ML Project Documentation

# Data Processing Module Documentation

# Data Processing Module Documentation

This module (`data\_processing.py`) contains essential functions for loading and preprocessing your dataset. It is designed to prepare raw data for subsequent analysis or modeling, particularly in contexts like fraud detection.

## Overview

The module provides two main functions:

- \*\*`load\_data(file\_path)`\*\*: Loads a CSV file into a Pandas DataFrame.

- \*\*`feature\_engineering(df)`\*\*: Enhances the dataset by creating new features that may improve model performance.

## Function Details

### `load\_data(file\_path)`

- \*\*Purpose:\*\*

Loads the dataset from a CSV file into a Pandas DataFrame for further processing.

- \*\*Parameters:\*\*

- `file\_path` (str): The path to the CSV file containing the dataset.

- \*\*Returns:\*\*

A Pandas DataFrame containing the loaded data.

- \*\*Usage Example:\*\*

```python

df = load\_data('data/creditcard.csv')

```

### `feature\_engineering(df)`

- \*\*Purpose:\*\*

Enhances the dataset by adding new, derived features. These transformations are aimed at capturing temporal patterns and normalizing numerical data.

- \*\*New Features Created:\*\*

- \*\*HourOfDay:\*\*

Derived from the 'Time' column. It calculates the hour of day by converting seconds into hours (dividing by 3600) and applying modulo 24.

- \*\*Amount\_scaled:\*\*

Standardizes the 'Amount' feature using `StandardScaler` from scikit-learn.

- \*\*RollingMeanAmount:\*\*

Computes the rolling mean of the 'Amount' over a window of 5 transactions.

- \*\*Parameters:\*\*

- `df` (Pandas DataFrame): The DataFrame containing the original dataset.

- \*\*Returns:\*\*

The modified DataFrame with the new features added.

- \*\*Usage Example:\*\*

```python

df = load\_data('data/creditcard.csv')

df = feature\_engineering(df)

```

## Summary

The `data\_processing.py` module is a critical component in your data pipeline. By loading and transforming the raw data, it helps ensure that subsequent analyses and machine learning models are built on a well-prepared foundation.

# Data Balancing Module Documentation

# Data Balancing Module Documentation

This module (`data\_balancing.py`) provides a utility to address imbalanced datasets using SMOTE (Synthetic Minority Over-sampling Technique). Balancing the data is crucial for tasks like fraud detection, where one class is significantly underrepresented.

## Overview

Imbalanced data can lead to biased models. The `balance\_data` function leverages SMOTE to create synthetic examples of the minority class, resulting in a more balanced dataset.

## Function Details

### `balance\_data(X, y)`

- \*\*Purpose:\*\*

Balances the dataset by oversampling the minority class using SMOTE.

- \*\*Parameters:\*\*

- `X`: The feature set containing the predictors.

- `y`: The target variable containing class labels.

- \*\*Returns:\*\*

A tuple `(X\_res, y\_res)` with the resampled features and target.

- \*\*Usage Example:\*\*

```python

from data\_balancing import balance\_data

X\_res, y\_res = balance\_data(X, y)

```

## Summary

Using the `balance\_data` function helps effectively handle imbalanced datasets by synthesizing new minority class examples.

# Model Ensemble Module Documentation

# Model Ensemble Module Documentation

This module (`model\_ensemble.py`) provides a function to train a Random Forest classifier with hyperparameter tuning using Grid Search.

## Overview

The function:

- Instantiates a `RandomForestClassifier`.

- Defines a grid of hyperparameters:

- `n\_estimators`: Number of trees.

- `max\_depth`: Maximum depth (None means no limit).

- `min\_samples\_split`: Minimum samples required to split a node.

- Uses `GridSearchCV` with 3-fold cross-validation, optimizing ROC AUC.

- Returns the best estimator.

## Function Details

### `train\_random\_forest(X\_train, y\_train)`

- \*\*Purpose:\*\*

Trains a Random Forest model with hyperparameter tuning.

- \*\*Parameters:\*\*

- `X\_train`: Training feature matrix.

- `y\_train`: Training labels.

- \*\*Returns:\*\*

The best estimator from `GridSearchCV`.

- \*\*Usage Example:\*\*

```python

from model\_ensemble import train\_random\_forest

best\_rf\_model = train\_random\_forest(X\_train, y\_train)

```

## Summary

`train\_random\_forest` leverages ensemble learning and hyperparameter optimization to build a robust classifier.

