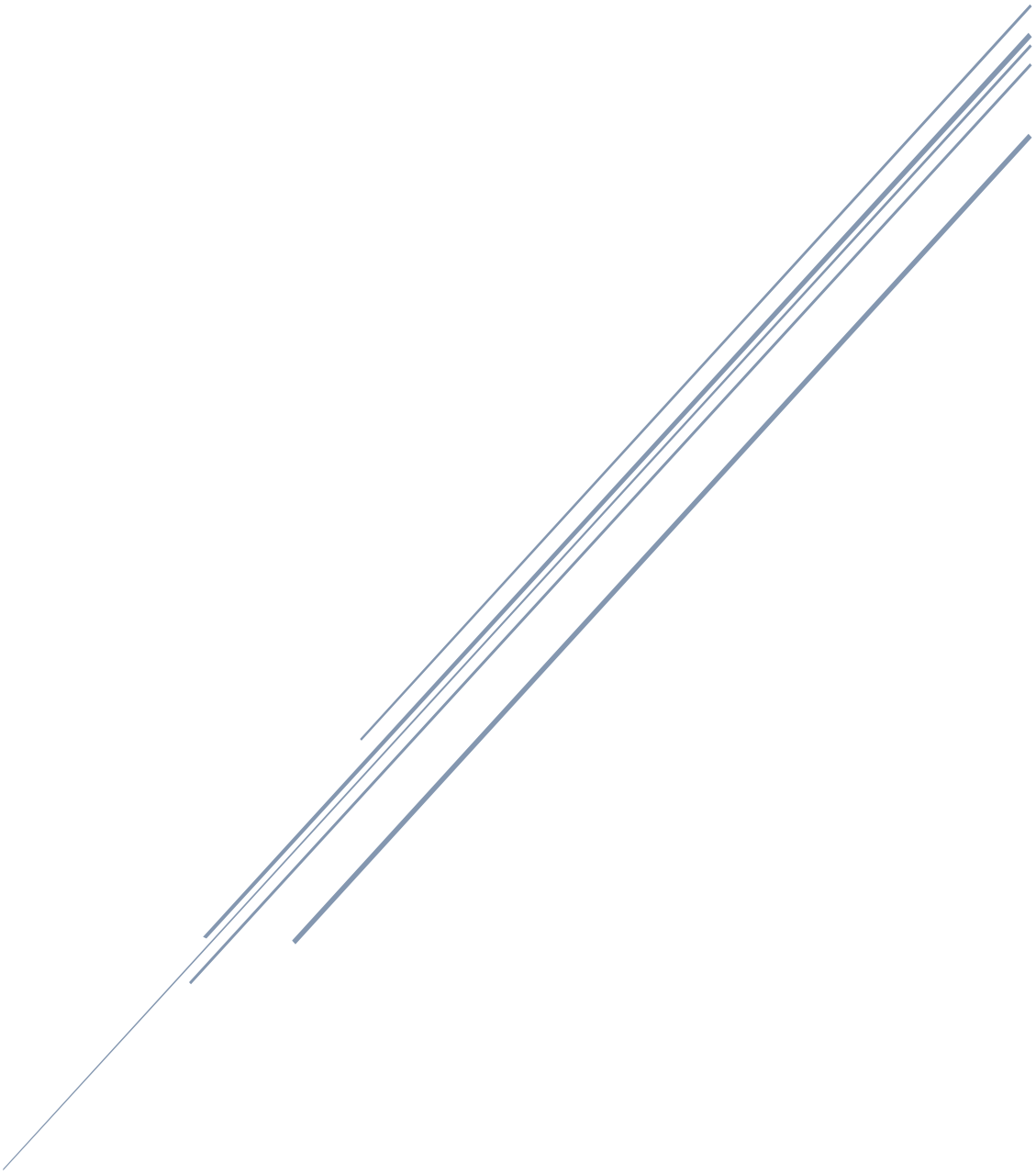
mysticML

A Python Library for Automated Preprocessing.



