**باسمه تعالی**

**تکلیف سری هفتم داده کاوی**

**سارا برادران (شماره دانشجویی: ۹۶۲۴۱۹۳)**

---

سوال ۱ )

a به کمک متد read_csv کتابخانه pandas ابتدا دیتاست bearst_cancer را می خوانیم.

```
features, target = load_breast_cancer(return_X_y=True)
features = pd.DataFrame(features)
```

(b

```
features['Cancer'] = target
```

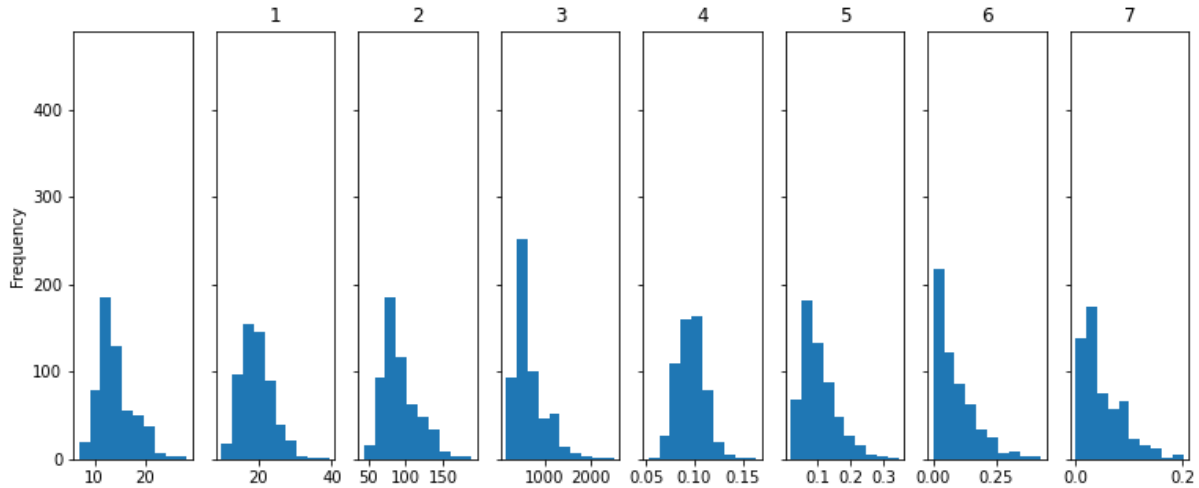
c) همانگونه که مشخص است دیتافریم bearst_cancerحاوی مقدار null نمی باشد.

