تحلیل مجموعه داده قلب[1]

محدثه اخوان‌فرد

اطلاعات گزارش

تاریخ: ۱۶ بهمن‌ماه ۱۴۰۰

واژگان کلیدی:
بیماری قلبی
خوشه‌بندی
طبقه‌بندی

چکیده

مجموعه داده قلب شامل ویژگی‌هایی است که می‌توانند بر بیماری قلبی اثر بگذارند. در این گزارش به بررسی تاثیر ویژگی‌های این مجموعه داده بر بیماری قلبی و استفاده از الگوریتم‌های خوشه‌بندی و طبقه‌بندی می‌پردازیم.

۱- مقدمه

مجموعه داده قلب از ۱۲ویژگی تشکیل شده است که برای هریک از آن‌ها نام ویژگی، نوع، بازه مقادیر، همچنین خصوصیات آماری آن‌ها شامل مقدار میانگین، میانه، مد، کمینه و بیشینه در جدول زیر مشخص شده است:

نام ویژگی	نوع	بازه مقادیر	میانگین	میانه	مد	کمینه	بیشینه
Age	Numeric (ratio)	28-77	53.51089	54	54	77	28
Sex	Categorical (Binary)	-	-	-	M	-	-
ChestPainType	Categorical (Nominal)	-	-	-	ASY	-	-
RestingBP	Numeric	0-200	132.39651	130	120	200	0
Cholesterol	Numeric	0-603	198.79956	223	0	603	0
FastingBS	Categorical (Binary)	-	-	-	0	-	-
RestingECG	Categorical (Nominal)	-	-	-	Normal	-	-
MaxHR	Numeric	60-202	136.80936	138	150	202	60
ExerciseAngina	Categorical (Binary)	-	-	-	N	-	-
Oldpeak	Numeric	-2.6-6.2	0.887364	0.6	0.0	6.2	-2.6
ST_Slope	Categorical (Nominal)	-	-	-	Flat	-	-
HeartDisease	Categorical (Binary)	-	-	-	1	-	-

جدول ۱-۱- مشخصات ویژگی‌ها

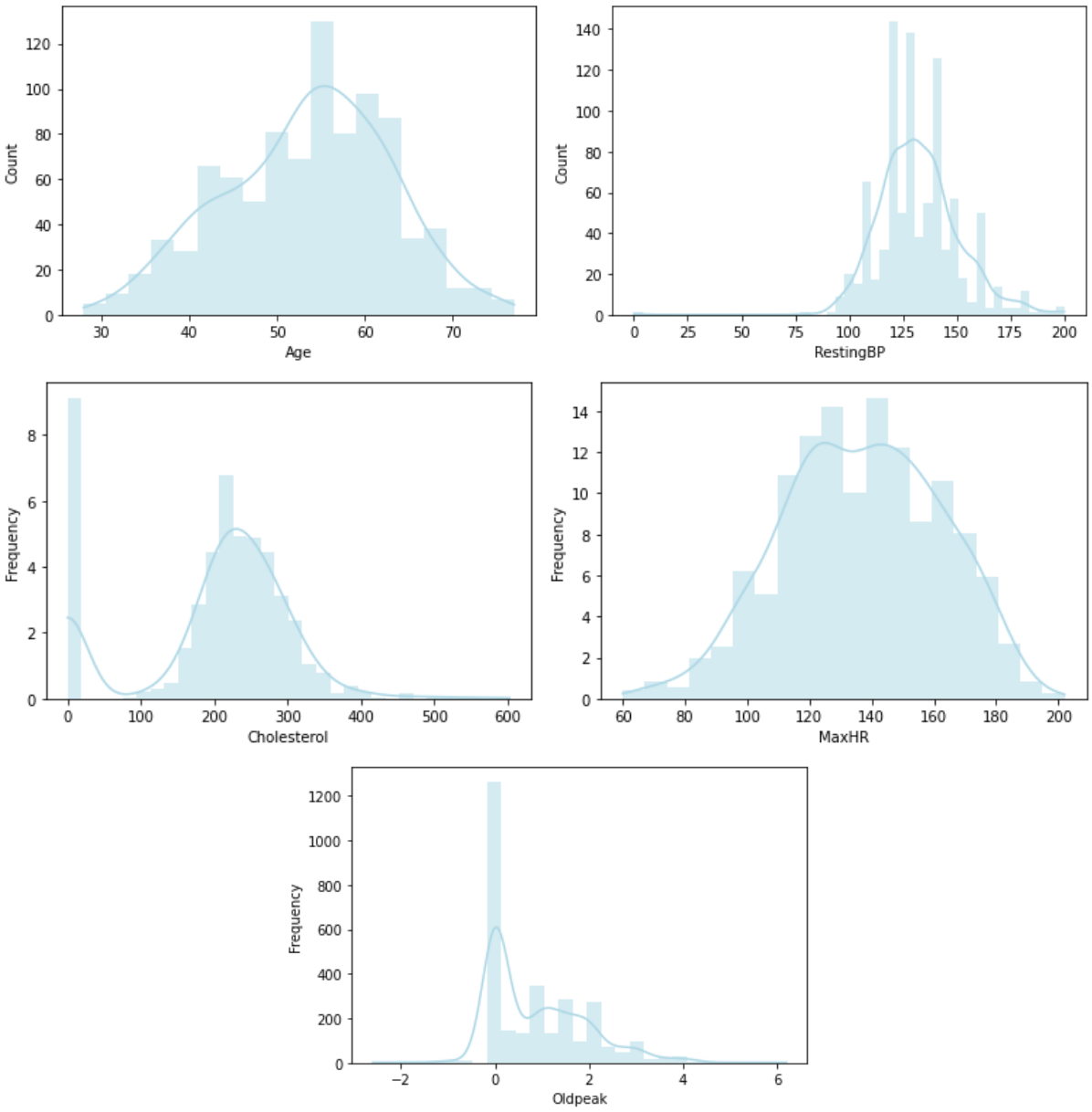
[1] Heart Dataset

این ویژگی‌ها به ترتیبی که در جدول آمده است، عبارتند از: سن، جنسیت، نوع درد قفسه سینه، فشار خون، کلسترول، قند خون، فعالیت الکتریکی قلب، حداکثر ضربان قلب، آنژین قلبی ناشی از ورزش، میزان افسردگی، شیب تغییرات ضربان قلب حین ورزش و داشتن بیماری قلبی.

۲- مراحل داده‌کاوی

۲-۱- شناخت داده

در ادامه از آنجایی که چولگی برای ویژگی های کیفی معنایی ندارد، برای داده های کمّی خواهیم داشت:



شکل ۲-۱- چولگی برای داده‌های کمّی

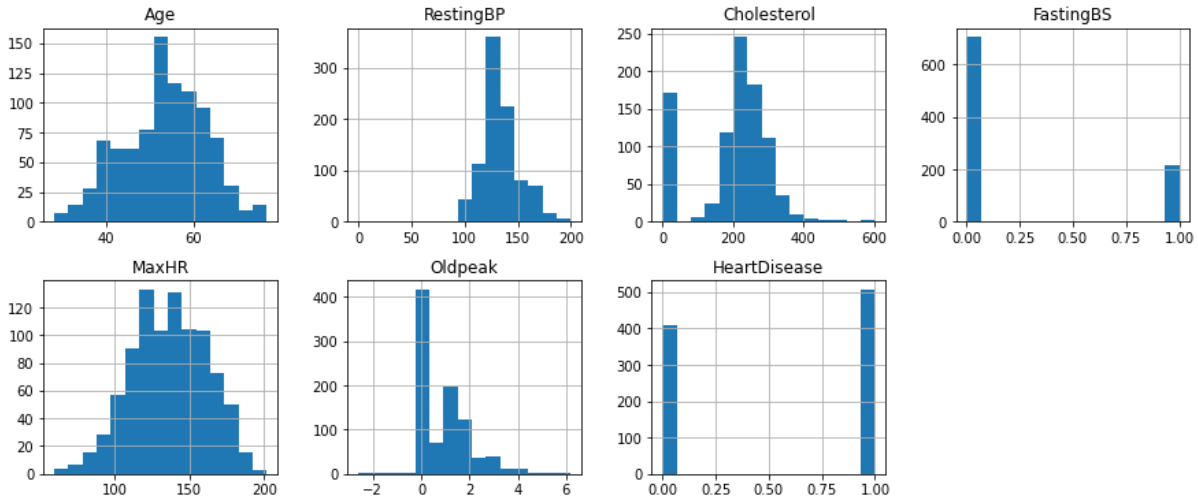
صحت و معتبربودن هر یک از ویژگی‌ها با استفاده از اطلاعاتی که در اختیار داده شده است [۱] در جدول زیر مشخص گردیده است. کامل بودن نیز با استفاده از دستور زیر مشخص گردیده و نشان می‌دهد که هیچ‌یک از ویژگی‌ها دارای مقدار گمشده نیستند.

```
[9]  data.isnull().sum()

     Age              0
     Sex              0
     ChestPainType    0
     RestingBP        0
     Cholesterol      0
     FastingBS        0
     RestingECG       0
     MaxHR            0
     ExerciseAngina   0
     Oldpeak          0
     ST_Slope         0
     HeartDisease     0
     dtype: int64
```

نام ویژگی	صحت	اعتبار	کامل بودن
Age	✔	✔	✔
Sex	✔	✔	✔
ChestPainType	✔	✔	✔
RestingBP	✔	✔	✔
Cholesterol	✔	✔	✔
FastingBS	✔	✔	✔
RestingECG	✔	✔	✔
MaxHR	✔	✔	✔
ExerciseAngina	✔	✔	✔
Oldpeak	✔	✔	✔
ST_Slope	✔	✔	✔
HeartDisease	✔	✔	✔

