R - The Power of Open Source Packages

Justin Nelson - Post 01

R is commonly referred to as the undisputed best programming language for statistical analysis. This is largely in part due to its robust and useful built in functionalities, such as vectorization and coercive properties. However, what is equally important is the myriad of readily available, user created R packages. As Matt Adams put it, “The vastness of the package ecosystem is definitely one of R’s strongest qualities – if a statistical tecnique exists, odds are there’s already an R package out there for it.” (InfoWorld)

This topic is of special interest to me due to my experience with data anlysis prior to this course, specifically in Python. In Data 8, much of our analysis was done by creating our own functions to iterate over lists, calculate specific values, etc. While we would import packages from external sources from time to time, the way in which it was done was not nearly as simple or straightforward as it was in R. Additionally, the readily available and concise documentation provided for packages and functions in R makes it all the easier to use.

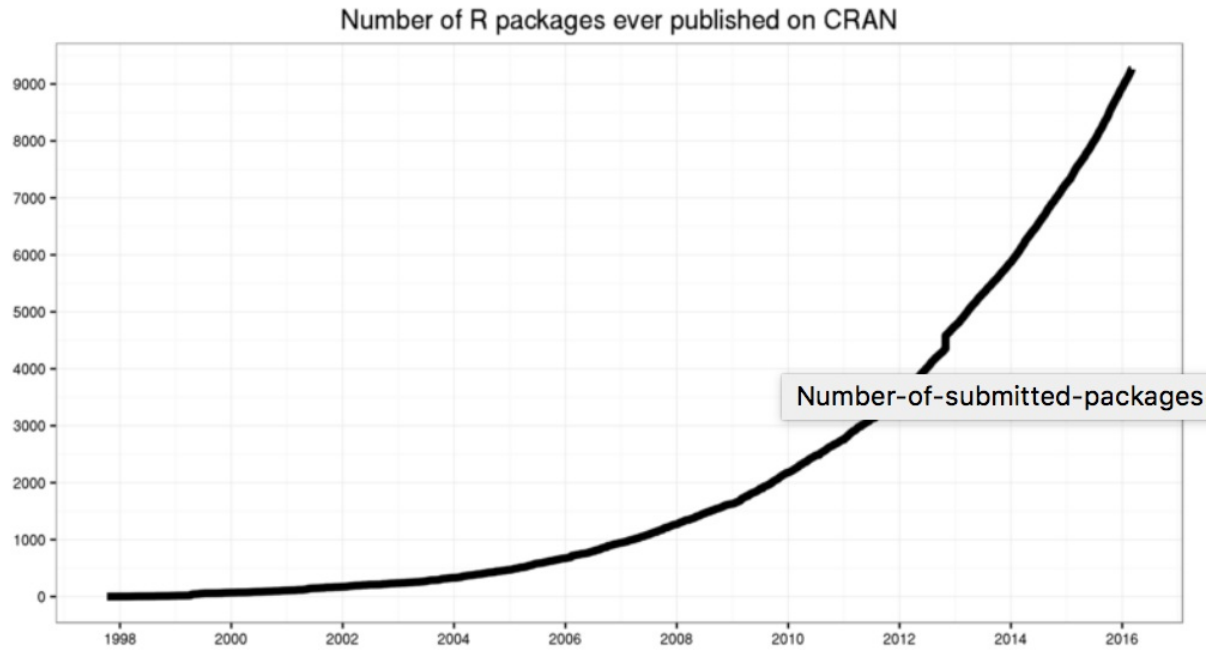
To offer a comprehensive overview of R’s strong open source offerings, this report will contain information on the following areas:

1. Overview, History, and Development of CRAN
2. Searching CRAN & Installing Packages Accordingly
3. Utilizing Packages for Common and Useful Applications
4. Conclusion

Overview, History, and Explanation of CRAN

CRAN, an acronym for Comprehensive R Archive Network, is R’s platform where users can both upload and download packages. CRAN was announced on April 23rd, 1997 by Kurt Jornik, and the original post can be found here. A package is essentially a collection of functions whose usages revolve around one general purpose. These packages are built by users and submitted to CRAN, where they are thoroughly reviewed both volunteerily by other R Users along with automated computing processes. After testing is completed, revisions made, and the functions are up to par, CRAN moderators will allow it to be added to the archive if the creator desires. However, for the package to be uploaded, the creater must agree to a set of terms and agreements that allow their work to be used freely by others. More info can be found here.

