Module 02

DataFrames

Data Science Developer

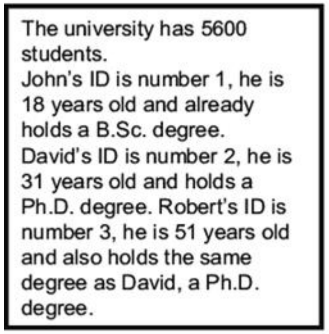
Purwadhika
Startup and Coding School

Outline

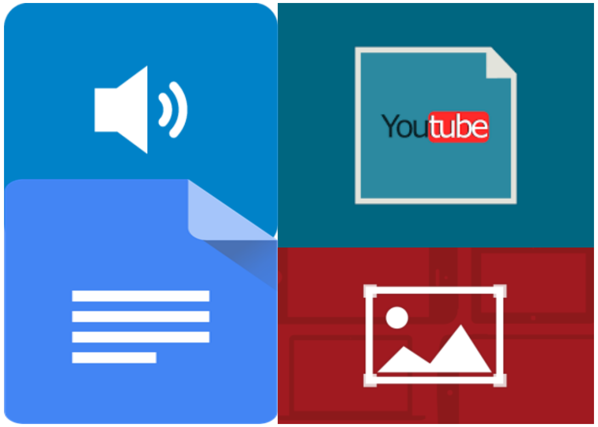
- What is Data ?
- What is a Data Frame ?
- DataFrame manipulation
  - Create
  - Selection
  - Addition
  - Deletion
  - Rename
  - Sorting

Data

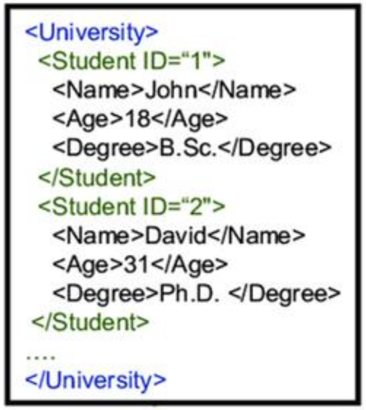
Unstructured Data



The university has 5600 students.
John's ID is number 1, he is 18 years old and already holds a B.Sc. degree.
David's ID is number 2, he is 31 years old and holds a Ph.D. degree. Robert's ID is number 3, he is 51 years old and also holds the same degree as David, a Ph.D. degree.

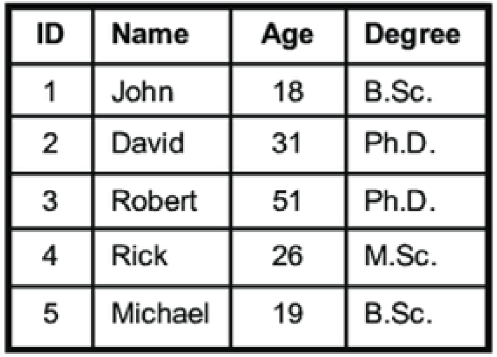


Semi-structured Data



```
<University>
 <Student ID="1">
   <Name>John</Name>
   <Age>18</Age>
   <Degree>B.Sc.</Degree>
 </Student>
 <Student ID="2">
   <Name>David</Name>
   <Age>31</Age>
   <Degree>Ph.D. </Degree>
</Student>
....
</University>
```

Structured Data



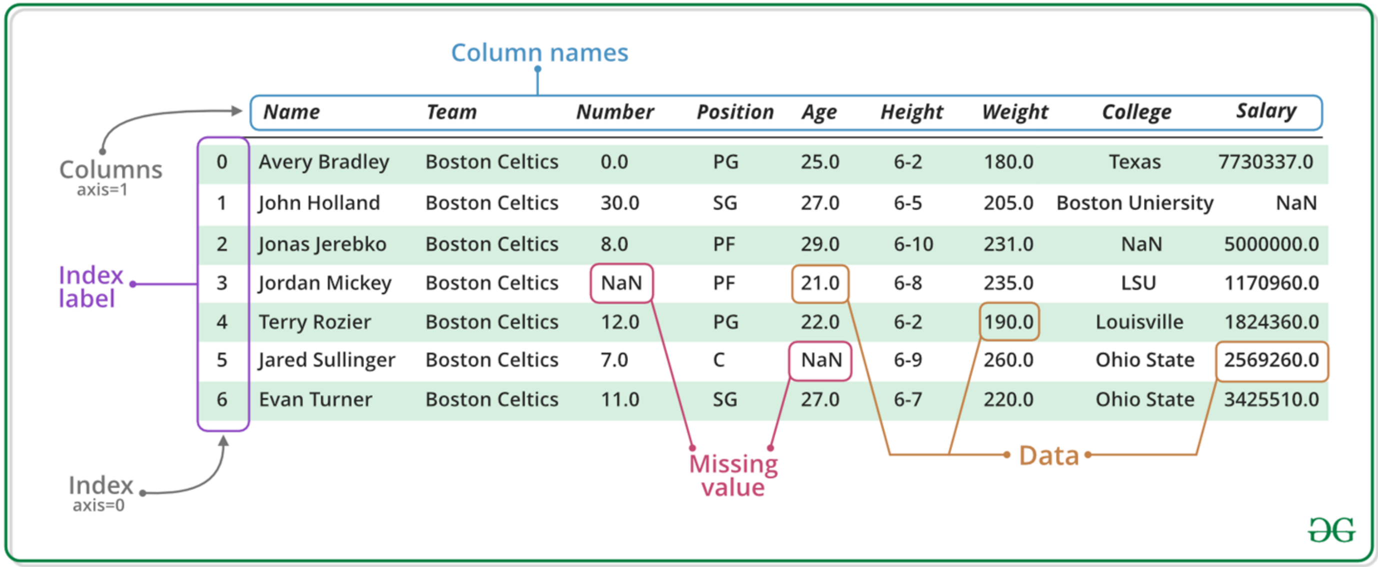
ID	Name	Age	Degree
1	John	18	B.Sc.
2	David	31	Ph.D.
3	Robert	51	Ph.D.
4	Rick	26	M.Sc.
5	Michael	19	B.Sc.

Purwadhika
Startup and Coding School

Structured Data : Tabular

- Data is individual units of information
- Data organized in a matrix (similar like numpy)
- Row represent observation
- Column represent a variable

Tabular Data Example



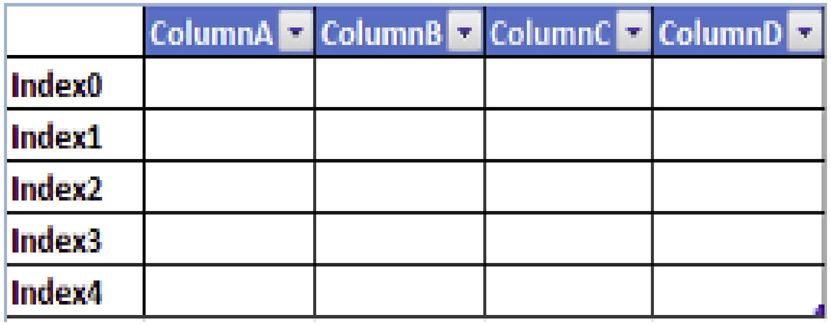
	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston Uniersity	NaN
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0	6-8	235.0	LSU	1170960.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN	6-9	260.0	Ohio State	2569260.0
6	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0

Observation (players)

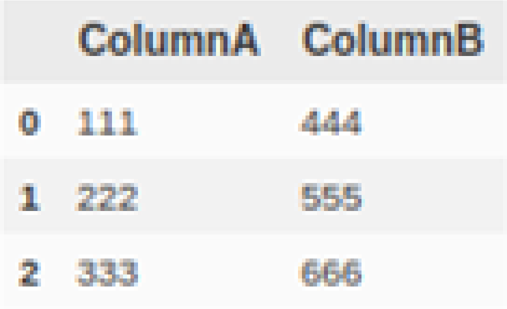
Variables (Player's Attributes)

DataFrame

- DataFrames are the workhorse of pandas and are directly inspired by the R programming language.
- It is a Table with rows and columns.
- We can think of a DataFrame as a bunch of Series objects put together to share the same index.



	ColumnA	ColumnB	ColumnC	ColumnD
Index0				
Index1				
Index2				
Index3				
Index4				



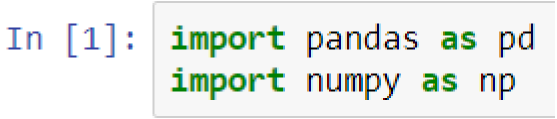
	ColumnA	ColumnB
0	111	444
1	222	555
2	333	666

Purwadhika
Startup and Coding School

Manipulating DataFrames

- Creation
- Selection
- Addition
- Deletion
- Indexing
- Sorting

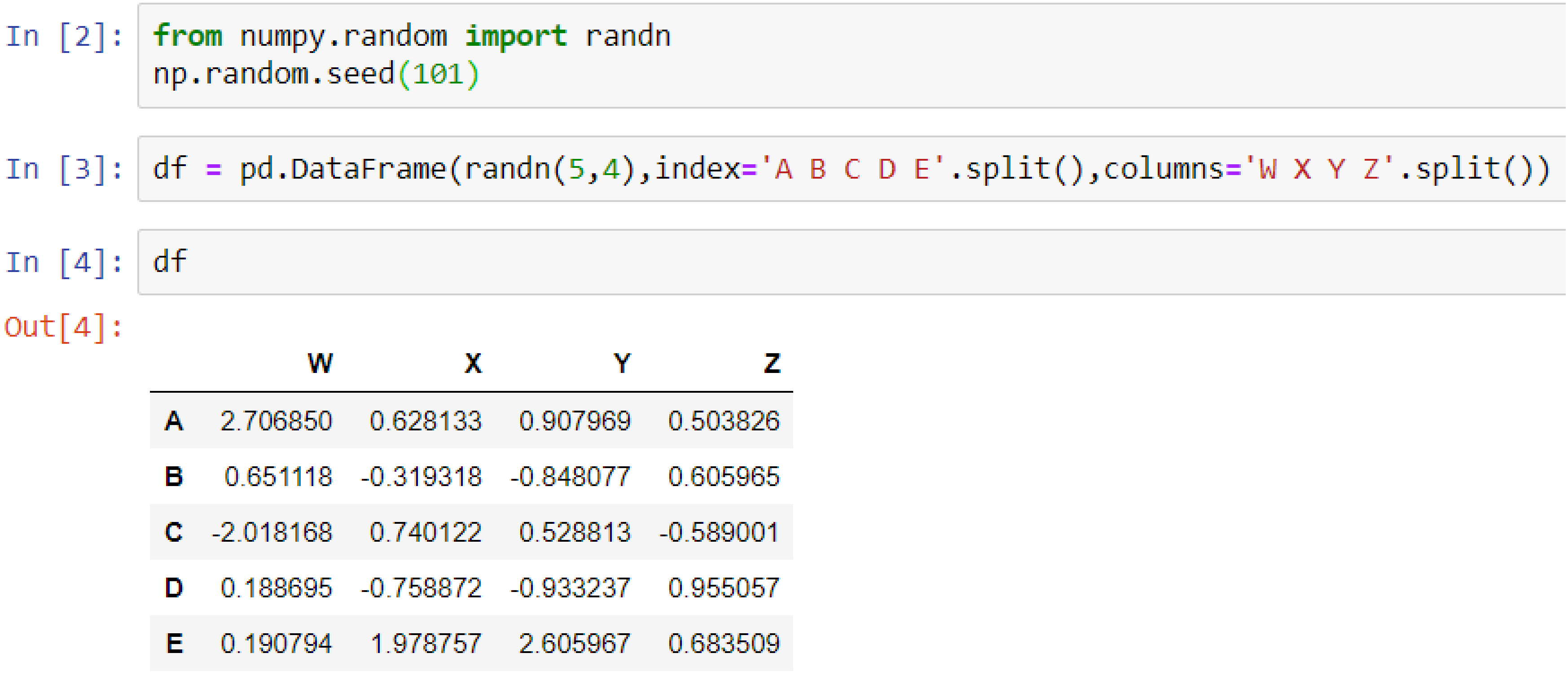
Using Numpy and Pandas



```
import pandas as pd
import numpy as np
```

