data)
```

```
## [1] "x1" "x2" "y"
```

```
str(example.data)
```

```
## 'data.frame': 100 obs. of 3 variables:  
## $ x1: num 20.4 32.7 24.9 27.7 13.5 ...  
## $ x2: num 0.302 1.347 1.179 0.936 2.079 ...  
## $ y : num 36.17 28.42 28.38 -4.52 12.17 ...
```

Indexing individual values

```
example.data[1,1]
```

```
## [1] 20.42405
```

```
example.data$x1[1]
```

```
## [1] 20.42405
```

```
example.data[8,3]
```

```
## [1] 23.72208
```

```
example.data$y[8]
```

```
## [1] 23.72208
```

Vector components of data frames

```
is.vector(example.data$x1)
```

```
## [1] TRUE
```

```
length(example.data$x1)
```

```
## [1] 100
```

```
mean(example.data$x1)
```

```
## [1] 19.86305
```

```
mean(example.data[,1])
```

```
## [1] 19.86305
```

```
example.data[2,]
```

```
##      x1      x2      y  
## 2 32.74049 1.346918 28.41578
```

```
sum(example.data[2,])
```

```
## [1] 62.50319
```

```
min(example.data$y)
```

```
## [1] -4.515564
```

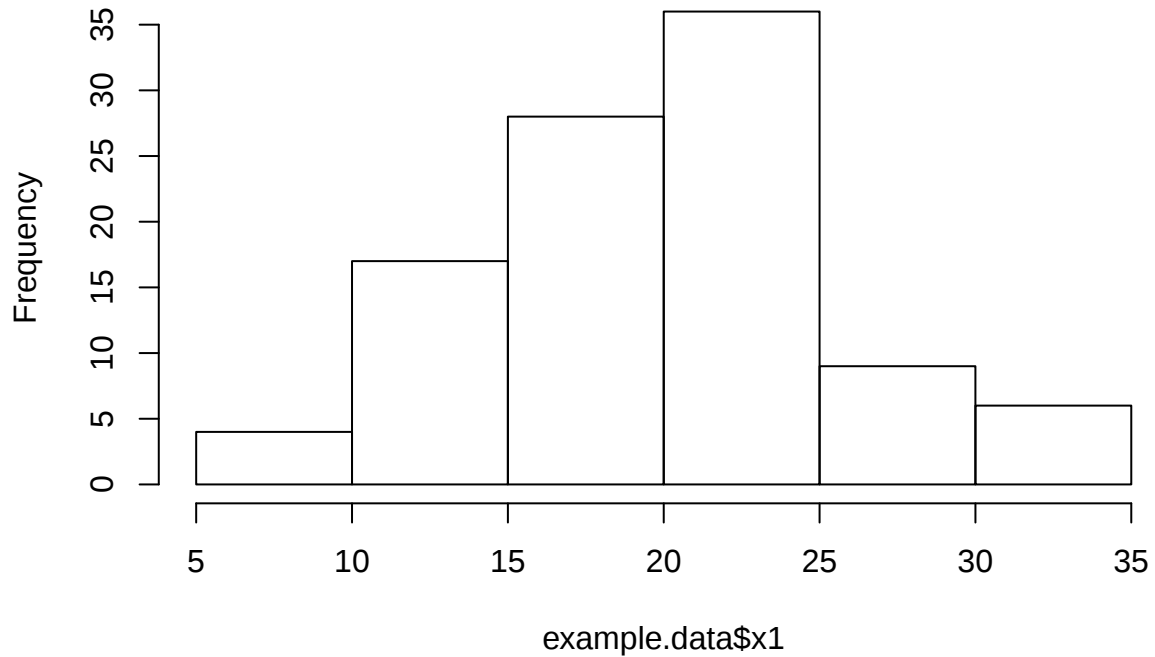
```
max(example.data$y)
```

