Project Overview:

You are tasked with creating a web application that processes and displays data, focusing on data type inference and conversion for datasets using Python and Pandas. The project comprises:

1. **Pandas Data Type Inference and Conversion (Backend Task):**
    - Develop a Python script using Pandas capable of inferring and converting data types in a dataset. The script should accurately identify the most appropriate data type for each column in each CSV or Excel file. You are free to use other Python libraries as well. The only restriction is that data must be imported from a CSV or Excel file into a Pandas DataFrame.
    - Address the common issue where many columns in a pandas DataFrame default to 'object' dtype, and design a solution that correctly infers types like dates, numerics, categories, etc.
    - Address the challenge where Pandas might incorrectly infer types, especially with mixed data types or data represented in non-standard date formats.
    - Ensure that your script can handle large files and conduct thorough testing to validate it.
    - Optimize the script for performance, especially for large datasets, to avoid memory issues and long processing times.

    Here are the common Pandas data types:

1. **object:** Typically used to store text or mixed numeric and non-numeric values. In most cases, when you see an object data type in a Pandas DataFrame, it usually means string (text) data, but it can also store arbitrary Python objects.
2. **int64, int32, int16, int8:** These are integer data types, which represent whole numbers. The number (64, 32, 16, 8) indicates the bit length of the integer, which affects the range of values that can be stored.

3. **float64, float32:** These are floating-point numbers, which are used to represent real numbers (numbers with a fractional part). Like integers, the number (64, 32) represents the precision of the float.
4. **bool:** Represents Boolean values (True or False).
5. **datetime64, timedelta[ns]: datetime64** is used for dates and times without time zone information. **timedelta[ns]** represents differences in times, expressed in different units (like days, hours, minutes, seconds).
6. **category:** This is a Pandas-specific data type, used for categorical data. Categorical data contains a finite list of values (categories). It can be more memory-efficient than **object** type for storing string data if the number of distinct strings is small compared to the number of rows in the DataFrame.
7. **complex:** Used for complex numbers. Complex numbers have a real and imaginary part, which are each a float.

Reference Code for Data Type Inference:

The Python script infer_data_types.py is provided as a basic reference and has significant limitations. It's intended as a starting point only. Keep in mind that no script will be 100% perfect in automatically inferring data types, as each dataset is unique. The aim is to develop a solution that performs well across a variety of datasets, striving to achieve as high accuracy as possible, while understanding that some level of customization or manual intervention might be necessary for specific cases.

2. **Django Backend Development:**
    - Set up a Django project incorporating your Python script for data processing.
    - Create Django models, views, and URLs to handle the data processing logic and user interactions.
    - Implement a backend API that works with the frontend, handling data input and output.

3. **Frontend Development using a JavaScript Framework (React Preferred):**
    - Develop a frontend application using a JavaScript framework, preferably React, to interact with the Django backend.

- The frontend should allow users to upload data (CSV/Excel files), submit it for processing, and display the processed data.
- The app could map the backend (Pandas) data types to user-friendly names; for instance, the 'object' data type could be mapped to 'Text', and 'datetime64' could be mapped to 'Date', etc.
- Consider offering users the option to override the data types inferred by the script, allowing them to set their own data types for columns if needed.

Project Requirements:

- Write clean, maintainable, and well-documented code.
- Include comprehensive error handling and validations in both backend and frontend.

Deliverables:

1. **Source Code:** Submit the complete source code via a Git repository.
2. **README File:** Provide a detailed README.md file that includes:
    - Instructions for setting up and running the application.
    - Additional notes or comments about the project.
3. **Demo Video**: Submit a short video demonstrating your web app in action.

Note on the Challenges of Automating Data Cleaning

You might find the problem to be overly broad and not well-structured or well-defined. That's the inherent challenge when automating data cleaning tasks. The difficulty lies in effectively generalizing solutions, considering the uniqueness of each dataset. Despite these datasets being distinct, a comprehensive approach must still be developed for automation. Additionally, for your specific situation, ensure that the data is imported from a CSV or Excel file into a Pandas DataFrame. We have provided only one sample CSV file (sample_data.csv). You need to do your own testing. We will do thorough testing of your code against our test cases, so ensure testing is thorough.