ITS INFORMATION AND COMMUNICATIONS TECHNOLOGY Academy

NOME MODULO: *IA*

UNITÀ DIDATTICA: IA.1

Lezione 2

Stefano Puglia

Biennio 2023-2025

ITS ICTAcademy

INFORMATION AND COMMUNICATIONS TECHNOLOGY

INDICE DEGLI ARGOMENTI

- Data wrangling
  - Accesso multisorgente
  - Pulizia
  - Fusione ed aggregazione
- Prime pipeline di analisi dati
- Introduzione a Matplotlib
  - Prime visualizzazioni (plot, chart)

Accesso a più sorgenti di dati

- Files
- URL
- Database

Accesso a più sorgenti di dati

```python
# Import from CSV
df_csv = pd.read_csv('customer_data.csv')
print("CSV Data (Customer Info):")
print(df_csv.head())
print("\n")

# Import from Excel
df_excel = pd.read_excel('transaction_data.xlsx')
print("Excel Data (Transactions):")
print(df_excel.head())
print("\n")

# Import from JSON
df_json = pd.read_json('preferences.json')
print("JSON Data (Preferences):")
print(df_json.head())
print("\n")

# Import from URL
URL = 'https://...'
df_url = pd.read_csv(URL)
print("Remote Data (Second hand cars):")
print(df_url.head())
print("\n")

# Import from PostgreSQL
QUERY = """SELECT * FROM product_info"""
engine = create_engine('postgresql+psycopg://postgres:postgres@postgresql:5432/titanic_db')
df_postgres = pd.read_sql_query(text(QUERY), con=engine.connect())
print("Postgres Data (Passengers):")
print(df_postgres.head())
print("\n")
```

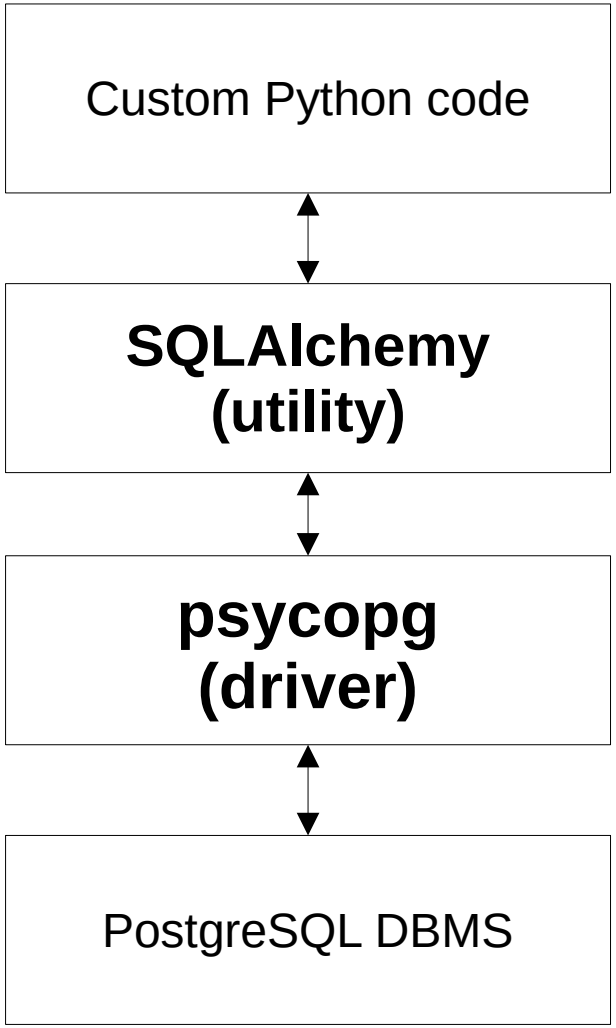
Accesso a PostgreSQL DB (Read)

```python
import pandas as pd
from sqlalchemy import create_engine, text

### LEGGI DATI DA PostgreSQL DB ###
SIMPLE_QUERY_1 = """
SELECT *
FROM public.passenger_info
"""
SIMPLE_QUERY_2 = """
SELECT *
FROM public.passenger_info
WHERE "Age" < 55
"""
# Set up della connessione ad un nostro database sul nostro PostgreSQL DB
engine = create_engine('postgresql+psycopg://postgres:postgres@postgresql:5432/prova_db')

print("\nQuery per la lettura dei dati sul nostro database PostgreSQL")
print(SIMPLE_QUERY_1)

# Leggi i dati da PostgreSQL ad un DataFrame
df_from_db = pd.read_sql_query(text(SIMPLE_QUERY_1), con=engine.connect())

print("\nDati letti dal nostro database PostgreSQL")
print("\n")
print(df_from_db)
```

Accesso a PostgreSQL DB (Read)

```python
import pandas as pd
from sqlalchemy import create_engine, text

### LEGGI DATI DA PostgreSQL DB ###
SIMPLE_QUERY_1 = """
SELECT *
FROM public.passenger_info
"""
SIMPLE_QUERY_2 = """
SELECT *
FROM public.passenger_info
WHERE "Age" < 55
"""
# Set up della connessione ad un nostro database sul nostro PostgreSQL DB
engine = create_engine('postgresql+psycopg://postgres:postgres@postgresql:5432/prova_db')

print("\nQuery per la lettura dei dati sul nostro database PostgreSQL")
print(SIMPLE_QUERY_1)

# Leggi i dati da PostgreSQL ad un DataFrame
df_from_db = pd.read_sql_query(text(SIMPLE_QUERY_1), con=engine.connect())

print("\nDati letti dal nostro database PostgreSQL")
print("\n")
print(df_from_db)
```

Accesso a PostgreSQL DB



Accesso a PostgreSQL DB (Write)

```
### SCRIVI DATI IN PostgreSQL DB ###
#Set up della connessione ad un nostro database sul nostro PostgreSQL DB
engine = create_engine('postgresql+psycopg://postgres:postgres@postgresql:5432/prova_db')

# Prepara una tabella
passenger_info = df[['Name', 'Sex', 'Age', 'Location']]
passenger_info.insert(1, "PassengerId", [1, 2, 3, 4], True)

# Scrivi la tabella in PostgreSQL
passenger_info.to_sql('passenger_info', engine, if_exists='replace', index=False)

print("\nTabella scritta nel nostro database PostgreSQL")
print("\n")
```

