Course Topic:

In this activity, we will employ the following procedures:

1. Create our own distance measurements.
    1.1. Manhattan Distance
    1.2. Minkowski Distance
2. Documentation
    2.1. Document maximum of 10 bugs.

Data Used:

Iris.csv downloaded from Kaggle

Dataset pt.1: Iris Data Set

Source of Data:

https://www.kaggle.com/datasets/uciml/iris

Data Information:

The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, and can also be found on the UCI Machine Learning Repository.

It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

The columns in this dataset are:

- Id
- SepalLengthCm
- SepalWidthCm
- PetalLengthCm
- PetalWidthCm
- Species

Problem Statement:

The task is to read and load the provided dataset file into the program. This can be done using appropriate functions or libraries depending on the programming language being used. After loading the dataset, the next step involves handling any missing or null values. It is crucial to ensure the dataset is free from such inconsistencies, as they can introduce errors and affect the accuracy of subsequent analysis. One common approach is to drop the rows or instances that contain null or empty values. This ensures that only complete and reliable data is used for further processing. The next one is to create a distance measurement to assess the similarity of two points in a multidimensional space. This procedure uses Euclidean, Manhattan and Minkowski.

Method:

Algorithm:

We will implement the K-Nearest Neighbors (KNN) classifier algorithm, which is a supervised machine learning technique suitable for classification and regression tasks. In this case, our focus will be on its application for classification. The KNN algorithm operates by using a set of labeled training data that represents the known class labels of points in the feature space. When a new, unlabeled data point is provided, the algorithm calculates the distance between this point and all the points in the training data. By selecting the K-nearest points based on the chosen distance metric, the algorithm determines the closest neighbors to the given point. Once the K-nearest neighbors are identified, the algorithm assigns the most frequent class label among those K points as the predicted class label for the given point. In simpler terms, it makes a decision by taking a majority vote among its nearest neighbors to determine the class label.

Solution:

To implement the KNN algorithm on the Iris dataset, several preprocessing steps are required to ensure that the data is in a suitable format for training and testing. Firstly, the Iris dataset is loaded using the appropriate function, providing access to the feature matrix (X) and the corresponding class labels (y).

Next, the dataset is split into training and testing sets using the train_test_split function from scikit-learn. This allows for independent evaluation of the trained model's performance on unseen data. It is important to specify the desired proportion of the test set and use a random state for reproducibility.

Depending on the nature of the data and the chosen distance metric, feature scaling may be necessary. This step ensures that all features are on a similar scale and prevents certain features from dominating the distance calculations. Common techniques for feature scaling include standardization, which involves subtracting the mean and dividing by the standard deviation, and normalization, which scales the values to a specific range.

In the case of missing values in the dataset, appropriate handling techniques should be applied. This can involve imputation, where missing values are replaced with estimated values based on other data points, or deletion of instances or features with missing values, depending on the extent and impact of the missing data.

Categorical variables need to be encoded into numerical values. This can be achieved through techniques such as one-hot encoding or label encoding, depending on the nature of the categorical variables and the requirements of the algorithm.

Additional data preprocessing steps may be required based on the specific characteristics of the dataset. This can include feature selection to identify the most relevant features, dimensionality reduction techniques to reduce the number of features, or outlier detection and handling to address any outliers that may exist in the data.

By performing these preprocessing steps, the Iris dataset is appropriately prepared for training the KNN algorithm. This ensures that the algorithm can effectively learn from the data and make accurate predictions on new, unseen instances. The specific preprocessing steps may vary depending on the dataset characteristics and the requirements of the algorithm being applied.

Evaluation:

In evaluating the performance of the K-Nearest Neighbors (KNN) algorithm we use various evaluation metrics can be utilized. These metrics provide insights into the algorithm's accuracy and confusion matrix.

