breast_cancer_detection

March 21, 2022

1 PROBLEM STATEMENT

Breast cancer is one of the most common causes of death among women worldwide. Early-stage detection would help the world to reduce the number of deaths. Breast cancer occurs when the cells begin to multiply abnormally and will result in a lump. The lumps present in the human body can be classified as cancerous and non-cancerous. Non-cancerous tumors are pretty common and they don't have any symptoms.

UCI has shared a historical dataset that describes the diagnosis results of whether a tumor is malignant or benign. Malignant tumors are cancerous whereas Benign tumors are non-cancerous. As a Data scientist, you are assigned to create a fully automated system that uses the columns(characteristics) in the dataset to predict if a tumor in the breast is malignant or benign.

[49]:
```
#Loading libraries

import pandas as pd
import sklearn.model_selection
import sklearn.metrics
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
```

[50]:
```
#Loading the dataset
df = pd.read_csv("../../datasets/breast.csv")
df
```

[50]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	\
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	
..	…	…	…	…	…	…	
564	926424	M	21.56	22.39	142.00	1479.0	
565	926682	M	20.13	28.25	131.20	1261.0	
566	926954	M	16.60	28.08	108.30	858.1	
567	927241	M	20.60	29.33	140.10	1265.0	
568	92751	B	7.76	24.54	47.92	181.0	

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	\
0	0.11840	0.27760	0.30010	0.14710	
1	0.08474	0.07864	0.08690	0.07017	
2	0.10960	0.15990	0.19740	0.12790	
3	0.14250	0.28390	0.24140	0.10520	
4	0.10030	0.13280	0.19800	0.10430	
..	…	…	…	…	
564	0.11100	0.11590	0.24390	0.13890	
565	0.09780	0.10340	0.14400	0.09791	
566	0.08455	0.10230	0.09251	0.05302	
567	0.11780	0.27700	0.35140	0.15200	
568	0.05263	0.04362	0.00000	0.00000	

	…	texture_worst	perimeter_worst	area_worst	smoothness_worst	\
0	…	17.33	184.60	2019.0	0.16220	
1	…	23.41	158.80	1956.0	0.12380	
2	…	25.53	152.50	1709.0	0.14440	
3	…	26.50	98.87	567.7	0.20980	
4	…	16.67	152.20	1575.0	0.13740	
..	…	…	…	…	…	
564	…	26.40	166.10	2027.0	0.14100	
565	…	38.25	155.00	1731.0	0.11660	
566	…	34.12	126.70	1124.0	0.11390	
567	…	39.42	184.60	1821.0	0.16500	
568	…	30.37	59.16	268.6	0.08996	

	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	\
0	0.66560	0.7119	0.2654	0.4601	
1	0.18660	0.2416	0.1860	0.2750	
2	0.42450	0.4504	0.2430	0.3613	
3	0.86630	0.6869	0.2575	0.6638	
4	0.20500	0.4000	0.1625	0.2364	
..	…	…	…	…	
564	0.21130	0.4107	0.2216	0.2060	

```
565          0.19220          0.3215           0.1628          0.2572
566          0.30940          0.3403           0.1418          0.2218
567          0.86810          0.9387           0.2650          0.4087
568          0.06444          0.0000           0.0000          0.2871

     fractal_dimension_worst  Unnamed: 32
0                    0.11890          NaN
1                    0.08902          NaN
2                    0.08758          NaN
3                    0.17300          NaN
4                    0.07678          NaN
..                       …            …
564                  0.07115          NaN
565                  0.06637          NaN
566                  0.07820          NaN
567                  0.12400          NaN
568                  0.07039          NaN

[569 rows x 33 columns]
```

The dataset contains 569 rows and 33 columns. Each row represents a unique entry for the breast cancer diagnosis results. Our target is to predict the value in the column "diagnosis".

[3]:
```
df["diagnosis"].unique()
```

[3]:
```
array(['M', 'B'], dtype=object)
```

The diagnosis column has the unique values "M" and "B" where "M" represents Malignant and "B" represents Benign.

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
```

```
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

The dataset contains categorical and numerical columns. In this dataset, the flat column "diagnosis" is our target. We found columns "unnamed: 32" in this dataset with null values. We can drop the column "Unnamed: 32" and "id" as it does not help build the model.

[5]:
```
#Drop the column with null values
df.drop(["Unnamed: 32", "id"], axis=1, inplace=True)
```

[6]:
```
#Replacing the Values in target column "diagnosis"
df.replace({"M": 1, "B": 0}, inplace=True)
```

2 EXPLORATORY DATA ANALYSIS

[7]:
```
px.histogram(df, x="diagnosis", title="diagnosis results count")
```

We have received a data set with 596 entries; 212 cases are diagnosed as "Malignant," whereas 357 cases are diagnosed as "Benign".

[8]:
```
rad_scat = px.scatter(df, x="radius_mean", color="diagnosis",
                      title="radius_mean related to Diagnosis")
rad_scat.update_traces(marker_size=5)
rad_scat.show()
```

From the above plot, we can find 2 clusters. As the value of radius_mean increases above 15, there is a high chance that the tumor is malignant.

[47]:
```
text_scat = px.scatter(df, x="texture_mean", color="diagnosis",
                       title="texture_mean related to Diagnosis")
text_scat.update_traces(marker_size=5)
text_scat.show()
```

We can observe that if the texture_mean value is less than 15, there is a high chance that the tumor is Benign.

[11]:
```
compact_scat = px.scatter(df, x="compactness_se", color="diagnosis",
                          title="compactness_se vs diagnosis")
compact_scat.update_traces(marker_size=5)
compact_scat.show()
```

In this relationship we found that if the compactness_se value is greater than 0.02 there is a high chance that the tumor is malignant

[12]:
```
concav_scat = px.scatter(df, x="concavity_se", color="diagnosis",
                         title="concavity_se vs diagnosis")
concav_scat.update_traces(marker_size=5)
concav_scat.show()
```

