SIM Global Education

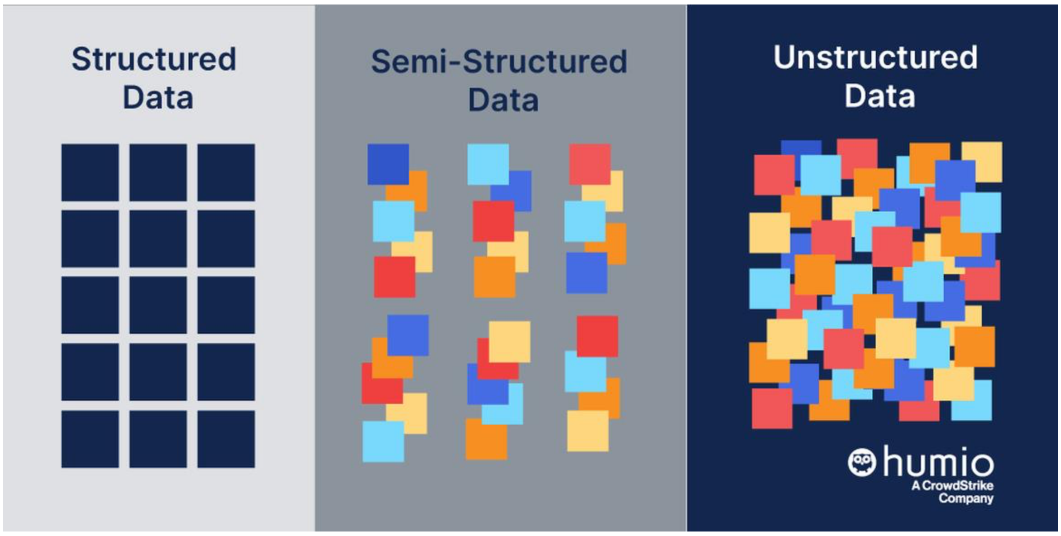
Block 2
Interacting with Data Structures

Learning Outcomes

After completing this topic and the recommended reading, you should be able to:

- Gain familiarity with various data types and structures as well as popular data-exchange formats (e.g. JSON, XML, CSV).
- Be able to work with various data types and structures and data-exchange formats in R and Python.

1. Data Categorisation



[Source: *https://www.humio.com/whats-new/blog/structured-logging-explained/*]

Structured Data

- Resides in predefined formats and models.
- Generally tabular data that is represented by columns (fields) and rows (records).
- Examples: relational databases

timestamp	latitude	longitude	altitude	distance	heart_rate	speed
2013-06-01 18:40:29	50.81381	-1.712606	80.20001	1805.94	133	4.060059
2013-06-01 18:40:30	50.81383	-1.712649	80.00000	1810.00	133	4.550049
2013-06-01 18:40:31	50.81385	-1.712700	79.79999	1814.55	133	2.979981
2013-06-01 18:40:32	50.81387	-1.712734	79.79999	1817.53	133	2.969971
2013-06-01 18:40:33	50.81388	-1.712777	79.59998	1820.50	133	3.650024
2013-06-01 18:40:34	50.81389	-1.712826	79.59998	1824.15	133	3.229980
2013-06-01 18:40:35	50.81391	-1.712862	79.40002	1827.38	133	4.650024
2013-06-01 18:40:36	50.81393	-1.712911	79.40002	1832.03	133	4.149902
2013-06-01 18:40:37	50.81395	-1.712963	79.20001	1836.18	133	2.000000
2013-06-01 18:40:38	50.81395	-1.712994	79.20001	1838.18	133	4.210083
2013-06-01 18:40:39	50.81396	-1.713053	79.00000	1842.39	133	5.189941

Unstructured Data

- Information that is text-heavy but may contain data such as numbers, dates, and facts.

- Stored in its natural format until it's extracted for analysis.
    - Extracted using machine learning, such as, *topic models* ("tm" R package).

```r
library("tm")
doc1 <- "I love programming in R and hate programming in Python"
doc2 <- "I love programming in Python and hate programming in R"
doc3 <- "I love programming in Python and R"
doc4 <- "I hate programming"
## Build a corpus and a document-term matrix
corpus <- Corpus(VectorSource(c(doc1, doc2, doc3, doc4)))
dt_mat <- DocumentTermMatrix(corpus)
as.matrix(dt_mat)
```

```
    Terms
Docs and hate love programming python
   1   1    1    1           2      1
   2   1    1    1           2      1
   3   1    0    1           1      1
   4   0    1    0           1      0
```

- Examples: videos; audio; and binary data files

Semi-structured Data

- Information that doesn't consist of structured data but still has some structure to it.
- A mix of both structured and unstructured data.
- Example: documents held in JSON format; R Markdown files

````
# My first R Markdown file

After a hard training day with Yoda, I decided to author my first [R
Markdown](https://rmarkdown.rstudio.com) file. This is a text chunk
written in *Markdown syntax*. I can write **bold** and *italic*, and
even record quotes I want to remember like

> *Do. Or do not. There is no try*
>
> Yoda, The Empire Strikes Back

I can also ask R to run code and return the results. For example, I can
ask R to print the quote

```{r quote}
print("Do. Or do not. There is no try")
```

I can also do complex arithmetic. For example, if your R installation
could do infinite arithmetic you could see that `1/81` has all single
digits numbers from 0 to 9 repeating in its decimal, except 8!

```{r arithmetic}
print(1/81, 15)
```
````

2. File Formats for Data Exchange

Plain Text

- Represents only characters of readable material but not its graphical representation nor other objects.
- May include *whitespace* characters that affect simple arrangement of text, such as spaces, line breaks, or tabulation characters.
- Extension: **.txt**

nile.txt

```
1120 1160 963 1210 1160 1160 813 1230 1370 1140 995 935 1110 994 1020 960 1180 799 958
1140 1100 1210 1150 1250 1260 1220 1030 1100 774 840 874 694 940 833 701 916 692 1020 1050
969 831 726 456 824 702 1120 1100 832 764 821 768 845 864 862 698 845 744 796 1040 759 781
865 845 944 984 897 822 1010 771 676 649 846 812 742 801 1040 860 874 848 890 744 749 838
1050 918 986 797 923 975 815 1020 906 901 1170 912 746 919 718 714 740
```

sunset-salvo.txt

```
The combination of some data and an aching desire for an answer does not ensure that a
reasonable answer can be extracted from a given body of data
```

Delimiter-Separated Values

- Stores two-dimensional arrays of data by separating the values in each row with specific delimiter characters, such as tabs, or commas
- Extension: **.tsv**; **.csv**

running_dat.tsv

```
"timestamp"	"latitude"	"longitude"	"altitude"	"distance"	"heart_rate"	"speed"
2013-06-01 18:40:29	50.813805	-1.7126063	80.2000122	1805.9399512	133	4.06005860000005
2013-06-01 18:40:30	50.8138298	-1.7126487	80	1810.0000098	133	4.55004880000001
2013-06-01 18:40:31	50.8138543	-1.7127005	79.7999878	1814.5500586	133	2.97998050000001
2013-06-01 18:40:32	50.8138709	-1.7127338	79.7999878	1817.5300391	133	2.96997069999998
2013-06-01 18:40:33	50.8138757	-1.7127769	79.5999756	1820.5000098	133	3.65002439999989
2013-06-01 18:40:34	50.8138862	-1.712826	79.5999756	1824.1500342	133	3.22998040000016
2013-06-01 18:40:35	50.8139051	-1.7128616	79.4000244	1827.3800146	133	4.65002449999997
2013-06-01 18:40:36	50.8139326	-1.7129115	79.4000244	1832.0300391	133	4.14990229999989
2013-06-01 18:40:37	50.8139517	-1.7129626	79.2000122	1836.1799414	133	2
2013-06-01 18:40:38	50.8139537	-1.7129945	79.2000122	1838.1799414	133	4.21008299999994
2013-06-01 18:40:39	50.8139622	-1.7130533	79	1842.3900244	133	5.18994140000018
```

running_dat.csv

```
"timestamp","latitude","longitude","altitude","distance","heart_rate","speed"
2013-06-01 18:40:29,50.813805,-1.7126063,80.2000122,1805.9399512,133,4.06005860000005
2013-06-01 18:40:30,50.8138298,-1.7126487,80,1810.0000098,133,4.55004880000001
2013-06-01 18:40:31,50.8138543,-1.7127005,79.7999878,1814.5500586,133,2.97998050000001
2013-06-01 18:40:32,50.8138709,-1.7127338,79.7999878,1817.5300391,133,2.96997069999998
2013-06-01 18:40:33,50.8138757,-1.7127769,79.5999756,1820.5000098,133,3.65002439999989
2013-06-01 18:40:34,50.8138862,-1.712826,79.5999756,1824.1500342,133,3.22998040000016
2013-06-01 18:40:35,50.8139051,-1.7128616,79.4000244,1827.3800146,133,4.65002449999997
2013-06-01 18:40:36,50.8139326,-1.7129115,79.4000244,1832.0300391,133,4.14990229999989
2013-06-01 18:40:37,50.8139517,-1.7129626,79.2000122,1836.1799414,133,2
2013-06-01 18:40:38,50.8139537,-1.7129945,79.2000122,1838.1799414,133,4.21008299999994
2013-06-01 18:40:39,50.8139622,-1.7130533,79,1842.3900244,133,5.18994140000018
```

