Lecture 2
Basic Data Science

Phayung Meesad, Ph.D.
King Mongkut's University of Technology
North Bangkok (KMUTNB)
Bangkok Thailand

Outlines

- Google Collaboratory
- Pandas
- Matplotlib
- Seaborn
- Plotly
- Scikt-Learn
- Tensors
- PyTorch
- TensorFlow

Google Collaboratory

- Google Colaboratory, also known as Colab, is a free, cloud-based platform that allows you to run Python code in your browser. It's a great way to learn Python, experiment with machine learning, and collaborate with others on projects.
- Free and cloud-based: You don't need to install anything on your computer to use Colab. Just open a browser and go to the Colab website.
- Runs Python code: Colab supports Python 2 and 3, and you can use all of the popular Python libraries.
- GPU and TPU support: Colab notebooks can run on Google's powerful GPUs and TPUs, which can speed up your code significantly.
- Collaborative: You can share your Colab notebooks with others, and they can collaborate on the same notebook in real time.

Google Collaboratory

- To get started with Google Colab, you can follow these steps:
  1. Go to the Colab website: https://colab.research.google.com/.
  2. Click the "New Notebook" button.
  3. Select the "Python 3" runtime.
  4. Start writing Python code in the notebook.

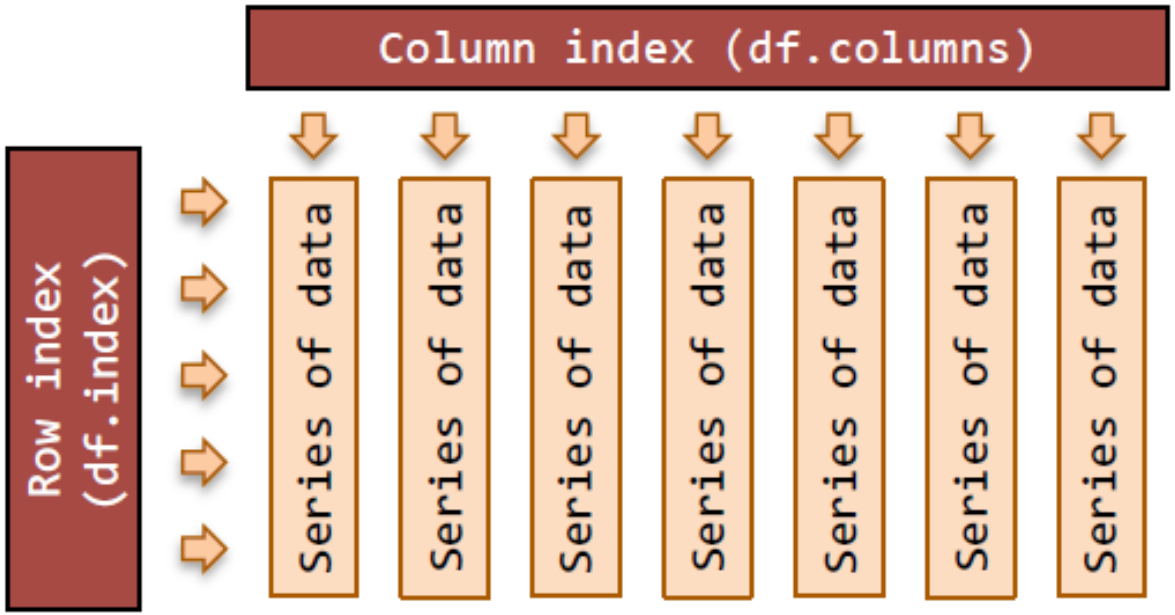
Python

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

    print(factorial(5))
```

Pandas DataFrame

- **DataFrame object:** The pandas DataFrame is a two dimensional table of data with column and row indexes. The columns are made up of pandas Series objects.
- **Series object:** an ordered, one-dimensional array of data with an index. All the data in a Series is of the same data type. Series arithmetic is vectorised after first aligning the Series index for each of the operands.



Pandas DataFrame

- **The index object:** The pandas Index provides the axis labels for the Series and DataFrame objects. It can only contain hashable objects. A pandas Series has one Index; and a DataFrame has two Indexes.

