```
'''
Matthew Wintersteen
Jessica Jorgenson
CSCI 347 Project 1
Python Code for Data Analysis
'''
import numpy as np
import pandas as pd
import math
from numpy import genfromtxt
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

'''
A function to compute the mean of a numerical, multidimensional data set
input as a 2-dimensional numpy array
'''
def computeMean(arr):
    mean = np.zeros(arr.shape[1])
    for a in arr:
        mean += a
    mean = mean / arr.shape[0]
    return mean

'''
A function to compute the sample covariance between two attributes that
are input as one-dimensional numpy vectors
'''
def computeCovar(v1, v2):
    v1mean = np.mean(v1)
    v2mean = np.mean(v2)
    n = np.size(v1)

    summ = 0

    for i in range(n):
        summ += (v1[i]-v1mean)*(v2[i]-v2mean)

    cov = summ/(n-1)

    return cov

'''
A function to compute the correlation between two attributes that are input as
one-dimensional numpy vectors
'''
```

```
def computeCorr(v1, v2):
    cov12 = computeCovar(v1,v2)
    cov1 = computeCovar(v1,v1)
    cov2 = computeCovar(v2,v2)

    corr = cov12/math.sqrt(cov1*cov2)
    return corr

'''
A function to range normalize a two-dimensional numpy array
'''
def rangeNorm(arr):
    normArr = arr
    minimum = np.amin(normArr, axis=0)
    maximum = np.amax(normArr, axis=0)

    normArr = normArr.astype('float32')

    for col in range(len(normArr)):
       for row in range(len(normArr[col,:])):
           normArr[col][row] = (normArr[col][row]-minimum[row])/(maximum[row]-
minimum[row])
    return normArr

'''
A function to standard normalize a two-dimensional numpy array
'''
def standardNorm(arr):
    normArr = arr
    std = np.std(normArr, axis=0)
    mean = computeMean(normArr)

    normArr = normArr.astype('float32')

    for col in range(len(normArr)):
       for row in range(len(normArr[col,:])):
           normArr[col][row] = (normArr[col][row]-mean[row])/(std[row])
    return normArr

'''
A function to compute the covariance matrix of a dataset
'''
def computeCovarMatrix(arr):
    n = arr.shape[1]
    covarMatrix = np.zeros([n,n], dtype = float)
    for col in range(n):
        for row in range(n):
            covarMatrix[col][row] = computeCovar(arr[:,col],arr[:,row])

    return covarMatrix

'''
A function to label-encode categorical data
'''
def labelEncode(v):
    encodedV = np.zeros(len(v), dtype = float)
    stringlist = []
    for i in range(len(v)):
        d = v[i].strip()
        if (d not in stringlist):
            stringlist.append(d)
            d = len(stringlist)
        else:
            d = stringlist.index(d)
        encodedV[i] = float(d)
    if (len(stringlist) > 0):
        print("Removed Strings")
    return encodedV

# A function to make a correlation matrix
def computeCorrMatrix(arr):
    n = arr.shape[1]
    corrMatrix = np.zeros([n,n], dtype = float)
    for col in range(n):
        for row in range(n):
            corrMatrix[col][row] = computeCorr(arr[:,col],arr[:,row])

    return corrMatrix

#Tests all the python functions written for Part 2
def testFunc():
    a = np.array([[7,14,33,48,-1],[5,15,34,50,0],[8,17,32,41,1]])
    b = np.array(["x-large","medium","large","medium","small"])
    v1 = np.array([1,2,3,2,4,1,2,1,1])
    v2 = np.array([4,1,3,1,1,0,2,1,3])

    #The equivalent solution with libraries is commented out

    print("Testing compute mean")
    print(computeMean(a))
##      print(np.mean(a, axis=0))
    print("Testing compute covariance")
    print(computeCovar(v1,v2))
##      print(np.cov(v1,v2)[1][0])
```

```
    print("Testing compute correlation")
    print(computeCorr(v1,v2))
##    print(np.corrcoef(v1,v2))
    print("Testing range normalization")
    print(rangeNorm(a))
##    scaler = MinMaxScaler()
##    scaler.fit(a)
##    print(scaler.transform(a))
    print("Testing standard normalization")
    print(standardNorm(a))
    print("Testing compute covariance matrix")
    print(computeCovarMatrix(a))
##    print(np.cov(a.transpose()))
    print("Testing label encoding")
    print(labelEncode(b))


#Driver for Part 3
def main():
    print("Reading input from file")
    df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/autos/
imports-85.data',header=None,names=columns, na_values=['?'])