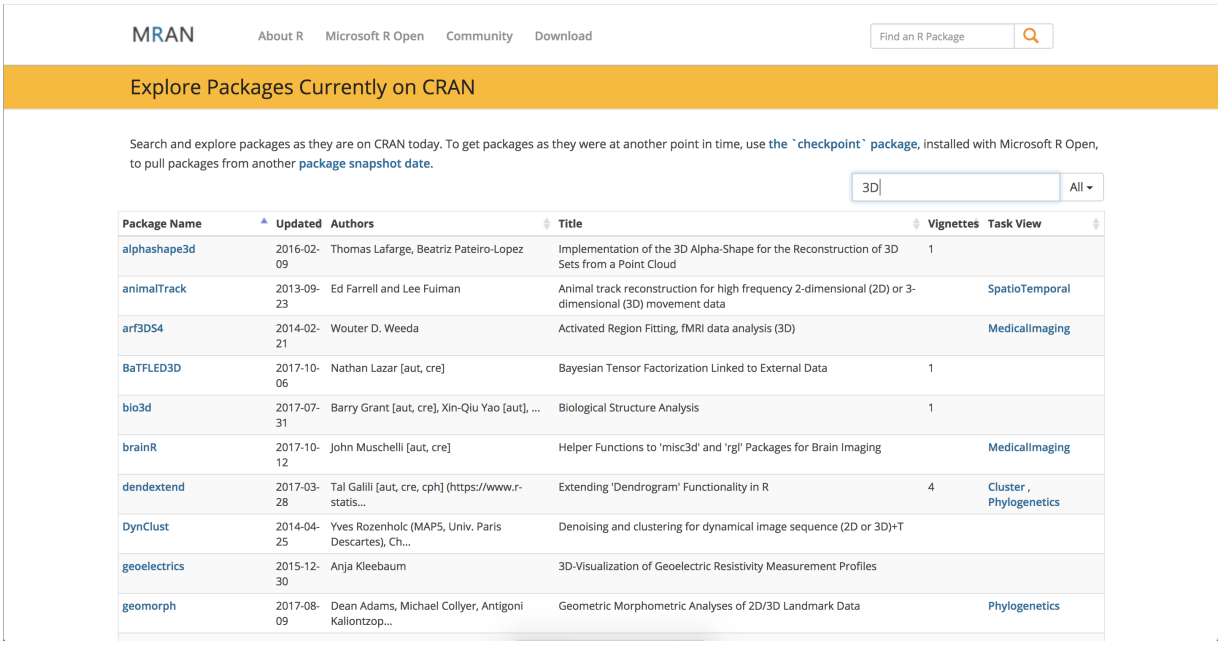
When base R is installed, it comes with 8 different packages. However, CRAN currently contains over 8,000 packages that are available for download. The progression of available packages throughout the years can be seen in this graph:



Number of Available R Packges (1998-2016)

Searching MRAN & Installing Packages Accordingly

While having 8,000 packages that all serve different purposes is largely advantageous, it also means that locating the packages you desire could prove difficult. Thankfully, there is MRAN. MRAN, an acronym for Microsoft Open R, “is the enhanced distribution of open source R from Microsoft Corporation”. Microsoft has created a very user friendly interface and platform for navigating the CRAN database. It can be accessed by visiting https://mran.microsoft.com/. With fields for “Package Name”, “Updated”, “Title”, and “Task View”, along with a search bar, locating your desired package is relatively easy. For example, say I wanted a package to help me with plotting things in three dimensions. I could then go to the search bar and type in “3D”. The results for packages related to 3D functionalities are displayed, and I can choose accordingly. This is documented in the picture below.



Suppose that after scanning through the various packages related to three dimensional functionalities, I come across the package "plot3d". I can then click on it, and explore the package's manual. Within the manual is documentation of a detailed description, author, last updated, usages, etc. After exploring this, I decide to use it. As we've already learned in lecture, to install the package you use the install.packages() function that's built into R. In this case, it would look like this:

```
#Use the built-in install.packages() function with the plot3D package in string form to download the package.
install.packages('plot3D', repos = "http://cran.us.r-project.org")
```

```
## Installing package into '/Users/JustinNelson/Library/R/3.3/library'
## (as 'lib' is unspecified)
```

```
##
## The downloaded binary packages are in
##  /var/folders/js/hj1wjrjs1wvgpcxfj411hp1r0000gn/T//RtmpuadB4g/downloaded_packages
```

With the package now installed on your machine, you can load it and start using it with the library() function.

```
#Use library() on the desired package to load it into the current session
library(plot3D)
```

While the installation and loading of packages is relatively simple and something we've done often in class, the important take-away from this section is how to find the specific package you're looking for. While it often may be the case that you already the name of the package you want or it is supplied to you beforehand (i.e. as it has been in class), it's important to know how to find the appropriate packages when you undertake your personal work.

