Class : XII (INFORMATICS PRACTICES)

Self Study Material / Notes – III

Covariance

Covariance is applied on series data. The Series object has a method cov to compute covariance between series objects. NA will be excluded automatically.

Cov Series

```
import pandas as pd
import numpy as np
s1 = pd.Series(np.random.randn(10))
s2 = pd.Series(np.random.randn(10))
print s1.cov(s2)
```

Its **output** is as follows −

```
-0.12978405324
```

Covariance method when applied on a DataFrame, computes **cov** between all the columns.

```
import pandas as pd
import numpy as np
frame = pd.DataFrame(np.random.randn(10, 5), columns=['a', 'b',
'c', 'd', 'e'])
print frame['a'].cov(frame['b'])
print frame.cov()
```

Its **output** is as follows −

```
-0.58312921152741437

          a         b         c         d         e
a  1.780628 -0.583129 -0.185575  0.003679 -0.136558
b -0.583129  1.297011  0.136530 -0.523719  0.251064
c -0.185575  0.136530  0.915227 -0.053881 -0.058926
d  0.003679 -0.523719 -0.053881  1.521426 -0.487694
e -0.136558  0.251064 -0.058926 -0.487694  0.960761
```

Note − Observe the **cov** between **a** and **b** column in the first statement and the same is the value returned by cov on DataFrame.

Correlation

Correlation shows the linear relationship between any two array of values (series). There are multiple methods to compute the correlation like pearson(default), spearman and kendall.

```
import pandas as pd
import numpy as np
frame = pd.DataFrame(np.random.randn(10, 5), columns=['a', 'b',
'c', 'd', 'e'])

print frame['a'].corr(frame['b'])
```

```
print frame.corr()
```

Its **output** is as follows −

```
-0.383712785514

          a         b         c         d         e
a  1.000000 -0.383713 -0.145368  0.002235 -0.104405
b -0.383713  1.000000  0.125311 -0.372821  0.224908
c -0.145368  0.125311  1.000000 -0.045661 -0.062840
d  0.002235 -0.372821 -0.045661  1.000000 -0.403380
e -0.104405  0.224908 -0.062840 -0.403380  1.000000
```

If any non-numeric column is present in the DataFrame, it is excluded automatically.

Linear Regression

In Linear Regression two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

The functions in Seaborn to find the linear regression relationship is regplot. The below example shows its use.

```
import seaborn as sb
from matplotlib import pyplot as plt
df = sb.load_dataset('tips')
sb.regplot(x = "total_bill", y = "tip", data = df)
plt.show()
```

Python Pandas - Introduction

Pandas is 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.

In 2008, developer Wes McKinney started developing pandas when in need of high performance, flexible tool for analysis of data.

Prior to Pandas, Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyze.

Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Key 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 deals with the following three data structures −

- Series
- DataFrame
- Panel

These data structures are built on top of Numpy array, which means they are fast.

Dimension & Description

The best way to think 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.

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

Building and handling two or more dimensional arrays is a tedious task, burden is placed on the user to consider the orientation of the data set when writing functions. But 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.

Note − DataFrame is widely used and one of the most important data structures. Panel is used much less.

Series

Series is a one-dimensional array like structure with homogeneous data. For example, the following series is a collection of integers 10, 23, 56, …

10	23	56	17	52	61	73	90	26	72

Key Points

- Homogeneous data
- Size Immutable
- Values of Data Mutable

DataFrame

DataFrame is a two-dimensional array with heterogeneous data. 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.

Data Type of Columns

The data types of the four columns are as follows −

Column	Type
Name	String
Age	Integer
Gender	String
Rating	Float

Key Points

- Heterogeneous data
- Size Mutable
- Data Mutable

Panel

Panel is a three-dimensional data structure with heterogeneous data. It is hard to represent the panel in graphical representation. But a panel can be illustrated as a container of DataFrame.

