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:

```python
import pandas as pd
df = pd.read_csv("engineering_test_data.csv")
print(df.describe())            # central tendency + spread
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:

```bash
# with conda
conda create -n pybootcamp python=3.10 -y
conda activate pybootcamp

# or with mamba
mamba create -n pybootcamp python=3.10 -y
mamba activate pybootcamp

pip install -r requirements.txt
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.

```python
import numpy as np
RNG_SEED = 42
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 02_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 03_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:

```python
def main():
    parser = build_parser()
    args = parser.parse_args()

    if not os.path.exists(args.input_dir):
        print(f"Error: Input directory '{args.input_dir}' not found")
        sys.exit(1)

    os.makedirs(args.output_dir, exist_ok=True)
    experiment_files = find_experiment_files(
        args.input_dir, args.file_pattern, verbose=args.verbose
    )

    all_results = []
    for path in experiment_files:
        result = process_single_experiment(
            path, args.output_dir, verbose=args.verbose
        )
        all_results.append(result)

    if args.summary:
        generate_batch_summary(all_results, args.output_dir)

if __name__ == "__main__":
    main()
```

Listing 4: Command-line entry point pattern

0.4 04_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 05_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

session3/01_create_test_data.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Test Data Creation (used by many examples)

Creates one main CSV file plus a small folder of experiment
files.

Note:
- Do not worry about the mathematics.
- Treat this script purely as a  data generator for the rest of the session.

@author: Hrishikesh Terdalkar
"""

###############################################################################

import os

import pandas as pd
import numpy as np

RNG_SEED = 42
np.random.seed(RNG_SEED)

###############################################################################


def create_engineering_test_data():
    """Create realistic engineering test data"""
    # Time points (0 to 120 minutes, every 5 minutes)
    time_points = np.arange(0, 121, 5)

    # Simulate different engineering parameters with realistic patterns
    temperature = (
        25
        + 8 * np.sin(2 * np.pi * time_points / 60)
        + np.random.normal(0, 1, len(time_points))
    )
    pressure = (
        101.3
        + 2 * np.cos(2 * np.pi * time_points / 40)
        + np.random.normal(0, 0.2, len(time_points))
    )
    flow_rate = (
        5
        + 2 * np.sin(2 * np.pi * time_points / 30)
        + np.random.normal(0, 0.1, len(time_points))
    )
    vibration = (
        0.1
        + 0.05 * np.sin(2 * np.pi * time_points / 20)
        + np.random.normal(0, 0.01, len(time_points))
    )

    df = pd.DataFrame(
        {
            "Time_min": time_points,
            "Temperature_C": temperature,
            "Pressure_kPa": pressure,
            "Flow_Rate_Lmin": flow_rate,
            "Vibration_mm": vibration,
        }
    )

    return df

