project

August 22, 2025

1 Breast Cancer Prediction Project

This project is the result of a general classification problem - determining whether a female has breast cancer or not.

[ ]:

```

```

2 Importing Project Tools and Libraries

Getting all the necessary project tools and libraries for the project at the start is essential for a concise and efficient project workflow. All the libraries used in this project are mentioned and imported in the below cell.

[60]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")

np.random.seed(seed = 2)

import sklearn
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪RandomizedSearchCV, GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,␣
↪recall_score, f1_score

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
```

```python
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer

from joblib import dump, load
```

[ ]:

```python

```

3 Fetching the Dataset

The dataset to be fed to the respective Machine Learning Models is fetched. Notice that it is part of the built-in dataset as provided by sklearn. It is fetched in the form of a python bunch object and is thus shifted into a Pandas DataFrame

[61]:

```python
obj = load_breast_cancer()

dataFrame = pd.DataFrame(obj.data, columns = obj.feature_names)

dataFrame["target"] = obj.target

dataFrame
```

[61]:

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0          17.99         10.38          122.80     1001.0          0.11840
1          20.57         17.77          132.90     1326.0          0.08474
2          19.69         21.25          130.00     1203.0          0.10960
3          11.42         20.38           77.58      386.1          0.14250
4          20.29         14.34          135.10     1297.0          0.10030
..           …             …               …          …                …
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0             0.27760         0.30010              0.14710         0.2419
1             0.07864         0.08690              0.07017         0.1812
2             0.15990         0.19740              0.12790         0.2069
3             0.28390         0.24140              0.10520         0.2597
4             0.13280         0.19800              0.10430         0.1809
..                …               …                    …              …
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
567           0.27700         0.35140              0.15200         0.2397
568           0.04362         0.00000              0.00000         0.1587
```

```
     mean fractal dimension  …  worst texture  worst perimeter  worst area  \
0                   0.07871  …          17.33           184.60      2019.0
1                   0.05667  …          23.41           158.80      1956.0
2                   0.05999  …          25.53           152.50      1709.0
3                   0.09744  …          26.50            98.87       567.7
4                   0.05883  …          16.67           152.20      1575.0
..                      …  …              …                …           …
564                 0.05623  …          26.40           166.10      2027.0
565                 0.05533  …          38.25           155.00      1731.0
566                 0.05648  …          34.12           126.70      1124.0
567                 0.07016  …          39.42           184.60      1821.0
568                 0.05884  …          30.37            59.16       268.6

     worst smoothness  worst compactness  worst concavity  \
0             0.16220            0.66560           0.7119
1             0.12380            0.18660           0.2416
2             0.14440            0.42450           0.4504
3             0.20980            0.86630           0.6869
4             0.13740            0.20500           0.4000
..                …                  …                …
564           0.14100            0.21130           0.4107
565           0.11660            0.19220           0.3215
566           0.11390            0.30940           0.3403
567           0.16500            0.86810           0.9387
568           0.08996            0.06444           0.0000

     worst concave points  worst symmetry  worst fractal dimension  target
0                  0.2654          0.4601                  0.11890       0
1                  0.1860          0.2750                  0.08902       0
2                  0.2430          0.3613                  0.08758       0
3                  0.2575          0.6638                  0.17300       0
4                  0.1625          0.2364                  0.07678       0
..                    …             …                      …      …
564                0.2216          0.2060                  0.07115       0
565                0.1628          0.2572                  0.06637       0
566                0.1418          0.2218                  0.07820       0
567                0.2650          0.4087                  0.12400       0
568                0.0000          0.2871                  0.07039       1

[569 rows x 31 columns]
```

[ ]:

4 Getting insights from the Dataset

The first and most important step is to find out the shape and form in which the dataset has been presented to us. It involves things like checking for missing values, analyzing data types and sample size, analyzing the cardinalities of attributes, determining attribute relationships etc.

[62]:
```
dataFrame.isna().sum()
```

[62]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

[63]:
```
correlation = dataFrame.corr()

correlation
```

[63]:
```
                         mean radius  mean texture  mean perimeter  mean area  \
mean radius                 1.000000      0.323782        0.997855   0.987357
mean texture                0.323782      1.000000        0.329533   0.321086
mean perimeter              0.997855      0.329533        1.000000   0.986507
mean area                   0.987357      0.321086        0.986507   1.000000
mean smoothness             0.170581     -0.023389        0.207278   0.177028
mean compactness            0.506124      0.236702        0.556936   0.498502
```

mean concavity	0.676764	0.302418	0.716136	0.685983
mean concave points	0.822529	0.293464	0.850977	0.823269
mean symmetry	0.147741	0.071401	0.183027	0.151293
mean fractal dimension	-0.311631	-0.076437	-0.261477	-0.283110
radius error	0.679090	0.275869	0.691765	0.732562
texture error	-0.097317	0.386358	-0.086761	-0.066280
perimeter error	0.674172	0.281673	0.693135	0.726628
area error	0.735864	0.259845	0.744983	0.800086
smoothness error	-0.222600	0.006614	-0.202694	-0.166777
compactness error	0.206000	0.191975	0.250744	0.212583
concavity error	0.194204	0.143293	0.228082	0.207660
concave points error	0.376169	0.163851	0.407217	0.372320
symmetry error	-0.104321	0.009127	-0.081629	-0.072497
fractal dimension error	-0.042641	0.054458	-0.005523	-0.019887
worst radius	0.969539	0.352573	0.969476	0.962746
worst texture	0.297008	0.912045	0.303038	0.287489
worst perimeter	0.965137	0.358040	0.970387	0.959120
worst area	0.941082	0.343546	0.941550	0.959213
worst smoothness	0.119616	0.077503	0.150549	0.123523
worst compactness	0.413463	0.277830	0.455774	0.390410
worst concavity	0.526911	0.301025	0.563879	0.512606
worst concave points	0.744214	0.295316	0.771241	0.722017
worst symmetry	0.163953	0.105008	0.189115	0.143570
worst fractal dimension	0.007066	0.119205	0.051019	0.003738
target	-0.730029	-0.415185	-0.742636	-0.708984

	mean smoothness	mean compactness	mean concavity \
mean radius	0.170581	0.506124	0.676764
mean texture	-0.023389	0.236702	0.302418
mean perimeter	0.207278	0.556936	0.716136
mean area	0.177028	0.498502	0.685983
mean smoothness	1.000000	0.659123	0.521984
mean compactness	0.659123	1.000000	0.883121
mean concavity	0.521984	0.883121	1.000000
mean concave points	0.553695	0.831135	0.921391
mean symmetry	0.557775	0.602641	0.500667
mean fractal dimension	0.584792	0.565369	0.336783
radius error	0.301467	0.497473	0.631925
texture error	0.068406	0.046205	0.076218
perimeter error	0.296092	0.548905	0.660391
area error	0.246552	0.455653	0.617427
smoothness error	0.332375	0.135299	0.098564
compactness error	0.318943	0.738722	0.670279
concavity error	0.248396	0.570517	0.691270
concave points error	0.380676	0.642262	0.683260
symmetry error	0.200774	0.229977	0.178009
fractal dimension error	0.283607	0.507318	0.449301

```
worst radius                  0.213120          0.535315        0.688236
worst texture                 0.036072          0.248133        0.299879
worst perimeter               0.238853          0.590210        0.729565
worst area                    0.206718          0.509604        0.675987
worst smoothness              0.805324          0.565541        0.448822
worst compactness             0.472468          0.865809        0.754968
worst concavity               0.434926          0.816275        0.884103
worst concave points          0.503053          0.815573        0.861323
worst symmetry                0.394309          0.510223        0.409464
worst fractal dimension       0.499316          0.687382        0.514930
target                       -0.358560         -0.596534       -0.696360

                         mean concave points  mean symmetry  \
mean radius                         0.822529       0.147741
mean texture                        0.293464       0.071401
mean perimeter                      0.850977       0.183027
mean area                           0.823269       0.151293
mean smoothness                     0.553695       0.557775
mean compactness                    0.831135       0.602641
mean concavity                      0.921391       0.500667
mean concave points                 1.000000       0.462497
mean symmetry                       0.462497       1.000000
mean fractal dimension              0.166917       0.479921
radius error                        0.698050       0.303379
texture error                       0.021480       0.128053
perimeter error                     0.710650       0.313893
area error                          0.690299       0.223970
smoothness error                    0.027653       0.187321
compactness error                   0.490424       0.421659
concavity error                     0.439167       0.342627
concave points error                0.615634       0.393298
symmetry error                      0.095351       0.449137
fractal dimension error             0.257584       0.331786
worst radius                        0.830318       0.185728
worst texture                       0.292752       0.090651
worst perimeter                     0.855923       0.219169
worst area                          0.809630       0.177193
worst smoothness                    0.452753       0.426675
worst compactness                   0.667454       0.473200
worst concavity                     0.752399       0.433721
worst concave points                0.910155       0.430297
worst symmetry                      0.375744       0.699826
worst fractal dimension             0.368661       0.438413
target                             -0.776614      -0.330499

                         mean fractal dimension  …  worst texture  \
mean radius                           -0.311631  …       0.297008
```

mean texture	-0.076437	…	0.912045
mean perimeter	-0.261477	…	0.303038
mean area	-0.283110	…	0.287489
mean smoothness	0.584792	…	0.036072
mean compactness	0.565369	…	0.248133
mean concavity	0.336783	…	0.299879
mean concave points	0.166917	…	0.292752
mean symmetry	0.479921	…	0.090651
mean fractal dimension	1.000000	…	-0.051269
radius error	0.000111	…	0.194799
texture error	0.164174	…	0.409003
perimeter error	0.039830	…	0.200371
area error	-0.090170	…	0.196497
smoothness error	0.401964	…	-0.074743
compactness error	0.559837	…	0.143003
concavity error	0.446630	…	0.100241
concave points error	0.341198	…	0.086741
symmetry error	0.345007	…	-0.077473
fractal dimension error	0.688132	…	-0.003195
worst radius	-0.253691	…	0.359921
worst texture	-0.051269	…	1.000000
worst perimeter	-0.205151	…	0.365098
worst area	-0.231854	…	0.345842
worst smoothness	0.504942	…	0.225429
worst compactness	0.458798	…	0.360832
worst concavity	0.346234	…	0.368366
worst concave points	0.175325	…	0.359755
worst symmetry	0.334019	…	0.233027
worst fractal dimension	0.767297	…	0.219122
target	0.012838	…	-0.456903

	worst perimeter	worst area	worst smoothness	\
mean radius	0.965137	0.941082	0.119616	
mean texture	0.358040	0.343546	0.077503	
mean perimeter	0.970387	0.941550	0.150549	
mean area	0.959120	0.959213	0.123523	
mean smoothness	0.238853	0.206718	0.805324	
mean compactness	0.590210	0.509604	0.565541	
mean concavity	0.729565	0.675987	0.448822	
mean concave points	0.855923	0.809630	0.452753	
mean symmetry	0.219169	0.177193	0.426675	
mean fractal dimension	-0.205151	-0.231854	0.504942	
radius error	0.719684	0.751548	0.141919	
texture error	-0.102242	-0.083195	-0.073658	
perimeter error	0.721031	0.730713	0.130054	
area error	0.761213	0.811408	0.125389	
smoothness error	-0.217304	-0.182195	0.314457	

```
compactness error              0.260516   0.199371          0.227394
concavity error                0.226680   0.188353          0.168481
concave points error           0.394999   0.342271          0.215351
symmetry error                -0.103753  -0.110343         -0.012662
fractal dimension error       -0.001000  -0.022736          0.170568
worst radius                   0.993708   0.984015          0.216574
worst texture                  0.365098   0.345842          0.225429
worst perimeter                1.000000   0.977578          0.236775
worst area                     0.977578   1.000000          0.209145
worst smoothness               0.236775   0.209145          1.000000
worst compactness              0.529408   0.438296          0.568187
worst concavity                0.618344   0.543331          0.518523
worst concave points           0.816322   0.747419          0.547691
worst symmetry                 0.269493   0.209146          0.493838
worst fractal dimension        0.138957   0.079647          0.617624
target                        -0.782914  -0.733825         -0.421465

                         worst compactness  worst concavity  \
mean radius                       0.413463         0.526911
mean texture                      0.277830         0.301025
mean perimeter                    0.455774         0.563879
mean area                         0.390410         0.512606
mean smoothness                   0.472468         0.434926
mean compactness                  0.865809         0.816275
mean concavity                    0.754968         0.884103
mean concave points               0.667454         0.752399
mean symmetry                     0.473200         0.433721
mean fractal dimension            0.458798         0.346234
radius error                      0.287103         0.380585
texture error                    -0.092439        -0.068956
perimeter error                   0.341919         0.418899
area error                        0.283257         0.385100
smoothness error                 -0.055558        -0.058298
compactness error                 0.678780         0.639147
concavity error                   0.484858         0.662564
concave points error              0.452888         0.549592
symmetry error                    0.060255         0.037119
fractal dimension error           0.390159         0.379975
worst radius                      0.475820         0.573975
worst texture                     0.360832         0.368366
worst perimeter                   0.529408         0.618344
worst area                        0.438296         0.543331
worst smoothness                  0.568187         0.518523
worst compactness                 1.000000         0.892261
worst concavity                   0.892261         1.000000
worst concave points              0.801080         0.855434
worst symmetry                    0.614441         0.532520
```

```
worst fractal dimension          0.810455        0.686511
target                          -0.590998       -0.659610

                         worst concave points  worst symmetry  \
mean radius                          0.744214        0.163953
mean texture                         0.295316        0.105008
mean perimeter                       0.771241        0.189115
mean area                            0.722017        0.143570
mean smoothness                      0.503053        0.394309
mean compactness                     0.815573        0.510223
mean concavity                       0.861323        0.409464
mean concave points                  0.910155        0.375744
mean symmetry                        0.430297        0.699826
mean fractal dimension               0.175325        0.334019
radius error                         0.531062        0.094543
texture error                       -0.119638       -0.128215
perimeter error                      0.554897        0.109930
area error                           0.538166        0.074126
smoothness error                    -0.102007       -0.107342
compactness error                    0.483208        0.277878
concavity error                      0.440472        0.197788
concave points error                 0.602450        0.143116
symmetry error                      -0.030413        0.389402
fractal dimension error              0.215204        0.111094
worst radius                         0.787424        0.243529
worst texture                        0.359755        0.233027
worst perimeter                      0.816322        0.269493
worst area                           0.747419        0.209146
worst smoothness                     0.547691        0.493838
worst compactness                    0.801080        0.614441
worst concavity                      0.855434        0.532520
worst concave points                 1.000000        0.502528
worst symmetry                       0.502528        1.000000
worst fractal dimension              0.511114        0.537848
target                              -0.793566       -0.416294

                         worst fractal dimension    target
mean radius                             0.007066 -0.730029
mean texture                            0.119205 -0.415185
mean perimeter                          0.051019 -0.742636
mean area                               0.003738 -0.708984
mean smoothness                         0.499316 -0.358560
mean compactness                        0.687382 -0.596534
mean concavity                          0.514930 -0.696360
mean concave points                     0.368661 -0.776614
mean symmetry                           0.438413 -0.330499
mean fractal dimension                  0.767297  0.012838
```

```
radius error                                      0.049559 -0.567134
texture error                                    -0.045655  0.008303
perimeter error                                   0.085433 -0.556141
area error                                        0.017539 -0.548236
smoothness error                                  0.101480  0.067016
compactness error                                 0.590973 -0.292999
concavity error                                   0.439329 -0.253730
concave points error                              0.310655 -0.408042
symmetry error                                    0.078079  0.006522
fractal dimension error                           0.591328 -0.077972
worst radius                                      0.093492 -0.776454
worst texture                                     0.219122 -0.456903
worst perimeter                                   0.138957 -0.782914
worst area                                        0.079647 -0.733825
worst smoothness                                  0.617624 -0.421465
worst compactness                                 0.810455 -0.590998
worst concavity                                   0.686511 -0.659610
worst concave points                              0.511114 -0.793566
worst symmetry                                    0.537848 -0.416294
worst fractal dimension                           1.000000 -0.323872
target                                           -0.323872  1.000000

[31 rows x 31 columns]
```

