مقدمه:

در این پروژه از ما خواسته شده است که دیتاست سرطان سینه(breast cancer) با رسم نمودار فیچر ها و روابط بین آن ها بررسی کرده و با اجرای از الگوریتم های classification یادگیری ماشین و آموزش مدل مورد نظر روی دیتاست دقت هر مدل را گزارش کنیم و درنهایت آن ها را با یکدیگر مقایسه نماییم.

نمایش و تحلیل دیتاست breast cancer:

در ابتدا با استفاده از ماژول pandas دیتاست مورد نظر که در یک فایل csv. ذخیره شده است را در پروژه مان به صورت یک DataFrame باگذاری می کنیم و سپس کل دیتاست به همراه ابعاد آن را برای مطمئن شدن در jupyter notebook نمایش میدهیم.

Breast Cancer data set configirations:

```
In [2]: df = pd.read_csv('dataR2.csv')   # loading the data set in to the pandas dataframe.
        df
```

Out[2]:

	Age	BMI	Glucose	Insulin	HOMA	Leptin	Adiponectin	Resistin	MCP.1	Classification
0	48	23.500000	70	2.707	0.467409	8.8071	9.702400	7.99585	417.114	1
1	83	20.690495	92	3.115	0.706897	8.8438	5.429285	4.06405	468.786	1
2	82	23.124670	91	4.498	1.009651	17.9393	22.432040	9.27715	554.697	1
3	68	21.367521	77	3.226	0.612725	9.8827	7.169560	12.76600	928.220	1
4	86	21.111111	92	3.549	0.805386	6.6994	4.819240	10.57635	773.920	1
...	...	...	...	...	...	...	...	...	...	...
111	45	26.850000	92	3.330	0.755688	54.6800	12.100000	10.96000	268.230	2
112	62	26.840000	100	4.530	1.117400	12.4500	21.420000	7.32000	330.160	2
113	65	32.050000	97	5.730	1.370998	61.4800	22.540000	10.33000	314.050	2
114	72	25.590000	82	2.820	0.570392	24.9600	33.750000	3.27000	392.460	2
115	86	27.180000	138	19.910	6.777364	90.2800	14.110000	4.35000	90.090	2

116 rows × 10 columns

شکل ۱

همانطور که در شکل ۱ مشخص است یک ساختار کلی از دیتاست را در notebook نمایش داده شده است و همچنین این دیتاست شامل ۱۱۶ ردیف که همان سمپل ها یا رکورد ها می باشند می باشد و ۱۰ ستون داریم که ستون آخر ستون لیبل های مسئله یا همان متغیر های وابسته (dependent) و ۹ ستون دیگر فیچر های ورودی مسئله یا همان متغیر های (independent) می باشند. توجه شود که ما ستون لیبل ها را از فیچر ها جدا می کنیم.

در ادامه برای تحلیل بیشتر و شناخت بهتر دیتاست مورد نظر اطلاعات کلی دیتاست مورد نظر را نشان میدهیم. برای این کار از یکی از توابع داخل pandas کمک میگیریم که describe نام دارد و یک اطلاعات کلی از دیتاست مورد مطالعه بدست می آوریم. این موارد در شکل ۲ آمده است.

```
In [5]: features.describe()
```

Out[5]:

	Age	BMI	Glucose	Insulin	HOMA	Leptin	Adiponectin	Resistin	MCP.1
count	116.000000	116.000000	116.000000	116.000000	116.000000	116.000000	116.000000	116.000000	116.000000
mean	57.301724	27.582111	97.793103	10.012086	2.694988	26.615080	10.180874	14.725966	534.647000
std	16.112766	5.020136	22.525162	10.067768	3.642043	19.183294	6.843341	12.390646	345.912663
min	24.000000	18.370000	60.000000	2.432000	0.467409	4.311000	1.656020	3.210000	45.843000
25%	45.000000	22.973205	85.750000	4.359250	0.917966	12.313675	5.474283	6.881763	269.978250
50%	56.000000	27.662416	92.000000	5.924500	1.380939	20.271000	8.352692	10.827740	471.322500
75%	71.000000	31.241442	102.000000	11.189250	2.857787	37.378300	11.815970	17.755207	700.085000
max	89.000000	38.578759	201.000000	58.460000	25.050342	90.280000	38.040000	82.100000	1698.440000

شکل ۲

از خروجی بالا میتوان نتیجه گرفت که میانگین سن افرادی که این آزمایش روی آن ها انجام گرفته است حدودا ۵۷ سال بوده اند و کوچکترین فرد در این آزمایش ۲۴ ساله بوده است و مسن ترین فرد در این آزمایش ۸۹ سال داشته است.

و همین طور از انحراف معیار(std) برای سن می توان فهمید که متوسط مجذور اختلاف داده ها از میانگین تقسیم بر درجه آزادی سیستم ۱۶ سال بوده است.

همینطور نیز چارک های اول و دوم و سوم داده ها نیز به ترتیب برای سن ۴۵ و ۵۶ و ۷۱ بوده است.

برای فیچر ها دیگر نیز می توانیم این نتایج را دریافت کنیم به عنوان مثال میانگین انسولین افراد در این آزمایش ۱۰.۱۲ بوده است و بیشترین میزان آن حدود ۵۸.۴۶ بوده و کمترین میزان آن ۲.۴۳ بوده است.

