Essex Summer School in Social Science Data Analysis
Panel Data Analysis for Comparative Research

LAB: Introduction to R

Christopher Adolph

Department of Political Science
and
Center for Statistics and the Social Sciences
University of Washington, Seattle

Why R?

Real question: Why programming?

Non-programmers stuck with package defaults

For your substantive problem, defaults may be

- inappropriate (not quite the right model, but "close")
- unintelligible (reams of non-linear coefficients and stars)

Programming allows you to match the methods to the data & question

Get better, more easily explained results.

Why R?

Many side benefits:

1. Never forget what you did: The code can be re-run.
2. Repeating an analysis n times? Write a loop!
3. Programming makes data processing/reshaping easy.
4. Programming makes replication easy.

Why R?

R is

- free
- open source
- growing fast
- widely used
- the future for most fields

But once you learn one language, the others are much easier

Introduction to R

R is a calculator that can store lots of information in memory

R stores information as "objects"

```
> x <- 2
> print(x)
[1] 2

> y <- "hello"
> print(y)
[1] "hello"

> z <- c(15, -3, 8.2)
> print(z)
[1] 15.0 -3.0  8.2
```

Introduction to R

```
> w <- c("gdp", "pop", "income")
> print(w)
[1] "gdp"    "pop"    "income"
>
```

Note the assignment operator, <-, not =

An object in memory can be called to make new objects

```
> a <- x^2
> print(x)
[1] 2
> print(a)
[1] 4

> b <- z + 10
> print(z)
[1] 15.0 -3.0  8.2
> print(b)
[1] 25.0  7.0 18.2
```

Introduction to R

```
> c <- c(w,y)
> print(w)
[1] "gdp"    "pop"    "income"
> print(y)
[1] "hello"
> print(c)
[1] "gdp"    "pop"    "income" "hello"
```

Commands (or “functions”) in R are always written `command()`

The usual way to use a command is:

```
output <- command(input)
```

We’ve already seen that `c()` pastes together variables.

A simple example:

```
> z <- c(15, -3, 8.2)
> mz <- mean(z)
> print(mz)
[1] 6.733333
```

Introduction to R

Some commands have multiple inputs. Separate them by commas:

`plot(var1,var2)` plots var1 against var2

Some commands have optional inputs. If omitted, they have default values.

`plot(var1)` plots var1 against the sequence $\{1,2,3,\ldots\}$

Inputs can be identified by their position or by name.

`plot(x=var1,y=var2)` plots var2 against var1

Entering code

You can enter code by typing at the prompt, by cutting or pasting, or from a file

If you haven't closed the parenthesis, and hit enter, `R` let's you continue with this prompt +

You can copy and paste multiple commands at once

You can run a text file containing a program using `source()`, with the name of the file as input (ie, in "")

I prefer the `source()` approach. Leads to good habits of retaining code.

Data types

R has three important data types to learn now

Numeric	`y <- 4.3`
Character	`y <- "hello"`
Logical	`y <- TRUE`

We can always check a variable's type, and sometimes change it:

```
population <- c("1276", "562", "8903")
print(population)
is.numeric(population)
is.character(population)
```

Oops! The data have been read in as characters, or "strings". R does not know they are numbers.

```
population <- as.numeric(population)
```

Some special values

Missing data	`NA`
A "blank"	`NULL`
Infinity	`Inf`
Not a number	`NaN`

Data structures

All R objects have a data type *and* a data structure

Data structures can contain numeric, character, or logical entries

Important structures:

Vector

Matrix

Dataframe

List (to be covered later)

Vectors in R

Vector is R are simply 1-dimensional lists of numbers or strings

Let's make a vector of random numbers:

```
x <- rnorm(1000)
```

x contains 1000 random normal variates drawn from a Normal distribution with mean 0 and standard deviation 1.

What if we wanted the mean of this vector?

```
mean(x)
```

What if we wanted the standard deviation?

```
sd(x)
```

Vectors in R

What if we wanted just the first element?

```
x[1]
```

or the 10th through 20th elements?

```
x[10:20]
```

what if we wanted the 10th percentile?

```
sort(x)[100]
```

Indexing a vector can be very powerful. Can apply to any vector object.

What if we want a histogram?

```
hist(x)
```

Vectors in R

Useful commands for vectors:

`seq(from, to, by)`	generates a sequence
`rep(x,times)`	repeats x
`sort()`	sorts a vector from least to greatest
`rev()`	reverses the order of a vector
`rev(sort())`	sorts a vector from greatest to least

Matrices in R

Vector are the standard way to store and manipulate variables in R

But usually our datasets have several variables measured on the same observations

Several variables collected together form a matrix with one row for each observation and one column for each variable

Matrices in R

Many ways to make a matrix in R

```
a <- matrix(data=NA, nrow, ncol, byrow=FALSE)
```

This makes a matrix of `nrow` $\times$ `ncol`, and fills it with missing values.

To fill it with data, substitute a vector of data for NA in the command. It will fill up the matrix column by column.

We could also paste together vectors, binding them by column or by row:

```
b <- cbind(var1, var2, var3)
c <- rbind(obs1, obs2)
```

Matrices in R

Optionally, R can remember names of the rows and columns of a matrix

To assign names, use the commands:

```
colnames(a) <- c("Var1", "Var2")
rownames(a) <- c("Case1", "Case2")
```

Substituting the actual names of your variables and observations (and making sure there is one name for each variable & observation)

Matrices in R

Matrices are indexed by row and column.

We can subset matrices into vectors or smaller matrices

`a[1,1]`	Gets the first element of a
`a[1:10,1]`	Gets the first ten rows of the first column
`a[,5]`	Gets every row of the fifth column
`a[4:6,]`	Gets every column of the 4th through 6th rows

To make a vector into a matrix, use `as.matrix()`

R defaults to treating one-dimensional arrays as vectors, not matrices

Useful matrix commands:

`nrow()`	Gives the number of rows of the matrix
`ncol()`	Gives the number of columns
`t()`	Transposes the matrix

Much more on matrices next week.

Dataframes in R

Dataframes are a special kind of matrix used to store datasets

To turn a matrix into a dataframe (note the extra .):

```
a <- as.data.frame(a)
```

Dataframes always have columns names, and these are set or retrieved using the `names()` command

```
names(a) <- c("Var1","Var2")
```

Dataframes can be “attached”, which makes each column into a vector with the appropriate name

```
attach(a)
```

Loading data

There are many ways to load data to R. I prefer using comma-separated variable files, which can be loaded with `read.csv`

You can also check the `foreign` library for other data file types

If your data have variable names, you can attach the dataset like so:

```
data <- read.csv("mydata.csv")
attach(data)
```

to access the variables directly

Benefits and dangers of `attach()`

If your data have variable names, you can also “attach” the dataset like so:

```
data <- read.csv("mydata.csv")
attach(data)
```

to access all the variables directly through newly created vectors.

Be careful! `attach()` is tricky.

1. If you attach a variable `data$x` in `data` and then modify `x`, the original `data$x` is unchanged.

2. If you have more than one dataset with the same variable names, `attach()` is a bad idea: only the first will be attached!

Sometimes `attach()` is handy, but be careful!

