POLS/CSSS 503:
Advanced Quantitative Political Methodology

Introduction to the Course & to R

Christopher Adolph

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

POLS 503 Course Goals

Course Goals:

1. Learn the properties and limitations of the linear regression model
2. Develop further skills in interpreting & fitting linear models
3. Study extensions of the linear model to deal with common problems
4. Become comfortable with matrix algebra represention of models
5. Gain proficiency in `R`, a powerful and popular statistical package
6. Get ready to take POLS/CSSS 510: MLE, or other advanced CSSS courses

Agenda for this week

- Introductory example: why focus on model interpretation?
- Overview of syllabus & course requirements
- Introduction to `R`

How do we intrepret our models?

For simple versions of linear regression,
coefficients & se's usefully summarize the relation between x and y

But how do we interpret β when we add to the linear regression model. . .

- transformed variables? $\log(\text{income}) = \alpha + \beta_1 \text{age} + \varepsilon$

  or

  $\text{wealth} = \alpha + \beta_2 \text{age} + \beta_3 \text{age}^2 + \varepsilon$

- interactions? $\text{policy} = \alpha + \beta_4 \text{preferences} + \beta_5 \text{autonomy} + \beta_6(\text{preferences} \times \text{autonomy}) + \varepsilon$

- time series dynamics? $\text{budget}_t = \alpha + \beta_7 \text{budget}_{t-1} + \beta_8 \text{partisanship}_t + \varepsilon_t$

- multiple equations? $\text{campaign spending} = \alpha + \beta_9 \text{competitiveness} + \varepsilon$

  and

  $\text{competitiveness} = \eta + \beta_{10} \text{campaign spending} + \nu$

- non-linear models? $\Pr(\text{war}) = (1 - \exp(-\alpha - \beta_{11} \text{distance}))^{-1}$

How do we intrepret our models?

As models get more complicated,
learning to effectively interpret and present them gets more important

Model coefficients are not always easy to interpret

Focus on "coefficients alone" (or worse, "stars alone") carries risks:

- being ignored or misunderstood by those who don't understand the model
- misunderstanding your own model
- failing to see the relevant implications of your model

Graphics usually do a better job of explaining and exploring regression models

A famous simple example shows this well

The *Challenger* launch decision

In 1986, the Challenger space shuttle exploded moments after liftoff

Decision to launch one other most scrutinized in history

Failure of O-rings in the solid-fuel rocket boosters blamed for explosion

Could this failure have been foreseen?

The *Challenger* launch decision

Flights with O-ring damage	
Flt Number	Temp (F)
2	70
41b	57
41c	63
41d	70
51c	53
61a	79
61c	58

Morton-Thiokol engineers made this table
and worried about launching below 53 degrees (Why?)

O-ring would erode or have "blow-by" (2 ways to fail) in cold temp

Failed to convince administrators there was a danger

(Counter-argument: "damages at low and high temps")

Are there problems with this presentation? with the use of data?

The *Challenger* launch decision

Engineerrs did not consider successes, only failures;
selection on the dependent variable

All flights, chronological order			
Damage?	Temp (F)	Damage?	Temp (F)
No	66	No	78
Yes	70	No	67
No	69	**Yes**	53
No	68	No	67
No	67	No	75
No	72	No	70
No	73	No	81
No	70	No	76
Yes	57	**Yes**	79
Yes	63	No	76
Yes	70	**Yes**	58

Other problems? Why sort by launch number?

The *Challenger* launch decision

O-ring damage pre-Challenger, by temperature at launch			
Damage?	Temp (F)	Damage?	Temp (F)
Yes	53	**Yes**	70
Yes	57	No	70
Yes	58	No	70
Yes	63	No	72
No	66	No	73
No	67	No	75
No	67	No	76
No	67	No	76
No	68	No	78
No	69	**Yes**	79
Yes	70	No	81

The evidence begins to speak for itself.

What if Morton-Thiokol engineers had made this table before the launch?

The *Challenger* launch decision

Why didn't NASA make the right decision?

Many answers in the literature:
bureaucratic politics; group think; bounded rationality, etc.

But Edward Tufte thinks it may have been a matter of presentation & modeling:

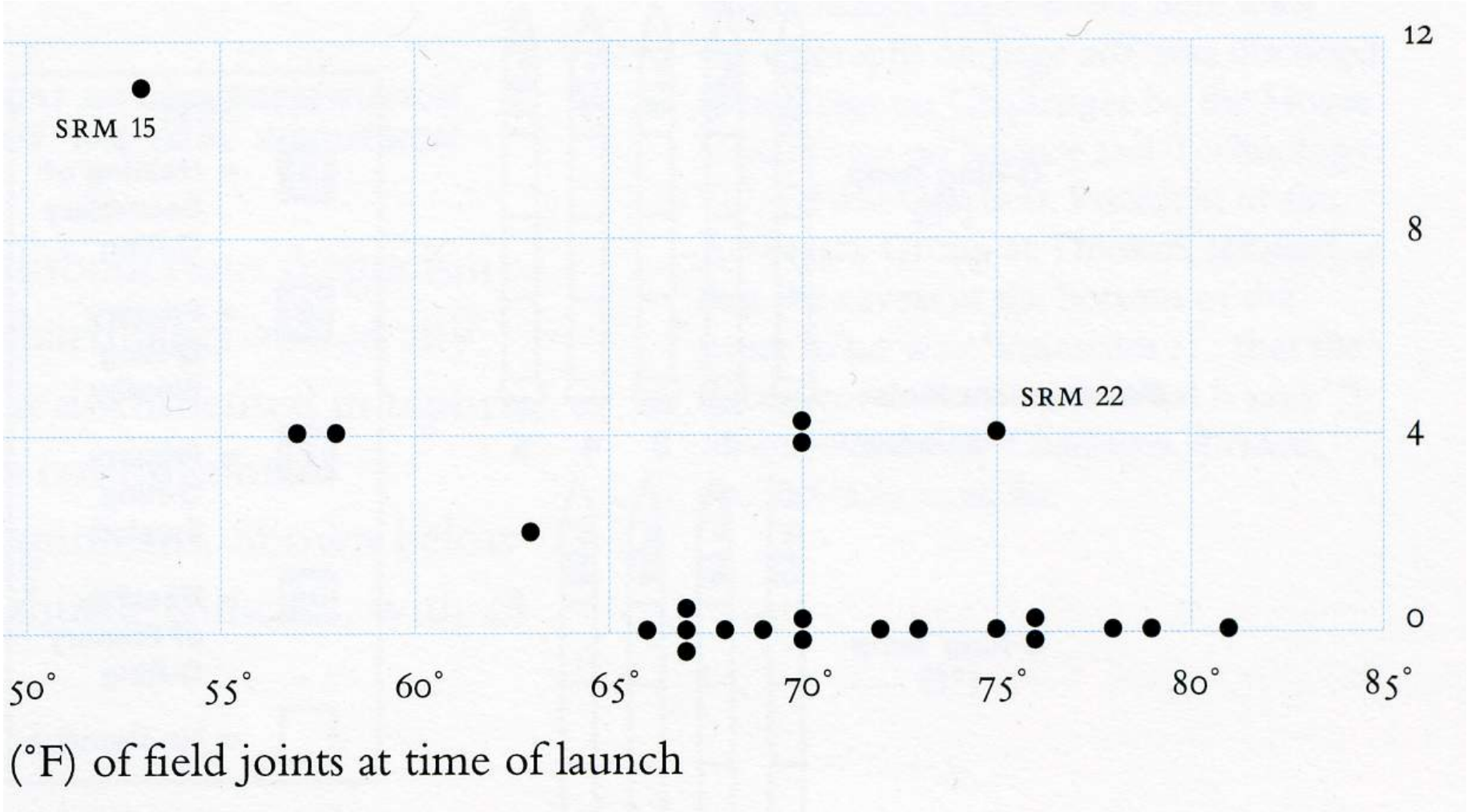
- Never made the right tables or graphics
- Selected only failure data
- Never considered a simple statistical model

What do you think? How would you approach the data?

