Name: SUN RUI
Student ID: 18083229g

Answer 1:
a.

I(Lifestyle, Street, Polarized) = $\frac{6}{15} * \log_2 \frac{6}{15} + \frac{4}{15} * \log_2 \frac{4}{15} + \frac{5}{15} * \log_2 \frac{5}{15} \approx 1.5656$

Entropy for TPR (Tear Production Rate):

TPR	Lifestyle	Street	Polarized	I(c1, c2, c3)
Reduced	2	3	2	1.5567
Normal	4	1	3	1.4056

E(TPR) = $\frac{7}{15} * 1.5567 + \frac{8}{15} * 1.4056 \approx 1.4761$

Information_Gain (TPR) = 1.5656 – 1.4761 = 0.0895

Entropy for Sex:

Sex	Lifestyle	Street	Polarized	I(c1, c2, c3)
M	4	1	3	1.4056
F	2	3	2	1.5567

E(Sex) = $\frac{8}{15} * 1.4056 + \frac{7}{15} * 1.5567 \approx 1.4761$

Information_Gain (Sex) = 1.5656 – 1.4761 = 0.0895

Entropy for Age:

TPR	Lifestyle	Street	Polarized	I(c1, c2, c3)
Young	3	0	2	0.971
Old	2	2	2	1.585
Middle	1	2	1	1.5

E(Age) = $\frac{5}{15} * 0.971 + \frac{6}{15} * 1.585 + \frac{4}{15} * 1.5 \approx 1.3577$

Information_Gain (Age) = 1.5656 – 1.3577 = 0.2079

Entropy for SP (Spectacle Prescription):

SP	Lifestyle	Street	Polarized	I(c1, c2, c3)
Myope	2	4	1	1.3788
Hypermetrope	4	0	4	1

E(SP) = $\frac{7}{15} * 1.3788 + \frac{8}{15} * 1 \approx 1.1768$

Information_Gain (SP) = 1.5656 – 1.1768 = 0.3888

Entropy for Astigmatism:

Astigmatism	Lifestyle	Street	Polarized	I(c1, c2, c3)
Yes	5	0	3	0.9544
No	1	4	2	1.3788

E(Astigmatism) = $\frac{8}{15} * 0.9544 + \frac{7}{15} * 1.3788 \approx 1.1525$

Information_Gain (Astigmatism) = 1.5656 – 1.1525 = 0.4131

According above tables, we can find that the Information Gain of Astigmatism is larger than others, so we can set the **Astigmatism as root node**.

Next, we continue choosing its child nodes when Astigmatism is Yes and No respectively:

Yes: I(Lifestyle, Street, Polarized) = $\frac{5}{8} * \log_2\frac{5}{8} + \frac{0}{8} * \log_2\frac{0}{8} + \frac{3}{8} * \log_2\frac{3}{8} \approx 0.9544$

Entropy for TPR (Tear Production Rate):

TPR	Lifestyle	Street	Polarized	I(c1, c2, c3)
Reduced	2	0	2	1
Normal	3	0	1	0.8113

E(TPR) = $\frac{4}{8} * 1 + \frac{4}{8} * 0.8113 \approx 0.9057$

Information_Gain (TPR) = 0.9544 – 0.9057 = 0.0487

Entropy for Sex:

Sex	Lifestyle	Street	Polarized	I(c1, c2, c3)
M	3	0	1	0.8113
F	2	0	2	1

E(Sex) = $\frac{4}{8} * 0.8113 + \frac{4}{8} * 1 \approx 0.9057$

Information_Gain (Sex) = 0.9544 – 0.9057 = 0.0487

Entropy for Age:

TPR	Lifestyle	Street	Polarized	I(c1, c2, c3)
Young	2	0	0	0
Old	2	0	2	1
Middle	1	0	1	1

E(Age) = $\frac{2}{8} * 0 + \frac{4}{8} * 1 + \frac{2}{8} * 1 \approx 0.75$

Information_Gain (Age) = 0.9544 – 0.75 = 0.2044

Entropy for SP (Spectacle Prescription):

SP	Lifestyle	Street	Polarized	I(c1, c2, c3)
Myope	1	0	1	1
Hypermetrope	4	0	2	0.9183

E(SP) = $\frac{2}{8} * 1 + \frac{6}{8} * 0.9183 \approx 0.9387$

Information_Gain (SP) = 0. 9544 – 0.9387 = 0.0157

According above three tables, we can find the Information Gain of Age is the largest one, so we choose **Age as one of child nodes** of Astigmatism when Astigmatism is Yes.