[ ]:

4.0.1 Eliminating useless columns as interpreted from correlation

[64]:

```
frameCols = [att for att in dataFrame]

col = len(correlation) - 1

for i in range(0, len(correlation)):
    if abs(correlation[frameCols[i]]["target"]) <= 0.2:
        dataFrame = dataFrame.drop({frameCols[i]}, axis = 1)
```

[65]:

```
dataFrame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 26 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   mean radius               569 non-null    float64
 1   mean texture              569 non-null    float64
 2   mean perimeter            569 non-null    float64
 3   mean area                 569 non-null    float64
 4   mean smoothness           569 non-null    float64
```

```
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   radius error             569 non-null    float64
 10  perimeter error          569 non-null    float64
 11  area error               569 non-null    float64
 12  compactness error        569 non-null    float64
 13  concavity error          569 non-null    float64
 14  concave points error     569 non-null    float64
 15  worst radius             569 non-null    float64
 16  worst texture            569 non-null    float64
 17  worst perimeter          569 non-null    float64
 18  worst area               569 non-null    float64
 19  worst smoothness         569 non-null    float64
 20  worst compactness        569 non-null    float64
 21  worst concavity          569 non-null    float64
 22  worst concave points     569 non-null    float64
 23  worst symmetry           569 non-null    float64
 24  worst fractal dimension  569 non-null    float64
 25  target                   569 non-null    int64
dtypes: float64(25), int64(1)
memory usage: 115.7 KB
```

[66]:
```
correlation = dataFrame.corr()

correlation
```

[66]:
```
                         mean radius  mean texture  mean perimeter  mean area  \
mean radius                 1.000000      0.323782        0.997855   0.987357
mean texture                0.323782      1.000000        0.329533   0.321086
mean perimeter              0.997855      0.329533        1.000000   0.986507
mean area                   0.987357      0.321086        0.986507   1.000000
mean smoothness             0.170581     -0.023389        0.207278   0.177028
mean compactness            0.506124      0.236702        0.556936   0.498502
mean concavity              0.676764      0.302418        0.716136   0.685983
mean concave points         0.822529      0.293464        0.850977   0.823269
mean symmetry               0.147741      0.071401        0.183027   0.151293
radius error                0.679090      0.275869        0.691765   0.732562
perimeter error             0.674172      0.281673        0.693135   0.726628
area error                  0.735864      0.259845        0.744983   0.800086
compactness error           0.206000      0.191975        0.250744   0.212583
concavity error             0.194204      0.143293        0.228082   0.207660
concave points error        0.376169      0.163851        0.407217   0.372320
worst radius                0.969539      0.352573        0.969476   0.962746
worst texture               0.297008      0.912045        0.303038   0.287489
worst perimeter             0.965137      0.358040        0.970387   0.959120
```

worst area	0.941082	0.343546	0.941550	0.959213
worst smoothness	0.119616	0.077503	0.150549	0.123523
worst compactness	0.413463	0.277830	0.455774	0.390410
worst concavity	0.526911	0.301025	0.563879	0.512606
worst concave points	0.744214	0.295316	0.771241	0.722017
worst symmetry	0.163953	0.105008	0.189115	0.143570
worst fractal dimension	0.007066	0.119205	0.051019	0.003738
target	-0.730029	-0.415185	-0.742636	-0.708984

	mean smoothness	mean compactness	mean concavity \
mean radius	0.170581	0.506124	0.676764
mean texture	-0.023389	0.236702	0.302418
mean perimeter	0.207278	0.556936	0.716136
mean area	0.177028	0.498502	0.685983
mean smoothness	1.000000	0.659123	0.521984
mean compactness	0.659123	1.000000	0.883121
mean concavity	0.521984	0.883121	1.000000
mean concave points	0.553695	0.831135	0.921391
mean symmetry	0.557775	0.602641	0.500667
radius error	0.301467	0.497473	0.631925
perimeter error	0.296092	0.548905	0.660391
area error	0.246552	0.455653	0.617427
compactness error	0.318943	0.738722	0.670279
concavity error	0.248396	0.570517	0.691270
concave points error	0.380676	0.642262	0.683260
worst radius	0.213120	0.535315	0.688236
worst texture	0.036072	0.248133	0.299879
worst perimeter	0.238853	0.590210	0.729565
worst area	0.206718	0.509604	0.675987
worst smoothness	0.805324	0.565541	0.448822
worst compactness	0.472468	0.865809	0.754968
worst concavity	0.434926	0.816275	0.884103
worst concave points	0.503053	0.815573	0.861323
worst symmetry	0.394309	0.510223	0.409464
worst fractal dimension	0.499316	0.687382	0.514930
target	-0.358560	-0.596534	-0.696360

	mean concave points	mean symmetry	radius error \
mean radius	0.822529	0.147741	0.679090
mean texture	0.293464	0.071401	0.275869
mean perimeter	0.850977	0.183027	0.691765
mean area	0.823269	0.151293	0.732562
mean smoothness	0.553695	0.557775	0.301467
mean compactness	0.831135	0.602641	0.497473
mean concavity	0.921391	0.500667	0.631925
mean concave points	1.000000	0.462497	0.698050
mean symmetry	0.462497	1.000000	0.303379

radius error	0.698050	0.303379	1.000000
perimeter error	0.710650	0.313893	0.972794
area error	0.690299	0.223970	0.951830
compactness error	0.490424	0.421659	0.356065
concavity error	0.439167	0.342627	0.332358
concave points error	0.615634	0.393298	0.513346
worst radius	0.830318	0.185728	0.715065
worst texture	0.292752	0.090651	0.194799
worst perimeter	0.855923	0.219169	0.719684
worst area	0.809630	0.177193	0.751548
worst smoothness	0.452753	0.426675	0.141919
worst compactness	0.667454	0.473200	0.287103
worst concavity	0.752399	0.433721	0.380585
worst concave points	0.910155	0.430297	0.531062
worst symmetry	0.375744	0.699826	0.094543
worst fractal dimension	0.368661	0.438413	0.049559
target	-0.776614	-0.330499	-0.567134

	…	worst texture	worst perimeter	worst area	\
mean radius	…	0.297008	0.965137	0.941082	
mean texture	…	0.912045	0.358040	0.343546	
mean perimeter	…	0.303038	0.970387	0.941550	
mean area	…	0.287489	0.959120	0.959213	
mean smoothness	…	0.036072	0.238853	0.206718	
mean compactness	…	0.248133	0.590210	0.509604	
mean concavity	…	0.299879	0.729565	0.675987	
mean concave points	…	0.292752	0.855923	0.809630	
mean symmetry	…	0.090651	0.219169	0.177193	
radius error	…	0.194799	0.719684	0.751548	
perimeter error	…	0.200371	0.721031	0.730713	
area error	…	0.196497	0.761213	0.811408	
compactness error	…	0.143003	0.260516	0.199371	
concavity error	…	0.100241	0.226680	0.188353	
concave points error	…	0.086741	0.394999	0.342271	
worst radius	…	0.359921	0.993708	0.984015	
worst texture	…	1.000000	0.365098	0.345842	
worst perimeter	…	0.365098	1.000000	0.977578	
worst area	…	0.345842	0.977578	1.000000	
worst smoothness	…	0.225429	0.236775	0.209145	
worst compactness	…	0.360832	0.529408	0.438296	
worst concavity	…	0.368366	0.618344	0.543331	
worst concave points	…	0.359755	0.816322	0.747419	
worst symmetry	…	0.233027	0.269493	0.209146	
worst fractal dimension	…	0.219122	0.138957	0.079647	
target	…	-0.456903	-0.782914	-0.733825	

	worst smoothness	worst compactness	worst concavity	\

mean radius	0.119616	0.413463	0.526911
mean texture	0.077503	0.277830	0.301025
mean perimeter	0.150549	0.455774	0.563879
mean area	0.123523	0.390410	0.512606
mean smoothness	0.805324	0.472468	0.434926
mean compactness	0.565541	0.865809	0.816275
mean concavity	0.448822	0.754968	0.884103
mean concave points	0.452753	0.667454	0.752399
mean symmetry	0.426675	0.473200	0.433721
radius error	0.141919	0.287103	0.380585
perimeter error	0.130054	0.341919	0.418899
area error	0.125389	0.283257	0.385100
compactness error	0.227394	0.678780	0.639147
concavity error	0.168481	0.484858	0.662564
concave points error	0.215351	0.452888	0.549592
worst radius	0.216574	0.475820	0.573975
worst texture	0.225429	0.360832	0.368366
worst perimeter	0.236775	0.529408	0.618344
worst area	0.209145	0.438296	0.543331
worst smoothness	1.000000	0.568187	0.518523
worst compactness	0.568187	1.000000	0.892261
worst concavity	0.518523	0.892261	1.000000
worst concave points	0.547691	0.801080	0.855434
worst symmetry	0.493838	0.614441	0.532520
worst fractal dimension	0.617624	0.810455	0.686511
target	-0.421465	-0.590998	-0.659610

	worst concave points	worst symmetry \
mean radius	0.744214	0.163953
mean texture	0.295316	0.105008
mean perimeter	0.771241	0.189115
mean area	0.722017	0.143570
mean smoothness	0.503053	0.394309
mean compactness	0.815573	0.510223
mean concavity	0.861323	0.409464
mean concave points	0.910155	0.375744
mean symmetry	0.430297	0.699826
radius error	0.531062	0.094543
perimeter error	0.554897	0.109930
area error	0.538166	0.074126
compactness error	0.483208	0.277878
concavity error	0.440472	0.197788
concave points error	0.602450	0.143116
worst radius	0.787424	0.243529
worst texture	0.359755	0.233027
worst perimeter	0.816322	0.269493
worst area	0.747419	0.209146

```
worst smoothness                          0.547691        0.493838
worst compactness                         0.801080        0.614441
worst concavity                           0.855434        0.532520
worst concave points                      1.000000        0.502528
worst symmetry                            0.502528        1.000000
worst fractal dimension                   0.511114        0.537848
target                                   -0.793566       -0.416294

                         worst fractal dimension    target
mean radius                             0.007066 -0.730029
mean texture                            0.119205 -0.415185
mean perimeter                          0.051019 -0.742636
mean area                               0.003738 -0.708984
mean smoothness                         0.499316 -0.358560
mean compactness                        0.687382 -0.596534
mean concavity                          0.514930 -0.696360
mean concave points                     0.368661 -0.776614
mean symmetry                           0.438413 -0.330499
radius error                            0.049559 -0.567134
perimeter error                         0.085433 -0.556141
area error                              0.017539 -0.548236
compactness error                       0.590973 -0.292999
concavity error                         0.439329 -0.253730
concave points error                    0.310655 -0.408042
worst radius                            0.093492 -0.776454
worst texture                           0.219122 -0.456903
worst perimeter                         0.138957 -0.782914
worst area                              0.079647 -0.733825
worst smoothness                        0.617624 -0.421465
worst compactness                       0.810455 -0.590998
worst concavity                         0.686511 -0.659610
worst concave points                    0.511114 -0.793566
worst symmetry                          0.537848 -0.416294
worst fractal dimension                 1.000000 -0.323872
target                                 -0.323872  1.000000

[26 rows x 26 columns]
```

[ ]:

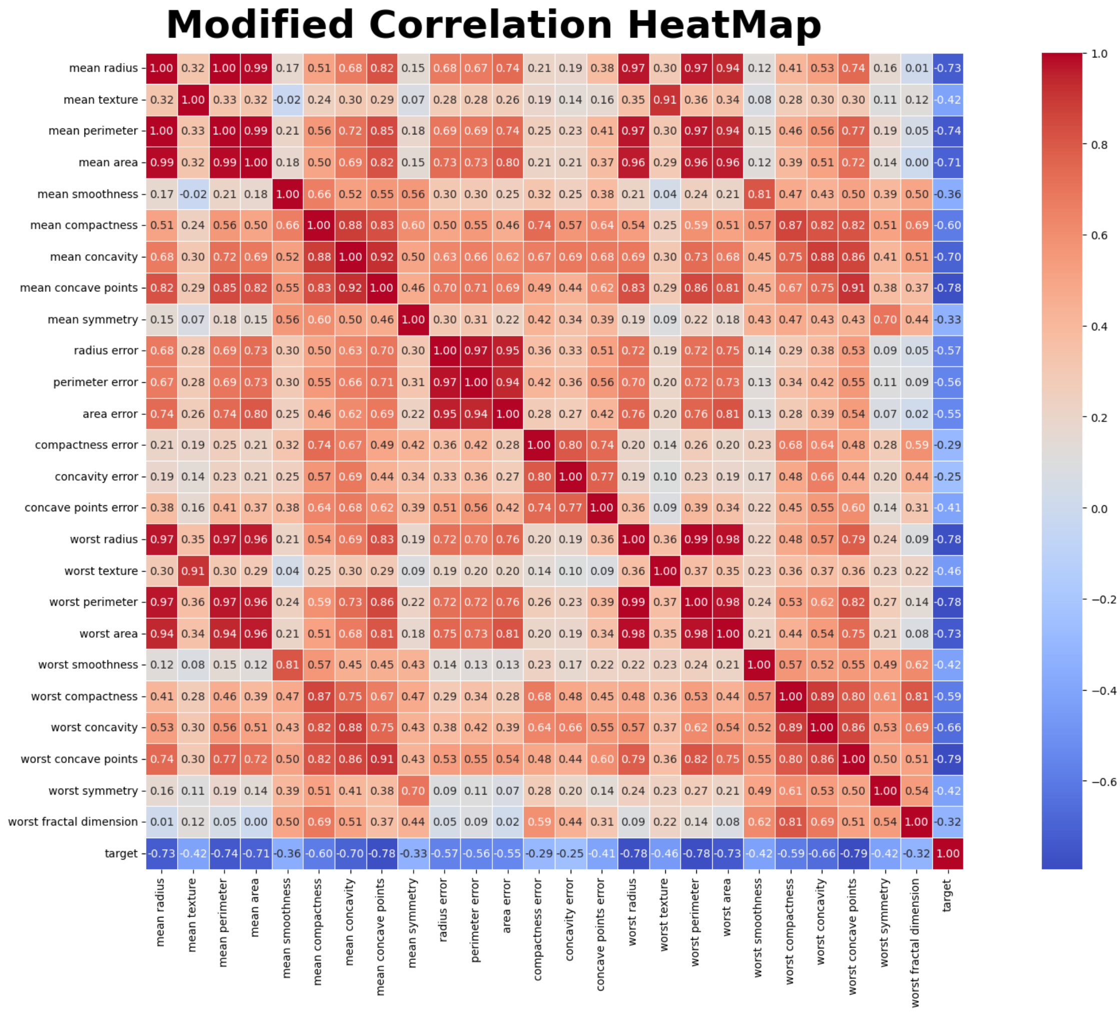
5 Analyzing the Modified Correlation through a HeatMap