The *Challenger* launch decision

How about a scatterplot? Better for seeing relationships than a table

Vertical axis is an O-ring damage index (due to Tufte, who made the plot)



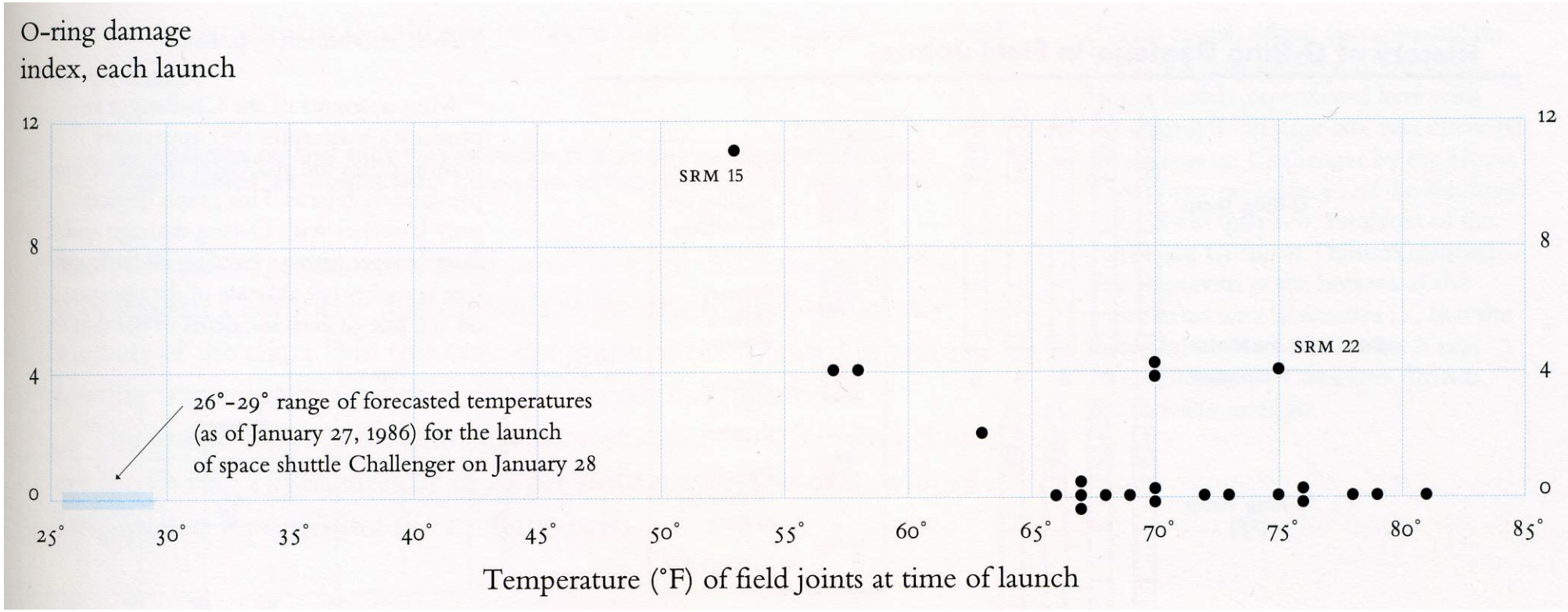
Suspicious. What the forecast temperature for launch?

The *Challenger* launch decision

What the forecast temperature for launch?

The *Challenger* launch decision

What the forecast temperature for launch? 26 to 29 degrees Fahrenheit!



The shuttle was launched in unprecendented cold

The *Challenger* launch decision

Imagine you are the analyst making the launch recommendation

You've made the scatterplot above. What would you add to it?

Put another way, what do you is the first question you expect from your boss?

"What's the chance of failure at 26 degrees?"

The scatterplot suggests the answer is "high," but that's vague

But what if the next launch is at 58 degrees? Or 67 degrees?

Clearly, we want a more precise way to state the probability of failure

We need a *model*, and a way to convey that model to the public

The *Challenger* launch decision

A simple exercise is to model the probability of O-ring damage
as a function of temperature

We can use a statistical tool called "logit" for this purpose

The model is nonlinear: $\Pr(\text{damage}) = (1 - \exp(-\beta_0 - \beta_1 \text{temperature}))^{-1}$

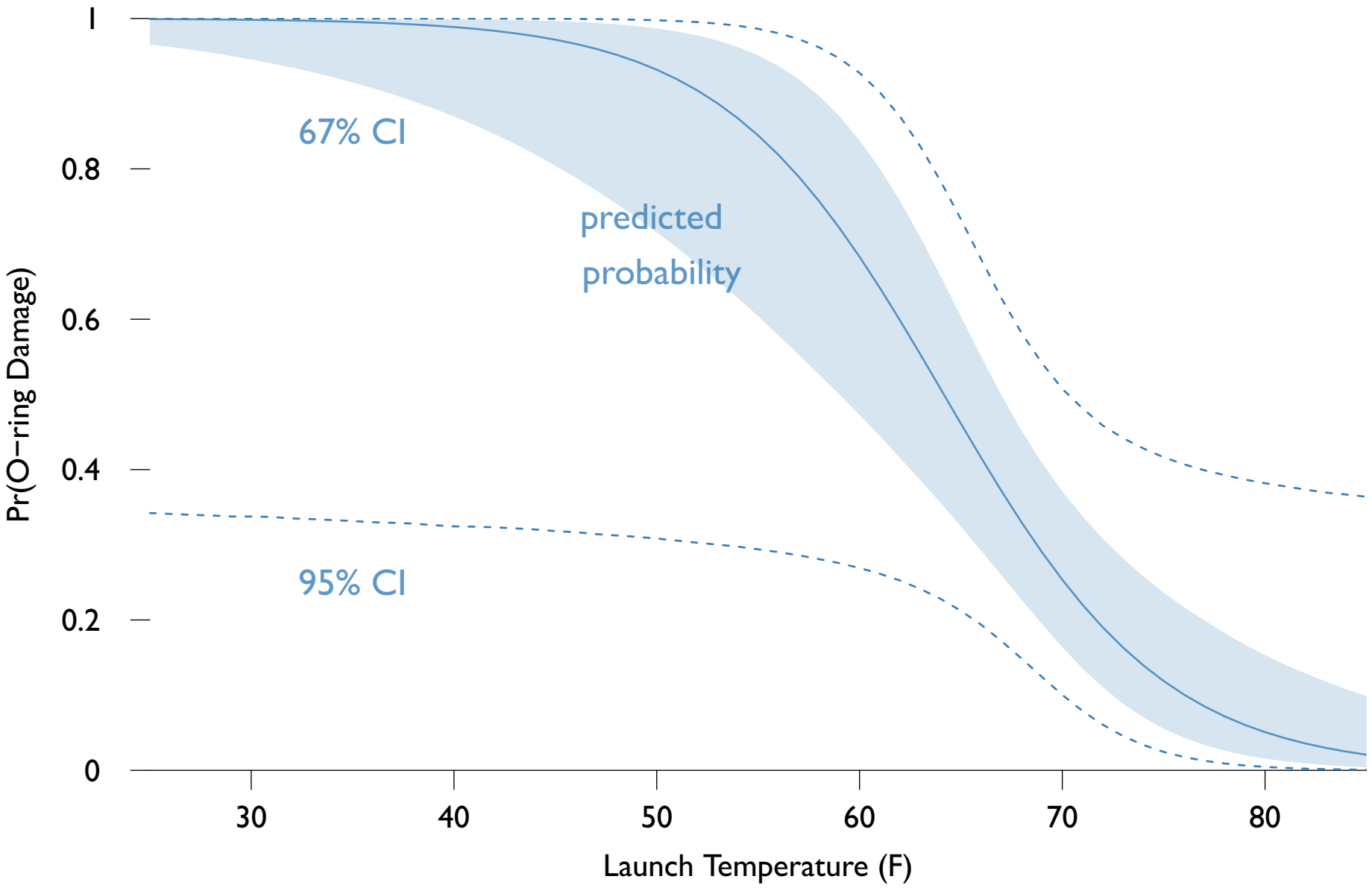
R gives us this lovely logit output. . .

Variable	est.	s.e.	p
Temperature (F)	-0.18	0.09	0.047
Constant	11.9	6.34	0.062
N	22		
log-likelihood	-10.9		

