Computing Laboratory

More on Python

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

December, 2020

List comprehension

```
import random
ls = [10, 20, 30, 40, 50]
random.shuffle(ls)
print(ls)
```

Output: [40, 20, 10, 50, 30]

Shuffling of data

```
import random
ls = [10, 20, 30, 40, 50]
random.shuffle(ls)
print(ls)
```

Output: [40, 20, 10, 50, 30]

Randomization of data

```
import random
random.randint(1, 100) # The interval is [1, 100)
```

Output: 15

```
import random
random.random() # The interval is [0.0, 1.0)
```

Output: 0.4289859005273219

Random sampling with numpy

```
import numpy as np
np.random.randint(1, 100) # From uniform distribution
```

Output: 97

The interval is [1, 100)

Random sampling with numpy

```
import numpy as np
np.random.randint(1, 100) # From uniform distribution
```

Output: 97 The interval is [1, 100)

```
import numpy as np
np.random.random(5) # From uniform distribution
```

Output: array([0.37076725, 0.10547203, 0.21298417, 0.96296838, 0.15390583]) The interval is [0.0, 1.0)

Random sampling with numpy

```
import numpy as np
np.random.randint(1, 100) # From uniform distribution
```

Output: 97 The interval is [1, 100)

```
import numpy as np
np.random.random(5) # From uniform distribution
```

Output: array([0.37076725, 0.10547203, 0.21298417, 0.96296838, 0.15390583]) The interval is [0.0, 1.0)

Note: Unlike the random() function in random module, the one in the random submodule of numpy module optionally takes size of the array.

Random sampling with numpy

```
import numpy as np
np.random.randn(5) # From normal distribution
```

Output: array([1.62029576, 0.29112406, 1.21198839, 0.26851418, -0.46712281])

```
import numpy as np
np.random.randn(3, 3) # From normal distribution
```

Output: array([[1.13217437, 0.56093212, -2.36792091], [1.42652606, 0.68953983, 0.521175], [1.36766496, 0.9488446 , -1.10042531]])

Effect of the seed value on random sampling

Code 1:

```
import numpy as np
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
```

Effect of the seed value on random sampling

Code 1:

```
import numpy as np
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
```

Code 2:

```
import numpy as np
np.random.seed(20)
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
```

Effect of the seed value on random sampling

Code 1:

```
import numpy as np
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
```

Code 2:

```
import numpy as np
np.random.seed(20)
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
print(np.random.randint(1, 100))
```

Note: Unlike Code 1, Code 2 will generate the same set of random values whenever executed.

Drawing samples from different distributions using `numpy`

beta(a, b[, size])	– Samples from Beta distribution
binomial(n, p[, size])	– Samples from binomial distribution
chisquare(df[, size])	– Samples from chi-square distribution
dirichlet(alpha[, size])	– Samples from Dirichlet distribution
exponential([scale, size])	– Samples from exponential distribution
f(dfnum, dfden[, size])	– Samples from F distribution
gamma(shape[, scale, size])	– Samples from Gamma distribution
geometric(p[, size])	– Samples from geometric distribution
gumbel([loc, scale, size])	– Samples from Gumbel distribution

Table: Functions in `random` submodule of `numpy`

Drawing samples from different distributions using `numpy`

hypergeometric(ngood, nbad, nsample[, size])	– Samples from hypergeometric distribution
laplace([loc, scale, size])	– Samples from Laplace distribution
logistic([loc, scale, size])	– Samples from logistic distribution
lognormal([mean, sigma, size])	– Samples from lognormal distribution
logseries(p[, size])	– Samples from logarithmic series distribution

Table: Functions in `random` submodule of `numpy`

Drawing samples from different distributions using `numpy`

multinomial(n, pvals[, size])	– Samples from multinomial distribution
multivariate_normal(mean, cov[, size, ...)	– Samples from multivariate normal distribution
negative_binomial(n, p[, size])	– Samples from negative binomial distribution
noncentral_chisquare(df, nonc[, size])	– Samples from noncentral chi-square distribution
noncentral_f(dfnum, dfden, nonc[, size])	– Samples from noncentral F distribution
normal([loc, scale, size])	– Samples from normal distribution

