First Things First: Taking a Look at Your Data

Dr. Michael Fix
mfix@gsu.edu

Georgia State University

24 January 2024

Some "Best Practices" for Research*

1. Everything starts with a question
2. Theory comes first, data next, modeling decisions last
3. Learn to listen to your data
4. Don't model data you don't understand
5. *Good* visualizations are always better
6. Tell a story

Data Types I

- The most basic way to differentiate types of data is to ask: what is it that we are describing?
- Discrete data: things you can count
  - All values are integers
  - These are often counts or categories of things
- Continuous data: things you can measure
  - Values can be fractions of a number
  - These are generally measurements of things

Data Types II: Variable Types

- We can also talk about different types of variables
  - As with discrete vs continuous data, we are still agnostic to things like software at this level of discussion
- Four types of variables:
  1. Nominal
  2. Ordinal
  3. Interval
  4. Ratio

Data Types III: R

- In R, there are six data types:
  1. character
  2. numeric
  3. integer
  4. logical
  5. complex
  6. raw
- Note: the relationship between variable types and R data types is not as clear as one might like. This can (and likely will) cause you headaches at times. However, understanding *both* is vital when working with quant data in R.

Identifying Data Types in R: Core Functions

- There are several core functions to use with data types

```
### DATA TYPES AND STRUCTURES ----

# Core functions to determine data type
class()
typeof()
str()

# Functions to convert data types
as.numeric()
as.integer()
as.character()

# Core functions to determine any object's class
is.numeric()
is.integer()
is.numeric()
is.na()
```

Identifying Data Types in R: Complex and Raw

```
> # Some of these data types will not be something we'll use (like complex and raw)
> my_complex <- c(22-1i)
> class(my_complex)
[1] "complex"
> # Raw type (holds raw bytes, so it is a very unusual data type)
> my_raw <- raw(5)
> print(my_raw)
[1] 00 00 00 00 00
> class(my_raw)
[1] "raw"
```

Identifying Data Types in R: Character

```
> # Character
> x <- "apple"
> class(x)
[1] "character"
> str(x) # prints class and content of the object
 chr "apple"
> # Using double quote marks, 4 becomes a character
> z <- c("apple", "4")
> str(z)
 chr [1:2] "apple" "4"
```

Identifying Data Types in R: Numeric, Integer, Double

```
> # Numeric & Integer & Double
> # When R stores a number in a variable, it converts the number into a “double”
> # value or a decimal type with at least two decimal places. This means that a
> # value such as “1” here, is stored as 1.00 with a type of double and a
> # class of numeric.
> # Double is a real number stored in “double-precision floating point format.”
> typeof(1)
[1] "double"
> class(1)
[1] "numeric"
> # the L tells R to store this as an integer. Many R programmers do not use this
> # mode since every integer value can be represented as a double.
> # An integer can be positive or negative.
> my_int <- 35L
> class(my_int)
[1] "integer"
> # You can convert numeric to integer
> my_num_int <- as.integer(my_num)
> class(my_num_int)
[1] "integer"
> # You can even convert numeric to character
> my_num_character <- as.character(my_num)
> str(my_num_character)
 chr [1:4] "5" "6" "7.1" "8.7"
```

Identifying Data Types in R: Logical

```
> # Logical (TRUE, FALSE, T, F)
> my_logical <- c(TRUE, FALSE, TRUE, TRUE)
> class(my_logical)
[1] "logical"
> my_logical2 <- c(F, F, T, T) # this is also logical
> str(my_logical2)
 logi [1:4] FALSE FALSE TRUE TRUE
> is.logical(my_logical)
[1] TRUE
> # Logical (TRUE, FALSE, T, F)
> my_logical <- c(TRUE, FALSE, TRUE, TRUE)
> class(my_logical)
[1] "logical"
> is.logical(my_logical)
[1] TRUE
> my_logical2 <- c(F, F, T, T) # this is also logical
> str(my_logical2)
 logi [1:4] FALSE FALSE TRUE TRUE
> # Here, I have numbers in my vector, and R will force it to numeric class
> my_logical2 <- c(T, 3, 5, F)
> str(my_logical2)
 num [1:4] 1 3 5 0
```

Goals of Descriptive Statistics

- Simplifying data samples by describing key attributes
- Reading meaningful information out of large lists
- Providing data for use in inference about the population
- Key descriptives focus on measures of center, range, distribution of data

Why Do This?

