

Python

Manipulation & Aggregation

Alish Bista - 11/11/2025

Data Manipulation

Data Manipulation

Why Manipulate?

- Raw data is not always analysis ready.
- Common Problems:
 - Missing values (NaN, empty cells)
 - Wrong data types (numbers stored as text)
 - Data needs aggregation (sales by region)
 - Duplicate records
 - Inconsistent formatting

```
import pandas as pd

raw_data = {
    'Name': ['Alice', 'Bob',
None, 'David'],
    'Sales': ['100', '150',
'200', 'N/A'],
    'Region': ['north',
'SOUTH', 'East', 'north']
}

df_raw = pd.DataFrame(raw_data)
print(df_raw)
```

Loading Data

Bringing Data In

- Pandas can read from multiple sources: CSV, Excel, databases, APIs, and more.
- Common Functions:
 - `read_csv()` - CSV files
 - `read_excel()` - Excel files
 - `read_json()` - JSON data

```
import pandas as pd

# Load CSV
df = pd.read_csv('sales_data.csv')

# Load Excel
df = pd.read_excel('sales_data.xlsx')

print(df.shape)
print(df.head())
print(df.tail())
print(df.info())
print(df.describe())

print(df.columns)
```

Find Missing Data

Identifying Missing Data

- Missing data appears as NaN (Not a Number) in pandas.
- Detection Methods:
 - `isnull()` - Returns True/False for each cell
 - `sum()` - Count missing values per column
 - `any()` - Check if any nulls exist

```
import pandas as pd

df = pd.DataFrame({
    'Name': ['Alice', 'Bob', None, 'David'],
    'Age': [25, None, 30, 28],
    'Sales': [100, 150, None, 200]
})

print(df.isnull())
print(df.isnull().sum())
print(df.isnull().sum().sum())
print(df.isnull().any())
```

Handle Missing Data

Drop The Missing

- Use when missing data is minimal and won't significantly reduce your dataset.
- `dropna()` - Remove rows/columns with nulls
- Parameters:
 - `axis=0` - Drop rows (default)
 - `axis=1` - Drop columns
 - `how='any'` - Drop if any null (default)
 - `how='all'` - Drop only if all nulls

```
df = pd.DataFrame({  
    'Name': ['Alice', 'Bob', None, 'David'],  
    'Age': [25, None, 30, 28],  
    'Sales': [100, 150, 200, 200]  
})
```

```
df_clean = df.dropna()  
print(df_clean)
```

```
df_clean = df.dropna(subset=['Name'])  
print(df_clean)
```

```
df_clean = df.dropna(axis=1)  
print(df_clean)
```

```
df.dropna(inplace=True)
```

Handle Missing Data

Replace The Missing

- Use when you want to preserve data but need to replace nulls with reasonable values.
- Common Fill Strategies:
 - Fill with a constant (0, “Unknown”)
 - Fill with mean/median (numeric columns)
 - Fill with mode (most common value)

```
df = pd.DataFrame({
    'Name': ['Alice', 'Bob', None, 'David'],
    'Age': [25, None, 30, 28],
    'Sales': [100, 150, None, 200]
})

df['Name'].fillna('Unknown', inplace=True)

mean_age = df['Age'].mean()
df['Age'].fillna(mean_age, inplace=True)

median_sales = df['Sales'].median()
df['Sales'].fillna(median_sales, inplace=True)

df.fillna({
    'Name': 'Unknown',
    'Age': df['Age'].mean(),
    'Sales': 0
})

print(df)
```

Data Type Conversion

Converting to Correct Data Types

- Wrong data types cause errors and prevent proper analysis.
- Common Issues:
 - Numbers stored as strings (“100” vs 100)
 - Dates stored as strings
 - Categories stored as objects

```
df = pd.DataFrame({
    'Product': ['A', 'B', 'A', 'C', 'B'],
    'Sales': ['100', '150', '200', '180', '120'],
    'Date': ['2024-01-01', '2024-01-02',
            '2024-01-03', '2024-01-04',
            '2024-01-05']
})

print(df.dtypes)

df['Sales'] = pd.to_numeric(df['Sales'])

df['Date'] = pd.to_datetime(df['Date'])

df['Product'] = df['Product'].astype('category')

print(df.dtypes)
```

group by

Grouping Together

- `groupby()` is one of the most powerful pandas operations - like Excel pivot tables.
- Common Aggregations:
 - `sum()` - Total per group
 - `mean()` - Average per group
 - `count()` - Count per group
 - `min()`, `max()` - Extremes per group

```
df = pd.DataFrame({  
    'Region': ['North', 'South', 'North', 'South', 'East'],  
    'Sales': [100, 150, 200, 180, 120],  
    'Orders': [5, 8, 10, 7, 6]  
})
```

```
total = df.groupby('Region')['Sales'].sum()  
print(total)
```

```
avg = df.groupby('Region')['Sales'].mean()
```

```
summary = df.groupby('Region').agg({  
    'Sales': ['sum', 'mean', 'count'],  
    'Orders': 'sum'  
})  
print(summary)
```

```
result = df.groupby('Region')['Sales'].sum().reset_index()
```

Q&A