Creating a DataFrame

Creating a DataFrame



```
In [2]: from numpy.random import randn
        np.random.seed(101)
```

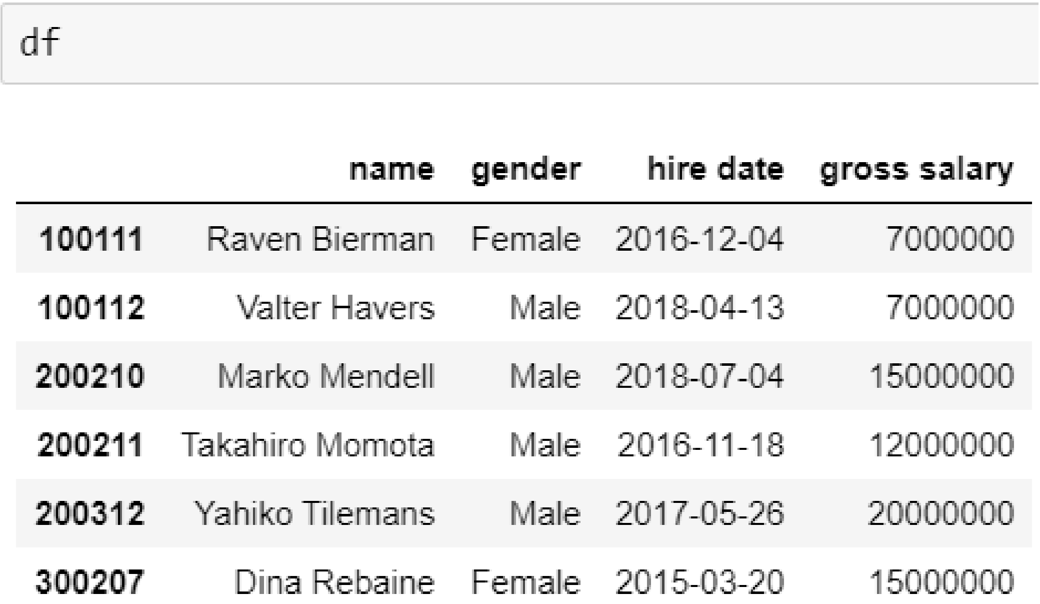
```
In [3]: df = pd.DataFrame(randn(5,4),index='A B C D E'.split(),columns='W X Y Z'.split())
```

```
In [4]: df
```

Out[4]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

Create this Dataframe!!!



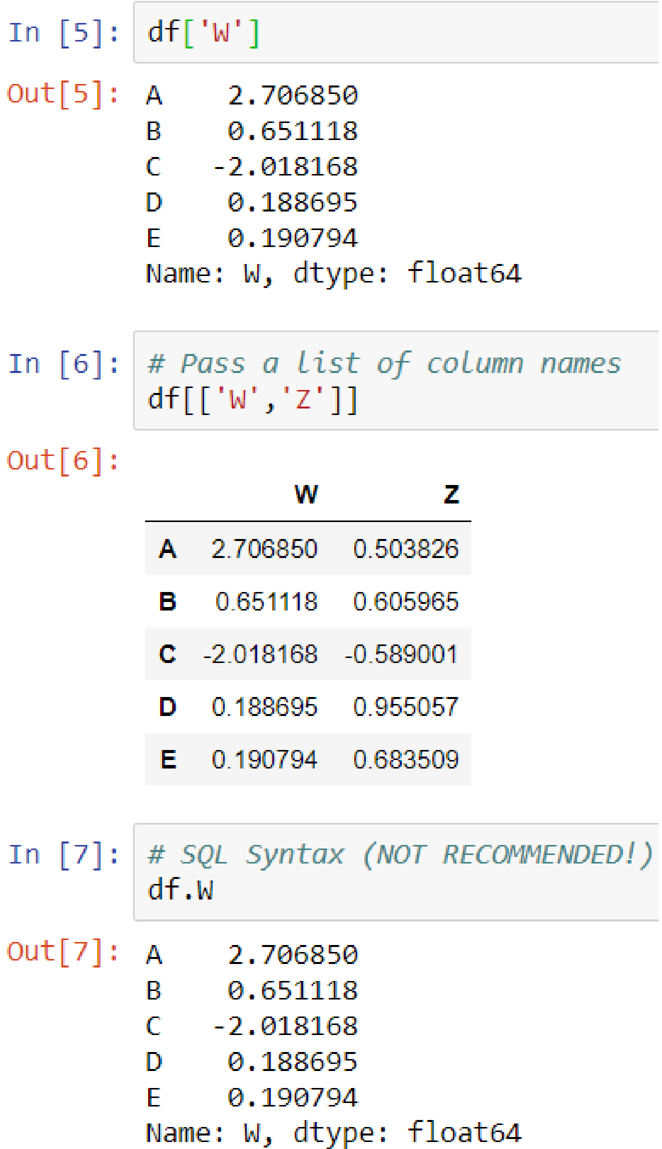
```
df
```

	name	gender	hire date	gross salary
100111	Raven Bierman	Female	2016-12-04	7000000
100112	Valter Havers	Male	2018-04-13	7000000
200210	Marko Mendell	Male	2018-07-04	15000000
200211	Takahiro Momota	Male	2016-11-18	12000000
200312	Yahiko Tilemans	Male	2017-05-26	20000000
300207	Dina Rebaine	Female	2015-03-20	15000000

Purwadhika
Startup and Coding School

Selection

Selection and Indexing



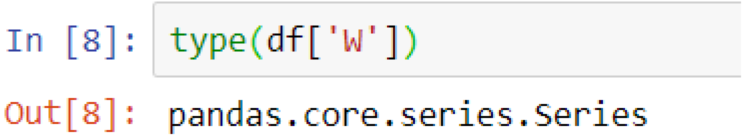
```
In [5]: df['W']

Out[5]: A    2.706850
        B    0.651118
        C   -2.018168
        D    0.188695
        E    0.190794
        Name: W, dtype: float64
```

```
In [6]: # Pass a list of column names
        df[['W','Z']]

Out[6]:
```

	W	Z
A	2.706850	0.503826
B	0.651118	0.605965
C	-2.018168	-0.589001
D	0.188695	0.955057
E	0.190794	0.683509

```
In [7]: # SQL Syntax (NOT RECOMMENDED!)
        df.W

Out[7]: A    2.706850
        B    0.651118
        C   -2.018168
        D    0.188695
        E    0.190794
        Name: W, dtype: float64
```

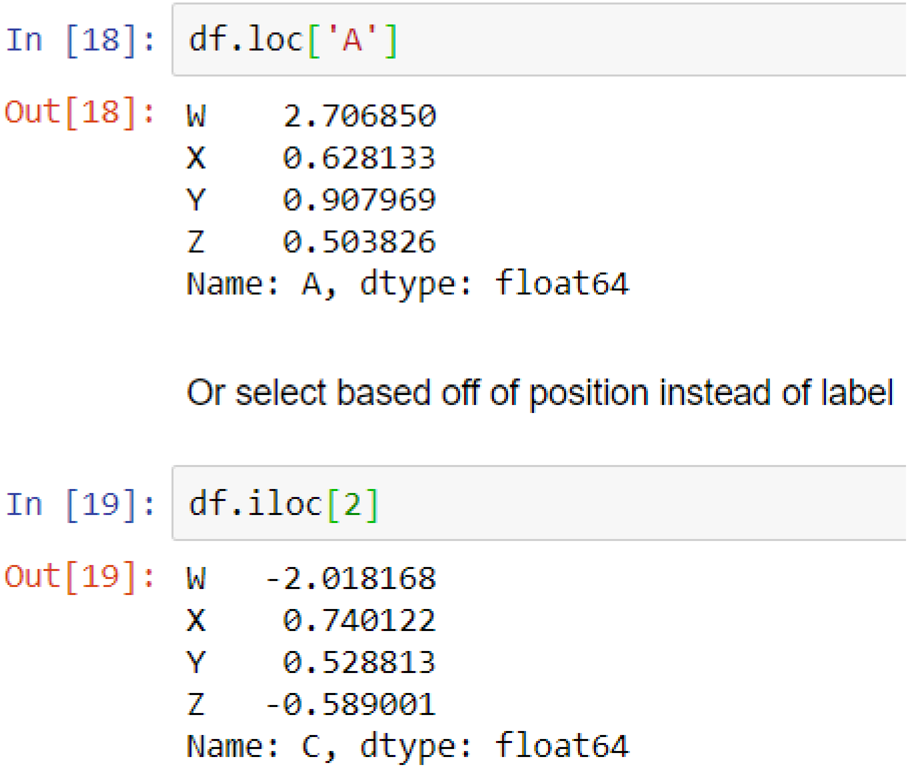
DataFrame Columns are just Series



```
In [8]: type(df['W'])

Out[8]: pandas.core.series.Series
```

Purwadhika
Startup and Coding School

Selecting Rows



```
In [18]: df.loc['A']
```

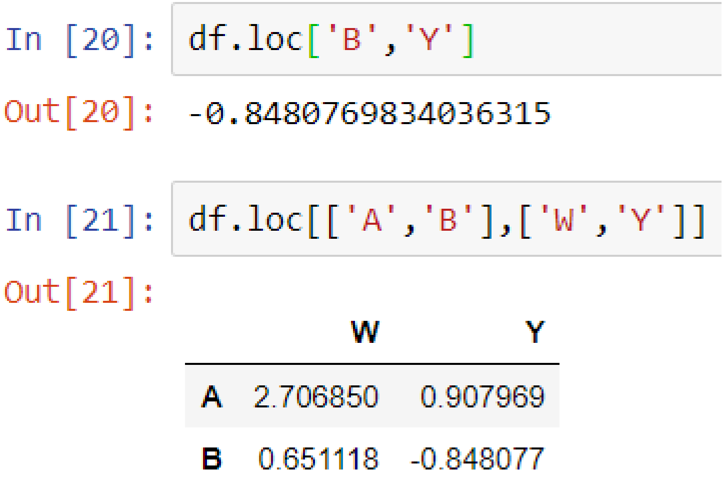
```
Out[18]: W    2.706850
         X    0.628133
         Y    0.907969
         Z    0.503826
         Name: A, dtype: float64
```

