

## What is S-Plus

S-Plus is a flexible environment for doing data analysis. Among other things it offers

- an extensive and coherent collection of tools for statistics and data analysis,
- a language for expressing statistical models and tools for using linear and non-linear statistical models,
- graphical facilities for data analysis and display either at a workstation or as hardcopy,
- an effective object-oriented programming language which can easily be extended by the user community.

(Venables and Ripley, page 1)

## Before Starting

Before starting S-Plus for the *first time*, type at the *UNIX* prompt the following

```
S CHAPTER
```

and press return.

This creates a folder, `.Data` in your current folder (where you executed the command `S CHAPTER`. In the folder `.Data`, S-Plus stores the objects created in a S-Plus session). Note that if you give the `S CHAPTER` command in your home directory, it creates a directory `MySwork` and places the `.Data` directory in that.

## Starting S-Plus

To start S-Plus, type

```
S -e
```

What you see know is the S-Plus prompt, `>`, a S-Plus *expression* is typed at the prompt and *evaluated* by pressing `return`. More than one expression can be typed in a single line by using `;` as a separator. Also the same expression can be typed in more than one line by pressing `return` when the expression is not completed. At that point, the prompt changes to `+` to show that the expression is unfinished. The symbol `#` tells S-Plus to ignore whatever is typed after the symbol on the command line (used to put comments).

The `-e` option used when starting S-Plus, in this case, tells S-Plus to use *emacs* type of command line editor. The following table is useful when editing your expressions at the S-Plus command line before evaluating:

---

Action	Keystroke
Backward character	Ctrl – b
Forward character	Ctrl – f
Previous line	Ctrl – p
Next line	Ctrl – n
Beginning of line	Ctrl – a
End of line	Ctrl – e
Delete character	Ctrl – d
Kill line	Ctrl – u

---

Note that these same keystrokes can be used in Unix with `tcsh`. If the above doesn't work in S-Plus, add the line

```
setenv S_CLEditor emacs
```

to your `.cshrc` file.

## Example S-Plus session

### Arithmetic

```
> 4 + 5 / 3      ## addition & division
[1] 5.666667
> (4 + 5) / 3    ## using parentheses
[1] 3
> log(10)        ## natural logarithm
[1] 2.302585
```

Output from an *expression* can be *assigned* to an object for later use. Assignments are done with the `<-` operator, or just `_` (easier to read `<-`).

```
> xbar <- (4 + 6 + 8 + 10)/4
                                ## assigns avg to xbar
> xbar                          ## the object xbar
[1] 7
> s <- sqrt(((−3)^2 + (−1)^2 + 1^2 + 3^2)/3)
> s
[1] 2.581989
> z <- (xbar - 10)/s
> z
[1] -1.161895
```

In this example, the value of expressions were assigned to objects and used one of the *functions* (`sqrt`) available in S-plus.

To see (list) the objects in a directory, use the `ls()` command

```
> ls()  
  
[1] ".Last.value" "cars"      "cityfuel"  
[4] "fuel.lm"     "highfuel" "mpg.lm"  
[7] "s"           "xbar"     "z"
```

To remove objects use the `rm()` command

```
> rm(s,z)  
  
> ls()  
  
[1] ".Last.value" "cars"      "cityfuel"  
[4] "fuel.lm"     "highfuel" "mpg.lm"  
[7] "xbar"
```

To quit S-Plus use the `q()` command. Note that all objects created in this session are available when S-Plus is started again (assuming that they weren't deleted, like `s` and `z` were).

## Objects in S-Plus

Objects are basically of three types: *vectors*, *lists*, and *functions*.

Vectors (numeric, logical, or characters strings)

```
> data <- c(4,6,8,10) ## concatenating
> data ## print data
[1] 4 6 8 10
> length(data) ## length of data
[1] 4
> data[3] ## 3rd element
[1] 8
> ind <- 2:3 ## sequence
> ind
[1] 2 3
> data[ind] ## subset
[1] 6 8
> data + 100
[1] 104 106 108 110
> sqrt(data) ## square root
[1] 2.000000 2.449490 2.828427 3.162278
> sum(data) ## sum of elements
[1] 28
```

Can do earlier example much more clearly using vectors

```
> xbar <- mean(data)    ## mean of data
> s <- sqrt( sum((data-xbar)^2) /
              (length(data)-1) )
> s.easy<-stdev(data)
> s
[1] 2.581989
> s.easy
[1] 2.581989
> z<-(xbar-10)/s
> z
[1] -1.161895
```

## Lists

Stores different types of data in a single object

```
> example.l
```

```
$id:
```

```
[1] 1 2 3 4
```

```
$data:
```

```
[1] 4 6 8 10
```

```
$xbar:
```

```
[1] 7
```

```
$sdev:
```

```
[1] 2.581989
```

```
> names(example.l) ## the names
```

```
[1] "id" "data" "xbar" "sdev"
```

```
> example.l$data ## item data of list
```

```
[1] 4 6 8 10
```

```
> example.l$data[2:3] ## elements 2-3 of  
data
```

```
[1] 6 8
```

