Heart Disease Patient Preidiction

In this project we will classify patients of heart disease. We are using **Statlog heart data set** for this project. In this project we will apply different Machine Learning Algorithms to analyze the dataset. we will be apply following Machine Learning Algorithms:

- **Unsupversied Learning Algorithm**: Gaussian Mixture
- **Unsupversied Learning Algorithm**: K-Mean Clustering
- **Supervised Learning Algorithm**: Decision Tree
- **Supervised Learning Algorithm**: Suppor Vector Machine Classifier

Importing Libraries ¶

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sb
from sklearn.neighbors import KNeighborsClassifier
from matplotlib.colors import ListedColormap
from sklearn.tree import DecisionTreeClassifier
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.model_selection import learning_curve
from sklearn.model_selection import validation_curve
from sklearn.model_selection import ShuffleSplit, train_test_split
from sklearn.metrics import plot_confusion_matrix, classification_report
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

Reading Dataset

By using pandas library we will read the dataset. This is a **Statlog Heat data set**. In this dataset we have 10 columns. In those 10 columns first 9 columns are features and the last one is class/target variable. In the dataset we have following features:

- **Age**: Age is a countinous variable.
- **Sex**: Sex is a binary viable (Male/Female).
- **Rest Blood Pressure**: Rest Blood Pressure is a continous variable.
- **Serum Cholestoral**: Serum Cholestoral is a continous variable.
- **Fasting Blood Sugar**: Fasting Blood Sugar is a binary variable.
- **Max Heart Rate**: Max Heart Rate is a continous variable.
- **Excercise Induced**: Excercise Induced is a binary variable.
- **Slope**: Slope is a continous variable.
- **Major Vessels**: Major Vessels is a continous variable.
- **Class**: This is target variable, if disease is **present then class label is 2**, if disease is **not present then class label is 1**

In [2]:

```
dataset = pd.read_csv("heart_data.csv")
dataset
```

Out[2]:

	Age	Sex	RestBloodPressure	SerumCholestoral	FastingBloodSugar	MaxHeartRate	Exerc
0	70	2	130	322	1	109	
1	67	1	115	564	1	160	
2	57	2	124	261	1	141	
3	64	2	128	263	1	105	
4	74	1	120	269	1	121	
...	...	...	...	...	...	...	
265	52	2	172	199	2	162	
266	44	2	120	263	1	173	
267	56	1	140	294	1	153	
268	57	2	140	192	1	148	
269	67	2	160	286	1	108	

270 rows × 10 columns

Spliting Features and Label

In [3]:

```
features = dataset.drop("Class", axis=1)
label = dataset["Class"]
```

Dataset Analysis

In this section we will do some analysis on our dataset, first of all we will do correlation analysis on this dataset and then we will plot the heatmap for the correlation analysis result. Then after that we will plot histograms for each feature variable, this will help us to learn about the features in proper details.

Correlation Analysis

For correlation analysis we will use pandas library, this libray provides a funcction **corr()**, this function helps us to calculate the corelation between each feature.

In [4]:

```
corr_analysis = features.corr()
corr_analysis
```

Out[4]:

	Age	Sex	RestBloodPressure	SerumCholestoral	FastingBloodSu
Age	1.000000	-0.094401	0.273053	0.220056	0.123
Sex	-0.094401	1.000000	-0.062693	-0.201647	0.042
RestBloodPressure	0.273053	-0.062693	1.000000	0.173019	0.155
SerumCholestoral	0.220056	-0.201647	0.173019	1.000000	0.025
FastingBloodSugar	0.123458	0.042140	0.155681	0.025186	1.000
MaxHeartRate	-0.402215	-0.076101	-0.039136	-0.018739	0.022
ExerciseInduced	0.098297	0.180022	0.082793	0.078243	-0.004
Slope	0.159774	0.050545	0.142472	-0.005755	0.044
MajorVessels	0.356081	0.086830	0.085697	0.126541	0.123

Visualizing Correlation Analysis

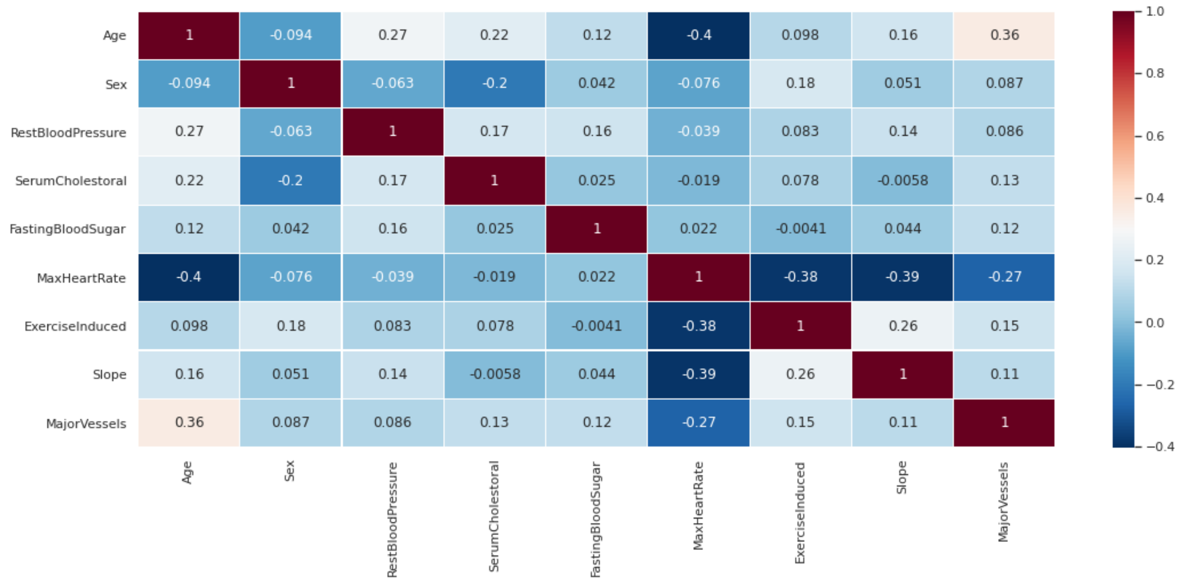
Now after calculating correlation between each feature, we will plot a heatmap to visualize the realtionship between each feature. For this purpose we will use **seaborn** library.

