

Python Programming & Data Analysis

Unit 15 — Capstone Project

Project Handout — CityBike — Bike-Sharing Operations & Usage Analytics Platform

Date: February 8, 2026
Estimated workload: 20–30 hours
Required libraries: pandas, numpy, matplotlib

Motivation & Context

Bike-sharing systems have become a cornerstone of modern urban transport in cities worldwide — from Berlin’s *Nextbike* to New York’s *Citi Bike* and London’s *Santander Cycles*. These systems generate vast amounts of operational data every day: trip records, station usage, fleet maintenance logs, and user activity.

Managing such a system is a real-world software engineering challenge. Operators need to:

- Track hundreds of bikes across dozens of stations
- Understand when and where demand peaks occur
- Schedule preventive maintenance to minimize downtime
- Analyze user behavior to optimize station placement and pricing
- Generate reports for city authorities and stakeholders

In this capstone project, you will build a **complete backend platform** for a fictional city bike-sharing service. You will model the domain with object-oriented design, load and clean operational datasets, implement algorithms, compute statistics, generate visualizations, and deliver a well-structured, version-controlled codebase.

Project Overview

This is the **final integrative project** of the course. It is designed to demonstrate mastery of every major topic covered across Units 1–13.

This project integrates:

- Modular code structure, functions, multi-file organization (Units 1–5)
- Object-oriented design: classes, inheritance, abstract base classes, design patterns (Units 7–8)
- Algorithms: sorting, searching, complexity analysis (Unit 9)
- NumPy for numerical computing (Unit 10)
- Pandas for data analysis, cleaning, and aggregation (Unit 11)
- Data visualization with Matplotlib (Unit 12)
- Version control with Git and GitHub

Unlike the earlier mini projects, this capstone requires you to demonstrate **all** of the above in a single, cohesive application. Quality expectations are higher: your code must be documented, modular, and professionally structured.

Functional Requirements

1) Object-Oriented Data Models (Units 7–8)

Design a class hierarchy to represent the bike-sharing domain.

Required Classes

- **Entity** (abstract base class): shared `id`, `created_at`; enforces `__str__` and `__repr__` in subclasses
- **Bike(Entity)**: `bike_id`, `bike_type`, `status` (`available/in_use/maintenance`)
 - **ClassicBike(Bike)**: `gear_count`
 - **ElectricBike(Bike)**: `battery_level`, `max_range_km`
- **Station(Entity)**: `station_id`, `name`, `capacity`, `latitude`, `longitude`
- **User(Entity)**: `user_id`, `name`, `email`, `user_type`
 - **CasualUser(User)**: `day_pass_count`
 - **MemberUser(User)**: `membership_start`, `membership_end`, `tier` (`basic/premium`)
- **Trip**: `trip_id`, `user`, `bike`, `start_station`, `end_station`, `start_time`, `end_time`, `distance_km`
- **MaintenanceRecord**: `record_id`, `bike`, `date`, `maintenance_type`, `cost`, `description`
- **BikeShareSystem**: orchestrates loading, cleaning, analysis, and reporting

OOP Requirements

- Proper `__init__` constructors with **input validation** (e.g., reject negative prices, invalid emails)
- String representations: `__str__` for user-friendly output, `__repr__` for debugging
- At least one **abstract base class** using Python's `abc` module
- At least one **inheritance hierarchy** with a minimum depth of two (e.g., `Entity` → `Bike` → `ElectricBike`)
- Use of **properties** (`@property`) for controlled attribute access where appropriate
- Implement at least **two design patterns**:
 - **Factory Pattern**: create `Bike` or `User` objects from CSV row dictionaries without exposing subclass constructors
 - **Strategy Pattern**: interchangeable pricing strategies (e.g., `CasualPricing`, `MemberPricing`, `PeakHourPricing`) that compute trip cost

2) Data Loading & Cleaning (Unit 11)

- Load the provided CSV files (`trips.csv`, `stations.csv`, `maintenance.csv`) into Pandas DataFrames
- Inspect structure, types, and missing values using `info()`, `describe()`, `isnull().sum()`
- Handle missing values with a **documented strategy** (drop, fill, interpolate — justify your choice)
- Validate and convert types: parse dates, ensure numeric columns are numeric, standardize categorical values
- Remove duplicates and invalid entries (e.g., trips where end time is before start time)
- Export cleaned, analysis-ready datasets to CSV

