Projects in Data Science
Activity A1b: Data Wrangling

INSERT STUDENT NAME HERE

THE NUMBER OF DAILY BIRTHS in the US varies over the year and from day to day. What's surprising to many people is that the variation from one day to the next can be huge: some days have only about 80% as many births as others. Why? In this activity we'll use basic data wrangling skills to understand some drivers of daily births.

The data table `Birthdays` in the `mosaicData` package gives the number of births recorded on each day of the year in each state from 1969 to 1988.[1]

Table 1: A subset of the initial birthday data.

state	date	year	births
AK	1969-01-01	1969	14
AL	1969-01-01	1969	174
AR	1969-01-01	1969	78
AZ	1969-01-01	1969	84
CA	1969-01-01	1969	824
CO	1969-01-01	1969	100

Data Wrangling Introduction

Tidy Data

⊕

There are different ways to store and represent the same data. In order to be consistent and to also take advantage of the vectorized nature of `R`, the `tidyverse` packages we'll use provide a set of three interrelated rules/conventions for a dataset to be `tidy`:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

Graphical demonstration of tidy data from the RStudio Data Import Cheat Sheet.
Figure 1: Graphical demonstration of tidy data from the RStudio Data Import Cheat Sheet.

One of the first things we'll often do when acquiring new data is to "tidy it" into this form. For now, we can already start thinking of a data frame (tibble) as a table whose rows are the individual cases and whose columns are the variables on which we have information for each individual case. Figure 1 from the data import cheat sheet summarizes this principle.

Data Verbs

There are six main data transformation verbs in the `dplyr` library. Each verb takes an input data frame along with additional arguments specifying the action, and returns a new data frame. We'll examine them in three pairs.

Verbs that change the variables (columns) but not the cases (rows)

The first two verbs change which variables (columns) are included in the data frame, but preserve the same set of cases (rows).

- **`select()`** chooses which columns to keep, or put another way, deletes those columnmns that are not selected. To specify the columns, we can either list them out, or use functions like `starts_with()`, `ends_with()`, or `contains()` to specify the titles of the variables we wish to keep.

- **`mutate()`** adds one or more columns to the data frame. Each column is a function of the other columns that is applied on a row by row basis. For example, we can use arithmetic operations like

adding two other variables or logical operations like checking if two columns are equal, or equal to a target number.

Example 1 (select and mutate) (a) Add two new variables to the `Birthdays` data: one that has only the last two digits of the year, and one that states whether there were more than 100 births in the given state on the given date. (b) Then form a new table that only has three columns: the state and your two new columns (c) What does the following operation return: `select(Birthdays,ends_with("te"))`?

Solution. The commands for the first two parts are

```
BirthdaysExtra <- mutate(Birthdays,
                         year_short=year-1900,
                         busy_birthday=(births>100))
BirthdaysExtraTable <- select(BirthdaysExtra,state,
                              year_short,busy_birthday)
```

The operation in (c) selects only the first two columns `state` and `date`.

Verbs that change the cases (rows) but not the variables (columns)

The next two verbs change which cases (rows) are included in the data frame, but

preserve the same set of variables (columns).

- **`filter()`** deletes some of the rows by specifying which rows to keep.

- **`arrange()`** reorders the rows according to a specified criteria. To sort in reverse order based on the variable `x`, use `arrange(desc(x))`.

Example 2 (filter and arrange) Create a table with only births in Massachusetts in 1979, and sort the days from those with the most births to those with the fewest.

Solution. We want to `filter` and then `arrange`:

```
MABirths1979 <- filter(Birthdays, state=="MA", year==1979)
MABirths1979Sorted <- arrange(MABirths1979, desc(births))
```

Table 2: Birthdays in Massachusetts in 1979, sorted from those dates with the most births to those dates with the fewest births.

state	date	year	births
MA	1979-09-28	1979	262
MA	1979-09-11	1979	252
MA	1979-12-28	1979	249
MA	1979-09-26	1979	246
MA	1979-07-24	1979	245
MA	1979-04-27	1979	243

When filtering, we often use logical comparison operators like ==, >, <, >= (greater than or equal to), <= (less than or equal to), and %in%, which compares the value to a list of entries.[2] For example, if we want all births in AK, CA, and MA, we can write

```
filter(Birthdays, state %in% c("AK","CA","MA"))
```

The c() here is for concatenate, which is how we form vectors in R.

