Tidy Data Exploration

Tai Chou-Kudu

Overview

Part 1: Airplane flight delays

Consider the following dataset:

		Los_Angeles	Phoenix	San_Diego	San_Francisco	Seattle
ALASKA	On_Time	497	221	212	503	1841
	Delayed	62	12	20	102	305
AM WEST	On_Time	694	4840	383	320	301
	Delayed	117	415	65	129	61

The above table describes arrival delays for two different airlines across several destinations. The numbers correspond the the number of flights that were in either the delayed category or the on time category.

Problem 1: Read the information from `flightdelays.csv` into R, and use `tidyr` and `dplyr` to convert this data into a tidy/tall format with names and complete data for all columns. Your final data frame should have `City`, `On_Time_Flights` and `Delayed_Flights` as columns (the exact names are up to you). In addition to `pivot_longer`, `pivot_wider` and `rename`, you might find the `tidyr` function `fill` helpful for completing this task efficiently. Although this is a small dataset that you could easily reshape by hand, you should solve this problem using tidyverse functions that do the work for you.

```
library(tidyr)
library(dplyr)
library(ggplot2)
library(readr)
library(stringr)
library(here)
```

Read Flight Delays Data

Clicked "raw" data on github to retrieve URL for dataset, not just URL for gitHub HTML page. Reading data and inspecting data's column info below.

```
flightdelays <- readr::read_csv("https://raw.githubusercontent.com/georgehagstrom/DATA607/ma

spec(flightdelays)
```

```
cols(
  ...1 = col_character(),
  ...2 = col_character(),
  Los_Angeles = col_double(),
  Phoenix = col_double(),
  San_Diego = col_double(),
  San_Francisco = col_double(),
  Seattle = col_double()
)
```

Tidy Flight Delays Dataset

Renaming columns to give Airlines a column name, filling in missing airline data, performing a tall pivot.

```
tall_flightdelays <- flightdelays %>%
  rename(Airline = ...1,
         Flight_Status = ...2) %>%
  fill(Airline, .direction = "down") %>%
  pivot_longer(
    cols = Los_Angeles:Seattle,
    names_to = "City",
    values_to = "Flights"
  )

tall_flightdelays <- tall_flightdelays %>%
  mutate(Airline = Airline %>%
           str_to_lower() %>%
           str_to_title() %>%
           str_replace_all(" ","_")

  )
```

```r
final_flightdelays <- tall_flightdelays %>%
  pivot_wider(
    names_from = Flight_Status,
    values_from = Flights
  ) %>%
  rename(
    On_Time_Flights = On_Time,
    Delayed_Flights = Delayed
  ) %>%
  arrange(City) %>%
  relocate(City, .before = Airline)

print(final_flightdelays)
```

```
# A tibble: 10 x 4
   City          Airline On_Time_Flights Delayed_Flights
   <chr>         <chr>             <dbl>           <dbl>
 1 Los_Angeles   Alaska              497              62
 2 Los_Angeles   Am_West             694             117
 3 Phoenix       Alaska              221              12
 4 Phoenix       Am_West            4840             415
 5 San_Diego     Alaska              212              20
 6 San_Diego     Am_West             383              65
 7 San_Francisco Alaska              503             102
 8 San_Francisco Am_West             320             129
 9 Seattle       Alaska             1841             305
10 Seattle       Am_West             301              61
```

Problem 2: Take the data-frame that you tidied and cleaned in Problem 1 and create additional columns which contain the fraction of on-time and delayed flights at each airport. Then create a Cleveland Multiway Dot Plot (see this tutorial page for a description for how) to visualize the difference in flight delays between the two airlines at each city in the dataset. Compare the airlines and airports using the dot-plot- what are your conclusions?

