QF627 Programming and Computational Finance

S0305: Data Manipulation and Visualization

**Learning Outcomes:**

1. (DNT) Introduction to Moving Average Crossover strategy
2. Download CC3.SI.csv from eLearn
3. Sample Python code has 29 lines (including comments and empty lines).
4. Sample MATLAB code has 30 lines (including comments and empty lines).
5. After running the sample Python code, 6 *Types* of variables are listed in the “Variable explorer”:

|  |  |
| --- | --- |
| * axes.support * DataFrame * figure.Figure | * DatatimeIndex * module * Series |

1. After running the sample MATLAB code, 3 *classes* of variables are listed in the “Workspace”:

|  |
| --- |
| * datatime * table * double |

1. Dr. Z first explains the  Python /  MATLAB sample code.
2. There are 4 **import** statements in the sample Python code, which are:

* import pandas as pd
* import numpy as np
* import matplotlib.pyplot as plt
* import matplotlib.dates as mdates

1. True /  False **read\_csv** can be found in the list of attributes of **pd**.
2. In the documentation of **pandas.read\_csv**, we find the following default values of some parameters:

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Default Value | Parameter | Default Value |
| sep | ',' | header | 'infer' |
| names | None | index\_col | None |
| usecols | None | mangle\_dupe\_cols | True |
| skiprows | None | skip\_blank\_lines | True |
| parse\_dates | False |  |  |

1. True /  False In the function header of function pandas.read\_csv, there is no \*- or \*\*-parameter.
2. A few examples are used to demonstrate the use of parameters “**header**”, “**names**”, “**index\_col**” and “**parse\_dates**”.

|  |  |  |  |
| --- | --- | --- | --- |
| header | names | index\_col | parse\_dates |
| 'infer' or 0 |  |  |  |
| None |  |  |  |
|  | range(7,0,-1) |  |  |
| 0 | range(7,0,-1) |  |  |
|  |  | 0 |  |
|  |  | 0 | True |

1. True /  False **pandas.read\_csv** read CSV file into a **DataFrame**.
2. True /  False In a **DataFrame**, say **data**, **data.columns** returns the column labels of the **DataFrame**, **data.index** returns the index/row labels of the **DataFrame**. Both are list-like objects (or 1D array-like objects).
3. True /  False **Pandas.DataFrame** is a two-dimensional size-mutable, tabular data structure with labeled axes (rows and columns), which can be thought of as a dict-like container for Series objects. **Pandas.Series** is a one-dimensional (Numpy) **ndarray** with axis labels.
4. True /  False In the example on slide 27, both **data['Open']** and **data.Open** (the syntactic sugar of **data['Open']**) can be used to reference the data in the column labeled ‘Open’. However, for the column labeled ‘Adj Close’, only **data['Adj Close']** is valid. Such a dict-like indexing method on a DataFrame, i.e. **DataFrame[colname]**, results a Series.
5. True /  False In the example on slide 32, **data.index** is **Index** type and elements in **data.index** are **str** type.
6. True /  False In the example on slide 33, with **parse\_dates=True** in the **read\_csv** function, **data.index** is **DatetimeIndex** type and elements in **data.index** are **Timestamp** type.
7. True /  False In the example on slide 34, **pandas.DataFrame.values** returns a Numpy representation of the **DataFrame**. Only the values in the **DataFrame** will be returned, the axes labels will be removed.
8. True /  False In the file CC3.SI.csv, there is a row where the ‘Volume’ eqauls 0. In the corresponding **DataFrame**, we also see this row. The following statement will remove this row:

**data.drop(data.index[data['Volume']==0],inplace=True)**

1. True /  False In the example on slide 36, **data['Volume']** is a **Series**.
2. True /  False Comparison between a **Series** and a scalar results a **Series**. For example, **data['Volume']==0** returns a **Series** of **True** and **False** with the same labels.
3. True /  False Numpy **ndarrays** and Pandas **Series**es support boolean indexing, i.e. to use a list or an array of **True** or **False** as the index. The size of a boolean index must match the indexed array’s size.
4. True /  False **data.index[data['Volume']==0]** returns the row indices corresponding to those rows where Volume equals zero.
5. True /  False If using “**inplace=False**”, **DataFrame.drop** will result a copy, original data remains unchanged.
6. True /  False **DataFrame.drop** uses **axis=0** as the default parameter value. It will remove rows with the specified labels. We can use **axis=1** and column labels to remove columns.
7. True /  False **Series.rolling** results a Rolling object. Methods available to a Rolling object include: **.sum()**, **.mean()**, **.std()**, **.max()**, etc.
8. True /  False **RollingObject.mean()** results a Series. Test the following:

**data['Adj Close'].rolling(15).mean()**

1. True /  False We can add a new column to a **DataFrame** as adding a new item to a dictionary with the keyword index. We cannot use the “dot notation” to add a column to a DataFrame.
2. Dr. Z explains why there are many **NaN** in the two newly added columns.
3. True /  False **numpy.round(Series, 3)** is equivalent to **Series.round(3)**.
4. True /  False Subtraction of two **Series**es (having the same labels) results a **Series** with the same labels.
5. True /  False In the assignment to a slice of a **Series**, **target=expression**, **expression** can be a scalar or an iterable having the same number of items as the number of items in the target.
6. True /  False **Series.diff** returns a **Series** of the same length (and same labels).
7. True /  False For a Series **y**, **y[(y>0) | (y<0)]** returns a series with those rows in series **y** whose values are either greater than zero or less than zero.
8. True /  False In the above expression, **y[(y>0) | (y<0)]**, we can omit the parenthesis and use **y[y>0 | y<0]**.
9. True /  False Python logical OR operator (**or**) is **not** an element-wise operator.
10. True /  False Python bitwise logical OR operator (**|**) is **not** an element-wise operator.
11. True /  False Numpy **ndarray** does not have logical OR operation. Numpy has the **logical\_or** function for the logical OR operation. It is an element-wise operation.
12. True /  False Numpy **ndarray** has bitwise OR operation. It is an element-wise operation. It is a common practice to use the bitwise OR operator (**|**) on Numpy logical **ndarray**s for the element-wise OR operation on logical arrays.
13. Find the values in the column “crossSell” for the following rows (corresponding to idxSell):

Use expression: data.loc[idxSell,'crossSell']

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Adj Close | crossSell | |
| **Before** running In-Class Exercise 31 | **After** running In-Class Exercise 31 |
| **2015-12-22** | **3.278104** | NaN | 3.278104 |
| **2016-01-08** | **3.232448** | NaN | 3.232448 |
| **2016-03-02** | **3.086349** | NaN | 3.086349 |
| **2016-08-31** | **3.489341** | NaN | 3.489341 |
| **2017-02-14** | **2.701995** | NaN | 2.701995 |
| **2017-04-26** | **2.729855** | NaN | 2.729855 |
| **2017-07-07** | **2.700000** | NaN | 2.700000 |

1. Dr. Z demonstrated what a Figure and an Axes are, respectively.
2. True /  False **matplotlib.pyplot.subplots** creates a Figure and a set of subplots/axes.
3. True /  False **matplotlib.pyplot.figure** creates a new Figure and make it the ***Current Figure***. **matplotlib.pyplot.axes** add an axes to the ***Current Figure*** and make it the ***Current Axes***.
4. True /  False Almost all functions from **pyplot**, such as **plt.plot()**, are implicitly either referring the an existing ***Current Figure*** and ***Current Axes***, or creating them anew if none exist.
5. True /  False For a complicated plot, it will be clearer and more convenient to use names for Figures and Axes.
6. True /  False (DNT) **matplotlib.pyplot.subplot2grid** creates an axes at specific location inside a regular grid. This function has an input parameter to tell which figure to place axes in. Default figure is the ***Current Figure***.
7. True /  False **pandas.DataFrame.plot** can specify axes, figure size and line styles through parameters **ax**, **figsize** and **style**, respectively.
8. Some color abbreviations supported in Matplotlib:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| blue | green | red | cyan | magenta | yellow | black | white |
| 'b' | 'g' | 'r' | 'c' | 'm' | 'y' | 'k' | 'w' |

1. Some line styles supported in Matplotlib:

|  |  |  |  |
| --- | --- | --- | --- |
| solid line | dashed line | dash-dot line | dotted line |
| '-' | '--' | '-.' | ':' |

1. True /  False The format string **'g-'** is for **green solid line** and **'ro'** is for **red circles**.
2. True /  False **DataFrame[list\_colnames]** returns a DataFrame corresponding to **list\_colnames**.
3. True /  False There are two **fill\_between** functions:

* **matplotlib.pyplot.fill\_between** and
* **matplotlib.axes.Axes.fill\_between**