حال با توجه به شکل ۱ و شکل ۲ و بررسی آن ها یک شناخت کلی نسبت به داده ها پیدا کردیم حال می خواهیم به کمک لیبل های دیتاست، دادگان را کمی جزئی تر مورد بررسی قرار بدهیم.

```
In [6]: df_dummy = df.copy(deep = True)
        df_dummy['Classification'].replace({1:'healthy', 2: 'breast cancer'}, inplace = True)
        df_dummy.groupby('Classification').mean()
```

Out[6]:

Classification	Age	BMI	Glucose	Insulin	HOMA	Leptin	Adiponectin	Resistin	MCP.1
breast cancer	56.671875	26.984740	105.562500	12.513219	3.623342	26.596512	10.061167	17.253777	563.016500
healthy	58.076923	28.317336	88.230769	6.933769	1.552398	26.637933	10.328205	11.614813	499.730692

شکل ۳

با توجه به این که در دیتاست لیبل ها برای داشتن سرطان ۲ و عدم داشتن آن مقدار ۱ می باشد، برای نمایش بهتر در خروجی ما آن ها را با یکسری رشته نمایشی جایگزین میکنیم و با استفاده از تابع groupby کتابخانه pandas، با استفاده از میانگین مقادیر را روی لیبل ها یا همان ستون classification فیلتر می کنیم.

توجه شود که این عملیات روی کل دیتاست انجام شده است و نه روی dataFrame جدا شده که در توضیحات قبلی بوده است که features نام داشته است.

توضیحات در شکل ۳ قابل مشاهده می باشند. با توجه به این شکل نتایج جالبی را می توانیم متوجه شویم:

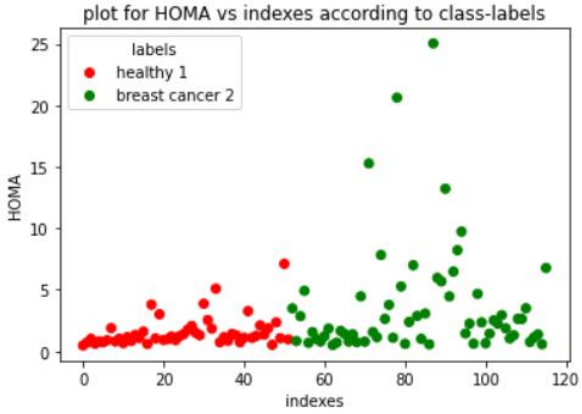
- میانگین سنی افرادی که به سرطان سینه مبتلا شده اند نسبتا پایین تر از افراد سالم در این آزمایش است. از این موضوع می توان نتیجه گرفت که بر خلاف باور عمومی، افراد با سن پایین احتمال بالا تری میتوانند دچار این بیماری شوند و سن کمتر دلیلی بر سرطان نگرفتن نمی باشد.

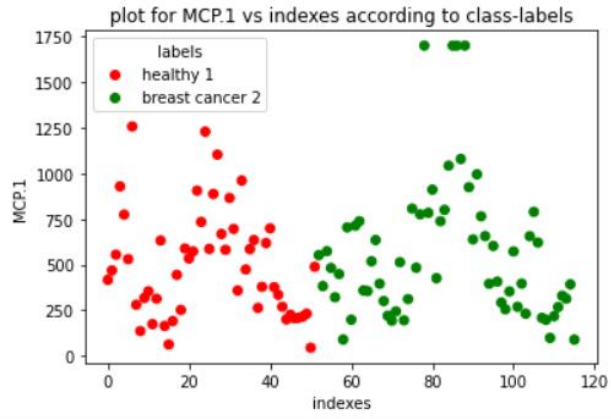
- با توجه به پارامتر HOMA و بررسی نمودار تک فیچری آن، می توان این مورد را دریافت که کسانی که مقدار این فیچر آن ها از ۳.۵ بالاتر بوده است، بیشتر مستعد گرفتن سرطان میباشند.

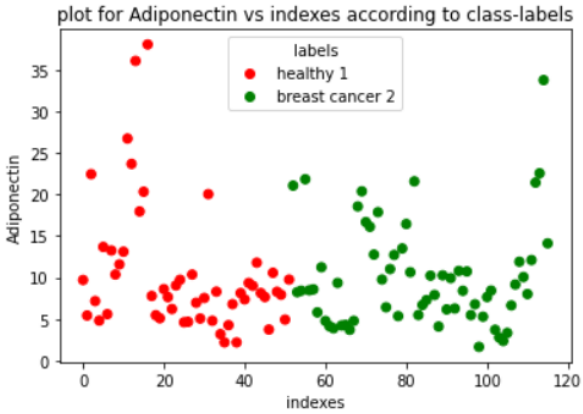
- همانند مورد دوم که ذکر شد این مورد برای پارامتر های MCP.1 ، Resistin ، Glucose و Insulin که نیز صادق می باشند. به عنوان مثال افرادی که میزان گلوکز آن ها به طور میانگین از ۱۰۵.۵ بیشتر است، مستعد مبتلا شدن به سرطان می باشند، همینطور افرادی که میزان MCP.1 آن ها بیشتر از ۵۶۳ میباشد بیشتر به سرطان سینه مبتلا می شوند.