A correlation is responsible for showing the type of relationship attributes have with each other (can be one of either two). It also shows relationship between attributes (features) and the column that needs to be predicted (target)

In other words, correlation represents the degree of relationship between variables (features)

This degree can be either: - Positive Degree (when both attributes have a positive correlation - this means that they are directly proportional) - Negative Degree (when either attribute (or both) have a negative correlation - this means that they are inversely proportional)

[182]:
```
fig, plot = plt.subplots(figsize = (20, 13))

fig.suptitle("Modified Correlation HeatMap", fontsize = 35, fontweight = "bold")

sns.heatmap(
    correlation,
    annot = True,
    fmt = ".2f",
    square = True,
    cbar = True,
    linewidths = 0.5,
    cmap = "coolwarm"
)

fig.savefig("Correlation HeatMap.png")

plt.tight_layout()
plt.show()
```



Strongest Features having influence on Target

- worst concave points
- worst perimeter
- worst radius
- worst area
- mean concave points
- mean perimeter
- mean radius

```
[ ]:
```

5.0.1 Checking the cardinality of each feature

[68]:
```
for att in dataFrame:
    print(dataFrame[att].value_counts())
    print("\n")
```

```
mean radius
12.340    4
11.060    3
10.260    3
12.770    3
13.050    3
         ..
19.810    1
13.540    1
13.080    1
9.504     1
15.340    1
Name: count, Length: 456, dtype: int64


mean texture
16.84    3
19.83    3
15.70    3
20.52    3
18.22    3
        ..
27.88    1
22.68    1
23.93    1
29.37    1
30.62    1
Name: count, Length: 479, dtype: int64


mean perimeter
134.70    3
87.76     3
82.61     3
70.79     2
113.40    2
         ..
76.20     1
108.30    1
71.79     1
68.89     1
120.90    1
```

```
Name: count, Length: 522, dtype: int64


mean area
512.2     3
477.3     2
321.6     2
514.3     2
361.6     2
         ..
1261.0    1
396.6     1
1265.0    1
1102.0    1
572.3     1
Name: count, Length: 539, dtype: int64


mean smoothness
0.10070    5
0.11500    4
0.10540    4
0.10750    4
0.11410    3
          ..
0.09469    1
0.09428    1
0.09688    1
0.05263    1
0.07956    1
Name: count, Length: 474, dtype: int64


mean compactness
0.11470    3
0.12060    3
0.15990    2
0.11170    2
0.12830    2
          ..
0.10340    1
0.03872    1
0.27700    1
0.05884    1
0.04052    1
Name: count, Length: 537, dtype: int64
```

```
mean concavity
0.000000    13
0.120400     3
0.197400     2
0.244800     2
0.111500     2
            ..
0.243900     1
0.056990     1
0.092510     1
0.059290     1
0.001487     1
Name: count, Length: 537, dtype: int64


mean concave points
0.00000    13
0.02864     3
0.10430     2
0.12420     2
0.01615     2
           ..
0.05843     1
0.09791     1
0.01238     1
0.07017     1
0.03711     1
Name: count, Length: 542, dtype: int64


mean symmetry
0.1893    4
0.1601    4
0.1714    4
0.1717    4
0.1769    4
         ..
0.1546    1
0.2054    1
0.2197    1
0.1586    1
0.1709    1
Name: count, Length: 432, dtype: int64


radius error
0.2204    3
0.2860    3
```

```
0.2976    2
0.3380    2
0.2684    2
         ..
0.1302    1
0.4564    1
0.1904    1
0.3857    1
1.1670    1
Name: count, Length: 540, dtype: int64


perimeter error
1.778    4
1.667    2
1.445    2
3.767    2
1.491    2
        ..
5.203    1
8.867    1
5.772    1
1.750    1
4.021    1
Name: count, Length: 533, dtype: int64


area error
16.97    3
16.64    3
17.67    3
18.54    3
20.67    2
        ..
32.55    1
44.74    1
30.66    1
15.34    1
17.25    1
Name: count, Length: 528, dtype: int64


compactness error
0.01104    3
0.01812    3
0.02310    3
0.01382    2
0.02772    2
```

```
          ..
0.02423    1
0.04960    1
0.06158    1
0.01067    1
0.01169    1
Name: count, Length: 541, dtype: int64


concavity error
0.000000    13
0.016980     2
0.018650     2
0.016520     2
0.020000     2
            ..
0.051980     1
0.016220     1
0.047300     1
0.012670     1
0.001487     1
Name: count, Length: 533, dtype: int64


concave points error
0.000000    13
0.014990     3
0.011100     3
0.011670     3
0.010110     2
            ..
0.022520     1
0.003608     1
0.023970     1
0.008849     1
0.015610     1
Name: count, Length: 507, dtype: int64


worst radius
12.36    5
13.50    4
13.34    4
13.45    3
15.05    3
        ..
32.49    1
13.61    1
```

```
21.58    1
13.03    1
17.91    1
Name: count, Length: 457, dtype: int64


worst texture
17.70    3
27.26    3
25.09    2
25.59    2
29.41    2
        ..
36.71    1
16.18    1
28.12    1
23.75    1
20.88    1
Name: count, Length: 511, dtype: int64


worst perimeter
117.70    3
105.90    3
101.70    3
158.80    2
119.40    2
         ..
123.40    1
136.80    1
77.80     1
88.10     1
86.12     1
Name: count, Length: 514, dtype: int64


worst area
458.0     2
472.4     2
706.0     2
708.8     2
1210.0    2
         ..
670.0     1
1124.0    1
1724.0    1
268.6     1
533.7     1
```

```
Name: count, Length: 544, dtype: int64


worst smoothness
0.1401    4
0.1312    4
0.1256    4
0.1415    4
0.1216    4
         ..
0.1396    1
0.1380    1
0.1768    1
0.1525    1
0.1354    1
Name: count, Length: 411, dtype: int64


worst compactness
0.1486    3
0.3416    3
0.1049    2
0.3735    2
0.2920    2
         ..
0.1922    1
0.1507    1
0.8681    1
0.4725    1
0.0937    1
Name: count, Length: 529, dtype: int64


worst concavity
0.000000    13
0.450400     3
0.137700     3
0.181100     2
0.396500     2
            ..
0.410700     1
0.071530     1
0.340300     1
0.004955     1
0.938700     1
Name: count, Length: 539, dtype: int64
```

```
worst concave points
0.00000    13
0.04306     3
0.18270     3
0.05556     3
0.11050     3
           ..
0.08568     1
0.25500     1
0.19840     1
0.18600     1
0.16590     1
Name: count, Length: 492, dtype: int64


worst symmetry
0.2369    3
0.3109    3
0.2383    3
0.2226    3
0.3196    3
         ..
0.2790    1
0.2329    1
0.2722    1
0.2473    1
0.2249    1
Name: count, Length: 500, dtype: int64


worst fractal dimension
0.07427    3
0.06386    2
0.10190    2
0.08950    2
0.12970    2
          ..
0.06637    1
0.06033    1
0.12400    1
0.06484    1
0.05737    1
Name: count, Length: 535, dtype: int64


target
1    357
0    212
```

```
Name: count, dtype: int64
```

[ ]:

6 Splitting the Data into X & Y

The features and the target variable are split into separate frames so as to prepare the features specifically for the data preprocessing phase

[69]:
```
x, y = dataFrame.drop("target", axis = 1), dataFrame["target"]

y = pd.DataFrame(y, columns = ["target"])

x.head(), y.head()
```

[69]:
```
(   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
 0        17.99         10.38          122.80     1001.0          0.11840
 1        20.57         17.77          132.90     1326.0          0.08474
 2        19.69         21.25          130.00     1203.0          0.10960
 3        11.42         20.38           77.58      386.1          0.14250
 4        20.29         14.34          135.10     1297.0          0.10030

    mean compactness  mean concavity  mean concave points  mean symmetry  \
 0           0.27760          0.3001              0.14710         0.2419
 1           0.07864          0.0869              0.07017         0.1812
 2           0.15990          0.1974              0.12790         0.2069
 3           0.28390          0.2414              0.10520         0.2597
 4           0.13280          0.1980              0.10430         0.1809

    radius error  …  worst radius  worst texture  worst perimeter  \
 0        1.0950  …         25.38          17.33           184.60
 1        0.5435  …         24.99          23.41           158.80
 2        0.7456  …         23.57          25.53           152.50
 3        0.4956  …         14.91          26.50            98.87
 4        0.7572  …         22.54          16.67           152.20

    worst area  worst smoothness  worst compactness  worst concavity  \
 0      2019.0            0.1622             0.6656           0.7119
 1      1956.0            0.1238             0.1866           0.2416
 2      1709.0            0.1444             0.4245           0.4504
 3       567.7            0.2098             0.8663           0.6869
 4      1575.0            0.1374             0.2050           0.4000

    worst concave points  worst symmetry  worst fractal dimension
 0                0.2654          0.4601                  0.11890
```

```
1                 0.1860          0.2750                  0.08902
2                 0.2430          0.3613                  0.08758
3                 0.2575          0.6638                  0.17300
4                 0.1625          0.2364                  0.07678

[5 rows x 25 columns],
   target
0       0
1       0
2       0
3       0
4       0)
```

[ ]:

```

```

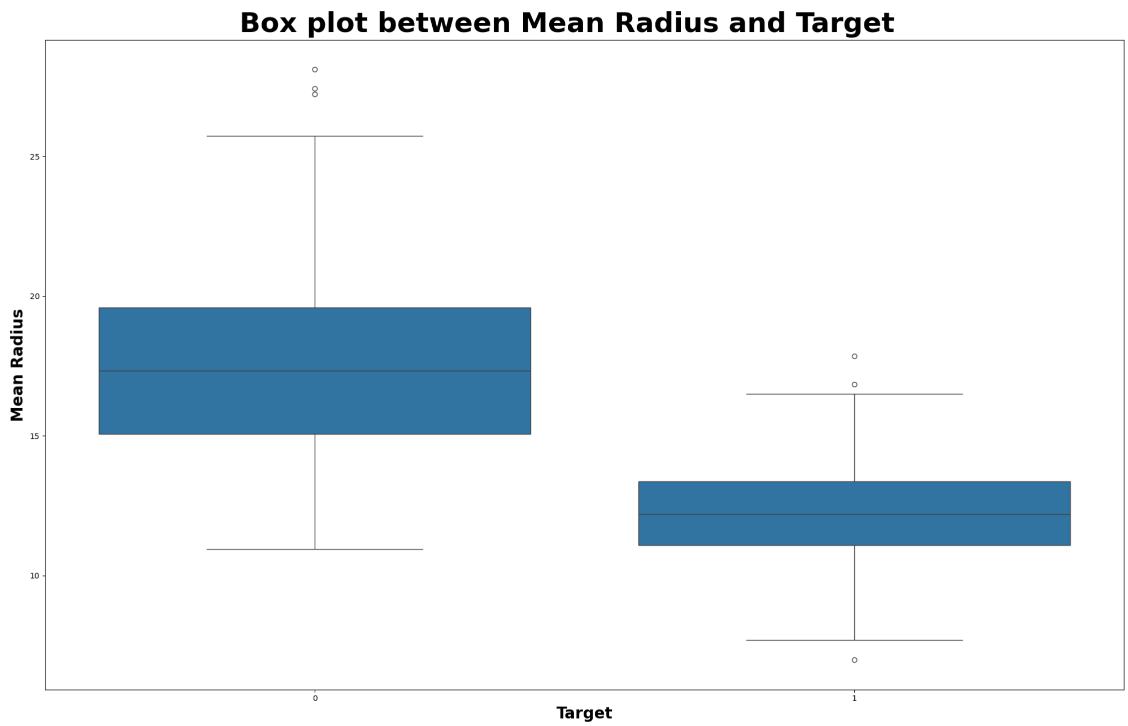
7 Analyzing Effect of a few Features on prediction of Target through BoxPlot

As earlier mentioned, a correlation between two variables explains the relationship between them. In this case, it is important to analyze the relationship of a few features with the target column to determine as to how a feature is going to help some machine learning model in predicting each value for the target column

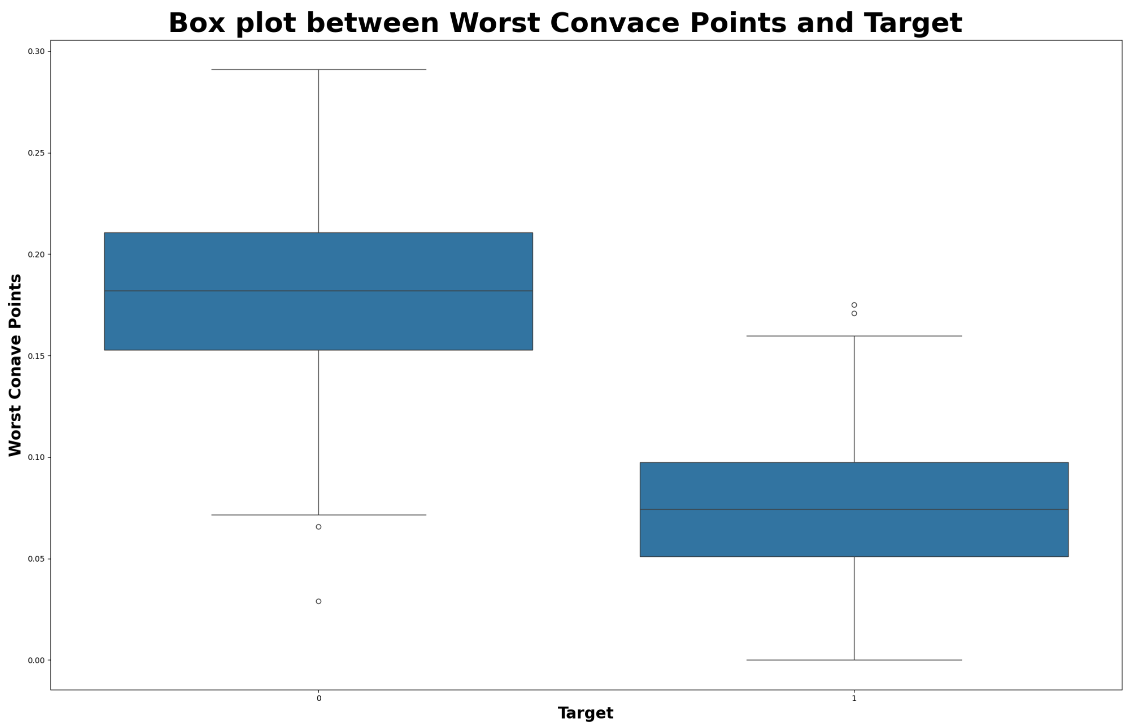
The box-plot below is responsible for showing how one feature helps in prediction one class of the target column. The more the two boxes overlap, the more confusion the model can have in distinguishing classes (it is optimal to have no overlaps). The smaller the size of a box, the more the variability of data for that feature meaning less consistency (smaller box sizes are preferred for lesser variability and more data consistency). The small dots below and above the lower and upper whiskers represent data outliers (they can be any invalid data entry, useless or meaningless information for that feature)