Table: Functions in `random` submodule of `numpy`

Drawing samples from different distributions using `numpy`

pareto(a[, size])	– Samples from pareto distribution
poisson([lam, size])	– Samples from Poisson distribution
power(a[, size])	– Samples from power distribution
rayleigh([scale, size])	– Samples from Rayleigh distribution
standard_cauchy([size])	– Samples from standard Cauchy distribution
standard_exponential([size])	– Samples from standard exponetial distribution
standard_gamma(shape[, size])	– Samples from standard Gamma distribution
standard_normal([size])	– Samples from standard normal distribution

Table: Functions in `random` submodule of `numpy`

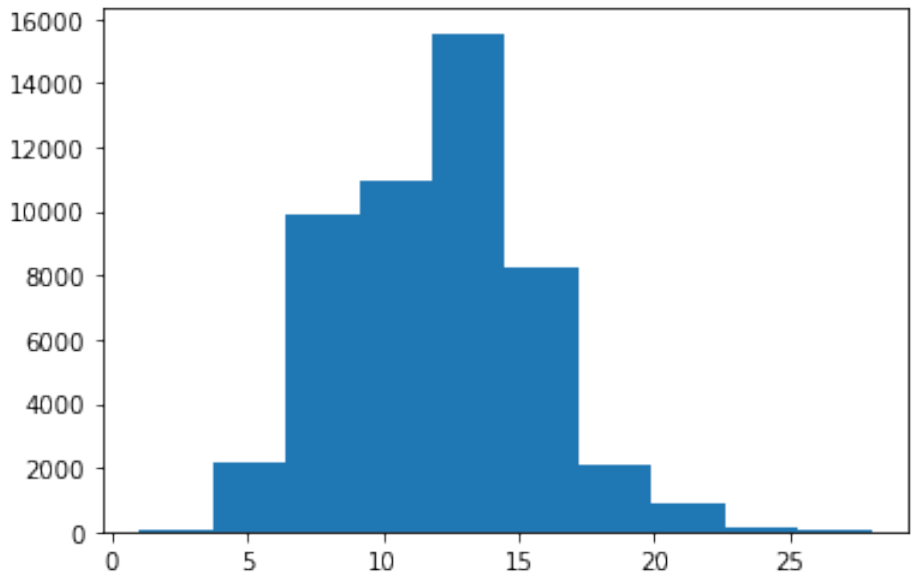
Drawing samples from different distributions using `numpy`

standard_t(df[, size])	– Samples from standard Student's t distribution
triangular(left, mode, right[, size])	– Samples from triangular distribution
uniform([low, high, size])	– Samples from uniform distribution
vonmises(mu, kappa[, size])	– Samples from von Mises distribution
wald(mean, scale[, size])	– Samples from Wald distribution
weibull(a[, size])	– Samples from Weibull distribution
zipf(a[, size])	– Samples from Zipf distribution

Table: Functions in `random` submodule of `numpy`

Visualizing a distribution

```
import numpy as np
sample = np.random.poisson(10, 50000)
import matplotlib.pyplot as plt
plt.hist(sample, 10)
plt.show()
```



Basics of `Series`

`Series` is a one-dimensional and heterogeneous data structure with labeled entries. It is capable of holding any kind of data type (integers, strings, real values, objects, etc.).

Note: The labels are denoted as index.

Using a Series

```
import pandas as pd
pd.Series(Data, index=<Labels>)
```

Using a Series

```
import pandas as pd
pd.Series(Data, index=<Labels>)
```

The Data can be any of the following:

- A scalar value (integer, character, string, etc.)
- An ndarray
- A dictionary

Using a Series

```
import pandas as pd
pd.Series(Data, index=<Labels>)
```

The Data can be any of the following:

- A scalar value (integer, character, string, etc.)
- An ndarray
- A dictionary

Note: If no <Labels> are passed, one will be created having values [0, ..., len(Data) - 1].

