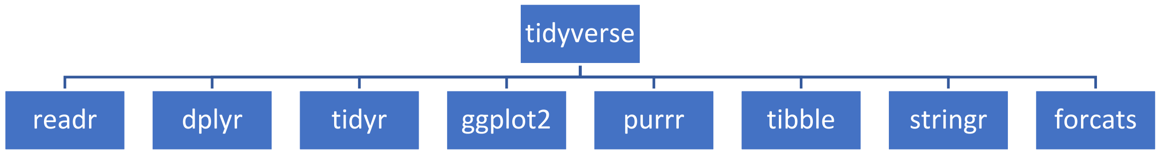
Data Wrangling – Part 1

tidyverse Package

Tidyverse: is a collection of R packages designed to facilitate data manipulation, exploration, visualization, and modeling.



- ***readr:*** provides flexible tools for reading and writing rectangular data.
- ***dplyr:*** provides a grammar of data manipulation.
- ***tidyr:*** reshape and tidy your data.
- ***ggplot2:*** allows you to build plots layer by layer.
- ***purrr:*** purrr provides functions to work with and manipulate data in a functional programming style.
- ***tibble:*** tibble is an enhanced data frame that provides better printing and handling of metadata.
- ***stringr:*** focuses on string manipulation.
- ***forcats:*** forcats is designed for working with categorical data.

Reading Data

1. If you have an Excel File that contains your data, you can read it using directories.
2. You can pull data directly from packages.

Note: In each of these methods, it will be saved as a dataframe in your environment.

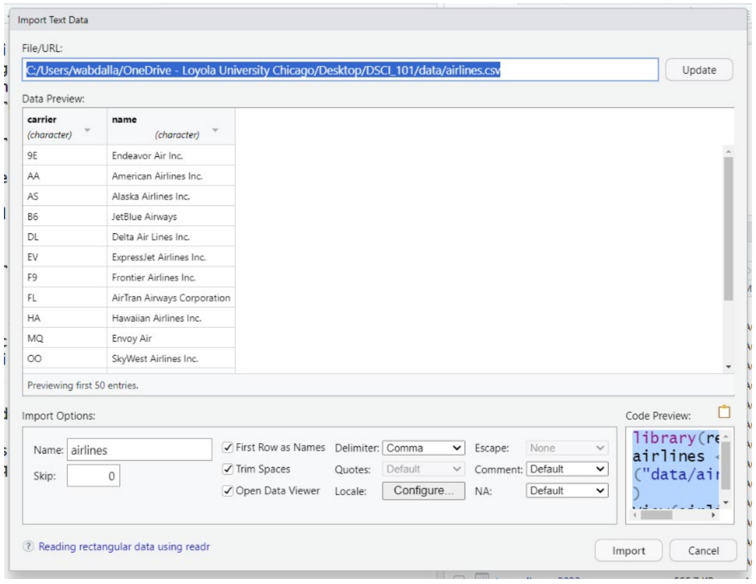
Reading Data: Method 1 – Excel files

Step 1: load the tidyverse package.

Step 2: Find the file using the Files window in RStudio.

Step 3: Click on the desired Excel file, then click on “Import Dataset”.

Step 4: A window will open. The desired directory is on the top of this window. Copy this directory as it is.


Import Text Data

File/URL: C:/Users/wabdalla/OneDrive - Loyola University Chicago/Desktop/DSCI_101/data/airlines.csv

Data Preview:

carrier (character)	name (character)
9E	Endeavor Air Inc.
AA	American Airlines Inc.
AS	Alaska Airlines Inc.
B6	JetBlue Airways
DL	Delta Air Lines Inc.
EV	ExpressJet Airlines Inc.
F9	Frontier Airlines Inc.
FL	AirTran Airways Corporation
HA	Hawaiian Airlines Inc.
MQ	Envoy Air
OO	SkyWest Airlines Inc.

Previewing first 50 entries.