```
# import modules
import pandas as pd
from pandas import DataFrame, Series


# --- get Index from Series and DataFrame
idx = s.index
idx = df.columns # the column index
idx = df.index # the row index
```

Pandas DataFrame

```
# --- some Index attributes
b = idx.is_monotonic_decreasing
b = idx.is_monotonic_increasing
b = idx.has_duplicates
i = idx.nlevels # multi-level indexes
# --- some Index methods
a = idx.values() # get as numpy array
l = idx.tolist() # get as a python list
idx = idx.astype(dtype)# change data type
b = idx.equals(o) # check for equality
idx = idx.union(o) # union of two indexes
i = idx.nunique() # number unique labels
label = idx.min() # minimum label
label = idx.max() # maximum label
```

Load Data

```
import pandas as pd

# load iris data
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
attributes = ['SepalLength','SepalWidth','PetalLength','PetalWidth','Class']
iris = pd.read_csv(url, sep=',', header=None, names=attributes,
index_col=None, )
```

Data Exploration

```
# Statistics and Visualization
# shape
print(iris.shape)

# head & tail
print(iris.head())
print(iris.tail())

# info & description
print(iris.info())
print(iris.describe())

# class distribution
print(iris.groupby('Class').size())
```

Matplotlib

```
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt

# Scatter Matrix Plot
scatter_matrix(iris)
plt.show()

# box plot
iris.plot(kind='box', subplots=False, layout=(3,2), sharex=False, sharey=False)
plt.show()

# histograms
iris.hist()
plt.show()
```

Seaborn

```
import seaborn as sns
sns.set(style="white", color_codes=True);

# Scatter pair plot with histogram in the diagonal
sns.pairplot(iris, hue="Class", diag_kind="hist");

# Scatter pair plot with kde in the diagonal
sns.pairplot(iris, hue="Class", diag_kind="kde");

# Another multivariate visualization technique pandas has is parallel_coordinates
# Parallel coordinates plots each feature on a separate column & then draws lines
```

Pandas Plotting

```
# connecting the features for each data sample
from pandas.plotting import parallel_coordinates
parallel_coordinates(iris, "Class")

# Andrews Curves involve using attributes of samples as coefficients for Fourier series
# and then plotting these
from pandas.plotting import andrews_curves
andrews_curves(iris, "Class")
```

Plotly

```
# x and y given as array_like objects
import plotly.express as px
fig = px.scatter(x=[0, 1, 2, 3, 4], y=[0, 1, 4, 9, 16])
fig.show();
```

Plotly Express

```
# x and y given as DataFrame columns
import plotly.express as px
df = px.data.iris() # iris is a pandas DataFrame
fig = px.scatter(df, x="sepal_width",
y="sepal_length")
fig.show();
```

Plotly Express

```
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x="sepal_width",
y="sepal_length", color="species",
                 size='petal_length',
hover_data=['petal_width'])
fig.show();
```

Plotly Express

```
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x="sepal_width",
y="sepal_length", color='petal_length')
fig.show();
```

Plotly Express

```
import plotly.express as px
df = px.data.iris()
fig = px.scatter(df, x="sepal_width",
y="sepal_length", color="species",
symbol="species")
fig.show()
```

Plotly Graph Objects

