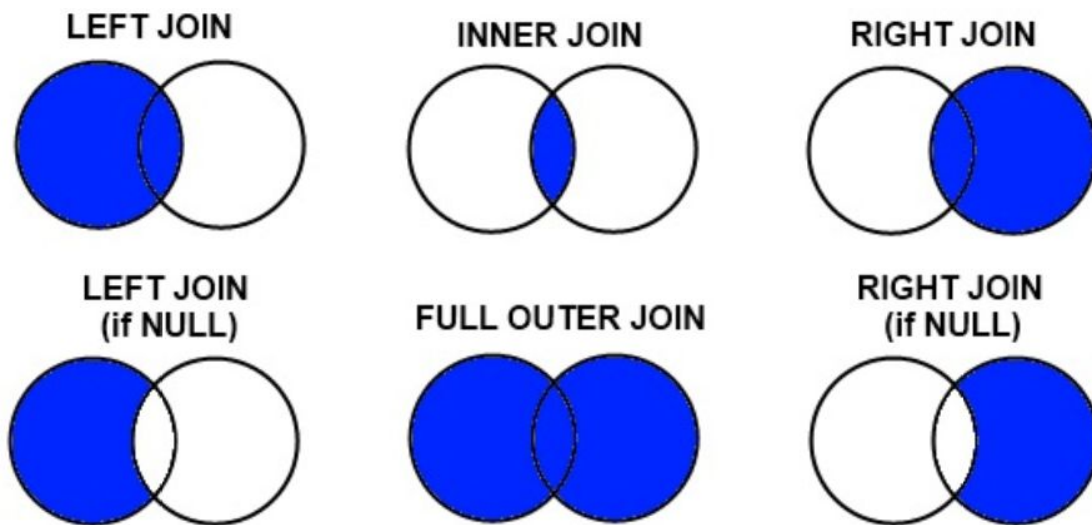


## T5 - Data analysis techniques and methodologies



- Merging on Dataframes Columns

We can merge Dataframes N:1 and N:N

**pandas.merge( <Dataframe\_1>, ..., <Dataframe\_n> )** -> Looks for strictly coincidences index and labels

**pandas.merge( <Dataframes>, on= <ColumnLabels> )** -> Label exact coincidence values, not indexes.

**pandas.merge( <dfs>, on = <CoLabels>, how= { 'inner', 'right', 'left', 'outer' } )** -> Order to merge:

inner: for labels, after indexes df1, ..., after indexes dfn. Default value for merge

right: for CoLabels, after labels dfn, ..., after labels df1. NaN not permitted on right dfs labels

left: for CoLabels, after labels df1, ..., after labels dfn. NaN not permitted on left dfs labels

outer: same as inned, but permits NaN for any non combination.

**pandas.merge( ....., suffixes=<suffix list for labels not in CoLabels> )**

- Merging on DataFrames Indexes

Merge index to index -> **left\_index = True, right\_index = True**

Merge label with index -> left\_on = < list of labels>, right\_index = True

| -> right\_on=< list of labels>, left\_index = True

- Joining Dataframes with same indexes

Joining dataframe to other dataframe: adding combinations and columns for items:

Example1: **df1.join(df2)**

df1	data	df2	profit	df1.j(df2)	data	profit
O	0	O	10	L	2	NaN
U	1	O	20	O	0	10.0
L	2	U	20	O	0	20.0
O	3			O	3	10.0
U	4			O	3	20.0
				U	1	20.0
				U	4	20.0

- Concatenation of Series

Concatenate/link them along specific access: **axis = 0** -> rows, **axis = 1** -> columns (generates Dataframe)