Import Options:

Name: airlines
Skip: 0
✓ First Row as Names
✓ Trim Spaces
✓ Open Data Viewer
Delimiter: Comma
Quotes: Default
Locale: Configure...
Escape: None
Comment: Default
NA: Default

Code Preview:

```
library(re
airlines
("data/ai
)
```

Step 5: Choose a name for your dataframe, then type in your RScript:

```
name_dataframe <- read_csv(“copy the directory here”)
```

This will save your file as a dataframe in your Environment, which now you can use to do any data analysis.

Reading Data: Method 2 – Pulling Data Directly from a Package

Step 1: load the package that you will use to extract the data from it.

Step 2: in your RScript, type:

```
data("name_of_data")
```

Example 1:

```
library(palmerpenguins)
data("penguins")
```

Note: When reading a data directly from a package, always type in the console:

```
?name_of_dataset
```

This will give you all the information on that data.

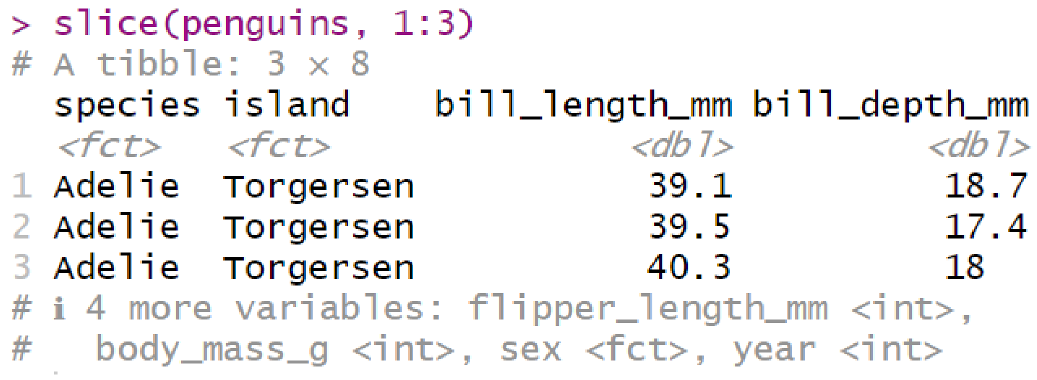
Examining the Data

To check imported data, you can examine it using functions like `glimpse()`, `slice()`, `str()`, `table()`, and `colnames()`

- `slice()`: Display the first few rows

Command: `slice(name_of_dataframe, 1:5)`

Example 2: `slice(penguins, 1:3)`

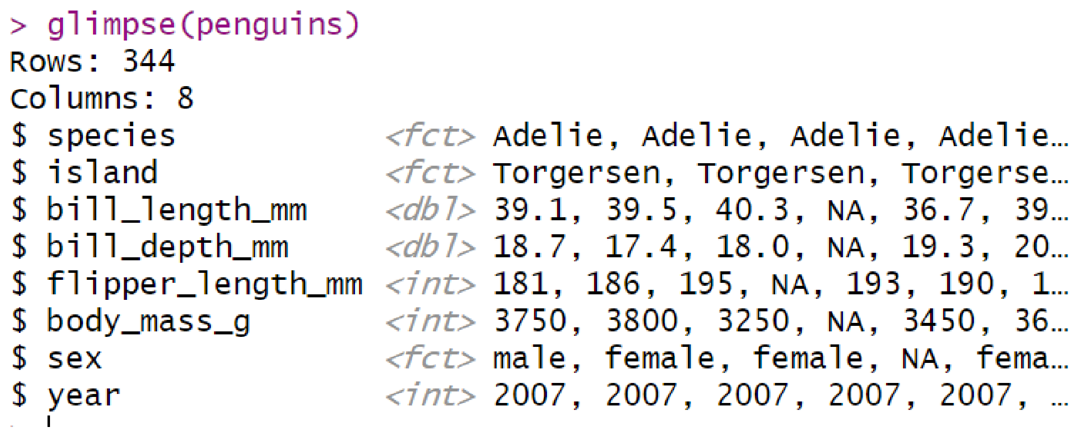


```
> slice(penguins, 1:3)
# A tibble: 3 × 8
  species island    bill_length_mm bill_depth_mm
  <fct>   <fct>              <dbl>         <dbl>
1 Adelie  Torgersen           39.1          18.7
2 Adelie  Torgersen           39.5          17.4
3 Adelie  Torgersen           40.3          18
# ℹ 4 more variables: flipper_length_mm <int>,
#   body_mass_g <int>, sex <fct>, year <int>
```