3) Algorithmic Analysis (Unit 9)

Implement **both** of the following from scratch:

- **Sorting**:
 - Implement your own sorting algorithm (e.g., merge sort, quicksort)
 - Apply it to sort trips by duration or distance

- Also sort using Python’s built-in `sorted()` / Pandas `sort_values()`
- Compare execution times using `timeit`
- **Searching:**
 - Implement your own search algorithm (e.g., binary search on sorted data)
 - Apply it to find trips, stations, or users by ID or attribute
 - Also search using built-in methods (e.g., `in`, Pandas `.loc[]`, `.query()`)
 - Compare execution times using `timeit`

Optional Analysis:

- Document the **time complexity (Big-O)** for each of your implementations
- Include a brief written comparison discussing the performance differences
- Explain why built-in functions are typically faster

4) Numerical Computing (Unit 10)

Use NumPy for at least the following:

- Compute **distances** between stations using the Euclidean distance formula on latitude/longitude arrays:

$$d = \sqrt{(\text{lat}_2 - \text{lat}_1)^2 + (\text{lon}_2 - \text{lon}_1)^2}$$

(Use a simplified flat-earth model; no need for the Haversine formula.)

- Compute vectorized **statistics** on trip durations and distances (mean, median, standard deviation, percentiles) without using Python loops
- Use NumPy arrays for any batch numerical computation (e.g., fare calculation across all trips, z-score outlier detection)

5) Analytics & Business Insights (Units 10–11)

Using Pandas and NumPy, answer at least **10** of the following questions:

1. Total number of trips, total distance traveled, and average trip duration
2. What are the **top 10 most popular** start stations and end stations?
3. What are the **peak usage hours** during the day?
4. Which **day of the week** has the highest trip volume?
5. What is the **average trip distance** by user type (casual vs. member)?
6. What is the **bike utilization rate** (percentage of time bikes are in use vs. available)?
7. Show the **monthly trip trend** over time — is ridership growing?
8. Who are the **top 15 most active users** by trip count?
9. What is the **total maintenance cost** per bike type (classic vs. electric)?
10. What are the **most common station-to-station routes** (top 10 origin–destination pairs)?
11. What is the **trip completion rate** (completed vs. cancelled trips)?
12. What is the **average number of trips per user**, segmented by user type?
13. Which bikes have the **highest maintenance frequency**?
14. Identify **outlier trips** (unusually long/short duration or distance) using statistical methods

6) Visualization (Unit 12)

Create at least **4 visualizations** using Matplotlib:

- **Bar chart:** trips per station or revenue by user type
- **Line chart:** monthly trip volume trend over time
- **Histogram:** trip duration or distance distribution
- **Box plot:** trip duration comparison between user types or bike types

All charts must have:

- Descriptive title
- Labeled axes with units
- Legend where applicable
- Clean, readable styling

Export all visualizations as **PNG files** to the `output/figures/` directory.

7) Reporting & Export (Units 5, 11–12)

- Export all cleaned datasets to CSV (`data/trips_clean.csv`, etc.)
- Generate a **summary report** (`.txt`) containing key metrics and answers to business questions
- Export top stations, top users, and maintenance summaries as separate CSV or text files

Clean Code & Documentation

- **Docstrings**: every module, class, and public function must have a docstring
- **Naming**: use clear, descriptive names (no single-letter variables except in loops)
- **Separation of concerns**: business logic must be separate from I/O and display
- **DRY principle**: avoid code duplication; extract common logic into helper functions
- **Functions**: prefer functions that **return values** rather than printing inside them
- **Type hints**: add type annotations to all function signatures
- **No magic numbers**: use named constants for fixed values

Optional Extension: Unit Testing with pytest (Optional)

This is **not required** but helps demonstrate best practices in software development. You can write unit tests for your OOP models and algorithms to ensure correctness and robustness.

If you choose to include tests, create a `tests/` directory and use `pytest`:

```
1 citybike/  
2   tests/  
3     __init__.py  
4     test_models.py           # Tests for OOP models  
5     test_algorithms.py      # Tests for sorting & searching
```

Guidelines:

- Write at least **10 test functions** covering model validation and algorithm correctness
- Use descriptive test names (e.g., `test_electric_bike_rejects_negative_battery`)
- Test both valid inputs (happy path) and invalid inputs (edge cases)
- All tests must pass when running:

```
1 $ pytest tests/ -v
```

Add `pytest` to your `requirements.txt` if you include tests.