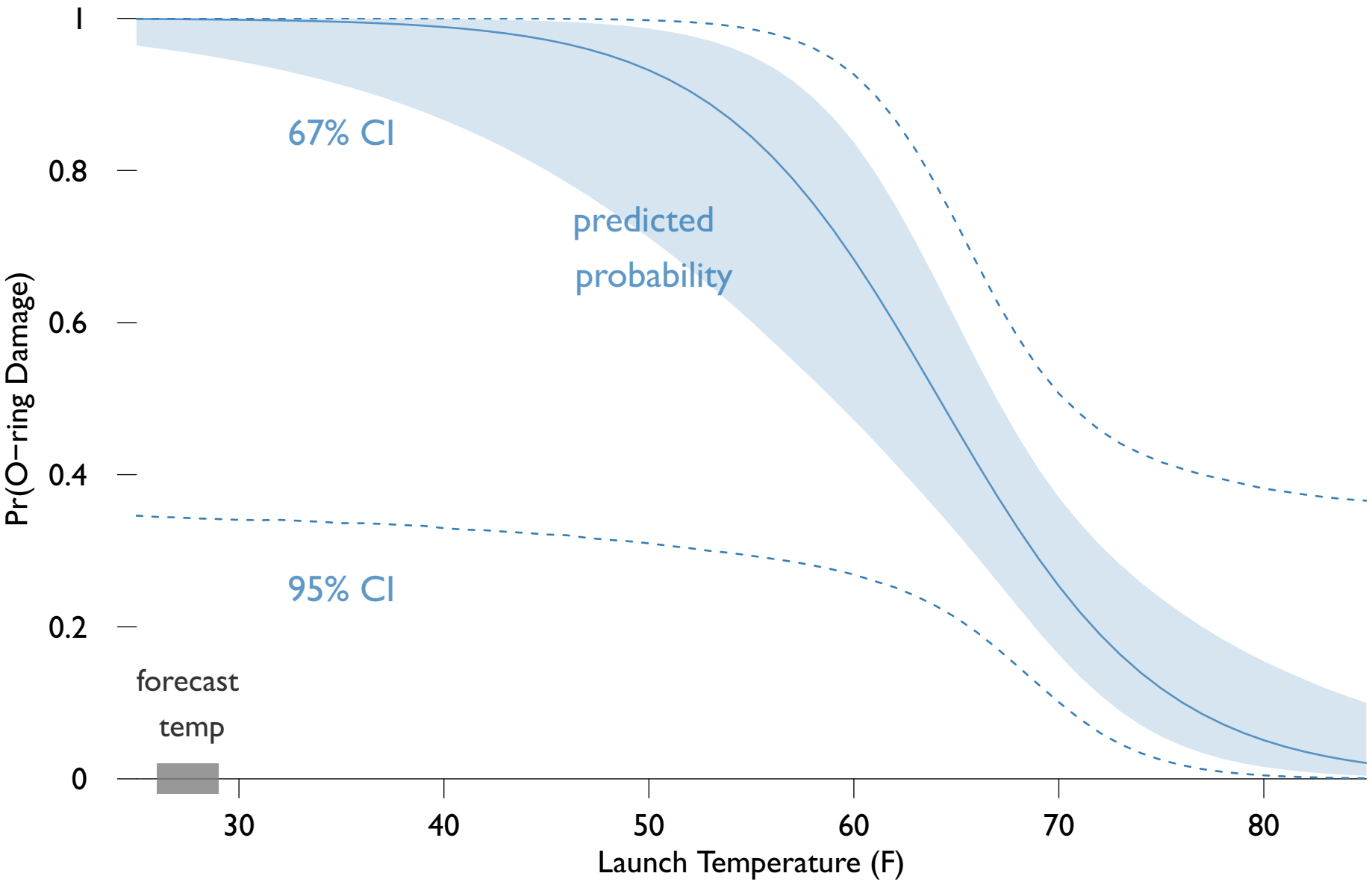
which most social scientists read as "a statistically significant negative relationship b/w temperature and probability of damage"

But that's pretty vague too

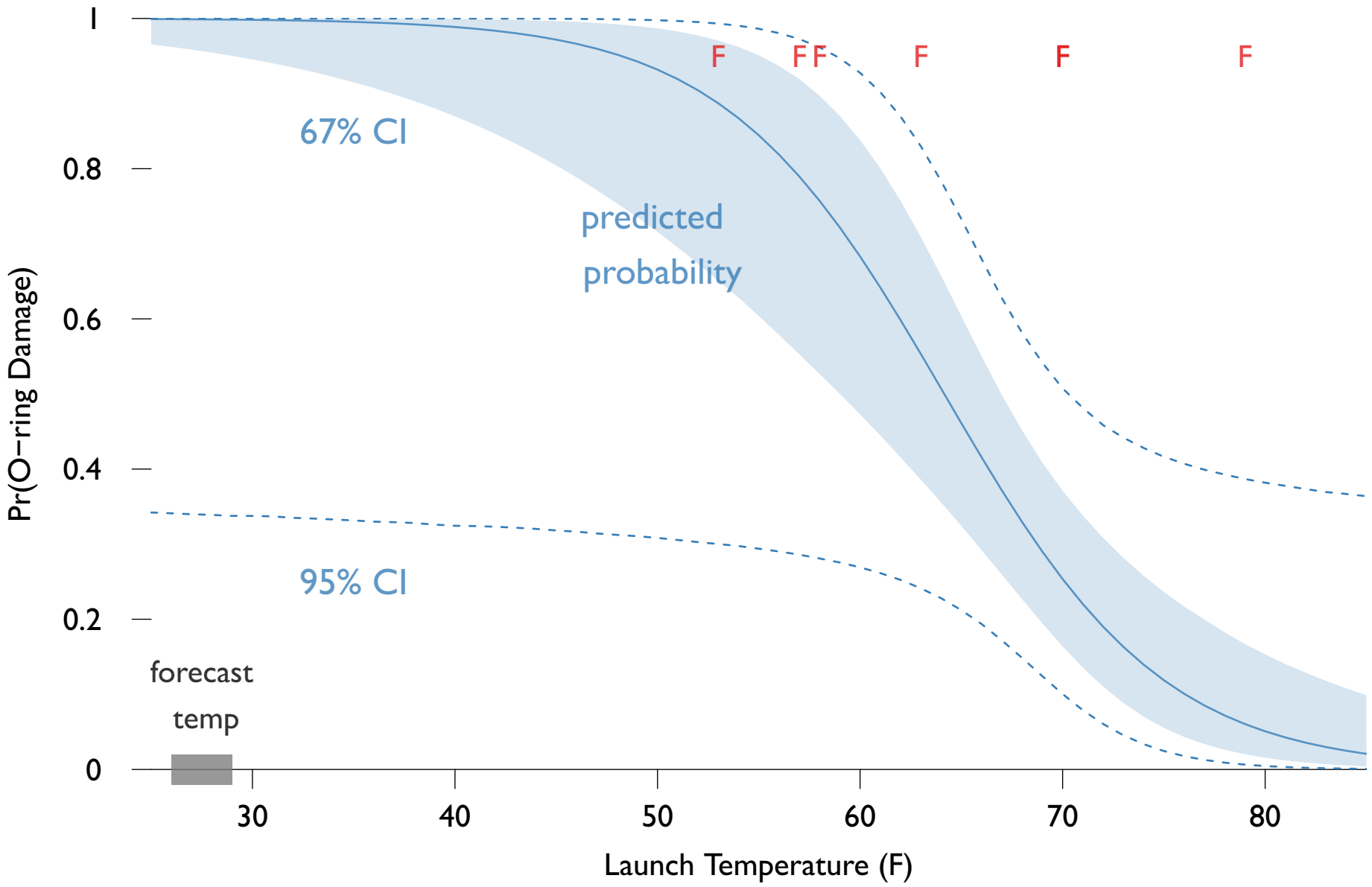
Is there a more persuasive/clear/useful way to present these results?



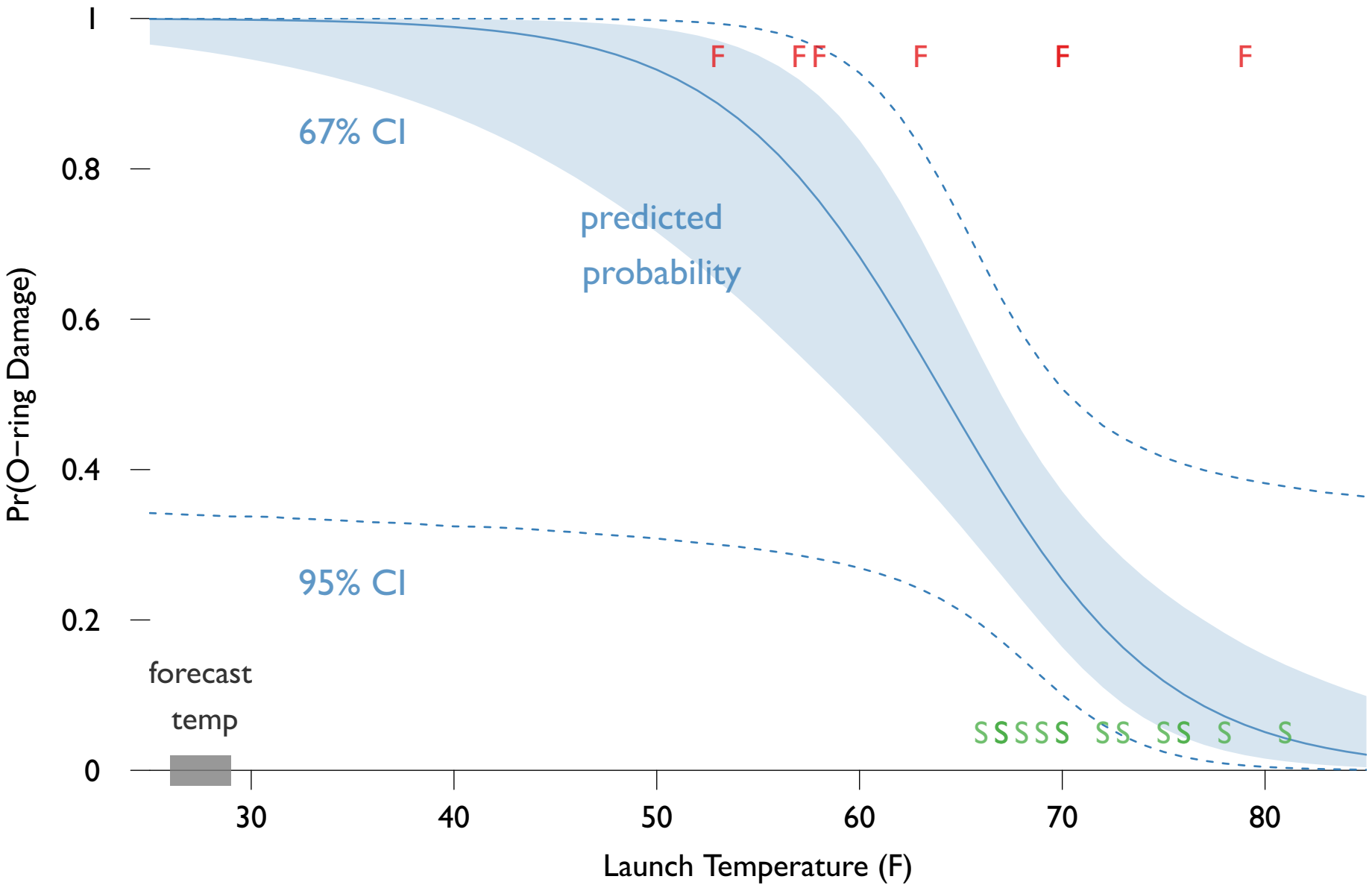
A picture clearly shows non-linear model predictions *and* uncertainty



