Data manipulation:
Data wrangling, aggregation, and group operations.

AAA-Python Edition

Plan

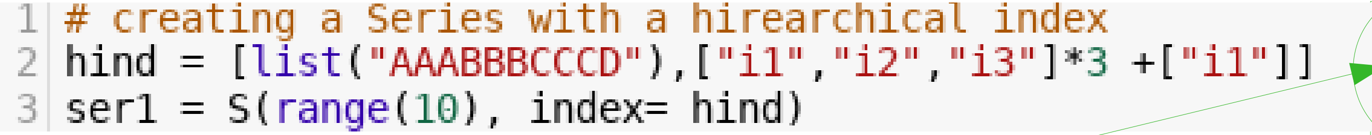
- 1- Hierarchical indexing
- 2- Combining and merging Data Sets
- 3- Reshaping and pivoting
- 4- Group by Mechanics
- 5- Data aggregation
- 6- Other aggregation operations

1- Hierarchical indexing

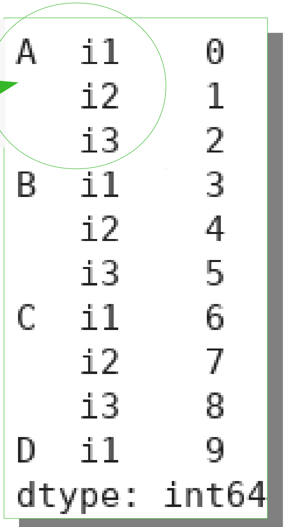
Data Wrangling and hierarchical indexing

- **Data wrangling** is the process of **cleaning** and **unifying** **messy** and **complex** data sets for **easy access** and **analysis**. (from: https://www.datawatch.com/what-is-data-wrangling/)
- **Hierarchical indexing**: is the use of **multiple** **indexes** at different **levels**

ser1



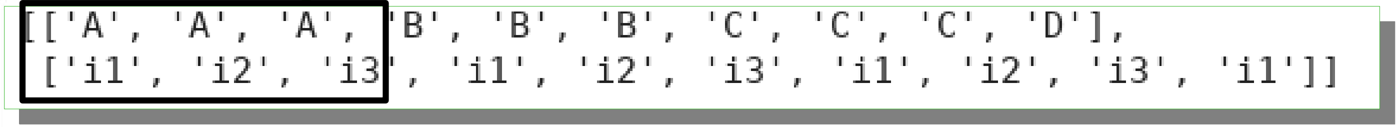
```
# creating a Series with a hirearchical index
hind = [list("AAABBBCCCD"),["i1","i2","i3"]*3 +["i1"]]
ser1 = S(range(10), index= hind)
```



```
A  i1    0
   i2    1
   i3    2
B  i1    3
   i2    4
   i3    5
C  i1    6
   i2    7
   i3    8
D  i1    9
dtype: int64
```

hind

i1, **i2** , **i3** will be In the level **A**

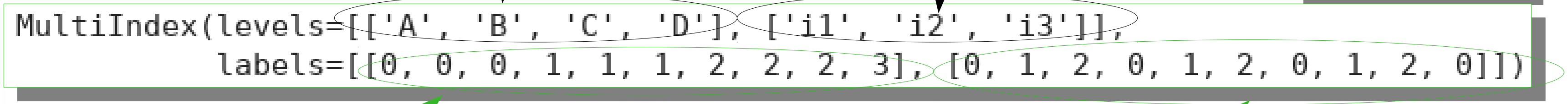


```
[['A', 'A', 'A', 'B', 'B', 'B', 'C', 'C', 'C', 'D'],
 ['i1', 'i2', 'i3', 'i1', 'i2', 'i3', 'i1', 'i2', 'i3', 'i1']]
```

ser1.index

level 1

level 2



```
MultiIndex(levels=[['A', 'B', 'C', 'D'], ['i1', 'i2', 'i3']],
           labels=[[0, 0, 0, 1, 1, 1, 2, 2, 2, 3], [0, 1, 2, 0, 1, 2, 0, 1, 2, 0]])
```

Indices of level 1

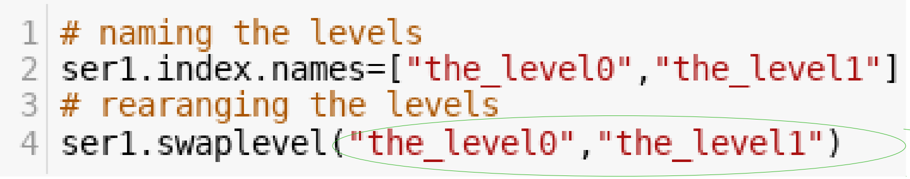
Indices of level 2

[By Amina Delali]

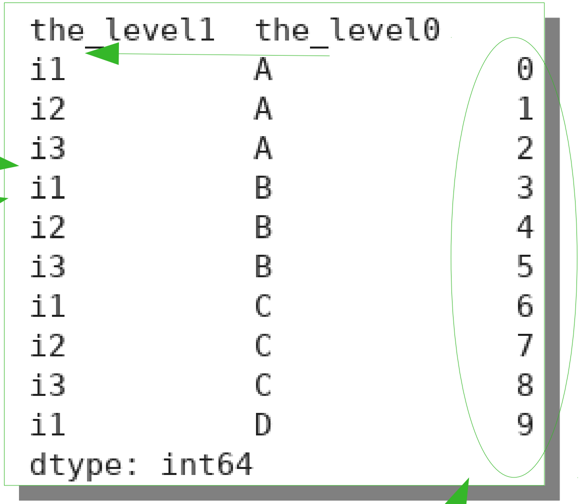
1- Hierarchical indexing

Reordering and sorting

- Reordering enables **interchanging the index levels** using the **swaplevel** method
- Sorting enables **sorting** the **data** by sorting **one level values**, using the **sort_index** method.



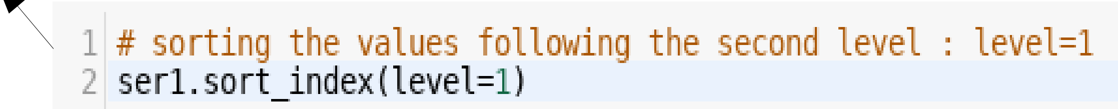
```
# naming the levels
ser1.index.names=["the_level0","the_level1"]
# rearanging the levels
ser1.swaplevel("the_level0","the_level1")
```



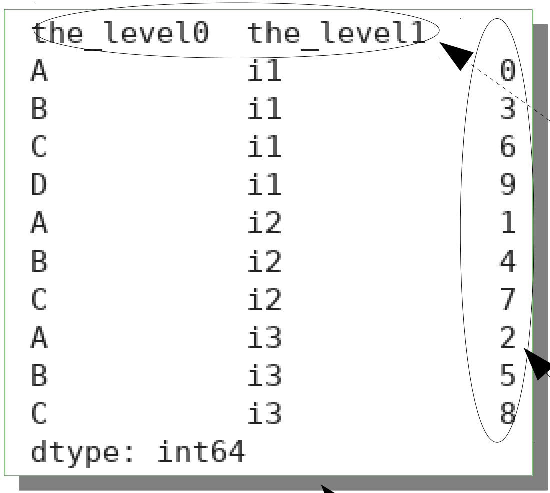
```
the_level1  the_level0
i1          A             0
i2          A             1
i3          A             2
i1          B             3
i2          B             4
i3          B             5
i1          C             6
i2          C             7
i3          C             8
i1          D             9
dtype: int64
```

The hierarchy of the indexes changed

The order of the data (so the indexes too) remains the same



```
# sorting the values following the second level : level=1
ser1.sort_index(level=1)
```



```
the_level0  the_level1
A           i1            0
B           i1            3
C           i1            6
D           i1            9
A           i2            1
B           i2            4
C           i2            7
A           i3            2
B           i3            5
C           i3            8
dtype: int64
```

The hierarchy of the Indexes didn’t change

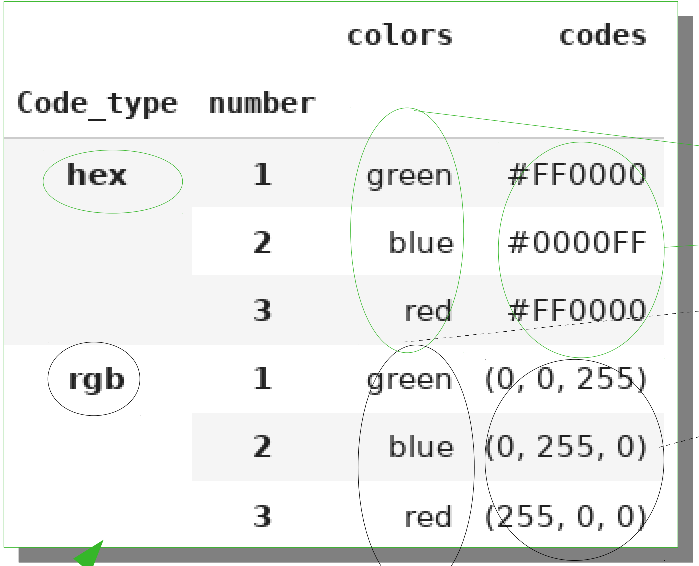
The order of the data (so the indexes too) changed: **the_level1** index was **sorted**

