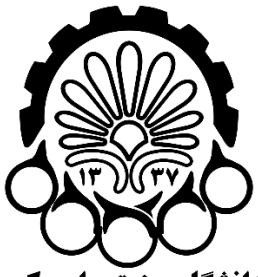


دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش تکلیف دوم درس داده‌کاوی

استاد درس: دکتر احمدی

تهیه کنندگان:

سید نیما محمودیان

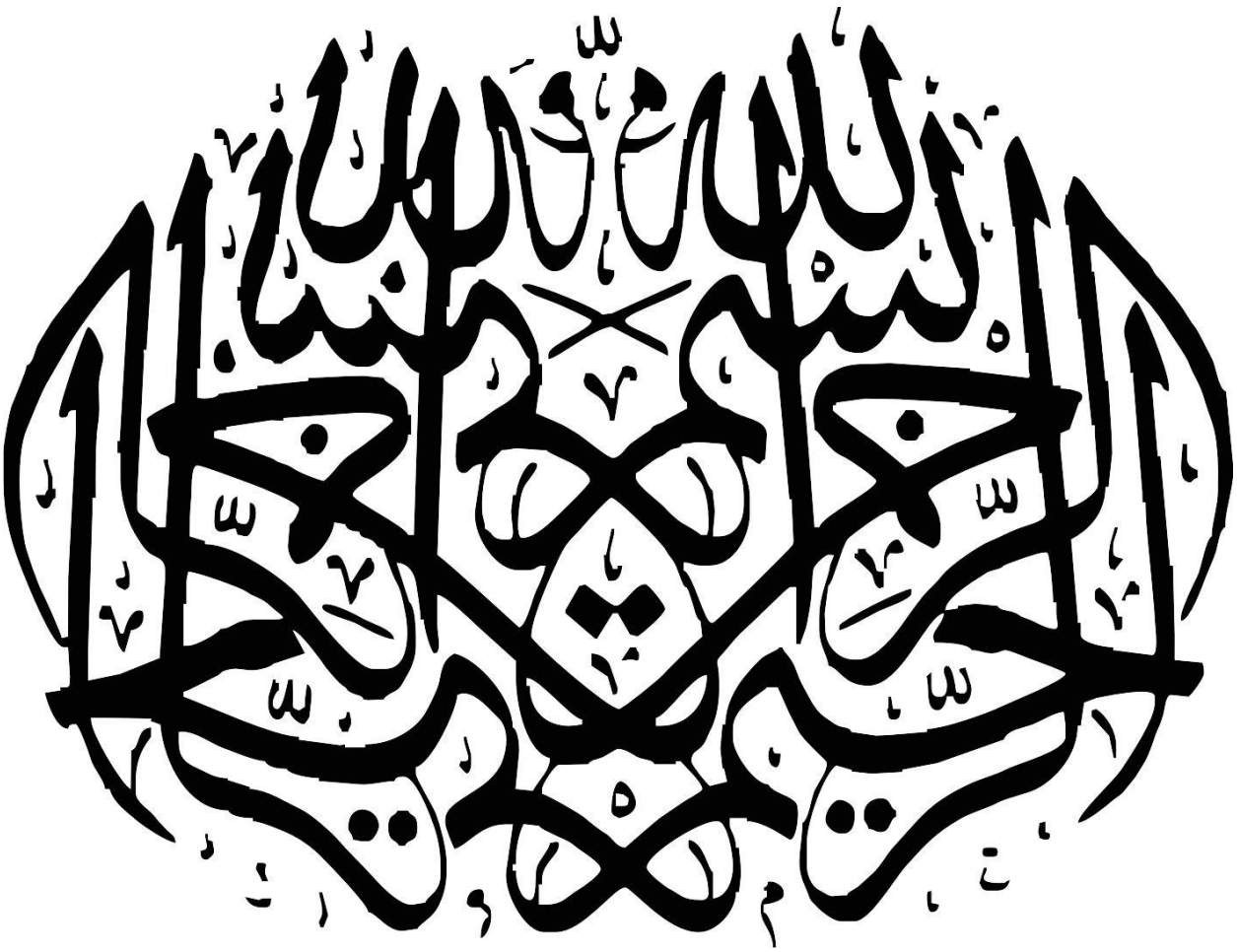
۴۰۲۱۲۵۰۰۵

معصومه بهبهانی‌زاده

۴۰۲۱۹۱۰۰۳

محسن فراغه

۴۰۱۲۲۵۰۰۲



فهرست مطالب

فهرست شکل‌ها

فهرست جدول‌ها

چکیده

در این مطالعه تحلیل بر روی مجموعه داده[1] دیابت انجام شده است. داده‌های مورد بررسی از UCI[2] گرفته شده است. پس از معرفی مجموعه داده، فرآیند آماده سازی داده‌ها و بررسی مقادیر گمشده و پرت را در دستور کار داریم، سپس به استاندارد سازی داده‌ها می‌پردازیم. در ادامه کار با استفاده از روش جست و جوی رو به جلو ویژگی‌های کاربردی تر را از بین ویژگی‌های موجود انتخاب می‌کنیم. سپس معیارهای دقت، حساسیت و اختصاصیت را برای این انتخاب ویژگی‌ها به دست می‌آوریم. در ادامه با استفاده از روش KNN [3] دسته بندی داده‌ها را انجام می‌دهیم. معیارهای نام برده در بالا را دوباره برای مدل ارائه شده محاسبه می‌کنیم و تمامی مراحل را با استفاده از روش دسته‌بندی بیزی در دستور کار خواهیم داشت.

مسئله دوم مورد بررسی در این پژوهش، شامل بررسی و به دست آوردن مرز تصمیم بین دو کلاس با استفاده از روش‌های نزدیکترین همسایگی و پرسپترون می‌باشد.

۱-معرفی مجموعه داده

مجموعه داده مورد بررسی، اطلاعات ۷۶۸ فرد را نمایش می‌دهد، هدف مجموعه داده این است که بر اساس اندازه‌گیری‌های تشخیصی خاص موجود در مجموعه داده، پیش‌بینی تشخیصی اینکه آیا بیمار مبتلا به دیابت است یا خیر را انجام دهد. در جمع‌آوری این مجموعه داده محدودیت‌هایی نیز ایجاد شده است. این مجموعه داده اطلاعات زنان بالای ۲۱ سال از میراث هندی پیما هستند. توضیحات هر یک از ویژگی مورد بررسی عبارتند از:

۱. بارداری: تعداد دفعاتی که فرد باردار شده است.
۲. گلوکز: غلظت کلوگز پلاسما در یک آزمایش دو ساعته
۳. فشار خون: فشار خون دیاستولیک
۴. ضخامت پوست: ضخامت چین‌های پوستس سه سر بازو بر حسب میلی متر
۵. انسولین: سرم دو ساعته انسولین
۶. شاخص توده بدنی BMI4: وزن بر حسب کیلوگرم / قد بر حسب متر به توان دو
۷. DiabetesPedigreeFunction: تابعی از سابقه خانوادگی فرد از ابتلا به دیابت
۸. سن: سن افراد شرکت کننده

۲- فراخوانی داده‌ها

۲-۱ سوال مساله یک

مجموعه داده pima-indian-diabetes از مخزن دادههای استاندارد دانشگاه کالیفرنیا UCI Machine Learning (Repository)مورد نظر است. سوالات زیر را پاسخ دهید.

---

[1] Data Set

[2]مخزن داده‌های استاندارد دانشگاه کالیفرنیا

[3] K-Nearest_Neighbor

[4]Body mass index

الف) بعد از پیش پردازش داده‌ها، با استفاده از روش جست و جوی رو به جلو، بهترین ترکیب ویژگی‌ها را به دست آورید. از روش نزدیکترین همسایگی جهت دسته بندی استفاده کنید. بدین منظور از روش Hold out استفاده کنید. مقدارهای دقت، حساسیت و اختصاصیت را گزارش کنید.

۲-۲ کدهای اولیه و فراخوانی داده‌ها

در ابتدا کتابخانه‌های مورد نیاز را فراخوانی می‌کنیم. در شکل ۱ بخشی از آن‌ها قابل مشاهده می‌باشند.

```
import pandas as pd
import numpy as np
import math
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer,SimpleImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.model_selection import cross_val_score
from sklearn.impute import IterativeImputer
from sklearn.metrics import confusion_matrix,recall_score,precision_score,f1_score,classification_report,accuracy_score,
from sklearn.impute import IterativeImputer,SimpleImputer
from sklearn.impute import KNNImputer
from sklearn.model_selection import train_test_split, GridSearchCV
from imblearn.over_sampling import SMOTE
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix,recall_score,precision_score,f1_score,classification_report,accuracy_score
from sklearn.model_selection import cross_val_score, KFold, StratifiedKFold
from sklearn.metrics import accuracy_score
from mlxtend.feature_selection import SequentialFeatureSelector
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.impute import KNNImputer
```

شکل ۱:کد فراخوانی کتابخانه‌ها

با توجه به این مهم، که برای بررسی این مجموعه داده فایل CSV[5] مربوط را دانلود کرده‌ایم، برای فراخوانی این مجموعه داده مطابق شکل ۲ پیش خواهیم رفت.

```
df=pd.read_csv("diabetes.csv")
df
```

شکل ۲:کد فراخوانی مجموعه داده

در جدول ۱ یک نمای کلی از مجموعه داده را نمایش می‌دهیم:

[6]Comma-Separated Values

جدول ۱: نمای کلی مجموعه داده

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

۳-۲ تحلیل توصیفی داده‌ها

در گذشته همانگونه که در شکل ۲ مشاهده کردید، یک قالب داده تحت عنوان "df" از کل مجموعه داده‌ای که در اختیار داشتیم، ایجاد کردیم. برای تسلط بیشتر بر روی مجموعه داده اقدامات زیر را انجام می‌دهیم:

df.head : برای مشاهده پنج سطر اول از مجموعه داده استفاده می‌شود.

df.tail: برای مشاهده پنج سطر آخر از مجموعه داده استفاده می‌شود.

df.nunique: تعداد مقادیر منحصر به فرد را در هر ستون از چارچوب داده برمیگرداند. این به ما نشان می‌دهد که در هر ستون چند مقدار مختلف وجود دارد و داده‌ها چقدر متنوع هستند. کد و خروجی این بخش در شکل ۳ نمایش داده شده است.

df.describe: تعداد داده‌های هر ستون، مقادیر میانگین، انحراف معیار، کمترین و بیشترین مقدار هر ستون و چارک‌ها را نمایش می‌دهد. کد و جدول خروجی در جدول ۲ قابل مشاهده است.

df.info : خلاصه‌ای از داده‌ها شامل فهرست، نام ستون‌ها، انواع داده‌ها، مقادیر غیرتهی، میزان استفاده از حافظه و سایر اطلاعات را چاپ می‌کند. این یک نمای کلی از چارچوب داده و ویژگی‌های آن به ما می‌دهد. کد و خروجی این بخش در شکل ۵ نمایش داده شده است

df.shape: تعداد ردیف‌ها و ستون‌ها را به ما برمی‌گرداند.

df.dtype: نوع هر داده را مشخص می‌کند. کد و خروجی این بخش در شکل۴ نمایش داده شده است.

