Problem 1: (15 points)

(a) Consider a data object corresponding to a set of nucleotides arranged in a certain order. What is this type of data?

(b) It is desired to partition customers into similar groups on the basis of their demographic profile. Which data mining problem is best suited to this task?

(c) Suppose in problem 1.b, the merchant already knows for some of the customers whether or not they have bought widgets. Which data mining problem would be suited to the task of identifying groups among the remaining customers, who might buy widgets in the future?

Assignment 2

1(a) Requested sort of information is utilized as a part of ~~~~ nucleotide organized portion

(b) The data mining problem which is best suited to this task is outliers. Outliers are the observation points that are distant from other observations. These may be due to wrong data entry or experimental errors. These outliers cause serious problems like wrong input, difficulty in clustering and performing statistical analysis

(c) Progressive part process is employed to ascertain the customer groups subsequently. Though progressive part process, we can recognize the duplicate quickly.

a. The associated task with this dataset is multiclass classification. Change the problem to binary classification and compute the proportion of each class in the binary case? Is this a balanced dataset?

In [36]:

```
# Importing pandas library
import pandas as pd
pd.set_option('display.max_rows', None)

keylist = ['age','sex','cp','trestbps','chol','fbs','restecg','thalach','exang','oldpeak','slope','ca','thal','num']
df = pd.read_csv('C:\\Users\\smrit\\Desktop\\spring\\dm\\hw\\Data\\processed.cleveland.data',sep = ',',engine = 'python',
                 names=keylist)

print("Cleveland Data has been Loaded.. and here is the Data  Count")
print(df.shape)
```

```
Cleveland Data has been Loaded.. and here is the Data  Count
(303, 14)
```

In [37]:

```
import numpy as np
df['target'] = np.where(df['num'] > 0, 1, 0)
print("Changeing the problem to binary classification: target Column shows if the patient has heart Desease or not")
df
```

```
Changeing the problem to binary classification: target Column shows if the patient has heart Desease or not
```

Out[37]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num	target
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	2	1
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0	0
5	56.0	1.0	2.0	120.0	236.0	0.0	0.0	178.0	0.0	0.8	1.0	0.0	3.0	0	0
6	62.0	0.0	4.0	140.0	268.0	0.0	2.0	160.0	0.0	3.6	3.0	2.0	3.0	3	1
7	57.0	0.0	4.0	120.0	354.0	0.0	0.0	163.0	1.0	0.6	1.0	0.0	3.0	0	0
8	63.0	1.0	4.0	130.0	254.0	0.0	2.0	147.0	0.0	1.4	2.0	1.0	7.0	2	1
9	53.0	1.0	4.0	140.0	203.0	1.0	2.0	155.0	1.0	3.1	3.0	0.0	7.0	1	1

In [3]:

```
print("Computing the proportion of each class in the binary case")
#df.groupby('target').size()
dfpercent=df.copy()
dfpercent = dfpercent.groupby(['num']).size().reset_index(name='counts')
dfpercent['percent'] = (dfpercent['counts'] /dfpercent['counts'].sum()) * 100

#print("We can see this is not a balanced data set as  target class has an uneven distribution of observations")
dfpercent
```

```
Computing the proportion of each class in the binary case
```

Out[3]:

	num	counts	percent
0	0	164	54.125413
1	1	55	18.151815
2	2	36	11.881188
3	3	35	11.551155
4	4	13	4.290429

We can see this **is not a balanced** data set as target class has an uneven distribution of observations

b. Remove all patients that have any missing values in their records, how many patients do you have now?

In [4]:
```
DF=df[df.isin(['?']).any(axis=1)]
print("set of records which has missing data,i.e represented as ?")
DF
```

```
set of records which has missing data,i.e represented as ?
```

Out[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num	target
87	53.0	0.0	3.0	128.0	216.0	0.0	2.0	115.0	0.0	0.0	1.0	0.0	?	0	0
166	52.0	1.0	3.0	138.0	223.0	0.0	0.0	169.0	0.0	0.0	1.0	?	3.0	0	0
192	43.0	1.0	4.0	132.0	247.0	1.0	2.0	143.0	1.0	0.1	2.0	?	7.0	1	1
266	52.0	1.0	4.0	128.0	204.0	1.0	0.0	156.0	1.0	1.0	2.0	0.0	?	2	1
287	58.0	1.0	2.0	125.0	220.0	0.0	0.0	144.0	0.0	0.4	2.0	?	7.0	0	0
302	38.0	1.0	3.0	138.0	175.0	0.0	0.0	173.0	0.0	0.0	1.0	?	3.0	0	0

In [5]:
```
dfmissing = df[(df.thal != '?') & (df.ca !=  '?')]
```

In [6]:
```
dfmissing
```

Out[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num	target
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	2	1
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0	0
5	56.0	1.0	2.0	120.0	236.0	0.0	0.0	178.0	0.0	0.8	1.0	0.0	3.0	0	0
6	62.0	0.0	4.0	140.0	268.0	0.0	2.0	160.0	0.0	3.6	3.0	2.0	3.0	3	1
7	57.0	0.0	4.0	120.0	354.0	0.0	0.0	163.0	1.0	0.6	1.0	0.0	3.0	0	0
8	63.0	1.0	4.0	130.0	254.0	0.0	2.0	147.0	0.0	1.4	2.0	1.0	7.0	2	1
9	53.0	1.0	4.0	140.0	203.0	1.0	2.0	155.0	1.0	3.1	3.0	0.0	7.0	1	1
10	57.0	1.0	4.0	140.0	192.0	0.0	0.0	148.0	0.0	0.4	2.0	0.0	6.0	0	0

In [7]:
```
print(dfmissing.shape)
```

```
(297, 15)
```

I don't wanted to mess up the perfectly loaded original data so I have created another dataframe where I am keeping only rows which does not have ? as a value. I am getting row count as 297.

c. Now, impute missing values by mean values of corresponding attributes. Report how this imputation affected the overall distribution of corresponding attributes?

```
#get mean from 297 dataset
dfmissing["thal"] = pd.to_numeric(dfmissing["thal"])
dfmissing["ca"] = pd.to_numeric(dfmissing["ca"])

thal_mean_value=dfmissing['thal'].mean()
ca_mean_value=dfmissing['ca'].mean()

print("Thal Column Mean Value:",thal_mean_value)
print("ca Column Mean Value:",ca_mean_value)

#replace ? with mean
df["thal"].replace({"?": thal_mean_value}, inplace=True)
df["ca"].replace({"?": ca_mean_value}, inplace=True)

df["thal"]= pd.to_numeric(df["thal"])
df["ca"] = pd.to_numeric(df["ca"])

#rounding the mean value
df["ca"]=df["ca"].apply(np.ceil)
df["thal"]=df["thal"].apply(np.ceil)
df
```

```
Thal Column Mean Value: 4.730639730639731
ca Column Mean Value: 0.6767676767676768
```