    #One-hot-encoding all categorical data
    df = pd.get_dummies(df, columns=categorical)

    for i in range(len(df.columns)):
        df.iloc[:, i].fillna(df.iloc[:, i].mean(), inplace=True)

    arr = df.to_numpy()
    colNames = list(df.columns.values)

    #This code generates the multivariate mean and covar matrix
    #and writes the output to a csv
    #commented out to simplify things

    #multivariate mean
    multMean = computeMean(arr)
    f = open("q1.csv", "w")
    for a in range(len(multMean)):
        f.write("{},{}\n".format(colNames[a],multMean[a]))
    f.close()

    #covariance matrix
    matrix = computeCovarMatrix(arr)
    f = open("covarMatrix.csv", "w")
    for i in range(matrix.shape[0]):
```

```
    for j in range(matrix.shape[1]):
        f.write("{}".format(matrix[i][j]))
        if (j != matrix.shape[1]-1):
            f.write(",")
        else:
            f.write("\n")
f.close()
#correlation matrix
matrix = computeCorrMatrix(arr)
f = open("corrMatrix.csv", "w")
for i in range(matrix.shape[0]):
    for j in range(matrix.shape[1]):
        f.write("{}".format(matrix[i][j]))
        if (j != matrix.shape[1]-1):
            f.write(",")
        else:
            f.write("\n")
f.close()

#total variance
matrix = computeCovarMatrix(arr)
totalVar = 0
for i in range(matrix.shape[0]):
    for j in range(matrix.shape[1]):
        if i == j:
            totalVar += matrix[i][j]
print(totalVar)

matrix = computeCovarMatrix(arr)
f = open("sampleCovarMatrix.csv", "w")
for i in range(matrix.shape[0]):
    for j in range(matrix.shape[1]):
        if j == i:
            f.write("{}".format(matrix[i][j]))
        else:
            f.write("0")
        if (j != matrix.shape[1]-1):
            f.write(",")
        else:
            f.write("\n")
f.close()

#select attributes to plot
#commented out to simplify things

plt.figure(1)
plt.scatter(arr[:, 38], arr[:, 1], color = 'red', marker = 'o')
plt.xlabel(colNames[38])
plt.ylabel(colNames[1])

plt.figure(2)
plt.scatter(arr[:, 11], arr[:, 6], color = 'red', marker = 'o')
plt.xlabel(colNames[11])
plt.ylabel(colNames[6])

plt.figure(3)
plt.scatter(arr[:, 6], arr[:, 7], color = 'red', marker = 'o')
plt.xlabel(colNames[6])
plt.ylabel(colNames[7])

plt.figure(4)
plt.scatter(arr[:, 51], arr[:, 7], color = 'red', marker = 'o')
plt.xlabel(colNames[51])
plt.ylabel(colNames[7])

plt.figure(5)
plt.scatter(arr[:, 14], arr[:, 15], color = 'red', marker = 'o')
plt.xlabel(colNames[14])
plt.ylabel(colNames[15])

plt.show()


#range normalize the numerical data

norm = rangeNorm(arr)

normMatrix = computeCovarMatrix(norm)
f = open("q3.csv", "w")
for i in range(16):
    for j in range(16):
        if j != i:
            f.write("{}".format(normMatrix[i][j]))
        else:
            f.write("0")
        if (j != 15):
            f.write(",")
        else:
            f.write("\n")
f.close()

#plot the normalized attribute pairs with the highest covariance
plt.figure(1)
plt.scatter(norm[:, 6], norm[:, 3], color = 'blue', marker = 'x')
plt.xlabel(colNames[6])