Grouped summaries

- **summarise()** (or equivalently **summarize()**) takes an entire data frame as input and outputs a single row with one or more summary statistics, such as mean, sum, sd, n_distinct(), or n() (which, like tally(), just counts the number of entries).

```
summarise(Birthdays,total_births=sum(births),
          average_births=mean(births),
          nstates=n_distinct(state),ncases=n())
```

```
##   total_births average_births nstates ncases
## 1     70486538       189.0409      51 372864
```

So summarise changes both the cases and the variables. Alone, summarise is not all that useful, because we can also access individual variables directly with the

dollar sign. For example, to find the total and average births, we can write

```
sum(Birthdays$births)
```

```
## [1] 70486538
```

```
mean(Birthdays$births)
```

```
## [1] 189.0409
```

Rather, we will mostly use it to create **grouped summaries**, which brings us to the last of the six main data verbs.

- **`group_by()`** groups the cases of a data frame by a specified set of variables. The size of the stored data frame does not actually change (neither the cases nor the variables change), but then other functions can be applied to the specified groups instead of the entire data set. We'll often use `group_by` in conjunction with `summarise` to get a grouped summary.

Example 3 (Grouped summary)

a. Find the average number of daily births in each year.

b. Find the average number of daily births in each year, by state.

Solution. We have to first group by the desired grouping and then perform a

summarise.

```
BirthdaysYear<-group_by(Birthdays,year)
summarise(BirthdaysYear, average=mean(births))
```

```
## # A tibble: 20 x 2
##     year  average
##    <int>    <dbl>
##  1  1969 192.3795
##  2  1970 199.7601
##  3  1971 191.2061
##  4  1972 174.8338
##  5  1973 168.8424
##  6  1974 170.1283
##  7  1975 169.2643
##  8  1976 170.0550
##  9  1977 178.8458
## 10  1978 179.1282
## 11  1979 188.0063
## 12  1980 193.8256
## 13  1981 195.2984
## 14  1982 197.9787
## 15  1983 195.6879
## 16  1984 196.8033
## 17  1985 202.2591
## 18  1986 202.0243
## 19  1987 204.8461
## 20  1988 209.6746
```

```
BirthdaysYearState<-group_by(Birthdays,year,state)
summarise(BirthdaysYearState, average=mean(births))
```

```
## # A tibble: 1,020 x 3
## # Groups:   year [?]
##     year state   average
##    <int> <chr>     <dbl>
##  1  1969    AK  18.64481
##  2  1969    AL 174.09783
##  3  1969    AR  91.25683
##  4  1969    AZ  93.32603
##  5  1969    CA 953.67027
```

```
##  6  1969    CO 109.86301
##  7  1969    CT 134.01635
##  8  1969    DC  75.28455
##  9  1969    DE  27.56403
## 10  1969    FL 292.14754
## # ... with 1,010 more rows
```

Piping

⊕

Pipes offer an efficient way to execute multiple operations at once. Here is a more efficient way to redo Example 2 with **the pipe**:

```
QuickMABirths1979<-
  Birthdays %>%
  filter(state=="MA",year==1979) %>%
  arrange(desc(births))
```

With the pipe notation, `x%>%f(y)` becomes `f(x,y)`, where in the first line here, `x` is `Birthdays`, the function `f` is `filter`, and `y` is `state=="MA",year==1979`. The really nice thing about piping is that you can chain together a bunch of different operations without having to save the intermediate results. This is what we have done above by chaining together a `filter` followed by an `arrange`.

Manipulating Dates

⊕

The `date` variable in `Birthdays` prints out in the conventional, human-readable way. But it is actually in a format (called `POSIX` date format) that automatically respects the order of time. The `lubridate` package contains helpful functions that will extract various information about any date. Here are some you might find useful:

- `year()`
- `month()`
- `week()`
- `yday()` — gives the day of the year as a number 1-366. This is often called the "Julian day."
- `mday()` — gives the day of the month as a number 1-31
- `wday()` — gives the weekday (e.g. Monday, Tuesday, ...). Use the optional argument `label=TRUE` to have the weekday spelled out rather than given as a number 1-7.

Using these `lubridate` functions, you can easily look at the data in more detail. For example, we can add columns to the date table for month and day of the week:[3]

```
Birthdays<-
  Birthdays %>%
  mutate(month=month(date,label=TRUE),
         weekday=wday(date,label=TRUE))
```

Here is what the data table looks like with our new columns:

Table 3: A subset of the birthday data with additional variables.

state	date	year	births	month	weekday
AK	1969-01-01	1969	14	Jan	Wed
AL	1969-01-01	1969	174	Jan	Wed
AR	1969-01-01	1969	78	Jan	Wed
AZ	1969-01-01	1969	84	Jan	Wed
CA	1969-01-01	1969	824	Jan	Wed
CO	1969-01-01	1969	100	Jan	Wed

Practice

Example 4 Make a table showing the five states with the most births between September 9, 1979 and September 11, 1979, inclusive. Arrange the table in descending order of births.

