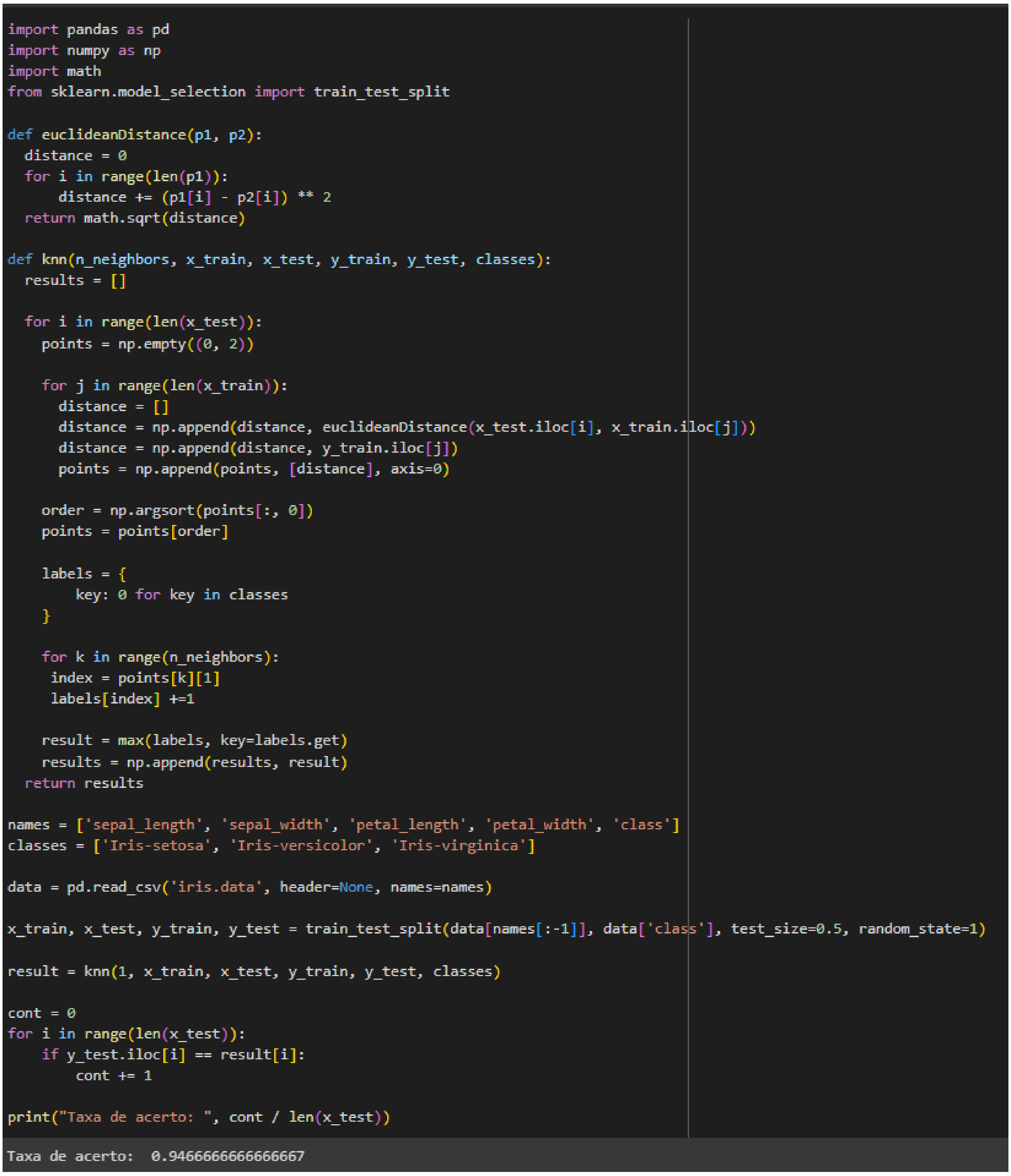
Aprendizagem de Máquina: Atividade 03 − K-NN

Carlos Emmanuel Pereira Alves
Curso de Bacharelado em Ciência da Computação
Universidade Federal do Agreste de Pernambuco (UFAPE)
Garanhuns, Brasil
carlos.emmanuel.236@gmail.com

1) Construa sua própria implementação do classificador pelo vizinho mais próximo (1 − NN) utilizando distância euclidiana. Avalie este classificador utilizando metade dos exemplos de cada classe da base Iris (archive.ics.uci.edu/ml/datasets/iris) como conjunto de teste e o restante como conjunto de treinamento. Atenção: construa também a implementação da distância euclidiana.