Christ (Deemed to be university)
M.Sc. AI and ML

II Trimester M.Sc. (AI & ML)

Advanced Machine Learning

Department of Computer Science

mysticML

by

Abhinav Bammidi (2348509)
Shiladitya Sarkar (2348556)
Swastik Banerjee (2348566)

January 2024

CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE · INDIA

CERTIFICATE

This is to certify that the report titled ***mysticML*** *is a bona fide record of work done by* ***Abhinav Bammidi (2348509), Shiladitya Sarkar (2348556),*** *and* ***Swastik Banerjee (2349566)*** *of CHRIST (Deemed to be University), Bangalore, in partial fulfilment of the requirements of II Trimester of M.Sc. Artificial Intelligence and Machine Learning during the year 2023-24.*

Valued-by (Evaluator Name and Signature) **Course Teacher**

1.

2.

Date of Exam:

Table of Contents

Team Details

Reg. no	Name	Summary of tasks performed
2348509	Abhinav Bammidi	Handled the operations of Duplicate, Controller class, Functional Testing, Comparative analysis, Documentation
2348556	Shiladitya Sarkar	Developed novel algorithm for Encoding, Feature Selection, Feature Extraction, Sampling, Hover functionality, Integration Testing
2348566	Swastik Banerjee	Automated functionalities: Outlier, Imputation, Transformation; Performed: Unit Testing, Performance Testing & Version Controlling

Abstract

mysticML, a Python library meticulously crafted for machine learning workflows, presents a comprehensive solution to streamline the intricate pre-processing pipeline. Designed with a focus on simplicity and efficiency, the library introduces automated classes for deduplication, outlier removal, missing data imputation, feature transformation, selection and extraction, and sampling. The inherent intelligence of the algorithms goes beyond mere automation, actively detecting when and to what extent a change is warranted.

Noteworthy is the user-centric design, offering both automation and flexibility. Users can opt to skip or force include specific steps in the pipeline, tailoring the data preparation process to their exact needs. Default processing, executed when no user choice is provided, ensures adaptability based on algorithmic insights. To optimize results, users are encouraged to make informed choices, only intervening when confident in their decisions. The abstract features functionalities executed selectively, ensuring a dynamic and efficient approach that aligns with relevant scenarios. Each functionality is briefly outlined, enhancing user understanding when applied in context.

Introduction

mysticML – 0.5.6

Overview

mysticML is a Python library designed to streamline the pre-processing pipeline involved in machine learning workflows. It aims to simplify the data preparation process by providing classes that automate the tasks of deduplication, outlier removal, missing data imputation, feature – transformation, selection and extraction, and sampling. The algorithms designed are smart enough to not only detect when a change is required but also how much of it is necessary.

Besides the automation, enough flexibility has been provided.

What does it mean for the user? A user can choose to skip or force include (we will see when this is relevant) any atomic step in the entire pipeline as they desire. If no choice is provided the processing will be done entirely as the algorithm seems fit. To get the best results, it is advised to pass choices only when the user is absolutely sure about what they are doing.
Note that the features listed below are functionalities that will only be executed in relevant scenarios. It is briefly explained what each does when needed.

Functional Description

- **Duplicates**: To remove duplicate data and sends it downstream for further processing.
- **Outliers**: Focuses on identifying and handling outliers. Data integrity is maintained at this level without eliminating expected variations.
- **Imputation**: Handles missing data by automatically determining appropriate imputation strategies, ensuring dataset completeness.
- **Transformation**: Streamlines data transformation processes, ensuring the dataset is appropriately modified for improved model performance at a future stage. The transformation strategy to be applied is also smartly decided by this class.
- **Encoding**: Identifies categorical variables and encodes them by a new novel encoding technique that works seamlessly in all conditions. It satisfies two constraints simultaneously – non ordinal attribute should not be ordered post encoding (problem with LabelEncoder) and dimensions should not increase (problem with OneHotEncoding).
- **Feature Selection**: At this level, advanced processing techniques start. Dataset optimization is done by selectively retaining only the most useful features.
- **Feature Extraction**: Features are extracted and combined in the best possible way to represent the original features in still lower dimension. These new features are called principal components, due to their nature.
- **Sampling**: Addresses class imbalance, contributing to a balanced representation of the target class in the dataset. It decides when to upsample, downsample and not sample.