Accesso a PostgreSQL DB (R/W)

Controllo comportamenti e rilascio risorse

```python
def store_on_database(self, df: pd.DataFrame) -> None:
    """Scrive dati in un database PostgreSQL"""
    table_name = "auto_info"
    engine = create_engine(self.config.db_uri)
    try:
        with engine.begin() as conn:
            df.to_sql(table_name, con=conn, if_exists='replace', index=False)
    except SQLAlchemyError as e:
        print(f"Error di scrittura in database: {e}")
    finally:
        engine.dispose()

def load_from_database(self) -> pd.DataFrame:
    """Carica dati da un database PostgreSQL"""
    query_def = "SELECT * FROM public.auto_info"
    engine = create_engine(self.config.db_uri)
    try:
        with engine.connect() as conn:
            df = pd.read_sql_query(text(query_def), con=conn)[['make', 'price']].head()
    except SQLAlchemyError as e:
        print(f"Errore di lettura da database: {e}")
        df = pd.DataFrame()
    finally:
        engine.dispose()
    return df
```

Accesso a PostgreSQL DB (R/W)

Controllo comportamenti e rilascio risorse

```python
def store_on_database(self, df: pd.DataFrame) -> None:
    """Scrive dati in un database PostgreSQL"""
    table_name = "auto_info"
    engine = create_engine(self.config.db_uri)
    try:
        with engine.begin() as conn:
            df.to_sql(table_name, con=conn, if_exists='replace', index=False)
    except SQLAlchemyError as e:
        print(f"Error di scrittura in database: {e}")
    finally:
        engine.dispose()

def load_from_database(self) -> pd.DataFrame:
    """Carica dati da un database PostgreSQL"""
    query_def = "SELECT * FROM public.auto_info"
    engine = create_engine(self.config.db_uri)
    try:
        with engine.connect() as conn:
            df = pd.read_sql_query(text(query_def), con=conn)[['make', 'price']].head()
    except SQLAlchemyError as e:
        print(f"Errore di lettura da database: {e}")
        df = pd.DataFrame()
    finally:
        engine.dispose()
    return df
```

Pulizia dei dati

- Identificazione e trasfomazione valori nulli

```
df.replace(A, np.nan, inplace=True)
df.isnull()
df.notnull()
```

- Correzione valori nulli

  Eliminazione (righe e/o colonne)

```
df.dropna
```

  Sostituzione (es. media, frequenza)

```
df.replace(np.nan, sost, inplace=True)
```

- Correzione dei tipi di dato

```
df.dtypes
df.convert_dtypes()
df.astype()
```

- Correzione di refusi e “contenimento” dei valori (clipping)
- Normalizzazione

Pulizia dei dati

```
# "numpy NaNs" al posto di valori mancanti
df.replace("?", np.nan, inplace=True)

# Quanti NaNs per colonna
missing_data = df.isnull()
for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())

# Media al posto di NaNs
avg = df["normalized-losses"].astype("float").mean(axis = 0)
df["normalized-losses"].replace(np.nan, avg, inplace = True)

# Max frequenza al posto di NaNs
df["num-of-doors"].replace(np.nan, df['num-of-doors'].value_counts().idxmax(), inplace = True)

# Eliminazione righe dove NaNs
df.dropna(subset=["price"], axis=0, inplace = True)
df.reset_index(drop = True, inplace = True)

# Conversione tipi di dato
df = df.convert_dtypes()
df[["normalized-losses"]] = df[["normalized-losses"]].astype("int")
df[["price"]] = df[["price"]].astype("float")

# Normalizzazione dei dati
df['length'] = (df['length']-df['length'].min())/(df['length'].max() - df['length'].min())

# Correzione "typos"
df['make'] = df['make'].replace({'alfa-romero': 'alfa-romeo'})

# Limita gli outliers
df['peak-rpm'] = df['peak-rpm'].clip(lower=4200, upper=5200)
```

its ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Fusione ed aggregazione dati

- Oprazioni SQL-like (es. JOIN) su DataFrame
- Combinazione di “diverse provenienze”
  - DataFrame come “astrattore”
  - DataFrame come “unificatore”
- Gestione omogenea di semplicità e complessità

Fusione ed aggregazione dati

```python
# First merge: Transactions with Customer info (like SQL INNER JOIN)
merged_df = pd.merge(df_excel, df_csv, on='customer_id', how='inner')
print("After merging transactions with customer info:")
print(merged_df.head())
print("\n")

# Second merge: Add product info (like SQL LEFT JOIN)
final_df = pd.merge(merged_df, df_postgres, on='product_id', how='left')
print("Final merged dataframe:")
print(final_df.head())
print("\n")

# Another Merge
df_merged = pd.merge(df_postgres, df_json, on='PassengerId')
print("After merging products with additional details:")
print(df_merged.head())
print("\n")
```

Esercizio

Cominciamo ad assemblare pipeline di analisi dati

Cosa, come, perchè

- “Mettere insieme i pezzi”
- Passi della catena di analysis/analytics
- Dal dato crudo ai risultati visuali (pandas, Matplotlib...)
- Classi e funzioni Python
- Modularità, incapsulamento, flessibilità, estensibilità

Data pipeline

Es. Auto usate (da /Lezione2/codice/autos/autos_data_pipeline.py)

```python
def run_pipeline(self) -> pd.DataFrame:
    """Esegue la pipeline completa"""
    # Carica dati da remoto
    remote_df = self.load_from_remote()
    print("Letto file remoto")
    # Salva dati in locale
    self.save_on_csv(remote_df)
    print("Salvato file remoto in locale")
    # Scrive dati in database
    self.store_on_database(remote_df)
    print("Scritto file remoto in una tabella su db")
    # Legge dati da database
    db_df = self.load_from_database()
    print("Letti dati da una tabella su db")
    self.data = db_df
    return db_df
```

its ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Data pipeline

```
(base) stefano@stefano-ThinkPad-T450s:~/Documents/Personal/Courses/ITS_Academy/Lezioni_IA.1$ docker exec -it -w /home/Lezione2/codice/autos its_dev python autos_data_pipeline.py
Letto file remoto
Salvato file remoto in locale
Scritto file remoto in una tabella su db
Letti dati da una tabella su db
Pipeline completata con successo!
          make  price
0  alfa-romero  13495
1  alfa-romero  16500
2  alfa-romero  16500
3         audi  13950
4         audi  17450
```