در ادامه نمودار تک فیچری بر مبنای کلاس ها فیچر های دیتاست آورده شده است که در notebook نیز موجود میباشند:


plot for BMI vs indexes according to class-labels
BMI
indexes
labels
healthy 1
breast cancer 2


plot for HOMA vs indexes according to class-labels
HOMA
indexes
labels
healthy 1
breast cancer 2

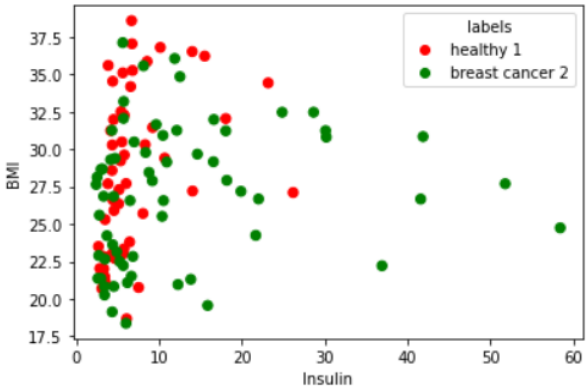

plot for MCP.1 vs indexes according to class-labels
MCP.1
indexes
labels
healthy 1
breast cancer 2

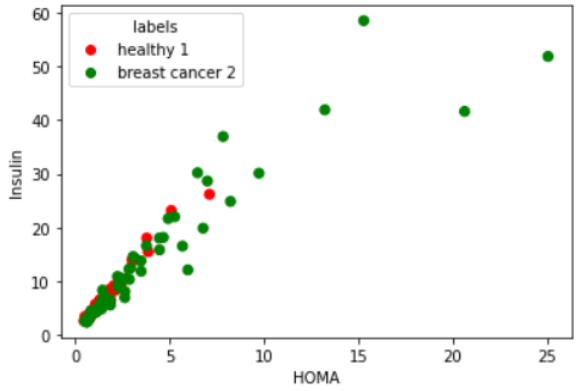


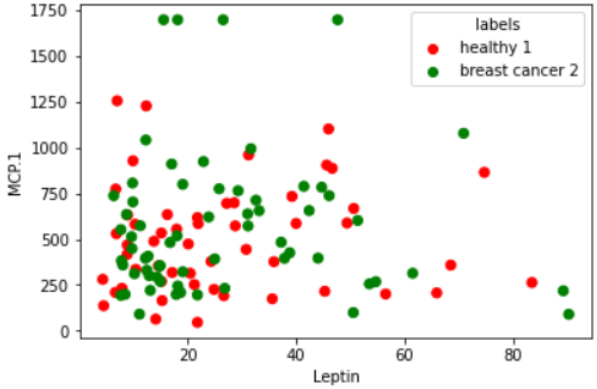
به طور کلی در این نمودار های تک فیچری ما دنبال خط در کل صفحه نیستیم که داده هایمان را جدا کند، بلکه دنبال یک خط افقی می گردیم (روی محور اندیس ها) که برای ما داده ها را تفکیک کنند. با بررسی های انجام شده می توانیم ببینیم که در این مساله نمی شود خطی روی نمودار های تک فیچری ها به صورت افقی فیت کرد که داده های کلاس های مختلف را از یکدیگر جدا کنند. بنابراین در این مساله تک فیچر جدا کننده نداریم.

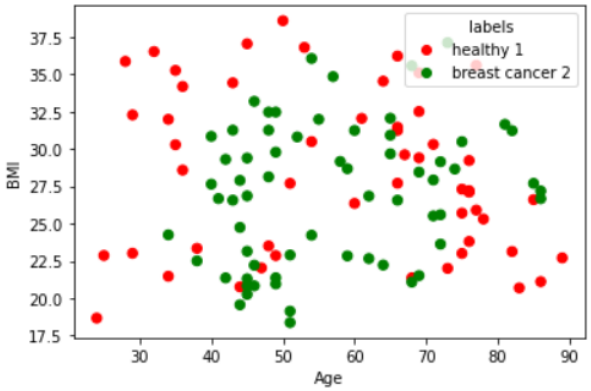
دومین کمکی که این مدل نمودار های تک فیچری می توانند به ما بکنند این است که ببینیم آیا ترکیب ۲ فیچر و رسم آن رکورد ها با آن ۲ فیچر(یکی از فیچر ها محور افقی و دیگری محور عمودی) جدا کننده هستند یا خیر که بین تمامی نمودار های تک فیچره بهترین گزینه ترکیب پارامتر سن و BMI بود و آن ها با یکدیگر نسبت به تمامی ترکیبات دیگر جدا کنندگی بهتری دارند.

