PANDAS 빠른 참조

DataFrame, 선택, 집계, 병합 등

DataFrame

DataFrame 생성

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

검사

`df.head(n)`	처음 n행 (기본값 5)
`df.tail(n)`	마지막 n행
`df.shape`	(행, 열) 튜플
`df.dtypes`	각 열의 데이터 타입
`df.info()`	열 타입, 비null 개수
`df.describe()`	수치 열 통계
`df.columns`	Index로 나타낸 열 이름
`df.index`	행 레이블

데이터 읽기

주요 리더

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

데이터 쓰기

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

읽기 옵션

`sep=";"`	사용자 정의 구분자
`header=None`	파일에 헤더 행 없음
`usecols=[0,2]`	특정 열만 읽기
`nrows=100`	처음 100행 읽기
`na_values=["N/A"]`	NaN으로 처리

선택

열

```
df["name"]              # single column (Series)
df[["name", "age"]]     # multiple columns (DataFrame)
df.name                 # attribute access (simple names)
```

loc / iloc으로 행 선택

```
df.loc[0]             # row by label
df.loc[0:2, "name"]   # rows 0-2, column "name"
df.iloc[0]            # row by position
df.iloc[0:2, 0:2]     # first 2 rows, 2 cols
```

loc vs iloc

`df.loc[row, col]`	**레이블**로 선택 (끝 포함)
`df.iloc[row, col]`	**위치**로 선택 (끝 미포함)
`df.at[row, col]`	레이블로 빠른 스칼라 접근
`df.iat[row, col]`	위치로 빠른 스칼라 접근

필터링

불리언 필터링

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

결측값 처리

```
df.isna().sum()          # NaN count per column
df.dropna()              # drop rows with any NaN
df.fillna(0)             # fill NaN with 0
df["col"].fillna(df["col"].mean())
```

정렬

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

집계

주요 집계

`df["col"].sum()`	열 합계
`df["col"].mean()`	평균
`df["col"].median()`	중앙값
`df["col"].std()`	표준 편차
`df["col"].min() / .max()`	최솟값 / 최댓값
`df["col"].count()`	비null 개수
`df["col"].nunique()`	고유값 수
`df["col"].value_counts()`	각 값의 빈도

다중 집계

```
df.agg({"age": "mean", "score": ["min", "max"]})
df.describe()  # summary stats for all numeric
```

GroupBy

기본 그룹화

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

다중 그룹

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

Transform 및 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"))
```

병합

Merge (SQL 스타일 조인)

```
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")
```

조인 유형

`how="inner"`	일치하는 행만 유지 (기본값)
`how="left"`	왼쪽 행 모두 유지, 불일치는 NaN
`how="right"`	오른쪽 행 모두 유지
`how="outer"`	양쪽 행 모두 유지

연결

```
pd.concat([df1, df2])            # stack rows
pd.concat([df1, df2], axis=1)    # side by side
pd.concat([df1, df2], ignore_index=True)
```

피벗 테이블

피벗 테이블

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

형태 변환

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

교차 집계

```
pd.crosstab(df["dept"], df["gender"])
pd.crosstab(df["dept"], df["gender"],
            normalize="index")  # row percentages
```

시계열

DateTime 기본

```
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()
```

날짜 범위 및 리샘플링

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

접근자 속성

`.dt.year / .dt.month / .dt.day`	날짜 구성 요소 추출
`.dt.hour / .dt.minute`	시간 구성 요소 추출
`.dt.day_name()`	요일 이름 (Monday 등)
`.dt.days_in_month`	해당 월의 일 수

일반 패턴

열 이름 변경

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

열 추가 / 수정

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

열 / 행 제거

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

문자열 연산

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