## Matrices

Data are often stored as tables (matrices)

```
> mat<-cbind(id=1:5,iq=rnorm(5,mean=100,sd=15))
```

```
> mat
```

```
      id      iq
[1,]  1  66.57908
[2,]  2 118.39228
[3,]  3 123.40596
[4,]  4  92.13515
[5,]  5 106.26696
```

```
> dim(mat)      ## dimensions of mat
```

```
[1] 5 2
```

```
> dimnames(mat) ## row and column names
```

```
[[1]]:
```

```
character(0)
```

```
[[2]]:
```

```
[1] "id" "iq"
```

```

> mat[1:2,]      ## first 2 rows of mat
  id          iq
  1  66.57908
  2 118.39228

> mat[,2]       ## 2nd column, by number
66.57908 118.3923 123.406 92.13515 106.267

> mat[, 'iq']   ## 2nd column, by name

66.57908 118.3923 123.406 92.13515 106.267

> mat.2 <- mat[-5,] ## remove last line
> mat.2

  id          iq
  1  66.57908
  2 118.39228
  3 123.40596
  4  92.13515

```

```

> ind <- mat[, 'iq'] > 115 ## iq's > 115
> ind ## vector of logicals (True/False)
  F T T F F
> mat.3 <- mat[ind,]
> mat.3
  id      iq
  2 118.3923
  3 123.4060

```

Other logical operators for direct comparison are < (less than), == (equal to), <= (less or equal), >= (greater or equal), != (not equal). Also can use & (AND), | (OR), and ! (NOT).

```

> mat[!( (mat[,2]>85) & (mat[,2]<115) ),]
  id      iq
  1  66.57908
  2 118.39228
  3 123.40596

```

## Functions

There are many *functions* built into S-Plus. A few examples are:

Arithmetical: `abs`, `log`, `sqrt`, `sin`, `asin`.

Matrix: `t` (transpose), `%*%` (matrix mult),  
`solve`, `qr` (qr decomposition).

Statistical: `mean`, `median`, `var`, `quantile`,  
`summary`, `lm` (linear model).

Distributions: `dnorm`, `pnorm`, `qnorm`, and `rnorm` for the density, cdf, quantile, and random numbers respectively for the normal distribution. Similarly for other common distributions, e.g., for the densities `dt` (t), `df` (F), `dchisq` ( $\chi^2$ ), `dgamma` (gamma), `dexp` (exponential), `dbinom` (binomial), etc

You can also write your own *functions* in S-Plus.

Here is an example for calculating the t statistic for a t-test.

```
> my.ttest
function(x, null = 0)
{
    n <- length(x)
    xbar <- mean(x)
    sd <- stdev(x)
    se <- sd/sqrt(n)
    tstat <- (xbar - null)/se
    tstat
}
> my.ttest(data)
[1] 5.422177
```

Note that this function isn't needed as one already exists. Here is the S-Plus t-test function output

```
> t.test(data)
```

```
One-sample t-Test
```

```
data: data
```

```
t = 5.4222, df = 3, p-value = 0.0123
```

```
alternative hypothesis: true mean is not  
equal to 0
```

```
95 percent confidence interval:
```

```
2.891479 11.108521
```

```
sample estimates:
```

```
mean of x
```

```
7
```

## A Sample Analysis

Often data you have been working with has already been entered into a file. For example, the data I want to use for this example is in the file `/home/irwin/Scourse/93cars.dat`. This data set contains data on 93 different 1993 model year cars. In this file, columns are separated with spaces.

Importing data – `data.frame` and `read.table`

An object of class `data.frame` is a list object which looks and feels like a table. To read the above data into S-Plus use the command

```
> cars<-read.table("/home/irwin/Scourse/93cars.dat"  
+                 , header=T,row.names=NULL)
```

For the example, I will be using a reduced version of the data set with many of the variables deleted. The reduced form of the data set is called `cars.df`. To access the whole data set plus the reduced form, in S-Plus give the command

```
> attach("/home/irwin/Scourse/.Data")
```

```

> dim(cars.df)
[1] 93 10

> class(cars.df)
[1] "data.frame"

> names(cars.df)
[1] "manu"      "model"      "cylinders"  "type"       "engsize"   "weight"
[7] "citympg"   "cityfuel"   "highmpg"    "highfuel"

> cars.df[1:5,]
  manu  model cylinders  type engsize weight citympg cityfuel
1 Acura Integra      4 Small    1.8  2705    25 4.000000
2 Acura  Legend      6 Midsize  3.2  3560    18 5.555556
3 Audi    90         6 Compact  2.8  3375    20 5.000000
4 Audi   100         6 Midsize  2.8  3405    19 5.263158
5 BMW    535i         4 Midsize  3.5  3640    22 4.545455

  highmpg highfuel
1      31 3.225806
2      25 4.000000
3      26 3.846154
4      26 3.846154
5      30 3.333333

