# Data Management Using Tidyverse

Myung Jung Kim

8/7/2023

# Introduction

- **Tidyverse** is a powerful data management tool. It's a collection of R packages like `ggplot2`, `dplyr`, `tidyr`, `stringr` – which include tools for data manipulation, visualization and string manipulation. In this section, we focus on data management tools (mainly `dplyr` and `tidyr`).
- Using tidyverse, we will learn how to do:
  1. Selecting and Sorting Columns
  2. Data Reduction and Expansion
  3. Grouping and Summarizing
- Basic grammar: **Pipe operator (%>%)** combines various functions without the result to a new object.
  - data %>% verb 1 %>% verb 2 %>% verb 3

# Open Demo Data sets: ***gapminder***, ***diamond***

```r
## Gapminder dataset (in gapminder package)
install.packages("gapminder")
library(gapminder)
data(gapminder) # load data
names(gapminder)# check the variable names
View(gapminder) # open a data viewer

## Diamond dataset (in tidyverse package)
install.packages("tidyverse")
library(tidyverse)
data(diamonds) # load data
names(diamonds)# check the variable names
View(diamonds) # open a data viewer
```

Section 1

# Selecting and Arranging Columns

# 1-1. select() function

- Selecting certain columns
  - gapminder has 6 variables: country, continent, year, lifeExp, pop, gdpPercap.
  - Let's say we only need 3 columns: country, year, gdpPercap.

```
## Put data name, %>% (pipe), and select() function
gapminder %>%
  select(country,year,gdpPercap)

## You can save it as a separate dataset (gapminder2)
gapminder2 <- gapminder %>%
  select(country,year,gdpPercap)
```

- De-selecting certain columns
  - You **de-select** certain columns using `select()` with **minus sign (-)**.
  - let's say you don't need 'continent' and 'pop' from the **gapminder**.

```
## De-select continent, pop
gapminder %>%
  select(-continent, -pop)

## Save it as a separate dataset (gapminder3)
gapminder3 <- gapminder %>%
  select(-continent, -pop)
```

- **everything()** function is useful when you want to locate few specific variables to the front and leave everything else as now.

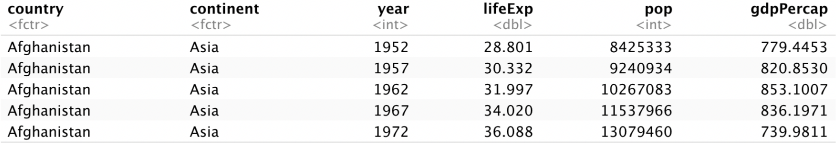
```
## move lifeExp to the front
gapminder %>%
  select(lifeExp, everything())
```

# Exercise 1-1. Selecting Columns

1. How many variables do we have in the diamonds dataset?
2. Can you list the names?
3. **Select** columns ‘carat’, ‘color’, ‘clarity’ from diamonds and save it as a separate dataset called diamond2.
4. **De-select** columns ‘x’, ‘y’, ‘z’ from the diamonds dataset.
5. Move ‘x’, ‘y’, ‘z’ to the very front from the original diamonds dataset and leave every other columns as the order of now.

# 1-2. arrange() function

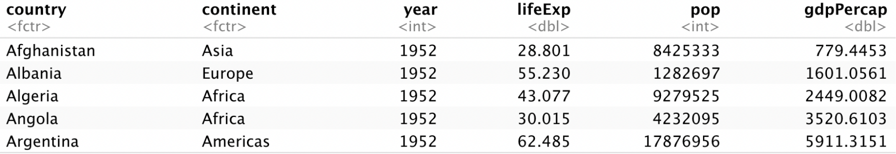
- **arrange()** helps reorder the rows of data by using column names.
- Right now, gapminder's column orders by "**country**", alphabetically.