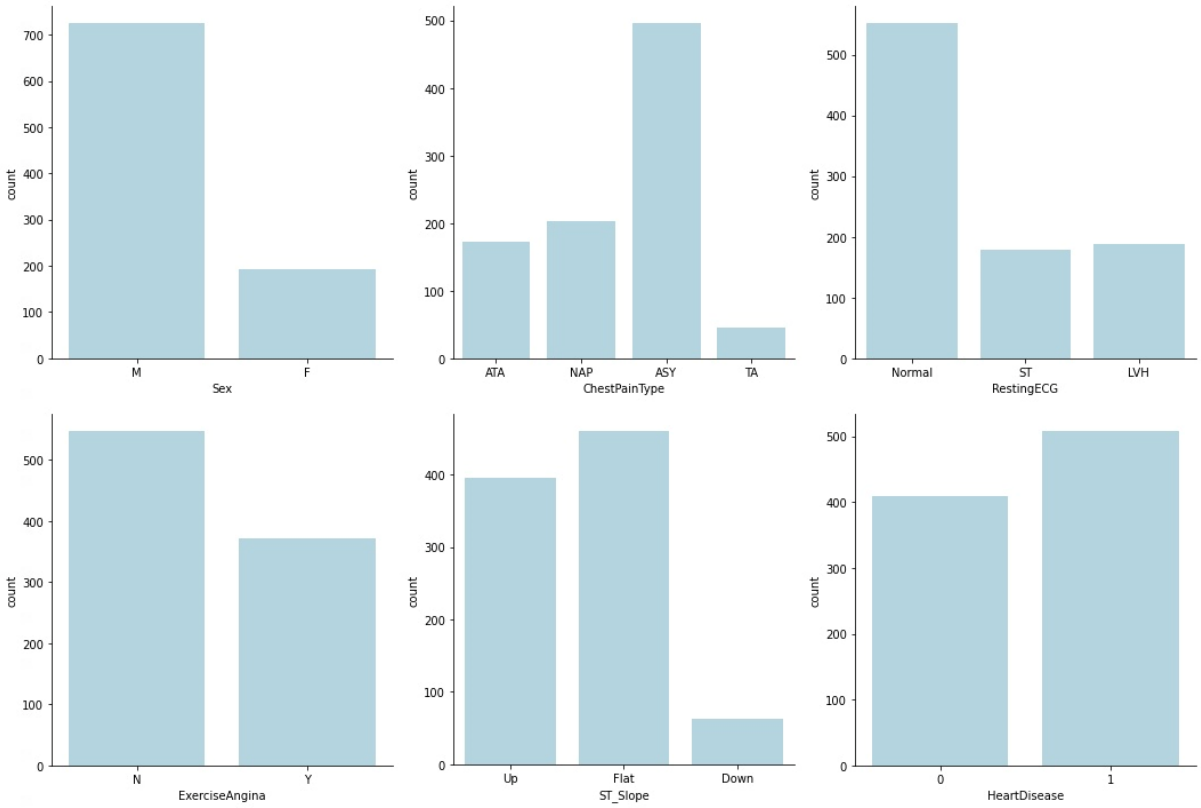
جدول ۲-۱- بررسی صحت، اعتبار و کامل بودن ویژگی‌ها

نمودارهای هیستوگرام برای هر یک از ویژگی‌های کمّی در شکل ۱-۲ مشخص شده است. (باید توجه داشت از آن‌جایی که ویژگی های FastingBS و HeartDisease دارای مقادیر ۰ و ۱ هستند، تابع استفاده شده برای رسم ویژگی‌های کمّی، این ویژگی‌ها را نیز کمّی در نظر گرفته است و نمودار هیستوگرام را برای آن‌ها رسم کرده است.)



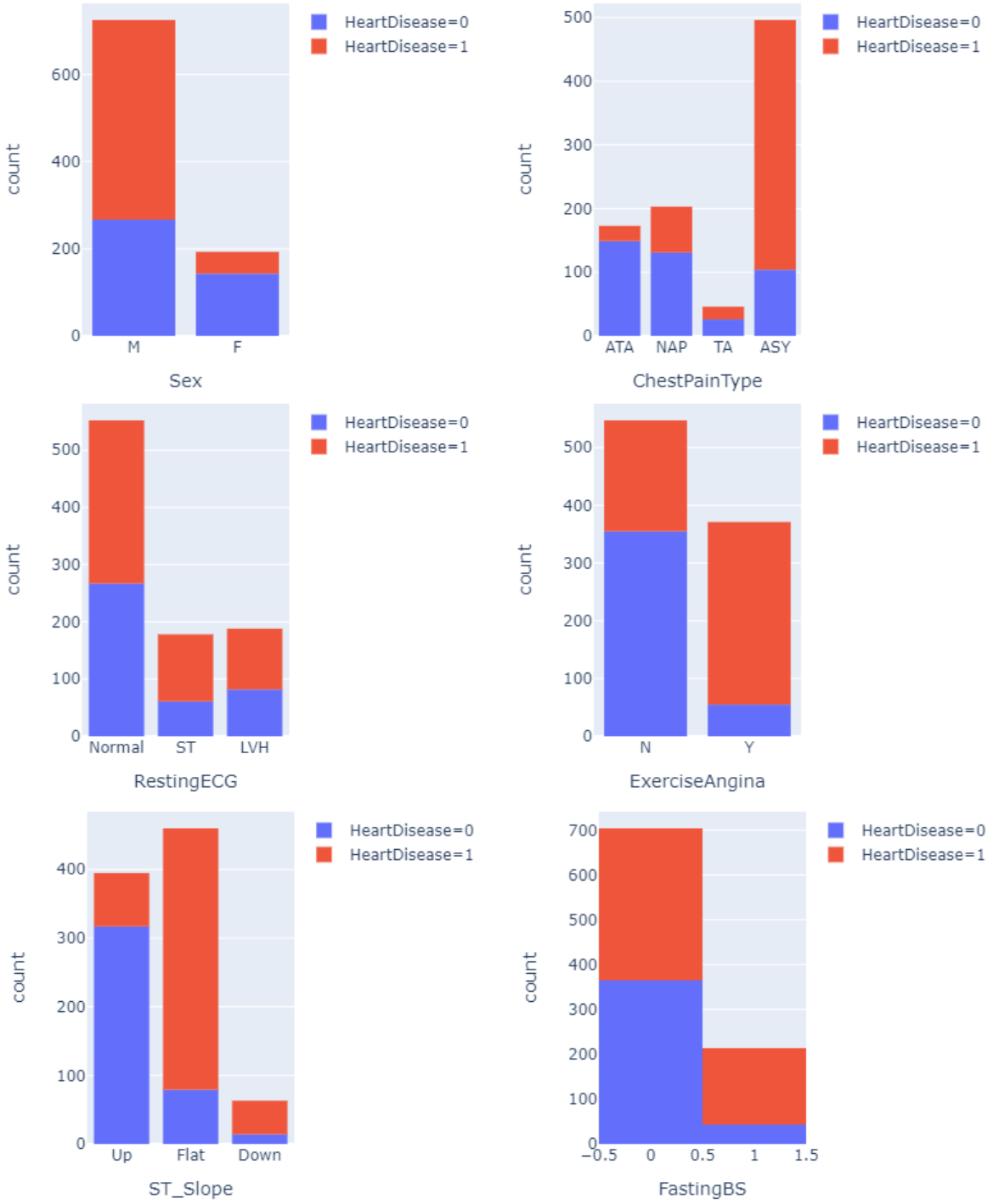
شکل ۲-۲- نمودار هیستوگرام برای ویژگی‌های کمّی

و برای داده‌های کیفی خواهیم داشت:



شکل ۲-۳- نمودار هیستوگرام برای ویژگی‌های کیفی

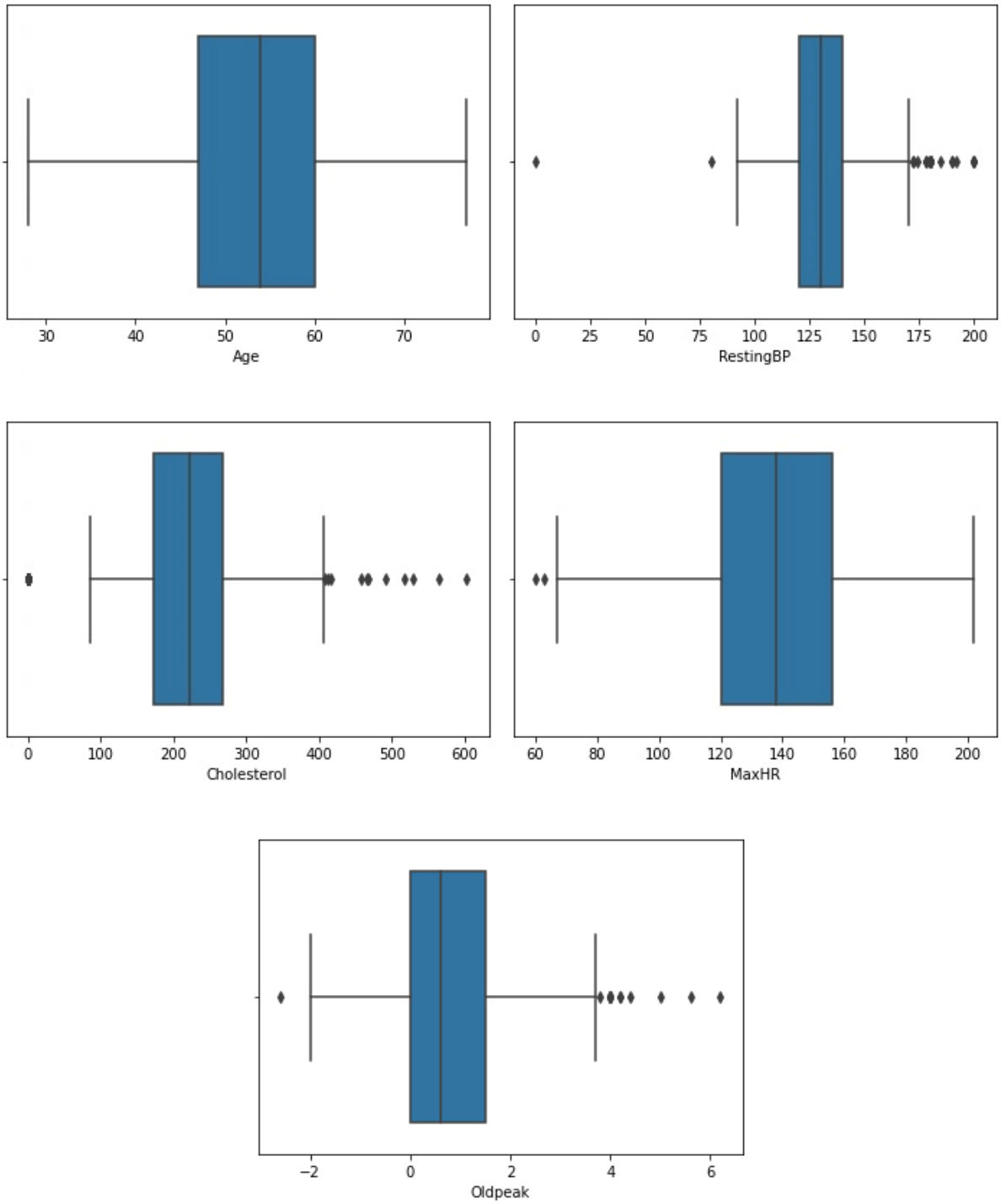
هم چنین برای مقایسه اینکه هر یک از ویژگی‌های کیفی چه میزان در بیماری قلبی (ویژگی HeartDisease) تاثیر دارند، می‌توانیم نمودارهای هیستوگرام زیر را رسم کنیم.



شکل ۲-۴- نمودارهای هیستوگرام ویژگی‌های کیفی و ویژگی HeartDisease

مشاهده می‌کنیم که بیماری قلبی در مردان شایع‌تر بوده و در بیماری قلبی، نوع درد ASY قفسه سینه بیش‌تر می‌باشد.

