باسمه تعالی

تکلیف سری ششم داده کاوی

سارا برادران (شماره دانشجویی: ۹۶۲۴۱۹۳)

سوال ۱)

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

b,c) از متد ()info استفاده کرده و اطلاعاتی درباره دیتافریم ایجاد شده بدست می آوریم. با توجه به این اطلاعات دیتافریم Camera حاوی یک ستون است که datatype آن object می باشد و سایرستون ها مقادیر عددی را شامل می شوند.

در گام اول ستون دسته ای (Model) را به نحوی به یک ستون عددی تبدیل می نماییم، برای اینکار از روش خوشه بندی استفاده کرده و برای مثال تمام دوربین هایی که متعلق به برند سامسونگ هستند را در یک خوشه قرار می دهیم، دوربین هایی که متعلق به برند سونی هستند در خوشه دیگر و... به همین ترتیب خوشه بندی مدل ها صورت میپذیرد و در مجموع تعداد۲۱ خوشه ایجاد میگردد.

```
Clusters for Model Column
['Agfa', 'Canon', 'Casio', 'Contax', 'Epson', 'Fujifilm', 'HP', 'JVC',
'Kodak', 'Kyocera', 'Leica', 'Nikon', 'Olympus', 'Panasonic', 'Pentax',
'Ricoh', 'Samsung', 'Sanyo', 'Sigma', 'Sony', 'Toshiba']
```

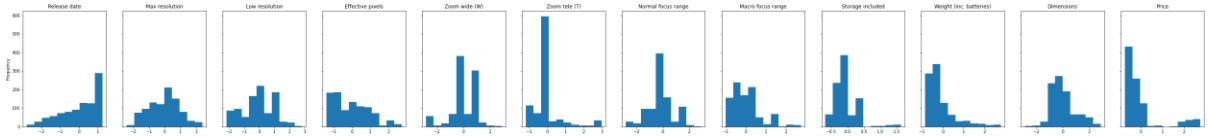
در ادامه مقادیر ستون Model را با مقدار خوشه های بدست آمده جایگذاری کرده و متغیر های dummy را ایجاد می نماییم. تمام مقادیر صفر موجود در دیتاست اولیه (بدون در نظر گیری ستون های dummy) را با مقدار NaN جایگذاری کرده و نهایتا با استفاده از متد SimpleImputer تمام مقادیر Null را با میانه دیگر مقادیر همان ستون جایگزین می کنیم. همانطور که در تصویر زیر مشخص است missing value های موجود در دیتافریم پس از این کار با مقادیر مناسب جایگذاری می گردد.

```
===== the number of missing value in each column at the beginning =====
Release date                0
Max resolution              1
Low resolution             54
Effective pixels           35
Zoom wide (W)              85
Zoom tele (T)              85
Normal focus range        137
Macro focus range         128
Storage included          125
Weight (inc. batteries)    23
Dimensions                 16
Price                       0
dtype: int64
===== the number of missing value in each column after imputing =====
Release date              0
Max resolution            0
Low resolution            0
Effective pixels          0
Zoom wide (W)             0
```

