

# 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

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

## Letture 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

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

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

<b>how="inner"</b>	Mantiene solo le righe corrispondenti (default)
<b>how="left"</b>	Mantiene tutte le righe sinistra, NaN se non c'è corrispondenza
<b>how="right"</b>	Mantiene tutte le righe destra
<b>how="outer"</b>	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)
```

## Table 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 Temporal

### 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

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