```

Now lets take a look at a summary of the data

```

> summary(cars.df)
      manu      model  cylinders      type      engsize
Ford: 8      Vision: 1    *: 1      Compact:16    Min.:1.000
Chevrolet: 8  Town_Car: 1    3: 3      Large:11     1st Qu.:1.800
Dodge: 6      Tercel: 1    4:49     Midsize:22    Median:2.400
Pontiac: 5     Tempo: 1    5: 2      Small:21     Mean:2.668
Mazda: 5      Taurus: 1    6:31     Sporty:14    3rd Qu.:3.300
Volkswagen: 4  Swift: 1    8: 7      Van: 9       Max.:5.700
(Other):57    (Other):87

      weight      citympg      cityfuel      highmpg
Min.:1695      Min.:15.00      Min.:2.174      Min.:20.00
1st Qu.:2620    1st Qu.:18.00    1st Qu.:4.000    1st Qu.:26.00
Median:3040     Median:21.00     Median:4.762     Median:28.00
Mean:3073       Mean:22.37       Mean:4.699       Mean:29.09
3rd Qu.:3525    3rd Qu.:25.00    3rd Qu.:5.556    3rd Qu.:31.00
Max.:4105       Max.:46.00       Max.:6.667       Max.:50.00

      highfuel
Min.:2.000
1st Qu.:3.226
Median:3.571
Mean:3.541
3rd Qu.:3.846
Max.:5.000

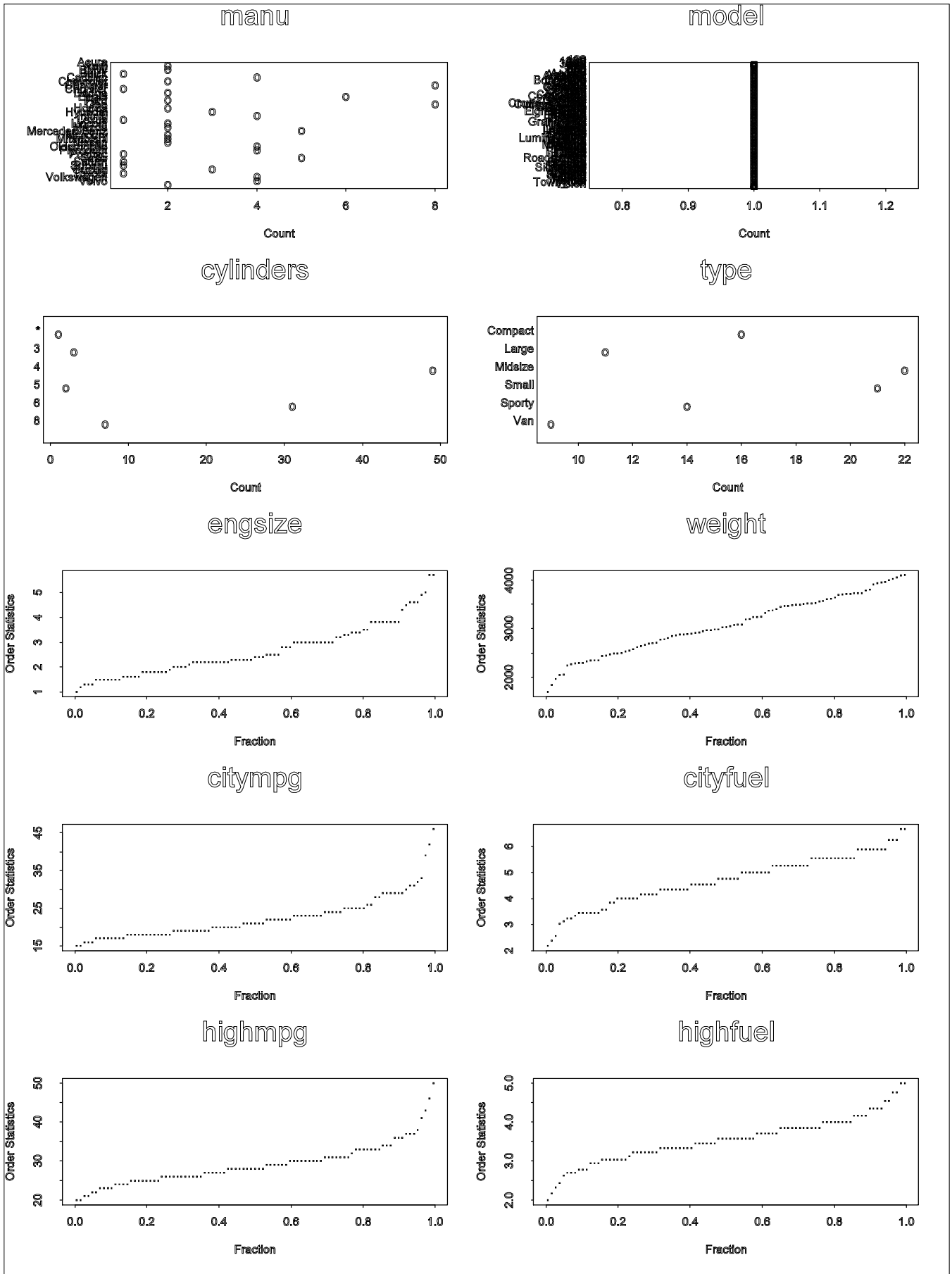
```

## A summary plot is also easy

```

> postscript("city.plot.ps",horiz=F)      ## put plot in file
> par(mfrow=c(5,2))                      ## 5 rows, 2 columns
> plot(cars.df,ask=F)                    ## generate summary plot for each variable
> dev.off()                              ## close graphics device

```



For example, lets look at the relationship between Citympg and Weight.