- As we will see this semester, many of the estimators we use depend upon assumptions about variable types, their distributions, their relationships with each other, etc
- Descriptive stats offer a quick look at individual variables (or sets of variables) to reveal important information
- This can also save us from making major errors when dealing with "canned" data

Key R Commands

```
# Basic functions to use for descriptives ----
summary()  # produces result summaries
mean()     # arithmetic mean
median()   # median
sd()       # standard deviation
table()    # shows frequencies of factor/category variables
var(x)     # (sample) variance
quantile() # quantile
min()      # minimum value
max()      # maximum value
range()    # range with minimum and maximum value

# I recommend using these with sapply() command, for instance
sapply(my_data, mean, na.rm = T) # will procude mean for every
variable in my_data
```

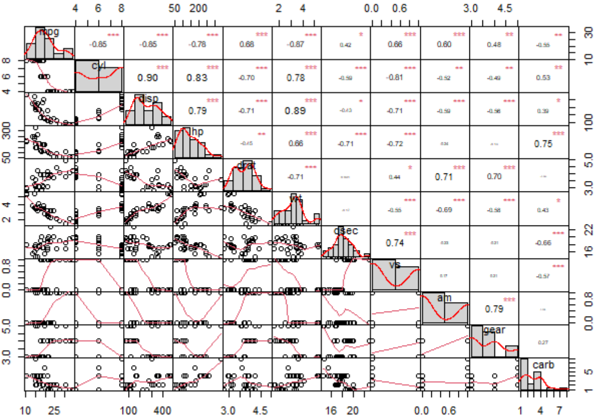
Key R Commands

```r
# There are several packages out there for quick descriptive
stats. I recommend:
library(psych)
describe(mtcars) # mtcars is a built-in data in R
```

```
> describe(mtcars)
     vars  n   mean     sd median trimmed    mad   min    max  range  skew kurtosis    se
mpg     1 32  20.09   6.03  19.20   19.70   5.41 10.40  33.90  23.50  0.61    -0.37  1.07
cyl     2 32   6.19   1.79   6.00    6.23   2.97  4.00   8.00   4.00 -0.17    -1.76  0.32
disp    3 32 230.72 123.94 196.30  222.52 140.48 71.10 472.00 400.90  0.38    -1.21 21.91
hp      4 32 146.69  68.56 123.00  141.19  77.10 52.00 335.00 283.00  0.73    -0.14 12.12
drat    5 32   3.60   0.53   3.70    3.58   0.70  2.76   4.93   2.17  0.27    -0.71  0.09
wt      6 32   3.22   0.98   3.33    3.15   0.77  1.51   5.42   3.91  0.42    -0.02  0.17
qsec    7 32  17.85   1.79  17.71   17.83   1.42 14.50  22.90   8.40  0.37     0.34  0.32
vs      8 32   0.44   0.50   0.00    0.42   0.00  0.00   1.00   1.00  0.24    -2.00  0.09
am      9 32   0.41   0.50   0.00    0.38   0.00  0.00   1.00   1.00  0.36    -1.92  0.09
gear   10 32   3.69   0.74   4.00    3.62   1.48  3.00   5.00   2.00  0.53    -1.07  0.13
carb   11 32   2.81   1.62   2.00    2.65   1.48  1.00   8.00   7.00  1.05     1.26  0.29
```

Key R Commands: Correlation Matrix

```
# For continous variables, I recommend using the correlation
matrix:
library(PerformanceAnalytics)
chart.Correlation(mtcars)
```



Why Graphs?

