Graphing with ggplot2 using employment and unemployment data in Illinois

Minu Sabu
School of Information Technology
Illinois State University
msabu@ilstu.edu

October 22, 2014

Introduction

One of the major factors that has played an important role throughout the ages and that impacts the well being of society is employment. It is defined as an activity in which one engages for paid work. The metrics that denotes the percentage of people employed to the total number of all individuals is called the employment rate. On the other hand, unemployment occurs when people are without work and actively seeking work. The percentage of people unemployed to the total number of all individuals is called the unemployment rate.

Illinois is one of the major states in the Midwestern United States. The population of Illinois in 2013 was 12,897,625, making Illinois the fifth largest state in the USA. So employment and unemployment rates in Illinois have been the focus of many studies and are therefore interesting. Illinois has the third worst unemployment rate in the nation. This can be attributed to factors like decline in the construction and manufacturing industries.

To study employment and unemployment in Illinois, we are using the data from Quandl. Quandl is a search engine for numerical data. The U.S Bureau of Labor Statistics (BLS) is the government agency that maintains and publishes data regarding both employment and unemployment. BLS was started in 1884 with a focus on statistical data. Quandl collects employment and unemployment data from the BLS database. The rates of employment and unemployment are calculated from surveys of households performed throughout the state of Illinois. We will be using the OSEMN approach in data science. In this tutorial, OSEMN stands for Obtaining, Scrubbing, Exploring, More exploring, and graphiNg the data. Let us look at each step in detail.

Obtaining the data

We will begin by obtaining data from Quandl using Quandl API. API stands for Application Programming Interface and is a collection of programming instructions and standards. It enables computer programs to directly communicate with one another. Many software companies release their APIs to the public and allow developers to create applications powered by their service. All data in Quandl is accessible via Quandl API. To use Quandl API in R Studio, we have to first install the `Quandl` package using the `Install Packages` option in the `Tools` menu. Once the package is installed, we have to load the package in R using the `library` command.

```
#loading the Quandl package
library(Quandl)
```

There is a daily limit of 50 downloads, or API calls, from Quandl for all unregistered users. In order to have unlimited downloads, we have to register by creating a free account at www.quandl.com. Quandl provides an authentication code to each user which gives unlimited access. We use the `Quandl.auth` command to add the unique authentication code.

```
#Authentication code will be different for each user
Quandl.auth("K1ZsbnWM7k3sASX2x24S")
```

We can now collect data from Quandl and store it in a data frame. We use the `Quandl` function which pulls data using Quandl API. The argument in the function given below is the R library code for accessing the data from Quandl. This will download the data directly to R. Let us start with the employment data and store it into a data frame called *EmploymentRawData*.

```
#Obtaining employment data from Quandl using Quandl function
EmploymentRawData <- Quandl("BLSI/LAUST170000000000005",
                            trim_start="2012-01-31",
                            trim_end="2014-07-31")

#Show EmploymentRawData
head(EmploymentRawData)

##         Date   Value
## 1 2014-07-31 6142902
## 2 2014-06-30 6131016
## 3 2014-05-31 6048065
## 4 2014-04-30 6029562
## 5 2014-03-31 5989489
## 6 2014-02-28 5930142
```

We have a total of 31 observations and we are only viewing the first few using the `head` command.

In the same way, let us collect the unemployment data from Quandl and store it into another data frame called *UnemploymentRawData*.

```
  #Obtaining unemployment data from Quandl using Quandl function.
UnemploymentRawData <- Quandl("BLSI/LAUST170000000000004",
                              trim_start="2012-01-31",
                              trim_end="2014-07-31")
#show UnemploymentRawData
head(UnemploymentRawData)

##         Date  Value
## 1 2014-07-31 464335
## 2 2014-06-30 471760
## 3 2014-05-31 466981
## 4 2014-04-30 469132
## 5 2014-03-31 544674
## 6 2014-02-28 612988
```

Again, we have a total of 31 observations and we are only viewing the first few using the `head` command.

Scrubbing the Data

The two data frames *EmploymentRawData* and *UnemploymentRawData* need to be combined into a single data frame. But before we merge them, we have to do some cleaning, also called scrubbing. A careful examination reveals that the second variable in both the data frames is named "Value." This could cause duplicate variable names during merging. To avoid confusion, let us rename the two variables to specifically say "Employment" and "Unemployment."