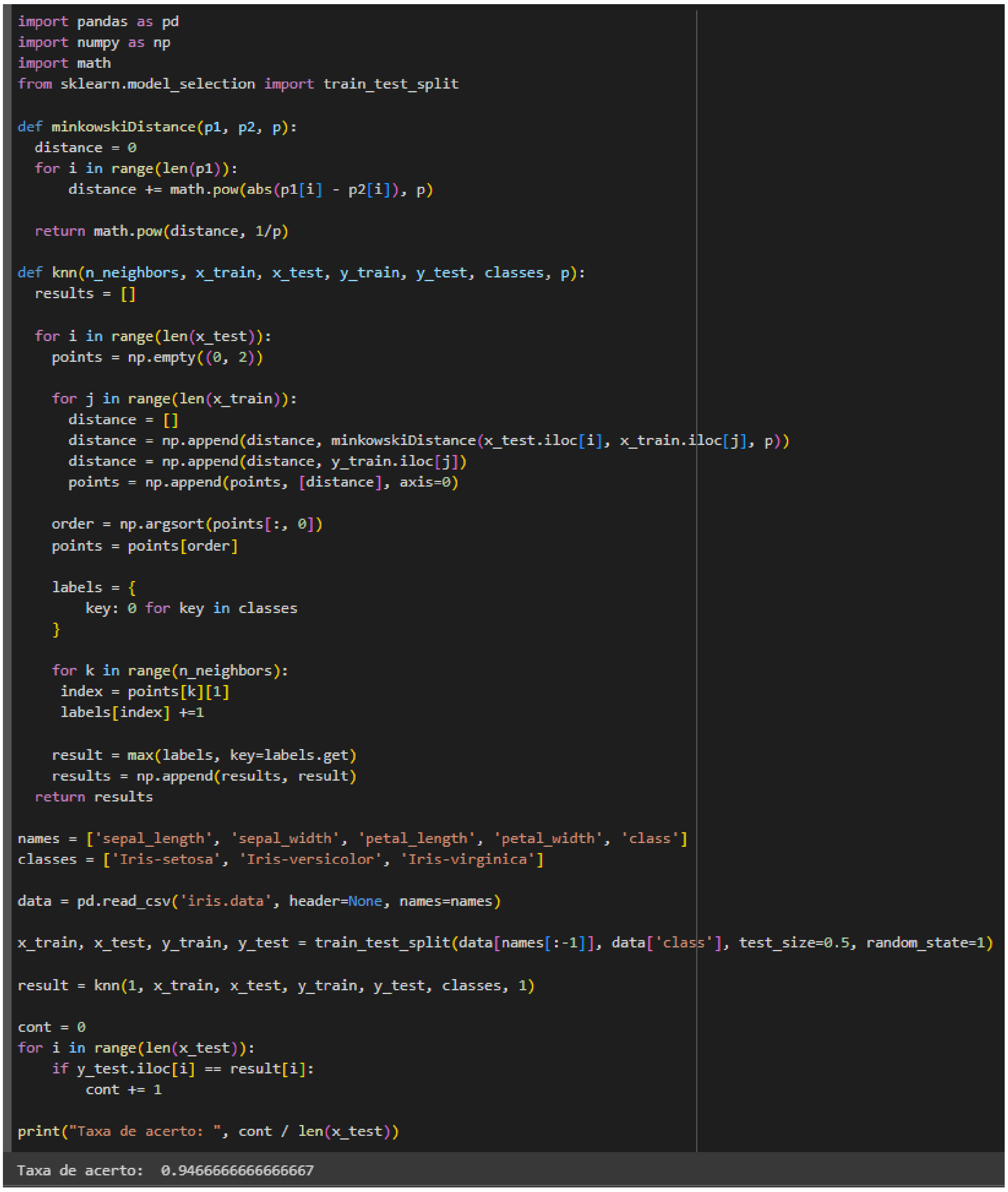


```python
import pandas as pd
import numpy as np
import math
from sklearn.model_selection import train_test_split

def euclideanDistance(p1, p2):
  distance = 0
  for i in range(len(p1)):
      distance += (p1[i] - p2[i]) ** 2
  return math.sqrt(distance)

def knn(n_neighbors, x_train, x_test, y_train, y_test, classes):
  results = []

  for i in range(len(x_test)):
    points = np.empty((0, 2))

    for j in range(len(x_train)):
      distance = []
      distance = np.append(distance, euclideanDistance(x_test.iloc[i], x_train.iloc[j]))
      distance = np.append(distance, y_train.iloc[j])
      points = np.append(points, [distance], axis=0)

    order = np.argsort(points[:, 0])
    points = points[order]

    labels = {
        key: 0 for key in classes
    }

    for k in range(n_neighbors):
     index = points[k][1]
     labels[index] +=1

    result = max(labels, key=labels.get)
    results = np.append(results, result)
  return results

names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
classes = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

data = pd.read_csv('iris.data', header=None, names=names)

x_train, x_test, y_train, y_test = train_test_split(data[names[:-1]], data['class'], test_size=0.5, random_state=1)

result = knn(1, x_train, x_test, y_train, y_test, classes)

cont = 0
for i in range(len(x_test)):
    if y_test.iloc[i] == result[i]:
        cont += 1

print("Taxa de acerto: ", cont / len(x_test))
```

```
Taxa de acerto:  0.9466666666666667
```

2) Refaça a questão anterior implementando a distância de Minkowski, descrita abaixo. Calcule os resultados para p = 1, p = 2 e p = 4.

