Data: Wrangle and Display it. Easily.

Contents

Data Cleaning

In this tutorial, we will be taking in some messy data from the World Bank, clean and reformat it to make it usable, join it with a classification table which we'll pull directly from Wiki, and produce some graphics from them. Happily, all of this can be done in about 50 lines of code (although we'll do extra stuff here).

The World Bank data is sourced here: located here http://databank.worldbank.org/data/reports.aspx?source=world-development-indicators

The Wiki that we are going to pull data from is: https://meta.wikimedia.org/wiki/List_of_countries_by_regional_classification.

Dependencies and setup

Make sure you have the following packages installed * `openxlsx` * `tidyr` * `dplyr` * `ggplot2` * `ggthemes` * `googleVis` * `htmltab` * `httr` * `magrittr` * `roperators`

I've attached an installation script that you can run to make sure the packages are all there:

```
pkgs = c("openxlsx", "tidyr", "dplyr", "ggplot2",
         "ggthemes", "htmltab", "data.table", "httr", "broom",
         "magrittr", "roperators")

# Hackfix tip:
# Opening the CRAN mirror in a browser can help with some restricted networks
utils::browseURL("http://cran.stat.auckland.ac.nz/")
utils::browseURL("https://meta.wikimedia.org/wiki/List_of_countries_by_regional_classification")


for (package in pkgs){
  if(package %in% rownames(installed.packages()) == FALSE)
    # Try will attempt to do what is in parentheses, but won't die if it doesn't work
    try(install.packages(package, repos = "http://cran.stat.auckland.ac.nz/"), silent = TRUE)
}

# Another hackfix for restricted networks.... just in case
httr::set_config( httr::config( ssl_verifypeer = 0L ) )
```

Load your required packages

```
require(openxlsx)
require(tidyr)
require(dplyr)
require(ggplot2)
require(ggthemes)
require(htmltab)
require(magrittr)
require(roperators)
```

Load the data

For convenience, I've downloaded data from the WorldBank already. Even though it's in excel format, we can still load it, but we need to use the `openxlsx` package to do so.

```
worldbank <- read.xlsx("../data/worldbank.xlsx")
```

Then, extract the country and region table from Wiki, you may need to run a special command to bypass some DHS network restrictiong.

```
httr::set_config(httr::config( ssl_verifypeer = 0L ))
wiki <- htmltab("https://meta.wikimedia.org/wiki/List_of_countries_by_regional_classification", which =
# which is used to select if multiple tables can be pulled from one page
```

Now, inspect the data you've loaded either from within RStudio's `Environment` tab, or by the `View()` command.

```
View(wiki)
View(worldbank)
names(worldbank)
```

`wiki` is okay, but `worldbank` has some serious problems:

1) Missing values are coded as ".." which has turned all of the numeric columns into text!
2) Values from each year are placed in different columns
3) Variables are seperated by row, not column meaning in each row are from different variables hence the yearly column statistics are meaningless
4) More than one row per observation (country & year)

Happily *this is easy to fix*

Recoding the missing values

Firstly, let's replace those ".." missing value codes with missing values (`NA`). We also want to convert the yearly columns into a `numeric` type. Think about the logic we want to use: we want to go through the columns of `worldbank` and in the year columns, which are named like `####.[YR####]`, turn all cells with `..` into `NA`, and then convert the columns into `numeric`.

Happily, this is all rather easy, but replacing the `..`s will requie a *regular expression* (periods are a bit tricky to deal with). *Regular expressions* are a tricky subject, basically they're specially formatted text used to select specific formats of text. You can use *regular expression*'s to do some very complicated text manipulation - but it's more efficient to just look on http://stackoverflow.com/questions/tagged/regex for an existing example of something similar to what you're wanting to do.

