جامعة عين شمس

كلية الهندسة

DATA MINING PROJECT

JANUARY 1, 2020

AIN SHAMS UNIVERSITY – FACULTY OF ENGINEERING

FACULTY OF ENGINEERING

AIN SHAMS UNIVERSITY

CSE 385: Data Mining & Business Intelligence

Data Mining Project

Submitted by:

Abdelrahman Amr Issawi	aid-issawi@hotmail.com	16P6001@eng.asu.edu.eg
Ahmed Hesham AlSaey	ahmed.alsai@hotmail.com	16P6000@eng.asu.edu.eg
Sherif Ashraf Ahmed	sherifashraf_98@hotmail.com	16P9033@eng.asu.edu.eg
Mayar Khaled Mohamed	MayarKhaledd_3@yahoo.com	16P6030@eng.asu.edu.eg

Submitted to:

Dr. Mahmoud Mounir,

Eng. Ahmed.

JANUARY 1, 2020

A report for Data Mining & Business Intelligence Course codded CSE385 with the requirements of Ain Shams University.

Table of Contents

1.0 INTRODUCTION

In this project we are applying the algorithms we studied. Regression, classification and clustering. Also, we are pre-processing the data before we apply any of the algorithms, and finally we get the results of applying these tools.

Regression and classification are supervised learning approaches that map an input to an output based on example input-output pairs, while clustering is an unsupervised learning approach.

Regression: It predicts continuous valued output. Regression analysis is the statistical model which is used to predict the numeric data instead of labels. It can also identify the distribution trends based on the available data or historic data. Predicting a person's income from their age, education is example of regression task.

Classification: It predicts discrete number of values. In classification the data is categorized under different labels according to some parameters and then the labels are predicted for the data. Classifying emails as either spam or not spam is example of classification problem.

Clustering: Clustering is the task of partitioning the dataset into groups, called clusters. The goal is to split up the data in such a way that data sets within single cluster are very similar and data sets in different clusters are different. It determines grouping among unlabeled data.

Classification is applied on the cancer dataset, Regression is applied on diamonds dataset, and Clustering is applied on the iris dataset.

In the report attached we are discussing our implementation, the added features, you will also find a user guide, and some test cases. At the end you will find our codes.

Anyway, I hope this quick introduction has helped you, Now let's read the report.

2.0 BRIEF DESCRIPTION

Implementation

There are 5 python files Main.py, Preprocessing.py, regression.py, classification.py, clustering.py. Main.py is the main function calling the other functions. The main function reads the data files needed for classification, regression, and clustering as well.

For classification and regression, the main function calls the **dataCleaning()** function defined in pre-processing class it splits the data set into training and testing data, and drops the unneeded columns. Then, the 2 classes are sent to the **encode()** function giving values to any un numerical data. Then the missing values dealing method is specified, whether to **drop()** the tuple, **replaceMean()**, or **replaceMode()**. The data set is **split()** into X_train_class, X_test_class, y_train_class, y_test_class. And finally the data is scaled using **scale()** using StandardScaler, MinMaxScaler, MaxScaler, or RobustScaler.

Then the data is ready for classification or regression, for classification the function **classify()** is called taking the chosen algorithm to classify with, X_train_class, y_train_class to train the machine, then we classify the X_test_class and gives the results to **calculateAccuracy()** calculateAccuracy to compare it with the y_test_class.

The same steps are done in the regression as well but we use the function **predict()**, and **getScore()**.

For clustering, we load the dataset to X_clust, y_clust, then we use the **splitCluster()** function to break the dataset into training and testing data. We **scale()** the data, and finally we **predict()** the clusters.

Implemented classifying methods:

- KNN,
- Decision Tree,
- Naive Bayes,
- Random Forest.

Implemented regression methods:

- Linear Regression,
- Decision Tree,
- Polynomial Regression,
- KNN Regression,
- Random Forest.