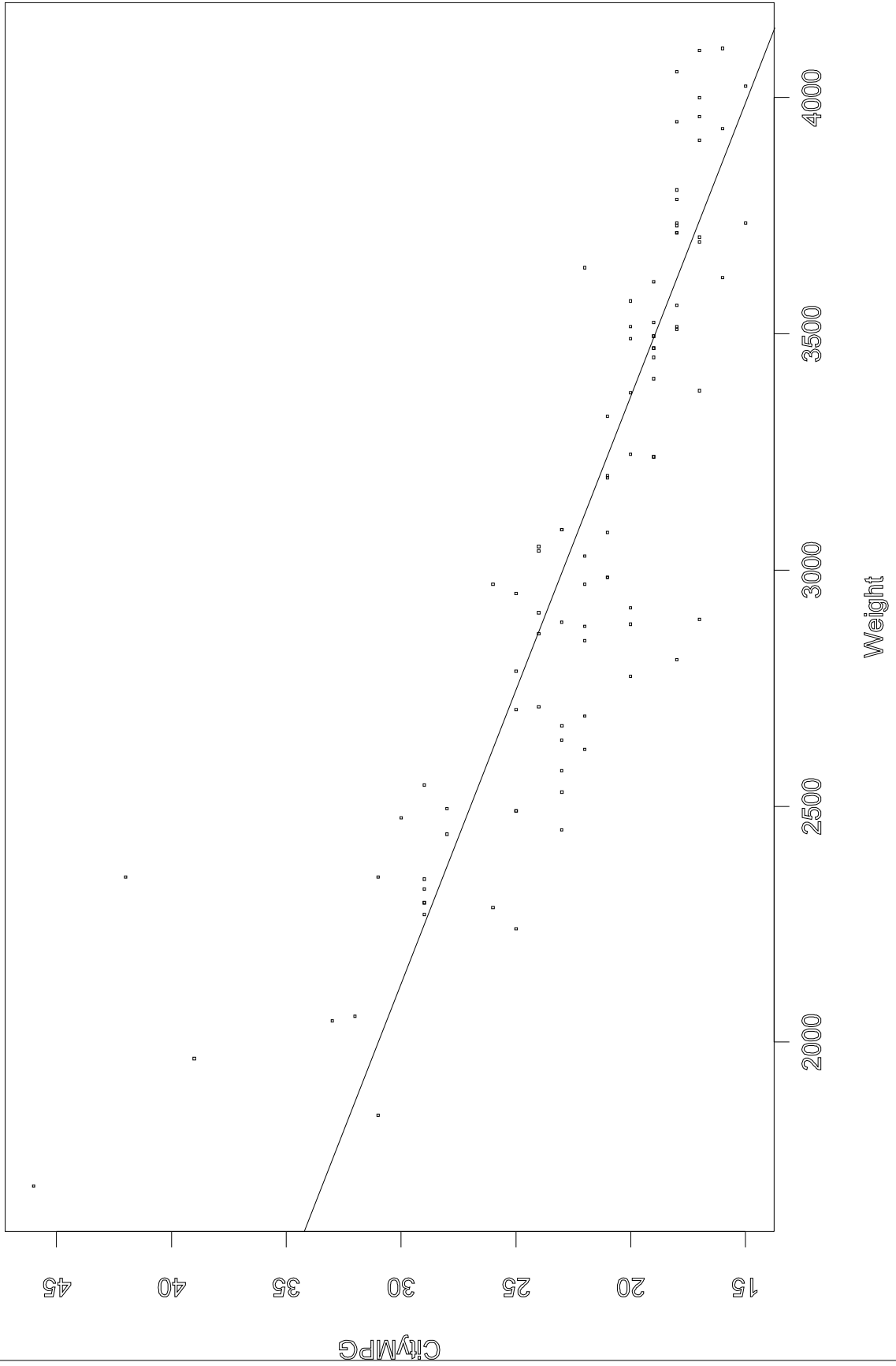
To plot the data and add the least squares fit use the following commands:

```
> postscript("citympg.ps",horiz=T)
> plot(cars.df$weight,cars.df$citympg,
+      xlab="Weight",ylab="CityMPG",
+      main="City MPG versus Weight")
> abline(lsfite(cars.df$weight,cars.df$citympg))
> dev.off()
```

Note that you can also create plots on the screen.

To do this you must be at a workstation or have Xwindows running on the Mac or PC. Instead of the `postscript()` device, use the `motif()` device. Also, you don't need to close the motif device after each plot. It will automatically clear the window for each new plot.

# City MPG versus Weight



```

> citympg.lm <- lm(citympg ~ weight, data = cars.df)

> citympg.lm
Call:
lm(formula = citympg ~ weight, data = cars.df)

Coefficients:
(Intercept)      weight
  47.04835  -0.008032392

Degrees of freedom: 93 total; 91 residual
Residual standard error: 3.03831

> summary(citympg.lm)

Call: lm(formula = citympg ~ weight, data = cars.df)
Residuals:
    Min     1Q   Median     3Q    Max
-6.795 -1.971  0.02486  1.186 13.83

Coefficients:
                Value Std. Error  t value Pr(>|t|)
(Intercept)  47.0484    1.6799   28.0064  0.0000
      weight  -0.0080    0.0005  -14.9583  0.0000

Residual standard error: 3.038 on 91 degrees of freedom
Multiple R-Squared:  0.7109
F-statistic: 223.8 on 1 and 91 degrees of freedom, the p-
value is 0

```

(Output from summary continued)

Correlation of Coefficients:

(Intercept)

weight -0.9823

```
> anova(citympg.lm)
```

Analysis of Variance Table

Response: citympg

Terms added sequentially (first to last)

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
weight	1	2065.519	2065.519	223.751	0
Residuals	91	840.051	9.231		

