Pandas,
Data visualisation

Pandas

- Pandas is an open-source data analysis and manipulation library for Python.
- It provides a wide range of tools for working with structured data, including the ability to read and write data from a variety of sources such as CSV, Excel, SQL databases, and more.
- With Pandas, you can easily clean, filter, transform, and reshape data, as well as perform calculations and statistical analysis.
- Pandas also provides powerful data visualization capabilities that are built on top of Matplotlib. It is widely used in data science and machine learning workflows for data preparation and exploratory data analysis.
- https://pandas.pydata.org/docs/user_guide/index.html

Data structures

Series

```
0    1
1    3
2    5
3    7
4    9
dtype: int64
```

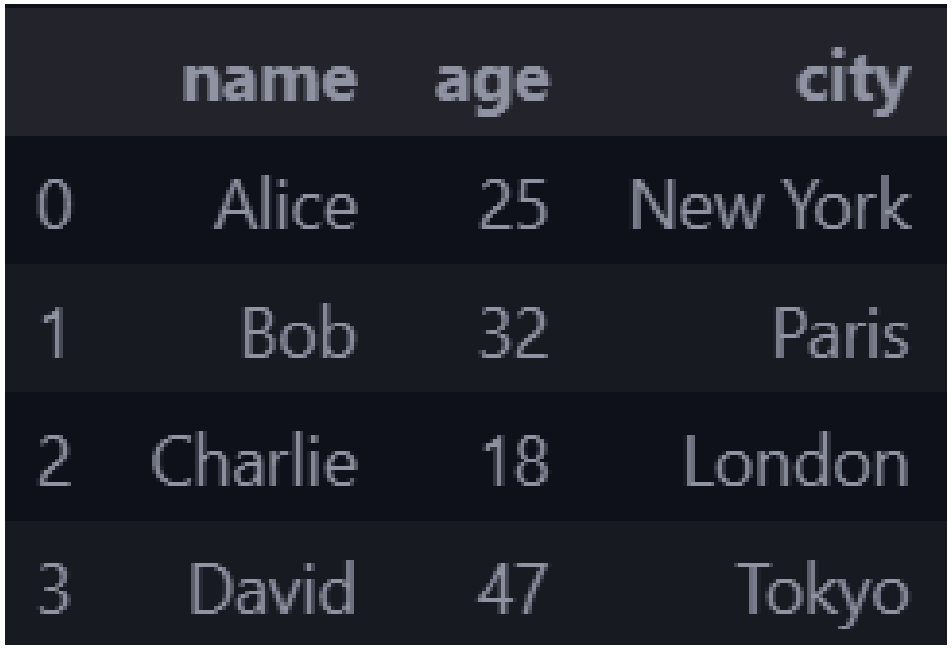
Dataframe

	name	age	city
0	Alice	25	New York
1	Bob	32	Paris
2	Charlie	18	London
3	David	47	Tokyo

Dataframe

column

Record, row



	name	age	city
0	Alice	25	New York
1	Bob	32	Paris
2	Charlie	18	London
3	David	47	Tokyo

Index label

Python example

Series

```python
import pandas as pd

# list if integers
a = [1, 7, 2]

# Create a Series from list
myvar = pd.Series(a)
```

Dataframe

```python
#import pandas package
import pandas as pd

# Create a DataFrame from a dictionary
data = {'name': ['Alice', 'Bob', 'Charlie', 'David'],
        'age': [25, 32, 18, 47],
        'city': ['New York', 'Paris', 'London', 'Tokyo']}
df = pd.DataFrame(data)
```

Head, Tail

```
df
```

✓ 0.0s

	Anime	Episodes	Year
0	One Piece	1009	1999
1	Naruto	720	2002
2	Bleach	366	2004
3	Hunter X Hunter	148	2011
4	Attack On Titan	74	2013
5	Gintama	366	2006
6	Code Geass	50	2007
7	Death Note	37	2008
8	Black Lagoon	24	2006
9	Classroom Of Elite	12	2016
10	Cowboy Bepop	26	1995
11	Jujutsu Kaisen	24	2020
12	Blue Period	12	2021

```
df.head()
```

