Deniz Aytemiz

ECE 5984- Homework 2

Question 1.

Excel file is imported to working directory and passed into pandas dataframe as seen in Figure 1.

```
+ Code  + Text

[529] import pandas as pd
      import numpy as np

[530] excel_file = 'Heart Disease.xlsx'

[531] ##dataframe is variable storing pandas dataframe object, read excel is method of pandas.
      dataframe = pd.read_excel(excel_file)
      dataframe.head()
```

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0	46820.0	63	M	3.0	145.0	233.0	1.0	0.0	150.0	0.0	2.3	0.0	NaN	1	O	1
1	46830.0	37	M	2.0	130.0	250.0	NaN	1.0	187.0	0.0	3.5	0.0	NaN	2	B+	1
2	46840.0	41	F	1.0	130.0	204.0	NaN	0.0	172.0	0.0	1.4	2.0	NaN	2	A+	1
3	46850.0	56	M	1.0	120.0	236.0	NaN	1.0	178.0	0.0	0.8	2.0	NaN	2	B-	1
4	46860.0	57	F	0.0	120.0	354.0	NaN	1.0	163.0	1.0	0.6	2.0	NaN	2	A+	1

```
[532] dataframe.shape

      (303, 16)
```

Figure 1

In Figure2 .describe() function is used to display the descriptive statistics of the dataframe with non-numeric values included.

```
##include all is for displaying non numerical columns too
new_df=dataframe.describe(include='all')
display(new_df)
```

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
count	301.000000	303.000000	302	302.000000	302.000000	300.000000	296.000000	302.000000	268.000000	302.000000	302.000000	301.000000	298.000000	303.000000	298	303.000000
unique	NaN	NaN	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5	NaN
top	NaN	NaN	M	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	O	NaN
freq	NaN	NaN	206	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	115	NaN
mean	48332.657807	54.366337	NaN	0.970199	131.615894	246.590000	0.152027	0.529801	149.130597	0.324503	1.043046	1.398671	0.741611	2.313531	NaN	0.544554
std	877.940533	9.082101	NaN	1.032257	17.566716	51.969652	0.359655	0.525849	22.595484	0.468966	1.161452	0.616872	1.026753	0.612277	NaN	0.498835
min	46820.000000	29.000000	NaN	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	NaN	0.000000
25%	47580.000000	47.500000	NaN	0.000000	120.000000	211.000000	0.000000	0.000000	133.750000	0.000000	0.000000	1.000000	0.000000	2.000000	NaN	0.000000
50%	48340.000000	55.000000	NaN	1.000000	130.000000	241.500000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000	0.000000	2.000000	NaN	1.000000
75%	49090.000000	61.000000	NaN	2.000000	140.000000	275.000000	0.000000	1.000000	165.000000	1.000000	1.600000	2.000000	1.000000	3.000000	NaN	1.000000
max	49840.000000	77.000000	NaN	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	NaN	1.000000

Figure 2

In order to find cardinality, nunique() function is called and each columns' number of unique value is displayed as shown in Figure 3. Then these values are passed into an array called cardinality as shown.

```
[534] #finding cardinality with nunique fnc, which returns the number of non-unique values in the columns
      dataframe.nunique(axis=0, dropna=False)

      member      302
      age          41
      sex           3
      cp            5
      trestbps     50
      chol        153
      fbs           3
      restecg       4
      thalach      88
      exang         3
      oldpeak      41
      slope         4
      ca            6
      thal          4
      bt            6
      target        2
      dtype: int64

[535] cardinality = dataframe.nunique(axis=0, dropna=False).values

[536] ##these values are stored in an array
      cardinality

      array([302,  41,   3,   5,  50, 153,   3,   4,  88,   3,  41,   4,   6,
               4,   6,   2])
```

