**Working with Dates and Times in Pandas**

* pd.to\_datetime()
* Extracting date parts: .dt.day, .dt.month, .dt.year
* pd.date\_range()
* Filtering by date/time

**All explained with real-world examples.**

**1. pd.to\_datetime()**

**Purpose:**

Convert string/object columns into proper datetime format.

**Real-Life Scenario:**

You get a dataset from an online store where "Order Date" is in text format. To analyze it by date, you must convert it.

**Example:**

import pandas as pd

data = {

'Order\_ID': [101, 102, 103],

'Order\_Date': ['2023-07-01', '2023-07-05', '2023-07-10']

}

df = pd.DataFrame(data)

# Convert to datetime

df['Order\_Date'] = pd.to\_datetime(df['Order\_Date'])

print(df.dtypes)

**Output:**

Order\_Date becomes a datetime64[ns] column.

**2. Extracting Date Parts**

**Purpose:**

Extract day, month, year, weekday, hour, etc.

**Example:**

df['Day'] = df['Order\_Date'].dt.day

df['Month'] = df['Order\_Date'].dt.month

df['Year'] = df['Order\_Date'].dt.year

df['Weekday'] = df['Order\_Date'].dt.day\_name()

print(df)

**Output:**

| **Order\_ID** | **Order\_Date** | **Day** | **Month** | **Year** | **Weekday** |
| --- | --- | --- | --- | --- | --- |
| 101 | 2023-07-01 | 1 | 7 | 2023 | Saturday |
| 102 | 2023-07-05 | 5 | 7 | 2023 | Wednesday |
| 103 | 2023-07-10 | 10 | 7 | 2023 | Monday |

**3. pd.date\_range()**

**Purpose:**

Create a range of dates — useful in generating calendars, timesheets, etc.

**Real-Life Scenario:**

You are building a dashboard that shows sales every day for a month, even if there were no sales.

**Example:**

dates = pd.date\_range(start='2023-01-01', end='2023-01-10', freq='D')

df\_calendar = pd.DataFrame({'Date': dates})

print(df\_calendar.head())

**Output:**

Creates 10 rows from Jan 1 to Jan 10, 2023.

You can change frequency:

* 'D': daily
* 'W': weekly
* 'M': month end
* 'H': hourly
* 'B': business days only

**4. Filtering by Date/Time**

**Example:**

You want to get only the orders placed in July 2023.

# Assuming df['Order\_Date'] is already datetime

filtered\_df = df[(df['Order\_Date'] >= '2023-07-01') & (df['Order\_Date'] <= '2023-07-31')]

print(filtered\_df)

**Output:**

All orders made in July 2023.

**BONUS: Advanced Examples**

**Example 1: Monthly Sales Summary**

# Simulated sales data

df = pd.DataFrame({

'Order\_Date': pd.date\_range(start='2023-01-01', periods=100, freq='D'),

'Sales': range(100)

})

df['Month'] = df['Order\_Date'].dt.to\_period('M')

monthly\_sales = df.groupby('Month')['Sales'].sum()

print(monthly\_sales)

**----------------------------------------------------------------------**

**Example 2: Find Weekend Orders**

df['Weekday'] = df['Order\_Date'].dt.dayofweek # Monday = 0, Sunday = 6

weekend\_orders = df[df['Weekday'] >= 5]

print(weekend\_orders.head())

**Example 3: Create Timesheet for Attendance**

employee\_dates = pd.date\_range(start='2025-07-01', end='2025-07-28', freq='B') # Business days only

attendance = pd.DataFrame({

'Date': employee\_dates,

'Status': ['Present'] \* len(employee\_dates)

})

print(attendance.head())

**Summary**

|  |  |
| --- | --- |
| Task | Method |
| Convert to datetime | pd.to\_datetime() |
| Extract parts | .dt.day, .dt.month, .dt.year |
| Generate range of dates | pd.date\_range() |
| Filter by date | Logical filtering (>=, <=) |
| Group by time periods | .dt.to\_period() |
| Get weekdays | .dt.day\_name() or .dt.dayofweek |

Great! Here's the continuation with:

**10 Practice Questions – Dates & Times in Pandas**

Use this sample DataFrame for most questions:

import pandas as pd

data = {

'Order\_ID': [101, 102, 103, 104, 105, 106],

'Order\_Date': ['2023-07-01', '2023-07-05', '2023-07-10', '2023-08-01', '2023-08-15', '2023-09-01'],

'Sales': [100, 150, 120, 130, 160, 180]

}

df = pd.DataFrame(data)

df['Order\_Date'] = pd.to\_datetime(df['Order\_Date'])

-----------------------------------------------------------------------------------------------------

**Practice Questions**

**Q1. Convert the 'Order\_Date' column into datetime format and print data types.**

**Q2. Extract day, month, year from 'Order\_Date' into new columns: 'Day', 'Month', 'Year'.**

**Q3. Extract the weekday name (like 'Monday') for each order.**

**Q4. Filter the records for the month of August 2023.**

**Q5. Create a DataFrame containing dates from 2023-09-01 to 2023-09-10, daily frequency.**

**Q6. From Q5, add a column 'Is\_Weekend' where Saturday/Sunday = True.**

**Q7. Count how many orders were placed in July 2023.**

**Q8. Create a summary: total sales month-wise.**

**Q9. Add a new column ‘Quarter’ showing which quarter the order belongs to (Q1, Q2, etc.).**

**Q10. Create a calendar DataFrame for all business days in August 2023.**

-------------------------------------------------------------------------------------------------------------------------

**Project 1: Attendance Tracker (Employee Timesheet)**

**Objective:**

Generate a timesheet for an employee showing working days, weekends, and attendance status.

import pandas as pd

# Create working calendar

dates = pd.date\_range(start='2025-07-01', end='2025-07-31', freq='D')

df = pd.DataFrame({'Date': dates})

# Mark weekends

df['Day'] = df['Date'].dt.day\_name()

df['Is\_Weekend'] = df['Day'].isin(['Saturday', 'Sunday'])

# Mark attendance randomly (simulate)

import numpy as np

np.random.seed(42)

df['Status'] = np.where(df['Is\_Weekend'], 'Holiday', np.random.choice(['Present', 'Absent'], size=len(df)))

print(df)

**Project 2: Monthly Sales Dashboard**

**Objective:**

Summarize monthly total sales and show month name.

# Sample data

df = pd.DataFrame({

'Order\_Date': pd.date\_range(start='2023-01-01', periods=120, freq='D'),

'Sales': np.random.randint(100, 500, size=120)

})

# Add month name

df['Month'] = df['Order\_Date'].dt.strftime('%B')

# Group by month

monthly\_sales = df.groupby('Month')['Sales'].sum().reset\_index()

print(monthly\_sales)

**Project 3: Analyze Website Traffic**

**Objective:**

Detect website traffic spikes on weekends.

# Simulate traffic data

traffic = pd.DataFrame({

'Date': pd.date\_range(start='2025-07-01', end='2025-07-31'),

'Visitors': np.random.randint(200, 2000, size=31)

})

traffic['Weekday'] = traffic['Date'].dt.day\_name()

traffic['Is\_Weekend'] = traffic['Weekday'].isin(['Saturday', 'Sunday'])

# Find average weekend traffic

weekend\_avg = traffic[traffic['Is\_Weekend']]['Visitors'].mean()

weekday\_avg = traffic[~traffic['Is\_Weekend']]['Visitors'].mean()

print("Weekend Avg Visitors:", weekend\_avg)

print("Weekday Avg Visitors:", weekday\_avg)