Assignment 1

Part 1

1. Consider the following sales data: [2, 15, 20, 5, 1, 4, 7, 9, 10, 3, 14, 8]. Apply the following binning techniques on the data: ¶

A. Equal-frequency binning

In [288]:

```
data1 = [2, 15, 20, 5, 1, 4, 7, 9, 10, 3, 14, 8]
```

In [289]:

```
maxdata1 = max(data1);
mindata1 = min(data1);
print (maxdata1);
print (mindata1);
```

```
20
1
```

In [311]:

```
import pandas as pd
import math
import numpy as np

df = pd.DataFrame()

df['Values']  = data1

nInstances = df.shape[0]

print (nInstances)

df
```

12

Out[311]:

	Values
0	2
1	15
2	20
3	5
4	1
5	4
6	7
7	9
8	10
9	3
10	14
11	8

In [312]:

```
numberBins = math.ceil(math.sqrt(nInstances))
print (numberBins)
```

4

In [313]:

```
df3 = df

df3['Bin qcut'] = pd.qcut(df3['Values'], numberBins)

df3
```

Out[313]:

	Values	Bin qcut
0	2	(0.999, 3.75]
1	15	(11.0, 20.0]
2	20	(11.0, 20.0]
3	5	(3.75, 7.5]
4	1	(0.999, 3.75]
5	4	(3.75, 7.5]
6	7	(3.75, 7.5]
7	9	(7.5, 11.0]
8	10	(7.5, 11.0]
9	3	(0.999, 3.75]
10	14	(11.0, 20.0]
11	8	(7.5, 11.0]

In [314]:

```
bin_labels2 = ['1', '2', '3', '4']
df3['Bin qcut Label'] = pd.qcut(df3['Values'],numberBins, labels = bin_labels2)
df3
```

Out[314]:

	Values	Bin qcut	Bin qcut Label
0	2	(0.999, 3.75]	1
1	15	(11.0, 20.0]	4
2	20	(11.0, 20.0]	4
3	5	(3.75, 7.5]	2
4	1	(0.999, 3.75]	1
5	4	(3.75, 7.5]	2
6	7	(3.75, 7.5]	2
7	9	(7.5, 11.0]	3
8	10	(7.5, 11.0]	3
9	3	(0.999, 3.75]	1
10	14	(11.0, 20.0]	4
11	8	(7.5, 11.0]	3

In [315]:

```
df3
```

Out[315]:

	Values	Bin qcut	Bin qcut Label
0	2	(0.999, 3.75]	1
1	15	(11.0, 20.0]	4
2	20	(11.0, 20.0]	4
3	5	(3.75, 7.5]	2
4	1	(0.999, 3.75]	1
5	4	(3.75, 7.5]	2
6	7	(3.75, 7.5]	2
7	9	(7.5, 11.0]	3
8	10	(7.5, 11.0]	3
9	3	(0.999, 3.75]	1
10	14	(11.0, 20.0]	4
11	8	(7.5, 11.0]	3

In [316]:

```
df3.sort_values("Values", axis = 0, ascending = True,
                inplace = True, na_position ='first')

df3
```

Out[316]:

	Values	Bin qcut	Bin qcut Label
4	1	(0.999, 3.75]	1
0	2	(0.999, 3.75]	1
9	3	(0.999, 3.75]	1
5	4	(3.75, 7.5]	2
3	5	(3.75, 7.5]	2
6	7	(3.75, 7.5]	2
11	8	(7.5, 11.0]	3
7	9	(7.5, 11.0]	3
8	10	(7.5, 11.0]	3
10	14	(11.0, 20.0]	4
1	15	(11.0, 20.0]	4
2	20	(11.0, 20.0]	4

B. Smoothing by bin means

In [317]:

```
mean1 = (df3.loc[df3['Bin qcut Label'] == '1']['Values']).mean()
mean2 = (df3.loc[df3['Bin qcut Label'] == '2']['Values']).mean()
mean3 = (df3.loc[df3['Bin qcut Label'] == '3']['Values']).mean()
mean4 = (df3.loc[df3['Bin qcut Label'] == '4']['Values']).mean()
```

In [351]:

```
print(mean1)
print(mean2)
print(mean3)
print(mean4)
```

```
2.0
5.333333333333333
9.0
16.333333333333332
```

In [344]:

```
index1 = df3.loc[df3['Bin qcut Label'] == '1'].index
index2 = df3.loc[df3['Bin qcut Label'] == '2'].index
index3 = df3.loc[df3['Bin qcut Label'] == '3'].index
index4 = df3.loc[df3['Bin qcut Label'] == '4'].index
```

In [325]:

```
print(index1)
print(index2)
print(index3)
print(index4)
```

```
Int64Index([4, 0, 9], dtype='int64')
Int64Index([5, 3, 6], dtype='int64')
Int64Index([11, 7, 8], dtype='int64')
Int64Index([10, 1, 2], dtype='int64')
```

In [303]:

```
df3.loc[index1,'Values'] = mean1
df3.loc[index2,'Values'] = mean2
df3.loc[index3,'Values'] = mean3
df3.loc[index4,'Values'] = mean4
```

In [304]:

```
df3
```

Out[304]:

	Values	Bin qcut	Bin qcut Label
0	2.000000	(0.999, 3.75]	1
1	16.333333	(11.0, 20.0]	4
2	16.333333	(11.0, 20.0]	4
3	5.333333	(3.75, 7.5]	2
4	2.000000	(0.999, 3.75]	1
5	5.333333	(3.75, 7.5]	2
6	5.333333	(3.75, 7.5]	2
7	9.000000	(7.5, 11.0]	3
8	9.000000	(7.5, 11.0]	3
9	2.000000	(0.999, 3.75]	1
10	16.333333	(11.0, 20.0]	4
11	9.000000	(7.5, 11.0]	3

C. Smoothing by bin boundaries

In [350]:

```
dataSlice1 = df3.loc[df3['Bin qcut Label'] == '1', ['Values']]
dataSlice2 = df3.loc[df3['Bin qcut Label'] == '2', ['Values']]
dataSlice3 = df3.loc[df3['Bin qcut Label'] == '3', ['Values']]
dataSlice4 = df3.loc[df3['Bin qcut Label'] == '4', ['Values']]
```

In [353]:

```
for i in  ( index1.tolist()):
    if dataSlice1.loc[i, "Values"] < mean1:
        df3.loc[i,'Values'] = dataSlice1['Values'].min()
    else:
        df3.loc[i,'Values'] = dataSlice1['Values'].max()

for i in  ( index2.tolist()):
    if dataSlice2.loc[i, "Values"] < mean2:
        df3.loc[i,'Values'] = dataSlice2['Values'].min()
    else:
        df3.loc[i,'Values'] = dataSlice2['Values'].max()

for i in  ( index3.tolist()):
    if dataSlice3.loc[i, "Values"] < mean3:
        df3.loc[i,'Values'] = dataSlice3['Values'].min()
    else:
        df3.loc[i,'Values'] = dataSlice3['Values'].max()

for i in  ( index4.tolist()):
    if dataSlice4.loc[i, "Values"] < mean4:
        df3.loc[i,'Values'] = dataSlice4['Values'].min()
    else:
        df3.loc[i,'Values'] = dataSlice4['Values'].max()
```

In [354]:

```
df3
```

Out[354]:

	Values	Bin qcut	Bin qcut Label
4	1	(0.999, 3.75]	1
0	3	(0.999, 3.75]	1
9	3	(0.999, 3.75]	1
5	4	(3.75, 7.5]	2
3	4	(3.75, 7.5]	2
6	7	(3.75, 7.5]	2
11	8	(7.5, 11.0]	3
7	10	(7.5, 11.0]	3
8	10	(7.5, 11.0]	3
10	14	(11.0, 20.0]	4
1	14	(11.0, 20.0]	4
2	20	(11.0, 20.0]	4