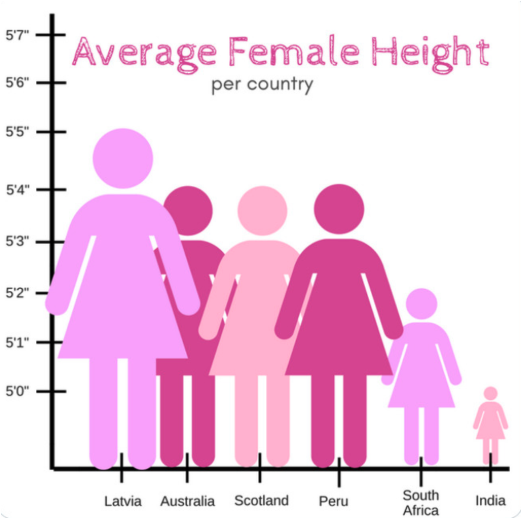
- Good Graphs:
  - Provide clear comparison
  - Are more intuitive to interpret than tables
  - Allow the reader to make an informed decision on the data
  - Can show confidence measures more intuitively than tables
- **Graphs are *always* better than tables.** However, if you have to make a table, here is a guide.

Common Graphing Problems

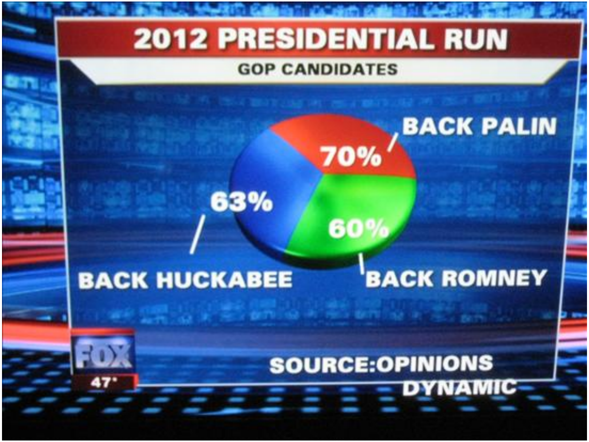
Scaling

- Leaving out the baseline
  - Exaggerates difference between similar numbers
  - Downplays major differences
- Deceptive/meaningless sizes, shapes, and/or scales
- Unclear or poorly labeled axes
- Leaving out corrections (inflation, time, population growth, etc.)
- Deceptive selection of base years

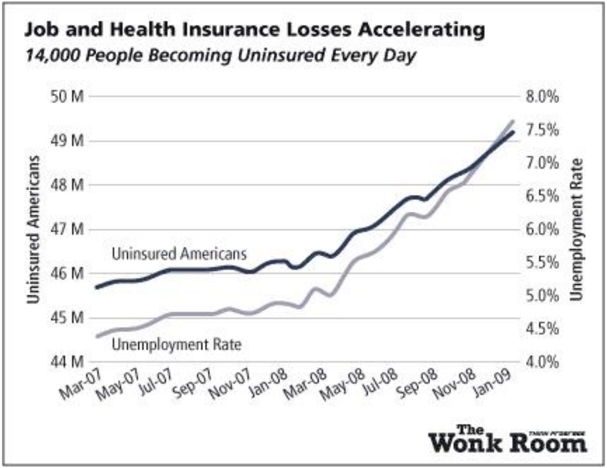
Bad Graphs Are Proliferate!



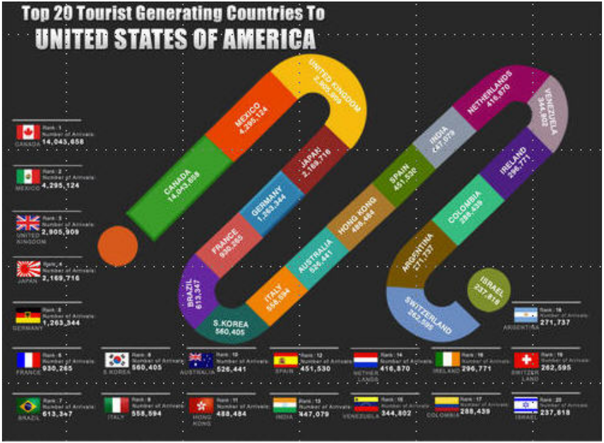
Bad Graphs Are Proliferate!



