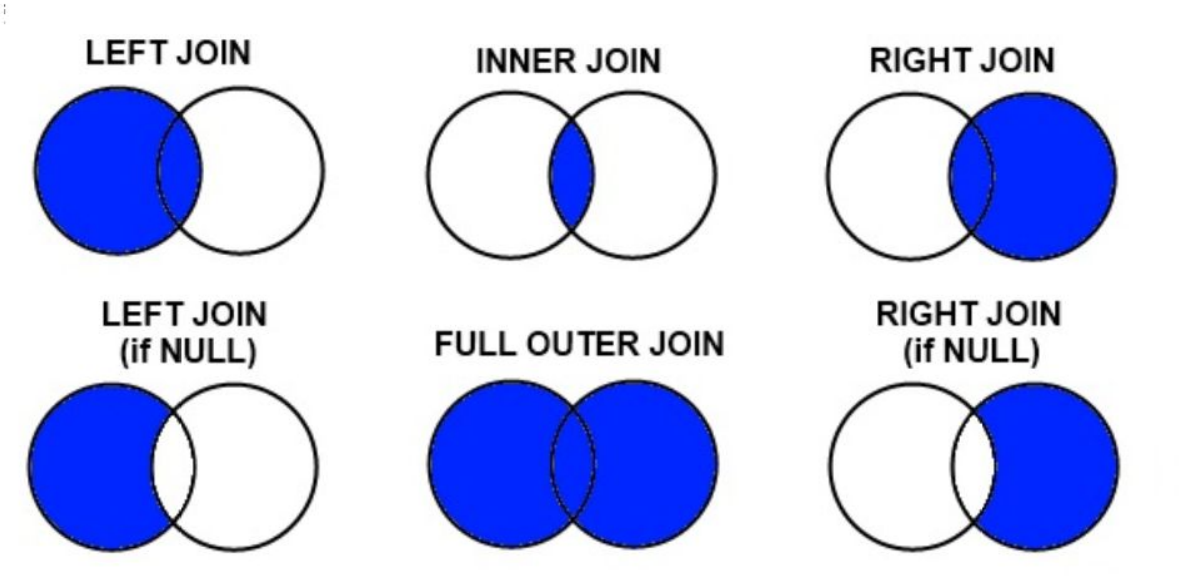
T5 - Data analysis techniques and methodologies



