Section 2: Data Wrangling and Visualization

Contents

1 Data Wrangling

1.1 Tidyverse Grammar

```
# Load the tidyverse package
library(tidyverse)
```

```
# this should display the homepage of the documentation for tidyverse
?tidyverse
```

```
## starting httpd help server ... done
```

```
# Set your working directory
setwd("C:/Users/shuoy/Dropbox/161/Sections/Section2")

# Example: Read egg price data.
df <- read.csv("DailyPrices.csv")
```

We can check first several rows by using `head`.

```
head(df)
```

```
##         Date   Egg.Class Market.Name Delivery.Name      Price.Unit Low.Price
## 1 2013-02-04       LARGE  CALIFORNIA       INVOICE CENTS PER DOZEN       171
## 2 2013-02-04       JUMBO  CALIFORNIA       INVOICE CENTS PER DOZEN       182
## 3 2013-02-04 EXTRA LARGE  CALIFORNIA       INVOICE CENTS PER DOZEN       175
## 4 2013-02-04       LARGE  CALIFORNIA       INVOICE CENTS PER DOZEN       171
## 5 2013-02-04      MEDIUM  CALIFORNIA       INVOICE CENTS PER DOZEN       139
## 6 2013-02-04      MEDIUM  CALIFORNIA       INVOICE CENTS PER DOZEN       139
##   High.Price Grade Mostly.Low Mostly.High
## 1        171  <NA>         NA          NA
## 2        182  <NA>         NA          NA
## 3        175  <NA>         NA          NA
## 4        171  <NA>         NA          NA
## 5        139  <NA>         NA          NA
## 6        139  <NA>         NA          NA
```

If you want to see all rows and columns, use `View()` or click the object in the environment pane.

```
View(df)
```

We can do a quick summary of data by using `summary`.

```
summary(df)
```

```
##      Date            Egg.Class         Market.Name        Delivery.Name
##  Length:42702       Length:42702       Length:42702       Length:42702
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##   Price.Unit          Low.Price       High.Price       Grade
##  Length:42702       Min.   :  0.0   Min.   :  5.0   Length:42702
##  Class :character   1st Qu.: 81.0   1st Qu.: 88.0   Class :character
##  Mode  :character   Median :128.0   Median :133.0   Mode  :character
##                     Mean   :154.7   Mean   :160.3
##                     3rd Qu.:193.0   3rd Qu.:196.0
##                     Max.   :899.0   Max.   :910.0
##                     NA's   :12      NA's   :12
##    Mostly.Low     Mostly.High
##  Min.   :  3.0   Min.   : 24.0
##  1st Qu.: 73.0   1st Qu.: 81.0
##  Median :104.0   Median :111.0
##  Mean   :130.1   Mean   :138.9
##  3rd Qu.:157.0   3rd Qu.:165.0
##  Max.   :620.0   Max.   :612.0
##  NA's   :18726   NA's   :24722
```

Question: What are the observations/units in the data? What are the variables?

This dataframe is in "tidy" form. This means that **each row is an observation, each column is a variable, and each cell is a single value.**

The `tidyverse` package that we installed earlier provides useful functions for transforming tidy data. In this section we will cover commonly used functions from the `tidyverse` package. Each of these commands work in a similar way.

- the first input is a tidy dataframe
- they output a new dataframe
- they can be "chained" together with piping - this means that the output of a function becomes the input of the next function

1.2 dplyr: Selecting Columns with `select()`

The `select()` function is used to choose specific columns in a data frame. - Use the colon operator `:` to select a range of variables. - Use the exclamation mark `!` or `-` to exclude variables from a selection. - Use `&` for intersection (common variables between sets). - Use `|` for union (all variables across sets). - Use c() to combine multiple selections.

It supports helper functions for selecting columns based on patterns.Useful helpers include:

- `everything()`: Moves specified columns to the beginning, retaining others in order (e.g., select(df, a, everything())).
- `starts_with()` / `ends_with()`: Selects columns by prefix or suffix.
- `contains()`: Matches columns containing a substring.
- `matches()`: Uses regular expressions for flexible pattern matching.