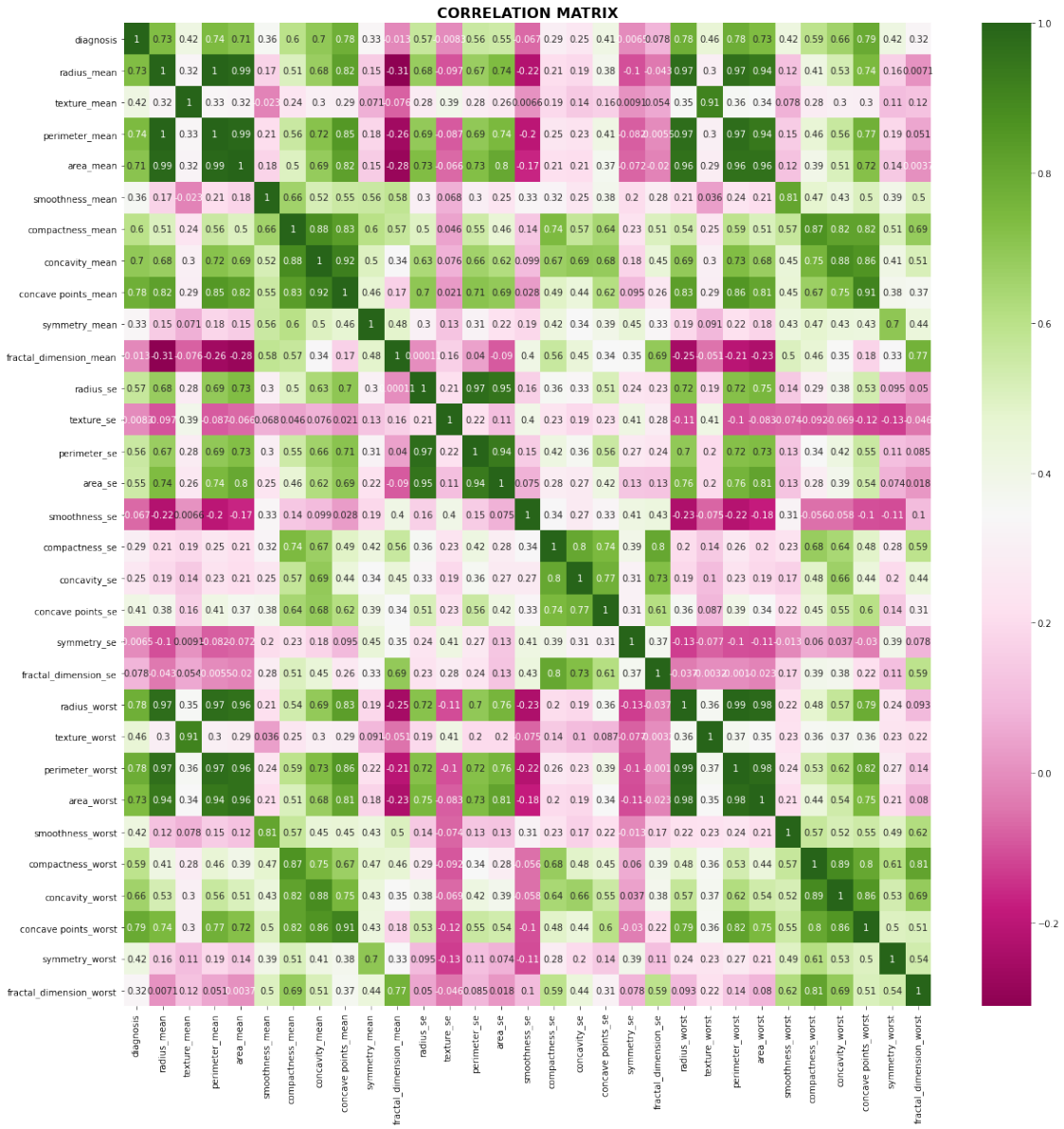
We can find 2 clusters in this plot; if the value of concavity_se is more significant than 0.03, there is a high chance that the tumor is malignant

[13]:
```
concavity_scat = px.scatter(df, x="concave points_se", color="diagnosis",
                            title="concavity points_se vs diagnosis")
concavity_scat.update_traces(marker_size=5)
concavity_scat.show()
```

In this plot also, we can find 2 clusters, where if the value of concavity point_se is more significant than 0.01274, then there is a high chance for the tumor to be malignant. In this plot, we found cases where the concave points_se has a value 0 with a clear case of Benign. Our current data set contains 32 columns, and let's focus on finding a correlation matrix for the automated system to differentiate cancerous and non-cancerous tumors.

[14]:
```
#Let's create a correlation matrix
correlation = df.corr()
plt.figure(figsize=(20, 20))
sns.heatmap(df.corr(), cmap="PiYG", annot=True)
plt.title("CORRELATION MATRIX", fontweight="bold", fontsize=16)
```

[14]:
```
Text(0.5, 1.0, 'CORRELATION MATRIX')
```



[15]:
```
df.corr()
```

[15]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	\
diagnosis	1.000000	0.730029	0.415185	0.742636	
radius_mean	0.730029	1.000000	0.323782	0.997855	
texture_mean	0.415185	0.323782	1.000000	0.329533	
perimeter_mean	0.742636	0.997855	0.329533	1.000000	
area_mean	0.708984	0.987357	0.321086	0.986507	
smoothness_mean	0.358560	0.170581	−0.023389	0.207278	
compactness_mean	0.596534	0.506124	0.236702	0.556936	
concavity_mean	0.696360	0.676764	0.302418	0.716136	

concave points_mean	0.776614	0.822529	0.293464	0.850977
symmetry_mean	0.330499	0.147741	0.071401	0.183027
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477
radius_se	0.567134	0.679090	0.275869	0.691765
texture_se	-0.008303	-0.097317	0.386358	-0.086761
perimeter_se	0.556141	0.674172	0.281673	0.693135
area_se	0.548236	0.735864	0.259845	0.744983
smoothness_se	-0.067016	-0.222600	0.006614	-0.202694
compactness_se	0.292999	0.206000	0.191975	0.250744
concavity_se	0.253730	0.194204	0.143293	0.228082
concave points_se	0.408042	0.376169	0.163851	0.407217
symmetry_se	-0.006522	-0.104321	0.009127	-0.081629
fractal_dimension_se	0.077972	-0.042641	0.054458	-0.005523
radius_worst	0.776454	0.969539	0.352573	0.969476
texture_worst	0.456903	0.297008	0.912045	0.303038
perimeter_worst	0.782914	0.965137	0.358040	0.970387
area_worst	0.733825	0.941082	0.343546	0.941550
smoothness_worst	0.421465	0.119616	0.077503	0.150549
compactness_worst	0.590998	0.413463	0.277830	0.455774
concavity_worst	0.659610	0.526911	0.301025	0.563879
concave points_worst	0.793566	0.744214	0.295316	0.771241
symmetry_worst	0.416294	0.163953	0.105008	0.189115
fractal_dimension_worst	0.323872	0.007066	0.119205	0.051019

