Class 6 Data Wrangling with R & Descriptive Analytics

Dr. Wei Miao

UCL School of Management

October 14, 2026

Section 1

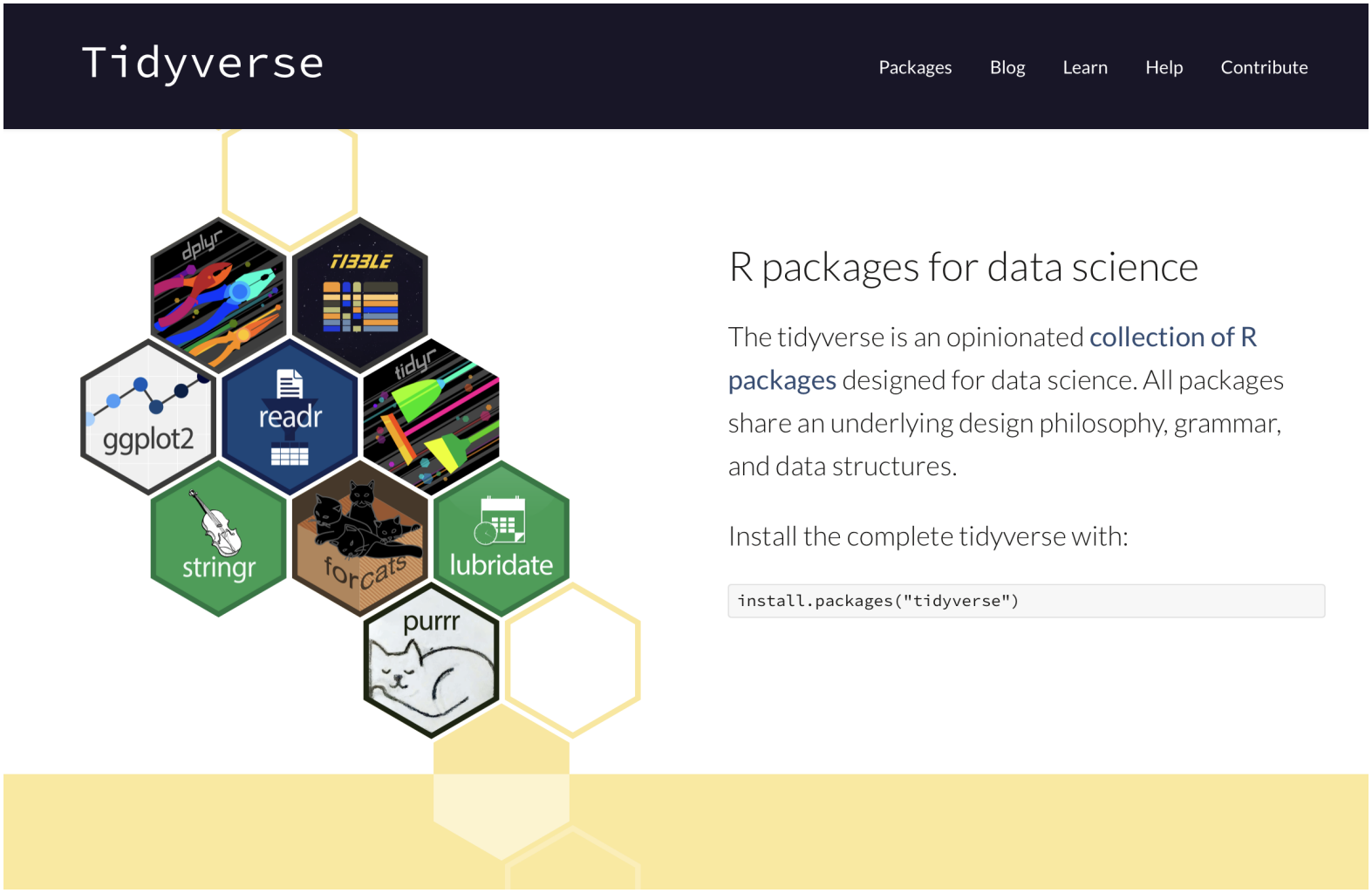
Data Wrangling with R

Data Frame Basics

- A data frame is the R object that we will deal with most of the time in the MSc programme. You can think of a `data.frame` as a spreadsheet.
  - Each row stands for an **observation**; usually a record for a customer.
  - Each column stands for a **variable**; each column should have a unique name.
  - Each column must contain the same data type, but different columns can store different data types.

