R Project - Aggregate Data in ADH

Ulrich Bergmann Lachlan Deer

We want to look for some aggregate facts about the US economy and its trade patterns to get a sense of how some macroeconomic indicators had evolved around China's WTO accession. In particular we will show that

- Expansion of Chinese Trade. Essentially all of US trade growth since the 1990s is from the expansion
- Fall in Real Interest Rates Around the time the Chinese trade expanded.
- Expansion of the Trade Deficit during this time period.

Load Necessary Packages

Install the ones you do not have yet.

```
library("fredr")
library("purrr")
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library("readr")
library("tidyr")
library("ggplot2")
library("magrittr")
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
## The following object is masked from 'package:purrr':
##
##     set_names
```

```
library("lubridate")
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
```

```
##
##     date
```

```
library("PerformanceAnalytics")
```

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

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

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

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

```
##
## Attaching package: 'xts'
```

```
## The following objects are masked from 'package:dplyr':
##
##     first, last
```

```
##
## Attaching package: 'PerformanceAnalytics'
```

```
## The following object is masked from 'package:graphics':
##
##     legend
```

Get Data from Federal Reserve

We need the following variables from FRED:

```
codes = c("GDP", "IMP0015", "IMPCH", "EXP0015", "GS1", "CPILFESL")
```

We also need to tell FRED our API key to authenticate ourselves. For this course, you can use the following command:

```
api_key = "d89c49825894bf6e766a2e0afd8ba4f7"
fredr_set_key(api_key)
```

1. Use the `fredr_series_observations` command on a single variable to get the data for that variable

```
fredr_series_observations(
  series_id = "GDP"
)
```

```
## # A tibble: 296 x 3
##    date       series_id value
##    <date>     <chr>     <dbl>
##  1 1946-01-01 GDP         NA
##  2 1946-04-01 GDP         NA
##  3 1946-07-01 GDP         NA
##  4 1946-10-01 GDP         NA
##  5 1947-01-01 GDP        243.
##  6 1947-04-01 GDP        246.
##  7 1947-07-01 GDP        250.
##  8 1947-10-01 GDP        260.
##  9 1948-01-01 GDP        266.
```

```
## 10 1948-04-01 GDP        273.
## # ... with 286 more rows
```

2. Now, find the option to pull only data starting at `1990-01-01`. Make sure you format this number as a date.

```
fredr_series_observations(
  series_id = "GDP",
  observation_start = as.Date("1990-01-01")
)
```

```
## # A tibble: 120 x 3
##    date       series_id value
##    <date>     <chr>     <dbl>
##  1 1990-01-01 GDP       5873.
##  2 1990-04-01 GDP       5960.
##  3 1990-07-01 GDP       6015.
##  4 1990-10-01 GDP       6005.
##  5 1991-01-01 GDP       6035.
##  6 1991-04-01 GDP       6127.
##  7 1991-07-01 GDP       6206.
##  8 1991-10-01 GDP       6265.
##  9 1992-01-01 GDP       6363.
## 10 1992-04-01 GDP       6471.
## # ... with 110 more rows
```

3. What happens if you use the previous command on `codes` instead of a single variable name?

```
fredr_series_observations(
  series_id = codes,
  observation_start = as.Date("1990-01-01")
)
```

```
## Error: Argument `series_id` must be of length 1.
```

4. Solve it by applying the correct map function which returns a dataframe by row-binding. Save your dataset as `df_raw`

```
df_raw = map_dfr(codes,
             fredr_series_observations,
             observation_start = as.Date("1990-01-01")
)
```

Data Transformations

For the rest of this part, take `df_raw` and save the transformed output as `df`:

5. Now split your data into columns

```
df = pivot_wider(df_raw,
    names_from = series_id,
    values_from = value,
    id_cols = date
    )
```

6. Rename your newly created columns as such:
   - `gdp <- GDP`,
   - `imp_ch <- IMPCH`,

- imp_all <- IMP0015,
- exp_all <- EXP0015,
- t_bill <- GS1,
- cpi <- CPILFESL

```
df %<>%   rename(gdp = GDP,
          imp_ch = IMPCH,
          imp_all = IMP0015,
          exp_all = EXP0015,
          t_bill = GS1,
          cpi = CPILFESL
  )
```

