Practice Prelim 1

AEM 2850 / AEM 5850

Preface

The goal of this prelim is to assess your facility with key data wrangling tasks we covered in the first five weeks of the course.

We will work with multiple data sets, most of which are already loaded and all of which are available in the working directory of the project (see the Files pane on the lower right).

Instructions

- You must complete Prelim 1 **in person** in Warren 150 during class
- Prelim 1 is open internet, but **do not communicate with classmates**
- Do not use packages outside the `tidyverse` packages we have already loaded for you (penalties may apply)
- When done, **upload BOTH your .qmd and .pdf files** to canvas

Additional notes

- The prelim is 100 points total, and each question states the number of points it is worth (*this is not true on the practice prelim but will be true on the actual prelim*)
- **Render early and often** to avoid wasting time sorting out what code needs debugging
- We will give partial credit if your answers are incomplete, especially if you provide comments or text that describes the logic of what you *would* do if you had more time
- If you have trouble rendering your document, do not delete your work in progress code. That will make it hard for us to give you partial credit. Instead, you can:
  - Comment out problematic code using `#` or keyboard shortcut Cntrl/Cmd-Shift-C
  - Replace `{r}` with `{r, eval = FALSE}` at the top of the relevant code chunk
  - Ask questions!

1. The data frame `tidy_us_registrations` contains information on vehicle registrations in the U.S. Use it to determine what make - model combination has the most registrations in the data.

```r
tidy_us_registrations |>
  group_by(make, model) |>
  summarize(total_count = sum(count, na.rm = TRUE)) |>
  ungroup() |>
  arrange(desc(total_count)) |>
  slice_head(n = 1)
```

```
`summarise()` has grouped output by 'make'. You can override using the
`.groups` argument.
```

```
# A tibble: 1 x 3
  make  model  total_count
  <chr> <chr>        <dbl>
1 HONDA ACCORD     6259479
```

```r
  #slice_max(total_count)
```

The make and model with the most registrations is...

2. Again using `tidy_us_registrations`, if we consider all different versions of the Ford F-150 to be a single model, rather than different models, would the answer to the previous question change? Why or why not?

```
tidy_us_registrations |>
  mutate(
    is_f150 = str_detect(model, "[Ff]-*150"),
    model2 = if_else(is_f150 == TRUE, "F150", model)
  ) |>
  group_by(make, model2) |>
  summarize(total_count = sum(count, na.rm = TRUE)) |>
  ungroup() |>
  arrange(desc(total_count))
```

```
`summarise()` has grouped output by 'make'. You can override using the
`.groups` argument.
```

```
# A tibble: 2,061 x 3
   make      model2   total_count
   <chr>     <chr>          <dbl>
 1 FORD      F150         6660600
 2 HONDA     ACCORD       6259479
 3 TOYOTA    CAMRY        5747726
 4 FORD      TAURUS       4871827
 5 HONDA     CIVIC        4183319
 6 TOYOTA    COROLLA      3721676
 7 CHEVROLET CAVALIER     3365436
 8 PONTIAC   GRAND AM     2589450
 9 FORD      ESCORT       2545994
10 BUICK     LESABRE      2327713
# i 2,051 more rows
```

