Lecture 11: Wrangling Data with `tidyr`

UNIVERSITY OF
SAN FRANCISCO

Abbie M. Popa

BSDS 100 - Intro to Data Science with `R`

Outline

- Tidying Data
  - The `tidyr` package
  - The `dplyr` package
- Piping and the magrittr package

Reference: Chapter 12 in *R for Data Science* book

Additionally Complex Analysis

- In many cases, data doesn't come in the structure we'd like it to for easy analysis and plotting
- It is not uncommon that a majority of the time spent visualizing data is not writing the visualization code, e.g, `ggplot`, but rather *restructuring* data (mainly data frames) so as to be able to visualize the data

Additionally Complex Analysis

- There are functions in base `R` such as `aggregate()` and `merge()` that can help with this, as well as functions such as `melt()` and `cast()` from the `reshape2` package
- Here, we will examine the use of the `tidyr` and `dplyr` packages

Tidy Data

- Data should always be tidy before you begin to work with it!
- All data should be organized such that
  1. **each column is a variable**
  2. **each row is an observation**
- `tidyr` provides four main functions for tidying messy data
  1. `gather()`
  2. `spread()`
  3. `separate()`
  4. `unite()`

The `gather()` function

- `gather()` takes multiple columns and gathers them into key-value pairs
- It makes *wide* data *longer*
- Equivalent to the `melt()` function in the `reshape2` package
- Arguments to `gather()` are a data frame (can be a tibble!) a "key" which is the name for the new column identifying type/category of the each gathered observation, a "value" which is the name for the new column with the value data for each observation, and the columns to be gathered, or as a template:

```
gather(data frame, key = something, value =
something, some columns)
```

An Example

- In an experiment, you compare the *normal* weekly sales to sales when using *fancy* lighting and signage in a grocery store in three different cities
- The data may be stored as follows

```
> myDataFrame <- data.frame(
    City = c("Austin", "Georgia", "Vancouver"),
    Fancy = c(35000, 43000, 106000),
    Normal = c(30000, 44000, 77000)
    )

> myDataFrame
       City  Fancy Normal
1    Austin  35000  30000
2   Georgia  43000  44000
3 Vancouver 106000  77000
```

Example, continued

- **Goal**: Create a nice ggplot comparing the `Fancy` sales with `normal` sales from each location.
- In this case, we cannot directly use the data frame as it is formatted
- We can use the `gather()` function to reshape the data frame into two columns so that we can directly compare.