For new behaviour (not sort by default) on **axis = 1** -> **sort = False**

Label indexes by **names=**

Set subindexes by **keys=**

```
# Concatenate/link numpy arrays
print '--- Concatenate a1, b1 rows(axis=0)'
print np.concatenate([a1, b1], axis=0)

print '--- Concatenate a1, b1 columns(axis=1)'
print np.concatenate([a1, b1], axis=1)

# Create Series
s1 = Series([100, 200, 300], index=['A', 'B', 'C'])
s2 = Series([400, 500], index=['D', 'E'])
print '--- s1 ---'
print s1
print '--- s2 ---'
print s2

# Concatenate/link Series
print '--- concat(s1,s2) axis = 0---'
print pd.concat([s1, s2])
print '--- concat(s2, s1) axis = 0---'
print pd.concat([s2, s1])
print '--- series concat(s1,s2) axis = 0---'
s = pd.concat([s1, s2], axis=0,
               keys=['s1', 's2'],
               names=['idx_s', 'idx'])
print s

print '--- series concat(s1,s2) axis = 1---'
s = pd.concat([s1, s2], axis=1, sort=False,
               keys=['s1', 's2'],
               names=['idx'])
print s
```

```
--- concat(s1,s2) axis = 0---
A    100
B    200
C    300
D    400
E    500
dtype: int64

--- concat(s2,s1) axis = 0---
D    400
E    500
A    100
B    200
C    300
dtype: int64

--- series concat(s1,s2) axis = 0---
idx_s  idx
s1     A    100
      B    200
      C    300
s2     D    400
      E    500
dtype: int64

--- series concat(s1,s2) axis = 1---
idx    s1    s2
A      100.0  NaN
B      200.0  NaN
C      300.0  NaN
D      NaN   400.0
E      NaN   500.0
```

- Concatenation of dataframes

**pandas.concat([df1, df2, axis = 0/1, sort=False, ignore\_index=True)**

Same as pandas.Series, but for recreate continuous index use **ignore\_index=True**.

Cell values not assigned were filled with NaN.

```
print '--- concat(s1,s2) axis = 0---'
print pd.concat([s1, s2])
print '--- concat(s2, s1) axis = 0---'
print pd.concat([s2, s1])
print '--- series concat(s1,s2) axis = 0---'
s = pd.concat([s1, s2], axis=0,
               keys=['s1', 's2'],
               names=['idx_s', 'idx'])
print s

print '--- series concat(s1,s2) axis = 1---'
s = pd.concat([s1, s2], axis=1, sort=False,
               keys=['s1', 's2'],
               names=['idx'])
print s

# Create Dataframes
df1 = DataFrame(random.randn(4, 3),
                 columns=['A', 'B', 'C'])
df2 = DataFrame(random.randn(3, 3),
                 columns=['B', 'D', 'A'])

print '--- df1 ---'
print df1

print '--- df2 ---'
print df2

# Concatenate/link Dataframes
print '--- df concat(df1, df2) axis=0 ---'
print pd.concat([df1, df2], axis=0, sort=False)
print '--- df concat(df1, df2) axis=0 ignore_index ---'
print pd.concat([df1, df2], axis=0, sort=False,
                 ignore_index=True)
print '--- df concat(df1, df2) axis=1 ---'
print pd.concat([df1, df2], axis=1, sort=False)
```

```
--- df1 ---
   A      B      C
0  0.904843  0.013675 -1.758741
1  0.607929  0.753118 -1.182747
2 -0.120983  1.201189  0.981723
3  1.009936  0.680958 -1.351300

--- df2 ---
   B      D      A
0  0.124771 -1.821855 -1.395193
1 -1.247934  2.219757 -1.057643
2 -2.446749  0.557201  0.168668

--- df concat(df1, df2) axis=0 ---
   A      B      C      D
0  0.904843  0.013675 -1.758741  NaN
1  0.607929  0.753118 -1.182747  NaN
2 -0.120983  1.201189  0.981723  NaN
3  1.009936  0.680958 -1.351300  NaN
4 -1.395193  0.124771  NaN -1.821855
5 -1.057643 -1.247934  NaN  2.219757
6  0.168668 -2.446749  NaN  0.557201

--- df concat(df1, df2) axis=1 ---
   A      B      C      B      D      A
0  0.904843  0.013675 -1.758741  0.124771 -1.821855 -1.395193
1  0.607929  0.753118 -1.182747 -1.247934  2.219757 -1.057643
2 -0.120983  1.201189  0.981723 -2.446749  0.557201  0.168668
3  1.009936  0.680958 -1.351300  NaN  NaN  NaN
```

