Data Visualization Analysis

Transforming Data into Business Insights

Brian Kim (Editor)

Global Master of Business Management, Tunghai University

Draft

December 17, 2025

Table of contents

1 Module 1: Foundations of Python and Data Handling 3
1.1 Section 1: Introduction to Python and VSCode Setup . . . 3
1.1.1 Objective . . . 3
1.1.2 Main Contents with Examples . . . 3
1.1.3 Lab Session . . . 6
1.2 Section 2: Python Basics for Data Visualization . . . 7
1.2.1 Objective . . . 7
1.2.2 Main Contents with Examples . . . 7
1.2.3 Lab Session . . . 11
1.3 Section 3: Data Structures and Pandas Fundamentals . . . 13
1.3.1 Objective . . . 13
1.3.2 Main Contents with Examples . . . 13
1.3.3 Lab Session . . . 17
1.4 Section 4: Data Cleaning and Preparation . . . 20
1.4.1 Objective . . . 20
1.4.2 Main Contents with Examples . . . 20
1.4.3 Lab Session . . . 25

2 Module 2: Core Visualization with Matplotlib 29
2.1 Section 1: Introduction to Matplotlib and Basic Plots . . . 29
2.1.1 Objective . . . 29
2.1.2 Main Contents with Examples . . . 29
2.1.3 Lab Session . . . 34
2.2 Section 2: Customizing Visualizations (Colors, Labels, Legends) . . . 37
2.2.1 Objective . . . 37
2.2.2 Main Contents with Examples . . . 37
2.2.3 Lab Session . . . 45
2.3 Section 3: Multiple Plots and Subplots . . . 49
2.3.1 Objective . . . 49
2.3.2 Main Contents with Examples . . . 49
2.3.3 Lab Session . . . 56
2.4 Section 4: Business Charts (Bar, Line, and Pie Charts) . . . 59
2.4.1 Objective . . . 59
2.4.2 Main Contents with Examples . . . 59
2.4.3 Lab Session . . . 66

3 Module 3: Statistical Visualization with Seaborn 71
3.1 Section 1: Introduction to Seaborn and Statistical Plots . . . 71
3.1.1 Objective . . . 71
3.1.2 Main Contents with Examples . . . 71
3.1.3 Lab Session . . . 77
3.2 Section 2: Distribution and Relationship Visualizations . . . 82
3.2.1 Objective . . . 82
3.2.2 Main Contents with Examples . . . 82
3.2.3 Lab Session . . . 90
3.3 Section 3: Categorical Data Visualization . . . 95

3.3.1 Objective 95
3.3.2 Main Contents with Examples 95
3.3.3 Lab Session 105
3.4 Section 4: Heatmaps and Correlation Analysis 111
3.4.1 Objective 111
3.4.2 Main Contents with Examples 111
3.4.3 Lab Session 120

4 Module 4: Interactive Visualization with Plotly 127
4.1 Section 1: Introduction to Plotly and Interactive Charts 127
4.1.1 Objective 127
4.1.2 Main Contents with Examples 127
4.1.3 Lab Session 135
4.2 Section 2: Advanced Interactive Visualizations 142
4.2.1 Objective 142
4.2.2 Main Contents with Examples 142
4.2.3 Lab Session 154
4.3 Section 3: Creating Dashboards and Business Reports 160
4.3.1 Objective 160
4.3.2 Main Contents with Examples 160
4.3.3 Lab Session 172
4.4 Section 4: Final Project - Comprehensive Business Data Analysis 178
4.4.1 Objective 178
4.4.2 Main Contents with Examples 178
4.4.3 Lab Session 189

Chapter 1

Module 1: Foundations of Python and Data Handling

1.1 Section 1: Introduction to Python and VSCode Setup

1.1.1 Objective

- Understand the role of Python in data visualization and business analytics
- Install and configure Python and VSCode for data analysis projects
- Navigate the VSCode interface and execute Python code
- Install essential packages (pandas, matplotlib, seaborn, plotly)

1.1.2 Main Contents with Examples

Why Python for Business Data Visualization?

Python has become the leading tool for business data visualization and analytics for several important reasons:

- **Versatility**: Python can handle everything from data collection to cleaning, analysis, and visualization in one environment
- **Business-Friendly Libraries**: Libraries like Pandas make working with business data (sales reports, financial statements, customer data) intuitive and powerful
- **Professional Visualizations**: Create publication-ready charts and interactive dashboards that can be shared with stakeholders
- **Automation**: Unlike Excel, Python can automate repetitive tasks, saving hours of manual work
- **Scalability**: Handle datasets from hundreds to millions of rows without performance issues
- **Career Value**: Python skills are highly demanded in business analytics, data science, and consulting roles

Comparison with Excel:

While Excel is excellent for quick analyses and familiar to most business users, Python offers advantages for serious data work:

- Excel becomes slow with large datasets (>100,000 rows), Python handles millions efficiently
- Python provides reproducibility - your analysis is documented in code
- Complex visualizations that require many manual steps in Excel can be automated in Python
- Python is free and open-source, while Excel requires licensing

Installing Python and VSCode

Step 1: Installing Python

Visit python.org and download the latest version of Python (3.10 or higher recommended):

- For Windows: Download the Windows installer (64-bit)

- For Mac: Download the macOS installer
- **Important**: During installation, check the box "Add Python to PATH" - this is crucial for accessing Python from any folder

After installation, verify by opening your command prompt (Windows) or terminal (Mac) and typing:

```
python --version
```

You should see something like "Python 3.11.5"

Step 2: Installing VSCode

VSCode (Visual Studio Code) is a free, powerful code editor developed by Microsoft:

- Visit code.visualstudio.com
- Download the version for your operating system
- Install with default settings

Why VSCode for data visualization?

- Clean, uncluttered interface ideal for beginners
- Excellent Python support through extensions
- Integrated terminal - no need to switch between windows
- IntelliSense (auto-completion) helps you write code faster
- Free and cross-platform

Understanding the VSCode Interface

When you first open VSCode, you'll see several key areas:

Left Sidebar (Activity Bar):

- **Explorer** (folder icon): Shows your project files and folders
- **Search** (magnifying glass): Find text across all files
- **Source Control** (branches): For version control (advanced feature)
- **Extensions** (blocks): Where we'll install the Python extension

Center Area (Editor):

- This is where you'll write your Python code
- You can open multiple files in tabs, just like a web browser

Bottom Panel:

- **Terminal**: Command line interface where you'll run Python programs
- **Problems**: Shows errors in your code
- **Output**: Shows results from running programs

Getting Started Workflow:

1. Create a project folder on your computer (e.g., "DataViz_Course")
2. In VSCode, go to File → Open Folder, select your project folder
3. The folder structure will appear in the Explorer sidebar
4. Create new files by right-clicking in Explorer → New File

Setting Up Python in VSCode

Installing the Python Extension:

1. Click the Extensions icon in the left sidebar (looks like building blocks)
2. Search for "Python" in the search box
3. Find the official Python extension by Microsoft
4. Click "Install"

This extension provides:

- Syntax highlighting (colors code for readability)
- Code completion (suggests as you type)
- Error detection (underlines problems)
- Ability to run Python code directly

Selecting Python Interpreter:

After installing the extension:

1. Press `Ctrl+Shift+P` (Windows) or `Cmd+Shift+P` (Mac) to open Command Palette
2. Type “Python: Select Interpreter”
3. Choose the Python version you installed

Package Management with pip

Pip is Python’s package manager - think of it as an app store for Python libraries. It comes pre-installed with Python.

Understanding Packages:

- **pandas**: Excel on steroids - handles tables of data (DataFrames)
- **matplotlib**: The foundation for creating static charts and graphs
- **seaborn**: Makes statistical visualizations beautiful with minimal code
- **plotly**: Creates interactive, web-based visualizations

Installing Packages:

Open the terminal in VSCode (View → Terminal or Ctrl+‘) and install each package:

```
pip install pandas
pip install matplotlib
pip install seaborn
pip install plotly
```

What’s happening behind the scenes:

- Pip connects to the Python Package Index (PyPI) online
- Downloads the requested package and its dependencies
- Installs everything in your Python environment
- Makes the package available for import in your code

Troubleshooting Tips:

- If “pip” is not recognized, try “python -m pip install pandas”
- If you get permission errors on Mac/Linux, don’t use sudo - instead create a virtual environment
- Installation may take a few minutes for all packages

Your First Python Program

Create a new file named `01_First_Program_Hello.py`:

```python
# This is a comment - Python ignores this line
# Comments help explain what your code does

print("Welcome to Data Visualization!")
print("Python is ready to analyze business data")
```

How to Run:

- Click the green “Run” button (play icon) in the top right, OR
- Right-click in the editor → “Run Python File in Terminal”

What print() does:

The `print()` function displays text or numbers in the terminal. Think of it as Python talking to you - showing results, messages, or data.

Verifying Package Installation

Create `02_Package_Test_Verification.py`:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

print("  All packages imported successfully!")
print("Pandas version:", pd.__version__)
print("Your data visualization environment is ready!")
```

Understanding imports:

- `import` loads a package so you can use its features
- `as pd` creates a short nickname (alias) to save typing
- Convention: pandas is `pd`, matplotlib.pyplot is `plt`, seaborn is `sns`

1.1.3 Lab Session

Lab 1: Setting Up Your Data Visualization Environment

Objective: Successfully install and configure your complete Python environment for data visualization work.

Pre-Lab Checklist:

- Computer with internet connection
- Administrator access (to install software)
- At least 2GB free disk space
- Notepad for writing down any error messages

Tasks:

Task 1: Software Installation (30 minutes)

1. Download and install Python from python.org
   - Remember to check "Add Python to PATH"
   - Write down the version number you installed
2. Download and install VSCode from code.visualstudio.com
3. Open VSCode and install the Python extension
4. Take a screenshot of VSCode with the Python extension installed

Task 2: Create Project Structure (10 minutes)

1. On your computer, create a folder structure:

   ```
   Documents/
     DataViz_Course/
         Module1/
         Module2/
         Module3/
         Module4/
   ```

2. Open the DataViz_Course folder in VSCode

3. Navigate to the Module1 folder in VSCode Explorer

Task 3: Install Required Packages (20 minutes) 1. Open the terminal in VSCode 2. Install each package one at a time: - pandas - matplotlib - seaborn - plotly 3. For each package, note if installation was successful 4. If errors occur, write them down to discuss with instructor

Task 4: Create Your First Python Program (15 minutes) 1. In the Module1 folder, create a file: `Lab01_YourName_Setup.py` 2. Write a program that: - Prints a welcome message with your name - Prints the course name "Data Visualization Analysis" - Calculates and prints: "Investment of $10,000 at 7% annual interest for 5 years" - Shows the result (Hint: 10000 * (1.07)**5) 3. Run the program successfully

Task 5: Verify Installation (10 minutes) 1. Create a new file: `Lab01_YourName_Verification.py` 2. Import all four visualization libraries 3. Print a success message 4. Print the version number of pandas 5. Run without errors

Deliverables:

1. Screenshot showing VSCode with your files and successful program output
2. Both Python files (.py) saved in Module1 folder
3. Written notes on any problems encountered and how you solved them

Success Criteria:

- □ Python installed and accessible from terminal
- □ VSCode installed with Python extension
- □ All four packages installed without errors
- □ Both Python programs run successfully
- □ Can create, save, and run .py files

1.2 Section 2: Python Basics for Data Visualization

1.2.1 Objective

- Master fundamental Python data types and variables for business data
- Perform mathematical and string operations relevant to business analytics
- Understand and use Python lists for managing multiple data points
- Apply conditional statements for business logic decisions
- Use loops to process repetitive data tasks
- Write reusable functions for common business calculations

1.2.2 Main Contents with Examples

Variables and Data Types - The Foundation of Data

Understanding Variables:

In business, we work with different types of information: company names, revenue figures, percentages, and yes/no decisions. Python uses variables to store this information.

Think of a variable as a labeled box where you store data: - The label is the variable name (you choose this) - The contents are the value (the actual data) - The type determines what kind of data you can store

The Four Essential Data Types:

1. Strings (Text Data) Used for: names, descriptions, categories, addresses

```
company_name = "TechCorp Industries"
product_category = "Electronics"
customer_feedback = "Excellent service!"
```

Rules for strings: - Always enclosed in quotes (single ’ or double “) - Can contain letters, numbers, spaces, punctuation - Case-sensitive:”Sales” and “sales” are different

2. Integers (Whole Numbers) Used for: counts, quantities, years, IDs

```
units_sold = 1500
employee_count = 247
fiscal_year = 2024
```

3. Floats (Decimal Numbers) Used for: money, percentages, measurements, rates

```
price = 299.99
profit_margin = 0.23
interest_rate = 5.75
```

4. Booleans (True/False) Used for: status flags, conditions, yes/no decisions

```
is_profitable = True
target_achieved = False
premium_customer = True
```

Why Data Types Matter:

Python treats different types differently: - You can do math with numbers: `100 + 50 = 150` - You can combine strings: `"Hello" + " World" = "Hello World"` - But you can't mix them incorrectly: `"Sales: " + 1000` causes an error

Mathematical Operations for Business Calculations

Basic Arithmetic:

Python uses standard mathematical operators, just like a calculator:

```python
# Quarterly Revenue Analysis
q1_revenue = 250000
q2_revenue = 280000

# Addition: Calculate total
total_revenue = q1_revenue + q2_revenue  # 530000

# Subtraction: Calculate growth
revenue_growth = q2_revenue - q1_revenue  # 30000

# Multiplication: Calculate with tax
revenue_with_tax = q2_revenue * 1.07  # 299600

# Division: Calculate average
average_revenue = total_revenue / 2  # 265000

# Exponentiation: Calculate compound growth
future_value = 100000 * (1.05)**3  # $115,762.50
```

Order of Operations:

Python follows standard mathematical rules (PEMDAS): 1. Parentheses: `(2 + 3)` 2. Exponentiation: `2**3` 3. Multiplication/Division: `*` and `/` 4. Addition/Subtraction: `+` and `-`

Example: `2 + 3 * 4` equals 14, not 20 To get 20: `(2 + 3) * 4`

Practical Business Example:

```python
# Calculate total cost with discount and tax
original_price = 1000
discount_rate = 0.15
tax_rate = 0.08

discounted_price = original_price * (1 - discount_rate)  # 850
final_price = discounted_price * (1 + tax_rate)  # 918
```

Working with Strings in Business Context

String Formatting for Professional Output:

The `f-string` (formatted string) is your tool for creating professional, readable output:

```python
company = "TechCorp"
revenue = 1500000
growth = 0.15

# Old way (hard to read):
print("Company: " + company + " Revenue: " + str(revenue))

# Modern way (f-string):
print(f"{company} generated ${revenue:,} in revenue")
```

```
# Output: TechCorp generated $1,500,000 in revenue

print(f"Growth rate: {growth:.1%}")
# Output: Growth rate: 15.0%
```

Formatting Options:

- `:,` adds thousand separators: `1500000` → `1,500,000`
- `:.2f` shows 2 decimal places: `3.14159` → `3.14`
- `:.1%` shows as percentage: `0.15` → `15.0%`

Lists - Managing Multiple Data Points

Understanding Lists:

In business, we rarely deal with single data points. We have monthly sales, multiple products, various regions. Lists let us store related data together.

A list is an ordered collection of items, like a spreadsheet column:

```
monthly_sales = [45000, 52000, 48000, 61000, 58000, 63000]
```

Why Lists are Powerful:

- Keep related data organized in one variable
- Maintain order (January is always first)
- Can grow or shrink as needed
- Can be processed with loops (covered next)

List Indexing:

Python uses "zero-based" indexing - counting starts at 0:

```
months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun"]

# Accessing items:
first_month = months[0]      # "Jan" (position 0)
third_month = months[2]      # "Mar" (position 2)
last_month = months[-1]      # "Jun" (negative counts from end)
```

Built-in List Functions:

Python provides functions to analyze lists quickly:

```
sales = [45000, 52000, 48000, 61000, 58000, 63000]

total_sales = sum(sales)       # 327000
average_sales = sum(sales) / len(sales)  # 54500
highest_sales = max(sales)     # 63000
lowest_sales = min(sales)      # 45000
number_of_months = len(sales)  # 6
```

Conditional Statements - Making Decisions

The Business Logic of Programming:

Businesses make decisions based on conditions: "If sales exceed target, pay bonus." Python uses `if` statements for this logic.

Basic Structure:

```
if condition:
    # do something when condition is True
```

Real Business Example:

```
actual_sales = 85000
target = 80000

if actual_sales >= target:
    print("Congratulations! Target achieved!")
    bonus = 5000
else:
    print("Target not met this month")
    bonus = 0
```

Multiple Conditions:

```
score = 78

if score >= 90:
    grade = "A - Excellent"
elif score >= 80:
    grade = "B - Good"
elif score >= 70:
    grade = "C - Satisfactory"
else:
    grade = "D - Needs Improvement"
```

The elif is short for "else if" - Python checks conditions from top to bottom - Executes the first True condition - Skips the rest

Loops - Processing Repetitive Data

Why Loops Matter:

Imagine you have 50 products, and need to calculate profit for each. You could write the same code 50 times, or use a loop to repeat the logic automatically.

For Loops - Processing Each Item:

```
products = ["Laptop", "Mouse", "Keyboard", "Monitor"]
prices = [1200, 25, 75, 350]

print("Product Price List:")
print("-" * 30)

for i in range(len(products)):
    print(f"{products[i]}: ${prices[i]}")
```

Understanding range():

- `range(4)` produces: 0, 1, 2, 3
- `range(len(products))` adjusts to list size automatically
- `i` is the loop variable that changes each iteration

Simpler Loop Pattern:

```
sales = [45000, 52000, 48000, 61000]

total = 0
for sale in sales:
    total = total + sale  # or: total += sale

average = total / len(sales)
```

When to Use Loops:

- Calculating totals or averages

- Applying the same operation to many items
- Searching through data
- Generating reports with many entries

Functions - Reusable Business Logic

The Problem Functions Solve:

Suppose you calculate profit margins often. Instead of rewriting the formula repeatedly, write it once as a function, then reuse it.

Creating a Function:

```python
def calculate_profit_margin(revenue, cost):
    profit = revenue - cost
    margin = (profit / revenue) * 100
    return margin

# Using the function:
margin1 = calculate_profit_margin(150000, 98000)  # 34.67%
margin2 = calculate_profit_margin(200000, 145000)  # 27.5%
margin3 = calculate_profit_margin(75000, 52000)   # 30.67%
```

Function Anatomy:

1. `def` keyword starts the function definition
2. Function name (use descriptive names)
3. Parameters in parentheses (inputs the function needs)
4. Colon `:` starts the function body
5. Indented code block (the function's logic)
6. `return` statement (the function's output)

Benefits of Functions:

- **Reusability**: Write once, use many times
- **Organization**: Complex programs become manageable
- **Testing**: Easy to verify one function works correctly
- **Collaboration**: Team members can use your functions

1.2.3 Lab Session

Lab 2: Python Programming for Business Analytics

Objective: Apply Python fundamentals to solve real business calculation problems.

Scenario: You are a business analyst at "RetailMax," a regional retail chain. Your manager has asked you to analyze recent sales data and create automated calculation tools.

Tasks:

Task 1: Semi-Annual Sales Analysis (25 points)

Create a file: `Lab02_YourName_SalesAnalysis.py`

You have six months of sales data for your store: - January: $45,000 - February: $52,000 - March: $48,000 - April: $61,000 - May: $58,000 - June: $63,000

Write a program that: 1. Stores the six sales figures in variables (jan_sales, feb_sales, etc.) 2. Calculates and prints: - Total sales for the 6-month period - Average monthly sales - Highest sales month (use max()) - Lowest sales month (use min()) - Difference between highest and lowest 3. Format all currency output with commas and 2 decimal places 4. Make output professional and easy to read

Expected Output Format:

```
RetailMax - Semi-Annual Sales Report
=====================================
Total Sales: $327,000.00
```

```
Average Monthly Sales: $54,500.00
Highest Month: $63,000.00
Lowest Month: $45,000.00
Sales Range: $18,000.00
```

Task 2: Product Performance Report (25 points)

Create a file: `Lab02_YourName_ProductReport.py`

You need to analyze five products: - Products: Laptop, Tablet, Smartphone, Headphones, Charger - Sales: $125,000, $87,000, $156,000, $23,000, $12,000

Write a program that: 1. Creates two lists (one for products, one for sales) 2. Uses a loop to print each product with its sales 3. Calculates total sales across all products 4. Finds and prints which product had the highest sales 5. Finds and prints which product had the lowest sales 6. Calculates what percentage of total sales each product represents

Hint for percentage: `(individual_sale / total_sales) * 100`

Task 3: Employee Commission Calculator (25 points)

Create a file: `Lab02_YourName_Commission.py`

Write a function called `calculate_commission()` that: - Takes sales amount as input - Calculates commission based on these rules: - Sales less than $50,000: 3% commission - Sales $50,000 - $99,999: 5% commission - Sales $100,000 or more: 8% commission - Returns the commission amount

Then test your function with these salespeople: - Alice: $45,000 - Bob: $78,000 - Charlie: $125,000 - Diana: $52,000

For each person, print: name, sales amount, commission rate, and commission earned.

Task 4: Quarterly Growth Analysis (25 points)

Create a file: `Lab02_YourName_GrowthAnalysis.py`

You have quarterly revenue data: - Q1: $450,000 - Q2: $485,000 - Q3: $510,000 - Q4: $545,000

Write a program that: 1. Stores the data in a list 2. Uses a loop to calculate quarter-over-quarter growth - Formula: `((current_quarter - previous_quarter) / previous_quarter) * 100` - Q2 growth = compare Q2 to Q1 - Q3 growth = compare Q3 to Q2 - Q4 growth = compare Q4 to Q3 3. Prints each growth rate formatted as a percentage 4. Calculates and prints average growth rate 5. Determines if growth is accelerating or decelerating

Deliverables:

- Four Python files (Lab02_YourName_*.py)
- Each program must run without errors
- Output must be clearly formatted and professional
- Include comments explaining your logic

Grading Rubric:

- Correct calculations: 50%
- Proper code structure and syntax: 25%
- Professional output formatting: 15%
- Code comments and clarity: 10%

Tips for Success:

- Test each task separately before moving to the next
- Use meaningful variable names
- Double-check your formulas
- Print intermediate results to verify calculations
- Ask for help if stuck for more than 15 minutes

1.3 Section 3: Data Structures and Pandas Fundamentals

1.3.1 Objective

- Understand dictionaries for storing structured business data
- Master Pandas DataFrames as the core tool for data analysis
- Read business data from CSV and Excel files
- Perform essential data exploration and summary operations
- Filter, sort, and group data to answer business questions
- Select and manipulate specific rows, columns, and cells

1.3.2 Main Contents with Examples

Dictionaries - Structured Data Storage

Moving Beyond Simple Lists:

Lists are great for single types of data (just sales numbers, just names). But business data is more complex - each product has a name, price, category, and quantity sold. We need a way to keep related information together.

Understanding Dictionaries:

A dictionary stores data in key-value pairs, like a real dictionary stores word-definition pairs:

- **Key**: The label or name (like "product_name")
- **Value**: The actual data (like "Laptop")

Think of it as a digital form where keys are the field labels and values are what you fill in:

```
employee = {
    "name": "Sarah Johnson",
    "department": "Marketing",
    "salary": 72000,
    "years_employed": 4,
    "is_manager": False
}

# Accessing information:
print(employee["name"])        # Sarah Johnson
print(employee["salary"])      # 72000
```

Why Dictionaries Matter:

- Keep related information together logically
- Access data by meaningful names, not positions
- Mirror how business data is actually structured
- Perfect for representing records (customers, products, employees)

Working with Multiple Records:

```
employees = [
    {"name": "Sarah", "dept": "Marketing", "salary": 72000},
    {"name": "Mike", "dept": "Sales", "salary": 68000},
    {"name": "Jennifer", "dept": "IT", "salary": 85000}
]

# Accessing specific data:
first_employee = employees[0]
first_name = employees[0]["name"]  # Sarah
```

The Limitation:

While dictionaries and lists work for small datasets, they become cumbersome with real business data. This is where Pandas comes in.

Introduction to Pandas DataFrames

What is Pandas?

Pandas is Python's data analysis library, and the DataFrame is its core tool. Think of a DataFrame as Excel inside Python: - Rows represent records (customers, sales transactions, products) - Columns represent attributes (name, price, quantity, date) - But with much more power than Excel

Why Pandas for Business Analysis? - Handles millions of rows efficiently - Built-in functions for common business calculations - Integrates seamlessly with visualization libraries - Reproducible analysis (code documents what you did)

Creating Your First DataFrame:

```python
import pandas as pd

# Business data: product sales
data = {
    "Product": ["Laptop", "Mouse", "Keyboard", "Monitor", "Headset"],
    "Price": [1200, 25, 75, 350, 80],
    "Units_Sold": [150, 500, 320, 200, 280],
    "Category": ["Electronics", "Accessories", "Accessories",
                 "Electronics", "Accessories"]
}

df = pd.DataFrame(data)
print(df)
```

Understanding the Structure:

- Each key in the dictionary becomes a column name
- Each list of values becomes a column
- All lists must be the same length
- Pandas automatically adds row numbers (index)

DataFrame Output:

```
    Product  Price  Units_Sold     Category
0    Laptop   1200         150  Electronics
1     Mouse     25         500  Accessories
2  Keyboard     75         320  Accessories
3   Monitor    350         200  Electronics
4   Headset     80         280  Accessories
```

Exploring DataFrames - Essential Commands

Getting to Know Your Data:

Before analyzing, you must understand what you're working with. Pandas provides powerful exploration tools:

1. head() and tail() - Preview Data

```python
df.head()      # Shows first 5 rows (default)
df.head(3)     # Shows first 3 rows
df.tail()      # Shows last 5 rows
```

Why this matters: Large datasets can't be printed entirely. Head/tail gives you a quick look without overwhelming output.

2. info() - Data Structure Report

```python
df.info()
```

Shows: - Number of rows and columns - Column names - Data types of each column (int, float, object/string) - Memory usage - Missing values (non-null counts)

Business use: Verify data loaded correctly, check for missing values, confirm data types.

3. describe() - Statistical Summary

```
df.describe()
```

For numeric columns, shows: - Count (number of values) - Mean (average) - Standard deviation (spread) - Min, Max (range) - Quartiles (25%, 50%, 75%)

Business use: Quick overview of sales ranges, average prices, revenue distribution.

4. shape, columns, and dtypes

```
df.shape      # (5, 4) means 5 rows, 4 columns
df.columns    # List of column names
df.dtypes     # Data type of each column
```

Adding Calculated Columns

Business Calculations in DataFrames:

One of Pandas' most powerful features is creating new columns from existing ones:

```
# Calculate total revenue per product
df["Revenue"] = df["Price"] * df["Units_Sold"]
```

What happens:

- Pandas multiplies each price by corresponding units
- Creates a new column called "Revenue"
- All done in one line (no loops needed!)

This is called vectorized operations - Pandas applies the operation to entire columns at once, making it fast and efficient.

Multiple Calculations:

```
df["Revenue"] = df["Price"] * df["Units_Sold"]
df["Revenue_Per_Unit"] = df["Revenue"] / df["Units_Sold"]
df["Price_Category"] = df["Price"] > 100  # True/False column
```

Reading Data from Files

Real Business Data Sources:

In practice, you'll rarely type data manually. Business data comes from: - CSV files (Comma-Separated Values) - universal format - Excel files (.xlsx) - common in business - Databases (SQL) - for large organizations - APIs (web data) - for real-time information

Reading CSV Files:

```
# Basic CSV reading
sales_data = pd.read_csv("sales_report.csv")

# With options
sales_data = pd.read_csv("sales_report.csv",
                         encoding="utf-8",      # Character encoding
                         thousands=",",          # Handle 1,000 format
                         parse_dates=["Date"])   # Convert date columns
```

Reading Excel Files:

```
financial_data = pd.read_excel("Q4_Report.xlsx")

# Specific sheet
financial_data = pd.read_excel("Annual_Report.xlsx",
                               sheet_name="Q4 Sales")
```

Common Issues and Solutions:

- **File not found:** Use complete file path or put file in same folder as Python script
- **Encoding errors:** Try `encoding="latin-1"` or `encoding="ISO-8859-1"`
- **Date not recognized:** Use `parse_dates=["Date_Column"]`

Accessing Data - Columns, Rows, and Cells

Selecting Columns:

There are two ways to select columns:

```
# Method 1: Single column (returns Series)
products = df["Product"]

# Method 2: Multiple columns (returns DataFrame)
subset = df[["Product", "Price", "Revenue"]]
```

Important distinction:

- Single brackets `[]` with one name = Series (one column)
- Double brackets `[[]]` with list of names = DataFrame (multiple columns)

Selecting Rows:

Pandas offers two main methods:

iloc - Select by position (integer location):

```
first_row = df.iloc[0]          # First row
first_three = df.iloc[0:3]      # Rows 0, 1, 2
last_row = df.iloc[-1]          # Last row
```

loc - Select by label:

```
row_zero = df.loc[0]            # Row with index label 0
rows_range = df.loc[0:2]        # Rows 0, 1, 2 (inclusive!)
```

Key difference: iloc uses position (0-based), loc uses index labels.

Selecting Specific Cells:

```
# Get price of first product
price = df.iloc[0, 1]             # Row 0, Column 1
price = df.loc[0, "Price"]        # Row 0, "Price" column
```

Filtering Data - Answering Business Questions

The Power of Conditional Selection:

Filtering lets you answer questions like: - "Which products cost more than $100?" - "Show me all electronics sales" - "Which months had revenue above target?"

Basic Filtering:

```
# Products priced over $100
expensive = df[df["Price"] > 100]

# Electronics only
electronics = df[df["Category"] == "Electronics"]

# High-volume sales (more than 250 units)
high_volume = df[df["Units_Sold"] > 250]
```

How it works:

1. `df["Price"] > 100` creates a True/False column
2. `df[True/False column]` keeps only True rows

Multiple Conditions:

Use `&` (AND) and `|` (OR), with parentheses around each condition:

```
# Electronics AND expensive
premium = df[(df["Category"] == "Electronics") & (df["Price"] > 100)]

# Either high price OR high volume
important = df[(df["Price"] > 200) | (df["Units_Sold"] > 300)]
```

Sorting and Grouping - Organizing Insights

Sorting for Rankings:

```
# Sort by revenue (lowest to highest)
df_sorted = df.sort_values("Revenue")

# Sort by revenue (highest to lowest)
df_sorted = df.sort_values("Revenue", ascending=False)

# Sort by multiple columns
df_sorted = df.sort_values(["Category", "Price"],
                           ascending=[True, False])
```

Grouping for Category Analysis:

Grouping is like pivot tables in Excel - summarize data by categories:

```
# Total revenue by category
category_revenue = df.groupby("Category")["Revenue"].sum()

# Multiple statistics
summary = df.groupby("Category").agg({
    "Revenue": "sum",
    "Units_Sold": "sum",
    "Price": "mean"
})
```

What groupby does:

1. Splits data into groups (by Category)
2. Applies function to each group (sum, mean, count, etc.)
3. Combines results into summary

Business Application:

- Compare regional sales
- Analyze product category performance
- Calculate per-customer metrics
- Identify trends across time periods

1.3.3 Lab Session

Lab 3: Working with Pandas DataFrames

Objective: Master Pandas fundamentals by analyzing real sales data and creating business reports.

Scenario: You work for "GlobalRetail Inc.," a company with sales teams across four regions. Your manager needs a comprehensive analysis of H1 (first half year) sales performance.

Pre-Lab Setup:

1. Create file: `Lab03_YourName_PandasAnalysis.py`
2. Import pandas at the top: `import pandas as pd`
3. All tasks should be in this one file, clearly labeled with comments

Task 1: Create Sales DataFrame (15 points)

Create a DataFrame with the following information for five salespeople:

Salesperson	Region	Q1_Sales	Q2_Sales
Alice Chen	North	150000	165000
Bob Martinez	South	135000	142000
Charlie Kim	East	162000	178000
Diana Lopez	West	148000	155000
Eve Johnson	North	158000	171000

Requirements:

1. Create the DataFrame from a dictionary
2. Print the entire DataFrame
3. Print the shape (rows, columns)
4. Print all column names
5. Print data types of each column

Task 2: Add Calculated Columns (20 points)

Add four new columns to your DataFrame:

1. **Total_Sales**: Sum of Q1_Sales and Q2_Sales
2. **Growth**: Difference between Q2_Sales and Q1_Sales (Q2 - Q1)
3. **Growth_Rate**: Percentage growth from Q1 to Q2
   - Formula: (Growth / Q1_Sales) * 100
4. **Performance_Tier**: Categorical rating based on Total_Sales
   - "Platinum": Total_Sales >= $320,000
   - "Gold": Total_Sales >= $300,000
   - "Silver": Total_Sales >= $280,000
   - "Bronze": Below $280,000

Hint for Performance_Tier: You'll need to use conditional logic. Look up `pd.cut()` or use multiple conditions with `np.where()`.

Print the updated DataFrame with all columns.

Task 3: Data Exploration and Analysis (20 points)

Answer these questions using Pandas methods (print each answer clearly labeled):

1. What is the total sales across all salespeople?
2. What is the average Q1 sales? Average Q2 sales?
3. Who had the highest Total_Sales? (use `idxmax()`)
4. Who had the lowest Growth_Rate?
5. What is the median Total_Sales?
6. Create a statistical summary of numeric columns using `describe()`

Task 4: Filtering and Segmentation (20 points)

Create and print the following filtered datasets:

1. **High_Performers**: Salespeople with Total_Sales > $315,000
   - Print the names and Total_Sales only
2. **Strong_Growth**: Salespeople with Growth > $12,000
   - Print all columns for these records
3. **North_Region**: Filter for North region only
   - Calculate the total sales for just North region
4. **Multi_Condition**: East OR West region AND Growth_Rate > 7%
   - Print Salesperson, Region, and Growth_Rate

Task 5: Regional Analysis (25 points)

Use `groupby()` to create a regional summary:

1. Group by Region and calculate:
    - Total Q1_Sales by region
    - Total Q2_Sales by region
    - Total Total_Sales by region
    - Average Growth_Rate by region
    - Count of salespeople per region
2. Print the regional summary table
3. Answer: Which region had the highest total sales? Which had the best average growth rate?
4. Sort the regional summary by Total_Sales (descending)

Bonus Task: CSV Operations (10 points)

1. Save your complete DataFrame to a CSV file: `Lab03_YourName_SalesData.csv`
2. Read it back into a new DataFrame called `df_reloaded`
3. Verify both DataFrames are identical by:
    - Comparing shapes
    - Comparing column names
    - Printing first 3 rows of reloaded data

Deliverables:

1. Python file: `Lab03_YourName_PandasAnalysis.py`
2. CSV file: `Lab03_YourName_SalesData.csv`
3. Document with answers to analysis questions (can be comments in Python file)

Expected Output Structure:

Your program should have clearly commented sections:

```
# ===== TASK 1: CREATE DATAFRAME =====
# (your code here)

# ===== TASK 2: ADD CALCULATED COLUMNS =====
# (your code here)

# ===== TASK 3: DATA EXPLORATION =====
# (your code here)
# etc.
```

Grading Rubric:

- Correct DataFrame creation: 15 points
- Accurate calculations: 25 points
- Proper filtering techniques: 20 points
- Successful groupby operations: 20 points
- Code organization and comments: 10 points
- CSV operations: 10 points

Common Mistakes to Avoid:

- Forgetting to assign new columns back to DataFrame
- Using single `&` or `|` without parentheses in conditions
- Confusing `iloc` and `loc`
- Not using meaningful variable names
- Printing without labels (we don't know what the output represents)

1.4 Section 4: Data Cleaning and Preparation

1.4.1 Objective

- Understand why data cleaning is critical for accurate business analysis
- Identify and handle missing values appropriately
- Detect and remove duplicate records
- Transform data types and standardize formats
- Identify and manage outliers that distort analysis
- Create a complete data cleaning workflow for business datasets

1.4.2 Main Contents with Examples

The Reality of Business Data

Why Data is Never Perfect:

In textbooks and tutorials, data is clean and ready to analyze. Real business data is messy: - Sales rep leaves "Region" blank on forms - Customer enters phone number as "555-1234" one time and "(555) 1234" another - Database crashes mid-entry leaving incomplete records - Someone accidentally enters sales as "$1,200,000" instead of "$120,000" - Same customer appears twice with slight name variations

The 80/20 Rule in Data Analysis:

Data professionals spend approximately 80% of their time cleaning data and only 20% on actual analysis. This isn't inefficiency - it's necessity. **Garbage in, garbage out** - analysis of dirty data produces unreliable results.

Types of Data Quality Issues:

1. **Missing values** - empty cells, NaN (Not a Number), null values
2. **Duplicates** - same record entered multiple times
3. **Inconsistent formats** - "New York" vs "NY" vs "new york"
4. **Wrong data types** - numbers stored as text
5. **Outliers** - extreme values that may be errors or real anomalies
6. **Invalid values** - negative quantities, future dates for past transactions

Understanding Missing Data

Types of Missing Data:

Not all missing data is the same. Understanding WHY data is missing affects how you handle it:

1. Missing Completely at Random (MCAR) - Pure chance - random data entry errors - Example: Scanner malfunction skips random records - Safe to delete these records

2. Missing at Random (MAR) - Missing relates to other variables - Example: Older customers more likely to skip "email" field - Consider filling with appropriate values

3. Missing Not at Random (MNAR) - Missingness is meaningful - Example: High earners don't report income - Deleting these records creates bias

Detecting Missing Values in Pandas:

```python
import pandas as pd
import numpy as np

# Create sample data with missing values
data = {
    "Date": ["2024-01-15", "2024-01-16", None, "2024-01-18"],
    "Product": ["Laptop", "Mouse", "Laptop", "Keyboard"],
    "Sales": [1200, 25, 1200, None],
    "Region": ["North", "South", "North", "South"]
}

df = pd.DataFrame(data)
```

```
# Check for missing values
print(df.isnull())           # True/False for each cell
print(df.isnull().sum())     # Count per column
print(df.info())             # Shows non-null counts
```

Strategies for Handling Missing Values:

Strategy 1: Deletion (when data is MCAR and abundant)

```
# Remove rows with ANY missing values
df_clean = df.dropna()

# Remove rows where specific column is missing
df_clean = df.dropna(subset=["Sales"])
```

When to use: Dataset is large, few missing values, missingness is random.

Strategy 2: Fill with Statistical Measure

```
# Fill missing sales with average
mean_sales = df["Sales"].mean()
df["Sales"].fillna(mean_sales, inplace=True)

# Or median (better for skewed data)
median_sales = df["Sales"].median()
df["Sales"].fillna(median_sales, inplace=True)
```

When to use: Numeric data, missingness is random, want to preserve dataset size.

Strategy 3: Fill with Placeholder

```
# Fill missing region with "Unknown"
df["Region"].fillna("Unknown", inplace=True)

# Fill missing dates with specific value
df["Date"].fillna("2024-01-17", inplace=True)
```

When to use: Categorical data, want to flag missing as separate category.

Strategy 4: Forward/Backward Fill

```
# Copy previous value down
df["Region"].fillna(method="ffill", inplace=True)

# Copy next value up
df["Region"].fillna(method="bfill", inplace=True)
```

When to use: Time series data where values change slowly.

Business Decision: The method you choose should be documented and justified. Different business contexts require different approaches.

Handling Duplicate Records

Why Duplicates Occur:

- Customer submits order twice (technical issue)
- Data merged from multiple sources
- Database synchronization errors
- Human error in data entry

Detecting Duplicates:

```
# Check for completely duplicate rows
print(df.duplicated())              # True/False for each row
print(df.duplicated().sum())        # Count of duplicates
print(df[df.duplicated()])          # View duplicate rows

# Check for duplicates in specific column
print(df.duplicated(subset=["Customer_ID"]))
```

Removing Duplicates:

```
# Remove duplicate rows (keeps first occurrence)
df_unique = df.drop_duplicates()

# Remove duplicates based on specific column
df_unique = df.drop_duplicates(subset=["Customer_ID"], keep="first")

# Options for 'keep':
# - "first": Keep first occurrence (default)
# - "last": Keep last occurrence
# - False: Remove all duplicates including original
```

Business Consideration: Before removing duplicates, verify they ARE duplicates: - Same customer might legitimately make multiple purchases - Same product might be sold multiple times - Check all relevant columns, not just one

Data Type Conversion

Why Data Types Matter:

Pandas must know data types to perform operations: - Mathematical operations require numeric types - Date operations require datetime type - String operations require object/string type

Common Type Issues:

- Numbers imported as text: “1500” instead of 1500
- Dates imported as text: “2024-01-15” instead of datetime
- Leading zeros lost: “00123” becomes 123

Checking and Converting Types:

```
# Check current data types
print(df.dtypes)

# Convert string to numeric
df["Sales"] = pd.to_numeric(df["Sales"])

# Convert to integer
df["Quantity"] = df["Quantity"].astype(int)

# Convert to datetime
df["Date"] = pd.to_datetime(df["Date"])

# Convert with error handling
df["Sales"] = pd.to_numeric(df["Sales"], errors="coerce")
# 'coerce' turns non-numeric values to NaN instead of error
```

Benefits of Correct Types:

- Enable proper sorting (1, 2, 10 instead of 1, 10, 2)
- Allow mathematical operations
- Enable date-based filtering and analysis
- Reduce memory usage

String Cleaning and Standardization

Common String Issues in Business Data:

- Inconsistent capitalization: “California” vs “california” vs “CALIFORNIA”
- Extra whitespace: ” Sales ” instead of “Sales”
- Different abbreviations: “New York” vs “NY”
- Special characters: “Department–Sales” vs “Department-Sales”

Pandas String Methods:

```
# Remove leading/trailing whitespace
df["Product"] = df["Product"].str.strip()

# Convert to consistent case
df["Region"] = df["Region"].str.lower()      # all lowercase
df["Region"] = df["Region"].str.upper()      # all uppercase
df["Region"] = df["Region"].str.title()      # Title Case
df["Region"] = df["Region"].str.capitalize() # Capitalize first

# Replace text
df["Category"] = df["Category"].str.replace("Accessories", "Access")

# Remove characters
df["Phone"] = df["Phone"].str.replace("-", "")  # Remove dashes
df["Phone"] = df["Phone"].str.replace(r"[^0-9]", "", regex=True)  # Keep only numbers
```

Creating Standardized Categories:

```
# Map variations to standard values
region_mapping = {
    "north": "North",
    "NORTH": "North",
    "N": "North",
    "Northern": "North"
}
df["Region"] = df["Region"].map(region_mapping)
```

Identifying and Handling Outliers

What Are Outliers?

Outliers are data points significantly different from others: - Sales of $1,000,000 when typical sales are $50,000 - Customer age of 150 years - Order quantity of -50

Why They Matter:

- Could be errors (data entry mistake)
- Could be real anomalies (legitimate big sale, VIP customer)
- Distort averages and statistical analyses
- Can mislead visualizations

Statistical Method: Interquartile Range (IQR)

The IQR method is a robust way to identify outliers:

```
# Calculate quartiles
Q1 = df["Sales"].quantile(0.25)    # 25th percentile
Q3 = df["Sales"].quantile(0.75)    # 75th percentile
IQR = Q3 - Q1                       # Interquartile range

# Define outlier boundaries
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

```
print(f"Normal range: ${lower_bound:,.0f} to ${upper_bound:,.0f}")
```

What these numbers mean:

- **Q1**: 25% of data is below this value
- **Q3**: 75% of data is below this value
- **IQR**: The middle 50% of data falls in this range
- **1.5 × IQR rule**: Standard statistical threshold for outliers

Handling Outliers:

```
# Identify outliers
outliers = df[(df["Sales"] < lower_bound) | (df["Sales"] > upper_bound)]
print("Outliers found:")
print(outliers)

# Option 1: Remove outliers
df_clean = df[(df["Sales"] >= lower_bound) & (df["Sales"] <= upper_bound)]

# Option 2: Cap outliers (replace with boundary values)
df["Sales_Capped"] = df["Sales"].clip(lower=lower_bound, upper=upper_bound)

# Option 3: Replace with median
median_sales = df["Sales"].median()
df["Sales_Clean"] = df["Sales"].apply(
    lambda x: median_sales if (x < lower_bound or x > upper_bound) else x
)
```

Business Decision: Don't automatically remove outliers: 1. Investigate: Is this a data error or legitimate? 2. Consider context: Holiday sales spike might be real 3. Document: Explain why outliers were kept or removed 4. Sometimes: Keep outliers, use median instead of mean

Creating a Complete Cleaning Pipeline

The Professional Approach:

Rather than cleaning ad-hoc, create a reusable function that applies all cleaning steps consistently:

```
def clean_sales_data(df):
    """
    Comprehensive data cleaning pipeline for sales data

    Steps:
    1. Remove duplicate records
    2. Handle missing values
    3. Standardize text fields
    4. Convert data types
    5. Remove outliers

    Returns: Cleaned DataFrame
    """

    print(f"Starting with {len(df)} records")

    # Step 1: Remove duplicates
    original_length = len(df)
    df = df.drop_duplicates()
    print(f"Removed {original_length - len(df)} duplicates")

    # Step 2: Handle missing values
    df["Sales"].fillna(df["Sales"].median(), inplace=True)
    df["Region"].fillna("Unknown", inplace=True)
    df = df.dropna(subset=["Date"])  # Date is critical

    # Step 3: Standardize strings
    df["Region"] = df["Region"].str.strip().str.title()
    df["Product"] = df["Product"].str.strip().str.title()

    # Step 4: Convert types
    df["Date"] = pd.to_datetime(df["Date"])
    df["Sales"] = pd.to_numeric(df["Sales"], errors="coerce")

    # Step 5: Remove outliers
    Q1 = df["Sales"].quantile(0.25)
    Q3 = df["Sales"].quantile(0.75)
    IQR = Q3 - Q1
    df = df[(df["Sales"] >= Q1 - 1.5*IQR) &
            (df["Sales"] <= Q3 + 1.5*IQR)]

    print(f"Finished with {len(df)} clean records")

    return df

# Usage
cleaned_df = clean_sales_data(raw_df)
```

Benefits of This Approach:

- Consistency across multiple datasets
- Easy to modify cleaning rules
- Documented process (code explains what you did)
- Reusable for similar datasets

1.4.3 Lab Session

Lab 4: Comprehensive Data Cleaning Project

Objective: Apply all data cleaning techniques to prepare a messy business dataset for analysis and visualization.