Or select based off of position instead of label

```
In [19]: df.iloc[2]
```

```
Out[19]: W   -2.018168
         X    0.740122
         Y    0.528813
         Z   -0.589001
         Name: C, dtype: float64
```

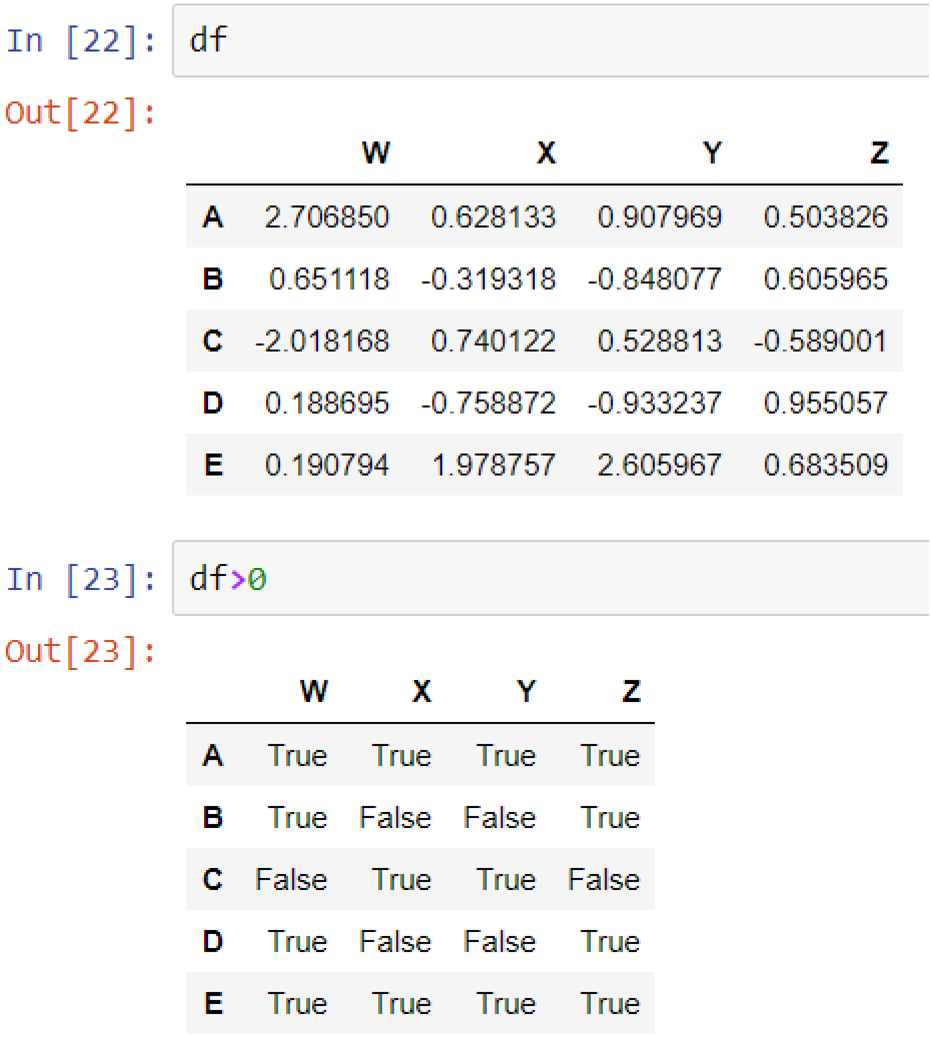
Selecting Subset of Rows and Columns



```
In [20]: df.loc['B','Y']

Out[20]: -0.8480769834036315

In [21]: df.loc[['A','B'],['W','Y']]

Out[21]:
```

	W	Y
A	2.706850	0.907969
B	0.651118	-0.848077

Conditional Selection



In [22]:
```
df
```

Out[22]:

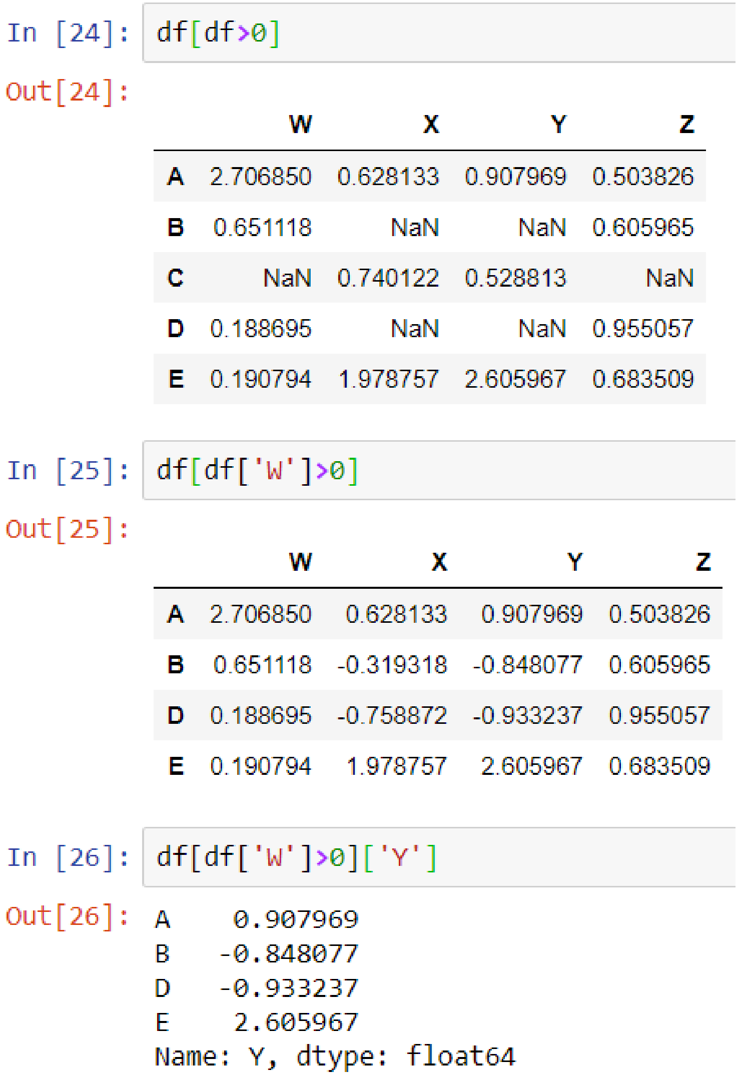
	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [23]:
```
df>0
```

Out[23]:

	W	X	Y	Z
A	True	True	True	True
B	True	False	False	True
C	False	True	True	False
D	True	False	False	True
E	True	True	True	True

Conditional Selection



```
In [24]: df[df>0]
```

Out[24]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	NaN	NaN	0.605965
C	NaN	0.740122	0.528813	NaN
D	0.188695	NaN	NaN	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
In [25]: df[df['W']>0]
```

Out[25]:

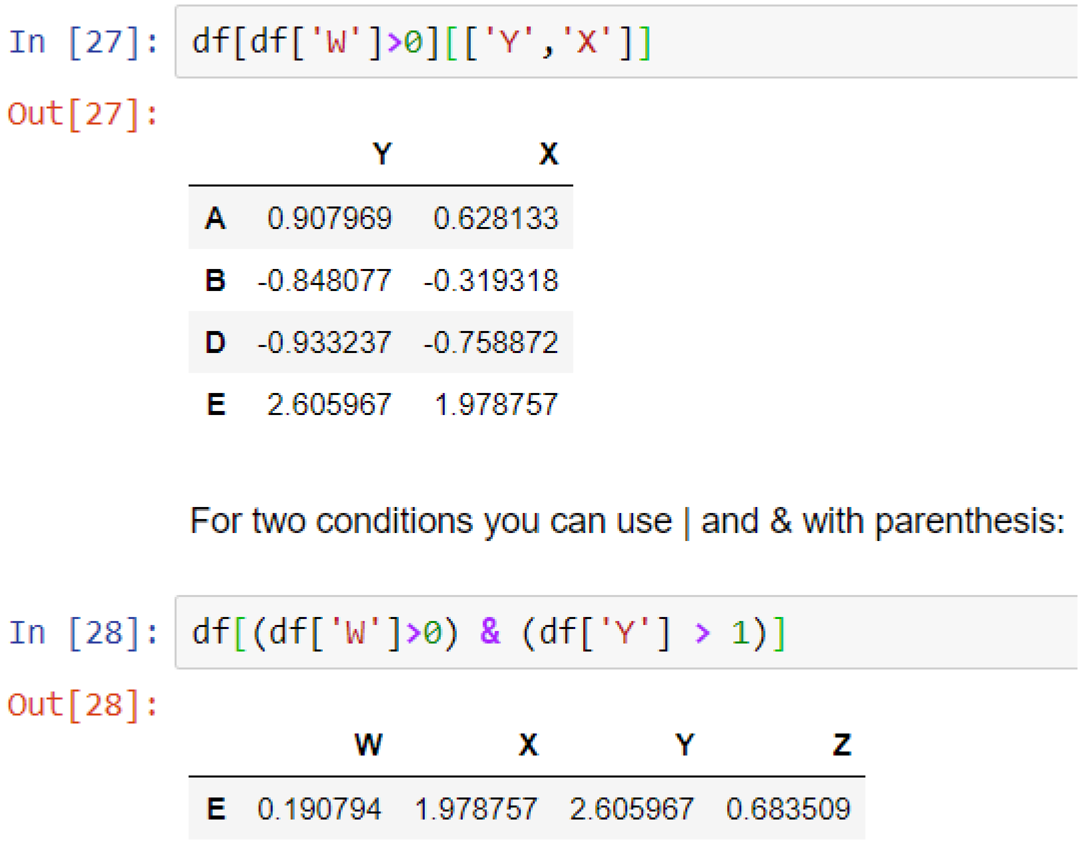
	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
In [26]: df[df['W']>0]['Y']
```

```
Out[26]: A    0.907969
         B   -0.848077
         D   -0.933237
         E    2.605967
         Name: Y, dtype: float64
```

Purwadhika
Startup and Coding School

Conditional Selection



```
In [27]: df[df['W']>0][['Y','X']]
```

Out[27]:

	Y	X
A	0.907969	0.628133
B	-0.848077	-0.319318
D	-0.933237	-0.758872
E	2.605967	1.978757

For two conditions you can use | and & with parenthesis:

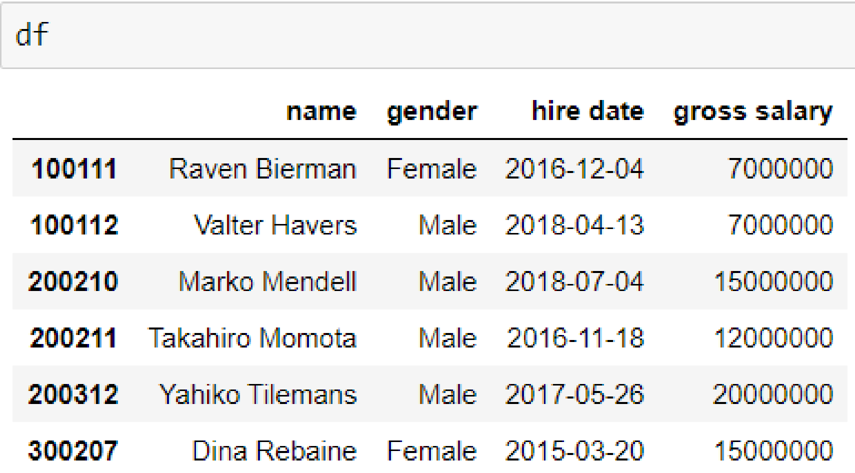
```
In [28]: df[(df['W']>0) & (df['Y'] > 1)]
```

Out[28]:

	W	X	Y	Z
E	0.190794	1.978757	2.605967	0.683509

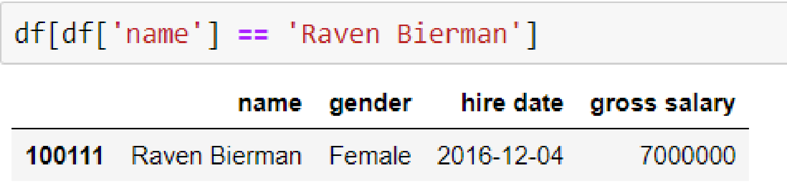
Purwadhika
Startup and Coding School

Conditional Selection



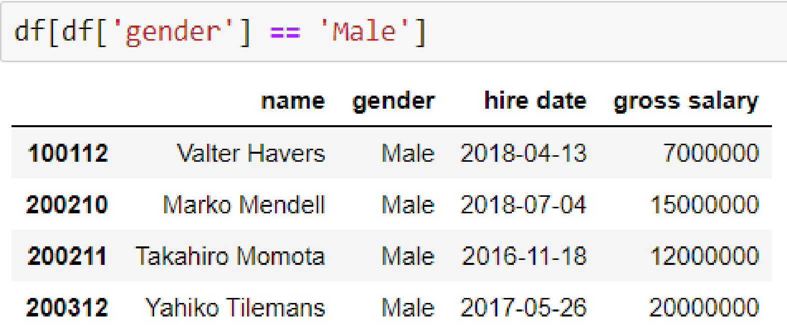
```
df
```

	name	gender	hire date	gross salary
100111	Raven Bierman	Female	2016-12-04	7000000
100112	Valter Havers	Male	2018-04-13	7000000
200210	Marko Mendell	Male	2018-07-04	15000000
200211	Takahiro Momota	Male	2016-11-18	12000000
200312	Yahiko Tilemans	Male	2017-05-26	20000000
300207	Dina Rebaine	Female	2015-03-20	15000000



```
df[df['name'] == 'Raven Bierman']
```

	name	gender	hire date	gross salary
100111	Raven Bierman	Female	2016-12-04	7000000

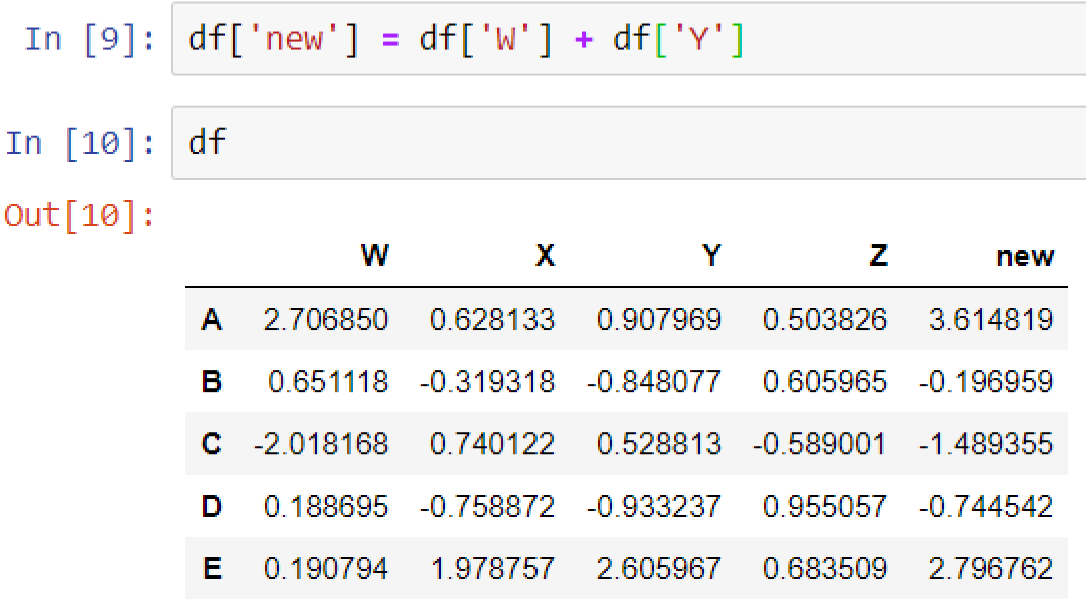


```
df[df['gender'] == 'Male']
```

	name	gender	hire date	gross salary
100112	Valter Havers	Male	2018-04-13	7000000
200210	Marko Mendell	Male	2018-07-04	15000000
200211	Takahiro Momota	Male	2016-11-18	12000000
200312	Yahiko Tilemans	Male	2017-05-26	20000000

Addition

Add a New Column



In [9]:
```
df['new'] = df['W'] + df['Y']
```

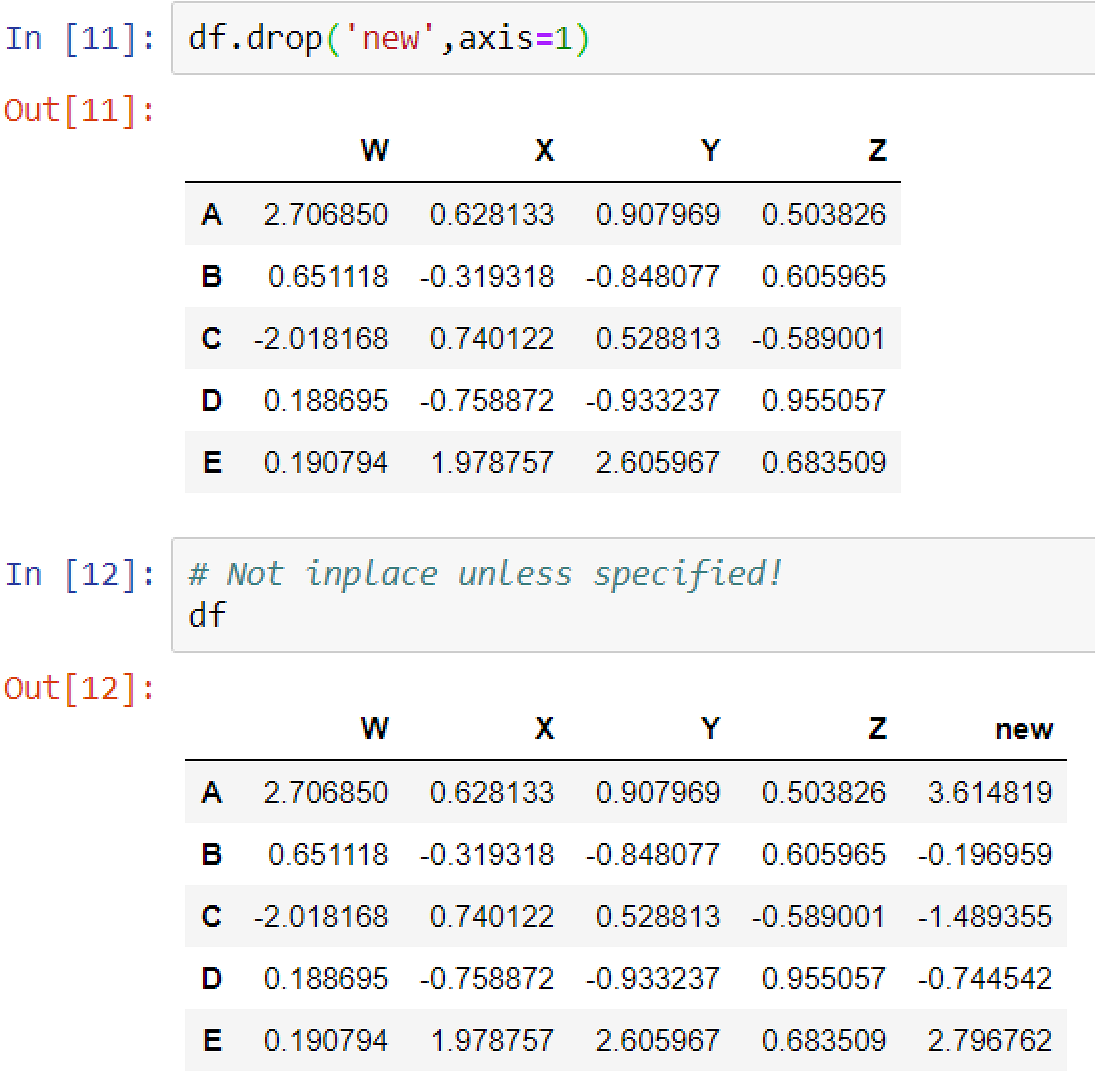
In [10]:
```
df
```

Out[10]:

	W	X	Y	Z	new
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

Deletion

Removing Columns
without inplace



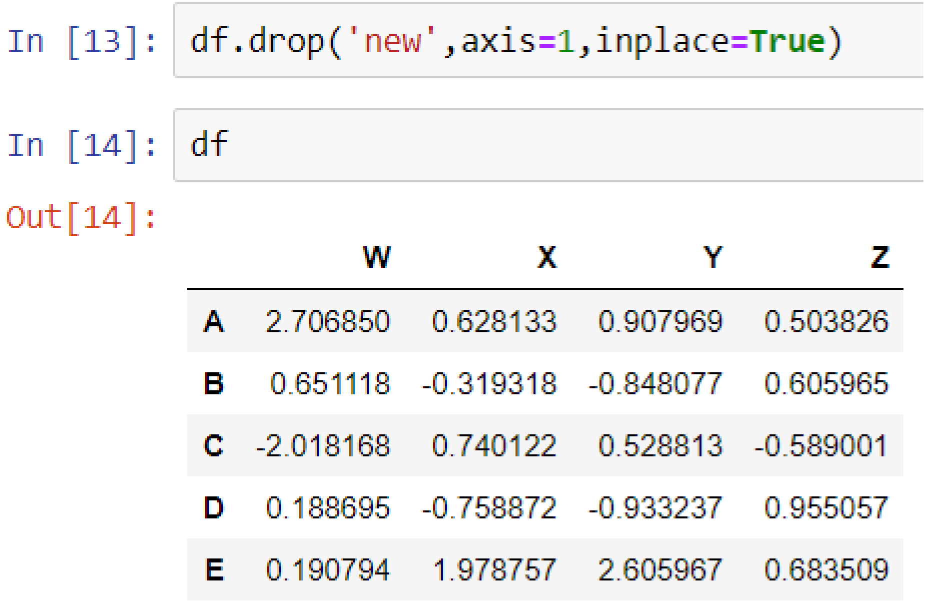
```
In [11]: df.drop('new',axis=1)
Out[11]:
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
In [12]: # Not inplace unless specified!
         df
Out[12]:
```