Utilizing Packages for Common and Useful Applications

Within the extensive documentation in the manual that was mentioned earlier, there is also information about how to utilize a packages functions. Things such as arguments, best practices, and examples are provided.

To demonstrate the usefulness of packages, suppose I have two goals in mind: 1. Graph the relationship between year, GDP in Billions of Current Dollars, and GDP in Billions of Chained 2009 Dollars to get a picture of the U.S. macro economy over the past ~80 years. 2. Analyze how a company that is currently being traded on the stock market has been performing over the past year.

Goal: Graphing the relationship between the three variables.

First, I'll import the dataset that I'll be working with. I found this data on the Bureau of Economic Analysis' website.

```
#Read in the CSV file that contains the data to be analyzed
gdp <- read.csv('Data/GDP Data.csv')
```

To graph the relationship between these variables simultaneously, we'll have to use a 3D graph. Since we've already installed and loaded the plot3d package above, there is no need to load it again. From here, I can read through the documentation of plot3D until I find the type of graph I want to use.

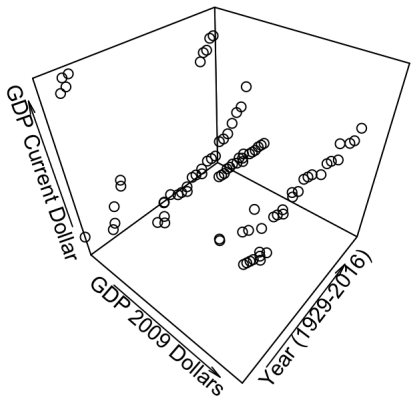
After scanning through examples, it's apparent that a 3-D scatter plot is best, so I'll use the scatter3D function to create it.

```
#Checking the columns of data to be used
gdp
```

```
##    Year GDP.Current.Dollar GDP.Chained.2009.Dollars
## 1  1929              104.6                 1,056.6
## 2  1930               92.2                   966.7
## 3  1931               77.4                   904.8
## 4  1932               59.5                   788.2
## 5  1933               57.2                   778.3
```

```
## 6  1934              66.8                  862.2
## 7  1935              74.3                  939.0
## 8  1936              84.9                1,060.5
## 9  1937              93.0                1,114.6
## 10 1938              87.4                1,077.7
## 11 1939              93.5                1,163.6
## 12 1940             102.9                1,266.1
## 13 1941             129.4                1,490.3
## 14 1942             166.0                1,771.8
## 15 1943             203.1                2,073.7
## 16 1944             224.6                2,239.4
## 17 1945             228.2                2,217.8
## 18 1946             227.8                1,960.9
## 19 1947             249.9                1,939.4
## 20 1948             274.8                2,020.0
## 21 1949             272.8                2,008.9
## 22 1950             300.2                2,184.0
## 23 1951             347.3                2,360.0
## 24 1952             367.7                2,456.1
## 25 1953             389.7                2,571.4
## 26 1954             391.1                2,556.9
## 27 1955             426.2                2,739.0
## 28 1956             450.1                2,797.4
## 29 1957             474.9                2,856.3
## 30 1958             482.0                2,835.3
## 31 1959             522.5                3,031.0
## 32 1960             543.3                3,108.7
## 33 1961             563.3                3,188.1
## 34 1962             605.1                3,383.1
## 35 1963             638.6                3,530.4
## 36 1964             685.8                3,734.0
## 37 1965             743.7                3,976.7
## 38 1966             815.0                4,238.9
## 39 1967             861.7                4,355.2
## 40 1968             942.5                4,569.0
## 41 1969           1,019.9                4,712.5
## 42 1970           1,075.9                4,722.0
## 43 1971           1,167.8                4,877.6
## 44 1972           1,282.4                5,134.3
## 45 1973           1,428.5                5,424.1
## 46 1974           1,548.8                5,396.0
## 47 1975           1,688.9                5,385.4
## 48 1976           1,877.6                5,675.4
## 49 1977           2,086.0                5,937.0
## 50 1978           2,356.6                6,267.2
## 51 1979           2,632.1                6,466.2
## 52 1980           2,862.5                6,450.4
## 53 1981           3,211.0                6,617.7
## 54 1982           3,345.0                6,491.3
## 55 1983           3,638.1                6,792.0
## 56 1984           4,040.7                7,285.0
## 57 1985           4,346.7                7,593.8
## 58 1986           4,590.2                7,860.5
## 59 1987           4,870.2                8,132.6
## 60 1988           5,252.6                8,474.5
## 61 1989           5,657.7                8,786.4
## 62 1990           5,979.6                8,955.0
## 63 1991           6,174.0                8,948.4
## 64 1992           6,539.3                9,266.6
## 65 1993           6,878.7                9,521.0
## 66 1994           7,308.8                9,905.4
## 67 1995           7,664.1               10,174.8
## 68 1996           8,100.2               10,561.0
## 69 1997           8,608.5               11,034.9
## 70 1998           9,089.2               11,525.9
## 71 1999           9,660.6               12,065.9
## 72 2000          10,284.8               12,559.7
## 73 2001          10,621.8               12,682.2
## 74 2002          10,977.5               12,908.8
## 75 2003          11,510.7               13,271.1
## 76 2004          12,274.9               13,773.5
## 77 2005          13,093.7               14,234.2
## 78 2006          13,855.9               14,613.8
## 79 2007          14,477.6               14,873.7
## 80 2008          14,718.6               14,830.4
## 81 2009          14,418.7               14,418.7
## 82 2010          14,964.4               14,783.8
## 83 2011          15,517.9               15,020.6
## 84 2012          16,155.3               15,354.6
## 85 2013          16,691.5               15,612.2
## 86 2014          17,427.6               16,013.3
## 87 2015          18,120.7               16,471.5
## 88 2016          18,624.5               16,716.2
```

```
#Generating a 3d scatter plot of the three variables.
scatter3D(as.numeric(gdp$GDP.Chained.2009.Dollars), as.numeric(gdp$Year), as.numeric(gdp$GDP.Current.Dollar), xlab = 'GDP 2009 Dollars', ylab = 'Year (1929-2016)', zlab = 'GDP Current Dollar', colkey = NULL, col = NULL, colvar = NULL)
```