در ادامه نمودار های ۲ فیچره دیتاست آورده شده است این نمودار ها در notebook نیز موجود میباشند:









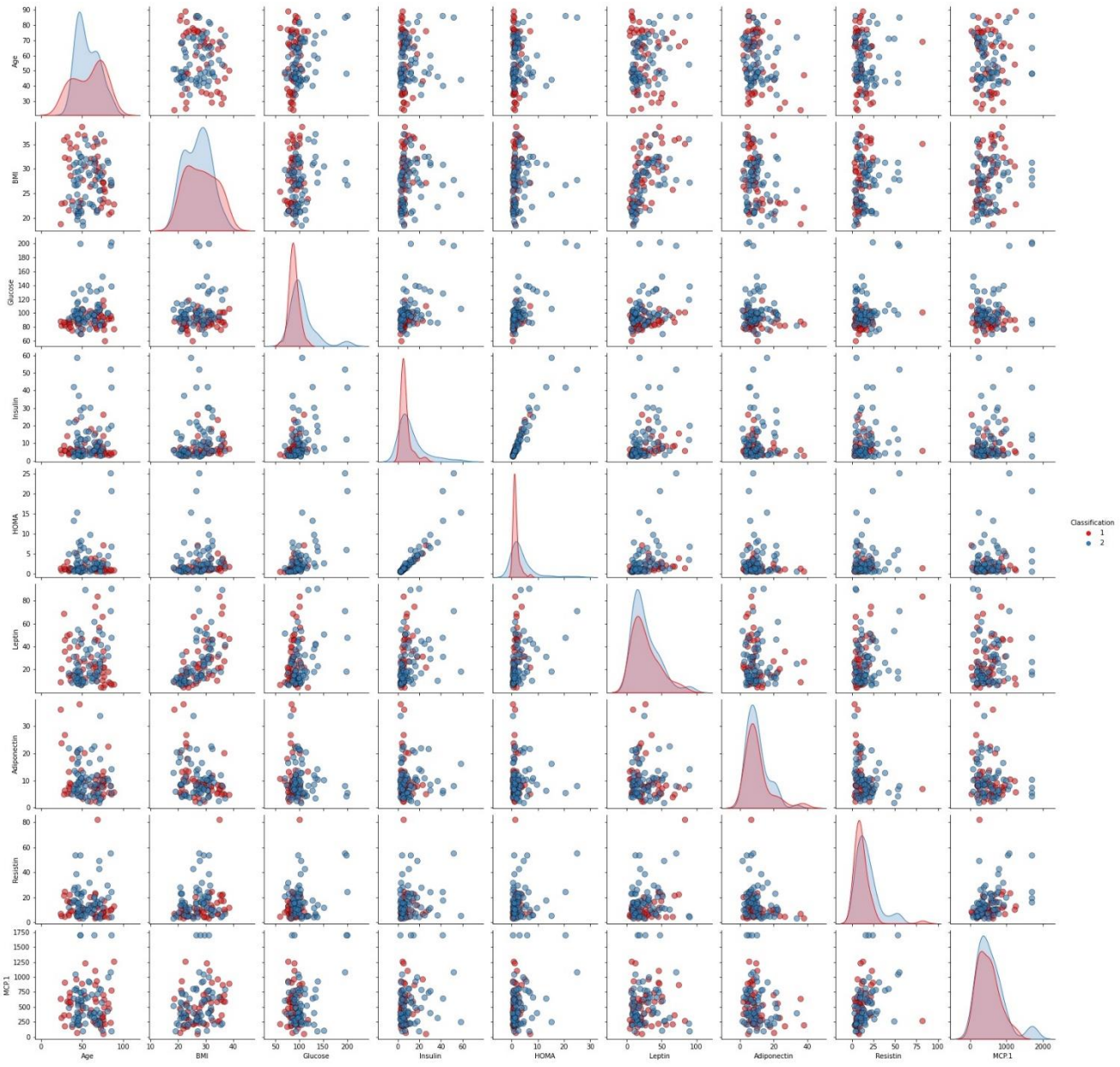
با توجه به نمودار ها تفکیک پذیر ترین نمودار ۲ فیچری که داده ها بهتر جدا میکند نمودار BMI برحسب Age می باشد.

تحلیل جالب دیگری که می توان در رابطه با این نمودار ها انجام داد این است که فیچر های insulin و HOMA همبستگی (correlation) بسیار بالایی دارند و همبستگی آن ها تقریبا ۱ می باشد زیرا تقریبا می شود روی data point های نمودار insulin برحسب HOMA یک خط فیت کرد.

در نمودار MCP.1 برحسب Leptin نیز شاهد correlation نسبتا بالایی هستیم.

در این موارد که داده ها با یکدیگر correlation بالایی دارند میتوان یکی از آن ها را حذف کرد و یا به شکلی از ترکیب آن ها استفاده کرد که این کار ها را می توان به کمک الگوریتم هایی نظیر PCA که برای کاهش ابعاد هستند انجام داد.

در ادامه نیز ماتریس scatter آورده شده است که نمودار های تمامی فیچر ها را نسبت به یکدیگر رسم شده است. این نمودار و کد آن نیز در notebook مورد نظر موجود می باشد.



- ✓ **تمامی plot ها و آنالیز های دیتاست که در صفحات قبلی توضیح داده شد در notebook به نام CancerDataPlots.ipynb موجود می باشند که این فایل در پوشه ایی به نام Plots در فایل ارسالی می باشد.**

استفاده از مدل های یادگیری ماشین برای دیتاست breast cancer:

در این بخش ما به کمک numpy و pandas و ماژول های کتابخانه sklearn اقدام پیاده کردن مدل های classification روی دیتاست میکنیم.

در ابتدا بعد از نشان دادن ابعاد دیتاست، سراغ پیش پردازش داده ها میرویم در ابتدا به کمک numpy و pandas دیتاست را به صورت رندوم shuffle می کنیم. خروجی در شکل ۴ نمایش داده شده است.