2. Use the below methods to normalize the following data: [10, 20, 35, 70, 100]:

A. min-max normalization with min=0 and max=1.

In [358]:

```
import pandas as pd
import math
import numpy as np

data2 = [10, 20, 35, 70, 100]

df = pd.DataFrame()

df['Values']  = data2

nInstances = df.shape[0]

print (nInstances)

df
```

5

Out[358]:

	Values
0	10
1	20
2	35
3	70
4	100

In [359]:

```
#normalized = (x-min(x))/(max(x)-min(x))
normalized_df= (df-df.min())/(df.max()-df.min())
```

In [360]:

```
normalized_df
```

Out[360]:

	Values
0	0.000000
1	0.111111
2	0.277778
3	0.666667
4	1.000000

In [361]:

```
data2 = [10, 20, 35, 70, 100]

df = pd.DataFrame()

df['Values']  = data2

df
```

Out[361]:

	Values
0	10
1	20
2	35
3	70
4	100

B. z-score normalization

In [362]:

```
z_score_normalized_df=(df-df.mean())/df.std()
z_score_normalized_df
```

Out[362]:

	Values
0	-0.990637
1	-0.722897
2	-0.321288
3	0.615801
4	1.419021

C. normalization with decimal scaling.

In [375]:

```
data2 = [10, 20, 35, 70, 100]

df = pd.DataFrame()

df['Values']  = data2

df
```

Out[375]:

	Values
0	10
1	20
2	35
3	70
4	100

In [376]:

```
for x in df:
    p = df[x].max()
    q = len(str(abs(int(p))))
    df[x] = df[x]/10**q
df
```

Out[376]:

	Values
0	0.010
1	0.020
2	0.035
3	0.070
4	0.100

3. Suppose that a hospital has kept records for the age and BMI (Body Mass Index) for 16 randomly selected individuals, with the data presented as below. Calculate the correlation coefficient for the two above attributes. Are these two attributes positively or negatively correlated?

Age	**23**	**23**	**27**	**27**	**39**	**41**	**47**	**49**	**52**	**54**	**54**	**56**	**57**	**58**	**58**	**60**
BMI	16	19	18	21	24	23	26	25	24	26	23	26	27	31	26	27

In [340]:

```
import pandas as pd
import math
import numpy as np

dataAge = [23,23,27,27,39,41,47,49,52,54,54,56,57,58,58,60]
dataBMI = [16,19,18,21,24,23,26,25,24,26,23,26,27,31,26,27]

df = pd.DataFrame({"Age": dataAge,
                   "BMI": dataBMI})
df
```

Out[340]:

	Age	BMI
0	23	16
1	23	19
2	27	18
3	27	21
4	39	24
5	41	23
6	47	26
7	49	25
8	52	24
9	54	26
10	54	23
11	56	26
12	57	27
13	58	31
14	58	26
15	60	27

In [341]:

```
corrrelation = df.corr(method="pearson");

print("Pearson correlation coefficient:");

print(corrrelation);
```

```
Pearson correlation coefficient:
          Age       BMI
Age  1.000000  0.891108
BMI  0.891108  1.000000
```

In [342]:

```
import pingouin as pg
pg.corr(x=df['Age'], y=df['BMI'])
```

Out[342]:

	n	r	CI95%	r2	adj_r2	p-val	BF10	power
pearson	16	0.891108	[0.71, 0.96]	0.794074	0.762393	0.000004	5138.535	0.999491

these two attributes positively correlated with a correlation coefficient of r = 0.89

4. Patients at two hospitals, Hospital A and Hospital B, have been provided with feedback forms on patient satisfaction, with the below responses recorded. Is patient satisfaction correlated with a specific hospital? Use a chi-square test to find out, assuming a significance level of 0.001 and a corresponding chi-square significance value of 10.828.

Rating/Hospital	Hospital A	Hospital B
Satisfied	71	129
Dissatisfied	37	73

In [148]:

```
import pandas as pd
import numpy as np
import pingouin as pg
from scipy.stats import chi2_contingency
```

In [153]:

```
df = pd.DataFrame({'Hospital A' : [71 , 37],
                   'Hospital B' : [129, 73 ]},
                   index =['Satisfied', 'Dissatisfied']
                  )
df
```

Out[153]:

	Hospital A	Hospital B
Satisfied	71	129
Dissatisfied	37	73

In [172]:

```
c, p, dof, expected = chi2_contingency(df,correction=False)
print('chi2 = ' , c)
print('p value = ' , p)
print('dof = ' , dof)
print('expected = \n' , expected)
```

```
chi2 =  0.10857544087742113
p value =  0.7417718412505765
dof =  1
expected =
 [[ 69.67741935 130.32258065]
 [ 38.32258065  71.67741935]]
```

Satisfaction is not correlated with a specific hospital beacause p value is more than 0.001 meaning they are independent

In [164]:

```
#data = pg.read_dataset('chi2_independence')
#expected, observed, stats = pg.chi2_independence(df, x='Hospital A', y='Hospital B')
#observed
```

5. Load the CSV file BL_books.csv, which includes records of books hosted at the British Library. Inspect the data and describe any issues observed with the data. Remove fields that refer to internal processes at the British Library and do not describe the books themselves (Corporate Author, Corporate Contributors, Issuance type, Former owner, Shelfmarks, Engraver). Clean data in column Date of Publication, as to remove the extra dates in square brackets (e.g. 1879 [1878] -> 1879) and convert date ranges to their start date, wherever present (e.g. 1860-63 -> 1860).

In [215]:

```
import pandas as pd
import numpy as np
import re

df = pd.read_csv('./BL-books.csv', index_col ="Identifier")
```

In [221]:

```
df.head()
```

Out[221]:

Identifier	Edition Statement	Place of Publication	Date of Publication	Publisher	Title	Author	Contributors
206	NaN	London	1879 [1878]	S. Tinsley & Co.	Walter Forbes. [A novel.] By A. A	A. A.	FORBES, Walter.
216	NaN	London; Virtue & Yorston	1868	Virtue & Co.	All for Greed. [A novel. The dedication signed...	A., A. A.	BLAZE DE BURY, Marie Pauline Rose - Baroness
218	NaN	London	1869	Bradbury, Evans & Co.	Love the Avenger. By the author of “All for Gr...	A., A. A.	BLAZE DE BURY, Marie Pauline Rose - Baroness
472	NaN	London	1851	James Darling	Welsh Sketches, chiefly ecclesiastical, to the...	A., E. S.	Appleyard, Ernest Silvanus.
480	A new edition, revised, etc.	London	1857	Wertheim & Macintosh	[The World in which I live, and my place in it...	A., E. S.	BROOME, John Henry.

In [216]:

```
#(df['Date of Publication'].str.strip().values == "nan" ).sum()

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

for col in df.columns:
    print('\t%s: %d' % ( col,df[col].isna().sum() ) )
```

```
Number of instances = 8287
Number of attributes = 14
        Edition Statement: 7514
        Place of Publication: 0
        Date of Publication: 181
        Publisher: 4195
        Title: 0
        Author: 1778
        Contributors: 0
        Corporate Author: 8287
        Corporate Contributors: 8287
        Former owner: 8286
        Engraver: 8287
        Issuance type: 0
        Flickr URL: 0
        Shelfmarks: 0
```