[By Amina Delali]

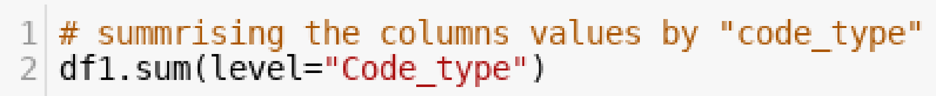
1- Hierarchical indexing

Operations by level

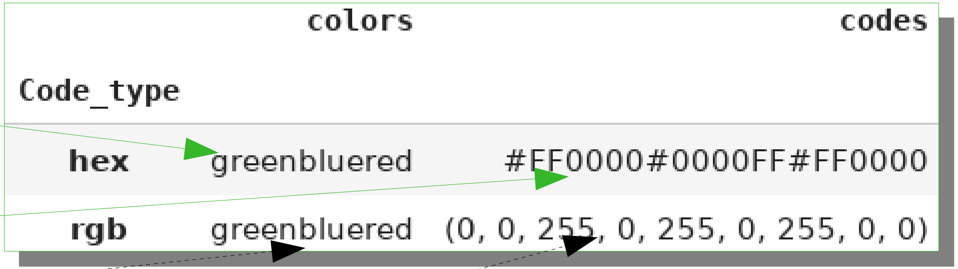
df1



		colors	codes
Code_type	number		
hex	1	green	#FF0000
	2	blue	#0000FF
	3	red	#FF0000
rgb	1	green	(0, 0, 255)
	2	blue	(0, 255, 0)
	3	red	(255, 0, 0)

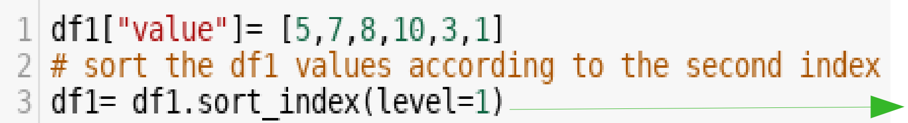


```python
# summrising the columns values by "code_type"
df1.sum(level="Code_type")
```



	colors	codes
Code_type		
hex	greenbluered	#FF0000#0000FF#FF0000
rgb	greenbluered	(0, 0, 255, 0, 255, 0, 255, 0, 0)

If sum was applied on number,
It would perform and addition instead
Of this concatenation



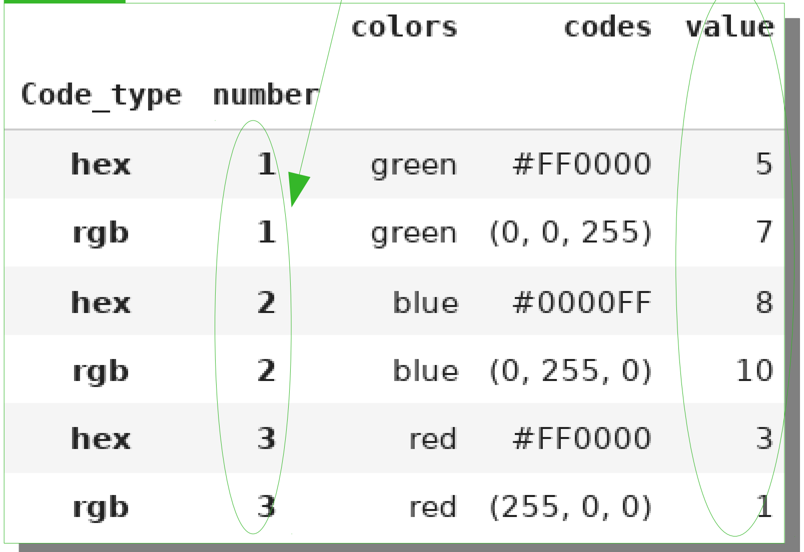
```python
df1["value"]= [5,7,8,10,3,1]
# sort the df1 values according to the second index
df1= df1.sort_index(level=1)
```

[By Amina Delali]

df1

sorted

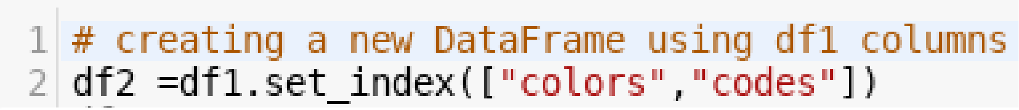
A new column added



		colors	codes	value
Code_type	number			
hex	1	green	#FF0000	5
rgb	1	green	(0, 0, 255)	7
hex	2	blue	#0000FF	8
rgb	2	blue	(0, 255, 0)	10
hex	3	red	#FF0000	3
rgb	3	red	(255, 0, 0)	1

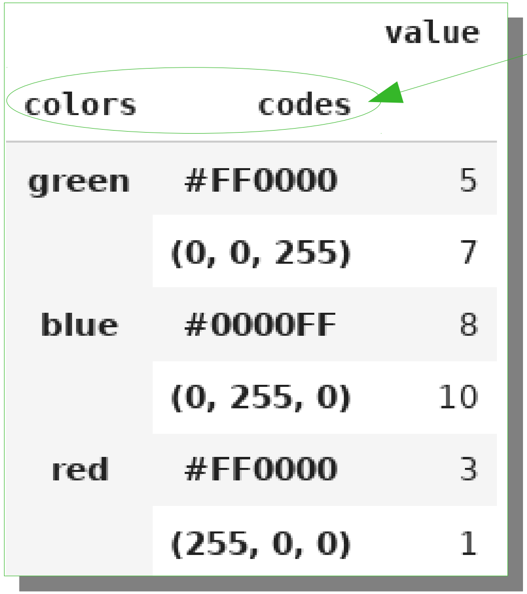
1- Hierarchical indexing

indexing



```
# creating a new DataFrame using df1 columns
df2 =df1.set_index(["colors","codes"])
```

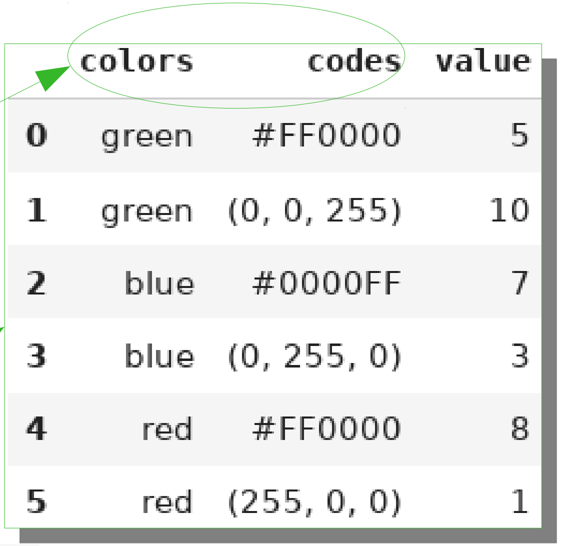
df2



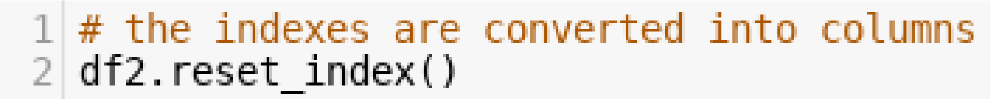
		value
colors	codes	
green	#FF0000	5
	(0, 0, 255)	7
blue	#0000FF	8
	(0, 255, 0)	10
red	#FF0000	3
	(255, 0, 0)	1

The previous df1 **columns** are now **indexes**

The **indexes** are converted into **columns**



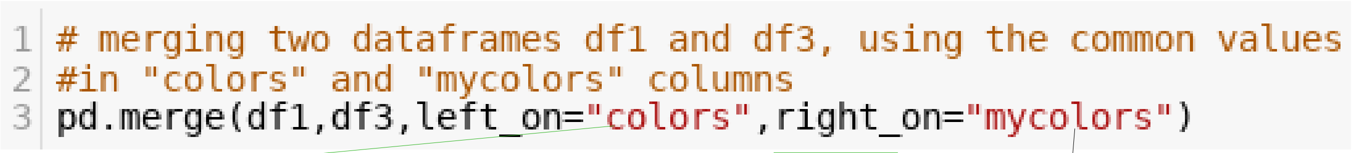
	colors	codes	value
0	green	#FF0000	5
1	green	(0, 0, 255)	10
2	blue	#0000FF	7
3	blue	(0, 255, 0)	3
4	red	#FF0000	8
5	red	(255, 0, 0)	1



```
# the indexes are converted into columns
df2.reset_index()
```

