Software Development for Data Analysis

Using pandas library

- exploring data
- creation and operation of data tables:
    - selection
    - summary
    - filtering
    - data aggregation

Topics

1. What is pandas and what does it mean for data analysis?
2. Creation and operation of series and rectangular data tables
3. Selection and retrieval of data in pandas
4. Summary, filtering and aggregation of data in pandas
5. Case Study

1. What is a pandas?

- `pandas` aims to be the fundamental element for practical, real analysis of data in Python (https://pandas.pydata.org/)

- In addition, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any programming language

- Provides a class of `DataFrame` objects for manipulating data with integrated indexing

1. What is a pandas?

Short Timeline:

- 2008: development of the `pandas` package at AQR Capital Management (Applied Quantitative Research, a global investment management firm based in Greenwich, Connecticut, USA) begins
- 2009: `pandas` becomes open source
- 2012: the first edition of *Python for Data Analysis* is published
- 2015: pandas becomes a project sponsored by NumFOCUS

1. What is a pandas?

- Intelligent data alignment and integrated missing data management
- Flexible remodeling of data sets
- Partitioning large sets of data based on labels
- Columns can be inserted and deleted from data structures for resizing
- Aggregation or transformation of data
- Combining and merging data sets

1. What is a pandas?

- Time series functionality: data range generation and frequency conversion, moving window statistics, date change and delay, etc.
- Tools for reading and writing data between data structures in memory and different formats: CSV and text files, MS Excel, SQL DB, etc.
- Optimized for performance, with critical code sequences written in Cython or C.
- Python with the pandas package is used in a wide variety of academic and business fields, including Finance, Economics, Statistics, Web Analytics and more.

2. Creation and operation of rectangular data series and tables

`pandas` introduces two data structures with which the whole package operates:

- `Series`
- `DataFrame`

2. Creation and operation of rectangular data series and tables

`Series` is a one-dimensional array-like object that contains a sequence of values (of types similar to `numpy` types) and an associated array of data labels, called its index.

Row 0	Val (0)
Row 1	Val (1)
Row 2	Val (2)
Row 3	Val (3)

2. Creation and operation of rectangular data series and tables

General syntax:

`class pandas.Series(data=None, index=None, dtype=None, name=None, copy=False)`

`data` – is a type structure of `dict`, `numpy.ndarray` or `pandas.DataFrame`

`index` – row names

`dtype` – `str`, `numpy.dtype`, sau `ExtensionDtype`; optional, `None` – inferred

`name` – str, optional, the name to give to the `Series`

`copy` – indicates copying of input data

2. Creation and operation of rectangular data series and tables

```
import pandas as pd
from pandas import Series

serie_1 = pd.Series(('a', 'b', 'c’))
print(serie_1)
print(type(serie_1))

serie_2 = Series([-2, 1.0, 7.4])
print(serie_2)
print(type(serie_2))
```

```
<class 'pandas.core.series.Series'>
0    a
1    b
2    c
dtype: object
<class 'pandas.core.series.Series'>
0   -2.0
1    1.0
2    7.4
dtype: float64
```

2. Creation and operation of rectangular data series and tables

```
# generation of n random values
# in the range [a, b]
# employing the methond np.random.rand(n)
# which returns values in the range [0, 1]
# f(a, b, g(x)) = a - (b - a)g(x), g(x) in [0, 1]

def randomAB(a=None, b=None, n=None):
    return a + np.random.rand(n) * (b - a)

list_1 = randomAB(-5, 5, 7)
print(type(list_1))
serie_3 = pd.Series(list_1,
        index=('L'+str(i+1) for i in range(7)))
print(serie_3)
```

```
<class 'numpy.ndarray'>
L1    1.601716
L2    0.872316
L3    1.810886
L4    0.059670
L5    3.252750
L6   -3.831093
L7   -2.937451
dtype: float64
```

2. Creation and operation of rectangular data series and tables

Compared to `numpy.ndarrays`, index tags can be used when a single value or set of values is selected. Here [‘a’, ‘c’, ‘d’] is interpreted as a list of indices, even if it contains strings instead of integers.

```
serie_4 =pd.Series(data=[-8, 'sir', True, 1.1],
                   index=('a', 'b', 'c', 'd'))
print(serie_4)
serie_4['d'] = -5.1
print(serie_4[['a', 'c', 'd']])
```

```
a      -8
b     sir
c    True
d     1.1
dtype: object
a      -8
c    True
d    -5.1
dtype: object
```

