# Frequency Distributions, Central Tendency

## Preamble

```
rm(list=ls())
setwd("C:/Users/19107/Desktop/R Stuff/2023 or Earlier/Recreation")
```

```
library(ggplot2) #This is a visualization package, VERY USEFUL (*)
library(states)  #This is a package which contains data sets on States
library(dplyr)   #This is a package for useful data manipulation
 ↪  commands (*)

# (*)these packages needn't be understood fully at this time but we will
 ↪  cover them later.
```

### A short tangent into the use of sink()

sink() is a function that allows you to record your console output in a .txt document it is relatively simple, you specify a file name when first call sink e.g. sink("myfile.txt"), this "opens the connection" between the file and R's console output, it then records all output until the user calls sink() which closes the connection. NOTE: R WILL NOT DISPLAY OUTPUTS WHILE SINK IS ACTIVE. As an illustration I will use sink() to produce an example output from this script.

```
sink("example_sink.txt")
```

## Getting Started

Once data has been collected we are in a position to start analysis, one of the most basic ways we analyze our data is through the use of frequency tables and visualization.

Let's make use of some pre-fabricated data from the "states" package and from base R's data on U.S. states.

```
usa=as.data.frame(state.x77) #data on U.S. states
states=cowstates # data on countries in international system
polity=polity # polity data on level of democracy

#Our polity and states data are separate which we do not want so let's
 ↪  "merge" them (or "join"). I won't explain too much of this process
 ↪  today but in future lessons you will learn plenty about merging.

# Both databases have a common identifier (COW code) but they are named
 ↪  differently which is a problem so I'll make a new variable #with the
 ↪  appropriate name (you can also just use rename but I don't like
 ↪  rename()'s syntax)

states$ccode=states$cowcode

# Next I just use dplyr's "inner_join" command and bingo the two sets
 ↪  are now one.
states=inner_join(states,polity,by="ccode")
```

```
               country_name      start        end microstate ccode year polity
1  United States of America 1816-01-01 9999-12-31      FALSE     2 1800      4
2  United States of America 1816-01-01 9999-12-31      FALSE     2 1801      4
3  United States of America 1816-01-01 9999-12-31      FALSE     2 1802      4
4  United States of America 1816-01-01 9999-12-31      FALSE     2 1803      4
5  United States of America 1816-01-01 9999-12-31      FALSE     2 1804      4
6  United States of America 1816-01-01 9999-12-31      FALSE     2 1805      4
7  United States of America 1816-01-01 9999-12-31      FALSE     2 1806      4
8  United States of America 1816-01-01 9999-12-31      FALSE     2 1807      4
9  United States of America 1816-01-01 9999-12-31      FALSE     2 1808      4
10 United States of America 1816-01-01 9999-12-31      FALSE     2 1809      9
```

Now that we have 'collected' our data we can start analyzing

## Frequency Distributions

As Frankfort-Nachmias says in CH.15 frequency tables are tables which detail the frequency of each level of a given variable. In more simple terms a frequency table tells you how common each value of a variable is. For example for a 0/1 variable the frequency table will tell you how many 1s and how many 0s you have (although your mean can also tell you that). This is most useful for variables which have more than 2 values but less than 10, we will still use it for Polity which has 21 valid values and an additional 3 'coder' values.

### Using "Table" command

Table is a base R function with really simple uses, as its name indicates it calculates a "table" of frequencies of EACH level in a variable or set of variables

#### An IR example

```
table(states$polity, dnn = "Polity") #dnn is a vector of names for the
 ↪  dimensions of your table
```

```
Polity
 -88  -77  -66  -10   -9   -8   -7   -6   -5   -4   -3   -2   -1    0    1    2
 410  264  301 1505 1459  594 2073 1492  579  874 1465  342  538  180  462  475
   3    4    5    6    7    8    9   10
 314  561  456  640  670  908  735 3226
```