No: I(Lifestyle, Street, Polarized) = $\frac{1}{7} * \log_2\frac{1}{7} + \frac{4}{7} * \log_2\frac{4}{7} + \frac{2}{7} * \log_2\frac{2}{7} \approx 1.3788$

Entropy for TPR (Tear Production Rate):

TPR	Lifestyle	Street	Polarized	I(c1, c2, c3)
Reduced	0	3	0	0
Normal	1	1	2	1.5

E(TPR) = $\frac{3}{7} * 0 + \frac{4}{7} * 1.5 \approx 0.8571$

Information_Gain (TPR) = 1.3788 – 0.8571 = 0.5217

Entropy for Sex:

Sex	Lifestyle	Street	Polarized	I(c1, c2, c3)
M	1	1	2	1.5
F	0	3	0	0

E(Sex) = $\frac{4}{7} * 1.5 + \frac{3}{7} * 0 \approx 0.8571$

Information_Gain (Sex) = 1.3788 – 0.8571 = 0.5217

Entropy for Age:

Age	Lifestyle	Street	Polarized	I(c1, c2, c3)
Young	1	0	2	0.9183
Old	0	2	0	0
Middle	0	2	0	0

E(Age) = $\frac{3}{7} * 0.9183 + \frac{2}{7} * 0 + \frac{2}{7} * 1 \approx 0.3936$

Information_Gain (Age) = 1.3788 – 0.3936 = 0.9852

Entropy for SP (Spectacle Prescription):

SP	Lifestyle	Street	Polarized	I(c1, c2, c3)
Myope	1	4	0	0.7219
Hypermetrope	0	0	2	0

E(SP) = $\frac{5}{7} * 0.7219 + \frac{2}{7} * 0 \approx 0.5156$

Information_Gain (SP) = 1.3788 – 0.5156 = 0.8632

According above three tables, we can find the Information Gain of Age is the largest one, so we choose **Age as another child nodes** of Astigmatism when Astigmatism is No.

Now, analyze present situation, we can conclude that:

IF Astigmatism = Yes AND Age = Young THEN Recommendation = Lifestyle

IF Astigmatism = No AND Age = Middle THEN Recommendation = Street

IF Astigmatism = No AND Age = Old THEN Recommendation = Street

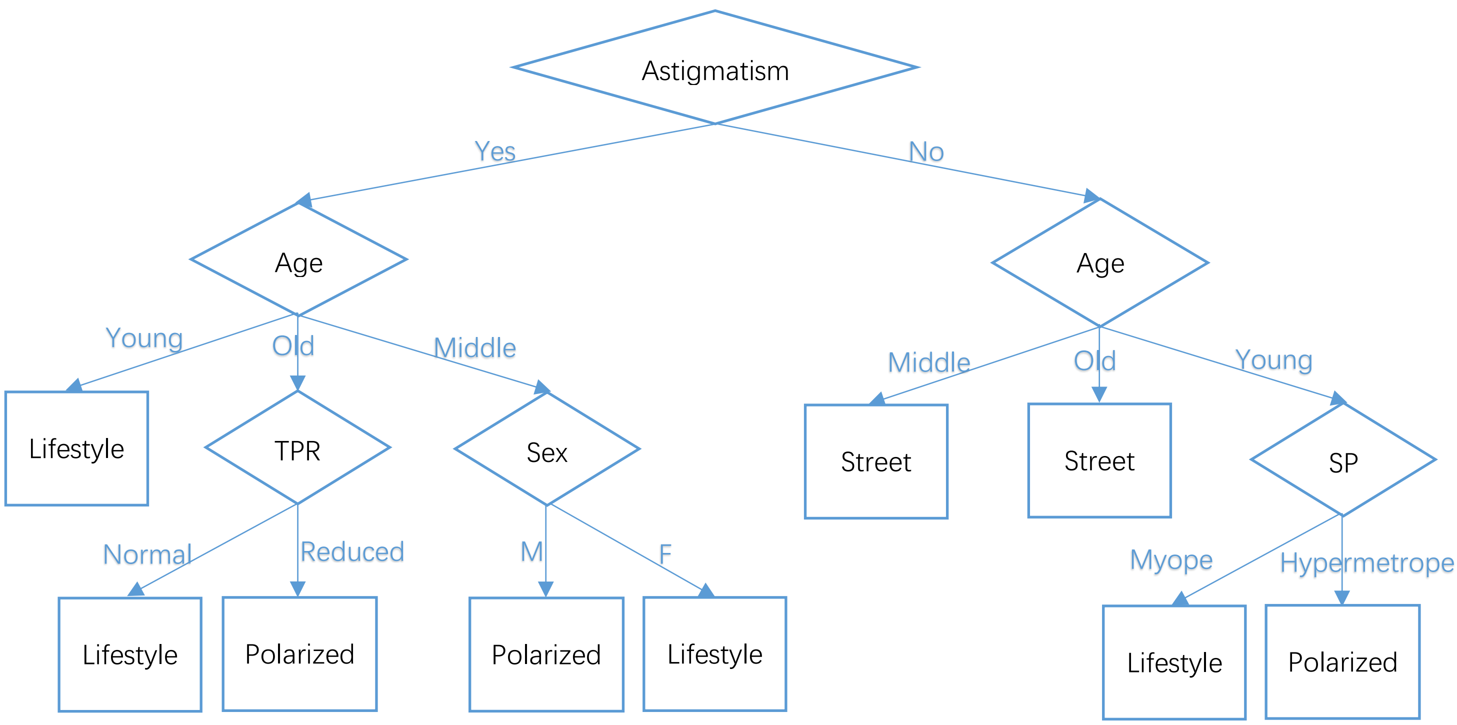
Because of this, we can ensure three leaf nodes. In additional, when Astigmatism is No, the Age only have one attribute Young which is not ensured, however the attribute SP can help us to

divide Recommendations in this branch if Age is Young, so we can draw a conclusion:
IF Astigmatism = No AND Age = Young AND SPC = Myope THEN Recommendation = Lifestyle
IF Astigmatism = No AND Age = Young AND SPC = Hypermetrope THEN Recommendation = Polarized

Until now, there is no more child branches when Astigmatism is No, we start consider that Astigmatism is Yes:
Firstly, when Astigmatism is Yes and Age is Old, we can find attribute TPR (Tear Production Rate) can divide this child branch perfectly:
IF Astigmatism = Yes AND Age = Old AND TPR = Normal THEN Recommendation = Lifestyle
IF Astigmatism = Yes AND Age = Old AND TPR = Reduced THEN Recommendation = Polarized

Secondly, when Astigmatism is Yes and Age is Middle, we can find attribute Sex can divide this child branch perfectly:
IF Astigmatism = Yes AND Age = Middle AND SEX = F THEN Recommendation = Lifestyle
IF Astigmatism = Yes AND Age = Middle AND SEX = M THEN Recommendation = Polarized

Finally, the decision tree:



Note: In order to use these data in program, we need to transform these data from strings to integers:

Tear Production Rate	Sex	Age	Spectacle Prescription	Astigmatism	Recommendation
Reduced (4)	M (2)	Young (6)	Myope (9)	Yes (1)	Lifestyle (11)
Normal (5)	F (3)	Old (7)	Hypermetrope (10)	No (0)	Street (12)
		Middle (8)			Polarized (13)

b. The testing accuracy rate **80%**, the data in last row of Testing Data Set is not matching in my decision tree. The result from my decision tree is Street(12), actually the true result is Lifestyle(11).