	area_mean	smoothness_mean	compactness_mean	\
diagnosis	0.708984	0.358560	0.596534	
radius_mean	0.987357	0.170581	0.506124	
texture_mean	0.321086	-0.023389	0.236702	
perimeter_mean	0.986507	0.207278	0.556936	
area_mean	1.000000	0.177028	0.498502	
smoothness_mean	0.177028	1.000000	0.659123	
compactness_mean	0.498502	0.659123	1.000000	
concavity_mean	0.685983	0.521984	0.883121	
concave points_mean	0.823269	0.553695	0.831135	
symmetry_mean	0.151293	0.557775	0.602641	
fractal_dimension_mean	-0.283110	0.584792	0.565369	
radius_se	0.732562	0.301467	0.497473	
texture_se	-0.066280	0.068406	0.046205	
perimeter_se	0.726628	0.296092	0.548905	
area_se	0.800086	0.246552	0.455653	
smoothness_se	-0.166777	0.332375	0.135299	
compactness_se	0.212583	0.318943	0.738722	
concavity_se	0.207660	0.248396	0.570517	
concave points_se	0.372320	0.380676	0.642262	
symmetry_se	-0.072497	0.200774	0.229977	
fractal_dimension_se	-0.019887	0.283607	0.507318	
radius_worst	0.962746	0.213120	0.535315	

texture_worst	0.287489	0.036072	0.248133
perimeter_worst	0.959120	0.238853	0.590210
area_worst	0.959213	0.206718	0.509604
smoothness_worst	0.123523	0.805324	0.565541
compactness_worst	0.390410	0.472468	0.865809
concavity_worst	0.512606	0.434926	0.816275
concave points_worst	0.722017	0.503053	0.815573
symmetry_worst	0.143570	0.394309	0.510223
fractal_dimension_worst	0.003738	0.499316	0.687382

	concavity_mean	concave points_mean	symmetry_mean	\
diagnosis	0.696360	0.776614	0.330499	
radius_mean	0.676764	0.822529	0.147741	
texture_mean	0.302418	0.293464	0.071401	
perimeter_mean	0.716136	0.850977	0.183027	
area_mean	0.685983	0.823269	0.151293	
smoothness_mean	0.521984	0.553695	0.557775	
compactness_mean	0.883121	0.831135	0.602641	
concavity_mean	1.000000	0.921391	0.500667	
concave points_mean	0.921391	1.000000	0.462497	
symmetry_mean	0.500667	0.462497	1.000000	
fractal_dimension_mean	0.336783	0.166917	0.479921	
radius_se	0.631925	0.698050	0.303379	
texture_se	0.076218	0.021480	0.128053	
perimeter_se	0.660391	0.710650	0.313893	
area_se	0.617427	0.690299	0.223970	
smoothness_se	0.098564	0.027653	0.187321	
compactness_se	0.670279	0.490424	0.421659	
concavity_se	0.691270	0.439167	0.342627	
concave points_se	0.683260	0.615634	0.393298	
symmetry_se	0.178009	0.095351	0.449137	
fractal_dimension_se	0.449301	0.257584	0.331786	
radius_worst	0.688236	0.830318	0.185728	
texture_worst	0.299879	0.292752	0.090651	
perimeter_worst	0.729565	0.855923	0.219169	
area_worst	0.675987	0.809630	0.177193	
smoothness_worst	0.448822	0.452753	0.426675	
compactness_worst	0.754968	0.667454	0.473200	
concavity_worst	0.884103	0.752399	0.433721	
concave points_worst	0.861323	0.910155	0.430297	
symmetry_worst	0.409464	0.375744	0.699826	
fractal_dimension_worst	0.514930	0.368661	0.438413	

	…	radius_worst	texture_worst	perimeter_worst	\
diagnosis	…	0.776454	0.456903	0.782914	
radius_mean	…	0.969539	0.297008	0.965137	
texture_mean	…	0.352573	0.912045	0.358040	

perimeter_mean	…	0.969476	0.303038	0.970387
area_mean	…	0.962746	0.287489	0.959120
smoothness_mean	…	0.213120	0.036072	0.238853
compactness_mean	…	0.535315	0.248133	0.590210
concavity_mean	…	0.688236	0.299879	0.729565
concave points_mean	…	0.830318	0.292752	0.855923
symmetry_mean	…	0.185728	0.090651	0.219169
fractal_dimension_mean	…	-0.253691	-0.051269	-0.205151
radius_se	…	0.715065	0.194799	0.719684
texture_se	…	-0.111690	0.409003	-0.102242
perimeter_se	…	0.697201	0.200371	0.721031
area_se	…	0.757373	0.196497	0.761213
smoothness_se	…	-0.230691	-0.074743	-0.217304
compactness_se	…	0.204607	0.143003	0.260516
concavity_se	…	0.186904	0.100241	0.226680
concave points_se	…	0.358127	0.086741	0.394999
symmetry_se	…	-0.128121	-0.077473	-0.103753
fractal_dimension_se	…	-0.037488	-0.003195	-0.001000
radius_worst	…	1.000000	0.359921	0.993708
texture_worst	…	0.359921	1.000000	0.365098
perimeter_worst	…	0.993708	0.365098	1.000000
area_worst	…	0.984015	0.345842	0.977578
smoothness_worst	…	0.216574	0.225429	0.236775
compactness_worst	…	0.475820	0.360832	0.529408
concavity_worst	…	0.573975	0.368366	0.618344
concave points_worst	…	0.787424	0.359755	0.816322
symmetry_worst	…	0.243529	0.233027	0.269493
fractal_dimension_worst	…	0.093492	0.219122	0.138957

	area_worst	smoothness_worst	compactness_worst	\
diagnosis	0.733825	0.421465	0.590998	
radius_mean	0.941082	0.119616	0.413463	
texture_mean	0.343546	0.077503	0.277830	
perimeter_mean	0.941550	0.150549	0.455774	
area_mean	0.959213	0.123523	0.390410	
smoothness_mean	0.206718	0.805324	0.472468	
compactness_mean	0.509604	0.565541	0.865809	
concavity_mean	0.675987	0.448822	0.754968	
concave points_mean	0.809630	0.452753	0.667454	
symmetry_mean	0.177193	0.426675	0.473200	
fractal_dimension_mean	-0.231854	0.504942	0.458798	
radius_se	0.751548	0.141919	0.287103	
texture_se	-0.083195	-0.073658	-0.092439	
perimeter_se	0.730713	0.130054	0.341919	
area_se	0.811408	0.125389	0.283257	
smoothness_se	-0.182195	0.314457	-0.055558	
compactness_se	0.199371	0.227394	0.678780	