Accuracy is a widely used metric that measures the proportion of correctly classified instances out of the total number of instances in the test set. It provides an overall assessment of how well the algorithm correctly predicts the class labels. A confusion matrix provides a detailed analysis of the classifier's performance by displaying the counts of true positives, true negatives, false positives, and false negatives. It allows for a more comprehensive understanding of the algorithm's predictive capabilities.

By combining these evaluation metrics, it becomes feasible to conduct a thorough analysis of the model's performance, allowing for a meticulous assessment of its strengths and weaknesses. These metrics will play a crucial role in evaluating the accuracy of the KNN model, especially when considering different distance metrics. The insights gained from this evaluation will serve as valuable information for future optimization and refinement of the model, facilitating its further development.

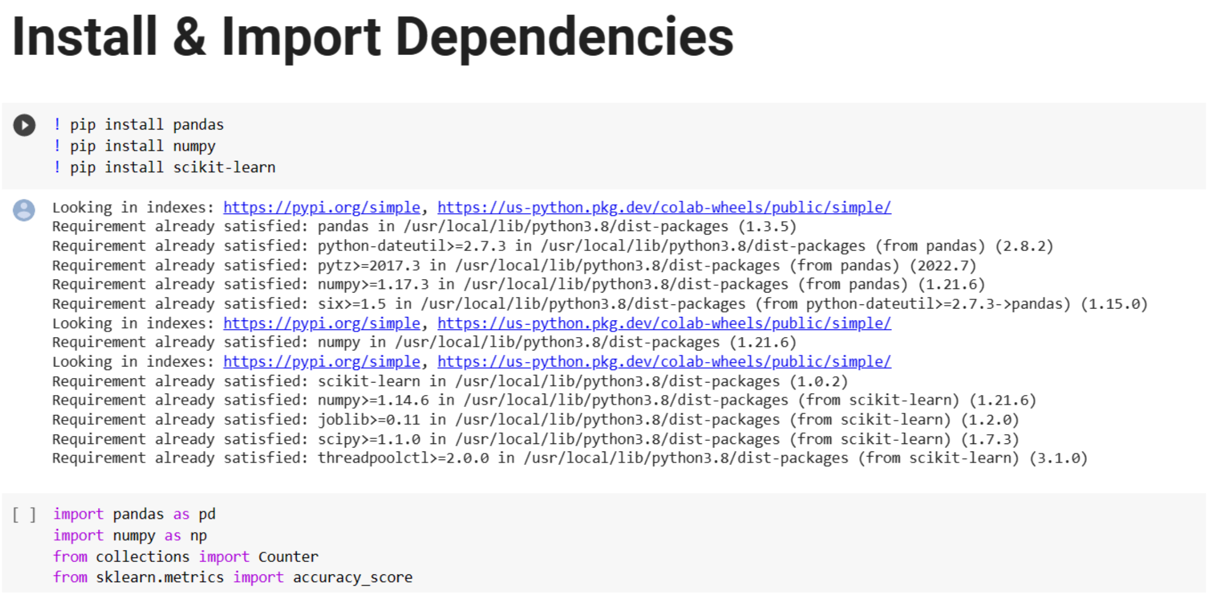
Data Preparation:

In this activity, these are the following processes we have done in data preparation.

1. Gathering/ loading dataset using the read_csv function
2. Checking for null or inconsistent values in the data.
3. Removing the null values.
4. Changing the needed values to new values

Results & Discussion

Data cleaning plays a fundamental role in the data analysis process as it ensures the accuracy, reliability, and quality of the data used for analysis. By identifying and addressing errors, inconsistencies, and missing values, data cleaning significantly enhances the integrity and usefulness of the dataset specifically the iris dataset that is used in this activity. First step is to load and clean and split the dataset, after that Euclidean, Manhattan, and Minkowski were used to calculate the test data and the training data.