- ## 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)**

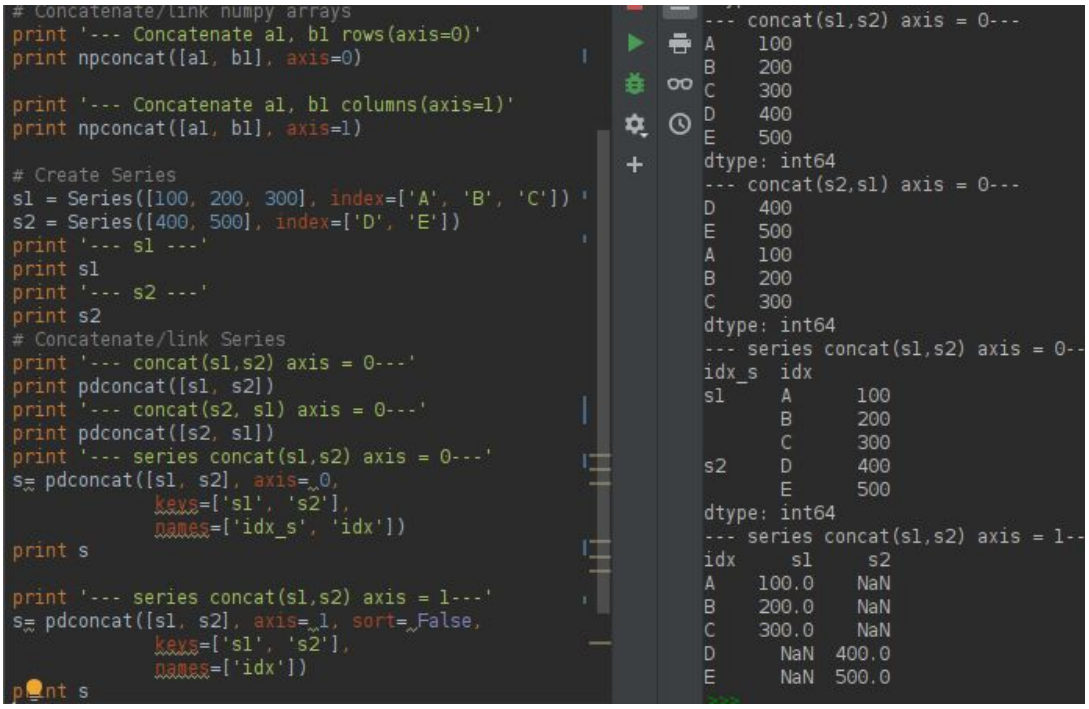
df1	**data**
O	0
U	1
L	2
O	3
U	4

df2	**profit**
O	10
O	20
U	20

df1.j(df2)	**data**	**profit**
L	2	NaN
O	0	10.0
O	0	20.0
O	3	10.0
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=**

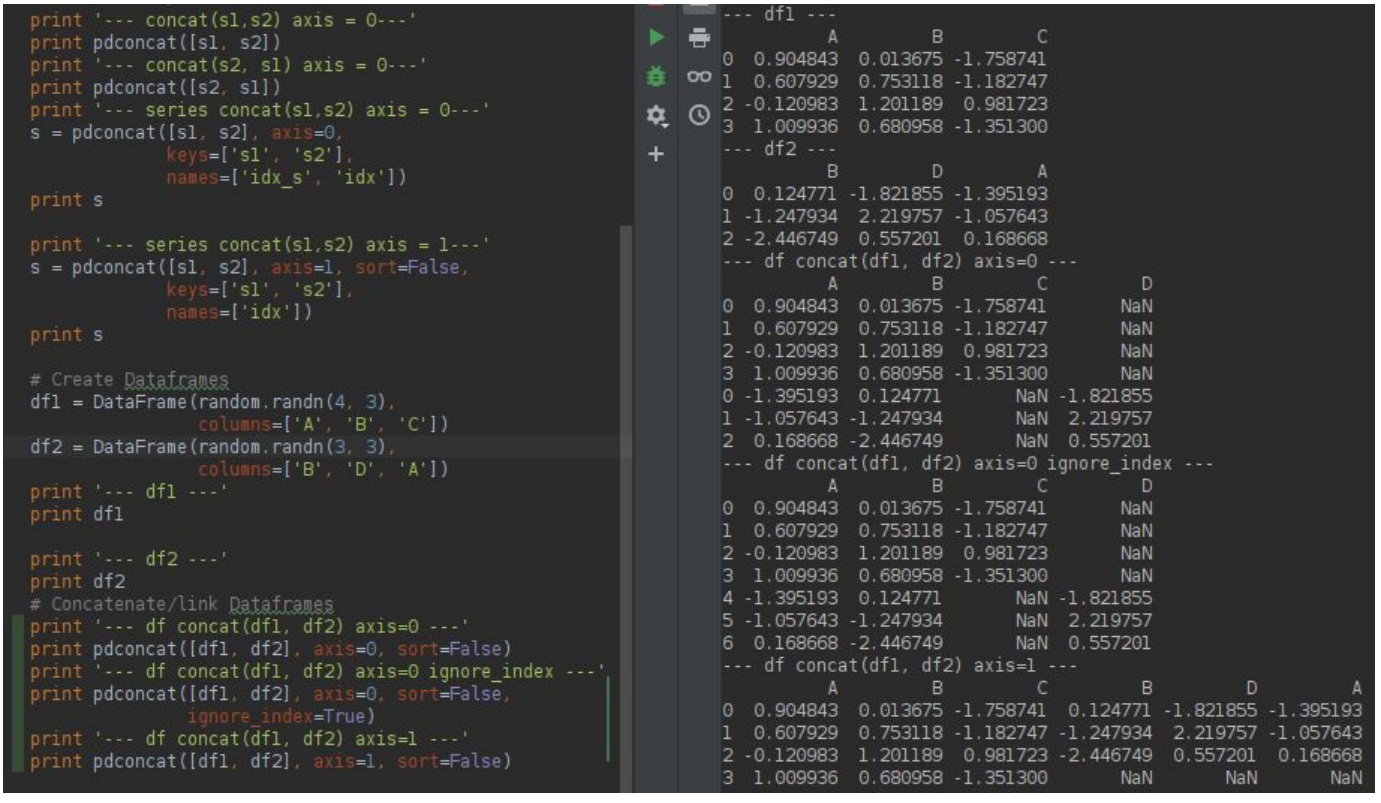


```
# Concatenate/link numpy arrays
print '--- Concatenate a1, b1 rows(axis=0)'
print npconcat([a1, b1], axis=0)

print '--- Concatenate a1, b1 columns(axis=1)'
print npconcat([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 pdconcat([s1, s2])
print '--- concat(s2, s1) axis = 0---'
print pdconcat([s2, s1])
print '--- series concat(s1,s2) axis = 0---'
s= pdconcat([s1, s2], axis=0,
            keys=['s1', 's2'],
            names=['idx_s', 'idx'])
print s

print '--- series concat(s1,s2) axis = 1---'
s= pdconcat([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 pdconcat([s1, s2])
print '--- concat(s2, s1) axis = 0---'
print pdconcat([s2, s1])
print '--- series concat(s1,s2) axis = 0---'
s = pdconcat([s1, s2], axis=0,
             keys=['s1', 's2'],
             names=['idx_s', 'idx'])
print s

print '--- series concat(s1,s2) axis = 1---'
s = pdconcat([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 pdconcat([df1, df2], axis=0, sort=False)
print '--- df concat(df1, df2) axis=0 ignore_index ---'
print pdconcat([df1, df2], axis=0, sort=False,
               ignore_index=True)
print '--- df concat(df1, df2) axis=1 ---'
print pdconcat([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
0 -1.395193  0.124771       NaN -1.821855
1 -1.057643 -1.247934       NaN  2.219757
2  0.168668 -2.446749       NaN  0.557201
--- df concat(df1, df2) axis=0 ignore_index ---
          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   5.0  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   5.0  20.0  10.0
1  10.0  25.0  20.0
2  15.0   NaN  35.0
```