We use the `rename` function from the `plyr` package to rename the variables. It is always a good idea to keep the raw data intact for future reference. Therefore, let us store the modified data in a new data frame.

```
#Renaming variable name in EmploymentRawData
EmploymentData<-plyr::rename(x=EmploymentRawData,
                             replace=c("Value"="Employment"))

#Renaming variable name UnemploymentRawData
UnemploymentData<-plyr::rename(x=UnemploymentRawData,
                               replace=c("Value"="Unemployment"))
```

The `x` argument in the above function specifies the data frame being renamed. The new variable names for the two data frames look like this:

```
#Column names for employment data
names(EmploymentData)

## [1] "Date"       "Employment"

#Column data for unemployment data
names(UnemploymentData)

## [1] "Date"         "Unemployment"
```

Now we have our two data frames ready to be merged. We will use the `merge` command to merge the two data frames to a single data frame called *CombinedData*. The `by` option will specify which field is used for merging them. Since "Date" variable is common for both the data frames, we will merge the data frames based on "Date."

```
#Merging the two data frames by "Date"
CombinedData<-merge(x=EmploymentData, y=UnemploymentData,
                    by="Date")

#show CombinedData
head(CombinedData)

##         Date Employment Unemployment
## 1 2012-01-31    5876820       635818
## 2 2012-02-29    5919039       623540
## 3 2012-03-31    5946426       593888
## 4 2012-04-30    5965440       557657
## 5 2012-05-31    5992902       560998
## 6 2012-06-30    6037349       626361
```

Therefore, scrubbing made it easier to read the merged data frame by giving relevant variable names.

Exploring the data

In this section, we will take a deeper look into our combined data. Our data is presented in a tabular format with three columns. The first column gives the date, the second column gives the number of people employed, and the third column gives the number of people unemployed.

Let us begin by determining the data type of *CombinedData* using the `class` function.

```
#Displaying the data type of the merged data set
class(CombinedData)

## [1] "data.frame"
```

So, we can see that *CombinedData* is a data frame. The `class` function can also be used to see the data type of each variable in the data frame.

```
#Displaying the type of each variable
class(CombinedData$Date)

## [1] "Date"

class(CombinedData$Employment)

## [1] "numeric"

class(CombinedData$Unemployment)

## [1] "numeric"
```

Cross checking with the data, we can see that our first variable stores the date and hence its data type is "Date". The second and the third variables have the number of people employed and unemployed respectively. So, they have "numeric" data type.

Now let us examine the structure of *CombinedData* using the `str` function. This function compactly displays the internal structure of an R object.

```
#Displaying the structure of the merged data set
str(CombinedData)

## 'data.frame': 31 obs. of  3 variables:
##  $ Date        : Date, format: "2012-01-31" "2012-02-29" ...
##  $ Employment  : num  5876820 5919039 5946426 5965440 5992902 ...
##  $ Unemployment: num  635818 623540 593888 557657 560998 ...
```

The above result explains that we have a data frame with 31 observations and 3 variables. It also explains the type and format of each variable in the data frame.

We can also examine the structure of each variable in the data frame separately.

```
#Displaying the structure for each variable
str(CombinedData$Date)

##  Date[1:31], format: "2012-01-31" "2012-02-29" "2012-03-31" "2012-04-30" ...
```

```
str(CombinedData$Employment)

##  num [1:31] 5876820 5919039 5946426 5965440 5992902 ...

str(CombinedData$Unemployment)

##  num [1:31] 635818 623540 593888 557657 560998 ...
```

Next, let us display the statistical summary of our data using the `summary` function. This function will give details like the minimum and the maximum allowable values, mean, median and quartiles for each variable in the data frame.

```
#Displaying the summary of the full data set
summary(CombinedData)

##       Date              Employment       Unemployment
##  Min.   :2012-01-31   Min.   :5876820   Min.   :464335
##  1st Qu.:2012-09-15   1st Qu.:5943117   1st Qu.:557958
##  Median :2013-04-30   Median :5976571   Median :576956
##  Mean   :2013-04-30   Mean   :5981266   Mean   :576552
##  3rd Qu.:2013-12-15   3rd Qu.:6016258   3rd Qu.:618264
##  Max.   :2014-07-31   Max.   :6142902   Max.   :668919
```

As we can see, the `summary` function gives the statistics of each variable. We can also display the statistics of each variable separately.

```
#Displaying the summary of each variable
summary(CombinedData$Date)

##         Min.      1st Qu.       Median         Mean      3rd Qu.
## "2012-01-31" "2012-09-15" "2013-04-30" "2013-04-30" "2013-12-15"
##         Max.
## "2014-07-31"

summary(CombinedData$Employment)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 5877000 5943000 5977000 5981000 6016000 6143000

summary(CombinedData$Unemployment)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  464300  558000  577000  576600  618300  668900
```

Thus, we are able to get a deeper understanding of the data using these functions.