[70]:

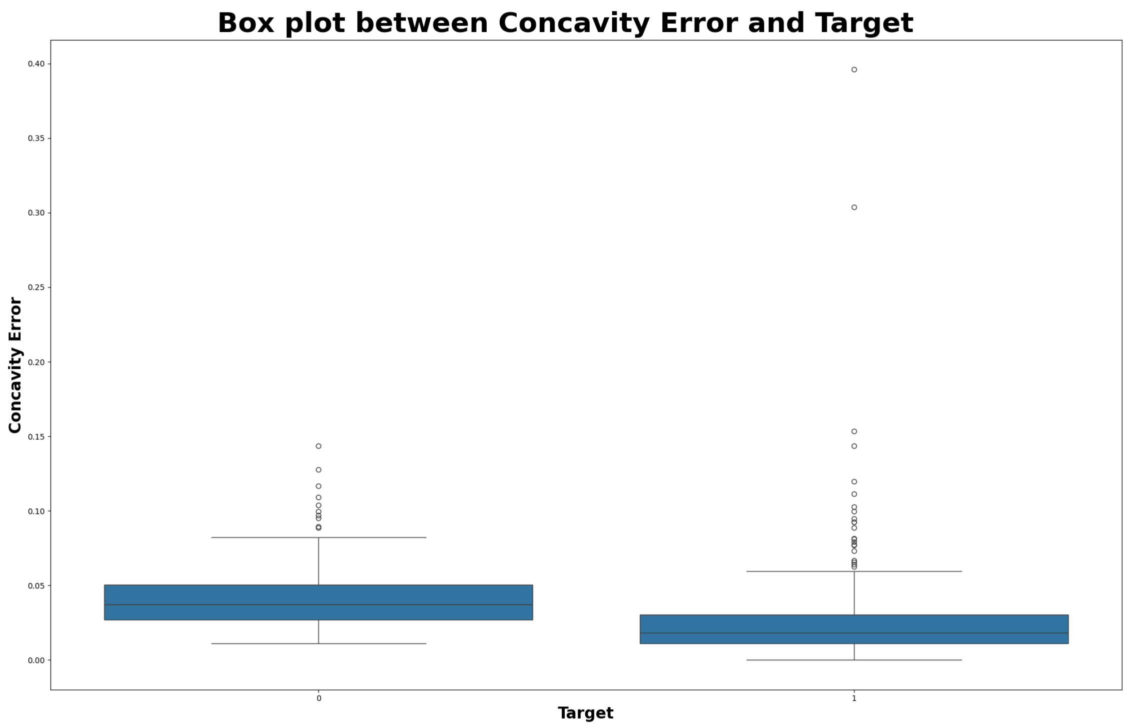
```
fig, plot = plt.subplots(figsize = (20, 13))

sns.boxplot(x = "target", y = "mean radius", data = dataFrame, ax = plot)

fig.suptitle("Box plot between Mean Radius and Target", fontsize = 35,␣
 ↪fontweight = "bold")
plot.set_xlabel("Target", fontsize = 20, fontweight = "bold")
plot.set_ylabel("Mean Radius", fontsize = 20, fontweight = "bold")

plt.tight_layout()
plt.show()
```



[71]:
```
fig, plot = plt.subplots(figsize = (20, 13))

sns.boxplot(x = "target", y = "worst concave points", data = dataFrame, ax =␣
 ↪plot)

fig.suptitle("Box plot between Worst Convace Points and Target", fontsize = 35,␣
 ↪fontweight = "bold")
plot.set_xlabel("Target", fontsize = 20, fontweight = "bold")
plot.set_ylabel("Worst Conave Points", fontsize = 20, fontweight = "bold")

plt.tight_layout()
plt.show()
```



[72]:
```
fig, plot = plt.subplots(figsize = (20, 13))

sns.boxplot(x = "target", y = "concavity error", data = dataFrame, ax = plot)

fig.suptitle("Box plot between Concavity Error and Target", fontsize = 35, fontweight = "bold")
plot.set_xlabel("Target", fontsize = 20, fontweight = "bold")
plot.set_ylabel("Concavity Error", fontsize = 20, fontweight = "bold")

plt.tight_layout()
plt.show()
```



[ ]:

8 Setting up the Data Processing Pipeline Workflow

8.0.1 Using Pipeline to implement imputation, column transformers and ML models

[73]:
```
numericCols = [att for att in x]

numericTransformer = Pipeline(steps = [
    ("impute", SimpleImputer(strategy = "mean"))
])

preprocessor = ColumnTransformer(transformers = [
    ("numeric", numericTransformer, numericCols)
])

rfc = Pipeline(steps = [
    ("preprocessor", preprocessor),
    ("model", RandomForestClassifier())
])

svc = Pipeline(steps = [
    ("preprocessor", preprocessor),
    ("model", SVC())
```

```
])

lr = Pipeline(steps = [
    ("preprocessor", preprocessor),
    ("model", LogisticRegression())
])
```

[ ]:

9 Splitting the data into Training, Validation and Testing Sets

- Training Data = 80%
- Testing Data = 20%

[74]:
```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.20)
```

10 Training each Classification Model on the Training Set

The training set will be used to train each classification model on data (this is the data that will be responsible for the model to learn and recognize patterns at the time of testing)

[75]:
```
rfc.fit(xtrain, ytrain)
```

[75]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('numeric',
                                                  Pipeline(steps=[('impute',
SimpleImputer())]),
                                                  ['mean radius',
                                                   'mean texture',
                                                   'mean perimeter',
                                                   'mean area',
                                                   'mean smoothness',
                                                   'mean compactness',
                                                   'mean concavity',
                                                   'mean concave points',
                                                   'mean symmetry',
                                                   'radius error',
                                                   'perimeter error',
                                                   'area error',
                                                   'compactness error',
                                                   'concavity error',
                                                   'concave points error',
                                                   'worst radius',
                                                   'worst texture',
                                                   'worst perimeter',
                                                   'worst area',
                                                   'worst smoothness',
```

```
                                                          'worst compactness',
                                                          'worst concavity',
                                                          'worst concave points',
                                                          'worst symmetry',
                                                          'worst fractal '
                                                          'dimension'])])),
                ('model', RandomForestClassifier())])
```

[76]:
```
svc.fit(xtrain, ytrain)
```

[76]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('numeric',
                                                  Pipeline(steps=[('impute',
SimpleImputer())]),
                                                  ['mean radius',
                                                   'mean texture',
                                                   'mean perimeter',
                                                   'mean area',
                                                   'mean smoothness',
                                                   'mean compactness',
                                                   'mean concavity',
                                                   'mean concave points',
                                                   'mean symmetry',
                                                   'radius error',
                                                   'perimeter error',
                                                   'area error',
                                                   'compactness error',
                                                   'concavity error',
                                                   'concave points error',
                                                   'worst radius',
                                                   'worst texture',
                                                   'worst perimeter',
                                                   'worst area',
                                                   'worst smoothness',
                                                   'worst compactness',
                                                   'worst concavity',
                                                   'worst concave points',
                                                   'worst symmetry',
                                                   'worst fractal '
                                                   'dimension'])])),
                ('model', SVC())])
```

[77]:
```
lr.fit(xtrain, ytrain)
```

[77]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('numeric',
                                                  Pipeline(steps=[('impute',
```

```
 SimpleImputer())]),
                                                                  ['mean radius',
                                                                   'mean texture',
                                                                   'mean perimeter',
                                                                   'mean area',
                                                                   'mean smoothness',
                                                                   'mean compactness',
                                                                   'mean concavity',
                                                                   'mean concave points',
                                                                   'mean symmetry',
                                                                   'radius error',
                                                                   'perimeter error',
                                                                   'area error',
                                                                   'compactness error',
                                                                   'concavity error',
                                                                   'concave points error',
                                                                   'worst radius',
                                                                   'worst texture',
                                                                   'worst perimeter',
                                                                   'worst area',
                                                                   'worst smoothness',
                                                                   'worst compactness',
                                                                   'worst concavity',
                                                                   'worst concave points',
                                                                   'worst symmetry',
                                                                   'worst fractal '
                                                                   'dimension'])])),
                 ('model', LogisticRegression())])
```

[ ]:

```
```

11 Creating a single Function for Calculating Classification Performance Metrics for each Model

After all 3 models have been trained on the training dataset, it is now time to actually test their performance and metrics by making predictions on unseen data (this is called the testing portion of the dataset split).

For better code efficiency and consistency, a single function has been defined so as to calculate the classification performance metrics for any respective classification machine learning model easily, just by passing a few specified parameters

[100]:

```
model_dict = {
    "Random Forest Classifier" : rfc,
    "Support Vector Classifier" : svc,
    "Logistic Regression" : lr
}
```

```python
def calculatePerformanceMetrics(models, ytest, xtest, printResults : bool) ->␣
↪[pd.DataFrame]:

    results0, results1 = [], []

    for model in models:

        ypredicted = models[model].predict(xtest)
        prec0, prec1 = precision_score(ytest, ypredicted, pos_label = 0),␣
↪precision_score(ytest, ypredicted, pos_label = 1)
        rec0, rec1 = recall_score(ytest, ypredicted, pos_label = 0),␣
↪recall_score(ytest, ypredicted, pos_label = 1)
        f1_0, f1_1 = f1_score(ytest, ypredicted, pos_label = 0),␣
↪f1_score(ytest, ypredicted, pos_label = 1)
        mean_acc = accuracy_score(ytest, ypredicted)

        results0.append([prec0, rec0, f1_0])
        results1.append([prec1, rec1, f1_1])

        if printResults:
            print(f"\nCalculating Performance Metrics for {model}:")
            print(f"\nPerformance for Class = 0")
            print(f"Precision: {prec0}\nRecall: {rec0}\nF1 Score: {f1_0}")
            print(f"\nPerformance for Class = 1")
            print(f"Precision: {prec1}\nRecall: {rec1}\nF1 Score: {f1_1}")
            print(f"\nMean Accuracy: {mean_acc}")

    resultsFrame0 = pd.DataFrame(
        [res for res in results0],
        columns = ["Precision", "Recall", "F1 Score"],
        index = list(models.keys())
    )

    resultsFrame1 = pd.DataFrame(
        [res for res in results1],
        columns = ["Precision", "Recall", "F1 Score"],
        index = list(models.keys())
    )

    return [resultsFrame0, resultsFrame1]
```

[101]:
```python
res = calculatePerformanceMetrics(model_dict, ytest, xtest, printResults = True)
```

```
Calculating Performance Metrics for Random Forest Classifier:
```

```
Performance for Class = 0
Precision: 0.9130434782608695
Recall: 0.9333333333333333
F1 Score: 0.9230769230769231

Performance for Class = 1
Precision: 0.9558823529411765
Recall: 0.9420289855072463
F1 Score: 0.948905109489051

Mean Accuracy: 0.9385964912280702

Calculating Performance Metrics for Support Vector Classifier:

Performance for Class = 0
Precision: 0.9047619047619048
Recall: 0.8444444444444444
F1 Score: 0.8735632183908046

Performance for Class = 1
Precision: 0.9027777777777778
Recall: 0.9420289855072463
F1 Score: 0.9219858156028369

Mean Accuracy: 0.9035087719298246

Calculating Performance Metrics for Logistic Regression:

Performance for Class = 0
Precision: 0.9111111111111111
Recall: 0.9111111111111111
F1 Score: 0.9111111111111111

Performance for Class = 1
Precision: 0.9420289855072463
Recall: 0.9420289855072463
F1 Score: 0.9420289855072463

Mean Accuracy: 0.9298245614035088
```

[ ]:

```

```

12 Visualizing the Classification Performance Metrics for each Model w.r.t each Class using BarPlot

It is always better to understand and interpret results and calculations visually using plots. A simple bar plot showing every models performance for each predictive class (0 or 1 - Breast Cancer

Yes/No) is constructed. The 3 most important classification metrics for each model w.r.t each predictive class are also represented in the below bar plot.

[104]:
```
res = calculatePerformanceMetrics(model_dict, ytest, xtest, printResults =␣
↪False)
classificationReportFrame0, classificationReportFrame1 = res[0], res[1]
```

[105]:
```
classificationReportFrame0
```

[105]:

	Precision	Recall	F1 Score
Random Forest Classifier	0.913043	0.933333	0.923077
Support Vector Classifier	0.904762	0.844444	0.873563
Logistic Regression	0.911111	0.911111	0.911111

[106]:
```
classificationReportFrame1
```

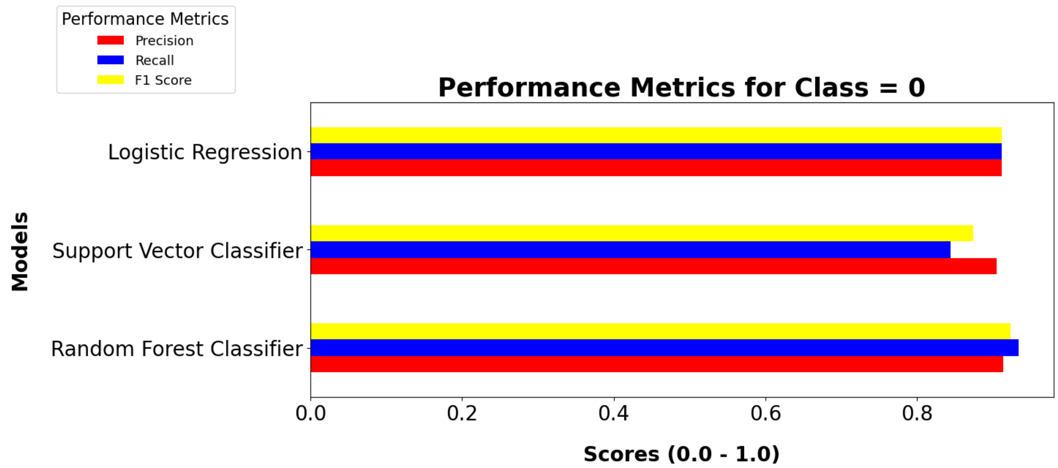
[106]:

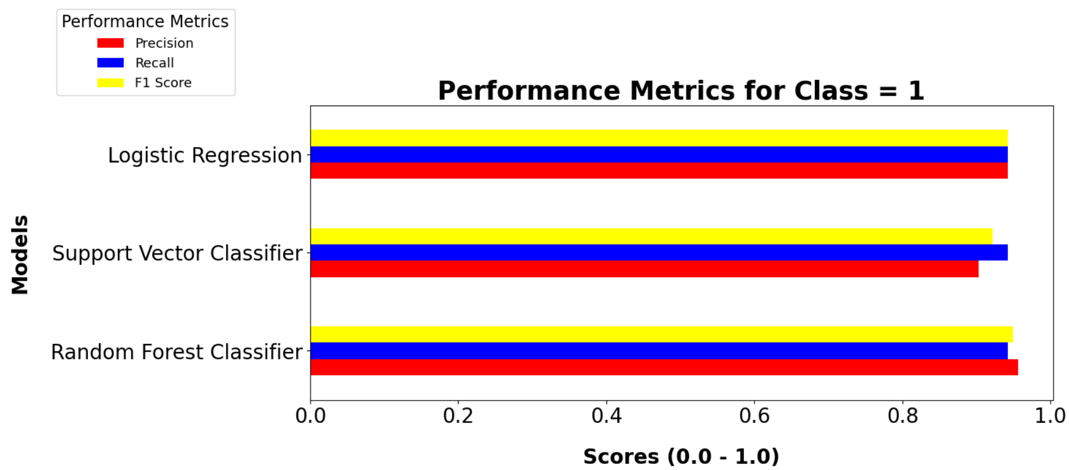
	Precision	Recall	F1 Score
Random Forest Classifier	0.955882	0.942029	0.948905
Support Vector Classifier	0.902778	0.942029	0.921986
Logistic Regression	0.942029	0.942029	0.942029

[181]:
```
fig, (plot1, plot2) = plt.subplots(2, 1, figsize = (15, 15))

