REFERENCIA RÁPIDA DE PANDAS

DataFrames, selección, agregación, unión y más

DataFrames

Crear DataFrames

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

Inspección

`df.head(n)`	Primeras n filas (por defecto 5)
`df.tail(n)`	Últimas n filas
`df.shape`	Tupla (filas, columnas)
`df.dtypes`	Tipo de dato de cada columna
`df.info()`	Tipos de columna, conteos no nulos
`df.describe()`	Estadísticas de columnas numéricas
`df.columns`	Nombres de columnas como Índice
`df.index`	Etiquetas de filas

Lectura de Datos

Lectores Comunes

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

Escritura de Datos

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

Opciones de Lectura

`sep=";"`	Delimitador personalizado
`header=None`	Sin fila de encabezado en el archivo
`usecols=[0,2]`	Leer solo columnas específicas
`nrows=100`	Leer las primeras 100 filas
`na_values=["N/A"]`	Tratar como NaN

Selección

Columnas

```
df["name"]              # columna simple (Series)
df[["name", "age"]]     # múltiples columnas (DataFrame)
df.name                 # acceso por atributo (nombres simples)
```

Filas con loc / iloc

```
df.loc[0]             # fila por etiqueta
df.loc[0:2, "name"]   # filas 0-2, columna "name"
df.iloc[0]            # fila por posición
df.iloc[0:2, 0:2]     # primeras 2 filas, 2 cols
```

loc vs iloc

`df.loc[fila, col]`	Seleccionar por **etiqueta** (fin inclusivo)
`df.iloc[fila, col]`	Seleccionar por **posición** (fin exclusivo)
`df.at[fila, col]`	Acceso escalar rápido por etiqueta
`df.iat[fila, col]`	Acceso escalar rápido por posición

Filtrado

Filtrado Booleano

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

Manejo de Datos Faltantes

```
df.isna().sum()          # conteo de NaN por columna
df.dropna()              # eliminar filas con cualquier NaN
df.fillna(0)             # rellenar NaN con 0
df["col"].fillna(df["col"].mean())
```

Ordenamiento

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

Agregación

Agregaciones Comunes

`df["col"].sum()`	Suma de la columna
`df["col"].mean()`	Media
`df["col"].median()`	Mediana
`df["col"].std()`	Desviación estándar
`df["col"].min() / .max()`	Mínimo / máximo
`df["col"].count()`	Conteo no nulo
`df["col"].nunique()`	Número de valores únicos
`df["col"].value_counts()`	Frecuencia de cada valor

Múltiples Agregaciones

```
df.agg({"age": "mean", "score": ["min", "max"]})
df.describe()  # estadísticas resumen de todos los numéricos
```

GroupBy

Agrupación Básica

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

Múltiples Grupos

```
df.groupby(["dept", "year"])["sales"].sum()
df.groupby("dept").size()  # filas por grupo
```

Transform y 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"))
```

Unión (Merging)

Merge (Unión estilo 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")
```

Tipos de Unión

`how="inner"`	Conservar solo filas coincidentes (predeterminado)
`how="left"`	Conservar todas las filas izquierdas, NaN donde no hay coincidencia
`how="right"`	Conservar todas las filas derechas
`how="outer"`	Conservar todas las filas de ambos lados

Concatenación

```
pd.concat([df1, df2])            # apilar filas
pd.concat([df1, df2], axis=1)    # uno al lado del otro
pd.concat([df1, df2], ignore_index=True)
```

Tablas Dinámicas

Tabla Dinámica

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

Reorganización

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

Tabulación Cruzada

```
pd.crosstab(df["dept"], df["gender"])
pd.crosstab(df["dept"], df["gender"],
            normalize="index")  # porcentajes por fila
```

Series de Tiempo

Fundamentos de Fechas

```
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()
```

Rangos de Fechas y Remuestreo

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

Atributos del Accessor

`.dt.year / .dt.month / .dt.day`	Extraer componentes de fecha
`.dt.hour / .dt.minute`	Extraer componentes de hora
`.dt.day_name()`	Nombre del día de la semana (lunes, etc.)
`.dt.days_in_month`	Días en ese mes

Patrones Comunes

Renombrar Columnas

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

Agregar / Modificar Columnas

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

Eliminar Columnas / Filas

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

Operaciones con Cadenas

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