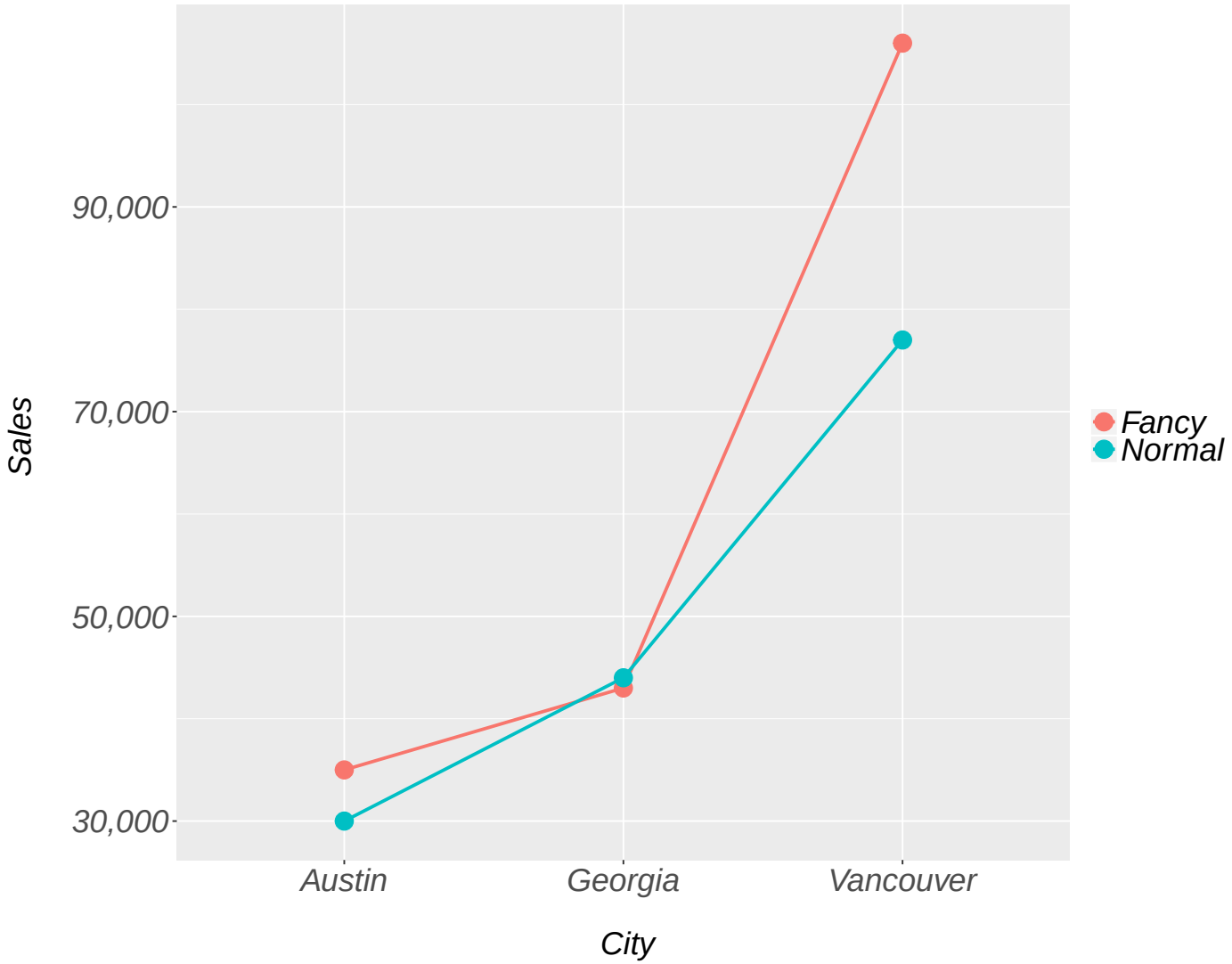
Example, continued

```
> library(tidyr)
> library(dplyr)

> (myTidyDataFrame <- gather(myDataFrame, key = lightSign,
    value = Sales,
    Fancy, Normal))

       City lightSign  Sales
1    Austin      Fancy  35000
2   Georgia      Fancy  43000
3 Vancouver      Fancy 106000
4    Austin     Normal  30000
5   Georgia     Normal  44000
6 Vancouver     Normal  77000
```

Now plot with ggplot

```
> library(ggplot2)

> ggplot(myTidyDataFrame,
    aes(x = City, y = Sales, group = lightSign,
     colour = lightSign)) +
     geom_line() +
     geom_point(size = 3)
```

The Result


90,000
70,000
50,000
30,000
Sales
Austin
Georgia
Vancouver
City
Fancy
Normal

The `separate()` function

- Sometimes variables are clumped together in a single column but we'd like to *separate* them
- The `separate()` function allows you to parse them
- The `extract()` function works similarly but uses regular expressions groups instead of a splitting pattern or position
- The opposite of `separate()` is `unite()`

Example

- There is an experiment which measures how much time (in hrs) a person has spent on their mobile phones at specific times in the day. The experiment controls for the mobile operating system (iOS or Android). Each subject has two readings taken at work and at home in the AM and PM.