In [5]:

```
sb.set(rc={"figure.figsize":(18, 7)})
sb.heatmap(corr_analysis,
            xticklabels=corr_analysis.columns,
            yticklabels=corr_analysis.columns,
            cmap='RdBu_r',
            annot=True,
            linewidth=0.1)
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f046b66afd0>
```



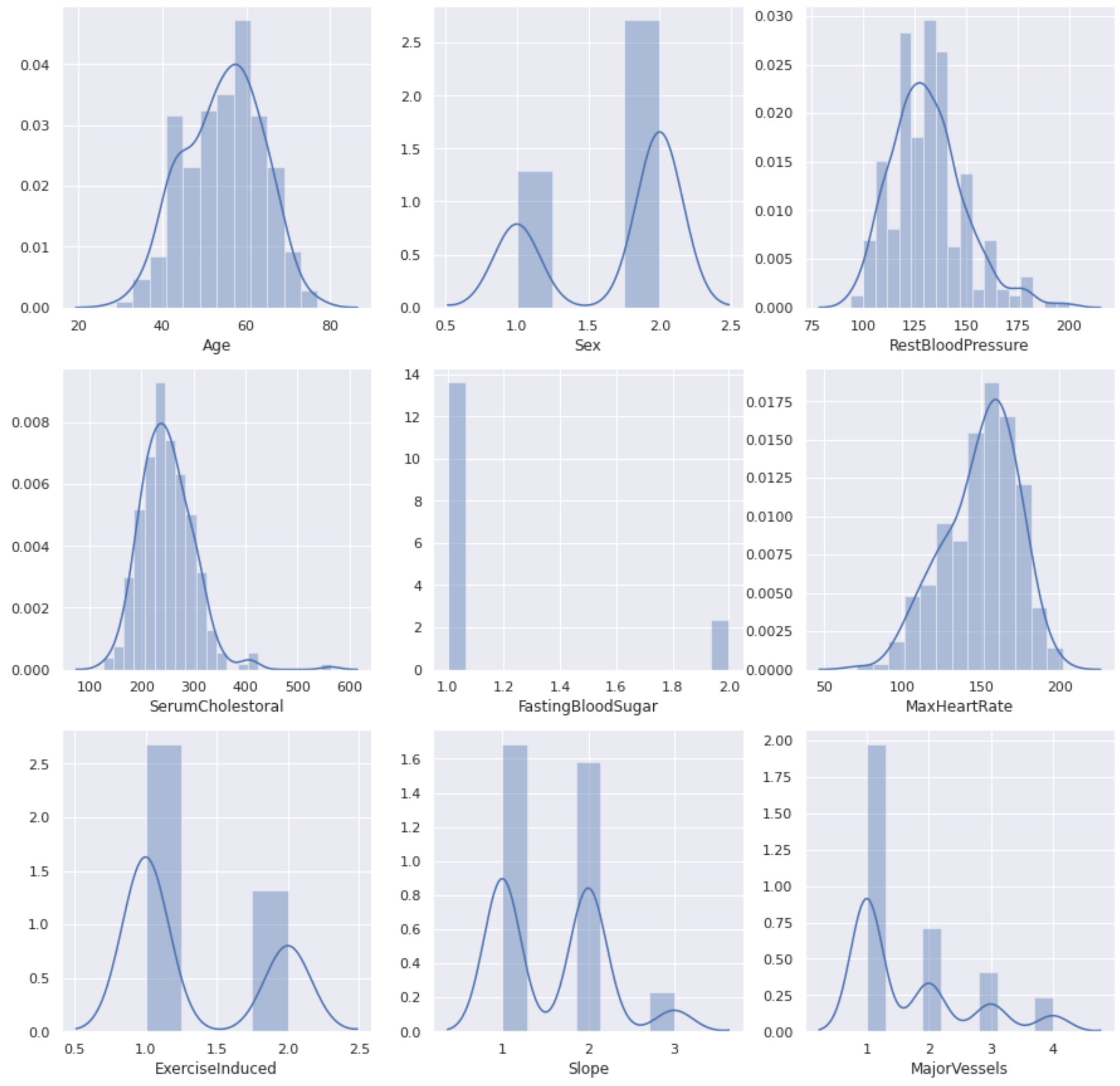
	Age	Sex	RestBloodPressure	SerumCholestoral	FastingBloodSugar	MaxHeartRate	ExerciseInduced	Slope	MajorVessels
Age	1	-0.094	0.27	0.22	0.12	-0.4	0.098	0.16	0.36
Sex	-0.094	1	-0.063	-0.2	0.042	-0.076	0.18	0.051	0.087
RestBloodPressure	0.27	-0.063	1	0.17	0.16	-0.039	0.083	0.14	0.086
SerumCholestoral	0.22	-0.2	0.17	1	0.025	-0.019	0.078	-0.0058	0.13
FastingBloodSugar	0.12	0.042	0.16	0.025	1	0.022	-0.0041	0.044	0.12
MaxHeartRate	-0.4	-0.076	-0.039	-0.019	0.022	1	-0.38	-0.39	-0.27
ExerciseInduced	0.098	0.18	0.083	0.078	-0.0041	-0.38	1	0.26	0.15
Slope	0.16	0.051	0.14	-0.0058	0.044	-0.39	0.26	1	0.11
MajorVessels	0.36	0.087	0.086	0.13	0.12	-0.27	0.15	0.11	1