| country<br>&lt;fctr&gt; | continent<br>&lt;fctr&gt; | year<br>&lt;int&gt; | lifeExp<br>&lt;dbl&gt; | pop<br>&lt;int&gt; | gdpPercap<br>&lt;dbl&gt; |
|---|---|---:|---:|---:|---:|
| Afghanistan | Asia | 1952 | 28.801 | 8425333 | 779.4453 |
| Afghanistan | Asia | 1957 | 30.332 | 9240934 | 820.8530 |
| Afghanistan | Asia | 1962 | 31.997 | 10267083 | 853.1007 |
| Afghanistan | Asia | 1967 | 34.020 | 11537966 | 836.1971 |
| Afghanistan | Asia | 1972 | 36.088 | 13079460 | 739.9811 |

- If you want to rearrange the data by "**year**":

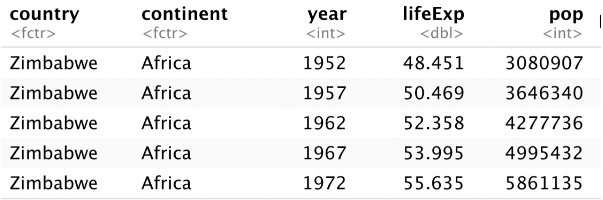
```
gapminder %>%   arrange(year)
```



| country<br>&lt;fctr&gt; | continent<br>&lt;fctr&gt; | year<br>&lt;int&gt; | lifeExp<br>&lt;dbl&gt; | pop<br>&lt;int&gt; | gdpPercap<br>&lt;dbl&gt; |
|---|---|---:|---:|---:|---:|
| Afghanistan | Asia | 1952 | 28.801 | 8425333 | 779.4453 |
| Albania | Europe | 1952 | 55.230 | 1282697 | 1601.0561 |
| Algeria | Africa | 1952 | 43.077 | 9279525 | 2449.0082 |
| Angola | Africa | 1952 | 30.015 | 4232095 | 3520.6103 |
| Argentina | Americas | 1952 | 62.485 | 17876956 | 5911.3151 |

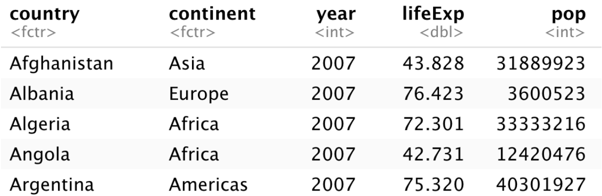
- In R, the default setting arranges data in **ascending** order (either alphabet or number). If you want to arrange data in **descending** order, you need to specify it using `desc()`.

```
gapminder %>%  arrange(desc(country))
```



| country<br>&lt;fctr&gt; | continent<br>&lt;fctr&gt; | year<br>&lt;int&gt; | lifeExp<br>&lt;dbl&gt; | pop<br>&lt;int&gt; |
|---|---|---|---|---|
| Zimbabwe | Africa | 1952 | 48.451 | 3080907 |
| Zimbabwe | Africa | 1957 | 50.469 | 3646340 |
| Zimbabwe | Africa | 1962 | 52.358 | 4277736 |
| Zimbabwe | Africa | 1967 | 53.995 | 4995432 |
| Zimbabwe | Africa | 1972 | 55.635 | 5861135 |

```
gapminder %>%  arrange(desc(year))
```



| country<br>&lt;fctr&gt; | continent<br>&lt;fctr&gt; | year<br>&lt;int&gt; | lifeExp<br>&lt;dbl&gt; | pop<br>&lt;int&gt; |
|---|---|---|---|---|
| Afghanistan | Asia | 2007 | 43.828 | 31889923 |
| Albania | Europe | 2007 | 76.423 | 3600523 |
| Algeria | Africa | 2007 | 72.301 | 33333216 |
| Angola | Africa | 2007 | 42.731 | 12420476 |
| Argentina | Americas | 2007 | 75.320 | 40301927 |

- rename() function: it's to change the name of a column.
    - data %>% **rename**(NEW name = OLD column name)

```
gapminder %>%
  rename (state = country)
```

# Exercise 1-2. Sorting Columns

1. Arrange diamonds dataset by a column **cut**.
2. Arrange diamonds dataset by **carat** in descending order.
3. Rename the column ‘clarity’ into ‘Clarity’.

