Assignment 1, part 1 of 2

1. Load the CSV file country-income-large.csv, which includes both numerical and categorical attributes. Replace NaN values with the mean of the corresponding variable. Display a scatter plot of the resulting dataset, using the numerical variables only, color-coded according to the "Region" column. You are allowed (and encouraged) to seek existing functions for these purposes (e.g., in Pandas).

CODE:

```
import pandas as pd
import numpy as np
import scipy.stats as stats
df = pd.read_csv('country-income-large.csv')
df.columns = ['Sample code', 'Region', 'Age', 'Income', 'Online Shopper']
df = df.drop(['Sample code'],axis=1)
print('Number of instances = %d' % (df.shape[0]))
print('Number of attributes = %d' % (df.shape[1]))
df.head()
```

OUTPUT:

```
Number of instances = 60
Number of attributes = 4

   Region   Age   Income Online Shopper
0   India  49.0  86400.0             No
1  Brazil  32.0  57600.0            Yes
2     USA  35.0  64800.0             No
3  Brazil  43.0  73200.0             No
4     USA  45.0      NaN            Yes
```

CODE:

```
df_bn = df[['Region', 'Age', 'Income', 'Online Shopper']]

print('Before replacing missing values:')
print(df_bn[4:7])
df_bn = df_bn.fillna(df_bn.mean())

print('\nAfter replacing missing values:')
print(df_bn[4:7])
```

OUTPUT:

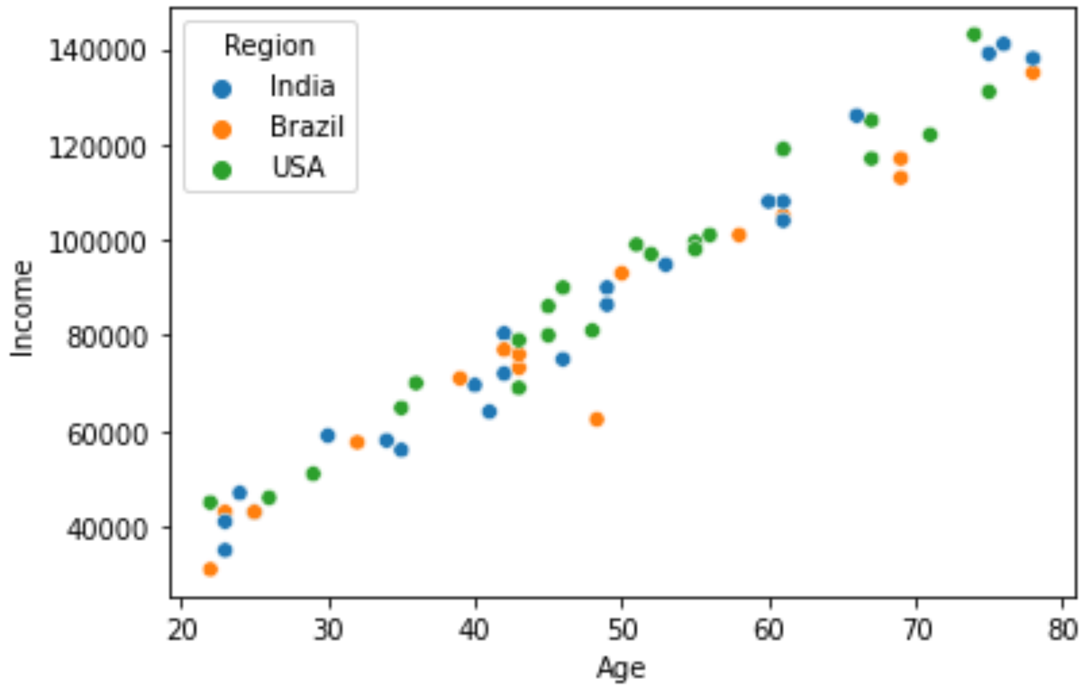
```
Before replacing missing values:
   Region   Age   Income Online Shopper
4     USA  45.0      NaN            Yes
5   India  40.0  69600.0            Yes
6  Brazil   NaN  62400.0             No
```

```
After replacing missing values:
   Region        Age        Income Online Shopper
4     USA  45.000000  86098.305085            Yes
5   India  40.000000  69600.000000            Yes
6  Brazil  48.305085  62400.000000             No
```

As we can see, the NaN values for the attributes "Age" and "Income" have been replaced with the mean of the corresponding variable. As there are no NaN values for the categorical attributes in the csv file, the replacing is performed only on the numerical attributes.

CODE:

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.scatterplot(x='Age', y='Income', hue='Region', data=df_bn)
plt.show()
```



The figure above displays a scatter plot of the resulting dataset, color-coded according to the “Region” column using the numerical attributes “Age” and “Income”.

2. Apply the following binning techniques on the previously cleaned data, assuming 5 bins in each case:
    a. Equal-frequency binning

    b. Equal-width binning
    Report the results.

CODE:

```
bins_age = pd.qcut(df_bn['Age'], 5)
bins_age.value_counts(sort=False)
```

OUTPUT:

```
(21.999, 33.6]    12
(33.6, 43.0]      14
(43.0, 51.4]      10
(51.4, 62.0]      12
(62.0, 78.0]      12
Name: Age, dtype: int64
```

The output above shows the Equal-frequency binning of the “Age” attribute, since it is a continuous valued attribute.

CODE:

```
bins_income = pd.qcut(df_bn['Income'], 5)
bins_income.value_counts(sort=False)
```

OUTPUT:

```
(30999.999, 57920.0]    12
(57920.0, 75600.0]      12
(75600.0, 95680.0]      12
(95680.0, 117000.0]     13
(117000.0, 143000.0]    11
Name: Income, dtype: int64
```

The output above shows the Equal-frequency binning of the “Income” attribute, since it is a continuous valued attribute.

CODE:

```
bins_age = pd.cut(df_bn['Age'], 5)
bins_age.value_counts(sort=False)
```

OUTPUT:

```
(21.944, 33.2]    12
(33.2, 44.4]      14
(44.4, 55.6]      14
(55.6, 66.8]       9
(66.8, 78.0]      11
Name: Age, dtype: int64
```

The output above shows the Equal-width binning of the “Age” attribute, since it is a continuous valued attribute.

CODE:

```
bins_income = pd.cut(df_bn['Income'], 5)
bins_income.value_counts(sort=False)
```

OUTPUT:

```
(30888.0, 53400.0]      10
(53400.0, 75800.0]      14
(75800.0, 98200.0]      14
(98200.0, 120600.0]     13
(120600.0, 143000.0]     9
Name: Income, dtype: int64
```

The output above shows the Equal-width binning of the “Income” attribute, since it is a continuous valued attribute.

3. Compute the Pearson correlation coefficient of the numerical variables. Would you say that the variables are strongly correlated? Did you need to compute the correlation coefficient to reach that conclusion?

CODE:

```
pearson = np.corrcoef(df_bn['Age'], df_bn['Income'])
print(pearson)
```

OUTPUT:

```
[[1.         0.98095182]
 [0.98095182 1.        ]]
```

CODE:

```
from scipy.stats import pearsonr
corr, _ = pearsonr(df_bn['Age'], df_bn['Income'])
print('Pearsons correlation: %.3f' % corr)
```

OUTPUT:

```
Pearsons correlation: 0.981
```

The Pearson correlation coefficient of the numerical attributes "Age" and "Income" is 0.981. Thus, we can conclude that they are positively correlated and have high level of correlation which is close to 1. We didn't need to compute the correlation coefficient to reach that conclusion because when we review the generated scatter plot, we can see an increasing trend and gives us the same conclusion.

4. Use the Chi-squared test to determine whether the categorical attributes are correlated.

CODE:

```
contigency= pd.crosstab(df_bn['Online Shopper'], df_bn['Region'],
margins=True, margins_name="Total")
contigency
```

OUTPUT:

```
Region          Brazil  India  USA  Total
Online Shopper
No                   9     12   15     36
Yes                  6     11    7     24
Total               15     23   22     60
```

CODE:

```
alpha = 0.01
chi_square = 0
rows = df_bn['Online Shopper'].unique()
columns = df_bn['Region'].unique()
for i in columns:
    for j in rows:
        O = contigency[i][j]
        E = contigency[i]['Total'] * contigency['Total'][j] /
contigency['Total']['Total']
        chi_square += (O-E)**2/E
critical_value = stats.chi2.ppf(1-alpha, (len(rows)-1)*(len(columns)-1))
conclusion = "Failed to reject the hypothesis."
if chi_square > critical_value:
    conclusion = "Hypothesis is rejected."

print("chisquare-score is:", chi_square, " and critical value is:",
critical_value)
print(conclusion)
```

OUTPUT:

```
chisquare-score is: 1.200592885375495  and critical value is:
9.21034037197618
Failed to reject the hypothesis.
```

In the chi-squared test the value for the chi-square needed to reject the hypothesis at the 0.01 significance level is 9.21 approximately. Since the value generated for the chi-square is approximately 1.20, the hypothesis cannot be rejected. Thus, we have enough evidence that there is no association between attributes "Online Shopper" and "Region", at 1% significance level.

5. Take the pre-processed breast cancer dataset from subsection 1.1 of this notebook (where we replaced any missing values with their median). Compute the principal components and report the variance explained by each of them (remove the "Class" column before doing PCA). Show the scatter plot of all samples along the first two principal components, color-coded according to the "Class" column. Ensure that your data is normalized by z-scores prior to performing PCA. Do you think PCA is useful as a preprocessing step for classification in this case?

CODE:

```
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data', header=None)
data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape',
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin',
                'Normal Nucleoli', 'Mitoses','Class']

data = data.drop(['Sample code'],axis=1)
print('Number of instances = %d' % (data.shape[0]))
print('Number of attributes = %d' % (data.shape[1]))
data.head()
```

OUTPUT:

```
Number of instances = 699
Number of attributes = 10

   Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape  \
0                5                        1                         1
1                5                        4                         4
2                3                        1                         1
3                6                        8                         8
4                4                        1                         1

   Marginal Adhesion  Single Epithelial Cell Size Bare Nuclei  \
0                  1                            2           1
1                  5                            7          10
2                  1                            2           2
3                  1                            3           4
4                  3                            2           1

   Bland Chromatin  Normal Nucleoli  Mitoses  Class
0                3                1        1      2
1                3                2        1      2
2                3                1        1      2
3                3                7        1      2
4                3                1        1      2
```

CODE:

```
import numpy as np

data = data.replace('?',np.NaN)

print('Number of instances = %d' % (data.shape[0]))
print('Number of attributes = %d' % (data.shape[1]))

print('Number of missing values:')
for col in data.columns:
    print('\t%s: %d' % (col,data[col].isna().sum()))
```

OUTPUT:

```
Number of instances = 699
Number of attributes = 10
Number of missing values:
      Clump Thickness: 0
      Uniformity of Cell Size: 0
      Uniformity of Cell Shape: 0
      Marginal Adhesion: 0
      Single Epithelial Cell Size: 0
      Bare Nuclei: 16
      Bland Chromatin: 0
      Normal Nucleoli: 0
      Mitoses: 0
      Class: 0
```

CODE:

```
data_bn = data['Bare Nuclei']
print('Before replacing missing values:')
print(data_bn[20:25])
data_bn = data_bn.fillna(data_bn.median())
data_bn = pd.to_numeric(data_bn)

print('\nAfter replacing missing values:')
print(data_bn[20:25])

data['Bare Nuclei'] = data_bn
```

OUTPUT:

```
Before replacing missing values:
20     10
21      7
22      1
23    NaN
24      1
Name: Bare Nuclei, dtype: object
```

```
After replacing missing values:
20    10.0
21     7.0
22     1.0
23     1.0
24     1.0
Name: Bare Nuclei, dtype: float64
```

CODE:

```
dataframe = data.drop(['Class'],axis=1)

Z = (dataframe-dataframe.mean())/dataframe.std()
Z[20:25]
```

OUTPUT:

```
    Clump Thickness  Uniformity of Cell Size  Uniformity of Cell Shape  \
20         0.917080                -0.044070                 -0.406284   
21         1.982519                 0.611354                  0.603167   
22        -0.503505                -0.699494                 -0.742767   
23         1.272227                 0.283642                  0.603167   
24        -1.213798                -0.699494                 -0.742767   

    Marginal Adhesion  Single Epithelial Cell Size  Bare Nuclei  \
20           2.519152                     0.805662     1.798376   
21           0.067638                     1.257272     0.970088   
22          -0.632794                    -0.549168    -0.686488   
23          -0.632794                    -0.549168    -0.686488   
24          -0.632794                    -0.549168    -0.686488   

    Bland Chromatin  Normal Nucleoli   Mitoses  
20         0.640688         0.371049  1.405526  
21         1.460910         2.335921 -0.343666  
22        -0.589645        -0.611387 -0.343666  
23         1.460910         0.043570 -0.343666  
24        -0.179534        -0.611387 -0.343666  
```

The data is now normalized by Z-scores.

CODE:

```
from sklearn.decomposition import PCA

numComponents = 9
pca = PCA(n_components=numComponents)
pca.fit(Z)

projected = pca.transform(Z)
projected = pd.DataFrame(projected,columns=['pc1', 'pc2', 'pc3', 'pc4',
'pc5', 'pc6', 'pc7', 'pc8', 'pc9'])
projected
```

OUTPUT:

```
          pc1       pc2       pc3       pc4       pc5       pc6       pc7  \
0   -1.455178 -0.110132 -0.574027 -0.019391 -0.151859  0.074616  0.325780
1    1.465230 -0.544504  0.282835 -0.659808  1.691591 -0.362340 -1.042467
2   -1.578181 -0.074800  0.037386 -0.106700 -0.065565 -0.275505  0.212995
3    1.504170 -0.558454 -0.612546  1.440129 -0.443261 -0.092396 -0.260809
4   -1.329599 -0.089592  0.027382 -0.317487 -0.150569  0.455792  0.265936
..        ...       ...       ...       ...       ...       ...       ...
694 -1.710025  0.187885 -0.074842  0.067632  0.452009 -0.033541 -0.421842
695 -2.061560  0.234057  0.182660  0.080331  0.134090  0.045552 -0.224844
696  3.822621 -0.180336  0.657078  2.510376 -0.304386 -0.296236  0.656572
697  2.267858 -1.112638  0.989670  0.721586 -1.024702 -0.480576  1.651215
698  2.662547 -1.196385  1.076480  0.395128 -0.209106 -0.370235  1.850259

          pc8       pc9
0    0.432258 -0.002052
1    0.361983  0.019424
2    0.232770  0.017212
3   -1.593659  0.186147
4    0.429853 -0.035275
..        ...       ...
694  0.100158  0.034800
695 -0.160412  0.044687
696 -0.540553 -0.061953
697 -0.096344  0.408816
698 -0.328122 -0.078085

[699 rows x 9 columns]
```

CODE:

```
pca.explained_variance_ratio_
```

OUTPUT:

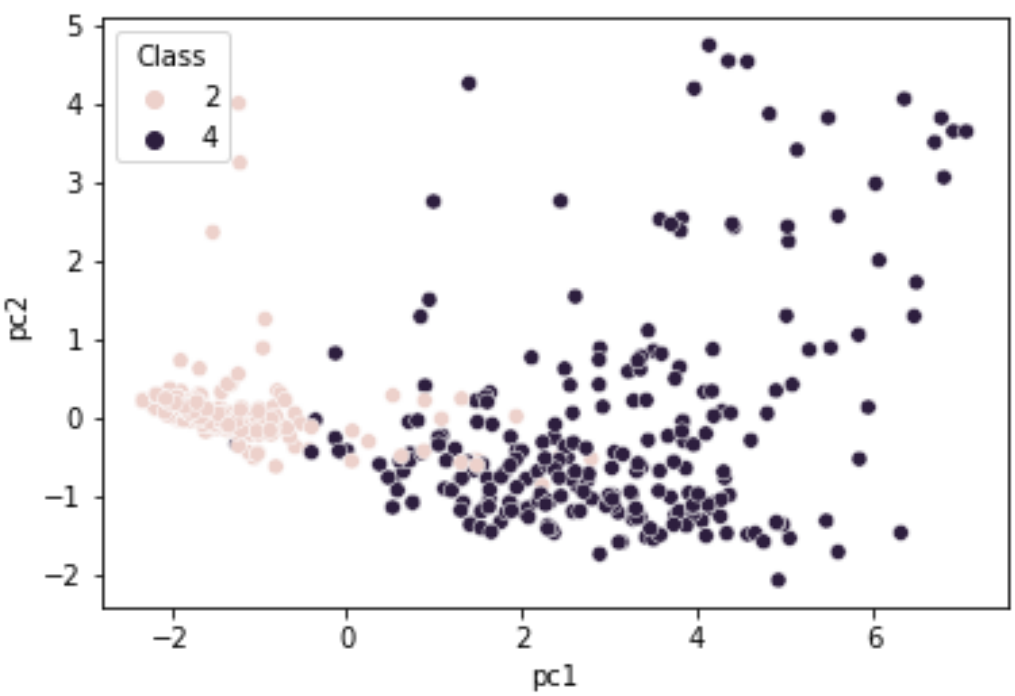
```
array([0.65445704, 0.0860859 , 0.05986995, 0.0517487 , 0.04227474,
       0.03373566, 0.03285235, 0.02910658, 0.00986908])
```

The output above shows the variance explained by the principal components.

CODE:

```
sns.scatterplot(x=projected['pc1'], y=projected['pc2'], hue=data['Class'])
plt.show()
```

OUTPUT:



The figure above displays the scatter plot of all samples along the first two principal components, color-coded according to the "Class" column. The data was normalised by Z-scores prior to performing PCA.

In this case, the attributes are possibly correlated and thus, PCA is useful as a pre-processing step for classification as it is converts these attributes into linearly uncorrelated attributes.

Assignment 1, part 2 of 2

1. Take the graduation rate dataset from the beginning of the notebook and change the 'parental level of education' attribute so that it is ordered, just as we did earlier. Split the dataset into two groups: those with a level of at least high school, and those below.
    a. Show a quantile-quantile plot of the 'parental income' for these two groups and report the results. What can you conclude about the relationship between these two attributes, just by looking at this plot?
    b. Now show a quantile-quantile plot, but of the 'years to graduate' variable, instead of 'parental income'. What can you infer?

CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

df = pd.read_csv('graduation_rate.csv')

print('Dataset (head and tail):')
display(df)

print('\nParental levels of education:')
print(df['parental level of education'].unique())
```

OUTPUT:

```
Dataset (head and tail):

     ACT composite score  SAT total score parental level of education  \
0                     30             2206             master's degree
1                     26             1953                some college
2                     28             2115            some high school
3                     33             2110            some high school
4                     30             2168           bachelor's degree
..                   ...              ...                         ...
995                   30             1967                 high school
996                   28             2066                some college
997                   27             1971                 high school
998                   30             2057                some college
999                   29             2054            some high school
```

```
     parental income  high school gpa  college gpa  years to graduate
0              94873              4.0          3.8                  3
1              42767              3.6          2.7                  9
2              46316              4.0          3.3                  5
3              52370              4.0          3.5                  4
4              92665              4.0          3.6                  4
..               ...              ...          ...                ...
995            49002              3.8          3.5                  6
996            83438              3.9          3.5                  4
997            68577              3.6          3.7                  5
998            56876              3.8          3.6                  3
999            40068              3.9          3.3                  5

[1000 rows x 7 columns]
```

CODE:

```
Parental levels of education:
["master's degree" 'some college' 'some high school' "bachelor's degree"
 "associate's degree" 'high school']

education_order = ['some high school', 'high school', 'some college',
"associate's degree", "bachelor's degree", "master's degree"]

df['parental level of education'] = pd.Categorical(df['parental level of
education'],
                                                    ordered=True,

categories=education_order)

display(df['parental level of education'])
```

OUTPUT:

```
0         master's degree
1            some college
2        some high school
3        some high school
4       bachelor's degree
              ...
995           high school
996          some college
997           high school
998          some college
999      some high school
Name: parental level of education, Length: 1000, dtype: category
Categories (6, object): ['some high school' < 'high school' < 'some college'
< 'associate's degree' <
                         'bachelor's degree' < 'master's degree']
```

Changing the 'parental level of education' attribute so that it is ordered.

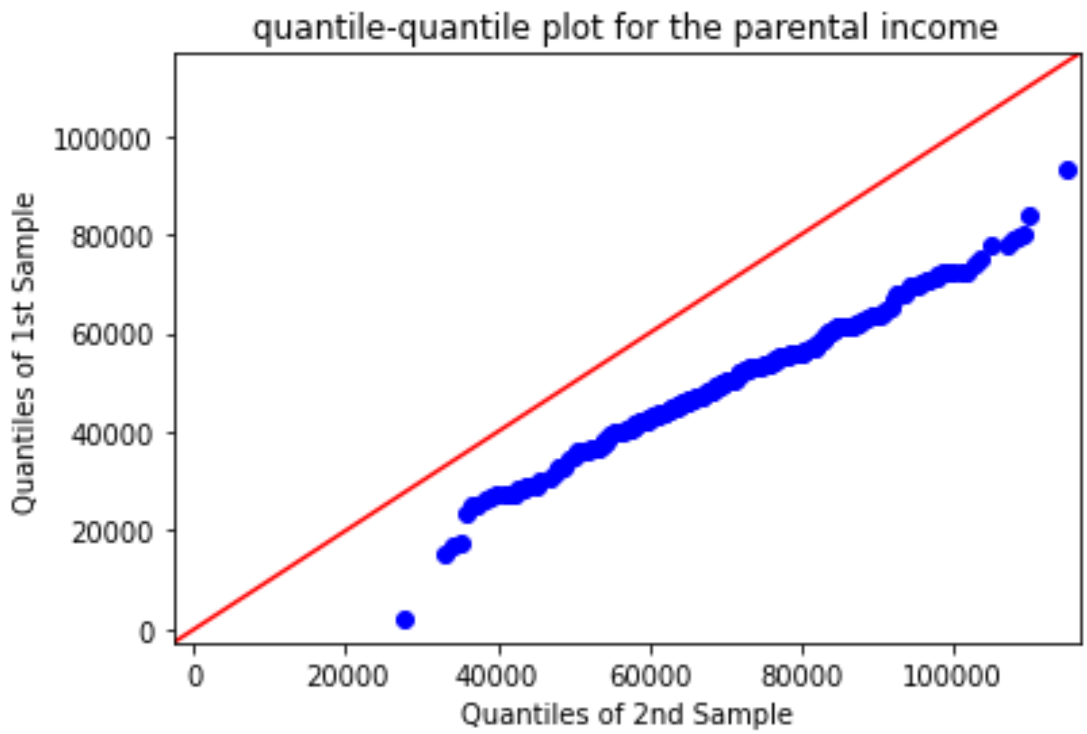
CODE:

```
atleast = df[df['parental level of education'] >= 'high school']
below = df[df['parental level of education'] < 'high school']
```

Splitting the dataset into two groups: those with a level of at least high school, and those below.

```
sm.qqplot_2samples(below['parental income'], atleast['parental income'],
line='45')
plt.title('quantile-quantile plot for the parental income')
plt.show()
```

OUTPUT:



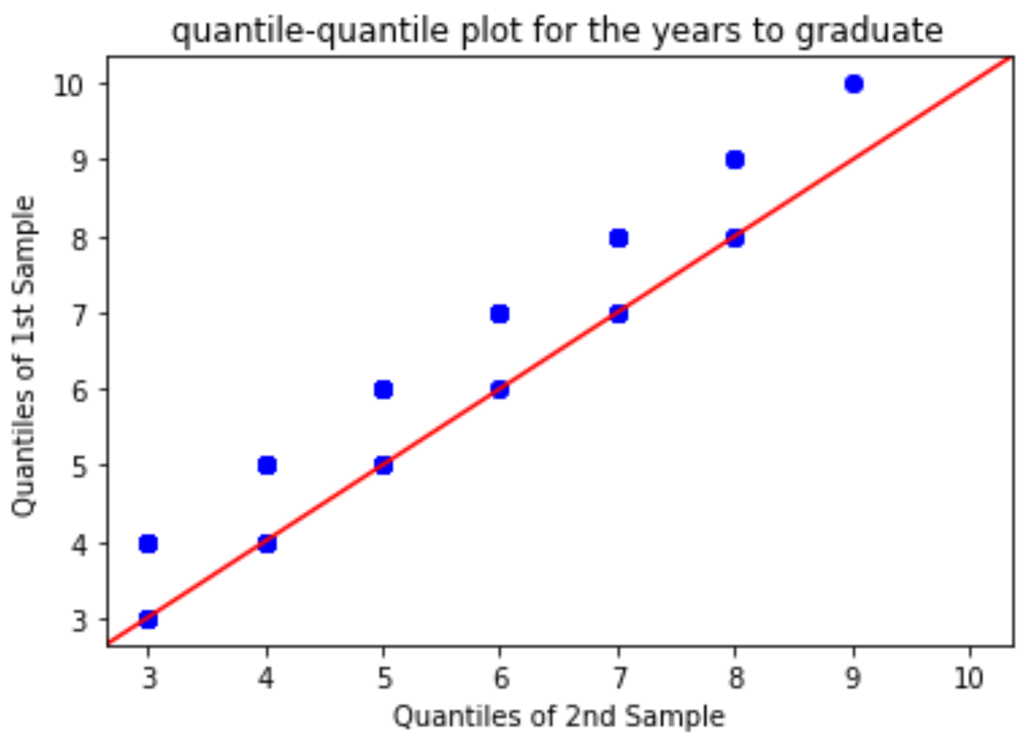
The figure above shows the quantile-quantile plot of the 'parental income' attribute for the two groups.

The q-q plot shows that these 2 attributes do not appear to have come from populations with a common distribution. The quantities of the 2nd sample is significantly higher than the corresponding 1st sample.

CODE:

```
sm.qqplot_2samples(below['years to graduate'], atleast['years to graduate'], 
line='45')
plt.title('quantile-quantile plot for the years to graduate')
plt.show()
```

OUTPUT:



The figure above shows the quantile-quantile plot of the 'years to graduate' attribute for the two groups.

The q-q plot shows that these 2 attributes have some values that appear to have come from populations with a common distribution and some values that do not where the quantities of the 1st sample is significantly higher than the corresponding 2nd sample.

2. Load the wine dataset (see the snippet at the bottom). Note that it contains observations of three classes. For each class, report the following univariate summaries of the 'alcohol' attribute: count, mean, min, max, quartiles and standard deviation.