Data pipeline

Es. Auto usate (da /Lezione2/codice/autos/autos_data_pipeline.py)

```python
def run_pipeline(self) -> pd.DataFrame:
    """Esegue la pipeline completa"""
    # Carica dati da remoto
    remote_df = self.load_from_remote()
    print("Letto file remoto")
    # Salva dati in locale
    self.save_on_csv(remote_df)
    print("Salvato file remoto in locale")
    # Scrive dati in database
    self.store_on_database(remote_df)
    print("Scritto file remoto in una tabella su db")
    # Legge dati da database
    db_df = self.load_from_database()
    print("Letti dati da una tabella su db")
    self.data = db_df
    return db_df
```

Andiamo al codice...

Data pipeline

Es. Titanic (da /Lezione2/codice/titanic/titanic_data_pipeline.py)

```python
def run_pipeline(self) -> pd.DataFrame:
    """Esegue la pipeline completa"""
    # Carica dati da più fonti
    db_df1, db_df2 = self.load_from_database()
    json_df = self.load_from_json()
    print("Letti dati da più fonti (db, JSON)")
    # Preprocessa dati (aggrega, espande e pulisce)
    merged_df = self.merge_data(db_df1, db_df2, json_df)
    expanded_df = self.expand_json_data(merged_df)
    cleaned_df = self.clean_data(expanded_df)
    print("Effettuato preprocessamento dati")
    # Visualizza risultati
    self.visualize(cleaned_df)
    print("Visualizzati risultati di analisi")
    self.data = cleaned_df
    return cleaned_df
```

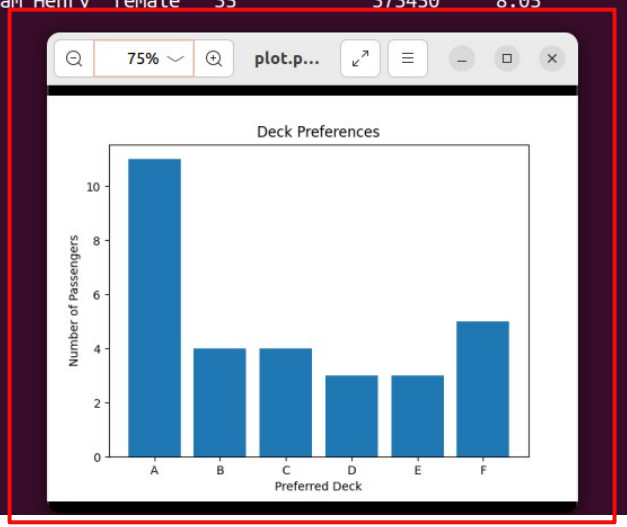
Data pipeline

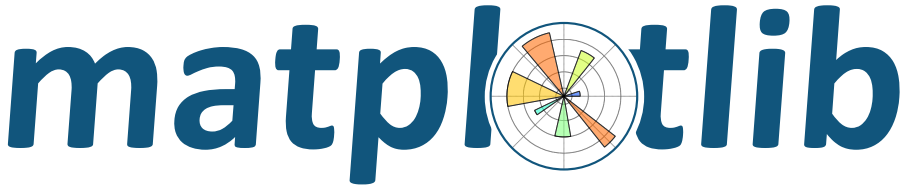
Es. Titanic (da /Lezione2/codice/titanic/titanic_data_pipeline.py)

```python
def run_pipeline(self) -> pd.DataFrame:
    """Esegue la pipeline completa"""
    # Carica dati da più fonti
    db_df1, db_df2 = self.load_from_database()
    json_df = self.load_from_json()
    print("Letti dati da più fonti (db, JSON)")
    # Preprocessa dati (aggrega, espande e pulisce)
    merged_df = self.merge_data(db_df1, db_df2, json_df)
    expanded_df = self.expand_json_data(merged_df)
    cleaned_df = self.clean_data(expanded_df)
    print("Effettuato preprocessamento dati")
    # Visualizza risultati
    self.visualize(cleaned_df)
    print("Visualizzati risultati di analisi")
    self.data = cleaned_df
    return cleaned_df
```

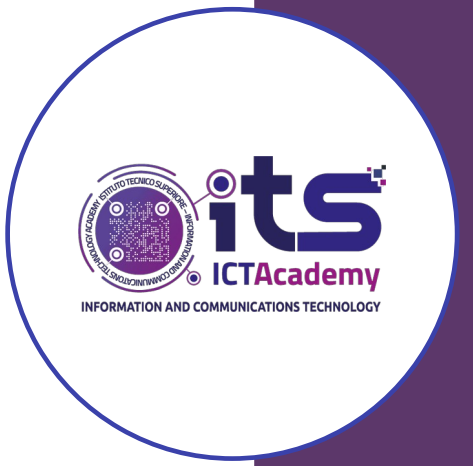
Data pipeline

```
(base) stefano@stefano-ThinkPad-T450s:~/Documents/Personal/Courses/ITS_Academy/Lezioni_IA.1$ docker exec -it -w /home/Lezione2/codice/autos its_dev python autos_data_pipeline.py
Letto file remoto
Salvato file remoto in locale
Scritto file remoto in una tabella su db
Letti dati da una tabella su db
Pipeline completata con successo!
          make  price
0  alfa-romero  13495
1  alfa-romero  16500
2  alfa-romero  16500
3         audi  13950
4         audi  17450
(base) stefano@stefano-ThinkPad-T450s:~/Documents/Personal/Courses/ITS_Academy/Lezioni_IA.1$ Lezione2/display_plot_script.sh
Letti dati da più fonti (db, JSON)
Effettuato preprocessamento dati
Visualizzati risultati di analisi
Pipeline completata con successo!
   PassengerId                                                Name     Sex  Age            Ticket     Fare preferred_deck dining_time   activity
0            1                             Braund, Mr. Owen Harris    male   22         A/5 21171     7.25             B       early    reading
1            2   Cumings, Mrs. John Bradley (Florence Briggs Th...  female   38          PC 17599  71.2833             F       early   sleeping
2            3                              Heikkinen, Miss Laina  female   26  STON/O2. 3101282   7925.0             C       early  exploring
3            4        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female   35            113803     53.1             C    flexible   sleeping
4            5                            Allen, Mr. William Henry  female   35            373450     8.05             F        late    reading
```

