DATAMEDIC

A Command-Line Tool for Data Cleaning and Preprocessing

Version: 1.0

Author: Ansh Malik

Date: July 2025

Table of Contents

Contents

1. Introduction

In most data science and machine learning projects, raw data rarely comes in a clean or usable format. Before you can even begin analysing or modelling, you have to deal with missing values, inconsistent types, unscaled features, categorical variables, and potential data leakage — all of which require significant effort and boilerplate code.

To address this, *datamedic* was created as a lightweight yet powerful command-line tool for handling essential preprocessing steps. It allows users to clean data, apply encodings, scale numerical values, perform dimensionality reduction with PCA, detect data leakage and generate an EDA report — all through a consistent, easy-to-use interface.

The tool is especially useful for:

- Data analysts working with CSV datasets
- Machine learning engineers preparing training data
- Students or researchers building reproducible pipelines
- Anyone tired of repeating the same pandas code for cleaning and prep

Unlike traditional Jupyter notebook workflows, *datamedic* provides a modular and CLI-driven approach — making your preprocessing steps traceable, repeatable, and easier to automate.

2. Why this tool was created

Being a Machine Learning student in India, I've spent countless hours working on different dataset, from college assignments to personal projects and hackathons. And every single time, I found myself writing the **same preprocessing code again and again**. Clean the data, handle missing values, encode categorical columns, scale features, run PCA — same steps, different dataset.

At first, it felt normal. But over time, I started thinking — *"Why is nobody talking about this? Why isn't there a simple tool that does all of this in one go?"*

And that's when the idea of ***datamedic*** was born.

This isn't just another Python package. This is a **solution built out of frustration**, late-night debugging, and the realization that **every ML journey starts with the same boring setup**. I didn't want future students or aspiring data scientists to waste time on boilerplate code. I wanted them to focus on what really matters — insights, impact, and innovation.

datamedic is built with that mindset. It's simple, fast, and powerful and most importantly, it's made by someone who has felt the exact pain it tries to solve.

If this tool saves you even an hour of repetitive work, or helps you feel a bit more confident in your ML journey, then it's served its purpose.

And for working professionals juggling deadlines and dashboards,
let *datamedic* handle the groundwork, so you can invest your time where it truly matters: deep analysis, better modelling, and business decisions.

3. Features

The *datamedic* CLI tool provides a complete, structured, and production-grade data preprocessing pipeline tailored for CSV datasets. Every step is modular, transparent, and user-friendly — with summaries and backup support built-in.

3.1. Data Cleaning

- **Missing Value Handling**: Drop rows/columns using *--dropna* or fill missing values with mean, median, mode, or a constant using *--fillna.*
- **Duplicate Removal**: Detects and removes duplicate rows from the dataset.
- **Summary & Backup**: Saves a backup file and prints a clear summary of what changed.

3.2. Encoding

- **Automatic Detection**: If columns are not specified, all object or category columns are automatically encoded.
- **Encodes Only Non-Numeric Columns**: Prevents accidental encoding of already numeric data.
- **Encoding Options**: Choose between label encoding and one-hot encoding.
- **Encoding Summary**: Outputs a clear summary of encoded columns and encoding method used.

3.3. Scaling

- **Scaler Options**: Choose from standard scaling or minmax scaling.
- **Auto Exclusion**: Automatically excludes binary columns (0/1) from scaling unless explicitly included.
- **Column Selection**: Scale specific columns using the --columns flag.

3.4. Principal Component Analysis (PCA)

- **Flexible Dimensionality Reduction**:
    - Use *--components* to retain a fixed number of components.
    - Use *--retain* to retain a specific percentage of variance.
- **Data Check**: Warns the user if data is not scaled or encoded.
- **Explained Variance Ratio**: Displays Explained Variance Ratio.

