Walkthrough Guide - Visualizations and Reporting in R

Contents

1 Introduction

Welcome to Visualizations and Reporting in R! Today we are going to cover how to create visualizations and reports in R but with a focus on Institutional Research type data.

1.1 Installing R/RStudio

By now you should have R and RStudio installed on your computer. Just in case here is some information on how to install these two free programs:

Installing the latest version of R (3.4.2) and RStudio (1.1.383) - a powerful user interface for R - on your computer.

R is available here: http://cran.r-project.org/

RStudio is available here: http://www.rstudio.com/products/rstudio/download/ + RStudio Support - https://support.rstudio.com/hc/en-us/categories/200035113-Documentation + More RStudio shortcuts - https://support.rstudio.com/hc/en-us/articles/200711853-Keyboard-Shortcuts

1.2 Installing Packages

This workshop will rely heavily on the use of the `tidyverse` a package of very useful packages. A package is a collection of code and functions written for the R language. They usually focus on a specific task or problem and most of the useful R applications appear in packages.

First we need to install the `tidyverse` package using the `install.packages()` function. The package name goes inside of the parentheses in double quotes: ' "tidyverse" '. You only have to do this once and you should be connected to the internet.

```
#install.packages("tidyverse")
```

Now the specific package is on your hard drive.

Once a package is installed, any time we start a new R session and we want to use functions inside of that package, we will need to load the package with the `library()` function.

```
library(tidyverse)
```

Now the specific package is in your R session.

For more information about tidyverse go here: https://www.tidyverse.org/

I will go through specific packages in `tidyverse` throughout this guide.

Great! Now our R session should be ready to go! Let's load some data.

2 Data

We are going to be working with everyone's favorite public data source IPEDS! Specifically, we are going analyze eight years of the Admissions Survey. The data that we are going to be working with is a product of a side project that I am working on with Emma Morgan from Tufts and my colleague Kathy Foley. Our goal is to be able to provide the community longitudinal and accessible IPEDS data and processed in R. We are still polishing code but more information about this project can be found here: https://github.com/emmamorgan-tufts/IPEDS_longitudinal

2.1 Read in Data

First, we must read in the dataset. There are many options including the Import Dataset button in RStudio, found in the Environment pane.

Or you can use R `read.csv()` function with the `file.choose()` function to pick the file that you want to open. The file you are looking for is "IPEDS_admissions_subset.csv".

```
ipeds_adm <- read.csv(file.choose(), stringsAsFactors = F, check.names = F)
```

In this line of code, we are also telling R to not read variables with strings as factors. Ever wonder why? Here is a fun article about the history of `stringsAsFactors` https://simplystatistics.org/2015/07/24/stringsasfactors-an-unauthorized-biography/

We are also setting `check.names` to `FALSE` so that our data variable names come in cleanly.

2.2 Explore the Data

Now that the data is loaded I like to run three lines to make sure everything read in properly:

1. `names()`
    - variable names
    - check that all variables were read in
2. `str()`
    - structure of your data and variable types
    - check that your variables are the right types (e.g. character, integer, factor)
3. `summary()`
    - to see a quick distribution of each variable
    - check for outliers or other weirdness

This is a large dataset so I have muted the output but here are those three functions:

```
names(ipeds_adm)
str(ipeds_adm)
summary(ipeds_adm)
```

For this workshop we are going to focus on public institutions from Massachusetts, New York, and New Jersey. Before filtering this data, it is always a good idea to look at the variables that we might filter on.

```
table(ipeds_adm$`Control of institution`)
```

```
##
## Private not-for-profit                    Public
```

```
##                     8405                      4278
```

```r
table(ipeds_adm$`State abbreviation`)
```

```
## 
##               Alabama                Alaska               Arizona 
##                   189                    16                    63 
##              Arkansas            California              Colorado 
##                   146                   761                   154 
##           Connecticut              Delaware District of Columbia 
##                   180                    32                    56 
##               Florida               Georgia                  Guam 
##                   434                   347                     7 
##                Hawaii                 Idaho              Illinois 
##                    61                    56                   476 
##               Indiana                  Iowa                Kansas 
##                   344                   253                   183 
##              Kentucky             Louisiana                 Maine 
##                   208                   180                   116 
##              Maryland         Massachusetts              Michigan 
##                   207                   562                   339 
##             Minnesota           Mississippi              Missouri 
##                   286                   117                   337 
##               Montana              Nebraska                Nevada 
##                    52                   135                    30 
##         New Hampshire            New Jersey            New Mexico 
##                    90                   229                    43 
##              New York        North Carolina          North Dakota 
##                  1084                   441                    56 
##                  Ohio              Oklahoma                Oregon 
##                   477                   163                   183 
##          Pennsylvania           Puerto Rico          Rhode Island 
##                   953                   265                    73 
##        South Carolina          South Dakota             Tennessee 
##                   224                    88                   320 
##                 Texas                  Utah               Vermont 
##                   552                    45                   135 
##        Virgin Islands              Virginia            Washington 
##                     8                   323                   167 
##         West Virginia             Wisconsin               Wyoming 
##                   147                   282                     8
```

2.3 Data Wrangling

We'll use the package `dplyr` the data wrangling package in `tidyverse`.

The package contains six main functions for wrangling data:

Function	Description
`filter()`	keep rows matching criteria
`select()`	pick columns by name
`arrange()`	reorder rows
`mutate()`	add new variables
`summarise()`	reduce variables to values
`group_by()`	group data into rows with the same value

You can find a cheat sheet for `dplyr` here.

Right now we will just use the `filter` function but will revisit the others later on.

```
ds <- filter(ipeds_adm, `State abbreviation_value`%in%c("MA", "NY", "NJ"),
                        `Control of institution`=="Public")
```

3 Visualizations

There are many ways to create visualizations in R. It was founded to be a free software environment for statistical computing and **graphics**. Instead of using R's base graphing features, we are going to focus this workshop on using a package that utilizes the Grammar of Graphics.

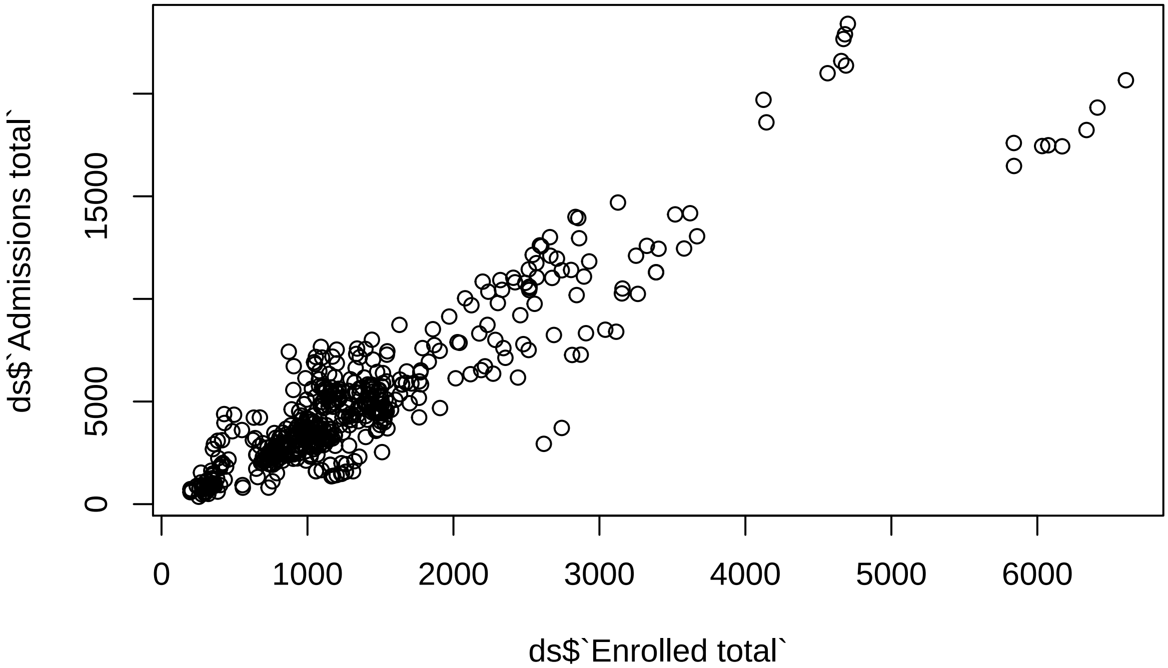
3.1 Motivation

Why? Well consider the following example. Say we want to see the relationship between total enrollment and total admissions. But then realize that we also need to show total admission.

3.1.0.1 Base R

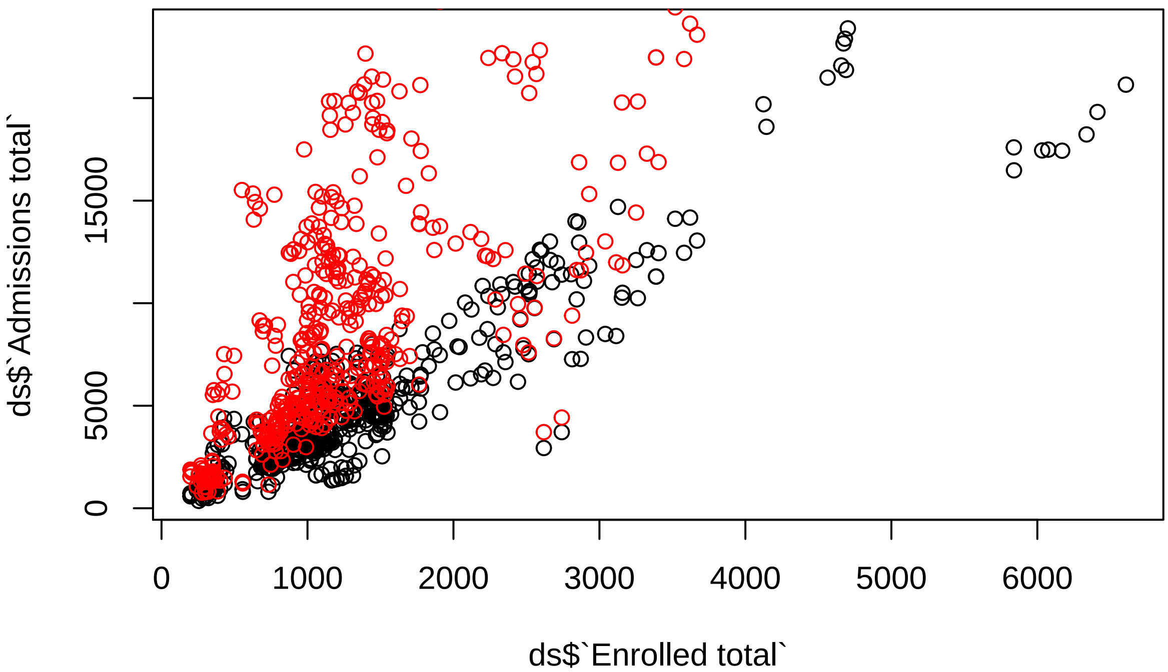
In base R:

```
plot(ds$`Enrolled total`, ds$`Admissions total`)
```



That's alright but now let's add applicants.

```
plot(ds$`Enrolled total`, ds$`Admissions total`)
points(ds$`Enrolled total`, ds$`Applicants total`, col="red")
```

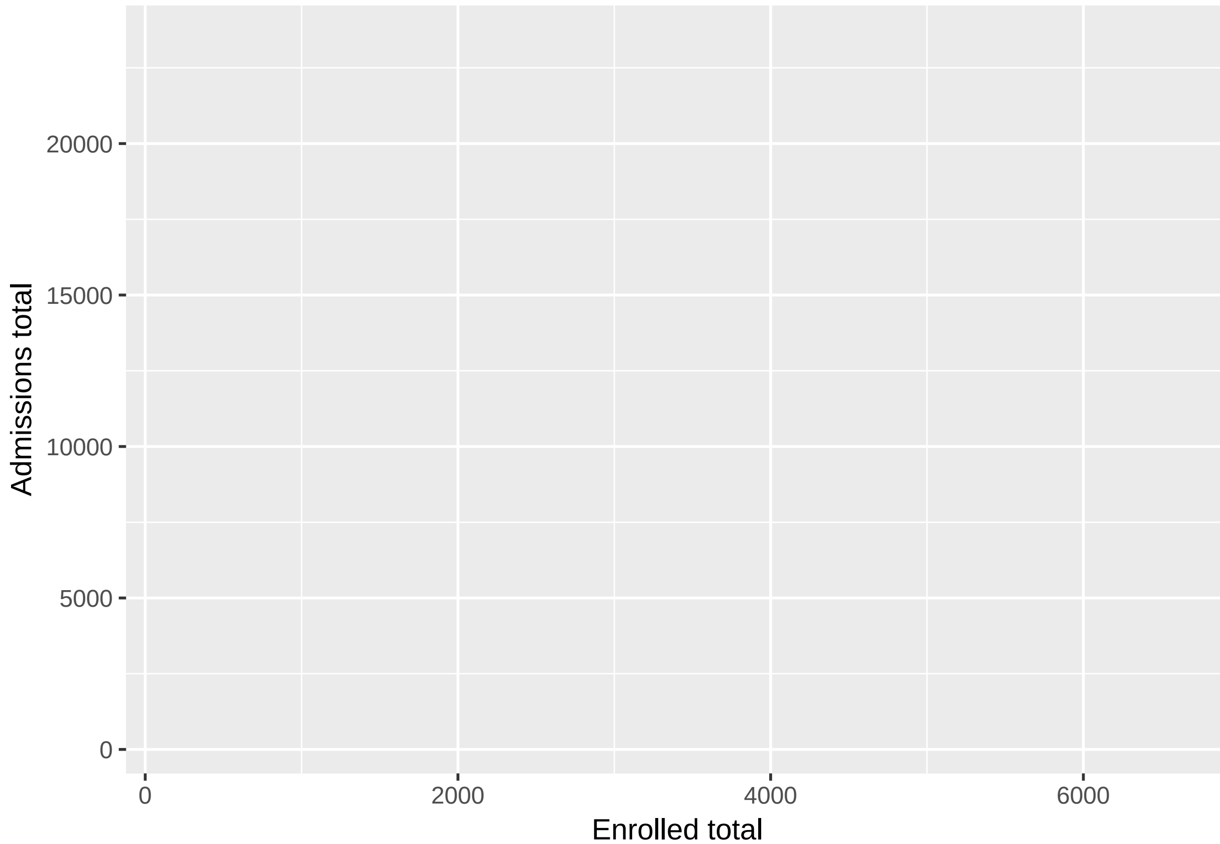


Notice the y axis did not change and the result is just a static image.

3.1.0.2 ggplot2

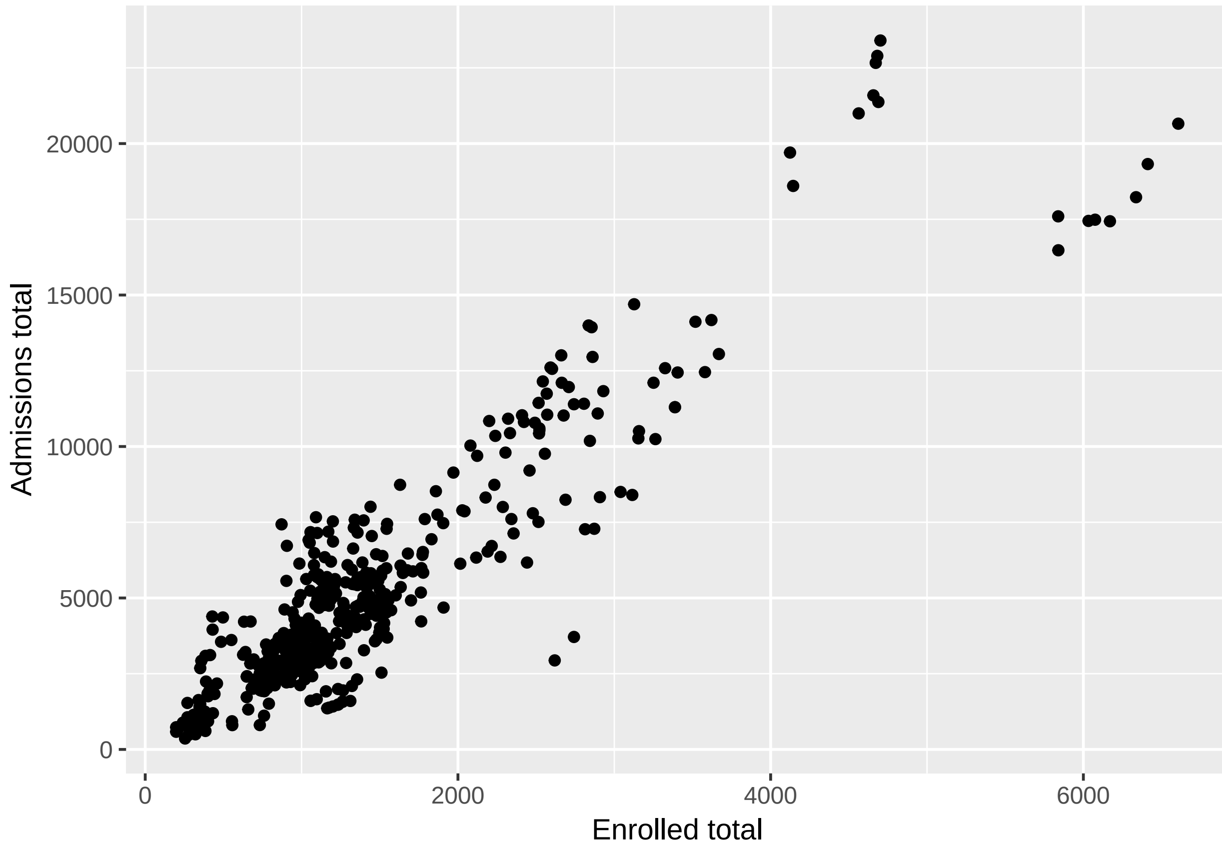
Using `ggplot2`:

```
ggplot(ds, aes(x = `Enrolled total`, y = `Admissions total`))
```



