

# 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 all seven mental models of data manipulation using method chaining
- Create visualizations that demonstrate your understanding of the data
- Do not echo the code that creates the visualizations

### 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 → Learn-

ing → 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 (as in original notebook)
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())

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())

```

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

=====

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

```

### i 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 Line Data:** Contains product category information:  
 - **partID:** Links to shipments data  
 - **productLine:** The 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

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_late']])
```

Added lateness calculations:

	shipID	plannedShipDate	actualShipDate	is_late	days_late
0	10001	2013-10-06	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.

## 2. Subset: Filtering Data

**Mental Model:** Select specific rows or columns based on conditions.

Let’s filter for only late shipments and see what we find:

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

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

## 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']

## 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

```

## 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%

## 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['productLine'].notna() # Only count as late if both are true
    )
)

print(f"After merging: {len(shipments_with_category)} rows")
print(f"Shipments with category info: {shipments_with_category['productLine'].notna().sum()}")
print("\nProduct categories available:")
print(shipments_with_category['productLine'].value_counts())

```

After merging: 4000 rows

Shipments with category info: 4000

Product categories available:

productLine	value
linea3	1067
linec1	783

```

line55      767
line49      269
line9b      254
lined6      195
line4b      142
line7a      76
line61      76
line4c      73
line6d      61
line71      58
line60      46
line56      42
line94      27
linef3      19
lineb8      11
linee2      9
line90      8
line39      7
line2f      6
line5f      2
linec8      2
Name: count, dtype: int64

```

## 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('productLine') # Split by product category
    .agg({
        'is_late': ['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
      count     sum    mean      mean max
productLine
line2f        6      2  0.333   3.833  19
line39        7      1  0.143  -3.857   3
line49       269     10  0.037  -1.022   6
line4b       142     18  0.127  -1.599  19
line4c        73      2  0.027  -0.932   3
line55       767    137  0.179  -1.003  21
line56        42      6  0.143  -2.214   4
line5f         2      0  0.000  -1.000   0
line60        46     22  0.478   2.435  19
line61        76     25  0.329  -1.211  19
line6d        61     10  0.164  -2.016   3
line71        58      1  0.017  -1.793   1
line7a       76     21  0.276  -0.303  21
line90        8      5  0.625  -0.875   3
line94       27      0  0.000  -2.778   0
line9b       254    39  0.154  -1.331  21
linea3      1067    69  0.065  -0.688  21
lineb8        11      0  0.000  -6.636  -4
linec1       783    76  0.097  -0.962  21
linec8         2      1  0.500  -0.500   5
lined6       195    10  0.051  -0.856  19
linee2        9      0  0.000  -3.889   0
linef3       19      1  0.053  -2.368   1

```

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', 'productLine']) # 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')['productLine'].transform('nun

```

```

        )
    )