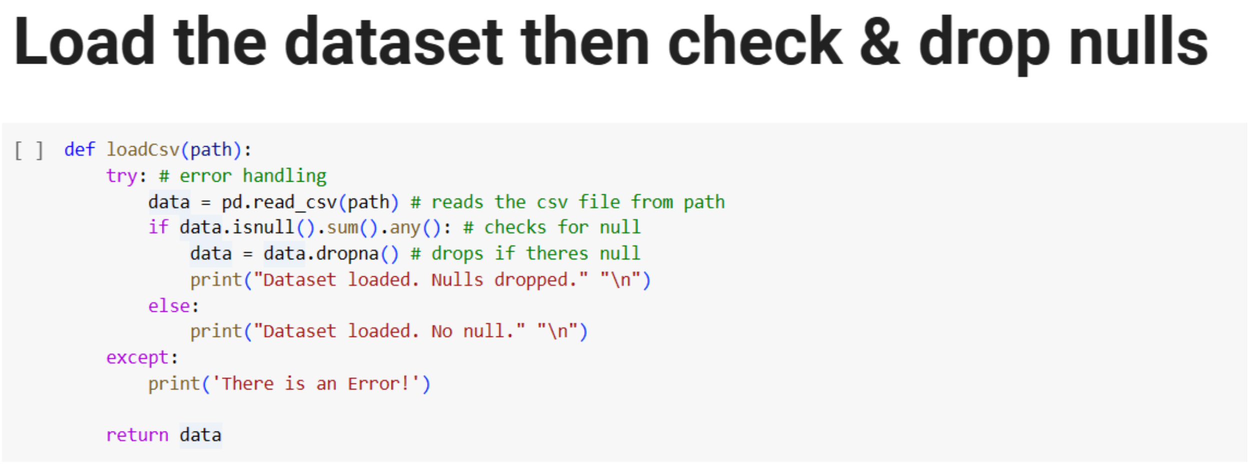


Install & Import Dependencies

```
! pip install pandas
! pip install numpy
! pip install scikit-learn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pandas in /usr/local/lib/python3.8/dist-packages (1.3.5)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas) (2022.7)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.8/dist-packages (from pandas) (1.21.6)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (1.21.6)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.8/dist-packages (1.0.2)
Requirement already satisfied: numpy>=1.14.6 in /usr/local/lib/python3.8/dist-packages (from scikit-learn) (1.21.6)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.8/dist-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn) (1.7.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn) (3.1.0)
```

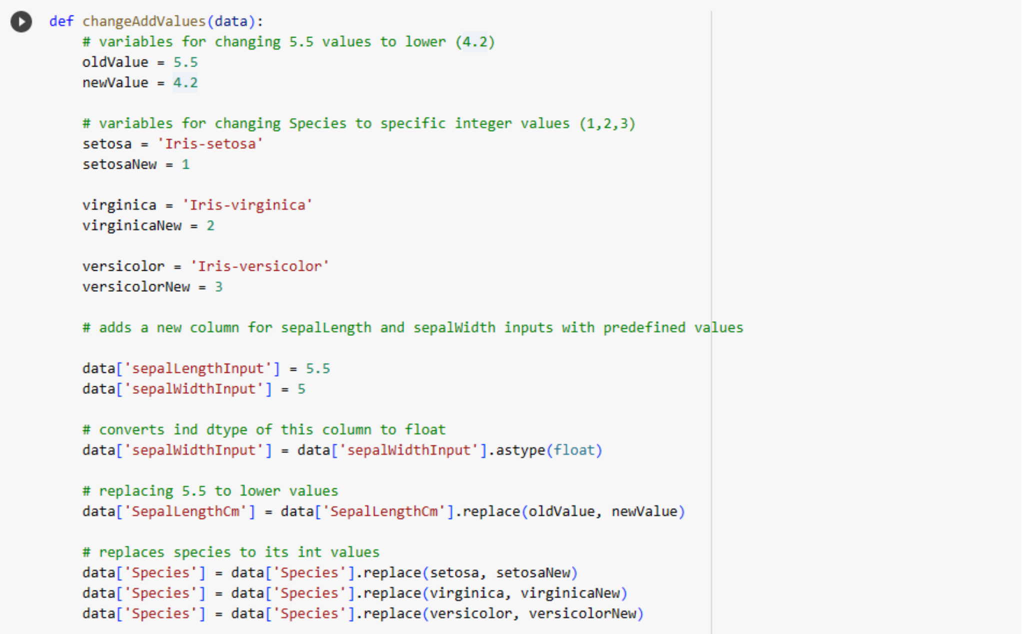
```
[ ] import pandas as pd
    import numpy as np
    from collections import Counter
    from sklearn.metrics import accuracy_score
```