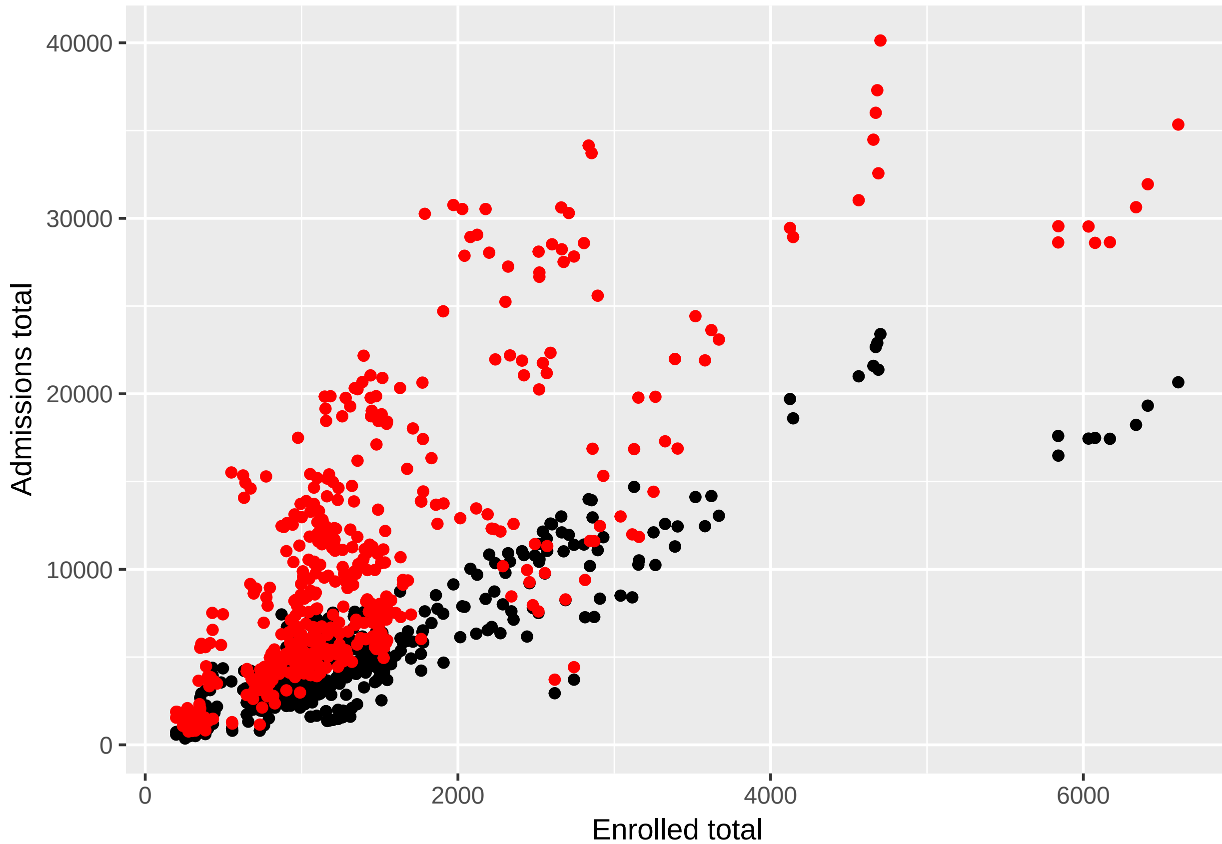
Creates the first layer our coordinate plane but now we need to add the layer of points.

```
ggplot(ds, aes(x = `Enrolled total`, y = `Admissions total`)) + geom_point()
```



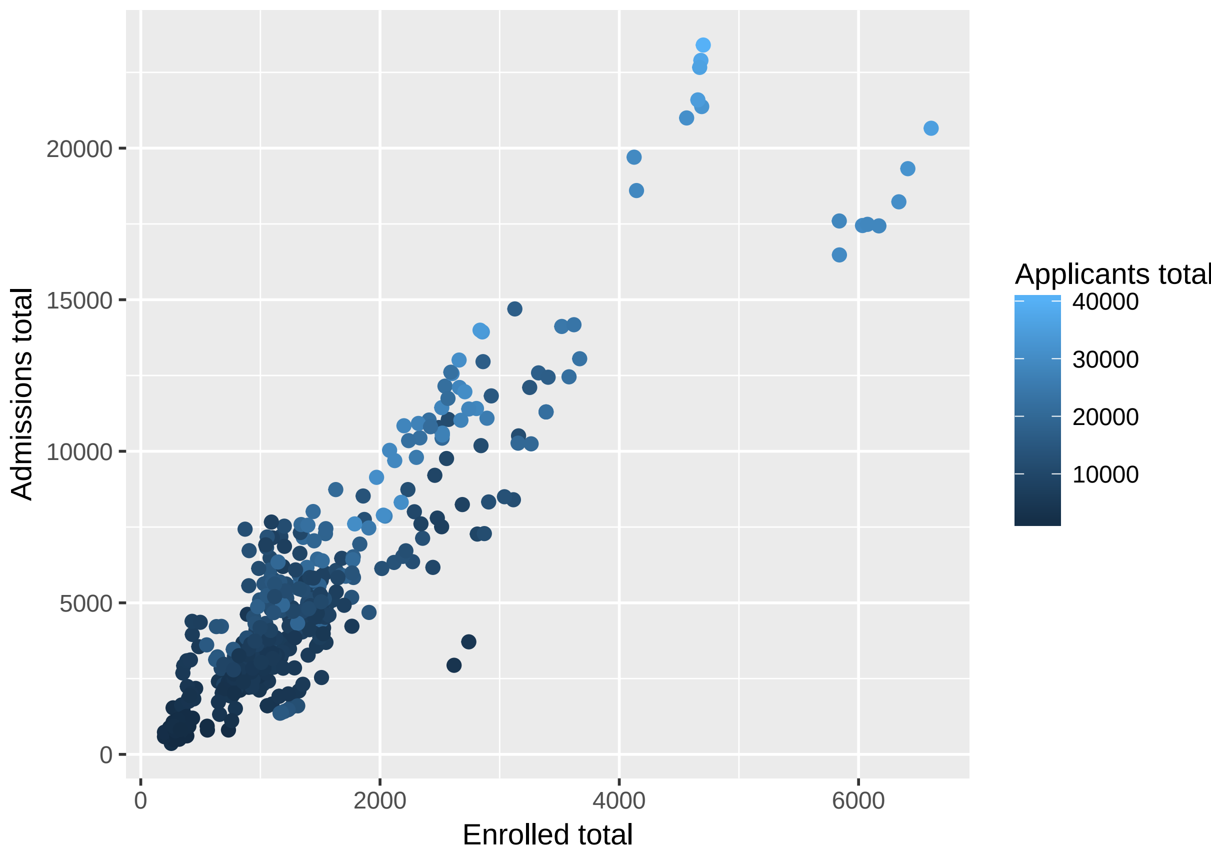
That looks similar, slightly better axis labels but now what happens if we want to add another layer of points.

```
ggplot(ds, aes(x = `Enrolled total`, y = `Admissions total`)) + geom_point() +
  geom_point(aes(x = `Enrolled total`, y = `Applicants total`), color = "red")
```



Notice the axis changed! This is because `ggplot` results in an object and will adapt to each new layer. Now this isn't exactly how one should use this function because `ggplot` when used correctly is even smarter and will even add a legend. Here is how that might work:

```
ggplot(ds, aes(x = `Enrolled total`, y = `Admissions total`,
               color = `Applicants total`)) +
  geom_point(size = 2)
```



Convinced?

3.1.1 Grammar of Graphics

The Grammar of Graphics by Leland Wilkinson was written for statisticians, computer scientists, geographers, research and applied scientists, and others interested in visualizing data. It is based on identifying the components of a graphic and the idea of building up a graphic from multiple layers of data.

3.1.2 Structure

There are 7 grammatical elements, the first three are required for every plot.

1. Data
2. Aesthetics
3. Geometries
4. Facets
5. Statistics
6. Coordinates
7. Themes

3.1.3 Syntax

Hadley Wickham took Wilkinson's work and created this graphical framework in R which resulted in `ggplot2`.

The main framework for a `ggplot` is:

```
ggplot(data = DATA, mapping = aes(MAPPINGS)) +
  GEOM_FUNCTION(stat = STAT,
                position = POSITION) +
  SCALE_FUNCTION() +
  FACET_FUNCTION() +
  COORDINATE_FUNCTION() +
  THEME_FUNCTION()
```

Similarly, only the `data =` (data), `aes()` (aesthetics), and `GEOM_FUNCTION()`(geometries) are required.

3.2 Research Question 1

What is the relationship between admit rate and yield for public institutions in MA, NY, and NJ?

To answer this question, we first need to calculate admit rate and yield. We will use the `mutate()` function from `dplyr` to add these two variables.

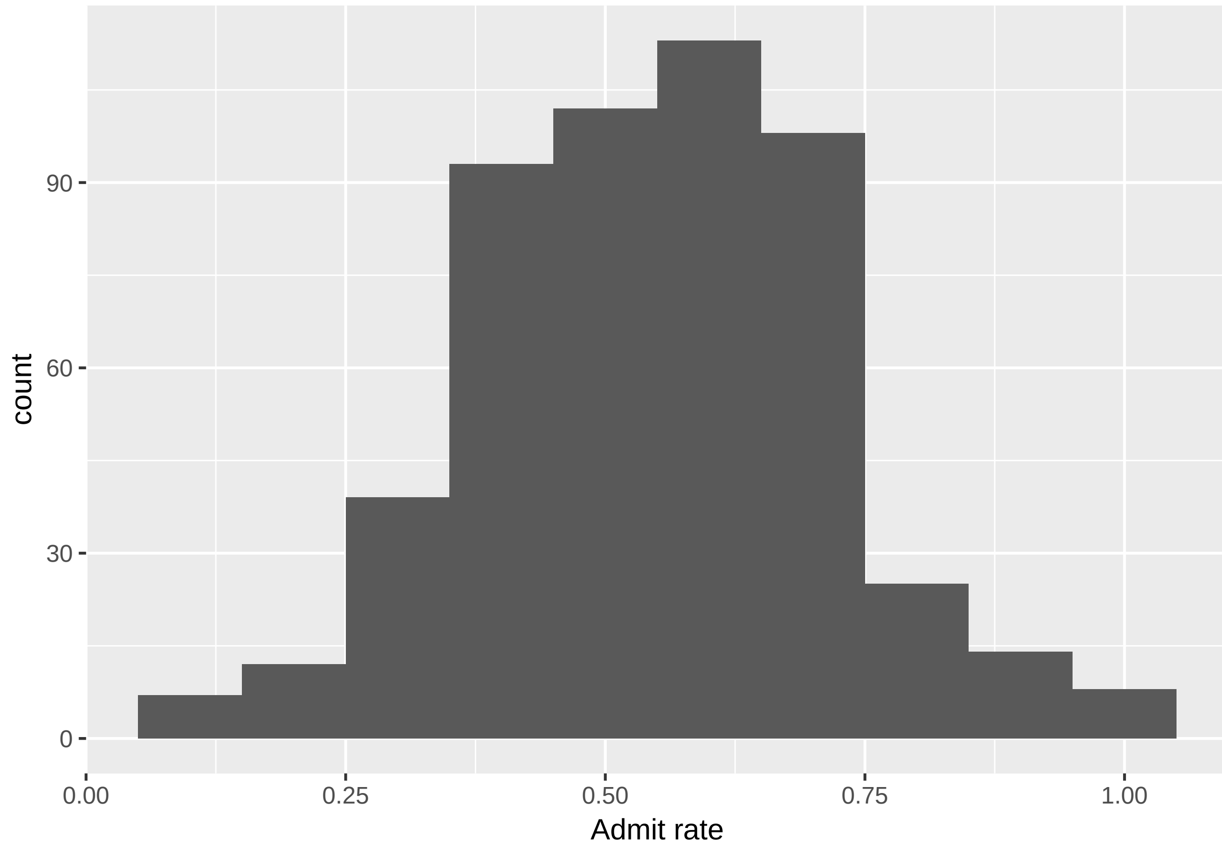
```
ds1 <- mutate(ds,
              `Admit rate` = `Admissions total`/`Applicants total`,
         `Yield` = `Enrolled total`/`Admissions total`)
```

3.2.1 Histogram

To check our calculations we can make histograms of these two new variables.

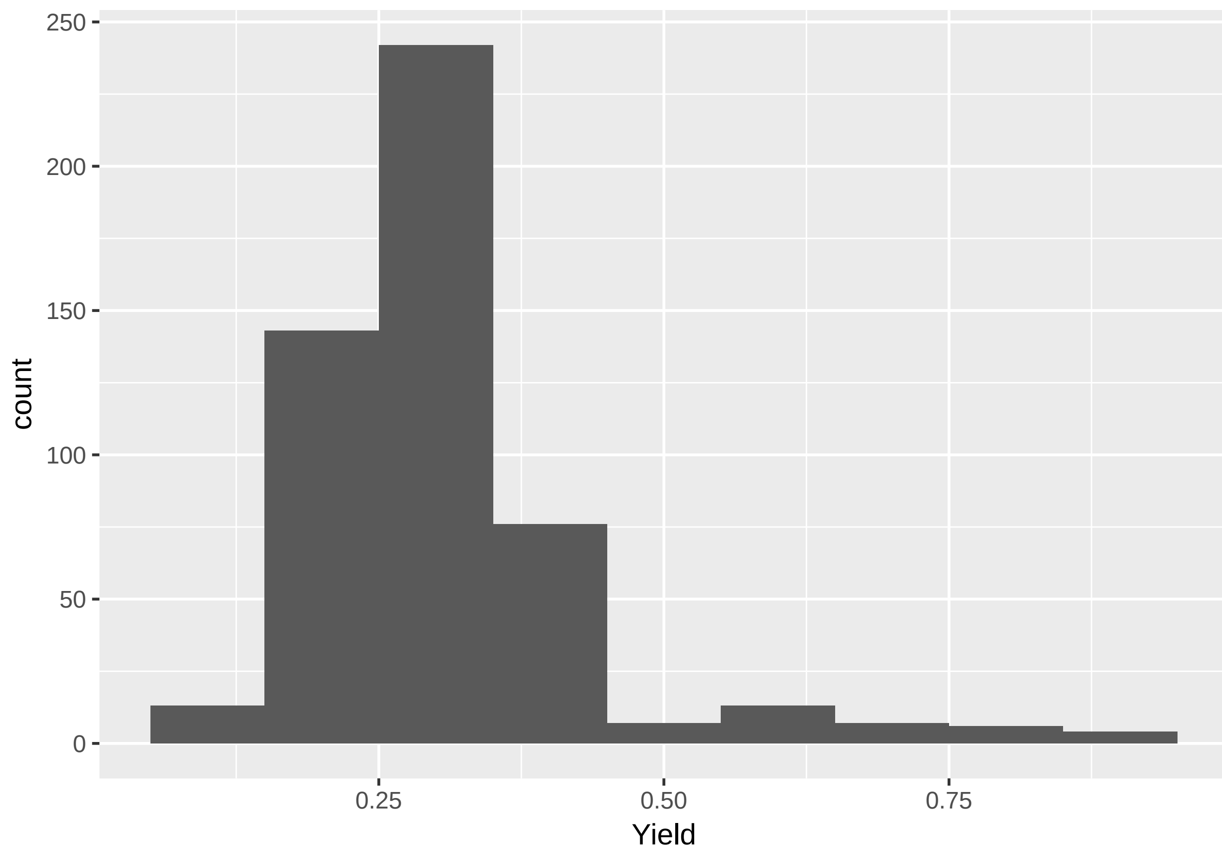
For Admit rate:

```
ggplot(ds1, aes(x = `Admit rate`)) + geom_histogram(binwidth = .1)
```



For Yield:

```
ggplot(ds1, aes(x = `Yield`)) + geom_histogram(binwidth = .1)
```

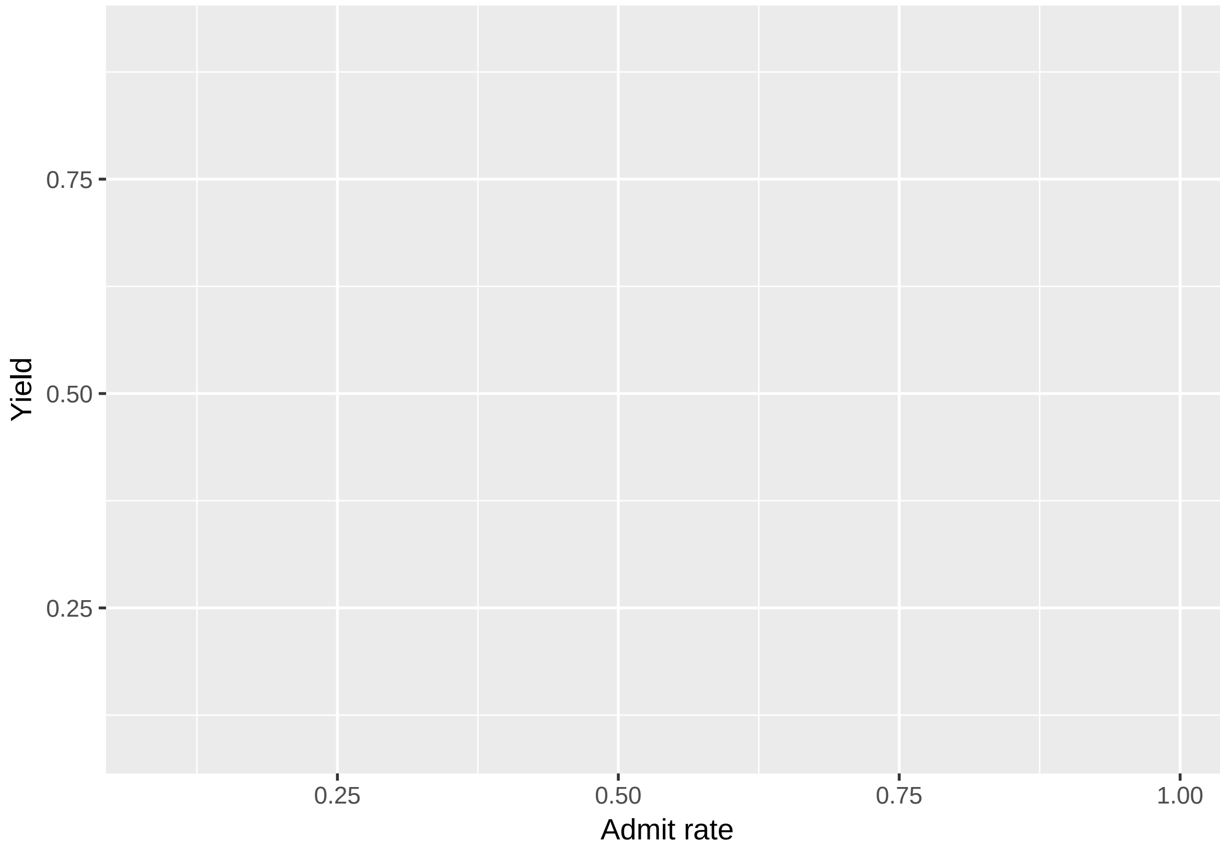


Now we will create a scatter plot with these two variables.