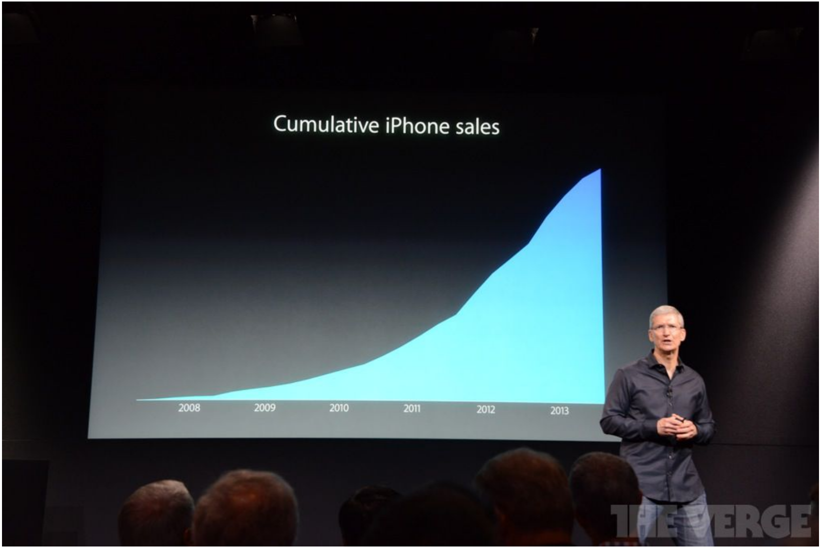
Bad Graphs Are Proliferate!



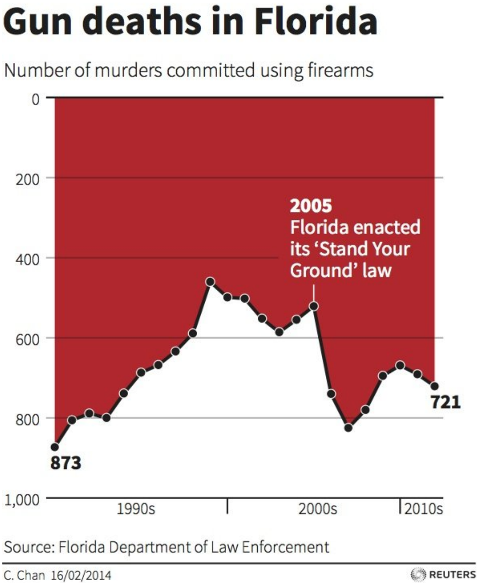
Bad Graphs Are Proliferate!



Bad Graphs Are Proliferate!



Bad Graphs Are Proliferate!



Common Graph Types for Descriptive Statistics

- Line Plots: good for showing changes over time
- Bar Plots: good for comparing discrete data or descriptives of different variables
- Histogram: similar to a bar plot, but for frequency distribution
- Box Plot: good for showing distribution of a variable
- Scatter Plot: useful for showing bivariate relationships

Common Graph Type Examples

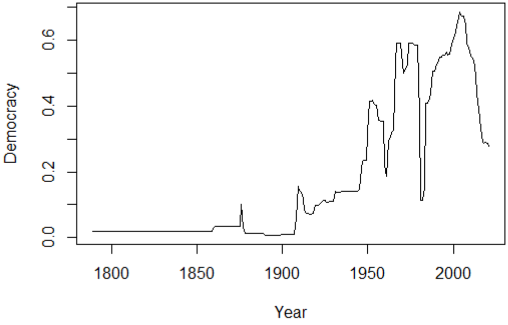
- For these examples, we'll use Varieties of Democracy (V-Dem) dataset

```
### GRAPHS ----
# Load data and make sure you have necessary packages
# We are using Varieties of Democracy version 12
library(tidyverse)
my_data <- readRDS("data/vdem12.rds")

# Let's change names of some of these variables for the sake of simplicity
my_data <- my_data |>
  rename(democracy = v2x_polyarchy,
         regime_type = v2x_regime,
         gdp = e_gdp,
         gdp_per_capita = e_gdppc)
```