Missing data

When loading a dataset, you can often tell R what symbol that file uses for missing data using the option `na.strings=`

So if your dataset codes missings as ., set `na.strings="."`

If your dataset codes missings as a blank, set `na.strings=""`

If your dataset codes missings in multiple ways, you could set, e.g., `na.strings=c(".","","NA")`

Missing data

Many R commands will not work properly on vectors, matrices, or dataframes containing missing data (NAs)

To check if a variables contains missings, use `is.na(x)`

To create a new variable with missings listwise deleted, use `na.omit`

If we have a dataset `data` with NAs at data[15,5] and data[17,3]

```
dataomitted <- na.omit(data)
```

will create a new dataset with the 15th and 17th rows left out

Be careful! If you have a variable with lots of NAs you are not using in your analysis, remove it from the dataset *before* using `na.omit()`

Mathematical Operations

R can do all the basic math you need

Binary operators:

`+  -  *  /  ^`

Binary comparisions:

`<  <=  >  >=  ==  !=`

Logical operators (and, or, not, control-flow and, control-flow not; use parentheses!):

`&  |  !  &&  ||`

Math/stat fns:

`log  exp  mean  median  min  max  sd  var  cov  cor`

Set functions (see `help(sets)`), Trigonometry (see `help(Trig)`),

R follows the usual order of operations; if it doubt, use parentheses

Example 1: US Economic growth

Let's investigate an old question in political economy:

Are there partisan cycles, or tendencies, in economic performance?

Does one party tend to produce higher growth on average?

(Theory: Left cares more about growth vis-a-vis inflation than the Right

If there is partisan control of the economy,
then Left should have higher growth ceteris paribus)

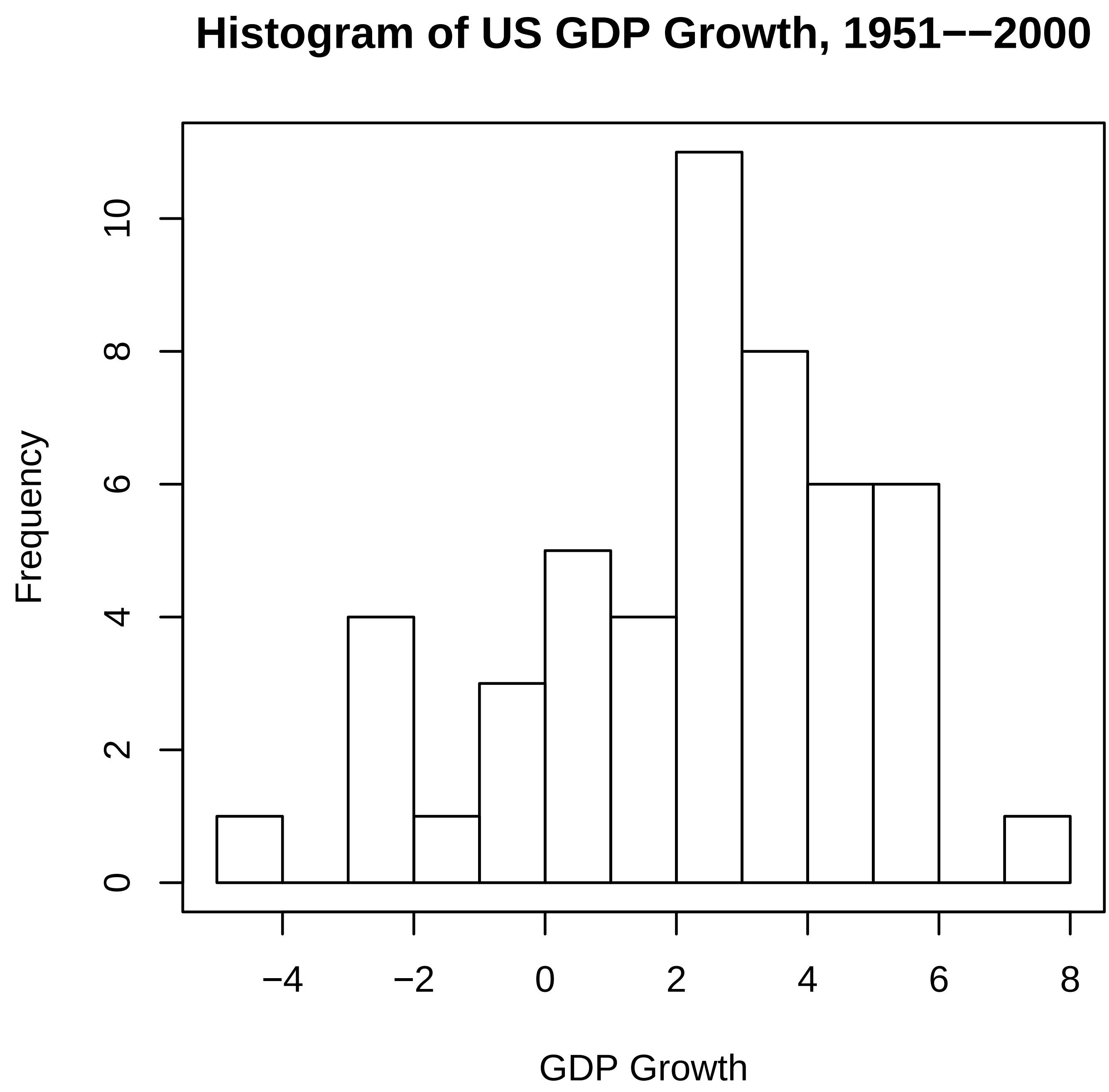
Data from the Penn World Tables (Annual growth rate of GDP in percent)

