PRACTICAL COURSE ON

Python with Statistics

MORNING SESSION HANDBOOK

Part 2: Data Wrangling with Pandas

⏰ Time: 11:15 AM - 12:45 PM (90 minutes)

Module 2: Data Wrangling with Pandas

What Is Pandas?

Pandas is Python's most powerful library for working with data. Think of it as Excel, but programmable. Instead of clicking and dragging, you write code to:

- Load data from CSV, Excel, databases, and URLs
- Explore and summarize data instantly
- Filter, sort, and transform data
- Group data and calculate statistics
- Handle missing values automatically
- Create visualizations

The core data structure in Pandas is the DataFrame: a table with rows and columns, just like a spreadsheet.

2.1 Loading Data

Importing Pandas

Every time you start a new notebook, import Pandas first:

📋 **Standard import:**

```
import pandas as pd
```

💡 **TIP:** The 'as pd' means we can type 'pd' instead of 'pandas' throughout our code. This is the universal convention—everyone uses 'pd'.

Method 1: Load from Built-in Datasets

Seaborn (a visualization library) includes free practice datasets. Perfect for learning!

📋 **Load built-in dataset:**

```
import pandas as pd
import seaborn as sns

# Load the 'tips' dataset (restaurant tipping data)
df = sns.load_dataset('tips')

print(df.head())  # Show first 5 rows
```

Output:

```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
1       10.34  1.66    Male     No  Sun  Dinner     3
2       21.01  3.50    Male     No  Sun  Dinner     3
3       23.68  3.31    Male     No  Sun  Dinner     2
4       24.59  3.61  Female     No  Sun  Dinner     4
```

💡 **TIP:** The variable name 'df' stands for DataFrame. This is the universal convention.

Common Built-in Datasets

Dataset Name	Description
tips	Restaurant tipping behavior
titanic	Titanic passenger survival
iris	Flower measurements (3 species)
penguins	Penguin measurements
diamonds	Diamond prices and characteristics

Method 2: Load from CSV File

📋 **Load CSV:**

```
import pandas as pd

# From a file on your computer
df = pd.read_csv('data.csv')

# From a URL (works great in Colab!)
url = 'https://raw.githubusercontent.com/mwaskom/seaborn-
data/master/tips.csv'
df = pd.read_csv(url)

print(df.head())
```

Method 3: Load from Excel

📋 **Load Excel:**

```
import pandas as pd

# Load from Excel file
df = pd.read_excel('report.xlsx', sheet_name='Sheet1')

print(df.head())
```

💡 **TIP:** For Excel files, you might need to install openpyxl first: !pip install openpyxl

Method 4: Create from Scratch

📋 **Manual creation:**

```
import pandas as pd

# Create a dictionary with your data
data = {
    'Student': ['Ali', 'Siti', 'Raj', 'Lin', 'Ahmad'],
    'Score':   [85,    72,     90,    68,    78],
    'Hours':   [6,     5,      9,     3,     7]
}

# Convert to DataFrame
df = pd.DataFrame(data)
print(df)
```

Output:

```
  Student  Score  Hours
0     Ali     85      6
1    Siti     72      5
2     Raj     90      9
3     Lin     68      3
4   Ahmad     78      7
```

2.2 Exploring Your Data — The Essential 5 Commands

Whenever you load a new dataset, ALWAYS run these 5 commands first. They tell you everything you need to know about the data structure.