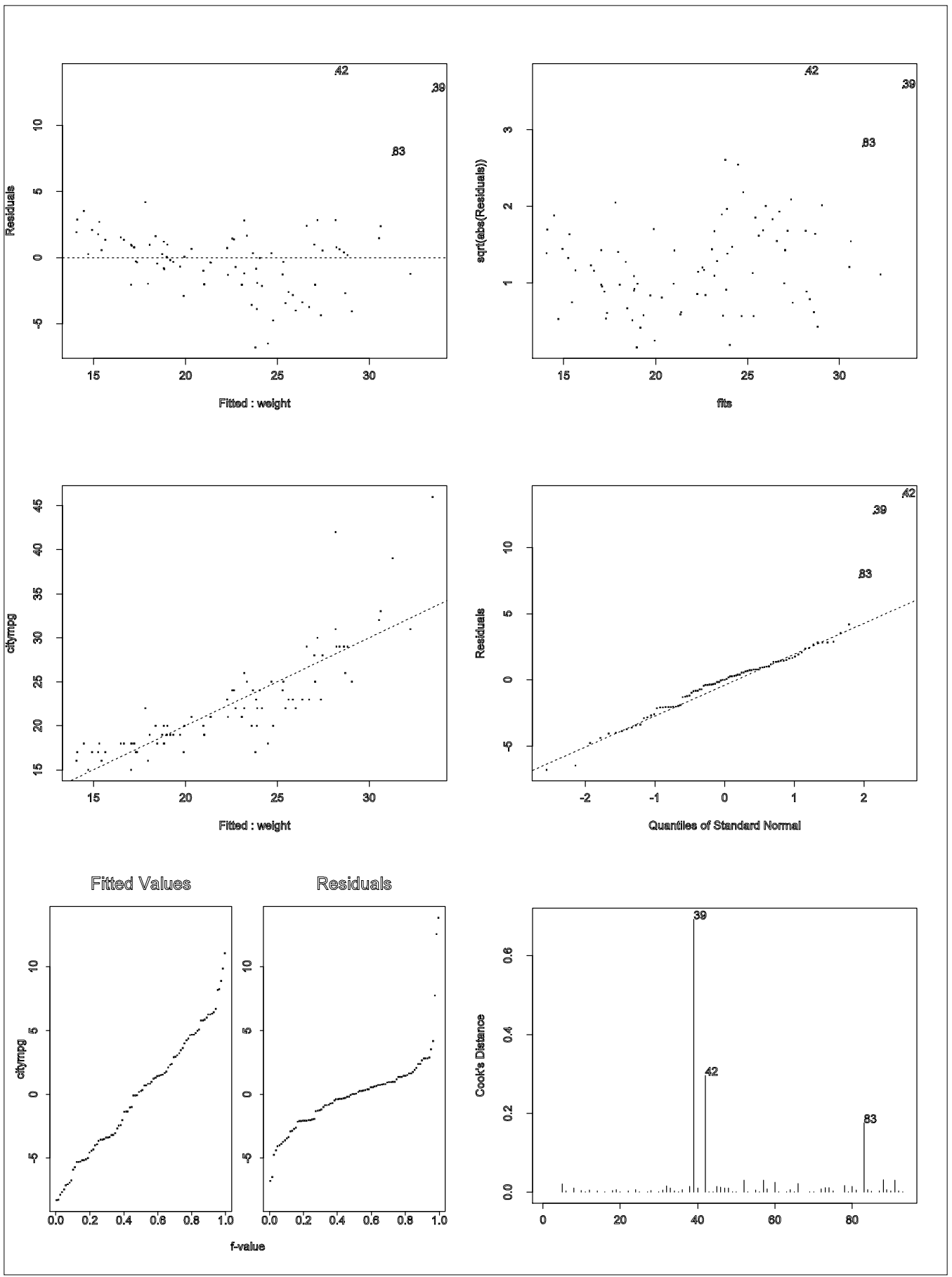
## Diagnostic Plots can be generated very easily

```
> postscript("citympgplot.ps",horiz=F)
```

```
> par(mfrow=c(3,2))
```

```
> plot(citympg.lm,prompt=F)
```

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



```

> cars.df[c(39,42,83),]
      manu model cylinders  type engsize weight
39   Geo Metro          3 Small    1.0   1695
42  Honda Civic          4 Small    1.5   2350
83 Suzuki Swift          3 Small    1.3   1965

      citympg cityfuel highmpg highfuel
39         46 2.173913        50 2.000000
42         42 2.380952        46 2.173913
83         39 2.564103        43 2.325581

```

Note that this model doesn't fit particularly well. Instead lets us a new response variable

$$\text{Cityfuel} = \frac{100}{\text{Citympg}}$$

It can be created with the command

```

> cars.df$cityfuel <- 100/cars.df$citympg