✓ 0.0s

	Anime	Episodes	Year
0	One Piece	1009	1999
1	Naruto	720	2002
2	Bleach	366	2004
3	Hunter X Hunter	148	2011
4	Attack On Titan	74	2013

```
df.tail()
```

✓ 0.0s

	Anime	Episodes	Year
8	Black Lagoon	24	2006
9	Classroom Of Elite	12	2016
10	Cowboy Bepop	26	1995
11	Jujutsu Kaisen	24	2020
12	Blue Period	12	2021

Handle NaN values

	A	B	C	D
0	NaN	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	NaN	NaN	NaN	NaN
3	NaN	3.0	NaN	4.0

fillna()

```
fill_Value = 0
df_fill = df.fillna(fill_Value)
df_fill.head()
```

✓ 0.0s

	A	B	C	D
0	0.0	2.0	0.0	0.0
1	3.0	4.0	0.0	1.0
2	0.0	0.0	0.0	0.0
3	0.0	3.0	0.0	4.0

replace()

```
replace_value = 0
df_replace = df.replace(np.nan, replace_value)
df_replace.head()
```

✓ 0.0s

	A	B	C	D
0	0.0	2.0	0.0	0.0
1	3.0	4.0	0.0	1.0
2	0.0	0.0	0.0	0.0
3	0.0	3.0	0.0	4.0

dropna()

```
df_drop = df.dropna()
df_drop.head()
```

✓ 0.0s

A	B	C	D

Select row

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]}, index=['a', 'b', 'c'])
df.head()
```

✓ 0.1s

	A	B	C
a	1	4	7
b	2	5	8
c	3	6	9

```
# select a single row using loc
row_a = df.loc['a']
row_a
```

✓ 0.0s

```
A    1
B    4
C    7
Name: a, dtype: int64
```

```
# select a single row using iloc
row_0 = df.iloc[0]
row_0
```

✓ 0.0s

```
A    1
B    4
C    7
Name: a, dtype: int64
```

```
#Selecting rows using square brackets
df[:2]
```

✓ 0.0s

	A	B	C
a	1	4	7
b	2	5	8

Select column

```
column_A = df['A']
column_A
```

✓ 0.0s

```
a    1
a    2
c    3
Name: A, dtype: int64
```

```
columns = df[['A', 'B']]
columns
```

✓ 0.0s

	A	B
a	1	4
b	2	5
c	3	6

```
columns = df.columns
columns
```

✓ 0.0s

```
Index(['A', 'B', 'C'], dtype='object')
```

Rename columns

```
import pandas as pd

df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]}, index=['a', 'b', 'c'])
df
```

✓ 0.0s

	A	B	C
a	1	4	7
b	2	5	8
c	3	6	9

```
df_renamed = df.rename(columns={'A': 'a', 'B': 'b', 'C': 'c'})
df_renamed
```

✓ 0.0s

	a	b	c
a	1	4	7
b	2	5	8
c	3	6	9

Inplace

```python
import pandas as pd

df = pd.DataFrame({
    "A": [1, 2, 3],
    "B": [4, 5, 6]
})

# Using inplace=True
df["C"] = df["A"] + df["B"]  # add a new column "C" to df
df  # the original df is modified in place with the new column "C"

# Using inplace=False
df_with_C = df.copy()  # make a copy of the original df
df_with_C["C"] = df_with_C["A"] + df_with_C["B"]  # add a new column "C" to the copy
df_with_C  # the copy with the new column "C" is returned, leaving the original df unchanged
```

✓ 0.0s

	A	B	C
0	1	4	5
1	2	5	7
2	3	6	9

Broadcasting

```
# create a DataFrame and a Series
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]})
s = pd.Series([10, 20, 30], index=['A', 'B', 'C'])
```

```
df_added = df + s
df_added
```

✓ 0.1s

	A	B	C
0	11	24	37
1	12	25	38
2	13	26	39

```
df_mulled = df * 2
df_mulled
```

✓ 0.0s

	A	B	C
0	2	8	14
1	4	10	16
2	6	12	18

If statement