```
## [1] 47.21957
```

A few quick and dirty plots

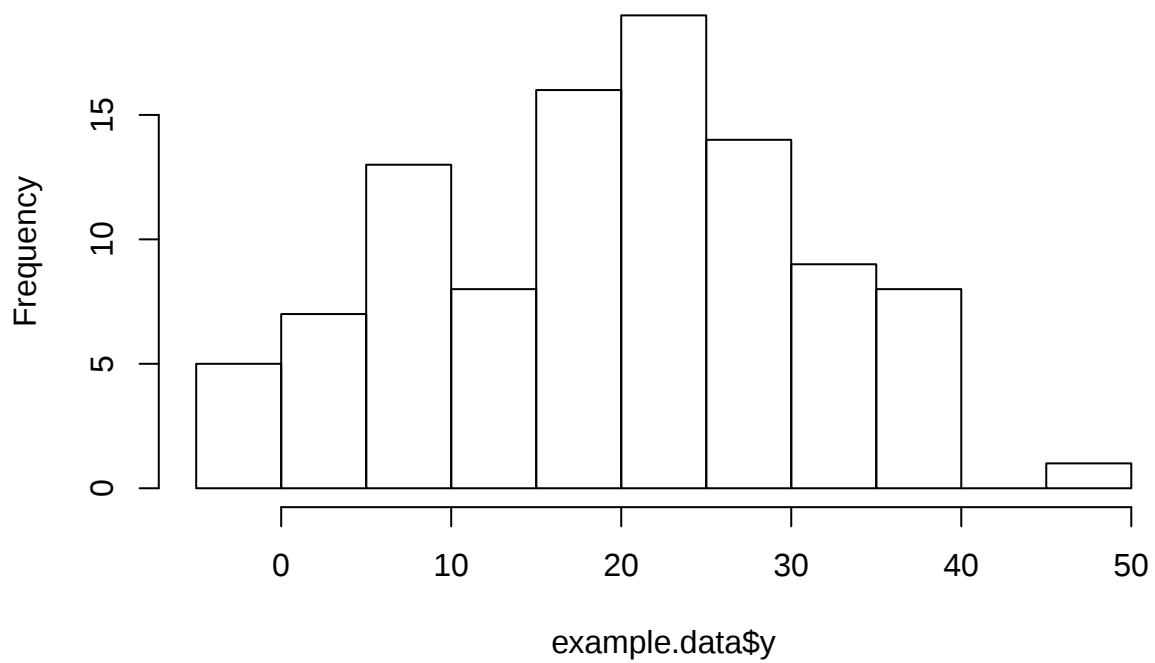
```
hist(example.data$x1)
```

Histogram of example.data\$x1

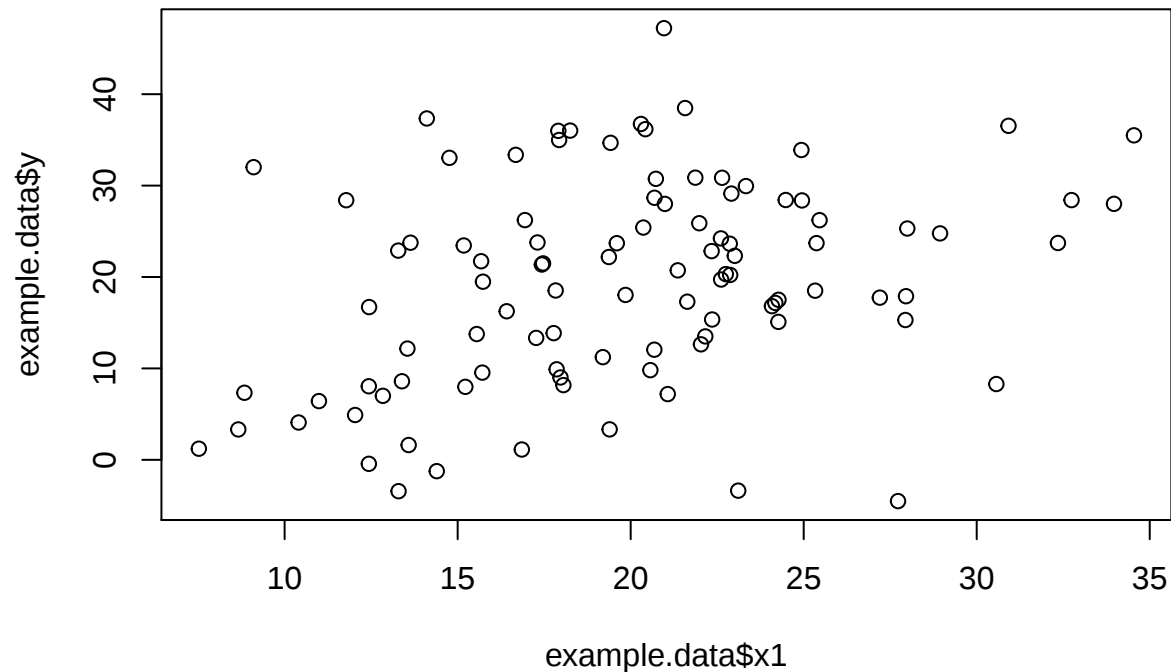