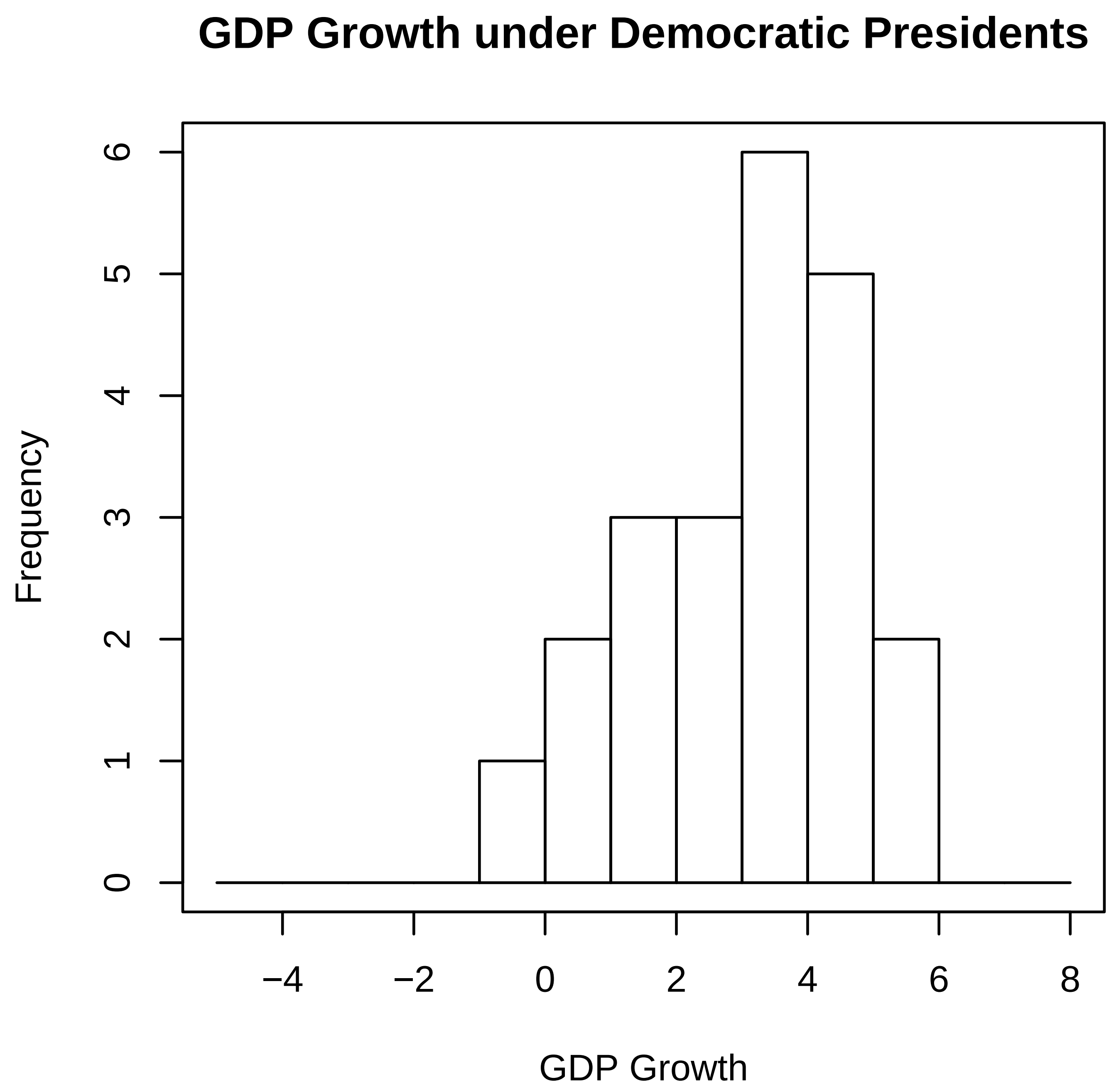
Two variables:

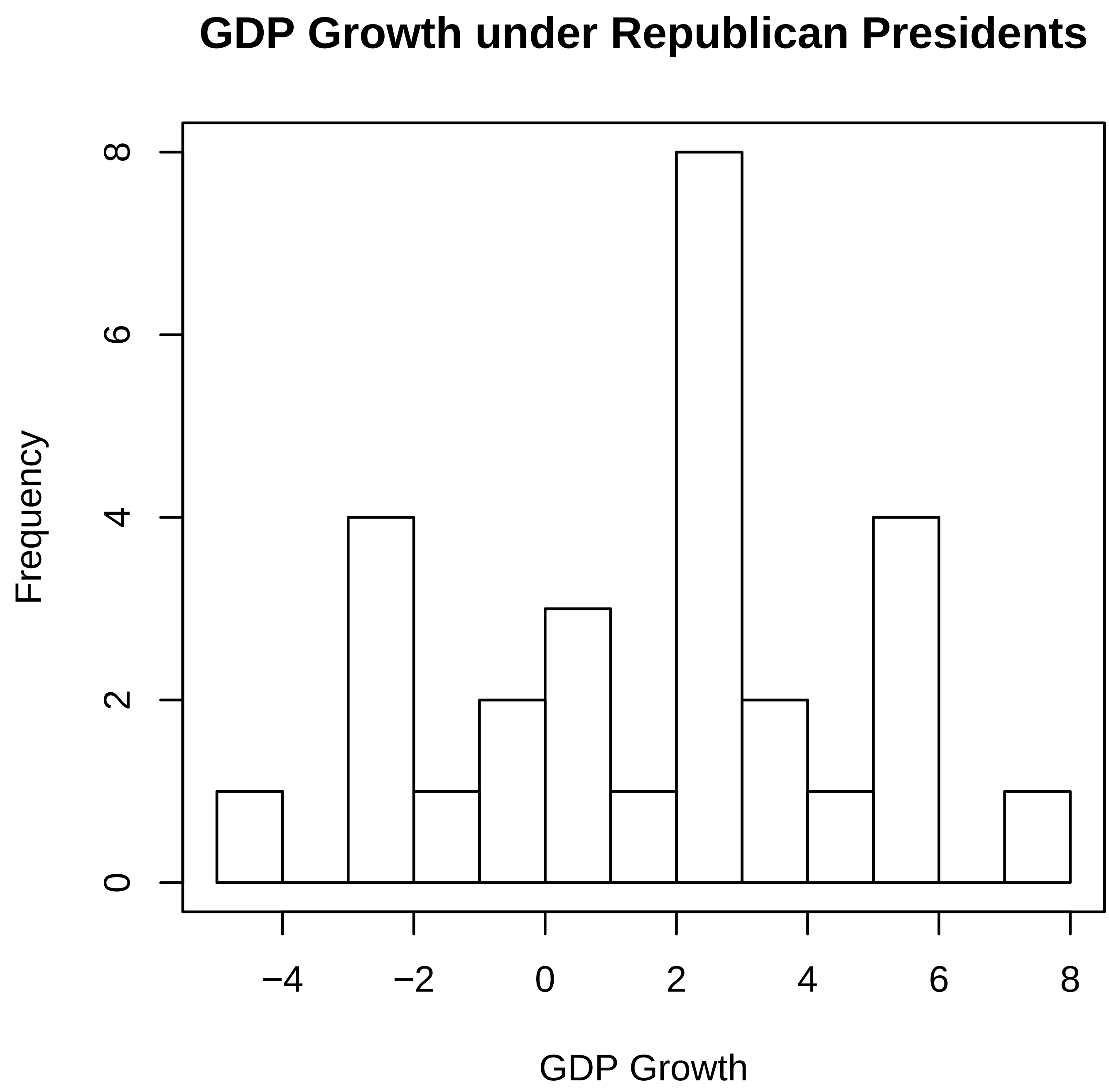
`grgdpch`	The per capita GDP growth rate
`party`	The party of the president (Dem = -1, Rep = 1)

Example 1: US Economic growth

```
# Load data
data <- read.csv("gdp.csv", na.strings="")
attach(data)

# Construct party specific variables
gdp.dem <- grgdpch[party==-1]
gdp.rep <- grgdpch[party==1]

# Make the histogram
hist(grgdpch,
     breaks=seq(-5,8,1),
     main="Histogram of US GDP Growth, 1951--2000",
     xlab="GDP Growth")
```


Histogram of US GDP Growth, 1951−−2000
Frequency
0
2
4
6
8
10
−4
−2
0
2
4
6
8
GDP Growth


GDP Growth under Democratic Presidents
Frequency
0
1
2
3
4
5
6
−4
−2
0
2
4
6
8
GDP Growth


GDP Growth under Republican Presidents
Frequency
0
2
4
6
8
−4
−2
0
2
4
6
8
GDP Growth

```
# Make a box plot
boxplot(grgdpch~as.factor(party),
        boxwex=0.3,
        range=0.5,
        names=c("Democratic\n Presidents",
          "Republican\n Presidents"),
        ylab="GDP growth",
        main="Economic performance of partisan governments")
```