- `glimpse()`: Get summary of the variables

Command: `glimpse(name_of_dataframe)`

Example 3: `glimpse(penguins)`

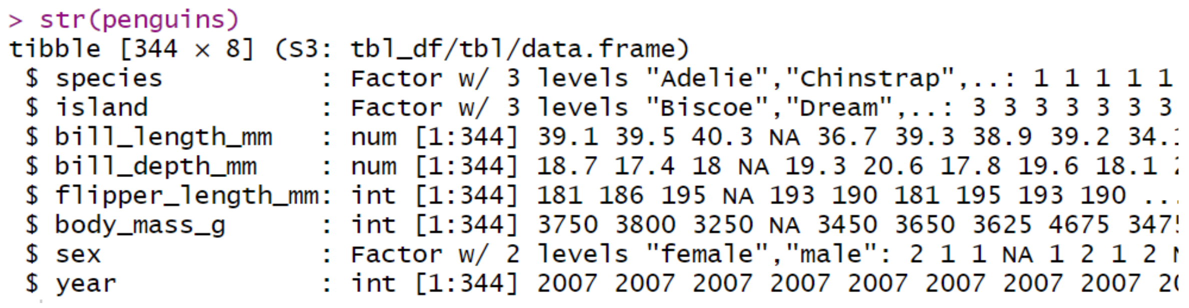


```
> glimpse(penguins)
Rows: 344
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie, Adelie…
$ island            <fct> Torgersen, Torgersen, Torgerse…
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39…
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20…
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 1…
$ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 36…
$ sex               <fct> male, female, female, NA, fema…
$ year              <int> 2007, 2007, 2007, 2007, 2007, …
```

- `str()`: Display the structure of the data

Command: `str(name_of_dataframe)`

Example 4: `str(penguins)`



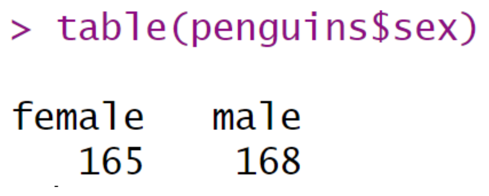
```
> str(penguins)
tibble [344 × 8] (S3: tbl_df/tbl/data.frame)
 $ species          : Factor w/ 3 levels "Adelie","Chinstrap",..: 1 1 1 1 1
 $ island           : Factor w/ 3 levels "Biscoe","Dream",..: 3 3 3 3 3 3 3
 $ bill_length_mm   : num [1:344] 39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.
 $ bill_depth_mm    : num [1:344] 18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1
 $ flipper_length_mm: int [1:344] 181 186 195 NA 193 190 181 195 193 190 ..
 $ body_mass_g      : int [1:344] 3750 3800 3250 NA 3450 3650 3625 4675 347
 $ sex              : Factor w/ 2 levels "female","male": 2 1 1 NA 1 2 1 2
 $ year             : int [1:344] 2007 2007 2007 2007 2007 2007 2007 2007 2
```

- `table()`: it's typically used for one or two columns. For one column, it tabulates each observation, and shows how many times each observation appeared (or repeats) in that column.

Command: `table(name_of_dataframe$name_of_column)`

`table(name_of_dataframe$name_of_column1, name_of_dataframe$name_of_column2)`

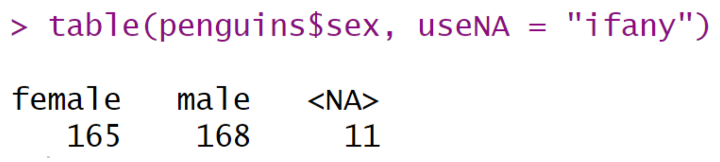
Example 5: `table(penguins$sex)`



```
> table(penguins$sex)

female   male 
   165    168 
```