- Stacking/unstacking to/from series

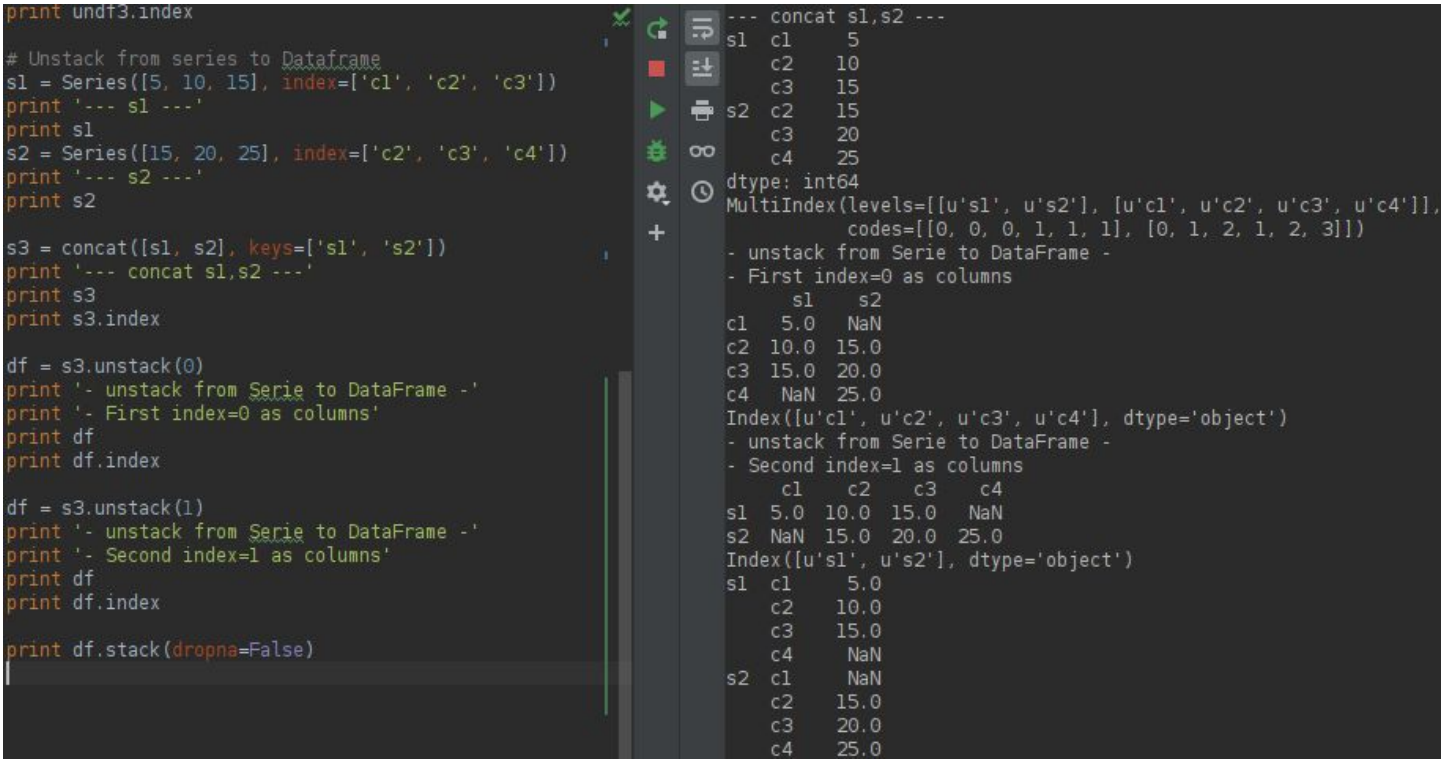
```
--- df1 ---
attributes  c1  c2  c3  c4
cabs
Uber         0   1   2   3
Grab         4   5   6   7
Index([u'Uber', u'Grab'], dtype='object', name=u'cabs')
```

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

```
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
# Unstac 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        Uber  Grab
attributes
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 agregation table for this values: summarize, average, count, std. deviation, ….

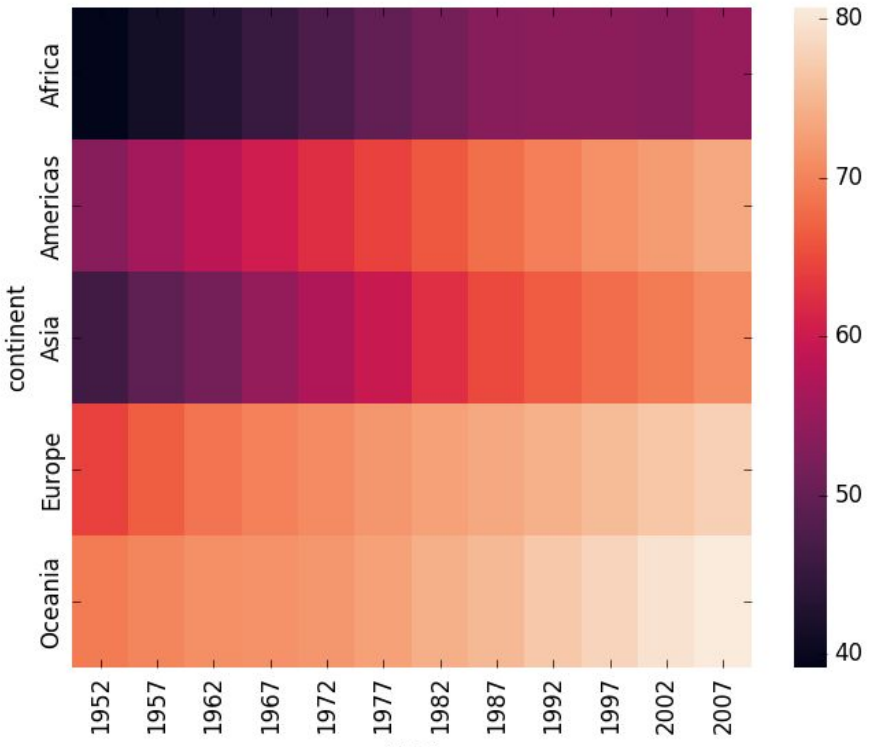
Example1: **df.pivot('year', 'month', 'passengers')** en flights.csv

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.