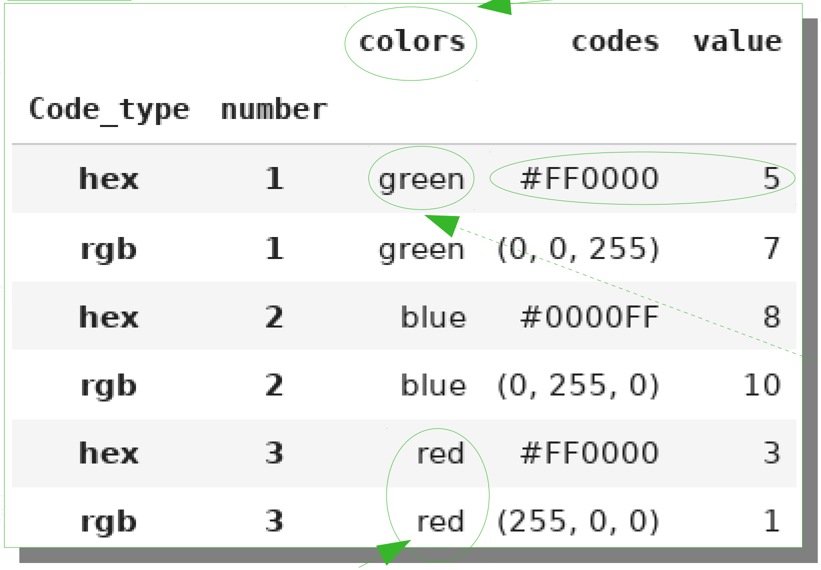
[By Amina Delali]

merge



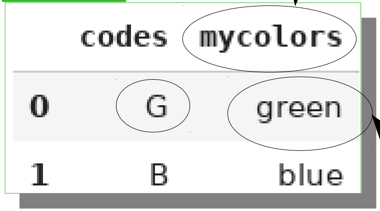
```
# merging two dataframes df1 and df3, using the common values
#in "colors" and "mycolors" columns
pd.merge(df1,df3,left_on="colors",right_on="mycolors")
```

df1



		colors	codes	value
Code_type	**number**			
hex	**1**	green	#FF0000	5
rgb	**1**	green	(0, 0, 255)	7
hex	**2**	blue	#0000FF	8
rgb	**2**	blue	(0, 255, 0)	10
hex	**3**	red	#FF0000	3
rgb	**3**	red	(255, 0, 0)	1

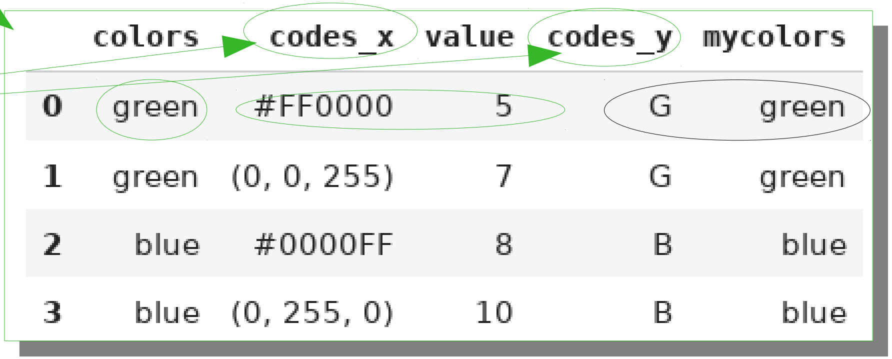
df3



	codes	mycolors
0	G	green
1	B	blue

Each row from **df1** with “**green**” value in “**colors**” column, will be combined with **each** row from **df3** with “**green**” value in “**mycolors**” column.

“**red”** rows weren’t included

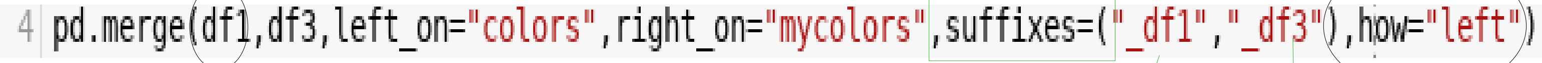


	colors	codes_x	value	codes_y	mycolors
0	green	#FF0000	5	G	green
1	green	(0, 0, 255)	7	G	green
2	blue	#0000FF	8	B	blue
3	blue	(0, 255, 0)	10	B	blue

Both **df1** and **df2** have “codes” column, so “**_x**” and “**_y**” suffixes were added.

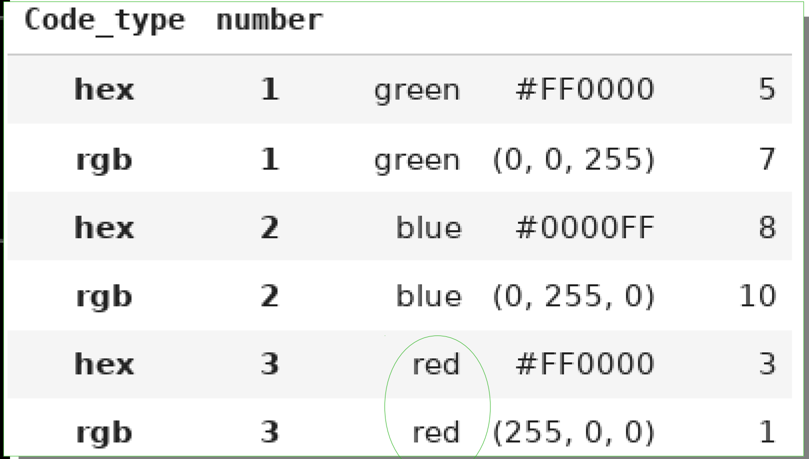
[By Amina Delali]

merge



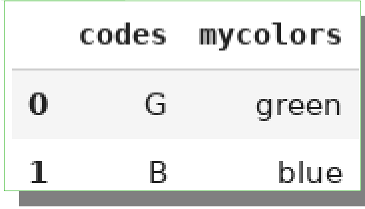
```
pd.merge(df1,df3,left_on="colors",right_on="mycolors",suffixes=("_df1","_df3"),how="left")
```

df1

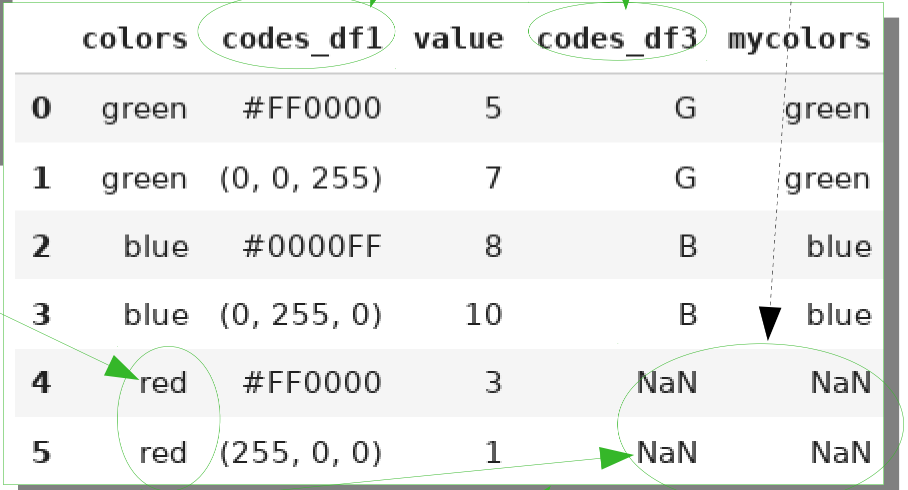


Code_type	number			
hex	1	green	#FF0000	5
rgb	1	green	(0, 0, 255)	7
hex	2	blue	#0000FF	8
rgb	2	blue	(0, 255, 0)	10
hex	3	red	#FF0000	3
rgb	3	red	(255, 0, 0)	1

df3



	codes	mycolors
0	G	green
1	B	blue



	colors	codes_df1	value	codes_df3	mycolors
0	green	#FF0000	5	G	green
1	green	(0, 0, 255)	7	G	green
2	blue	#0000FF	8	B	blue
3	blue	(0, 255, 0)	10	B	blue
4	red	#FF0000	3	NaN	NaN
5	red	(255, 0, 0)	1	NaN	NaN