- Combining Series and Dataframes

Example combining series if values are NaN:

```
from numpy import nan, float64, arange, where
from pandas import Series, DataFrame, isnull

s1=Series([5, nan, 6, nan], index=['A', 'B', 'C', 'D'])
print '--- s1 ---'
print s1

s2=Series(arange(4), dtype=float64, index=s1.index)
print '--- s2 ---'
print s2

# Substitution values for NaN with where
# isnull() selects choice s1 or s2 values
s3 = Series(where(isnull(s1), s2, s1), index=s1.index)
print '- combine s2 values if s1 value is NaN -'
print s3

# The same with combine_first function
s4 = s1.combine_first(s2)
print '---- Same with combine_first method ---'
print s4
```

```
--- s1 ---
A    5.0
B    NaN
C    6.0
D    NaN
dtype: float64
--- s2 ---
A    0.0
B    1.0
C    2.0
D    3.0
dtype: float64
- combine s2 values if s1 value is NaN -
A    5.0
B    1.0
C    6.0
D    3.0
dtype: float64
---- Same with combine_first method ---
A    5.0
B    1.0
C    6.0
D    3.0
dtype: float64
```

Example combining Dataframes, same method **combine\_first**:

```
df1 = DataFrame({'col1': [5, nan, 15],
                 'col2': [20, 25, nan],
                 'col3': [nan, nan, 35]})
print '--- df1 ---'
print df1

df2 = DataFrame({'col1': [0, 10, 15],
                 'col3': [10, 20, 30]})
print '--- df2 ---'
print df2

# Combine Dataframes
print '- df1 combine_first df2'
print df1.combine_first(df2)
```

```
--- df1 ---
   col1  col2  col3
0     5   20.0   NaN
1    NaN   25.0   NaN
2    15.0   NaN  35.0
--- df2 ---
   col1  col3
0     0    10
1    10    20
2    15    30
- df1 combine_first df2
   col1  col2  col3
0     5   20.0  10.0
1    10.0  25.0  20.0
2    15.0   NaN  35.0
```

- Stacking/unstacking  
to/from series

Example:

- how to stack Dataframe to Serie.
- unstack from several Serie indexes.

```
--- df1 ---
attributes  c1  c2  c3  c4
cabs
Uber        0   1   2   3
Grab        4   5   6   7
Index([u'Uber', u'Grab'], dtype='object', name=u'cabs')
```

```
from numpy import arange
from pandas import Series, DataFrame, Index

# Create DataFrame
df1 = DataFrame(arange(8).reshape(2, 4),
                index=Index(['Uber', 'Grab'],
                             name='cabs'),
                columns=Index(['c1', 'c2', 'c3', 'c4'],
                              name='attributes'))

print '--- df1 ---'
print df1
print df1.index

# Stack to series df1 for attributes
df2 = df1.stack('attributes')
print '- stacked df1 for attributes -'
print df2
print df2.index

# Unstack from serie to DataFrame
undf2 = df2.unstack('attributes')
print '--- unstacked df2 for attributes---'
print undf2
print undf2.index

undf3 = df2.unstack('cabs')
print '--- unstacked df2 for cabs---'
print undf3
print undf3.index
```

```
- stacked df1 for attributes -
cabs  attributes
Uber  c1          0
      c2          1
      c3          2
      c4          3
Grab  c1          4
      c2          5
      c3          6
      c4          7
dtype: int64
MultiIndex(levels=[[u'Uber', u'Grab'], [u'c1', u'c2', u'c3', u'c4']],
            codes=[[0, 0, 0, 0, 1, 1, 1, 1], [0, 1, 2, 3, 0, 1, 2, 3]],
            names=[u'cabs', u'attributes'])
--- unstacked df2 for attributes---
attributes  c1  c2  c3  c4
cabs
Uber        0   1   2   3
Grab        4   5   6   7
Index([u'Uber', u'Grab'], dtype='object', name=u'cabs')
--- unstacked df2 for cabs---
cabs  attributes
Uber  c1          0   4
      c2          1   5
      c3          2   6
      c4          3   7
Index([u'c1', u'c2', u'c3', u'c4'], dtype='object', name=u'attributes')
```



Other example from series to DataFrame with dropna utility to fill NaN values instead drop.

```
print undf3.index

# Unstack from series to DataFrame
s1 = Series([5, 10, 15], index=['c1', 'c2', 'c3'])
print '--- s1 ---'
print s1
s2 = Series([15, 20, 25], index=['c2', 'c3', 'c4'])
print '--- s2 ---'
print s2

s3 = concat([s1, s2], keys=['s1', 's2'])
print '--- concat s1,s2 ---'
print s3
print s3.index

df = s3.unstack(0)
print '- unstack from Serie to DataFrame -'
print '- First index=0 as columns'
print df
print df.index

df = s3.unstack(1)
print '- unstack from Serie to DataFrame -'
print '- Second index=1 as columns'
print df
print df.index

print df.stack(dropna=False)
```

```
--- concat s1,s2 ---
s1  c1    5
    c2   10
    c3   15
s2  c2   15
    c3   20
    c4   25
dtype: int64
MultiIndex(levels=[[u's1', u's2'], [u'c1', u'c2', u'c3', u'c4']],
            codes=[[0, 0, 0, 1, 1, 1], [0, 1, 2, 1, 2, 3]])
- unstack from Serie to DataFrame -
- First index=0 as columns
      s1    s2
c1    5.0  NaN
c2   10.0  15.0
c3   15.0  20.0
c4    NaN  25.0
Index([u'c1', u'c2', u'c3', u'c4'], dtype='object')
- unstack from Serie to DataFrame -
- Second index=1 as columns
      c1    c2    c3    c4
s1    5.0  10.0  15.0   NaN
s2    NaN  15.0  20.0  25.0
Index([u's1', u's2'], dtype='object')
s1  c1    5.0
    c2   10.0
    c3   15.0
    c4    NaN
s2  c1    NaN
    c2   15.0
    c3   20.0
    c4   25.0
```

- Pivot Tables

Were a resume table for near value data or equal. New table is an aggregation table for this values: summarize, average, count, std. deviation, ....

Example1: `df.pivot('year', 'month', 'passengers')` en flights.csv

Example2: `df.pivot_table(index='continent', columns='year', values='lifeExp', aggfunc='mean')`

It keeps simple: Download csv as dataframe, and make pivot table, after render this data with seaborn.