3.2.2 Scatter plot

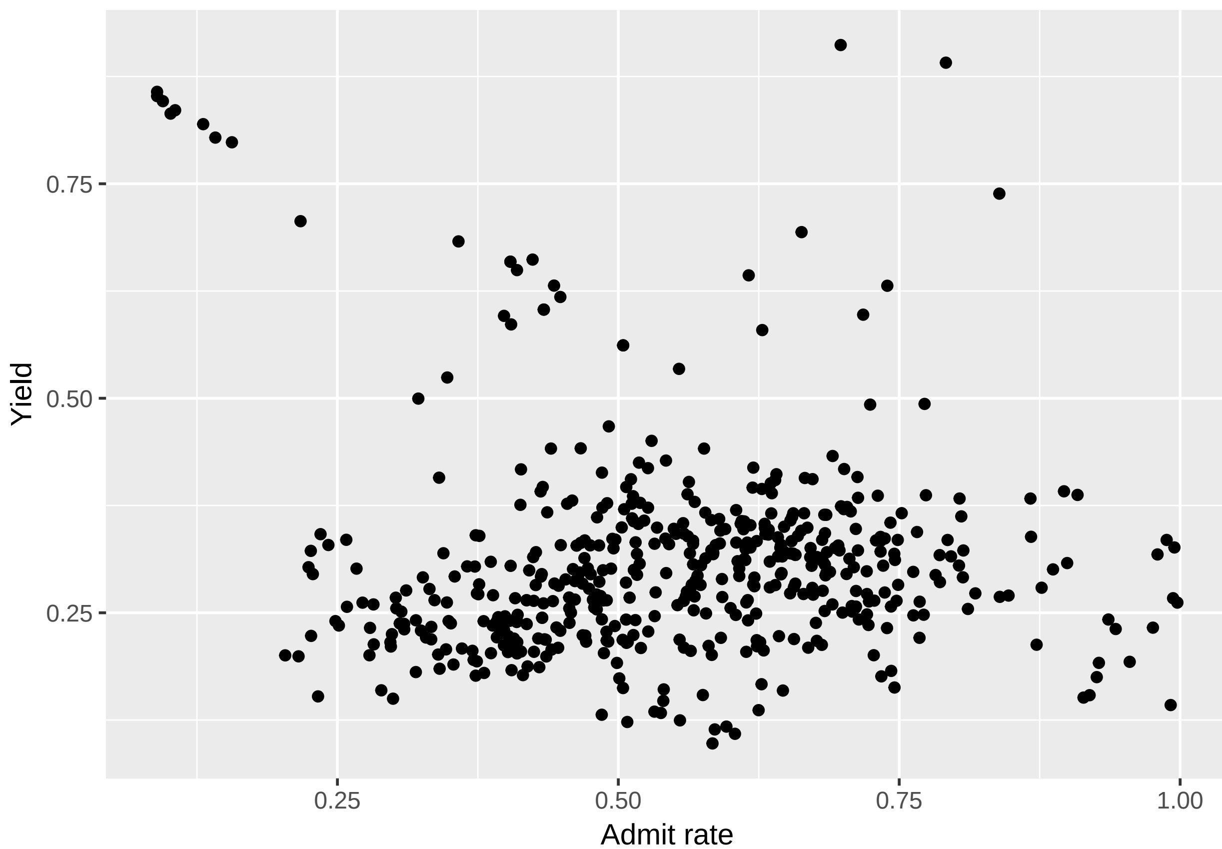
First layer is the data and aesthetics - the mapping of the data to the coordinate plane. Here we map `Admit rate` to the x axis and `Yield` to the y axis.

```
ggplot(ds1, aes(x = `Admit rate`, y = Yield))
```



Next, we add points.

```
ggplot(ds1, aes(x = `Admit rate`, y = Yield)) + geom_point()
```



Then, we will map the three states to the color of each of the points. To get a nice layer we will add this aesthetic to the first line. Note you could also add this aesthetic to `geom_point()`. Try it!

```
ggplot(ds1, aes(x = `Admit rate`, y = Yield, color = `State abbreviation`)) +
  geom_point()
```

Note you could also add this aesthetic to `geom_point()`, like below. Try it!

```
ggplot(ds1, aes(x = `Admit rate`, y = Yield)) +
  geom_point(aes(color =`State abbreviation` ))
```

Now that the data is all in the right place we want to make the graphic more digestible. So we will next add information to the coordinate or scale layer.

```
ggplot(ds1, aes(x = `Admit rate`, y = Yield, color = `State abbreviation`)) +
  geom_point() +
  scale_x_continuous(name = "Admit Rate",
                     limits = c(0, 1),
                     labels = scales::percent) +
  scale_y_continuous(name = "Yield",
                     limits = c(0, 1),
                     labels = scales::percent)
```

There are many options for `scale_` functions, you can see them on the cheat sheet. The key is that the second word matches the aesthetic and the third word matches the data type. Here both admit rate and yield are continuous variables.

Here are some more scale options:

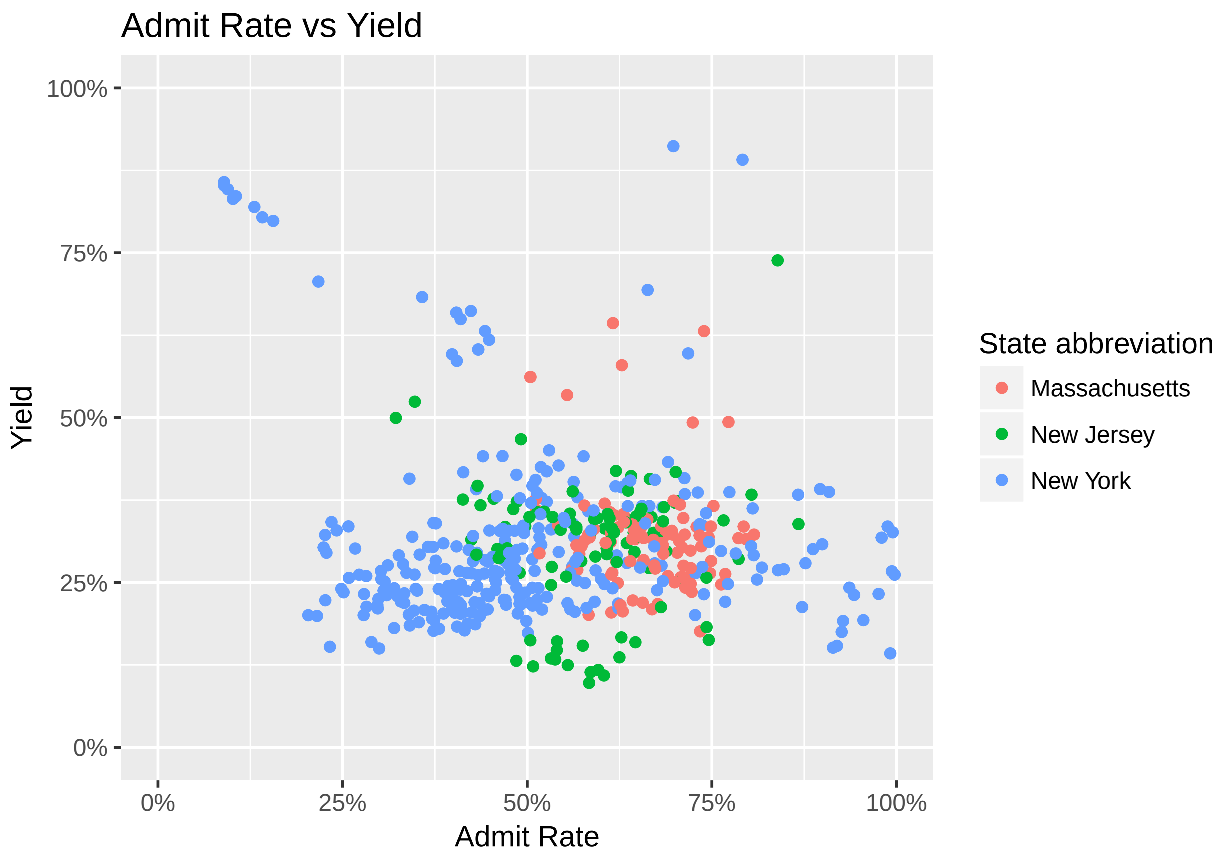
Function	Description
`scale_*_continuous()`	map continuous values to visual values
`scale_*_discrete()`	map discrete values to visual values
`scale_*_identity()`	use data values as visual values
`scale_*_manual(values = c())`	map discrete values to manually chosen visual values

Where * can be x, y, alpha, color, fill, linetype, shape, or size. See the cheat sheet for more information.

We can also add a title.

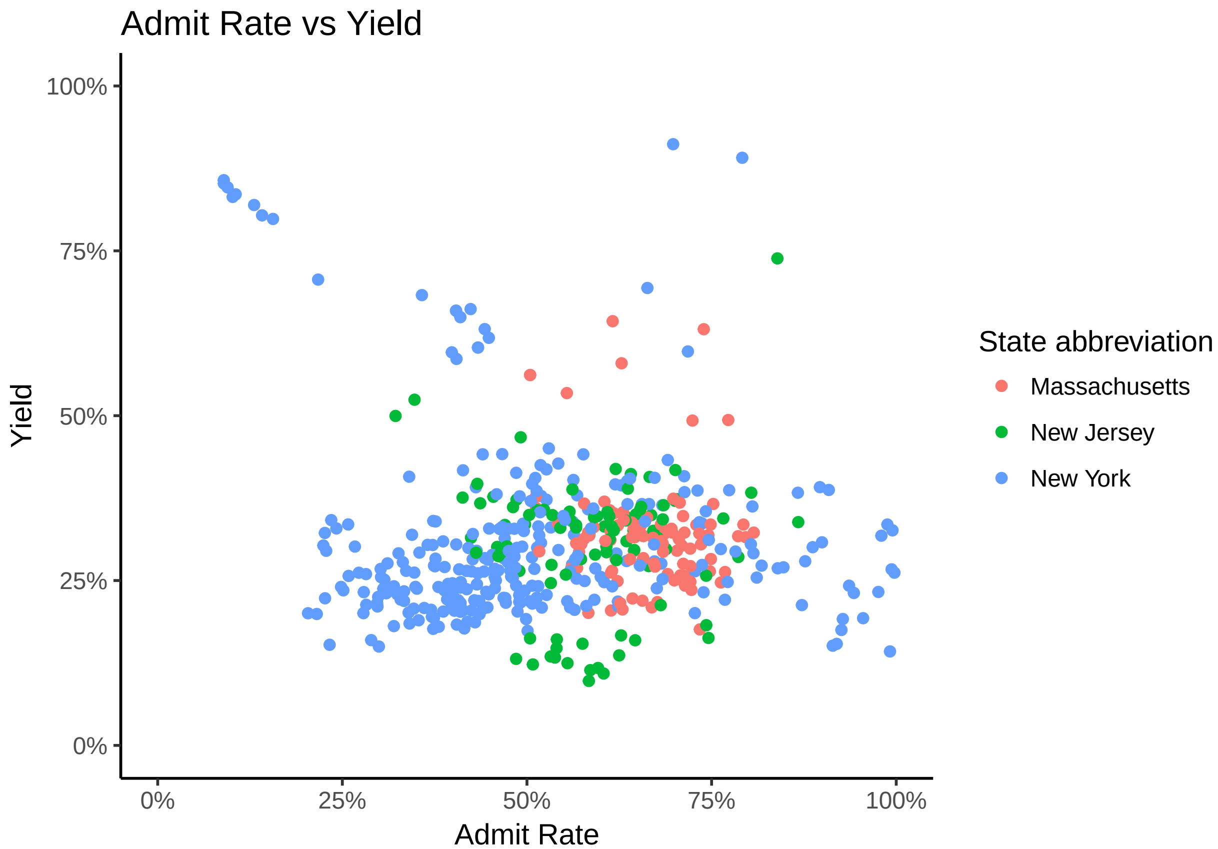
```
ggplot(ds1, aes(x = `Admit rate`, y = Yield, color = `State abbreviation`)) +
  geom_point() +
  scale_x_continuous(name = "Admit Rate",
                     limits = c(0, 1),
```

```
                       labels = scales::percent) +
  scale_y_continuous(name = "Yield",
                       limits = c(0, 1),
                       labels = scales::percent) +
  ggtitle("Admit Rate vs Yield")
```



Lastly, the grey background is too much but we can change it by changing the theme layer. There are many theme options, including creating your own and a whole other package (`ggthemes`)!

```
ggplot(ds1, aes(x = `Admit rate`, y = Yield, color = `State abbreviation`)) +
  geom_point() +
  scale_x_continuous(name = "Admit Rate",
                       limits = c(0, 1),
                       labels = scales::percent) +
  scale_y_continuous(name = "Yield",
                       limits = c(0, 1),
                       labels = scales::percent) +
  ggtitle("Admit Rate vs Yield") +
  theme_classic()
```



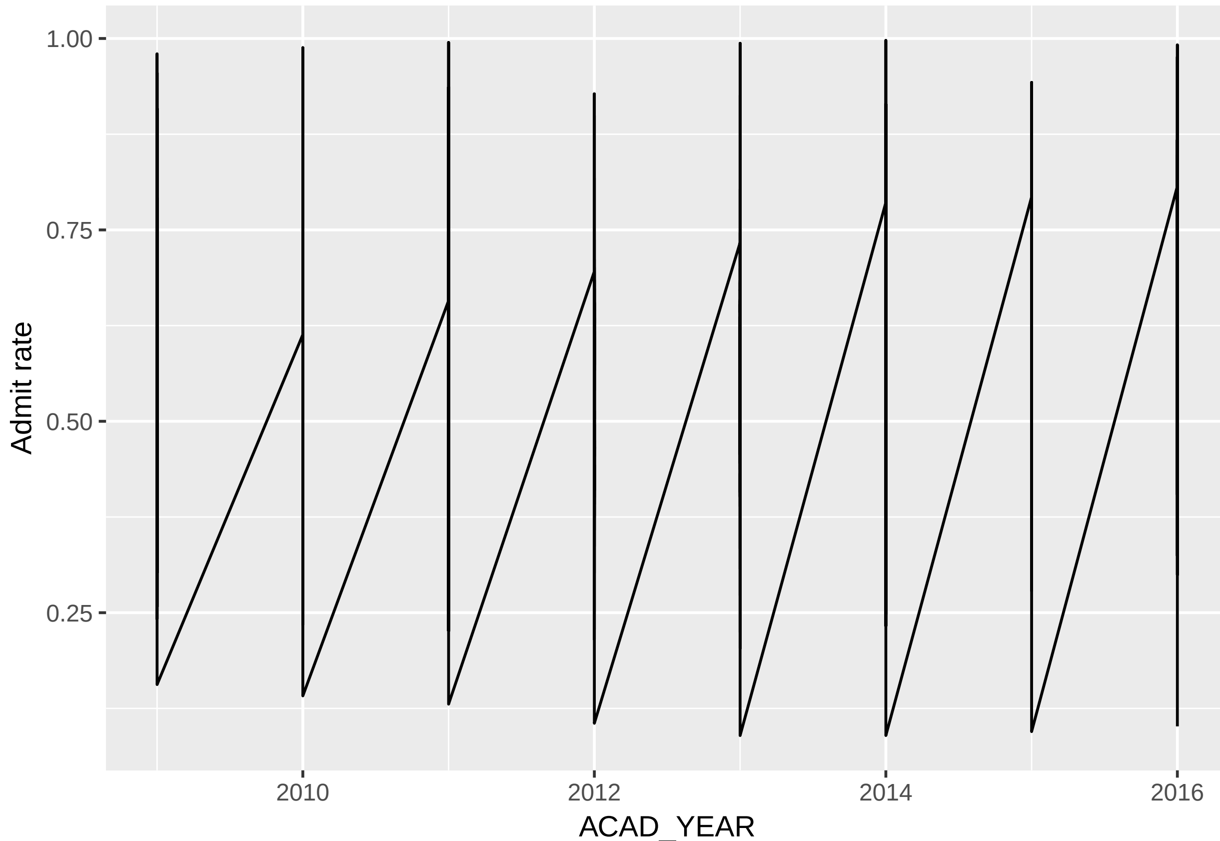
3.3 Research Question 2

Has the average admit rate changed over time for public institutions in MA, NY, or NJ?