Scenario: You've been given a raw sales dataset exported from an old system. It has numerous quality issues that must be resolved before creating visualizations for the executive team. Your manager needs a clean dataset and a report documenting what was cleaned.

Part A: Create Messy Dataset (5 points)

Create file: `Lab04_YourName_DataCleaning.py`

Create this intentionally problematic dataset:

```
import pandas as pd
import numpy as np

data = {
    "Transaction_ID": [1001, 1002, 1003, 1004, 1005, 1006, 1002, 1007, 1008, 1009],
    "Date": ["2024-01-15", "2024-01-16", None, "2024-01-18", "2024-01-19",
             "2024-01-20", "2024-01-16", "2024-01-21", "2024-01-22", "2024-01-23"],
    "Product": ["  Laptop", "MOUSE  ", "laptop", "Keyboard", "  Monitor",
                "mouse", "MOUSE  ", "Tablet", "Headphones", "Laptop"],
    "Quantity": ["50", "200", "50", "150", "75", "200", "200", "100", "300", "2"],
    "Price": [1200, 25, 1200, 75, 350, 25, 25, 450, 80, 1200],
    "Sales": [60000, 5000, 60000, None, 26250, 5000, 5000, 45000, 24000, 2400],
    "Region": ["north", "SOUTH", "North", "south", None, "South", "SOUTH",
               "East", "West", "North"],
    "Sales_Rep": ["Alice", "Bob", "Alice", "Charlie", "Diana", "Bob", "Bob",
                  "Eve", "Frank", "Alice"]
}

df = pd.DataFrame(data)
print("Original messy data created")
print(df)
```

Part B: Data Quality Assessment (15 points)

Before cleaning, assess the problems:

1. **Check for missing values:**
    - Print count of missing values per column
    - Calculate percentage of missing data per column
    - Print rows that have any missing values
2. **Check for duplicates:**
    - Print duplicate rows (complete duplicates)
    - Check for duplicate Transaction_IDs
    - Count how many duplicates exist
3. **Check data types:**
    - Print current data type of each column
    - Identify which columns need type conversion
4. **Check for anomalies:**
    - Print basic statistics for numeric columns
    - Identify which values look suspicious
5. **Create a summary report:**

    Print a formatted report showing:

    - Total rows in original dataset
    - Number of missing values per column
    - Number of duplicate rows
    - Columns with wrong data types

Part C: Step-by-Step Cleaning (50 points)

Clean the data following these steps (print results after each step):

Step 1: Handle Duplicates (10 points)

- Identify and print duplicate rows
- Remove duplicate rows (keep first occurrence)
- Print how many duplicates were removed
- Print the DataFrame shape after removal

Step 2: Handle Missing Values (10 points)

- For “Sales” column: Fill missing values with the median sales
- For “Region” column: Fill missing values with “Unknown”
- For “Date” column: Fill missing values with “2024-01-17”
- Print the DataFrame to verify no missing values remain

Step 3: Standardize Text Fields (10 points)

- Product column: Remove whitespace, convert to title case
- Region column: Remove whitespace, capitalize properly

- Sales_Rep column: Ensure consistent capitalization
- Print unique values in Product and Region to verify

Step 4: Convert Data Types (10 points)

- Convert “Date” to datetime format
- Convert “Quantity” to integer type
- Convert “Price” to float type
- Verify “Sales” is float/numeric type
- Print data types to confirm changes

Step 5: Validate and Correct Business Logic (10 points)

- Recalculate Sales: Should equal Quantity × Price
- Create a new column “Sales_Calculated” with correct values
- Compare with existing “Sales” column
- Replace “Sales” with correct calculations
- Print any rows where Sales was incorrect

Part D: Outlier Analysis (15 points)

1. **Analyze Sales column for outliers:**
    - Calculate Q1, Q3, and IQR
    - Calculate lower and upper bounds
    - Print the bounds clearly
    - Identify and print outlier rows
2. **Business Decision:**
    - Review the outlier (Transaction_ID 1009)
    - This is a sale of 2 laptops at $1,200 each = $2,400
    - Determine if this is an error or legitimate
    - Document your decision with reasoning
3. **Create two versions:**
    - `df_with_outliers`: Keep all data
    - `df_no_outliers`: Remove outliers
    - Print record counts for both

Part E: Create Cleaning Function (10 points)

Write a reusable function called `clean_sales_dataset()` that: 1. Takes a raw DataFrame as input 2. Performs all cleaning steps from Parts C and D 3. Returns a clean DataFrame 4. Prints a summary of changes made

Test it by re-creating the messy dataset and running it through your function.

Part F: Export and Documentation (5 points)

1. Save the cleaned DataFrame to: `Lab04_YourName_CleanData.csv`
2. Create a text file: `Lab04_YourName_CleaningReport.txt` containing:
    - Original number of records
    - Number of duplicates removed
    - Number and location of missing values filled
    - Columns that were standardized
    - Data types that were converted
    - Number of outliers identified and decision made
    - Final number of clean records
    - Any assumptions or decisions you made

Deliverables:

1. `Lab04_YourName_DataCleaning.py` - Complete Python file with all parts
2. `Lab04_YourName_CleanData.csv` - Cleaned dataset
3. `Lab04_YourName_CleaningReport.txt` - Documentation of cleaning process

Grading Rubric:

- Data quality assessment: 15 points
- Cleaning steps completed correctly: 50 points
- Outlier analysis and justification: 15 points
- Reusable cleaning function: 10 points
- Documentation and export: 10 points

Bonus Challenge (+10 points):

Add advanced features to your cleaning function: - Add parameter to control whether to remove outliers - Add logging of each cleaning step to a file - Create before/after comparison visualizations - Handle edge cases (what if all values are duplicates?)

Tips for Success:

- Work through one step at a time
- Print intermediate results to verify each step
- Comment your code explaining WHY you made each decision
- Save your work frequently
- If stuck, clean the data manually first, then convert to function

Common Pitfalls to Avoid:

- Removing data without understanding why
- Using mean instead of median for skewed data
- Not verifying calculations (Sales = Quantity × Price)
- Over-cleaning (removing legitimate anomalies)
- Poor documentation (future you won't remember why you did something)

End of Module 1: Foundations of Python and Data Handling

Key Takeaways:

- Python and VSCode provide a powerful, professional environment for data work
- Python basics (variables, lists, loops, functions) enable automated data processing
- Pandas DataFrames are the core tool for business data analysis
- Real data is always messy - cleaning is essential before analysis
- Systematic cleaning workflows ensure consistent, reproducible results

You're now ready to move to Module 2: Visualization!

Chapter 2

Module 2: Core Visualization with Matplotlib

2.1 Section 1: Introduction to Matplotlib and Basic Plots

2.1.1 Objective

- Understand the role of data visualization in business communication
- Learn the fundamental architecture of Matplotlib
- Create basic line plots and scatter plots for business data
- Master the essential workflow for creating and displaying visualizations
- Understand when to use different plot types for different data stories

2.1.2 Main Contents with Examples

Why Data Visualization Matters in Business

The Power of Visual Communication:

Human brains process visual information 60,000 times faster than text. In business contexts, this matters enormously:

- **Executive Dashboards**: CEOs don't have time to read 50-page reports. A well-designed dashboard conveys quarterly performance at a glance.
- **Sales Presentations**: Showing a trend line of revenue growth is more persuasive than listing monthly numbers.
- **Operational Monitoring**: A spike in a line chart immediately alerts managers to problems.
- **Stakeholder Reports**: Investors understand market share better through pie charts than through percentages in text.

When Visualization Beats Tables:

Consider this sales data: - Q1: $450,000, Q2: $485,000, Q3: $510,000, Q4: $545,000

As a table, you need to mentally calculate: "Growth each quarter… looks consistent… upward trend…"

As a line chart, you immediately see: "Steady upward trajectory, accelerating slightly."

The Business Impact:

- **Faster decision-making**: Trends and patterns emerge instantly
- **Better communication**: Non-technical stakeholders understand visual data
- **Error detection**: Anomalies stand out visually but hide in tables
- **Persuasion**: Visual evidence is more convincing in presentations

Understanding Matplotlib Architecture

What is Matplotlib?

Matplotlib is Python's foundational visualization library, created in 2003 to bring MATLAB-style plotting to Python. Think of it as the "engine" that powers most Python visualizations - even libraries like Seaborn are built on top of Matplotlib.

Why Start with Matplotlib?

- **Industry standard**: Most widely used in business and science
- **Highly customizable**: Complete control over every visual element
- **Publication-ready**: Creates professional graphics for reports and papers
- **Foundation knowledge**: Understanding Matplotlib helps you master other libraries

The Two-Layer Architecture:

Matplotlib has two main components you need to understand:

1. Figure (The Canvas):

Think of this as your blank poster board or PowerPoint slide. It's the overall container for your visualization.

```python
import matplotlib.pyplot as plt

# Create a figure (canvas)
fig = plt.figure(figsize=(10, 6))  # Width: 10 inches, Height: 6 inches
```

The `figsize` parameter controls dimensions: - Standard report: (10, 6) - Wide dashboard: (12, 4) - Square social media: (8, 8) - Presentation slide: (12, 6)

2. Axes (The Plot Area):

This is where your actual chart lives - the area with data, axes, and grid. One figure can contain multiple axes (multiple charts).

```python
# Create figure and axes together
fig, ax = plt.subplots(figsize=(10, 6))
```

Understanding the Relationship:

- **Figure** = The entire window/image
- **Axes** = The chart(s) within that window
- You can have one figure with multiple axes (charts side-by-side)

Two Approaches to Matplotlib:

Approach 1: pyplot interface (simpler, good for quick plots)

```python
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [10, 20, 25, 30])
plt.show()
```

Approach 2: Object-oriented interface (professional, more control)

```python
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.plot([1, 2, 3, 4], [10, 20, 25, 30])
plt.show()
```

We'll use the object-oriented approach because: - More explicit and readable - Better for complex visualizations - Industry best practice - Easier to customize

Your First Matplotlib Visualization

The Basic Workflow:

Every Matplotlib visualization follows these steps:

1. Import the library
2. Prepare your data
3. Create figure and axes
4. Plot the data
5. Display the plot

Creating a Simple Line Plot:

```
import matplotlib.pyplot as plt

# Step 1: Prepare data (monthly sales)
months = [1, 2, 3, 4, 5, 6]
sales = [45000, 52000, 48000, 61000, 58000, 63000]

# Step 2: Create figure and axes
fig, ax = plt.subplots(figsize=(10, 6))

# Step 3: Create the plot
ax.plot(months, sales)

# Step 4: Display
plt.show()
```

What's Happening:

- `months` goes on the x-axis (horizontal)
- `sales` goes on the y-axis (vertical)
- `plot()` connects the points with lines
- `show()` displays the visualization

The plt.show() Command:

- In scripts: Required to display the plot window
- In Jupyter notebooks: Optional (plots display automatically)
- Pauses program execution until you close the plot window

Line Plots - Showing Trends Over Time

When to Use Line Plots:

Line plots are ideal for:

- **Time series data**: Sales over months, stock prices over days
- **Trends**: Showing direction of change
- **Continuous data**: Temperature, revenue, website traffic
- **Comparisons over time**: This year vs. last year

Not good for:

- Categorical comparisons (use bar charts)
- Part-to-whole relationships (use pie charts)
- Distributions (use histograms)

Anatomy of a Line Plot:

```
import matplotlib.pyplot as plt

# Quarterly revenue data
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
revenue = [450000, 485000, 510000, 545000]

# Create plot
fig, ax = plt.subplots(figsize=(10, 6))
```

```
ax.plot(quarters, revenue)

# Add labels (critical for business context)
ax.set_xlabel('Quarter')
ax.set_ylabel('Revenue ($)')
ax.set_title('Quarterly Revenue - 2024')

plt.show()
```

Essential Plot Elements:

- **Title**: What is this chart showing? (Always include!)
- **X-axis label**: What does the horizontal axis represent?
- **Y-axis label**: What does the vertical axis represent? (Include units!)
- **Data**: The actual line connecting points

Improving Readability:

```
import matplotlib.pyplot as plt

quarters = ['Q1', 'Q2', 'Q3', 'Q4']
revenue = [450000, 485000, 510000, 545000]

fig, ax = plt.subplots(figsize=(10, 6))

# Plot with markers to emphasize data points
ax.plot(quarters, revenue, marker='o', linewidth=2, markersize=8)

ax.set_xlabel('Quarter', fontsize=12)
ax.set_ylabel('Revenue ($)', fontsize=12)
ax.set_title('Quarterly Revenue Growth - 2024', fontsize=14, fontweight='bold')

# Add grid for easier reading
ax.grid(True, alpha=0.3)

plt.show()
```

Parameter Explanations:

- `marker='o'`: Adds circular markers at data points
- `linewidth=2`: Makes the line thicker (default is 1)
- `markersize=8`: Controls size of markers
- `grid(True, alpha=0.3)`: Adds grid lines with 30% opacity

Scatter Plots - Showing Relationships

When to Use Scatter Plots:

Scatter plots reveal relationships between two variables:

- **Correlation**: Does advertising spend correlate with sales?
- **Outliers**: Which customers have unusual behavior?
- **Clustering**: Do data points group together?
- **Distribution**: How spread out are the values?

Business Examples:

- Price vs. Sales Volume
- Marketing Spend vs. Revenue
- Employee Experience vs. Salary
- Customer Age vs. Purchase Amount

Creating a Scatter Plot:

```python
import matplotlib.pyplot as plt

# Employee data: years of experience vs. salary
experience = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
salary = [45000, 48000, 52000, 55000, 58000, 62000, 65000, 68000, 72000, 75000]

fig, ax = plt.subplots(figsize=(10, 6))

# Use scatter instead of plot
ax.scatter(experience, salary, s=100, alpha=0.6)

ax.set_xlabel('Years of Experience', fontsize=12)
ax.set_ylabel('Annual Salary ($)', fontsize=12)
ax.set_title('Salary vs. Experience Analysis', fontsize=14, fontweight='bold')

ax.grid(True, alpha=0.3)
plt.show()
```

Scatter Plot Parameters:

- `s=100`: Size of markers (s stands for size)
- `alpha=0.6`: Transparency (0=invisible, 1=opaque)
- No lines connecting points (unlike line plots)

Reading Scatter Plots:

What to look for:

- **Positive correlation**: Points trend upward (as X increases, Y increases)
- **Negative correlation**: Points trend downward (as X increases, Y decreases)
- **No correlation**: Points scattered randomly
- **Outliers**: Points far from the general pattern

Choosing Between Line and Scatter Plots

Decision Framework:

Use **Line Plots** when:

- X-axis represents time or sequential order
- You want to emphasize trends
- Connecting points makes sense
- Example: Monthly sales, daily temperature, yearly growth

Use **Scatter Plots** when:

- Investigating relationships between two variables
- Each point is an independent observation
- Connecting points doesn't make sense
- Example: Height vs. Weight, Age vs. Income, Price vs. Demand

The Same Data, Different Stories:

Imagine plotting store locations by (Latitude, Longitude):

- Scatter plot: Shows geographic distribution
- Line plot: Would incorrectly suggest a path between stores

Saving Your Visualizations

Why Save Plots:

In business, you'll need to:

- Include charts in reports (Word, PowerPoint)
- Share visualizations via email
- Post on internal dashboards

- Archive for compliance

Saving to File:

```
import matplotlib.pyplot as plt

# Create your plot
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot([1, 2, 3, 4], [10, 20, 25, 30])
ax.set_title('Sample Chart')

# Save before plt.show()
plt.savefig('monthly_sales.png', dpi=300, bbox_inches='tight')
plt.show()
```

Important Parameters:

- `dpi=300`: Resolution (300 is publication quality, 150 is screen quality)
- `bbox_inches='tight'`: Removes extra whitespace around the plot
- File extension determines format: .png, .pdf, .jpg, .svg

Format Guide:

- **PNG**: Best for reports, presentations, web (our go-to choice)
- **PDF**: Vector format, scales perfectly, good for print
- **JPG**: Smaller files but lower quality
- **SVG**: Vector format, editable in design software

File Naming Best Practices:

- Descriptive: `Q4_2024_Sales_Trend.png` not `plot1.png`
- Include date if relevant: `Revenue_2024_12_15.png`
- No spaces: Use underscores or hyphens

2.1.3 Lab Session

Lab 1: Creating Your First Business Visualizations

Objective: Master the fundamentals of Matplotlib by creating line plots and scatter plots for real business scenarios.

Scenario: You work for "TechGadget Inc.," an electronics retailer. The marketing director has requested visualizations to present at the upcoming board meeting. You'll create professional charts analyzing sales performance and customer patterns.

Pre-Lab Setup:

1. Create a new file: `M2L01_YourName_FirstPlots.py`
2. Import necessary libraries at the top
3. Create a folder called "Lab1_Outputs" to save your charts

Part A: Monthly Sales Trend Analysis (25 points)

Context: Your company's sales for the first half of 2024 need to be visualized to show the board how performance has trended.

Data:

- Months: January through June
- Sales: $125,000, $142,000, $138,000, $156,000, $151,000, $168,000

Tasks:

1. **Create a basic line plot (10 points):**
   - Use the object-oriented approach (fig, ax = plt.subplots())
   - Set figure size to (12, 6) for presentation format
   - Plot months on x-axis and sales on y-axis

   - Display the plot

2. **Add professional formatting (10 points):**
   - Add clear title: "TechGadget Inc. - H1 2024 Sales Performance"
   - Label x-axis: "Month"
   - Label y-axis: "Sales Revenue ($)"
   - Increase font sizes (title: 14pt bold, labels: 12pt)
   - Add markers at each data point with size 10
   - Make line thickness 2.5
   - Add a light grid for readability

3. **Save the visualization (5 points):**
   - Save as: "M2L01_YourName_MonthlySales.png"
   - Use high resolution (dpi=300)
   - Remove extra whitespace (bbox_inches='tight')

Expected Outcome: A professional line chart suitable for presentation that clearly shows sales growth over 6 months.

Part B: Product Price vs. Sales Analysis (30 points)

Context: The product team wants to understand if there's a relationship between product price and units sold to optimize pricing strategy.

Data:

Ten products with their prices and units sold: - Prices: $25, $45, $120, $85, $199, $350, $75, $450, $150, $299 - Units Sold: 850, 720, 245, 380, 180, 95, 425, 68, 210, 125

Tasks:

1. **Create a scatter plot (12 points):**
   - Figure size: (10, 8)
   - Plot price on x-axis and units sold on y-axis
   - Use scatter plot (not line plot)
   - Set marker size to 150
   - Set transparency (alpha) to 0.7

2. **Format for analysis (12 points):**
   - Title: "Product Price vs. Sales Volume Analysis"
   - X-axis label: "Product Price ($)" with font size 12
   - Y-axis label: "Units Sold" with font size 12
   - Title font size: 14pt, bold
   - Add a grid with 30% opacity
   - Use a color other than blue for markers (e.g., 'green', 'red', 'orange')

3. **Interpret and annotate (6 points):**
   - Look at your scatter plot
   - In a comment in your code, write 2-3 sentences describing:
     - What relationship (if any) do you see between price and units sold?
     - Are there any outlier products?
     - What business insight does this provide?

Part C: Comparative Analysis (25 points)

Context: The CEO wants to compare this year's performance against last year to see if the company is improving.

Data:

- Months: Jan, Feb, Mar, Apr, May, Jun
- 2023 Sales: $118,000, $125,000, $130,000, $135,000, $140,000, $145,000
- 2024 Sales: $125,000, $142,000, $138,000, $156,000, $151,000, $168,000

Tasks:

1. **Create a comparison line plot (15 points):**
   - Figure size: (12, 6)
   - Plot BOTH years on the same chart
   - 2023 line should be one color with one marker style
   - 2024 line should be a different color with different marker style
   - Both lines should have thickness 2
   - Markers should be size 8
2. **Add distinguishing features (10 points):**
   - Title: "Year-over-Year Sales Comparison: 2023 vs 2024"
   - Proper axis labels with units
   - Legend showing which line is which year
     - Use: `ax.legend(['2023', '2024'], fontsize=11)`
   - Grid for readability
   - Save as: "M2L01_YourName_YearComparison.png"

Part D: Customer Age vs. Purchase Amount (20 points)

Context: Marketing wants to understand if customer age correlates with purchase amount to target advertising appropriately.

Data:

- Customer Ages: 22, 35, 45, 28, 52, 38, 29, 48, 33, 41, 26, 55, 31, 44, 37
- Purchase Amounts: $85, $245, $420, $165, $580, $310, $190, $495, $225, $380, $140, $625, $210, $450, $290

Tasks:

1. **Create an analytical scatter plot (15 points):**
   - Figure size: (10, 7)
   - Appropriate title and axis labels
   - Marker size: 120
   - Use a professional color (not default blue)
   - Add transparency
   - Include grid
2. **Save and analyze (5 points):**
   - Save as: "M2L01_YourName_AgeAnalysis.png"
   - In comments, answer:
     - Is there a correlation between age and purchase amount?
     - What age group spends the most?
     - What recommendation would you make to marketing?

Bonus Challenge (+10 points):

Create one additional visualization of your choice using the provided data or your own business scenario: - Must use either line or scatter plot appropriately - Must be professionally formatted - Must include a business context (explain what it shows and why it matters) - Save as: "M2L01_YourName_Bonus.png"

Deliverables:

1. Python file: `M2L01_YourName_FirstPlots.py` with all parts clearly commented
2. Four PNG files (three required + bonus if attempted):
   - `M2L01_YourName_MonthlySales.png`
   - `M2L01_YourName_YearComparison.png`
   - `M2L01_YourName_AgeAnalysis.png`
   - `M2L01_YourName_Bonus.png` (if bonus attempted)

Grading Rubric:

- Part A (Monthly Sales): 25 points

- Part B (Price vs. Sales): 30 points
- Part C (Year Comparison): 25 points
- Part D (Age Analysis): 20 points
- Bonus: +10 points

Success Criteria:

☐ All plots display correctly when code is run
☐ Professional formatting applied to all visualizations
☐ Files saved with correct names and high quality
☐ Appropriate plot types used for each scenario
☐ Business insights documented in comments
☐ Code is well-organized with clear comments

Common Mistakes to Avoid:

- Forgetting to save plots before plt.show()
- Missing axis labels or titles
- Using line plots for non-sequential data
- Poor color choices (too bright, hard to read)
- File names with spaces or unclear naming
- Not setting appropriate figure sizes

Tips for Success:

- Test each part separately before moving to the next
- View your saved PNG files to verify quality
- Use meaningful variable names
- Comment your code as you go
- If a plot looks wrong, check your data first
- Reference the section examples for syntax

2.2 Section 2: Customizing Visualizations (Colors, Labels, Legends)

2.2.1 Objective

- Master color selection for professional, accessible visualizations
- Apply comprehensive labeling strategies for business clarity
- Create and position legends effectively for multi-series plots
- Control text properties (fonts, sizes, styles) for readability
- Format axes and ticks for professional presentations
- Understand design principles for business communication

2.2.2 Main Contents with Examples

The Importance of Customization in Business Context

Why Default Settings Aren't Enough:

Matplotlib's default visualizations are functional but generic. In business contexts, customization is essential because:

1. Branding and Professionalism:

- Corporate reports require company colors
- Presentations need consistent visual identity
- Professional appearance builds credibility

2. Accessibility:

- Default colors may not be colorblind-friendly

- Text might be too small for presentations
- Poor contrast makes charts unreadable

3. Clarity:

- Generic labels don’t provide business context
- Missing legends confuse viewers
- Poorly formatted numbers obscure insights

4. Storytelling:

- Colors can emphasize important data
- Annotations guide attention
- Formatting reinforces key messages

The Professional Standard:

Compare these descriptions:

- **Default**: Blue line, “x” label, “y” label
- **Professional**: Corporate blue (#1f77b4), “Fiscal Year 2024”, “Revenue (Millions USD)”, clear legend, formatted axis ticks, appropriate title

The second version communicates authority and attention to detail.

Working with Colors Effectively

Understanding Color in Matplotlib:

Colors in Matplotlib can be specified multiple ways: 1. **Named colors**: ‘red’, ‘blue’, ‘green’ (140 named colors available) 2. **Hex codes**: ‘#1f77b4’ (web standard) 3. **RGB tuples**: (0.5, 0.2, 0.8) (values 0-1) 4. **Built-in color maps**: ‘viridis’, ‘plasma’, ‘tab10’

Basic Color Application:

```python
import matplotlib.pyplot as plt

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
sales = [45000, 52000, 48000, 61000, 58000, 63000]

fig, ax = plt.subplots(figsize=(10, 6))

# Specify color
ax.plot(months, sales, color='darkblue', linewidth=2.5, marker='o')

ax.set_title('Monthly Sales Trend')
plt.show()
```

Common Business Color Choices:

Different colors convey different meanings in business:

- **Blue shades**: Trust, professionalism, stability (most common in business)
  - Navy: ‘#000080’, Royal: ‘#4169E1’, Sky: ‘#87CEEB’
- **Green shades**: Growth, profit, positive performance
  - Forest: ‘#228B22’, Money: ‘#2E7D32’, Mint: ‘#98D8C8’
- **Red shades**: Loss, decline, urgent attention
  - Crimson: ‘#DC143C’, Burgundy: ‘#800020’
- **Gray shades**: Neutral, supporting information
  - Dark: ‘#4A4A4A’, Medium: ‘#808080’, Light: ‘#D3D3D3’
- **Orange/Yellow**: Caution, moderate concern, highlights
  - Gold: ‘#FFD700’, Amber: ‘#FFBF00’

Color Psychology in Business Charts:

```python
# Profit/Loss visualization
months = ['Q1', 'Q2', 'Q3', 'Q4']
```

```python
profit = [120000, -45000, 85000, 150000]

fig, ax = plt.subplots(figsize=(10, 6))

# Color bars based on profit/loss
colors = ['green' if p > 0 else 'red' for p in profit]
ax.bar(months, profit, color=colors)

ax.set_title('Quarterly Profit/Loss Analysis')
ax.set_ylabel('Amount ($)')
plt.show()
```

Accessibility: Colorblind-Friendly Palettes:

Approximately 8% of men and 0.5% of women have color vision deficiency. Use colorblind-safe palettes:

Good combinations:

- Blue and Orange
- Blue and Yellow
- Purple and Green

Avoid:

- Red and Green (most common colorblindness)
- Red and Brown
- Blue and Purple (for some types)

Using Colorblind-Friendly Palettes:

```python
# IBM's colorblind-safe palette
colors = ['#648FFF', '#FFB000', '#DC267F', '#FE6100', '#785EF0']

# Or use built-in safe colormaps
ax.plot(x, y, color='#648FFF')  # Blue
ax.plot(x, y2, color='#FFB000')  # Orange
```

Comprehensive Labeling Strategies

The Three Essential Labels:

Every business visualization must have:

1. **Title**: What is this chart showing?
2. **X-axis label**: What does the horizontal axis represent?
3. **Y-axis label**: What does the vertical axis represent?

Missing any of these makes your chart ambiguous and unprofessional.

Creating Effective Titles:

Titles should be:

- **Specific**: Not “Sales Chart” but “Regional Sales Performance Q4 2024”
- **Informative**: Tell what, when, where if relevant
- **Concise**: One line preferred, two maximum
- **Professional**: Avoid casual language

```python
import matplotlib.pyplot as plt

data = [120, 145, 135, 160]
quarters = ['Q1', 'Q2', 'Q3', 'Q4']

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(quarters, data)
```

```python
# Poor title:
# ax.set_title('Sales')

# Better title:
# ax.set_title('Sales Data')

# Best title:
ax.set_title('TechGadget Inc. - Quarterly Sales Performance 2024',
             fontsize=14, fontweight='bold', pad=20)

ax.set_xlabel('Quarter', fontsize=12)
ax.set_ylabel('Sales ($1000s)', fontsize=12)

plt.show()
```

Title Parameters Explained:

- `fontsize=14`: Larger than body text (12) to emphasize
- `fontweight='bold'`: Makes title stand out
- `pad=20`: Adds space between title and plot (default is 6)

Axis Label Best Practices:

Always include units:

- Not: "Revenue"
- But: "Revenue ($ Millions)" or "Revenue ($)" or "Temperature (°F)"

Be specific:

- Not: "Time"
- But: "Month" or "Fiscal Quarter" or "Week Number"

Use appropriate precision:

- If numbers are in thousands: "Sales ($1000s)" or "Sales (Thousands)"
- If numbers are percentages: "Market Share (%)"

```python
# Complete labeling example
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(months, revenue)

ax.set_title('Monthly Revenue Analysis - Fiscal Year 2024',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Month', fontsize=12, fontweight='medium')
ax.set_ylabel('Revenue ($ Millions)', fontsize=12, fontweight='medium')

# Add subtitle if needed (smaller text below title)
ax.text(0.5, 1.05, 'North American Region',
        transform=ax.transAxes, ha='center', fontsize=10, style='italic')

plt.show()
```

Legends - Distinguishing Multiple Data Series

When Legends are Essential:

A legend is required when you have: - Multiple lines on one plot - Multiple categories in a bar chart - Different groups in a scatter plot - Comparison of different metrics

Without a legend, viewers can't distinguish what each line/color represents.

Creating Basic Legends:

```
import matplotlib.pyplot as plt

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
sales_2023 = [110, 125, 120, 135, 130, 145]
sales_2024 = [125, 142, 138, 156, 151, 168]

fig, ax = plt.subplots(figsize=(10, 6))

# Plot both years
ax.plot(months, sales_2023, marker='o', linewidth=2, label='2023')
ax.plot(months, sales_2024, marker='s', linewidth=2, label='2024')

# Create legend
ax.legend()

ax.set_title('Year-over-Year Sales Comparison')
ax.set_xlabel('Month')
ax.set_ylabel('Sales ($1000s)')

plt.show()
```

Key Point: Use the `label` parameter in your plot commands, then call `ax.legend()` to display the legend.

Positioning Legends Strategically:

Legend placement matters - it shouldn't obscure data:

```
# Position options:
ax.legend(loc='upper left')     # Top left
ax.legend(loc='upper right')    # Top right (default)
ax.legend(loc='lower left')     # Bottom left
ax.legend(loc='lower right')    # Bottom right
ax.legend(loc='center left')    # Middle left
ax.legend(loc='center')         # Dead center
ax.legend(loc='best')           # Matplotlib decides (usually good)

# Place outside plot area
ax.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
```

Customizing Legend Appearance:

```
fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(months, sales_2023, label='2023', linewidth=2.5)
ax.plot(months, sales_2024, label='2024', linewidth=2.5)

# Customized legend
ax.legend(
    loc='upper left',
    fontsize=11,
    frameon=True,           # Box around legend
    shadow=True,            # Drop shadow
    fancybox=True,          # Rounded corners
    title='Fiscal Year',    # Legend title
    title_fontsize=12
)

plt.show()
```

Multiple-Column Legends:

For many items, use columns to save space:

```
ax.legend(loc='upper center', ncol=3,  # 3 columns
          bbox_to_anchor=(0.5, -0.15))  # Below plot
```

Font Control and Text Styling

Why Typography Matters:

The right fonts make visualizations:

- More readable (especially in presentations)
- More professional
- Aligned with brand guidelines
- Appropriate for context (formal vs. informal)

Global Font Settings:

Set fonts for all text in a plot:

```
import matplotlib.pyplot as plt

# Set font for all elements
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.sans-serif'] = ['Arial', 'Helvetica', 'DejaVu Sans']
plt.rcParams['font.size'] = 11

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot([1, 2, 3], [10, 20, 15])
ax.set_title('Professional Typography')
plt.show()
```

Common Business Fonts:

- **Arial**: Clean, professional, universally available
- **Helvetica**: Similar to Arial, elegant
- **Times New Roman**: Traditional, formal reports
- **Calibri**: Modern, Microsoft default
- **DejaVu Sans**: Good Matplotlib default with many characters

Individual Text Customization:

```
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(months, sales)

# Title with custom font
ax.set_title('Quarterly Performance Report',
             fontsize=16,
             fontweight='bold',
             fontfamily='serif',
             color='#2C3E50')

# X-axis label
ax.set_xlabel('Quarter',
              fontsize=12,
              fontstyle='italic',
              fontweight='medium')

# Y-axis label
ax.set_ylabel('Revenue ($M)',
              fontsize=12,
              fontweight='bold')
```

```python
plt.show()
```

Font Property Options:

- `fontsize`: Number (points) or string ('small', 'medium', 'large', 'x-large')
- `fontweight`: 'normal', 'bold', 'heavy', 'light', or numeric (100-900)
- `fontstyle`: 'normal', 'italic', 'oblique'
- `fontfamily`: 'serif', 'sans-serif', 'monospace', 'fantasy', 'cursive'
- `color`: Any valid color specification

Text Hierarchy in Business Charts:

Create visual hierarchy through sizing:

- **Title**: 14-16pt, bold
- **Axis labels**: 11-12pt, medium weight
- **Tick labels**: 10-11pt, normal weight
- **Annotations**: 9-10pt, normal weight
- **Legend**: 10-11pt, normal weight

Axes Formatting and Number Display

The Challenge of Readability:

Compare these Y-axis displays: - `1500000` - Hard to read quickly - `1,500,000` - Better, but takes space - `1.5M` - Clear and compact - `$1.5M` - Even more informative

Formatting Axis Numbers:

```python
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

revenue = [1500000, 2300000, 1800000, 2750000]
quarters = ['Q1', 'Q2', 'Q3', 'Q4']

fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(quarters, revenue)

# Format Y-axis as millions with dollar sign
def millions_formatter(x, pos):
    return f'${x/1e6:.1f}M'

ax.yaxis.set_major_formatter(ticker.FuncFormatter(millions_formatter))

ax.set_title('Quarterly Revenue')
ax.set_ylabel('Revenue')

plt.show()
```

Common Number Formatters:

```python
import matplotlib.ticker as ticker

# Add thousand separators
ax.yaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))

# Percentage format
ax.yaxis.set_major_formatter(ticker.PercentFormatter(xmax=1))

# Currency format
ax.yaxis.set_major_formatter(ticker.StrMethodFormatter('${x:,.0f}'))
```

```
# Millions with 1 decimal
ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, p: f'{x/1e6:.1f}M'))

# Thousands shorthand
ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, p: f'{x/1000:.0f}K'))
```

Controlling Tick Marks:

```
# Set specific tick locations
ax.set_xticks([0, 1, 2, 3, 4, 5])
ax.set_xticklabels(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun'])

# Rotate tick labels for readability
ax.set_xticklabels(labels, rotation=45, ha='right')

# Set number of ticks
ax.yaxis.set_major_locator(ticker.MaxNLocator(6))  # Approximately 6 ticks
```

Bringing It All Together: Complete Customization

A Professional, Fully-Customized Chart:

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

# Data
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
revenue_2023 = [1100000, 1250000, 1200000, 1350000, 1300000, 1450000]
revenue_2024 = [1250000, 1420000, 1380000, 1560000, 1510000, 1680000]

# Create figure with specific size
fig, ax = plt.subplots(figsize=(12, 7))

# Plot data with custom colors and styling
ax.plot(months, revenue_2023,
        color='#7FB3D5',      # Soft blue
        linewidth=3,
        marker='o',
        markersize=8,
        label='FY 2023',
        alpha=0.8)

ax.plot(months, revenue_2024,
        color='#148F77',      # Dark green
        linewidth=3,
        marker='s',
        markersize=8,
        label='FY 2024',
        alpha=0.8)

# Comprehensive labeling
ax.set_title('TechGadget Inc. - Year-over-Year Revenue Comparison',
             fontsize=16,
             fontweight='bold',
             pad=20,
             color='#2C3E50')

ax.set_xlabel('Month',
              fontsize=13,
              fontweight='medium',
              color='#34495E')

ax.set_ylabel('Revenue',
              fontsize=13,
              fontweight='medium',
              color='#34495E')

# Format Y-axis as millions
ax.yaxis.set_major_formatter(
    ticker.FuncFormatter(lambda x, p: f'${x/1e6:.1f}M')
)

# Customize legend
ax.legend(loc='upper left',
          fontsize=11,
          frameon=True,
          shadow=True,
          fancybox=True,
          title='Fiscal Year')

# Add grid for readability
ax.grid(True, alpha=0.3, linestyle='--', linewidth=0.8)

# Adjust layout
plt.tight_layout()

# Save and display
plt.savefig('M2L02_Revenue_Comparison.png', dpi=300, bbox_inches='tight')
plt.show()
```

Design Principles Applied:

1. **Color**: Professional, accessible palette
2. **Contrast**: Clear difference between years
3. **Typography**: Hierarchical sizing
4. **Clarity**: Formatted axis, clear labels
5. **Professionalism**: Clean, polished appearance

2.2.3 Lab Session

Lab 2: Professional Chart Customization

Objective: Transform basic Matplotlib visualizations into polished, business-ready charts through comprehensive customization.

Scenario: You're preparing for the annual shareholders' meeting at "GlobalTech Solutions." The CFO has reviewed your initial charts but requests significant improvements to make them boardroom-ready. Your task is to apply professional customization to create presentation-quality visualizations.

Pre-Lab Setup:

1. Create file: `M2L02_YourName_Customization.py`
2. Import libraries: matplotlib.pyplot and matplotlib.ticker
3. Create output folder: "Lab2_Outputs"

Part A: Revenue Trend Enhancement (30 points)

Context: The basic revenue chart lacks the polish needed for executive presentation. Transform it into a professional visualization.

Data:

```
quarters = ['Q1 2024', 'Q2 2024', 'Q3 2024', 'Q4 2024']
revenue = [2400000, 2750000, 2650000, 3100000]
target = [2500000, 2600000, 2700000, 2800000]
```

Tasks:

1. **Create a dual-line comparison (12 points):**
   - Figure size: (12, 7)
   - Plot actual revenue with thick line (linewidth=3)
   - Plot target as dashed line (linestyle=‘–’)
   - Actual revenue: Dark blue color (#1F4788)
   - Target: Gray color (#7F8C8D)
   - Add circular markers (size 10) to actual revenue
   - Add square markers (size 10) to target
   - Include label for each line
2. **Apply professional formatting (12 points):**
   - Title: “GlobalTech Solutions - FY2024 Revenue vs. Target”
     - Font size: 16pt
     - Bold weight
     - Color: Dark gray (#2C3E50)
     - 20px padding above
   - X-axis label: “Quarter” (size 13, medium weight)
   - Y-axis label: “Revenue” (size 13, medium weight)
   - Format Y-axis numbers as: “$2.4M”, “$2.8M”, etc.
   - Add grid with 25% opacity, dashed lines
3. **Create an informative legend (6 points):**
   - Position: upper left
   - Font size: 12
   - Add frame with shadow
   - Rounded corners (fancybox=True)
   - Legend title: “Performance Metrics”

Part B: Regional Sales Customization (30 points)

Context: Create a professional bar chart comparing regional performance with clear visual hierarchy and formatting.

Data:

```
regions = ['North America', 'Europe', 'Asia Pacific', 'Latin America', 'Middle East']
sales = [5800000, 4200000, 6500000, 2100000, 1800000]
```

Tasks:

1. **Create a styled bar chart (10 points):**
   - Figure size: (12, 6)
   - Use horizontal bars (ax.barh() instead of ax.bar())
   - Bar color: Professional green (#27AE60)
   - Add edge color: Dark green (#1E8449), width 1.5
   - Set alpha (transparency) to 0.8
2. **Format for executive presentation (15 points):**
   - Title: “Regional Sales Performance - 2024”
     - Font size: 15pt, bold
     - Color: Navy (#154360)
   - X-axis label: “Annual Sales Revenue”
   - Y-axis label: “Region”
   - Both labels: size 12pt, medium weight

    - Format X-axis as millions: “$5.8M”, “$4.2M”, etc.
    - Rotate X-axis labels 45 degrees if needed for clarity

3. **Add data labels on bars (5 points):**
    - Display exact sales value on each bar
    - Format as: “$5.8M”
    - Position at end of each bar
    - Font size: 11pt
    - Hint: Use a loop with ax.text()

Part C: Multi-Series Comparison (25 points)

Context: The product team needs a comparison of three product lines over time with distinct, accessible styling.

Data:

```
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
product_a = [85000, 92000, 88000, 95000, 102000, 98000]
product_b = [62000, 68000, 71000, 74000, 78000, 81000]
product_c = [45000, 43000, 48000, 52000, 55000, 58000]
```

Tasks:

1. **Create accessible visualization (12 points):**
    - Figure size: (12, 6)
    - Use colorblind-friendly palette:
        - Product A: Blue (#648FFF)
        - Product B: Orange (#FFB000)
        - Product C: Purple (#785EF0)
    - Line width: 2.5 for all
    - Different markers for each: ‘o’, ‘s’, ‘^’
    - Marker size: 9
    - Add labels for legend

2. **Professional formatting (8 points):**
    - Title: “Product Line Performance - H1 2024”
    - Clear axis labels with units
    - Format Y-axis with thousand separators: “85,000”
    - Add grid with subtle styling
    - Legend positioned to not obscure data

3. **Enhance readability (5 points):**
    - Add slight transparency (alpha=0.85) to lines
    - Ensure all text is size 11pt or larger
    - Use consistent font throughout
    - Add padding where needed for spacing

Part D: Advanced Customization Challenge (15 points)

Context: Create a visualization showing profit margins over time with conditional formatting (colors change based on performance).

Data:

```
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
profit_margins = [0.15, 0.18, 0.12, 0.21]  # 15%, 18%, 12%, 21%
```

Tasks:

1. **Create a conditional bar chart (10 points):**
    - If margin >= 0.15 (15%): Color green (#27AE60)
    - If margin < 0.15: Color red (#E74C3C)

    - Add edge color (black, width 2)
    - Figure size: (10, 6)

2. **Format as percentages (5 points):**
    - Y-axis should show: 15%, 18%, 12%, 21%
    - Add horizontal reference line at 15% (target)
        - Use ax.axhline()
        - Dashed line, gray color
        - Label it "Target: 15%"
    - Professional title and labels

Bonus Challenge (+15 points):

Create a "Dashboard-Ready" Chart:

Using any of the previous datasets or your own business scenario:

1. Apply company branding:
    - Choose a professional 2-3 color scheme
    - Use consistent typography
    - Add subtle design elements (e.g., accent lines)
2. Maximize information density:
    - Include multiple relevant metrics
    - Add annotations for key insights
    - Format all numbers appropriately
3. Make it presentation-ready:
    - High resolution (dpi=300)
    - Perfect sizing for PowerPoint (16:9 ratio)
    - No matplotlib artifacts (tight layout)

Deliverables:

1. Python file: `M2L02_YourName_Customization.py` with all parts
2. Five PNG files:
    - `M2L02_YourName_RevenueVsTarget.png`
    - `M2L02_YourName_RegionalSales.png`
    - `M2L02_YourName_ProductComparison.png`
    - `M2L02_YourName_ProfitMargins.png`
    - `M2L02_YourName_Bonus.png` (if attempted)

Grading Rubric:

- Part A: 30 points
- Part B: 30 points
- Part C: 25 points
- Part D: 15 points
- Bonus: +15 points

Success Criteria:

☐ All visualizations are professionally formatted
☐ Colors are appropriate and accessible
☐ All axes are clearly labeled with units
☐ Number formatting is business-appropriate
☐ Legends are clear and well-positioned
☐ Text hierarchy is established through sizing
☐ Charts are saved at high resolution
☐ Code is well-commented

Professional Standards Checklist:

For each chart, verify:

☐ Title clearly states what is shown
☐ Both axes labeled with units

☐ Numbers formatted appropriately (millions, thousands, percentages)
☐ Legend included when needed
☐ Color choices are intentional and accessible
☐ Font sizes are appropriate (title largest, labels medium, ticks smallest)
☐ Grid added for readability (subtle, not overwhelming)
☐ No default matplotlib styling remaining
☐ Saved with descriptive filename
☐ High resolution for professional use

Tips for Success:

- Review Section 2 examples before starting
- Test color combinations for readability
- View saved images to verify quality
- Get feedback on visual clarity
- Use comment blocks to organize code
- Reference matplotlib documentation for specific formatters

2.3 Section 3: Multiple Plots and Subplots

2.3.1 Objective

- Understand when and why to use multiple plots in a single figure
- Master the subplot architecture for creating multi-panel visualizations
- Create effective comparison layouts for business dashboards
- Control spacing, sizing, and alignment of multiple plots
- Share axes across subplots when appropriate
- Design cohesive multi-plot figures for reports and presentations

2.3.2 Main Contents with Examples

The Business Case for Multiple Plots

Why Single Plots Aren't Always Enough:

Business analysis often requires viewing multiple perspectives simultaneously:

Scenario 1: Regional Comparison Instead of four separate charts for four regions, one figure with four subplots lets executives:

- Compare patterns across regions instantly
- Identify which regions are outperforming
- Spot regional trends vs. company-wide trends
- Present comprehensive analysis on one slide

Scenario 2: Multi-Metric Dashboard A product manager needs to see:

- Sales volume (line chart)
- Profit margins (bar chart)
- Customer satisfaction (line chart)
- Market share (area chart)

Four separate images become confusing. One dashboard figure tells the complete story.

Scenario 3: Time Period Comparison Comparing this year vs. last year:

- Option A: Overlay on same plot (can be cluttered)
- Option B: Side-by-side subplots (clearer comparison)
- Option C: Vertical stack (emphasizes temporal progression)

Benefits of Subplots:

1. **Space Efficiency**: Multiple insights on one page

2. **Easier Comparison**: Eye doesn't need to switch between pages
3. **Professional Presentation**: Cohesive, dashboard-like appearance
4. **Storytelling**: Guide viewers through related analyses
5. **Print-Friendly**: One figure to include in reports

When NOT to Use Subplots:

- Data relationships aren't related (create separate figures)
- Too many subplots (>6 becomes crowded)
- Plots need different scales that confuse comparison
- Each plot tells a completely different story

Understanding Subplot Architecture

The Conceptual Model:

Think of subplots as a grid:

- **Rows**: Vertical divisions
- **Columns**: Horizontal divisions
- **Position**: Each subplot occupies a cell (or multiple cells)

```
2 rows, 2 columns:

 (0,0)    (0,1)     Row 0

 (1,0)    (1,1)     Row 1

Column 0  Column 1
```

Basic Subplot Creation:

```python
import matplotlib.pyplot as plt

# Create a figure with 2x2 grid of subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Access individual subplots
axes[0, 0]  # Top-left
axes[0, 1]  # Top-right
axes[1, 0]  # Bottom-left
axes[1, 1]  # Bottom-right
```

Important Notes:

- `axes` is a 2D array if you have rows and columns
- `axes` is a 1D array if you have only rows OR columns
- Use `axes.flatten()` to convert 2D to 1D for easier iteration

Creating Different Layout Configurations:

```python
# One row, three columns (horizontal dashboard)
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Three rows, one column (vertical report)
fig, axes = plt.subplots(3, 1, figsize=(10, 12))

# 2x3 grid (six panels)
fig, axes = plt.subplots(2, 3, figsize=(15, 8))
```

Creating a Simple Multi-Plot Dashboard

Example: Quarterly Business Dashboard

```python
import matplotlib.pyplot as plt
import numpy as np

# Sample data
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
revenue = [2400000, 2750000, 2650000, 3100000]
expenses = [1800000, 2100000, 2000000, 2300000]
profit = [r - e for r, e in zip(revenue, expenses)]
customers = [15000, 17500, 18200, 21000]

# Create 2x2 dashboard
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Top-left: Revenue trend
axes[0, 0].plot(quarters, revenue, marker='o', linewidth=2, color='#2E86AB')
axes[0, 0].set_title('Quarterly Revenue', fontsize=13, fontweight='bold')
axes[0, 0].set_ylabel('Revenue ($)')
axes[0, 0].grid(True, alpha=0.3)

# Top-right: Profit margins
axes[0, 1].bar(quarters, profit, color='#06A77D', edgecolor='black', linewidth=1.5)
axes[0, 1].set_title('Quarterly Profit', fontsize=13, fontweight='bold')
axes[0, 1].set_ylabel('Profit ($)')
axes[0, 1].grid(True, alpha=0.3, axis='y')

# Bottom-left: Customer growth
axes[1, 0].plot(quarters, customers, marker='s', linewidth=2, color='#D9A72E')
axes[1, 0].set_title('Customer Growth', fontsize=13, fontweight='bold')
axes[1, 0].set_ylabel('Number of Customers')
axes[1, 0].grid(True, alpha=0.3)

# Bottom-right: Revenue vs. Expenses comparison
axes[1, 1].plot(quarters, revenue, marker='o', label='Revenue', linewidth=2)
axes[1, 1].plot(quarters, expenses, marker='s', label='Expenses', linewidth=2)
axes[1, 1].set_title('Revenue vs. Expenses', fontsize=13, fontweight='bold')
axes[1, 1].set_ylabel('Amount ($)')
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

# Overall title for dashboard
fig.suptitle('Quarterly Business Performance Dashboard - 2024',
             fontsize=16, fontweight='bold', y=0.995)

plt.tight_layout()
plt.savefig('M2L03_Business_Dashboard.png', dpi=300, bbox_inches='tight')
plt.show()
```

Key Techniques Applied:

1. Each subplot accessed by index: `axes[row, col]`
2. Each subplot formatted independently
3. `fig.suptitle()` adds overall title
4. `tight_layout()` prevents overlapping elements

Controlling Subplot Spacing and Layout

The Problem of Overlap:

Default spacing often causes:

- Titles overlapping adjacent plots

- Axis labels getting cut off
- Cramped appearance
- Unprofessional look

Solution 1: tight_layout() - Automatic Adjustment

```
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# ... create your plots ...