concavity_se	0.188353	0.168481	0.484858
concave points_se	0.342271	0.215351	0.452888
symmetry_se	-0.110343	-0.012662	0.060255
fractal_dimension_se	-0.022736	0.170568	0.390159
radius_worst	0.984015	0.216574	0.475820
texture_worst	0.345842	0.225429	0.360832
perimeter_worst	0.977578	0.236775	0.529408
area_worst	1.000000	0.209145	0.438296
smoothness_worst	0.209145	1.000000	0.568187
compactness_worst	0.438296	0.568187	1.000000
concavity_worst	0.543331	0.518523	0.892261
concave points_worst	0.747419	0.547691	0.801080
symmetry_worst	0.209146	0.493838	0.614441
fractal_dimension_worst	0.079647	0.617624	0.810455

	concavity_worst	concave points_worst	\
diagnosis	0.659610	0.793566	
radius_mean	0.526911	0.744214	
texture_mean	0.301025	0.295316	
perimeter_mean	0.563879	0.771241	
area_mean	0.512606	0.722017	
smoothness_mean	0.434926	0.503053	
compactness_mean	0.816275	0.815573	
concavity_mean	0.884103	0.861323	
concave points_mean	0.752399	0.910155	
symmetry_mean	0.433721	0.430297	
fractal_dimension_mean	0.346234	0.175325	
radius_se	0.380585	0.531062	
texture_se	-0.068956	-0.119638	
perimeter_se	0.418899	0.554897	
area_se	0.385100	0.538166	
smoothness_se	-0.058298	-0.102007	
compactness_se	0.639147	0.483208	
concavity_se	0.662564	0.440472	
concave points_se	0.549592	0.602450	
symmetry_se	0.037119	-0.030413	
fractal_dimension_se	0.379975	0.215204	
radius_worst	0.573975	0.787424	
texture_worst	0.368366	0.359755	
perimeter_worst	0.618344	0.816322	
area_worst	0.543331	0.747419	
smoothness_worst	0.518523	0.547691	
compactness_worst	0.892261	0.801080	
concavity_worst	1.000000	0.855434	
concave points_worst	0.855434	1.000000	
symmetry_worst	0.532520	0.502528	
fractal_dimension_worst	0.686511	0.511114	

	symmetry_worst	fractal_dimension_worst
diagnosis	0.416294	0.323872
radius_mean	0.163953	0.007066
texture_mean	0.105008	0.119205
perimeter_mean	0.189115	0.051019
area_mean	0.143570	0.003738
smoothness_mean	0.394309	0.499316
compactness_mean	0.510223	0.687382
concavity_mean	0.409464	0.514930
concave points_mean	0.375744	0.368661
symmetry_mean	0.699826	0.438413
fractal_dimension_mean	0.334019	0.767297
radius_se	0.094543	0.049559
texture_se	-0.128215	-0.045655
perimeter_se	0.109930	0.085433
area_se	0.074126	0.017539
smoothness_se	-0.107342	0.101480
compactness_se	0.277878	0.590973
concavity_se	0.197788	0.439329
concave points_se	0.143116	0.310655
symmetry_se	0.389402	0.078079
fractal_dimension_se	0.111094	0.591328
radius_worst	0.243529	0.093492
texture_worst	0.233027	0.219122
perimeter_worst	0.269493	0.138957
area_worst	0.209146	0.079647
smoothness_worst	0.493838	0.617624
compactness_worst	0.614441	0.810455
concavity_worst	0.532520	0.686511
concave points_worst	0.502528	0.511114
symmetry_worst	1.000000	0.537848
fractal_dimension_worst	0.537848	1.000000

[31 rows x 31 columns]

[16]:
```
#We are going to generate the column that has correlation value greater than 59
correlation[abs(correlation["diagnosis"]) > 0.59].index

#Create inputs and targets
inputs = df[["radius_mean", "perimeter_mean", "area_mean",
             "compactness_mean", "concavity_mean", "concave points_mean",
             "radius_worst", "perimeter_worst", "area_worst",
             "compactness_worst", "concavity_worst", "concave points_worst"]]
target = "diagnosis"
```

3 SCALING NUMERIC FEATURE

We can find a wide range of values for each numerical feature. Hence we need to scale the numerical values to ensure that any particular matter doesn't cause a disproportionate impact on the loss value of the model.

[17]:
```
numerical_value = df[["radius_mean", "perimeter_mean", "area_mean",
                      "compactness_mean", "concavity_mean",
                      "concave points_mean", "radius_worst",
                      "perimeter_worst", "area_worst", "compactness_worst",
                      "concavity_worst", "concave points_worst"]]

target = df["diagnosis"]


#scaling the data
scaler = StandardScaler()
scaler.fit(numerical_value)
scaled_value = scaler.transform(numerical_value)
```

4 SPLITTING DATA SET INTO TRAINING SET AND TEST SET

[18]:
```
X = scaled_value
Y = target
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
```

5 LOGISTIC REGRESSION MODEL WITH HYPER PARAMETER TUNING

[35]:
```
#create object model
model = LogisticRegression()

#define the parameters
parameters_grid = parameters_grid = [
    {"penalty": ["l2"],"C": [100, 10, 1.0, 0.1, 0.01],
     "solver": ["lbfgs","newton-cg","liblinear"],
     "max_iter": [100, 1000,2500, 5000]
    }
]
#definining Gridsearch
model_2 = GridSearchCV(model, param_grid=parameters_grid,
                       cv=3, verbose=True, n_jobs=-1)
```

```
#Fit model to data
model_2.fit(X_train, Y_train)

#Generate accuracy and best Hperparameters
print("Accuracy of best logistic regression classfier= {:.2f}"
     .format(model_2.best_score_))
print("Best found hyperparameters of logistic regression classifier= {}"
      .format(model_2.best_params_))
```

```
Fitting 3 folds for each of 60 candidates, totalling 180 fits
Accuracy of best logistic regression classfier= 0.95
Best found hyperparameters of logistic regression classifier= {'C': 10,
'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}
```