Figure 3

In Figure 4, a new, empty dataframe is created so all the staticsc will be added here to create final output. Then, with a for loop array elements are matched with the column names of df and this information is kept in data_to_append and it is appended to dataframe resulting in adding the first row to df, which indicates cardinality of each column.

```
[537] #create new empty dataframe
      column_names = ["member", "age", "sex","cp", "trestbps", "chol","fbs", "restecg", "thalach", "exang", "oldpeak", "slope","ca","thal","bt","target" ]

      df = pd.DataFrame(columns = column_names)
      display(df)
```

member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target

```
[538] #the elements of the array are added to data frame as row
      data_to_append = {}
      for i in range(len(df.columns)):
          data_to_append[df.columns[i]] = cardinality[i]
      df = df.append(data_to_append, ignore_index = True)

[539] df
```

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0	302	41	3	5	50	153	3	4	88	3	41	4	6	4	6	2

Figure 4

Second row of df will be the mean values of each column. This information was already in new_df data frame that can be observed in Figure 5.

```
display(new_df)
```

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
count	301.000000	303.000000	302	302.000000	302.000000	300.000000	296.000000	302.000000	268.000000	302.000000	302.000000	301.000000	298.000000	303.000000	298	303.000000
unique	NaN	NaN	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5	NaN
top	NaN	NaN	M	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	O	NaN
freq	NaN	NaN	206	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	115	NaN
mean	48332.657807	54.366337	NaN	0.970199	131.615894	246.590000	0.152027	0.529801	149.130597	0.324503	1.043046	1.398671	0.741611	2.313531	NaN	0.544554
std	877.940533	9.082101	NaN	1.032257	17.566716	51.969652	0.359655	0.525849	22.595484	0.468966	1.161452	0.616872	1.026753	0.612277	NaN	0.498835
min	46820.000000	29.000000	NaN	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	NaN	0.000000
25%	47580.000000	47.500000	NaN	0.000000	120.000000	211.000000	0.000000	0.000000	133.750000	0.000000	0.000000	1.000000	0.000000	2.000000	NaN	0.000000
50%	48340.000000	55.000000	NaN	1.000000	130.000000	241.500000	0.000000	1.000000	152.000000	0.000000	0.800000	1.000000	0.000000	2.000000	NaN	1.000000
75%	49090.000000	61.000000	NaN	2.000000	140.000000	275.000000	0.000000	1.000000	165.000000	1.000000	1.600000	2.000000	1.000000	3.000000	NaN	1.000000
max	49840.000000	77.000000	NaN	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	NaN	1.000000

Figure 5

Therefore the row ‘mean’ in new_df is copied to variable first as seen in Figure 6 and in Figure 7 it is added to the output dataframe df.

```
[541] first = new_df.loc["mean"]
```

```
[542] print(first)
```

```
member      48332.657807
age            54.366337
sex                  NaN
cp              0.970199
trestbps      131.615894
chol             246.59
fbs             0.152027
restecg         0.529801
thalach       149.130597
exang           0.324503
oldpeak         1.043046
slope           1.398671
ca              0.741611
thal            2.313531
bt                   NaN
target          0.544554
Name: mean, dtype: object
```

Figure 6

```
[543] #second row mean is added, it is extarcted from new_df dataframe
      df = df.append(first, ignore_index = True)
```

```
df
```

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0	302	41	3	5	50	153	3	4	88	3	41	4	6	4	6	2
1	48332.657807	54.366337	NaN	0.970199	131.615894	246.59	0.152027	0.529801	149.130597	0.324503	1.043046	1.398671	0.741611	2.313531	NaN	0.544554

