Data Wrangling I

What is Data Wrangling?

Data wrangling makes it easier to get your data into R in a useful form for visualization and modelling. Here, we discuss the following three tools.

- **tibbles** the variant of the data frame.
- **Data import** in rectangular formats.
- **tidy data** a consistent way of storing your data that makes transformation, visualization, and modelling easier.

Tibbles

Tibbles are data frames, which makes easier for data wrangling. Tibbles can be created using the tibble package. Tibbles have an enhanced print() method which can be used with large datasets containing complex objects.

```{r}
library(tibble)
as_tibble(iris)
```

A tibble: 150 × 5

Sepal.Length <dbl>	Sepal.Width <dbl>	Petal.Length <dbl>	Petal.Width <dbl>	Species <fctr>
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa

You can create a new tibble from individual vectors with tibble() function.

```{r}
library(tibble)
tibble(
  x = 1:5,
  y = 1,
  z = x ^2 +y
)
```

A tibble: 5 × 3

x <int>	y <dbl>	z <dbl>
1	1	2
2	1	5
3	1	10
4	1	17
5	1	26

Tibbles vs. data.frame

Tibbles shows only the first 10 rows, and all the columns fit on screen, which makes it much easier to work with large data. In addition to its name, each column reports its data type.

```{r}
tibble(
  a = lubridate::now(),
  b = lubridate::today(),
  c = 1:1e3,
  d = runif(1e3),
  e = sample(letters, 1e3, replace = TRUE)
)
```

A tibble: 1,000 × 5

a <S3: POSIXct>	b <date>	c <int>	d <dbl>	e <chr>
2023-08-05 10:24:25	2023-08-05	1	0.981967779	l
2023-08-05 10:24:25	2023-08-05	2	0.932754264	z
2023-08-05 10:24:25	2023-08-05	3	0.566451138	h
2023-08-05 10:24:25	2023-08-05	4	0.104079346	q
2023-08-05 10:24:25	2023-08-05	5	0.696408494	e
2023-08-05 10:24:25	2023-08-05	6	0.465676787	f
2023-08-05 10:24:25	2023-08-05	7	0.549894462	h
2023-08-05 10:24:25	2023-08-05	8	0.940030203	r
2023-08-05 10:24:25	2023-08-05	9	0.056910700	r
2023-08-05 10:24:25	2023-08-05	10	0.317450699	x

Consider the below flights data set which shows on-time data for all flights that departed NYC in 2013, in nycflights13 package. If you want to see only 5 observations, use the below code. The option width = Inf shows all variables.

```{r}
library(nycflights13)
nycflights13::flights %>%
  print(n = 5, width = Inf)
```

A tibble: 336,776 × 19

year <int>	month <int>	day <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	carrier <chr>
2013	1	1	517	515	2	830	819	11	UA
2013	1	1	533	529	4	850	830	20	UA
2013	1	1	542	540	2	923	850	33	AA
2013	1	1	544	545	-1	1004	1022	-18	B6
2013	1	1	554	600	-6	812	837	-25	DL
2013	1	1	554	558	-4	740	728	12	UA
2013	1	1	555	600	-5	913	854	19	B6
2013	1	1	557	600	-3	709	723	-14	EV
2013	1	1	557	600	-3	838	846	-8	B6
2013	1	1	558	600	-2	753	745	8	AA

1-10 of 336,776 rows | 1-10 of 19 columns

Previous 1 2 3 4 5 6 ... 100 Next

To see the data in the default output of a tibble, use the below codes.

```{r}
options(tibble.width = Inf)
nycflights13::flights %>%
  print()
```

A tibble: 336,776 × 19

year <int>	month <int>	day <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	carrier <chr>
2013	1	1	517	515	2	830	819	11	UA
2013	1	1	533	529	4	850	830	20	UA
2013	1	1	542	540	2	923	850	33	AA
2013	1	1	544	545	-1	1004	1022	-18	B6
2013	1	1	554	600	-6	812	837	-25	DL
2013	1	1	554	558	-4	740	728	12	UA
2013	1	1	555	600	-5	913	854	19	B6
2013	1	1	557	600	-3	709	723	-14	EV
2013	1	1	557	600	-3	838	846	-8	B6
2013	1	1	558	600	-2	753	745	8	AA

1-10 of 336,776 rows | 1-10 of 19 columns

Previous 1 2 3 4 5 6 ... 100 Next

Data import

To read plain-text rectangular files into R, use the readr package, which is part of the core tidyverse.