```
import plotly.graph_objects as go
import numpy as np
N = 100000
fig = go.Figure(data=go.Scattergl(
    x = np.random.randn(N),
    y = np.random.randn(N),
    mode='markers',
    marker=dict(
        color=np.random.randn(N),
        colorscale='Viridis',
        line_width=1
    )
))
fig.show()
```

Scikit-Learn

- Scikit-learn is a free and open-source machine learning library for the Python programming language.
- It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN.
- It is built on top of the NumPy and SciPy libraries, and is designed to be easy to use and efficient.
- Scikit-learn is a popular library for machine learning in Python, and is used by a wide range of people, from students and researchers to data scientists and engineers.
- It is well-documented and has a large community of users and contributors.

Some of the features of Scikit-learn

- **A wide range of algorithms:** Scikit-learn includes a wide range of machine learning algorithms, including both supervised and unsupervised learning algorithms.
- **Easy to use:** Scikit-learn is designed to be easy to use, with a consistent and intuitive API.
- **Efficient**: Scikit-learn is designed to be efficient, and can handle large datasets.
- **Well-documented:** Scikit-learn is well-documented, with extensive tutorials and documentation.
- **Active community:** Scikit-learn has a large and active community of users and contributors.

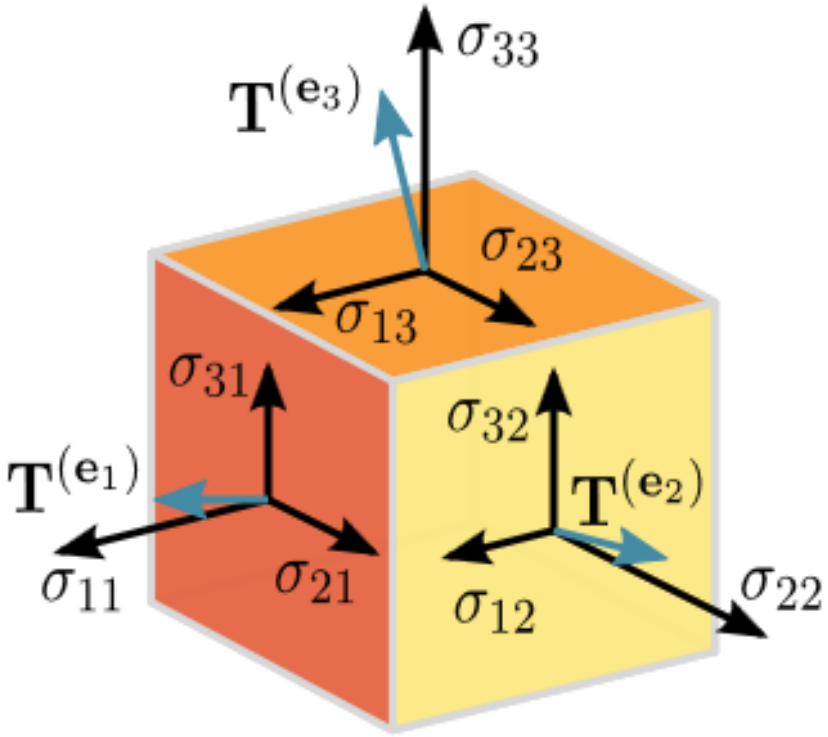
Scikit-learn can be used for

- **Classification:** This is the task of predicting a categorical outcome, such as whether a customer will click on an ad or not.
- **Regression:** This is the task of predicting a continuous outcome, such as the price of a house or the number of sales.
- **Clustering:** This is the task of grouping similar data points together.
- **Dimensionality reduction**: This is the task of reducing the number of features in a dataset.
- **Model selection:** This is the task of choosing the best machine learning model for a given dataset.

Tensors

- Tensors are multilinear maps from vector spaces to the real numbers.
- A tensor can be represented as a multidimensional array of numbers.
- A tensor is a geometric object mapping in a multi-linear manner geometric vectors, scalars, and other tensors to a resulting tensor.
- Vectors and scalars often used in physics and engineering applications are the simplest tensors.
- Vectors from the dual space of the vector space supplying the geometric vectors are also tensors.

Tensors Data Structure



- Tensors are geometric objects with a shape, rank, and type, used to hold a multidimensional array.

PyTorch

- Torch is an open-source machine learning library, a scientific computing framework, and a script language based on the Lua programming language. It provides a wide range of algorithms for deep learning, and uses the scripting language LuaJIT, and an underlying C implementation. It was created at IDIAP at EPFL. As of 2018, Torch is no longer in active development.
- However PyTorch, which is based on the Torch library, is actively developed as of May 2022.
- PyTorch is an open source machine learning framework based on the Torch library used for applications such as computer vision and natural language processing, primarily developed by Meta AI.
- It is free and open-source software released under the Modified BSD license. Although the Python interface is more polished and the primary focus of development,

Create Numpy Array