Tidyverse in R

- The `tidyverse` is a collection of R packages designed for data science. It includes packages for data manipulation, visualisation, and more.



Install and Load the dplyr package

- In R, we use the dplyr package for data cleaning and manipulation.[1]

```
pacman::p_load(dplyr)
```

- Load a CSV-format dataset called data_full using read.csv()

```
data_full <- read.csv("https://www.dropbox.com/scl/fi/2q7ppqtyca0pd3j48
```

- To browse the whole dataset, simply click the object in the Environment pane.

[1] pacman is a package management tool that makes our lives easier by loading multiple packages at once.

First Look at the Dataset

- Which variables does the dataset have? What are the data types of each variable?

```
glimpse(data_full)
```

```
Rows: 2,000
Columns: 22
$ ID                 <int> 5524, 2174, 4141, 6182, 5324, 7446, 965, 6177, 485~
$ MntWines           <int> 635, 11, 426, 11, 173, 520, 235, 76, 14, 28, 5, 6,~
$ MntFruits          <int> 88, 1, 49, 4, 43, 42, 65, 10, 0, 0, 5, 16, 61, 2, ~
$ MntMeatProducts    <int> 546, 6, 127, 20, 118, 98, 164, 56, 24, 6, 6, 11, 4~
$ MntFishProducts    <int> 172, 2, 111, 10, 46, 0, 50, 3, 3, 1, 0, 11, 225, 3~
$ MntSweetProducts   <int> 88, 1, 21, 3, 27, 42, 49, 1, 3, 1, 2, 1, 112, 5, 1~
$ MntGoldProds       <int> 88, 6, 42, 5, 15, 14, 27, 23, 2, 13, 1, 16, 30, 14~
$ NumDealsPurchases  <int> 3, 2, 1, 2, 5, 2, 4, 2, 1, 1, 1, 1, 1, 3, 1, 1, 3,~
$ NumWebPurchases    <int> 8, 1, 8, 2, 5, 6, 7, 4, 3, 1, 1, 2, 3, 6, 1, 7, 3,~
$ NumCatalogPurchases <int> 10, 1, 2, 0, 3, 4, 3, 0, 0, 0, 0, 0, 4, 1, 0, 6, 0~
$ NumStorePurchases  <int> 4, 2, 10, 4, 6, 10, 7, 4, 2, 0, 2, 3, 8, 5, 3, 12,~
$ NumWebVisitsMonth  <int> 7, 5, 4, 6, 5, 6, 6, 8, 9, 20, 7, 8, 2, 6, 8, 3, 8~
$ Complain           <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
$ Response           <int> 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,~
$ Year_Birth         <int> 1957, 1954, 1965, 1984, 1981, 1967, 1971, 1985, 19~
$ Education          <chr> "Graduation", "Graduation", "Graduation", "Graduat~
$ Marital_Status     <chr> "Single", "Single", "Together", "Together", "Marri~
$ Income             <int> 58138, 46344, 71613, 26646, 58293, 62513, 55635, 3~
$ Kidhome            <int> 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,~
$ Teenhome           <int> 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,~
$ Dt_Customer        <chr> "04/09/2012", "08/03/2014", "21/08/2013", "10/02/2~
$ Recency            <int> 58, 38, 26, 26, 94, 16, 34, 32, 19, 68, 11, 59, 82~
```

First Few Rows of the Dataset

- We can use `slice_head()` to view the first few rows of the dataset. Similarly, `slice_tail()` can be used to view the last few rows of the dataset.

```
slice_head(data_full, n = 3)
```

```
    ID MntWines MntFruits MntMeatProducts MntFishProducts MntSweetProducts
1 5524      635        88              546             172                88
2 2174       11         1                6               2                 1
3 4141      426        49              127             111                21
  MntGoldProds NumDealsPurchases NumWebPurchases NumCatalogPurchases
1           88                 3               8                  10
2            6                 2               1                   1
3           42                 1               8                   2
  NumStorePurchases NumWebVisitsMonth Complain Response Year_Birth  Education
1                 4                 7        0        1       1957 Graduation
2                 2                 5        0        0       1954 Graduation
3                10                 4        0        0       1965 Graduation
  Marital_Status Income Kidhome Teenhome Dt_Customer Recency
1         Single  58138       0        0  04/09/2012      58
2         Single  46344       1        1  08/03/2014      38
3       Together  71613       0        0  21/08/2013      26
```