برای تشخیص داده‌های پرت هر یک از ویژگی‌ها با استفاده از boxplot خواهیم داشت:



شکل ۲-۵- نمودار boxplot

داده‌های پرت هر یک از ویژگی‌ها در جدول ۲-۲ مشخص شده است؛

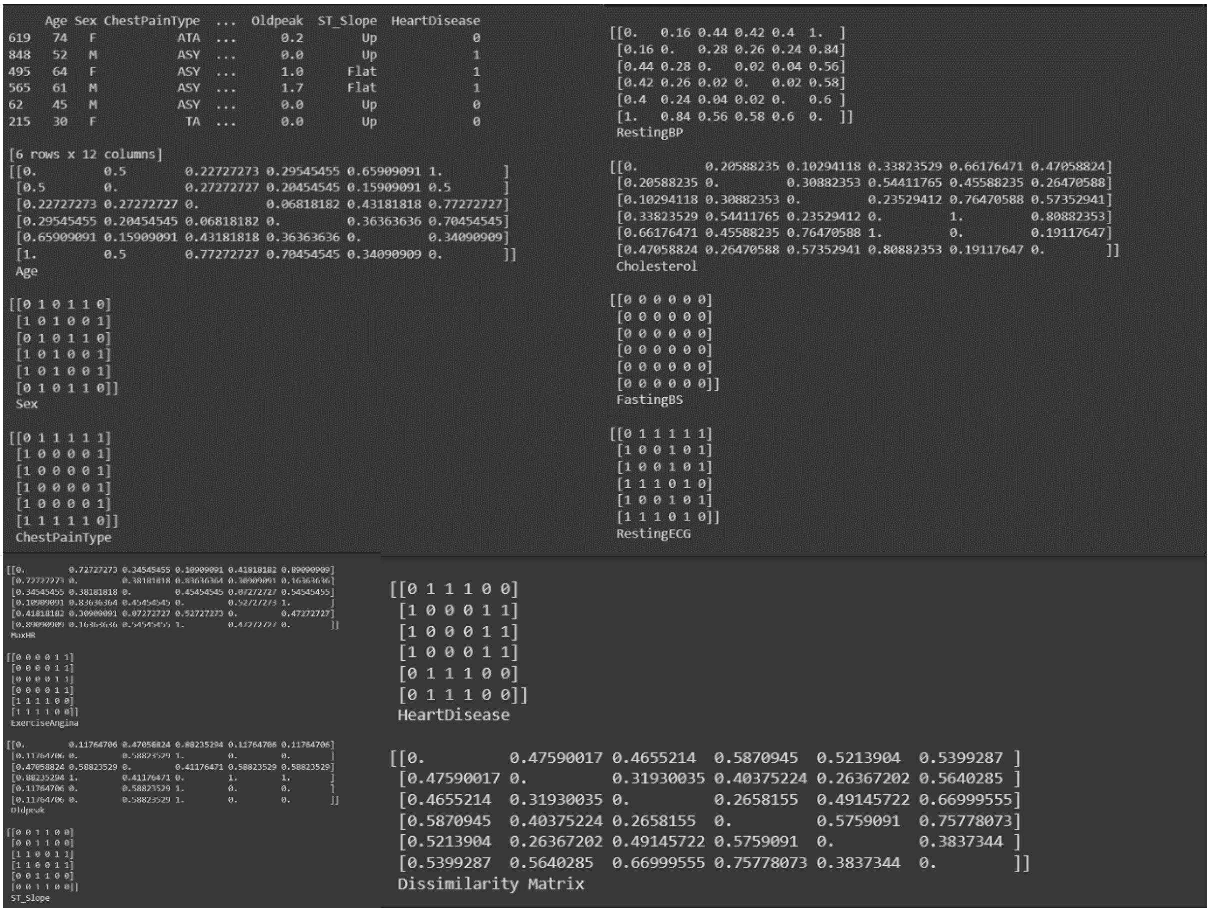
نام ویژگی	داده‌های پرت
Age	ندارد.
RestingBP	170, 190, 180, 170, 180, 180, 170, 170, 200, 180, 180, 170, 180, 80, 200, 185, 170, 200, 180, 180, 170, 0, 178, 170, 170, 170, 172, 180, 190, 170, 174, 178, 180, 200, 192, 178, 180, 170, 180, 172, 170, 170

Cholesterol	468, 518, 412, 529, 466, 603, 491, 0, 458, 0, 0, 0, 0, 0, 0, 564, 417, 409
MaxHR	63, 60
Oldpeak	4.0, 5.0, -2.6, 4.0, 4.0, 4.0, 4.0, 4.0, 4.2, 4.0, 5.6, 3.8, 4.2, 6.2, 4.4, 4.0

جدول ۱-۳- داده‌های پرت

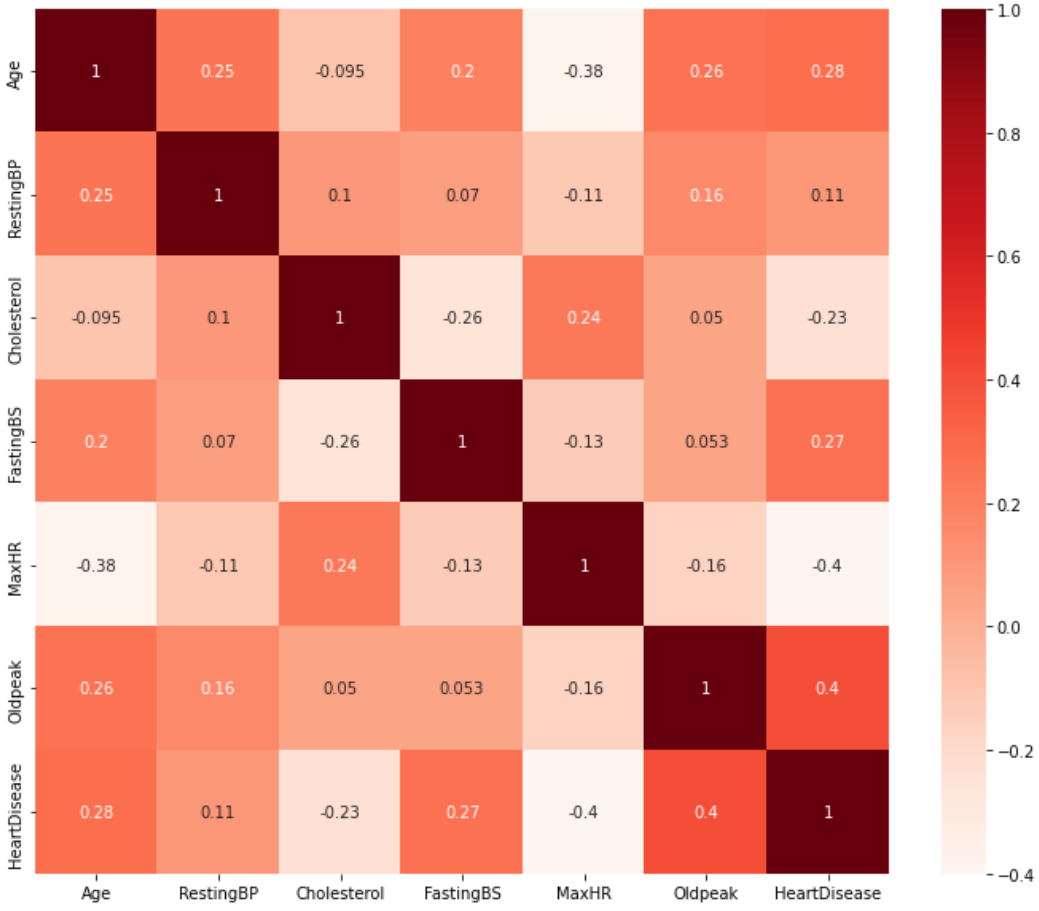
برای اینکه زیرمجموعه‌ای از داده‌ها را بدون غرض و به صورت اتفاقی انتخاب کنیم از sample استفاده می کنیم؛ که n تعداد رکوردهایی است که می‌خواهیم به صورت تصادفی انتخاب شوند با استفاده از تابع dissimilarity ماتریس عدم شباهت را برای هر یک از ویژگی ها، با توجه به کمّی یا کیفی بودن، به‌دست می‌آوریم و در آخر برای به دست آوردن ماتریس عدم شباهت از gower_matrix که میانگین را برای هر یک از خانه های ماتریس محاسبه می کند، استفاده می کنیم. (وزن در نظر گرفته شده برای هر یک از ویژگی ها برابر ۱ است.)

در اینجا برای اینکه بتوانیم ماتریس را به طور کامل نشان دهیم تعداد داده‌های تصادفی را برابر با ۶ در نظر گرفته ایم.