In [424]:
```
# df = df.sample(frac = 1)
# df
# suffeling the dataset.
df = df.iloc[np.random.permutation(df.index)].reset_index(drop=True)
print("after suffeling the data set:\n")
df.head(10)
```

```
after suffeling the data set:
```

Out[424]:

	Age	BMI	Glucose	Insulin	HOMA	Leptin	Adiponectin	Resistin	MCP.1	Classification
0	75	25.700000	94	8.079	1.873251	65.9260	3.741220	4.49685	206.802	1
1	38	23.340000	75	5.782	1.069670	15.2600	17.950000	9.35000	165.020	1
2	76	23.800000	118	6.470	1.883201	4.3110	13.251320	5.10420	280.694	1
3	72	25.590000	82	2.820	0.570392	24.9600	33.750000	3.27000	392.460	2
4	45	23.140496	116	4.902	1.402626	17.9973	4.294705	5.26330	518.586	2
5	45	29.384757	90	4.713	1.046286	23.8479	6.644245	15.55625	621.273	2
6	49	29.777778	70	8.396	1.449709	51.3387	10.731740	20.76801	602.486	2
7	48	31.250000	199	12.162	5.969920	18.1314	4.104105	53.63080	1698.440	2
8	64	34.529723	95	4.427	1.037394	21.2117	5.462620	6.70188	252.449	1
9	38	22.499637	95	5.261	1.232828	8.4380	4.771920	15.73606	199.055	2

شکل ۴

و در ادامه بعد از جدا کردن feature ها(X) و class label ها (y) در dataFrame ، به سراغ نرمال کردن feature ها مرویم این مورد در شکل ۵ نشان داده شده است.

Normalizing dataset:

```
In [426]: # normilizing the dataset values.
scaler = preprocessing.MinMaxScaler()
names = X.columns
d = scaler.fit_transform(X)
X = pd.DataFrame(d, columns=names)
X.head(10)
```

Out[426]:

	Age	BMI	Glucose	Insulin	HOMA	Leptin	Adiponectin	Resistin	MCP.1
0	0.784615	0.362714	0.241135	0.100789	0.057188	0.716712	0.057311	0.016312	0.097398
1	0.215385	0.245933	0.106383	0.059792	0.024499	0.127360	0.447834	0.077830	0.072115
2	0.800000	0.268695	0.411348	0.072071	0.057593	0.000000	0.318692	0.024011	0.142110
3	0.738462	0.357271	0.156028	0.006925	0.004189	0.240191	0.882091	0.000761	0.209741
4	0.323077	0.236061	0.397163	0.044085	0.038043	0.159200	0.072523	0.026027	0.286061
5	0.323077	0.545049	0.212766	0.040712	0.023548	0.227255	0.137099	0.156500	0.348197
6	0.384615	0.564497	0.070922	0.106447	0.039959	0.547031	0.249443	0.222563	0.336829
7	0.369231	0.637347	0.985816	0.173663	0.223835	0.160760	0.067285	0.639128	1.000000
8	0.615385	0.799640	0.248227	0.035607	0.023186	0.196591	0.104623	0.044263	0.125019
9	0.215385	0.204349	0.248227	0.050493	0.031136	0.048006	0.085639	0.158779	0.092710

شکل ۵

توجه شود که X همان ماتریس فیچر ها که به ازای هر سمپل دیتا مقادیری دارد می باشد که در پایتون به صورت dataFrame شبیه سازی شده است.

Creating train and test sets:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state = 0)

#y_test_bin = y_test.replace({1 : 0, 2 : 1}, inplace = False).to_numpy()   # -> 1 => cancer , 0 => healthy
# y_test_MinusPlusOne = y_test.replace({1 : -1, 2 : 1}, inplace = False).to_numpy()  # -> 1 => cancer , -1 => healthy
#y_train_bin = y_train.replace({1 : 0, 2 : 1}, inplace = False).to_numpy()
# y_train_MinusPlusOne = y_train.replace({1 : -1, 2 : 1}, inplace = False).to_numpy()

y_train_bin = y_train.to_numpy()
y_test_bin = y_test.to_numpy()
X_train = X_train.to_numpy()
X_test = X_test.to_numpy()

print(X_train[:10, :])     # printing 10 first records of train features matrix.
```