Load the dataset then check & drop nulls

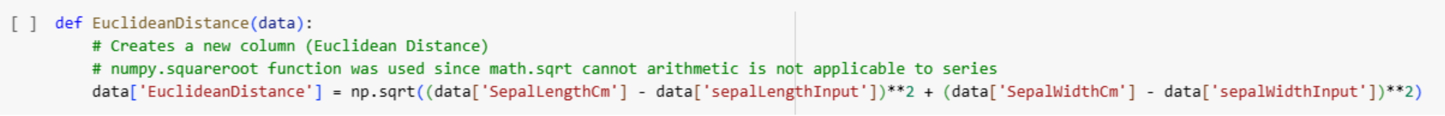
```
[ ] def loadCsv(path):
        try: # error handling
            data = pd.read_csv(path) # reads the csv file from path
            if data.isnull().sum().any(): # checks for null
                data = data.dropna() # drops if theres null
                print("Dataset loaded. Nulls dropped." "\n")
            else:
                print("Dataset loaded. No null." "\n")
        except:
            print('There is an Error!')

        return data
```

Changing the needed values to new values

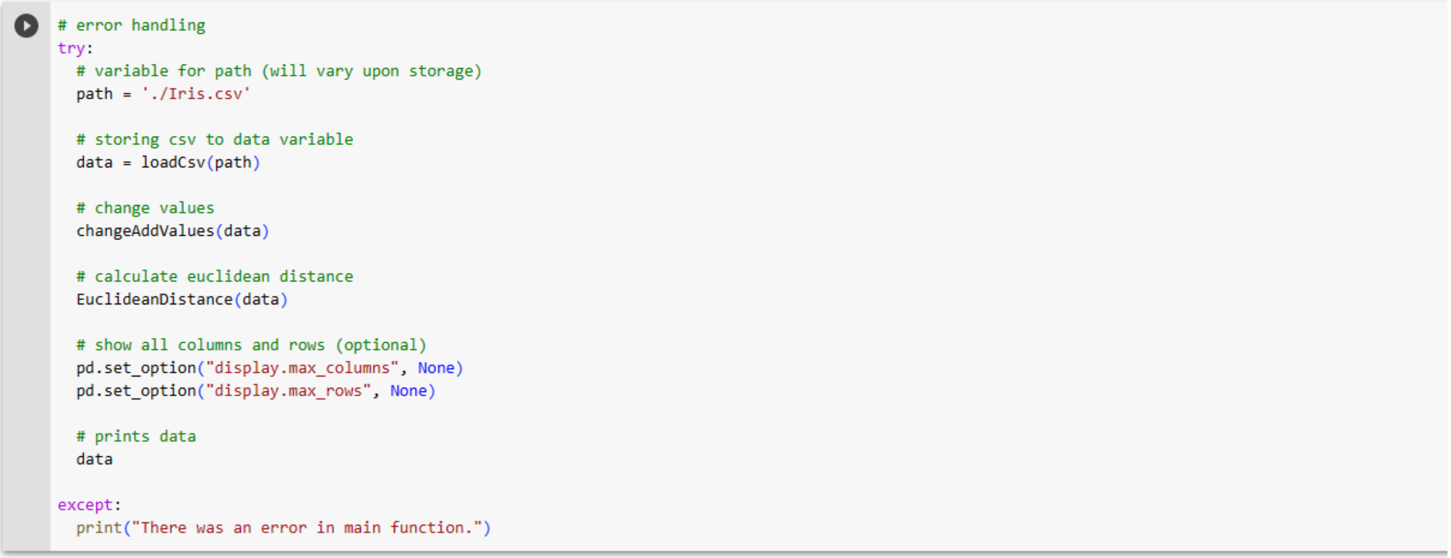


```python
def changeAddValues(data):
    # variables for changing 5.5 values to lower (4.2)
    oldValue = 5.5
    newValue = 4.2

    # variables for changing Species to specific integer values (1,2,3)
    setosa = 'Iris-setosa'
    setosaNew = 1

    virginica = 'Iris-virginica'
    virginicaNew = 2

    versicolor = 'Iris-versicolor'
    versicolorNew = 3

    # adds a new column for sepalLength and sepalWidth inputs with predefined values

    data['sepalLengthInput'] = 5.5
    data['sepalWidthInput'] = 5

    # converts ind dtype of this column to float
    data['sepalWidthInput'] = data['sepalWidthInput'].astype(float)

    # replacing 5.5 to lower values
    data['SepalLengthCm'] = data['SepalLengthCm'].replace(oldValue, newValue)

    # replaces species to its int values
    data['Species'] = data['Species'].replace(setosa, setosaNew)
    data['Species'] = data['Species'].replace(virginica, virginicaNew)
    data['Species'] = data['Species'].replace(versicolor, versicolorNew)
```