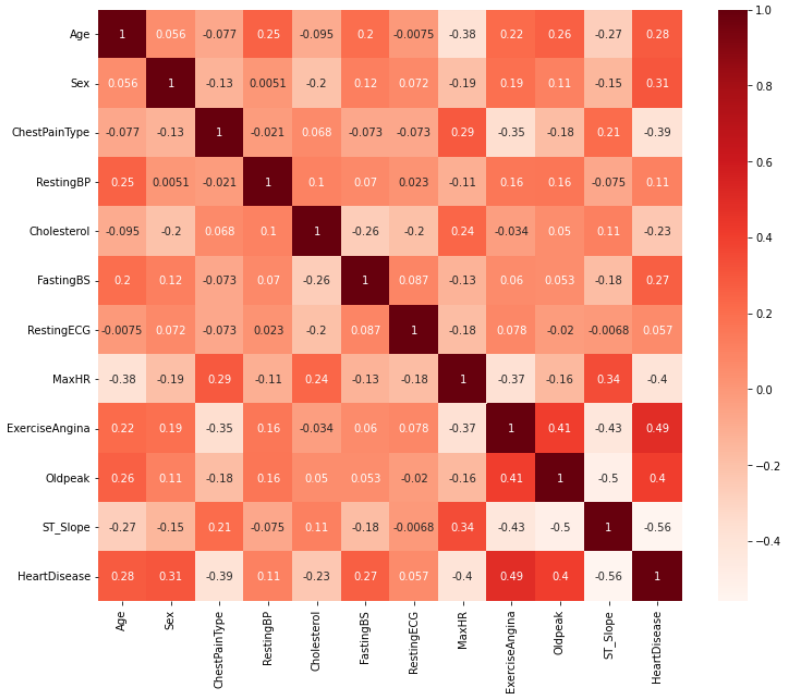
```
     Age Sex ChestPainType  ...  Oldpeak  ST_Slope  HeartDisease
619   74   F           ATA  ...      0.2        Up             0
848   52   M           ASY  ...      0.0        Up             1
495   64   F           ASY  ...      1.0      Flat             1
565   61   M           ASY  ...      1.7      Flat             1
62    45   M           ASY  ...      0.0        Up             0
215   30   F            TA  ...      0.0        Up             0

[6 rows x 12 columns]
[[0.         0.5        0.22727273 0.29545455 0.65909091 1.        ]
 [0.5        0.         0.27272727 0.20454545 0.15909091 0.5       ]
 [0.22727273 0.27272727 0.         0.06818182 0.43181818 0.77272727]
 [0.29545455 0.20454545 0.06818182 0.         0.36363636 0.70454545]
 [0.65909091 0.15909091 0.43181818 0.36363636 0.         0.34090909]
 [1.         0.5        0.77272727 0.70454545 0.34090909 0.        ]]
 Age

[[0 1 0 1 1 0]
 [1 0 1 0 0 1]
 [0 1 0 1 1 0]
 [1 0 1 0 0 1]
 [1 0 1 0 0 1]
 [0 1 0 1 1 0]]
 Sex

[[0 1 1 1 1 1]
 [1 0 0 0 0 1]
 [1 0 0 0 0 1]
 [1 0 0 0 0 1]
 [1 0 0 0 0 1]
 [1 1 1 1 1 0]]
 ChestPainType

[[0.   0.16 0.44 0.42 0.4  1.  ]
 [0.16 0.   0.28 0.26 0.24 0.84]
 [0.44 0.28 0.   0.02 0.04 0.56]
 [0.42 0.26 0.02 0.   0.02 0.58]
 [0.4  0.24 0.04 0.02 0.   0.6 ]
 [1.   0.84 0.56 0.58 0.6  0.  ]]
 RestingBP

[[0.         0.20588235 0.10294118 0.33823529 0.66176471 0.47058824]
 [0.20588235 0.         0.30882353 0.54411765 0.45588235 0.26470588]
 [0.10294118 0.30882353 0.         0.23529412 0.76470588 0.57352941]
 [0.33823529 0.54411765 0.23529412 0.         1.         0.80882353]
 [0.66176471 0.45588235 0.76470588 1.         0.         0.19117647]
 [0.47058824 0.26470588 0.57352941 0.80882353 0.19117647 0.        ]]
 Cholesterol

[[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 0 0 0 0]]
 FastingBS

[[0 1 1 1 1 1]
 [1 0 0 1 0 1]
 [1 0 0 1 0 1]
 [1 1 1 0 1 0]
 [1 0 0 1 0 1]
 [1 1 1 0 1 0]]
 RestingECG

[[0.         0.72727273 0.34545455 0.10909091 0.41818182 0.89090909]
 [0.72727273 0.         0.38181818 0.83636364 0.30909091 0.16363636]
 [0.34545455 0.38181818 0.         0.45454545 0.07272727 0.54545455]
 [0.10909091 0.83636364 0.45454545 0.         0.52727273 1.        ]
 [0.41818182 0.30909091 0.07272727 0.52727273 0.         0.47272727]
 [0.89090909 0.16363636 0.54545455 1.         0.47272727 0.        ]]
 MaxHR

[[0 0 0 0 1 1]
 [0 0 0 0 1 1]
 [0 0 0 0 1 1]
 [0 0 0 0 1 1]
 [1 1 1 1 0 0]
 [1 1 1 1 0 0]]
 ExerciseAngina

[[0.         0.11764706 0.47058824 0.88235294 0.11764706 0.11764706]
 [0.11764706 0.         0.58823529 1.         0.         0.        ]
 [0.47058824 0.58823529 0.         0.41176471 0.58823529 0.58823529]
 [0.88235294 1.         0.41176471 0.         1.         1.        ]
 [0.11764706 0.         0.58823529 1.         0.         0.        ]
 [0.11764706 0.         0.58823529 1.         0.         0.        ]]
 Oldpeak

[[0 0 1 1 0 0]
 [0 0 1 1 0 0]
 [1 1 0 0 1 1]
 [1 1 0 0 1 1]
 [0 0 1 1 0 0]
 [0 0 1 1 0 0]]
 ST_Slope

[[0 1 1 1 0 0]
 [1 0 0 0 1 1]
 [1 0 0 0 1 1]
 [1 0 0 0 1 1]
 [0 1 1 1 0 0]
 [0 1 1 1 0 0]]
 HeartDisease

[[0.         0.47590017 0.4655214  0.5870945  0.5213904  0.5399287 ]
 [0.47590017 0.         0.31930035 0.40375224 0.26367202 0.5640285 ]
 [0.4655214  0.31930035 0.         0.2658155  0.49145722 0.66999555]
 [0.5870945  0.40375224 0.2658155  0.         0.5759091  0.75778073]
 [0.5213904  0.26367202 0.49145722 0.5759091  0.         0.3837344 ]
 [0.5399287  0.5640285  0.66999555 0.75778073 0.3837344  0.        ]]
 Dissimilarity Matrix
```

با استفاده از heatmap میزان همبستگی داده ها را مورد بررسی قرار می‌دهیم.



شکل ۲-۶- بررسی همبستگی ویژگی‌های کمّی

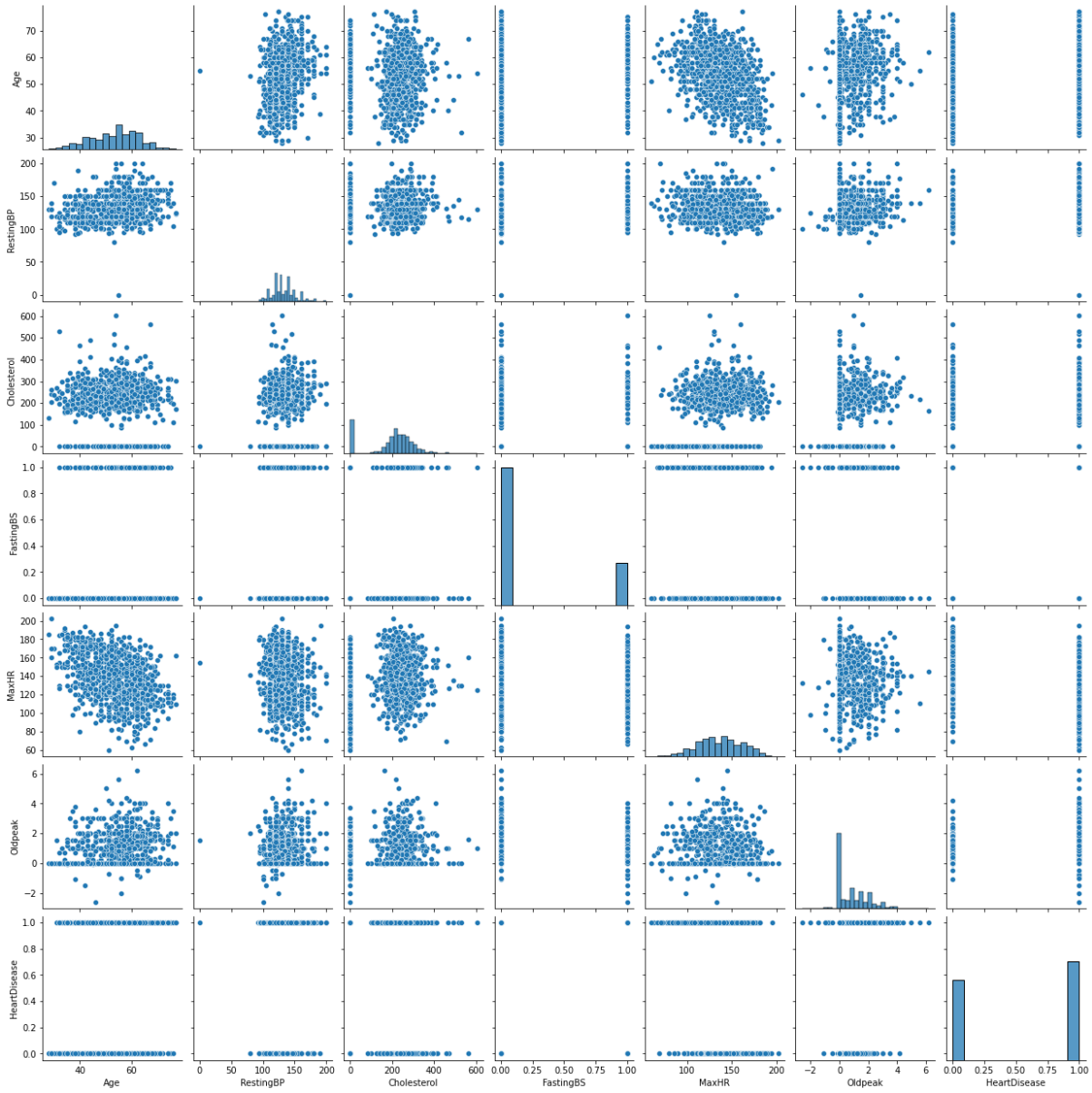
همچنین با در نظر گرفتن ویژگی‌های کیفی و تبدیل آن‌ها به ویژگی کمّی، نمودار همبستگی زیرا خواهیم داشت:



شکل ۲-۷- بررسی همبستگی تمامی ویژگی‌ها

همانطور که مشاهده می‌شود ExerciseAngina با HeartDisease همبستگی مثبت و نزدیک به ۰/۵ دارد که نشان می‌دهد در صورت کاهش(افزایش) یکی از آن‌ها دیگری نیز کاهش(افزایش) می‌یابد یا به عبارتی با یکدیگر همبستگی مثبت قوی دارند. هم‌چنین ویژگی‌هایی مانند Cholesterol و ChestPainType با HeartDisease همبستگی منفی ضعیف دارند.

نمودار scatter برای ویژگی‌های کمّی به همراه دو ویژگی کیفی و باینری FastingBS و HeartDisease مشخص شده است.



شکل ۲-۸- نمودار scatter برای ویژگی‌های کمّی

با توجه به نمودار به‌دست آمده می‌توان گفت ویژگی‌های MaxHR و Age همبستگی‌ای نسبتا منفی و قوی دارند.

۲-۱- پیش‌پردازش [۲]

برای پاک سازی داده ها، مجموع داده های گم شده برای هریک از ویژگی‌ها با استفاده از data.isnull().sum مشخص شده و با استفاده از data.dropna می توان این داده‌ها را حذف کرد. (این مجموعه‌داده هیچ داده گم‌شده‌ای ندارد.)

داده‌های تکراری با استفاده از data.duplicated مشخص می‌شوند (به صورت True/False) و در صورتی که مقدار برگردانده شده True باشد ، این داده ها با استفاده از drop_duplicates حذف خواهند شد. (این مجموعه‌داده هیچ داده تکراری ندارد.)

با استفاده از تابع remove_outliers می توان داده‌های پرت را برای هر یک از ویژگی‌های کمّی حذف کرد. در صورتی‌که مجموع داده‌های پرت یک ویژگی از ۱۵۰تا کمتر باشد، داده پرت را با میانگین داده ها جای‌گذاری می کند، در غیر اینصورت آن را حذف می کند.