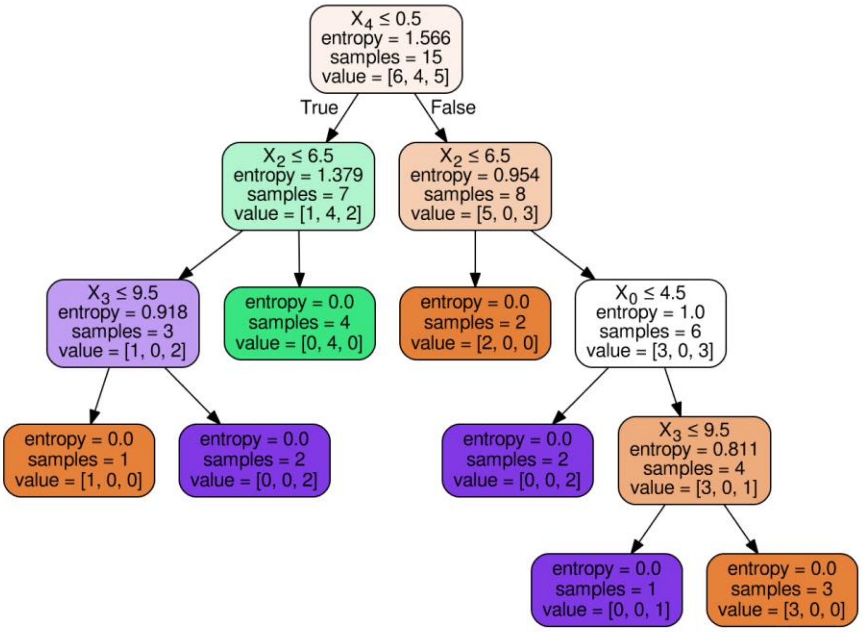
c.

The result by Scikit-Learn:

```
(.env_h)
rico_@DESKTOP-I1BPPTO MINGW64 /d/Msc
$ py decision_tree_ques1.py
testing Y data: [11 12 13 13 11]
prediction Y data: [11 12 13 13 12]
accuracy rate: 80.0%
```

We can find that the accuracy rate is 80% by Scikit-Learn, the fault result is also Street(12), actually the result should be Lifestyle, which is same to my conclusion. In conclusion, Scikit-Learn has the similar prediction with my decision.

The decision tree by scikit-learn:



But, we can find the structure of this tree is a little different from my tree, the first layer and second layer are same in both this tree and my tree, and the count of samples are same, too. But other layers are different.

The reference code:

```
# -*- encoding:utf-8 -*-

import pandas as pd
from sklearn import tree
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
```

```python
df_train = pd.read_csv("training_data_set.csv")
df_train.columns = ["Tear Production Rate", "Sex", "Age", "Spectacle Prescription",
"Astigmatism", "Recommendation"]
train_X = df_train.values[:, 0:5]
train_Y = df_train.values[:, 5]

df_test = pd.read_csv("test_data_set.csv")
df_test.columns = ["Tear Production Rate", "Sex", "Age", "Spectacle Prescription",
"Astigmatism", "Recommendation"]
test_X = df_test.values[:, 0:5]
test_Y = df_test.values[:, 5]

dtree = tree.DecisionTreeClassifier(criterion="entropy")
dtree.fit(train_X, train_Y)

Y_pred = dtree.predict(test_X)
print("testing Y data: {0}".format(test_Y))
print("prediction Y data: {0}".format(Y_pred))
print("accuracy rate: {0}%".format(dtree.score(test_X, test_Y)*100))

dot_data = StringIO()
export_graphviz(dtree, out_file=dot_data, filled=True, rounded=True, special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_pdf("decision_tree.pdf")
```

Answer 2:

a).

step1, normalize data ((x-min)/(max-min)):

