Lab 2: Data Tidying

Overview

In this assignment you will work to tidy, clean, and analyze two different datasets, the first is a small dataset contained in a csv file called flightdelays.csv, and the second called MixedDrinkRecipes-Prep.csv.

The most important book chapters which cover the techniques you will practice here are R4DS Chapters 5 and 7. Also helpful are the `tidyr` vignette on pivoting and the ggplot help page on the geom_dotplot.

Submit your completed assignment on the course brightspace page by uploading your `.qmd` file and a compiled `pdf` or link to a compiled `html`, which you could host on your `github` or `rpubs` page as you wish.

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.

Problems

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.

Step 1: Load packages

```r
library(tidyverse)
library(arrow)
library(scales)
library(widyr)
```

Step 2: Create a new data frame using data hosted on github

```r
#Read raw data

flight_delays <- read_csv("https://raw.githubusercontent.com/georgehagstrom/DATA607/main/webs

print(flight_delays)
```

```
# A tibble: 4 x 7
  ...1    ...2    Los_Angeles Phoenix San_Diego San_Francisco Seattle
  <chr>   <chr>         <dbl>   <dbl>     <dbl>         <dbl>   <dbl>
1 ALASKA  On_Time         497     221       212           503    1841
2 <NA>    Delayed          62      12        20           102     305
3 AM WEST On_Time         694    4840       383           320     301
4 <NA>    Delayed         117     415        65           129      61
```

Step 3: Tidy data

```r
flight_delays <- flight_delays |>
  fill(1, .direction = "down") |>
  rename(airline = 1, status = 2) |>
  pivot_longer(
    cols = !c("airline", "status"),
    names_to = "city",
    values_to = "count") |>
  print(n=10)
```

```
# A tibble: 20 x 4
   airline status  city          count
   <chr>   <chr>   <chr>         <dbl>
 1 ALASKA  On_Time Los_Angeles     497
 2 ALASKA  On_Time Phoenix         221
 3 ALASKA  On_Time San_Diego       212
 4 ALASKA  On_Time San_Francisco   503
 5 ALASKA  On_Time Seattle        1841
 6 ALASKA  Delayed Los_Angeles      62
 7 ALASKA  Delayed Phoenix          12
 8 ALASKA  Delayed San_Diego        20
 9 ALASKA  Delayed San_Francisco   102
10 ALASKA  Delayed Seattle         305
# i 10 more rows
```

```
#Looks tidy but let's add those "on-time" and "delayed" columns with
#`pivot_wider()`

flight_delays <- flight_delays |>
  pivot_wider(names_from = status, values_from = count) |>
  rename(on_time_flights = On_Time, delayed_flights = Delayed) |>
print()
```

```
# A tibble: 10 x 4
   airline city          on_time_flights delayed_flights
   <chr>   <chr>                   <dbl>           <dbl>
 1 ALASKA  Los_Angeles               497              62
 2 ALASKA  Phoenix                   221              12
 3 ALASKA  San_Diego                 212              20
 4 ALASKA  San_Francisco             503             102
 5 ALASKA  Seattle                  1841             305
 6 AM WEST Los_Angeles               694             117
 7 AM WEST Phoenix                  4840             415
 8 AM WEST San_Diego                 383              65
 9 AM WEST San_Francisco             320             129
10 AM WEST Seattle                   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?

Step 1: Create a data frame for our plot, including the calculated values we'd like to compare.

```
flight_delays <- flight_delays |>
  mutate(
    total_flights = on_time_flights + delayed_flights,
    percent_on_time = on_time_flights / total_flights,
    percent_delayed = delayed_flights / total_flights)


#Make cities a factor, ordered by average percent_on_time, so they will appear
#in ascending order on our plot
city_order <- flight_delays |>
              group_by(city) |>
              summarise(on_time_flights = sum(on_time_flights),
                        total_flights = sum(total_flights)) |>
              mutate(percent_on_time = on_time_flights / total_flights) |>
              arrange(percent_on_time) |>
              mutate(city = factor(city, levels = city))


#Copy the factor levels for city from 'city_order' to our plotting df
flight_delays <- flight_delays |>
  mutate(city = factor(city, levels = city_order$city))
```

Step 2: Slice up our plotting df to create labels and highlight the biggest differences.