# Automatically adjust spacing
plt.tight_layout()
```

What `tight_layout()` does:

- Calculates optimal spacing
- Prevents overlapping elements
- Adjusts subplot sizes
- Usually "just works"

Solution 2: subplots_adjust() - Manual Control

For finer control:

```
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# ... create your plots ...

# Manual spacing adjustment
plt.subplots_adjust(
    left=0.1,       # Left margin
    right=0.95,     # Right margin
    top=0.93,       # Top margin
    bottom=0.1,     # Bottom margin
    wspace=0.3,     # Width space between subplots
    hspace=0.4      # Height space between subplots
)
```

When to use manual adjustment:

- `tight_layout()` doesn't work well
- Need consistent spacing across multiple figures
- Creating very specific layouts
- Preparing for publication with strict requirements

Sharing Axes Across Subplots

The Comparison Challenge:

When comparing across subplots, inconsistent axis scales mislead viewers:

Without shared axes:

- Plot A: Y-axis 0-1000
- Plot B: Y-axis 0-5000
- Visual impression: Similar heights, but values are 5x different!

Solution: Share X and/or Y Axes

```
# Share Y-axis across all subplots (common scale)
fig, axes = plt.subplots(2, 2, figsize=(12, 10), sharey=True)

# Share X-axis across all subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 10), sharex=True)
```

```
# Share both
fig, axes = plt.subplots(2, 2, figsize=(12, 10), sharex=True, sharey=True)
```

Practical Example: Regional Sales Comparison

```
import matplotlib.pyplot as plt

# Data for four regions
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
north = [45000, 52000, 48000, 61000, 58000, 63000]
south = [38000, 42000, 45000, 48000, 51000, 55000]
east = [52000, 58000, 54000, 67000, 64000, 70000]
west = [41000, 45000, 43000, 54000, 52000, 58000]

# Create 2x2 grid with shared Y-axis for fair comparison
fig, axes = plt.subplots(2, 2, figsize=(14, 10), sharey=True)

# North region
axes[0, 0].plot(months, north, marker='o', color='#2E86AB', linewidth=2)
axes[0, 0].set_title('North Region', fontsize=12, fontweight='bold')
axes[0, 0].grid(True, alpha=0.3)

# South region
axes[0, 1].plot(months, south, marker='o', color='#2E86AB', linewidth=2)
axes[0, 1].set_title('South Region', fontsize=12, fontweight='bold')
axes[0, 1].grid(True, alpha=0.3)

# East region
axes[1, 0].plot(months, east, marker='o', color='#2E86AB', linewidth=2)
axes[1, 0].set_title('East Region', fontsize=12, fontweight='bold')
axes[1, 0].set_xlabel('Month')
axes[1, 0].grid(True, alpha=0.3)

# West region
axes[1, 1].plot(months, west, marker='o', color='#2E86AB', linewidth=2)
axes[1, 1].set_title('West Region', fontsize=12, fontweight='bold')
axes[1, 1].set_xlabel('Month')
axes[1, 1].grid(True, alpha=0.3)

# Common Y-label (only on left plots due to sharey)
axes[0, 0].set_ylabel('Sales ($)', fontsize=11)
axes[1, 0].set_ylabel('Sales ($)', fontsize=11)

fig.suptitle('Regional Sales Performance - H1 2024', fontsize=15, fontweight='bold')

plt.tight_layout()
plt.show()
```

Benefits of Sharing Axes:

- Honest visual comparison (same scale)
- Less cluttered (fewer axis labels)
- Professional appearance
- Easier to spot performance differences

Mixing Different Plot Types

Telling Different Stories in One Figure:

Complex business analyses require different visualization types for different metrics.

Example: Comprehensive Product Analysis

```python
import matplotlib.pyplot as plt
import numpy as np

# Data
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
units_sold = [850, 920, 880, 1050, 1020, 1100]
price = [299, 299, 289, 289, 279, 279]
revenue = [u * p for u, p in zip(units_sold, price)]
satisfaction = [4.2, 4.3, 4.1, 4.5, 4.6, 4.7]

# Create 3-panel vertical layout
fig, axes = plt.subplots(3, 1, figsize=(12, 10))

# Panel 1: Sales Volume (Line Chart)
axes[0].plot(months, units_sold, marker='o', linewidth=2.5,
             color='#2E86AB', markersize=8)
axes[0].set_title('Units Sold', fontsize=13, fontweight='bold', loc='left')
axes[0].set_ylabel('Units', fontsize=11)
axes[0].grid(True, alpha=0.3)

# Panel 2: Revenue and Price (Dual Line Chart)
ax2_twin = axes[1].twinx()  # Create secondary Y-axis
axes[1].plot(months, revenue, marker='s', linewidth=2.5,
             color='#06A77D', label='Revenue', markersize=8)
ax2_twin.plot(months, price, marker='^', linewidth=2.5,
              color='#D9A72E', label='Price', markersize=8)
axes[1].set_title('Revenue & Pricing', fontsize=13, fontweight='bold', loc='left')
axes[1].set_ylabel('Revenue ($)', fontsize=11, color='#06A77D')
ax2_twin.set_ylabel('Price ($)', fontsize=11, color='#D9A72E')
axes[1].grid(True, alpha=0.3)

# Combine legends
lines1, labels1 = axes[1].get_legend_handles_labels()
lines2, labels2 = ax2_twin.get_legend_handles_labels()
axes[1].legend(lines1 + lines2, labels1 + labels2, loc='upper left')

# Panel 3: Customer Satisfaction (Bar Chart)
axes[2].bar(months, satisfaction, color='#8E44AD', edgecolor='black', linewidth=1.5)
axes[2].set_title('Customer Satisfaction', fontsize=13, fontweight='bold', loc='left')
axes[2].set_xlabel('Month', fontsize=11)
axes[2].set_ylabel('Rating (out of 5)', fontsize=11)
axes[2].axhline(y=4.0, color='red', linestyle='--', linewidth=2, label='Target: 4.0')
axes[2].legend()
axes[2].grid(True, alpha=0.3, axis='y')
axes[2].set_ylim([0, 5])

# Overall title
fig.suptitle('Product Performance Dashboard - H1 2024',
             fontsize=16, fontweight='bold')

plt.tight_layout()
plt.show()
```

Design Principles Demonstrated:

1. **Visual Hierarchy**: Most important metric on top
2. **Plot Type Selection**: Line for trends, bars for ratings
3. **Color Coding**: Different colors for different metrics

4. **Reference Lines**: Target satisfaction marked
5. **Consistent Formatting**: Similar styling across panels

Advanced Layouts with GridSpec

Beyond Simple Grids:

Sometimes you need asymmetric layouts where subplots have different sizes.

GridSpec Allows:

- Subplots spanning multiple cells
- Unequal subplot sizes
- Complex dashboard layouts
- Professional report structures

Example: Executive Summary Layout

```python
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

fig = plt.figure(figsize=(14, 10))

# Create custom grid: 3 rows, 3 columns
gs = gridspec.GridSpec(3, 3, figure=fig, hspace=0.4, wspace=0.3)

# Large main plot spanning top 2 rows, all columns
ax_main = fig.add_subplot(gs[0:2, :])

# Three smaller plots in bottom row
ax_bottom_left = fig.add_subplot(gs[2, 0])
ax_bottom_center = fig.add_subplot(gs[2, 1])
ax_bottom_right = fig.add_subplot(gs[2, 2])

# Main plot: Primary metric (revenue trend)
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
revenue = [2400000, 2750000, 2650000, 3100000]
ax_main.plot(quarters, revenue, marker='o', linewidth=3, markersize=12,
 ↪  color='#2E86AB')
ax_main.set_title('Annual Revenue Trend', fontsize=15, fontweight='bold')
ax_main.set_ylabel('Revenue ($M)', fontsize=12)
ax_main.grid(True, alpha=0.3)

# Bottom plots: Supporting metrics
# Left: Customer count
customers = [15000, 17500, 18200, 21000]
ax_bottom_left.bar(quarters, customers, color='#06A77D', edgecolor='black')
ax_bottom_left.set_title('Customers', fontsize=11, fontweight='bold')

# Center: Profit margin
margins = [0.25, 0.24, 0.25, 0.26]
ax_bottom_center.bar(quarters, margins, color='#D9A72E', edgecolor='black')
ax_bottom_center.set_title('Profit Margin', fontsize=11, fontweight='bold')

# Right: Market share
share = [0.15, 0.16, 0.17, 0.18]
ax_bottom_right.plot(quarters, share, marker='s', linewidth=2, color='#8E44AD')
ax_bottom_right.set_title('Market Share', fontsize=11, fontweight='bold')

fig.suptitle('Executive Dashboard - 2024 Annual Performance',
             fontsize=16, fontweight='bold')
```

```
plt.savefig('M2L03_Executive_Dashboard.png', dpi=300, bbox_inches='tight')
plt.show()
```

When to Use GridSpec:

- Dashboard with featured metric + supporting metrics
- Report with main chart + detail insets
- Comparison where one metric needs more emphasis
- Publications with specific layout requirements

2.3.3 Lab Session

Lab 3: Building Multi-Panel Dashboards

Objective: Create professional multi-plot visualizations that present comprehensive business analyses in cohesive, easy-to-understand formats.

Scenario: You're the Business Intelligence Analyst at "RetailMax Corporation." The Q4 Board Meeting is next week, and executives need comprehensive dashboards showing performance across multiple dimensions. You'll create three different multi-panel visualizations for different audiences.

Pre-Lab Setup:

1. Create file: `M2L03_YourName_Dashboards.py`
2. Import matplotlib.pyplot and any needed libraries
3. Create output folder: "Lab3_Outputs"

Part A: Regional Performance Dashboard (35 points)

Context: The VP of Sales needs to compare performance across four regions simultaneously to identify best practices and struggling areas.

Data:

```
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
northeast = [145000, 158000, 152000, 171000, 168000, 182000]
southeast = [132000, 138000, 145000, 151000, 158000, 165000]
midwest = [158000, 171000, 165000, 183000, 178000, 195000]
west = [141000, 145000, 148000, 162000, 171000, 178000]
```

Tasks:

1. **Create 2x2 regional comparison (20 points):**
   - Figure size: (14, 10)
   - Create four subplots (2 rows, 2 columns)
   - Share Y-axis across all subplots for fair comparison
   - Layout: Northeast (top-left), Southeast (top-right), Midwest (bottom-left), West (bottom-right)
   - Each subplot: Line plot with markers
   - Use consistent color (#2E86AB) across all regions
   - Line width: 2.5, marker size: 8
2. **Format each subplot (10 points):**
   - Title for each: "{Region} Region Sales"
   - X-label on bottom row only: "Month"
   - Y-label on left column only: "Sales Revenue ($)"
   - Grid on all plots (alpha=0.3)
   - Overall figure title: "Regional Sales Performance - H1 2024"
3. **Polish and save (5 points):**
   - Use tight_layout()
   - Save as: `M2L03_YourName_RegionalDashboard.png`
   - High resolution (dpi=300)

Part B: Product Performance Vertical Dashboard (30 points)

Context: The Product Manager needs a comprehensive view of how the flagship product is performing across multiple metrics.

Data:

```
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
units_sold = [8500, 9200, 11000, 12500]
avg_price = [349, 339, 329, 319]
customer_satisfaction = [4.1, 4.3, 4.5, 4.6]
return_rate = [0.08, 0.07, 0.06, 0.05]  # 8%, 7%, 6%, 5%
```

Tasks:

1. **Create 4-panel vertical stack (15 points):**
   - Figure size: (12, 14)
   - Four subplots in one column (4 rows, 1 column)
   - Share X-axis across all subplots
   - Panel 1: Units Sold (line plot, markers, color: #2E86AB)
   - Panel 2: Average Price (line plot, markers, color: #D9A72E)
   - Panel 3: Customer Satisfaction (bar plot, color: #06A77D)
   - Panel 4: Return Rate (bar plot, color: #E74C3C)
2. **Professional formatting (10 points):**
   - Each panel title: “{Metric Name}” (left-aligned, size 13, bold)
   - Y-labels appropriate for each metric
   - X-label only on bottom plot: “Quarter”
   - Format satisfaction as “4.5” (one decimal)
   - Format return rate as percentage: “8%”, “7%”, etc.
   - Grid on all plots
3. **Add business context (5 points):**
   - On satisfaction plot: Add horizontal red dashed line at 4.0 (target)
   - On return rate plot: Add horizontal red dashed line at 0.07 (industry average)
   - Include legends for reference lines
   - Overall title: “Flagship Product Performance Dashboard - 2024”

Part C: Executive KPI Dashboard (35 points)

Context: The CEO needs a high-level dashboard with the most important KPIs visible at a glance, with one primary metric emphasized.

Data:

```
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov',
↪  'Dec']
monthly_revenue = [2100000, 2250000, 2400000, 2350000, 2500000, 2650000,
                   2600000, 2750000, 2850000, 2800000, 3100000, 3200000]
monthly_profit = [420000, 450000, 480000, 470000, 500000, 530000,
                  520000, 550000, 570000, 560000, 620000, 640000]
customer_count = [45000, 47000, 49000, 51000, 54000, 57000,
                  60000, 63000, 66000, 69000, 73000, 78000]
employee_count = [450, 455, 460, 465, 470, 475,
                  480, 485, 490, 495, 500, 505]
```

Tasks:

1. **Create custom layout using GridSpec (15 points):**
   - Figure size: (16, 10)
   - Use GridSpec to create:
     - Top section: One large plot spanning full width (2 rows)

        - Bottom section: Three equal smaller plots (1 row, 3 columns)
    - Main plot: Revenue trend (line, thick, prominent)
    - Bottom-left: Profit trend (line)
    - Bottom-center: Customer growth (line or bar)
    - Bottom-right: Employee headcount (bar)

2. **Emphasize hierarchy (12 points):**
    - Main plot styling:
        - Larger title (size 15, bold)
        - Thicker line (width 4)
        - Larger markers (size 12)
        - Color: #1F4788 (professional blue)
        - More prominent grid
    - Supporting plots styling:
        - Smaller titles (size 11, bold)
        - Standard line width (2.5)
        - Consistent color scheme
        - Lighter grids

3. **Professional finish (8 points):**
    - Format main plot Y-axis as millions: “$2.1M”, “$3.2M”, etc.
    - Format all numbers appropriately
    - Overall figure title: “RetailMax Corporation - 2024 Annual Performance”
        - Size 17, bold, positioned properly
    - Use subplots_adjust or tight_layout for optimal spacing
    - Save as: `M2L03_YourName_ExecutiveDashboard.png`

Bonus Challenge (+20 points):

Create an Interactive Comparison Dashboard:

Use the following data for 3 products:

```python
quarters = ['Q1 2023', 'Q2 2023', 'Q3 2023', 'Q4 2023',
            'Q1 2024', 'Q2 2024', 'Q3 2024', 'Q4 2024']
product_a_sales = [450, 480, 510, 540, 580, 620, 650, 680]
product_b_sales = [380, 390, 410, 430, 460, 490, 520, 550]
product_c_sales = [290, 310, 330, 360, 390, 420, 450, 480]
```

Requirements:

1. Create a 3x3 grid (9 subplots)
2. Top row: Sales trends for each product (3 separate plots)
3. Middle row: Year-over-year growth rates for each product
    - Calculate: (2024 value - 2023 value) / 2023 value
4. Bottom row: Market share comparison (stacked or grouped bars)
5. Use shared axes where appropriate
6. Apply professional color scheme (colorblind-friendly)
7. Include comprehensive labels and legends
8. Add overall insights as text annotations

Deliverables:

1. Python file: `M2L03_YourName_Dashboards.py`
2. PNG files:
    - `M2L03_YourName_RegionalDashboard.png`
    - `M2L03_YourName_ProductDashboard.png`
    - `M2L03_YourName_ExecutiveDashboard.png`
    - `M2L03_YourName_Bonus.png` (if bonus attempted)

Grading Rubric:

- Part A (Regional Dashboard): 35 points

- Part B (Product Dashboard): 30 points
- Part C (Executive Dashboard): 35 points
- Bonus: +20 points

Success Criteria:

☐ All subplots display correctly
☐ Shared axes used appropriately for comparisons
☐ Visual hierarchy established (main vs. supporting plots)
☐ Consistent formatting within each dashboard
☐ Professional appearance suitable for executive presentation
☐ All numbers formatted appropriately
☐ Proper spacing (no overlapping elements)
☐ High-resolution outputs
☐ Code well-organized with comments

Design Best Practices:

1. **Consistency**: Use same colors, fonts, styles within each dashboard
2. **Hierarchy**: Emphasize most important information
3. **Comparison**: Shared axes when comparing similar metrics
4. **Context**: Reference lines, targets, benchmarks where relevant
5. **Clarity**: Each subplot should be self-explanatory
6. **Professionalism**: Clean, polished, presentation-ready

Common Pitfalls to Avoid:

- Inconsistent colors across related subplots
- Missing labels on any subplot
- Overlapping titles or labels
- Different scales making comparison difficult
- Too much information crowded together
- Inconsistent formatting across panels
- Poor use of space (too cramped or too spread out)

2.4 Section 4: Business Charts (Bar, Line, and Pie Charts)

2.4.1 Objective

- Master bar charts for categorical comparisons
- Create effective line charts for temporal analysis
- Understand when (and when not) to use pie charts
- Apply horizontal vs. vertical bar chart layouts appropriately
- Use grouped and stacked bars for complex comparisons
- Follow visualization best practices for business communication

2.4.2 Main Contents with Examples

Bar Charts - The Foundation of Categorical Comparison

Why Bar Charts Dominate Business:

Bar charts are the most common visualization in business because they excel at answering the fundamental question: **"How do these categories compare?"**

Perfect for:

- Sales by region
- Revenue by product line
- Performance by salesperson
- Budget by department
- Customer count by segment

The Human Visual System:

- We're excellent at comparing bar lengths
- Horizontal alignment makes comparison natural
- Clear, unambiguous representation
- Works well in black-and-white (important for printing)

Basic Vertical Bar Chart:

```
import matplotlib.pyplot as plt

# Sales by product category
categories = ['Electronics', 'Clothing', 'Home Goods', 'Sports', 'Books']
sales = [2850000, 1920000, 1650000, 1280000, 980000]

fig, ax = plt.subplots(figsize=(10, 6))

ax.bar(categories, sales, color='#2E86AB', edgecolor='black', linewidth=1.5)

ax.set_title('Sales by Product Category - Q4 2024', fontsize=14, fontweight='bold')
ax.set_xlabel('Product Category', fontsize=12)
ax.set_ylabel('Sales Revenue ($)', fontsize=12)

# Format Y-axis as millions
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x/1e6:.1f}M'))

ax.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()
```

Key Bar Chart Elements:

- **Width**: Bars should be consistent width
- **Spacing**: Slight gap between bars aids readability
- **Baseline**: Always starts at zero (critical for honest comparison)
- **Edge color**: Defines bar boundaries clearly
- **Grid**: Horizontal grid lines help reading values

Horizontal Bar Charts - When to Switch Orientation

Use Horizontal Bars When:

1. **Long category names**: "North American Technology Division" won't fit under vertical bar
2. **Many categories**: >8 categories become crowded vertically
3. **Emphasis on ranking**: Horizontal layout emphasizes order
4. **Reading flow**: Top-to-bottom feels natural for rankings

Creating Horizontal Bars:

```
import matplotlib.pyplot as plt

# Top 10 salespeople (long names)
salespeople = ['Jennifer Martinez-Rodriguez', 'Michael Chen',
               'Sarah O\'Connor', 'David Kim', 'Emily Johnson',
               'Robert Thompson', 'Maria Garcia', 'James Wilson',
               'Linda Davis', 'William Brown']
sales = [685000, 672000, 658000, 645000, 639000,
         628000, 615000, 608000, 595000, 587000]

fig, ax = plt.subplots(figsize=(10, 8))

# Use barh() for horizontal
```

```python
ax.barh(salespeople, sales, color='#06A77D', edgecolor='black', linewidth=1.5)

ax.set_title('Top 10 Salespeople - Annual Performance 2024',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Annual Sales ($)', fontsize=12)
ax.set_ylabel('Salesperson', fontsize=12)

# Format X-axis as thousands with K suffix
ax.xaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x/1000:.0f}K'))

ax.grid(True, alpha=0.3, axis='x')

plt.tight_layout()
plt.show()
```

Horizontal Bar Advantages:

- Category labels are always readable (no rotation needed)
- Natural top-to-bottom ranking
- Better for data stories emphasizing order

Grouped Bar Charts - Comparing Multiple Series

The Business Need:

Compare the same categories across different groups:

- This year vs. last year by quarter
- Actual vs. budget by department
- Male vs. female customers by age group
- Product sales across different regions

Creating Grouped Bars:

```python
import matplotlib.pyplot as plt
import numpy as np

# Quarterly revenue: 2023 vs 2024
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
revenue_2023 = [2100000, 2250000, 2350000, 2500000]
revenue_2024 = [2400000, 2650000, 2750000, 3100000]

fig, ax = plt.subplots(figsize=(12, 6))

# Position bars side-by-side
x = np.arange(len(quarters))  # [0, 1, 2, 3]
width = 0.35  # Width of each bar

# Create grouped bars
bars1 = ax.bar(x - width/2, revenue_2023, width, label='2023',
               color='#7FB3D5', edgecolor='black', linewidth=1.5)
bars2 = ax.bar(x + width/2, revenue_2024, width, label='2024',
               color='#148F77', edgecolor='black', linewidth=1.5)

# Set x-axis labels
ax.set_xticks(x)
ax.set_xticklabels(quarters)

ax.set_title('Year-over-Year Revenue Comparison', fontsize=14, fontweight='bold')
ax.set_xlabel('Quarter', fontsize=12)
ax.set_ylabel('Revenue', fontsize=12)
```

```
# Format Y-axis
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x/1e6:.1f}M'))

ax.legend(title='Fiscal Year', fontsize=11)
ax.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()
```

Grouped Bar Formula:

- Calculate positions: `x = np.arange(len(categories))`
- Offset bars: `x - width/2` for first group, `x + width/2` for second
- For three groups: `x - width`, `x`, `x + width`

When to Use Grouped Bars:

- Comparing 2-4 groups
- Each category has multiple values
- Differences within categories matter more than totals

Stacked Bar Charts - Showing Part-to-Whole

The Composition Question:

Stacked bars answer: "What makes up this total?"

- Revenue by product line per quarter (total quarterly revenue)
- Budget allocation by department per year (total yearly budget)
- Employee distribution by level per division (total employees)

Creating Stacked Bars:

```
import matplotlib.pyplot as plt
import numpy as np

# Quarterly revenue by product line
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
electronics = [1200000, 1350000, 1400000, 1650000]
clothing = [650000, 720000, 750000, 850000]
home_goods = [450000, 580000, 600000, 600000]

fig, ax = plt.subplots(figsize=(12, 6))

# Stack bars on top of each other
ax.bar(quarters, electronics, label='Electronics',
       color='#2E86AB', edgecolor='black', linewidth=1.5)
ax.bar(quarters, clothing, bottom=electronics, label='Clothing',
       color='#06A77D', edgecolor='black', linewidth=1.5)
ax.bar(quarters, home_goods,
       bottom=np.array(electronics) + np.array(clothing),
       label='Home Goods', color='#D9A72E', edgecolor='black', linewidth=1.5)

ax.set_title('Quarterly Revenue by Product Line - 2024',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Quarter', fontsize=12)
ax.set_ylabel('Revenue', fontsize=12)

# Format Y-axis
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x/1e6:.1f}M'))
ax.legend(loc='upper left', fontsize=11)
ax.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()
```

Stacked Bar Logic:

- First series: Normal bar chart
- Second series: Use `bottom=first_series` to stack on top
- Third series: `bottom=first_series + second_series`

When to Use Stacked Bars:

- Total matters AND components matter
- Showing composition over time
- Limited categories (3-5 stacks maximum)
- Colors are distinctive

Warning: Stacked bars make it hard to compare middle sections. Only the bottom section and total are easy to compare.

Line Charts - Temporal Trends and Continuous Data

The Time Series Specialist:

Line charts are the default for showing change over time because:

- Lines imply continuity between points
- Slopes show rate of change
- Trends are immediately visible
- Multiple series can be compared easily

When Line Charts Excel:

- Monthly sales trends
- Stock prices over time
- Temperature readings
- Website traffic patterns
- Any sequential data

Professional Line Chart:

```
import matplotlib.pyplot as plt

# Monthly website traffic
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
          'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
visitors = [245000, 268000, 252000, 289000, 312000, 335000,
            358000, 342000, 375000, 398000, 425000, 448000]

fig, ax = plt.subplots(figsize=(12, 6))

ax.plot(months, visitors, marker='o', linewidth=2.5, markersize=8,
        color='#2E86AB')

# Highlight peak month
max_idx = visitors.index(max(visitors))
ax.plot(months[max_idx], visitors[max_idx], marker='*',
        markersize=20, color='#E74C3C', zorder=3)

ax.set_title('Website Traffic - 2024 Monthly Performance',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Month', fontsize=12)
ax.set_ylabel('Monthly Visitors', fontsize=12)

# Format Y-axis with K suffix
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'{x/1000:.0f}K'))

# Annotate peak
ax.annotate(f'Peak: {max(visitors):,} visitors',
            xy=(months[max_idx], visitors[max_idx]),
            xytext=(10, -30), textcoords='offset points',
            fontsize=10, color='#E74C3C',
            bbox=dict(boxstyle='round,pad=0.5', facecolor='yellow', alpha=0.7),
            arrowprops=dict(arrowstyle='->', color='#E74C3C', lw=2))

ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

Advanced Line Chart Features:

- **Markers**: Emphasize data points
- **Annotations**: Highlight important events
- **Reference lines**: Show targets or benchmarks
- **Shading**: Highlight specific periods
- **Multiple lines**: Compare trends

Pie Charts - Use Sparingly and Wisely

The Controversial Chart:

Pie charts are controversial in data visualization:

- **Proponents**: Intuitive, shows parts of whole clearly
- **Critics**: Humans are bad at comparing angles/areas

When Pie Charts Work:

- **Few categories** (3-5 maximum, ideally 2-3)
- **Clear majority**: One category dominates (>50%)
- **Simple message**: "X makes up most of our Y"
- **Non-technical audience**: Familiar and accessible

When to Avoid Pie Charts:

- Many categories (use bar chart)
- Similar-sized categories (angles too hard to compare)
- Comparing multiple pies (very difficult)
- Precise comparisons needed (bar chart better)

Creating an Effective Pie Chart:

```
import matplotlib.pyplot as plt

# Market share (simple, clear story)
companies = ['Our Company', 'Competitor A', 'Competitor B', 'Others']
market_share = [35, 28, 22, 15]

fig, ax = plt.subplots(figsize=(10, 8))

# Create pie chart
colors = ['#2E86AB', '#7FB3D5', '#A3C9E2', '#CCCCCC']
explode = (0.1, 0, 0, 0)  # Emphasize our company
```

```python
wedges, texts, autotexts = ax.pie(market_share,
                                    labels=companies,
                                    autopct='%1.1f%%',
                                    startangle=90,
                                    colors=colors,
                                    explode=explode,
                                    shadow=True)

# Format text
for text in texts:
    text.set_fontsize(12)
    text.set_fontweight('bold')

for autotext in autotexts:
    autotext.set_color('white')
    autotext.set_fontsize(11)
    autotext.set_fontweight('bold')

ax.set_title('Smartphone Market Share - Q4 2024',
             fontsize=14, fontweight='bold', pad=20)

plt.tight_layout()
plt.show()
```

Pie Chart Best Practices:

1. **Order**: Largest to smallest (clockwise from 12 o'clock)
2. **Explode**: Pull out most important slice
3. **Labels**: Direct labels better than legend when possible
4. **Percentages**: Show actual numbers
5. **Colors**: Use meaningful colors (blue for us, gray for others)

Better Alternatives to Pie Charts:

For most situations, consider:

- **Horizontal bar chart**: Easier comparison of categories
- **Donut chart**: Same as pie, but central hole can hold title/total
- **Treemap**: For hierarchical data
- **Stacked percentage bar**: Shows composition more clearly

Choosing the Right Chart Type

Decision Framework:

Question: How do categories compare? → **Bar Chart** (vertical or horizontal)

Question: How has this changed over time? → **Line Chart**

Question: What is the composition of this whole? → **Stacked Bar Chart** or **Pie Chart** (if 2-3 categories)

Question: How do multiple groups compare across categories? → **Grouped Bar Chart**

Question: What is the relationship between two variables? → **Scatter Plot** (covered in Section 1)

Chart Selection Table:

Data Type	Best Chart	When to Use	Avoid If
Categories vs Values	Vertical Bar	Few categories, short names	>12 categories
Categories vs Values	Horizontal Bar	Long names, many categories, rankings	Categories unordered

Data Type	Best Chart	When to Use	Avoid If
Time Series	Line Chart	Continuous data, trends important	Few time points
Comparison Over Time	Grouped Bar	2-4 groups, emphasis on periods	Many groups
Part-to-Whole	Stacked Bar	Composition changes, total matters	>5 components
Part-to-Whole	Pie Chart	2-4 categories, simple message	Precise comparison needed

2.4.3 Lab Session

Lab 4: Comprehensive Business Charts

Objective: Master bar charts, line charts, and pie charts by creating a complete analytical report for a business scenario.

Scenario: You're the Data Analyst at "MetroMart Retail," a regional supermarket chain. The Annual Strategy Meeting is approaching, and leadership needs comprehensive visualizations analyzing 2024 performance across products, time, regions, and categories. You'll create a professional analytical report with multiple chart types.

Pre-Lab Setup:

1. Create file: `M2L04_YourName_BusinessCharts.py`
2. Import necessary libraries
3. Create output folder: "Lab4_Outputs"

Part A: Product Category Performance (25 points)

Context: The Merchandising VP needs to see which product categories drove the most revenue.

Data:

```
categories = ['Fresh Produce', 'Dairy & Eggs', 'Meat & Seafood',
              'Bakery', 'Frozen Foods', 'Beverages', 'Snacks', 'Health & Beauty']
annual_revenue = [4850000, 3920000, 3650000, 2180000, 2950000,
                  3120000, 2680000, 1850000]
```

Tasks:

1. **Create professional vertical bar chart (15 points):**
   - Figure size: (12, 7)
   - Bars with color: #2E86AB
   - Black edge color, line width 1.5
   - Sort bars from highest to lowest revenue
   - Add horizontal grid lines (alpha=0.3)
2. **Format professionally (10 points):**
   - Title: "Annual Revenue by Product Category - 2024"
   - Y-axis label: "Revenue"
   - X-axis label: "Product Category"
   - Format Y-axis as millions: "$4.9M", "$3.9M", etc.
   - Rotate X-axis labels 45 degrees, right-aligned
   - Save as: `M2L04_YourName_CategoryRevenue.png`

Part B: Monthly Sales Trend Analysis (25 points)

Context: The Finance team needs to see monthly revenue patterns to identify seasonality and trends.

Data:

```
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
          'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
monthly_revenue = [2180000, 2050000, 2320000, 2280000, 2450000, 2390000,
                   2520000, 2480000, 2350000, 2580000, 2850000, 3150000]
```

Tasks:

1. **Create trend line chart (12 points):**
   - Figure size: (14, 6)
   - Line color: #148F77 (dark green)
   - Line width: 3
   - Circular markers, size 10
   - Add semi-transparent markers (alpha=0.6)
2. **Add analytical features (8 points):**
   - Highlight December (peak month) with red star marker (size 20)
   - Add annotation pointing to December with text: "Holiday Peak: $3.15M"
   - Add horizontal reference line at $2.5M (dashed, red, label "Target")
   - Include legend for reference line
3. **Professional formatting (5 points):**
   - Title: "MetroMart - 2024 Monthly Revenue Trend"
   - Format Y-axis as millions
   - Grid with alpha=0.3
   - Save as: `M2L04_YourName_MonthlyTrend.png`

Part C: Regional Comparison (25 points)

Context: Operations needs to compare performance across store regions.

Data:

```
regions = ['Downtown Metro', 'Suburban North', 'Suburban South',
           'Suburban East', 'Suburban West', 'Urban Center']
q1_sales = [2850000, 3120000, 2980000, 2750000, 3050000, 2890000]
q2_sales = [3020000, 3280000, 3150000, 2920000, 3180000, 3050000]
q3_sales = [2950000, 3180000, 3080000, 2850000, 3120000, 2980000]
q4_sales = [3350000, 3580000, 3420000, 3180000, 3450000, 3280000]
```

Tasks:

1. **Create horizontal bar chart for total annual sales (15 points):**
   - Calculate total sales for each region (sum of 4 quarters)
   - Use horizontal bars (barh)
   - Sort regions by total sales (highest at top)
   - Figure size: (12, 7)
   - Professional color scheme
   - Edge color black, width 1.5
2. **Add data labels (5 points):**
   - Show exact revenue on each bar
   - Format as: "$12.8M"
   - Position at end of bar
   - Font size: 11, bold
3. **Format and save (5 points):**
   - Title: "Annual Sales by Region - 2024"
   - X-axis label: "Total Annual Sales"
   - Format X-axis as millions
   - Grid on X-axis

- Save as: `M2L04_YourName_RegionalSales.png`

Part D: Quarterly Performance Grouped Bars (25 points)

Context: Strategic planning needs quarterly performance comparison for top 4 product categories.

Data:

```
quarters = ['Q1', 'Q2', 'Q3', 'Q4']
produce = [1150000, 1220000, 1180000, 1300000]
dairy = [980000, 1020000, 950000, 970000]
meat = [890000, 920000, 910000, 930000]
frozen = [720000, 750000, 730000, 750000]
```

Tasks:

1. **Create grouped bar chart (15 points):**
   - Figure size: (14, 7)
   - Four groups of bars (one per quarter)
   - Four bars per group (one per category)
   - Use colorblind-friendly palette:
     - Produce: #648FFF (blue)
     - Dairy: #FFB000 (orange)
     - Meat: #DC267F (pink)
     - Frozen: #785EF0 (purple)
   - Each bar: edge color black, line width 1
2. **Professional formatting (10 points):**
   - Title: "Quarterly Performance by Product Category - 2024"
   - X-axis: Quarter labels
   - Y-axis: "Revenue"
   - Format Y-axis as millions
   - Legend with title "Product Category"
   - Grid on Y-axis
   - Save as: `M2L04_YourName_QuarterlyGrouped.png`

Part E: Market Composition Pie Chart (Optional - 20 points)

Context: The executive summary needs a simple visual showing revenue composition by major category groups.

Data:

```
category_groups = ['Perishables', 'Packaged Goods', 'Non-Food']
revenue_by_group = [14900000, 8750000, 1850000]
```

Tasks:

1. **Create effective pie chart (15 points):**
   - Figure size: (10, 8)
   - Explode the largest category (0.1)
   - Use professional color scheme (avoid garish colors)
   - Show percentages with one decimal place
   - Start angle: 90 degrees
   - Add shadow for depth
2. **Professional presentation (5 points):**
   - Title: "2024 Revenue Composition by Category Group"
   - Bold labels, size 12
   - White percentage text, bold, size 11
   - Save as: `M2L04_YourName_Composition.png`

Bonus Challenge (+25 points):

Create a Comprehensive Dashboard Combining Multiple Chart Types:

Using all the data provided above, create ONE figure that combines:

1. A main trend line (monthly revenue) - largest subplot
2. A supporting bar chart (top 5 categories) - medium subplot
3. A pie chart (category groups) - small subplot

Requirements:

- Use GridSpec for custom layout
- Professional styling throughout
- Consistent color scheme
- Clear hierarchy (main chart prominent)
- Overall dashboard title
- Save as: `M2L04_YourName_Dashboard.png`

Deliverables:

1. Python file: `M2L04_YourName_BusinessCharts.py`
2. PNG files:
   - `M2L04_YourName_CategoryRevenue.png`
   - `M2L04_YourName_MonthlyTrend.png`
   - `M2L04_YourName_RegionalSales.png`
   - `M2L04_YourName_QuarterlyGrouped.png`
   - `M2L04_YourName_Composition.png` (if Part E attempted)
   - `M2L04_YourName_Dashboard.png` (if bonus attempted)

Grading Rubric:

- Part A (Category Bar Chart): 25 points
- Part B (Monthly Trend Line): 25 points
- Part C (Regional Horizontal Bars): 25 points
- Part D (Grouped Bars): 25 points
- Part E (Pie Chart): +20 points
- Bonus (Dashboard): +25 points

Success Criteria:

☐ Appropriate chart type for each scenario
☐ All axes labeled with units
☐ Numbers formatted professionally
☐ Colors chosen intentionally
☐ Titles descriptive and specific
☐ High resolution output (dpi=300)
☐ Sorted data where appropriate
☐ No overlapping text
☐ Professional appearance suitable for executive presentation

Chart Selection Checklist:

Before creating each chart, ask: - [ ] Is this categorical comparison? → Bar chart - [ ] Is this showing change over time? → Line chart - [ ] Do I need to compare groups? → Grouped bars - [ ] Am I showing composition? → Stacked bars or pie - [ ] Are there many categories? → Horizontal bars - [ ] Do labels fit? → Rotate or use horizontal layout

Professional Touches:

- Sort bars by value (unless order is meaningful)
- Use consistent colors across related charts
- Add reference lines to show targets
- Annotate important points
- Format large numbers with M/K suffixes
- Include appropriate decimal places

- Add light grids for readability

End of Module 2: Core Visualization with Matplotlib

Key Takeaways:

- Matplotlib provides complete control over visualization appearance
- Professional customization transforms basic plots into boardroom-ready charts
- Multiple subplots enable comprehensive dashboards and comparisons
- Choosing the right chart type is critical for effective communication
- Bar, line, and pie charts each serve specific business communication needs

Chapter 3

Module 3: Statistical Visualization with Seaborn

3.1 Section 1: Introduction to Seaborn and Statistical Plots

3.1.1 Objective

- Understand Seaborn's role as a high-level statistical visualization library
- Learn the relationship between Seaborn and Matplotlib
- Master Seaborn's styling and theme system for professional aesthetics
- Create fundamental statistical plots: histograms, KDE plots, and box plots
- Interpret statistical visualizations for business insights
- Apply Seaborn's figure-level vs. axes-level plotting functions

3.1.2 Main Contents with Examples

What is Seaborn and Why Use It?

The Evolution Beyond Matplotlib:

While Matplotlib gives you complete control, it requires significant code for common statistical visualizations. Seaborn was created to solve this problem by providing:

- **Statistical focus**: Built-in statistical calculations and visualizations
- **Beautiful defaults**: Professional appearance with minimal customization
- **Less code**: Complex plots in single commands
- **Integrated statistics**: Automatic confidence intervals, regression lines, distributions
- **Pandas integration**: Works seamlessly with DataFrames

A Revealing Comparison:

Consider creating a visualization showing the relationship between two variables with a regression line:

Matplotlib approach: 30+ lines of code

- Calculate regression manually
- Plot scatter points
- Calculate and plot regression line
- Calculate and plot confidence intervals
- Customize colors, labels, styles

Seaborn approach: 1 line of code

```
sns.regplot(data=df, x='advertising_spend', y='revenue')
```

Seaborn handles all statistical calculations and styling automatically.

When to Use Each Library:

Use Matplotlib when:

- You need pixel-perfect custom control
- Creating novel visualization types
- Building interactive visualizations
- Working on specialized scientific plots

Use Seaborn when:

- Exploring data statistically
- Creating standard business analytics plots
- Need beautiful visualizations quickly
- Working with categorical and continuous data together
- Presenting to non-technical stakeholders

Important Understanding: Seaborn is built on top of Matplotlib, not a replacement. You'll often use both together - Seaborn for the plot structure, Matplotlib for fine-tuning.

Installing and Importing Seaborn

Installation:

Seaborn should already be installed from Module 1, but if needed:

```
pip install seaborn
```

Standard Import Convention:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

The alias `sns` is universally used (named after Samuel Norman Seaborn, a character from "The West Wing" TV show).

Version Check:

```
import seaborn as sns
print(sns.__version__)  # Should be 0.12 or higher
```

Seaborn's Theme and Style System

The Aesthetic Advantage:

One of Seaborn's most appreciated features is its built-in aesthetic system. With one line, you can transform your visualizations from basic to professional.

Setting Overall Style:

Seaborn provides five preset themes:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Available styles:
sns.set_style("darkgrid")   # Dark background with grid (default)
sns.set_style("whitegrid")  # White background with grid
sns.set_style("dark")       # Dark background, no grid
sns.set_style("white")      # White background, no grid
sns.set_style("ticks")      # White background with tick marks
```

Style Characteristics:

- **darkgrid**: Best for presentations, easy to read values

- **whitegrid**: Professional reports, clean appearance
- **white**: Minimal, modern, emphasizes data
- **ticks**: Traditional academic/scientific look
- **dark**: Bold, modern, good for screens

Understanding Context:

Seaborn also provides context settings that scale all elements appropriately for different uses:

```
# Contexts scale font sizes and element sizes
sns.set_context("paper")     # Smallest, for journal papers
sns.set_context("notebook")  # Default, good for screen viewing
sns.set_context("talk")      # Larger, for presentations
sns.set_context("poster")    # Largest, for posters/large displays
```

Business Context Usage:

- **Notebook**: Daily analysis, sharing via email
- **Talk**: PowerPoint presentations, board meetings
- **Poster**: Trade shows, lobby displays, large screens

Color Palettes:

Seaborn provides sophisticated color palettes designed by professionals:

```
# Set color palette
sns.set_palette("deep")      # Deep, saturated colors (default)
sns.set_palette("muted")     # Softer, muted colors (professional)
sns.set_palette("bright")    # Bright, vibrant colors
sns.set_palette("pastel")    # Pastel colors (subtle)
sns.set_palette("dark")      # Dark colors
sns.set_palette("colorblind") # Colorblind-safe palette (recommended!)
```

Professional Setup for Business:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Configure for business presentations
sns.set_theme(style="whitegrid",      # Clean, professional
              context="talk",          # Large enough for presentations
              palette="colorblind")    # Accessible to all
```

This single setup ensures all subsequent Seaborn plots have a professional, accessible appearance.

Understanding Seaborn's Two Interfaces

A Critical Concept:

Seaborn has two types of functions, and understanding the difference is essential:

1. **Figure-level functions (newer, recommended):**

- Create entire figures with built-in structure
- Names end with "plot": `relplot()`, `displot()`, `catplot()`
- Return FacetGrid objects
- Better for complex, multi-panel visualizations
- Automatic legend and label management

2. **Axes-level functions (older, more control):**

- Create plots on existing axes
- Direct names: `histplot()`, `boxplot()`, `scatterplot()`
- Return matplotlib axes objects
- Better for integration with existing figures
- More matplotlib-like control

Practical Example:

```
# Axes-level: You control the figure
fig, ax = plt.subplots(figsize=(10, 6))
sns.histplot(data=df, x='sales', ax=ax)
ax.set_title('Sales Distribution')
plt.show()

# Figure-level: Seaborn controls the figure
sns.displot(data=df, x='sales', height=6, aspect=1.67)
plt.show()
```

Which to Use?