Figure 7

For adding the next row which is the median of each row a loop is used. It iterates over the column names of dataframe(initial dataframe with excel file is passed to) and if column name is not ‘sex’ or ‘bt’, since they have no median, data_to_append holds the median value for the specific column under the same column name with dataframe. Hence it can be later appended to df, the output dataframe. Variable i holds the index in this loop. This is displayed in Figure 8 and Figure 9. One can observe in Figure 9, median value for columns ‘sex’ and ‘bt’ is NaN since data_to_append has no columns of ‘sex’ and ‘bt’ so nothing is assigned to them.

```
Name: member, Length: 303, dtype: float64
```

```
data_to_append = {}
i=0
for col in dataframe.columns:
  if col != 'sex' and col != 'bt':
    data_to_append[dataframe.columns[i]] = dataframe[col].median()
  else:
    pass
  i=i+1
df = df.append(data_to_append, ignore_index = True)
```

Figure 8

```
[547] df
```

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0	302	41	3	5	50	153	3	4	88	3	41	4	6	4	6	2
1	48332.657807	54.366337	NaN	0.970199	131.615894	246.59	0.152027	0.529801	149.130597	0.324503	1.043046	1.398671	0.741611	2.313531	NaN	0.544554
2	48340.0	55.0	NaN	1.0	130.0	241.5	0.0	1.0	152.0	0.0	0.8	1.0	0.0	2.0	NaN	1.0

Figure 9

```
df.iloc[2,:]
```

```
member      48340.0
age            55.0
sex             NaN
cp              1.0
trestbps      130.0
chol          241.5
fbs             0.0
restecg         1.0
thalach       152.0
exang           0.0
oldpeak         0.8
slope           1.0
ca              0.0
thal            2.0
bt              NaN
target          1.0
Name: 2, dtype: object
```

Figure 10: df.iloc[2,:] indicates 2 indexed row which is median row

```
[551] dataframe.iloc[:,0]
```

```
0      46820.0
1      46830.0
2      46840.0
3      46850.0
4      46860.0
        ...
298    49800.0
299    49810.0
300    49820.0
301    49830.0
302    49840.0
Name: member, Length: 303, dtype: float64
```

Figure 11: dataframe.iloc[:,0] indicates 0 indexed column which is member column

In Figure 12, number of data that are the same with median value is found. The loop iterates over all columns of the dataframe and if that value is equal to any value in df's second row(median row) it is added to total and total is stored in variable count. Again this value is passed to the corresponding column and stored in data_to_append. Then it is appended to output data frame df.

```
[569] data_to_append = {}
      for i in range(16):
        count = (dataframe.iloc[:,i] == df.iloc[2,i]).sum()
        data_to_append[dataframe.columns[i]]=count
      df = df.append(data_to_append, ignore_index = True)
```

Figure 12: loop for finding n_at_median

[570] df

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0	302	41	3	5	50	153	3	4	88	3	41	4	6	4	6	2
1	48332.657807	54.366337	NaN	0.970199	131.615894	246.59	0.152027	0.529801	149.130597	0.324503	1.043046	1.398671	0.741611	2.313531	NaN	0.544554
2	48340.0	55.0	NaN	1.0	130.0	241.5	0.0	1.0	152.0	0.0	0.8	1.0	0.0	2.0	NaN	1.0
3	1	8	0	50	36	0	251	152	7	204	13	139	170	166	0	165

Figure 13: df with n_at_median added

```
[565] data_to_append

{'age': 8,
 'bt': 0,
 'ca': 170,
 'chol': 0,
 'cp': 50,
 'exang': 204,
 'fbs': 251,
 'member': 1,
 'oldpeak': 13,
 'restecg': 152,
 'sex': 0,
 'slope': 139,
 'target': 165,
 'thal': 166,
 'thalach': 7,
 'trestbps': 36}
```

Figure 14: data_to_append

```
[571] temp=dataframe.mode()
```

```
# has 301 rows since no element of member is repeated more than twice. So we can take the first row only
temp
```

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0	46820.0	58.0	M	0.0	120.0	197.0	0.0	1.0	162.0	0.0	0.0	2.0	0.0	2.0	O	1.0
1	46830.0	NaN	NaN	NaN	NaN	204.0	NaN	NaN	163.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	46840.0	NaN	NaN	NaN	NaN	234.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	46850.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	46860.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
296	49800.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
297	49810.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
298	49820.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
299	49830.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
300	49840.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