	W	X	Y	Z	new
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

Removing Columns
with inplace



```
In [13]: df.drop('new',axis=1,inplace=True)
```

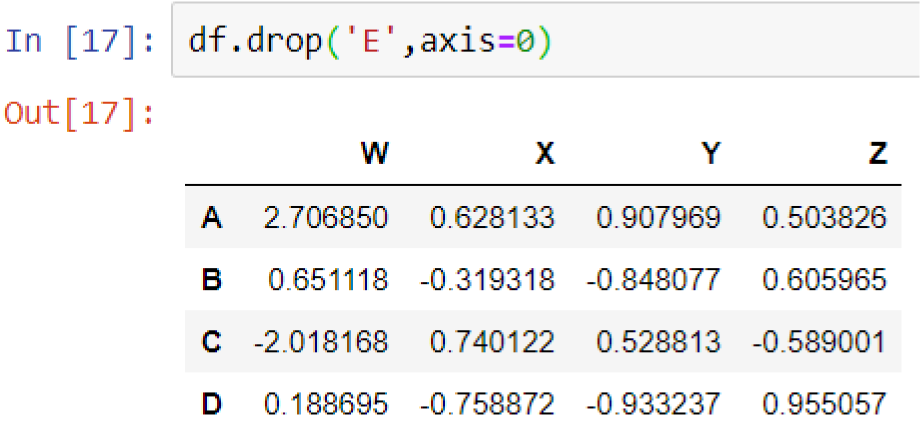
```
In [14]: df
```

Out[14]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

Removing Rows

Same with drop columns, the difference is the axis:



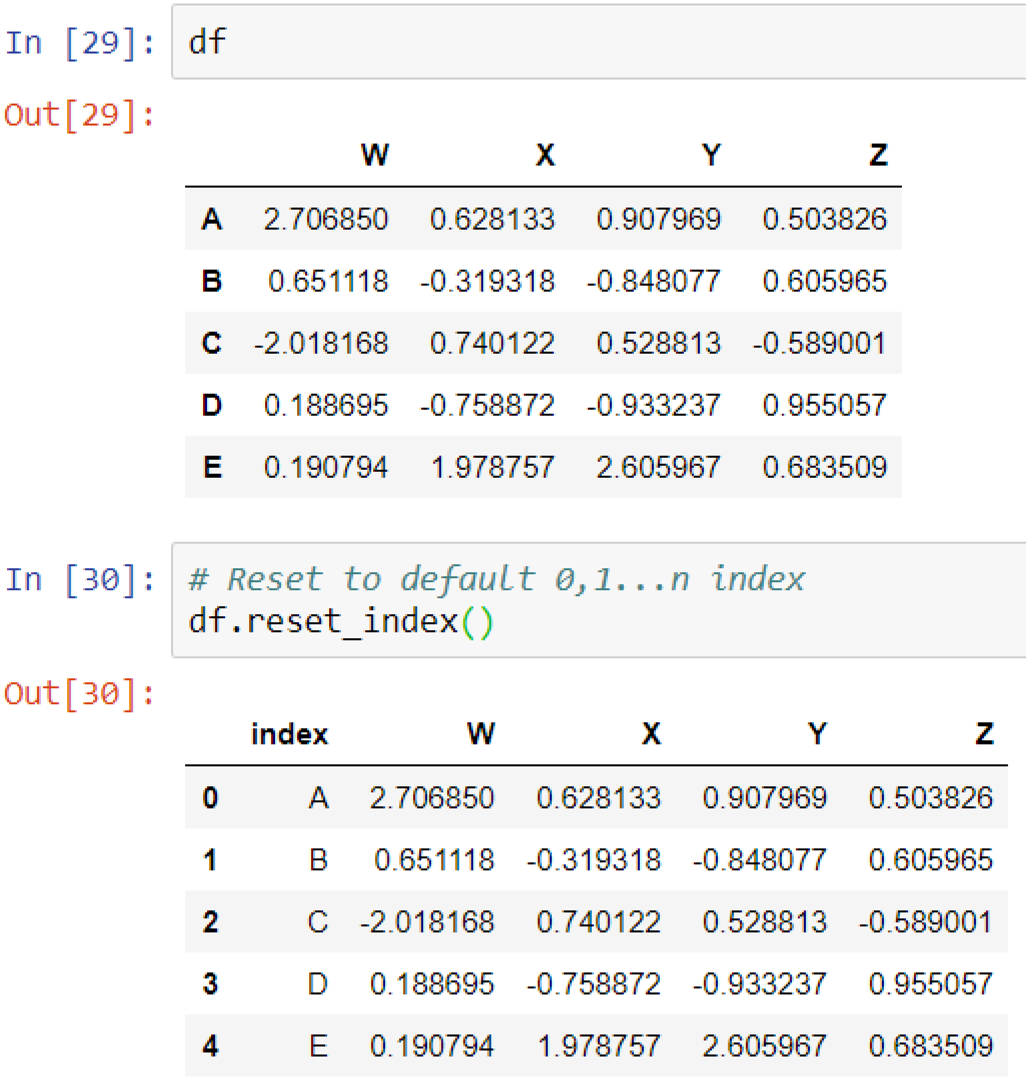
```
In [17]: df.drop('E',axis=0)
```

```
Out[17]:
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057

Indexing

More Index Details



```
In [29]: df
```

Out[29]:

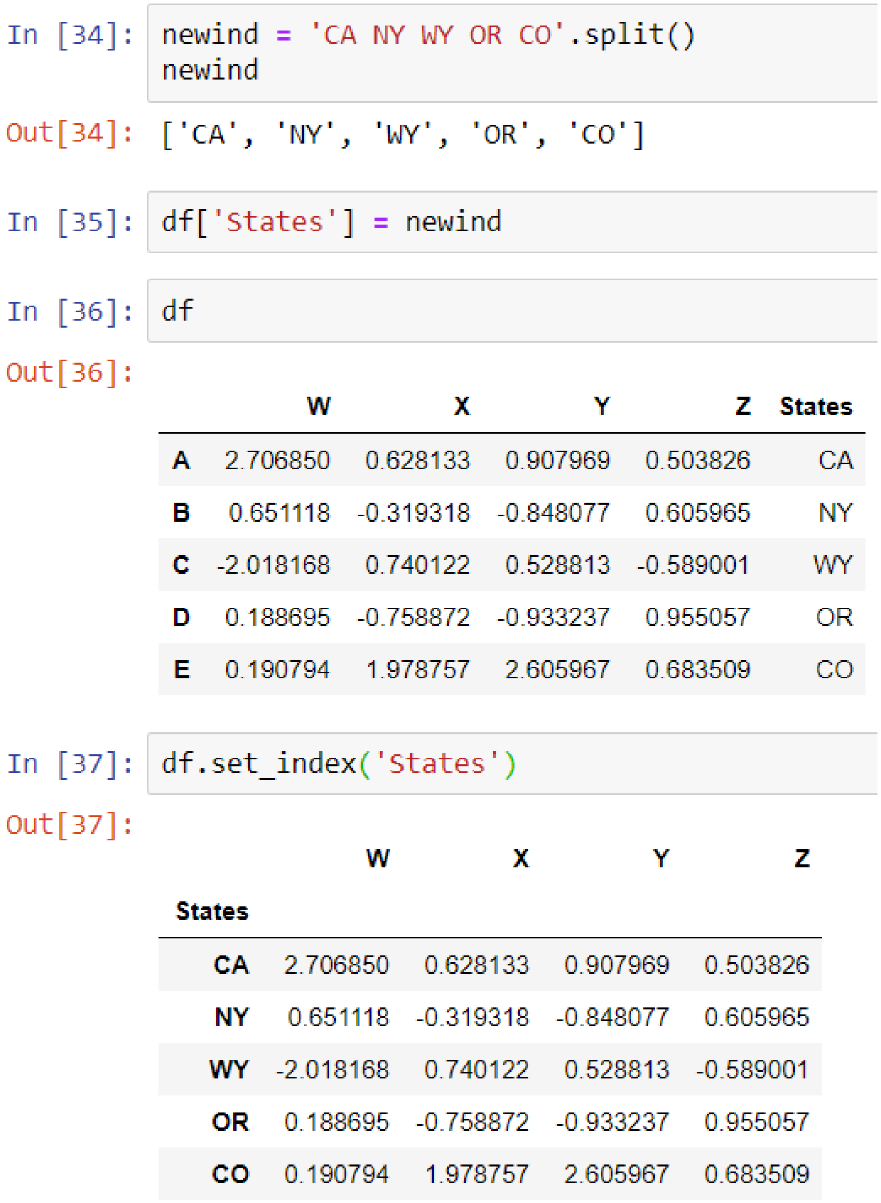
	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
In [30]: # Reset to default 0,1...n index
         df.reset_index()
```

Out[30]:

	index	W	X	Y	Z
0	A	2.706850	0.628133	0.907969	0.503826
1	B	0.651118	-0.319318	-0.848077	0.605965
2	C	-2.018168	0.740122	0.528813	-0.589001
3	D	0.188695	-0.758872	-0.933237	0.955057
4	E	0.190794	1.978757	2.605967	0.683509

More Index Details



```
In [34]: newind = 'CA NY WY OR CO'.split()
         newind
Out[34]: ['CA', 'NY', 'WY', 'OR', 'CO']

In [35]: df['States'] = newind

In [36]: df
Out[36]:
```