- **Learning**: Start with axes-level (more familiar if you know Matplotlib)
- **Quick exploration**: Figure-level (less code)
- **Custom layouts**: Axes-level (more control)
- **Multi-panel**: Figure-level (automatic faceting)

Histograms - Understanding Data Distribution

Why Histograms Matter in Business:

Before making decisions based on averages, you must understand the distribution:

- Are most customers spending around the average, or are there two distinct groups?
- Is revenue normally distributed, or heavily skewed by a few big deals?
- Are there outliers affecting our analysis?

Histograms answer these questions visually by showing:

- **Shape**: Normal, skewed, bimodal?
- **Center**: Where is the typical value?
- **Spread**: How variable is the data?
- **Outliers**: Any extreme values?

Creating a Histogram with Seaborn:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Sample data: Customer purchase amounts
data = {
    'purchase_amount': [45, 52, 48, 61, 58, 63, 55, 72, 68, 51,
                        88, 45, 59, 67, 73, 49, 56, 78, 82, 65,
                        92, 54, 61, 75, 69, 57, 64, 71, 85, 58]
}
df = pd.DataFrame(data)

# Create histogram
fig, ax = plt.subplots(figsize=(10, 6))
sns.histplot(data=df, x='purchase_amount', bins=10,
             kde=False, color='#2E86AB', ax=ax)

ax.set_title('Distribution of Customer Purchase Amounts',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Purchase Amount ($)', fontsize=12)
ax.set_ylabel('Number of Customers', fontsize=12)

plt.tight_layout()
plt.show()
```

Understanding Histogram Parameters:

- **bins**: Number of bars (groups). Too few = oversimplified, too many = noisy
  - Rule of thumb: √n (square root of number of observations)
  - For 30 observations: ~5-6 bins; for 100: ~10 bins; for 1000: ~30 bins
- **kde**: Kernel Density Estimate - smooth curve showing distribution
  - `kde=True` adds a smooth line over the bars
  - Helps visualize the overall shape
- **color**: Single color for the bars

Reading the Story:

Looking at a histogram, ask: 1. **Shape**: Is it symmetric (normal) or skewed (one tail longer)? 2. **Peaks**: One peak (unimodal) or multiple peaks (bimodal)? 3. **Outliers**: Any bars far from the main cluster? 4. **Range**: What's the spread from minimum to maximum?

Business Interpretation Example:

A histogram of customer purchase amounts showing:

- **Right-skewed**: Most customers spend $40-60, but a few big spenders reach $150+
- **Business insight**: Can't rely on average alone; need strategies for both typical customers and high-value customers

KDE Plots - Smooth Distribution Curves

Beyond Histogram Bars:

Kernel Density Estimation (KDE) creates a smooth curve representing the data distribution. Think of it as a smoothed histogram that's easier to compare across groups.

Advantages of KDE:

- Smooth, professional appearance
- Easier to overlay multiple distributions
- Shows shape clearly without bar edge artifacts
- Better for presentations (less "technical" looking)

Creating a KDE Plot:

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Revenue data for two product lines
product_a_revenue = [45, 52, 48, 61, 58, 63, 55, 72, 68, 51, 59, 67, 73]
product_b_revenue = [38, 42, 45, 39, 51, 48, 43, 55, 52, 46, 49, 58, 61]

df = pd.DataFrame({
    'revenue': product_a_revenue + product_b_revenue,
    'product': ['A']*len(product_a_revenue) + ['B']*len(product_b_revenue)
})

fig, ax = plt.subplots(figsize=(10, 6))

# KDE plots for both products
sns.kdeplot(data=df, x='revenue', hue='product',
            fill=True, alpha=0.5, linewidth=2, ax=ax)

ax.set_title('Revenue Distribution Comparison: Product A vs B',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Revenue ($1000s)', fontsize=12)
ax.set_ylabel('Density', fontsize=12)

plt.tight_layout()
plt.show()
```

KDE Parameters Explained:

- **hue**: Variable to separate by color (creates multiple curves)
- **fill**: Fill the area under the curve (True/False)
- **alpha**: Transparency (important when overlapping)
- **linewidth**: Thickness of the outline

Interpreting KDE Plots:

The Y-axis shows “density” - not counts, but relative concentration:

- **Higher peaks**: More observations in this range
- **Wider spread**: More variability
- **Multiple peaks**: Distinct subgroups

Business Use Case:

Comparing customer lifetime value distributions across regions:

- West Coast: Narrow, high peak around $500 = consistent customers
- East Coast: Wide, low peak = highly variable customer value
- **Insight**: West Coast needs retention strategy, East Coast needs segmentation

Box Plots - The Statistical Summary

The Five-Number Summary:

Box plots (also called box-and-whisker plots) compress a distribution into five key statistics:

1. **Minimum**: Lowest value (excluding outliers)
2. **Q1 (First Quartile)**: 25% of data is below this
3. **Median (Q2)**: Middle value (50% below, 50% above)
4. **Q3 (Third Quartile)**: 75% of data is below this
5. **Maximum**: Highest value (excluding outliers)
6. **Outliers**: Points beyond whiskers (shown as individual dots)

Why Box Plots Matter:

In business, you often need to compare distributions across categories:

- Sales performance across regions
- Salaries across departments
- Product ratings across age groups

Box plots excel at this by showing multiple distributions side-by-side compactly.

Anatomy of a Box Plot:

```
Outlier •

--------  Maximum (within 1.5×IQR of Q3)
    |


         Q3 (75th percentile)
 ---     Median (50th percentile)  ← Line inside box
         Q1 (25th percentile)


    |
--------  Minimum (within 1.5×IQR of Q1)

Outlier •
```

Creating Box Plots:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Salesperson performance across regions
```

```
data = {
    'region': ['North']*15 + ['South']*15 + ['East']*15 + ['West']*15,
    'sales': [
        # North: generally higher, consistent
        145, 152, 148, 161, 158, 163, 155, 172, 168, 151, 159, 167, 173, 149, 156,
        # South: lower, more variable
        112, 128, 115, 135, 122, 138, 125, 148, 132, 119, 127, 141, 152, 118, 129,
        # East: highest performance
        168, 175, 172, 181, 178, 183, 175, 192, 188, 171, 179, 187, 193, 169, 176,
        # West: mixed, with outliers
        141, 145, 98, 162, 148, 155, 143, 178, 152, 146, 149, 165, 172, 142, 151
    ]
}
df = pd.DataFrame(data)

fig, ax = plt.subplots(figsize=(10, 6))

sns.boxplot(data=df, x='region', y='sales',
            palette='Set2', ax=ax)

ax.set_title('Sales Performance by Region - Q4 2024',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Region', fontsize=12)
ax.set_ylabel('Sales ($1000s)', fontsize=12)

plt.tight_layout()
plt.show()
```

Reading Box Plots for Business Insights:

1. **Compare Medians (center lines):**
   - East region has highest median → top performer overall
2. **Compare Box Heights (IQR - interquartile range):**
   - North has small box → consistent performance
   - South has tall box → high variability (some stars, some struggles)
3. **Identify Outliers (dots beyond whiskers):**
   - West has low outlier → investigate underperforming salesperson
   - Outliers may indicate data errors or special cases
4. **Compare Whisker Ranges:**
   - Overall spread shows best-case and worst-case scenarios

Business Application:

A CFO looking at departmental budget variance box plots:

- **Finance department**: Small box, no outliers = predictable, well-controlled
- **Marketing department**: Large box, several outliers = needs budget management attention
- **Action**: Implement stricter controls in Marketing, use Finance as best practice model

3.1.3 Lab Session

Lab 1: Statistical Visualization Fundamentals

Objective: Master Seaborn's foundational capabilities by creating professional statistical visualizations that reveal data distributions and enable comparisons.

Scenario: You're the Analytics Manager at "HealthyLife Fitness," a gym chain with 500+ members. The executive team is reviewing Q4 performance and needs statistical visualizations to understand member behavior, revenue patterns, and facility usage. Your task is to create a comprehensive statistical analysis report.

Pre-Lab Setup:

1. Create file: `S1234567_YourName_StatPlots.py`
2. Import libraries:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

3. Set Seaborn theme at the top of your file:

```
sns.set_theme(style="whitegrid", context="talk", palette="colorblind")
```

4. Create output folder: "Lab1_Outputs"

Part A: Member Age Distribution Analysis (25 points)

Context: The marketing team wants to understand the age demographics of your membership to target advertising effectively.

Data:

```
# Generate realistic age distribution (you'll learn to load real data later)
np.random.seed(42)
member_ages = np.concatenate([
    np.random.normal(28, 5, 150),   # Young professionals
    np.random.normal(45, 8, 200),   # Middle-aged
    np.random.normal(65, 6, 100),   # Seniors
    np.random.uniform(18, 75, 50)   # Random others
])
member_ages = member_ages[(member_ages >= 18) & (member_ages <= 80)]

df_ages = pd.DataFrame({'age': member_ages})
```

Tasks:

1. **Create a histogram with KDE overlay (15 points):**
   - Figure size: (12, 6)
   - Use `sns.histplot()` with `kde=True`
   - Use 20 bins
   - Color: professional choice (avoid default blue)
   - Add edge color to bars (black, linewidth=1)
   - Title: "HealthyLife Fitness - Member Age Distribution"
   - X-axis label: "Age (years)"
   - Y-axis label: "Number of Members"
2. **Interpret the distribution (10 points):**
   - In code comments, answer:
     - How many distinct age groups (peaks) do you see?
     - What's the approximate age range with most members?
     - Are there any gaps or unusual patterns?
     - What does this mean for marketing strategy?
   - Add a vertical line at the median age using: `ax.axvline(df_ages['age'].median(), color='red', linestyle='--', linewidth=2, label='Median Age')`
   - Include legend
3. **Save the visualization:**
   - Filename: `S1234567_YourName_AgeDistribution.png`
   - High resolution (dpi=300)

Part B: Revenue Distribution Comparison (30 points)

Context: The finance director needs to compare monthly revenue distributions across different membership tiers (Basic, Premium, Elite) to assess pricing strategy effectiveness.

Data:

```
np.random.seed(42)
basic_revenue = np.random.normal(450, 75, 180)      # Basic: $450 avg
premium_revenue = np.random.normal(850, 120, 150)   # Premium: $850 avg
elite_revenue = np.random.normal(1500, 200, 70)     # Elite: $1500 avg

df_revenue = pd.DataFrame({
    'monthly_revenue': np.concatenate([basic_revenue, premium_revenue,
     ↪  elite_revenue]),
    'tier': ['Basic']*180 + ['Premium']*150 + ['Elite']*70
})
```

Tasks:

1. **Create overlapping KDE plots (15 points):**
   - Figure size: (12, 7)
   - Use `sns.kdeplot()` with:
     - `x='monthly_revenue'`
     - `hue='tier'`
     - `fill=True`
     - `alpha=0.5`
     - `linewidth=2.5`
   - Title: "Monthly Revenue Distribution by Membership Tier"
   - X-axis label: "Monthly Revenue per Member ($)"
   - Y-axis label: "Density"
   - Ensure legend is visible and well-positioned
2. **Add reference lines (10 points):**
   - Calculate and add vertical lines for mean revenue of each tier
   - Use different line styles for each tier
   - Add text labels showing the exact mean values
   - Hint: Use `df_revenue.groupby('tier')['monthly_revenue'].mean()`
3. **Business interpretation (5 points):**
   - In comments, answer:
     - Which tier has the most consistent (least variable) revenue?
     - Is there any overlap between tiers?
     - What percentage of Basic members might be good candidates for Premium upgrade?
     - Save as: `S1234567_YourName_RevenueTiers.png`

Part C: Facility Usage Box Plot Comparison (25 points)

Context: Operations needs to compare average weekly gym visits across different times of day and days of week to optimize staffing.

Data:

```
np.random.seed(42)

# Weekly visits by time of day
morning_visits = np.random.poisson(12, 100)     # Morning: avg 12/week
afternoon_visits = np.random.poisson(8, 100)    # Afternoon: avg 8/week
evening_visits = np.random.poisson(15, 100)     # Evening: avg 15/week (busiest)
night_visits = np.random.poisson(5, 100)        # Night: avg 5/week

df_visits = pd.DataFrame({
    'weekly_visits': np.concatenate([morning_visits, afternoon_visits,
                                     evening_visits, night_visits]),
    'time_period': ['Morning (6-10am)']*100 + ['Afternoon (10am-4pm)']*100 +
                   ['Evening (4-8pm)']*100 + ['Night (8pm-12am)']*100
})
```

Tasks:

1. **Create professional box plot (15 points):**
   - Figure size: (12, 7)
   - Use `sns.boxplot()` with:
     - `x='time_period'`
     - `y='weekly_visits'`
     - Choose professional color palette
   - Title: "Member Weekly Visits by Time Period"
   - X-axis label: "Time Period"
   - Y-axis label: "Average Weekly Visits"
   - Rotate X-axis labels if needed for readability
2. **Enhance with statistical information (10 points):**
   - Add horizontal line showing overall median visits across all periods
   - Use distinct color (e.g., red) with dashed style
   - Add label: "Overall Median"
   - Make line width 2
   - Calculate and print (in comments):
     - Which period has highest median?
     - Which period has most variability (tallest box)?
     - Identify any outliers and their values
3. **Save and document:**
   - Save as: `S1234567_YourName_UsageByTime.png`
   - In comments, provide staffing recommendations based on the visualization

Part D: Multi-Facility Comparison (20 points)

Context: The VP of Operations oversees 5 locations and needs to compare member satisfaction scores across facilities to identify underperforming locations.

Data:

```
np.random.seed(42)

facilities = ['Downtown', 'Suburban North', 'Suburban South', 'University District',
 ↪  'Business Park']
all_scores = []
all_facilities = []

for facility in facilities:
    if facility == 'University District':
        scores = np.random.normal(4.8, 0.3, 80)  # Highest satisfaction
    elif facility == 'Business Park':
        scores = np.random.normal(3.9, 0.7, 60)  # Lowest, most variable
    else:
        scores = np.random.normal(4.3, 0.5, 70)  # Average

    scores = np.clip(scores, 1, 5)  # Keep between 1-5
    all_scores.extend(scores)
    all_facilities.extend([facility] * len(scores))

df_satisfaction = pd.DataFrame({
    'facility': all_facilities,
    'satisfaction_score': all_scores
})
```

Tasks:

1. **Create comparison box plot (12 points):**
    - Figure size: (14, 7)
    - Use horizontal box plot (`orient='h'` or swap x/y)
    - Sort facilities by median satisfaction (highest to lowest)
    - Use color gradient (best = green, worst = red)
    - Title: "Member Satisfaction by Facility Location - Q4 2024"
    - Include axis labels with units
2. **Add performance benchmarks (8 points):**
    - Add vertical line at satisfaction score of 4.5 (corporate target)
    - Add vertical line at satisfaction score of 4.0 (minimum acceptable)
    - Use different colors and label each line
    - Shade the area above 4.5 in light green (excellent zone)
    - Hint: Use `ax.axvspan()` for shading
3. **Save and analyze:**
    - Save as: `S1234567_YourName_FacilitySatisfaction.png`
    - In comments, identify which facilities need immediate attention

Bonus Challenge (+20 points):

Create a Comprehensive Statistical Dashboard:

Combine all four analyses into a single 2×2 subplot figure:

- Top-left: Age distribution histogram
- Top-right: Revenue KDE comparisons
- Bottom-left: Usage box plots
- Bottom-right: Facility satisfaction box plots

Additional Requirements:

- Figure size: (20, 16)
- Consistent styling across all subplots
- Overall figure title: "HealthyLife Fitness - Q4 2024 Statistical Analysis Dashboard"
- Proper spacing (no overlapping elements)
- Each subplot should be readable and professional
- Save as: `S1234567_YourName_Dashboard.png`

Deliverables:

1. Python file: `S1234567_YourName_StatPlots.py`
2. Four PNG files (five if bonus attempted):
    - `S1234567_YourName_AgeDistribution.png`
    - `S1234567_YourName_RevenueTiers.png`
    - `S1234567_YourName_UsageByTime.png`
    - `S1234567_YourName_FacilitySatisfaction.png`
    - `S1234567_YourName_Dashboard.png` (bonus)

Grading Rubric:

- Part A (Age Distribution): 25 points
- Part B (Revenue Comparison): 30 points
- Part C (Usage Box Plots): 25 points
- Part D (Facility Comparison): 20 points
- Bonus (Dashboard): +20 points

Success Criteria:

☐ Seaborn theme configured at top of file
☐ All visualizations use appropriate Seaborn functions
☐ Professional styling and colors applied
☐ Statistical features (median lines, reference lines) included
☐ Business interpretations documented in comments
☐ All plots have clear titles and axis labels
☐ High-resolution outputs saved
☐ Code is well-organized with clear sections

Statistical Interpretation Guide:

When analyzing distributions, always assess: 1. **Central Tendency**: Where is the typical value? (mean/median) 2. **Spread**: How variable is the data? (range, IQR) 3. **Shape**: Symmetric? Skewed? Multiple peaks? 4. **Outliers**: Any unusual values? Why? 5. **Comparisons**: How do groups differ?

Common Mistakes to Avoid:

- Using too many or too few bins in histograms
- Forgetting to set transparency when overlaying KDE plots
- Not explaining outliers in box plots
- Missing labels or units on axes
- Poor color choices (too bright, not colorblind-safe)
- Not leveraging Seaborn's theme system

Tips for Success:

- Start by setting the Seaborn theme globally
- Test each plot individually before combining
- Use meaningful color palettes (not random colors)
- Always include units in axis labels
- Comment your interpretations as you code
- View your saved images to verify quality

3.2 Section 2: Distribution and Relationship Visualizations

3.2.1 Objective

- Create and interpret joint plots showing relationships between two variables
- Master pair plots for multivariate exploratory analysis
- Use regression plots to visualize linear relationships
- Apply residual plots to assess model fit quality
- Understand the statistical interpretation of relationship visualizations
- Leverage these tools for business predictive analytics

3.2.2 Main Contents with Examples

Beyond Single Variables - Exploring Relationships

The Business Reality:

Most business questions involve relationships between variables:

- Does advertising spend correlate with revenue?
- Do experienced employees have higher satisfaction scores?
- Is there a relationship between product price and sales volume?
- Do customer age and purchase frequency correlate?

Single-variable distributions (histograms, box plots) don't answer these questions. We need **bivariate visualizations** that show two variables simultaneously.

Types of Relationships to Explore:

1. Positive Correlation:

- As X increases, Y increases
- Example: Years of experience → Salary
- Business implication: Investing in X may improve Y

2. Negative Correlation:

- As X increases, Y decreases
- Example: Product price → Sales volume
- Business implication: Trade-off between X and Y

3. No Correlation:

- X and Y move independently
- Example: Employee shoe size → Sales performance
- Business implication: X won't help predict Y

4. Non-linear Relationship:

- Relationship exists but isn't a straight line
- Example: Advertising spend → Revenue (diminishing returns)
- Business implication: Complex relationship requires different models

Scatter Plots with Regression Lines

The Foundation of Relationship Analysis:

We introduced scatter plots in Module 2, but Seaborn enhances them significantly by adding statistical elements automatically.

Seaborn's regplot() - Scatter + Regression:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Sample data: Marketing spend vs Revenue
np.random.seed(42)
marketing_spend = np.linspace(10000, 100000, 50)
revenue = 50000 + 2.5 * marketing_spend + np.random.normal(0, 30000, 50)

df = pd.DataFrame({
    'marketing_spend': marketing_spend,
    'revenue': revenue
})

fig, ax = plt.subplots(figsize=(10, 6))

# Create scatter plot with regression line
sns.regplot(data=df, x='marketing_spend', y='revenue',
            scatter_kws={'alpha': 0.6, 's': 80},
            line_kws={'color': 'red', 'linewidth': 2},
            ax=ax)

ax.set_title('Marketing Spend vs Revenue Analysis',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Marketing Spend ($)', fontsize=12)
ax.set_ylabel('Revenue ($)', fontsize=12)

# Format axes
ax.xaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x/1000:.0f}K'))
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x/1000:.0f}K'))
```

```
plt.tight_layout()
plt.show()
```

What regplot() Provides Automatically:

1. **Scatter points**: Shows actual data points
2. **Regression line**: Best-fit straight line through the data
3. **Confidence interval**: Shaded area around the line (95% confidence by default)

Understanding the Confidence Interval:

The shaded area represents uncertainty in our estimate:

- **Narrow band**: High confidence, strong relationship, lots of data
- **Wide band**: Less confidence, weaker relationship, or less data
- **Interpretation**: "We're 95% confident the true relationship falls within this band"

Business Application:

Looking at the marketing spend plot:

- **Positive slope**: More marketing spend → More revenue (good!)
- **Confidence interval**: Gets wider at extremes (less data at very high/low spend)
- **Scatter around line**: Some variation, but overall strong pattern
- **Decision support**: Provides evidence that marketing investment correlates with revenue growth

Joint Plots - Combining Multiple Perspectives

The Multi-View Approach:

Joint plots are one of Seaborn's most powerful features, combining three visualizations in one figure:

- **Center**: Scatter plot showing relationship
- **Top**: Histogram of X variable
- **Right**: Histogram of Y variable

This provides comprehensive understanding:

- See the relationship (center)
- See X distribution (top)
- See Y distribution (right)

Creating a Joint Plot:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Employee data: Years of experience vs Salary
np.random.seed(42)
experience = np.random.uniform(0, 20, 100)
salary = 40000 + 3500 * experience + np.random.normal(0, 8000, 100)

df = pd.DataFrame({
    'years_experience': experience,
    'salary': salary
})

# Create joint plot
g = sns.jointplot(data=df, x='years_experience', y='salary',
                  kind='reg',  # Includes regression line
                  height=8,
                  color='#2E86AB')
```

```
# Customize
g.fig.suptitle('Employee Compensation Analysis: Experience vs Salary',
               fontsize=14, fontweight='bold', y=1.02)
g.set_axis_labels('Years of Experience', 'Annual Salary ($)', fontsize=12)

plt.tight_layout()
plt.show()
```

Joint Plot "kind" Options:

Seaborn offers several relationship types:

1. **kind='scatter'** (default)
   - Basic scatter plot
   - Best for initial exploration
2. **kind='reg'**
   - Adds regression line
   - Best when you suspect linear relationship
3. **kind='kde'**
   - Contour plot showing density
   - Best for large datasets where points overlap
   - Shows concentration of data
4. **kind='hex'**
   - Hexagonal binning
   - Best for very large datasets (1000+ points)
   - Shows density through color

Reading a Joint Plot:

Top histogram:

- Shows distribution of experience
- Check: Are employees evenly distributed across experience levels?

Right histogram:

- Shows distribution of salaries
- Check: Is salary distribution normal or skewed?

Center scatter:

- Shows relationship
- Check: Linear? Non-linear? Outliers?

Business Insights Example:

From employee compensation joint plot:

- **Center**: Clear positive trend (experience → higher salary)
- **Top**: Most employees have 5-15 years (middle career)
- **Right**: Salary slightly right-skewed (few very high earners)
- **Outlier**: One employee with 15 years making only $50K (investigate!)

Pair Plots - Multivariate Exploration

The "Look at Everything" Tool:

In business, you often have many variables and don't know which relationships matter. Pair plots solve this by creating a matrix of visualizations showing all possible pairs.

Example Scenario:

You have data on:

- Customer age
- Income
- Years as customer
- Average purchase amount
- Satisfaction score

Which variables correlate? Pair plots show all relationships at once.

Creating a Pair Plot:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Sample customer data
np.random.seed(42)
n = 100

df_customers = pd.DataFrame({
    'age': np.random.normal(40, 15, n),
    'income': np.random.normal(65000, 25000, n),
    'years_customer': np.random.uniform(0, 10, n),
    'avg_purchase': np.random.normal(150, 50, n)
})

# Ensure reasonable values
df_customers['age'] = df_customers['age'].clip(18, 80)
df_customers['income'] = df_customers['income'].clip(20000, 200000)
df_customers['avg_purchase'] = df_customers['avg_purchase'].clip(20, 500)

# Create pair plot
g = sns.pairplot(df_customers,
                 diag_kind='kde',  # Distribution on diagonal
                 plot_kws={'alpha': 0.6},
                 height=2.5)

g.fig.suptitle('Customer Metrics - Pairwise Relationship Analysis',
               fontsize=14, fontweight='bold', y=1.01)

plt.tight_layout()
plt.show()
```

Understanding the Pair Plot Grid:

Diagonal (top-left to bottom-right):

- Shows distribution of each variable (KDE or histogram)
- Check: Is each variable normally distributed? Skewed?

Off-diagonal:

- Each cell is a scatter plot
- Row variable on Y-axis, column variable on X-axis
- Example: Row 2, Column 1 = Income (Y) vs Age (X)

Reading Strategy:

1. **Scan diagonals**: Understand each variable's distribution
2. **Look for patterns**: Strong linear relationships show clear diagonal patterns
3. **Identify weak relationships**: Random clouds indicate no correlation
4. **Spot outliers**: Points far from main clusters

Business Application - Customer Segmentation:

From the pair plot, you might discover:

- **Age vs Income**: Positive correlation (older → higher income)
- **Income vs Avg Purchase**: Strong positive (wealth → spending)
- **Years Customer vs Age**: No clear pattern (customers of all ages)
- **Action**: Segment by income/spending, not by tenure

Adding Categories with Hue:

```
# Add customer segment
df_customers['segment'] = pd.cut(df_customers['income'],
                                  bins=[0, 50000, 100000, 300000],
                                  labels=['Budget', 'Standard', 'Premium'])

# Pair plot with segments colored
g = sns.pairplot(df_customers,
                 hue='segment',  # Color by segment
                 palette='Set2',
                 diag_kind='kde',
                 height=2.5)

plt.tight_layout()
plt.show()
```

Now each scatter plot shows three colors (Budget, Standard, Premium), revealing:

- Do segments have different age distributions?
- Do purchase behaviors differ by segment?
- Are segments well-separated or overlapping?

Residual Plots - Assessing Model Quality

Beyond the Line - How Good is the Fit?

A regression line looks nice, but how well does it actually fit the data? Residual plots answer this question.

What are Residuals?

Residuals are the differences between actual values and predicted values:

- **Residual = Actual Y - Predicted Y**
- Positive residual: Point is above the line
- Negative residual: Point is below the line

Why Residuals Matter:

A good linear regression should have residuals that: 1. **Randomly scattered**: No pattern around zero 2. **Constant spread**: Same variance throughout 3. **Normally distributed**: Roughly bell-shaped

If residuals show patterns, the linear model may be inappropriate.

Creating a Residual Plot:

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Create data with different patterns
np.random.seed(42)

# Good linear fit
x_good = np.linspace(0, 10, 100)
y_good = 2 * x_good + 5 + np.random.normal(0, 2, 100)

# Non-linear pattern (quadratic)
```

```
x_bad = np.linspace(0, 10, 100)
y_bad = 2 * x_bad**2 + np.random.normal(0, 5, 100)

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Good fit residual plot
sns.residplot(x=x_good, y=y_good,
              lowess=True,
              scatter_kws={'alpha': 0.5},
              line_kws={'color': 'red', 'linewidth': 2},
              ax=axes[0])
axes[0].set_title('Good Linear Fit\n(Random Scatter)',
                  fontsize=12, fontweight='bold')
axes[0].set_xlabel('X Variable')
axes[0].set_ylabel('Residuals')
axes[0].axhline(0, color='black', linestyle='--', linewidth=1)

# Poor fit residual plot
sns.residplot(x=x_bad, y=y_bad,
              lowess=True,
              scatter_kws={'alpha': 0.5},
              line_kws={'color': 'red', 'linewidth': 2},
              ax=axes[1])
axes[1].set_title('Poor Linear Fit\n(Curved Pattern)',
                  fontsize=12, fontweight='bold')
axes[1].set_xlabel('X Variable')
axes[1].set_ylabel('Residuals')
axes[1].axhline(0, color='black', linestyle='--', linewidth=1)

plt.tight_layout()
plt.show()
```

Interpreting Residual Plots:

Good Pattern (Left):

- Points scattered randomly around zero
- No curved pattern
- Constant spread (homoscedasticity)
- **Conclusion**: Linear model is appropriate

Bad Pattern (Right):

- Clear curved pattern (U-shape or inverted U)
- **Conclusion**: Relationship is non-linear, need different model

Other Warning Signs:

Fan shape (heteroscedasticity):

- Spread increases as X increases
- **Issue**: Predictions less reliable at high X values
- **Solution**: Transform data (log, square root)

Outliers:

- Points very far from zero
- **Action**: Investigate these cases (errors? special circumstances?)

Business Example:

Scenario: Modeling sales based on advertising spend

Residual plot shows curve:

- Spending \$10K → Over-prediction (residuals negative)
- Spending \$50K → Good prediction (residuals near zero)
- Spending \$100K → Under-prediction (residuals positive)

Interpretation: Relationship is non-linear (diminishing returns at high spend)

Action: Use polynomial or log model instead of linear

Practical Workflow for Relationship Analysis

Step-by-Step Business Analysis Process:

When exploring relationships in business data:

Step 1: Pair Plot Overview

- Load your data
- Create pair plot of all numeric variables
- Identify interesting relationships

Step 2: Deep Dive with Joint Plots

- For each interesting relationship, create joint plot
- Add regression line (`kind='reg'`)
- Examine strength and direction

Step 3: Verify with Residual Plot

- For relationships you want to model
- Create residual plot
- Check if linear model is appropriate

Step 4: Business Interpretation

- Translate statistical findings to business language
- Quantify relationships (correlation coefficients)
- Make recommendations

Complete Example Workflow:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Sample sales data
np.random.seed(42)
df_sales = pd.DataFrame({
    'ad_spend': np.random.uniform(5000, 50000, 100),
    'sales_calls': np.random.poisson(30, 100),
    'revenue': np.random.normal(100000, 30000, 100),
    'customer_satisfaction': np.random.uniform(3.5, 5.0, 100)
})

# Add realistic relationships
df_sales['revenue'] = (50000 +
                       1.2 * df_sales['ad_spend'] +
                       500 * df_sales['sales_calls'] +
                       np.random.normal(0, 10000, 100))

# Step 1: Pair plot overview
sns.pairplot(df_sales)
plt.suptitle('Sales Data - Initial Exploration', y=1.01)
plt.show()

# Step 2: Deep dive - Ad Spend vs Revenue
```

```python
g = sns.jointplot(data=df_sales, x='ad_spend', y='revenue',
                  kind='reg', height=8)
g.fig.suptitle('Deep Dive: Advertising Impact on Revenue', y=1.02)
plt.show()

# Step 3: Check residuals
fig, ax = plt.subplots(figsize=(10, 6))
sns.residplot(data=df_sales, x='ad_spend', y='revenue',
              lowess=True, ax=ax)
ax.set_title('Residual Analysis: Ad Spend Model')
ax.axhline(0, color='black', linestyle='--')
plt.show()

# Step 4: Quantify (calculate correlation)
correlation = df_sales[['ad_spend', 'revenue']].corr().iloc[0, 1]
print(f"Correlation between Ad Spend and Revenue: {correlation:.3f}")
```

Statistical Significance and Business Decisions

Understanding Correlation Strength:

Correlation coefficients range from -1 to +1:

- **0.9 to 1.0**: Very strong positive (rare in business)
- **0.7 to 0.9**: Strong positive
- **0.4 to 0.7**: Moderate positive
- **0.1 to 0.4**: Weak positive
- **0.0**: No linear relationship
- **Negative values**: Same scale, but inverse relationship

Business Decision Framework:

Correlation = 0.85 (Strong):

- High confidence in relationship
- Can use for predictive models
- Justified to invest resources based on this

Correlation = 0.35 (Weak):

- Relationship exists but variable
- Multiple factors influence outcome
- Need additional variables for good predictions
- Be cautious with major decisions

Remember: Correlation Causation

Just because two variables correlate doesn't mean one causes the other:

- Ice cream sales and drowning deaths correlate (both peak in summer)
- Company revenue and CEO's golf score might correlate (both improve with company growth)

Always ask: Could a third variable explain both?

3.2.3 Lab Session

Lab 2: Relationship and Correlation Analysis

Objective: Master bivariate analysis techniques to uncover, visualize, and interpret relationships between business variables for predictive insights.

Scenario: You're the Business Intelligence Analyst at "TechStart Solutions," a B2B software company with 200+ customers. The executive team is planning 2025 strategy and needs data-driven insights about what drives customer success, revenue growth, and satisfaction. Your mission is to analyze relationships between key business metrics to guide strategic decisions.

Pre-Lab Setup:

1. Create file: `M3L02_YourName_Relationships.py`
2. Import libraries and set theme:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

sns.set_theme(style="whitegrid", context="talk", palette="colorblind")
```

3. Create output folder: “Lab2_Outputs”

Part A: Customer Success Metrics Exploration (30 points)

Context: Your Customer Success VP wants to understand which factors correlate with customer satisfaction to prioritize improvement efforts.

Data:

```python
np.random.seed(42)
n_customers = 200

# Generate realistic customer data
df_customers = pd.DataFrame({
    'customer_id': range(1, n_customers + 1),
    'contract_value': np.random.uniform(5000, 100000, n_customers),
    'support_tickets': np.random.poisson(8, n_customers),
    'response_time_hours': np.random.exponential(12, n_customers),
    'features_used': np.random.randint(3, 25, n_customers),
    'satisfaction_score': np.random.uniform(2.5, 5.0, n_customers)
})

# Create realistic relationships
df_customers['satisfaction_score'] = (
    3.0 +
    0.00001 * df_customers['contract_value'] +  # Higher value → slightly higher
     ↪  satisfaction
    -0.08 * df_customers['support_tickets'] +     # More tickets → lower satisfaction
    -0.03 * df_customers['response_time_hours'] + # Slower response → lower
     ↪  satisfaction
    0.04 * df_customers['features_used'] +        # More features → higher
     ↪  satisfaction
    np.random.normal(0, 0.3, n_customers)
)

# Clip to valid range
df_customers['satisfaction_score'] = df_customers['satisfaction_score'].clip(1.0, 5.0)
```

Tasks:

1. **Create comprehensive pair plot (15 points):**
   - Include all numeric variables except customer_id
   - Use KDE on diagonal
   - Set appropriate figure height (around 3)
   - Set plot transparency (alpha=0.6)
   - Add overall title: “Customer Success Metrics - Pairwise Correlation Analysis”
   - Ensure the title doesn’t overlap with plots
2. **Identify key relationships (10 points):**
   - Calculate correlation matrix for all numeric variables

- Print the correlation matrix
- In code comments, identify:
  - Which variable has the STRONGEST correlation with satisfaction?
  - Which variable has the WEAKEST correlation with satisfaction?
  - Are there any unexpected correlations between other variables?
  - Which metrics should the Customer Success team focus on?

3. **Save and document (5 points):**

   - Save the pair plot as: `M3L02_YourName_CustomerPairplot.png`
   - Create a text file or detailed comments with your findings
   - Include specific correlation values in your analysis

Part B: Deep Dive - Support Response Time Impact (30 points)

Context: The support team is debating whether faster response times truly impact satisfaction. They need statistical evidence to justify hiring more support staff.

Tasks:

1. **Create joint plot analysis (15 points):**

   - Create a joint plot of response_time_hours (x) vs satisfaction_score (y)
   - Use `kind='reg'` to include regression line
   - Set height=9 for better visibility
   - Choose professional color (not default)
   - Title: "Support Response Time Impact on Customer Satisfaction"
   - Properly label axes with units

2. **Analyze the relationship (10 points):**

   - Look at the regression line slope
   - Examine the confidence interval (shaded area)
   - Check the histograms on top and right
   - In comments, answer:
     - Is there a positive or negative correlation?
     - How strong is the relationship?
     - What does the scatter around the line tell us?
     - If we reduce response time by 10 hours, what's the expected satisfaction impact?
   - Add these insights as comments in your code

3. **Create residual plot (5 points):**

   - Create a residual plot for response_time vs satisfaction
   - Add lowess line (smooth curve)
   - Add horizontal line at y=0
   - Title: "Residual Analysis: Response Time Model"
   - Check if the linear model is appropriate
   - Save both plots:
     - `M3L02_YourName_ResponseTimeJoint.png`
     - `M3L02_YourName_ResponseResiduals.png`

Part C: Revenue Drivers Analysis (25 points)

Context: The Sales and Marketing teams are arguing about what drives revenue. Marketing says features used (product engagement) matters most. Sales says it's all about contract value and relationship (support interactions). Settle the debate with data.

Data preparation:

```
# Add revenue data with realistic relationships
df_customers['annual_revenue'] = (
    df_customers['contract_value'] * 1.2 +  # Base revenue from contract
    500 * df_customers['features_used'] +    # Upsell from feature usage
    -1000 * df_customers['support_tickets'] + # Cost of support
    np.random.normal(0, 5000, n_customers)
)

df_customers['annual_revenue'] = df_customers['annual_revenue'].clip(0, None)
```

Tasks:

1. **Create three regression plots (18 points):**
   - Create a figure with 1 row, 3 columns (figsize=(18, 6))
   - Plot 1: contract_value vs annual_revenue
   - Plot 2: features_used vs annual_revenue
   - Plot 3: support_tickets vs annual_revenue
   - Each with regression line and confidence interval
   - Consistent styling across all three
   - Professional color choices
   - All with proper titles and labels
2. **Comparative analysis (7 points):**
   - Calculate correlation coefficient for each relationship
   - Print all three correlations
   - In comments, answer:
     - Which variable has the strongest correlation with revenue?
     - Is the support_tickets correlation positive or negative? Why?
     - Who wins the debate - Marketing or Sales? Why?
     - What's your strategic recommendation?
   - Save as: `M3L02_YourName_RevenueDrivers.png`

Part D: Customer Segmentation Insights (15 points)

Context: The Product team wants to understand if customer behavior differs by company size (small, medium, large based on contract value).

Tasks:

1. **Create customer segments (5 points):**

```
# Create size segments
df_customers['company_size'] = pd.cut(df_customers['contract_value'],
                                       bins=[0, 25000, 60000, 150000],
                                       labels=['Small', 'Medium', 'Large'])
```

2. **Create segmented pair plot (10 points):**
   - Create pair plot of: features_used, support_tickets, satisfaction_score
   - Use `hue='company_size'` to color by segment
   - Include appropriate palette
   - Height: 3
   - Title: "Customer Behavior by Company Size"
   - In comments, analyze:
     - Do larger companies use more features?
     - Do larger companies have more support tickets?
     - Does satisfaction vary by company size?
     - What insights can inform product/sales strategy?
   - Save as: `M3L02_YourName_Segmentation.png`

Bonus Challenge (+25 points):

Predictive Model Validation:

Build a comprehensive analysis demonstrating whether you can predict satisfaction from other metrics:

Requirements:

1. **Create a regression plot showing predicted vs actual satisfaction:**

    - Use all available metrics to predict satisfaction
    - Create scatter plot with actual satisfaction on one axis
    - Show perfect prediction line (45-degree line)
    - Calculate R-squared value
    - Annotate R-squared on the plot

2. **Create comprehensive residual analysis:**
    - Residuals vs predicted values
    - Histogram of residuals (should be normal)
    - Q-Q plot if you're adventurous
    - Document whether predictions are reliable

3. **Business recommendations document:**
    - Create detailed comments or separate text file
    - List top 3 factors driving satisfaction
    - Quantify expected impact of improvements
    - Provide specific, actionable recommendations
    - Estimate ROI of each recommendation

4. **Save all bonus deliverables with clear names**

Deliverables:

1. Python file: `M3L02_YourName_Relationships.py`
2. PNG files:
    - `M3L02_YourName_CustomerPairplot.png`
    - `M3L02_YourName_ResponseTimeJoint.png`
    - `M3L02_YourName_ResponseResiduals.png`
    - `M3L02_YourName_RevenueDrivers.png`
    - `M3L02_YourName_Segmentation.png`
    - Bonus files if attempted

Grading Rubric:

- Part A (Pair Plot Exploration): 30 points
- Part B (Response Time Analysis): 30 points
- Part C (Revenue Drivers): 25 points
- Part D (Segmentation): 15 points
- Bonus (Predictive Validation): +25 points

Success Criteria:

☐ All correlation analyses include numeric values
☐ Business interpretations are specific and actionable
☐ Visualizations are professional and properly labeled
☐ Statistical concepts (correlation, residuals) applied correctly
☐ Clear distinction between correlation and causation in comments
☐ All plots follow consistent styling
☐ High-resolution outputs
☐ Code well-organized with clear sections

Statistical Analysis Checklist:

For each relationship analyzed, document:

☐ Direction (positive/negative)
☐ Strength (correlation coefficient)
☐ Statistical significance
☐ Practical significance (business impact)
☐ Potential confounding factors
☐ Actionable recommendations

Common Pitfalls to Avoid:

- Claiming causation from correlation

- Ignoring weak correlations that might matter
- Not checking residual plots before trusting models
- Forgetting to consider business context
- Using inappropriate color schemes
- Not quantifying findings (use actual numbers!)

Professional Tips:

- Always start with pair plots for overview
- Use joint plots for detailed investigation
- Check residuals before trusting regression
- Calculate and report correlation coefficients
- Translate statistical findings to business language
- Consider confounding variables
- Make specific, quantified recommendations

3.3 Section 3: Categorical Data Visualization

3.3.1 Objective

- Master specialized plots for categorical data analysis
- Create effective count plots, bar plots, and point plots
- Use violin plots and swarm plots for distribution comparisons
- Apply strip plots and box plots for categorical comparisons
- Understand when to use each categorical plot type
- Leverage FacetGrid for multi-dimensional categorical analysis

3.3.2 Main Contents with Examples

The Nature of Categorical Data in Business

Understanding Categorical Variables:

In business, many critical variables are categorical (also called qualitative or nominal):

- **Regions**: North, South, East, West
- **Product Categories**: Electronics, Clothing, Home Goods
- **Customer Segments**: Budget, Standard, Premium
- **Employee Departments**: Sales, Marketing, IT, Finance
- **Time Periods**: Q1, Q2, Q3, Q4
- **Status**: Active, Churned, On Hold

Why Categorical Data Requires Different Visualizations:

Unlike continuous variables (revenue, age, temperature), categorical variables:

- Have **discrete groups** with no inherent order (usually)
- Can't be averaged meaningfully (what's the average of "Sales" and "Marketing"?)
- Need **comparison-focused** visualizations
- Often involve **counts or aggregations** within categories

Business Questions Answered by Categorical Visualizations:

1. **How many?** - "How many customers in each segment?"
2. **Which is highest?** - "Which region has highest sales?"
3. **How do distributions differ?** - "Do salaries vary by department?"
4. **Are there patterns?** - "Do customer satisfaction scores differ by product line?"

Count Plots - Visualizing Frequencies

The Most Basic Categorical Question: "How Many?"

Count plots are the categorical equivalent of histograms - they show how many observations fall into each category.

Business Use Cases:

- Customer distribution across segments
- Employee count by department
- Product sales by category
- Support tickets by priority level

Creating a Count Plot:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Sample data: Customer segments
np.random.seed(42)
segments = np.random.choice(['Budget', 'Standard', 'Premium', 'Enterprise'],
                            size=500,
                            p=[0.4, 0.35, 0.20, 0.05])  # Weighted probabilities

df = pd.DataFrame({'segment': segments})

fig, ax = plt.subplots(figsize=(10, 6))

sns.countplot(data=df, x='segment',
              order=['Budget', 'Standard', 'Premium', 'Enterprise'],
              palette='Set2', ax=ax)

ax.set_title('Customer Distribution by Segment',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Customer Segment', fontsize=12)
ax.set_ylabel('Number of Customers', fontsize=12)

# Add count labels on bars
for container in ax.containers:
    ax.bar_label(container, fontsize=11)

plt.tight_layout()
plt.show()
```

Key Parameters Explained:

- **order**: Specify category order (important for logical flow)
  - Without order: Alphabetical (often meaningless)
  - With order: Logical sequence (Budget → Enterprise)
- **palette**: Color scheme
  - 'Set1', 'Set2', 'Set3': Distinct colors
  - 'Blues', 'Greens': Single-hue gradients
  - Custom list: ['#FF6B6B', '#4ECDC4', '#45B7D1']

Adding Percentages:

```
# Calculate and display percentages
total = len(df)
fig, ax = plt.subplots(figsize=(10, 6))

sns.countplot(data=df, x='segment',
              order=['Budget', 'Standard', 'Premium', 'Enterprise'],
              palette='Set2', ax=ax)
```

```python
# Add percentage labels
for container in ax.containers:
    labels = [f'{int(v.get_height())}\n({v.get_height()/total*100:.1f}%)'
              for v in container]
    ax.bar_label(container, labels=labels, fontsize=10)

ax.set_title('Customer Distribution by Segment (with Percentages)',
             fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```

Business Interpretation:

From the count plot:

- **Budget segment**: 200 customers (40%) - largest group
- **Enterprise segment**: 25 customers (5%) - smallest but likely highest revenue
- **Strategic insight**: Marketing budget should reflect distribution, but sales focus on high-value Enterprise

Bar Plots - Showing Aggregated Values

Beyond Counts - Showing Calculated Values:

While count plots show frequencies, bar plots show **aggregated metrics** for each category:

- Average revenue by region
- Total sales by product category
- Median salary by department

The Critical Difference:

- **Count plot**: "How many customers in each region?"
- **Bar plot**: "What's the average revenue per customer in each region?"

Creating a Bar Plot:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Sales data by region
np.random.seed(42)
regions = ['North', 'South', 'East', 'West'] * 50
sales = np.concatenate([
    np.random.normal(85000, 15000, 50),    # North
    np.random.normal(75000, 12000, 50),    # South
    np.random.normal(92000, 18000, 50),    # East
    np.random.normal(80000, 14000, 50)     # West
])

df_sales = pd.DataFrame({
    'region': regions,
    'sales': sales
})

fig, ax = plt.subplots(figsize=(10, 6))

# Bar plot shows MEAN by default
sns.barplot(data=df_sales, x='region', y='sales',
            order=['North', 'South', 'East', 'West'],
            palette='viridis',
            errorbar='sd',  # Standard deviation error bars
            ax=ax)

ax.set_title('Average Sales by Region (with Std Dev)',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Region', fontsize=12)
ax.set_ylabel('Average Sales ($)', fontsize=12)

# Format Y-axis
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x/1000:.0f}K'))

plt.tight_layout()
plt.show()
```

Understanding Error Bars:

The black lines on top of bars represent uncertainty or variability:

- **errorbar='sd'**: Standard deviation (shows data spread)
- **errorbar='se'**: Standard error (shows estimate uncertainty)
- **errorbar=('ci', 95)**: 95% confidence interval (default)
- **errorbar=None**: No error bars

Business Interpretation:

- **East region**: Highest average ($92K) but large error bars = high variability
- **South region**: Lower average ($75K) but small error bars = consistent
- **Decision**: East has potential but needs consistency; South is stable

Changing the Aggregation Function:

```python
# Instead of mean, show median or sum
sns.barplot(data=df_sales, x='region', y='sales',
            estimator='median',  # Use median instead of mean
            errorbar=None)  # Median doesn't have standard CI

# Or total sales per region
sns.barplot(data=df_sales, x='region', y='sales',
            estimator='sum')
```

Violin Plots - Distribution Shapes Within Categories

The Problem with Bar Plots:

Bar plots only show one summary statistic (mean, median). They hide:

- Distribution shape (normal? skewed? bimodal?)
- Outliers
- Full range of data

Violin plots solve this by showing the **entire distribution** for each category.

