ITS INFORMATION AND COMMUNICATIONS TECHNOLOGY Academy

NOME MODULO: *IA*

UNITÀ DIDATTICA: IA.1

Lezione 1

Stefano Puglia

Biennio 2024-2026

Parte 1

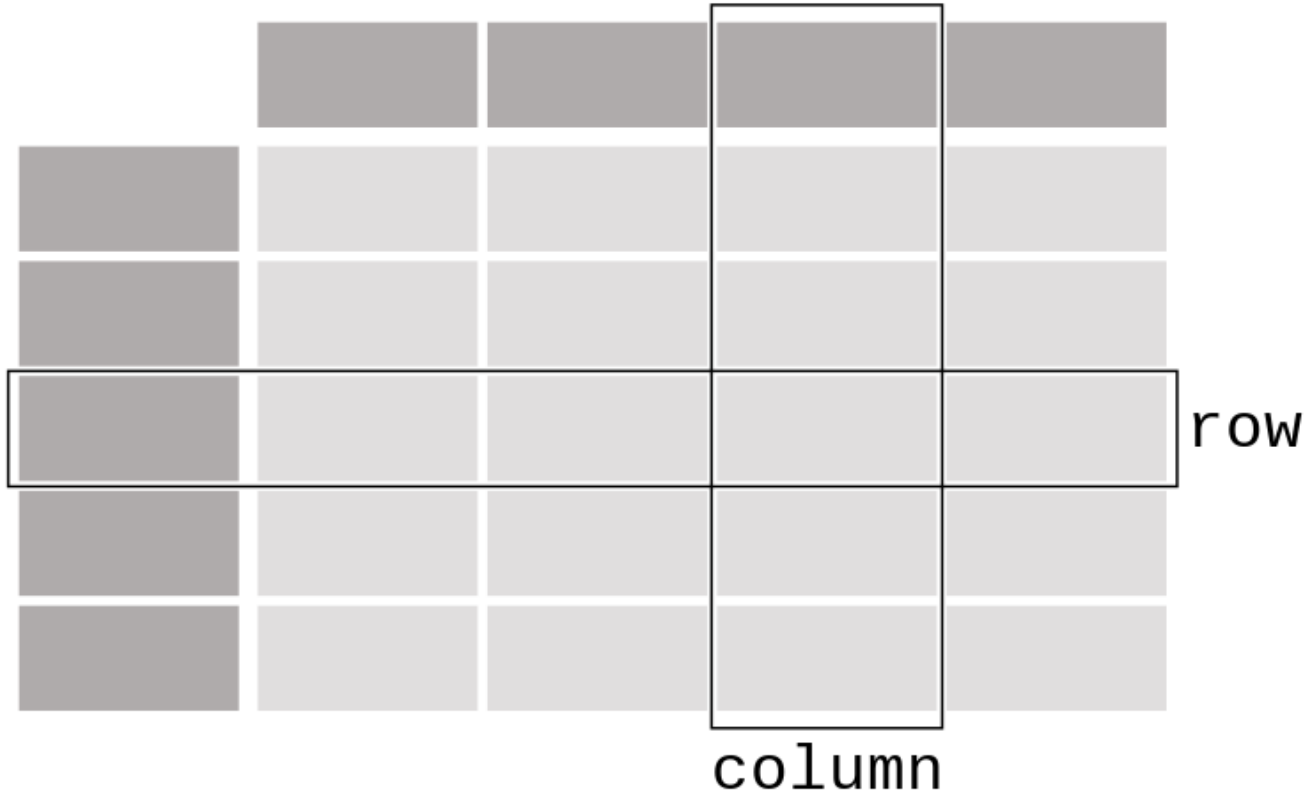

its
ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

INDICE DEGLI ARGOMENTI

- Introduzione a DataFrame e Series di pandas
- Leggere dati in un DataFrame
- Selezione colonne
- Selezione righe
- “Affettare” un DataFrame
- “Filtrare” un DataFrame in base a criteri
- Scrivere un DataFrame in un file


```
import pandas as pd
```