```
hist(example.data$y)
```

Histogram of example.data\$y



```
plot(example.data$x1,example.data$y)
```



Multiple Linear Regression with the `lm()` function

Open the description of `lm` using `help`. Note that `lm` expects the first parameter to be a formula. In this example, the formula is `y~x1+x2`. This indicates that `y` is the dependent variable and `x1` and `x2` are the independent variables (covariates). By default an intercept term is used. To explicitly ask for an intercept term use `y~1+x1+x2`. To eliminate the intercept term use `y~0+x1+x2`.

```
lm.results <- lm(y~x1+x2,data=example.data)
```

Note that “`lm.results`” is the name of the object that stores all the information from the `lm` function. To extract the usual regression table results from `lm.results`, use the `summary` function. The `summary` function can tell what kind of object is passed to it and responds appropriately.

```
summary(lm.results)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2, data = example.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -28.8878  -7.0357  -0.7815   7.2315  27.4582
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.9103     4.1817   1.413 0.160753
## x1             0.6385     0.1874   3.407 0.000958 ***
## x2             0.8105     2.1384   0.379 0.705508
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 10.65 on 97 degrees of freedom
## Multiple R-squared:  0.1139, Adjusted R-squared:  0.09566
## F-statistic: 6.236 on 2 and 97 DF,  p-value: 0.002833
```

The `aov()` function extracts the sum of squares from `lm.results`. The `summary` function for `aov` objects, prints this in a table.

```
summary(aov(lm.results))
```

```
##              Df Sum Sq Mean Sq F value    Pr(>F)
## x1             1   1399   1399.2   12.328 0.000679 ***
## x2             1     16     16.3    0.144 0.705508
## Residuals     97   11009    113.5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Example of a more complex data set in .csv format

Look at the `test.csv` file using Excel The blank cells (missing data) will be converted to the R numeric missing value NA

```
dd <- read.csv(file="test.csv",header=T)
str(dd)
```

```
## 'data.frame':    200 obs. of  5 variables:
## $ age : int  30 35 45 61 27 61 63 49 48 20 ...
## $ sex : int  0 0 0 0 0 0 0 0 0 0 ...
## $ race: int  0 1 0 0 2 2 0 1 0 0 ...
## $ LE  : int  19 29 28 24 27 33 NA 39 21 28 ...
## $ dep : int  16 15 24 21 29 25 10 21 29 31 ...
```

```
head(dd)
```

```
##   age sex race LE dep
## 1  30   0   0 19  16
## 2  35   0   1 29  15
## 3  45   0   0 28  24
## 4  61   0   0 24  21
## 5  27   0   2 27  29
## 6  61   0   2 33  25
```

```
tail(dd)
```

```
##   age sex race LE dep
## 195 31   1   0 35  24
## 196 46   1   0 18  15
## 197 23   1   0 20  14
## 198 58   1   0 26  33
## 199 33   1   0 16  10
## 200 NA   1   0 23  33
```