حال اطلاعاتی مانند میانگین، انحراف معیار، کمینه و بیشینه و چارک های اول، دوم، سوم و چهارم را با استفاده از temp.describe().T در جدول مشاهده خواهیم کرد.

```
print(data.isnull().sum()) #Checking for the missing values
data.dropna()

data.duplicated() #To find duplicate records
data.drop_duplicates(subset=list(data.columns), inplace=True) #remove duplicate records


#remove outliers
temp = data.copy()

def remove_outliers(att):
  Q1=temp[att].quantile(0.25)
  Q3=temp[att].quantile(0.75)
  IQR=Q3-Q1
  temp[att] = np.where((temp[att] < (Q1-1.5*IQR)) | (temp[att] > (Q3+1.5*IQR)), np.nan, temp[att])
  if temp[att].isnull().sum() < 150:
    temp[att].fillna(temp[att].median(), inplace=True)
  else:
    temp[att] = temp[att].dropna(axis = 0)
  return temp

numerics = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']

for col in numerics:
  remove_outliers(col)
temp.describe().T
```

```
Age              0
Sex              0
ChestPainType    0
RestingBP        0
Cholesterol      0
FastingBS        0
RestingECG       0
MaxHR            0
ExerciseAngina   0
Oldpeak          0
ST_Slope         0
HeartDisease     0
dtype: int64
```

	count	mean	std	min	25%	50%	75%	max
Age	918.0	53.510893	9.432617	28.0	47.0	54.0	60.0	77.0
RestingBP	918.0	131.079521	15.597206	92.0	120.0	130.0	140.0	170.0
Cholesterol	735.0	241.038095	51.164730	85.0	207.0	236.0	274.0	407.0
FastingBS	918.0	0.233115	0.423046	0.0	0.0	0.0	0.0	1.0
MaxHR	918.0	136.976035	25.215656	67.0	120.0	138.0	156.0	202.0
Oldpeak	918.0	0.827669	0.958516	-2.0	0.0	0.5	1.5	3.7
HeartDisease	918.0	0.553377	0.497414	0.0	0.0	1.0	1.0	1.0

از آنجایی که برای دیتاست اجتماع سازی[1] صورت نگرفت و هیچ یک از ویژگی ها از ویژگی دیگری مشتق نشده است، افزونگی در سطح رکورد یا ویژگی وجود ندارد که آن را برطرف کنیم.

در صورتی که تعداد رکوردها زیاد باشد برای تحلیل سریعتر داده ها عملیات کاهش را انجام می‌دهیم و چون ما در فازهای بعد نیازی به کاهش داده نداشتیم و با مشکلی مواجه نشدیم در این جا برای نمونه، ۲۵ درصد از رکوردها را به صورت تصادفی انتخاب کردیم.

```
subset = data.sample(frac =.25, replace = False) #get .25 % of the rows
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
762	40	M	ASY	110	167	0	LVH	114	Y	2.0	Flat	1
98	56	M	ASY	120	85	0	Normal	140	N	0.0	Up	0
102	40	F	ASY	150	392	0	Normal	130	N	2.0	Flat	1
464	59	M	NAP	131	0	0	Normal	128	Y	2.0	Down	1
905	67	M	NAP	152	212	0	LVH	150	N	0.8	Flat	1
...	...	...	...	...	...	...	...	...	...	...	...	...
871	61	M	NAP	150	243	1	Normal	137	Y	1.0	Flat	0
220	46	M	ASY	130	222	0	Normal	112	N	0.0	Flat	1
442	51	M	ASY	128	0	1	ST	125	Y	1.2	Flat	1
466	55	M	NAP	120	0	0	ST	125	Y	2.5	Flat	1
795	42	M	NAP	120	240	1	Normal	194	N	0.8	Down	0

230 rows × 12 columns

برای انجام عملیات نرمال سازی ابتدا ویژگی های کیفی را به کمّی تبدیل کرده و سپس از روش min_max استفاده کردیم.

[1] Integration

```
# categorical convert to numeric
cp_data = data.copy()

cat_columns = cp_data.select_dtypes(['object']).columns
cp_data[cat_columns] = cp_data[cat_columns].apply(lambda x: pd.factorize(x)[0])
cp_data.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	0	0	140	289	0	0	172	0	0.0	0	0
1	49	1	1	160	180	0	0	156	0	1.0	1	1
2	37	0	0	130	283	0	1	98	0	0.0	0	0
3	48	1	2	138	214	0	0	108	1	1.5	1	1
4	54	0	1	150	195	0	0	122	0	0.0	0	0

```
def normalize(df,column_name):
    df[column_name] = (df[column_name] - df[column_name].min()) / (df[column_name].max() - df[column_name].min())

dataset = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak','Sex','ChestPainType','RestingECG', 'ExerciseAngina', 'ST_Slope']

for i in range(len(dataset)):
  column_name = dataset[i]
  normalize(cp_data,column_name)

display(cp_data)
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	0.244898	0.0	0.000000	0.70	0.479270	0	0.0	0.788732	0.0	0.295455	0.0	0
1	0.428571	1.0	0.333333	0.80	0.298507	0	0.0	0.676056	0.0	0.409091	0.5	1
2	0.183673	0.0	0.000000	0.65	0.469320	0	0.5	0.267606	0.0	0.295455	0.0	0
3	0.408163	1.0	0.666667	0.69	0.354892	0	0.0	0.338028	1.0	0.465909	0.5	1
4	0.530612	0.0	0.333333	0.75	0.323383	0	0.0	0.436620	0.0	0.295455	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
913	0.346939	0.0	1.000000	0.55	0.437811	0	0.0	0.507042	0.0	0.431818	0.5	1
914	0.816327	0.0	0.666667	0.72	0.320066	1	0.0	0.570423	0.0	0.681818	0.5	1
915	0.591837	0.0	0.666667	0.65	0.217247	0	0.0	0.387324	1.0	0.431818	0.5	1
916	0.591837	1.0	0.000000	0.65	0.391376	0	1.0	0.802817	0.0	0.295455	0.5	1

۲-۳- داده‌کاوی

۲-۳-۱- الگوریتم‌های خوشه‌بندی

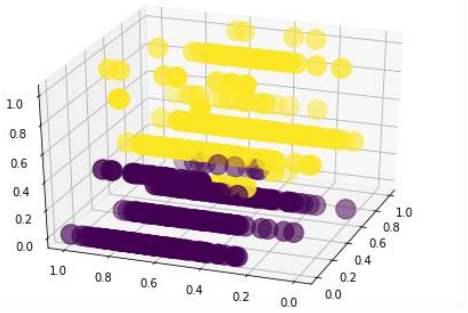
الگوریتم K-Means [۳]

یکی از روش های مورد استفاده در خوشه بندی روش K-means است که برای انجام آن باید تعداد خوشه ها را مشخص کنیم و در این دیتاست ما دو خوشه داریم. برای رسم نمودار نتیجه این روش در دیتاست، ما از سه ویژگی که بیشترین همبستگی را با هم داشتند یعنی ST_Slope و MaxHR و ChestPainType استفاده کردیم.

```python
from sklearn.metrics import confusion_matrix,classification_report
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

X = cp_data[['ChestPainType', 'MaxHR','ST_Slope']].values

model = KMeans(n_clusters=2)
model.fit_predict(X)
pred = model.fit_predict(X)
labels = model.labels_

print(confusion_matrix(cp_data["HeartDisease"],model.labels_))
print(classification_report(cp_data["HeartDisease"],model.labels_,zero_division=1))

fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(X[:,0], X[:,1], X[:,2], c=model.labels_, s=300)
ax.view_init(azim=200)
plt.show()

print("number of labels: ", set(labels))
print("number of cluster found: {}".format(len(set(labels))))
n_noise_ = list(labels).count(-1)
```

و نتیجه به دست آمده در شکل ۲-۳-۱ قابل مشاهده است که دو خوشه تشخیص داده میشود و نویزی تشخیص نمیدهد.

```
[[329  81]
 [ 88 420]]
              precision    recall  f1-score   support

           0       0.79      0.80      0.80       410
           1       0.84      0.83      0.83       508

    accuracy                           0.82       918
   macro avg       0.81      0.81      0.81       918
weighted avg       0.82      0.82      0.82       918
```



شکل ۲-۳-۱- نمودرا scatter برای K-means

و اگر بخواهیم این روش را روی تمام ویژگی های دیتاست اجرا کنیم داریم:

```
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix,classification_report
X =  cp_data.drop(["HeartDisease"],axis = 1)

kmeans = KMeans(n_clusters=2).fit(X)
y_predicted = kmeans.predict(X)

print(confusion_matrix(data["HeartDisease"],kmeans.labels_))
print(classification_report(data["HeartDisease"],kmeans.labels_))
```

```
[[355  55]
 [192 316]]
              precision    recall  f1-score   support

           0       0.65      0.87      0.74       410
           1       0.85      0.62      0.72       508

    accuracy                           0.73       918
   macro avg       0.75      0.74      0.73       918
weighted avg       0.76      0.73      0.73       918
```

و می‌بینیم میزان دقت ۷۳ درصد بدست می‌آید و نسبت به استفاده از سه ویژگی، کاهش دقت داریم.

الگوریتم DBScan [۴]

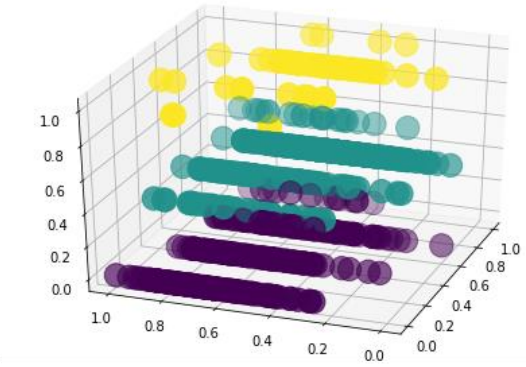
از روش دیگری به نام DBScan نیز برای خوشه بندی استفاده میکنیم که باید پارامتر eps برای حداکثر شعاع همسایگی و min_samples برای مشخص کردن حداقل تعداد داده در هر خوشه را طوری مقداردهی کنیم که بیشترین دقت را بدست آوریم. برای رسم نمودار نتیجه این روش در دیتاست، ما از سه ویژگی که بیشترین همبستگی را با هم داشتند یعنی ST_Slope و MaxHR و ChestPainType استفاده کردیم.

از NearestNeighbors برای یافتن مقدار eps بهینه استفاده کردیم و پارمتر n_neighbors را با ۲۰ مقدار دهی کردیم تا فاصله را با ۲۰ همسایه نزدیک محاسبه و سپس محدوده کوچکتری را برای eps بهینه بدست آوریم (در شکل ۲-۳-۳) که در این جا میبینیم مقدار ۰.۱ تا ۰.۵ بهترین مقدار برای eps و ۴۰ تا ۶۰ برای min_samples (با آزمایش) بدست می‌آید. سپس با استفاده از معیار silhouette_score بهترین eps و min_samples را از بین محدوده بدست آمده مشخص میکنیم به این صورت که بهترین min_samples و eps، بیشترین مقدار silhouette_score را نتیجه میدهد مشاهده میکنیم در eps=0. 3 و min_samples=43 بیشترین مقدار silhouette_score که برابر است با ۴۶ درصد بدست می‌آید. در نتیجه زمانی که تابع DBScan را با پارمتر های بدست آمده اجرا میکنیم می‌بینیم ۳ خوشه تشخیص میدهد با دقت ۷۶ درصد که نتیجه درستی هم روی دیتاست ما ندارد چون ما درحقیقت دو کلاس داریم.

در نتیجه روش K-means نتیجه بهتری روی دیتاست ما دارد.

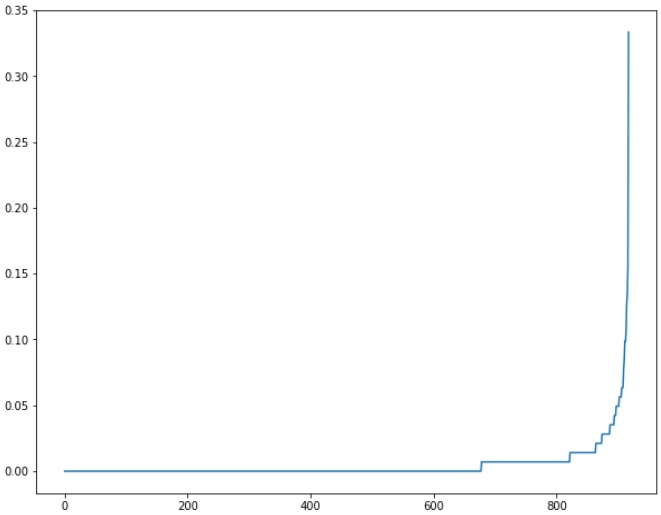
```
from sklearn.cluster import DBSCAN

X = cp_data[['ChestPainType', 'MaxHR','ST_Slope']].values

nn = NearestNeighbors(n_neighbors=20).fit(X)
distances, indices = nn.kneighbors(X)
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.figure(figsize=(10,8))
plt.plot(distances)

min_samples = range(40,60)
eps = np.arange(0.1,0.5, 0.05)
output = []

for ms in min_samples:
    for ep in eps:
        labels = DBSCAN(min_samples=ms, eps = ep).fit(X).labels_
        score = silhouette_score(X, labels)
        output.append((ms, ep, score))

min_samples, eps, score = sorted(output, key=lambda x:x[-1])[-1]
print(f"Best silhouette_score: {score}")
print(f"min_samples: {min_samples}")
print(f"eps: {eps}")
```

```
model = DBSCAN(eps=0.3, min_samples=43)
model.fit_predict(X)
pred = model.fit_predict(X)
labels = model.labels_

print(confusion_matrix(cp_data["HeartDisease"],model.labels_))
print(classification_report(cp_data["HeartDisease"],model.labels_,zero_division=1))

fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(X[:,0], X[:,1], X[:,2], c=model.labels_, s=300)
ax.view_init(azim=200)
plt.show()

print("number of labels: ", set(labels))
print("number of cluster found: {}".format(len(set(labels))))
n_noise_ = list(labels).count(-1)
print('number of noise(s): ', n_noise_)
```



```
Best silhouette_score: 0.4663385464259479
min_samples: 43
eps: 0.30000000000000004
[[317  79  14]
 [ 78 381  49]
 [  0   0   0]]
              precision    recall  f1-score   support

           0       0.80      0.77      0.79       410
           1       0.83      0.75      0.79       508
           2       0.00      1.00      0.00         0

    accuracy                           0.76       918
   macro avg       0.54      0.84      0.52       918
weighted avg       0.82      0.76      0.79       918
```

شکل ۲-۳-۲- نمودرا scatter برای DBScan



شکل ۲-۳-۳- نمودار محدوده نقاط بهینه eps

و اگر بخواهیم این روش را روی تمام ویژگی های دیتاست اجرا کنیم داریم:

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import confusion_matrix,classification_report

X =  cp_data.drop(["HeartDisease"],axis = 1)
model = DBSCAN(eps=0.45,min_samples=47)
model.fit_predict(X)
yhat = model.fit_predict(X)

print(confusion_matrix(data["HeartDisease"],model.labels_))
print(classification_report(data["HeartDisease"],model.labels_,zero_division=1))
```

```
[[  0   0   0]
 [277 125   8]
 [403  12  93]]
              precision    recall  f1-score   support

          -1       0.00      1.00      0.00         0
           0       0.91      0.30      0.46       410
           1       0.92      0.18      0.31       508

    accuracy                           0.24       918
   macro avg       0.61      0.50      0.25       918
weighted avg       0.92      0.24      0.37       918
```

که مشاهده میکنیم دقت بسیار پایین می‌آید و روشی کارآمد نیست.

۲-۳-۲- الگوریتم‌های طبقه‌بندی

الگوریتم MLP [۵]

ابتدا می‌خواهیم بصورت تصادفی تعدادی از پارامترهای تابع MLPClassifier را انتخاب و آن‌ها را مقداردهی کنیم. در صورتی که شبکه عصبی را سه‌لایه در نظر بگیریم، به‌شکلی که در لایه مخفی اول ۳۰۰، لایه مخفی دوم ۲۰۰ و لایه مخفی سوم نیز ۲۰۰ نورون داشته باشیم، با قرار دادن warm_start = True و random_state=21 خواهیم دید که دقت به ۹۲ درصد خواهد رسید.

```
[[73 11]
 [ 4 96]]
              precision    recall  f1-score   support

           0       0.95      0.87      0.91        84
           1       0.90      0.96      0.93       100

    accuracy                           0.92       184
   macro avg       0.92      0.91      0.92       184
weighted avg       0.92      0.92      0.92       184
```

در صورتی که به پارامترهای بالا، activation = ‘logistic’ را اضافه کنیم، به دقت ۹۹ درصد می‌رسیم که در این حالت، احتمال بیش‌برازش وجود داشته و مناسب نیست.

```
[[ 70   2]
 [  0 112]]
              precision    recall  f1-score   support

           0       1.00      0.97      0.99        72
           1       0.98      1.00      0.99       112

    accuracy                           0.99       184
   macro avg       0.99      0.99      0.99       184
weighted avg       0.99      0.99      0.99       184
```

در واقع، مدل‌ها می توانند فراپارامتر[1]های زیادی داشته باشند و یافتن بهترین ترکیب از پارامترها می‌تواند به عنوان یک مشکل جستجو در نظر گرفته شود.
پس برای اینکه بدانیم کدام یک از پارامترهای داده شده دقت بهتری به ما می‌دهد، از GridSearchCV استفاده می‌کنیم.
با انتخاب پارامترهای زیر و انتخاب cv=10 برای GridSearchCV، از میان آن‌ها مناسب‌ترین پارامترها خواهیم داشت.[۶]

[1] hyperparameter

```
#Split the data set into training data and test data
X = data.drop('HeartDisease', axis=1)
y = data['HeartDisease']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 10)

parameters={
  'learning_rate': ["constant", "invscaling", "adaptive"],
  'hidden_layer_sizes': [(300, 200, 100), (200,200,200),
                         (300, 300, 300), (200, 100, 100)],
  'alpha': [1, 0.1, 0.01, 0.001, 0.0001],
  'activation': ["logistic", "relu", "tanh"]
}

clf = GridSearchCV(MLPClassifier(), parameters, verbose=2, n_jobs=-1, cv=10)

# fitting the model for grid search
clf.fit(X_train, y_train)

# print best parameter after tuning
print(clf.best_params_)
clf_predictions = clf.predict(X_test)

# print classification report
print(classification_report(y_test, clf_predictions))
```

```
Fitting 10 folds for each of 180 candidates, totalling 1800 fits
{'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (200, 100, 100), 'learning_rate': 'constant'}
              precision    recall  f1-score   support

           0       0.79      0.87      0.83        86
           1       0.88      0.80      0.83        98

    accuracy                           0.83       184
   macro avg       0.83      0.83      0.83       184
weighted avg       0.84      0.83      0.83       184
```

می‌بینیم که به دقت بالایی نرسیده‌ایم، در صورتی که بصورت تصادفی تعدادی از داده‌ها (۵۰۰تا) را انتخاب کنیم، دقت اندکی بالاتر خواهد رفت و برخی پارامترها نیز تغییر خواهند کرد.

```
Fitting 10 folds for each of 180 candidates, totalling 1800 fits
{'activation': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': (300, 300, 300), 'learning_rate': 'constant'}
              precision    recall  f1-score   support

           0       0.89      0.84      0.86        49
           1       0.85      0.90      0.88        51

    accuracy                           0.87       100
   macro avg       0.87      0.87      0.87       100
weighted avg       0.87      0.87      0.87       100
```

الگوریتم SVM [۷]

یکی از پارامترهای تابع SVC، C می‌باشد که پارامتر منظم‌سازی قدرت است.
با انتخاب فراپارامترهای مختلف و مشخص کردن دقت تا چهاررقم اعشار خواهیم دید:

- SVC با پارامترهای پیش‌فرض (C=1.0, kernel=rbf and gamma=auto)

```
# Prepare training data for building the model
X = scaled_data
y = data['HeartDisease']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)

#Training the Algorithm
svclassifier = SVC()

## Train/Fit the model
svclassifier.fit(X_train, y_train)

#Making Predictions
y_pred = svclassifier.predict(X_test)

#Evaluating the Algorithm
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
[[58 15]
 [14 97]]
              precision    recall  f1-score   support

           0       0.81      0.79      0.80        73
           1       0.87      0.87      0.87       111

    accuracy                           0.84       184
   macro avg       0.84      0.83      0.83       184
weighted avg       0.84      0.84      0.84       184

Model accuracy score with default hyperparameters: 0.8424
```

- SVC با پارامترهای kernel=rbf and C=100.0

```
[[74 17]
 [ 8 85]]
              precision    recall  f1-score   support

           0       0.90      0.81      0.86        91
           1       0.83      0.91      0.87        93

    accuracy                           0.86       184
   macro avg       0.87      0.86      0.86       184
weighted avg       0.87      0.86      0.86       184

Model accuracy score with rbf kernel and C=100.0: 0.8641
```

- SVC با پارامترهای kernel=rbf and C=1000.0

```
[[68 22]
 [11 83]]
              precision    recall  f1-score   support

           0       0.86      0.76      0.80        90
           1       0.79      0.88      0.83        94

    accuracy                           0.82       184
   macro avg       0.83      0.82      0.82       184
weighted avg       0.82      0.82      0.82       184

Model accuracy score with rbf kernel and C=1000.0: 0.8207
```

همانطور که مشاهده می‌شود دقت با C=1000 کاهش یافته است.