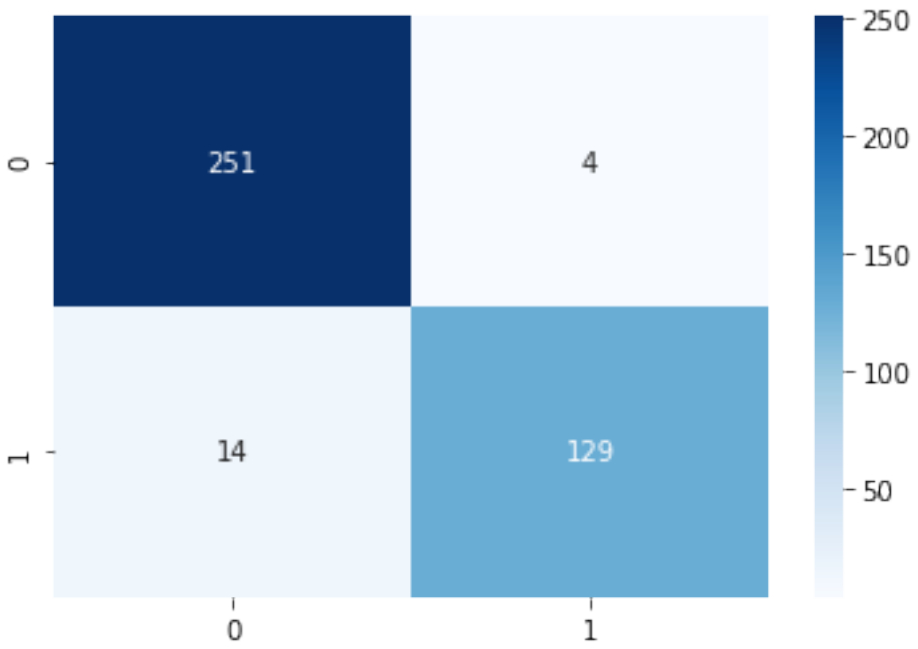
6 SVM

[20]:
```
#Create and train the model
model1 = SVC(C=1, kernel="rbf", cache_size=200)
model1.fit(X_train, Y_train)

#Generate predictions
predictions1 = model1.predict(X_train)

#Generate classification report and confusion matrix
print(classification_report(Y_train, predictions1))
print(confusion_matrix(Y_train, predictions1))
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       255
           1       0.97      0.90      0.93       143

    accuracy                           0.95       398
   macro avg       0.96      0.94      0.95       398
weighted avg       0.96      0.95      0.95       398

[[251   4]
 [ 14 129]]
```

[22]:
```
sns.heatmap(confusion_matrix(Y_train, predictions1),
            annot=True, cmap="Blues", fmt="g")
```

[22]:
```
<AxesSubplot:>
```



As per the above report generated, this model has a precision value of 95% for predicting Bening and 97% for predicting Malignant. From the above confusion matrix results, our model is giving 100% "True Negative" results, 3% "False positive," 6% False-negative results, and 60% True Positive results.

True Negative means the actual test result is "B," and the predicted result is "B."

False Positive means the actual test result is "B," and the predicted result is "M."

False-negative means the actual test result is "M," and the predicted result is "B."

True Positive means the actual test result is "M," and the predicted result is "M."

