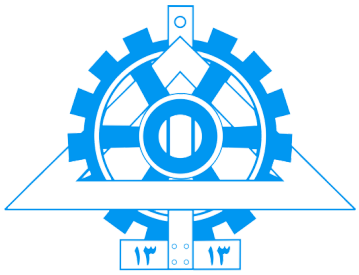


University Of Tehran

Faculty of Electrical and Computer Engineering

Artificial Intelligence

Computer Assignment #4

Machine Learning

Instructor: Dr. Yaghoobzadeh

Prepared by: Masoud Tahmasbi Fard - 810198429

12/27/2022

❖ بررسی مجموعه داده

❖ سوال 1

```
data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	635.000000	654.000000	680.000000	624.000000	680.000000	684.000000	590.000000	655.000000	768.000000
mean	3.700787	113.422018	68.786765	20.386218	80.123529	32.083626	0.466676	33.157252	0.348958
std	3.518126	202.816831	19.724841	15.987049	115.681140	7.800741	0.322408	13.829831	0.476951
min	-22.000000	-5000.000000	-2.000000	0.000000	0.000000	0.000000	0.078000	-150.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.375000	0.243250	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	34.000000	32.300000	0.368000	29.000000	0.000000
75%	6.000000	140.750000	80.000000	32.000000	129.250000	36.600000	0.611500	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.329000	81.000000	1.000000

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Pregnancies              635 non-null    float64
 1   Glucose                  654 non-null    float64
 2   BloodPressure            680 non-null    float64
 3   SkinThickness            624 non-null    float64
 4   Insulin                  680 non-null    float64
 5   BMI                      684 non-null    float64
 6   DiabetesPedigreeFunction 590 non-null    float64
 7   Age                      655 non-null    float64
 8   Outcome                  768 non-null    int64
dtypes: float64(8), int64(1)
memory usage: 54.1 KB
```

❖ سوال 2

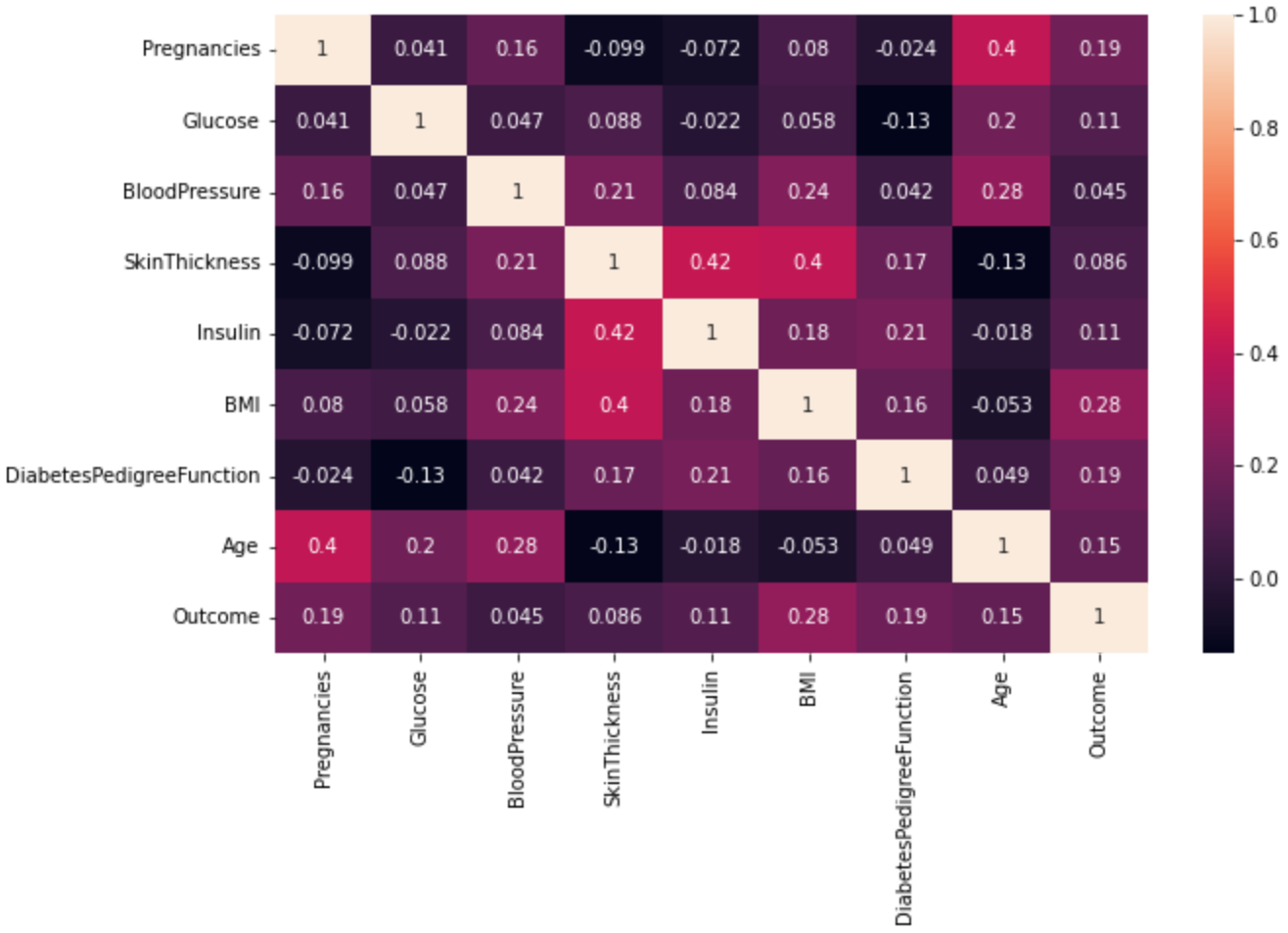
Pregnancies	17.32
Glucose	14.84
BloodPressure	11.46
SkinThickness	18.75
Insulin	11.46
BMI	10.94
DiabetesPedigreeFunction	23.18
Age	14.71
Outcome	0.00
dtype: float64	

درصد داده‌های از دست رفته در هر ویژگی

Pregnancies	133
Glucose	114
BloodPressure	88
SkinThickness	144
Insulin	88
BMI	84
DiabetesPedigreeFunction	178
Age	113
Outcome	0
dtype: int64	

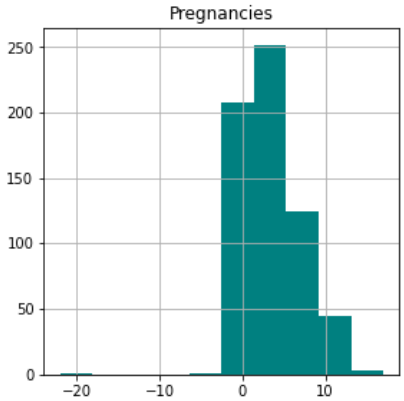
درصد داده‌های از دست رفته در هر ویژگی

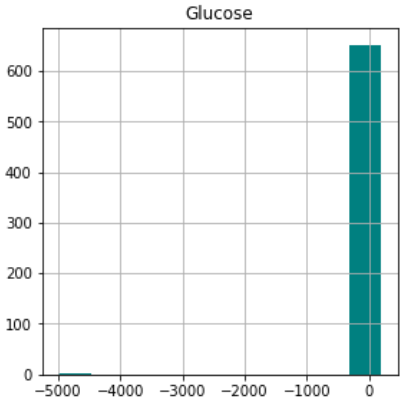
❖ سوال 3

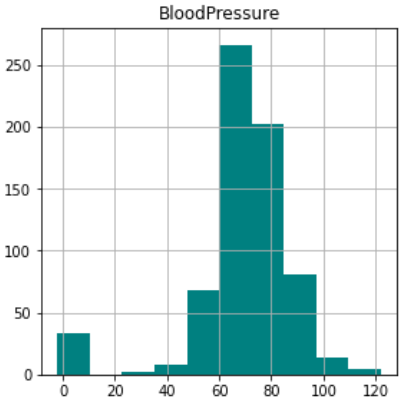


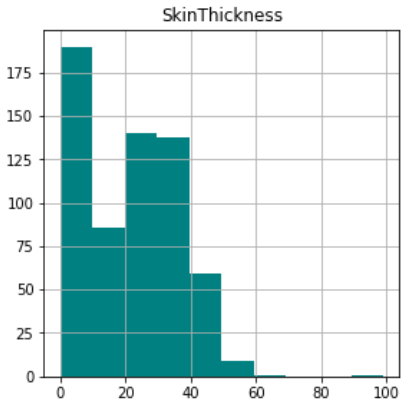
با استفاده از نمودار فوق، می‌توان گفت که ویژگی‌های BMI، DiabetesPedigreeFunction، Age **و** Pregnancies **ارتباط بیشتری دارد.**

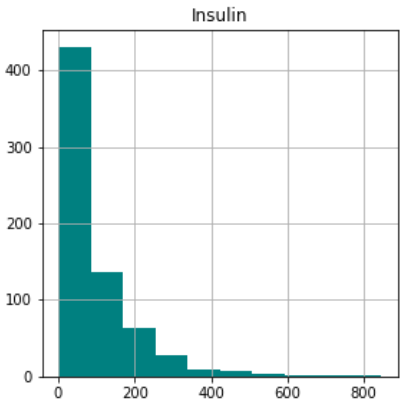
❖ سوال 4

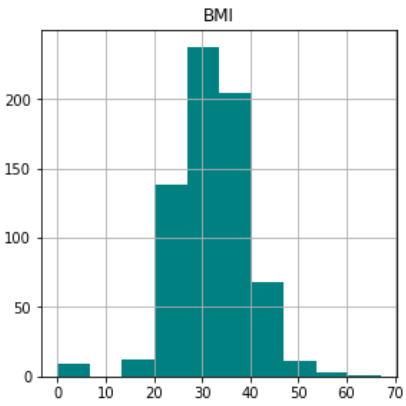

Pregnancies
250
200
150
100
50
0
−20
−10
0
10

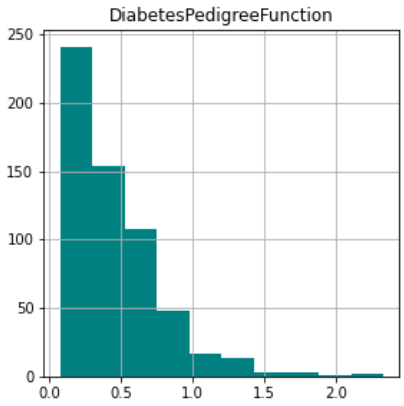

Glucose
600
500
400
300
200
100
0
−5000
−4000
−3000
−2000
−1000
0

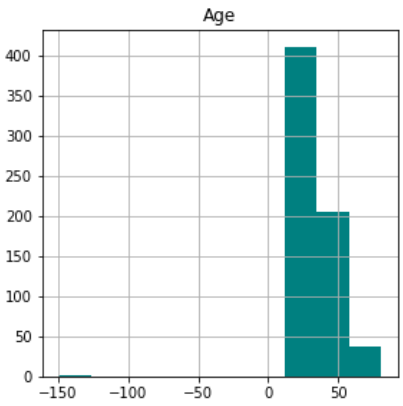

BloodPressure
250
200
150
100
50
0
0
20
40
60
80
100
120

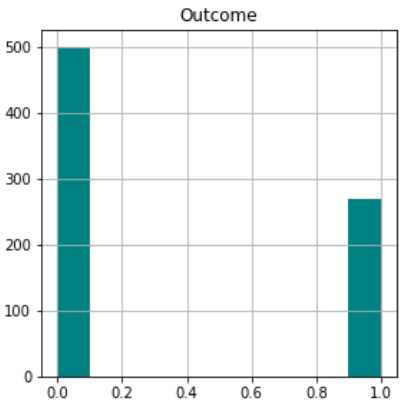

SkinThickness
175
150
125
100
75
50
25
0
0
20
40
60
80
100

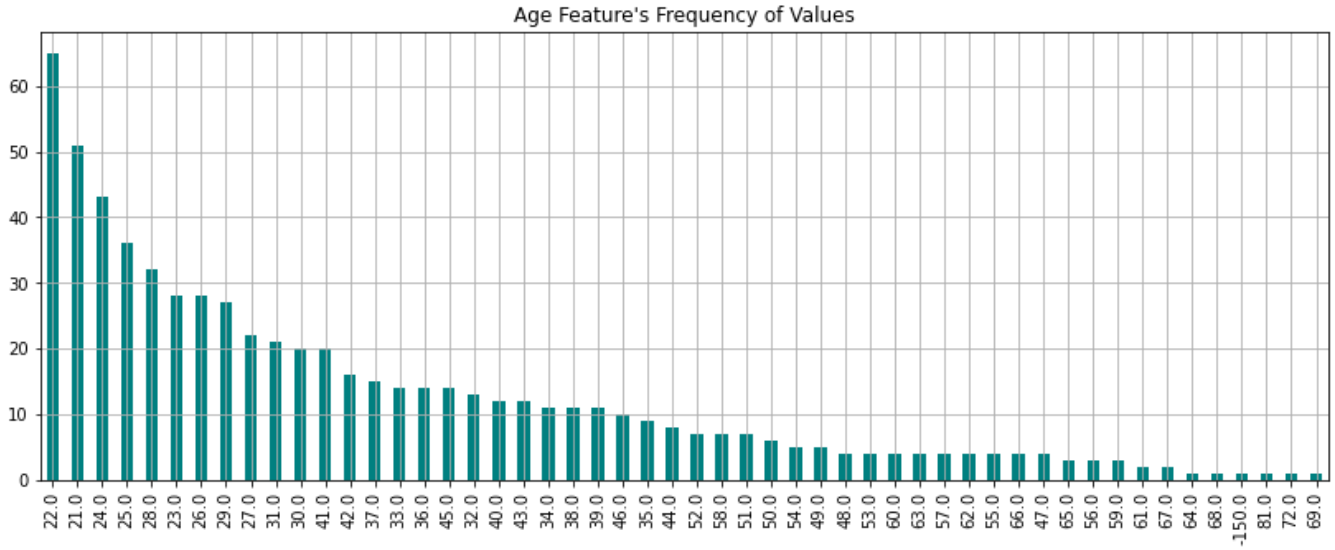

Insulin
400
300
200
100
0
0
200
400
600
800

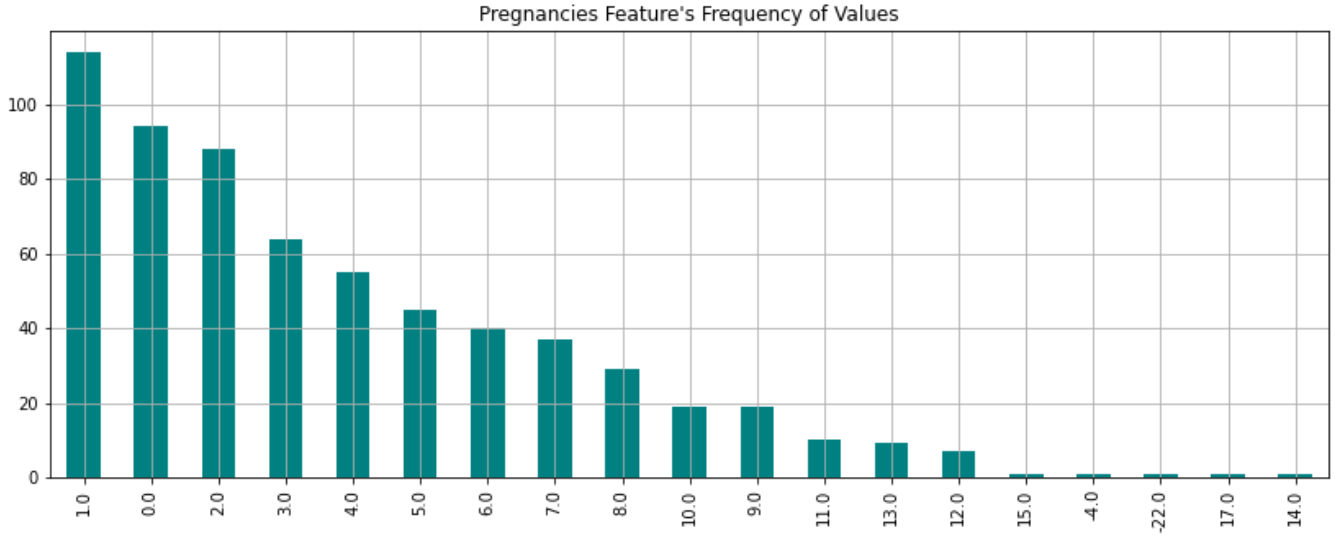

BMI
200
150
100
50
0
0
10
20
30
40
50
60
70

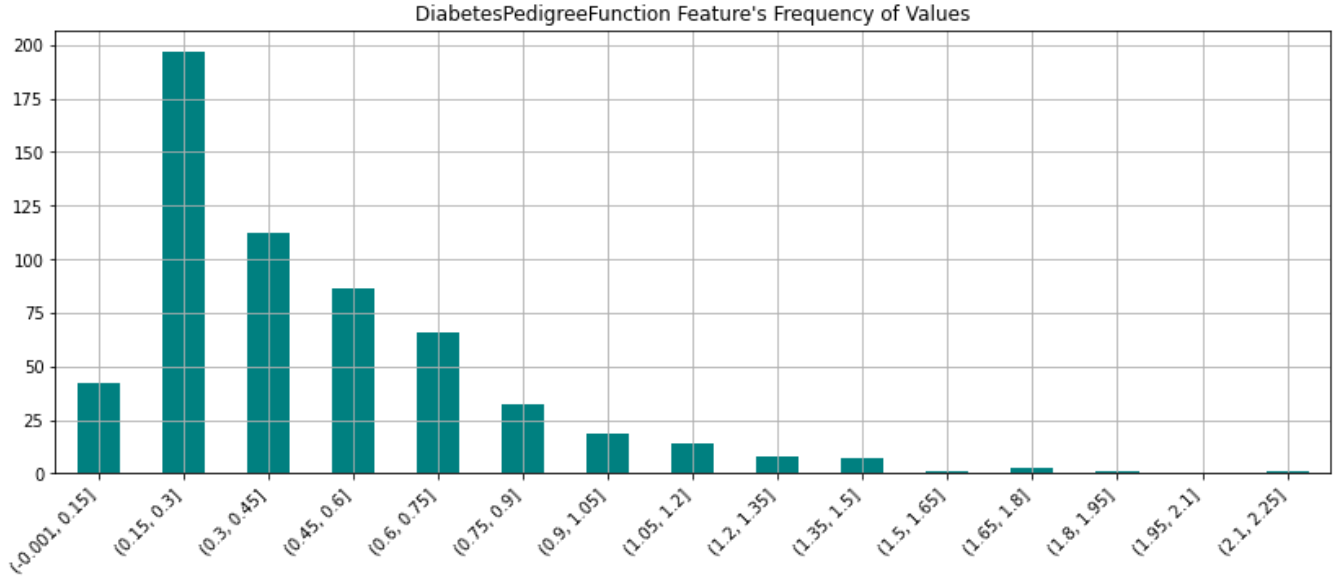

DiabetesPedigreeFunction
250
200
150
100
50
0
0.0
0.5
1.0
1.5
2.0

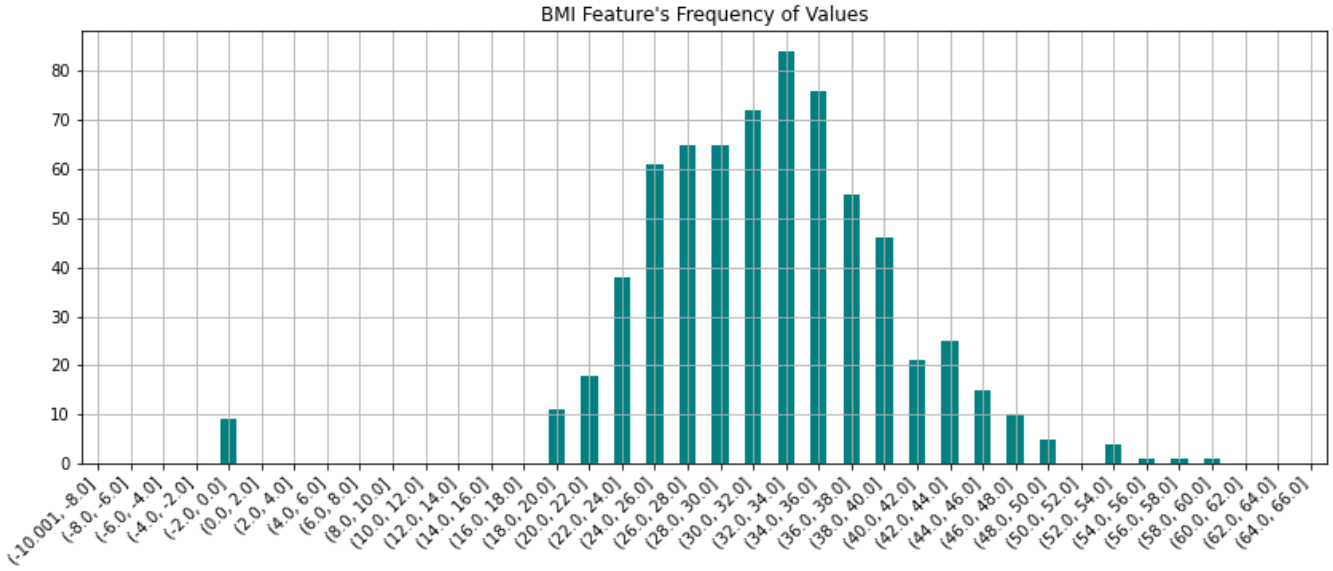

Age
400
350
300
250
200
150
100
50
0
−150
−100
−50
0
50


Outcome
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0

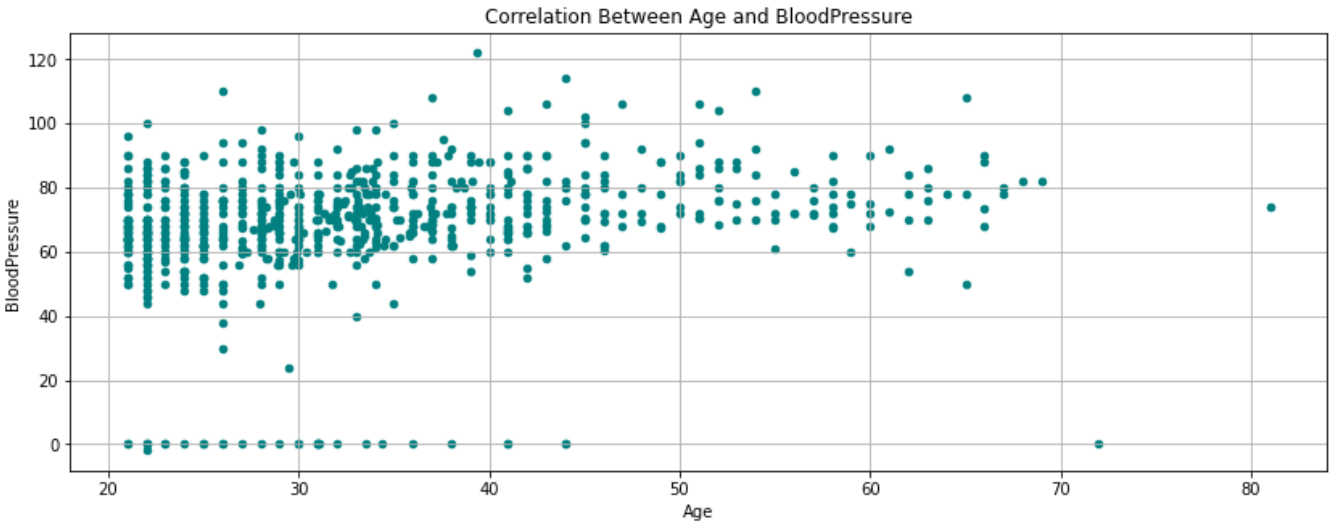

Age Feature's Frequency of Values
0
10
20
30
40
50
60
22.0
21.0
24.0
25.0
28.0
23.0
26.0
29.0
27.0
31.0
30.0
41.0
42.0
37.0
33.0
36.0
45.0
32.0
40.0
43.0
34.0
38.0
39.0
46.0
35.0
44.0
52.0
58.0
51.0
50.0
54.0
49.0
48.0
53.0
60.0
63.0
57.0
62.0
55.0
66.0
47.0
65.0
56.0
59.0
61.0
67.0
64.0
68.0
-150.0
81.0
72.0
69.0

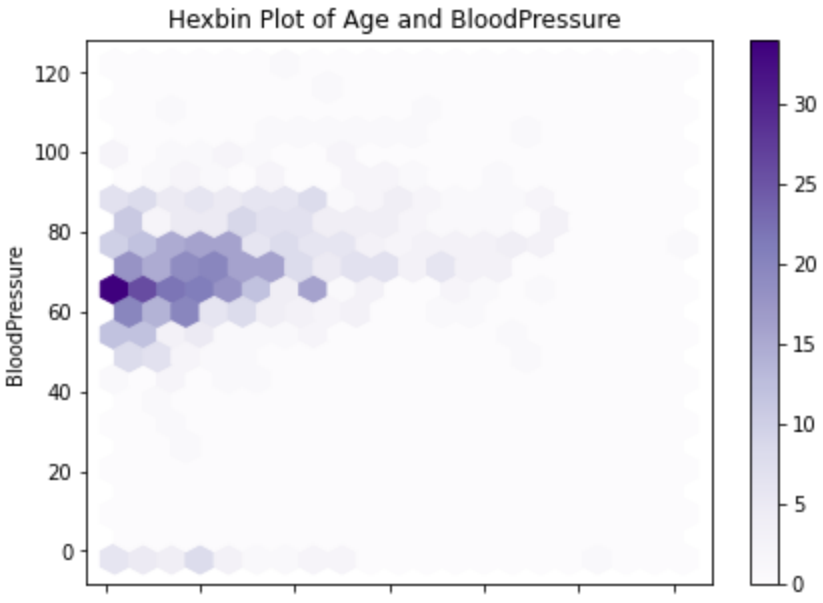

Pregnancies Feature's Frequency of Values
0
20
40
60
80
100
1.0
0.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
10.0
9.0
11.0
13.0
12.0
15.0
-4.0
-22.0
17.0
14.0

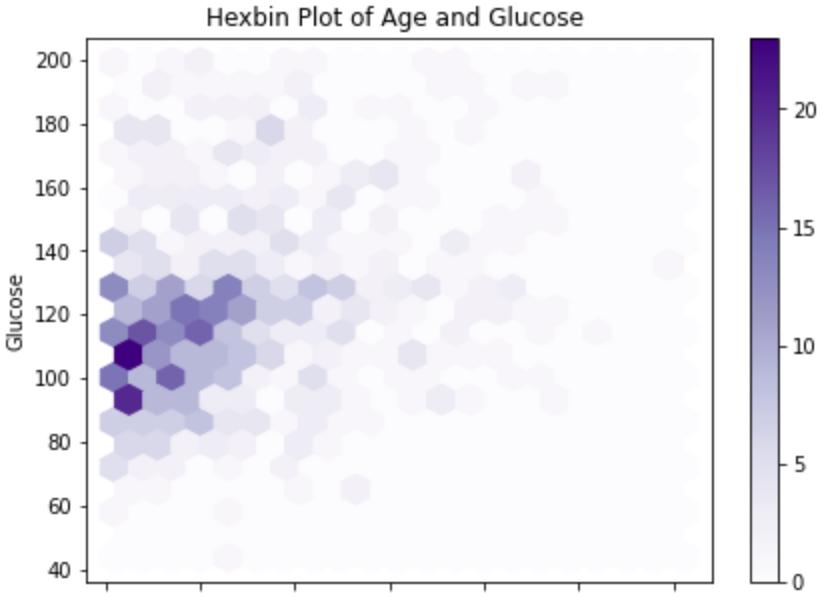

DiabetesPedigreeFunction Feature's Frequency of Values
0
25
50
75
100
125
150
175
200
(-0.001, 0.15]
(0.15, 0.3]
(0.3, 0.45]
(0.45, 0.6]
(0.6, 0.75]
(0.75, 0.9]
(0.9, 1.05]
(1.05, 1.2]
(1.2, 1.35]
(1.35, 1.5]
(1.5, 1.65]
(1.65, 1.8]
(1.8, 1.95]
(1.95, 2.1]
(2.1, 2.25]


BMI Feature's Frequency of Values
0
10
20
30
40
50
60
70
80
(-10.001, -8.0]
(-8.0, -6.0]
(-6.0, -4.0]
(-4.0, -2.0]
(-2.0, 0.0]
(0.0, 2.0]
(2.0, 4.0]
(4.0, 6.0]
(6.0, 8.0]
(8.0, 10.0]
(10.0, 12.0]
(12.0, 14.0]
(14.0, 16.0]
(16.0, 18.0]
(18.0, 20.0]
(20.0, 22.0]
(22.0, 24.0]
(24.0, 26.0]
(26.0, 28.0]
(28.0, 30.0]
(30.0, 32.0]
(32.0, 34.0]
(34.0, 36.0]
(36.0, 38.0]
(38.0, 40.0]
(40.0, 42.0]
(42.0, 44.0]
(44.0, 46.0]
(46.0, 48.0]
(48.0, 50.0]
(50.0, 52.0]
(52.0, 54.0]
(54.0, 56.0]
(56.0, 58.0]
(58.0, 60.0]
(60.0, 62.0]
(62.0, 64.0]
(64.0, 66.0]

❖ سوال 5

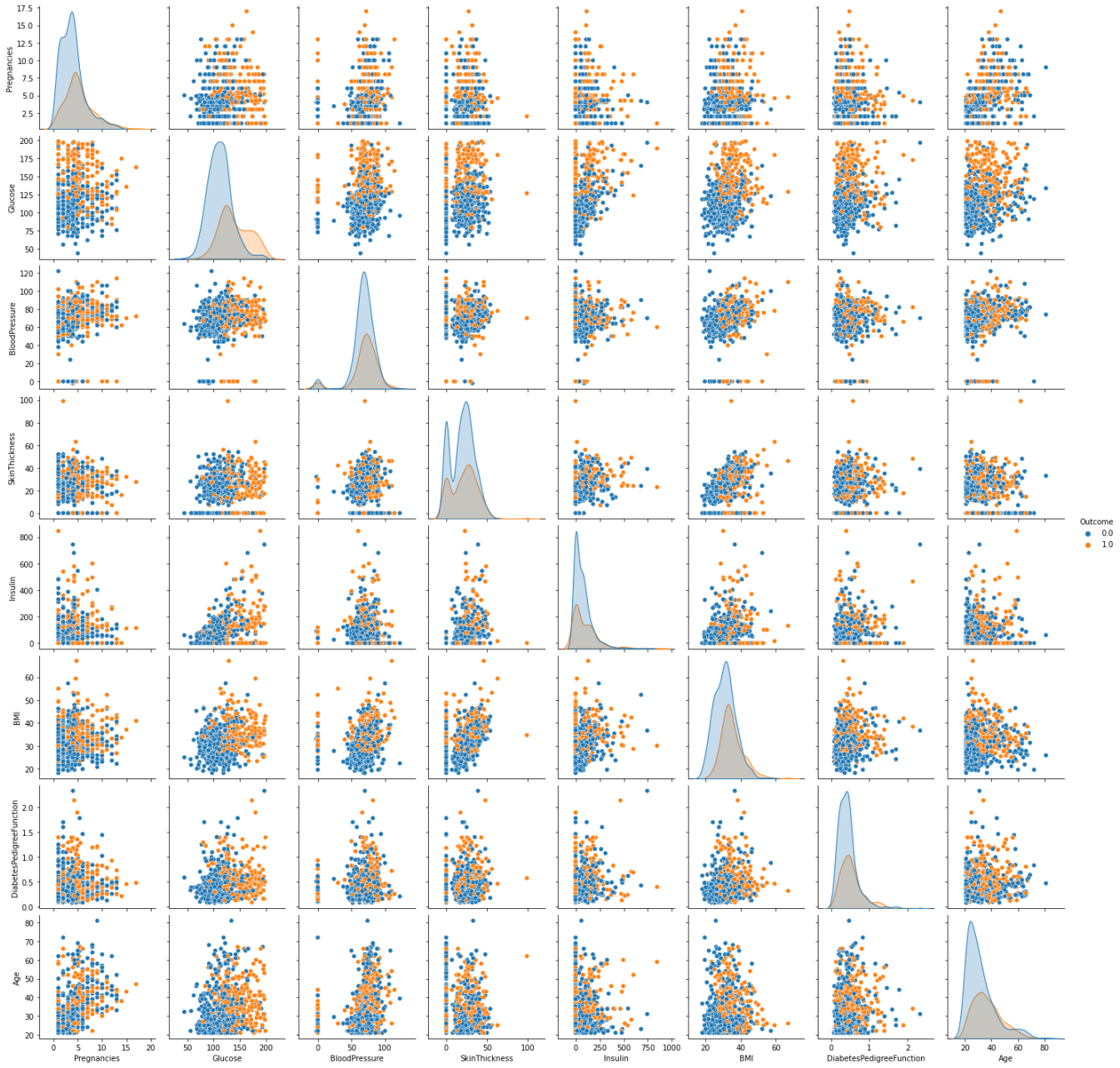

Correlation Between Age and BloodPressure
BloodPressure
0
20
40
60
80
100
120
Age
20
30
40
50
60
70
80

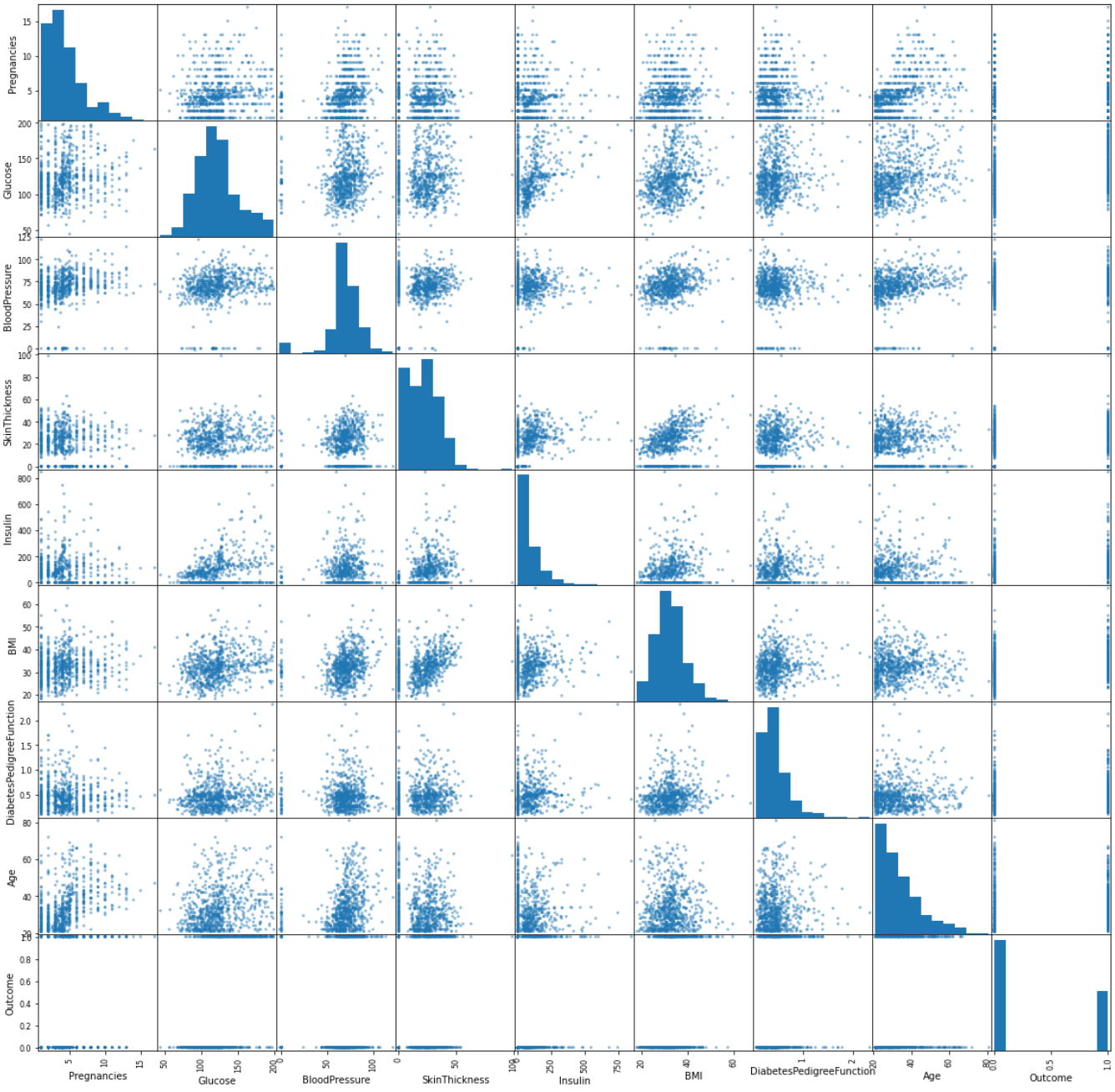

Hexbin Plot of Age and BloodPressure
BloodPressure
0
20
40
60
80
100
120
0
5
10
15
20
25
30



نمودارهای مربوط به ارتباط بقیه‌ی ویژگی‌ها در Notebook آپلود شده قابل مشاهده است و برای جلوگیری از شلوغی بیش از حد گزارش‌کار، از گزارش بقیه‌ی نمودارها صرف نظر شده است.

❖ سوال 6


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
0.0
1.0


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome

❖ پیش‌پردازش مجموعه داده

❖ سوال 1

برای جایگزین کردن داده‌های از دست‌رفته، روش‌های متفاوتی وجود دارد. مشهورترین و شاید ساده‌ترین این روش‌ها، حذف ستون دارای داده‌های از دست‌رفته یا جایگذاری این داده‌ها با میانگین، میانه یا مد است. اما همانگونه که اشاره شد، این روش‌ها روش‌های ساده‌ای هستند. برخی روش‌های دیگر برای جایگزین کردن داده‌های از دست‌رفته، در ادامه آورده شده است.

- جایگزین کردن با صفر یا یک مقدار ثابت: در این روش، داده‌های از دست‌رفته را با یک مقدار ثابت یا صفر جایگزین می‌کنیم. این روش ساده باعث می‌شود در هنگام انجام محاسبات دچار مشکل نشویم. اما یک روش منطقی به نظر نمی‌رسد چرا که می‌تواند مشکلاتی همچون اضافه‌کردن داده‌های دورافتاده، تغییر توزیع واقعی داده‌ها و ... را در پی داشته باشد.
- استفاده از k-NN : k-NN الگوریتمی است که برای طبقه‌بندی استفاده می‌شود. این الگوریتم از شباهت ویژگی‌ها برای پیش‌بینی مقادیر در نقاط مورد نظر استفاده می‌کند؛ بدین صورت که با توجه به میزان شباهتی که نقطه‌ی جدید به دیگر نقاط دارد، پیش‌بینی را انجام می‌دهد. این کار می‌تواند برای پیش‌بینی مقدار داده‌های از دست‌رفته مفید باشد و با توجه به همسایه‌های داده‌ی از دست‌رفته، مقدار آن را پیش‌بینی کنیم.

این روش نسبت به روش‌های اشاره‌شده‌ی قبلی، دقت به مراتب بالاتری می‌تواند داشته باشد، اما نیازمند محاسبات سنگین‌تری می‌باشد. همچنین این روش نسبت به داده‌های پرت آسیب‌پذیر است.

- استفاده از (Imputation Using Multivariate Imputation by Chained Equation (MICE:
این روش بدین گونه عمل می‌کند که داده‌های از دست رفته را چندین بار پیش‌بینی می‌کند. Multiple Imputation نسبت به single imputation عملکرد بهتری دارد، چرا که با پیش‌بینی چندباره‌ی داده‌های از دست‌رفته، بهتر می‌توانند نایقینی‌ها را حل کنند. chained equationها نیز بسیار انعطاف‌پذیر هستند و می‌توانند انواع داده‌ها (داده‌های پیوسته، باینری و ...) را پشتیبانی کنند.

- استفاده از شبکه‌های عصبی عمیق: استفاده از شبکه‌های عصبی عمیق یکی دیگر از روش‌های جایگزینی داده‌های از دست‌رفته است که بخصوص بر روی داده‌های غیرعددی به خوبی جواب می‌دهد. کتابخانه‌های موجود از الگوریتم‌های یادگیری ماشین با استفاده از شبکه‌های عصبی عمیق برای پیش‌بینی داده‌های از دست‌رفته استفاده می‌کنند. این کتابخانه‌ها را می‌توان هم بر روی CPU و هم بر روی GPU اجرا کرد.

دقت بالا نسبت به روش‌های قبل، پشتیبانی از داده‌های غیر عددی و قابلیت اجرا بر روی GPU از مزیت‌های این روش هستند. در عین حال این روش بدون عیب نمی‌باشد. از معایب این روش می‌توان به جایگزینی تنها یک ستون و مدت زمان اجرای بالا برای داده‌های بزرگ اشاره کرد. همچنین در این روش ستون‌های حاوی اطلاعات و مرتبط با ستونی که می‌خواهیم مقادیر از دست‌رفته‌ی آن را جایگزین کنیم را باید مشخص کنیم.

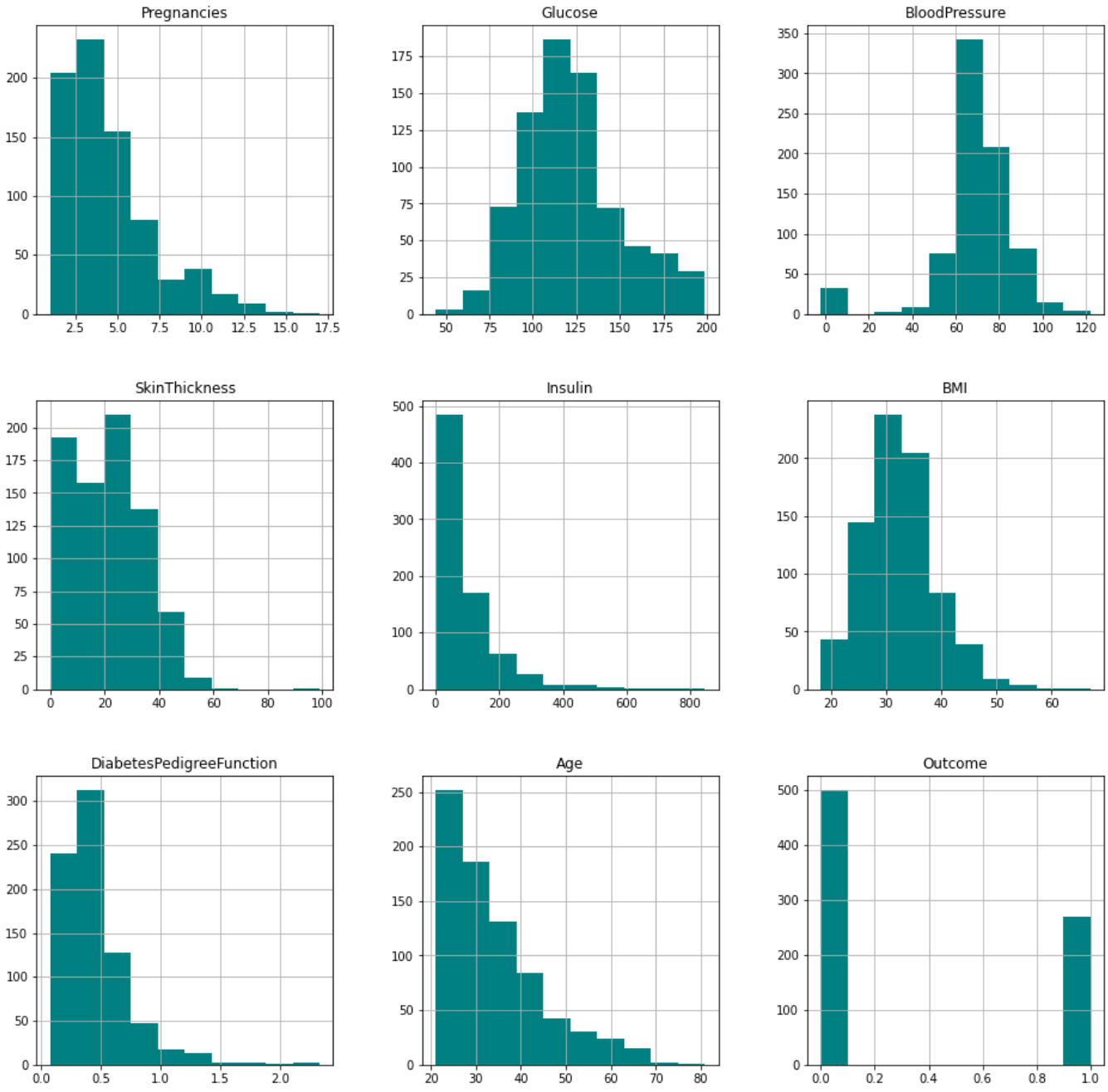
- استفاده از Stochastic regression imputation: این روش سعی می‌کند با استفاده از رگرسیون بر روی نقاط موجود و برخی نقاط تصادفی دیگر، داده‌های از دست‌رفته را پیش‌بینی کند.
- استفاده از درون‌یابی یا برون‌یابی
- جایگزینی با مقدار سطر قبلی: این روش نیز هر چند ساده است، اما از آنجا که از بقیه‌ی ویژگی‌های موجود برای پیش‌بینی استفاده نمی‌کند، نمی‌تواند نتیجه‌ی آنچنان مناسبی داشته باشد.

❖ سوال 2

با توجه به نتایج بخش قبل، ویژگی‌های Pregnancies، SkinThickness ، DiabetesPedigreeFunction بیشترین داده‌های از دست‌رفته را دارند.

روشی که ما برای جایگزین کردن داده‌های از دست‌رفته در این مسئله پیش می‌گیریم، استفاده از KNNImputer که مبنای کار آن بر اساس k-NN می‌باشد.

همچنین برای اینکه داده‌های پرت را نیز دور بریزیم، به جای داده‌های پرت، NaN قرار می‌دهیم و سعی دوباره مقدار آن‌ها را با استفاده از KNNImputer پیش‌بینی می‌کنیم. پس از اعمال تغییرات فوق، توزیع داده‌ها به صورت زیر خواهد بود.



همانگونه که مشخص است، KNNImputer به خوبی توانسته است مقادیر از دست رفته را پیش‌بینی کند به طوری که توزیع اصلی داده‌ها دچار تغییر نشود.

❖ سوال 3

استانداردسازی یا تبدیل Z-score، برای یک مجموعه داده، بدست آوردن مقدارهایی است که دارای میانگین صفر و واریانس یا انحراف استاندارد ۱ باشند. بنابراین اگر میانگین داده‌های اصلی برابر با μ و انحراف معیار آن‌ها نیز σ باشد، مقدار Z را براساس رابطه زیر می‌توان بدست آورد.

$$Z = \frac{x - \mu}{\sigma}$$

استانداردسازی زمانی که با ویژگی و داده‌هایی با مقیاس‌های مختلف سروکار داریم، بسیار مهم است. برای مثال در الگوریتم گرادیان کاهشی که یک روش برای بهینه‌سازی محسوب می‌شود، ممکن است بعضی از متغیرها با توجه به مقیاس متفاوتی که دارند، باعث کاهش سریعتر مشتق در یک بُعد شوند. یا به عنوان یک مثال دیگر می‌توان به الگوریتم k-نزدیکترین همسایه (K-Nearest Neighbor- KNN) نیز اشاره کرد که با توجه به مقیاس داده‌ها و بهره‌گیری از تابع «فاصله اقلیدسی» (Euclidean Distance)، وزن بیشتری به متغیرها با مقیاس یا واحدهای بزرگتر بدهد در نتیجه به شکل نادرست گروه یا دسته‌ها تشکیل خواهند شد.

یکی دیگر از روش‌های تغییر مقیاس، استفاده از روش نرمال‌سازی Min-Max است. به این ترتیب علاوه بر یکسان سازی مقیاس داده‌ها، کران‌های تغییر آن‌ها نیز در بازه [0,1] خواهد بود. این تبدیل به صورت زیر تعریف می‌شود.

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

این شیوه محاسبه اغلب در زمانی استفاده می‌شود که می‌خواهیم میزان شباهت بین نقاط را مشخص کنیم. برای مثال در پردازش تصویر و تشخیص پیکسل‌های مشابه از این تبدیل استفاده شده و سپس از الگوریتم‌های خوشه‌بندی برای کاهش تعداد رنگ استفاده می‌شود.

بنابراین با توجه به دلایل مطرح شده، داده‌ها نیازمند standardizing یا normalizing هستند. ما از روش standardizing استفاده می‌کنیم. آنجا که می‌خواهیم از روش k-NN برای پر کردن داده‌های از دست‌رفته استفاده کنیم، ابتدا داده‌ها را استاندارسازی می‌کنیم، سپس این الگوریتم را اعمال می‌کنیم.

❖ سوال 4

داده‌های دسته‌ای(کیفی) به دو بخش تقسیم می‌شوند؛ داده‌های ترتیبی و داده‌های اسمی. داده‌های ترتیبی، داده‌هایی هستند که ترتیب در آن‌ها برایمان مهم است، اما در داده‌های اسمی، ترتیب داده‌ها اهمیتی ندارد. یکی از راه‌های استفاده از این دسته داده‌ها، encoding است، به این معنی که به هر یک از دسته‌ها یک عدد نسبت دهیم. روش‌های مختلفی برای encoding وجود دارد که به برخی از آن‌ها در ادامه اشاره شده است:

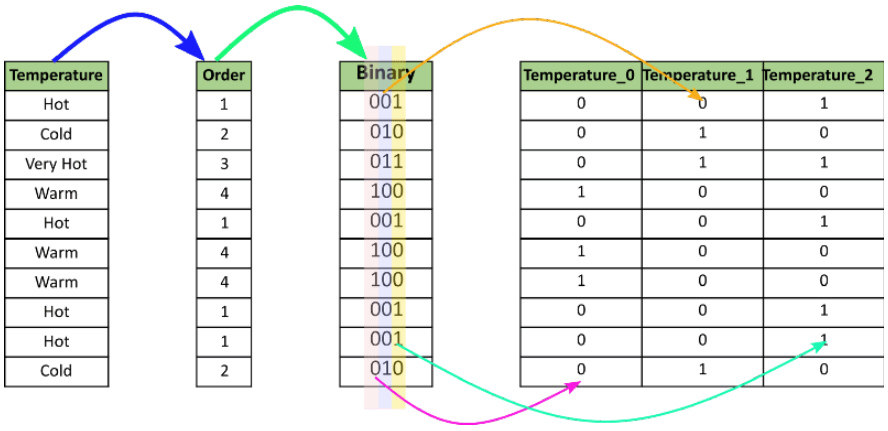
- One-hot Encoding: یک بردار به طول تعداد دسته‌ها ایجاد می‌کنیم. هرگاه آن ویژگی وجود داشت، مقدار متناظر با آن ویژگی را در بردار تعریف شده برابر 1 و در غیر این صورت برابر صفر در نظر می‌گیریم. شکل زیر نمونه‌ای از این روش را نشان می‌دهد.

	gender	class	city
0	Male	A	Delhi
1	Female	B	Gurugram
2	Male	C	Delhi
3	Female	D	Delhi
4	Female	A	Gurugram

→

	gender_1	gender_2	class	city_1	city_2
0	1	0	A	1	0
1	0	1	B	0	1
2	1	0	C	1	0
3	0	1	D	1	0
4	0	1	A	0	1

- Binary Encoding:در این روش ابتدا به دسته‌ها، اعداد ده‌دهی اختصاص داده می‌شود. سپس اعداد اختصاص یافته را به صورت باینری درمی‌آوریم.



Temperature
Hot
Cold
Very Hot
Warm
Hot
Warm
Warm
Hot
Hot
Cold

Order
1
2
3
4
1
4
4
1
1
2

Binary
001
010
011
100
001
100
100
001
001
010

Temperature_0	Temperature_1	Temperature_2
0	0	1
0	1	0
0	1	1
1	0	0
0	0	1
1	0	0
1	0	0
0	0	1
0	0	1
0	1	0

روش‌های دیگری مانند Label Encoding، Ordinal Encoding، Frequency Encoding و ... وجود دارد.

برای روش‌هایی مانند درخت تصمیم، نیازی به encoding نداریم ولی برای روش‌هایی مانند شبکه‌های عصبی، باید encoding انجام شود.

❖ سوال 5

ما به دنبال پیش‌بینی مقادیر ستون Outcome هستیم و می‌خواهیم اینکار را با استفاده از مقادیر بقیه‌ی ستون‌ها انجام دهیم. بنابراین اگر ستونی داشته باشیم که برای این پیش‌بینی به ما کمکی نکند، می‌توانیم آن را کنار بگذاریم. به عنوان مثال با توجه به نتایج بخش قبل،ستون‌های SkinThickness و BloodPressure همبستگی خیلی کمی با Outcome دارند، بنابراین برای ساده‌تر شدن محاسبات می‌توانیم این ستون‌ها را کنار بگذاریم. همچنین اگر مقادیر زیادی از یک ستون از دست رفته باشد نیز کنار گذاشتن آن ستون می‌تواند انتخاب خوبی باشد.

❖ سوال 6,7

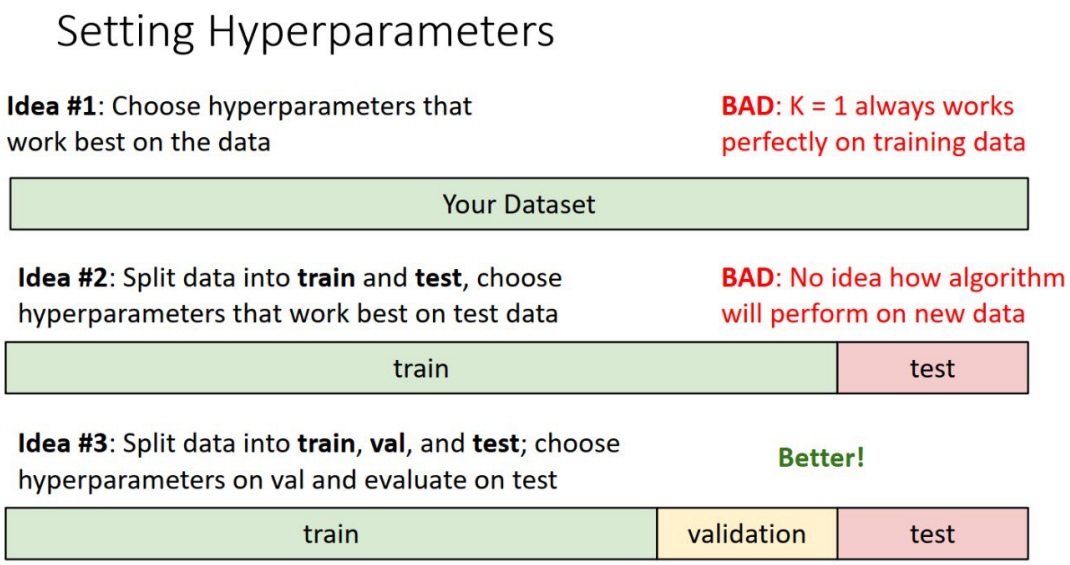
کل داده‌ها معمولا به سه بخش تقسیم می‌شوند؛ داده‌های Train، داده‌های Test و داده‌های validation.

- داده‌های Train: این داده‌ها برای آموزش مدل استفاده می‌شوند. معمولا 70 یا 80 درصد کل داده‌های در دسترس، به این دسته تعلق دارد. وظیفه اصلی این داده‌ها تنظیم دقیق پارامترهای مدل است.

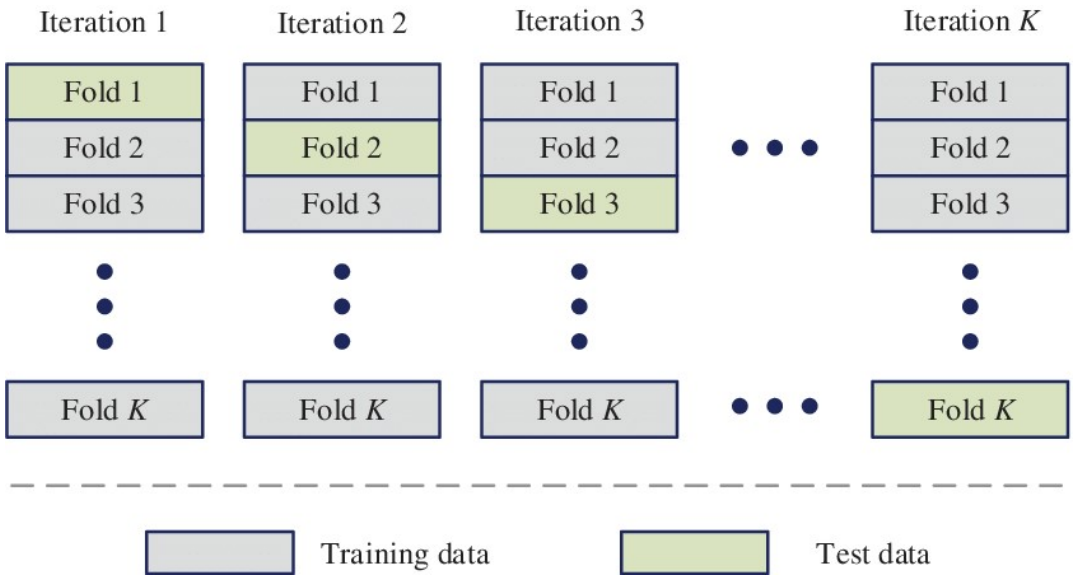
به عنوان مثال در یک شبکه عصبی عمیق که از تعدادی لایه تشکیل شده است، داده‌های train از لایه اول به لایه آخر رفته و سپس با توجه به خروجی، طی back-propagation وزن‌ها آپدیت می‌شوند. اما داده‌های test و validation به صورت مستقیم به شبکه داده می‌شوند و تاثیری بر روی وزن‌های مدل ندارند. به همین دلیل است که معمولا حجم زیادی از داده‌ها به این دسته تعلق میگیرد تا آپدیت شدن پارامترهای مدل با احتمال بیشتری متناسب با توزیع داده‌های واقعی باشد.

- داده‌های validation: در طی فرآیند یادگیری، معمولا داده‌ها چندین بار به مدل داده می‌شوند و متناسب با آن، پارامترهای آپدیت می‌شوند. این موضوع باعث می‌شود تا موضوع کم بودن داده‌ها رو با نشان دادن چند باره داده‌ها جبران کنیم. اما پس از چند بار اعمال ورودی‌ها، مدل به جای یادگیری، به سمت حفظ کردن ورودی خواهد رفت و به اصطلاح دچار overfitting خواهد شد. اینجاست که از داده‌های validation استفاده می‌کنیم. در هر Epoch، مدل را روی داده‌های validation امتحان می‌کنیم و عملکرد مدل را ارزیابی می‌کنیم. تا یک حدی، با کم شدن خطای مدل روی داده های آموزش، خطا روی داده‌های validation نیز کاهش می‌یابد؛ اما از جایی به بعد که مدل داده‌های آموزش را حفظ می‌کند، خطای مدل روی داده‌های اعتبارسنجی به جای کم شدن، بیشتر می‌شود. اینجاست که متوجه می‌شویم مدل دچار overfit شده است. همچنین از داده‌های validation برای سِت کردن مقادیر hyper-parameter های مدل استفاده می‌کنیم. استفاده از این داده‌ها به جای داده‌های تست برای این کار، باعث می‌شود تا نتایج بدست آمده از داده‌های test دقیق‌تر باشند، چرا که مدل واقعا داده‌های test را قبلا ندیده و هیچگونه بایاسی وجود ندارد.

- داده‌های test: در نهایت برای بررسی عملکرد مدل و محک زدن آن، داده‌های test را به مدل می‌دهیم و با مقایسه‌ی خروجی شبکه نسبت به این ورودی‌ها با خروجی‌های صحیح، دقت مدل را ارزیابی می‌کنیم. شکل زیر نحوه‌ی تقسیم داده‌ها را نشان می‌دهد.



برای جداسازی داده‌ها روش‌های متنوعی وجود دارد اما معروف‌ترین تابع، تابع train_test_split از کتابخانه‌ی sklearn است. البته به طور دستی و بدون استفاده از کتابخانه‌های آماده نیز می‌توان داده‌ها را به نسبت‌های مورد نظر جدا کرد. برای جدا کردن داده‌های validation همچنین می‌توان از روش Cross Validation استفاده کرد که شماتیک آن در ادامه آورده شده است.



❖ آموزش، ارزیابی و تنظیم

❖ سوال 1 و 2

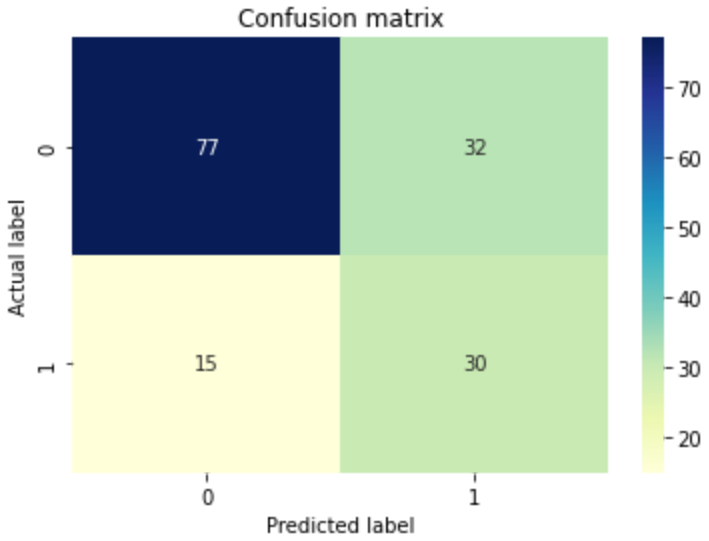
- درخت تصمیم:

```
Decision Tree Accuarcy on Train Data:73.0%
```

دقت بدست آمده بر روی داده‌های train

```
DecisionTreeClassifier(criterion='entropy', max_depth=2, max_leaf_nodes=2,
                       min_samples_split=0.001)
```

مقادیر بهینه‌ی بدست آمده برای پارامترها



	precision	recall	f1-score	support
0.0	0.84	0.71	0.77	109
1.0	0.48	0.67	0.56	45
accuracy			0.69	154
macro avg	0.66	0.69	0.66	154
weighted avg	0.73	0.69	0.71	154

Classification Report

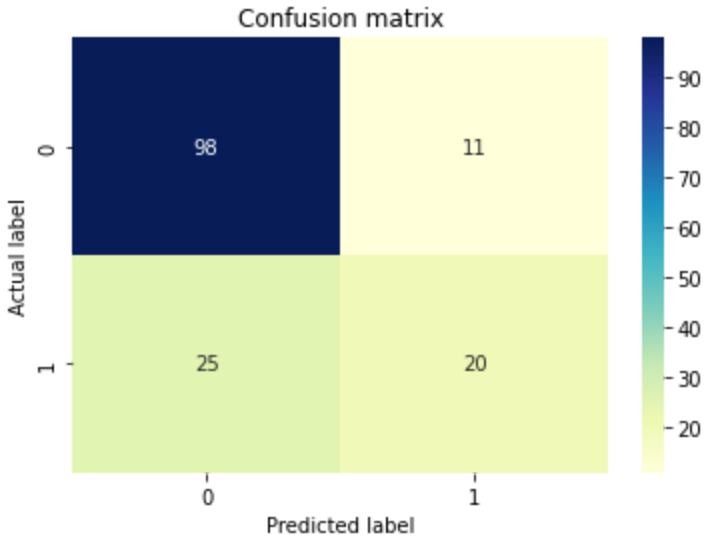
- K نزدیک ترین همسایه:

```
Decision Tree Accuarcy on Train Data:78.0%
```

دقت بدست آمده بر روی داده‌های train

```
KNeighborsClassifier(n_neighbors=10)
```

مقادیر بهینه‌ی بدست آمده برای پارامترها



	precision	recall	f1-score	support
0.0	0.80	0.90	0.84	109
1.0	0.65	0.44	0.53	45
accuracy			0.77	154
macro avg	0.72	0.67	0.69	154
weighted avg	0.75	0.77	0.75	154

Classification Report

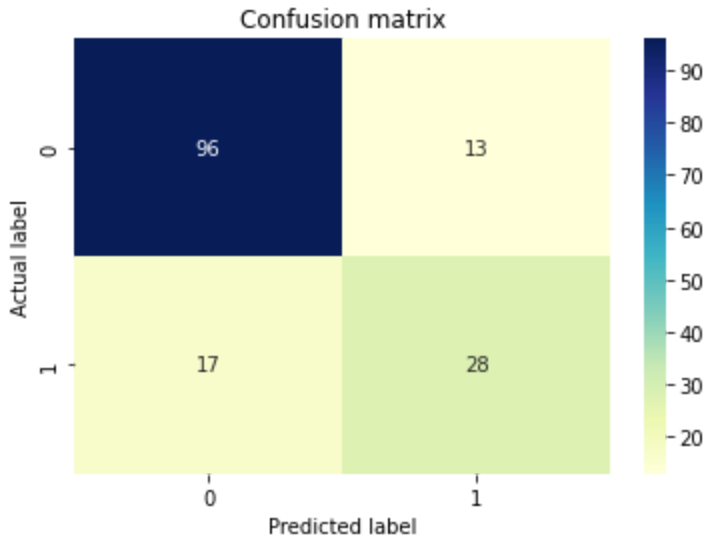
- رگرسیون لجستیک:

```
Decision Tree Accuarcy on Train Data:78.0%
```

دقت بدست آمده بر روی داده‌های train

```
LogisticRegression(C=1000.0)
```

مقادیر بهینه‌ی بدست آمده برای پارامترها

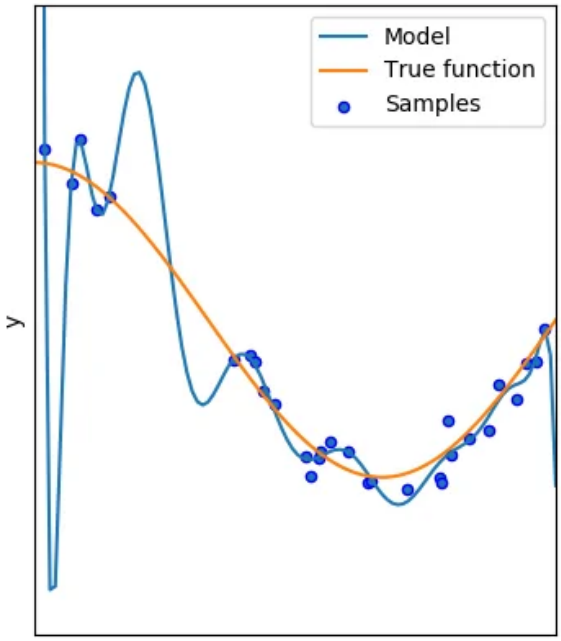


	precision	recall	f1-score	support
0.0	0.85	0.88	0.86	109
1.0	0.68	0.62	0.65	45
accuracy			0.81	154
macro avg	0.77	0.75	0.76	154
weighted avg	0.80	0.81	0.80	154

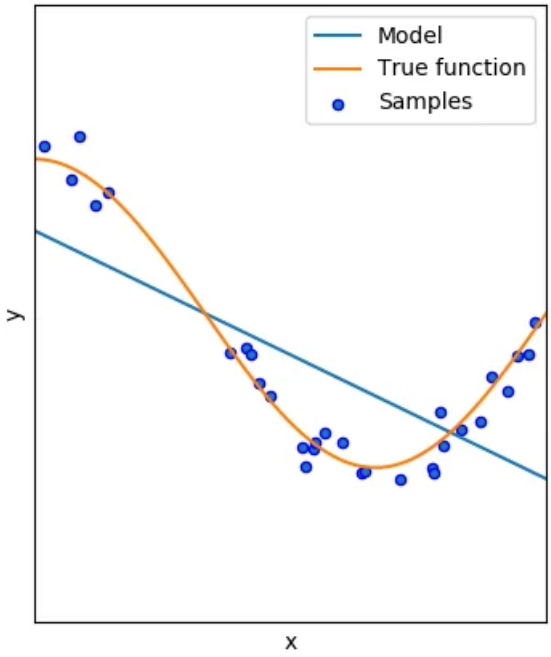
Classification Report

❖ سوال 3

Overfit شدن به معنای این است که الگوریتم فقط داده‌هایی که در مجموعه آموزشی (train set) یاد گرفته است را می‌تواند به درستی پیش‌بینی کند ولی اگر داده‌ای کمی از مجموعه‌ی آموزشی فاصله داشته باشد، الگوریتمی که Overfit شده باشد، نمی‌تواند به درستی پاسخی برای این داده‌های جدید پیدا کند و آن‌ها را با اشتباه زیادی طبقه‌بندی می‌کند. به عبارت دیگر، مدل overfit، مدلی بسیار پیچیده برای داده‌ها است. به این معنی که در تحلیل رگرسیونی، مدلی با بیشترین پارامترها ایجاد می‌شود. در چنین حالتی، مدل با تغییرات جهشی سعی در پوشش داده‌های حاصل از نمونه و حتی مقدارهای نویز می‌کند. در حالیکه چنین مدلی باید منعکس کننده رفتار جامعه باشد. در اینگونه موارد، اگر مدل رگرسیون بدست آمده، برای پیش‌بینی نمونه دیگری به کار رود، مقدارهای پیش‌بینی شده اصلا مناسب به نظر نخواهند رسید.



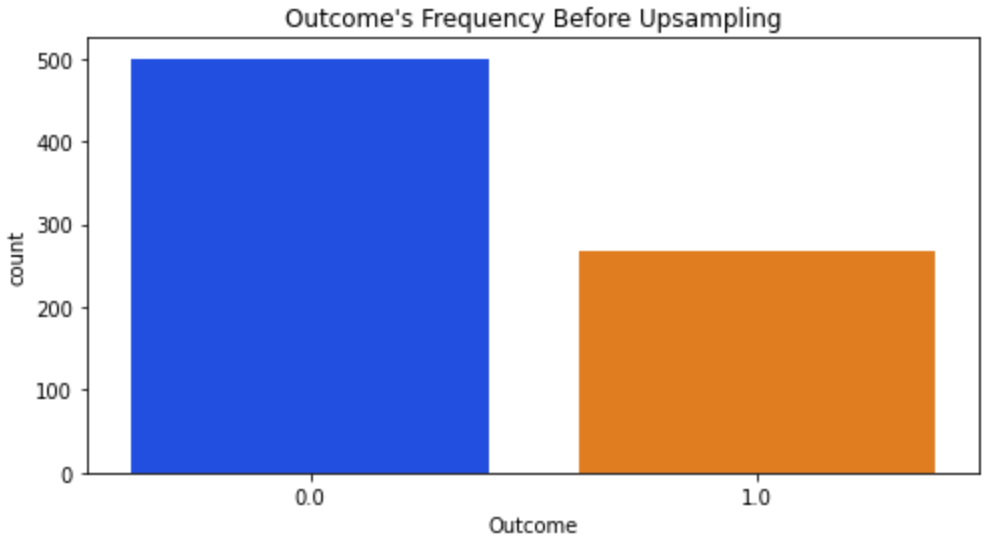
Underfit شدن نیز زمانی رخ می دهد که الگوریتم یک مدلِ خیلی کلی از مجموعه آموزشی به دست می‌آورد. یعنی حتی اگر خودِ داده‌های مجموعه‌ی آموزشی را نیز به این الگوریتم بدهیم، این الگوریتم خطایی قابل توجه خواهد داشت. زمانی که پارامترهای مدل رگرسیونی به صورت کم برازش برآورد می‌شوند، جانب احتیاط حفظ شده و مدل سعی می‌کند با کمترین پارامترها، عمل برازش را انجام دهد. در نتیجه خطای حاصل از این مدل حتی براساس نمونه‌های به کار رفته نیز بسیار زیاد است.

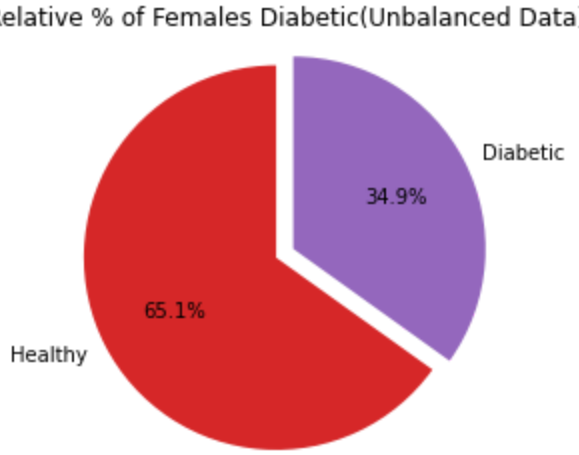


یکی از نشانه‌های مهم برای تشخیص Overfit شدن، تفاوت چشم‌گیر بین دقت روی داده‌های train و test است، به طوریکه مدل با وجود اینکه روی داده‌های train عملکرد خوبی دارد، روی داده‌های test عملکرد ضعیفی از خود نشان می‌دهد. با توجه به نتایج بدست آمده این موضوع در مدل‌های ما وجود ندارد. همچنین اگر دقت مدل حتی روی داده‌های train هم خوب نباشد، می‌تواند نشانگر underfit شدن مدل باشد. در مدل‌های فوق، به دقت‌های قابل قبولی دست یافته‌ایم اما با انجام برخی تغییرات می‌توانیم به دقت‌های بهتری برسیم.

❖ سوال 4

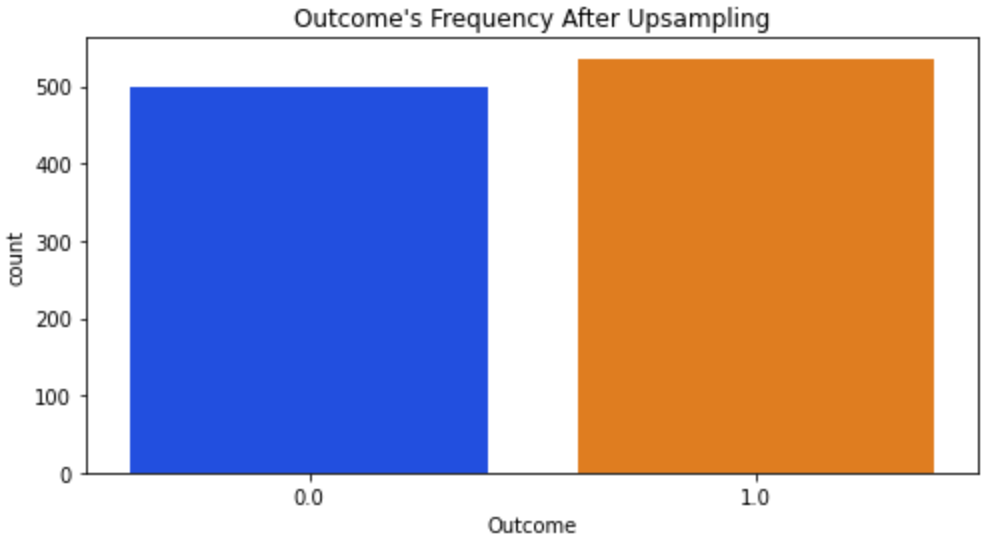
اگر به نتایج بدست آمده در بخش قبل توجه کنیم، متوجه می‌شویم که یکی از عوامل کاهش دقت مدل‌ها، عدم توانایی آن‌ها در پیش‌بینی درست داده‌های مربوط به Outcome = 1 است. این موضوع به دلیل عدم وجود توازن میان تعداد داده‌های دو کلاس است. در تصویر زیر، تعداد داده‌های مربوط به هر کلاس نشان داده شده است.

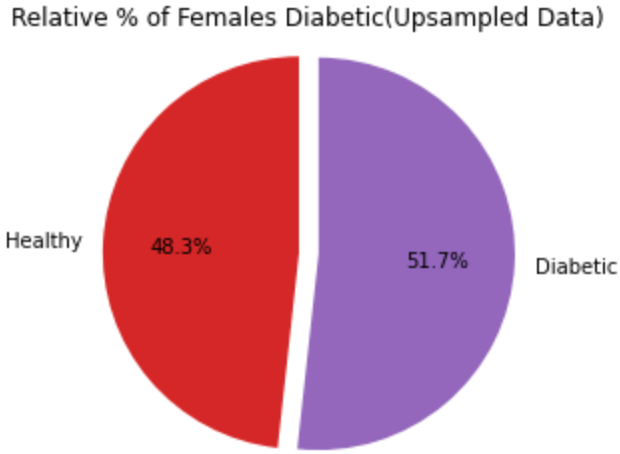




برای رفع این مشکل، یکی از راه‌های متداول، استفاده از Upsampling است. در این روش برای اینکه تعداد داده‌های موجود در کلاس‌ها یا یکدیگر مساوی باشد، داده‌های مربوط به کلاس با تعداد داده‌های کمتر را تکرار می‌کنند. این روش با وجود اینکه داده‌های جدید تولید نمی‌کند و عملا برخی

داده‌ها را چندین بار به مدل می‌دهد، می‌تواند تا حد خوبی مشکلات موجود در داده‌های نامتوازن را برطرف کند. در مسئله‌ی ما نیز، استفاده از این روش، موجب افزایش چشم‌گیر دقت مدل‌ها و پایدار شدن آن‌ها می‌شود.





نتایج بدست آمده پس از Upsampling به صورت زیر می‌باشد:

	precision	recall	f1-score	support
0.0	0.71	0.74	0.72	91
1.0	0.79	0.77	0.78	117
accuracy			0.75	208
macro avg	0.75	0.75	0.75	208
weighted avg	0.76	0.75	0.76	208

Decision Tree

	precision	recall	f1-score	support
0.0	0.89	0.74	0.81	91
1.0	0.82	0.93	0.87	117
accuracy			0.85	208
macro avg	0.86	0.83	0.84	208
weighted avg	0.85	0.85	0.84	208

KNN

	precision	recall	f1-score	support
0.0	0.70	0.87	0.77	91
1.0	0.87	0.71	0.78	117
accuracy			0.78	208
macro avg	0.79	0.79	0.78	208
weighted avg	0.80	0.78	0.78	208

Logistic Regression

همانگونه که مشخص است، نتایج بدست آمده پس از Upsampling به مقدار قابل توجهی بهتر از قبل است.

❖ روش‌های یادگیری جمعی

❖ سوال 1

تابع RandomForestClassifier در کتابخانه‌ی Scikit Learn و در کلاس ensemble قرار دارد. برخی از پارامترهای این تابع عبارتنداز:

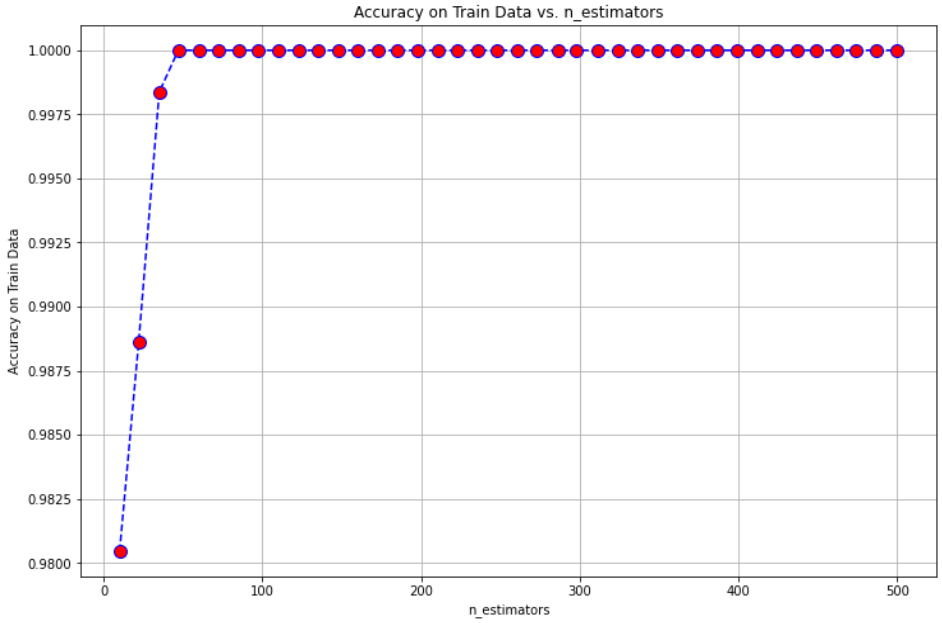
- پارامتر n_estimators: این پارامتر تعداد درخت‌های مورد استفاده در جنگل را مشخص می‌کند. مقدار default این پارامتر برابر 100 است.
- پارامتر criterion: این پارامتر معیار شکستن درخت را مشخص می‌کند. معیارهای پشتیبانی شده توسط این تابع، معیارهای {"gini", "entropy", "log_loss"} می‌باشند. معیارهای log_loss و entropy از آنتروپی و information gain شانون استفاده می‌کنند. gini نیز از Gini Impurity استفاده می‌کند. فرمول Gini Impurity از رابطه‌ی زیر بدست می‌آید.

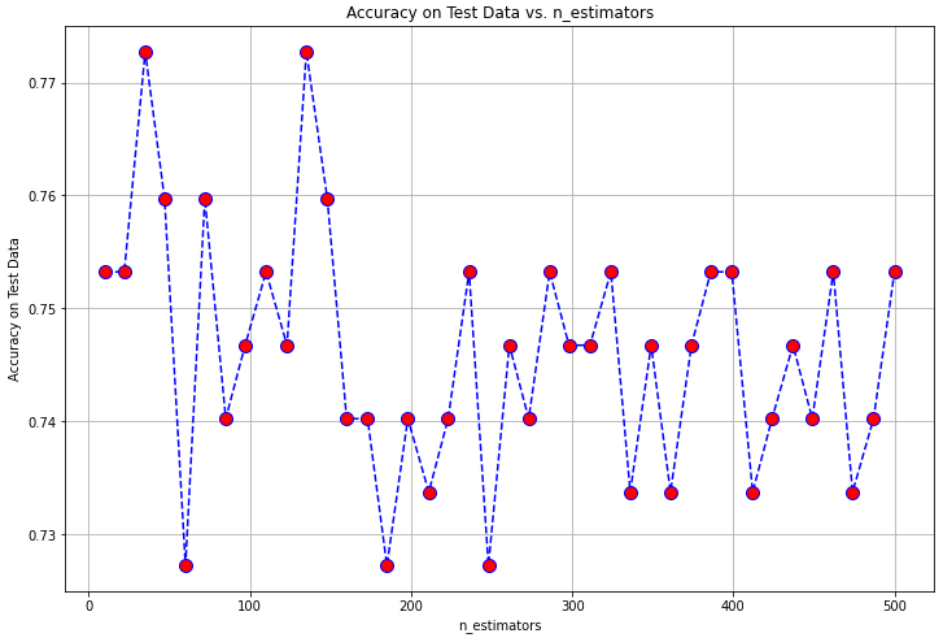
$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

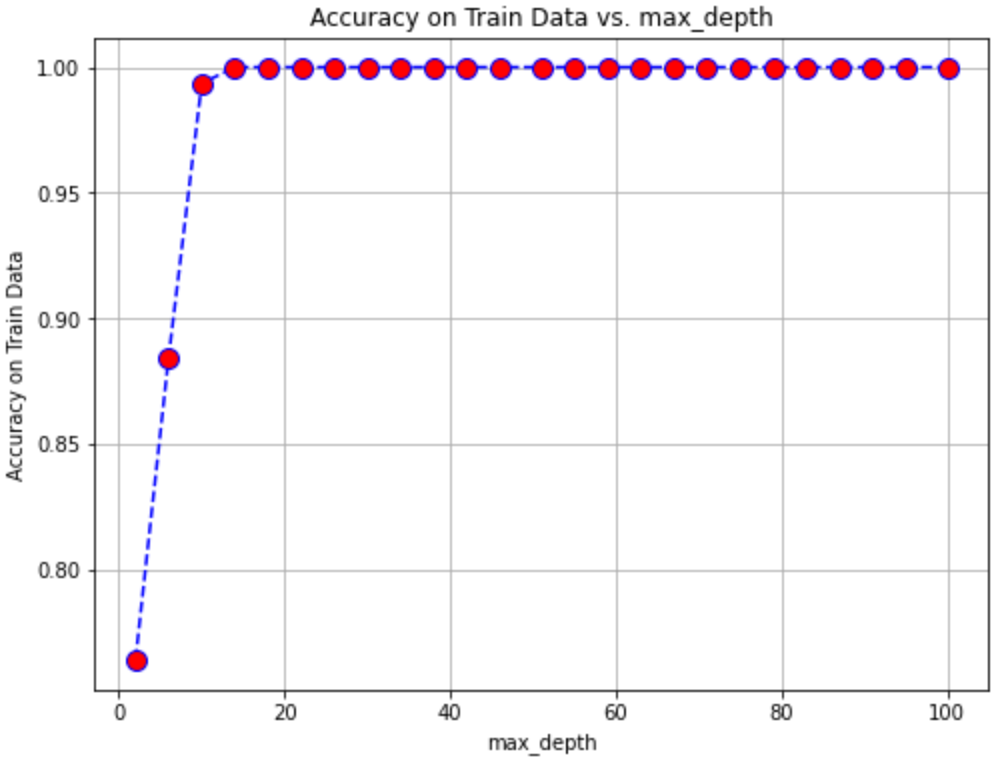
مقادیر ممکن برای Gini Impurity بین 0 و 1 است. اگر مقدار این معیار برابر 1 باشد، یعنی داده‌ها به صورت کاملا تصادفی بین کلاس‌ها تقسیم شده‌اند و اگر برابر 0.5 باشد، یعنی داده‌ها به صورت یکنواخت روی برخی کلاس‌ها تقسیم شده‌اند.

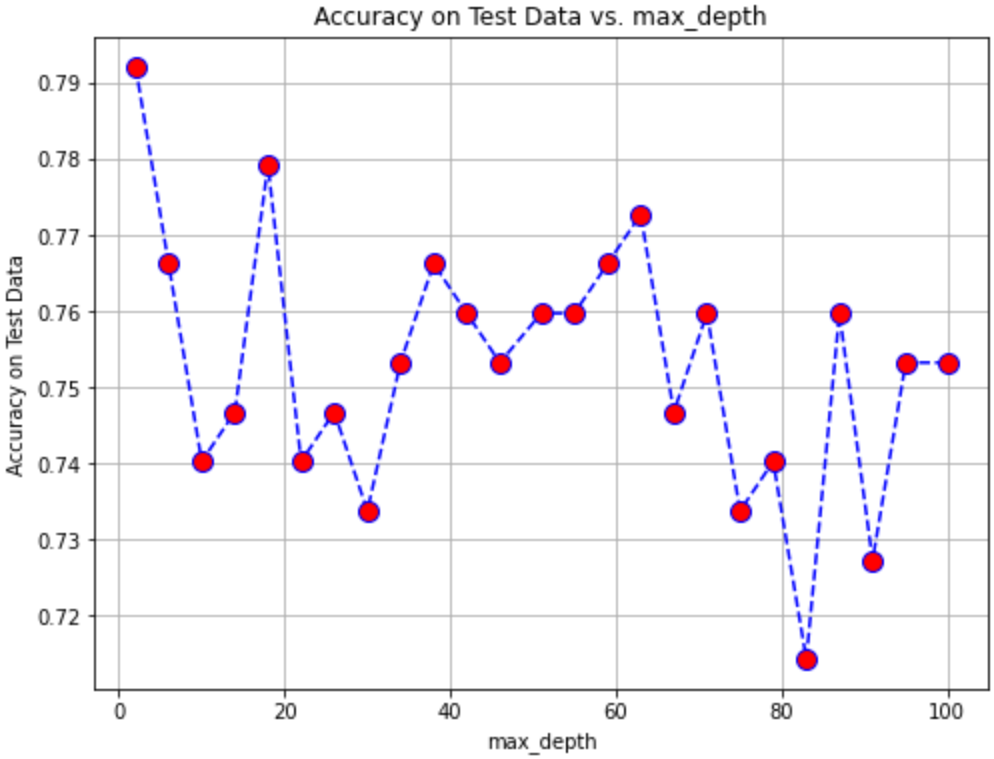
- پارامتر max_depth: این پارامتر حداکثر عمق درخت‌های موجود در جنگل را مشخص می‌کند.
- پارامتر bootstrap: این پارامتر مشخص می‌کند که برای ساخت درخت‌ها از کل داده‌ها استفاده شود یا تنها از نمونه‌های bootstrap.

در ادامه مقادیر مختلفی را برای 2 پارامتر max_depth و n_estimators در نظر می‌گیریم و تاثیر آن‌ها بر دقت مدل را بررسی می‌کنیم. ابتدا برای داده‌های غیرمتوازن خواهیم داشت:

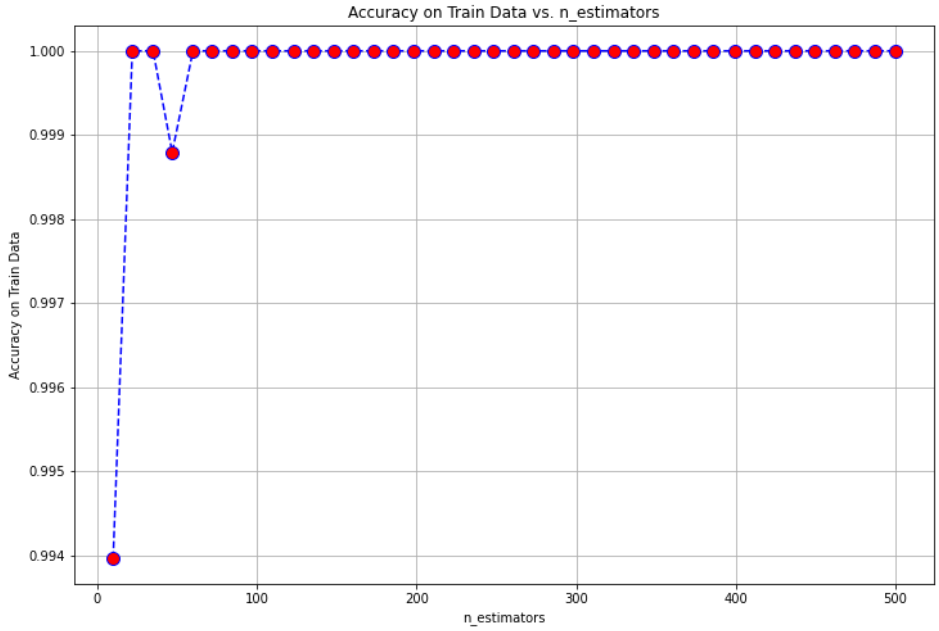

Accuracy on Train Data vs. n_estimators
Accuracy on Train Data
n_estimators
1.0000
0.9975
0.9950
0.9925
0.9900
0.9875
0.9850
0.9825
0.9800
0
100
200
300
400
500

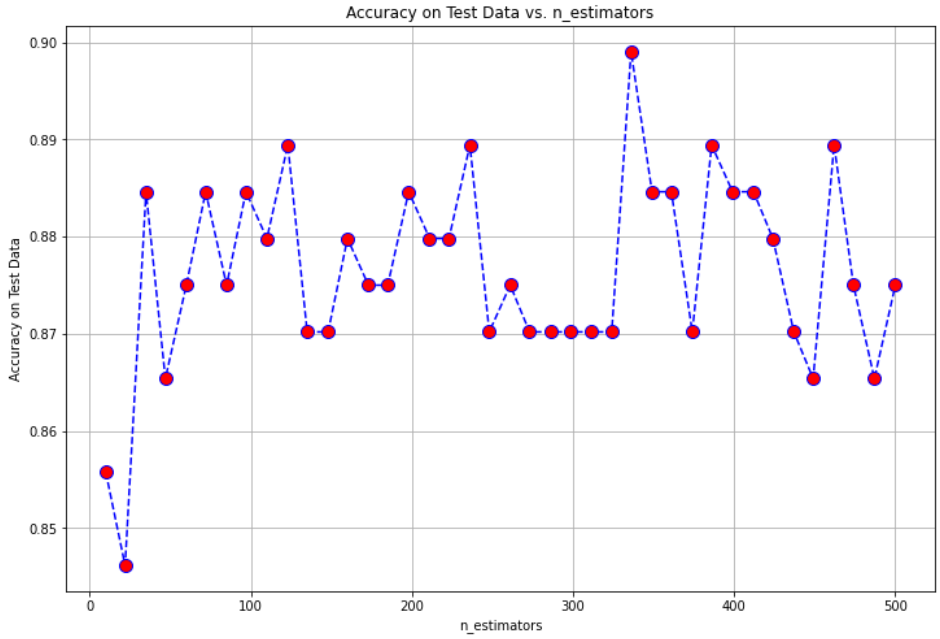

Accuracy on Test Data vs. n_estimators
Accuracy on Test Data
n_estimators
0.77
0.76
0.75
0.74
0.73
0
100
200
300
400
500