Add Columns and Visualize Flight Delays

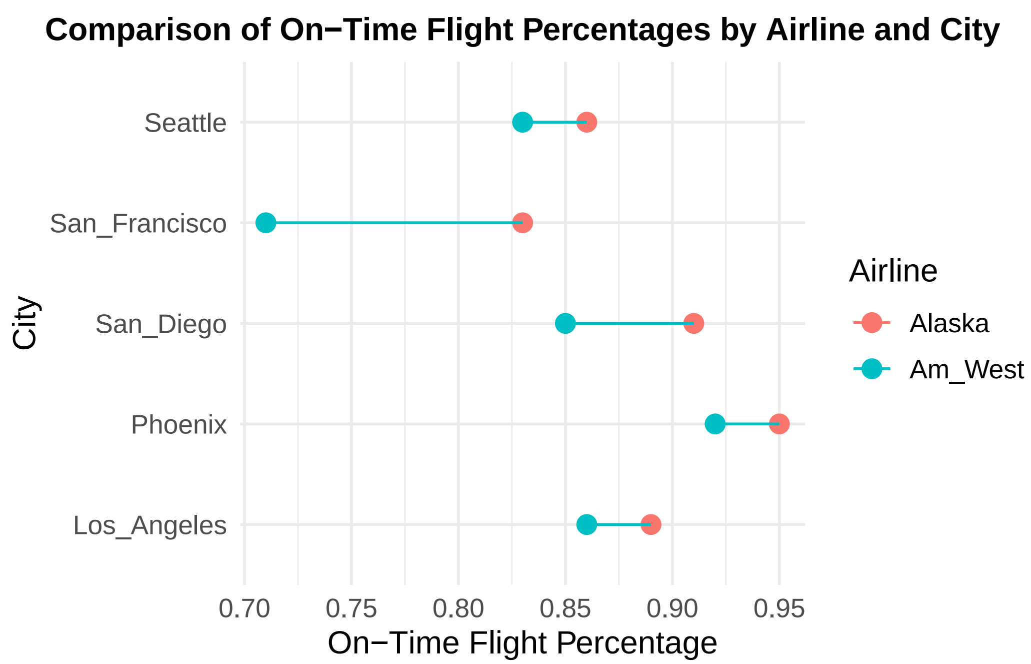
Add calculated percentage columns, visualize flight delays with a Cleveland Multiway Dot Plot.

```r
final_flightdelays <- final_flightdelays %>%
  group_by(City) %>%
  mutate(
    Total_Flights = On_Time_Flights + Delayed_Flights,
    On_Time_Pct= round(On_Time_Flights / Total_Flights, 2),
    Delayed_Pct= round(Delayed_Flights / Total_Flights, 2)
  ) %>%
  ungroup()

final_flightdelays
```

```
# A tibble: 10 x 7
   City        Airline On_Time_Flights Delayed_Flights Total_Flights On_Time_Pct
   <chr>       <chr>             <dbl>           <dbl>         <dbl>       <dbl>
 1 Los_Angeles Alaska              497              62           559        0.89
 2 Los_Angeles Am_West             694             117           811        0.86
 3 Phoenix     Alaska              221              12           233        0.95
 4 Phoenix     Am_West            4840             415          5255        0.92
 5 San_Diego   Alaska              212              20           232        0.91
 6 San_Diego   Am_West             383              65           448        0.85
 7 San_Franci~ Alaska              503             102           605        0.83
 8 San_Franci~ Am_West             320             129           449        0.71
 9 Seattle     Alaska             1841             305          2146        0.86
10 Seattle     Am_West             301              61           362        0.83
# i 1 more variable: Delayed_Pct <dbl>
```

```r
final_flightdelays <- final_flightdelays %>%
  mutate(City = factor(City, levels = unique(City)),
         Airline = factor(Airline, levels = unique(Airline)))

# Create the Cleveland Multiway Dot Plot
ggplot(final_flightdelays, aes(x = On_Time_Pct, y = City, color = Airline)) +
  geom_point(size = 3) +
  geom_line(aes(group = City), linetype = "solid") +
  labs(
    title = "Comparison of On-Time Flight Percentages by Airline and City",
    x = "On-Time Flight Percentage",
    y = "City",
    color = "Airline"
  ) +
  theme_minimal() +
```

```r
  theme(
    axis.title = element_text(size = 12, margin = margin(t = 30)),
    axis.text = element_text(size = 10),
    legend.title = element_text(size = 12),
    legend.text = element_text(size = 10),
    plot.title = element_text(size = 12, face = "bold",  hjust = 0.5)
  )
```



Optional: If you want to make a fancier visualization consider adding text labels containing the airline names above the dots using `geom_text` and `position = position_nudge(...)` with appropriate arguments.