Euclidean Distance FOrmula

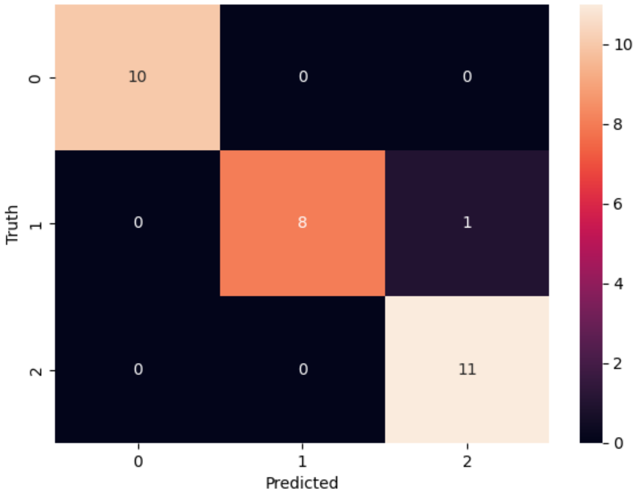


```python
def EuclideanDistance(data):
    # Creates a new column (Euclidean Distance)
    # numpy.squareroot function was used since math.sqrt cannot arithmetic is not applicable to series
    data['EuclideanDistance'] = np.sqrt((data['SepalLengthCm'] - data['sepalLengthInput'])**2 + (data['SepalWidthCm'] - data['sepalWidthInput'])**2)
```

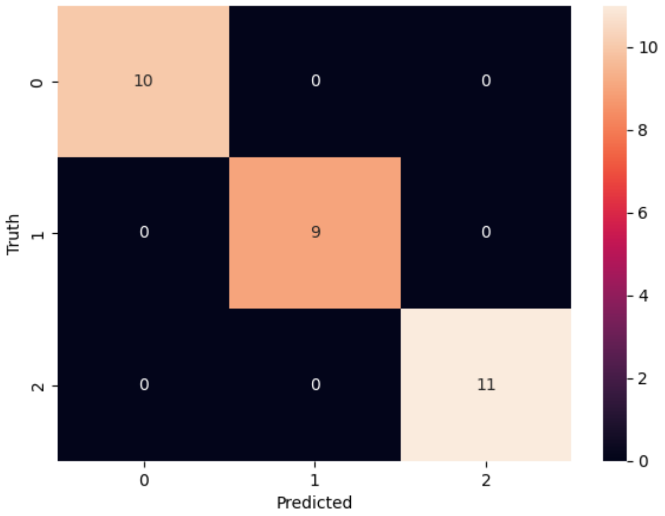


```python
# error handling
try:
  # variable for path (will vary upon storage)
  path = './Iris.csv'

  # storing csv to data variable
  data = loadCsv(path)

  # change values
  changeAddValues(data)

  # calculate euclidean distance
  EuclideanDistance(data)

  # show all columns and rows (optional)
  pd.set_option("display.max_columns", None)
  pd.set_option("display.max_rows", None)

  # prints data
  data

except:
  print("There was an error in main function.")
```

