

Preparatory Notes

Understanding External data

Data most times won't be recorded manually and when they are recorded. You'd rather not do so with R. why? It is an excruciatingly difficult process to create data with R. It is actually way easier to record data with spreadsheets than to record it with R, and this is even when you use the `edit()` function.

Data Sources

When we want to import data into R, we need to know where to get it from. ***A data source is the place where the data we want to use originates from. This can be a physical or digital place*** A data source could be any of a live measurements from physical devices, a database, a flat file or plain-text file, scraped web data, or any of the innumerable static and streaming data services which abound across the internet. An example of a data source is data.gov, and [wikipedia](https://en.wikipedia.org) and these are web data. Another example that is actually close to you is your mobile phone. It's a mobile database holding contact information, music data, games, and so on.

Data Format

Data formats represent the form, structure and organization of data. It defines how information is stored, accessed, and interpreted. It's a standardized way to represent data, whether in files or databases, and is crucial for efficient data management and processing. This brings us to another aspect that is important to knowing how to handle data, and that's

file extension. If you've taken a closer look at the music/audio file on any of your device, you usually see a dot which separates the name of the file and a text which is usually fairly consistent depending on your file organization. This text is regarded to as the **file extension**. This is also true for your video, and picture files. For example, you could see the following extensions:

Table 1: Some of the common multimedia file format including audio, picture and video files.

Audio	Pictures	Video
.mp3	.png	.mp4
.aac	.jpeg	.mov
.flac	.gif	.avi
.wav	.webp	.web,
.ogg	.tiff	.mov

For data set, there are some common file format such as:

- flat file : .csv, .tsv, .txt, .rtf
- some web data format excluding those above: .html, .json, xml
- spreadsheet: .xlsx, .xls, .xlsm, .ods, .gsheet
- others: .shp, .hdsf, .sav, and many more.

Importing Data

Data is imported into R based on the file type you are dealing with. We will cover some of the file types you come across when working with data in R. We can use base R to import data, but we won't. Instead, we will make use of packages for our data importation. There will be little focus on base R, as its implementation isn't different from that of the external packages. Where necessary, both base R and the package implementation of data importation will be given.

Flat file data

Flat Files they are one of the most common forms in which data are stored. To import text files, we can use either of `data.table` or `readr`. Of course, let's not forget that base R's `util` can import flat files data too. There are still other packages that we can use to import flat file data, but, the goal is not to learn about the packages we can use to import data but how to import data. Firstly let's install `pacman` and use its `p_load()` function to import the packages we need:

```
install.packages("pacman")
```

`pacman` makes data import very easy. It throws an error when packages are not available. What makes easy to use is, you do not need to remember the quotation marks during package installation. It also have the `p_load()` function which is used to download, install, and load packages at once, performing the function of `install.packages`, and `library` at once.

```
library(pacman)
p_load(readr, data.table)
```

Without `pacman`, the steps would be:

```
install.packages(c("data.table", "readr")) # this can be further broken down
library(data.table)
library(readr)
```

This is what makes `pacman` shines to me. You could also run the below, if you do not want to load `pacman` itself but just make use of its function.

```
pacman::p_load(readr, data.table)
```

Now that we are done with installing and loading our packages, let's take a closer look at the different flat files. There are two things we should consider when dealing with flat files:

- The extension of the file.
- The structure of file when opened.