Solution. The plan of attack is to first filter the dates, then group by state, then use a `summarise` to add up totals for each state, and finally arrange them in descending order to find the top 5.[4]

```
SepTable<-
  Birthdays %>%
  filter(date >= ymd(19790909), date <=ymd(19790911)) %>%
  group_by(state) %>%
  summarise(total=sum(births)) %>%
  arrange(desc(total)) %>%
  head(n=5)

knitr::kable(
  SepTable[,], caption = 'States with the
```

```
  most births between September 9, 1979
  and September 11, 1979, inclusive.'
)
```

Table 4: States with the most births between September 9, 1979 and September 11, 1979, inclusive.

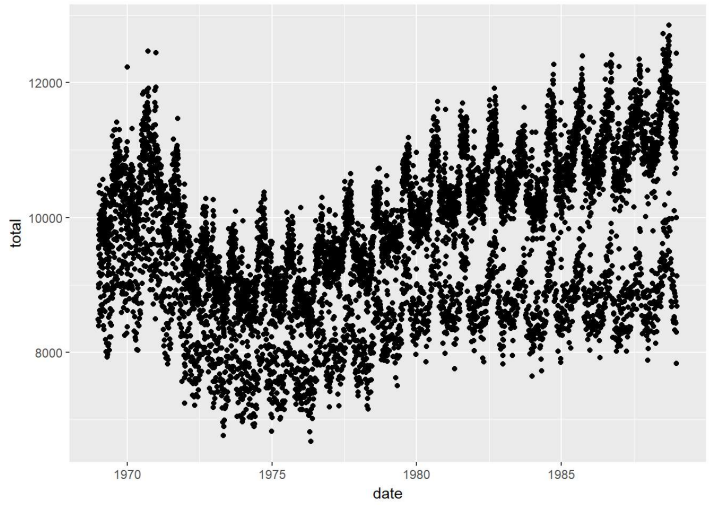
state	total
CA	3246
TX	2347
NY	1943
IL	1673
OH	1408

Driver 1: Seasonality

For this activity, we need to work with data aggregated across the states.

Exercise 1 (Total Across States) Create a new data table, `DailyBirths`, that adds up all the births for each day across all the states. Plot out daily births vs date.

```
DailyBirths<-
  Birthdays %>%
  group_by(date) %>%
  summarise(total=sum(births))
ggplot(DailyBirths,aes(date,total))+geom_point()
```



Exercise 2 (Examine Seasonality) To examine seasonality in birth rates, look at the number of births aggregated over all the years by

a. each week

b. each month

c. each Julian day

When are the most babies born? The fewest?

```
SeasonBirthdays <- Birthdays %>%
  select(date=date,births=births) %>%
  mutate(week=week(date),month=month(date),Julian=yday(date))
```

a. Seasonality by week (Max and min)

```
WeekBirths <-
  SeasonBirthdays %>%
  group_by(week) %>%
  summarise(total=sum(births))
WeekMaxMin <-
  WeekBirths%>%
```

```
  filter(total==max(total) | total==min(total))
knitr::kable(WeekMaxMin[,])
```

week	total
38	1477758
53	244602

The most births are on 39th week, and least on the last week of the year.

b. Seasonality by month

```
MonthBirths <-
  SeasonBirthdays %>%
  group_by(month) %>%
  summarise(total=sum(births))
MonthMaxMin <-
  MonthBirths %>%
  filter(total==max(total) | total==min(total))
knitr::kable(MonthMaxMin[,])
```

month	total
2	5362585
8	6309764

The most births are on August, and least on February

c. Seasonality by Julian day (max and min)

```
JulianBirths <-
  SeasonBirthdays %>%
  group_by(Julian) %>%
  summarise(total=sum(births))
JulianMaxMin <-
  JulianBirths%>%
  filter(total==max(total) | total==min(total))
knitr::kable(JulianMaxMin[,])
```

