Clean Code Development

Here are some key points why my code is considered to be clean

1. **Meaningful Variable and Function Names:**
    - The code makes suitable use of the variable df, which is a standard shorthand for a data frame. With distinct function names like read_csv, drop, and drop_duplicates, the method chain is organized logically.

2. **Consistent Formatting:**
    - The code follows the PEP 8 style guide, maintaining uniform formatting. Throughout the code, there are consistent line breaks, space, and indentation.

3. **Modularization and Single Responsibility:**
    - Without needless complexity, each code block does a particular task (cleaning and converting data kinds). This complies with the principle of single responsibility.

4. **Handling Missing Values:**
    - By explicitly handling missing values with the fillna method, the code makes sure that there are no unforeseen problems and that the dataset is ready for additional analysis.

5. **Data Transformation and Encapsulation:**
    - The code uses a label encoder to efficiently transform category data. Because this modification is contained in a loop, applying it to numerous columns is simple.

CLEAN CODE DEVELOPMENT CHEAT SHEET

1. **Descriptive Naming:**
    - Give variables, functions, and classes descriptive names that express what they do.
2. **Consistent Formatting:**
    - Adhere to standard naming conventions, space, and indentation in your code.
3. **Modularization:**
    - Code should be divided into more manageable, single-purpose functions or methods while following the Single Responsibility Principle.
4. **Comments and Documentation:**
    - When explaining complicated logic or the purpose of a function, include comments. Keep the codebase's documentation current.
5. **Error Handling:**
    - Put in place appropriate error handling to strengthen the code and stop unanticipated malfunctions.
6. **Avoid Magic Numbers:**
    - To improve the readability of the code, replace magic numbers with named constants or variables.
7. **Avoid Deep Nesting:**
    - Code minimizes nesting, enhancing readability
8. **Unit Testing:**
    - To verify that certain methods and classes are proper, create thorough unit tests.
9. **Version Control:**
    - Use version control tools (like Git) to efficiently track changes and cooperate.
10. **Refactoring:**
    - Refactor code frequently to make it more organized and manageable. Refactoring needs to be a continuous endeavor.
11. **Continuous Learning:**
    - Keep up on emerging technologies, programming languages, and best practices. Never stop trying to get better at coding.
12. **Readable code over clever code:**
    - Prioritize readability over extremely clever code.