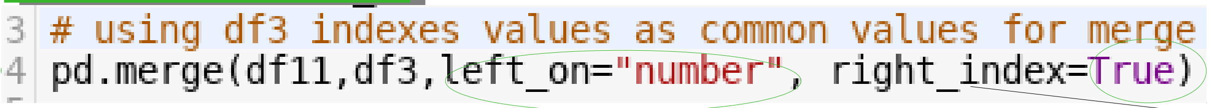
Specifying the argument “**how”** as “**left**”,all rows from **df1** were included (even if no matching value exists in **df3**)

The suffixes argument used to customize the suffixes added to columns with the same name

[By Amina Delali]

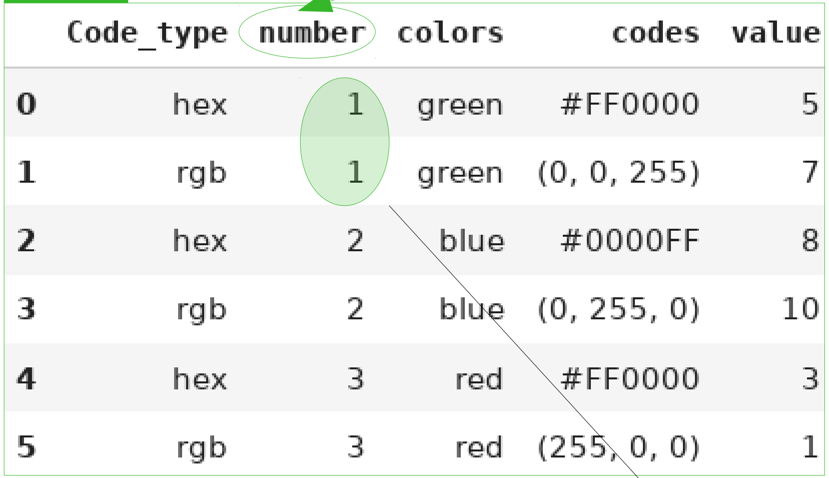
2- Combining and merging Data Sets

merge



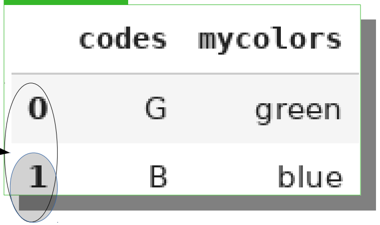
```
# using df3 indexes values as common values for merge
pd.merge(df11,df3,left_on="number", right_index=True)
```

df11

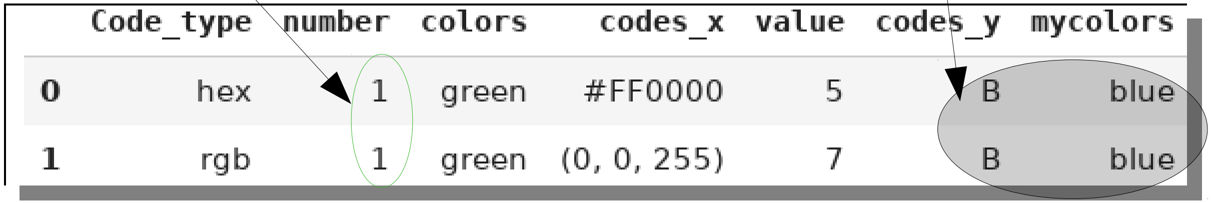


	Code_type	number	colors	codes	value
0	hex	1	green	#FF0000	5
1	rgb	1	green	(0, 0, 255)	7
2	hex	2	blue	#0000FF	8
3	rgb	2	blue	(0, 255, 0)	10
4	hex	3	red	#FF0000	3
5	rgb	3	red	(255, 0, 0)	1

df3



	codes	mycolors
0	G	green
1	B	blue



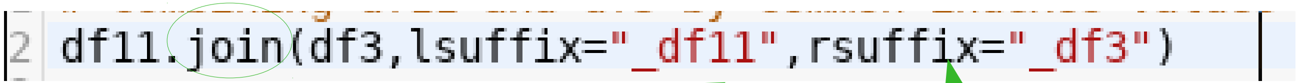
	Code_type	number	colors	codes_x	value	codes_y	mycolors
0	hex	1	green	#FF0000	5	B	blue
1	rgb	1	green	(0, 0, 255)	7	B	blue

Combining df11 and df3 by matching values from “number” column from df11, with Index values frm df3

[By Amina Delali]

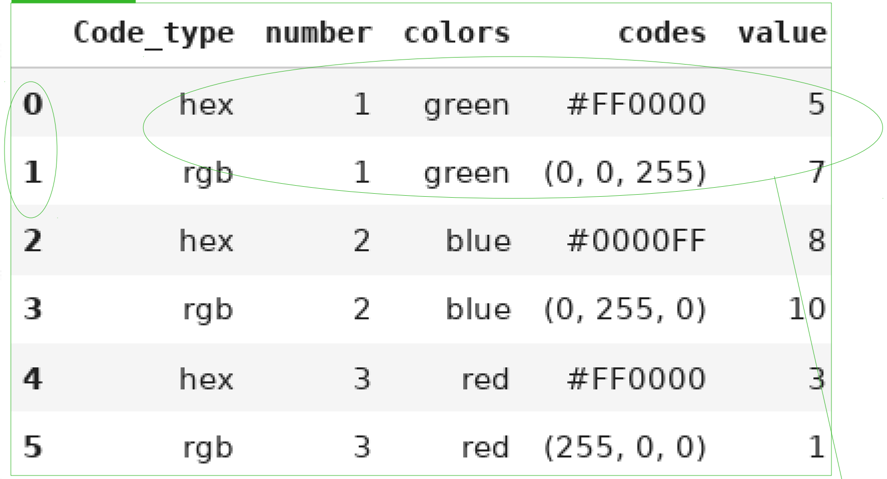
2- Combining and merging Data Sets

join



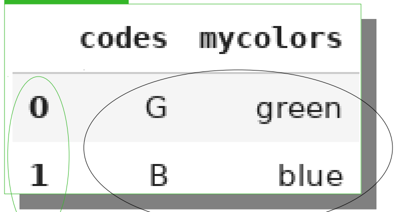
```
df11.join(df3,lsuffix="_df11",rsuffix="_df3")
```

df11



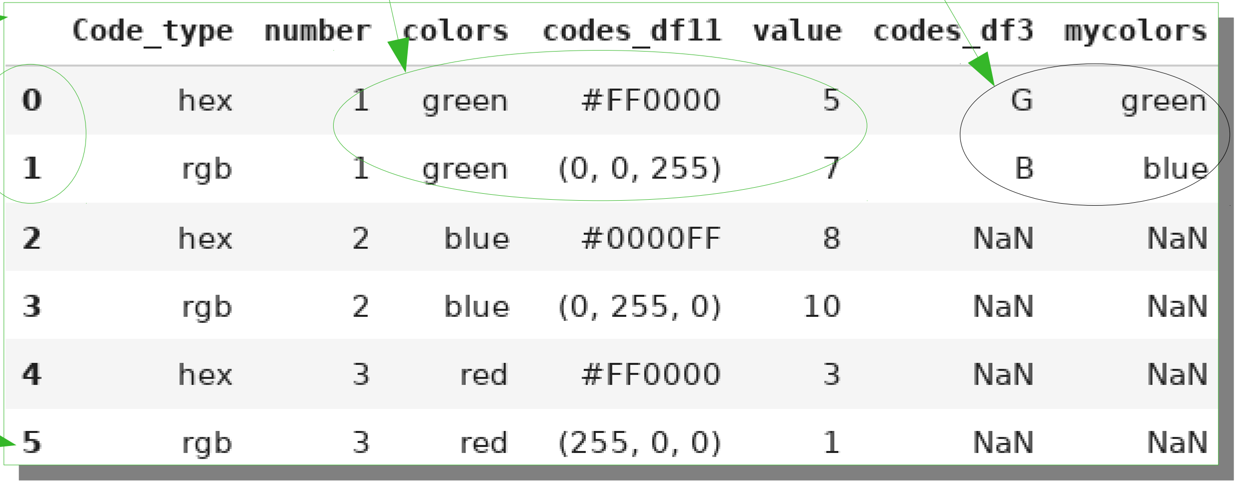
	Code_type	number	colors	codes	value
0	hex	1	green	#FF0000	5
1	rgb	1	green	(0, 0, 255)	7
2	hex	2	blue	#0000FF	8
3	rgb	2	blue	(0, 255, 0)	10
4	hex	3	red	#FF0000	3
5	rgb	3	red	(255, 0, 0)	1