classificationReportFrame0.plot(kind = "barh", ax = plot1, color = ["Red",␣
↪"Blue", "Yellow"])

fig.suptitle("Comparison of Performace Metrics for RFC, SVC and LR", fontsize =␣
↪35, fontweight = "bold", y = 1.02)

plot1.set_title("Performance Metrics for Class = 0", fontsize = 25, fontweight␣
↪= "bold")
plot1.set_ylabel("Models", fontsize = 20, fontweight = "bold")
plot1.set_xlabel("Scores (0.0 - 1.0)", fontsize = 20, fontweight = "bold")

plot1.xaxis.labelpad = 20
plot1.yaxis.labelpad = 20

plot1.tick_params(axis = "both", labelsize = 20)

plot1.legend(
    title = "Performance Metrics",
    fontsize = 13,
    title_fontsize = 16,
    loc = "upper left",
    bbox_to_anchor = (-0.35, 1.35)
)
```

```python
classificationReportFrame1.plot(kind = "barh", ax = plot2, color = ["Red",␣
↪"Blue", "Yellow"])

plot2.set_title("Performance Metrics for Class = 1", fontsize = 25, fontweight␣
↪= "bold")
plot2.set_ylabel("Models", fontsize = 20, fontweight = "bold")
plot2.set_xlabel("Scores (0.0 - 1.0)", fontsize = 20, fontweight = "bold")

plot2.xaxis.labelpad = 20
plot2.yaxis.labelpad = 20

plot2.tick_params(axis = "both", labelsize = 20)

plot2.legend(
    title = "Performance Metrics",
    fontsize = 13,
    title_fontsize = 16,
    loc = "upper left",
    bbox_to_anchor = (-0.35, 1.35)
)

fig.savefig("Performance Metrics BarPlot between RFC, SVC and LR.png")

plt.tight_layout(pad = 2.0)
plt.show()
```

Comparison of Performace Metrics for RFC, SVC and LR





[ ]:

```

```

13 Cross Validating the General Accuracy for each Model

Cross Validation is very important to understand in the context of interpreting a machine learning models prediction efficiency, stability and reliability over various different parts of the same dataset. We know that the model is only tested on some specified portion of the dataset split (in our case, 20% for testing data and the rest of the 80% for training data). However, sometimes, more critical and crucial data/learning patterns for the model might exist in some other portion of the complete dataset. To test the model's efficiency, consistency and reliability across the complete dataset, we use Cross Validation

In simple words, it validates the Mean Accuracy Score of a machine learning model by training and then testing it on different "folds" of the original dataset

For example, the "cv" parameters is used to determine the number of "folds" to make of the original complete dataset. Since our partition is described as 80% for training and 20% for testing, it will take 4 folds for training and then test the model on the remaining fold. Then, it will take 4 new folds and then test the model on some other fold (that might have been used as a training fold in some previous iteration)

In this way, all the folds are utilized as training and testing, one by one, based on the split and the value passed to the cv parameter in the cross validation function. This helps in better understanding the general prediction performance of a model

[176]:
```
cross_val_score(rfc, x, y, cv = 5, scoring = "accuracy")
```

[176]:
```
array([0.9122807 , 0.94736842, 0.99122807, 0.96491228, 0.97345133])
```

[85]:
```
cross_val_score(svc, x, y, cv = 5, scoring = "accuracy")
```

[85]:
```
array([0.85087719, 0.89473684, 0.92982456, 0.93859649, 0.9380531 ])
```

[86]:
```
cross_val_score(lr, x, y, cv = 5, scoring = "accuracy")
```

[86]:
```
array([0.93859649, 0.93859649, 0.96491228, 0.95614035, 0.96460177])
```

[ ]:
```

```

14 Confusion Matrix Visualization for each Classification Model using HeatMap

A confusion matrix is used for understanding precision and recall better. It can be interpreted as a point of view of the model itself and as to how it is distinguishing classes for the predictive category.

A confusion matrix is used mostly for binary classification problems (like our current one, in which we have to predict one of only two class choices). The matrix visualizes how many predictions the model is making correctly and incorrectly.

The word "confusion" means that a confusion matrix can help us understand exactly where the model is underperforming, predicting incorrect classes or being unable to efficiently distinguish binary classes from each other

The matrix represents a total of four cases among the 2x2 grid: - True Negative (0, 0): The actual class was 0 and the model correctly predicted it as a 0 - False Positive (0, 1): The actual class was 0 but the model incorrectly predicted it as a 1 (it is also called a false alarm) - False Negative (1, 0): The actual class was 1 but the model incorrectly predicted it as a 0 (wrong prediction) - True Positive (1, 1): The actual class was 1 and the model correctly predicted it as a 1

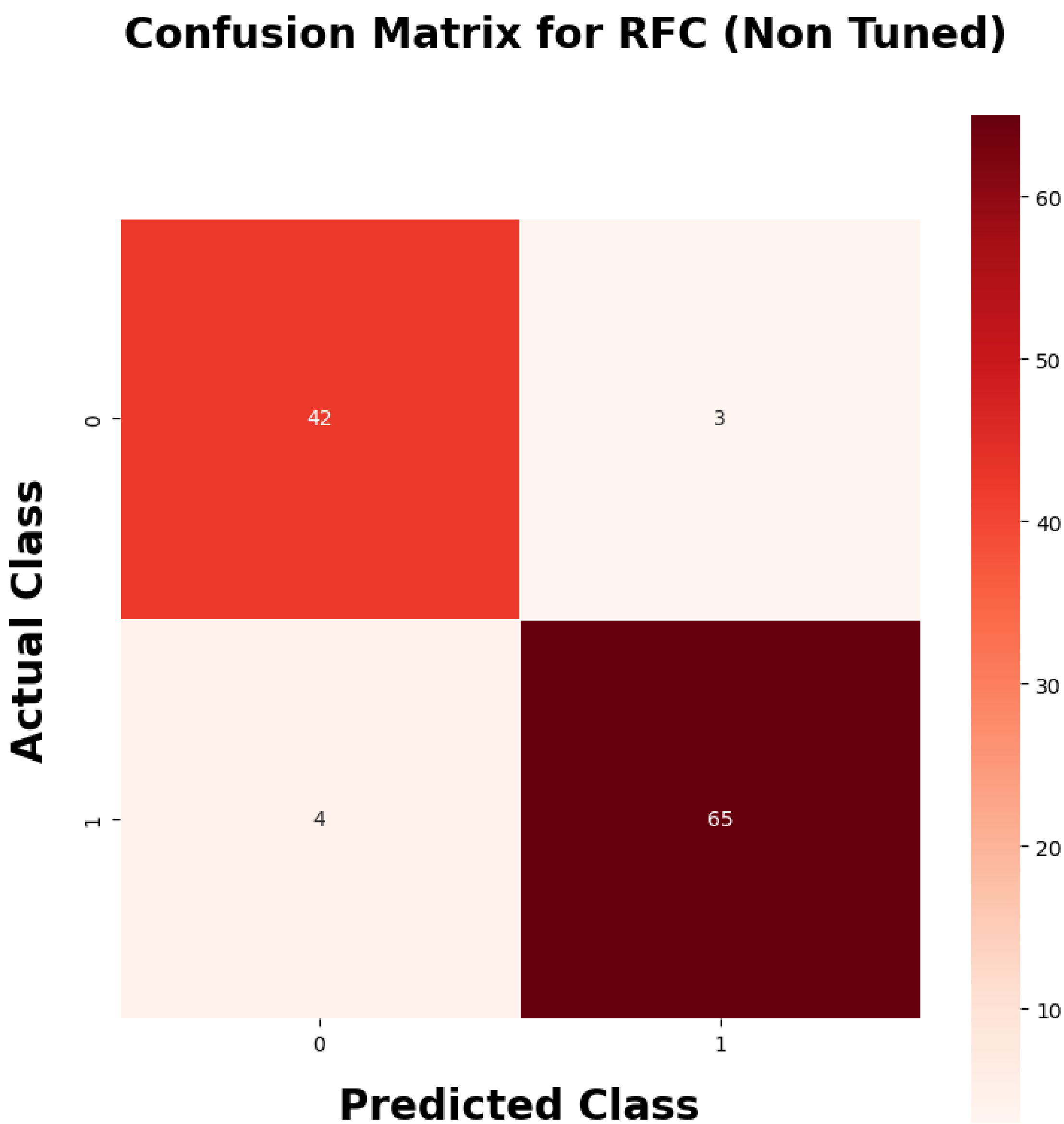
Each value inside each cell of the matrix represents the number of predictions for each case (out of all the 4 confusion cases)

After analyzing all this, it can be concluded that it is best optimal for a model to have majority of the prediction cases in the leading principle diagonal of the confusion matrix (so they fall in either True Negative or True Positive cases). Having more values in the other diagonal are not generally preferred, since they point to model predictive imbalance, wrong predictions or can even lead to class imabalance

[87]:
```
confusion_matrix(ytest, rfc.predict(xtest))
```

[87]:
```
array([[42,  3],
       [ 4, 65]])
```

[88]:
```
fig, plot = plt.subplots(figsize = (8, 8))

sns.heatmap(
    confusion_matrix(ytest, rfc.predict(xtest)),
    annot = True,
    linewidths = 0.5,
    cmap = "Reds",
    cbar = True,
    square = True,
    ax = plot
)

fig.suptitle("Confusion Matrix for RFC (Non Tuned)", fontsize = 20, fontweight␣
 ↪= "bold")

plot.set_xlabel("Predicted Class", fontsize = 20, fontweight = "bold")
plot.set_ylabel("Actual Class", fontsize = 20, fontweight = "bold")

plot.xaxis.labelpad = 15
plot.yaxis.labelpad = 15

plt.tight_layout(pad = 2.0)
plt.show()
```

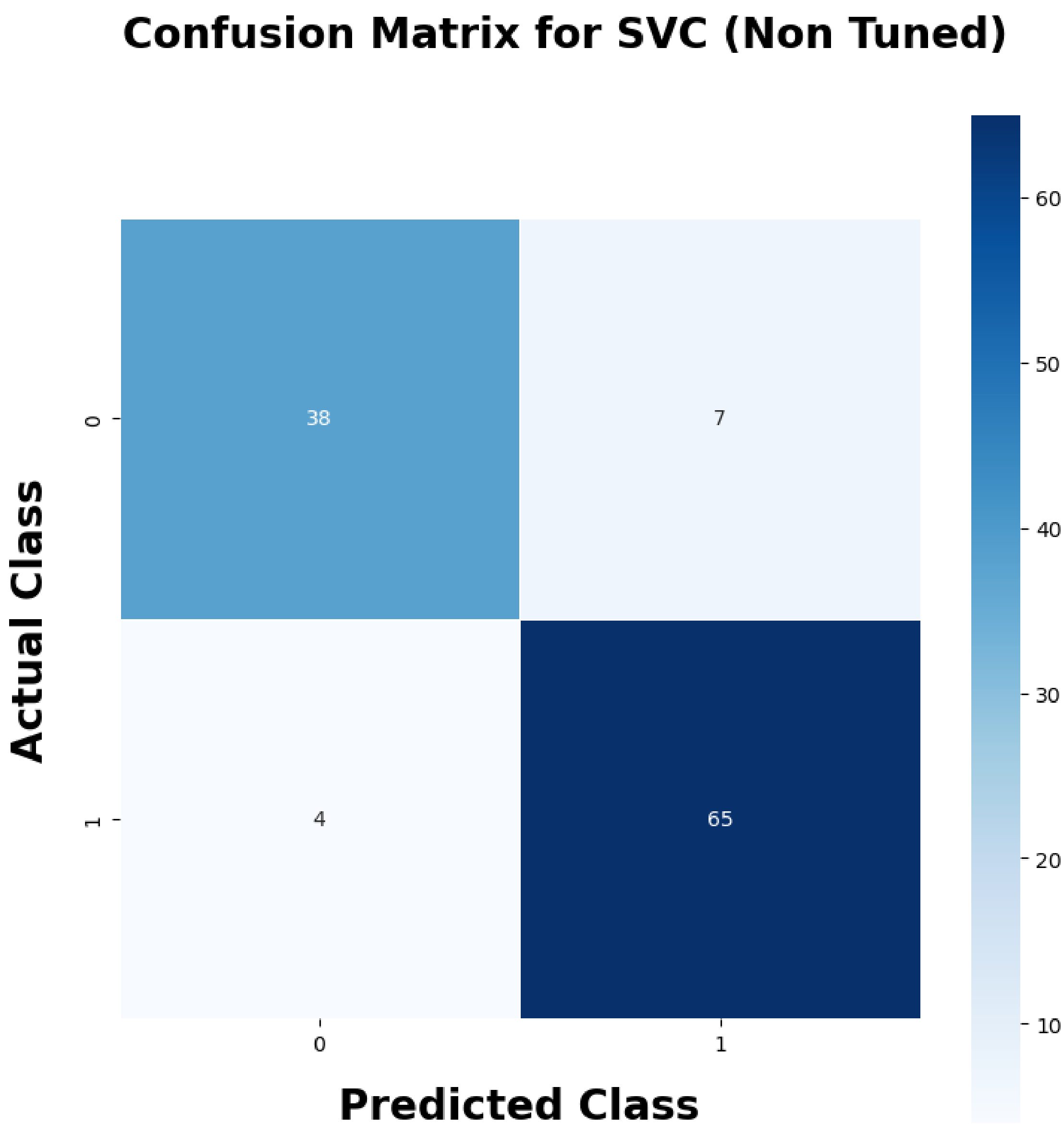


[89]:
```
confusion_matrix(ytest, svc.predict(xtest))
```

[89]:
```
array([[38,  7],
       [ 4, 65]])
```

[90]:
```
fig, plot = plt.subplots(figsize = (8, 8))

sns.heatmap(
    confusion_matrix(ytest, svc.predict(xtest)),
    annot = True,
    linewidths = 0.5,
```