```
#Create regular labels
right_label <- flight_delays |>
  group_by(city) |>
  slice_max(percent_on_time)

left_label <- flight_delays |>
  group_by(city) |>
  slice_min(percent_on_time)


#Find differences in percent_on_time >5% between airlines. Create new df to
#use for highlight labels
big_diff <- flight_delays |>
```

```r
  pivot_wider(names_from = airline, values_from = percent_on_time) |>
  rename(AM_WEST = "AM WEST") |>
  group_by(city) |>
  summarise(AM_WEST = sum(AM_WEST, na.rm = TRUE),
            ALASKA = sum(ALASKA, na.rm = TRUE)) |>
  mutate(max = pmax(ALASKA, AM_WEST),
         min = pmin(ALASKA, AM_WEST),
         diff = max / min -1) |>
  filter(diff > .05)


#Create highlight components for our plot utilizing 'big_diff'
highlight <- flight_delays |>
  filter(city %in% big_diff$city)

right_highlight_label <- flight_delays |>
  group_by(city) |>
  slice_max(percent_on_time) |>
  filter(city %in% big_diff$city)

left_highlight_label <- flight_delays |>
  group_by(city) |>
  slice_min(percent_on_time) |>
  filter(city %in% big_diff$city)
```

3. Create plot. The Alaska airline has had a higher percentage of flights on-time than Am West for all 5 cities. There were only 2 cities where the Alaska airline had more than 5% greater on-time flights than Am West, highlighting how small the margins were between the two airlines.

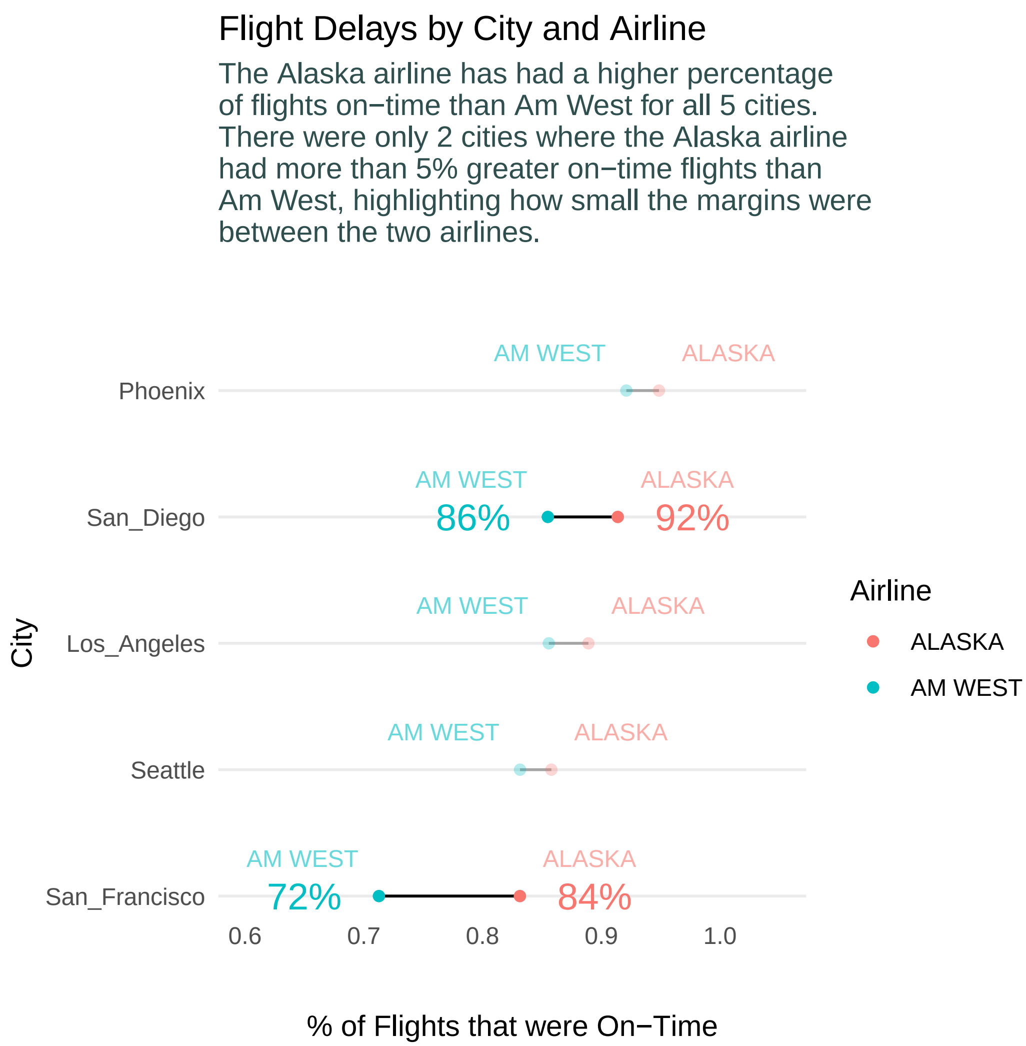
```r
flight_delays |>