1. True /  False In the **fill\_between** function, **x** and **y1** are compulsory arguments, **y2** is an optional argument. **x** can be a 1D array or a sequence type. **y1** and **y2** can be 1D arrays, sequences of the same length or a scalar.
2. For slide 79, Dr. Z demonstrated different choices for **x** and explained the **xy** in the sample code.
3. True /  False In **fill\_between**, we can set the fill color by the parameter **color**, and the transparency of the fill color by the parameter **alpha**.
4. Sample code for plotting the **moving average** and **moving standard deviation** using **matplotlib.pyplot.fill\_between**.
5. (DNT) Sample code for the GUI that contains an Axes using PyQt5.
6. True /  False There is a library function **Matplotlib.finance.candlestick\_ohlc** for the candlestick graph. It has a parameter **ax** to specify the Axes instance to plot to. The parameter **quotes** needs to be a sequence of **(time, open, high, low, close, …)** sequences, as long as the first 5 elements are these values, the record can be as long as you want (e.g., it may store volume). **time** must be in float days format (see **date2num**).
7. Complete the following code as given in In-Class Exercise 35.

|  |
| --- |
| **%234567890123456789012345678901234567890123456789012345678901234567890**  **from** **matplotlib.finance** **import candlestick\_ohlc**  **from matplotlib.dates** **import date2num**  **r=data.iloc[:15, :]**  **fig,ax=plt.subplots()**  **d=date2num(r.index.date)**  **candlestick\_ohlc(ax, zip(d,r.Open,r.High,r.Low,r.Close)**  **width=0.5, colorup='g', colordown='r', alpha=1)**  **plt.setp(ax.get\_xticklabels(), rotation=30)**  **ax.xaxis\_date()**  **plt.show()** |

1. (DNT) Dr. Z demonstrated the use of **QFileDialog.getOpenFileName**.
2. True /  False Numpy arrays use a different structure which uses less pace, runs faster and has optimized functions. A Python list’s advantage is its flexibility.
3. True /  False Python Lists do not support element-wise operations. Element-wise operations can be realized through the list comprehension.
4. Introduction to two Python built-in functions: **all(iterable)** and **any(iterable)**.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Return value |  | Return value |
| **all([1, 2, 3])** | True | **any([1, 2, 3])** | True |
| **all([1, 0, 3])** | False | **any([1, 0, 3])** | True |
| **all([])** | True | **any([])** | False |
| **all([[]])** | False | **any([[]])** | False |

1. Use list comprehension to solve the following problems.

x=[1, 2, 3, 4]

|  |  |
| --- | --- |
| Compare x>1 element by element. | **[i>1 for i in x]** |
| Is there any element of x greater than 1? | **any([i-1 for i in x])** |
| Are all elements of x greater than 1? | **all([i-1 for i in x])** |
| Compute x+1 element by element. | **[i+1 for i in x]** |
| Compute x\*\*2 element by element. | **[i\*\*2 for i in x]** |
| Test x in range(5) element by element. | **[i in range(5) for i in x]** |
| Is there any element of x in range(5)? | **any([i in range(5) for i in x])** |
| Are all elements of x in range(5)? | **all([i in range(5) for i in x])** |

1. True /  False Numpy arrays support some vectorized computations, such as comparison, addition, power, etc. The membership test operation (**in**) is not among these operations. Elementwise membership test is done by the library function **numpy.isin**.
2. True /  False **numpy.array** creates an array (**numpy.ndarray** type). It can convert a list of numbers to a 1D array, and a nested list of numbers (rectangular shape) to a 2D array.
3. True /  False Pandas DataFrame supports some vectorized computations, such as comparison, addition, power, etc. Elementwise membership test is done by the library function **pandas.DataFrame.isin**.
4. True /  False **pandas.DataFrame** creates/is the primary pandas data structure. The parameter **data** can be a Numpy ndarray, a dictionary, or a list-like object, etc. It is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Parameters **index** and **columns** are used to specify row labels and column labels, respectively, which will default to **np.arange(n)**.
5. True /  False Python built-in functions **any** and **all** only work on 1D arrays. For 2D arrays, we can use Numpy library functions **numpy.any** and **numpy.all**.
6. True /  False Numpy library function **numpy.any** returns single Boolean unless **axis** is not **None**. For a 2D array, for **axis=0,** logical OR is applied along the first index, i.e. on the elements in the same column; for **axis=1,** logical OR is applied along the second index, i.e. on the elements in the same row.
7. Return values of **numpy.any**:

|  |  |
| --- | --- |
|  | Return Value |
| **np.any([[True,False],[True,False]])** | **True** |
| **np.any([[True,False],[True,False]], axis=0)** | **[True,False]** |
| **np.any([[True,False],[True,False]], axis=1)** | **[True,True]** |
| **np.any([[True,False],[True,False]], axis=-1)** | **[True,True]** |