]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num	target
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	2	1
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0	0
5	56.0	1.0	2.0	120.0	236.0	0.0	0.0	178.0	0.0	0.8	1.0	0.0	3.0	0	0
6	62.0	0.0	4.0	140.0	268.0	0.0	2.0	160.0	0.0	3.6	3.0	2.0	3.0	3	1
7	57.0	0.0	4.0	120.0	354.0	0.0	0.0	163.0	1.0	0.6	1.0	0.0	3.0	0	0
8	63.0	1.0	4.0	130.0	254.0	0.0	2.0	147.0	0.0	1.4	2.0	1.0	7.0	2	1
9	53.0	1.0	4.0	140.0	203.0	1.0	2.0	155.0	1.0	3.1	3.0	0.0	7.0	1	1
10	57.0	1.0	4.0	140.0	192.0	0.0	0.0	148.0	0.0	0.4	2.0	0.0	6.0	0	0
11	56.0	0.0	2.0	140.0	294.0	0.0	2.0	153.0	0.0	1.3	2.0	0.0	3.0	0	0
12	56.0	1.0	3.0	130.0	256.0	1.0	2.0	142.0	1.0	0.6	2.0	1.0	6.0	2	1
13	44.0	1.0	2.0	120.0	263.0	0.0	0.0	173.0	0.0	0.0	1.0	0.0	7.0	0	0
14	52.0	1.0	3.0	172.0	199.0	1.0	0.0	162.0	0.0	0.5	1.0	0.0	7.0	0	0
15	57.0	1.0	3.0	150.0	168.0	0.0	0.0	174.0	0.0	1.6	1.0	0.0	3.0	0	0
16	48.0	1.0	2.0	110.0	229.0	0.0	0.0	168.0	0.0	1.0	3.0	0.0	7.0	1	1
17	54.0	1.0	4.0	140.0	239.0	0.0	0.0	160.0	0.0	1.2	1.0	0.0	3.0	0	0
18	48.0	0.0	3.0	130.0	275.0	0.0	0.0	139.0	0.0	0.2	1.0	0.0	3.0	0	0
19	49.0	1.0	2.0	130.0	266.0	0.0	0.0	171.0	0.0	0.6	1.0	0.0	3.0	0	0

Statistics on 297 rows (after removing missing rows)

In [43]:
```
round(100*(dfmissing.isnull().sum()/len(dfmissing.index)),2)
dfmissing.describe()
```

Out[43]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	
count	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.0
mean	54.542088	0.676768	3.158249	131.693603	247.350168	0.144781	0.996633	149.599327	0.326599	1.055556	1.602694	0.676768	4.7
std	9.049736	0.468500	0.964859	17.762806	51.997583	0.352474	0.994914	22.941562	0.469761	1.166123	0.618187	0.938965	1.9
min	29.000000	0.000000	1.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	1.000000	0.000000	3.0
25%	48.000000	0.000000	3.000000	120.000000	211.000000	0.000000	0.000000	133.000000	0.000000	0.000000	1.000000	0.000000	3.0
50%	56.000000	1.000000	3.000000	130.000000	243.000000	0.000000	1.000000	153.000000	0.000000	0.800000	2.000000	0.000000	3.0
75%	61.000000	1.000000	4.000000	140.000000	276.000000	0.000000	2.000000	166.000000	1.000000	1.600000	2.000000	1.000000	7.0
max	77.000000	1.000000	4.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	3.000000	3.000000	7.0

Statistics on 303 rows after imputation

In [42]:
```
round(100*(df.isnull().sum()/len(df.index)),2)
df.describe()
```

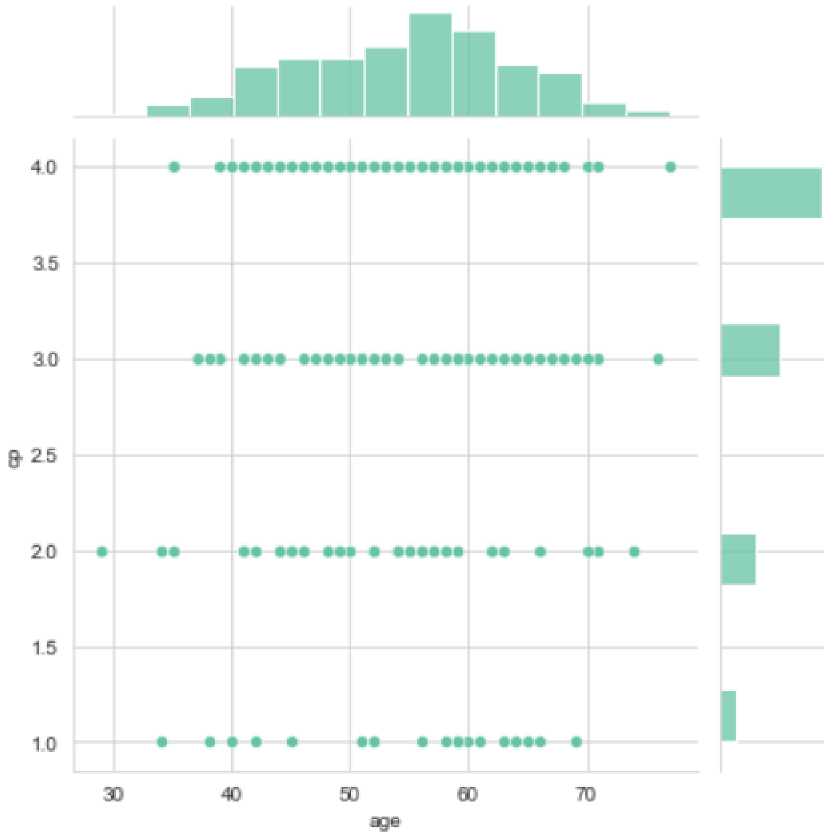
Out[42]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0
mean	54.438944	0.679868	3.158416	131.689769	246.693069	0.148515	0.990099	149.607261	0.326733	1.039604	1.600660	0.676568	4.7
std	9.038662	0.467299	0.960126	17.599748	51.776918	0.356198	0.994971	22.875003	0.469794	1.161075	0.616226	0.931963	1.9
min	29.000000	0.000000	1.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	1.000000	0.000000	3.0
25%	48.000000	0.000000	3.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	3.0
50%	56.000000	1.000000	3.000000	130.000000	241.000000	0.000000	1.000000	153.000000	0.000000	0.800000	2.000000	0.000000	3.0
75%	61.000000	1.000000	4.000000	140.000000	275.000000	0.000000	2.000000	166.000000	1.000000	1.600000	2.000000	1.000000	7.0
max	77.000000	1.000000	4.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	3.000000	3.000000	7.0

d. Draw a scatter plot and explain the relationship between chest pain type and age?

```
def chng(sex):
    if sex == 0:
        return 'female'
    else:
        return 'male'
df['sex'] = df['sex'].apply(chng)

def chng2(target):
    if target == 0:
        return 'Heart Disease'
    else:
        return 'No Heart Disease'
df['target'] = df['target'].apply(chng2)
```

```
#Draw a scatter plot and explain the relationship between chest pain type and age
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style('whitegrid')
sns.set_palette('Set2')
sns.jointplot('age','cp',df)
plt.show()
```

