Graphics and Plotting in R

Ruijie Zhou

10/29/2017

Graphics and Plotting in R

Introduction

The purpose of this post is to discuss certain features asscoiated with creating plots and graphs with R and RStudio. More specifically, this post mainly focuses on the plotting features of the R built-in functions and the external package of `ggplot2` . The posts would mainly consist of plotting and graphing examples from both the R built-in functions and the `ggplot2` package, and some comparisons between the 2 ways of creating plots and graphs. This post serves as a summary and certain extensions to what I have learned about graphing and plotting in R in the past few weeks. The first half of the post would primarily focus on summarizing what we have already seen in class, and the second half would be new, interesting examples that I have seen during my research of the topic.

Motivation and Background

Taking the Stat 133 class is the first time that I come in contact with the functional programming language R. Ever since the first homework of the class, I was greatly intrigued by R's incredible ability to create graphs and plots so conveniently, especially with the help of the R markdown file, so that the graph could easily be displayed. Therefore, I would like to take some time here to discuss what I have learned in the past few weeks about creating graphs and plots using the 2 main tools, the R built-in plotting functions and the `ggplot2` package, in this post. At the end of the post, I hope to demonstrate how flexible the `ggplot2` package can be in terms of creating plots and graphs.

Data Preparation

I will be performing some setup for further use in the post in this section. For convenience, I'll be primarily working with the `landdata-state.csv` file, which is referenced from the Introduction to R website (See Reference). I will also be working with the `nba2017-teams.csv` file, which was generated during homework 3 as part of the assignment and contains a lot of information regarding NBA teams. Using this file is primarily for reflecting on what we have already learned in the class. All the data files could be located in the data folder associated with the post.

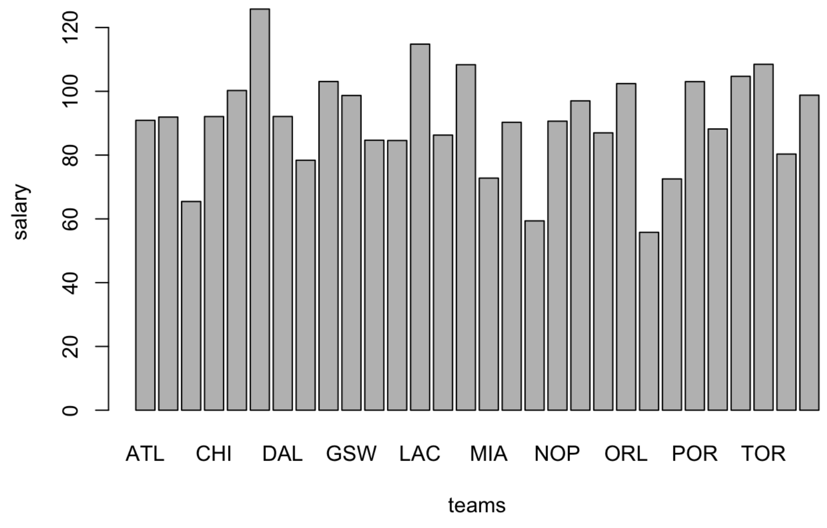
```r
# Importing the ggplot library
library(ggplot2)

# Importing the data file
teams <- read.csv(file = "../data/nba2017-teams.csv", stringsAsFactors = FALSE)
housing <- read.csv(file = "../data/landdata-states.csv", stringsAsFactors = FALSE)

# Create a simple dataset for demonstration of pie charts in base R
simpleHousing <- data.frame(
  houseCount <- c(10, 12, 4, 16, 8),
  region <- c("West", "East", "North", "South", "NA")
)
```

Basic Graphics and Plots Examples

Barplots with R built-in functions

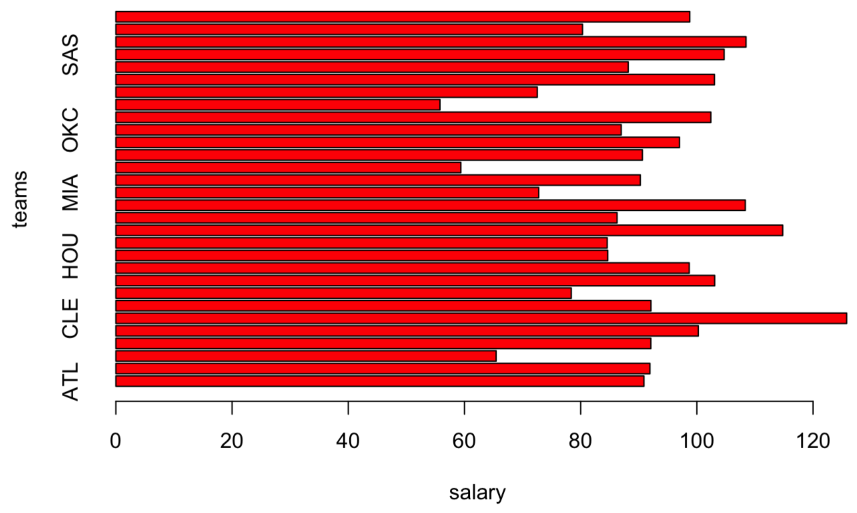
One of the most common graphs that we have created during the first few weeks of the class is barplots. And one of the direct way to create barplots is to use the built-in `barplot` function. In this function, the most essential parameter `height` is either a vector or a matrix. If `height` is a vector, then the values determine the heights of the bars in the plots. If `height` is a matrix, then each bar of the plot corresponds to a column of height, with the values in the column giving the heights of stacked or juxtaposed "sub-bars" given `besides` is FALSE or TRUE. It also has `names.arg` to label the bars. I'll be going into more details into the use of `besides` in the next section of more complex graphs. The `horiz = TRUE` creates a horizontal bar. The other parameters such as `xlab` , `ylab` , `xlim` , `ylim` are common among all the plotting functions. Below is a rather simple barplot that demonstrates the teams and their respective total salaries.

```r
barplot(teams$salary, names.arg = teams$team, ylab = "salary", xlab = "teams")
```



