
METU Computer Engineering

# CEng 240 – Spring 2021
# Week 14

## Scientific and Engineering Libraries

## Part 2: Pandas and Matplotlib

Sinan Kalkan



# This Week

- Scientific and Engineering Libraries
  - Pandas for data handling and analysis
  - Matplotlib for plotting

# Administrative Notes

- Lab 9
- ~~Midterm: 1 June, Tuesday, 17:40~~
- Final: 8 July, 9:30

METU Computer Engineering


pandas

# Outline

- Overview
- Installation
- DataFrames
- Accessing data in DataFrames
- Modifying data in DataFrames
- Analyzing data in DataFrames
- Presenting data in DataFrames

METU Computer Engineering



# Overview

pandas

- A handy library for:
  - working with files of different formats
  - manipulating & analyzing data
- Data types & structures for
  - tables, especially numerical tables,
  - time series
- Name comes from “panel data”



# Installation

pandas

- On your Linux environment:

  $ pip install pandas

  or

  $ conda install pandas

- On Windows/Mac: install anaconda first
- On Colab, it is already installed
- `import pandas as pd`

# Supported Files

METU Computer Engineering

- A wide collection of file formats
- Each format has a reader and a writer

| Format Type | Data Description | Reader | Writer |
|---|---|---|---|
| text | CSV | read_csv | to_csv |
| text | Fixed-Width Text File | read_fwf | - |
| text | JSON | read_json | to_json |
| text | HTML | read_html | to_html |
| text | Local clipboard | read_clipboard | to_clipboard |
| - | MS Excel | read_excel | to_excel |
| binary | OpenDocument | read_excel | - |
| binary | HDF5 Format | read_hdf | to_hdf |
| binary | Feather Format | read_feather | to_feather |
| binary | Parquet Format | read_parquet | to_parquet |
| binary | ORC Format | read_orc | - |
| binary | Msgpack | read_msgpack | to_msgpack |
| binary | Stata | read_stata | to_stata |
| binary | SAS | read_sas | - |
| binary | SPSS | read_spss | - |
| binary | Python Pickle Format | read_pickle | to_pickle |
| SQL | SQL | read_sql | to_sql |
| SQL | Google BigQuery | read_gbq | to_gbq |

For an up-to-date list: https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html



# Data Frames

pandas

- Similar to NumPy's ndarray datatype, Pandas has a very fundamental data type called `DataFrame`
- A `DataFrame` is created by
  - Data loaded from files (using a reader)
  - The constructor `DataFrame()`

# Data Frames

## Loading data from files

```
# Import the necessary libraries
import pandas as pd

# Read the file named 'ch10_example.csv'
df = pd.read_csv('ch10_example.csv')

# Print the CSV file's contents:
print("The CSV file contains the following:\n", df, "\n")

# Check the types of each column
df.dtypes
```

This produces the following output:

```
The CSV file contains the following:
       Name  Grade  Age
0      Jack   40.2   20
1    Amanda   30.0   25
2      Mary   60.2   19
3      John   85.0   30
4     Susan   70.0   28
5      Bill   58.0   28
6      Jill   90.0   27
7       Tom   90.0   24
8     Jerry   72.0   26
9    George   79.0   22
10   Elaine   82.0   23

Name       object
Grade     float64
Age         int64
dtype: object
```

For more information about the CSV file format, have a look at the File Handling chapter.

Sample file ‘ch10_example.csv’ at:
https://raw.githubusercontent.com/sinankalkan/CENG240/master/figures/ch10_example.csv

# Data Frames

## Loading data from files

```
4 # Import the necessary libraries
5 import pandas as pd
6
7 # Read the file named 'ch10_example.csv'
8 df = pd.read_csv('ch10_example.csv')
9
10 # Print the CSV file's contents:
11 print("The CSV file contains the following:\n", df, "\n")
12
13 # Check the types of each column
14 df.dtypes
```

More on `pd.read_csv()`:

- Automatically loads column headers
- If your file does not have a header, use: `pd.read_csv(filename, header=None)`
- If you want to read specific columns, use:

  `pd.read_csv(filename, usecols=['column name 1', ...])`

- For more information & control, see `help(pd.read_csv)`



# Data Frames

## Create a DataFrame from Python data

Use the `pd.DataFrame()` function:

```
1 lst = [('Jack', 40.2, 20), ('Amanda', 30, 25), ('Mary', 60.2, 19)]
2 df = pd.DataFrame(data = lst, columns=['Name', 'Grade', 'Age'])
3 print(df)
```

```
     Name  Grade  Age
0    Jack   40.2   20
1  Amanda   30.0   25
2    Mary   60.2   19
```

If you need keys/names for each row, then:

```
1 names = ['Jack', 'Amanda', 'Mary']
2 lst = [(40.2, 20), (30, 25), (60.2, 19)]
3 df = pd.DataFrame(data = lst, index=names, columns=['Grade', 'Age'])
4 print(df)
```

```
        Grade  Age
Jack     40.2   20
Amanda   30.0   25
Mary     60.2   19
```

# Data Frames

Create a `DataFrame` from Python data

It is also possible to create the columns of data in a dictionary and pass that to the `pd.DataFrame()` function:

```
1 d = {'Grade': [40.2, 30, 60.2],
2       'Age': [20, 25, 19]}
3 names = ['Jack', 'Amanda', 'Mary']
4 df = pd.DataFrame(data = d, index=names)
5 print(df)
```

```
        Grade  Age
Jack     40.2   20
Amanda   30.0   25
Mary     60.2   19
```

Note that the column names were retrieved from the keys of the dictionary

# Accessing Data

Column-wise access

- Use column names & row names like keys in a dictionary
- `df['Name']` returns the 'Name' column
  - Then you can use integer index or named index (key) in each row

```
    >>> print(df)
            Grade  Age
Jack         40.2   20
Amanda       30.0   25
Mary         60.2   19
    >>> print(df['Grade'][1])
    30.0
    >>> print(df['Grade']['Amanda'])
    30.0
```

# Accessing Data

Row-wise access

- `df.iloc[<row index>`
  - for integer indexes
- `df.loc[<row name>]`
  - for named indexes
- Row & column indexing can be combined:
  - `df.loc['Amanda', 'Grade']`
  - `df.iloc[1, 1]`
- With integer indexes, Python's slicing ([start:end:step]) can be used

```
>>> print(df)
        Grade  Age
Jack     40.2   20
Amanda   30.0   25
Mary     60.2   19
>>> print(df.iloc[1])
Grade    30.0
Age      25.0
Name: Amanda, dtype: float64
>>> print(df.loc['Amanda'])
Grade    30.0
Age      25.0
Name: Amanda, dtype: float64
```



# Modifying Data

pandas

- Modifying data is very easy
- Need to be careful about chained indexing
- No guarantee on `df['Grade']` being a copy or a direct access to the 'Grade' column

```
>>> print(df)
            Grade  Age
Jack         40.2   20
Amanda       30.0   25
Mary         60.2   19
>>> df['Grade']['Amanda'] = 45
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
```



# Modifying Data

pandas

- Specify row & column in one step/go
- Avoid chained indexing when modifying data

```
>>> print(df)
        Grade  Age
Jack     40.2   20
Amanda   30.0   25
Mary     60.2   19
>>> df.loc['Amanda','Grade'] = 45
>>> df.iloc[1,1] = 30
>>> print(df)
        Grade  Age
Jack     40.2   20
Amanda   45.0   30
Mary     60.2   19
```

# Analyzing Data

METU Computer Engineering

- Pandas provides many facilities for analyzing your data in a DataFrame
- `df.describe()`
- `df.value_counts()`
- `df.max()` or `df.min()`
- `df.sort_values(by=<col name>)`
- `df.nlargest(<n>)`

```
1 print(df)
2 df.describe()
```

```
        Grade  Age
Jack     40.2   20
Amanda   30.0   25
Mary     60.2   19
```

| | Grade | Age |
|---|---|---|
| count | 3.000000 | 3.000000 |
| mean | 43.466667 | 21.333333 |
| std | 15.362725 | 3.214550 |
| min | 30.000000 | 19.000000 |
| 25% | 35.100000 | 19.500000 |
| 50% | 40.200000 | 20.000000 |
| 75% | 50.200000 | 22.500000 |
| max | 60.200000 | 25.000000 |