DataFrame

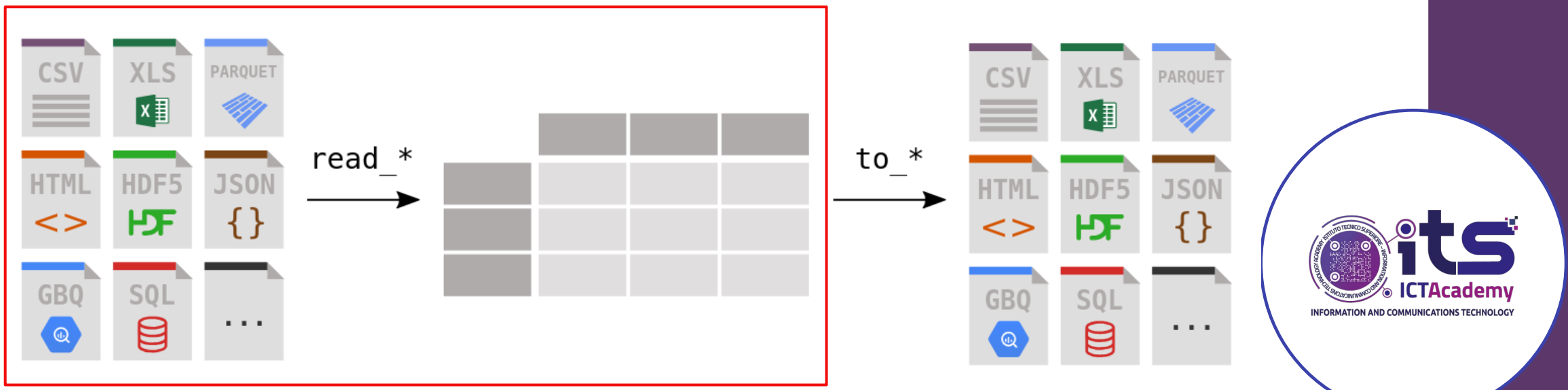


ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Un DataFrame

Artist	Album	Released	Length	Genre	Music recording sales (millions)	Claimed sales (millions)	Released	Soundtrack	Rating (friends)
Michael Jackson	Thriller	1982	00:42:19	Pop, rock, R&B	46	65	30-Nov-82		10.0
AC/DC	Back in Black	1980	00:42:11	Hard rock	26.1	50	25-Jul-80		8.5
Pink Floyd	The Dark Side of the Moon	1973	00:42:49	Progressive rock	24.2	45	01-Mar-73		9.5
Whitney Houston	The Bodyguard	1992	00:57:44	Soundtrack/R&B, soul, pop	26.1	50	25-Jul-80	Y	7.0
Meat Loaf	Bat Out of Hell	1977	00:46:33	Hard rock, progressive rock	20.6	43	21-Oct-77		7.0
Eagles	Their Greatest Hits (1971-1975)	1976	00:43:08	Rock, soft rock, folk rock	32.2	42	17-Feb-76		9.5
Bee Gees	Saturday Night Fever	1977	1:15:54	Disco	20.6	40	15-Nov-77	Y	9.0
Fleetwood Mac	Rumours	1977	00:40:01	Soft rock	27.9	40	04-Feb-77		9.5

Creazione di un DataFrame



- Da un dizionario Python

```
df = pd.DataFrame(dict)
```

- Da un file (es. csv, Excel)

```
df = pd.read_csv("../dati/clean_data.csv")
```

- Da tabelle di un database

Creazione di un DataFrame

```python
import pandas as pd

dict =    {

        "Name": [

            "Braund, Mr. Owen Harris",

            "Allen, Mr. William Henry",

            "Bonnell, Miss. Elizabeth",

            "Taylor, Miss. Jane"

        ],

        "Age": [22, 35, 58, 55],

        "Sex": ["male", "male", "female", "female"],

        "Location": ["Rome", "London", "Berlin", "New York"],

    }

df = pd.DataFrame(dict)
print("Un dataframe creato da un dizionario Python")
print(df)
print("\n")
```

```
In [3]: runfile('/home/stefano/Documents/Personal/Courses/
ITS_Academy/Lezioni_IA.1/Lezione1/codice/primeprovepandas.py',
wdir='/home/stefano/Documents/Personal/Courses/ITS_Academy/
Lezioni_IA.1/Lezione1/codice')
Un dataframe creato da un dizionario Python
                        Name  Age     Sex  Location
0   Braund, Mr. Owen Harris   22    male      Rome
1  Allen, Mr. William Henry   35    male    London
2  Bonnell, Miss. Elizabeth   58  female    Berlin
3        Taylor, Miss. Jane   55  female  New York


In [4]:
```

ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Creazione di un DataFrame

```python
import pandas as pd

students = pd.read_csv("../dati/clean_data.csv")
print("Un dataframe creato da un file csv")
print(students)
print("\n")
```

```
Un dataframe creato da un file csv
     age gender  screen_time_hours  ...  math_score  science_score  is_healthy
0     15      M                2.0  ...          75             69       False
1     14      M                4.8  ...          73             75       False
2     15      F                6.8  ...          74             70       False
3     12      F                3.5  ...          80             76       False
4     12      F                6.4  ...          75             78       False
..   ...    ...                ...  ...         ...            ...         ...
495   15      F                7.7  ...          81             78       False
496   13      M                5.0  ...          72             75       False
497   14      M                2.8  ...          75             61       False
498   14      F                6.2  ...          74             57       False
499   15      F                8.0  ...          77             65       False

[500 rows x 9 columns]
```

Informazioni su un DataFrame df

- `df.head()`, prime righe (default 5)
- `df.tail()`, ultime righe (default 5)
- `df.describe()`, statistiche di base
- `df.dtypes`, tipi di dato colonne
- `df.info()`, sommario tecnico
- `df.columns`, lista intestazioni colonne
- `df.index`, lista indici di riga
- `df.shape`, formato del DataFrame

Selezione colonne

Metodo veloce (solo colonne con nome “unico”)

- `df.column_name`, restituisce una Series

Metodo robusto (qualsiasi nome colonna, più colonne)

- `df['column']`, restituisce una Series
- `df[['column 1', 'column 2']]`, restituisce un DataFrame

Selezione colonne

```
print(df.Location)
print("\n")
print(df[['Age', 'Sex']])
```

```
Un dataframe creato da un dizionario Python
                        Name  Age     Sex  Location
0   Braund, Mr. Owen Harris   22    male      Rome
1  Allen, Mr. William Henry   35    male    London
2  Bonnell, Miss. Elizabeth   58  female    Berlin
3        Taylor, Miss. Jane   55  female  New York


0        Rome
1      London
2      Berlin
3    New York
Name: Location, dtype: object


   Age     Sex
0   22    male
1   35    male
2   58  female
3   55  female
```

Selezione righe

Filtro per etichetta dell’indice/colonna

- `df.loc[label]`, restituisce una Series
- `df.loc[[label]]`, restituisce un DataFrame

Filtro per posizione dell’indice/colonna

- `df.iloc[index]`, restituisce una Series
- `df.iloc[[index]]`, restituisce un DataFrame

Selezione righe

```
print(df.iloc[[2]])
print("\n")
df.set_index('Name', inplace=True)
print(df.loc[['Taylor, Miss. Jane']])
```

```
Un dataframe creato da un dizionario Python
                        Name  Age     Sex  Location
0   Braund, Mr. Owen Harris   22    male      Rome
1  Allen, Mr. William Henry   35    male    London
2  Bonnell, Miss. Elizabeth   58  female    Berlin
3        Taylor, Miss. Jane   55  female  New York


                        Name  Age     Sex Location
2  Bonnell, Miss. Elizabeth   58  female   Berlin


                     Age     Sex  Location
Name
Taylor, Miss. Jane   55  female  New York
```

‘Affettare’ un DataFrame

‘Affettare’ un DataFrame

Selezione righe:

- `df.iloc[0:2]` → Righe 0 and 1 (esclude 2)
- `df.loc[0:2]` → Righe 0, 1, and 2 (include 2)

Selezione colonne:

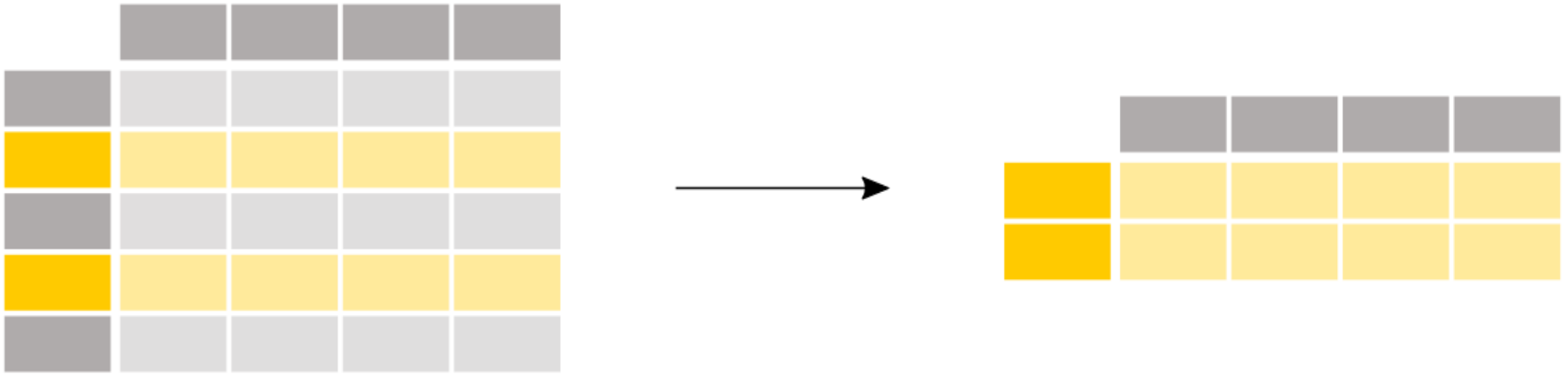
- `df.iloc[0:3]` → Colonne in posizioni 0, 1, 2
- `df.loc['Name':'Sex']` → Colonne da Name a Sex