We could also make the graph horizontal. This time, we can also add some colors to the graph with the `col` argument in the `barplot` function

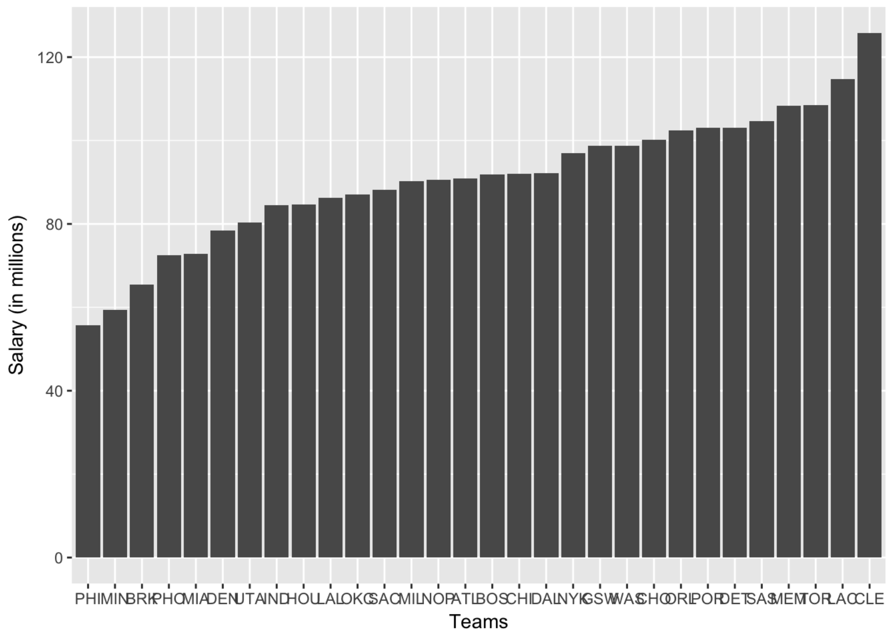
```
barplot(teams$salary, names.arg = teams$team, horiz = TRUE, xlab = "salary", ylab = "teams", col = "red")
```



Barplots with `ggplot2`

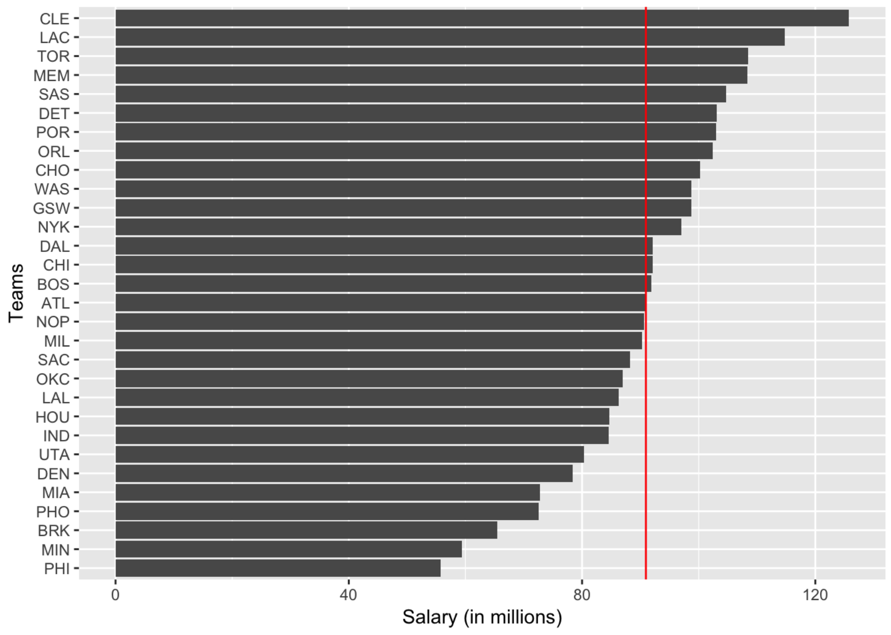
As we can see from above, creating barplots with the built-in `barplot` functions is fairly straightforward; however, we can see that the simple barplots created are still left wanting in that: 1) the names of all the teams are not fully displayed, 2) the bars are not ordered. Undoubtedly, there are ways to work around these little problems with the R built-in functions, but `ggplot2` provides us with a more convenient solution to create more elegant and delicate graphs. In the following example, we simply use `aes()` to specify the x-coord to be the teams, and use the `reorder` function to order the x-coord in accending order of the total salary. The names of all the teams are automatically displayed. We cannot simply use the `reorder` function in the built-in `barplot` because the `height` argument has to be a vector. So one way to work around this is to manually reorder the teams before applying the `barplot` function, but this is significantly more work than the `ggplot` solution, which is shown below: we can also use the `labs` function in the `ggplot2` package to add labels to the graph. The `geom_bar(stat = 'identity')` specifies that the heights of bars are used to represent the values in the data.

```
ggplot(data = teams, aes(x = reorder(team, salary), y = salary)) +
  geom_bar(stat = 'identity') +
  labs(x = "Teams", y = "Salary (in millions)" )
```



We could also make the above graph horizontal. The only thing we need to add is the `coord_flip()` function. Depending on where we insert the `coord_flip()` function, we may need to exchange the label names between x-coord and y-coord because the coordinates are flipped. In the example below, there is no need to change the labels since `coord_flip()` comes at the very end. We could also add a line in the x-axis that demonstrates the avearage of the salaries among the teams. To achieve this, we simply need the `geom_hline()` function added to the command, see example below:

```r
ggplot(data = teams, aes(x = reorder(team, salary), y = salary)) +
  geom_bar(stat = 'identity') +
  labs(x = "Teams", y = "Salary (in millions)" ) +
  coord_flip() +
  geom_hline(yintercept = mean(teams$salary), color = "red")
```