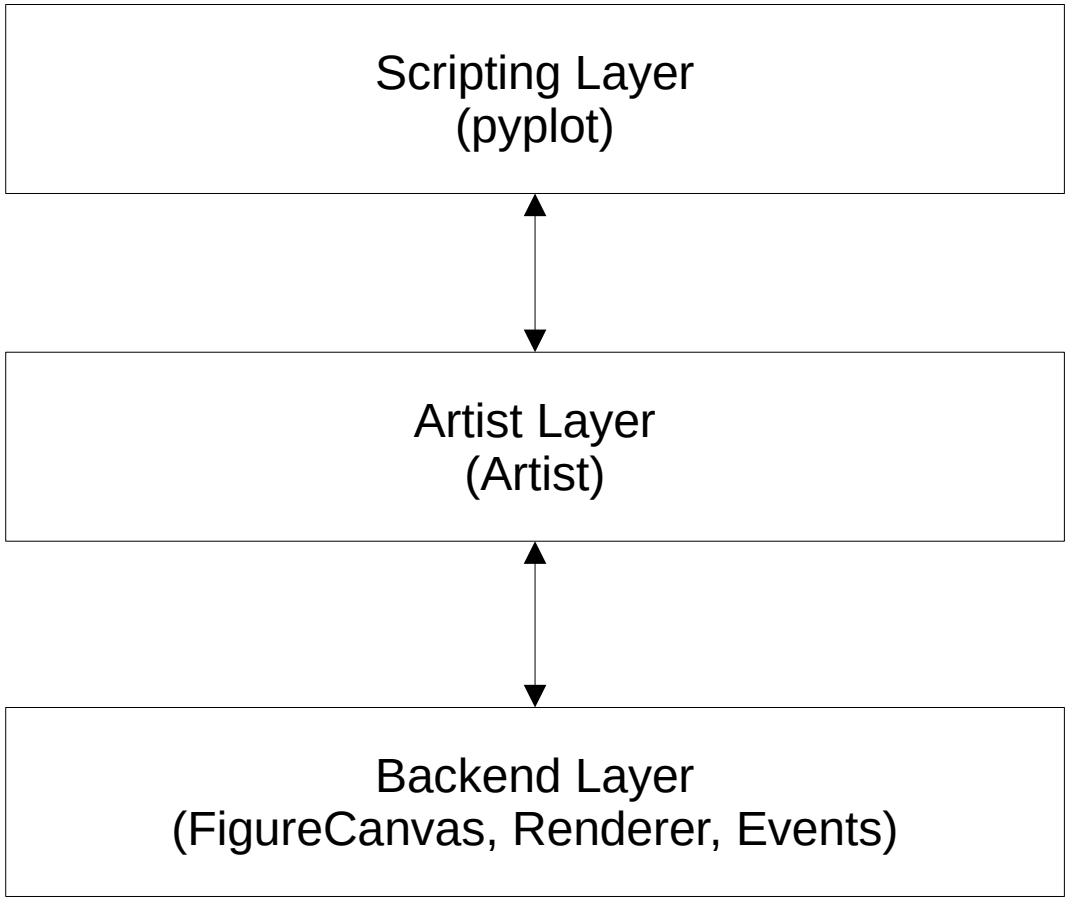





```
import matplotlib.pyplot as plt
```