- Goal Generate a plot with the mean time spent on mobile device (y), time of day (x) by location (work/home) and operating system (iOS/Android)

Example: Generating the Data

```
> set.seed(1979)

(mobile_time <- data.frame(
  uniqueId = 1:4,
  treatment = sample(rep(c('iOS', 'Android'), each = 2)),
  work_am = runif(4, 0, 1),
  home_am = runif(4, 0, 1),
  work_pm = runif(4, 1, 2),
  home_pm = runif(4, 1, 2)
))
```

	uniqueId	treatment	work_am	home_am	work_pm	home_pm
1	1	Android	0.1655928	0.47930296	1.912491	1.068928
2	2	iOS	0.2641590	0.36626685	1.276391	1.317642
3	3	iOS	0.6108628	0.34724530	1.011851	1.572617
4	4	Android	0.3877454	0.06951258	1.222138	1.759057

Gathering the data first

- We want to rearrange the `work_am`, `home_am`, `work_pm`, and `home_pm` columns, easier than listing all four of those, we can tell `R` which columns are not included in the `gather` using the minus sign

```
> mobile_time_tidy <- gather(mobile_time,
      key = sample, value = time,
      -uniqueId, -treatment)

> head(mobile_time_tidy)
  uniqueId treatment   sample      time
1        1   Android work_am 0.1655928
2        2       iOS work_am 0.2641590
3        3       iOS work_am 0.6108628
4        4   Android work_am 0.3877454
5        1   Android home_am 0.4793030
6        2       iOS home_am 0.3662669
```

Separating the Data

- The `separate()` function takes one column and makes it into two
- The arguments include the data frame to act on, the column in that data frame to separate, name for the new columns (`into =`), and what to separate on (`sep =`)
- Or, as a template:

```
separate(data frame, column name, into = c("first new column",
"second new column"), sep = "what to separate on")
```

Now separating the data

STEP 2: Split `sample` into location (work, home) and time of day (AM, PM) using `separate()`

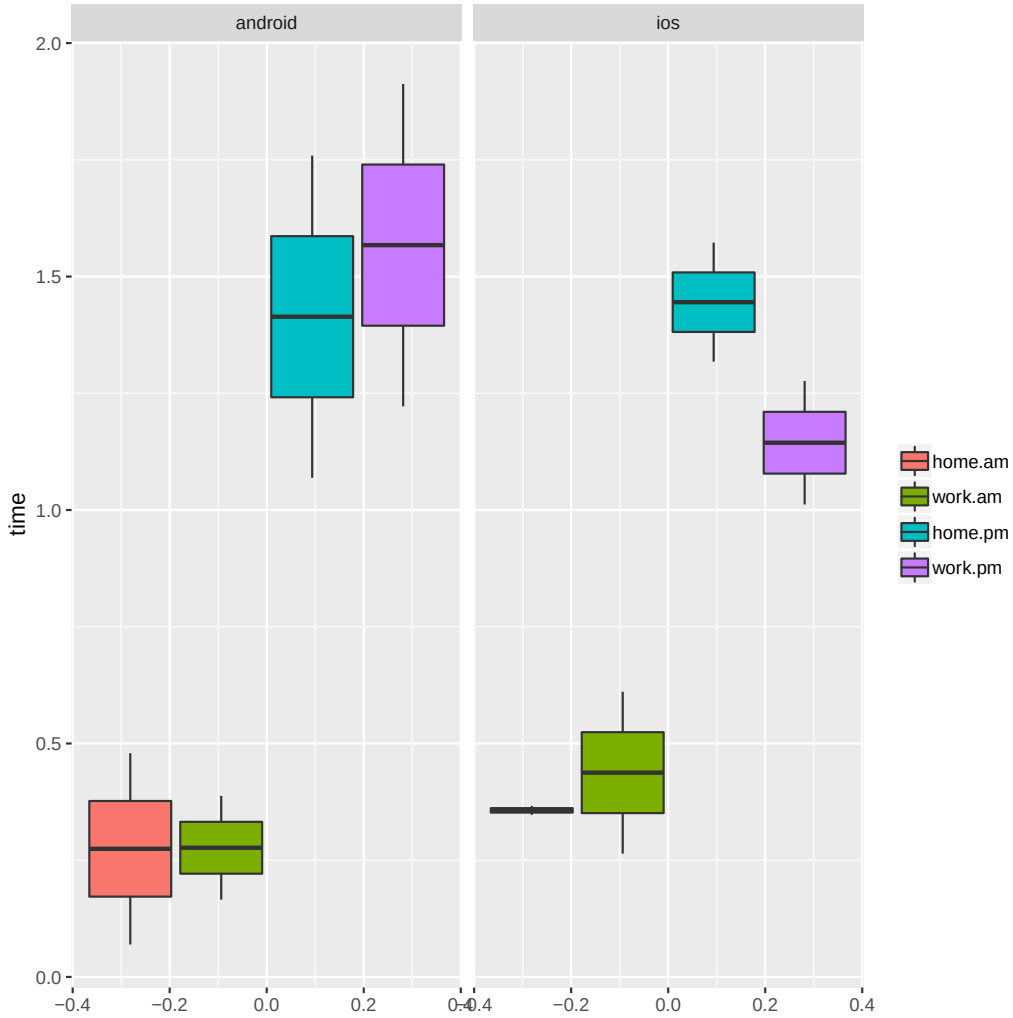
```
> myTidierDataFrame <- separate(myTidyDataFrame, sample,
    into = c("location", "timeOfDay"), sep = "\\_")
```

```
> head(myTidierDataFrame)
  uniqueId treatment location timeOfDay      time
1        1   Android     work        am 0.1655928
2        2       iOS     work        am 0.2641590
3        3       iOS     work        am 0.6108628
4        4   Android     work        am 0.3877454
5        1   Android     home        am 0.4793030
6        2       iOS     home        am 0.3662669
```