- SVC با پارامترهای C=1.0, kernel=linear

```
[[65 14]
 [ 9 96]]
              precision    recall  f1-score   support

           0       0.88      0.82      0.85        79
           1       0.87      0.91      0.89       105

    accuracy                           0.88       184
   macro avg       0.88      0.87      0.87       184
weighted avg       0.88      0.88      0.87       184

Model accuracy score with kernel=linear, C=1.0: 0.8750
```

- SVC با پارامترهای C=100.0, kernel=linear

```
[[ 64   8]
 [  8 104]]
              precision    recall  f1-score   support

           0       0.89      0.89      0.89        72
           1       0.93      0.93      0.93       112

    accuracy                           0.91       184
   macro avg       0.91      0.91      0.91       184
weighted avg       0.91      0.91      0.91       184

Model accuracy score with linear kernel and C=100.0: 0.9130
```

- SVC با پارامترهای C=1000.0, kernel=linear

```
[[ 64   8]
 [  9 103]]
              precision    recall  f1-score   support

           0       0.88      0.89      0.88        72
           1       0.93      0.92      0.92       112

    accuracy                           0.91       184
   macro avg       0.90      0.90      0.90       184
weighted avg       0.91      0.91      0.91       184

Model accuracy score with linear kernel and C=1000.0: 0.9076
```

در اینجا هم مشاهده می‌کنیم دقت با C=1000 کاهش یافته است.

الگوریتم KNN [۸]

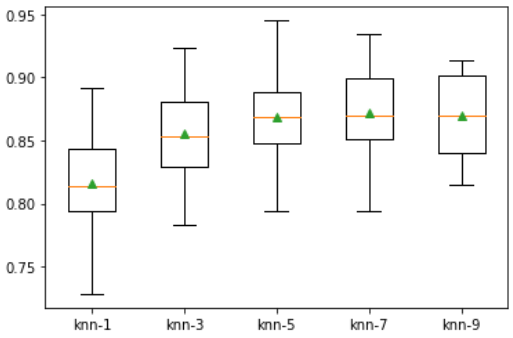
در اینجا به صورت دلخواه k=3 را به عنوان پارامتر KNeighborsClassifier در نظر می‌گیریم.

```
#Split the data set into training data and test data
# Prepare training data for building the model
X = scaled_data
y = data['HeartDisease']

x_training_data, x_test_data, y_training_data, y_test_data = train_test_split(X, y, test_size = 0.2)

#Train the model and make predictions
model = KNeighborsClassifier(n_neighbors = 3) #k=3
model.fit(x_training_data, y_training_data)
predictions = model.predict(x_test_data)

#Performance measurement
print(confusion_matrix(y_test_data, predictions))
print(classification_report(y_test_data, predictions))
print('Model accuracy score with k=3: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
[[71 13]
 [12 88]]
              precision    recall  f1-score   support

           0       0.86      0.85      0.85        84
           1       0.87      0.88      0.88       100

    accuracy                           0.86       184
   macro avg       0.86      0.86      0.86       184
weighted avg       0.86      0.86      0.86       184
```

حال می‌خواهیم مناسب‌ترین k را با مقایسه دقت‌های به‌دست آمده مقایسه کنیم.

```
Accuracy: 0.82 (+/- 0.04) [knn-1]
Accuracy: 0.86 (+/- 0.03) [knn-3]
Accuracy: 0.87 (+/- 0.03) [knn-5]
Accuracy: 0.87 (+/- 0.03) [knn-7]
Accuracy: 0.87 (+/- 0.03) [knn-9]
```



همانطور که مشاهده می‌شود، برای kهای ۵ به بعد دقت تغییری نمی‌کند و می‌توان گفت دقت تقریبا مناسبی داده می‌شود.

الگوریتم Bayes [۹]

در این الگوریتم ما از تابع GaussianNB() برای طبقه بندی استفاده کردیم و بعد با آزمایش کردن اندازه های مختلف برای مجموعه تست به این نتیجه رسیدیم که با مجموعه تست با اندازه ۰.۱ از دیتاست بیشترین دقت حاصل میشود.
همچنین زمانی که random_state را برابر با یک قرار میدهیم (به این معنا که با هر بار اجرا نتیجه تغییری نمیکند چون داده های مجموعه آموزش و آزمون در هر بار اجرا مانند اجرا قبلی خواهد بود) مشاهده میکنیم دقت افزایش داشته و در با هر بار اجرا دقت ۹۱ درصد بدست می‌آید

```
X =  cp_data.drop(["HeartDisease"],axis = 1)
y = data['HeartDisease']

# splitting X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1)

# training the model on training set
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# making predictions on the testing set
y_pred = gnb.predict(X_test)

# comparing actual response values (y_test) with predicted response values (y_pred)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

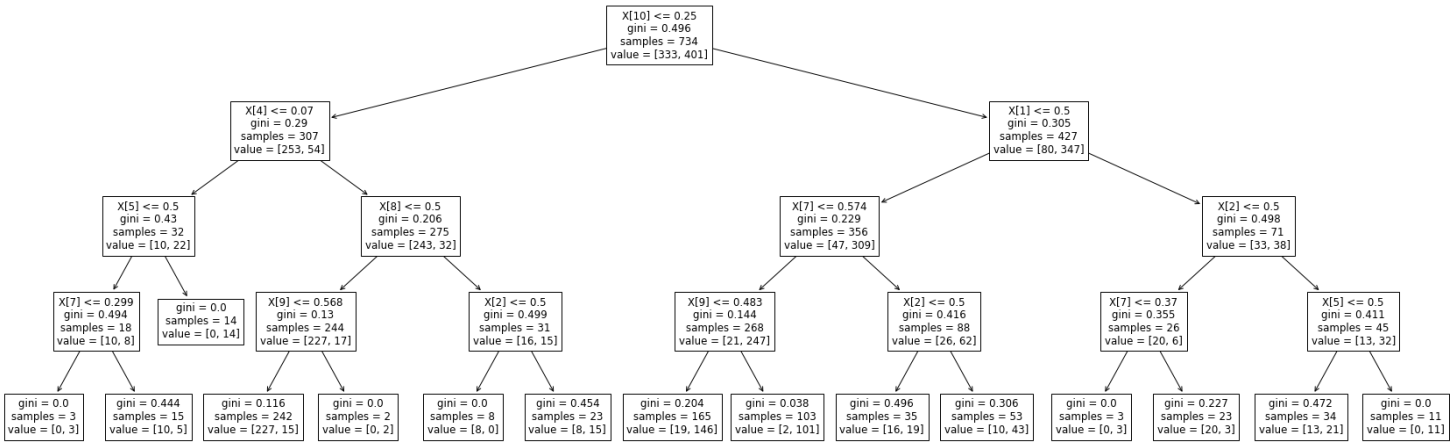
```
[[34  3]
 [ 5 50]]
              precision    recall  f1-score   support

           0       0.87      0.92      0.89        37
           1       0.94      0.91      0.93        55

    accuracy                           0.91        92
   macro avg       0.91      0.91      0.91        92
weighted avg       0.91      0.91      0.91        92
```

الگوریتم Decision Tree [۱۰]

در این طبقه بندی از DecisionTreeClassifier() استفاده میکنیم و حداکثر ارتفاع درخت را برابر با ۴ قرار دادیم چون با این اندازه، حداکثر دقت که برابر است با ۸۶ درصد بدست می‌آید و زمانی که هیچکدام از پارمتر های این تابع رامقدار دهی نکنیم دقت مجموعه آموزش ۱۰۰ درصد میشود و به عبارتی overfitting رخ میدهد.

```
X =  cp_data.drop(["HeartDisease"],axis = 1)
y = data['HeartDisease']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)

# instantiate the DecisionTreeClassifier model
dtree = DecisionTreeClassifier( max_depth=4)

# fit the model
dtree.fit(X_train, y_train)

# Predict the Test set result
y_pred = dtree.predict(X_test)

# print the scores on training and test set
print('Model accuracy score : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
print('Training set score: {:.4f}'.format(dtree.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(dtree.score(X_test, y_test)))

plt.figure(figsize=(30,10))
tree.plot_tree(dtree.fit(X_train, y_train))
```

```
Model accuracy score : 0.8641
Training set score: 0.8760
Test set score: 0.8641
```



شکل ۲-۳-۴- نمودار درخت تصمیم

همان طور که در شکل ۲-۳-۴ مشخص است، نتیجه اجرای این الگوریتم با تنظیم حداکثر ارتفاع ۴، درختی است که ریشه آن ویژگی دهم دیتاست یعنی ST_Slope است و اگر هر داده ای با ویژگی ST_Slope مساوی یا کمتر از ۰.۲۵ باشد در سمت چپ و در غیر این صورت در راست درخت قرار میگیرد و سپس توسط ویژگی های بعدی دوباره تقسیم بندی صورت میگیرد.

الگوریتم Ensemble [۱۱]

برای اینکه مجموعه داده آموزشی و آزمایشی را توسط چندین مدل آزمایش کرده و با استفاده از رای نتیجه را برای داده آزمایشی به‌دست آورد از دو شیوه رای‌دهی hard و soft استفاده می‌کنیم و برای ارزیابی نیز از StratifiedKFold با پارامترهای مشخص شده استفاده می‌کنیم، پس خواهیم داشت:

- Hard Voting

```
X = scaled_data
y = data['HeartDisease']

kf = StratifiedKFold(n_splits=10, shuffle=True)

clf1 = SVC(kernel='rbf')
clf2 = KNeighborsClassifier(n_neighbors = 5)
clf3 = MLPClassifier(hidden_layer_sizes=(300,200,200), activation='logistic',
                     random_state=21, warm_start=True)

eclf = VotingClassifier(estimators=[('svm', clf1), ('KNN', clf2), ('MLP', clf3)], voting='hard')

for clf, label in zip([clf1, clf2, clf3, eclf], ['svm', 'KNN', 'MLP', 'Ensemble']):
    scores = cross_val_score(clf, X, y, scoring='accuracy', cv=kf)
    print("Accuracy: %0.4f (+/- %0.4f) [%s]" % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.8671 (+/- 0.0346) [svm]
Accuracy: 0.8649 (+/- 0.0315) [KNN]
Accuracy: 0.8540 (+/- 0.0337) [MLP]
Accuracy: 0.8714 (+/- 0.0330) [Ensemble]
```

- Soft Voting

```
X = scaled_data
y = cp_data['HeartDisease']

kf = StratifiedKFold(n_splits=10, shuffle=True)

clf1 = SVC(kernel='rbf', probability=True)
clf2 = KNeighborsClassifier(n_neighbors = 5)
clf3 = MLPClassifier(hidden_layer_sizes=(300,200,200), activation='logistic',
                     random_state=21, warm_start=True)

eclf = VotingClassifier(estimators=[('svm', clf1), ('KNN', clf2), ('MLP', clf3)],voting='soft')

for clf, label in zip([clf1, clf2, clf3, eclf], ['svm', 'KNN', 'MLP', 'Ensemble']):
    scores = cross_val_score(clf, X, y, scoring='accuracy', cv=kf)
    print("Accuracy: %0.4f (+/- %0.4f) [%s]" % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.8627 (+/- 0.0347) [svm]
Accuracy: 0.8715 (+/- 0.0271) [KNN]
Accuracy: 0.8497 (+/- 0.0321) [MLP]
Accuracy: 0.8682 (+/- 0.0277) [Ensemble]
```

در نتیجه hard voting اندکی دقت بالاتری خواهد داشت.

هم‌چنین در صورتی‌که از تابع bagging به‌عنوان یک مدل از Ensemble استفاده کنیم و آن را با RepeatedStratifiedKFold ارزیابی کنیم، خواهیم داشت:

```python
# evaluate bagging algorithm for classification
from numpy import mean, std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import BaggingClassifier
# define dataset
X =  data.drop(["HeartDisease"],axis = 1)
y = data['HeartDisease']
# define the model
model = BaggingClassifier()
# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=4)
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
# report performance
print('Accuracy: %.4f (+/- %.4f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy: 0.8442 (+/- 0.0361)
```

۲-۴- الگوهای پرتکرار و قوانین انجمنی [۱۲]

برای مشخص کردن الگوهای پرتکرار ابتدا ویژگی‌های کیفی را از مجموعه‌داده انتخاب شده‌اند و سپس برای تفهیم نتایج مقادیر آن‌ها کدگذاری کرده و به‌صورت مجموعه‌ای از تراکنش‌ها نشان داده شده‌اند. با در نظرگرفتن اطمینان ۰/۴ تمامی الگوهای پرتکرار مشاهده می‌شوند.

	support	itemsets
0	0.540305	(ChestPainType ASY)
1	0.404139	(ExerciseAngina)
2	0.553377	(HeartDisease)
3	0.789760	(Male)
4	0.595861	(No ExerciseAngina)
5	0.766885	(No FastingBS)
6	0.446623	(No HeartDisease)
7	0.601307	(RestingECG Normal)
8	0.501089	(ST_Slope Flat)
9	0.430283	(ST_Slope Up)
10	0.427015	(ChestPainType ASY, HeartDisease)
11	0.464052	(Male, ChestPainType ASY)
12	0.498911	(Male, HeartDisease)
13	0.415033	(ST_Slope Flat, HeartDisease)
14	0.432462	(No ExerciseAngina, Male)
15	0.584967	(Male, No FastingBS)
16	0.472767	(RestingECG Normal, Male)
17	0.419390	(ST_Slope Flat, Male)
18	0.469499	(No ExerciseAngina, No FastingBS)
19	0.480392	(RestingECG Normal, No FastingBS)

جدول ۲-۴-۱- الگوهای پرتکرار با اطمینان ۰/۴

با استفاده از تابع association_rules می‌توان قوانین انجمنی را به همراه lift به‌دست آورد.

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(ChestPainType ASY)	(HeartDisease)	0.540305	0.553377	0.427015	0.790323	1.428181
1	(HeartDisease)	(ChestPainType ASY)	0.553377	0.540305	0.427015	0.771654	1.428181
2	(Male)	(ChestPainType ASY)	0.789760	0.540305	0.464052	0.587586	1.087508
3	(ChestPainType ASY)	(Male)	0.540305	0.789760	0.464052	0.858871	1.087508
4	(Male)	(HeartDisease)	0.789760	0.553377	0.498911	0.631724	1.141580
5	(HeartDisease)	(Male)	0.553377	0.789760	0.498911	0.901575	1.141580
6	(ST_Slope Flat)	(HeartDisease)	0.501089	0.553377	0.415033	0.828261	1.496739
7	(HeartDisease)	(ST_Slope Flat)	0.553377	0.501089	0.415033	0.750000	1.496739
8	(No ExerciseAngina)	(Male)	0.595861	0.789760	0.432462	0.725777	0.918984
9	(Male)	(No ExerciseAngina)	0.789760	0.595861	0.432462	0.547586	0.918984
10	(Male)	(No FastingBS)	0.789760	0.766885	0.584967	0.740690	0.965842
11	(No FastingBS)	(Male)	0.766885	0.789760	0.584967	0.762784	0.965842
12	(RestingECG Normal)	(Male)	0.601307	0.789760	0.472767	0.786232	0.995532
13	(Male)	(RestingECG Normal)	0.789760	0.601307	0.472767	0.598621	0.995532
14	(ST_Slope Flat)	(Male)	0.501089	0.789760	0.419390	0.836957	1.059760
15	(Male)	(ST_Slope Flat)	0.789760	0.501089	0.419390	0.531034	1.059760
16	(No ExerciseAngina)	(No FastingBS)	0.595861	0.766885	0.469499	0.787934	1.027448
17	(No FastingBS)	(No ExerciseAngina)	0.766885	0.595861	0.469499	0.612216	1.027448
18	(RestingECG Normal)	(No FastingBS)	0.601307	0.766885	0.480392	0.798913	1.041764
19	(No FastingBS)	(RestingECG Normal)	0.766885	0.601307	0.480392	0.626420	1.041764

جدول ۲-۴-۲- قوانین انجمنی به همراه lift

درجدول بالا برخی از قوانین انجمنی (شماره‌های ۸ تا ۱۳) دارای lift کمتر از یک هستند، در نتیجه همبستگی آن‌ها منفی بوده و در صورت وجود یکی احتمال وجود دیگری کاهش می‌یابد.

برای مشخص کردن همبستگی‌های مثبت (lift بیشتر از یک)، جدول زیر را خواهیم داشت:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift
0	(ChestPainType ASY)	(HeartDisease)	0.540305	0.553377	0.427015	0.790323	1.428181
1	(HeartDisease)	(ChestPainType ASY)	0.553377	0.540305	0.427015	0.771654	1.428181
2	(Male)	(ChestPainType ASY)	0.789760	0.540305	0.464052	0.587586	1.087508
3	(ChestPainType ASY)	(Male)	0.540305	0.789760	0.464052	0.858871	1.087508
4	(Male)	(HeartDisease)	0.789760	0.553377	0.498911	0.631724	1.141580
5	(HeartDisease)	(Male)	0.553377	0.789760	0.498911	0.901575	1.141580
6	(ST_Slope Flat)	(HeartDisease)	0.501089	0.553377	0.415033	0.828261	1.496739
7	(HeartDisease)	(ST_Slope Flat)	0.553377	0.501089	0.415033	0.750000	1.496739
8	(ST_Slope Flat)	(Male)	0.501089	0.789760	0.419390	0.836957	1.059760
9	(Male)	(ST_Slope Flat)	0.789760	0.501089	0.419390	0.531034	1.059760
10	(No ExerciseAngina)	(No FastingBS)	0.595861	0.766885	0.469499	0.787934	1.027448
11	(No FastingBS)	(No ExerciseAngina)	0.766885	0.595861	0.469499	0.612216	1.027448
12	(RestingECG Normal)	(No FastingBS)	0.601307	0.766885	0.480392	0.798913	1.041764
13	(No FastingBS)	(RestingECG Normal)	0.766885	0.601307	0.480392	0.626420	1.041764

جدول ۲-۴-۳- قوانین انجمنی به همراه liftهای بزرگتر از ۱

۳-نتیجه‌گیری

با توجه به الگوریتم‌های بررسی شده برای مجموعه داده‌ها می‌توان گفت الگوریتم‌های خوشه‌بندی (K-Means و DBScan) مناسب این مجموعه‌داده نبوده و الگوریتم‌های طبقه‌بندی (SVM و MLP با پارامترهایی خاص) مناسب‌تر هستند.
از چالش‌هایی که برای الگوریتم‌های خوشه‌بندی با آن مواجه شدیم می‌توان به تعداد نامناسب خوشه‌های تشخیص داده اشاره کرد، زیرا با وجود از بین بردن داده‌های پرت و نرمال‌سازی داده‌ها انتظار می‌رفت که خوشه‌های تشخیص دادهشده برابر با دو باشد که برخلاف انتظار، سه خوشه تشخیص داده شد.
هم‌چنین انتظار می‌رفت پس از استفاده از GridSearchCV برای الگوریتم MLP دقت بالایی به دست آید که بر خلاف انتظار دقت تغییر چشم‌گیری نکرد. و مشکل دیگر این بود که زمان بالایی برای اجرا صرف شد.
در SVM نیز به‌طور معمول با افزایش C می‌توان به دقت بالاتر رسید، اما در این مجموعه داده با وجود kernel = rbf دقت لزوما افزایش نمی‌یابد.

۴-منابع

[1] "*Heart Disease UCI*", Kaggle, https://www.kaggle.com/ronitf/heart-disease-uci. Accessed 4 February 2022.

[2] Verma, Ujjawal. "*Data Cleaning and Preprocessing*", medium, 19 November 2019, https://medium.com/analytics-vidhya/data-cleaning-and-preprocessing-a4b751f4066f, Accessed 4 February 2022.

[3] Kwiatkowski, Robert. *"Customers clustering: K-Means, DBSCAN and AP"*, Kaggle, 20 January 2021, https://www.kaggle.com/datark1/customers-clustering-k-means-dbscan-and-ap, Accessed 4 February 2022.

[4] Mane, Tanmay. "*Nearest Neighbors to find optimal 'eps' in DBSCAN*", Kaggle, 19 March 2022, https://www.kaggle.com/tanmaymane18/nearestneighbors-to-find-optimal-eps-in-dbscan. Accessed 4 February 2022.

[5] Choudhury, Kaushik. "*Deep Neural Multilayer Perceptron (MLP) with Scikit-learn*", towardsdatascience, 31 Aguest 2020, https://towardsdatascience.com/deep-neural-multilayer-perceptron-mlp-with-scikit-learn-2698e77155e . Accessed 4 February 2022.

[6]"*sklearn.model_selection.GridSearchCV*",scikitlearn,https://scikitlearn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html?highlight=gridsearchcv#sklearn.model_selection.GridSearchCV. Accessed 4 February 2022.

[7] Verma, Niraj. "*Support Vector Machine detail analysis*", Kaggle, 31 January 2022, https://www.kaggle.com/nirajvermafcb/support-vector-machine-detail-analysis . Accessed 4 February 2022.

[8] McCullum, Nick. "*K Nearest Neighbors in Python - A Step-by-Step Guide*", nickmccullum, https://nickmccullum.com/python-machine-learning/k-nearest-neighbors-python/ . Accessed 4 February 2022.

[9] Surabhi. "*A Guide to the Naive Bayes Algorithm*", analyticsvidhya, 16 January 2021, https://www.analyticsvidhya.com/blog/2021/01/a-guide-to-the-naive-bayes-algorithm/. Accessed 4 February 2022.

[10] Banerjee, Prashant. "*Decision-Tree Classifier Tutorial*", Kaggle, 16 June 2020, https://www.kaggle.com/prashant111/decision-tree-classifier-tutorial. Accessed 4 February 2022.

[11] "*Ensemble methods*", scikitlearn, https://scikit-learn.org/stable/modules/ensemble.html. Accessed 4 February 2022.

[12] Moffitt, Chris. "*Introduction to Market Basket Analysis in Python*", pbpython, 3 July 2017, https://pbpython.com/market-basket-analysis.html. Accessed 4 February 2022.