To explore this question we will use `geom_line()`

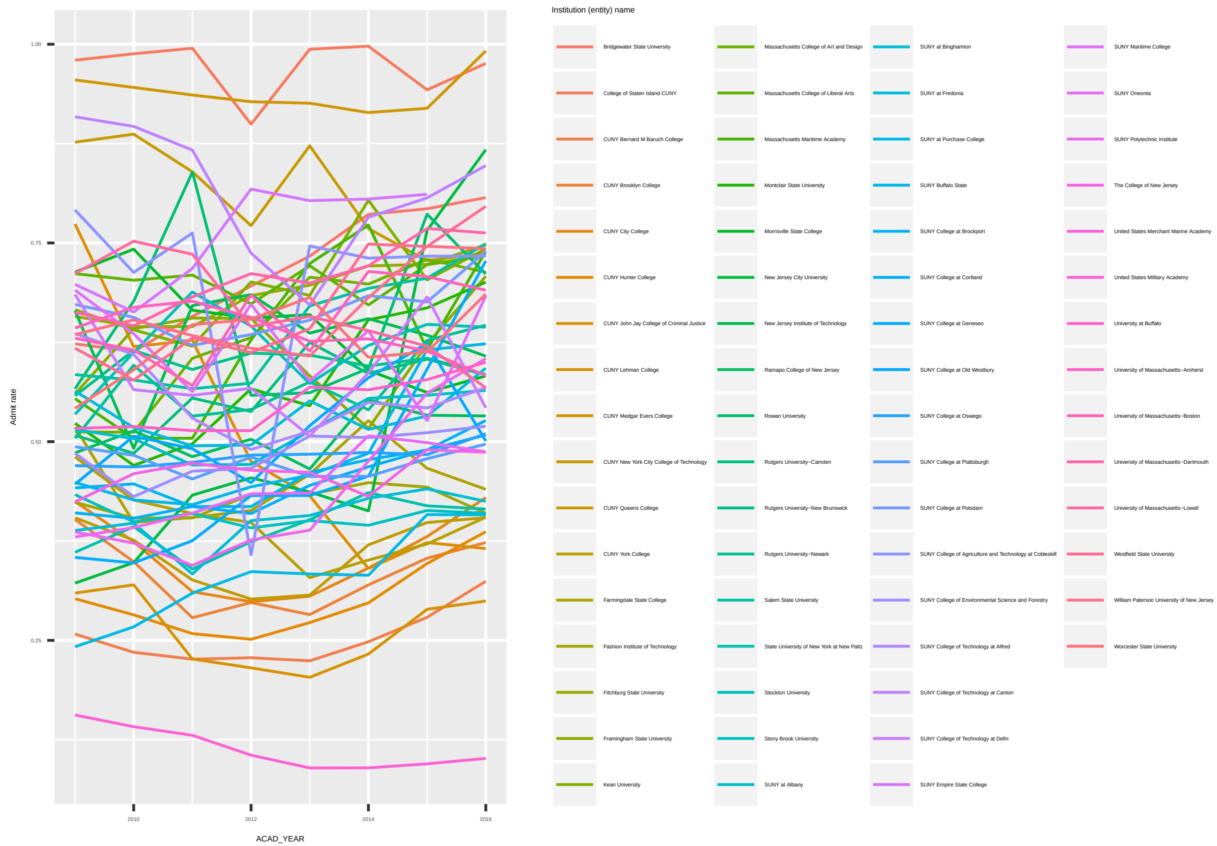
Ideally we would be able to take our data as is and plot time versus admit rate but here's what that would look like:

```
ggplot(ds1, aes(x = `ACAD_YEAR`, y = `Admit rate`)) +
  geom_line()
```



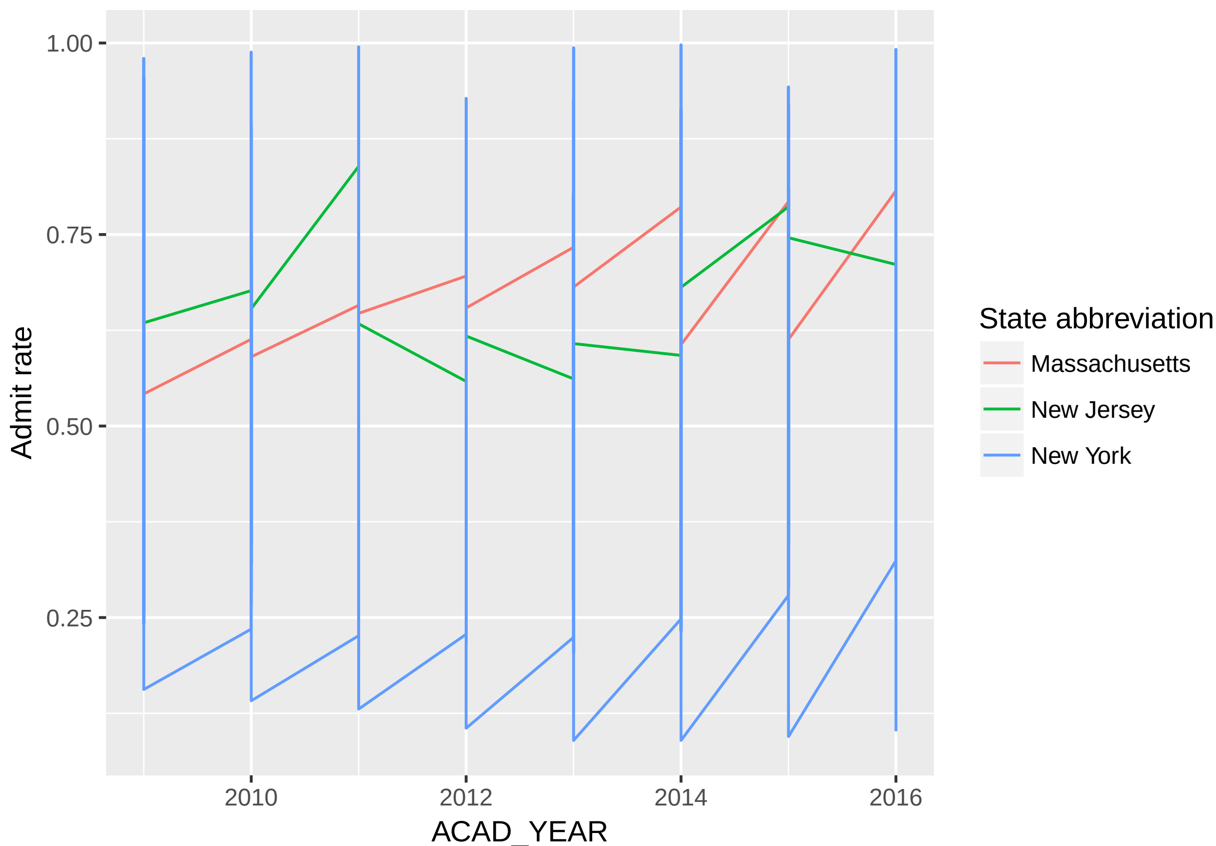
That's awful. It is trying to connect every point with a line segment, there is no aggregation happening. Let's give it some more information.

```
ggplot(ds1, aes(x = `ACAD_YEAR`, y = `Admit rate`,
                color = `Institution (entity) name`)) +
  geom_line()
```



Now we have spaghetti! What we really want is one line per state. Let's see what that would look like

```
ggplot(ds1, aes(x = `ACAD_YEAR`, y = `Admit rate`, color = `State abbreviation`)) +
  geom_line()
```



The problem is that we have to first calculate the average admit rate for each group. We are going to use `dplyr`'s `group_by()` and `summarise()` to accomplish this goal.

3.3.1 More Data Wrangling

3.3.1.1 `group_by()`

First we say how we want our data grouped.

```
ds1_group <- group_by(ds1, `State abbreviation`,`ACAD_YEAR`)
```

This function does not change your underlying data, instead it adds a metadata to your data. It changes the scope of each function from operating on the entire dataset to operating on it group-by-group.

3.3.1.2 `summarise()`

Next we specify how we want to aggregate the data.

```
ds1_group <- summarise(group_by(ds1, `State abbreviation`,`ACAD_YEAR`),
                       mean(`Admit rate`))
```

Is it just me or is the above code confusing to follow? Let's change that.

3.3.1.3 %>%

The pipe operator `%>%` allows us to make code more readable when we are applying multiple functions to the same data set.

Traditionally we have specified `z <- function(x, y)` but using the pipe we will write `z <- x %>% function(y)` you can think of it as the word 'then'.

```
ds2_group <- ds1 %>%
  group_by(`State abbreviation`,`ACAD_YEAR`) %>%
  summarise(N = n(), avg_admit = mean(`Admit rate`))
```

Now we have the average admit rate per state and year.

Aside we could have also done this with our creation of `ds` and `ds1` like so:

```
ds <- ipeds_adm %>%  filter(`State abbreviation_value`%in%c("MA", "NY", "NJ"),
                            `Control of institution`=="Public")
```

```
ds1 <- ds %>% mutate(`Admit rate` = `Admissions total`/`Applicants total`,
                     `Yield` = `Enrolled total`/`Admissions total`)
```

We could have even strung all of these together like this:

```
ds2_group <- ipeds_adm %>%
  filter(`State abbreviation_value`%in%c("MA", "NY", "NJ"),
         `Control of institution`=="Public") %>%
  mutate(`Admit rate` = `Admissions total`/`Applicants total`,
         `Yield` = `Enrolled total`/`Admissions total`) %>%
  group_by(`State abbreviation`,`ACAD_YEAR`) %>%
  summarise(N = n(),
            avg_admit = mean(`Admit rate`))
```

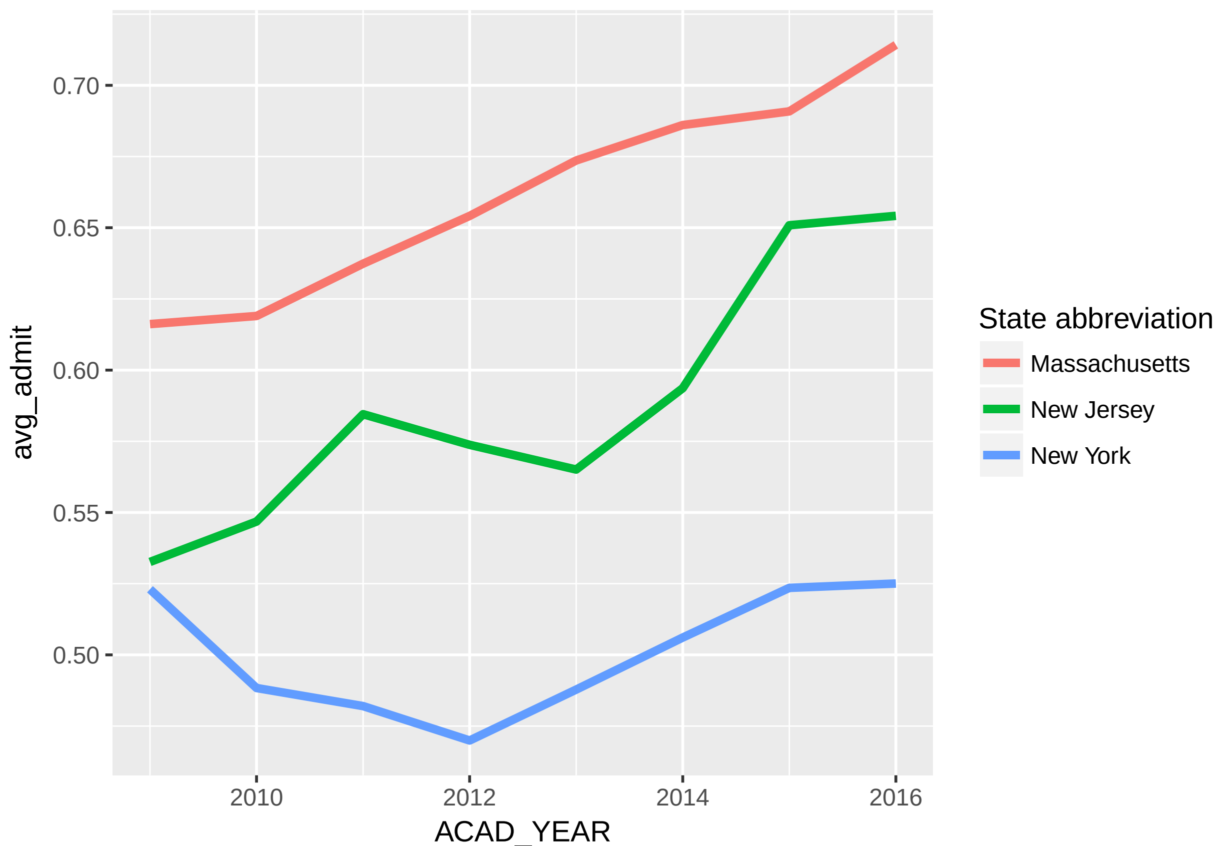
3.3.2 Line plot

Now that our data is all set let's build some layers.

First the base plot.

```
p2 <- ggplot(ds2_group, aes(x = ACAD_YEAR, y = avg_admit,
                            group = `State abbreviation`,
                            color = `State abbreviation`,
                            shape = `State abbreviation`)) +
  geom_line(lwd = 1.5)

p2
```

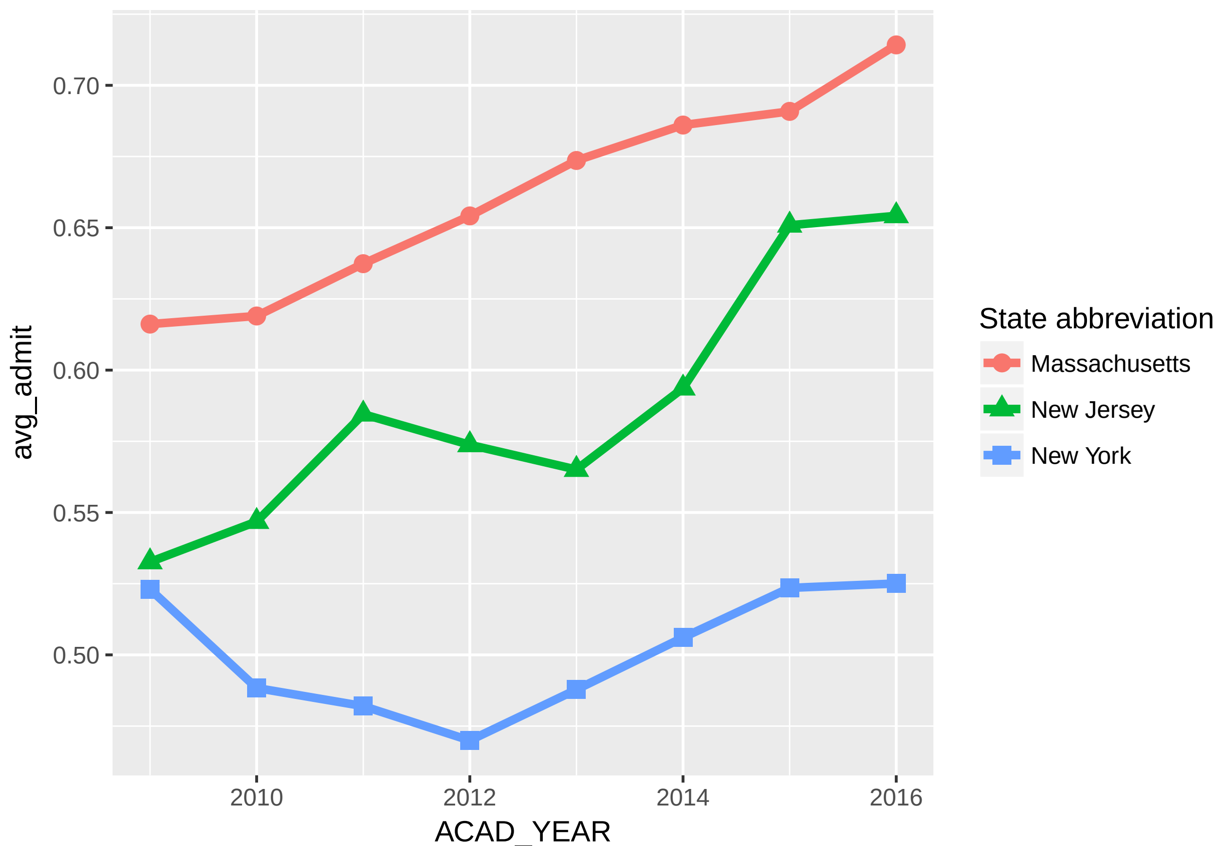


To really demonstrate that we are working with an object I have assigned the plot to object `p2`

Next let's add a layer of points in case this gets printed in black and white.