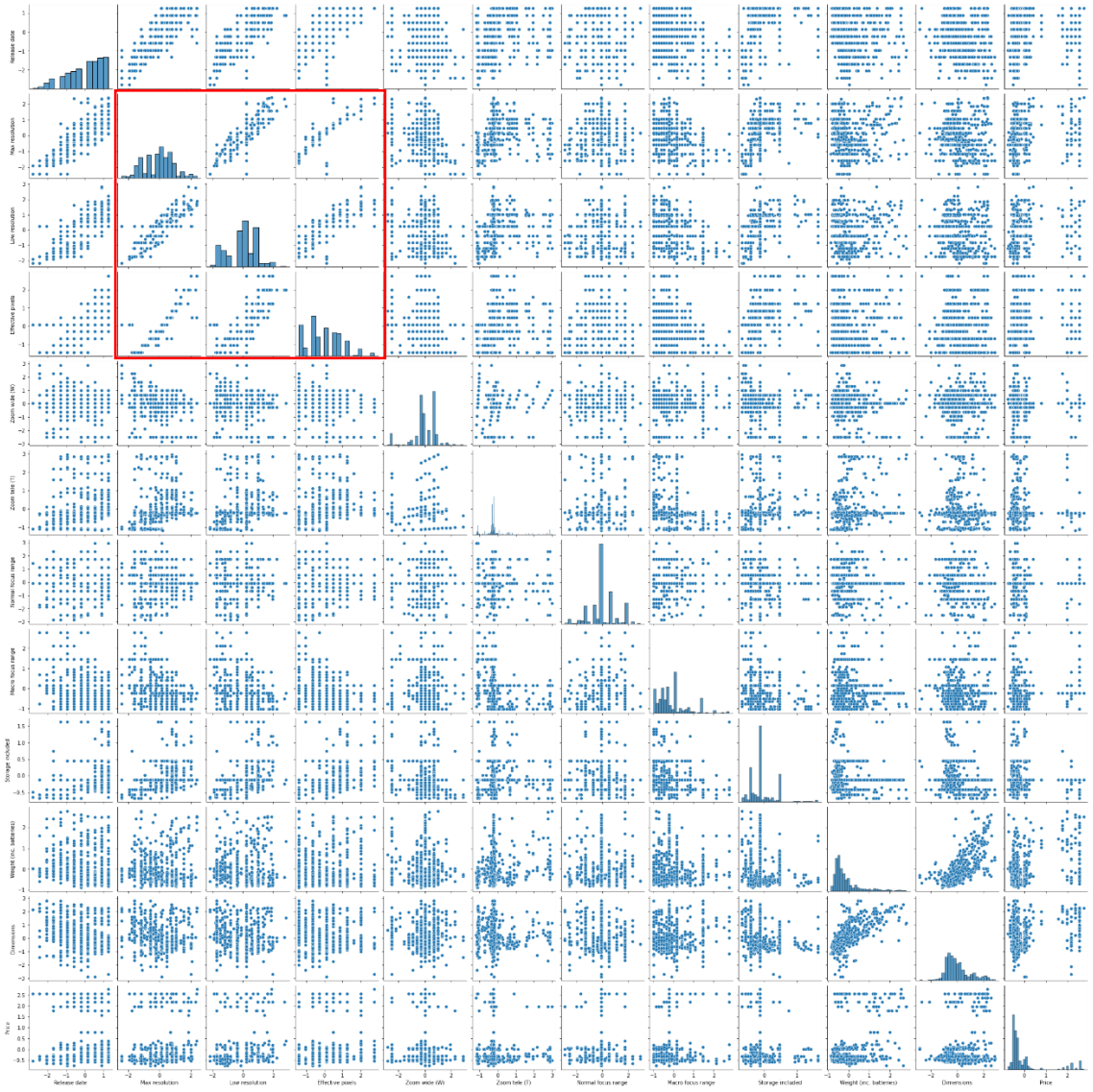
```
Zoom tele (T)                0
Normal focus range           0
Macro focus range            0
Storage included             0
Weight (inc. batteries)      0
Dimensions                   0
Price                        0
dtype: int64
```

با توجه به اینکه رنج متغیر های عددی متنوع داخل دیتافریم داریم نیاز است با روشی این رنج هارا یکسان سازی نماییم لذا رنج ستون های عددی دیتافریم را با روش zscore یکسان سازی کرده و سپس داده های پرت را با روش zscore از دیتافریم حذف نموده و در انتها ستون های عددی و ستون های dummy ایجاد شده در مراحل قبل را concat کرده و یک دیتافریم یکپارچه آماده می کنیم.

توزیع مقادیر ستون های عددی دیتافریم پس از zscore نیز به صورت زیر می باشد:

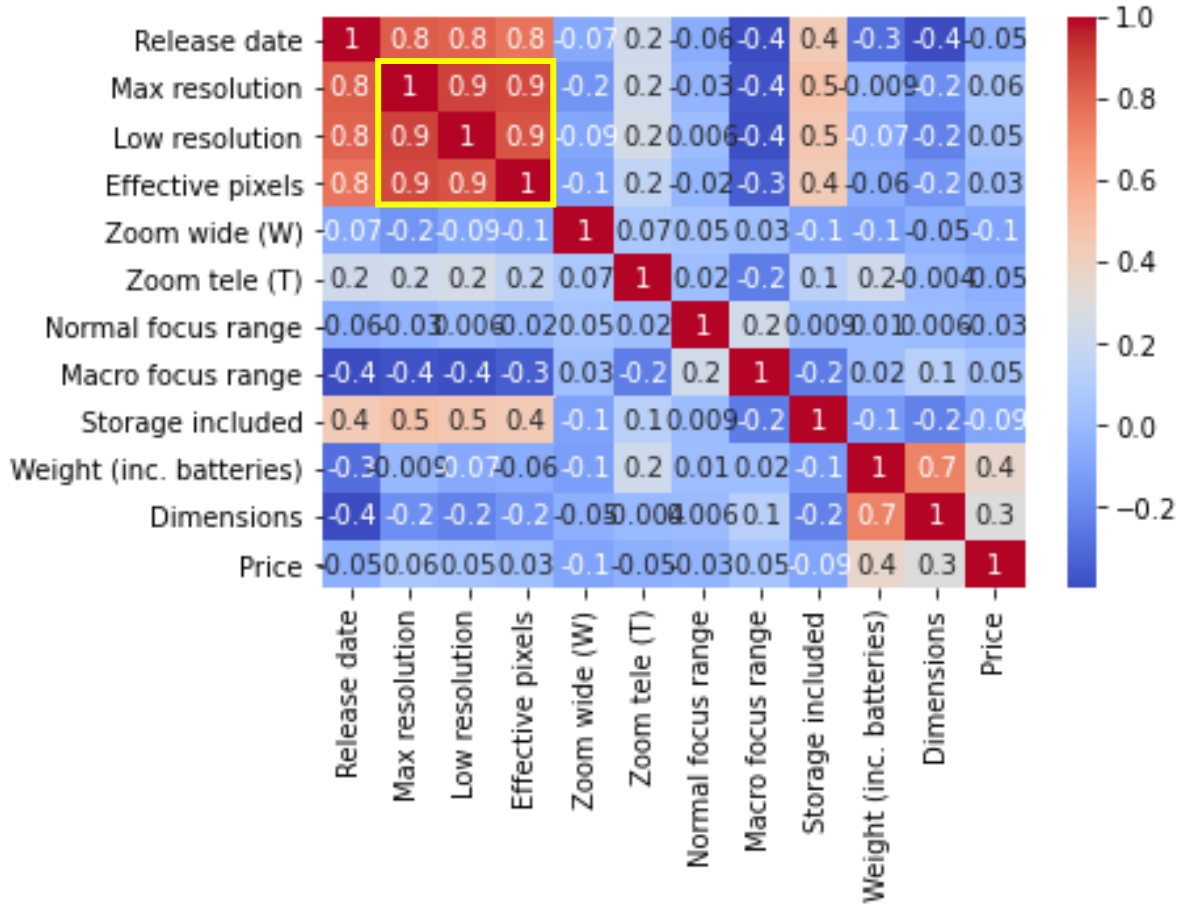


d) نمودار pairplot نظیر ستون های عددی دیتافریم به صورت زیر می باشد:



همانطور که از نمودار بدست آمده فوق مشخص است برخی از متغیر ها رابطه نسبتا خطی با هم دارند. به طور مشخص دو متغیر Max resolution و Effective pixels به نظر می رسد بیشترین میزان ارتباط را با یکدیگر دارند. این رابطه نسبتا خطی برای دو متغیر Max resolution و Low resolution و دو متغیر Low resolution و Effective pixels نیز برقرار است.

برای اثبات بهتر این موضوع می توان نمودار heatmap را نیز رسم نمود:



همانگونه که از نمودار heatmap هم بدست می آید دو متغیر { Max resolution و Effective pixels } و { Max resolution و Low resolution } و { Effective pixels و Low resolution } دوبه دو مقدار correlation به میزان 0.9 دارند که بسیار به 1 نزدیک می باشد. و این نشان از ارتباط مستقیم و قوی دو به دوی این متغیر هاست.

(e

```
x = Camera.drop(columns=['Price'])
y = Camera[['Price']]
```

f) 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 = 5)
```

g) به کمک متد LinearRegression کتابخانه sklearn مدل را تشکیل می دهیم و داده های یادگیری را به آن fit می کنیم.