Notes: - In R, functions working on data-frames are transitory: if you don't *save* the result, it just prints it. - You do not need to put quotation marks around the variable names. This is a convenient feature of all `tidyverse` functions.

```
selected_df <- df %>%
  select(Date, Low.Price, High.Price)
colnames(selected_df)
```

```
## [1] "Date"       "Low.Price"  "High.Price"
```

```
selected_df <- df %>%
  select(Date:High.Price)
colnames(selected_df)
```

```
## [1] "Date"          "Egg.Class"     "Market.Name"   "Delivery.Name"
## [5] "Price.Unit"    "Low.Price"     "High.Price"
```

```
selected_df <- df %>%
  select(!c(Low.Price, High.Price))
colnames(selected_df)
```

```
## [1] "Date"          "Egg.Class"     "Market.Name"   "Delivery.Name"
## [5] "Price.Unit"    "Grade"         "Mostly.Low"    "Mostly.High"
```

```
selected_df <- df %>%
  select(-Low.Price, -High.Price)
colnames(selected_df)
```

```
## [1] "Date"          "Egg.Class"     "Market.Name"   "Delivery.Name"
## [5] "Price.Unit"    "Grade"         "Mostly.Low"    "Mostly.High"
```

```
reordered_df <- df %>%
  select(Low.Price, everything())
colnames(reordered_df)
```

```
##  [1] "Low.Price"     "Date"          "Egg.Class"     "Market.Name"
##  [5] "Delivery.Name" "Price.Unit"    "High.Price"    "Grade"
##  [9] "Mostly.Low"    "Mostly.High"
```

```
selected_df <- df %>%
  select(starts_with("D"))
colnames(selected_df)
```

```
## [1] "Date"          "Delivery.Name"
```

```
selected_df <- df %>%
  select(ends_with("e"))
colnames(selected_df)
```

```
## [1] "Date"          "Market.Name"   "Delivery.Name" "Low.Price"
## [5] "High.Price"    "Grade"
```

```
selected_df <- df %>%
  select(contains("Price"))
colnames(selected_df)
```

```
## [1] "Price.Unit" "Low.Price"  "High.Price"
```

```
selected_df <- df %>%
  select(matches("^L.+e$")) # Regular expressions
colnames(selected_df)
```

```
## [1] "Low.Price"
```