Note the unusual first input: this is an R formula

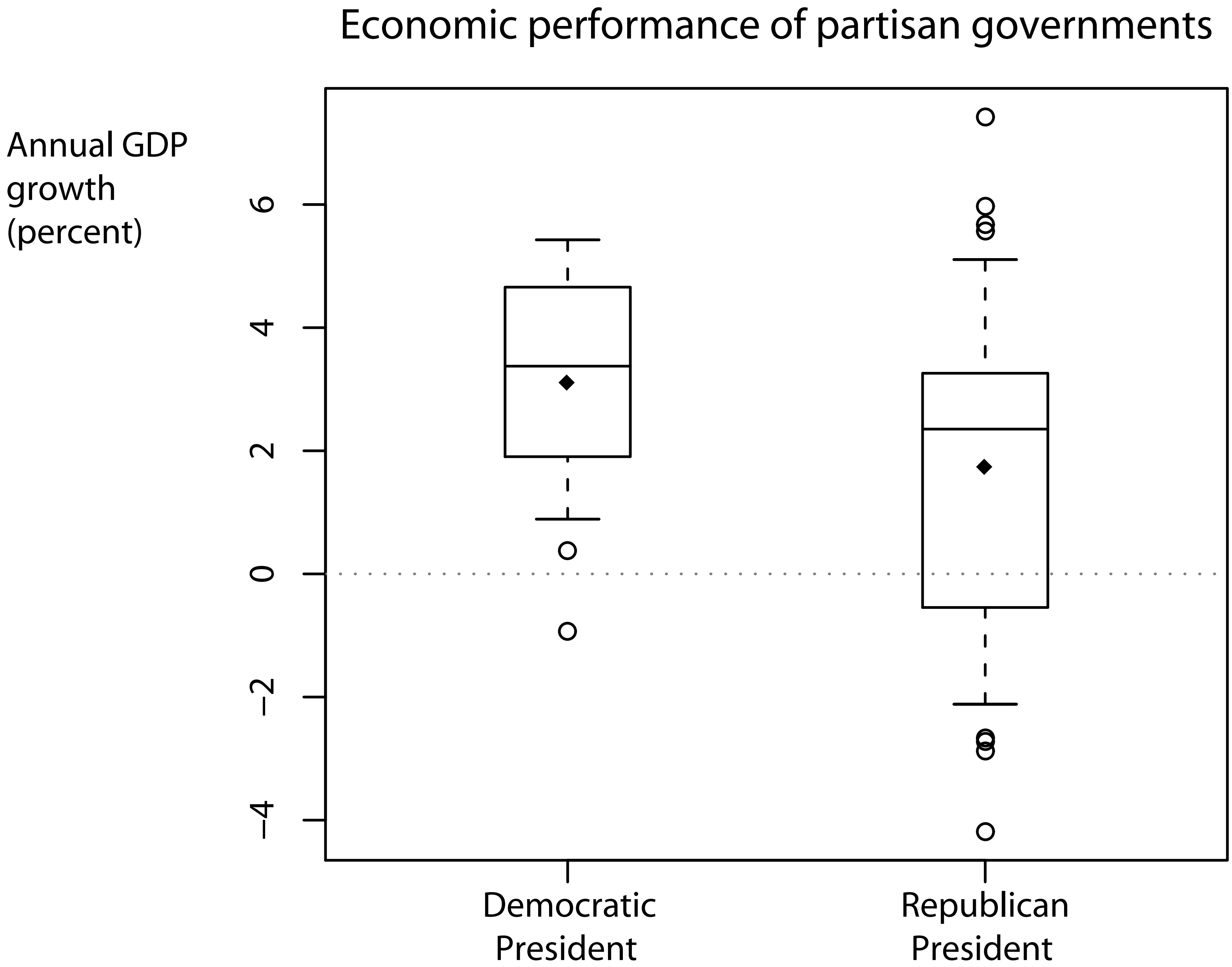
`y~x1+x2+x3`

In this case, `grgdpch` is being "modelled" as a function of `party`

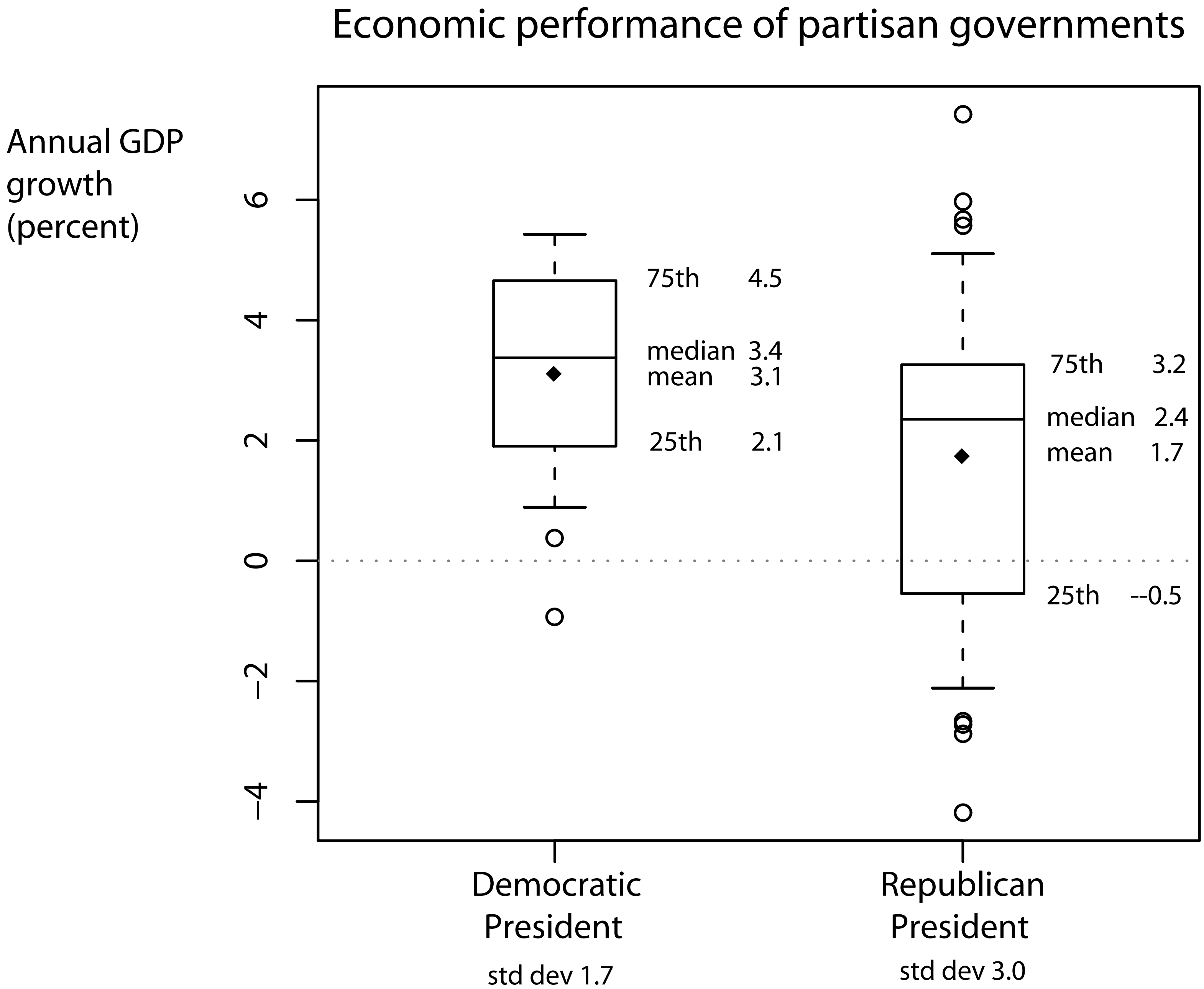
`boxplot()` needs party to be a "factor" or an explicitly categorical variable