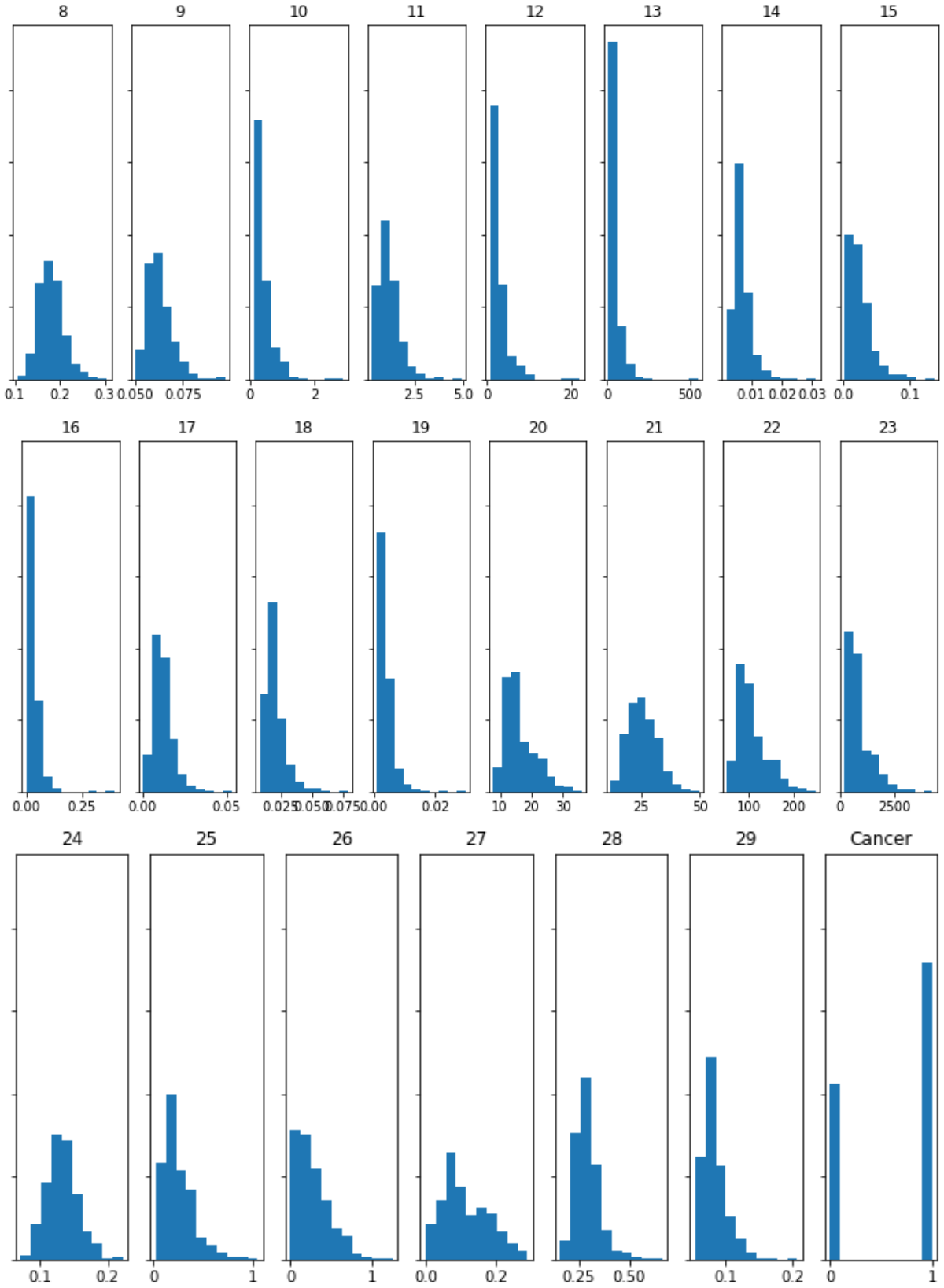
```
# 3_c
print(features.isna().sum())
```

```
0         0
1         0
2         0
3         0
4         0
5         0
6         0
7         0
8         0
9         0
10        0
11        0
12        0
13        0
14        0
15        0
16        0
17        0
18        0
19        0
20        0
21        0
22        0
23        0
24        0
25        0
26        0
27        0
28        0
29        0
Cancer    0
dtype: int64
```



نمودار هیستوگرام نظیر هر یک از ستون های این دیتافریم را رسم کرده و توزیع متغیر ها را بررسی میکنیم :






رنج مقادیر ستون های گوناگون در دیتافریم breast_cancer بسیار متنوع است برای مثال چند مورد از این رنج ها در زیر آمده است. ستون ۰ حاوی مقادیری حدودا در رنج ۶ تا ۲۸ است درحالی که ستون ۲ حاوی مقداری در رنج حدودا ۴۳ تا ۱۸۸ می باشد لذا می بایست به نحوی این رنج مقادیر را یکسان سازی نماییم :

```
column :  0
min =  6.981
max =  28.11
========================================
column :  1
min =  9.71
max =  39.28
========================================
column :  2
min =  43.79
max =  188.5
========================================
```

از متد normalize استفاده کرده و رنج ستون ها را یکسان سازی می نماییم به علاوه این متد توزیع داده هایی که نرمال نیستند را به نرمال نزدیک تر میکند. رنج های تغییر یافته را میتوان در زیر مشاهده کرده :

```
column :  0
min =  0.005511893190855578
max =  0.028469831238522775
========================================
column :  1
min =  0.004568255800758577
max =  0.08660830402261978
========================================
column :  2
min =  0.03639624222914814
max =  0.17858385760419887
========================================
```

d) ابتدا متغیر هدف را از دیگر متغیر های دیتافریم مجزا کرده ایم:

```
x = features_normalize.drop(columns=['Cancer'])
y = features_normalize[['Cancer']]
```

e) به کمک متد train_test_split داده های تست و یادگیری را جدا میکنیم:

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size = 0.2, random_state = 7)
```

f) به کمک متد LogisticRegression کتابخانه sklearn مدل را تشکیل می دهیم و داده های یادگیری را به آن fit می کنیم سپس مقدار Cancer را برای داده های تست پیش بینی می کنیم:

```
model = LogisticRegression()
model.fit(x_train, y_train.values.ravel())
y_predict = model.predict(x_test)
print(accuracy_score(y_test,y_predict))
```

g) میزان دقت بدست آمده برای این مدل 0.7807017543859649 می باشد. یعنی حدودا ۷۸ درصد دقت در تست داشته ایم.

h) دقت مدل های بدست آمده توسط solver های saga و liblinear بر روی داده های تست برابر و هر دو همان دقت بدست آمده در قسمت قبل یعنی 0.7807017543859649 می باشد.

```
# 1_h
model_liblinear = LogisticRegression(solver='liblinear')
model_liblinear.fit(x_train, y_train.values.ravel())
y_predict = model_liblinear.predict(x_test)
print(accuracy_score(y_test,y_predict))
```

```
0.7807017543859649
```

```
model_saga = LogisticRegression(solver='saga')
model_saga.fit(x_train, y_train.values.ravel())
y_predict = model_saga.predict(x_test)
print(accuracy_score(y_test,y_predict))
```

```
0.7807017543859649
```

تفاوت دو Solver :

sag: Stands for Stochastic Average Gradient Descent. More efficient solver with large datasets.

saga: Saga is a variant of Sag and it can be used with l1 Regularization. It's a quite time-efficient solver and usually the go-to solver with very large datasets.

liblinear: More efficient solver with small datasets. Only useful for one-versus-rest problems won't work with multiclass problems unlike other solvers here. Also doesn't work with **l2** or **none** parameter values for penalty.

سوال ۲)

a) به کمک متد read_csv کتابخانه pandas ابتدا دیتاست boston را می خوانیم.

```
features, target = load_boston(return_X_y=True)
```

(b

```
features = pd.DataFrame(features)
```

(c

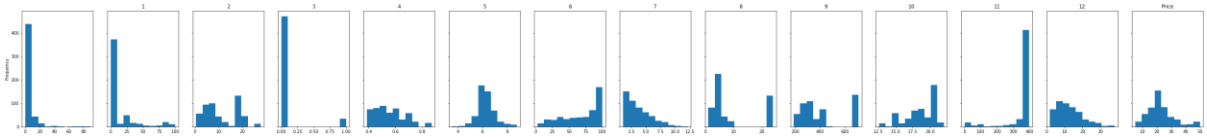
```
features['Price'] = target
```

i) همانگونه که مشخص است دیتافریم boston حاوی مقدار null نمی باشد.

```
# 2_i
print(features.isna().sum())
```

```
0        0
1        0
2        0
3        0
4        0
5        0
6        0
7        0
8        0
9        0
10       0
11       0
12       0
Price    0
dtype: int64
```

نمودار هیستوگرام نظیر هر یک از ستون های این دیتافریم را رسم کرده و توزیع متغیر ها را بررسی میکنیم :

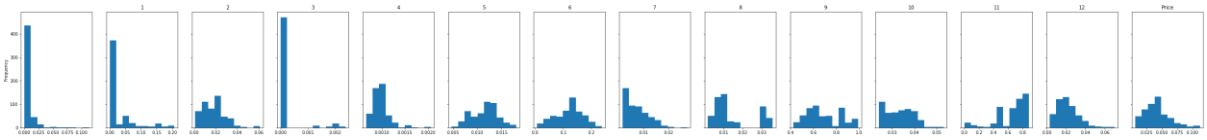


بنابر نتایج زیر رنج مقادیر ستون های گوناگون در دیتافریم boston بسیار متنوع است برای مثال ستون ۱ حاوی مقادیری در رنج ۰ تا ۱۰۰ است درحالی که ستون ۴ حاوی مقداری کمتر از ۱ است. لذا می بایست به نحوی این رنج مقادیر را یکسان سازی نماییم :