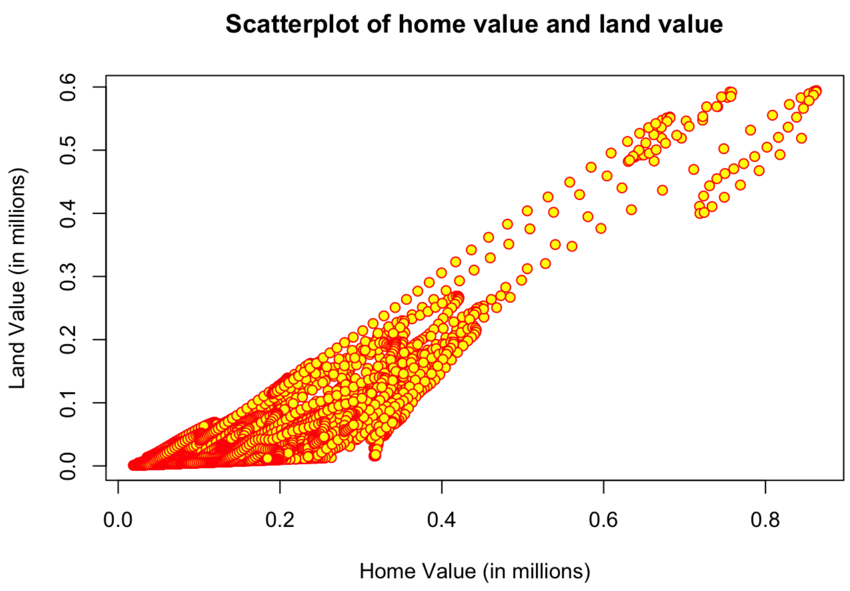


As we can see from above, this almost looks like what we have created in homework 3 where we have to create the horizontal barplot that ranks the NBA teams by total salary, except for the title, which could be simply added by the `ggtitle` command. This is a good example that demonstrates how `ggplot` could be useful in plotting more complex graphs. I will cover more about this later in the next section.

Scatterplots with R built-in functions

Another kind of plots that we have been dealing a lot with is the scatterplot. To introduce something new into this post, I will be working with the `landdata-state.csv` data set from this point. For instance, with the help of the base scatterplots, we could look into the relationship between the home value and the land value using the `plot` function from the base R functions. Despite the fact that this function seems quite simple, there are still a few things that I would like to reiterate regarding the parameters of this function. The `x` and `y` are self-explanatory. The `pch` specifies the shape of the points, and there are 25 to choose from. I use 21 in the following example since 21 gives a background color in the parameter `bg`. `cex` controls the size of the points. `col` specifies the color of the edge of the points. The other parameters are fairly standard, and the example is shown as follows:

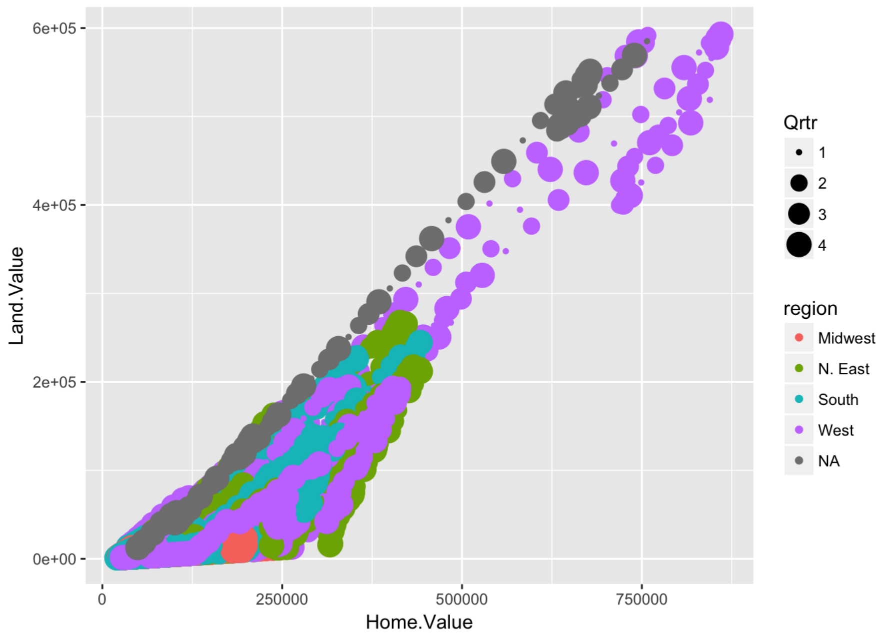
```
plot(housing$Home.Value / 1000000, housing$Land.Value / 1000000,
     pch = 21:21,
     col = "red",
     bg =  "yellow",
     cex = 1,
     xlab = "Home Value (in millions)",
     ylab = "Land Value (in millions)",
     main = "Scatterplot of home value and land value")
```



Scatterplots with `ggplot2`

Similarly, the above scatterplot could be easily created with functions from the `ggplot2` packages. Additionally, the `ggplot2` package enables us to create more informative plots by, for instance, specifying the color of the point with respect to the region of the hosue, and specifying the size of the point with respect to the quarter of the year in which the house is sold. Here, we use the `geom_point()` to create scatterplot. And we use `color = region`, `size = Qrtr` to add more information into the graphs.