301 rows × 16 columns

Figure 15: mode of dataframe

In Figure 15, mode of dataframe is displayed. The reason for multiple rows is that the member column has multiple mode values since no value of the ‘member’ is repeated more than once. Hence we can get the first row of temp and add that to df. This can be observed in Figure 16.

```
[573] df = df.append(temp.iloc[0], ignore_index = True)
```

```
[574] df
```

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0	302	41	3	5	50	153	3	4	88	3	41	4	6	4	6	2
1	48332.657807	54.366337	NaN	0.970199	131.615894	246.59	0.152027	0.529801	149.130597	0.324503	1.043046	1.398671	0.741611	2.313531	NaN	0.544554
2	48340.0	55.0	NaN	1.0	130.0	241.5	0.0	1.0	152.0	0.0	0.8	1.0	0.0	2.0	NaN	1.0
3	1	8	0	50	36	0	251	152	7	204	13	139	170	166	0	165
4	46820.0	58.0	M	0.0	120.0	197.0	0.0	1.0	162.0	0.0	0.0	2.0	0.0	2.0	O	1.0

Figure 16: Updated df

In Figure 17, n_at_mode is found the same way as n_at_median is found in Figure 12.

```
[575] data_to_append = {}
      for i in range(16):
        count = (dataframe.iloc[:,i] == df.iloc[4,i]).sum()
        data_to_append[dataframe.columns[i]]=count
      df = df.append(data_to_append, ignore_index = True)
```

```
[576] df
```

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0	302	41	3	5	50	153	3	4	88	3	41	4	6	4	6	2
1	48332.657807	54.366337	NaN	0.970199	131.615894	246.59	0.152027	0.529801	149.130597	0.324503	1.043046	1.398671	0.741611	2.313531	NaN	0.544554
2	48340.0	55.0	NaN	1.0	130.0	241.5	0.0	1.0	152.0	0.0	0.8	1.0	0.0	2.0	NaN	1.0
3	1	8	0	50	36	0	251	152	7	204	13	139	170	166	0	165
4	46820.0	58.0	M	0.0	120.0	197.0	0.0	1.0	162.0	0.0	0.0	2.0	0.0	2.0	O	1.0
5	1	19	206	142	37	6	251	152	9	204	98	141	170	166	115	165

Figure 17: n_at_mode

```
df = df.append(new_df.iloc[5], ignore_index = True)
```

```
[580] df
```

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0	302	41	3	5	50	153	3	4	88	3	41	4	6	4	6	2
1	48332.657807	54.366337	NaN	0.970199	131.615894	246.59	0.152027	0.529801	149.130597	0.324503	1.043046	1.398671	0.741611	2.313531	NaN	0.544554
2	48340.0	55.0	NaN	1.0	130.0	241.5	0.0	1.0	152.0	0.0	0.8	1.0	0.0	2.0	NaN	1.0
3	1	8	0	50	36	0	251	152	7	204	13	139	170	166	0	165
4	46820.0	58.0	M	0.0	120.0	197.0	0.0	1.0	162.0	0.0	0.0	2.0	0.0	2.0	O	1.0
5	1	19	206	142	37	6	251	152	9	204	98	141	170	166	115	165
6	877.940533	9.082101	NaN	1.032257	17.566716	51.969652	0.359655	0.525849	22.595484	0.468966	1.161452	0.616872	1.026753	0.612277	NaN	0.498835

Figure 18: std_dev is copied from new_df(descriptive statistics dataframe)

```
df = df.append(new_df.iloc[6], ignore_index = True)
```

```
[582] df = df.append(new_df.iloc[10], ignore_index = True)
```

```
[583] df
```