```
assignm3.py
import pandas as pd
import seaborn as sns
# Set link to download csv
csv = 'https://raw.githubusercontent.com/resbaz/' \
      'r-novice-gapminder-files/master/data/' \
      'gapminder-FiveYearData.csv'
# Load Dataframe from csv that's in link
df = pd.read_csv(csv)
# Print Dataframe for test
print '- Printing Dataframe extracted from link -'
print df
print '- Making a pivot table, aggregation = average -'
dfpv = df.pivot_table(index='continent',
                      columns='year',
                      values='lifeExp',
                      aggfunc='mean')
# Print Pivot Table for test
print '- Printing pivot table for test -'
print dfpv
# Generating Heat Map
print '- Generating heat map -'
sns.heatmap(dfpv).get_figure().savefig('assign3.png')
```

```
assignm3(26)  assignm3(27)  assignm3(28)  assignm3(29)
1686      Zambia  1982   6100407.0    Africa   51.821  1408.678565
1687      Zambia  1987   7272406.0    Africa   50.821  1213.315116
1688      Zambia  1992   8381163.0    Africa   46.100  1210.884633
1689      Zambia  1997   9417789.0    Africa   40.238  1071.353818
1690      Zambia  2002  10595811.0    Africa   39.193  1071.613938
1691      Zambia  2007  11746035.0    Africa   42.384  1271.211593
1692    Zimbabwe  1952   3080907.0    Africa   48.451   406.884115
1693    Zimbabwe  1957   3646340.0    Africa   50.469   518.764268
1694    Zimbabwe  1962   4277736.0    Africa   52.358   527.272182
1695    Zimbabwe  1967   4995432.0    Africa   53.995   569.795071
1696    Zimbabwe  1972   5861135.0    Africa   55.635   799.362176
1697    Zimbabwe  1977   6642107.0    Africa   57.674   685.587682
1698    Zimbabwe  1982   7636524.0    Africa   60.363   788.855041
1699    Zimbabwe  1987   9216418.0    Africa   62.351   706.157306
1700    Zimbabwe  1992  10704340.0    Africa   60.377   693.420786
1701    Zimbabwe  1997  11404948.0    Africa   46.809   792.449960
1702    Zimbabwe  2002  11926563.0    Africa   39.989   672.038623
1703    Zimbabwe  2007  12311143.0    Africa   43.487   469.709298

[1704 rows x 6 columns]
- Making a pivot table, aggregation = average -
- Printing pivot table for test -
year            1952       1957       1962  ...       1997       2002       2007
continent                                    ...
Africa     39.135500  41.266346  43.319442  ...  53.598269  53.325231  54.806038
Americas   53.279840  55.960280  58.398760  ...  71.150480  72.422040  73.608120
Asia       46.314394  49.318544  51.563223  ...  68.020515  69.233879  70.728485
Europe     64.408500  66.703067  68.539233  ...  75.505167  76.700600  77.648600
Oceania    69.255000  70.295000  71.085000  ...  78.190000  79.740000  80.719500