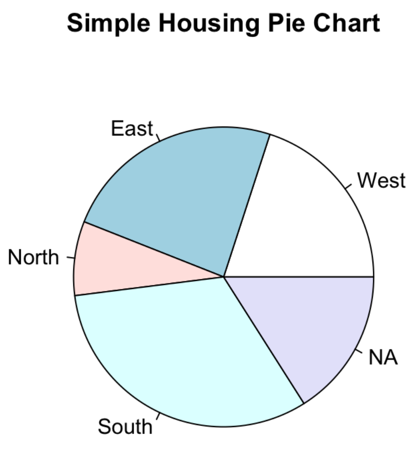
```
# Sized and colored scatterplot using ggplot
ggplot(data = housing, aes(x = Home.Value, y = Land.Value)) +
  geom_point(aes(color = region, size = Qrtr))
```



Pie Charts with R built-in functions

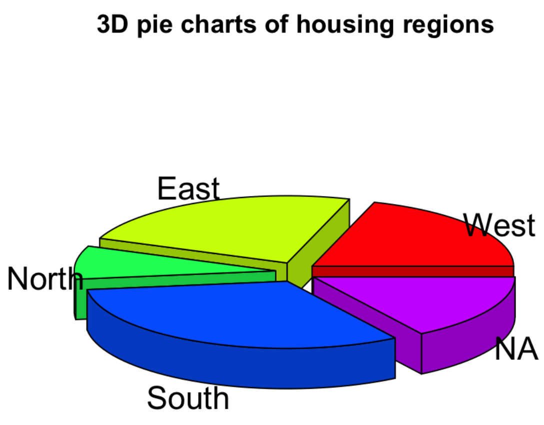
So far we have reflected on the 2 kinds of graphs that have been addressed in the past few week in class, and have created some new examples. Now I will be examining a new kind of graph, the pie chart, which is a really useful kind of plot, and we have not covered too much about it in the class materials. First of all, we could look at a simple pie chart example that is referenced from the Quick-R site. For convenience, I have created a copy of the simple data set used in the example in the Data Preparation section above. The simple pie chart utilizes the `pie()` function from the built-in R functions, shown below:

```r
pie(simpleHousing$houseCount,
    labels = simpleHousing$region,
    main = "Simple Housing Pie Chart")
```



A more interesting example would be looking at the 3D pie chart of such a graph. Note that the function `pie3D()` used in the below example is not actually from the base R functions, but from a package named `plotrix`. I have included this example here mainly because it seems rather interesting, the syntax is very similar to the base function `pie()`, and the graph generated seems very nice. The `explode` argument here specifies the degree of the seperations between each sector of the pie:

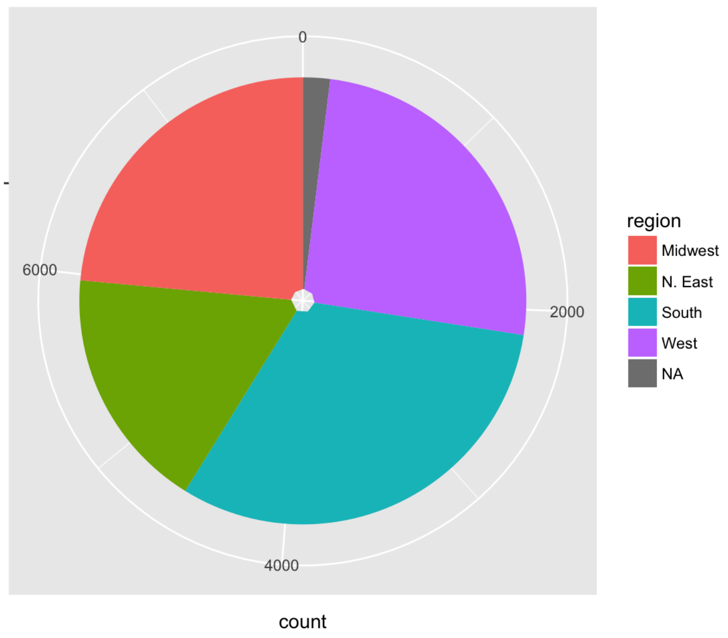
```r
# Import the library
library(plotrix)

pie3D(simpleHousing$houseCount,
      labels = simpleHousing$region,
      explode = 0.1,
      main = "3D pie charts of housing regions")
```



Pie Charts with `ggplot2`

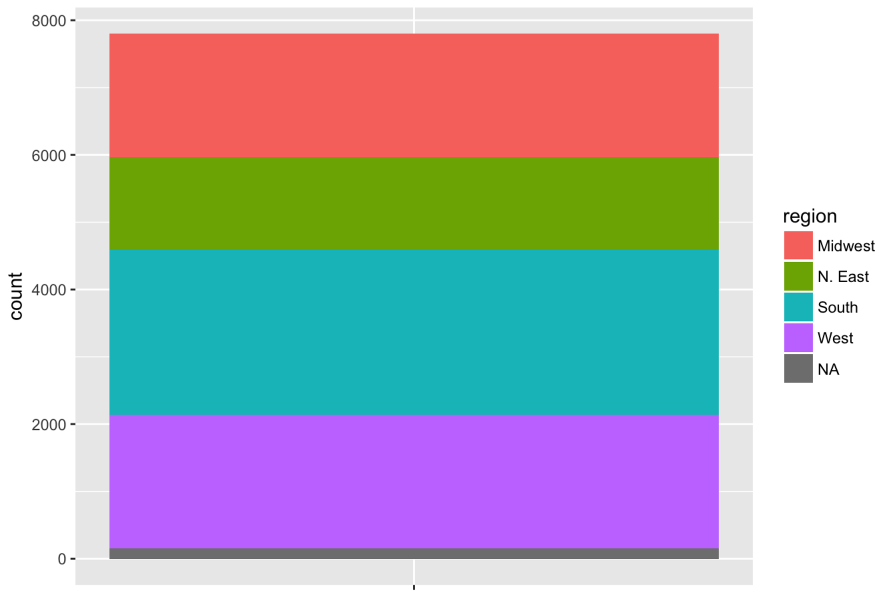
Similarly, such pie charts could be easily created with the `ggplot2` package. Here we can take a look at the example of the actual count of the homes with regard to its region from the `landdata-state.csv`. Because there is really no varaible for the x-axis, I will just use `factor()` to compute an empty variable on the fly. The reason that I did not do this for the pie charts using the base R functions is that the `x` vector in the `pie` function does not accept `factor()` as an input. Computing the counts seperately would also be extra work, so this is another example that demonstrates the flexibility of `ggplot2`. Here is the simple example:

```
ggplot(housing,
  aes(x = factor(""), fill = region) ) +
  geom_bar() +
  coord_polar(theta = "y") +
  scale_x_discrete("")
```



