Experiment No.: 4

Title: Implementation of exploratory data analysis using Pandas dataframes.

Objectives: 1. To learn pandas.

Theory:

Pandasis an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.

Pandas can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyze.

Key Features:

Following are the important features of pandas −

- Fast and efficient DataFrame object with default and customized indexing.
- Tools for loading data into in-memory data objects from different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of date sets.
- Label-based slicing, indexing and subsetting of large data sets.
- Columns from a data structure can be deleted or inserted.
- Group by data for aggregation and transformations.
- High performance merging and joining of data.
- Time Series functionality.

Pandas Data Structures:

Pandas deals with the following three data structures that are built on top of Numpy array& due to this reason they are fast.

Data Structure	Dimensions	Description
Series	1	1-D labeled homogeneous array, sizeimmutable.
Data Frames	2	General 2-D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.
Panel	3	General 3-D labeled, size-mutable array.

Best way tothink of these data structures is that the higher dimensional data structure is a container of its lower dimensional data structure. For example, DataFrame is a container of Series, Panel is a container of DataFrame.

Using Pandas data structures, the mental effort of the user is reduced.For example, with tabular data (DataFrame) it is more semantically helpful to think of the index (the rows) and the columns rather than axis 0 and axis 1.

Mutability

All Pandas data structures are value mutable (can be changed) and except Series all are size mutable. Series is size immutable.

Series:

Seriesis a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index.

The basic method to create a Series is to call:

s = pd.Series(data, index=index)

Here, **data** can be many different things:

- a Python dictionary
- an ndarray
- a scalar value (e.g 5)
- Series acts very similarly to andarray, and is a valid argument to most NumPy functions. However, operations such as slicing will also slice the index.
- A Series is like a fixed-size dict in that you can get and set values by index label.(e.g. s[‘a’])
- Series can also have a name attribute:

  s = pd.Series(np.random.randn(5), name='something')

The passed index is a list of axis labels. Thus, this separates into a few cases depending on what data is.

1. When the data is a dict, and an index is not passed, the Series index will be ordered by the dict’s insertion order, if you’re using Python version >= 3.6 and Pandas version >= 0.23.

   ```
   d = {'b': 1, 'a': 0, 'c': 2}

   In [1]: pd.Series(d)
   Out[1]:
       b   1
       a   0
       c   2
       dtype: int64
   ```

2. If **data** is an **ndarray**, index must be the same length as data. If no index is passed, one will be created having values [0, ...,len(data) - 1].

```
s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])

In [4]: s
Out[4]:
a  0.469112
b  -0.282863
c  -1.509059
d  -1.135632
e  1.212112dtype: float64
```

3. If **data** is a **scalar** value, an index must be provided. The value will be repeated to match the length of index.

```
In [12]: pd.Series(5., index=['a', 'b', 'c', 'd', 'e'])
Out[12]:
a  5.0
b  5.0
c  5.0
d  5.0
e  5.0
dtype: float64
```

DataFrames

A DataFrame is a two-dimensional array with heterogeneous data.That is, DataFrame accepts many different kinds of input:

- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- Structured or record ndarray
- A Series
- Another DataFrame

Along with the data, you can optionally pass index (row labels) and columns (column labels) arguments. If you pass an index and / or columns, you are guaranteeing the index and / or columns of the resulting DataFrame. Thus, a dict of Series plus a specific index will discard all data not matching up to the passed index.

Data is aligned in a tabular fashion in rows and columns. For example,

Name	Age	Gender	Rating
Steve	32	Male	3.45
Lia	28	Female	4.6
Vin	45	Male	3.9
Katie	38	Female	2.78

The table represents the data of a sales team of an organization with their overall performance rating. The data is represented in rows and columns. Each column represents an attribute and each row represents a person.

Key Points

- Homogeneous data
- Size- Mutable
- Data Mutable
- Potentially columns are of different types
- Labeled axes (rows and columns)
- Can Perform Arithmetic operations on rows and columns

The parameters of the constructor are as follows −

Sr.No	Parameter & Description
1	**data** data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
2	**index** For the row labels, the Index to be used for the resulting frame is Optional Default np.arrange(n) if no index is passed.
3	**columns** For column labels, the optional default syntax is - np.arrange(n). This is only true if no index is passed.
4	**dtype** Data type of each column.
5	**copy** This command (or whatever it is) is used for copying of data, if the default is False.

DataFrameFrom dict of Series or dicts

```
d = {'one': pd.Series([1., 2., 3.], index=['a', 'b', 'c']),  'two': pd.Series([1., 2., 3., 4.],
      index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
```

```
df
Out[1]:
        one  two
        a  1.0  1.0
        b  2.0  2.0
        c  3.0  3.0
        d  NaN  4.0
```

DataFramefrom dict of ndarrays / lists

```
d = {'one': [1., 2., 3., 4.],  'two': [4., 3., 2., 1.]}
pd.DataFrame(d)
Out[1]:
        one  two
        0  1.0  4.0
        1  2.0  3.0
        2  3.0  2.0
        3  4.0  1.0
```

DataFrame fromstructured or record array

```
data = np.zeros((2, ), dtype=[('A', 'i4'), ('B', 'f4'), ('C', 'a10')])

data[:] = [(1, 2., 'Hello'), (2, 3., "World")]

pd.DataFrame(data)
Out[1]:
          A    B        C
        0  1  2.0  b'Hello'
        1    2  3.0  b'World'
```

DataFrame from a Series

The result will be a DataFrame with the same index as the input Series, and with one column whose name is the original name of the Series (only if no other column name provided).

Indexing/Selection:

The basics of indexing are as follows:

Operation	Syntax	Result
Select column	df[col]	Series
Select row by label	df.loc[label]	Series
Select row by integer location	df.iloc[loc]	Series
Slice rows	df[5:10]	DataFrame
Select rows by boolean vector	df[bool_vec]	DataFrame