Introduction to R (with a tidyverse focus)

Matthew Sisk msisk1@nd.edu & James Ng, PhD james.ng@nd.edu

Navari Family Center for Digital Scholarship, University of Notre Dame

The Basics of using R

First, you should download and install R (https://cran.rstudio.com/) and RStudio (https://rstudio.com/products/rstudio/download/) if you have not done so yet. Both should work on any of the major opperating systems and be fairly easy to install

If it absolutly does not work, you can use RStudio in the cloud! Sign up with google or github account at rstudio.cloud.

Once the software is all set up, you should run RStudio.

The first thing we will want to do is load the tidyverse package. This brings in a lot of the newer ways of manipulating data in R. all external libraries (called packages) are brought in with the *library()* command. If you have never installed it, you will need to run *install.packages("tidyverse")* first.

```
library(tidyverse)
```

```
## -- Attaching packages -----------------------------------------------------------------------------------
## v ggplot2 3.2.0     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
## -- Conflicts ---------------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
## you may need to install tidyverse first:
# install.packages('tidyverse')
```

Then we will want to clear out any existing data and set our working directory for any files we want to load or save. Note: your WD will be different and on windows computers you should use two backslashes between parts of the location

```
rm(list=ls())
# setwd("G:\\My Drive\\Teaching\\Datasets")
```

Then you are ready to load some data

Part 1: Loading Data

R has many baked-in example datasets. You can see a list of what is available in your version of R by running the command *data()* in your console

Let's load one of these datasets, mpg. This is Fuel economy data from 1999 and 2008 for 38 popular models of car. More info: https://ggplot2.tidyverse.org/reference/mpg.html Here, we assign the data as a *data.frame()* for ease. It is a tibble, which is the next iteration of data in R, but the support in RSudio is not yet as good for tibbles

```
datauto <- data.frame(mpg)
```

Here, we have taken the built-in mpg dataset and made a copy of it into a local object called *datauto*. The arrow <- is how we assign values in R. An = also works in most cases. We can assign other variables the same way

```
twice <- 2
animal <- "goat"
moods <- c("happy", "sad", "indifferent")
```

The c() syntax is how R knows something is a list of values

Alternatively, you can load many other file types directly into R. The newer way of doing this is with the readr package (which comes as part of the tidyverse). you can read comma-separated values (csv) files with read_csv

```
census.data <- read_csv("State_ACS2016.csv")
```

```
## Parsed with column specification:
## cols(
##   Geo_NAME = col_character(),
##   `Total Population` = col_double(),
##   `Population Density (Per Sq. Mile)` = col_double(),
##   `Area (Land)` = col_double(),
##   Male = col_double(),
##   Female = col_double(),
##   `Median Household Income` = col_double(),
##   `Median Year Structure Built` = col_double(),
##   `Median House Value` = col_double(),
##   `Median Gross Rent` = col_double(),
##   `Number of Familites:` = col_double(),
##   `Families with Income Below Poverty Level` = col_double(),
##   `Average Commute to Work` = col_double(),
##   `Foreign Born Population` = col_double()
## )
```

Alternatively, you import excel files directly with read_exel, though you need to import the *readxl* package and specify the sheet.

```
library(readxl)
prisons <-  read_excel("US_Prisons.xls", sheet="US_Prisons")
```

In the Environment section of RStudio you can see the differences between values and the data we loaded. The datauto object is a data frame, which is the R version of a table. A datafame has a series of observations (rows) and variables (columns)

#2. Summarizing Data There are a number of different ways of getting a sense of your data in R. Here are a few of them

To get the data frame dimensions and store them in new objects.

```
dim(datauto)
```

```
## [1] 234  11
```

```
n.obs <- dim(datauto)[1]
n.cols <- dim(datauto)[2]
```

To get a sense of the data use *glimpse()*.

```
glimpse(datauto)
```

To return first few rows use *head()*

```r
head(datauto)
```

To get basic summary stats use *summary()*

```r
summary(datauto)
```

We can reference individual rows, colums and records within the data frame. To reference an individual column use the name of the dataframe $ and the name of the column

```r
mean(datauto$cty)
```

```
## [1] 16.85897
```