Plotting Histogram

Now after correlation analysis, we will now plot histogram for each variable. Histograms represent the data distribution by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin. For plotting histograms we will use seaborn library, we will plot each histogram in a single figure.

In [6]:

```
fig, axes = plt.subplots(3, 3, figsize=(15, 15))

columns = features.columns

index = 0
for i in range(3):
    for j in range(3):
        sb.distplot(features[columns[index]], ax=axes[i, j])
        index = index +1
```



Applying Unsupervised Learning Method

Gaussian Mixture Model

Gaussian Mixture is an Unsupervised learning model. Gaussian Mixture Model is probabilistic model and it uses the soft clustering approach for distributing the points in different clusters.

First of all we will convert our whole dataset into numpy arrays, in this way they can be used in the gaussian mixture model.

In [7]:

```
X = np.array(features)
y = np.array(label)
```

Potting Dataset

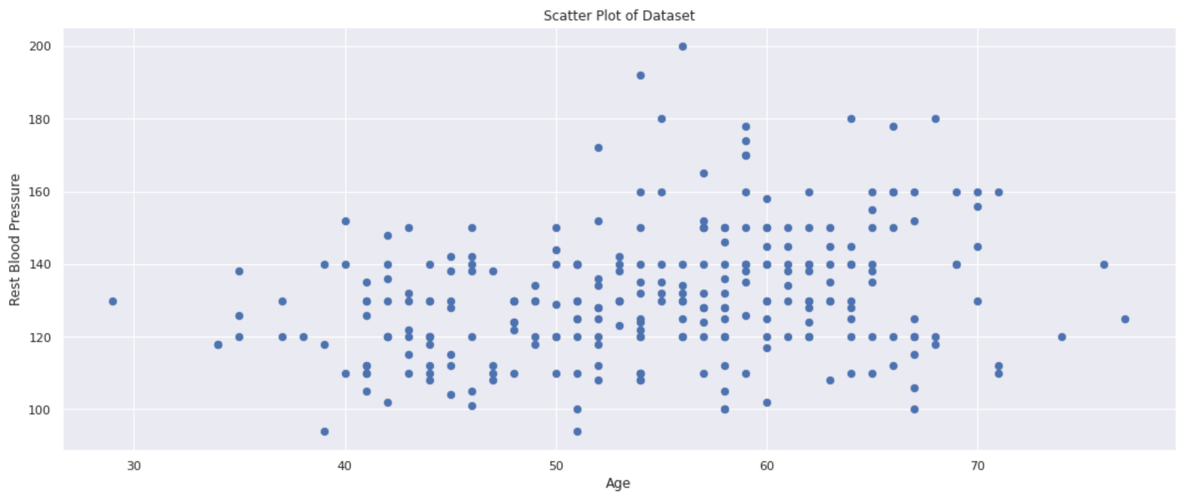
In the code section we will plot the dataset without the target variable becuase we are working with unsupervised learning model. Now for the x-aixs we will be using **Age** variable and for y-axis we will be using **Rest Blood Pressure** variable.

In [8]:

```
plt.scatter(X[:, 0], X[:, 2], s=40, cmap='viridis')
plt.xlabel("Age")
plt.ylabel("Rest Blood Pressure")
plt.title("Scatter Plot of Dataset")
```