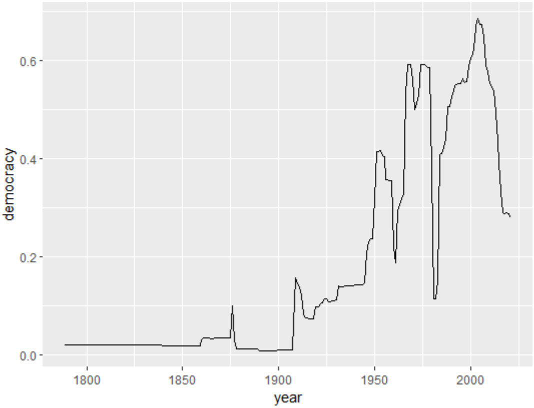
Line Plot in Base R

```
# This is a huge data, so let's focus on just one country here.
# Let's plot Turkey's democracy score over time using base R.
plot(x = my_data[my_data$country_name %in% "Turkey",]$year,
     y = my_data[my_data$country_name %in% "Turkey",]$democracy,
     type = "l",
     xlab = "Year", ylab = "Democracy")
```



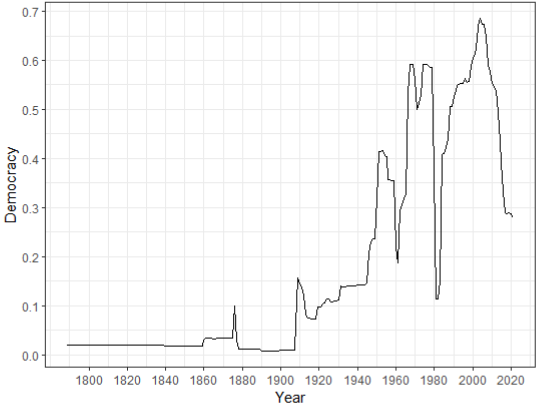
Line Plot in ggplot2

```
# ggplot version
my_data |>
  filter(country_name == "Turkey") |>
  ggplot(aes(x = year, y = democracy)) +
  geom_line()
```



Line Plot in ggplot2, but publishable quality!

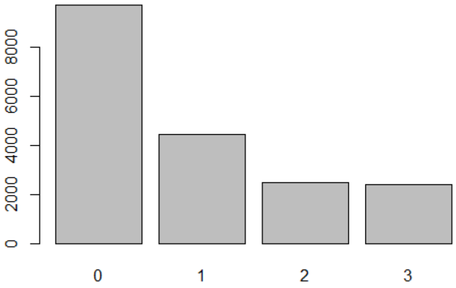
```
# Let's make it pretty
my_data |>
  filter(country_name == "Turkey") |>
  ggplot(aes(x = year, y = democracy)) +
  geom_line() +
  theme_bw() +
  labs(x = "Year", y = "Democracy") +
  scale_x_continuous(breaks = seq(1800, 2020, by = 20)) +
  scale_y_continuous(breaks = seq(0, 1, by = 0.1))
```



Bar Plot in Base R

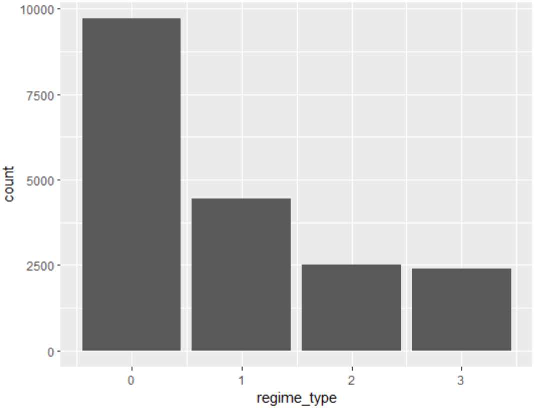
```
# Base R: simple and fast!
str(my_data$regime_type)

plot(as.factor(my_data$regime_type))
```



Bar Plot in ggplot2

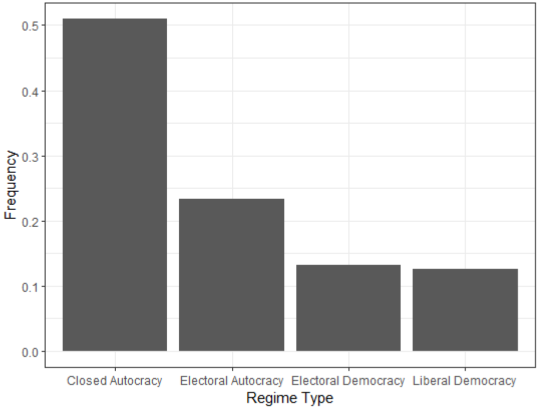
```
# ggplot is also similarly easy!
my_data |>
  ggplot(aes(x = regime_type)) +
  geom_bar()
```



Bar Plot in ggplot2, but publishable quality!