1.3 Filtering Rows with `filter()`

The `filter()` function is used to select rows based on logical conditions.

```
# Filter rows where `Low.Price` equals 171
df_filtered <- df %>%
  filter(Low.Price == 171)
summary(df_filtered)
```

```
##      Date            Egg.Class         Market.Name        Delivery.Name
##  Length:120         Length:120         Length:120         Length:120
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##   Price.Unit          Low.Price     High.Price       Grade
##  Length:120         Min.   :171   Min.   :171.0   Length:120
##  Class :character   1st Qu.:171   1st Qu.:171.0   Class :character
##  Mode  :character   Median :171   Median :171.0   Mode  :character
##                     Mean   :171   Mean   :174.8
##                     3rd Qu.:171   3rd Qu.:179.0
##                     Max.   :171   Max.   :183.0
##
##    Mostly.Low     Mostly.High
##  Min.   :172.0   Min.   :174.0
##  1st Qu.:172.0   1st Qu.:174.0
##  Median :172.5   Median :174.0
##  Mean   :172.7   Mean   :174.8
##  3rd Qu.:173.0   3rd Qu.:176.0
##  Max.   :175.0   Max.   :176.0
##  NA's   :90      NA's   :95
```

```r
# Exclude rows where `Market.Name` is "CALIFORNIA"
df_filtered <- df %>%
  filter(Market.Name != "CALIFORNIA")
table(df_filtered$Market.Name)
```

```
##
##                  CHICAGO IOWA-MINNESOTA-WISCONSIN                   MIDWEST
##                     8994                     8994                     8998
##      SOUTHERN CALIFORNIA
##                     2476
```

```r
# Filter rows where `Egg.Class` matches "LARGE" or "JUMBO"
df_filtered <- df %>%
  filter(Egg.Class %in% c("LARGE", "JUMBO"))
table(df_filtered$Egg.Class)
```

```
##
## JUMBO LARGE
##  3929 12924
```

```r
# Filter rows with multiple conditions (AND condition)
df_filtered <- df %>%
  filter(Low.Price >= 200 & High.Price <= 200)
summary(df_filtered)
```

```
##      Date            Egg.Class         Market.Name        Delivery.Name
##  Length:52          Length:52          Length:52          Length:52
```

```
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##   Price.Unit          Low.Price     High.Price     Grade
##  Length:52          Min.   :200   Min.   :200   Length:52
##  Class :character   1st Qu.:200   1st Qu.:200   Class :character
##  Mode  :character   Median :200   Median :200   Mode  :character
##                     Mean   :200   Mean   :200
##                     3rd Qu.:200   3rd Qu.:200
##                     Max.   :200   Max.   :200
##
##    Mostly.Low   Mostly.High
##  Min.   : NA   Min.   : NA
##  1st Qu.: NA   1st Qu.: NA
##  Median : NA   Median : NA
##  Mean   :NaN   Mean   :NaN
##  3rd Qu.: NA   3rd Qu.: NA
##  Max.   : NA   Max.   : NA
##  NA's   :52    NA's   :52
```

```
# Filter rows with multiple conditions (OR condition)
df_filtered <- df %>%
  filter(Low.Price >= 200 | High.Price <= 200)
summary(df_filtered)
```

```
##      Date            Egg.Class         Market.Name        Delivery.Name
##  Length:42282       Length:42282       Length:42282       Length:42282
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##   Price.Unit          Low.Price       High.Price       Grade
##  Length:42282       Min.   :  0.0   Min.   :  5.0   Length:42282
##  Class :character   1st Qu.: 80.0   1st Qu.: 88.0   Class :character
##  Mode  :character   Median :127.0   Median :132.0   Mode  :character
##                     Mean   :154.3   Mean   :159.9
##                     3rd Qu.:190.0   3rd Qu.:194.0
##                     Max.   :899.0   Max.   :910.0
##
##    Mostly.Low     Mostly.High
##  Min.   :  3.0   Min.   : 24
##  1st Qu.: 73.0   1st Qu.: 81
##  Median :103.0   Median :110
##  Mean   :129.2   Mean   :138
##  3rd Qu.:154.0   3rd Qu.:160
##  Max.   :620.0   Max.   :612
##  NA's   :18634   NA's   :24557
```

```
# Exclude rows with missing values in `Mostly.Low`
df_filtered <- df %>%
  filter(!is.na(Mostly.Low))
summary(df_filtered)
```

```
##      Date            Egg.Class         Market.Name        Delivery.Name
##  Length:23976       Length:23976       Length:23976       Length:23976
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##   Price.Unit          Low.Price       High.Price       Grade
##  Length:23976       Min.   :  1.0   Min.   :  5.0   Length:23976
##  Class :character   1st Qu.: 71.0   1st Qu.: 79.0   Class :character
##  Mode  :character   Median :102.0   Median :110.0   Mode  :character
##                     Mean   :128.1   Mean   :136.2
##                     3rd Qu.:156.0   3rd Qu.:164.0
##                     Max.   :616.0   Max.   :626.0
##
##    Mostly.Low     Mostly.High
##  Min.   :  3.0   Min.   : 24.0
##  1st Qu.: 73.0   1st Qu.: 81.0
##  Median :104.0   Median :111.0
##  Mean   :130.1   Mean   :138.9
##  3rd Qu.:157.0   3rd Qu.:165.0
##  Max.   :620.0   Max.   :612.0
##                  NA's   :5996
```

Question The original dataframe `df` contains 42,702 rows and 10 columns. If we select the variables `Date`, `Market.Name`, `Egg.Class`, `Low.Price`, and `High.Price` and filter the rows where `Egg.Class` is either "LARGE" or "JUMBO," how many rows and columns will remain in the resulting dataframe?

```
df_filtered <- df %>%
  select(Date, Market.Name, Egg.Class, Low.Price, High.Price) %>%
  filter(Egg.Class %in% c("LARGE", "JUMBO"))

dim(df)
```

```
## [1] 42702    10
```

```
dim(df_filtered)
```

```
## [1] 16853     5
```