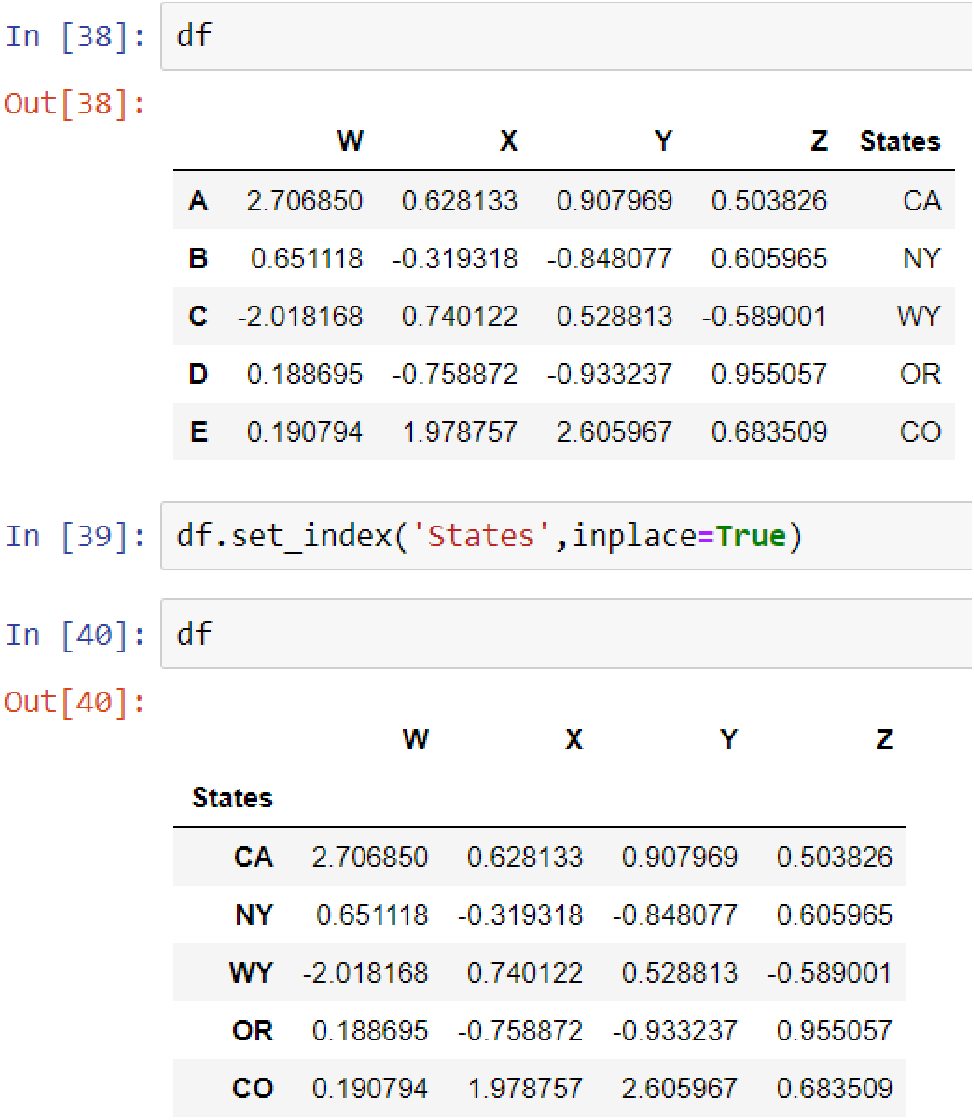
	W	X	Y	Z	States
A	2.706850	0.628133	0.907969	0.503826	CA
B	0.651118	-0.319318	-0.848077	0.605965	NY
C	-2.018168	0.740122	0.528813	-0.589001	WY
D	0.188695	-0.758872	-0.933237	0.955057	OR
E	0.190794	1.978757	2.605967	0.683509	CO

```
In [37]: df.set_index('States')
Out[37]:
```

	W	X	Y	Z
States				
CA	2.706850	0.628133	0.907969	0.503826
NY	0.651118	-0.319318	-0.848077	0.605965
WY	-2.018168	0.740122	0.528813	-0.589001
OR	0.188695	-0.758872	-0.933237	0.955057
CO	0.190794	1.978757	2.605967	0.683509

Purwadhika
Startup and Coding School

More Index Details



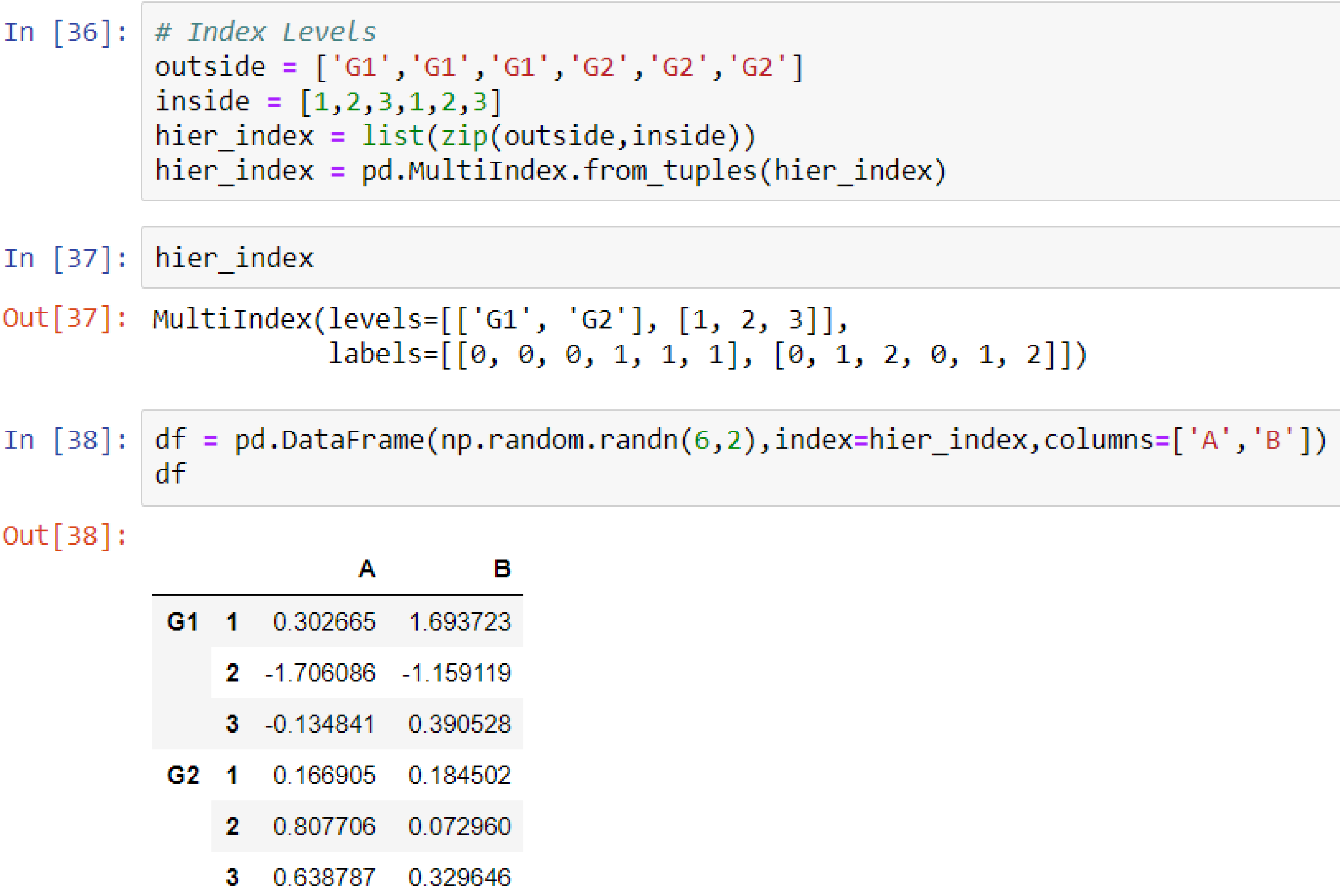
```
In [38]: df
Out[38]:
          W          X          Y          Z   States
A   2.706850   0.628133   0.907969   0.503826     CA
B   0.651118  -0.319318  -0.848077   0.605965     NY
C  -2.018168   0.740122   0.528813  -0.589001     WY
D   0.188695  -0.758872  -0.933237   0.955057     OR
E   0.190794   1.978757   2.605967   0.683509     CO
```

```
In [39]: df.set_index('States',inplace=True)
```

```
In [40]: df
Out[40]:
               W          X          Y          Z
States
    CA   2.706850   0.628133   0.907969   0.503826
    NY   0.651118  -0.319318  -0.848077   0.605965
    WY  -2.018168   0.740122   0.528813  -0.589001
    OR   0.188695  -0.758872  -0.933237   0.955057
    CO   0.190794   1.978757   2.605967   0.683509
```

Purwadhika
Startup and Coding School

Multi-Index and Index Hierarchy



```
In [36]: # Index Levels
         outside = ['G1','G1','G1','G2','G2','G2']
         inside = [1,2,3,1,2,3]
         hier_index = list(zip(outside,inside))
         hier_index = pd.MultiIndex.from_tuples(hier_index)
```

```
In [37]: hier_index
```

```
Out[37]: MultiIndex(levels=[['G1', 'G2'], [1, 2, 3]],
                    labels=[[0, 0, 0, 1, 1, 1], [0, 1, 2, 0, 1, 2]])
```

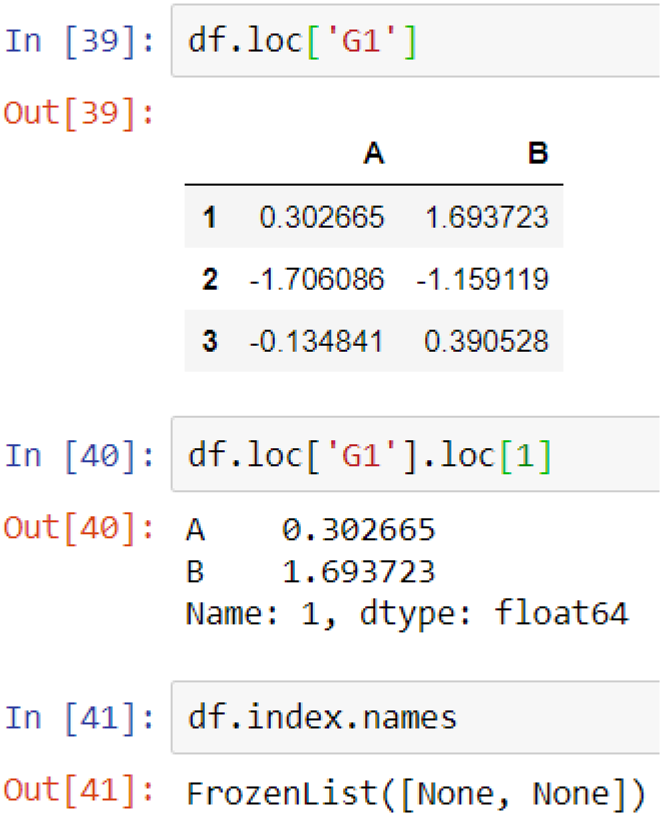
```
In [38]: df = pd.DataFrame(np.random.randn(6,2),index=hier_index,columns=['A','B'])
         df
```

Out[38]:

		A	B
G1	1	0.302665	1.693723
	2	-1.706086	-1.159119
	3	-0.134841	0.390528
G2	1	0.166905	0.184502
	2	0.807706	0.072960
	3	0.638787	0.329646

Multi-Index and Index Hierarchy

- Now let's show how to index this! For index hierarchy we use df.loc[], if this was on the columns axis, you would just use normal bracket notation df[]. Calling one level of the index returns the sub-dataframe:



```
In [39]: df.loc['G1']
```