From this graph, we can see that, in general, GDP in current dollars and GDP in 2009 chained dollars rose in unison. This did not hold true during the 1970's when the US was experiencing inflation rates that were far above standard (>10%). It's also interesting to note the stark dropoff in both forms of GDP during the early 1930's (early part of Y-axis) when the Great Depression occurred.

By utilizing the 3d functionalities available through plot3D, we're able to visualize economic trends that occurred in the United States over an 80 year period.

Goal: Analyzing the performance of a stock over several different time periods

Suppose I'm interested in exploring NVidia's stock a bit. To gather this data I could go onto a website that collects information on stock performance, find Nvidia, and see if there are available files in a convienant format (aka CSV) to download. But instead of that, why not see if an R package already exists to do that in a simplified fashion?

A quick search of MRAN led me to the package "quantmod", a package that "…is designed to assist the quantitative trader in the development, testing, and deployment of statistically based trading models."

Let's start by downloading the package from CRAN.

```
#Using install.packages() to acquire the package.
install.packages('quantmod', repos = "http://cran.us.r-project.org")
```

```
## Installing package into '/Users/JustinNelson/Library/R/3.3/library'
## (as 'lib' is unspecified)
```

```
##
## The downloaded binary packages are in
##  /var/folders/js/hj1wjrjs1wvgpcxfj411hp1r0000gn/T//RtmpuadB4g/downloaded_packages
```

```
#Load the package so we can use it
library('quantmod')
```

```
## Loading required package: xts
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: TTR
```

```
## Version 0.4-0 included new data defaults. See ?getSymbols.
```

Now that the package is loaded, let's gather some data about NVIDIA. A quick read through of the documentaiton tells us that we can use the

getSymbols() function to load data about a company, and a Google search yields that Nvidia's ticker label is NVDA.

```
getSymbols("NVDA",src="google")
```