```python
# Sample data
data = {'A': [1, 2, 3, 4, 5], 'B': [6, 7, 8, 9, 10]}

# Create a DataFrame
df = pd.DataFrame(data)

# Add a new column 'C' with values based on 'A' column values
df['C'] = ''

# Use if statement to assign values to 'C' column
for i in range(len(df)):
    if df['A'][i] > 3:
        df['C'][i] = 'Greater than 3'
    else:
        df['C'][i] = 'Less than or equal to 3'

print(df)
```

✓ 0.0s

```
   A   B                       C
0  1   6  Less than or equal to 3
1  2   7  Less than or equal to 3
2  3   8  Less than or equal to 3
3  4   9           Greater than 3
4  5  10           Greater than 3
```

Iteration

```
# create a sample dataframe
data = {'Name': ['Alice', 'Bob', 'Charlie', 'Dave'],
        'Age': [25, 30, 35, 40],
        'Country': ['USA', 'UK', 'Canada', 'Australia']}
df = pd.DataFrame(data)

# iterate over the rows of the dataframe
for index, row in df.iterrows():
    print(row['Name'], row['Age'], row['Country'])
```

✓ 0.0s

```
Alice 25 USA
Bob 30 UK
Charlie 35 Canada
Dave 40 Australia
```

Subset

```python
# create a sample dataframe
data = {'Name': ['Alice', 'Bob', 'Charlie', 'Dave'],
        'Age': [25, 30, 35, 40],
        'Country': ['USA', 'UK', 'Canada', 'Australia']}
df = pd.DataFrame(data)

# subset the dataframe based on a condition
subset = df[df['Age'] > 30]

# print the subsetted dataframe
print(subset)
```

✓ 0.0s

```
      Name  Age    Country
2  Charlie   35     Canada
3     Dave   40  Australia
```

Group by

```
# create a sample dataframe
data = {'Name': ['Alice', 'Bob', 'Charlie', 'Dave', 'Eva', 'Frank'],
        'Age': [25, 30, 35, 40, 25, 30],
        'Country': ['USA', 'UK', 'Canada', 'Australia', 'USA', 'Canada']}
df = pd.DataFrame(data)

# group the dataframe by the 'Country' column and calculate the mean age for each group
grouped = df.groupby('Country')['Age'].mean()

# print the resulting groups and their mean ages
print(grouped)
```

✓ 0.0s

```
Country
Australia    40.0
Canada       32.5
UK           30.0
USA          25.0
Name: Age, dtype: float64
```

Numpy usecase example #1

```
data = {'Name': ['Alice', 'Bob', 'Charlie', 'Dave'],
        'Age': [25, 30, 35, 40],
        'Height': [1.68, 1.78, 1.75, 1.82],
        'Weight': [65, 70, 80, 75]}  # add weight information
df = pd.DataFrame(data)

# Calculate the BMI for each person
bmi = np.round(df['Weight'] / df['Height'] ** 2, 2)

# Add the BMI column to the dataframe
df['BMI'] = bmi

# Print the resulting dataframe
df
```

✓ 0.0s

	Name	Age	Height	Weight	BMI
0	Alice	25	1.68	65	23.03
1	Bob	30	1.78	70	22.09
2	Charlie	35	1.75	80	26.12
3	Dave	40	1.82	75	22.64

Numpy usecase example #2

```
# Create a sample dataframe
df = pd.DataFrame({'numbers': [1, 2, 3, 4, 5]})

# Filter the dataframe to only include even numbers
filtered_df = df.loc[np.where(df['numbers'] % 2 == 0)]

# Print the resulting filtered dataframe
print(filtered_df)
```

✓ 0.0s

```
   numbers
1        2
3        4
```

Numpy usecase example #3

```python
# create a sample dataframe
data = {'A': [1, 2, 3, 4],
        'B': [5, 6, 7, 8],
        'C': [9, 10, 11, 12]}
df = pd.DataFrame(data)