#adds base data
  ggplot(aes(percent_on_time, city, label = percent(percent_on_time, 2))) +
  geom_line(aes(group = city), alpha = .3) +
  geom_point(aes(color = airline), alpha = .3) +
  geom_text(
            data = left_label,
            aes(color = airline,
                label = airline),
            position = position_nudge (x = .03, y = .3),
            show.legend = F,
```

```
            hjust = 1.5, alpha = .6,
            size = 3) +
  geom_text(
            data = right_label,
            aes(color = airline,
                label = airline),
            position = position_nudge(x = -.02, y =.3),
            show.legend = F,
            hjust = -.5,
            alpha = .6,
            size = 3) +

#Adds highlighted data aesthetics
  geom_line(data = highlight, aes(group = city)) +
  geom_point(data = highlight, aes(color = airline)) +
  geom_text(
            data = left_highlight_label,
            aes(color = airline),
            show.legend = F,
            hjust = 1.5,
            size = 5) +
  geom_text(
            data = right_highlight_label,
            aes(color = airline),
            show.legend = F,
            hjust = -.5,
            size = 5) +
  scale_x_continuous(limits = c(.6, 1.05)) +

#Adjust theme and labels
  theme_minimal() +
  theme(
    panel.grid.major.x = element_blank(),
    panel.grid.minor = element_blank(),
    plot.subtitle = element_text(color = "darkslategrey", margin = margin(b = 25)),
    axis.text.x = element_text(vjust = 12)) +
  labs(color = "Airline",
       x = "% of Flights that were On-Time",
       y = "City",
       title = "Flight Delays by City and Airline",
       subtitle = str_wrap("The Alaska airline has had a higher percentage of flights on-time
```



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.

Problems

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.

Step 1: Read the data into df

```
mixed_drink_recipes_prep <- read_csv("https://raw.githubusercontent.com/georgehagstrom/DATA6

print(n=5)
```

```
Rows: 990 Columns: 8
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (8): name, category, ingredient1, ingredient2, ingredient3, ingredient4,...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# A tibble: 990 x 8
  name      category ingredient1 ingredient2 ingredient3 ingredient4 ingredient5
  <chr>     <chr>    <chr>       <chr>       <chr>       <chr>       <chr>
1 Gauguin   Cocktai~ Light Rum   Passion Fr~ Lemon Juice Lime Juice  <NA>
2 Fort Lau~ Cocktai~ Light Rum   Sweet Verm~ Juice of O~ Juice of a~ <NA>
```