Command 1: .head() — See First Few Rows

📋 **View the data:**

```
import pandas as pd
import seaborn as sns

df = sns.load_dataset('tips')
print(df.head())  # Default: first 5 rows

# Or specify how many rows
print(df.head(10))  # First 10 rows
```

Command 2: .shape — Get Dimensions

📋 **How big is this dataset?**

```
df = sns.load_dataset('tips')
print(df.shape)
```

Output:

```
(244, 7)
```

Interpretation: 244 rows, 7 columns

Command 3: .info() — Column Details

📋 **What's in each column?**

```
df = sns.load_dataset('tips')
print(df.info())
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   total_bill  244 non-null    float64
 1   tip         244 non-null    float64
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64
```

What this tells you:

- Column names: total_bill, tip, sex, smoker, day, time, size
- Data types: float64 (decimals), category (text), int64 (integers)

- Non-Null Count: All 244 rows have values (no missing data)

Command 4: .describe() — Statistical Summary

📋 **Instant statistics:**

```
df = sns.load_dataset('tips')
print(df.describe())
```

Output:

```
       total_bill         tip        size
count  244.000000  244.000000  244.000000
mean    19.785943    2.998279    2.569672
std      8.902412    1.383638    0.951100
min      3.070000    1.000000    1.000000
25%     13.347500    2.000000    2.000000
50%     17.795000    2.900000    2.000000
75%     24.127500    3.562500    3.000000
max     50.810000   10.000000    6.000000
```

Reads like:

- count: How many values (244 bills)
- mean: Average bill is $19.79, average tip is $2.99
- std: Standard deviation (spread of values)
- min/max: Smallest and largest values
- 25%/50%/75%: Quartiles (dividing data into quarters)

Command 5: .isnull().sum() — Find Missing Values

📋 **Check for missing data:**

```
df = sns.load_dataset('tips')
print(df.isnull().sum())
```

Output:

```
total_bill    0
tip           0
sex           0
smoker        0
day           0
time          0
size          0
dtype: int64
```

Interpretation: Zero missing values in all columns. Clean data!

💡 **TIP:** ALWAYS check for missing values. Real-world data is messy. You need to know what's missing before you start analysis.

✓ CHECKPOINT EXERCISE 10

Load the 'penguins' dataset using Seaborn.
Then run all 5 exploration commands:
1. .head()

2. .shape
3. .info()
4. .describe()
5. .isnull().sum()

Answer: How many penguins are in the dataset? Are there any missing values?
Hint: df = sns.load_dataset('penguins')

2.3 Selecting and Filtering Data

Selecting One Column

📋 **Get a single column:**

```
df = sns.load_dataset('tips')

# Select the 'tip' column
tips = df['tip']
print(tips.head())
```

Output:

```
0    1.01
1    1.66
2    3.50
3    3.31
4    3.61
Name: tip, dtype: float64
```

💡 **TIP:** Selecting one column gives you a Series (like a single-column table). Selecting multiple columns gives you a DataFrame (multi-column table).

Selecting Multiple Columns

📋 **Get multiple columns:**

```
df = sns.load_dataset('tips')

# Select specific columns (double brackets!)
subset = df[['total_bill', 'tip', 'sex']]
print(subset.head())
```

Output:

```
   total_bill   tip     sex
0       16.99  1.01  Female
1       10.34  1.66    Male
2       21.01  3.50    Male
3       23.68  3.31    Male
4       24.59  3.61  Female
```

⚠️ **COMMON MISTAKE:** Forgetting the double brackets for multiple columns. df['tip', 'total_bill'] will error. Must use df[['tip', 'total_bill']]

Filtering Rows by Condition

📋 **Filter with condition:**

```
df = sns.load_dataset('tips')

# Get only dinner bills
dinner = df[df['time'] == 'Dinner']
print(dinner.head())
print('Dinner count:', len(dinner))
```

Output:

```
   total_bill   tip     sex smoker  day    time  size
0       16.99  1.01  Female     No  Sun  Dinner     2
```

```
1        10.34  1.66     Male     No   Sun   Dinner     3
2        21.01  3.50     Male     No   Sun   Dinner     3
...
Dinner count: 176
```

Multiple Conditions (AND)

📋 **Combine conditions with &:**

```
df = sns.load_dataset('tips')

# Dinner bills AND total > $30
big_dinners = df[(df['time'] == 'Dinner') & (df['total_bill'] > 30)]
print(big_dinners.head())
print('Count:', len(big_dinners))
```

⚠️ **COMMON MISTAKE:** Using 'and' instead of '&'. In Pandas, use & for AND, | for OR. Also, each condition MUST be in parentheses!