What Violin Plots Show:

A violin plot is essentially a **rotated, mirrored KDE plot** for each category:

- **Width**: Indicates density (more data at that value)
- **Shape**: Shows distribution characteristics
- Often includes a box plot inside for key statistics

Creating Violin Plots:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Employee satisfaction by department
np.random.seed(42)
departments = ['Sales', 'Marketing', 'Engineering', 'Finance', 'HR'] * 40

satisfaction = []
for dept in departments:
    if dept == 'Engineering':
        satisfaction.extend(np.random.normal(4.5, 0.4, 1))  # High, consistent
    elif dept == 'Sales':
        satisfaction.extend(np.random.normal(3.8, 0.8, 1))  # Lower, variable
    elif dept == 'HR':
        satisfaction.extend(np.random.normal(4.2, 0.3, 1))  # Good, consistent
    else:
        satisfaction.extend(np.random.normal(4.0, 0.5, 1))

df_satisfaction = pd.DataFrame({
    'department': departments,
    'satisfaction': np.clip(satisfaction, 1, 5)  # Keep in 1-5 range
})

fig, ax = plt.subplots(figsize=(12, 6))

sns.violinplot(data=df_satisfaction, x='department', y='satisfaction',
               palette='muted',
               inner='box',  # Show box plot inside
               ax=ax)

ax.set_title('Employee Satisfaction Distribution by Department',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Department', fontsize=12)
ax.set_ylabel('Satisfaction Score (1-5)', fontsize=12)
ax.set_ylim(1, 5)

# Add reference line at target satisfaction
ax.axhline(4.0, color='red', linestyle='--', linewidth=2, alpha=0.7, label='Target:
 ↪  4.0')
ax.legend()

plt.tight_layout()
plt.show()
```

Violin Plot Parameters:

- **inner=‘box’**: Shows box plot inside (median, quartiles)
- **inner=‘quartile’**: Shows quartile lines only
- **inner=‘point’**: Shows individual points
- **inner=None**: Just the violin shape
- **split=True**: When you have a hue variable, creates split violins for comparison

Reading Violin Plots:

Engineering Department (example):

- **Narrow, tall violin**: Most employees clustered around 4.5
- **Little bulge at bottom**: Very few low scores

- **Interpretation**: Consistently high satisfaction

Sales Department (example):

- **Wide, short violin**: Satisfaction scores spread across range
- **Bulges at multiple points**: Multiple subgroups with different satisfaction
- **Interpretation**: Variable experience; investigate further

Business Application:

HR uses violin plots to:

- Identify departments with satisfaction problems
- Spot bimodal distributions (two distinct employee groups)
- Compare not just averages but entire experience ranges
- Target interventions to specific issues

Box Plots for Categorical Comparisons

The Statistical Workhorse:

We introduced box plots in Section 1, but they shine brightest when comparing distributions across categories.

Enhanced Categorical Box Plots:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Product ratings by category
np.random.seed(42)
categories = ['Electronics', 'Clothing', 'Home', 'Sports', 'Books'] * 60

ratings = []
for cat in categories:
    if cat == 'Electronics':
        ratings.extend(np.random.normal(4.3, 0.6, 1))
    elif cat == 'Books':
        ratings.extend(np.random.normal(4.5, 0.4, 1))
    elif cat == 'Clothing':
        ratings.extend(np.random.normal(3.9, 0.8, 1))
    else:
        ratings.extend(np.random.normal(4.1, 0.5, 1))

df_ratings = pd.DataFrame({
    'category': categories,
    'rating': np.clip(ratings, 1, 5)
})

fig, ax = plt.subplots(figsize=(12, 6))

sns.boxplot(data=df_ratings, x='category', y='rating',
            palette='Set2',
            showmeans=True,  # Show mean as well as median
            meanprops={'marker': 'D', 'markerfacecolor': 'red', 'markersize': 8},
            ax=ax)

ax.set_title('Product Ratings by Category (Q4 2024)',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Product Category', fontsize=12)
ax.set_ylabel('Customer Rating (1-5 stars)', fontsize=12)
ax.set_ylim(1, 5)

# Add target line
ax.axhline(4.0, color='green', linestyle='--', linewidth=2, alpha=0.5, label='Target:
 ↪  4.0')
ax.legend()

plt.tight_layout()
plt.show()
```

Box Plot vs Violin Plot - When to Use Each:

Use Box Plots when:

- Precise statistics matter (median, quartiles, outliers)
- Comparing many categories (violin plots get cluttered)
- Audience prefers familiar formats
- Space is limited (box plots are compact)

Use Violin Plots when:

- Distribution shape matters
- Looking for bimodal patterns
- Audience can interpret density plots
- Have fewer categories to compare

Point Plots - Showing Trends Across Categories

Emphasizing Changes and Comparisons:

Point plots are like line charts for categorical data, excellent for showing:

- Changes across ordered categories (quarters, years)
- Comparing multiple groups across categories
- Emphasizing trends in categorical data

Creating Point Plots:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Customer satisfaction over quarters for different tiers
np.random.seed(42)
quarters = ['Q1', 'Q2', 'Q3', 'Q4'] * 3
tiers = ['Basic'] * 4 + ['Premium'] * 4 + ['Enterprise'] * 4

satisfaction = [
    3.8, 3.9, 4.0, 4.1,      # Basic: steady improvement
    4.2, 4.3, 4.4, 4.6,      # Premium: strong improvement
    4.5, 4.6, 4.5, 4.7       # Enterprise: already high, slight fluctuation
]

df_quarterly = pd.DataFrame({
    'quarter': quarters,
    'tier': tiers,
    'satisfaction': satisfaction
})

fig, ax = plt.subplots(figsize=(12, 6))

sns.pointplot(data=df_quarterly, x='quarter', y='satisfaction',
              hue='tier',
              markers=['o', 's', 'D'],
              linestyles=['-', '--', '-.'],
              palette='Set1',
              ax=ax)

ax.set_title('Quarterly Customer Satisfaction Trends by Tier - 2024',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Quarter', fontsize=12)
ax.set_ylabel('Average Satisfaction Score', fontsize=12)
ax.legend(title='Membership Tier', fontsize=11)
ax.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

Point Plot Features:

- **Points**: Show the mean (or other statistic) for each category
- **Lines**: Connect points to emphasize trends
- **Error bars**: Show confidence intervals (95% by default)
- **Multiple series**: Use hue parameter for comparisons

Business Insights from Point Plots:

Looking at the satisfaction trends:

- **All tiers improving**: Good sign of company-wide initiatives working
- **Premium tier**: Steepest growth (Δ0.4 points) - successful retention efforts
- **Enterprise tier**: Already high but plateauing - may need special attention
- **Gap widening**: Higher tiers pulling away - good for premium positioning

Strip and Swarm Plots - Showing Individual Data Points

When You Need to See Every Observation:

Sometimes aggregations hide important details. Strip and swarm plots show **every individual data point** while organizing by category.

Strip Plots (with jitter):

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Sales rep performance (fewer data points, want to see individuals)
np.random.seed(42)
reps = ['Amy', 'Bob', 'Carlos', 'Diana', 'Erik']
performance = []
rep_names = []

for rep in reps:
    n_deals = 8  # Each rep closed 8 deals this quarter
    if rep == 'Diana':
        deals = np.random.normal(25000, 5000, n_deals)  # Top performer
    elif rep == 'Erik':
        deals = np.random.normal(15000, 3000, n_deals)  # Struggling
    else:
        deals = np.random.normal(20000, 4000, n_deals)

    performance.extend(deals)
    rep_names.extend([rep] * n_deals)

df_reps = pd.DataFrame({
    'sales_rep': rep_names,
    'deal_size': performance
})

fig, ax = plt.subplots(figsize=(12, 6))

sns.stripplot(data=df_reps, x='sales_rep', y='deal_size',
              jitter=0.3,  # Add horizontal jitter to prevent overlap
              alpha=0.6,   # Transparency
              size=8,      # Point size
              palette='deep',
              ax=ax)

ax.set_title('Individual Deal Sizes by Sales Representative - Q4 2024',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Sales Representative', fontsize=12)
ax.set_ylabel('Deal Size ($)', fontsize=12)

# Format Y-axis
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x/1000:.0f}K'))

# Add mean line for each rep
for i, rep in enumerate(reps):
    mean_val = df_reps[df_reps['sales_rep'] == rep]['deal_size'].mean()
    ax.hlines(mean_val, i-0.4, i+0.4, colors='red', linewidth=3, alpha=0.7)

plt.tight_layout()
plt.show()
```

Strip Plot Parameters:

- **jitter**: Amount of horizontal random spread (0-0.5 typical)
  - Without jitter: Points stack vertically (hard to see count)
  - With jitter: Points spread horizontally (see all observations)
- **size**: Point size (adjust based on number of points)
- **alpha**: Transparency (important when points overlap)

Swarm Plots (organized arrangement):

Swarm plots are like strip plots but organize points to avoid overlap:

```
fig, ax = plt.subplots(figsize=(12, 6))

sns.swarmplot(data=df_reps, x='sales_rep', y='deal_size',
              palette='deep',
              size=7,
              ax=ax)

ax.set_title('Deal Sizes by Sales Rep (Swarm Plot)',
             fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```

Swarm Plot Characteristics:

- **No overlap**: Each point is visible
- **Density visible**: Width shows concentration
- **Beautiful**: Aesthetically pleasing arrangement

Warning: Swarm plots don't scale well beyond ~500 points per category.

Business Use Cases:

Use strip/swarm plots when:

- Small dataset (< 500 points total)
- Individual observations matter (sales deals, major contracts)
- Looking for outliers or unusual patterns
- Need to identify specific data points
- Transparency about data size important

Combining Multiple Categorical Plots

Layering Plots for Rich Insights:

Seaborn allows combining plot types for comprehensive visualization:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Generate data
np.random.seed(42)
categories = ['Product A', 'Product B', 'Product C'] * 30
values = np.concatenate([
    np.random.normal(75, 15, 30),
    np.random.normal(85, 12, 30),
    np.random.normal(70, 20, 30)
])

df = pd.DataFrame({'product': categories, 'score': values})

fig, ax = plt.subplots(figsize=(12, 6))

# Layer 1: Violin plot for distributions
sns.violinplot(data=df, x='product', y='score',
               inner=None, palette='pastel', alpha=0.5, ax=ax)

# Layer 2: Box plot for statistics
sns.boxplot(data=df, x='product', y='score',
            width=0.3, palette='dark', ax=ax)

# Layer 3: Individual points
sns.stripplot(data=df, x='product', y='score',
              size=4, color='black', alpha=0.3, jitter=0.2, ax=ax)

ax.set_title('Multi-Layer Visualization: Product Performance Scores',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Product Line', fontsize=12)
ax.set_ylabel('Performance Score', fontsize=12)

plt.tight_layout()
plt.show()
```

What This Reveals:

- **Violin**: Overall distribution shape
- **Box**: Key statistics (median, quartiles)
- **Points**: Individual observations and outliers

This comprehensive view helps:

- Identify distribution shapes
- See statistical summaries
- Verify claims with raw data
- Spot unusual patterns

3.3.3 Lab Session

Lab 3: Categorical Data Analysis

Objective: Master categorical visualization techniques to analyze business performance across departments, products, regions, and customer segments.

Scenario: You're the Operations Analyst at "GlobalManufacture Inc.," a mid-size manufacturing company with multiple product lines, 8 regional offices, and 300+ employees. The COO is preparing the annual operations review and needs comprehensive categorical analysis to identify performance patterns, problem areas, and opportunities. Your task is to create professional categorical visualizations that tell clear stories about operational performance.

Pre-Lab Setup:

1. Create file: `M3L03_YourName_Categorical.py`
2. Import and configure:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

sns.set_theme(style="whitegrid", context="talk", palette="colorblind")
```

3. Create output folder: "Lab3_Outputs"

Part A: Product Line Performance Analysis (25 points)

Context: The Product VP needs to understand which product lines are performing well and which need attention.

Data:

```python
np.random.seed(42)

# 5 product lines, 60 sales transactions each
products = ['Industrial Tools', 'Safety Equipment', 'Machinery Parts',
            'Automation Systems', 'Maintenance Supplies'] * 60

revenue = []
for product in products:
    if product == 'Automation Systems':
        revenue.append(np.random.normal(85000, 20000, 1)[0])  # High value, variable
    elif product == 'Industrial Tools':
        revenue.append(np.random.normal(45000, 8000, 1)[0])   # Medium, consistent
    elif product == 'Safety Equipment':
        revenue.append(np.random.normal(25000, 5000, 1)[0])   # Low, consistent
    elif product == 'Machinery Parts':
        revenue.append(np.random.normal(55000, 15000, 1)[0])  # Medium-high, variable
    else:
        revenue.append(np.random.normal(15000, 3000, 1)[0])   # Low volume

df_products = pd.DataFrame({
    'product_line': products,
    'deal_revenue': np.maximum(revenue, 5000)  # Ensure positive values
})
```

Tasks:

1. **Create comprehensive box plot comparison (12 points):**
   - Figure size: (14, 7)
   - Horizontal box plot (swap x and y for better label reading)
   - Sort product lines by median revenue (highest to lowest)
   - Show means with red diamond markers
   - Use professional color palette
   - Title: "Deal Revenue Distribution by Product Line - 2024"
   - Format revenue axis as "$85K", "$45K", etc.
   - Add vertical reference line at company average revenue
2. **Create layered visualization (8 points):**
   - Same data, new figure
   - Combine violin plot (background, pastel, alpha=0.4)
   - With box plot overlay (narrower width=0.4)
   - And strip plot (small dots, alpha=0.3, jitter=0.2)
   - Same title and formatting
   - This shows distribution, statistics, and individual deals simultaneously
3. **Business analysis (5 points):**
   - In code comments, answer:
     - Which product line has highest median revenue?
     - Which has most variability (widest IQR)?
     - Identify any outliers and speculate why
     - Which product lines are above company average?
     - What strategic recommendations would you make?
   - Save both plots:
     - `M3L03_YourName_ProductBoxPlot.png`
     - `M3L03_YourName_ProductLayered.png`

Part B: Regional Performance Trends (30 points)

Context: The Regional Directors meeting is next week. Each director needs to see how their region performed quarterly throughout 2024 compared to other regions.

Data:

```
np.random.seed(42)

regions = ['Northeast', 'Southeast', 'Midwest', 'Southwest',
           'West', 'Northwest', 'Central', 'Florida']
quarters = ['Q1', 'Q2', 'Q3', 'Q4']

data_list = []
for region in regions:
    base_performance = np.random.uniform(70, 90)  # Each region has base performance
    trend = np.random.uniform(-2, 5)  # Each region has a trend

    for q, quarter in enumerate(quarters):
        # Performance with trend and some randomness
        performance = base_performance + (trend * q) + np.random.normal(0, 3)
        performance = np.clip(performance, 50, 100)  # Keep in reasonable range

        data_list.append({
            'region': region,
            'quarter': quarter,
            'performance_score': performance
        })
```

```
df_regional = pd.DataFrame(data_list)
```

Tasks:

1. **Create point plot showing trends (15 points):**
   - Figure size: (14, 8)
   - X-axis: quarter
   - Y-axis: performance_score
   - Hue: region (different line for each region)
   - Use different markers for top 4 regions vs bottom 4
   - Include confidence intervals (error bars)
   - Title: "Regional Performance Trends - 2024 Quarterly"
   - Add horizontal line at 80 (target performance)
   - Legend outside plot area (right side)
2. **Create categorical comparison by quarter (10 points):**
   - Create 1×4 subplots (one for each quarter)
   - Each subplot: bar plot of average performance by region
   - Sort regions by performance (descending) in each quarter
   - Consistent color scheme across subplots
   - Overall title: "Regional Performance by Quarter - 2024"
   - Identify which regions improved most from Q1 to Q4
3. **Analysis and recommendations (5 points):**
   - Calculate quarterly growth rate for each region
   - Identify top 3 improving regions and bottom 3
   - In comments, provide:
     - Which region was most consistent (least variance)?
     - Which region improved most across the year?
     - Which region(s) declined and need intervention?
     - Specific recommendations for next year
   - Save as: `M3L03_YourName_RegionalTrends.png` and `M3L03_YourName_QuarterlyComparison.png`

Part C: Employee Department Analysis (25 points)

Context: HR is conducting an employee engagement survey analysis. They need to understand satisfaction and salary distributions across departments to identify equity issues and engagement problems.

Data:

```
np.random.seed(42)

departments = ['Engineering', 'Sales', 'Marketing', 'Operations',
               'Finance', 'HR', 'Customer Success', 'R&D']
n_per_dept = 40

data_list = []
for dept in departments:
    for _ in range(n_per_dept):
        # Different departments have different characteristics
        if dept == 'Engineering':
            salary = np.random.normal(95000, 15000, 1)[0]
            satisfaction = np.random.normal(4.2, 0.5, 1)[0]
        elif dept == 'Sales':
            salary = np.random.normal(75000, 25000, 1)[0]  # High variance
 ↪  (commission)
            satisfaction = np.random.normal(3.8, 0.9, 1)[0]  # Variable satisfaction
        elif dept == 'R&D':
            salary = np.random.normal(105000, 12000, 1)[0]  # Highest salaries
            satisfaction = np.random.normal(4.5, 0.3, 1)[0]  # Highest satisfaction
        elif dept == 'Customer Success':
            salary = np.random.normal(65000, 10000, 1)[0]
            satisfaction = np.random.normal(3.5, 0.7, 1)[0]  # Lowest satisfaction
        else:
            salary = np.random.normal(80000, 15000, 1)[0]
            satisfaction = np.random.normal(4.0, 0.5, 1)[0]

        data_list.append({
            'department': dept,
            'salary': max(salary, 40000),
            'satisfaction': np.clip(satisfaction, 1, 5)
        })

df_employees = pd.DataFrame(data_list)
```

Tasks:

1. **Create violin plot for salary distribution (12 points):**
    - Figure size: (14, 8)
    - Show salary distribution by department
    - Use split violin if you add a second category (e.g., gender - you'd need to generate this)
    - Include inner box plot (inner='box')
    - Sort departments by median salary
    - Title: "Salary Distribution by Department"
    - Format Y-axis as "$95K", "$75K", etc.
    - Add horizontal lines at 25th, 50th, 75th percentile of overall salary
    - Color code: Use palette that emphasizes high vs low paid departments
2. **Create satisfaction vs salary scatter with categories (8 points):**
    - New figure (12, 7)
    - Scatter plot: salary (x) vs satisfaction (y)
    - Color by department
    - Add size variation based on count
    - Include trend line for each department using sns.lmplot or manual regression
    - Title: "Salary vs Satisfaction by Department"
    - Does higher pay correlate with satisfaction? Does it vary by department?
3. **HR recommendations (5 points):**
    - Calculate key statistics:
        - Department with highest/lowest median salary
        - Department with highest/lowest satisfaction
        - Correlation between salary and satisfaction overall
        - Identify departments with satisfaction issues despite good pay
    - In comments, provide:
        - Departments requiring immediate HR attention
        - Evidence of pay equity across departments
        - Recommendations for improving engagement
    - Save as: `M3L03_YourName_SalaryViolin.png` and `M3L03_YourName_SalarySatisfaction.png`

Part D: Customer Segment Count and Revenue Analysis (20 points)

Context: Marketing wants to understand customer distribution and value across segments for 2025 budget allocation.

Data:

```
np.random.seed(42)

# Generate customer base with realistic distribution
segments_list = (['Small Business'] * 450 +
                ['Mid-Market'] * 250 +
                ['Enterprise'] * 80 +
                ['Startup'] * 220)

industries = ['Manufacturing', 'Technology', 'Healthcare',
              'Retail', 'Finance', 'Education']

data_list = []
for segment in segments_list:
    industry = np.random.choice(industries)

    if segment == 'Enterprise':
        revenue = np.random.lognormal(12, 0.5)  # Very high, right-skewed
    elif segment == 'Mid-Market':
        revenue = np.random.lognormal(10, 0.6)
    elif segment == 'Small Business':
        revenue = np.random.lognormal(8, 0.7)
    else:  # Startup
        revenue = np.random.lognormal(7, 0.9)

    data_list.append({
        'segment': segment,
        'industry': industry,
        'annual_revenue': revenue
    })

df_customers = pd.DataFrame(data_list)
```

Tasks:

1. **Create count plot with percentages (10 points):**
   - Figure size: (12, 6)
   - Count plot of customer segments
   - Order: Enterprise, Mid-Market, Small Business, Startup (value-based)
   - Add count labels on bars with percentages
   - Color gradient: Enterprise (gold) to Startup (blue)
   - Title: "Customer Base Distribution by Segment"
   - Second plot: Stacked bar showing industry mix within each segment
2. **Create bar plot comparing average revenue (10 points):**
   - Figure size: (12, 6)
   - Show average revenue by segment
   - Include error bars (95% CI)
   - Add text annotations showing total revenue contribution: count × average
   - Use log scale if needed to show all segments clearly
   - Title: "Average Annual Revenue by Customer Segment"
   - Calculate: Which segment generates most total revenue despite count?
   - Save as: `M3L03_YourName_CustomerCount.png` and `M3L03_YourName_SegmentRevenue.png`

Bonus Challenge (+25 points):

Create a Comprehensive Categorical Dashboard:

Build a 2×3 grid (6 subplots) showing: 1. Product line revenue distributions (violin plot) 2. Regional performance trends (point plot) 3. Department satisfaction scores (box plot) 4. Customer segment counts (count plot with percentages) 5. Salary distribution by department (horizontal box plot) 6. Industry vs Segment heatmap (count of customers in each combination)

Requirements:

- Figure size: (20, 12)

- Consistent styling across all subplots
- Professional color schemes (use different palettes for different contexts)
- All plots properly labeled
- Overall title: "GlobalManufacture Inc. - 2024 Operations Dashboard"
- Proper spacing (use subplots_adjust or tight_layout)
- Each subplot should be readable and informative
- Save as: `M3L03_YourName_CategoricalDashboard.png`

Deliverables:

1. Python file: `M3L03_YourName_Categorical.py`
2. PNG files (9 files minimum):
   - `M3L03_YourName_ProductBoxPlot.png`
   - `M3L03_YourName_ProductLayered.png`
   - `M3L03_YourName_RegionalTrends.png`
   - `M3L03_YourName_QuarterlyComparison.png`
   - `M3L03_YourName_SalaryViolin.png`
   - `M3L03_YourName_SalarySatisfaction.png`
   - `M3L03_YourName_CustomerCount.png`
   - `M3L03_YourName_SegmentRevenue.png`
   - `M3L03_YourName_CategoricalDashboard.png` (bonus)

Grading Rubric:

- Part A (Product Analysis): 25 points
- Part B (Regional Trends): 30 points
- Part C (Employee Analysis): 25 points
- Part D (Customer Segments): 20 points
- Bonus (Dashboard): +25 points

Success Criteria:

☐ Appropriate plot types selected for each question
☐ Categories ordered logically (not alphabetically)
☐ All visualizations include proper labels and units
☐ Color choices are professional and meaningful
☐ Statistical features (means, medians, reference lines) included
☐ Business insights documented in comments
☐ Layered plots show complementary information
☐ High-resolution outputs

Categorical Visualization Decision Framework:

Question	Best Plot Type
How many in each category?	Count plot
What's the average per category?	Bar plot
How do distributions differ?	Violin or Box plot
What are trends across categories?	Point plot
Need to see all data points?	Strip or Swarm plot
Complex distribution comparison?	Layered: Violin + Box + Strip

Common Mistakes to Avoid:

- Alphabetical ordering when logical ordering exists
- Too many categories in one plot (>8 becomes cluttered)
- Not showing variability (error bars, distributions)
- Ignoring outliers without investigation
- Poor color choices (rainbow palette, not colorblind-safe)
- Missing reference lines or benchmarks
- Not connecting categorical findings to business actions

3.4 Section 4: Heatmaps and Correlation Analysis

3.4.1 Objective

- Create and interpret correlation heatmaps for multivariate analysis
- Understand correlation coefficients and their business implications
- Use annotated heatmaps to communicate numerical relationships
- Create pivot table heatmaps for two-way categorical analysis
- Apply appropriate color schemes for different data types
- Identify multicollinearity and redundant variables
- Use clustermap for hierarchical relationship visualization

3.4.2 Main Contents with Examples

The Challenge of Multivariate Data

Beyond Two Variables:

So far, we've explored:

- Single variables (histograms, KDE)
- Two variables (scatter plots, joint plots)
- Categories with one metric (box plots, bar plots)

But business data is **multivariate** - many variables interacting:

- Customer data: Age, income, tenure, purchase frequency, satisfaction, lifetime value
- Product metrics: Price, cost, sales volume, returns, ratings, reviews
- Financial data: Revenue, expenses, profit, assets, liabilities, ratios

The Visualization Problem:

How do you visualize 10+ variables simultaneously?

- Pair plot works but becomes overwhelming (10 variables = 45 plots!)
- Need a compact way to see all relationships at once

Solution: Heatmaps

Heatmaps show many values at once using color intensity:

- **Rows**: Variables
- **Columns**: Variables (or categories)
- **Color**: Strength of relationship or value

One glance reveals patterns across dozens of relationships.

Understanding Correlation

What is Correlation?

Correlation measures the **strength and direction** of a linear relationship between two variables.

Correlation Coefficient (r):

- Ranges from **-1 to +1**
- **+1**: Perfect positive linear relationship (as X↑, Y↑)
- **0**: No linear relationship
- **-1**: Perfect negative linear relationship (as X↑, Y↓)

Interpretation Scale:

Coefficient Range	Interpretation	Business Meaning
0.9 to 1.0	Very strong positive	Highly predictive
0.7 to 0.9	Strong positive	Reliable relationship
0.5 to 0.7	Moderate positive	Noticeable pattern
0.3 to 0.5	Weak positive	Slight tendency
0.0 to 0.3	Very weak	Minimal relationship

Coefficient Range	Interpretation	Business Meaning
-0.3 to 0.0	Very weak negative	Minimal inverse
-0.5 to -0.3	Weak negative	Slight inverse
-0.7 to -0.5	Moderate negative	Noticeable inverse
-0.9 to -0.7	Strong negative	Reliable inverse
-1.0 to -0.9	Very strong negative	Highly inverse

Business Examples:

Strong Positive (r = 0.85):

- Advertising spend → Sales
- **Interpretation**: Reliable predictor, invest with confidence

Moderate Positive (r = 0.55):

- Customer age → Average purchase amount
- **Interpretation**: Pattern exists but other factors matter

Weak Negative (r = -0.35):

- Product price → Sales volume
- **Interpretation**: Price affects sales slightly, but demand is relatively inelastic

Very Weak (r = 0.12):

- Employee height → Salary
- **Interpretation**: No meaningful relationship (as expected!)

Creating Correlation Matrices

Calculating Correlations in Pandas:

```python
import pandas as pd
import numpy as np

# Sample business data
np.random.seed(42)

df = pd.DataFrame({
    'revenue': np.random.normal(100000, 20000, 100),
    'marketing_spend': np.random.normal(15000, 3000, 100),
    'sales_calls': np.random.poisson(25, 100),
    'customer_satisfaction': np.random.uniform(3.5, 5.0, 100),
    'product_price': np.random.uniform(50, 200, 100)
})

# Add realistic relationships
df['revenue'] = (50000 +
                 2.5 * df['marketing_spend'] +
                 500 * df['sales_calls'] +
                 np.random.normal(0, 5000, 100))

# Calculate correlation matrix
correlation_matrix = df.corr()
print(correlation_matrix)
```

Understanding the Correlation Matrix:

The output is a square table:

```
                     revenue  marketing_spend  sales_calls  satisfaction  price
revenue              1.000000         0.856234     0.723451      0.234567  -0.102345
marketing_spend      0.856234         1.000000     0.145678      0.198765   0.034567
```

```
sales_calls          0.723451         0.145678     1.000000      0.287654  -0.056789
satisfaction         0.234567         0.198765     0.287654      1.000000   0.123456
price               -0.102345         0.034567    -0.056789      0.123456   1.000000
```

Reading the Matrix:

- **Diagonal**: Always 1.0 (perfect correlation with itself)
- **Symmetric**: Upper and lower triangles are mirrors
- **Focus on**: Off-diagonal values (actual relationships)

Key Insights:

- Revenue & Marketing Spend: r=0.86 (strong positive - good ROI!)
- Revenue & Sales Calls: r=0.72 (strong positive - calls matter!)
- Revenue & Price: r=-0.10 (weak negative - price doesn't hurt much)

Creating Basic Correlation Heatmaps

Visualizing the Correlation Matrix:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Generate comprehensive business metrics data
np.random.seed(42)
n = 200

df = pd.DataFrame({
    'Revenue': np.random.normal(100000, 20000, n),
    'Marketing_Spend': np.random.normal(15000, 3000, n),
    'Sales_Team_Size': np.random.randint(5, 25, n),
    'Customer_Count': np.random.poisson(150, n),
    'Avg_Deal_Size': np.random.normal(5000, 1500, n),
    'Customer_Satisfaction': np.random.uniform(3.5, 5.0, n),
    'Employee_Satisfaction': np.random.uniform(3.0, 5.0, n),
    'Product_Quality_Score': np.random.normal(85, 10, n)
})

# Create realistic relationships
df['Revenue'] = (30000 +
                 2.5 * df['Marketing_Spend'] +
                 2000 * df['Sales_Team_Size'] +
                 300 * df['Customer_Count'] +
                 np.random.normal(0, 8000, n))

df['Customer_Satisfaction'] = (2.5 +
                               0.02 * df['Product_Quality_Score'] +
                               0.00001 * df['Revenue'] +
                               np.random.normal(0, 0.3, n))
df['Customer_Satisfaction'] = df['Customer_Satisfaction'].clip(1, 5)

# Calculate correlation matrix
corr_matrix = df.corr()

# Create heatmap
fig, ax = plt.subplots(figsize=(12, 10))

sns.heatmap(corr_matrix,
            annot=True,           # Show correlation values
            fmt='.2f',            # 2 decimal places
            cmap='coolwarm',      # Color scheme
            center=0,             # Center colormap at 0
            square=True,          # Square cells
            linewidths=1,         # Lines between cells
            cbar_kws={'shrink': 0.8},  # Colorbar size
            ax=ax)

ax.set_title('Business Metrics Correlation Heatmap',
             fontsize=16, fontweight='bold', pad=20)

plt.tight_layout()
plt.show()
```

Heatmap Parameters Explained:

annot=True: Shows actual correlation values

- Critical for business presentations
- Enables precise interpretation
- Numbers complement colors

fmt='.2f': Format for annotations

- '.2f': Two decimals (0.85)
- '.3f': Three decimals (0.853)
- '.1f': One decimal (0.9)

cmap='coolwarm': Color scheme

- **coolwarm**: Blue (negative) to Red (positive), white (zero)
- **RdYlGn**: Red (low) to Green (high) - good for all positive
- **viridis**: Perceptually uniform, colorblind-safe
- **vlag**: Similar to coolwarm, more saturated

center=0: Where to center the colormap

- **0**: Natural for correlations (-1 to +1)
- Makes positive/negative clear

square=True: Makes cells square

- Easier to read
- More professional appearance

Reading a Correlation Heatmap:

Strong Positive (Dark Red):

- Revenue & Marketing_Spend: 0.86
- **Action**: Continue investing in marketing

Strong Negative (Dark Blue):

- (If present) indicates inverse relationships
- Example: Price & Volume might be -0.65

Weak Correlations (White/Light):

- Employee_Satisfaction & Revenue: 0.15
- **Interpretation**: No direct link (may be indirect through other variables)

Surprising Patterns:

- Customer_Satisfaction & Revenue: Only 0.32
- **Question**: Why isn't satisfaction driving revenue more?
- **Possible answer**: Long-term effect not captured in current period data

Advanced Heatmap Techniques

Masking the Upper Triangle (Avoid Redundancy):

Since correlation matrices are symmetric, showing both triangles is redundant:

```
import numpy as np

# Generate mask for upper triangle
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

fig, ax = plt.subplots(figsize=(12, 10))

sns.heatmap(corr_matrix,
            mask=mask,            # Hide upper triangle
            annot=True,
            fmt='.2f',
            cmap='coolwarm',
            center=0,
            square=True,
            linewidths=1,
            cbar_kws={'shrink': 0.8},
            ax=ax)

ax.set_title('Correlation Heatmap (Lower Triangle)',
             fontsize=16, fontweight='bold')

plt.tight_layout()
plt.show()
```

Benefits:

- Cleaner, more professional
- Eliminates distraction
- Saves space for larger variable sets

Highlighting Strong Correlations:

```
# Create custom annotations to highlight strong correlations
annot_array = corr_matrix.values.copy()
annotations = []

for i in range(len(annot_array)):
    row_annot = []
    for j in range(len(annot_array[i])):
        val = annot_array[i, j]
        if abs(val) > 0.7 and i != j:  # Strong correlation, not diagonal
            row_annot.append(f'{val:.2f}*')  # Add asterisk
        else:
            row_annot.append(f'{val:.2f}')
    annotations.append(row_annot)

fig, ax = plt.subplots(figsize=(12, 10))

sns.heatmap(corr_matrix,
            annot=np.array(annotations),
            fmt='',  # Don't format, we already did
            cmap='coolwarm',
            center=0,
            square=True,
            linewidths=1,
            ax=ax)

ax.set_title('Correlation Heatmap (* indicates |r| > 0.7)',
             fontsize=16, fontweight='bold')

plt.tight_layout()
plt.show()
```

Pivot Table Heatmaps - Two-Way Categorical Analysis

Beyond Correlation - Categorical Relationships:

Heatmaps aren't just for correlations. They excel at showing **values across two categorical dimensions**:

- Average sales by Region × Quarter
- Customer count by Segment × Industry
- Satisfaction scores by Department × Tenure

Creating Pivot Table Heatmaps:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Generate sales data by region and quarter
np.random.seed(42)

regions = ['North', 'South', 'East', 'West']
quarters = ['Q1', 'Q2', 'Q3', 'Q4']

data_list = []
for region in regions:
    for quarter in quarters:
        # Different regions have different patterns
        if region == 'East':
            base = 85000
        elif region == 'West':
            base = 78000
        elif region == 'North':
            base = 82000
        else:
            base = 75000

        # Seasonal variation by quarter
        if quarter == 'Q4':
            seasonal = 15000  # Holiday boost
        elif quarter == 'Q1':
            seasonal = -5000  # Post-holiday slump
        else:
            seasonal = 0

        sales = base + seasonal + np.random.normal(0, 5000)
        data_list.append({
            'Region': region,
            'Quarter': quarter,
            'Sales': sales
        })

df_sales = pd.DataFrame(data_list)
```

```
# Create pivot table
pivot_table = df_sales.pivot_table(values='Sales',
                                    index='Region',
                                    columns='Quarter',
                                    aggfunc='mean')

# Create heatmap
fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(pivot_table,
            annot=True,
            fmt='.0f',  # No decimals for currency
            cmap='YlGnBu',  # Yellow to Green to Blue
            linewidths=2,
            linecolor='white',
            cbar_kws={'label': 'Average Sales ($)'},
            ax=ax)

ax.set_title('Average Sales by Region and Quarter - 2024',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Quarter', fontsize=12)
ax.set_ylabel('Region', fontsize=12)

plt.tight_layout()
plt.show()
```

Reading Two-Way Heatmaps:

Hotspots (Darker colors):

- East + Q4: Highest sales ($100K)
- **Insight**: East region + Holiday season = best performance

Cold spots (Lighter colors):

- South + Q1: Lowest sales ($70K)
- **Insight**: South struggles post-holiday, needs Q1 promotion

Patterns:

- All regions: Q4 is best (holiday effect)
- East region: Consistently outperforms (across all quarters)

Business Actions:

- Allocate more inventory to East in Q4
- Create Q1 promotions for South
- Study East's practices to replicate elsewhere

Clustermaps - Discovering Hidden Groupings

The Pattern Recognition Problem:

Sometimes relationships aren't obvious from raw order. Clustermaps **automatically reorder** rows and columns to group similar variables together.

What Clustermaps Reveal:

- Which variables behave similarly
- Natural groupings in your metrics
- Redundant variables (measure same thing)
- Hierarchical relationships

Creating a Clustermap:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Generate diverse business metrics
np.random.seed(42)
n = 150

df = pd.DataFrame({
    # Financial metrics (should cluster)
    'Revenue': np.random.normal(100000, 20000, n),
    'Profit': np.random.normal(25000, 8000, n),
    'Cash_Flow': np.random.normal(30000, 10000, n),

    # Customer metrics (should cluster)
    'Customer_Count': np.random.poisson(200, n),
    'Customer_Satisfaction': np.random.uniform(3.5, 5.0, n),
    'Net_Promoter_Score': np.random.normal(45, 15, n),

    # Operational metrics (should cluster)
    'Employee_Count': np.random.randint(20, 100, n),
    'Productivity_Score': np.random.normal(75, 12, n),
    'Efficiency_Rating': np.random.uniform(65, 95, n),

    # Marketing metrics (should cluster)
    'Marketing_Spend': np.random.normal(15000, 5000, n),
    'Lead_Generation': np.random.poisson(50, n),
    'Conversion_Rate': np.random.uniform(0.15, 0.35, n)
})

# Create realistic relationships within clusters
df['Profit'] = 0.25 * df['Revenue'] + np.random.normal(0, 3000, n)
df['Cash_Flow'] = 0.30 * df['Revenue'] + np.random.normal(0, 4000, n)
df['Net_Promoter_Score'] = 20 + 6 * df['Customer_Satisfaction'] + np.random.normal(0,
 ↪  5, n)
df['Productivity_Score'] = 50 + 0.25 * df['Employee_Count'] + np.random.normal(0, 8,
 ↪  n)
df['Conversion_Rate'] = 0.15 + 0.000003 * df['Marketing_Spend'] + np.random.normal(0,
 ↪  0.03, n)

# Calculate correlation
corr_matrix = df.corr()

# Create clustermap
g = sns.clustermap(corr_matrix,
                   annot=True,
                   fmt='.2f',
                   cmap='coolwarm',
                   center=0,
                   figsize=(14, 14),
                   linewidths=0.5,
                   cbar_kws={'label': 'Correlation Coefficient'})

g.fig.suptitle('Clustered Correlation Heatmap - Business Metrics',
               fontsize=16, fontweight='bold', y=0.98)

plt.tight_layout()
plt.show()
```

Reading a Clustermap:

Dendrograms (Tree structures on sides):

- Show how variables are grouped
- Shorter branches = more similar
- Longer branches = less similar

Reordered Matrix:

- Variables automatically rearranged
- Similar variables placed adjacent
- Reveals natural groupings

Business Insights from Clusters:

Cluster 1: Financial Metrics

- Revenue, Profit, Cash_Flow group together
- **Insight**: These move together (as expected)
- **Action**: Can use Revenue as proxy for financial health

Cluster 2: Customer Metrics

- Customer_Satisfaction, NPS strongly linked
- **Insight**: Satisfied customers promote us
- **Action**: Focus on satisfaction, NPS will follow

Cluster 3: Operational Metrics

- Employee_Count, Productivity separate from others
- **Insight**: Operations independent of customer metrics
- **Question**: Should they be? Investigate connection

Multicollinearity and Variable Selection

The Redundancy Problem:

In predictive modeling or analysis, highly correlated independent variables cause problems:

- **Multicollinearity**: When predictors correlate with each other
- **Result**: Unstable models, unclear which variable matters
- **Solution**: Remove redundant variables

Using Heatmaps to Identify Redundancy:

```
# Identify highly correlated variable pairs (excluding diagonal)
high_corr_pairs = []
threshold = 0.85

for i in range(len(corr_matrix.columns)):
    for j in range(i+1, len(corr_matrix.columns)):
        if abs(corr_matrix.iloc[i, j]) >= threshold:
            high_corr_pairs.append({
                'Variable 1': corr_matrix.columns[i],
                'Variable 2': corr_matrix.columns[j],
                'Correlation': corr_matrix.iloc[i, j]
            })

df_high_corr = pd.DataFrame(high_corr_pairs)
print("\nHighly Correlated Variable Pairs (|r| >= 0.85):")
print(df_high_corr)
```

Business Decision Framework:

If Revenue and Cash_Flow correlate at r=0.92: 1. **Keep one, drop the other** for modeling 2. **Which to keep?** - The one easier to measure - The one more relevant to the business question - The one with less measurement error

Example:

- Revenue and Profit: r=0.88
- **For sales forecasting**: Keep Revenue (direct measure)
- **For profitability analysis**: Keep Profit (more relevant)

Color Scheme Selection for Different Contexts

Choosing the Right Colors Matters:

Different data types and questions require different color schemes:

Diverging (Best for Correlations):

```
# Data centered at 0, both positive and negative values
cmap='coolwarm'  # Blue (negative) → White (zero) → Red (positive)
cmap='RdBu_r'    # Red → Blue reversed
cmap='PiYG'      # Pink → Yellow → Green
```

Sequential (Best for One-directional Values):

```
# All positive values, low to high
cmap='YlGnBu'    # Yellow → Green → Blue
cmap='Blues'     # Light blue → Dark blue
cmap='Reds'      # Light red → Dark red
```

Qualitative (Best for Categories):

```
# Distinct categories, no order
cmap='Set1'
cmap='Set2'
cmap='tab10'
```

Business Context Examples:

Financial Performance (diverging):

- Negative (loss) = Red
- Zero (break-even) = White
- Positive (profit) = Green

```
cmap='RdYlGn'  # Red-Yellow-Green
```

Sales Performance (sequential):

- All positive values
- Low = Light, High = Dark

```
cmap='YlOrRd'  # Yellow-Orange-Red (heat map feel)
```

Customer Satisfaction (sequential):

- All positive (1-5 scale)
- Low satisfaction = Red, High = Green

```
cmap='RdYlGn'  # But not centered at 0, normalized to data range
```

3.4.3 Lab Session

Lab 4: Correlation and Heatmap Analysis

Objective: Master correlation analysis and heatmap visualization to uncover complex relationships in multivariate business data and communicate insights through color-coded matrices.

Scenario: You're the Senior Data Analyst at "MarketPro Analytics," a marketing analytics firm. Your client, a major e-commerce company, has provided comprehensive data on their operations. The CMO is struggling to understand which metrics drive success and where to focus improvement efforts. Your task is to create correlation analyses and heatmaps that reveal the hidden relationships in their complex business data and provide clear, actionable insights for strategic planning.

Pre-Lab Setup:

1. Create file: `M3L04_YourName_Heatmaps.py`
2. Import and configure:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

sns.set_theme(style="white", context="talk")
```

3. Create output folder: "Lab4_Outputs"

Part A: Comprehensive Business Metrics Correlation Analysis (30 points)

Context: The client wants to understand how their various business metrics relate to each other to identify key drivers of success.

Data:

```
np.random.seed(42)
n = 250  # 250 days of data

# Generate comprehensive e-commerce metrics
df_metrics = pd.DataFrame({
    'Daily_Revenue': np.random.normal(50000, 12000, n),
    'Website_Traffic': np.random.poisson(15000, n),
    'Conversion_Rate': np.random.uniform(0.02, 0.08, n),
    'Avg_Order_Value': np.random.normal(85, 20, n),
    'Marketing_Spend': np.random.normal(3000, 800, n),
    'Email_Open_Rate': np.random.uniform(0.15, 0.35, n),
    'Social_Media_Engagement': np.random.poisson(500, n),
    'Customer_Service_Tickets': np.random.poisson(45, n),
    'Product_Returns': np.random.poisson(25, n),
    'Page_Load_Time': np.random.uniform(1.5, 4.5, n),
    'Mobile_Traffic_Pct': np.random.uniform(0.45, 0.75, n),
    'Customer_Satisfaction': np.random.uniform(3.8, 4.8, n)
})

# Create realistic relationships
df_metrics['Daily_Revenue'] = (
    10000 +
    1.8 * df_metrics['Website_Traffic'] +
    200000 * df_metrics['Conversion_Rate'] +
    300 * df_metrics['Avg_Order_Value'] +
    2.5 * df_metrics['Marketing_Spend'] +
    -2000 * df_metrics['Page_Load_Time'] +
    np.random.normal(0, 5000, n)
)

df_metrics['Conversion_Rate'] = (
    0.03 +
    0.00000015 * df_metrics['Website_Traffic'] +
    0.01 * df_metrics['Email_Open_Rate'] +
    -0.003 * df_metrics['Page_Load_Time'] +
    0.002 * df_metrics['Customer_Satisfaction'] +
    np.random.normal(0, 0.005, n)
)
df_metrics['Conversion_Rate'] = df_metrics['Conversion_Rate'].clip(0.01, 0.10)

df_metrics['Customer_Satisfaction'] = (
    3.0 +
    0.3 * (5 - df_metrics['Page_Load_Time']) +
    -0.015 * df_metrics['Customer_Service_Tickets'] +
    -0.020 * df_metrics['Product_Returns'] +
    np.random.normal(0, 0.2, n)
)
df_metrics['Customer_Satisfaction'] = df_metrics['Customer_Satisfaction'].clip(1, 5)
```

Tasks:

1. **Create comprehensive correlation heatmap (12 points):**
    - Calculate full correlation matrix
    - Figure size: (16, 14)
    - Use ‘coolwarm’ colormap centered at 0
    - Annotate with correlation values (2 decimals)
    - Square cells with white lines (linewidth=1)
    - Add colorbar with label “Correlation Coefficient”
    - Title: “E-Commerce Business Metrics - Correlation Analysis”
    - Make text readable (appropriate font sizes)
2. **Create lower triangle version (8 points):**
    - Same heatmap but mask upper triangle
    - Eliminates redundancy
    - Cleaner professional appearance
    - Save as: `M3L04_YourName_FullCorrelation.png` (full) and `M3L04_YourName_LowerTriangle.png` (masked)
3. **Identify and document key insights (10 points):**
    - Find the THREE strongest positive correlations (excluding diagonal)
    - Find the THREE strongest negative correlations
    - Create a summary report in comments:
        - List these correlations with values
        - Explain what each relationship means for business
        - Identify surprising or counter-intuitive findings
        - Recommend which metrics to focus on for improving revenue
        - Identify any potential multicollinearity issues (r > 0.85)

Part B: Revenue Driver Deep Dive (25 points)

Context: The CMO specifically wants to know which factors most strongly influence revenue. Create a focused analysis on revenue correlations.