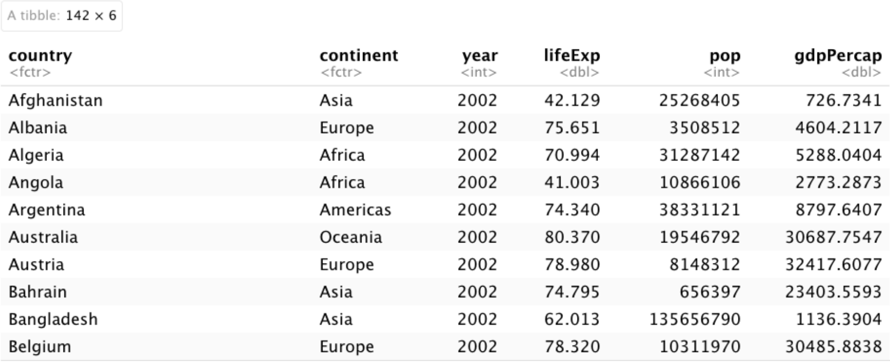
# Section 2

## Data Reduction and Expansion

# 2-1. filter() function: conditional filtering

- You use filter when you want to look only at a subset of your observations, based on a particular condition.

```
gapminder %>%
  filter(year == 2002)
```

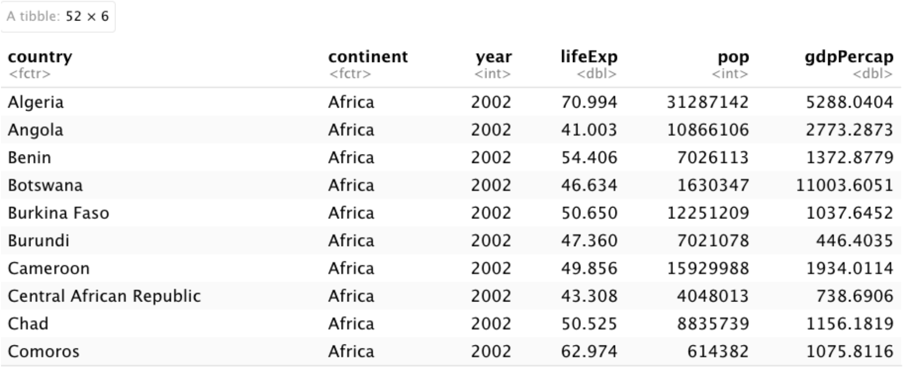


A tibble: 142 × 6

| country<br>&lt;fctr&gt; | continent<br>&lt;fctr&gt; | year<br>&lt;int&gt; | lifeExp<br>&lt;dbl&gt; | pop<br>&lt;int&gt; | gdpPercap<br>&lt;dbl&gt; |
|---|---|---:|---:|---:|---:|
| Afghanistan | Asia | 2002 | 42.129 | 25268405 | 726.7341 |
| Albania | Europe | 2002 | 75.651 | 3508512 | 4604.2117 |
| Algeria | Africa | 2002 | 70.994 | 31287142 | 5288.0404 |
| Angola | Africa | 2002 | 41.003 | 10866106 | 2773.2873 |
| Argentina | Americas | 2002 | 74.340 | 38331121 | 8797.6407 |
| Australia | Oceania | 2002 | 80.370 | 19546792 | 30687.7547 |
| Austria | Europe | 2002 | 78.980 | 8148312 | 32417.6077 |
| Bahrain | Asia | 2002 | 74.795 | 656397 | 23403.5593 |
| Bangladesh | Asia | 2002 | 62.013 | 135656790 | 1136.3904 |
| Belgium | Europe | 2002 | 78.320 | 10311970 | 30485.8838 |