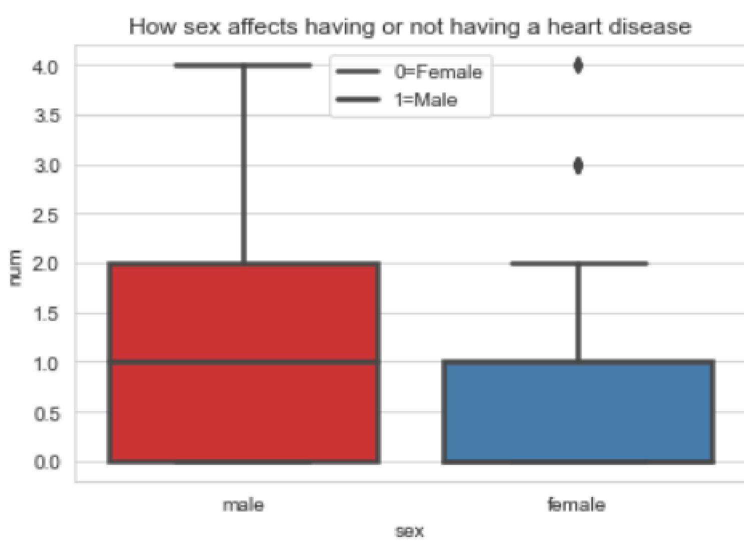


As per the plot trend says most people within range 40-70 has heart disease .most density of dots we can see above cp=2, so as age increases the cp severity also increases.

e. How sex affects having or not having a heart disease? Draw a box plot and explain.

```
#data Preparation for plotting graphs
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style('whitegrid')
sns.set_palette('Set2')

ax = sns.boxplot(x="sex", y="num",
                 data=df, palette="Set1",  linewidth=2.5)
plt.legend(labels=["0=Female","1=Male"])
plt.title('How sex affects having or not having a heart disease')
```

```
Text(0.5, 1.0, 'How sex affects having or not having a heart disease')
```



As per the above plot we can see in the Cleveland data set mostly males has heart disease, females who has the disease are almost half and the severity of the disease is also lesser than men.

f. Generate 6 random samples (without replacement) of size 50 and answer the following:

i. What the proportion of each class in each sample? Is each sample a balanced dataset?

ii. How sex affects having or not having a heart disease in each sample? Draw a box plot.

```
df1 = df1.groupby(['num']).size().reset_index(name='counts')
df1['percent'] = (df1['counts'] /df1['counts'].sum()) * 100
df1['Sample']='Sample 1'

df2 = df2.groupby(['num']).size().reset_index(name='counts')
df2['percent'] = (df2['counts'] /df2['counts'].sum()) * 100
df2['Sample']='Sample 2'

df3 = df3.groupby(['num']).size().reset_index(name='counts')
df3['percent'] = (df3['counts'] /df3['counts'].sum()) * 100
df3['Sample']='Sample 3'

df4 = df4.groupby(['num']).size().reset_index(name='counts')
df4['percent'] = (df4['counts'] /df4['counts'].sum()) * 100
df4['Sample']='Sample 4'

df5 = df5.groupby(['num']).size().reset_index(name='counts')
df5['percent'] = (df5['counts'] /df5['counts'].sum()) * 100
df5['Sample']='Sample 5'

df6 = df6.groupby(['num']).size().reset_index(name='counts')
df6['percent'] = (df6['counts'] /df6['counts'].sum()) * 100
df6['Sample']='Sample 6'


display(df1, df2, df3, df4,df5,df6)
```

	num	counts	percent	Sample
0	0	28	56.0	Sample 1
1	1	7	14.0	Sample 1
2	2	7	14.0	Sample 1
3	3	6	12.0	Sample 1
4	4	2	4.0	Sample 1

	num	counts	percent	Sample
0	0	30	60.0	Sample 2
1	1	7	14.0	Sample 2
2	2	5	10.0	Sample 2
3	3	5	10.0	Sample 2
4	4	3	6.0	Sample 2

	num	counts	percent	Sample
0	0	22	44.0	Sample 3
1	1	7	14.0	Sample 3
2	2	11	22.0	Sample 3
3	3	8	16.0	Sample 3
4	4	2	4.0	Sample 3

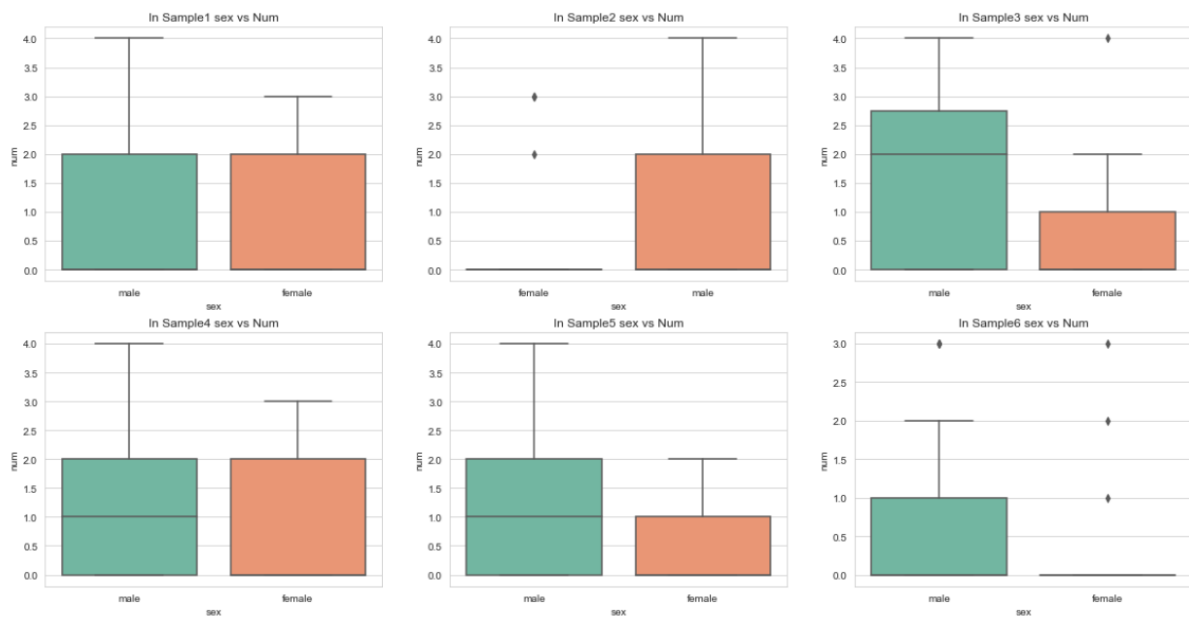
	num	counts	percent	Sample
0	0	23	46.0	Sample 4
1	1	6	12.0	Sample 4
2	2	9	18.0	Sample 4
3	3	10	20.0	Sample 4
4	4	2	4.0	Sample 4

	num	counts	percent	Sample
0	0	27	54.0	Sample 5
1	1	9	18.0	Sample 5
2	2	9	18.0	Sample 5
3	3	4	8.0	Sample 5
4	4	1	2.0	Sample 5

	num	counts	percent	Sample
0	0	32	64.0	Sample 6
1	1	10	20.0	Sample 6
2	2	4	8.0	Sample 6
3	3	4	8.0	Sample 6