Part 2: Mixed Drink Recipes

In the second part of this assignment we will be working with a dataset containing ingredients for different types of mixed drinks. This dataset is untidy and messy- it is in a wide data format and contains some inconsistencies that should be fixed.

Problem 3: Load the mixed drink recipe dataset into R from the file `MixedDrinkRecipes-prep.csv`, which you can download from my github page by clicking here. The variables `ingredient1` through `ingredient6` list the ingredients of the cocktail listed in the `name` column. Notice that there are many `NA` values in the ingredient columns, indicating that most cocktails have under 6 ingredients.

Tidy this dataset using `pivot_longer` to create a new data frame where each there is a row corresponding to each ingredient of all the cocktails, and an additional variable specifying the "rank" of that cocktail in the original recipe, i.e. it should look like this:

name	category	Ingredient_Rank	Ingredient
Gauguin	Cocktail Classics	1	Light Rum
Gauguin	Cocktail Classics	2	Passion Fruit Syrup
Gauguin	Cocktail Classics	3	Lemon Juice
Gauguin	Cocktail Classics	4	Lime Juice
Fort Lauderdale	Cocktail Classics	1	Light Rum

where the data-type of `Ingredient_Rank` is an integer. Hint: Use the `parse_number()` function in mutate after your initial pivot.

Read Mixed Drinks Data

Read data using readr package, use here::here to specify file local to the project root instead of copying a full pathname/ device-specific local path.

```
mixed_drinks <- readr::read_csv(here::here("MixedDrinkRecipes-Prep.csv"))
```

Tidy Mixed Drinks Dataset

Pivot longer to display tidy dataset where each ingredient of all the drinks has a row, including the rank in the original recipe and dataset.

```
tidy_drinks <- mixed_drinks %>%
  pivot_longer(
    cols = ingredient1:ingredient6,
    names_to = "Ingredient_Rank",
    values_to = "Ingredient",
    names_prefix = "ingredient") %>%
           mutate(Ingredient_Rank =
                    parse_number(Ingredient_Rank)) %>%
                      filter(!is.na(Ingredient)
    )

print(tidy_drinks)
```

```
# A tibble: 3,934 x 4
   name            category                   Ingredient_Rank Ingredient
   <chr>           <chr>                                <dbl> <chr>
 1 Gauguin         Cocktail Classics                        1 Light Rum
 2 Gauguin         Cocktail Classics                        2 Passion Fruit Syrup
 3 Gauguin         Cocktail Classics                        3 Lemon Juice
 4 Gauguin         Cocktail Classics                        4 Lime Juice
 5 Fort Lauderdale Cocktail Classics                        1 Light Rum
 6 Fort Lauderdale Cocktail Classics                        2 Sweet Vermouth
 7 Fort Lauderdale Cocktail Classics                        3 Juice of Orange
 8 Fort Lauderdale Cocktail Classics                        4 Juice of a Lime
 9 Apple Pie       Cordials and Liqueurs                    1 Apple schnapps
10 Apple Pie       Cordials and Liqueurs                    2 Cinnamon schnapps
# i 3,924 more rows
```