# Analyzing Data

pandas

METU Computer Engineering

```
1 print("Maximum grade is: ", df['Grade'].max())
2 print("\nRecords sorted according to age:\n", df.sort_values(by="Age"))
3 print("\n\nTop two grades are:\n", df['Grade'].nlargest(2))
```

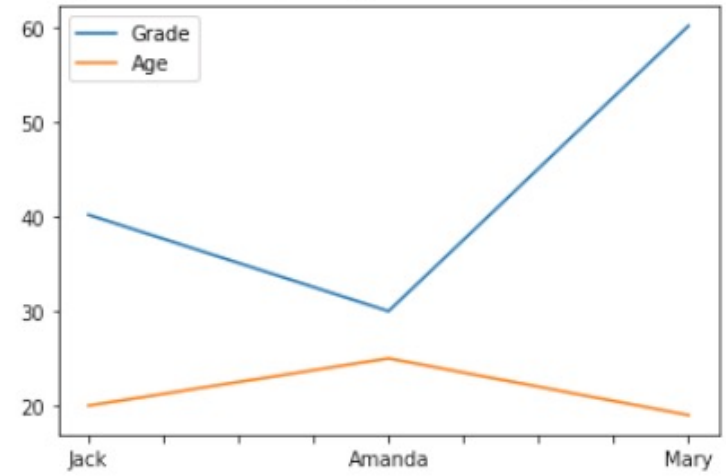
```
Maximum grade is:  60.2

Records sorted according to age:
         Grade  Age
Mary      60.2   19
Jack      40.2   20
Amanda    30.0   25


Top two grades are:
 Mary    60.2
Jack    40.2
Name: Grade, dtype: float64
```

# Presenting Data

- plot() function

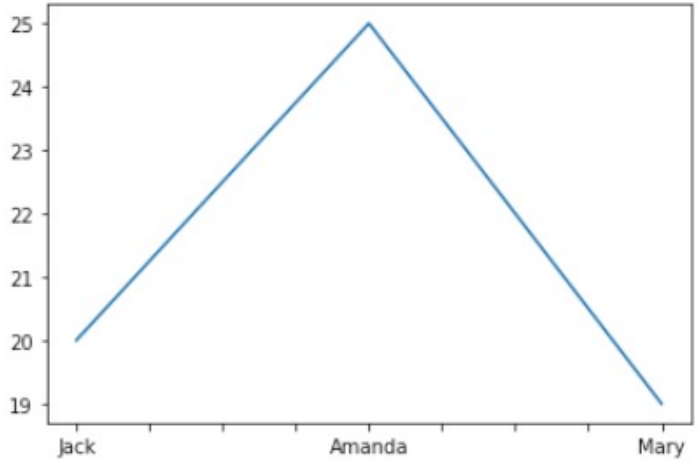
```
1 #Plots all columns in different colours
2 df.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcd35f86c88>
```




```
1 # Plots a single column
2 df['Age'].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcd364ab048>
```




matplotlib

# Outline

matplotlib

- Overview
- Installation
- Anatomy of a figure/plot
- Preparing your data
- Drawing single plots
- Drawing multiple plots
- Changing elements of a plot

METU Computer Engineering

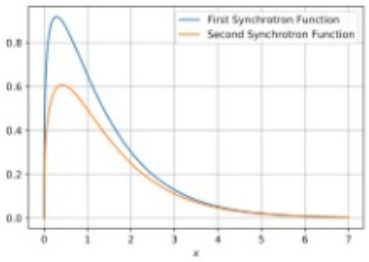
# Overview

matplotlib

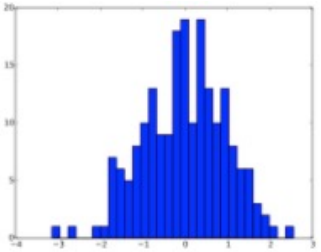
- A drawing library for Python
- A free and open source alternative to Matlab
- Allows 2D & 3D plots