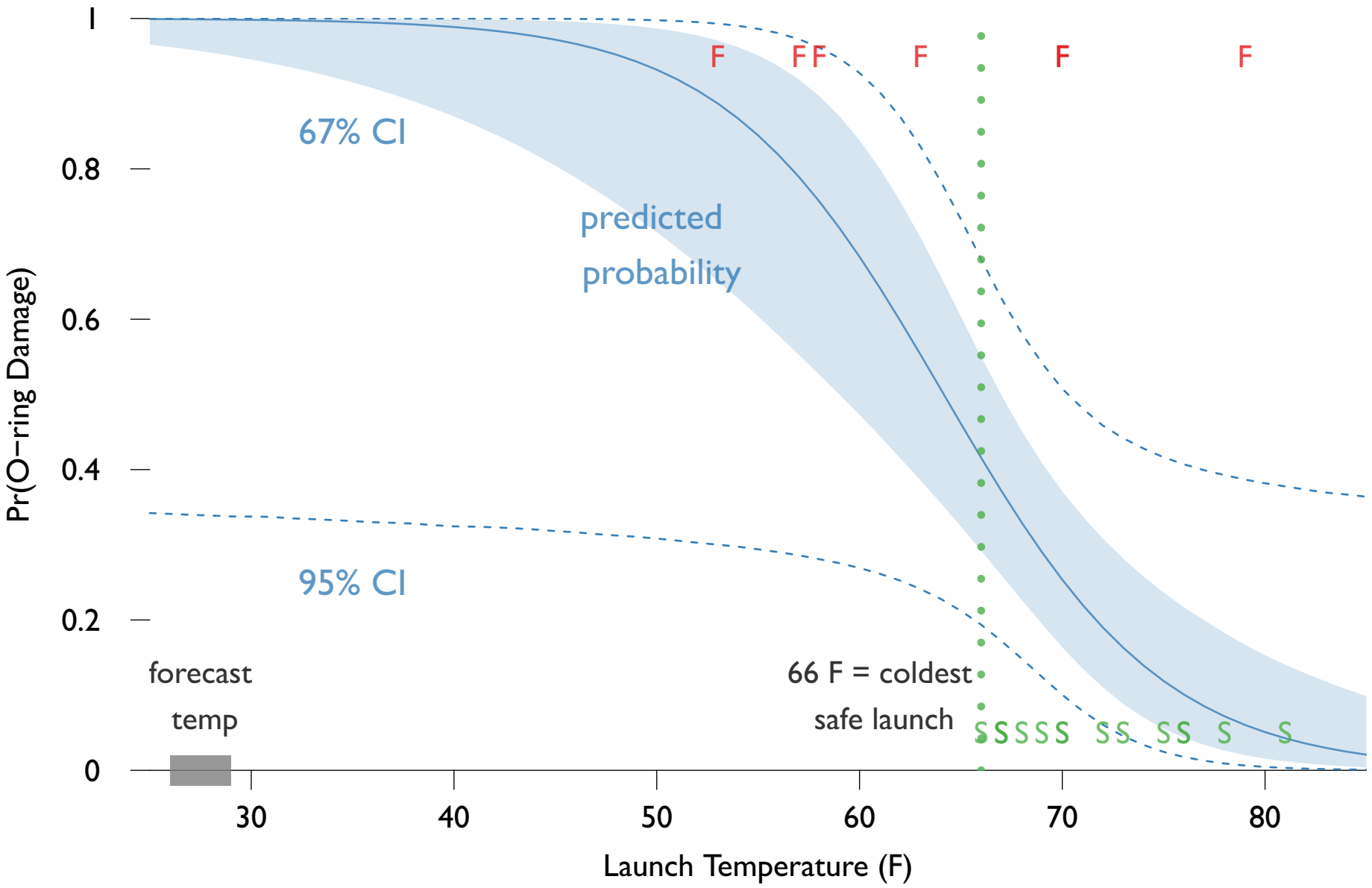
And gives a more precise sense of how foolhardy launching at 29 F is.



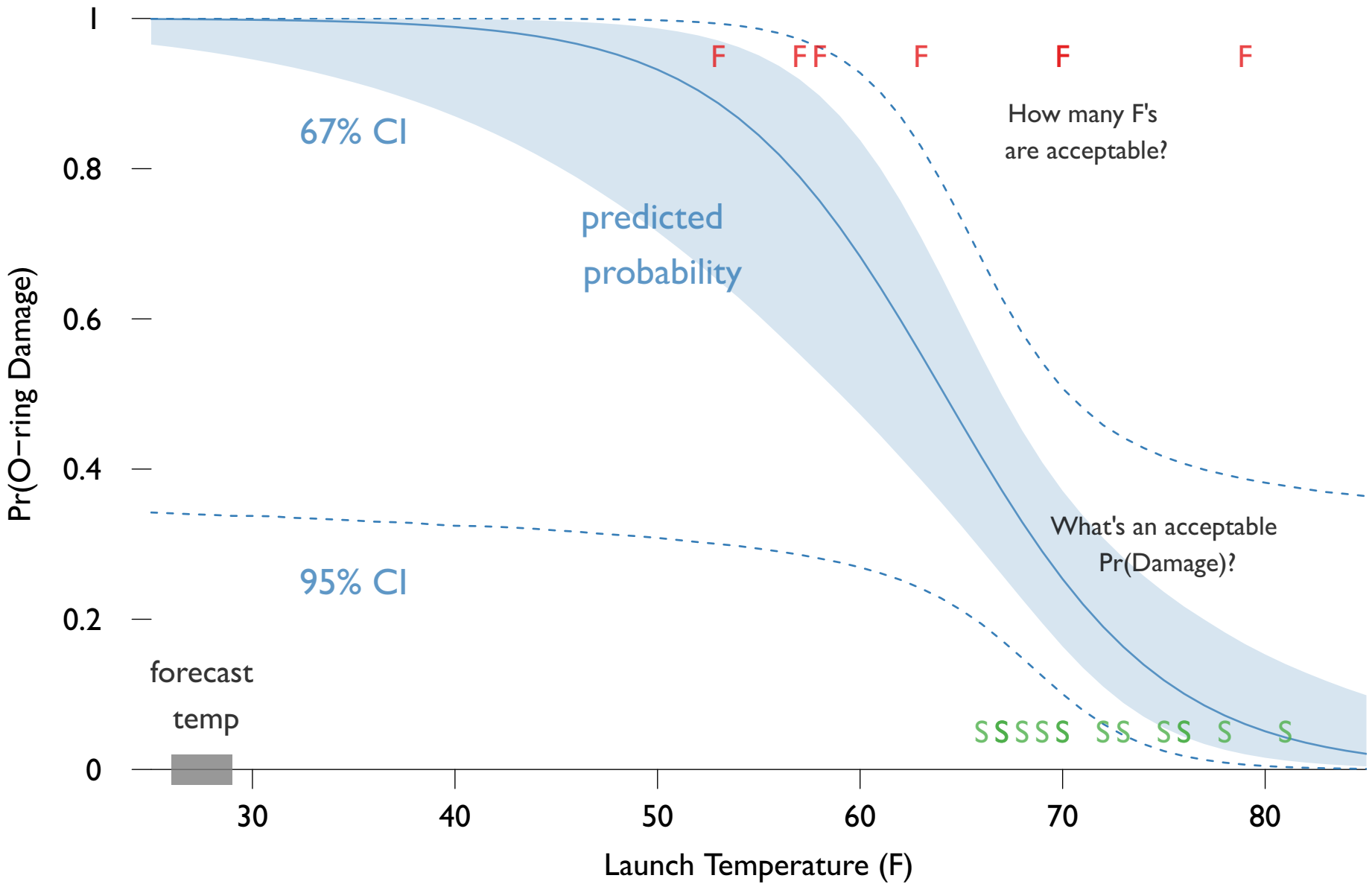
It's also good to show the data giving rise to the model.