1. Indexing and Slicing of a nested list.

**x=[[0,1,2],[3,4,5],[6,7,8]]**

|  |  |
| --- | --- |
| **Expression** | **Value (or Error)** |
| **x** | **[[0,1,2],[3,4,5],[6,7,8]]** |
| **x[0]** | **[0,1,2]** |
| **x[:1]** | **[[0,1,2]]** |
| **x[1][1]** | **4** |
| **x[:1][:1]** | **[[0,1,2]]** |
| **x[1,1]** | **error** |
| **x[:1,:1]** | **error** |

1. Indexing and Slicing of a Numpy 2D array.

**import numpy as np**

**x=np.arange(9).reshape(3,-1)**

|  |  |
| --- | --- |
| **Expression** | **Value (or Error)** |
| **x** | **array([[0,1,2]**  **[3,4,5]**  **[6,7,8]])** |
| **x[0]** | **array([0,1,2])** |
| **x[:1]** | **array([[0,1,2]])** |
| **x[1][1]** | **4** |
| **x[:1][:1]** | **array([[0,1,2]])** |
| **x[1,1]** | **4** |
| **x[:1,:1]** | **array([[0]])** |

1. True /  False List indices must be integers or slices, not tuple or list.
2. On slide 115, more examples on Numpy arrays’ indexing and slicing are given, including boolean indexing (using a list of True or False) and fancy indexing (using a list of indices).
3. True /  False For a Numpy 2D array , say **x**, **x[1]** and **x[[1]]** are of different structures though they contain the same numbers.
4. True /  False (Review) Slicing of a list is a shallow copy.
5. True /  False (Review) Only iterables can be assigned to the slicing of a list. The iterable assigned to extended slicing must match in size.
6. True /  False Slicing of a Numpy array is a view. Fancy indexing returns a copy. We can use Numpy library function **numpy.ndarray.base** to tell whether memory is from some other object. For example,

|  |
| --- |
| **import numpy as np**  **x=np.arange(4)**  **y=x[:2]**  **z=x[[0,1]]**  **print(y.base is x, z.base is x) #True False**  **x[0]=100**  **print(y, z) #[100 1] [0 1]** |

1. True /  False Comparison of single indexing and slicing via **[]** on Pandas DataFrame and Numpy 2D arrays. Slicing will return a DataFrame or a 2D array on rows, respectively. Indexing and fancy indexing on Numpy 2D arrays will return rows. Indexing and fancy indexing on DataFrames will return columns. Numpy arrays can use multiple indexing/slicing. Pandas DataFrame does not have multiple indexing/slicing via **[]**.

|  |  |  |  |
| --- | --- | --- | --- |
| **import numpy as np**  **import pandas as pd**  **x=np.arange(9).reshape(3,-1)**  **y=pd.DataFrame(x)** | | | |
| x[0] | **y[0]** | x[:1] | y[:1] |
| [0 1 2] | **0 0**  **1 3**  **2 6**  **Name: 0, dtype: int32** | [[0 1 2]] | 0 1 2  0 0 1 2 |
| x[[0]] | **y[[0]]** | x[1,1] | **y[1,1]** |
| [[0 1 2]] | **0**  **0 0**  **1 3**  **2 6** | 4 | **ERROR** |

1. True /  False For Pandas DataFrame data selection via **.iloc**, which is integer position based, **df.iloc[1]** is equivalent to **df.iloc[1,:]**. All single indexing and slicing via **.iloc** return rows.
2. True /  False For Pandas DataFrame data selection via **.loc**, which is label based, **df.loc[1]** is equivalent to **df.loc[1,:]**. All single indexing and slicing via **.loc** return rows. **IMPORTANT:** The slice object with labels **includes both the start and the stop**. This is different from usually Python slices.
3. True /  False Numpy library function **numpy.arange** returns evenly spaced values within a given interval. When using a non-integer step, such as 0.1, the results will often not be consistent. It is better to use **numpy.linspace** for these cases.
4. True /  False Numpy library function **numpy.ndarray.reshape** returns an array containing the **same data** with a new shape.
5. True /  False With Numpy ndarray’s broadcasting, we can perform binary operations on arrays of different sizes. For example,

|  |  |
| --- | --- |
| **Expression** | **Value (or Error)** |
| **np.arange(3)+5** | **[5 6 7]** |
| **np.ones((3,3))+np.arange(3))** | **[[1. 2. 3.]**  **[1. 2. 3.]**  **[1. 2. 3.]]** |
| **np.arange(3).reshape((3,1))+np.arange(3)** | **[[0 1 2]**  **[1 2 3]**  **[2 3 4]]** |