so that's nice but what if we want things to be nicer looking? Go to the data.frame!

```
df=as.data.frame(table(states$polity)) #this transforms our table into a
 ↪  dataframe
colnames(df)=c("Polity", "Freq") #this command let's me rename the
 ↪  columns i.e. variables
df$Percent=(df$Freq/sum(df$Freq))*100 # this command is making a new
 ↪  variable which is a percent form of the Freq
df
```

```
   Polity Freq    Percent
1     -88  410  1.9977586
2     -77  264  1.2863616
```

```
3      -66  301  1.4666472
4      -10 1505  7.3332359
5       -9 1459  7.1090971
6       -8  594  2.8943137
7       -7 2073 10.1008624
8       -6 1492  7.2698923
9       -5  579  2.8212250
10      -4  874  4.2586367
11      -3 1465  7.1383326
12      -2  342  1.6664230
13      -1  538  2.6214491
14       0  180  0.8770648
15       1  462  2.2511329
16       2  475  2.3144764
17       3  314  1.5299907
18       4  561  2.7335185
19       5  456  2.2218974
20       6  640  3.1184525
21       7  670  3.2646299
22       8  908  4.4243044
23       9  735  3.5813478
24      10 3226 15.7189495
```

```r
#side note: I don't like how long the decimals are going so I'll use
 ↪  round to simplify things
df$Percent=round(df$Percent, digits=2)
df
```

```
   Polity Freq Percent
1     -88  410    2.00
2     -77  264    1.29
3     -66  301    1.47
4     -10 1505    7.33
5      -9 1459    7.11
6      -8  594    2.89
7      -7 2073   10.10
8      -6 1492    7.27
9      -5  579    2.82
10     -4  874    4.26
11     -3 1465    7.14
12     -2  342    1.67
```

```
13      -1   538    2.62
14       0   180    0.88
15       1   462    2.25
16       2   475    2.31
17       3   314    1.53
18       4   561    2.73
19       5   456    2.22
20       6   640    3.12
21       7   670    3.26
22       8   908    4.42
23       9   735    3.58
24      10  3226   15.72
```

much better!

## An Americanist example

Table can also be fed logical arguments. Say I wanted to know how many states have an average Income greater than the national average, I simply tell table to evaluate that logical statement.

```r
table(usa$Income>mean(usa$Income), dnn = "Income Above Mean")
```

```
Income Above Mean
FALSE  TRUE 
   21    29 
```

So 29 states are above the national average and 21 are below.

We can take it further too, say I wanted to know A)how many were a full standard deviation above the mean or B)how many are over 1 standard deviation but less than 2

```r
#A)
table(usa$Income>(mean(usa$Income)+sd(usa$Income)))
```

```

FALSE  TRUE 
   42     8 
```

```
#B)
table(usa$Income>(mean(usa$Income)+sd(usa$Income))&usa$Income<(mean(usa$Income)+2*sd(usa$I
```

```
FALSE  TRUE 
   43     7 
```

As you can see there's quite a lot of possibilities here, learning how to leverage logical statements in your analysis and coding is very useful. But what if tables of numbers just aren't your thing??

## Using Graphs

Graphs can tell us many of the things that Tables do and sometimes more. I am only going to focus on two types of graphs today, densities and histograms These both show the same info just a little differently. Density graphs are a form of line graph that plots frequency across the range of values but it is best suited to variables with a high number of levels and especially continuous variables. Conversely, histograms are a subset of bar graphs and are better suited to variables with a low number of levels and especially categorical or ordinal variables.

### Densities and Histograms

I'll briefly explain what we see here, but again future lessons will tackle plots in greater detail.

ggplot() is where I tell R what dataframe I'm plotting

geom_ is where I tell R what sort of plot I'd like with geom_

aes() is where I specify the variables I'm plotting

```
#Density
ggplot(states)+geom_density(aes(as.numeric(polity)))
```