‘Affettare’ un DataFrame

```
print(df.iloc[0:2, 0:3])
print("\n")
print(df.loc[2:])
print("\n")
print(df.loc[0:2, 'Name':'Sex'])
print("\n")
print(df.loc[0:2][['Age','Location']])
```

```
Un dataframe creato da un dizionario Python
                      Name  Age     Sex  Location
0   Braund, Mr. Owen Harris   22    male      Rome
1  Allen, Mr. William Henry   35    male    London
2  Bonnell, Miss. Elizabeth   58  female    Berlin
3       Taylor, Miss. Jane   55  female  New York


                      Name  Age   Sex
0   Braund, Mr. Owen Harris   22  male
1  Allen, Mr. William Henry   35  male


                      Name  Age     Sex  Location
2  Bonnell, Miss. Elizabeth   58  female    Berlin
3       Taylor, Miss. Jane   55  female  New York


                      Name  Age     Sex
0   Braund, Mr. Owen Harris   22    male
1  Allen, Mr. William Henry   35    male
2  Bonnell, Miss. Elizabeth   58  female


   Age Location
0   22     Rome
1   35   London
2   58   Berlin
```

‘Filtrare’ un DataFrame



‘Filtrare’ un DataFrame

Definire ed applicare una condizione:

- `condition_1 = df['Age'] >= 40`
- `df[condition_1]`

Definire ed applicare più condizioni:

- `condition_2 = df['Location'] == 'New York'`
- `df[condition_1 & condition_2]`

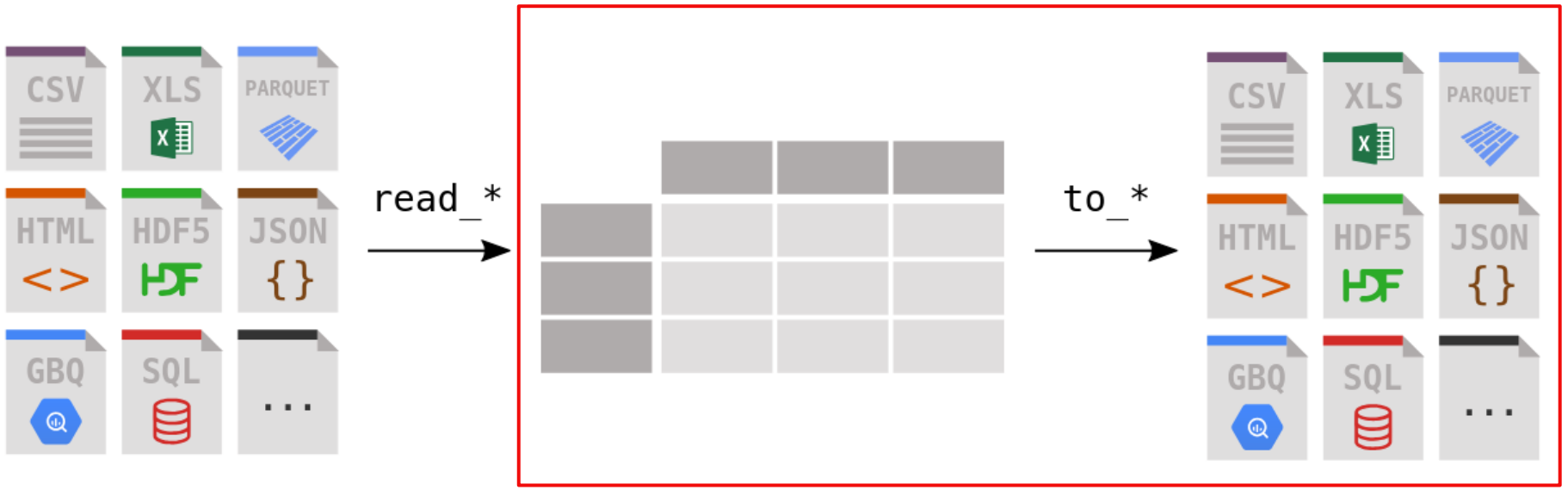
‘Filtrare’ un DataFrame

```
simple_condition_1 = df['Age'] >= 40
print(simple_condition_1)
print("\n")
print(df[simple_condition_1])
print("\n")
simple_condition_2 = df['Location'].str.contains('b', case=False)
print(simple_condition_2)
print("\n")
print(df[simple_condition_1 & simple_condition_2])
```

```
Un dataframe creato da un dizionario Python
                        Name  Age     Sex  Location
0   Braund, Mr. Owen Harris   22    male      Rome
1  Allen, Mr. William Henry   35    male    London
2  Bonnell, Miss. Elizabeth   58  female    Berlin
3        Taylor, Miss. Jane   55  female  New York


0    False
1    False
2     True
3     True
Name: Age, dtype: bool


                        Name  Age     Sex  Location
2  Bonnell, Miss. Elizabeth   58  female    Berlin
3        Taylor, Miss. Jane   55  female  New York


0    False
1    False
2     True
3    False
Name: Location, dtype: bool


                        Name  Age     Sex Location
2  Bonnell, Miss. Elizabeth   58  female   Berlin
```

its ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

Scrittura di un DataFrame



- In un dizionario Python

```
df.to_dict()
```

- In un file (es. csv, Excel)

```
df.to_csv("../dati/output.csv", index=False)
df.to_excel("titanic.xlsx", sheet_name="passengers", index=False)
```

- In una tabella di un database

Parte 2


its
ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

INDICE DEGLI ARGOMENTI

- Data wrangling
  - Pulizia
  - Accesso multisorgente
  - Fusione ed aggregazione
- Prime pipeline di analisi dati

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

```python
# "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

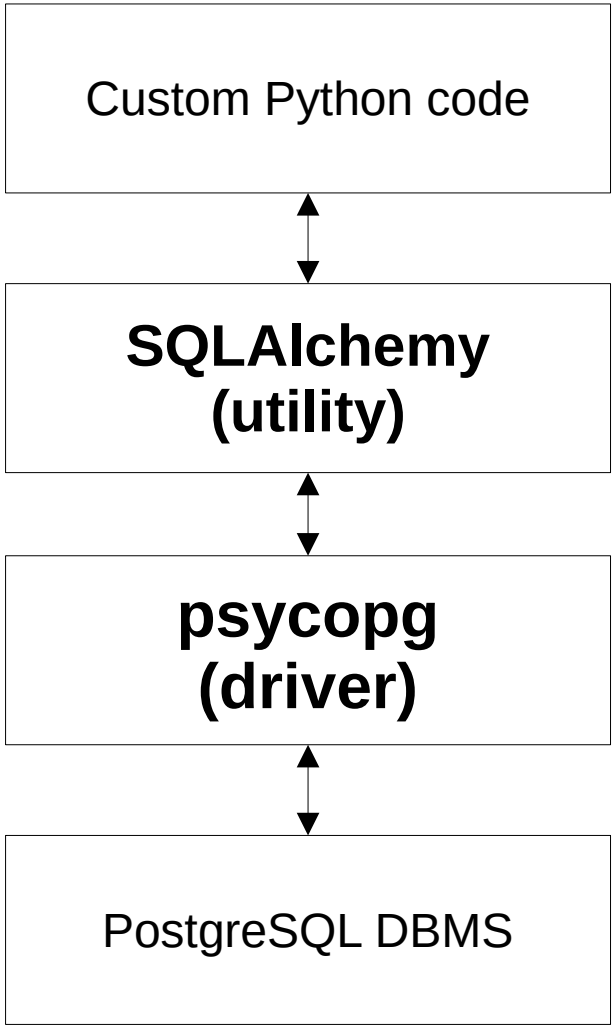
Accesso a più sorgenti di dati

- 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



SQLAlchemy

```
from sqlalchemy import create_engine, text
from sqlalchemy.exc import SQLAlchemyError
```

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
```