print("Comprehensive analysis - shipments with multiple categories:")
multi_category_shipments = comprehensive_analysis[comprehensive_analysis['has_multiple_categ
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() /
```

Comprehensive analysis - shipments with multiple categories:  
 Shipments with multiple categories: 286  
 Total unique shipments: 997  
 Percentage with multiple categories: 28.7%

## Student Analysis Section: Mastering Data Manipulation

**Your Task:** Demonstrate your mastery of the seven mental models by creating comprehensive analysis and visualizations. You'll need to complete three key components:

### 1. Service Level Analysis by Product Category

**Create a visualization showing:** - Service level (on-time percentage) by product category - Average days late by product category - Do not echo the code that creates the visualization

**Add Brief Discussion of the Visualization** - Which product categories have the best/worst service levels? - Are there significant differences between categories? - What business implications does this have for ZappTech?

#### ! Visualization Requirements

Create two plots: 1. **Service Level Bar Chart:** Show on-time percentage by product category 2. **Days Late Box Plot:** Show distribution of days late by product category Use clear labels and consider using color coding to highlight differences.

### 2. Cross-Category Order Analysis

**Your Task:** Analyze how often customers order products from multiple categories.

**Create one visualization with two side-by-side plots showing:** - Distribution of categories per shipment - Percentage of shipments with multiple categories over time

**Your analysis should explain:** - What percentage of shipments include multiple product categories? - Are there trends over time in cross-category ordering? - What does this tell us about customer behavior?

#### ! Cross-Category Analysis Requirements

Create a comprehensive analysis showing: 1. **Category Distribution:** Histogram of number of categories per shipment 2. **Time Series:** Percentage of multi-category shipments over time

- Use the same y-axis scale for comparison if creating side-by-side plots
- Do not echo the code that creates the visualization

### 3. Advanced Method Chaining Challenge

**Your Task:** Create a single, complex method chain that demonstrates all seven mental models.

**Create a comprehensive analysis that:** - Uses all seven mental models in one method chain - Answers a specific business question - Shows advanced pandas techniques

**Your analysis should demonstrate:** - **Assign:** Create new calculated fields - **Subset:** Filter data based on business logic - **Drop:** Remove unnecessary columns/rows - **Sort:** Order results meaningfully - **Aggregate:** Calculate summary statistics - **Split-Apply-Combine:** Group analysis - **Merge:** Combine multiple data sources

#### ! Advanced Method Chaining Requirements

Create a single method chain that: - Uses all seven mental models - Answers: “What are the service level trends for each product category over time?” - Includes at least 8 method calls in the chain - Produces a meaningful business insight  
Show the complete method chain and explain each step.

## Challenge Requirements

### Minimum Requirements for Any Points on Challenge

1. **Complete All Seven Mental Models:** Demonstrate each of the seven mental models using method chaining in your analysis.
2. **Create Visualizations:** Complete the three visualization requirements in the Student Analysis Section.

3. **Advanced Method Chain:** Create a single, complex method chain that uses all seven mental models to answer a business question.
4. **Professional Analysis:** Write clear, business-focused analysis that explains your findings and their implications.

## Getting Started: Repository Setup

### ! Getting Started

- Step 1:** Create a new repository in your GitHub account named “dataManipulation-Challenge”  
**Step 2:** Clone your repository locally using Cursor (or VS Code)  
**Step 3:** Copy this `index.qmd` file to your repository  
**Step 4:** You’re ready to start! The data loading code is already provided.

### 💡 Why This Challenge Matters

#### Real-world Skills:

- **Method chaining:** The foundation of modern data analysis
- **Business analysis:** Answering real questions with data
- **Code organization:** Writing clean, readable analysis code
- **Problem solving:** Breaking complex questions into manageable steps

**What you’ll learn:** - How to think about data manipulation systematically - When to use each mental model - How to combine multiple operations efficiently - How to write professional data analysis reports

## Getting Started Tips

### ℹ 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.*

### 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!*

## **Grading Rubric**

### What You're Really Being Graded On

**This is a data analysis report, not just a coding exercise.** You're analyzing ZappTech's shipment data and reporting your findings like a professional analyst would. Think of this as a brief you'd write for the CEO about service levels and customer behavior patterns.

#### **What makes a great report:**

- **Clear narrative:** Tell the story of what you discovered about ZappTech's operations
- **Insightful analysis:** Focus on the most interesting findings about service levels and customer behavior
- **Professional presentation:** Clean, readable, and engaging
- **Concise conclusions:** No AI babble or unnecessary technical jargon
- **Human insights:** Your interpretation of what the data actually means for the business
- **Practical implications:** What ZappTech should do based on your findings

**What we're looking for:** A compelling 2-3 minute read that demonstrates both your

data manipulation skills and your ability to extract business insights from data.

### **Questions to Answer for 75% Grade on Challenge**

- 1. Service Level Analysis:** Provide a clear, well-reasoned analysis of service levels by product category. Your analysis should demonstrate understanding of the data and identify meaningful differences between categories.

### **Questions to Answer for 85% Grade on Challenge**

- 2. Cross-Category Analysis:** Provide a thorough analysis of how often customers order from multiple categories. Your analysis should explain what this tells us about customer behavior and business opportunities.

### **Questions to Answer for 95% Grade on Challenge**

- 3. Advanced Method Chaining:** Your analysis should include a sophisticated method chain that demonstrates mastery of all seven mental models. Focus on the technical execution and business value of your analysis.

### **Questions to Answer for 100% Grade on Challenge**

- 4. Professional Presentation:** Your analysis should be written in a professional, engaging style that would be appropriate for a business audience. Use clear visualizations and focus on practical insights rather than technical jargon.

### **Submission Checklist**

#### **Minimum Requirements (Required for Any Points):**

- Created repository named “dataManipulationChallenge” in your GitHub account
- Cloned repository locally using Cursor (or VS Code)
- Completed all three analysis sections with visualizations
- Document rendered to HTML successfully
- HTML files uploaded to your repository
- GitHub Pages enabled and working
- Site accessible at [https://\[your-username\].github.io/dataManipulationChallenge/](https://[your-username].github.io/dataManipulationChallenge/)

#### **75% Grade Requirements:**

- Clear analysis of service levels by product category
- Discussion of business implications for ZappTech

**85% Grade Requirements:**

- Thorough analysis of cross-category ordering patterns
- Explanation of what this tells us about customer behavior

**95% Grade Requirements:**

- Complete advanced method chain using all seven mental models
- Discussion of technical approach and business value

**100% Grade Requirements:**

- Professional presentation style appropriate for business audience
- Clear, engaging narrative that tells a compelling story
- Practical insights that would help ZappTech's management

**Report Quality (Critical for Higher Grades):**

- Clear, engaging narrative that tells a story
- Focus on the most interesting findings about ZappTech's operations
- Professional writing style (no AI-generated fluff)
- Concise analysis that gets to the point
- Practical insights that would help a real business
- Well-designed visualizations that support your analysis