Data Manipulation Challenge

A Mental Model for Method Chaining in Pandas

🔗 Data Manipulation Challenge - A Mental Model for Method Chaining in Pandas

📊 **Challenge Requirements In Section** Student Analysis Section

- Complete discussion questions for at least 5 of the 7 mental models (for higher grades, complete all 7 plus additional requirements — see Grading Rubric for details)

Getting Started: Repository Setup 🚀

📁 **Getting Started**

Step 1: Fork the challenge repository to your own GitHub account

- Go to https://github.com/flyaflya/dataManipulationChallenge
- Click the **"Fork"** button (top-right corner) to create your own copy of the repository under your GitHub account
- **Important:** You must fork first so that you have your own repository to push changes to and deploy via GitHub Pages

Step 2: Clone **your fork** (not the original) to your local machine

- Navigate to **your forked repository** on GitHub (it will be at `https://github.com/YOUR-USERNAME/dataManipulationChallenge`)
- Click the green **"<> Code"** button on **your fork's** GitHub page
- Copy the URL (HTTPS is recommended)
- Open Cursor, open a terminal, and run: `git clone <paste-your-URL-here>`
- Open the cloned folder in Cursor

Step 3: Set up your Python environment (see Python Setup section below)

Step 4: You're ready to start! The data loading code is already provided in this file.

Note: This challenge uses the same `index.qmd` file you're reading right now — you'll edit it to complete your analysis.

💾 **Important: Save Your Work Frequently!**

Before you start: Make sure to commit your work often using the Source Control panel in Cursor (Ctrl+Shift+G or Cmd+Shift+G). This prevents the AI from overwriting your progress and ensures you don't lose your work.

Commit after each major step:

- After completing each mental model section
- After adding your visualizations
- After completing your advanced method chain

- Before asking the AI for help with new code

How to commit:

1. Open Source Control panel (Ctrl+Shift+G)
2. Stage your changes (+ button)
3. Write a descriptive commit message
4. Click the checkmark to commit

Remember: Frequent commits are your safety net!

🎯 Note on Python Usage

Recommended Workflow: Use Your Existing Virtual Environment If you completed the Tech Setup Challenge Part 2, you already have a virtual environment set up! Here's how to use it for this new challenge:

1. **Fork and clone this challenge repository** (see [Getting Started] section above for the link and detailed instructions)
2. **Open the cloned repository in Cursor**
3. **Set this project to use your existing Python interpreter:**
   - Press `Ctrl+Shift+P` → "Python: Select Interpreter"
   - Navigate to and choose the interpreter from your existing virtual environment (e.g., `your-previous-project/venv/Scripts/python.exe`)
4. **Activate the environment in your terminal:**
   - Open terminal in Cursor (`Ctrl + ``)
   - Navigate to your previous project folder where you have the `venv` folder
   - 💡 **Pro tip:** You can quickly navigate by typing `cd` followed by dragging the folder from your file explorer into the terminal
   - Activate using the appropriate command for your system:
     - **Windows Command Prompt:** `venv\Scripts\activate`
     - **Windows PowerShell:** `.\venv\Scripts\Activate.ps1`
     - **Mac/Linux:** `source venv/bin/activate`
   - You should see `(venv)` at the beginning of your terminal prompt
5. **Install additional packages if needed:** `pip install pandas numpy matplotlib seaborn`

⚠️ Cloud Storage Warning

Avoid using Google Drive, OneDrive, or other cloud storage for Python projects! These services can cause issues with: - Package installations failing due to file locking - Virtual environment corruption - Slow performance during pip operations

Best practice: Keep your Python projects in a local folder like `C:\Users\YourName\Documents\` or `~/Documents/` instead of cloud-synced folders.

Alternative: Create a New Virtual Environment If you prefer a fresh environment, follow the Quarto documentation: https://quarto.org/docs/projects/virtual-environments.html. Be sure to follow the instructions to activate the environment, set it up as your default Python interpreter for the project, and install the necessary packages (e.g. pandas) for this challenge. For installing the packages, you can use the `pip install -r requirements.txt` command since you already have the requirements.txt file in your project. Some steps do take a bit of time, so be patient.

Why This Works: Virtual environments are portable - you can use the same environment across multiple projects, and Cursor automatically activates it when you select the interpreter!