Key Points

- Heterogeneous data
- Size Mutable
- Data Mutable

pandas.Series

A pandas Series can be created using the following constructor −

```
pandas.Series( data, index, dtype, copy)
```

The parameters of the constructor are as follows −

Sr.No	Parameter & Description
1	**data** data takes various forms like ndarray, list, constants
2	**index** Index values must be unique and hashable, same length as data. Default **np.arrange(n)** if no index is passed.
3	**dtype** dtype is for data type. If None, data type will be inferred
4	**copy** Copy data. Default False

A series can be created using various inputs like −

- Array
- Dict
- Scalar value or constant

Create an Empty Series

A basic series, which can be created is an Empty Series.

Example

```
#import the pandas library and aliasing as pd
import pandas as pd
s = pd.Series()
print s
```

Its **output** is as follows −

```
Series([], dtype: float64)
```

Create a Series from ndarray

If data is an ndarray, then index passed must be of the same length. If no index is passed, then by default index will be **range(n)** where **n** is array length, i.e., [0,1,2,3…. **range(len(array))-1].**

Example 1

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data)
print s
```

Its **output** is as follows −

```
0   a
1   b
2   c
3   d
dtype: object
```

We did not pass any index, so by default, it assigned the indexes ranging from 0 to **len(data)-1**, i.e., 0 to 3.

Example 2

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[100,101,102,103])
print s
```

Its **output** is as follows −

```
100  a
101  b
102  c
103  d
dtype: object
```

We passed the index values here. Now we can see the customized indexed values in the output.

Create a Series from dict

A **dict** can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index. If **index** is passed, the values in data corresponding to the labels in the index will be pulled out.

Example 1

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
print s
```

Its **output** is as follows −

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

Observe − Dictionary keys are used to construct index.

Example 2

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data,index=['b','c','d','a'])
print s
```

Its **output** is as follows −

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

Observe − Index order is persisted and the missing element is filled with NaN (Not a Number).

Create a Series from Scalar

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

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
s = pd.Series(5, index=[0, 1, 2, 3])
print s
```

Its **output** is as follows −

```
0  5
1  5
2  5
3  5
dtype: int64
```

Accessing Data from Series with Position

Data in the series can be accessed similar to that in an **ndarray.**

Example 1

Retrieve the first element. As we already know, the counting starts from zero for the array, which means the first element is stored at zeroth position and so on.

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the first element
print s[0]
```

Its **output** is as follows −

```
1
```

Example 2

Retrieve the first three elements in the Series. If a : is inserted in front of it, all items from that index onwards will be extracted. If two parameters (with : between them) is used, items between the two indexes (not including the stop index)

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the first three element
print s[:3]
```

Its **output** is as follows −

```
a  1
b  2
c  3
dtype: int64
```

Example 3

Retrieve the last three elements.

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the last three element
print s[-3:]
```

Its **output** is as follows −

```
c  3
d  4
e  5
dtype: int64
```

ASSIGNMENT

1.Write a Numpy program to compute the covariance matrix of two given arrays.
Required output should be
Original Array 1 [0 1 2]
Original Array 2 [2 1 0]
Covariance Matrix of the said arrays :
[[1. -1.]
[-1. 1]]

2. Write a Numpy program to compute cross-correlation of the two given arrays.
Original Array 1 [0 1 3]
Original Array 2 [2 4 5]
Cross Relation of the said arrays :
[[2.333333333 2.1666666667]
[2.16666666667 2.3333333333]]

3. Write a Numpy program to compute Pearson's product moment correlation coefficient of the two given arrays.
Original Array 1 [0 1 3]
Original Array 2 [2 4 5]
Pearson's product moment correlation coefficient of the said arrays :
[[1. 0.92857143]
[0.92857143 1.]]

4. Differentiate between covariance and correlation.
5. Computer the linear relationship between head size and brain weight using the concept of linear regression. The datasets are as under :

Head Size (CM^3)	Brain Weight (grams)
4512	1530
3738	1297
4261	1335
3777	1282

Brain weight = c + m + Headsize

Reference :

Variance

In probability, the variance of some random variable X is a measure of how much values in the distribution vary on average with respect to the mean.