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0	302	41	3	5	50	153	3	4	88	3	41	4	6	4	6	2
1	48332.657807	54.366337	NaN	0.970199	131.615894	246.59	0.152027	0.529801	149.130597	0.324503	1.043046	1.398671	0.741611	2.313531	NaN	0.544554
2	48340.0	55.0	NaN	1.0	130.0	241.5	0.0	1.0	152.0	0.0	0.8	1.0	0.0	2.0	NaN	1.0
3	1	8	0	50	36	0	251	152	7	204	13	139	170	166	0	165
4	46820.0	58.0	M	0.0	120.0	197.0	0.0	1.0	162.0	0.0	0.0	2.0	0.0	2.0	O	1.0
5	1	19	206	142	37	6	251	152	9	204	98	141	170	166	115	165
6	877.940533	9.082101	NaN	1.032257	17.566716	51.969652	0.359655	0.525849	22.595484	0.468966	1.161452	0.616872	1.026753	0.612277	NaN	0.498835
7	46820.0	29.0	NaN	0.0	94.0	126.0	0.0	0.0	71.0	0.0	0.0	0.0	0.0	0.0	NaN	0.0
8	49840.0	77.0	NaN	3.0	200.0	564.0	1.0	2.0	202.0	1.0	6.2	2.0	4.0	3.0	NaN	1.0

Figure 19: min and max are copied from new_df(row 7 is min, row 8 is max)

```
[584] data_to_append = {}
      for i in range(len(dataframe.columns)):
        count = (dataframe.iloc[:,i] == 0).sum()
        data_to_append[dataframe.columns[i]]=count
      df = df.append(data_to_append, ignore_index = True)
```

```
[585] df
```

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0	302	41	3	5	50	153	3	4	88	3	41	4	6	4	6	2
1	48332.657807	54.366337	NaN	0.970199	131.615894	246.59	0.152027	0.529801	149.130597	0.324503	1.043046	1.398671	0.741611	2.313531	NaN	0.544554
2	48340.0	55.0	NaN	1.0	130.0	241.5	0.0	1.0	152.0	0.0	0.8	1.0	0.0	2.0	NaN	1.0
3	1	8	0	50	36	0	251	152	7	204	13	139	170	166	0	165
4	46820.0	58.0	M	0.0	120.0	197.0	0.0	1.0	162.0	0.0	0.0	2.0	0.0	2.0	O	1.0
5	1	19	206	142	37	6	251	152	9	204	98	141	170	166	115	165
6	877.940533	9.082101	NaN	1.032257	17.566716	51.969652	0.359655	0.525849	22.595484	0.468966	1.161452	0.616872	1.026753	0.612277	NaN	0.498835
7	46820.0	29.0	NaN	0.0	94.0	126.0	0.0	0.0	71.0	0.0	0.0	0.0	0.0	0.0	NaN	0.0
8	49840.0	77.0	NaN	3.0	200.0	564.0	1.0	2.0	202.0	1.0	6.2	2.0	4.0	3.0	NaN	1.0
9	0	0	0	142	0	0	251	146	0	204	98	21	170	2	0	138

Figure 20: Number of zeros is count

```
[586] data_to_append = {}
      for i in range(len(dataframe.columns)):
        count = (dataframe.iloc[:,i].isnull()).sum()
        data_to_append[dataframe.columns[i]]=count
      df = df.append(data_to_append, ignore_index = True)
```