Defining Series with a scalar value

```
import pandas as pd
s = pd.Series(10., index=['r1', 'r2', 'r3', 'r4', 'r5'])
print(s)
```

Defining Series with a scalar value

```
import pandas as pd
s = pd.Series(10., index=['r1', 'r2', 'r3', 'r4', 'r5'])
print(s)
```

Output:

```
r1    10.0
r2    10.0
r3    10.0
r4    10.0
r5    10.0
dtype: float64
```

Defining Series with an ndarray

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

Defining Series with an ndarray

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

Output:

```
a    1
b    2
c    3
d    4
dtype: int64
```

Defining Series with an ndarray

```
import pandas as pd
s = pd.Series(['malay', 'amitava', 'utpal'])
print(s)
```

Defining Series with an ndarray

```
import pandas as pd
s = pd.Series(['malay', 'amitava', 'utpal'])
print(s)
```

Output:

```
0      malay
1    amitava
2      utpal
dtype: object
```

Defining Series with an ndarray

```
import numpy as np
import pandas as pd
s = pd.Series(np.random.randn(5))
print(s)
```

Defining Series with an ndarray

```
import numpy as np
import pandas as pd
s = pd.Series(np.random.randn(5))
print(s)
```

Output:

```
0   -0.594763
1   -0.410549
2    1.153695
3    0.454634
4   -0.665784
dtype: float64
```

Defining Series with a dictionary

```
import pandas as pd
s = pd.Series({'name': 'MB', 'age': 37, 'gender': 'M'})
print(s)
```

Defining Series with a dictionary

```
import pandas as pd
s = pd.Series({'name': 'MB', 'age': 37, 'gender': 'M'})
print(s)
```

Output:

```
name       MB
age        37
gender      M
dtype: object
```

Pulling data from a Series

```
import pandas as pd
s = pd.Series({'name': 'MB', 'age': 37, 'gender': 'M'})
print(s)
s0 = pd.Series(s, index=['name', 'gender', 'height'])
print(s0)
```

Pulling data from a Series

```
import pandas as pd
s = pd.Series({'name': 'MB', 'age': 37, 'gender': 'M'})
print(s)
s0 = pd.Series(s, index=['name', 'gender', 'height'])
print(s0)
```

Output:

```
name       MB
age        37
gender      M
dtype: object
name       MB
gender      M
height    NaN
dtype: object
```

Pulling data from a Series

```
import pandas as pd
s = pd.Series({'name': 'MB', 'age': 37, 'gender': 'M'})
print(s[1])
print(s[1:2])
print(s[[2, 1]])
```

Pulling data from a Series

```
import pandas as pd
s = pd.Series({'name': 'MB', 'age': 37, 'gender': 'M'})
print(s[1])
print(s[1:2])
print(s[[2, 1]])
```

Output:

```
37
age    37
dtype: object
gender     M
age       37
dtype: object
```

Searching in a Series

```
import pandas as pd
s = pd.Series({'name': 'MB', 'age': 37, 'gender': 'M'})
print(s['name'])
print('age' in s)
print('height' in s)
print('MB' in s) # Searching for the index 'MB'
```

Searching in a Series

```
import pandas as pd
s = pd.Series({'name': 'MB', 'age': 37, 'gender': 'M'})
print(s['name'])
print('age' in s)
print('height' in s)
print('MB' in s) # Searching for the index 'MB'
```

Output:

```
MB
True
False
False
```

Basics of `DataFrame`

`DataFrame` is a two-dimensional, heterogeneous and size-mutable data structure with entries labeled by rows and columns.

Note: The labels are denoted as index.

Using a DataFrame

```
import pandas as pd
pd.DataFrame(Data, index=<Labels>)
```

Using a DataFrame

```
import pandas as pd
pd.DataFrame(Data, index=<Labels>)
```

A DataFrame can have any of the following inputs:

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

Using a DataFrame

```
import pandas as pd
pd.DataFrame(Data, index=<Labels>)
```

A DataFrame can have any of the following inputs:

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

Note: If no <Labels> are passed, one will be created having values [0, ..., len(Data) - 1].