2. Creation and operation of rectangular data series and tables

Using `numpy` functions or similar `numpy` operations, such as boolean array filtering, scalar multiplication, or applying mathematical functions, will keep the link between index and value:

```
serie_5 = pd.Series(data=[-8, 7.0, 12, 1.1],
                    index=['a', 'b', 'c', 'd'])
print(serie_5[serie_5 > 0.0])
print(serie_5*2)
print(np.exp(serie_5))
```

```
b     7.0
c    12.0
d     1.1
dtype: float64
a   -16.0
b    14.0
c    24.0
d     2.2
dtype: float64
a         0.000335
b      1096.633158
c    162754.791419
d         3.004166
dtype: float64
```

2. Creation and operation of rectangular data series and tables

A `DataFrame` is a two-dimensional data structure that consists of an ordered collection of columns, each of which can have a different type of value (numeric, string, boolean, etc.). A `DataFrame` has both row and column indexes.

	Coloana 0	Coloana 1	Coloana 2
Randul 0	Val (0, 0)	Val (0, 1)	Val (0, 2)
Randul 1	Val (1, 0)	Val (1, 1)	Val (1, 2)
Randul 2	Val (2, 0)	Val (2, 1)	Val (2, 2)
Randul 3	Val (3, 0)	Val (2, 1)	Val (3, 2)

2. Creation and operation of rectangular data series and tables

General syntax:

class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)

data – is a type structure of `dict`, `numpy.ndarray` or `pandas.DataFrame`

index – row names

columns – column names

dtype – `str`, `numpy.dtype`, sau `ExtensionDtype`; optional, `None` - inferred

copy – indicates copying of input data; the parameter applies only to two-dimensional numpy.ndarray

2. Creation and operation of rectangular data series and tables

Possible parameters for the `DataFrame` constructor:

- 2D `ndarray` - a data array, optionally row and column labels
- `dict` of vectors, lists or tuples - each sequence becomes a column in the `DataFrame`; all sequences must be the same length
- `dict` of `Series` - each value becomes a column; the indexes in each series are grouped together to form the index of the result row, unless an explicit index is passed
- `dict` of `dict` - each inner `dict` becomes a column; the keys are joined to form the row index as in the `dict` of `Series`

2. Creation and operation of rectangular data series and tables

Parametri posibili pentru contructorul `DataFrame`:

- `list` of `dict` or `Series` - each element becomes a row in the `DataFrame`; the reunion of `dict` keys or `Series` indexes becomes the labels of the `DataFrame` columns
- `list` of `list` or `tuples` - treated as the 2D `ndarray` case
- another `DataFrame` - `DataFrame` indexes are used unless others are provided

2. Creation and operation of rectangular data series and tables

Possible parameters for the `DataFrame` constructor:

- 2D `ndarray` - a data array, optionally row and column labels

```
vector = randomAB(1, 3, 15)
nda_1 = np.ndarray(shape=(5, 3), buffer=vector,
                   dtype=float)
df_1 = pd.DataFrame(data=nda_1)
print(df_1)
```

	0	1	2
0	2.602743	1.728672	2.124315
1	1.052538	1.250193	2.028332
2	1.687388	2.590899	1.939280
3	2.033422	2.590036	2.227769
4	1.847186	2.173542	2.711257

2. Creation and operation of rectangular data series and tables

Possible parameters for the `DataFrame` constructor:

- `dict` of vectors, `lists` or `tuples` - each sequence becomes a column in the `DataFrame`; all sequences must be the same length

```
# DataFrame from dict of lists or tuples
dict_2 = {'Alina':[10.00, 9.50, 8.90, 9.40],
          'Anul':(2018, 2019, 2020, 2021),
          'Luna':('ianuarie', 'februarie',
                  'martie', 'aprilie')}
df_2 = pd.DataFrame(data=dict_2,
                    index=(1, 2, 3, 4))
print(df_2)
```

```
   Alina  Anul       Luna
1   10.0  2018   ianuarie
2    9.5  2019  februarie
3    8.9  2020     martie
4    9.4  2021    aprilie
```