7 SVM WITH HYPERPARAMTER TUNING

[36]:
```
#Define parameters
parameters_grid = {
    "kernel": ["rbf"],
    "C": [0.001, 0.01, 0.1, 1, 10, 100],
    "gamma": [1, 0.1, 0.01, 0.001, 0.0001]
}

#Define Grid search
model2 = sklearn.model_selection.GridSearchCV(sklearn.svm.SVC(),
                                               parameters_grid,
                                               scoring="accuracy",
```

```
                                                cv=5, verbose=3)
#Fit the model to data
model2.fit(X_train, Y_train)


#Generate accuracy and best hyperparameters
print("Accuracy of best SVM classfier = {:.2f}".format(model2.best_score_))
print("Best found hyperparameters of SVM classifier = {}".format(model2.
 ↪best_params_))
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
[CV 1/5] END …C=0.001, gamma=1, kernel=rbf;, score=0.637 total time=   0.0s
[CV 2/5] END …C=0.001, gamma=1, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END …C=0.001, gamma=1, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END …C=0.001, gamma=1, kernel=rbf;, score=0.646 total time=   0.0s
[CV 5/5] END …C=0.001, gamma=1, kernel=rbf;, score=0.646 total time=   0.0s
[CV 1/5] END …C=0.001, gamma=0.1, kernel=rbf;, score=0.637 total time=   0.0s
[CV 2/5] END …C=0.001, gamma=0.1, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END …C=0.001, gamma=0.1, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END …C=0.001, gamma=0.1, kernel=rbf;, score=0.646 total time=   0.0s
[CV 5/5] END …C=0.001, gamma=0.1, kernel=rbf;, score=0.646 total time=   0.0s
[CV 1/5] END …C=0.001, gamma=0.01, kernel=rbf;, score=0.637 total time=   0.0s
[CV 2/5] END …C=0.001, gamma=0.01, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END …C=0.001, gamma=0.01, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END …C=0.001, gamma=0.01, kernel=rbf;, score=0.646 total time=   0.0s
[CV 5/5] END …C=0.001, gamma=0.01, kernel=rbf;, score=0.646 total time=   0.0s
[CV 1/5] END ..C=0.001, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 2/5] END ..C=0.001, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END ..C=0.001, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END ..C=0.001, gamma=0.001, kernel=rbf;, score=0.646 total time=   0.0s
[CV 5/5] END ..C=0.001, gamma=0.001, kernel=rbf;, score=0.646 total time=   0.0s
[CV 1/5] END .C=0.001, gamma=0.0001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 2/5] END .C=0.001, gamma=0.0001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END .C=0.001, gamma=0.0001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END .C=0.001, gamma=0.0001, kernel=rbf;, score=0.646 total time=   0.0s
[CV 5/5] END .C=0.001, gamma=0.0001, kernel=rbf;, score=0.646 total time=   0.0s
[CV 1/5] END …C=0.01, gamma=1, kernel=rbf;, score=0.637 total time=   0.0s
[CV 2/5] END …C=0.01, gamma=1, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END …C=0.01, gamma=1, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END …C=0.01, gamma=1, kernel=rbf;, score=0.646 total time=   0.0s
[CV 5/5] END …C=0.01, gamma=1, kernel=rbf;, score=0.646 total time=   0.0s
[CV 1/5] END …C=0.01, gamma=0.1, kernel=rbf;, score=0.637 total time=   0.0s
[CV 2/5] END …C=0.01, gamma=0.1, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END …C=0.01, gamma=0.1, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END …C=0.01, gamma=0.1, kernel=rbf;, score=0.646 total time=   0.0s
[CV 5/5] END …C=0.01, gamma=0.1, kernel=rbf;, score=0.646 total time=   0.0s
[CV 1/5] END …C=0.01, gamma=0.01, kernel=rbf;, score=0.637 total time=   0.0s
```

```
[CV 2/5] END …C=0.01, gamma=0.01, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END …C=0.01, gamma=0.01, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END …C=0.01, gamma=0.01, kernel=rbf;, score=0.646 total time=   0.0s
[CV 5/5] END …C=0.01, gamma=0.01, kernel=rbf;, score=0.646 total time=   0.0s
[CV 1/5] END …C=0.01, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 2/5] END …C=0.01, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END …C=0.01, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END …C=0.01, gamma=0.001, kernel=rbf;, score=0.646 total time=   0.0s
[CV 5/5] END …C=0.01, gamma=0.001, kernel=rbf;, score=0.646 total time=   0.0s
[CV 1/5] END ..C=0.01, gamma=0.0001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 2/5] END ..C=0.01, gamma=0.0001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END ..C=0.01, gamma=0.0001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END ..C=0.01, gamma=0.0001, kernel=rbf;, score=0.646 total time=   0.0s
[CV 5/5] END ..C=0.01, gamma=0.0001, kernel=rbf;, score=0.646 total time=   0.0s
[CV 1/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.863 total time=   0.0s
[CV 2/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.812 total time=   0.0s
[CV 3/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.838 total time=   0.0s
[CV 4/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.911 total time=   0.0s
[CV 5/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.937 total time=   0.0s
[CV 1/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.938 total time=   0.0s
[CV 2/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.938 total time=   0.0s
[CV 3/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.938 total time=   0.0s
[CV 4/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.949 total time=   0.0s
[CV 5/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.937 total time=   0.0s
[CV 1/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.925 total time=   0.0s
[CV 2/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.912 total time=   0.0s
[CV 3/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.938 total time=   0.0s
[CV 4/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.911 total time=   0.0s
[CV 5/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.911 total time=   0.0s
[CV 1/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 2/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.662 total time=   0.0s
[CV 3/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.650 total time=   0.0s
[CV 4/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 5/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.658 total time=   0.0s
[CV 1/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 2/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.646 total time=   0.0s
[CV 5/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.646 total time=   0.0s
[CV 1/5] END …C=1, gamma=1, kernel=rbf;, score=0.938 total time=   0.0s
[CV 2/5] END …C=1, gamma=1, kernel=rbf;, score=0.887 total time=   0.0s
[CV 3/5] END …C=1, gamma=1, kernel=rbf;, score=0.912 total time=   0.0s
[CV 4/5] END …C=1, gamma=1, kernel=rbf;, score=0.949 total time=   0.0s
[CV 5/5] END …C=1, gamma=1, kernel=rbf;, score=0.911 total time=   0.0s
[CV 1/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.938 total time=   0.0s
[CV 2/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.938 total time=   0.0s
[CV 3/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.938 total time=   0.0s
[CV 4/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.962 total time=   0.0s
```

```
[CV 5/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.949 total time=   0.0s
[CV 1/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.938 total time=   0.0s
[CV 2/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.938 total time=   0.0s
[CV 3/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.938 total time=   0.0s
[CV 4/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.937 total time=   0.0s
[CV 5/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.937 total time=   0.0s
[CV 1/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.925 total time=   0.0s
[CV 2/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.912 total time=   0.0s
[CV 3/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.938 total time=   0.0s
[CV 4/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.911 total time=   0.0s
[CV 5/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.899 total time=   0.0s
[CV 1/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.650 total time=   0.0s
[CV 2/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.662 total time=   0.0s
[CV 3/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.662 total time=   0.0s
[CV 4/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.671 total time=   0.0s
[CV 5/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.684 total time=   0.0s
[CV 1/5] END …C=10, gamma=1, kernel=rbf;, score=0.912 total time=   0.0s
[CV 2/5] END …C=10, gamma=1, kernel=rbf;, score=0.912 total time=   0.0s
[CV 3/5] END …C=10, gamma=1, kernel=rbf;, score=0.912 total time=   0.0s
[CV 4/5] END …C=10, gamma=1, kernel=rbf;, score=0.962 total time=   0.0s
[CV 5/5] END …C=10, gamma=1, kernel=rbf;, score=0.899 total time=   0.0s
[CV 1/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.963 total time=   0.0s
[CV 2/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.963 total time=   0.0s
[CV 3/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.938 total time=   0.0s
[CV 4/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.975 total time=   0.0s
[CV 5/5] END …C=10, gamma=0.1, kernel=rbf;, score=0.949 total time=   0.0s
[CV 1/5] END …C=10, gamma=0.01, kernel=rbf;, score=0.938 total time=   0.0s
[CV 2/5] END …C=10, gamma=0.01, kernel=rbf;, score=0.950 total time=   0.0s
[CV 3/5] END …C=10, gamma=0.01, kernel=rbf;, score=0.938 total time=   0.0s
[CV 4/5] END …C=10, gamma=0.01, kernel=rbf;, score=0.975 total time=   0.0s
[CV 5/5] END …C=10, gamma=0.01, kernel=rbf;, score=0.911 total time=   0.0s
[CV 1/5] END …C=10, gamma=0.001, kernel=rbf;, score=0.950 total time=   0.0s
[CV 2/5] END …C=10, gamma=0.001, kernel=rbf;, score=0.938 total time=   0.0s
[CV 3/5] END …C=10, gamma=0.001, kernel=rbf;, score=0.938 total time=   0.0s
[CV 4/5] END …C=10, gamma=0.001, kernel=rbf;, score=0.937 total time=   0.0s
[CV 5/5] END …C=10, gamma=0.001, kernel=rbf;, score=0.924 total time=   0.0s
[CV 1/5] END …C=10, gamma=0.0001, kernel=rbf;, score=0.925 total time=   0.0s
[CV 2/5] END …C=10, gamma=0.0001, kernel=rbf;, score=0.912 total time=   0.0s
[CV 3/5] END …C=10, gamma=0.0001, kernel=rbf;, score=0.938 total time=   0.0s
[CV 4/5] END …C=10, gamma=0.0001, kernel=rbf;, score=0.911 total time=   0.0s
[CV 5/5] END …C=10, gamma=0.0001, kernel=rbf;, score=0.899 total time=   0.0s
[CV 1/5] END …C=100, gamma=1, kernel=rbf;, score=0.912 total time=   0.0s
[CV 2/5] END …C=100, gamma=1, kernel=rbf;, score=0.900 total time=   0.0s
[CV 3/5] END …C=100, gamma=1, kernel=rbf;, score=0.912 total time=   0.0s
[CV 4/5] END …C=100, gamma=1, kernel=rbf;, score=0.949 total time=   0.0s
[CV 5/5] END …C=100, gamma=1, kernel=rbf;, score=0.899 total time=   0.0s
[CV 1/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.963 total time=   0.0s
[CV 2/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.950 total time=   0.0s
```

```
[CV 3/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.950 total time=   0.0s
[CV 4/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.975 total time=   0.0s
[CV 5/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.962 total time=   0.0s
[CV 1/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.950 total time=   0.0s
[CV 2/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.963 total time=   0.0s
[CV 3/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.950 total time=   0.0s
[CV 4/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.975 total time=   0.0s
[CV 5/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.949 total time=   0.0s
[CV 1/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.963 total time=   0.0s
[CV 2/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.950 total time=   0.0s
[CV 3/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.938 total time=   0.0s
[CV 4/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.975 total time=   0.0s
[CV 5/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.924 total time=   0.0s
[CV 1/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.950 total time=   0.0s
[CV 2/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.938 total time=   0.0s
[CV 3/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.938 total time=   0.0s
[CV 4/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.949 total time=   0.0s
[CV 5/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.924 total time=   0.0s
Accuracy of best SVM classfier = 0.96
Best found hyperparameters of SVM classifier = {'C': 100, 'gamma': 0.1,
'kernel': 'rbf'}
```