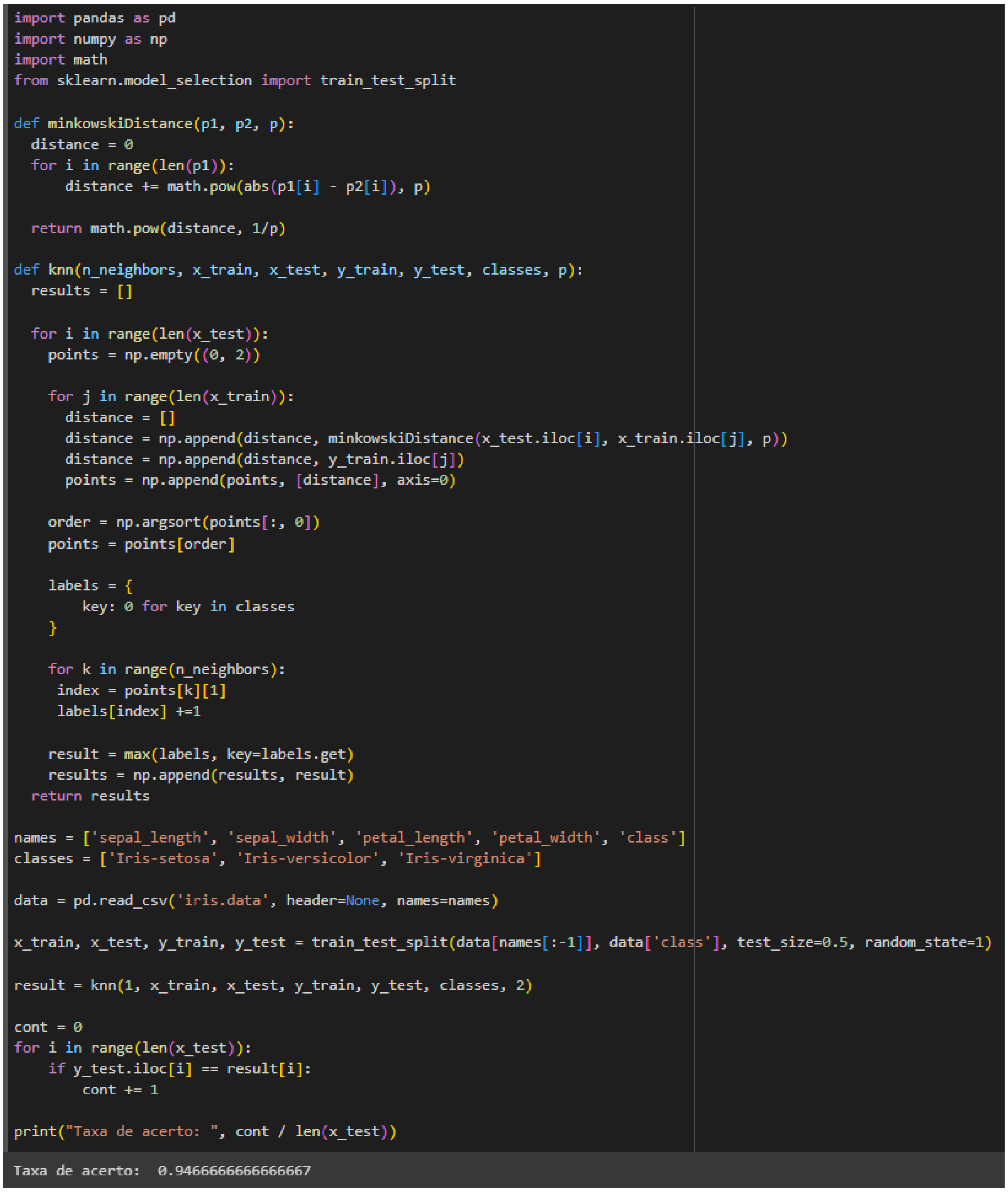
p=1



```
import pandas as pd
import numpy as np
import math
from sklearn.model_selection import train_test_split

def minkowskiDistance(p1, p2, p):
  distance = 0
  for i in range(len(p1)):
      distance += math.pow(abs(p1[i] - p2[i]), p)

  return math.pow(distance, 1/p)

def knn(n_neighbors, x_train, x_test, y_train, y_test, classes, p):
  results = []

  for i in range(len(x_test)):
    points = np.empty((0, 2))

    for j in range(len(x_train)):
      distance = []
      distance = np.append(distance, minkowskiDistance(x_test.iloc[i], x_train.iloc[j], p))
      distance = np.append(distance, y_train.iloc[j])
      points = np.append(points, [distance], axis=0)

    order = np.argsort(points[:, 0])
    points = points[order]

    labels = {
        key: 0 for key in classes
    }

    for k in range(n_neighbors):
     index = points[k][1]
     labels[index] +=1

    result = max(labels, key=labels.get)
    results = np.append(results, result)
  return results

names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
classes = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

data = pd.read_csv('iris.data', header=None, names=names)

x_train, x_test, y_train, y_test = train_test_split(data[names[:-1]], data['class'], test_size=0.5, random_state=1)

result = knn(1, x_train, x_test, y_train, y_test, classes, 1)

cont = 0
for i in range(len(x_test)):
    if y_test.iloc[i] == result[i]:
        cont += 1

print("Taxa de acerto: ", cont / len(x_test))
```

```
Taxa de acerto:  0.9466666666666667
```

