Vaccine Safety Analysis Project

Overview

This project evaluates the safety of the TAK vaccine using a synthetic dataset of Serious Adverse Events (SAEs) from vaccine reports. The analysis follows a rigorous, data-driven approach to clean, explore, model, and interpret SAE data, aiming to identify risk factors, estimate adverse event probabilities, and assess causality. The workflow incorporates best practices in pharmacovigilance and data science as of 2025, leveraging advanced techniques like causal inference, survival analysis, and NLP to provide actionable insights for vaccine safety monitoring.

The project is implemented in Python, using libraries such as `pandas`, `scikit-learn`, `xgboost`, `causalml`, `lifelines`, `sentence-transformers`, and `plotly` for data processing, modeling, and visualization. The analysis produces detailed markdown reports and interactive visualizations to facilitate interpretation by stakeholders, including a Dash dashboard for dynamic exploration. The code is modularized for maintainability, with separate modules for data cleaning, EDA, feature engineering, modeling, causal inference, and utilities.

Project Objectives

- **Data Quality**: Ensure the dataset is clean, consistent, and reliable for analysis.
- **Exploratory Analysis**: Identify patterns, correlations, and risk factors associated with SAEs.
- **Predictive Modeling**: Develop models to predict severe AEs and identify key features.
- **Causal Inference**: Estimate the causal effect of risk factors (e.g., allergies) on AEs using methods like PSM, TMLE, Causal Forest, DoubleML, and DoWhy.
- **Temporal Analysis**: Assess AE onset timing and update safety beliefs over time.
- **Reporting**: Generate comprehensive reports, logs, and visualizations for actionable insights.

Project Structure

```
vaccine_sae_analysis/
├── src/
│   ├── init.py                # Makes src a package
│   ├── data_cleaning.py           # Data loading, cleaning, consistency checks
│   ├── eda.py                     # Exploratory data analysis, stats, 
visualizations
│   ├── feature_engineering.py     # Feature creation, encoding, NLP
│   ├── modeling.py                # Predictive modeling, SHAP, RFE, Bayesian 
logistic, survival, clustering, anomaly detection
│   ├── causal_inference.py        # Causal models (PSM, TMLE, Causal Forest, 
DoubleML, DoWhy)
│   ├── utils.py                   # Logging, saving summaries
│   └── main.py                    # Orchestrates the full pipeline
├── dataset/
│   └── synthetic_vaccine_sae_data.csv  # Synthetic SAE dataset
├── plots/                               # Output directory for visualizations
```

```
(auto-created)
├── summaries/                # Output directory for markdown reports
and logs (auto-created)
├── requirements.txt          # Dependencies
└── README.md                 # Project documentation
```

Analysis Plan and Implementation Steps

The pipeline is executed via `main.py`, which calls functions from other modules in sequence. Each module has docstrings, comments, and logging for clarity.

Step 1: Data Cleaning and Quality Assurance (data_cleaning.py)

1.1 Convert Types and Handle Missing Values

- **What**: Convert columns to appropriate data types (numerical, categorical, datetime) and impute missing values.
- **Why**: Ensures data consistency for downstream analysis; missing values can bias results in pharmacovigilance studies.
- **How**:
  - Numerical columns are coerced to numeric using `pd.to_numeric`.
  - Categorical columns filled with 'Unknown' and cast to strings.
  - Date columns converted with `pd.to_datetime` and forward-filled.
  - Text columns filled with 'No description'.
  - Drop rows with all dates missing to maintain temporal integrity.
- **Tools**: `pandas`, `numpy`.

1.2 Check Data Consistency

- **What**: Enforce boundaries (e.g., `age` 0-120).
- **Why**: Prevents unrealistic values that skew analyses.
- **How**: Clip values using `clip()`.
- **Tools**: `pandas`.

1.3 Check Data Logic and Conflicts

- **What**: Validate logical relationships (e.g., gender exclusivity, timeline validity).
- **Why**: Detects reporting errors common in AE data.
- **How**:
  - Ensure gender is mutually exclusive.
  - Nullify second-dose fields if doses < 2.
  - Drop invalid timings (`onset_date` > `vaccine_1_date`).
  - Reset indices to avoid duplicates.
- **Tools**: `pandas`.

1.4 Enhanced Processing

- **What**: Compute quality metrics and detect outliers.
- **Why**: Assesses data reliability; outliers can distort models.
- **How**:
    - Calculate completeness, skewness, kurtosis.
    - Use Isolation Forest for anomaly detection.
- **Tools**: `pandas`, `scipy`, `sklearn`.

Step 2: Exploratory Data Analysis (eda.py)

2.1 Prepare Data for EDA

- **What**: Create targets (e.g., `has_severe_AE`) and basic features.
- **Why**: Prepares data for stats and visualizations.
- **How**: Add binary targets, temporal features, comorbidity index.
- **Tools**: `pandas`.