[5 rows x 12 columns]
- Generating heat map -
>>>
```

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.

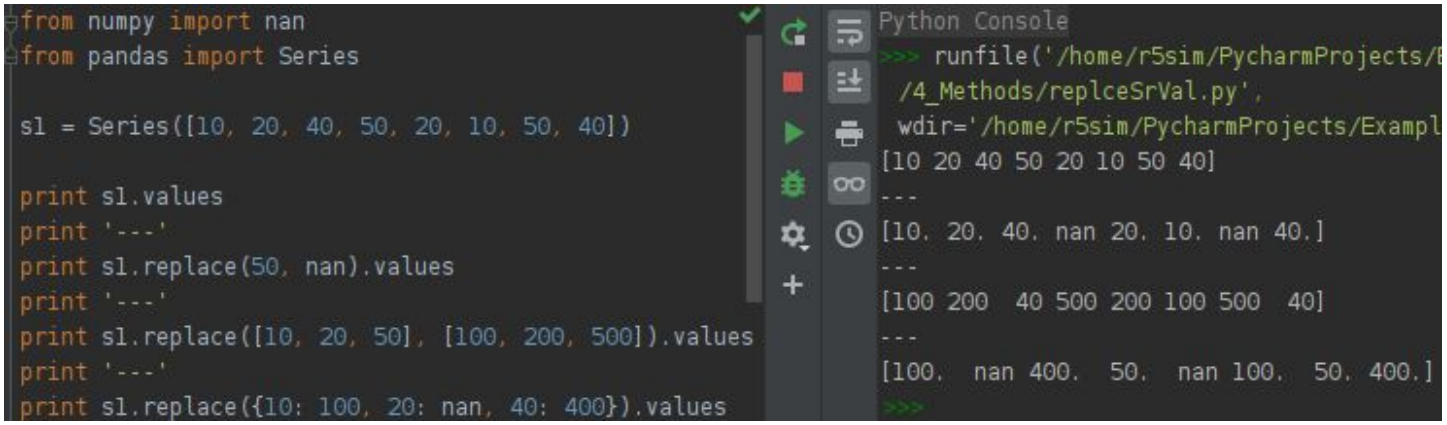
```
from pandas import DataFrame

df = DataFrame({'city': ['Tarragona', 'Barcelona', 'Lle
                '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
```

```
>>> runfile('/home/r5sim/
 /4_Methods/MapDf.py',
wdir='/home/r5sim/Pychar
       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.

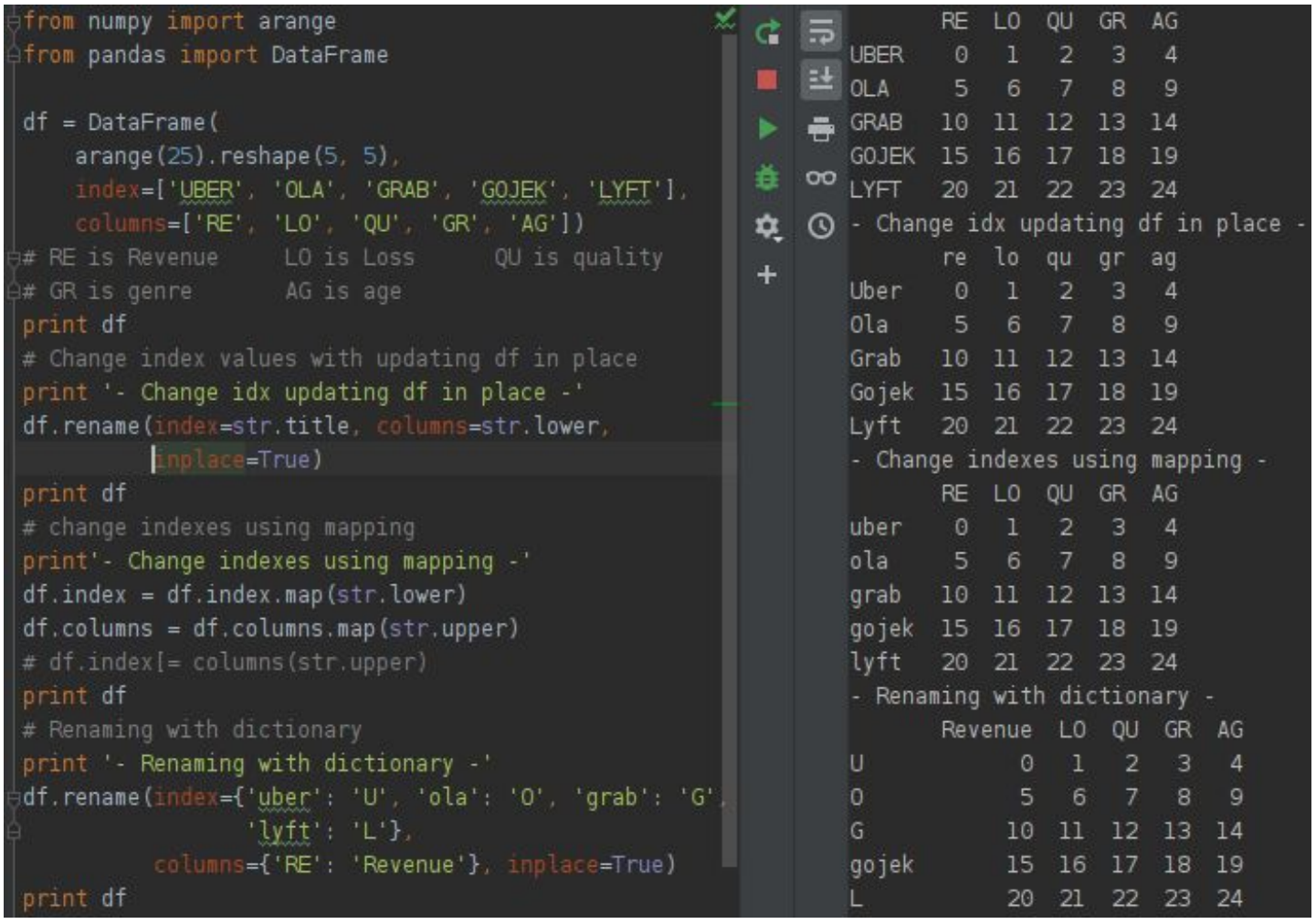


```
from numpy import nan
from pandas import Series

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/E
 /4_Methods/replceSrVal.py',
 wdir='/home/r5sim/PycharmProjects/Exampl
[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',
                 '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
```