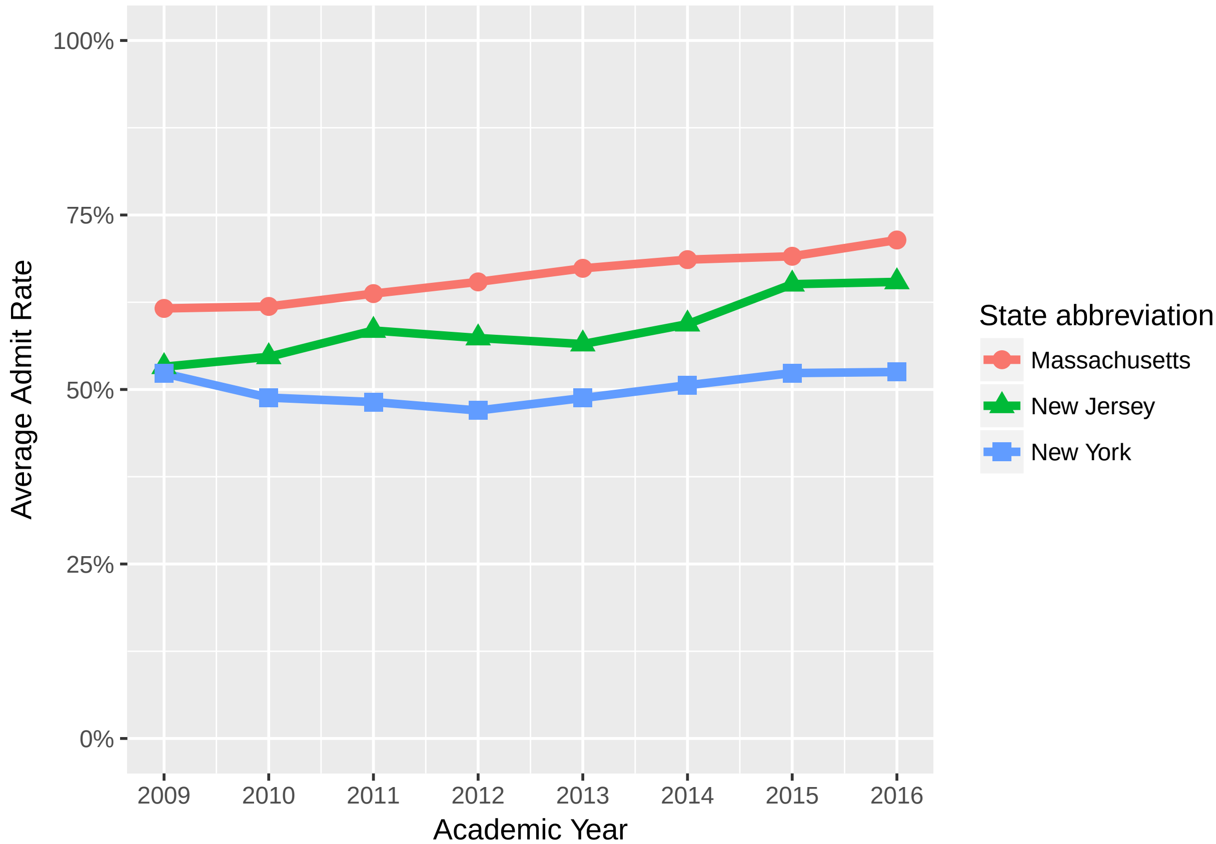
```
p2 <- p2 + geom_point(size = 3)

p2
```



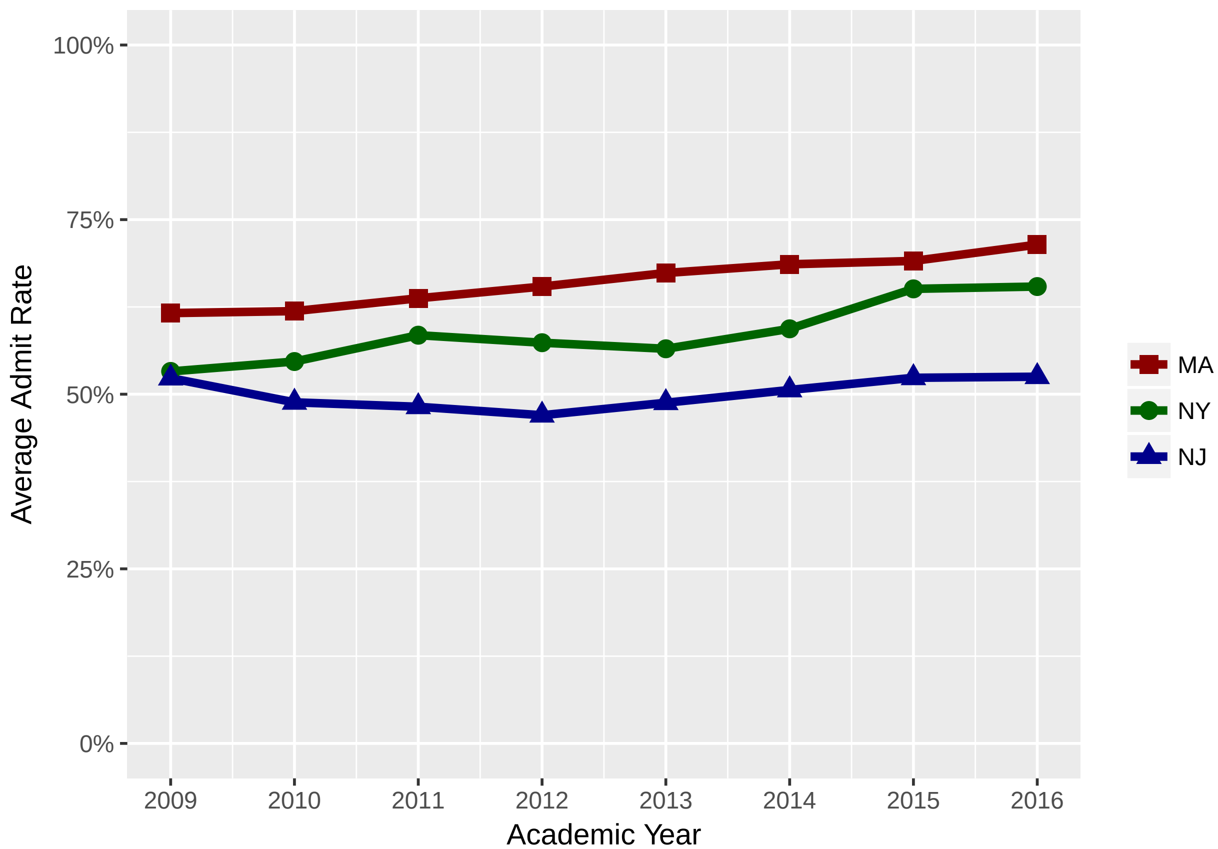
Next we will make the plot less deceiving by changing the y limits and making the axes labels more friendly.

```
p2 <- p2 + scale_x_continuous(name = 'Academic Year',
                    breaks = seq(2009,2016, by=1)) +
           scale_y_continuous(name = 'Average Admit Rate',
                    limits = c(0,1),
                    labels = scales::percent)
p2
```



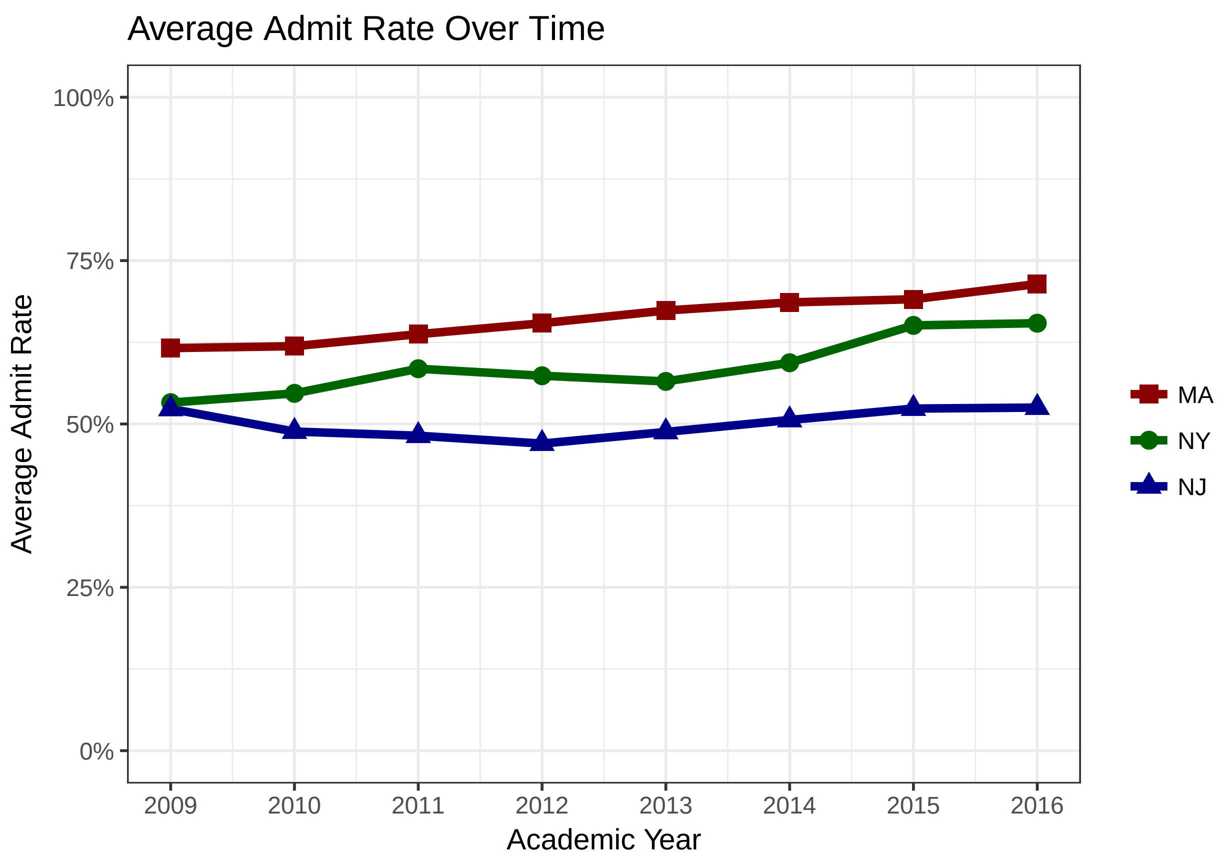
Let's add some more personalization and format the color and shape aesthetics.

```
p2 <- p2 + scale_colour_manual(name = '',
                      labels = c('MA', 'NY', "NJ"),
                      values = c('darkred', 'darkgreen', 'darkblue')) +
  scale_shape_manual(name = '',
                     labels = c('MA', 'NY', "NJ"),
                     values = c(15, 16, 17))
p2
```



Lastly, we will add the title and change the theme

```
p2 <- p2 + ggtitle('Average Admit Rate Over Time') +
  theme_bw()
p2
```



Vola! Research question 2 answered.

3.4 Research Question 3

Are public institutions in MA, NY and NJ becoming test optional?

As we saw in plot 2 sometimes we have to do the calculation but other times `ggplot` can calculate for us. Let's see this in action using `geom_bar()`.

3.4.1 Bar plot

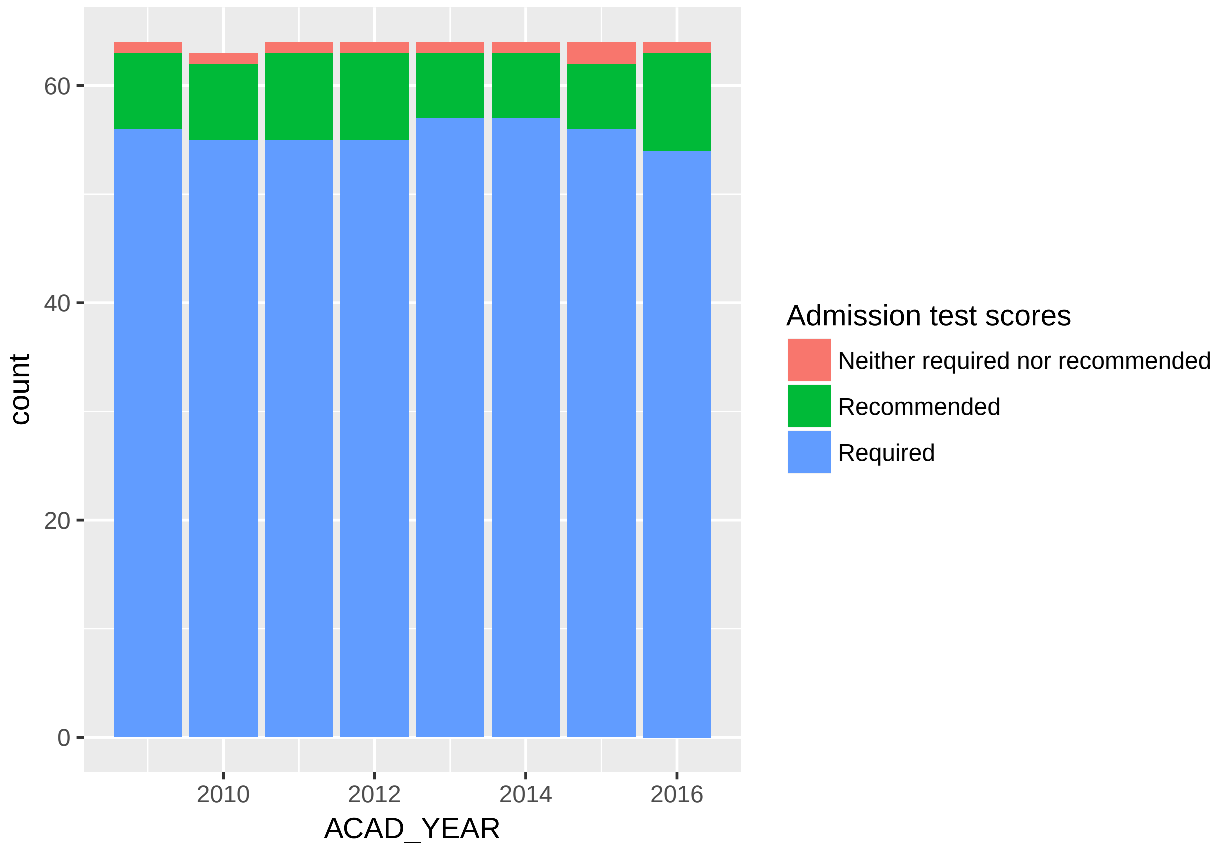
Although not necessary sometimes you just want to have in your dataset the variables that are of interest. Let's use `dplyr`'s `select()` function to only select the columns that we will use to answer question 3.

```
ds3 <- ds1 %>% select(UNITID,
                      `Institution (entity) name`,
                      `State abbreviation`,
                      `Admission test scores`,
                      ACAD_YEAR)
```

Notice that when you are using the pipe operator you can also benefit from RStudio's autocomplete feature.

For bar plots instead of specifying the y axis we need to specify what we want to *fill* the bar with. In our case that will be their admission test score policy.

```
ggplot(ds3, aes(x = ACAD_YEAR, fill = `Admission test scores`)) +
  geom_bar()
```



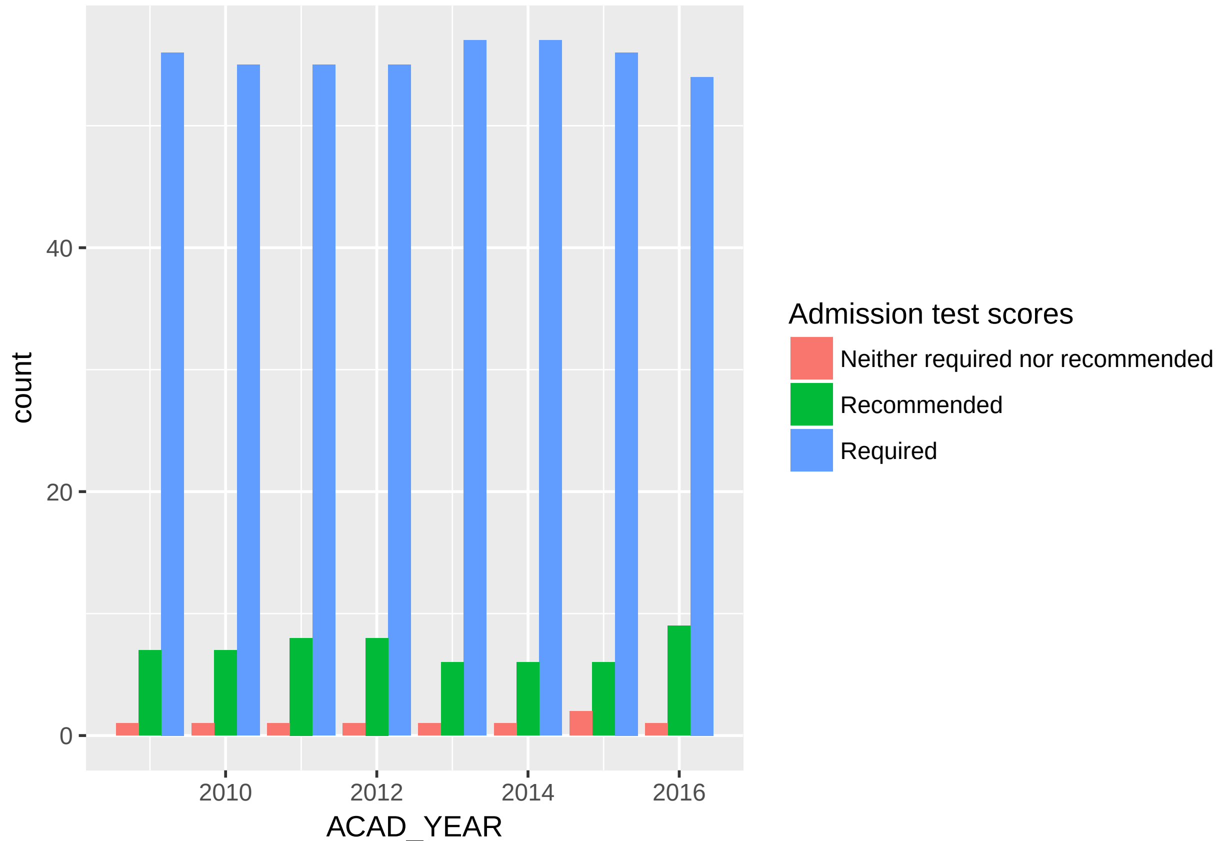
The result is a stacked bar chart with the count of institutions in each category.

3.4.1.1 `position`

We can change the type of bar chart with the `position` parameter, by default `position = 'stack'`.

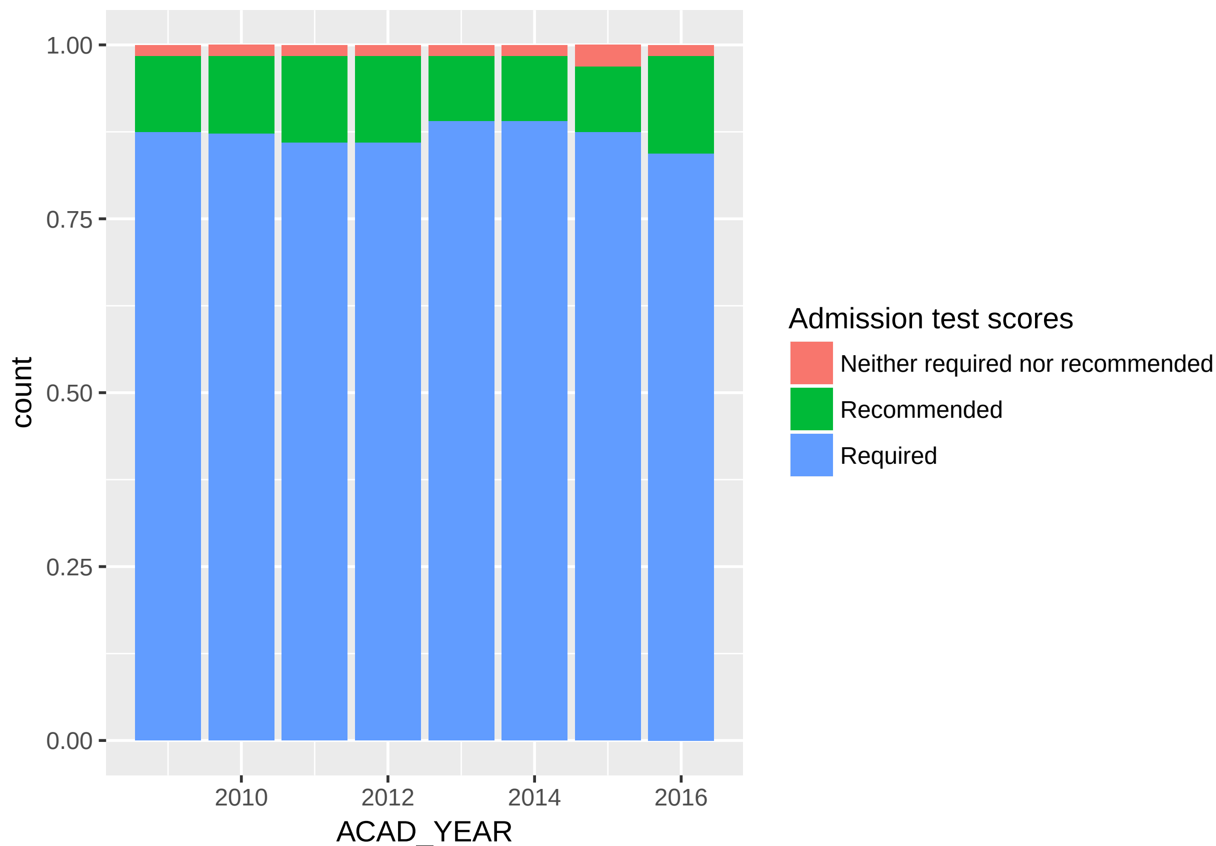
To make the bars side by side we set `position = 'dodge'`

```
ggplot(ds3, aes(x = ACAD_YEAR, fill = `Admission test scores`)) +
  geom_bar(position = 'dodge')
```



Or we can make it a proportional stacked bar chart by setting `position = 'fill'`.