Hence we pass boxplot `as.factor(party)`, which turns the numeric variable into a factor

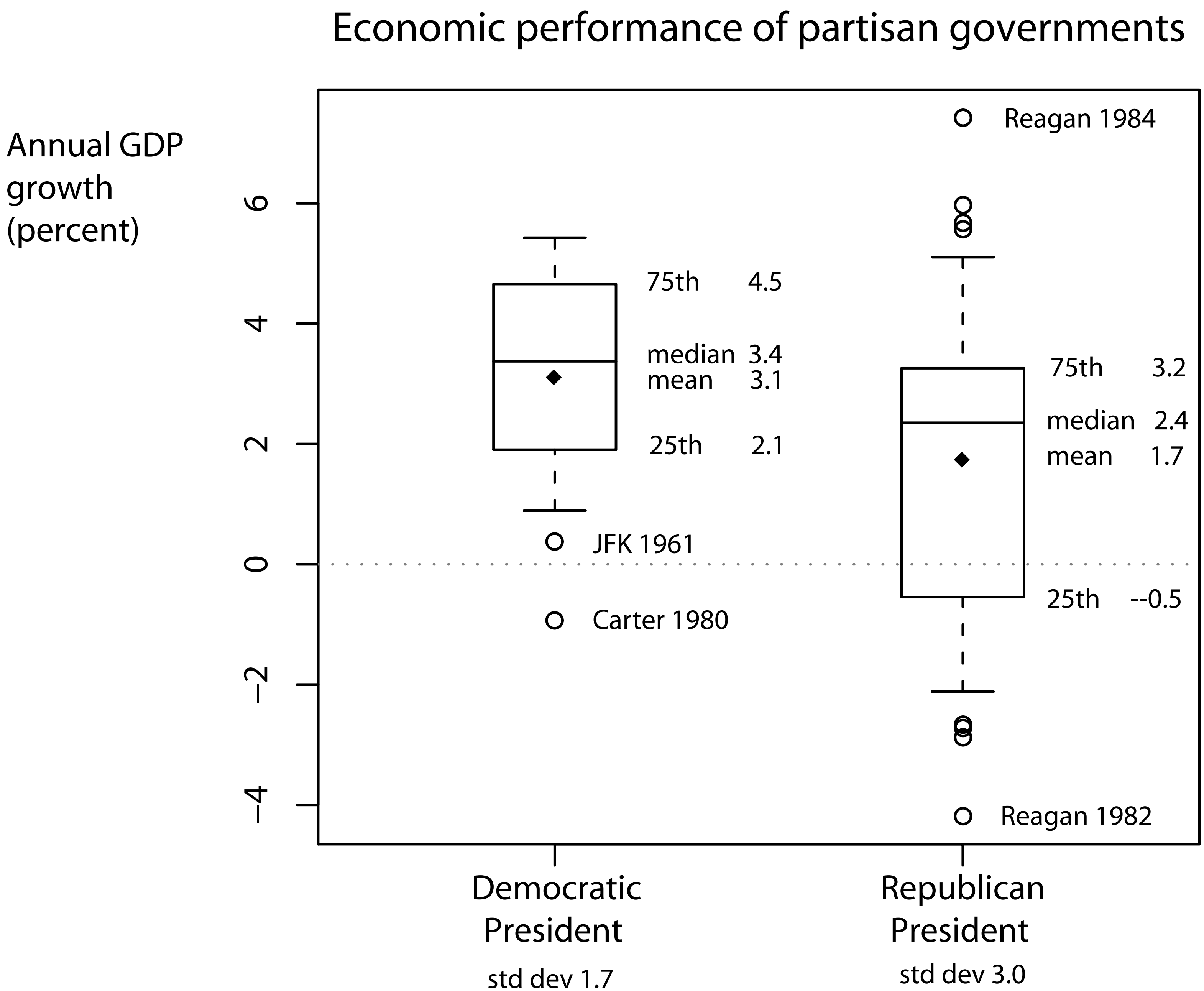
Box plots: Annual US GDP growth, 1951–2000



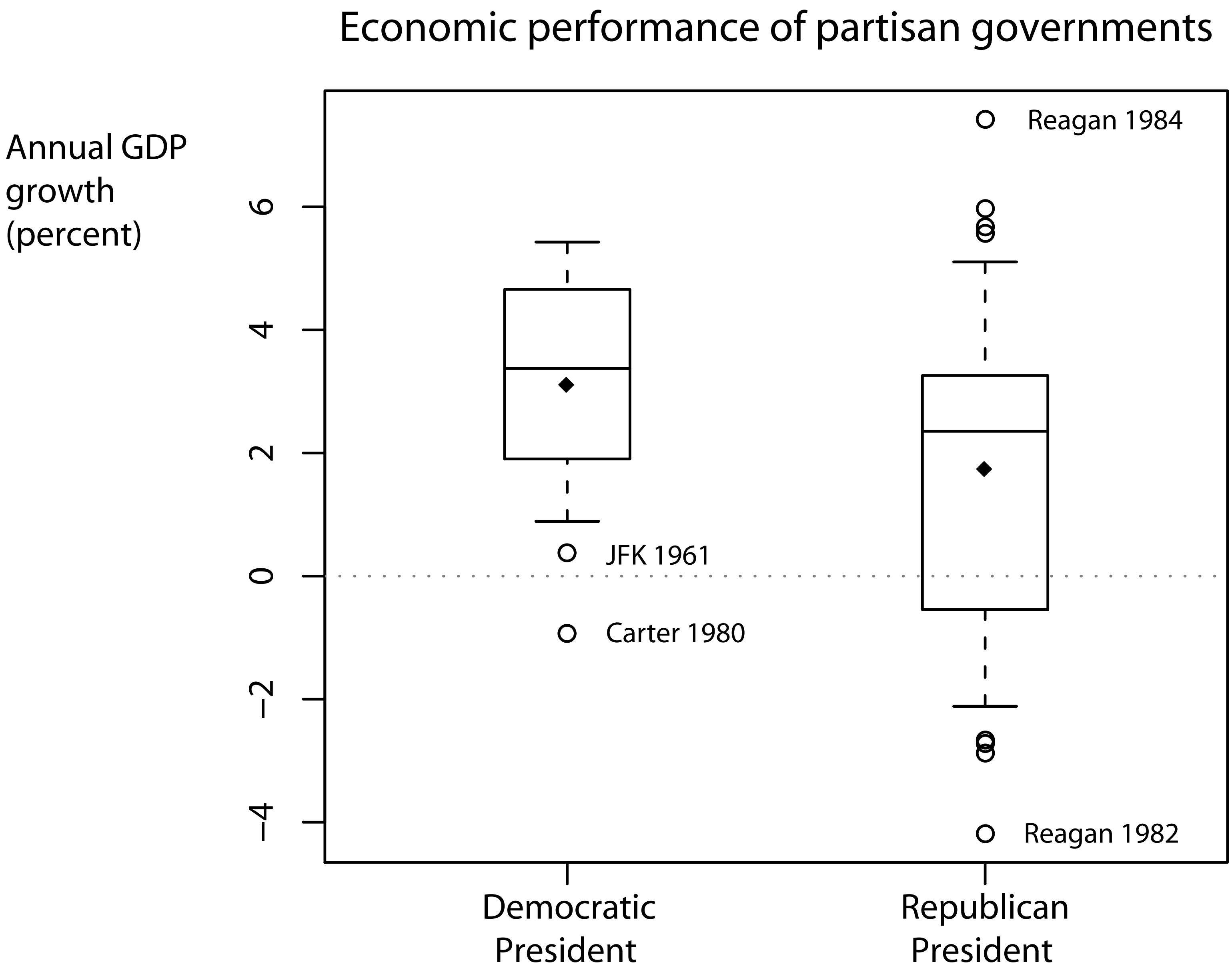
Box plots: Annual US GDP growth, 1951–2000