- Binning Values

Separating elements into bin sort categories. How to create bins, add elements to bins, categorize elements into the frame bins.

```
from pandas import cut, value_counts

primnums = [2, 3, 5, 7, 11, 13, 17, 19,
            23, 29, 31, 37, 41, 43, 47]
numbins = [0, 10, 20, 30, 40, 50]
catg = cut(primnums, numbins)
print catg
print '---'
print value_counts(catg)
print '- auto categorize -'
# precision store/display
catg = cut(primnums, 3, precision=1)
print value_counts(catg)
```

```
 /Example1/4_Methods')
[(0, 10], (0, 10], (0, 10], (0, 10],
 (10, 20], ..., (30, 40], (30, 40],
 (40, 50], (40, 50], (40, 50]]
Length: 15
Categories (5, interval[int64]): [(0,
 10] < (10, 20] < (20, 30] < (30, 40]
  < (40, 50]]
---
(10, 20]    4
(0, 10]     4
(40, 50]    3
(30, 40]    2
(20, 30]    2
dtype: int64
- auto categorize -
(2.0, 17.0]     7
(32.0, 47.0]    4
(17.0, 32.0]    4
dtype: int64
```

- Observation, filtering and Basic Analysis on Dataframes.

df.head() -> First (five) elements of df dataframe
df.tail() -> Last (five) elements of df dataframe.
df.describe()-> each column dataframe: count, mean, std, min, max, 25%, 50%, 75% values dataframe.
column = df [0] Series as columns of dataframe: df[0],... you can use previous methods.
You can filter column with conditions.eg: **column[np.abs(column)>1]**
Same for dataframe **df.column[(np.abs(df)>3).any(1)]** where any(1) is one of more coordinate coincidence. Replace values for condition on dataframe: **df[(np.abs(df>3)] = np.sign(df)*<valor>**

```
from numpy import random, abs
from pandas import DataFrame

df = DataFrame(random.randn(300, 6))
print '- Head -'
print df.head()
print '- tail -'
print df.tail()
print '- description -'
print df.describe()

print 'condition on dataframe'
print df[(abs(df) > 3).any(1)]
```

```
- Head -
          0         1         2         3         4         5
0  0.395326  0.350652  0.486284 -0.200360  0.748497  1.129702
1 -0.726854  0.639944 -1.149157 -0.891027 -0.714333  1.596035
2 -0.178330  0.207914  0.791249  1.376595 -1.755830 -1.203099
3 -0.535323 -1.722779 -1.249950 -0.051593  0.588967  0.100919
4  0.700991 -0.016208 -0.799536 -1.345773  0.688871  1.387114
- tail -
           0         1         2         3         4         5
295 -0.123751  0.187325  0.134778 -0.091010  0.645569 -1.053574
296  1.431165 -0.883932 -0.347978  0.684907 -0.506576  1.700571
297 -0.536441 -0.408682  1.260484 -1.329479  1.385930  0.119962
298 -0.813200 -0.368584  1.179449 -1.454540  0.944520 -1.430976
299  1.973873  0.550871  0.059220  1.230382  0.158260  0.998530
- description -
                0           1           2           3           4           5
count  300.000000  300.000000  300.000000  300.000000  300.000000  300.000000
mean     0.013811   -0.128243    0.072664   -0.035737   -0.057174    0.081647
std      0.963472    1.041952    0.993132    1.108172    1.016685    0.982238
min     -3.286583   -3.391898   -2.691259   -3.205952   -3.017711   -2.816689
