```
“””
Simple example of using Spark to compute Kmeans


Walter Stevens  2019

“””


from pyspark.mllib.clustering import KMeans
from numpy import array, random
from math import sqrt
from pyspark import SparkConf, SparkContext
from sklearn.preprocessing import scale

K = 5

# Standard Spark :
conf = SparkConf().setMaster("local").setAppName("SparkKMeans")
sc = SparkContext(conf = conf)

#Create fake income/age clusters for N people in k clusters
def createClusteredData(N, k):
    random.seed(10)
    pointsPerCluster = float(N)/k
    X = []
    for i in range (k):
        incomeCentroid = random.uniform(20000.0, 200000.0)
        ageCentroid = random.uniform(20.0, 70.0)
        for j in range(int(pointsPerCluster)):
            X.append([random.normal(incomeCentroid, 10000.0), random.normal(ageCentroid, 2.0)])
```

```
    X = array(X)
    return X

random.seed(0)

# Create an RDD, load the data; I am normalizing it with scale()
data = sc.parallelize(scale(createClusteredData(100, K)))

# Build the model (cluster the data)
clusters = KMeans.train(data, K, maxIterations=10,
        initializationMode="random")

# Print out the cluster assignments
resultRDD = data.map(lambda point: clusters.predict(point)).cache()
#the cache call is avoid re-computing results

print("Counts by value:")
counts = resultRDD.countByValue()
print(counts)

print("Cluster assignments:")
results = resultRDD.collect()
print(results)



# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = data.map(lambda point: error(point)).reduce(lambda x, y: x + y)
```

```
print("Within Set Sum of Squared Error = " + str(WSSSE))
```