```

```python
def create_multiple_experiments():
    """Create multiple experiment files for batch processing"""
    experiments_dir = "experiments"
    os.makedirs(experiments_dir, exist_ok=True)

    experiment_types = [
        ("thermal_study", "Temperature", "C", 20, 35),
        ("pressure_test", "Pressure", "kPa", 100, 110),
        ("flow_analysis", "FlowRate", "L/min", 1, 10),
        ("vibration_test", "Vibration", "mm", 0.05, 0.15),
    ]

    for exp_type, param, unit, min_val, max_val in experiment_types:
        for i in range(1, 4):  # Create 3 of each type
            exp_name = f"{exp_type}_{i:02d}"
            time_points = np.arange(0, 61, 2)  # 1 hour of data

            # Different patterns for different experiments
            if exp_type == "thermal_study":
                pattern = np.sin(2 * np.pi * time_points / 30)
            elif exp_type == "pressure_test":
                pattern = np.cos(2 * np.pi * time_points / 20)
            elif exp_type == "flow_analysis":
                pattern = np.sin(2 * np.pi * time_points / 15)
            else:  # vibration_test
                pattern = np.sin(2 * np.pi * time_points / 10)

            measurements = min_val + (max_val - min_val) * pattern
            measurements += np.random.normal(
                0, (max_val - min_val) * 0.05, len(time_points)
            )

            df = pd.DataFrame(
                {
                    "Time_min": time_points,
                    f"{param}_{unit}": measurements,
                    "Experiment_ID": exp_name,
                }
            )

            filename = os.path.join(experiments_dir, f"{exp_name}.csv")
            df.to_csv(filename, index=False)
            print(f"Created: {filename}")


def main():
    """Main function to create test data"""
    print("Creating test data for Session 3 examples...")

    # Create single comprehensive test file
    comprehensive_data = create_engineering_test_data()
    comprehensive_data.to_csv("engineering_test_data.csv", index=False)
    print("Created: engineering_test_data.csv")

    # Create multiple experiment files for batch processing
    create_multiple_experiments()
    print("Created multiple experiment files in 'experiments/' directory")

    print("\nTest data creation complete!")
    print("\nNext steps (common scripts):")
    print(
        "  python session3/03_batch_processor.py --input-dir experiments --output-dir
    batch_results --summary"
    )
    print(
        "  python session3/04_basic_analyzer.py engineering_test_data.csv --stats --verbose"
    )
    print(
        "  python session3/05_advanced_analyzer.py -i engineering_test_data.csv --stats --
    trends --plot"
    )


if __name__ == "__main__":
    main()
```

Listing 5: Reproducible test data generator

session3/02_file_operations.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
File Operations Module
Basic file handling and data organization functions

@author: Hrishikesh Terdalkar
"""

###############################################################################

import os
import csv
import json
from datetime import datetime

import pandas as pd

###############################################################################


def create_sample_experiment_data():
    """Create sample experimental data for demonstration"""
    # Temperature and pressure readings over time
    time_points = [0, 5, 10, 15, 20, 25, 30]  # minutes
    temperatures = [25.0, 26.5, 28.2, 27.8, 26.9, 25.5, 24.8]  # deg C
    pressures = [101.3, 101.5, 101.8, 101.6, 101.4, 101.2, 101.1]  # kPa

    return time_points, temperatures, pressures


def save_data_to_csv(filename, times, temps, pressures):
    """Save experimental data to CSV file manually"""
    with open(filename, "w", newline="") as csvfile:
        writer = csv.writer(csvfile)
        # Write header
        writer.writerow(["Time_min", "Temperature_C", "Pressure_kPa"])
        # Write data rows
        for i in range(len(times)):
            writer.writerow([times[i], temps[i], pressures[i]])
    print(f"Data saved to {filename}")


def read_data_from_csv(filename):
    """Read experimental data from CSV file manually"""
    times, temps, pressures = [], [], []

    with open(filename, "r") as csvfile:
        reader = csv.reader(csvfile)
        next(reader)  # Skip header
        for row in reader:
            times.append(float(row[0]))
            temps.append(float(row[1]))
            pressures.append(float(row[2]))

    return times, temps, pressures


def use_pandas_for_data_handling(times, temps, pressures):
    """Demonstrate pandas for easier data handling"""
    # Create DataFrame
    df = pd.DataFrame(
        {"Time_min": times, "Temperature_C": temps, "Pressure_kPa": pressures}
    )

    print("DataFrame created:")
    print(df)

    # Basic statistics
    print("\nBasic statistics:")
    print(df.describe())

    return df


def handle_json_configuration():
    """Work with JSON files for experiment configuration"""
    experiment_info = {
        "experiment_id": "thermal_study_001",
        "researcher": "PhD Student",
        "date": datetime.now().isoformat(),
        "equipment": {
            "sensor": "Thermocouple Type K",
            "data_logger": "Arduino Uno",
            "sampling_rate": "1 sample/minute",
        },
        "conditions": {"ambient_temperature": 25.0, "humidity": 45.0},
    }