```
[[0.15384615 0.67322368 0.19148936 0.03744556 0.0205317  0.28427922
  0.1645433  0.03062391 0.31885572]
 [0.95384615 0.43594959 0.55319149 0.31195117 0.25668033 1.
  0.34229295 0.0144505  0.02677422]
 [0.8        0.43693926 0.24113475 0.20771757 0.11369495 0.36734172
  0.21137352 0.06598555 0.20052318]
 [0.64615385 0.6367828  0.15602837 0.03121653 0.01538743 0.13858135
  0.07176469 0.00103625 0.3562629 ]
 [0.15384615 0.15339883 0.12765957 0.0185086  0.00813682 0.11933371
  0.31480833 0.04702751 0.18683139]
 [0.64615385 0.64707294 0.21276596 0.12159991 0.06447487 0.48449325
  0.23909534 0.25569654 0.63915583]
 [1.         0.21426353 0.12056738 0.04030128 0.01722246 0.03085996
  0.10812025 0.12328686 0.73232615]
 [0.07692308 0.68785956 0.17021277 0.06029128 0.02995669 0.48050576
  0.12515439 0.27117886 0.51987145]
 [0.64615385 0.46168101 0.21276596 0.06443207 0.03554968 0.23886517
  0.16479877 0.04430473 0.10894187]
 [0.24615385 0.6168875  0.4822695  0.70432641 0.51905619 0.31089695
  0.12381754 0.18183585 0.35847699]]
```

شکل ۶

در ادامه ما داده های train و test را به کمک کتابخانه sklearn و ماژول model-selection جدا میکنم. ۳۰ درصد داده ها برای تست انتخاب شده اند و ۷۰ درصد دیگر داده ها برای آموزش مدل انتخاب شده اند و در ادامه برای اطمینان بیشتر ۱۰ رکورد دیتاست train را چاپ میکنیم. این موارد در شکل ۶ مشخص شده اند.

در ادامه به سراغ ساختن مدل های یادگیری ماشین و اعمال آن ها به داده ها میرویم.

ساختار کد برای مدل Logistic Regression توضیح داده می شود و بقیه مدل ها به همین ترتیب روی داده ها اعمال شده اند و دقت ها گزارش شده اند.

در ابتدا به کمک ماژول linear_model در کتابخانه sklearn مدل LogisticRegression خود را میسازیم و به کمک داده های train، با کمک تابع fit مدل را رو این داده آموزش می دهیم. و در ادامه دقت مدل را روی داده های train می سنجیم که در این مورد دقت داده های train برابر است با: 0.7283950617283951

در ادامه نیز با دادن داده های تست به مدل آموزش دیده، مقادیر پیش بینی شده توسط مدل را نیز چاپ میکنیم و همینطور به کمک ماژول metrics در sklearn مقادیر confusion matrix وclassification report را علاوه بر مدل رگرسیون، برای تمامی مدل ها نمایش میدهیم.

مقدار دقت یا همان accuracy که تابع score به ما میدهد را نیز میتوان از روی confusion matrix برای هر مدل بدست آورد کافی است عناصر قطر اصلی یعنی مقادیر TP و TN را با یکدیگر جمع کنیم و تقسیم بر کل مقادیر ماتریس کنیم با این روش accuracy بدست می آید.

همچنین در classification report مقادیر پارامتر های دیگر برای سنجش مدل ها یادگیری ماشین آورده شده اند مانند recall و f1-score و...

در نهایت نیز به کمک داده های تست از داده های test مقدار accuracy می گیریم که برابر است با:
0.6285714285714286

تمامی توضیحات بالا در شکل ۷ و شکل ۸ آورده شده اند.

Logistic Regression on Cancer data set:

```
In [428]: from sklearn.linear_model import LogisticRegression
          from sklearn import metrics

          logisticRegression_model = LogisticRegression()
          logisticRegression_model.fit(X_train, y_train_bin)

Out[428]: LogisticRegression()

In [429]: logistic_train_score = logisticRegression_model.score(X_train, y_train_bin)
          print('logistic train score: ', logistic_train_score)

          logistic train score:  0.7283950617283951

In [430]: #Predict the response for test dataset
          y_pred = logisticRegression_model.predict(X_test)    # predict using test data.
          y_pred

Out[430]: array([1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0], dtype=int64)
```

شکل ۷

```
In [431]: from sklearn.metrics import confusion_matrix, classification_report

          cm = confusion_matrix(y_test_bin, y_pred)
          report = classification_report(y_test_bin, y_pred)

          print ("Confusion Matrix for test : \n", cm)
          print("\nclassification for logistic regression : \n", report)

          Confusion Matrix for test :
           [[ 6 10]
           [ 3 16]]

          classification for logistic regression :
                        precision    recall  f1-score   support

                     0       0.67      0.38      0.48        16
                     1       0.62      0.84      0.71        19

              accuracy                           0.63        35
             macro avg       0.64      0.61      0.60        35
          weighted avg       0.64      0.63      0.61        35

In [432]: logistic_test_score = logisticRegression_model.score(X_test, y_test_bin)   # Accuracy of the test-set.
          print('logistic test score: ', logistic_test_score)