1. True /  False Numpy library function **numpy.ones** returns a new array of given shape and type, filled with ones.
2. True /  False Pandas DataFrame does not have Numpy ndarray’s broadcasting.
3. Dr. Z demonstrated an example of subtraction of two Pandas Series with different labels.
4. Implementation of the algorithm on slide 139 (binomial tree method for option pricing):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Variable Names | | | | | | | |
|  |  |  |  |  |  |  |  |
| S | K | r | q | t | T | sigma | N |
|  |  |  |  |  |  |  |  |
| dt | u | d | p | V | i | j | ??? |

1. True /  False Numpy library function **numpy.maximum(x1,x2)** compares two arrays and returns a new array containing the element-wise maxima. If one of the elements being compared is a NaN, then that element (NaN) is returned. If both elements are NaNs then the first (NaN) is returned. When comparing two arrays of different sizes, Numpy array’s broadcasting will apply.
2. **Binomial Tree Algorithm Implentations 1&2**: f[i][j] using a nested list.

|  |
| --- |
| **#23456789012345678901234567890123456789012345678901234567890123456789012345**  **from math import exp, sqrt**  **def BTA(S,K,r,q,tau,sigma,N=100):**  **#1**  **deltaT=tau/N**  **u=exp(sigma\*sqrt(deltaT))**  **d=1/u**  **p=(exp((r-q)\*deltaT)-d)/(u-d)**  **#2**  **fc=[[0.0 for j in range(i+1)] for i in range(N+1)]**  **fp=[[0.0 for j in range(i+1)] for i in range(N+1)]**  **#if using list comprehension**  **#fc=[0]\*(N+1)**  **#fp=[0]\*(N+1)**    **for j in range(N+1):**  **fc[N][j]=max(0,S\*(u\*\*j)\*(d\*\*(N-j))-K)**  **fp[n][j]=max(0,K-S\*(u\*\*j)\*(d\*\*(N-j)))**  **#if using list comprehension**  **#fc[N]=[max(0, S\*(u\*\*j)\*(d\*\*(N-j))-K) for j in range(N+1)]**  **#fp[N]=[max(0, K-S\*(u\*\*j)\*(d\*\*(N-j))) for j in range(N+1)]**  **#3**  **ert=exp(-r\*deltaT)**  **p1=1-p**  **for i in range(N-1,0-1,-1):**  **for j in range(i+1):**  **fc[i][j]=ert\*(p\*fc[i+1][j+1]+p1\*fc[i+1][j])**  **fp[i][j]=ert\*(p\*fp[i+1][j+1]+p1\*fp[i+1][j])**  **#if using list comprehension**  **#fc[i]=[ert\*(p\*fc[i+1][j+1]+p1\*fc[i+1][j]) for j in range(i+1)]**  **#fp[i]=[ert\*(p\*fp[i+1][j+1]+p1\*fp[i+1][j]) for j in range(i+1)]**  **#4**  **return (fc[0][0], fp[0][0])**  **if \_\_name\_\_=='\_\_main\_\_':**  **S=50.0**  **K=50.0**  **t=0**  **T=183/365**  **sigma=0.4**  **r=0.04**  **q=0.01**  **N=100**  **print(BTA(S,K,r,q,t,T,sigma,N))** |

1. **Binomial Tree Algorithm Implentation 3**: f[i,j] using a Numpy 2D array.

|  |
| --- |
| **#23456789012345678901234567890123456789012345678901234567890123456789012345**  **import numpy as np**  **def BTA(S, K, r, q, tau, sigma, N=100):**  **#1**  **deltaT=tau/N**  **u=np.exp(sigma\*np.sqrt(deltaT)**  **d=1/u**  **p=(np.exp((r-q)\*deltaT)-d)/(u-d)**  **#2**  **p1=1-p**  **ert=np.exp(-r\*deltaT)**  **for i in range(N-1, 0-1, -1):**  **fc[i, 0:i+1]=ert\*(p\*fc[i+1, 0+1:i+1+1]+p1\*fc[i+1,0:i+1])**  **fp[i, 0:i+1]=ert\*(p\*fp[i+1, 0+1:i+1+1]+p1\*fp[i+1,0:i+1])**  **#3**  **p1=1-p**  **ert=np.exp(-r\*deltaT)**  **for i in range(N-1,0-1,-1):**  **fc[i,0:i+1]=ert\*(p\*fc[i+1,0+1:i+1+1]+p1\*fc[i+1,0:i+1])**  **fp[i,0:i+1]=ert\*(p\*fp[i+1,0+1:i+1+1]+p1\*fp[i+1,0:i+1])**  **#4**  **return (fc[0, 0], fp[0, 0])**  **if \_\_name\_\_=='\_\_main\_\_':**  **S=50.0**  **K=50.0**  **t=0**  **T=183/365**  **sigma=0.4**  **r=0.04**  **q=0.01**  **N=100**  **print(BTA(S,K,r,q,t,T,sigma,N))** |