Out[8]:

```
Text(0.5, 1.0, 'Scatter Plot of Dataset')
```



Implementing Gaussian Mixture Model

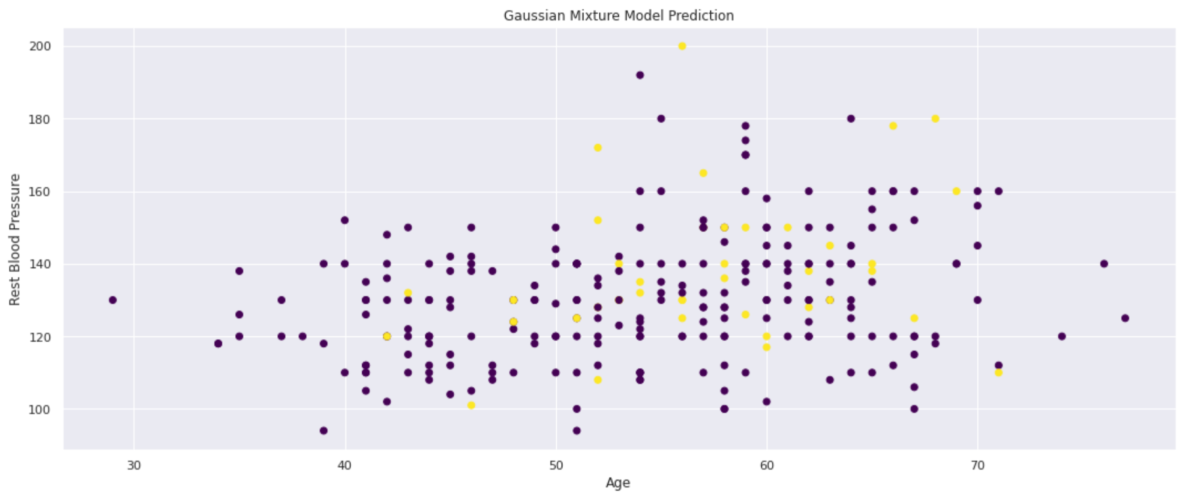
Now we will implement gaussian mixture model, in this model we will set the **Number of components** to **2** because we have to find 2 clusters of data, one for sick patients and the other one for healty patients.

In [9]:

```
gmm = GaussianMixture(n_components=2, max_iter=10000)
gmm.fit(X)

gmm_pred_labels = gmm.predict(X)

plt.scatter(X[:, 0], X[:, 2], c=gmm_pred_labels, cmap='viridis');
plt.title("Gaussian Mixture Model Prediction")
plt.xlabel("Age")
plt.ylabel("Rest Blood Pressure")
```

Out[9]:

```
Text(0, 0.5, 'Rest Blood Pressure')
```



As we can see in the upper graph the gaussian mixture model has successfully made 2 clusters of data one for sick pateints and the other one for healthy patients.

K Mean Clustering

k-means clustering is a distance-based algorithm. This means that it tries to group the closest points to form a cluster.

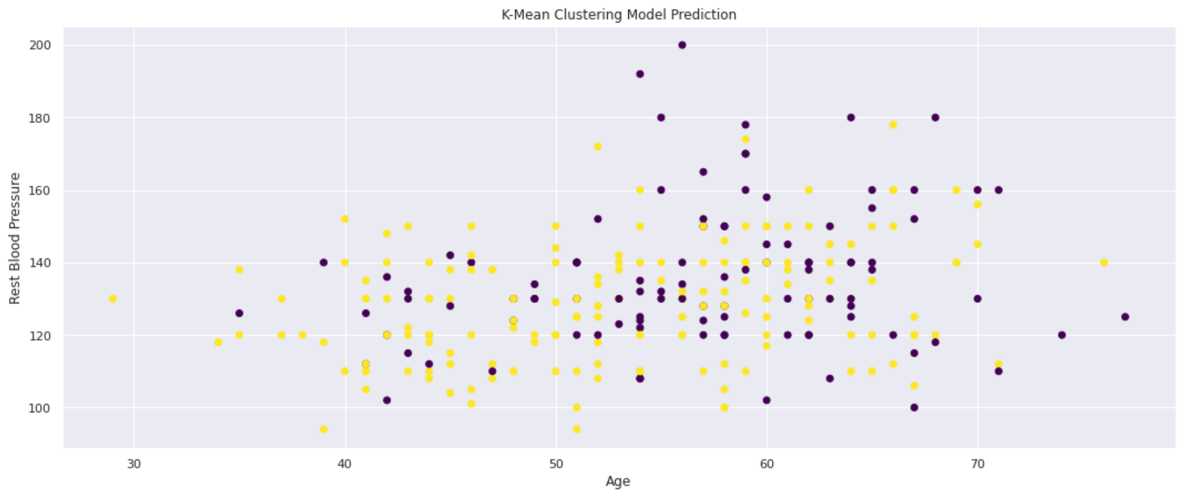
For this algorithm we will use the same numpy arrays of features and labels which we had used in Gaussian Mixture Model.

In [10]:

```
kmean_model = KMeans(n_clusters=2)
kmean_model.fit(X)

kmean_pred = kmean_model.predict(X)

plt.scatter(X[:, 0], X[:, 2], c=kmean_pred, cmap='viridis');
plt.title("K-Mean Clustering Model Prediction")
plt.xlabel("Age")
plt.ylabel("Rest Blood Pressure")
```

Out[10]:

```
Text(0, 0.5, 'Rest Blood Pressure')
```



Gaussian Mixture Predictions Sick patient Subset

So in this section we will see how many patients have been predicted as sick and how many are predicted as sick by both models. So now we will use **Class label 1** for sick patients and **Class label 0** for healthy patients

In [11]:

```
index_to_del = []
for index, pred_data in enumerate(gmm_pred_labels):
    if pred_data == 0:
        index_to_del.append(index)
index_to_del = list(reversed(index_to_del))

sick_patients = features

for index in index_to_del:
    sick_patients = sick_patients.drop(sick_patients.index[index])

print("The number of patients which have higher risk of heart disease are: ", sick_
```

```
The number of patients which have higher risk of heart disease are:  4
0
```

In [12]:

```
sick_patients[:10]
```

Out[12]:

	Age	Sex	RestBloodPressure	SerumCholestoral	FastingBloodSugar	MaxHeartRate	Exercis
6	56	2	130	256	2	142	
17	53	2	140	203	2	155	
24	54	1	132	288	2	159	
29	71	1	110	265	2	130	
35	48	2	130	256	2	150	
43	46	2	101	197	2	156	
44	59	2	126	218	2	134	
45	58	2	140	211	2	165	
51	52	2	128	205	2	184	
52	65	1	140	417	2	157	

K-Mean Clusterin Predictions Sick patient Subset

So in this section we will see how many patients have been predicted as sick and how many are predicted as sick by both models. So now we will use **Class label 1** for sick patients and **Class label 0** for healthy patients

In [13]:

```
index_to_del = []
for index, pred_data in enumerate(kmean_pred):
    if pred_data == 0:
        index_to_del.append(index)
index_to_del = list(reversed(index_to_del))

sick_patients = features

for index in index_to_del:
    sick_patients = sick_patients.drop(sick_patients.index[index])

print("The number of patients which have higher risk of heart disease are: ", sick_
```

```
The number of patients which have higher risk of heart disease are:  1
68
```

In [14]:

```
sick_patients[:10]
```

Out[14]:

	Age	Sex	RestBloodPressure	SerumCholestoral	FastingBloodSugar	MaxHeartRate	Exercis
5	65	2	120	177	1	140	
6	56	2	130	256	2	142	
7	59	2	110	239	1	142	
10	59	2	135	234	1	161	
11	53	2	142	226	1	111	
12	44	2	140	235	1	180	
13	61	2	134	234	1	145	
15	71	1	112	149	1	125	
17	53	2	140	203	2	155	
18	64	2	110	211	1	144	

Supervised Learning Models

Now in this section we will implement supervised Learning models to detect sick and healthy patients. For this purpose we will use **Decision Tree Classifier** and **Support Vector Machine Classifier**. But before going straight to the implemention of models we will first normalize the countinous features in our dataset, this will help the model to learn in a better way.

Normalizing Continous Features

So now we weill normalize the features, we will simply take the maximum number from each column and then we will divide each value of the column by its maximum value to get the normalized features

In [15]:

```
for column in features.columns:
    if column not in ["Sex", "FastingBloodSugar", "ExerciseInduced"]:
        max_value = np.max(features[column])
        features[column] = features[column]/max_value
```

In [16]:

```
features
```

Out[16]:

	Age	Sex	RestBloodPressure	SerumCholestoral	FastingBloodSugar	MaxHeartRate
0	0.909091	2	0.650	0.570922	1	0.539604
1	0.870130	1	0.575	1.000000	1	0.792079
2	0.740260	2	0.620	0.462766	1	0.698020
3	0.831169	2	0.640	0.466312	1	0.519802
4	0.961039	1	0.600	0.476950	1	0.599010
...	...	...	...	...	...	...
265	0.675325	2	0.860	0.352837	2	0.801980
266	0.571429	2	0.600	0.466312	1	0.856436
267	0.727273	1	0.700	0.521277	1	0.757426
268	0.740260	2	0.700	0.340426	1	0.732673
269	0.870130	2	0.800	0.507092	1	0.534653

270 rows × 9 columns

Model Learning Curves And Model Complexity graph

Now in this section we will implement Model Learning curves and we will also implement Model Complexity graph. These two techniques will help us to prevent overfitting and by the help of these techniques we will get the best parameters for our models.

Decision Tree Learning Curves

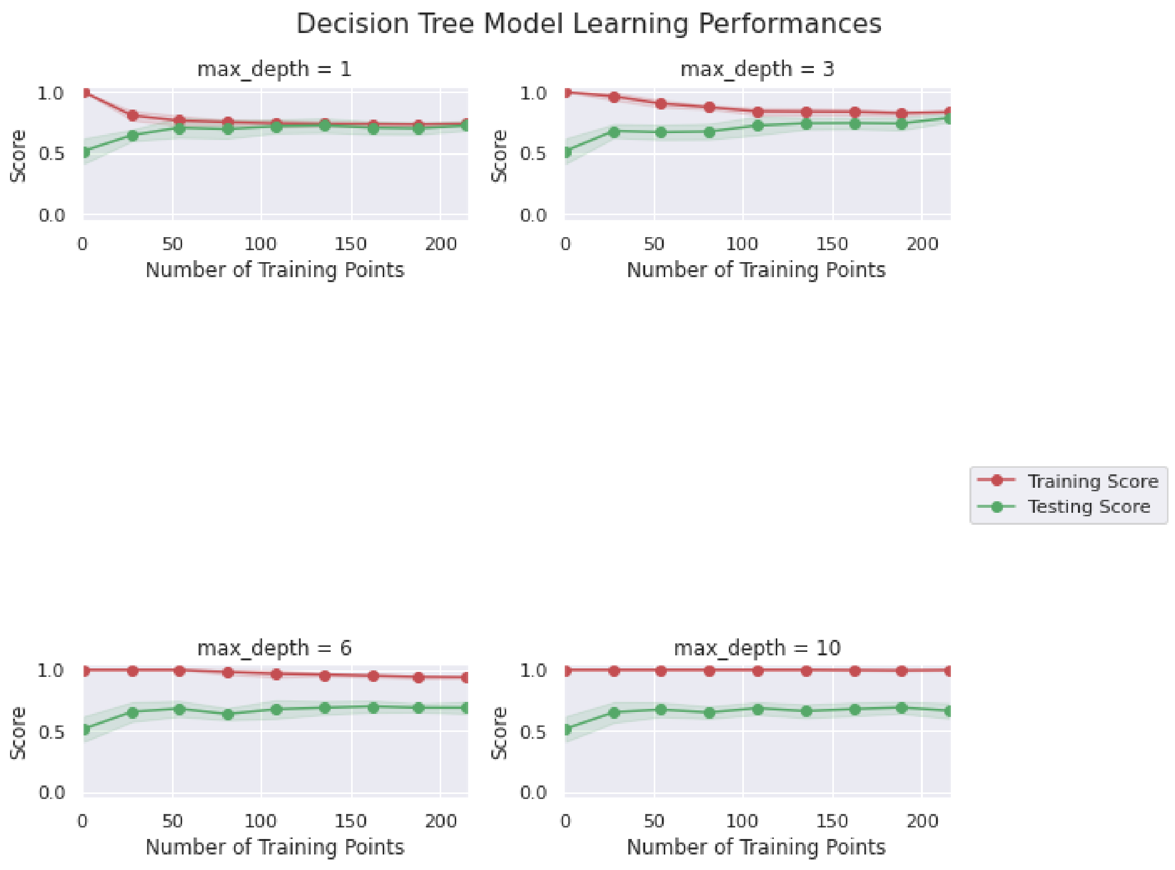
Now first of all we will implement Learning Curves for our Decision tree model, for this purpose we will input different **maximum depth** values and we will see which will perfrom better