Note that what is interesting here is that we did not use a command of the form `geom_pie()`, but we still use the `geom_bar()` that is actually used to create bar charts in `ggplot`. In fact, there does not really exist a command called `geom_pie()` designated to create pie charts. The reason that we could create a pie chart with `geom_bar()` is that we change the coordinate system to polar from cartesian. By simply changing the coordinate system to polar, we can create beautiful pie charts from bar charts without too much work. This may seem complicated at first, but the notation becomes quite intuitive if you really think about it. We could look at the above example before changing the coordinate system to polar:

```
ggplot(housing,
  aes(x = factor(""), fill = region) ) +
  geom_bar() +
  scale_x_discrete("")
```



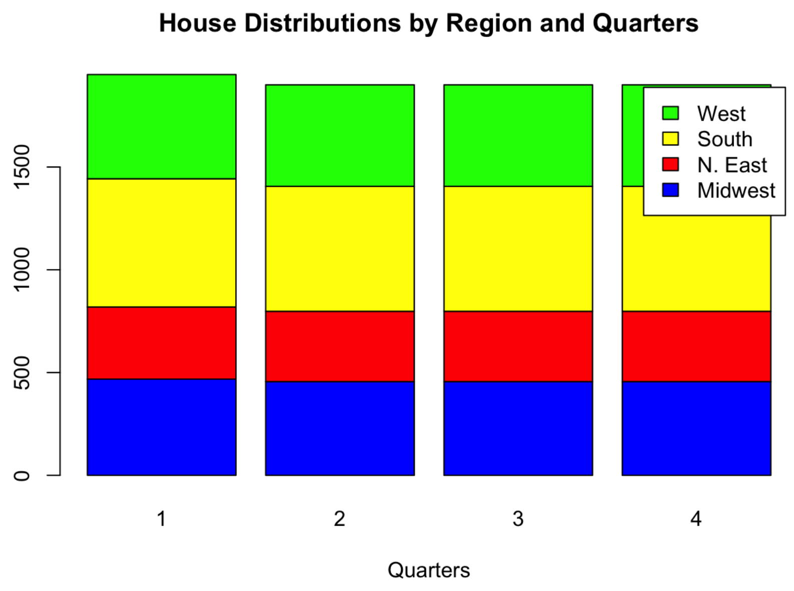
Here we have the orignal barcharts of the above pie charts. And the conversion between the two form only require a simple change of coordinate system. This again demonstrates how flexible the `ggplot2` pacakge can be.

More Complex Graphs

Barplots Revisited

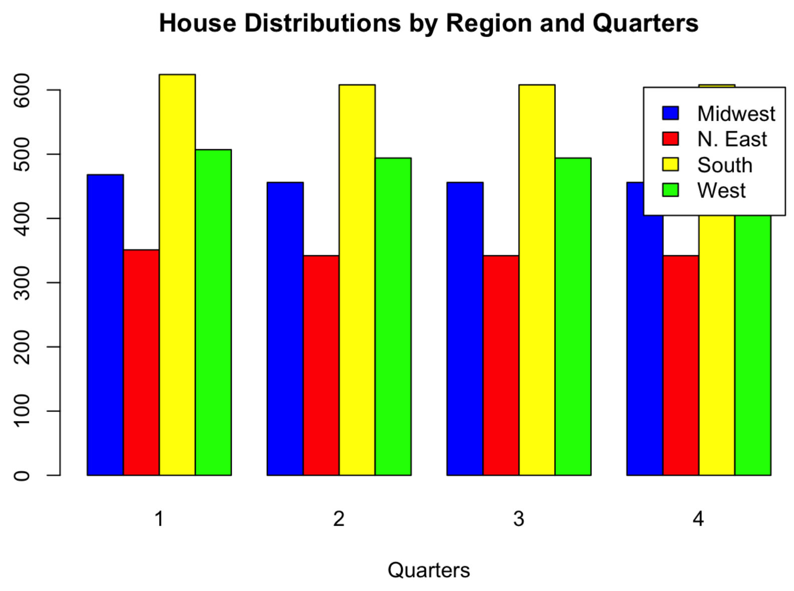
Here we will take a look at the creation of more complex graphs. For starters, we will look at a few examples with R base functions. In the following examples, I'll be generating the stacked and the grouped barplot of the house distributions by its region and the quarter in which the house is sold. Note that we use the `legend` argument to display the correspondance between the color and the its repective region.

```
counts <- table(housing$region, housing$Qrtr)

barplot(counts, col=c("blue", "red", "yellow", "green"), main = "House Distributions by Region and Quarters", xlab
= "Quarters", legend = rownames(counts))
```



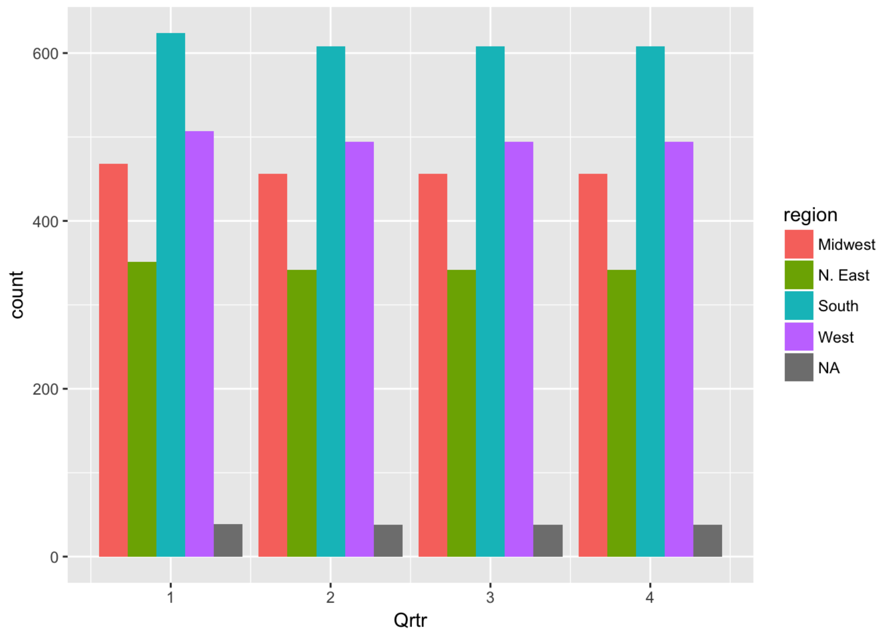
The counts created here serve as a matrix, which is meant to be used with the argument `besides`. The use of `besides` is demonstrated in the last section. If we set it to be TRUE, then we would obtain a grouped (juxtaposed) barplots as follows:

```
barplot(counts, col=c("blue", "red", "yellow", "green"), main = "House Distributions by Region and Quarters", xlab
= "Quarters", legend = rownames(counts), beside = TRUE)
```