In [193]:

```
df.loc[667,'Date of Publication']
```

Out[193]:

```
nan
```

In [233]:

```
import pandas as pd
import numpy as np
import re

df = pd.read_csv('./BL-books.csv', index_col ="Identifier")

df['Date of Publication'] = df['Date of Publication'].replace(np.NaN,9999)


df['Date of Publication'] = df['Date of Publication'].astype(str)

N=4


def filterNumber(n):

    if(len(n)==4):
        return True
    else:
        return False


for index in df.index:

    temp = []
    tempBracket = []
    res = []
    resBracket = []


    tempBracket = re.findall(r'\[.*?\]', df.loc[index,'Date of Publication'])

    if len(tempBracket) > 0:

        tempBracket = re.findall(r'\d+', tempBracket[0])


    a_string = df.loc[index,'Date of Publication']
    b_string = re.sub(r" ?\[[^)]+\]", "", a_string )
    temp = re.findall(r'\d+', b_string)


#    for word in a_string.split():
#        if word.isdigit():
#             temp.append(word)


    if len(temp) > 0 and temp[0] == '9999':

        temp = re.findall(r'\d+', df.loc[index,'Place of Publication'])


    res = list(filter(filterNumber, temp))
    resBracket = list(filter(filterNumber, tempBracket))


    if len(res) > 0 :

        intrRes = [int(i) for i in res]
```

```
        minDate = min(intrRes)
        df.at[index,'Date of Publication'] = str(minDate)
        #print(str(minDate))

    elif len(resBracket) > 0 and len(res) <= 0 :

        intrRes = [int(i) for i in resBracket]
        minDate = min(intrRes)
        df.at[index,'Date of Publication'] = str(minDate)
        #print(str(minDate))
```

In [244]:

```
print("These books had no date anywhere in their records!")
df.loc[df['Date of Publication'].str.len() != 4 ]['Place of Publication']
```

```
These books had no date anywhere in their records!
```

Out[244]:

```
Identifier
516336     London
3605102    Oxford
Name: Place of Publication, dtype: object
```

In [236]:

```
df.loc[2167220,'Date of Publication']
```

Out[236]:

```
'1890'
```

In [230]:

```
df = df.drop(columns=['Contributors', 'Corporate Author','Corporate Contributors','Former owner','Engraver','Issuance type'])
```

In [231]:

```
df
```

Out[231]:

Identifier	Edition Statement	Place of Publication	Date of Publication	Publisher	Title	Author	
206	NaN	London	1879	S. Tinsley & Co.	Walter Forbes. [A novel.] By A. A	A. A.	http://ww
216	NaN	London; Virtue & Yorston	1868	Virtue & Co.	All for Greed. [A novel. The dedication signed...	A., A. A.	http://ww
218	NaN	London	1869	Bradbury, Evans & Co.	Love the Avenger. By the author of “All for Gr...	A., A. A.	http://ww
472	NaN	London	1851	James Darling	Welsh Sketches, chiefly ecclesiastical, to the...	A., E. S.	http://ww
480	A new edition, revised, etc.	London	1857	Wertheim & Macintosh	[The World in which I live, and my place in it...	A., E. S.	http://ww
...	...	...	...	...	...	...	
4158088	NaN	London	1838	NaN	The Parochial History of Cornwall, founded on,...	GIDDY, afterwards GILBERT, Davies.	http://ww
4158128	NaN	Derby	1831	M. Mozley & Son	The History and Gazetteer of the County of Der...	GLOVER, Stephen - of Derby	http://ww
4159563	NaN	London	1806	T. Cadell and W. Davies	Magna Britannia; being a concise topographical...	LYSONS, Daniel - M.A., F.R.S., and LYSONS (Sam...	http://ww
4159587	NaN	Newcastle upon Tyne	1834	Mackenzie & Dent	An historical, topographical and descriptive v...	Mackenzie, E. (Eneas)	http://ww
4160339	NaN	London	1834	NaN	Collectanea Topographica et Genealogica. [Firs...	NaN	http://ww

8287 rows × 8 columns

6. Load the CSV file country-income.csv which includes both numerical and categorical values. Perform data cleaning in order to replace any NaN values with the mean of the value for a given field. Then replace any categorical labels with numerical labels. Display the resulting dataset. You can use the sklearn.impute and sklearn.preprocessing packages to asssist you.

In [299]:

```
import pandas as pd
import numpy as np
import re
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('./country-income.csv')
```

In [301]:

```
df
```

Out[301]:

	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
5	India	40.0	69600.0	Yes
6	Brazil	NaN	62400.0	No
7	India	53.0	94800.0	Yes
8	USA	55.0	99600.0	No
9	India	42.0	80400.0	Yes

In [303]:

```
mean1 = (df.loc[df['Region'] == 'USA','Income']).mean()

print(mean1)

df.at[df['Region'] == 'USA','Income'] = df.loc[df['Region'] == 'USA','Income'].fillna(m
ean1)

df['Income']
```

```
82200.0
```

Out[303]:

```
0    86400.0
1    57600.0
2    64800.0
3    73200.0
4    82200.0
5    69600.0
6    62400.0
7    94800.0
8    99600.0
9    80400.0
Name: Income, dtype: float64
```

In [304]:

```
mean2 = (df.loc[df['Region'] == 'Brazil','Age'].mean())

print(mean2)

df.at[df['Region'] == 'Brazil','Age'] = df.loc[df['Region'] == 'Brazil','Age'].fillna(m
ean2)

df['Age']
```

```
37.5
```

Out[304]:

```
0    49.0
1    32.0
2    35.0
3    43.0
4    45.0
5    40.0
6    37.5
7    53.0
8    55.0
9    42.0
Name: Age, dtype: float64
```

In [305]:

```
def Encoder(df):
    columnsToEncode = list(df.select_dtypes(include=['category','object']))
    le = LabelEncoder()
    for feature in columnsToEncode:
        try:
            df[feature] = le.fit_transform(df[feature])
        except:
            print('Error encoding '+feature)
    return df
df = Encoder(df)
```

In [306]:

```
df
```

Out[306]:

	Region	Age	Income	Online Shopper
0	1	49.0	86400.0	0
1	0	32.0	57600.0	1
2	2	35.0	64800.0	0
3	0	43.0	73200.0	0
4	2	45.0	82200.0	1
5	1	40.0	69600.0	1
6	0	37.5	62400.0	0
7	1	53.0	94800.0	1
8	2	55.0	99600.0	0
9	1	42.0	80400.0	1

7. Load the CSV file shoesize.csv, which includes measurements of shoe size and height (in inches) for 408 subjects, both female and male. Plot the scatterplots of shoe size versus height for female and male subjects separately. Compute the Pearson's correlation coefficient of shoe size versus height for female and male subjects separately. What can be inferred by the scatterplots and computed correlation coefficients? You can implement your own formulation of the correlation coefficient or use the scipy.stats package to assist you.

In [347]:

```
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
import math
import pingouin as pg


from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('./shoesize.csv', index_col ="Index")
```

In [348]:

```
df.head()
```

Out[348]:

Index	Gender	Size	Height
1	F	5.5	60.0
2	F	6.0	60.0
3	F	7.0	60.0
4	F	8.0	60.0
5	F	8.0	60.0

In [316]:

```
df.loc[df['Gender'] == 'F',['Height','Size']]
```

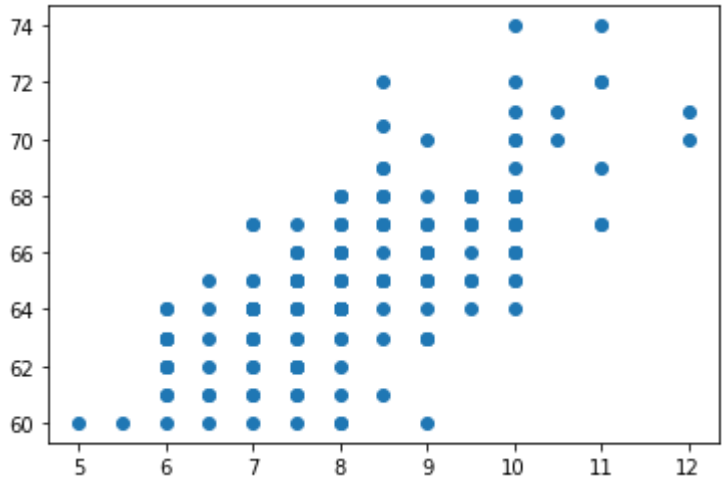
Out[316]:

Index	Height	Size
1	60.0	5.5
2	60.0	6.0
3	60.0	7.0
4	60.0	8.0
5	60.0	8.0
...	...	...
183	72.0	10.0
184	72.0	11.0
185	72.0	11.0
186	74.0	10.0
187	74.0	11.0