- Filtering condition **“A and B”**: we can specify multiple conditions in the filter. Each of the conditions is separated by a comma:

```
gapminder %>%
  filter(year == 2002, continent == "Africa")
```

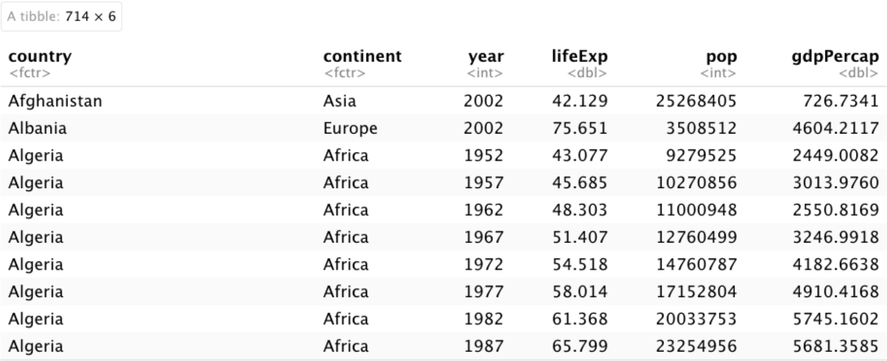


A tibble: 52 × 6

| country<br>&lt;fctr&gt; | continent<br>&lt;fctr&gt; | year<br>&lt;int&gt; | lifeExp<br>&lt;dbl&gt; | pop<br>&lt;int&gt; | gdpPercap<br>&lt;dbl&gt; |
|---|---|---:|---:|---:|---:|
| Algeria | Africa | 2002 | 70.994 | 31287142 | 5288.0404 |
| Angola | Africa | 2002 | 41.003 | 10866106 | 2773.2873 |
| Benin | Africa | 2002 | 54.406 | 7026113 | 1372.8779 |
| Botswana | Africa | 2002 | 46.634 | 1630347 | 11003.6051 |
| Burkina Faso | Africa | 2002 | 50.650 | 12251209 | 1037.6452 |
| Burundi | Africa | 2002 | 47.360 | 7021078 | 446.4035 |
| Cameroon | Africa | 2002 | 49.856 | 15929988 | 1934.0114 |
| Central African Republic | Africa | 2002 | 43.308 | 4048013 | 738.6906 |
| Chad | Africa | 2002 | 50.525 | 8835739 | 1156.1819 |
| Comoros | Africa | 2002 | 62.974 | 614382 | 1075.8116 |

- Filtering condition **“A or B”**: we can specify multiple conditions in the filter. Each of the conditions is separated by a “|” :

```
gapminder %>%
  filter(year == 2002 | continent == "Africa")
```

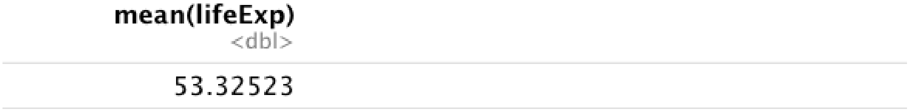


A tibble: 714 × 6

| country<br><fctr> | continent<br><fctr> | year<br><int> | lifeExp<br><dbl> | pop<br><int> | gdpPercap<br><dbl> |
|---|---|---|---|---|---|
| Afghanistan | Asia | 2002 | 42.129 | 25268405 | 726.7341 |
| Albania | Europe | 2002 | 75.651 | 3508512 | 4604.2117 |
| Algeria | Africa | 1952 | 43.077 | 9279525 | 2449.0082 |
| Algeria | Africa | 1957 | 45.685 | 10270856 | 3013.9760 |
| Algeria | Africa | 1962 | 48.303 | 11000948 | 2550.8169 |
| Algeria | Africa | 1967 | 51.407 | 12760499 | 3246.9918 |
| Algeria | Africa | 1972 | 54.518 | 14760787 | 4182.6638 |
| Algeria | Africa | 1977 | 58.014 | 17152804 | 4910.4168 |
| Algeria | Africa | 1982 | 61.368 | 20033753 | 5745.1602 |
| Algeria | Africa | 1987 | 65.799 | 23254956 | 5681.3585 |