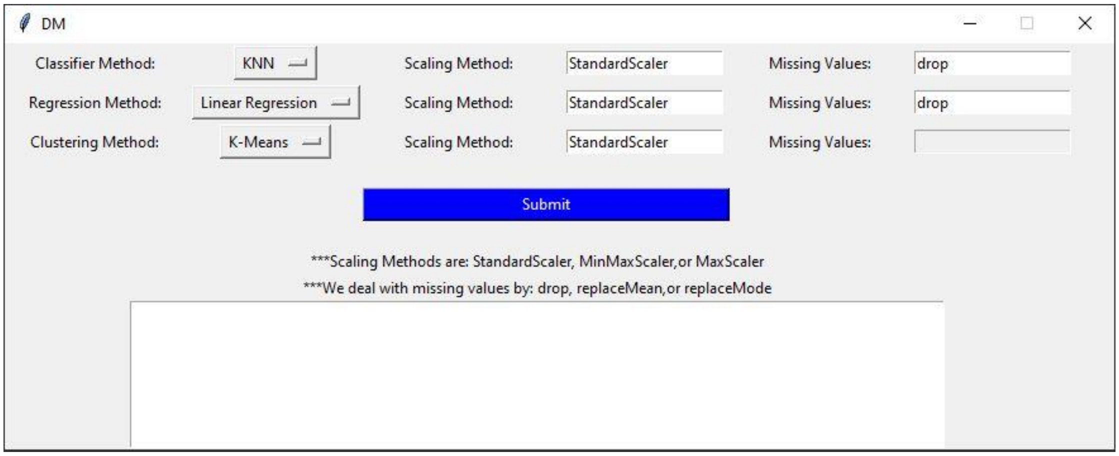
Implemented clustering methods:

- K-means.

Added Features “Bonus”

- Extra regression algorithm is added “Random Forest”

- GUI is implemented so that it’s easier to choose the algorithm you need to apply, also you can write multiple missing values dealing methods, and multiple scaling methods. The score, accuracy, and clusters are displayed in it.
  The GUI is implemented using tkinter library in python, it’s defined to use the labels, buttons, entry fields, and list-box.



DM

Classifier Method: KNN Scaling Method: StandardScaler Missing Values: drop

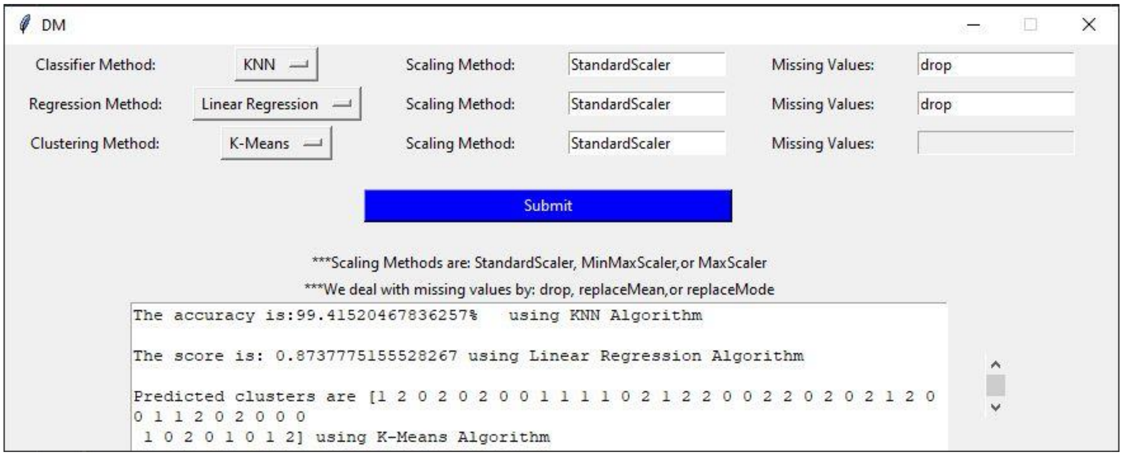
Regression Method: Linear Regression Scaling Method: StandardScaler Missing Values: drop

Clustering Method: K-Means Scaling Method: StandardScaler Missing Values:

***Scaling Methods are: StandardScaler, MinMaxScaler,or MaxScaler

***We deal with missing values by: drop, replaceMean,or replaceMode

- High classifying accuracy.