Data Preprocessing and Exploration

Data Understanding and Exploration

In the preliminary phase of Data Understanding and Exploration, our approach is characterized by a systematic evaluation to determine the automated applicability of specific functionalities on the user-provided dataset. Employing custom-defined check conditions, meticulously tailored for this purpose, we scrutinize the dataset's inherent characteristics. This automated scrutiny seeks to discern whether a given functionality can be applied judiciously. Upon validation, automated operations are triggered to seamlessly execute the necessary enhancements on the dataset. The comprehensive results of this automated assessment, along with the corresponding actions taken or abstentions, are systematically documented in the subsequent report. In instances where a functionality is deemed unsuitable for automated application on the provided dataset, it is conscientiously omitted, reflecting a discerning and tailored approach to data manipulation. This meticulous and automated process ensures that each operation aligns with the intrinsic properties of the dataset, contributing to a nuanced and well-informed data exploration endeavour.

Data Cleaning and Handling Missing Values

In the pivotal phase of Data Cleaning and Handling Missing Values, addressing the absence of data, often synonymous with cleaning the dataset, emerges as a cornerstone in the data preprocessing pipeline. Recognizing the paramount significance of this process, our approach integrates a specialized functionality encapsulated within a dedicated class. This class operates systematically to automate the handling of missing values throughout the entire dataset. Leveraging automated routines, this functionality meticulously addresses missing values through strategic measures, either by suitable replacement or by judiciously dropping dimensions.

The automation embedded within this class streamlines and expedites the process of mitigating missing values, a task critical for ensuring the integrity and reliability of the dataset. Through automated replacement with appropriate measures or automated dimension dropping, the functionality offers a nuanced and efficient solution, aligning with the intricate requirements of diverse datasets. The outcomes of this automated data cleaning process are methodically documented, contributing to a comprehensive and transparent data preprocessing pipeline.

Data Integration

In the sphere of Data Integration, our library adopts a systematic and automated approach through custom-defined workflows. These workflows are strategically designed to prioritize foundational preprocessing functionalities, ensuring that basic dataset requirements are addressed before delving into more advanced concepts. Automation is seamlessly embedded within each workflow, expediting the execution of fundamental tasks and facilitating a coherent transition to advanced data preprocessing techniques.

The hallmark of our approach lies in the meticulous orchestration of these workflows. Through automated routines, the library ensures the efficient integration of disparate datasets while maintaining a structured progression from basic to advanced preprocessing. This systematic and automated integration process not only enhances the efficiency of dataset amalgamation but also underscores the library's commitment to a well-organized and streamlined approach in the realm of data integration.

Algorithm Implementation

Functionalities

- **Duplicate Handling:** The Duplicate class is meticulously crafted to oversee and manage redundant data entries within a given dataset. It systematically scrutinizes the dataset, ensuring the identification and removal of duplicate records. This meticulous process is imperative to uphold the integrity and precision of dataset analyses, especially preceding any sampling operations.
- **Outlier Identification:** The Outlier class specializes in the identification and management of outliers inherent in a dataset. Employing adept statistical techniques, it discerns and excludes instances that fall beyond our custom statistical range. This class plays a pivotal role in cultivating a robust and dependable dataset, laying a solid foundation for subsequent analyses.
- **Data Imputation:** The Impute class assumes control over addressing missing data intricacies within a dataset. Through a comprehensive evaluation, it scrutinizes the dataset for any instances of missing values. Guided by custom criteria, it judiciously decides whether to execute imputation or opt for dimension removal. Additionally, the class specifies the imputation strategy, ensuring the preservation of data completeness and integrity.
- **Data Transformation:** The Transform class takes on the responsibility of assessing the suitability of applying transformations to a given dataset. It intelligently discerns the necessity for specific transformations based on dataset characteristics and selects the most appropriate techniques. This class is pivotal in elevating the overall quality and utility of the dataset through thoughtful transformation approaches.
- **Data Encoding:** The Encode class is exclusively devoted to the management of data encoding, with a keen focus on both nominal and ordinal columns. It adeptly makes decisions regarding the most fitting encoding method for each variable, considering the variable's category count. Notably, this class implements a novel approach to handle both types of categorical variables, preserving meaningful representation.
- **Feature Selection:** The Feature Selection class meticulously evaluates the dataset to ascertain the need for feature selection, a crucial dimensionality reduction technique. By reducing time complexity, it anticipates enhancing the efficiency of algorithms in subsequent stages, solidifying its role as a cornerstone in the comprehensive data preprocessing pipeline.
- **Feature Extraction:** The Feature Extraction class specializes in unsupervised dimensionality reduction, intelligently determining the applicability of reduction techniques. It adeptly decides the optimal number of dimensions for transforming the data, playing a pivotal role in refining data representation for heightened effectiveness.
- **Sampling Techniques:** The Sample class tackles data imbalances by executing suitable sampling techniques. It meticulously checks for imbalances in the target variable and applies necessary up-sampling or down-sampling measures. Running this class before the duplicate class ensures a thorough and cohesive data preprocessing pipeline.

