Programming *for* Analytics

Project Prompt: API-Driven Application with OOP and Visualizations

Create a Python application that retrieves data from one or more APIs, processes the data using Object-Oriented Programming (OOP) principles, and presents insights through clear visualizations. The project should demonstrate effective use of programming concepts, modular design, and thorough documentation. You may also combine API usage with static text files or SQLite databases.

Requirements:

1. **API Integration**:
    - Identify and integrate one or more APIs (and possibly static datasets) relevant to your chosen topic. Handle API responses, including error cases (e.g., API limits, connectivity issues, or invalid requests).
2. **Object-Oriented Programming**:
    - Design your project using OOP principles:
        - Use classes to model and encapsulate key components of your application (e.g., API interactions, data processing, visualization generation).
        - Apply concepts such as inheritance, encapsulation, and polymorphism where applicable.
        - Ensure your code is modular and reusable.
3. **Data Visualization**:
    - Process the retrieved data to extract meaningful insights.
    - Create at least **two distinct visualizations** (e.g., bar charts, scatter plots, line graphs, or heatmaps) using libraries like Matplotlib, Seaborn, or Plotly.

    - Visualizations must be well-labeled and clearly represent the insights derived from the data.
4. **Error Handling**:
    - Implement robust error handling to manage potential issues with API requests, data processing, and visualization generation.
5. **Documentation**:
    - Include a **README** file that explains:
        - The project's purpose and functionality.
        - Instructions on how to install dependencies and run the project.
        - A description of the API(s) used and data visualizations created.
    - Add meaningful inline comments to explain the logic and flow of your code.
6. **Originality**:
    - Apply your creativity to define the scope and features of the application.
    - Explore a unique topic or use case that interests you.
7. **Functionality**:
    - The application must run smoothly and meet all specified requirements.

Deliverables:

1. A Python project structured into multiple files and published on GitHub. **Be sure to not include your API key(s) in any uploaded code, including processed output** (like `print(key)`).
2. A well-documented **README** file, including how to place the API key file.
3. Upload your API key(s) in the Brightspace assignment. Ensure they are a separate file that can be easily imported into your project.
4. A video presentation showcasing the project functionality. Include a short slideshow, code walkthrough, and demonstration.

Grading Rubric:

Category	Exemplary (10)	Proficient (7)	Developing (4)	Needs Improvement (1)
API Integration (25%)	Successfully integrates APIs with accurate data retrieval and robust error handling.	Integrates APIs with minor data retrieval issues and basic error handling.	Partially integrates APIs; issues with data retrieval; limited error handling.	Fails to integrate APIs; unable to retrieve usable data or uses only static data files; no error handling.
Object-Oriented Programming Use (25%)	Effectively uses OOP principles such as classes, encapsulation, inheritance, and polymorphism to create modular and reusable code.	Uses OOP principles to a satisfactory degree, with room for improvement in modularity or design.	Limited use of OOP principles; code may lack modularity or clarity.	No use of OOP principles; code is procedural and unstructured.
Visualization (15%)	Creates highly effective, clear, and well-labeled visualizations linked to analysis.	Creates clear visualizations with minor improvements needed in relevance or labeling.	Creates basic visualizations that are unclear or poorly connected to the analysis.	No visualizations provided or they are irrelevant/confusing.
Code Quality, Organization, & Error Handling (15%)	Implements well-structured, readable, best-practice code with comprehensive error handling.	Code is organized and readable, following most best practices; adequate error handling.	Code is disorganized or difficult to read with minimal comments or error handling.	Code is messy, unreadable, or non-functional with no comments or error handling.
Originality & Creativity (10%)	Demonstrates exceptional originality and creative problem-solving.	Shows creativity but follows largely standard approaches.	Project is functional but lacks originality or creative approaches.	Project lacks effort or creativity; appears incomplete or rushed.
Documentation (5%)	Provides clear, comprehensive documentation, including a detailed README and comments.	Provides sufficient documentation but lacks detail or clarity in some areas.	Provides limited documentation; difficult to understand project purpose or design.	No documentation provided or documentation is irrelevant/inaccurate.
Overall Functionality (5%)	Project is fully functional, exceeds requirements, and is polished.	Project is functional and meets most requirements with minor flaws.	Project is partially functional but does not fully meet requirements.	Project is non-functional and fails to meet basic requirements.