Question: what are some ways we could check that we have filtered down to the correct observations?

```
# Hint: `summary()` is a useful function for checking the range of values of a continous variable.

df_filtered <- df %>%
```

```
  select(Date, Market.Name, Egg.Class, Low.Price, High.Price) %>%
  filter(Market.Name != "CALIFORNIA" & Low.Price >= 200 & High.Price <= 300)

# Check price range
summary(df_filtered)
```

```
##      Date           Market.Name         Egg.Class           Low.Price
##  Length:3051        Length:3051        Length:3051        Min.   :200.0
##  Class :character   Class :character   Class :character   1st Qu.:214.0
##  Mode  :character   Mode  :character   Mode  :character   Median :231.0
##                                                           Mean   :235.9
##                                                           3rd Qu.:253.0
##                                                           Max.   :293.0
##    High.Price
##  Min.   :204.0
##  1st Qu.:223.0
##  Median :240.0
##  Mean   :244.6
##  3rd Qu.:262.0
##  Max.   :300.0
```

```
# Check the count of each market
table(df_filtered$Market.Name)
```

```
##
##                   CHICAGO IOWA-MINNESOTA-WISCONSIN                   MIDWEST
##                      1105                       524                      1038
##      SOUTHERN CALIFORNIA
##                       384
```

1.4 Adding or Changing Columns with `mutate()`

The mutate() function adds new columns or modifies existing ones.

```
# Applying multiple mutate() operations
df_new <- df %>%
  select(Date, Market.Name, Low.Price, High.Price) %>%
  mutate(Avg.Price = (Low.Price + High.Price) / 2,
         Price.Group= cut(Avg.Price, c(0, 200, 400, 600, 800, 1000)),
         High.Price.Flag = if_else(Low.Price > 150, "Above 150", "Below 150"))

unique(df_new$Price.Group)
```

```
## [1] (0,200]     (200,400]   <NA>        (400,600]   (600,800]   (800,1e+03]
## Levels: (0,200] (200,400] (400,600] (600,800] (800,1e+03]
```

```
head(df_new)
```

```
##         Date Market.Name Low.Price High.Price Avg.Price Price.Group
## 1 2013-02-04  CALIFORNIA       171        171       171     (0,200]
```

```
## 2 2013-02-04  CALIFORNIA       182        182       182     (0,200]
## 3 2013-02-04  CALIFORNIA       175        175       175     (0,200]
## 4 2013-02-04  CALIFORNIA       171        171       171     (0,200]
## 5 2013-02-04  CALIFORNIA       139        139       139     (0,200]
## 6 2013-02-04  CALIFORNIA       139        139       139     (0,200]
##   High.Price.Flag
## 1       Above 150
## 2       Above 150
## 3       Above 150
## 4       Above 150
## 5       Below 150
## 6       Below 150
```