The same graphs could also be created using functions from `ggplot`. The syntax is similar to simple barplots. We only need to specify the postion of the bars in `geom_bar()`. If `position` is set to stacked, then it is a stacked graph. If `position` is set to dodge, then it is a grouped graph. I'll be only displayed the grouped graph to save space since they are pretty much the same as the ones above:

```
ggplot(housing, aes(Qrtr, fill = region) ) +
  geom_bar(position = "dodge")
```

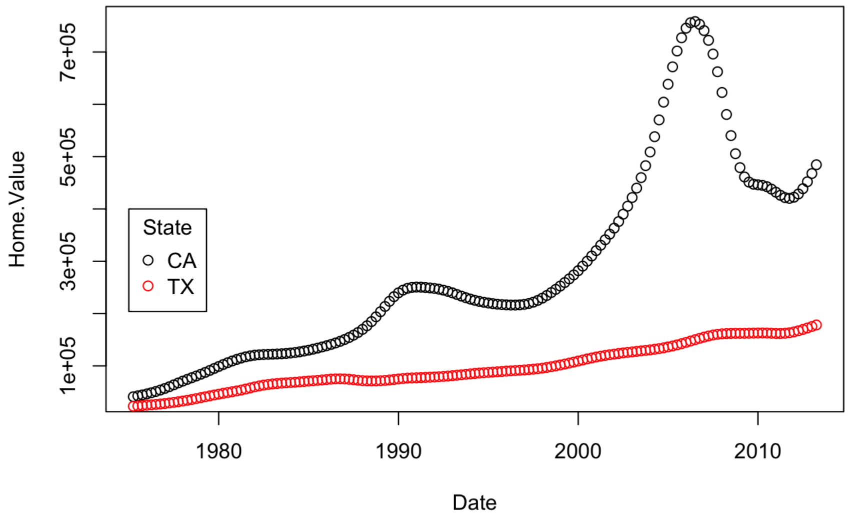


```
# Change position to "stacked" if statcked barplot is wanted
```

Scatterplots Revisted

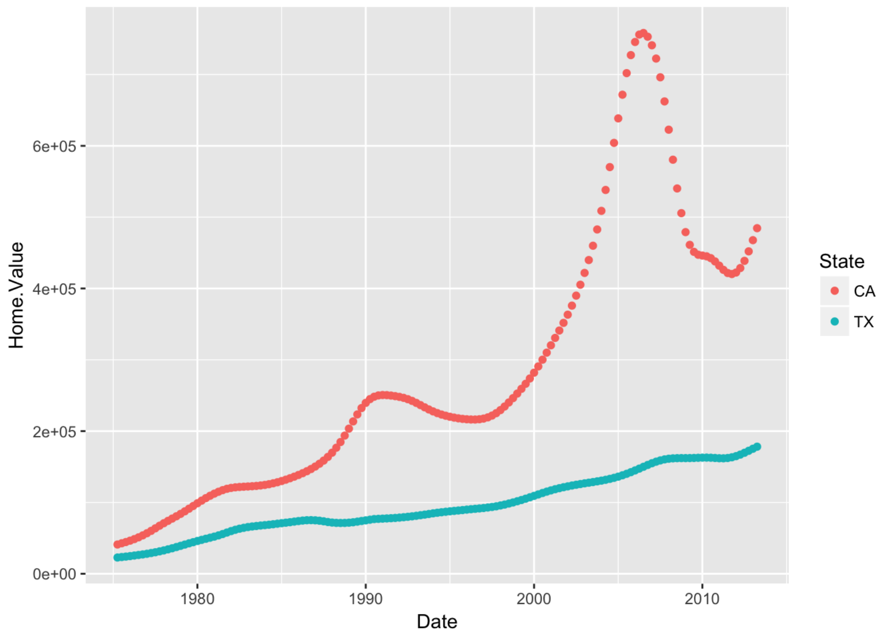
Next we will be revisiting scatterplots. This time, we want to look at the house values distribution within the states of, say California and Texas, by year. The original example is from Introuction to R Graphics. (See Reference). Firstly, we construct such a graph using the `legend` functions that we have just examined above. Here we use `points` instead of `plot` for the second state so that the 2 graphs could be shown in the same plot. We specify the starting year to be 1975 in the `legend` function and choose CA to be black and TX to be red:

```
plot(Home.Value ~ Date,
     data=subset(housing, State == "CA"))
points(Home.Value ~ Date, col="red",
       data=subset(housing, State == "TX"))
legend(1975, 400000,
       c("CA", "TX"), title="State",
       col=c("black", "red"),
       pch=c(1, 1))
```



The similar example could be created with `ggplot`. Here we use the pipeline structure that is demonstrated in the `dplyr` package to select the state that we want. The notation looks like `%in%`, followed by the vector of states, which are CA and TX in this case.

```
ggplot(subset(housing, State %in% c("CA", "TX")),
       aes(x=Date,
           y=Home.Value,
           color=State))+
  geom_point()
```