# create a numpy array from the dataframe
arr = df.to_numpy()

# calculate the mean of each column using NumPy
col_means = np.mean(arr, axis=0)

# subtract the column means from the original array
centered = arr - col_means

# create a new dataframe from the centered array
df_centered = pd.DataFrame(centered, columns=df.columns)

df_centered
```

✓ 0.0s

	A	B	C
0	-1.5	-1.5	-1.5
1	-0.5	-0.5	-0.5
2	0.5	0.5	0.5
3	1.5	1.5	1.5

Lambda operations

```
# Create a sample dataframe
df = pd.DataFrame({'numbers': [1, 2, 3, 4, 5]})

# Apply a lambda function to the 'numbers' column to double each value
df['doubled'] = df['numbers'].apply(lambda x: x * 2)

# Print the resulting dataframe
df
```

✓ 0.0s

	numbers	doubled
0	1	2
1	2	4
2	3	6
3	4	8
4	5	10

```
# create a dictionary of data
data = {'Name': ['Alice', 'Bob', 'Charlie', 'Dave'],
        'Age': [25, 30, 35, 40],
        'Height': [1.68, 1.78, 1.75, 1.82]}

# create a DataFrame from the dictionary
df = pd.DataFrame(data)

# Apply a lambda function to convert age to a categorical variable
df['Age Group'] = df['Age'].apply(lambda age: 'Under 30' if age < 30 else 'Over 30')

df
```

✓ 0.0s

	Name	Age	Height	Age Group
0	Alice	25	1.68	Under 30
1	Bob	30	1.78	Over 30
2	Charlie	35	1.75	Over 30
3	Dave	40	1.82	Over 30

What data sources can be used?

1. *CSV (Comma Separated Values) files*
2. *Excel spreadsheets*
3. *SQL databases (using the SQLalchemy library)*
4. *JSON (JavaScript Object Notation) files*
5. *HTML tables*
6. *Clipboard contents (e.g., copied from Excel or a web page)*
7. *Python dictionaries and lists*

Read from CSV

```
# Read the data from csv file
performance_df = pd.read_csv("StudentsPerformance.csv")

performance_df.head()
```

✓ 0.0s

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

Visualisation

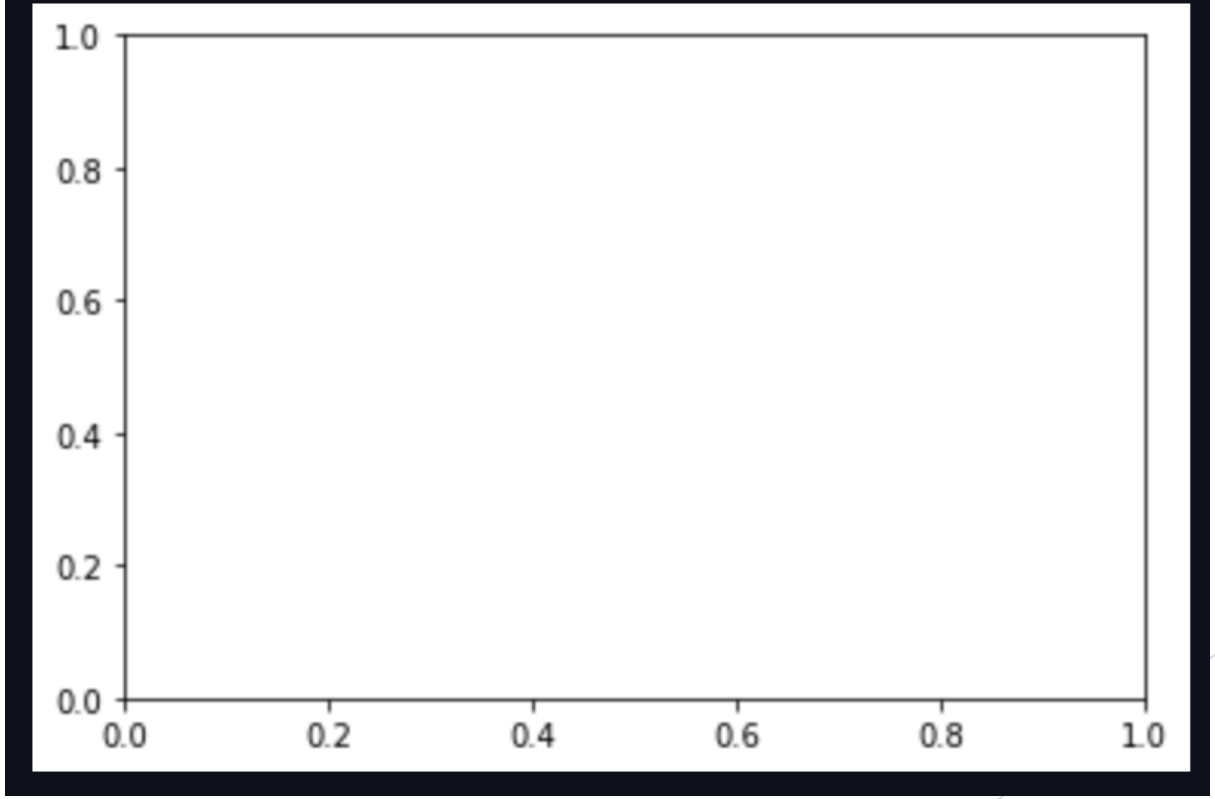
- Matplotlib is a plotting library for Python that provides a wide variety of tools to create visualizations of data in the form of line plots, scatter plots, bar plots, histograms, and many others. It is built on NumPy and provides a high-level interface for drawing attractive and informative statistical graphics.

- https://matplotlib.org/stable/gallery/index

Figure and the axes

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
```