Figure 21: Number of null entries is count

```
df
```

	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0	302	41	3	5	50	153	3	4	88	3	41	4	6	4	6	2
1	48332.657807	54.366337	NaN	0.970199	131.615894	246.59	0.152027	0.529801	149.130597	0.324503	1.043046	1.398671	0.741611	2.313531	NaN	0.544554
2	48340.0	55.0	NaN	1.0	130.0	241.5	0.0	1.0	152.0	0.0	0.8	1.0	0.0	2.0	NaN	1.0
3	1	8	0	50	36	0	251	152	7	204	13	139	170	166	0	165
4	46820.0	58.0	M	0.0	120.0	197.0	0.0	1.0	162.0	0.0	0.0	2.0	0.0	2.0	O	1.0
5	1	19	206	142	37	6	251	152	9	204	98	141	170	166	115	165
6	877.940533	9.082101	NaN	1.032257	17.566716	51.969652	0.359655	0.525849	22.595484	0.468966	1.161452	0.616872	1.026753	0.612277	NaN	0.498835
7	46820.0	29.0	NaN	0.0	94.0	126.0	0.0	0.0	71.0	0.0	0.0	0.0	0.0	0.0	NaN	0.0
8	49840.0	77.0	NaN	3.0	200.0	564.0	1.0	2.0	202.0	1.0	6.2	2.0	4.0	3.0	NaN	1.0
9	0	0	0	142	0	0	251	146	0	204	98	21	170	2	0	138
10	2	0	1	1	1	3	7	1	35	1	1	2	5	0	5	0

Figure 22: df with all rows are added

```
[588] stat = ['cardinality','mean','median','n_at_median','mode','n_at_mode','stddev','min','max','nzero','nmissing']
      df.insert(0, "stat", stat)
```

Figure 23: stat column is added to df

df

	stat	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0	cardinality	302	41	3	5	50	153	3	4	88	3	41	4	6	4	6	2
1	mean	48332.657807	54.366337	NaN	0.970199	131.615894	246.59	0.152027	0.529801	149.130597	0.324503	1.043046	1.398671	0.741611	2.313531	NaN	0.544554
2	median	48340.0	55.0	NaN	1.0	130.0	241.5	0.0	1.0	152.0	0.0	0.8	1.0	0.0	2.0	NaN	1.0
3	n_at_median	1	8	0	50	36	0	251	152	7	204	13	139	170	166	0	165
4	mode	46820.0	58.0	M	0.0	120.0	197.0	0.0	1.0	162.0	0.0	0.0	2.0	0.0	2.0	O	1.0
5	n_at_mode	1	19	206	142	37	6	251	152	9	204	98	141	170	166	115	165
6	stddev	877.940533	9.082101	NaN	1.032257	17.566716	51.969652	0.359655	0.525849	22.595484	0.468966	1.161452	0.616872	1.026753	0.612277	NaN	0.498835
7	min	46820.0	29.0	NaN	0.0	94.0	126.0	0.0	0.0	71.0	0.0	0.0	0.0	0.0	0.0	NaN	0.0
8	max	49840.0	77.0	NaN	3.0	200.0	564.0	1.0	2.0	202.0	1.0	6.2	2.0	4.0	3.0	NaN	1.0
9	nzero	0	0	0	142	0	0	251	146	0	204	98	21	170	2	0	138
10	nmissing	2	0	1	1	1	3	7	1	35	1	1	2	5	0	5	0

Figure 24: Final output data frame

.Question 2.

```
df.to_excel("output.xlsx")
```

Figure 25: Dataframe is pushed to excel file

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
		stat	member	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	bt	target
0		cardinality	302	41	3	5	50	153	3	4	88	3	41	4	6	4	6	2
1		mean	48332.66	54.36634		0.970199	131.6159	246.59	0.152027	0.529801	149.1306	0.324503	1.043046	1.398671	0.741611	2.313531		0.544554
2		median	48340	55		1	130	241.5	0	1	152	0	0.8	1	0	2		1
3		n_at_medi	1	8	0	50	36	0	251	152	7	204	13	139	170	166	0	165
4		mode	46820	58	M	0	120	197	0	1	162	0	0	2	0	2	O	1
5		n_at_mode	1	19	206	142	37	6	251	152	9	204	98	141	170	166	115	165
6		stddev	877.9405	9.082101		1.032257	17.56672	51.96965	0.359655	0.525849	22.59548	0.468966	1.161452	0.616872	1.026753	0.612277		0.498835
7		min	46820	29		0	94	126	0	0	71	0	0	0	0	0		0
8		max	49840	77		3	200	564	1	2	202	1	6.2	2	4	3		1
9		nzero	0	0	0	142	0	0	251	146	0	204	98	21	170	2	0	138
10		nmissing	2	0	1	1	1	3	7	1	35	1	1	2	5	0	5	0

Figure 25: Excel file

Question 3.

a.&b.