df3



	codes	mycolors
0	G	green
1	B	blue

You have to specify the suffixes if the dataframes have coulmns with same names



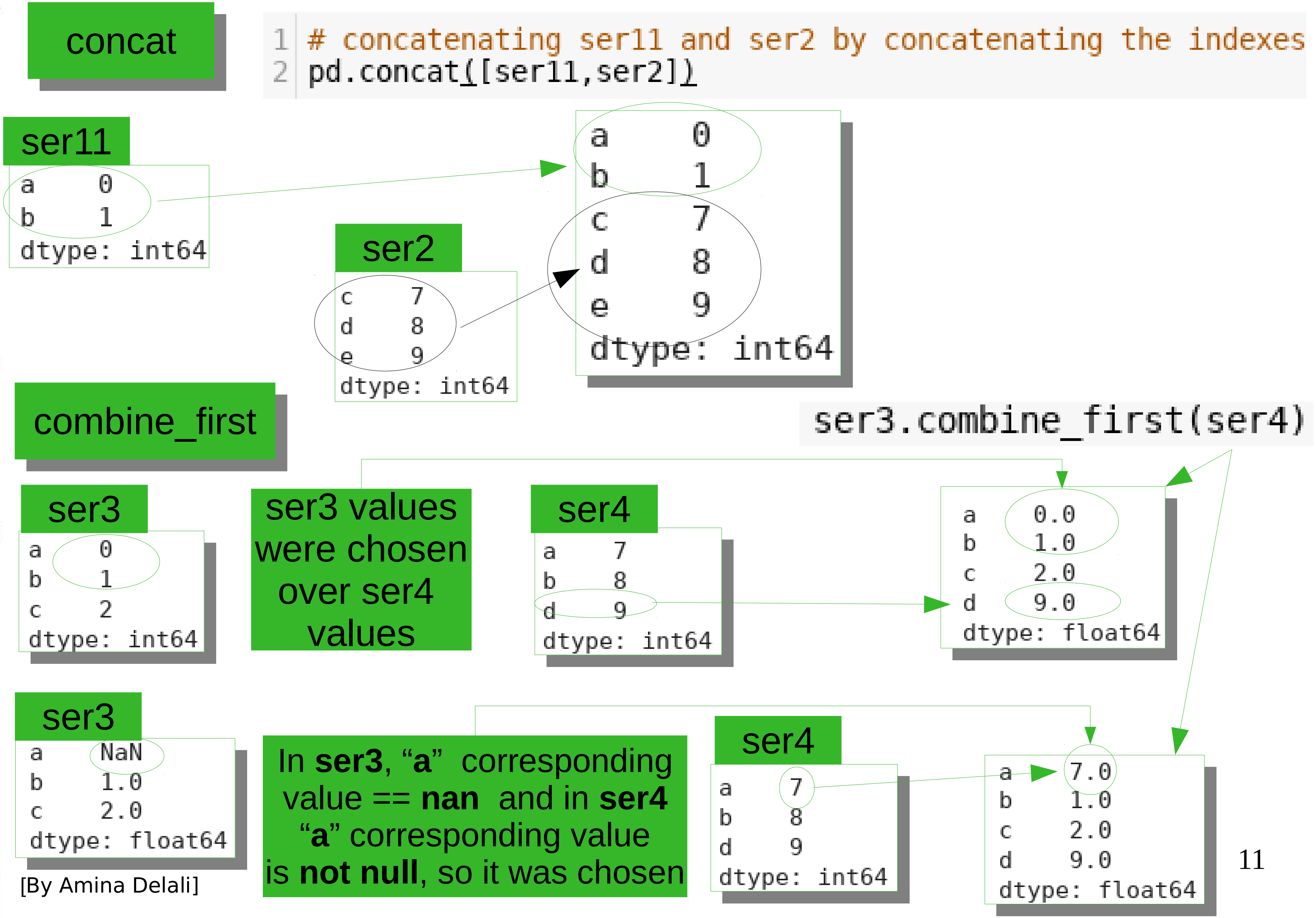
	Code_type	number	colors	codes_df11	value	codes_df3	mycolors
0	hex	1	green	#FF0000	5	G	green
1	rgb	1	green	(0, 0, 255)	7	B	blue
2	hex	2	blue	#0000FF	8	NaN	NaN
3	rgb	2	blue	(0, 255, 0)	10	NaN	NaN
4	hex	3	red	#FF0000	3	NaN	NaN
5	rgb	3	red	(255, 0, 0)	1	NaN	NaN

The dataframes are combined by matching indexes values

By default all the values of **df11** were added

[By Amina Delali]

2- Combining and merging Data Sets



concat

```
# concatenating ser11 and ser2 by concatenating the indexes
pd.concat([ser11,ser2])
```

ser11

```
a    0
b    1
dtype: int64
```

ser2

```
c    7
d    8
e    9
dtype: int64
```

```
a    0
b    1
c    7
d    8
e    9
dtype: int64
```

combine_first

```
ser3.combine_first(ser4)
```

ser3

```
a    0
b    1
c    2
dtype: int64
```

ser3 values were chosen over ser4 values

ser4

```
a    7
b    8
d    9
dtype: int64
```

```
a    0.0
b    1.0
c    2.0
d    9.0
dtype: float64
```

ser3

```
a    NaN
b    1.0
c    2.0
dtype: float64
```

In **ser3**, “**a**” corresponding value == **nan** and in **ser4** “**a**” corresponding value is **not null**, so it was chosen

ser4

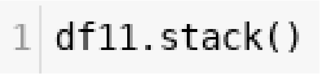
```
a    7
b    8
d    9
dtype: int64
```

```
a    7.0
b    1.0
c    2.0
d    9.0
dtype: float64
```

[By Amina Delali]

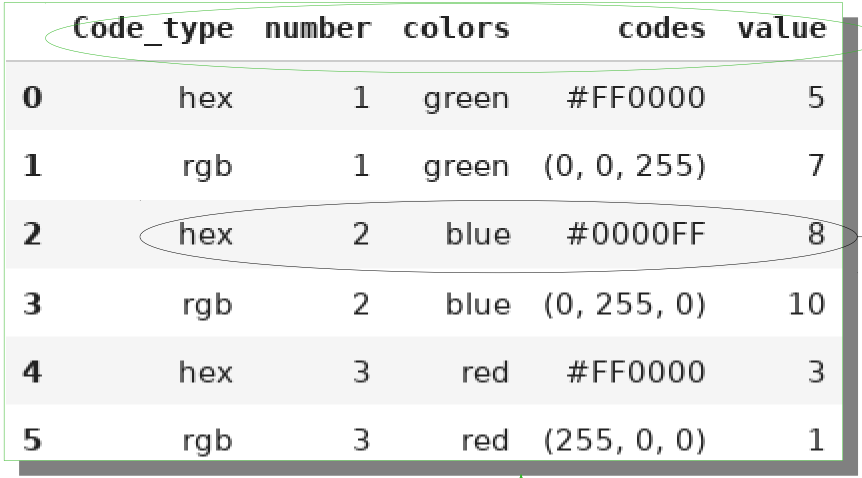
3- Reshaping and pivoting

stack & unstack

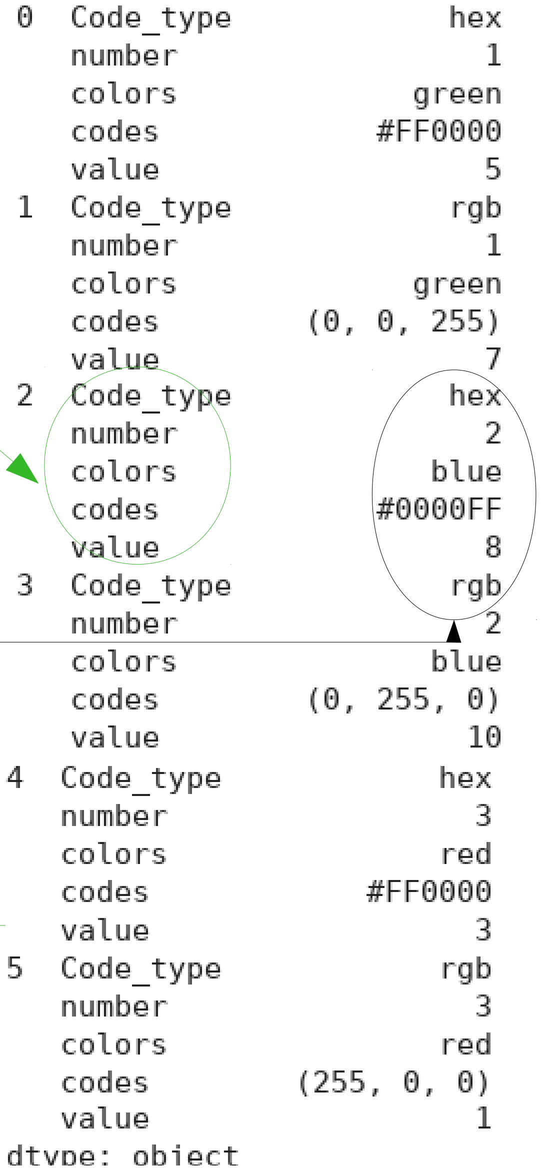


```
df11.stack()
```

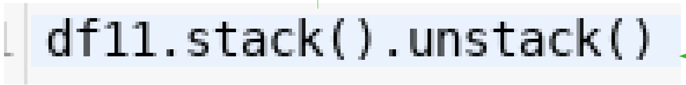
- **stack**: pivot **columns** label to **rows** indexes
- **unstack**: pivot **rows** indexes to **columns** labels



