readxl, dplyr and the pipe

Matt McDonald

4/21/2020

readxl

Tidyverse Packages

R includes some useful tools to help with data analysis and manipulation. One of the primary sources of data we use can be found in Excel spreadsheets. The "readxl" package can help us access this data directly.

In order to access the readxl package, we need to load it. Fortunately, readxl is a component of a suite of R packages called the "tidyverse". The packages included in the "tidyverse" suite are cutting edge, best-in-class tools for data manipulation, visualization and programming, and are supported by RStudio.

We can load the "tidyverse" suite of models with the following command:

```
library(tidyverse)
```

```
## -- Attaching packages -------------------------------------------------------------------------------

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ----------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Loading the Data

Once tidyverse is loaded, we can load data into R directly from Excel. We use the "<-" operator to indicate variable assignment. The following code uses the "read_excel" function from the readxl package to load the data into a variable called "return_data". We can see the contents of our new variable by typing its name into the console (the equivalent of the 2nd line of code in the block below).

```
return_data <- readxl::read_excel('31var.xlsx', sheet='34var')
return_data
```

```
## # A tibble: 60,628 x 34
##    tickers date                   ret     cfp   cashpr datedif     agr     cash
##      <dbl> <dttm>               <dbl>   <dbl>    <dbl>   <dbl>   <dbl>    <dbl>
##  1    2664 2004-12-31 00:00:00  0.209   1.59     -0.319     141 -0.195   0.133
##  2       1 2005-06-30 00:00:00  0.0354 -0.118  -282.        170  0.0304  0.00335
##  3       1 2005-12-30 00:00:00  0.231  -0.359    -9.98      176  0.0889  0.0949
##  4       1 2006-06-30 00:00:00  0.914   0.186    NA         182  0.0594 NA
##  5       1 2006-12-29 00:00:00  0.902   0.263    NA         188  0.0731 NA
##  6       1 2007-06-29 00:00:00  0.403   0.229    NA         194  0.205  NA
##  7       1 2007-12-28 00:00:00 -0.499   0.193    NA         200  0.123  NA
##  8       1 2008-06-30 00:00:00 -0.511   0.201    NA         206  0.253  NA
##  9       1 2008-12-31 00:00:00  1.31    0.829    NA         212  0.0738 NA
## 10       1 2009-06-30 00:00:00  0.117   0.0567   NA         218  0.141  NA
## # ... with 60,618 more rows, and 26 more variables: stdturn <dbl>, sp <dbl>,
## #   sgr <dbl>, quick <dbl>, roaq <dbl>, roeq <dbl>, roic <dbl>, saleinv <dbl>,
## #   salecash <dbl>, salerec <dbl>, orgcap <dbl>, ag <dbl>, `CGS1&2` <dbl>,
## #   `CLT-1` <dbl>, `CURE R` <dbl>, LSZ <dbl>, PS <dbl>, `R&D` <dbl>, TWX <dbl>,
## #   xing <dbl>, mve <dbl>, marketcap <dbl>, EPS <dbl>, LEV <dbl>, LGR <dbl>,
## #   PB <dbl>
```

Getting Help

The read_excel function is very flexible, and there are many ways to refer to the data. You can see that we simply pointed it to the sheet containing our important data with the "sheet" paramater. You can get very specific about where the data is that you want to input, such as by the range or how many rows to read by using the paramers for the function. Any time you're looking for help on a function in R, simply type the command "?" followed by the function you're interested in, and R will show you the appropriate help file.

```
#> ? read_excel
```

Data Frames

When we load our data into R, the typical format used is called a DataFrame (sometimes people call them tibbles). Data frames can be thought of as a sheet of data in an Excel file, with rows and columns. One useful feature of DataFrames is that each column must be of the same data type. This helps give us confidence that we can perform the same operations across all of our data, solving a pitfall that sometimes we encounter in Excel. Check out the "Environment" tab in RStudio to inspect the data types that we loaded in the "return_data" DataFrame.

For more information about data types in R, check out:

https://swcarpentry.github.io/r-novice-inflammation/13-supp-data-structures/

Data Manipulation with dplyr

dplyr is another package that gets loaded when we load the tidyverse, and it is very useful for data manipulation. In this section, I will outline some useful functions in dplyr, and point out their equivalents in Excel

select

Select allows you to choose columns from your data frame that you want to work with. This is similar to hiding columns in your Excel spreadsheet.

In this example, I will only focus on three columns of my data: date, ret and marketcap:

```
select(return_data, date, ret, marketcap)
```

```
## # A tibble: 60,628 x 3
##    date                    ret    marketcap
##    <dttm>                <dbl>        <dbl>
##  1 2004-12-31 00:00:00  0.209    359294006.
##  2 2005-06-30 00:00:00  0.0354 11538725344.
##  3 2005-12-30 00:00:00  0.231  11947347995.
##  4 2006-06-30 00:00:00  0.914  14710415446.
##  5 2006-12-29 00:00:00  0.902  28156046496.
##  6 2007-06-29 00:00:00  0.403  57427589654.
##  7 2007-12-28 00:00:00 -0.499  88525515797
##  8 2008-06-30 00:00:00 -0.511  46175411255.
##  9 2008-12-31 00:00:00  1.31   29377403389.
## 10 2009-06-30 00:00:00  0.117  67760564687.
## # ... with 60,618 more rows
```

mutate

mutate allows us to create new variables or override existing variables. This is the equivalent of adding a new calculated column to our data and "copy/paste down" the formula to all the applicable cells.

In this example, I'm taking the ret column, which is annualized returns, and converting them to quarterly returns. You can see that there are some benefits to the syntax of R...I don't have to "copy/paste down" my formula...it's applied to all the rows of my data automatically!

```
mutate(select(return_data, date, ret, marketcap), qtr_ret=(1 + ret) ^ (1/4) - 1)
```

```
## # A tibble: 60,628 x 4
##    date                    ret    marketcap  qtr_ret
##    <dttm>                <dbl>        <dbl>    <dbl>
##  1 2004-12-31 00:00:00  0.209    359294006.  0.0485
##  2 2005-06-30 00:00:00  0.0354 11538725344.  0.00874
##  3 2005-12-30 00:00:00  0.231  11947347995.  0.0534
##  4 2006-06-30 00:00:00  0.914  14710415446.  0.176
##  5 2006-12-29 00:00:00  0.902  28156046496.  0.174
##  6 2007-06-29 00:00:00  0.403  57427589654.  0.0883
##  7 2007-12-28 00:00:00 -0.499  88525515797  -0.159
##  8 2008-06-30 00:00:00 -0.511  46175411255. -0.164
##  9 2008-12-31 00:00:00  1.31   29377403389.  0.232
## 10 2009-06-30 00:00:00  0.117  67760564687.  0.0280
## # ... with 60,618 more rows
```

A quick diversion: pipes!

The code we wrote above is already starting to exhibit some readability issues. R has a very handy syntax tool that helps make our code more readable and prevents us from having to have "nested" function calls. "nested" function calls are what makes some very long excel cell formulas difficult to read.

Here's how it works: the characters "%>%" are called the "pipe", and they allow us to take a value and "pipe" it into the first parameter of a function. So, the command:

```
2 %>% sqrt()
```

```
## [1] 1.414214
```

is the equivalent of the command:

```
sqrt(2)
```

```
## [1] 1.414214
```

This doesn't seem like that great of an innovation in this example, but using it to rewrite the code we wrote above yields:

```
return_data %>%
  select(date, ret, marketcap) %>%
  mutate(qtr_ret = (1 + ret) ^ (1/4) - 1)
```

```
## # A tibble: 60,628 x 4
##    date                    ret    marketcap  qtr_ret
##    <dttm>                <dbl>        <dbl>    <dbl>
##  1 2004-12-31 00:00:00  0.209    359294006.  0.0485
##  2 2005-06-30 00:00:00  0.0354 11538725344.  0.00874
##  3 2005-12-30 00:00:00  0.231  11947347995.  0.0534
##  4 2006-06-30 00:00:00  0.914  14710415446.  0.176
##  5 2006-12-29 00:00:00  0.902  28156046496.  0.174
##  6 2007-06-29 00:00:00  0.403  57427589654.  0.0883
##  7 2007-12-28 00:00:00 -0.499  88525515797  -0.159
##  8 2008-06-30 00:00:00 -0.511  46175411255. -0.164
##  9 2008-12-31 00:00:00  1.31   29377403389.  0.232
## 10 2009-06-30 00:00:00  0.117  67760564687.  0.0280
## # ... with 60,618 more rows
```

Which is really nice! We don't have to start at the middle of our function and read out, we can start at the top of our code and read down to understand all the steps we've taken, in order. We'll be using pipes for the rest of our examples

Back to data manipulation with dplyr

filter

filter allows us to select rows of our data based on some rules we apply to the data. This is the equivalent of using the filtering functionality on the data ribbon in Excel.

For example, if we only want to see data that had a quarterly return greater than 30%, we could use the following code:

```
return_data %>%
  select(date, ret, marketcap) %>%
  mutate(qtr_ret = (1 + ret) ^ (1/4) - 1) %>%
  filter(qtr_ret > 0.4)
```