As per the results we obtained an accuracy of Accuracy of best SVM classfier = 96%

8 KNN

KNN algorithm is one of the simplest supervised machine learning algorithm. KNN identifies likeness of the available data and when a new data is identified, KNN will compare the similarity and accomodate the new data,

We need to choose the ”K” number of neighbours

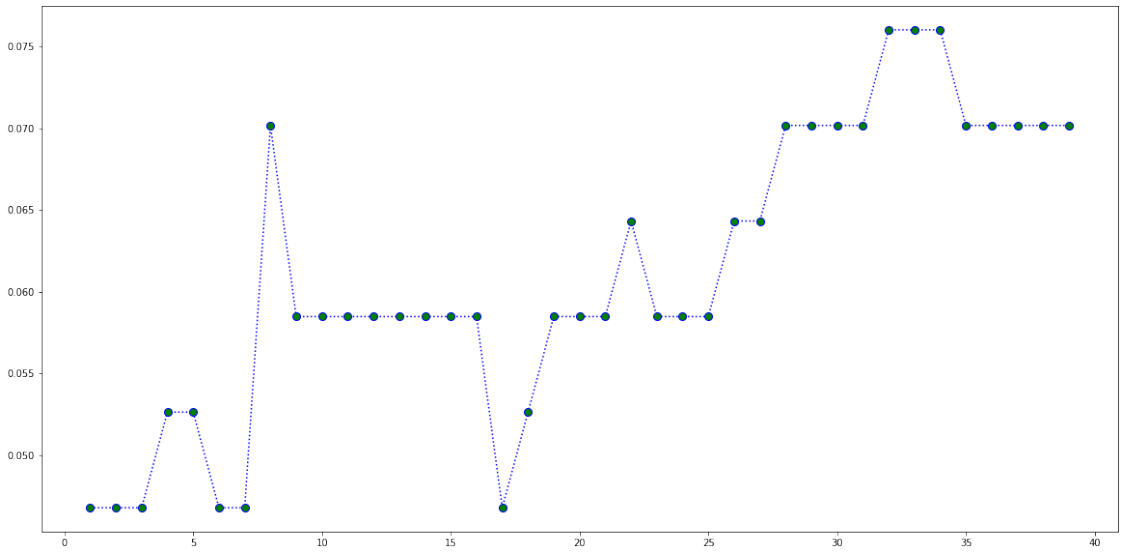
calculate euclidean distance

count the datapoints in each category

The new data is assigned to the category that has maximum count value

[25]:
```
#choose the "K" number of neighbours

array = [] #Let's create an array for passing the error values

#create an object with the number of neighbours "j" and fit the model
for j in range(1, 40):
    model = KNeighborsClassifier(n_neighbors=j)
    model.fit(X_train, Y_train)

#Generate prediction
    prediction2 = model.predict(X_test)
```

```
    #prediction2 !=Y_test will return False = 0 or True = 1. Take the avergae␣
 ↪of the results and append it to the error rate array
    array.append(np.mean(prediction2 != Y_test))


#Let's plot a graph between error rate(array) vs k value (1-40) for getting␣
 ↪value of K with minimum error rates

plt.figure(figsize=(20, 10))  # size of the figure

#plotting the values
plt.plot(range(1, 40), array, color="blue", linestyle="dotted",␣
 ↪marker="o",markerfacecolor="green", markersize=8)

plt.show()
```



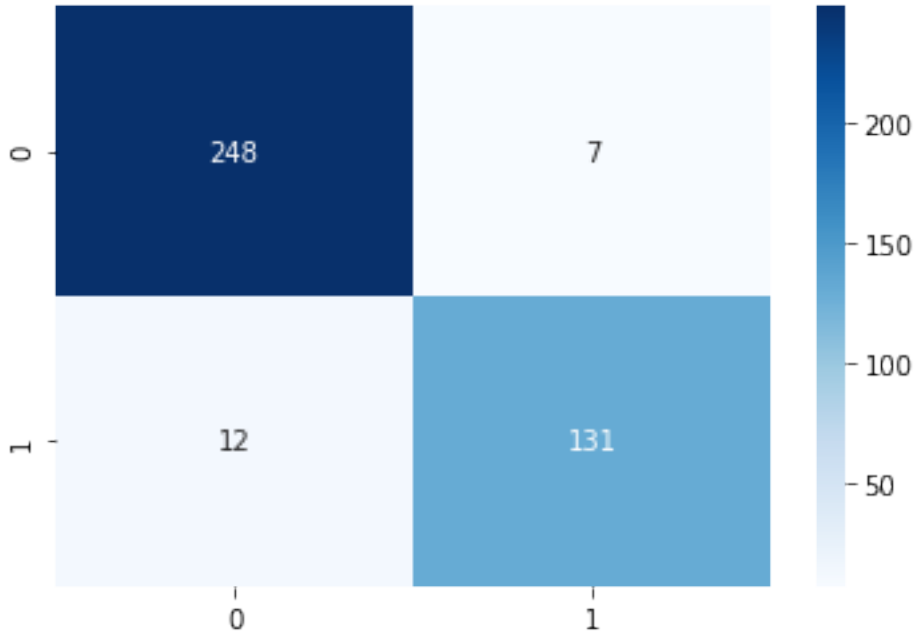
From the above plot we can choose the value of ”K” with minimum error rate.