In [17]:

```
def ModelLearning(X, y, max_depths):
    # Create 10 cross-validation sets for training and testing
    cv = ShuffleSplit(n_splits = 10, test_size = 0.2, random_state = 0)

    # Generate the training set sizes increasing by 50
    train_sizes = np.rint(np.linspace(1, X.shape[0]*0.8 - 1, 9)).astype(int)

    # Create the figure window
    fig = plt.figure(figsize=(10,7))

    # Create different models based on max_depth
    for k, depth in enumerate(max_depths):

        # Create a Decision tree Classifier at max_depth = depth
        classifier = DecisionTreeClassifier(max_depth = depth)

        # Calculate the training and testing scores
        sizes, train_scores, test_scores = learning_curve(classifier, X, y, cv = cv

        # Find the mean and standard deviation for smoothing
        train_std = np.std(train_scores, axis = 1)
        train_mean = np.mean(train_scores, axis = 1)
        test_std = np.std(test_scores, axis = 1)
        test_mean = np.mean(test_scores, axis = 1)

        # Subplot the learning curve
        ax = fig.add_subplot(2, 2, k+1)
        ax.plot(sizes, train_mean, 'o-', color = 'r', label = 'Training Score')
        ax.plot(sizes, test_mean, 'o-', color = 'g', label = 'Testing Score')
        ax.fill_between(sizes, train_mean - train_std, \
            train_mean + train_std, alpha = 0.15, color = 'r')
        ax.fill_between(sizes, test_mean - test_std, \
            test_mean + test_std, alpha = 0.15, color = 'g')

        # Labels
        ax.set_title('max_depth = %s'%(depth))
        ax.set_xlabel('Number of Training Points')
        ax.set_ylabel('Score')
        ax.set_xlim([0, X.shape[0]*0.8])
        ax.set_ylim([-0.05, 1.05])

    # Visual aesthetics
    ax.legend(bbox_to_anchor=(1.05, 2.05), loc='lower left', borderaxespad = 0.)
    fig.suptitle('Decision Tree Model Learning Performances', fontsize = 16, y = 1.
    fig.tight_layout()
    fig.show()


def ModelComplexity(X, y):
    # Create 10 cross-validation sets for training and testing
    cv = ShuffleSplit(n_splits = 10, test_size = 0.2, random_state = 0)

    # Vary the max_depth parameter from 1 to 10
    max_depth = np.arange(1,11)

    # Calculate the training and testing scores
    train_scores, test_scores = validation_curve(DecisionTreeClassifier(), X, y,
        param_name = "max_depth", param_range = max_depth, cv = cv)
    # Find the mean and standard deviation for smoothing
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    # Plot the validation curve
    plt.figure(figsize=(7, 5))
    plt.title('Decision Tree Model Complexity Performance')
    plt.plot(max_depth, train_mean, 'o-', color = 'r', label = 'Training Score')
    plt.plot(max_depth, test_mean, 'o-', color = 'g', label = 'Validation Score')
    plt.fill_between(max_depth, train_mean - train_std, \
        train_mean + train_std, alpha = 0.15, color = 'r')
    plt.fill_between(max_depth, test_mean - test_std, \
        test_mean + test_std, alpha = 0.15, color = 'g')

    # Visual aesthetics
    plt.legend(loc = 'lower right')
    plt.xlabel('Maximum Depth')
    plt.ylabel('Score')
    plt.ylim([-0.05,1.05])
    plt.show()
```