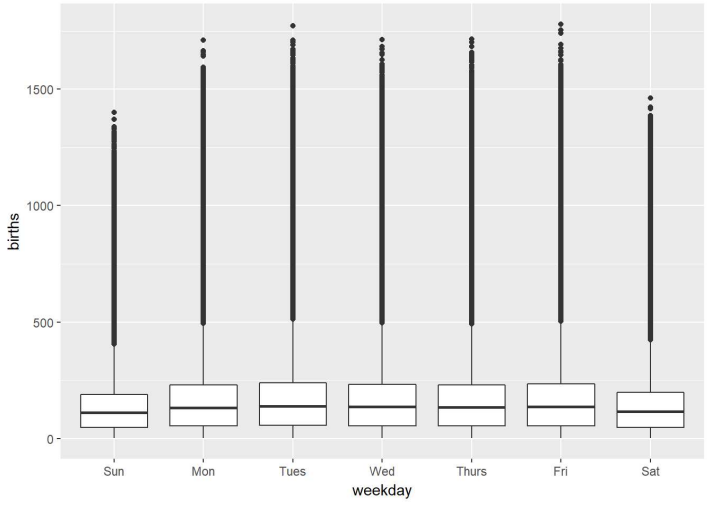
Julian	total
259	214372
366	46961

The most births are on Julian day 259, and the least are on the day 366

Driver 2: Day of the Week

Exercise 3 (Examine Patterns within the Week) To examine patterns within the week, make a box plot showing the number of births by day of the week. Interpret your results.

```
ggplot(Birthdays,aes(x=weekday,y=births))+geom_boxplot()
```

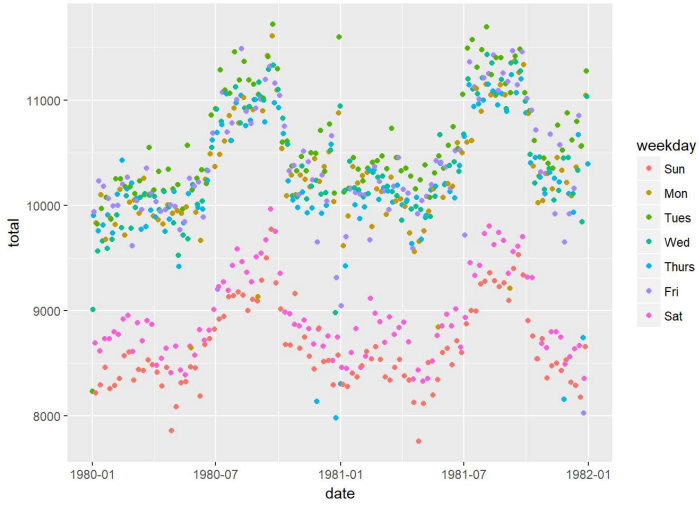


It seems like there are not as many births on Saturday and Sunday as weekdays. There are also a group of outliers with lots of births for all weekdays, probably during certain season or holiday period.

Driver 3: Holidays

Exercise 4 (Two Year Sample) Pick a two-year span of the `Birthdays` that falls in the 1980s, say, 1980/1981. Extract out the data just in this interval, calling it `MyTwoYears`. (Hint: `filter()`, `year()`). Plot out the births in this two-year span day by day. Color each date according to its day of the week. Explain the pattern that you see.

```
MyTwoYears <-
  Birthdays %>%
  filter(year==1980| year==1981)%>%
  group_by(date)%>%
  summarise(total=sum(births))%>%
  mutate(weekday=wday(date,label=TRUE))
ggplot(MyTwoYears,aes(date,total,colour=weekday))+geom_point()
```



It could be observed from the plot that weekends births are generally lower than weekdays. We could also observe the month seasonality and outliers.

THE PLOT YOU GENERATE for Exercise 4 should be generally consistent

with the weekend effect and seasonal patterns we have already seen; however, a few days each year stand out as excepetions. We are going to examine the hypothesis that these are holidays. You can find a data set listing US federal holidays at `http://tiny.cc/dcf/US-Holidays.csv`. Read it in as follows:[5]

```
Holidays <- read.csv("https://tiny.cc/dcf/US-Holidays.csv") %>%
  mutate(date = as.POSIXct(lubridate::dmy(date)))
```

Exercise 5 (Holidays) Now let's update the plot from Exercise 4 to include the holidays.[6]

a. Add a variable to `MyTwoYears` called `is_holiday`. It should be `TRUE` when the day is a holiday, and `FALSE` otherwise. One way to do this is with the transformation verb `%in%`, for instance, `is_holiday = date %in% Holidays$date`.

b. Add a `geom_point` layer to your plot that sets the color of the points based on the day of the week and the shape of the points based on whether or not the day is a holiday.

c. Finally, some holidays seem to have more of an effect than others. It would be helpful to label them. Use `geom_text` with the holiday data to add labels to each of the holidays.

a.

```
MyTwoYears <- mutate(MyTwoYears,is_holiday = date %in% Holidays$date)
```

b.