[26]:
```
#Create and train the model
model3 = KNeighborsClassifier(n_neighbors=5, p=2, weights="uniform",
                              algorithm="auto", leaf_size=30)
model3.fit(X_train, Y_train)

#Create prediction
prediction3 = model3.predict(X_train)

#Create classification report and confusion matrix
print(classification_report(Y_train,prediction3))
print(confusion_matrix(Y_train,prediction3))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       255
           1       0.95      0.92      0.93       143

    accuracy                           0.95       398
   macro avg       0.95      0.94      0.95       398
weighted avg       0.95      0.95      0.95       398

[[248   7]
 [ 12 131]]
```

[28]:
```
sns.heatmap(confusion_matrix(Y_train, prediction3),
            annot=True, cmap="Blues", fmt="g")
```

[28]: <AxesSubplot:>



As per the above report generated, this model has a precision value of 95% for predicting Bening and 95% for predicting Malignant. From the above confusion matrix results, our model is giving 98% "True Negative" results, 0% "False positive," 13% False-negative results, and 60% True Positive results.

True Negative means the actual test result is "B," and the predicted result is "B."

False Positive means the actual test result is "B," and the predicted result is "M."

False-negative means the actual test result is "M," and the predicted result is "B."

True Positive means the actual test result is ”M,” and the predicted result is ”M.”

9 KNN WITH HYPER PARAMETER TUNING

[37]:
```
#define parameters
parameters_grid = {
    "n_neighbors": [1, 5, 10, 15, 20],
    "metric": ["minkowski", "euclidean", "manhattan"]
}

#define grid search for training model
model_3 = sklearn.model_selection.GridSearchCV(sklearn.neighbors.
 ↪KNeighborsClassifier(),
                                                parameters_grid,
                                                scoring="accuracy",
                                                cv=5, n_jobs=-1)

#fit model to data
model_3.fit(X_train, Y_train)

#generate accuracy
print("Accuracy of best KNN classfier= {:.2f}".format(model_3.best_score_))
print("Best found hyperparameters of KNN classifier= {}"
      .format(model_3.best_params_))
```

```
Accuracy of best KNN classfier= 0.94
Best found hyperparameters of KNN classifier= {'metric': 'manhattan',
'n_neighbors': 10}
```

10 DECISION TREE CLASSIFIER

[38]:
```
#create the object model
model4 = DecisionTreeClassifier(random_state=42)

#Fit model to data
model4.fit(X_train, Y_train)

#Generate predictions
Tree_pred = model4.predict(X_test)

#Generate classificaion report
print(classification_report(Y_test, Tree_pred))
```

```
              precision    recall  f1-score   support
```

```
              0       0.92      0.95      0.93       102
              1       0.92      0.87      0.90        69

       accuracy                           0.92       171
      macro avg       0.92      0.91      0.91       171
   weighted avg       0.92      0.92      0.92       171
```

Let's call gridsearch method for improvement in accuracy for the decision tree classifier

[39]:
```
#define parameters
parameters_grid = {"max_depth":range(1, model4.tree_.max_depth+1, 2),
                   "max_features": range(1, len(model4.feature_importances_)+1)}

#define grid search for training model
model5 = GridSearchCV(DecisionTreeClassifier(random_state=42),
                      param_grid=parameters_grid,
                      scoring='accuracy',
                      n_jobs=-1)

model5.fit(X_train, Y_train)

#generate accuracy
print("Accuracy of best Decision Tree classifier= {:.2f}".format(model5.
 ↪best_score_))
print("Best found hyperparameters of Decision Tree classifier= {}"
      .format(model5.best_params_))
```

```
Accuracy of best Decision Tree classifier= 0.95
Best found hyperparameters of Decision Tree classifier= {'max_depth': 3,
'max_features': 4}
```

11 TESTING THE BEST MODEL

[40]:
```
Final_prediction = model2.predict(X_test)
accuracy = sklearn.metrics.accuracy_score(Y_test, Final_prediction)
confusion_matrix = sklearn.metrics.confusion_matrix(Y_test, Final_prediction)
precision, recall, f1, support = sklearn.metrics.
 ↪precision_recall_fscore_support(Y_test, Final_prediction)

print("Accuracy =", accuracy)
print("Precision =", precision)
print("Recall =", recall)
print("F1-Score =", f1)
print("Confusion Matrix:\n", confusion_matrix)
```

```
Accuracy = 0.9590643274853801
```

```
Precision = [0.97979798 0.93055556]
Recall = [0.95098039 0.97101449]
F1-Score = [0.96517413 0.95035461]
Confusion Matrix:
 [[97  5]
 [ 2 67]]
```

We tested our model with Decision-Tree algorithm and we have obtained an accuracy of 96%. Our model has achieved a precision value of 97% for predicting the case of "B" and 93% for predicting the case of "M".

12 CONCLUSION

We were dealing with a medical condition called "breast cancer" which is widely spreading across the world among women. Early stage detection of this disease can prevent the death rates. We have chosen "Decision Tree" as the best model as it is giving an accuracy of 96%. It is very important for the model to reduce the rate of producing "False negative" and "False positive" results.

Our model is giving 3% "False positive," 6% False-negative which is fair enough. Our model has achieved a precision value of 97% for predicting the case of "B" and 93% for predicting the case of "M".

For the better preformance of the model, we need to train the model with a dataset that contains the first stage and second stage feature's of the Malignant condition. In this condition model would be able to detect the breast cancer at early stage.

13 References:

Source: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

Data is also available in UCI machine learning repository

Sckikit learn official website: https://scikit-learn.org/stable/index.html?msclkid=3fb899c9a6db11eca99e835b88c35

Breast Cancer Detection and Prediction using Machine Learning
https://www.researchgate.net/publication/342303246_Breast_Cancer_Detection_and_Prediction_using_Mach

[ ]: