PANDAS RIFERIMENTO RAPIDO

DataFrame, selezione, aggregazione, unione e altro

DataFrame

Creazione di DataFrame

```
import pandas as pd
df = pd.DataFrame({
    "name": ["Alice", "Bob", "Carol"],
    "age": [25, 30, 35],
    "score": [88, 92, 79]
})
```

Ispezione

`df.head(n)`	Prime n righe (default 5)
`df.tail(n)`	Ultime n righe
`df.shape`	Tupla di (righe, colonne)
`df.dtypes`	Tipo di dato di ogni colonna
`df.info()`	Tipi colonne, conteggi non-null
`df.describe()`	Statistiche per colonne numeriche
`df.columns`	Nomi delle colonne come Index
`df.index`	Etichette delle righe

Lettura Dati

Lettori Comuni

```
df = pd.read_csv("data.csv")
df = pd.read_excel("data.xlsx")
df = pd.read_json("data.json")
df = pd.read_sql(query, connection)
```

Scrittura Dati

```
df.to_csv("out.csv", index=False)
df.to_excel("out.xlsx", index=False)
df.to_json("out.json", orient="records")
```

Opzioni di Lettura

`sep=";"`	Delimitatore personalizzato
`header=None`	Nessuna riga di intestazione nel file
`usecols=[0,2]`	Legge solo colonne specifiche
`nrows=100`	Legge le prime 100 righe
`na_values=["N/A"]`	Tratta come NaN

Selezione

Colonne

```
df["name"]              # colonna singola (Series)
df[["name", "age"]]     # colonne multiple (DataFrame)
df.name                 # accesso attributo (nomi semplici)
```

Righe con loc / iloc

```
df.loc[0]             # riga per etichetta
df.loc[0:2, "name"]   # righe 0-2, colonna "name"
df.iloc[0]            # riga per posizione
df.iloc[0:2, 0:2]     # prime 2 righe, 2 colonne
```

loc vs iloc

`df.loc[row, col]`	Seleziona per **etichetta** (fine inclusa)
`df.iloc[row, col]`	Seleziona per **posizione** (fine esclusa)
`df.at[row, col]`	Accesso scalare veloce per etichetta
`df.iat[row, col]`	Accesso scalare veloce per posizione

Filtraggio

Filtraggio Booleano

```
df[df["age"] > 25]
df[df["name"].str.contains("li")]
df[(df["age"] > 25) & (df["score"] > 80)]
df[df["name"].isin(["Alice", "Bob"])]
```

Gestione dei Dati Mancanti

```
df.isna().sum()          # conteggio NaN per colonna
df.dropna()              # rimuovi righe con NaN
df.fillna(0)             # riempi NaN con 0
df["col"].fillna(df["col"].mean())
```

Ordinamento

```
df.sort_values("age")                # crescente
df.sort_values("age", ascending=False)
df.sort_values(["age", "score"])     # multiplo
```

Aggregazione

Aggregazioni Comuni

`df["col"].sum()`	Somma della colonna
`df["col"].mean()`	Media
`df["col"].median()`	Mediana
`df["col"].std()`	Deviazione standard
`df["col"].min() / .max()`	Minimo / massimo
`df["col"].count()`	Conteggio non-null
`df["col"].nunique()`	Numero di valori unici
`df["col"].value_counts()`	Frequenza di ogni valore

Aggregazioni Multiple

```
df.agg({"age": "mean", "score": ["min", "max"]})
df.describe()  # statistiche sommario per tutti i numerici
```

GroupBy

Raggruppamento di Base

```
df.groupby("dept")["salary"].mean()
df.groupby("dept").agg(
    avg_sal=("salary", "mean"),
    count=("salary", "count")
)
```

Gruppi Multipli

```
df.groupby(["dept", "year"])["sales"].sum()
df.groupby("dept").size()  # righe per gruppo
```

Transform e Apply

```
df["z_score"] = df.groupby("dept")["salary"] \
    .transform(lambda x: (x - x.mean()) / x.std())
df.groupby("dept").apply(lambda g: g.nlargest(3, "salary"))
```

Unione

Merge (Join in stile SQL)

```
pd.merge(df1, df2, on="id")          # inner
pd.merge(df1, df2, on="id", how="left")
pd.merge(df1, df2, left_on="uid",
         right_on="user_id")
```

Tipi di Join

`how="inner"`	Mantiene solo le righe corrispondenti (default)
`how="left"`	Mantiene tutte le righe sinistra, NaN se non c'è corrispondenza
`how="right"`	Mantiene tutte le righe destra
`how="outer"`	Mantiene tutte le righe da entrambi i lati

Concatenazione

```
pd.concat([df1, df2])            # impila righe
pd.concat([df1, df2], axis=1)    # affiancato
pd.concat([df1, df2], ignore_index=True)
```

Tabelle Pivot

Tabella Pivot

```
df.pivot_table(
    values="sales", index="region",
    columns="quarter", aggfunc="sum"
)
```

Reshaping

```
df.melt(id_vars=["name"],
        value_vars=["q1", "q2"],
        var_name="quarter", value_name="sales")
```

Tabulazione Incrociata

```
pd.crosstab(df["dept"], df["gender"])
pd.crosstab(df["dept"], df["gender"],
            normalize="index")  # percentuali per riga
```

Serie Temporali

Basi di DateTime

```
df["date"] = pd.to_datetime(df["date"])
df["year"] = df["date"].dt.year
df["month"] = df["date"].dt.month
df["weekday"] = df["date"].dt.day_name()
```

Intervalli di Date e Ricampionamento

```
pd.date_range("2025-01-01", periods=12, freq="ME")
df.set_index("date").resample("ME")["sales"].sum()
```

Attributi dell'Accessor

`.dt.year / .dt.month / .dt.day`	Estrae componenti della data
`.dt.hour / .dt.minute`	Estrae componenti dell'ora
`.dt.day_name()`	Nome del giorno della settimana (Lunedì, ecc.)
`.dt.days_in_month`	Giorni nel mese

Pattern Comuni

Rinomina Colonne

```
df.rename(columns={"old": "new"})
df.columns = ["a", "b", "c"]  # sostituisce tutte
```

Aggiunge / Modifica Colonne

```
df["total"] = df["q1"] + df["q2"]
df["grade"] = df["score"].apply(
    lambda x: "A" if x >= 90 else "B"
)
```

Rimuove Colonne / Righe

```
df.drop(columns=["temp"])
df.drop_duplicates(subset=["name"])
df.reset_index(drop=True)
```

Operazioni su Stringhe

```
df["name"].str.lower()
df["name"].str.contains("ali", case=False)
df["name"].str.split(" ").str[0]  # primo nome
```