Toronto Metropolitan University

CIND 830
Python Programming for Data Science

Apartment for Rent Analysis

Individual Assignment

Introduction

This assignment is an individual effort to analyze and visualize the `Apartment for Rent Classified` dataset. You will apply core programming concepts—data structures, algorithms, and object-oriented design—to explore and extract insights from rental listing data. Use of libraries such as `pandas`, `NumPy`, and `Matplotlib` is optional but recommended for efficient data handling and visualization.

Dataset

The dataset for this assignment is sourced from the UCI Machine Learning Repository:
Apartment for Rent Classified [Dataset]. (2019). UCI Machine Learning Repository.
https://doi.org/10.24432/C5X623.

The dataset contains around 100,000 instances and 21 features describing rental apartments in the USA, including numerical fields (e.g., `price`, `square_feet`, `bathrooms`), categorical fields (e.g., `bedrooms`, `cityname`, `state`, `amenities`), and location coordinates (`latitude`, `longitude`).

Assignment Tasks

Complete all of the tasks below in a single or a set of Jupyter Notebooks. Each task corresponds to a programming concept:

1. Data Structures

- Load the dataset using `ucimlrepo.fetch_ucirepo()` or by reading the CSV/7z file directly into Python (e.g., using `pandas.read_csv()`).
- Inspect the dataset to understand its columns, data types, and check for any missing values.
- Clean the data by handling inconsistencies or missing entries (if any). For example, ensure that `price` and `square_feet` are numeric and convert categorical variables (e.g., `cityname`, `state`) into consistent string formats.
- Compute basic descriptive statistics (mean, median, mode) for numerical features such as `price`, `square_feet`, and `bathrooms` using either built-in Python functions or `NumPy`.

2. Algorithms

- Implement search algorithms to locate specific records—such as apartments with a given `price` (e.g., $1,500) or those in a particular `cityname` (e.g., "Denver") using linear search. Then sort by `price` and apply binary search to find the same target.

- Develop at least two custom sorting algorithms (for example, `bubble sort` and `insertion sort`) to order the dataset by *price*, *square_feet*, or *number of bathrooms*. Do not use built-in sorting for these custom algorithms.

- Compare the runtime performance (using simple timing) of your custom implementations versus the built-in `sorted()` function or `pandas.DataFrame.sort_values()` when sorting by the same keys.

3. Object-Oriented Programming (OOP)

- Define an `Apartment` class to encapsulate all relevant attributes (e.g., `id`, `price`, `square_feet`, `bedrooms`, `bathrooms`, `cityname`, `state`, `latitude`, `longitude`, `amenities`). Include an `__init__()` constructor and at least one method that returns a summary string (e.g., `get_summary()` that prints price, size, and location).

- Create a `DatasetManager` class responsible for loading, preprocessing, and providing access to a list of `Apartment` objects. This class should include:

  - A method to load and clean the raw data (e.g., `load_data()`).
  - A method to convert each row of the cleaned DataFrame into an `Apartment` instance (e.g., `create_apartments()`).
  - Any additional helper methods for data transformations (e.g., encoding categorical variables).

- Demonstrate inheritance by creating at least one subclass (for example, `PriceAnalysis` that extends `DatasetManager` and adds methods to compute price statistics—mean, median, percentiles—or `LocationAnalysis` that adds methods to filter apartments by proximity to a given coordinate).

- Show polymorphism by overriding a method in your subclass to provide specialized behavior (for instance, `get_summary()` in `PriceAnalysis` might return price-specific insights across all apartments).

4. Visualization Tasks

- Use `Matplotlib` (or `seaborn`, if preferred) to create at least two of the following visualizations:

  - **Histogram**: Distribution of `price` to identify outliers or skewness.
  - **Scatter Plot**: Relationship between `square_feet` and `price`.
  - **Bar Chart**: Average `price` grouped by `number of bedrooms` (e.g., 1-bedroom vs. 2-bedroom apartments).
  - **Heatmap**: Correlation matrix among all numerical features (e.g., `price`, `square_feet`, `bathrooms`, `latitude`, `longitude`).

- For each plot, include:

  - **Title**: A concise description (e.g., "Square Feet vs. Price Scatter Plot").

    - **Axis Labels**: Clearly label both axes.
    - **Legends**: If multiple series are plotted.
    - **Gridlines**: Where appropriate, to improve readability.

Submission Guidelines

1. Submit a single Jupyter Notebook (`.ipynb`) file via the course shell.
2. Ensure that your notebook runs from start to finish without errors. All code cells should be executed, and outputs must be visible.
3. Include clear, inline comments describing your logic and approach in each code section.
4. Use Markdown cells to:
    - Explain your methodology for each task.
    - Summarize any challenges or design decisions in a section titled `Reflections on Work`.
5. Name your submission file `FirstName_LastName_BookPlanner.ipynb` and confirm that all code cells are executed and outputs are visible.
6. If you include visualizations, ensure all plots have titles, axis labels, and legends (when applicable).

    For additional guidance, consult the *Submit an assignment* tutorial:
    https://www.torontomu.ca/courses/students/tutorials/assignments/

Grading Rubric

Criteria	**Points**
Data Loading & Cleaning	15
Descriptive Statistics	10
Search & Sorting Algorithms	20
Performance Comparison	5
Apartment Class & OOP Design	10
DatasetManager & Inheritance	15
Visualization	15
Code Quality & Documentation	5
Notebook Functionality & Submission Completeness	5
Total	**100**

End of assignment