**Data Processing Module:**

This module, named data\_processing.py, provides a comprehensive set of tools for handling user input related to machine learning model training and processing the input data accordingly. The module consists of two main classes, UserInput and DataProcessor, each serving a specific purpose.

**UserInput Class:**

The UserInput class is designed to gather user input for model training. It includes methods to prompt the user for a valid file path, the type of machine learning task (regression or classification), and the target column in the dataset. The class ensures that the user provides valid input and handles various scenarios such as non-existent files or incompatible data types for the chosen machine learning type.

Methods:

* \_\_init\_\_: Initializes the class instance variables.
* run\_all: Executes the sequence of methods to gather user input.
* get\_file\_path: Prompts the user for a valid file path and loads CSV data.
* get\_ml\_type: Prompts the user for the type of machine learning: 'regressor' or 'classifier'.
* get\_target\_column: Prompts the user for the target column and validates its compatibility with the selected ML type.

Usage Example:

* “from data\_processing import UserInput”
* “user\_input = UserInput()”
* “user\_input.run\_all()”

**DataProcessor Class:**

The DataProcessor class processes the input data based on the user's choices. It handles tasks such as checking for missing values, filling missing data, scaling numeric features (for regression tasks), and creating dummy variables for categorical columns. The class also allows users to make a copy of the original data file before processing.

Methods:

* \_\_init\_\_: Initializes the class instance variables.
* run\_all: Executes all data processing steps.
* check\_missing\_values: Checks for missing values in the dataset and prompts the user for actions.
* fill\_missing\_values: Fills missing values in the dataset using various strategies.
* check\_value\_types: Checks and validates the data types of columns, handling categorical data.
* create\_dummy\_variables: Creates dummy variables for categorical columns.
* save\_copy: Saves a copy of the original file.
* scaler: Scales numeric features using StandardScaler if the machine learning type is "regressor."

Usage Example:

* “from data\_processing import DataProcessor”
* “data\_processor = DataProcessor(user\_input.data, user\_input.file\_path, user\_input.ml\_type, user\_input.target\_column)”
* “data\_processor.run\_all()”

**Usage Guidelines**:

1. Import the Module:

* “from data\_processing import UserInput, DataProcessor.”

1. Initialize UserInput:

* “user\_input = UserInput()”
* “user\_input.run\_all()”

1. Initialize DataProcessor:

* “data\_processor = DataProcessor(user\_input.data, user\_input.file\_path, user\_input.ml\_type, user\_input.target\_column)”
* “data\_processor.run\_all()”

1. Follow Console Prompts:

* Respond to prompts and instructions in the console to ensure accurate and seamless data processing.

**Important Notes:**

* This module interacts with the model\_trainer module; ensure it is present for subsequent model training.
* Customize the usage according to your specific needs or incorporate it into your broader machine learning pipeline.

Feel free to explore the provided functionalities and adapt the module to your project requirements. The detailed comments within the code offer further insights into the inner workings of each class and method.

**Model Trainer Module:**

**Regressor Class:**

The Regressor class is designed to facilitate regression tasks using various machine learning models. It provides a streamlined process for training, tuning hyperparameters, and evaluating the performance of different regression algorithms on a given dataset.

Available Regression Models:

* Linear Regression
* Polynomial Regression
* Lasso Regression
* Ridge Regression
* Elastic Net Regression
* Support Vector Regressor (SVR)
* k-Nearest Neighbors (KNN) Regressor
* Decision Tree Regressor
* Random Forest Regressor

Methods**:**

* run\_all

Runs all regression models and compares their performance.

* linear\_model

Trains a Linear Regression model with hyperparameter tuning.

* polynomial\_model

Trains a Polynomial Regression model with hyperparameter tuning.

* lasso\_model

Trains a Lasso Regression model with hyperparameter tuning.

* ridge\_model

Trains a Ridge Regression model with hyperparameter tuning.