Multiple Conditions (OR)

📋 **Combine with |:**

```
df = sns.load_dataset('tips')

# Bills on Saturday OR Sunday
weekend = df[(df['day'] == 'Sat') | (df['day'] == 'Sun')]
print('Weekend bills:', len(weekend))
```

Output:

```
Weekend bills: 163
```

Numeric Comparisons

Operator	Meaning & Example
==	Equal to: df[df['size'] == 2]
!=	Not equal: df[df['smoker'] != 'Yes']
>	Greater than: df[df['tip'] > 5]
>=	Greater or equal: df[df['total_bill'] >= 20]
<	Less than: df[df['size'] < 3]
<=	Less or equal: df[df['tip'] <= 2]

✓ CHECKPOINT EXERCISE 11

Using the tips dataset:
1. Filter for bills where the tip was greater than $5
2. Print how many such bills exist
3. Print the average total_bill for these high-tip meals

💡 *Hint:* high_tips = df[df['tip'] > 5], then len(high_tips) and high_tips['total_bill'].mean()

2.4 Creating New Columns

Simple Calculations

📋 **Add a new column:**

```
df = sns.load_dataset('tips')

# Calculate tip as percentage of bill
df['tip_percent'] = (df['tip'] / df['total_bill']) * 100

print(df[['total_bill', 'tip', 'tip_percent']].head())
```

Output:

```
   total_bill   tip  tip_percent
0       16.99  1.01     5.944673
1       10.34  1.66    16.054159
2       21.01  3.50    16.658734
3       23.68  3.31    13.978041
4       24.59  3.61    14.680765
```

Conditional Columns

📋 **Categorize based on condition:**

```
df = sns.load_dataset('tips')

# Create generous/standard tipper category
df['tipper_category'] = 'Standard'  # Default value
df.loc[df['tip'] > 5, 'tipper_category'] = 'Generous'

print(df[['tip', 'tipper_category']].head(10))
```

Output:

```
    tip tipper_category
0  1.01         Standard
1  1.66         Standard
2  3.50         Standard
3  3.31         Standard
4  3.61         Standard
5  4.71         Standard
6  2.00         Standard
7  3.12         Standard
8  1.96         Standard
9  3.23         Standard
```

2.5 Grouping and Aggregating

This is the MOST POWERFUL Pandas operation. Grouping lets you split data into categories and calculate statistics for each group.

Basic Grouping

📋 **Group by one column:**

```
df = sns.load_dataset('tips')

# Average tip by day
avg_by_day = df.groupby('day')['tip'].mean()
print(avg_by_day)
```

Output:

```
day
Thur    2.771452
Fri     2.734737
Sat     2.993103
Sun     3.255132
Name: tip, dtype: float64
```

Interpretation: Sunday diners tip highest on average ($3.26)

Multiple Statistics

📋 **Multiple aggregations:**

```
df = sns.load_dataset('tips')

# Get mean, median, and std for each day
stats_by_day = df.groupby('day')['tip'].agg(['mean', 'median', 'std',
'count'])
print(stats_by_day)
```

Output:

```
          mean  median       std  count
day
Thur  2.771452    2.50  1.350199     62
Fri   2.734737    2.50  1.172670     19
Sat   2.993103    2.87  1.462465     87
Sun   3.255132    3.00  1.503923     76
```

Grouping by Multiple Columns

📋 **Multiple group-by columns:**

```
df = sns.load_dataset('tips')

# Average tip by day AND time
avg_by_day_time = df.groupby(['day', 'time'])['tip'].mean()
print(avg_by_day_time)
```