To some degree, the extension of your file could give you a hint on how to read it on how to read it on R. .csv, would need a csv file reading function, .txt, would need its own also. Notwithstanding, the internal structure is what makes reading a file easier. In flat files data, the data presented in a way that mimics standard spreadsheet table, with the difference being in how their recorded are separated. While a spreadsheet is having its rows and column clearly defined, flat files have its own separated using a consistent sign or symbol in the document. These signs and symbols are regarded to as **delimiters**. For example, Figure 1 have it's columns separated by commas, so the delimiter is a comma, thus the name comma separated values.

```
1 Timestamp,PM2.5,Temperature,Humidity,Soil_Moisture,Biodiversity_Index,Nutrient_Level,Water_Quality,Air_Quality_Index,Pollution_Level,Soil
2 2018-01-01 08:00:00,119.68396949346183,21.88582826784701,93.955602677343734,22.47978411388143,9,50,0,82.59492848705489,Low,
3 5.264882424242424,6.555424242424242,7,24.11972892727704,9.73133560475279,44.79405504497619,Ecologically Healthy
4 2018-01-01 01:00:00,74.72323651077937,19.07953674863838,54.298952029402614,23.980314321161206,9,0,0,127.41848403138505,Moderate,
5 6.107887490623757,7.5426076495366985,164.5849555273063,178.79360244394263,205.78701838141671,Ecologically Stable
6 2018-01-01 02:00:00,69.11417921854768,26.675868597849306,98.99122239709898,11.566319038501375,7,50,0,95.21541885818964,Low,
7 8.36157617261117,8.621085240118288,24.818365757128436,25.33288626712642,448.3867577921339,Ecologically Healthy
8 2018-01-01 03:00:00,69.11511471659003,20.178066328959515,36.41645978345206,36.144569499065035,10,0,0,85.53428670093444,Low,
9 7.92976578469351,7.421998509194271,248.72787073789508,58.94812792050839,359.2593827684935,Ecologically Healthy
10 2018-01-01 04:00:00,232.48572061153254,21.915745200239222,79.35562468027564,43.53254808905471,11,0,1,80.31495632293126,Low,
11 5.37841849344895,7.2318678336575895,271.0623375950484,106.11366338625126,118.3836604066327,Ecologically Critical
12 2018-01-01 05:00:00,143.3353115482111,15.021385179665616,96.33045929098851,37.930729162182296,12,0,0,102.10155132906004,Low,
13 5.579343943877647,7.229238575341893,280.21082134695894,120.85935856875368,84.70938242721665,Ecologically Healthy
14 2018-01-01 06:00:00,56.55390808704495,22.946270298507315,73.710380089984554,40.767319957446965,7,0,0,84.88673256374913,Low,
15 6.00130032423489,5.920145747029245,53.976623080252506,147.70096060987425,331.14433007379709,Ecologically Stable
16 2018-01-01 07:00:00,123.49072367150666,15.337928155663754,87.08562683267727,19.95596094378062,17,50,1,77.74736524075072,High,
17 6.1494377922246,8.751127612700293,136.06773168550563,61.41602016307321,153.88617032112554,Ecologically Degraded
18 2018-01-01 08:00:00,99.9480132032325,7.94880605048988,41.718545509964756,11.177830325737414,9,0,0,77.70951559158064,High,
19 8.270204592953835,7.820768468751011,136.95207646738427,151.02817627526483,430.5389877661993,Ecologically Healthy
20 2018-01-01 09:00:00,10.7957471452927,22.572905758688805,52.97977928130249,40.64787421328479,9,0,0,101.99050597778653,Low,
21 7.268345276167183,7.613868119963708,40.547813784897,82.21111587719554,200.071641924095156,Ecologically Stable
22 2018-01-01 10:00:00,333.56211413438531,23.066449987150915,40.28955196405273,38.75265914977947,18,0,0,136.67377749325463,Low,
23 5.89280299893932,6.165889250066889,87.9278436028715,181.28281721515404,469.2609640664438,Ecologically Degraded
24 2018-01-01 11:00:00,105.30828186740817,26.89197088367715,81.57501562167357,4.939448675693497,11,100,0,89.77102292958304,Moderate,
25 5.332705844942997,4.896607741009147,86.12336369824028,93.93539281349324,463.45189575773221,Ecologically Stable
26 2018-01-01 12:00:00,218.25884337417642,20.959940094139476,86.08698127875825,2.349385360241949,9,0,0,120.43371949989641,Low,