Remembering that the Failures are only meaningful compared to Successes

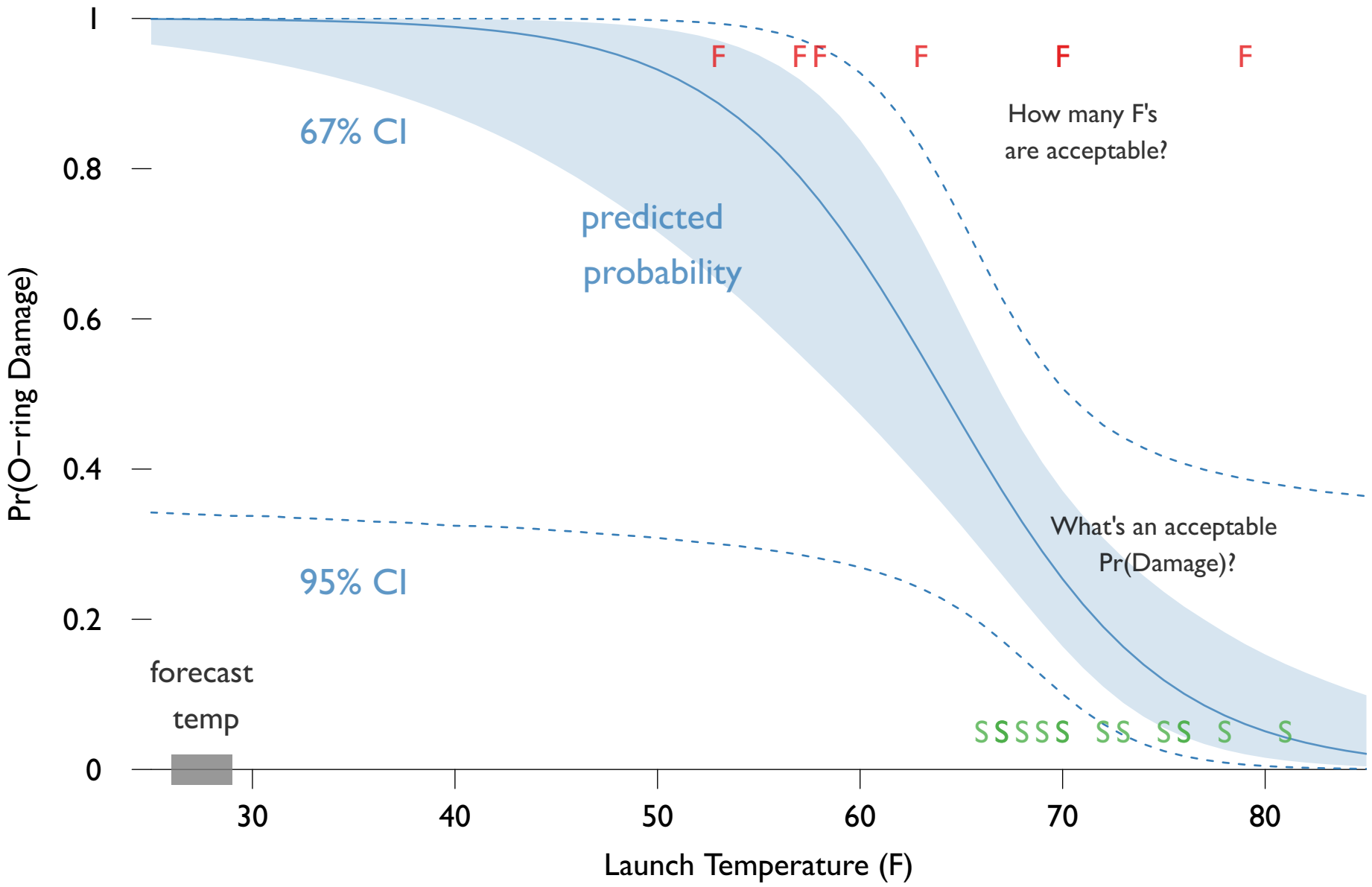


Looking just at the data tempts us to say that launches under 66 F are virtually guaranteed O-ring failures. This inference is based on an unstated model.



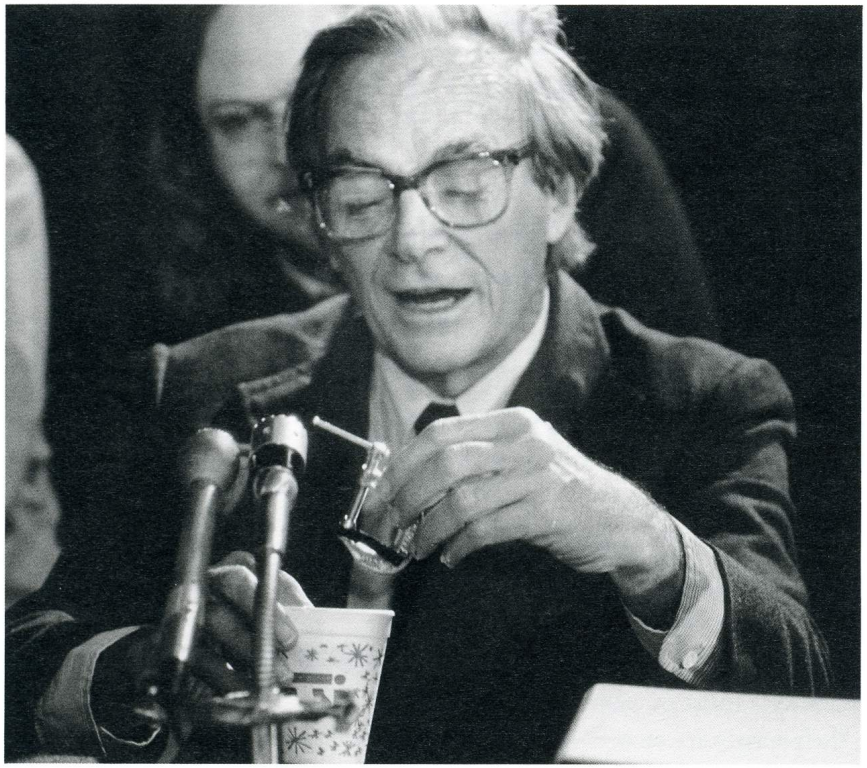
But the estimated logit model should give us pause.

There is a significant risk of failure across the board.



What's an acceptable risk of O-ring failure?

Was the shuttle safe at any temperature?



In a hearing, Richard Feynmann dramatically showed O-rings lose resilence when cold by dropping one in his ice water.

Experiment cut thru weeks of technical gibberish concealing flaws in the O-ring

But it shouldn't have taken a Nobel laureate:
any scientist with a year of statistical training could have
used the launch record to reach the same conclusion

And it would take no more than a single graphic to show the result

Going further

The Challenger example involves a simple, bivariate model

But even it goes beyond linear regression: the binary outcome requires a logit model

What else about these data might go beyond the simple linear regression framework?

- Serial correlation? Does wear from the last launch makes damage more likely next time?
- Panel structure? Perhaps each rocket booster needs its own model

Going further

Social science data tend to be even more complicated:

Always many variables

Often interactive effects on responses

Serial correlation, heteroskedasticity, reverse causation, missing data. . .

In 503, we'll expand the linear model to cope with these problems and more

And prepare for future classes expanding your toolkit beyond linear modeling

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,. . . }

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")
```

You can access a variable from a dataframe directly using `$`:

```
a$Var1
```

Dataframes can also 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

Suppose you load a dataset using

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

You can check out the names of the variables using `names(data)`

And access any variables, such as `gdp`, using `data$gdp`

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 one dataset can 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, and 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:

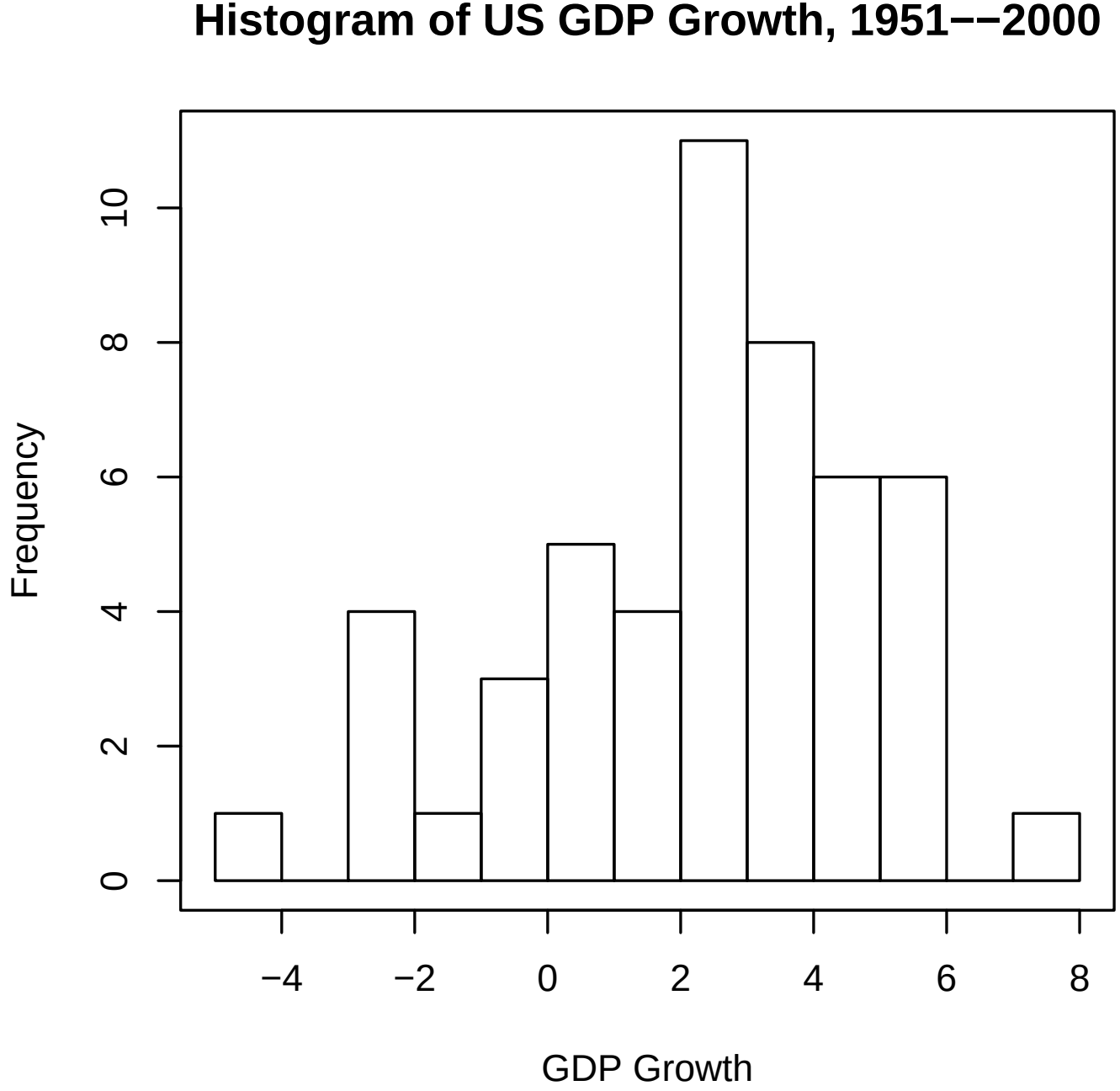
`grgdpch`	The per capita GDP growth rate
`party`	The party of the president (Dem = -1, Rep = 1)

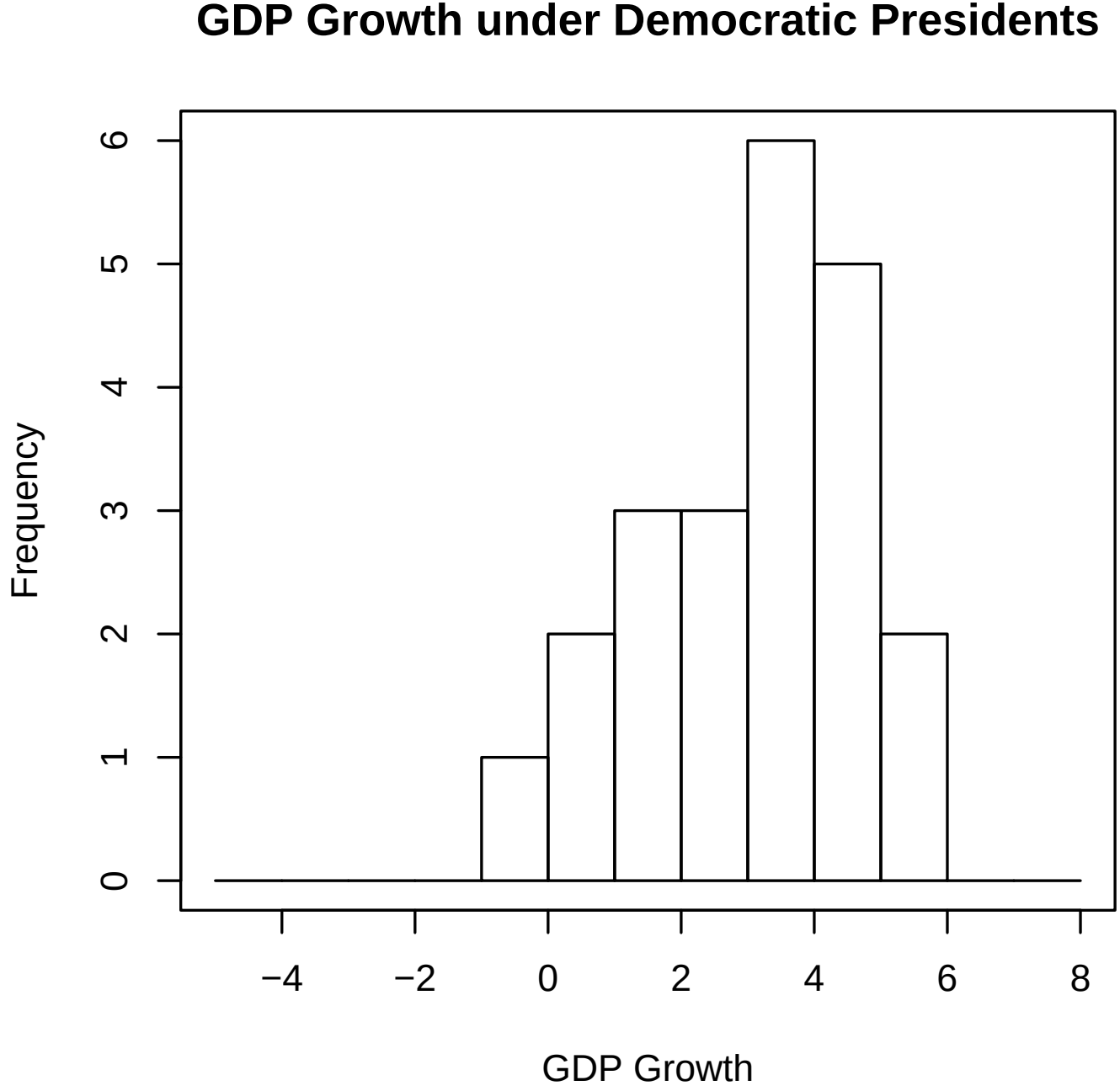
Example 1: US Economic growth

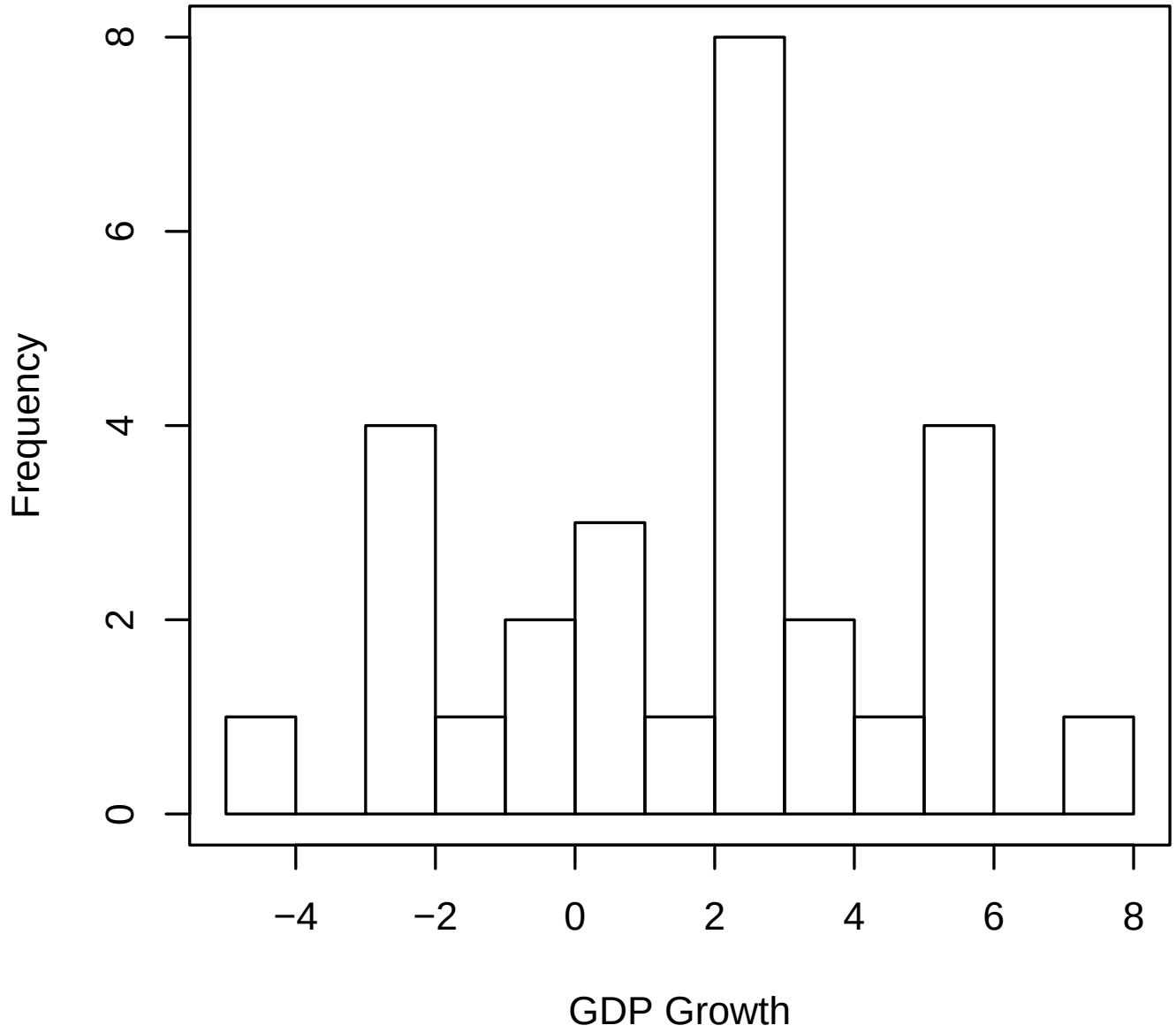
```
# Load data
data <- read.csv("gdp.csv",na.strings="")
attach(data)