در ادامه برای شناخت بهتر مجموعه داده و ارتباط دو به دو هر یک از ویژگی‌ها با یکدیگر نمودارهای زیر را رسم کرده‌ایم:

نمودار Pairplot نشان داده شده در شکل ۶ که به عنوان ماتریس پراکندگی نیز شناخته می شود، ماتریسی از نمودارها می‌باشد که امکان تجسم رابطه بین هر جفت متغیر در یک مجموعه داده را فراهم می‌کند. هر دو نمودار هیستوگرام و پراکندگی را ترکیب می کند و یک نمای کلی منحصر به فرد از توزیع‌ها و همبستگی‌های مجموعه داده ارائه می‌دهد.

```
df.nunique()
```

```
Pregnancies                 17
Glucose                    136
BloodPressure               47
SkinThickness               51
Insulin                    186
BMI                        248
DiabetesPedigreeFunction   517
Age                         52
Outcome                      2
dtype: int64
```

شکل ۳: خروجی کد df.nunique

```
df.dtypes
```

```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

شکل ۴: خروجی کد df.dtype

جدول ۲: خروجی کد df.describe

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
df.info()
```

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

شکل ۵: خروجی کدdf.nunique

نمودار Heatmap[6] ترسیم شده، همانگونه که در شکل ۷ پیداست، برای نشان دادن روابط بین دو متغیر استفاده می‌شود که یکی در هر محور ترسیم شده است. با مشاهده چگونگی تغییر رنگ سلول‌ها در هر محور، می‌توانید مشاهده کنید که آیا الگوهایی در ارزش یک یا هر دو متغیر وجود دارد یا خیر.

۲-۴ پیش پردازش داده‌ها

۲-۴-۱پیدا کردن داده‌های تکراری

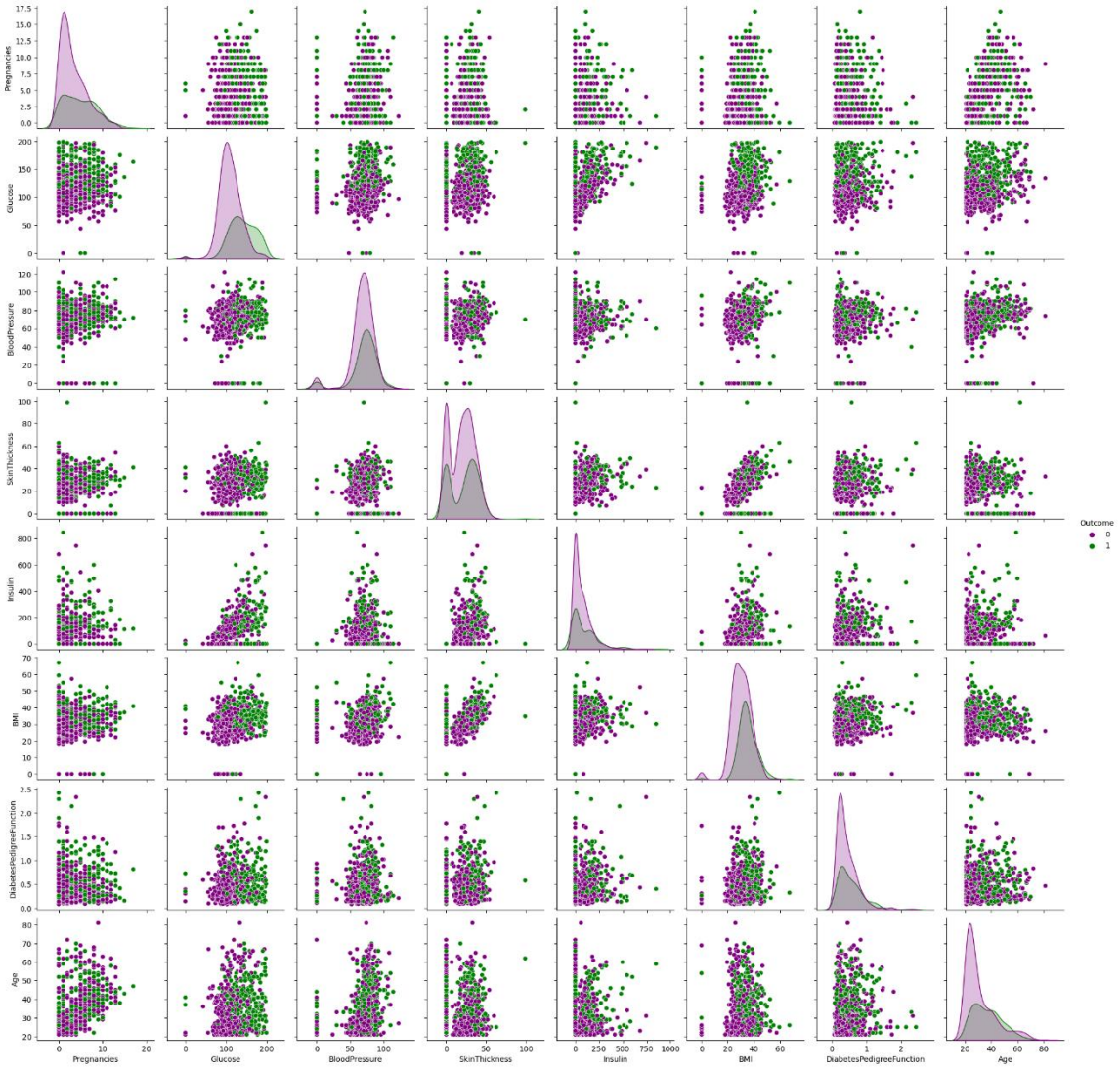
برای بررسی مجموعه داده باید اطمینان حاصل کنیم که هیچ داده تکراری در مجموعه داده ما وجود ندارد. اگر ستون‌های کاملا مشابهی در مجموعه داده ما وجود دارد، باید حذف شوند. که در مجموعه داده مورد بررسی ما طبق کد دستوری در شکل۸ ، هیچ دو ستون مشابهی موجود نمی‌باشد

۲-۴-۲ پیدا کردن مقادیر گمشده

یکی از مهم‌ترین چالش‌هایی که در آماده سازی داده‌ها با آن سر و کار داریم، داشتن مقادیر گمشده می‌باشد. مقادیر گمشده ممکن است در پیش‌بینی‌های آینده یا در روند کار برای ما مشکلاتی را ایجاد کنند، از این رو شناسایی آن‌ها حائز اهمیت است. کداجرا شده در شکل ۹ تعداد داده‌های گم شده را نمایش می‌دهد:

---

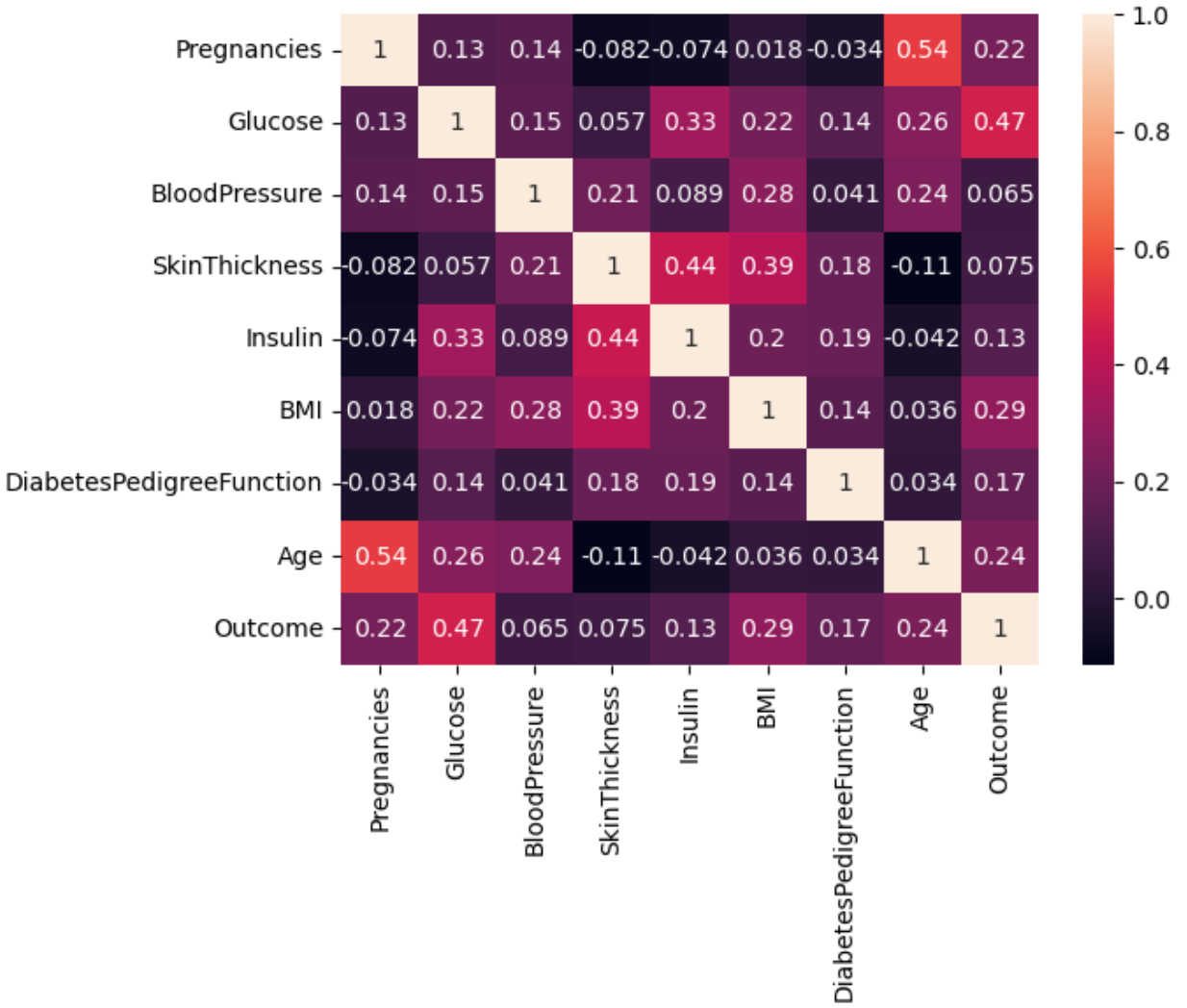
[6] نقشه حرارتی



شکل ۶: Pairplot

همانگونه که در شکل ۱۰ مشاهده می‌شود، هیچگونه مقدار گمشده‌ای در این مجموعه داده موجود نمی‌باشد. اما نکته قابل تامل در این بخش شناسایی داده‌هایی با مقدار " ؟ " و یا " ۰ " می‌باشد. همان‌طور که متوجه شده‌اید با کد موجود در شکل ۱۰ تنها مقادیری که وجود ندارند، مشخص می‌شود. این در حالی است که گاهی وجود " ۰ " برای یک شاخص معنی ندارد و حضور " ؟ " نیز در مجموعه داده به معنای عدم وجود مقدار است.

همانگونه که مشاهده می‌شود برای شاخص‌های بارداری، گلوکز، فشار خون، ضخامت پوست، انسولین، BMI و خروجی مجموعه داده، مقادیر صفر داریم. در این بین برای شاخص‌های گلوکز، فشار خون، ضخامت پوست، انسولین و BMI ، مقدار صفر، مقداری نامعتبر و نشدنی است، پس نتیجه می‌گیریم که این مقادیر در اصل ناموجود یا گمشده بوده‌اند که با جای خالی گذاشتن، مقدارصفر را جایگزین کرده‌اند.



شکل ۷: نمودار Heatmap

```
df.duplicated().sum()
```

```
0
```

شکل ۸: مشخص کردن داده¬های تکراری

پس از شناسایی داده‌های گمشده، شناسایی داده‌های پرت (نویز[۷]) را در دستور کار خواهیم داشت.

[7] Noise

```
df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

شکل ۹: نمایش مقادیر گمشده

```
for i in df_ADD.columns:
    print(i,len(df_ADD[df[i] == '?']))
```

```
Pregnancies 0
Glucose 0
BloodPressure 0
SkinThickness 0
Insulin 0
BMI 0
DiabetesPedigreeFunction 0
Age 0
Outcome 0
```

شکل ۱۰: نمایش تعداد مقادیر " ؟"

همانگونه که مشاهده می‌شود، هیچگونه داده‌ای با مقدار " ؟ " وجود ندارد. حال تعداد " ۰ " را نیز در شکل ۱۱ بررسی می‌کنیم.

۲-۴-۳ پیدا کردن مقادیر پرت

نویز، داده‌هایی را شامل می‌شود که به هر دلیلی دارای مقادیر نادرست یا غلط باشند. مانند عکس‌های تار در دوربین ثبت تخلف، تفاوت در واحد پول چند ویژگی مختلف

داده‌های پرت نیز به داده‌هایی گفته می‌شود که در فاصله‌ی غیرعادی از سایر مقادیر داده در یک نمونه‌ی تصادفی از یک جمعیت مشاهده می‌شود.

شناسایی نویزها و داده‌های پرت امری حیاتی و تاثیرگذار در مبحث آماده سازی داده‌ها می‌باشد. از این رو باید در صدد رفع و شناسایی علت آن‌ها باشیم. روش‌های شناسایی این مشکلات عبارتند از:

- شناسایی نقطه پرت بر اساس مفهوم آماری پراکندگی
- استفاده از روش‌های خوشه بندی
- گسسته کردن داده‌ها

```
for i in df_ADD.columns:
    print(i,len(df_ADD[df[i] == 0]))
```

```
Pregnancies 111
Glucose 5
BloodPressure 35
SkinThickness 227
Insulin 374
BMI 11
DiabetesPedigreeFunction 0
Age 0
Outcome 500
```

شکل ۱۱: نمایش مقادیر " ۰ "

در این پژوهش، همانگونه که در شکل ۱۲ قابل مشاهده است، برای شناسایی مقادیر پرت از روش شش سیگما [۸] ، استفاده کرده‌ایم. این روش مقادیر مثبت بیشتر از ۳ سیگما و مقادیر منفی کمتر از ۳ سیگما را داده پرت شناسایی می‌کند.

```
temp = ['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']

df_ADD2 = df_ADD.copy()

for i in temp:
    lower_limit = df_ADD[i].mean() -3*df_ADD[i].std()
    upper_limit = df_ADD[i].mean() +3*df_ADD[i].std()
    outlier = df_ADD[(df_ADD[i] > upper_limit) | (df_ADD[i] < lower_limit)]
    print(i + ':')
    display( outlier)
    print('\n')
```

شکل ۱۲: کد مربوط به نمایش مقادیر پرت

در جدول‌های ۳،۴، ۵، ۶، ۷، ۸، ۹ و ۱۰ مقادیر پرت هر شاخص نمایش داده شده است:

جدول ۳: مقادیر پرت شاخص بارداری

Pregnancies:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
88	15	136	70	32	110	37.1	0.153	43	1
159	17	163	72	41	114	40.9	0.817	47	1
298	14	100	78	25	184	36.6	0.412	46	1
455	14	175	62	30	0	33.6	0.212	38	1

[8] Six Sigma

جدول ۴: مقادیر پرت شاخص گلوکز

Glucose:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
75	1	0	48	20	0	24.7	0.140	22	0
182	1	0	74	20	23	27.7	0.299	21	0
342	1	0	68	35	0	32.0	0.389	22	0
349	5	0	80	32	0	41.0	0.346	37	1
502	6	0	68	41	0	39.0	0.727	41	1

جدول ۵ : مقادیر پرت شاخص فشار خون

BloodPressure:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
7	10	115	0	0	0	35.3	0.134	29	0
15	7	100	0	0	0	30.0	0.484	32	1
49	7	105	0	0	0	0.0	0.305	24	0
60	2	84	0	0	0	0.0	0.304	21	0
78	0	131	0	0	0	43.2	0.270	26	1
81	2	74	0	0	0	0.0	0.102	22	0
172	2	87	0	23	0	28.9	0.773	25	0
193	11	135	0	0	0	52.3	0.578	40	1
222	7	119	0	0	0	25.2	0.209	37	0

جدول ۶: مقادیر پرت شاخص ضخامت پوست

SkinThickness:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
579	2	197	70	99	0	34.7	0.575	62	1

جدول ۷: بخشی از مقادیر پرت شاخص انسولین

Insulin:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
8	2	197	70	45	543	30.5	0.158	53	1
13	1	189	60	23	846	30.1	0.398	59	1
111	8	155	62	26	495	34.0	0.543	46	1
153	1	153	82	42	485	40.6	0.687	23	0
186	8	181	68	36	495	30.1	0.615	60	1
220	0	177	60	29	478	34.6	1.072	21	1
228	4	197	70	39	744	36.7	2.329	31	0
247	0	165	90	33	680	52.3	0.427	23	0
286	5	155	84	44	545	38.7	0.619	34	0

جدول ۸: مقادیر پرت شاخص BMI

BMI:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
9	8	125	96	0	0	0.0	0.232	54	1
49	7	105	0	0	0	0.0	0.305	24	0
60	2	84	0	0	0	0.0	0.304	21	0
81	2	74	0	0	0	0.0	0.102	22	0
145	0	102	75	23	0	0.0	0.572	21	0
177	0	129	110	46	130	67.1	0.319	26	1
371	0	118	64	23	89	0.0	1.731	21	0
426	0	94	0	0	0	0.0	0.256	25	0

جدول ۹: مقادیر پرت شاخص DiabetesPedigreeFunction

DiabetesPedigreeFunction:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
4	0	137	40	35	168	43.1	2.288	33	1
45	0	180	66	39	0	42.0	1.893	25	1
58	0	146	82	0	0	40.5	1.781	44	0
228	4	197	70	39	744	36.7	2.329	31	0
330	8	118	72	19	0	23.1	1.476	46	0
370	3	173	82	48	465	38.4	2.137	25	1
371	0	118	64	23	89	0.0	1.731	21	0
395	2	127	58	24	275	27.7	1.600	25	0
445	0	180	78	63	14	59.4	2.420	25	1

جدول ۱۰: مقادیر پرت شاخص سن

Age:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
123	5	132	80	0	0	26.8	0.186	69	0
453	2	119	0	0	0	19.6	0.832	72	0
459	9	134	74	33	60	25.9	0.460	81	0
666	4	145	82	18	0	32.5	0.235	70	1
684	5	136	82	0	0	0.0	0.640	69	0

برای رفع مشکل وجود داده‌های پرت، چندین راه حل وجود دارد:

- حذف داده‌های پرت
- جایگذاری داده‌های پرت با مقدار میانگین
- جایگذاری داده‌های پرت با مقادیر روی حد بالا و پایین

در این بخش، مطابق شکل ۱۳، داده‌های بیشتر از حد را با مقادیر روی حد بالا و کمتر را با مقادیر روی حد پایین جایگذاری می‌کنیم.

```
    cleaned_df = df.copy()

for col in df.columns:

        lower_limit = df_ADD[col].mean() - 3 * df_ADD[col].std()
        upper_limit = df_ADD[col].mean() + 3 * df_ADD[col].std()



        cleaned_df[col] = df_ADD[col].apply(lambda x: lower_limit if x < lower_limit else
                                            (upper_limit if x > upper_limit else x))

cleaned_df = df_ADD
df_ADD
```

شکل ۱۳: جایگذاری داده‌های پرت با مقادیر حد بالا و پایین

در این بخش به رفع مشکل داده‌های گمشده می‌پردازیم. با توجه به جایگذاری داده‌های پرت با مقادیر حد بالا و پایین، در ابتدا مجددا تعداد صفرهای موجود در هر شاخص را بررسی می‌کنیم، زیرا ممکن است تعدادی از صفرها حذف و یا به تعداد آن‌ها افزوده شده باشد. شکل ۱۴، این مهم را نمایش می‌دهد.

```
for i in df_ADD.columns:
    print(i,len(df_ADD[df[i] == 0]))
```

```
Pregnancies 111
Glucose 5
BloodPressure 35
SkinThickness 227
Insulin 374
BMI 11
DiabetesPedigreeFunction 0
Age 0
Outcome 500
```

شکل ۱۴: نمایش مجدد تعداد مقادیر " ۰ "

با توجه به آنچه که از خروجی شکل ۱۴ به دست آمده است، متوجه می‌شویم که شاهد تغییر خاصی در مقادیر " ۰ " نیستیم. حال مطابق شکل ۱۵، تمامی مقادیر " ۰ " را با مقدار nan جایگذاری می‌کنیم.

از این بخش به بعد، باید به دنبال جایگذاری داده‌های گمشده باشیم، روش‌های مقابله با این مشکل عبارتند از :

- حذف مقادیر گمشده
- جایگذاری آن‌ها با میانگین هر شاخص
- پر کردن داده‌ها با مقادیر قبلی یا بعدی
- پر کردن با استفاده از مقادیر مشابه
- پر کردن با استفاده از مدل‌های پیش‌بینی

در این بخش، با استفاده از روش‌های Iterative، Mean و Single به رفع این مشکل پرداخته ایم. روش IterativeImputer در ماشین لرنینگ یک روش پر کردن مقادیر گمشده در داده‌ها است.

```
df_ADD[['Glucose','BloodPressure','SkinThickness' , 'BMI', 'Insulin']]=
df_ADD[['Glucose','BloodPressure','SkinThickness' , 'BMI', 'Insulin']].replace(0,np.nan)
df_ADD.isnull().sum()
```

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

شکل ۱۵: جایگذاری مقادیر nan به جای " ۰ "

این روش، بر اساس روشی تکراری عمل می‌کند. ابتدا الگوریتم یک مدل پیش‌بینی را انتخاب می‌کند، مانند یک مدل رگرسیون یا یک مدل شبکه عصبی. سپس مقادیر گم شده را با استفاده از این مدل پیش‌بینی می‌کند.

مرحله بعدی این است که مقادیر پیش‌بینی شده را به عنوان مقادیر جدید برای مقادیر گم شده استفاده کنیم و این مقادیر را در داده‌ها جایگزین کنیم. با این کار، مقادیر گم شده جایگزین شده و یک نسخه جدید از داده‌ها به دست می‌آید. این فرآیند تکرار می‌شود و هر بار مدل پیش‌بینی بر اساس داده‌های جدید آموزش داده می‌شود. این روند تا زمانی ادامه پیدا می‌کند که مقادیر گم شده به اندازه کافی پر شوند یا معیاری مانند تعداد تکرارها مشخص شود.

روش IterativeImputer برای پر کردن مقادیر گمشده از اطلاعات موجود در داده‌ها بهره می‌برد. با استفاده از روش تکراری، مدل پیش‌بینی تلاش می‌کند تا از روابط و الگوهای موجود در داده‌ها استفاده کند و مقادیر مفقود را تخمین بزند.

Single Imputer روشی است که مقادیر گمشده را با استفاده از یک مقدار واحد برای هر ویژگی در مجموعه داده به حساب می‌آورد.

Mean Imputer محاسبه‌گر میانگین نوعی از محاسبه‌گر منفرد است که مقادیر گمشده را با استفاده از مقدار میانگین ویژگی محاسبه می‌کند.

شکل‌های ۱۶ و ۱۷ کدهای مربوط را نمایش می‌دهند.

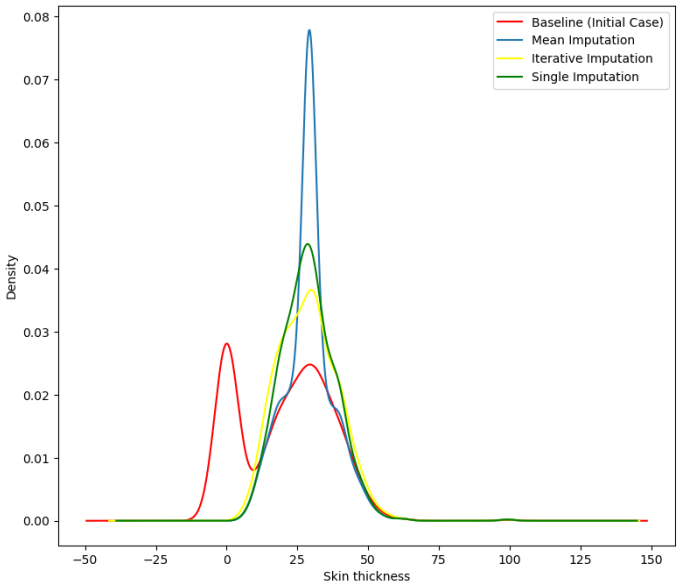
```
# Single Imputer
SingleImputer = IterativeImputer(missing_values=np.nan)
DfSingle = df_3Sigma.copy(deep=True)
DfSingle.iloc[:, :] = SingleImputer.fit_transform(DfSingle)

# Mean Imputer
MeanImputer =SimpleImputer(missing_values=np.nan, strategy='mean')
DfMean = df_3Sigma_1.copy(deep=True)
DfMean.iloc[:, :] = MeanImputer.fit_transform(DfMean)

# Iterative Imputer
IterativeImputer = IterativeImputer(missing_values=np.nan, sample_posterior=True, min_value=0,
                                    random_state=0)
DfIterative = df_3Sigma_1.copy(deep=True)
DfIterative.iloc[:, :] = IterativeImputer.fit_transform(DfIterative)
```

شکل ۱۶: کد اجرا شده برای جایگذاری مقادیر گمشده

```
plt.figure(figsize=(20,8))
plt.subplot(1,2,1)
df['SkinThickness'].plot(kind='kde', c='red')
DfMean['SkinThickness'].plot(kind='kde')
DfIterative['SkinThickness'].plot(kind='kde', c='yellow')
DfSingle['SkinThickness'].plot(kind='kde', c='green')
labels = ['Baseline (Initial Case)', 'Mean Imputation', 'Iterative Imputation',
'Single Imputation']

plt.legend(labels)
plt.xlabel('Skin thickness')
plt.show()
```

شکل ۱۷: کد اجرا شده برای نمایش نمودار مقایسه‌ای هر یک از روش‌ها

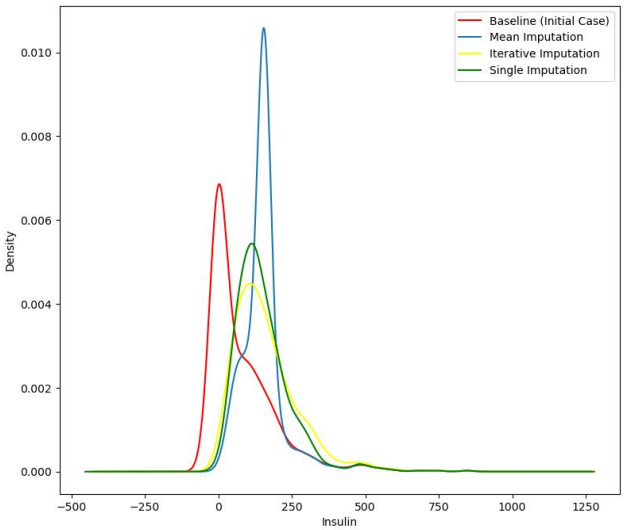
در شکل ۱۸ سه روش گفته شده برای پر کردن داده‌های گمشده را برای این ویژگی انجام دادیم، خط قرمز رنگ در نمودار خود داده اصلی هست. ما می‌خواهیم ببینیم کدوم یک از روش‌ها با این خط قرمز همپوشانی بیشتری دارد. در این بخش مشاهده می‌شود که نمودار زرد رنگ تشابه بیشتری به نسبت بقیه نمودارها با نمودار اصلی دارد.



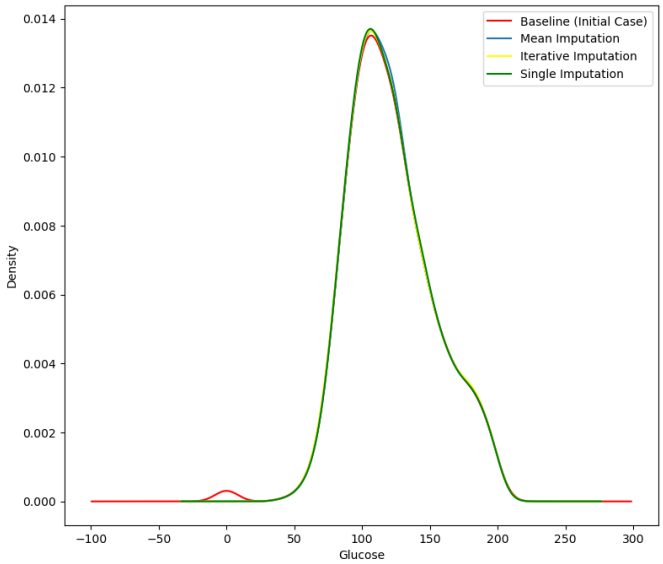
شکل ۱۸: نمودار ضخامت پوست

در شکل ۱۹ویژگی انسولین را هم مانند قبل تحلیل می‌کنیم. در این بخش نموداری که تشابه زیادی با نمودار قرمز رنگ داشته باشد موجود نیست. در شکل ۲۰ برای نمودار گلوکز مشاهده می‌کنیم که تمامی روش‌های جایگذاری بر روی نمودار اصلی تقریبا منطبق شده‌اند. این مهم به علت تعداد کم داده‌های گمشده برای این ویژگی می‌باشد. اما با توجه و دقت بیشتر مشاهده می‌کنیم که نمودار زرد رنگ نزدیکی بیشتری به نمودار اصلی دارد. در شکل ۲۱، نمودار فشارخون، تحلیل مانند قبل صورت می‌گیرد. نزدیکترین نمودار به نمودار اصلی، نمودار زرد رنگ می‌باشد. با مشاهده نمودار شکل ۲۲ برای نمودار BMI، متوجه می‌شویم که این ویژگی نیز عملکردی مشابه ویژگی‌های قبل دارد. نزیکترین نمودار به نمودار اصلی، نمودار زرد رنگ می‌باشد.

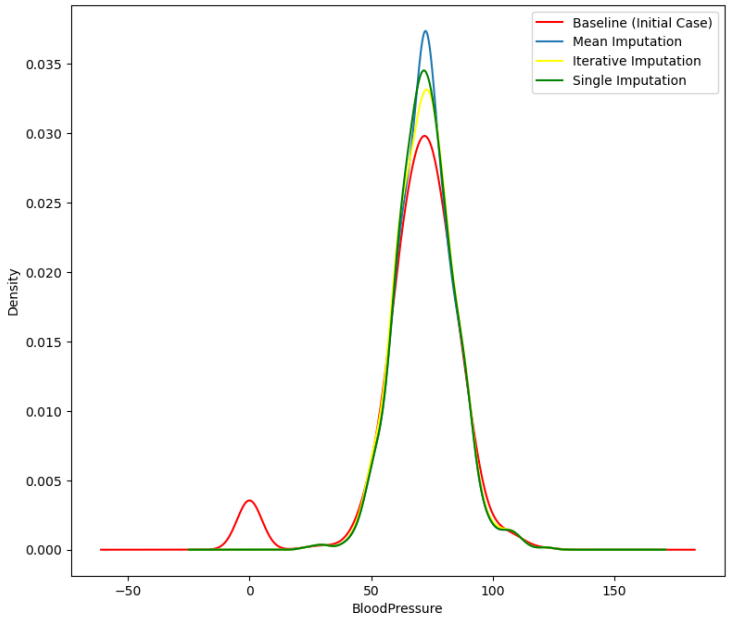
طبق مشاهدات صورت گرفته در این بخش، نزدیکترین مدل جایگذاری مقادیر گمشده به مجموعه داده اصلی روشIterativeImputer می‌باشد.



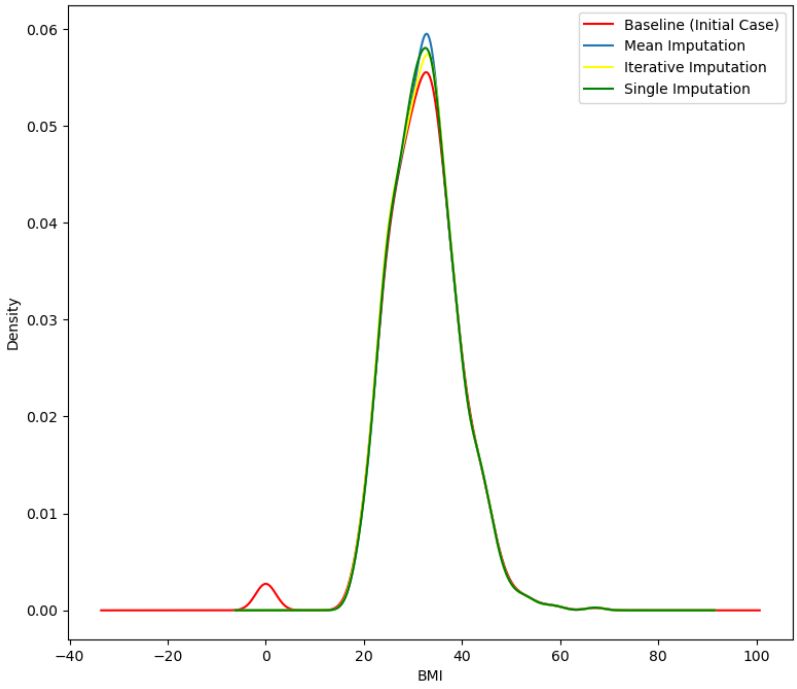
شکل ۱۹: نمودار انسولین



شکل ۲۰: نمودار گلوکز



شکل ۲۱: نمودار فشارخون



شکل ۲۲: نمودار BMI

۴-۴-۲ استاندارد سازی داده‌ها

در این بخش برای استانداردسازی داده ها از روش مین مکس استفاده کرده‌ایم(شکل شماره ۲۳). این روش، نوعی الگوریتم بهینه سازی است که برای یافتن بهترین راه حل ممکن برای یک مسئله معین با جست و جو در تمام راه حل‌های ممکن و انتخاب راه حل با بالترین ارزش یا کمترین هزینه استفاده می‌شود. با به حداقل رساندن حداکثر هزینه یا به حداکثر رساندن حداقل مقدار یک مجموعه معین از پارامترها کار می‌کند. خروجی داده‌ها در جدول ۱۱ نمایش داده شده است.

```
Min_Max_Scaled = MinMaxScaler().fit_transform(DfIterative)
df_Min_Max = pd.DataFrame(Min_Max_Scaled, columns = list(DfIterative.columns))
df_Min_Max
```

شکل ۲۳: استاندارد سازی داده

جدول ۱۱: داده‌های استاندارد شده به روش مین مکس

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.352941	0.670968	0.489796	0.316818	0.397873	0.314928	0.234415	0.483333	1.0
1	0.058824	0.264516	0.428571	0.252770	0.159744	0.171779	0.116567	0.166667	0.0
2	0.470588	0.896774	0.408163	0.105817	0.430949	0.104294	0.253629	0.183333	1.0
3	0.058824	0.290323	0.428571	0.188722	0.111111	0.202454	0.038002	0.000000	0.0
4	0.000000	0.600000	0.163265	0.316818	0.198582	0.509202	0.943638	0.200000	1.0
...	...	...	...	...	...	...	...	...	...
763	0.588235	0.367742	0.530612	0.455590	0.212766	0.300613	0.039710	0.700000	0.0
764	0.117647	0.503226	0.469388	0.231421	0.359708	0.380368	0.111870	0.100000	0.0
765	0.294118	0.496774	0.489796	0.188722	0.132388	0.163599	0.071307	0.150000	0.0
766	0.058824	0.529032	0.367347	0.267434	0.148029	0.243354	0.115713	0.433333	1.0
767	0.058824	0.316129	0.469388	0.274119	0.105036	0.249489	0.101196	0.033333	0.0

768 rows × 9 columns

۲-۴-۵ تقسیم بندی داده‌ها به دو دسته آموزشی و آزمایشی

یکی از مهم‌ترین کارها در پردازش داده، تقسیم بندی داده‌ها به گروه‌های آموزشی و آزمایشی می‌باشد. در شکل ۲۴، ۳۵ درصد از داده‌ها را به دسته آزمایشی و ۶۵ درصد را به آموزشی تقسیم بندی می‌کنیم.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=7)
scaler = preprocessing.StandardScaler().fit(X_train)
X_train_transformed = scaler.transform(X_train)
clf = KNeighborsClassifier(n_neighbors=5).fit(X_train_transformed, y_train)
X_test_transformed = scaler.transform(X_test)
clf.score(X_test_transformed, y_test)
```

```
0.7323420074349443
```

```
X_test.shape, y_test.shape
```

```
((269, 8), (269,))
```

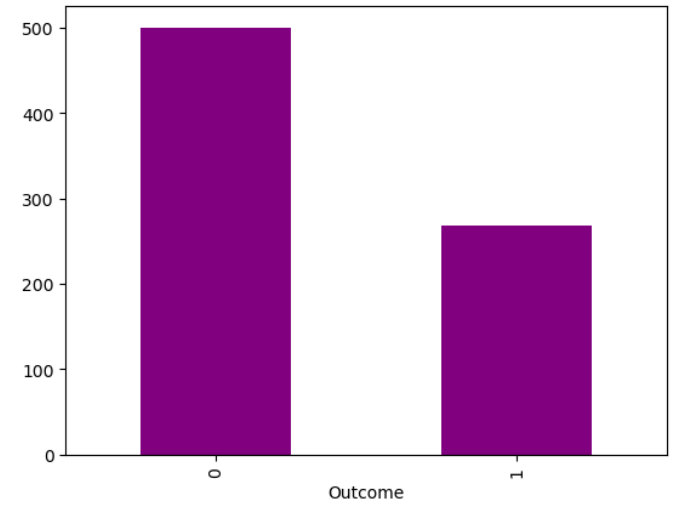
```
X_train.shape, y_train.shape
```

```
((499, 8), (499,))
```

شکل ۲۴: کد تقسیم داده¬ها به دو دسته آموزشی و آزمایشی

در این بخش می‌خواهیم داده را متعادل کنیم. شکل شماره ۲۵ تعداد داده‌های موجود در هر کلاس را نمایش می‌دهد. با توجه به نمودار مربوط خواهیم دید که داده‌ها متعادل نیستند.

```
counts.plot(kind="bar", color ='purple')
plt.show()
```



شکل ۲۵: نمایش نمودار میله‌ای داده‌ها

همانگونه که می‌دانیم برای متعادل سازی داده‌ها، چندین روش وجود دارد. در شکل ۲۶، از روش OverSampling برای متعادل سازی داده‌ها استفاده کرده‌ایم. نتایج خروجی آن در شکل ۲۷ قابل مشاهده می‌باشد.

```
y_train_discrete = np.where(y_train > 0, 1, 0)

sm = SMOTE(random_state=2)

print("\nClass 1 before Over Sampling --> ", sum(y_train_discrete == 1))
print("\nClass 0 before Over Sampling --> ", sum(y_train_discrete == 0))

X_train_OS, y_train_OS = sm.fit_resample(X_train, y_train_discrete)

print("\nThe shape of X after Over Sampling -->", X_train_OS.shape)
print("\nThe shape of Y after Over Sampling -->", y_train_OS.shape)

print("\nClass 1 after Over Sampling --> ", sum(y_train_OS == 1))
print("\nClass 0 after Over Sampling --> ", sum(y_train_OS == 0))
print("\n")
```

شکل ۲۶: کد متعادل سازی مجموعه داده

```
Class 1 before Over Sampling -->  173

Class 0 before Over Sampling -->  326

The shape of X after Over Sampling --> (652, 8)

The shape of Y after Over Sampling --> (652,)

Class 1 after Over Sampling -->  326

Class 0 after Over Sampling -->  326
```

شکل ۲۷: خروجی پس از متعادل سازی مجموعه داده

۲-۴-۶ انتخاب ویژگی با روش جست و جو رو به جلو

انتخاب ویژگی، به معنی انتخاب زیرمجموعه‌ای از ویژگی‌ها یا فاکتورهای مرتبط با مجموعه داده‌ها می‌باشد که بیشترین اطلاعات مفید را برای مدلسازی فراهم می‌کنند. زمانی که از روش انتخاب ویژگی استفاده می‌کنیم، تعداد ویژگی‌های ورودی به مدل کاهش پیدا می‌کند. دلیل استفاده از این روش این است که معمولاً مدل‌های یادگیری ماشین برای مجموعه داده‌ها با تعداد ویژگی‌های بالا ، به دلیل ابعاد بالا و مواجه شدن با عملیات محاسباتی زیاد و در نهایت نتایج ضعیف، می‌توانند با مشکل مواجه شوند. بنابراین، با کاهش تعداد ویژگی‌ها، می‌توانیم هر دو مشکل را حل کنیم. با این کار، میتوانیم از بار محاسباتی کمتری در مدلسازی استفاده کرده و در مقایسه با مجموعه داده‌های اولیه، مدل ایجاد شده با دقت بیشتری پیش‌بینی و پایداری داشته باشد.

روش رو به جلو در انتخاب ویژگی تکنیکی است برای انتخاب زیرمجموعه‌هایی از ویژگی‌ها از داده‌های اصلی که بیشترین ارتباط را برای کار پیش‌بینی دارند. ایده این است که با مجموعه‌ای خالی از ویژگی‌ها شروع کنید و هر بار یک ویژگی را بر اساس میزان بهبود عملکرد مدل اضافه کنید. این فرآیند تا زمانی تکرار میشود که با افزودن ویژگی‌های بیشتر، بهبود بیشتری حاصل نشود. شکل شماره ۲۸ کد این بخش را نمایش می‌دهد.

```
: forward_feature_selection = (SequentialFeatureSelector(KNeighborsClassifier(n_neighbors=5 , metric='euclidean'),
                                                         k_features = (1,8),
                                                         forward=True,
                                                         floating=False,
                                                         verbose=2,
                                                         scoring='accuracy',
                                                         cv = 5)).fit(X_train_OS, y_train_OS)

forward_columns = list(forward_feature_selection.k_feature_names_)
print("\nNo. of Selected columns : ", len(forward_columns))
print("Selected features by forward --->",forward_columns)
X_train_forward = X_train_OS[forward_columns]
X_test_forward = X_test[forward_columns]
X_train_forward
```

شکل ۲۸: کد روش جست و جو رو به جلو

در نهایت با استفاده از این روش، ویژگی‌های موجود در شکل ۲۹ را خواهیم داشت:

	Glucose	Insulin	BMI	Age
0	0.445161	0.113021	0.261759	0.000000
1	0.109677	0.126941	0.331288	0.416667
2	0.406452	0.337366	0.130879	0.216667
3	0.400000	0.186761	0.259714	0.050000
4	0.412903	0.210402	0.353783	0.050000

شکل ۲۹: ویژگی انتخابی از روش جست و جو رو به جلو

در ادامه برای اعتبار سنجی روش ارائه شده، معیارهای دقت در شکل ۳۰، حساسیت در شکل ۳۱و اختصاصیت در شکل ۳۲ را گزارش می‌کنیم:

```
model = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
model.fit(X_train_forward, y_train_OS)
y_pred = model.predict(X_test_forward)
accuracy = accuracy_score(y_test, y_pred)
```

شکل ۳۰: ارائه مدل برای اعتبارسنجی

```
print("Accuracy of the model with selected features: {:.2f}".format(accuracy))
```

```
Accuracy of the model with selected features: 0.71
```

شکل ۳۱: گزارش دقت

-

```
print(' Sensitivity :', recall_score(y_test, y_pred))
```

```
 Sensitivity : 0.6947368421052632
```

شکل ۳۱: گزارش حساسیت

```
conf_matrix = confusion_matrix(y_test, y_pred)
TN = conf_matrix[0, 0]
FP = conf_matrix[0, 1]

specificity = TN / (TN + FP)
print("Specificity of the model with selected features: {:.2f}".format(specificity))
```

```
Specificity of the model with selected features: 0.72
```

شکل ۳۲: گزارش اختصاصیت

با توجه به معیارهای گزارش شده، میتوانیم نتیجه بگیریم، روش انتخاب ویژگی از سه منظر دقت، حساسیت و اختصاصیت، عملکرد مطلوب و قابل قبولی داشته است.

۳-اجرای الگوریتم KNN

صورت سوال: حال از روش - kNN جهت دستهبندی استفاده کنید. ۶۵ درصد دادهها را برای آموزش و الباقی را برای تست به کار ببرید. ابتدا کای مناسب را بر اساس دقت بدست آورید. سپس ماتریس در هم ریختگی را تشکیل داده و نتایج مندرج در آن را بر اساس معیارهای دقت، حساسیت و اختصاصیت شرح دهید. همچنین در ادامه با روش leave-one-out عملکرد روش KNN را ارزیابی نموده و معیارهای دقت، حساسیت و اختصاصیت را گزارش کنید.

۳-۲-پیدا کردن تعداد همسایه مناسب

برای پیدا کردن K مناسب ابتدا محدودهی مناسب برای K را تعریف میکنیم. حد بالای مقدار K برابر با $\sqrt{m}$ است. ۶۵۲ داده آموزشی داریم، ریشه دوم این عدد برابر با ۲۵.۵۳ است. بنابراین یک حد بالای مناسب برای جست و جو را میتوان $K = ۲۵$ در نظر گرفت. برای پیدا کردن K مناسب از یک حلقه استفاده میکنیم. به طوری که ابتدا K را برابر ۱ قرار میدهیم، مدلی را با این پارامتر آموزش میدهیم و تست میکنیم، مقدار دقت را محاسبه میکنیم و مقدار به دست آمده را در یک لیست ذخیره میکنیم. سپس با استفاده از تابع argmax() از کتابخانه نامپای، K متناظر با بیشترین مقدار دقت را نمایش میدهیم. شکل ۳۳ کدی که وظیفه پیدا کردن K را به عهده دارد را نمایش میدهد.

For finding the suitable K, we will use grid search

```
# Define the range of K values to search
k_values = list(range(1, 26))

# Train KNN models with different K values and calculate validation accuracy
val_scores = []
for k in k_values:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train_forward, y_train_OS)
    y_pred = model.predict(X_test_forward)
    accuracy = accuracy_score(y_test, y_pred)
    val_scores.append(accuracy)

# Find the best K value
best_k = k_values[np.argmax(val_scores)]
print("Best K:", best_k)
```

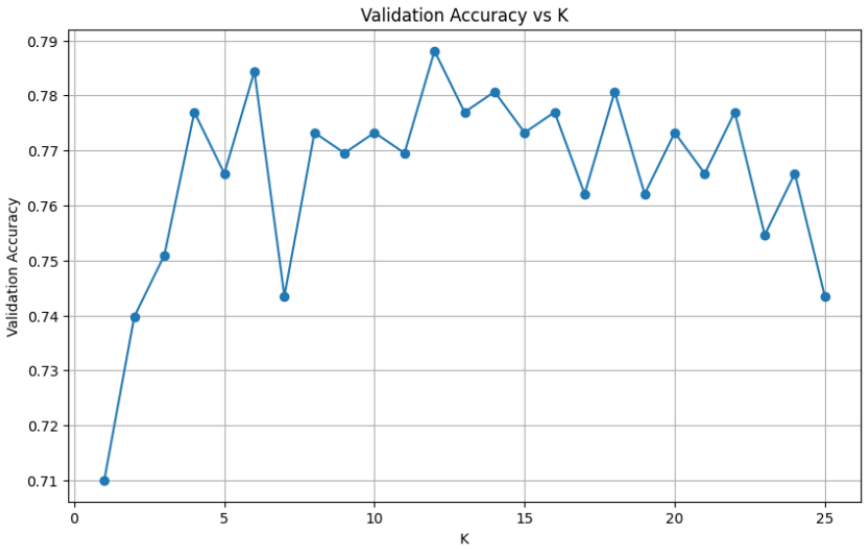
```
Best K: 12
```

شکل ۳۳: کد پیدا کردن K بهینه

با توجه به بررسی صورت گرفته، $K = ۱۲$ بهترین دقت را دارد. همچنین نمودار دقت بر اساس K را نیز رسم میکنیم. شکل ۳۴ این نمودار را نمایش میدهد.

حال بار دیگر با پارامتر $K = ۱۲$ و همین دیتاست، مدل را آموزش و ارزیابی میکنیم. سپس ماتریس درهمریختگی، و معیارهای دقت، حساسیت و اختصاصیت را با استفاده از فرمولهای مربوطه محاسبه میکنیم. برای محاسبه دقت از تابع پیشساخته accuracy_score() در سایکیت لرن استفاده میکنیم. شکل ۳۵ ماتریس درهم ریختگی و شکل ۳۶ محاسبه سه معیار را نمایش میدهد.

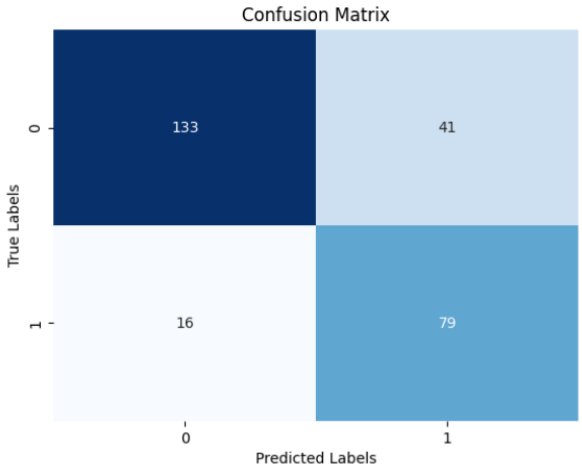
```python
# Plot accuracy vs K
plt.figure(figsize=(10, 6))
plt.plot(k_values, val_scores, marker='o', linestyle='-')
plt.xlabel('K')
plt.ylabel('Validation Accuracy')
plt.title('Validation Accuracy vs K')
plt.grid(True)
plt.show()
```



شکل ۳۴: نمودار معیار دقت بر اساس K

confusion matrix

```python
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=model.classes_, yticklabels=model.classes_)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



شکل ۳۵: ماتریس درهم‌ریختگی و کد ایجاد کننده آن

```
# Extract values from confusion matrix
tn, fp, fn, tp = conf_matrix.ravel()
```

accuracy

```
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

```
0.7881040892193308
```

sensitivity

```
sensitivity = tp / (tp + fn)
sensitivity
```

```
0.8315789473684211
```

specificity

```
specificity = tn / (tn + fp)
specificity
```

```
0.764367816091954
```

شکل ۳۶: محاسبه سه مقدار دقت، حساسیت و اختصاصیت به ترتیب

۳-۲-ارزیابی عملکرد KNN با روش Leave One Out

ابتدا یک مدل KNN با پارامتر $K = ۱۲$ ایجاد می‌کنیم و روی داده‌های آموزشی آن را آموزش می‌دهیم. شکل ۳۷ نحوه انجام این کار را نمایش می‌دهد.

Initializing the model

```
model = KNeighborsClassifier(n_neighbors=12)
model.fit(X_train_forward, y_train_OS)
```

شکل ۳۷: ایجاد مدل و آموزش مدل

حال مدل آموزش داده شده را با استفاده از روش Leave One Out ارزیابی می‌کنیم. ابتدا از کلاس LeaveOneOut() یک شی به نام CV ایجاد می‌کنیم. سپس با استفاده از تابع cross_val_score() ارزیابی را انجام می‌دهیم. این تابع، مدل ایجاد شده، دیتاست آموزشی و نحوه انجام ارزیابی را دریافت می‌کند و سپس داده‌ها را بر اساس نحوه انجام دریافت شده تقسیم می‌کند و مدل را ارزیابی می‌کند. سپس آرایه‌ی حاوی ارزیابی‌ها در یک دیتافریم ذخیره می‌شود و از آن میانگین گرفته می‌شود. شکل ۳۸ نحوه کارکرد این کد را نمایش می‌دهد.

Performing LOOCV

```
CV = LeaveOneOut()
CV_scores = cross_val_score(model, X_train_forward, y_train_OS, cv=CV)
CV_score_df = pd.DataFrame(CV_scores, columns = ['Accuracy'])
```

✓ 6.4s

```
print('The Average of CV Scores -->', CV_score_df.Accuracy.mean())
```

✓ 0.0s

```
The Average of CV Scores --> 0.7791411042944786
```

```
y_pred = model.predict(X_test_forward)
```

✓ 0.1s

شکل ۳۸: نحوه انجام ارزیابی Leave One Out

در ادامه سه معیار دقت، حساسیت و اختصاصیت این مدل محاسبه می‌شود. شکل ۳۹ این سه مقدار را به ترتیب نمایش می‌دهد.

accuracy

```
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

✓ 0.0s

```
0.7881040892193308
```

sensitivity

```
sensitivity = tp / (tp + fn)
sensitivity
```

✓ 0.0s

```
0.8315789473684211
```

specificity

```
specificity = tn / (tn + fp)
specificity
```

✓ 0.0s

```
0.764367816091954
```

شکل ۳۹: سه معیار دقت حساسیت و اختصاصیت حاصل از Leave One Out

۴-دسته‌بندی بیزی

برای انجام دسته‌بندی بیزی از همان دیتاست بخش قبلی استفاده می‌کنیم. با استفاده از کلاس GaussianNB() از ماژول scikit-learn مدل بیزی را آموزش می‌دهیم. برای این منظور ابتدا یک شی به نام nb_classifier از این کلاس می‌سازیم و با پاس دادن دیتاست ویژگی‌ها و ستون لیبل متناظر آنها به عنوان آرگومان به متد fit()، مدل دسته‌بندی بیزی راآموزش می‌دهیم. در ادامه با استفاده از متد predict() و پاس دادن دیتاست تست، پیش‌بینی‌های مدل از این دیتاست را به دست می‌آوریم و در متغیر y_pred ذخیره می‌کنیم. شکل ۴۰ نحوه انجام این عمل را نمایش می‌دهد.

Bayesian classification

```
nb_classifier = GaussianNB()

# Train the classifier on the training data
nb_classifier.fit(X_train_forward, y_train_OS)

# Make predictions on the testing data
y_pred = nb_classifier.predict(X_test_forward)
```

✓ 0.0s

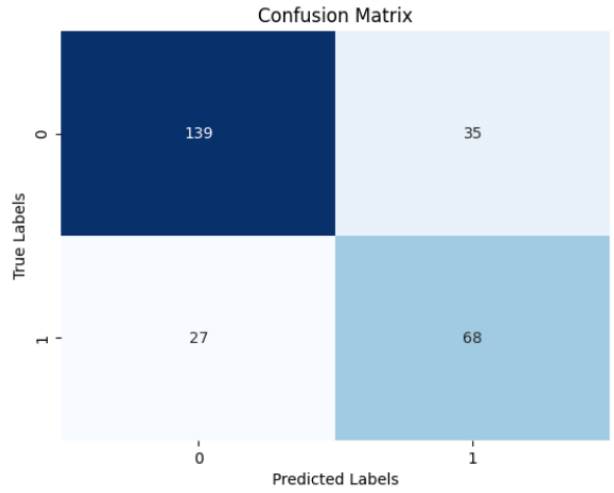
شکل ۴۰: کد مربوط به دسته‌بندی بیزی

سپس همانند دو بخش قبلی به محاسبه ماتریس درهم‌ریختگی و معیارهای ارزیابی می‌پردازیم. شکل ۴۱ ماتریس درهم‌ریختگی روش دسته‌بندی بیز را نشان می‌دهد. همچنین شکل ۴۲ مقدار سه معیار دقت حساسیت و اختصاصیت را برای روش دسته‌بندی بیز نشان می‌دهد.

confusion matrix

```
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False, )
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

✓ 0.1s



شکل ۴۱: ماتریس درهم ریختگی حاصل از دسته‌بندی بیزی

accuracy

```
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

✓ 0.0s

```
0.7695167286245354
```

sensitivity

```
sensitivity = tp / (tp + fn)
sensitivity
```

✓ 0.0s

```
0.8315789473684211
```

specificity

```
specificity = tn / (tn + fp)
specificity
```

✓ 0.0s

```
0.764367816091954
```

شکل ۴۲: سه معیار دقت حساسیت و اختصاصیت برای مدل دسته‌بندی بیزی

۵- مسئله دوم

داده‌های زیر را در نظر بگیرید(۴,۴) (۴,۳) (۳,۴) (۳,۳) (۱,۱) (۱,۰) (۰,۱) (۰,۰) داده ها فوق را در محور مختصات دو بعدی ترسیم نمایید. به صورت چشمی آنها را به دو کلاس تخصیص دهید(برچسب داده ها را معین کنید). ابتدا با کمک روش نزدیکترین همسایگی، مرز دو کلاس را بیابید. سپس با روش پرسپترون مرز دو کلاس را بیابید.

۵-۱ ترسیم در محور مختصات دو بعدی و تعیین برچسب داده ها

تعریف داده‌ها : داده‌ها به صورت یک لیست از جفت‌های مختصات (x,y) تعریف شده است.

تقسیم داده‌ها به دو کلاس: داده‌ها بر اساس موقعیتشان در محور مختصات دوبعدی به دو کلاس تقسیم می‌شوند. در اینجا، داده‌هایی که مختصات x و y آن‌ها کوچکتر ۳ هستند در کلاس ۱ و داده‌های دیگر در کلاس ۲ قرار می‌گیرند.

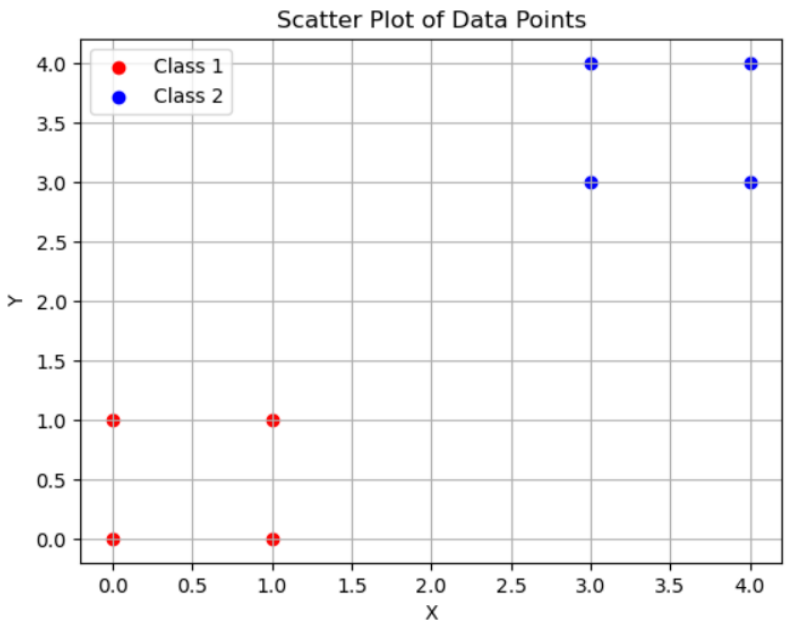
ترسیم داده‌ها :با استفاده از تابع plt.scatter() داده‌ها در محور مختصات دوبعدی نمایش داده می‌شوند. داده‌های کلاس ۱ با رنگ قرمز و داده‌های کلاس ۲ با رنگ آبی نمایش داده شده‌اند.

تنظیمات نمودار: محور x و y به ترتیب به "X" و "Y" تنظیم شده‌اند. همچنین عنوان نمودار به "Scatter Plot of Data Points" تغییر یافته است. همچنین با استفاده از plt.legend() برچسب‌های کلاس‌ها به درستی به نمودار اضافه شده‌اند و با فراخوانی plt.grid(True) خطوط شبکه در نمودار نیز نمایش داده شده‌اند. و در نهایت با استفاده از plt.show() نمودار نمایش داده می‌شود. کد ها و خروجی های این بخش در شکل های ۴۳ و ۴۴ قرار گرفته شده است.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
```

```python
# Define the data points
data = [(0, 0),(1, 0), (0, 1), (1, 1), (3, 3), (4, 3), (3, 4),(4, 4)]

# Divide the data into two classes
class_1 = []
class_2 = []

for point in data:
    if point[0] < 3 and point[1] < 3:
        class_1.append(point)
    else:
        class_2.append(point)

# Plot the data points
plt.scatter([point[0] for point in class_1], [point[1] for point in class_1], color='red', label='Class 1')
plt.scatter([point[0] for point in class_2], [point[1] for point in class_2], color='blue', label='Class 2')

# Chart settings
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Scatter Plot of Data Points')
plt.legend(loc='upper left')
plt.grid(True)
plt.show()

```

شکل ۴۳:کدهای قسمت ۱



شکل ۴۴:خروجی قسمت ۱

۵-۲ تعیین مرز دو کلاس با روش نزدیکترین همسایگی

تعریف داده‌ها و برچسب‌ها: در این قسمت، داده‌ها به صورت نقاطی در فضای دو بعدی تعریف شده‌اند. همچنین برچسب‌های متناظر با هر نقطه نیز تعیین شده‌اند که نشان می‌دهد هر نقطه به کدام دسته تعلق دارد. به عنوان مثال، اگر اولین چهار نقطه به یک دسته و آخرین چهار نقطه به دسته دیگر تعلق داشته باشند، برچسب‌های متناظر با این نقاط به ترتیب ۱ و ۲ است.

ساخت مدلKNN : در این بخش، یک مدل KNN با استفاده از کتابخانه scikit-learn ایجاد می‌شود. این مدل تعیین کننده‌ی تعداد همسایه‌هایی است که برای هر نقطه برای پیش‌بینی برچسب استفاده می‌شود. در اینجا=k ۱ است، بنابراین تنها نزدیک‌ترین نقطه برای هر نقطه برای پیش‌بینی برچسب در نظر گرفته می‌شود.

آموزش مدل: مدل KNN با استفاده از داده‌های آموزشی و برچسب‌هایش آموزش داده می‌شود. این به این معناست که مدل با داده‌های آموزشی تنظیم می‌شود تا بتواند برچسب‌های نقاط جدید را پیش‌بینی کند.

نمایش مرز تصمیم و پیش‌بینی برچسب نقاط جدید: بعد از آموزش مدل، می‌توان با استفاده از مدل برچسب‌های نقاط جدید را پیش‌بینی کرد. این کار با فراخوانی تابع predict انجام می‌شود که بر اساس مدل آموزش دیده، برچسب‌های نقاط جدید را پیش‌بینی می‌کند.

برای نمایش مرز تصمیم، محدوده‌ای از فضای داده‌ها (با استفاده از حداقل و حداکثر مقادیر x وy) مشخص شده و برای هر نقطه در این محدوده، برچسب پیش‌بینی شده توسط مدل رسم می‌شود. این کار باعث تشکیل یک مرز تصمیم بین دسته‌ها می‌شود که می‌تواند به صورت یک خط یا حتی یک منحنی نمایش داده شود.

این بخش از کد در واقع مربوط به ایجاد یک شبکه‌ی دقیق از نقاط برای ایجاد مرز تصمیم است. برای این کار، محدوده‌ای از مقادیر x و y را ایجاد می‌کند که در آن محدوده، اطراف داده‌ها همگرا شده‌اند. سپس یک شبکه از نقاط روی این محدوده ایجاد می‌شود. با استفاده از این شبکه، برای هر نقطه از آن، مقدار پیش‌بینی شده توسط مدل KNN محاسبه می‌شود. این مقادیر سپس برای رسم مرز تصمیم استفاده می‌شوند.

خط۱۳ کد: این خط کمینه و بیشینه مقادیر x را مشخص می‌کند و سپس یک واحد اضافه می‌کند تا مرزهای داده را افزایش دهد. **خط۱۴ کد** هم این کار را برای مقادیر y انجام می‌دهد.

خط۱۵ و ۱۶ کد: این تابع یک شبکه از نقاط را ایجاد می‌کند که در آن مقادیر x و y از x_min تا x_max و y_min تاy_max با گام ۰.۱ قرار می‌گیرند.

خط۱۷ کد: برای هر نقطه از شبکه، مقدار پیش‌بینی شده توسط مدل KNN محاسبه می‌شود.

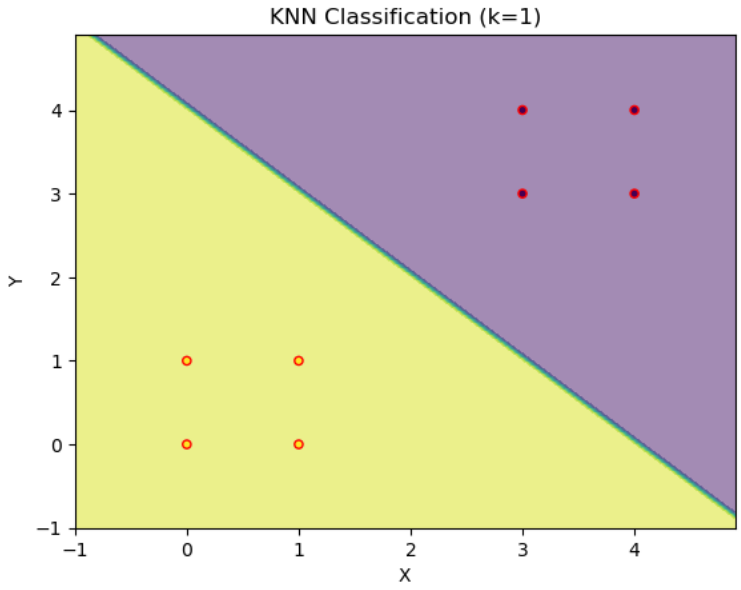
خط۱۸ کد: شکل Z را با استفاده از ابعاد شبکه xx تغییر می‌دهد تا درست متناظر با آن شود.

در نهایت، یک نمودار با استفاده از کتابخانه matplotlib ایجاد می‌شود که داده‌ها به همراه مرز تصمیم بر روی آن نمایش داده می‌شود. این نمودار به ما کمک می‌کند تا دسته‌بندی و مرز تصمیم را به صورت بصری درک کنیم.

کد ها و خروجی های این بخش در شکل های ۴۵ و ۴۶ قرار گرفته شده است.

```
# Define the data points
X = np.array([[0, 0], [1, 0], [0, 1], [1, 1],[3, 3], [4, 3], [3, 4], [4, 4]])
# Define the labels for each point (assuming first four points belong to one class, last four to another)
y = np.array([1, 1, 1, 1, -1, -1, -1, -1])

# Create KNN classifier with k=1 (nearest neighbor)
knn = KNeighborsClassifier(n_neighbors=1)
# Fit the classifier to the data
knn.fit(X, y)

# Plot the decision boundary
h=0.1 #step size
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the data points and decision boundary
plt.contourf(xx, yy, Z, alpha=0.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='r')
plt.title("KNN Classification (k=1)")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

شکل ۴۵: :کدهای روش NN



شکل ۴۶: خروجی روش NN

۳-۵ تعیین مرز دو کلاس با روش پرسپترون

تعریف داده‌ها و برچسب‌ها: X یک ماتریس است که هر سطر آن نقطه‌ای در فضای دوبعدی است و y یک بردار است که برچسب متناظر با هر نقطه را نشان می‌دهد. به عبارت دیگر، برچسب‌های دسته‌بندی برای هر نقطه ارائه شده است.

مقداردهی اولیه وزن‌ها و انحراف:

Weights یک بردار بر اساس لیبل های موجود تعریف میکنیم که با ابعاد برابر تعداد ویژگی‌ها (در اینجا ۲) است.

Bias به صورت رندوم ۱ قرار می دهیم.

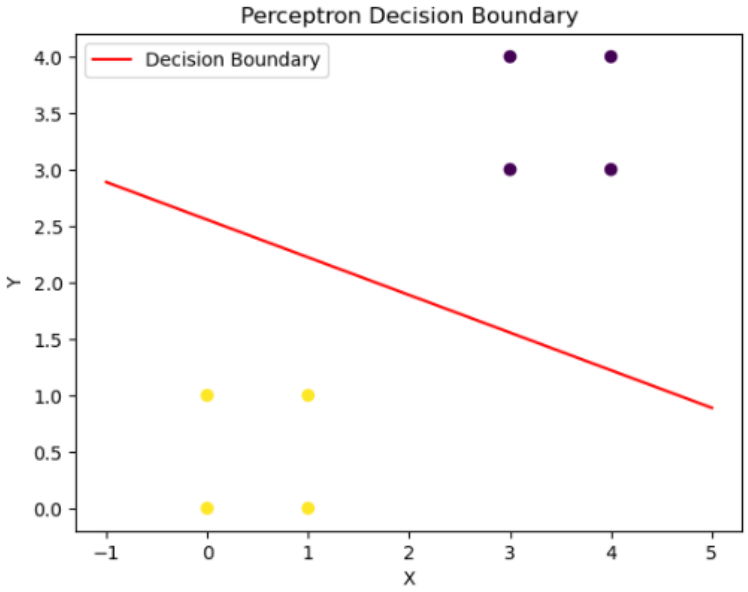
تعریف تابع آموزش پرسپترون: یک تابع با نام perceptron_train تعریف شده است که مسئول آموزش مدل پرسپترون بر روی داده‌ها است. این تابع برای تعداد مشخصی از تکرارها (epochs) روی داده‌ها حلقه می‌زند و وزن‌ها و انحراف را بر اساس نقاط داده و برچسب‌هایشان به‌روزرسانی می‌کند.(به طور رندوم و با انجام تکرار های مختلف نرخ یادگیری ۰.۴ دارای بیشترین دقت که ۸۰ درصد است را میدهد.)

آموزش مدل پرسپترون: مدل پرسپترون با استفاده از تابع آموزش perceptron_train آموزش داده می‌شود. در هر مرحله از آموزش، وزن‌ها و انحراف به‌روزرسانی می‌شوند تا مدل بتواند به درستی داده‌ها را دسته‌بندی کند.

نمایش مرز تصمیم: مرز تصمیم بین دو دسته با استفاده از وزن‌ها و انحراف محاسبه شده و در نهایت روی نمودار نشان داده می‌شود (با استفاده از فرمول موجود). این مرز تصمیم به صورت یک خط در فضای داده نمایش داده می‌شود تا بتوانیم داده‌ها و دسته‌بندی آن‌ها را بصری‌سازی کنیم. کد ها و خروجی های این بخش در شکل های ۴۷ و ۴۸ قرار گرفته شده است.

```python
# Define the data points
X = np.array([[0, 0], [1, 0], [0, 1], [1, 1],[3, 3], [4, 3], [3, 4], [4, 4]])
# Define the labels for each point (assuming first four points belong to one class, last four to another)
y = np.array([1, 1, 1, 1, -1, -1, -1, -1])

# Initialize weights and bias
weights = [1,-1]
bias = 1

# Define perceptron training function
def perceptron_train(X, y, weights, bias, learning_rate=0.4, epochs=1000):
    for _ in range(epochs):
        for i in range(len(X)):
            # Compute the predicted class
            predicted = np.sign((np.dot(X[i], weights)) - bias)

            # Update weights and bias if misclassified
            if (y[i] * predicted) <= 0:
                weights += 2*learning_rate * y[i] * X[i]
                bias += -2*learning_rate * y[i]
    return weights, bias

# Train the perceptron model
weights, bias = perceptron_train(X, y, weights, bias)

# Plot the data points and decision boundary
plt.scatter(X[:, 0], X[:, 1], c=y)
x_values = np.linspace(np.min(X[:, 0]) - 1, np.max(X[:, 0]) + 1, 100)
y_values = (-weights[0] * x_values + bias) / weights[1]
plt.plot(x_values, y_values, color='red', label='Decision Boundary')
plt.title("Perceptron Decision Boundary")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.show()

```

شکل ۴۷: کدهای روش پرسپترون



شکل ۴۸: خروجی روش پرسپترون