DM

Classifier Method: KNN Scaling Method: StandardScaler Missing Values: drop

Regression Method: Linear Regression Scaling Method: StandardScaler Missing Values: drop

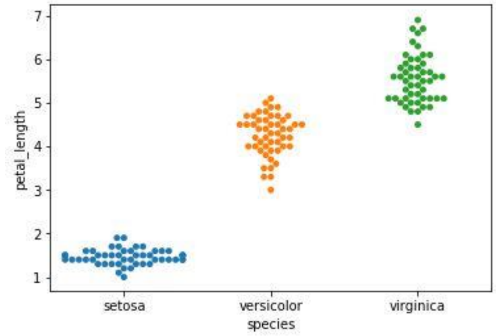
Clustering Method: K-Means Scaling Method: StandardScaler Missing Values:

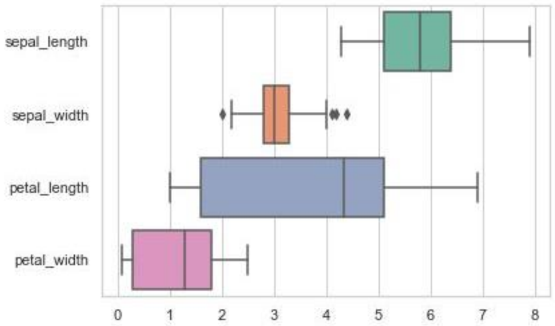
***Scaling Methods are: StandardScaler, MinMaxScaler,or MaxScaler

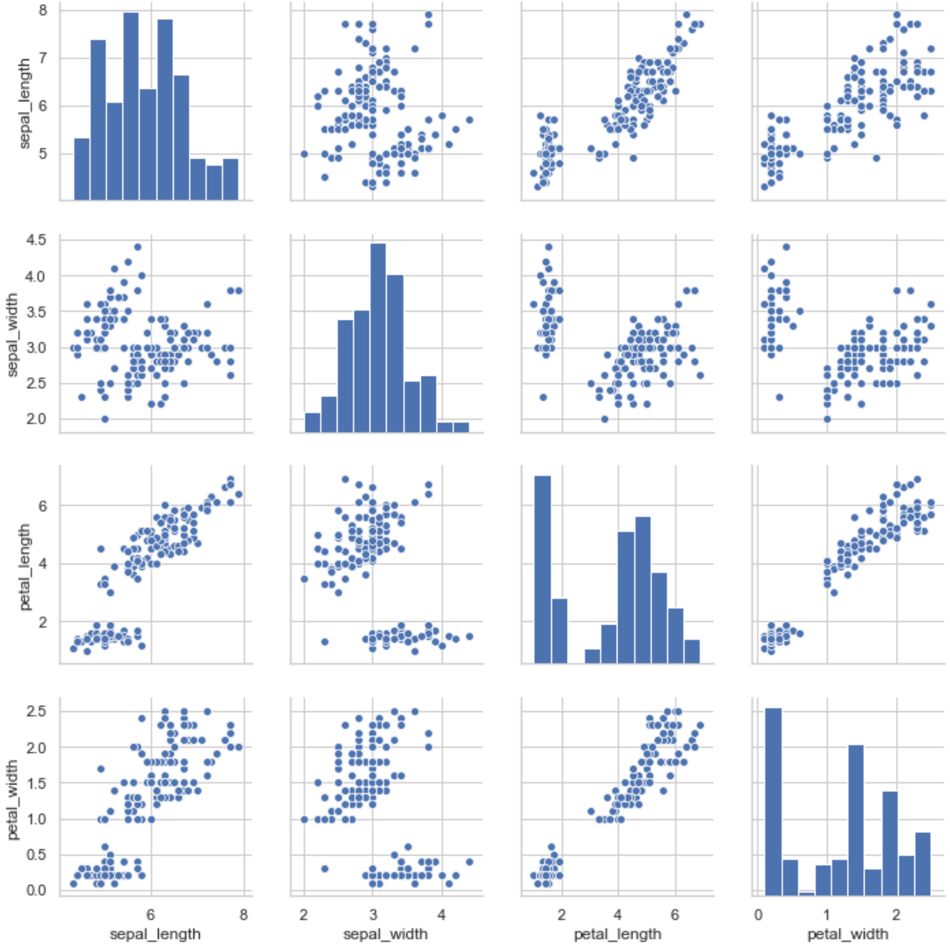
***We deal with missing values by: drop, replaceMean,or replaceMode

```
The accuracy is:99.41520467836257%   using KNN Algorithm

The score is: 0.8737775155528267 using Linear Regression Algorithm

Predicted clusters are [1 2 0 2 0 2 0 0 1 1 1 1 0 2 1 2 2 0 0 2 2 0 2 0 2 1 2 0
0 1 1 2 0 2 0 0 0
 1 0 2 0 1 0 1 2] using K-Means Algorithm
```

