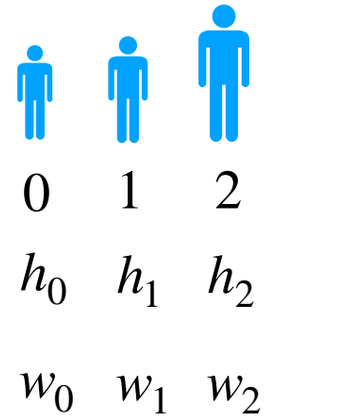
RESHAPING OPERATIONS

Note on Pandas

CONSIDER A SURVEY

- With 3 participants (subjects)
- And 2 measurements (HEIGHT,WEIGHT)=(H,W)



9 peices of information

Example-1:

	ID	H	W
0	ID0	H0	W0
1	ID1	H1	W1
2	ID2	H2	W2

"dummy index"

data

```
print(df1.columns)
Index(['ID', 'H', 'W'], dtype='object')
```

```
print(df1.index)    Int64Index([0, 1, 2], dtype='int64')
```

Example-2:

```
df2.index=df1['ID']
df2=df2.drop(['ID'],axis=1)
print(df2)
```

ID	H	W
ID0	H0	W0
ID1	H1	W1
ID2	H2	W2

data

```
print(df1.columns)

Index(['H', 'W'], dtype='object')
```

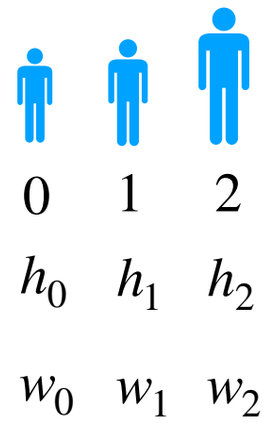
USE CAUTION: PANDAS TENDS TO RE-DEFINE DUMMY INDEX

```
print(df1.index)   Index(['ID0', 'ID1', 'ID2'], dtype='object', name='ID')
```

Long vs wide data

CONSIDER A SURVEY

- With 3 participants (subjects)
- And 2 measurements (HEIGHT,WEIGHT)=(H,W)



9 peices of information

```
-----------
WIDE FORMAT
-----------
    ID   H   W
0  ID0  H0  W0
1  ID1  H1  W1
2  ID2  H2  W2
```

- no entries repeated in first column

- each row only contains ALL measures for a given subject

- more condensed

- more inuitive for averaging (accross row or down column)

```
-----------
LONG FORMAT
-----------
    ID variable value
0  ID0        H    H0
1  ID1        H    H1
2  ID2        H    H2
3  ID0        W    W0
4  ID1        W    W1
5  ID2        W    W2
```

NOTICE THAT NO INFORMATION IS LOST

- each row only contains ONE measure for a given subject

- entries repeat in first column

- less condensed

- easier to add meta data about measurements

What is an observation? one subject (person) or one measurement?

Numpy reshaping

Original: `A=np.array([[11,12,13],[21,22,23]])` →

11 12 13
21 22 23

shape: 2×3

Transformed:

`A.reshape(6,1)` →

11
12
13
21
22
23

shape: 6×1

`A.reshape(1,6)` → 11 12 13 21 22 23 **shape:** 1×6

`A.reshape(3,2)` →

11 12
13 21
22 23

shape: 3×2

Transposing (row --> colums): $A_{ij} = A_{ji}$

```
A.T
np.transpose(A)
```

→

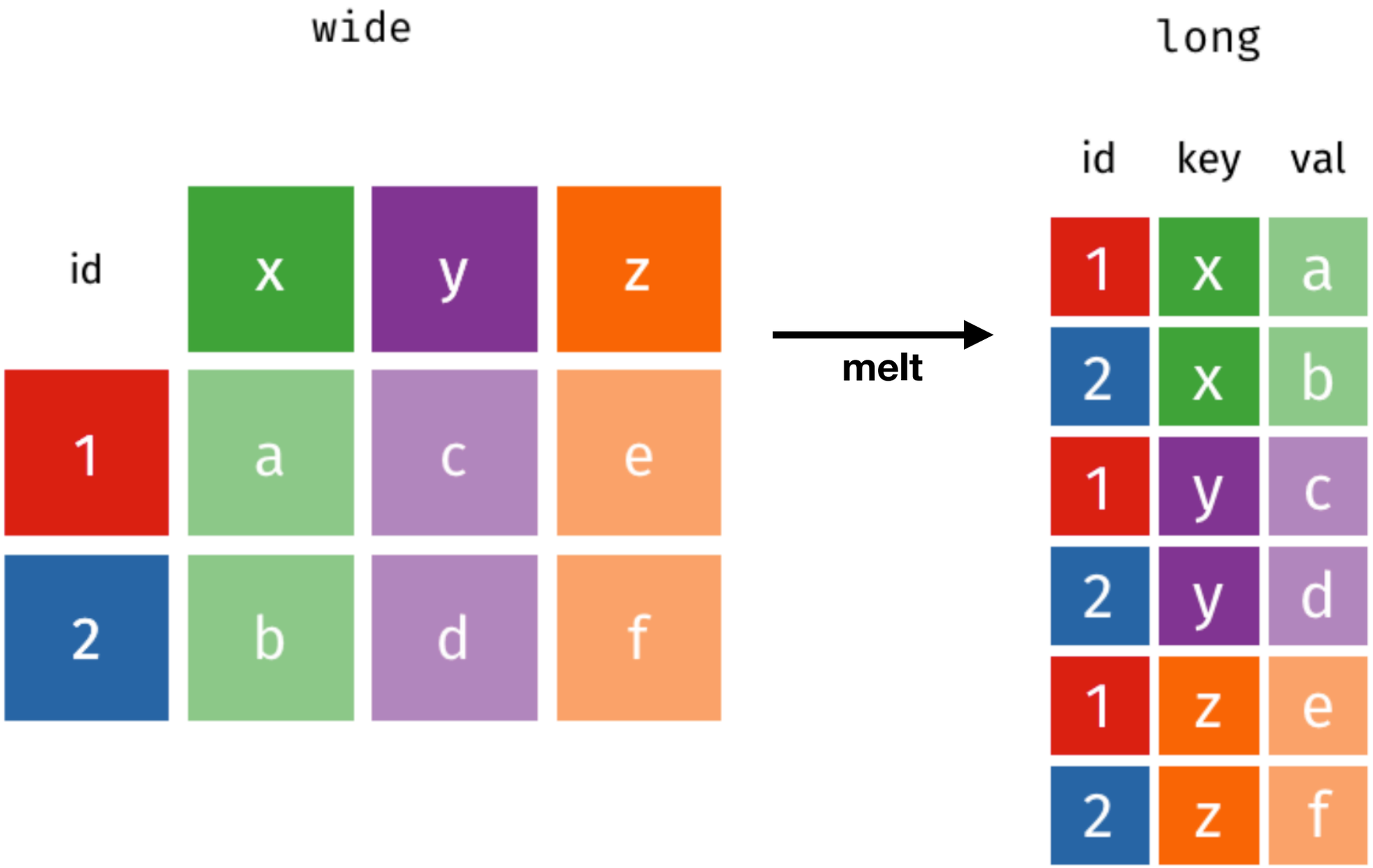
11 12
13 21
22 23

shape: 3×2

Melting

tidyr::pivot_longer/tidyr::gather/pd.melt collects multiple columns into (typically) 2 columns

- One column represents the ID
- One column represents the variable name (key)
- One column represents the values in the column (val)
- All other columns are repeated to keep all the data properly associated



Examples

CONSIDER AGAIN THE SURVEY

- With 3 participants (subjects)
- And 2 measurements (HEIGHT,WEIGHT)=(H,W)



0 1 2

h_0 h_1 h_2

w_0 w_1 w_2

```
-----------
WIDE FORMAT
-----------
    ID   H   W
0  ID0  H0  W0
1  ID1  H1  W1
2  ID2  H2  W2
```

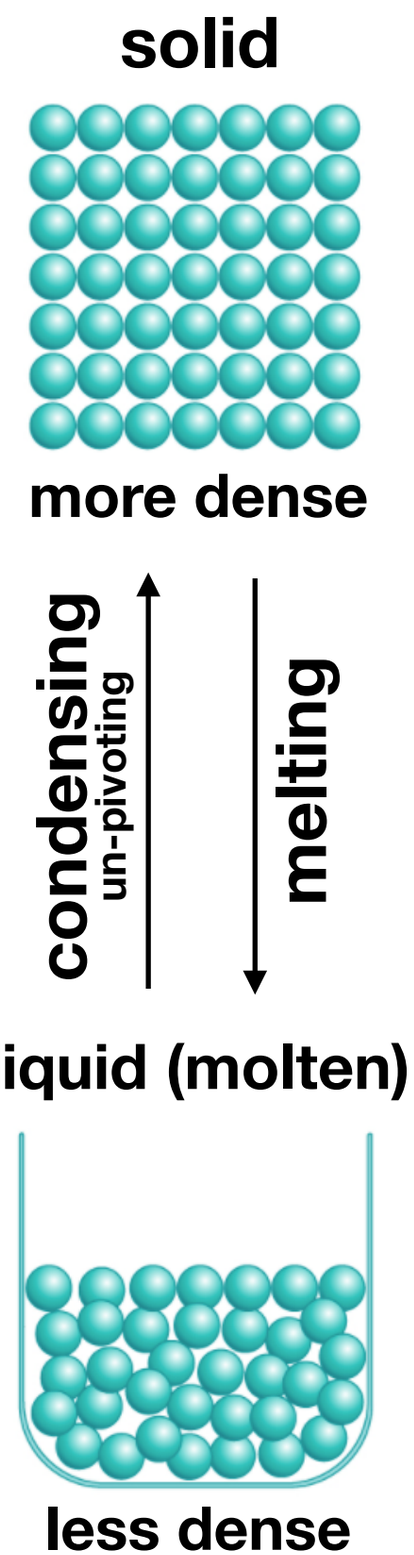
```
import pandas as pd

print("----WIDE FORMAT----")
df = pd.DataFrame(
    {'ID': {0: 'ID0', 1: 'ID1', 2: 'ID2'},
     'H' : {0: "H0" , 1: "H1" , 2: "H2"},
     'W' : {0: "W0" , 1: "W1" , 2: "W2"}
    })
print(df)

print("----LONG FORMAT----")
df=pd.melt(df, id_vars =['ID'])
print(df)
```

Week-02/pandas-examples.py

```
-----------
LONG FORMAT
-----------
    ID variable value
0  ID0        H    H0
1  ID1        H    H1
2  ID2        H    H2
3  ID0        W    W0
4  ID1        W    W1
5  ID2        W    W2
```



Examples

Example-1

```
------ORIGINAL-----
     ID    H    W    A
0   ID0   H0   W0   A0
1   ID1   H1   W1   A1
2   ID2   H2   W2   A2
```

```
df2=pd.melt(df, id_vars =['ID'])
print(df2)
```

```
    ID   variable value
0   ID0         H    H0
1   ID1         H    H1
2   ID2         H    H2
3   ID0         W    W0
4   ID1         W    W1
5   ID2         W    W2
6   ID0         A    A0
7   ID1         A    A1
8   ID2         A    A2
```

Example-2

```
------ORIGINAL-----
     ID    H    W    A
0   ID0   H0   W0   A0
1   ID1   H1   W1   A1
2   ID2   H2   W2   A2
```

```
print(pd.melt(df, id_vars =['H','ID']))
```

```
    H   ID variable value
0   H0  ID0        W    W0
1   H1  ID1        W    W1
2   H2  ID2        W    W2
3   H0  ID0        A    A0
4   H1  ID1        A    A1
5   H2  ID2        A    A2
```