# Anomaly Detection Module Documentation

# Anomaly Detection Module Documentation

This module (`anomaly\_detection.py`) provides two key functions for unsupervised anomaly detection:

1. \*\*Isolation Forest:\*\* Detects anomalies by isolating observations.

2. \*\*Autoencoder:\*\* Uses a neural network to learn compressed representations and identifies anomalies via reconstruction error.

## Overview

- \*\*`train\_isolation\_forest(X)`\*\*:

Trains an Isolation Forest model to flag anomalies.

- \*\*`build\_autoencoder(input\_dim)`\*\*:

Constructs a simple autoencoder using TensorFlow and Keras.

## Function Details

### `train\_isolation\_forest(X)`

- \*\*Purpose:\*\*

Train an Isolation Forest on the dataset.

- \*\*Parameters:\*\*

- `X`: Feature matrix for training.

- \*\*Returns:\*\*

A trained Isolation Forest model.

- \*\*Usage Example:\*\*

```python

from anomaly\_detection import train\_isolation\_forest

iso\_forest = train\_isolation\_forest(X)

```

### `build\_autoencoder(input\_dim)`

- \*\*Purpose:\*\*

Builds and compiles a simple autoencoder.

- \*\*Parameters:\*\*

- `input\_dim`: Number of features in the input data.

- \*\*Returns:\*\*

A compiled autoencoder model.

- \*\*Usage Example:\*\*

```python

from anomaly\_detection import build\_autoencoder

autoencoder = build\_autoencoder(input\_dim)

```

## Summary

This module provides robust methods for anomaly detection using both Isolation Forest and Autoencoder approaches.

# Explainability Module Documentation

# Explainability Module Documentation

This module (`explainability.py`) provides a function to generate visual explanations for tree-based models using SHAP.

## Overview

The function `explain\_model(model, X\_sample)`:

- Initializes SHAP's `TreeExplainer` with the given model.

- Computes SHAP values for a sample of data.

- Displays a summary plot of feature importance.

## Function Details

### `explain\_model(model, X\_sample)`

- \*\*Purpose:\*\*

Explains predictions by visualizing SHAP values.

- \*\*Parameters:\*\*

- `model`: A trained tree-based model.

- `X\_sample`: A sample of the input data.

- \*\*Returns:\*\*

Displays a SHAP summary plot.

- \*\*Usage Example:\*\*

```python

from explainability import explain\_model

explain\_model(rf\_model, X\_sample)

```

## Summary

`explain\_model` helps in understanding the contributions of individual features to model predictions.

# Main Module Documentation

# Main Module Documentation

This module (`main.py`) is the entry point of the project and orchestrates the fraud detection pipeline.

## Overview

The script follows these steps:

1. \*\*Data Ingestion & Feature Engineering:\*\*

Loads and transforms raw data.

2. \*\*Data Balancing:\*\*

Uses SMOTE to oversample the minority class.

3. \*\*Model Development (Ensemble Approach):\*\*

Trains a Random Forest classifier with hyperparameter tuning.

4. \*\*Unsupervised Anomaly Detection:\*\*

Implements:

- \*\*Isolation Forest:\*\* Flags outliers.

- \*\*Autoencoder:\*\* Detects anomalies via reconstruction error.

5. \*\*Model Explainability:\*\*

Uses SHAP to visualize feature importance.

## Detailed Workflow

### Data Ingestion & Feature Engineering

- \*\*Loading Data:\*\*

Uses `load\_data` to read the CSV file.

- \*\*Feature Engineering:\*\*

`feature\_engineering` adds features like `HourOfDay`, `Amount\_scaled`, and `RollingMeanAmount`, plus PCA features (`V1` to `V28`).

### Data Splitting and Balancing

- \*\*Splitting:\*\*

Splits the data into training and testing sets.

- \*\*Balancing:\*\*

Applies SMOTE via `balance\_data` on the training data.

### Model Development - Ensemble Approach

- \*\*Random Forest Training:\*\*

Uses `train\_random\_forest` to train and tune the model.

- \*\*Evaluation:\*\*

Prints a classification report based on the test set.

### Unsupervised Anomaly Detection

- \*\*Isolation Forest:\*\*

Trains on test data and prints anomaly scores.

- \*\*Autoencoder:\*\*

Trains on normal transactions and computes reconstruction errors.

### Model Explainability

- \*\*SHAP Explanation:\*\*

Selects a sample from test data and generates a SHAP summary plot.

## Running the Script

The script is executed via:

```python

if \_\_name\_\_ == "\_\_main\_\_":

main()

```

## Summary

`main.py` ties all components together into a complete fraud detection pipeline, covering data processing, model training, anomaly detection, and explainability.