Results

In this tutorial, we have been studying the trends in employment and unemployment in Illinois from January 2012 to July 2014. Let us take a look at our data.

```
##          Date Employment Unemployment
## 1  2012-01-31    5876820       635818
## 2  2012-02-29    5919039       623540
## 3  2012-03-31    5946426       593888
## 4  2012-04-30    5965440       557657
## 5  2012-05-31    5992902       560998
## 6  2012-06-30    6037349       626361
## 7  2012-07-31    6023812       626706
## 8  2012-08-31    5973863       595710
## 9  2012-09-30    6020643       545951
## 10 2012-10-31    6036419       558662
## 11 2012-11-30    6011874       547518
## 12 2012-12-31    5977220       581915
## 13 2013-01-31    5882467       668919
## 14 2013-02-28    5894191       654342
## 15 2013-03-31    5905710       604625
## 16 2013-04-30    5939808       565811
## 17 2013-05-31    5971651       576956
## 18 2013-06-30    6001669       648494
## 19 2013-07-31    6002938       629996
## 20 2013-08-31    5962724       597271
## 21 2013-09-30    5976571       563823
## 22 2013-10-31    5959994       569320
## 23 2013-11-30    5989436       558259
## 24 2013-12-31    5964554       560872
## 25 2014-01-31    5914552       589820
## 26 2014-02-28    5930142       612988
## 27 2014-03-31    5989489       544674
## 28 2014-04-30    6029562       469132
## 29 2014-05-31    6048065       466981
## 30 2014-06-30    6131016       471760
## 31 2014-07-31    6142902       464335
```

Table 1: Employment and unemployment data set

As we can see, Table 1 shows the number of people employed and unemployed in the state of Illinois. A careful examination reveals that the number of employed people is much higher than the number of unemployed people in

Illinois. There is no steady increase or decrease in the employment or unemployment rate over the time frame. But it can be noticed that there certain months of the year, like July and November, when more people have a job. To complement this, during these months, the number of people without a job decreases proportionately.

So far, we have collected the data, cleaned, and prepared it for plotting. Graphical representation of data is called plotting. This makes it is easier to understand the data and gives us a clear picture of the pattern in which employment and unemployment varies over the given time frame.

Now, let us begin plotting the data. Our complete data set is stored in the data frame called *CombinedData*. The data is stored in wide format in our data frame. To plot the values on a chart, we should convert the data from wide format to long format. This conversion can be made by using the `melt` function from the `reshape2` package. So, we should install and load the `reshape2` package to melt the data.

```
#loading the reshape2 package
library(reshape2)
#Melting the data frame based on "Date"
MoltenData <- melt(data=CombinedData, id.vars="Date")
#show MoltenData
head(MoltenData)

##         Date   variable   value
## 1 2012-01-31 Employment 5876820
## 2 2012-02-29 Employment 5919039
## 3 2012-03-31 Employment 5946426
## 4 2012-04-30 Employment 5965440
## 5 2012-05-31 Employment 5992902
## 6 2012-06-30 Employment 6037349
```

The `data` argument in the above function specifies the data frame being melted. The data frame has to be melted based on a given variable. This is specified in the `id.vars` argument. Objects created by `melt` are often referred to as "molten" data in the *reshape2* documentation. That is why we gave our new data frame the name *MoltenData*.

Now, let us do some cleaning on the molten data. First, we will change the variable names to make it relevant to the data.

```
#Renaming the two variables
MoltenData<-plyr::rename(x=MoltenData,
                         replace=c("variable" = "Type",
                                   "value" = "Number"))
```

In the above step, we replaced the second variable in *MoltenData* called "variable" to "Type". Also, we changed the third variable "value" to "Number". Now, let us sort the data by "Date" and then by "Type".

```
# Sorting by Date, then by Type
MoltenData<-MoltenData[order(MoltenData$Date, MoltenData$Type),]
#show MoltenData after sorting
head(MoltenData, 10)

##          Date         Type  Number
## 1  2012-01-31   Employment 5876820
## 32 2012-01-31 Unemployment  635818
## 2  2012-02-29   Employment 5919039
## 33 2012-02-29 Unemployment  623540
## 3  2012-03-31   Employment 5946426
## 34 2012-03-31 Unemployment  593888
## 4  2012-04-30   Employment 5965440
## 35 2012-04-30 Unemployment  557657
## 5  2012-05-31   Employment 5992902
## 36 2012-05-31 Unemployment  560998
```

Now, our data frame is ready to be plotted. Let us begin plotting by installing and loading all the required add-on packages. For plotting the data, we need to install the `ggplot2` package. The `scales` package defines the scale functions for the plot. Finally, `gridExtra` package specifies the high-level grid functions. Once all these packages are installed, we should load them in R using the `library` function.

```
#loading all the packages needed for plotting
library(ggplot2)
library(scales)
library(gridExtra)
```

We use the `ggplot` function to plot the chart. We can plot different types of charts using this function. In this tutorial, we are plotting a line chart.

```
#Plotting the combined data set
ggplot(MoltenData,
       aes(as.Date(Date, "%e %b %Y"), Number, color = Type)) +
  geom_line() + geom_point()+
  scale_y_continuous(labels = comma) +
  xlab("Date") + ylab("Number of People")+
  theme_bw(base_size = 15)
```

Here, *MoltenData* is the data frame in long format. In our plot, we have the date on the x-axis and the number of people employed or unemployed on the

y-axis. So, we set the x and y axes using the "Date" and "Number" variables respectively. The `as.Date` function will format the given date to a specified format. The format is specified in its argument as `"%e %b %Y"` which formats the date to"dd mmm yyyy" format. We also told `ggplot` that the lines should have different colors based on the "Type" of data they represent. `geom_line` specifies that we want a line graph, and `geom_point` specifies that each value on the line should be highlighted with a point. `scale_y_continuous` is used to format y-axis as a continuous-valued axis and `labels = comma` will apply a comma style to the ticks on the y-axis. `comma` style is defined in the `scales` package. So we should install and load the `scales` package to use it. `xlab` and `ylab` set the axes' labels. Finally, the function `theme_bw` gives the plot a simple black and white theme. We added the argument `base_size = 15` to increase the plot's font size. The above code produces a graph like this:

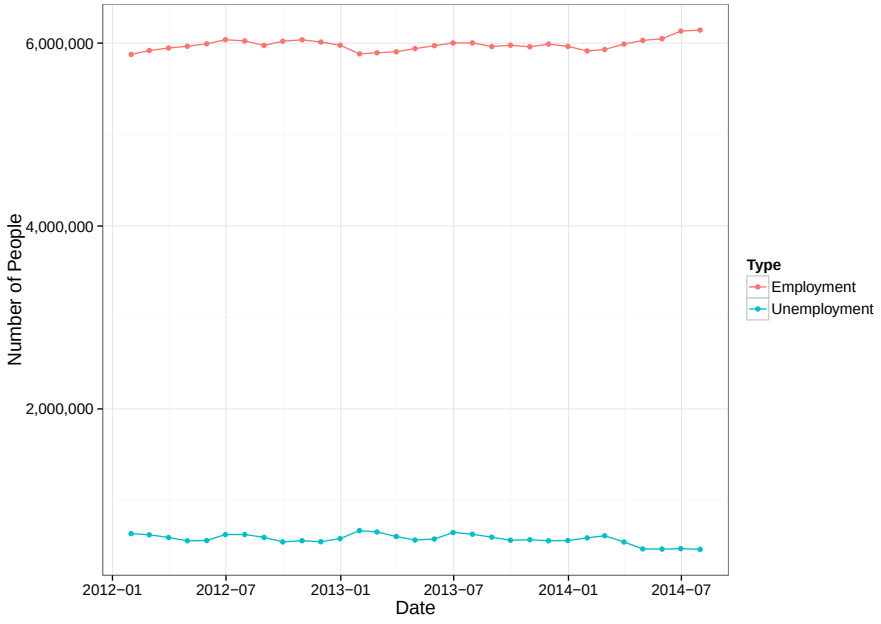

Figure 1: Employment and unemployment in Illinois plotted using the merged data set

Figure 1 shows the employment and unemployment trends for the state of Illinois from January 2012 to July 2014. It can be noticed that almost every year the employment rate increases before the middle of the first quarter and reaches a peak in July. Thus there are more jobs during the summer than the winter months early in the year. The employment rate tapers off in August. This could be attributed to the fact that students return back to school during that month after summer jobs. The employment rate then increases through the holiday season when the consumer and other retail industries hire more people

for this period; after which it shows a downward trend into the beginning of the next year.

We are not able to clearly understand the graphical pattern from Figure 1. This is because we tried to plot the combined data frame. But the large gap between the two numbers for a given date made the scale on the y-axis too big. Therefore, they look like two parallel lines.