1.5 Sorting Data with `arrange()`

The `arrange()` function sorts rows in a data frame. Use it to reorder data by one or more columns, either in ascending or descending order.

```r
# Sort data by a single column (ascending order by default)
df_arrange <- df %>%
  arrange(Low.Price)
head(df_arrange)
```

```
##         Date Egg.Class              Market.Name     Delivery.Name
## 1 2016-04-01     SMALL IOWA-MINNESOTA-WISCONSIN PAID TO PRODUCERS
## 2 2016-04-04     SMALL IOWA-MINNESOTA-WISCONSIN PAID TO PRODUCERS
## 3 2016-04-05     SMALL IOWA-MINNESOTA-WISCONSIN PAID TO PRODUCERS
## 4 2016-04-06     SMALL IOWA-MINNESOTA-WISCONSIN PAID TO PRODUCERS
## 5 2016-04-07     SMALL IOWA-MINNESOTA-WISCONSIN PAID TO PRODUCERS
## 6 2016-04-08     SMALL IOWA-MINNESOTA-WISCONSIN PAID TO PRODUCERS
##        Price.Unit Low.Price High.Price Grade Mostly.Low Mostly.High
## 1 CENTS PER DOZEN         0          6  <NA>         NA          NA
## 2 CENTS PER DOZEN         1          6  <NA>         NA          NA
## 3 CENTS PER DOZEN         1          6  <NA>         NA          NA
## 4 CENTS PER DOZEN         1          6  <NA>         NA          NA
## 5 CENTS PER DOZEN         1          6  <NA>         NA          NA
## 6 CENTS PER DOZEN         1          6  <NA>         NA          NA
```

```r
# Sort data in descending order
df_arrange <- df %>%
  arrange(desc(Low.Price))
head(df_arrange)
```

```
##         Date   Egg.Class Market.Name Delivery.Name      Price.Unit Low.Price
## 1 2024-12-23       JUMBO  CALIFORNIA       INVOICE CENTS PER DOZEN       899
## 2 2024-12-23 EXTRA LARGE  CALIFORNIA       INVOICE CENTS PER DOZEN       899
## 3 2024-12-23 EXTRA LARGE  CALIFORNIA       INVOICE CENTS PER DOZEN       899
## 4 2024-12-23       JUMBO  CALIFORNIA       INVOICE CENTS PER DOZEN       899
## 5 2024-12-26       JUMBO  CALIFORNIA       INVOICE CENTS PER DOZEN       899
## 6 2024-12-26 EXTRA LARGE  CALIFORNIA       INVOICE CENTS PER DOZEN       899
##   High.Price Grade Mostly.Low Mostly.High
## 1        899  <NA>         NA          NA
```

```
## 2        899  <NA>         NA          NA
## 3        899  <NA>         NA          NA
## 4        899  <NA>         NA          NA
## 5        899  <NA>         NA          NA
## 6        899  <NA>         NA          NA
```

```r
# Sort by multiple columns
df_arrange <- df %>%
  arrange(Market.Name, desc(Low.Price))
head(df_arrange)
```