2. Creation and operation of rectangular data series and tables

Possible parameters for the `DataFrame` constructor:

- `dict` of `Series` - each value becomes a column; the indexes in each series are grouped together to form the index of the result row, unless an explicit index is passed

```
# DataFrame from dict of Series, indexes are merge
s_1 = pd.Series(data=(1, 2, 3),
    index=['Lin1'+str(i+1) for i in range(3)])
s_2 = pd.Series(data=(4, 5, 6),
    index=['Lin2'+str(i+1) for i in range(3)])
dict_3 = {'Col1':s_1, 'Col2':s_2}
df_3 = pd.DataFrame(data=dict_3)
print(df_3)
```

```
       Col1  Col2
Lin11   1.0   NaN
Lin12   2.0   NaN
Lin13   3.0   NaN
Lin21   NaN   4.0
Lin22   NaN   5.0
Lin23   NaN   6.0
```

2. Creation and operation of rectangular data series and tables

Possible parameters for the `DataFrame` constructor:

- `dict` of `dict` - each inner dict becomes a column; the keys are joined to form the row index as in the `dict` of `Series`

```
d_1 = {'Maria':1, 'Ioana':2, 'Marin':3, 'Cornel':4}
d_2 = {'Maricica':1, 'Ion':2, 'Marina':3, 'Cornel':4}
d_3 = {'An 1':d_1, 'An 2':d_2}
df_4 = pd.DataFrame(data=d_3)
print(df_4)
```

```
   Maria  Ioana  Marin  Cornel  Maricica  Ion  Marina
0    1.0    2.0    3.0       4       NaN  NaN     NaN
1    NaN    NaN    NaN       4       1.0  2.0     3.0
```

2. Creation and operation of rectangular data series and tables

Possible parameters for the `DataFrame` constructor:

- `list` of `dict` - each element becomes a row in the `DataFrame`; the reunion of `dict` keys or `Series` indexes becomes the labels of the `DataFrame` columns

```
d_1 = {'Maria':1, 'Ioana':2, 'Marin':3, 'Cornel':4}
d_2 = {'Maricica':1, 'Ion':2, 'Marina':3, 'Cornel':4}
list_1 = [d_1, d_2]
df_5 = pd.DataFrame(data=list_1)
print(df_5)
```

	Maria	Ioana	Marin	Cornel	Maricica	Ion	Marina
0	1.0	2.0	3.0	4	NaN	NaN	NaN
1	NaN	NaN	NaN	4	1.0	2.0	3.0

2. Creation and operation of rectangular data series and tables

Possible parameters for the `DataFrame` constructor:

- `list` of `Series` – each item becomes a row in the `DataFrame`; the reunion of `dict` keys or `Series` indexes becomes the labels of the `DataFrame` columns

```
s_1 = pd.Series(data=(1, 2, 3), index=['Lin1'+str(i+1) for i in
range(3)])
s_2 = pd.Series(data=(4, 5, 6), index=['Lin2'+str(i+1) for i in
range(3)])
list_1 = [s_1, s_2]
df_6 = pd.DataFrame(data=list_1)
print(df_6)
```

	Lin11	Lin12	Lin13	Lin21	Lin22	Lin23
0	1.0	2.0	3.0	NaN	NaN	NaN
1	NaN	NaN	NaN	4.0	5.0	6.0

2. Creation and operation of rectangular data series and tables

Possible parameters for the `DataFrame` constructor:

- `list` of `lists` or `tuples` - treated as the 2D `ndarray` case

```
# list of list or tuple (equivalent of 2D ndarray)
list_2 = [(1, 2, 3),
          (4, 5, 6)]
df_7 = pd.DataFrame(data=list_2)
print(df_7)
```

```
   0  1  2
0  1  2  3
1  4  5  6
```

3. Selection and retrieval of data in pandas

Access to row and column names is through the `index` and `columns` atributes. These properties are of `pandas.Index` class. Items can be accessed through the `get_values()` method of the Index class. The numeric index of a value can be obtained by the `get_loc(value)` method, where value is the name of a row or column.

```
note = pd.DataFrame({"Data Structures":[5, 4, 6],
        ‘Data Analysis':[10, 6, 7],
        'Algebra':[7, 8, 7]},
    index=["Ionescu Dan", "Popescu Diana",
           'Georgescu Radu'])
print(note)
print(note.index, note.columns, sep='\n')
print(note.index[0], note.columns[1])
```