We can see this **is not a balanced** data set as target class has an uneven distribution of observations

```python
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

#create six samples of size 50
df1 = df.sample(n=50)
df2 = df.sample(n=50)
df3 = df.sample(n=50)
df4 = df.sample(n=50)
df5 = df.sample(n=50)
df6 = df.sample(n=50)

fig, axes = plt.subplots(2, 3, figsize=(20, 10))
sns.boxplot(ax=axes[0, 0], data=df1, x='sex', y='num').set(title='In Sample1 sex vs Num')
sns.boxplot(ax=axes[0, 1], data=df2,  x='sex', y='num').set(title='In Sample2 sex vs Num')
sns.boxplot(ax=axes[0, 2], data=df3,  x='sex', y='num').set(title='In Sample3 sex vs Num')
sns.boxplot(ax=axes[1, 0], data=df4,  x='sex', y='num').set(title='In Sample4 sex vs Num')
sns.boxplot(ax=axes[1, 1], data=df5,  x='sex', y='num').set(title='In Sample5 sex vs Num')
sns.boxplot(ax=axes[1, 2], data=df6,  x='sex', y='num').set(title='In Sample6 sex vs Num')
```



Out of six sample we could see mostly men has heart disease and in average the severity of the disease it 2.so we can see this trend is matching with other boxplot which we did on the overall data set.

Problem 3: (10 points)

You are given a set of m objects that is divided into K groups, where the i-th group is of size m_i . If the goal is to obtain a sample of size n < m , what is the difference between the following two sampling schemes? (Assume sampling with replacement.)

a. We randomly select $n * m_i$ /m elements from each group.

b. We randomly select n elements from the data set, without regard for the group to which an object belongs.

3)(a) We randomly select $n \times m$ 1/m elements for each group.

It is a proportional sampling. which is proportionate - that's the sample form every cluster is proportional to its size relative to the whole variety of objects.

(b) We randomly select n elements from the dataset, without regard for the group to which an object belongs. It may be a easy random sampling theme. proportional sampling generally has two benefits in the case Once the objects in each group are undiversified

(i) The primary is that we tend to are assured of a sample form every cluster, which might be accustomed estimate numerous statistical parameters of that cluster whenever the corresponding sample size is large enough.

ii) The second advantage is ~~tat~~ that the variance of the sample distribation is always smaller than the variance of the easy sampling

→ Since the latter has consointly to incorporate the variance between the various teams.

Hence stratified sampling is generally more correct than simple random sampling.

Problem 4: (20 points)

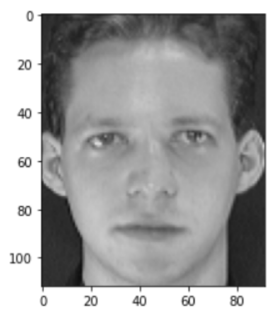
Download the image hw2_2022_problem4_Face.pgm from the class homework data folder. Find a PCA package and use it to compute eigenvectors and eigenvalues for this image.

a. Compute 2, 5, and 10 principal components and show original and the resulting images.

b. What is the minimal number of principal components needed to retain 80% of data variance?

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import cv2
from scipy.stats import stats
import matplotlib.image as mpimg
img = cv2.cvtColor(cv2.imread('C:\\Users\\smrit\\Desktop\\spring\\dm\\hw\\Data\\hw2_2022_problem4_Face.pbm'), cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.show()
```

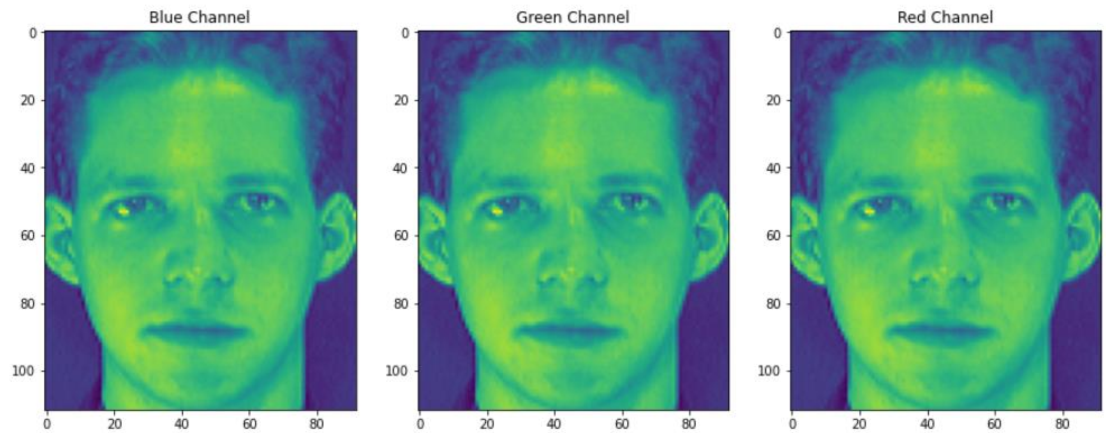
```
img = cv2.cvtColor(cv2.imread('C:\\Users\\smrit\\Desktop\\spring\\dm\\hw\\Data\\hw2_2022_problem4_Face.pbm'), cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.show()
```




```
img.shape
```

```
(112, 92, 3)
```

```python
#Splitting into channels
blue,green,red = cv2.split(img)
# Plotting the images
fig = plt.figure(figsize = (15, 7.2))
fig.add_subplot(131)
plt.title("Blue Channel")
plt.imshow(blue)
fig.add_subplot(132)
plt.title("Green Channel")
plt.imshow(green)
fig.add_subplot(133)
plt.title("Red Channel")
plt.imshow(red)
#plt.show()
```