```
    cmap = "Blues",
    cbar = True,
    square = True,
    ax = plot
)

fig.suptitle("Confusion Matrix for SVC (Non Tuned)", fontsize = 20, fontweight␣
 ↪= "bold")

plot.set_xlabel("Predicted Class", fontsize = 20, fontweight = "bold")
plot.set_ylabel("Actual Class", fontsize = 20, fontweight = "bold")

plot.xaxis.labelpad = 15
plot.yaxis.labelpad = 15

plt.tight_layout(pad = 2.0)
plt.show()
```

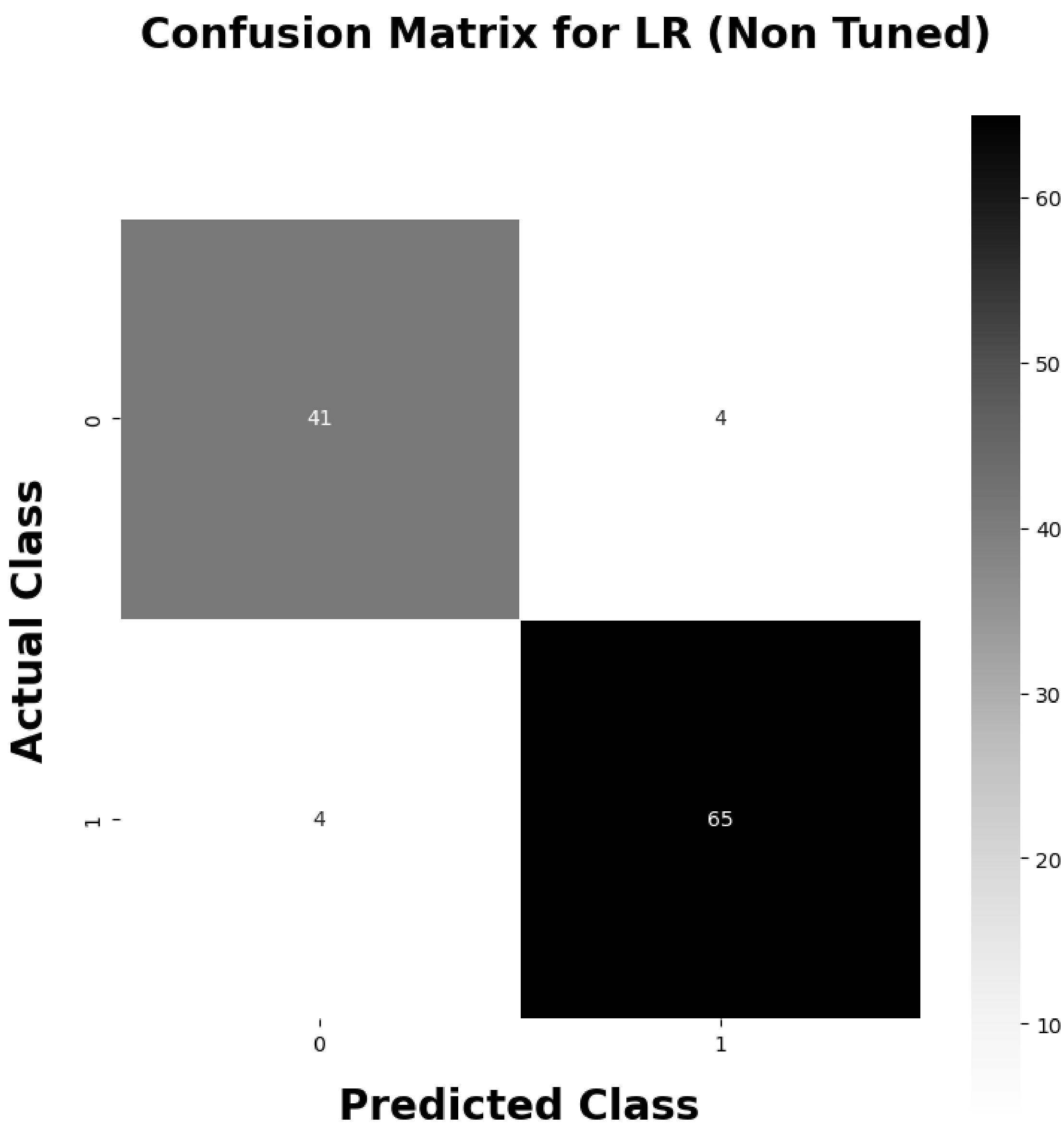


[91]:
```
confusion_matrix(ytest, lr.predict(xtest))
```

[91]:
```
array([[41,  4],
       [ 4, 65]])
```

[92]:
```
fig, plot = plt.subplots(figsize = (8, 8))

sns.heatmap(
    confusion_matrix(ytest, lr.predict(xtest)),
    annot = True,
    linewidths = 0.5,
```

```
    cmap = "Greys",
    cbar = True,
    square = True,
    ax = plot
)

fig.suptitle("Confusion Matrix for LR (Non Tuned)", fontsize = 20, fontweight = 
  "bold")

plot.set_xlabel("Predicted Class", fontsize = 20, fontweight = "bold")
plot.set_ylabel("Actual Class", fontsize = 20, fontweight = "bold")

plot.xaxis.labelpad = 15
plot.yaxis.labelpad = 15

plt.tight_layout(pad = 2.0)
plt.show()
```



[ ]:

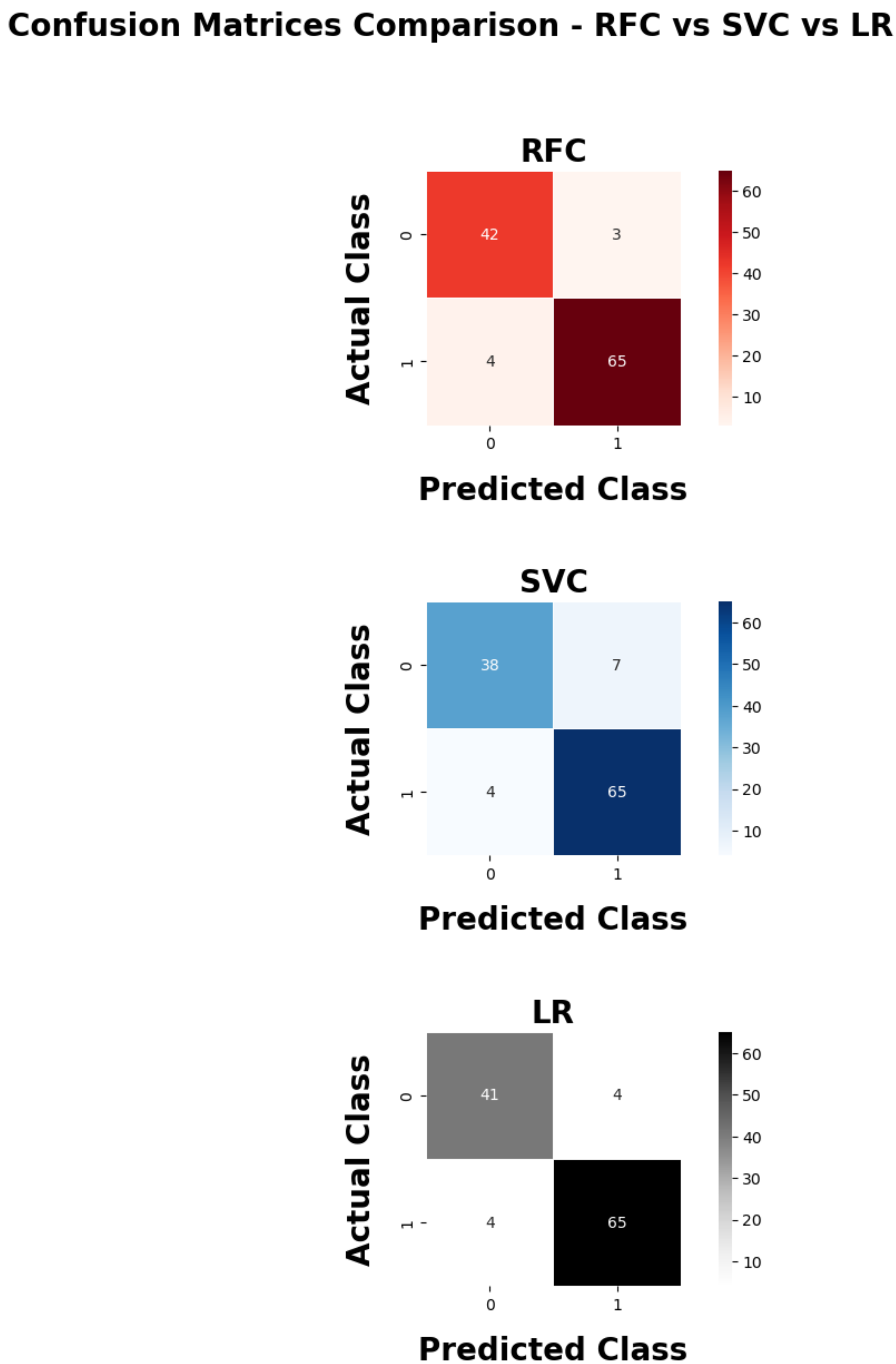
15 Comparing Confusion Matrices of all 3 Models (RFC, SVC, LR) using HeatMap

Comparing all the 3 confusion matrices for Random Forest Classifier, Support Vector Classifier and Logistic Regression

[180]:
```
fig, (plot1, plot2, plot3) = plt.subplots(3, 1, figsize = (8, 13))
```

```python
sns.heatmap(
    confusion_matrix(ytest, rfc.predict(xtest)),
    annot = True,
    linewidths = 0.5,
    cmap = "Reds",
    cbar = True,
    square = True,
    ax = plot1
)

sns.heatmap(
    confusion_matrix(ytest, svc.predict(xtest)),
    annot = True,
    linewidths = 0.5,
    cmap = "Blues",
    cbar = True,
    square = True,
    ax = plot2
)

sns.heatmap(
    confusion_matrix(ytest, lr.predict(xtest)),
    annot = True,
    linewidths = 0.5,
    cmap = "Greys",
    cbar = True,
    square = True,
    ax = plot3
)

fig.suptitle("Confusion Matrices Comparison - RFC vs SVC vs LR", fontsize = 20,␣
 ↪fontweight = "bold")

plot1.set_title("RFC", fontsize = 20, fontweight = "bold")
plot1.set_xlabel("Predicted Class", fontsize = 20, fontweight = "bold")
plot1.set_ylabel("Actual Class", fontsize = 20, fontweight = "bold")
plot1.xaxis.labelpad = 15
plot1.yaxis.labelpad = 15

plot2.set_title("SVC", fontsize = 20, fontweight = "bold")
plot2.set_xlabel("Predicted Class", fontsize = 20, fontweight = "bold")
plot2.set_ylabel("Actual Class", fontsize = 20, fontweight = "bold")
plot2.xaxis.labelpad = 15
plot2.yaxis.labelpad = 15

plot3.set_title("LR", fontsize = 20, fontweight = "bold")
plot3.set_xlabel("Predicted Class", fontsize = 20, fontweight = "bold")
```

```
plot3.set_ylabel("Actual Class", fontsize = 20, fontweight = "bold")
plot3.xaxis.labelpad = 15
plot3.yaxis.labelpad = 15

fig.savefig("Confusion Matrix HeatMap between RFC, SVC and LR.png")

plt.tight_layout(pad = 4.0)
plt.show()
```


Confusion Matrices Comparison - RFC vs SVC vs LR
RFC
42
3
4
65
SVC
38
7
4
65
LR
41
4
4
65
Actual Class
Predicted Class
0
1
10
20
30
40
50
60

[ ]:

```

```

16 Choosing model for tuning = Random Forest Classifier

Selecting Random Forest Classifier as our choice for model tuning

[93]:

```
rfc.get_params()
```

[93]:

```
{'memory': None,
 'steps': [('preprocessor',
   ColumnTransformer(transformers=[('numeric',
                                    Pipeline(steps=[('impute',
SimpleImputer())]),
                                    ['mean radius', 'mean texture',
                                     'mean perimeter', 'mean area',
                                     'mean smoothness', 'mean compactness',
                                     'mean concavity', 'mean concave points',
                                     'mean symmetry', 'radius error',
                                     'perimeter error', 'area error',
                                     'compactness error', 'concavity error',
                                     'concave points error', 'worst radius',
                                     'worst texture', 'worst perimeter',
                                     'worst area', 'worst smoothness',
                                     'worst compactness', 'worst concavity',
                                     'worst concave points', 'worst symmetry',
                                     'worst fractal dimension'])])),
  ('model', RandomForestClassifier())],
 'transform_input': None,
 'verbose': False,
 'preprocessor': ColumnTransformer(transformers=[('numeric',
                                  Pipeline(steps=[('impute', SimpleImputer())]),
                                  ['mean radius', 'mean texture',
                                   'mean perimeter', 'mean area',
                                   'mean smoothness', 'mean compactness',
                                   'mean concavity', 'mean concave points',
                                   'mean symmetry', 'radius error',
                                   'perimeter error', 'area error',
                                   'compactness error', 'concavity error',
                                   'concave points error', 'worst radius',
                                   'worst texture', 'worst perimeter',
                                   'worst area', 'worst smoothness',
                                   'worst compactness', 'worst concavity',
                                   'worst concave points', 'worst symmetry',
                                   'worst fractal dimension'])]),
```

```
'model': RandomForestClassifier(),
'preprocessor__force_int_remainder_cols': True,
'preprocessor__n_jobs': None,
'preprocessor__remainder': 'drop',
'preprocessor__sparse_threshold': 0.3,
'preprocessor__transformer_weights': None,
'preprocessor__transformers': [('numeric',
  Pipeline(steps=[('impute', SimpleImputer())]),
  ['mean radius',
   'mean texture',
   'mean perimeter',
   'mean area',
   'mean smoothness',
   'mean compactness',
   'mean concavity',
   'mean concave points',
   'mean symmetry',
   'radius error',
   'perimeter error',
   'area error',
   'compactness error',
   'concavity error',
   'concave points error',
   'worst radius',
   'worst texture',
   'worst perimeter',
   'worst area',
   'worst smoothness',
   'worst compactness',
   'worst concavity',
   'worst concave points',
   'worst symmetry',
   'worst fractal dimension'])],
'preprocessor__verbose': False,
'preprocessor__verbose_feature_names_out': True,
'preprocessor__numeric': Pipeline(steps=[('impute', SimpleImputer())]),
'preprocessor__numeric__memory': None,
'preprocessor__numeric__steps': [('impute', SimpleImputer())],
'preprocessor__numeric__transform_input': None,
'preprocessor__numeric__verbose': False,
'preprocessor__numeric__impute': SimpleImputer(),
'preprocessor__numeric__impute__add_indicator': False,
'preprocessor__numeric__impute__copy': True,
'preprocessor__numeric__impute__fill_value': None,
'preprocessor__numeric__impute__keep_empty_features': False,
'preprocessor__numeric__impute__missing_values': nan,
'preprocessor__numeric__impute__strategy': 'mean',
```

