

Python Bootcamp - Session 3

Practical Python Programming

Hrishikesh Terdalkar

November 30, 2025

Session Overview

Session 3 moves beyond syntax drills and focuses on the practical workflows you need for day-to-day research. Each module demonstrates how to collect data, keep it tidy, and automate common analyses so results are reproducible. The emphasis throughout is on hands-on practice with file handling, command-line tooling, and batch processing.

What You Will Build

- A small, reproducible dataset pack for hands-on analysis.
- A batch analysis CLI that scans folders and emits JSON summaries.
- Two analyzers (basic/advanced) that compute stats, detect trends, and optionally plot.
- Cleaned CSV and JSON artefacts ready to reuse in Session 4.

Session Plan

- Environment check and data generation.
- File operations lab (manual vs pandas) and JSON metadata.
- Batch CLI lab (discover, process, summarise).
- Basic analyzer lab (descriptive statistics + outputs).
- Advanced analyzer lab (filters, trends, correlations, plots).
- Wrap-up, reproducibility notes, and preview of Session 4.

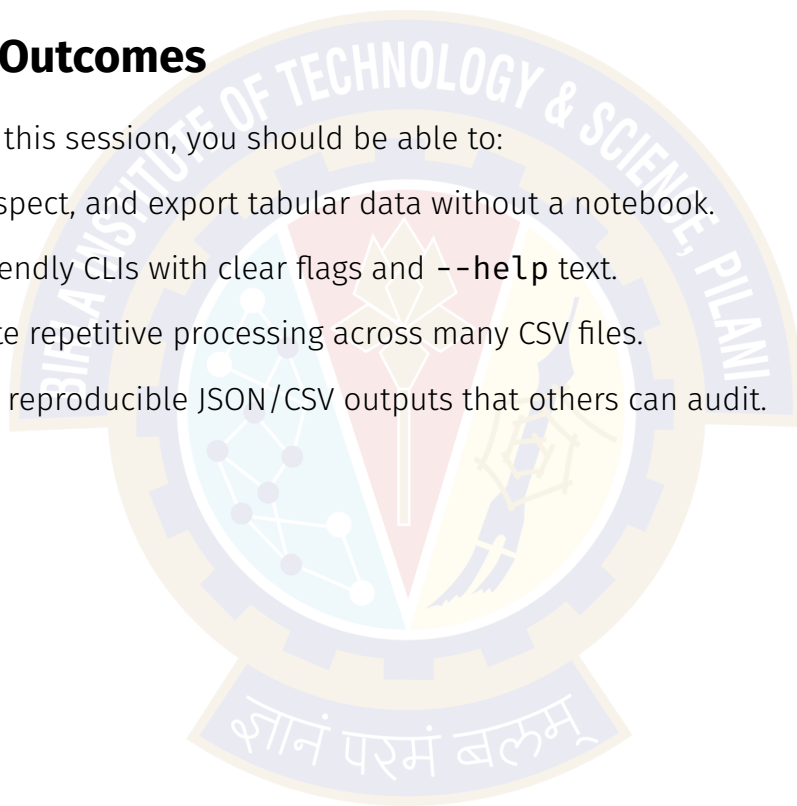
Skills You'll Practice

- Structure Python programs so future-you can understand them at a glance.
- Compare manual and pandas-based file handling and know when to reach for each.
- Create reusable helper functions and document them via docstrings.
- Work with CSV and JSON without having to open Excel or a browser.
- Build approachable CLIs with argparse, including help text and defaults.
- Automate batch processing so routine analysis takes seconds, not evenings.

Learning Outcomes

By the end of this session, you should be able to:

- Load, inspect, and export tabular data without a notebook.
- Write friendly CLIs with clear flags and `--help` text.
- Automate repetitive processing across many CSV files.
- Produce reproducible JSON/CSV outputs that others can audit.



Theory Essentials

- **CSV vs JSON:** CSV is row/column data for tables; JSON is hierarchical and ideal for metadata and configuration.
- **Pandas DataFrames:** Think of them as spreadsheets with powerful indexing and vectorised operations.
- **CLI design:** Prefer explicit flags (e.g. `--input`, `--output`) and helpful `--help` messages.
- **Descriptive statistics:** Mean, median, standard deviation, and range establish baselines before advanced modelling.
- **Trends & correlations:** Linear fits and correlation matrices are quick sanity checks for relationships between variables.

Quick checks you can run on any numeric CSV:

```
1 import pandas as pd
2 df = pd.read_csv("engineering_test_data.csv")
3 print(df.describe())           # central tendency + spread
4 print(df.corr(numeric_only=True)) # relationships between variables
```

Listing 1: Fast EDA checklist

Prerequisites and Setup

Use a managed environment so dependencies stay isolated. If you have Conda/-Mamba, this is the simplest route:

```
1 # with conda
2 conda create -n pybootcamp python=3.10 -y
3 conda activate pybootcamp
4
5 # or with mamba
6 mamba create -n pybootcamp python=3.10 -y
7 mamba activate pybootcamp
8
9 pip install -r requirements.txt
10 make data # generate shared CSVs for this session
```

Listing 2: Recommended: Conda/Mamba environment

If you prefer, a standard virtualenv also works; the scripts themselves don't depend on Conda-specific features.