Data Model

You will work with **three CSV files**:

trips.csv

```
1 trip_id,user_id,user_type,bike_id,bike_type,start_station_id,
2 end_station_id,start_time,end_time,duration_minutes,
3 distance_km,status
```

stations.csv

```
1 station_id,station_name,capacity,latitude,longitude
```

maintenance.csv

```
1 record_id,bike_id,bike_type,date,maintenance_type,cost,
2 description
```

Notes:

- Data may contain missing values, duplicates, or inconsistencies (by design)
- `start_time` and `end_time` may need parsing to `datetime`
- `duration_minutes` and `distance_km` may be stored as strings; convert to float
- `status`: completed or cancelled
- `maintenance_type`: tire_repair, brake_adjustment, battery_replacement, chain_lubrication, general_inspection
- Some trips may have `end_time` before `start_time` — these are invalid

Code Organization (Required)

Use a multi-module project structure:

```
1 citybike/
2   main.py           # Entry point: orchestration
3   models.py         # OOP classes (Entity, Bike, Station, ...)
4   analyzer.py       # BikeShareSystem: analysis methods
5   algorithms.py     # Sorting, searching implementations
6   numerical.py      # NumPy-based computations
7   visualization.py  # Matplotlib chart functions
8   pricing.py        # Strategy Pattern: pricing strategies
9   factories.py      # Factory Pattern: object creation
10  utils.py           # Validation, formatting, helpers
11  data/
12    trips.csv        # Raw trip data
13    stations.csv     # Station metadata
14    maintenance.csv  # Maintenance records
15    trips_clean.csv  # Cleaned trip data (generated)
16    stations_clean.csv # Cleaned station data (generated)
17  output/
18    summary_report.txt # Key metrics and insights
19    top_stations.csv  # Most popular stations
20    top_users.csv     # Most active users
21    figures/          # Visualization PNGs
```

Module responsibilities:

File	Responsibility
<code>main.py</code>	Entry point; orchestrate the full pipeline from loading to reporting
<code>models.py</code>	Domain classes with validation, inheritance, abstract base class
<code>analyzer.py</code>	Data analysis: groupby, filtering, aggregation, business metrics
<code>algorithms.py</code>	Custom sorting and searching implementations with benchmarks
<code>numerical.py</code>	NumPy operations: distance calculations, statistics, outlier detection
<code>visualization.py</code>	All Matplotlib chart creation and export functions
<code>pricing.py</code>	Strategy Pattern: pricing strategy interface and implementations
<code>factories.py</code>	Factory Pattern: create model objects from raw data dictionaries
<code>utils.py</code>	Input validation, date parsing, formatting helpers

Design principles:

- Keep business logic separate from I/O and printing
- Prefer functions that return values instead of printing inside
- Each module should have a single, clear responsibility
- Write docstrings for all public functions and classes
- Use type hints in function signatures

Version Control Requirements (Git & GitHub)

You **must** use Git for version control and host your project on GitHub.

Repository Setup

- Create a new GitHub repository for this project
- Initialize with a `README.md` and `.gitignore` (Python template)
- Clone the repository locally and work from there

Commit Requirements

- Make **at least 15 meaningful commits** throughout development
- Each commit should represent a logical unit of work
- Write clear, descriptive commit messages following conventional style:
 - `feat: add Station class with capacity validation`
 - `fix: handle missing values in trip duration column`
 - `test: add unit tests for sorting algorithms`
 - `docs: update README with setup instructions`

Branching (Required)

- Use feature branches for major components (e.g., `feature/models`, `feature/analysis`, `feature/visualization`)
- Merge completed features into `main` via pull requests or local merge
- The `main` branch must always contain working code

Required Files in Repository

- `README.md` — project description, setup instructions, usage guide, and a brief summary of findings
- `.gitignore` — exclude `__pycache__`, `.venv`, IDE files, `output/figures/*.png`
- `requirements.txt` — list all dependencies (`pandas`, `numpy`, `matplotlib`)

Milestones

Follow these milestones to structure your development process. Each milestone should result in at least one commit.