In summary, our library's specialized classes offer a meticulous and automated approach to data preprocessing. From Duplicate Handling to Sampling Techniques, each class addresses specific dataset nuances, ensuring a seamless integration of fundamental and advanced preprocessing steps. This structured approach enhances efficiency while upholding data integrity, making our library a robust tool for precise and confident data manipulation.

Library Guide

Installation

```
#Installing the library
%pip install mysticML
#upgrading the library
%pip install --upgrade mysticML
```

Working

This library offers versatile functionality through two distinct approaches: ***one that involves specifying various parameters and another that operates without them.***

Additionally, it provides a range of data preprocessing options, allowing users to choose from ***basic***, ***intermediate***, *or* ***advanced*** levels to tailor their data manipulation to specific needs. This flexibility empowers users to seamlessly integrate the library into their workflows while accommodating varied levels of data complexity.

What preprocessing components are encompassed within each tier of the combination?

- **Basic** – duplicate, outlier, impute
- **Intermediate** – duplicate, outlier, impute, transform, encode
- **Advanced** – duplicate, outlier, impute, transform, encode, feature_sel, feature_ext, sampling

```
import mysticML
from mysticML import Preprocess

preprocess = Preprocess(df) #where df contains the target dataframe
                            #type pandas.DataFrame

preprocess.fit(target = str,
               combo = str,
               duplicate = bool,
               outlier = bool,
               impute = bool,
               transform = bool,
               encode = bool,
               feature_sel = bool,
               feature_ext = bool,
               sampling = bool) -> pandas.DataFrame
```

- **target:** Accepts a string input; when not explicitly defined, assumes the last column of the input dataframe as the target. Alternatively, users can specify the target column explicitly.
- **combo:** Accepts a string input, limited to custom options ***(basic, intermediate, advanced)***, representing different levels of manipulation. Each level entails a custom set of procedures for execution on the input dataframe. If the user does not explicitly specify a combo, the default applied is the ***"advanced"*** level.

- **duplicate:** Takes a boolean value to determine the execution of the duplicate removal class.
- **outlier:** Takes a boolean value to specify the inclusion or exclusion of the outlier removal class.
- **impute:** Accepts a boolean input to determine whether the imputation class should be executed.
- **transform:** Accepts a boolean value, indicating the execution status of the transformation class.
- **encode:** Requires a boolean input to determine the execution of the encoding class.
- **feature_sel:** Accepts a boolean value to dictate the execution status of the feature selection class.
- **feature_ext:** Takes a boolean input to specify the inclusion or exclusion of the feature extraction class.
- **sampling:** Requires a boolean input to determine the execution of the sampling class.

Efficient Coding and Algorithm Execution

Efficiency in coding and algorithm execution is a hallmark of our library, evident in the virtually instantaneous preprocessing of dataframes. The meticulous application of proper code indentation not only facilitates bug identification but also lays the foundation for scalable functionalities within each class. The modular structure meticulously adhered to throughout the developmental phase ensures a seamless user experience, allowing for easy package importation. This approach underscores our commitment to providing a sophisticated and user-friendly framework, where precision in execution and accessibility are paramount considerations.

Model Evaluation and Performance Analysis

Evaluation Metrics and Performance Assessment

In this section, we assess the effectiveness of our data preprocessing library using popular datasets: Iris, Customer Churn and Breast Cancer for Classification and California Housing for regression. This analysis aims to uncover insights into the library's adaptability, efficiency, and impact across diverse machine learning scenarios. Join us as we delve into the outcomes of our library's application on these benchmark datasets, offering a concise glimpse into its performance and reliability.

