pandas クイックリファレンス

DataFrame、選択、集計、マージなど

DataFrame

DataFrame の作成

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

確認

df.head(n)	最初の n 行（デフォルト 5）
df.tail(n)	最後の n 行
df.shape	（行数、列数）のタプル
df.dtypes	各列のデータ型
df.info()	列の型と非 null 値の数
df.describe()	数値列の統計情報
df.columns	列名の Index
df.index	行ラベル

データの読み込み

一般的なリーダー

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

データの書き出し

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

読み込みオプション

sep=";"	カスタム区切り文字
header=None	ファイルにヘッダー行なし
usecols=[0,2]	特定の列のみ読み込み
nrows=100	最初の 100 行を読み込み
na_values=["N/A"]	NaN として扱う値

選択

列

```
df["name"]              # 単一列（Series）
df[["name", "age"]]     # 複数列（DataFrame）
df.name                 # 属性アクセス（シンプルな列名）
```

loc / iloc による行の選択

```
df.loc[0]              # ラベルで行を選択
df.loc[0:2, "name"]    # 行0〜2、列"name"
df.iloc[0]             # 位置で行を選択
df.iloc[0:2, 0:2]     # 最初の2行、2列
```

loc vs iloc

df.loc[row, col]	**ラベル**で選択（末尾を含む）
df.iloc[row, col]	**位置**で選択（末尾を含まない）
df.at[row, col]	ラベルによる高速スカラーアクセス
df.iat[row, col]	位置による高速スカラーアクセス

フィルタリング

ブールフィルタリング

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

欠損データの処理

```
df.isna().sum()          # 列ごとのNaN数
df.dropna()              # NaNを含む行を削除
df.fillna(0)             # NaNを0で埋める
df["col"].fillna(df["col"].mean())
```

ソート

```
df.sort_values("age")                # 昇順
df.sort_values("age", ascending=False)
df.sort_values(["age", "score"])      # 複数列
```

集計

一般的な集計

df["col"].sum()	列の合計
df["col"].mean()	平均
df["col"].median()	中央値
df["col"].std()	標準偏差
df["col"].min() / .max()	最小値/最大値
df["col"].count()	非 null 値の数
df["col"].nunique()	ユニーク値の数
df["col"].value_counts()	各値の出現頻度

複数の集計

```
df.agg({"age": "mean", "score": ["min", "max"]})
df.describe()  # 全数値列のサマリー統計
```

GroupBy

基本的なグループ化

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

複数グループ

```
df.groupby(["dept", "year"])["sales"].sum()
df.groupby("dept").size()  # グループごとの行数
```

Transform と 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"))
```

マージ

マージ（SQL スタイルのジョイン）

```
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")
```

ジョインの種類

how="inner"	一致する行のみ保持（デフォルト）
how="left"	左の全行、一致なしは NaN
how="right"	右の全行を保持
how="outer"	両側の全行を保持

連結

```
pd.concat([df1, df2])             # 行をスタック
pd.concat([df1, df2], axis=1)     # 横に並べる
pd.concat([df1, df2], ignore_index=True)
```

ピボットテーブル

ピボットテーブル

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

形状変換

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

クロス集計

```
pd.crosstab(df["dept"], df["gender"])
pd.crosstab(df["dept"], df["gender"],
            normalize="index")  # 行の割合
```

時系列

日時の基本

```
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()
```

日付範囲とリサンプリング

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

アクセサの属性

.dt.year / .dt.month / .dt.day	日付コンポーネントの抽出
.dt.hour / .dt.minute	時刻コンポーネントの抽出
.dt.day_name()	曜日名（月曜日など）
.dt.days_in_month	その月の日数

よくあるパターン

列名の変更

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

列の追加/変更

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

列/行の削除

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

pandas クイックリファレンス

文字列操作

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