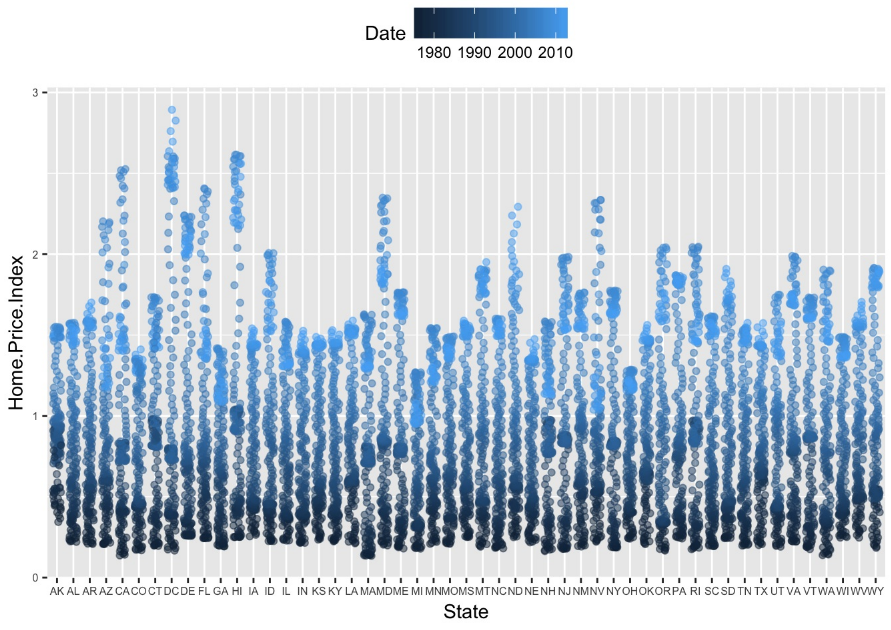


We can see that if we want to add more information to the graph, using `ggplot` would be a much better option as shown by the length and clarity of the code. For instance, if we want to check more states' house value distribution by year, we would have to add another line of `points(...)` to include a new state in the base R function method; however, we simply need to add another state name in the vector in the `ggplot` method.

Scale Modification Examples

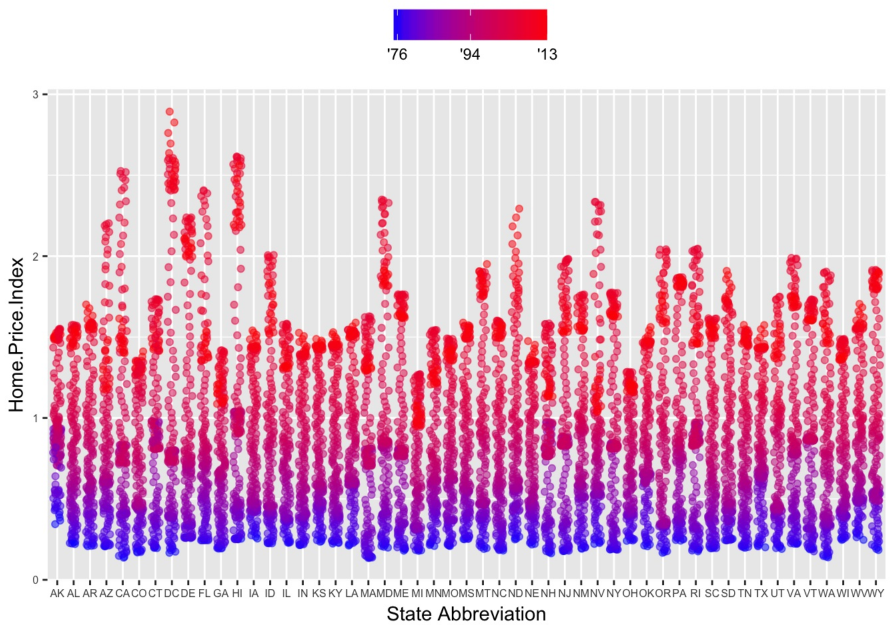
This example also comes from Introduction to R graphics. I think it is an excellent example that demonstrates how `ggplot2` handles complex plots elegantly. Here we want to examine the distirbution of home values by date and state. The `theme()` function here to modify individual compoenents of a theme. Here we use `theme()` to adjust the position of the `legend` in the graph to be on the top, and to also adjust the text size. Here we base the color of the dots on date, and we use `position_jitter()` to add random noise to the plot so that it can be easier to read. Jittering is particularly useful for small datasets with at least one discrete position. See the example below:

```
p1 <- ggplot(housing,
             aes(x = State,
                 y = Home.Price.Index)) +
        theme(legend.position="top", axis.text=element_text(size = 6))
p2 <- p1 + geom_point(aes(color = Date),
                      alpha = 0.5,
                      size = 1.5,
                      position = position_jitter(width = 0.25, height = 0))

p2
```



We can further this example by changing the color of the high and low values (low and high corresponding to the year). We set the high values to be red and the low values to be blue so that there are more distinctions within the graph. Here we use `scale_x_discrete` to add the label for the x-axis. We use `scale_color_continuous` to differentiate between the high and low values. This function follows the gradient color scales, we just need to set the breaks and color properly for it to work. Here the breaks are set to be year 1976, 1994, 2013. After creating a proper label for the breaks, we would have the example below:

```
p2 +
  scale_x_discrete(name="State Abbreviation") +
  scale_color_continuous(name="",
                         breaks = c(1976, 1994, 2013),
                         labels = c("'76", "'94", "'13"),
                         low = "blue", high = "red")
```

