Class_2_Notes

Data Prep

Load the following libraries *(we're adding stringr and lubridate now)*, and use read_csv to get the StoreSales.csv data. Put that in a dataframe.

```
library(tidyverse)
library(stringr)
library(lubridate)

# you may not need to do this - just use the read_csv
# but you will want to start organizing your workspace into folders
# in which case, you'll do something like setwd("~/DA1/Section 1") on the server

setwd("/home/ellen/Documents/Spring2020/DA1/Section I/class 2/Class 2 Data Files/")

dfStoreSales = read_csv("StoreSales.csv")
```

Inspect the data. We want this in **Tidy** format, which is:

- Each variable must have its own column.
- Each observation must have its own row.
- Each value must have its own cell.

So, we have work to do, as each store value *(observation)* has it's own column, and that's not tidy *("Store 1" is a value, not a variable)*. There are a couple of handy pivot functions *(pivot_longer and pivot_wider)* in R that we'll introduce here - they are particularly useful with business / accounting data *(especially all the junk you'll get from spreadsheets)*

First, let's transform the date to a date datatype *(did you notice chr when you inspected the data?)* Then, we'll transform the StoreSales into a long format:

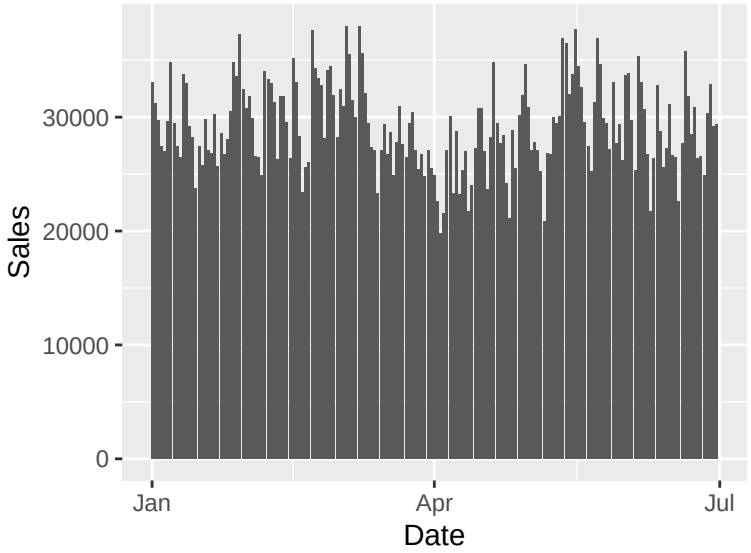
```
dfStoreSales = dfStoreSales %>% mutate(Date2 = mdy(Date))
# do it this way because if mdy fails, it will overwrite your data with NAs
# and you'll have to reload.

# you can use select to reduce, rename and reorder columns:

dfStoreSales = dfStoreSales %>% select(Date = Date2, "Store 1", "Store 2", "Store 3")

# Now for the pivot:

dfStoreSalesLong = dfStoreSales  %>%
pivot_longer(
   cols = 2:ncol(dfStoreSales),
   names_to = "Store",
   values_to = "Sales",
   values_drop_na = TRUE
 )
```

Notice the use of **ncol()** to get the last column number *(the syntax of pivot_longer takes a range first column : last column)*. This is the range of columns you're "gathering" up and creating new rows defined in "names_to". You also need to tell the pivot to write the values to a column names *("Sales" in this case)*. Just type ?pivot_longer for reference and help.