Extract a Single Column

- We can use the $ operator to extract a single column from a data frame.
- For example, `data_full$Income` extracts the `Income` column from the `data_full` data frame, returning a vector of income values.

```
# extract the Income column and show the first 5 values
data_full$Income[1:5]
```

```
[1] 58138 46344 71613 26646 58293
```

```
# compute the mean of the Income column, ignoring missing values
mean(data_full$Income, na.rm = TRUE)
```

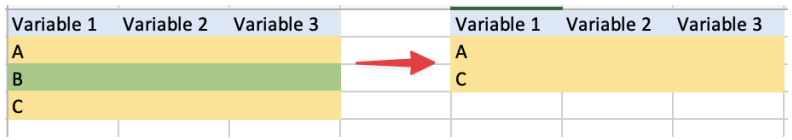
```
[1] 52139.7
```

Common Data Wrangling Operations

- **Filter rows** (`filter`)
- **Sort rows** (`arrange`)
- **Select columns** (`select`)
- **Generate new columns** (`mutate`)
- **Group-wise aggregation** (`group_by`)
- Merge datasets (`join`)

Subset Rows Based on Conditions: `filter`

- We can use `filter()` to select rows that meet certain logical criteria.



- **Important**: To keep the new subset of data in RStudio, assign it to a new object.

Subset Rows Based on Conditions: `filter`

Example: From `data_full`, find customers who are single

```
# keep only single customers
filter(data_full, Marital_Status == "Single" )
```

The Pipe Operator

Pipe Operator

%>% passes the **object in front** as the **first argument** of the **subsequent function**.[2]

input %>% functionName(, ...)

other inputs

functionName(input, ...)

other inputs

- The tidyverse is designed to work with the pipe operator, using a consistent syntax.

[2]Starting from R 4.1, base R introduced the native pipe operator |>

Example of the Pipe Operator

```
# without using pipe
filter(data_full, Marital_Status == 'Single')

# with pipe
data_full %>% filter(Marital_Status == 'Single')
```

Why Do We Need Pipe Operator for Data Wrangling?

- **Exercise**: find **single** customers who have a **PhD** without using the pipe.

```
# Write the code below
```

Why Do We Need Pipe Operator for Data Wrangling?

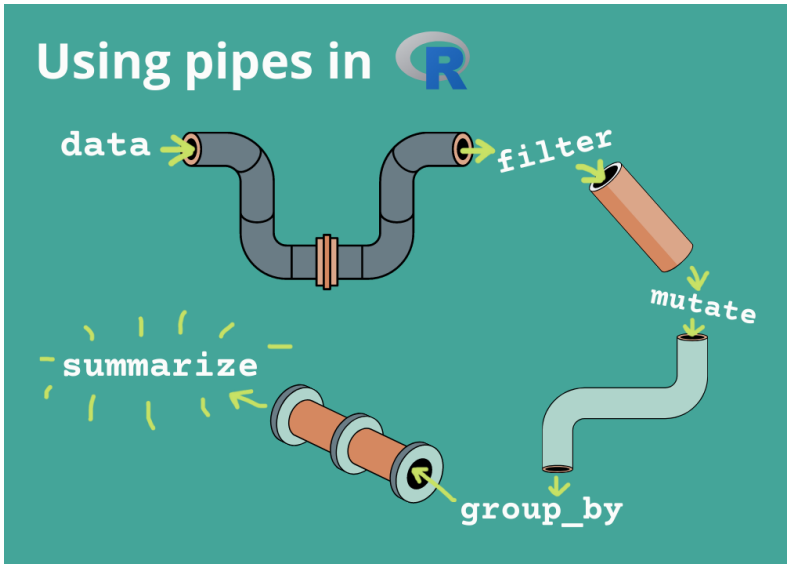
- **Exercise**: find **single** customers who have a **PhD** using the pipe.

```
data_full_single_PhD <- data_full %>%
  filter(Marital_Status == 'Single') %>%
  filter(Education == 'PhD')

## You can even continue with more filter steps with more pipe operator
```

Why Do We Need Pipe Operator for Data Wrangling?

- The pipe works like a conveyor belt in a factory, passing the intermediate outputs from the previous data wrangling step to the next step for further processing until you finish your data wrangling task.



- This makes your code more readable and easier to debug. You can chain multiple (as many as) data wrangling steps together in a single pipeline.