```

To get the plot of Cityfuel versus Weight

```

> postscript("cityfuel.ps",horiz=T)

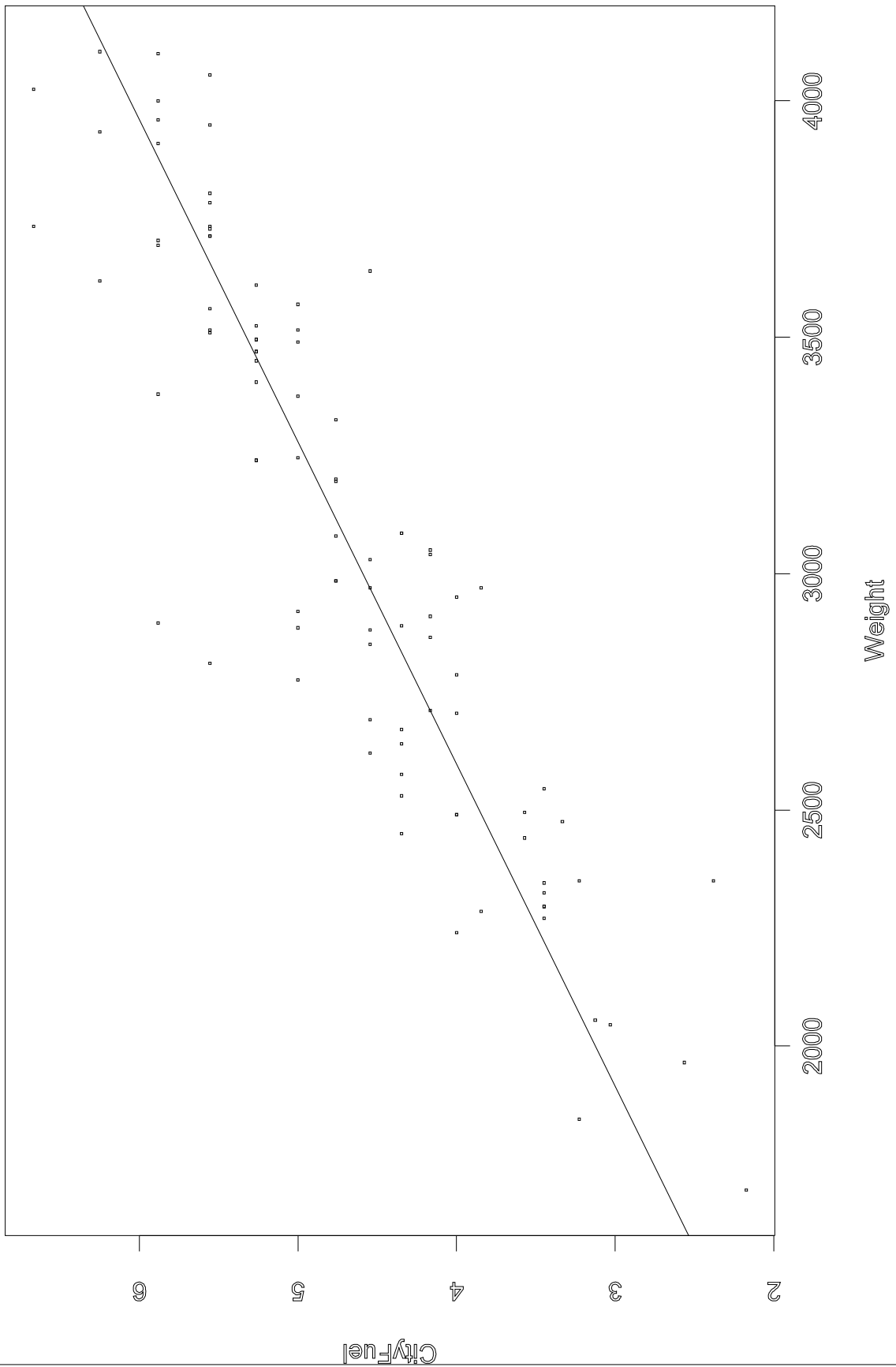
> plot(cars.df$weight,cars.df$cityfuel,xlab="Weight "
+ , ylab="CityFuel",main="City Fuel versus Weight")

> abline(lsfite(cars.df$weight,cars.df$cityfuel))

> dev.off()

```

# City Fuel versus Weight



```
> cityfuel.lm<-lm(cityfuel~weight,data=cars.df)
```

```
> cityfuel.lm
```

```
Call:
```

```
lm(formula = cityfuel ~ weight, data = cars.df)
```

```
Coefficients:
```

```
(Intercept)      weight  
  0.1936667  0.001466229
```

```
Degrees of freedom: 93 total; 91 residual
```

```
Residual standard error: 0.4288011
```

```
> summary(cityfuel.lm)
```

```
Call: lm(formula = cityfuel ~ weight, data = cars.df)
```

```
Residuals:
```

```
      Min       1Q   Median       3Q      Max  
-1.258 -0.2277 -0.08177  0.2393  1.444
```

```
Coefficients:
```

```
              Value Std. Error t value Pr(>|t|)  
(Intercept)  0.1937   0.2371    0.8169  0.4161  
      weight  0.0015   0.0001   19.3471  0.0000
```

```
Residual standard error: 0.4288 on 91 degrees of freedom
```

```
Multiple R-Squared: 0.8044
```

```
F-statistic: 374.3 on 1 and 91 degrees of freedom, the p-  
value is 0
```

(Output from summary continued)

Correlation of Coefficients:

(Intercept)