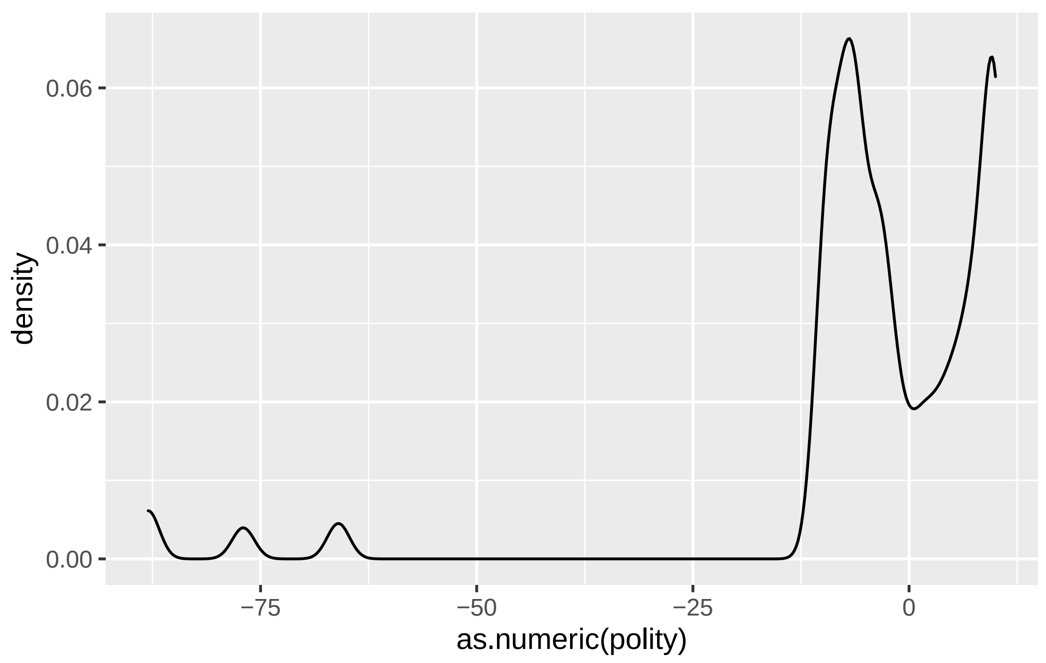

Figure 1: A density of Polity across states

```
#Histogram
ggplot(states)+geom_histogram(aes(as.numeric(polity)))
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

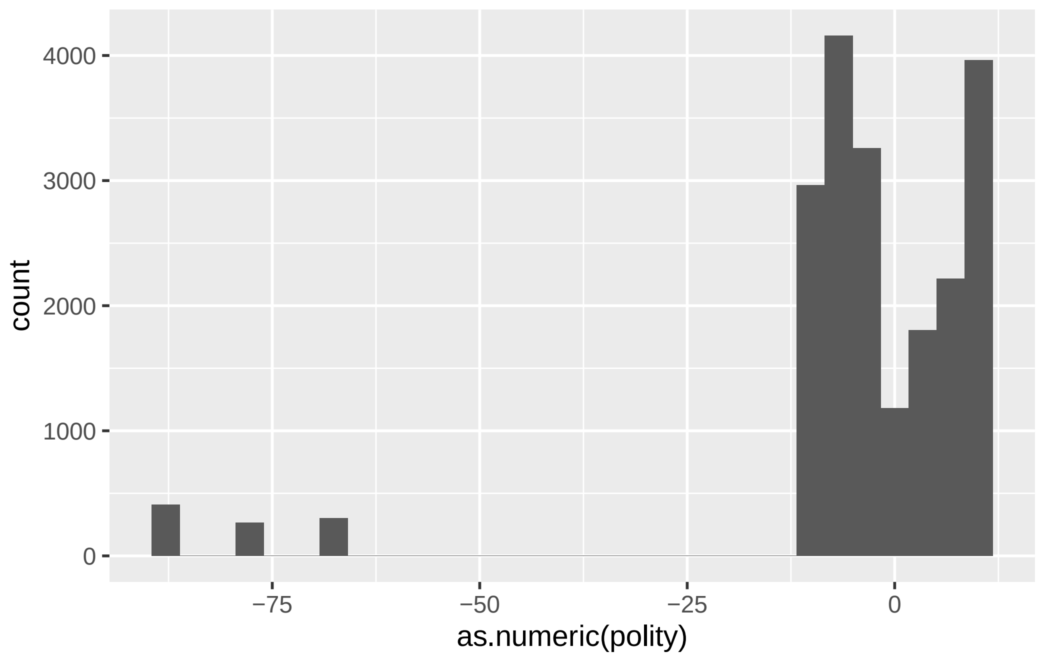

Figure 2: A histogram of Polity across states

These plots still leave much to be desired but in future lessons we'll work on ways to improve the appearance and informativeness of our plots.

## Measures of Central Tendency

### Mode

Remember there is no statistical mode function in base R