CODE:

```
from sklearn import datasets
data=datasets.load_wine()
dfw = pd.DataFrame(data= np.c_[data['data'], data['target']], columns=
data['feature_names'] + ['target'])
display(dfw)
```

OUTPUT:

```
     alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols
\
0      14.23        1.71  2.43               15.6      127.0           2.80
1      13.20        1.78  2.14               11.2      100.0           2.65
2      13.16        2.36  2.67               18.6      101.0           2.80
3      14.37        1.95  2.50               16.8      113.0           3.85
4      13.24        2.59  2.87               21.0      118.0           2.80
..       ...         ...   ...                ...        ...            ...
173    13.71        5.65  2.45               20.5       95.0           1.68
174    13.40        3.91  2.48               23.0      102.0           1.80
175    13.27        4.28  2.26               20.0      120.0           1.59
176    13.17        2.59  2.37               20.0      120.0           1.65
177    14.13        4.10  2.74               24.5       96.0           2.05

     flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue
\
0          3.06                  0.28             2.29             5.64  1.04
1          2.76                  0.26             1.28             4.38  1.05
2          3.24                  0.30             2.81             5.68  1.03
3          3.49                  0.24             2.18             7.80  0.86
4          2.69                  0.39             1.82             4.32  1.04
..          ...                   ...              ...              ...   ...
173        0.61                  0.52             1.06             7.70  0.64
174        0.75                  0.43             1.41             7.30  0.70
175        0.69                  0.43             1.35            10.20  0.59
176        0.68                  0.53             1.46             9.30  0.60
177        0.76                  0.56             1.35             9.20  0.61

     od280/od315_of_diluted_wines  proline  target
0                            3.92   1065.0     0.0
1                            3.40   1050.0     0.0
2                            3.17   1185.0     0.0
3                            3.45   1480.0     0.0
4                            2.93    735.0     0.0
..                            ...      ...     ...
173                          1.74    740.0     2.0
174                          1.56    750.0     2.0
175                          1.56    835.0     2.0
176                          1.62    840.0     2.0
177                          1.60    560.0     2.0
```

```
[178 rows x 14 columns]
```

CODE:

```
print('Univariate summaries: alcohol')
display(dfw.groupby('target')['alcohol'].describe())
```

OUTPUT:

```
Univariate summaries: alcohol

        count       mean       std    min     25%     50%     75%    max
target
0.0      59.0  13.744746  0.462125  12.85  13.400  13.750  14.100  14.83
1.0      71.0  12.278732  0.537964  11.03  11.915  12.290  12.515  13.86
2.0      48.0  13.153750  0.530241  12.20  12.805  13.165  13.505  14.34
```

The output above shows the univariate summaries of the 'alcohol' attribute: count, mean, min, max, quartiles and standard deviation, for the three classes in the dataset.

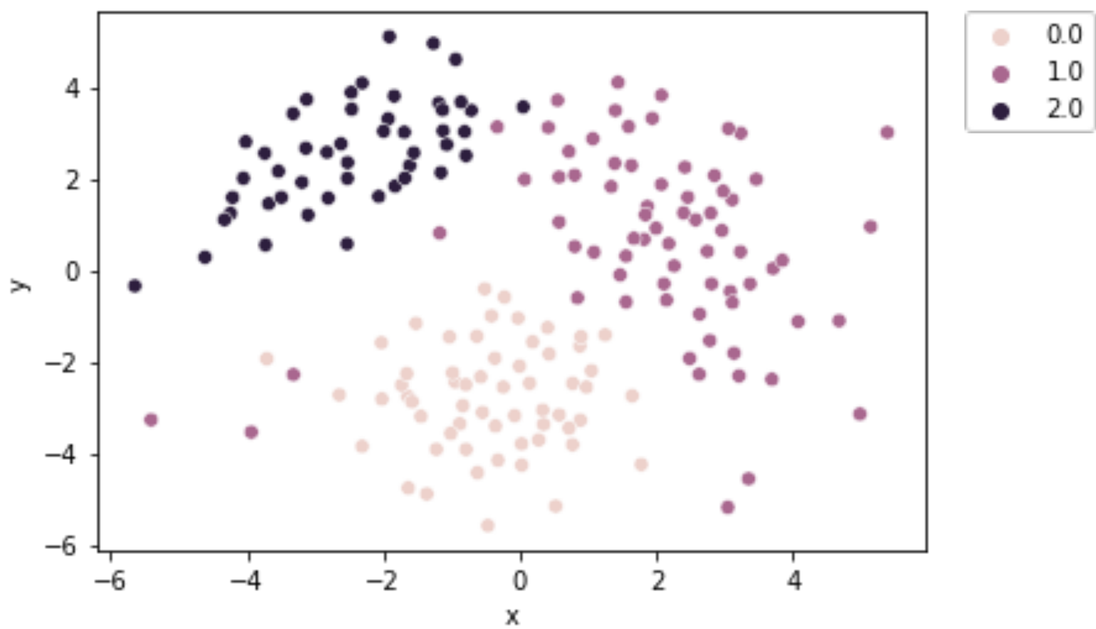
3. Drop the 'target' attribute from the wine dataset and draw a 2D scatter plot using MDS. Use the code from section 1.10 in this notebook. Is the plot satisfactory? If yes, what steps did you take to improve the quality of the representation? If not, try to run preprocessing steps to improve the quality of the plot.

CODE:

```
from sklearn.preprocessing import StandardScaler

df_sorted = dfw.sort_values(by='target', ascending=True)
target_sorted = df_sorted['target']

X = df_sorted.drop(columns='target').to_numpy()
scaler = StandardScaler()
X = scaler.fit_transform(X)

from sklearn.manifold import MDS

embedding = MDS(n_components=2)

Xp = embedding.fit_transform(X)
df_projection = pd.DataFrame({'x': Xp[:, 0], 'y': Xp[:, 1],
                              'target': target_sorted})
```

```python
sns.scatterplot(x='x', y='y', hue='target', data=df_projection)

plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()
```

OUTPUT:



The figure above shows a 2D scatter plot using MDS after dropping the 'target' attribute.

Yes, the plot is satisfactory. The steps taken to improve the quality of the representation are dropping the target attribute and performing standard scaler which removes the mean and scales the data to the unit variance; computing an MDS-based 2D embedding; displaying a scatter plot using the resulting points and color-coded according to the target attribute that had been sorted and stored inside a variable before dropping.