3.5. Data Leakage Detection

- **Leakage Risk Identification**: Detects columns that may leak information into the target (e.g., perfect correlation, high-cardinality IDs).
- **Detailed Report**: Generates a structured data leakage report.
- **Variable Threshold**: Allows users to specify a customised threshold value according to their needs.

3.6. Exploratory Data Analysis (EDA)

- **Automated Summary Report**: Includes dataset shape, missing values, column types, descriptive stats, and correlations.
- **Visualizations**: Saves graphs like histograms, boxplots, class balance plots, and heatmaps to a separate *generated_graphs/* folder.
- **Smart Features**:
    - Supports skipping graphs with –skip_graphs
    - Automatically samples large datasets for performance
    - Caches heavy operations like correlation
- **Redundant Feature Detection**: Identifies constant or near-constant columns.
- **Dynamic Suggestions**: Based on EDA findings, suggests next preprocessing or modelling steps.

3.7. Backup and Summaries

- **Rollback Support**: Every step saves a backup before applying changes.
- **Step-wise Summary**: Clear explanation of what was changed at each stage.

3.8. Workflow Flexibility

- **Chained Execution**: Steps can be used independently or sequentially.
- **Interactive & Scripted Use**: Suitable for both exploratory use and automation.

3.9. User-Centric CLI Design

- **Defaults and Optional Arguments**: Most arguments have sensible defaults to minimize user effort.
- **Warnings & Suggestions**: Informs the user of potential issues, assumptions, or best practices.
- **Minimal Learning Curve**: Designed using Click with intuitive flag naming and CLI help.

3.10. Step-wise Output Saving

- **File per Step**: After each transformation (cleaning, encoding, scaling, etc.), a new CSV file is automatically saved — ensuring transparency and preserving intermediate results.
- **Consistent Naming**: Files follow a clear naming convention like cleaned.csv, encoded.csv, scaled.csv, etc.

3.11. Smart Conflict Detection

- **Argument Conflict Handling**: If the user provides logically conflicting options (e.g., both --*dropna* and --*fillna*), the tool detects the conflict and exits with a helpful error message.
- **Safe Defaults**: The tool avoids accidental overwrites or invalid combinations by enforcing safe and expected behaviours.

3.12 Testing and Reliability

To ensure reliability and robustness, *datamedic* has undergone extensive manual and edge-case testing.

- **50+ CLI test cases** designed and verified across all major features
- **Real-world validation by 5+ early users** to simulate practical use cases
- Each module was tested for edge behaviour, unexpected inputs, and fallback handling
- Final version passed all test scenarios before public release

4. Installation

datamedic is designed to be lightweight and easy to set up. Whether you're installing it in a fresh virtual environment or adding it to an existing project, setup takes less than a minute.

4.1. Install via pip

If you're using the latest version of Python (>= 3.8), you can install the package directly:

```pip install datamedic```

(Make sure your environment is activated before installing.)

4.2. Install from Source (for development)

If you want to clone the project and modify it locally:

```git clone https://github.com/Ansh-Malik1/datamedic

cd datamedic

pip install . ```

(This is useful if you're planning to contribute, or just want to extend the tool for your own workflows.)

## 4.3. Optional: Create a virtual environment (Recommended)

```python -m venv venv

source venv/bin/activate  # On Windows: venv\Scripts\activate

pip install datamedic```

(Keeping dependencies isolated ensures clean reproducibility and avoids version conflicts.)
```

5. Usage

Once installed, *datamedic* can be used entirely from the command line to process your CSV datasets in a modular, step-by-step fashion.

Each operation — cleaning, encoding, scaling, PCA, leakage detection, or EDA — is triggered by a specific command. The syntax remains predictable, lightweight, and beginner-friendly.

5.1. Basic Example

Imagine you have a raw dataset named 1000_companies.csv. Here's how you might use *datamedic* to clean it:

> Input: 1000_companies.csv
> Action: Apply cleaning
> Output: cleaned.csv

The cleaned file will have null values handled, duplicates removed, and column names fixed based on what option you are choosing.