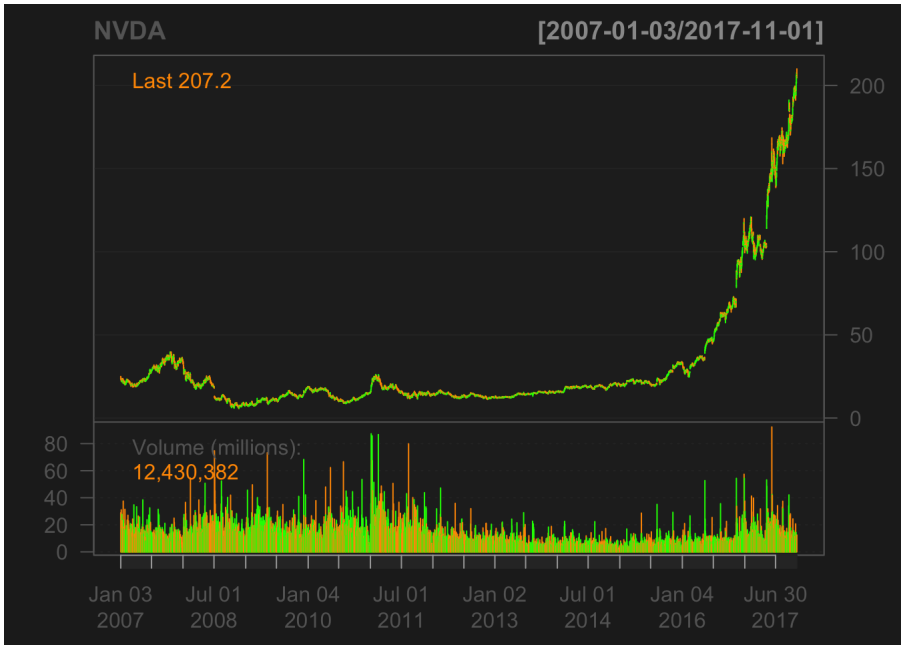
```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
## [1] "NVDA"
```

```
#Note that the getSymbols function automatically creates a data frame for the loaded data that is named by the tic
ker label. This means that we don't have to assign the loaded data to a variable - it's already done!
```

Let's take a quick look at a chart of this stock's performance since it's inception to gain an idea of what we're dealing with.

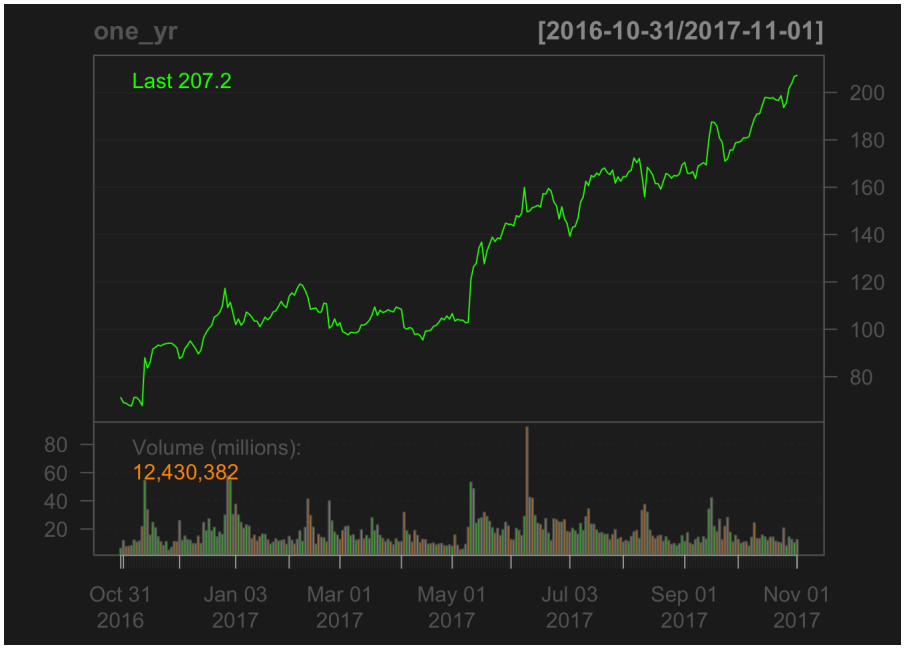
```
#Generate a barchart of the stock's history.
barChart(NVDA)
```



That's handy, but it's nothing that we couldn't find online without relative ease.