Visualization

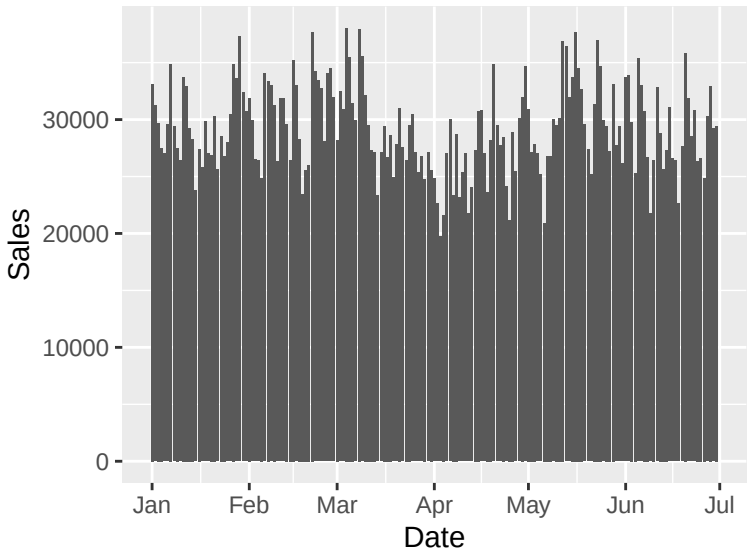
Let's take a look at the data. Just use a geom_bar:

```
p1 = ggplot(dfStoreSalesLong, aes(Date, Sales)) + geom_bar(stat = "identity")
p1
```



Let's show each month *(scale_x_date is handy - type ?scale_x_date)*:

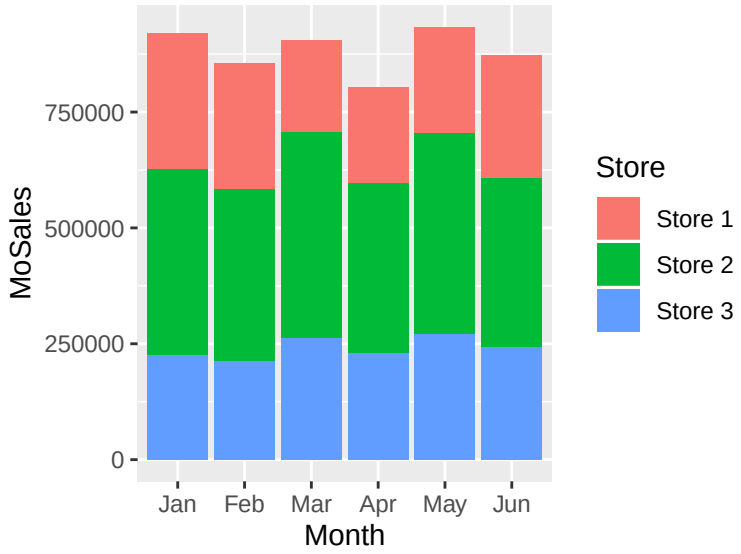
```
p1 = ggplot(dfStoreSalesLong, aes(Date, Sales)) + geom_bar(stat = "identity") +
  scale_x_date(breaks = "1 month", date_labels = "%b")
p1
```



```
# ?scale_x_date for documentation
```

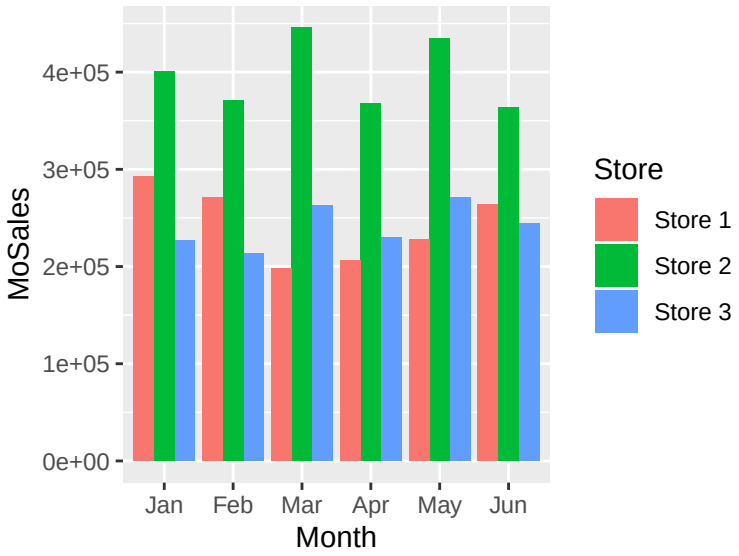
This is just too detailed because we have a sales amount for each day. Let's summarize Sales by month *(easy because the data is tidy :))*. Then look at sales by month, coloring by Store:

```
dfStoreSalesSummary = dfStoreSalesLong %>%
   group_by(Month = month(Date, label = T), Store) %>%
   summarise(MoSales = sum(Sales))

# label = T creates month text instead of month number

p1 = ggplot(dfStoreSalesSummary, aes(Month, MoSales, fill = Store)) +
   geom_bar(stat = "identity")
p1
```