```
## # A tibble: 112 x 4
##    date                  ret   marketcap qtr_ret
##    <dttm>              <dbl>       <dbl>   <dbl>
##  1 2014-12-31 00:00:00  4.04 7211796497.   0.498
##  2 2006-12-29 00:00:00  3.43  817244736    0.450
##  3 2015-06-30 00:00:00  3.06 6141837856    0.419
##  4 2008-12-31 00:00:00  3.16  976247091.   0.428
##  5 2006-12-29 00:00:00  2.84  962500000    0.400
##  6 2012-06-29 00:00:00  7.34  148669662.   0.700
##  7 2006-12-29 00:00:00  4.26 1190619677.   0.515
##  8 2009-06-30 00:00:00  3.96 2122614990    0.492
##  9 2015-06-30 00:00:00  3.81 2560571141.   0.481
## 10 2009-06-30 00:00:00  7.52  507897587.   0.708
## # ... with 102 more rows
```

summarize

summarize allows us to calculate summary statistics, like sum, average or count. It's the equivalent of using a function like "sum" or "average" in Excel, or the summary stats Excel shows you in the lower right hand corner of the screen when you highlight some data.

The following example calculates the average marketcap for all companies with a quarterly return > 40%.

```
return_data %>%
  select(date, ret, marketcap) %>%
  mutate(qtr_ret = (1 + ret) ^ (1/4) - 1) %>%
  filter(qtr_ret > 0.4) %>%
  summarize(mean_mkt_cap = mean(marketcap))
```

```
## # A tibble: 1 x 1
##   mean_mkt_cap
##          <dbl>
## 1  2354650641.
```

group_by

group_by is usually used with summarize, and allows us to get summary statistics by group. In Excel, you would normally use some sort of pivot functionality to do this.

In this example, we're getting the average marketcap by reporting date for companies that had a quarterly return greater than 40%:

```
return_data %>%
  select(date, ret, marketcap) %>%
  mutate(qtr_ret = (1 + ret) ^ (1/4) - 1) %>%
  filter(qtr_ret > 0.4) %>%
  group_by(date) %>%
  summarize(mean_mkt_cap = mean(marketcap))
```

```
## # A tibble: 16 x 2
##    date                mean_mkt_cap
##    <dttm>                     <dbl>
##  1 2005-12-30 00:00:00   374614313.
##  2 2006-12-29 00:00:00  1458874291.
##  3 2007-06-29 00:00:00  2594115473.
##  4 2007-12-28 00:00:00   200195165.
##  5 2008-12-31 00:00:00   634587436.
##  6 2009-06-30 00:00:00  1349174006.
##  7 2010-06-30 00:00:00  1496160000
##  8 2011-06-30 00:00:00  1869625659.
##  9 2012-06-29 00:00:00   148669662.
## 10 2013-12-31 00:00:00  4280721199.
## 11 2014-12-31 00:00:00  4444829666.
## 12 2015-06-30 00:00:00  2799957204.
## 13 2015-12-31 00:00:00  1231733333.
## 14 2016-06-30 00:00:00  5137953441.
## 15 2017-06-30 00:00:00  3491233674.
## 16 2017-12-29 00:00:00  1210967800
```

arrange

arrange allows us to order our data. It's the equivalent of using the sort functionality on the data ribbon in Excel.

In this example, we will sort by the average marketcap for our sample data, descending.

```
return_data %>%
  select(date, ret, marketcap) %>%
  mutate(qtr_ret = (1 + ret) ^ (1/4) - 1) %>%
  filter(qtr_ret > 0.4) %>%
  group_by(date) %>%
  summarize(mean_mkt_cap = mean(marketcap)) %>%
  arrange(desc(mean_mkt_cap))
```

```
## # A tibble: 16 x 2
##    date                mean_mkt_cap
##    <dttm>                     <dbl>
##  1 2016-06-30 00:00:00  5137953441.
##  2 2014-12-31 00:00:00  4444829666.
##  3 2013-12-31 00:00:00  4280721199.
##  4 2017-06-30 00:00:00  3491233674.
##  5 2015-06-30 00:00:00  2799957204.
##  6 2007-06-29 00:00:00  2594115473.
##  7 2011-06-30 00:00:00  1869625659.
##  8 2010-06-30 00:00:00  1496160000
##  9 2006-12-29 00:00:00  1458874291.
## 10 2009-06-30 00:00:00  1349174006.
## 11 2015-12-31 00:00:00  1231733333.
## 12 2017-12-29 00:00:00  1210967800
## 13 2008-12-31 00:00:00   634587436.
## 14 2005-12-30 00:00:00   374614313.
## 15 2007-12-28 00:00:00   200195165.
## 16 2012-06-29 00:00:00   148669662.
```

Notes

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

sources

I took a lot of inspiration for this lesson from the following site:

https://rfortherestofus.com/2019/06/a-guide-to-r-for-excel-users/