	Code_type	number	colors	codes	value
0	hex	1	green	#FF0000	5
1	rgb	1	green	(0, 0, 255)	7
2	hex	2	blue	#0000FF	8
3	rgb	2	blue	(0, 255, 0)	10
4	hex	3	red	#FF0000	3
5	rgb	3	red	(255, 0, 0)	1



```
0  Code_type           hex
   number                1
   colors            green
   codes           #FF0000
   value                 5
1  Code_type           rgb
   number                1
   colors            green
   codes       (0, 0, 255)
   value                 7
2  Code_type           hex
   number                2
   colors             blue
   codes           #0000FF
   value                 8
3  Code_type           rgb
   number                2
   colors             blue
   codes       (0, 255, 0)
   value                10
4  Code_type           hex
   number                3
   colors              red
   codes           #FF0000
   value                 3
5  Code_type           rgb
   number                3
   colors              red
   codes       (255, 0, 0)
   value                 1
dtype: object
```



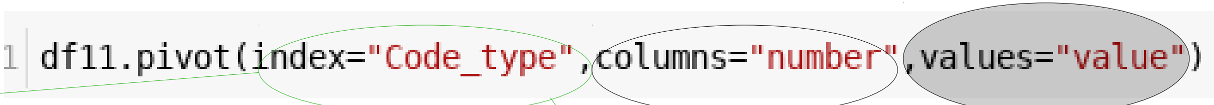
```
df11.stack().unstack()
```

[By Amina Delali]

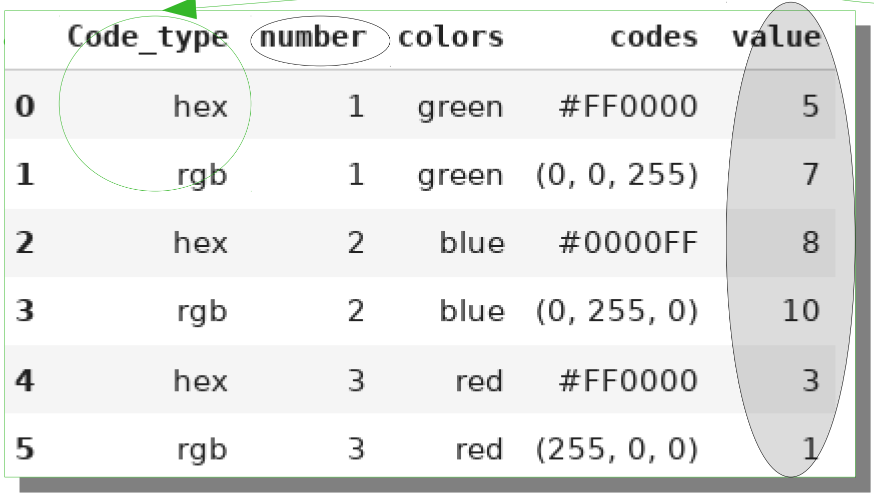
school of a

3- Reshaping and pivoting

Pivot (long to wide)



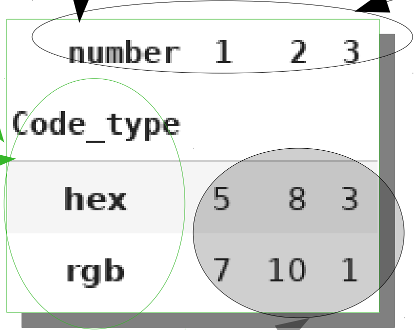
```
df11.pivot(index="Code_type",columns="number",values="value")
```



	Code_type	number	colors	codes	value
0	hex	1	green	#FF0000	5
1	rgb	1	green	(0, 0, 255)	7
2	hex	2	blue	#0000FF	8
3	rgb	2	blue	(0, 255, 0)	10
4	hex	3	red	#FF0000	3
5	rgb	3	red	(255, 0, 0)	1

3 unique values
3 columns

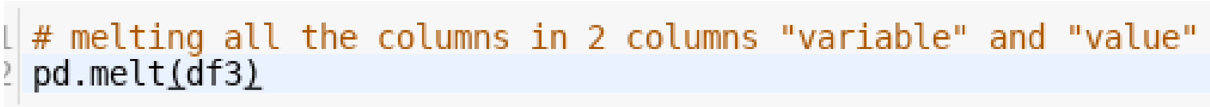
Only **2** unique values for **Code_type**



number	1	2	3
Code_type			
hex	5	8	3
rgb	7	10	1

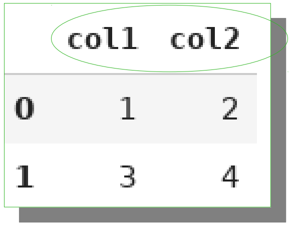
The values are obtained from “**value**”by matching “**Code_type**”and “**number**” values

Melt (wide to long)

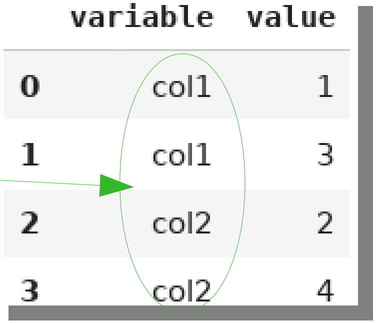


```
# melting all the columns in 2 columns "variable" and "value"
pd.melt(df3)
```

df3



	col1	col2
0	1	2
1	3	4



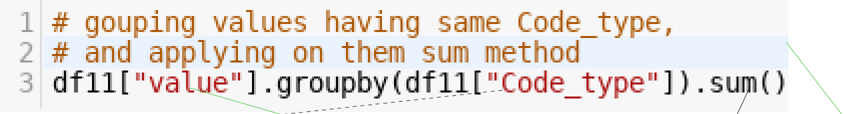
	variable	value
0	col1	1
1	col1	3
2	col2	2
3	col2	4

[By Amina Delali]

school of a

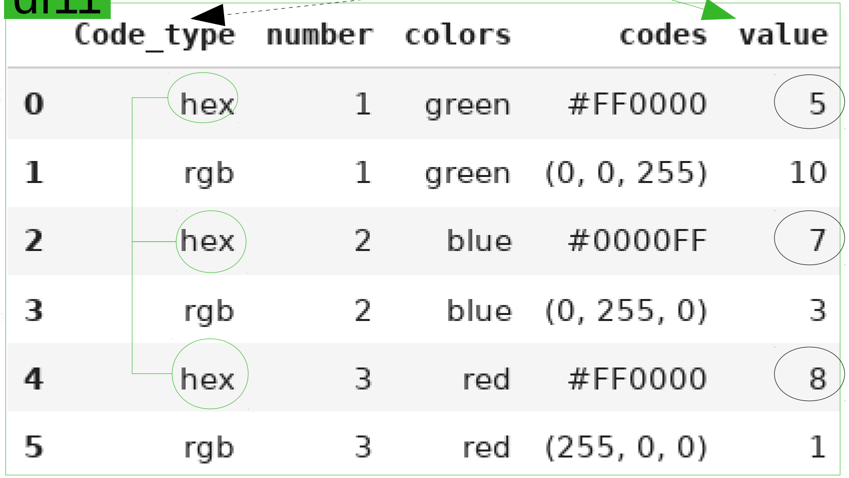
4- Group by Mechanics

groupby



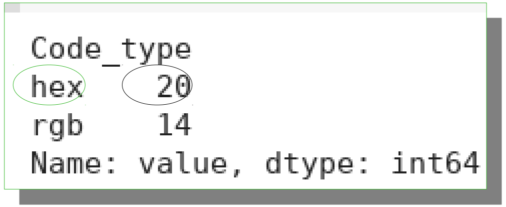
```
# gouping values having same Code_type,
# and applying on them sum method
df11["value"].groupby(df11["Code_type"]).sum()
```

df11

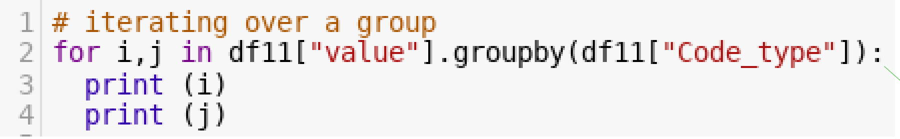


	Code_type	number	colors	codes	value
0	hex	1	green	#FF0000	5
1	rgb	1	green	(0, 0, 255)	10
2	hex	2	blue	#0000FF	7
3	rgb	2	blue	(0, 255, 0)	3
4	hex	3	red	#FF0000	8
5	rgb	3	red	(255, 0, 0)	1