Reproducibility

Synthetic data and randomised examples are seeded so that every run produces the same values. If you want different noise each time, change the seed once at the top of the script.

```
1 import numpy as np
2 RNG_SEED = 42
3 np.random.seed(RNG_SEED)
```

Listing 3: Deterministic randomness

Data Dictionary

The main CSV produced by the data generator is `engineering_test_data.csv`.

Column	Meaning
Time_min	Elapsed time in minutes from experiment start
Temperature_C	Temperature in degrees Celsius
Pressure_kPa	Pressure in kilopascals
Flow_Rate_Lmin	Volumetric flow rate in litres per minute
Vibration_mm	Peak vibration amplitude in millimetres

How to Run the Examples

1. Generate datasets: make `data` or `python session3/01_create_test_data.py`.
2. Explore file operations: open `session3/02_file_operations.py` and run the main function.
3. Automate batch analysis: run the batch processor with `--summary` and `--verbose`.
4. Compare basic and advanced analyzers on the same CSV; try adding `--plot` and a simple filter.

Practice Exercises

1. Add a new column to `engineering_test_data.csv` (e.g. a moving average of temperature) and update the analyzers to compute its statistics.
2. Modify the batch processor so it also saves a compact Markdown summary next to each JSON report.
3. Extend the advanced analyzer filter to support expressions like `"Temperature_C between 22 and 30"`.

Lab Guide

0.1 01_create_test_data.py

- Produces the canonical `engineering_test_data.csv`.
- Builds an `experiments/` folder with multiple experiment types for batch tests.
- Can be rerun at any point to reset the workspace.

0.2 o2_file_operations.py

- Generates tidy time/temperature/pressure lists.
- Saves/loads CSV data both manually (`csv.writer`) and via pandas.
- Stores experiment metadata in a JSON file for later reuse.

0.3 o3_batch_processor.py

- Discovers CSV files with `glob` and processes them with pandas.
- Uses `argparse` switches such as `--summary` and `--verbose`, reinforcing good CLI hygiene.
- Emits per-experiment JSON plus an optional batch summary for a quick debrief e-mail.

A representative CLI skeleton:

```
1 def main():
2     parser = build_parser()
3     args = parser.parse_args()
4
5     if not os.path.exists(args.input_dir):
6         print(f"Error: Input directory '{args.input_dir}' not found")
7         sys.exit(1)
8
9     os.makedirs(args.output_dir, exist_ok=True)
10    experiment_files = find_experiment_files(
11        args.input_dir, args.file_pattern, verbose=args.verbose
12    )
13
14    all_results = []
15    for path in experiment_files:
16        result = process_single_experiment(
17            path, args.output_dir, verbose=args.verbose
18        )
19        all_results.append(result)
20
21    if args.summary:
22        generate_batch_summary(all_results, args.output_dir)
23
24 if __name__ == "__main__":
25     main()
```

Listing 4: Command-line entry point pattern

0.4 o4_basic_analyzer.py

- Reads a single CSV (typically `engineering_test_data.csv`).
- Computes mean/median/std/min/max/range for every numeric column.
- Prints a lightweight summary and stores it as `analysis.json`.

0.5 o5_advanced_analyzer.py

- Adds trend analysis, correlations, filtering, and optional plotting.
- Supports data export (`--export-csv`) so plots and processed data stay in sync.
- Keeps the CLI style identical to the basic analyzer, making upgrades painless for end users.
- Demonstrates how to extend your tooling without rewriting everything from scratch.

Best Practices

0.6 Error Handling

- Wrap file I/O in `try/except` blocks and print clear messages that include the path.
- Validate CLI inputs (paths, flags, column names) early; exit with non-zero status on errors.
- Prefer explicit defaults and `-v/--verbose` to make scripts self-explanatory during runs.

0.7 Documentation

- Add docstrings with one-line purpose and key parameters/returns.
- Provide `-help` examples in argparse epilogues so learners can copy/paste.
- Use type hints for function signatures to aid beginners and editors.