# Overview

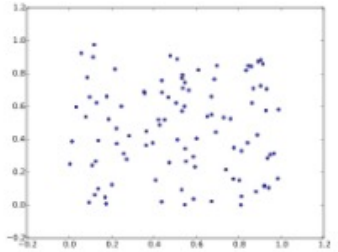
matplotlib



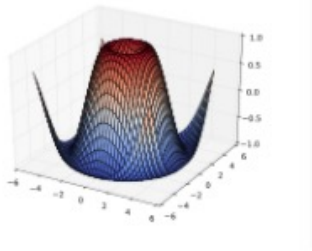

Line plot



Histogram



Scatter plot



3D plot

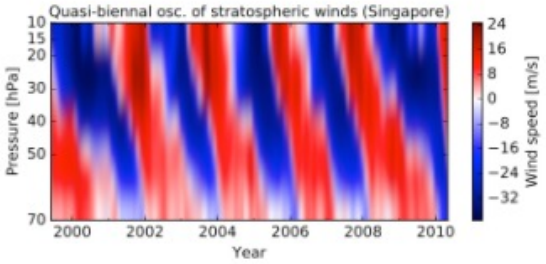
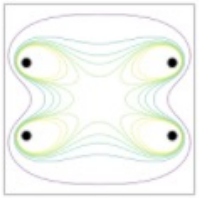
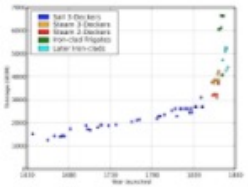



Image plot

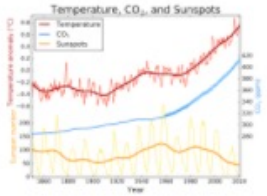


Contour plot

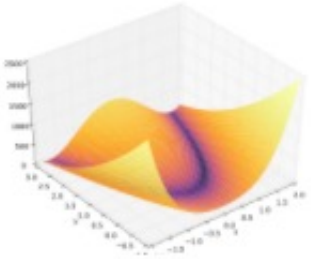


Scatter plot

Polar plot




Line plot



3-D plot

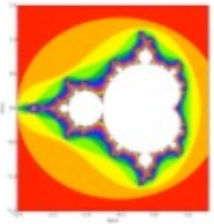


Image plot

# Installation


- On your Linux environment:

  $ pip install matplotlib

  or

  $ conda install matplotlib

- On Windows/Mac: install anaconda first
- On Colab, it is already installed
- `import matplotlib.pyplot as plt`

# Anatomy of a plot

matplotlib

- Canvas / drawing area
  - scatter plot, line plot, ...
- Axes
  - ticks, tick labels, axis labels
- figure title
- legend