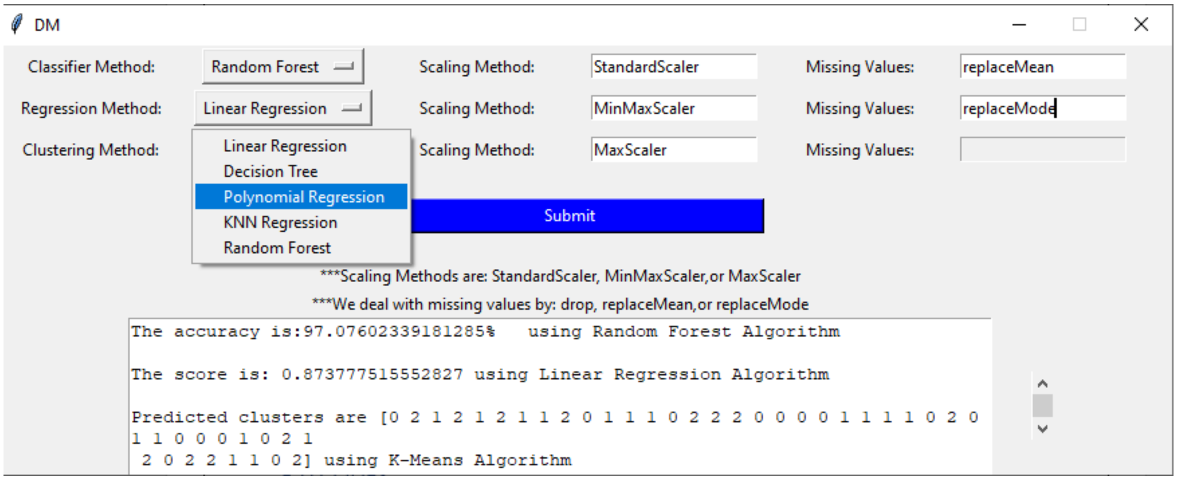
- Clustering visualization.







3.0 USER GUIDE



Classifier Method: Random Forest
Scaling Method: StandardScaler
Missing Values: replaceMean

Regression Method: Linear Regression
Scaling Method: MinMaxScaler
Missing Values: replaceMode

Clustering Method:
Scaling Method: MaxScaler
Missing Values:

***Scaling Methods are: StandardScaler, MinMaxScaler,or MaxScaler

***We deal with missing values by: drop, replaceMean,or replaceMode

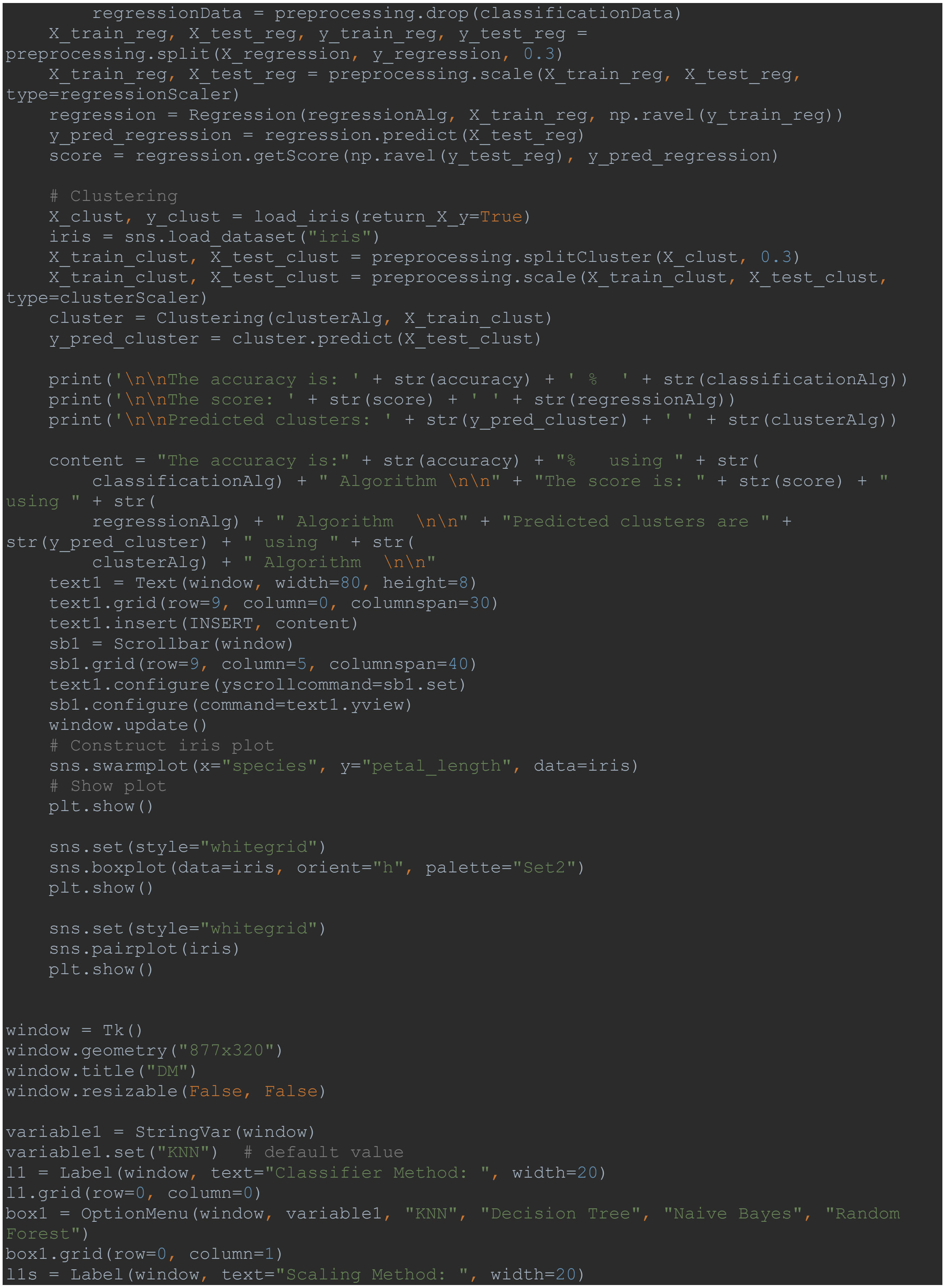
```
The accuracy is:97.07602339181285%   using Random Forest Algorithm

The score is: 0.873777515552827 using Linear Regression Algorithm

Predicted clusters are [0 2 1 2 1 2 1 1 2 0 1 1 1 0 2 2 2 0 0 0 0 1 1 1 1 0 2 0
1 1 0 0 0 1 0 2 1
 2 0 2 2 1 1 0 2] using K-Means Algorithm
```

1. Choose the Algorithm.
2. Choose the scaling method.
3. Choose the missing values dealing method.
4. Click submit.
5. Observe your results.