- You can summarize information of the filtered data.

```
## What's the mean life expectancy in Africa in year 2002?
gapminder %>%
  filter(year == 2002, continent == "Africa")%>%
  summarize(mean(lifeExp))
```

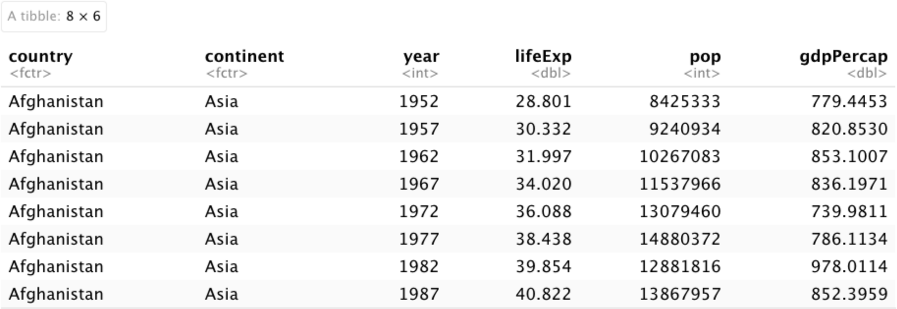


| mean(lifeExp)<br><dbl> |
|---:|
| 53.32523 |

# 2-2. slice() function:

- Filtering rows works well when we know the data columns and values by name. But if that information is not as readily available, we can reduce the data using slice() functions:
  - %>%**slice**(100:300): slice rows 100 to 300
  - %>%**slice_head**(n= 10): show the first n rows
  - %>%**slice_tail**(n= 10): show the last n rows
  - %>%**slice_min**(column name): show the smallest value of the column
  - %>%**slice_max**(column name): show the largest value of the column

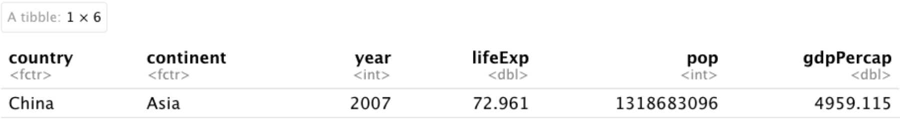
```
gapminder %>%
   slice_head(n=8)
```



A tibble: 8 × 6

| country<br><fctr> | continent<br><fctr> | year<br><int> | lifeExp<br><dbl> | pop<br><int> | gdpPercap<br><dbl> |
|---|---|---:|---:|---:|---:|
| Afghanistan | Asia | 1952 | 28.801 | 8425333 | 779.4453 |
| Afghanistan | Asia | 1957 | 30.332 | 9240934 | 820.8530 |
| Afghanistan | Asia | 1962 | 31.997 | 10267083 | 853.1007 |
| Afghanistan | Asia | 1967 | 34.020 | 11537966 | 836.1971 |
| Afghanistan | Asia | 1972 | 36.088 | 13079460 | 739.9811 |
| Afghanistan | Asia | 1977 | 38.438 | 14880372 | 786.1134 |
| Afghanistan | Asia | 1982 | 39.854 | 12881816 | 978.0114 |
| Afghanistan | Asia | 1987 | 40.822 | 13867957 | 852.3959 |

```
gapminder %>%
   slice_max(pop)
```



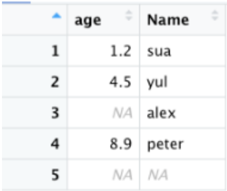
A tibble: 1 × 6

| country<br><fctr> | continent<br><fctr> | year<br><int> | lifeExp<br><dbl> | pop<br><int> | gdpPercap<br><dbl> |
|---|---|---:|---:|---:|---:|
| China | Asia | 2007 | 72.961 | 1318683096 | 4959.115 |