4. Based on a scatter matrix of the wine dataset, select three attributes that you think would be useful for building a classifier. Remember to justify your choice. Use the selected features to compute an MDS-based 2D embedding and show the resulting points in a scatter plot. Is the result comparable to the one obtained in the previous question? What are the advantages and/or disadvantages of this approach?

CODE:

```
sns.pairplot(dfw, hue='target', diag_kind='hist')

plt.show()
```

OUTPUT:

CODE:

```
df_sorted_new = dfw.sort_values(by='target', ascending=True)
target_sorted_new = df_sorted_new['target']
```

```
Xnew = df_sorted_new.drop(columns=['target', 'od280/od315_of_diluted_wines',
'hue', 'color_intensity', 'proanthocyanins', 'nonflavanoid_phenols',
'total_phenols', 'magnesium', 'alcalinity_of_ash', 'ash',
'malic_acid']).to_numpy()
Xnew = scaler.fit_transform(Xnew)
```

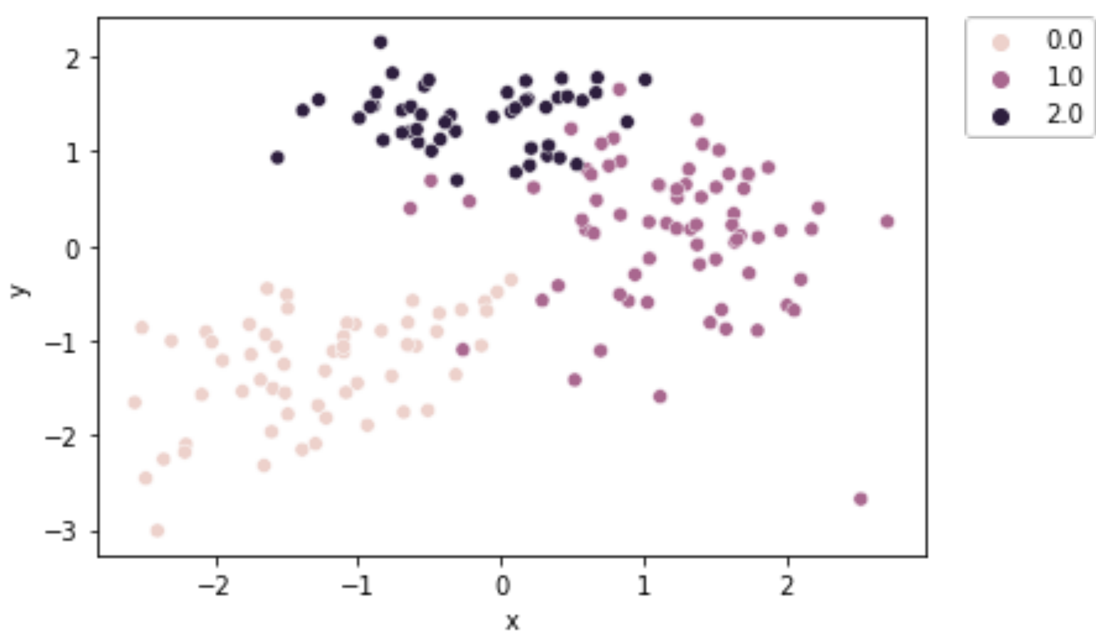
Dropping attrbutes other than the three selected ones useful for building a classifier.

Selecting the attributes 'proline', 'flavanoids', 'alcohol' , since the scatter matrix shows that the three classes are somewhat separable and thus, can be useful for classification.

CODE:

```
Xp_new = embedding.fit_transform(Xnew)
df_projection_new = pd.DataFrame({'x': Xp_new[:, 0], 'y': Xp_new[:, 1],
                              'target': target_sorted_new})

sns.scatterplot(x='x', y='y', hue='target', data=df_projection_new)

plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

plt.show()
```

OUTPUT:



The result is somewhat comparable to the plot obtained for the previous question.

The advantages and/or disadvantages of this approach is that: it reduces the training time due to less data but when the number of variables is large the computation time might be significant; it reduces overfitting due to less redundant data but when the number of observations is insufficient the risk of overfitting increases.

5. Consider the `mystery.csv` dataset. Compute its first principal direction (or principal component) using PCA (that is, the leading eigenvector of the covariance matrix). Reshape this vector to dimension 60x79 and plot it as a heatmap. What do you see? What type of data do you think this dataset contains?

CODE:

```
dfm = pd.read_csv('mystery.csv', delimiter = ' ', header = None)
display(dfm)
```