Let's zoom in on the past year specificially.

```
#Gathering data for specifically the past year.
one_yr <- NVDA['2016-10-30::']
lineChart(one_yr)
```



That's some impressive growth for a stock! How much exactly is it though?

```
#Finding the closing price on October 31, 2016
close1 <- as.numeric(Cl(one_yr[1]))
#Finding the closing price on October 27th, 2017.
close2 <- as.numeric(Cl(one_yr[251]))

#Calculating total percent change over the year.
(close2 / close1) * 100
```

```
## [1] 283.6706
```

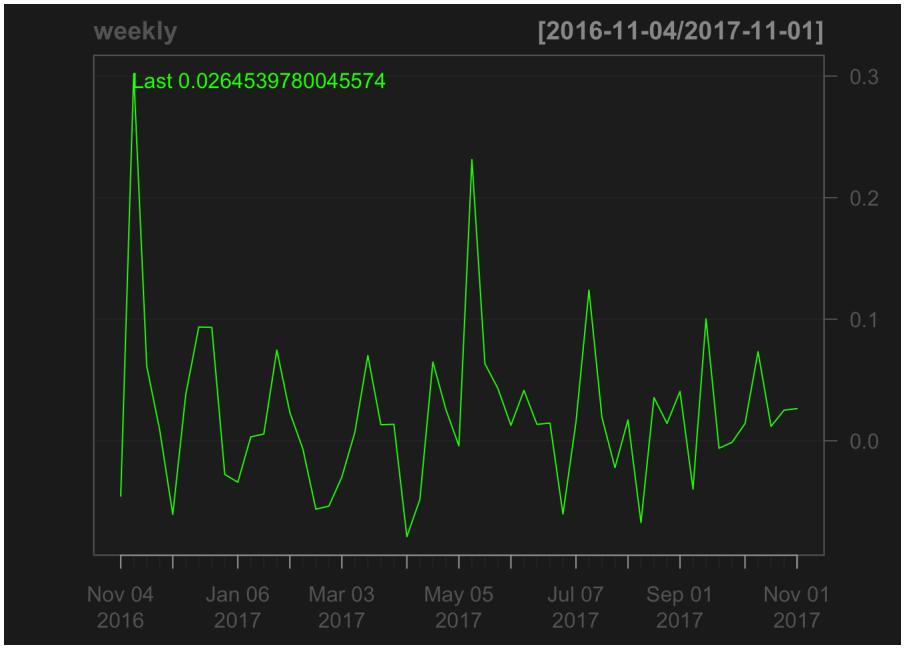
Once again, useful to know but something that could be found relatively easily online. Let's look at something a litle more complex that would only be easily done with this package or some other complex software.

Let's find the number of weeks over the past year that NVDA's stock went positive, and express that as a percentage as well.

```
#The built in weeklyReturn function yeilds the return values for each week.
weekly <- (weeklyReturn(one_yr))
```

Before finding the number of weeks that the stock went positive, let's take a look at it graphically:

```
#Generating a line chart of the stocks weekly performance.
chartSeries(weekly)
```



Overall, this looks strong with mostly positive weeks and some negatives. Let's find the exact value of positive weeks:

```r
#Converting the data into numeric types so we can work with it.
weekly <- as.numeric(weekly)
#Creating a factor of booleans for when returns were postive.
positive <- weekly > 0
#Counting number of weeks were positive.
pos_weeks <- sum(positive)
#35 total positive weeks. And how about as a percentage for the entire year?
pos_weeks / length(weekly)
```

```
## [1] 0.6792453
```

So, we found that in the past 51 weeks, there were 35 in which NVDA's stock rose (or 67.3% of the time). This is the type of data that would not be easily extractable via most finance websites.

Conclusion

When undertaking a data analysis project in R, it is highly recommended to consult CRAN. While constructing your own functions or scouring the web for data is definitely useful in specific situations, it is highly likeley that a package already exists for the analysis you're trying to do. Packages exist for some of the most basic analysis like linear regression, to much more complex things such as Neural Network Algorithms. In addition to statistical analysis, packages are also available for manipulating data in R, which is useful for transforming or reworking data into a desired form so it can be explored further. Spend your time wisely by driving towards the conclusion you seek, not by developing work that is readily available AND free!