The Problem: Mastering Data Manipulation Through Method Chaining

Core Question: How can we efficiently manipulate datasets using `pandas` method chaining to answer complex business questions?

The Challenge: Real-world data analysis requires combining multiple data manipulation techniques in sequence. Rather than creating intermediate variables at each step, method chaining allows us to write clean, readable code that flows logically from one operation to the next.

Our Approach: We'll work with ZappTech's shipment data to answer critical business questions about service levels and cross-category orders, using the seven mental models of data manipulation through pandas method chaining.

> ⚠️ AI Partnership Required
>
> This challenge pushes boundaries intentionally. You'll tackle problems that normally require weeks of study, but with Cursor AI as your partner (and your brain keeping it honest), you can accomplish more than you thought possible.
>
> **The new reality:** The four stages of competence are Ignorance → Awareness → Learning → Mastery. AI lets us produce Mastery-level work while operating primarily in the Awareness stage. I focus on awareness training, you leverage AI for execution, and together we create outputs that used to require years of dedicated study.

The Seven Mental Models of Data Manipulation

The seven most important ways we manipulate datasets are:

1. **Assign:** Add new variables with calculations and transformations
2. **Subset:** Filter data based on conditions or select specific columns
3. **Drop:** Remove unwanted variables or observations
4. **Sort:** Arrange data by values or indices
5. **Aggregate:** Summarize data using functions like mean, sum, count
6. **Merge:** Combine information from multiple datasets
7. **Split-Apply-Combine:** Group data and apply functions within groups

Data and Business Context

We analyze ZappTech's shipment data, which contains information about product deliveries across multiple categories. This dataset is ideal for our analysis because:

- **Real Business Questions:** CEO wants to understand service levels and cross-category shopping patterns
- **Multiple Data Sources:** Requires merging shipment data with product category information
- **Complex Relationships:** Service levels may vary by product category, and customers may order across categories
- **Method Chaining Practice:** Perfect for demonstrating all seven mental models in sequence

Data Loading and Initial Exploration

Let's start by loading the ZappTech shipment data and understanding what we're working with.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta

# Load the shipment data
shipments_df = pd.read_csv(
    "https://raw.githubusercontent.com/flyaflya/persuasive/main/shipments.csv",
    parse_dates=['plannedShipDate', 'actualShipDate']
)

# Load product line data
product_line_df = pd.read_csv(
    "https://raw.githubusercontent.com/flyaflya/persuasive/main/productLine.csv"
)

# Reduce dataset size for faster processing (4,000 rows instead of 96,805 rows)
shipments_df = shipments_df.head(4000)

print("Shipments data shape:", shipments_df.shape)
print("\nShipments data columns:", shipments_df.columns.tolist())
print("\nFirst few rows of shipments data:")
print(shipments_df.head(10))

print("\n" + "="*50)
print("Product line data shape:", product_line_df.shape)
print("\nProduct line data columns:", product_line_df.columns.tolist())
print("\nFirst few rows of product line data:")
print(product_line_df.head(10))
```

```
Shipments data shape: (4000, 5)

Shipments data columns: ['shipID', 'plannedShipDate', 'actualShipDate', 'partID', 'quantity']

First few rows of shipments data:
   shipID plannedShipDate actualShipDate       partID  quantity
0   10001      2013-11-06     2013-10-04  part92b16c5         6
1   10002      2013-10-15     2013-10-04   part66983b         2
2   10003      2013-10-25     2013-10-07  part8e36f25         1
3   10004      2013-10-14     2013-10-08  part30f5de0         1
4   10005      2013-10-14     2013-10-08  part9d64d35         6
5   10006      2013-10-14     2013-10-08  part6cd6167        15
6   10007      2013-10-14     2013-10-08  parta4d5fd1         2
7   10008      2013-10-14     2013-10-08  part08cadf5         1
8   10009      2013-10-14     2013-10-08  part5cc4989        10
9   10010      2013-10-14     2013-10-08  part912ae4c         1
```

```
==================================================
Product line data shape: (11997, 3)

Product line data columns: ['partID', 'productLine', 'prodCategory']