```python
#Let's verify the data of the blue channel:
blue_temp_df = pd.DataFrame(data = blue)
blue_temp_df
```

	0	1	2	3	4	5	6	7	8	9	...	82	83	84	85	86	87	88	89	90	91
0	48	49	45	47	49	57	39	42	53	49	...	58	46	41	43	56	55	51	56	56	54
1	45	52	39	46	56	45	39	47	48	40	...	57	47	38	39	39	51	53	52	50	51
2	45	50	42	51	51	45	40	48	44	37	...	54	52	47	41	33	49	51	48	53	50
3	49	46	47	47	50	47	42	45	40	44	...	72	57	46	39	35	31	43	43	50	51
4	46	46	47	48	48	44	43	44	60	54	...	72	64	43	46	31	34	38	41	53	48
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
107	49	47	49	47	52	50	48	51	47	49	...	37	42	38	45	45	42	50	41	48	49
108	45	52	43	52	48	49	51	52	46	47	...	35	39	43	43	47	42	48	45	48	45
109	50	48	50	46	50	47	51	50	48	46	...	40	41	39	41	46	46	44	45	46	46
110	45	54	49	46	50	50	47	46	50	47	...	37	41	39	42	43	50	45	46	47	47
111	51	51	51	45	52	50	47	48	46	52	...	39	44	40	41	49	42	44	47	46	46

112 rows × 92 columns

PCA 2 Computation results:

```
#I will divide all the data of all channels by 255 so that the data is scaled between 0 and 1.
df_blue = blue/255
df_green = green/255
df_red = red/255
```

```
#Fit and transform the data in PCA
pca_b = PCA(n_components=2)
pca_b.fit(df_blue)
trans_pca_b = pca_b.transform(df_blue)
pca_g = PCA(n_components=2)
pca_g.fit(df_green)
trans_pca_g = pca_g.transform(df_green)
pca_r = PCA(n_components=2)
pca_r.fit(df_red)
trans_pca_r = pca_r.transform(df_red)
```

```
print(trans_pca_b.shape)
print(trans_pca_r.shape)
print(trans_pca_g.shape)
```

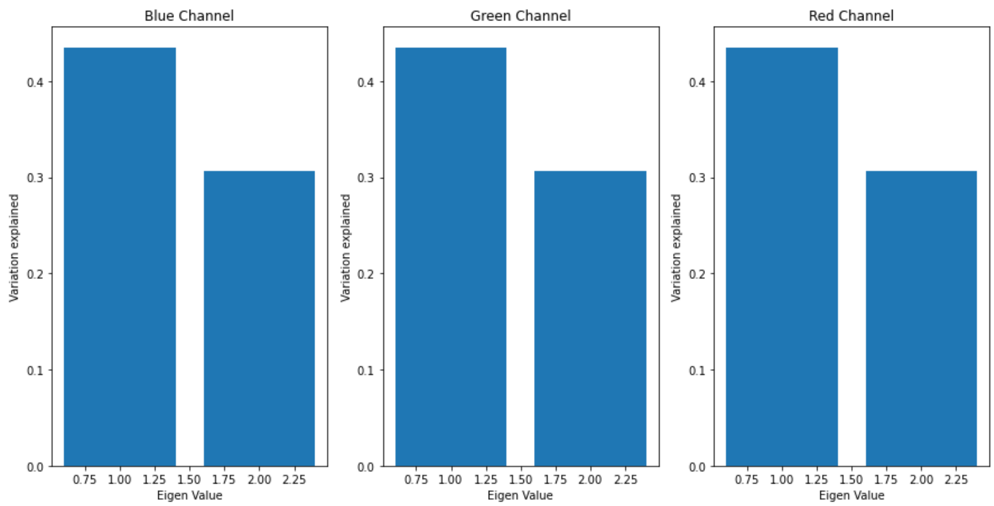
```
(112, 2)
(112, 2)
(112, 2)
```

```
#Let's check the sum of explained variance ratios of the 2 PCA components (i.e. most dominated 2 Eigenvalues) for each channel.
print(f"Blue Channel : {sum(pca_b.explained_variance_ratio_)}")
print(f"Green Channel: {sum(pca_g.explained_variance_ratio_)}")
print(f"Red Channel  : {sum(pca_r.explained_variance_ratio_)}")
```

```
Blue Channel : 0.7412615172183636
Green Channel: 0.7412615172183636
Red Channel  : 0.7412615172183636
```

```
#Let's plot bar charts to check the explained variance ratio by each Eigenvalues separately for each of the 3 channels
fig = plt.figure(figsize = (15, 7.2))
fig.add_subplot(131)
plt.title("Blue Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,3)),pca_b.explained_variance_ratio_)
fig.add_subplot(132)
plt.title("Green Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,3)),pca_g.explained_variance_ratio_)
fig.add_subplot(133)
plt.title("Red Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,3)),pca_r.explained_variance_ratio_)
plt.show()
```

In [10]:
```
#Let's plot bar charts to check the explained variance ratio by each Eigenvalues separately for each of the 3 channels
fig = plt.figure(figsize = (15, 7.2))
fig.add_subplot(131)
plt.title("Blue Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,3)),pca_b.explained_variance_ratio_)
fig.add_subplot(132)
plt.title("Green Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,3)),pca_g.explained_variance_ratio_)
fig.add_subplot(133)
plt.title("Red Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,3)),pca_r.explained_variance_ratio_)
plt.show()
```



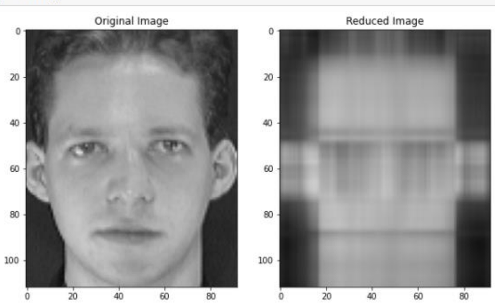
In [11]:
```
#Reconstruct the image and visualize
b_arr = pca_b.inverse_transform(trans_pca_b)
g_arr = pca_g.inverse_transform(trans_pca_g)
r_arr = pca_r.inverse_transform(trans_pca_r)
print(b_arr.shape, g_arr.shape, r_arr.shape)
```

```
(112, 92) (112, 92) (112, 92)
```

In [12]:
```
img_reduced= (cv2.merge((b_arr, g_arr, r_arr)))
print(img_reduced.shape)
```