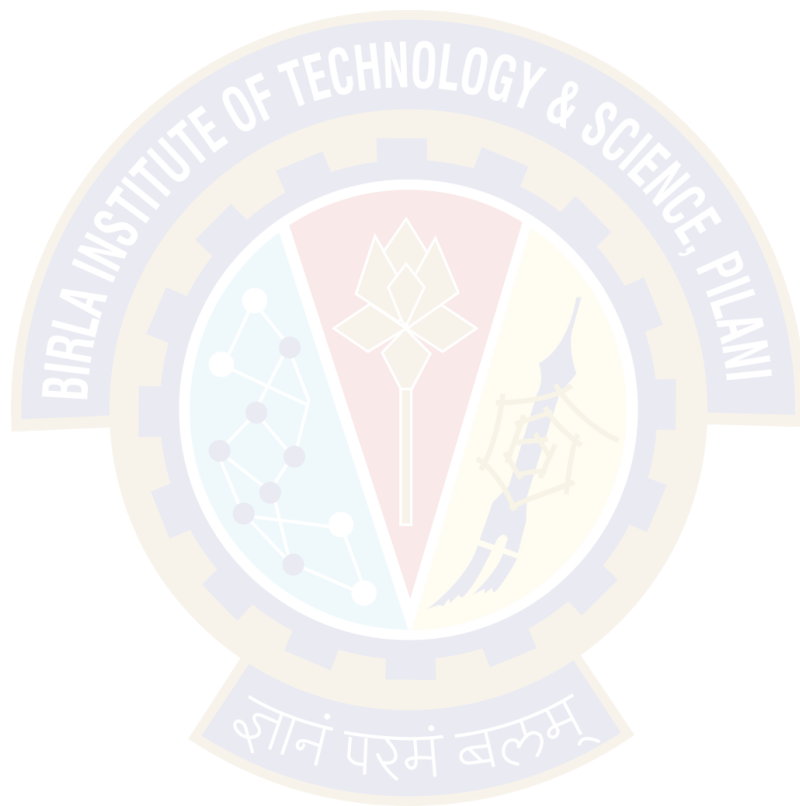
Common Pitfalls

- **CSV headers:** Ensure column names match exactly (case and underscores) when filtering or plotting.
- **Paths:** Prefer relative paths so your scripts work regardless of the working directory.
- **Verbose mode:** Use `-v` to reveal what a script is doing before you trust its outputs.

Further Reading

- Official Python tutorial: <https://docs.python.org/3/tutorial/>
- Gentle Python overview: <https://www.w3schools.com/python/>

- Practical examples and articles: <https://realpython.com/>
- pandas documentation (data analysis): <https://pandas.pydata.org/docs/>



Appendix: Full Code Listings

session3/o1_create_test_data.py

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Test Data Creation (used by many examples)
5
6  Creates one main CSV file plus a small folder of experiment
7  files.
8
9  Note:
10 - Do not worry about the mathematics.
11 - Treat this script purely as a data generator for the rest of the session.
12
13 @author: Hrishikesh Terdalkar
14 """
15
16 #####
17
18 import os
19
20 import pandas as pd
21 import numpy as np
22
23 RNG_SEED = 42
24 np.random.seed(RNG_SEED)
25
26 #####
27
28
29 def create_engineering_test_data():
30     """Create realistic engineering test data"""
31     # Time points (0 to 120 minutes, every 5 minutes)
32     time_points = np.arange(0, 121, 5)
33
34     # Simulate different engineering parameters with realistic patterns
35     temperature = (
36         25
37         + 8 * np.sin(2 * np.pi * time_points / 60)
38         + np.random.normal(0, 1, len(time_points))
39     )
40     pressure = (
41         101.3
42         + 2 * np.cos(2 * np.pi * time_points / 40)
43         + np.random.normal(0, 0.2, len(time_points))
44     )
45     flow_rate = (
46         5
47         + 2 * np.sin(2 * np.pi * time_points / 30)
48         + np.random.normal(0, 0.1, len(time_points))
49     )
50     vibration = (
51         0.1
52         + 0.05 * np.sin(2 * np.pi * time_points / 20)
53         + np.random.normal(0, 0.01, len(time_points))
54     )
55
56     df = pd.DataFrame(
57         {
58             "Time_min": time_points,
59             "Temperature_C": temperature,
60             "Pressure_kPa": pressure,
61             "Flow_Rate_Lmin": flow_rate,
62             "Vibration_mm": vibration,
63         }
64     )
65
66     return df
67
68
```



```

69 def create_multiple_experiments():
70     """Create multiple experiment files for batch processing"""
71     experiments_dir = "experiments"
72     os.makedirs(experiments_dir, exist_ok=True)
73
74     experiment_types = [
75         ("thermal_study", "Temperature", "C", 20, 35),
76         ("pressure_test", "Pressure", "kPa", 100, 110),
77         ("flow_analysis", "FlowRate", "L/min", 1, 10),
78         ("vibration_test", "Vibration", "mm", 0.05, 0.15),
79     ]
80
81     for exp_type, param, unit, min_val, max_val in experiment_types:
82         for i in range(1, 4): # Create 3 of each type
83             exp_name = f"{exp_type}_{i:02d}"
84             time_points = np.arange(0, 61, 2) # 1 hour of data
85
86             # Different patterns for different experiments
87             if exp_type == "thermal_study":
88                 pattern = np.sin(2 * np.pi * time_points / 30)
89             elif exp_type == "pressure_test":
90                 pattern = np.cos(2 * np.pi * time_points / 20)
91             elif exp_type == "flow_analysis":
92                 pattern = np.sin(2 * np.pi * time_points / 15)
93             else: # vibration_test
94                 pattern = np.sin(2 * np.pi * time_points / 10)
95
96             measurements = min_val + (max_val - min_val) * pattern
97             measurements += np.random.normal(
98                 0, (max_val - min_val) * 0.05, len(time_points)
99             )
100
101             df = pd.DataFrame(
102                 {
103                     "Time_min": time_points,
104                     f"{param}_{unit}": measurements,
105                     "Experiment_ID": exp_name,
106                 }
107             )
108
109             filename = os.path.join(experiments_dir, f"{exp_name}.csv")
110             df.to_csv(filename, index=False)
111             print(f"Created: {filename}")
112
113
114 def main():
115     """Main function to create test data"""
116     print("Creating test data for Session 3 examples...")
117
118     # Create single comprehensive test file
119     comprehensive_data = create_engineering_test_data()
120     comprehensive_data.to_csv("engineering_test_data.csv", index=False)
121     print("Created: engineering_test_data.csv")
122
123     # Create multiple experiment files for batch processing
124     create_multiple_experiments()
125     print("Created multiple experiment files in 'experiments/' directory")
126
127     print("\nTest data creation complete!")
128     print("\nNext steps (common scripts):")
129     print(
130         "    python session3/03_batch_processor.py --input-dir experiments --output-dir\n"
131         "    batch_results --summary"
132     )
133     print(
134         "    python session3/04_basic_analyzer.py engineering_test_data.csv --stats --verbose"
135     )
136     print(
137         "    python session3/05_advanced_analyzer.py -i engineering_test_data.csv --stats --\n"
138         "    trends --plot"
139     )
140
141 if __name__ == "__main__":

```