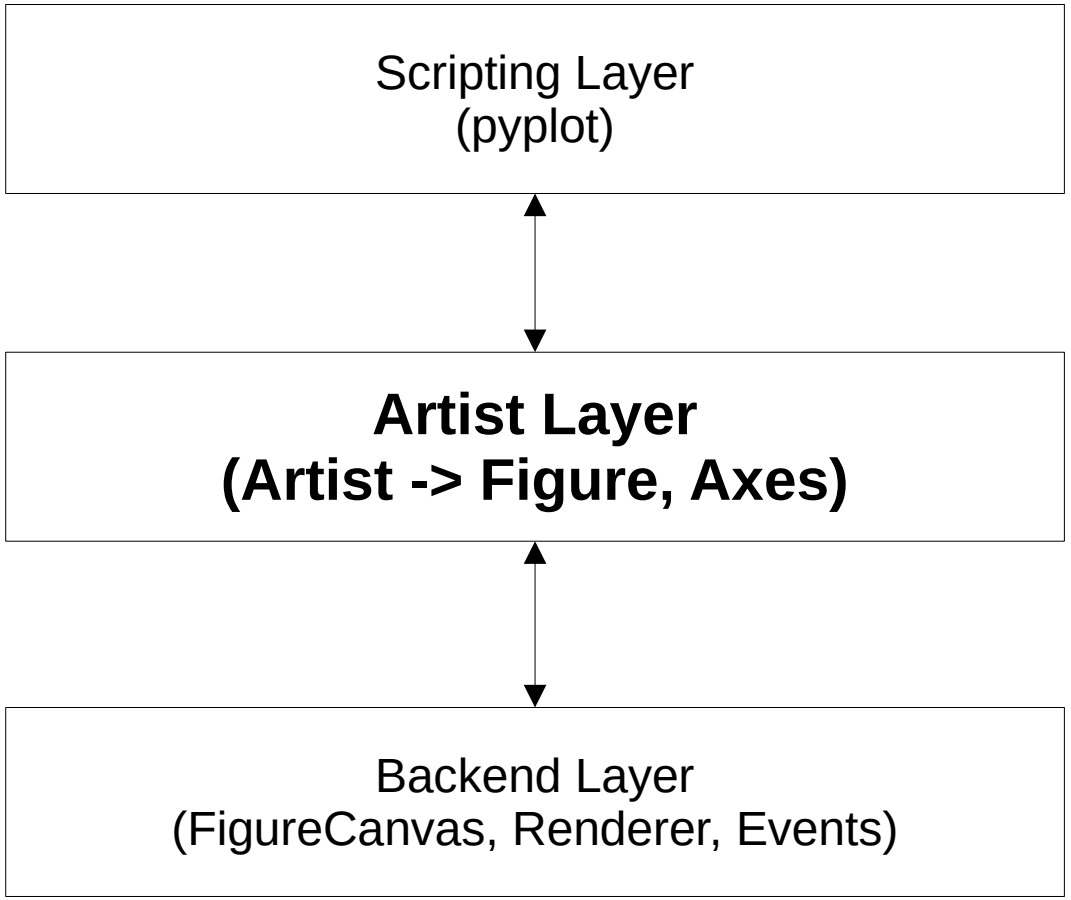


Matplotlib - Architettura

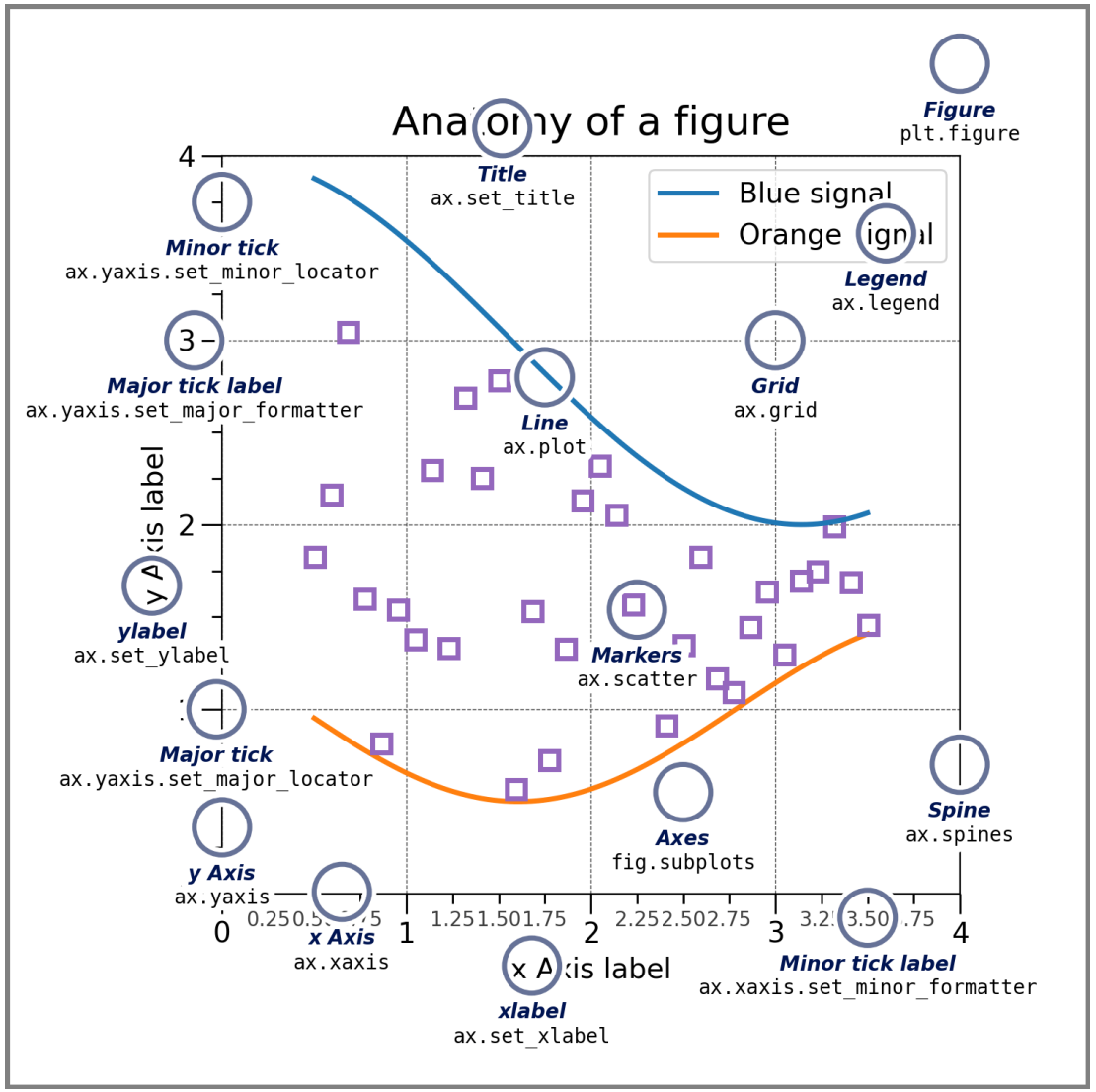


Per approfondimenti: The Architecture of Open Source Applications (Volume 2) - matplotlib