weight -0.9823

```
> anova(cityfuel.lm)
```

Analysis of Variance Table

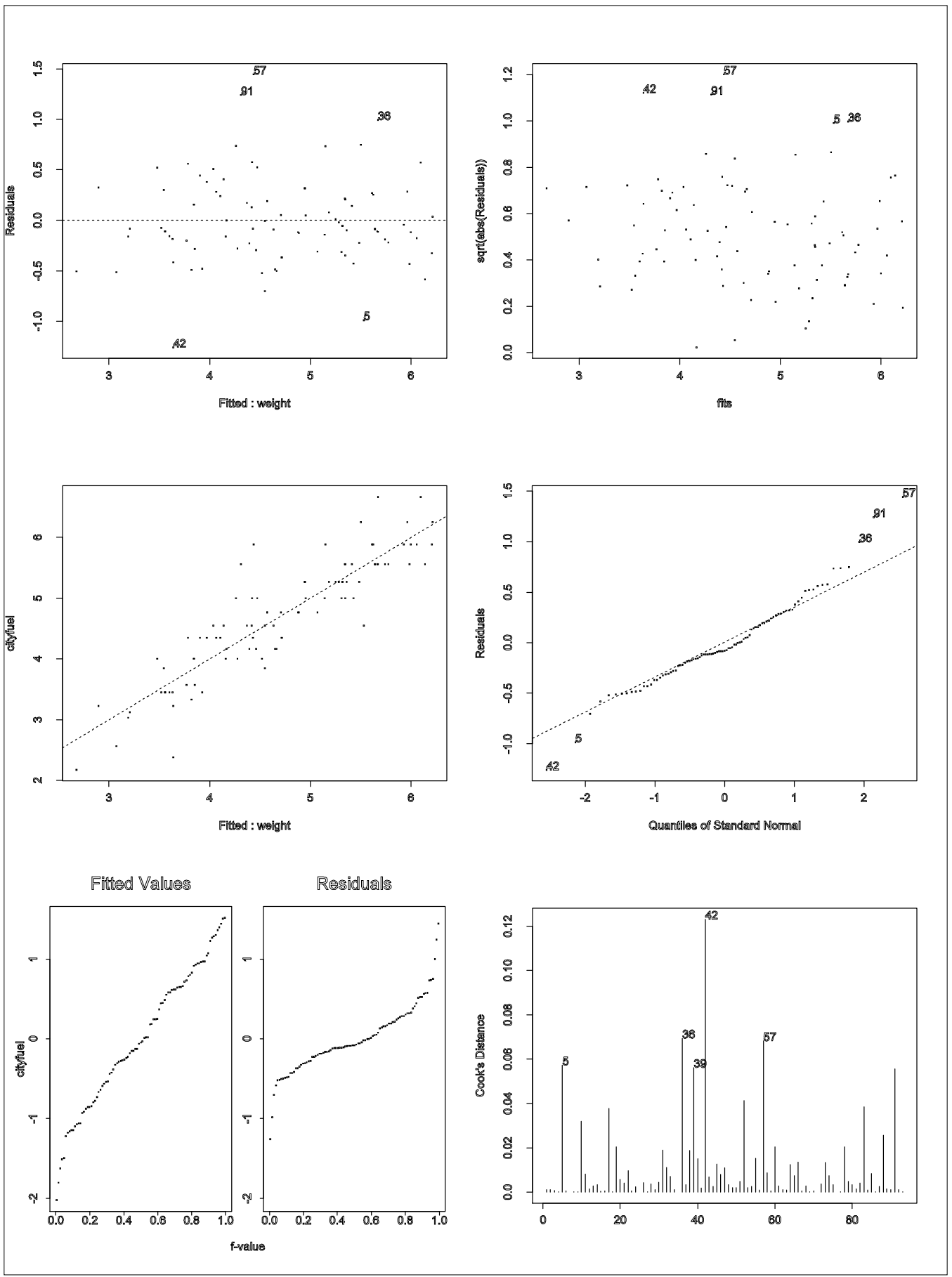
Response: cityfuel

Terms added sequentially (first to last)

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
weight	1	68.82453	68.82453	374.31	0
Residuals	91	16.73221	0.18387		

Again, diagnostic plots are easy to get. This time, I'm having the 5 most extreme points flagged, instead of the default 3.

```
> postscript("cityfuelplot.ps",horiz=F)
> par(mfrow=c(3,2))
> plot(cityfuel.lm,prompt=F,id.n=5)
> dev.off()
```