Member: Numeric and ID. Because it is the identifier of each data instance. 2 values are missing,

Age: Feature. It is numeric. No missing values.

Sex: Feature. It is binary since it can be either male or female. 1 missing value.

Cp: Feature. It is categorical.1 missing value.

Trestbps: Feature. Numeric. Continuous. 1 missing value.

Chol: Numeric. Feature. Continuous. 3 missing values.

Fbs: Feature. Binary. Because it can take 1 or 0 only. 7 missing values.

Restecg: Feature. Ordinal. 0 missing values.

Thalach: Feature. Numerical. 35 missing values.

Exang: Feature. Binary. 1 missing.

Oldpeak: Feature. Numeric. Continuous. 1 missing value.
Slope: Feature. Categorical. 2 missing value.
Ca: Feature. Numerical. 5 missing and 5 invalid values (invalid value is 4 because the range is supposed to be 0-3)
Thal: Feature. Categorical. Only 118 values are valid according to the data sheet because it says the values in the column must be 3,6 and 7 but there are only 0,1,2 and 3 values.
Bt: Categorical. Feature. 5 missing values.
Target: Target. Binary. No missing values.

c.
Members: Can be discarded since it is not a feature but id.
Sex: Replace with most frequently seen value since it is categorical.
Cp: Delete row or use smote. No replacing because this value is crucial for the target and it is categorical.
Trestbps: Replace with mean.
Thalach: Replace with mean.
Exang: Smote or replace with most frequent value
Oldpeak: Replace with mean.
Slope: Smote or delete row.
Ca: Replace with mode.
Thal: Drop feature.
Bt: Delete row, since its value cannot be predicted. Or smote.
I think the 'thal' column should be removed completely since it has too many invalid values.

Question 4.
Covariance matrix:

	member	age	trestbps	chol	thalach	oldpeak	ca	target
member	768218.8							
age	1480.577	82.21233						
trestbps	1697.479	44.51458	307.5677					
chol	846.1829	102.62	110.5826	2691.842				
thalach	-8146.83	-73.0569	-15.3469	16.40282	508.6509			
oldpeak	307.2909	2.237494	3.943852	3.533738	-8.95472	1.344505		
ca	344.0951	2.557886	1.816236	3.522033	-4.19026	0.261877	1.042273	
target	-377.021	-1.01797	-1.27155	-2.0686	4.649978	-0.25133	-0.19916	0.248015

Figure 26: Covariance matrix

Correlation:

	member	age	trestbps	chol	thalach	oldpeak	ca	target
member	1							
age	0.185808	1						
trestbps	0.110312	0.279508	1					
chol	0.018508	0.217346	0.121418	1				
thalach	-0.40954	-0.36278	-0.03798	0.013848	1			
oldpeak	0.300974	0.212657	0.193315	0.058719	-0.34325	1		
ca	0.383924	0.276326	0.101285	0.066333	-0.17823	0.221042	1	
target	-0.86329	-0.22544	-0.14555	-0.08	0.413508	-0.43539	-0.39172	1

Figure 27: Correlation matrix

Question 5.

3 predictions most highly correlated with target are; member (but it's not a predictor so it will be ignored). Oldpeak, thalach, ca are the most correlated ones with target with values -0.43539, 0.413508 and -0.39172 respectively.

3 most highly correlated predictors are; (member is ignored again), age and thalac with -0.36278, oldpeak and thalac with -0.34325 and, age and trestbps with 0.279508.