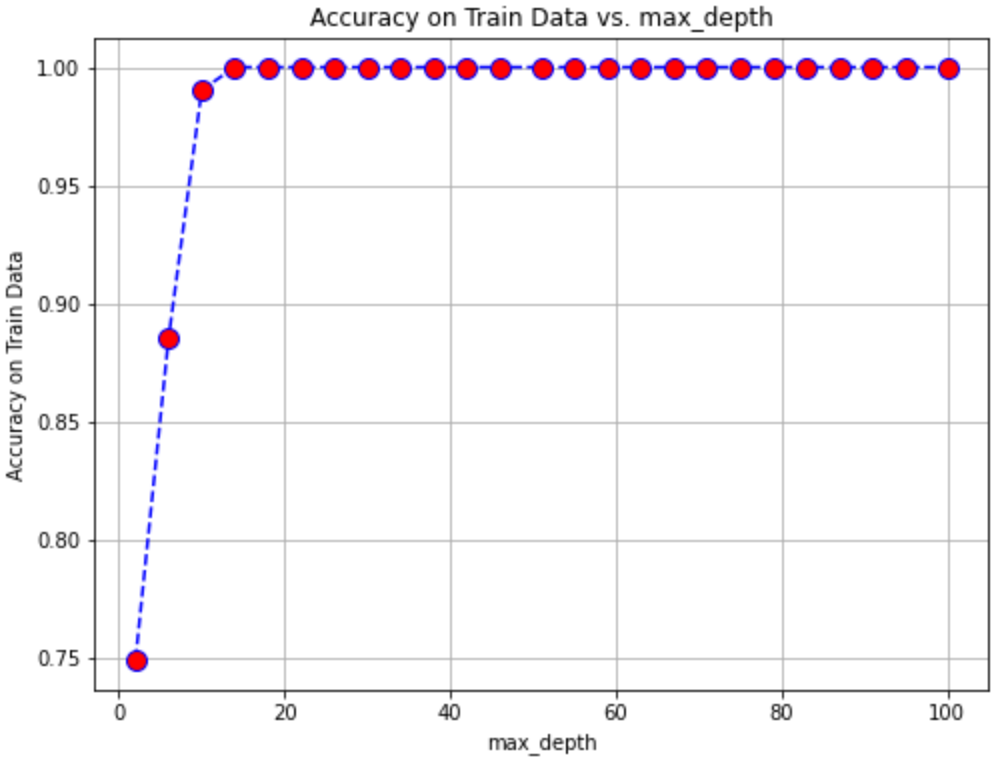

Accuracy on Train Data vs. max_depth
Accuracy on Train Data
max_depth
1.00
0.95
0.90
0.85
0.80
0
20
40
60
80
100

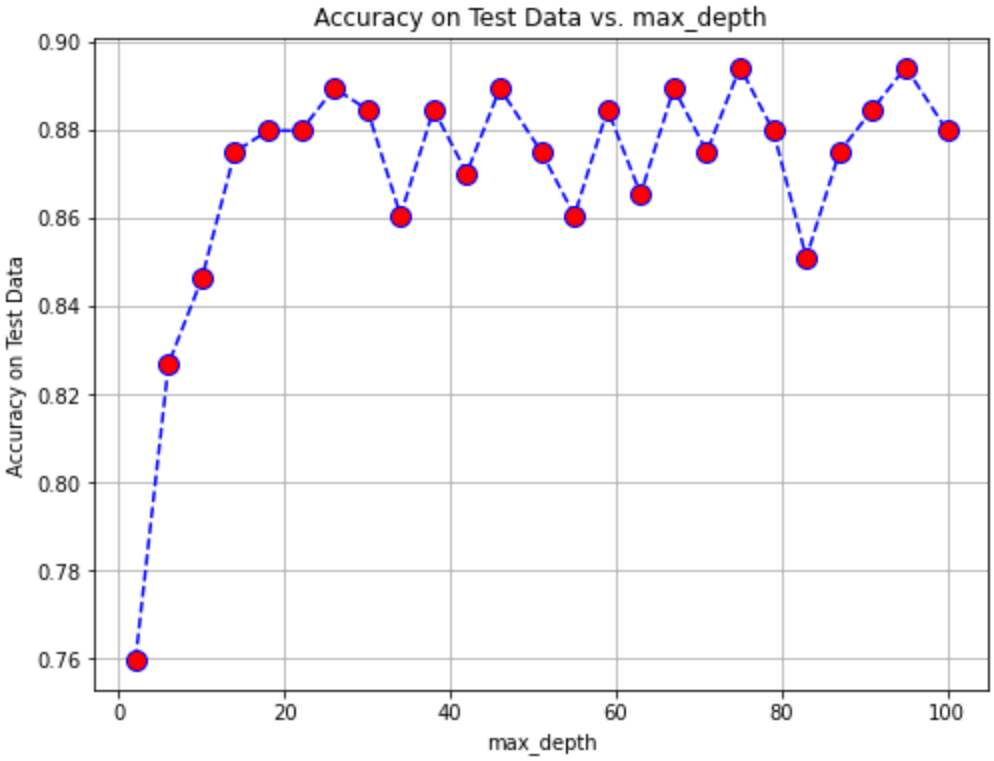

Accuracy on Test Data vs. max_depth
Accuracy on Test Data
max_depth
0.79
0.78
0.77
0.76
0.75
0.74
0.73
0.72
0
20
40
60
80
100

همچنین نتایج برای داده‌های متوازن نیز به صورت زیر خواهد بود:






Accuracy on Train Data vs. max_depth
Accuracy on Train Data
max_depth
1.00
0.95
0.90
0.85
0.80
0.75
0
20
40
60
80
100


Accuracy on Test Data vs. max_depth
Accuracy on Test Data
max_depth
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0
20
40
60
80
100

❖ سوال 2

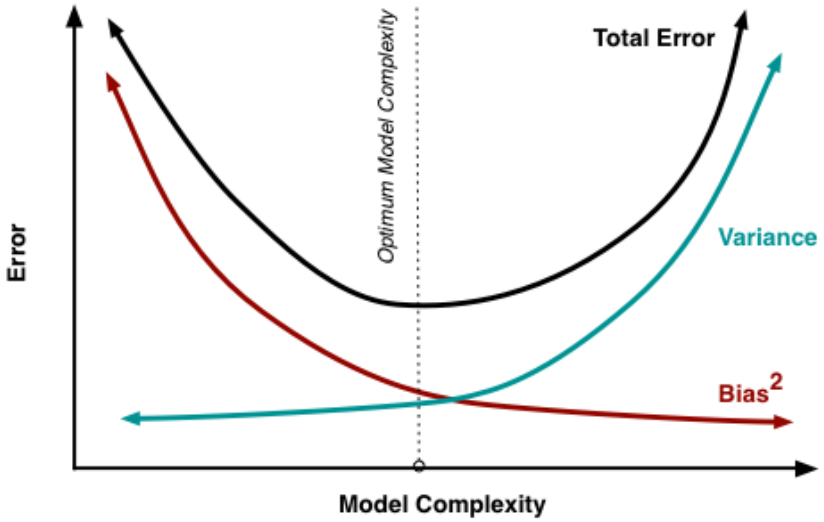
نتایج مربوط به تمام مدل‌ها در شکل زیر قابل مشاهده است:

Train/Test Accuracy %	Imbalanced Data	Balanced Data
Desicion Tree	73/69	74/75
Desicion Tree(AdaBoost)	100/74	100/89
KNN	78/77	100/85
Logistic Regression	78/81	75/78
Random Forest	100/75	100/88
AdaBoost(GridSearch)	100/75	100/88

همانگونه که مشخص است، نتایج مربوط به مدل RandomForest بهتر از نتایج Single Desicion Tree است.

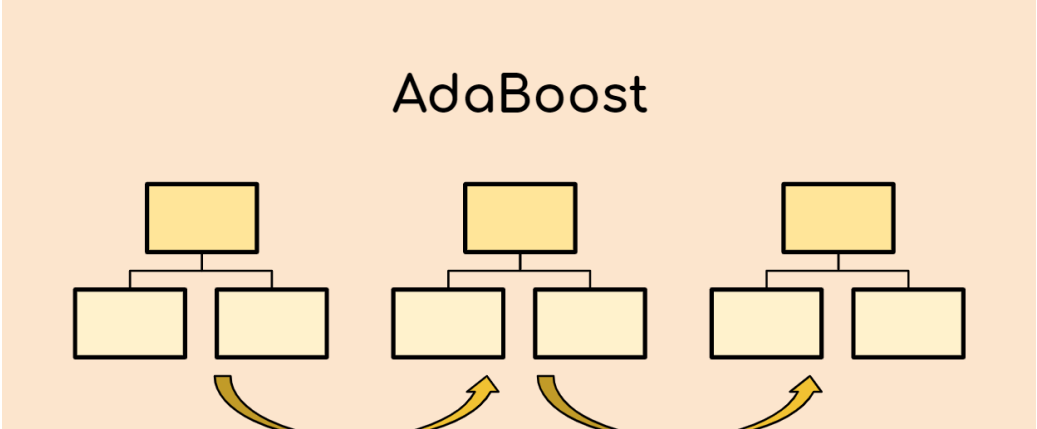
- خطای بایاس: وجود فرضیه‌های مختلف روی مدل و الگوریتم یادگیری منجر به ایجاد خطای اریبی می‌شود. بزرگ بودن اریبی می‌تواند الگوریتم یا مدل آماری را از کشف روابط بین ویژگی‌ها (Features) و متغیر پاسخ (Target Variable) باز دارد. اغلب بزرگ بودن خطای اریبی، منجر به «کم‌برازش» (Underfitting) می‌شود.
- خطای واریانس: حساسیت زیاد مدل با تغییرات کوچک روی داده‌های آموزشی، نشانگر وجود واریانس زیاد است. این امر نشانگر آن است که اگر مدل آموزش داده شده را روی داده‌های آزمایشی به کار گیریم، نتایج حاصل با داده‌های واقعی فاصله زیادی خواهند داشت. متاسفانه افزایش واریانس در این حالت منجر به مدل‌بندی مقادیر نوفه (Noise) شده و به جای پیش‌بینی صحیح، دچار پیچیدگی و مشکل «بیش‌برازش» (Overfitting) می‌شود.

شکل زیر، trade-off موجود بین بایاس و واریانس را به خوبی نمایش می‌دهد:



درخت‌های تصمیم، بایاس پایینی دارند چرا که سعی می‌کنند به بیشترین مقدار ممکن، به داده‌های آموزش overfit شوند. همین overfit شدن باعث می‌شود تا خطای واریانس این مدل‌ها افزایش یابد. Random forest ها این مشکل را حل می‌کنند. در random forest ها، چندین درخت و هر کدام بر روی بخشی از داده آموزش می‌بینند. از آنجاییکه درخت‌ها بر روی داده‌های مختلف آموزش می‌بینند، می‌توان گفت که از یکدیگر مستقل عمل می‌کنند و هر کدام تصمیم‌گیری خود را بر اساس ویژگی‌های مختلفی انجام می‌دهند. استفاده‌ی همزمان از این درخت‌ها موجب پایدارتر شدن مدل و افزایش دقت آن می‌شود. نتایج بدست آمده در مسئله‌ی ما نیز این موضوع را تایید می‌کند.

یکی دیگر از روش‌های یادگیری جمعی، AdaBoost است. AdaBoost یا Adaptive Boost یک الگوریتم طبقه‌بندی یادگیری ماشین نسبتا جدید است. این الگوریتم بسیاری از یادگیرندگان ضعیف (درخت‌های تصمیم‌گیری) را ترکیب کرده و آن‌ها را به یک یادگیرنده قوی تبدیل می‌کند. بنابراین، الگوریتم آن روش‌های گردآوری و تقویت را برای ایجاد یک پیش‌بینی‌کننده پیشرفته کاهش می‌دهد.



AdaBoost شبیه جنگل‌های تصادفی است به این معنی که پیش‌بینی‌ها از بسیاری از درخت‌های تصمیم‌گیری گرفته می‌شوند. با این حال، سه تفاوت اصلی وجود دارد که AdaBoost را منحصر به فرد می‌سازد:

اول، AdaBoost به جای درخت، جنگلی از کنده درختان ایجاد می‌کند. کنده درختی است که تنها از یک گره و دو برگ تشکیل شده‌است (مانند تصویر بالا).

دوم، کنده‌هایی که ایجاد می‌شوند وزن مساوی در تصمیم نهایی (پیش‌بینی نهایی) ندارند. کنده‌هایی که خطای بیشتری ایجاد می‌کنند در تصمیم نهایی کم‌تر نقش خواهند داشت.

در نهایت ترتیبی که در آن کنده ایجاد می‌شود مهم است چون هر کنده قصد دارد خطاهایی که کنده قبلی ایجاد کرده‌است را کاهش دهد.

در AdaBoost، مدلها به صورت سلسله مراتبی آموزش داده می شوند و نحوه آموزش مدل‌ها به این صورت است که هر مدل هدفش برطرف کردن ایرادات(خطای) مدل‌های قبل تر از خودش است و براساس میزان خطایی که هر مدل بدست می آورد، یک وزنی به آن اختصاص می باید که از این وزن‌ها در پروسه تست، به عنوان میزان اهمیت رای مدل‌ها استفاده می‌شود.

همانطور که اشاره شد، مدل‌ها به صورت سلسله مراتبی آموزش می‌بینند، یعنی اول مدل 1 آموزش می‌بیند، سپس مدل 2، مدل 3 و به همین ترتیب تا آخرین مدل.

هر مدل هدفش این است که خطای مدل‌های قبلی را جبران کند، یعنی اینکه تلاش می‌کند تا نمونه‌هایی که مدل قبلی نتوانسته به درستی طبقه بندی کند را به درستی طبقه بندی کند.

با این رویکرد ساده ، آدابوست با کمک مدلهای ضعیف یک مسئله پیچیده را حل میکند و یا به عبارتی توان مدلهای ضعیف را تقویت میکند تا بتوانند یک مسئله پیچیده را به راحتی حل کنند.

برای این پروژه، از الگوریتم AdaBoost نیز استفاده شده است که نتایج آن نیز گزارش شده است.