```
(112, 92, 3)
```

In [13]:
```
fig = plt.figure(figsize = (10, 7.2))
fig.add_subplot(121)
plt.title("Original Image")
plt.imshow(img)
fig.add_subplot(122)
plt.title("Reduced Image")
plt.imshow(img_reduced)
plt.show()
```



PCA computation 5:

In [6]:
```
#I will divide all the data of all channels by 255 so that the data is scaled between 0 and 1.
df_blue = blue/255
df_green = green/255
df_red = red/255
```

In [7]:
```
#Fit and transform the data in PCA
pca_b = PCA(n_components=5)
pca_b.fit(df_blue)
trans_pca_b = pca_b.transform(df_blue)
pca_g = PCA(n_components=5)
pca_g.fit(df_green)
trans_pca_g = pca_g.transform(df_green)
pca_r = PCA(n_components=5)
pca_r.fit(df_red)
trans_pca_r = pca_r.transform(df_red)
```

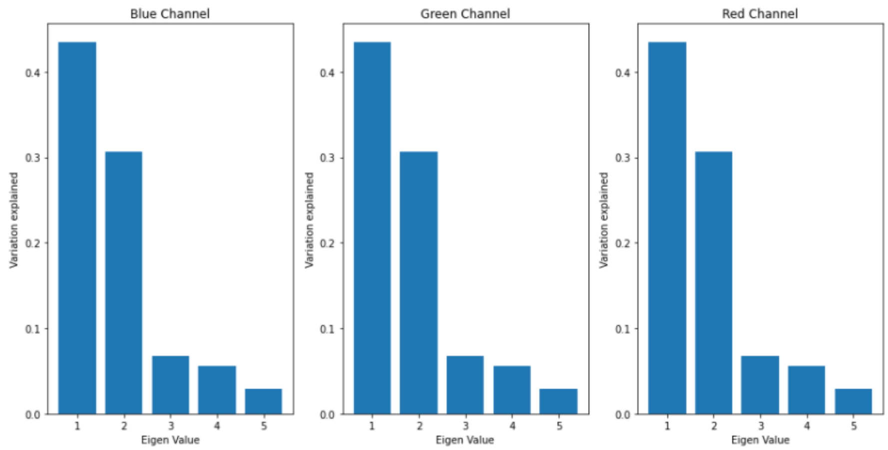
In [8]:
```
print(trans_pca_b.shape)
print(trans_pca_r.shape)
print(trans_pca_g.shape)
```

```
(112, 5)
(112, 5)
(112, 5)
```

In [9]:
```
#Let's check the sum of explained variance ratios of the 2 PCA components (i.e. most dominated 2 Eigenvalues) for each channel.
print(f"Blue Channel : {sum(pca_b.explained_variance_ratio_)}")
print(f"Green Channel: {sum(pca_g.explained_variance_ratio_)}")
print(f"Red Channel  : {sum(pca_r.explained_variance_ratio_)}")
```

```
Blue Channel : 0.8937039422948817
Green Channel: 0.8937039422948817
Red Channel  : 0.8937039422948817
```

In [10]:
```
#Let's plot bar charts to check the explained variance ratio by each Eigenvalues separately for each of the 3 channels
fig = plt.figure(figsize = (15, 7.2))
fig.add_subplot(131)
plt.title("Blue Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,6)),pca_b.explained_variance_ratio_)
fig.add_subplot(132)
plt.title("Green Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,6)),pca_g.explained_variance_ratio_)
fig.add_subplot(133)
plt.title("Red Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,6)),pca_r.explained_variance_ratio_)
plt.show()
```



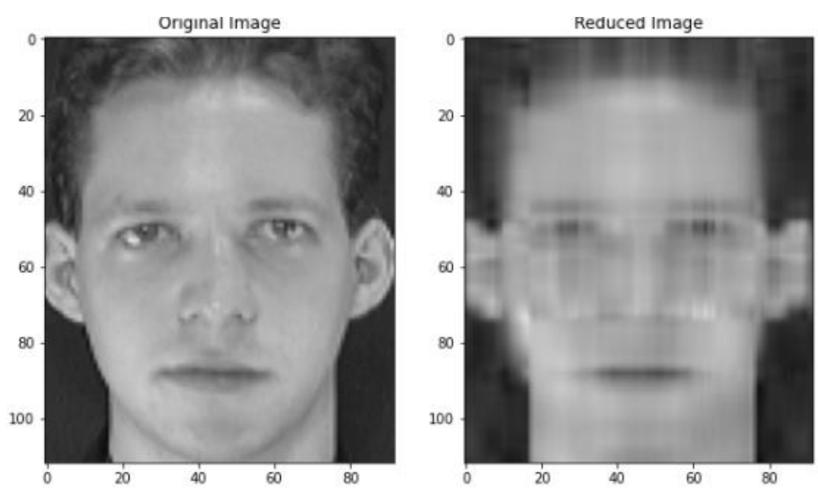
In [11]:
```
#Reconstruct the image and visualize
b_arr = pca_b.inverse_transform(trans_pca_b)
g_arr = pca_g.inverse_transform(trans_pca_g)
r_arr = pca_r.inverse_transform(trans_pca_r)
print(b_arr.shape, g_arr.shape, r_arr.shape)
```

```
(112, 92) (112, 92) (112, 92)
```

In [12]:
```
img_reduced= (cv2.merge((b_arr, g_arr, r_arr)))
print(img_reduced.shape)
```

```
(112, 92, 3)
```

In [13]:
```
fig = plt.figure(figsize = (10, 7.2))
fig.add_subplot(121)
plt.title("Original Image")
plt.imshow(img)
fig.add_subplot(122)
plt.title("Reduced Image")
plt.imshow(img_reduced)
plt.show()
```



PCA computation for 10:

In [7]:
```
#Fit and transform the data in PCA
pca_b = PCA(n_components=10)
pca_b.fit(df_blue)
trans_pca_b = pca_b.transform(df_blue)
pca_g = PCA(n_components=10)
pca_g.fit(df_green)
trans_pca_g = pca_g.transform(df_green)
pca_r = PCA(n_components=10)
pca_r.fit(df_red)
trans_pca_r = pca_r.transform(df_red)
```

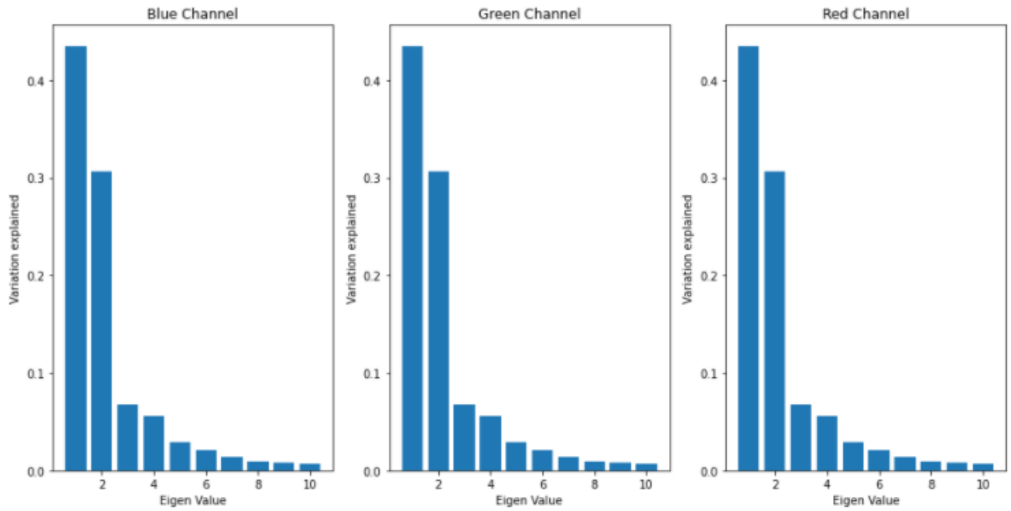
In [8]:
```
print(trans_pca_b.shape)
print(trans_pca_r.shape)
print(trans_pca_g.shape)
```