4.0 CODES

Main.py

```
import pandas as pd
import numpy as np
from preprocessing import Preprocessor
from classification import Classifier
from regression import Regression
from clustering import Clustering
from sklearn.datasets import load_iris
import seaborn as sns
import matplotlib.pyplot as plt
from tkinter import *


def buttonClick():
    classificationAlg = variable1.get()
    regressionAlg = variable2.get()
    clusterAlg = variable3.get()

    classificationScaler = box1s.get()
    regressionScaler = box2s.get()
    clusterScaler = box3s.get()

    classificationMethod = box1F.get()
    regressionMethod = box2F.get()

    preprocessing = Preprocessor()

    # Classification
    classificationData = pd.read_csv('cancer.csv')
    X_class, y_class = preprocessing.dataCleaning(classificationData, 'cancer')
    X_class, y_class = preprocessing.encode(X_class, y_class, 'cancer')
    if classificationMethod == "drop":
        classificationData = preprocessing.drop(classificationData)
    elif classificationMethod == "replaceMean":
        classificationData = preprocessing.replaceMean(classificationData)
    elif classificationMethod == "replaceMode":
        classificationData = preprocessing.replaceMode(classificationData)
    else:
        classificationData = preprocessing.drop(classificationData)
    X_train_class, X_test_class, y_train_class, y_test_class =
preprocessing.split(X_class, y_class, 0.3)
    X_train_class, X_test_class = preprocessing.scale(X_train_class, X_test_class,
type=classificationScaler)
    classifier = Classifier(classificationAlg, X_train_class, np.ravel(y_train_class))
    y_pred_class = classifier.classify(X_test_class)
    accuracy = classifier.calculateAccuracy(np.ravel(y_test_class), y_pred_class)

    # Regression
    regressionData = pd.read_csv('diamonds.csv')
    X_regression, y_regression = preprocessing.dataCleaning(regressionData,
'diamonds')
    X_regression, y_regression = preprocessing.encode(X_regression, y_regression,
'diamonds')
    if regressionMethod == "drop":
        regressionData = preprocessing.drop(regressionData)
    elif regressionMethod == "replaceMean":
        regressionData = preprocessing.replaceMean(regressionData)
    elif regressionMethod == "replaceMode":
        regressionData = preprocessing.replaceMode(regressionData)
    else:
```



```
        regressionData = preprocessing.drop(classificationData)
    X_train_reg, X_test_reg, y_train_reg, y_test_reg =
preprocessing.split(X_regression, y_regression, 0.3)
    X_train_reg, X_test_reg = preprocessing.scale(X_train_reg, X_test_reg,
type=regressionScaler)
    regression = Regression(regressionAlg, X_train_reg, np.ravel(y_train_reg))
    y_pred_regression = regression.predict(X_test_reg)
    score = regression.getScore(np.ravel(y_test_reg), y_pred_regression)

    # Clustering
    X_clust, y_clust = load_iris(return_X_y=True)
    iris = sns.load_dataset("iris")
    X_train_clust, X_test_clust = preprocessing.splitCluster(X_clust, 0.3)
    X_train_clust, X_test_clust = preprocessing.scale(X_train_clust, X_test_clust,
type=clusterScaler)
    cluster = Clustering(clusterAlg, X_train_clust)
    y_pred_cluster = cluster.predict(X_test_clust)

    print('\n\nThe accuracy is: ' + str(accuracy) + ' %  ' + str(classificationAlg))
    print('\n\nThe score: ' + str(score) + ' ' + str(regressionAlg))
    print('\n\nPredicted clusters: ' + str(y_pred_cluster) + ' ' + str(clusterAlg))

    content = "The accuracy is:" + str(accuracy) + "%   using " + str(
        classificationAlg) + " Algorithm \n\n" + "The score is: " + str(score) + "
using " + str(
        regressionAlg) + " Algorithm  \n\n" + "Predicted clusters are " +
str(y_pred_cluster) + " using " + str(
        clusterAlg) + " Algorithm  \n\n"
    text1 = Text(window, width=80, height=8)
    text1.grid(row=9, column=0, columnspan=30)
    text1.insert(INSERT, content)
    sb1 = Scrollbar(window)
    sb1.grid(row=9, column=5, columnspan=40)
    text1.configure(yscrollcommand=sb1.set)
    sb1.configure(command=text1.yview)
    window.update()
    # Construct iris plot
    sns.swarmplot(x="species", y="petal_length", data=iris)
    # Show plot
    plt.show()

    sns.set(style="whitegrid")
    sns.boxplot(data=iris, orient="h", palette="Set2")
    plt.show()

    sns.set(style="whitegrid")
    sns.pairplot(iris)
    plt.show()


window = Tk()
window.geometry("877x320")
window.title("DM")
window.resizable(False, False)

variable1 = StringVar(window)
variable1.set("KNN")  # default value
l1 = Label(window, text="Classifier Method: ", width=20)
l1.grid(row=0, column=0)
box1 = OptionMenu(window, variable1, "KNN", "Decision Tree", "Naive Bayes", "Random
Forest")
box1.grid(row=0, column=1)
l1s = Label(window, text="Scaling Method: ", width=20)
```

```
l1s.grid(row=0, column=2)
box1s = Entry(window)
box1s.grid(row=0, column=3)
box1s.insert(END, "StandardScaler")

l1F = Label(window, text="Missing Values: ", width=20)
l1F.grid(row=0, column=4)
box1F = Entry(window)
box1F.insert(END, "drop")
box1F.grid(row=0, column=5)

variable2 = StringVar(window)
variable2.set("Linear Regression")  # default value
l2 = Label(window, text="Regression Method: ", width=20)
l2.grid(row=1, column=0)
box2 = OptionMenu(window, variable2, "Linear Regression", "Decision Tree", "Polynomial Regression", "KNN Regression",
                  "Random Forest")
box2.grid(row=1, column=1)
l2s = Label(window, text="Scaling Method: ", width=20)
l2s.grid(row=1, column=2)
box2s = Entry(window)
box2s.insert(END, "StandardScaler")
box2s.grid(row=1, column=3)

l2F = Label(window, text="Missing Values: ", width=20)
l2F.grid(row=1, column=4)
box2F = Entry(window)
box2F.insert(END, "drop")
box2F.grid(row=1, column=5)

variable3 = StringVar(window)
variable3.set("K-Means")  # default value

l3 = Label(window, text="Clustering Method: ", width=20)
l3.grid(row=2, column=0)
box3 = OptionMenu(window, variable3, "K-Means")
box3.grid(row=2, column=1)
l3s = Label(window, text="Scaling Method: ", width=20)
l3s.grid(row=2, column=2)
box3s = Entry(window)
box3s.insert(END, "StandardScaler")
box3s.grid(row=2, column=3)

l3F = Label(window, text="Missing Values: ", width=20)
l3F.grid(row=2, column=4)
box3F = Entry(window, state='disabled')
box3F.insert(END, "drop")
box3F.grid(row=2, column=5)

l4 = Label(window, text="")
l4.grid(row=3, column=1, columnspan=2)

b1 = Button(window, text="Submit", width=40, fg="white", bg="blue", command=buttonClick)
b1.grid(row=5, column=2, columnspan=2)

l6 = Label(window, text="")
l6.grid(row=6, column=1, columnspan=2)

l5 = Label(window, text="***Scaling Methods are: StandardScaler, MinMaxScaler,or MaxScaler")
l5.grid(row=7, column=0, columnspan=8)
```

```
l5 = Label(window, text="***We deal with missing values by: drop, replaceMean,or replaceMode")
l5.grid(row=8, column=0, columnspan=8)

text1 = Text(window, width=80, height=7)
text1.grid(row=9, column=0, columnspan=30)

# algorithm_classification = ""
# algorithm_regression = ""
# algorithm_cluster = ""
# classificationScaler = ""
# regressionScaler = ""
# clusterScaler = ""


window.mainloop()
```