```
141 main()
```

Listing 5: Reproducible test data generator

session3/o2_file_operations.py

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  File Operations Module
5  Basic file handling and data organization functions
6
7  @author: Hrishikesh Terdalkar
8  """
9
10 #####
11
12 import os
13 import csv
14 import json
15 from datetime import datetime
16
17 import pandas as pd
18
19 #####
20
21
22 def create_sample_experiment_data():
23     """Create sample experimental data for demonstration"""
24     # Temperature and pressure readings over time
25     time_points = [0, 5, 10, 15, 20, 25, 30] # minutes
26     temperatures = [25.0, 26.5, 28.2, 27.8, 26.9, 25.5, 24.8] # deg C
27     pressures = [101.3, 101.5, 101.8, 101.6, 101.4, 101.2, 101.1] # kPa
28
29     return time_points, temperatures, pressures
30
31
32 def save_data_to_csv(filename, times, temps, pressures):
33     """Save experimental data to CSV file manually"""
34     with open(filename, "w", newline="") as csvfile:
35         writer = csv.writer(csvfile)
36         # Write header
37         writer.writerow(["Time_min", "Temperature_C", "Pressure_kPa"])
38         # Write data rows
39         for i in range(len(times)):
40             writer.writerow([times[i], temps[i], pressures[i]])
41     print(f"Data saved to {filename}")
42
43
44 def read_data_from_csv(filename):
45     """Read experimental data from CSV file manually"""
46     times, temps, pressures = [], [], []
47
48     with open(filename, "r") as csvfile:
49         reader = csv.reader(csvfile)
50         next(reader) # Skip header
51         for row in reader:
52             times.append(float(row[0]))
53             temps.append(float(row[1]))
54             pressures.append(float(row[2]))
55
56     return times, temps, pressures
57
58
59 def use_pandas_for_data_handling(times, temps, pressures):
60     """Demonstrate pandas for easier data handling"""
61     # Create DataFrame
62     df = pd.DataFrame(
63         {"Time_min": times, "Temperature_C": temps, "Pressure_kPa": pressures}
64     )
65

```

```

66     print("DataFrame created:")
67     print(df)
68
69     # Basic statistics
70     print("\nBasic statistics:")
71     print(df.describe())
72
73     return df
74
75
76 def handle_json_configuration():
77     """Work with JSON files for experiment configuration"""
78     experiment_info = {
79         "experiment_id": "thermal_study_001",
80         "researcher": "PhD Student",
81         "date": datetime.now().isoformat(),
82         "equipment": {
83             "sensor": "Thermocouple Type K",
84             "data_logger": "Arduino Uno",
85             "sampling_rate": "1 sample/minute",
86         },
87         "conditions": {"ambient_temperature": 25.0, "humidity": 45.0},
88     }
89
90     # Save to JSON file
91     with open("experiment_config.json", "w") as jsonfile:
92         json.dump(experiment_info, jsonfile, indent=2)
93     print("Experiment configuration saved to JSON")
94
95     # Read from JSON file
96     with open("experiment_config.json", "r") as jsonfile:
97         loaded_config = json.load(jsonfile)
98
99     print("\nExperiment configuration loaded:")
100    for key, value in loaded_config.items():
101        print(f"{key}: {value}")
102
103
104 def demonstrate_file_operations():
105     """Demonstrate all file operations"""
106     print("=== FILE OPERATIONS DEMONSTRATION ===")
107
108     # Create sample data
109     times, temps, pressures = create_sample_experiment_data()
110
111     # Save and read CSV manually
112     save_data_to_csv("manual_data.csv", times, temps, pressures)
113     times_read, temps_read, pressures_read = read_data_from_csv(
114         "manual_data.csv"
115     )
116     print(f>Data read back: {len(times_read)} measurements")
117
118     # Use pandas
119     df = use_pandas_for_data_handling(times, temps, pressures)
120     df.to_csv("pandas_data.csv", index=False)
121     df_read = pd.read_csv("pandas_data.csv")
122     print(f>Pandas data read back: {df_read.shape}")
123
124     # JSON configuration
125     handle_json_configuration()
126
127
128 if __name__ == "__main__":
129     demonstrate_file_operations()