          logistic test score:  0.6285714285714286
```

شکل ۸

در ادامه با انجام همین فرآیند برای هر کدام از الگوریتم های دیگر به این مقادیر برای accuracy بدست آمد:

- **SVM with linear kernel:**

Train accuracy: 0.7037037037037037
Test accuracy: 0.6571428571428571

- **SVM with RBF kernel:**

Train accuracy: 0.8641975308641975
Test accuracy: 0.8571428571428571

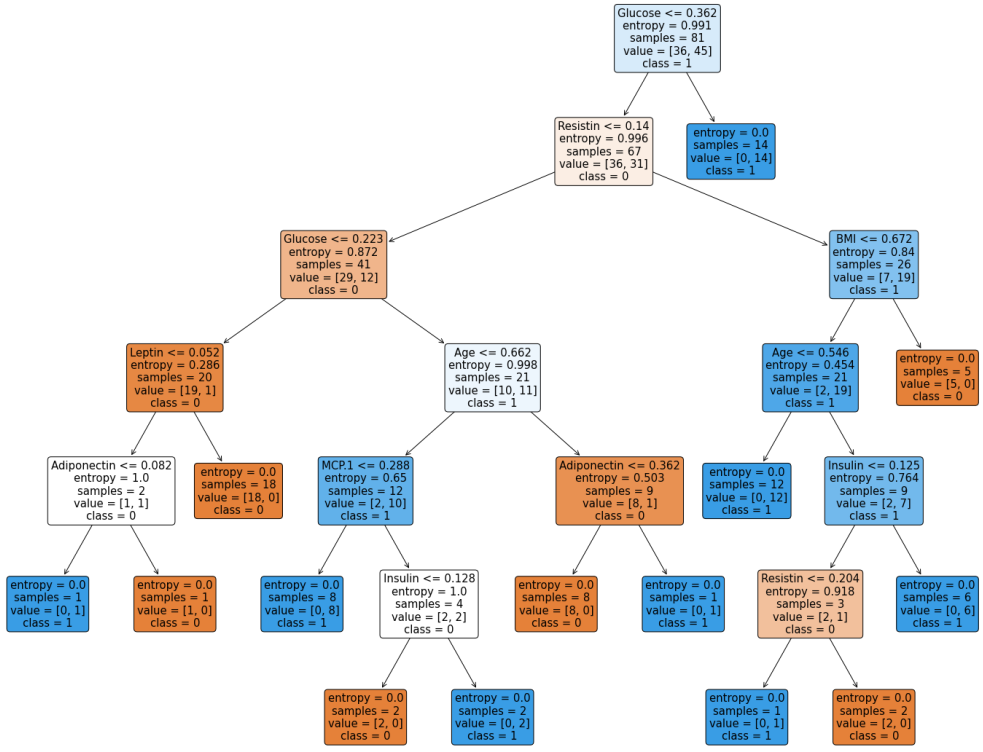
- **Decision Tree:**

Train accuracy: 1.0

Test accuracy: 0.7428571428571429

همچنین به کمک matplotlib درخت تصمیم مدل ساخته شده به صورت گرافیکی نیز در زیر نمایش داده شده است.

دقت شود که شاخص تصمیم در هر گره درخت تصمیم، آنتروپی داده ها در نظر گرفته شده است.



- **KNN:**

برای مدل KNN چالشی که وجود دارد انتخاب یک k مناسب برای الگوریتم می باشد که برای این کار ابتدا به صورت تستی و با k های مختلف مدل را آموزش می دهیم و داده تست را روی آن ها امتحان میکنیم و سپس accuracy های هر k را در یک دیکشنری که key های آن مقدار k و value های آن مقدار score در آن k مورد نظر برای مدل میباشد.

در نهایت عناصر دیکشنری را بر اساس score مرتب میکنیم و key بزرگترین score مورد نظر را برمیگردانیم که همان k بهینه است. نمودار های آن را نیز به ازای k های مختلف و داده های train و test برای درک بهتر نمایش داده شده اند.

این موارد در شکل ۹ نمایش داده شده است.

شکل ۹

با توجه به نمودار بهترین k برای این مدل 11 در نظر گرفته شده است.

در ادامه برای دقت train و test مدل **knn** خواهیم داشت:

Train accuracy: 0.7037037037037037
Test accuracy: 0.7428571428571429

در ادامه به سراغ ارزیابی واقعی و دقیق تر مدل های گفته شده میرویم.

K-fold Cross validation:

ما در این روش دیتاست اولیه را به k پارت تقسیم میکنیم، k-1 بخش آن به داده های train اخصاص داده می شوند و یک بخش آن به داده های test تخصیص داده می شوند. و ما باید score یا همان دقت مدل مورد نظر را در هر fold یا iteration مشخص از اجرای الگوریتم را در یک لیست نگه داریم تا برای سنجش کارایی مدل از آن لیست میانگین بگیریم. این روش معیار خوبی برای مقایسه کارایی مدل هایی است که از آن ها می خواهیم استفاده کنیم. ما در ادامه به همین موضوع پرداخته ایم.

ما برای پیاده سازی k-fold cross validation از ماژول model selection در کتابخانه sklearn استفاده میکنیم.

مقدار k_split را طبق صورت پروژه 5 در نظر گرفته ایم و یک instance از kFold می سازیم.

به کمک تابع split در KFold می توانیم در هر iteration فولد های مشخص را برای داده ها train و test مشخص کنیم و با داده های train یا همان fold آموزشی مدل های گفته شده را train کنیم.

و در ادامه در هر iteration الگوریتم، ما accuracy مدل ها را در لیست های مخصوص به خودشان که در یک دیکشنری می باشند ذخیره میکنیم. ساختار این دیکشنری به این شکل است که key های هر عنصر همان اسم مدل است و value آن عنصر یک لیستی مخصوص به نگه داری score مدل train شده در iteration، k ام الگوریتم می باشد.

این موارد در شکل ۱۰ مشخص شده اند.

در ادامه شکل ۱۱ پس از اتمام فرایند این الگوریتم برای درک بهتر مقدار این دیکشنری را چاپ میکنیم تا score های متفاوت برای مدل ها را مشاهده کنیم.