Customer No.	Sex	Average No. of Transactions	Average Monthly Payment	Average No. of months in Silver
1	1	0.2632	0.0718	0.0
2	0	0.7895	0.4628	0.3636
3	1	0.1053	0.0824	0.4545
4	0	0.0	0.0931	0.1818
5	1	0.4211	0.6596	0.5455
6	0	0.0	1.0	0.8182
7	1	0.3158	0.5745	0.0909
8	0	0.3684	0.0691	0.2727
9	1	0.2105	0.0	0.7273
10	1	0.8947	0.8005	1.0
11	0	1.0	0.6809	0.4545
12	1	0.5789	0.2367	0.1818
13	0	0.3684	0.359	0.3636
14	0	0.6316	0.5452	0.2727
15	0	0.5263	0.4548	0.6364

step2, normalize sample data:

Sex	Average No. of Transactions	Average Monthly Payment	Average No. of months in Silver
0	0.3158	0.3617	0.0909

step3, calculate euclidean distances among the sample and all 15 training data:

Customer No.	distance
1	1.0465
2	0.5559
3	1.12
4	0.4244
5	1.143
6	1.0179
7	1.0224

Customer No.	distance
8	0.3486
9	1.2437
10	1.5343
11	0.838
12	1.0455
13	0.2778
14	0.408
15	0.5921

step4, select smallest 5 distances and their Customer No., further more to find their Decisions:

distance	0.2778	0.3486	0.408	0.4244	0.5559
Customer No.	13	8	14	4	2
Decision	Remain	Upgrade	Remain	Downgrade	Downgrade

step5, select Decision with largest count number:
According above table in step4, we find that the count of both Remain and Downgrade is 2, so we can get these two results. If we must get one result, I think that we can choose Remain as final result, because it has smaller distances than Downgrade.

b)
The method is similar to question a), the difference is that the target is to find Average No. of Transactions rather than Decision, the detail steps:
step1, normalize data ((x-min)/(max-min)):

Customer No.	Sex	Average Monthly Payment	Average No. of months in Silver
1	1	0.0718	0.0
2	0	0.4628	0.3636
3	1	0.0824	0.4545
4	0	0.0931	0.1818
5	1	0.6596	0.5455
6	0	1.0	0.8182
7	1	0.5745	0.0909
8	0	0.0691	0.2727
9	1	0.0	0.7273
10	1	0.8005	1.0
11	0	0.6809	0.4545

12	1	0.2367	0.1818
13	0	0.359	0.3636
14	0	0.5452	0.2727
15	0	0.4548	0.6364

step2, normalize sample data:

Sex	Average Monthly Payment	Average No. of months in Silver
1	0.2821	0.4655

step3, calculate euclidean distances among the sample and all 15 training data:

Customer No.	distance
1	0.5108
2	1.0213
3	0.2
4	1.0565
5	0.3859
6	1.2805
7	0.4752

Customer No.	distance
8	1.0405
9	0.3848
10	0.7446
11	1.0766
12	0.2874
13	1.0081
14	1.0519
15	1.0291

step4, select smallest 5 distances and their Customer No., further more to find their Average No. of Transactions and to calculate weights:

distance	0.2	0.2874	0.3848	0.3859	0.4752
Customer No.	3	12	9	5	7
Average No. of Transactions	5	14	7	11	9
Weights	0.317	0.2206	0.1647	0.1643	0.1334

Note: method of calculating weights:

(distance/sum(distances))^-1 / sum((distance/sum(distances))^-1)

step5, calculate the final Average No. of Transactions through multiply the 5 Average No. of Transactions with their weights respectively in step4, then plus these five results together:

5*0.317 + 14 * 0.2206 + 7*0.1647 + 11*0.1643 + 9*0.1334 = 8.8342

In conclusion, we expect the average no. of transactions of the customer is **8.8342**

c)

In part a), I will choose k=3. If k = 3, we can get only decision of Remain. And the RMSE is 0.089 if k=3, which is less than 0.206 if k=5. As a conclusion, k=3 is better than k=5.