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

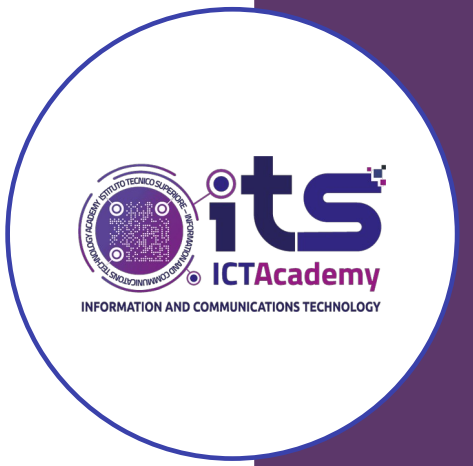
Es. Auto usate (da /Lezione1/codice/autos/autos_data_pipeline.py)

```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")
    # Pulizia dati
    clean_df = self.clean_data(db_df)
    print("   -Pulizia dati completata e file pulito salvato")
    # Visualizza risultati
    self.visualize(clean_df)
    print("   -Analisi e visualizzazione dati terminate")
    self.data = clean_df
    return clean_df
```

Data pipeline

Es. Auto usate (da /Lezione1/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")
    # Pulizia dati
    clean_df = self.clean_data(db_df)
    print("   -Pulizia dati completata e file pulito salvato")
    # Visualizza risultati
    self.visualize(clean_df)
    print("   -Analisi e visualizzazione dati terminate")
    self.data = clean_df
    return clean_df
```



Data pipeline

Es. Auto usate (da /Lezione1/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")
    # Pulizia dati
    clean_df = self.clean_data(db_df)
    print("   -Pulizia dati completata e file pulito salvato")
    # Visualizza risultati
    self.visualize(clean_df)
    print("   -Analisi e visualizzazione dati terminate")
    self.data = clean_df
    return clean_df
```