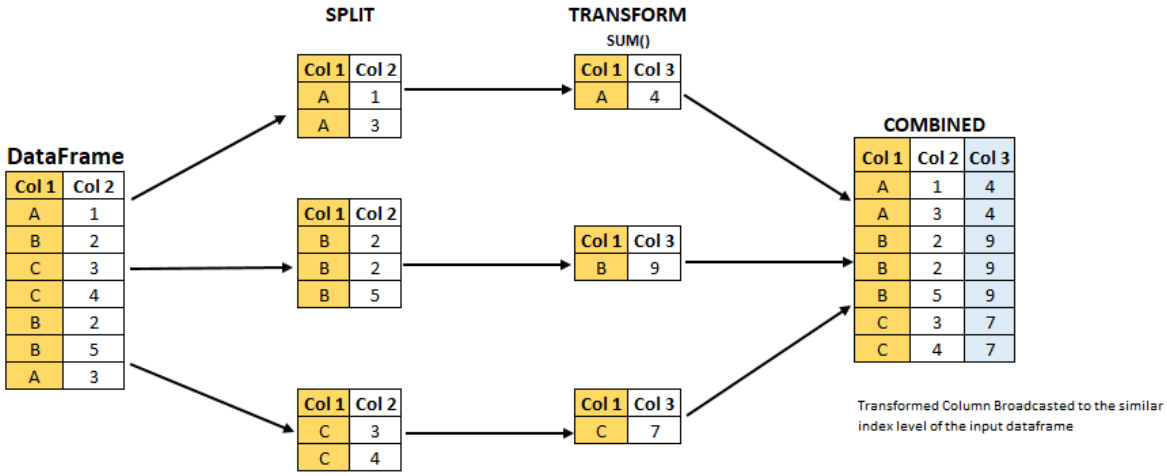
```
##         Date   Egg.Class Market.Name Delivery.Name      Price.Unit Low.Price
## 1 2024-12-23       JUMBO  CALIFORNIA       INVOICE CENTS PER DOZEN       899
## 2 2024-12-23 EXTRA LARGE  CALIFORNIA       INVOICE CENTS PER DOZEN       899
## 3 2024-12-23 EXTRA LARGE  CALIFORNIA       INVOICE CENTS PER DOZEN       899
## 4 2024-12-23       JUMBO  CALIFORNIA       INVOICE CENTS PER DOZEN       899
## 5 2024-12-26       JUMBO  CALIFORNIA       INVOICE CENTS PER DOZEN       899
## 6 2024-12-26 EXTRA LARGE  CALIFORNIA       INVOICE CENTS PER DOZEN       899
##   High.Price Grade Mostly.Low Mostly.High
## 1        899  <NA>         NA          NA
## 2        899  <NA>         NA          NA
## 3        899  <NA>         NA          NA
## 4        899  <NA>         NA          NA
## 5        899  <NA>         NA          NA
## 6        899  <NA>         NA          NA
```

```r
# Challenge: Handling missing values (place them at the end)
df_arrange <- df %>%
  arrange(desc(!is.na(Low.Price)), Low.Price)
head(df_arrange)
```

```
##         Date Egg.Class              Market.Name     Delivery.Name
## 1 2016-04-01     SMALL IOWA-MINNESOTA-WISCONSIN PAID TO PRODUCERS
## 2 2016-04-04     SMALL IOWA-MINNESOTA-WISCONSIN PAID TO PRODUCERS
## 3 2016-04-05     SMALL IOWA-MINNESOTA-WISCONSIN PAID TO PRODUCERS
## 4 2016-04-06     SMALL IOWA-MINNESOTA-WISCONSIN PAID TO PRODUCERS
## 5 2016-04-07     SMALL IOWA-MINNESOTA-WISCONSIN PAID TO PRODUCERS
## 6 2016-04-08     SMALL IOWA-MINNESOTA-WISCONSIN PAID TO PRODUCERS
##        Price.Unit Low.Price High.Price Grade Mostly.Low Mostly.High
## 1 CENTS PER DOZEN         0          6  <NA>         NA          NA
## 2 CENTS PER DOZEN         1          6  <NA>         NA          NA
## 3 CENTS PER DOZEN         1          6  <NA>         NA          NA
## 4 CENTS PER DOZEN         1          6  <NA>         NA          NA
## 5 CENTS PER DOZEN         1          6  <NA>         NA          NA
## 6 CENTS PER DOZEN         1          6  <NA>         NA          NA
```

1.6 Grouping data with `group_by()`

The `group_by` operation is a powerful and flexible tool used in data analysis to split data into groups based on some criteria, perform computations on each group, and then combine the results into a summary form. The diagram below illustrates the general steps.




```
# save the same dataframe but grouped by race
df_group <- df %>%
  group_by(Market.Name)

glimpse(df_group)
```

```
## Rows: 42,702
## Columns: 10
## Groups: Market.Name [5]
## $ Date          <chr> "2013-02-04", "2013-02-04", "2013-02-04", "2013-02-04", ~
## $ Egg.Class     <chr> "LARGE", "JUMBO", "EXTRA LARGE", "LARGE", "MEDIUM", "MED~
## $ Market.Name   <chr> "CALIFORNIA", "CALIFORNIA", "CALIFORNIA", "CALIFORNIA", ~
## $ Delivery.Name <chr> "INVOICE", "INVOICE", "INVOICE", "INVOICE", "INVOICE", "~
## $ Price.Unit    <chr> "CENTS PER DOZEN", "CENTS PER DOZEN", "CENTS PER DOZEN",~
## $ Low.Price     <int> 171, 182, 175, 171, 139, 139, 175, 182, 137, 60, 101, 13~
## $ High.Price    <int> 171, 182, 175, 171, 139, 139, 175, 182, 145, 66, 110, 14~
## $ Grade         <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, "A", "A", "A", N~
## $ Mostly.Low    <int> NA, NA, NA, NA, NA, NA, NA, NA, 138, NA, 103, 133, 135, ~
## $ Mostly.High   <int> NA, NA, NA, NA, NA, NA, NA, NA, 140, NA, 106, 136, 138, ~
```

```
group_keys(df_group)
```

```
## # A tibble: 5 x 1
##   Market.Name
##   <chr>
## 1 CALIFORNIA
## 2 CHICAGO
## 3 IOWA-MINNESOTA-WISCONSIN
## 4 MIDWEST
## 5 SOUTHERN CALIFORNIA
```