3. Selection and retrieval of data in pandas

Running the previous code sequence will generate the following result:

```
                Structuri de date  Analiza datelor  Algebra
Ionescu Dan                     5               10        7
Popescu Diana                   4                6        8
Georgescu Radu                  6                7        7
Index(['Ionescu Dan', 'Popescu Diana', 'Georgescu Radu'], dtype='object')
Index(['Structuri de date', 'Analiza datelor', 'Algebra'], dtype='object')
Ionescu Dan Analiza datelor
```

3. Selection and retrieval of data in pandas

- Selection of columns specifying column names in square brackets:

```
frame_name [['column1', ..., 'columnK’]]
```

- The selection of lines can be done by partitioning expressions:

```
index_begin : index_end
```

  where indexes are optional (if missing all lines are selected)

- The selection is made from `index_begin` to `index_end-1`

- Selection and partitioning can be done through the `loc` and `iloc` attributes, using the item name (for `loc`) or its index (for `iloc`).

3. Selection and retrieval of data in pandas

For example, in the case of the `DataFrame` created in the previous example, we can continue writing the following code sequence:

```
print(note.loc["Popescu Diana","Algebra"])
print(note.loc["Popescu Diana"]["Algebra"])
print(note.loc["Popescu Diana",:"Algebra"])
print(note.loc["Popescu Diana"][:"Algebra"])
print(note.loc["Popescu Diana":,:"Algebra"])
print(note.loc["Popescu Diana":][:"Algebra"])
print(note.iloc[1, 2])
print(note.iloc[1][2])
print(note.iloc[[1,2],2]) #print(note.iloc[[1,2]][2])  # wrong
print(note.iloc[1:, 2])   #print(note.iloc[1:][2])  # wrong
print(note.iloc[[1,0],[1,2]])
print(note.iloc[1][[1,2]])
print(note.iloc[1,1:])
print(note.iloc[1][1:])
print(note.iloc[1:,1:])
```

3. Selection and retrieval of data in pandas

Modifying the values in a column can be done by direct assignment:

```
frame_name ['column_name'] = value
```

When assigning a list or a mass, they must have a number of items equal to the number of items in the `DataFrame`.

Deleting a column is done by del command:

```
del frame_name ['column_name']
```

Inserting a column is done by the insert method:

```
DataFrame.insert (loc, column, value, allow_duplicates =
    False)
```

where `loc` is the insert index, `column` is the name of the column, and value is the value associated with the items for the inserted column.

3. Selection and retrieval of data in pandas

Another way to delete both columns and rows is by the drop method:

DataFrame.drop (self, labels = None, axis = 0, index = None, columns = None, level = None, inplace = False, errors = 'raise')

where:

labels - represent a list of row or column names;

axis - is 0 or 1 as `labels` is interpreted as row or column names;

columns - is a list of columns, and is an alternative to `labels` and `axis` = 1;

inplace - is a boolean that indicates whether the change is made by replacing the current object (`True`) or a modified copy is made and returned without changing the current object.

3. Selection and retrieval of data in pandas

Sorting tables can be done by the sort_values method:

> ***DataFrame.sort_values (by, axis = 0, ascending = True, inplace = False, kind = 'quicksort', na_position = 'last')***

by - represents the criterion. Maybe a string representing the name of the column or a list of strings for a multicriteria sort - ["name1", "name2", ...].

axis - represents the sorting direction - 0 or ‘index’, 1 or ‘columns’.

ascending - the meaning of sorting. It is provided as a criterion, logical value or list of logical values.

inplace - indicates sorting in the same table.

kind - sorting method. Variant: {‘quicksort’, ‘mergesort’, ‘heapsort’}.

na_position - the place where the rows / columns that have NaN are placed for the sorting criteria. Variant: {‘first’, ‘last’}.