2.2 Compute Descriptive Stats

- **What**: Calculate means, skew, kurtosis, value counts.
- **Why**: Provides baseline insights into data distribution.
- **How**: Use `df.describe()`, `skew()`, `kurtosis()`, save to MD.
- **Tools**: `pandas`, `scipy`.

2.3 Compute Association Stats

- **What**: AE rates, correlations, chi-square tests.
- **Why**: Identifies relationships between variables and AEs.
- **How**: Groupby for rates, `corr()` for correlations, `chi2_contingency` for categoricals.
- **Tools**: `pandas`, `scipy`.

2.4 Generate Visualizations

- **What**: Create and save plots (heatmap, violin, pairplot, bar, boxplot, histogram, Sankey).
- **Why**: Visualizes patterns for better understanding.
- **How**: Use Seaborn for static plots, Plotly for interactive; save interpretations in MD.
- **Tools**: `seaborn`, `matplotlib`, `plotly`.

Step 3: Feature Engineering (feature_engineering.py)

- **What**: Create advanced features (temporal, encoded, text embeddings).
- **Why**: Enhances model input with derived insights from raw data.
- **How**:
    - Temporal: Calculate onset durations.
    - Encoding: One-hot for categoricals.
    - Text: TF-IDF and Sentence Transformers for descriptions.
    - Normalize numericals.
- **Tools**: `sklearn`, `sentence_transformers`.

Step 4: Modeling (modeling.py)

4.1 Prepare Data

- **What**: Split into X/y, train/test sets.
- **Why**: Sets up for modeling.
- **How**: Drop non-numeric, stratify split.
- **Tools**: `sklearn`.

4.2 Train XGBoost

- **What**: Tune and train XGBoost for prediction.
- **Why**: Handles complex patterns in AE data.
- **How**: Optuna for hyperparams, fit on train.
- **Tools**: `xgboost`, `optuna`.

4.3 Evaluate Model

- **What**: Compute AUC, F1 on test set.
- **Why**: Measures predictive performance.
- **How**: Use `roc_auc_score`, `f1_score`.
- **Tools**: `sklearn`.

4.4 Explain with SHAP

- **What**: Generate SHAP summary plot.
- **Why**: Interprets model decisions.
- **How**: `shap.Explainer` and `summary_plot`.
- **Tools**: `shap`.

4.5 Perform RFE

- **What**: Select top features.
- **Why**: Identifies key risk factors.
- **How**: Recursive elimination with logistic regression.
- **Tools**: `sklearn`.

4.6 Train Bayesian Logistic

- **What**: Train PyTorch logistic model.
- **Why**: Provides probabilistic predictions.
- **How**: Custom NN class, Adam optimizer.
- **Tools**: `torch`.

4.7 Survival Analysis

- **What**: Kaplan-Meier curve.
- **Why**: Analyzes time-to-event for AEs.

- **How**: Manual estimator, plot survival.
- **Tools**: `numpy`, `matplotlib`.

4.8 Clustering

- **What**: KMeans on features.
- **Why**: Groups similar patients for risk profiling.
- **How**: Fit KMeans with 4 clusters.
- **Tools**: `sklearn`.

4.9 Anomaly Detection

- **What**: Isolation Forest for outliers.
- **Why**: Flags rare/unusual AEs.
- **How**: Fit and predict anomalies.
- **Tools**: `sklearn`.

Step 5: Causal Inference (causal_inference.py)

- **What**: Run PSM, TMLE, Causal Forest, DoubleML (DoWhy commented out for speed).
- **Why**: Estimates causal effects beyond associations.
- **How**:
  - PSM: Propensity matching with KDTree.
  - TMLE: Double-robust estimation with logistic models.
  - Causal Forest: Machine learning for heterogeneous effects.
  - DoubleML: Debiased ML for ATE.
- **Tools**: `statsmodels`, `econml`, `doubleml`.

Step 6: Interpretation and Reporting

- **What**: Save logs, summaries; run sensitivity; create dashboard.
- **Why**: Consolidates results for users.
- **How**: Markdown files, Dash app, sensitivity by rerunning without 'age'.
- **Tools**: `logging`, `dash`.

Prerequisites

- Python 3.8+
- Install dependencies: `pip install -r requirements.txt`

Usage

1. Place `synthetic_vaccine_sae_data.csv` in `dataset/`.
2. Run `python src/main.py`.
3. View outputs in `plots/` and `summaries/`.
4. Dashboard at `http://127.0.0.1:8050`.

Notes

- Synthetic data; extend to real for production.
- DoWhy commented out to avoid long runtimes; uncomment if needed.
- Limitations: Observational data biases, small sample.

License

MIT License