submission

August 15, 2025

0.1 Full Repo

https://github.com/MostafaBelo/Konecta_Assignments/tree/main

0.2 Imports

[1]:
```
import pandas as pd

import numpy as np
from matplotlib import pyplot as plt

from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

from scipy.stats import randint, uniform, loguniform

from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve, auc, confusion_matrix,␣
 ↪classification_report, roc_auc_score
```

[2]:
```
df = pd.read_csv("diabetes.csv")

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column
Non-Null Count  Dtype
---  ------
--------------  -----
 0   Number of times pregnant
```

```
768 non-null    int64
 1   Plasma glucose concentration a 2 hours in an oral glucose tolerance test
768 non-null    int64
 2   Diastolic blood pressure (mm Hg)
768 non-null    int64
 3   Triceps skin fold thickness (mm)
768 non-null    int64
 4   2-Hour serum insulin (mu U/ml)
768 non-null    int64
 5   Body mass index (weight in kg/(height in m)^2)
768 non-null    float64
 6   Diabetes pedigree function
768 non-null    float64
 7   Age (years)
768 non-null    int64
 8   Class variable
768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

0.3 Preprocessing

[3]:
```
df.describe()
# unreasonable values are some fo the zeros (which probably are the equivalent␣
 ↪to null or mising for some of the medical columns)
```

[3]:
```
       Number of times pregnant  \
count                768.000000
mean                   3.845052
std                    3.369578
min                    0.000000
25%                    1.000000
50%                    3.000000
75%                    6.000000
max                   17.000000

       Plasma glucose concentration a 2 hours in an oral glucose tolerance test
\
count                                         768.000000
mean                                          120.894531
std                                            31.972618
min                                             0.000000
25%                                            99.000000
50%                                           117.000000
75%                                           140.250000
max                                           199.000000
```

```
       Diastolic blood pressure (mm Hg)  Triceps skin fold thickness (mm)  \
count                        768.000000                        768.000000
mean                          69.105469                         20.536458
std                           19.355807                         15.952218
min                            0.000000                          0.000000
25%                           62.000000                          0.000000
50%                           72.000000                         23.000000
75%                           80.000000                         32.000000
max                          122.000000                         99.000000

       2-Hour serum insulin (mu U/ml)  \
count                      768.000000
mean                        79.799479
std                        115.244002
min                          0.000000
25%                          0.000000
50%                         30.500000
75%                        127.250000
max                        846.000000

       Body mass index (weight in kg/(height in m)^2)  \
count                                       768.000000
mean                                         31.992578
std                                           7.884160
min                                           0.000000
25%                                          27.300000
50%                                          32.000000
75%                                          36.600000
max                                          67.100000

       Diabetes pedigree function  Age (years)  Class variable
count                  768.000000   768.000000      768.000000
mean                     0.471876    33.240885        0.348958
std                      0.331329    11.760232        0.476951
min                      0.078000    21.000000        0.000000
25%                      0.243750    24.000000        0.000000
50%                      0.372500    29.000000        0.000000
75%                      0.626250    41.000000        1.000000
max                      2.420000    81.000000        1.000000
```

[4]:
```
df.head()
```

[4]:
```
   Number of times pregnant  \
0                         6
1                         1
2                         8
3                         1
```

```
4                        0

   Plasma glucose concentration a 2 hours in an oral glucose tolerance test  \
0                                                148
1                                                 85
2                                                183
3                                                 89
4                                                137

   Diastolic blood pressure (mm Hg)  Triceps skin fold thickness (mm)  \
0                                72                                35
1                                66                                29
2                                64                                 0
3                                66                                23
4                                40                                35

   2-Hour serum insulin (mu U/ml)  \
0                               0
1                               0
2                               0
3                              94
4                             168

   Body mass index (weight in kg/(height in m)^2)  Diabetes pedigree function  \
0                                            33.6                       0.627
1                                            26.6                       0.351
2                                            23.3                       0.672
3                                            28.1                       0.167
4                                            43.1                       2.288

   Age (years)  Class variable
0           50               1
1           31               0
2           32               1
3           21               0
4           33               1
```

[5]:
```
((df.isnull()) | (df == 0)).sum()
```

[5]:
```
Number of times pregnant                                                  111
Plasma glucose concentration a 2 hours in an oral glucose tolerance test    5
Diastolic blood pressure (mm Hg)                                           35
Triceps skin fold thickness (mm)                                          227
2-Hour serum insulin (mu U/ml)                                            374
Body mass index (weight in kg/(height in m)^2)                             11
Diabetes pedigree function                                                  0
Age (years)                                                                 0
```

```
Class variable                                                    500
dtype: int64
```

[6]:
```python
def list_uniques(df: pd.DataFrame, lim=30):
    for col in df.columns:
        uq_vals = df[col].unique()
        print(col)
        if lim != -1 and len(uq_vals) > lim:
            print("Too many values")
        else:
            print(uq_vals)
        print()

# No nulls/missing values found
# Target is already encoded as 0/1
list_uniques(df, -1)
```

```
Number of times pregnant
[ 6  1  8  0  5  3 10  2  4  7  9 11 13 15 17 12 14]

Plasma glucose concentration a 2 hours in an oral glucose tolerance test
[148  85 183  89 137 116  78 115 197 125 110 168 139 189 166 100 118 107
 103 126  99 196 119 143 147  97 145 117 109 158  88  92 122 138 102  90
 111 180 133 106 171 159 146  71 105 101 176 150  73 187  84  44 141 114
  95 129  79   0  62 131 112 113  74  83 136  80 123  81 134 142 144  93
 163 151  96 155  76 160 124 162 132 120 173 170 128 108 154  57 156 153
 188 152 104  87  75 179 130 194 181 135 184 140 177 164  91 165  86 193
 191 161 167  77 182 157 178  61  98 127  82  72 172  94 175 195  68 186
 198 121  67 174 199  56 169 149  65 190]

Diastolic blood pressure (mm Hg)
[ 72  66  64  40  74  50   0  70  96  92  80  60  84  30  88  90  94  76
  82  75  58  78  68 110  56  62  85  86  48  44  65 108  55 122  54  52
  98 104  95  46 102 100  61  24  38 106 114]

Triceps skin fold thickness (mm)
[35 29  0 23 32 45 19 47 38 30 41 33 26 15 36 11 31 37 42 25 18 24 39 27
 21 34 10 60 13 20 22 28 54 40 51 56 14 17 50 44 12 46 16  7 52 43 48  8
 49 63 99]

2-Hour serum insulin (mu U/ml)
[  0  94 168  88 543 846 175 230  83  96 235 146 115 140 110 245  54 192
 207  70 240  82  36  23 300 342 304 142 128  38 100  90 270  71 125 176
  48  64 228  76 220  40 152  18 135 495  37  51  99 145 225  49  50  92
 325  63 284 119 204 155 485  53 114 105 285 156  78 130  55  58 160 210
 318  44 190 280  87 271 129 120 478  56  32 744 370  45 194 680 402 258
 375 150  67  57 116 278 122 545  75  74 182 360 215 184  42 132 148 180
 205  85 231  29  68  52 255 171  73 108  43 167 249 293  66 465  89 158
```

```
  84  72  59  81 196 415 275 165 579 310  61 474 170 277  60  14  95 237
 191 328 250 480 265 193  79  86 326 188 106  65 166 274  77 126 330 600
 185  25  41 272 321 144  15 183  91  46 440 159 540 200 335 387  22 291
 392 178 127 510  16 112]

Body mass index (weight in kg/(height in m)^2)
[33.6 26.6 23.3 28.1 43.1 25.6 31.  35.3 30.5  0.  37.6 38.  27.1 30.1
 25.8 30.  45.8 29.6 43.3 34.6 39.3 35.4 39.8 29.  36.6 31.1 39.4 23.2
 22.2 34.1 36.  31.6 24.8 19.9 27.6 24.  33.2 32.9 38.2 37.1 34.  40.2
 22.7 45.4 27.4 42.  29.7 28.  39.1 19.4 24.2 24.4 33.7 34.7 23.  37.7
 46.8 40.5 41.5 25.  25.4 32.8 32.5 42.7 19.6 28.9 28.6 43.4 35.1 32.
 24.7 32.6 43.2 22.4 29.3 24.6 48.8 32.4 38.5 26.5 19.1 46.7 23.8 33.9
 20.4 28.7 49.7 39.  26.1 22.5 39.6 29.5 34.3 37.4 33.3 31.2 28.2 53.2
 34.2 26.8 55.  42.9 34.5 27.9 38.3 21.1 33.8 30.8 36.9 39.5 27.3 21.9
 40.6 47.9 50.  25.2 40.9 37.2 44.2 29.9 31.9 28.4 43.5 32.7 67.1 45.
 34.9 27.7 35.9 22.6 33.1 30.4 52.3 24.3 22.9 34.8 30.9 40.1 23.9 37.5
 35.5 42.8 42.6 41.8 35.8 37.8 28.8 23.6 35.7 36.7 45.2 44.  46.2 35.
 43.6 44.1 18.4 29.2 25.9 32.1 36.3 40.  25.1 27.5 45.6 27.8 24.9 25.3
 37.9 27.  26.  38.7 20.8 36.1 30.7 32.3 52.9 21.  39.7 25.5 26.2 19.3
 38.1 23.5 45.5 23.1 39.9 36.8 21.8 41.  42.2 34.4 27.2 36.5 29.8 39.2
 38.4 36.2 48.3 20.  22.3 45.7 23.7 22.1 42.1 42.4 18.2 26.4 45.3 37.
 24.5 32.2 59.4 21.2 26.7 30.2 46.1 41.3 38.8 35.2 42.3 40.7 46.5 33.5
 37.3 30.3 26.3 21.7 36.4 28.5 26.9 38.6 31.3 19.5 20.1 40.8 23.4 28.3
 38.9 57.3 35.6 49.6 44.6 24.1 44.5 41.2 49.3 46.3]

Diabetes pedigree function
[0.627 0.351 0.672 0.167 2.288 0.201 0.248 0.134 0.158 0.232 0.191 0.537
 1.441 0.398 0.587 0.484 0.551 0.254 0.183 0.529 0.704 0.388 0.451 0.263
 0.205 0.257 0.487 0.245 0.337 0.546 0.851 0.267 0.188 0.512 0.966 0.42
 0.665 0.503 1.39  0.271 0.696 0.235 0.721 0.294 1.893 0.564 0.586 0.344
 0.305 0.491 0.526 0.342 0.467 0.718 0.962 1.781 0.173 0.304 0.27  0.699
 0.258 0.203 0.855 0.845 0.334 0.189 0.867 0.411 0.583 0.231 0.396 0.14
 0.391 0.37  0.307 0.102 0.767 0.237 0.227 0.698 0.178 0.324 0.153 0.165
 0.443 0.261 0.277 0.761 0.255 0.13  0.323 0.356 0.325 1.222 0.179 0.262
 0.283 0.93  0.801 0.207 0.287 0.336 0.247 0.199 0.543 0.192 0.588 0.539
 0.22  0.654 0.223 0.759 0.26  0.404 0.186 0.278 0.496 0.452 0.403 0.741
 0.361 1.114 0.457 0.647 0.088 0.597 0.532 0.703 0.159 0.268 0.286 0.318
 0.272 0.572 0.096 1.4   0.218 0.085 0.399 0.432 1.189 0.687 0.137 0.637
 0.833 0.229 0.817 0.204 0.368 0.743 0.722 0.256 0.709 0.471 0.495 0.18
 0.542 0.773 0.678 0.719 0.382 0.319 0.19  0.956 0.084 0.725 0.299 0.244
 0.745 0.615 1.321 0.64  0.142 0.374 0.383 0.578 0.136 0.395 0.187 0.905
 0.15  0.874 0.236 0.787 0.407 0.605 0.151 0.289 0.355 0.29  0.375 0.164
 0.431 0.742 0.514 0.464 1.224 1.072 0.805 0.209 0.666 0.101 0.198 0.652
 2.329 0.089 0.645 0.238 0.394 0.293 0.479 0.686 0.831 0.582 0.446 0.402
 1.318 0.329 1.213 0.427 0.282 0.143 0.38  0.284 0.249 0.926 0.557 0.092
 0.655 1.353 0.612 0.2   0.226 0.997 0.933 1.101 0.078 0.24  1.136 0.128
 0.422 0.251 0.677 0.296 0.454 0.744 0.881 0.28  0.259 0.619 0.808 0.34
 0.434 0.757 0.613 0.692 0.52  0.412 0.84  0.839 0.156 0.215 0.326 1.391
```

```
 0.875 0.313 0.433 0.626 1.127 0.315 0.345 0.129 0.527 0.197 0.731 0.148
 0.123 0.127 0.122 1.476 0.166 0.932 0.343 0.893 0.331 0.472 0.673 0.389
 0.485 0.349 0.279 0.346 0.252 0.243 0.58  0.559 0.302 0.569 0.378 0.385
 0.499 0.306 0.234 2.137 1.731 0.545 0.225 0.816 0.528 0.509 1.021 0.821
 0.947 1.268 0.221 0.66  0.239 0.949 0.444 0.463 0.803 1.6   0.944 0.196
 0.241 0.161 0.135 0.376 1.191 0.702 0.674 1.076 0.534 1.095 0.554 0.624
 0.219 0.507 0.561 0.421 0.516 0.264 0.328 0.233 0.108 1.138 0.147 0.727
 0.435 0.497 0.23  0.955 2.42  0.658 0.33  0.51  0.285 0.415 0.381 0.832
 0.498 0.212 0.364 1.001 0.46  0.733 0.416 0.705 1.022 0.269 0.6   0.571
 0.607 0.17  0.21  0.126 0.711 0.466 0.162 0.419 0.63  0.365 0.536 1.159
 0.629 0.292 0.145 1.144 0.174 0.547 0.163 0.738 0.314 0.968 0.409 0.297
 0.525 0.154 0.771 0.107 0.493 0.717 0.917 0.501 1.251 0.735 0.804 0.661
 0.549 0.825 0.423 1.034 0.16  0.341 0.68  0.591 0.3   0.121 0.502 0.401
 0.601 0.748 0.338 0.43  0.892 0.813 0.693 0.575 0.371 0.206 0.417 1.154
 0.925 0.175 1.699 0.682 0.194 0.4   0.1   1.258 0.482 0.138 0.593 0.878
 0.157 1.282 0.141 0.246 1.698 1.461 0.347 0.362 0.393 0.144 0.732 0.115
 0.465 0.649 0.871 0.149 0.695 0.303 0.61  0.73  0.447 0.455 0.133 0.155
 1.162 1.292 0.182 1.394 0.217 0.631 0.88  0.614 0.332 0.366 0.181 0.828
 0.335 0.856 0.886 0.439 0.253 0.598 0.904 0.483 0.565 0.118 0.177 0.176
 0.295 0.441 0.352 0.826 0.97  0.595 0.317 0.265 0.646 0.426 0.56  0.515
 0.453 0.785 0.734 1.174 0.488 0.358 1.096 0.408 1.182 0.222 1.057 0.766
 0.171]

Age (years)
[50 31 32 21 33 30 26 29 53 54 34 57 59 51 27 41 43 22 38 60 28 45 35 46
 56 37 48 40 25 24 58 42 44 39 36 23 61 69 62 55 65 47 52 66 49 63 67 72
 81 64 70 68]

Class variable
[1 0]
```

[7]:
```
# Removing missing values

df_cleaned = df.copy()

df_cleaned.drop(df_cleaned[df_cleaned["Plasma glucose concentration a 2 hours␣
 ↪in an oral glucose tolerance test"] == 0].index, inplace=True)
df_cleaned.drop(df_cleaned[df_cleaned["Diastolic blood pressure (mm Hg)"] == 0].
 ↪index, inplace=True)
# df_cleaned.drop(df_cleaned[df_cleaned["Triceps skin fold thickness (mm)"] ==␣
 ↪0].index, inplace=True) # too many zeros
# df_cleaned.drop(df_cleaned[df_cleaned["2-Hour serum insulin (mu U/ml)"] == 0].
 ↪index, inplace=True) # too many zeros
df_cleaned.drop(df_cleaned[df_cleaned["Body mass index (weight in kg/(height in␣
 ↪m)^2)"] == 0].index, inplace=True)
```

```
df = df_cleaned.copy()

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 724 entries, 0 to 767
Data columns (total 9 columns):
 #   Column
Non-Null Count  Dtype
---  ------
--------------  -----
 0   Number of times pregnant
724 non-null    int64
 1   Plasma glucose concentration a 2 hours in an oral glucose tolerance test
724 non-null    int64
 2   Diastolic blood pressure (mm Hg)
724 non-null    int64
 3   Triceps skin fold thickness (mm)
724 non-null    int64
 4   2-Hour serum insulin (mu U/ml)
724 non-null    int64
 5   Body mass index (weight in kg/(height in m)^2)
724 non-null    float64
 6   Diabetes pedigree function
724 non-null    float64
 7   Age (years)
724 non-null    int64
 8   Class variable
724 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 56.6 KB
```

[8]:
```
## Handling outliers

df_z = (df - df.mean())/df.std()

(df_z.abs() > 3).sum()
```

[8]:
```
Number of times pregnant                                                    4
Plasma glucose concentration a 2 hours in an oral glucose tolerance test    0
Diastolic blood pressure (mm Hg)                                            8
Triceps skin fold thickness (mm)                                            1
2-Hour serum insulin (mu U/ml)                                             18
Body mass index (weight in kg/(height in m)^2)                              5
Diabetes pedigree function                                                 10
Age (years)                                                                 3
Class variable                                                              0
dtype: int64
```

[9]:
```
# Inspecting outliers

df[df_z.abs()["Diabetes pedigree function"] > 3]
```

[9]:

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)
4	0	137	40	35	168
45	0	180	66	39	0
58	0	146	82	0	0
228	4	197	70	39	744
330	8	118	72	19	0
370	3	173	82	48	465
395	2	127	58	24	
445	0	180	78	63	
593	2	82	52	22	
621	2	92	76	20	

```
395                              275
445                               14
593                              115
621                                0

     Body mass index (weight in kg/(height in m)^2)  \
4                                                43.1
45                                               42.0
58                                               40.5
228                                              36.7
330                                              23.1
370                                              38.4
395                                              27.7
445                                              59.4
593                                              28.5
621                                              24.2

     Diabetes pedigree function  Age (years)  Class variable
4                         2.288           33               1
45                        1.893           25               1
58                        1.781           44               0
228                       2.329           31               0
330                       1.476           46               0
370                       2.137           25               1
395                       1.600           25               0
445                       2.420           25               1
593                       1.699           25               0
621                       1.698           28               0
```

[10]:
```
for col in df.columns:
    print(col)
    print((3 * df[col].std()) + df[col].mean(), (-3 * df[col].std()) + df[col].
 ↪mean())
    print(df[col].describe())
    print()
```

```
Number of times pregnant
13.954430556217089 -6.22238635732206
count    724.000000
mean       3.866022
std        3.362803
min        0.000000
25%        1.000000
50%        3.000000
75%        6.000000
max       17.000000
Name: Number of times pregnant, dtype: float64
```

```
Plasma glucose concentration a 2 hours in an oral glucose tolerance test
214.13268663667898 29.632506733486764
count    724.000000
mean     121.882597
std       30.750030
min       44.000000
25%       99.750000
50%      117.000000
75%      142.000000
max      199.000000
Name: Plasma glucose concentration a 2 hours in an oral glucose tolerance test,
dtype: float64

Diastolic blood pressure (mm Hg)
109.540163451345 35.2609415210307
count    724.000000
mean      72.400552
std       12.379870
min       24.000000
25%       64.000000
50%       72.000000
75%       80.000000
max      122.000000
Name: Diastolic blood pressure (mm Hg), dtype: float64

Triceps skin fold thickness (mm)
68.6416380558638 -25.754897724372075
count    724.000000
mean      21.443370
std       15.732756
min        0.000000
25%        0.000000
50%       24.000000
75%       33.000000
max       99.000000
Name: Triceps skin fold thickness (mm), dtype: float64

2-Hour serum insulin (mu U/ml)
435.54401552792194 -266.5550652516788
count    724.000000
mean      84.494475
std      117.016513
min        0.000000
25%        0.000000
50%       48.000000
75%      130.500000
max      846.000000
Name: 2-Hour serum insulin (mu U/ml), dtype: float64
```

```
Body mass index (weight in kg/(height in m)^2)
53.13394985105282 11.800304292593598
count    724.000000
mean      32.467127
std        6.888941
min       18.200000
25%       27.500000
50%       32.400000
75%       36.600000
max       67.100000
Name: Body mass index (weight in kg/(height in m)^2), dtype: float64

Diabetes pedigree function
1.4717102037537415 -0.52217981701341
count    724.000000
mean       0.474765
std        0.332315
min        0.078000
25%        0.245000
50%        0.379000
75%        0.627500
max        2.420000
Name: Diabetes pedigree function, dtype: float64

Age (years)
68.64700701077848 -1.9453495522149424
count    724.000000
mean      33.350829
std       11.765393
min       21.000000
25%       24.000000
50%       29.000000
75%       41.000000
max       81.000000
Name: Age (years), dtype: float64

Class variable
1.7699547510607738 -1.0821094471933703
count    724.000000
mean       0.343923
std        0.475344
min        0.000000
25%        0.000000
50%        0.000000
75%        1.000000
max        1.000000
Name: Class variable, dtype: float64
```

[11]:
```
# Dropping outliers in relevant columns (according to medical relvance to the␣
 ↪outlier thresholds)

cols = list(df.columns)
print(cols)

df_no_outliers = df.copy()

df_no_outliers.drop(df[(df_z.abs()[cols[:6]] > 3).any(axis=1)].index,␣
 ↪inplace=True)

df = df_no_outliers.copy()

df
```

```
['Number of times pregnant', 'Plasma glucose concentration a 2 hours in an oral
glucose tolerance test', 'Diastolic blood pressure (mm Hg)', 'Triceps skin fold
thickness (mm)', '2-Hour serum insulin (mu U/ml)', 'Body mass index (weight in
kg/(height in m)^2)', 'Diabetes pedigree function', 'Age (years)', 'Class
variable']
```

[11]:

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test
0	6	148
1	1	85
2	8	183
3	1	89
4	0	137
..	…	…
763	10	101
764	2	122
765	5	121
766	1	126
767	1	93

```
     Diastolic blood pressure (mm Hg)  Triceps skin fold thickness (mm)  \
0                                  72                                 35
1                                  66                                 29
2                                  64                                  0
3                                  66                                 23
4                                  40                                 35
..                                 …                                  …
763                                76                                 48
764                                70                                 27
765                                72                                 23
766                                60                                  0
767                                70                                 31

     2-Hour serum insulin (mu U/ml)  \
0                                 0
1                                 0
2                                 0
3                                94
4                               168
..                              …
763                             180
764                               0
765                             112
766                               0
767                               0

     Body mass index (weight in kg/(height in m)^2)  \
0                                              33.6
1                                              26.6
2                                              23.3
3                                              28.1
4                                              43.1
..                                              …
763                                            32.9
764                                            36.8
765                                            26.2
766                                            30.1
767                                            30.4

     Diabetes pedigree function  Age (years)  Class variable
0                         0.627           50               1
1                         0.351           31               0
2                         0.672           32               1
3                         0.167           21               0
4                         2.288           33               1
..                          …            …             …
```

```
763                      0.171          63              0
764                      0.340          27              0
765                      0.245          30              0
766                      0.349          47              1
767                      0.315          23              0

[690 rows x 9 columns]
```

[12]:
```
df.describe()
```

[12]:
```
       Number of times pregnant  \
count                690.000000
mean                   3.837681
std                    3.261648
min                    0.000000
25%                    1.000000
50%                    3.000000
75%                    6.000000
max                   13.000000

       Plasma glucose concentration a 2 hours in an oral glucose tolerance test
\
count                                         690.000000
mean                                          120.288406
std                                            29.826095
min                                            44.000000
25%                                            99.000000
50%                                           115.000000
75%                                           138.750000
max                                           199.000000

       Diastolic blood pressure (mm Hg)  Triceps skin fold thickness (mm)  \
count                        690.000000                        690.000000
mean                          72.231884                         20.727536
std                           11.626686                         15.316818
min                           38.000000                          0.000000
25%                           64.000000                          0.000000
50%                           72.000000                         23.000000
75%                           80.000000                         32.000000
max                          108.000000                         60.000000

       2-Hour serum insulin (mu U/ml)  \
count                      690.000000
mean                        72.400000
std                         90.265707
min                          0.000000
25%                          0.000000
```

```
50%                          42.500000
75%                         126.000000
max                         415.000000

       Body mass index (weight in kg/(height in m)^2)  \
count                                      690.000000
mean                                        32.110580
std                                          6.502858
min                                         18.200000
25%                                         27.300000
50%                                         32.000000
75%                                         36.100000
max                                         52.900000

       Diabetes pedigree function  Age (years)  Class variable
count                  690.000000   690.000000      690.000000
mean                     0.466939    33.257971        0.327536
std                      0.313856    11.706875        0.469655
min                      0.078000    21.000000        0.000000
25%                      0.245000    24.000000        0.000000
50%                      0.372500    29.000000        0.000000
75%                      0.613750    41.000000        1.000000
max                      2.288000    81.000000        1.000000
```

[13]:
```
# Normalizing Values

cols = df.columns
X = df[cols[:-1]]
y = df["Class variable"]

X = (X-X.mean())/X.std()

X.describe()
```

[13]:
```
       Number of times pregnant  \
count              6.900000e+02
mean               3.346759e-17
std                1.000000e+00
min               -1.176608e+00
25%               -8.700146e-01
50%               -2.568276e-01
75%                6.629529e-01
max                2.809107e+00

       Plasma glucose concentration a 2 hours in an oral glucose tolerance test
\
count                                       6.900000e+02
```

mean	6.950962e-17
std	1.000000e+00
min	-2.557774e+00
25%	-7.137510e-01
50%	-1.773080e-01
75%	6.189746e-01
max	2.639018e+00

	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)
count	6.900000e+02	6.900000e+02
mean	-9.525392e-17	-1.081261e-16
std	1.000000e+00	1.000000e+00
min	-2.944251e+00	-1.353253e+00
25%	-7.080164e-01	-1.353253e+00
50%	-1.994412e-02	1.483640e-01
75%	6.681281e-01	7.359534e-01
max	3.076381e+00	2.564009e+00

	2-Hour serum insulin (mu U/ml)
count	6.900000e+02
mean	-9.267949e-17
std	1.000000e+00
min	-8.020765e-01
25%	-8.020765e-01
50%	-3.312443e-01
75%	5.938025e-01
max	3.795461e+00

	Body mass index (weight in kg/(height in m)^2)
count	6.900000e+02
mean	2.986339e-16
std	1.000000e+00
min	-2.139149e+00
25%	-7.397639e-01
50%	-1.700479e-02
75%	6.134872e-01
max	3.196967e+00

	Diabetes pedigree function	Age (years)
count	6.900000e+02	6.900000e+02
mean	1.081261e-16	4.569614e-17
std	1.000000e+00	1.000000e+00
min	-1.239230e+00	-1.047075e+00
25%	-7.071377e-01	-7.908149e-01
50%	-3.008999e-01	-3.637154e-01
75%	4.677657e-01	6.613233e-01
max	5.802225e+00	4.078119e+00

[14]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪shuffle = True, random_state = 42)
```

0.4 Tuning, Training & Evaluation

[15]:
```
store = []
def eval_and_collect(name, model, params, method="grid"):
    global store

    assert method in ["grid", "random"], "Invalid Method"

    if method == "grid":
        grid_search = GridSearchCV(estimator=model, param_grid=params,␣
↪verbose=1, cv=5, n_jobs=-1, scoring="f1")

        grid_result = grid_search.fit(X_train, y_train)
        print(grid_search.best_params_)

        y_pred=grid_result.predict(X_test)

    elif method == "random":
        random_search = RandomizedSearchCV(estimator=model,␣
↪param_distributions=params, verbose=1, cv=5, n_iter=400, n_jobs=-1,␣
↪random_state=42, scoring="f1")

        random_result = random_search.fit(X_train, y_train)
        print(random_search.best_params_)

        y_pred=random_result.predict(X_test)

    print(classification_report(y_test, y_pred, target_names=["Non-Diabetic",␣
↪"Diabetic"]))
    metrics = classification_report(y_test, y_pred,␣
↪target_names=["Non-Diabetic", "Diabetic"], output_dict=True)
    store.append({"Name": name, "model": model, "metrics": metrics})
```

0.4.1 SVM

[16]:
```
svm = svm.SVC()
params_grid = {
    "kernel":["linear", "rbf", "sigmoid", "poly"],

    "C": [0.1, 1, 10],
    "gamma": ["scale", "auto"],
    "degree": [2, 3],

    "class_weight":["balanced"]
```

```
}
eval_and_collect("SVM", svm, params_grid, "grid")
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
{'C': 1, 'class_weight': 'balanced', 'degree': 2, 'gamma': 'scale', 'kernel':
'rbf'}
              precision    recall  f1-score   support

Non-Diabetic       0.87      0.77      0.82        92
    Diabetic       0.62      0.76      0.69        46

    accuracy                           0.77       138
   macro avg       0.75      0.77      0.75       138
weighted avg       0.79      0.77      0.77       138
```

0.4.2 KNN

[17]:
```
knn = KNeighborsClassifier()
params_grid = {
    "n_neighbors":range(4,31)
}
eval_and_collect("KNN", knn, params_grid, "grid")
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
{'n_neighbors': 9}
              precision    recall  f1-score   support

Non-Diabetic       0.77      0.89      0.82        92
    Diabetic       0.68      0.46      0.55        46

    accuracy                           0.75       138
   macro avg       0.72      0.67      0.68       138
weighted avg       0.74      0.75      0.73       138
```

0.4.3 Logistic Regression

[18]:
```
lrg = LogisticRegression(solver="saga")
params_grid = [
    {'penalty': ['l1'], 'C': [0.1, 1, 10], 'solver': ['saga'], "class_weight":
 ↪["balanced"],"random_state":[42]},
    {'penalty': ['l2'], 'C': [0.1, 1, 10], 'solver': ['saga'], "class_weight":
 ↪["balanced"],"random_state":[42]},
    {'penalty': ['elasticnet'], 'C': [0.1, 1, 10], 'l1_ratio': [0.1, 0.5, 0.9],␣
 ↪'solver': ['saga'], "class_weight":["balanced"],"random_state":[42]},
]
```

```
eval_and_collect("Logistic Regression", lrg, params_grid, "grid")
```

```
Fitting 5 folds for each of 15 candidates, totalling 75 fits
{'C': 0.1, 'class_weight': 'balanced', 'l1_ratio': 0.9, 'penalty': 'elasticnet',
'random_state': 42, 'solver': 'saga'}
              precision    recall  f1-score   support

Non-Diabetic       0.84      0.83      0.84        92
    Diabetic       0.67      0.70      0.68        46

    accuracy                           0.78       138
   macro avg       0.76      0.76      0.76       138
weighted avg       0.79      0.78      0.78       138
```

0.4.4 Decision Tree

[19]:
```
clf = DecisionTreeClassifier()
params_grid = {
    "max_depth":np.linspace(1,21).astype(np.int32),
    "class_weight":["balanced"],
    "random_state":[42]
}
eval_and_collect("Decision Tree", clf, params_grid, "grid")
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
{'class_weight': 'balanced', 'max_depth': np.int32(5), 'random_state': 42}
              precision    recall  f1-score   support

Non-Diabetic       0.89      0.64      0.75        92
    Diabetic       0.54      0.85      0.66        46

    accuracy                           0.71       138
   macro avg       0.72      0.74      0.70       138
weighted avg       0.78      0.71      0.72       138
```

0.4.5 Random Forest

[ ]:
```
rf = RandomForestClassifier()
param_distributions = {
    'n_estimators': randint(100, 300),       # number of trees
    'max_depth': randint(2,15),
    'min_samples_split': randint(2, 10),    # min samples to split a node
    'min_samples_leaf': randint(1, 5),      # min samples in a leaf
    'max_features': ['sqrt', 'log2', None],  # features to consider at split
    "class_weight": ["balanced"],
```

```
    "random_state":[42],
}
eval_and_collect("Random Forest", rf, param_distributions, "random")
# best results found: {'class_weight': 'balanced', 'max_depth': 3,␣
 ↪'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 3,␣
 ↪'n_estimators': 152, 'random_state': 42}
```

```
Fitting 5 folds for each of 400 candidates, totalling 2000 fits
{'class_weight': 'balanced', 'max_depth': 3, 'max_features': 'log2',
'min_samples_leaf': 1, 'min_samples_split': 3, 'n_estimators': 152,
'random_state': 42}
              precision    recall  f1-score   support

Non-Diabetic       0.87      0.78      0.82        92
    Diabetic       0.64      0.76      0.69        46

    accuracy                           0.78       138
   macro avg       0.75      0.77      0.76       138
weighted avg       0.79      0.78      0.78       138
```

0.4.6 XGBoost

[ ]:
```
pos_weight = np.sum(y_train == 0) / np.sum(y_train == 1)
xgb_model = XGBClassifier()
param_distributions = {
    "n_estimators":randint(100,300),        # number of boosting rounds
    "learning_rate":loguniform(1e-5,1e-1),      # step size shrinkage
    "max_depth":randint(2,15),             # max depth of trees
    "subsample":uniform(.7,.2),           # fraction of samples per tree
    "colsample_bytree":uniform(.7,.2),    # fraction of features per tree
    "scale_pos_weight":[pos_weight],

    "eval_metric":["logloss"],
    "random_state":[42],
}
eval_and_collect("XGBoost", xgb_model, param_distributions, "random")
# best results found: {'colsample_bytree': np.float64(0.8972421488959206),␣
 ↪'eval_metric': 'logloss', 'learning_rate': np.float64(0.
 ↪006203643835492842 4), 'max_depth': 2, 'n_estimators': 260, 'random_state':␣
 ↪42, 'scale_pos_weight': np.float64(2.066666666666667), 'subsample': np.
 ↪float64(0.7822413441744372)}
```

```
Fitting 5 folds for each of 400 candidates, totalling 2000 fits
{'colsample_bytree': np.float64(0.8972421488959206), 'eval_metric': 'logloss',
'learning_rate': np.float64(0.0062036438354928424), 'max_depth': 2,
'n_estimators': 260, 'random_state': 42, 'scale_pos_weight':
np.float64(2.066666666666667), 'subsample': np.float64(0.7822413441744372)}
```

```
              precision    recall  f1-score   support

Non-Diabetic       0.89      0.78      0.83        92
    Diabetic       0.65      0.80      0.72        46

    accuracy                           0.79       138
   macro avg       0.77      0.79      0.78       138
weighted avg       0.81      0.79      0.79       138
```

0.5 Evaluation

[34]:
```
flattened_store = {}
for item in store:
    combined = {}
    for k,v in item["metrics"].items():
        if isinstance(v, dict):
            for sub_key, sub_value in v.items():
                combined[f"{k}_{sub_key}"] = sub_value
        else:
            combined[k] = v

    flattened_store[item["Name"]] = combined
flattened_df = pd.DataFrame.from_dict(flattened_store, orient="index")
flattened_df
```

[34]:
```
                     Non-Diabetic_precision  Non-Diabetic_recall  \
SVM                                0.865854             0.771739
KNN                                0.766355             0.891304
Logistic Regression                0.844444             0.826087
Decision Tree                      0.893939             0.641304
Random Forest                      0.867470             0.782609
XGBoost                            0.888889             0.782609

                     Non-Diabetic_f1-score  Non-Diabetic_support  \
SVM                               0.816092                  92.0
KNN                               0.824121                  92.0
Logistic Regression               0.835165                  92.0
Decision Tree                     0.746835                  92.0
Random Forest                     0.822857                  92.0
XGBoost                           0.832370                  92.0

                     Diabetic_precision  Diabetic_recall  Diabetic_f1-score  \
SVM                            0.625000         0.760870           0.686275
KNN                            0.677419         0.456522           0.545455
Logistic Regression            0.666667         0.695652           0.680851
Decision Tree                  0.541667         0.847826           0.661017
```

```
Random Forest                  0.636364         0.760870           0.693069
XGBoost                        0.649123         0.804348           0.718447

                     Diabetic_support  accuracy  macro avg_precision  \
SVM                              46.0  0.768116             0.745427
KNN                              46.0  0.746377             0.721887
Logistic Regression              46.0  0.782609             0.755556
Decision Tree                    46.0  0.710145             0.717803
Random Forest                    46.0  0.775362             0.751917
XGBoost                          46.0  0.789855             0.769006

                     macro avg_recall  macro avg_f1-score  macro avg_support  \
SVM                          0.766304            0.751183              138.0
KNN                          0.673913            0.684788              138.0
Logistic Regression          0.760870            0.758008              138.0
Decision Tree                0.744565            0.703926              138.0
Random Forest                0.771739            0.757963              138.0
XGBoost                      0.793478            0.775408              138.0

                     weighted avg_precision  weighted avg_recall  \
SVM                                0.785569             0.768116
KNN                                0.736710             0.746377
Logistic Regression                0.785185             0.782609
Decision Tree                      0.776515             0.710145
Random Forest                      0.790434             0.775362
XGBoost                            0.808967             0.789855

                     weighted avg_f1-score  weighted avg_support
SVM                               0.772819                 138.0
KNN                               0.731232                 138.0
Logistic Regression               0.783727                 138.0
Decision Tree                     0.718229                 138.0
Random Forest                     0.779595                 138.0
XGBoost                           0.794395                 138.0
```

[37]:
```
summary = flattened_df.sort_values("weighted avg_f1-score", ascending=False)
summary["weighted avg_f1-score"]
```

[37]:
```
XGBoost                0.794395
Logistic Regression    0.783727
Random Forest          0.779595
SVM                    0.772819
KNN                    0.731232
Decision Tree          0.718229
Name: weighted avg_f1-score, dtype: float64
```

[38]:
```
summary
```

[38]:

	Non-Diabetic_precision	Non-Diabetic_recall	Non-Diabetic_f1-score	Non-Diabetic_support	Diabetic_precision	Diabetic_recall	Diabetic_f1-score	Diabetic_support	accuracy	macro avg_precision	macro avg_recall	macro avg_f1-score	macro avg_support	weighted avg_precision	weighted avg_recall
XGBoost	0.888889	0.782609	0.832370	92.0	0.649123	0.804348	0.718447	46.0	0.789855	0.769006	0.793478	0.775408	138.0	0.808967	0.789855
Logistic Regression	0.844444	0.826087	0.835165	92.0	0.666667	0.695652	0.680851	46.0	0.782609	0.755556	0.760870	0.758008	138.0	0.785185	0.782609
Random Forest	0.867470	0.782609	0.822857	92.0	0.636364	0.760870	0.693069	46.0	0.775362	0.751917	0.771739	0.757963	138.0	0.790434	0.775362
SVM	0.865854	0.771739	0.816092	92.0	0.625000	0.760870	0.686275	46.0	0.768116	0.745427	0.766304	0.751183	138.0	0.785569	0.768116
KNN	0.766355	0.891304	0.824121	92.0	0.677419	0.456522	0.545455	46.0	0.746377	0.721887	0.673913	0.684788	138.0	0.736710	0.746377
Decision Tree	0.893939	0.641304	0.746835	92.0	0.541667	0.847826	0.661017	46.0	0.710145	0.717803	0.744565	0.703926	138.0	0.776515	0.710145

	weighted avg_f1-score	weighted avg_support
XGBoost	0.794395	138.0
Logistic Regression	0.783727	138.0
Random Forest	0.779595	138.0
SVM	0.772819	138.0
KNN	0.731232	138.0
Decision Tree	0.718229	138.0

XGBoost was found to be the best model (according the metric of weighted avg f1-score), although most of the results are comparable