Plotting the data

```
ggplot(mobile_time_tidier,
           aes(group = interaction(location, time_of_day),
                  y = time,
                  fill = interaction(location, time_of_day))) +
  geom_boxplot() +
  facet_wrap(~treatment) +
  theme(legend.title = element_blank())
```

The Resulting Plot



Review Activity

We have a tibble named `rest_profit` that looks like this:

name	`1999`	`2000`
<chr>	<int>	<int>
Papalote	745	2666
Nopa	737	8488
Jannah	2258	2766

Put the parameters in the blanks to gather the 1999 and 2000 columns into a key column named "year" and a value column named "profit"

```
gather(____,key = ____, value = ____, ____, ____)
```

Parameters:

profit	`1999`	year
rest_profit	`2000`	

The `spread()` function

- The opposite of the `gather()` function is the `spread()` function, which takes *long* data and makes it *wide*
- Recall the example from a few slides ago, where *wide* data was made *long*

```
> head(tidy_df <- gather(my_df, light_sign, sales,
    fancy, normal), 4)
```

```
       City lightSign  Sales
1    Austin      Fancy  35000
2   Georgia      Fancy  43000
3 Vancouver      Fancy 106000
4    Austin     Normal  30000
```

The `spread()` function

- Undo what was done to the data and return it to it's original form using `spread()`

```
> (spread(tidy_df, light_sign, sales))
       City  Fancy Normal
1    Austin  35000  30000
2   Georgia  43000  44000
3 Vancouver 106000  77000
```

The `unite()` function

- Undo what was done with `separate()`
- `unite(mobile_time_tidier, col = sample, location, time_of_day)`

Practice with `spread()` and `unite()`

- Look at the dataset built into tidyr named `table3`
- `unite` the columns country and year to make a new column country_year by filling in the blanks:

  ```
  unite(table3, col = _____, _____, _____)
  ```

- `spread` the data so that there is a column for the rate in 1999 and column for the rate in 2000 by filling in the blanks:

  ```
  separate(table3, _____, _____)
  ```