```
arr = np.array([1,2,3,4,5])
print(arr)
print(arr.dtype)
print(type(arr))
```

Create Torch Tensor

```
x = torch.from_numpy(arr)
# Equivalent to x = torch.as_tensor(arr)
print(x)
# Print the type of data held by the tensor
print(x.dtype)
# Print the tensor object type
print(type(x))
print(x.type()) # this is more specific!
```

Tensor Datatypes

TYPE	NAME	EQUIVALENT	TENSOR TYPE
32-bit integer (signed)	torch.int32	torch.int	IntTensor
64-bit integer (signed)	torch.int64	torch.long	LongTensor
16-bit integer (signed)	torch.int16	torch.short	ShortTensor
32-bit floating point	torch.float32	torch.float	FloatTensor
64-bit floating point	torch.float64	torch.double	DoubleTensor
16-bit floating point	torch.float16	torch.half	HalfTensor
8-bit integer (signed)	torch.int8		CharTensor
8-bit integer (unsigned)	torch.uint8		ByteTensor

```
arr2 = np.arange(0.,12.).reshape(4,3)
print(arr2)

x2 = torch.from_numpy(arr2)
print(x2)
print(x2.type())
```

Copying vs. sharing

- torch.from_numpy()
- torch.as_tensor()
- torch.tensor()
- There are a number of different functions available for creating tensors.
- When using torch.from_numpy() and torch.as_tensor(), the PyTorch tensor and the source NumPy array share the same memory.
- This means that changes to one affect the other.
- However, the torch.tensor() function always makes a copy.

```
# Using torch.from_numpy()
arr = np.arange(0,5)
t = torch.from_numpy(arr)
print(t)
arr[2]=77
print(t)

# Using torch.tensor()
arr = np.arange(0,5)
t = torch.tensor(arr)
print(t)
arr[2]=77
print(t)
```

Class constructors

- torch.Tensor()
- torch.FloatTensor()
- torch.LongTensor(), etc.

- There's a subtle difference between using the factory function torch.tensor(data) and the class constructor torch.Tensor(data).
- The factory function determines the dtype from the incoming data, or from a passed-in dtype argument.
- The class constructor torch.Tensor() is simply an alias for torch.FloatTensor(data).

```
data = np.array([1,2,3])
a = torch.Tensor(data)  # Equivalent to cc = torch.FloatTensor(data)
print(a, a.type())
b = torch.tensor(data)
print(b, b.type())
c = torch.tensor(data, dtype=torch.long)
print(c, c.type())
```

Creating tensors

- Uninitialized tensors with .empty()
- torch.empty() returns an uninitialized tensor. Essentially a block of memory is allocated according to the size of the tensor, and any values already sitting in the block are returned. This is similar to the behavior of numpy.empty().

```
x = torch.empty(4, 3)
print(x)
```

- Initialized tensors with .zeros() and .ones()
- torch.zeros(size)
- torch.ones(size)
- It's a good idea to pass in the intended dtype.

```
x = torch.zeros(4, 3, dtype=torch.int64)
print(x)
```

Tensors from ranges

- torch.arange(start,end,step)
- torch.linspace(start,end,steps)
- Note that with .arange(), end is exclusive, while with linspace(), end is inclusive.

```
x = torch.arange(0,18,2).reshape(3,3)
print(x)
x = torch.linspace(0,18,12).reshape(3,4)
print(x)
```

Tensors from data

- torch.tensor() will choose the dtype based on incoming data:

```
x = torch.tensor([1, 2, 3, 4])
print(x)
print(x.dtype)
print(x.type())
x = torch.FloatTensor([5,6,7])
print(x)
print(x.dtype)
print(x.type())
x = torch.FloatTensor([5,6,7])
print(x)
print(x.dtype)
print(x.type())
```

Changing the dtype of existing tensors

- Don't be tempted to use x = torch.tensor(x, dtype=torch.type) as it will raise an error about improper use of tensor cloning.
- Instead, use the tensor .type() method.