XML (eXtensible Markup Language)

- Defines a set of rules for encoding documents (structured and semi-structured) in a format that is both human-readable and machine-readable
- Simple and very flexible text format derived from SGML (Standard Generalized Markup Language)
- Great format for storing hierarchical data
- Syntax:

  <markup> content </markup>

  <element>

  <child element> data </child element>

  </element>

- Extension: **.xml**

statisticians.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- A list of famous statisticians -->
<records>
  <statistician>
    <name>Ronald</name>
    <middle>Aylmer</middle>
    <surname>Fisher</surname>
    <dob>17/02/1890</dob>
    <wiki>https://en.wikipedia.org/wiki/Ronald_Fisher</wiki>
  </statistician>
  <statistician>
    <name>William</name>
    <middle>Sealy</middle>
    <surname>Gosset</surname>
    <dob>13/08/1876</dob>
    <wiki>https://en.wikipedia.org/wiki/William_Sealy_Gosset</wiki>
  </statistician>
  <statistician>
    <name>David</name>
    <middle>Roxbee</middle>
    <surname>Cox</surname>
    <dob>15/07/1924</dob>
    <wiki>https://en.wikipedia.org/wiki/David_Cox_(statistician)</wiki>
  </statistician>
  <statistician>
```

JSON (JavaScript Object Notation)

- Open standard file format and lightweight data interchange format, easy for humans to read and write, and easy for machines to parse and generate.

- Uses human-readable text to store and transmit data objects consisting of attribute-value pairs and arrays.
- Supports basic variable types, including strings, numbers, Booleans, null, arrays and objects.

```
{
    "language": "Python",
    "release": 1991,
    "os": ["Linux", "macOS", "Windows"],
    "oo": true,
    "pastnames": null
}
```

- Syntax:

  Object: { … … }

  "key": "value"

- Extension: **.json**

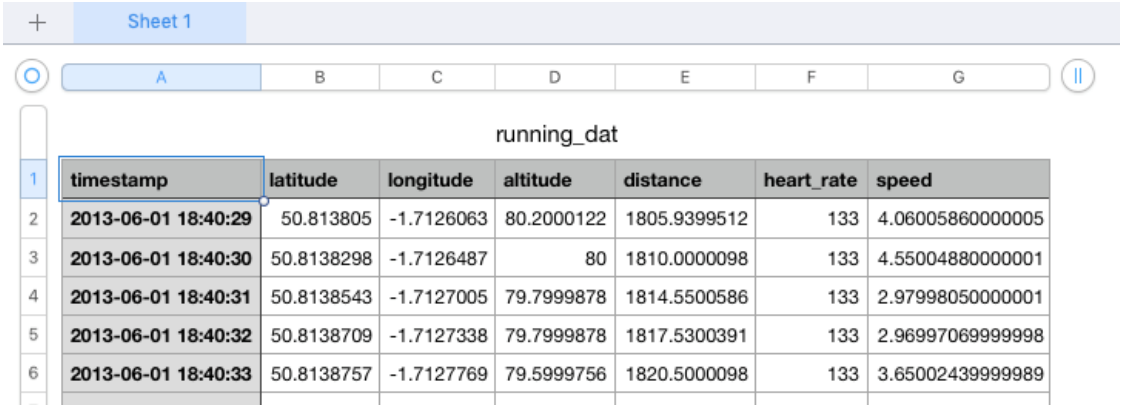
statisticians.json

```
[
    {
        "name": "Ronald",
        "middle": "Aylmer",
        "surname": "Fisher",
        "dob": "17/02/1890",
        "wiki": "https://en.wikipedia.org/wiki/Ronald_Fisher"
    },
    {
        "name": "William",
        "middle": "Sealy",
        "surname": "Gosset",
        "dob": "13/08/1876",
        "wiki": "https://en.wikipedia.org/wiki/William_Sealy_Gosset"
    },
    {
        "name": "David",
        "middle": "Roxbee",
        "surname": "Cox",
        "dob": "15/07/1924",
        "wiki": "https://en.wikipedia.org/wiki/David_Cox_(statistician)"
    },
    {
        "name": "Thomas",
        "middle": null,
```

Spreadsheets

- Computer application for organisation, analysis, and storage of data in tabular form.
- Program operates on data (numeric, text, or formulas) entered in cells of a table.

- Example: Microsoft Excel; LibreOffice Calc; Apple Numbers
- Extension: **<u>.xlsx</u>**; **<u>.ods</u>**; **<u>.numbers</u>**



running_dat

timestamp	latitude	longitude	altitude	distance	heart_rate	speed
2013-06-01 18:40:29	50.813805	-1.7126063	80.2000122	1805.9399512	133	4.06005860000005
2013-06-01 18:40:30	50.8138298	-1.7126487	80	1810.0000098	133	4.55004880000001
2013-06-01 18:40:31	50.8138543	-1.7127005	79.7999878	1814.5500586	133	2.97998050000001
2013-06-01 18:40:32	50.8138709	-1.7127338	79.7999878	1817.5300391	133	2.96997069999998
2013-06-01 18:40:33	50.8138757	-1.7127769	79.5999756	1820.5000098	133	3.65002439999989

- Read data from a file, as "character" type
    - nile_char <- scan("nile.txt", what = character())
- Show variable content
    - nile_char

```
 [1] "1120" "1160" "963"  "1210" "1160" "1160" "813"  "1230" "1370" "1140"
[11] "995"  "935"  "1110" "994"  "1020" "960"  "1180" "799"  "958"  "1140"
[21] "1100" "1210" "1150" "1250" "1260" "1220" "1030" "1100" "774"  "840"
[31] "874"  "694"  "940"  "833"  "701"  "916"  "692"  "1020" "1050" "969"
[41] "831"  "726"  "456"  "824"  "702"  "1120" "1100" "832"  "764"  "821"
[51] "768"  "845"  "864"  "862"  "698"  "845"  "744"  "796"  "1040" "759"
[61] "781"  "865"  "845"  "944"  "984"  "897"  "822"  "1010" "771"  "676"
[71] "649"  "846"  "812"  "742"  "801"  "1040" "860"  "874"  "848"  "890"
[81] "744"  "749"  "838"  "1050" "918"  "986"  "797"  "923"  "975"  "815"
[91] "1020" "906"  "901"  "1170" "912"  "746"  "919"  "718"  "714"  "740"
```

- Check data type
    - typeof(nile_char)

```
[1] "character"
```

- Default delimiter: "white-space"
- Read data from a file, separated by ";"
    - sunset <- scan("sunset-salvo-sem.txt", what = character(), sep = ";")

```
sunset-salvo-sem.txt
The;combination;of;some;data;and;an;aching;desire;for;an;answer;does;not;ensure;that;a;rea
sonable;answer;can;be;extracted;from;a;given;body;of;data
```

- Show variable content
    - Sunset

```
 [1] "The"   "combination" "of"         "some"   "data"
 [6] "and"   "an"          "aching"     "desire" "for"
[11] "an"    "answer"      "does"       "not"    "ensure"
[16] "that"  "a"           "reasonable" "answer" "can"
[21] "be"    "extracted"   "from"       "a"      "given"
[26] "body"  "of"          "data"
```

- Different data types; skip lines
- Read data from a file

    - running <- scan("running_dat.tsv", what = list("", 1.0, 1.0, 1.0, 1.0, 1.0, 1.0), sep = "\t", skip = 1)