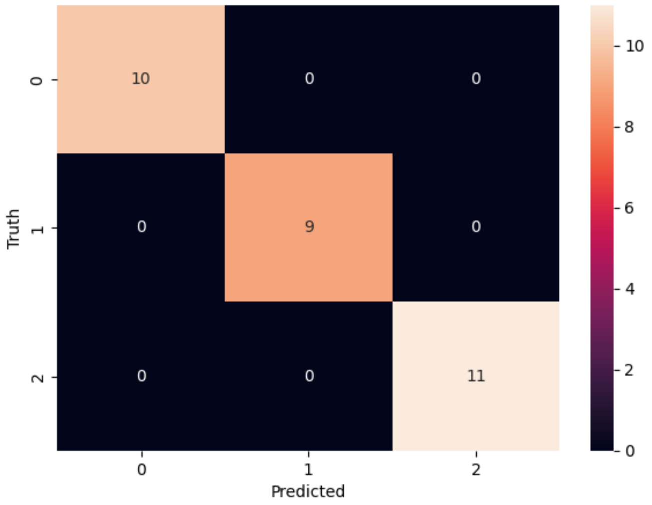
Euclidean Distance Manually and Scikit



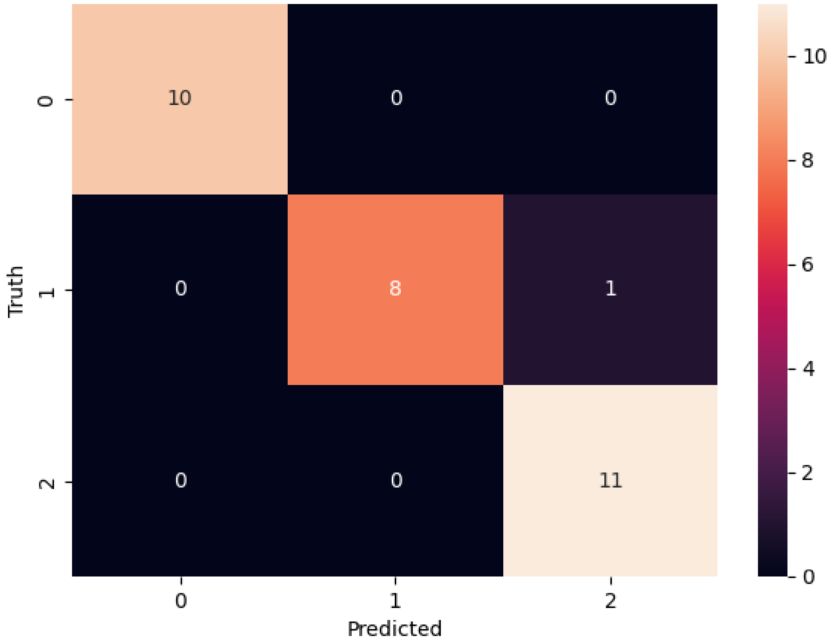
Manhattan Distance Manually and Scikit



Minkowski Distance Manually



Minkowski Distance Scikit



The bug is that the Minkowski distance using SciKit Learn function has different accuracy with the manual Minkowski distance.

Codes: Activity1.ipynb - Colaboratory (google.com)
DataMiningAss1.ipynb - Colaboratory (google.com)
1stActivity.ipynb - Colaboratory (google.com)