* elastic\_model

Trains an ElasticNet Regression model with hyperparameter tuning.

* svr\_model

Trains a Support Vector Regressor (SVR) model with hyperparameter tuning.

* knn\_regressor\_model

Trains a k-Nearest Neighbors (KNN) Regressor model with hyperparameter tuning.

* tree\_regressor\_model

Trains a Decision Tree Regressor model with hyperparameter tuning.

* forest\_regressor\_model

Trains a Random Forest Regressor model with hyperparameter tuning.

* run\_regressor

Performs hyperparameter tuning, training, and evaluation for a given regression model.

* compare\_regressor

Compares the performance of different regressors and identifies the best-performing regressor.

* save\_regressor

Saves the recommended regression model if desired by the user.

Example Usage:

* # Instantiate Regressor class

“regressor\_instance = Regressor(data, target\_column, file\_path)”

* # Run all regression models

“regressor\_instance.run\_all()”

* # Compare and get information about the best-performing regressor

“best\_regressor\_info = regressor\_instance.compare\_regressor()”

* # Save the recommended model

“regressor\_instance.save\_regressor()”

**Classification Analysis using Multiple Classifiers:**

This Python script provides a comprehensive classification analysis using various classifiers on a given dataset. The Classifier class is designed to handle tasks such as data preprocessing, model training, hyperparameter tuning, and performance evaluation. The script supports the following classifiers:

* Logistic Regression
* k-Nearest Neighbors (KNN)
* Support Vector Classifier (SVC)
* Decision Tree Classifier
* Random Forest Classifier

Requirements:

Ensure you have the following libraries installed:

* pandas
* scikit-learn
* matplotlib
* seaborn

You can install these libraries using the following command:

* “pip install pandas scikit-learn matplotlib seaborn”

Methods:

* \_\_init\_\_(self, data, target\_column, file\_path): Initialize the Classifier class.
* run\_all(): Run multiple classifiers and compare their performance.
* compare\_classifier(): Compare classifiers and identify the best-performing one.
* save\_classifier(): Save the recommended classifier to a file.
* Specific classifier training methods:
  + logistic\_classifier()
  + knn\_classifier()
  + svc\_classifier()
  + tree\_classifier\_model()
  + forest\_classifier\_model()

The script includes several methods within the Classifier class, each serving a specific purpose. Here's a brief overview:

* Initialization:
  + \_\_init\_\_(self, data, target\_column, file\_path): Initialize the Classifier class with provided data, target column, and file path.
* Classification Analysis:
  + run\_all(): Runs multiple classifiers, evaluates their performance, and compares the results.
* Classifier Training Methods:
  + logistic\_classifier(): Train a Logistic Regression Classifier with hyperparameter tuning and evaluation.
  + knn\_classifier(): Train a k-Nearest Neighbors (KNN) Classifier and evaluate its performance.
  + svc\_classifier(): Train a Support Vector Classifier (SVC) and evaluate its performance.
  + tree\_classifier\_model(): Train a Decision Tree Classifier and evaluate its performance.
  + forest\_classifier\_model(): Train a Random Forest Classifier and evaluate its performance.
* Helper Methods:
  + run\_classifier(model, param\_grid, X\_train, y\_train, X\_test, y\_test, cv=10): Train a classifier using grid search for hyperparameter tuning and evaluate its performance.
  + compare\_classifier(): Compare and identify the best-performing classifier based on accuracy.
  + save\_classifier(): Save the recommended classifier to a file.

**Example Usage:**

**“import pandas as pd”**

**# Load your dataset:**

**“your\_data = pd.read\_csv("your\_dataset.csv")”**

**# Specify the target column(s):**

**“target\_column = "target"”**

**# Instantiate Classifier:**

**“your\_classifier = Classifier(your\_data, target\_column, "your\_dataset.csv")”**

**# Run classification analysis:**

**“your\_classifier.run\_all()”**

**# Save the recommended classifier:**

**“your\_classifier.save\_classifier()”**