Output:

```
day   time
```

```
Thur  Lunch     2.767705
      Dinner    3.000000
Fri   Lunch     2.382857
      Dinner    2.940000
Sat   Dinner    2.993103
Sun   Dinner    3.255132
Name: tip, dtype: float64
```

Grouping with Multiple Columns and Stats

📋 **Complete analysis:**

```
df = sns.load_dataset('tips')

# Compare smokers vs non-smokers by day
comparison = df.groupby(['day', 'smoker'])['total_bill'].agg(['mean',
'count'])
print(comparison)
```

Output:

```
                  mean  count
day  smoker
Thur Yes     19.190588     17
     No      17.113111     45
Fri  Yes     16.813333      6
     No      18.420000     13
Sat  Yes     21.276667     30
     No      19.661778     57
Sun  Yes     24.120000     25
     No      20.506667     51
```

💡 **TIP:** .groupby() is how you answer questions like: 'What's the average score by gender?', 'How does revenue differ by region?', 'Which treatment group had better outcomes?'

✓ CHECKPOINT EXERCISE 12

Using the tips dataset:
1. Group by 'sex' (Male/Female)
2. Calculate the mean and count of 'total_bill' for each group
3. Who spends more on average, male or female diners?

💡 *Hint:* df.groupby('sex')['total_bill'].agg(['mean', 'count'])

2.6 Basic Visualization with Seaborn

Visualization helps you SEE patterns in data. Seaborn makes it incredibly easy—most charts are just 1-2 lines of code.

Setup

📋 **Import visualization libraries:**

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

df = sns.load_dataset('tips')
```

Histogram — Distribution of One Variable

📋 **Create histogram:**

```
# Show distribution of total bills
sns.histplot(data=df, x='total_bill', bins=20, kde=True)
plt.title('Distribution of Total Bills')
plt.show()
```

What it shows: Most bills are between $10-$25. Few bills exceed $40.

💡 **TIP:** bins=20 controls how many bars. kde=True adds a smooth curve showing the distribution shape.

Box Plot — Compare Groups

📋 **Create box plot:**

```
# Compare tip amounts by day
sns.boxplot(data=df, x='day', y='tip')
plt.title('Tip Amount by Day of Week')
plt.show()
```

What it shows:

- The box shows the middle 50% of tips
- The line inside the box is the median
- Dots are outliers (unusually high/low tips)

Bar Chart — Category Counts

📋 **Create bar chart:**

```
# Count how many bills on each day
sns.countplot(data=df, x='day')
plt.title('Number of Bills by Day')
plt.show()
```

Scatter Plot — Relationship Between Two Variables

📋 **Create scatter plot:**

```
# Relationship between bill total and tip
sns.scatterplot(data=df, x='total_bill', y='tip')
plt.title('Relationship: Total Bill vs Tip')
plt.show()
```

What it shows: Positive relationship—higher bills tend to have higher tips.

Colored by Category

📋 **Add color for groups:**

```
# Scatter plot colored by time (Lunch/Dinner)
sns.scatterplot(data=df, x='total_bill', y='tip', hue='time')
plt.title('Bill vs Tip, colored by Time of Day')
plt.show()
```

💡 **TIP:** The 'hue' parameter colors points by a categorical variable. Great for comparing groups visually.

Chart Customization Basics

📋 **Customize appearance:**