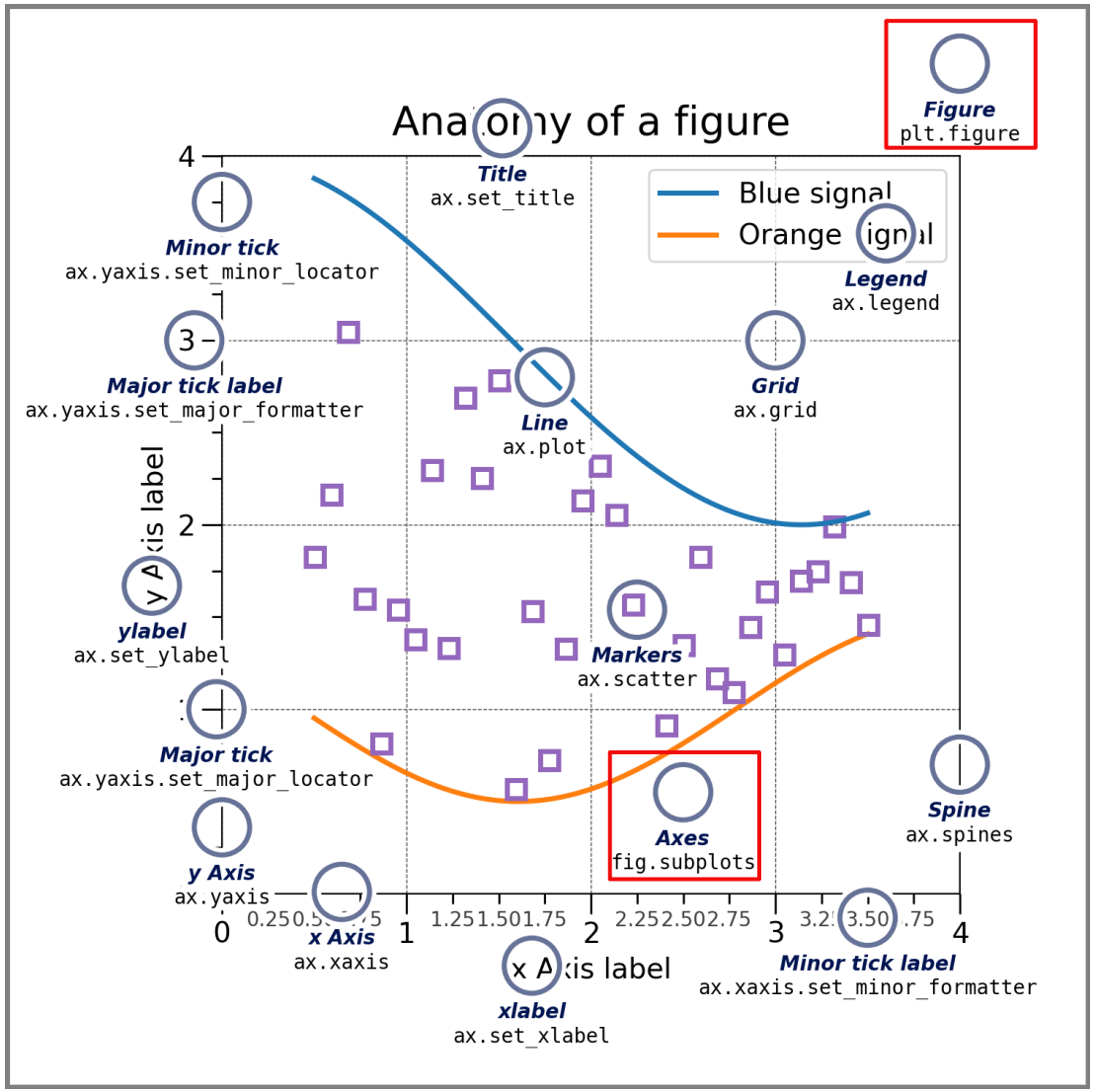
Matplotlib - Artist



Matplotlib - Artist



Matplotlib - Artist



ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

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

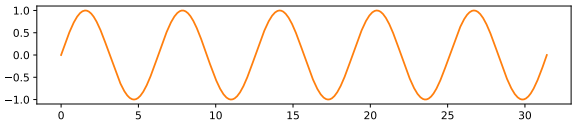
2 Prepare

```
X = np.linspace(0, 10*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
plt.show()
```

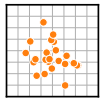
4 Observe



Choose

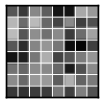
Matplotlib offers several kind of plots (see Gallery):

```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```



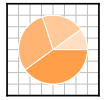
```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```

```
Z = np.random.uniform(0, 1, (8, 8))

ax.imshow(Z)
```

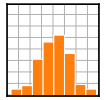


```
Z = np.random.uniform(0, 1, (8, 8))

ax.contourf(Z)
```

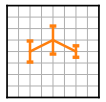
```
Z = np.random.uniform(0, 1, 4)

ax.pie(Z)
```



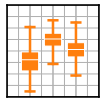
```
Z = np.random.normal(0, 1, 100)

ax.hist(Z)
```



```
X = np.arange(5)
Y = np.random.uniform(0, 1, 5)
ax.errorbar(X, Y, Y/4)
```



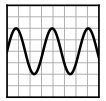
```
Z = np.random.normal(0, 1, (100, 3))

ax.boxplot(Z)
```



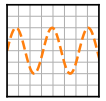
Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

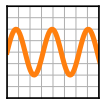
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```



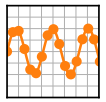
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```



```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```



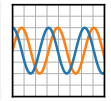
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```



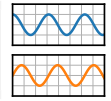
Organize

You can plot several data on the same figure, but you can also split a figure in several subplots (named *Axes*):

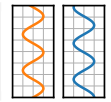
```
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)
```



```
fig, (ax1, ax2) = plt.subplots(2, 1)
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```

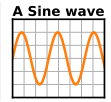


```
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```

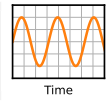


Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```




```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```



Explore

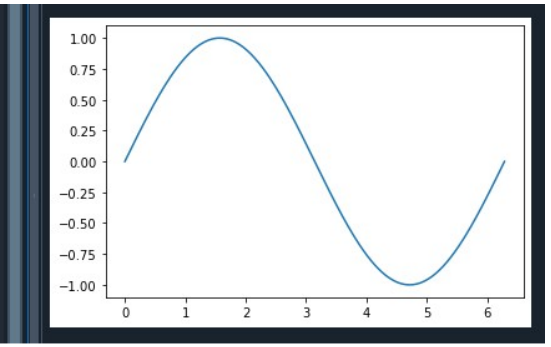
Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```