A total of 6 cars get flagged in the diagnostic plots.  
They happen to be

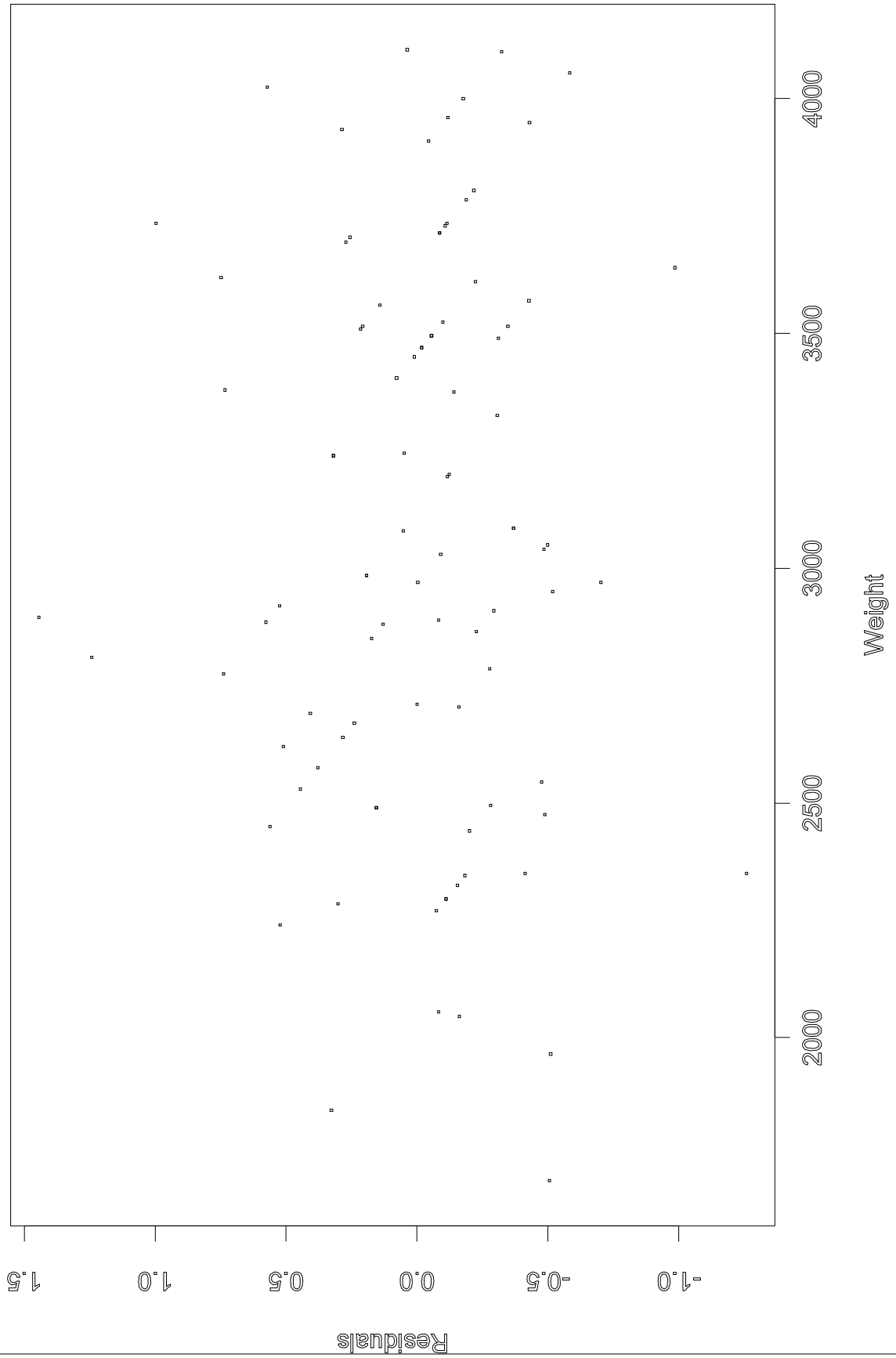
```
> cars.df[c(5,36,39,42,57,91),1:6]
      manu      model cylinders      type engsize  weight
5      BMW      535i         4 Midsize    3.5    3640
36     Ford Aerostar         6      Van    3.0    3735
39      Geo      Metro         3   Small    1.0    1695
42     Honda    Civic         4   Small    1.5    2350
57     Mazda    RX-7          * Sporty    1.3    2895
91 Volkswagen Corrado         6 Sporty    2.8    2810

      citympg cityfuel highmpg highfuel
5         22  4.545455         30  3.333333
36         15  6.666667         20  5.000000
39         46  2.173913         50  2.000000
42         42  2.380952         46  2.173913
57         17  5.882353         25  4.000000
91         18  5.555556         25  4.000000
```

Note that it is easy to access different values from the results of the regression. For example, it is easy to get the residuals as seen in the following plot commands

```
> postscript("cityfuelresid.ps",horiz=T)
> plot(cars.df$weight,resid(cityfuel.lm),
+      xlab="Weight" ,ylab="Residuals",
+      main="Residual Plot - City Fuel versus Weight")
```

Residual Plot - City Fuel versus Weight



# Getting help in S-Plus

## 2 forms

`help(command)` or `?command`

They give the same output.

`help(mean)`

Mean Value (Arithmetic Average)

### DESCRIPTION:

Returns a number which is the mean of the data. A fraction to be trimmed from each end of the ordered data can be specified.

### USAGE:

```
mean(x, trim = 0, na.rm = F)
```

### REQUIRED ARGUMENTS:

`x`  
numeric object. Missing values (NA) are allowed.

### OPTIONAL ARGUMENTS:

`trim`  
fraction (between 0 and .5, inclusive) of values to be trimmed from each end of the ordered data. If `trim=.5`, the result is the median.

`na.rm`  
logical flag: should missing values be removed before computation?

### VALUE:

(trimmed) mean of `x`.

### DETAILS:

If `x` contains any NAs, the result will be NA unless `na.rm=TRUE`. If `x` is a factor, a warning will be generated and the result will be NA.

When `trim` is positive, approximately `trim*length(x)` largest values and `trim*length(x)` smallest values are ignored; the mean of the

remaining values is returned. When `trim=.25`, the result is often called the "midmean".

#### REFERENCES:

Hoaglin, D. C., Mosteller, F. and Tukey, J. W., editors (1983). *Understanding Robust and Exploratory Data Analysis*. Wiley, New York.

#### SEE ALSO:

`apply` , `colMeans` , `location.m` , `median` , `stdev` , `var` .

#### EXAMPLES:

```
algebra <- testscores[,3] # vector of 25 algebra testscores
mean(algebra) # Computes the average
mean(algebra, trim = .1) # Computes the 10% trimmed mean of scores
apply(testscores, 2, mean) # vector of the means of the columns of
scores
```

There is also a graphical help system in S-Plus. It can be accessed with the command `help.start()`. Note that this is only accessible when X windows is active, either by logging onto a workstation directly, or with a X windows emulator from a Mac (eXodus) or a PC (Exceed or X-Win32).

I find the text based system much faster and it has many of the features of the graphical system since it is based on html and uses lynx, a text based web browser.

To check the arguments for a function, you may use the `args` command

```
> args(mean)
function(x, trim = 0, na.rm = F)
NULL
```

To examine a function, just type its name without parentheses

```
> mean
function(x, trim = 0, na.rm = F)
.Call("S_c_use_method", "mean", COPY =
NULL, CLASSES = NULL)
```

```
> log      (calls a c routine)
function(x)
.Call("S_c_use_method", "log")
```

```
> log10    (uses the log function)
# common logarithms
function(x)
log(x)/.Internal(log(10), "do_math", T, 106)
```

```
> stdev    (uses the colVars function)
function(x, na.rm = F)
sqrt(colVars(c(x), na.rm = na.rm))
```