```
column :  0
min =  0.00632
max =  88.9762
========================================
column :  1
min =  0.0
max =  100.0
========================================
column :  2
min =  0.46
max =  27.74
========================================
column :  3
min =  0.0
max =  1.0
========================================
column :  4
min =  0.385
max =  0.871
========================================
column :  5
min =  3.561
max =  8.78
```

```
========================================
column :  6
min =  2.9
max =  100.0
========================================
column :  7
min =  1.1296
max =  12.1265
========================================
column :  8
min =  1.0
max =  24.0
========================================
column :  9
min =  187.0
max =  711.0
========================================
column :  10
min =  12.6
max =  22.0
========================================
column :  11
min =  0.32
max =  396.9
========================================
column :  12
min =  1.73
max =  37.97
========================================
column :  Price
min =  5.0
max =  50.0
========================================
```

از متد normalize استفاده کرده و رنج ستون ها را یکسان سازی می نماییم به علاوه این متد توزیع داده هایی که نرمال نیستند را به نرمال نزدیک تر میکند. توزیع های بعضا به نرمال نزدیک شده و رنج های تغییر یافته را میتوان در زیر مشاهده کرده :



```
column :  0
min =  1.2624301979668727e-05
max =  0.11307149231633647
========================================
column :  1
min =  0.0
max =  0.20719815523714544
========================================
column :  2
min =  0.0009577250059326233
max =  0.061206189719183145
========================================
column :  3
min =  0.0
max =  0.0023693882235679368
========================================
```

```
column :  4
min =  0.0006594774970217481
max =  0.002063737142727673
========================================
column :  5
min =  0.0046788495876117315
max =  0.018057878660818944
========================================
column :  6
min =  0.00642031137838965
max =  0.23749580604948395
========================================
column :  7
min =  0.0014505684613511393
max =  0.02755890854941367
========================================
column :  8
min =  0.0017292766942189324
max =  0.03579387594042457
========================================
column :  9
min =  0.4249796642675427
max =  0.9932800573467817
========================================
column :  10
min =  0.02403902992596624
max =  0.05306777085939242
========================================
column :  11
min =  0.0004749098114976438
max =  0.8875236298734167
========================================
column :  12
min =  0.0030871138764995477
max =  0.07002418716732209
========================================
column :  Price
min =  0.006376486195013187
max =  0.11259326053166861
========================================
```

j) ابتدا متغیر هدف را از دیگر متغیر های دیتافریم مجزا کرده و به کمک متد train_test_split داده های تست و یادگیری را جدا میکنیم:

```
x = features_normalize.drop(columns=['Price'])
y = features_normalize[['Price']]
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size = 0.2, random_state = 5)
```

d) به کمک متد PoissonRegressor کتابخانه sklearn مدل را تشکیل می دهیم و داده های یادگیری را به آن fit می کنیم سپس مقدار price را برای داده های تست پیش بینی می کنیم:

```
model = PoissonRegressor()
model.fit(x_train, y_train.values.ravel())
y_predict = model.predict(x_test)
```

e) مقادیر RMSE ، MSE ، MAE به دو صورت هم با استفاده از متد های آماده و هم به صورت دستی بدست آمده است:

PoissonRegressor

```
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error
# 2_e
################### methods for calculation of MAE, MSE, RMSE ###################
print('methods for calculation of MAE, MSE, RMSE')
print('MAE  = ', mean_absolute_error(y_test, y_predict))
print('MSE  = ', mean_squared_error(y_test, y_predict))
print('RMSE = ', np.sqrt(mean_squared_error(y_test, y_predict)))

################### manual calculation of MAE, MSE, RMSE ###################
print('\nmanual calculation of MAE, MSE, RMSE')
bound = y_test.shape[0]
MAE = 0; MSE = 0
for i in range(0, bound):
    MSE += ((y_test.values.ravel()[i] - y_predict[i]) ** 2)
    MAE += (abs(y_test.values.ravel()[i] - y_predict[i]))

print('MAE  = ', MAE/bound)
print('MSE  = ', MSE/bound)
print('RMSE = ', np.sqrt(MSE/bound))
```

```
methods for calculation of MAE, MSE, RMSE
MAE  =  0.015605455209345188
MSE  =  0.00041950229836795505
RMSE =  0.020481755256031037

manual calculation of MAE, MSE, RMSE
MAE  =  0.015605455209345188
MSE  =  0.0004195022983679551
RMSE =  0.02048175525603104
```



LinearRegression