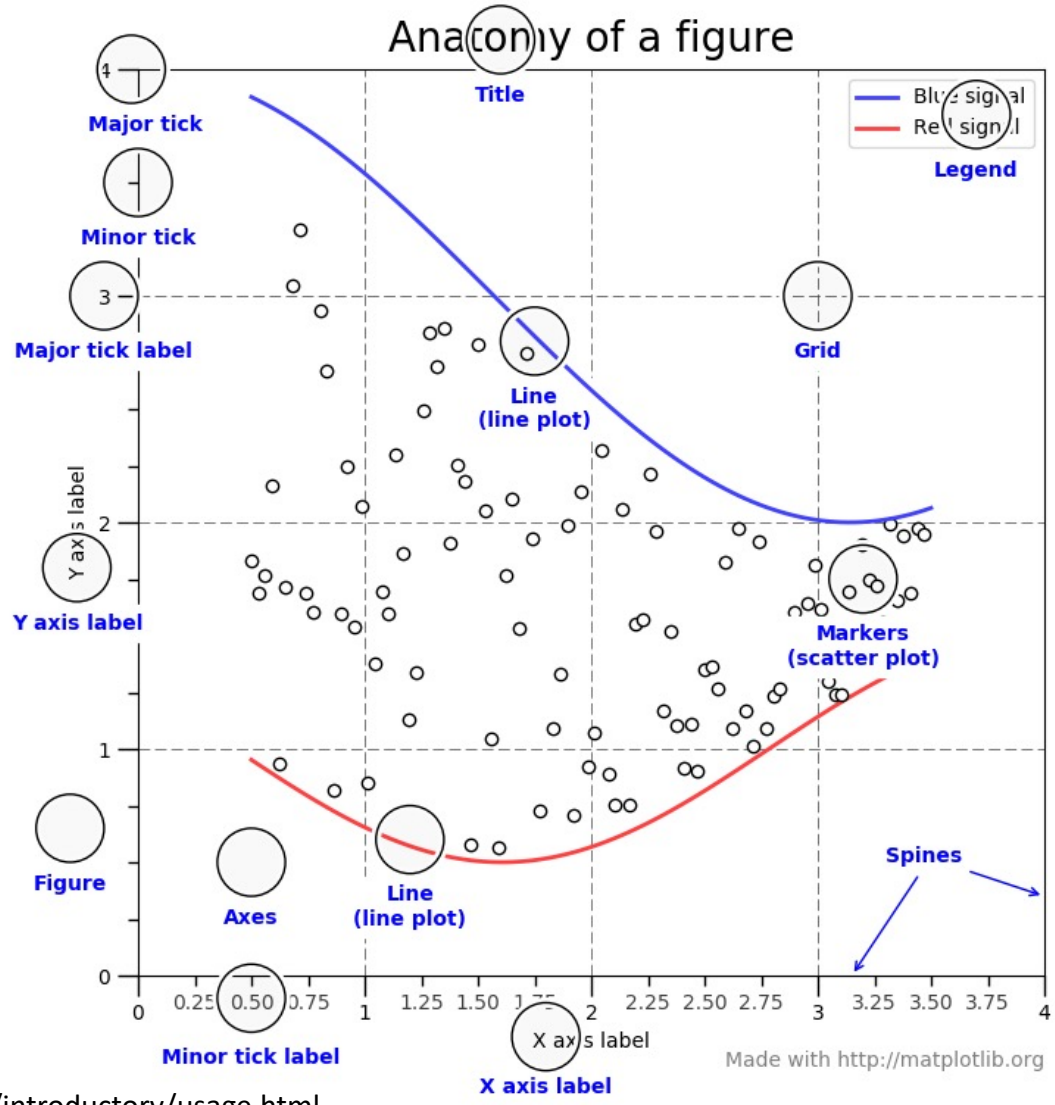



Figure from: https://matplotlib.org/tutorials/introductory/usage.html



METU Computer Engineering

# Preparing your data

matplotlib

- Matplotlib expects NumPy arrays
- Convert your data to NumPy
  - If your data is a Python data type, use `array()` function to do the conversion
  - If your data is a DataFrame, use `df.values`, e.g.:

```
1 print(df)
2 age_array = df['Age'].values
3 print("The `Age` values in an array form are:", age_array)
4 print("The type of our new data is: ", type(age_array))
```

```
         Grade  Age
Jack      40.2   20
Amanda    30.0   25
Mary      60.2   19
The `Age` values in an array form are: [20 25 19]
The type of our new data is:  <class 'numpy.ndarray'>
```

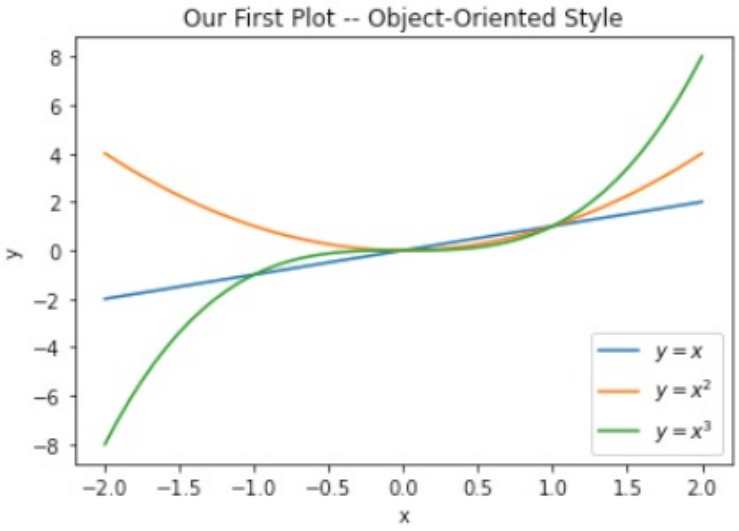
METU Computer Engineering

# Drawing single plots

matplotlib

## Drawing in an Object-Oriented Style

- Create a figure object and axes object
- Use their member functions & variables