The variance is denoted as the function Var() on the variable.

Var[X]

Variance is calculated as the average squared difference of each value in the distribution from the expected value. Or the expected squared difference from the expected value.

Var[X] = E[(X - E[X])^2]

Assuming the expected value of the variable has been calculated (E[X]), the variance of the random variable can be calculated as the sum of the squared difference of each example from the expected value multiplied by the probability of that value.

If the probability of each example in the distribution is equal, variance calculation can drop the individual probabilities and multiply the sum of squared differences by the reciprocal of the number of examples in the distribution.

In the abstract, the sample variance is denoted by the lower case sigma with a 2 superscript indicating the units are squared, not that you must square the final value. The sum of the squared differences is multiplied by the reciprocal of the number of examples minus 1 to correct for a bias.

In NumPy, the variance can be calculated for a vector or a matrix using the var() function. By default, the var() function calculates the population variance. To calculate the sample variance, you must set the ddof argument to the value 1.

The example below defines a 6-element vector and calculates the sample variance.

```
from numpy import array
from numpy import var
v = array([1,2,3,4,5,6])
print(v)
result = var(v, ddof=1)
print(result)
```

Running the example first prints the defined vector and then the calculated sample variance of the values in the vector

[1 2 3 4 5 6]

3.5

The var function can calculate the row or column variances of a matrix by specifying the axis argument and the value 0 or 1 respectively, the same as the mean function above.

The example below defines a 2×6 matrix and calculates both column and row sample variances.

```
from numpy import array
from numpy import var
M = array([[1,2,3,4,5,6],[1,2,3,4,5,6]])
print(M)
col_mean = var(M, ddof=1, axis=0)
print(col_mean)
row_mean = var(M, ddof=1, axis=1)
print(row_mean)
```

Running the example first prints the defined matrix and then the column and row sample variance values.

```
[[1 2 3 4 5 6]
 [1 2 3 4 5 6]]
 [ 0.  0.  0.  0.  0.  0.]

[ 3.5  3.5]
```

The standard deviation is calculated as the square root of the variance and is denoted as lowercase “s”.

To keep with this notation, sometimes the variance is indicated as s^2, with 2 as a superscript, again showing that the units are squared.

NumPy also provides a function for calculating the standard deviation directly via the std() function. As with the var() function, the dd of argument must be set to 1 to calculate the unbiased sample standard deviation and column and row standard deviations can be calculated by setting the axis argument to 0 and 1 respectively.

The example below demonstrates how to calculate the sample standard deviation for the rows and columns of a matrix.

```
from numpy import array
from numpy import std
M = array([[1,2,3,4,5,6],[1,2,3,4,5,6]])
print(M)
col_mean = std(M, ddof=1, axis=0)
print(col_mean)
row_mean = std(M, ddof=1, axis=1)
print(row_mean)
```

Running the example first prints the defined matrix and then the column and row sample standard deviation values.

```
[[1 2 3 4 5 6]
 [1 2 3 4 5 6]]

[ 0.  0.  0.  0.  0.  0.]

[ 1.87082869  1.87082869]
```

Covariance

In probability, covariance is the measure of the joint probability for two random variables. It describes how the two variables change together.

It is denoted as the function cov(X, Y), where X and Y are the two random variables being considered.

cov(X,Y)
Covariance is calculated as expected value or average of the product of the differences of each random variable from their expected values, where E[X] is the expected value for X and E[Y] is the expected value of y.

cov(X, Y) = E[(X - E[X]) . (Y - E[Y])]

Assuming the expected values for X and Y have been calculated, the covariance can be calculated as the sum of the difference of x values from their expected value multiplied by the difference of the y values from their expected values multiplied by the reciprocal of the number of examples in the population.

cov(X, Y) = sum (x - E[X]) * (y - E[Y]) * 1/n

In statistics, the sample covariance can be calculated in the same way, although with a bias correction, the same as with the variance.

cov(X, Y) = sum (x - E[X]) * (y - E[Y]) * 1/(n - 1)

The sign of the covariance can be interpreted as whether the two variables increase together (positive) or decrease together (negative). The magnitude of the covariance is not easily interpreted. A covariance value of zero indicates that both variables are completely independent.