187 rows × 2 columns

In [349]:

```
# Here we subset your dataframe where the temperature is 90, which will give you a
# boolean array for your dataframe.
df1 = df.loc[df['Gender'] == 'F',['Height','Size']]
# Apply your boolean against your dataframe to grab the correct rows:


# Now plot your scatter plot
plt.scatter(x=df1['Size'] ,y=df1['Height'])
plt.show()
```



In [350]:

```
corrrelation = df1.corr(method="pearson");

print("Pearson correlation coefficient:");

print(corrrelation);
```

```
Pearson correlation coefficient:
          Height      Size
Height  1.000000  0.707812
Size    0.707812  1.000000
```

In [351]:

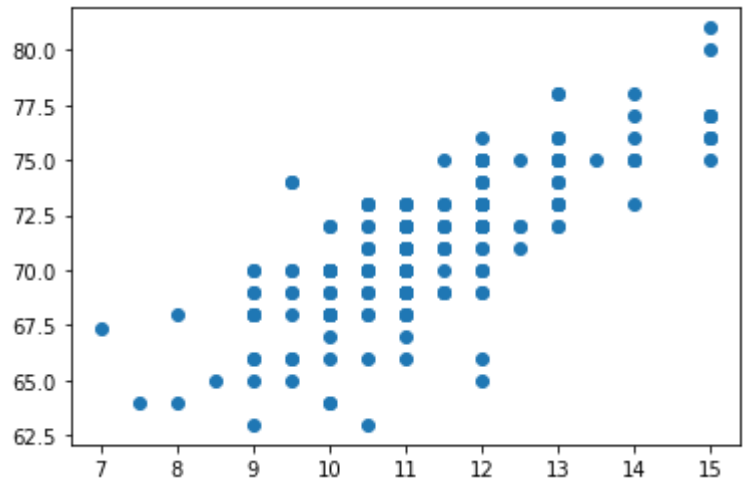
```
pg.corr(x=df1['Height'], y=df1['Size'])
```

Out[351]:

	n	r	CI95%	r2	adj_r2	p-val	BF10	power
pearson	187	0.707812	[0.63, 0.77]	0.500998	0.495574	9.773451e-30	3.894e+26	1.0

In [352]:

```
df2 = df.loc[df['Gender'] == 'M',['Height','Size']]
# Apply your boolean against your dataframe to grab the correct rows:


# Now plot your scatter plot
plt.scatter(x=df2['Size'] ,y=df2['Height'])
plt.show()
```



In [355]:

```
corrrelation = df2.corr(method="pearson");

print("Pearson correlation coefficient:");

print(corrrelation);
```

```
Pearson correlation coefficient:
          Height      Size
Height  1.000000  0.767709
Size    0.767709  1.000000
```

In [354]:

```
pg.corr(x=df2['Height'], y=df2['Size'])
```

Out[354]:

	n	r	CI95%	r2	adj_r2	p-val	BF10	power
pearson	221	0.767709	[0.71, 0.82]	0.589378	0.58561	3.285711e-44	7.449e+40	1.0

From the scatterplot and the correlation coefficient we infer that the height and are relatively correlated fro both genders

8. Using the breast cancer dataset from section 1 of this notebook, perform Principal Component Analysis with 2 components. Compute the explained variance ratio for each component, and plot the scatterplot of all samples along the two principal components, color-coded according to the "Class" column (this column should not be used in the PCA analysis). Ensure that your data is normalized prior to performing PCA. What insights can you obtain by the explained variance ratio of each component, and by viewing the scatterplot of the principal components?

In [179]:

```
import matplotlib.image as mpimg
import numpy as np
import pandas as pd
from scipy import stats
import re
import matplotlib.pyplot as plt
import math
import pingouin as pg
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler



data = pd.read_csv('./breast-cancer-wisconsin.csv', header=None)
data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformit
y of Cell Shape',
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bla
nd 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()

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

data['Bare Nuclei'] = pd.to_numeric(data['Bare Nuclei'])

data[(np.abs(stats.zscore(data)) < 3).all(axis=1)]

data.drop_duplicates()
```

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

Out[179]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mi
0	5	1	1	1	2	1.0	3	1	
1	5	4	4	5	7	10.0	3	2	
2	3	1	1	1	2	2.0	3	1	
3	6	8	8	1	3	4.0	3	7	
4	4	1	1	3	2	1.0	3	1	
...	...	...	...	...	...	...	...	...	
693	3	1	1	1	2	1.0	2	1	
694	3	1	1	1	3	2.0	1	1	
696	5	10	10	3	7	3.0	8	10	
697	4	8	6	4	3	4.0	10	6	
698	4	8	8	5	4	5.0	10	4	

457 rows × 10 columns

In [180]:

```
features = list(data.columns)
features.remove('Class')
features
```

Out[180]:

```
['Clump Thickness',
 'Uniformity of Cell Size',
 'Uniformity of Cell Shape',
 'Marginal Adhesion',
 'Single Epithelial Cell Size',
 'Bare Nuclei',
 'Bland Chromatin',
 'Normal Nucleoli',
 'Mitoses']
```

In [181]:

```
features = list(data.columns)
features.remove('Class')
features

# Separating out the features
x = data.loc[:, features].values
# Separating out the target
y = data.loc[:,['Class']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)
```

In [182]:

```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents , columns = ['principal component 1', 'principal component 2'])
```

In [183]:

```
finalDf = pd.concat([principalDf, data[['Class']]], axis = 1)
```

In [184]:

```
finalDf
```

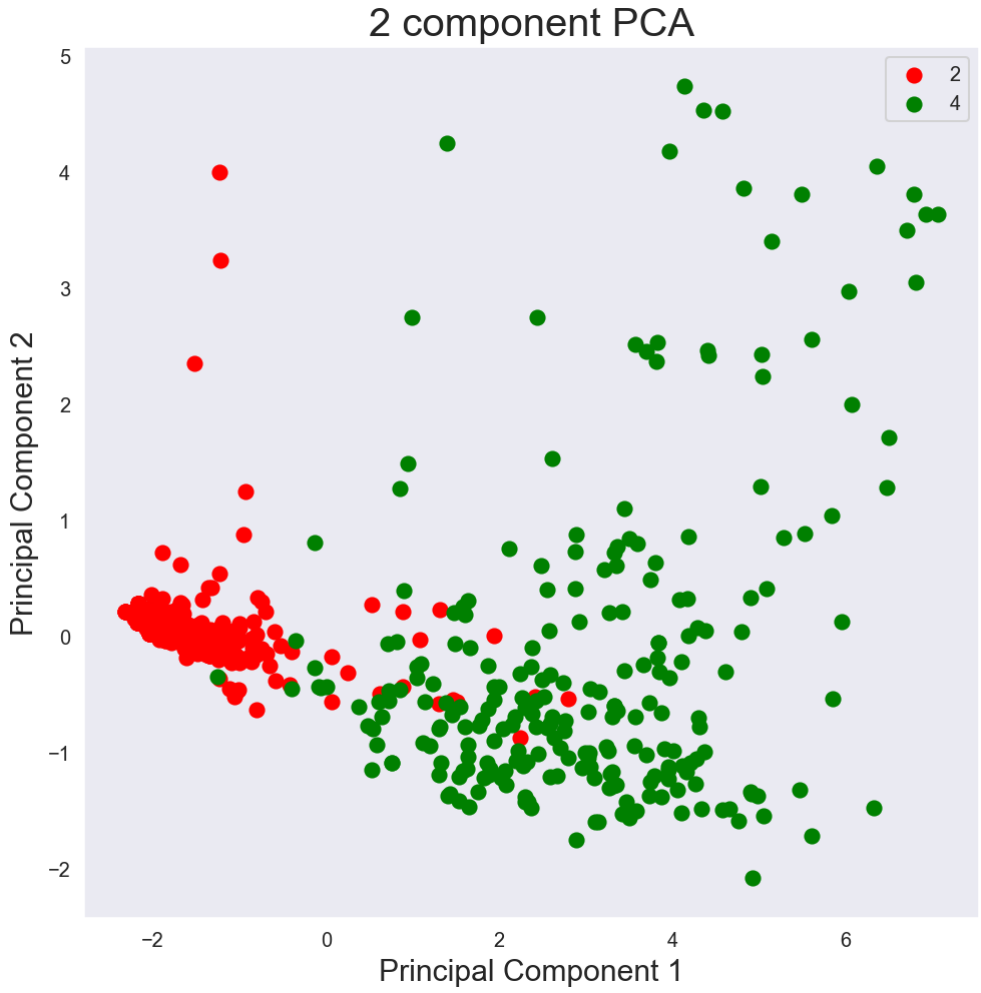
Out[184]:

	principal component 1	principal component 2	Class
0	-1.456220	-0.110210	2
1	1.466279	-0.544894	2
2	-1.579311	-0.074854	2
3	1.505247	-0.558853	2
4	-1.330551	-0.089657	2
...	...	...	...
694	-1.711249	0.188019	2
695	-2.063036	0.234224	2
696	3.825359	-0.180466	4
697	2.269482	-1.113435	4
698	2.664453	-1.197242	4

699 rows × 3 columns

In [185]:

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)

class_list = data['Class'].tolist()
myset = set(class_list)
targets = list(myset)

colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Class'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1'], finalDf.loc[indices
ToKeep, 'principal component 2'], c = color, s = 50)
ax.legend(targets)
ax.grid()
```



In [186]:

```
print(pca.explained_variance_)
```

```
[5.8985519  0.77588307]
```

In [187]:

```
print(pca.explained_variance_ratio_)
```

```
[0.65445704 0.0860859 ]
```

By using the attribute explained_variance*ratio*, you can see that the first principal component contains 65.45% of the variance and the second principal component contains 8.61% of the variance.

Part 2

1. In Section 1, what kind of relationship can be inferred from summary statistics regarding `ACT composite score` and `SAT total score` ? Which visualisations make this relationship apparent?

In [61]:

```
import pandas as pd

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())

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

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

print('Univariate summaries:')
display(df.describe())
print("Frequency of parental levels of education:")
freq_education = df['parental level of education'].value_counts()/len(df)
display(freq_education)

print("\nCorrelation coefficients:")
display(df.corr())
```

Univariate summaries:

	ACT composite score	SAT total score	parental income	high school gpa	college gpa	years to graduate
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000
mean	28.557000	1997.803000	66564.905000	3.712900	3.36990	4.992000
std	2.776051	142.736442	19451.865744	0.283415	0.23622	1.403533
min	19.000000	1498.000000	1882.000000	2.700000	2.70000	3.000000
25%	27.000000	1907.750000	53443.000000	3.500000	3.20000	4.000000
50%	29.000000	1999.000000	65441.000000	3.800000	3.40000	5.000000
75%	30.000000	2092.250000	79845.250000	4.000000	3.50000	6.000000
max	36.000000	2397.000000	120391.000000	4.000000	4.00000	10.000000

Frequency of parental levels of education:

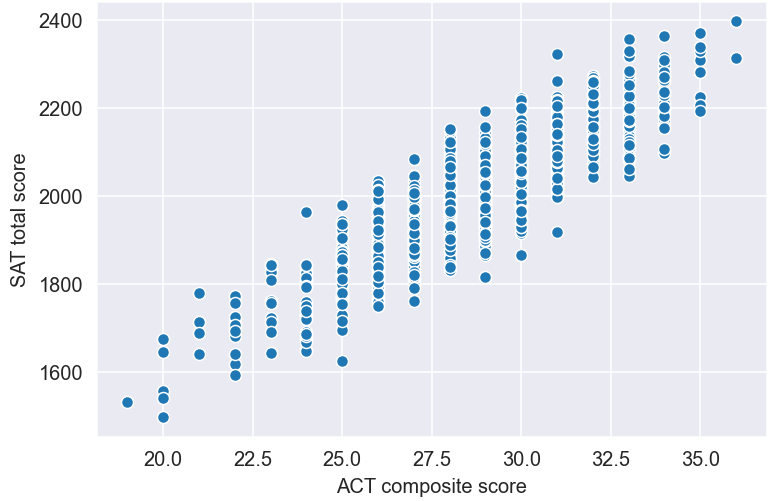
```
some college          0.224
associate's degree    0.213
high school           0.196
some high school      0.196
bachelor's degree     0.103
master's degree       0.068
Name: parental level of education, dtype: float64
```

Correlation coefficients:

	ACT composite score	SAT total score	parental income	high school gpa	college gpa	years to graduate
ACT composite score	1.000000	0.885884	0.183879	0.874206	0.507349	-0.129880
SAT total score	0.885884	1.000000	0.247556	0.910425	0.518257	-0.125523
parental income	0.183879	0.247556	1.000000	0.227238	0.460863	-0.239500
high school gpa	0.874206	0.910425	0.227238	1.000000	0.492489	-0.119524
college gpa	0.507349	0.518257	0.460863	0.492489	1.000000	-0.467499
years to graduate	-0.129880	-0.125523	-0.239500	-0.119524	-0.467499	1.000000

In [6]:

```
sns.scatterplot(x='ACT composite score', y='SAT total score', data=df)
plt.show()
```

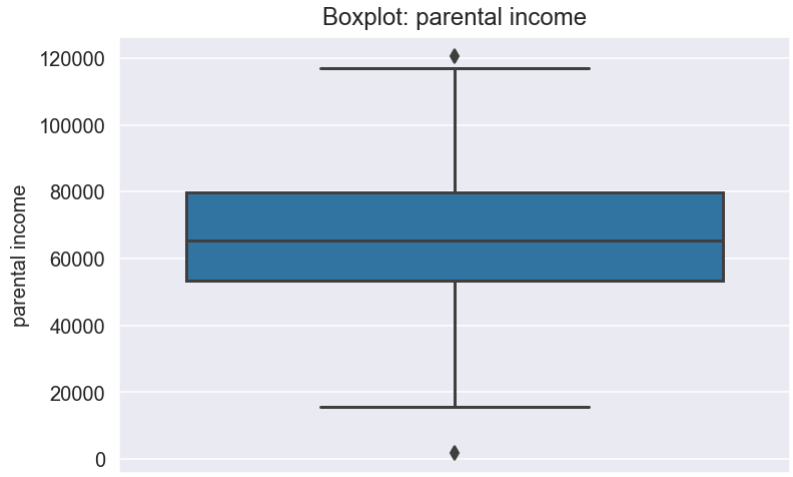


the relatonship between ACT composite score and SAT total score is a positive the scatter plot makes the relationship apparent

2. Based on the box plots presented in Section 1, what is the relationship between parental level of education and parental income? Using table visualisation, find and show the entire rows that correspond to the outliers regarding parental income whose parents have a master's degree.

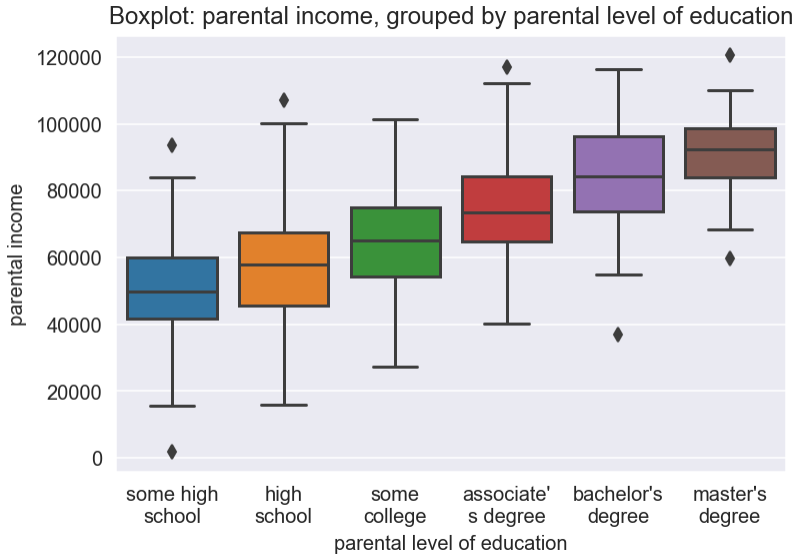
In [7]:

```
sns.boxplot(df['parental income'], orient='v')
plt.title('Boxplot: parental income')
plt.show()
```



In [8]:

```
ax = sns.boxplot(x='parental level of education', y='parental income', data=df)
plt.title('Boxplot: parental income, grouped by parental level of education')

# Wrap xticks
import textwrap
ax.set_xticklabels([textwrap.fill(t.get_text(), 10)  for t in ax.get_xticklabels()])

plt.show()
```



In [14]:

```
df.head()
```

Out[14]:

	ACT composite score	SAT total score	parental level of education	parental income	high school gpa	college gpa	years to graduate
0	30	2206	master's degree	94873	4.0	3.8	3
1	26	1953	some college	42767	3.6	2.7	9
2	28	2115	some high school	46316	4.0	3.3	5
3	33	2110	some high school	52370	4.0	3.5	4
4	30	2168	bachelor's degree	92665	4.0	3.6	4

In [18]:

```
#display(df[df['parental level of education'] == "master's degree"].sort_values(by='high school gpa', ascending=False))
(df.loc[df['parental level of education'] == "master's degree",["parental income"]])
```

Out[18]:

	parental income
0	94873
27	96573
37	90775
65	75974
99	85135
...	...
912	83814
927	92185
929	79398
964	93314
968	85534

68 rows × 1 columns

In [53]:

```
import numpy as np
from scipy import stats


data = df.loc[df['parental level of education'] == "master's degree","parental income"]
.tolist()


mean = np.mean(data)
std = np.std(data)
print('mean of the dataset is', mean)
print('std. deviation is', std)

threshold = 2
outlier = []
for i in data:
    z = (i-mean)/std
    if np.abs(z) > threshold:
        outlier.append(i)
print('outlier in dataset is', outlier)
```

```
mean of the dataset is 91718.63235294117
std. deviation is 10965.103920947971
outlier in dataset is [120391, 59724, 68319]
```

In [59]:

```
df.loc[df['parental income'].isin(outlier)]
```

Out[59]:

	ACT composite score	SAT total score	parental level of education	parental income	high school gpa	college gpa	years to graduate
411	31	2108	master's degree	120391	4.0	3.6	4
420	28	2097	master's degree	59724	3.9	3.2	4
641	32	2274	master's degree	68319	4.0	3.6	4

In [ ]:

```

```

3. Using an example, explain the importance of scaling features so that their magnitudes are comparable when computing distances.

Most of the times, a dataset will contain features highly varying in magnitudes, units and range. But since, most of the machine learning algorithms use Eucledian distance between two data points in their computations, this is a problem. If left alone, these algorithms only take in the magnitude of features neglecting the units.

For example, The results would vary greatly between different units, 5kg and 5000gms. The features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes.

To supress this effect, we need to bring all features to the same level of magnitudes. This can be acheived by feature scaling.

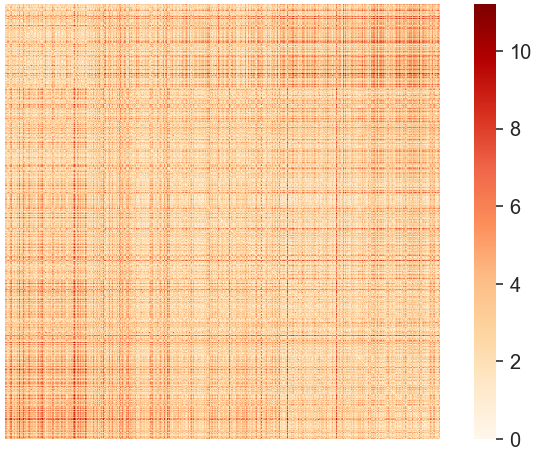
4. In Section 1, the distance matrix visualisation is not very informative. However, it is still possible to infer that the average distance between students whose parents only have some high school education and students whose parents have a master's degree is larger than the average distance between students whose parents only have some high school education. Explain how this inference is possible from the visualisation.

In [84]:

```
from sklearn.preprocessing import StandardScaler
from scipy.spatial import distance

display(df.shape)

df_sorted = df.sort_values(by='parental level of education', ascending=True)
parental_education_sorted = df_sorted['parental level of education']

#display(parental_education_sorted)

X = df_sorted.drop(columns='parental level of education').to_numpy()



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

dist = distance.squareform(distance.pdist(X))
sns.heatmap(dist, square=True, xticklabels=False, yticklabels=False, cmap='OrRd')

display(dist.shape)

plt.show()
```

```
(1000, 7)
```

```
(1000, 1000)
```



As we can see from the Heatmap the top left corner where the distance between students whose parents only have some high school education lay is light shaded which infer the distances are short.

However, as you move to the lower left corner where the distances between students whose parents only have some high school education and students whose parents have a master's degree lay is darker shaded which infer the distances are larger.

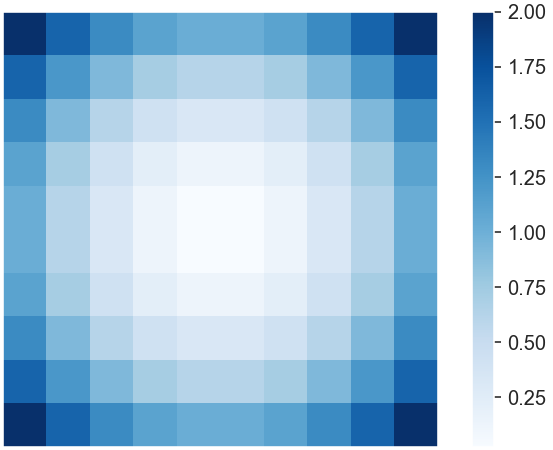
5. In Section 2, increase the number of evenly spaced numbers from 10 to 100 for both axes and observe the corresponding heat map created through nearest neighbour interpolation. Read about this interpolation method and explain what you observed.

In [85]:

```
import numpy as np
x_range = np.linspace(-1, 1, 10)
y_range = np.linspace(-1, 1, 10)

# meshgrid: X[i, j] == x_range[j] and Y[i, j] == y_range[i]
X, Y = np.meshgrid(x_range, y_range)

# Z[i, j] == f(x_range[j], y_range[i])
Z = X**2 + Y**2

# Dataset representation
df = pd.DataFrame({'x': X.reshape(-1), 'y': Y.reshape(-1), 'z = f(x,y)': Z.reshape(-1
)})
display(df)

# Interpolation: point (x, y) is colored according to the value z of the nearest point
 in the dataset
plt.imshow(Z, cmap='Blues', aspect='equal', interpolation='nearest')
plt.colorbar()

# xticks and yticks would show Z matrix indices
plt.xticks([])
plt.yticks([])

plt.show()
```

	x	y	z = f(x,y)
0	-1.000000	-1.0	2.000000
1	-0.777778	-1.0	1.604938
2	-0.555556	-1.0	1.308642
3	-0.333333	-1.0	1.111111
4	-0.111111	-1.0	1.012346
...	...	...	...
95	0.111111	1.0	1.012346
96	0.333333	1.0	1.111111
97	0.555556	1.0	1.308642
98	0.777778	1.0	1.604938
99	1.000000	1.0	2.000000

100 rows × 3 columns

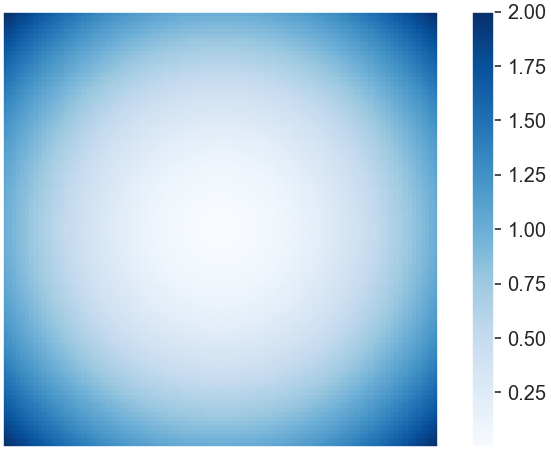


In [147]:

```
import numpy as np
x_range = np.linspace(-1, 1, 100)
y_range = np.linspace(-1, 1, 100)

# meshgrid: X[i, j] == x_range[j] and Y[i, j] == y_range[i]
X, Y = np.meshgrid(x_range, y_range)

# Z[i, j] == f(x_range[j], y_range[i])
Z = X**2 + Y**2

#display(z.shape)


# Dataset representation
df = pd.DataFrame({'x': X.reshape(-1), 'y': Y.reshape(-1), 'z = f(x,y)': Z.reshape(-1
)})
display(df)

# Interpolation: point (x, y) is colored according to the value z of the nearest point
 in the dataset
plt.imshow(Z, cmap='Blues', aspect='equal', interpolation='nearest')
plt.colorbar()

# xticks and yticks would show Z matrix indices
plt.xticks([])
plt.yticks([])

plt.show()
```

	x	y	z = f(x,y)
0	-1.000000	-1.0	2.000000
1	-0.979798	-1.0	1.960004
2	-0.959596	-1.0	1.920824
3	-0.939394	-1.0	1.882461
4	-0.919192	-1.0	1.844914
...	...	...	...
9995	0.919192	1.0	1.844914
9996	0.939394	1.0	1.882461
9997	0.959596	1.0	1.920824
9998	0.979798	1.0	1.960004
9999	1.000000	1.0	2.000000

10000 rows × 3 columns



Interpolation is the process of finding a value between two points on a line or a curve. To help us remember what it means, we should think of the first part of the word, 'inter,' as meaning 'enter,' which reminds us to look 'inside' the data we originally had. This tool, interpolation, is not only useful in statistics, but is also useful in science, business, or when there is a need to predict values that fall within two existing data points.

And here we are using the nearest neighbor interpolation algorithm and the principle of this algorithm is to find the nearest point in the original image, and then insert the pixel value of this point into the target image. The advantage of the nearest neighbor interpolation algorithm is that the algorithm is simple and easy achieve.

as we chnaged te 10x10 to the 100x100 the image got smoother

6. The function load_wine from sklearn.datasets can be used to load the wine dataset into a DataFrame by using the commands df = load_wine(), df = pd.DataFrame(data.data, columns=data.feature_names), and df['target'] = pd.Series(data.target).

6.1 Load the wine dataset. Which feature is categorical? Compute the frequency of each value of the categorical feature.

In [108]:

```
from sklearn.datasets import load_wine
import matplotlib.image as mpimg
import numpy as np
import pandas as pd
from scipy import stats
import re
import matplotlib.pyplot as plt
import math
import pingouin as pg
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

wine_data = load_wine()
df = pd.DataFrame(wine_data.data, columns=wine_data.feature_names)
df['target'] = pd.Series(wine_data.target)
```

In [123]:

```
display( df.describe() )
display( df )
display(wine_data.target_names)
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flav
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonf
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	
...	...	...	...	...	...	...	...	
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	

178 rows × 14 columns

```
array(['class_0', 'class_1', 'class_2'], dtype='<U7')
```

In [118]:

```
train_data = np.array(wine_data.data)
train_labels = np.array(wine_data.target)

num_features = wine_data.data.shape[1]
unique_labels = np.unique(train_labels)
num_classes = len(unique_labels)

#display(train_labels)
display(unique_labels)
display(num_classes)
```

In [117]:

```
print("The wine dataset has " + str(num_features) + " features")
print(wine_data.feature_names)
print("The wine dataset has " + str(num_classes) + " categoryes")
print(wine_data.target_names)
```

```
The wine dataset has 13 features
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_
phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_
intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']
The wine dataset has 3 categoryes
['class_0' 'class_1' 'class_2']
```

In [125]:

```
print("Frequency of the categorical feature:")
freq_categorical_feature = df['target'].value_counts()/len(df)
display(freq_categorical_feature)
```

```
Frequency of the categorical feature:

1    0.398876
0    0.331461
2    0.269663
Name: target, dtype: float64
```

6.2 Compute univariate and multivariate summaries for all the numerical features. Group observations by the categorical feature and compute the corresponding median for each remaining numerical feature

In [128]:

```
print('Univariate summaries:')
display(df.describe())

print("\nCorrelation coefficients:")
display(df.corr())

print('Group observations by the categorical feature and compute the corresponding medi
an for each remaining numerical feature:')
display(df.groupby('target').median())
```

Univariate summaries:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flav
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.

Correlation coefficients:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium
alcohol	1.000000	0.094397	0.211545	-0.310235	0.270798
malic_acid	0.094397	1.000000	0.164045	0.288500	-0.054575
ash	0.211545	0.164045	1.000000	0.443367	0.286587
alcalinity_of_ash	-0.310235	0.288500	0.443367	1.000000	-0.083333
magnesium	0.270798	-0.054575	0.286587	-0.083333	1.000000
total_phenols	0.289101	-0.335167	0.128980	-0.321113	0.214401
flavanoids	0.236815	-0.411007	0.115077	-0.351370	0.195784
nonflavanoid_phenols	-0.155929	0.292977	0.186230	0.361922	-0.256294
proanthocyanins	0.136698	-0.220746	0.009652	-0.197327	0.236441
color_intensity	0.546364	0.248985	0.258887	0.018732	0.199950
hue	-0.071747	-0.561296	-0.074667	-0.273955	0.055398
od280/od315_of_diluted_wines	0.072343	-0.368710	0.003911	-0.276769	0.066004
proline	0.643720	-0.192011	0.223626	-0.440597	0.393351
target	-0.328222	0.437776	-0.049643	0.517859	-0.209179

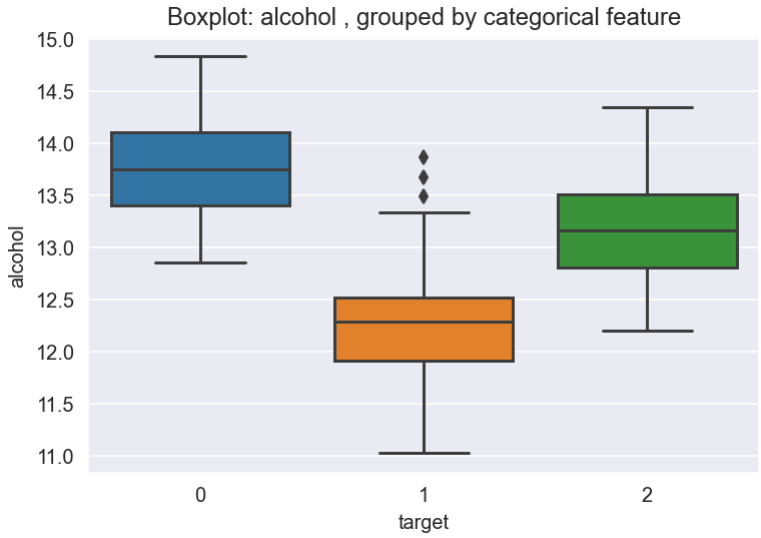
Group observations by the categorical feature and compute the correspondin
g median for each remaining numerical feature:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	no
target								
0	13.750	1.770	2.44	16.8	104.0	2.800	2.980	
1	12.290	1.610	2.24	20.0	88.0	2.200	2.030	
2	13.165	3.265	2.38	21.0	97.0	1.635	0.685	