Assessment on Iris Dataset

```
# Import necessary libraries
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset
iris = datasets.load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target

# Split the dataset into features (X) and target variable (y)
X = iris_df.drop('target', axis=1)
y = iris_df['target']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=2)

# # Decision Tree
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy * 100)
print("Decision Tree Classification Report:\n", classification_report(y_test,
dt_predictions))
print("\n" + "="*50 + "\n")
```

```
Decision Tree Accuracy: 93.33333333333333
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.88      0.88      0.88         8
           2       0.88      0.88      0.88         8

    accuracy                           0.93        30
   macro avg       0.92      0.92      0.92        30
weighted avg       0.93      0.93      0.93        30
```

```
iris_df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
from mysticML import Preprocess as pp
ob = pp(iris_df)
df = ob.fit()
df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	0.173913	0.833333	0.039216	0.045455	0
1	0.086957	0.416667	0.039216	0.045455	0
2	0.000000	0.583333	0.019608	0.045455	0
3	0.456522	0.500000	0.058824	0.045455	0
4	0.130435	0.916667	0.039216	0.045455	0

```
ob.report_
```

```
['Duplicate', 'Outlier', 'Imputation', 'Transformation']
```

```
# Split the dataset into features (X) and target variable (y)
X = df.drop('target', axis=1)
y = df['target']
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=33)
# Decision Tree
```

```
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy * 100)
print("Decision Tree Classification Report:\n", classification_report(y_test,
dt_predictions))
```

```
Decision Tree Accuracy: 96.66666666666667
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      0.90      0.95        10
           2       0.89      1.00      0.94         8

    accuracy                           0.97        30
   macro avg       0.96      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30
```

Assessment on Customer Churn Dataset

```
# Import necessary libraries
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the Customer Churn dataset
data = pd.read_csv("/content/CustomerChurn.csv")

# Split the dataset into features (X) and Exited variable (y)
X = data.drop(columns = ['Exited','Surname','Geography','Gender'], axis=1)
y = data['Exited']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Decision Tree
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy * 100)
print("Decision Tree Classification Report:\n", classification_report(y_test,
dt_predictions))
```

```
print("\n" + "="*50 + "\n")

# K-Nearest Neighbors (KNN)
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, y_train)
knn_predictions = knn_classifier.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
print("K-Nearest Neighbors Accuracy:", knn_accuracy * 100)
print("K-Nearest Neighbors Classification Report:\n",
classification_report(y_test, knn_predictions))
```

```
Decision Tree Accuracy: 78.54394522374042
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.86      0.86     26033
           1       0.49      0.51      0.50      6974

    accuracy                           0.79     33007
   macro avg       0.68      0.68      0.68     33007
weighted avg       0.79      0.79      0.79     33007


==================================================

K-Nearest Neighbors Accuracy: 72.72093798285212
K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.88      0.84     26033
           1       0.25      0.15      0.19      6974

    accuracy                           0.73     33007
   macro avg       0.52      0.51      0.51     33007
weighted avg       0.68      0.73      0.70     33007
```

```
from mysticML import Preprocess as pp
ob = pp(data)
df = ob.fit()
df
```

	Geography	Gender	Age	Balance	NumOfProducts	IsActiveMember	Exited
0	0.429118	0.435571	0.307692	0.000000	1.0	0.0	0
1	0.429118	0.435571	0.307692	0.000000	1.0	1.0	0
2	0.429118	0.435571	0.576923	0.000000	1.0	0.0	0
3	0.429118	0.435571	0.346154	0.994869	0.0	1.0	0
4	0.780572	0.435571	0.307692	0.000000	1.0	1.0	0

```
df.shape
```

```
(165034, 7)
```

```
ob.report_
```

```
['Outlier', 'Imputation', 'Transformation', 'Encoding', 'FeatureSel']
```

```
# Import necessary libraries
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Split the dataset into features (X) and Exited variable (y)
X = df.drop('Exited', axis=1)
y = df['Exited']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Decision Tree
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy * 100)
print("Decision Tree Classification Report:\n", classification_report(y_test,
dt_predictions))
print("\n" + "="*50 + "\n")