27 6.47868233090693,7.001054284198806,101.69198406809231,71.32939024510083,208.2785701882804,Ecologically Stable
28 2018-01-01 13:00:00,69.59208914799373,22.731351125972726,99.60228565585830,2.184207006392753,4,50,0,90.61255851685978,Low,
29 8.49708708916795,6.38420201200635,1.2100662398646173,15.140788680102158,375.3780531469331,Ecologically Healthy
30 2018-01-01 14:00:00,52.90137195271518,25.319610809627093,70.43870957239372,57.073172167893524,7,100,0,134.2920107636378,High,
31 5.62659391364228,6.308393220415704,299.6358151820461,188.21217442472255,461.5569092097606,Ecologically Healthy
32 2018-01-01 15:00:00,90.70037707360828,16.813918613737933,73.00535660613531,38.432042507902636,5,0,0,100.82707623517261,Low,
33 7.57880236012866,3.6490869807858916,209.01471602144443,112.59337303439088,39.58633313379509,Ecologically Stable
34 2018-01-01 16:00:00,50.26422403590408,16.18096476882445,80.9718702090666,33.02587697267432,7,0,0,90.465818763288,Low,
35 7.90520204191657,10.35096192573463,236.80425583001754,85.44319227026229,356.3784326420081,Ecologically Healthy
36 2018-01-01 17:00:00,65.88706905695942,18.96147741965165,95.12443660999655,45.43643347144485,12,100,1,103.41904568985949,Low,
37 8.09481626521909,4.939764101051264,294.67823588898443,34.13094369168348,321.7840039976218,Ecologically Stable
38 2018-01-01 18:00:00,82.46512394701323,14.896757439285006,87.08023829207745,58.88162512205105,12,100,1,122.67843073522998,Low,
39 7.24911608082899,9.706357115572171,12.343939326493114,63.4739821994871,120.713087634914,Ecologically Healthy
40 2018-01-01 19:00:00,23.00680524219417,26.69317762950267,62.49880850930732,46.42226567264945,19,100,1,101.60145802225379,Moderate,
41 7.409660296561433,4.92546640904543,141.1939246724691,72.79572873381623,335.5140734970934,Ecologically Healthy
42 2018-01-01 20:00:00,148.50300471065697,16.79040001214196,89.30128704991793,19.84573861980827,10,0,1,126.52145689425862,Moderate,
43 8.019319397153513,6.648713430002088,278.2647023377781,122.30885159288817,330.8495295860567,Ecologically Degraded
44 2018-01-01 21:00:00,26.759072313952004,19.02416734848438,55.44030943700152,5.72583450054139,5,100,2,128.8427239489277,Low,
45 7.236942931444238,8.45778225961333,27.747184352788025,108.4956116294621,193.9388330934224,Ecologically Healthy
46 2018-01-01 22:00:00,23.63067219173367,26.908201260761736,51.402028089074335,6.729762107616224,11,50,0,107.16189499217077,Low,
47 8.23693430573012,4.669435417095521,56.99828536126389,138.48163411046747,23.82678881396430,Ecologically Healthy
48 2018-01-01 23:00:00,96.69752089440674,25.349499570228396,79.86242089539846,31.13973013944009,5,100,0,125.85916631729302,Moderate,
49 7.31070132731331,0.783163994516502,110.22799591057377,36.93849994996675,160.8369870302185,Ecologically Stable
50 2018-01-02 00:00:00,140.66769039895252,20.140255273100657,81.97299512504452,52.0728294818551,12,0,0,106.0679627076207,Low,
51 7.036781547427843,4.437007948595327,81.62260449045165,19.47593953486052,491.1848807756901,Ecologically Stable
52 2018-01-02 01:00:00,74.89177871785353,20.79191444710878,74.8379986377794,25.014750285867294,5,0,0,74.00050042989473,Moderate,
53 7.32346117474464,4.153992784042823,246.57195958454258,68.2327532722613,171.1014400154784,Ecologically Healthy
54 2018-01-02 02:00:00,76.08895885423004,20.263067592027515,80.15806577867973,41.256527354905465,13,0,1,127.8564019191987,High,
55 7.807104303480269,4.661225840128121,283.7035348252378,5.244204694595078,108.67559825063883,Ecologically Healthy
56 2018-01-02 03:00:00,65.36907174917371,19.127127840167752,84.9884933113033,9.54213804043368,14,0,0,111.19473949318396,High,
57 8.024069575886145,8.151266390641945,129.50541845887104,112.4663867875721,117.18405805909798,Ecologically Healthy
58 2018-01-02 04:00:00,56.9954915343252,26.17333116767334,69.95256942346856,21.128650085274024,7,0,3,98.20657631380478,Low,
59 6.615716774609992,6.798466190804201,247.6708058056477,168.99033036753,302.97049458377637,Ecologically Healthy
```