- read_csv() reads comma delimited files, read_csv2() reads semicolon separated files, read_tsv() reads tab delimited files, and read_delim() reads in files with any delimiter.
- read_fwf() reads fixed width files. You can specify fields either by their widths with fwf_widths() or their position with fwf_positions(). read_table() reads a common variation of fixed width files where columns are separated by white space.

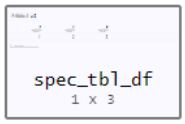
- read_log() reads Apache style log files. (But also check out webreadr which is built on top of read_log() and provides many more helpful tools.)

Suppose you have a csv file named heights in a folder called data. Then, to read it use

heights<-read_csv("data/heights.csv")
If some meta data included in the first two lines of the data set, and you want to skip those two lines, use

heights<-read_csv("data/heights.csv", skip=2)

```
  {r}
library(readr)
read_csv("The first line of metadata
  The second line of metadata
  x,y,z
  1,2,3", skip = 2)
```



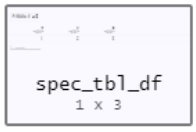
A tibble: 1 × 3

x	y	z
<dbl>	<dbl>	<dbl>
1	2	3

1 row

If there is a comment in the data set, use the below codes.

```{r}
read_csv("# A comment I want to skip
  x,y,z
  1,2,3", comment = "#")
```



A tibble: 1 × 3

x <dbl>	y <dbl>	z <dbl>
1	2	3

1 row

If the variable names are not included in the data set, you can specify them and read the data as heights<-read_csv("data/heights.csv", col_names = c("x", "y", "z"))

If the missing values are indicated as "." in the data set, you can change them to NA as below: heights<-read_csv("data/heights.csv", NA= ".")

If you data contains other characters such as $100, 20% or etc, then to convert them to data use parse_number() function.

```{r}
parse_number("$100")
```

```
[1] 100
```

```{r}
parse_number("20%")
```

```
[1] 20
```

```{r}
parse_number("It cost $123.45")
```

```
[1] 123.45
```

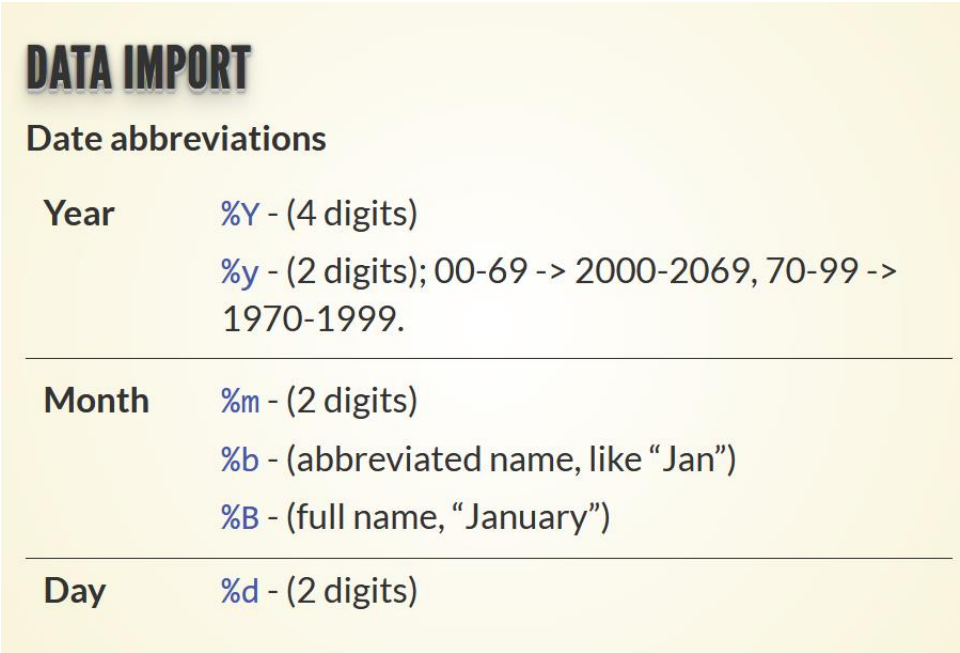
```{r}
parse_number("$123,456,789")
```

```
[1] 123456789
```

