Open in Colab

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta
from IPython.display import display
```

# PART 1: BASIC JOINS AND MERGES

These exercises cover fundamental join operations with clear edge cases and detailed explanations.

## Key Concepts

### 1. Merge — `pd.merge()`

Most versatile method for database-style joins

Syntax: `pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None)`

Best for:

- Combining DataFrames based on common columns or indices

Let's create a couple of dataframes to bring together in differnet ways

In [ ]:
```
# @title
# create the first DataFrame
df_mkt = pd.DataFrame({"fruit" : ["apple", "banana", "avocado", "grapes", "blueberr
                    "market_price" : [26.99, 29.99, 41.99, 23.99, 26.99] })
print("The first DataFrame")
display(df_mkt)

# create the second DataFrame
df_whl = pd.DataFrame({"fruit" : ["grapes", "blueberries", "banana", "apple", "avoc
                    "wholesaler_price" :  [19.99, 14.99, 15.99, 20.99, 30.99] })
print("\nThe second DataFrame")
display(df_whl)
```

```
The first DataFrame
```

|   | fruit | market_price |
|---|---|---|
| **0** | apple | 26.99 |
| **1** | banana | 29.99 |
| **2** | avocado | 41.99 |
| **3** | grapes | 23.99 |
| **4** | blueberries | 26.99 |

```
The second DataFrame
```

|   | fruit | wholesaler_price |
|---|---|---|
| **0** | grapes | 19.99 |
| **1** | blueberries | 14.99 |
| **2** | banana | 15.99 |
| **3** | apple | 20.99 |
| **4** | avocado | 30.99 |

```
In [ ]: # joining the DataFrames
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

## 2. Join — `DataFrame.join()`

Specialized method for index-based joining

Syntax: `df1.join(df2, how='left', lsuffix='', rsuffix='')`

Best for:

- Combining DataFrames using their indices

```
In [ ]:
```

```
In [ ]:
```

## 3. Concatenate — `pd.concat()`

Used for combining DataFrames along an axis

Syntax: `pd.concat([df1, df2], axis=0)`

Best for:

- Stacking DataFrames vertically (axis=0)
- Combining DataFrames side-by-side (axis=1)

In [ ]:

```
# Stacking DataFrames vertically (axis=0)
```

In [ ]:

```

```

In [ ]:

```
# Combining DataFrames side-by-side (axis=1)
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

## 4. Additional Methods for Combining Data

- `merge_asof()` for time-series-like joining
- `combine_first()` for updating missing values

In [ ]:

```
# @title
df_trades = pd.DataFrame({
    'time': pd.to_datetime(['20160525 13:30:00.023',
                            '20160525 13:30:00.038',
                            '20160525 13:30:00.048',
                            '20160525 13:30:00.048',
                            '20160525 13:30:00.048']),
    'ticker': ['MSFT', 'MSFT','GOOG', 'GOOG', 'AAPL'],
    'price': [51.95, 51.95,720.77, 720.92, 98.00],
    'quantity': [75, 155,100, 100, 100]},
    columns=['time', 'ticker', 'price', 'quantity'])

df_quotes = pd.DataFrame({
    'time': pd.to_datetime(['20160525 13:30:00.023',
                            '20160525 13:30:00.023',
                            '20160525 13:30:00.030',
                            '20160525 13:30:00.041',
                            '20160525 13:30:00.048',
                            '20160525 13:30:00.049',
                            '20160525 13:30:00.072',
                            '20160525 13:30:00.075']),
    'ticker': ['GOOG', 'MSFT', 'MSFT','MSFT', 'GOOG', 'AAPL', 'GOOG','MSFT'],
    'bid': [720.50, 51.95, 51.97, 51.99,720.50, 97.99, 720.50, 52.01],
    'ask': [720.93, 51.96, 51.98, 52.00,720.93, 98.01, 720.88, 52.03]},
    columns=['time', 'ticker', 'bid', 'ask'])
display(df_trades)
display(df_quotes)
```

In [ ]:

In [ ]:

In [ ]:

```python
# @title
# First DataFrame with some missing values
sales_data = pd.DataFrame({
    'Month': ['January', 'February', 'March', 'April'],
    'Sales': [200, None, 150, None]
})

# Second DataFrame with historical sales data
backup_data = pd.DataFrame({
      'Month': ['January', 'February', 'March', 'April'],
      'Sales': [180, 220, None, 170]
})

display(sales_data)
display(backup_data)
```

In [ ]:

```python
# Use combine_first to update missing values in sales_data with values from backup_

# Display the updated sales data
```

# Task: Basic Join Types

**Understanding Different Join Types** The hidden code below creates two DataFrames: 'df_students' and 'df_grades'

Key Learning Points:

- INNER join: Only keeps matches present in both DataFrames
- LEFT join: Keeps all records from left DataFrame, fills missing with NaN
- RIGHT join: Keeps all records from right DataFrame, fills missing with NaN
- OUTER join: Keeps all records from both DataFrames, fills missing with NaN

Edge Cases:

- Students with no grades
- Grades with no matching student
- Duplicate student entries

**Instructions** Create an inner, left, right and outer joins of df_students and df_grades on student_id.

In [ ]:

```python
# @title
# Sample data
df_students = pd.DataFrame({
    'student_id': [1, 2, 3, 4, 5, 5],  # Note: ID 5 is duplicated
    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Eve'],
    'major': ['CS', 'Physics', 'Math', 'CS', 'Physics', 'Physics']
})

df_grades = pd.DataFrame({
    'student_id': [1, 2, 2, 6, 7],  # IDs 6,7 don't exist in students
    'course': ['Math', 'Physics', 'CS', 'Math', 'Physics'],
    'grade': [85, 92, 88, 90, 87]
})
```

In [ ]:

```
```

## Create dataframes based on the different types of joins

Inner, left, right and outer

In [ ]:

```
# Solutions demonstrating different join types
```

In [ ]:

```
print("Inner Join")
display(inner_join)

print("Outer Join")
display(outer_join)

print("Left Join")
display(left_join)

print("Right Join")
display(right_join)
```

## Time-Based Joins with Window Functions

In [ ]:

```
# Time-Based Joins with Window Functions

# Sample data
stock_prices = pd.DataFrame({
    'timestamp': pd.date_range('2024-01-01', periods=5, freq='H'),
    'price': [100, 101, 99, 102, 103]
})

company_events = pd.DataFrame({
    'timestamp': pd.date_range('2024-01-01 01:30:00', periods=3, freq='2H'),
    'event_type': ['News', 'Earnings', 'Press Release']
})
```

# PART 2: FASTSHIP, INC CASE

**— FastShip Transactional Data —**

Shipments:
https://raw.githubusercontent.com/iamctodd/datasets/refs/heads/main/joins/shipments.csv

Orders:
https://raw.githubusercontent.com/iamctodd/datasets/refs/heads/main/joins/orders.csv

Feedback:
https://raw.githubusercontent.com/iamctodd/datasets/refs/heads/main/joins/cmr_feedback.csv

**— SpeedyDelivery Data —**

Speedy Orders:
https://raw.githubusercontent.com/iamctodd/datasets/refs/heads/main/joins/speedy_orders.csv

GPS Tracking:
https://raw.githubusercontent.com/iamctodd/datasets/refs/heads/main/joins/gps_tracking.csv

Warehouse:
https://raw.githubusercontent.com/iamctodd/datasets/refs/heads/main/joins/warehouse.csv

**— Customer Support —**

Segments:
https://raw.githubusercontent.com/iamctodd/datasets/refs/heads/main/joins/segments.csv

Interactions:
https://raw.githubusercontent.com/iamctodd/datasets/refs/heads/main/joins/interactions.csv

Outcomes:
https://raw.githubusercontent.com/iamctodd/datasets/refs/heads/main/joins/outcomes.csv

**— Product Analytics —**

Inventory:
https://raw.githubusercontent.com/iamctodd/datasets/refs/heads/main/joins/inventory.csv

Sales: https://raw.githubusercontent.com/iamctodd/datasets/refs/heads/main/joins/sales.csv

Products:
https://raw.githubusercontent.com/iamctodd/datasets/refs/heads/main/joins/products.csv

## Part 1: Combining the data

**Create a summary report showing:**

- Total orders per region
- Average delivery time
- Average customer rating

- Percentage of orders with feedback

In [ ]:

```
# First import all the data!
df_orders = pd.read_csv('https://raw.githubusercontent.com/iamctodd/datasets/refs/h
df_ship = pd.read_csv('https://raw.githubusercontent.com/iamctodd/datasets/refs/hea
df_fbk = pd.read_csv('https://raw.githubusercontent.com/iamctodd/datasets/refs/head
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
# Create a summary report
```

In [ ]:

```
```

In [ ]:

```
# Percentage of orders with feedback
```

In [ ]:

```
# Total orders per region
""" YOUR CODE HERE """
```

In [ ]:

```
# Average delivery times
""" YOUR CODE HERE """
```

In [ ]:

```
# Average Customer Rating
""" YOUR CODE HERE """
```

In [ ]:

```
# Percent of orders with feedback
""" YOUR CODE HERE """
```

**What are the key strategic challenges facing FastShip Inc.?**

- ADD YOUR ANSWER HERE

**How might an integrated data analytics help address these challenges?**

- ADD YOUR ANSWER HERE

**Evaluate the timing of the SpeedyDelivery acquisition. What additional complexities does this add to Pablo's analytics integration project**

- This is my reply to this quesiont

**Write the Python code needed to:**

- Calculate delivery times
- Create a unified customer view across both companies

- Calculate standardized delivery times
- Integrate GPS tracking data with delivery performance metrics

In [ ]:

```
# Calculate delivery times
""" YOUR CODE HERE """
```

In [ ]:

```
# Create a unified customer view across both companies
""" YOUR CODE HERE """
```

In [ ]:

```
# Calculate standardized delivery times
""" YOUR CODE HERE """
```

In [ ]:

```
# Integrate GPS tracking with delivery performance
""" YOUR CODE HERE """
```

In [ ]:

```

```

**Evaluate the timing of the SpeedyDelivery acquisition. **

- Your Answer Here

**What additional complexities does this add to Pablo’s analytics integration project?**

- Your Answer Here

- **Create a customer service report by**
- Combine all three datasets to analyze resolution times by segment
- Identify customers with interactions but missing outcomes
- Calculate the average resolution time by channel and segment

In [ ]:

```
""" YOUR CODE HERE """
```

- Create a complete sales report with ** product details and inventory levels ** Analyze sales patterns by category ** Calculate revenue by category and date

In [ ]:

```

```

- **Address the following:** ** Different column names and metrics across both companeis ** Inconsistent business rules (e.g., delivery time calculations) ** Real-time GPS data integration ** Historical data compatibility