5.2. Step-by-Step Workflow (Recommended)

You can perform each preprocessing step **individually** in sequence. For example:

1. **Clean the raw data** → Output: cleaned.csv
2. **Encode categorical columns** → Output: encoded.csv
3. **Scale numeric values** → Output: scaled.csv
4. **Apply PCA for dimensionality reduction** → Output: pca.csv
5. **Detect potential data leakage**
6. **Run EDA to generate summary visuals and reports**

This modular pipeline allows full control, ensures traceability, and prevents irreversible mistakes. Additionally, you can run EDA command in the start as well to see the discrepancies present

5.3. What Happens at Each Step?

- **Reads** your input CSV file
- **Backup** your current file before operation begins.
- **Applies** the selected transformation (based on your options)
- **Saves** the result to a new CSV file (named by you)
- **Displays** a short summary directly in the terminal

You'll always have a clear idea of what was done and what changed.

5.4. List of Available Commands

Command	Purpose
clean	Clean missing values, duplicates, outliers
encode	Encode categorical columns (label/one-hot)
scale	Normalize/standardize numerical features
pca	Apply dimensionality reduction
detect-leakage	Identify potential data leakage risks
eda	Generate visual EDA summary reports

To explore options for any command, just run:

```datamedic <command> --help```

Example:

```datamedic clean –help```

Tip

Use **clear filenames** for intermediate outputs — like cleaned.csv, encoded.csv, scaled.csv. It helps *with tracking progress and rolling back if needed.*

6. Available Commands

Below is a detailed list of all commands supported by *datamedic*. Each command is modular and can be run independently or as part of a larger pipeline.

6.1. clean — Clean the dataset

Purpose:
Handles missing values, removes duplicates, drops or fills nulls, and optionally detects/removes outliers.

Key Options:

- --dropna → Drop rows with missing values
- --fillna → Fill missing values with strategy (mean, median, mode, constant)
- --drop-duplicates → Remove duplicate rows
- --outliers → Detect and optionally drop statistical outliers

Smart Handling:
If both --dropna and --fillna are passed, the tool will throw an error and ask the user to choose only one. This prevents unintended data loss.

Output:
A new cleaned CSV file saved to the specified path.

6.2. encode — Encode categorical columns

Purpose:
Transforms categorical/text data into numeric format using Label or One-Hot encoding.

Key Options:

- --method → Choose encoding type: label or onehot
- --columns → (Optional) Specify columns to encode. If not specified, it automatically encodes all object/category dtype columns.

Output:
New encoded CSV file with numeric features.

6.3. scale — Scale numerical features

Purpose:
Standardizes or normalizes numerical columns for model readiness.

Key Options:

- --method → standard, minmax, or robust
- --columns → (Optional) Specify which columns to scale. If not provided, all suitable numeric columns (excluding binary one-hot) are scaled.

Output:
A new CSV file with scaled numeric columns.

6.4. pca — Principal Component Analysis

Purpose:
Reduces dimensionality of data while retaining maximum variance.

Key Options:

- --components → Number of PCA components
- --retain → Percentage of variance to retain (e.g., 0.95)
- Automatically shows warning if data is not scaled/encoded.

Output:
CSV with reduced dimensions.

6.5. detect-leakage — Check for data leakage

Purpose:
Scans for columns in the dataset that are highly correlated with the target (especially in training data), which might lead to data leakage.

Key Options:

- --target → Specify the target column name
- --threshold → Correlation threshold for leakage (default: 0.9)

Output:

6.6. eda — Exploratory Data Analysis

Purpose:
Generates graphical and statistical summaries of your dataset in one go.

Key Outputs:

- Distribution plots
- Correlation heatmap
- Missing value map
- Constant/low variance column alerts
- Dynamic "Suggestions & Next Steps" based on data patterns

Options:

- --skip_graphs→ Skip graph generation (for speed on large datasets)

Output:
Saves plots in a dedicated *generated_graphs/* folder and a text report with insights.