Preprocessing.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import LabelEncoder


class Preprocessor:
    def split(self, X, y, test_size):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size,
random_state=58)  # 58
        return X_train, X_test, y_train.astype(np.int64), y_test.astype(np.int64)

    def splitCluster(self, X, test_size):
        X_train, X_test = train_test_split(X, test_size=test_size)
        return X_train, X_test

    def scale(self, X_train, X_test, type):
        if type == "StandardScaler":
            standardScaler = StandardScaler()
            standardScaler.fit(X_train)
            X_train = standardScaler.transform(X_train)
            X_test = standardScaler.transform(X_test)
            return X_train, X_test

        elif type == "MinMaxScaler":
            minMaxScaler = MinMaxScaler()
            minMaxScaler.fit(X_train)
            X_train = minMaxScaler.transform(X_train)
            X_test = minMaxScaler.transform(X_test)
            return X_train, X_test
        elif type == "MaxScaler":

            maxScaler = MaxAbsScaler()
            maxScaler.fit(X_train)
            X_train = maxScaler.transform(X_train)
            X_test = maxScaler.transform(X_test)
            return X_train, X_test

        elif type == "RobustScaler":
            robustScaler = RobustScaler()
            robustScaler.fit(X_train)
            X_train = robustScaler.transform(X_train)
            X_test = robustScaler.transform(X_test)
            return X_train, X_test

    def encode(self, X, y, dataset_name):
        if dataset_name == 'diamonds':
            labelencoder = LabelEncoder()
            X[:, 1] = labelencoder.fit_transform(X[:, 1])
            X[:, 2] = labelencoder.fit_transform(X[:, 2])
            X[:, 3] = labelencoder.fit_transform(X[:, 3])
        elif dataset_name == 'cancer':
            labelencoder = LabelEncoder()
            y[:, 0] = labelencoder.fit_transform(y[:, 0])
        return X, y