Sort Rows: `arrange`

- `arrange()` orders the rows by the values of selected columns.
- Ascending order by default; for descending order, put a minus sign in front of the variable.
- Allows multiple sorting variables, separated by a comma.
- **Example**: sort customers based on income in descending order.

```
data_full %>%
  arrange(-Income)
```

- **Exercise**: sort customers based on income in descending order and age in ascending order.

Generate New Variables: `mutate`

- `mutate()` generates new variables in the dataset while preserving existing variables.
- **Example**: create a new variable named `Age` from `Year_Birth`.

```
data_full %>%
  mutate(Age = 2025 - Year_Birth)
```

- **Exercise**: create a new variable named `totalkids`, which is the sum of `Kidhome` and `Teenhome`.

Aggregation by Groups: group_by

- group_by() allows us to aggregate data by group and compute statistics for each group.

```
# group by marital status
data_full %>%
    group_by(Marital_Status)
```

- Internally, the dataset is already grouped based on the specified variable(s).

Aggregation by Groups: `group_by() %>% summarise()`

- The `summarise()` function is typically used after `group_by()` to create a new data frame of summary statistics for each group.
- It collapses each group into a single row.
- Inside `summarise()`, you can compute various summary statistics (e.g., `mean()` for mean, `sum()` for sum, `n()` for count, `sd()` for standard deviation) for each group. The result is a new data frame with one row per group, containing the grouping variables and the calculated summary statistics.

```
# compute the average income for each marital status group
data_full %>%
  group_by(Marital_Status) %>%
  summarise(avg_income = mean(Income, na.rm = TRUE)) %>%
  ungroup()
```

- What if you replace `summarise()` with `mutate()`?

Comparison of `summarise()` and `mutate()`

- `mutate()` will add a new column to the original data frame. The new column will contain the group-wise calculation, repeated for each row within the group. The number of rows in the data frame remains unchanged.
- In contrast, `summarise()` collapses the data frame to one row per group, containing only the grouping variables and the newly created summary statistics.

Aggregation by Groups: `group_by()` Multiple Groups

- You can also group by multiple variables. For example, we can compute the average income for each combination of `Marital_Status` and `Education`.

```
# compute the average income for each marital, education group
data_full %>%
  group_by(Marital_Status,Education) %>%
  summarise(avg_income = mean(Income, na.rm = TRUE)) %>%
  ungroup() %>%
  slice_head(n = 5)
```

Section 2

Data Cleaning

Missing Values

- In R, missing values are represented by the symbol `NA` (i.e., not available).
- Most statistical models cannot handle missing values, so we need to deal with them in R.
- If there are just a few missing values: remove them from analysis.
- If there are many missing values: need to collect better data or replace them with appropriate values:
  - mean/median/statistical imputation

Outliers

- **Outliers** are data points that are significantly different from other data points in the dataset, such as unusually large and small values.
- **Winsorisation** is a common method to deal with outliers. It replaces the extreme values with the nearest non-extreme value, usually the 99th or 1st percentile (or other thresholds as appropriate).