Matplotlib - Artist

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

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)
fig, ax = plt.subplots() # Oggetti di tipo figura e asse per il grafico
ax.plot(x, y)
plt.savefig('matplotlib_sine_line.png')
plt.show()
```

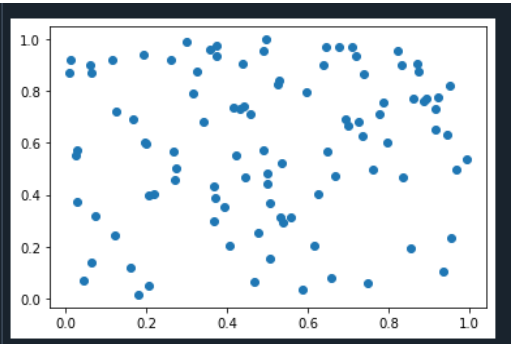


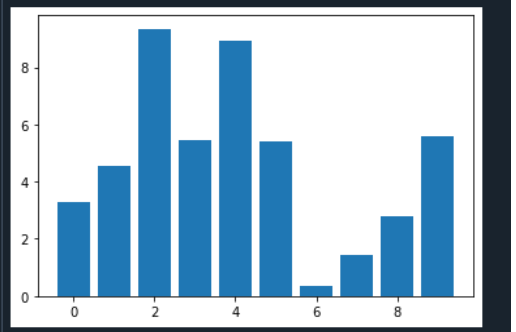
ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Matplotlib - Artist

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

X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
fig, ax = plt.subplots() # Oggetti di tipo figura e asse per il grafico
ax.scatter(X, Y)
plt.savefig('matplotlib_scatter.png')
plt.show()
```

```python
X = np.arange(10)
Y = np.random.uniform(0, 10, 10)
fig, ax = plt.subplots() # Oggetti di tipo figura e asse per il grafico
ax.bar(X, Y)
plt.savefig('matplotlib_bar.png')
plt.show()
```



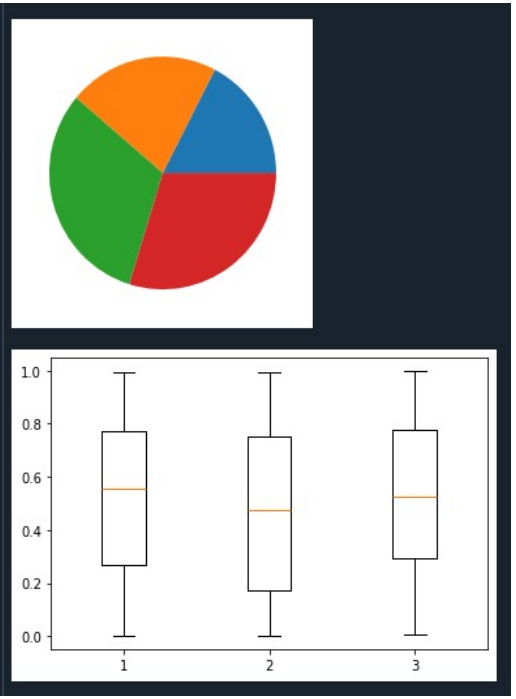


ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Matplotlib - Artist

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

Z =np.random.uniform(0, 1, 4)
fig, ax = plt.subplots() # Oggetti di tipo figura e asse per il grafico
ax.pie(Z)
plt.savefig('matplotlib_pie.png')
plt.show()
```

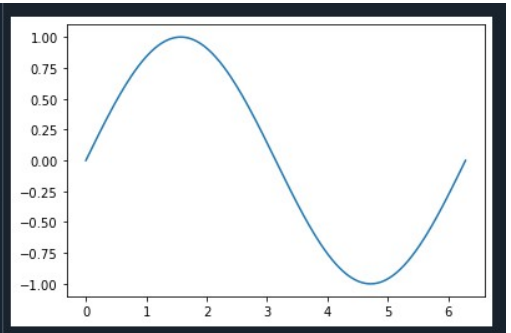
```python
Z =np.random.uniform(0, 1, (100, 3))
fig, ax = plt.subplots() # Oggetti di tipo figura e asse per il grafico
ax.boxplot(Z)
plt.savefig('matplotlib_boxplot.png')
plt.show()
```



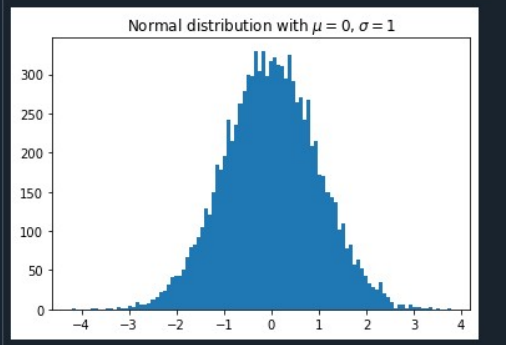
ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Matplotlib - Artist

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

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)
fig, ax = plt.subplots() # Oggetti di tipo figura e asse per il grafico
ax.plot(x, y)
plt.savefig('matplotlib_sine_line.png')
plt.show()
```



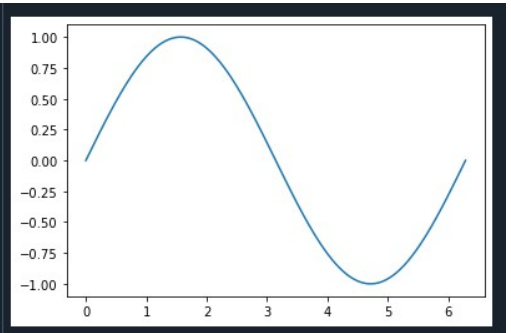

```python
x = np.random.randn(10000) # 10000 punti distribuiti secondo una gaussiana (standard normal distribution)
                           # centrata sullo zero e con varianza uguale a 1.
                           # Es. -0.455, 0.947, -1.123, ..., 1.675)
plt.hist(x, 100)           # numero di punti raggruppati in 100 intervalli (bins)
plt.title(r'Normal distribution with $\mu=0, \sigma=1$')
plt.savefig('matplotlib_histogram.png')
plt.show()
```



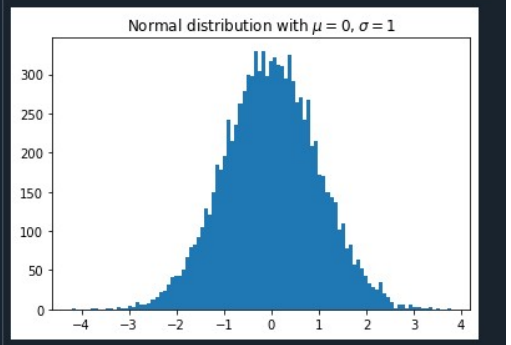
ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Matplotlib - Artist e pyplot