Tasks:

1. **Create revenue-focused correlation visualization (15 points):**
    - Extract correlations of all variables with ‘Daily_Revenue’
    - Create a horizontal bar plot showing these correlations
    - Sort by correlation strength (highest to lowest by absolute value)
    - Color bars: Green for positive, Red for negative
    - Figure size: (12, 8)
    - Title: “Revenue Correlation Analysis - Key Drivers Identified”
    - Add vertical reference lines at -0.3, 0, +0.3, +0.6 (weak, none, moderate, strong thresholds)
    - Annotate bars with correlation values

2. **Statistical significance and interpretation (10 points):**
    - Calculate correlation with p-values using scipy.stats.pearsonr
    - In comments, document:
        - Top 3 revenue drivers (highest positive correlations)
        - Whether these correlations are statistically significant ($p < 0.05$)
        - Which metrics have surprisingly weak correlation with revenue
        - Actionable recommendations for increasing revenue based on findings
    - Create a text annotation on the plot highlighting the #1 revenue driver
    - Save as: `M3L04_YourName_RevenueDrivers.png`

Part C: Marketing Performance Pivot Heatmap (25 points)

Context: The marketing team runs campaigns across different channels (Email, Social, PPC, Display) in different customer segments (New, Returning, VIP). They need to see performance by Channel × Segment.

Data:

```
np.random.seed(42)

channels = ['Email', 'Social Media', 'PPC', 'Display Ads']
segments = ['New Customers', 'Returning', 'VIP']

marketing_data = []
for channel in channels:
    for segment in segments:
        # Different channels perform differently with different segments
        if channel == 'Email' and segment == 'VIP':
            roi = np.random.normal(4.5, 0.5, 30)  # Email works great for VIP
        elif channel == 'Social Media' and segment == 'New Customers':
            roi = np.random.normal(3.8, 0.6, 30)  # Social good for acquisition
        elif channel == 'PPC' and segment == 'Returning':
            roi = np.random.normal(3.5, 0.4, 30)
        elif channel == 'Display Ads':
            roi = np.random.normal(2.2, 0.5, 30)  # Display generally weaker
        else:
            roi = np.random.normal(3.0, 0.5, 30)

        for r in roi:
            marketing_data.append({
                'Channel': channel,
                'Segment': segment,
                'ROI': max(r, 0.5)  # Ensure positive ROI
            })

df_marketing = pd.DataFrame(marketing_data)
```

Tasks:

1. **Create pivot table heatmap (15 points):**
    - Pivot: Channels as rows, Segments as columns, Average ROI as values
    - Figure size: (10, 6)
    - Use 'RdYlGn' colormap (Red=low, Yellow=medium, Green=high)
    - Annotate with ROI values (1 decimal place)
    - Add colorbar labeled "Average ROI ($return per $1 spent)"
    - Title: "Marketing ROI by Channel and Customer Segment"
    - Thick white lines between cells (linewidth=2)
2. **Add conditional formatting context (5 points):**
    - Calculate overall average ROI

- Add subtitle text below title: "Overall Average ROI: $X.XX"
- In annotations, mark cells with * if ROI > overall average + 0.5
- This highlights exceptional performance

3. **Business recommendations (5 points):**
    - In code comments, identify:
        - Best performing Channel-Segment combination
        - Worst performing combination
        - Where to increase marketing budget
        - Where to decrease or eliminate spending
        - Unexpected patterns or opportunities
    - Save as: `M3L04_YourName_MarketingROI.png`

Part D: Customer Behavior Clustermap (20 points)

Context: The product team has metrics on customer behavior but doesn't know which behaviors cluster together or which customer segments behave similarly.

Data:

```
np.random.seed(42)

# Customer behavior metrics
behaviors = pd.DataFrame({
    'Purchase_Frequency': np.random.gamma(2, 2, 200),
    'Avg_Session_Duration': np.random.normal(8, 3, 200),
    'Pages_Per_Session': np.random.poisson(5, 200),
    'Cart_Abandonment_Rate': np.random.uniform(0.3, 0.8, 200),
    'Wishlist_Usage': np.random.poisson(3, 200),
    'Review_Activity': np.random.poisson(2, 200),
    'Social_Shares': np.random.poisson(1, 200),
    'Coupon_Usage_Rate': np.random.uniform(0.1, 0.6, 200),
    'Customer_Service_Contact': np.random.poisson(1, 200),
    'Mobile_App_Usage': np.random.uniform(0.2, 0.9, 200)
})

# Create some natural relationships
behaviors['Purchase_Frequency'] = (
    behaviors['Purchase_Frequency'] *
    (1 + 0.3 * behaviors['Wishlist_Usage'] / behaviors['Wishlist_Usage'].max())
)

behaviors['Review_Activity'] = (
    behaviors['Review_Activity'] +
    0.3 * behaviors['Purchase_Frequency']
)

behaviors['Social_Shares'] = (
    behaviors['Social_Shares'] +
    0.5 * behaviors['Review_Activity']
)

df_behaviors = behaviors
```

Tasks:

1. **Create clustermap (15 points):**
    - Calculate correlation matrix
    - Create clustermap with:
        - Figure size: (14, 14)
        - 'coolwarm' colormap centered at 0

    - Annotations (2 decimals)
    - Appropriate dendrogram ratio
  - Title: “Customer Behavior Clustering Analysis”
  - Both row and column clustering enabled

2. **Interpret clusters (5 points):**

   - In detailed comments, identify:
     - How many natural behavior clusters exist?
     - Which behaviors group together? (Name the clusters meaningfully)
     - Example: “Engagement Cluster: Review_Activity, Social_Shares, Wishlist_Usage”
     - Which behaviors are independent/isolated?
     - What do these clusters tell us about customer types?
     - How can marketing use this segmentation?
   - Save as: `M3L04_YourName_BehaviorClustermap.png`

Bonus Challenge (+30 points):

Create a Comprehensive Executive Dashboard with Multiple Heatmaps:

Build a figure with 4 subplots (2×2) showing:

1. **Top-left: Correlation heatmap of top 6 business metrics**
   - Select the 6 most important metrics
   - Lower triangle only
   - Highlight strong correlations
2. **Top-right: Time series heatmap**
   - Create data: Metrics (rows) × Weeks (columns)
   - Show how metrics evolved over 12 weeks
   - Use sequential colormap
3. **Bottom-left: Comparison heatmap**
   - This Year vs Last Year performance
   - Metrics as rows, Year as columns
   - Show % change with diverging colors
4. **Bottom-right: Priority matrix**
   - Impact (Low/Medium/High) × Effort (Low/Medium/High)
   - Place top 9 initiatives in appropriate cells
   - Color by priority level

Requirements:

- Figure size: (20, 18)
- Consistent, professional styling
- Overall title: “MarketPro Analytics - Executive Dashboard”
- Each subplot clearly labeled
- Annotations where appropriate
- Color schemes selected for context
- Save as: `M3L04_YourName_ExecutiveDashboard.png`

Deliverables:

1. Python file: `M3L04_YourName_Heatmaps.py`
2. PNG files (minimum 5):
   - `M3L04_YourName_FullCorrelation.png`
   - `M3L04_YourName_LowerTriangle.png`
   - `M3L04_YourName_RevenueDrivers.png`
   - `M3L04_YourName_MarketingROI.png`
   - `M3L04_YourName_BehaviorClustermap.png`
   - `M3L04_YourName_ExecutiveDashboard.png` (bonus)

Grading Rubric:

- Part A (Correlation Analysis): 30 points
- Part B (Revenue Drivers): 25 points
- Part C (Marketing Pivot): 25 points

- Part D (Clustermap): 20 points
- Bonus (Dashboard): +30 points

Success Criteria:

- □ Correlation values calculated correctly
- □ Appropriate color schemes selected for data context
- □ Annotations clear and readable
- □ Business insights documented in detail
- □ Multicollinearity issues identified
- □ Clustermaps reveal meaningful groupings
- □ Pivot heatmaps show actionable patterns
- □ All visualizations properly labeled with titles and colorbars
- □ High-resolution professional outputs

Heatmap Best Practices Checklist:

- □ Choose colormap appropriate for data type (diverging vs sequential)
- □ Center diverging colormaps at meaningful value (usually 0)
- □ Always annotate correlation heatmaps with values
- □ Use appropriate number formats (correlations: .2f, money: .0f)
- □ Include colorbar with descriptive label
- □ Make cells square for easier reading
- □ Use lines between cells for clarity
- □ Consider masking upper triangle for correlations
- □ Ensure text is readable (font size, contrast)
- □ Add meaningful titles that explain what's shown

Common Mistakes to Avoid:

- Using rainbow/jet colormap (not perceptually uniform)
- Forgetting to center diverging colormaps
- Too many variables (>15 becomes unreadable)
- Poor color contrast with annotations
- Missing colorbar or unlabeled colorbar
- Not interpreting business meaning of correlations
- Confusing correlation with causation
- Ignoring statistical significance

Professional Tips:

- Strong correlation doesn't mean causation
- Always consider lag effects (marketing → sales may take time)
- Look for unexpected weak correlations (why isn't X related to Y?)
- Use clustermaps to discover natural groupings
- Pivot heatmaps excellent for two-way categorical analysis
- Document threshold for "strong" correlation in your context
- Consider removing redundant highly-correlated variables for modeling

---

End of Module 3: Statistical Visualization with Seaborn

Key Takeaways:

- Seaborn provides high-level statistical visualizations with beautiful defaults
- Distribution plots (histogram, KDE, box, violin) reveal data characteristics
- Relationship plots (scatter with regression, joint, pair) uncover correlations
- Categorical plots (count, bar, box, violin, point) compare groups effectively
- Heatmaps and clustermaps visualize multivariate relationships compactly
- Always interpret statistical findings in business context
- Correlation does not imply causation

Chapter 4

Module 4: Interactive Visualization with Plotly

4.1 Section 1: Introduction to Plotly and Interactive Charts

4.1.1 Objective

- Understand the advantages of interactive visualizations for business
- Master Plotly's architecture and syntax differences from Matplotlib
- Create basic interactive plots: scatter, line, and bar charts
- Implement hover information and zoom capabilities
- Customize interactive elements for professional presentations
- Export interactive visualizations for web and reports

4.1.2 Main Contents with Examples

Why Interactive Visualizations Transform Business Communication

The Limitation of Static Charts:

In Modules 2 and 3, we created beautiful static visualizations with Matplotlib and Seaborn. But static charts have inherent limitations:

- **Fixed perspective**: Shows only one view of the data
- **Information overload**: Must include all details upfront (cluttered)
- **No exploration**: Viewers can't dig deeper into interesting patterns
- **Print-focused**: Designed for paper, not screens

The Interactive Advantage:

Interactive visualizations solve these problems:

1. Progressive Disclosure:

- Start with overview
- Users drill down into areas of interest
- Hover reveals details on demand
- Zoom focuses on specific regions

2. Enhanced Engagement:

- Users actively explore data
- Discover insights themselves
- Memorable and impactful
- Encourages questions and discussion

3. Richer Information:

- Pack more data without clutter
- Multiple metrics in one chart
- Time series with range selection
- Multi-dimensional views

Business Impact Examples:

Sales Dashboard (Static):

- Shows 12 months of sales
- Can't see daily fluctuations
- Questions require new charts

Sales Dashboard (Interactive):

- Click-drag to zoom into specific weeks
- Hover to see exact daily values
- Click legend to hide/show regions
- Export filtered view for reports

Result: Executives answer their own questions in real-time during presentations.

Understanding Plotly - The Interactive Visualization Library

What is Plotly?

Plotly is a graphing library that creates interactive, web-based visualizations:

- Built on **D3.js** and **JavaScript** (web standard)
- Python API (plotly.py) for easy creation
- Outputs **HTML** files viewable in any browser
- Hosted online or embedded in applications

Plotly Versions:

1. Plotly Express (px) - High-Level API:

```
import plotly.express as px
# Simple, concise, opinionated
fig = px.scatter(df, x='sales', y='profit')
```

- **Best for**: Quick exploration, standard charts
- **Advantages**: One-line plots, automatic styling
- **Used when**: Learning, rapid prototyping

2. Plotly Graph Objects (go) - Low-Level API:

```
import plotly.graph_objects as go
# Detailed control, more verbose
fig = go.Figure(data=[go.Scatter(x=sales, y=profit)])
```

- **Best for**: Custom visualizations, fine control
- **Advantages**: Complete flexibility
- **Used when**: Specific requirements, complex layouts

We'll focus primarily on Plotly Express (easier to learn, covers 90% of business needs).

Installation Check:

```
import plotly
print(plotly.__version__)  # Should be 5.x or higher
```

If not installed:

```
pip install plotly
```

Your First Interactive Chart - The Plotly Workflow

Understanding the Fundamental Difference:

Matplotlib/Seaborn workflow:

1. Create figure and axes
2. Add data
3. Customize
4. Show or save as PNG/PDF

Plotly workflow:

1. Create figure with data
2. Customize (optional)
3. Show in browser OR save as HTML

Creating a Simple Interactive Scatter Plot:

```python
import plotly.express as px
import pandas as pd
import numpy as np

# Sample business data: Marketing spend vs Revenue
np.random.seed(42)
df = pd.DataFrame({
    'marketing_spend': np.random.uniform(5000, 50000, 100),
    'revenue': np.random.uniform(50000, 500000, 100),
    'region': np.random.choice(['North', 'South', 'East', 'West'], 100),
    'quarter': np.random.choice(['Q1', 'Q2', 'Q3', 'Q4'], 100)
})

# Add realistic relationship
df['revenue'] = 30000 + 6 * df['marketing_spend'] + np.random.normal(0, 30000, 100)

# Create interactive scatter plot
fig = px.scatter(df,
                 x='marketing_spend',
                 y='revenue',
                 title='Marketing Spend vs Revenue Analysis')

# Display in browser
fig.show()
```

What Just Happened:

When you run `fig.show()`: 1. Plotly generates an HTML file 2. Opens it in your default browser 3. You see an interactive chart with: - **Hover**: Move mouse over points to see values - **Zoom**: Click-drag to zoom into areas - **Pan**: Shift-click-drag to move around - **Reset**: Double-click to reset view - **Toolbar**: Top-right corner with tools

The Magic of Interactivity:

Try these interactions:

- **Hover over any point**: See exact marketing_spend and revenue values
- **Click-drag across interesting region**: Zoom in
- **Double-click**: Zoom back out
- **Click camera icon**: Download as PNG

No additional code needed - interactivity is automatic!

Customizing Hover Information

The Default Hover is Good, But We Can Do Better:

Default hover shows:

- Variable names (marketing_spend, revenue)
- Values

Professional hover should show:

- Formatted numbers ($50,000 not 50000)
- Additional context (Region, Quarter)
- Readable labels (“Marketing Spend” not “marketing_spend”)

Enhanced Hover with Custom Text:

```
import plotly.express as px
import pandas as pd
import numpy as np

# Same data as before
np.random.seed(42)
df = pd.DataFrame({
    'marketing_spend': np.random.uniform(5000, 50000, 100),
    'revenue': np.random.uniform(50000, 500000, 100),
    'region': np.random.choice(['North', 'South', 'East', 'West'], 100),
    'quarter': np.random.choice(['Q1', 'Q2', 'Q3', 'Q4'], 100),
    'campaign': [f'Campaign {i}' for i in range(100)]
})

df['revenue'] = 30000 + 6 * df['marketing_spend'] + np.random.normal(0, 30000, 100)

# Create hover template
fig = px.scatter(df,
                 x='marketing_spend',
                 y='revenue',
                 color='region',  # Color by region
                 hover_data=['quarter', 'campaign'],  # Additional info
                 title='Marketing ROI Analysis by Region')

# Update labels for readability
fig.update_layout(
    xaxis_title='Marketing Spend ($)',
    yaxis_title='Revenue Generated ($)',
    font=dict(size=12)
)

# Format hover to show currency
fig.update_traces(
    hovertemplate='<b>%{fullData.name}</b><br>' +
                  'Marketing: $%{x:,.0f}<br>' +
                  'Revenue: $%{y:,.0f}<br>' +
                  '<extra></extra>'
)

fig.show()
```

Understanding Hover Customization:

hover_data parameter:

- List of additional columns to show on hover
- Example: `['quarter', 'campaign']`
- Shows these fields without needing to map to x or y

hovertemplate:

- Custom HTML-like template
- **%{x}**: X-axis value
- **%{y}**: Y-axis value
- **:,.0f**: Format as number with commas, 0 decimals
- **…**: Bold text
- : Line break
- : Remove default trace info

Creating Interactive Line Charts

Time Series with Interactive Features:

Line charts are perfect for exploring temporal patterns interactively.

```
import plotly.express as px
import pandas as pd
import numpy as np

# Generate monthly revenue data for 3 years
np.random.seed(42)
dates = pd.date_range('2022-01-01', '2024-12-31', freq='D')

df_revenue = pd.DataFrame({
    'date': dates,
    'revenue': np.random.normal(100000, 15000, len(dates)),
    'region': np.random.choice(['North', 'South', 'East'], len(dates))
})

# Add trend and seasonality
df_revenue['revenue'] = (
    80000 +
    (df_revenue.index * 30) +  # Growth trend
    20000 * np.sin(df_revenue.index / 365 * 2 * np.pi) +  # Yearly seasonality
    df_revenue['revenue'] * 0.2
)

# Aggregate by month and region
df_monthly = df_revenue.groupby([pd.Grouper(key='date', freq='M'),
 ↪  'region'])['revenue'].sum().reset_index()

# Create interactive line chart
fig = px.line(df_monthly,
              x='date',
              y='revenue',
              color='region',
              title='Monthly Revenue Trends by Region (2022-2024)',
              labels={'revenue': 'Monthly Revenue ($)', 'date': 'Month'})

# Enhance layout
fig.update_layout(
    hovermode='x unified',  # Show all series at once when hovering
    xaxis=dict(
        rangeslider=dict(visible=True),  # Add range slider
        type='date'
    ),
    yaxis=dict(
        tickformat='$,.0f'  # Format Y-axis as currency
    ),
    legend=dict(
        title='Region',
        yanchor='top',
        y=0.99,
        xanchor='right',
        x=0.99
    )
)

fig.show()
```

Interactive Features Explained:

1. Range Slider (rangeslider):

- Appears below the chart
- Drag handles to select time range
- Chart auto-updates to show selected period
- Perfect for long time series (years of data)

2. Hover Mode 'x unified':

- Hover over any point on X-axis
- Shows values for ALL series at that X
- Compare multiple lines simultaneously
- Essential for multi-line charts

3. Legend Interaction:

- Click region name to hide/show
- Double-click to isolate one region
- Perfect for comparing specific series

Business Use Case:

Executive asks: "Did the South region dip in Q3 2023?" - Click-drag range slider to Q3 2023 - Hover over September 2023 - See all three regions' values - Answer immediately: "Yes, South dropped 15% while others remained stable"

Interactive Bar Charts with Drill-Down

Bar Charts with Rich Context:

```
import plotly.express as px
import pandas as pd
import numpy as np

# Product category sales data
np.random.seed(42)
categories = ['Electronics', 'Clothing', 'Home Goods', 'Sports', 'Books', 'Toys']

df_sales = pd.DataFrame({
    'category': categories,
    'q1_sales': np.random.uniform(200000, 500000, len(categories)),
    'q2_sales': np.random.uniform(250000, 550000, len(categories)),
    'q3_sales': np.random.uniform(230000, 520000, len(categories)),
    'q4_sales': np.random.uniform(300000, 600000, len(categories)),
    'units_sold': np.random.randint(5000, 25000, len(categories)),
    'avg_rating': np.random.uniform(3.8, 4.8, len(categories))
})

df_sales['total_sales'] = (df_sales['q1_sales'] + df_sales['q2_sales'] +
                           df_sales['q3_sales'] + df_sales['q4_sales'])

# Sort by total sales
```

```
df_sales = df_sales.sort_values('total_sales', ascending=True)

# Create horizontal bar chart
fig = px.bar(df_sales,
             y='category',
             x='total_sales',
             orientation='h',
             title='Annual Sales by Product Category - 2024',
             labels={'total_sales': 'Total Annual Sales ($)', 'category': ''},
             hover_data={
                 'total_sales': ':$,.0f',
                 'units_sold': ':,.0f',
                 'avg_rating': ':.1f'
             },
             color='total_sales',
             color_continuous_scale='Blues')

# Customize layout
fig.update_layout(
    showlegend=False,
    xaxis=dict(
        title='Annual Sales',
        tickformat='$,.0f',
        showgrid=True,
        gridcolor='lightgray'
    ),
    yaxis=dict(
        title=''
    ),
    height=500,
    coloraxis_colorbar=dict(
        title='Sales ($)',
        tickformat='$,.0f'
    )
)

fig.show()
```

Advanced Hover Formatting:

In `hover_data`:

- **Dictionary format**: `{column: format_string}`
- **':$,.0f'**: Dollar sign, comma separator, no decimals
- **':,.0f'**: Comma separator, no decimals
- **':.1f'**: One decimal place
- **':.2%'**: Percentage with 2 decimals

Color Scales:

- Single color intensifies with value
- Shows magnitude visually
- Popular scales: 'Blues', 'Reds', 'Greens', 'Viridis', 'Plasma'

Saving and Sharing Interactive Visualizations

Export Options:

1. HTML (Interactive):

```
fig.write_html('M4L01_Interactive_Sales.html')
```

- Fully interactive
- Opens in any browser
- Can be shared via email
- Embeds in websites
- File size: Usually 500KB - 2MB

2. Static Image (Non-Interactive):

```
fig.write_image('M4L01_Sales_Chart.png', width=1200, height=800)
```

- Requires: `pip install kaleido`
- Standard PNG/PDF/SVG
- For print reports
- Loses interactivity

3. For Presentations:

```
# Higher quality for projection
fig.write_image('M4L01_Presentation.png', width=1920, height=1080, scale=2)
```

Embedding in Reports:

Interactive HTML charts can be embedded in:

- **Jupyter notebooks**: Displays inline automatically
- **Web applications**: Iframe or direct embed
- **Confluence/SharePoint**: Upload HTML, link in document
- **Email**: Attach HTML file (recipients open in browser)

Plotly Express vs Graph Objects - When to Use Each

Quick Reference:

Use Plotly Express when:

- Creating standard chart types (scatter, line, bar, histogram)
- Quick exploration and analysis
- Learning Plotly
- Want concise, readable code
- Default styling is acceptable

Use Graph Objects when:

- Need complete customization control
- Creating complex, multi-trace figures
- Combining different chart types
- Building dynamic visualizations
- Need precise control over every element

Example Comparison:

Plotly Express (Simple):

```
import plotly.express as px

fig = px.scatter(df, x='sales', y='profit', color='region')
fig.show()
```

Graph Objects (More Control):

```
import plotly.graph_objects as go

fig = go.Figure()

for region in df['region'].unique():
    df_region = df[df['region'] == region]
    fig.add_trace(go.Scatter(
        x=df_region['sales'],
        y=df_region['profit'],
        mode='markers',
        name=region,
        marker=dict(size=10, opacity=0.7)
    ))

fig.update_layout(title='Sales vs Profit by Region')
fig.show()
```

Both produce similar results, but Express is much more concise for standard cases.

4.1.3 Lab Session

Lab 1: Creating Interactive Business Visualizations

Objective: Master Plotly's interactive capabilities by creating professional, explorable visualizations that enable stakeholders to discover insights through interaction.

Scenario: You're the Business Intelligence Developer at "DataDrive Consulting," hired by "MegaRetail Corp" to modernize their data presentation. The executive team is tired of static PowerPoint charts that can't answer follow-up questions. Your mission is to create interactive visualizations that executives can explore during meetings, allowing them to answer their own questions in real-time.

Pre-Lab Setup:

1. Create file: `M4L01_YourName_Interactive.py`
2. Import libraries:

```
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd
import numpy as np
```

3. Create output folder: "Lab1_Outputs"
4. **Important**: Install kaleido if not already: `pip install kaleido`

Part A: Interactive Sales Performance Explorer (25 points)

Context: The VP of Sales wants to explore 3 years of daily sales data across 5 regions. They need to zoom into specific periods, compare regions, and identify trends without requesting new charts.

Data:

```
np.random.seed(42)

# Generate 3 years of daily sales data
dates = pd.date_range('2022-01-01', '2024-12-31', freq='D')
regions = ['Northeast', 'Southeast', 'Midwest', 'Southwest', 'West']

data_list = []
for date in dates:
    for region in regions:
        # Base sales with regional differences
        base = {'Northeast': 25000, 'Southeast': 22000, 'Midwest': 28000,
                'Southwest': 20000, 'West': 30000}[region]

        # Add trend, seasonality, and randomness
        day_of_year = date.timetuple().tm_yday
        trend = (date.year - 2022) * 1000  # Growth over years
        seasonality = 5000 * np.sin(day_of_year / 365 * 2 * np.pi)  # Yearly cycle
        random_var = np.random.normal(0, 3000)

        sales = base + trend + seasonality + random_var

        data_list.append({
            'date': date,
            'region': region,
            'daily_sales': max(sales, 5000)  # Ensure positive
        })

df_sales = pd.DataFrame(data_list)

# Aggregate to monthly for cleaner visualization
df_monthly = df_sales.groupby([pd.Grouper(key='date', freq='M'), 'region'])['daily_sales'].sum().reset_index()
df_monthly['date'] = df_monthly['date'].dt.to_period('M').dt.to_timestamp()
```

Tasks:

1. **Create interactive line chart with range slider (12 points):**
   - Use `px.line()` with date on x-axis, daily_sales on y-axis
   - Color by region (5 different lines)
   - Title: "MegaRetail Corp - Monthly Sales Performance (2022-2024)"
   - Enable range slider on x-axis (`rangeslider=dict(visible=True)`)
   - Format y-axis as currency with commas
   - Set hovermode to 'x unified' (shows all regions when hovering)
   - Label axes properly: "Month" and "Monthly Sales ($)"
2. **Enhance interactivity and formatting (8 points):**
   - Format hover template to show:
     - Region name (bold)
     - Month in readable format (e.g., "Jan 2023")
     - Sales amount as "$1,234,567"
   - Position legend inside plot (top-right corner)
   - Add grid lines for easier reading
   - Set figure height to 600 pixels
   - Make legend clickable to hide/show regions
3. **Save and document (5 points):**
   - Save as interactive HTML: `M4L01_YourName_SalesExplorer.html`
   - Also save as PNG: `M4L01_YourName_SalesExplorer.png`
   - In code comments, document:
     - How to use the range slider to focus on Q4 2023
     - How to compare only Northeast and West (hiding others)
     - What seasonal pattern is visible

Part B: Marketing Campaign ROI Interactive Scatter (30 points)

Context: The CMO needs to evaluate 50 marketing campaigns across different channels and customer segments. They want to explore which campaigns had best ROI, identify outliers, and understand patterns through interaction.

Data:

```
np.random.seed(42)

channels = ['Email', 'Social Media', 'PPC', 'Display', 'Influencer']
segments = ['New Customers', 'Existing Customers', 'VIP', 'Lapsed']

campaigns_list = []
```

```python
for i in range(50):
    channel = np.random.choice(channels)
    segment = np.random.choice(segments)

    # Different channels have different cost structures
    if channel == 'Influencer':
        cost = np.random.uniform(15000, 50000)
        revenue = cost * np.random.uniform(2.0, 5.0)
    elif channel == 'PPC':
        cost = np.random.uniform(5000, 25000)
        revenue = cost * np.random.uniform(2.5, 4.5)
    elif channel == 'Email':
        cost = np.random.uniform(2000, 8000)
        revenue = cost * np.random.uniform(3.0, 6.0)
    elif channel == 'Social Media':
        cost = np.random.uniform(3000, 15000)
        revenue = cost * np.random.uniform(2.0, 5.0)
    else:  # Display
        cost = np.random.uniform(8000, 30000)
        revenue = cost * np.random.uniform(1.5, 3.5)

    campaigns_list.append({
        'campaign_id': f'CAMP-{i+1:03d}',
        'channel': channel,
        'segment': segment,
        'cost': cost,
        'revenue': revenue,
        'roi': (revenue - cost) / cost,
        'duration_days': np.random.randint(14, 90),
        'impressions': int(cost * np.random.uniform(5, 20))
    })

df_campaigns = pd.DataFrame(campaigns_list)
```

Tasks:

1. **Create advanced interactive scatter plot (15 points):**
    - X-axis: cost, Y-axis: revenue
    - Color by channel (5 colors)
    - Size by ROI (larger markers = higher ROI)
    - Title: “Marketing Campaign Performance: Cost vs Revenue by Channel”
    - Use `size='roi'` parameter with appropriate size range (size_max=30)
    - Add trendline: `trendline='ols'` (ordinary least squares regression)
    - Include campaign_id, segment, duration_days in hover_data
2. **Format hover information professionally (10 points):**
    - Create custom hover template showing:
        - Campaign ID (bold, larger)
        - Channel and Segment
        - Cost: formatted as “$15,234”
        - Revenue: formatted as “$45,678”
        - ROI: formatted as “234.5%”
        - Duration: “45 days”
        - Impressions: “123,456”
    - Remove the default trace info ()
    - Ensure hover text is readable and professional
3. **Add business intelligence features (5 points):**

- Add annotations for:
  - Best performing campaign (highest ROI)
  - Most expensive campaign
- Add reference lines:
  - Diagonal line showing break-even (revenue = cost)
  - Horizontal line showing average revenue
- In comments, document:
  - Which channel has best ROI?
  - Which segment is most profitable?
  - Any concerning patterns or outliers?
- Save as: `M4L01_YourName_CampaignROI.html` and `.png`

Part C: Product Category Sales Comparison (25 points)

Context: The Product team wants an interactive bar chart showing quarterly sales by category, allowing them to toggle between quarters and compare performance.

Data:

```python
np.random.seed(42)

categories = ['Electronics', 'Clothing', 'Home & Garden', 'Sports & Outdoors',
              'Books & Media', 'Toys & Games', 'Health & Beauty', 'Automotive']

quarters_list = []
for quarter in ['Q1', 'Q2', 'Q3', 'Q4']:
    for category in categories:
        # Different categories perform differently by quarter
        base_sales = np.random.uniform(500000, 2000000)

        # Q4 boost for certain categories
        if quarter == 'Q4' and category in ['Electronics', 'Toys & Games']:
            boost = 1.5
        elif quarter == 'Q1' and category in ['Health & Beauty', 'Sports & Outdoors']:
            boost = 1.3
        else:
            boost = 1.0

        quarters_list.append({
            'quarter': quarter,
            'category': category,
            'sales': base_sales * boost,
            'units_sold': int(base_sales / np.random.uniform(50, 200)),
            'profit_margin': np.random.uniform(0.15, 0.35)
        })

df_products = pd.DataFrame(quarters_list)
```

Tasks:

1. **Create grouped bar chart with animation (15 points):**
   - Use `px.bar()` with animation_frame='quarter'
   - X-axis: category, Y-axis: sales
   - Color by profit_margin (use 'RdYlGn' colorscale - red low, green high)
   - Sort categories by Q4 sales (descending)
   - Title: "Quarterly Product Category Performance"
   - Include units_sold and profit_margin in hover data
   - Format sales as currency, profit_margin as percentage
2. **Enhance animation controls (5 points):**
   - Make animation smooth (transition duration)

- Add play/pause button
- Show clear quarter label during animation
- Format hover to show:
  - Category name (bold)
  - Quarter
  - Sales: “$1.2M”
  - Units sold: “12,345 units”
  - Profit margin: “25.3%”

3. **Analysis and insights (5 points):**

   - In code comments, document:
     - Which categories benefit most from Q4 holiday season?
     - Which category has most consistent performance across quarters?
     - Which category has highest profit margin?
     - Recommendations for inventory planning
   - Save as: `M4L01_YourName_ProductQuarterly.html` and `.png`

Part D: Employee Satisfaction Multi-Dimensional Analysis (20 points)

Context: HR wants to explore employee satisfaction across departments, tenure, and locations using an interactive bubble chart.

Data:

```
np.random.seed(42)

departments = ['Engineering', 'Sales', 'Marketing', 'Operations', 'Finance', 'HR',
 ↪  'Customer Support']
locations = ['New York', 'San Francisco', 'Chicago', 'Austin', 'Remote']

employees_list = []
for dept in departments:
    n_employees = np.random.randint(25, 60)
    for _ in range(n_employees):
        location = np.random.choice(locations)
        tenure = np.random.uniform(0.5, 15)

        # Different departments have different satisfaction patterns
        if dept == 'Engineering':
            base_satisfaction = 4.2
        elif dept == 'Sales':
            base_satisfaction = 3.7
        elif dept == 'Customer Support':
            base_satisfaction = 3.5
        else:
            base_satisfaction = 4.0

        # Longer tenure slightly higher satisfaction
        tenure_effect = min(tenure * 0.05, 0.5)

        satisfaction = base_satisfaction + tenure_effect + np.random.normal(0, 0.4)
        satisfaction = np.clip(satisfaction, 1, 5)

        salary = np.random.normal(
            {'Engineering': 110000, 'Sales': 85000, 'Marketing': 80000,
             'Operations': 75000, 'Finance': 95000, 'HR': 78000,
             'Customer Support': 65000}[dept],
            15000
        )
        employees_list.append({
            'department': dept,
            'location': location,
            'tenure_years': tenure,
            'satisfaction_score': satisfaction,
            'salary': max(salary, 45000)
        })

df_employees = pd.DataFrame(employees_list)
```

Tasks:

1. **Create multi-dimensional bubble chart (12 points):**
    - Use `px.scatter()` with:
        - X-axis: tenure_years
        - Y-axis: satisfaction_score
        - Color: department
        - Size: salary (larger bubble = higher salary)
    - Title: "Employee Satisfaction Analysis: Tenure, Salary, and Department"
    - Add location to hover_data
    - Set size_max=20 for readable bubbles
    - Use 'Set2' color palette
2. **Interactive filtering capabilities (5 points):**
    - Enable legend click to filter departments
    - Format hover to show:
        - Department (bold)
        - Location
        - Tenure: "5.3 years"
        - Satisfaction: "4.2/5.0"
        - Salary: "$95,000"
    - Add reference lines:
        - Horizontal line at satisfaction = 4.0 (target)
        - Vertical line at tenure = 5.0 (mid-career)
3. **Insights documentation (3 points):**
    - In comments, answer:
        - Which department has lowest satisfaction?
        - Does longer tenure correlate with satisfaction?
        - Any patterns by location?
        - Recommendations for HR interventions
    - Save as: `M4L01_YourName_EmployeeSatisfaction.html` and `.png`

Bonus Challenge (+20 points):

Create an Interactive Dashboard with Dropdown Selectors:

Build a single interactive figure that allows users to: 1. Select which metric to display via dropdown (Sales, Profit, ROI) 2. Select time period via dropdown (Monthly, Quarterly, Yearly) 3. Toggle between chart types via buttons (Line, Bar, Area)

Requirements:

- Use `updatemenus` for dropdown controls
- Multiple traces that toggle visibility
- Professional styling
- Clear instructions text on the chart
- Save as: `M4L01_YourName_InteractiveDashboard.html`

Hint: Research `fig.update_layout(updatemenus=[...])` in Plotly documentation.

Deliverables:

1. Python file: `M4L01_YourName_Interactive.py`
2. HTML files (4-5):
   - `M4L01_YourName_SalesExplorer.html`
   - `M4L01_YourName_CampaignROI.html`
   - `M4L01_YourName_ProductQuarterly.html`
   - `M4L01_YourName_EmployeeSatisfaction.html`
   - `M4L01_YourName_InteractiveDashboard.html` (bonus)
3. PNG files (same names as HTML but .png extension)

Grading Rubric:

- Part A (Sales Explorer): 25 points
- Part B (Campaign ROI): 30 points
- Part C (Product Quarterly): 25 points
- Part D (Employee Satisfaction): 20 points
- Bonus (Interactive Dashboard): +20 points

Success Criteria:

- □ All charts are fully interactive in browser
- □ Hover information is clear and well-formatted
- □ Range sliders work on time series
- □ Legends are clickable for filtering
- □ Animations play smoothly (where applicable)
- □ Both HTML and PNG versions saved
- □ Business insights documented in comments
- □ Charts are professional and presentation-ready

Interactivity Checklist:

For each visualization, verify:

- □ Hover reveals all relevant information
- □ Zoom (click-drag) works properly
- □ Pan (shift-click-drag) functions
- □ Double-click resets view
- □ Legend items are clickable
- □ Numbers are properly formatted (currency, percentages)
- □ Color schemes are professional and accessible
- □ Tooltips don't obscure important data

Common Mistakes to Avoid:

- Not formatting currency with $ and commas
- Leaving default variable names in hover (use custom labels)
- Forgetting to set hovermode='x unified' for time series
- Not testing interactivity (zoom, pan, legend clicks)
- Poor color choices (too bright, not accessible)
- Missing axis labels or unclear titles
- Saving only PNG (loses interactivity) without HTML

Professional Tips:

- Test HTML files in browser before submitting
- Use `fig.show()` during development to preview
- Interactive charts should answer follow-up questions
- Include enough context in hover (but not too much)
- Consider your audience's technical level
- Embed explanatory text when needed
- Always provide both interactive (HTML) and static (PNG) versions

---

4.2 Section 2: Advanced Interactive Visualizations

4.2.1 Objective

- Create complex multi-trace visualizations combining different chart types
- Master faceted plots for multi-dimensional comparisons
- Implement custom buttons and dropdown menus for user control
- Create synchronized multi-plot dashboards
- Use animations to show changes over time effectively
- Build 3D visualizations for complex spatial data
- Apply advanced styling and theming for brand consistency

4.2.2 Main Contents with Examples

Multi-Trace Figures - Combining Different Visualizations

The Power of Layered Information:

Real business analysis often requires showing different data types on the same chart:

- Actual values + targets + forecasts
- Multiple metrics on same timeline
- Data + statistical summaries
- Comparisons with different chart types

Example: Sales with Target Lines and Average:

```python
import plotly.graph_objects as go
import pandas as pd
import numpy as np

# Generate monthly sales data
np.random.seed(42)
months = pd.date_range('2024-01-01', '2024-12-31', freq='M')
actual_sales = np.random.normal(100000, 15000, 12)
target_sales = np.full(12, 110000)
forecast = actual_sales * 1.1  # Next year forecast

# Create figure with multiple traces
fig = go.Figure()

# Trace 1: Actual sales (bar chart)
fig.add_trace(go.Bar(
    x=months,
    y=actual_sales,
    name='Actual Sales',
    marker_color='lightblue',
    hovertemplate='<b>Actual</b><br>%{x|%b %Y}<br>$%{y:,.0f}<extra></extra>'
))

# Trace 2: Target line
fig.add_trace(go.Scatter(
    x=months,
    y=target_sales,
    name='Target',
    line=dict(color='red', width=3, dash='dash'),
    hovertemplate='<b>Target</b><br>$%{y:,.0f}<extra></extra>'
))

# Trace 3: Forecast line
fig.add_trace(go.Scatter(
    x=months,
    y=forecast,
    name='2025 Forecast',
    line=dict(color='green', width=2, dash='dot'),
    hovertemplate='<b>Forecast</b><br>%{x|%b %Y}<br>$%{y:,.0f}<extra></extra>'
))

# Trace 4: Average line
avg_sales = np.mean(actual_sales)
fig.add_hline(y=avg_sales,
              line_dash='dash',
              line_color='gray',
              annotation_text=f'Average: ${avg_sales:,.0f}',
              annotation_position='right')

# Update layout
fig.update_layout(
    title='2024 Sales Performance: Actual vs Target with 2025 Forecast',
    xaxis_title='Month',
    yaxis_title='Sales ($)',
    yaxis_tickformat='$,.0f',
    hovermode='x unified',
    legend=dict(
        orientation='h',
        yanchor='bottom',
        y=1.02,
        xanchor='right',
        x=1
    ),
    height=600
)

fig.show()
```

Understanding Multi-Trace Construction:

Why Use Graph Objects Here:

- Mixing different chart types (bar + line)
- Need precise control over each trace
- Adding reference lines (hline)
- Complex hover templates

Key Techniques:

- `add_trace()`: Adds new data series
- `add_hline()` / `add_vline()`: Reference lines
- Each trace has independent styling
- `hovermode='x unified'`: Compare all traces at once

Business Value:

- Shows performance (actual)
- Shows expectations (target)
- Shows future (forecast)
- Shows context (average)
- All in ONE interactive chart

Faceted Plots - Small Multiples for Comparison

The Small Multiples Principle:

When comparing many groups, sometimes separate plots work better than overlaying: - Each group gets full space - Patterns within groups clearer - Easy to spot outliers per group - Still comparable (same scales)

Creating Faceted Plots with Plotly Express:

```
import plotly.express as px
import pandas as pd
import numpy as np

# Generate sales data for multiple products and regions
np.random.seed(42)
products = ['Product A', 'Product B', 'Product C', 'Product D']
regions = ['North', 'South', 'East', 'West']
months = pd.date_range('2024-01-01', '2024-12-31', freq='M')

data_list = []
for product in products:
    for region in regions:
        for month in months:
            # Different products have different patterns
            base = {'Product A': 50000, 'Product B': 35000,
                    'Product C': 45000, 'Product D': 30000}[product]

            # Regional variation
            regional_mult = {'North': 1.1, 'South': 0.9,
                           'East': 1.2, 'West': 1.0}[region]

            # Trend and noise
            month_num = month.month
            sales = base * regional_mult * (1 + 0.1 * np.sin(month_num/6 * np.pi)) +
↪   np.random.normal(0, 3000)

            data_list.append({
                'date': month,
                'product': product,
                'region': region,
                'sales': max(sales, 10000)
            })

df_sales = pd.DataFrame(data_list)

# Create faceted line chart
fig = px.line(df_sales,
              x='date',
              y='sales',
              color='region',
              facet_col='product',
              facet_col_wrap=2,  # 2 columns, multiple rows
              title='Sales Performance by Product and Region - 2024',
              labels={'sales': 'Monthly Sales ($)', 'date': 'Month'},
              height=800)

# Update all subplot titles
fig.for_each_annotation(lambda a: a.update(text=a.text.split('=')[1]))

# Format y-axes
fig.update_yaxes(tickformat='$,.0f')
```

```
# Improve facet spacing
fig.update_layout(
    showlegend=True,
    legend=dict(
        title='Region',
        orientation='v',
        yanchor='top',
        y=1,
        xanchor='left',
        x=1.02
    )
)

fig.show()
```

Faceting Parameters:

facet_col: Creates columns of subplots

- `facet_col='product'`: One column per product

facet_row: Creates rows of subplots

- `facet_row='region'`: One row per region

facet_col_wrap: Wraps columns to multiple rows

- `facet_col_wrap=2`: Maximum 2 columns, then wrap to next row
- Useful when you have many categories

When to Use Facets:

- Comparing 4+ groups
- Each group has its own pattern
- Want to avoid cluttered single plot
- Groups should be compared on same scale

Dropdown Menus and Buttons - User-Controlled Interactivity

Beyond Hover and Zoom:

Sometimes users need to change WHAT data is displayed, not just HOW it's viewed: - Switch between metrics (Revenue vs Profit vs Units) - Change time aggregation (Daily vs Monthly vs Quarterly) - Filter by category - Toggle between chart types

Creating Dropdown Menus:

```
import plotly.graph_objects as go
import pandas as pd
import numpy as np

# Generate data
np.random.seed(42)
months = pd.date_range('2024-01-01', '2024-12-31', freq='M')
revenue = np.random.normal(100000, 15000, 12)
profit = revenue * np.random.uniform(0.15, 0.30, 12)
units = revenue / np.random.uniform(45, 55, 12)

# Create figure with multiple traces (initially all visible)
fig = go.Figure()

# Revenue trace
fig.add_trace(go.Scatter(
    x=months,
    y=revenue,
    name='Revenue',
    line=dict(color='blue', width=3),
    visible=True
))

# Profit trace
fig.add_trace(go.Scatter(
    x=months,
    y=profit,
    name='Profit',
    line=dict(color='green', width=3),
    visible=False  # Initially hidden
))

# Units trace
fig.add_trace(go.Scatter(
    x=months,
    y=units,
    name='Units Sold',
    line=dict(color='orange', width=3),
    visible=False  # Initially hidden
))

# Create dropdown menu
fig.update_layout(
    updatemenus=[
        dict(
            buttons=list([
                dict(
                    args=[{'visible': [True, False, False]},
                          {'yaxis': {'title': 'Revenue ($)', 'tickformat': '$,.0f'}}],
                    label='Revenue',
                    method='update'
                ),
                dict(
                    args=[{'visible': [False, True, False]},
                          {'yaxis': {'title': 'Profit ($)', 'tickformat': '$,.0f'}}],
                    label='Profit',
                    method='update'
                ),
                dict(
                    args=[{'visible': [False, False, True]},
                          {'yaxis': {'title': 'Units Sold', 'tickformat': ',.0f'}}],
                    label='Units',
                    method='update'
                ),
            ]),
            direction='down',
            pad={'r': 10, 't': 10},
            showactive=True,
            x=0.11,
            xanchor='left',
            y=1.15,
            yanchor='top'
        ),
    ],
    title='Business Performance Dashboard - Select Metric',
    xaxis_title='Month',
    yaxis_title='Revenue ($)',
    yaxis_tickformat='$,.0f',
    height=600
)

fig.show()
```

Understanding Update Menus:

Structure:

```
updatemenus=[
    dict(
        buttons=[...],       # List of button definitions
        direction='down',    # Dropdown direction
        showactive=True,     # Highlight selected button
        x=0.11,             # Position (0-1 scale)
        y=1.15              # Position (0-1 scale)
    )
]
```

Button Definition:

```
dict(
    args=[{trace updates}, {layout updates}],
    label='Button Text',
    method='update'  # 'update' changes both traces and layout
)
```

args Structure:

- First dict: Trace properties (`{'visible': [True, False, False]}`)
- Second dict: Layout properties (`{'yaxis': {'title': 'New Title'}}`)

Business Application:

Executive dashboard where users select:

- What to view (Revenue, Profit, Costs)
- Appropriate Y-axis automatically updates
- Formatting changes based on metric
- One clean interface, multiple views

Animation - Showing Change Over Time

When Animation Enhances Understanding:

Animation is powerful when:

- Showing temporal evolution (year-over-year growth)
- Demonstrating process flows
- Revealing how relationships change
- Engaging presentation audiences

Creating Animated Scatter Plot:

```
import plotly.express as px
import pandas as pd
import numpy as np

# Generate data: Company metrics over 5 years
np.random.seed(42)
years = range(2020, 2025)
```

```
companies = ['TechCorp', 'InnovateLtd', 'DataSystems', 'CloudNet',
             'AIVentures', 'DevOps Inc', 'CyberGuard']

data_list = []
for year in years:
    for company in companies:
        # Each company grows differently
        year_factor = (year - 2020) + 1

        revenue = np.random.uniform(10, 100) * year_factor
        profit_margin = np.random.uniform(0.05, 0.30)
        employees = int(revenue * np.random.uniform(5, 15))

        data_list.append({
            'year': year,
            'company': company,
            'revenue': revenue,
            'profit_margin': profit_margin,
            'employees': employees
        })

df_companies = pd.DataFrame(data_list)

# Create animated scatter plot
fig = px.scatter(df_companies,
                 x='revenue',
                 y='profit_margin',
                 animation_frame='year',
                 animation_group='company',
                 size='employees',
                 color='company',
                 hover_name='company',
                 size_max=60,
                 range_x=[0, 500],
                 range_y=[0, 0.35],
                 title='Company Growth Animation: Revenue vs Profitability
 ↪  (2020-2024)',
                 labels={
                     'revenue': 'Revenue ($M)',
                     'profit_margin': 'Profit Margin',
                     'year': 'Year'
                 })

# Format hover
fig.update_traces(
    hovertemplate='<b>%{hovertext}</b><br>' +
                  'Revenue: $%{x:.1f}M<br>' +
                  'Profit Margin: %{y:.1%}<br>' +
                  '<extra></extra>'
)

# Improve animation controls
fig.layout.updatemenus[0].buttons[0].args[1]['frame']['duration'] = 1000
fig.layout.updatemenus[0].buttons[0].args[1]['transition']['duration'] = 500

fig.show()
```

Animation Parameters:

animation_frame: Variable that defines time steps

- Each unique value becomes a frame
- Example: `'year'` creates 5 frames (2020-2024)

animation_group: Variable that tracks objects across frames

- Keeps same company as same point through animation
- Enables smooth transitions

Fixed Ranges (Important!):

```
range_x=[0, 500],
range_y=[0, 0.35]
```

Without fixed ranges, axes rescale each frame (disorienting).