First few rows of product line data:
        partID productLine prodCategory
0  part00005ba      line4c      Liquids
1  part000b57d      line61     Machines
2  part00123bf      linec1  Marketables
3  part0021fc9      line61     Machines
4  part0027e86      line2f     Machines
5  part002ed95      line4c      Liquids
6  part0030856      lineb8     Machines
7  part0033dfd      line49      Liquids
8  part0037a2a      linea3  Marketables
9  part003caee      linea3  Marketables
```

> 💡 Understanding the Data
>
> **Shipments Data:** Contains individual line items for each shipment, including: - `shipID`: Unique identifier for each shipment - `partID`: Product identifier - `plannedShipDate`: When the shipment was supposed to go out - `actualShipDate`: When it actually shipped - `quantity`: How many units were shipped
>
> **Product Category and Line Data:** Contains product category information: - `partID`: Links to shipments data - `productLine`: The specific product line within a category - `prodCategory`: The broader category each product belongs to
>
> **Business Questions We'll Answer:** 1. Does service level (on-time shipments) vary across product categories? 2. How often do orders include products from more than one category?

The Seven Mental Models: A Progressive Learning Journey

> 🎯 Method Chaining Philosophy
>
> > "Each operation should build naturally on the previous one"
>
> *Think of method chaining like building with LEGO blocks - each piece connects to the next, creating something more complex and useful than the individual pieces.*

Now we'll work through each of the seven mental models using method chaining, starting simple and building complexity.

1. Assign: Adding New Variables

Mental Model: Create new columns with calculations and transformations.

Let's start by calculating whether each shipment was late:

```
# Simple assignment - calculate if shipment was late
shipments_with_lateness = (
```

```
    shipments_df
    .assign(
        is_late=lambda df: df['actualShipDate'] > df['plannedShipDate'],
        days_late=lambda df: (df['actualShipDate'] - df['plannedShipDate']).dt.days
    )
)

print("Added lateness calculations:")
print(shipments_with_lateness[['shipID', 'plannedShipDate', 'actualShipDate', 'is_late', 'days_la
```

```
Added lateness calculations:
   shipID plannedShipDate actualShipDate  is_late  days_late
0   10001      2013-11-06     2013-10-04    False        -33
1   10002      2013-10-15     2013-10-04    False        -11
2   10003      2013-10-25     2013-10-07    False        -18
3   10004      2013-10-14     2013-10-08    False         -6
4   10005      2013-10-14     2013-10-08    False         -6
```

💡 Method Chaining Tip for New Python Users

Why use `lambda df:`? When chaining methods, we need to reference the current state of the dataframe. The `lambda df:` tells pandas "use the current dataframe in this calculation." Without it, pandas would look for a variable called `df` that doesn't exist.

Alternative approach: You could also write this as separate steps, but method chaining keeps related operations together and makes the code more readable.

🤔 Discussion Questions: Assign Mental Model

Question 1: Data Types and Date Handling - What is the `dtype` of the `actualShipDate` series? How can you find out using code? - Why is it important that both `actualShipDate` and `plannedShipDate` have the same data type for comparison?

Question 2: String vs Date Comparison - Can you give an example where comparing two dates as strings would yield unintuitive results, e.g. what happens if you try to compare "04-11-2025" and "05-20-2024" as strings vs as dates?

Question 3: Debug This Code

```
# This code has an error - can you spot it?
shipments_with_lateness = (
    shipments_df
    .assign(
        is_late=lambda df: df['actualShipDate'] > df['plannedShipDate'],
        days_late=lambda df: (df['actualShipDate'] - df['plannedShipDate']).dt.days,
        lateStatement="Darn Shipment is Late" if shipments_df['is_late'] else "Shipment is on Time"
    )
)
```

What's wrong with the `lateStatement` assignment and how would you fix it?

Your Answer: Data Types and Date Handling

Replace this with your answer to Question 1.

Your Answer: String vs Date Comparison

Replace this with your answer to Question 2.

Your Answer: Debug This Code

Replace this with your answer to Question 3.

2. Subset: Querying Rows and Filtering Columns

Mental Model: Query rows based on conditions and filter to keep specific columns.

Let's query for only late shipments and filter to keep the columns we need:

```
# Query rows for late shipments and filter to keep specific columns
late_shipments = (
    shipments_with_lateness
    .query('is_late == True')  # Query rows where is_late is True
    .filter(['shipID', 'partID', 'plannedShipDate', 'actualShipDate', 'days_late'])  # Filter to
)

print(f"Found {len(late_shipments)} late shipments out of {len(shipments_with_lateness)} total")
print("\nLate shipments sample:")
print(late_shipments.head())
```

```
Found 456 late shipments out of 4000 total

Late shipments sample:
     shipID       partID plannedShipDate actualShipDate  days_late
776   10192  part0164a70      2013-10-09     2013-10-14          5
777   10192  part9259836      2013-10-09     2013-10-14          5
778   10192  part4526c73      2013-10-09     2013-10-14          5
779   10192  partbb47e81      2013-10-09     2013-10-14          5
780   10192  part008482f      2013-10-09     2013-10-14          5
```

🔍 Understanding the Methods

- `.query()`: Query rows based on conditions (like SQL WHERE clause)
- `.filter()`: Filter to keep specific columns by name
- **Alternative**: You could use `.loc[]` for more complex row querying, but `.query()` is often more readable

🤔 Discussion Questions: Subset Mental Model

Question 1: Query vs Boolean Indexing - What's the difference between using `.query('is_late == True')` and `[df['is_late'] == True]`? - Which approach is more readable and why?

Question 2: Additional Row Querying - Can you show an example of using a variable like `late_threshold` to query rows for shipments that are at least `late_threshold` days late, e.g. what if you wanted to query rows for shipments that are at least 5 days late?

Your Answer: Query vs Boolean Indexing

Replace this with your answer to Question 1.

Your Answer: Additional Row Querying

Replace this with your answer to Question 2.

3. Drop: Removing Unwanted Data

Mental Model: Remove columns or rows you don't need.

Let's clean up our data by removing unnecessary columns:

```
# Create a cleaner dataset by dropping unnecessary columns
clean_shipments = (
    shipments_with_lateness
    .drop(columns=['quantity'])  # Drop quantity column (not needed for our analysis)
    .dropna(subset=['plannedShipDate', 'actualShipDate'])  # Remove rows with missing dates
)

print(f"Cleaned dataset: {len(clean_shipments)} rows, {len(clean_shipments.columns)} columns")
print("Remaining columns:", clean_shipments.columns.tolist())
```

```
Cleaned dataset: 4000 rows, 6 columns
Remaining columns: ['shipID', 'plannedShipDate', 'actualShipDate', 'partID', 'is_late',
'days_late']
```

🤔 Discussion Questions: Drop Mental Model

Question 1: Drop vs Filter Strategies - What's the difference between `.drop(columns=['quantity'])` and `.filter()` with a list of columns you want to keep? - When would you choose to drop columns vs filter to keep specific columns?

Question 2: Handling Missing Data - What happens if you use `.dropna()` without specifying `subset`? How is this different from `.dropna(subset=['plannedShipDate', 'actualShipDate'])`? - Why might you want to be selective about which columns to check for missing values?

Your Answer: Drop vs Filter Strategies

Replace this with your answer to Question 1.

Your Answer: Handling Missing Data

Replace this with your answer to Question 2.

4. Sort: Arranging Data

Mental Model: Order data by values or indices.

Let's sort by lateness to see the worst offenders:

```
# Sort by days late (worst first)
sorted_by_lateness = (
    clean_shipments
    .sort_values('days_late', ascending=False)  # Sort by days_late, highest first
    .reset_index(drop=True)  # Reset index to be sequential
)

print("Shipments sorted by lateness (worst first):")
print(sorted_by_lateness[['shipID', 'partID', 'days_late', 'is_late']].head(10))
```

```
Shipments sorted by lateness (worst first):
   shipID        partID  days_late  is_late
0   10956  partb6208b5         21     True
1   10956  part04ef2f7         21     True
2   10956  part4875f85         21     True
3   10956  partb722d53         21     True
4   10956  partc979912         21     True
5   10956  parta27d449         21     True
6   10956  partc653823         21     True
7   10956  part82e69e9         21     True
8   10956  partf23fd1e         21     True
9   10956  part825873c         21     True
```

🤔 Discussion Questions: Sort Mental Model

Question 1: Sorting Strategies - What's the difference between `ascending=False` and `ascending=True` in sorting? - How would you sort by multiple columns (e.g., first by `is_late`, then by `days_late`)?

Question 2: Index Management - Why do we use `.reset_index(drop=True)` after sorting? - What happens to the original index when you sort? Why might this be problematic?

Your Answer: Sorting Strategies

Replace this with your answer to Question 1.

Your Answer: Index Management

Replace this with your answer to Question 2.

5. Aggregate: Summarizing Data

Mental Model: Calculate summary statistics across groups or the entire dataset.

Let's calculate overall service level metrics:

```
# Calculate overall service level metrics
service_metrics = (
    clean_shipments
    .agg({
        'is_late': ['count', 'sum', 'mean'],  # Count total, count late, calculate percentage
        'days_late': ['mean', 'max']  # Average and maximum days late
    })
    .round(3)
```

```
)

print("Overall Service Level Metrics:")
print(service_metrics)

# Calculate percentage on-time directly from the data
on_time_rate = (1 - clean_shipments['is_late'].mean()) * 100
print(f"\nOn-time delivery rate: {on_time_rate:.1f}%")
```

```
Overall Service Level Metrics:
        is_late  days_late
count  4000.000        NaN
sum     456.000        NaN
mean      0.114     -0.974
max         NaN     21.000

On-time delivery rate: 88.6%
```

🤔 Discussion Questions: Aggregate Mental Model

Question 1: Boolean Aggregation - Why does `sum()` work on boolean values? What does it count?

Your Answer: Boolean Aggregation

Replace this with your answer to Question 1.

6. Merge: Combining Information

Mental Model: Join data from multiple sources to create richer datasets.

Now let's analyze service levels by product category. First, we need to merge our data:

```
# Merge shipment data with product line data
shipments_with_category = (
    clean_shipments
    .merge(product_line_df, on='partID', how='left')  # Left join to keep all shipments
    .assign(
        category_late=lambda df: df['is_late'] & df['prodCategory'].notna()  # Only count as late
    )
)

print("\nProduct categories available:")
print(shipments_with_category['prodCategory'].value_counts())
```

```
Product categories available:
prodCategory
Marketables    1850
Machines        846
SpareParts      767
```

```
Liquids        537
Name: count, dtype: int64
```

🤔 Discussion Questions: Merge Mental Model

Question 1: Join Types and Data Loss - Why does your professor think we should use `how='left'` in most cases? - How can you check if any shipments were lost during the merge?

Question 2: Key Column Matching - What happens if there are duplicate `partID` values in the `product_line_df`?

Your Answer: Join Types and Data Loss

Replace this with your answer to Question 1.

Your Answer: Key Column Matching

Replace this with your answer to Question 2.

7. Split-Apply-Combine: Group Analysis

Mental Model: Group data and apply functions within each group.

Now let's analyze service levels by category:

```
# Analyze service levels by product category
service_by_category = (
    shipments_with_category
    .groupby('prodCategory')  # Split by product category
    .agg({
        'is_late': ['any', 'count', 'sum', 'mean'],  # Count, late count, percentage late
        'days_late': ['mean', 'max']  # Average and max days late
    })
    .round(3)
)

print("Service Level by Product Category:")
print(service_by_category)
```

```
Service Level by Product Category:
             is_late                    days_late    
                 any count  sum   mean      mean max
prodCategory                                        
Liquids         True   537   22  0.041    -0.950  19
Machines        True   846  152  0.180    -1.336  21
Marketables     True  1850  145  0.078    -0.804  21
SpareParts      True   767  137  0.179    -1.003  21
```

🤔 Discussion Questions: Split-Apply-Combine Mental Model

Question 1: GroupBy Mechanics - What does `.groupby('prodCategory')` actually do? How does it "split" the data? - Why do we need to use `.agg()` after grouping? What happens if you don't?

Question 2: Multi-Level Grouping - Explore grouping by `['shipID', 'prodCategory']`? What question does this answer versus grouping by `'prodCategory'` alone? (HINT: There may be many rows with identical shipID's due to a particular order having multiple partID's.)

Your Answer: GroupBy Mechanics

Replace this with your answer to Question 1.

Your Answer: Multi-Level Grouping

Replace this with your answer to Question 2.

Answering A Business Question

Putting It All Together: Combine multiple data manipulation techniques to answer complex business questions.

Let's create a comprehensive analysis by combining shipment-level data with category information:

```
# Create a comprehensive analysis dataset
comprehensive_analysis = (
    shipments_with_category
    .groupby(['shipID', 'prodCategory'])  # Group by shipment and category
    .agg({
        'is_late': 'any',  # True if any item in this shipment/category is late
        'days_late': 'max'  # Maximum days late for this shipment/category
    })
    .reset_index()
    .assign(
        has_multiple_categories=lambda df: df.groupby('shipID')['prodCategory'].transform('nuniqu
    )
)

print("Comprehensive analysis - shipments with multiple categories:")
multi_category_shipments = comprehensive_analysis[comprehensive_analysis['has_multiple_categories
print(f"Shipments with multiple categories: {multi_category_shipments['shipID'].nunique()}")
print(f"Total unique shipments: {comprehensive_analysis['shipID'].nunique()}")
print(f"Percentage with multiple categories: {multi_category_shipments['shipID'].nunique() / comp
```

```
Comprehensive analysis - shipments with multiple categories:
Shipments with multiple categories: 232
Total unique shipments: 997
Percentage with multiple categories: 23.3%
```

🤔 Discussion Questions: Answering A Business Question

Question 1: Business Question Analysis - What business question does this comprehensive analysis answer? - How does grouping by `['shipID', 'prodCategory']` differ from grouping by just `'prodCategory'` ? - What insights can ZappTech's management gain from knowing the percentage of multi-category shipments?

Your Answer: Business Question Analysis

> *Replace this with your answer to Question 1.*

Student Analysis Section: Mastering Data Manipulation

Your Task: Demonstrate your mastery of the seven mental models through comprehensive discussion and analysis. The bulk of your grade comes from thoughtfully answering the discussion questions for each mental model. See below for more details.

Core Challenge: Discussion Questions Analysis

For each mental model, provide: - Clear, concise answers to all discussion questions - Code examples where appropriate to support your explanations

> 📊 Discussion Questions Requirements
>
> **Complete all discussion question sections:** 1. **Assign Mental Model:** Data types, date handling, and debugging 2. **Subset Mental Model:** Filtering strategies and complex queries 3. **Drop Mental Model:** Data cleaning and quality management 4. **Sort Mental Model:** Data organization and business logic 5. **Aggregate Mental Model:** Summary statistics and business metrics 6. **Merge Mental Model:** Data integration and quality control 7. **Split-Apply-Combine Mental Model:** Group analysis and advanced operations 8. **Answering A Business Question:** Combining multiple data manipulation techniques to answer a business question

Professional Visualizations (For 100% Grade)

Your Task: Create a professional visualization that supports your analysis and demonstrates your understanding of the data.

Create visualizations showing: - Service level (on-time percentage) by product category

Your visualizations should: - Use clear labels and professional formatting - Support the insights from your discussion questions - Be appropriate for a business audience - Do not `echo` the code that creates the visualizations

Grading Rubric 🎓

75% Grade: Complete discussion questions for at least 5 of the 7 mental models with clear, thoughtful responses.

85% Grade: Complete discussion questions for all 7 mental models with comprehensive, well-reasoned responses.

95% Grade: Complete all discussion questions plus the "Answering A Business Question" section.

100% Grade: Complete all discussion questions plus create a professional visualization showing service level by product category.

Submission Checklist ✅

Setup & Deployment (Required for Any Points):

- [ ] Forked repository from [flyaflya/dataManipulationChallenge]() to your GitHub account
- [ ] Cloned **your fork** locally using Cursor (or VS Code)
- [ ] Document rendered to HTML successfully
- [ ] HTML files pushed to your repository
- [ ] GitHub Pages enabled and working
- [ ] Site accessible at `https://[your-username].github.io/dataManipulationChallenge/`

Content (See Grading Rubric for Grade Tiers):

- [ ] Discussion questions completed for at least 5 of 7 mental models (75%)
- [ ] Discussion questions completed for all 7 mental models (85%)
- [ ] "Answering A Business Question" discussion questions completed (95%)
- [ ] Professional visualization showing service level by product category (100%)

Report Quality (Critical for Higher Grades):

- [ ] Professional writing style (no AI-generated fluff)
- [ ] Concise analysis that gets to the point
- [ ] Code examples where appropriate to support explanations