Module 2 Homework

ADD IN YOUR NAME

February 22, 2018

Module 2 Homework Assignments

This module consists of 4 homework assignments and corresponds to assignments 6, 7, 8, and 9 of the semester.

Remember to set eval to T and echo to T after you have answered a code chunk so that your code and plots will appear when you knit the document.

Module 2 Homework Assignment 1

Question 1

Please review the following functions shown in class today. You should use the help documentation for each function in answering these questions. To access the help use `?function` or help(function). Please describe the following for each function:

- What are the inputs to the function?
- What are the outputs from the function?
- When is this function useful?

Example Answer:

- sum()
  - The sum() function takes one or more vectors of numbers as an argument. Additionally, it accepts the na.rm argument which allows sum() to be used even when NA values are present.
  - The output of the sum() function is a single number, the sum of all values contained in the inputs
  - The function is useful when looking to find the sum of multiple numbers. It is also useful with TRUE/FALSE values.

Please discuss the following functions:

- for
- if
- else
- identical()
- str_detect()

Question 2

Today we are going to be thinking about investing and retirement.

Let's think about things we could do with money we have left after we've paid our bills. Three main options we will be looking at today:

- Savings account
- Stock market
- Bonds

We are going to look at how the total value of our investment would change over a ten year period, starting with our savings account. Savings rates change, but over the last ten years they were very low. Let's say you have a 2% interest rate on your money. This means that every year you get 2% of the money you have in the bank added to your account.

Let's look at a simple example, where we start with $100 in our savings account and grow it.

```
# A = P(1+r*t)
# where A = account balance, P = principal, r = interest rate,
# and t = periods of time (in this case, one year)
P = 100

# After one year:
A = P*(1 + 0.02)

print(A)
```

```
## [1] 102
```

a) How much money did we make after one year?

b) Use a for loop to answer this: what would our account balance be after ten years, with an initial investment of $100 and interest rate of 2% annually?

Hint: you will update the variable account_balance ten times.

c) How much money would we have in our account after ten years, if we add additional $100 to our account each month?

d) How much of this account balance would be made from interest?

Hint: figure out the amount of total deposits made and subtract that from the account balance.

Question 3

Now we are going to read in and clean some stock data. Later, we will build portfolios using this stock data to model your retirement savings!

a) Read in the following files: AAPL.csv, FORD.csv, GE.csv, JPM.csv, EXC.csv and JCP.csv from the Hw_Data folder. List the column names of each dataset.

b) Two of these data sets have columns that the others don't. Select out the columns that these datasets have, that are unlike the others.

c) Convert the date colums into a date object for each data frame.

d) Pick one stock other than AAPL and plot its close price over time. Be sure to include appropriate titles and labels.

Your chart should look like this.

e) What do you notice about the stock's trend over time? (Hint: talk about dips, highs, general trends.)

Module 2 homework assignment 2

Question 1

Please review the following functions shown in class today. You should use the help documentation for each function in answering these questions. To access the help use `?function` or help(function). Please describe the following for each function:

- What are the inputs to the function?
- What are the outputs from the function?
- When is this function useful?

Example Answer:

- sum()
  - The sum() function takes one or more vectors of numbers as an argument. Additionally, it accepts the na.rm argument which allows sum() to be used even when NA values are present.
  - The output of the sum() function is a single number, the sum of all values contained in the inputs
  - The function is useful when looking to find the sum of multiple numbers. It is also useful with TRUE/FALSE.

Please discuss the following functions:

- %in%
- left_join()
- str_replace()
- str_split_fixed()
- str_trim()

Question 2

Rerun your code from last week so that all of our data is clean before we start today.

a) Using your FORD data set and the gather function, transform your close and volume columns so that have a key column called: "measure" and a value column called "amount".

b) Uh oh, our date values were transformed. What we wanted were three columns: date, measure, and value, but we only got two because we forgot to protect our date column. Using your FORD data, create transformed_FORD again but this time make sure that our date column is kept intact.

Your answer should look like this, again, I am using the AAPL data set *Hint: you will need to use the - sign*

c) Part of why gather is so useful is that it makes plotting with ggplot really easy! Now that I have a single column with multiple measure variables, I can use it to control ggplot aesthetics, such as color. Using your new transformed_FORD data, make a plot similar to the one I am making with the AAPL data.

d) What units should we use for the y-axis of our plot showing the closing price and volume? Does it make sense to have these two things plotted on the same chart?

e) Just as before we used gather to transform our data from wide to long, we can also use spread() to transform our data from long to wide. Using spread() turn your long version of FORD, transformed_FORD back into the wide version of the data, your data set should be identical to the FORD data you started with.