```

Listing 6: File operations and JSON metadata

session3/o3_batch_processor.py

```

1 #!/usr/bin/env python3

```

```

2  # -*- coding: utf-8 -*-
3  """
4  Batch Experiment Processor
5
6  Conceptually simple:
7  - look for CSV files in a folder
8  - compute a few statistics for each file
9  - save one JSON result per file
10 - (optionally) save a compact batch summary
11
12 Everything is written as plain functions so you
13 can read it top to bottom without worrying about
14 classes or advanced patterns.
15
16 @author: Hrishikesh Terdalkar
17 """
18
19 #####
20
21 import os
22 import sys
23 import json
24 import glob
25 import argparse
26 from datetime import datetime
27
28 import numpy as np
29 import pandas as pd
30
31 RNG_SEED = 42
32 np.random.seed(RNG_SEED)
33
34 #####
35
36
37 def build_parser() -> argparse.ArgumentParser:
38     """Setup batch processing parser"""
39     parser = argparse.ArgumentParser(
40         description="Batch Process Multiple Experiments (function-based version)",
41         epilog=(
42             "Example: python session3/o3_batch_processor.py "
43             "--input-dir experiments/ --output-dir results/ --summary"
44         ),
45     )
46
47     parser.add_argument(
48         "--input-dir",
49         "-i",
50         required=True,
51         help="Directory containing experiment CSV files",
52     )
53     parser.add_argument(
54         "--output-dir",
55         "-o",
56         required=True,
57         help="Output directory for per-file results",
58     )
59     parser.add_argument(
60         "--file-pattern",
61         default="*.csv",
62         help="File pattern to match (default: *.csv)",
63     )
64     parser.add_argument(
65         "--summary", action="store_true", help="Generate summary report"
66     )
67     parser.add_argument(
68         "--verbose", "-v", action="store_true", help="Print progress details"
69     )
70
71     return parser
72
73
74 def find_experiment_files(input_dir: str, pattern: str, verbose: bool = False):
75     """Find all experiment files matching pattern"""
76     search_pattern = os.path.join(input_dir, pattern)

```

```
77     files = sorted(glob.glob(search_pattern))
78
79     if not files:
80         print(f"No files found matching {search_pattern}")
81         return []
82
83     if verbose:
84         print(f"Found {len(files)} experiment files:")
85         for file in files:
86             print(f"    {os.path.basename(file)}")
87
88     return files
89
90
91 def process_single_experiment(file_path: str, output_dir: str, verbose: bool):
92     """Process a single experiment file"""
93     try:
94         exp_name = os.path.splitext(os.path.basename(file_path))[0]
95         exp_output_dir = os.path.join(output_dir, exp_name)
96         os.makedirs(exp_output_dir, exist_ok=True)
97
98         # Load data
99         df = pd.read_csv(file_path)
100
101         # Basic analysis
102         results = {
103             "experiment": exp_name,
104             "file": file_path,
105             "timestamp": datetime.now().isoformat(),
106             "data_shape": df.shape,
107             "columns": list(df.columns),
108         }
109
110         # Calculate metrics for numeric columns only
111         numeric_cols = df.select_dtypes(include=[np.number]).columns
112         metrics = {}
113
114         for col in numeric_cols:
115             metrics[col] = {
116                 "mean": float(df[col].mean()),
117                 "std": float(df[col].std()),
118                 "min": float(df[col].min()),
119                 "max": float(df[col].max()),
120             }
121
122         results["metrics"] = metrics
123
124         # Save results
125         output_file = os.path.join(exp_output_dir, "analysis.json")
126         with open(output_file, "w", encoding="utf-8") as f:
127             json.dump(results, f, indent=2)
128
129         if verbose:
130             print(f"[OK] Processed {exp_name}")
131
132         return results
133
134     except Exception as exc: # pragma: no cover - simple demo
135         print(f"[ERROR] Processing {file_path}: {exc}")
136         return {"error": str(exc), "file": file_path}
137
138
139 def generate_batch_summary(all_results, output_dir: str):
140     """Generate and print a simple summary report"""
141     successful = [r for r in all_results if "error" not in r]
142     failed = [r for r in all_results if "error" in r]
143
144     summary = {
145         "batch_date": datetime.now().isoformat(),
146         "total_files": len(all_results),
147         "successful": len(successful),
148         "failed": len(failed),
149         "experiments": [],
150     }
151
```

```

152     for result in successful:
153         summary["experiments"].append(
154             {
155                 "name": result["experiment"],
156                 "data_points": result["data_shape"][0],
157                 "variables": result["data_shape"][1],
158             }
159         )
160
161     # Save summary
162     summary_file = os.path.join(output_dir, "batch_summary.json")
163     with open(summary_file, "w", encoding="utf-8") as f:
164         json.dump(summary, f, indent=2)
165
166     # Print summary
167     print("\n" + "=" * 50)
168     print("BATCH PROCESSING SUMMARY")
169     print("=" * 50)
170     print(f"Total files: {summary['total_files']}")
171     print(f"Successful: {summary['successful']}")
172     print(f"Failed: {summary['failed']}")
173
174     if successful:
175         print("\nSuccessful experiments:")
176         for exp in summary["experiments"][:5]: # Show first 5
177             print(f"    {exp['name']}: {exp['data_points']} points")
178
179     if failed:
180         print("\nFailed experiments:")
181         for failure in failed[:3]: # Show first 3 failures
182             print(f"    {failure['file']}: {failure['error']}")
183
184
185 def create_test_experiments():
186     """Create test experiment files for demonstration"""
187     test_dir = "test_experiments"
188     os.makedirs(test_dir, exist_ok=True)
189
190     experiments = [
191         ("thermal_001", "Temperature", "C", 20, 35),
192         ("pressure_002", "Pressure", "kPa", 100, 110),
193         ("flow_003", "FlowRate", "L/min", 1, 10),
194     ]
195
196     for exp_name, param, unit, min_val, max_val in experiments:
197         time_points = np.arange(0, 60, 2)
198         measurements = min_val + (max_val - min_val) * np.sin(
199             2 * np.pi * time_points / 30
200         )
201         measurements += np.random.normal(
202             0, (max_val - min_val) * 0.05, len(time_points)
203         )
204
205         df = pd.DataFrame(
206             {"Time_min": time_points, f"{param}_{unit}": measurements}
207         )
208
209         filename = os.path.join(test_dir, f"{exp_name}.csv")
210         df.to_csv(filename, index=False)
211         print(f"Created: {filename}")
212
213     return test_dir
214
215
216 def main():
217     """Main function for the batch processor"""
218     # Create test data if running directly
219     if not os.path.exists("test_experiments"):
220         print("Creating test experiment files...")
221         create_test_experiments()
222
223     parser = build_parser()
224     args = parser.parse_args()
225