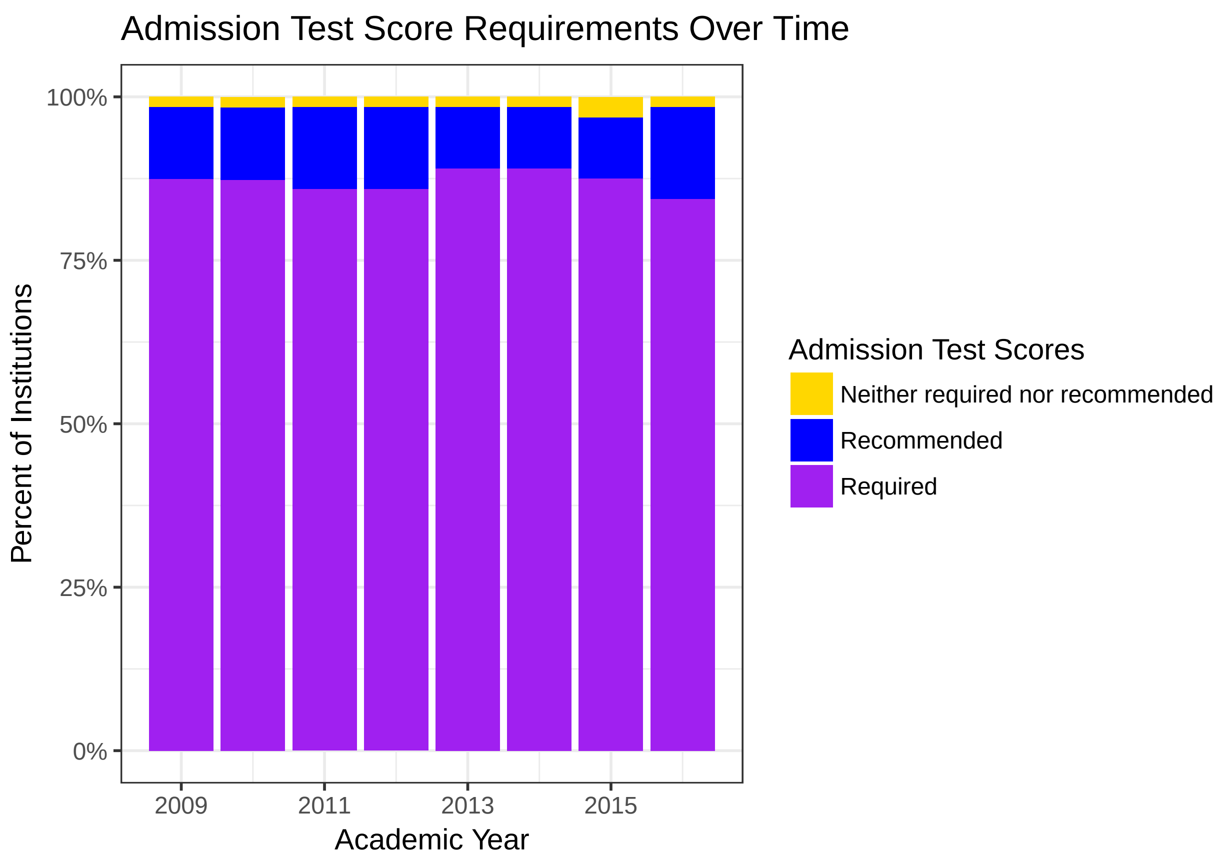
```
p3 <- ggplot(ds3, aes(x = ACAD_YEAR, fill = `Admission test scores`)) +
  geom_bar(position = 'fill')
p3
```



Let's stick with fill since some schools appear to be missing for some of the years.

Now let's make it look prettier changing the scales, title, and theme.

```
p3 <- ggplot(ds3, aes(x = ACAD_YEAR, fill = `Admission test scores`)) +
  geom_bar(position = 'fill') +
  scale_y_continuous(name = "Percent of Institutions",
                     limits = c(0,1),
                     labels = scales::percent) +
  scale_x_continuous(name = "Academic Year",
                     breaks = seq(2009, 2016, by=2)) +
  scale_fill_manual(name = "Admission Test Scores",
                    values = c("gold", "blue", "purple")) +
  ggtitle("Admission Test Score Requirements Over Time") +
  theme_bw()
p3
```

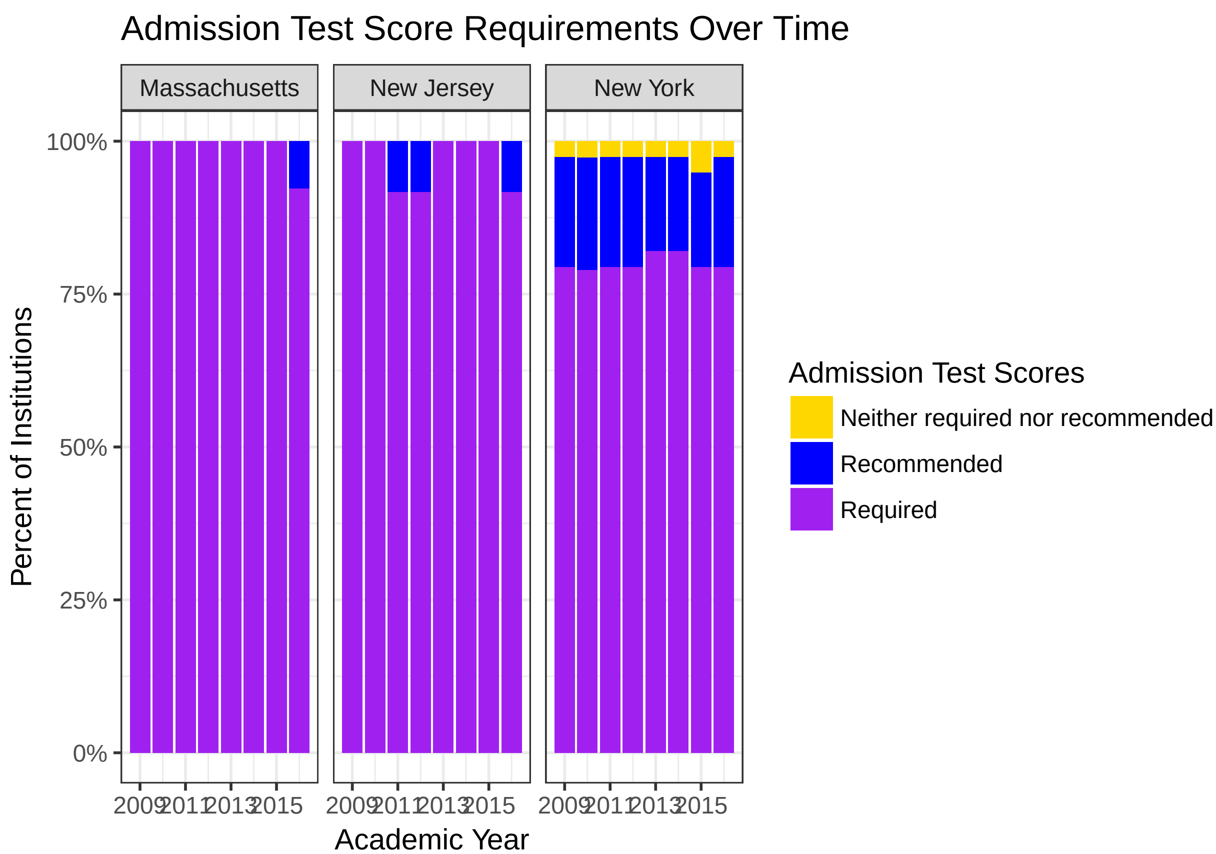


But we are still missing a key piece. Viewing this chart across states.

3.4.1.2 `facet_grid()`

In essences we want to repeat this same graph for each of the states but still want to compare them side by side. This is the case for adding a `FACET_FUNCTION()` or a facet layer.

```
p3 + facet_grid(.~`State abbreviation`)
```



The syntax for facet uses the formula rows by columns. `.~z` means keep the rows as one and have one column per category in `z`.

Try switching it!

```
p3 + facet_grid(`State abbreviation`~.)
```

There is still so much that you can do in `ggplot`, there are 34 other `geom` functions to explore but hopefully these three figures will get you started.

Next we will discuss how to distribute these figures.

4 Reporting

The next logical question after you have created a visualization is how do I to share my work with others?

This workshop will explore two options. The first is a standard export and the second is a bit more advanced.

4.1 Export Image

In RStudio there is an Export button in the Plots pane. You have three options, "Save as Image…", "Save as pdf…", and "Copy to Clipboard…". These options are good if you need to include a figure in an email or external document.

You can also use the function `ggsave`.

```
ggsave("filename.png", p3)
```

4.2 RMarkdown

R Markdown documents are fully reproducible. RStudio fully supports RMarkdown and provides an interface to weave together narrative text and code to produce elegantly formatted output.

You can make documents, interactive documents, dashboards, presentations, books, websites, and journal articles with built in templates.

It is very powerful. We will now take the three plots and create a RMarkdown document.

A RMarkdown document has three parts, the header, text, and code.

To begin first open up a new RMarkdown, go to File -> New File -> R Markdown… This screen should pop up:

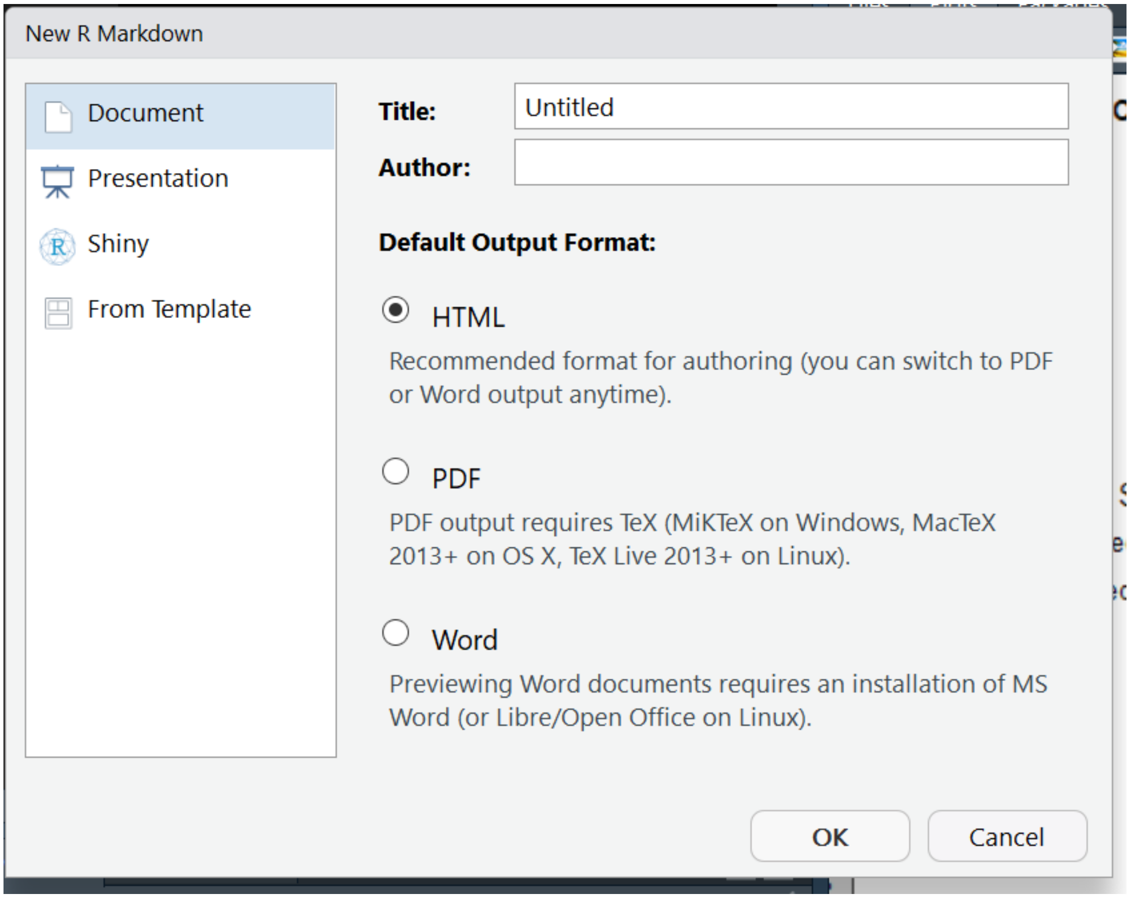


Figure 1: New RMarkdown Prompt

Go ahead and give your document a title and author, the default html is good (all three of these things you can change later) and then click OK

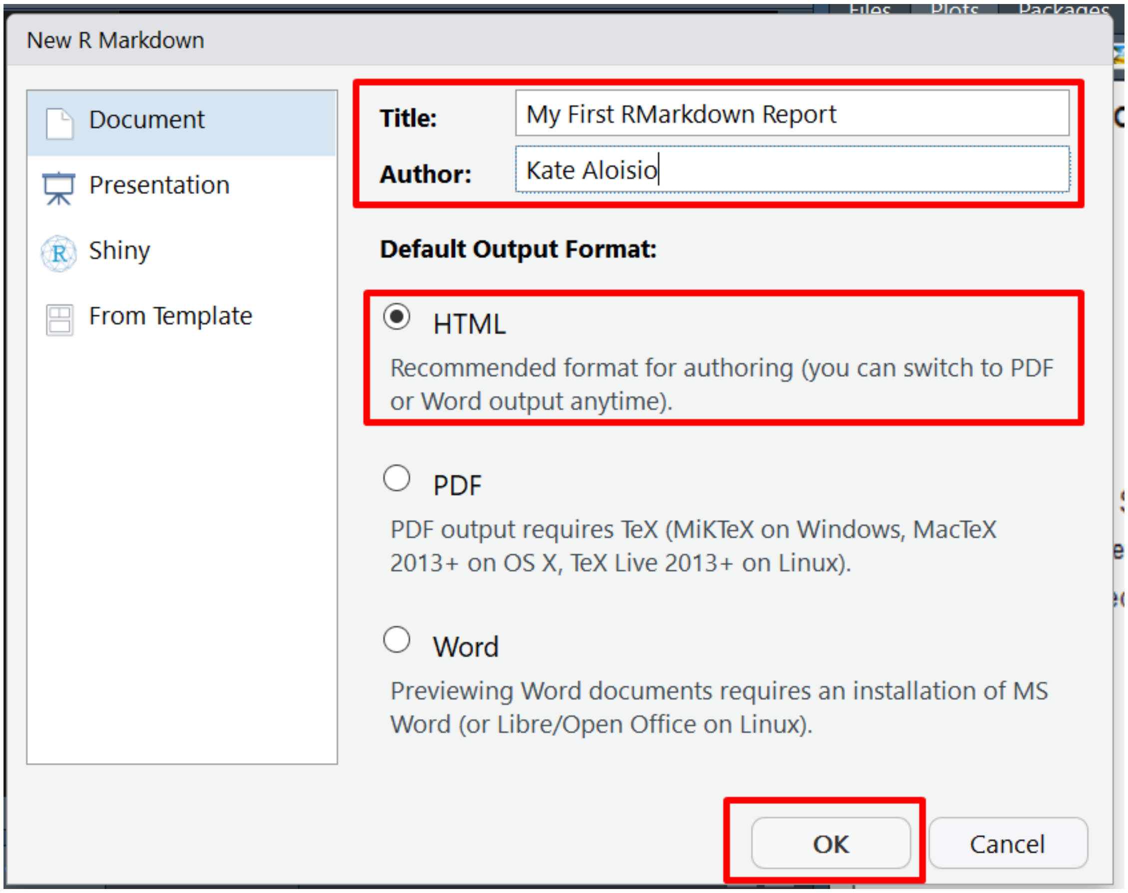


Figure 2: New RMarkdown Prompt Select

Then you should see RStudio's default report.