    # Save to JSON file
    with open("experiment_config.json", "w") as jsonfile:
        json.dump(experiment_info, jsonfile, indent=2)
    print("Experiment configuration saved to JSON")

    # Read from JSON file
    with open("experiment_config.json", "r") as jsonfile:
        loaded_config = json.load(jsonfile)

    print("\nExperiment configuration loaded:")
    for key, value in loaded_config.items():
        print(f"{key}: {value}")


def demonstrate_file_operations():
    """Demonstrate all file operations"""
    print("=== FILE OPERATIONS DEMONSTRATION ===")

    # Create sample data
    times, temps, pressures = create_sample_experiment_data()

    # Save and read CSV manually
    save_data_to_csv("manual_data.csv", times, temps, pressures)
    times_read, temps_read, pressures_read = read_data_from_csv(
        "manual_data.csv"
    )
    print(f"Data read back: {len(times_read)} measurements")

    # Use pandas
    df = use_pandas_for_data_handling(times, temps, pressures)
    df.to_csv("pandas_data.csv", index=False)
    df_read = pd.read_csv("pandas_data.csv")
    print(f"Pandas data read back: {df_read.shape}")

    # JSON configuration
    handle_json_configuration()


if __name__ == "__main__":
    demonstrate_file_operations()
```

Listing 6: File operations and JSON metadata

session3/03_batch_processor.py

```python
#!/usr/bin/env python3
```

```python
# -*- coding: utf-8 -*-
"""
Batch Experiment Processor

Conceptually simple:
- look for CSV files in a folder
- compute a few statistics for each file
- save one JSON result per file
- (optionally) save a compact batch summary

Everything is written as plain functions so you
can read it top to bottom without worrying about
classes or advanced patterns.

@author: Hrishikesh Terdalkar
"""

###############################################################################

import os
import sys
import json
import glob
import argparse
from datetime import datetime

import numpy as np
import pandas as pd

RNG_SEED = 42
np.random.seed(RNG_SEED)

###############################################################################


def build_parser() -> argparse.ArgumentParser:
    """Setup batch processing parser"""
    parser = argparse.ArgumentParser(
        description="Batch Process Multiple Experiments (function-based version)",
        epilog=(
            "Example: python session3/03_batch_processor.py "
            "--input-dir experiments/ --output-dir results/ --summary"
        ),
    )

    parser.add_argument(
        "--input-dir",
        "-i",
        required=True,
        help="Directory containing experiment CSV files",
    )
    parser.add_argument(
        "--output-dir",
        "-o",
        required=True,
        help="Output directory for per-file results",
    )
    parser.add_argument(
        "--file-pattern",
        default="*.csv",
        help="File pattern to match (default: *.csv)",
    )
    parser.add_argument(
        "--summary", action="store_true", help="Generate summary report"
    )
    parser.add_argument(
        "--verbose", "-v", action="store_true", help="Print progress details"
    )

    return parser


def find_experiment_files(input_dir: str, pattern: str, verbose: bool = False):
    """Find all experiment files matching pattern"""
    search_pattern = os.path.join(input_dir, pattern)
    files = sorted(glob.glob(search_pattern))

    if not files:
        print(f"No files found matching {search_pattern}")
        return []

    if verbose:
        print(f"Found {len(files)} experiment files:")
        for file in files:
            print(f"  {os.path.basename(file)}")

    return files


def process_single_experiment(file_path: str, output_dir: str, verbose: bool):
    """Process a single experiment file"""
    try:
        exp_name = os.path.splitext(os.path.basename(file_path))[0]
        exp_output_dir = os.path.join(output_dir, exp_name)
        os.makedirs(exp_output_dir, exist_ok=True)

        # Load data
        df = pd.read_csv(file_path)

        # Basic analysis
        results = {
            "experiment": exp_name,
            "file": file_path,
            "timestamp": datetime.now().isoformat(),
            "data_shape": df.shape,
            "columns": list(df.columns),
        }