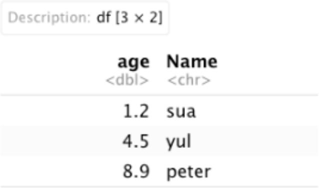
# 2-3. dropping NAs: drop_na()

- If you don't know a value is missing or not, you can use the base R function **is.na()**.
- **drop_na()** function removes NA values.

```
table(is.na(gapminder))# no NAs.
# Create a vector with NAs
v <- data.frame("age"=c(1.2, 4.5, NA, 8.9, NA),
                "Name" = c("sua", "yul", "alex","peter",NA))
# Drop NAs
v %>%  drop_na
```



| | age | Name |
|---|---|---|
| 1 | 1.2 | sua |
| 2 | 4.5 | yul |
| 3 | NA | alex |
| 4 | 8.9 | peter |
| 5 | NA | NA |



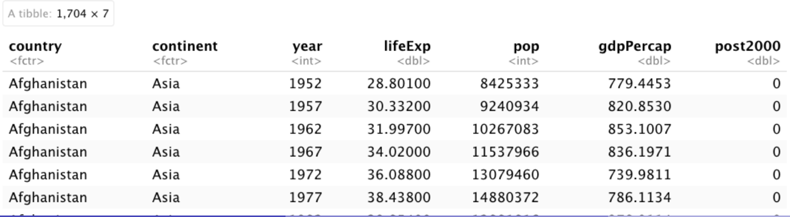
Description: df [3 × 2]

| age<br><dbl> | Name<br><chr> |
|---|---|
| 1.2 | sua |
| 4.5 | yul |
| 8.9 | peter |

# 2-4. mutate() function: create columns of new information

- **mutate()** creates new variables (and preserves existing ones).
- **transmute()** adds new variables and drops existing ones.
- Let's make a new dummy variable called `post2000` if `year` is 2000 or later.
- **ifelse**(test_expression, x, y): The output vector has the element x if the output of the test_expression is TRUE. If the output is FALSE, then the element in the output vector will be y.

```
# New column "post2000" using mutate()
gapminder %>%
  mutate(post2000 = ifelse(year >=2000, 1, 0))
```



A tibble: 1,704 × 7

| country<br>&lt;fctr&gt; | continent<br>&lt;fctr&gt; | year<br>&lt;int&gt; | lifeExp<br>&lt;dbl&gt; | pop<br>&lt;int&gt; | gdpPercap<br>&lt;dbl&gt; | post2000<br>&lt;dbl&gt; |
|---|---|---|---|---|---|---|
| Afghanistan | Asia | 1952 | 28.80100 | 8425333 | 779.4453 | 0 |
| Afghanistan | Asia | 1957 | 30.33200 | 9240934 | 820.8530 | 0 |
| Afghanistan | Asia | 1962 | 31.99700 | 10267083 | 853.1007 | 0 |
| Afghanistan | Asia | 1967 | 34.02000 | 11537966 | 836.1971 | 0 |
| Afghanistan | Asia | 1972 | 36.08800 | 13079460 | 739.9811 | 0 |
| Afghanistan | Asia | 1977 | 38.43800 | 14880372 | 786.1134 | 0 |

# Exercise 2. Data Reduction and Expansion

1. From the `diamonds` dataset, filter rows so that you only observe **Premium** cut **AND** carat size **above 0.70**. What's the ***mean price*** of the diamonds that meet the conditions?
2. Print the last 10 rows of the `diamond` dataset.
3. Is there NA in the diamond dataset? If so, drop all NAs and save it as `diamonds3`.
4. Create a new column called "standard" in the `diamonds` dataset and fill it with "high" if the cut is either Preimium or Ideal, and otherwise "low".

