DATA SCIENCE - SVM CLASSIFIER

M.Nivethithaa

Breast Cancer SVM Classification

Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

Displaying Dataset

In [2]:
```
df = pd.read_csv("data.csv")
df.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	842302	M	17.99	10.38	122.80	1001.0	0.1184
1	842517	M	20.57	17.77	132.90	1326.0	0.0847
2	84300903	M	19.69	21.25	130.00	1203.0	0.1096
3	84348301	M	11.42	20.38	77.58	386.1	0.1425
4	84358402	M	20.29	14.34	135.10	1297.0	0.1003

5 rows × 33 columns

Columns present in the dataset

In [3]:
```
df.columns
```

Out[3]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

Shape of the dataset

In [4]:
```
df.shape
```

Out[4]:
```
(569, 33)
```

Exploratory Data Analysis

Summary of the dataset

In [5]:
```
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
```

In [6]:
```
df.describe()
```

Out[6]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400

8 rows × 32 columns

Checking for missing values

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
id                           0
diagnosis                    0
radius_mean                  0
texture_mean                 0
perimeter_mean               0
area_mean                    0
smoothness_mean              0
compactness_mean             0
concavity_mean               0
concave points_mean          0
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

Cleaning Missing Values

In [8]:
```
df= df.drop(['Unnamed: 32','id'],axis=1)
df
```

Out[8]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
0	M	17.99	10.38	122.80	1001.0	0.11840	
1	M	20.57	17.77	132.90	1326.0	0.08474	
2	M	19.69	21.25	130.00	1203.0	0.10960	
3	M	11.42	20.38	77.58	386.1	0.14250	
4	M	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	
564	M	21.56	22.39	142.00	1479.0	0.11100	
565	M	20.13	28.25	131.20	1261.0	0.09780	
566	M	16.60	28.08	108.30	858.1	0.08455	
567	M	20.60	29.33	140.10	1265.0	0.11780	
568	B	7.76	24.54	47.92	181.0	0.05263	

569 rows × 31 columns

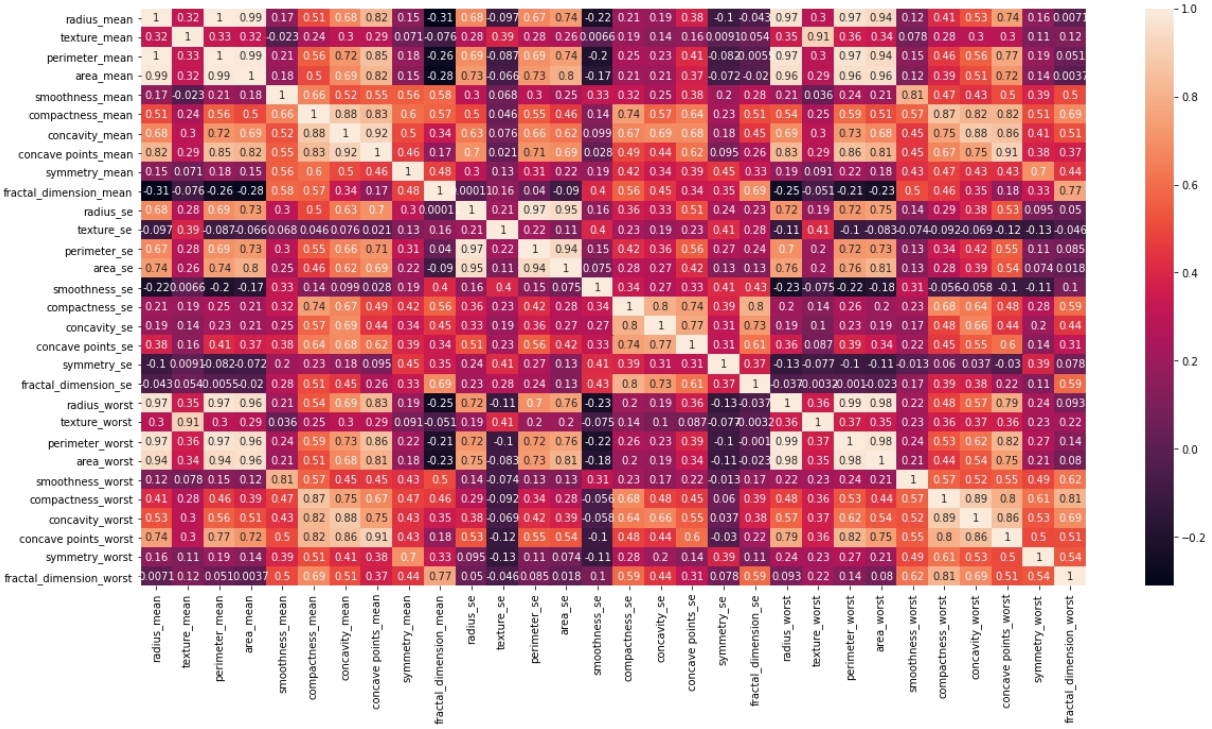
Detect prescence(Malignant-M) or absence(Benign-B) of cancer cells

In [9]:
```
df['diagnosis'].value_counts()
```

Out[9]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

Checking for the correlation

In [10]:
```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(),annot=True)
plt.ioff()
```

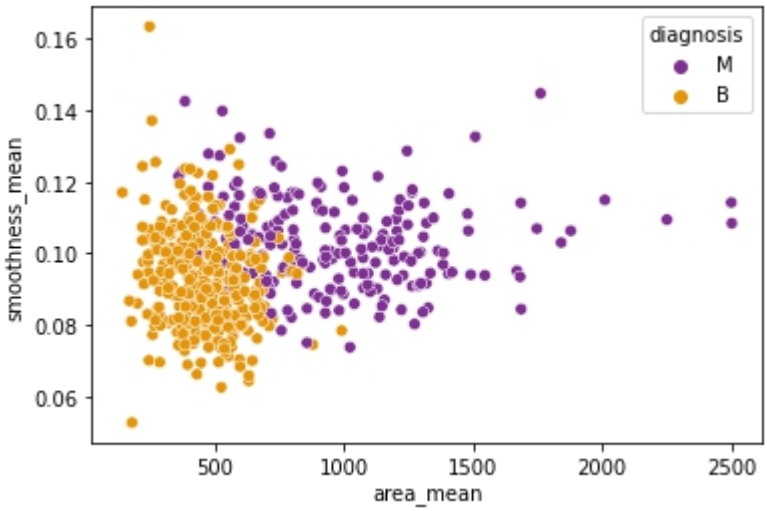


Data Visualization

ScatterPlot

In [11]:
```
sns.scatterplot(x= 'area_mean', y= 'smoothness_mean', hue= 'diagnosis', data=df,
```

Out[11]:
```
<AxesSubplot:xlabel='area_mean', ylabel='smoothness_mean'>
```

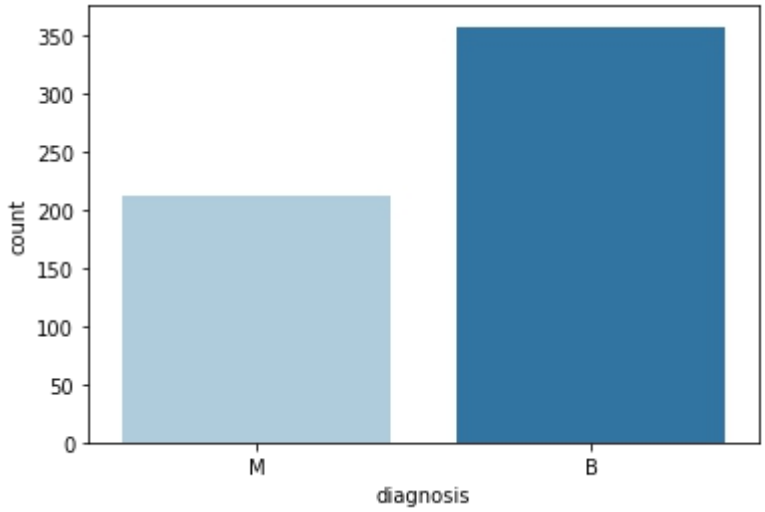


Count Plot

In [12]:
```
sns.countplot(df['diagnosis'],palette='Paired')
```

```
c:\users\nivethitha\appdata\local\programs\python\python39\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

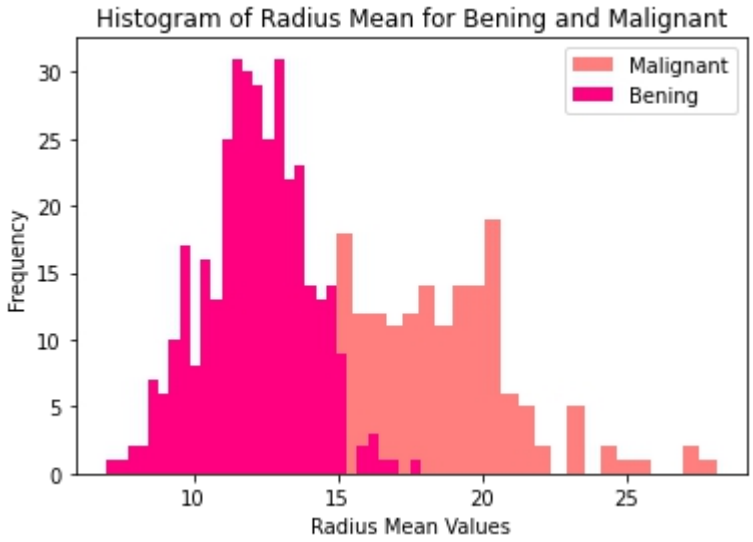
Out[12]:
```
<AxesSubplot:xlabel='diagnosis', ylabel='count'>
```



Histogram

Histogram of Radius Mean for Bening and Malignant

In [13]:
```
m = plt.hist(df[df["diagnosis"] == "M"].radius_mean,bins=30,fc = (1,0,0,0.5),lab
b = plt.hist(df[df["diagnosis"] == "B"].radius_mean,bins=30, fc = (1,0,0.5), lab

plt.legend()
plt.xlabel ("Radius Mean Values")
plt.ylabel ("Frequency")
plt.title("Histogram of Radius Mean for Bening and Malignant")
plt.show()
```



Encoding categorical data

Encoding malignan as 1 and benign as 0

In [14]:
```
LEncoder = LabelEncoder()

df['diagnosis'] = LEncoder.fit_transform(df['diagnosis'])
df
```

Out[14]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
0	1	17.99	10.38	122.80	1001.0	0.11840	
1	1	20.57	17.77	132.90	1326.0	0.08474	
2	1	19.69	21.25	130.00	1203.0	0.10960	
3	1	11.42	20.38	77.58	386.1	0.14250	
4	1	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	...
564	1	21.56	22.39	142.00	1479.0	0.11100	
565	1	20.13	28.25	131.20	1261.0	0.09780	
566	1	16.60	28.08	108.30	858.1	0.08455	
567	1	20.60	29.33	140.10	1265.0	0.11780	
568	0	7.76	24.54	47.92	181.0	0.05263	

569 rows × 31 columns

Pre-Modeling Tasks

Separating the independant and the dependant variable

In [15]:
```
X = df.drop('diagnosis',axis=1).values
y = df['diagnosis'].values
```

Splitting the dataset

In [16]:
```
random_state = 42

x_train, x_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_s
```

Feature Scaling

In [17]:
```
sc = StandardScaler()

X_train = sc.fit_transform(x_train)
X_test= sc.transform(x_test)
```

Modeling

In [18]:
```
svc = SVC(probability=True)

svc.fit(X_train,y_train)

y_pred_svc = svc.predict(X_test)
```

In [19]:
```
X_train.shape, y_train.shape,X_test.shape, y_test.shape
```

Out[19]:
```
((455, 30), (455,), (114, 30), (114,))
```

Evaluating the model

Confusion Matrix

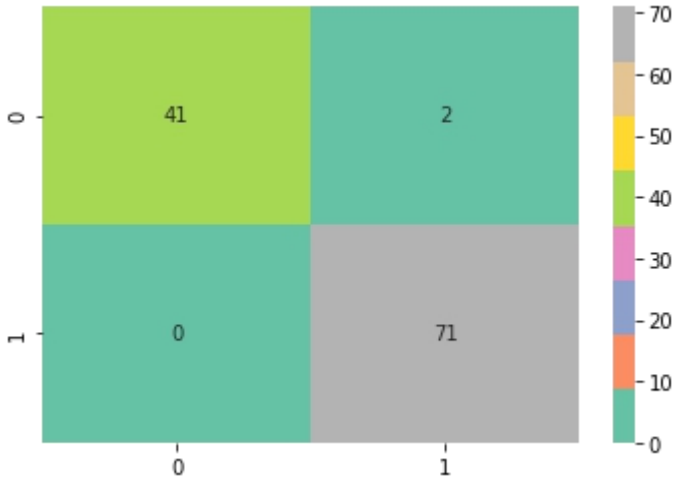
In [20]:
```
cm = np.array(confusion_matrix(y_test, y_pred_svc, labels=[1,0]))

confusion_mat= pd.DataFrame(cm, index = ['cancer', 'healthy'],
                            columns =['predicted_cancer','predicted_healthy'])

confusion_mat
```

Out[20]:

	predicted_cancer	predicted_healthy
cancer	41	2
healthy	0	71

In [21]:
```
sns.heatmap(cm,annot=True,fmt='g',cmap='Set2')
```

Out[21]: <AxesSubplot:>



True Positive(TP) : Model predicted as Healthy, and is actually Healthy
True Negative(TN) : Model predicted as No Cancer, and is actually No Cancer
False Positive(FP): Model predicted as Healthy, but actually No Cancer
False Negative(FN): Model predicted as No Cancer, but actually Healthy

Accuracy_Score

In [22]:
```
print(accuracy_score(y_test, y_pred_svc))  #(TP+TN)/total
```

```
0.9824561403508771
```

Classification Report

In [23]:
```
print(classification_report(y_test, y_pred_svc))
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        71
           1       1.00      0.95      0.98        43

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

Result

True Positive(TP) : 71

True Negative(TN) : 41

False Positive(FP): 2

False Negative(FN): 0

How often Model predicted as Healthy, and is actually Healthy?

True Positive Rate = TP/TP+FP = 71/(71+2) = 0.97

How often the model predicts Healthy when it's actually No Cancer?

False Positive Rate = FP/FP+TN = 2/2+41 = 0.04

In [ ]:
```
```