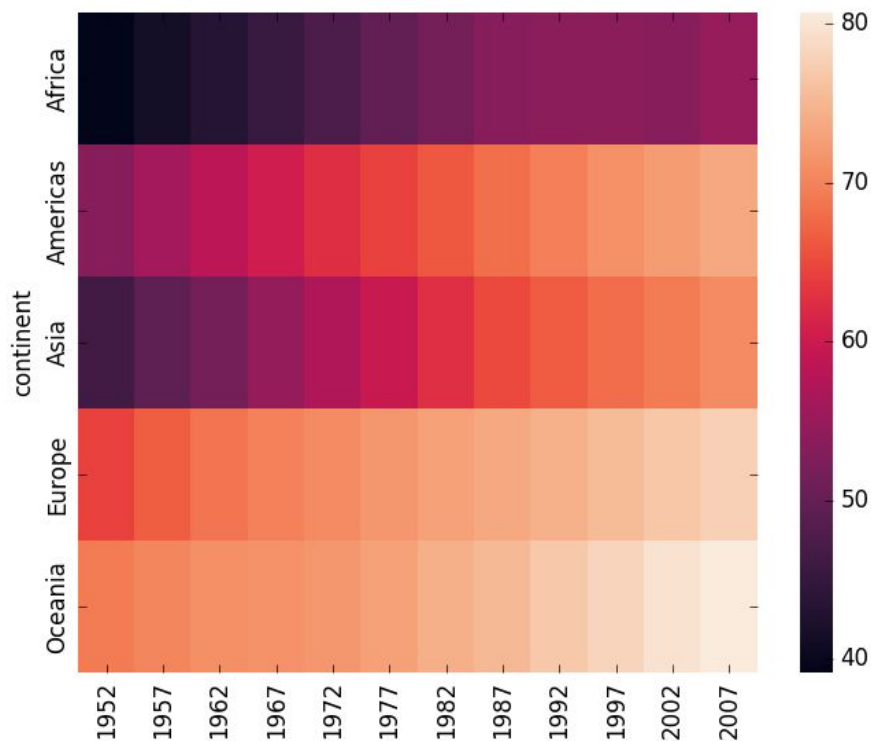
```

1 import pandas as pd
2 import seaborn as sns
3 # Set link to download csv
4 csv = 'https://raw.githubusercontent.com/resbaz/' \
5       'r-novice-gapminder-files/master/data/' \
6       'gapminder-FiveYearData.csv'
7 # Load Dataframe from csv that's in link
8 df = pd.read_csv(csv)
9 # Print Dataframe for test
10 print '- Printing Dataframe extracted from link -'
11 print df
12 print '- Making a pivot table, aggregation = average -'
13 dfpv = df.pivot_table(index='continent',
14                        columns='year',
15                        values='lifeExp',
16                        aggfunc='mean')
17 # Print Pivot Table for test
18 print '- Printing pivot table for test -'
19 print dfpv
20 # Generating Heat Map
21 print '- Generating heat map -'
22 sns.heatmap(dfpv).get_figure().savefig('assign3.png')

```

continent	1952	1957	1962	1967	1972	1977	1982	1987	1992	1997	2002	2007
Africa	39.135500	41.266346	43.319442	44.319442	45.319442	46.319442	47.319442	48.319442	49.319442	50.319442	51.319442	52.319442
Americas	53.279840	55.960280	58.398760	60.398760	62.398760	64.398760	66.398760	68.398760	70.398760	72.398760	74.398760	76.398760
Asia	46.314394	49.318544	51.563223	53.563223	55.563223	57.563223	59.563223	61.563223	63.563223	65.563223	67.563223	69.563223
Europe	64.408500	66.703067	68.539233	70.539233	72.539233	74.539233	76.539233	78.539233	80.539233	82.539233	84.539233	86.539233
Oceania	69.255000	70.295000	71.085000	72.085000	73.085000	74.085000	75.085000	76.085000	77.085000	78.085000	79.085000	80.085000

Graphical representation of pivot table as heatmap:



- Dealing with duplicates on Python DataFrame

**df.duplicated(<columns>)** -> How to find indexes for True duplicated columns on rows.

**df.drop\_duplicates(<columns>, keep=...)** -> remove duplicates same columns values, keep= 'first' or 'last'.

```
from pandas import DataFrame

df = DataFrame({'col1': ['uber', 'uber', 'grab', 'grab'],
               'col2': [5, 4, 4, 2]})

print df

print '- Duplicated for all column coincidence -'
print df.duplicated()
print '- Dropped duplicates (nothing), all column -'
print df.drop_duplicates()

print '- Dropped duplicates, col1 -'
print df.drop_duplicates(['col1'])

print '- Duplicated for col2 coincidence -'
print df.duplicated(['col2'])
print '- Dropped duplicates keeping first, col2 -'
print df.drop_duplicates(['col2'], keep='first')
print '- Dropped duplicates keeping last, col2 -'
print df.drop_duplicates(['col2'], keep='last')
```

- Dropped duplicates (nothing), all column -	
col1	col2
0	uber 5
1	uber 4
2	grab 4
3	grab 2