```
getmode <- function(v) {
  v=subset(v,is.na(v)==FALSE)
  uniqv <- unique(v, na.rm=TRUE)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}
getmode(states$polity)
```

```
[1] 10
```

So ‘perfect’ democracy (Polity = 10) is the modal category among states across time.

##Median

```
median(usa$Area)
```

```
[1] 54277
```

The Median size of a U.S. state is 54,277 square miles

##Mean

```
mean(usa$Area)
```

```
[1] 70735.88
```

The average area of a U.S. state is 70735.88 square miles

### Compare the mean and median in the data

```
median(usa$Area)==mean(usa$Area) # Symmetric/Unskewed
```

```
[1] FALSE
```

```
median(usa$Area)>mean(usa$Area) # Left/Negative Skew
```

```
[1] FALSE
```

```
median(usa$Area)<mean(usa$Area) # Right/Positive Skew
```

```
[1] TRUE
```

So the distribution of area of U.S. states is skewed right/positive.

## Measures of Dispersion Simple/Mean Oriented

### Simple

#### Range

The largest and smallest values of a set

```
range(usa$Population)
```

```
[1]   365 21198
```

#### Interquartile range

The distance between the 1st and 3rd quartiles

```
IQR(usa$Population)
```

```
[1] 3889
```

### Mean Oriented

#### Standard Deviation

How distant from the mean do the data tend to be

```
sd=sd(usa$Population)
sd
```

```
[1] 4464.491
```

### Coefficient of Variation

A ratio of standard deviation to mean

```
mean=mean(usa$Population)
sd/mean
```

```
[1] 1.051354
```

Usually 1 or greater is considered dispersed but there are no universal rules.

## The Normal Curve

We didn't get to the Normal Distribution, but here you can look at it.

```
set.seed(123456)
ggplot()+geom_density(aes(rnorm(500000)))
```

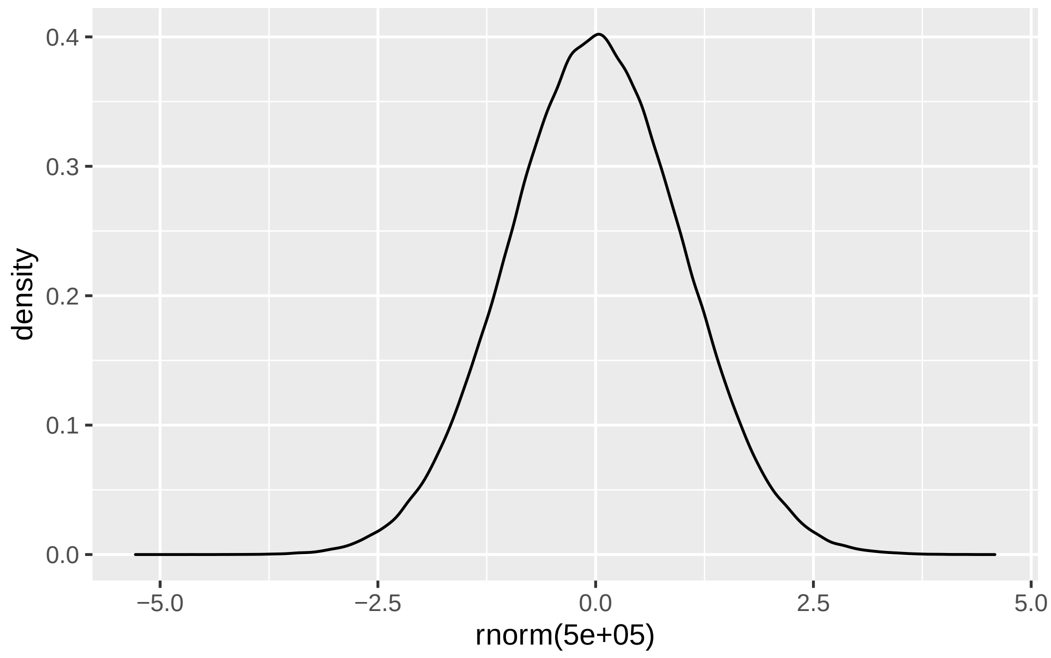

Figure 3: A randomly generated normal distribution

```
# Close my Sink connection
sink()
```