```
 'model__bootstrap': True,
 'model__ccp_alpha': 0.0,
 'model__class_weight': None,
 'model__criterion': 'gini',
 'model__max_depth': None,
 'model__max_features': 'sqrt',
 'model__max_leaf_nodes': None,
 'model__max_samples': None,
 'model__min_impurity_decrease': 0.0,
 'model__min_samples_leaf': 1,
 'model__min_samples_split': 2,
 'model__min_weight_fraction_leaf': 0.0,
 'model__monotonic_cst': None,
 'model__n_estimators': 100,
 'model__n_jobs': None,
 'model__oob_score': False,
 'model__random_state': None,
 'model__verbose': 0,
 'model__warm_start': False}
```

[ ]:

17 Tuning Hyperparameters of Random Forest Classifier

17.1 Using method of Randomized Search Cross Validation (RSCV)

[94]:
```
rfc_rscv_params = {
    "model__max_features" : ["auto", "sqrt"],
    "model__n_estimators" : [num for num in range(100, 250, 10)],
    "model__min_samples_split" : [num for num in range(2, 6, 1)],
    "model__min_samples_leaf" : [num for num in range(1, 5, 1)],
    "model__max_depth" : [None]
}

rfc_rscv = RandomizedSearchCV(
    estimator = rfc,
    cv = 5,
    param_distributions = rfc_rscv_params,
    verbose = True,
    n_iter = 250
)
```

[95]:
```
rfc_rscv.fit(xtrain, ytrain)
```

```
Fitting 5 folds for each of 250 candidates, totalling 1250 fits
```

```
[95]: RandomizedSearchCV(cv=5,
                       estimator=Pipeline(steps=[('preprocessor',
      ColumnTransformer(transformers=[('numeric',
      Pipeline(steps=[('impute',
                       SimpleImputer())]),
      ['mean '
      'radius',
      'mean '
      'texture',
      'mean '
      'perimeter',
      'mean '
      'area',
      'mean '
      'smoothness',
      'mean '
      'compactness',
      'mean '
      'concavity',
      'mean '
      'concave '
      'points',
      'mean '
      'symmetry',
      'radius '
      'error',
      'perimeter '
      'error',…
      'worst '
      'symmetry',
      'worst '
      'fractal '
      'dimension'])])),
                                                   ('model',
                                                    RandomForestClassifier())]),
                       n_iter=250,
                       param_distributions={'model__max_depth': [None],
                                            'model__max_features': ['auto', 'sqrt'],
                                            'model__min_samples_leaf': [1, 2, 3, 4],
                                            'model__min_samples_split': [2, 3, 4,
                                                                         5],
                                            'model__n_estimators': [100, 110, 120,
                                                                    130, 140, 150,
                                                                    160, 170, 180,
                                                                    190, 200, 210,
                                                                    220, 230,
                                                                    240]},
```

```
                        verbose=True)
```

[ ]:

```

```

17.1.1 Evaluating the best tuned hyperparameters of RFC and best score as of tuning by RSCV

[96]:

```
rfc_rscv.best_params_, rfc_rscv.best_score_
```

[96]:

```
({'model__n_estimators': 130,
  'model__min_samples_split': 4,
  'model__min_samples_leaf': 2,
  'model__max_features': 'sqrt',
  'model__max_depth': None},
 np.float64(0.9648351648351647))
```

[97]:

```
rfc_rscv_best = rfc_rscv.best_estimator_
```

[98]:

```
rfc_rscv_best
```

[98]:

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('numeric',
                                                  Pipeline(steps=[('impute',
SimpleImputer())]),
                                                  ['mean radius',
                                                   'mean texture',
                                                   'mean perimeter',
                                                   'mean area',
                                                   'mean smoothness',
                                                   'mean compactness',
                                                   'mean concavity',
                                                   'mean concave points',
                                                   'mean symmetry',
                                                   'radius error',
                                                   'perimeter error',
                                                   'area error',
                                                   'compactness error',
                                                   'concavity error',
                                                   'concave points error',
                                                   'worst radius',
                                                   'worst texture',
                                                   'worst perimeter',
                                                   'worst area',
                                                   'worst smoothness',
                                                   'worst compactness',
                                                   'worst concavity',
                                                   'worst concave points',
```

```
                                                              'worst symmetry',
                                                              'worst fractal '
                                                              'dimension'])])),
                                ('model',
                                 RandomForestClassifier(min_samples_leaf=2, min_samples_split=4,
                                                        n_estimators=130))])
```

[ ]:

18 Evaluating new Classification Performance Metrics for RSCV Tuned Random Forest Clasifier

[108]:
```python
rfc_rscv_res = calculatePerformanceMetrics({"RSCV Tuned Random Forest␣
↪Classifier" : rfc_rscv_best}, ytest, xtest, printResults = True)
```

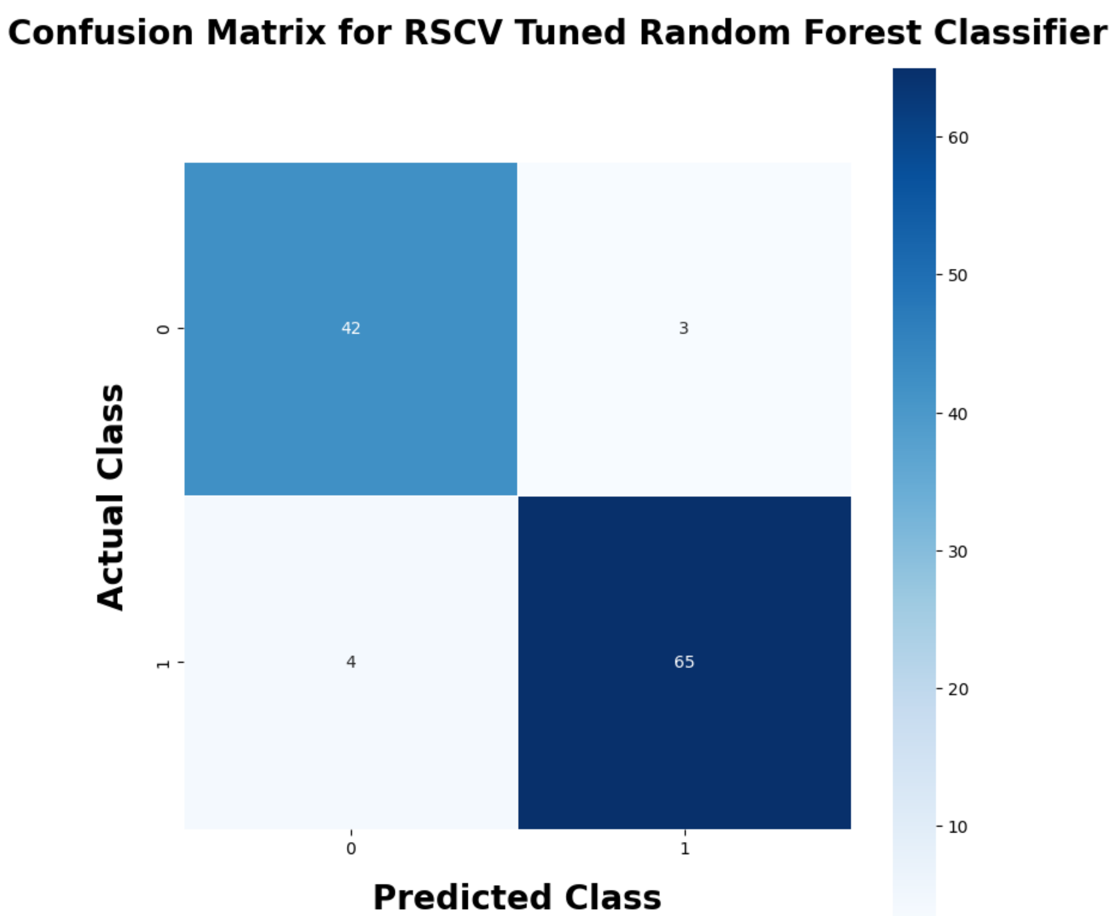
```
Calculating Performance Metrics for RSCV Tuned Random Forest Classifier:

Performance for Class = 0
Precision: 0.9130434782608695
Recall: 0.9333333333333333
F1 Score: 0.9230769230769231

Performance for Class = 1
Precision: 0.9558823529411765
Recall: 0.9420289855072463
F1 Score: 0.948905109489051

Mean Accuracy: 0.9385964912280702
```

[ ]:

19 Visualizing Confusion Matrix for RSCV Tuned Random Forest Classifier using HeatMap

[109]:
```python
fig, plot = plt.subplots(figsize = (8, 8))

sns.heatmap(
    confusion_matrix(ytest, rfc_rscv_best.predict(xtest)),
    annot = True,
    linewidths = 0.5,
    cmap = "Blues",
    cbar = True,
    square = True,
    ax = plot
)
```

```
fig.suptitle("Confusion Matrix for RSCV Tuned Random Forest Classifier",␣
 ↪fontsize = 20, fontweight = "bold")

plot.set_xlabel("Predicted Class", fontsize = 20, fontweight = "bold")
plot.set_ylabel("Actual Class", fontsize = 20, fontweight = "bold")

plot.xaxis.labelpad = 15
plot.yaxis.labelpad = 15

plt.tight_layout()
plt.show()
```



[ ]:

```

```

19.1 Using method of Grid Search Cross Validation (GSCV)

[110]:
```
rfc_gscv_params = {
    "model__max_features" : ["sqrt"],
    "model__n_estimators" : [num for num in range(150, 200, 10)],
    "model__min_samples_split" : [num for num in range(2, 5, 1)],
    "model__min_samples_leaf" : [num for num in range(1, 4, 1)],
    "model__max_depth" : [None]
}

rfc_gscv = GridSearchCV(
    estimator = rfc,
    cv = 5,
    param_grid = rfc_gscv_params,
    verbose = True
)
```

[111]:
```
rfc_gscv.fit(xtrain, ytrain)
```

```
Fitting 5 folds for each of 45 candidates, totalling 225 fits
```

[111]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('numeric',
Pipeline(steps=[('impute',
          SimpleImputer())]),
                                                                         ['mean
'
'radius',
                                                                          'mean
'
'texture',
                                                                          'mean
'
'perimeter',
                                                                          'mean
'
'area',
                                                                          'mean
'
'smoothness',
                                                                          'mean
'
'compactness',
                                                                          'mean
'
'concavity',
                                                                          'mean
```

```
'
'concave '
'points',
                                                                          'mean
'
'symmetry',
'radius '
'error',
'perimeter '
'error',
                                                                          'area
'…
'smoothness',
                                                                          'worst
'
'compactness',
                                                                          'worst
'
'concavity',
                                                                          'worst
'
'concave '
'points',
                                                                          'worst
'
'symmetry',
                                                                          'worst
'
'fractal '
'dimension'])])),
                                       ('model', RandomForestClassifier())]),
             param_grid={'model__max_depth': [None],
                         'model__max_features': ['sqrt'],
                         'model__min_samples_leaf': [1, 2, 3],
                         'model__min_samples_split': [2, 3, 4],
                         'model__n_estimators': [150, 160, 170, 180, 190]},
             verbose=True)
```

[ ]:

19.1.1 Evaluating the best tuned hyperparameters of RFC and best score as of tuning by RSCV

[112]:
```
rfc_gscv.best_params_, rfc_gscv.best_score_
```

[112]:
```
({'model__max_depth': None,
  'model__max_features': 'sqrt',
```

```
  'model__min_samples_leaf': 2,
  'model__min_samples_split': 2,
  'model__n_estimators': 170},
 np.float64(0.9626373626373625))
```

[113]:
```python
rfc_gscv_best = rfc_gscv.best_estimator_

rfc_gscv_best
```

[113]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('numeric',
                                                  Pipeline(steps=[('impute',
SimpleImputer())]),
                                                  ['mean radius',
                                                   'mean texture',
                                                   'mean perimeter',
                                                   'mean area',
                                                   'mean smoothness',
                                                   'mean compactness',
                                                   'mean concavity',
                                                   'mean concave points',
                                                   'mean symmetry',
                                                   'radius error',
                                                   'perimeter error',
                                                   'area error',
                                                   'compactness error',
                                                   'concavity error',
                                                   'concave points error',
                                                   'worst radius',
                                                   'worst texture',
                                                   'worst perimeter',
                                                   'worst area',
                                                   'worst smoothness',
                                                   'worst compactness',
                                                   'worst concavity',
                                                   'worst concave points',
                                                   'worst symmetry',
                                                   'worst fractal '
                                                   'dimension'])])),
                ('model',
                 RandomForestClassifier(min_samples_leaf=2, n_estimators=170))])
```

[ ]:
```python

```

20 Evaluating new Classification Performance Metrics for GSCV Tuned Random Forest Classifier

[114]:
```
rfc_gscv_res = calculatePerformanceMetrics({"GSCV Tuned Random Forest␣
↪Classifier" : rfc_gscv_best}, ytest, xtest, printResults = True)
```