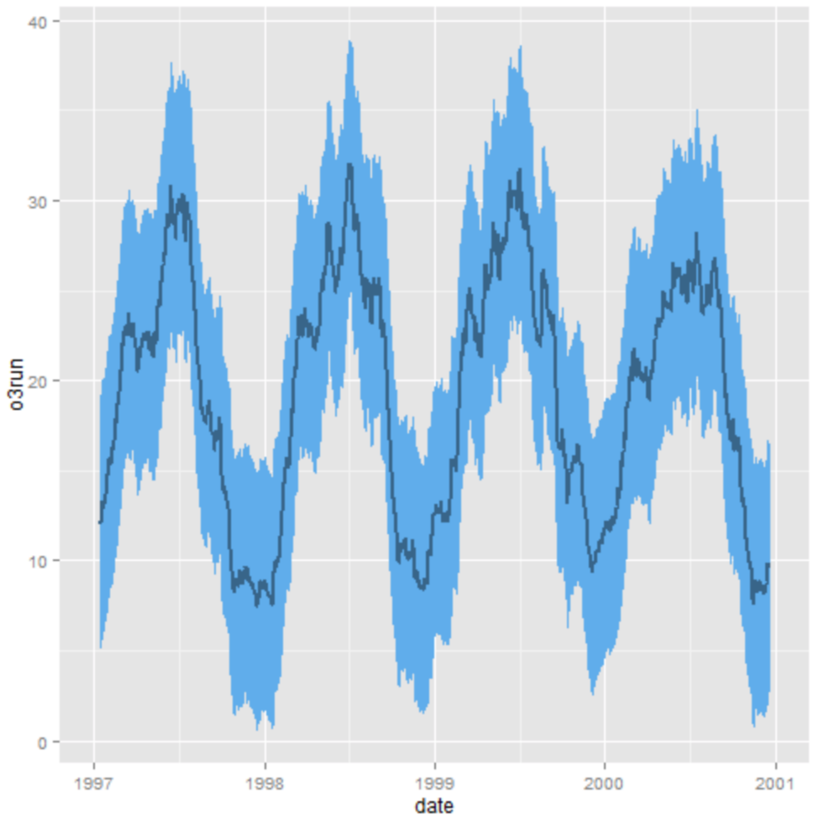


This also demonstrates the convenience of using `ggplot2` to create graphs. We could continuously add more constraints or other conditions to the existing graph by the `+` operator, which is not possbile if we are using R base functions.

Other Graphs with `ggplot`

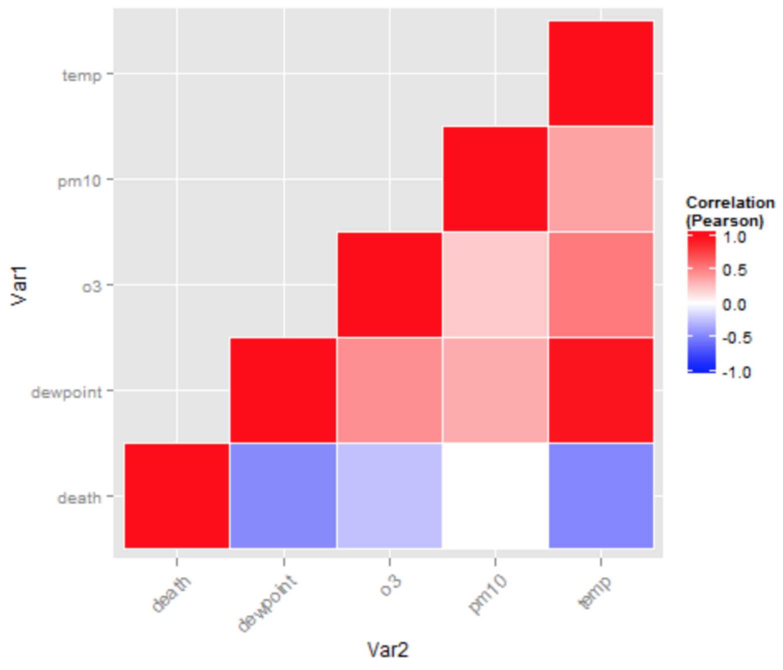
There are tons of other features and tricks in creating graphs and plots using the base R functions and the `ggplot` package. Here I have attached a couple of plots that might be interesting to look at created using `ggplot`. The images could be located in the images folder associated with the post.

```
nmmaps$mino3<-nmmaps$o3run-sd(nmmaps$o3run, na.rm=T)
nmmaps$maxo3<-nmmaps$o3run+sd(nmmaps$o3run, na.rm=T)

ggplot(nmmaps, aes(date, o3run))+geom_ribbon(aes(ymin=mino3, ymax=maxo3),
  geom_line(color="steelblue4", lwd=1)
```



Standard Deviation Ribbon

```
ggplot(thecor, aes(Var2, Var1))+
 geom_tile(data=thecor, aes(fill=value), color="white")+
 scale_fill_gradient2(low="blue", high="red", mid="white",
  midpoint=0, limit=c(-1,1),name="Correlation\n(Pearson)")+
 theme(axis.text.x = element_text(angle=45, vjust=1, size=11, hjust=1))+
 coord_equal()
```



Tiled Correlation Plot

Conclusion

This post mainly focuses on some basic techniques in creating graphs and plots using the R base functions and the `ggplot2` package functions by using examples starting from simple ones to more complicated ones. As demonstrated by the examples above, it is clear that while the R base functions provide us with some nice graphing and plotting features, they are, sometimes, quite complicated to use and sometimes need a lot of work, especially when we need to view data sets in groups. The `ggplot2` package, on the other hand, provides us with a higher degree of flexibility in creating graphs and plots more coveniently and efficiently. This concludes my learning experience of graphing with R in the past few weeks. I hope the reader of this post would find some of the examples to be helpful.

References

- Quick-R: https://www.statmethods.net/graphs/bar.html
- Quick-R: https://www.statmethods.net/graphs/pie.html
- Quick-R: https://www.statmethods.net/graphs/pie.html
- r4stats: http://r4stats.com/examples/graphics-ggplot2/
- Introduction to R graphics: http://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html
- Introduction to R graphics dataset: http://tutorials.iq.harvard.edu/R/Rgraphics.zip
- Simple ggplot examples: http://rstudio-pubs-static.s3.amazonaws.com/12581_042080eb6d9a498da1f7dc99238e2efc.html
- Beautiful Plotting in R: http://zevross.com/blog/2014/08/04/beautiful-plotting-in-r-a-ggplot2-cheatsheet-3/