```
In [453]: from sklearn.model_selection import KFold

# this variable is fixed according to the project Description.
k_split = 5

CV = KFold(n_splits = k_split, shuffle = False)
dict_scores = {'logisticRegression_model':[], 'SVM_linearKernel_model':[],
               'SVM_RBFKernel_model':[], 'DecisionTree_model':[], 'knn_model':[]}

for train_index, test_index in CV.split(X):
    print("Train Index: ", train_index, "\n")
    print("Test Index: ", test_index)

    X_train, X_test, y_train, y_test = X.iloc[train_index], X.iloc[test_index], y[train_index], y[test_index]  # fold partioton.

    logisticRegression_model.fit(X_train, y_train)
    SVM_linearKernel_model.fit(X_train, y_train)
    SVM_RBFKernel_model.fit(X_train, y_train)
    DecisionTree_model.fit(X_train, y_train)
    knn_model.fit(X_train, y_train)

    dict_scores['logisticRegression_model'].append(logisticRegression_model.score(X_test, y_test))
    dict_scores['SVM_linearKernel_model'].append(SVM_linearKernel_model.score(X_test, y_test))
    dict_scores['SVM_RBFKernel_model'].append(SVM_RBFKernel_model.score(X_test, y_test))
    dict_scores['DecisionTree_model'].append(DecisionTree_model.score(X_test, y_test))
    dict_scores['knn_model'].append(knn_model.score(X_test, y_test))

for key, value in dict_scores.items():
    print("\n")
    print(key, value)
```

شکل ۱۰

```
logisticRegression_model [0.5833333333333334, 0.5217391304347826, 0.6956521739130435, 0.5652173913043478, 0.6521739130434783]

SVM_linearKernel_model [0.5833333333333334, 0.5652173913043478, 0.6956521739130435, 0.5652173913043478, 0.6956521739130435]

SVM_RBFKernel_model [0.625, 0.8695652173913043, 0.782608695652174, 0.7391304347826086, 0.6956521739130435]

DecisionTree_model [0.7916666666666666, 0.8260869565217391, 0.782608695652174, 0.7391304347826086, 0.5217391304347826]

knn_model [0.5833333333333334, 0.6956521739130435, 0.6956521739130435, 0.6956521739130435, 0.6521739130434783]
```

شکل ۱۱

در ادامه شکل ۱۲ نیز برای هر مدل میانگین score هایش را در دیکشنری بدست می آوریم و نشان میدهیم.

Comparing models using average:

```
In [454]: avg_dict = {'logisticRegression_model': 0, 'SVM_linearKernel_model': 0,
                      'SVM_RBFKernel_model': 0, 'DecisionTree_model': 0, 'knn_model': 0}

          for key in dict_scores.keys():
              avg_dict[key] = np.mean(dict_scores[key])

          # printing the best model according to their k-fold cross validations mean.
          avg_dict = {k: v for k, v in sorted(avg_dict.items(), key=lambda item: item[1])}
          avg_dict

Out[454]: {'logisticRegression_model': 0.6036231884057971,
           'SVM_linearKernel_model': 0.6210144927536232,
           'knn_model': 0.6644927536231885,
           'DecisionTree_model': 0.7322463768115942,
           'SVM_RBFKernel_model': 0.7423913043478261}
```

شکل ۱۲

همانطور که مشاهده می شود نتایج به ترتیب مرتب شده اند و بهترین مدل از لحاظ عملکرد SVM با کرنل RBF می باشد که دقت حدودا 0.742 برای آن بدست آمده است.

و به ترتیب مدل درخت تصمیم، knn، SVM با کرنل خطی و Logistic Regression با دقت های مشخص شده قابل ملاحظه می باشند.

ما می توانستیم کوتاه تر به این نتایج برسیم و به کمک ماژول KFold و cross_val_score در پایتون بدون ساختن fold ها برای داده های test و train ، یک ارزیابی از مدل ها داشته باشیم این کار نیز در این پروژه انجام گرفته است که جزیئات در شکل ۱۳ آمده است.

```
In [455]: # using cross_val_score from sklearn to test the pervious results.

from sklearn.model_selection import cross_val_score

CV2 = KFold(n_splits = k_split, shuffle=False)

result = cross_val_score(logisticRegression_model , X, y, cv = CV2)
print("Avg score for logisticRegression_model : {}".format(result.mean()))

result = cross_val_score(SVM_linearKernel_model , X, y, cv = CV2)
print("Avg score for SVM_linearKernel_model : {}".format(result.mean()))

result = cross_val_score(SVM_RBFKernel_model , X, y, cv = CV2)
print("Avg score for SVM_RBFKernel_model : {}".format(result.mean()))

result = cross_val_score(DecisionTree_model , X, y, cv = CV2)
print("Avg score for DecisionTree_model : {}".format(result.mean()))

result = cross_val_score(knn_model , X, y, cv = CV2)
print("Avg score for knn_model : {}".format(result.mean()))
```

```
Avg score for logisticRegression_model : 0.6036231884057971
Avg score for SVM_linearKernel_model : 0.6210144927536232
Avg score for SVM_RBFKernel_model : 0.7423913043478261
Avg score for DecisionTree_model : 0.7583333333333334
Avg score for knn_model : 0.6644927536231885
```

شکل ۱۳

- ✓ **تمامی توضیحات داده شده در رابطه با مدل های یادگیری ماشین در notebook با اسم ML_Models.ipynb قرار دارند. این notebook در پوشه Models قرار دارد.**