This tells us a lot more. We can see that Store 2 is doing more business than 1 or 3, and we can see that our best month was May and the worst month was Apr. We could probably make this a little more clear:

```
p1 = ggplot(dfStoreSalesSummary, aes(Month, MoSales, fill = Store)) +
   geom_bar(stat = "identity", position = "dodge")
p1
```



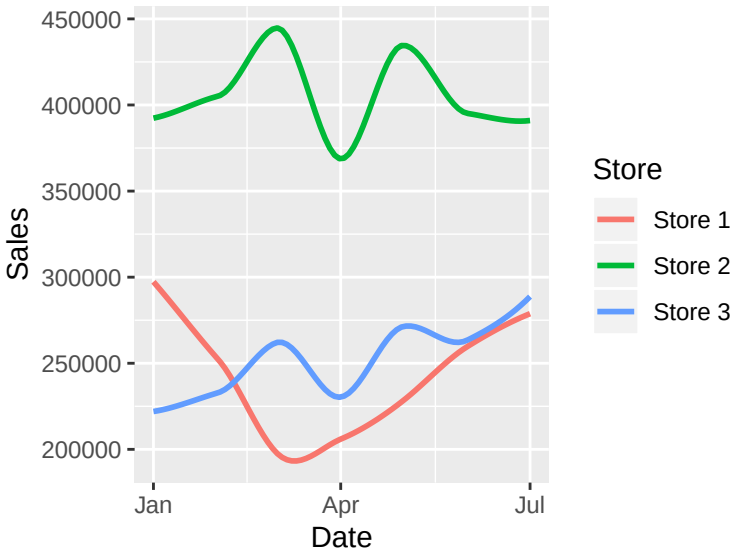
Even better! Now it's clear how much better Store 2 is doing - with it's best month being Mar.

Forecast

We need to prepare a forecast for July. Management is expecting Store 3 to increase Jun sales by 20%, Store 2 by 10% and Store 1 by 5%. One way to build this is to pull the Jun sales out and then multiply each stores sales by the appropriate factor, and then insert it back. But first, we'll pivot it back to wide format so we can operate on each stores data separately *(there are better ways to do this, but we'll save that for later)*:

Then we'll take a look at the new data using geom_smooth to look for trends:

```
dfStoreSalesSummaryWide <- dfStoreSalesSummary %>%
  pivot_wider(names_from = "Store", values_from = "MoSales")

dfStoreSalesSummaryWide = dfStoreSalesSummaryWide %>%
   mutate(Date = make_date(month = Month, day = 1, year = 2017))

Forecast = dfStoreSalesSummaryWide %>%
   ungroup() %>%
   filter(Date == "2017-06-01") %>%
   mutate(Date = Date + months(1)) %>%
   mutate (Month = month(Date, label = T)) %>%
   mutate(`Store 3` = `Store 3` *1.2, `Store 2` = `Store 2` * 1.1,
          `Store 1` = `Store 1` * 1.05) %>%
   bind_rows(dfStoreSalesSummaryWide) %>%
   pivot_longer(
   cols = starts_with("Store"),
   names_to = "Store",
   values_to = "Sales",
   values_drop_na = TRUE
 )

p2 = ggplot(Forecast, aes(Date, Sales, color = Store)) + geom_smooth(se = F)
p2
```



Note:

- Ungroup see r4ds 5.6.6. Generally, you can't perform row operations on a grouped dataframe or tibble. Check your environment to see if the object is grouped.
- Months(x) adds, month(x) extracts, see r4ds ch 16 *(watch for the s)*
- Cols = starts_with is a handy function to get all the columns that start with "xx..".