Question 3

Now we are almost ready to join our data together. But if we do that with the data as it is, look what happens:

```
AAPL_FORD <- left_join(AAPL, FORD, by = c("date"))
names(AAPL_FORD)
```

R saw that all the column names were the same, so it assigned its own! But we won't know which price belongs to which stock by just seeing x and y. So it's better to rename the columns ourselves *before* joining the data.

a) Rename the `close` and `volume` columns to `AAPL_close`, `AAPL_volume`, etc. We leave `date` the same so that we can join on it later.

b) Join all four data sets into one, and call it `stock_data`. Hint: you will have to use join multiple times.

Question 4

a) Select all of the close price columns from `stock_data` and use `gather()` to create a suitable data frame for plotting, with the columns date, Stock, and Close. Call the data frame `close_price_data`.

b) We could plot our data as is, but the values in the `Stock` column are not very clean. Rename the values of the `Stock` column in `close_price_data` from, for example, AAPL_close to AAPL. Hint: consult your notes on `str_split_fixed()`.

c) Let's plot our data, now that it's in gathered form. Reproduce the plot below, noting the labels and titles.

d) What are your major takeaways from this chart? What stocks would you want in your retirement portfolio, if you were retiring in 2018? Did our stocks maintain their relative price order (were the stocks with the highest prices in 2008 still our highest priced stocks in 2018?)

Question 5: Adding bond data

```
bond_data <- read_csv("Hw_Data/bond_data.csv") %>%
  mutate(date = as.Date(date, format = "%Y-%m-%d"))
```

a) What are the names of the two Bond funds we are looking at?

b) Join `close_price_data` with this bond data. (Hint: we are hoping to capture all of the rows in both datasets, so `left_join()` will not give the desired result.)

c) Create a dummy variable called `Bond` that equals TRUE if the Stock name refers to a bond, and FALSE otherwise. What will this variable be useful for?

d) Plot `stock_bond_data`, reproducing the chart below.

Module 2 Homework Assignment 3

Question 1

Please review the following functions shown in class today. You should use the help documentation for each function in answering these questions. To access the help use `?function` or help(function). Please describe the following for each function:

- What are the inputs to the function?
- What are the outputs from the function?
- When is this function useful?

Example Answer:

- sum()
  - The sum() function takes one or more vectors of numbers as an argument. Additionally, it accepts the na.rm argument which allows sum() to be used even when NA values are present.
  - The output of the sum() function is a single number, the sum of all values contained in the inputs
  - The function is useful when looking to find the sum of multiple numbers. It is also useful with TRUE/FALSE values.

Please discuss the following functions:

- str_sub()
- require()
- warning()
- stop()
- range()

Question 2

Now that we have all of data nicely cleaned and put together, it is time for us to use it so that we can analyze it to create our very own retirement strategies.

a)

We will need to look at monthly returns for each of our stocks and bond funds. To do this we will need our stock prices for only the last day of each month for which we have data. Using lubridate: create a year, a month, and a day column in our dataframe for each row which breaks out the components of our date columns.

b) What is the last day of every month in which we have data for each stock or bond? Using our new year, month, and day columns and dplyr, add a new column: max_day which has the number of last day for each combination of year, month, and stock

c) Now that we know our last day of the month for which we have data for a given stock, we can use filter to only take our end of month data (rows where day = max_day) and then sort our data to be increasing (so that our last row is in 2017).

Note: We only want data for full months, so make sure not to include January 2018 data

d) Now that we have our month end values for each stock, we can find the growth percentage for each month using dplyr::lag function. Using group_by, mutate, dplyr::lag, and whatever else you need, find the growth in the closing price for each stock in each month.

Note that growth is: (final_price/initial_price) - 1

The first month of data should all have NA values for growth, why?

e) Remove the rows of our dataset with an NA value and then make a line chart showing the monthly growth values for stocks and bonds. Be sure to use appropriate labelling and add a line at 0 using geom_hline().

What are your takeaways from this chart? Do any stocks seem consistently better than any other? Which one seems the most erratic?

f) Why do we care about growth rates and not just the closing price values?