Box plots: Annual US GDP growth, 1951–2000



Box plots: Annual US GDP growth, 1951–2000



Help!

To get help on a known command `x`, type `help(x)` or `?x`

To search the help files using a keyword string `s`, type `help.search(s)`

Note that this implies to search on the word regression, you should type `help.search("regression")`

but to get help for the command `lm`, you should type `help(lm)`

Installing R on a PC

- Go to the Comprehensive R Archive Network (CRAN)
  `http://cran.r-project.org/`

- Under the heading "Download and Install R", click on "Windows"

- Click on "base"

- Download and run the R setup program.
  The name changes as R gets updated;
  the current version is "R-3.4.0-win.exe"

- Once you have R running on your computer,
  you can add new libraries from inside R by selecting
  "Install packages" from the Packages menu

Installing R on a Mac

- Go to the Comprehensive R Archive Network (CRAN)
  `http://cran.r-project.org/`
- Under the heading "Download and Install R", click on "MacOS X"
- Download and run the R setup program.
  The name changes as R gets updated;
  the current version is "R-3.4.0.pkg"
  (El Capitan or higher OS)
- Once you have R running on your computer,
  you can add new libraries from inside R by selecting
  "Install packages" from the Packages menu

Editing scripts

Don't use Microsoft Word to edit R code!

Word adds lots of "stuff" to text; R needs the script in a plain text file.

Some text editors:

- **Notepad:** Free, and comes with Windows (under Start → Programs → Accessories). Gets the job done; not powerful.
- **TextEdit:** Free, and comes with Mac OS X. Gets the job done; not powerful.
- **TINN-R:** Free and fairly powerful. Windows only. `http://www.sciviews.org/Tinn-R/`
- **Emacs:** Free and very powerful (my preference). Can use for R and Latex. Available for Mac and PC.

  For Mac (easy installation): `http://aquamacs.org/`

  For Windows (see the README): `http://ftp.gnu.org/gnu/emacs/windows/`

Editing data

R can load many other packages' data files

See the `foreign` library for commands

For simplicity & universality, I prefer Comma-Separated Variable (CSV) files

Microsoft Excel can edit and export CSV files (under Save As)

R can read them using `read.csv()`

OpenOffice is free alternative to Excel & makes CSV files (for all platforms):
`http://www.openoffice.org/`

My detailed guide to installing social science software on the Mac:
`http://thewastebook.com/?post=social-science-computing-for-mac`

Focus on steps 1.1 and 1.3 for now; come back later for Latex in step 1.2

Example 2: A simple linear regression

Let's investigate a bivariate relationship

Cross-national data on fertility (children born per adult female) and the percentage of women practicing contraception.

Data are from 50 developing countries.

Source: Robey, B., Shea, M. A., Rutstein, O. and Morris, L. (1992) "The reproductive revolution: New survey findings." *Population Reports.* Technical Report M-11.

Example 2: A simple linear regression

```
# Load data
data <- read.csv("robeymore.csv", na.strings="")
completedata <- na.omit(data)
attach(completedata)

# Transform variables
contraceptors <- contraceptors/100

# Run linear regression
res.lm <- lm(tfr~contraceptors)
print(summary(res.lm))

# Get predicted values
pred.lm <- predict(res.lm)
```

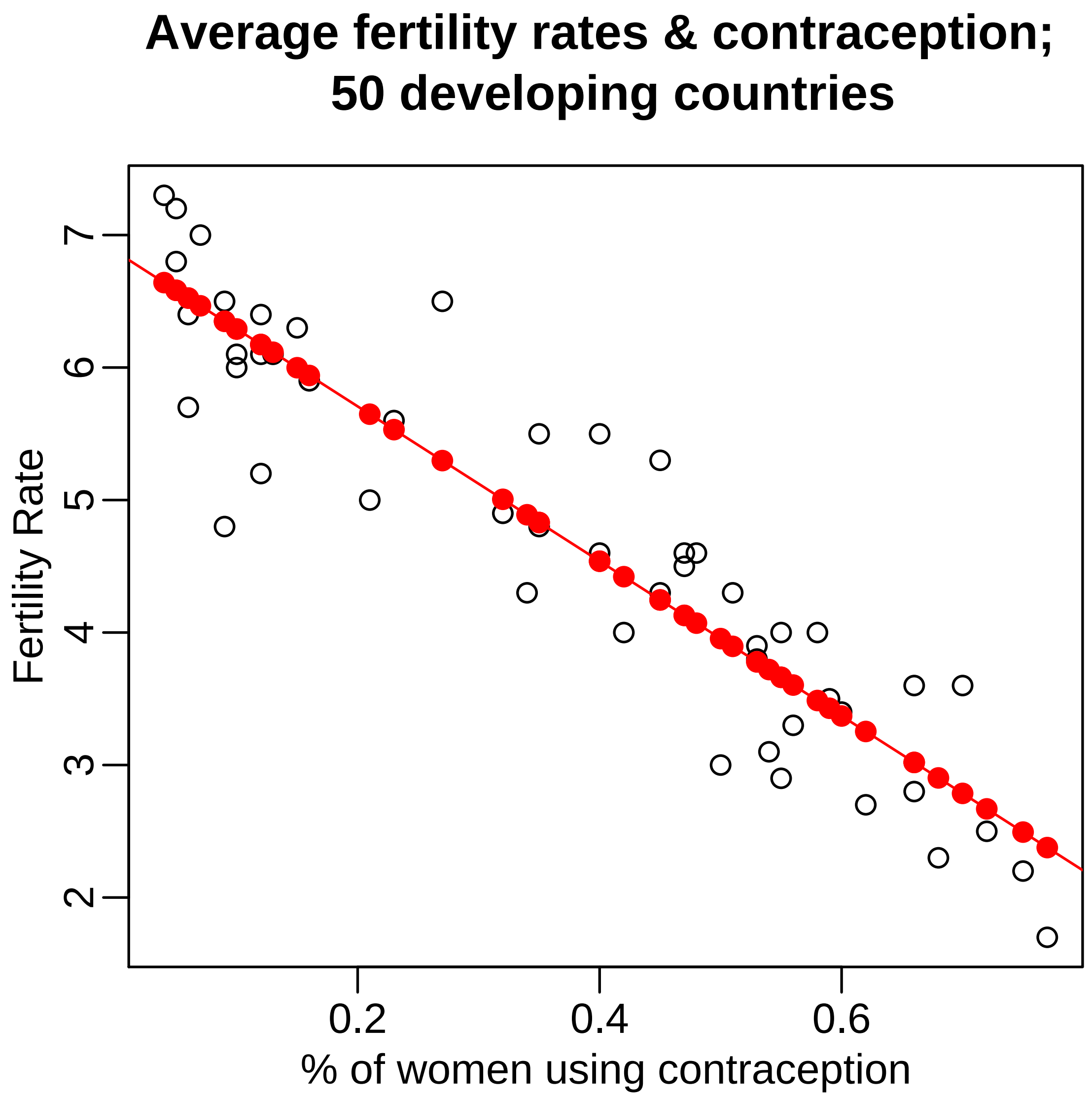
Example 2: A simple linear regression

```
# Make a plot of the data
plot(x=contraceptors,
     y=tfr,
     ylab="Fertility Rate",
     xlab="% of women using contraception",
     main="Average fertility rates & contraception; \n
           50 developing countries",
     xaxp=c(0,1,5)
     )

# Add predicted values to the plot
points(x=contraceptors,y=pred.lm,pch=16,col="red")
```