```
(112, 10)
(112, 10)
(112, 10)
```

In [9]:
```
#Let's check the sum of explained variance ratios of the 50 PCA components (i.e. most dominated 2 Eigenvalues) for each channel.
print(f"Blue Channel : {sum(pca_b.explained_variance_ratio_)}")
print(f"Green Channel: {sum(pca_g.explained_variance_ratio_)}")
print(f"Red Channel  : {sum(pca_r.explained_variance_ratio_)}")
```

```
Blue Channel : 0.9517910734021818
Green Channel: 0.9517910734021818
Red Channel  : 0.9517910734021818
```

In [10]:
```
#Let's plot bar charts to check the explained variance ratio by each Eigenvalues separately for each of the 3 channels
fig = plt.figure(figsize = (15, 7.2))
fig.add_subplot(131)
plt.title("Blue Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,11)),pca_b.explained_variance_ratio_)
fig.add_subplot(132)
plt.title("Green Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,11)),pca_g.explained_variance_ratio_)
fig.add_subplot(133)
plt.title("Red Channel")
plt.ylabel('Variation explained')
plt.xlabel('Eigen Value')
plt.bar(list(range(1,11)),pca_r.explained_variance_ratio_)
plt.show()
```



In [11]:
```
#Reconstruct the image and visualize
b_arr = pca_b.inverse_transform(trans_pca_b)
g_arr = pca_g.inverse_transform(trans_pca_g)
r_arr = pca_r.inverse_transform(trans_pca_r)
print(b_arr.shape, g_arr.shape, r_arr.shape)
```

```
(112, 92) (112, 92) (112, 92)
```

In [12]:
```
img_reduced= (cv2.merge((b_arr, g_arr, r_arr)))
print(img_reduced.shape)
```

```
(112, 92, 3)
```

In [13]:
```
fig = plt.figure(figsize = (10, 7.2))
fig.add_subplot(121)
plt.title("Original Image")
plt.imshow(img)
fig.add_subplot(122)
plt.title("Reduced Image")
plt.imshow(img_reduced)
plt.show()
```

We can see we need minimum PCA 2 to retain 74% of data variance and having minimum 5 PCA we are able to retain 89% of data variance. So retail 80% of data variance the minimum PCA needed will be between 2-5, after running the above program for PCA 3 we are able to get 80% data variance. Here is the results:

```
In [17]: #Let's check the sum of explained variance ratios of the 2 PCA components
import math
#Fit and transform the data in PCA
pca_b = PCA(n_components=3)
pca_b.fit(df_blue)
trans_pca_b = pca_b.transform(df_blue)
pca_g = PCA(n_components=3)
pca_g.fit(df_green)
trans_pca_g = pca_g.transform(df_green)
pca_r = PCA(n_components=3)
pca_r.fit(df_red)
trans_pca_r = pca_r.transform(df_red)
print(f"Blue Channel : {math.floor(sum(pca_b.explained_variance_ratio_)*100)}")
print(f"Green Channel: {math.floor(sum(pca_g.explained_variance_ratio_)*100)}")
print(f"Red Channel  : {math.floor(sum(pca_r.explained_variance_ratio_)*100)}")

Blue Channel : 80
Green Channel: 80
Red Channel  : 80
```

Problem 5: (30 points)

The decision-makers at GymX would like to improve their services using data mining and machine learning techniques to better understand their customers. They have a large database that contains many fields such as customer_id, customer_name, age, sex, height, weight, membership_type, diet_restrictions, and more. The problem is that the database has many missing data, because most customer do not fill all necessary fields when they join the gym. This problem will affect their customer analysis. Help GymX to solve their problem. Download hw2_2022_problem5_GymX.cvs dataset from the class homework data folder. The dataset contains the following attributes:

- Customer ID
- Customer Name
- Age
- Sex (male = 1, female = 0)
- Height in feet
- Weight in pounds
- Membership type (adult, youth, or kids)

a. Report the number of missing values in each feature.

```
In [20]: #gymx Question 4 implementation
import pandas as pd
dfgymx = pd.read_csv('C:\\Users\\smrit\\Desktop\\spring\\dm\\hw\\Data\\hw2_2022_problem5_GymX.csv')
#Total Row Count
print(dfgymx.shape)

(4998, 7)

In [21]: dfgymx.isna().sum()

Out[21]: customer_id           0
customer_name         0
age                   0
sex                   0
height             2976
weight             2817
membership_type    1455
dtype: int64
```

b. Using a naive solution could solve the missing data problem. What are the advantages/disadvantages of this solution?

```
In [22]: #naïve approaches for handling missing data
         #Dropping rows with missing values
         dfset1 = dfgymx.copy()
         dfset1.dropna(how='any',inplace=True)

In [23]: dfset1.shape

Out[23]: (671, 7)
```

The naive approach I have used in this case **is dropping the rows having null values.** The rows which are having one or more columns values as null can also be dropped. The **advantages is** removal of all missing values creates a robust model.

Disadvantages : Loss of a lot of information.Works weakly if the percentage of missing values is excessive in comparison to the complete dataframe.

c. Propose a better solution to solve the missing data problem.

- **Data bining for Membership type:**
  I have taken a careful look at the data and found we can identify the persons membership_type based on their age. So I used data bining in this case

```
In [36]: #Data bining for Menbership_type
         dfnew = dfgymx.copy()
         bins= [6,12,17,170]
         labels = ['kids','youth','Adult']
         dfnew['membership_type'] = pd.cut(dfnew['age'], bins=bins, labels=labels, right=False)
         dfnew.isna().sum()

Out[36]: customer_id          0
         customer_name        0
         age                  0
         sex                  0
         height            2976
         weight            2817
         membership_type      0
         dtype: int64
```

- Next I have updated the missing height and weight : I have grouped height and weight with memship type and got the mean, next I updated NAN with mean value.