Let us see if we can try to generate a plot which will clearly show the variation in trend. Instead of plotting the combined data frame, we will plot two separate graphs, one for employment and the other for unemployment. We will then place the graphs one on top of the other. So, we have to generate two separate plots and give each of them a name. Let us call the top chart `g.top` and the bottom chart `g.bottom`.

```
#plotting the employment data
g.top<-ggplot(EmploymentData,
              aes(as.Date(Date, "%e %b %Y"), Employment)) +
  geom_line() + geom_point() +
  scale_y_continuous(labels = comma) +
  xlab("") +
  theme_bw()+
  theme(axis.text.x = element_blank(),
        plot.margin = unit(c(0,5,-26,1),units="points"))

#plotting the unemployment data
g.bottom<-ggplot(UnemploymentData,
                 aes(as.Date(Date, "%e %b %Y"), Unemployment)) +
  geom_line() + geom_point() +
  scale_y_continuous(labels = comma) +
  xlab("Date")+
  theme_bw() +
  theme(plot.margin = unit(c(0,5,1,9),units="points"))

#Finally, arranging one plot on top of the other
grid.arrange(g.top,g.bottom)
```

Both the graphs have the same x-axis. So we just need one x-axis for both the graphs to share. We will use the x-axis of the bottom plot. `axis.text.x = element_blank()` will hide the x-axis markings on `g.top` which will make it look like they have only one x-axis. `plot.margin` adjusts the top, right, bottom and left margins for the plot measured in points. We adjusted the margins of both the plots to make them look like as shown in Figure 2.

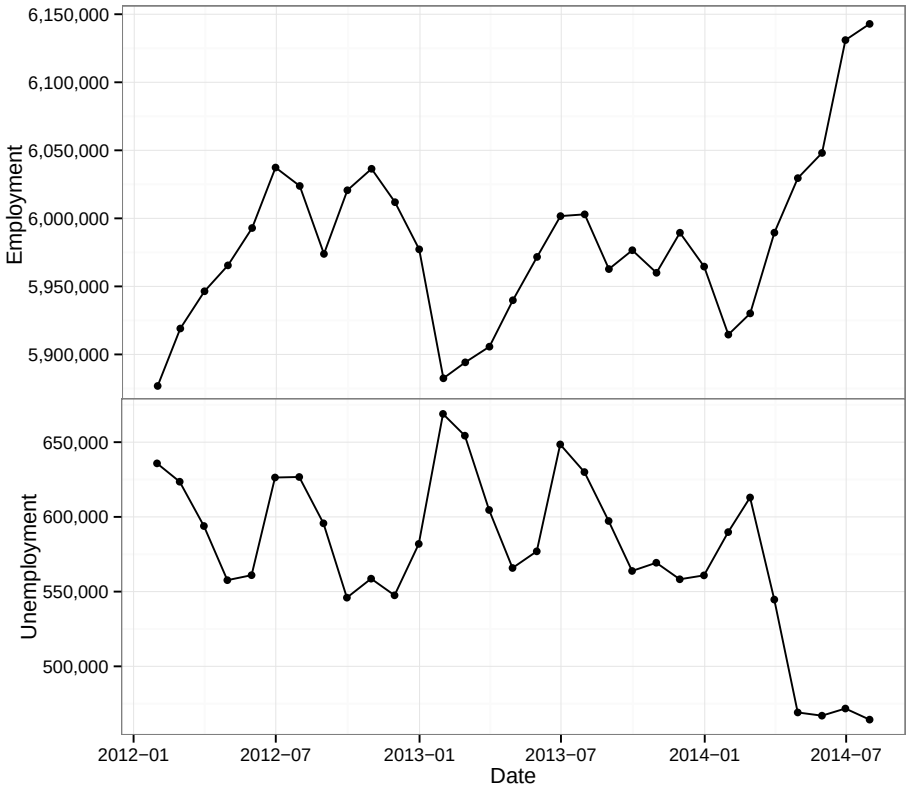

Figure 2: Employment and unemployment in Illinois plotted separately for better pattern visibility

The variation in trend is clearly visible now. It can be noticed that the bottom plot is almost a mirror image of the top plot. Therefore we can say that employment and unemployment complements each other. When more people get a job, unemployment level goes down and vice versa.

Conclusion

In this tutorial, we studied how to create graphs in R using ggplot2. For this, we used the employment and unemployment data in Illinois for the last two years. The first step was to **O**btain the data from Quandl. This was followed by **S**crubbing or cleaning the data so that it can be displayed in both table and graph form. Then we **E**xplored the data, displayed the data in tabular form (**M**ore exploring). Finally, we plotted the data(graphi**N**g). Therefore, this completes all the OSEMN steps.