```

```

226     if not os.path.exists(args.input_dir):
227         print(f"Error: Input directory '{args.input_dir}' not found")
228         sys.exit(1)
229
230     # Create output directory
231     os.makedirs(args.output_dir, exist_ok=True)
232
233     # Find experiment files
234     experiment_files = find_experiment_files(
235         args.input_dir, args.file_pattern, verbose=args.verbose
236     )
237
238     if not experiment_files:
239         sys.exit(1)
240
241     # Process each experiment
242     all_results = []
243
244     for file_path in experiment_files:
245         result = process_single_experiment(
246             file_path, args.output_dir, verbose=args.verbose
247         )
248         all_results.append(result)
249
250     # Generate summary if requested
251     if args.summary:
252         generate_batch_summary(all_results, args.output_dir)
253
254     print(f"\n[OK] Batch processing complete! Results in {args.output_dir}")
255
256     success = True
257
258     if success:
259         sys.exit(0)
260     else:
261         sys.exit(1)
262
263 if __name__ == "__main__":
264     main()
265

```

Listing 7: Batch processing CLI

session3/o4_basic_analyzer.py

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Basic Research Data Analyzer
5  Simple command-line tool for data analysis
6
7  @author: Hrishikesh Terdalkar
8  """
9
10 #####
11
12 import os
13 import sys
14 import json
15 import argparse
16 from datetime import datetime
17
18 import pandas as pd
19 import numpy as np
20
21 #####
22
23
24 def calculate_basic_stats(data):
25     """Calculate basic statistics for a dataset"""
26     stats = {

```

```
27     "mean": float(np.mean(data)),
28     "median": float(np.median(data)),
29     "std_dev": float(np.std(data)),
30     "min": float(min(data)),
31     "max": float(max(data)),
32     "range": float(max(data) - min(data)),
33 }
34 return stats
35
36
37 def setup_basic_parser():
38     """Setup basic argument parser"""
39     parser = argparse.ArgumentParser(
40         description="Basic Research Data Analyzer",
41         epilog=(
42             "Example: python session3/04_basic_analyzer.py "
43             "engineering_test_data.csv --stats --output results"
44         ),
45     )
46
47     parser.add_argument(
48         "input", help="Input CSV file containing experimental data"
49     )
50     parser.add_argument(
51         "--output",
52         "-o",
53         default="analysis_results",
54         help="Output directory for results",
55     )
56     parser.add_argument(
57         "--stats", action="store_true", help="Calculate basic statistics"
58     )
59     parser.add_argument(
60         "--verbose", "-v", action="store_true", help="Enable verbose output"
61     )
62
63     return parser
64
65
66 def analyze_data(args):
67     """Main analysis function"""
68     # Create output directory
69     os.makedirs(args.output, exist_ok=True)
70
71     try:
72         # Load data
73         df = pd.read_csv(args.input)
74
75         if args.verbose:
76             print(f"Loaded data from {args.input}")
77             print(f>Data shape: {df.shape}")
78             print(f"Columns: {list(df.columns)}")
79
80         results = {
81             "analysis_date": datetime.now().isoformat(),
82             "input_file": args.input,
83             "data_shape": df.shape,
84             "columns": list(df.columns),
85         }
86
87         # Calculate statistics if requested
88         if args.stats:
89             stats_results = {}
90             numeric_cols = df.select_dtypes(include=[np.number]).columns
91
92             for col in numeric_cols:
93                 stats_results[col] = calculate_basic_stats(df[col])
94
95             results["statistics"] = stats_results
96
97             if args.verbose:
98                 print("Calculated statistics for numeric columns")
99
100         # Save results
101         output_file = os.path.join(args.output, "analysis.json")
```