Question 3

It's time to build our retirement portfolio simulator. Let's start by trying to recreate what would happen if we invested in JCPenny in 2008.

```
# Pull out all the dates we will loop through
dates <- unique(month_end$date)
# Invest $100 at the beginning, no monthly contribution
investment_value <- 100

for(d in dates){
  investment_value <- investment_value* (1 + month_end[month_end$date == d &
                                                       month_end$Stock == "JCP", "growth"])
}
names(investment_value) <- "Value"
print(investment_value)
```

Wow, what a terrible investment, our $100 turned to $7 in ten years!

a) One easy way we can make our code better is by getting rid of that whole month_end[month_end$date ... section. We can do this by making a function.

Write a function: growth_finder with the following characteristics

- Inputs: date, stock symbol, reference table (use month_end)
- Output: growth value for a stock on a given date

Test your function and make sure you get the same output:

```
# Test the function, check the raw table to be sure you get the correct value
growth_finder(as.Date("2008-12-31"),
              "JCP",
              month_end)
```

Let's re-envision our retirement forecasting loop with this new function

```
# Pull out all the dates we will loop through
dates <- unique(month_end$date)
# Invest $100 at the beginning, no monthly contribution
investment_value <- 100

for(d in dates){
  investment_value <- investment_value* (1 + growth_finder(d, "JCP", month_end))
}
print(investment_value)
```

c) Now let's replace the right side of our for loop assignment statement: investment_value * (1 + ...) with a new function called: monthly_update

- Inputs: investment_value at beginning of the month, all inputs to growth_finder
- Outputs: investment value at end of month

Check that you get the same result as I do:

```
monthly_update(100, as.Date("2008-12-31"),
               "JCP", month_end)
```

Let's envision our retirement loop with our new function now:

```
# Pull out all the dates we will loop through
dates <- sort(unique(month_end$date))
# Invest $100 at the beginning, no monthly contribution
investment_value <- 100

for(d in dates){
  investment_value <- monthly_update(investment_value,
                                     d, "JCP", month_end)
}
print(investment_value)
```

d) Using the code from above, fill in the below loop so that we can forecast for two stocks instead of just one

Question 4

a) Instead of using a for loop, let's turn the code for question 3d into a function that also allows us to divide up our allocated contributions among our different stocks. Make sure that you get the same results as I do when you run the function:

```
portfolio_finder(month_end, 1000, "JCP", "AAPL", 0.5, 0.5)
```

So, we've managed to update our portfolio_finder() function to work with 2 stocks, but in our data we have 8 different stocks! Instead of adding in more arguments to our function, we should create a single function to do all of the work for a single stock and then write a wrapper around it so that we can have that function do our work for all 8 stocks separately.

Below I've written a function to do just that and incorporate monthly contributions into our investments

Remember, the initial value is how much we invest on the first day, while the monthly contribution is a fixed sum of money that we add to the investment portfolio each month.

- Inputs: stock symbol, reference table, initial investment, monthly allocation, share of contributions to be invested in that stock
- Output: final value of our investment

b) Using our single_stock_value() function rewrite the portfolio_finder() function.

Inputs: reference table, initial contribution, monthly contribution, vector of stock symbols, vector of allocation shares for initial/monthly contributions

Output: Table showing the stocks invested in and their final values

Test your function to be sure that you get the same results as I do.

```
portfolio_finder(month_end, 100, 50,
                 c("BND", "VBLTX", "AAPL", "FORD", "JPM", "GE", "JCP", "EXC"),
                 c(0.05, 0.05, 0.3, 0.1, 0.1, 0.1, 0.1, 0.2))
```

c) Time to practice good function hygiene and update our portfolio_finder function.

- what value should all of our contribution shares add up to? Make sure that they do.

Test that your version gives the same results as mine for the following call:

```
portfolio_finder(month_end, 1000, 500,
                 c("BND", "VBLTX", "AAPL", "FORD", "JPM", "GE", "JCP", "EXC"),
                 c(0.05, 0.05, 0.3, 0.1, 0.1, 0.1, 0.1, 0.2))
```

d) Now let's try creating some hypothetical portfolios to see how large your investments could have gotten.

- Create 3 different portfolios, each with a $100 initial contriubtion and a $50 monthly contribution
- For each portfolio you have to invest in at least 4 of the bonds or stocks and you can only put a maximum of 50% of your funds to any single stock/bond in your portfolio. (No, you can't only buy AAPL)
- Of your fictional portfolios which one was worth the most at the end of 10 years? which one was worth the least? What was different about your best and worst portfolios?