# 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

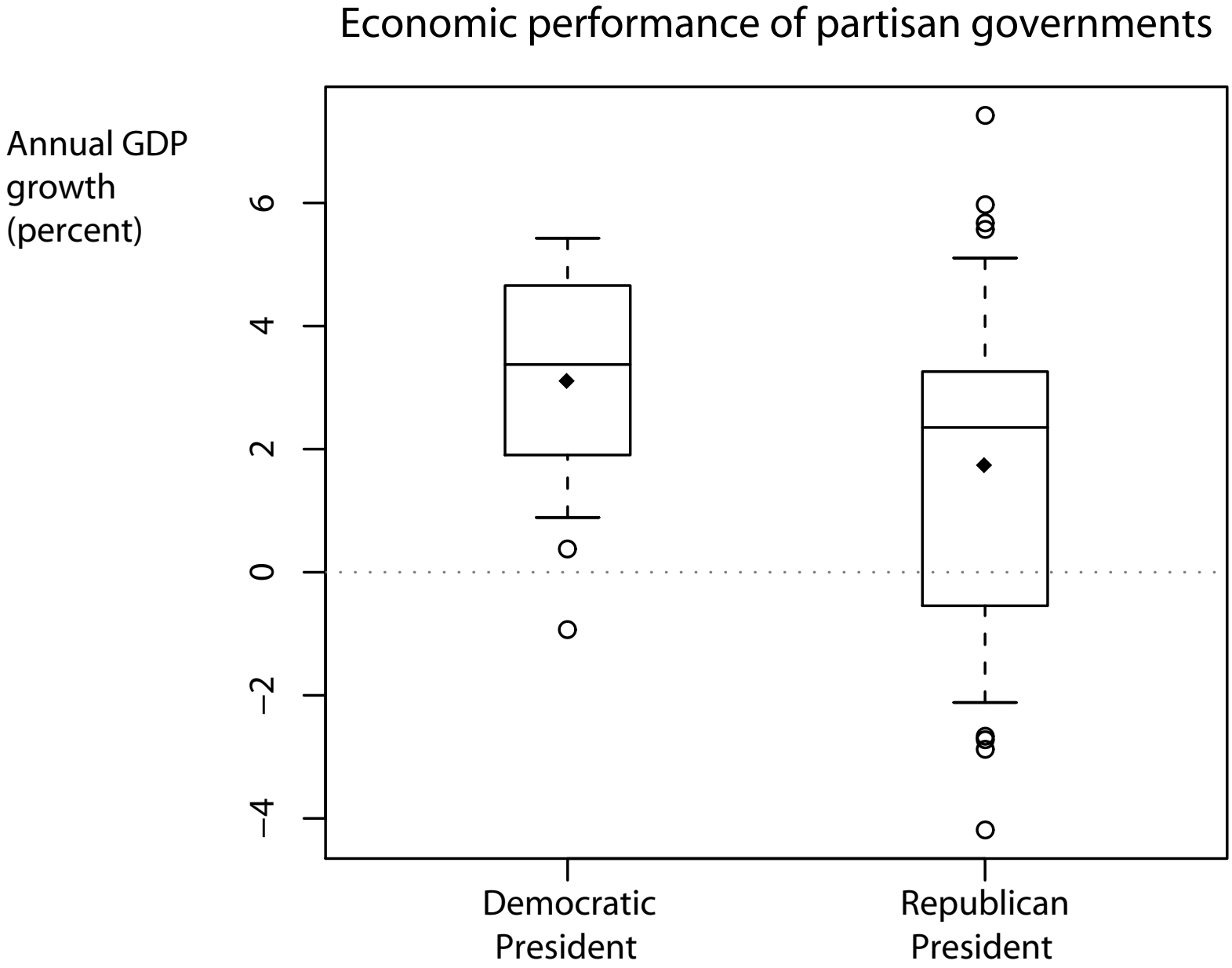
`y~x1+x2+x3`

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

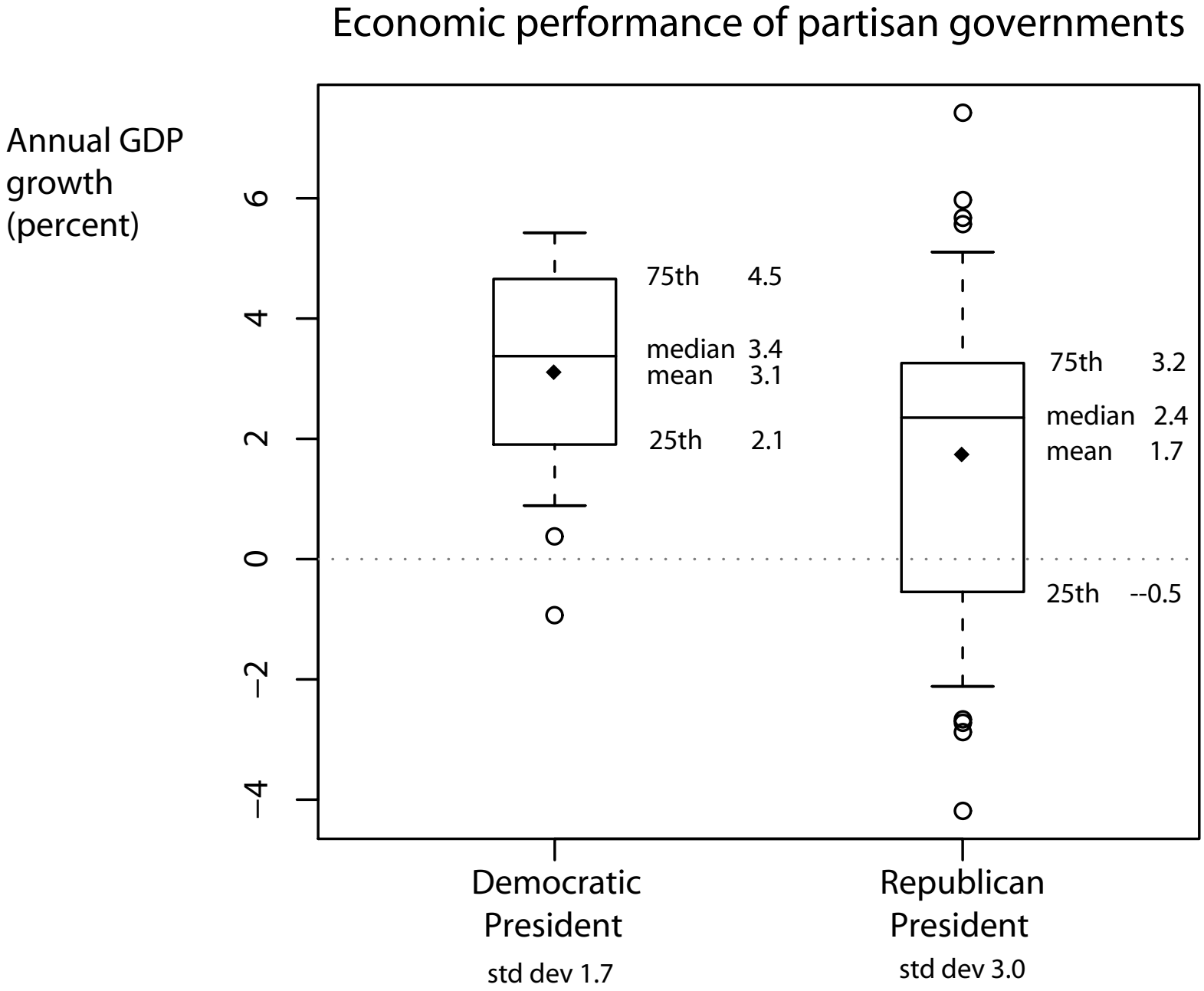
`boxplot()` needs party to be a "factor" or an explicitly categorical variable