In [18]:

```
ModelLearning(features, label, [1, 3, 6, 10])
```

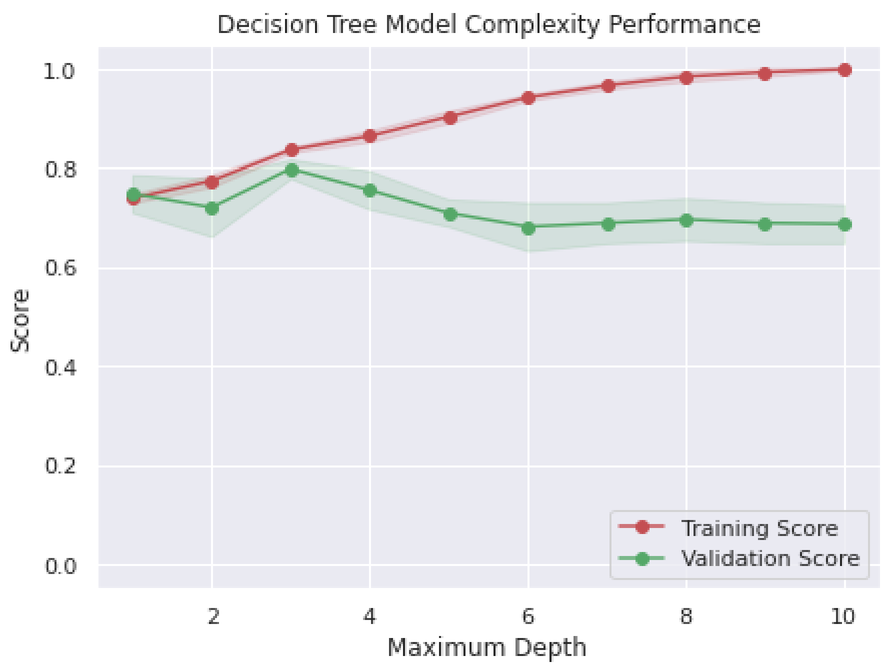


As we can see from learning curves that **maximum depth 3** is the optimal parameter for the decision tree model, if we use above depth than 3 it will overfit the model badly, and if we use depth lower than 3 then it will result in underfitting, so from learning curves we can say that we will chose **Maximum depth of 3**

Model Complexity Graph for Decision Tree

In [19]:

```
ModelComplexity(features, label)
```



Now again in the model complexity graph we can see that the best **maximum depth is 3** for our decision tree model, if we chose greater value than 3 it will overfit the dataset and if we chose lower value than 3 it will underfit the dataset, so we will chose **maximum depth 3**

Training Decision Tree Model

Now we will train our Decision tree model, we have seen which is the best parameter for Decision Tree model, so we will use that information.

First we will split the dataset into training and testing subsets and then we will train our model

In [20]:

```
X_train, X_test, y_train, y_test = train_test_split(features, label, test_size=0.2,
```

In [21]:

```
decision_tree_model = DecisionTreeClassifier(max_depth=3)
decision_tree_model.fit(X_train, y_train)
training_score = decision_tree_model.score(X_train, y_train)

print("Training Score of our Decision Tree model is: ", training_score)
```

```
Training Score of our Decision Tree model is:  0.8611111111111112
```

In [22]:

```
y_pred = decision_tree_model.predict(X_test)
testing_accuracy = accuracy_score(y_pred=y_pred, y_true=y_test)

print("Testing Accuracy for our Decision Tree model is: ", testing_accuracy)
```

```
Testing Accuracy for our Decision Tree model is:  0.7222222222222222
```

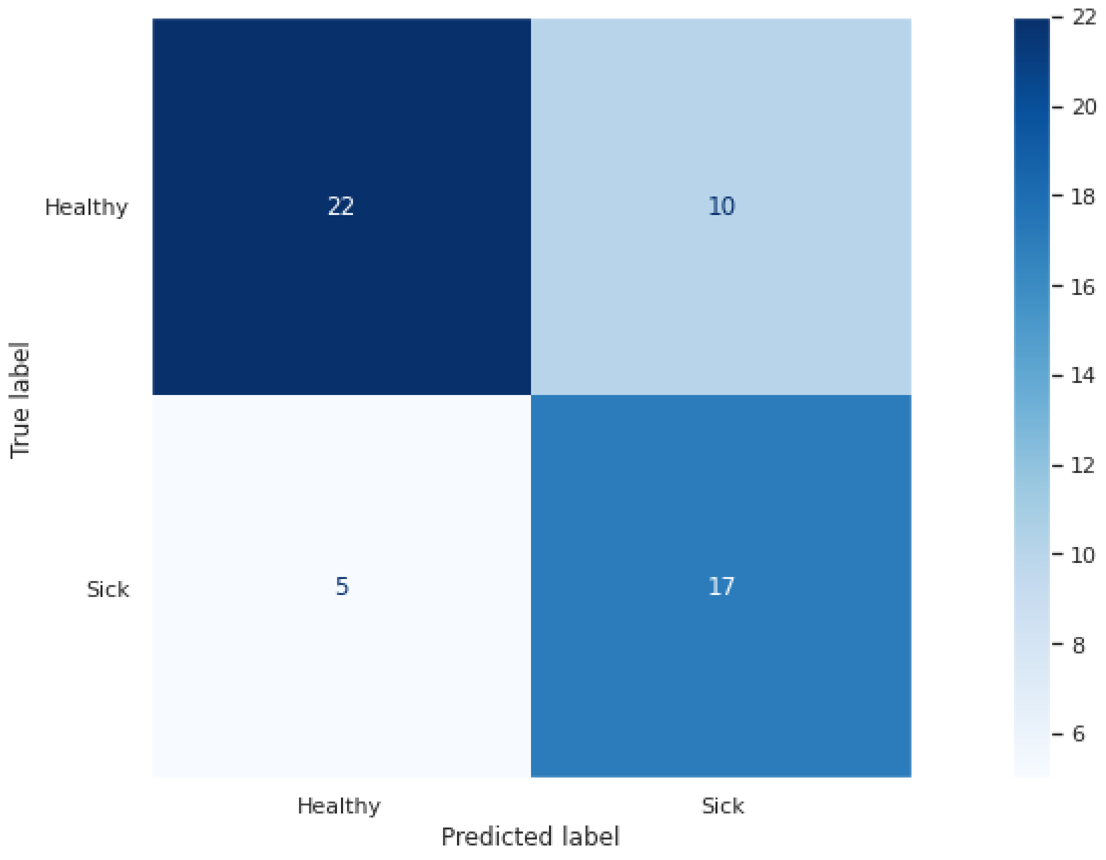
Confusion Matrix for our Decision Tree Model