plt.ylabel(colNames[3])

plt.figure(2)
plt.scatter(norm[:, 15], norm[:, 6], color = 'blue', marker = 'x')
plt.xlabel(colNames[15])
plt.ylabel(colNames[6])

plt.figure(3)
plt.scatter(norm[:, 14], norm[:, 13], color = 'blue', marker = 'x')
plt.xlabel(colNames[14])
plt.ylabel(colNames[13])

plt.figure(4)
plt.scatter(norm[:, 6], norm[:, 4], color = 'blue', marker = 'x')
plt.xlabel(colNames[6])
plt.ylabel(colNames[4])

plt.figure(5)
plt.scatter(norm[:, 3], norm[:, 2], color = 'blue', marker = 'x')
plt.xlabel(colNames[3])
plt.ylabel(colNames[2])

plt.show()


#Take the standard norm and find the correlation for numerical attributes
norm = standardNorm(arr)

normMatrix = computeCorrMatrix(norm)
f = open("q4.csv", "w")
for i in range(16):
    for j in range(16):
        if j != i:
            f.write("{}".format(normMatrix[i][j]))
        else:
            f.write("0")
        if (j != 15):
            f.write(",")
        else:
            f.write("\n")
f.close()


#These plots are for greatest correlation
plt.figure(1)
plt.scatter(norm[:, 14], norm[:, 13], color = 'green', marker = '+')
plt.xlabel(colNames[14])
plt.ylabel(colNames[13])

plt.figure(2)
plt.scatter(norm[:, 6], norm[:, 3], color = 'green', marker = '+')
plt.xlabel(colNames[6])
plt.ylabel(colNames[3])

plt.figure(3)
plt.scatter(norm[:, 3], norm[:, 2], color = 'green', marker = '+')
plt.xlabel(colNames[3])
plt.ylabel(colNames[2])

plt.figure(4)
plt.scatter(norm[:, 6], norm[:, 4], color = 'green', marker = '+')
plt.xlabel(colNames[6])
plt.ylabel(colNames[4])

plt.figure(5)
plt.scatter(norm[:, 15], norm[:, 7], color = 'green', marker = '+')
plt.xlabel(colNames[15])
plt.ylabel(colNames[7])

plt.show()


#These plots are for least correlation
plt.figure(1)
plt.scatter(norm[:, 13], norm[:, 11], color = 'purple', marker = 'v')
plt.xlabel(colNames[13])
plt.ylabel(colNames[11])

plt.figure(2)
plt.scatter(norm[:, 14], norm[:, 6], color = 'purple', marker = 'v')
plt.xlabel(colNames[14])
plt.ylabel(colNames[6])

plt.figure(3)
plt.scatter(norm[:, 14], norm[:, 11], color = 'purple', marker = 'v')
plt.xlabel(colNames[14])
plt.ylabel(colNames[11])

plt.figure(4)
plt.scatter(norm[:, 13], norm[:, 6], color = 'purple', marker = 'v')
plt.xlabel(colNames[13])
plt.ylabel(colNames[6])
```

```
    plt.figure(5)
    plt.scatter(norm[:, 14], norm[:, 3], color = 'purple', marker = 'v')
    plt.xlabel(colNames[14])
    plt.ylabel(colNames[3])

    plt.show()



columns = ['symboling','normalized-losses','make','fuel-type','aspiration','num-of-doors','body-style','drive-wheels','engine-location','wheel-base','length','width','height','curb-weight','engine-type','num-of-cylinders','engine-size','fuel-system','bore','stroke','compression-ratio','horsepower','peak-rpm','city-mpg','highway-mpg','price']
categorical = ['make','fuel-type','aspiration','num-of-doors','body-style','drive-wheels','engine-location','engine-type','num-of-cylinders','fuel-system']

#TestFunc Runs all the python functions for Part 2
testFunc()
#Main runs the python data analysis needed for Part 3
main()
```