`table(penguins$sex, useNA = "ifany")`



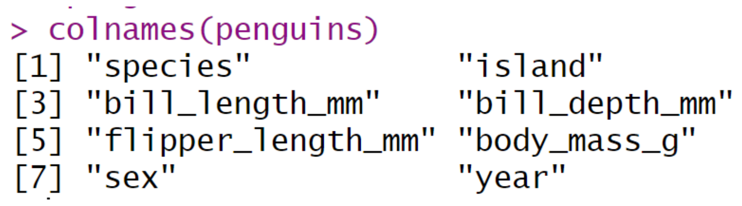
```
> table(penguins$sex, useNA = "ifany")

female   male   <NA> 
   165    168     11 
```

- `colnames()`: Display the structure of the data

Command: `colnames(name_of_dataframe)`

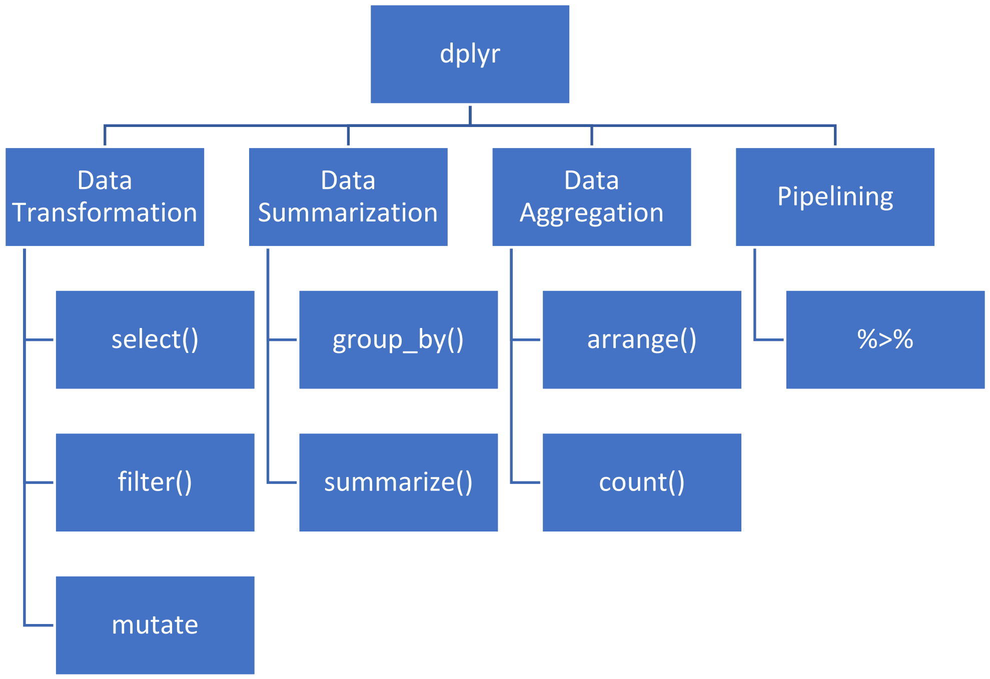
Example 6: `colnames(penguins)`



```
> colnames(penguins)
[1] "species"           "island"           
[3] "bill_length_mm"    "bill_depth_mm"    
[5] "flipper_length_mm" "body_mass_g"      
[7] "sex"               "year"             
```

dplyr Package

dplyr: provides a powerful and efficient toolkit for data manipulation in R.



- ***Data Transformation***
  - ✓ `select()`: This function is used to select columns from a data frame based on their names.
  - ✓ `filter()`: It's used to filter rows based on specified conditions.
  - ✓ `mutate()`: This function adds new columns or modifies existing ones, creating a transformed version of the data.
- ***Data Summarization***
  - ✓ `group_by()`: This function is used to group data by one or more variables.
  - ✓ `summarize()`: It's used in combination with group_by() to compute summary statistics for each group.
- ***Data Aggregation***
  - ✓ `arrange()`: This function orders rows based on specified variables, allowing ascending or descending order.
  - ✓ `count()`: It's used to count the occurrences of unique combinations of variables.
- ***Pipelining*** (`%>%` Operator): allows you to chain together multiple operations, to improving code readability and making it easier to follow the flow of transformations.