5 + 7 +8 = 20



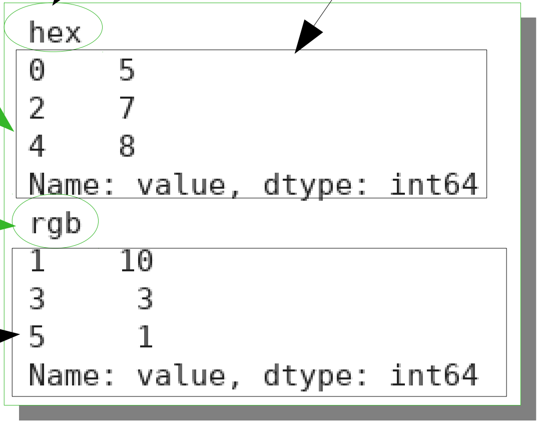
```
Code_type
hex     20
rgb     14
Name: value, dtype: int64
```



```
# iterating over a group
for i,j in df11["value"].groupby(df11["Code_type"]):
  print (i)
  print (j)
```

i

j



```
hex
0     5
2     7
4     8
Name: value, dtype: int64
rgb
1     10
3      3
5      1
Name: value, dtype: int64
```

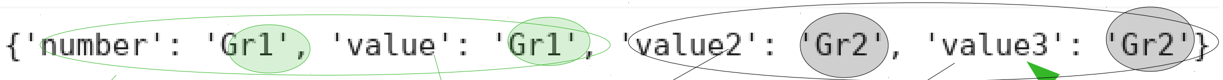
i

j

[By Amina Delali]

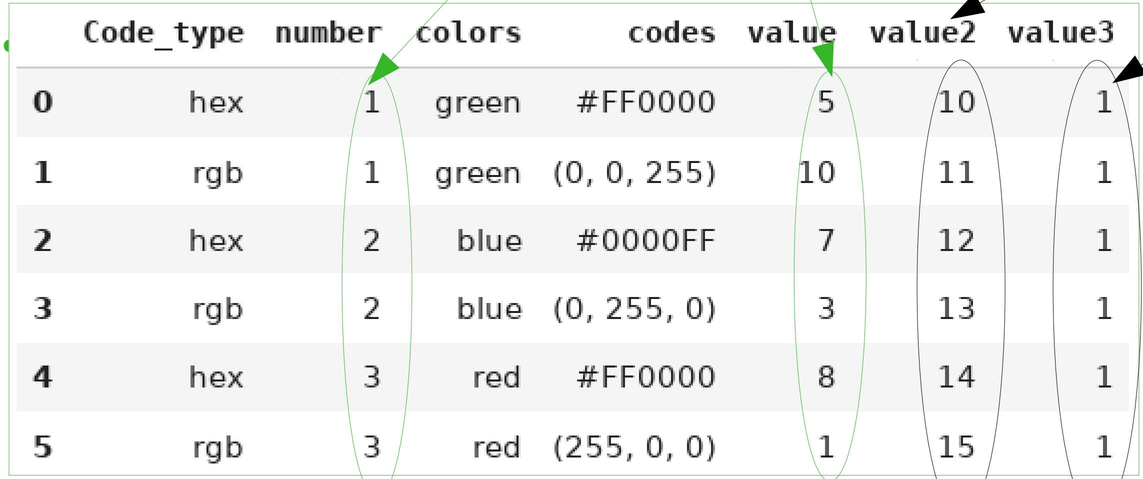
4- Group by Mechanics

groupby

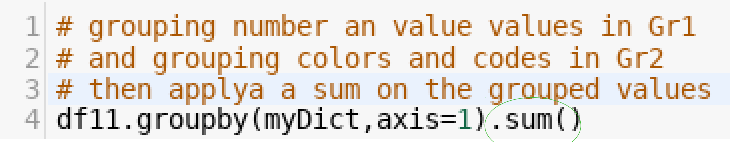


```
{'number': 'Gr1', 'value': 'Gr1', 'value2': 'Gr2', 'value3': 'Gr2'}
```

myDict

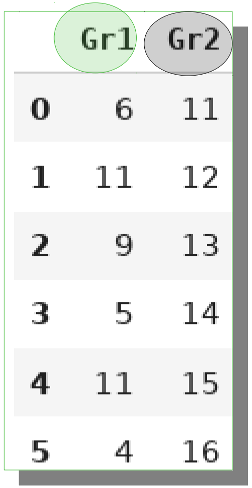


	Code_type	number	colors	codes	value	value2	value3
0	hex	1	green	#FF0000	5	10	1
1	rgb	1	green	(0, 0, 255)	10	11	1
2	hex	2	blue	#0000FF	7	12	1
3	rgb	2	blue	(0, 255, 0)	3	13	1
4	hex	3	red	#FF0000	8	14	1
5	rgb	3	red	(255, 0, 0)	1	15	1



```
# grouping number an value values in Gr1
# and grouping colors and codes in Gr2
# then applya a sum on the grouped values
df11.groupby(myDict,axis=1).sum()
```

Gr1 values are summed together.
And Gr2 values are also summed together



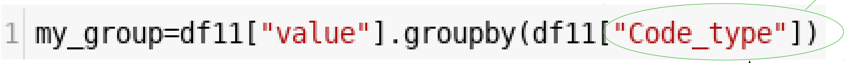
	Gr1	Gr2
0	6	11
1	11	12
2	9	13
3	5	14
4	11	15
5	4	16

[By Amina Delali]

school of ai

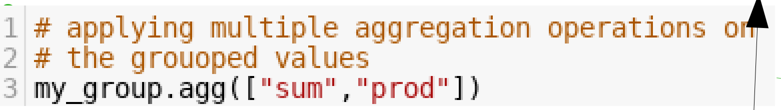
5- Data aggregation

agg

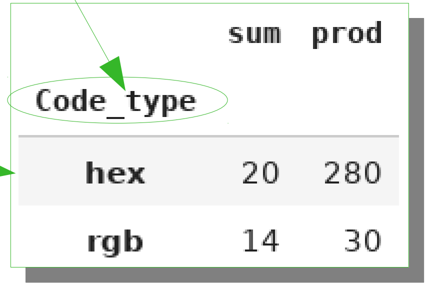


```
my_group=df11["value"].groupby(df11["Code_type"])
```

Became an index



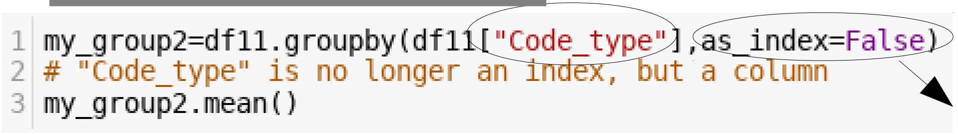
```
# applying multiple aggregation operations on
# the grouoped values
my_group.agg(["sum","prod"])
```



	sum	prod
Code_type		
hex	20	280
rgb	14	30

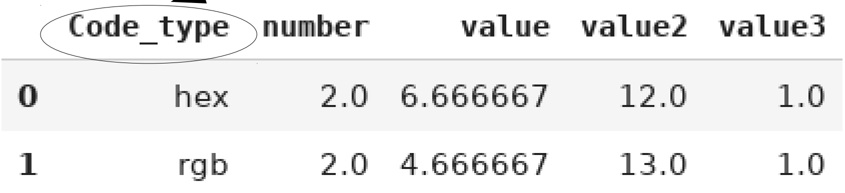
We could just write:
groupby(“Code_type”)

Columns kept columns



```
my_group2=df11.groupby(df11["Code_type"],as_index=False)
# "Code_type" is no longer an index, but a column
my_group2.mean()
```