You can also reference columns or rows by number or name within brackets. The row number/name comes first, followed by a comma, then the column number/name. If you are referencing

```r
print(datauto[4,"trans"])
```

```
## [1] "auto(av)"
```

```r
datauto[4,]
```

```
##   manufacturer model displ year cyl    trans drv cty hwy fl   class
## 4         audi    a4     2 2008   4 auto(av)   f  21  30  p compact
```

```r
max(datauto[,c("hwy","cty")])
```

```
## [1] 44
```

##3. Manipulating Data

The command *unique()* will give all of the possible values

```r
unique( datauto[, 'manufacturer'] )
```

```
##  [1] "audi"       "chevrolet"  "dodge"      "ford"       "honda"
##  [6] "hyundai"    "jeep"       "land rover" "lincoln"    "mercury"
## [11] "nissan"     "pontiac"    "subaru"     "toyota"     "volkswagen"
```

```r
# equivalently:
# unique( datauto$manufacturer )
```

Reassigning columns is done the same way variables are assigned. Here, the variable *class* is categorical so let's factorize it. This basically makes it so there is not as much repetative text stored in the file and analyses run faster. Summary() is now more useful when applied to a factor.

```r
datauto$f.class <- as.factor(datauto$class)
summary(datauto$f.class)
```

```
##    2seater    compact    midsize    minivan     pickup subcompact
##          5         47         41         11         33         35
##        suv
##         62
```

We can also sort and arrange our data off of any field. Here we sort by manufacturer, model and year.

```r
datauto <- arrange(datauto, manufacturer, model, year)
```

One of the newest innovations in R is the tidyverse, which relys on pipes (the %>% symbol) to string together multiple commands. This is part of the dplyr package (which comes with the *library(tidyverse)* command earlier) So here, if we want to know how many models each manufactor, we first specify the data, then pipe that data to the group_by() command and then pipe that result to the summarize() command which creates a new variable *n.models* by counting the number of distinct values in the model column

```r
datauto %>%
  group_by(manufacturer) %>%
  summarize( n.models = n_distinct(model) )
```

We can even group_by multiple columns

```r
datauto %>%
  group_by(manufacturer, year) %>%
  summarize( n.models = n_distinct(model) )
```

Filter() does something similar, but makes a new data frame with only those manufactors with on model retained

```r
datauto %>%
  group_by(manufacturer, year) %>%
  filter( n_distinct(model) == 1 )
```

The results of these sorts of dplyr manipulations can be saved as new data frames for further use later, like here with the number of each class of vehical produced each year.

```r
classcounts <- datauto %>% group_by(f.class, year) %>% summarize(n = n())
```