7. `cpi` is coded in billions of USD while exports and imports are in millions, multiply `cpi` by `1000` to have all values in millions

```
df %<>% mutate(gdp = gdp * 1000)
```

8. create additional variables for our date using the `lubridate` module. We want columns `year`, `quarter`, `month`, `day` that only contain this part from the `date` column. Find the correct functions in `lubridate` to achieve this

```
df %<>%  mutate(
  year = year(date),
  quarter = quarter(date),
  month = month(date),
  day = day(date)
         )
```

9. Sort your data by `date`

```
df %<>% arrange(date)
```

We could have also achieved tasks 5-9 in a single step using multiple pipes like this:

```
df = pivot_wider(df_raw,
                 names_from  = series_id,
                 values_from = value,
                 id_cols     = date
    ) %>%
  rename(gdp     = GDP,
         imp_ch  = IMPCH,
         imp_all = IMP0015,
         exp_all = EXP0015,
         t_bill  = GS1,
         cpi     = CPILFESL
    ) %>%
  mutate(gdp     = gdp * 1000,
         year    = year(date),
         quarter = quarter(date),
         month   = month(date),
         day     = day(date)
         ) %>%
  arrange(date)
```

We see that gdp is coded quaterly while the imports and exports are per month. We need the data grouped annually and quarterly respectively for the next two parts:

Data Grouping

10. Group `df` quarterly into a dataframe called `df_quarter`. In this dataset you want

- `start_date` as the minimum of date,
- `gdp`, `imp_all`, `imp_ch`, `exp_all` all aggregated as sums (how do you deal with NA's?)
- `cpi`, `t_bill` aggregated as averages

```
df_quarter = group_by(df, year, quarter) %>%
  summarise(
    start_date = min(date),
    gdp        = sum(gdp,     na.rm = TRUE),
    imp_all    = sum(imp_all, na.rm = TRUE),
    imp_ch     = sum(imp_ch,  na.rm = TRUE),
    exp_all    = sum(exp_all, na.rm = TRUE),

    cpi        = mean(cpi,    na.rm = TRUE),
    t_bill     = mean(t_bill, na.rm = TRUE)
  )
```

11. Group `df` annually into a dataframe called 'df_year. In this dataset you want

- `gdp`, `imp_all`, `imp_ch` aggregated as sums (how do you deal with NA's?)

```
df_year = group_by(df, year) %>%
  summarise(
    gdp     = sum(gdp,     na.rm = TRUE),
    imp_all = sum(imp_all, na.rm = TRUE),
    imp_ch  = sum(imp_ch,  na.rm = TRUE),
    exp_all = sum(exp_all, na.rm = TRUE)
  )
```

Fact 1: Increase of Imports from China and the Rest of the World

For this exercise, we are going to use `df_year`

1. drop data from 2020
2. We want to create the following three variables:

- `global_share = 100 * imp_all / gdp`
- `china_share = 100 * imp_ch / gdp`
- `nonchina_share = global_share - china_share`