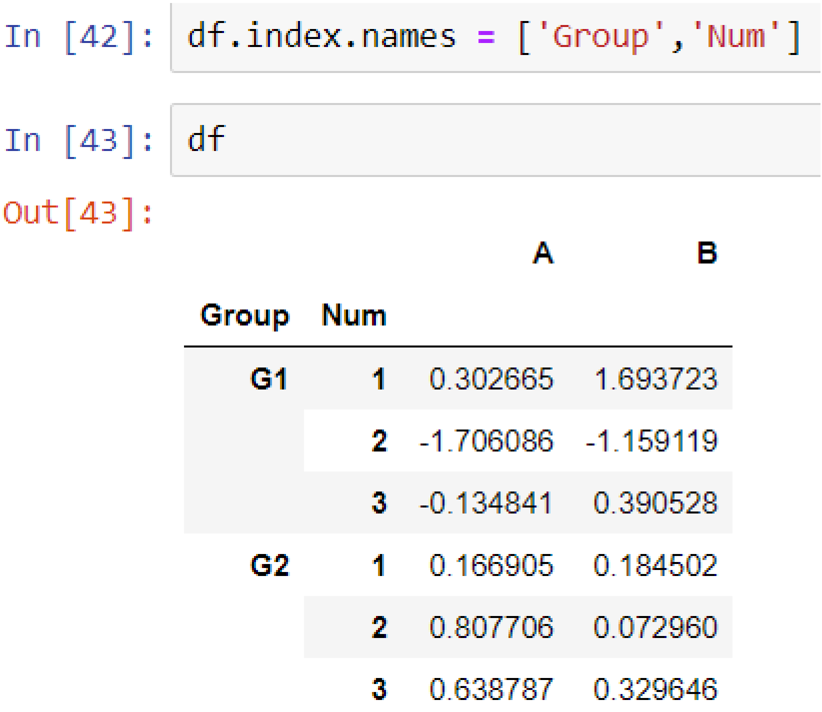
Out[39]:

	A	B
1	0.302665	1.693723
2	-1.706086	-1.159119
3	-0.134841	0.390528

```
In [40]: df.loc['G1'].loc[1]

Out[40]: A    0.302665
         B    1.693723
         Name: 1, dtype: float64
```

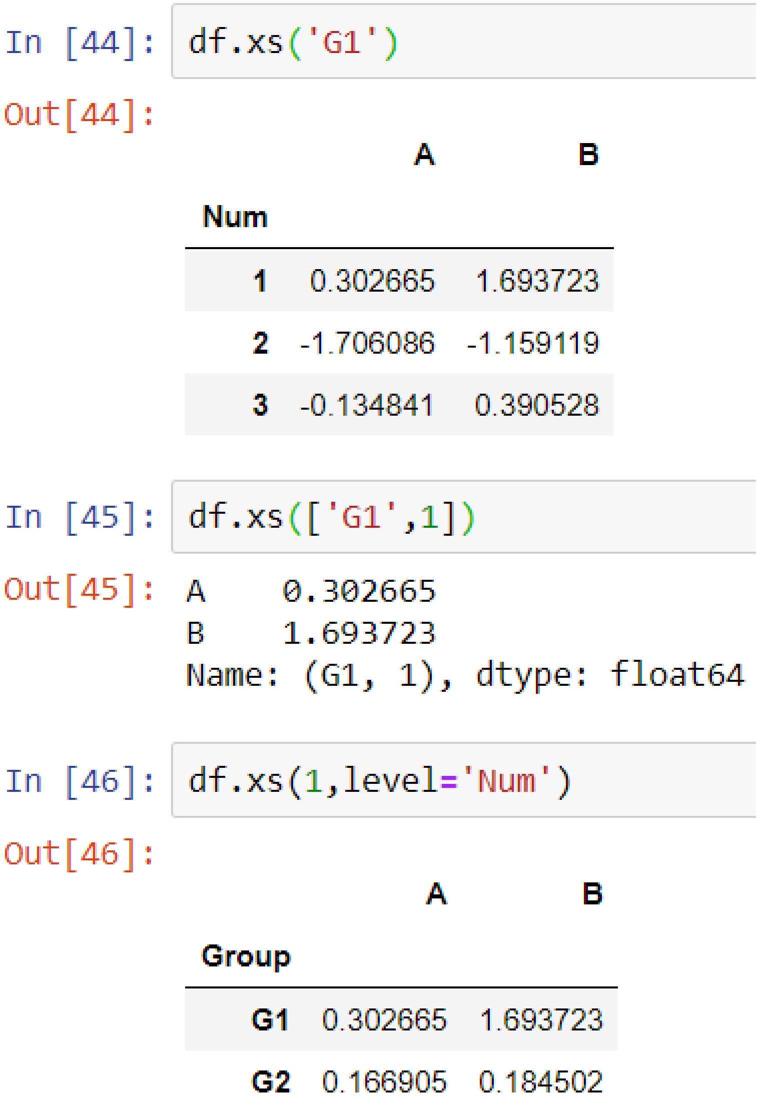
```
In [41]: df.index.names

Out[41]: FrozenList([None, None])
```

Multi-Index and Index Hierarchy



```
In [42]: df.index.names = ['Group','Num']

In [43]: df

Out[43]:
                    A          B
Group  Num
   G1    1   0.302665   1.693723
         2  -1.706086  -1.159119
         3  -0.134841   0.390528
   G2    1   0.166905   0.184502
         2   0.807706   0.072960
         3   0.638787   0.329646
```

Multi-Index and Index Hierarchy



```
In [44]: df.xs('G1')
```

Out[44]:

	A	B
Num		
1	0.302665	1.693723
2	-1.706086	-1.159119
3	-0.134841	0.390528

```
In [45]: df.xs(['G1',1])
```

```
Out[45]: A    0.302665
         B    1.693723
         Name: (G1, 1), dtype: float64
```

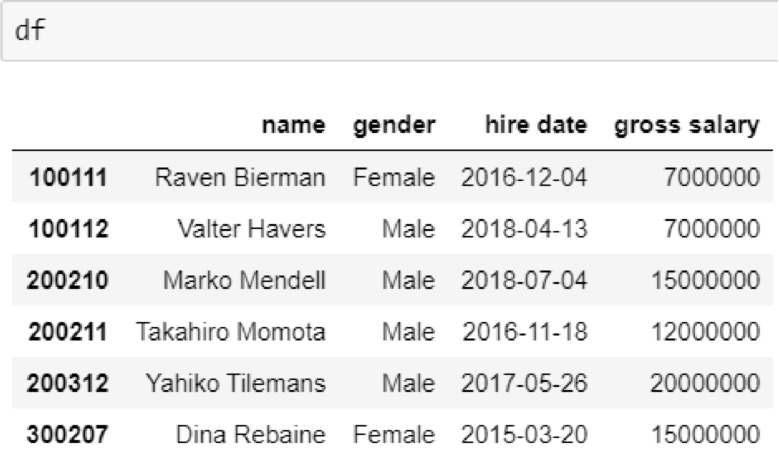
```
In [46]: df.xs(1,level='Num')
```

Out[46]:

	A	B
Group		
G1	0.302665	1.693723
G2	0.166905	0.184502

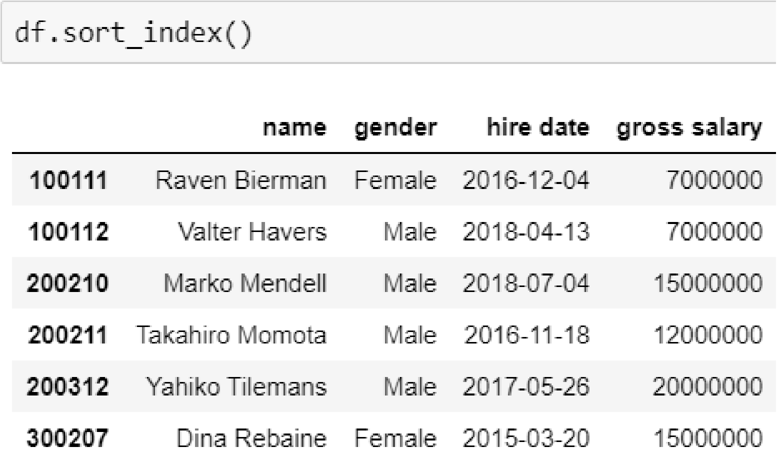
Sorting

Sorting by Index



```
df
```

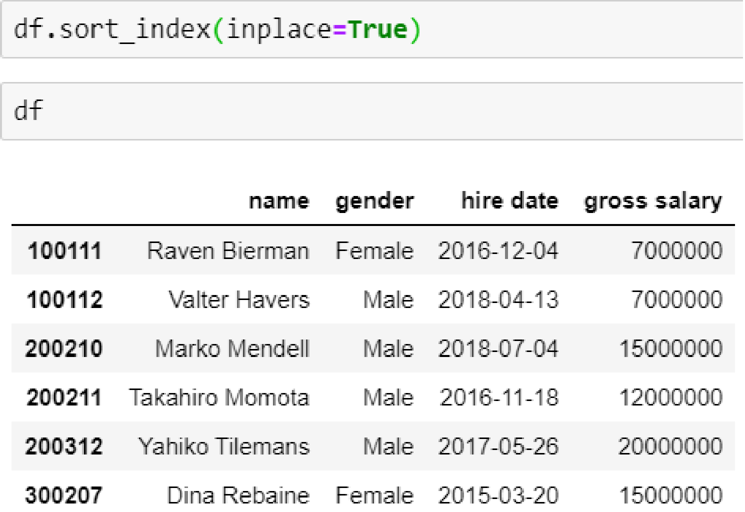
	name	gender	hire date	gross salary
100111	Raven Bierman	Female	2016-12-04	7000000
100112	Valter Havers	Male	2018-04-13	7000000
200210	Marko Mendell	Male	2018-07-04	15000000
200211	Takahiro Momota	Male	2016-11-18	12000000
200312	Yahiko Tilemans	Male	2017-05-26	20000000
300207	Dina Rebaine	Female	2015-03-20	15000000



```
df.sort_index()
```

	name	gender	hire date	gross salary
100111	Raven Bierman	Female	2016-12-04	7000000
100112	Valter Havers	Male	2018-04-13	7000000
200210	Marko Mendell	Male	2018-07-04	15000000
200211	Takahiro Momota	Male	2016-11-18	12000000
200312	Yahiko Tilemans	Male	2017-05-26	20000000
300207	Dina Rebaine	Female	2015-03-20	15000000

Permanently saved the result



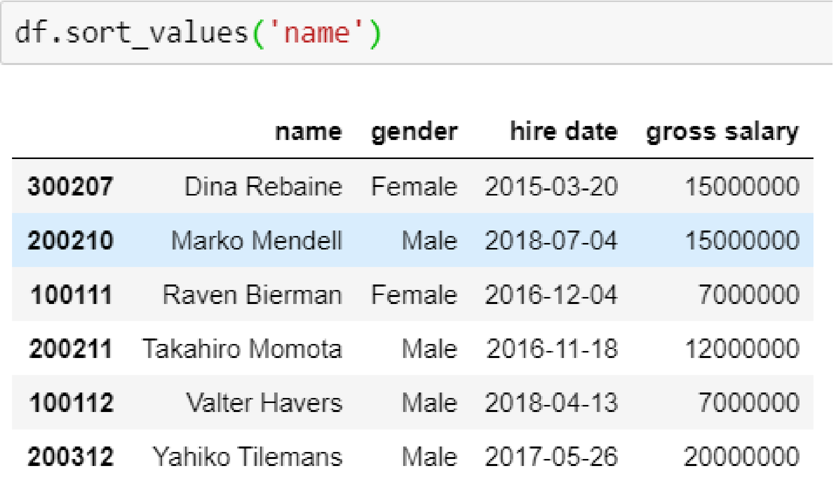
```
df.sort_index(inplace=True)
```

```
df
```

	name	gender	hire date	gross salary
100111	Raven Bierman	Female	2016-12-04	7000000
100112	Valter Havers	Male	2018-04-13	7000000
200210	Marko Mendell	Male	2018-07-04	15000000
200211	Takahiro Momota	Male	2016-11-18	12000000
200312	Yahiko Tilemans	Male	2017-05-26	20000000
300207	Dina Rebaine	Female	2015-03-20	15000000