Note that the below code is a bit odd...

```
# You could also use an apply statement, but that is harder to read
for(this_column in names(worldbank)){
   if(grepl("YR", this_column)){
    # if "YR" is in this column name...
    worldbank[[this_column]] %regex<-% c("\\.\\.", NA)
    worldbank[[this_column]] <- num(worldbank[[this_column]])
    # Without roperators:
    # worldbank[[this_column]] <- as.numeric(gsub("\\.\\.", NA, worldbank[[this_column]]))
   }
}
```

For the sake of demonstration, we can also subset out the `Country.Code` and `Series.Code` columns like so:

```
df <- select(worldbank, which(!names(worldbank) %in% c("Country.Code", "Series.Code")))
```

I've cast that into a new `tibble` (a dataframe with some `dplyr` extras) called `df` - that will make it easier to go back and redo things, it also makes it easier to reference.

Take a look at `df` now, in particular the yearly value columns.

```
View(df)
```

Transform into long format

Now, we can focus on turning `df` into a (useable) *long format* dataset by gathering all of the year colluns together and spreading out the values of the `Series.Name` into different columns. This is where `tidyr` comes in handy.

Gather the yearly data together

`tidyr` has a handy function called `gather()` to do just this. You only need to specify:

data `key` - what the resulting aggregated variable will be called *`value` - what the new column for values will be called

```
df2 <- gather(df, key = year, value = value, -c(Country.Name, Series.Name))
head(df2, 5)
```

```
##   Country.Name                  Series.Name          year        value
## 1  Afghanistan            GDP (current US$) 1960.[YR1960] 5.377778e+08
## 2  Afghanistan        GDP growth (annual %) 1960.[YR1960]           NA
## 3  Afghanistan GDP per capita (current US$) 1960.[YR1960] 5.978768e+01
## 4  Afghanistan Population growth (annual %) 1960.[YR1960] 1.813677e+00
## 5  Afghanistan            Population, total 1960.[YR1960] 8.994793e+06
```

This is much better, but we still have to deal with

1) ~~Missing values are coded as ".." which has turned all of the numeric columns into text!~~
2) ~~Values from each year are placed in different columns~~
3) Variables are seperated by row, not column meaning in each row are from different variables hence the yearly column statistics are meaningless
4) More than one row per observation (country & year)

But now...

5) Year is a single column, but it is far from a nice number

To turn the values of `year` into nice `integers`, another regular expression is needed:

```
df2$year %regex% c("\\..*", "")
df2$year <- int(df2$year)
#<- as.integer(gsub("\\..*","",df2$year))
head(df2)
```

```
##   Country.Name                                                 Series.Name
## 1  Afghanistan                                        GDP (current US$)
## 2  Afghanistan                                      GDP growth (annual %)
## 3  Afghanistan                                GDP per capita (current US$)
## 4  Afghanistan                                Population growth (annual %)
## 5  Afghanistan                                           Population, total
## 6  Afghanistan Incidence of HIV (% of uninfected population ages 15-49)
##   year        value
## 1 1960 5.377778e+08
## 2 1960           NA
## 3 1960 5.978768e+01
## 4 1960 1.813677e+00
## 5 1960 8.994793e+06
## 6 1960           NA
```

Fixed. Now, to fix our last remaining issues.

Spread out each series into its own column

To fix our 3rd and 4th problems, we'll need to spread out the `Series.Name` column. Again, `tidyr` makes that easy for us.

`tidyr` has a handy function called `spread()` to do just this. You only need to specify:

*`data` `key` - The column whose values will be used as column headings *`value` - The column whose values will populate the cells.

```
df3 <- spread(df2, key = Series.Name, value =  value)
```

```
View(df3)
```

Excellent! Your data should now be in long format. All we need to do is add our region labels that we pulled from Wiki earlier and we'll be ready to do some work.

1) ~~Missing values are coded as ".." which has turned all of the numeric columns into text!~~
2) ~~Values from each year are placed in different columns~~
3) ~~Variables are seperated by row, not column meaning in each row are from different variables hence the yearly column statistics are meaningless~~
4) ~~More than one row per observation (country & year)~~
5) ~~Year is a single column, but it if far from a nice number~~

Join the two datasets together

There are several different ways to join data in R, be it base R, `dplyr`, or `data.table` (which wears the performance crown at the moment). Here, we'll only focus on staying within the **tidyverse** by `dplyr` to join two datasets.