✓ 1.1s

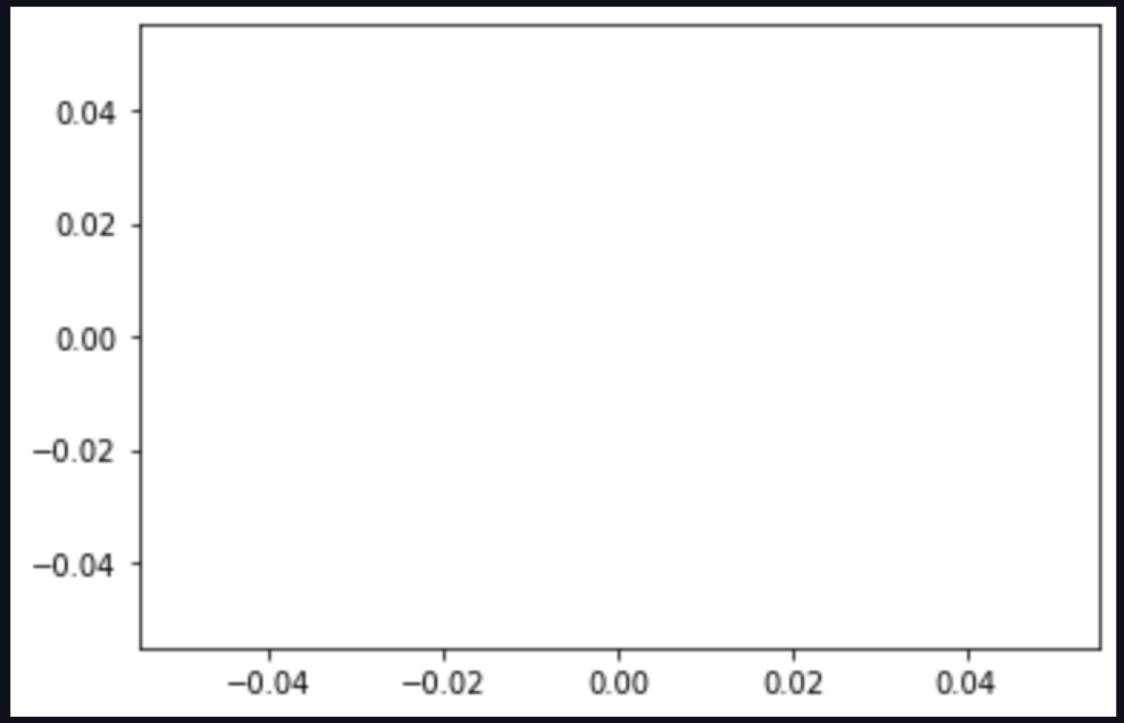


The plt interface

```
import matplotlib.pyplot as plt

plt.plot()
```

✓ 0.1s

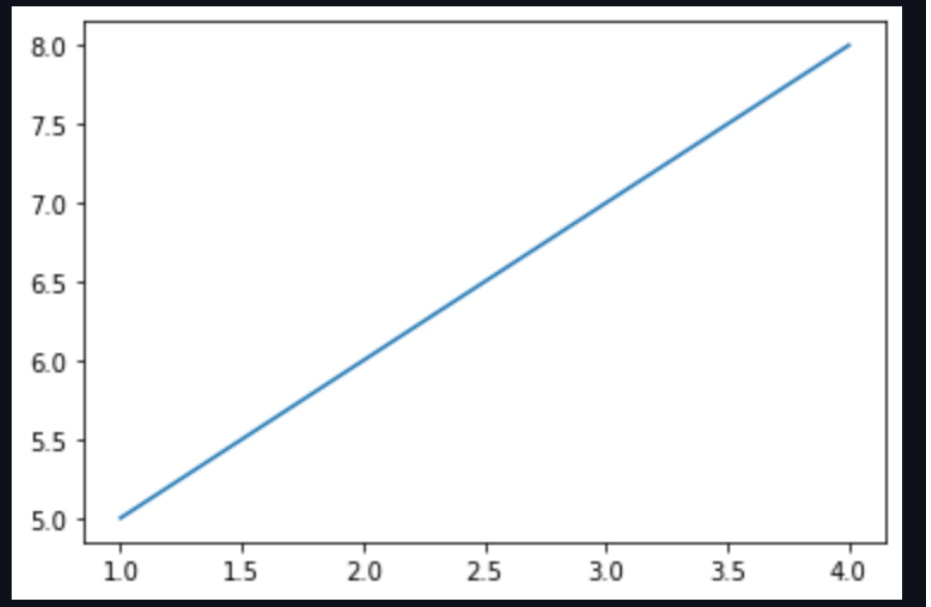
```
[]
```