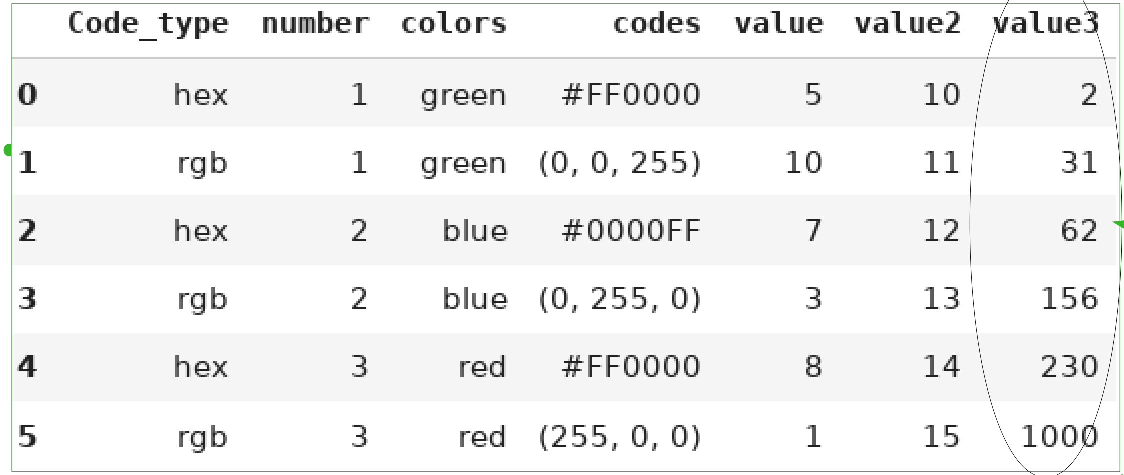
Remains a column



	Code_type	number	value	value2	value3
0	hex	2.0	6.666667	12.0	1.0
1	rgb	2.0	4.666667	13.0	1.0

[By Amina Delali]

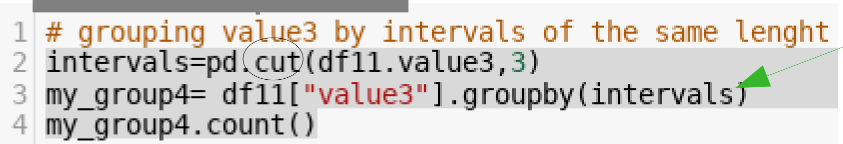
6- Other aggregation operations



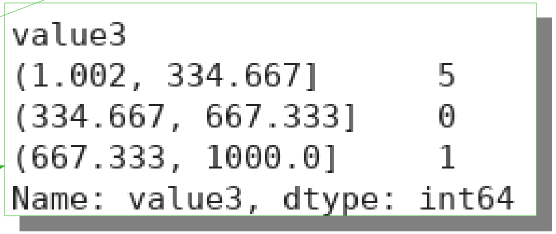
	Code_type	number	colors	codes	value	value2	value3
0	hex	1	green	#FF0000	5	10	2
1	rgb	1	green	(0, 0, 255)	10	11	31
2	hex	2	blue	#0000FF	7	12	62
3	rgb	2	blue	(0, 255, 0)	3	13	156
4	hex	3	red	#FF0000	8	14	230
5	rgb	3	red	(255, 0, 0)	1	15	1000

The data values in “**value3**” column are grouped by intervals created by **cut**(same length) and **qcut** (same size)

cut & qcut

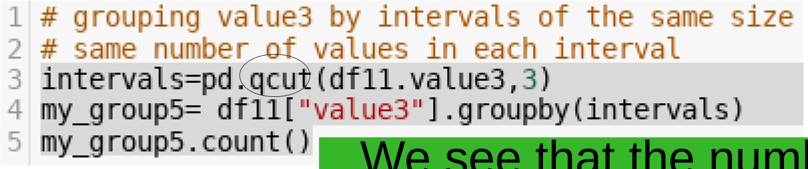


```
# grouping value3 by intervals of the same lenght
intervals=pd.cut(df11.value3,3)
my_group4= df11["value3"].groupby(intervals)
my_group4.count()
```

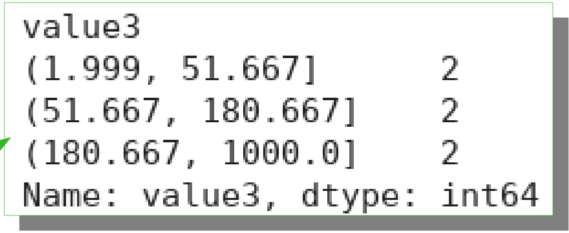


```
value3
(1.002, 334.667]      5
(334.667, 667.333]    0
(667.333, 1000.0]     1
Name: value3, dtype: int64
```

We see that the number values in each interval is different from the others



```
# grouping value3 by intervals of the same size
# same number of values in each interval
intervals=pd.qcut(df11.value3,3)
my_group5= df11["value3"].groupby(intervals)
my_group5.count()
```

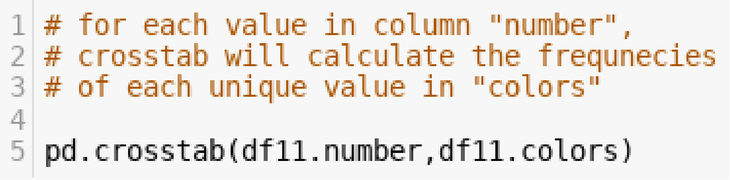


```
value3
(1.999, 51.667]       2
(51.667, 180.667]     2
(180.667, 1000.0]     2
Name: value3, dtype: int64
```

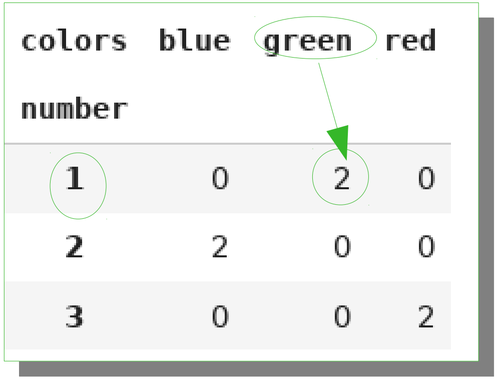
We see that the number values in each interval are all the same == **2**

[By Amina Delali]

crosstab



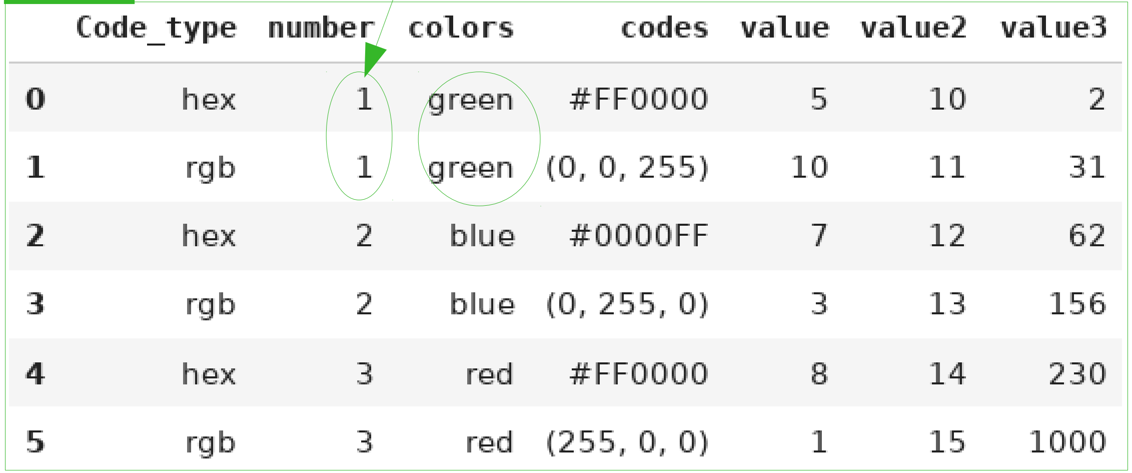
```
# for each value in column "number",
# crosstab will calculate the frequnecies
# of each unique value in "colors"

pd.crosstab(df11.number,df11.colors)
```



colors	blue	green	red
number			
1	0	2	0
2	2	0	0
3	0	0	2

For “number” value **==1,** corresponds: **2** values == **green** and **0** value in **blue** and **red** in “colors”

df11



	Code_type	number	colors	codes	value	value2	value3
0	hex	1	green	#FF0000	5	10	2
1	rgb	1	green	(0, 0, 255)	10	11	31
2	hex	2	blue	#0000FF	7	12	62
3	rgb	2	blue	(0, 255, 0)	3	13	156
4	hex	3	red	#FF0000	8	14	230
5	rgb	3	red	(255, 0, 0)	1	15	1000

[By Amina Delali]

References

- Datawatch. What is data wrangling? On-line at https://www.datawatch.com/what-is-data-wrangling/. Accessed on 31-10-2018.
- Wes McKinney. Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. O’Reilly Media, Inc, 2018.
- pydata.org. Pandas documentation. On-line at https://pandas.pydata. org/. Accessed on 19-10-2018.

school of ai

Thank you!

FOR ALL YOUR TIME