7. Example Workflow

Let's take a real-world example: you're working with a dataset named *1000companies.csv*, which contains financial and categorical details about different companies.

Your goal is to preprocess the data before feeding it into a machine learning model. Below is the step-by-step workflow using *datamedic:*

Step 1: Clean the Data

```
datamedic clean 1000companies.csv --dropna --drop-duplicates --
output cleaned.csv
```

What it does:

- **Removes rows with any missing values** (--dropna).
- **Eliminates duplicate rows** if present (--drop-duplicates).
- Saves the cleaned version as cleaned.csv.

What you'll see:

A terminal summary showing:

- Number of rows dropped due to nulls or duplicates
- Total rows/columns before and after cleaning
- A backup of 1000companies.csv saved automatically

Note: If you want to fill missing values instead of dropping them, use --fillna mean or --fillna mode.

Step 2: Encode Categorical Columns

```
datamedic encode cleaned.csv --method onehot --output encoded.csv
```

What it does:

- Scans all categorical columns (object or category dtype).
- Applies **One-Hot Encoding** by default, turning each category into a new column.
- Outputs a new CSV file: encoded.csv.

What you'll see:

- A summary showing:
    - Which columns were encoded
    - How many new columns were created
- Original column is dropped (replaced by the new binary columns)

Note: If no column is specified, all eligible categorical columns are encoded automatically.

Step 3: Scale Numerical Columns

```
datamedic scale encoded.csv --method standard --output scaled.csv
```

What it does:

- Applies **Standard Scaling** (zero mean, unit variance) to all numeric columns — except those that look like one-hot encoded (0/1 only).
- Outputs a new CSV: scaled.csv.

What you'll see:

- Summary of:
    - Which columns were scaled
    - Mean and standard deviation before scaling (if needed)
- Ensures features are on the same scale, which is important for many ML models.

Note: If your dataset contains columns with large differences in units (like revenue in millions and profit margin in percentage), scaling helps models learn better.

Step 4: Apply PCA (Dimensionality Reduction)

```
datamedic pca scaled.csv --retain 0.95 --output pca.csv
```

What it does:

- Applies Principal Component Analysis (PCA) to reduce feature space while retaining **95% of variance**.
- Outputs compressed dataset to pca.csv.

What you'll see:

- A summary showing:
    - How many components were selected
    - Explained variance for each component
    - Total variance retained

Note: Warning shown if the input isn't scaled or encoded to ensures you don't accidentally apply PCA on raw data.

Step 5: Detect Data Leakage

```
datamedic detect-leakage pca.csv --target Profit
```

What it does:

- Analyzes correlations between the Profit column and all other features.
- Identifies any suspiciously high correlations that might indicate leakage.

What you'll see:

- A ranked list of features highly correlated with Profit
- Clear **warnings** if potential leakage is detected
- Suggests which columns may be derived from or strongly dependent on the target

Step 6: Perform EDA (Exploratory Data Analysis)

```
datamedic eda cleaned.csv --output eda_report
```

What it does:

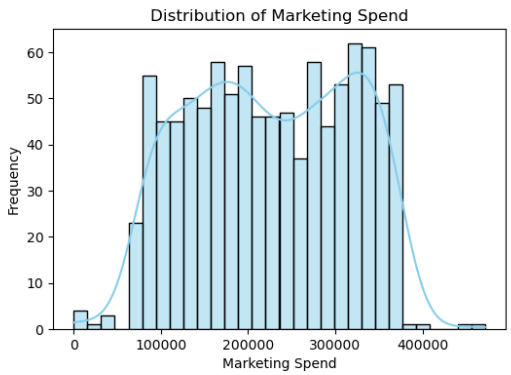
- Generates:
    - Distribution plots for numerical columns
    - Count plots for categorical columns
    - Correlation heatmaps
    - Missing value heatmaps
    - Outlier detection and class imbalance checks
- Saves all plots and a summary report in the folder eda_report/