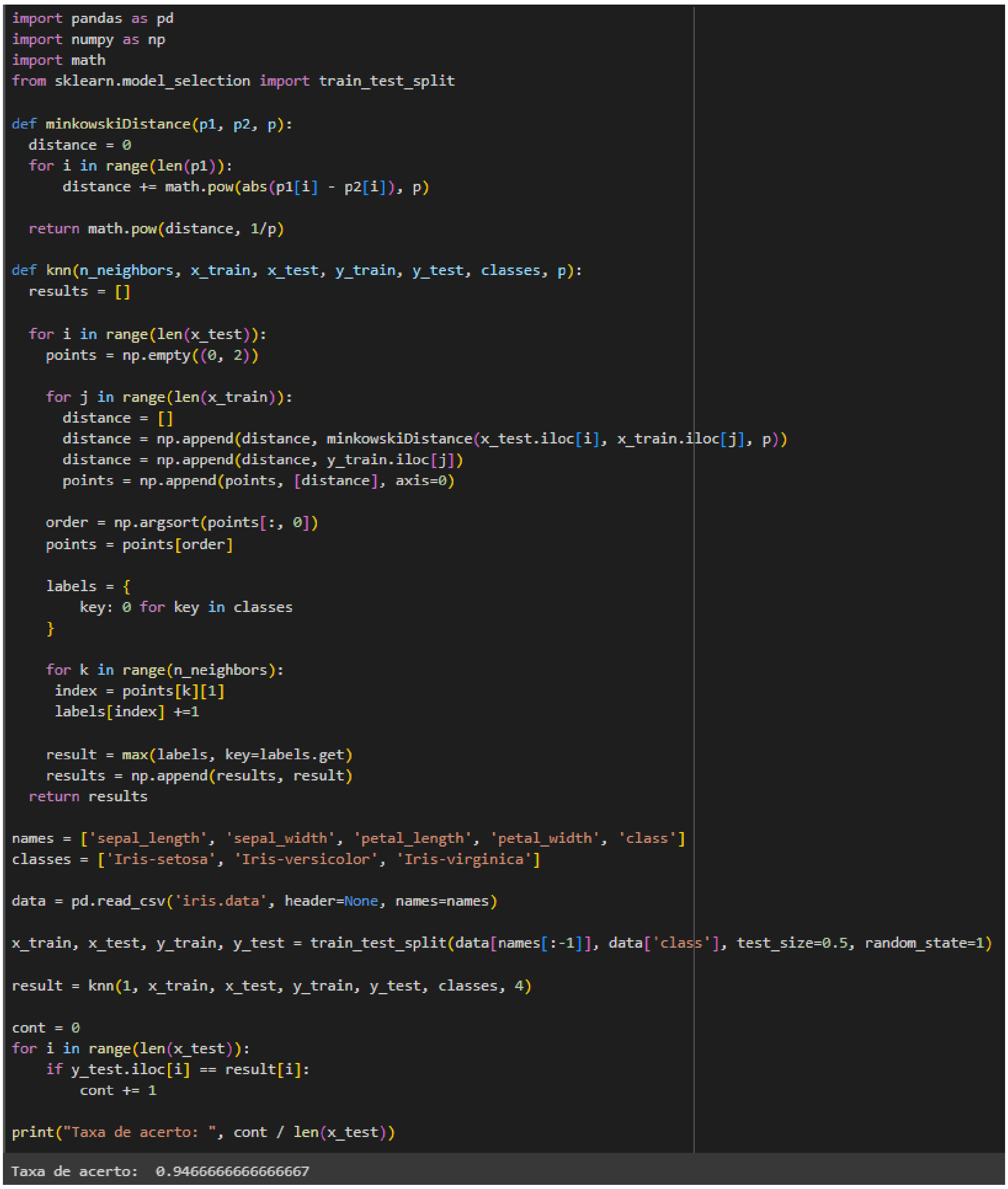
p=2



```python
import pandas as pd
import numpy as np
import math
from sklearn.model_selection import train_test_split

def minkowskiDistance(p1, p2, p):
  distance = 0
  for i in range(len(p1)):
      distance += math.pow(abs(p1[i] - p2[i]), p)

  return math.pow(distance, 1/p)

def knn(n_neighbors, x_train, x_test, y_train, y_test, classes, p):
  results = []

  for i in range(len(x_test)):
    points = np.empty((0, 2))

    for j in range(len(x_train)):
      distance = []
      distance = np.append(distance, minkowskiDistance(x_test.iloc[i], x_train.iloc[j], p))
      distance = np.append(distance, y_train.iloc[j])
      points = np.append(points, [distance], axis=0)

    order = np.argsort(points[:, 0])
    points = points[order]

    labels = {
        key: 0 for key in classes
    }

    for k in range(n_neighbors):
     index = points[k][1]
     labels[index] +=1

    result = max(labels, key=labels.get)
    results = np.append(results, result)
  return results

names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
classes = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

data = pd.read_csv('iris.data', header=None, names=names)

x_train, x_test, y_train, y_test = train_test_split(data[names[:-1]], data['class'], test_size=0.5, random_state=1)

result = knn(1, x_train, x_test, y_train, y_test, classes, 2)

cont = 0
for i in range(len(x_test)):
    if y_test.iloc[i] == result[i]:
        cont += 1

print("Taxa de acerto: ", cont / len(x_test))
```

```
Taxa de acerto:  0.9466666666666667
```

p=4

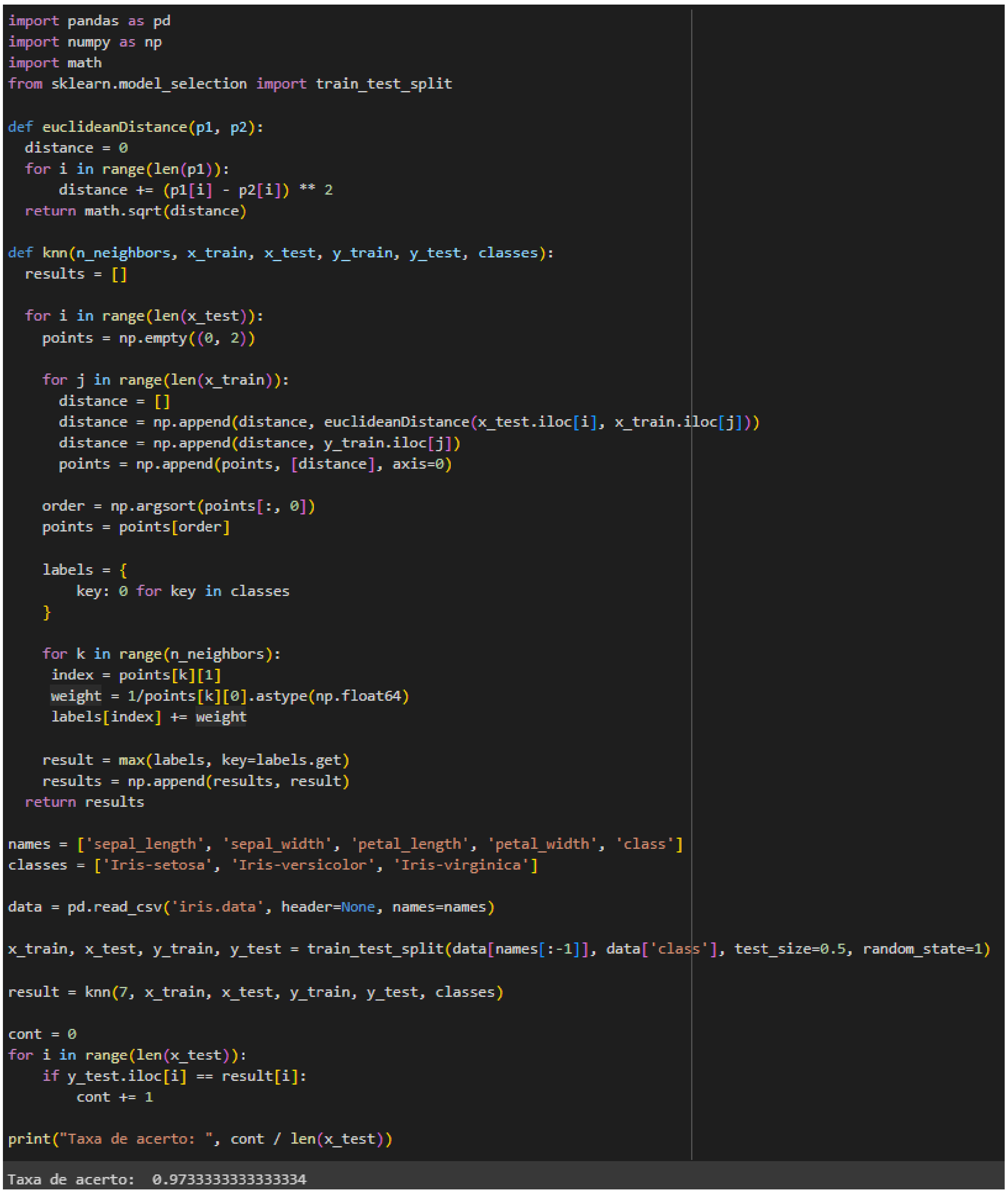


```
import pandas as pd
import numpy as np
import math
from sklearn.model_selection import train_test_split

def minkowskiDistance(p1, p2, p):
  distance = 0
  for i in range(len(p1)):
      distance += math.pow(abs(p1[i] - p2[i]), p)

  return math.pow(distance, 1/p)

def knn(n_neighbors, x_train, x_test, y_train, y_test, classes, p):
  results = []

  for i in range(len(x_test)):
    points = np.empty((0, 2))

    for j in range(len(x_train)):
      distance = []
      distance = np.append(distance, minkowskiDistance(x_test.iloc[i], x_train.iloc[j], p))
      distance = np.append(distance, y_train.iloc[j])
      points = np.append(points, [distance], axis=0)

    order = np.argsort(points[:, 0])
    points = points[order]

    labels = {
        key: 0 for key in classes
    }

    for k in range(n_neighbors):
     index = points[k][1]
     labels[index] +=1

    result = max(labels, key=labels.get)
    results = np.append(results, result)
  return results

names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
classes = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

data = pd.read_csv('iris.data', header=None, names=names)

x_train, x_test, y_train, y_test = train_test_split(data[names[:-1]], data['class'], test_size=0.5, random_state=1)

result = knn(1, x_train, x_test, y_train, y_test, classes, 4)

cont = 0
for i in range(len(x_test)):
    if y_test.iloc[i] == result[i]:
        cont += 1

print("Taxa de acerto: ", cont / len(x_test))
```

```
Taxa de acerto:  0.9466666666666667
```