First, rename the `Country.Name` column in `df3` to match the `Country` column in the `wiki` data. Remember that `dplyr` requires that you merge on columns with the same names.

```r
# Remember we need to explicitly change df3
df3 <- rename(df3, Country = Country.Name)
```

Then check for any countries in our data, `df3`, that don't match up to the names in the `wiki` data. This is a common issue with matching countries such as the Democratic People's Republic of Korea aka North Korea, DPRK, and Dem Rep Korea.

```r
uniq_df   <- unique(df3$Country)
uniq_wiki <- unique(wiki$Country)

## Print the mismatched names from df3 and wiki
uniq_df[!uniq_df %in% uniq_wiki]
```

```
##  [1] "Bahamas, The"                  "British Virgin Islands"
##  [3] "Cabo Verde"                    "Congo, Dem. Rep."
##  [5] "Congo, Rep."                   "Cote d'Ivoire"
##  [7] "Curacao"                       "Egypt, Arab Rep."
##  [9] "Gambia, The"                   "High income"
## [11] "Hong Kong SAR, China"          "Iran, Islamic Rep."
## [13] "Korea, Dem. People's Rep."     "Korea, Rep."
## [15] "Kosovo"                        "Kyrgyz Republic"
## [17] "Lao PDR"                       "Macedonia, FYR"
## [19] "Moldova"                       "Slovak Republic"
## [21] "St. Kitts and Nevis"           "St. Lucia"
## [23] "St. Martin (French part)"      "St. Vincent and the Grenadines"
## [25] "Tanzania"                      "Venezuela, RB"
## [27] "Virgin Islands (U.S.)"         "West Bank and Gaza"
## [29] "Yemen, Rep."
```

```r
# sort(uniq_wiki[!uniq_wiki %in% uniq_df]) # This prints a lot
```

There are quite a few mismatches, in practice we'd want to fix all of them, but for the sake of time (and tedium) we'll only change a few...

```r
# Feel free to copy and paste!!
# (in reality I'd have a lookup table as a seperate file to tidy up my code)
# Country names we want to change in wiki data
country_wiki <- c("Tanzania, United Republic of", "Korea, Democratic People's Republic of",
                  "Korea, Republic of" , "Iran, Islamic Republic of", "Congo, The Democratic Republic o
                  "Moldova, Republic of", "Hong Kong", "Egypt","Yemen")
# What to change them to
country_df <- c("Tanzania", "Korea, Dem. People's Rep.",  "Korea, Rep." ,
                "Iran, Islamic Rep.", "Congo, Dem. Rep.", "Moldova",
                "Hong Kong SAR, China", "Egypt, Arab Rep.","Yemen, Rep.")
idx <- 1
for(x in country_wiki){
  # print(x)
  # print(idx)
  wiki$Country[wiki$Country == x] <- country_df[idx]
  idx %+=% 1
}
```

Now that the ID columns match, you can use `dplyr` to merge the two together. We'll want

```r
df_full <- left_join(df3, wiki, by = "Country")
```

Inspect dt_full now - note the last two columns.

Left vs Right joins

We use a left join because we care about the `df3` data i.e. keep `dt3`'s data, fill in fields that aren't matched in `wiki` with `NA`.

`dplyr`

We just used `dplyr` to merge our datasets with `left_join`, let's take a more detailed look at it and see why it's currently the most popular R package. Think of `dplyr` like a data manipulation pipeline. While a lot of its fucntions can be achieved in base R, it offers a cleaner syntax while allowing opperations to be chained together. The syntax invokes the pipe opperator, `%>%`, which passes the result of one opperation to the next. You can think of `%>%` as meaning **THEN**

The workfolow therefore becomes:

`function_1(data) %>% function_2 %>% function_3`

Where results are returned at the end of the pipeline and data are passed implicitly from `function_1` to `function_2` and from `function_2` to `function_3`