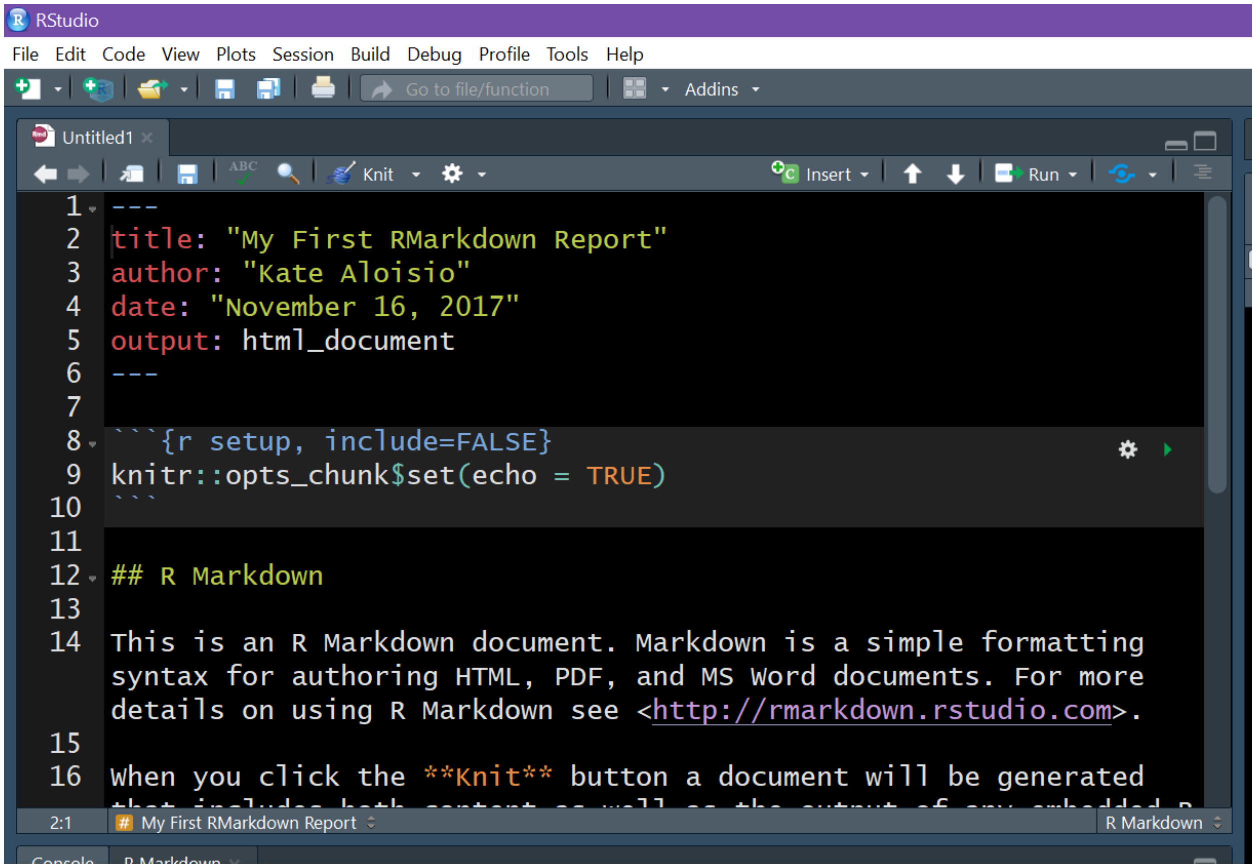


Figure 3: Default report

The file has three main parts, the header:

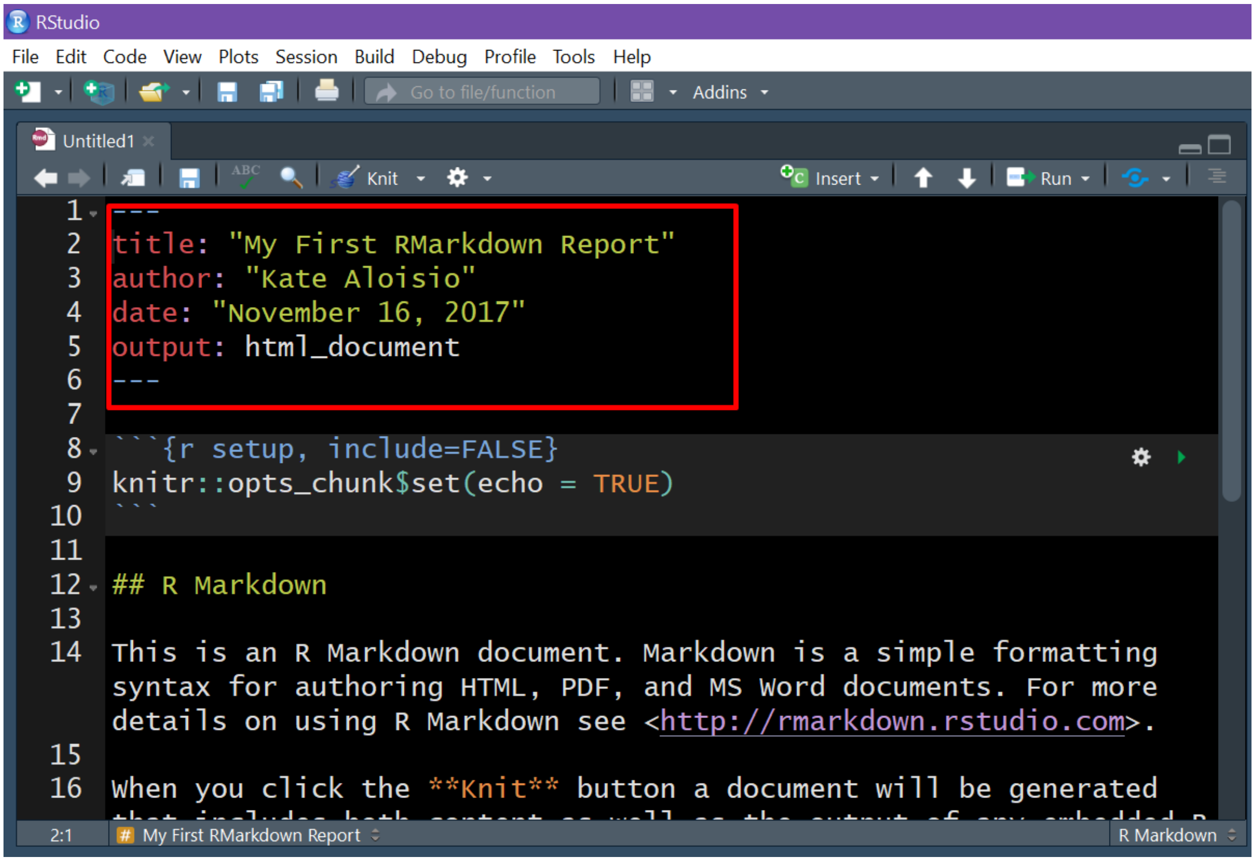


Figure 4: Header

code chunks:

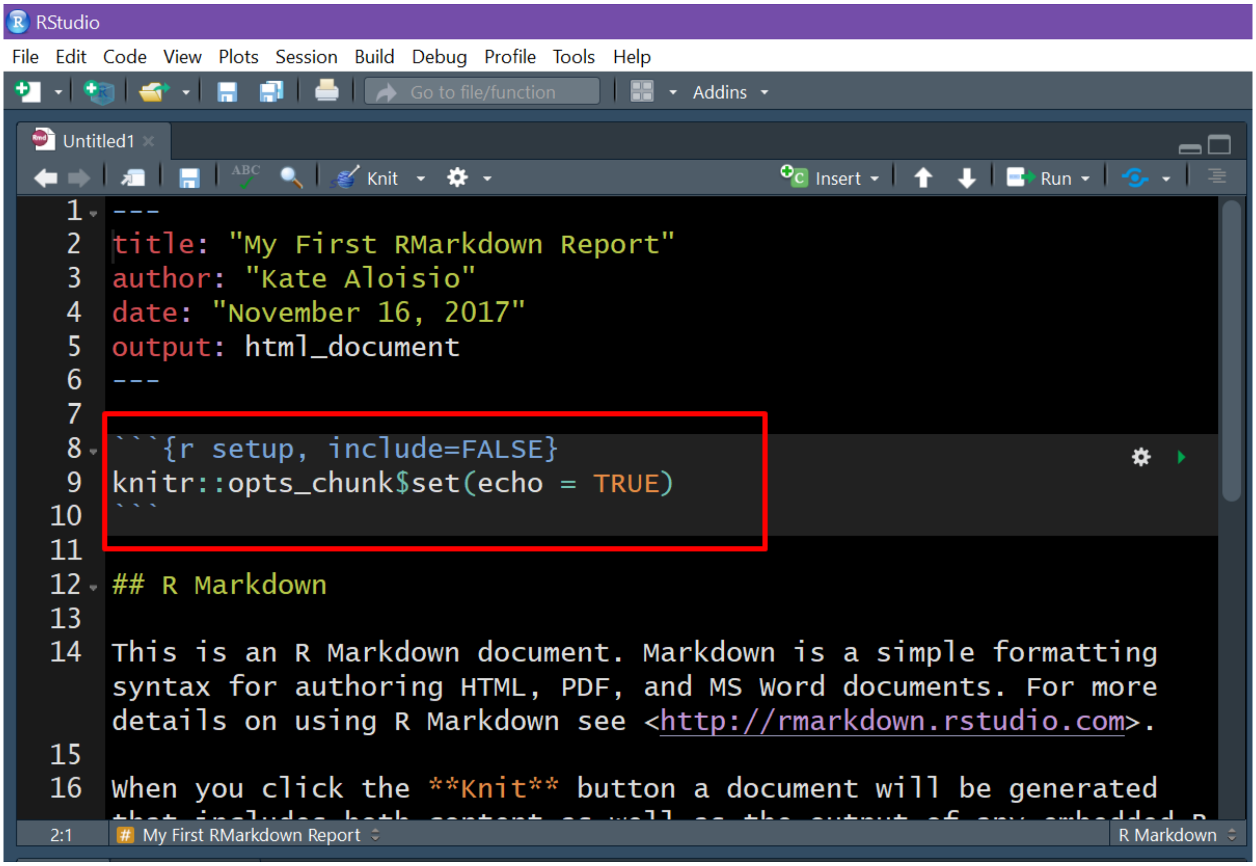


Figure 5: Code Chunk

and text:

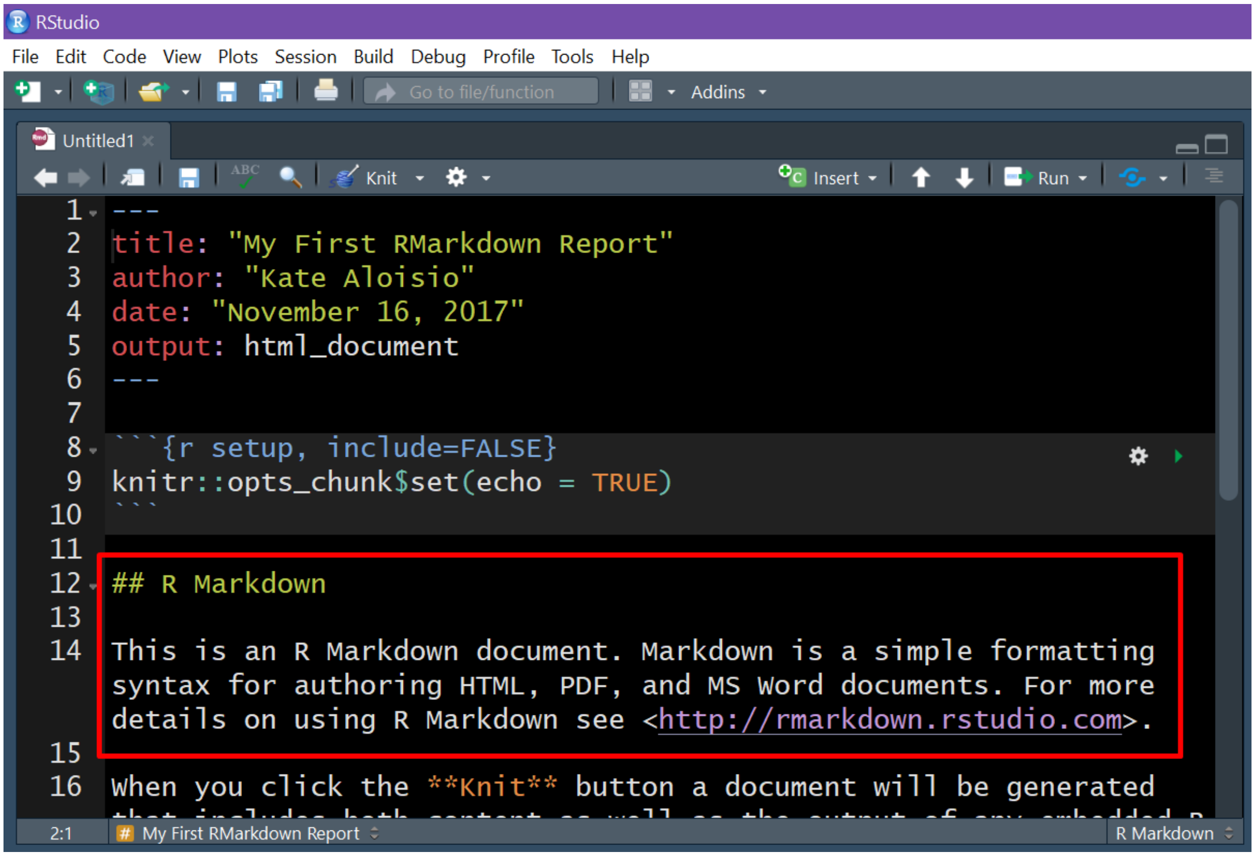


Figure 6: Markdown text

You'll see that you can add hyperlinks, tables, bold and other text types. To find out more go to Help -> Markdown Quick Reference

To compile the file click the knit icon:

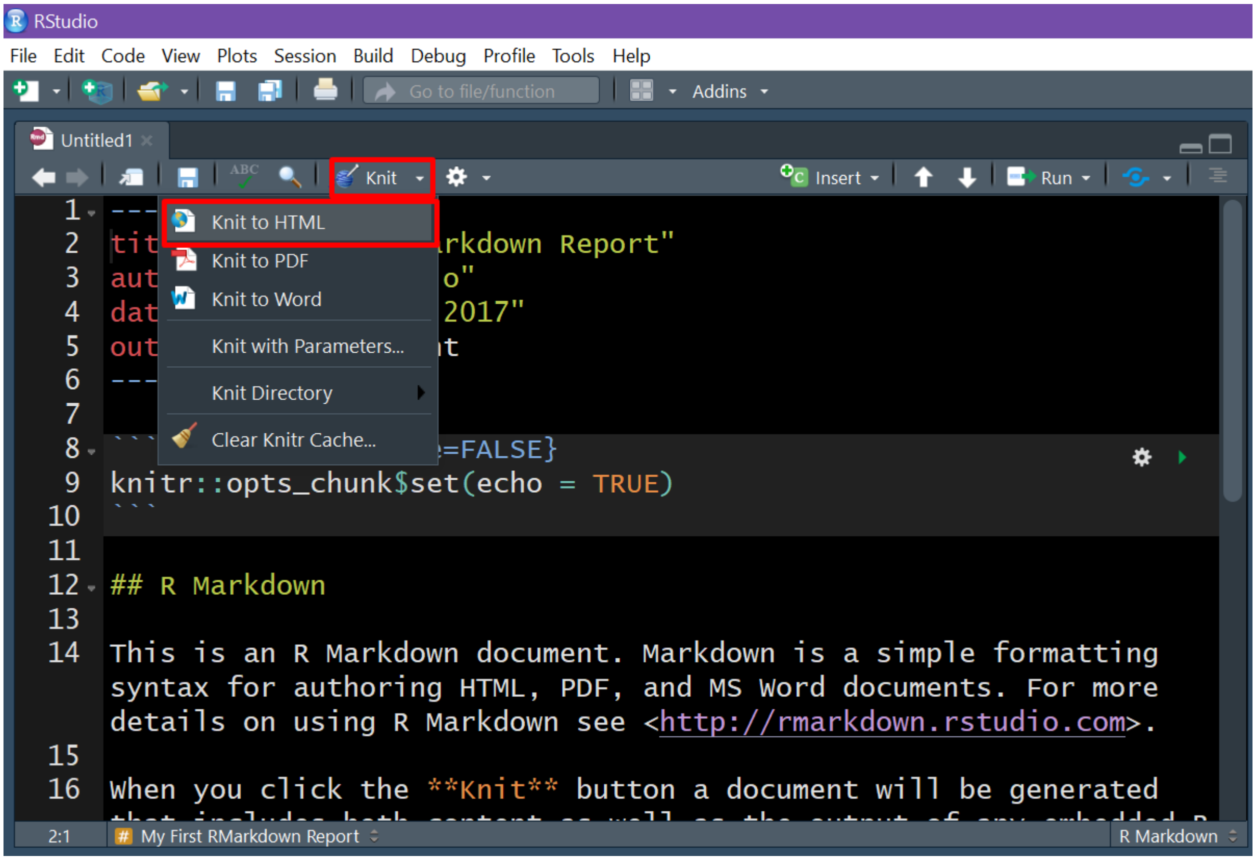


Figure 7: Knit

Note that from this menu you could also knit to pdf (which requires LaTeX) or work (which requires Microsoft Word).

After you click knit to html, it will prompt you to save the file, you only have to do this once. The results will then appear in the viewer or you could open the file from your file explorer.