…

3. Use `case_when()` to make a table of the share of registered vehicles that fall into the following age groups: `0-2 years old`, `3-5 years old`, `6-10 years old`, `11-15 years old`, and `16+ years old`. Use those age ranges in the table so the results are clear to the reader, and put them in order from youngest to oldest. Present the numbers as percentage points, so that they sum to 100.

```
tidy_us_registrations |>
  mutate(
    age_group =
        case_when(
          age <= 2 ~ "0-2 years old",
          age <= 5 ~"3-5 years old",
          age <= 10 ~  "6-10 years old",
          age <= 15 ~  "11-15 years old",
          age >= 16 ~  "16+ years old"
        )
  ) |>
  group_by(age_group) |>
  summarise(total_count = sum(count, na.rm = TRUE)) |>
  ungroup() |>
  mutate(percent = total_count / sum(total_count) * 100)
```

```
# A tibble: 5 x 3
  age_group       total_count percent
  <chr>                 <dbl>   <dbl>
1 0-2 years old      27894211    12.0
2 11-15 years old    47868541    20.6
3 16+ years old      44833804    19.3
4 3-5 years old      45826828    19.8
5 6-10 years old     65563173    28.3
```

```
# solution with table ordering
tidy_us_registrations |>
  arrange(age) |>
  mutate(
    age_group_aux =
        case_when(
          age >= 16 ~ 5,
          age >= 11 ~ 4,
          age >= 6 ~ 3,
          age >= 3 ~ 2,
```

```r
        age >= 0 ~ 1
      ),

    age_groups =
      case_when(
        age_group_aux == 5 ~ "16+ years old",
        age_group_aux == 4 ~ "11-15 years old",
        age_group_aux == 3 ~ "6-10 years old",
        age_group_aux == 2~ "3-5 years old",
        age_group_aux == 1~ "0-2 years old"
      )
  ) |>
  group_by(age_group_aux, age_groups) |>
  summarise(total_count = sum(count, na.rm = TRUE)) |>
  ungroup() |>
  mutate(percent = total_count / sum(total_count) * 100) |>
  arrange(age_group_aux) |>
  select(!age_group_aux)
```

```
`summarise()` has grouped output by 'age_group_aux'. You can override using the
`.groups` argument.
```

```
# A tibble: 5 x 3
  age_groups      total_count percent
  <chr>                 <dbl>   <dbl>
1 0-2 years old      27894211    12.0
2 3-5 years old      45826828    19.8
3 6-10 years old     65563173    28.3
4 11-15 years old    47868541    20.6
5 16+ years old      44833804    19.3
```

Product reviews

4a. The data frame `tidy` contains product reviews from Amazon. How many different products are there, based on different values of `name`?

```
tidy |> distinct(name) |> nrow()
```

```
[1] 61
```

…

4b. What is the most expensive product in `tidy`?

```
tidy |>
  arrange(desc(price)) |>
  slice_head(n = 1)
```

```
# A tibble: 1 x 9
  brand  name           price reviews.date reviews.doRecommend reviews.numHelpful
  <chr>  <chr>          <dbl> <date>       <lgl>                            <dbl>
1 Amazon "Kindle Fire~    464 NA           NA                                 402
# i 3 more variables: reviews.rating <dbl>, reviews.text <chr>,
#   reviews.title <chr>
```

```
tidy |> slice_max(price) |> distinct(name)
```

```
# A tibble: 1 x 1
  name
  <chr>
1 "Kindle Fire HDX 8.9\""
```

…

5a. Let's use `tidy` to look at reviews and ratings. Which product has the largest number of reviews that others marked as helpful?

```
tidy |>
  mutate(
    is_helpful = if_else(!is.na(reviews.numHelpful) & reviews.numHelpful > 0, TRUE, FALSE)
  ) |>
  group_by(name) |>
    summarize(n_helpful = sum(is_helpful)) |>
    ungroup() |>
    slice_max(n_helpful)
```

```
# A tibble: 1 x 2
  name                                                    n_helpful
  <chr>                                                       <int>
1 Amazon Tap - Alexa-Enabled Portable Bluetooth Speaker          39
```