The verbs of `dplyr` we will use

`dplyr` has quite a few functions (including its own version of joins), the main ones we will focus on here are:

`filter` *- Get a subset of rows* `select` - Get a subset of columns ***`group_by`*** *- Tag data for grouped calculations* `summarise` - Create aggregated data summaries and apply functions to data ***`mutate`*** *- Add a new variable (also works with grouped data)* `rename` - Rename variables ***`arrange`*** *- Sort the data by selected columns* `do` - Do an arbitrary thing

Try using the verbs

`rename` some variables

Make the long an cumbersome names to something nicer - note syntax looks backward

```
# Note the back quotes for non-standard column names
df_full <- rename(df_full, growth = `GDP growth (annual %)`,
                  inflation = `Inflation, consumer prices (annual %)`)
```

`select` the important data

Take a subset of columns you'll actually use here

```
df_sub <- select(df_full, Country, Region, year, growth, inflation)
names(df_sub)
```

```
## [1] "Country"   "Region"    "year"      "growth"    "inflation"
```

Use `select` to rearrange columns

```
# Before
names(df_sub)
```

```
## [1] "Country"   "Region"    "year"      "growth"    "inflation"
```

```
# After (use everything() as a shortcut for everything else)
df_sub <- select(df_sub, year, Region, everything())
names(df_sub)
```

```
## [1] "year"      "Region"    "Country"   "growth"    "inflation"
```

filter to take a subset of rows

Find all African countries that begin with an S

```
filter(wiki, Region == "Africa" & grepl("^S", Country))
```

```
##                 Country Region Global South
## 1            Seychelles Africa Global South
## 2           South Sudan Africa Global South
## 3          Saint Helena Africa Global South
## 4          Sierra Leone Africa Global South
## 5               Senegal Africa Global South
## 6 Sao Tome and Principe Africa Global South
## 7             Swaziland Africa Global South
## 8          South Africa Africa Global South
```

group_by and summarise- time to chain

Find the average growth for all African countries that begin with an S

```
df_sub %>%
  filter(Region == "Africa" & grepl("^S", Country)) %>%
  group_by(Country) %>%
  summarise(avg_growth = mean(growth, na.rm = TRUE))
```

```
## # A tibble: 7 x 2
##   Country               avg_growth
##   <chr>                      <dbl>
## 1 Sao Tome and Principe       5.01
## 2 Senegal                     2.86
## 3 Seychelles                  4.59
## 4 Sierra Leone                2.68
## 5 South Africa                3.14
## 6 South Sudan                -4.29
## 7 Swaziland                   4.92
```

...then arrange the previous summary by average growth

```
df_sub %>%
  filter(Region == "Africa" & grepl("^S", Country)) %>%
  group_by(Country) %>%
  summarise(avg_growth = mean(growth, na.rm = TRUE)) %>%
  arrange(desc(avg_growth))
```

```
## # A tibble: 7 x 2
##   Country               avg_growth
```

```
##   <chr>                      <dbl>
## 1 Sao Tome and Principe       5.01
## 2 Swaziland                   4.92
## 3 Seychelles                  4.59
## 4 South Africa                3.14
## 5 Senegal                     2.86
## 6 Sierra Leone                2.68
## 7 South Sudan                -4.29
```

do something a bit different

Let's try regressions between inflation and growth in our African countries that begin with S

```
df_sub %>%
  filter(Region == "Africa" & grepl("^S", Country)) %>%
  group_by(Country) %>%
  do(model_list = lm(inflation ~ growth + year, data = . )) %>%
  # Since data isn't the first argument in lm(), normal piping won't work
  # To get around that, we use a . to refer to the dataframe in the pipe
  broom::tidy(model_list) %>%
  # broom::tidy(object_list) unpacks the models we made into a tible
  # Let's find only significant year effects
  filter(p.value < 0.05 & term == "year")
```

```
## # A tibble: 3 x 6
## # Groups:   Country [3]
##   Country    term  estimate std.error statistic p.value
##   <chr>      <chr>    <dbl>     <dbl>     <dbl>   <dbl>
## 1 Senegal    year    -0.196    0.0760     -2.58  0.0132
## 2 Seychelles year    -0.212    0.0987     -2.15  0.0372
## 3 Swaziland  year    -0.150    0.0608     -2.47  0.0180
```