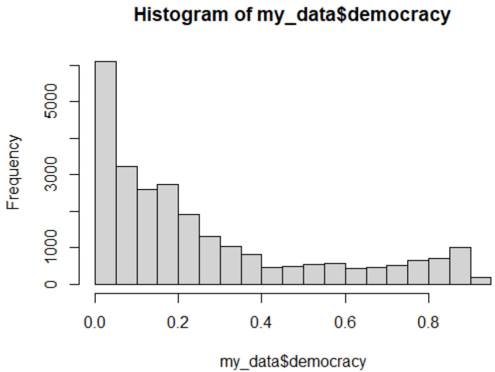
```
# Make it fancy
my_labels <- c("Closed Autocracy", "Electoral Autocracy",
               "Electoral Democracy", "Liberal Democracy")

my_data |>
  filter(!is.na(regime_type)) |>
  ggplot(aes(x = as.factor(regime_type), y = (..count..)/sum(..count..))) +
  geom_bar() +
  theme_bw() +
  labs(x = "Regime Type", y = "Frequency") +
  scale_x_discrete(labels = my_labels)
```



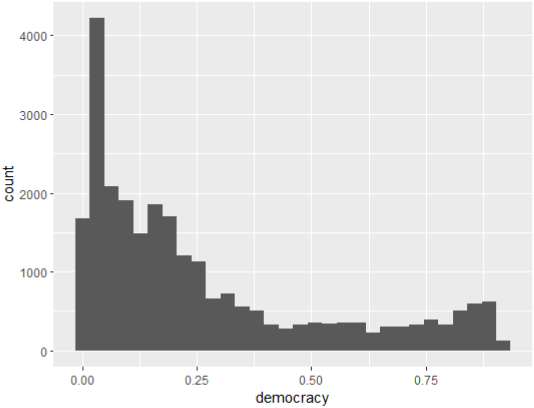
Histogram in Base R

```
# Base R histogram is super simple!
hist(my_data$democracy)
```



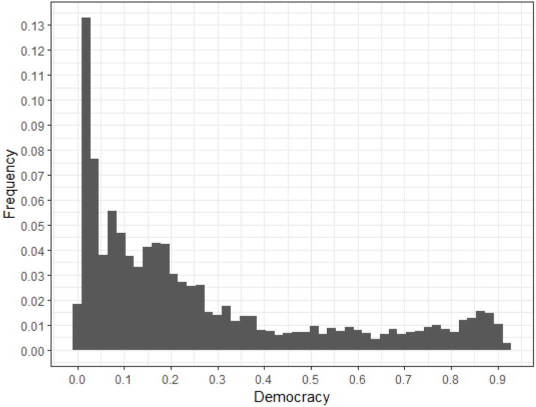
Histogram in ggplot2

```
# ggplot is also easy-to-do!
my_data |>
  ggplot(aes(x = democracy)) +
  geom_histogram()
```



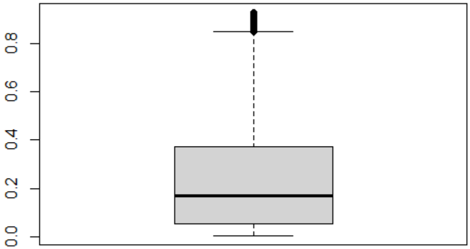
Histogram in ggplot2, but publishable quality!

```
# Let's make it fancy
my_data |>
  ggplot(aes(x = democracy, y = (..count..)/sum(..count..))) +
  geom_histogram(bins = 50) +
  labs(x = "Democracy", y = "Frequency") +
  theme_bw() +
  scale_x_continuous(breaks = seq(0, 1, by = 0.1)) +
  scale_y_continuous(breaks = seq(0, 0.15, by = 0.01))
```