1. Milestone 1 — Project Setup

Create GitHub repo, set up project structure, add `README.md`, `.gitignore`, `requirements.txt`. Generate or place raw data files.

2. Milestone 2 — Domain Models

Implement all OOP classes (`Entity`, `Bike`, `Station`, `User`, `Trip`, `MaintenanceRecord`) with validation, inheritance, and string representations. Implement Factory and Strategy patterns.

3. Milestone 3 — Data Loading & Cleaning

Load CSV files into DataFrames. Inspect, clean, validate, and export cleaned datasets.

4. Milestone 4 — Algorithms

Implement custom sorting and searching algorithms. Benchmark against built-in functions. Document Big-O analysis.

5. Milestone 5 — Numerical Computing

Implement NumPy-based distance calculations, statistical summaries, and outlier detection.

6. Milestone 6 — Analytics

Answer all 10+ business questions using Pandas and NumPy. Generate summary report.

7. Milestone 7 — Visualization

Create 4+ charts with proper labels and styling. Export as PNG.

8. Milestone 8 — Polish & Delivery

Add docstrings and type hints throughout. Update `README.md` with final instructions and findings summary. Final review and commit.

Quality Checklist

Before submission, verify every item:

Code Quality & Clean Code:

- Code is organized into multiple modules with clear responsibilities
- All public functions and classes have docstrings
- Function signatures include type hints
- No magic numbers; named constants are used
- No code duplication; helpers are extracted
- No crashes on invalid input

OOP Design:

- Abstract base class using `abc` module
- Inheritance hierarchy with at least 2 levels
- Constructors validate input
- `__str__` and `__repr__` implemented on all domain classes
- Properties (`@property`) used where appropriate
- Factory Pattern implemented and used
- Strategy Pattern implemented and used

Data Analysis:

- All three CSV files properly loaded, inspected, and cleaned
- Missing value strategy documented
- NumPy used for numerical computations (not Python loops)
- Pandas used for aggregation, filtering, and grouping
- At least 10 business questions answered with code and output

Algorithms:

- Custom sorting algorithm implemented and correct
- Custom searching algorithm implemented and correct
- Built-in equivalents also used for comparison
- Performance comparison with `timeit` included
- Big-O complexity documented for each algorithm

Visualization:

- At least 4 charts created
- All charts have title, axis labels, and legend (where applicable)
- Charts are readable and professionally styled
- Exported as PNG files

Git & GitHub:

- Repository is public (or shared with instructor)
- At least 15 meaningful commits with clear messages
- Feature branches used for development
- `README.md` with project description, setup, and usage
- `requirements.txt` included
- `.gitignore` properly configured

Appendix: Synthetic Data Generator

Use the following script to generate the three dataset files. You may also modify or extend the generator to create additional scenarios.

```

1 import pandas as pd
2 import numpy as np
3 from datetime import datetime, timedelta
4
5 np.random.seed(42)
6
7 # --- Station data ---
8 station_names = [
9     "Central Station", "University Campus", "City Hall",
10    "Riverside Park", "Market Square", "Tech Hub",
11    "Old Town", "Harbor View", "Sports Arena",
12    "West End", "North Gate", "Museum Quarter",
13    "Business District", "Lakeside", "Airport Terminal"
14 ]
15
16 stations = []
17 for i, name in enumerate(station_names):
18     stations.append({
19         "station_id": f"ST{100 + i}",
20         "station_name": name,
21         "capacity": np.random.choice([10, 15, 20, 25, 30]),
22         "latitude": round(48.75 + np.random.uniform(0, 0.15), 6),
23         "longitude": round(9.15 + np.random.uniform(0, 0.15), 6),
24     })
25
26 stations_df = pd.DataFrame(stations)
27 stations_df.to_csv("stations.csv", index=False)
28
29 # --- Trip data ---
30 n_trips = 1500
31 user_ids = [f"USR{np.random.randint(1000, 1200)}"
32             for _ in range(80)]
33 bike_ids = [f"BK{np.random.randint(200, 350)}"
34            for _ in range(60)]
35 start_date = datetime(2024, 1, 1)
36
37 trips = []
38 for i in range(n_trips):
39     user_type = np.random.choice(
40         ["casual", "member"], p=[0.35, 0.65]
41     )
42     bike_type = np.random.choice(
43         ["classic", "electric"], p=[0.6, 0.4]
44     )
45     start_st = np.random.choice(stations_df["station_id"])
46     end_st = np.random.choice(stations_df["station_id"])
47     start_time = start_date + timedelta(
48         days=np.random.randint(0, 365),
49         hours=np.random.randint(6, 23),
50         minutes=np.random.randint(0, 60),
51     )
52     duration = max(2, np.random.exponential(25))
53     end_time = start_time + timedelta(minutes=duration)
54     distance = round(np.random.uniform(0.5, 15.0), 2)
55     status = np.random.choice(

