```
import pandas as pd
import numpy as np
import math
import csv
from sklearn.preprocessing import StandardScaler
import random
import operator
import warnings
warnings.filterwarnings("ignore")

from google.colab import files
import io

uploaded = files.upload()
data = pd.read_csv(io.BytesIO(uploaded['galexa.csv']))

X = data.iloc[:,1:]
Y = data.iloc[:,0]

# Feature Scaling

scaler = StandardScaler()
X = scaler.fit_transform(X)

data = pd.DataFrame(X)
data["class"] = Y
data.head()
```

Choose Files galexa.csv

- **galexa.csv**(n/a) - 2240491 bytes, last modified: 11/19/2019 - 100% done

```
Saving galexa.csv to galexa.csv
```

	0	1	2	3	4	5	6	7	
0	-0.939842	-1.199701	0.444210	0.396735	0.807325	1.103595	1.131451	1.544424	0.
1	-0.905516	-1.131516	1.023331	1.515638	1.085099	0.496391	0.310947	1.623933	-0.
2	-0.915016	-1.128648	0.738771	0.090590	-0.482489	-0.558285	-0.611033	0.832115	1.
3	-1.000545	-1.090998	0.310241	0.362857	-0.137998	-0.325304	-0.450122	1.373157	0.
4	-0.982357	-1.067140	0.245849	0.617630	1.197558	1.519469	1.426867	1.412500	-0.

```
# Reading the scaled Dataset (Standard Scaler used to scale data)

uploaded = files.upload()
data = pd.read_csv(io.BytesIO(uploaded['galexa_scaled.csv']))

# data.head()
```

Choose Files galexa_scaled.csv

- **galexa_scaled.csv**(n/a) - 2418719 bytes, last modified: 11/25/2019 - 100% done

```
Saving galexa_scaled.csv to galexa_scaled.csv
```

```
# Formatting the dataset as required for the KNN Algorithm

dataset = list()
with open("galexa_scaled.csv", 'r') as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        if not row:
            continue
        dataset.append(row)
# print(dataset[3])


# Calculating the euclidean distance between the two vectors/instances

# def edistance(inst1, inst2, _norm=np.linalg.norm):
#     i1 = np.array(inst1[:-1])
#     i2 = np.array(inst2[:-1])
#     return _norm(i1-i2)

def edistance(inst1, inst2):
  d = 0.0
  max_iter = len(inst1)-1
  i = 0
  while i < max_iter:
    d += (float(inst1[i]) - float(inst2[i]))**2
    i += 1
  # print(i)
  return math.sqrt(d)


# Getting the nearest neighbors for the instance

def neighs(train_data, tst_data_row, no_of_neighs):
    distances = []

    max_iter = len(train_data)
    i = 0
    while(i < max_iter):
        distance = edistance(tst_data_row,train_data[i],)
        distances.append((train_data[i],distance))
        i += 1

    distances = sorted(distances,key = lambda t:t[1])
    neighbors = []

    j = 0
    max_neigh = no_of_neighs;
    while(j < max_neigh):
        neighbors.append(distances[j][0])
        j += 1

    return neighbors


# Predict the class
```

```
def class_predict(train_data, tst_data_row, no_of_neighs):
    neighbors = neighs(train_data, tst_data_row, no_of_neighs)
    output = []

    max_iter = len(neighbors)
    i = 0
    while i < max_iter:
        output.append(neighbors[i][-1])
        i += 1
    output_vals = set(output)

    prediction = max(output_vals, key=output.count)
    return prediction


# Accuracy Metric

def calc_accuracy(y_true, y_pred):
  crt_count = 0
  max_iter = len(y_true)
  i = 0
  while i < max_iter:
    if y_true[i] != y_pred[i]:
      pass
    else:
      crt_count += 1
    i += 1
  result = crt_count / float(max_iter)
  return result


# KNN Algorithm

def knn_algo(train_data, test_data, no_of_neighs):
  p = list()
  max_iter = len(test_data)
  i = 0
  while i < max_iter:
    opt = class_predict(train_data,test_data[i], no_of_neighs)
    p.append(opt)
    i += 1

  return(p)


# KFold Validation : Splitting the dataset into k folds

def c_v_split(df, k_folds):
  df_split = list()
  copy_df = list(df)
  size = int(len(df) / k_folds)
  max_iter = k_folds
  i = 0
  while i < max_iter:
    f = list()
    while len(f) < size:
      index = random.randrange(len(copy_df))
```

```
      index = random.randrange(len(copy_df))
      f.append(copy_df.pop(index))
    df_split.append(f)
    i += 1
  return df_split


# Finally evaluating the algorithm using a cross-validation-split

def evaluate(dataset, algo, k_folds, *args):
  folds = c_v_split(dataset, k_folds)
  res = list()
  max_iter = len(folds)
  i = 0
  while i < max_iter:
    train_s = list(folds)
    train_s.remove(folds[i])
    train_s = sum(train_s, [])
    test_s = list()
    fold = folds[i]
    for r in fold:
      copy_r = list(r)
      test_s.append(copy_r)
      copy_r[-1] = None
    y_pred = algo(train_s, test_s, *args)
    y_true = [r[-1] for r in fold]
    acc = calc_accuracy(y_true, y_pred)
    res.append(acc)
    i += 1

  return res


# Test Case 1

k_folds = 8
no_of_neighbors = 10
result = evaluate(dataset, knn_algo, k_folds, no_of_neighbors)


print('%s' % result)
print('Accuracy: %.3f%%' % (sum(result)/float(len(result))))
```

```
[0.8838095238095238, 0.8866666666666667, 0.8828571428571429, 0.8838095238095238, 0.88
Accuracy: 0.886%
```