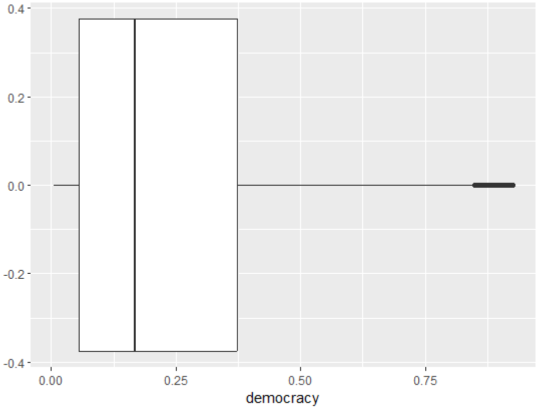
Box Plot in Base R

```
### Box plot ----
boxplot(my_data$democracy)
```



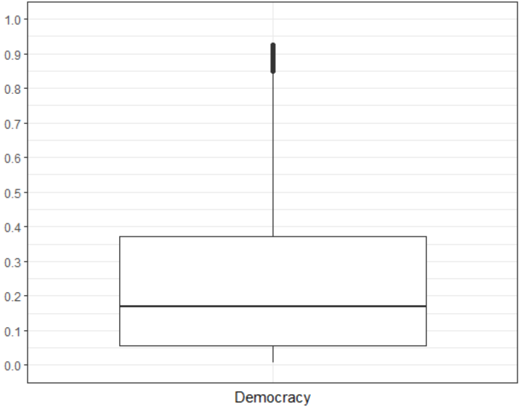
Box Plot in ggplot2

```
# ggplot version
my_data |>
  ggplot(aes(x = democracy)) +
  geom_boxplot()
```



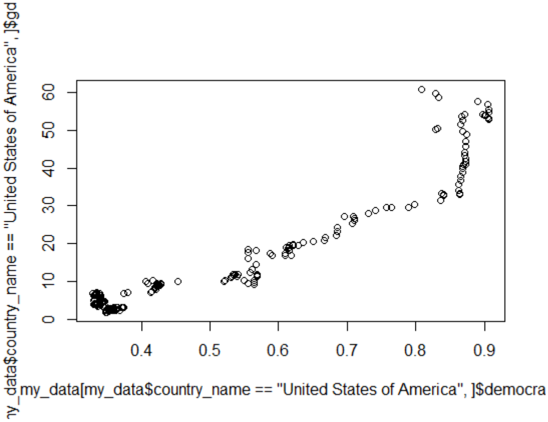
Box Plot in ggplot2, but publishable quality!

```
# Make it fancy!
my_data |>
  ggplot(aes(x = factor(0), y = democracy)) +
  geom_boxplot() +
  labs(x = "Democracy", y = "") +
  theme_bw() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank()) +
  scale_y_continuous(limits = c(0, 1), breaks = seq(0, 1, by = 0.1))
```



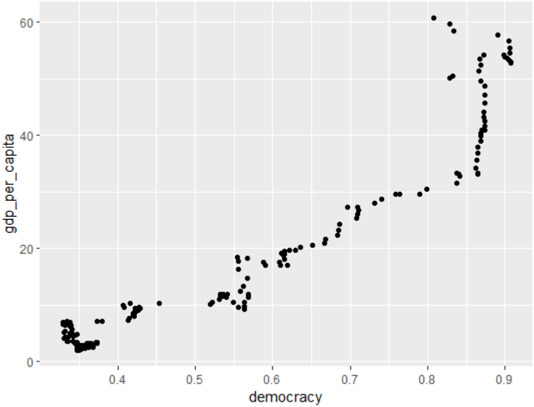
Scatter Plot in Base R

```
# Base R
plot(x = my_data[my_data$country_name == "United States of America",]$democracy,
     y = my_data[my_data$country_name == "United States of America",]$gdp_per_capita)
```



Scatter Plot in ggplot2

```
# ggplot
my_data |>
  filter(country_name == "United States of America") |>
  ggplot(aes(x = democracy, y = gdp_per_capita)) +
  geom_point()
```



Scatter Plot in ggplot2, but publishable quality!

```
# make it fancy
my_data |>
  filter(country_name == "United States of America") |>
  ggplot(aes(x = democracy, y = gdp_per_capita)) +
  geom_point() +
  theme_bw() +
  scale_x_continuous(breaks = seq(0, 1, by = 0.1)) +
  scale_y_continuous(breaks = seq(0, 60, by = 10)) +
  labs(x = "Democracy", y = "GDP per capita") +
  geom_smooth(method = lm)
```