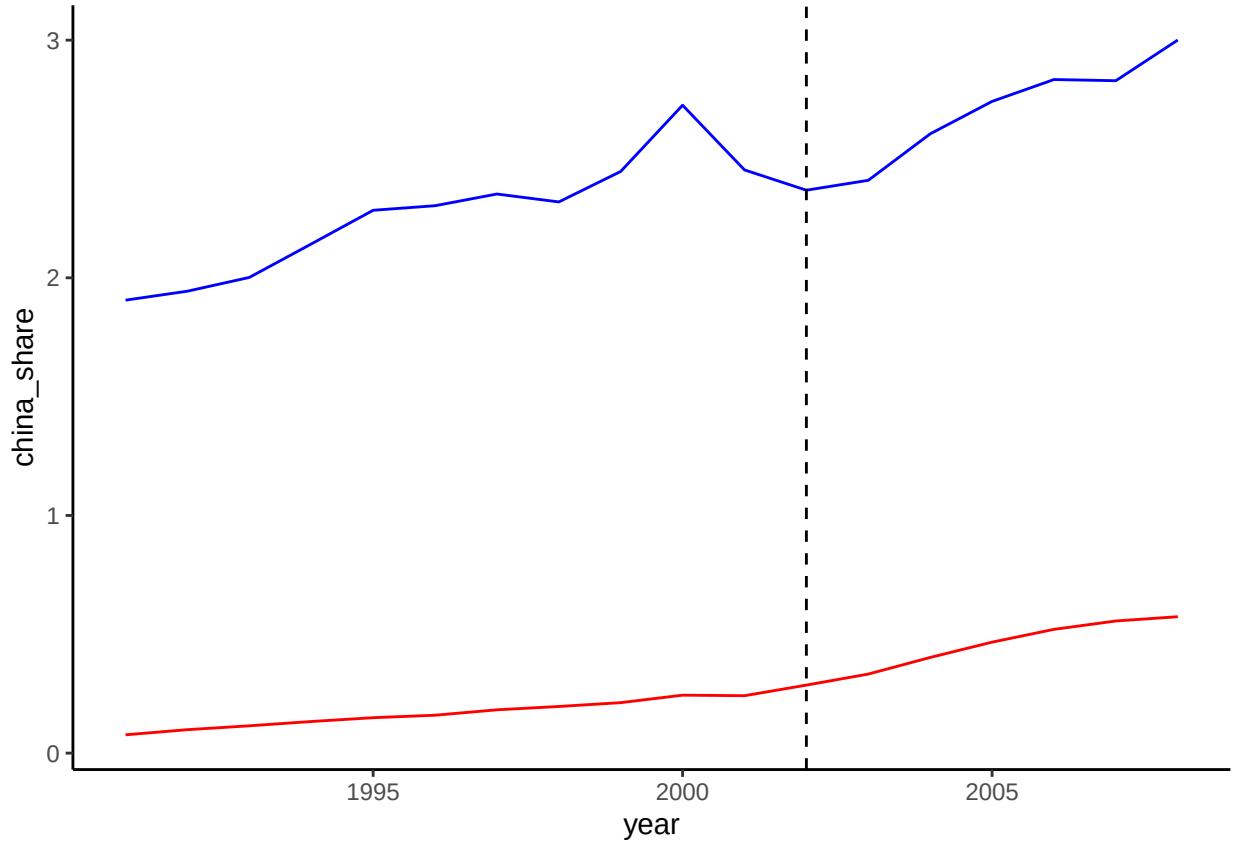
Steps 1 and 2 combined:

```
df_year %<>% filter(year != 2020) %>%
  mutate(global_share = 100 * imp_all / gdp,
         china_share = 100 * imp_ch / gdp,
         nonchina_share = global_share - china_share
  )
```

3. Create the following graph

- years between `1991` and `2008`
- x-axis: years
- y-axis: `china_share` and `nonchina_share` (add to lines to your plot with different colors)
- a vertical line for `x == 2002`

```r
df_year %>%
  filter(between(year, 1991, 2008)) %>%
  ggplot +
  geom_line(aes(
    x = year,
    y = china_share),
    color = "red"
  ) +
  geom_line(aes(
    x = year,
    y = nonchina_share),
    color = "blue"
  ) +
  geom_vline(xintercept = 2002,
             linetype = 2
             ) +
  theme_classic()
```



Fact 2: Increasing trade-deficit of the US

For this we will work with `df_year` again.

1. drop data from 2020

We have done already, if not repeat first step of `Fact 1: Step 1`

2. We want to create a variable `trade_deficit` (imports - exports)
3. And a variable `trade_deficit_share` for the share of the trade deficit compared to the gdp