Figure 1: Example of the structure of a CSV file. the file is having its columns separated by a comma delimiter

If it is separated by semi-colon, the delimiter is a semi colon, but the name remains the same. The interesting thing is that comma separated files can still have their name saved in .txt format, that's why checking the file internal structure is important. Although almost

anything can be used as a delimiter. Some common ones includes:

- comma - ,
- tab - \t
- semicolon - ;
- pipe - |,
- whitespace

To read a csv into R, you can run any of the following. They all have a first common argument which is the file path:

```
# Base R implementation
```

```
my_data <- read.csv(file = "data/ecological_health_dataset.csv")
```

```
head(my_data)
```

	Timestamp	PM2.5	Temperature	Humidity	Soil_Moisture
1	2018-01-01 00:00:00	119.68397	21.88583	53.95560	22.47978
2	2018-01-01 01:00:00	74.72324	19.07956	54.29895	23.98031
3	2018-01-01 02:00:00	69.11418	26.67587	98.99122	11.56632
4	2018-01-01 03:00:00	69.11511	20.17007	36.41646	36.14457
5	2018-01-01 04:00:00	232.48572	21.91575	79.35562	43.53254
6	2018-01-01 05:00:00	143.33531	15.02139	96.33046	37.93073

	Biodiversity_Index	Nutrient_Level	Water_Quality	Air_Quality_Index
1	9	50	0	82.59493
2	9	0	0	127.41848
3	7	50	0	95.21542
4	10	0	0	65.53427
5	11	0	1	80.31496
6	12	0	0	102.10155

	Pollution_Level	Soil_pH	Dissolved_Oxygen	Chemical_Oxygen_Demand
1	Low	5.284388	6.555422	24.11973
2	Moderate	6.107887	7.542608	164.58496
3	Low	8.361576	6.821085	24.81837

4	Low	7.929766	7.421999	248.72788
5	Low	5.378418	7.231868	271.06234
6	Low	5.579344	7.229231	280.21082
		Biochemical_Oxygen_Demand	Total_Dissolved_Solids	Ecological_Health_Label
1		9.731336	44.79406	Ecologically Healthy
2		178.793602	205.78702	Ecologically Stable
3		25.332886	448.38676	Ecologically Healthy
4		58.940128	359.25938	Ecologically Healthy
5		106.113663	118.30366	Ecologically Critical
6		120.859359	84.70938	Ecologically Healthy

`head()` returns the firsts 6 records of the dataset.

The `readr` implementation is also quite straight forward, instead of `read.csv()` it uses `read_csv()`.

This will be the first time we are seeing the operator `|>`. It is called the **pipe operator** and takes the result from its left hand side to the function/operation on the right hand side for evaluation. It is a nice way to chain operations without the need to break down your code. The above could be written as:

```
# Written as this
my_data <- read_csv("data/ecological_health_dataset.csv")
head(my_data)

# Or
head(read_csv("data/ecological_health_dataset.csv"))
```

The pipe will be used a lot moving forward so we get used to it.

The result of `read_csv()`, and `read.csv()` seems different, and that's because one is a tibble—often regarded as the modern and clean version of `data.frame`—and the other is a `data.frame`. They show the same data with difference in presentation. The tibble is more detail and displays information about the dimension of the data, its column specification, then the data itself with each data type displayed under the column name. To learn more about tibble visit the [Tibbles](#) chapter in the R for Data Science 2nd Edition Book.