1. True /  False The **numpy.random** module has library functions to generate random numbers from various distributions, e.g. **numpy.random.randn** returns a sample (or samples) from the standard normal distribution, **numpy.random.random** returns random floats in the half-open interval [0.0, 1.0) (a.k.a. the continuous uniform distribution).

|  |  |
| --- | --- |
| Generate a 3-by-4 2D ndarray of random numbers from the normal distribution, and name this array x. | **import numpy as np**  **x=np.random.randn(3,4)**  **#or**  **#x=np.random.standard\_normal((3,4))** |
| Generate a 1D array of 10 random numbers from the normal distribution, and name this array x. | **import numpy as np**  **x=np.random.randn(1,10)**  **#or**  **#x=np.random.standard\_normal((1,10))** |
| Generate a 3-by-4 2D ndarray of random numbers from the U[0,1) distribution, and name this array x. | **import numpy as np**  **x=np.random.random((3,4))** |
| Generate a 1D array of 10 random numbers from the U[0,1) distribution, and name this array x. | **import numpy as np**  **x=np.random.random((1,10))** |

1. True /  False Numpy library function **numpy.append(*arr*,*values*,*axis=None*)** appends ***values*** to a copy of ***arr***. If the parameter **axis** is **None**, the function returns a flattened array. ***values*** must be of the correct shape (same shape as ***arr***, excluding ***axis***). For two 2D arrays, with **axis=0** it will append the second array with shape **(m2, n)** to the first array with shape **(m1, n)** as new rows, and with **axis=1** it will append the second array with shape **(m, n2)** to the first array with shape **(m, n1)** as new columns.
2. True /  False Numpy library function **numpy.concatenate** join a sequence of arrays along an existing **axis**. The default value of **axis** is 0. **numpy.concatenate(([1,2],[3,4]))** is equivalent to **numpy.append([1,2],[3,4])**.
3. True /  False Numpy library function **numpy.random.seed** seeds the random number generator. It can be called again to re-seed the generator.

Q: What is printed by the third print statement?

|  |
| --- |
| **Code** |
| **import numpy as np**  **numpy.random.seed(0)**  **print(np.random.randn(2)) #1**  **print(np.random.randn(2)) #2**  **numpy.random.seed(0)**  **print(np.random.randn(4)) #3** |
| **Output** |
| **[1.76405235 0.40015721]**  **[0.97873798 2.2408932]**  **[1.76405235 0.40015721 0.97873798 2.2408932]** |

1. **Monte Carlo Simulation Algorithm Implementation (Sample Exam, Question 6)**

|  |
| --- |
| **import numpy as np** |
| Given , , , , , , , . |
| **S=50; K=50, T=183/365, sigma=0.4, r=0.04, q=0.01, N=10\_000\_000** |
| **Use one command** to create a vector of random numbers with standard normal distribution, denoted as : |
| **d=np.random.randn(N/2)** |
| **Use one command** and the library function **numpy.concatenate** to create a vector of elements by using as the first half, and as the second half. Name this vector as . |
| **p=np.concatenate((d,-d))** |
| **Use one command** to compute the mean and standard deviation of the numbers in the vector , and denote them as and , respectively. |
| **mu,sigma=np.mean(p),np.std(p)** |
| **Use one command** to update every element in the vector as follows: |
| **p[:,:]=(p[:,:]-mu)/sigma** |
| **Use one command** to obtain the vector computed as follows: |
| **Y=S\*np.exp((r-(sigma\*\*2)/2)\*T+sigma\*mp.sqrt(T)\*p)** |
| **Use one command** to obtain the vector computed as follows: |
| **h=np.maxmize(Y-K,0)** |
| **Use one command** to calculate and return as follows: |
| **V=np.exp(-r\*T)\*np.mean(h)** |