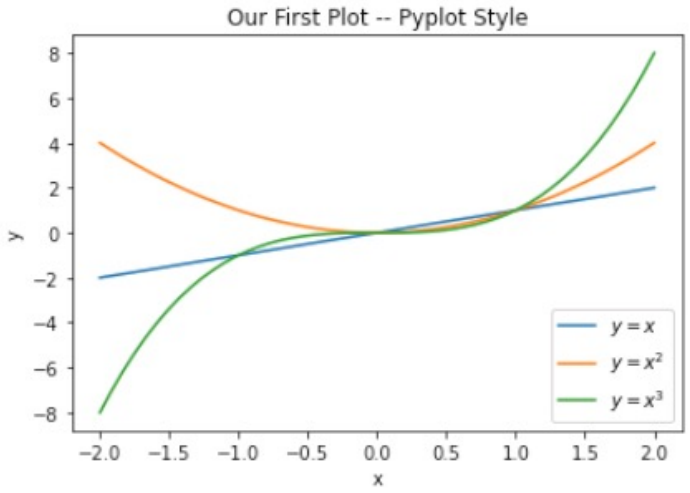

```python
import matplotlib.pyplot as plt
import numpy as np

# Uniformly sample 50 x values between -2 and 2:
x = np.linspace(-2, 2, 50)

# Create an empty figure
fig, ax = plt.subplots()

# Plot y = x
ax.plot(x, x, label='$y=x$')

# Plot y = x^2
ax.plot(x, x**2, label='$y=x^2$')

# Plot y = x^3
ax.plot(x, x**3, label='$y=x^3$')

# Set the labels for x and y axes:
ax.set_xlabel('x')
ax.set_ylabel('y')

# Set the title of the figure
ax.set_title("Our First Plot -- Object-Oriented Style")

# Create a legend
ax.legend()
```

# Drawing single plots

matplotlib

## Drawing in an Pyplot Style

- Use `matplotlib.pylot` directly



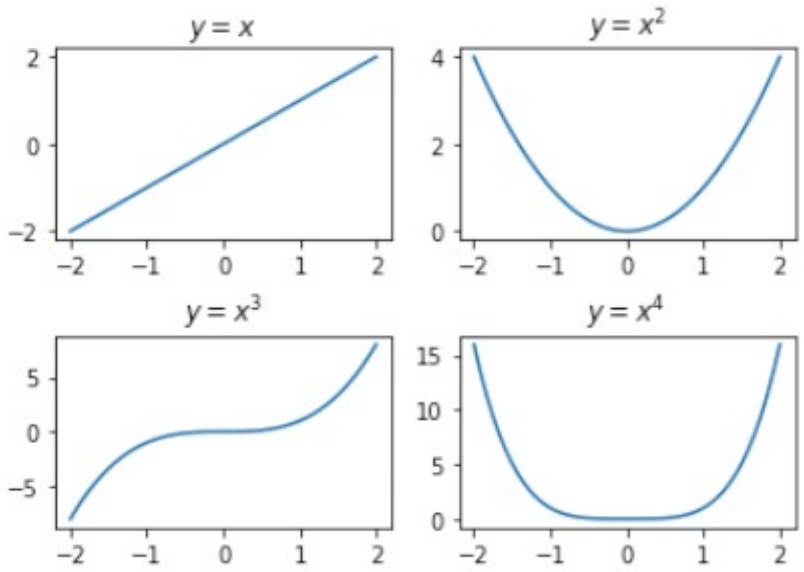

```
1 # Uniformly sample 50 x values between -2 and 2:
2 x = np.linspace(-2, 2, 50)
3
4 # Plot y = x
5 plt.plot(x, x, label='$y=x$')
6
7 # Plot y = x^2
8 plt.plot(x, x**2, label='$y=x^2$')
9
10 # Plot y = x^3
11 plt.plot(x, x**3, label='$y=x^3$')
12
13 # Set the labels for x and y axes:
14 plt.xlabel('x')
15 plt.ylabel('y')
16
17 # Set the title of the figure
18 plt.title("Our First Plot -- Pyplot Style")
19
20 # Create a legend
21 plt.legend()
```