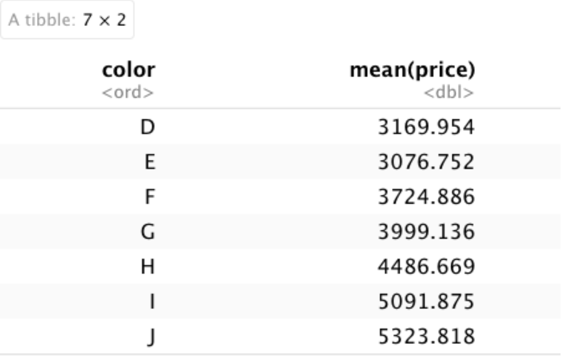
Section 3

# Grouping and Summarizing

# group_by() function with summarize() function

- If we want to assess information by group, we use **group_by()** together with **summarize()**.

```
## There are 7 color groups in the diamonds dataset.
# What's the average price of each color group?
diamonds %>%  group_by(color)%>%  summarize(mean(price))
```



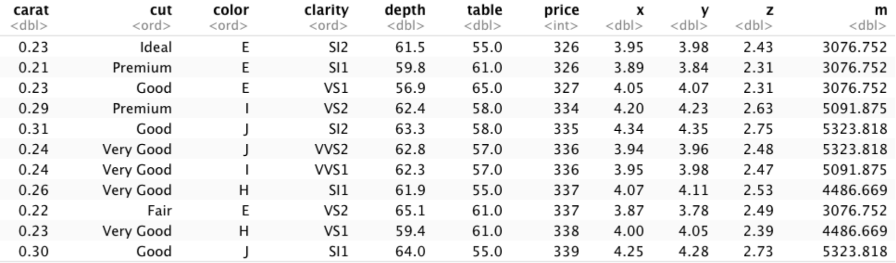
A tibble: 7 × 2

| color<br><ord> | mean(price)<br><dbl> |
|---|---|
| D | 3169.954 |
| E | 3076.752 |
| F | 3724.886 |
| G | 3999.136 |
| H | 4486.669 |
| I | 5091.875 |
| J | 5323.818 |

# group_by(), mutate(), and ungroup() together

- We could also utilize `mutate()` after `group_by()` to add a new column based on the group.

```r
## A new column 'm' for mean price of same color
diamonds %>%
  group_by(color)%>%
  mutate(m = mean(price))%>%
  ungroup() #ungroup after creating new object based on group
```

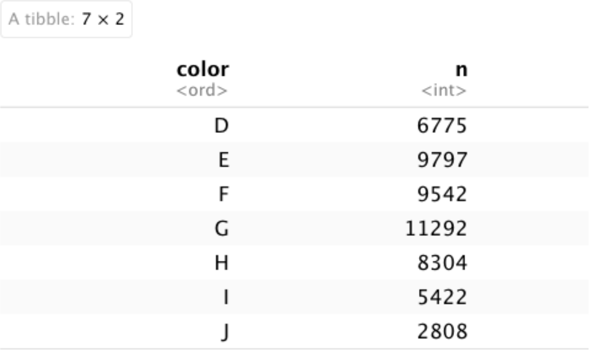


| carat<br><dbl> | cut<br><ord> | color<br><ord> | clarity<br><ord> | depth<br><dbl> | table<br><dbl> | price<br><int> | x<br><dbl> | y<br><dbl> | z<br><dbl> | m<br><dbl> |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.23 | Ideal | E | SI2 | 61.5 | 55.0 | 326 | 3.95 | 3.98 | 2.43 | 3076.752 |
| 0.21 | Premium | E | SI1 | 59.8 | 61.0 | 326 | 3.89 | 3.84 | 2.31 | 3076.752 |
| 0.23 | Good | E | VS1 | 56.9 | 65.0 | 327 | 4.05 | 4.07 | 2.31 | 3076.752 |
| 0.29 | Premium | I | VS2 | 62.4 | 58.0 | 334 | 4.20 | 4.23 | 2.63 | 5091.875 |
| 0.31 | Good | J | SI2 | 63.3 | 58.0 | 335 | 4.34 | 4.35 | 2.75 | 5323.818 |
| 0.24 | Very Good | J | VVS2 | 62.8 | 57.0 | 336 | 3.94 | 3.96 | 2.48 | 5323.818 |
| 0.24 | Very Good | I | VVS1 | 62.3 | 57.0 | 336 | 3.95 | 3.98 | 2.47 | 5091.875 |
| 0.26 | Very Good | H | SI1 | 61.9 | 55.0 | 337 | 4.07 | 4.11 | 2.53 | 4486.669 |
| 0.22 | Fair | E | VS2 | 65.1 | 61.0 | 337 | 3.87 | 3.78 | 2.49 | 3076.752 |
| 0.23 | Very Good | H | VS1 | 59.4 | 61.0 | 338 | 4.00 | 4.05 | 2.39 | 4486.669 |
| 0.30 | Good | J | SI1 | 64.0 | 55.0 | 339 | 4.25 | 4.28 | 2.73 | 5323.818 |