Section 3

Descriptive Analytics

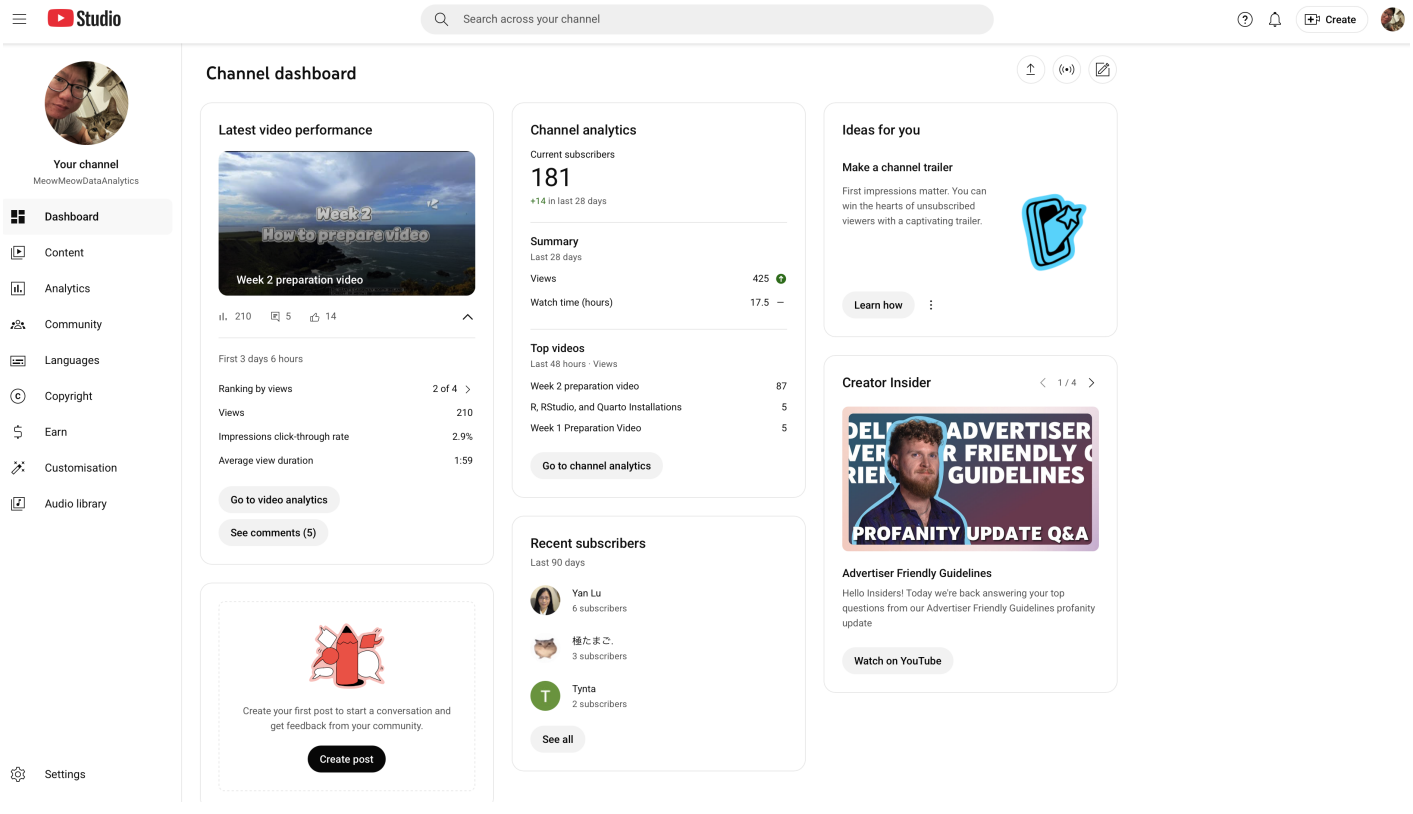
Two Major Tasks of Descriptive Analytics

- You can think of descriptive analytics as **creating a dashboard** to display the key information you would like to know for your business. For instance:

1. Describe the data for your business purposes
   - "How much do our customers spend each month on average?"
   - "What percentage of our customers are unprofitable?"
   - "What is the difference between the retention rates across different demographic groups?"
2. Conduct statistical tests (such as t-tests) for hypothesis testing.

- Is there a significant difference in the average spending between different age/gender groups?
  - Based on our test mailing, can we conclude that ad-copy A works better than ad-copy B?

Example of Descriptive Analytics Dashboard



Summary Statistics

- **Summary statistics** are used to summarise a set of observations, in order to communicate the largest amount of information as simply as possible.
- There are two main types of summary statistics used in evaluation:
  - **measures of central tendency**: number of observations, mean, min, 25th percentile, median, 75th percentile, max, etc.
  - **measures of dispersion:** range and standard deviation.

Summary Statistics with R

- In R, a powerful package to report summary statistics is called `modelsummary`.
- `datasummary_skim()` is a shortcut to compute basic summary statistics.
- For more features, refer to the package tutorial here

```
pacman::p_load(modelsummary)
## Summary statistics for numeric variables
data_full %>%
  datasummary_skim(type = "numeric")

## Summary statistics for categorical variables
data_full %>%
  datasummary_skim(type = "categorical")
```

Section 4

M&S Descriptive Analytics

M&S Descriptive Analytics

Let's move on to the Quarto document to see how to apply descriptive analytics to the M&S dataset.

After-Class

- (essential) Cheatsheet for `dplyr`. This cheatsheet provides a quick reference for the most commonly used functions in the `dplyr` package. It's very important to familiarise yourself with these functions as you will use them a lot in your future projects.
- (optional) Complete the after-class exercise for Week 3. If you still have time, you can also complete the DataCamp exercise on the dplyr package. The link is here.