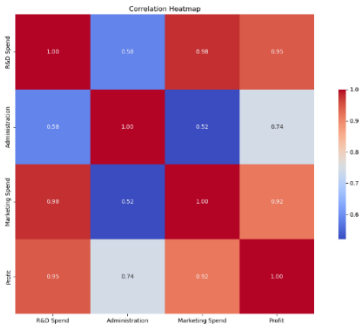
What you'll see:

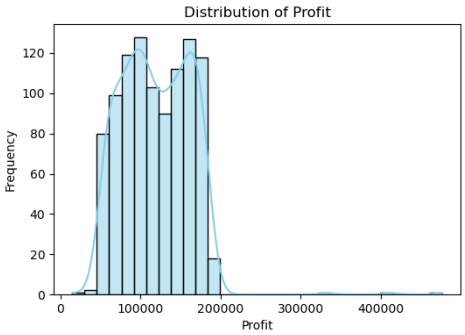
- Graphs stored in eda_report/
- A summary.txt file with:
    - Insights
    - Warnings (e.g., low-variance columns, missing data)
    - Suggestions for preprocessing or modeling

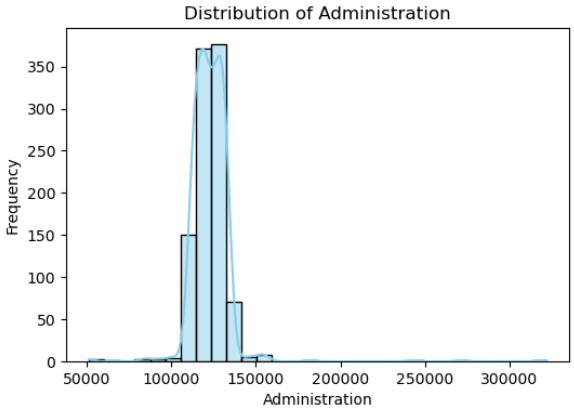
Example Suggestion:
"Column Revenue has several extreme outliers. Consider using log transformation or outlier capping."

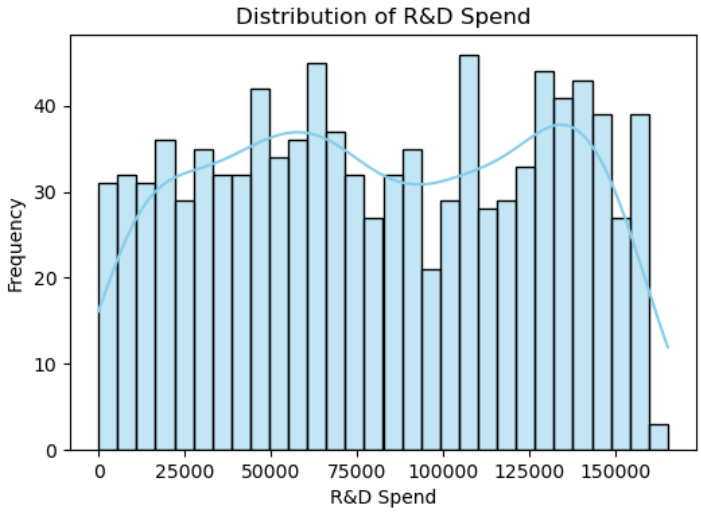
Example Graphs/Plots:











Recap

By chaining these commands, you've:

- Cleaned the raw data
- Encoded and scaled features
- Applied dimensionality reduction
- Checked for data leakage
- Generated EDA reports

And all of this — from raw data to modelling-ready — was done with just a few CLI commands.

Thus, wiith datamedic, you focus on decisions, not boilerplate code.
Let the tool handle repetitive preprocessing while you dive deep into real analysis.
Saves hours of setup time in every new project — especially useful for students, analysts, and ML practitioners.
Ideal for hackathons, prototypes, or even day-to-day corporate workflows.