or computing mean cty, mean hwy and number of vehicles by class

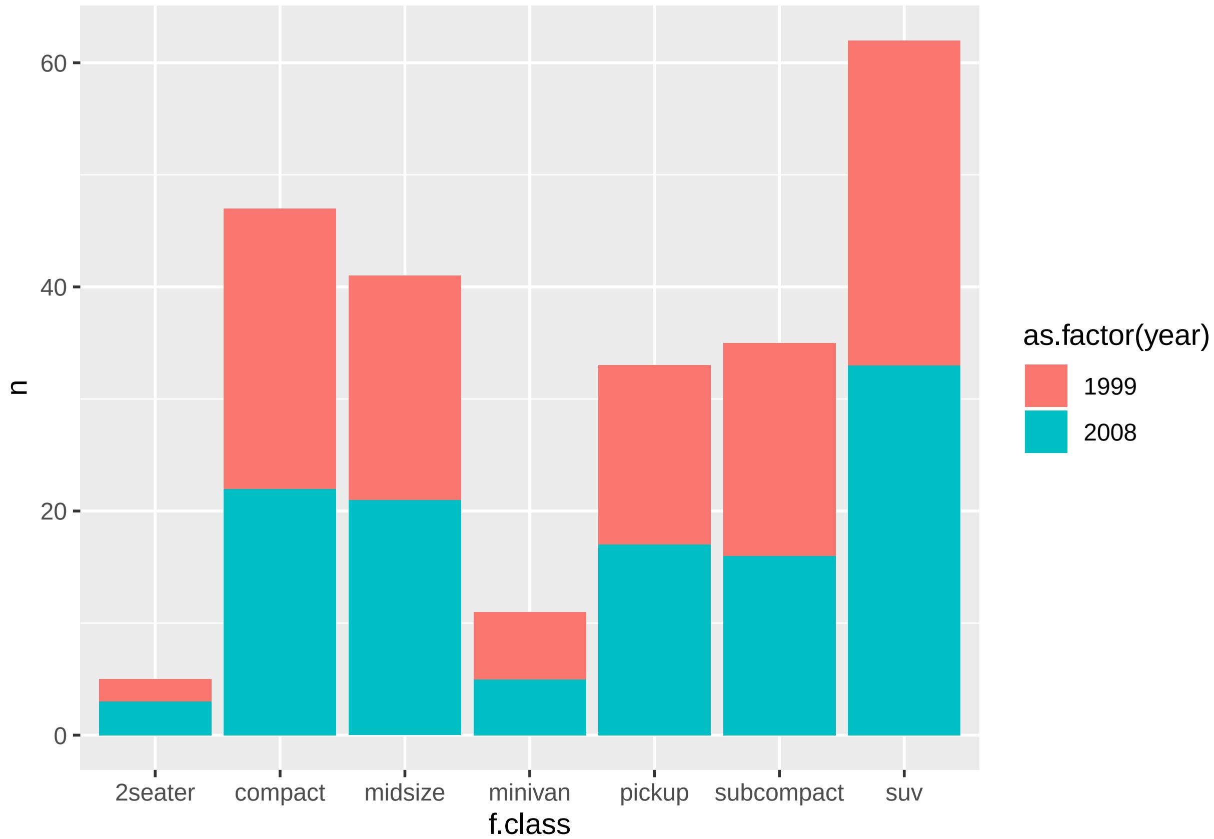
```r
datauto.means <- datauto %>%
  group_by(f.class) %>%
  summarize(mean.cty = mean(cty), mean.hwy = mean(hwy), n = n() )
datauto.means
```

```
## # A tibble: 7 x 4
##   f.class    mean.cty mean.hwy     n
##   <fct>         <dbl>    <dbl> <int>
## 1 2seater        15.4     24.8     5
## 2 compact        20.1     28.3    47
## 3 midsize        18.8     27.3    41
## 4 minivan        15.8     22.4    11
## 5 pickup         13       16.9    33
## 6 subcompact     20.4     28.1    35
## 7 suv            13.5     18.1    62
```