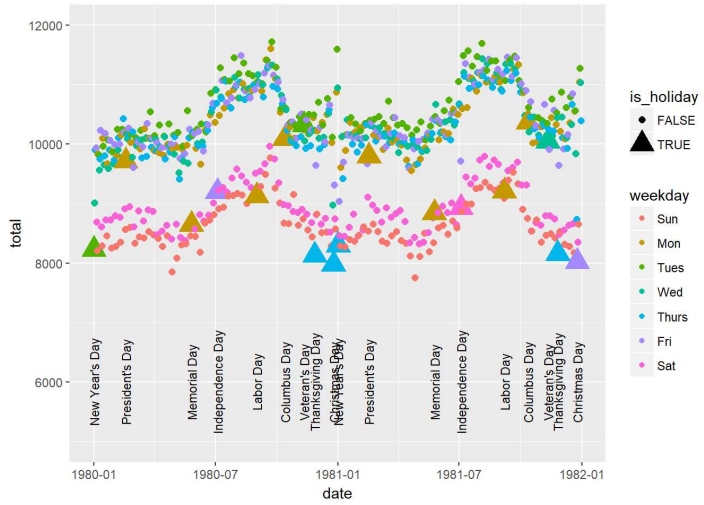
```
p=ggplot(MyTwoYears,aes(date,total))+geom_point(data=MyTwoYears,aes(col
p
```

```
## Warning: Using size for a discrete variable is not advised.
```

c.

```
TwoYearsHolidays <-
  Holidays %>%
  filter(year==1980 | year==1981)
p+geom_text(data=TwoYearsHolidays,aes(x=date,y=6000,label=holiday,angle
```

```
## Warning: Using size for a discrete variable is not advised.
```

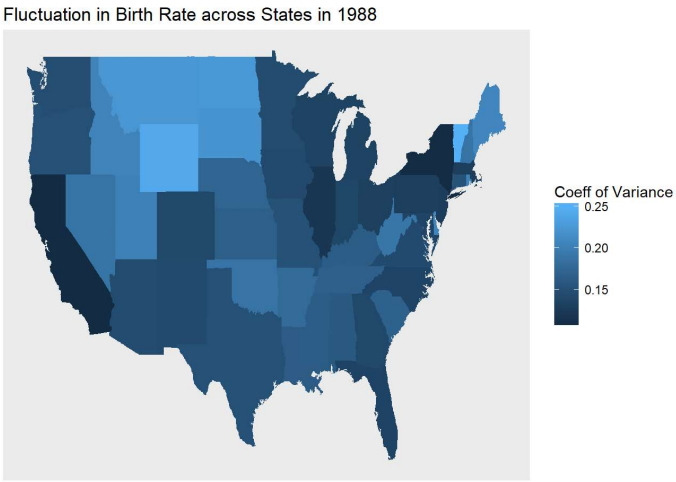


Driver 4: Geography

Exercise 6 (Examine the Effect of Geography) In any way you choose, explore the effect of geography on birth patterns. For example, do parents in Minnesota have fewer winter babies than in other states? Which states have the largest increases or decreases in their portion of US births over time? Is the weekend effect less strong for states with a higher percentage of their populations living in rural areas? Pick any issue (not all of these) that interests you, explore it, and create a graphic to illustrate your findings.

```
# Seasonality fluctuation comparision across states in 1988 using coeff
StateBirths98 <-
  Birthdays %>%
  filter(year==1988)%>%
  group_by(state)%>%
  summarise(cv=sd(births)/mean(births))
name <- cbind(state.abb,tolower(state.name))
name <- rbind(name,c("DC","district of columbia"))
states <-
```

```
  merge(map_data("state"),name,by.x="region",by.y = "V2")
my.df=merge(states,StateBirths98,by.x="state.abb",by.y="state")
ggplot()+
  labs(title="Fluctuation in Birth Rate across States in 1988",x='', y=
  geom_polygon(data=my.df,aes(long,lat,group=group,fill=cv))+
  scale_y_continuous(breaks=c()) +
  scale_x_continuous(breaks=c()) +
  scale_fill_continuous(name="Coeff of Variance")
```



Driver 5: Superstition

This article from FiveThirtyEight demonstrates that fewer babies are born on the 13th of each month, and the effect is even stronger when the 13th falls on a Friday. If you have extra time or want some extra practice, you can try to recreate the first graphic in the article.