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

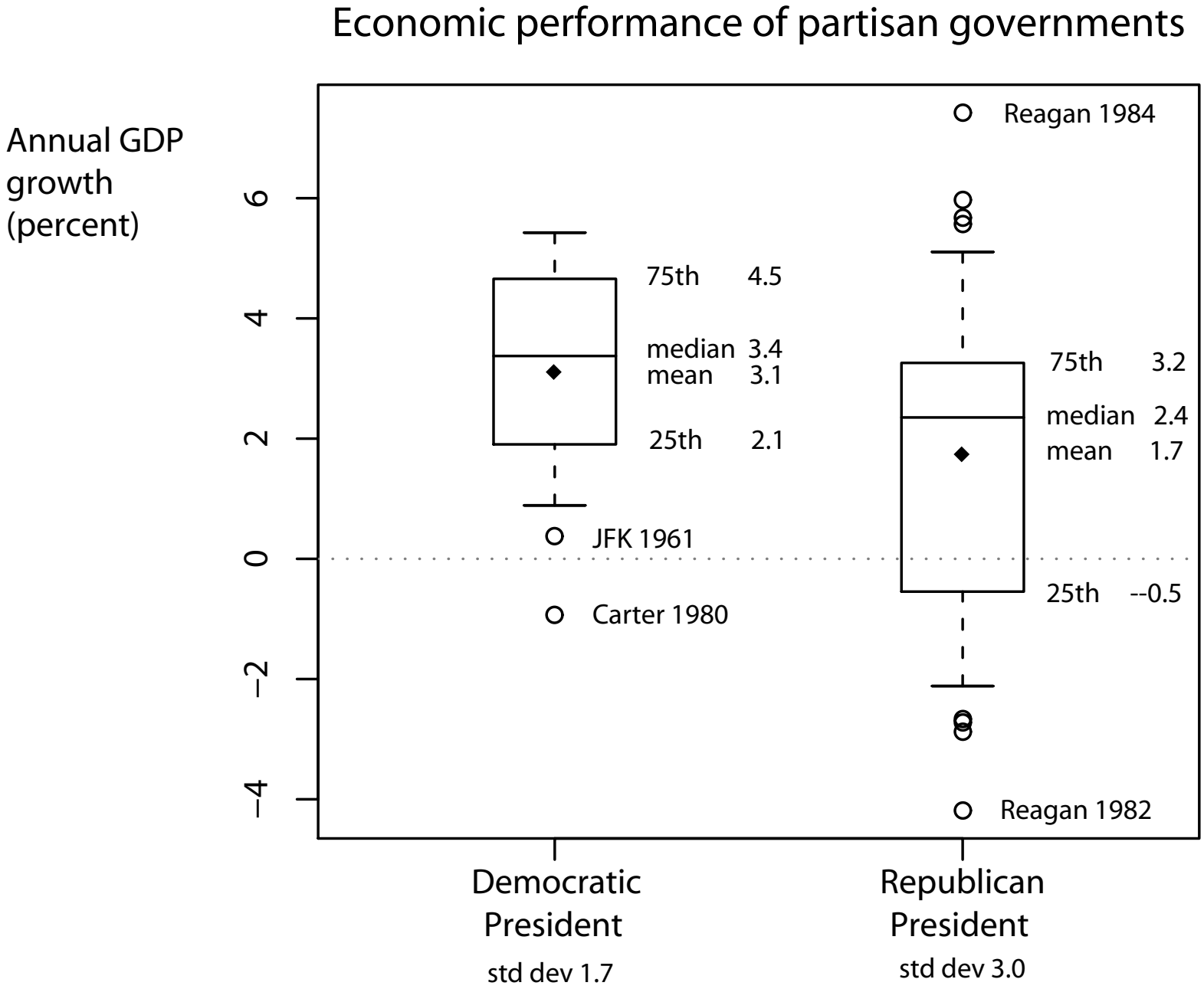
Box plots: Annual US GDP growth, 1951–2000



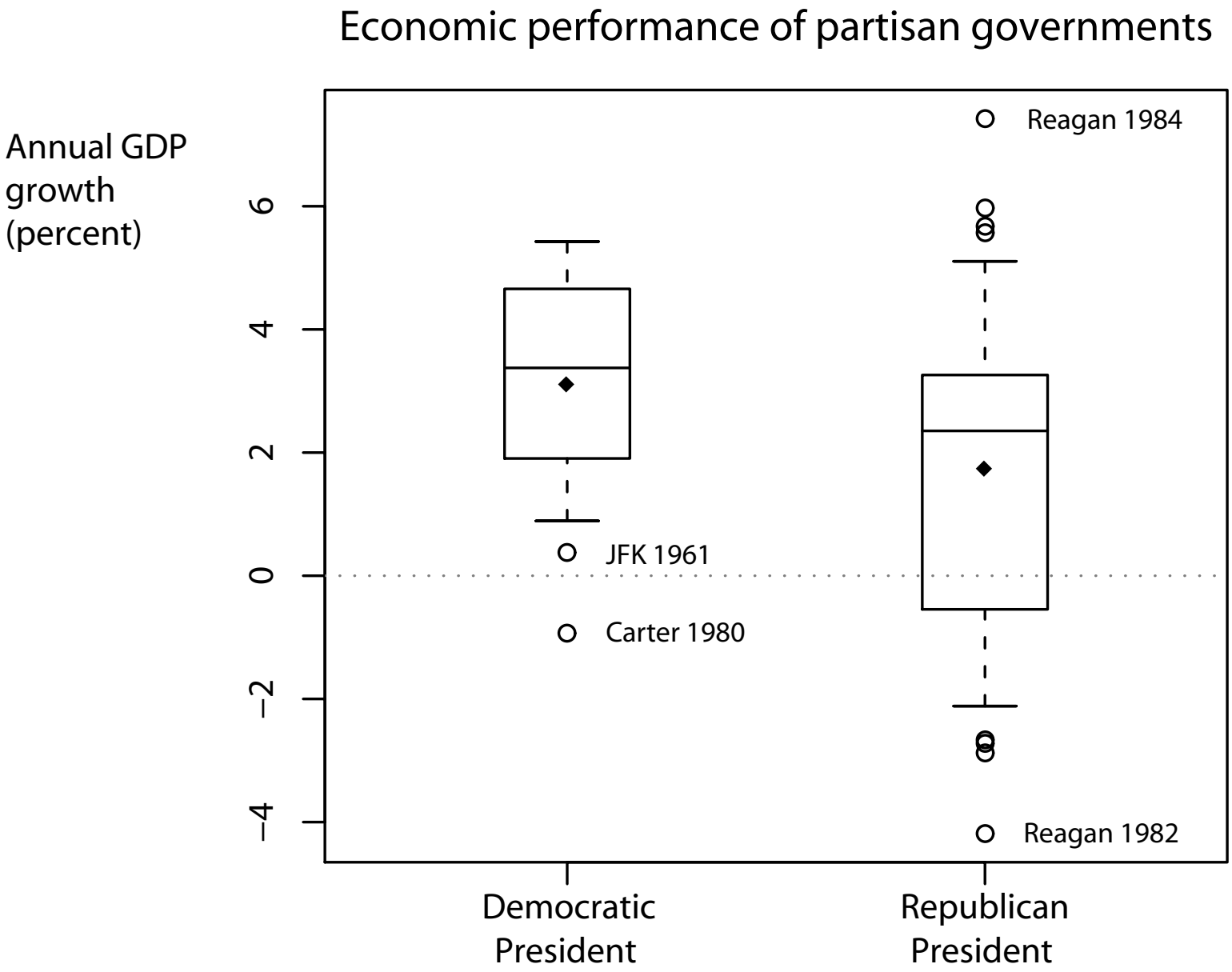
Box plots: Annual US GDP growth, 1951–2000



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)`

Hard to use Google directly for R help ("r" is kind of a common letter)

Easiest way to get help from the web: `rseek.org`

Rseek tries to limit results to R topics (not wholly successful)

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-3.0.3-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-3.0.3.pkg"

- 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 powerful. Windows only. `http://www.sciviews.org/Tinn-R/`
- **Emacs:** Free and *very* powerful (my preference). Can use for R, Latex, and any other language. Available for Mac, PC, and Linux.

  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 free alternative to Excel (for Windows and Unix):
`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",header=T,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