Saving with ->

It can be handy to save results, you can do that nicely with the right-assignment arrow at the end of your pipeline like so:

```
df_sub %>%
  filter(year > 2000) %>%
  group_by(Country, Region) %>%
  summarise(av_growth = round(mean(growth, na.rm = TRUE),1),
            av_inflation = round(mean(inflation, na.rm = TRUE),1)) ->
  df4
```

Pipe and save with '%<>%

Furthermore, if you want to use a pipeline and save over the original data at the end, you can use one of the most underrated `magrittr` functions to do just that: `%<>%`

let's sort the factor levels in df4 such that they're in descending order relative to average growth. Happily it's just a matter of turning `Country` into a factor [again] with levels in the order that they aoppear in df4.

```
# To sort the rows of df4 by averge growth in descending order....
df4 %<>%
```

```
  ungroup() %>%  # The group by country we did earlier still holds so we need to ungroup it
  arrange(-(av_growth)) %>%
  mutate(Country = factor(.$Country, levels = .$Country))

# You can use . to refer to the data in the pipe in its present state -
# we did that because we want to feed in the factor levels as a sorted vector
```

ggplot2 & making graphs with help from dplyr

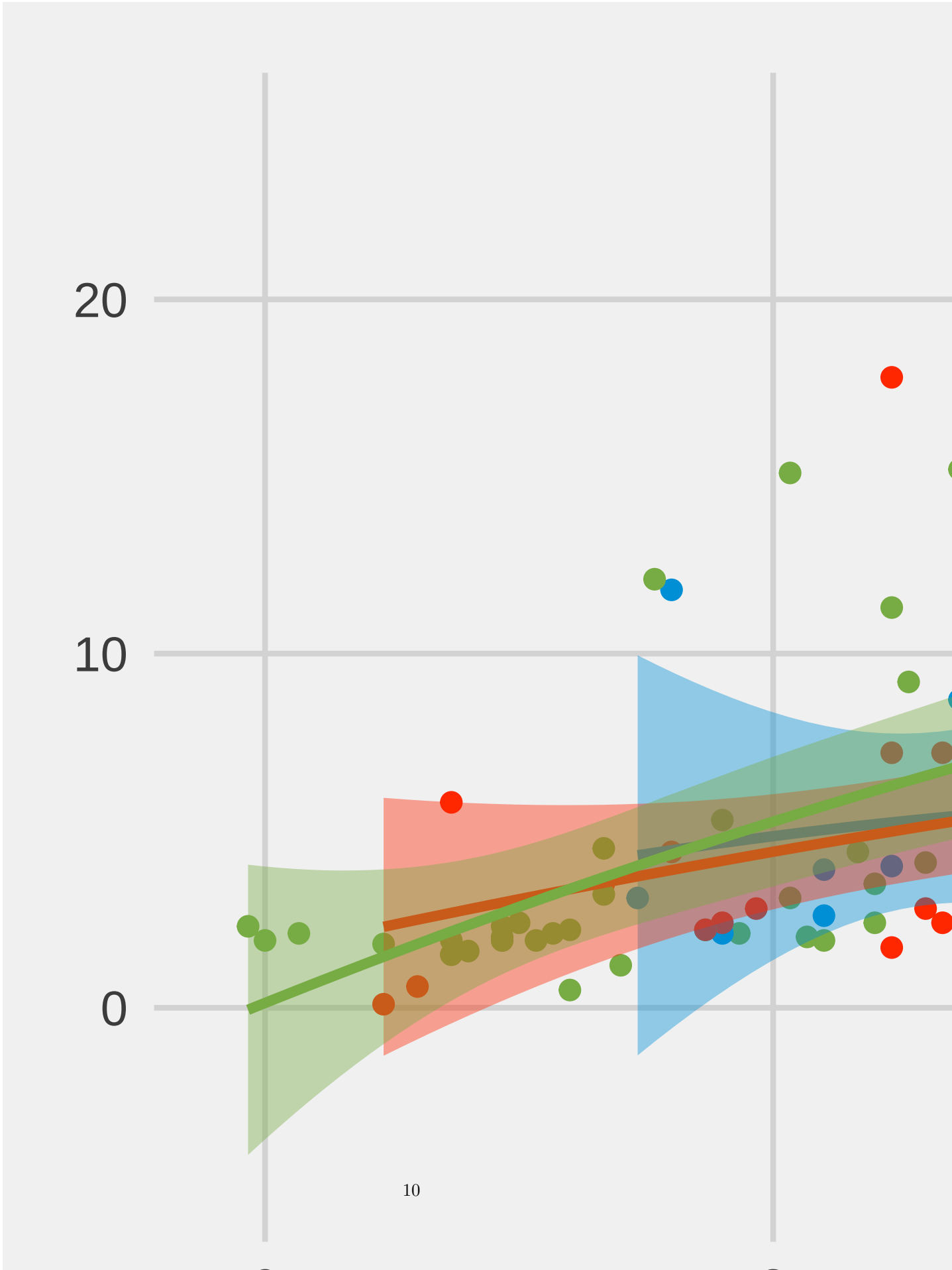
Now, let's combine ggplot2 and dplyr to show a graph of average growth and inflation from only countries from Europe and North America.

```
df4 %>%
  filter(Region %in% c("Europe","Asia & Pacific","Arab States") &
           av_inflation < 100) %>%
  ggplot(aes(x = av_growth, y = av_inflation, color = Region,
             fill = Region)) +
  # aes = aesthetic mapping, or what data ggplot will draw figures to
  geom_point() +
  geom_smooth(method = lm, formula = y ~ x + poly(x,2))+