…

5b. Which product has the lowest average rating?

```
tidy |>
  filter(!is.na(reviews.rating)) |>
  group_by(name) |>
  summarize(avg_rating = mean(reviews.rating)) |>
  ungroup() |>
  slice_min(avg_rating)
```

```
# A tibble: 1 x 2
  name                                            avg_rating
  <chr>                                                <dbl>
1 Alexa Voice Remote for Amazon Fire TV and Fire TV Stick       2.08
```

```
tidy |>
  group_by(name) |>
  summarize(avg_rating = mean(reviews.rating, na.rm = TRUE)) |>
  ungroup() |>
  slice_min(avg_rating)
```

```
# A tibble: 1 x 2
  name                                            avg_rating
  <chr>                                                <dbl>
1 Alexa Voice Remote for Amazon Fire TV and Fire TV Stick       2.08
```

```
tidy |>
  group_by(name) |>
  mutate(avg_rating = mean(reviews.rating, na.rm = TRUE)) |>
  distinct(name, avg_rating) |>
  ungroup() |>
  slice_min(avg_rating)
```

```
# A tibble: 1 x 2
  name                                            avg_rating
  <chr>                                                <dbl>
1 Alexa Voice Remote for Amazon Fire TV and Fire TV Stick       2.08
```

…

6. ***Logic, strings, and regular expressions:*** **Let's zoom into products with "Kindle" in the name, but excluding cover and charger accessories. Use `tidy` to generate a logical variable/column in the data that indicates whether each product is a `tablet`, based on whether "Fire" is in a product's name. For each of the two categories – tablet or not – compute the number of products and average rating. Present the information in a table with two rows and three columns. Did the tablets receive higher or lower ratings than non-tablets (i.e., e-readers)?**

```
tidy |>
  mutate(
    is_tablet = str_detect(name, regex("Kindle Fire", ignore_case = TRUE))
  ) |>
  group_by(is_tablet) |>
  summarize(
    n_product = n_distinct(name),
    avg_rating = mean(reviews.rating, na.rm = TRUE)
  ) |>
  ungroup()
```

```
# A tibble: 2 x 3
  is_tablet n_product avg_rating
  <lgl>         <int>      <dbl>
1 FALSE            58       4.38
2 TRUE              3       4.12
```

…

Taxi cabs

7a. *Joins:* Now we'll use data on taxi trips in `trips` and taxi locations in `locations`. Join them based on the pick up location (`PULocation`) from `trips` and the location ID (`LocationID`) from `locations`, taking care to retain all rows from the trip data and no rows from the location data that do not match with the trips data. How many rows are in the joined data?

```r
trips |>
left_join(locations, by = join_by(PULocationID == LocationID))
```

```
# A tibble: 56,443 x 19
   pickup_datetime     dropoff_datetime    passenger_count trip_distance
   <dttm>              <dttm>                        <dbl>         <dbl>
 1 2021-01-28 00:05:10 2021-01-28 00:30:05               1         15.2
 2 2021-01-28 00:01:31 2021-01-28 00:07:04               2          1.4
 3 2021-01-28 00:22:05 2021-01-28 00:30:23               1          1.12
 4 2021-01-28 00:00:16 2021-01-28 00:14:51               1          3.3
 5 2021-01-28 00:24:26 2021-01-28 00:27:27               2          0.7
 6 2021-01-28 00:14:16 2021-01-28 00:18:35               1          0.7
 7 2021-01-28 00:23:51 2021-01-28 00:39:43               1          9.9
 8 2021-01-28 00:57:33 2021-01-28 01:13:33               5          3.61
 9 2021-01-28 00:49:08 2021-01-28 00:53:24               2          0.83
10 2021-01-28 00:17:51 2021-01-28 00:42:35               1          6.15
# i 56,433 more rows
# i 15 more variables: RatecodeID <dbl>, PULocationID <dbl>,
#   DOLocationID <dbl>, payment_type <dbl>, fare_amount <dbl>, extra <dbl>,
#   mta_tax <dbl>, tip_amount <dbl>, tolls_amount <dbl>,
#   improvement_surcharge <dbl>, total_amount <dbl>,
#   congestion_surcharge <dbl>, Borough <chr>, Zone <chr>, service_zone <chr>
```