```

```

56         ["completed", "cancelled", np.nan],
57         p=[0.82, 0.12, 0.06]
58     )
59
60     trips.append({
61         "trip_id": f"TR{10000 + i}",
62         "user_id": np.random.choice(user_ids),
63         "user_type": user_type,
64         "bike_id": np.random.choice(bike_ids),
65         "bike_type": bike_type,
66         "start_station_id": start_st,
67         "end_station_id": end_st,
68         "start_time": start_time.strftime(
69             "%Y-%m-%d %H:%M:%S"
70         ),
71         "end_time": end_time.strftime(
72             "%Y-%m-%d %H:%M:%S"
73         ),
74         "duration_minutes": round(duration, 1),
75         "distance_km": distance,
76         "status": status,
77     })
78
79     trips_df = pd.DataFrame(trips)
80
81     # Inject some messiness
82     idx = np.random.choice(n_trips, 30, replace=False)
83     trips_df.loc[idx[:10], "duration_minutes"] = np.nan
84     trips_df.loc[idx[10:20], "distance_km"] = np.nan
85     trips_df.loc[idx[20:25], "end_time"] = (
86         trips_df.loc[idx[20:25], "start_time"]
87     )
88     dup_rows = trips_df.sample(15)
89     trips_df = pd.concat(
90         [trips_df, dup_rows], ignore_index=True
91     )
92     trips_df.to_csv("trips.csv", index=False)
93
94     # --- Maintenance data ---
95     maint_types = [
96         "tire_repair", "brake_adjustment",
97         "battery_replacement", "chain_lubrication",
98         "general_inspection"
99     ]
100
101     records = []
102     for i in range(200):
103         bike = np.random.choice(bike_ids)
104         btype = np.random.choice(["classic", "electric"])
105         mtype = np.random.choice(maint_types)
106         cost = round(np.random.uniform(10, 150), 2)
107         if mtype == "battery_replacement":
108             cost = round(np.random.uniform(80, 250), 2)
109             btype = "electric"
110
111         records.append({
112             "record_id": f"MR{5000 + i}",
113             "bike_id": bike,
114             "bike_type": btype,

```

```
115         "date": (
116             start_date
117             + timedelta(days=np.random.randint(0, 365))
118         ).strftime("%Y-%m-%d"),
119         "maintenance_type": mtype,
120         "cost": cost,
121         "description": f"{mtype.replace('_', ' ').title()}"
122                        f" for bike {bike}",
123     })
124
125 maint_df = pd.DataFrame(records)
126 maint_df.loc[
127     np.random.choice(200, 8, replace=False), "cost"
128 ] = np.nan
129 maint_df.to_csv("maintenance.csv", index=False)
130
131 print("Generated: stations.csv, trips.csv, "
132       "maintenance.csv")
```

Submission

Submit:

1. **GitHub repository URL** (must be accessible to the instructor)
2. Repository must contain:
 - All source code modules (.py files)
 - README.md with project description, setup instructions, and usage guide
 - requirements.txt
 - Raw data files (trips.csv, stations.csv, maintenance.csv)
 - Cleaned datasets (trips_clean.csv, stations_clean.csv)
 - Summary report with answers to business questions (summary_report.txt)
 - Visualization files (output/figures/*.png)

Execution: Your project should run successfully via:

```
1 $ pip install -r requirements.txt
2 $ python main.py
```

Deadline: announced separately. Late submissions lose 10% per calendar day.

Tip: Start with the data generator and models. Commit after every milestone. A working project with clean code is worth more than a feature-rich project that crashes.