```
In [27]: dfnew['height'] = dfnew['height'].fillna(dfnew.groupby('membership_type')['height'].transform('mean'))
         dfnew['weight'] = dfnew['weight'].fillna(dfnew.groupby('membership_type')['weight'].transform('mean'))
```

```
In [28]: dfnew.isna().sum()
```

```
Out[28]: customer_id        0
         customer_name      0
         age                0
         sex                0
         height             0
         weight             0
         membership_type    0
         dtype: int64
```

```
In [29]: #No Null values in new data frame
         dfnew.isna().sum().sum()
```

```
Out[29]: 0
```

With above approach we can solve missing data problem.

d. Compare results of the naïve handling of missing data vs your better solutions based on:

   i. Plot a histogram of customers' age.

   ii. Plot a histogram of height for all customers and report mean and standard deviation

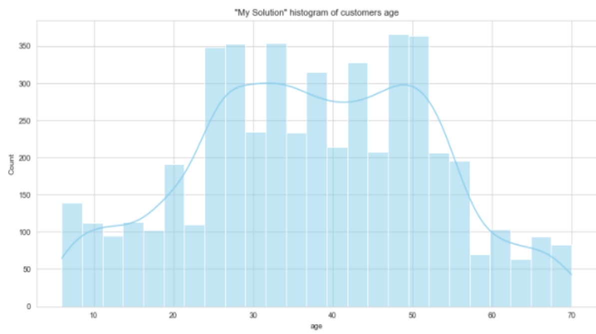
   iii. Plot a histogram of weight for all customers and report mean and standard deviation

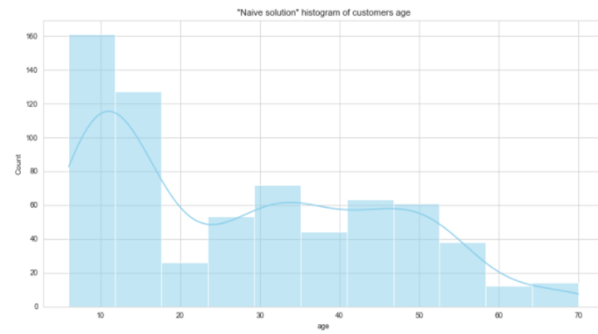
   iv. Create a bar plot that shows the number of customers from each sex from each membership type.

```
# Plot a histogram of customers' age.
#dfnew.hist(column='age')
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns

fig, axes = plt.subplots(4, 2, figsize=(30, 30))
sns.set_style('darkgrid')
sns.histplot(ax=axes[0, 0],color="skyblue",data=dfnew, x="age", kde=True).set(title='"My Solution" histogram of customers age')
sns.histplot(ax=axes[0, 1],color="skyblue",data=dfset1, x="age", kde=True).set(title='"Naive solution" histogram of customers age
sns.histplot(ax=axes[1, 0],color="olive",data=dfnew, x="height",bins=10,kde=True).set(title='"My Solution" histogram of customers
sns.histplot(ax=axes[1, 1],color="olive",data=dfset1, x="height", kde=True).set(title='"Naive solution" histogram of customers he

sns.histplot(ax=axes[2, 0],color="gold",data=dfnew, x="weight", kde=True).set(title='"My Solution" histogram of customers weight'
sns.histplot(ax=axes[2, 1],color="gold",data=dfset1, x="weight", kde=True).set(title='"Naive solution" histogram of customers wei

sns.countplot(ax=axes[3,0],color="teal",data=dfnew, x='membership_type', hue='sex').set(title='"My Solution" number of customers
sns.countplot(ax=axes[3,1],color="teal",data=dfset1, x='membership_type', hue='sex').set(title='"Naive solution" number of custor



plt.show(sns)
```

LHS is My Solution

RHS: Naïve solution

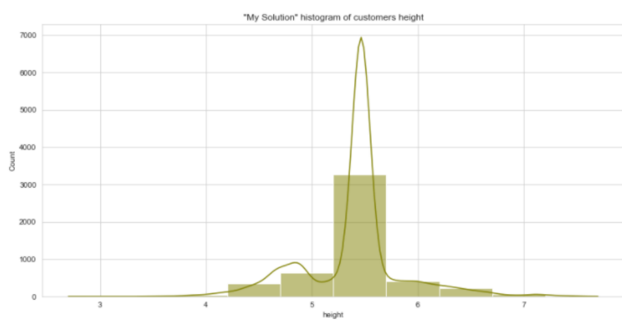


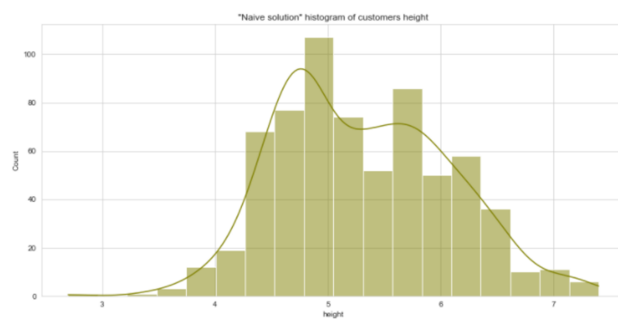


As in my solution we have more record we can see a trend that more customers 30-50.

LHS is My Solution

RHS: Naïve solution

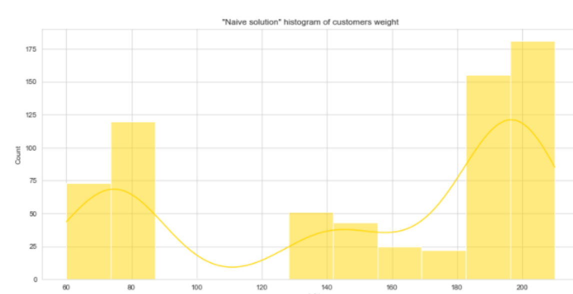




As in my solution we have more record we can see a trend that says most customers has height ranges 5-7.

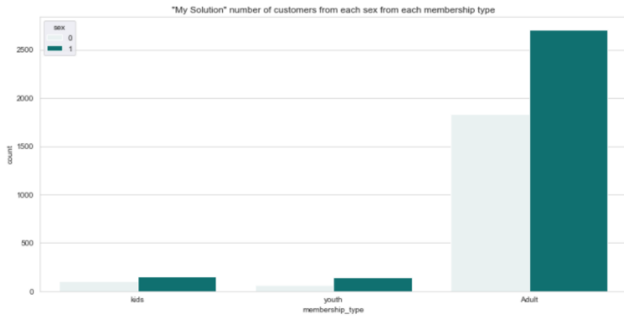
LHS is My Solution

RHS: Naïve solution

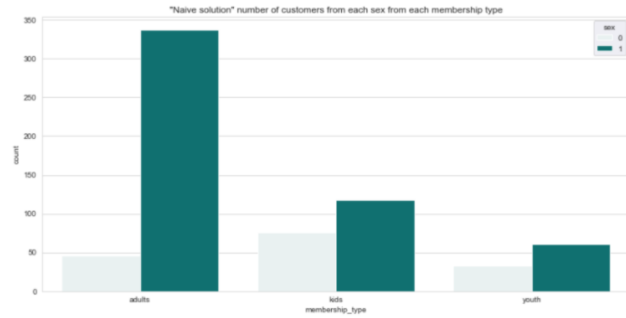


As in my solution we have more record we can see a trend that most customers weight 180-200

LHS is My Solution



RHS: Naïve solution



As in my solution we have more record we can see a trend that most customers are adult.

So, we can see from the above plots that with more data we can answer more questions.as the naïve solution removes all null value rows we are loosing data and could not see trends properly.