```

102     with open(output_file, "w") as f:
103         json.dump(results, f, indent=2)
104
105     if args.verbose:
106         print(f"Results saved to {output_file}")
107
108     # Print summary
109     print_summary(results)
110
111     return True
112
113 except Exception as e:
114     print(f"Error: {e}")
115     return False
116
117
118 def print_summary(results):
119     """Print analysis summary"""
120     print("\n" + "=" * 50)
121     print("ANALYSIS SUMMARY")
122     print("=" * 50)
123     print(f"Input file: {results['input_file']}")
124     print(f"Data shape: {results['data_shape']}")
125
126     if "statistics" in results:
127         print("\nStatistical Summary:")
128         for col, stats in results["statistics"].items():
129             print(f"    {col}:")
130             print(f"        Mean: {stats['mean']:.2f}")
131             print(f"        Std: {stats['std_dev']:.2f}")
132             print(f"        Range: {stats['min']:.2f} to {stats['max']:.2f}")
133
134
135 def main():
136     """Main function"""
137     parser = setup_basic_parser()
138     args = parser.parse_args()
139
140     # Validate input file exists
141     if not os.path.exists(args.input):
142         print(f"Error: Input file '{args.input}' not found")
143         sys.exit(1)
144
145     # Run analysis
146     success = analyze_data(args)
147
148     if success:
149         print(f"\n[OK] Analysis complete! Results in {args.output}/")
150         sys.exit(0)
151     else:
152         sys.exit(1)
153
154
155 if __name__ == "__main__":
156     main()

```

Listing 8: Basic research analyzer

session3/05_advanced_analyzer.py

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Advanced Research Data Analyzer (optional / stretch)
5
6  This script shows more realistic analysis options
7  for students who are already comfortable with
8  basic Python and pandas. If you are new to Python,
9  you can safely treat this as an optional example.
10

```

```

11 @author: Hrishikesh Terdalkar
12 """
13
14 #####
15
16 import os
17 import sys
18 import json
19 import argparse
20 from datetime import datetime
21
22 import numpy as np
23 import pandas as pd
24 import matplotlib.pyplot as plt
25
26 #####
27
28 class ResearchDataAnalyzer:
29     """Advanced research data analysis tool"""
30
31     def __init__(self):
32         self.parser = self.setup_parser()
33
34     def setup_parser(self):
35         """Setup advanced argument parser"""
36         parser = argparse.ArgumentParser(
37             description="Advanced Research Data Analyzer",
38             formatter_class=argparse.RawDescriptionHelpFormatter,
39             epilog=f"""
40 Examples:
41 python {__file__} -i engineering_test_data.csv --stats --plot
42 python {__file__} -i engineering_test_data.csv --trends --correlations
43 python {__file__} -i engineering_test_data.csv --filter "Temperature_C > 25" --verbose
44 """
45         )
46
47         # Input/output
48         parser.add_argument(
49             "--input", "-i", required=True, help="Input CSV file"
50         )
51         parser.add_argument(
52             "--output",
53             "-o",
54             default="advanced_analysis",
55             help="Output directory",
56         )
57
58         # Analysis options
59         parser.add_argument(
60             "--stats", action="store_true", help="Calculate statistics"
61         )
62         parser.add_argument(
63             "--trends", action="store_true", help="Analyze trends"
64         )
65         parser.add_argument(
66             "--correlations",
67             action="store_true",
68             help="Calculate correlations",
69         )
70
71         # Output options
72         parser.add_argument(
73             "--plot", action="store_true", help="Generate plots"
74         )
75         parser.add_argument(
76             "--export-csv", action="store_true", help="Export processed data"
77         )
78
79         # Filtering
80         parser.add_argument(
81             "--filter",
82             type=str,
83             help='Filter condition (e.g., "Temperature > 25")',
84         )
85