To write dates in a given format, you can use parse_date() function.

```
```{r}
parse_date("01/02/15", "%m/%d/%y")
parse_date("01/02/15", "%d/%m/%y")
parse_date("01/02/15", "%y/%m/%d")
```
```

```
[1] "2015-01-02"
[1] "2015-02-01"
[1] "2001-02-15"
```

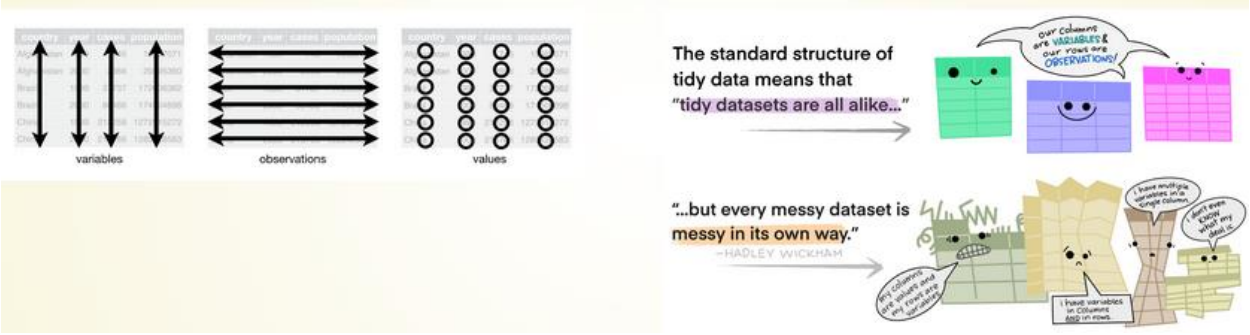


Tidy Data

In tidy data

- each variable forms a column
- each observation forms a row
- each cell is a single measurement

When working with tidy data, we can use the same tools in similar ways for different data sets. dplyr, ggplot2, and all the other packages in the tidyverse are designed to work with tidy data.



Advantages of your data to be tidy?

- The general advantage is that you can select one consistent way to store your data. If you have a consistent data structure, it's easier to learn the tools that work with it since they have an underlying uniformity.
- When variables are in columns, you can use the vectorised nature of R. Most of the built-in R functions work with vectors of values.

Run the following codes and identify which data set/s is/are tidy.

```{r}
library(tidyverse)
table1
table2
table3
table4a
table4b
```

R Console | tbl_df 6 x 4 | tbl_df 12 x 4 | tbl_df 6 x 3 | tbl_df 3 x 3 | tbl_df 3 x 3

A tibble: 6 × 4

country <chr>	year <dbl>	cases <dbl>	population <dbl>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

6 rows

Run the following codes.

```{r}
# Compute rate per 10,000
library(dplyr)
table1 %>%
  mutate(rate = cases / population * 10000)
```

A tibble: 6 × 5

country <chr>	year <dbl>	cases <dbl>	population <dbl>	rate <dbl>
Afghanistan	1999	745	19987071	0.372741
Afghanistan	2000	2666	20595360	1.294466
Brazil	1999	37737	172006362	2.193930
Brazil	2000	80488	174504898	4.612363
China	1999	212258	1272915272	1.667495
China	2000	213766	1280428583	1.669488

6 rows

```{r}
# Compute cases per year
table1 %>%
  count(year, wt = cases)
```

A tibble: 2 × 2

year <dbl>	n <dbl>
1999	250740
2000	296920

If we have untidy data, how do we make them tidy?

