Référence rapide pandas

DataFrames, sélection, agrégation, fusion et plus

DataFrames

Créer des DataFrames

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

Inspection

`df.head(n)`	n premières lignes (5 par défaut)
`df.tail(n)`	n dernières lignes
`df.shape`	Tuple de (lignes, colonnes)
`df.dtypes`	Type de données de chaque colonne
`df.info()`	Types de colonnes, comptes non nuls
`df.describe()`	Statistiques pour les colonnes numériques
`df.columns`	Noms des colonnes en tant qu'Index
`df.index`	Étiquettes des lignes

Lecture des données

Lecteurs courants

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

Écriture des données

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

Options de lecture

`sep=";"`	Délimiteur personnalisé
`header=None`	Pas de ligne d'en-tête dans le fichier
`usecols=[0,2]`	Lire uniquement des colonnes spécifiques
`nrows=100`	Lire les 100 premières lignes
`na_values=["N/A"]`	Traiter comme NaN

Sélection

Colonnes

```
df["name"]              # colonne unique (Series)
df[["name", "age"]]     # colonnes multiples (DataFrame)
df.name                 # accès par attribut (noms simples)
```

Lignes avec loc / iloc

```
df.loc[0]             # ligne par étiquette
df.loc[0:2, "name"]   # lignes 0-2, colonne "name"
df.iloc[0]            # ligne par position
df.iloc[0:2, 0:2]     # 2 premières lignes, 2 cols
```

loc vs iloc

`df.loc[row, col]`	Sélection par **étiquette** (fin inclusive)
`df.iloc[row, col]`	Sélection par **position** (fin exclusive)
`df.at[row, col]`	Accès scalaire rapide par étiquette
`df.iat[row, col]`	Accès scalaire rapide par position

Filtrage

Filtrage booléen

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

Gestion des données manquantes

```
df.isna().sum()          # compte NaN par colonne
df.dropna()              # supprimer les lignes avec NaN
df.fillna(0)             # remplir NaN par 0
df["col"].fillna(df["col"].mean())
```

Tri

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

Agrégation

Agrégations courantes

`df["col"].sum()`	Somme de la colonne
`df["col"].mean()`	Moyenne
`df["col"].median()`	Médiane
`df["col"].std()`	Écart-type
`df["col"].min() / .max()`	Min / max
`df["col"].count()`	Compte non nul
`df["col"].nunique()`	Nombre de valeurs uniques
`df["col"].value_counts()`	Fréquence de chaque valeur

Agrégations multiples

```
df.agg({"age": "mean", "score": ["min", "max"]})
df.describe()  # statistiques résumées pour le numérique
```

GroupBy

Regroupement de base

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

Groupes multiples

```
df.groupby(["dept", "year"])["sales"].sum()
df.groupby("dept").size()  # lignes par groupe
```

Transform et 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"))
```

Fusion

Merge (jointure style 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")
```

Types de jointures

`how="inner"`	Conserver uniquement les lignes correspondantes (défaut)
`how="left"`	Toutes les lignes gauches, NaN si pas de correspondance
`how="right"`	Toutes les lignes droites
`how="outer"`	Toutes les lignes des deux côtés

Concaténation

```
pd.concat([df1, df2])             # empiler les lignes
pd.concat([df1, df2], axis=1)     # côte à côte
pd.concat([df1, df2], ignore_index=True)
```

Tableaux croisés dynamiques

Tableau croisé dynamique

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

Remodelage

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

Tableau de contingence

```
pd.crosstab(df["dept"], df["gender"])
pd.crosstab(df["dept"], df["gender"],
            normalize="index")  # pourcentages par ligne
```

Séries temporelles

Bases 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()
```

Plages de dates et rééchantillonnage

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

Attributs d'accesseur

`.dt.year / .dt.month / .dt.day`	Extraire les composantes de date
`.dt.hour / .dt.minute`	Extraire les composantes d'heure
`.dt.day_name()`	Nom du jour de la semaine (Monday, etc.)
`.dt.days_in_month`	Jours dans ce mois

Motifs courants

Renommer les colonnes

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

Ajouter / modifier des colonnes

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

Supprimer des colonnes / lignes

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

Opérations sur les chaînes

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