```
running_dat.tsv
"timestamp"	"latitude"	"longitude"	"altitude"	"distance"	"heart_rate"	"speed"
2013-06-01 18:40:29	50.813805	-1.7126063	80.2000122	1805.9399512	133	4.06005860000005
2013-06-01 18:40:30	50.8138298	-1.7126487	80	1810.0000098	133	4.55004880000001
2013-06-01 18:40:31	50.8138543	-1.7127005	79.7999878	1814.5500586	133	2.97998050000001
2013-06-01 18:40:32	50.8138709	-1.7127338	79.7999878	1817.5300391	133	2.96997069999998
2013-06-01 18:40:33	50.8138757	-1.7127769	79.5999756	1820.5000098	133	3.65002439999989
2013-06-01 18:40:34	50.8138862	-1.712826	79.5999756	1824.1500342	133	3.22998040000016
2013-06-01 18:40:35	50.8139051	-1.7128616	79.4000244	1827.3800146	133	4.65002449999997
2013-06-01 18:40:36	50.8139326	-1.7129115	79.4000244	1832.0300391	133	4.14990229999989
2013-06-01 18:40:37	50.8139517	-1.7129626	79.2000122	1836.1799414	133	2
2013-06-01 18:40:38	50.8139537	-1.7129945	79.2000122	1838.1799414	133	4.21008299999994
2013-06-01 18:40:39	50.8139622	-1.7130533	79	1842.3900244	133	5.18994140000018
```

- Show variable content
    - running

```
[[1]]
 [1] "2013-06-01 18:40:29" "2013-06-01 18:40:30" "2013-06-01 18:40:31"
 [4] "2013-06-01 18:40:32" "2013-06-01 18:40:33" "2013-06-01 18:40:34"
 [7] "2013-06-01 18:40:35" "2013-06-01 18:40:36" "2013-06-01 18:40:37"
[10] "2013-06-01 18:40:38" "2013-06-01 18:40:39"

[[2]]
 [1] 50.81381 50.81383 50.81385 50.81387 50.81388 50.81389 50.81391 50.81393
 [9] 50.81395 50.81395 50.81396

[[3]]
 [1] -1.712606 -1.712649 -1.712700 -1.712734 -1.712777 -1.712826 -1.712862
 [8] -1.712911 -1.712963 -1.712994 -1.713053

[[4]]
 [1] 80.20001 80.00000 79.79999 79.79999 79.59998 79.59998 79.40002 79.40002
 [9] 79.20001 79.20001 79.00000

[[5]]
 [1] 1805.94 1810.00 1814.55 1817.53 1820.50 1824.15 1827.38 1832.03 1836.18
[10] 1838.18 1842.39

[[6]]
 [1] 133 133 133 133 133 133 133 133 133 133 133

[[7]]
 [1] 4.060059 4.550049 2.979981 2.969971 3.650024 3.229980 4.650024 4.149902
 [9] 2.000000 4.210083 5.189941
```

Export Plain Text Files in R

- Output objects to a text file, concatenate and print
    - cat(nile, file = "nile1.txt", sep = ",")

```
nile1.txt
1120,1160,963,1210,1160,1160,813,1230,1370,1140,995,935,1110,994,1020,960,1180,799,958,114
0,1100,1210,1150,1250,1260,1220,1030,1100,774,840,874,694,940,833,701,916,692,1020,1050,96
9,831,726,456,824,702,1120,1100,832,764,821,768,845,864,862,698,845,744,796,1040,759,781,8
65,845,944,984,897,822,1010,771,676,649,846,812,742,801,1040,860,874,848,890,744,749,838,1
050,918,986,797,923,975,815,1020,906,901,1170,912,746,919,718,714,740
```

Import Delimiter-Separated Values Files in R

- Read data as table from a file
  - running <- read.table("running_dat.tsv", header = TRUE, sep = "\t")

```
running_dat.tsv
"timestamp"          "latitude"    "longitude"    "altitude"    "distance"     "heart_rate"    "speed"
2013-06-01 18:40:29  50.813805     -1.7126063     80.2000122    1805.9399512   133    4.06005860000005
2013-06-01 18:40:30  50.8138298    -1.7126487     80     1810.0000098    133    4.55004880000001
```

- Show variable content
  - running

```
              timestamp latitude longitude altitude distance heart_rate    speed
1  2013-06-01 18:40:29 50.81381 -1.712606 80.20001  1805.94        133 4.060059
2  2013-06-01 18:40:30 50.81383 -1.712649 80.00000  1810.00        133 4.550049
3  2013-06-01 18:40:31 50.81385 -1.712700 79.79999  1814.55        133 2.979981
4  2013-06-01 18:40:32 50.81387 -1.712734 79.79999  1817.53        133 2.969971
5  2013-06-01 18:40:33 50.81388 -1.712777 79.59998  1820.50        133 3.650024
6  2013-06-01 18:40:34 50.81389 -1.712826 79.59998  1824.15        133 3.229980
7  2013-06-01 18:40:35 50.81391 -1.712862 79.40002  1827.38        133 4.650024
8  2013-06-01 18:40:36 50.81393 -1.712911 79.40002  1832.03        133 4.149902
9  2013-06-01 18:40:37 50.81395 -1.712963 79.20001  1836.18        133 2.000000
10 2013-06-01 18:40:38 50.81395 -1.712994 79.20001  1838.18        133 4.210083
11 2013-06-01 18:40:39 50.81396 -1.713053 79.00000  1842.39        133 5.189941
```

- Automatically figure out the variable types
  - str(running)

- Read data as table from a CSV file
  - running <- read.table("running_dat.csv", header = TRUE, sep = ",")
  - running <- read.csv("running_dat.csv")

```
running_dat.csv
"timestamp","latitude","longitude","altitude","distance","heart_rate","speed"
2013-06-01 18:40:29,50.813805,-1.7126063,80.2000122,1805.9399512,133,4.06005860000005
2013-06-01 18:40:30,50.8138298,-1.7126487,80,1810.0000098,133,4.55004880000001
```

- Using "." for decimal points

- Read data from ";" separated, no header
  - sunset <- read.csv2("sunset-salvo-sem.txt", header = FALSE)

```
   V1           V2  V3    V4    V5   V6  V7      V8      V9  V10  V11     V12   V13  V14
1  The combination  of  some  data  and  an  aching desire  for   an  answer  does  not
     V15  V16 V17          V18     V19  V20  V21         V22  V23  V24    V25   V26  V27
1 ensure that    a  reasonable answer can   be  extracted from    a  given  body   of
   V28
1 data
```

- Read data from ";" separated, no observations
    - sunset <- read.csv2("sunset-salvo-sem.txt")

```
 [1] The         combination of          some        data        and
 [7] an          aching      desire      for.        an.1        answer
[13] does        not         ensure      that        a           reasonable
[19] answer.1    can         be          extracted   from        a.1
[25] given       body        of.1        data.1
<0 rows> (or 0-length row.names)
```

Export Delimiter-Separated Values Files in R

- Write data in table to a data file
    - write.table(running, file = "running.dat", sep = "&&", dec = "*")

running.dat

```
"timestamp"&&"latitude"&&"longitude"&&"altitude"&&"distance"&&"heart_rate"&&"speed"
"1"&&"2013-06-01 18:40:29"&&50*813805&&-1*7126063&&80*2000122&&1805*9399512&&133&&4*06005860000005
"2"&&"2013-06-01 18:40:30"&&50*8138298&&-1*7126487&&80&&1810*0000098&&133&&4*55004880000001
"3"&&"2013-06-01 18:40:31"&&50*8138543&&-1*7127005&&79*7999878&&1814*5500586&&133&&2*97998050000001
"4"&&"2013-06-01 18:40:32"&&50*8138709&&-1*7127338&&79*7999878&&1817*5300391&&133&&2*96997069999998
"5"&&"2013-06-01 18:40:33"&&50*8138757&&-1*7127769&&79*5999756&&1820*5000098&&133&&3*65002439999989
"6"&&"2013-06-01 18:40:34"&&50*8138862&&-1*712826&&79*5999756&&1824*1500342&&133&&3*22998040000016
"7"&&"2013-06-01 18:40:35"&&50*8139051&&-1*7128616&&79*4000244&&1827*3800146&&133&&4*65002449999997
"8"&&"2013-06-01 18:40:36"&&50*8139326&&-1*7129115&&79*4000244&&1832*0300391&&133&&4*14990229999989
"9"&&"2013-06-01 18:40:37"&&50*8139517&&-1*7129626&&79*2000122&&1836*1799414&&133&&2
"10"&&"2013-06-01 18:40:38"&&50*8139537&&-1*7129945&&79*2000122&&1838*1799414&&133&&4*21008299999994
"11"&&"2013-06-01 18:40:39"&&50*8139622&&-1*7130533&&79&&1842*3900244&&133&&5*18994140000018
```

- Inspect the lines in the file
    - readLines("running.dat")