To do this, you can use the functions pivot_longer() and pivot_wider() in tidyr.

```
library(tidyverse)
table4a
```

A tibble: 3 × 3

country <chr>	1999 <dbl>	2000 <dbl>
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

3 rows

Here, the column names 1999 and 2000 represent values of the year variable, and the values of these columns represent values of the cases variable. Note that each row represents two observations, but not one.

Since **one variable spreads across multiple columns** in table4a , we use pivot_longer() function to make it tidy as below.

```
library(dplyr)
table4a %>%
  pivot_longer(c(`1999`, `2000`), names_to = "year", values_to = "cases")
```

A tibble: 6 × 3

country <chr>	year <chr>	cases <dbl>
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

6 rows

In table2 data set, **one observation is scattered across multiple rows**. Therefore, we have to use pivot_wider() function to make it tidy.

```{r}
library(tidyverse)
table2
```

A tibble: 12 × 4

country <chr>	year <dbl>	type <chr>	count <dbl>
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272

1-10 of 12 rows Previous

```{r}
table2 %>%
    pivot_wider(names_from = type, values_from = count)
```

A tibble: 6 × 4

country <chr>	year <dbl>	cases <dbl>	population <dbl>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

6 rows

In the following data set, we can separate the rate variables to two variables. We separate the rate variables to cases and population variables.

```
library(tidyverse)
table3
```

A tibble: 6 × 3

country <chr>	year <dbl>	rate <chr>
Afghanistan	1999	745/19987071
Afghanistan	2000	2666/20595360
Brazil	1999	37737/172006362
Brazil	2000	80488/174504898
China	1999	212258/1272915272
China	2000	213766/1280428583

6 rows

```{r}
table3 %>%
  separate(rate, into = c("cases", "population"), convert = TRUE)
```

A tibble: 6 × 4

country <chr>	year <dbl>	cases <int>	population <int>
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Chunk 21 R

We can also use sep option with separate function to separate integers.

```{r}
table3 %>%
  separate(year, into = c("century", "year"), sep = 2)
```

A tibble: 6 × 4

country <chr>	century <chr>	year <chr>	rate <chr>
Afghanistan	19	99	745/19987071
Afghanistan	20	00	2666/20595360
Brazil	19	99	37737/172006362
Brazil	20	00	80488/174504898
China	19	99	212258/1272915272
China	20	00	213766/1280428583

6 rows

Using unite() function we can combine multiple columns into a single column.

Now, we use unite() function to rejoin the *century* and *year* columns that we created in the previous example.

```{r}
library(tidyverse)
table5<-table3 %>%
  separate(year, into = c("century", "year"), sep = 2)
table5 %>%
  unite(new, century, year, sep = "")# Run without sep="" option, and see the difference
```

A tibble: 6 × 3

country <chr>	new <chr>	rate <chr>
Afghanistan	1999	745/19987071
Afghanistan	2000	2666/20595360
Brazil	1999	37737/172006362
Brazil	2000	80488/174504898
China	1999	212258/1272915272
China	2000	213766/1280428583

6 rows

Tidy Data (Missing values)

A value can be missing in two ways, **Explicitly** (i.e. flagged with NA), **Implicitly** (i.e. simply not present in the data).

In this data set, the return for the fourth quarter of 2015 is explicitly missing, and the return for the first quarter of 2016 is implicitly missing.

```{r}
stocks <- tibble(
year= c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
qtr = c(   1,    2,    3,    4,    2,    3,    4),
return = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66)
)
stocks
```

A tibble: 7 × 3

year <dbl>	qtr <dbl>	return <dbl>
2015	1	1.88
2015	2	0.59
2015	3	0.35
2015	4	*NA*
2016	2	0.92
2016	3	0.17
2016	4	2.66

7 rows

To make missing values explicit in tidy data, we can use complete() function.

```{r}
stocks %>%
  complete(year, qtr)
```

A tibble: 8 × 3

year <dbl>	qtr <dbl>	return <dbl>
2015	1	1.88
2015	2	0.59
2015	3	0.35
2015	4	*NA*
2016	1	*NA*
2016	2	0.92
2016	3	0.17
2016	4	2.66

8 rows