```
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error
# 2_e
################### methods for calculation of MAE, MSE, RMSE ###################
print('methods for calculation of MAE, MSE, RMSE')
print('MAE  = ', mean_absolute_error(y_test, y_predict))
print('MSE  = ', mean_squared_error(y_test, y_predict))
print('RMSE = ', np.sqrt(mean_squared_error(y_test, y_predict)))

################### manual calculation of MAE, MSE, RMSE ###################
print('\nmanual calculation of MAE, MSE, RMSE')
bound = y_test.shape[0]
MAE = 0; MSE = 0
for i in range(0, bound):
    MSE += ((y_test.values.ravel()[i] - y_predict[i]) ** 2)
    MAE += (abs(y_test.values.ravel()[i] - y_predict[i]))

print('MAE  = ', MAE/bound)
print('MSE  = ', MSE/bound)
print('RMSE = ', np.sqrt(MSE/bound))
```

```
methods for calculation of MAE, MSE, RMSE
MAE  =  0.006067158017582162
MSE  =  7.987491470538267e-05
RMSE =  0.008937276694014944

manual calculation of MAE, MSE, RMSE
MAE  =  [0.00606716]
MSE  =  [7.98749147e-05]
RMSE =  [0.00893728]
```

می دانیم که هرچه مقدار MAE ، MSE کمتر باشد در حقیقت دقت مدل بیشتر بوده و مقادیر پیش بینی شده به مقادیر واقعی نزدیک تر هستند.

همانطور که از دو تصویر فوق مشخص است مقدار MAE برای PoissonRegressor مقدار بیشتری (در حدود ۱۵ هزارم) بدست آمده است در حالی که برای LinearRegression این پارامتر در حدود ۶ هزارم ارزیابی شده بود. در مورد MSE و RMSE نیز به همین صورت است برای LinearRegression مقدار MSE در حدود $7.9 * 10^{-5}$ و برای PoissonRegressor این مقدار در حدود $4 * 10^{-4}$ می باشد به این ترتیب به نظر می رسد دقت مدل رگرسیون خطی بهتر بوده است.

سوال ۳)

a) به کمک متد read فایل ابتدا دیتاست Heart را می خوانیم.

```
import pandas as pd

#Sequence = pd.read_csv('Sequence.csv', ',', header=None, names=[ i for i in range(11)])
Sequence = open('Sequence.csv', 'r')
transactions = Sequence.read()
print(transactions)
```

```
Coffee Powder,Sugar,Sweet,Panner,Milk,Bread,
Lassi,Sugar,Yougurt,Tea Powder,Cheese,Milk,
Panner,Coffee Powder,Cheese,Bread,
Panner,Sugar,Butter,Bread,Ghee,Coffee Powder,
Bread,Ghee,Yougurt,Panner,Milk,
Sweet,Bread,Milk,
Coffee Powder,Milk,Butter,Sweet,Cheese,Panner,Bread,
Tea Powder,Cheese,Bread,
Sweet,Cheese,Milk,Tea Powder,
Milk,Cheese,Bread,Lassi,
Panner,Butter,Milk,Yougurt,Cheese,Sugar,Bread,Sweet,
Tea Powder,Ghee,Sweet,Sugar,Butter,Coffee Powder,
Milk,Bread,Butter,Cheese,Sweet,Ghee,
Panner,Sugar,
Cheese,Sugar,Tea Powder,Sweet,
Cheese,Milk,Lassi,Sweet,Panner,Ghee,Yougurt,Bread,
Lassi,Butter,Panner,Bread,Coffee Powder,Ghee,
Lassi,Butter,Cheese,Yougurt,Panner,Bread,Milk,Sugar,
Cheese,Panner,Ghee,Tea Powder,Lassi,
```

b) سپس لیست تراکنش ها را با ساختار مناسب برای ورود به تابع GSP آماده می کنیم و سپس تابع GSP را فراخوانی کرده و متد search را با پارامتر ۰٫۳ بر روی آن اعمال می کنیم:

```
from gsppy.gsp import GSP

item_list = []
transactions_list = []
for line in transactions.split('\n'):
    item_list = []
    for item in line[:-1].split(','):
        item_list.append(item)

    transactions_list.append(item_list)

print(transactions_list)
```

```
[['Coffee Powder', 'Sugar', 'Sweet', 'Panner', 'Milk', 'Bread'], ['Lassi', 'Sugar', 'Yougurt', 'Tea Powder', 'Cheese', 'Mil
k'], ['Panner', 'Coffee Powder', 'Cheese', 'Bread'], ['Panner', 'Sugar', 'Butter', 'Bread', 'Ghee', 'Coffee Powder'], ['Brea
d', 'Ghee', 'Yougurt', 'Panner', 'Milk'], ['Sweet', 'Bread', 'Milk'], ['Coffee Powder', 'Milk', 'Butter', 'Sweet', 'Cheese',
'Panner', 'Bread'], ['Tea Powder', 'Cheese', 'Bread'], ['Sweet', 'Cheese', 'Milk', 'Tea Powder'], ['Milk', 'Cheese', 'Bread',
'Lassi'], ['Panner', 'Butter', 'Milk', 'Yougurt', 'Cheese', 'Sugar', 'Bread', 'Sweet'], ['Tea Powder', 'Ghee', 'Sweet', 'Suga
r', 'Butter', 'Coffee Powder'], ['Milk', 'Bread', 'Butter', 'Cheese', 'Sweet', 'Ghee'], ['Panner', 'Sugar'], ['Cheese', 'Suga
r', 'Tea Powder', 'Sweet'], ['Cheese', 'Milk', 'Lassi', 'Sweet', 'Panner', 'Ghee', 'Yougurt', 'Bread'], ['Lassi', 'Butter',
'Panner', 'Bread', 'Coffee Powder', 'Ghee'], ['Lassi', 'Butter', 'Cheese', 'Yougurt', 'Panner', 'Bread', 'Milk', 'Sugar'],
['Cheese', 'Panner', 'Ghee', 'Tea Powder', 'Lassi'], ['Yougurt', 'Butter', 'Bread', 'Cheese', 'Panner', 'Tea Powder', 'Coffee
Powder', 'Milk', 'Ghee', 'Sugar'], ['Bread', 'Ghee', 'Butter', 'Sugar', 'Coffee Powder', 'Panner'], ['Sugar', 'Cheese', 'Lass
i', 'Bread', 'Tea Powder', 'Yougurt', 'Butter', 'Sweet'], ['Cheese', 'Sweet', 'Tea Powder', 'Bread', 'Yougurt', 'Ghee', 'Butt
```

```
result = GSP(transactions_list).search(0.3)
```

نتایج بدست آمده برای sequence هایی که حداقل در ۳۰ درصد از تراکنش ها تکرار شده اند به صورت زیر می باشد:

```
[{('Sugar',): 1278,
  ('Sweet',): 1260,
  ('Tea Powder',): 1213,
  ('Bread',): 1236,
  ('Cheese',): 1215,
  ('Coffee Powder',): 1264,
  ('Yougurt',): 1244,
  ('Lassi',): 1234,
  ('Butter',): 1282,
  ('Ghee',): 1211,
```

```
('Milk',): 1254,
('Panner',): 1235}]
```

c) الگوی bread,sweet در این sequence پس از کالاهای زیر آمده است که ۵ مورد اول پرتکرار ترین ها هستند.

```
================== prev words ==================
Yougurt           22
Cheese            21
Sugar             20
Panner            19
Coffee Powder     19
Butter            17
Lassi             17
Milk              15
Ghee              11
Tea Powder        10
```

سوال ۴)

a) به کمک متد read_csv کتابخانه pandas ابتدا دیتاست Heart را می خوانیم.

```
Heart = pd.read_csv('Heart.csv', sep=',')
```

تنها به ازای اولین رکورد از دیتاست متغیر maximum heart rate achieved مقدار null اخذ کرده لذا آن را با میانگین دیگر مقادیر همین ستون جایگذاری می کنیم.