Example 2: A simple linear regression

```
> summary(res.lm)

Call:
lm(formula = tfr ~ contraceptors)

Residuals:
     Min       1Q   Median       3Q      Max
-1.54934 -0.30133  0.02540  0.39570  1.20214

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)     6.8751     0.1569   43.83   <2e-16 ***
contraceptors  -5.8416     0.3584  -16.30   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5745 on 48 degrees of freedom
Multiple R-Squared: 0.847,      Adjusted R-squared: 0.8438
F-statistic: 265.7 on 1 and 48 DF,  p-value: < 2.2e-16
```

Data and Prediction



Matrix Algebra in R

`det(a)`	Computes the determinant of matrix `a`
`solve(a)`	Computes the inverse of matrix `a`
`t(a)`	Takes the transpose of `a`
`a%*%b`	Matrix multiplication of `a` by `b`
`a*b`	Element by element multiplication

An R list is a basket containing many other variables

```
> x <- list(a=1, b=c(2,15), giraffe="hello")

> x$a
[1] 1

> x$b
[1]  2 15

> x$b[2]
[1] 15

> x$giraffe
[1] "hello"

> x[3]
$giraffe
[1] "hello"

> x[["giraffe"]]
[1] "hello"
```

R lists

Things to remember about lists

- Lists can contain any number of variables of any type
- Lists can contain other lists
- Contents of a list can be accessed by name or by position
- Allow us to move lots of variables in and out of functions
- Functions often return lists (only way to have multiple outputs)

`lm()` basics

```
# To run a regression
res <- lm(y~x1+x2+x3,  # A model formula
          data        # A dataframe (optional)
          )

# To print a summary
summary(res)

# To get the coefficients
res$coefficients

# or
coef(res)

#To get residuals
res$residuals

#or

resid(res)
```

lm() basics

```
# To get the variance-covariance matrix of the regressors
vcov(res)

# To get the standard errors
sqrt(diag(vcov(res)))

# To get the fitted values
predict(res)

# To get expected values for a new observation or dataset
predict(res,
        newdata,                   # a dataframe with same x vars
                                   # as data, but new values
        interval = "confidence",   # alternative: "prediction"
        level = 0.95
        )
```

R lists & Object Oriented Programming

A list object in R can be given a special "class" using the `class()` function

This is just a metatag telling other R functions that this list object conforms to a certain format

So when we run a linear regression like this:

```
res <- lm(y~x1+x2+x3, data)
```

The result `res` is a list object of class `''lm''`

Other functions like `plot()` and `predict()` will react to `res` in a special way because of this class designation

Specifically, they will run functions called `plot.lm()` and `predict.lm()`

Object-oriented programming:
a function does different things depending on class of input object

An example: Party systems & Redistribution

Cross sectional data on industrial democracies:

povertyReduction	Percent of citizens lifted out of poverty by taxes and transfers
effectiveParties	Effective number of parties
partySystem	Whether the party system is Majoritarian, Proportional, or Unanimity (Switzerland)

Source of data & plot: Torben Iversen and David Soskice, 2002, "Why do some democracies redistribute more than others?" Harvard University.

Considerations:

1. The marginal effect of each extra party is probably diminishing, so we want to log the effective number of parties

2. The party system variable needs to be "dummied out;" there are several ways to do this

An example: Party systems & Redistribution

```
# Clear memory of all objects
rm(list=ls())

# Load libraries
library(RColorBrewer)       # For nice colors

# Load data
file <- "iverRevised.csv"
iversen <- read.csv(file,header=TRUE)

# Create dummy variables for each party system
iversen$majoritarian <- as.numeric(iversen$partySystem=="Majoritarian")
iversen$proportional <- as.numeric(iversen$partySystem=="Proportional")
iversen$unanimity <- as.numeric(iversen$partySystem=="Unanimity")

# A bivariate model, using a formula to log transform a variable
model1 <- povertyReduction ~ log(effectiveParties)
lm.res1 <- lm(model1, data=iversen)
summary(lm.res1)
```

An example: Party systems & Redistribution

```
Call:
lm(formula = model1, data = iversen)

Residuals:
    Min      1Q  Median      3Q     Max
-48.907  -4.115   8.377  11.873  18.101

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)              21.80      16.15   1.349   0.2021
log(effectiveParties)    24.17      12.75   1.896   0.0823 .
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 19.34 on 12 degrees of freedom
Multiple R-squared:  0.2305,Adjusted R-squared:  0.1664
F-statistic: 3.595 on 1 and 12 DF,  p-value: 0.08229
```

An example: Party systems & Redistribution

```
# A new model with multiple regressors
model2 <- povertyReduction ~ log(effectiveParties) + majoritarian
            + proportional
lm.res2 <- lm(model2, data=iversen)
summary(lm.res2)
```

An example: Party systems & Redistribution

```
Call:
lm(formula = model2, data = iversen)

Residuals:
     Min       1Q   Median       3Q      Max
-23.3843  -1.4903   0.6783   6.2687  13.9376

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)             -31.29      26.55  -1.178  0.26588
log(effectiveParties)    26.69      14.15   1.886  0.08867 .
majoritarian             48.95      17.86   2.740  0.02082 *
proportional             58.17      13.52   4.302  0.00156 **
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 12.37 on 10 degrees of freedom
Multiple R-squared:  0.7378,Adjusted R-squared:  0.6592
F-statistic: 9.381 on 3 and 10 DF,  p-value: 0.002964
```