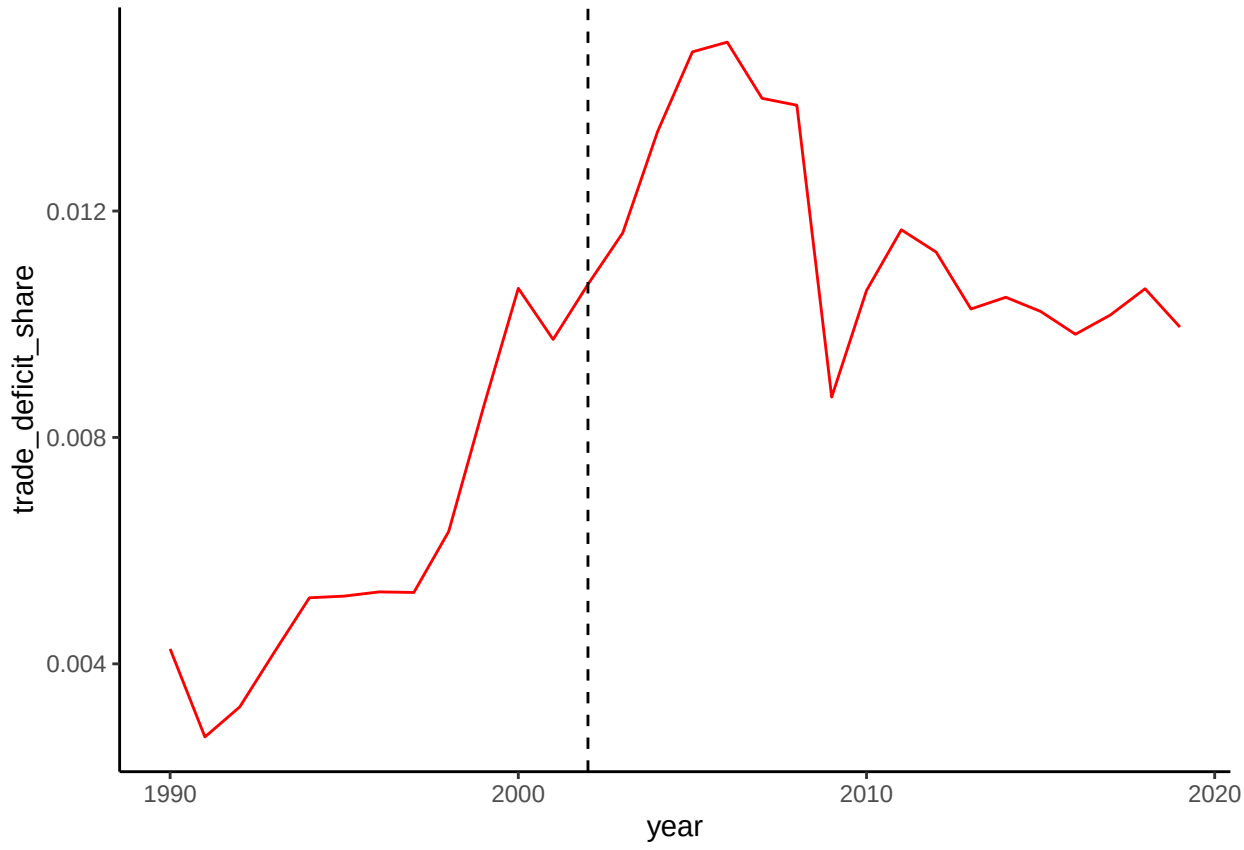
Steps 2 and 3 combined:

```
df_year %<>% mutate(trade_deficit       = imp_all - exp_all,
                    trade_deficit_share = trade_deficit / gdp
)
```

4. Plot `trade_deficit_share` over time
5. Pick a nice color
6. Add a vertical line to the plot for `year == 2002`

Steps 4-6 combined:

```
df_year %>%
  ggplot +
  geom_line(aes(
    x = year,
    y = trade_deficit_share),
    color = "red"
  ) +
  geom_vline(xintercept = 2002,
             linetype = 2) +
  theme_classic()
```



Fact 3: 400 basis point fall in real Interest rates leading into China Expansion

For this we will work with `df_quarter` again.