Crude frequency tables using the `table()` function. Setting the `exclude` tag to `NULL` causes the missing values to be counted.

```
table(dd$race)
```

```
##
##    0    1    2
## 137   39   23
```

```
sum(table(dd$race))
```

```
## [1] 199
```

```
table(dd$race,exclude=NULL)
```

```
##
```

```
##      0      1      2 <NA>
```

```
## 137   39   23     1
```

```
table(dd$sex,exclude=NULL)
```

```
##
```

```
##      0      1 <NA>
```

```
##   80  120     0
```

```
table(dd$sex,dd$race,exclude=NULL)
```

```
##
```

```
##           0  1  2 <NA>
```

```
##  0      53 18  9     0
```

```
##  1      84 21 14     1
```

```
## <NA>    0  0  0     0
```

Categorical variables in R are called factors. Variables read in as numeric must be converted to factor variables to be processed properly statistically. In our example data set, we need to make race a factor.

```
is.numeric(dd$race)
```

```
## [1] TRUE
```

```
is.factor(dd$race)
```

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

```
dd$race <- as.factor(dd$race)
```

```
is.numeric(dd$race)
```

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

```
is.factor(dd$race)
```

```
## [1] TRUE
```

Multiple linear regression when one of the covariates is categorical

```
dep.fit <- lm(dep ~ age + sex + race + LE, data=dd)
```

```
summary(dep.fit)
```

```
##
```

```
## Call:
```

```
## lm(formula = dep ~ age + sex + race + LE, data = dd)
```

```
##
```

```
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
```

```
## -16.588  -4.703  -0.445   5.160  17.312
```

```
##
```

```
## Coefficients:
```

```
##              Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)  8.048733   2.408161   3.342  0.00100 **
```

```
## age          -0.006278    0.043425   -0.145    0.88521
## sex          2.680808    1.005462    2.666    0.00835 **
## race1       -2.439656    1.373056   -1.777    0.07723 .
## race2        1.967839    1.614197    1.219    0.22436
## LE           0.438257    0.097832    4.480    1.3e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.669 on 186 degrees of freedom
## (8 observations deleted due to missingness)
## Multiple R-squared:  0.1561, Adjusted R-squared:  0.1334
## F-statistic: 6.881 on 5 and 186 DF,  p-value: 6.458e-06
```

Creating a new data frame from an existing data frame using subset()

Selecting cases

```
newdat <- subset(dd,age<=35)
dim(newdat)
```

```
## [1] 65  5
```

```
head(newdat)
```

```
##   age sex race LE dep
## 1  30  0   0 19  16
## 2  35  0   1 29  15
## 5  27  0   2 27  29
## 10 20  0   0 28  31
## 11 27  0   0 19  19
## 12 26  0   0 17  14
```

```
max(newdat$age)
```

```
## [1] 35
```

Selecting variables

```
newdat2 <- subset(dd,select=c(age,race,dep))
dim(newdat2)
```

```
## [1] 200  3
```

```
head(newdat2)
```

```
##   age race dep
## 1  30   0  16
## 2  35   1  15
## 3  45   0  24
## 4  61   0  21
## 5  27   2  29
## 6  61   2  25
```

Adding a variable to a data frame

```
dd$ID <- 1001:1200
```

```
head(dd)
```

```
##   age sex race LE dep  ID
## 1  30   0    0 19  16 1001
## 2  35   0    1 29  15 1002
## 3  45   0    0 28  24 1003
## 4  61   0    0 24  21 1004
## 5  27   0    2 27  29 1005
## 6  61   0    2 33  25 1006
```

```
tail(dd)
```

```
##   age sex race LE dep  ID
## 195 31   1    0 35  24 1195
## 196 46   1    0 18  15 1196
## 197 23   1    0 20  14 1197
## 198 58   1    0 26  33 1198
## 199 33   1    0 16  10 1199
## 200 NA   1    0 23  33 1200
```