4. Summary, filtering and aggregation of data in pandas

`DataFrame` objects can be subjected to aggregation, transformation, filtering operations. The first operation that is performed before aggregation, transformation or filtering is the division of data into groups according to various criteria (split). The `groupby` method does this:

```
DataFrame.groupby(by=None, axis=0, level=None, as_index=True,
sort=True, group_keys=True, squeeze=False, observed=False, **kwargs)
```

by - grouping criterion. It can be the name of a column, a series or a vector, or a function that identifies groups.

axis - grouping axis (0 - grouping on columns).

as_index - indicates the use of grouping keys as an index in the processing result, if the processing is an aggregation

sort - automatic sorting by grouping criteria keys

4. Summary, filtering and aggregation of data in pandas

The result of the processing is an object

pandas.core.groupby.DataFrameGroupBy

The methods of this class allow aggregation, transformation, filtering operations.

`DataFrameGroupBy` methods for amounts, averages, counting, variance, standard deviation and product:

GroupBy.sum (** kwargs)

GroupBy.mean (* args, ** kwargs)

GroupBy.count ()

GroupBy.var (ddof = 1, * args, ** kwargs)

GroupBy.std (ddof = 1, * args, ** kwargs)

GroupBy.prod (** kwargs)

4. Summary, filtering and aggregation of data in pandas

Data transformation can be achieved through the transform function:

`GroupBy.transform (func, * args, ** kwargs)`

`func` - is the applied function. The lambda operator can be used to transmit group data for transformation.

The functions that allow custom data aggregation are `agg` and `apply`:

`GroupBy.apply (func, * args, ** kwargs)`

`GroupBy.agg (arg, * args, ** kwargs)`

`func`, `arg` functions are the functions applied to groups.

4. Summary, filtering and aggregation of data in pandas

Example of aggregation, average by groups:

```
import pandas.core.groupby as gby

t1 = pd.DataFrame({
    'c1': [1, 1, 2, 1, 1, 2, 2, 2, 1],
    'c2': [10, 20, 30, 10, 40, 110, 140, 125, 100],
    'c3': [2.3, 2.6, 3, 0, 14, 10, 5.5, 11, 11.5]},
    index=['i1', 'i2', 'i3', 'i4', 'i5', 'i6', 'i7', 'i8', 'i9'])
print(t1)
g = t1.groupby('c1')

assert isinstance(g, gby.DataFrameGroupBy)
print("Medii pe grupe:", g.mean(), sep='\n')
```

```
Medii pe grupe:
        c2     c3
c1
1    36.00  6.080
2   101.25  7.375
```

4. Summary, filtering and aggregation of data in pandas

Example of aggregation, calculation of weights by groups:

```
# computations of weights on columns
def ponderi(x):
    return x / x.sum()

# call by employing a lambda expression
print("Ponderi pe grupe:",
    g.transform(func=lambda x: ponderi(x)),
sep="\n")
```

```
Ponderi pe grupe:
          c2        c3
i1  0.055556  0.075658
i2  0.111111  0.085526
i3  0.074074  0.101695
i4  0.055556  0.000000
i5  0.222222  0.460526
i6  0.271605  0.338983
i7  0.345679  0.186441
i8  0.308642  0.372881
i9  0.555556  0.378289
```

4. Summary, filtering and aggregation of data in pandas

Example of aggregation, calculation of standardized values by groups:

```
# Standardize
def standardize(x):
    medie = x.mean()
    std = x.std()
    xStd = (x - medie) / std
    return xStd

print("Valori standardizate pe grupe:“,
    g.transform(func=standardizare),
sep="\n")
```

```
Valori standardizate pe grupe:
          c2        c3
i1 -0.687552 -0.606319
i2 -0.423109 -0.558199
i3 -1.452494 -1.159814
i4 -0.687552 -0.975244
i5  0.105777  1.270383
i6  0.178377  0.695888
i7  0.789953 -0.497063
i8  0.484165  0.960989
i9  1.692435  0.869378
```

4. Summary, filtering and aggregation of data in pandas

Example of aggregation, calculation of weights by groups:

```
# computations of weights on columns
def ponderi(x):
    return x / x.sum()

# call by passing reference to function
print("Ponderi pe grupe:",
      g.transform(func=ponderi), sep="\n")
```

```
Ponderi pe grupe:
          c2        c3
i1  0.055556  0.075658
i2  0.111111  0.085526
i3  0.074074  0.101695
i4  0.055556  0.000000
i5  0.222222  0.460526
i6  0.271605  0.338983
i7  0.345679  0.186441
i8  0.308642  0.372881
i9  0.555556  0.378289
```