Command Illustration

```
new_dataframe_name <- dataframe_name %>%
     select(column_name1) %>%
     select(-column_name2)
```

`select()` Function

Illustration_Data

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

Different ways to use `` `select()` ``:

1. Choose specific column name from the data frame

Example 7: Select columns "Age" and "total_Income"

```
example_7 <- Illustration_Data %>%
     select(Age, total_Income)
```

Output: The output is a dataframe that looks like this

Age	total_Income
18	18000
25	25000
30	30000
40	40000
45	45000

2. Select Columns by Name Patterns: You can use special helper functions like `` `starts_with()` ``, `` `ends_with()` ``, `` `contains()` ``, `` `matches()` ``, and `` `everything()` `` to select columns based on their names.

Example 8: Select columns that start with "var"

```
example_8 <- Illustration_Data %>%
     select(starts_with(“var”))
```

Output: The output is a dataframe that looks like this

var_1	var_2	var_3
apple	carrots	elephant
grapes	carrots	tiger
bananas	carrots	lion
peaches	carrots	rabbit
bananas	carrots	shark

Example 9: Select columns that contains "total" in their names

```
example_9 <- Illustration_Data %>%
     select(contains("total"))
```

Output: The output is a dataframe that looks like this

total_Income	cat_total
18000	0
25000	1
30000	2
40000	1
45000	1

3. Exclude Columns: To exclude specific columns, you can use the "`-`" (minus) sign before the column name.

Example 10: Exclude columns "honesty" and "zipcode"

```
example_10 <- Illustration_Data %>%
     select(-honesty, -zipcode)
```

Output: The output is a dataframe that looks like this

Name	Age	total_Income	var_1	var_2	var_3	cat_total
Val	18	18000	apple	carrots	elephant	0
Derek	25	25000	grapes	carrots	tiger	1
Whitney	30	30000	bananas	carrots	lion	2
Sasha	40	40000	peaches	carrots	rabbit	1
Daniella	45	45000	bananas	carrots	shark	1

4. Select Columns by Index and Range: you can use a numeric vector to select the columns according to their index position.

Example 11: Exclude columns "honesty" and "zipcode"

```
example_11 <- Illustration_Data %>%
     select(c(1,3,5,6))
```

Output: The output is a dataframe that looks like this

Name	total_Income	var_2	var_3
Val	18000	carrots	elephant
Derek	25000	carrots	tiger
Whitney	30000	carrots	lion
Sasha	40000	carrots	rabbit
Daniella	45000	carrots	shark

Data Wrangling – Part 2

`filter()` Function

Filter Function - `filter()`: when filtering think about filtering ROWS. Filter uses Boolean logic.

Command Illustration

```
new_dataframe_name <- dataframe_name %>%
     filter(boolean expression using the name of a column)
```

For the illustration examples, assume the dataframe is the following:

Illustration_Data

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

Ways to filter:

1. ***Simple Conditions (one boolean expression) -*** For example: Greater than (`>`), less than (`<`), or equal to (`==`)

Example 1: Select rows where "Age" is greater than 30

```
example_1 <- Illustration_Data %>%
  filter(Age > 30)
```

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

Example 2: Select rows with "agree" in the "honesty" column

```
example_2 <- Illustration_Data %>%
  filter(honesty == "agree")
```

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

2. ***Multiple Conditions -*** you can combine conditions using logical operators like `&` (AND) and `|` (OR).

Example 3: Select rows where "Age" is greater than 30 and "total_Income" is less than 50000

```
example_3 <- Illustration_Data %>%
  filter(Age > 30 & total_Income < 50000)
```

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

Example 4: Select rows where "Age" is greater than 20 and "total_Income" is less than 30000 and zipcode is equal to 60073

```
example_4 <- Illustration_Data %>%
  filter(Age > 20 & total_Income < 30000 & zipcode == 60073)
```

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1

Example 5: Select rows where "Age" is greater than 30 or "total_Income" is greater than 20000

```
example_5 <- Illustration_Data %>%
  filter(Age > 30 | total_Income > 20000)
```

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

3. ***Exclusion -*** to exclude certain rows, you can use the `!=` operator (not equal to).