METU Computer Engineering

# Drawing multiple plots

matplotlib

- This example uses the object-oriented approach




```
# Create a 2x2 grid of plots
fig, axes = plt.subplots(2, 2)

# Plot (1,1)
axes[0,0].plot(x, x)
axes[0,0].set_title("$y=x$")

# Plot (1,2)
axes[0,1].plot(x, x**2)
axes[0,1].set_title("$y=x^2$")

# Plot (2,1)
axes[1,0].plot(x, x**3)
axes[1,0].set_title("$y=x^3$")

# Plot (2,2)
axes[1,1].plot(x, x**4)
axes[1,1].set_title("$y=x^4$")

# Adjust vertical space between rows
plt.subplots_adjust(hspace=0.5)
```

# Drawing multiple plots

matplotlib

METU Computer Engineering

Multiple plots PyPlot style

```
# Plot (1,1)
plt.subplot(2, 2, 1)
plt.plot(x, x)
plt.title('$y=x$')

# Plot (1,2)
plt.subplot(2, 2, 2)
plt.plot(x, x**2)
plt.title('$y=x^2$')

# Plot (2,1)
plt.subplot(2, 2, 3)
plt.plot(x, x**3)
plt.title('$y=x^3$')

# Plot (2,2)
plt.subplot(2, 2, 4)
plt.plot(x, x**4)
plt.title('$y=x^4$')

# Adjust vertical space between rows
plt.subplots_adjust(hspace=0.5)
```

Multiple plots OOP style

```
# Create a 2x2 grid of plots
fig, axes = plt.subplots(2, 2)

# Plot (1,1)
axes[0,0].plot(x, x)
axes[0,0].set_title("$y=x$")

# Plot (1,2)
axes[0,1].plot(x, x**2)
axes[0,1].set_title("$y=x^2$")

# Plot (2,1)
axes[1,0].plot(x, x**3)
axes[1,0].set_title("$y=x^3$")

# Plot (2,2)
axes[1,1].plot(x, x**4)
axes[1,1].set_title("$y=x^4$")

# Adjust vertical space between rows
plt.subplots_adjust(hspace=0.5)
```

# Changing plot elements




- All elements of a plot are changeable
  - ticks, tick labels, ...
  - line/dot color, line/dot size, shape, ..
  - legends, titles, ...
  - font style, size, ...
  - Latex support
- See
  - `help(plt.plot)`
  - https://matplotlib.org/2.1.1/contents.html



# Examples (from the book)

- Create a simple CSV file using your favorite spreadsheet editor (e.g. Microsoft Excel or Google Spreadsheets) and create a file with your exams and their grades as two separate columns. Save the file, upload it to the Colab notebook and do the following:
  - Load the file using Pandas.
  - Calculate the mean of your exam grades.
  - Calculate the standard deviation of your grades.
- Using Matplotlib, generate the following plots with suitable names for the axes and the titles.
  - Draw the following four functions in separate single plots: $\sin(x)$,$\cos(x)$,$\tan(x)$,$\cot(x)$.
  - Draw these four functions in a single plot.
  - Draw a multiple 2x2 plot where each subplot is one of the four functions.



# Final Words: Important Concepts

- Pandas, DataFrame, loading files with Pandas.
- Accessing and modifying content in DataFrames.
- Analyzing and presenting data in DataFrames.
- Matplotlib and different ways to make plots.
- Drawing single and multiple plots. Changing elements of a plot.



# THAT'S ALL FOLKS!
# STAY HEALTHY