Answer 3:

a)

Without pre-processing data, the result:

```
(.env_h)
rico_@DESKTOP-I1BPPT0 MINGW64 /d/Msc_learn/homework_and_project/AI/assignments (master)
$ py decision_tree_ques2.py
mean accuracy: 0.8719374999999998
RMSE error:0.35785821214553676
```

I make the training step loop for **100000** times, which gets mean accuracy about **87.19%** and RMSE error about **0.3579**. Reference code:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn import metrics

file_name = "cardiac.csv"
df_train = pd.read_csv(file_name)
df_train.columns = ["bhr", "basebp", "basedp", "pkhr", "sbp", "dp", "dose",
"maxhr", "%mphr(b)", "mbp", "dpmaxdo",
                    "dobdose", "age", "gender", "baseEF", "dobEF", "chestpain",
"posECG", "equivecg", "restwma",
                    "posSE", "newMI", "newPTCA", "newCABG", "hxofHT", "hxofdm",
"hxofcig", "hxofMI",
                    "hxofPTCA", "hxofCABG", "death"]
X = df_train.values[:, 0:-1]
Y = df_train.values[:, -1]

accuracy_ls = []
test_y_ls = []
pre_y_ls = []

times = 100000
for i in range(times):
    print("finish {0}%\r".format(int(i / times * 100)), end='')
    trainX, testX, trainY, testY = train_test_split(X, Y, test_size=0.3)
    test_y_ls.append(testY)
    dtree = tree.DecisionTreeClassifier(criterion="entropy")
    dtree.fit(trainX, trainY)
    Y_pred = dtree.predict(testX)
    pre_y_ls.append(Y_pred)
    accuracy_ls.append(dtree.score(testX, testY))

print("mean accuracy: {0}".format(np.mean(accuracy_ls)))
print("RMSE error:{0}".format(np.sqrt(metrics.mean_squared_error(test_y_ls,
pre_y_ls))))
```

b)

After observing data set, I found some data is much dispersive in a large range, so I decide to divide them into some intervals, the way:

max<=50	Interval is 5: x0~x2 → x0, x3~x7 → x5, x8~y0 → y0 (y=x+1)
50<max<=100	Interval is 10: x0~x4 → x0, x5~x9 → y0 (y=x+1)
100<max	only left first two higher bits, other lower bits are set 0, if <100, refer to "50<max<=100"

I will put the code in the last page, (ref1).

Through the dispose in according above methods, we can get the result by the same way to part a):

```
(.env_h)
rico_@DESKTOP-I1BPPT0 MINGW64 /d/Msc_learn/homework_and_project/AI/assignments (master)
$ py decision_tree_ques2.py
mean accuracy: 0.8776916666666669
RMSE error:0.34972608328995614
```

We can get mean accuracy about **87.77%** and RMSE error about **0.3497**. We can find that the mean accuracy is higher than part a) about **0.58%**, and RMSE error is lower than a) about **0.0082**, which are better than part a).

Then I found the data of several attributes are too much single (by a python script to count data, I will put the code in the last page, ref2), so after delete these attributes, we see the performance:

hxofPTCA		hxofcig		newCABG		newPTCA		posECG	
data	count	data	count	data	count	data	count	data	count
0	74	0.0	66	0	71	0	73	0	72
1	3	1.5	6	1	6	1	4	1	5
		1.0	5						

```
(.env_h)
rico_@DESKTOP-I1BPPT0 MINGW64 /d/Msc_learn/homework_and_project/AI/assignments (master)
$ py decision_tree_ques2_d.py
mean accuracy: 0.8815299999999998
RMSE error:0.34419471233591026
```

We can get mean accuracy about **88.15%** and RMSE error about **0.3442**. We can find that the mean accuracy is higher than part a) about **0.96%**, and RMSE error is lower than a) about **0.0137**, which are better than part a).

Draw a conclusion, through above two steps, we can improve accuracy of the classification process to some extent, but the performance is limited.

ref1:

```python
# -*- encoding:utf-8 -*-

import pandas as pd
import numpy as np

df_data = pd.read_csv("cardiac.csv")
df_data.columns = ["bhr", "basebp", "basedp", "pkhr", "sbp", "dp", "dose",
"maxhr", "%mphr(b)", "mbp", "dpmaxdo",
                  "dobdose", "age", "gender", "baseEF", "dobEF", "chestpain",
"posECG", "equivecg", "restwma",
                  "posSE", "newMI", "newPTCA", "newCABG", "hxofHT", "hxofdm",
"hxofcig", "hxofMI",
                  "hxofPTCA", "hxofCABG", "death"]


dic_clean_data = {}
# bhr_series = df_data["bhr"]
dic_clean_data["bhr"] = df_data["bhr"]
# basebp_series = df_data["basebp"]
dic_clean_data["basebp"] = df_data["basebp"]
# basedp_series = df_data["basedp"]
dic_clean_data["basedp"] = df_data["basedp"]
# pkhr_series = df_data["pkhr"]
dic_clean_data["pkhr"] = df_data["pkhr"]
# sbp_series = df_data["sbp"]
dic_clean_data["sbp"] = df_data["sbp"]
# dp_series = df_data["dp"]
dic_clean_data["dp"] = df_data["dp"]
# maxhr_series = df_data["maxhr"]
dic_clean_data["maxhr"] = df_data["maxhr"]
# mphr_b_series = df_data["%mphr(b)"]
dic_clean_data["%mphr(b)"] = df_data["%mphr(b)"]
# mbp_series = df_data["mbp"]
dic_clean_data["mbp"] = df_data["mbp"]
# dpmaxdo_series = df_data["dpmaxdo"]
dic_clean_data["dpmaxdo"] = df_data["dpmaxdo"]
# age_series = df_data["age"]
dic_clean_data["age"] = df_data["age"]
# baseEF_series = df_data["baseEF"]
dic_clean_data["baseEF"] = df_data["baseEF"]
# dobEF_series = df_data["dobEF"]
dic_clean_data["dobEF"] = df_data["dobEF"]
for (k, v) in dic_clean_data.items():
    # max_item = max(v)
    # min_item = min(v)
    temp_ls = []
    for each_data in v:
        if max(v) <= 50:
            if 3 <= each_data % 10 <= 7:
                new_data = each_data//10 * 10 + 5
            elif 7 < each_data % 10 <= 9:
                new_data = each_data//10 * 10 + 10
            elif 1 <= each_data % 10 < 3:
                new_data = each_data // 10 * 10
            else:
                new_data = each_data
            temp_ls.append(new_data)
```

ref1:

```
        elif 50 < max(v) <= 100:
            if 1 <= each_data % 10 <= 4:
                new_data = each_data//10 * 10
            elif 5 <= each_data % 10 <= 9:
                new_data = each_data//10 * 10 + 10
            else:
                new_data = each_data
            temp_ls.append(new_data)
        else:
            if each_data < 100:
                if 1 <= each_data % 10 <= 4:
                    new_data = each_data // 10 * 10
                elif 5 <= each_data % 10 <= 9:
                    new_data = each_data // 10 * 10 + 10
                else:
                    new_data = each_data
                temp_ls.append(new_data)
            else:
                num_len = len(str(each_data))
                new_data = (each_data//np.power(10, num_len-2)) * np.power(10,
num_len-2)
                temp_ls.append(new_data)
    df_data[k] = temp_ls

df_data.to_csv("nor_data1.csv")    # normalization data
```

ref2:

```
# -*- encoding:utf-8 -*-

import pandas as pd

df_data = pd.read_csv("nor_data1.csv")
df_data.columns = ["bhr", "basebp", "basedp", "pkhr", "sbp", "dp", "dose",
"maxhr", "%mphr(b)", "mbp", "dpmaxdo",
                  "dobdose", "age", "gender", "baseEF", "dobEF", "chestpain",
"posECG", "equivecg", "restwma",
                  "posSE", "newMI", "newPTCA", "newCABG", "hxofHT", "hxofdm",
"hxofcig", "hxofMI",
                  "hxofPTCA", "hxofCABG", "death"]
count = 0
for col in df_data.columns:
    count += 1
    print(df_data[col].value_counts())
print(count)
```