# count(), tally()

- When we want to count observations by group, we use **count()** or **tally()** . They are convenient ways to get a sense of the distribution of values in a dataset.

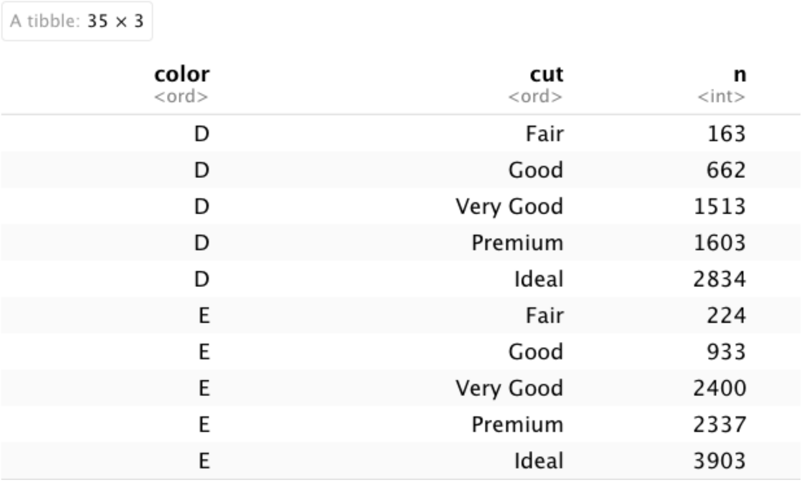
```
diamonds %>%  count(color)
diamonds %>%  group_by(color)%>% tally() #Same results
```



A tibble: 7 × 2

| color<br><ord> | n<br><int> |
|---|---|
| D | 6775 |
| E | 9797 |
| F | 9542 |
| G | 11292 |
| H | 8304 |
| I | 5422 |
| J | 2808 |

```
# You can count based on multiple groupping conditions
## 7 colors and 5 cuts => 35 groups
diamonds %>%  count(color, cut)
```



A tibble: 35 × 3

| color | cut | n |
|---:|---:|---:|
| <ord> | <ord> | <int> |
| D | Fair | 163 |
| D | Good | 662 |
| D | Very Good | 1513 |
| D | Premium | 1603 |
| D | Ideal | 2834 |
| E | Fair | 224 |
| E | Good | 933 |
| E | Very Good | 2400 |
| E | Premium | 2337 |
| E | Ideal | 3903 |

# Exercise 3. Grouping and Summarizing

1. From the `gapminder` dataset, what's the mean life expectancy of each continent?
2. From the `gapminder` dataset, what's the mean life expectancy of each continent in **year 2007**?
3. From the `gapminder` dataset, create a new column called **`continent_avg_life`** which shows each **continent's average life expectancy** (lifeExp) in the **particular year**. Save it as a new data called **`gapminder4`**.
4. Using **group_by()**, count how many countries belong to each continent. (Hint: use **unique(country)** somewhere)

***Any Question?***
*Feel free to reach out for anything :)*
*- Myung Jung Kim (mjkim@illinois.edu)*