Problem 4: Some of the ingredients in the ingredient list have different names, but are nearly the same thing. An example of such a pair is `Lemon Juice` and `Juice of a lemon`, which are considered different ingredients in this dataset, but which perhaps should be treated as the same depending on the analysis you are doing. Make a list of the ingredients appearing in the ingredient list ranked by how commonly they occur along with the number of occurrences, and print the first 10 elements of the list here. Then check more ingredients (I suggest looking at more ingredients and even sorting them alphabetically using `arrange(asc(ingredient))`) and see if you can spot pairs of ingredients that are similar but have different names. Use `if_else`(click here for if_else) or `case_when` in combination with `mutate` to make it so that the pairs of ingredients you found have the same name. You don't have to find all pairs, but find at least 5 pairs of ingredients to rename. Because the purpose of this renaming is to facilitate a hypothetical future analysis, you can choose your own criteria for similarity as long as it is somewhat justifiable.

Notice that there are some ingredients that appear to be two or more ingredients strung together with commas. These would be candidates for more cleaning though this exercise doesn't ask you to fix them.

Standardize Data and Count Top 10 Ingredients

Use Case When to standardize at least 5 pairs of ingredients, sort by the top 10 from the standardized dataset.

```
ingredient_counts <- tidy_drinks %>%
  mutate(
    Ingredient = case_when(
      Ingredient == "Fresh lemon juice" ~ "Lemon Juice",
```

```
      Ingredient == "Juice of a Lime" ~ "Fresh Lime Juice",
      Ingredient == "Juice of Orange" ~ "Fresh orange juice",
      Ingredient == "ginger ale" ~ "Ginger ale",
      Ingredient == "Juice of a Lemon" ~ "Lemon Juice",
      TRUE ~ Ingredient
    )
  ) %>%
    count(Ingredient, sort = TRUE)


print(head(ingredient_counts, 10))
```

```
# A tibble: 10 x 2
   Ingredient           n
   <chr>            <int>
 1 Lemon Juice        268
 2 Gin                176
 3 Fresh Lime Juice   142
 4 Simple Syrup       115
 5 Light Rum          114
 6 Vodka              114
 7 Dry Vermouth       107
 8 Triple Sec         107
 9 Powdered Sugar      90
10 Grenadine           85
```

Problem 5: Some operations are easier to do on `wide` data rather than `tall` data. Find the 10 most common pairs of ingredients occurring in the top 2 ingredients in a recipe. It is much easier to do this with a `wide` dataset, so use `pivot_wider` to change the data so that each row contains all of the ingredients of a single cocktail, just like in the format of the original data-set. Then use `count` on the `1` and `2` columns to determine the most common pairs (see chapter 3 for a refresher on `count`).

Note: You may be interested to read about the `widyr` package here: widyr page. It is designed to solve problems like this one and uses internal pivot steps to accomplish it so that the final result is tidy. I'm actually unaware of any easy ways of solving problem 5 without pivoting to a wide dataset.

Count Top 10 Ingredient Pairs

Pivot wider so that we can see each drink and their 1 and 2 ranked ingredients. Count and sort to see the top 10 ingredient pairs.

```r
wide_drinks <- tidy_drinks %>%
  filter(Ingredient_Rank <= 2) %>%
  pivot_wider(
    names_from = Ingredient_Rank,
    values_from = Ingredient,
    names_prefix = "Ingredient_"
  )

ingredient_pairs <- wide_drinks %>%
  count(Ingredient_1, Ingredient_2, sort = TRUE)

print(head(ingredient_pairs, 10))
```

```
# A tibble: 10 x 3
   Ingredient_1     Ingredient_2          n
   <chr>            <chr>             <int>
 1 Gin              Dry Vermouth         23
 2 Juice of a Lemon Powdered Sugar       23
 3 Whole Egg        Powdered Sugar       13
 4 Light Rum        Fresh Lime Juice     12
 5 Gin              Triple Sec            9
 6 Bourbon whiskey  Fresh lemon juice     8
 7 Brandy           Sweet Vermouth        7
 8 Gin              Sweet Vermouth        7
 9 Light Rum        Pineapple Juice       7
10 Light Rum        Sweet Vermouth        7
```