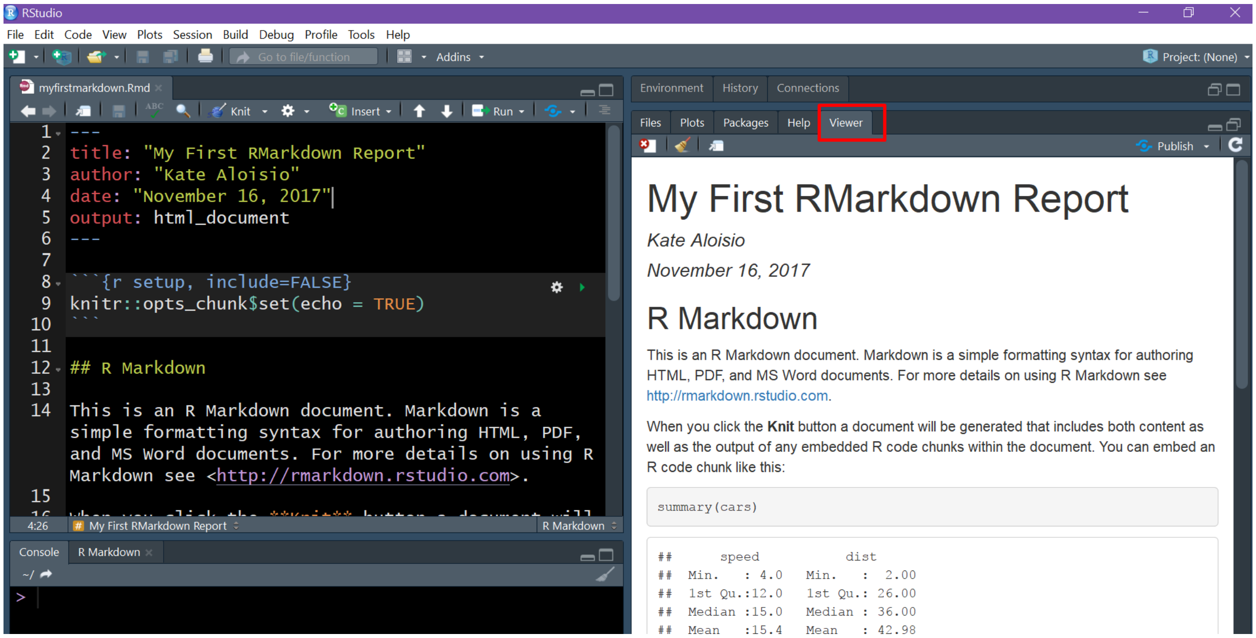


Figure 8: Viewer

You can also start from scratch. Go ahead and delete everything in the document.

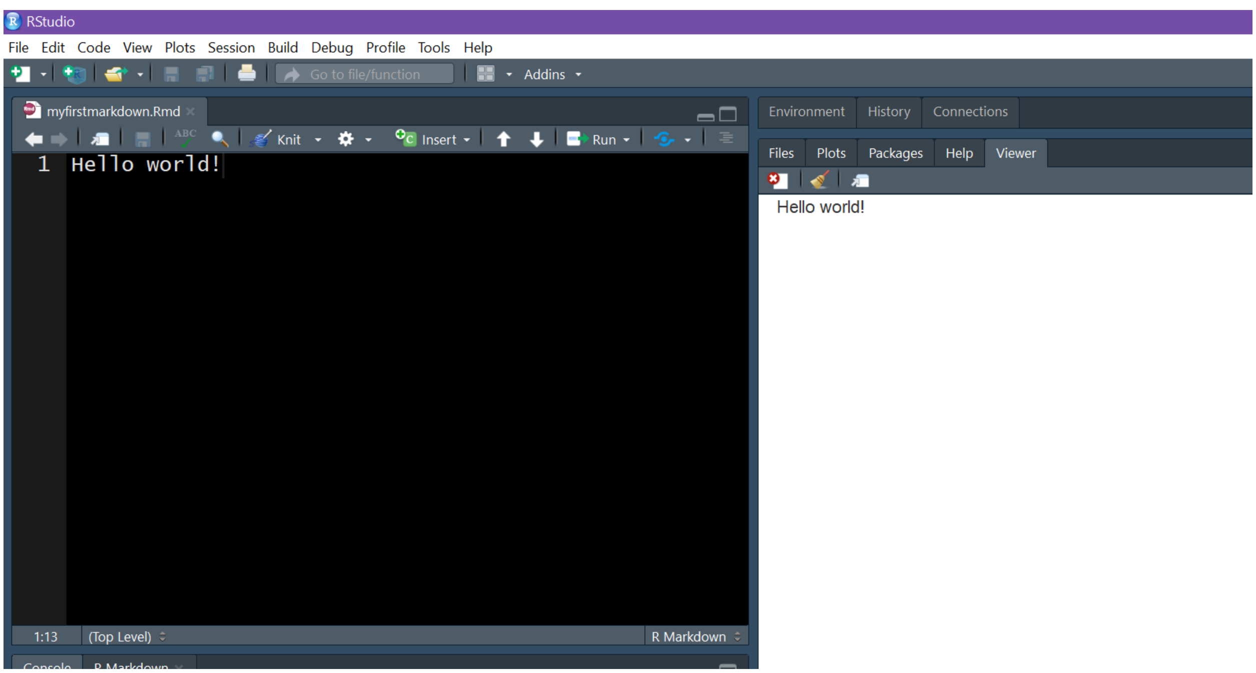


Figure 9: Empty Document

Note that it is perfectly okay to knit an empty document. You can also knit the document without a header, it will just use it's default settings.

Let's add back in the header. Notice that you can even change the theme of your report to add that extra customization. Here we are using the readable theme.

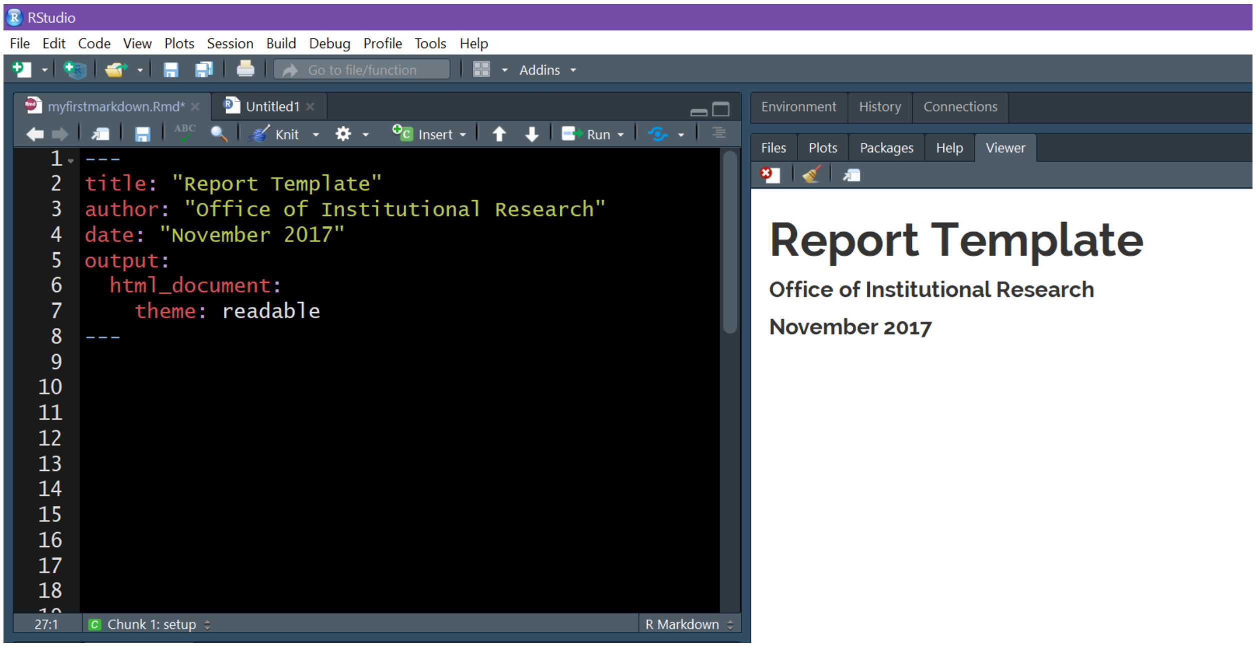


Figure 10: Add a header

Most RMarkdown files will then have a setup chunk. This is where I like to library packages and read in data. By using the option `include = FALSE` we are ensuring that none of this chunk will appear in the report.

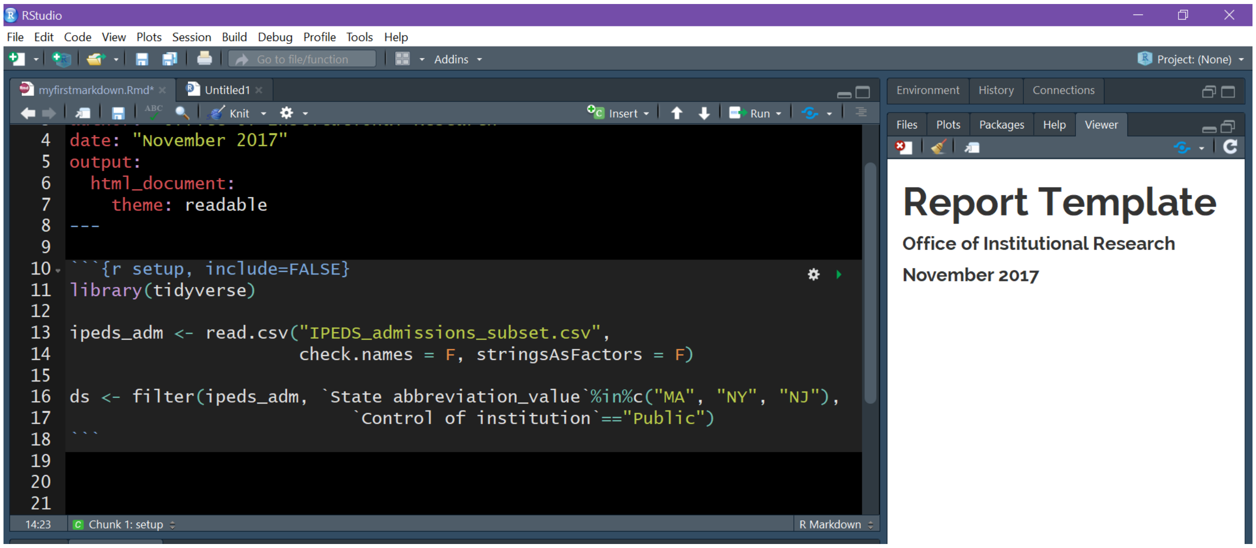


Figure 11: Add a setup

Now feel free to add some text and insert a code chunk for our plot 1.

You can insert a code chunk by clicking on the Insert button. Or you can use the shortcut ctrl+alt+i

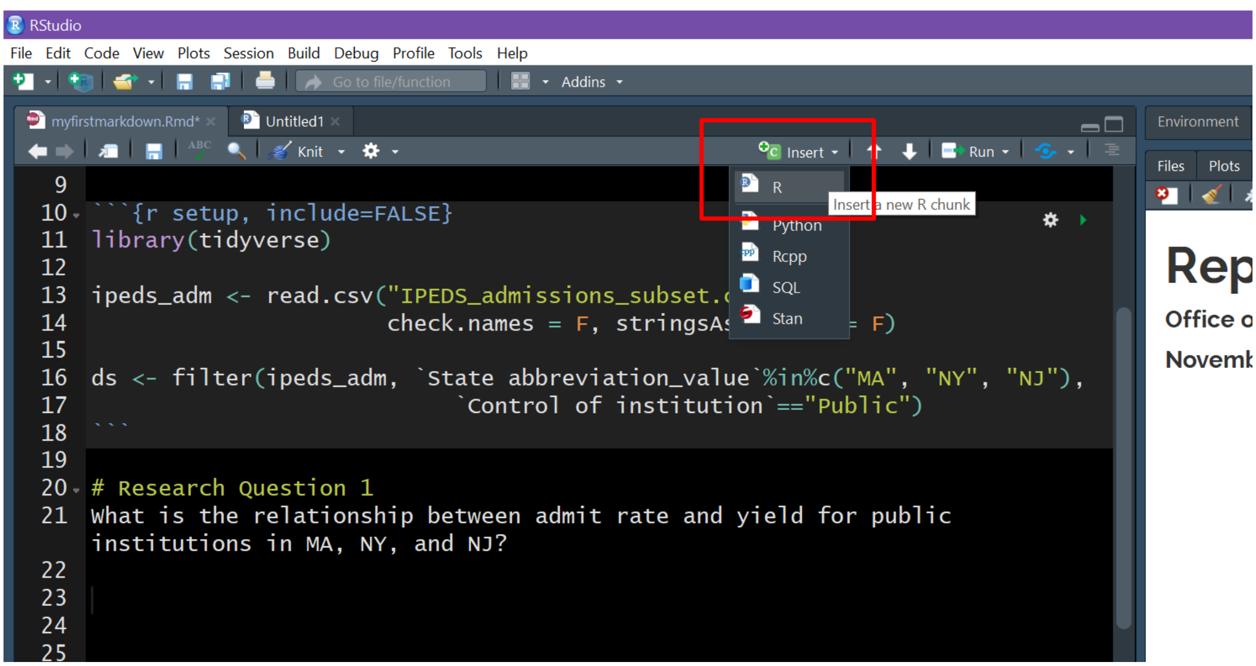


Figure 12: Add a R chunk

Next, copy and paste p1 into this chunk. If we compiled the report as is, it would also include the code as seen below:

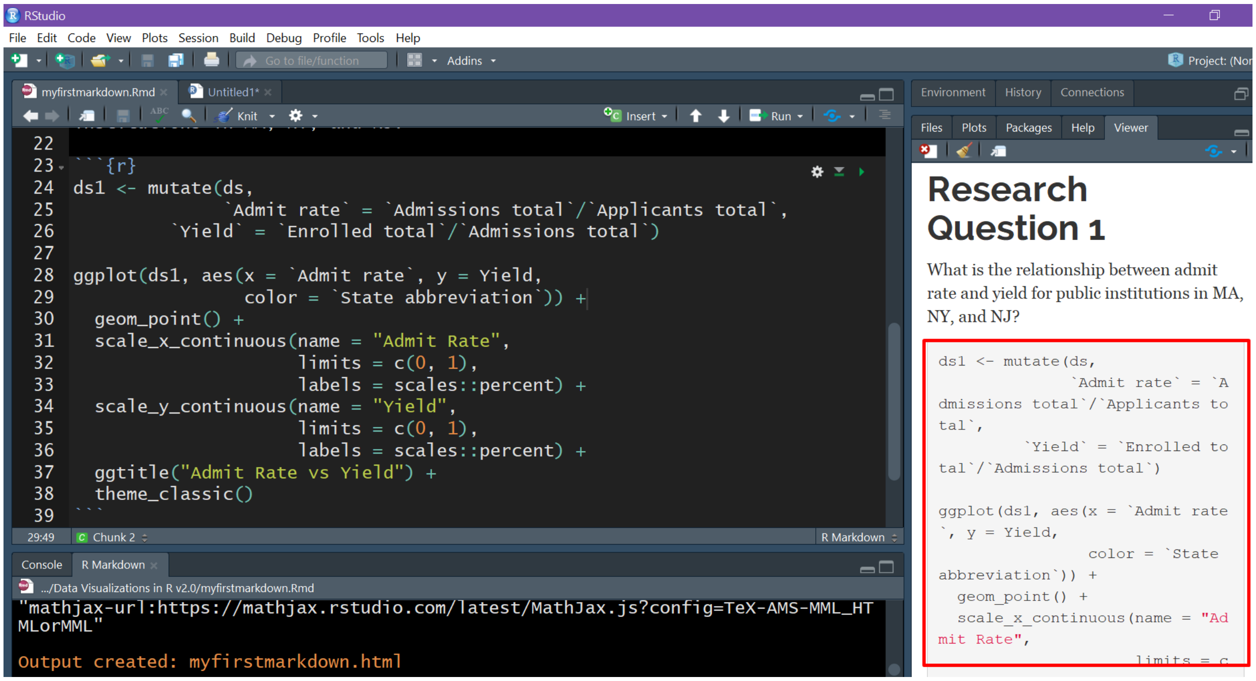


Figure 13: Compile with echo

This is gross. Include the option `echo = F` to prevent the code from appearing in the output.

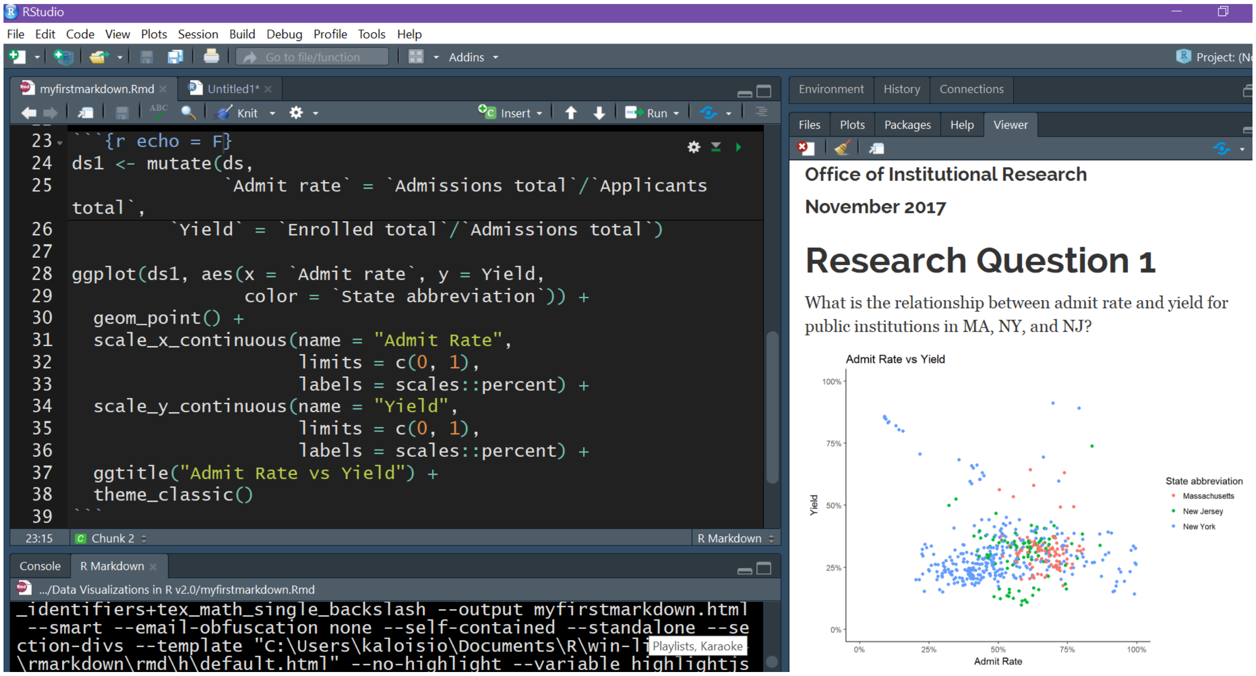


Figure 14: Compile without echo

You can continue this process until your report is ready. Make sure you save your work and knit before closing your file.

In your materials is a more documented example of a completed report.

Now you have a document that you can share with the world and it is completely reproducible!