# K-Nearest Neighbors (KNN)
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, y_train)
knn_predictions = knn_classifier.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
print("K-Nearest Neighbors Accuracy:", knn_accuracy * 100)
print("K-Nearest Neighbors Classification Report:\n",
classification_report(y_test, knn_predictions))
```

```
Decision Tree Accuracy: 80.84951676917017
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.90      0.88     26009
           1       0.56      0.47      0.51      6998

    accuracy                           0.81     33007
   macro avg       0.71      0.69      0.70     33007
weighted avg       0.80      0.81      0.80     33007


==================================================

K-Nearest Neighbors Accuracy: 81.19489805192838
K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.91      0.88     26009
           1       0.57      0.46      0.51      6998

    accuracy                           0.81     33007
   macro avg       0.72      0.68      0.70     33007
weighted avg       0.80      0.81      0.80     33007
```

Assessment on California Housing Dataset

```python
# Import necessary libraries
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load the California Housing dataset
california_housing = fetch_california_housing()
california_df = pd.DataFrame(data=california_housing.data,
columns=california_housing.feature_names)
california_df['target'] = california_housing.target

# Split the dataset into features (X) and target variable (y)
X = california_df.drop('target', axis=1)
y = california_df['target']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# K-Nearest Neighbors Regressor (KNN)
```

```
knn_regressor = KNeighborsRegressor(n_neighbors=3)
knn_regressor.fit(X_train, y_train)
knn_predictions = knn_regressor.predict(X_test)
knn_mse = mean_squared_error(y_test, knn_predictions)
knn_r2 = r2_score(y_test, knn_predictions)
print("KNN Mean Squared Error:", knn_mse)
print("KNN R-squared:", knn_r2)
```

```
KNN Mean Squared Error: 1.1694144088518572
KNN R-squared: 0.10759585116572867
```

```
california_df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	target
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

```
california_df.shape
```

```
(20640, 9)
```

```
from mysticML import Preprocess as pp
ob = pp(california_df)
df = ob.fit()
df
```

	MedInc	AveRooms	AveBedrms	AveOccup	target
0	0.417505	0.708688	0.426225	0.104632	4.526
1	0.417505	0.607716	0.388022	0.079602	3.585
2	0.417505	0.885190	0.462742	0.118485	3.521
3	0.811071	0.550761	0.462457	0.104204	3.413
4	0.495709	0.613633	0.468358	0.083625	3.422

```
# Split the dataset into features (X) and target variable (y)
X = df.drop('target', axis=1)
y = df['target']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# K-Nearest Neighbors Regressor (KNN)
knn_regressor = KNeighborsRegressor(n_neighbors=3)
knn_regressor.fit(X_train, y_train)
```

```
knn_predictions = knn_regressor.predict(X_test)
knn_mse = mean_squared_error(y_test, knn_predictions)
knn_r2 = r2_score(y_test, knn_predictions)
print("KNN Mean Squared Error:", knn_mse)
print("KNN R-squared:", knn_r2)
```

```
KNN Mean Squared Error: 0.6524791315631397
KNN R-squared: 0.5132475245254584
```