1.7 Summarizing Data with `summarize()`

The `summarize()` function creates summary statistics. Use it with `group_by()` for group-level summaries.

```r
# Compute the average miles per gallon
df_summarize <- df %>%
  summarize(avg_low_price = mean(Low.Price))
df_summarize
```

```
##   avg_low_price
## 1            NA
```

```r
# Challenge: Missing values
df_summarize <- df %>%
  filter(!is.na(Low.Price)) %>%
  summarize(avg_low_price = mean(Low.Price))
df_summarize
```

```
##   avg_low_price
## 1      154.7022
```

```r
# Summarize by group
df_summarize <- df %>%
  filter(!is.na(Low.Price)) %>%
  group_by(Market.Name) %>%
  summarize(avg_low_price = mean(Low.Price), count = n())
df_summarize
```

```
## # A tibble: 5 x 3
##   Market.Name              avg_low_price count
##   <chr>                            <dbl> <int>
## 1 CALIFORNIA                        217.  13240
## 2 CHICAGO                           138.   8988
## 3 IOWA-MINNESOTA-WISCONSIN           90.3  8994
## 4 MIDWEST                           132.   8992
## 5 SOUTHERN CALIFORNIA               201.   2476
```

1.8 Practice

Starting with the original tidy dataset, perform the following calculations for each market:

- Calculate the average low price for eggs classified as "EXTRA LARGE."
- Calculate the proportion of "LARGE" eggs with High.Price ranging between 200 and 500.

```r
extra_large_avg <- df %>%
  group_by(Market.Name) %>%
  filter(Egg.Class == "EXTRA LARGE") %>%
  summarize(Average.Low.Price = mean(Low.Price, na.rm = TRUE))
print(extra_large_avg)
```

```
## # A tibble: 4 x 2
##   Market.Name         Average.Low.Price
##   <chr>                           <dbl>
```

```
## 1 CALIFORNIA                          228.
## 2 CHICAGO                             150.
## 3 MIDWEST                             143.
## 4 SOUTHERN CALIFORNIA                 211.
```

```
large_egg_prop <- df %>%
  filter(Egg.Class == "LARGE") %>%
  group_by(Market.Name) %>%
  summarize(
    Proportion = sum(High.Price >= 200 & High.Price <= 500, na.rm = TRUE) / n()
  )
print(large_egg_prop)
```

```
## # A tibble: 5 x 2
##   Market.Name              Proportion
##   <chr>                         <dbl>
## 1 CALIFORNIA                    0.330
## 2 CHICAGO                       0.220
## 3 IOWA-MINNESOTA-WISCONSIN      0.182
## 4 MIDWEST                       0.216
## 5 SOUTHERN CALIFORNIA           0.304
```

2 Data Visualization

The most common set of tools for creating plots in R is `ggplot2`, which we will use throughout this course.

In late 2022, egg prices quintupled due to a severe outbreak of avian flu (H5N1), which led to the culling of millions of chickens and significantly reduced the supply of eggs. A similar situation is unfolding now, with California wholesale egg prices surpassing their 2022 peak (unadjusted for inflation). For more details, refer to the blog Eggflation Returns with a Vengeance. Let's replicate the first figure from the blog here.

