Lab-14

your name here

5/3/23

Preface

The goal of this assignment is to help you gain familiarity with writing functions. As always, please come to office hours and reach out to your teaching staff if you have any questions.

Data

We will start by wrapping up the example from Tuesday using data from the S&P 500.

Then we will work with the monthly Zillow data from Zillow Research that we have seen before. As a reminder, `Metro_median_sale_price_uc_sfrcondo_month.csv` contains raw monthly median sale price data. Smoothed and seasonally adjusted price data is in `Metro_median_sale_price_uc_sfrcondo_sm_sa_month.csv`.

Part 1: S&P 500 Returns

By the end of class on Tuesday, we had created a function that took a stock's ticker symbol as its argument. For valid tickers, it computed and returned annualized returns over 2018-2022. For invalid tickers, it returned an informative error message. Here is that function:

```
get_annual_return <- function(ticker){
  symbols <- prices |> pull(symbol) # or instead put prices$symbol within the if()
  if(ticker %in% symbols){
    prices |>
      filter(symbol == ticker) |>
      filter(date==min(date) | date==max(date)) |>
      mutate(cumulative_return = (adjusted - lag(adjusted)) / lag(adjusted) ) |>
      filter(!is.na(cumulative_return)) |>
      transmute(annualized_return = ((1 + cumulative_return)^(1/5) - 1) * 100) |>
      pull(annualized_return)
  } else {
    str_glue("The ticker {ticker} is not available.")
  }
}
```

Here are some examples of what we get from calling that function:

```
get_annual_return("AAPL")
```

```
[1] 25.97665
```

```
get_annual_return("DYSON")
```

```
The ticker DYSON is not available.
```

1. Mapping over multiple tickers: We could extend our computations to a vector of tickers using imperative or functional programming techniques. Let's use the latter. Use `map()` to compute returns for `c("AAPL", "AMZN", "TSLA")` rather than calling `get_annual_return()` separately for each ticker.

Now, instead of returning the numbers as a list, using `map_dbl()` to return them as a double vector.

Note: if you use a vector of tickers that includes one or more invalid tickers, `map()` will still work fine. However, `map_dbl()` will produce an error because our function `get_annual_return` is designed to return an error message rather than a missing value. If we wanted, we could instead have it `print()` an error and then return `NA`, and then `map_dbl()` would work.

2. Now let's allow the function to compute returns for multiple tickers directly, rather than using `map()` to do so. Write a function that filters the data to contain rows that meet the criterion `symbol %in% ticker`, compute annualized returns for each one, and then return the results in a data frame. Test it out with the tickers `c("AAPL", "AMZN", "TSLA")`, and then with the tickers `c("AAPL", "DYSON", "TSLA")`.

3. Modify your code from 2. to make sure that tickers that are absent from the data are returned rather than removed. Test it out with the tickers `c("AAPL", "DYSON", "TSLA")`. *Hint: at the end of the function, you could use `tibble()` to convert `ticker` into a data frame, and then join this data frame and the data frame of annual returns.*

Part 2: Zillow Data

Here is some code that imports and tidies Zillow sales price data in a few steps: (1) importing a file based on the string it starts with, (2) dropping the observations for the region of "United States", (3) seperating `RegionName` into two variables `City` and `State`, (4) pivoting the data longer, and (5) converting `date` into date format and generating the variables `year` and `month`.

```
raw_tidy_example <- "Metro_median_sale_price_uc_sfrcondo_month.csv" |>
  read_csv() |>
  filter(RegionName != "United States") |>
  separate(RegionName, c("City","State"), sep =",") |>
  pivot_longer(-c(RegionID, SizeRank, City, State, RegionType, StateName),
               names_to = "date", values_to = "price") |>
  mutate(date = ymd(date), year = year(date), month = month(date))
```

1. Use the code on the previous page to write a function `tidy_data` that takes the argument `filename` and returns the tidied data. Call this function once to tidy the raw and once to tidy the adjusted price data, assigning the output to `raw_tidy` and `adj_tidy`. What years do `raw_tidy` and `adj_tidy` cover, respectively?

2. Can you modify the function in question 1 so that it can be used to tidy either price data or sales data, based on a second argument that specifies which one the user is asking for? What years do the sales data cover?

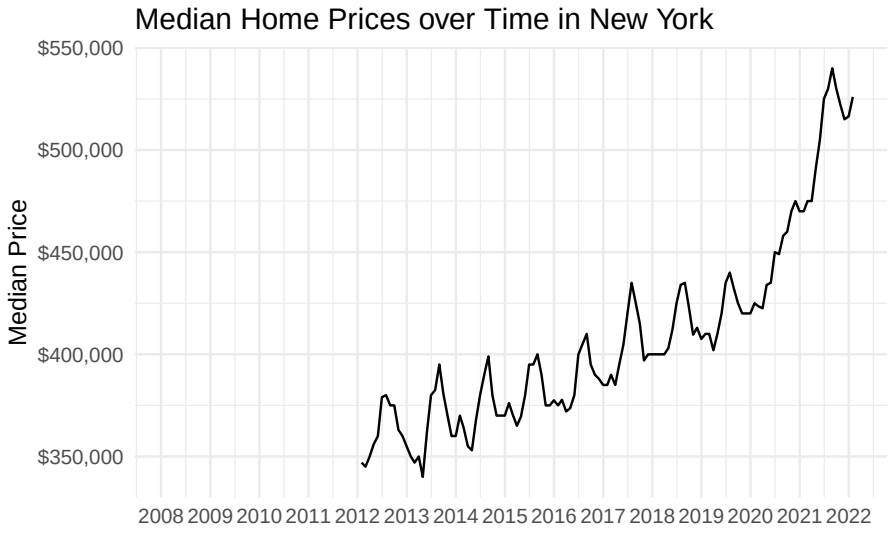
`Metro_sales_count_now_uc_sfrcondo_month.csv` contains monthly sales counts.

Note: an alternative way to approach this would be to detect whether the data are price or sales based on the filenames. We could also start by checking to make sure the file exists, and returning an error if not. See the solutions for an example.

Plotting Zillow Data

Here is some code that plots the time series of raw prices over time for one city:

```
raw_tidy_example |> # use data we just tidied
  filter(City == "New York") |>
  ggplot() +
  geom_line(aes(x = date, y = price)) +
  scale_x_date(breaks = scales::breaks_width("1 year"),
               labels = scales::label_date("%Y"))  +
  scale_y_continuous(labels=scales::dollar_format()) +
  labs(x = "",
       y = "Median Price",
       title = "Median Home Prices over Time in New York") +
  theme_minimal()
```



3. Adapt the code above to create a function `plot_city_price` that plots the time series of raw prices over time for whatever city is provided as its argument. Call this function to make a plot for `"Los Angeles-Long Beach-Anaheim"`.

4. Uncomment the code below to see how we can use `map()` in conjunction with `walk(print)` to print the output of map without returning the objects themselves and their annoying indices.

```
# c("New York", "Los Angeles-Long Beach-Anaheim", "Chicago") |>
#   map(plot_city_price) |>
#   walk(print)
```