```
1 Heart.isna().sum()
```

```
age                         0
gender                      0
chest pain type             0
serum cholestoral           0
fasting blood sugar         0
resting electrocardiographic  0
maximum heart rate achieved   1
exercise induced angina     0
target                      0
dtype: int64
```

```
1 Heart['maximum heart rate achieved'].mean()
```

```
149.6456953642384
```

```
1 Heart[Heart['maximum heart rate achieved'].isna()]
```

| | age | gender | chest pain type | serum cholestoral | fasting blood sugar | resting electrocardiographic | maximum heart rate achieved | exercise induced angina | target |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 63 | 1 | 3 | 233 | 1 | 0 | NaN | 0 | 1 |

```
1 Heart.loc[0, 'maximum heart rate achieved'] = Heart['maximum heart rate achieved'].mean()
```

b) می بایست تغییراتی در دیتافریم ایجاد نماییم تا بتوان از متد apriori بر روی آن استفاده نمود. به این ترتیب ابتدا متغیر های پیوسته موجود را سبد بندی کرده و سپس با استفاده از dummy کردن ستون ها یک دیتا فریم جدید و مناسب برای کاوش قواعد پیشرفته ایجاد می کنیم که به صورت زیر خواهد بود.

```
import jenkspy

breaks = jenkspy.jenks_breaks(Heart['age'], nb_class=3)
print(breaks)

l = ['29_49','49_60','60_77']
Heart['age'] = pd.cut(Heart['age'], bins=breaks, labels=l)
```

```
[29.0, 49.0, 60.0, 77.0]
```

```

breaks = jenkspy.jenks_breaks(Heart['maximum heart rate achieved'], nb_class=3)
print(breaks)

l = ['71_133','133_160','160_202']
Heart['maximum heart rate achieved'] = pd.cut(Heart['maximum heart rate achieved'], bins=breaks, labels=l)
```

```
[71.0, 133.0, 160.0, 202.0]
```

```

breaks = jenkspy.jenks_breaks(Heart['serum cholestoral '], nb_class=3)
print(breaks)

l = ['126_226','226_290','290_564']
Heart['serum cholestoral '] = pd.cut(Heart['serum cholestoral '], bins=breaks, labels=l)
```

```
[126.0, 226.0, 290.0, 564.0]
```

```
Heart =  pd.get_dummies(Heart,columns=Heart.columns)
```

```
Heart.head()
```

| | age_29_49 | age_49_60 | age_60_77 | gender_0 | gender_1 | chest pain type_0 | chest pain type_1 | chest pain type_2 | chest pain type_3 | serum cholestoral _126_226 | ... | resting electrocardiographic_0 | resting electrocardiographic_1 | ele |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | ... | 1 | 0 | |
| **1** | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | ... | 0 | 1 | |
| **2** | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | ... | 1 | 0 | |
| **3** | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | ... | 0 | 1 | |
| **4** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | ... | 0 | 1 | |

5 rows × 24 columns

با توجه به اینکه مقدار `min_support` برابر ۰,۲ تنظیم شده است یعنی آیتم ست هایی می بایست استخراج شوند که حداقل در ۲۰ درصد از کل دیتاست رخ داده باشند. به این ترتیب تعداد ۱۴۰ آیتم ست از دیتاست استخراج شده است که برای مثال آیتم ست (target_1, age_29_49) در ۲۰,۱۳ درصد از کل رکورد های دیتاست رخ داده است یعنی در ۲۰,۱۳ درصد از رکورد های دیتاست برای افرادی که سن آن ها در رنج 29 الی 49 بوده متغیر خروجی نیز 1 شده است.

| | support | itemsets |
|---|---|---|
| **22** | 0.201320 | (target_1, age_29_49) |
| **116** | 0.201320 | (chest pain type_0, target_0, resting electroc... |
| **126** | 0.201320 | (maximum heart rate achieved_160_202, resting ... |
| **119** | 0.204620 | (chest pain type_2, target_1, exercise induced... |
| **112** | 0.204620 | (target_0, gender_1, exercise induced angina_1) |
| **...** | ... | ... |
| **39** | 0.574257 | (gender_1, fasting blood sugar_0) |
| **72** | 0.577558 | (exercise induced angina_0, fasting blood suga... |
| **15** | 0.673267 | (exercise induced angina_0) |
| **4** | 0.683168 | (gender_1) |
| **9** | 0.851485 | (fasting blood sugar_0) |

140 rows × 2 columns