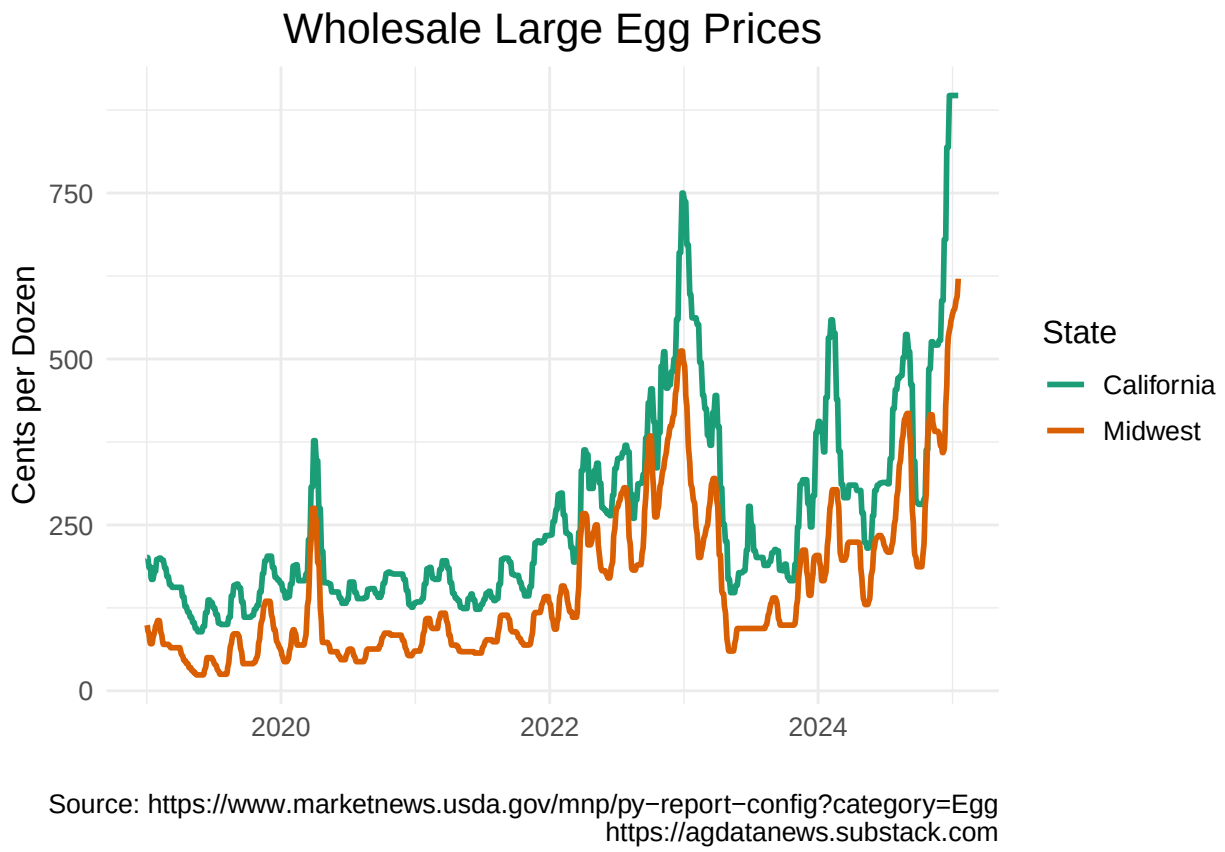
```
df$Date <- as.Date(df$Date, format = "%Y-%m-%d")

start_date <- as.Date("2019-01-01")

## Plot prices
plot_prices <- df %>%
  arrange(Date) %>%
  filter(Date>=start_date) %>%
  filter(Egg.Class %in% c("LARGE")) %>%
  filter(Market.Name %in% c("CALIFORNIA","IOWA-MINNESOTA-WISCONSIN")) %>%
  mutate(Market.Name=str_to_title(
  ifelse(Market.Name=="IOWA-MINNESOTA-WISCONSIN","Midwest",Market.Name))) %>%
  mutate(Price=(Low.Price+High.Price)/2) %>%
  ggplot(
    aes(
      x=Date,
      y=Price,
      color=factor(Market.Name, levels=c("California","Midwest")))) +
  geom_line(linewidth=1) +
  labs(x = "",
       y="Cents per Dozen",
```

```
       color="State",
       caption="Source: https://www.marketnews.usda.gov/mnp/py-report-config?category=Egg\n https://agda
  ggtitle("Wholesale Large Egg Prices")+
  theme_minimal()+
  scale_color_brewer(palette = "Dark2") +
  theme(plot.title = element_text(hjust = 0.5,size=16), text = element_text(size=12))

# draw and save plot
plot_prices
```




```
ggsave(paste0("daily_egg_prices_2024_",start_date,".png"),bg="white")
```

```
## Saving 6.5 x 4.5 in image
```