8. Roadmap

Phase 1
- CLI Framework Setup
  - Created the core commad-line interface using '*Click*' library.

Phase 2
- Data Preprocessing Commands
  - Implemented clean,encode, scale and pca modules.

Phase 3
- Data Leakage Detection
  - Added support for detecting target leakage based on correlation.

Phase 4
- Exploratory Data Analysis (EDA)
  - Implemented EDA module with graphs,summaries and smart recommendations.

Phase 5
- Smart Enhancement and Automation
  - Added features like automatic column detection,PCA warning,one hot check.

Phase 6
- Final Polishing and Optimisation
  - EDA speed improved, sampling, figure cleanup, CLI safety checks.

Phase 7
- Documentation and Deployment
  - Created professional doc, README.md and publish it PyPI & Github.

Contribution Guidelines

We welcome and encourage contributions to ***datamedic*** from the community!

Whether you're fixing a bug, adding a new feature, improving documentation, or suggesting enhancements, your help is valuable.

How to Contribute

1. **Fork the Repository**
   Go to https://github.com/Ansh-Malik1/datamedic and click the **Fork** button to create your copy.

2. **Clone Your Fork Locally**

```
git clone https://github.com/Ansh-Malik1/datamedic
cd datamedic
```

3. **Create a New Branch**

```
git checkout -b feature-name
```

4. **Make Your Changes**
   Please follow our coding style (PEP8 for Python) and structure. Modular functions and meaningful commit messages are appreciated.

5. **Test Your Code**
   Ensure your change works correctly with existing commands and doesn't break other functionality.

6. **Submit a Pull Request (PR)**
   Push your branch and open a PR. Briefly describe what you changed and why.

Contribution Tips

- For bug fixes, try to include a reproducible example or test case.
- For new features, open an issue first if it's a major addition — this helps align the direction.
- Stick to clean CLI syntax and user-friendly behavior.
- Avoid using obscure libraries unless absolutely necessary.

Code of Conduct

We follow a simple rule: **Be respectful and constructive.**
This project is a learning resource for many, so kindness goes a long way.

FAQs

1. **Do I need to know Python to use datamedic?**
   Ans. No. datamedic is designed as a CLI tool — you just need to run commands in your terminal. No coding knowledge is required.
2. **Which file formats are supported?**
   Ans. Currently, datamedic works with .csv files only. Support for .xlsx or .json may be added in future versions.
3. **Does it support multilingual datasets or non-English column names?**
   Ans. It depends on the encoding of your CSV file. As long as it's UTF-8 encoded, it should work fine.
4. **What if my dataset is very large (1M+ rows)?**
   Ans. datamedic is optimized for performance and handles large files efficiently. However, during EDA, it automatically samples rows (if needed) and avoids memory-heavy operations to ensure speed.
5. **How is this different from pandas/sklearn preprocessing?**
   Ans. datamedic wraps many preprocessing steps into simple commands, so you don't have to write repetitive code. It's ideal for fast experimentation and reproducible pipelines.
6. **Do I need to run commands in a specific order?**
   Ans. Yes. It is recommended to clean → encode → scale → apply PCA → detect leakage → perform EDA. Some steps like PCA may not work properly without encoding or scaling.
   Additionally, you can perform EDA before starting as well to get an idea of the data
7. **What happens if I run conflicting options like --dropna and --fillna together?**
   Ans. datamedic automatically detects such conflicts and shows a clear error message. It ensures that only one valid preprocessing strategy is applied at a time.
8. **Where are the output files saved?**
   Ans. Every transformation step (cleaning, encoding, scaling, etc.) saves the resulting file in the current working directory. You can specify the output filename using the --output flag.
9. **Does it overwrite the original CSV file?**
   Ans. No, datamedic never modifies your original file. It always creates a new output file, ensuring your raw data stays untouched.
10. **Can I automate my full preprocessing pipeline?**
    Ans. Yes! Just chain the commands step by step on intermediate files, or write a shell script that sequentially calls each step.
11. **Can I revert changes after applying a step?**
    Ans. Yes. Backups are saved at every step and can be restored manually.
12. **How do I report a bug or request a feature?**
    Ans. Open an issue on the GitHub repository. Contributions and suggestions are welcome!
13. **Will this tool support GUI or web interface in future?**
    Ans. Yes. A lightweight GUI (via Streamlit) is under consideration for future versions.
14. **Why does datamedic create backups at every step?**
    Ans. Every transformation in a data pipeline alters the dataset — sometimes subtly, sometimes drastically. Datamedic preserves the *input* to each step as a versioned snapshot, enabling complete traceability, reproducibility, and rollback.
    These backup files are not redundancies — they form a transparent audit trail of your data's evolution. Whether you're debugging, comparing alternative processing paths, or handing off work to a teammate, these saved inputs act as a reliable source of truth.