ICTAcademy
INFORMATION AND COMMUNICATIONS TECHNOLOGY

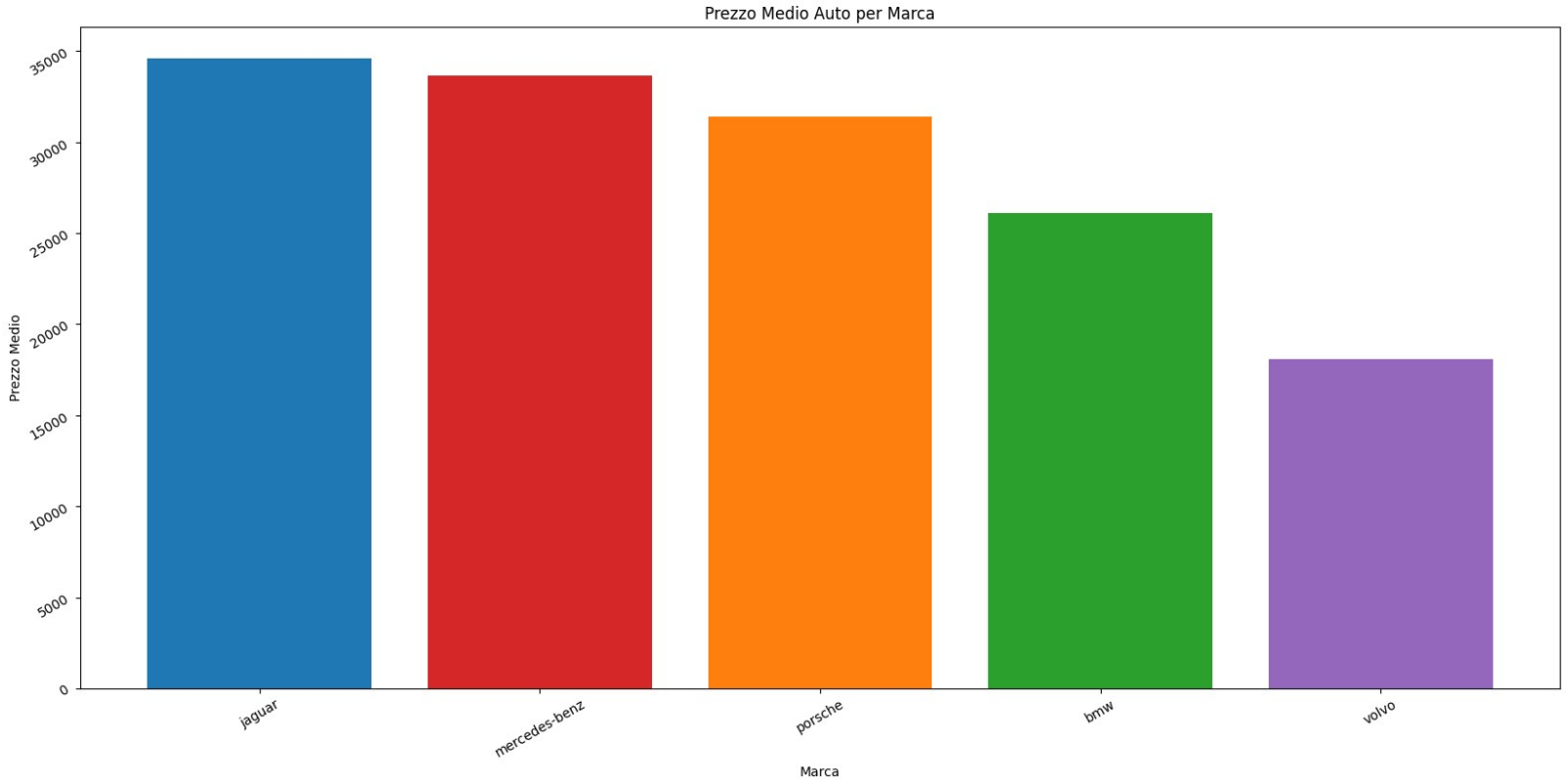
Data pipeline

Es. Auto usate (da /Lezione1/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")
    # Pulizia dati
    clean_df = self.clean_data(db_df)
    print("   -Pulizia dati completata e file pulito salvato")
    # Visualizza risultati
    self.visualize(clean_df)
    print("   -Analisi e visualizzazione dati terminate")
    self.data = clean_df
    return clean_df
```

Data pipeline

Es. Auto usate (da /Lezione1/codice/autos/autos_data_pipeline.py)



Data pipeline

Es. Titanic (da /Lezione1/codice/titanic/titanic_data_pipeline.py)

```python
def run_pipeline(self) -> pd.DataFrame:
    """Esegue la pipeline completa"""
    # Carica dati
    db_df1, db_df2 = self.load_from_database()
    print("   -Letti dati da due tabelle su db")
    json_df = self.load_from_json()
    print("   -Letti dati da un file JSON")
    # Preprocessa dati
    merged_df = self.merge_data(db_df1, db_df2, json_df)
    expanded_df = self.expand_json_data(merged_df)
    print("   -Aggregazione ed espansione dati effettuate")
    clean_df = self.clean_data(expanded_df)
    print("   -Pulizia dati completata")
    # Visualizza risultati
    self.visualize(clean_df)
    print("   -Analisi e visualizzazione dati terminate")
    self.data = clean_df
    return clean_df
```

Data pipeline

Es. Titanic (da /Lezione1/codice/titanic/titanic_data_pipeline.py)