  # geom_ribbon
  theme_fivethirtyeight() + # from ggthemes package
  scale_color_fivethirtyeight() +
  scale_fill_fivethirtyeight() +
  xlab("Average GDP growth") +
  ylab("Average inflation")
```

```
## Warning in predict.lm(model, newdata = data.frame(x = xseq), se.fit = se, :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = data.frame(x = xseq), se.fit = se, :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = data.frame(x = xseq), se.fit = se, :
## prediction from a rank-deficient fit may be misleading
```


20
10
0

Now, on to something a little more complicated

#TODO ggthemes seems a bit poorly now...

```
df4 %>%
  filter(Region %in% c("Europe", "North America")) %>%
  ggplot(aes(x = Country, y = av_growth)) +
  geom_bar(stat = "identity", alpha = 0.7, aes(fill = av_growth < 0)) +
  theme_solarized() +
  # a dashed horozontal line at 1
  geom_hline(aes(yintercept = 0), linetype = "dashed", size = 1) +
  # change the y label
  ylab("Average yearly growth since 2000") +
  # put the x-axis text on an angle
  theme(axis.text.x = element_text(angle = 90)) +
  # use a colorscheme to match the theme
  scale_fill_solarized(guide = FALSE) +
  # set the y axis ticks be
  scale_y_continuous(breaks = seq(from = -5, to = 5,by = .5)) +
  # Add label for the US
  geom_label(data=subset(df4, Country == "United States"),
             aes(label="USA"), vjust = 0, nudge_y = 0.02) +
  geom_label(data=subset(df4, Country == "Canada"),
             aes(label="Canada"), vjust = 0, nudge_y = 0.02)
```

```
## Warning: Removed 2 rows containing missing values (position_stack).
```


Average yearly growth since 2000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
rmenia
of Man
eorgia
elarus
oldova
onaco
lbania
huania
urkey
Latvia
omania
n Federation
stonia
oland
ulgaria
d Herzegovina
srael
eland
n Marino