4. Summary, filtering and aggregation of data in pandas

Example of aggregation, calculation of weights by groups:

```python
# compute the sum of square weights
def sume2(x):
    p = x / x.sum()
    p2 = p * p
    return p2.sum()


print("Sume de patrate (apply):",
    g.apply(func=sume2), sep="\n")
print("Sume de patrate (agg):",
    g.agg(func=sume2), sep="\n")
```

```
Sume de patrate (apply):
      c1        c2        c3
c1
1   0.20  0.376543  0.368226
2   0.25  0.294010  0.299052
Sume de patrate (agg):
          c2        c3
c1
1   0.376543  0.368226
2   0.294010  0.299052
```

4. Summary, filtering and aggregation of data in pandas

The junction of the tables can be done by the merge method:

```
DataFrame.merge(right, how='inner', on=None, left_on=None,
right_on=None, left_index=False, right_index=False, sort=False,
suffixes=('_x', '_y'), copy=True, indicator=False,
validate=None)
```

right - the other table

how - the method of junction; default inner - on common keys

on - the name of the column after which the junction is made. Must belong to both tables

left_on - the name of the column after which the junction is made in the current table (considered left)

right_on - the name of the column after which the junction is made in the right table

4. Summary, filtering and aggregation of data in pandas

The junction of the tables can be done by the merge method:

left_index, right_index - indicates whether the junction is indexed for the current or right table; if the junction is made after the index, the column name no longer needs to be specified.

sort - indicates sorting by keys after junction

suffixes - suffix inserted for columns with common names in the two tables

The method returns the new table with the results of the junction. If the junction is made after indexes, the resulting table will retrieve the index of the tables, otherwise the resulting table will not keep any index.

4. Summary, filtering and aggregation of data in pandas

Example of merge without columns provide for junction:

```
t1 = pd.DataFrame({
    'c1': [1, 2, 3, 4, 5, 6, 7, 8, 9],
    'c2': [10, 20, 30, 10, 40, 110, 140, 125, 100],
    'c3': [2.3, 2.6, 3, 0, 14, 10, 5.5, 11, 11.5]},
    index=['i1', 'i2', 'i3', 'i4', 'i5', 'i6', 'i7’,
       'i8', 'i9'])
t2 = pd.DataFrame({
    'c4': [1, 3, 5, 7, 9],
    'c5': ["A", "B", "C", "D", "E"],
    'c6': [100, 200, 300, 100, 400]},
    index=['i1', 'i3', 'i5', 'i7', 'i9'])
t_1 = t1.merge(t2, left_index=True,
      right_index=True)
print(t_1)
```

```
    c1   c2    c3  c4 c5   c6
i1   1   10   2.3   1  A  100
i3   3   30   3.0   3  B  200
i5   5   40  14.0   5  C  300
i7   7  140   5.5   7  D  100
i9   9  100  11.5   9  E  400
```

4. Summary, filtering and aggregation of data in pandas

Example of merge with supplied columns for junction:

```
t1 = pd.DataFrame({
    'c1': [1, 2, 3, 4, 5, 6, 7, 8, 9],
    'c2': [10, 20, 30, 10, 40, 110, 140, 125, 100],
    'c3': [2.3, 2.6, 3, 0, 14, 10, 5.5, 11, 11.5]},
    index=['i1', 'i2', 'i3', 'i4', 'i5', 'i6', 'i7',
       'i8', 'i9'])
t2 = pd.DataFrame({
    'c4': [1, 3, 5, 7, 9],
    'c5': ["A", "B", "C", "D", "E"],
    'c6': [100, 200, 300, 100, 400]},
    index=['i1', 'i3', 'i5', 'i7', 'i9'])
t_2 = t1.merge(t2, left_on="c1",
      right_on="c4")
print(t_2)
```

```
   c1   c2    c3  c4 c5   c6
0   1   10   2.3   1  A  100
1   3   30   3.0   3  B  200
2   5   40  14.0   5  C  300
3   7  140   5.5   7  D  100
4   9  100  11.5   9  E  400
```