```
reg = LinearRegression()
reg.fit(x_train, y_train)
```

h) مقادیر _intercept و _coef برای مدل بدست آمده مطابق زیر می باشد :

```
intercept =  [-0.11426346]

coef =
[0.03835546   0.12225162  0.07991715  -0.09510279  0.072148
-0.22023072  -0.03180925  0.07840773  -0.1634233   0.50897977
0.068275      0.26117423 -0.05361409  -0.06030488  0.37099139
-0.07052242  -0.18518896 -0.21885141  -0.62582345 -0.31988654
0.93700384    0.12970012  0.00580881   0.62489612  1.61447844
-0.05810955  -0.02438444 -0.14180303  -0.06091987 -1.96503787
0.19800629 -0.35761274]
```

مقدار intercept در حقیقت مقدار پیش بینی شده توسط مدل برای رکوردی است که مقادیر همه feature های آن 0 باشد. این موضوع به صورت دستی نیز تست شده و نتیجه در زیر مشخص است :

```
x = [0,0,0,0,0,0,0,0,
     0,0,0,0,0,0,0,0,
     0,0,0,0,0,0,0,0,
     0,0,0,0,0,0,0,0]

test = pd.DataFrame(columns=x_test.columns)
test.loc[1] = x
display(test)

r = reg.predict(test)
print(r == reg.intercept_)
```

	Release date	Max resolution	Low resolution	Effective pixels	Zoom wide (W)	Zoom tele (T)	Normal focus range	Macro focus range	Storage included	Weight (inc. batteries)	...	Nikon	Olympus	Panasonic	Pentax	Ricoh	Samsung	S
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	

1 rows × 32 columns

```
[[ True]]
```

مقادیر آرایه coef نیز در حقیقت ضرایب هر یک از feature ها را در مدل نشان می دهد.

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

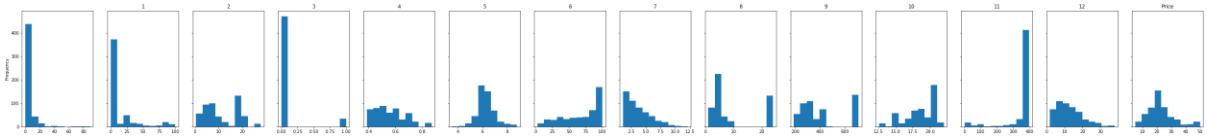
```
features['Price'] = target
```

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 :

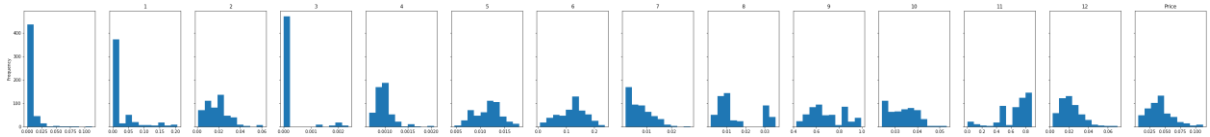


بنابر نتایج زیر رنج مقادیر ستون های گوناگون در دیتافریم boston بسیار متنوع است برای مثال ستون ۱ حاوی مقادیری در رنج ۰ تا ۱۰۰ است درحالی که ستون ۴ حاوی مقداری کمتر از ۱ است. لذا می بایست به نحوی این رنج مقادیر را یکسان سازی نماییم :

```
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 = 0.3, random_state = 5)
```

d) به کمک متد LinearRegression 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:

```
reg = LinearRegression()
reg.fit(x_train, y_train)
y_predict = reg.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:

```
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.

با توجه به نتایج بدست آمده MAE بسیار کوچک است و در حدود ۶ هزارم ارزیابی شده است البته نکته حائز اهمیت این است که رنج مقادیر نیز می بایست در نظر گرفته شود با توجه به اینکه رنج مقادیر ستون ها پس از نرمالایز شدن از حدود 10^{-5} تا 1 تغییر می کند مقدار 0.006 بدست آمده برای MAE مقدار مناسبی ارزیابی می شود و این حاکی از دقت نسبتا مناسب مدل است به همین ترتیب چون میزان MAE بسیار کمتر از ۱ است طبیعتا تک تک خطاها از ۱ کمتر بوده است و لذا انتظار میرود MSE نیز بسیار کوچکتر از MAE باشد که این امر در نتایج بدست آمده نیز به خوبی مشخص است و مقدار MSE چیزی در حدود 0.00007 است.

f)

```
from sklearn.model_selection import cross_val_score
# 2_f
cvs = cross_val_score(reg, x_train, y_train, cv=5)
print(cvs)
print('mean = ', cvs.mean())
```