Show

```
x = [1, 2, 3, 4]
y = [5, 6, 7, 8]

plt.plot(x, y)
plt.show()
```

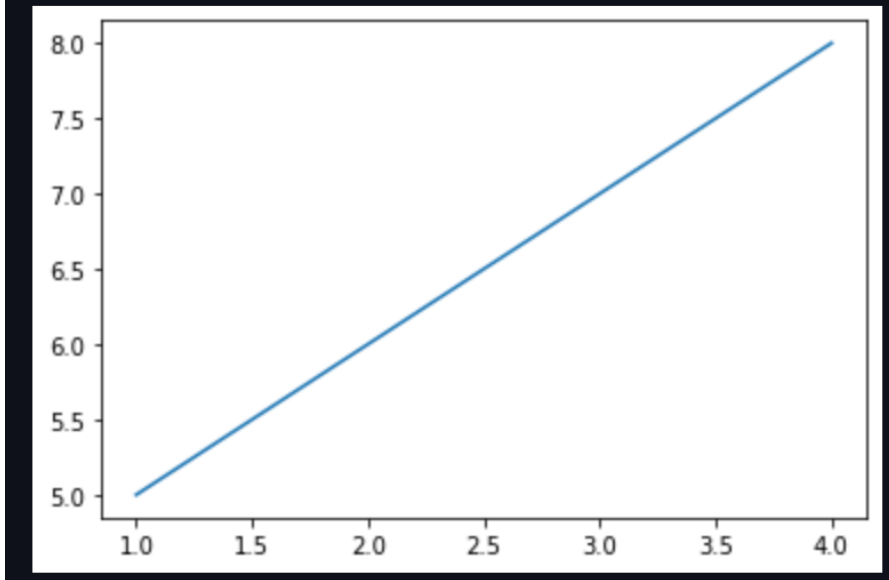
✓ 0.1s




```
x = [1, 2, 3, 4]
y = [5, 6, 7, 8]

plt.plot(x, y)
```