NumPy does not have a function to calculate the covariance between two variables directly. Instead, it has a function for calculating a covariance matrix called cov() that we can use to retrieve the covariance. By default, the cov()function will calculate the unbiased or sample covariance between the provided random variables.

The example below defines two vectors of equal length with one increasing and one decreasing. We would expect the covariance between these variables to be negative.

We access just the covariance for the two variables as the [0,1] element of the square covariance matrix returned.

```
from numpy import array
from numpy import cov
x = array([1,2,3,4,5,6,7,8,9])
print(x)
y = array([9,8,7,6,5,4,3,2,1])
print(y)
Sigma = cov(x,y)[0,1]
print(Sigma)
```

Running the example first prints the two vectors followed by the covariance for the values in the two vectors. The value is negative, as we expected.

```
[1 2 3 4 5 6 7 8 9]
[9 8 7 6 5 4 3 2 1]

-7.5
```

The covariance can be normalized to a score between -1 and 1 to make the magnitude interpretable by dividing it by the standard deviation of X and Y. The result is called the correlation of the variables, also called the Pearson correlation coefficient, named for the developer of the method.

```
r = cov(X, Y) / sX sY
```

Where r is the correlation coefficient of X and Y, cov(X, Y) is the sample covariance of X and Y and sX and sY are the standard deviations of X and Y respectively.

NumPy provides the corrcoef() function for calculating the correlation between two variables directly. Like cov(), it returns a matrix, in this case a correlation matrix. As with the results from cov() we can access just the correlation of interest from the [0,1] value from the returned squared matrix.

```
from numpy import array
from numpy import corrcoef
x = array([1,2,3,4,5,6,7,8,9])
print(x)
y = array([9,8,7,6,5,4,3,2,1])
print(y)
Sigma = corrcoef(x,y)
print(Sigma)
```

Running the example first prints the two defined vectors followed by the correlation coefficient. We can see that the vectors are maximally negatively correlated as we designed.

```
[1 2 3 4 5 6 7 8 9]
[9 8 7 6 5 4 3 2 1]

-1.0
```

Covariance Matrix

The covariance matrix is a square and symmetric matrix that describes the covariance between two or more random variables.

The diagonal of the covariance matrix are the variances of each of the random variables.

A covariance matrix is a generalization of the covariance of two variables and captures the way in which all variables in the dataset may change together.

The covariance matrix is denoted as the uppercase Greek letter Sigma. The covariance for each pair of random variables is calculated.

The covariance matrix provides a useful tool for separating the structured relationships in a matrix of random variables. This can be used to decorrelate variables or applied as a transform to other variables. It is a key element used in the Principal Component Analysis data reduction method, or PCA for short.

The covariance matrix can be calculated in NumPy using the cov() function. By default, this function will calculate the sample covariance matrix.

The cov() function can be called with a single matrix containing columns on which to calculate the covariance matrix, or two arrays, such as one for each variable.

Below is an example that defines two 9-element vectors and calculates the unbiased covariance matrix from them.

```
from numpy import array

from numpy import cov

x = array([1,2,3,4,5,6,7,8,9])

print(x)
```

```
y = array([9,8,7,6,5,4,3,2,1])

print(y)

Sigma = cov(x,y)

print(Sigma)
```

Running the example first prints the two vectors and then the calculated covariance matrix.

The values of the arrays were contrived such that as one variable increases, the other decreases. We would expect to see a negative sign on the covariance for these two variables, and this is what we see in the covariance matrix.