Week-02/pandas-examples.py

Un-Tidy-1: Column headers are values, not variable names

Example-1: Pew solution: melting

religion	<$10k	$10–20k	$20–30k	$30–40k	$40–50k	$50–75k
Agnostic	27	34	60	81	76	137
Atheist	12	27	37	52	35	70
Buddhist	27	21	30	34	33	58
Catholic	418	617	732	670	638	1116
Don't know/refused	15	14	15	11	10	35
Evangelical Prot	575	869	1064	982	881	1486
Hindu	1	9	7	9	11	34
Historically Black Prot	228	244	236	238	197	223
Jehovah's Witness	20	27	24	24	21	30
Jewish	19	19	25	25	30	95

melt →

religion	income	freq
Agnostic	<$10k	27
Agnostic	$10–20k	34
Agnostic	$20–30k	60
Agnostic	$30–40k	81
Agnostic	$40–50k	76
Agnostic	$50–75k	137
Agnostic	$75–100k	122
Agnostic	$100–150k	109
Agnostic	>150k	84
Agnostic	Don't know/refused	96

This dataset explores the relationship between income and religion in the US

This dataset has three variables, religion, income and frequency

unit of analysis="group" NOT religion

Example-2: observational unit="specific version of vechicle"

	red	blue	en-4.0	en-3.0
ford ranger	400	100	X	
ford ranger	450	210		X
ford escort	250	200		X

type_ID	make	model	color	N_made	disp
V1	ford	ranger	red	400	4.0
V2	ford	ranger	blue	100	4.0
V1	ford	ranger	red	450	3.0
V2	ford	ranger	blue	210	3.0
V3	ford	escort	red	250	3.0
V4	ford	escort	blue	200	3.0

ID
group-1
group-2
group-3
⋮

religion	income	freq
Agnostic	<$10k	27
Agnostic	$10–20k	34
Agnostic	$20–30k	60
Agnostic	$30–40k	81
Agnostic	$40–50k	76
Agnostic	$50–75k	137
Agnostic	$75–100k	122
Agnostic	$100–150k	109
Agnostic	>150k	84
Agnostic	Don't know/refused	96

Tidy data: Journal of Statistical Software, Wickham (2014)

Billboard Column headers are values, not variable names

Example-2: Billboard solution: melting then split

year	artist	track	time	date.entered	wk1	wk2	wk3
2000	2 Pac	Baby Don't Cry	4:22	2000-02-26	87	82	72
2000	2Ge+her	The Hardest Part Of ...	3:15	2000-09-02	91	87	92
2000	3 Doors Down	Kryptonite	3:53	2000-04-08	81	70	68
2000	98^0	Give Me Just One Nig...	3:24	2000-08-19	51	39	34
2000	A*Teens	Dancing Queen	3:44	2000-07-08	97	97	96
2000	Aaliyah	I Don't Wanna	4:15	2000-01-29	84	62	51
2000	Aaliyah	Try Again	4:03	2000-03-18	59	53	38
2000	Adams, Yolanda	Open My Heart	5:30	2000-08-26	76	76	74

Table 7: The first eight Billboard top hits for 2000. Other columns not shown are `wk4`, `wk5`, ..., `wk75`.

year	artist	time	track	date	week	rank
2000	2 Pac	4:22	Baby Don't Cry	2000-02-26	1	87
2000	2 Pac	4:22	Baby Don't Cry	2000-03-04	2	82
2000	2 Pac	4:22	Baby Don't Cry	2000-03-11	3	72
2000	2 Pac	4:22	Baby Don't Cry	2000-03-18	4	77
2000	2 Pac	4:22	Baby Don't Cry	2000-03-25	5	87
2000	2 Pac	4:22	Baby Don't Cry	2000-04-01	6	94
2000	2 Pac	4:22	Baby Don't Cry	2000-04-08	7	99
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-02	1	91
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-09	2	87
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-16	3	92
2000	3 Doors Down	3:53	Kryptonite	2000-04-08	1	81
2000	3 Doors Down	3:53	Kryptonite	2000-04-15	2	70
2000	3 Doors Down	3:53	Kryptonite	2000-04-22	3	68
2000	3 Doors Down	3:53	Kryptonite	2000-04-29	4	67
2000	3 Doors Down	3:53	Kryptonite	2000-05-06	5	66

Table 8: First fifteen rows of the tidied Billboard dataset. The `date` column does not appear in the original table, but can be computed from `date.entered` and `week`.

Split into two tables

static attributes of song

id	artist	track	time
1	2 Pac	Baby Don't Cry	4:22
2	2Ge+her	The Hardest Part Of ...	3:15
3	3 Doors Down	Kryptonite	3:53
4	3 Doors Down	Loser	4:24
5	504 Boyz	Wobble Wobble	3:35
6	98^0	Give Me Just One Nig...	3:24
7	A*Teens	Dancing Queen	3:44
8	Aaliyah	I Don't Wanna	4:15
9	Aaliyah	Try Again	4:03
10	Adams, Yolanda	Open My Heart	5:30
11	Adkins, Trace	More	3:05
12	Aguilera, Christina	Come On Over Baby	3:38
13	Aguilera, Christina	I Turn To You	4:00
14	Aguilera, Christina	What A Girl Wants	3:18
15	Alice Deejay	Better Off Alone	6:50

temporally variable attributes of song

id	date	rank
1	2000-02-26	87
1	2000-03-04	82
1	2000-03-11	72
1	2000-03-18	77
1	2000-03-25	87
1	2000-04-01	94
1	2000-04-08	99
2	2000-09-02	91
2	2000-09-09	87
2	2000-09-16	92
3	2000-04-08	81
3	2000-04-15	70
3	2000-04-22	68
3	2000-04-29	67
3	2000-05-06	66

Table 13: Normalized Billboard dataset split up into song dataset (left) and rank dataset (right). First 15 rows of each dataset shown; `genre` omitted from song dataset, `week` omitted from rank dataset.

Python example: Column headers are values, not variable names

```
import pandas as pd
import numpy  as np

print("----------------")
df1 = pd.read_csv('data/pew.csv')
print(df1)

print("----------------")
df2 = df1.melt(id_vars=["religion"])
print(df2)
```

before:

```
----------------
                  religion  <$10k  $10-20k  ...  $100-150k  >150k  Don't know/refused
0                 Agnostic     27       34  ...        109     84                  96
1                  Atheist     12       27  ...         59     74                  76
2                 Buddhist     27       21  ...         39     53                  54
3                 Catholic    418      617  ...        792    633                1489
4       Don’t know/refused     15       14  ...         17     18                 116
5         Evangelical Prot    575      869  ...        723    414                1529
6                    Hindu      1        9  ...         48     54                  37
7   Historically Black Prot    228      244  ...         81     78                 339
8         Jehovah's Witness     20       27  ...         11      6                  37
9                   Jewish     19       19  ...         87    151                 162
10           Mainline Prot    289      495  ...        753    634                1328
11                  Mormon     29       40  ...         49     42                  69
12                  Muslim      6        7  ...          8      6                  22
13                Orthodox     13       17  ...         42     46                  73
14         Other Christian      9        7  ...         14     12                  18
15            Other Faiths     20       33  ...         40     41                  71
16   Other World Religions      5        2  ...          4      4                   8
17             Unaffiliated    217      299  ...        321    258                 597

[18 rows x 11 columns]
```

after:

```
                  religion             variable  value
0                 Agnostic                <$10k     27
1                  Atheist                <$10k     12
2                 Buddhist                <$10k     27
3                 Catholic                <$10k    418
4       Don’t know/refused                <$10k     15
..                     ...                  ...    ...
175               Orthodox  Don't know/refused     73
176         Other Christian  Don't know/refused     18
177            Other Faiths  Don't know/refused     71
178  Other World Religions  Don't know/refused      8
179            Unaffiliated  Don't know/refused    597

[180 rows x 3 columns]
```

Tidy data: Journal of Statistical Software, Wickham (2014)

R example: Column headers are values, not variable names

```
library(tidyverse)

pew <- read_csv('data/pew.csv')
head(pew)

# WE NEED TO MELT THIS DATASET TO MAKE IT TIDY
pew1 <- pew %>% pivot_longer(cols = c(-religion), names_to = 'income_groups', values_to =
'counts')
head(pew1)
```

before:

```
# A tibble: 18 × 11
   religion `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k` `$75-100k`
   <chr>      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>      <dbl>
 1 Agnostic      27        34        60        81        76       137        122
 2 Atheist       12        27        37        52        35        70         73
 3 Buddhist      27        21        30        34        33        58         62
 4 Catholic     418       617       732       670       638      1116        949
 5 Don't k…      15        14        15        11        10        35         21
 6 Evangel…     575       869      1064       982       881      1486        949
 7 Hindu          1         9         7         9        11        34         47
 8 Histori…     228       244       236       238       197       223        131
 9 Jehovah…      20        27        24        24        21        30         15
10 Jewish        19        19        25        25        30        95         69
11 Mainlin…     289       495       619       655       651      1107        939
12 Mormon        29        40        48        51        56       112         85
13 Muslim         6         7         9        10         9        23         16
14 Orthodox      13        17        23        32        32        47         38
15 Other C…       9         7        11        13        13        14         18
16 Other F…      20        33        40        46        49        63         46
17 Other W…       5         2         3         4         2         7          3
18 Unaffil…     217       299       374       365       341       528        407
# … with 3 more variables: $100-150k <dbl>, >150k <dbl>,
#   Don't know/refused <dbl>
```

after:

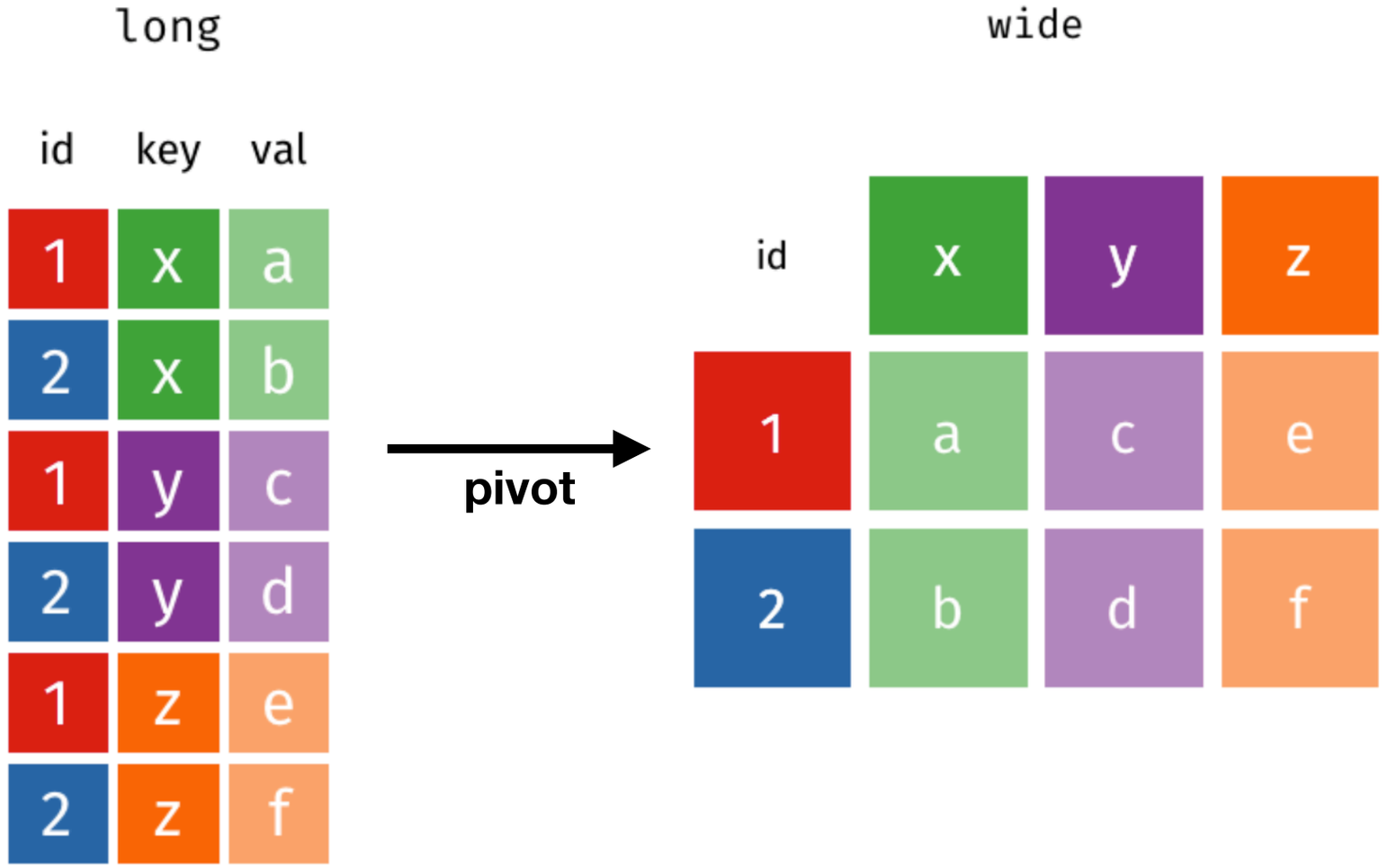
```
# A tibble: 180 × 3
   religion income_groups      counts
   <chr>    <chr>               <dbl>
 1 Agnostic <$10k                  27
 2 Agnostic $10-20k                34
 3 Agnostic $20-30k                60
 4 Agnostic $30-40k                81
 5 Agnostic $40-50k                76
 6 Agnostic $50-75k               137
 7 Agnostic $75-100k              122
 8 Agnostic $100-150k             109
 9 Agnostic >150k                  84
10 Agnostic Don't know/refused     96
# … with 170 more rows
```