        # Calculate metrics for numeric columns only
        numeric_cols = df.select_dtypes(include=[np.number]).columns
        metrics = {}

        for col in numeric_cols:
            metrics[col] = {
                "mean": float(df[col].mean()),
                "std": float(df[col].std()),
                "min": float(df[col].min()),
                "max": float(df[col].max()),
            }

        results["metrics"] = metrics

        # Save results
        output_file = os.path.join(exp_output_dir, "analysis.json")
        with open(output_file, "w", encoding="utf-8") as f:
            json.dump(results, f, indent=2)

        if verbose:
            print(f"[OK] Processed {exp_name}")

        return results

    except Exception as exc:  # pragma: no cover - simple demo
        print(f"[ERROR] Processing {file_path}: {exc}")
        return {"error": str(exc), "file": file_path}


def generate_batch_summary(all_results, output_dir: str):
    """Generate and print a simple summary report"""
    successful = [r for r in all_results if "error" not in r]
    failed = [r for r in all_results if "error" in r]

    summary = {
        "batch_date": datetime.now().isoformat(),
        "total_files": len(all_results),
        "successful": len(successful),
        "failed": len(failed),
        "experiments": [],
    }

    for result in successful:
        summary["experiments"].append(
            {
                "name": result["experiment"],
                "data_points": result["data_shape"][0],
                "variables": result["data_shape"][1],
            }
        )

    # Save summary
    summary_file = os.path.join(output_dir, "batch_summary.json")
    with open(summary_file, "w", encoding="utf-8") as f:
        json.dump(summary, f, indent=2)

    # Print summary
    print("\n" + "=" * 50)
    print("BATCH PROCESSING SUMMARY")
    print("=" * 50)
    print(f"Total files: {summary['total_files']}")
    print(f"Successful: {summary['successful']}")
    print(f"Failed: {summary['failed']}")

    if successful:
        print("\nSuccessful experiments:")
        for exp in summary["experiments"][:5]:  # Show first 5
            print(f"  {exp['name']}: {exp['data_points']} points")

    if failed:
        print("\nFailed experiments:")
        for failure in failed[:3]:  # Show first 3 failures
            print(f"  {failure['file']}: {failure['error']}")


def create_test_experiments():
    """Create test experiment files for demonstration"""
    test_dir = "test_experiments"
    os.makedirs(test_dir, exist_ok=True)

    experiments = [
        ("thermal_001", "Temperature", "C", 20, 35),
        ("pressure_002", "Pressure", "kPa", 100, 110),
        ("flow_003", "FlowRate", "L/min", 1, 10),
    ]

    for exp_name, param, unit, min_val, max_val in experiments:
        time_points = np.arange(0, 60, 2)
        measurements = min_val + (max_val - min_val) * np.sin(
            2 * np.pi * time_points / 30
        )
        measurements += np.random.normal(
            0, (max_val - min_val) * 0.05, len(time_points)
        )

        df = pd.DataFrame(
            {"Time_min": time_points, f"{param}_{unit}": measurements}
        )

        filename = os.path.join(test_dir, f"{exp_name}.csv")
        df.to_csv(filename, index=False)
        print(f"Created: {filename}")

    return test_dir


def main():
    """Main function for the batch processor"""
    # Create test data if running directly
    if not os.path.exists("test_experiments"):
        print("Creating test experiment files...")
        create_test_experiments()

    parser = build_parser()
    args = parser.parse_args()
    if not os.path.exists(args.input_dir):
        print(f"Error: Input directory '{args.input_dir}' not found")
        sys.exit(1)

    # Create output directory
    os.makedirs(args.output_dir, exist_ok=True)

    # Find experiment files
    experiment_files = find_experiment_files(
        args.input_dir, args.file_pattern, verbose=args.verbose
    )

    if not experiment_files:
        sys.exit(1)