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

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)
fig, ax = plt.subplots() # Oggetti di tipo figura e asse per il grafico
ax.plot(x, y)
plt.savefig('matplotlib_sine_line.png')
plt.show()
```

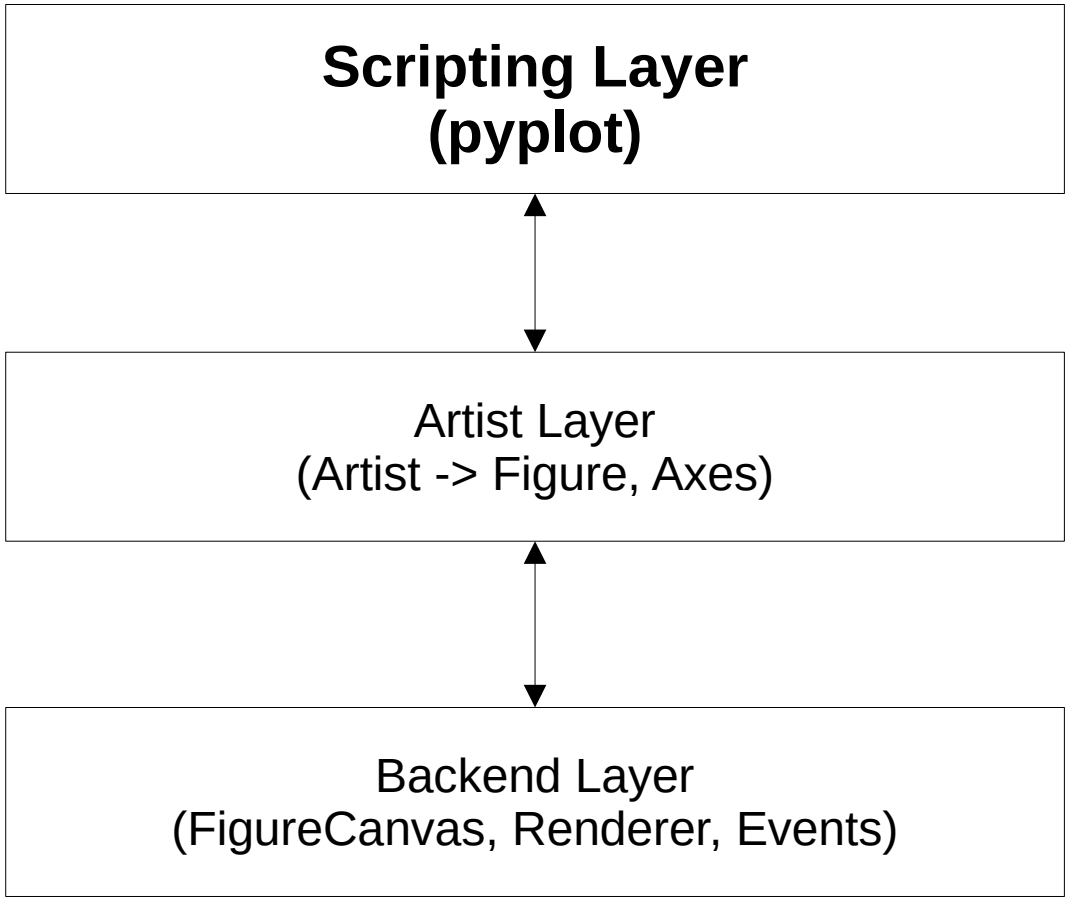



```python
x = np.random.randn(10000) # 10000 punti distribuiti secondo una gaussiana (standard normal distribution)
                           # centrata sullo zero e con varianza uguale a 1.
                           # Es. -0.455, 0.947, -1.123, ..., 1.675)
plt.hist(x, 100)           # numero di punti raggruppati in 100 intervalli (bins)
plt.title(r'Normal distribution with $\mu=0, \sigma=1$')
plt.savefig('matplotlib_histogram.png')
plt.show()
```



ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Matplotlib - pyplot



Matplotlib - Artist e pyplot

```
df_agg_1 = df.groupby('preferred_deck').agg(deck_count=('PassengerId', 'count'))
```

```
X = df_agg_1.index.astype(str)
Y = df_agg_1['deck_count']
fig, ax = plt.subplots()
ax.bar(X, Y)
```

OPPURE

```
plt.bar(x=df_agg_1.index.astype(str), height=df_agg_1['deck_count'])
```

Matplotlib - Artist e pyplot

```
df_agg_1 = df.groupby('preferred_deck').agg(deck_count=('PassengerId', 'count'))
```

```
X = df_agg_1.index.astype(str)
Y = df_agg_1['deck_count']
fig, ax = plt.subplots()
ax.bar(X, Y)
```

OPPURE

```
plt.bar(x=df_agg_1.index.astype(str), height=df_agg_1['deck_count'])
```

OPPURE

Matplotlib(Artist e pyplot) e pandas

```
df_agg_1 = df.groupby('preferred_deck').agg(deck_count=('PassengerId', 'count'))
```

```
X = df_agg_1.index.astype(str)
Y = df_agg_1['deck_count']
fig, ax = plt.subplots()
ax.bar(X, Y)
```

OPPURE

```
plt.bar(x=df_agg_1.index.astype(str), height=df_agg_1['deck_count'])
```

OPPURE

```
df_agg_1.plot(kind = 'bar')
```

Matplotlib (pyplot) e pandas

```
df_agg_1 = df.groupby('preferred_deck').agg(deck_count=('PassengerId', 'count'))
```

```
X = df_agg_1.index.astype(str)
Y = df_agg_1['deck_count']
fig, ax = plt.subplots()
ax.bar(X, Y)
```

OPPURE

```
plt.bar(x=df_agg_1.index.astype(str), height=df_agg_1['deck_count'])
```

OPPURE

```
df_agg_1.plot(kind = 'bar')
```

RIFERIMENTI BIBLIOGRAFICI

- click to pandas documentation
- click to matplotlib documentation