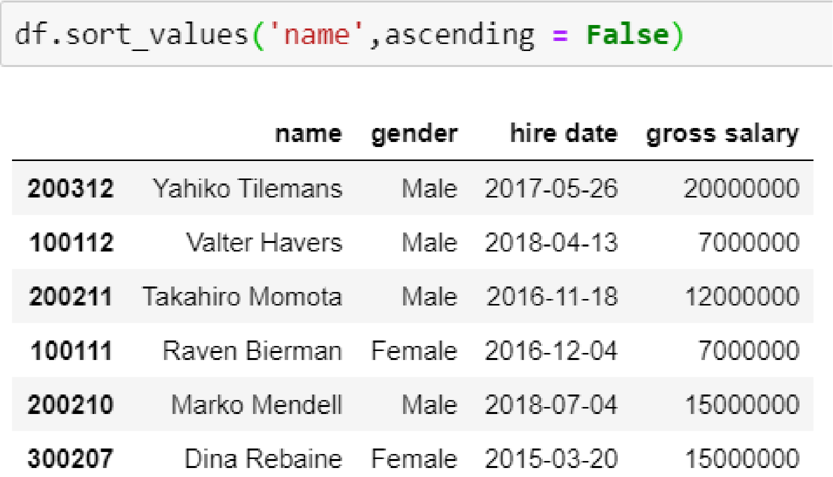
Purwadhika
Startup and Coding School

Sorting by any columns



```
df.sort_values('name')
```

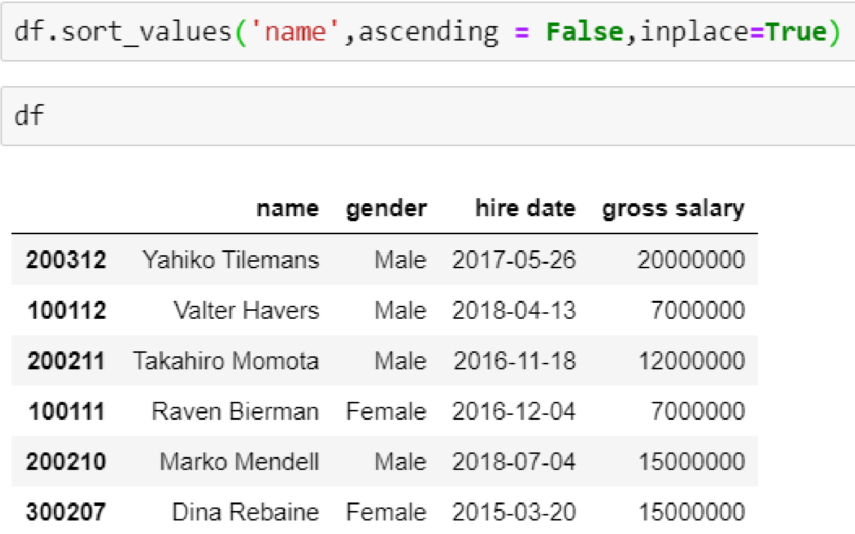
	name	gender	hire date	gross salary
300207	Dina Rebaine	Female	2015-03-20	15000000
200210	Marko Mendell	Male	2018-07-04	15000000
100111	Raven Bierman	Female	2016-12-04	7000000
200211	Takahiro Momota	Male	2016-11-18	12000000
100112	Valter Havers	Male	2018-04-13	7000000
200312	Yahiko Tilemans	Male	2017-05-26	20000000



```
df.sort_values('name',ascending = False)
```

	name	gender	hire date	gross salary
200312	Yahiko Tilemans	Male	2017-05-26	20000000
100112	Valter Havers	Male	2018-04-13	7000000
200211	Takahiro Momota	Male	2016-11-18	12000000
100111	Raven Bierman	Female	2016-12-04	7000000
200210	Marko Mendell	Male	2018-07-04	15000000
300207	Dina Rebaine	Female	2015-03-20	15000000

Permanently saved the result



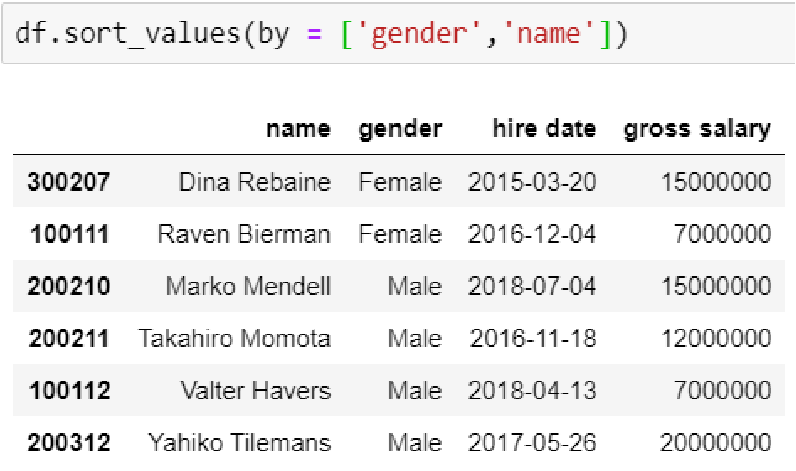
```
df.sort_values('name',ascending = False,inplace=True)
```

```
df
```

	name	gender	hire date	gross salary
200312	Yahiko Tilemans	Male	2017-05-26	20000000
100112	Valter Havers	Male	2018-04-13	7000000
200211	Takahiro Momota	Male	2016-11-18	12000000
100111	Raven Bierman	Female	2016-12-04	7000000
200210	Marko Mendell	Male	2018-07-04	15000000
300207	Dina Rebaine	Female	2015-03-20	15000000

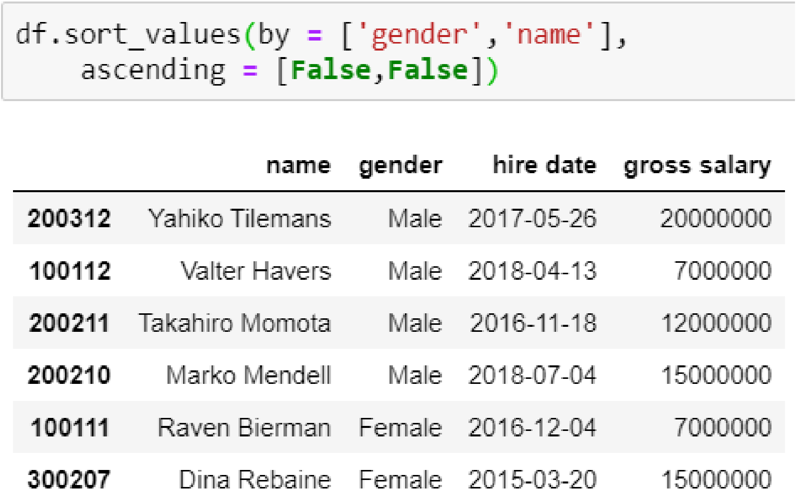
Purwadhika
Startup and Coding School

Sorting by more than one columns



```
df.sort_values(by = ['gender','name'])
```

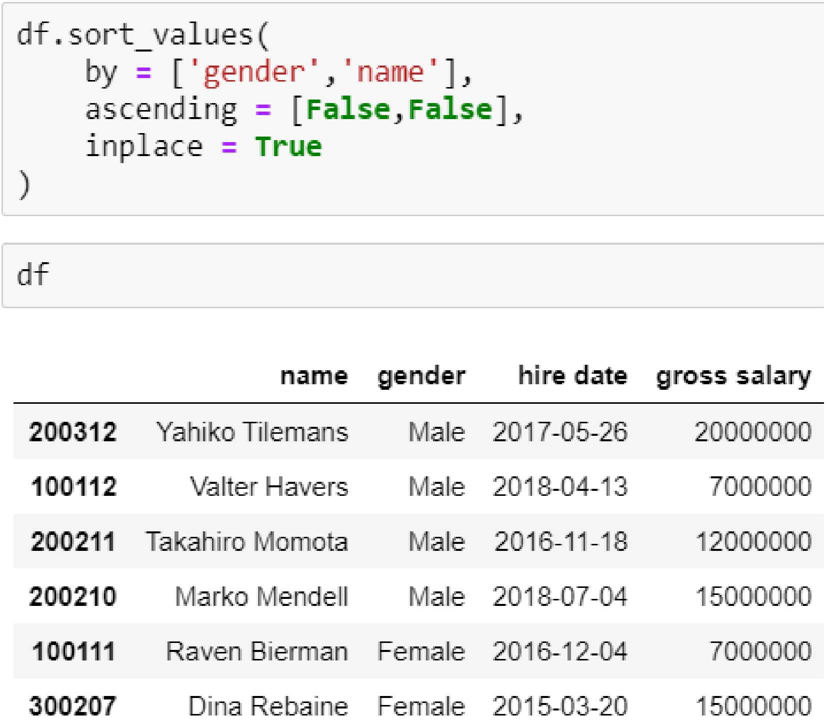
	name	gender	hire date	gross salary
300207	Dina Rebaine	Female	2015-03-20	15000000
100111	Raven Bierman	Female	2016-12-04	7000000
200210	Marko Mendell	Male	2018-07-04	15000000
200211	Takahiro Momota	Male	2016-11-18	12000000
100112	Valter Havers	Male	2018-04-13	7000000
200312	Yahiko Tilemans	Male	2017-05-26	20000000



```
df.sort_values(by = ['gender','name'],
    ascending = [False,False])
```

	name	gender	hire date	gross salary
200312	Yahiko Tilemans	Male	2017-05-26	20000000
100112	Valter Havers	Male	2018-04-13	7000000
200211	Takahiro Momota	Male	2016-11-18	12000000
200210	Marko Mendell	Male	2018-07-04	15000000
100111	Raven Bierman	Female	2016-12-04	7000000
300207	Dina Rebaine	Female	2015-03-20	15000000

Permanently saved the result



```
df.sort_values(
    by = ['gender','name'],
    ascending = [False,False],
    inplace = True
)
```

```
df
```

	name	gender	hire date	gross salary
200312	Yahiko Tilemans	Male	2017-05-26	20000000
100112	Valter Havers	Male	2018-04-13	7000000
200211	Takahiro Momota	Male	2016-11-18	12000000
200210	Marko Mendell	Male	2018-07-04	15000000
100111	Raven Bierman	Female	2016-12-04	7000000
300207	Dina Rebaine	Female	2015-03-20	15000000

Refrences

https://www.researchgate.net/figure/Unstructured-semi-structured-and-structured-data_fig4_236860222
https://orbitingweb.com/blog/view-youtube-thumbnail-image/
https://www.wohler.com/product/audio-monitoring/iam-audio-1/