1. We need to calculate the inflation rate from the consumer price index (`cpi`) column
   - find out how to calculate this

```
df_quarter %>% mutate(infl = (cpi - lag(cpi)) / lag(cpi))
```

```
## # A tibble: 121 x 10
## # Groups:   year [31]
##     year quarter start_date     gdp imp_all imp_ch exp_all   cpi t_bill     infl
##    <dbl>   <int> <date>       <dbl>   <dbl>  <dbl>   <dbl> <dbl>  <dbl>    <dbl>
##  1  1990       1 1990-01-01 5872701 119756.  3112.  97596.  133.   8.13 NA
##  2  1990       2 1990-04-01 5960028 120937.  3488. 100626.  134.   8.27  0.0131
##  3  1990       3 1990-07-01 6015116 124329.  4494.  93903.  136.   7.83  0.0149
##  4  1990       4 1990-10-01 6004733 130292.  4147. 101466.  138.   7.30  0.0115
##  5  1991       1 1991-01-01 6035178 115929.  3551. 102760.  140.   6.44 NA
##  6  1991       2 1991-04-01 6126862 119158.  4012. 107640.  141.   6.24  0.00904
##  7  1991       3 1991-07-01 6205937 123923.  5623. 101468.  143.   5.89  0.0108
##  8  1991       4 1991-10-01 6264540 129447.  5788. 109862.  144.   4.87  0.00933
##  9  1992       1 1992-01-01 6363102 122405.  5049. 111229.  145.   4.36 NA
## 10  1992       2 1992-04-01 6470763 130883.  5712. 112984.  147.   4.22  0.00871
## # ... with 111 more rows
```

```
- look at the first lines of your dataset. What's the problem?
```

The data is still grouped in years. Lag returns a `NA` avalue value for the first period (Q1) of a year as it does not compute lags between groups. We need to ungroup the dataset first.

```
- ungroup the dataset first before you repeat the previous step.
```

```
df_quarter %<>%
  ungroup %>%
  mutate(infl = (cpi - lag(cpi)) / lag(cpi))
```

```
- look at the first lines of your dataset again.
```

```
head(df_quarter)
```

```
## # A tibble: 6 x 10
##    year quarter start_date     gdp imp_all imp_ch exp_all   cpi t_bill     infl
##   <dbl>   <int> <date>       <dbl>   <dbl>  <dbl>   <dbl> <dbl>  <dbl>    <dbl>
## 1  1990       1 1990-01-01 5872701 119756.  3112.  97596.  133.   8.13 NA
## 2  1990       2 1990-04-01 5960028 120937.  3488. 100626.  134.   8.27  0.0131
## 3  1990       3 1990-07-01 6015116 124329.  4494.  93903.  136.   7.83  0.0149
## 4  1990       4 1990-10-01 6004733 130292.  4147. 101466.  138.   7.30  0.0115
## 5  1991       1 1991-01-01 6035178 115929.  3551. 102760.  140.   6.44  0.0145
## 6  1991       2 1991-04-01 6126862 119158.  4012. 107640.  141.   6.24  0.00904
```

This worked as expected. You now have the quaterly inflation rate

2. Try to apply the `Return.annualized` function from the `PerformanceAnalytics` package to the newly created `infl` column inside `mutate`
   - what's the problem?

```
df_quarter %>% transmute(annum_return = Return.annualized(infl, scale = 4))
```

```
## # A tibble: 121 x 1
##    annum_return
```

```
##           <dbl>
##  1       0.0236
##  2       0.0236
##  3       0.0236
##  4       0.0236
##  5       0.0236
##  6       0.0236
##  7       0.0236
##  8       0.0236
##  9       0.0236
## 10       0.0236
## # ... with 111 more rows
```

The output of the function is equal for all rows because it is not vectorized. It actually just uses the first value of the column (Test this!)

3. We will solve this problem in two ways
    1. Use the function inside `map_dbl` instead (not in a `mutate`)
        - parse the output to a new column of `df_quarter` called `annum_return`

```
df_quarter$annum_return = map_dbl(df_quarter$infl, Return.annualized, scale = 4)
```

```
2. Use `Vectorize(Return.annualized)` inside `mutate` instead an save the output to `annum_return2`
    - What does the function do?
```

We actually have to define a vectorized version of the function first:

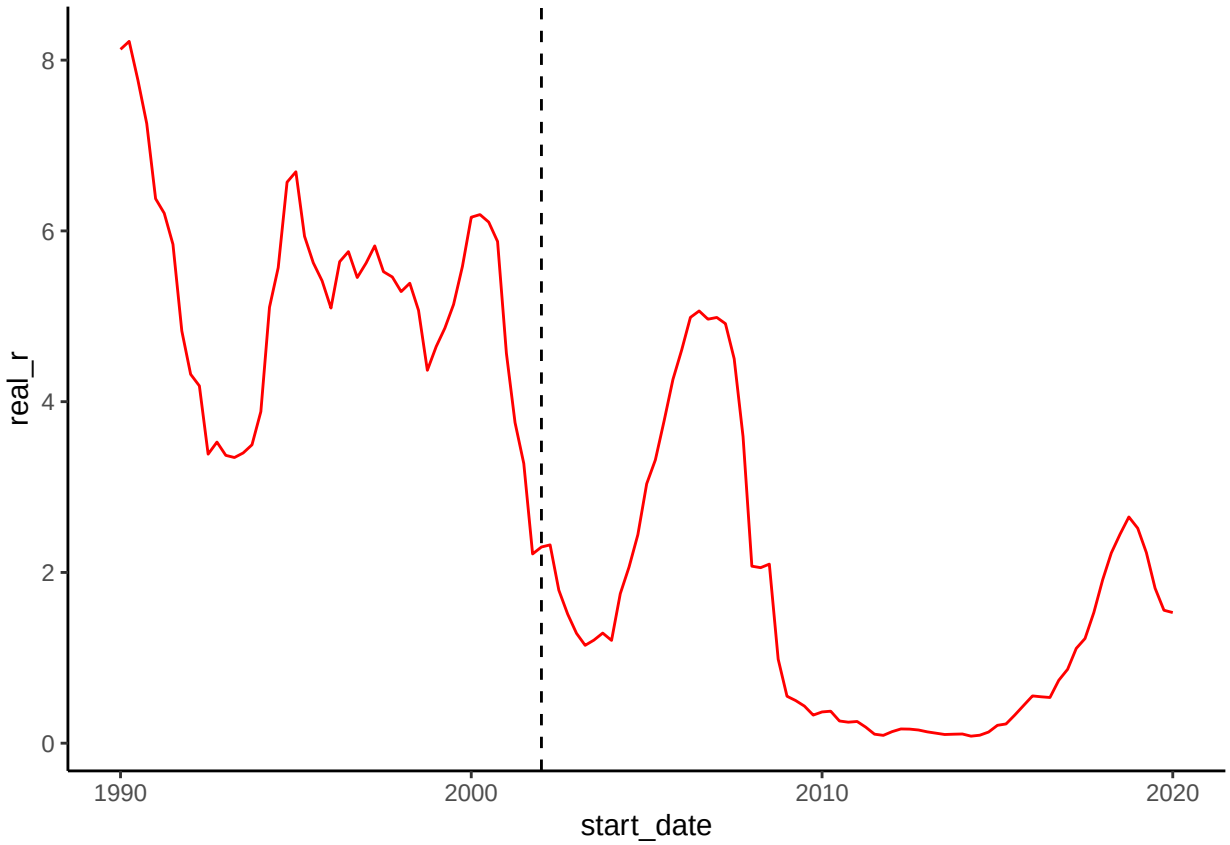
```
ret_an_v = Vectorize(Return.annualized)

df_quarter %<>% mutate(annum_return2 = ret_an_v(infl, scale = 4))
```

4. Create a new variable called `real_r` as the difference between `t_bill` and one of your `annum_return` columns

```
df_quarter %<>% mutate(real_r = t_bill - annum_return)
```

5. Plot `real_r` against date
6. Add a vertical line at the date `2002-01-01`
    - Hint: You have to parse the date first as a `date` and then convert it into a `numerical` value

```
df_quarter %>%
  ggplot +
  geom_line(aes(x = start_date,
                y = real_r),
            color = "red",
            ) +
  geom_vline(xintercept = as.numeric(as.Date("2002-01-01")),
             linetype = 2
             ) +
  theme_classic()
```


real_r
8
6
4
2
0
1990
2000
2010
2020
start_date