Troubleshooting

While using ***datamedic***, you may occasionally run into common issues. Below is a list of frequent problems and how to resolve them:

1. ModuleNotFoundError: No module named 'datamedic'

Cause: The package isn't installed or your virtual environment isn't activated.
Solution:

- Make sure you're in the correct virtual environment.
- Run:

```
pip install datamedic
```

2. FileNotFoundError: 'yourfile.csv' does not exist

Cause: The specified input CSV file path is incorrect or missing.
Solution:

- Double-check the filename and its location.
- Use relative or absolute path correctly.

```
datamedic clean data/1000companies.csv --output cleaned.csv
```

3. CLI Flags Conflict (e.g.,--dropna and--fillna)

Cause: Conflicting options passed simultaneously.
Solution:

- datamedic is designed to detect such conflicts and raise an informative error.
- Use only one missing value handling strategy per run.

4. Nothing happens after running a command

Cause: Possibly incorrect command syntax or missing mandatory arguments.
Solution:

- Use the help flag to verify correct usage:

```
datamedic clean --help
```

5. Graphs not showing up in EDA output

Cause: Headless environments or skipped due to --no-graphs flag.
Solution:

- Check if --no-graphs was passed.
- Ensure matplotlib and seaborn are installed correctly.
- Check the output directory for saved images instead of expecting them to open directly.

6. PCA Not Working Properly

Cause: PCA requires encoded and scaled data.
Solution:

- Make sure you've run encode and scale commands before running pca.
- datamedic gives a warning if these steps seem to be skipped.

7. Output file not saving

Cause: Missing --output argument or permission issues.
Solution:

- Always provide an output path or filename.
- Ensure the directory has write permissions.

8. Encoding or Scaling not applied as expected

Cause: Columns might not be detected by the program.
Solution:

- Check CLI messages for "No applicable columns found".
- Ensure you're targeting the correct file from the previous step.
- For encoding, make sure the columns are of object or category dtype.

9. UnicodeDecodeError while reading CSV

Cause: CSV file might have some special characters inside.
Solution: Pass the correct encoding manually, e.g.:

```
datamedic clean yourfile.csv --encoding utf-8
```

10. PCA Warning Despite Scaling

If PCA throws a scaling warning even after you've scaled your data:

- Ensure you used datamedic's own scale command.
- The warning is skipped only if the tool detects scaling history in summaries.

License

Data privacy and security

datamedic operates fully offline and does not initiate any network requests. All data transformations occur locally, ensuring complete user data privacy.

Contact/Support

For feedback, suggestions, or collaboration:

Ansh Malik
Email: anshmalik845gmail.*com*
GitHub: https://github.com/Ansh-Malik1
Feedback form: https://forms.gle/CTG7bZRPqqGQofnS7

Still unsure? Just run datamedic clean yourdata.csv. Let the tool speak for itself.