    # Process each experiment
    all_results = []

    for file_path in experiment_files:
        result = process_single_experiment(
            file_path, args.output_dir, verbose=args.verbose
        )
        all_results.append(result)

    # Generate summary if requested
    if args.summary:
        generate_batch_summary(all_results, args.output_dir)

    print(f"\n[OK] Batch processing complete! Results in {args.output_dir}")

    success = True

    if success:
        sys.exit(0)
    else:
        sys.exit(1)


if __name__ == "__main__":
    main()
```

Listing 7: Batch processing CLI

session3/04_basic_analyzer.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Basic Research Data Analyzer
Simple command-line tool for data analysis

@author: Hrishikesh Terdalkar
"""

###############################################################################

import os
import sys
import json
import argparse
from datetime import datetime

import pandas as pd
import numpy as np

###############################################################################


def calculate_basic_stats(data):
    """Calculate basic statistics for a dataset"""
    stats = {
        "mean": float(np.mean(data)),
        "median": float(np.median(data)),
        "std_dev": float(np.std(data)),
        "min": float(min(data)),
        "max": float(max(data)),
        "range": float(max(data) - min(data)),
    }
    return stats


def setup_basic_parser():
    """Setup basic argument parser"""
    parser = argparse.ArgumentParser(
        description="Basic Research Data Analyzer",
        epilog=(
            "Example: python session3/04_basic_analyzer.py "
            "engineering_test_data.csv --stats --output results"
        ),
    )

    parser.add_argument(
        "input", help="Input CSV file containing experimental data"
    )
    parser.add_argument(
        "--output",
        "-o",
        default="analysis_results",
        help="Output directory for results",
    )
    parser.add_argument(
        "--stats", action="store_true", help="Calculate basic statistics"
    )
    parser.add_argument(
        "--verbose", "-v", action="store_true", help="Enable verbose output"
    )

    return parser


def analyze_data(args):
    """Main analysis function"""
    # Create output directory
    os.makedirs(args.output, exist_ok=True)

    try:
        # Load data
        df = pd.read_csv(args.input)

        if args.verbose:
            print(f"Loaded data from {args.input}")
            print(f"Data shape: {df.shape}")
            print(f"Columns: {list(df.columns)}")

        results = {
            "analysis_date": datetime.now().isoformat(),
            "input_file": args.input,
            "data_shape": df.shape,
            "columns": list(df.columns),
        }

        # Calculate statistics if requested
        if args.stats:
            stats_results = {}
            numeric_cols = df.select_dtypes(include=[np.number]).columns

            for col in numeric_cols:
                stats_results[col] = calculate_basic_stats(df[col])

            results["statistics"] = stats_results

            if args.verbose:
                print("Calculated statistics for numeric columns")

        # Save results
        output_file = os.path.join(args.output, "analysis.json")
        with open(output_file, "w") as f:
            json.dump(results, f, indent=2)

        if args.verbose:
            print(f"Results saved to {output_file}")

        # Print summary
        print_summary(results)

        return True

    except Exception as e:
        print(f"Error: {e}")
        return False


def print_summary(results):
    """Print analysis summary"""
    print("\n" + "=" * 50)
    print("ANALYSIS SUMMARY")
    print("=" * 50)
    print(f"Input file: {results['input_file']}")
    print(f"Data shape: {results['data_shape']}")

    if "statistics" in results:
        print("\nStatistical Summary:")
        for col, stats in results["statistics"].items():
            print(f"  {col}:")
            print(f"    Mean: {stats['mean']:.2f}")
            print(f"    Std: {stats['std_dev']:.2f}")
            print(f"    Range: {stats['min']:.2f} to {stats['max']:.2f}")


def main():
    """Main function"""
    parser = setup_basic_parser()
    args = parser.parse_args()

    # Validate input file exists
    if not os.path.exists(args.input):
        print(f"Error: Input file '{args.input}' not found")
        sys.exit(1)

    # Run analysis
    success = analyze_data(args)

    if success:
        print(f"\n[OK] Analysis complete! Results in {args.output}/")
        sys.exit(0)
    else:
        sys.exit(1)


if __name__ == "__main__":
    main()
```