```
print('Old:', x.type())
x = x.type(torch.int64)
print('New:', x.type())
```

Random number tensors

- torch.rand(size) returns random samples from a uniform distribution over [0, 1)
- torch.randn(size) returns samples from the "standard normal" distribution [σ = 1]
- Unlike rand which is uniform, values closer to zero are more likely to appear.
- torch.randint(low,high,size) returns random integers from low (inclusive) to high (exclusive)

```
x = torch.rand(4, 3)
print(x)
x = torch.randn(4, 3)
print(x)
x = torch.randint(0, 5, (4, 3))
print(x)
```

Random number tensors

- torch.rand_like(input)
- torch.randn_like(input)
- torch.randint_like(input,low,high)
- these return random number tensors with the same size as input

```
x = torch.zeros(2,5)
print(x)
x2 = torch.randn_like(x)
print(x2)
x3 = torch.ones_like(x2)
print(x3)
```

Setting the random seed

- torch.manual_seed(int) is used to obtain reproducible results

```
torch.manual_seed(42)
x = torch.rand(2, 3)
print(x)
torch.manual_seed(42)
x = torch.rand(2, 3)
print(x)
```

Tensor attributes

- Besides dtype, we can look at other tensor attributes like shape, device and layout

```
x.shape
x.size()  # equivalent to x.shape
```

- PyTorch supports use of multiple devices, harnessing the power of one or more GPUs in addition to the CPU.

x.device

TensorFlow

- TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.
- TensorFlow was developed by the Google Brain team for internal Google use in research and production. The initial version was released under the Apache License 2.0 in 2015.
- Google released the updated version of TensorFlow, named TensorFlow 2.0, in September 2019.
- TensorFlow can be used in a wide variety of programming languages, most notably Python, as well as Javascript, C++, and Java. This flexibility lends itself to a range of applications in many different sectors.

Numpy & TensorFlow

Numpy	TensorFlow
a=np.zeros((2,2));b=np.ones((2,2))	a=tf.zeros((2,2)),b=tf.ones((2,2))
np.sum(b,axis=1)	tf.reduce_sum(a,reduction_indices=[1])
a.shape	a.get_shape()
np.reshape(a,(1,4))	tf.reshape(a,(1,4))
b*5+1	b*5+1
np.dot(a,b)	tf.matmul(a,b)
a[0,0], a[:,0], a[0,:]	a[0,0],a[:,0],a[0,:]

Numpy

```
In [1]: import numpy as np

In [2]: a = np.zeros((2,2));

In [3]: b = np.ones((2,2))

In [4]: print(np.sum(b, axis=1))
        [2. 2.]

In [5]: print(a.shape)
        (2, 2)

In [6]: print(np.reshape(a, (1,4)))
        [[0. 0. 0. 0.]]
```

Tensorflow

```
import tensorflow as tf
a = tf.zeros((2,2));
b = tf.ones((2,2))
print(tf.reduce_sum(b,
reduction_indices=1).eval())
print(a.get_shape())
print(tf.reshape(a, (1, 4)).eval())
```

Rank and Shape

- Rank describes each tensor. It identifies the number of dimensions of the tensor.
- Rank is the order or n-dimensions of a tensor.
- Scalars have rank zero.
- Vectors have rank one.
- Matrixes have rank two.

Rank and Shape

```
In [1]: import tensorflow as tf

In [2]: scalar = tf.constant(100)

In [3]: vector = tf.constant([1.,2.])

In [4]: matrix = tf.constant([[1.,2.],[3.,4.]])

In [5]: cube_matrix = tf.constant([[[1.,1.,1.],[2.,2.,2.],[3.,3.,3.]],
                                   [[4.,4.,4.],[5.,5.,5.],[6.,6.,6.]],
                                   [[7.,7.,7.],[8.,8.,8.],[9.,9.,9.]]])
```

Rank and Shape

```
In [6]: print(scalar.get_shape())
()

In [7]: print(vector.get_shape())
(2,)

In [8]: print(matrix.get_shape())
(2, 2)

In [9]: print(cube_matrix.get_shape())
(3, 3, 3)
```