```
[0.83578354 0.84730115 0.88960487 0.81785426 0.88363407]
mean =  0.8548355805342333
```

سوال ۳)

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

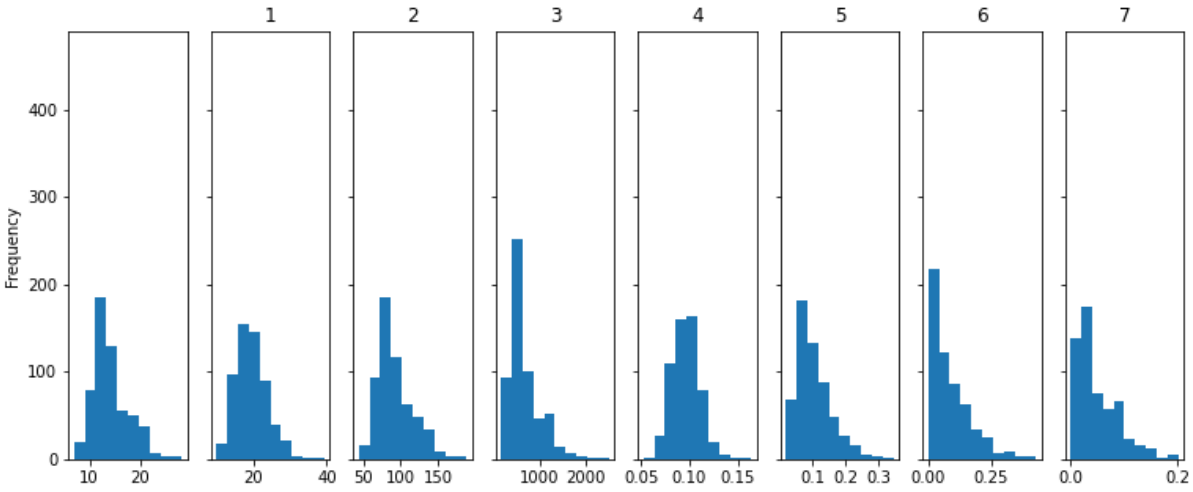
```
features['Cancer'] = target
```

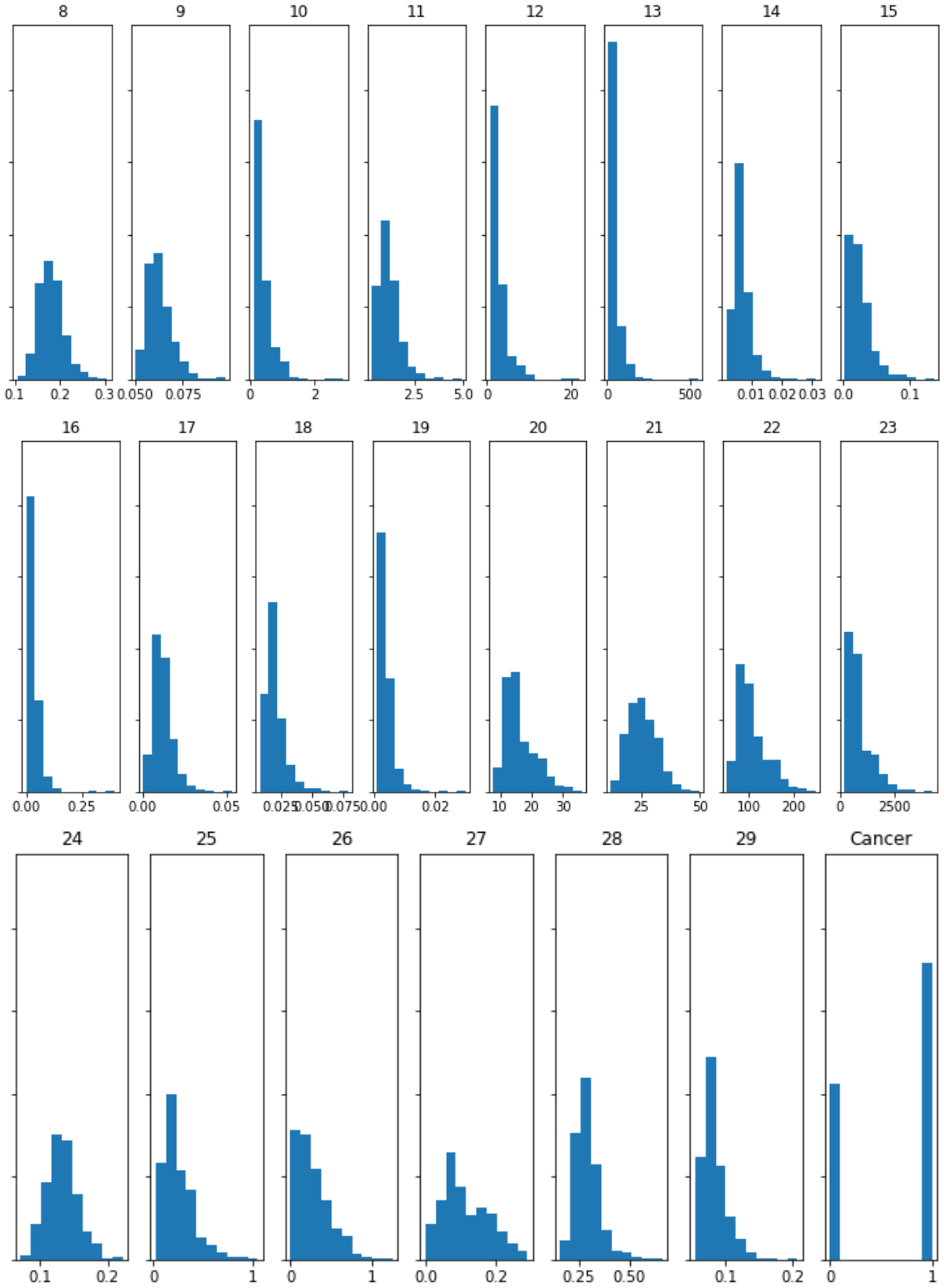
c) همانگونه که مشخص است دیتافریم bearst_cancerحاوی مقدار null نمی باشد.