25%     -0.599171   -0.725603   -0.620242   -0.794939   -0.742209   -0.514275
50%      0.061233   -0.111774    0.123687    0.016023    0.024312    0.093642
75%      0.615239    0.584670    0.699852    0.715435    0.647024    0.675512
max      2.417312    3.183064    3.362810    3.323146    2.671871    2.835463
condition on dataframe
           0         1         2         3         4         5
16  -0.837905  0.071305  3.362810  0.100070  1.793270  0.679372
23   0.209472  0.984246 -0.480356 -3.205952 -1.886118  0.103550
68   0.094521 -3.391898  1.335944 -1.117206 -0.637098  0.060504
```

Substitute value:

```
from numpy import random, abs, sign
from pandas import DataFrame

df = DataFrame(random.randn(300, 6))
# print '- Head -'
# print df.head()
# print '- tail -'
# print df.tail()
print '- description -'
print df.describe()

# One of more coord. with value > 3
print '- Condition on DataFrame -'
print df[(abs(df) > 3).any(1)]

# Substitution values with condition
print '- Substitute for sign*5 -'
df[abs(df) > 3] = sign(df)*5
print df.describe()
```

```
                0           1           2           3           4           5
count  300.000000  300.000000  300.000000  300.000000  300.000000  300.000000
mean     0.026879    0.062116    0.025306    0.055370   -0.033593   -0.045183
std      1.025849    0.998384    0.914618    0.980184    0.951583    1.060344
min     -3.179544   -2.941613   -2.298552   -2.743556   -2.985757   -3.098537
25%     -0.667897   -0.623047   -0.592780   -0.610851   -0.694270   -0.727860
50%      0.041532    0.096691    0.022237    0.022262    0.001365   -0.048152
75%      0.712260    0.715488    0.580947    0.642533    0.540159    0.722797
max      3.821092    2.740674    3.052471    3.138921    3.521441    2.655050
- Condition on DataFrame -
           0         1         2         3         4         5
39   0.539435 -0.047923  3.052471  1.006636 -2.301634 -1.955157
61   0.224267  0.542995  0.199061 -1.716833  3.521441  0.934967
76  -0.362065 -1.740101 -2.008180  0.308733  2.507323 -3.098537
84  -1.381330  1.008431  0.390338  3.138921 -0.140735 -0.063606
102  3.821092  1.723147 -0.106004 -0.800158  0.535789  2.209474
127 -3.179544  1.464841 -0.445884  0.390668 -0.516392  1.438888
- Substitute for sign*5 -
                0           1           2           3           4           5
count  300.000000  300.000000  300.000000  300.000000  300.000000  300.000000
mean     0.024740    0.062116    0.031798    0.061574   -0.028664   -0.051521
std      1.066329    0.998384    0.942658    1.005332    0.973630    1.084074
min     -5.000000   -2.941613   -2.298552   -2.743556   -2.985757   -5.000000
25%     -0.667897   -0.623047   -0.592780   -0.610851   -0.694270   -0.727860
50%      0.041532    0.096691    0.022237    0.022262    0.001365   -0.048152
75%      0.712260    0.715488    0.580947    0.642533    0.540159    0.722797
max      5.000000    2.740674    5.000000    5.000000    5.000000    2.655050
```