```
(c
rules = association_rules(frq_items_apriori, metric ="lif
t", min_threshold = 1)
no_stroke = rules[rules['consequents'] == frozenset({'tar
get_0'})]
stroke    = rules[rules['consequents'] == frozenset({'tar
get_1'})]
```

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|---|---|
| 4 | (chest pain type_0, resting electrocardiograph... | (target_0) | 0.257426 | 0.455446 | 0.201320 | 0.782051 | 1.717113 | 0.084077 | 2.498544 |
| 21 | (exercise induced angina_1, gender_1) | (target_0) | 0.254125 | 0.455446 | 0.204620 | 0.805195 | 1.767928 | 0.088880 | 2.795380 |
| 36 | (exercise induced angina_1, fasting blood suga... | (target_0) | 0.273927 | 0.455446 | 0.204620 | 0.746988 | 1.640126 | 0.079861 | 2.152287 |
| 79 | (fasting blood sugar_0, resting electrocardiog... | (target_0) | 0.399340 | 0.455446 | 0.211221 | 0.528926 | 1.161337 | 0.029344 | 1.155984 |
| 106 | (gender_1, resting electrocardiographic_0) | (target_0) | 0.339934 | 0.455446 | 0.217822 | 0.640777 | 1.406923 | 0.063000 | 1.515922 |

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|---|---|
| 1 | (age_29_49) | (target_1) | 0.287129 | 0.544554 | 0.201320 | 0.701149 | 1.287565 | 0.044963 | 1.523991 |
| 15 | (chest pain type_2, exercise induced angina_0) | (target_1) | 0.250825 | 0.544554 | 0.204620 | 0.815789 | 1.498086 | 0.068033 | 2.472419 |
| 41 | (fasting blood sugar_0, gender_1, exercise ind... | (target_1) | 0.356436 | 0.544554 | 0.204620 | 0.574074 | 1.054209 | 0.010522 | 1.069307 |
| 48 | (serum cholestoral _126_226, fasting blood sug... | (target_1) | 0.323432 | 0.544554 | 0.204620 | 0.632653 | 1.161781 | 0.028494 | 1.239824 |
| 73 | (gender_0, exercise induced angina_0) | (target_1) | 0.244224 | 0.544554 | 0.211221 | 0.864865 | 1.588206 | 0.078228 | 3.370297 |

f ) همانگونه که در شکل فوق مشخص شده است یک قاعده به صورت

{age_29_49} → {target_1}

استخراج شده است به طور کلی می توان گفت :

پارامتر antecedent support نمایش میدهد که آیتم ست antecedent در چه درصدی از کل دیتاست تکرار شده است .

برای مثال در قاعده ۱ مقدار این پارامتر نشان می دهد که age_29_49 در 28.71 درصد از رکورد های کل دیتاست موجود بوده است.

پارامتر consequent support نمایش میدهد که آیتم ست consequent در چه درصدی از کل دیتاست تکرار شده است.

برای مثال در قاعده ۱ مقدار این پارامتر نشان می دهد که target_1 در 54.45 درصد از رکورد های کل دیتاست موجود بوده است.

پارامتر support نشان می دهد که در چند درصد از کل رکورد های موجود در دیتاست به طور همزمان دو آیتم ست consequent و antecedent باهم ظاهر شده اند.

برای مثال در قاعده ۱ مقدار این پارامتر نشان می دهد که age_29_49 و target_1 به طور همزمان در 20.13 درصد از رکورد های کل دیتاست باهم ظاهر شده اند.

پارامتر confidence نشان می دهد که در چند درصد از رکورد هایی که حاوی آیتم ست antecedent است آیتم ست consequent نیز ظاهر شده است.

برای مثال در قاعده ۱ مقدار این پارامتر نشان می دهد که در 70.11 درصد از رکورد های دیتاست که سن افراد در آن در رنج 29 الی 49 بوده است مقدار متغیر هدف یعنی target یک بدست آمده است.

پارامتر Lift نشان می دهد اگر دو قاعده مستقل از هم بودند تا چه میزان support مشاهده می شد. میزان lift بالاتر از ۱ نشان می دهد که قواعد مستقل از هم هستند.