Assessment on Breast Cancer Dataset

```
# Import necessary libraries
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the Breast Cancer dataset
breast_cancer = datasets.load_breast_cancer()
breast_cancer_df = pd.DataFrame(data=breast_cancer.data,
columns=breast_cancer.feature_names)
breast_cancer_df['target'] = breast_cancer.target

# Split the dataset into features (X) and target variable (y)
X = breast_cancer_df.drop('target', axis=1)
y = breast_cancer_df['target']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=33)

# Decision Tree
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy * 100)
print("Decision Tree Classification Report:\n", classification_report(y_test,
dt_predictions))
print("\n" + "="*50 + "\n")

# Random Forest
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train, y_train)
```

```
rf_predictions = rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Random Forest Accuracy:", rf_accuracy * 100)
print("Random Forest Classification Report:\n", classification_report(y_test,
rf_predictions))
print("\n" + "="*50 + "\n")

# K-Nearest Neighbors (KNN)
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, y_train)
knn_predictions = knn_classifier.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
print("K-Nearest Neighbors Accuracy:", knn_accuracy * 100)
print("K-Nearest Neighbors Classification Report:\n",
classification_report(y_test, knn_predictions))
```

```
Decision Tree Accuracy: 88.59649122807018
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.81      0.84        43
           1       0.89      0.93      0.91        71

    accuracy                           0.89       114
   macro avg       0.88      0.87      0.88       114
weighted avg       0.89      0.89      0.89       114
```

```
Random Forest Accuracy: 94.73684210526315
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.91      0.93        43
           1       0.95      0.97      0.96        71

    accuracy                           0.95       114
   macro avg       0.95      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114
```

```
K-Nearest Neighbors Accuracy: 90.35087719298247
K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.84      0.87        43
           1       0.91      0.94      0.92        71

    accuracy                           0.90       114
   macro avg       0.90      0.89      0.90       114
weighted avg       0.90      0.90      0.90       114
```

```
breast_cancer_df.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	sr
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	

5 rows × 31 columns

```
breast_cancer_df.shape
```

```
(569, 31)
```

```
from mysticML import Preprocess as pp
ob = pp(breast_cancer_df)
df_d = ob.fit()
#duplicate=True,outlier=False,impute=False,transform=False,encode=False,feature_sel=False,feature_ext=False,sampling=False)
df_d
```

	pca_comp_0	pca_comp_1	pca_comp_2	pca_comp_3	pca_comp_4	pca_comp_5	pca_comp_6	pca_comp_7	target
0	0.330004	0.109432	0.470400	-0.200473	-0.162174	-0.354981	-0.079325	0.083929	0
1	1.058200	0.311806	0.568610	0.520587	0.576215	0.219962	0.142256	0.145219	0
2	0.040170	0.176374	0.345475	-0.372276	0.029025	-0.111407	-0.044940	-0.487222	0
3	1.381492	0.072388	0.693751	-0.187157	0.642197	-0.224894	0.142839	-0.083923	0
4	0.545700	0.652586	-0.137374	-0.286400	0.187988	-0.069430	-0.011071	0.183589	0
...	...	...	...	...	...	...	...	...	...
554	0.352397	-0.291432	-0.222964	-0.311465	-0.015002	0.037643	-0.046490	-0.384390	0
555	-0.057484	0.194895	-0.115482	0.164224	0.033979	0.012273	-0.029371	0.063116	0
556	1.084944	-0.024691	0.799136	-0.152025	0.406090	-0.311661	0.062338	-0.096386	0
557	0.795152	-0.221059	-0.044826	0.168178	-0.060676	-0.008699	-0.020409	-0.026792	0
558	-0.902689	-0.144288	0.301044	-0.028977	-0.301109	-0.128349	-0.041200	0.187947	1

559 rows × 9 columns

```
ob.report_
```

```
['Outlier', 'Imputation', 'Transformation', 'FeatureSel', 'FeatureExt']
```

```
# Import necessary libraries
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
# Split the dataset into features (X) and target variable (y)
X = df_d.drop('target', axis=1)
y = df_d['target']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=33)

# Decision Tree
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy * 100)
print("Decision Tree Classification Report:\n", classification_report(y_test,
dt_predictions))
print("\n" + "="*50 + "\n")

# Random Forest
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train, y_train)
rf_predictions = rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Random Forest Accuracy:", rf_accuracy * 100)
print("Random Forest Classification Report:\n", classification_report(y_test,
rf_predictions))
print("\n" + "="*50 + "\n")

# K-Nearest Neighbors (KNN)
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, y_train)
knn_predictions = knn_classifier.predict(X_test)
knn_accuracy = accuracy_score(y_test, knn_predictions)
print("K-Nearest Neighbors Accuracy:", knn_accuracy * 100)
print("K-Nearest Neighbors Classification Report:\n",
classification_report(y_test, knn_predictions))
```

```
Decision Tree Accuracy: 94.64285714285714
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.91      0.93        43
           1       0.94      0.97      0.96        69

    accuracy                           0.95       112
   macro avg       0.95      0.94      0.94       112
weighted avg       0.95      0.95      0.95       112
```

