Introduction to Pandas

Series vs DataFrame

- A column in a DataFrame is a **Series** object
- Accessing a Series: `df['col']`
- Accessing a DataFrame: `df[['col']]`

```
import pandas as pd
df = pd.DataFrame({'col': [1, 2, 3]})
type(df['col'])     # pandas.Series
type(df[['col']])   # pandas.DataFrame
```

First Look at a DataFrame

- `df.head()`
- `df.tail()`
- `df.describe()`
- `df.index`
- `df.sort_index()`
- `df.sort_values()`

```
df.head()
df.describe()
df.sort_values('col')
```

⚠️ DataFrames are **mutable**.

Selecting Data

- Row slicing: `df[0:3]` → returns a **DataFrame**
- Versatile accessors: `.loc`, `.iloc`
- Single value access: `.at`, `.iat`

```
df.loc[0, 'col']   # label-based
df.iloc[0, 0]      # index-based
df.at[0, 'col']    # faster, label-based
df.iat[0, 0]       # faster, index-based
```

Boolean Masks

1. Create a mask using a condition
2. Apply the mask to the data

```
mask = df['col'] > 1
filtered = df[mask]
```

Setting New Values

- Use `.at` or `.loc` to change data
- ⚠️ Watch out for slicing pitfalls with Series

```
df.at[0, 'col'] = 100
df.loc[df['col'] > 2, 'col'] = 999
```

Missing Values

Key methods:

- `.dropna()`
- `.fillna()`
- `.isna()`

```
df.dropna()
df.fillna(0)
df.isna().sum()
```

Custom Transformations

Key methods:

- `.agg()` for aggregations
- `.transform()` for element-wise operations

```
df.groupby('group')['value'].agg(['mean', 'sum'])
df['value'].transform(lambda x: x**2)
```

Concatenating DataFrames

Key methods:

- `pd.concat()`
- `pd.merge()`

❌ Avoid building DataFrames row-by-row with `concat`

```
pd.concat([df1, df2], axis=0)
pd.merge(df1, df2, on='id')
```

GroupBy

Implements the **split → apply → combine** paradigm

- Can return **MultiIndex DataFrames**

```
grouped = df.groupby(['group_col']).agg({'value': 'mean'})
```

⚠️ Multi-index results can be tricky to handle

Making DataFrames Long (Narrower)

Use when **columns contain values**, not variable names

- `melt()` creates new columns
- `stack()` creates a new index

```
pd.melt(df, id_vars=['id'], value_vars=['a', 'b'])
df.stack()
```

Making DataFrames Wide

Use when **rows contain values that should be columns**

- `pivot()` or `pivot_table()`

```
df.pivot(index='id', columns='category', values='value')
pd.pivot_table(df, index='id', columns='category', values='value', aggfunc='sum')
```

`pivot_table()` is more flexible

Categories

- Use `.astype('category')` to convert columns
- Supports **missing categories**

```
df['col'] = df['col'].astype('category')
df['col'].cat.categories
```