Animation Speed:

```
fig.layout.updatemenus[0].buttons[0].args[1]['frame']['duration'] = 1000  # 1 second per frame
fig.layout.updatemenus[0].buttons[0].args[1]['transition']['duration'] = 500  # 0.5s transition
```

When to Use Animation:

- Presenting to audience (engaging)
- Showing temporal progression (growth, decline)
- Demonstrating cause-effect with timing
- Limited when: User needs to compare non-sequential frames

3D Visualizations - Adding Another Dimension

Three-Dimensional Data:

Some business problems naturally have three continuous dimensions:

- Geographic data (latitude, longitude, elevation/value)
- Product data (price, quality, market share)
- Customer segments (age, income, spending)

Creating 3D Scatter Plot:

```
import plotly.express as px
import pandas as pd
import numpy as np

# Generate customer data with 3 dimensions
np.random.seed(42)
n_customers = 300

df_customers = pd.DataFrame({
    'customer_id': range(1, n_customers + 1),
    'age': np.random.normal(40, 15, n_customers),
    'annual_income': np.random.normal(65000, 25000, n_customers),
    'annual_spending': np.random.normal(25000, 10000, n_customers),
    'segment': np.random.choice(['Budget', 'Standard', 'Premium'], n_customers)
})

# Ensure reasonable values
df_customers['age'] = df_customers['age'].clip(18, 80)
df_customers['annual_income'] = df_customers['annual_income'].clip(20000, 200000)
df_customers['annual_spending'] = df_customers['annual_spending'].clip(5000, 80000)

# Create realistic relationships
```

```
df_customers['annual_spending'] = (
    5000 +
    0.2 * df_customers['annual_income'] +
    np.random.normal(0, 5000, n_customers)
)

# Segment based on behavior
df_customers['segment'] = pd.cut(
    df_customers['annual_spending'] / df_customers['annual_income'],
    bins=[0, 0.2, 0.4, 1.0],
    labels=['Budget', 'Standard', 'Premium']
)

# Create 3D scatter plot
fig = px.scatter_3d(df_customers,
                    x='age',
                    y='annual_income',
                    z='annual_spending',
                    color='segment',
                    size_max=10,
                    title='Customer Segmentation Analysis (3D)',
                    labels={
                        'age': 'Age (years)',
                        'annual_income': 'Annual Income ($)',
                        'annual_spending': 'Annual Spending ($)'
                    },
                    color_discrete_map={
                        'Budget': 'lightblue',
                        'Standard': 'lightgreen',
                        'Premium': 'gold'
                    })

# Format hover
fig.update_traces(
    hovertemplate='<b>Customer</b><br>' +
                  'Age: %{x:.0f}<br>' +
                  'Income: $%{y:,.0f}<br>' +
                  'Spending: $%{z:,.0f}<br>' +
                  '<extra></extra>'
)

# Update layout for better 3D viewing
fig.update_layout(
    scene=dict(
        xaxis_title='Age',
        yaxis_title='Income ($)',
        zaxis_title='Spending ($)',
        camera=dict(
            eye=dict(x=1.5, y=1.5, z=1.3)  # Viewing angle
        )
    ),
    height=700
)

fig.show()
```

3D Interactivity:

Users can:

- **Rotate**: Click and drag to rotate the 3D space
- **Zoom**: Scroll to zoom in/out
- **Pan**: Shift + drag to pan
- **Hover**: See values for all 3 dimensions

When to Use 3D:

Good use cases:

- Geographic data with a third metric (map with elevation)
- Truly three-dimensional relationships
- Impressive presentations (when appropriate)

Avoid 3D when:

- 2D would work (3D adds complexity)
- Precise reading of values needed (2D clearer)
- Audience isn't comfortable with 3D visualization
- Printing (3D doesn't work on paper)

Alternative: Multiple 2D views often better than 3D

Synchronized Subplots - Coordinated Multiple Views

The Coordinated View Concept:

Show multiple related views that interact together:

- Hover on one plot highlights related data in others
- Zoom on one plot adjusts others
- Selection in one filters others

Creating Synchronized Subplots:

```python
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pandas as pd
import numpy as np

# Generate product data
np.random.seed(42)
products = ['Product A', 'Product B', 'Product C', 'Product D', 'Product E']
metrics = []

for product in products:
    metrics.append({
        'product': product,
        'revenue': np.random.uniform(500000, 2000000),
        'profit': np.random.uniform(50000, 400000),
        'units_sold': np.random.randint(5000, 25000),
        'customer_satisfaction': np.random.uniform(3.5, 4.8),
        'return_rate': np.random.uniform(0.02, 0.12)
    })

df_products = pd.DataFrame(metrics)

# Create subplots with shared x-axis
fig = make_subplots(
    rows=3, cols=1,
    shared_xaxes=True,
    vertical_spacing=0.05,
    subplot_titles=('Revenue by Product', 'Customer Satisfaction', 'Return Rate')
)
```

```python
# Add revenue bars
fig.add_trace(
    go.Bar(x=df_products['product'], y=df_products['revenue'],
           name='Revenue', marker_color='lightblue',
           hovertemplate='%{x}<br>Revenue: $%{y:,.0f}<extra></extra>'),
    row=1, col=1
)

# Add satisfaction line
fig.add_trace(
    go.Scatter(x=df_products['product'], y=df_products['customer_satisfaction'],
               mode='lines+markers', name='Satisfaction',
               line=dict(color='green', width=3),
               hovertemplate='%{x}<br>Satisfaction: %{y:.2f}/5<extra></extra>'),
    row=2, col=1
)

# Add return rate bars
fig.add_trace(
    go.Bar(x=df_products['product'], y=df_products['return_rate'],
           name='Return Rate', marker_color='salmon',
           hovertemplate='%{x}<br>Returns: %{y:.1%}<extra></extra>'),
    row=3, col=1
)

# Update layout
fig.update_layout(
    title_text='Product Performance Dashboard - Synchronized Views',
    showlegend=True,
    height=900,
    hovermode='x unified'  # Synchronize hover across all subplots
)

# Update axes
fig.update_yaxes(title_text='Revenue ($)', row=1, col=1, tickformat='$,.0f')
fig.update_yaxes(title_text='Satisfaction (1-5)', row=2, col=1)
fig.update_yaxes(title_text='Return Rate', row=3, col=1, tickformat='.1%')
fig.update_xaxes(title_text='Product', row=3, col=1)

fig.show()
```

Key Synchronization Features:

shared_xaxes=True:

- All subplots share same X-axis
- Zoom on one subplot zooms all
- Maintains alignment

hovermode=‘x unified’:

- Hover on one subplot shows data from all
- Single vertical line across all plots
- Compare metrics simultaneously

Business Value:

Hover over “Product C”:

- See its revenue (top plot)
- See its satisfaction (middle plot)
- See its return rate (bottom plot)

- All at once - immediate insight into product health

Custom Styling and Themes

Beyond Default Aesthetics:

Professional applications require brand consistency:

- Company colors
- Corporate fonts
- Style guidelines
- Print/web variations

Creating a Custom Theme:

```python
import plotly.graph_objects as go
import plotly.express as px

# Define company theme
COMPANY_COLORS = {
    'primary': '#1F4788',    # Corporate blue
    'secondary': '#F39C12',  # Accent orange
    'success': '#27AE60',    # Green
    'danger': '#E74C3C',     # Red
    'background': '#F8F9FA', # Light gray
    'text': '#2C3E50'        # Dark blue-gray
}

COMPANY_TEMPLATE = go.layout.Template(
    layout=go.Layout(
        font=dict(
            family='Arial, sans-serif',
            size=12,
            color=COMPANY_COLORS['text']
        ),
        title=dict(
            font=dict(size=18, color=COMPANY_COLORS['text']),
            x=0.5,
            xanchor='center'
        ),
        plot_bgcolor=COMPANY_COLORS['background'],
        paper_bgcolor='white',
        colorway=[COMPANY_COLORS['primary'], COMPANY_COLORS['secondary'],
                  COMPANY_COLORS['success'], COMPANY_COLORS['danger']],
        xaxis=dict(
            showgrid=True,
            gridcolor='white',
            showline=True,
            linecolor=COMPANY_COLORS['text'],
            linewidth=2
        ),
        yaxis=dict(
            showgrid=True,
            gridcolor='white',
            showline=True,
            linecolor=COMPANY_COLORS['text'],
            linewidth=2
        )
    )
)
```

```
# Use the custom template
fig = px.bar(df, x='category', y='sales')
fig.update_layout(template=COMPANY_TEMPLATE)
fig.show()
```

Applying Theme Globally:

```
import plotly.io as pio

# Set as default template
pio.templates['company_theme'] = COMPANY_TEMPLATE
pio.templates.default = 'company_theme'

# Now all plots use this theme automatically
fig = px.scatter(df, x='x', y='y')
fig.show()  # Automatically uses company theme
```

4.2.3 Lab Session

Lab 2: Advanced Interactive Visualizations

Objective: Master advanced Plotly techniques including multi-trace figures, animations, user controls, and 3D visualizations to create sophisticated, business-ready interactive dashboards.

Scenario: You're the Lead Data Visualization Engineer at "GlobalInsights Analytics," working for a Fortune 500 client launching a new product line. The executive committee needs advanced interactive visualizations for their quarterly board presentation. Standard charts won't suffice - they need visualizations that tell stories, respond to questions, and impress stakeholders. Your mission is to create cutting-edge interactive visualizations that showcase data sophistication.

Pre-Lab Setup:

1. Create file: `M4L02_YourName_Advanced.py`
2. Import libraries:

```
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
import pandas as pd
import numpy as np
```

3. Create output folder: "Lab2_Outputs"

Part A: Multi-Metric Performance Dashboard with Dropdown Controls (30 points)

Context: The CFO needs one dashboard that can switch between viewing Revenue, Profit, Operating Costs, and ROI - all with proper formatting and context for each metric.

Data:

```
np.random.seed(42)

months = pd.date_range('2024-01-01', '2024-12-31', freq='M')

df_financial = pd.DataFrame({
    'month': months,
    'revenue': np.random.uniform(800000, 1200000, 12) + np.arange(12) * 10000,
    'costs': np.random.uniform(500000, 700000, 12) + np.arange(12) * 5000,
    'marketing': np.random.uniform(50000, 100000, 12),
    'r_and_d': np.random.uniform(80000, 150000, 12)
})
```

```
df_financial['profit'] = df_financial['revenue'] - df_financial['costs']
df_financial['operating_costs'] = df_financial['marketing'] + df_financial['r_and_d']
df_financial['roi'] = (df_financial['profit'] / df_financial['operating_costs']) * 100
```

Tasks:

1. **Create multi-trace figure with dropdown selector (20 points):**
   - Build using Graph Objects (not Express - need fine control)
   - Create 4 separate traces (Revenue, Profit, Operating Costs, ROI)
   - Initially show only Revenue (others hidden)
   - Add dropdown menu with 4 options to switch between metrics
   - Each option should:
     - Update trace visibility
     - Update Y-axis title
     - Update Y-axis format (currency for $, percentage for ROI)
     - Update colors appropriately (green for profit, red for costs, etc.)
   - Title: “Financial Performance Dashboard - Select Metric”
2. **Add target lines and annotations (5 points):**
   - Revenue target: $1,000,000 (horizontal dashed line)
   - Profit target: $400,000
   - ROI target: 150%
   - Target lines should update visibility with dropdown
   - Add annotation showing YTD average for displayed metric
3. **Professional styling (5 points):**
   - Dropdown positioned top-left, clearly labeled
   - Hover mode: ‘x unified’
   - Grid lines for readability
   - Legend showing what’s displayed
   - Height: 700 pixels
   - Save as: `M4L02_YourName_FinancialDashboard.html`

Part B: Animated Market Share Evolution (25 points)

Context: Marketing needs to show how market share evolved quarter-over-quarter for top 8 competitors, with animation revealing the competitive dynamics.

Data:

```
np.random.seed(42)

quarters = ['Q1 2023', 'Q2 2023', 'Q3 2023', 'Q4 2023',
            'Q1 2024', 'Q2 2024', 'Q3 2024', 'Q4 2024']
companies = ['Our Company', 'Competitor A', 'Competitor B', 'Competitor C',
             'Competitor D', 'Competitor E', 'Competitor F', 'Others']

market_data = []
for i, quarter in enumerate(quarters):
    # Each company's market share evolves
    shares = {
        'Our Company': 22 + i * 1.5 + np.random.uniform(-1, 1),
        'Competitor A': 18 - i * 0.3 + np.random.uniform(-1, 1),
        'Competitor B': 15 + i * 0.5 + np.random.uniform(-1, 1),
        'Competitor C': 12 - i * 0.2 + np.random.uniform(-1, 1),
        'Competitor D': 11 + i * 0.3 + np.random.uniform(-1, 1),
        'Competitor E': 9 - i * 0.4 + np.random.uniform(-1, 1),
        'Competitor F': 7 + i * 0.1 + np.random.uniform(-1, 1),
    }
    # Others is remainder
    shares['Others'] = 100 - sum(shares.values())

    for company, share in shares.items():
        market_data.append({
            'quarter': quarter,
            'company': company,
            'market_share': max(share, 1)  # Ensure positive
        })

df_market = pd.DataFrame(market_data)
```

Tasks:

1. **Create animated bar chart race (15 points):**
   - Use animated horizontal bar chart
   - Animation frame: quarter
   - X-axis: market_share
   - Y-axis: company (sorted by share each frame)
   - Color by company (consistent colors throughout)
   - Fixed X-axis range: [0, 35]
   - Title: "Market Share Evolution: Quarterly Competitive Analysis"
   - Make "Our Company" stand out (different color, bold in legend)
2. **Optimize animation quality (5 points):**
   - Smooth transitions (500ms transition time)
   - Appropriate frame duration (1500ms)
   - Show current quarter prominently
   - Add annotations showing:
     - "Our Company" growth rate
     - Total market leader
   - Format as percentages with 1 decimal
3. **Business insights overlay (5 points):**
   - Add text box showing key insights:
     - Our Company's share in first vs last quarter
     - Biggest competitor threat
     - Market consolidation trend
   - Save as: `M4L02_YourName_MarketShareAnimation.html`
   - In comments, document the competitive story the animation tells

Part C: 3D Product Portfolio Analysis (25 points)

Context: The product team needs to visualize their portfolio across three dimensions: Price, Quality Score, and Market Demand, with segments identified.

Data:

```
np.random.seed(42)

n_products = 50

df_portfolio = pd.DataFrame({
    'product_id': [f'SKU-{i:03d}' for i in range(1, n_products + 1)],
    'price': np.random.uniform(20, 500, n_products),
    'quality_score': np.random.uniform(60, 100, n_products),
    'market_demand': np.random.uniform(1000, 50000, n_products),
    'profit_margin': np.random.uniform(0.10, 0.40, n_products),
    'category': np.random.choice(['Electronics', 'Accessories', 'Software'],
    ↪  n_products)
})

# Create natural relationships
df_portfolio['market_demand'] = df_portfolio['market_demand'] * (
    1 + (100 - df_portfolio['price']) / 100
)  # Lower price = higher demand

df_portfolio['profit_margin'] = df_portfolio['profit_margin'] * (
    1 + (df_portfolio['quality_score'] - 60) / 100
)  # Higher quality = higher margin

# Strategic segments
df_portfolio['strategic_segment'] = 'Standard'
df_portfolio.loc[(df_portfolio['quality_score'] > 85) &
                 (df_portfolio['profit_margin'] > 0.25), 'strategic_segment'] =
 ↪  'Premium Stars'
df_portfolio.loc[(df_portfolio['market_demand'] > 30000) &
                 (df_portfolio['profit_margin'] > 0.20), 'strategic_segment'] =
 ↪  'Volume Winners'
df_portfolio.loc[(df_portfolio['quality_score'] < 70) |
                 (df_portfolio['profit_margin'] < 0.15), 'strategic_segment'] =
 ↪  'Review Needed'
```

Tasks:

1. **Create interactive 3D scatter plot (15 points):**
   - X-axis: price
   - Y-axis: quality_score
   - Z-axis: market_demand
   - Color by strategic_segment
   - Size by profit_margin
   - Include product_id and category in hover
   - Title: "Product Portfolio Analysis: Price-Quality-Demand Matrix (3D)"
   - Custom colors:
     - Premium Stars: gold
     - Volume Winners: green
     - Standard: lightblue
     - Review Needed: red
2. **Enhance 3D visualization (5 points):**
   - Set optimal camera angle for presentation
   - Add axis labels with units
   - Format hover to show:
     - Product ID (bold)
     - Category
     - Price: "$XXX.XX"
     - Quality: "XX/100"
     - Demand: "XX,XXX units"
     - Profit Margin: "XX.X%"
   - Size range: 5-20 (readable but not overwhelming)
3. **Strategic insights (5 points):**
   - In code comments, identify:
     - How many "Premium Stars" products?
     - Characteristics of "Volume Winners"
     - Which products need review and why?
     - Recommendation: Where should R&D focus?
   - Create 2D projection (Price vs Quality colored by segment) for comparison

- Save both: M4L02_YourName_Portfolio3D.html and M4L02_YourName_Portfolio2D.html

Part D: Synchronized Multi-View Sales Dashboard (20 points)

Context: Regional managers need a dashboard showing sales performance, with synchronized views of different metrics that highlight together.

Data:

```
np.random.seed(42)

regions = ['North', 'South', 'East', 'West', 'Central']

df_regional = pd.DataFrame({
    'region': regions,
    'q1_sales': [450000, 380000, 520000, 410000, 390000],
    'q2_sales': [470000, 390000, 550000, 430000, 405000],
    'q3_sales': [490000, 405000, 570000, 445000, 420000],
    'q4_sales': [580000, 480000, 650000, 520000, 495000],
    'customer_count': [2500, 2100, 2900, 2300, 2200],
    'avg_deal_size': [800, 750, 900, 850, 775],
    'satisfaction': [4.3, 4.1, 4.5, 4.2, 4.0]
})

df_regional['total_sales'] = (df_regional['q1_sales'] + df_regional['q2_sales'] +
                              df_regional['q3_sales'] + df_regional['q4_sales'])
df_regional['growth_rate'] = (df_regional['q4_sales'] - df_regional['q1_sales']) /
↪   df_regional['q1_sales']
```

Tasks:

1. **Create synchronized 4-panel dashboard (15 points):**
   - Use make_subplots() with 2 rows, 2 columns
   - Panel 1 (top-left): Total sales by region (bar chart)
   - Panel 2 (top-right): Growth rate by region (horizontal bar)
   - Panel 3 (bottom-left): Customer count vs satisfaction (scatter)
   - Panel 4 (bottom-right): Quarterly trend (line chart, all regions)
   - Shared hover: hovering over a region in any panel highlights it everywhere
   - Title: "Regional Sales Performance - Synchronized Dashboard"
2. **Implement interactivity (3 points):**
   - Configure hovermode='x unified' where appropriate
   - Make panels share color coding (North always same color)
   - Enable legend click to filter
   - Format all currency as "$XXX,XXX"
   - Format percentages properly
3. **Layout optimization (2 points):**
   - Appropriate spacing between subplots
   - Clear subplot titles
   - Overall figure height: 900 pixels
   - All text readable
   - Save as: M4L02_YourName_SynchronizedDashboard.html

Bonus Challenge (+25 points):

Create an Executive Decision Support Tool:

Build an advanced interactive dashboard combining multiple techniques:

1. **Main view**: Animated scatter plot showing company performance over time
   - X: Revenue, Y: Profit Margin, Size: Employees, Color: Division
   - Animation: Quarterly progression

2. **Control panel**: Buttons and dropdowns for:
    - Metric selection (Revenue/Profit/ROI/Customer Sat)
    - Time period (Quarterly/Monthly/YTD)
    - Division filter (All/Sales/Engineering/Operations)
    - Chart type toggle (Scatter/Line/Bar)
3. **Side panel**: Synchronized KPI cards showing:
    - Selected metric's current value
    - Change from previous period
    - Rank among divisions
    - Projected next quarter (simple trend)
4. **Interactive features**:
    - Click a division to isolate it
    - Hover shows detailed drill-down
    - Export selected view as PNG
    - Reset button to default view

Requirements:

- Use Graph Objects for complete control
- Professional corporate styling
- Smooth animations
- Responsive layout
- Comprehensive hover information
- Save as: `M4L02_YourName_ExecutiveTool.html`

Deliverables:

1. Python file: `M4L02_YourName_Advanced.py`
2. HTML files (5-6):
    - `M4L02_YourName_FinancialDashboard.html`
    - `M4L02_YourName_MarketShareAnimation.html`
    - `M4L02_YourName_Portfolio3D.html`
    - `M4L02_YourName_Portfolio2D.html`
    - `M4L02_YourName_SynchronizedDashboard.html`
    - `M4L02_YourName_ExecutiveTool.html` (bonus)

Grading Rubric:

- Part A (Dropdown Dashboard): 30 points
- Part B (Market Share Animation): 25 points
- Part C (3D Portfolio): 25 points
- Part D (Synchronized Dashboard): 20 points
- Bonus (Executive Tool): +25 points

Success Criteria:

☐ Dropdown menus work smoothly
☐ Animations play without glitches
☐ 3D plot is interactive (rotate, zoom)
☐ Synchronized hover works across subplots
☐ All formatting (currency, %) correct
☐ Professional appearance
☐ Business insights documented
☐ HTML files open and function in browser

Advanced Techniques Checklist:

☐ updatemenus configured correctly
☐ Animation frame and group parameters set
☐ 3D camera positioned for optimal viewing
☐ Subplots synchronized with shared axes/hover
☐ Custom color schemes applied
☐ Hover templates formatted professionally
☐ Appropriate chart types for each data story

□ Interactivity tested (clicks, hovers, zooms)

Common Mistakes to Avoid:

- Dropdown buttons that don't update properly
- Animation with unfixed axis ranges (disorienting)
- 3D plots with default camera angle (poor view)
- Unsynchronized subplots (defeating the purpose)
- Inconsistent color schemes across views
- Missing hover information
- Slow-loading animations (too many data points)

4.3 Section 3: Creating Dashboards and Business Reports

4.3.1 Objective

- Design comprehensive multi-chart dashboards using Plotly
- Master subplot layouts for professional dashboard creation
- Create KPI cards and metric displays
- Build narrative-driven business reports with visualizations
- Export dashboards for stakeholder distribution
- Implement responsive design principles for different screen sizes
- Integrate text, images, and branding into visual reports

4.3.2 Main Contents with Examples

The Anatomy of an Effective Business Dashboard

What Makes a Dashboard Effective:

A dashboard is not just multiple charts on one page. An effective business dashboard has:

1. Clear Hierarchy:

- Most important metrics prominently displayed (top-left, largest)
- Supporting details in secondary positions
- Visual flow guides the eye through insights

2. Context and Comparison:

- Current values with historical comparison
- Targets/benchmarks clearly shown
- Trend indicators (↑↓) immediately visible

3. Actionability:

- Red/green indicators for performance
- Drill-down capabilities where needed
- Clear calls-to-action implied

4. Appropriate Density:

- Not too sparse (wasted space)
- Not too cluttered (overwhelming)
- ~4-8 key visualizations optimal

Dashboard Design Principles:

The 5-Second Rule:

- Key message understandable in 5 seconds
- Detailed exploration available for those who need it
- Progressive disclosure of information

The "So What?" Test:

- Every chart answers a specific business question

- Stakeholder should immediately know: “So what should I do?”
- No chart exists just to fill space

Building a Multi-Panel Dashboard Layout

Strategic Layout Patterns:

```python
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pandas as pd
import numpy as np

# Generate comprehensive business data
np.random.seed(42)
months = pd.date_range('2024-01-01', '2024-12-31', freq='M')

df_business = pd.DataFrame({
    'month': months,
    'revenue': np.random.uniform(800000, 1200000, 12) + np.arange(12) * 20000,
    'costs': np.random.uniform(500000, 700000, 12) + np.arange(12) * 10000,
    'new_customers': np.random.randint(150, 300, 12),
    'churn_rate': np.random.uniform(0.02, 0.08, 12)
})

df_business['profit'] = df_business['revenue'] - df_business['costs']
df_business['profit_margin'] = df_business['profit'] / df_business['revenue']

# Regional breakdown
regions = ['North', 'South', 'East', 'West']
regional_sales = [3200000, 2800000, 3500000, 2900000]

# Product categories
categories = ['Product A', 'Product B', 'Product C', 'Product D']
category_revenue = [4200000, 3500000, 2800000, 2100000]

# Create dashboard layout: 2x2 grid
fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=('Monthly Revenue Trend',
                    'Regional Sales Distribution',
                    'Profit Margin Evolution',
                    'Product Revenue Breakdown'),
    specs=[[{'type': 'scatter'}, {'type': 'bar'}],
           [{'type': 'scatter'}, {'type': 'pie'}]],
    vertical_spacing=0.12,
    horizontal_spacing=0.1
)

# Chart 1: Revenue trend line (top-left)
fig.add_trace(
    go.Scatter(
        x=df_business['month'],
        y=df_business['revenue'],
        mode='lines+markers',
        name='Revenue',
        line=dict(color='#2E86AB', width=3),
        marker=dict(size=8),
        hovertemplate='<b>Revenue</b><br>%{x|%b %Y}<br>$%{y:,.0f}<extra></extra>'
    ),
    row=1, col=1
```

```
)

# Add target line
fig.add_hline(y=1000000, line_dash='dash', line_color='red',
              annotation_text='Target: $1M',
              annotation_position='right',
              row=1, col=1)

# Chart 2: Regional sales bars (top-right)
fig.add_trace(
    go.Bar(
        x=regions,
        y=regional_sales,
        name='Regional Sales',
        marker_color=['#FF6B6B', '#4ECDC4', '#45B7D1', '#FFA07A'],
        text=[f'${v/1e6:.1f}M' for v in regional_sales],
        textposition='outside',
        hovertemplate='<b>%{x}</b><br>$%{y:,.0f}<extra></extra>'
    ),
    row=1, col=2
)

# Chart 3: Profit margin line (bottom-left)
fig.add_trace(
    go.Scatter(
        x=df_business['month'],
        y=df_business['profit_margin'],
        mode='lines+markers',
        name='Profit Margin',
        line=dict(color='#06A77D', width=3),
        fill='tozeroy',
        fillcolor='rgba(6, 167, 125, 0.2)',
        hovertemplate='<b>Profit Margin</b><br>%{x|%b %Y}<br>%{y:.1%}<extra></extra>'
    ),
    row=2, col=1
)

# Chart 4: Product revenue pie (bottom-right)
fig.add_trace(
    go.Pie(
        labels=categories,
        values=category_revenue,
        name='Products',

 ↪  hovertemplate='<b>%{label}</b><br>$%{value:,.0f}<br>%{percent}<extra></extra>',
        marker=dict(colors=['#FF6B6B', '#4ECDC4', '#45B7D1', '#FFA07A'])
    ),
    row=2, col=2
)

# Update layout
fig.update_layout(
    title_text='<b>Business Performance Dashboard - 2024 Overview</b>',
    title_font_size=20,
    title_x=0.5,
    showlegend=False,
    height=800,
    plot_bgcolor='#F8F9FA',
    paper_bgcolor='white'
)

# Format axes
fig.update_xaxes(showgrid=True, gridcolor='white', row=1, col=1)
fig.update_yaxes(title_text='Revenue ($)', tickformat='$,.0f', row=1, col=1)
fig.update_yaxes(title_text='Sales ($)', tickformat='$,.0f', row=1, col=2)
fig.update_yaxes(title_text='Margin', tickformat='.0%', row=2, col=1)

fig.show()
```

Layout Strategy Explained:

Top Row (Strategic Overview):

- **Top-left**: Primary metric (Revenue) - largest, most important
- **Top-right**: Comparison metric (Regional breakdown)

Bottom Row (Supporting Details):

- **Bottom-left**: Efficiency metric (Profit margin)
- **Bottom-right**: Composition (Product mix)

Visual Flow: Top → Bottom, Left → Right (Western reading pattern)

Creating KPI Cards and Metric Displays

KPI Cards: The Dashboard Headlines

Key Performance Indicators should be immediately visible, large, and comparative.

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Calculate KPIs
current_month_revenue = 1180000
previous_month_revenue = 1100000
revenue_change = ((current_month_revenue - previous_month_revenue) /
                  previous_month_revenue)

current_customers = 4850
previous_customers = 4720
customer_change = ((current_customers - previous_customers) /
                   previous_customers)

current_margin = 0.28
previous_margin = 0.26
margin_change = current_margin - previous_margin

# Create figure with annotation-based KPI cards
fig = go.Figure()

# Add invisible trace to set up the canvas
fig.add_trace(go.Scatter(
    x=[0, 1],
    y=[0, 1],
    mode='markers',
    marker=dict(size=0.1, color='white'),
    showlegend=False,
    hoverinfo='none'
))
```

```
# KPI Card 1: Revenue
fig.add_annotation(
    x=0.15, y=0.85,
    xref='paper', yref='paper',
    text=f'<b>Monthly Revenue</b><br>' +
         f'<span style="font-size:32px;
          ↪  color:#2E86AB">${current_month_revenue:,.0f}</span><br>' +
         f'<span style="color:{"green" if revenue_change > 0 else "red"}">{"↑" if
          ↪  revenue_change > 0 else "↓"} {revenue_change:.1%} vs last month</span>',
    showarrow=False,
    bgcolor='#F8F9FA',
    bordercolor='#2E86AB',
    borderwidth=2,
    borderpad=20,
    font=dict(size=12),
    align='center'
)

# KPI Card 2: Customers
fig.add_annotation(
    x=0.50, y=0.85,
    xref='paper', yref='paper',
    text=f'<b>Active Customers</b><br>' +
         f'<span style="font-size:32px;
          ↪  color:#06A77D">{current_customers:,}</span><br>' +
         f'<span style="color:{"green" if customer_change > 0 else "red"}">{"↑" if
          ↪  customer_change > 0 else "↓"} {customer_change:.1%} vs last
          ↪  month</span>',
    showarrow=False,
    bgcolor='#F8F9FA',
    bordercolor='#06A77D',
    borderwidth=2,
    borderpad=20,
    font=dict(size=12),
    align='center'
)

# KPI Card 3: Profit Margin
fig.add_annotation(
    x=0.85, y=0.85,
    xref='paper', yref='paper',
    text=f'<b>Profit Margin</b><br>' +
         f'<span style="font-size:32px;
          ↪  color:#FFA07A">{current_margin:.1%}</span><br>' +
         f'<span style="color:{"green" if margin_change > 0 else "red"}">{"↑" if
          ↪  margin_change > 0 else "↓"} {abs(margin_change):.1%} vs last
          ↪  month</span>',
    showarrow=False,
    bgcolor='#F8F9FA',
    bordercolor='#FFA07A',
    borderwidth=2,
    borderpad=20,
    font=dict(size=12),
    align='center'
)

# Update layout
fig.update_layout(
    title='<b>Key Performance Indicators - December 2024</b>',
    title_x=0.5,
    title_font_size=20,
    xaxis=dict(visible=False),
    yaxis=dict(visible=False),
    plot_bgcolor='white',
    paper_bgcolor='white',
    height=300,
    margin=dict(l=20, r=20, t=80, b=20)
)

fig.show()
```

KPI Card Best Practices:

1. Structure:

- Label (what is it)
- Big number (current value)
- Comparison (vs target/previous period)
- Trend indicator (↑↓)

2. Color Coding:

- Green: Good performance, improvement
- Red: Poor performance, decline
- Gray/Blue: Neutral, informational

3. Positioning:

- Top of dashboard
- Most important KPI leftmost
- 3-5 KPIs maximum (avoid overload)

Building a Complete Executive Dashboard

Integrated Dashboard with All Elements:

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pandas as pd
import numpy as np

# Generate data
np.random.seed(42)
months = pd.date_range('2024-01-01', '2024-12-31', freq='M')

df = pd.DataFrame({
    'month': months,
    'revenue': np.cumsum(np.random.uniform(80000, 120000, 12)),
    'customers': np.cumsum(np.random.randint(50, 150, 12)),
    'satisfaction': np.random.uniform(4.0, 4.8, 12)
})

# Create complex dashboard
fig = make_subplots(
    rows=4, cols=3,
    row_heights=[0.15, 0.35, 0.35, 0.15],
    column_widths=[0.33, 0.33, 0.34],
    specs=[
        [{'type': 'indicator'}, {'type': 'indicator'}, {'type': 'indicator'}],
        [{'type': 'scatter', 'colspan': 2}, None, {'type': 'bar'}],
        [{'type': 'scatter', 'colspan': 2}, None, {'type': 'pie'}],
        [{'type': 'table', 'colspan': 3}, None, None]
    ],
    subplot_titles=['', '', '',
                    'Revenue Growth Trend', '', 'Top Products',
                    'Customer Acquisition', '', 'Channel Mix',
                    ''],
    vertical_spacing=0.08,
    horizontal_spacing=0.05
)

# Row 1: KPI Indicators
fig.add_trace(go.Indicator(
    mode='number+delta',
    value=df['revenue'].iloc[-1],
    delta={'reference': df['revenue'].iloc[-2], 'relative': True, 'valueformat':
 ↪  '.1%'},
    title={'text': 'Monthly Revenue'},
    number={'prefix': '$', 'valueformat': ',.0f'},
    domain={'x': [0, 1], 'y': [0, 1]}
), row=1, col=1)

fig.add_trace(go.Indicator(
    mode='number+delta',
    value=df['customers'].iloc[-1],
    delta={'reference': df['customers'].iloc[-2], 'relative': True, 'valueformat':
 ↪  '.1%'},
    title={'text': 'Total Customers'},
    number={'valueformat': ','},
    domain={'x': [0, 1], 'y': [0, 1]}
), row=1, col=2)

fig.add_trace(go.Indicator(
    mode='gauge+number+delta',
    value=df['satisfaction'].iloc[-1],
    delta={'reference': 4.0},
    gauge={'axis': {'range': [1, 5]},
           'bar': {'color': '#06A77D'},
           'threshold': {'line': {'color': 'red', 'width': 4},
                         'thickness': 0.75, 'value': 4.0}},
    title={'text': 'Satisfaction'},
    domain={'x': [0, 1], 'y': [0, 1]}
), row=1, col=3)

# Row 2: Revenue trend and Top Products
fig.add_trace(go.Scatter(
    x=df['month'], y=df['revenue'],
    mode='lines+markers',
    line=dict(color='#2E86AB', width=3),
    fill='tozeroy',
    fillcolor='rgba(46, 134, 171, 0.2)',
    name='Revenue'
), row=2, col=1)

products = ['Product A', 'Product B', 'Product C', 'Product D']
product_sales = [450000, 380000, 320000, 280000]
fig.add_trace(go.Bar(
    x=product_sales, y=products,
    orientation='h',
    marker_color='#FF6B6B',
    name='Sales'
), row=2, col=3)

# Row 3: Customer acquisition and Channel mix
fig.add_trace(go.Scatter(
    x=df['month'], y=df['customers'],
    mode='lines+markers',
    line=dict(color='#06A77D', width=3),
    name='Customers'
), row=3, col=1)

channels = ['Direct', 'Partner', 'Online', 'Reseller']
channel_revenue = [35, 28, 22, 15]
fig.add_trace(go.Pie(
    labels=channels, values=channel_revenue,
    marker_colors=['#FF6B6B', '#4ECDC4', '#45B7D1', '#FFA07A'],
    name='Channels'
), row=3, col=3)

# Row 4: Summary table
fig.add_trace(go.Table(
    header=dict(values=['<b>Metric</b>', '<b>Current</b>', '<b>Target</b>',
 ↪  '<b>Status</b>'],
                fill_color='#2E86AB',
                font=dict(color='white', size=12),
                align='left'),
    cells=dict(values=[
        ['Revenue', 'Customers', 'Profit Margin', 'Satisfaction'],
        ['$1.18M', '4,850', '28%', '4.5/5'],
        ['$1.10M', '4,500', '25%', '4.0/5'],
        ['  Exceeds', '  Exceeds', '  Exceeds', '  Exceeds']
    ],
    fill_color=[['#F8F9FA', 'white']*4],
    font=dict(size=11),
    align='left')
), row=4, col=1)

# Update layout
fig.update_layout(
    title_text='<b>Executive Dashboard - December 2024</b><br>' +
               '<sub>Key metrics and performance indicators</sub>',
    title_x=0.5,
    title_font_size=22,
    showlegend=False,
    height=1200,
    plot_bgcolor='#F8F9FA',
    paper_bgcolor='white'
)

# Format axes
fig.update_xaxes(showgrid=True, gridcolor='white')
fig.update_yaxes(showgrid=True, gridcolor='white')

fig.show()
```

Dashboard Features:

1. **Top Row - KPIs:**
    - Immediate performance snapshot
    - Comparison to previous period
    - Color-coded delta indicators
2. **Middle Rows - Details:**
    - Trends over time
    - Breakdowns by category
    - Mix of chart types for variety
3. **Bottom Row - Summary:**
    - Tabular data for precise values
    - Status indicators
    - Quick reference

Creating Narrative-Driven Reports

Beyond Dashboards: Analytical Reports

Reports tell a story with data, guiding readers through analysis:

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Create narrative report structure
fig = make_subplots(
    rows=5, cols=1,
    row_heights=[0.1, 0.25, 0.25, 0.25, 0.15],
    specs=[
        [{'type': 'table'}],
        [{'type': 'scatter'}],
        [{'type': 'bar'}],
        [{'type': 'scatter'}],
        [{'type': 'table'}]
    ],
    subplot_titles=['',
                    '1. Revenue Shows Strong Growth Throughout 2024',
                    '2. Regional Performance Varies Significantly',
                    '3. Customer Satisfaction Remains High Despite Growth',
                    ''],
    vertical_spacing=0.08
)

# Executive Summary (top)
fig.add_trace(go.Table(
    header=dict(values=['<b>Executive Summary: 2024 Performance Review</b>'],
                fill_color='#2E86AB',
                font=dict(color='white', size=14),
                height=40),
    cells=dict(values=[
        ['Our analysis reveals strong growth in 2024 with revenue increasing 45% YoY.
 ↪   ' +
         'Regional performance shows East region leading growth at 52%, while South
          ↪  lags at 28%. ' +
         'Customer satisfaction remained consistently above 4.5/5 despite rapid
          ↪  scaling. ' +
         'Recommendations: Invest in South region training, maintain quality focus
          ↪  during growth.']
    ],
    fill_color='#F8F9FA',
    font=dict(size=12),
    height=30,
    align='left')
), row=1, col=1)

# Chart 1: Revenue trend with annotations
months = pd.date_range('2024-01-01', '2024-12-31', freq='M')
revenue = np.cumsum(np.random.uniform(80000, 120000, 12))

fig.add_trace(go.Scatter(
    x=months, y=revenue,
    mode='lines+markers',
    line=dict(color='#2E86AB', width=3),
    name='Revenue'
), row=2, col=1)

# Add annotation for key insight
fig.add_annotation(
    x=months[8], y=revenue[8],
    text='Q3 surge due to<br>new product launch',
    showarrow=True,
    arrowhead=2,
    ax=-50, ay=-50,
    row=2, col=1
)

# Chart 2: Regional comparison
regions = ['North', 'South', 'East', 'West']
growth = [42, 28, 52, 38]
fig.add_trace(go.Bar(
    x=regions, y=growth,
    marker_color=['#06A77D' if g > 40 else '#FFA07A' for g in growth],
    text=[f'{g}%' for g in growth],
    textposition='outside'
), row=3, col=1)

# Chart 3: Satisfaction trend
satisfaction = np.random.uniform(4.3, 4.7, 12)
fig.add_trace(go.Scatter(
    x=months, y=satisfaction,
    mode='lines+markers',
    line=dict(color='#06A77D', width=3)
), row=4, col=1)

# Add target line
fig.add_hline(y=4.0, line_dash='dash', line_color='red',
              annotation_text='Minimum Target',
              row=4, col=1)

# Recommendations (bottom)
fig.add_trace(go.Table(
    header=dict(values=['<b>Strategic Recommendations</b>'],
                fill_color='#06A77D',
                font=dict(color='white', size=14)),
    cells=dict(values=[
        ['1. South Region: Deploy best practices from East region, provide additional
 ↪  training<br>' +
         '2. Scaling: Maintain quality controls as customer base grows, hire support
          ↪  staff<br>' +
        '3. Products: Capitalize on Q3 success, plan Q2 2025 launch<br>' +
        '4. Investment: Allocate additional budget to high-growth regions']
    ],
    fill_color='#F8F9FA',
    font=dict(size=11),
    align='left')
), row=5, col=1)

fig.update_layout(
    title='<b>2024 Annual Performance Analysis</b><br>' +
          '<sub>Prepared for Executive Leadership Team | January 2025</sub>',
    title_x=0.5,
    title_font_size=20,
    showlegend=False,
    height=1400,
    plot_bgcolor='white',
    paper_bgcolor='white'
)

fig.show()
```

Narrative Report Structure:

1. Executive Summary:

- Key findings upfront
- For busy executives who may not read further

2. Body (Charts with Context):

- Each chart answers a specific question
- Titles are statements, not questions
- Annotations highlight key insights
- Logical flow from overview to details

3. Recommendations:

- Actionable next steps
- Specific, measurable
- Based on data shown

Exporting and Sharing Professional Reports

Export Options for Business Distribution:

1. Interactive HTML (Best for Digital Distribution):

```
fig.write_html(
    'Executive_Dashboard_Dec2024.html',
    config={
        'displayModeBar': True,  # Show toolbar
        'displaylogo': False,    # Remove Plotly logo
        'modeBarButtonsToRemove': ['lasso2d', 'select2d']  # Remove unnecessary
        ↪ buttons
    }
)
```

2. Static Image (For PowerPoint/Documents):

```
fig.write_image(
    'Dashboard_Report.png',
    width=1920,
    height=1080,
    scale=2  # High resolution
)
```

3. PDF Report (For Print):

```
fig.write_image(
    'Annual_Report_2024.pdf',
    width=1200,
    height=1600
)
```

Embedding in Business Tools:

SharePoint/Confluence:

1. Save as HTML
2. Upload to document library
3. Embed using iframe or direct link

Email Distribution:

1. Save HTML file
2. Attach to email
3. Recipients open in browser (full interactivity)

PowerPoint Presentation:

1. Export as high-res PNG
2. Insert as image in slides
3. Note: Loses interactivity

Responsive Design for Different Devices

Making Dashboards Mobile-Friendly:

```
import plotly.graph_objects as go

fig = go.Figure()

# Add your charts...

# Configure for responsive viewing
fig.update_layout(
    autosize=True,  # Adapt to container size
    margin=dict(l=40, r=40, t=80, b=40),  # Reasonable margins
    font=dict(size=12),  # Readable font size

    # Mobile-specific optimizations
    xaxis=dict(
        tickangle=-45  # Angle labels if needed
    ),

    # Adjust legend for mobile
    legend=dict(
        orientation='v',
        yanchor='top',
        y=0.99,
        xanchor='left',
        x=0.01
    )
)