```
Random Forest Accuracy: 95.53571428571429
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.93      0.94        43
           1       0.96      0.97      0.96        69

    accuracy                           0.96       112
   macro avg       0.95      0.95      0.95       112
weighted avg       0.96      0.96      0.96       112
```

```
K-Nearest Neighbors Accuracy: 91.96428571428571
K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.81      0.89        43
           1       0.89      0.99      0.94        69

    accuracy                           0.92       112
   macro avg       0.93      0.90      0.91       112
weighted avg       0.92      0.92      0.92       112
```

Comparative Analysis

In this comparative analysis, we explore the transformative impact of two distinct Python libraries, mysticML and autoClean, on the performance of machine learning algorithms across diverse datasets. The datasets under consideration encompass a range of applications, including the renowned Iris dataset, Customer Churn data, California Housing data, and Breast Cancer data. The primary focus of this evaluation is to scrutinize the efficacy of these libraries in automating the data preprocessing and cleaning steps, with a keen eye on the subsequent enhancement or detriment to algorithmic accuracy.

Across the Iris dataset, mysticML showcases its prowess by substantially boosting the accuracy of the Decision Tree algorithm, elevating it from 93.3% to an impressive 96.6%. Conversely, autoClean and mdkearns demonstrates a different impact, with their Decision Tree accuracy showing 92.1% and 96.3% respectively. This dichotomy in results is further evident in the Customer Churn dataset, where mysticML propels the Decision Tree accuracy from 78.5% to 80.8%, while both autoClean and mdkearns experiences a decrement to 56.9% and 68.6% respectively for the same dataset and for same parameters. Beyond Decision Trees, the evaluation extends to K-NN algorithms and regression metrics such as K-NN r-squared and Mean Squared Error (MSE) for the California Housing dataset. Through this analysis, we aim to unravel the nuanced contributions of each library, shedding light on their respective strengths and implications for efficient machine learning workflows.

Dataset	**Algorithm**	**Before preprocessing**	**After mysticML**	**After mdkearns**	**After autoClean**
Iris	Decision Tree	93.3	96.6	96.3	92.1
CustomerChurn	Decision Tree	78.5	80.8	68.6	56.9
CustomerChurn	K-NN	72.7	81.1	75.6	70.1
California Housing	K-NN R^2	0.10	0.51	0.23	0.24
California Housing	K-NN MSE	1.16	0.65	0.67	1.89
Breast Cancer	Decision Tree	88.5	94.6	88.0	89.0
Breast Cancer	Random Forest	94.7	95.5	92.1	91.8
Breast Cancer	K-NN	90.3	91.9	90.0	89.0

Limitations

As of now, mysticML faces limitations in its ability to process and analyse diverse types of data, restricting its functionality to numerical and categorical data exclusively. This means that it currently cannot effectively handle non-traditional data forms such as images, audio, video, and other multimodal data commonly encountered in real-world scenarios. However, there are plans to enhance mysticML's capabilities in the future by incorporating support for these diverse data types. This anticipated expansion would significantly broaden the application scope of mysticML, enabling it to tackle more complex and varied datasets, making it a more versatile and comprehensive machine learning tool for a wider range of tasks and industries. This strategic development aligns with the evolving landscape of machine learning, where addressing the challenges posed by different data modalities is crucial for achieving more sophisticated and holistic analytical outcomes.

Future Scope

Rapid alpha stage developments are under process in order to include suggestions, modelling and multiple datatype handling capabilities. From the user's perspective this will mean two functionalities that this library will extend to provide –

- suggestions on what is best for the dataset provided - including what not to do on top of what has been automatically done and which model to use.
- the library will retain the dataset post preprocessing to smartly determine which model should fit the given data the best. In return, a helper method shall produce a completely trained model that is ready for download and use.
- The user will be able to feed in data of varied types (images, audio, video, etc.)

~team mysticML

References

- numpy library - https://numpy.org/doc/
- pandas library - https://pandas.pydata.org/docs/
- scikit-learn library - https://scikit-learn.org/
- math library - https://docs.python.org/3/library/math.html
- scipy library - https://scipy.org/
- Shapiro - https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.shapiro.html
- KS-Test - https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kstest.html
- Z-Score - https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.zscore.html