- Dropped duplicates, col1 -	
col1	col2
0	uber 5
2	grab 4

- Duplicated for col2 coincidence -	
0	False
1	False
2	True
3	False

dtype: bool

- Dropped duplicates keeping first, col2 -	
col1	col2
0	uber 5
1	uber 4
3	grab 2

- Dropped duplicates keeping last, col2 -	
col1	col2
0	uber 5
2	grab 4
3	grab 2

- Mapping in a DataFrame

Previous we have a new dictionary object to complete one column of dataframe with a column index.

```
from pandas import DataFrame

df = DataFrame({'city': ['Tarragona', 'Barcelona', 'Lleida', 'Girona'],
               'pref': ['977', '93', '973', '972']})

print df

Habs = {'Tarragona': 0.8,
        'Barcelona': 5.6,
        'Lleida': 0.4,
        'Girona': 0.7}

df['Mhabs'] = df['city'].map(Habs)

print df
```

	city	pref
0	Tarragona	977
1	Barcelona	93
2	Lleida	973
3	Girona	972

	city	pref	Mhabs
0	Tarragona	977	0.8
1	Barcelona	93	5.6
2	Lleida	973	0.4
3	Girona	972	0.7

Map functions are a powerful tool to main default element wise transformations.



- Replacing values on Series

We can replace values with other values, list of values with other values, replace through dictionary.

```
from numpy import nan
from pandas import Series

s1 = Series([10, 20, 40, 50, 20, 10, 50, 40])

print s1.values
print '---'
print s1.replace(50, nan).values
print '---'
print s1.replace([10, 20, 50], [100, 200, 500]).values
print '---'
print s1.replace({10: 100, 20: nan, 40: 400}).values
```

Python Console

```
>>> runfile('/home/r5sim/PycharmProjects/Example/4_Methods/replceSrVal.py',
wdir='/home/r5sim/PycharmProjects/Example')
[10 20 40 50 20 10 50 40]
---
[10. 20. 40. nan 20. 10. nan 40.]
---
[100 200 40 500 200 100 500 40]
---
[100. nan 400. 50. nan 100. 50. 400.]
>>>
```

- Rename Indexes on Series, Dataframes

How we can change the values of indexes, via rename method, mapping functions, or using dictionaries

```
from numpy import arange
from pandas import DataFrame

df = DataFrame(
    arange(25).reshape(5, 5),
    index=['UBER', 'OLA', 'GRAB', 'GOJEK', 'LYFT'],
    columns=['RE', 'LO', 'QU', 'GR', 'AG'])
# RE is Revenue    LO is Loss    QU is quality
# GR is genre      AG is age

print df
# Change index values with updating df in place
print '- Change idx updating df in place -'
df.rename(index=str.title, columns=str.lower,
          inplace=True)

print df
# change indexes using mapping
print '- Change indexes using mapping -'
df.index = df.index.map(str.lower)
df.columns = df.columns.map(str.upper)
# df.index!= columns(str.upper)
print df
# Renaming with dictionary
print '- Renaming with dictionary -'
df.rename(index={'uber': 'U', 'ola': 'O', 'grab': 'G',
                'gojek': 'J', 'lyft': 'L'},
          columns={'RE': 'Revenue'}, inplace=True)
print df
```

	RE	LO	QU	GR	AG
UBER	0	1	2	3	4
OLA	5	6	7	8	9
GRAB	10	11	12	13	14
GOJEK	15	16	17	18	19
LYFT	20	21	22	23	24

- Change idx updating df in place -

	re	lo	qu	gr	ag
Uber	0	1	2	3	4
Ola	5	6	7	8	9
Grab	10	11	12	13	14
Gojek	15	16	17	18	19
Lyft	20	21	22	23	24

- Change indexes using mapping -

	RE	LO	QU	GR	AG
uber	0	1	2	3	4
ola	5	6	7	8	9
grab	10	11	12	13	14
gojek	15	16	17	18	19
lyft	20	21	22	23	24

- Renaming with dictionary -

	Revenue	LO	QU	GR	AG
U	0	1	2	3	4
O	5	6	7	8	9
G	10	11	12	13	14
gojek	15	16	17	18	19
L	20	21	22	23	24