Example 6: Exclude rows with "zipcode" equal to 60111

```
example_6 <- Illustration_Data %>%
  filter(zipcode != 60111)
```

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

4. ***Filter rows based on vector of conditions -*** The `%in%` operator is useful for filtering rows with values in a specified vector.

Example 7: Select rows where "var_1" is either "bananas" or "grapes"

```
example_7 <- Illustration_Data %>%
  filter(var_1 %in% c("bananas", "grapes"))
```

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

Data Wrangling – Part 3

`mutate()` Function

`mutate()`: Creates a new column typically based on computations related to other columns in the dataset.

Command Illustration

```
new_dataframe_name <- dataframe_name %>%
     mutate(new_column_name = computation you want to do)
```

For the illustration examples, assume the dataframe is the following:

Illustration_Data

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

Example 1: Add a new variable "income_per_month" calculated from "total_Income"

```
example_1 <- Illustration_Data %>%
  mutate(income_per_month = total_Income / 12)
```

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total	
Val	18	18000	apple	carrots	elephant	60001	agree	0	
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1	
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2	
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1	
Daniella	45	45000	bananas	carrots	shark	60155	agree	1	

We can use other functions (like `ifelse`) within mutate to help us make a new variable.

Example 2: Create a new variable called "status" based on "age"

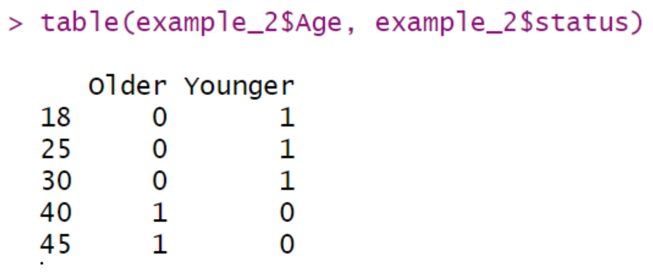
```
example_2 <- Illustration_Data %>%
  mutate(status = ifelse(Age > 30, "Older", "Younger"))
```

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total	
Val	18	18000	apple	carrots	elephant	60001	agree	0	
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1	
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2	
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1	
Daniella	45	45000	bananas	carrots	shark	60155	agree	1	

You can use the table function with two columns to count how many people fall into each category.

```
table(example_2$Age, example_2$status)
```



```
> table(example_2$Age, example_2$status)

     Older Younger
  18     0       1
  25     0       1
  30     0       1
  40     1       0
  45     1       0
```

If you have multiple condition you can use the `case_when()` function and list out your possible options.

```
    case_when(
       boolean expression ~ value_1,
       boolean expression ~ value_2,
       ...,
       TRUE ~ default_value
     )
```

Example 3: Create a new variable "group" based on "age". If the person's age is smaller than 30, they are "Young", if their age is between 30 and 40 (inclusive), then they are "Middle-Aged", if their age is greater than 40, then they are "old".

```
example_3 <- Illustration_Data %>%
  mutate(group = case_when(
     Age < 30 ~ "Young",
     Age >= 30 & Age <= 40 ~ "Middle-Aged",
     Age > 40 ~ "Old"
  ))
```

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total	
Val	18	18000	apple	carrots	elephant	60001	agree	0	
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1	
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2	
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1	
Daniella	45	45000	bananas	carrots	shark	60155	agree	1	

Example 4: Create a new variable "group" based on "age". If the person's age is smaller than 30, they are "Young", if their age is between 30 and 40 (inclusive), then they are "Middle-Aged", anything else, simply have it be "No Category".

```
example_4 <- Illustration_Data %>%
  mutate(group = case_when(
     Age < 30 ~ "Young",
     Age >= 30 & Age <= 40 ~ "Middle-aged",
     TRUE ~ "No Category"
  ))
```

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total	
Val	18	18000	apple	carrots	elephant	60001	agree	0	
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1	
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2	
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1	
Daniella	45	45000	bananas	carrots	shark	60155	agree	1	

Example 5: You can use the mutate function to change the variable type. The variable "total_Income" is numeric. Change it to character.

```
class(Illustration_Data$total_Income)
```

Output:

Code:

```
class(Illustration_Data$total_Income)
```

Output:

Data Wrangling – Part 4

Data Summary: Using `` `group_by()` `` and `` `summarise()` `` together allows you to efficiently compute summary statistics, aggregations, or any other computations of data based on different groups defined by one or more variables.