#4 Plotting Data

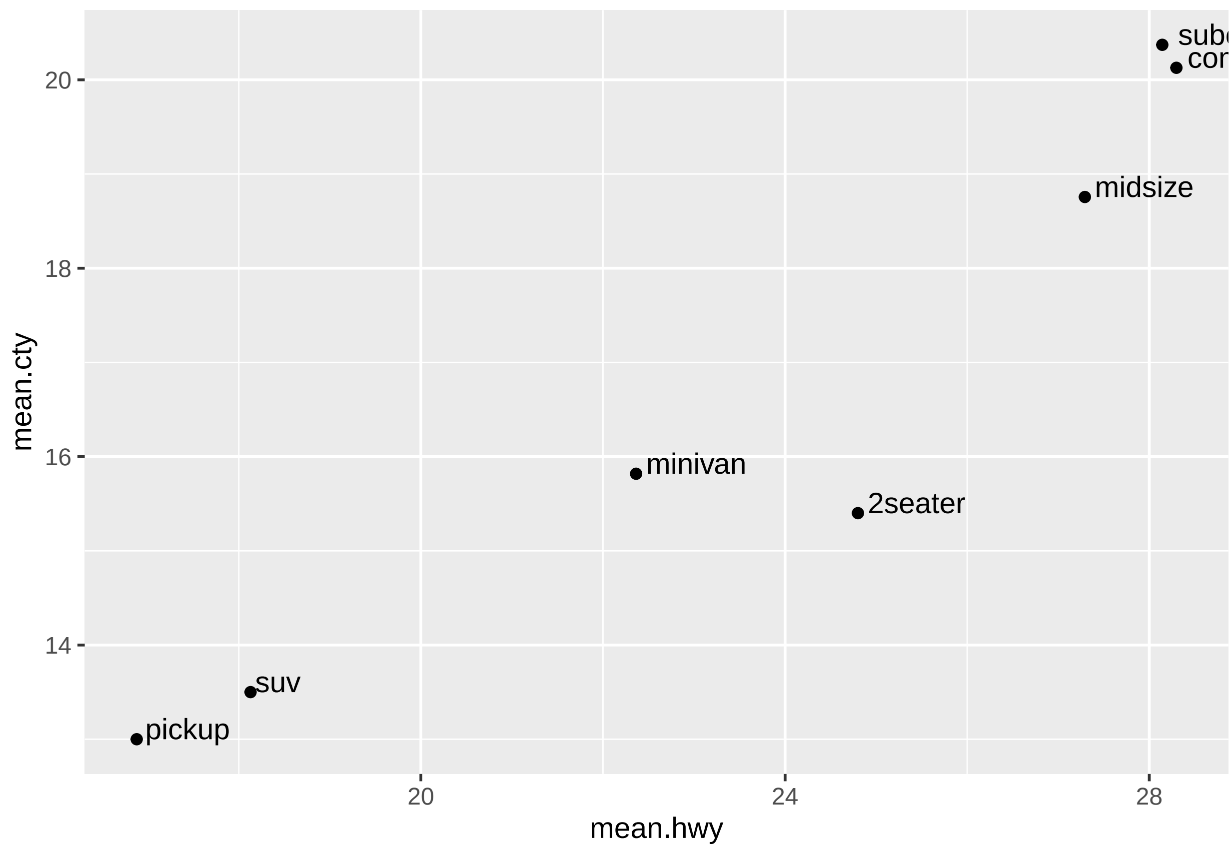
The main plotting engine for the tidyverse is a library called ggplot. It also builds thing incrimentally, but instead of a pipe, it uses a plus sign to string together the components of the plot. Basic syntax for a ggplot is to define a ggplot entity with ggplot(), define the data you areusing, and then use the aes() to assign the x and y axes, and if color or shape mean anything. The type of chart is then added on with a *geom_** function. Here we are using a geom_bar() with the stat of identity (the count of values)

```r
ggplot(dat=classcounts, aes(y=n, x=f.class, fill=as.factor(year))) +
  geom_bar(stat="identity")
```