Table 2: Reading a CSV file with readr's read_csv. Produces a tibble instead of a data.frame.

```
read_csv("data/ecological_health_dataset.csv") |> head()
```

Rows: 61345 Columns: 16

-- Column specification -----

Delimiter: ","

chr (2): Pollution_Level, Ecological_Health_Label

dbl (13): PM2.5, Temperature, Humidity, Soil_Moisture, Biodiversity_Index, ...

dtm (1): Timestamp

i Use `spec()` to retrieve the full column specification for this data.

i Specify the column types or set `show_col_types = FALSE` to quiet this message.

A tibble: 6 x 16

	Timestamp	PM2.5	Temperature	Humidity	Soil_Moisture
	<dtm>	<dbl>	<dbl>	<dbl>	<dbl>
1	2018-01-01 00:00:00	120.	21.9	54.0	22.5
2	2018-01-01 01:00:00	74.7	19.1	54.3	24.0
3	2018-01-01 02:00:00	69.1	26.7	99.0	11.6
4	2018-01-01 03:00:00	69.1	20.2	36.4	36.1
5	2018-01-01 04:00:00	232.	21.9	79.4	43.5
6	2018-01-01 05:00:00	143.	15.0	96.3	37.9

i 11 more variables: Biodiversity_Index <dbl>, Nutrient_Level <dbl>,
 # Water_Quality <dbl>, Air_Quality_Index <dbl>, Pollution_Level <chr>,
 # Soil_pH <dbl>, Dissolved_Oxygen <dbl>, Chemical_Oxygen_Demand <dbl>,
 # Biochemical_Oxygen_Demand <dbl>, Total_Dissolved_Solids <dbl>,
 # Ecological_Health_Label <chr>

Next is `data.table` implementation of reading files. It is the easiest, and fastest of all three when it comes to data reading, and we will see that later on in time. It uses the function `fread()`.

```
fread("data/ecological_health_dataset.csv") |>
  head()
```

	Timestamp	PM2.5	Temperature	Humidity	Soil_Moisture
	<POS<	<num>	<num>	<num>	<num>
1:	2018-01-01 00:00:00	119.68397	21.88583	53.95560	22.47978
2:	2018-01-01 01:00:00	74.72324	19.07956	54.29895	23.98031
3:	2018-01-01 02:00:00	69.11418	26.67587	98.99122	11.56632
4:	2018-01-01 03:00:00	69.11511	20.17007	36.41646	36.14457
5:	2018-01-01 04:00:00	232.48572	21.91575	79.35562	43.53254
6:	2018-01-01 05:00:00	143.33531	15.02139	96.33046	37.93073

	Biodiversity_Index	Nutrient_Level	Water_Quality	Air_Quality_Index
	<int>	<int>	<int>	<num>
1:	9	50	0	82.59493
2:	9	0	0	127.41848
3:	7	50	0	95.21542
4:	10	0	0	65.53427
5:	11	0	1	80.31496
6:	12	0	0	102.10155

	Pollution_Level	Soil_pH	Dissolved_Oxygen	Chemical_Oxygen_Demand
	<char>	<num>	<num>	<num>
1:	Low	5.284388	6.555422	24.11973
2:	Moderate	6.107887	7.542608	164.58496
3:	Low	8.361576	6.821085	24.81837
4:	Low	7.929766	7.421999	248.72788
5:	Low	5.378418	7.231868	271.06234
6:	Low	5.579344	7.229231	280.21082

	Biochemical_Oxygen_Demand	Total_Dissolved_Solids	Ecological_Health_Label
	<num>	<num>	<char>
1:	9.731336	44.79406	Ecologically Healthy

2:	178.793602	205.78702	Ecologically Stable
3:	25.332886	448.38676	Ecologically Healthy
4:	58.940128	359.25938	Ecologically Healthy
5:	106.113663	118.30366	Ecologically Critical
6:	120.859359	84.70938	Ecologically Healthy

Of the three implementations, only the tibbles limits the column display to only what the screen/document can contain at a point in time, while the others have no limits.