Tidy data: Journal of Statistical Software, Wickham (2014)

Pivoting reverse of melt

tidyr::pivot_wider/tidyr::spread/pd.pivot takes two columns and makes them multiple columns

- The values in one column form the headers to different new columns
- The values in the other column represent the values in the corresponding cells
- The other columns are repeated to start with, but reduce repetitions to make all associated data stay together



Example:

```
  ID variable value
0  ID0        H    H0
1  ID1        H    H1
2  ID2        H    H2
3  ID0        W    W0
4  ID1        W    W1
5  ID2        W    W2
```

```
#CONVERT BACK
df2=df2.pivot(*df2).rename_axis(columns =
None).reset_index()
print(df2)
```

```
  ID  A  H  W
0 ID0 A0 H0 W0
1 ID1 A1 H1 W1
2 ID2 A2 H2 W2
```

Splitting columns

Example-1:

```
#-------------------------------
# SPLITTING STRINGS INTO MULTIPLE COLUMN
#-------------------------------
df1 = pd.DataFrame([['m143'], ['f1232']], columns=['A'])
print("ORIGNIAL\n",df1)
```

```
ORIGNIAL
      A
0   m143
1  f1232
```

↓

```
#SPLITTING m143 --> m 143
#Series.str()-- >Vectorized string
# function for Series and Index.
df1["MF"]  = df1.A.str[0]
df1["NUM"] = df1.A.str[1:]
df1=df1.drop('A',axis=1)
print("MODIFIED\n",df1)
```

```
MODIFIED
   MF  NUM
0   m   143
1   f  1232
```

Example-2:

```
#-------------------------------
# SPLITTING A LIST STORED AS A STRING
#-------------------------------
# #Series.str()-- >Vectorized string functions for Series and Index.
df1 = pd.DataFrame([['[1, 2]','[3, 4]'], ['[NA, 6]','[7, 8]']], columns=['A', 'B'])
print("ORIGNIAL-1\n",df1)
```

↓

```
ORIGNIAL
        A        B
0   [1, 2]   [3, 4]
1  [NA, 6]   [7, 8]
```

↓

```
df1[['A1','A2']] = pd.DataFrame(df1.A.str.replace('[','', regex=True).str.replace(']','', regex=True).str.split(',').tolist())
df1[['B1','B2']] = pd.DataFrame(df1.B.str.replace('[','', regex=True).str.replace(']','', regex=True).str.split(',').tolist())
df1=df1.drop(['A','B'],axis=1)
print("MODIFIED-1\n",df1)
```

↓

```
MODIFIED-1
   A1  A2 B1  B2
0   1   2  3   4
1  NA   6  7   8
```

Un-Tidy-2: Multiple variables are stored in one column.

Example:

country	year	m014	m1524	m2534	m3544	m4554	m5564	m65	mu	f014
AD	2000	0	0	1	0	0	0	0	—	—
AE	2000	2	4	4	6	5	12	10	—	3
AF	2000	52	228	183	149	129	94	80	—	93
AG	2000	0	0	0	0	0	0	1	—	1
AL	2000	2	19	21	14	24	19	16	—	3
AM	2000	2	152	130	131	63	26	21	—	1
AN	2000	0	0	1	2	0	0	0	—	0
AO	2000	186	999	1003	912	482	312	194	—	247
AR	2000	97	278	594	402	419	368	330	—	121
AS	2000	—	—	—	—	1	1	—	—	—

country	year	column	cases
AD	2000	m014	0
AD	2000	m1524	0
AD	2000	m2534	1
AD	2000	m3544	0
AD	2000	m4554	0
AD	2000	m5564	0
AD	2000	m65	0
AE	2000	m014	2
AE	2000	m1524	4
AE	2000	m2534	4
AE	2000	m3544	6
AE	2000	m4554	5
AE	2000	m5564	12
AE	2000	m65	10
AE	2000	f014	3

(a) Molten data

country	year	sex	age	cases
AD	2000	m	0–14	0
AD	2000	m	15–24	0
AD	2000	m	25–34	1
AD	2000	m	35–44	0
AD	2000	m	45–54	0
AD	2000	m	55–64	0
AD	2000	m	65+	0
AE	2000	m	0–14	2
AE	2000	m	15–24	4
AE	2000	m	25–34	4
AE	2000	m	35–44	6
AE	2000	m	45–54	5
AE	2000	m	55–64	12
AE	2000	m	65+	10
AE	2000	f	0-14	3

(b) Tidy data

Table 10: Tidying the TB dataset requires first melting, and then splitting the `column` column into two variables: `sex` and `age`.

Tidy data: Journal of Statistical Software, Wickham (2014)

Example Multiple variables are stored in one column.

Week-02/tb.r and Week-02/tb.py

Python:

```python
import pandas as pd

#READ
tb = pd.read_csv('data/tb.csv')
print("ORIGINAL \n", tb.head(3))

#MELT
tb1 = tb.melt(id_vars = ['iso2','year'], var_name = 'demo', value_name='counts')
print("AFTER MELT \n", tb1.head(3))

#SPLIT
tb2 = (tb1.assign(                                  # create new columns
  gender = lambda x: x.demo.str[0].astype(str),
  age = lambda x: x.demo.str[1:].astype(str))
  .drop('demo', axis=1)                              # Remove old column
)
print("AFTER SPLIT \n", tb1.head(3))

print(tb2.head(3))
```

R:

```r
library("tidyverse")

#READ
tb <- read_csv('data/tb.csv')
print("ORIGINAL")
head(tb, 3)

#MELT
tb1 <- tb %>% pivot_longer(cols=m014:f65, names_to='demo', values_to='counts')
print("AFTER MELT")
head(tb1, 3)

#SPLIT
tb2 <- tb1 %>% separate(demo, c('gender','age'), sep = 1)
print("AFTER SPLIT ")
print(head(tb2, 3))
```

open code and run

Un-Tidy-3: Variables are stored in both rows and columns

Example:

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax	—	—	—	—	—	—	—	—
MX17004	2010	1	tmin	—	—	—	—	—	—	—	—
MX17004	2010	2	tmax	—	27.3	24.1	—	—	—	—	—
MX17004	2010	2	tmin	—	14.4	14.4	—	—	—	—	—
MX17004	2010	3	tmax	—	—	—	—	32.1	—	—	—
MX17004	2010	3	tmin	—	—	—	—	14.2	—	—	—
MX17004	2010	4	tmax	—	—	—	—	—	—	—	—
MX17004	2010	4	tmin	—	—	—	—	—	—	—	—
MX17004	2010	5	tmax	—	—	—	—	—	—	—	—
MX17004	2010	5	tmin	—	—	—	—	—	—	—	—

Table 11: Original weather dataset. There is a column for each possible day in the month. Columns d9 to d31 have been omitted to conserve space.

id	date	element	value
MX17004	2010-01-30	tmax	27.8
MX17004	2010-01-30	tmin	14.5
MX17004	2010-02-02	tmax	27.3
MX17004	2010-02-02	tmin	14.4
MX17004	2010-02-03	tmax	24.1
MX17004	2010-02-03	tmin	14.4
MX17004	2010-02-11	tmax	29.7
MX17004	2010-02-11	tmin	13.4
MX17004	2010-02-23	tmax	29.9
MX17004	2010-02-23	tmin	10.7

(a) Molten data

id	date	tmax	tmin
MX17004	2010-01-30	27.8	14.5
MX17004	2010-02-02	27.3	14.4
MX17004	2010-02-03	24.1	14.4
MX17004	2010-02-11	29.7	13.4
MX17004	2010-02-23	29.9	10.7
MX17004	2010-03-05	32.1	14.2
MX17004	2010-03-10	34.5	16.8
MX17004	2010-03-16	31.1	17.6
MX17004	2010-04-27	36.3	16.7
MX17004	2010-05-27	33.2	18.2

(b) Tidy data

Table 12: (a) Molten weather dataset. This is almost tidy, but instead of values, the `element` column contains names of variables. Missing values are dropped to conserve space. (b) Tidy weather dataset. Each row represents the meteorological measurements for a single day. There are two measured variables, minimum (`tmin`) and maximum (`tmax`) temperature; all other variables are fixed.

Tidy data: Journal of Statistical Software, Wickham (2014)

Un-Tidy-4: Multiple types in one table

Example:

year	artist	track	time	date.entered	wk1	wk2	wk3
2000	2 Pac	Baby Don't Cry	4:22	2000-02-26	87	82	72
2000	2Ge+her	The Hardest Part Of ...	3:15	2000-09-02	91	87	92
2000	3 Doors Down	Kryptonite	3:53	2000-04-08	81	70	68
2000	98^0	Give Me Just One Nig...	3:24	2000-08-19	51	39	34
2000	A*Teens	Dancing Queen	3:44	2000-07-08	97	97	96
2000	Aaliyah	I Don't Wanna	4:15	2000-01-29	84	62	51
2000	Aaliyah	Try Again	4:03	2000-03-18	59	53	38
2000	Adams, Yolanda	Open My Heart	5:30	2000-08-26	76	76	74

Table 7: The first eight Billboard top hits for 2000. Other columns not shown are `wk4`, `wk5`, ..., `wk75`.

year	artist	time	track	date	week	rank
2000	2 Pac	4:22	Baby Don't Cry	2000-02-26	1	87
2000	2 Pac	4:22	Baby Don't Cry	2000-03-04	2	82
2000	2 Pac	4:22	Baby Don't Cry	2000-03-11	3	72
2000	2 Pac	4:22	Baby Don't Cry	2000-03-18	4	77
2000	2 Pac	4:22	Baby Don't Cry	2000-03-25	5	87
2000	2 Pac	4:22	Baby Don't Cry	2000-04-01	6	94
2000	2 Pac	4:22	Baby Don't Cry	2000-04-08	7	99
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-02	1	91
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-09	2	87
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-16	3	92
2000	3 Doors Down	3:53	Kryptonite	2000-04-08	1	81
2000	3 Doors Down	3:53	Kryptonite	2000-04-15	2	70
2000	3 Doors Down	3:53	Kryptonite	2000-04-22	3	68
2000	3 Doors Down	3:53	Kryptonite	2000-04-29	4	67
2000	3 Doors Down	3:53	Kryptonite	2000-05-06	5	66

id	artist	track	time
1	2 Pac	Baby Don't Cry	4:22
2	2Ge+her	The Hardest Part Of ...	3:15
3	3 Doors Down	Kryptonite	3:53
4	3 Doors Down	Loser	4:24
5	504 Boyz	Wobble Wobble	3:35
6	98^0	Give Me Just One Nig...	3:24
7	A*Teens	Dancing Queen	3:44
8	Aaliyah	I Don't Wanna	4:15
9	Aaliyah	Try Again	4:03
10	Adams, Yolanda	Open My Heart	5:30
11	Adkins, Trace	More	3:05
12	Aguilera, Christina	Come On Over Baby	3:38
13	Aguilera, Christina	I Turn To You	4:00
14	Aguilera, Christina	What A Girl Wants	3:18
15	Alice Deejay	Better Off Alone	6:50

id	date	rank
1	2000-02-26	87
1	2000-03-04	82
1	2000-03-11	72
1	2000-03-18	77
1	2000-03-25	87
1	2000-04-01	94
1	2000-04-08	99
2	2000-09-02	91
2	2000-09-09	87
2	2000-09-16	92
3	2000-04-08	81
3	2000-04-15	70
3	2000-04-22	68
3	2000-04-29	67
3	2000-05-06	66

Table 13: Normalized Billboard dataset split up into song dataset (left) and rank dataset (right). First 15 rows of each dataset shown; `genre` omitted from song dataset, `week` omitted from rank dataset.