✓ 0.1s

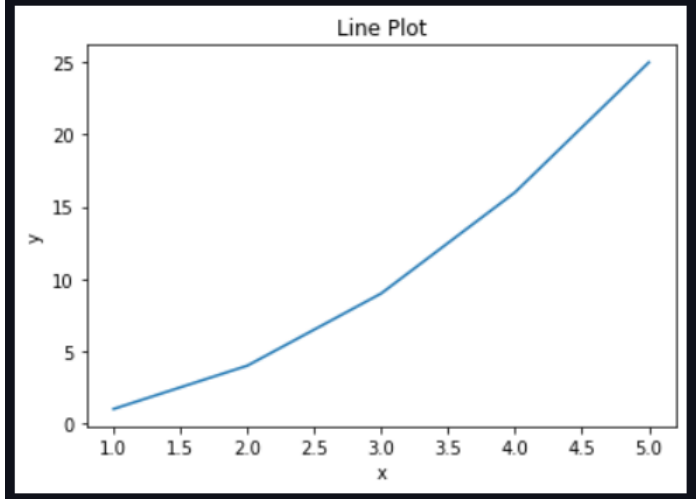
```
[<matplotlib.lines.Line2D at 0x1bc735b8970>]
```



Simple visualisation example

```python
# create some data
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# create a line plot
plt.plot(x, y)

# set the x and y axis labels and title
plt.xlabel('x')
plt.ylabel('y')
plt.title('Line Plot')

# show the plot
plt.show()
```