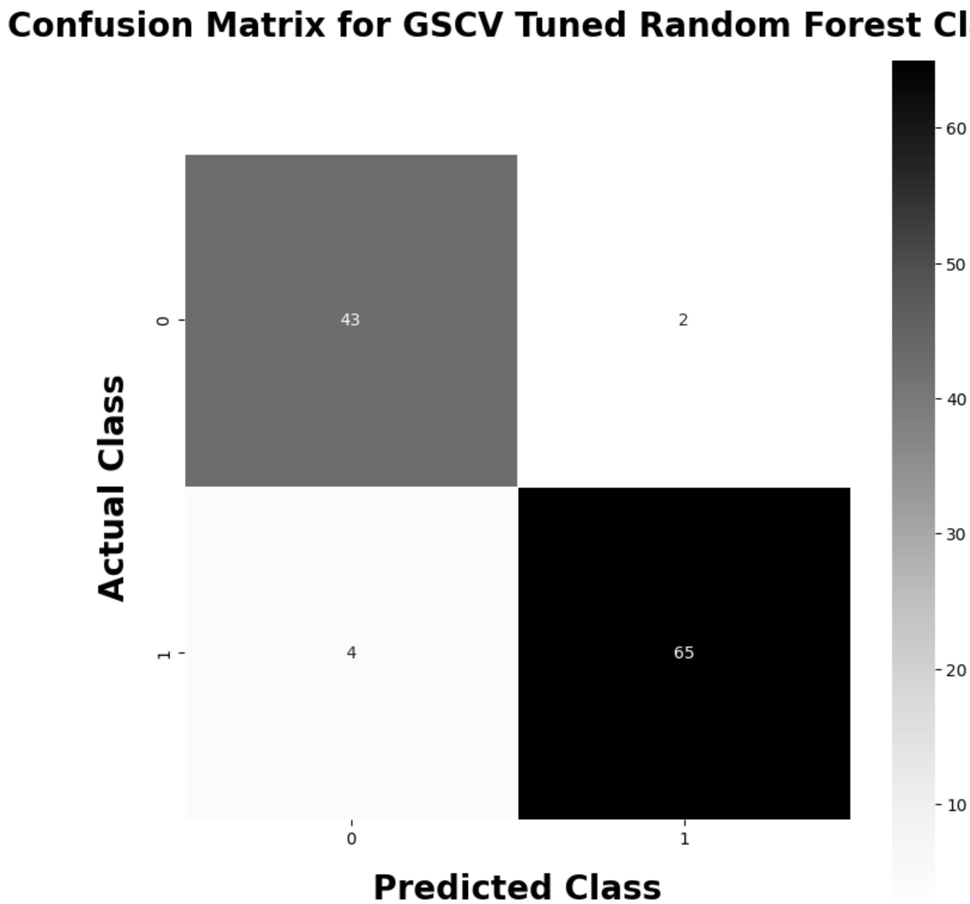
```
Calculating Performance Metrics for GSCV Tuned Random Forest Classifier:

Performance for Class = 0
Precision: 0.9148936170212766
Recall: 0.9555555555555556
F1 Score: 0.9347826086956522

Performance for Class = 1
Precision: 0.9701492537313433
Recall: 0.9420289855072463
F1 Score: 0.9558823529411765

Mean Accuracy: 0.9473684210526315
```

[ ]:

21 Visualizing Confusion Matrix for GSCV Tuned Random Forest Classifier

[115]:
```
fig, plot = plt.subplots(figsize = (8, 8))

sns.heatmap(
    confusion_matrix(ytest, rfc_gscv_best.predict(xtest)),
    annot = True,
    cmap = "Greys",
    linewidths = 0.5,
    square = True,
    cbar = True,
    ax = plot
)

fig.suptitle("Confusion Matrix for GSCV Tuned Random Forest Classifier",␣
↪fontsize = 20, fontweight = "bold")

plot.set_xlabel("Predicted Class", fontsize = 20, fontweight = "bold")
plot.set_ylabel("Actual Class", fontsize = 20, fontweight = "bold")

plot.xaxis.labelpad = 15
plot.yaxis.labelpad = 15
```

```
plt.tight_layout()
plt.show()
```



[ ]:

```

```

22 Comparing Confusion Matrices of all 3 versions of Random Forest Classifier (Non Tuned, RSCV Tuned, GSCV Tuned) using HeatMap

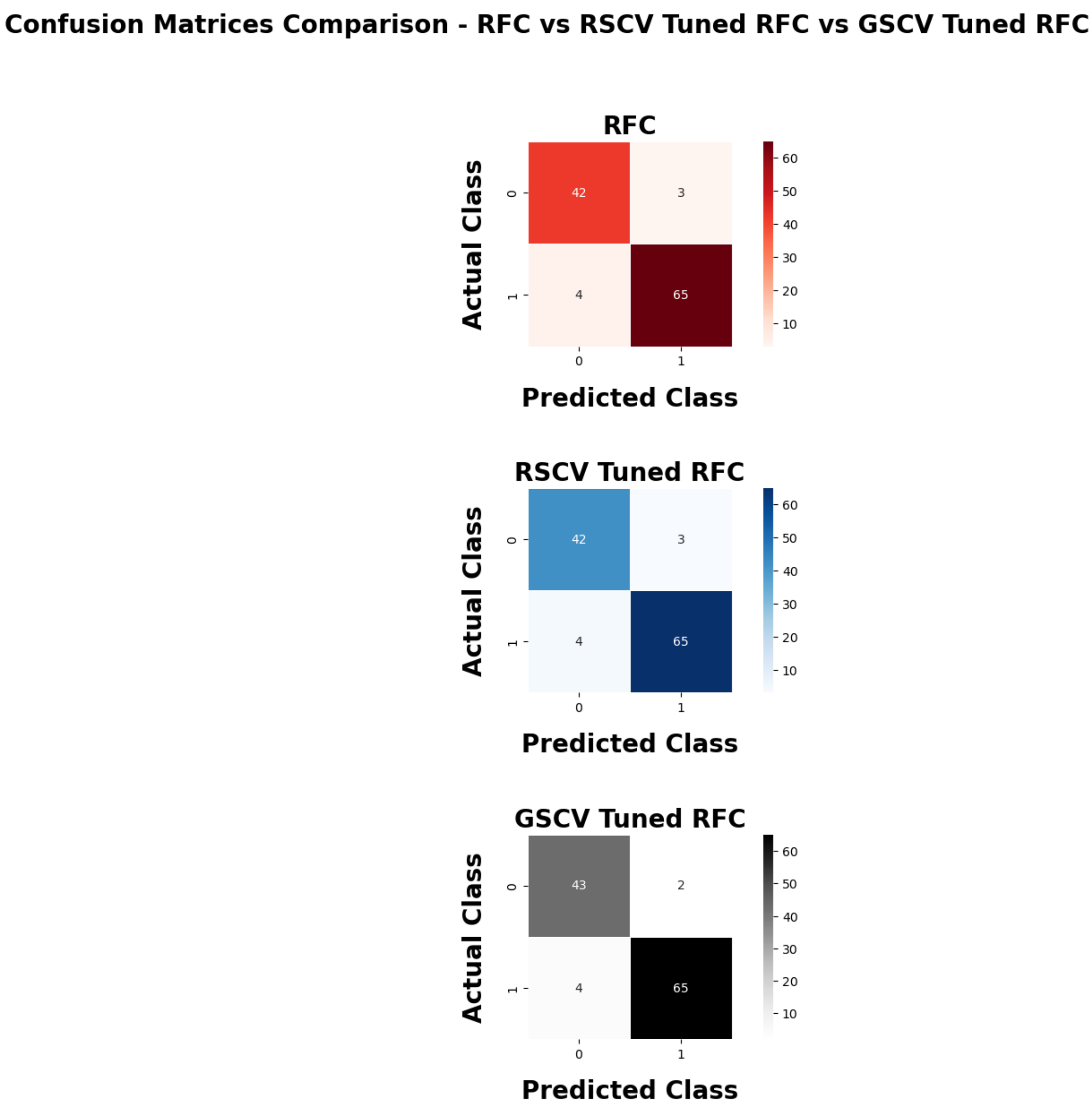
Comparing the confusion matrices for Non Tuned Random Forest Classifier, RSCV Tuned Random Forest Classifier and GSCV Tuned Random Forest Classifier

[179]:

```
fig, (plot1, plot2, plot3) = plt.subplots(3, 1, figsize = (8, 13))

sns.heatmap(
    confusion_matrix(ytest, rfc.predict(xtest)),
    annot = True,
```

```
    linewidths = 0.5,
    cmap = "Reds",
    cbar = True,
    square = True,
    ax = plot1
)

sns.heatmap(
    confusion_matrix(ytest, rfc_rscv_best.predict(xtest)),
    annot = True,
    linewidths = 0.5,
    cmap = "Blues",
    cbar = True,
    square = True,
    ax = plot2
)

sns.heatmap(
    confusion_matrix(ytest, rfc_gscv_best.predict(xtest)),
    annot = True,
    linewidths = 0.5,
    cmap = "Greys",
    cbar = True,
    square = True,
    ax = plot3
)

fig.suptitle("Confusion Matrices Comparison - RFC vs RSCV Tuned RFC vs GSCV␣
↪Tuned RFC", fontsize = 20, fontweight = "bold")

plot1.set_title("RFC", fontsize = 20, fontweight = "bold")
plot1.set_xlabel("Predicted Class", fontsize = 20, fontweight = "bold")
plot1.set_ylabel("Actual Class", fontsize = 20, fontweight = "bold")
plot1.xaxis.labelpad = 15
plot1.yaxis.labelpad = 15

plot2.set_title("RSCV Tuned RFC", fontsize = 20, fontweight = "bold")
plot2.set_xlabel("Predicted Class", fontsize = 20, fontweight = "bold")
plot2.set_ylabel("Actual Class", fontsize = 20, fontweight = "bold")
plot2.xaxis.labelpad = 15
plot2.yaxis.labelpad = 15

plot3.set_title("GSCV Tuned RFC", fontsize = 20, fontweight = "bold")
plot3.set_xlabel("Predicted Class", fontsize = 20, fontweight = "bold")
plot3.set_ylabel("Actual Class", fontsize = 20, fontweight = "bold")
plot3.xaxis.labelpad = 15
plot3.yaxis.labelpad = 15
```

```
fig.savefig("Confusion Matrix HeatMap between RFC, RSCV RFC and GSCV RFC.png")

plt.tight_layout(pad = 4.0)
plt.show()
```




```
[ ]:
```

23 Comparing Classification Performance Metrics for all 3 versions of Random Forest Classifier (Non Tuned, RSCV Tuned, GSCV Tuned) using BarPlot

Using a bar plot to visualize and interpret precision, recall and f1 score for all the three versions of the Random Forest Classifier after tuning

[165]:
```
rfcResults = calculatePerformanceMetrics(
    {
        "Random Forest Classifier" : rfc,
        "RSCV Random Forest Classifier" :  rfc_rscv_best,
        "GSCV Random Forest Classifier" : rfc_gscv_best
    },
    ytest,
    xtest,
    printResults = False
)
```

[166]:
```
rfcResultsFrame0, rfcResultsFrame1 = rfcResults[0], rfcResults[1]
```

[167]:
```
rfcResultsFrame0
```

[167]:

	Precision	Recall	F1 Score
Random Forest Classifier	0.913043	0.933333	0.923077
RSCV Random Forest Classifier	0.913043	0.933333	0.923077
GSCV Random Forest Classifier	0.914894	0.955556	0.934783

[168]:
```
rfcResultsFrame1
```

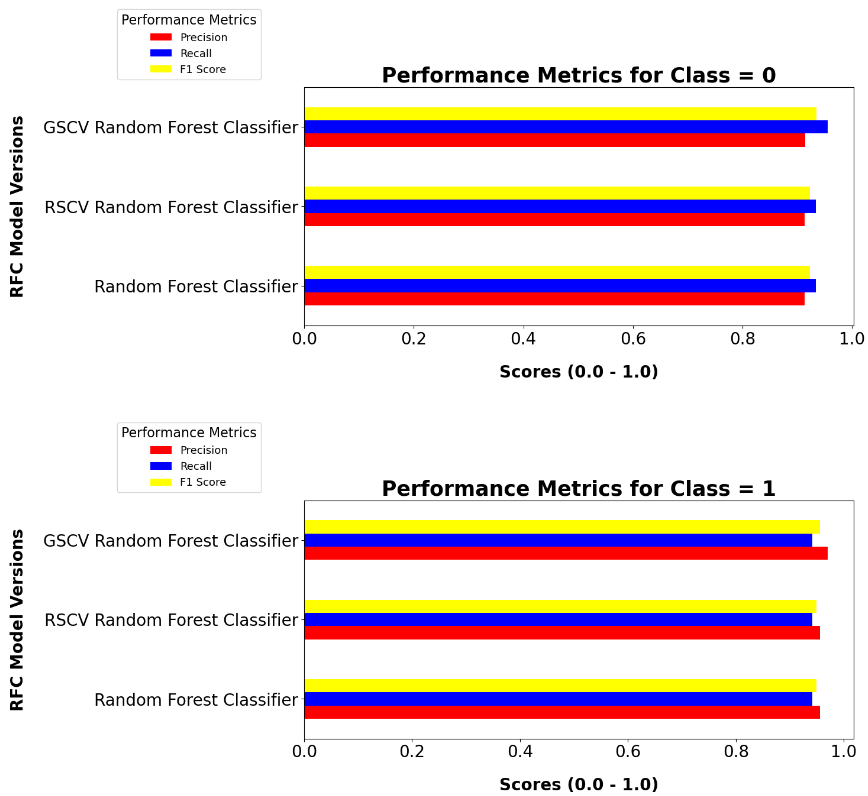
[168]:

	Precision	Recall	F1 Score
Random Forest Classifier	0.955882	0.942029	0.948905
RSCV Random Forest Classifier	0.955882	0.942029	0.948905
GSCV Random Forest Classifier	0.970149	0.942029	0.955882

[178]:
```
fig, (plot1, plot2) = plt.subplots(2, 1, figsize = (15, 15))

rfcResultsFrame0.plot(kind = "barh", ax = plot1, color = ["Red", "Blue",␣
↪"Yellow"])

fig.suptitle("Comparison of Performace Metrics for RFC, RSCV RFC and GSCV RFC",␣
↪fontsize = 35, fontweight = "bold", y = 1.02)

plot1.set_title("Performance Metrics for Class = 0", fontsize = 25, fontweight␣
↪= "bold")
plot1.set_ylabel("RFC Model Versions", fontsize = 20, fontweight = "bold")
plot1.set_xlabel("Scores (0.0 - 1.0)", fontsize = 20, fontweight = "bold")
```

```
plot1.xaxis.labelpad = 20
plot1.yaxis.labelpad = 20

plot1.tick_params(axis = "both", labelsize = 20)

plot1.legend(
    title = "Performance Metrics",
    fontsize = 13,
    title_fontsize = 16,
    loc = "upper left",
    bbox_to_anchor = (-0.35, 1.35)
)

rfcResultsFrame1.plot(kind = "barh", ax = plot2, color = ["Red", "Blue",␣
 ↪"Yellow"])

plot2.set_title("Performance Metrics for Class = 1", fontsize = 25, fontweight␣
 ↪= "bold")
plot2.set_ylabel("RFC Model Versions", fontsize = 20, fontweight = "bold")
plot2.set_xlabel("Scores (0.0 - 1.0)", fontsize = 20, fontweight = "bold")

plot2.xaxis.labelpad = 20
plot2.yaxis.labelpad = 20

plot2.tick_params(axis = "both", labelsize = 20)

plot2.legend(
    title = "Performance Metrics",
    fontsize = 13,
    title_fontsize = 16,
    loc = "upper left",
    bbox_to_anchor = (-0.35, 1.35)
)

fig.savefig("Performance Metrics BarPlot between RFC, RSCV RFC and GSCV RFC.
 ↪png")

plt.tight_layout(pad = 2.0)
plt.show()
```

Comparison of Performace Metrics for RFC, RSCV RFC and GSCV RFC



[ ]:

```

```

23.1 Saving the final model (after tuning) = GSCV Tuned Random Forest Classifier

After all operations, the GSCV Tuned version of Random Forest Classifier is chosen as the final and best version for this model (and also among the other two models, SVC and LR)

[172]:

```
best_model = rfc_gscv_best

dump(best_model, "best_model.joblib")
```

[172]: ['best_model.joblib']

[ ]:

```

```