3) Construa sua própria implementação dos classificadores:

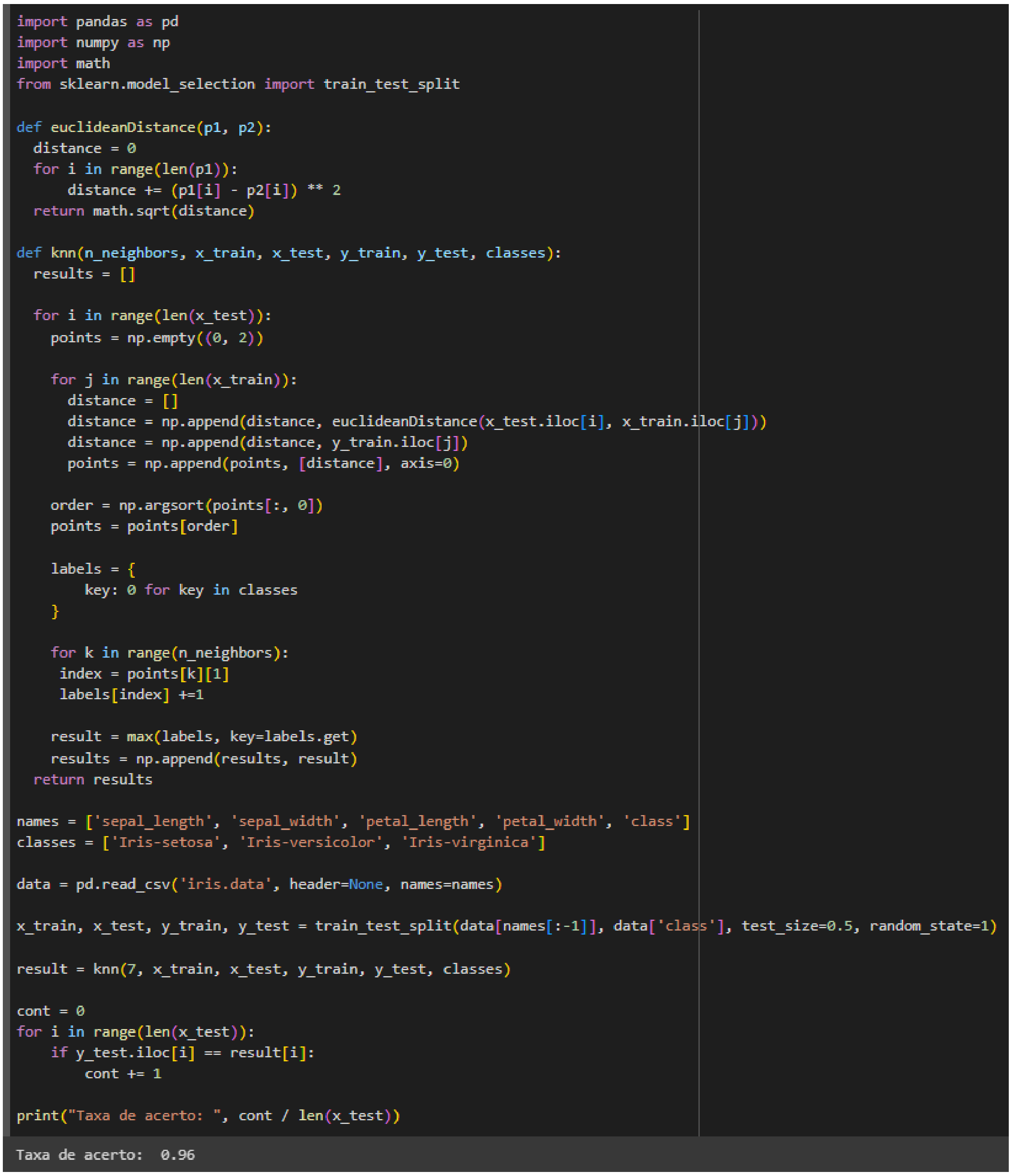
a) 7-NN com peso



```python
import pandas as pd
import numpy as np
import math
from sklearn.model_selection import train_test_split

def euclideanDistance(p1, p2):
  distance = 0
  for i in range(len(p1)):
      distance += (p1[i] - p2[i]) ** 2
  return math.sqrt(distance)

def knn(n_neighbors, x_train, x_test, y_train, y_test, classes):
  results = []

  for i in range(len(x_test)):
    points = np.empty((0, 2))

    for j in range(len(x_train)):
      distance = []
      distance = np.append(distance, euclideanDistance(x_test.iloc[i], x_train.iloc[j]))
      distance = np.append(distance, y_train.iloc[j])
      points = np.append(points, [distance], axis=0)

    order = np.argsort(points[:, 0])
    points = points[order]

    labels = {
        key: 0 for key in classes
    }

    for k in range(n_neighbors):
     index = points[k][1]
     weight = 1/points[k][0].astype(np.float64)
     labels[index] += weight

    result = max(labels, key=labels.get)
    results = np.append(results, result)
  return results

names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
classes = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

data = pd.read_csv('iris.data', header=None, names=names)

x_train, x_test, y_train, y_test = train_test_split(data[names[:-1]], data['class'], test_size=0.5, random_state=1)

result = knn(7, x_train, x_test, y_train, y_test, classes)

cont = 0
for i in range(len(x_test)):
    if y_test.iloc[i] == result[i]:
        cont += 1

print("Taxa de acerto: ", cont / len(x_test))
```

```
Taxa de acerto:  0.9733333333333334
```