Listing 8: Basic research analyzer

session3/05_advanced_analyzer.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Advanced Research Data Analyzer (optional / stretch)

This script shows more realistic analysis options
for students who are already comfortable with
basic Python and pandas. If you are new to Python,
you can safely treat this as an optional example.

```

```python
@author: Hrishikesh Terdalkar
"""

################################################################################

import os
import sys
import json
import argparse
from datetime import datetime

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

################################################################################


class ResearchDataAnalyzer:
    """Advanced research data analysis tool"""

    def __init__(self):
        self.parser = self.setup_parser()

    def setup_parser(self):
        """Setup advanced argument parser"""
        parser = argparse.ArgumentParser(
            description="Advanced Research Data Analyzer",
            formatter_class=argparse.RawDescriptionHelpFormatter,
            epilog=f"""
Examples:
  python {__file__} -i engineering_test_data.csv --stats --plot
  python {__file__} -i engineering_test_data.csv --trends --correlations
  python {__file__} -i engineering_test_data.csv --filter "Temperature_C > 25" --verbose
            """,
        )

        # Input/output
        parser.add_argument(
            "--input", "-i", required=True, help="Input CSV file"
        )
        parser.add_argument(
            "--output",
            "-o",
            default="advanced_analysis",
            help="Output directory",
        )

        # Analysis options
        parser.add_argument(
            "--stats", action="store_true", help="Calculate statistics"
        )
        parser.add_argument(
            "--trends", action="store_true", help="Analyze trends"
        )
        parser.add_argument(
            "--correlations",
            action="store_true",
            help="Calculate correlations",
        )

        # Output options
        parser.add_argument(
            "--plot", action="store_true", help="Generate plots"
        )
        parser.add_argument(
            "--export-csv", action="store_true", help="Export processed data"
        )

        # Filtering
        parser.add_argument(
            "--filter",
            type=str,
            help='Filter condition (e.g., "Temperature > 25")',
        )

        # General
        parser.add_argument(
            "--verbose", "-v", action="store_true", help="Verbose output"
        )

        return parser

    def load_data(self, input_file):
        """Load data from CSV file"""
        try:
            df = pd.read_csv(input_file)
            if self.args.verbose:
                print(f"[OK] Loaded {len(df)} rows from {input_file}")
            return df
        except Exception as e:
            print(f"[ERROR] Loading {input_file}: {e}")
            return None

    def calculate_statistics(self, df):
        """Calculate comprehensive statistics"""
        stats = {}
        numeric_cols = df.select_dtypes(include=[np.number]).columns

        for col in numeric_cols:
            stats[col] = {
                "mean": float(df[col].mean()),
                "median": float(df[col].median()),
                "std": float(df[col].std()),
                "min": float(df[col].min()),
                "max": float(df[col].max()),
                "q1": float(df[col].quantile(0.25)),
                "q3": float(df[col].quantile(0.75)),
            }

        return stats

    def analyze_trends(self, df):
        """Analyze trends in time series data"""
        trends = {}
        numeric_cols = df.select_dtypes(include=[np.number]).columns

        # Try to find time column
        time_cols = [col for col in df.columns if "time" in col.lower()]
        time_col = time_cols[0] if time_cols else None

        for col in numeric_cols:
            if col != time_col:
                if time_col:
                    # Linear regression using time
                    slope, intercept = np.polyfit(df[time_col], df[col], 1)
                else:
                    # Use index as proxy for time
                    slope, intercept = np.polyfit(range(len(df)), df[col], 1)

                trends[col] = {
                    "slope": float(slope),
                    "intercept": float(intercept),
                    "trend": "increasing" if slope > 0 else "decreasing",
                }

        return trends

    def generate_plots(self, df, output_dir):
        """Generate visualization plots"""
        numeric_cols = df.select_dtypes(include=[np.number]).columns