```python
def run_pipeline(self) -> pd.DataFrame:
    """Esegue la pipeline completa"""
    # Carica dati
    db_df1, db_df2 = self.load_from_database()
    print("   -Letti dati da due tabelle su db")
    json_df = self.load_from_json()
    print("   -Letti dati da un file JSON")
    # Preprocessa dati
    merged_df = self.merge_data(db_df1, db_df2, json_df)
    expanded_df = self.expand_json_data(merged_df)
    print("   -Aggregazione ed espansione dati effettuate")
    clean_df = self.clean_data(expanded_df)
    print("   -Pulizia dati completata")
    # Visualizza risultati
    self.visualize(clean_df)
    print("   -Analisi e visualizzazione dati terminate")
    self.data = clean_df
    return clean_df
```

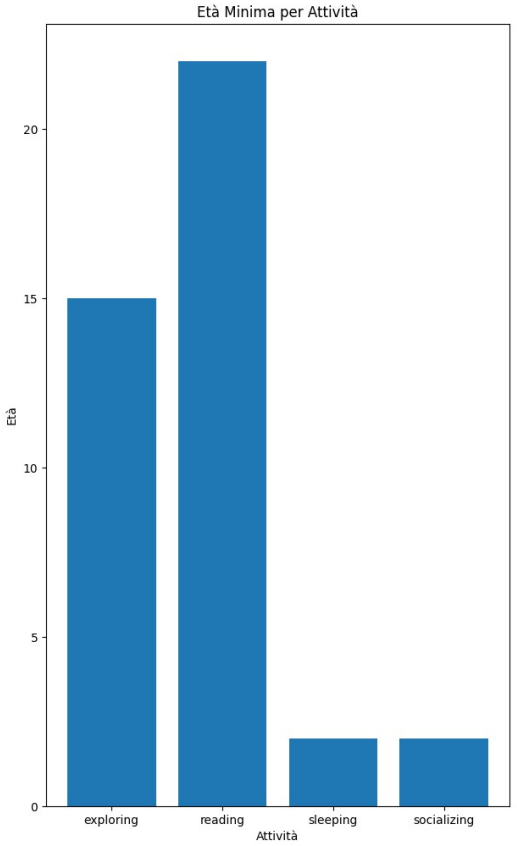
Data pipeline

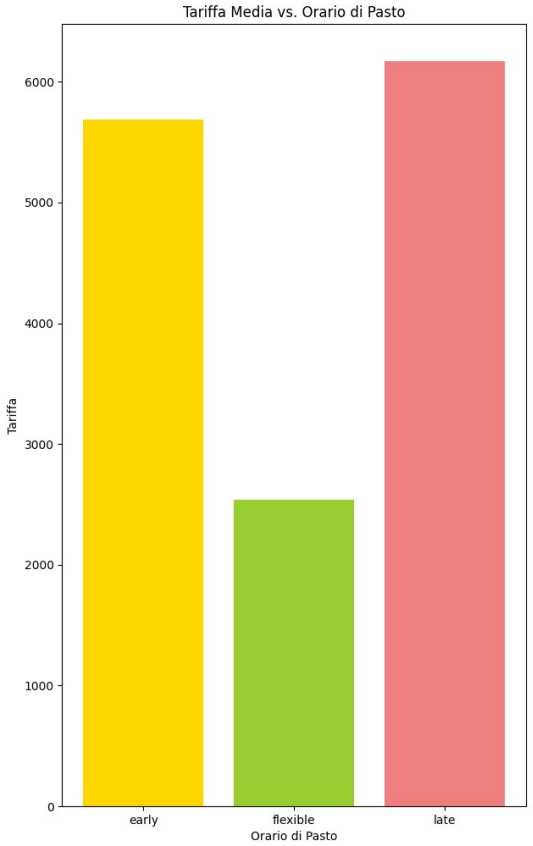
Es. Titanic (da /Lezione1/codice/titanic/titanic_data_pipeline.py)

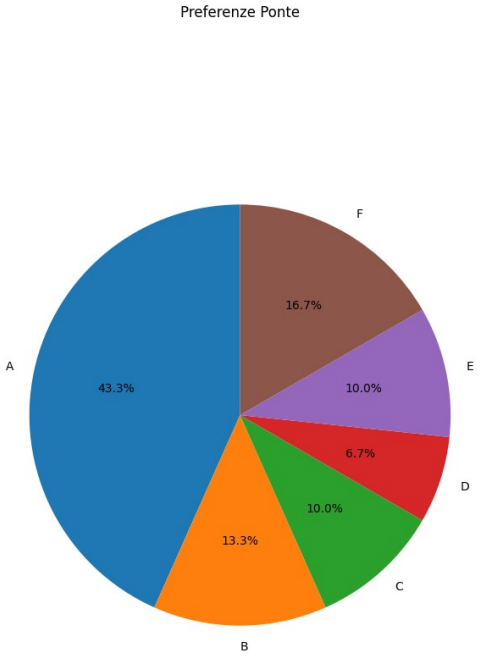
```python
def run_pipeline(self) -> pd.DataFrame:
    """Esegue la pipeline completa"""
    # Carica dati
    db_df1, db_df2 = self.load_from_database()
    print("   -Letti dati da due tabelle su db")
    json_df = self.load_from_json()
    print("   -Letti dati da un file JSON")
    # Preprocessa dati
    merged_df = self.merge_data(db_df1, db_df2, json_df)
    expanded_df = self.expand_json_data(merged_df)
    print("   -Aggregazione ed espansione dati effettuate")
    clean_df = self.clean_data(expanded_df)
    print("   -Pulizia dati completata")
    # Visualizza risultati
    self.visualize(clean_df)
    print("   -Analisi e visualizzazione dati terminate")
    self.data = clean_df
    return clean_df
```

Data pipeline

Es. Titanic (da /Lezione1/codice/titanic/titanic_data_pipeline.py)







RIFERIMENTI BIBLIOGRAFICI

- click to pandas documentation
- click to SQLAlchemy documentation