`group_by()`: is used to group a data frame by one or more variables. This creates a "grouped" data frame where subsequent operations are performed within each group separately. This works best with categorical variables or factor variables. Using `group_by()` in it's own doesn't change the "look" of the data.

`summarise()`: is used to compute summary statistics or other values for each group. It condenses the grouped data into a single row per group, summarizing the specified variables.

Command Illustrations

```
new_dataframe_name <- dataframe_name %>%
     summarise(new_column_name = function_name(column_name2))
```

If using group_by:

```
new_dataframe_name <- dataframe_name %>%
     group_by(column_name1) %>%
     summarise(new_column_name = function_name(column_name2))
```

For the illustration examples, assume the dataframe is the following:

Illustration_Data

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

Example 1: Compute mean and median of total_Income

```
example_1 <- Illustration_Data %>%
      summarise(mean_income = mean(total_Income),
                median_income = median(total_Income))
```

Output will be a dataframe that looks like:

Example 2: Compute mean and median of total_Income, grouped by favorite fruit (var_1)

```
example_2 <- Illustration_Data %>%
     group_by(var_1) %>%
     summarise(mean_income = mean(total_Income),
               median_income = median(total_Income))
```

Output will be a dataframe that looks like:

Example 3: Compute sum and mean of total_Income, grouped by honesty (var_1) and cat_total

```
example_3 <- Illustration_Data %>%
     group_by(honesty, cat_total) %>%
     summarise(sum_income = sum(total_Income),
               mean_income = mean(total_Income))
```

Output will be a dataframe that looks like:

Example 4: Compute the number of pro dancers per honesty

```
example_4 <- Illustration_Data %>%
      group_by(honesty) %>%
      summarise(Num_Pro = n())
```

Output will be a dataframe that looks like:

Data Wrangling – Part 5

`arrange()`: is used to reorder the rows of a data frame based on one or more variables. The default order is from smallest to largest (or alphabetical for strings). To change the order from largest to smallest use "desc()".

Command Illustration

```
new_dataframe_name <- dataframe_name %>%
     arrange(column_name)
```

For the illustration examples, assume the dataframe is the following:

Illustration_Data

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

Example 1: Arrange data by "cat_total" in ascending order (smallest to largest)

```
example_1<-Illustration_Data %>%
  arrange(cat_total)
```

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2

Example 2: Arrange data by "cat_total" in descending order (largest to smallest).

```
example_2<-Illustration_Data %>%
  arrange(desc(cat_total))
```

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1
Val	18	18000	apple	carrots	elephant	60001	agree	0

Example 3: Arrange data in alphabetical order by name.

```
example_3<-Illustration_Data %>%
  arrange(Name)
```

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Daniella	45	45000	bananas	carrots	shark	60155	agree	1
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Val	18	18000	apple	carrots	elephant	60001	agree	0
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2

Example 4: Count the number of occurrences each fruit had in "var_1"

```
example_4<-Illustration_Data %>%
  group_by(var_1) %>%
  summarize(N = n())
```

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

Output will be a dataframe that looks like:

Example 5: Arrange by honesty and within category of honesty, arrange by cat_total.

```
example_5 <- Illustration_Data %>%
  arrange(honesty, cat_total)
```

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Daniella	45	45000	bananas	carrots	shark	60155	agree	1

Output will be a dataframe that looks like:

Name	Age	total_Income	var_1	var_2	var_3	zipcode	honesty	cat_total
Val	18	18000	apple	carrots	elephant	60001	agree	0
Daniella	45	45000	bananas	carrots	shark	60155	agree	1
Derek	25	25000	grapes	carrots	tiger	60073	disagree	1
Sasha	40	40000	peaches	carrots	rabbit	60111	disagree	1
Whitney	30	30000	bananas	carrots	lion	60109	disagree	2