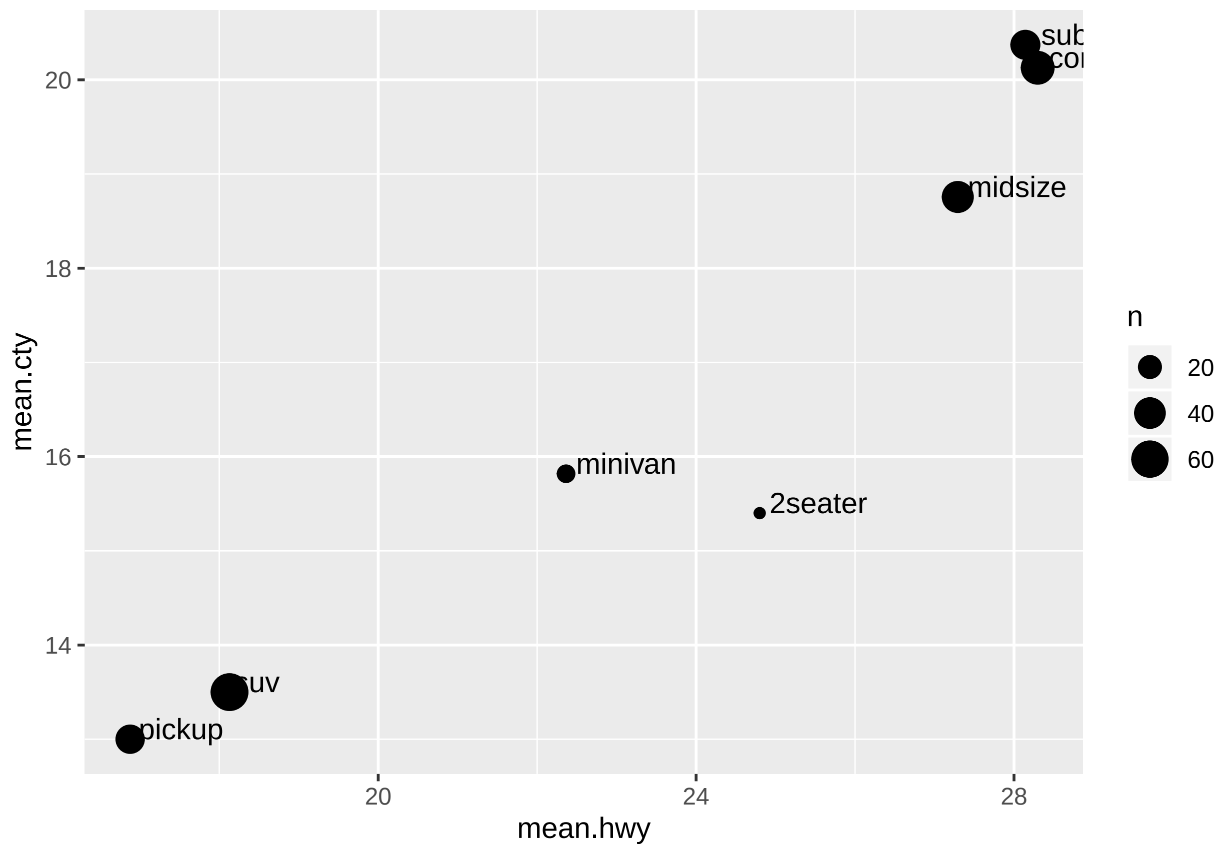
You can also assign a ggplot item to a variable. This will allow you to incrimentally add things onto it.

```
plot1 <- ggplot(data=datauto.means, aes(x = mean.hwy, y = mean.cty)) +
  geom_point()+
  geom_text(aes(label=f.class),hjust=-.1, vjust=0)
plot1
```