b) 7-NN sem peso



```
import pandas as pd
import numpy as np
import math
from sklearn.model_selection import train_test_split

def euclideanDistance(p1, p2):
  distance = 0
  for i in range(len(p1)):
      distance += (p1[i] - p2[i]) ** 2
  return math.sqrt(distance)

def knn(n_neighbors, x_train, x_test, y_train, y_test, classes):
  results = []

  for i in range(len(x_test)):
    points = np.empty((0, 2))

    for j in range(len(x_train)):
      distance = []
      distance = np.append(distance, euclideanDistance(x_test.iloc[i], x_train.iloc[j]))
      distance = np.append(distance, y_train.iloc[j])
      points = np.append(points, [distance], axis=0)

    order = np.argsort(points[:, 0])
    points = points[order]

    labels = {
        key: 0 for key in classes
    }

    for k in range(n_neighbors):
     index = points[k][1]
     labels[index] +=1

    result = max(labels, key=labels.get)
    results = np.append(results, result)
  return results

names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
classes = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

data = pd.read_csv('iris.data', header=None, names=names)

x_train, x_test, y_train, y_test = train_test_split(data[names[:-1]], data['class'], test_size=0.5, random_state=1)

result = knn(7, x_train, x_test, y_train, y_test, classes)

cont = 0
for i in range(len(x_test)):
    if y_test.iloc[i] == result[i]:
        cont += 1

print("Taxa de acerto: ", cont / len(x_test))
```

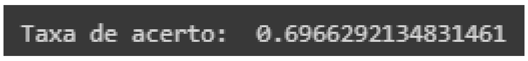
```
Taxa de acerto:  0.96
```

e avalie os classificadores utilizando metade dos exemplos de cada classe da base Iris como conjunto de teste e a outra metade como conjunto de treinamento.

4) Utilizando sua implementação do k-NN e, divida a base Wine archive.ics.uci.edu/ml/datasets/Wine utilizando 50% da classe para treino

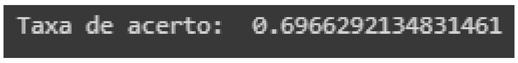
e o restante para teste. Avalie vários valores de k e determine qual é aquele que gera a maior taxa de acerto.

1-nn:



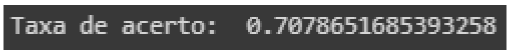
```
Taxa de acerto:  0.6966292134831461
```

3-nn:



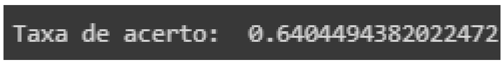
```
Taxa de acerto:  0.6966292134831461
```

5-nn:



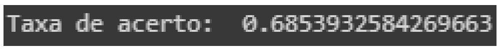
```
Taxa de acerto:  0.7078651685393258
```

7-nn:



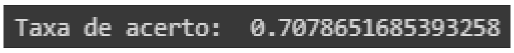
```
Taxa de acerto:  0.6404494382022472
```

9-nn:



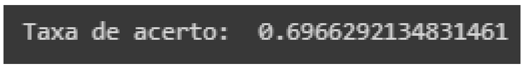
```
Taxa de acerto:  0.6853932584269663
```

11-nn:



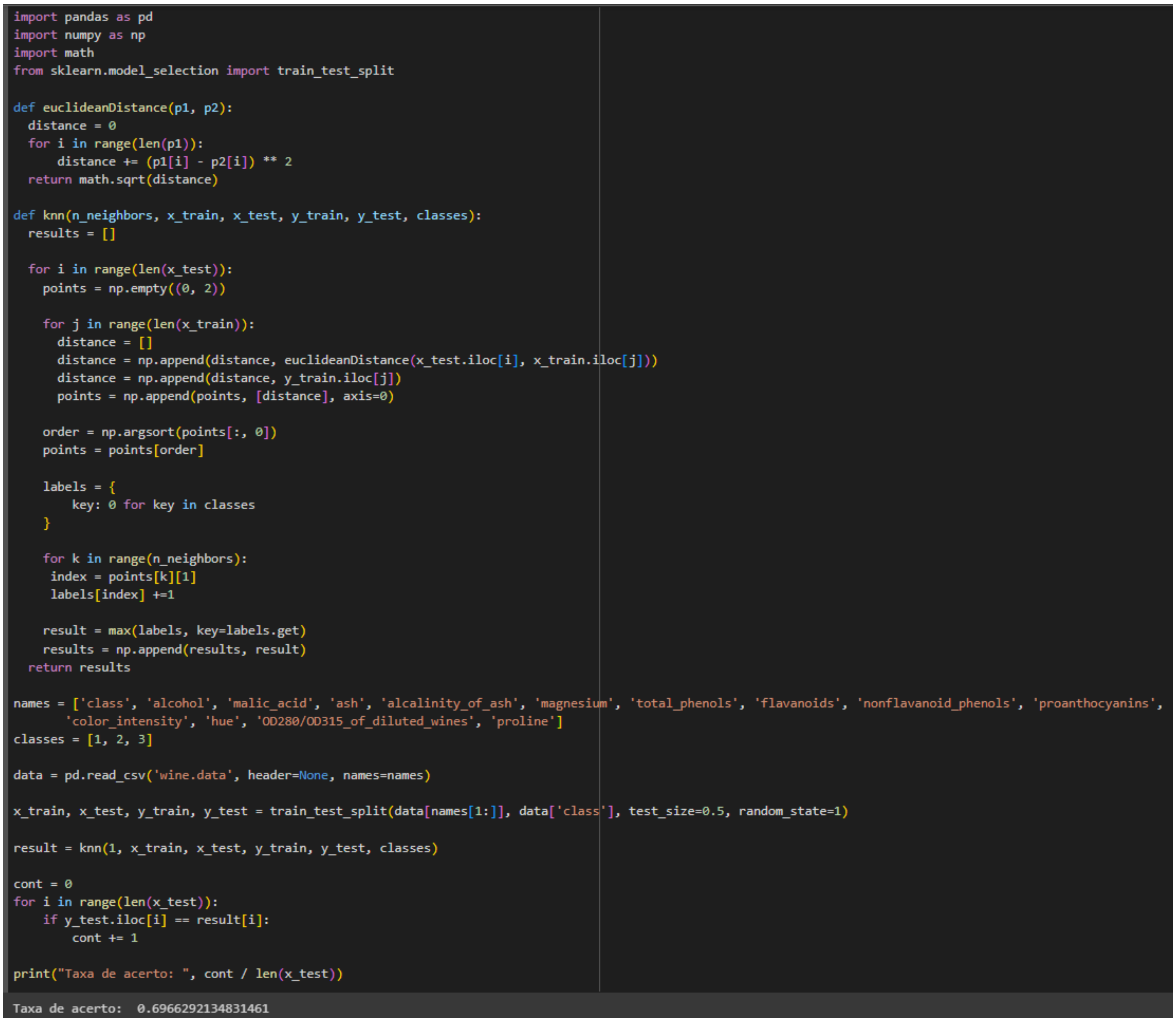
```
Taxa de acerto:  0.7078651685393258
```

13-nn:



```
Taxa de acerto:  0.6966292134831461
```

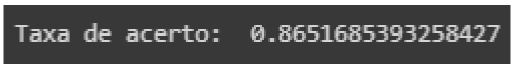
Os melhores valores foram 5 e 11.



```
import pandas as pd
import numpy as np
import math
from sklearn.model_selection import train_test_split

def euclideanDistance(p1, p2):
  distance = 0
  for i in range(len(p1)):
      distance += (p1[i] - p2[i]) ** 2
  return math.sqrt(distance)

def knn(n_neighbors, x_train, x_test, y_train, y_test, classes):
  results = []

  for i in range(len(x_test)):
    points = np.empty((0, 2))

    for j in range(len(x_train)):
      distance = []
      distance = np.append(distance, euclideanDistance(x_test.iloc[i], x_train.iloc[j]))
      distance = np.append(distance, y_train.iloc[j])
      points = np.append(points, [distance], axis=0)

    order = np.argsort(points[:, 0])
    points = points[order]

    labels = {
        key: 0 for key in classes
    }

    for k in range(n_neighbors):
     index = points[k][1]
     labels[index] +=1

    result = max(labels, key=labels.get)
    results = np.append(results, result)
  return results

names = ['class', 'alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins',
       'color_intensity', 'hue', 'OD280/OD315_of_diluted_wines', 'proline']
classes = [1, 2, 3]

data = pd.read_csv('wine.data', header=None, names=names)

x_train, x_test, y_train, y_test = train_test_split(data[names[1:]], data['class'], test_size=0.5, random_state=1)

result = knn(1, x_train, x_test, y_train, y_test, classes)

cont = 0
for i in range(len(x_test)):
    if y_test.iloc[i] == result[i]:
        cont += 1

print("Taxa de acerto: ", cont / len(x_test))
```

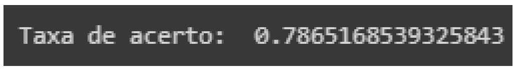
```
Taxa de acerto:  0.6966292134831461
```

5) Refaça o experimento da questão anterior removendo a última coluna da base.
   a) Compare com os resultados da questão anterior.
      1-nn:



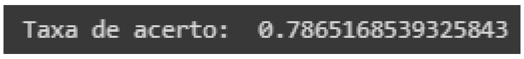
```
Taxa de acerto:  0.8651685393258427
```

      3-nn:



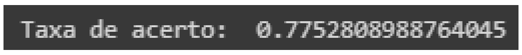
```
Taxa de acerto:  0.7865168539325843
```

5-nn:



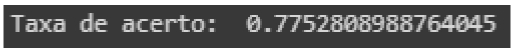
```
Taxa de acerto:  0.7865168539325843
```

7-nn:



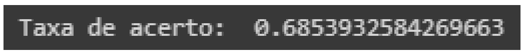
```
Taxa de acerto:  0.7752808988764045
```

9-nn:



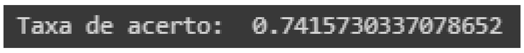
```
Taxa de acerto:  0.7752808988764045
```