An example: Party systems & Redistribution

```
# A new model with multiple regressors and no constant
model3 <- povertyReduction ~ log(effectiveParties) + majoritarian
    + proportional + unanimity - 1
lm.res3 <- lm(model3, data=iversen)
summary(lm.res3)
```

An example: Party systems & Redistribution

```
Call:
lm(formula = model3, data = iversen)

Residuals:
     Min        1Q    Median        3Q       Max
-23.3843  -1.4903   0.6783   6.2687  13.9376

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
log(effectiveParties)    26.69      14.15   1.886   0.0887 .
majoritarian             17.66      12.69   1.392   0.1941
proportional             26.88      21.18   1.269   0.2331
unanimity               -31.29      26.55  -1.178   0.2659
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 12.37 on 10 degrees of freedom
Multiple R-squared:  0.9636,Adjusted R-squared:  0.949
F-statistic: 66.13 on 4 and 10 DF,  p-value: 3.731e-07
```

An example: Party systems & Redistribution

```
# A new model with multiple regressors and an interaction
model4 <- povertyReduction ~ log(effectiveParties) + majoritarian
    + proportional + log(effectiveParties):majoritarian
lm.res4 <- lm(model4, data=iversen)
summary(lm.res4)
```

An example: Party systems & Redistribution

```
Call:
lm(formula = model4, data = iversen)

Residuals:
     Min       1Q   Median       3Q      Max
-22.2513   0.0668   2.8532   4.7318  12.9948

Coefficients:
                                   Estimate Std. Error t value Pr(>|t|)
(Intercept)                          -14.83      31.42  -0.472  0.64813
log(effectiveParties)                 16.78      17.39   0.965  0.35994
majoritarian                          16.34      37.65   0.434  0.67445
proportional                          56.18      13.70   4.102  0.00267 **
log(effectiveParties):majoritarian    29.55      30.02   0.984  0.35065
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 12.39 on 9 degrees of freedom
Multiple R-squared:  0.7633,Adjusted R-squared:  0.6581
F-statistic: 7.256 on 4 and 9 DF,  p-value: 0.006772
```

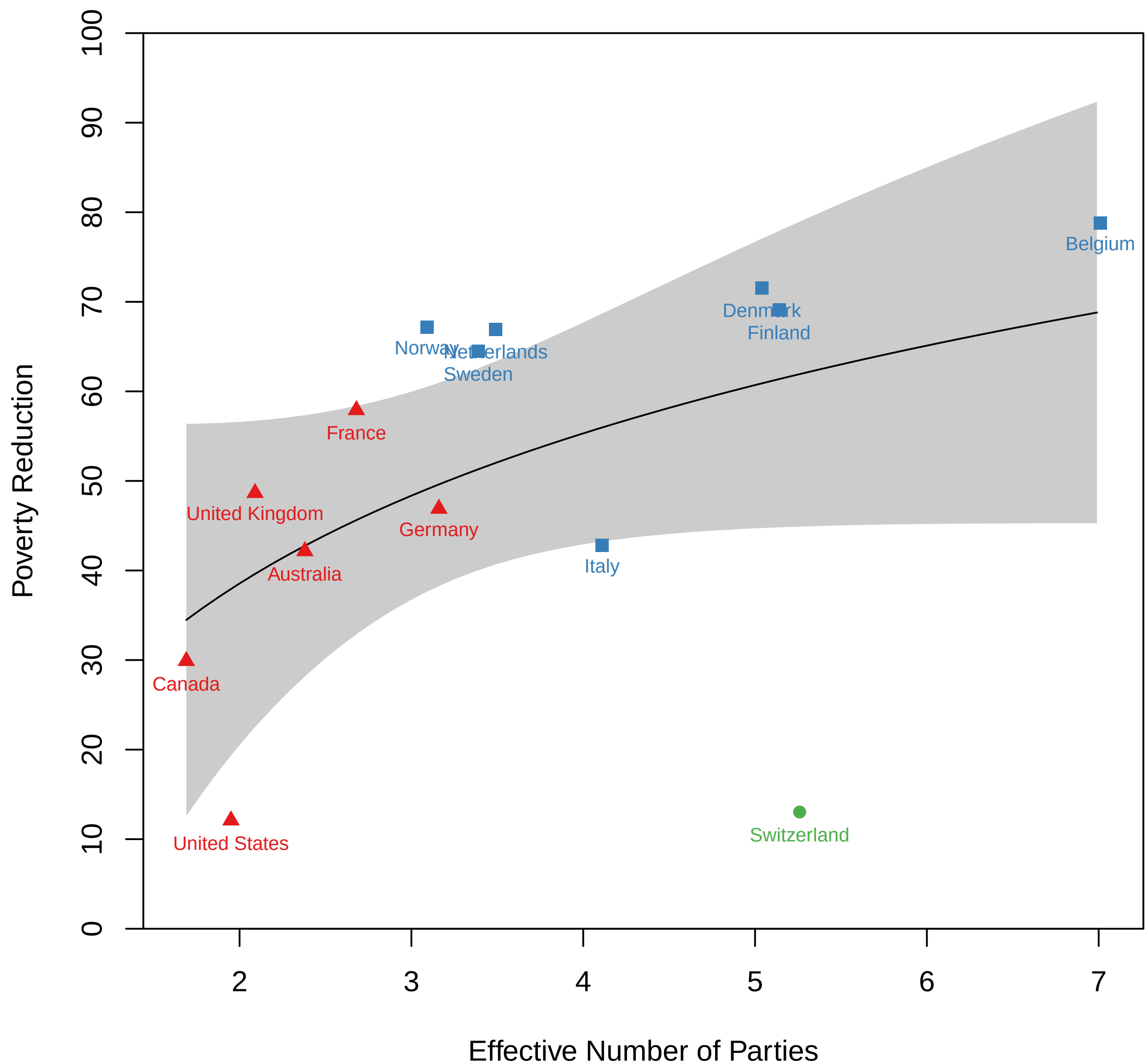
An example: Party systems & Redistribution

```
# A more efficient way to specify an interaction
model5 <- povertyReduction ~ log(effectiveParties)*majoritarian +
    proportional
lm.res5 <- lm(model5, data=iversen)
summary(lm.res5)
```

An example: Party systems & Redistribution

```
Call:
lm(formula = model5, data = iversen)

Residuals:
     Min       1Q   Median       3Q      Max
-22.2513   0.0668   2.8532   4.7318  12.9948

Coefficients:
                                   Estimate Std. Error t value Pr(>|t|)
(Intercept)                          -14.83      31.42  -0.472  0.64813
log(effectiveParties)                 16.78      17.39   0.965  0.35994
majoritarian                          16.34      37.65   0.434  0.67445
proportional                          56.18      13.70   4.102  0.00267 **
log(effectiveParties):majoritarian    29.55      30.02   0.984  0.35065
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 12.39 on 9 degrees of freedom
Multiple R-squared:  0.7633,Adjusted R-squared:  0.6581
F-statistic: 7.256 on 4 and 9 DF,  p-value: 0.006772
```

Plotting a best fit line



Let's turn to the code to see how we can make this plot using R base graphics