```
# 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 و به کمک متد train_test_split داده های تست و یادگیری را جدا میکنیم:

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

e) به کمک متد MLPClassifier 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:

```
mlp = MLPClassifier(hidden_layer_sizes=(20, 20, 20),
max_iter=1000, verbose=False)
mlp.fit(x_train, y_train.values.ravel())
y_predict = mlp.predict(x_test)
```

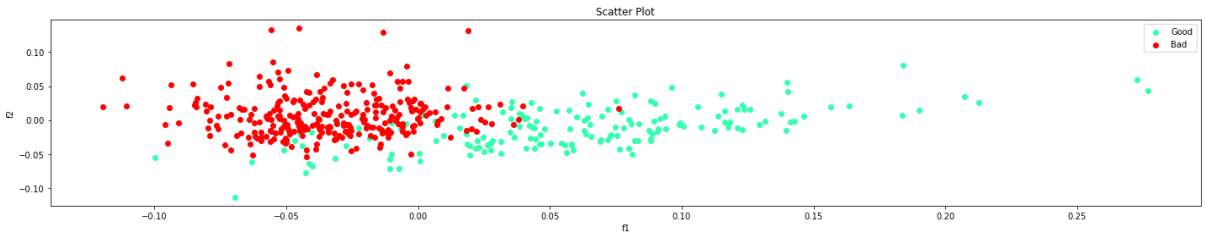
f) میزان دقت بدست آمده برای این مدل 0.9210526315789473 می باشد. یعنی حدودا ۹۲ درصد دقت در تست داشته ایم.

g) از متد PCA کتابخانه sklearn استفاده می کنیم و پارامتر ورودی n_components را برابر ۲ قرار می دهیم:

```
pca = PCA(n_components=2)
pca.fit(x_train)
train_pca = pd.DataFrame(pca.transform(x_train), columns=
['f1', 'f2'])
train_pca['Cancer'] = y_train.values.ravel()

pca = PCA(n_components=2)
pca.fit(x_test)
test_pca = pd.DataFrame(pca.transform(x_test), columns=['
f1', 'f2'])
test_pca['Cancer'] = y_test.values.ravel()
```

نمودار scatter plot داده های موجود به صورت زیر می باشد که نواحی قرمز رنگ سرطانی و سبز رنگ خوش خیم است:



h) مجددا شبکه عصبی را اینبار با داده های بدست امده از متد PCA اموزش داده و داده های تست را ارزیابی می کنیم:

میزان دقت بدست آمده برای این مدل 0.9035087719298246 می باشد. یعنی حدودا ۹۰ درصد دقت در تست داشته ایم.

i) واضح است که dimension reduction اندکی تاثیر منفی در دقت مدل داشته است چرا که ۳۰ ویژگی موثر در تعیین متغیر هدف در دو ویژگی جمع بندی شده است پس کاهش دقت طبیعی است اما با وجود این حجم از کاهش ابعاد تنها ۲ درصد دقت مدل تحت تاثیر قرار گرفته است که مزیت استفاده از PCA در دیتاست های با ابعاد زیاد را نشان می دهد.

سوال ۴)

```
Pip install mlxtend (a
```

b) به کمک متد read_csv کتابخانه pandas ابتدا دیتاست Basket را می خوانیم.

```
Basket = pd.read_csv('Basket.csv', sep=',', header=None,
error_bad_lines=False)
```

c) می بایست تغییراتی در دیتافریم ایجاد نماییم تا بتوان از متد apriori بر روی آن استفاده نمود. به این ترتیب یک دیتا فریم جدید ایجاد می کنیم که ستون های نظیر آن تمام اقلام موجود در لیست خرید باشند سپس به ازای هر رکورد در دیتافریم اصلی کالاهای خریداری شده را بدست آورده و در دیتافریم ثانویه ایجاد شده مقدار ستون نظیر هر یک از این اقلام را ۱ و دیگر اقلامی که در این رکورد خریداری نشده اند را ۰ قرار می دهیم. به این ترتیب دیتا فریمی مطابق زیر بدست می آید:

	Panner	Butter	Milk	Bread	Ghee	Yougurt	Lassi	Tea Powder	Sugar	Coffee Powder	Sweet	Cheese
0	0	1	0	0	1	1	1	0	0	1	0	1
1	0	0	0	0	1	0	0	0	0	1	0	0
2	0	1	0	0	0	0	1	1	0	0	0	1
3	1	1	0	1	0	0	0	1	0	1	0	1
4	0	1	0	0	0	1	0	0	1	1	1	1

با توجه به اینکه مقدار min_support برابر ۰,۱ تنظیم شده است یعنی آیتم ست هایی می بایست استخراج شوند که حداقل در ۱۰ درصد از کل لیست خرید رخ داده باشند. به این ترتیب تعداد ۷۸ آیتم ست از لیست خرید استخراج شده است که برای مثال آیتم ست (Cheese, Coffee Powder) در ۱۲,۲۹ درصد از خریدها رخ داده است یعنی در ۱۲,۲۹ درصد از کل خرید ها آیتم های cheese و coffee powder با یکدیگر خریداری شده اند. یا میزان support برای آیتم ست Milk برابر ۰,۳۵۵۷ بدست آمده است به این معنا که در ۳۵,۵۷ درصد از کل خرید ها ایتم Milk وجود داشته است.

	support	itemsets
0	0.346541	(Panner)
1	0.349404	(Butter)
2	0.355703	(Milk)
3	0.349404	(Bread)
4	0.356505	(Ghee)
...	...	...
73	0.11372	(Sugar, Sweet)
74	0.11727	(Sugar, Cheese)
75	0.115437	(Sweet, Coffee Powder)
76	0.122996	(Cheese, Coffee Powder)
77	0.12082	(Sweet, Cheese)

78 rows × 2 columns

d) تفاوت دو الگوریتم apriori و fpgrowth به صورت خلاصه در شکل زیر نمایش داده شده است :

FP growth algorithm	Apriori algorithm
FP growth algorithm is faster than Apriori algorithm.	It is slower than FP growth algorithm.
FP growth algorithm is an array based algorithm.	Apriori algorithm is a tree-based algorithm.
FP growth algorithm required only two database scan.	It requires multiple database scan to generate a candidate set.
It uses depth-first search	It uses breadth-first search.

الگوریتم `fpgrowth` به صورت `DFS` آیتم ست ها را استخراج می کند در حالی که الگوریتم `apriori` به صورت `BFS` اینکار را انجام می دهد.