OUTPUT:

```
       0      1      2      3      4     5      6      7      8      9     ...
\
0    0.910  0.910  0.910  0.910  0.91  0.910  0.910  0.910  0.910  0.910  ...
1    0.910  0.910  0.910  0.910  0.91  0.910  0.910  0.910  0.910  0.910  ...
2    0.910  0.910  0.910  0.910  0.91  0.910  0.910  0.910  0.910  0.910  ...
3    0.910  0.910  0.910  0.910  0.91  0.910  0.910  0.910  0.910  0.910  ...
4    0.906  0.910  0.910  0.910  0.91  0.910  0.906  0.910  0.910  0.910  ...
..     ...    ...    ...    ...   ...    ...    ...    ...    ...    ...  ...
161  0.729  0.816  0.788  0.773  0.78  0.843  0.827  0.878  0.843  0.788  ...
162  0.910  0.910  0.910  0.910  0.91  0.910  0.910  0.910  0.910  0.910  ...
163  0.910  0.910  0.910  0.910  0.91  0.910  0.910  0.910  0.910  0.910  ...
164  0.910  0.910  0.910  0.910  0.91  0.910  0.910  0.910  0.910  0.910  ...
165  0.910  0.910  0.910  0.910  0.91  0.910  0.910  0.910  0.910  0.910  ...

      4730   4731   4732   4733   4734   4735   4736   4737   4738   4739
0    0.859  0.863  0.863  0.859  0.859  0.855  0.859  0.855  0.863  0.863
1    0.843  0.855  0.839  0.855  0.851  0.863  0.855  0.859  0.863  0.863
2    0.843  0.855  0.839  0.855  0.851  0.863  0.855  0.859  0.863  0.863
3    0.839  0.855  0.843  0.855  0.851  0.863  0.855  0.859  0.859  0.859
4    0.188  0.373  0.608  0.675  0.675  0.643  0.686  0.647  0.655  0.667
..     ...    ...    ...    ...    ...    ...    ...    ...    ...    ...
```

```
161  0.831  0.851  0.847  0.847  0.859  0.831  0.847  0.843  0.855  0.855
162  0.820  0.851  0.847  0.827  0.820  0.839  0.839  0.839  0.859  0.820
163  0.749  0.729  0.749  0.827  0.827  0.843  0.812  0.820  0.812  0.831
164  0.706  0.694  0.718  0.816  0.808  0.839  0.788  0.800  0.780  0.800
165  0.761  0.745  0.765  0.835  0.827  0.851  0.827  0.835  0.831  0.851

[166 rows x 4740 columns]
```

CODE:

```
from sklearn.decomposition import PCA

numComponents = 1
pca = PCA(n_components=numComponents)
pca.fit(dfm)

projected = pca.transform(dfm)
projected = pd.DataFrame(projected,columns=['pc1'])
projected
```

OUTPUT:

```
           pc1
0    -7.402291
1    -3.200268
2    -3.200268
3    -2.957951
4   -15.833543

..         ...
161  17.685175
162  -1.189839
163   3.900968
164   3.494921
165  -2.350328

[166 rows x 1 columns]
```

CODE:

```
first_element = pca.components_[0]
display(first_element)
vecs = np.reshape(first_element,[60,79])
display(vecs)
```

OUTPUT:

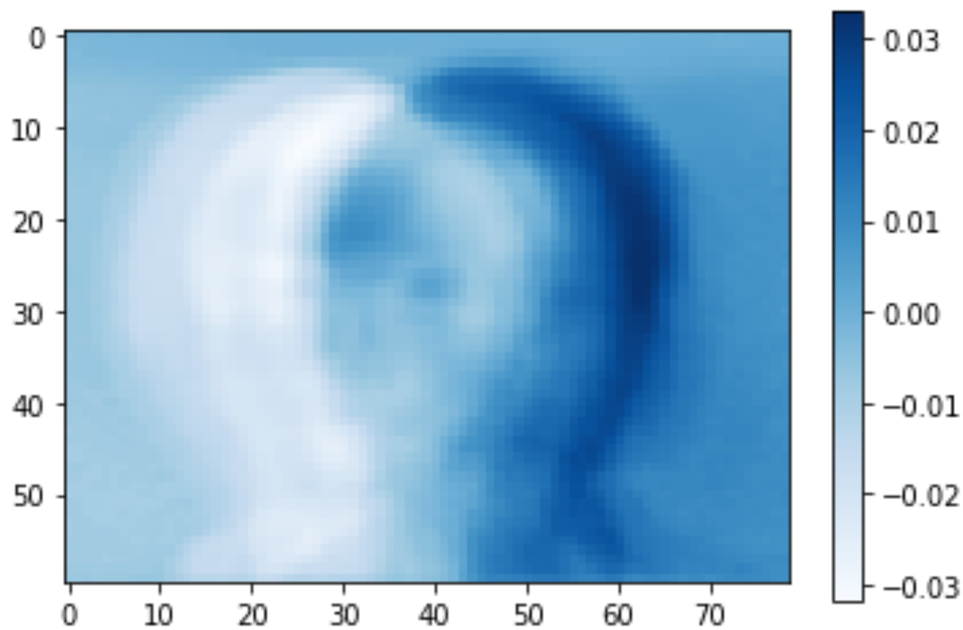
```
array([-0.00225291, -0.00180377, -0.00175121, ...,  0.01003465,
        0.00942367,  0.00921256])

array([[-0.00225291, -0.00180377, -0.00175121, ...,  0.00020022,
         0.0001019 ,  0.00027446],
       [-0.0022105 , -0.00241894, -0.00237282, ...,  0.00027768,
         0.00059927,  0.00041418],
       [-0.00281167, -0.00307247, -0.00235293, ...,  0.00074637,
         0.00080696,  0.00081662],
       ...,
       [-0.00859255, -0.00842063, -0.00840406, ...,  0.01006397,
         0.01004524,  0.01014138],
       [-0.007544  , -0.00753034, -0.00771311, ...,  0.01011198,
         0.00964703,  0.00970361],
       [-0.00734457, -0.0074662 , -0.00691304, ...,  0.01003465,
         0.00942367,  0.00921256]])
```

Computing its first principal direction (or principal component) using PCA (that is, the leading eigenvector of the covariance matrix) and reshaping this vector to dimension 60x79.

CODE:

```
plt.imshow(vecs, cmap= 'Blues', aspect='equal', interpolation='nearest')
plt.colorbar()

plt.show()
```

OUTPUT :



Plotting it as a heatmap.

We can see a blurred face of a man. Hence, we can conclude that this dataset contains an image data and the data type is composite.