```
 [1] "\"timestamp\"&&\"latitude\"&&\"longitude\"&&\"altitude\"&&\"distance\"&&\"heart_rate\"&&\"speed\""
 [2] "\"1\"&&\"2013-06-01 18:40:29\"&&50*813805&&-1*7126063&&80*2000122&&1805*9399512&&133&&4*06005860000005"
 [3] "\"2\"&&\"2013-06-01 18:40:30\"&&50*8138298&&-1*7126487&&80&&1810*0000098&&133&&4*55004880000001"
 [4] "\"3\"&&\"2013-06-01 18:40:31\"&&50*8138543&&-1*7127005&&79*7999878&&1814*5500586&&133&&2*97998050000001"
 [5] "\"4\"&&\"2013-06-01 18:40:32\"&&50*8138709&&-1*7127338&&79*7999878&&1817*5300391&&133&&2*96997069999998"
 [6] "\"5\"&&\"2013-06-01 18:40:33\"&&50*8138757&&-1*7127769&&79*5999756&&1820*5000098&&133&&3*65002439999989"
 [7] "\"6\"&&\"2013-06-01 18:40:34\"&&50*8138862&&-1*712826&&79*5999756&&1824*1500342&&133&&3*22998040000016"
 [8] "\"7\"&&\"2013-06-01 18:40:35\"&&50*8139051&&-1*7128616&&79*4000244&&1827*3800146&&133&&4*65002449999997"
 [9] "\"8\"&&\"2013-06-01 18:40:36\"&&50*8139326&&-1*7129115&&79*4000244&&1832*0300391&&133&&4*14990229999989"
[10] "\"9\"&&\"2013-06-01 18:40:37\"&&50*8139517&&-1*7129626&&79*2000122&&1836*1799414&&133&&2"
[11] "\"10\"&&\"2013-06-01 18:40:38\"&&50*8139537&&-1*7129945&&79*2000122&&1838*1799414&&133&&4*21008299999994"
[12] "\"11\"&&\"2013-06-01 18:40:39\"&&50*8139622&&-1*7130533&&79&&1842*3900244&&133&&5*18994140000018"
```

Import XML Files in R

- Use "xml2" package, more automatic than "XML"
    - install.packages("xml2")

  - library(“xml2”)

- Read data from a file
  - stats_people <- read.xml(“statisticians.xml”)

```
statisticians.xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- A list of famous statisticians -->
<records>
  <statistician>
    <name>Ronald</name>
    <middle>Aylmer</middle>
    <surname>Fisher</surname>
    <dob>17/02/1890</dob>
    <wiki>https://en.wikipedia.org/wiki/Ronald_Fisher</wiki>
  </statistician>
  <statistician>
    <name>William</name>
    <middle>Sealy</middle>
    <surname>Gosset</surname>
    <dob>13/08/1876</dob>
    <wiki>https://en.wikipedia.org/wiki/William_Sealy_Gosset</wiki>
  </statistician>
  <statistician>
    <name>David</name>
    <middle>Roxbee</middle>
```

- Show variable content
  - stats_people

```
{xml_document}
<records>
[1] <statistician>\n  <name>Ronald</name>\n  <middle>Aylmer</middle>\n  <surn ...
[2] <statistician>\n  <name>William</name>\n  <middle>Sealy</middle>\n  <surn ...
[3] <statistician>\n  <name>David</name>\n  <middle>Roxbee</middle>\n  <surna ...
[4] <statistician>\n  <name>Thomas</name>\n  <middle/>\n  <surname>Bayes</sur ...
[5] <statistician>\n  <name>Karl</name>\n  <middle/>\n  <surname>Pearson</sur ...
[6] <statistician>\n  <name>John</name>\n  <middle>Wilder</middle>\n  <surnam ...
```

- Extract tag name and children
  - xml_name(stats_people)

```
[1] "records"
```

  - xml_children(stats_people)

```
{xml_nodeset (6)}
[1] <statistician>\n  <name>Ronald</name>\n  <middle>Aylmer</middle>\n  <surn ...
[2] <statistician>\n  <name>William</name>\n  <middle>Sealy</middle>\n  <surn ...
[3] <statistician>\n  <name>David</name>\n  <middle>Roxbee</middle>\n  <surna ...
[4] <statistician>\n  <name>Thomas</name>\n  <middle/>\n  <surname>Bayes</sur ...
[5] <statistician>\n  <name>Karl</name>\n  <middle/>\n  <surname>Pearson</sur ...
[6] <statistician>\n  <name>John</name>\n  <middle>Wilder</middle>\n  <surnam ...
```

- Extract tag elements content
  - surname_nodes <- xml_find_all(stats_people, “.//surname”)
  - surname_nodes

```
{xml_nodeset (6)}
[1] <surname>Fisher</surname>
[2] <surname>Gosset</surname>
[3] <surname>Cox</surname>
[4] <surname>Bayes</surname>
[5] <surname>Pearson</surname>
[6] <surname>Tukey</surname>
```

- Extract elements data
  - xml_text(surname_nodes)

```
[1] "Fisher"  "Gosset"  "Cox"     "Bayes"   "Pearson" "Tukey"
```

- Extract data in tabular form, use “XML” package
  - library(“XML”)
  - stats_people_df <- xmlToDataFrame(“statisticians.xml”)
  - stats_people_df

```
     name middle surname        dob
1  Ronald Aylmer  Fisher 17/02/1890
2 William  Sealy  Gosset 13/08/1876
3   David Roxbee     Cox 15/07/1924
4  Thomas          Bayes 07/04/1761
5    Karl        Pearson 27/03/1857
6    John Wilder   Tukey 16/06/1915
                                                 wiki
1            https://en.wikipedia.org/wiki/Ronald_Fisher
2     https://en.wikipedia.org/wiki/William_Sealy_Gosset
3 https://en.wikipedia.org/wiki/David_Cox_(statistician)
4             https://en.wikipedia.org/wiki/Thomas_Bayes
5             https://en.wikipedia.org/wiki/Karl_Pearson
6               https://en.wikipedia.org/wiki/John_Tukey
```

Export XML Files in R

- Remove record
  - stats_people

```
{xml_document}
<records>
[1] <statistician>\n  <name>Ronald</name>\n  <middle>Aylmer</middle>\n  <surn ...
[2] <statistician>\n  <name>William</name>\n  <middle>Sealy</middle>\n  <surn ...
[3] <statistician>\n  <name>David</name>\n  <middle>Roxbee</middle>\n  <surna ...
[4] <statistician>\n  <name>Thomas</name>\n  <middle/>\n  <surname>Bayes</sur ...
[5] <statistician>\n  <name>Karl</name>\n  <middle/>\n  <surname>Pearson</sur ...
[6] <statistician>\n  <name>John</name>\n  <middle>Wilder</middle>\n  <surnam ...
```

- bayes_record <- xml_children(stats_peple)[[4]]
- xml_remove(bayes_record)
- stats_people

```
{xml_document}
<records>
[1] <statistician>\n  <name>Ronald</name>\n  <middle>Aylmer</middle>\n  <surn ...
[2] <statistician>\n  <name>William</name>\n  <middle>Sealy</middle>\n  <surn ...
[3] <statistician>\n  <name>David</name>\n  <middle>Roxbee</middle>\n  <surna ...
[4] <statistician>\n  <name>Karl</name>\n  <middle/>\n  <surname>Pearson</sur ...
[5] <statistician>\n  <name>John</name>\n  <middle>Wilder</middle>\n  <surnam ...
```

- write objects to a xml file
    - library("xml2")
    - write_xml (stats_people, "statisticians_no_bayes.xml")

statisticians_no_bayes.xml

```
  </statistician>
  <statistician>
    <name>Karl</name>
    <middle/>
    <surname>Pearson</surname>
    <dob>27/03/1857</dob>
    <wiki>https://en.wikipedia.org/wiki/Karl_Pearson</wiki>
  </statistician>
  <statistician>
    <name>John</name>
    <middle>Wilder</middle>
    <surname>Tukey</surname>
    <dob>16/06/1915</dob>
    <wiki>https://en.wikipedia.org/wiki/John_Tukey</wiki>
  </statistician>
</records>
```

Import JSON Files in R

- Use "jsonlite" package
    - library("jsonlite")
- Read JSON documents from a file

- stats_people <- fromJSON("statisticians.json")

```
statisticians.json
[
    {
        "name": "Ronald",
        "middle": "Aylmer",
        "surname": "Fisher",
        "dob": "17/02/1890",
        "wiki": "https://en.wikipedia.org/wiki/Ronald_Fisher"
    },
    {
        "name": "William",
        "middle": "Sealy",
        "surname": "Gosset",
        "dob": "13/08/1876",
        "wiki": "https://en.wikipedia.org/wiki/William_Sealy_Gosset"
    },
    {
        "name": "David",
        "middle": "Roxbee",
        "surname": "Cox",
        "dob": "15/07/1924",
        "wiki": "https://en.wikipedia.org/wiki/David_Cox_(statistician)"
```

- Show variable content
  - stats_people

```
     name middle surname        dob
1  Ronald Aylmer  Fisher 17/02/1890
2 William  Sealy  Gosset 13/08/1876
3   David Roxbee     Cox 15/07/1924
4  Thomas   <NA>   Bayes 07/04/1761
5    Karl   <NA> Pearson 27/03/1857
6    John Wilder   Tukey 16/06/1915
                                                   wiki
1           https://en.wikipedia.org/wiki/Ronald_Fisher
2     https://en.wikipedia.org/wiki/William_Sealy_Gosset
3 https://en.wikipedia.org/wiki/David_Cox_(statistician)
4            https://en.wikipedia.org/wiki/Thomas_Bayes
5            https://en.wikipedia.org/wiki/Karl_Pearson
6              https://en.wikipedia.org/wiki/John_Tukey
```

Export JSON Files in R

- convert tabular objects to JSON object
  - running_json <- toJSON(running)
  - running_json

```
[{"timestamp":"2013-06-01 18:40:29","latitude":50.8138,"longitude":-1.7126,"altitude":80.2,"distance
":1805.94,"heart_rate":133,"speed":4.0601},{"timestamp":"2013-06-01 18:40:30","latitude":50.8138,"lo
ngitude":-1.7126,"altitude":80,"distance":1810,"heart_rate":133,"speed":4.55},{"timestamp":"2013-06-
01 18:40:31","latitude":50.8139,"longitude":-1.7127,"altitude":79.8,"distance":1814.5501,"heart_rate
":133,"speed":2.98},{"timestamp":"2013-06-01 18:40:32","latitude":50.8139,"longitude":-1.7127,"altit
ude":79.8,"distance":1817.53,"heart_rate":133,"speed":2.97},{"timestamp":"2013-06-01 18:40:33","lati
```

- prettify(running_json)

```
[
    {
        "timestamp": "2013-06-01 18:40:29",
        "latitude": 50.8138,
        "longitude": -1.7126,
        "altitude": 80.2,
        "distance": 1805.94,
        "heart_rate": 133,
        "speed": 4.0601
    },
    {
        "timestamp": "2013-06-01 18:40:30",
        "latitude": 50.8138,
        "longitude": -1.7126,
        "altitude": 80,
        "distance": 1810,
        "heart_rate": 133,
        "speed": 4.55
    },
```

- write tabular objects to json file
  - write_json(running, “running.json”, pretty = TRUE)

running.json

```
[
  {
    "timestamp": "2013-06-01 18:40:29",
    "latitude": 50.8138,
    "longitude": -1.7126,
    "altitude": 80.2,
    "distance": 1805.94,
    "heart_rate": 133,
    "speed": 4.0601
  },
  {
    "timestamp": "2013-06-01 18:40:30",
    "latitude": 50.8138,
    "longitude": -1.7126,
    "altitude": 80,
    "distance": 1810,
    "heart_rate": 133,
    "speed": 4.55
  },
  {
```

4. Data Types in R

Logical

- Stores logical or **Boolean** values of ***TRUE*** or ***FALSE***
  - a <- TRUE
  - b <- FALSE
  - class(a)

```
[1] "logical"
```

- Logical operators
  - Conjunction (AND): "**&**"
  - Disjunction (OR): "|"
  - Negation (NOT): "**!**"

<table>
<tr><th>a</th><th>b</th><th>a & b</th><th>a | b</th><th>!a</th></tr>
<tr><td>T</td><td>T</td><td>T</td><td>T</td><td>F</td></tr>
<tr><td>T</td><td>F</td><td>F</td><td>T</td><td>F</td></tr>
<tr><td>F</td><td>T</td><td>F</td><td>T</td><td>T</td></tr>
<tr><td>F</td><td>F</td><td>F</td><td>F</td><td>T</td></tr>
</table>

Character

- Stores sequences of characters (letters, numbers, and symbols)
- Uses double (" ") or single (' ') quotation marks to represent text
  - str1 <- "Data"
  - str2 <- 'Structures'
  - str3 <- "in 'R'"
  - class(str1)

```
[1] "character"
```

- Concatenate strings
  - str4 <- paste(str1, str2, str3)
  - print(str4)

```
[1] "Data Structures in 'R'"
```

  - cat(str1, str2, str3, “\n”)

Numeric and Integer

- Stores numeric values, the default data type for numbers
  - x <- 2.5
  - class(x)

```
[1] "numeric"
```

  - y <- 2
  - class(y)

```
[1] "numeric"
```

- Stores as Integer
  - z <- as.integer(5)

    or

    z <- 5L
  - class(z)

```
[1] "integer"
```

- Basic arithmetic operators

- 3 + 2 # *Addition* => 5
- 5 – 2 # *Subtraction* => 3
- 5 * -2 # *Multiplication* => -10
- 5 / 2.5 # *Division* => 2
- 2**2 # *Exponentiation* => 4 (also 2^2)
- 10 %% 3 # *Modulus* => 1

- Logical comparison operators
  - 2 > 4 => FALSE
  - exp(pi) < pi^exp(1) => FALSE
  - log(4) < 4 => TRUE
  - a <- 2.321
    a == 2.321 => TRUE

Changing Data Types in R

- Coerce objects from one type to another, using "as.datatype()"
  - as.character(FALSE) => "FALSE"
  - as.character(1.123) => "1.123"
  - as.integer(1.123) => 1
  - as.logical(4) => TRUE
  - as.logical(0) => FALSE
  - as.numeric(1L) => 1
  - as.numeric("abc") => NA (not available)
  - as.list(mtcars) =>

```
$mpg
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.
[16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.
[31] 15.0 21.4

$cyl
 [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8

$disp
 [1] 160.0 160.0 108.0 258.0 360.0 225.0 360.0 146.7 140.8 167.
[13] 275.8 275.8 472.0 460.0 440.0  78.7  75.7  71.1 120.1 318.
[25] 400.0  79.0 120.3  95.1 351.0 145.0 301.0 121.0
```

Testing Data Types in R

- Test the type of an object, using "is.datatype()"
    - is.character(3) => FALSE
    - is.character("hello") => TRUE
    - is.logical(4) => FALSE
    - is.logical(TRUE) => TRUE
    - is.numeric(4) => TRUE
    - is.numeric(FALSE) => FALSE
    - is.complex(43.3) => FALSE
    - is.complex(43.3i + 2) => TRUE

None Data Types in R

- ***NA*** (not available)
    - mean("a")
    - as.numeric("b")
- ***NaN*** (not available number)
    - log(-1)
    - sqrt(-1)

5. Data Structures in R

Vectors

- One-dimensional array
- Homogenous: only contains elements of the same data types

- Create using combine function "c()", separate elements by comma ","
    - marks <- c(80, 75, 90, 99, 100)
    - names <- c("John", "Jane", "Richard", "Emma")
    - logicals <- c(FALSE, TRUE, TRUE, FALSE, FALSE, TRUE)

- Show variable content
    - marks

      ```
      [1]  80  75  90  99 100
      ```

    - names

      ```
      [1] "John"    "Jane"    "Richard" "Emma"
      ```

    - logicals

      ```
      [1] FALSE  TRUE  TRUE FALSE FALSE  TRUE
      ```

- Show elements data type
    - class(marks)

      ```
      [1] "numeric"
      ```

    - class(names)

      ```
      [1] "character"
      ```

- class(logicals)

```
[1] "logical"
```

- Give names to elements
  - Full_Name <- c("John", "Doe")
  - names(Full_Name) <- c("First Name", "Last Name")
  - Full_Name

```
First Name  Last Name
    "John"      "Doe"
```

  - name(marks) <- c("John", "Jane", "Richard", "Emma", "Tim")
  - marks

```
   John     Jane  Richard     Emma      Tim
     80       75       90       99      100
```

- Calculation between vectors
  - x <- c(2, 3, 4)
  - y <- c(5, 6, 7)
  - x + y   *# Addition*   => 7 9 11
  - x – y   *# Subtraction*   => -3 -3 -3
  - x * y   *# Multiplication*   => 10 18 28
  - y / x   *# Division*   => 2.50 2.00 1.75
  - y ^ x   *# Exponentiation*   => 25 216 2401
  - y %% x   *# Modulus*   => 1 0 3

- Comparison (<, <=, >, >=, ==, !=) between vectors
  - x < y   => TRUE TRUE TRUE
  - x == y   => FALSE FALSE FALSE

- More useful functions
  - marks

```
   John     Jane  Richard     Emma      Tim
     80       75       90       99      100
```

  - length(marks) # *Number of elements* => 5
  - sum(marks) # *Total* => 444
  - min(marks) # *Minimum* => 75
  - max(marks) # *Maximum* => 100
  - mean(marks) # *Average* => 88.8
  - median(marks) # *Median* => 90
  - sd(marks) # *Standard Deviation* => 11.16692
  - var(marks) # *Variance* => 124.7
  - sort(marks, decreasing = FALSE)

```
   Jane     John  Richard     Emma      Tim
     75       80       90       99      100
```

  - marks1 <- c(80, 75, 90, 99, 100)
  - marks2 <- c(85, 50, 64, 95, 45)
  - cov(marks1, marks2) # *Covariance* => 27.95
  - cor(marks1, marks2) # *Correlation* => 0.1152433

- Accessing specific elements
  - marks

```
   John     Jane  Richard     Emma      Tim
     80       75       90       99      100
```

  - marks[1]

```
John
  80
```

- marks[length(marks)]

```
Tim
100
```

- marks[2:4]

```
    Jane Richard    Emma
      75      90      99
```

- marks[c(1,3,5)]

```
    John Richard     Tim
      80      90     100
```

- marks[c("John", "Jane", "Tim")]

```
John Jane  Tim
  80   75  100
```

Factors

- Categorise unique values, store them as levels
- Store ordinal (with order) or nominal (without rank) categorical variables
  - colour_vector <- c("blue","red","green","green","blue","green","yellow","grey")
  - colour_vector_factor <- factor(colour_vector)
- Show variable content
  - colour_vector_factor

```
[1] blue   red    green  green  blue   green  yellow grey
Levels: blue green grey red yellow
```

  - class(colour_vector_factor)

```
[1] "factor"
```

- Levels rank in alphabetical order for nominal category
- Rename levels
  - customer_satisfaction <- factor(c("L", "M", "L", "L", "H"))
  - customer_satisfaction

```
[1] L M L L H
Levels: H L M
```

  - levels(customer_satisfaction) <- c("High", "Low", "Medium")
  - customer_satisfaction

```
[1] Low    Medium Low    Low    High
Levels: High Low Medium
```

- Explicit ranking order for ordinal category
  - speed <- c("fast", "slow", "slow", "fast", "medium"))
  - speed_factor <- factor(speed, ordered = TRUE, levels = c("slow", "medium", "fast")
  - speed_factor

```
[1] fast   slow   slow   fast   medium
Levels: slow < medium < fast
```

- Comparison for ordinal category
  - speed_factor[1] > speed_factor[2]   => TRUE
  - speed_factor[3] > speed_factor[4]   => FALSE
- Summary

- summary(speed_factor)

```
  slow medium   fast
     2      1      2
```

Matrices

- Two-dimensional array; an extension of vectors
- Elements arranged into rows and columns
  - my_matrix_1 <- matrix(1:12, byrow = TRUE, nrow = 3)
  - my_matrix_1

```
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

  - my_matrix_2 <- matrix(1:12, byrow = FALSE, nrow = 3)
  - my_matrix_2

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

  - my_matrix_3 <- matrix(1:12, byrow = FALSE, nrow = 4)
  - my_matrix_3

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

- Create matrices from a collection of vectors
  - temperature_week_1

```
   Monday   Tuesday Wednesday  Thursday    Friday 
       10        11        12        15        13 
```

- temperature_week_2

```
   Monday   Tuesday Wednesday  Thursday    Friday 
       10         9        13        15        16 
```

- temperature_combined <- c(temperature_week_1, temperature_week_2)
- temperature_combined

```
   Monday   Tuesday Wednesday  Thursday    Friday    Monday   Tuesday Wednesday 
       10        11        12        15        13        10         9        13 
 Thursday    Friday 
       15        16 
```

- temperature_week_1_and_2 <- matrix(temperature_combined, byrow = TRUE, nrow = 2)
- temperature_week_1_and_2

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   11   12   15   13
[2,]   10    9   13   15   16
```

- Naming the rows “rownames()” and columns “colnames()”
  - temperature_week_1_and_2

```
     [,1] [,2] [,3] [,4] [,5]
[1,]   10   11   12   15   13
[2,]   10    9   13   15   16
```

  - weeks <- c(“Week 1”, “Week 2”)
  - weekdays <- c(“Monday”, “Tuesday”, “Wednesday”, “Thursday”, “Friday”)
  - rownames(temperature_week_1_and_2) <- weeks
  - colnames(temperature_week_1_and_2) <- weekdays

- temperature_week_1_and_2

```
       Monday Tuesday Wednesday Thursday Friday
Week 1     10      11        12       15     13
Week 2     10       9        13       15     16
```

- Adding rows “rbind()” and columns “cbind()”
  - earnings_combined

```
     Monday Tuesday Wednesday Thursday Friday
John     50      60        55       74     80
Jane     53      57        79       88     93
```

  - earnings_combined_weekend

```
     Saturday Sunday
John      110    120
Jane      100    130
```

  - earnings_whole_week <- cbind(earnings_combined, earnings_combined_weekend)
  - earnings_whole_week

```
     Monday Tuesday Wednesday Thursday Friday Saturday Sunday
John     50      60        55       74     80      110    120
Jane     53      57        79       88     93      100    130
```

  - earnings_Tim <- c(40, 48, 75, 65, 29, 67, 84)
  - earnings_whole_week <- rbind(earnings_whole_week, earnings_Tim)
  - rownames(earning_whole_week) <- c(“John”, “Jane”, “Tim”)
  - earnings_whole_week

```
     Monday Tuesday Wednesday Thursday Friday Saturday Sunday
John     50      60        55       74     80      110    120
Jane     53      57        79       88     93      100    130
Tim      40      48        75       65     29       67     84
```

- Totalling rows "rowSums()" and columns "colSums()" values
    - earnings_whole_week

```
     Monday Tuesday Wednesday Thursday Friday Saturday Sunday
John     50      60        55       74     80      110    120
Jane     53      57        79       88     93      100    130
Tim      40      48        75       65     29       67     84
```

    - total_earnings_per_day <- colSums(earnings_whole_week)
    - total_earnings_per_day

```
   Monday   Tuesday Wednesday  Thursday    Friday  Saturday    Sunday 
      143       165       209       227       202       277       334 
```

    - total_earnings_per_week <- rowSums(earnings_whole_week)
    - total_earnings_per_week

```
John Jane  Tim 
 549  600  408 
```

- Specific rows or columns
    - John_only <- earnings_whole_week[1,]

```
   Monday   Tuesday Wednesday  Thursday    Friday  Saturday    Sunday 
       50        60        55        74        80       110       120 
```

    - Wednesday_only <- earnings_whole_week[, 3]

```
John Jane  Tim 
  55   79   75 
```

    - John_and_Tim_only <- earnings_whole_week[c(1,3),]

```
     Monday Tuesday Wednesday Thursday Friday Saturday Sunday
John     50      60        55       74     80      110    120
Tim      40      48        75       65     29       67     84
```

    - Thursday_to_Sunday_only <- earnings_whole_week[, 4:7]

```
     Thursday Friday Saturday Sunday
John       74     80      110    120
Jane       88     93      100    130
Tim        65     29       67     84
```

- selection_1 <- earnings_whole_week[c(1,3), 4:7]
- selection_2 <- earnings_whole_week[c("John", "Tim"), c("Thursday", "Friday", "Saturday", "Sunday")]

```
     Thursday Friday Saturday Sunday
John       74     80      110    120
Tim        65     29       67     84
```

- Arithmetic operations
  - mat1 <- matrix(1:4, nrow = 2)

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

  - mat2 <- matrix(5:8, nrow = 2)

```
     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

  - mat1 + mat2

```
     [,1] [,2]
[1,]    6   10
[2,]    8   12
```

  - mat2 – mat1

```
     [,1] [,2]
[1,]    4    4
[2,]    4    4
```

  - mat1 ^ mat2

```
     [,1]  [,2]
[1,]    1  2187
[2,]   64 65536
```

- mat1 * mat2

```
     [,1] [,2]
[1,]    5   21
[2,]   12   32
```

- mat1 / mat2

```
          [,1]      [,2]
[1,] 0.2000000 0.4285714
[2,] 0.3333333 0.5000000
```

- Standard matrix multiplication
  - A <- matrix(c(2,4,6,8), nrow = 2)

```
     [,1] [,2]
[1,]    2    6
[2,]    4    8
```

  - B <- matrix(c(1,3,5,7,9,10), nrow = 2)

```
     [,1] [,2] [,3]
[1,]    1    4    9
[2,]    3    7   10
```

  - AB <- A %*% B

```
     [,1] [,2] [,3]
[1,]   20   50   78
[2,]   28   72  116
```

Arrays

- Multi-dimensional matrices

- **Data** vector is stored in the **dimension** of rows, columns, and matrices
- Syntax: *array(vector, dim = c(rows, columns, matrices))*
    - vector <- c(1, 2, 3, 5, 7, 1, 3, 6, 7, 8, 9, 9)
    - my_array <- array(vector, dim = c(3, 2, 2))
    - my_array

```
, , 1

     [,1] [,2]
[1,]    1    5
[2,]    2    7
[3,]    3    1

, , 2

     [,1] [,2]
[1,]    3    8
[2,]    6    9
[3,]    7    9
```

    - dim(my_array)   => 3 2 2

- Naming array dimensions
    - marks_combined

```
, , 1

     [,1] [,2] [,3]
[1,]   74   78   85
[2,]   72   80   90
[3,]   71   80   72
[4,]   79   88   77
[5,]   90   82   86

, , 2

     [,1] [,2] [,3]
[1,]   63   60   55
[2,]   43   27   72
[3,]   62   74   64
[4,]   85   63   47
[5,]   65   57   75

, , 3

     [,1] [,2] [,3]
[1,]   81   99   80
[2,]   83   77   94
[3,]   90   99   95
[4,]   84   87   87
[5,]   94   91   80
```

- dim(marks_combine) => 5 3 3
- matrix_names <- c("John", "Jane", "Tim")
- row_names <- c("Test 1", "Test 2", "Test 3", "Test 4", "Test 5")
- column_names <- c("Term 1", "Term 2", "Term 3")
- dimnames(marks_combined) <- list(row_names, column_names, matrix_names)
- marks_combined

```
, , John

       Term 1 Term 2 Term 3
Test 1     74     78     85
Test 2     72     80     90
Test 3     71     80     72
Test 4     79     88     77
Test 5     90     82     86

, , Jane

       Term 1 Term 2 Term 3
Test 1     63     60     55
Test 2     43     27     72
Test 3     62     74     64
Test 4     85     63     47
Test 5     65     57     75

, , Tim

       Term 1 Term 2 Term 3
Test 1     81     99     80
Test 2     83     77     94
Test 3     90     99     95
Test 4     84     87     87
Test 5     94     91     80
```

- Accessing array elements
  - marks_combined[, , 1]

```
       Term 1 Term 2 Term 3
Test 1     74     78     85
Test 2     72     80     90
Test 3     71     80     72
Test 4     79     88     77
Test 5     90     82     86
```

  - marks_combined[, , “Tim”]

```
       Term 1 Term 2 Term 3
Test 1     81     99     80
Test 2     83     77     94
Test 3     90     99     95
Test 4     84     87     87
Test 5     94     91     80
```

  - marks_combined[“Test 2”, “Term 2”, c(“John”, “Tim”)]

```
John  Tim
  80   77
```

- Apply calculations on array
  - sum(marks_combined) # *all*

```
[1] 3437
```

  - apply(marks_combined, c(1), sum) # *rows only*

```
Test 1 Test 2 Test 3 Test 4 Test 5
   675    638    707    697    720
```

  - apply(marks_combined, c(2), sum) # *columns only*

```
Term 1 Term 2 Term 3
  1136   1142   1159
```

  - apply(marks_combined, c(1,2), sum) # *both rows and columns*

```
       Term 1 Term 2 Term 3
Test 1    218    237    220
Test 2    198    184    256
Test 3    223    253    231
Test 4    248    238    211
Test 5    249    230    241
```

Lists

- Permit multiple data types
  - my_vector <- 1:10
  - my_matrix <- matrix(c(4, 6, 7, 1), nrow = 2)
  - my_list <- list(my_vector, my_matrix)
  - my_list

```
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10

[[2]]
     [,1] [,2]
[1,]    4    7
[2,]    6    1
```

- my_names <- c("I am a Vector", "I am a Matrix")
- names(my_list) <- my_names
- my_list

```
$`I am a Vector`
 [1]  1  2  3  4  5  6  7  8  9 10

$`I am a Matrix`
     [,1] [,2]
[1,]    4    7
[2,]    6    1
```

- Accessing components in list, by index "[ ]" or by name "$"
  - my_list[[1]]
  - my_list$`I am a Vector`

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

  - my_list$`I am a Matrix`
  - my_list[["I am a Matrix"]]

```
     [,1] [,2]
[1,]    4    7
[2,]    6    1
```

  - my_list[[1]][4]
  - my_list$`I am a Vector`[4]

```
[1] 4
```

  - my_list[[2]][1,]

```
[1] 4 7
```

- my_list[[2]][1,1]

```
[1] 4
```

Data Frames

- Table or two-dimensional array-like structure
- Permit multiple data types, with same data type for each column
- Column contains values of one variable
- Row contains one set of values for each column
    - Student <- c(“Jane”, “John”, “Tim”, “Michael”, “Emma”)
    - Marks <- c(80, 95, 98.5, 78.4, 50.25)
    - Rank <- as.integer(c(3, 2, 1, 4, 5)
    - Passed <- c(TRUE, TRUE, TRUE, TRUE, FALSE)
    - Class_DF <- data.frame(Student, Marks, Rank, Passed)
    - str(Class_DF)

```
'data.frame':   5 obs. of  4 variables:
 $ Student: chr  "Jane" "John" "Tim" "Michael" ...
 $ Marks  : num  80 95 98.5 78.4 50.2
 $ Rank   : int  3 2 1 4 5
 $ Passed : logi  TRUE TRUE TRUE TRUE FALSE
```

- Selecting elements in data frame
    - Class_DF

```
  Student Marks Rank Passed
1    Jane 80.00    3   TRUE
2    John 95.00    2   TRUE
3     Tim 98.50    1   TRUE
4 Michael 78.40    4   TRUE
5    Emma 50.25    5  FALSE
```

  - Class_DF[, 1]
  - Class_DF$Student

```
[1] "Jane"    "John"    "Tim"     "Michael" "Emma"
```

  - Class_DF[1,]

```
  Student Marks Rank Passed
1    Jane    80    3   TRUE
```

  - Class_DF[1:3,]

```
  Student Marks Rank Passed
1    Jane  80.0    3   TRUE
2    John  95.0    2   TRUE
3     Tim  98.5    1   TRUE
```

  - Class_DF[c(2,4),]

```
  Student Marks Rank Passed
2    John  95.0    2   TRUE
4 Michael  78.4    4   TRUE
```

- Selecting a subset based on condition
  - subset(Class_DF, Passed == TRUE)

```
  Student Marks Rank Passed
1    Jane  80.0    3   TRUE
2    John  95.0    2   TRUE
3     Tim  98.5    1   TRUE
4 Michael  78.4    4   TRUE
```

  - subset(Class_DF, Marks > 90)

```
  Student Marks Rank Passed
2    John  95.0    2   TRUE
3     Tim  98.5    1   TRUE
```

- subset(Class_DF, nchar(as.character(Student)) == 4)

```
  Student Marks Rank Passed
1    Jane 80.00    3   TRUE
2    John 95.00    2   TRUE
5    Emma 50.25    5  FALSE
```

6. Data Types in Python

Float and Integer

- Stores real numbers
  - a = 4.6
  - print(type(a))

    ```
    <class 'float'>
    ```

- Stores integers
  - b = 10
  - print(type(b))

    ```
    <class 'int'>
    ```

- Conversion
  - int(a) *# convert float to int* => 4
  - float(b) *# convert int to float* => 10.0
- Basic arithmetic operators
  - 3 + 2 *# Addition* => 5
  - 5 – 2 *# Subtraction* => 3
  - 5 * -2 *# Multiplication* => -10
  - 5 / 2.5 *# Division* => 2.0
  - 2**2 *# Exponentiation* => 4
  - 10 % 3 *# Modulus* => 1
  - 10 // 3 *# Floor Division* => 3

String

- Stores strings
  - phrase = ‘All models are wrong, but some are useful.’

- phrase[0:3] *# slicing character 0 up to 2* => All
- phrase.find('models') *# find the starting index of word* => 4
- phrase.find('right') *# word not found* => -1
- phrase.lower() *# set to lower case* => 'all models are wrong, but some are useful.'
- phrase.upper() *# set to upper case* => 'ALL MODELS ARE WRONG, BUT SOME ARE USEFUL.'
- phrase.split(',') *# split strings into list, base on delimiter* => ['All models are wrong', ' but some are useful.']

Boolean

- Stores logical or **<u>Boolean</u>** values of ***TRUE*** or ***FALSE***
  - k = 1 > 3
  - print(k)

```
False
```

  - print(type(k))

```
<class 'bool'>
```

- Logical operators
  - Conjunction (AND): "**and**"
  - Disjunction (OR): "**or**"
  - Negation (NOT): "**not**"

a	**b**	**a and b**	**a or b**	**not a**
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

7. Data Structures in Python

Tuples

- Store ordered collection of objects
- Immutable: elements cannot be modified, added or deleted
- Written with round brackets "( )"
  - tuple1 = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
  - tuple2 = ("Handsome Koh", 4896, 13.14, True)
- Accessing elements by indexing
  - tuple1[0] # *first element index* => 'apple'
  - tuple1[-1] # *last element index* => 'mango'
  - tuple1[2:5] # *range of elements* => ('cherry', 'orange', 'kiwi')

Lists

- Store ordered collection of objects; mutable
- Written with square brackets "[ ]"
  - list1 = ["apple", "banana", "cherry"]
  - list2 = ["Handsome Koh", 4896, 13.14, True]
- Changing elements
  - list1.append("orange") # *add to last position* => ['apple', 'banana', 'cherry', 'orange']
  - list1[2] = "coconut" # *modify index element* => ['apple', 'banana', 'coconut', 'orange']

- list1.remove("apple") # *delete elements*
  => ['banana', 'coconut', 'orange']
- list1.insert(2, "durian") # *insert element at position*
  => ['banana', 'coconut', 'durian', 'orange']

Sets

- Store unordered, unindexed, nonduplicates collection of objects
- Written with square brackets "{ }"
  - set1 = {"apple", "banana", "cherry"}
  - set2 = {"apple", "samsung"}
- Set operations
  - set1.union(set2) # *Union both sets*
    => {'apple', 'banana', 'cherry', 'samsung'}
  - set1.intersection(set2) # *Intersect both sets*
    => {'apple'}

Dictionaries

- Store unordered collection of objects
- Written with square brackets "{ }", and "key:value" pair
  - thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964}
- Accessing/modifying elements by key name
  - thisdict["model"] => 'Mustang'
  - thisdist["year"] = 2018 thisdist["color"] = "red" => {'brand': 'Ford', 'model': 'Mustang', 'year': 2018, 'color': 'red'}

8. Data Analysis with Python

Numpy

- Python library used for working with arrays
- Performs numerical processing
    - import **numpy** as np

- NumPy arrays (ndarray)
    - array0d = np.array(42) # 0-dimensional array
    - array1d = np.array([1, 2, 3, 4, 5]) # 1-dimensional array
    - print(type(array1d))

```
<class 'numpy.ndarray'>
```

- array2d = np.array([[1, 2, 3], [4, 5, 6]]) # 2-dimensional array
- print(array2d)

```
[[1 2 3]
 [4 5 6]]
```

- Commonly used arrays
    - a = np.zeros((2,2)) # array of all zeros
    - print(a)

```
[[0. 0.]
 [0. 0.]]
```

- b = np.ones((1,2)) # array of all ones
- print(b)

```
[[1. 1.]]
```

- c = np.full((2,2), 7) # constant array
- print(c)

```
[[7 7]
 [7 7]]
```

- d = np.eye(2) # identity matrix
- print(a)

```
[[1. 0.]
 [0. 1.]]
```

- Random arrays
  - np.random.seed(10)
  - e = np.random.random((2,2)) # array with random values
  - print(np.round(e, 3)) # round to 3 d.p.

```
[[0.771 0.021]
 [0.634 0.749]]
```

- More arrays
  - rng = np.arrange(10)
  - print(rng)

```
[0 1 2 3 4 5 6 7 8 9]
```

  - print(np.sqrt(rng))

```
[0.         1.         1.41421356 1.73205081 2.         2.23606798
 2.44948974 2.64575131 2.82842712 3.        ]
```

- NumPy operations
  - a = np.array([[1.0, 2.0, 4.0], [-1.0, 2.0, -5.0]])
  - print(a)

```
[[ 1.  2.  4.]
 [-1.  2. -5.]]
```

  - print(a.shape)

```
(2, 3)
```

  - print(a.sum(axis=0)) *# rows*

```
[ 0.  4. -1.]
```

  - print(a.sum(axis=1)) *# columns*

```
[ 7. -4.]
```

  - print(a.sum()) *# all*

```
3.0
```

  - b = np.transpose(a)
  - print(b)

```
[[ 1. -1.]
 [ 2.  2.]
 [ 4. -5.]]
```

  - print(b.shape)

```
(3, 2)
```

  - print(np.dot(a,b)) *# a * b*

    print(a @ b)

```
[[ 21. -17.]
 [-17.  30.]]
```

- print(a[0:2, 1:3]) *# subset*

```
[[ 2.  4.]
 [ 2. -5.]]
```

- print(np.sum((a<0) & (a>1)) *# and* => 0

  # or "|" => 5

Pandas

- Python library used for data manipulation and analysis
- Work with heterogenous data, as data frame
  - import **pandas** as pd

- Pandas Series, one-dimensional labelled array
- Syntax: "pd.Series(data, index=index)"
  - np.random.seed(1)
  - s = pd.Series(np.random.randn(5), index = ['a', 'b', 'c', 'd', 'e']

```
a    1.624345
b   -0.611756
c   -0.528172
d   -1.072969
e    0.865408
dtype: float64
```

- Pandas Data Frames, two-dimensional labelled data structure
  - d = {'one': [1., 2., 3., 4.], 'two': [4., 3., 2., 1.]}
  - df = pd.DataFrame(d)

	one	two
0	1.0	4.0
1	2.0	3.0
2	3.0	2.0
3	4.0	1.0

- df.describe()

	one	two
count	4.000000	4.000000
mean	2.500000	2.500000
std	1.290994	1.290994
min	1.000000	1.000000
25%	1.750000	1.750000
50%	2.500000	2.500000
75%	3.250000	3.250000
max	4.000000	4.000000

- print(df['one'])

```
0    1.0
1    2.0
2    3.0
3    4.0
Name: one, dtype: float64
```

9. Data-Exchange File in Python

Import Plain Text Files in Python

- Read data from a file
  - text_file = open('Sentence.txt', 'r')

```
Sentence.txt
All models are wrong, but some are useful
```

  - lines = text_file.read()
- Show variable content
  - lines

```
'All models are wrong, but some are useful'
```

Import/Export CSV Files in Python

- Read data as data frame a file
  - import pandas as pd
  - df = pd.read_csv('titanic.csv')

```
titanic.csv
survived,pclass,sex,age,sibsp,parch,fare,embarked,class,who,adult_male,deck,embark_town,alive,alone,
0,3,male,22,1,0,7.25,S,Third,man,TRUE,,Southampton,no,FALSE,
1,1,female,38,1,0,71.2833,C,First,woman,FALSE,C,Cherbourg,yes,FALSE,
1,3,female,26,0,0,7.925,S,Third,woman,FALSE,,Southampton,yes,TRUE,
1,1,female,35,1,0,53.1,S,First,woman,FALSE,C,Southampton,yes,FALSE,
0,3,male,35,0,0,8.05,S,Third,man,TRUE,,Southampton,no,TRUE,
0,3,male,,0,0,8.4583,Q,Third,man,TRUE,,Queenstown,no,TRUE,
0,1,male,54,0,0,51.8625,S,First,man,TRUE,E,Southampton,no,TRUE,
0,3,male,2,3,1,21.075,S,Third,child,FALSE,,Southampton,no,FALSE,
1,3,female,27,0,2,11.1333,S,Third,woman,FALSE,,Southampton,yes,FALSE,
1,2,female,14,1,0,30.0708,C,Second,child,FALSE,,Cherbourg,yes,FALSE,
1,3,female,4,1,1,16.7,S,Third,child,FALSE,G,Southampton,yes,FALSE,
```

- Show variable content
  - df

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True
...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True

A view of the previous CSV file as a pandas data frame.

- Export
  - df.to_csv(r'path', index – False, header = True)

Import/Export JSON Files in Python

- Read data as data frame a file
  - import pandas as pd
  - df = pd.read_json('example.json')

example.json

```
{
        "Product":
                {"0":"Desktop Computer","1":"Tablet","2":"iPhone","3":"Laptop"},
        "Price":
                {"0":700,"1":250,"2":800,"3":1200}
}
```

- Show variable content
  - df

	Product	Price
0	Desktop Computer	700
1	Tablet	250
2	iPhone	800
3	Laptop	1200

- Export
  - df.to_json(r'path)

Import/Export XML Files in Python

- Read data from file
  - import xml.etree.ElementTree as et
  - tree = et.parse(‘example.xml’)

example.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<data>
    <items>
        <item name="item1">item1 abc</item>
        <item name="item2">item2 abc</item>
    </items>
</data>
```

  - root = tree.getroot()
  - print(‘Item #1 attribute: ’, root[0][0].attrib)

```
Item #1 attribute:  {'name': 'item1'}
```

  - print(‘\Item #2 data: ’, root[0][1].text)

```
\Item #2 data:  item2 abc
```

Useful Resources

-
    - http://