Sometimes we do have files that we want to import that are available online. These functions read online files without trouble, just ensure you know the file extension, so your data gets imported as expected. The import speed now depends on your internet speed and the file size.

```
read_csv("https://raw.githubusercontent.com/EU-Study-Assist/data-for-r4r/refs/heads/main/r4r-
head()")
```

Rows: 61345 Columns: 16

-- Column specification -----

Delimiter: ","

chr (2): Pollution_Level, Ecological_Health_Label

dbl (13): PM2.5, Temperature, Humidity, Soil_Moisture, Biodiversity_Index, ...

dtm (1): Timestamp

i Use `spec()` to retrieve the full column specification for this data.

i Specify the column types or set `show_col_types = FALSE` to quiet this message.

A tibble: 6 x 16

	Timestamp	PM2.5	Temperature	Humidity	Soil_Moisture
	<dtm>	<dbl>	<dbl>	<dbl>	<dbl>
1	2018-01-01 00:00:00	120.	21.9	54.0	22.5
2	2018-01-01 01:00:00	74.7	19.1	54.3	24.0
3	2018-01-01 02:00:00	69.1	26.7	99.0	11.6
4	2018-01-01 03:00:00	69.1	20.2	36.4	36.1
5	2018-01-01 04:00:00	232.	21.9	79.4	43.5

```

6 2018-01-01 05:00:00 143.          15.0      96.3          37.9
# i 11 more variables: Biodiversity_Index <dbl>, Nutrient_Level <dbl>,
#   Water_Quality <dbl>, Air_Quality_Index <dbl>, Pollution_Level <chr>,
#   Soil_pH <dbl>, Dissolved_Oxygen <dbl>, Chemical_Oxygen_Demand <dbl>,
#   Biochemical_Oxygen_Demand <dbl>, Total_Dissolved_Solids <dbl>,
#   Ecological_Health_Label <chr>

```

In addition to `read.csv` or `read_csv`, we have other `read` functions that are named according to their file extensions. A list of some `read_*` functions are give below. There's a super function that reads a lot of flat file, that's the `read.delim` or `read_delim`, you only have to specify your delimiter in the right argument—`sep` for `read.delim()` and `delim` for `read_delim`.

file extension	base R	readr	data.table
txt	<code>read.table</code>	<code>read.table</code>	<code>fread</code>
tsv	<code>read.delim</code>	<code>read_tsv</code>	<code>fread</code>
dat	<code>read.table</code>	<code>read_table</code>	<code>fread</code>
log	<code>readLines</code>	<code>read_lines</code>	<code>fread</code>
tab	<code>read.delim</code>	<code>read_delim</code>	<code>fread</code>
psv	<code>read.table</code>	<code>read_delim</code>	<code>fread</code>
fixed-width	<code>read.fwf</code>	<code>read_fwf</code>	<code>fread</code>

The good thing about `data.table` `fread` is that it guesses delimiter for its user, making it easier to import files. This in addition to how fast the package is, make it a useful package to learn.

There are other useful arguments you should take note off when you are importing data. Reading the documentation by using `help()` would expose you to these arguments. This include arguments like, `header/col_names`, `sep/delim`, `na`, `skip`, etc.

SpreadSheet

Base R, `readr`, and `data.table` cannot import spreadsheet data. Instead, packages are used. When we hear spreadsheet, MS Excel is the first thing that comes to mind—well maybe that's

my mind—but, spreadsheet are also different, and we can tell this by their file extensions. We have ods, xlsx, xls, gsheet. Except from gsheet which is usually a link in its non-native format, the rest can be seen as you tradition file extension on your personal computer.

WIP

- Spreadsheet
- Web API
- Web Scraping
- Introduction to Document Creation
 - Creating reports
 - Creating manuscripts
 - Website creation
- Capstone Project