6.3 Group observations by the categorical feature and create one box plot of alcohol for each group.

In [127]:

```
ax = sns.boxplot(x='target', y='alcohol', data=df)
plt.title('Boxplot: alcohol , grouped by categorical feature')

# Wrap xticks
import textwrap
ax.set_xticklabels([textwrap.fill(t.get_text(), 10)  for t in ax.get_xticklabels()])

plt.show()
```



6.4 Create a scatter plot for the pair of distinct numerical features with the highest correlation.

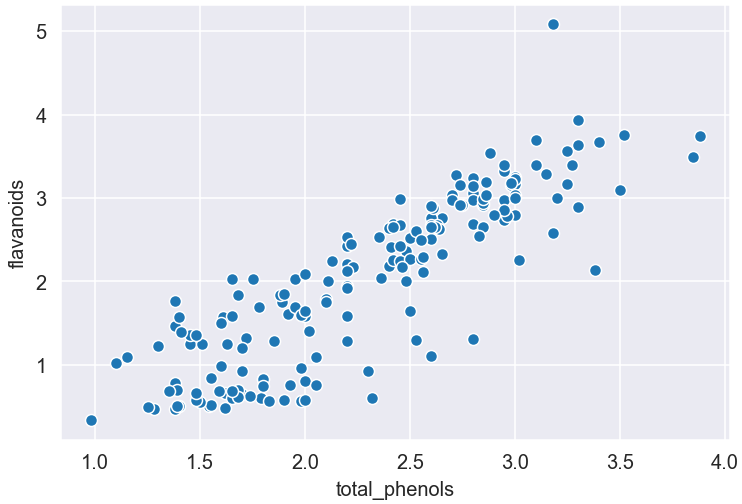
In [132]:

```
def get_feature_correlation(df, top_n=None, corr_method='spearman',
                            remove_duplicates=True, remove_self_correlations=True):
    """
    Compute the feature correlation and sort feature pairs based on their correlation

    :param df: The dataframe with the predictor variables
    :type df: pandas.core.frame.DataFrame
    :param top_n: Top N feature pairs to be reported (if None, all of the pairs will be
returned)
    :param corr_method: Correlation compuation method
    :type corr_method: str
    :param remove_duplicates: Indicates whether duplicate features must be removed
    :type remove_duplicates: bool
    :param remove_self_correlations: Indicates whether self correlations will be remove
d
    :type remove_self_correlations: bool

    :return: pandas.core.frame.DataFrame
    """
    corr_matrix_abs = df.corr(method=corr_method).abs()
    corr_matrix_abs_us = corr_matrix_abs.unstack()
    sorted_correlated_features = corr_matrix_abs_us \
        .sort_values(kind="quicksort", ascending=False) \
        .reset_index()

    # Remove comparisons of the same feature
    if remove_self_correlations:
        sorted_correlated_features = sorted_correlated_features[
            (sorted_correlated_features.level_0 != sorted_correlated_features.level_1)
        ]

    # Remove duplicates
    if remove_duplicates:
        sorted_correlated_features = sorted_correlated_features.iloc[:-2:2]

    # Create meaningful names for the columns
    sorted_correlated_features.columns = ['Feature 1', 'Feature 2', 'Correlation (abs)'
]

    if top_n:
        return sorted_correlated_features[:top_n]

    return sorted_correlated_features


print(get_feature_correlation(df,2))
```

```
     Feature 1      Feature 2  Correlation (abs)
14  flavanoids  total_phenols           0.879404
16      target     flavanoids           0.854908
```

In [136]:

```
sns.scatterplot(x='total_phenols', y='flavanoids', data=df)
plt.show()
```



6.5 Exclude the categorical feature, standardize the numerical features, and display a projection obtained by multidimensional scaling. Color the points by the categorical feature.

In [143]:

```
features = list(df.columns)
features.remove('target')
features

# Separating out the features
x = df.loc[:, features].values
# Separating out the target
y = df.loc[:,['target']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)
```

In [144]:

```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents , columns = ['principal component
1', 'principal component 2'])
```

In [145]:

```
finalDf = pd.concat([principalDf, df[['target']]], axis = 1)
finalDf
```

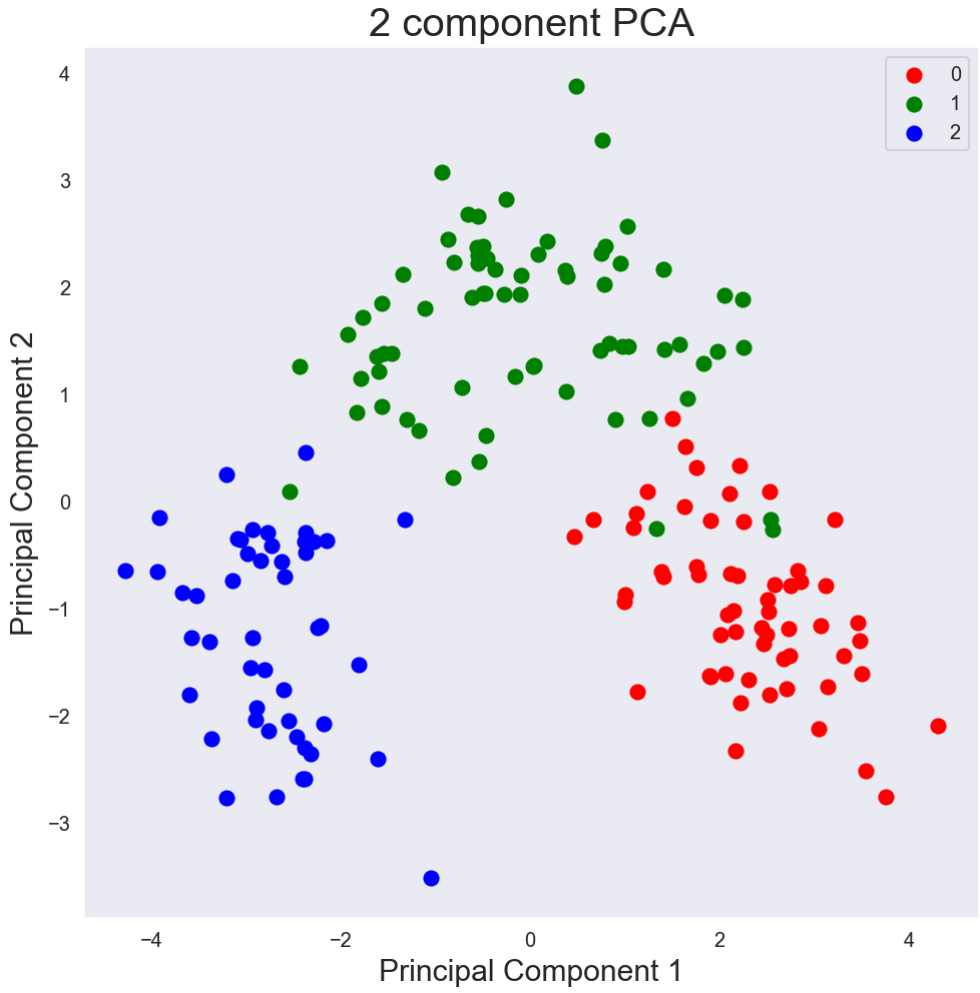
Out[145]:

	principal component 1	principal component 2	target
0	3.316751	-1.443463	0
1	2.209465	0.333393	0
2	2.516740	-1.031151	0
3	3.757066	-2.756372	0
4	1.008908	-0.869831	0
...	...	...	...
173	-3.370524	-2.216289	2
174	-2.601956	-1.757229	2
175	-2.677839	-2.760899	2
176	-2.387017	-2.297347	2
177	-3.208758	-2.768920	2

178 rows × 3 columns

In [146]:

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)

class_list = df['target'].tolist()
myset = set(class_list)
targets = list(myset)

colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['target'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1'], finalDf.loc[indices
ToKeep, 'principal component 2'], c = color, s = 50)
ax.legend(targets)
ax.grid()
```



In [ ]:

```

```