…

7b. Are there any rows in `trips` with pick up locations that do not match a location in `locations`? If so, how many?

```
anti_join(trips, locations, by = join_by(PULocationID == LocationID))
```

```
# A tibble: 0 x 16
# i 16 variables: pickup_datetime <dttm>, dropoff_datetime <dttm>,
#   passenger_count <dbl>, trip_distance <dbl>, RatecodeID <dbl>,
#   PULocationID <dbl>, DOLocationID <dbl>, payment_type <dbl>,
#   fare_amount <dbl>, extra <dbl>, mta_tax <dbl>, tip_amount <dbl>,
#   tolls_amount <dbl>, improvement_surcharge <dbl>, total_amount <dbl>,
#   congestion_surcharge <dbl>
```

```
trips |>
anti_join(locations, by = join_by(PULocationID == LocationID))
```

```
# A tibble: 0 x 16
# i 16 variables: pickup_datetime <dttm>, dropoff_datetime <dttm>,
#   passenger_count <dbl>, trip_distance <dbl>, RatecodeID <dbl>,
#   PULocationID <dbl>, DOLocationID <dbl>, payment_type <dbl>,
#   fare_amount <dbl>, extra <dbl>, mta_tax <dbl>, tip_amount <dbl>,
#   tolls_amount <dbl>, improvement_surcharge <dbl>, total_amount <dbl>,
#   congestion_surcharge <dbl>
```

…

7c. Are there any rows in the `locations` data that do not match the pick up location of at least one taxi trip? If so, how many?

```
anti_join(locations, trips,  by = join_by(LocationID == PULocationID))
```

```
# A tibble: 30 x 4
   LocationID Borough       Zone                                service_zone
        <dbl> <chr>         <chr>                               <chr>
 1          1 EWR           Newark Airport                      EWR
 2          2 Queens        Jamaica Bay                         Boro Zone
 3          6 Staten Island Arrochar/Fort Wadsworth             Boro Zone
 4          8 Queens        Astoria Park                        Boro Zone
 5         30 Queens        Broad Channel                       Boro Zone
 6         54 Brooklyn      Columbia Street                     Boro Zone
 7         58 Bronx         Country Club                        Boro Zone
 8         59 Bronx         Crotona Park                        Boro Zone
 9         84 Staten Island Eltingville/Annadale/Prince's Bay Boro Zone
10         99 Staten Island Freshkills Park                     Boro Zone
# i 20 more rows
```

…

8. Based on the joined data from 7a, how many trips originated in each borough? Please present the results as a small data frame with the same number of rows as there are boroughs in NYC.

```
trips |>
  left_join(locations, by = join_by(PULocationID == LocationID)) |>
  group_by(Borough) |>
  count()
```

```
# A tibble: 6 x 2
# Groups:   Borough [6]
  Borough           n
  <chr>         <int>
1 Bronx           550
2 Brooklyn       1169
3 Manhattan     51918
4 Queens         2086
5 Staten Island     9
6 Unknown         711
```

9. Now join the data to find the borough where the passenger was picked up as well as the borough where the passenger was dropped off (`DOLocationID`). Make a table of the most common three borough-to-borough trip combinations that did *NOT* start and end in the same borough. Please include the number of trips for each combination in the table.

```r
trips |>
  left_join(locations, by = join_by(PULocationID == LocationID)) |>
  rename(pu_borough = Borough, pu_zone = Zone, pu_service_zone = service_zone ) |>
  left_join(locations, by = join_by(DOLocationID == LocationID)) |>
  rename(do_borough = Borough, do_zone = Zone, do_service_zone = service_zone ) |>
    group_by(pu_borough, do_borough) |>
    count() |>
  ungroup() |>
  filter(pu_borough != do_borough) |>
  rename(n_trips = n) |>
  slice_max(n_trips, n =3)
```

```
# A tibble: 3 x 3
  pu_borough do_borough n_trips
  <chr>      <chr>        <int>
1 Manhattan  Brooklyn      1254
2 Manhattan  Queens         917
3 Queens     Manhattan      585
```