```
3 Apple Pie Cordial~ Apple schn~ Cinnamon s~ Apple slice <NA>        <NA>
4 Cuban Co~ Cocktai~ Juice of a~ Powdered S~ Light Rum   <NA>        <NA>
5 Cool Car~ Cocktai~ Dark rum    Cranberry ~ Pineapple ~ Orange cur~ Sour Mix
# i 985 more rows
# i 1 more variable: ingredient6 <chr>
```

Step 2: Pivot_longer

```
drink_recipes_clean <- mixed_drink_recipes_prep |>
  pivot_longer(
              cols = contains("ingredient"),
              names_to = "ingredient_rank",
              values_to = "ingredient",
              values_drop_na = TRUE) |>
  mutate(ingredient_rank = parse_number(ingredient_rank),
         ingredient = tolower(ingredient)) |>

print(n=5)
```

```
# 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
# i 3,929 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.

Step 1: Sort ingredients by how commonly they appear and by alphabetically. After examining, I decided that I would go replace some brand name spirits by their generic name, because I just want to know what spirits I need for a drink, and I'll buy my own favorite brand.

```
drink_ingredients_count_order <- drink_recipes_clean |>
  count(ingredient) |>
  arrange(desc(n)) |>
  print(n=10)
```

```
# A tibble: 652 x 2
   ingredient             n
   <chr>              <int>
 1 gin                  176
 2 fresh lemon juice    138
 3 simple syrup         115
 4 light rum            114
 5 vodka                114
 6 dry vermouth         107
 7 fresh lime juice     107
 8 triple sec           107
 9 powdered sugar        92
10 grenadine             85
# i 642 more rows
```

```
drink_ingredients_alphabetical_order <- drink_ingredients_count_order |>
  arrange(ingredient)
```

Step 2: I filtered the count ordered dataframe to only show rows where the ingredient contains rum, gin, or vodka. Then, I replacement some of the most common spirits with their generic names.

```
top_spirits <- drink_ingredients_count_order |>
  filter(grepl("gin|rum|vodka", ingredient, ignore.case = TRUE))


#Create a new column with the generic spirit names for 5 ingredients
drink_recipes_clean <- drink_recipes_clean |>
  mutate(generic_spirit =
          case_when(
                    ingredient == "old tom gin" ~ "gin",
                    ingredient == "old mr. boston dry gin" ~ "gin",
```

```
                         ingredient == "sloe gin" ~ "gin",
                         ingredient == "old mr. boston vodka" ~ "vodka",
                         ingredient == "mr. boston gin" ~ "gin",
                         .default = ingredient))
```

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.

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`).

1. This method shows us the most common ingredient pairings in order of Ingredient1 and Ingredient2. This would be most helpful if Ingredient_n represented the step that the ingredient gets added to the cup.

```
drink_recipes_clean |>
  pivot_wider(
              id_cols = name,
              names_from = ingredient_rank,
              values_from = ingredient) |>
  rename("ingredient" = 2:7) |>
  count(ingredient1, ingredient2, sort = TRUE)
```

```
# A tibble: 699 x 3
   ingredient1      ingredient2           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
# i 689 more rows
```

2. If we want to see the most common ingredient pairings overall within Ingredient1 and Ingredient2, we can use the `pairwise_count` argument. This shows us the most common ingredient pairings in any particular order. Comparing to the last dataset, we can see the gin and dry vermouth have an additional 5 pairings where dry vermouth must be the ingredient1.

```
drink_recipes_clean |>
  filter(ingredient_rank %in% c(1,2)) |>
  pairwise_count(ingredient, name, upper = FALSE) |>
  arrange(desc(n))
```

```
# A tibble: 662 x 3
   item1           item2                n
   <chr>           <chr>            <dbl>
 1 gin             dry vermouth        28
 2 powdered sugar  juice of a lemon    25
 3 powdered sugar  whole egg           14
 4 light rum       fresh lime juice    12
 5 sweet vermouth  gin                  9
 6 sweet vermouth  dry vermouth         9
 7 gin             triple sec           9
 8 bourbon whiskey fresh lemon juice    8
 9 light rum       sweet vermouth       7
10 light rum       brandy               7
# i 652 more rows
```

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.