```
# Create histogram with customization
sns.histplot(data=df, x='total_bill', bins=25, color='skyblue',
edgecolor='black')
plt.title('Distribution of Total Bills', fontsize=14,
fontweight='bold')
plt.xlabel('Bill Amount ($)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.show()
```

Quick Reference: Common Charts

Chart Type	When to Use
sns.histplot()	Distribution of one numeric variable
sns.boxplot()	Compare numeric variable across categories
sns.violinplot()	Like boxplot but shows full distribution shape
sns.countplot()	Count of items in each category
sns.scatterplot()	Relationship between two numeric variables
sns.barplot()	Average of numeric variable by category

✓ CHECKPOINT EXERCISE 13

Using the tips dataset:
1. Create a box plot comparing 'total_bill' by 'time' (Lunch vs Dinner)
2. Add a title: 'Bill Amount by Time of Day'
3. What do you observe about lunch vs dinner bills?

💡 *Hint:* sns.boxplot(data=df, x='time', y='total_bill')

2.7 Putting It All Together: Complete Workflow

Here's a typical data analysis workflow combining everything you learned:

📋 **Complete Analysis:**

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

# ---- STEP 1: LOAD DATA ----
df = sns.load_dataset('tips')

# ---- STEP 2: EXPLORE ----
print("Dataset shape:", df.shape)
print("\nFirst few rows:")
print(df.head())
print("\nMissing values:")
print(df.isnull().sum())
print("\nStatistical summary:")
print(df.describe())

# ---- STEP 3: CREATE NEW COLUMN ----
df['tip_percent'] = (df['tip'] / df['total_bill']) * 100

# ---- STEP 4: FILTER DATA ----
dinner_data = df[df['time'] == 'Dinner']
print("\nDinner meals count:", len(dinner_data))

# ---- STEP 5: GROUP & ANALYZE ----
avg_by_day = dinner_data.groupby('day')['tip_percent'].agg(['mean',
'count'])
print("\nAverage tip percentage by day (Dinner only):")
print(avg_by_day)

# ---- STEP 6: VISUALIZE ----
sns.boxplot(data=dinner_data, x='day', y='tip_percent')
plt.title('Tip Percentage by Day (Dinner Only)')
plt.ylabel('Tip Percentage (%)')
plt.show()

sns.scatterplot(data=dinner_data, x='total_bill', y='tip', hue='sex')
plt.title('Bill vs Tip at Dinner (by Gender)')
plt.show()
```

Final Checkpoint: Can You Do This?

✓ CHECKPOINT EXERCISE 14

Complete Pandas Workflow Challenge:

Load the 'penguins' dataset and perform this analysis:

1. Check the shape and missing values
2. Filter for only 'Adelie' species
3. Create a new column: body_mass_kg = body_mass_g / 1000
4. Group by 'sex' and calculate mean body_mass_kg

5. Create a box plot: body_mass_kg by sex

This tests everything you learned in Module 2!
Hint: Follow the 6-step workflow above. Load penguins, explore, filter, create column, group, visualize.

Module 2 Summary: Key Takeaways

Congratulations! You now know the essential Pandas operations for data analysis. Here's what you learned:

Core Skills

- Load data from CSV, Excel, URLs, and built-in datasets
- Explore data with .head(), .shape, .info(), .describe(), .isnull().sum()
- Select specific columns and filter rows by conditions
- Create new calculated columns
- Group data and calculate statistics by category
- Create visualizations to see patterns

Most Important Commands

Command	Purpose
pd.read_csv()	Load data
df.head()	See first rows
df[df['col'] > 50]	Filter rows
df.groupby('cat')['val'].mean()	Group and aggregate
sns.boxplot()	Visualize comparisons

Common Mistakes to Avoid

- Forgetting to import: import pandas as pd, import seaborn as sns
- Single brackets for multiple columns: df['a', 'b'] → Use df[['a', 'b']]
- Using 'and' instead of & in conditions: (condition1) and (condition2) → (condition1) & (condition2)
- Forgetting parentheses around each condition when combining
- Not checking for missing values before analysis

What's Next?

After lunch, you'll apply everything you learned this morning to real-world case studies. You'll use Pandas to:

- Explore a complete dataset from start to finish
- Calculate descriptive statistics and interpret results
- Run hypothesis tests to answer business questions
- Create professional visualizations for stakeholders

Take a break, grab lunch, and get ready for hands-on data analysis!