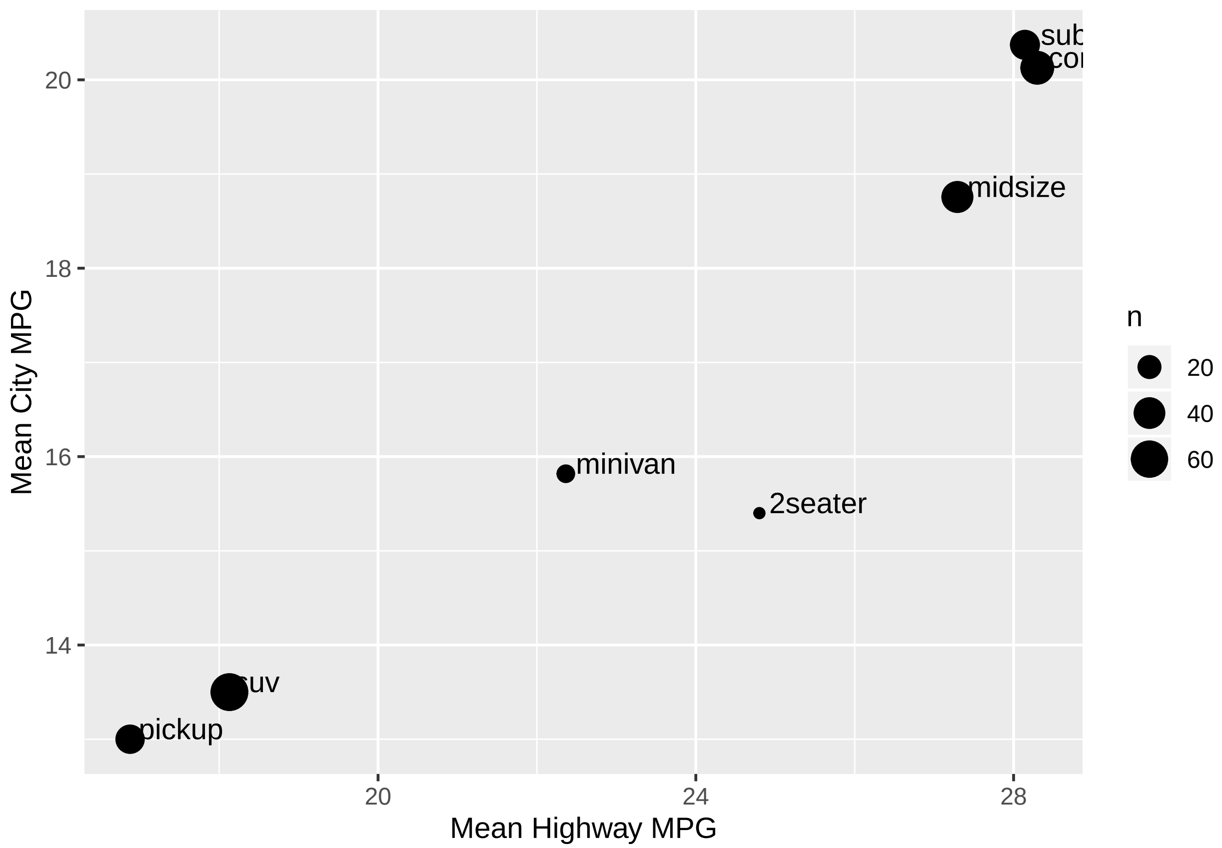
Vary point size by number

```
plot1 <- plot1 + geom_point(aes(size=n))
plot1
```



Change the labels

```
plot1 <- plot1 + xlab("Mean Highway MPG" ) +
  ylab("Mean City MPG")
plot1
```



#5. Analysis Find correlation between average combined cty and hwy mpg in 1999 vs 2008. To do this, need to 'spread' (reshape) the data.

```
# first get average combined cty and hwy, call it mpg
datauto %>% mutate( mpg = (cty+hwy)/2 )
```

```
# then summarize by manufacturer, model, year
tempdat <- datauto %>% mutate(mpg = (cty+hwy)/2) %>%
  group_by(manufacturer, model, year) %>%
  summarize( mpg=mean(mpg) )
tempdat
```

```
# spread mpg into mpg_1999 and mpg_2008
tempdat %>% spread( key=year, value=mpg)
```

```
# finally, rename 1999 and 2008 columns
dat.mpg.wide <- tempdat %>% spread( key=year, value=mpg) %>%
  rename( mpg_1999 = `1999`, mpg_2008 = `2008` )
dat.mpg.wide
```

Linear Model: What's the relationship between fuel economy in 1999 and fuel economy in 2008?

```
modl <- lm(formula = mpg_2008 ~ mpg_1999, data=dat.mpg.wide)
summary(modl)
```

```
##
## Call:
## lm(formula = mpg_2008 ~ mpg_1999, data = dat.mpg.wide)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.0981 -0.5320 -0.0038  0.7256  2.8691
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.77916    1.05927    1.68    0.102
## mpg_1999     0.93669    0.05267   17.79   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.523 on 36 degrees of freedom
## Multiple R-squared:  0.8978, Adjusted R-squared:  0.895
## F-statistic: 316.3 on 1 and 36 DF,  p-value: < 2.2e-16
```

Want to learn more? Two great resources:

R for Data Science by Hadley Wickham

Programming with R by Software Carpentry