        # Time series plots
        time_cols = [col for col in df.columns if "time" in col.lower()]
        if time_cols:
            time_col = time_cols[0]
            for col in numeric_cols:
                if col != time_col:
                    plt.figure(figsize=(10, 6))
                    plt.plot(df[time_col], df[col], "bo-", alpha=0.7)
                    plt.xlabel(time_col)
                    plt.ylabel(col)
                    plt.title(f"{col} vs {time_col}")
                    plt.grid(True, alpha=0.3)
                    plt.savefig(
                        f"{output_dir}/{col}_plot.png",
                        dpi=300,
                        bbox_inches="tight",
                    )
                    plt.close()

        if self.args.verbose:
            print("[OK] Plots generated")

    def run_analysis(self, args):
        """Run complete analysis"""
        self.args = args

        # Create output directory
        os.makedirs(args.output, exist_ok=True)

        # Load data
        df = self.load_data(args.input)
        if df is None:
            return False

        results = {
            "analysis_date": datetime.now().isoformat(),
            "input_file": args.input,
            "data_shape": df.shape,
        }

        # Apply filter if specified
        if args.filter:
            try:
                # Simple filter implementation
                if ">" in args.filter:
                    col, value = args.filter.split(">")
                    col, value = col.strip(), float(value.strip())
                    initial_count = len(df)
                    df = df[df[col] > value]
                    results["filter_applied"] = args.filter
                    results["records_filtered"] = (
                        f"{initial_count} -> {len(df)}"
                    )
            except Exception as e:
                print(f"Warning: Filter not applied - {e}")

        # Perform analyses
        if args.stats:
            results["statistics"] = self.calculate_statistics(df)

        if args.trends:
            results["trends"] = self.analyze_trends(df)

        if (
            args.correlations
            and len(df.select_dtypes(include=[np.number]).columns) > 1
        ):
            corr_matrix = df.select_dtypes(include=[np.number]).corr()
            results["correlations"] = corr_matrix.to_dict()

        # Generate outputs
        if args.plot:
            self.generate_plots(df, args.output)

        if args.export_csv:
            df.to_csv(f"{args.output}/processed_data.csv", index=False)

        # Save results
        with open(f"{args.output}/analysis_results.json", "w") as f:
            json.dump(results, f, indent=2)

        # Print summary
        if args.verbose:
            self.print_summary(results)

        return True

    def print_summary(self, results):
        """Print analysis summary"""
        print("\n" + "=" * 50)
        print("ADVANCED ANALYSIS SUMMARY")
        print("=" * 50)

        print(f"Input: {results['input_file']}")
        print(f"Records: {results['data_shape'][0]}")

        if "statistics" in results:
            print("\nStatistics:")
            for col, stats in results["statistics"].items():
                print(
                    f"  {col}: mean={stats['mean']:.2f}, std={stats['std']:.2f}"
                )

        if "trends" in results:
            print("\nTrends:")
            for col, trend in results["trends"].items():
                print(
                    f"  {col}: slope={trend['slope']:.4f} ({trend['trend']})"
                )


def main():
    """Main function"""
    analyzer = ResearchDataAnalyzer()
    args = analyzer.parser.parse_args()

    if not os.path.exists(args.input):
        print(f"Error: Input file '{args.input}' not found")
        sys.exit(1)

    success = analyzer.run_analysis(args)

    if success:
        print(f"\n[OK] Advanced analysis complete! Results in {args.output}/")
        sys.exit(0)
    else:
        sys.exit(1)


if __name__ == "__main__":
    main()
```

Listing 9: Advanced analyzer with trends and plots