    def drop(self, data):
        data_modified = data.dropna(axis=0)
        return data_modified

    def replaceMean(self, data):
        data_modified = data.fillna(data.mean(), inplace=True)
        return data_modified

    def replaceMode(self, data):
        data_modified = data.fillna(data.mode(), inplace=True)
        return data_modified

    def dataCleaning(self, data, name):
        if name == 'cancer':
            data.drop(data.columns[0], axis='columns', inplace=True)
            X = data.iloc[:, 1:].to_numpy()
            y = data.iloc[:, 0].values.reshape(569, 1)

        elif name == 'diamonds':
            data.drop(data.columns[0], axis='columns', inplace=True)
            data_modified = data.drop(labels='price', axis=1)
            X = data_modified.iloc[:, 0:8].to_numpy()
            y = data.iloc[:, 6].values.reshape(53940, 1)
        return X, y
```

regression.py

```
from sklearn import metrics
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import PolynomialFeatures


class Regression:
    def __init__(self, regressor_name, X, y):
        self.regressor_name = regressor_name
        self.X_train = X
        self.y_train = y

    def predict(self, X_test):
        if self.regressor_name == 'Decision Tree':
            decision_tree = DecisionTreeRegressor(random_state=0, max_depth=2)
            decision_tree.fit(self.X_train, self.y_train)
            return decision_tree.predict(X_test)

        elif self.regressor_name == 'Polynomial Regression':
            poly_features = PolynomialFeatures(degree=2)
            X_poly = poly_features.fit_transform(self.X_train)
            X_poly_test = poly_features.transform(X_test)
            poly_model = LinearRegression()
            poly_model.fit(X_poly, self.y_train)
            return poly_model.predict(X_poly_test)

        elif self.regressor_name == 'Linear Regression':
            linear_regressor = LinearRegression()
            linear_regressor.fit(self.X_train, self.y_train)
            return linear_regressor.predict(X_test)

        elif self.regressor_name == "Random Forest":
            # Random trees uses many random decision trees to output many different
results
            # we take the mean of the results
            random = RandomForestRegressor(n_estimators=100, max_depth=2)
            random.fit(self.X_train, self.y_train)
            return random.predict(X_test)

        elif self.regressor_name == 'KNN Regression':
            neigh = KNeighborsRegressor(n_neighbors=4)
            neigh.fit(self.X_train, self.y_train)
            return neigh.predict(X_test)

    def getScore(self, y_test, y_pred):
        if self.regressor_name == 'Polynomial Regression':
            return -metrics.r2_score(y_test, y_pred)
        else:
            return metrics.r2_score(y_test, y_pred)
```

classification.py

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics


class Classifier:
    def __init__(self, name, X, y):
        self.name = name
        self.X_train = X
        self.y_train = y
        self.fact = 0.32

    def classify(self, X_test):
        if self.name == 'KNN':
            n = KNeighborsClassifier(n_neighbors=4)
            n.fit(self.X_train, self.y_train)
            return n.predict(X_test)

        elif self.name == 'Decision Tree':
            decision_tree = DecisionTreeClassifier(max_depth=3)
            decision_tree.fit(self.X_train, self.y_train)
            return decision_tree.predict(X_test)

        elif self.name == 'Naive Bayes':
            b = GaussianNB()
            b.fit(self.X_train, self.y_train)
            return b.predict(X_test)

        elif self.name == "Random Forest":
            r = RandomForestClassifier(n_estimators=200, max_depth=3)
            r.fit(self.X_train, self.y_train)
            return r.predict(X_test)

    def calculateAccuracy(self, y_test, y_pred):
        return metrics.accuracy_score(y_test, y_pred) * 100
```

clustering.py

```python
from sklearn.cluster import KMeans


class Clustering:
    def __init__(self, algorithm_cluster, X_train):
        self.algorithm_cluster = algorithm_cluster
        self.X_train = X_train

    def predict(self, X_test):
        kmeans = KMeans(n_clusters=3)
        kmeans.fit(self.X_train)
        return kmeans.predict(X_test)
```