Datasets often involve values collected at multiple levels, on different types of observational units

During tidying, each type of observational unit should be stored in its own table.

TRANSFORMING DATA

Transformations

We'll consider four main transformations

Verb (tidyverse)	Functionality
mutate	Transform a column with some function
select	Select some columns in the data
filter	Keep only rows that meet some data criterion
arrange	Order the data frame by values of a column(s)

Tidy data: Journal of Statistical Software, Wickham (2014)

- Filter: subsetting or removing observations based on some condition.
- Transform: adding or modifying variables. These modifications can involve either a single variable (e.g., log-transformation), or multiple variables (e.g., computing density from weight and volume).
- Aggregate: collapsing multiple values into a single value (e.g., by summing or taking means).
- Sort: changing the order of observations.

These aren't explicity verbs in Python, but operationally they are similar to base R (non-tidyverse) manipulations

EXAMPLE:

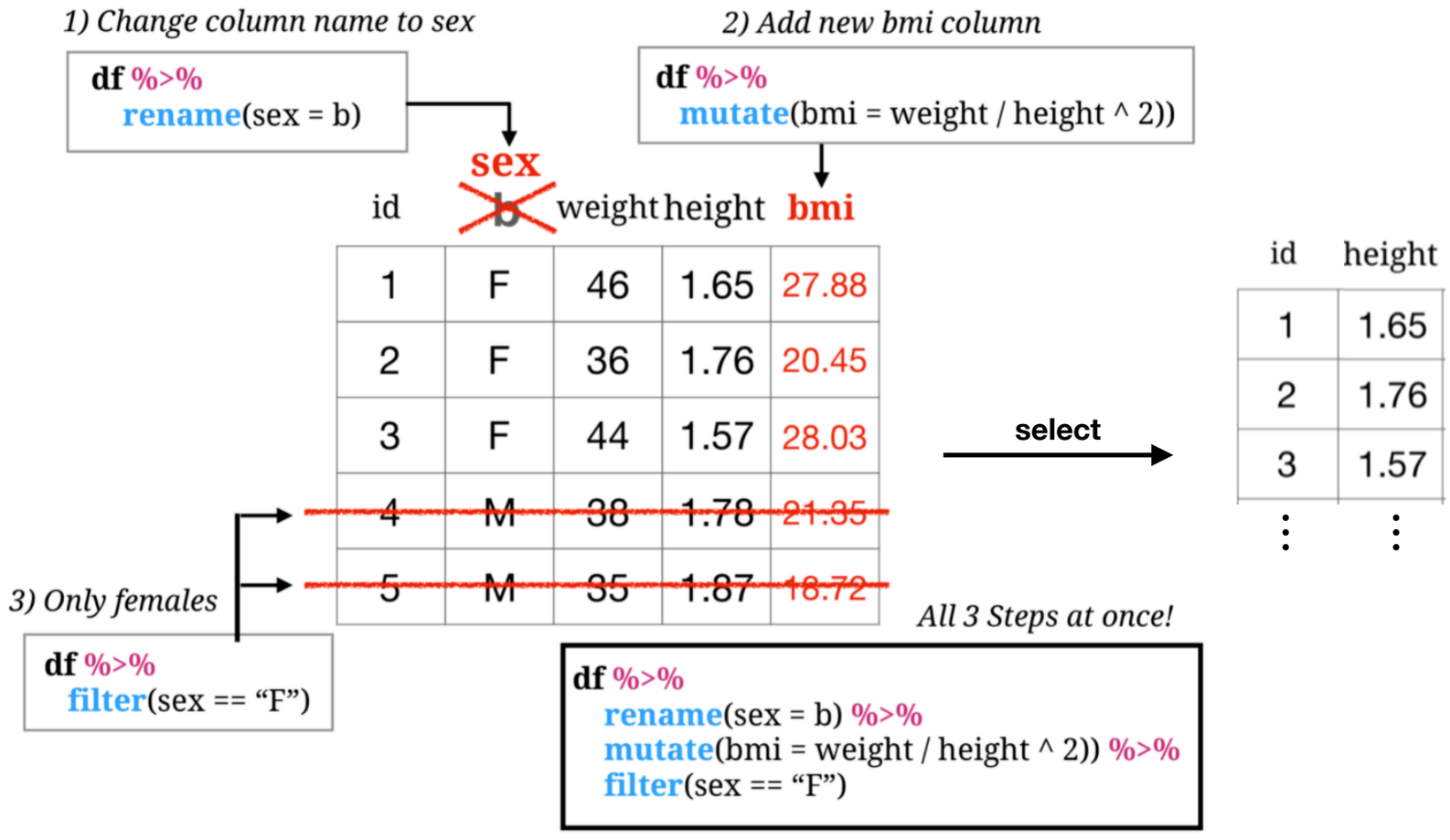


IMAGE SOURCE: https://therbootcamp.github.io/Erfurt_2018June/_sessions/D1S2_Wrangling/Wrangling_practical.html

MPG: PD example mpg.py

```
import pandas as pd

#-----------------------------
print("----READ----")
#-----------------------------
mpg = pd.read_csv('data/mpg.csv')
print(mpg.head(10))
```

original →

```
----READ----
   Unnamed: 0 manufacturer       model  displ  year  cyl       trans drv  cty  hwy fl    class
0           1          audi          a4    1.8  1999    4    auto(l5)   f   18   29  p  compact
1           2          audi          a4    1.8  1999    4  manual(m5)   f   21   29  p  compact
2           3          audi          a4    2.0  2008    4  manual(m6)   f   20   31  p  compact
3           4          audi          a4    2.0  2008    4    auto(av)   f   21   30  p  compact
4           5          audi          a4    2.8  1999    6    auto(l5)   f   16   26  p  compact
5           6          audi          a4    2.8  1999    6  manual(m5)   f   18   26  p  compact
6           7          audi          a4    3.1  2008    6    auto(av)   f   18   27  p  compact
7           8          audi  a4 quattro    1.8  1999    4  manual(m5)   4   18   26  p  compact
8           9          audi  a4 quattro    1.8  1999    4    auto(l5)   4   16   25  p  compact
9          10          audi  a4 quattro    2.0  2008    4  manual(m6)   4   20   28  p  compact
```

```
#-----------------------------
print("----MUTATE----")
#-----------------
# mile/gallon --> km/l
mpg['cty'] = mpg['cty'] * 1.609/3.7854
mpg['hwy'] = mpg['hwy'] * 1.609/3.7854
mpg['avg'] = (mpg.cty + mpg.hwy)/2
print(mpg.head())
```

→

```
----MUTATE----
   Unnamed: 0 manufacturer model  displ  year  cyl       trans drv       cty       hwy fl    class       avg
0           1         audi    a4    1.8  1999    4    auto(l5)   f  7.650975  12.326571  p  compact  9.988773
1           2         audi    a4    1.8  1999    4  manual(m5)   f  8.926137  12.326571  p  compact 10.626354
2           3         audi    a4    2.0  2008    4  manual(m6)   f  8.501083  13.176679  p  compact 10.838881
3           4         audi    a4    2.0  2008    4    auto(av)   f  8.926137  12.751625  p  compact 10.838881
4           5         audi    a4    2.8  1999    6    auto(l5)   f  6.800866  11.051408  p  compact  8.926137
```

```
#-----------------------------
print("----SELECT----")
#-----------------------------
print(mpg[['year','manufacturer','avg']])
```

→

```
----SELECT----
     year manufacturer        avg
0    1999         audi   9.988773
1    1999         audi  10.626354
2    2008         audi  10.838881
3    2008         audi  10.838881
4    1999         audi   8.926137
```

```
#-----------------------------
print("----FILTER----")
#-----------------------------
print(mpg[mpg.manufacturer=='audi'])
```

→

```
----FILTER----
   Unnamed: 0 manufacturer       model  displ  year  cyl       trans drv       cty       hwy fl    class       avg
0           1         audi          a4    1.8  1999    4    auto(l5)   f  7.650975  12.326571  p  compact  9.988773
1           2         audi          a4    1.8  1999    4  manual(m5)   f  8.926137  12.326571  p  compact 10.626354
2           3         audi          a4    2.0  2008    4  manual(m6)   f  8.501083  13.176679  p  compact 10.838881
3           4         audi          a4    2.0  2008    4    auto(av)   f  8.926137  12.751625  p  compact 10.838881
4           5         audi          a4    2.8  1999    6    auto(l5)   f  6.800866  11.051408  p  compact  8.926137
5           6         audi          a4    2.8  1999    6  manual(m5)   f  7.650975  11.051408  p  compact  9.351191
6           7         audi          a4    3.1  2008    6    auto(av)   f  7.650975  11.476462  p  compact  9.563718
7           8         audi  a4 quattro    1.8  1999    4  manual(m5)   4  7.650975  11.051408  p  compact  9.351191
```

```
#-----------------------------
print("----ARRANGE----")
#-----------------------------
print(mpg.sort_values(by=['avg'],
ascending=False))
```

→

```
----ARRANGE----
     Unnamed: 0 manufacturer            model  displ  year  cyl       trans drv       cty       hwy fl       class       avg
221         222   volkswagen       new beetle    1.9  1999    4  manual(m5)   f  14.876895  18.702383  d  subcompact 16.789639
212         213   volkswagen            jetta    1.9  1999    4  manual(m5)   f  14.026787  18.702383  d     compact 16.364585
222         223   volkswagen       new beetle    1.9  1999    4    auto(l4)   f  12.326571  17.427220  d  subcompact 14.876895
196         197       toyota          corolla    1.8  2008    4  manual(m5)   f  11.901516  15.727004  r     compact 13.814260
195         196       toyota          corolla    1.8  1999    4  manual(m5)   f  11.051408  14.876895  r     compact 12.964152
```

MPG: R example

mpg.r

```r
library(tidyverse)

#----------------------------
print("----READ----")
#----------------------------
mpg1 <- read_csv('data/mpg.csv')
head(mpg1)

#----------------------------
print("----MUTATE----")
#----------------------------
# OPTION-1: SEND MPG TO MUTATE
 mpg2 <- mutate(mpg1, cty = cty * 1.6/3.8,
         hwy = hwy * 1.6/3.8,
         avg = (cty + hwy)/2)
head(mpg2); #print(mpg2['avg'])

#----------------------------
print("----MUTATE----")
#----------------------------
#OPTION-2: PIPE MPG INTO MUTATE
mpg1 <- mpg1 %>% mutate(cty = cty * 1.6/3.8,
         hwy = hwy * 1.6/3.8,
         avg = (cty + hwy)/2)
head(mpg1)

#----------------------------
print("----SELECT----")
#----------------------------
mpg2 <- mpg1 %>% select(year, manufacturer, avg)
head(mpg2)

#----------------------------
print("----FILTER----")
#----------------------------
mpg2 <- mpg1 %>% filter(manufacturer=='audi')
head(mpg2)

#----------------------------
print("----ARRANGE----")
#----------------------------
mpg2 <-mpg1 %>% arrange(-year)
head(mpg2)

#----------------------------
print("----FILTER THEN ARRANGE----")
#----------------------------
#CHAIN TOGETHER
mpg2 <-mpg1 %>% filter(manufacturer=='audi') %>%
arrange(-year)
head(mpg2)
```

OUTPUT

```
# A tibble: 6 × 12
   ...1 manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
  <dbl> <chr>        <chr> <dbl> <dbl> <dbl> <chr> <chr> <dbl> <dbl> <chr> <chr>
1     1 audi         a4      1.8  1999     4 auto… f        18    29 p     comp…
2     2 audi         a4      1.8  1999     4 manu… f        21    29 p     comp…
3     3 audi         a4      2    2008     4 manu… f        20    31 p     comp…
4     4 audi         a4      2    2008     4 auto… f        21    30 p     comp…
5     5 audi         a4      2.8  1999     6 auto… f        16    26 p     comp…
6     6 audi         a4      2.8  1999     6 manu… f        18    26 p     comp…
[1] "----MUTATE----"
# A tibble: 6 × 13
   ...1 manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
  <dbl> <chr>        <chr> <dbl> <dbl> <dbl> <chr> <chr> <dbl> <dbl> <chr> <chr>
1     1 audi         a4      1.8  1999     4 auto… f      7.58  12.2 p     comp…
2     2 audi         a4      1.8  1999     4 manu… f      8.84  12.2 p     comp…
3     3 audi         a4      2    2008     4 manu… f      8.42  13.1 p     comp…
4     4 audi         a4      2    2008     4 auto… f      8.84  12.6 p     comp…
5     5 audi         a4      2.8  1999     6 auto… f      6.74  10.9 p     comp…
6     6 audi         a4      2.8  1999     6 manu… f      7.58  10.9 p     comp…
# … with 1 more variable: avg <dbl>
[1] "----MUTATE----"
# A tibble: 6 × 13
   ...1 manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
  <dbl> <chr>        <chr> <dbl> <dbl> <dbl> <chr> <chr> <dbl> <dbl> <chr> <chr>
1     1 audi         a4      1.8  1999     4 auto… f      7.58  12.2 p     comp…
2     2 audi         a4      1.8  1999     4 manu… f      8.84  12.2 p     comp…
3     3 audi         a4      2    2008     4 manu… f      8.42  13.1 p     comp…
4     4 audi         a4      2    2008     4 auto… f      8.84  12.6 p     comp…
5     5 audi         a4      2.8  1999     6 auto… f      6.74  10.9 p     comp…
6     6 audi         a4      2.8  1999     6 manu… f      7.58  10.9 p     comp…
# … with 1 more variable: avg <dbl>
[1] "----SELECT----"
# A tibble: 6 × 3
   year manufacturer   avg
  <dbl> <chr>        <dbl>
1  1999 audi          9.89
2  1999 audi         10.5
3  2008 audi         10.7
4  2008 audi         10.7
5  1999 audi          8.84
6  1999 audi          9.26
[1] "----FILTER----"
# A tibble: 6 × 13
   ...1 manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
  <dbl> <chr>        <chr> <dbl> <dbl> <dbl> <chr> <chr> <dbl> <dbl> <chr> <chr>
1     1 audi         a4      1.8  1999     4 auto… f      7.58  12.2 p     comp…
2     2 audi         a4      1.8  1999     4 manu… f      8.84  12.2 p     comp…
3     3 audi         a4      2    2008     4 manu… f      8.42  13.1 p     comp…
4     4 audi         a4      2    2008     4 auto… f      8.84  12.6 p     comp…
5     5 audi         a4      2.8  1999     6 auto… f      6.74  10.9 p     comp…
6     6 audi         a4      2.8  1999     6 manu… f      7.58  10.9 p     comp…
# … with 1 more variable: avg <dbl>
[1] "----ARRANGE----"
# A tibble: 6 × 13
   ...1 manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
  <dbl> <chr>        <chr> <dbl> <dbl> <dbl> <chr> <chr> <dbl> <dbl> <chr> <chr>
1     3 audi         a4      2    2008     4 manu… f      8.42  13.1 p     comp…
2     4 audi         a4      2    2008     4 auto… f      8.84  12.6 p     comp…
3     7 audi         a4      3.1  2008     6 auto… f      7.58  11.4 p     comp…
4    10 audi         a4 q…   2    2008     4 manu… 4      8.42  11.8 p     comp…
5    11 audi         a4 q…   2    2008     4 auto… 4      8     11.4 p     comp…
6    14 audi         a4 q…   3.1  2008     6 auto… 4      7.16  10.5 p     comp…
# … with 1 more variable: avg <dbl>
[1] "----FILTER THEN ARRANGE----"
# A tibble: 6 × 13
   ...1 manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
  <dbl> <chr>        <chr> <dbl> <dbl> <dbl> <chr> <chr> <dbl> <dbl> <chr> <chr>
1     3 audi         a4      2    2008     4 manu… f      8.42  13.1 p     comp…
2     4 audi         a4      2    2008     4 auto… f      8.84  12.6 p     comp…
3     7 audi         a4      3.1  2008     6 auto… f      7.58  11.4 p     comp…
4    10 audi         a4 q…   2    2008     4 manu… 4      8.42  11.8 p     comp…
5    11 audi         a4 q…   2    2008     4 auto… 4      8     11.4 p     comp…
6    14 audi         a4 q…   3.1  2008     6 auto… 4      7.16  10.5 p     comp…
# … with 1 more variable: avg <dbl>
```

Changing data types

Datatypes:

R	Python
numeric	float
integer	int
character	str
factor	category

Both R and Python have the same data types.
The Categorical type below is unique to **pandas**

Categoricals are a pandas data type corresponding to categorical variables in statistics. A categorical variable takes on a limited, and usually fixed, number of possible values

Examples: gender, social class, blood type, country affiliation

R:

```
#----------------------------
#CONVERT TO FACTOR
#----------------------------
mpg2['model']=as.factor(mpg2$model)
head(mpg2)
```

```
   ...1 manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
  <dbl> <chr>        <chr> <dbl> <dbl> <dbl> <chr> <chr> <dbl> <dbl> <chr> <chr>
1     3 audi         a4      2    2008     4 manu… f      8.42  13.1 p     comp…
2     4 audi         a4      2    2008     4 auto… f      8.84  12.6 p     comp…
3     7 audi         a4      3.1  2008     6 auto… f      7.58  11.4 p     comp…
# … with 1 more variable: avg <dbl>
```

```
# A tibble: 3 × 13
   ...1 manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
  <dbl> <chr>        <fct> <dbl> <dbl> <dbl> <chr> <chr> <dbl> <dbl> <chr> <chr>
1     3 audi         a4      2    2008     4 manu… f      8.42  13.1 p     comp…
2     4 audi         a4      2    2008     4 auto… f      8.84  12.6 p     comp…
3     7 audi         a4      3.1  2008     6 auto… f      7.58  11.4 p     comp…
```

Python:

```
#----------------------------
print("----READ----")
#----------------------------
mpg = pd.read_csv('data/mpg.csv')
print(mpg.head(10))

#----------------------------
print("----CONVERT----")
#----------------------------
print(mpg.dtypes)
mpg['manufacturer']=mpg.manufacturer.astype('category')
mpg['model']=mpg.model.astype('string')
print('--------')
print(mpg.dtypes)
```

```
----CONVERT----
Unnamed: 0          int64
manufacturer       object
model              object
displ             float64
year                int64
.
.
--------
Unnamed: 0          int64
manufacturer    category
model             string
displ             float64
year                int64
.
.
```

JOINING DATASETS

- Quite often, data on individuals lie in different tables
  - Clinical, demographic and bioinformatic data
  - Drug, procedure, and payment data (think Medicare)
  - Personal health data across different healthcare entities

Pandas Concatenating

The simplest case is when we just need to add more data to existing data

Concatinating Series:

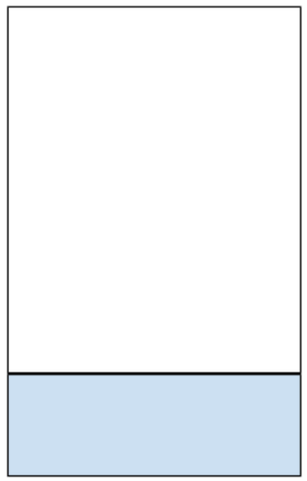
```
import pandas as pd

#CONCATINATING SERIES
s1 = pd.Series(['a', 'b'])
s2 = pd.Series(['c', 'd'])
print(s1)
print("----CONCAT SERIES---")
print(pd.concat([s1, s2],axis=0))
print(pd.concat([s1, s2],axis=1))
```

```
----CONCAT SERIES---
0    a
1    b
dtype: object
0    a
1    b
0    c
1    d
dtype: object
   0  1
0  a  c
1  b  d
```

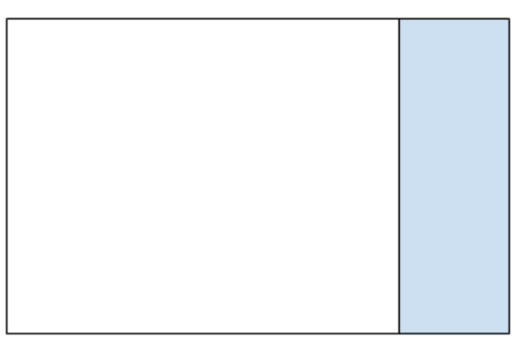
Concatinating dataframes:

Pandas: pd.concat(x,y, axis=0)



Add rows

Pandas: pd.concat(x,y, axis=1)



Add columns

```
import pandas as pd

#CONCATINATING DATA FRAMES
print("----CONCAT DATAFRAMES---")
df1 = pd.DataFrame([['a', 1], ['b', 2]], columns=['letter', 'number'])
df2 = pd.DataFrame([['c', 3], ['d', 4]], columns=['letter', 'number'])
print("INPUTS:")
print(df1)
print(df2)
print("OUTPUTS:")
print(pd.concat([df1, df2],axis=0))
print(pd.concat([df1, df2],axis=1))
```

```
----CONCAT DATAFRAMES---
INPUTS:
  letter  number
0      a       1
1      b       2
  letter  number
0      c       3
1      d       4
```

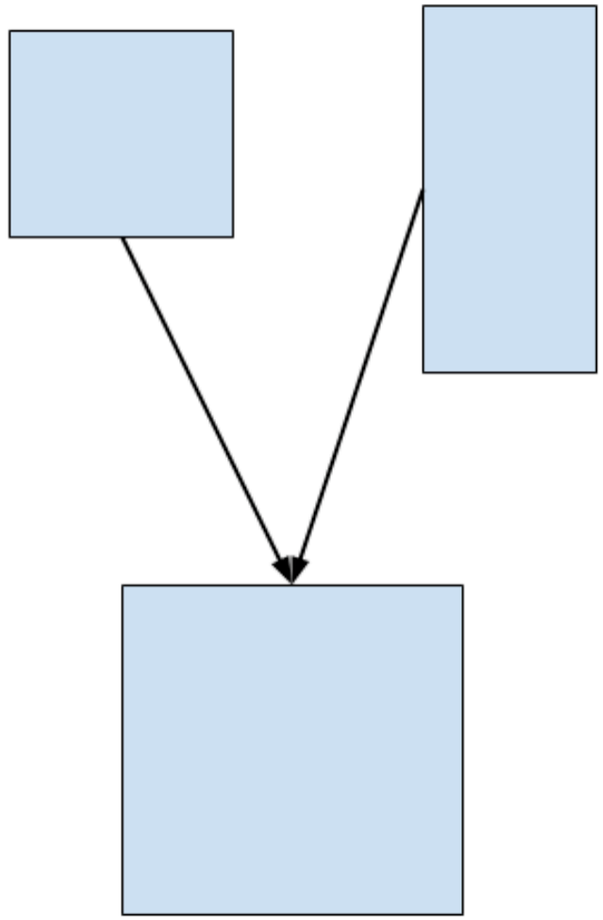
```
OUTPUTS:
  letter  number
0      a       1
1      b       2
0      c       3
1      d       4
  letter  number letter  number
0      a       1      c       3
1      b       2      d       4
```

Joins

We will talk about more general ways of joining two datasets

We will assume:

1. We have two rectangular data sets (so data.frame or tibble)
2. **There is at least one variable (column) in common**, even if they have different names
   - Patient ID number
   - SSN (Social Security number)
   - Identifiable information

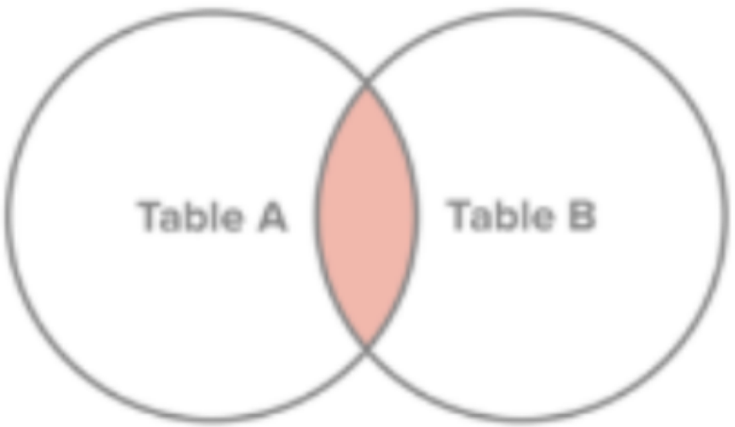


The **"join condition" are the common variables in the two datasets**, i.e. rows are selected if the values of the common variables in the left dataset matches the values of the common variables in the right dataset

Different types of joins

pandas	R	SQL	Description
left	left_join	left outer	keep all rows on left
right	right_join	right outer	keep all rows on right
outer	full_join	full outer	keep all rows from both
inner	inner_join	inner	keep only rows with common keys

Inner Joins

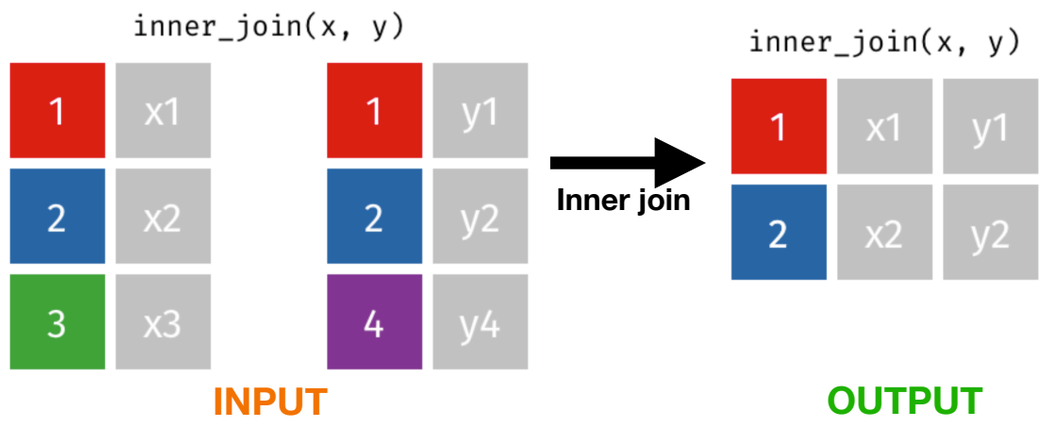


Select all records from Table A and Table B, where the join condition is met.

intersection of two sets A and B, denoted by $A \cap B$

$$A \cap B = \{x : x \in A \textbf{ and } x \in B\}.$$

$$\{1,2,3\} \cap \{2,3,4\} = \{2,3\} \qquad \{1,2,3\} \cap \{4,5,6\} = \emptyset$$



EXAMPLE:

```
print("----JOINING DATAFRAMES---")
df1 = pd.DataFrame([['x1', 'y1'], ['x2', 'y2'], ['x3', 'y3']], columns=['x', 'y'])
df2 = pd.DataFrame([['x1', 'y1','w1', 'z1']
                  , [‘x2', 'y2', 'w2', 'z2']
                  , [‘x4', 'y4', 'w4', 'z4']],columns=['x', 'y','w', 'z'])
print("INPUTS:")
print(df1,'\n')
print(df2)

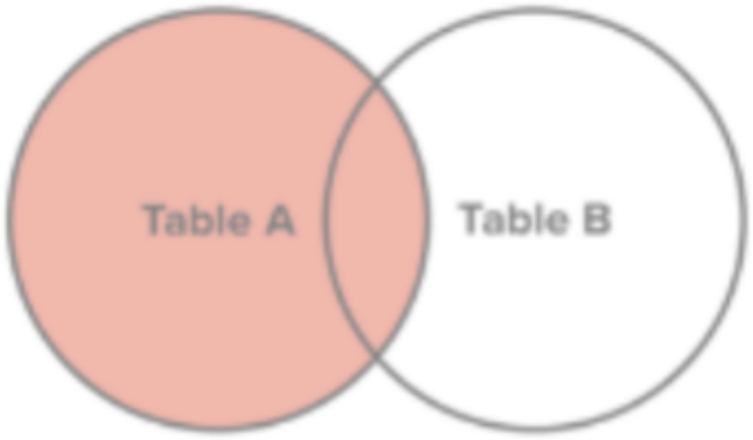
print("------")
print('INNER: \n',df1.merge(df2, how='inner'))
```

```
INPUTS:
    x   y
0  x1  y1
1  x2  y2
2  x3  y3

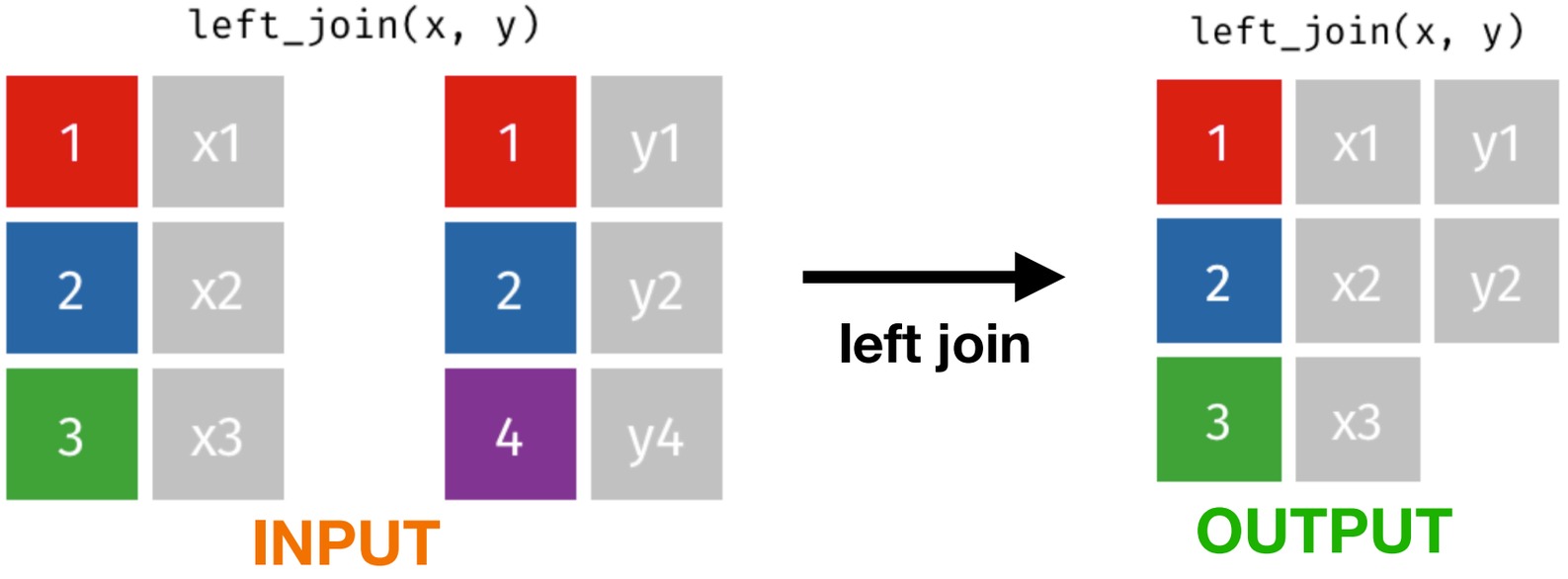
    x   y   w   z
0  x1  y1  w1  z1
1  x2  y2  w2  z2
2  x4  y4  w4  z4
```

```
INNER JOIN:
    x   y   w   z
0  x1  y1  w1  z1
1  x2  y2  w2  z2
```

Left Joins



Select all records from Table A,
along with records from Table B
for which the join condition is met
(if at all).



EXAMPLE:

```
print("----JOINING DATAFRAMES---")
df1 = pd.DataFrame([['x1', 'y1'], ['x2', 'y2'], ['x3', 'y3']], columns=['x', 'y'])
df2 = pd.DataFrame([['x1', 'y1','w1', 'z1']
                  , [‘x2', 'y2', 'w2', 'z2']
                  , [‘x4', 'y4', 'w4', 'z4']],columns=['x', 'y','w', 'z'])
print("INPUTS:")
print(df1,'\n')
print(df2)

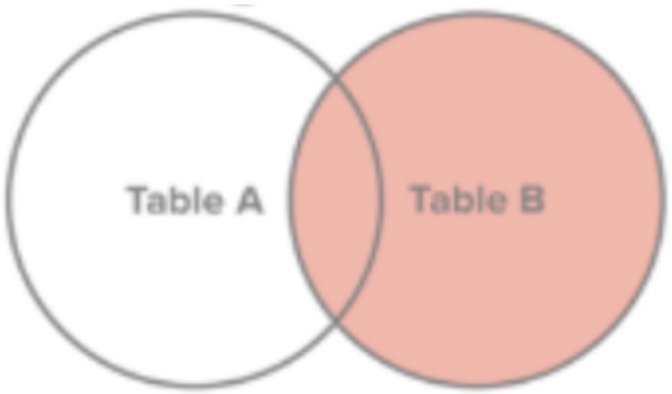
print("------")
print(‘LEFT JOIN:  \n',df1.merge(df2, how='left'))
```

```
INPUTS:
    x   y
0  x1  y1
1  x2  y2
2  x3  y3

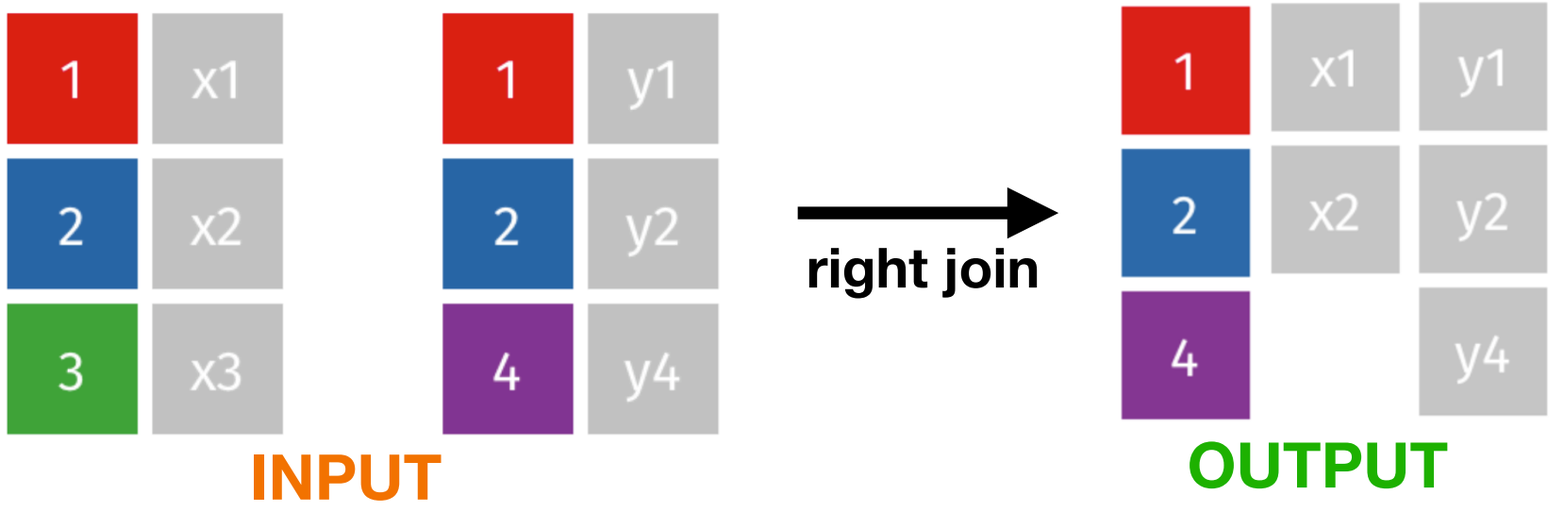
    x   y   w   z
0  x1  y1  w1  z1
1  x2  y2  w2  z2
2  x4  y4  w4  z4
```

```
LEFT JOIN:
    x   y    w    z
0  x1  y1   w1   z1
1  x2  y2   w2   z2
2  x3  y3  NaN  NaN
```

Right Joins



Select all records from Table B,
along with records from Table A
for which the join condition is met
(if at all).



EXAMPLE:

```
print("----JOINING DATAFRAMES---")
df1 = pd.DataFrame([['x1', 'y1'], ['x2', 'y2'], ['x3', 'y3']], columns=['x', 'y'])
df2 = pd.DataFrame([['x1', 'y1','w1', 'z1']
                  , [‘x2', 'y2', 'w2', 'z2']
                  , [‘x4', 'y4', 'w4', 'z4']],columns=['x', 'y','w', 'z'])
print("INPUTS:")
print(df1,'\n')
print(df2)

print("------")
print(‘RIGHT JOIN:  \n',df1.merge(df2, how=‘right'))
```

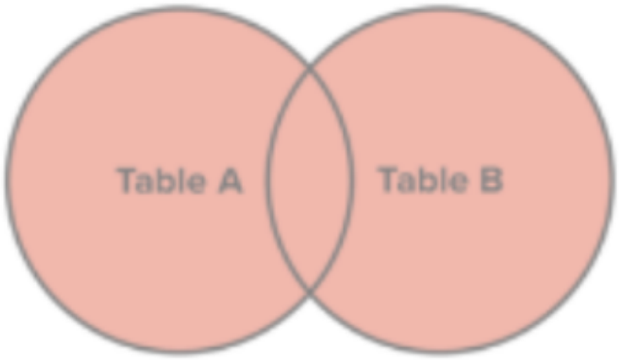
```
INPUTS:
    x   y
0  x1  y1
1  x2  y2
2  x3  y3

    x   y   w   z
0  x1  y1  w1  z1
1  x2  y2  w2  z2
2  x4  y4  w4  z4
```

```
RIGHT JOINS:
    x   y   w   z
0  x1  y1  w1  z1
1  x2  y2  w2  z2
2  x4  y4  w4  z4
```

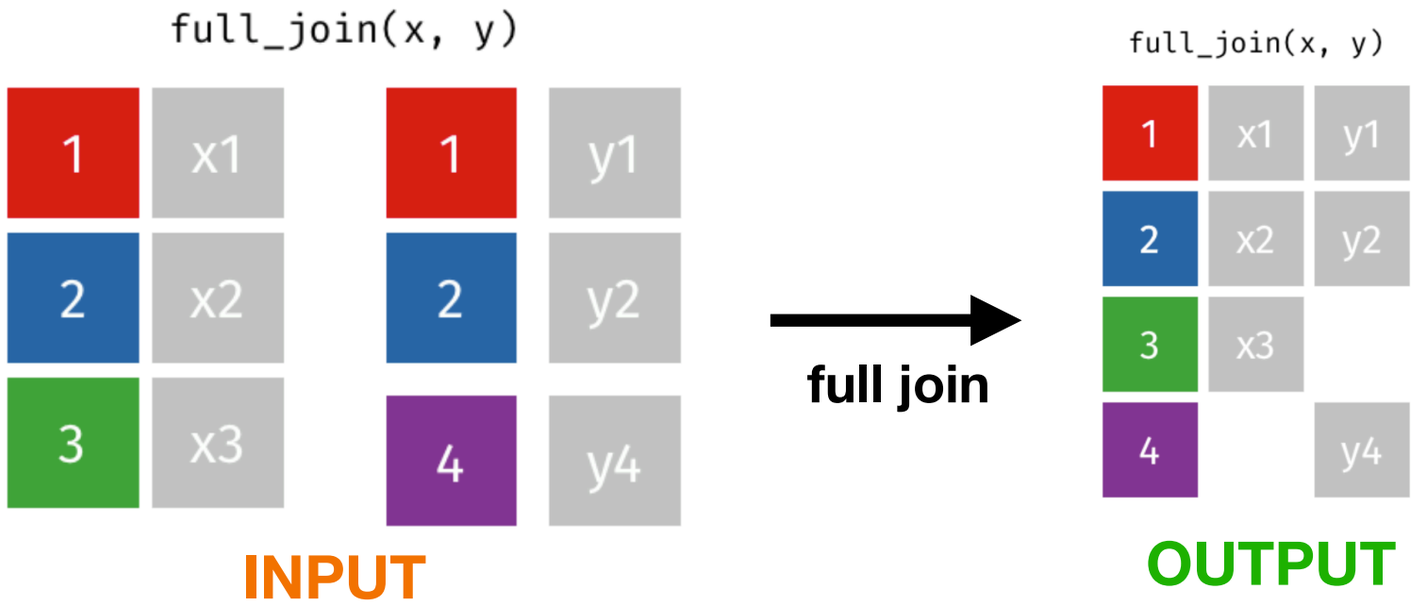
Full (outer) Joins

The union of two sets *A* and *B* is the set of elements which are in *A*, in *B*, or in both *A* and *B*.



Select all records from Table A and Table B, regardless of whether the join condition is met or not.

$$A \cup B = \{x : x \in A \textbf{ or } x \in B\}$$



EXAMPLE:

```
print("----JOINING DATAFRAMES---")
df1 = pd.DataFrame([['x1', 'y1'], ['x2', 'y2'], ['x3', 'y3']], columns=['x', 'y'])
df2 = pd.DataFrame([['x1', 'y1','w1', 'z1']
                    , [‘x2', 'y2', 'w2', 'z2']
                    , [‘x4', 'y4', 'w4', 'z4']],columns=['x', 'y','w', 'z'])
print("INPUTS:")
print(df1,'\n')
print(df2)

print("------")
print(‘OUTER JOIN:  \n',df1.merge(df2, how='outer'))
```

```
INPUTS:
    x   y
0  x1  y1
1  x2  y2
2  x3  y3

    x   y   w   z
0  x1  y1  w1  z1
1  x2  y2  w2  z2
2  x4  y4  w4  z4
```

```
OUTER JOIN:
     x    y    w    z
0   x1   y1   w1   z1
1   x2   y2   w2   z2
2   x3   y3  NaN  NaN
3   x4   y4   w4   z4
```

R Join examples

```r
library('tidyverse')

#LOAD DATA
df1 <- read_csv('data/toy-1.csv')
df2 <- read_csv('data/toy-2.csv')

print("----ORIGINAL----")
head(df1)
head(df2)

print("----INNER JOIN----")
df1 %>% inner_join(df2) %>% head()
df1 %>% inner_join(df2, by = c('x')) %>% head()
df1 %>% inner_join(df2, by = c('x' , 'w')) %>% head()

print("----LEFT JOIN----")
df1 %>% left_join(df2) %>% head()
# df1 %>% left_join(df2,by = c('x' , 'y')) %>% head()

print("----RIGHT JOIN----")
df1 %>% right_join(df2) %>% head()

print("----FULL JOIN----")
df1 %>% full_join(df2) %>% head()
```

OUTPUT

```
[1] "----ORIGINAL----"
# A tibble: 3 × 3
  x     y     w
  <chr> <chr> <chr>
1 x1    y1    w1
2 x2    y2    w2
3 x5    y5    w5
# A tibble: 4 × 4
  x     y     w     z
  <chr> <chr> <chr> <chr>
1 x1    y1    w1    z1
2 x2    y2    w2    z2
3 x3    y3    w3    z3
4 x4    y4    w4    z4
[1] "----INNER JOIN----"
Joining, by = c("x", "y", "w")
# A tibble: 2 × 4
  x     y     w     z
  <chr> <chr> <chr> <chr>
1 x1    y1    w1    z1
2 x2    y2    w2    z2
# A tibble: 2 × 6
  x     y.x   w.x   y.y   w.y   z
  <chr> <chr> <chr> <chr> <chr> <chr>
1 x1    y1    w1    y1    w1    z1
2 x2    y2    w2    y2    w2    z2
# A tibble: 2 × 5
  x     y.x   w     y.y   z
  <chr> <chr> <chr> <chr> <chr>
1 x1    y1    w1    y1    z1
2 x2    y2    w2    y2    z2
[1] "----LEFT JOIN----"
Joining, by = c("x", "y", "w")
# A tibble: 3 × 4
  x     y     w     z
  <chr> <chr> <chr> <chr>
1 x1    y1    w1    z1
2 x2    y2    w2    z2
3 x5    y5    w5    NA
[1] "----RIGHT JOIN----"
Joining, by = c("x", "y", "w")
# A tibble: 4 × 4
  x     y     w     z
  <chr> <chr> <chr> <chr>
1 x1    y1    w1    z1
2 x2    y2    w2    z2
3 x3    y3    w3    z3
4 x4    y4    w4    z4
[1] "----FULL JOIN----"
Joining, by = c("x", "y", "w")
# A tibble: 5 × 4
  x     y     w     z
  <chr> <chr> <chr> <chr>
1 x1    y1    w1    z1
2 x2    y2    w2    z2
3 x5    y5    w5    NA
4 x3    y3    w3    z3
5 x4    y4    w4    z4
```

SPLIT-APPLY-COMBINE

Overview

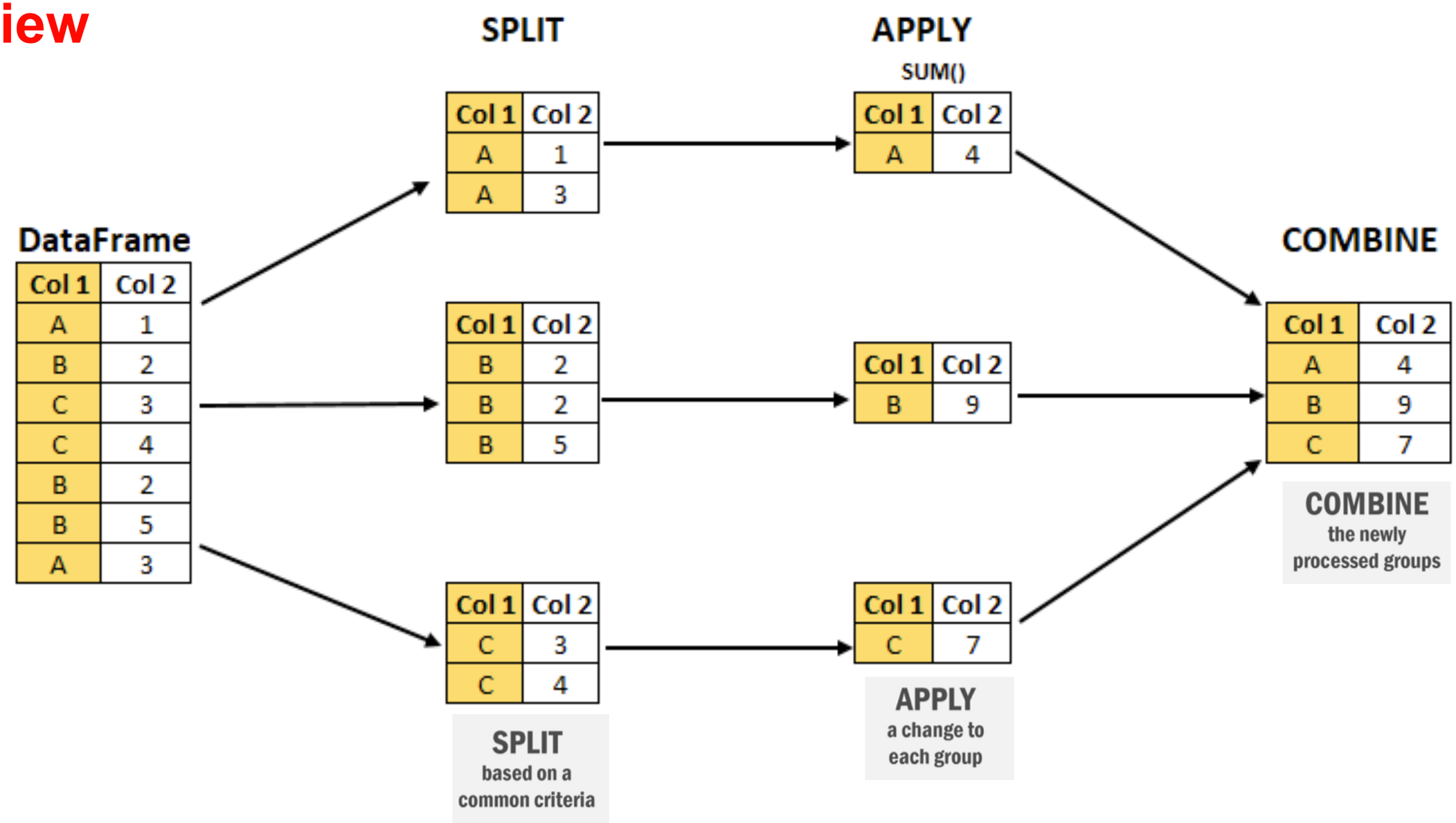
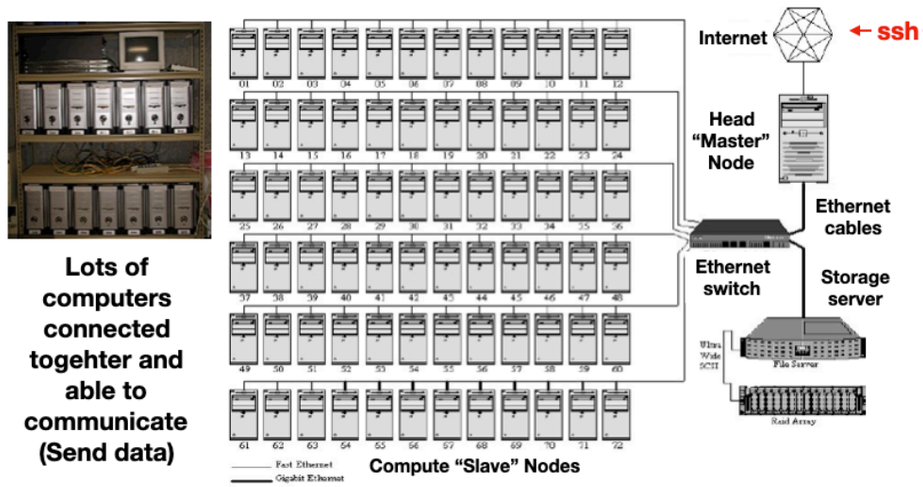


IMAGE SOURCE: https://medium.com/analytics-vidhya/split-apply-combine-strategy-for-data-mining-4fd6e2a0cc99

1. **Split:** Split the data into groups based on some criteria thereby creating a GroupBy object. (We can use the column or a combination of columns to split the data into groups)
2. **Apply:** Apply a function to each group independently. (Aggregate, Transform, or Filter the data in this step)
3. **Combine:** Combine the results into a data structure (Pandas Series, Pandas DataFrame)

Process can be distributed for "big" data



GAPMINDER

What is it?

https://www.gapminder.org/

Gapminder is an independent Swedish foundation with no political, religious or economic affiliations. Gapminder fights devastating misconceptions and promotes a fact-based worldview everyone can understand.

DATA: https://www.gapminder.org/data/

Search...

Babies per woman

CO2 emissions per person

Child mortality

Income

Life expectancy

Population

- Communication
- Economy
- Education
- Energy
- Environment
- Health
- Infrastructure
- Population
- Society
- Sustainability
- Work
- For advanced users
- Other datasets

CO2 emissions (tonnes per person)

description: Carbon dioxide emissions from the burning of fossil fuels (metric tonnes of CO2 per person).
source_url: https://cdiac.ess-dive.lbl.gov/

VIEW AS: DOWNLOAD AS: CSV XLSX

country	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
Tonga	1.26	1.11	1.17	1.27	1.13	0.991	1.04	1.12	1.13	1.2	1.27	1.29	1.31
Trinidad and Tobago	32.7	34.7	33.7	33.9	36	35.1	33.7	34.3	33.9	33.7	31.8	31.4	31.3
Tunisia	2.22	2.3	2.34	2.31	2.55	2.38	2.43	2.45	2.52	2.62	2.57	2.67	2.73
Turkey	4.1	4.5	4.4	4.43	4.35	4.63	4.74	4.55	4.69	4.86	5.03	5.25	5.2

Nuclear electricity production, total

description: The amount of electricity produced by nuclear power plants in a given year, counted in tonnes of oil equivalent (toe).
source_url: https://data.worldbank.org/indicator/EG.ELC.NUCL.KH

VIEW AS: DOWNLOAD AS: CSV XLSX

country	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
Angola	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Albania	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
United Arab Emirates	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Argentina	641k	685k	641k	611k	531k	607k	501k	651k	677k	591k	661k	621k	630k	702k	617k	
Armenia	200k	138k	137k	179k	172k	171k	196k	172k	189k	234k	227k	220k	212k	214k	214k	

R contains a small subset:

```
install.packages("gapminder")
library("gapminder")  #TO PULL DATASET
library("tidyverse")  #FOR WRITING
print(gapminder)
write_csv(gapminder,'gapminder.csv')
```

→

```
# A tibble: 1,704 × 6
   country     continent  year lifeExp      pop gdpPercap
   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
 1 Afghanistan Asia       1952    28.8  8425333      779.
 2 Afghanistan Asia       1957    30.3  9240934      821.
 3 Afghanistan Asia       1962    32.0 10267083      853.
 4 Afghanistan Asia       1967    34.0 11537966      836.
 5 Afghanistan Asia       1972    36.1 13079460      740.
 6 Afghanistan Asia       1977    38.4 14880372      786.
 7 Afghanistan Asia       1982    39.9 12881816      978.
 8 Afghanistan Asia       1987    40.8 13867957      852.
 9 Afghanistan Asia       1992    41.7 16317921      649.
10 Afghanistan Asia       1997    41.8 22227415      635.
# … with 1,694 more rows
```

Example-1

```
import numpy  as np
import pandas as pd

df = pd.read_csv('data/gapminder.csv')
print("----ORIGINAL----")
print(df.head())
```

```
----ORIGINAL----
       country continent  year  lifeExp       pop   gdpPercap
0  Afghanistan      Asia  1952   28.801   8425333  779.445314
1  Afghanistan      Asia  1957   30.332   9240934  820.853030
2  Afghanistan      Asia  1962   31.997  10267083  853.100710
3  Afghanistan      Asia  1967   34.020  11537966  836.197138
4  Afghanistan      Asia  1972   36.088  13079460  739.981106
```

```
print("----GROUP BY CONTINENT----")
print((df.groupby('continent')).head())
# print('----\n',df.groupby('continent').tail())
```

```
----GROUP BY CONTINENT----
       country continent  year  lifeExp       pop     gdpPercap
0   Afghanistan      Asia  1952   28.801   8425333    779.445314
1   Afghanistan      Asia  1957   30.332   9240934    820.853030
2   Afghanistan      Asia  1962   31.997  10267083    853.100710
3   Afghanistan      Asia  1967   34.020  11537966    836.197138
4   Afghanistan      Asia  1972   36.088  13079460    739.981106
12      Albania    Europe  1952   55.230   1282697   1601.056136
13      Albania    Europe  1957   59.280   1476505   1942.284244
```

```
print("----AVERAGE LIFE EXPECTANCY BY CONTINENT----")
print(df.groupby('continent')['lifeExp'].mean())
```

```
----AVERAGE LIFE EXPECTANCY BY CONTINENT----
continent
Africa      48.865330
Americas    64.658737
Asia        60.064903
Europe      71.903686
Oceania     74.326208
```

```
print("----AVERAGE LIFE EXPECTANCY BY COUNTRY----")
print(df.groupby('country')['lifeExp'].mean())
```

```
----AVERAGE LIFE EXPECTANCY BY COUNTRY----
country
Afghanistan           37.478833
Albania               68.432917
Algeria               59.030167
Angola                37.883500
Argentina             69.060417
                          ...
```

```
print("----AVERAGE EXPECTANCY BY CONTINENT AND YEAR----")
print(df.groupby(['continent','year'])['lifeExp'].mean())
```

```
----AVERAGE LIFE EXPECTANCY BY CONTINENT AND YEAR----
continent  year
Africa     1952    39.135500
           1957    41.266346
           1962    43.319442
           1967    45.334538
           1972    47.450942
           1977    49.580423
           1982    51.592865
           1987    53.344788
           1992    53.629577
           1997    53.598269
           2002    53.325231
           2007    54.806038
Americas   1952    53.279840
           1957    55.960280
           1962    58.398760
```

Example-2

```r
library('tidyverse')

print("----ORIGINAL----")
gap <- read_csv('data/gapminder.csv')
head(gap)

print("----GROUP BY CONTINENT----")
gap %>% group_by(continent)

print("----AVERAGE LIFE EXPECTANCY BY
CONTINENT----")
gap %>% group_by(continent) %>%
summarize(avgLifeExp = mean(lifeExp))


print("----AVERAGE LIFE EXPECTANCY BY
COUNTRY----")
gap %>% group_by(country) %>%
summarize(avgLifeExp = mean(lifeExp))

print("----AVERAGE LIFE EXPECTANCY BY
CONTINENT AND YEAR----")
gap %>% group_by(continent, year) %>%
summarize(mean(lifeExp))

print("----AVERAGE+MEDIAN EXPECTANCY BY
YEAR----")
gap %>% group_by(year) %>%
summarize(means = mean(lifeExp),
medians = median(lifeExp))
```

```
[1] "----ORIGINAL----"
# A tibble: 6 × 6
  country     continent  year lifeExp      pop gdpPercap
  <chr>       <chr>     <dbl>   <dbl>    <dbl>     <dbl>
1 Afghanistan Asia       1952    28.8  8425333      779.
2 Afghanistan Asia       1957    30.3  9240934      821.
3 Afghanistan Asia       1962    32.0 10267083      853.
4 Afghanistan Asia       1967    34.0 11537966      836.
5 Afghanistan Asia       1972    36.1 13079460      740.
6 Afghanistan Asia       1977    38.4 14880372      786.
[1] "----GROUP BY CONTINENT----"
# A tibble: 1,704 × 6
# Groups:   continent [5]
   country     continent  year lifeExp      pop gdpPercap
   <chr>       <chr>     <dbl>   <dbl>    <dbl>     <dbl>
 1 Afghanistan Asia       1952    28.8  8425333      779.
 2 Afghanistan Asia       1957    30.3  9240934      821.
 3 Afghanistan Asia       1962    32.0 10267083      853.
 4 Afghanistan Asia       1967    34.0 11537966      836.
 5 Afghanistan Asia       1972    36.1 13079460      740.
# … with 1,694 more rows
[1] "----AVERAGE LIFE EXPECTANCY BY CONTINENT----"
# A tibble: 5 × 2
  continent avgLifeExp
  <chr>          <dbl>
1 Africa          48.9
2 Americas        64.7
3 Asia            60.1
4 Europe          71.9
5 Oceania         74.3
[1] "----AVERAGE LIFE EXPECTANCY BY COUNTRY----"
# A tibble: 142 × 2
   country     avgLifeExp
   <chr>            <dbl>
 1 Afghanistan       37.5
 2 Albania           68.4
 3 Algeria           59.0
 4 Angola            37.9
 5 Argentina         69.1
# … with 132 more rows
[1] "----AVERAGE LIFE EXPECTANCY BY CONTINENT AND YEAR----"
# A tibble: 60 × 3
# Groups:   continent [5]
   continent  year `mean(lifeExp)`
   <chr>     <dbl>           <dbl>
 1 Africa     1952            39.1
 2 Africa     1957            41.3
 3 Africa     1962            43.3
 4 Africa     1967            45.3
 5 Africa     1972            47.5
# … with 50 more rows
[1] "----AVERAGE+MEDIAN EXPECTANCY BY YEAR----"
# A tibble: 12 × 3
    year means medians
   <dbl> <dbl>   <dbl>
 1  1952  49.1    45.1
 2  1957  51.5    48.4
 3  1962  53.6    50.9
 4  1967  55.7    53.8
```