Defining DataFrame with a dictionary of 1D ndarrays/lists

```
import pandas as pd
s = pd.DataFrame({'col1': [1, 2, 3, 4], 'col2': [5, 6,
7, 8]}, index=['a', 'b', 'c', 'd'])
print(s)
```

Defining DataFrame with a dictionary of 1D ndarrays/lists

```
import pandas as pd
s = pd.DataFrame({'col1': [1, 2, 3, 4], 'col2': [5, 6,
7, 8]}, index=['a', 'b', 'c', 'd'])
print(s)
```

Output:

```
   col1 col2
a     1    5
b     2    6
c     3    7
d     4    8
dtype: int64
```

Defining DataFrame with a dictionary of dicts/series

```
import pandas as pd
s = pd.DataFrame({'col1': pd.Series([.1, .2, .3, .4,
.5], index=['a', 'b', 'c', 'd', 'e']), 'col2':
pd.Series([1., 2., 3.], index=['a', 'b', 'd'])})
print(s)
```

Defining DataFrame with a dictionary of dicts/series

```
import pandas as pd
s = pd.DataFrame({'col1': pd.Series([.1, .2, .3, .4,
.5], index=['a', 'b', 'c', 'd', 'e']), 'col2':
pd.Series([1., 2., 3.], index=['a', 'b', 'd'])})
print(s)
```

Output:

```
   col1  col2
a   0.1   1.0
b   0.2   2.0
c   0.3   NaN
d   0.4   3.0
e   0.5   NaN
```

Defining DataFrame with a structured or record ndarray

```
import numpy as np
import pandas as pd
data = np.zeros((2, ), dtype=[('Year', 'i4'),
('Company', 'a12'), ('Share/Mutual Fund', 'f4')])
print(data)
data[:] = [(1907, 'TATA STEEL', 344.00), (2019, "TATA
DIGITAL INDIA FUND", '19.14')]
s = pd.DataFrame(data)
print(s)
```

Defining DataFrame with a structured or record ndarray

```
import numpy as np
import pandas as pd
data = np.zeros((2, ), dtype=[('Year', 'i4'),
('Company', 'a12'), ('Share/Mutual Fund', 'f4')])
print(data)
data[:] = [(1907, 'TATA STEEL', 344.00), (2019, "TATA
DIGITAL INDIA FUND", '19.14')]
s = pd.DataFrame(data)
print(s)
```

Output:

```
[(0, b'', 0.) (0, b'', 0.)]
   Year          Company  Share/Mutual Fund
0  1907    b'TATA STEEL'         344.000000
1  2019  b'TATA DIGITAL'          19.139999
```

Problems – Day 4

1. Write a function `uniquify` that
   - takes two arguments: an array `A` containing at most 100 non-negative integers, and n, the number of integers contained in the array, and
   - stores in `A`, a list of the distinct integers contained in `A` if the integers in `A` are sorted in increasing order, and returns the number of distinct integers in `A`;
   - returns -1 otherwise.
2. Suppose that we choose to represent sets by arrays of non-negative integers, containing at most 100 elements each. Let A and B be 2 such sets of integers. Implement the following set operations:
   - $A \cup B$;
   - $A \cap B$;
   - $A \subseteq B$ (returns 1 if A is a subset of B, 0 otherwise);
   - prints the power set of A on the screen.

Problems – Day 4

3. Write a program that will print '0' with a probability of 0.5 and '1' in rest of the cases. Now write a function `RUN` to count the number of runs in the binary stream generated with your program. The number of runs will simply explore and count dichotomous (reflecting contrast between two things) sequence of values.
   Note: The following sequence of 0's and 1's has a run count of nine.
   0 0 1 1 0 1 0 0 0 0 1 0 0 1 1 1 1 1 0 0
4. Write a program to find out the summation of the following series given n as the user input.

$$\frac{1}{1} + \frac{11}{12} + \frac{111}{123} + \cdots + \frac{111 \ldots n \text{ times}}{123 \ldots n}$$