نتایج بدست آمده در مرحله قبل توسط `fpgrowth` هم بدست خواهد آمد اما اینبار با ترتیب متفاوت. لذا دیتافریم حاوی آیتم ست های استخراج شده اینبار هم ۷۸ سطر دارد.

	support	itemsets
0	0.346541	(Panner)
1	0.349404	(Butter)
2	0.355703	(Milk)
3	0.349404	(Bread)
4	0.356505	(Ghee)
...	...	...
73	0.11372	(Sugar, Sweet)
74	0.11727	(Sugar, Cheese)
75	0.115437	(Sweet, Coffee Powder)
76	0.122996	(Cheese, Coffee Powder)
77	0.12082	(Sweet, Cheese)

78 rows × 2 columns

اگر توسط متد `sort_values` هر دو دیتافریم بدست آمده را بر اساس میزان `support` مرتب نماییم مشخص می شود که محتویات هر دو دیتافریم یکسان است اما بنابر تفاوت دو الگوریتم رکورد ها با ترتیب متفاوتی استخراج شده است :

	support	itemsets
61	0.109711	(Panner, Tea Powder)
48	0.110284	(Butter, Tea Powder)
50	0.110742	(Panner, Lassi)
75	0.111315	(Tea Powder, Ghee)
31	0.111773	(Bread, Tea Powder)
...	...	...
2	0.353298	(Coffee Powder)
7	0.355589	(Yougurt)
6	0.355703	(Milk)
9	0.356505	(Ghee)
0	0.356734	(Cheese)

78 rows × 2 columns

	support	itemsets
41	0.109711	(Panner, Tea Powder)
38	0.110284	(Butter, Tea Powder)
33	0.110742	(Panner, Lassi)
45	0.111315	(Tea Powder, Ghee)
40	0.111773	(Bread, Tea Powder)
...	...	...
3	0.353298	(Coffee Powder)
2	0.355589	(Yougurt)
11	0.355703	(Milk)
1	0.356505	(Ghee)
0	0.356734	(Cheese)

78 rows × 2 columns

(e
```
rules = association_rules(frq_items_apriori, metric ="lif
t", min_threshold = 1)
```

```
1 # 4_e
2 rules = association_rules(frq_items_apriori, metric ="lift", min_threshold = 1)
3 print(rules)
```

```
      antecedents       consequents  antecedent support  consequent support  \
0  (Coffee Powder)           (Ghee)            0.353298            0.356505
1           (Ghee)  (Coffee Powder)            0.356505            0.353298

    support  confidence      lift  leverage  conviction
0  0.125973    0.356564  1.000166  0.000021    1.000092
1  0.125973    0.353357  1.000166  0.000021    1.000091
```

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

{coffee powder} → {ghee}

و یک قاعده دیگر به صورت {coffee powder} → {ghee}

استخراج شده است به طور کلی می توان گفت :

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

برای مثال در قاعده 0 مقدار این پارامتر نشان می دهد که coffee powder در ۳۵,۳۲ درصد از رکورد های کل لیست خرید موجود بوده است.

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

برای مثال در قاعده 0 مقدار این پارامتر نشان می دهد که Ghee در ۳۵,۶۵ درصد از رکورد های کل لیست خرید موجود بوده است.

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

برای مثال در قاعده 0 مقدار این پارامتر نشان می دهد که coffee powder و Ghee در ۱۲,۵۹درصد از رکورد های کل لیست خرید باهم خریداری شده اند.

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

برای مثال در قاعده 0 مقدار این پارامتر نشان می دهد که Ghee در ۳۵,۶۵ درصد از رکورد های لیست خرید که حاوی کالای coffee powder بوده اند موجود بوده 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.

بنابر تعاریف ارائه شده نتایج فوق به صورت دستی نیز محاسبه شده است که دقیقا همان مقادیر بدست آمده از طریق متد اماده association_rules خواهد بود:

```
In [25]:
# support for both rules 1 & 0
len(df[(df['Coffee Powder'] == 1) & (df['Ghee'] == 1)]) / len(df)

Out[25]: 0.12597343105817682

In [27]:
# confidence for rule 1
len(df[(df['Coffee Powder'] == 1) & (df['Ghee'] == 1)]) / len(df[df['Ghee'] == 1])

Out[27]: 0.35335689045936397

In [28]:
# confidence for rule 0
len(df[(df['Coffee Powder'] == 1) & (df['Ghee'] == 1)]) / len(df[df['Coffee Powder'] == 1])

Out[28]: 0.3565640194489465
```