```

```

86
87     # General
88     parser.add_argument(
89         "--verbose", "-v", action="store_true", help="Verbose output"
90     )
91
92     return parser
93
94     def load_data(self, input_file):
95         """Load data from CSV file"""
96         try:
97             df = pd.read_csv(input_file)
98             if self.args.verbose:
99                 print(f"[OK] Loaded {len(df)} rows from {input_file}")
100             return df
101         except Exception as e:
102             print(f"[ERROR] Loading {input_file}: {e}")
103             return None
104
105     def calculate_statistics(self, df):
106         """Calculate comprehensive statistics"""
107         stats = {}
108         numeric_cols = df.select_dtypes(include=[np.number]).columns
109
110         for col in numeric_cols:
111             stats[col] = {
112                 "mean": float(df[col].mean()),
113                 "median": float(df[col].median()),
114                 "std": float(df[col].std()),
115                 "min": float(df[col].min()),
116                 "max": float(df[col].max()),
117                 "q1": float(df[col].quantile(0.25)),
118                 "q3": float(df[col].quantile(0.75)),
119             }
120
121         return stats
122
123     def analyze_trends(self, df):
124         """Analyze trends in time series data"""
125         trends = {}
126         numeric_cols = df.select_dtypes(include=[np.number]).columns
127
128         # Try to find time column
129         time_cols = [col for col in df.columns if "time" in col.lower()]
130         time_col = time_cols[0] if time_cols else None
131
132         for col in numeric_cols:
133             if col != time_col:
134                 if time_col:
135                     # Linear regression using time
136                     slope, intercept = np.polyfit(df[time_col], df[col], 1)
137                 else:
138                     # Use index as proxy for time
139                     slope, intercept = np.polyfit(range(len(df)), df[col], 1)
140
141                 trends[col] = {
142                     "slope": float(slope),
143                     "intercept": float(intercept),
144                     "trend": "increasing" if slope > 0 else "decreasing",
145                 }
146
147         return trends
148
149     def generate_plots(self, df, output_dir):
150         """Generate visualization plots"""
151         numeric_cols = df.select_dtypes(include=[np.number]).columns
152
153         # Time series plots
154         time_cols = [col for col in df.columns if "time" in col.lower()]
155         if time_cols:
156             time_col = time_cols[0]
157             for col in numeric_cols:
158                 if col != time_col:
159                     plt.figure(figsize=(10, 6))
160                     plt.plot(df[time_col], df[col], "bo-", alpha=0.7)

```

```
161         plt.xlabel(time_col)
162         plt.ylabel(col)
163         plt.title(f"{col} vs {time_col}")
164         plt.grid(True, alpha=0.3)
165         plt.savefig(
166             f"{output_dir}/{col}_plot.png",
167             dpi=300,
168             bbox_inches="tight",
169         )
170         plt.close()
171
172     if self.args.verbose:
173         print("[OK] Plots generated")
174
175     def run_analysis(self, args):
176         """Run complete analysis"""
177         self.args = args
178
179         # Create output directory
180         os.makedirs(args.output, exist_ok=True)
181
182         # Load data
183         df = self.load_data(args.input)
184         if df is None:
185             return False
186
187         results = {
188             "analysis_date": datetime.now().isoformat(),
189             "input_file": args.input,
190             "data_shape": df.shape,
191         }
192
193         # Apply filter if specified
194         if args.filter:
195             try:
196                 # Simple filter implementation
197                 if ">" in args.filter:
198                     col, value = args.filter.split(">")
199                     col, value = col.strip(), float(value.strip())
200                     initial_count = len(df)
201                     df = df[df[col] > value]
202                     results["filter_applied"] = args.filter
203                     results["records_filtered"] = (
204                         f"{initial_count} -> {len(df)}"
205                     )
206             except Exception as e:
207                 print(f"Warning: Filter not applied - {e}")
208
209         # Perform analyses
210         if args.stats:
211             results["statistics"] = self.calculate_statistics(df)
212
213         if args.trends:
214             results["trends"] = self.analyze_trends(df)
215
216         if (
217             args.correlations
218             and len(df.select_dtypes(include=[np.number]).columns) > 1
219         ):
220             corr_matrix = df.select_dtypes(include=[np.number]).corr()
221             results["correlations"] = corr_matrix.to_dict()
222
223         # Generate outputs
224         if args.plot:
225             self.generate_plots(df, args.output)
226
227         if args.export_csv:
228             df.to_csv(f"{args.output}/processed_data.csv", index=False)
229
230         # Save results
231         with open(f"{args.output}/analysis_results.json", "w") as f:
232             json.dump(results, f, indent=2)
233
234         # Print summary
235         if args.verbose:
```

```
236         self.print_summary(results)
237
238     return True
239
240 def print_summary(self, results):
241     """Print analysis summary"""
242     print("\n" + "=" * 50)
243     print("ADVANCED ANALYSIS SUMMARY")
244     print("=" * 50)
245
246     print(f"Input: {results['input_file']}")
247     print(f"Records: {results['data_shape'][0]}")
248
249     if "statistics" in results:
250         print("\nStatistics:")
251         for col, stats in results["statistics"].items():
252             print(
253                 f"    {col}: mean={stats['mean']:.2f}, std={stats['std']:.2f}"
254             )
255
256     if "trends" in results:
257         print("\nTrends:")
258         for col, trend in results["trends"].items():
259             print(
260                 f"    {col}: slope={trend['slope']:.4f} ({trend['trend']})"
261             )
262
263
264 def main():
265     """Main function"""
266     analyzer = ResearchDataAnalyzer()
267     args = analyzer.parser.parse_args()
268
269     if not os.path.exists(args.input):
270         print(f"Error: Input file '{args.input}' not found")
271         sys.exit(1)
272
273     success = analyzer.run_analysis(args)
274
275     if success:
276         print(f"\n[OK] Advanced analysis complete! Results in {args.output}/")
277         sys.exit(0)
278     else:
279         sys.exit(1)
280
281
282 if __name__ == "__main__":
283     main()
```

Listing 9: Advanced analyzer with trends and plots