11-nn:



```
Taxa de acerto:  0.6853932584269663
```

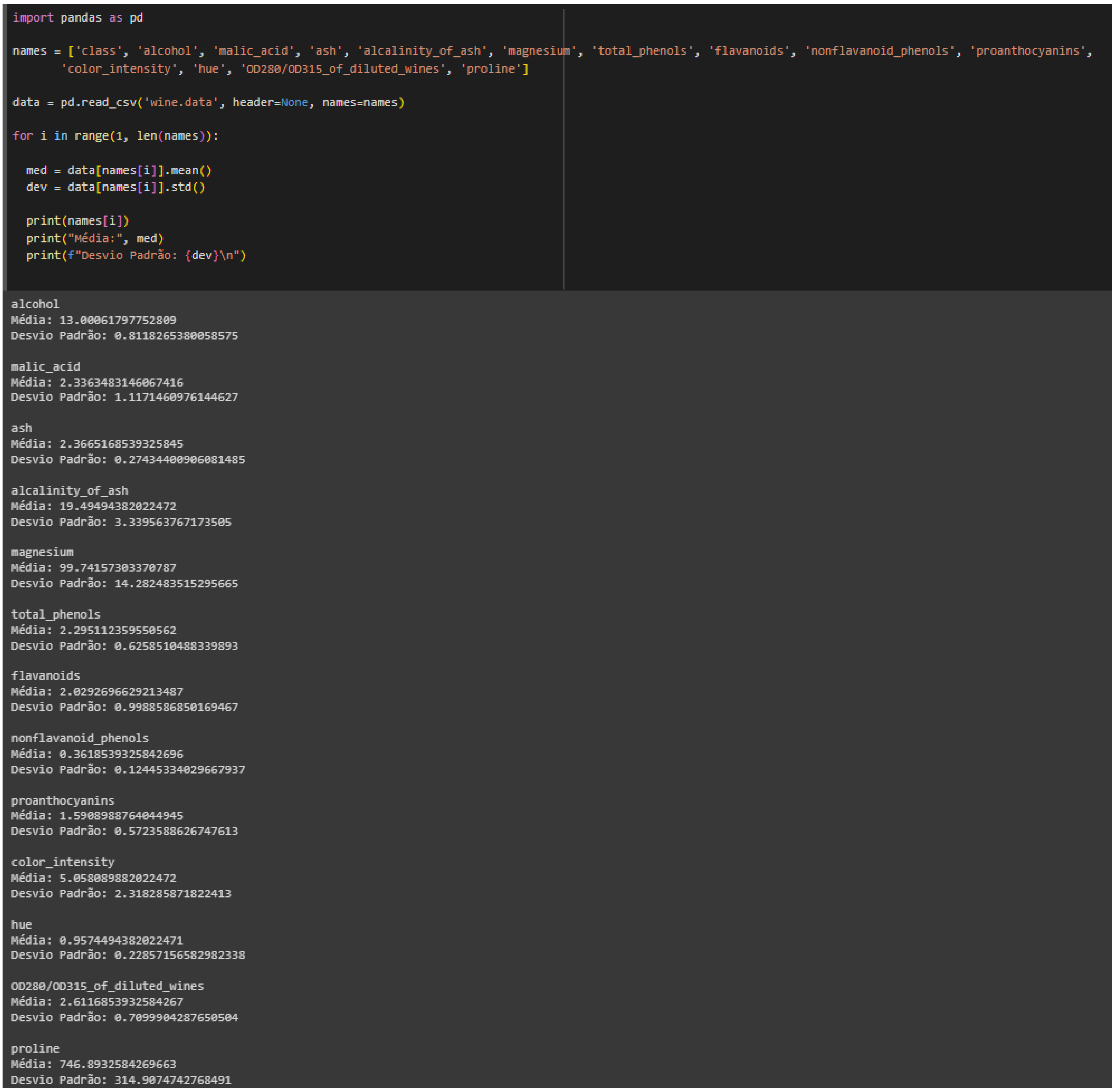
13-nn:



```
Taxa de acerto:  0.7415730337078652
```

Os resultados foram melhores.

b) Calcule a média e o desvio padrão de cada característica e utilize estas informações para justificar a diferença de resultados em relação a questão anterior.



```python
import pandas as pd

names = ['class', 'alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins',
       'color_intensity', 'hue', 'OD280/OD315_of_diluted_wines', 'proline']

data = pd.read_csv('wine.data', header=None, names=names)

for i in range(1, len(names)):

  med = data[names[i]].mean()
  dev = data[names[i]].std()

  print(names[i])
  print("Média:", med)
  print(f"Desvio Padrão: {dev}\n")
```

```
alcohol
Média: 13.000617977528089
Desvio Padrão: 0.8118265380058575

malic_acid
Média: 2.3363483146067416
Desvio Padrão: 1.1171460976144627

ash
Média: 2.3665168539325845
Desvio Padrão: 0.27434400906081485

alcalinity_of_ash
Média: 19.49494382022472
Desvio Padrão: 3.339563767173505

magnesium
Média: 99.74157303370787
Desvio Padrão: 14.282483515295665

total_phenols
Média: 2.295112359550562
Desvio Padrão: 0.6258510488339893

flavanoids
Média: 2.0292696629213487
Desvio Padrão: 0.9988586850169467

nonflavanoid_phenols
Média: 0.3618539325842696
Desvio Padrão: 0.12445334029667937

proanthocyanins
Média: 1.5908988764044945
Desvio Padrão: 0.5723588626747613

color_intensity
Média: 5.058089882022472
Desvio Padrão: 2.318285871822413

hue
Média: 0.9574494382022471
Desvio Padrão: 0.22857156582982338

OD280/OD315_of_diluted_wines
Média: 2.6116853932584267
Desvio Padrão: 0.7099904287650504

proline
Média: 746.8932584269663
Desvio Padrão: 314.9074742768491
```

O atributo proline destoa muito dos outros na média e no desvio padrão, isso pode acontecer por diversos fatores, como outliers ou outras diferenças na base. Essa diferença impacta diretamente em como o k-nn classifica os vinhos da base, então quando retirada a coluna as taxas de acerto aumentam, por isso é importante sempre fazer um pré-processamento dos dados.