# For HTML export with responsive container
```

```
fig.write_html(
    'responsive_dashboard.html',
    config={'responsive': True}
)
```

Design Considerations:

Desktop (>1200px width):

- Multi-column layouts
- Side-by-side charts
- Rich interactivity

Tablet (768-1200px):

- 2-column layouts
- Stacked panels
- Touch-friendly controls

Mobile (<768px):

- Single column
- Simplified charts
- Essential metrics only

4.3.3 Lab Session

Lab 3: Building Professional Business Dashboards

Objective: Design and create comprehensive, multi-chart interactive dashboards that communicate complex business insights clearly and professionally for executive audiences.

Scenario: You've been promoted to Lead Business Intelligence Developer at "DataVision Corp." Your first major assignment: create three distinct dashboards for different stakeholder groups at your client, "MegaMart Retail" - a national retail chain preparing for their annual investor meeting, board presentation, and operational review. Each dashboard must tell a clear story, be visually impressive, and provide actionable insights through interactivity.

Pre-Lab Setup:

1. Create file: `M4L03_YourName_Dashboards.py`
2. Import libraries:

```
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
import pandas as pd
import numpy as np
```

3. Create output folder: "Lab3_Outputs"

Part A: Executive KPI Dashboard (30 points)

Context: The CEO needs a clean, high-level dashboard for the investor meeting showing key performance indicators, trends, and status against targets. Must fit on one screen, be immediately understandable, and look professional.

Data Generation:

```
np.random.seed(42)

# Generate 2024 monthly data
months = pd.date_range('2024-01-01', '2024-12-31', freq='M')

df_executive = pd.DataFrame({
    'month': months,
    'revenue': np.random.uniform(8000000, 12000000, 12) + np.arange(12) * 200000,
    'operating_costs': np.random.uniform(5000000, 7000000, 12) + np.arange(12) *
     ↪  100000,
    'new_customers': np.random.randint(1500, 3000, 12),
    'churned_customers': np.random.randint(200, 600, 12),
    'nps_score': np.random.uniform(45, 75, 12),
    'employee_satisfaction': np.random.uniform(3.8, 4.6, 12)
})

df_executive['profit'] = df_executive['revenue'] - df_executive['operating_costs']
df_executive['profit_margin'] = df_executive['profit'] / df_executive['revenue']
df_executive['net_customer_growth'] = df_executive['new_customers'] -
 ↪  df_executive['churned_customers']

# Year-over-year data for comparison
ytd_revenue = df_executive['revenue'].sum()
previous_ytd_revenue = ytd_revenue * 0.85
ytd_customers = df_executive['new_customers'].sum()
previous_ytd_customers = int(ytd_customers * 0.92)
```

Tasks:

1. **Create KPI indicator row (10 points):**
   - Build using make_subplots with 4 columns, 1 row
   - Use `go.Indicator` for each KPI
   - KPI 1: Total Revenue YTD (with delta vs previous year)
   - KPI 2: Total New Customers (with delta vs previous year)
   - KPI 3: Average Profit Margin (gauge showing current vs target of 25%)
   - KPI 4: Current NPS Score (with delta vs target of 60)
   - All indicators should use appropriate colors (green for positive, red for negative)
   - Format currency properly with $ and commas
2. **Create main dashboard body (15 points):**
   - Below KPIs, create 2×2 grid of charts:
     - Top-left: Monthly revenue and profit trend (dual lines, different colors)
     - Top-right: Net customer growth (bar chart, color-coded positive/negative)
     - Bottom-left: Profit margin evolution (area chart with target line)
     - Bottom-right: Key metrics table (Revenue, Profit, Customers, NPS for last 3 months)
   - Use professional color scheme
   - All charts should have clear titles
   - Proper axis formatting (currency, percentages)
   - Enable hover mode ‘x unified’ for time series
3. **Professional finishing (5 points):**
   - Overall title: “MegaMart Retail - Executive Performance Dashboard”
   - Subtitle: “Year-to-Date 2024 | Prepared for Investor Meeting”
   - Clean layout with appropriate spacing
   - Background colors that look professional (not pure white/default)
   - Total height: 1000 pixels
   - Save as: `M4L03_YourName_ExecutiveDashboard.html`

Part B: Operational Performance Dashboard (35 points)

Context: The COO needs a detailed operational dashboard showing store performance, inventory metrics, employee productivity, and supply chain efficiency. More detailed than executive dashboard, with drill-down capabilities.

Data Generation:

```python
np.random.seed(42)

# Store performance data
stores = [f'Store {str(i).zfill(3)}' for i in range(1, 26)]  # 25 stores
regions = ['North', 'South', 'East', 'West', 'Central']

store_data = []
for store in stores:
    region = np.random.choice(regions)
    store_data.append({
        'store_id': store,
        'region': region,
        'monthly_sales': np.random.uniform(200000, 600000),
        'foot_traffic': np.random.randint(5000, 15000),
        'conversion_rate': np.random.uniform(0.15, 0.45),
        'avg_transaction': np.random.uniform(45, 120),
        'inventory_turnover': np.random.uniform(4, 12),
        'employee_count': np.random.randint(15, 45),
        'customer_rating': np.random.uniform(3.5, 4.9)
    })

df_operations = pd.DataFrame(store_data)
df_operations['sales_per_employee'] = df_operations['monthly_sales'] /
 ↪  df_operations['employee_count']
df_operations['sales_efficiency'] = df_operations['monthly_sales'] /
 ↪  df_operations['foot_traffic']

# Time series for inventory
weeks = pd.date_range('2024-01-01', '2024-12-31', freq='W')
inventory_levels = np.random.uniform(8000000, 12000000, len(weeks))
df_inventory = pd.DataFrame({'week': weeks, 'inventory_value': inventory_levels})

# Regional aggregations
df_regional = df_operations.groupby('region').agg({
    'monthly_sales': 'sum',
    'foot_traffic': 'sum',
    'conversion_rate': 'mean',
    'customer_rating': 'mean'
}).reset_index()
```

Tasks:

1. **Create regional overview section (12 points):**
    - Top section with regional performance comparison
    - Use animated bar chart (animation_frame could be a metric selection)
    - Show sales by region with color intensity by customer_rating
    - Include regional average conversion rate as text on bars
    - Add annotation highlighting best and worst performing regions
    - Format all currency values properly
2. **Create store performance scatter analysis (12 points):**
    - Middle-left: Scatter plot of foot_traffic (x) vs monthly_sales (y)
    - Size bubbles by employee_count
    - Color by region
    - Add trendline
    - Show sales_per_employee and customer_rating in hover
    - Title: “Store Efficiency Analysis: Traffic vs Sales”
    - Identify outliers with annotations (exceptionally high/low performers)

3. **Create inventory and efficiency panels (11 points):**
   - Middle-right: Inventory levels over time (line chart)
   - Add reference bands: Healthy range (green shade), Overstocked (red shade)
   - Bottom: Table showing top 10 and bottom 10 stores by sales_per_employee
   - Include store_id, region, sales_per_employee, customer_rating
   - Color code the table (top 10 green tint, bottom 10 red tint)
   - Sort appropriately
4. **Add interactive filters and insights (10 points):**
   - Create dropdown to filter by region (show all regions by default)
   - Add buttons to toggle between different metrics (Sales, Efficiency, Rating)
   - Include text box with automated insights:
     - Calculate and display: Highest performing store
     - Calculate and display: Average conversion rate by region
     - Calculate and display: Total sales across all stores
   - Save as: `M4L03_YourName_OperationalDashboard.html`

Part C: Marketing Analytics Report Dashboard (35 points)

Context: The CMO needs a narrative-driven dashboard that tells the story of 2024 marketing performance, channel effectiveness, customer acquisition, and ROI. Should be presentation-ready with clear flow from overview to recommendations.

Data Generation:

```
np.random.seed(42)

# Campaign data
channels = ['Email', 'Social Media', 'PPC', 'Display Ads', 'Influencer', 'TV',
 ↪  'Radio']
months = pd.date_range('2024-01-01', '2024-12-31', freq='M')

campaign_data = []
for month in months:
    for channel in channels:
        spend = np.random.uniform(10000, 100000)

        # Different channels have different effectiveness
        if channel == 'Email':
            roi = np.random.uniform(3.5, 6.0)
            leads = int(spend * np.random.uniform(0.8, 1.5))
        elif channel == 'Social Media':
            roi = np.random.uniform(2.5, 5.0)
            leads = int(spend * np.random.uniform(0.6, 1.2))
        elif channel == 'PPC':
            roi = np.random.uniform(2.0, 4.5)
            leads = int(spend * np.random.uniform(0.4, 1.0))
        elif channel == 'Influencer':
            roi = np.random.uniform(2.8, 5.5)
            leads = int(spend * np.random.uniform(0.5, 1.3))
        else:
            roi = np.random.uniform(1.5, 3.5)
            leads = int(spend * np.random.uniform(0.3, 0.8))

        revenue = spend * roi
        conversions = int(leads * np.random.uniform(0.15, 0.35))

        campaign_data.append({
            'month': month,
            'channel': channel,
            'spend': spend,
            'revenue': revenue,
            'roi': roi,
            'leads': leads,
            'conversions': conversions
        })

df_marketing = pd.DataFrame(campaign_data)
df_marketing['cpa'] = df_marketing['spend'] / df_marketing['conversions']  # Cost per
↪  acquisition
df_marketing['conversion_rate'] = df_marketing['conversions'] / df_marketing['leads']

# Customer journey data
touchpoints = ['Awareness', 'Consideration', 'Decision', 'Purchase', 'Loyalty']
funnel_values = [50000, 25000, 12000, 8000, 5500]
df_funnel = pd.DataFrame({'stage': touchpoints, 'customers': funnel_values})
```

Tasks:

1. **Create executive summary section (10 points):**
    - Top: Title with executive summary text
    - Use table or annotation to display:
        - "2024 Marketing Investment: $X.XX million"
        - "Total Revenue Generated: $X.XX million"
        - "Overall ROI: X.X:1"
        - "Total Leads Generated: XXX,XXX"
        - "Average Cost Per Acquisition: $XXX"
    - Make this section visually distinct (background color, border)
    - Calculate these metrics from df_marketing
2. **Create channel performance comparison (12 points):**
    - Section title: "Channel Effectiveness: Where Should We Invest?"
    - Left side: Horizontal bar chart of total spend by channel
    - Right side: Scatter plot of spend (x) vs ROI (y) by channel
    - Color code by channel type (paid vs organic vs traditional)
    - Annotate the "star" channels (high ROI, reasonable spend)
    - Add text insights identifying best performing channel
3. **Create temporal trends section (8 points):**
    - Section title: "Marketing Performance Trends Throughout 2024"
    - Stacked area chart showing spend by channel over months
    - Line overlay showing total monthly ROI
    - Enable legend interactions to show/hide channels
    - Identify and annotate peak performance months
4. **Create customer journey funnel (5 points):**
    - Section title: "Customer Acquisition Funnel"
    - Funnel chart or horizontal bar chart showing conversion stages
    - Calculate and display drop-off percentages between stages
    - Add annotations highlighting the biggest drop-off point
    - Color code by stage health (good conversion = green, poor = red)
5. **Create recommendations section (10 points):**
    - Bottom: Strategic recommendations based on data
    - Create an automated insights table showing:
        - Top 3 channels by ROI (with values)
        - Channels to increase budget (high ROI, lower current spend)
        - Channels to decrease budget (low ROI despite high spend)

    - Seasonal patterns identified
- Format as professional report table
- Use conditional formatting (colors) to highlight recommendations
- Save as: `M4L03_YourName_MarketingReport.html`

Bonus Challenge (+30 points):

Create an Integrated Master Dashboard with Tab Navigation:

Build a single HTML file with tabbed interface allowing users to switch between: 1. Executive Dashboard (Part A) 2. Operations Dashboard (Part B) 3. Marketing Report (Part C)

Requirements:

- Use `updatemenus` with buttons for tab switching
- Each "tab" shows/hides relevant charts
- Consistent branding across all tabs
- Company logo (can be text-based)
- Date stamp and report version
- Print-friendly option (button to show all at once)
- Navigation breadcrumbs showing current view
- Save as: `M4L03_YourName_MasterDashboard.html`

Hint: Use visibility toggling on subplot traces with button controls.

Deliverables:

1. Python file: `M4L03_YourName_Dashboards.py`
2. HTML files (3-4):
    - `M4L03_YourName_ExecutiveDashboard.html`
    - `M4L03_YourName_OperationalDashboard.html`
    - `M4L03_YourName_MarketingReport.html`
    - `M4L03_YourName_MasterDashboard.html` (bonus)

Grading Rubric:

- Part A (Executive Dashboard): 30 points
- Part B (Operational Dashboard): 35 points
- Part C (Marketing Report): 35 points
- Bonus (Master Dashboard): +30 points

Success Criteria:

□ All dashboards tell clear stories
□ KPI indicators work correctly
□ Multiple chart types used appropriately
□ Professional color schemes and styling
□ All numbers formatted properly (currency, %)
□ Interactive elements function smoothly
□ Dashboards are visually balanced (not cluttered)
□ Business insights are evident
□ HTML files display perfectly in browser

Dashboard Design Checklist:

Visual Hierarchy:

□ Most important information in top-left
□ Clear visual flow from overview to detail
□ Appropriate use of size and color for emphasis

Interactivity:

□ Hover information is comprehensive
□ Filters/dropdowns work correctly
□ Legend interactions enabled
□ Zoom and pan function where appropriate

Professionalism:

- □ Consistent color scheme
- □ Readable fonts and sizes
- □ Proper spacing and alignment
- □ No overlapping elements
- □ Clean, uncluttered appearance

Business Value:

- □ Answers specific business questions
- □ Actionable insights visible
- □ Comparisons clearly shown
- □ Trends easy to identify
- □ Anomalies highlighted

Common Mistakes to Avoid:

- Too many colors (stick to 3-5 main colors)
- Inconsistent styling across dashboard
- Missing titles or labels
- Poor data-to-ink ratio (chart junk)
- Overuse of 3D effects or animations
- Not testing in actual browser
- Forgetting to format numbers
- No clear call-to-action or insights

4.4 Section 4: Final Project - Comprehensive Business Data Analysis

4.4.1 Objective

- Integrate all skills from Modules 1-4 into a complete analysis project
- Apply the full data analysis workflow: import, clean, analyze, visualize
- Create a professional, multi-visualization analytical report
- Present findings with both static and interactive visualizations
- Demonstrate mastery of Python, Pandas, Matplotlib, Seaborn, and Plotly
- Deliver publication-quality business intelligence deliverables

4.4.2 Main Contents with Examples

The Complete Data Analysis Workflow

Professional Analysis Process:

Real-world business analysis follows a structured workflow:

1. Business Understanding:

- What questions need answering?
- Who is the audience?
- What decisions will be made?

2. Data Collection:

- Identify data sources
- Load data into Python
- Initial exploration

3. Data Cleaning:

- Handle missing values
- Remove duplicates

- Fix data types
- Address outliers

4. **Exploratory Analysis:**
   - Summary statistics
   - Distribution analysis
   - Correlation discovery
   - Pattern identification
5. **Visualization:**
   - Static charts for reports (Matplotlib/Seaborn)
   - Interactive dashboards for exploration (Plotly)
   - Multiple chart types for different insights
6. **Insights & Recommendations:**
   - Synthesize findings
   - Translate to business language
   - Provide actionable recommendations
   - Quantify impacts
7. **Presentation:**
   - Create comprehensive report
   - Prepare stakeholder presentation
   - Document methodology
   - Enable reproducibility

Project Specification and Requirements

Your Final Project Mission:

You will complete a comprehensive analysis of a business dataset, applying every technique learned in this course.

Deliverables Required:

1. **Python Analysis Script:**
   - Complete, well-commented code
   - Follows workflow from data loading to visualization
   - Modular and reproducible
2. **Static Visualizations (Matplotlib/Seaborn):**
   - Minimum 6 high-quality charts
   - Different chart types
   - Publication-ready quality
3. **Interactive Dashboard (Plotly):**
   - Multi-panel dashboard
   - Interactive elements
   - Professional styling
4. **Written Report:**
   - Executive summary
   - Methodology description
   - Findings with visualizations
   - Recommendations

Sample Project Structure

Example: Retail Sales Analysis Project

```
"""
Final Project: Comprehensive Retail Sales Analysis
Course: Data Visualization Analysis
Author: [Your Name]
Date: [Submission Date]

Business Question:
Analyze 2024 sales performance across products, regions, and time periods
to identify growth opportunities and operational improvements for 2025 planning.

Dataset: Retail sales data with 10,000+ transactions
"""

# ============================================================================
# SECTION 1: SETUP AND DATA LOADING
# ============================================================================

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Set visualization styles
sns.set_theme(style='whitegrid', context='talk', palette='colorblind')
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['font.size'] = 11

print("="*70)
print("RETAIL SALES ANALYSIS PROJECT - 2024")
print("="*70)

# ============================================================================
# SECTION 2: DATA IMPORT AND INITIAL EXPLORATION
# ============================================================================

# Load data (in actual project, this would be from file)
# For demonstration, we'll generate realistic sample data
np.random.seed(42)

# Generate comprehensive retail dataset
n_transactions = 10000
dates = pd.date_range('2024-01-01', '2024-12-31', freq='H')
products = ['Electronics', 'Clothing', 'Home Goods', 'Sports Equipment',
            'Books', 'Toys', 'Food & Beverage', 'Health & Beauty']
regions = ['North', 'South', 'East', 'West', 'Central']
channels = ['Online', 'In-Store', 'Mobile App']

transactions = []
for i in range(n_transactions):
    product = np.random.choice(products, p=[0.15, 0.18, 0.12, 0.10, 0.08, 0.09, 0.15,
 ↪  0.13])
    region = np.random.choice(regions)
    channel = np.random.choice(channels, p=[0.45, 0.40, 0.15])

    # Price varies by product category
```

```python
    price_ranges = {
        'Electronics': (50, 500), 'Clothing': (20, 150),
        'Home Goods': (30, 300), 'Sports Equipment': (40, 400),
        'Books': (10, 50), 'Toys': (15, 100),
        'Food & Beverage': (5, 50), 'Health & Beauty': (10, 100)
    }
    price = np.random.uniform(*price_ranges[product])

    # Quantity varies by channel
    if channel == 'Online':
        quantity = np.random.randint(1, 5)
    else:
        quantity = np.random.randint(1, 3)

    # Customer satisfaction varies
    satisfaction = np.random.uniform(3.0, 5.0)

    # Shipping time (online only)
    shipping_days = np.random.randint(2, 10) if channel == 'Online' else 0

    transactions.append({
        'transaction_id': f'TXN{i:06d}',
        'date': np.random.choice(dates),
        'product_category': product,
        'region': region,
        'channel': channel,
        'unit_price': price,
        'quantity': quantity,
        'customer_satisfaction': satisfaction,
        'shipping_days': shipping_days
    })

df = pd.DataFrame(transactions)

# Calculate derived fields
df['total_amount'] = df['unit_price'] * df['quantity']
df['year_month'] = df['date'].dt.to_period('M')
df['day_of_week'] = df['date'].dt.day_name()
df['hour'] = df['date'].dt.hour

print("\n1. Dataset Overview:")
print("-" * 70)
print(f"Total Transactions: {len(df):,}")
print(f"Date Range: {df['date'].min()} to {df['date'].max()}")
print(f"Total Revenue: ${df['total_amount'].sum():,.2f}")
print(f"Average Transaction Value: ${df['total_amount'].mean():.2f}")

print("\n2. Data Structure:")
print(df.info())

print("\n3. Sample Records:")
print(df.head())

# ============================================================================
# SECTION 3: DATA CLEANING AND PREPARATION
# ============================================================================

print("\n" + "="*70)
print("DATA CLEANING AND PREPARATION")
print("="*70)

# Check for missing values
print("\n1. Missing Values Check:")
print(df.isnull().sum())

# Check for duplicates
duplicates = df.duplicated(subset=['transaction_id']).sum()
print(f"\n2. Duplicate Transactions: {duplicates}")

# Data quality checks
print("\n3. Data Quality Checks:")
print(f"   - Negative prices: {(df['unit_price'] < 0).sum()}")
print(f"   - Zero quantities: {(df['quantity'] == 0).sum()}")
print(f"   - Invalid satisfaction scores: {((df['customer_satisfaction'] < 1) |
 ↪  (df['customer_satisfaction'] > 5)).sum()}")

# Outlier detection (IQR method)
Q1 = df['total_amount'].quantile(0.25)
Q3 = df['total_amount'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df['total_amount'] < Q1 - 1.5*IQR) | (df['total_amount'] > Q3 +
 ↪  1.5*IQR)]
print(f"\n4. Statistical Outliers Detected: {len(outliers)}
 ↪  ({len(outliers)/len(df)*100:.2f}%)")

# Clean dataset
df_clean = df.copy()
# Keep outliers but flag them
df_clean['is_outlier'] = ((df_clean['total_amount'] < Q1 - 1.5*IQR) |
                           (df_clean['total_amount'] > Q3 + 1.5*IQR))

print(f"\n5. Clean Dataset: {len(df_clean):,} transactions ready for analysis")

# ============================================================================
# SECTION 4: EXPLORATORY DATA ANALYSIS
# ============================================================================

print("\n" + "="*70)
print("EXPLORATORY DATA ANALYSIS")
print("="*70)

# Summary statistics
print("\n1. Summary Statistics:")
print(df_clean[['unit_price', 'quantity', 'total_amount',
 ↪  'customer_satisfaction']].describe())

# Category analysis
print("\n2. Performance by Product Category:")
category_summary = df_clean.groupby('product_category').agg({
    'total_amount': ['sum', 'mean', 'count'],
    'customer_satisfaction': 'mean'
}).round(2)
category_summary.columns = ['Total_Revenue', 'Avg_Transaction', 'Count',
 ↪  'Avg_Satisfaction']
print(category_summary.sort_values('Total_Revenue', ascending=False))
```

```
# Regional analysis
print("\n3. Performance by Region:")
regional_summary = df_clean.groupby('region').agg({
    'total_amount': ['sum', 'mean'],
    'transaction_id': 'count'
}).round(2)
regional_summary.columns = ['Total_Revenue', 'Avg_Transaction', 'Transaction_Count']
print(regional_summary.sort_values('Total_Revenue', ascending=False))

# Channel analysis
print("\n4. Performance by Channel:")
channel_summary = df_clean.groupby('channel').agg({
    'total_amount': ['sum', 'mean'],
    'customer_satisfaction': 'mean'
}).round(2)
channel_summary.columns = ['Total_Revenue', 'Avg_Transaction', 'Avg_Satisfaction']
print(channel_summary.sort_values('Total_Revenue', ascending=False))

# Correlation analysis
print("\n5. Correlation Analysis:")
correlation_vars = df_clean[['unit_price', 'quantity', 'total_amount',
                              'customer_satisfaction', 'shipping_days']].corr()
print(correlation_vars.round(3))

# ============================================================================
# SECTION 5: STATIC VISUALIZATIONS (Matplotlib & Seaborn)
# ============================================================================

print("\n" + "="*70)
print("CREATING STATIC VISUALIZATIONS")
print("="*70)

# Create output directory for charts
import os
os.makedirs('FinalProject_Outputs', exist_ok=True)

# Visualization 1: Revenue trend over time
print("\n1. Creating: Monthly revenue trend...")
monthly_revenue = df_clean.groupby('year_month')['total_amount'].sum()

fig, ax = plt.subplots(figsize=(14, 6))
ax.plot(monthly_revenue.index.astype(str), monthly_revenue.values,
        marker='o', linewidth=3, markersize=8, color='#2E86AB')
ax.fill_between(range(len(monthly_revenue)), monthly_revenue.values, alpha=0.3,
 ↪  color='#2E86AB')
ax.set_title('Monthly Revenue Trend - 2024', fontsize=16, fontweight='bold', pad=20)
ax.set_xlabel('Month', fontsize=12)
ax.set_ylabel('Revenue ($)', fontsize=12)
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x/1e6:.2f}M'))
ax.grid(True, alpha=0.3)
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('FinalProject_Outputs/FP_01_Revenue_Trend.png', dpi=300,
 ↪  bbox_inches='tight')
plt.close()

# Visualization 2: Category performance comparison
print("2. Creating: Category performance bars...")
```

```
category_rev =
 ↪  df_clean.groupby('product_category')['total_amount'].sum().sort_values(ascending=True)

fig, ax = plt.subplots(figsize=(12, 8))
colors = plt.cm.viridis(np.linspace(0.3, 0.9, len(category_rev)))
ax.barh(category_rev.index, category_rev.values, color=colors, edgecolor='black',
 ↪  linewidth=1.5)
ax.set_title('Total Revenue by Product Category', fontsize=16, fontweight='bold',
 ↪  pad=20)
ax.set_xlabel('Revenue ($)', fontsize=12)
ax.xaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x/1e6:.1f}M'))

# Add value labels
for i, v in enumerate(category_rev.values):
    ax.text(v + 50000, i, f'${v/1e6:.2f}M', va='center', fontsize=10,
 ↪  fontweight='bold')

ax.grid(axis='x', alpha=0.3)
plt.tight_layout()
plt.savefig('FinalProject_Outputs/FP_02_Category_Revenue.png', dpi=300,
 ↪  bbox_inches='tight')
plt.close()

# Visualization 3: Customer satisfaction distribution
print("3. Creating: Satisfaction distribution...")
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Overall distribution
sns.histplot(data=df_clean, x='customer_satisfaction', bins=30, kde=True,
             color='#06A77D', ax=axes[0])
axes[0].set_title('Overall Customer Satisfaction Distribution', fontsize=14,
 ↪  fontweight='bold')
axes[0].set_xlabel('Satisfaction Score (1-5)', fontsize=11)
axes[0].axvline(df_clean['customer_satisfaction'].mean(), color='red',
                linestyle='--', linewidth=2, label=f"Mean:
 ↪  {df_clean['customer_satisfaction'].mean():.2f}")
axes[0].legend()

# By channel
sns.boxplot(data=df_clean, x='channel', y='customer_satisfaction',
            palette='Set2', ax=axes[1])
axes[1].set_title('Satisfaction by Channel', fontsize=14, fontweight='bold')
axes[1].set_xlabel('Channel', fontsize=11)
axes[1].set_ylabel('Satisfaction Score', fontsize=11)
axes[1].axhline(4.0, color='red', linestyle='--', linewidth=2, alpha=0.5,
 ↪  label='Target: 4.0')
axes[1].legend()

plt.tight_layout()
plt.savefig('FinalProject_Outputs/FP_03_Satisfaction_Analysis.png', dpi=300,
 ↪  bbox_inches='tight')
plt.close()

# Visualization 4: Regional performance heatmap
print("4. Creating: Regional performance heatmap...")
regional_channel = df_clean.pivot_table(values='total_amount',
                                        index='region',
                                        columns='channel',
```

```python
                                         aggfunc='sum')

fig, ax = plt.subplots(figsize=(10, 6))
sns.heatmap(regional_channel, annot=True, fmt=',.0f', cmap='YlGnBu',
            linewidths=1, cbar_kws={'label': 'Revenue ($)'}, ax=ax)
ax.set_title('Revenue by Region and Channel', fontsize=16, fontweight='bold', pad=20)
ax.set_xlabel('Channel', fontsize=12)
ax.set_ylabel('Region', fontsize=12)
plt.tight_layout()
plt.savefig('FinalProject_Outputs/FP_04_Regional_Heatmap.png', dpi=300,
 ↪  bbox_inches='tight')
plt.close()

# Visualization 5: Price vs Quantity relationship
print("5. Creating: Price-quantity scatter...")
fig, ax = plt.subplots(figsize=(12, 7))
for category in df_clean['product_category'].unique():
    df_cat = df_clean[df_clean['product_category'] == category]
    ax.scatter(df_cat['unit_price'], df_cat['quantity'],
               alpha=0.6, s=50, label=category)

ax.set_title('Unit Price vs Quantity Sold by Category', fontsize=16,
 ↪  fontweight='bold', pad=20)
ax.set_xlabel('Unit Price ($)', fontsize=12)
ax.set_ylabel('Quantity', fontsize=12)
ax.legend(title='Product Category', bbox_to_anchor=(1.05, 1), loc='upper left')
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('FinalProject_Outputs/FP_05_Price_Quantity_Scatter.png', dpi=300,
 ↪  bbox_inches='tight')
plt.close()

# Visualization 6: Correlation heatmap
print("6. Creating: Correlation matrix...")
fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(correlation_vars, annot=True, fmt='.3f', cmap='coolwarm',
            center=0, square=True, linewidths=1, cbar_kws={'label': 'Correlation'},
 ↪  ax=ax)
ax.set_title('Correlation Matrix: Key Metrics', fontsize=16, fontweight='bold',
 ↪  pad=20)
plt.tight_layout()
plt.savefig('FinalProject_Outputs/FP_06_Correlation_Matrix.png', dpi=300,
 ↪  bbox_inches='tight')
plt.close()

print("\n  All static visualizations saved to FinalProject_Outputs/")

# ============================================================================
# SECTION 6: INTERACTIVE DASHBOARD (Plotly)
# ============================================================================

print("\n" + "="*70)
print("CREATING INTERACTIVE DASHBOARD")
print("="*70)

# Prepare aggregated data for dashboard
monthly_data = df_clean.groupby('year_month').agg({
    'total_amount': 'sum',
    'transaction_id': 'count',
    'customer_satisfaction': 'mean'
}).reset_index()
monthly_data['year_month'] = monthly_data['year_month'].astype(str)

category_data = df_clean.groupby('product_category').agg({
    'total_amount': 'sum',
    'customer_satisfaction': 'mean',
    'transaction_id': 'count'
}).reset_index()

# Create comprehensive dashboard
fig = make_subplots(
    rows=3, cols=3,
    row_heights=[0.15, 0.40, 0.45],
    column_widths=[0.33, 0.33, 0.34],
    specs=[
        [{'type': 'indicator'}, {'type': 'indicator'}, {'type': 'indicator'}],
        [{'type': 'scatter', 'colspan': 2}, None, {'type': 'bar'}],
        [{'type': 'bar', 'colspan': 2}, None, {'type': 'pie'}]
    ],
    subplot_titles=['', '', '',
                    'Monthly Revenue Trend', '', 'Top Categories',
                    'Regional Performance', '', 'Channel Distribution'],
    vertical_spacing=0.10,
    horizontal_spacing=0.08
)

# Row 1: KPI Indicators
total_revenue = df_clean['total_amount'].sum()
total_transactions = len(df_clean)
avg_satisfaction = df_clean['customer_satisfaction'].mean()

fig.add_trace(go.Indicator(
    mode='number',
    value=total_revenue,
    title={'text': 'Total Revenue'},
    number={'prefix': '$', 'valueformat': ',.0f'},
    domain={'x': [0, 1], 'y': [0, 1]}
), row=1, col=1)

fig.add_trace(go.Indicator(
    mode='number',
    value=total_transactions,
    title={'text': 'Total Transactions'},
    number={'valueformat': ','},
    domain={'x': [0, 1], 'y': [0, 1]}
), row=1, col=2)

fig.add_trace(go.Indicator(
    mode='gauge+number',
    value=avg_satisfaction,
    title={'text': 'Avg Satisfaction'},
    gauge={
        'axis': {'range': [1, 5]},
        'bar': {'color': '#06A77D'},
        'threshold': {
            'line': {'color': 'red', 'width': 4},
            'thickness': 0.75,
            'value': 4.0
        }
    },
    domain={'x': [0, 1], 'y': [0, 1]}
), row=1, col=3)

# Row 2: Monthly trend and Top categories
fig.add_trace(go.Scatter(
    x=monthly_data['year_month'],
    y=monthly_data['total_amount'],
    mode='lines+markers',
    line=dict(color='#2E86AB', width=3),
    marker=dict(size=10),
    name='Revenue',
    hovertemplate='<b>%{x}</b><br>Revenue: $%{y:,.0f}<extra></extra>'
), row=2, col=1)

top_categories = category_data.nlargest(5, 'total_amount')
fig.add_trace(go.Bar(
    x=top_categories['total_amount'],
    y=top_categories['product_category'],
    orientation='h',
    marker_color='#FF6B6B',
    name='Categories',
    hovertemplate='<b>%{y}</b><br>$%{x:,.0f}<extra></extra>'
), row=2, col=3)

# Row 3: Regional bars and Channel pie
regional_totals =
 ↪  df_clean.groupby('region')['total_amount'].sum().sort_values(ascending=False)
fig.add_trace(go.Bar(
    x=regional_totals.index,
    y=regional_totals.values,
    marker_color='#4ECDC4',
    name='Regions',
    hovertemplate='<b>%{x}</b><br>$%{y:,.0f}<extra></extra>'
), row=3, col=1)

channel_totals = df_clean.groupby('channel')['total_amount'].sum()
fig.add_trace(go.Pie(
    labels=channel_totals.index,
    values=channel_totals.values,
    marker_colors=['#45B7D1', '#FFA07A', '#98D8C8'],
    name='Channels',
    hovertemplate='<b>%{label}</b><br>$%{value:,.0f}<br>%{percent}<extra></extra>'
), row=3, col=3)

# Update layout
fig.update_layout(
    title_text='<b>Retail Sales Analytics Dashboard - 2024</b><br>' +
               '<sub>Interactive Performance Overview</sub>',
    title_x=0.5,
    title_font_size=20,
    showlegend=False,
    height=1100,
    plot_bgcolor='#F8F9FA',
    paper_bgcolor='white'
```

```
)

# Update axes
fig.update_xaxes(showgrid=True, gridcolor='white')
fig.update_yaxes(showgrid=True, gridcolor='white')
fig.update_yaxes(tickformat='$,.0f', row=2, col=1)
fig.update_xaxes(tickformat='$,.0f', row=2, col=3)
fig.update_yaxes(tickformat='$,.0f', row=3, col=1)

# Save dashboard
fig.write_html('FinalProject_Outputs/FP_Interactive_Dashboard.html')
print("\n  Interactive dashboard saved: FP_Interactive_Dashboard.html")

# ============================================================================
# SECTION 7: KEY FINDINGS AND RECOMMENDATIONS
# ============================================================================

print("\n" + "="*70)
print("KEY FINDINGS AND RECOMMENDATIONS")
print("="*70)

print("\n  KEY FINDINGS:")
print("-" * 70)

# Finding 1: Revenue trends
total_rev = df_clean['total_amount'].sum()
print(f"\n1. REVENUE PERFORMANCE:")
print(f"   • Total 2024 revenue: ${total_rev:,.2f}")
print(f"   • Average monthly revenue: ${total_rev/12:,.2f}")
best_month = df_clean.groupby('year_month')['total_amount'].sum().idxmax()
print(f"   • Best performing month: {best_month}")

# Finding 2: Category insights
top_category = category_data.nlargest(1, 'total_amount').iloc[0]
print(f"\n2. PRODUCT CATEGORIES:")
print(f"   • Top category: {top_category['product_category']}")
print(f"   • Category revenue: ${top_category['total_amount']:,.2f}")
print(f"   • Category satisfaction: {top_category['customer_satisfaction']:.2f}/5.0")

# Finding 3: Regional performance
best_region = df_clean.groupby('region')['total_amount'].sum().idxmax()
worst_region = df_clean.groupby('region')['total_amount'].sum().idxmin()
print(f"\n3. REGIONAL ANALYSIS:")
print(f"   • Best region: {best_region}")
print(f"   • Underperforming region: {worst_region}")

# Finding 4: Channel effectiveness
channel_perf = df_clean.groupby('channel').agg({
    'total_amount': 'sum',
    'customer_satisfaction': 'mean'
})
best_channel = channel_perf['total_amount'].idxmax()
print(f"\n4. CHANNEL PERFORMANCE:")
print(f"   • Leading channel: {best_channel}")
print(f"   • Overall satisfaction: {avg_satisfaction:.2f}/5.0")

print("\n" + "="*70)
print("  STRATEGIC RECOMMENDATIONS:")
print("="*70)

print(f"""
1. INVESTMENT PRIORITIES:
   • Expand {top_category['product_category']} inventory (top performer)
   • Increase {best_channel} capacity (highest revenue channel)
   • Deploy best practices from {best_region} to other regions

2. IMPROVEMENT AREAS:
   • Focus on {worst_region} region - investigate and address gap
   • Improve satisfaction in lower-rated categories
   • Optimize shipping times for online orders

3. GROWTH OPPORTUNITIES:
   • Launch targeted campaigns in underperforming regions
   • Develop mobile app features (growing channel)
   • Create bundle offers for complementary categories

4. OPERATIONAL EXCELLENCE:
   • Maintain satisfaction above 4.0 benchmark
   • Standardize processes across high-performing regions
   • Monitor and address outlier transactions

5. 2025 STRATEGIC INITIATIVES:
   • Set aggressive growth targets for {worst_region} (+30%)
   • Expand {top_category['product_category']} product line
   • Enhance {best_channel} customer experience
   • Implement predictive analytics for inventory management
""")

print("\n" + "="*70)
print("PROJECT COMPLETE - All deliverables generated!")
print("="*70)
print("\nDeliverables created:")
print("   6 static visualizations (Matplotlib/Seaborn)")
print("   1 interactive dashboard (Plotly)")
print("   Comprehensive analysis script")
print("   Business insights and recommendations")
print("\nAll files saved to: FinalProject_Outputs/")
```

4.4.3 Lab Session

Final Project: Comprehensive Business Data Analysis

Objective: Apply all skills learned throughout the course to complete a professional, end-to-end business intelligence project demonstrating mastery of Python, data analysis, and visualization.

Scenario: You are the Senior Business Intelligence Analyst at a company of your choice. The executive team has commissioned a comprehensive annual analysis to guide strategic planning for the coming year. Your mission is to analyze the provided dataset (or one you choose), uncover actionable insights, and present findings through a combination of static reports and interactive dashboards.

Project Options (Choose One):

Option A: E-commerce Performance Analysis

- Dataset: Online retail transactions
- Focus: Sales trends, customer behavior, product performance
- Stakeholders: CEO, CMO, Head of E-commerce

Option B: HR Analytics and Employee Retention

- Dataset: Employee data, satisfaction surveys, performance metrics
- Focus: Retention factors, satisfaction drivers, compensation analysis
- Stakeholders: CHRO, Department Heads, Executive Team

Option C: Marketing Campaign Effectiveness

- Dataset: Multi-channel marketing campaigns, leads, conversions
- Focus: ROI by channel, customer acquisition cost, attribution
- Stakeholders: CMO, Marketing Directors, Finance

Option D: Financial Performance and Profitability

- Dataset: Revenue, costs, profit by product/region/time
- Focus: Margin analysis, cost drivers, growth opportunities
- Stakeholders: CFO, CEO, Board of Directors

You may also choose your own business scenario with instructor approval.

Project Requirements:

Part 1: Data Preparation and Cleaning (20 points)

Deliverable: Python script section demonstrating: 1. Data import from CSV/Excel (or generated data) 2. Initial exploratory analysis: - Dataset dimensions and structure - Summary statistics - Data type verification 3. Data quality assessment: - Missing value detection and handling - Duplicate identification and removal - Outlier detection using IQR method - Data type corrections 4. Derived field creation: - Calculate relevant business metrics - Create time-based features (month, quarter, etc.) - Add categorical groupings as needed 5. Documentation: - Comment all cleaning decisions - Document assumptions made - Report data quality metrics

Success Criteria:

☐ Data successfully loaded and parsed
☐ Missing values handled appropriately
☐ Duplicates removed with justification
☐ Outliers identified and managed
☐ Clean dataset ready for analysis
☐ All steps documented in comments

Part 2: Exploratory Analysis (20 points)

Deliverable: Python script section with: 1. Descriptive statistics: - Summary stats for all numeric variables - Frequency tables for categorical variables - Distributions by key segments 2. Correlation analysis: - Calculate correlation matrix - Identify strong relationships - Document business implications 3. Segment analysis: - Group by key dimensions (region, category, time) - Calculate aggregated metrics - Compare performance across segments 4. Trend identification: - Time series patterns - Seasonal effects - Growth rates 5. Insights documentation: - Summarize findings in comments - Highlight surprising patterns - Note questions for further analysis

Success Criteria:

☐ Comprehensive statistical summary
☐ Correlation analysis completed
☐ Segmentation reveals patterns
☐ Trends clearly identified
☐ Findings documented in text

Part 3: Static Visualizations (25 points)

Deliverable: Minimum 8 high-quality static charts using Matplotlib and Seaborn:

Required Chart Types (must include):

1. **Time series trend** (line chart showing change over time)
2. **Category comparison** (bar chart comparing groups)
3. **Distribution analysis** (histogram/KDE showing spread)
4. **Relationship plot** (scatter plot with correlation)
5. **Composition chart** (pie or stacked bar showing parts of whole)

6. **Heatmap** (correlation matrix or pivot table)

Additional Charts (choose 2):

7. Box plot for group comparisons
8. Pair plot for multivariate analysis
9. Violin plot for distribution comparison
10. Faceted plot showing multiple dimensions

Each chart must:

- Have descriptive title
- Include axis labels with units
- Use appropriate colors (accessible)
- Format numbers properly (currency, percentages)
- Include legends where needed
- Be saved as high-resolution PNG (300 dpi)

Success Criteria:

☐ All 8 required charts created
☐ Professional appearance and formatting
☐ Appropriate chart types for data stories
☐ All charts properly labeled
☐ Files saved with descriptive names
☐ Insights visible from each chart

Part 4: Interactive Dashboard (25 points)

Deliverable: Comprehensive Plotly dashboard (single HTML file) containing:

Dashboard Structure:

1. **KPI Section** (top):
   - 3-4 key performance indicators
   - Use indicator or gauge visualizations
   - Show current value and comparison (delta or vs target)
   - Clear, large numbers
2. **Main Analysis Section** (middle):
   - 4-6 interactive charts
   - Mix of chart types (scatter, line, bar, etc.)
   - Coordinated hover information
   - Consistent color scheme
3. **Detail Section** (bottom):
   - Supporting tables or charts
   - Additional breakdowns
   - Summary statistics

Interactive Features (must include at least 3):

- Hover tooltips with detailed information
- Dropdown menus or buttons for filtering
- Zoom and pan capabilities
- Click interactions (hide/show in legend)
- Range sliders for time series
- Animation (if appropriate for data story)

Success Criteria:

☐ Dashboard tells cohesive story
☐ Multiple interactive features work smoothly
☐ Professional styling and colors
☐ Responsive layout (charts don't overlap)
☐ All numbers formatted correctly
☐ Opens perfectly in web browser
☐ Saved as single HTML file

Part 5: Executive Report (10 points)

Deliverable: Written document (PDF or Markdown) containing:

Structure:

1. **Executive Summary (1 page):**
   - Business context and objectives
   - Key findings (3-5 bullet points)
   - Primary recommendations
2. **Methodology (½ page):**
   - Data sources and scope
   - Analysis approach
   - Tools used
3. **Findings (2-3 pages):**
   - Major insights with supporting visualizations
   - Statistical evidence
   - Business implications
4. **Recommendations (1 page):**
   - 5-7 specific, actionable recommendations
   - Prioritized by impact
   - Implementation considerations
5. **Appendix:**
   - Technical details
   - Additional charts
   - Data dictionary

Success Criteria:

☐ Clear, professional writing
☐ Executive summary is standalone
☐ Findings supported by data
☐ Recommendations are specific and actionable
☐ Visualizations embedded appropriately
☐ Professional formatting

Final Deliverables Checklist:

Submit all of the following: 1. **Python Script:** `FinalProject_YourName_Analysis.py` - Complete, well-commented code - Follows project structure - Runs without errors

2. **Static Visualizations Folder:** `Static_Charts/`
   - 8+ PNG files (300 dpi)
   - Descriptive filenames
   - Professional quality
3. **Interactive Dashboard:** `FinalProject_YourName_Dashboard.html`
   - Single HTML file
   - Fully functional in browser
   - Professional appearance
4. **Executive Report:** `FinalProject_YourName_Report.pdf`
   - Professional business document
   - Includes key visualizations
   - Clear recommendations
5. **Readme File:** `README.md`
   - Project overview
   - How to run the code
   - File descriptions
   - Dependencies required

Grading Rubric:

Component	Points	Criteria
Data Preparation	20	Cleaning thoroughness, documentation
Exploratory Analysis	20	Statistical rigor, insight depth
Static Visualizations	25	Quality, variety, appropriateness
Interactive Dashboard	25	Functionality, design, interactivity
Executive Report	10	Clarity, professionalism, actionability
Total	**100**	

Bonus Opportunities (+20 points):

- Advanced statistical analysis (regression, clustering)
- Custom interactive features (beyond requirements)
- Exceptional design and branding
- Video presentation (5-min walkthrough)
- Deployed dashboard (publicly accessible)

Evaluation Criteria:

Technical Excellence (40%):

- Code quality and organization
- Proper use of libraries
- Reproducibility
- Error handling

Analytical Depth (30%):

- Insight quality
- Statistical rigor
- Question formulation
- Pattern identification

Visual Communication (20%):

- Chart appropriateness
- Design quality
- Accessibility
- Information density

Business Value (10%):

- Actionable recommendations
- Stakeholder alignment
- Clear impact
- Implementation feasibility

Common Mistakes to Avoid:

- Generic analysis (not specific to business context)
- Chart overload (quantity over quality)
- Missing business interpretation
- Poor code documentation
- Inconsistent styling across deliverables
- Recommendations not tied to data
- Non-reproducible analysis

Tips for Success:

1. **Start Early:** This is comprehensive project
2. **Choose Appropriate Data:** Not too simple, not too complex
3. **Tell a Story:** Connect all pieces with narrative
4. **Iterate:** Review and refine visualizations

5. **Test Everything:** Run code fresh, open HTML in browser
6. **Get Feedback:** Show drafts to classmates
7. **Document Well:** Future you will thank you
8. **Focus on Impact:** What decisions will this drive?

Submission Instructions:

- Create a ZIP file with all deliverables
- Name: `FinalProject_YourName.zip`
- Submit via course platform by deadline
- Ensure all files are included and functional

Academic Integrity:

- All work must be your own
- Cite any external data sources
- Do not share code with classmates
- Collaboration on concepts OK, code must be individual

** CONGRATULATIONS!**

Upon completing this project, you will have demonstrated:

Proficiency in Python for data analysis   Mastery of Pandas for data manipulation   Expertise in Matplotlib and Seaborn for static visualization   Advanced skills in Plotly for interactive dashboards   Ability to communicate data insights to business audiences   End-to-end project execution capability

You are now prepared for professional roles in:

- Business Intelligence Analysis
- Data Visualization Specialist
- Business Analytics
- Data Science (with additional learning)
- Management Consulting (analytical roles)