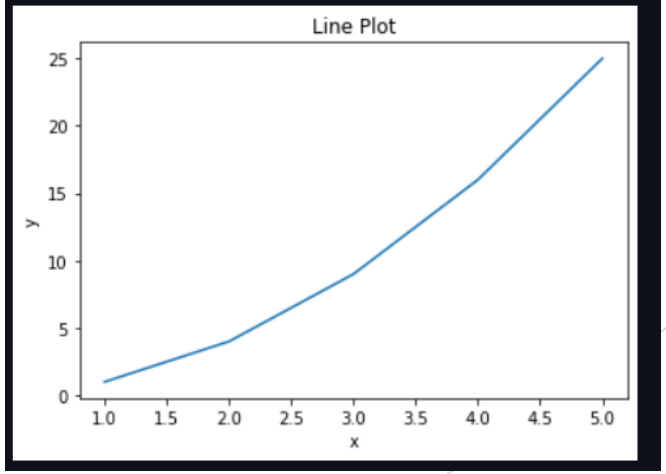
✓ 0.1s



OR

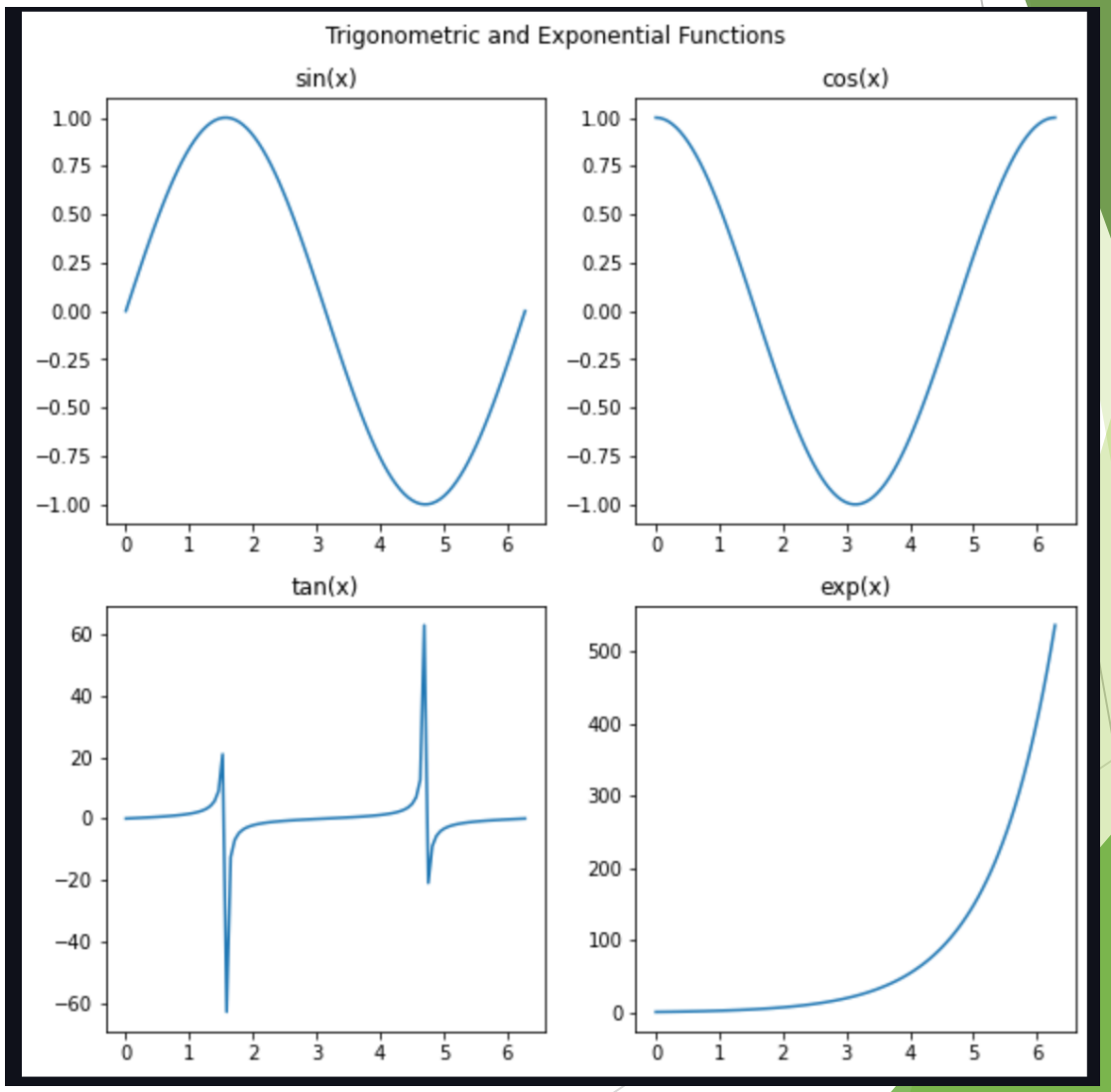
```python
# create some data
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

# create a figure and axis object
fig, ax = plt.subplots()

# plot the data on the axis
ax.plot(x, y)

# set the x and y axis labels and title
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('Line Plot')

# show the plot
plt.show()
```

✓ 0.1s



Subplots

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

# create some data
x = np.linspace(0, 2*np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)
y3 = np.tan(x)
y4 = np.exp(x)

# create a 2x2 grid of subplots
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 8))

# plot the data on each subplot
axes[0, 0].plot(x, y1)
axes[0, 0].set_title('sin(x)')

axes[0, 1].plot(x, y2)
axes[0, 1].set_title('cos(x)')

axes[1, 0].plot(x, y3)
axes[1, 0].set_title('tan(x)')

axes[1, 1].plot(x, y4)
axes[1, 1].set_title('exp(x)')

# set the overall title for the figure
fig.suptitle('Trigonometric and Exponential Functions')

# adjust the spacing between subplots
fig.tight_layout()

# display the plot
plt.show()
```



Plot from Dataframe

```python
# create a sample dataframe
data = {'year': [2015, 2016, 2017, 2018, 2019],
        'sales': [100, 150, 200, 250, 300]}
df = pd.DataFrame(data)

# create a figure and axis object
fig, ax = plt.subplots()

# plot the data on the axis
ax.plot(df['year'], df['sales'])

# set the x and y axis labels and title
ax.set_xlabel('Year')
ax.set_ylabel('Sales (in thousands)')
ax.set_title('Sales over the years')

# show the plot
plt.show()
```

✓ 0.1s