Now we will visualize the results by using confusion matrix.

In [23]:

```
print(classification_report(y_test, y_pred))

plot_confusion_matrix(decision_tree_model, X_test, y_test, cmap=plt.cm.Blues, displ
plt.grid(None)
```

```
              precision    recall  f1-score   support

           1       0.81      0.69      0.75        32
           2       0.63      0.77      0.69        22

    accuracy                           0.72        54
   macro avg       0.72      0.73      0.72        54
weighted avg       0.74      0.72      0.72        54
```



SVM Learning Curves

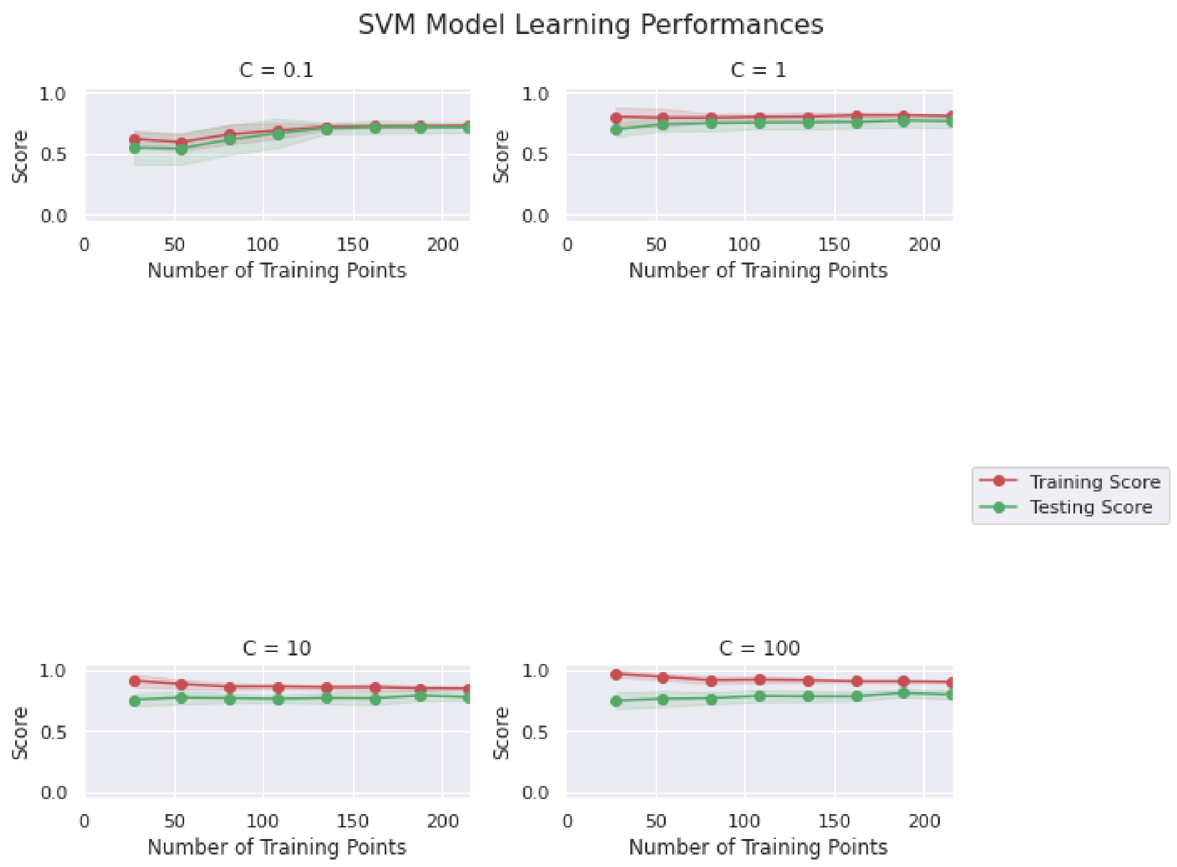
Now first of all we will implement Learning Curves for our Support Vector Machine model, for this purpose we will input different **C** values and we will see which will perfrom better

In [24]:

```
def ModelLearningSVM(X, y):
    # Create 10 cross-validation sets for training and testing
    cv = ShuffleSplit(n_splits = 10, test_size = 0.2, random_state = 0)

    # Generate the training set sizes increasing by 50
    train_sizes = np.rint(np.linspace(1, X.shape[0]*0.8 - 1, 9)).astype(int)

    # Create the figure window
    fig = plt.figure(figsize=(10,7))

    # Create different models based on C
    for k, C in enumerate([0.1, 1, 10, 100]):

        # Create a SVM Classifier at max_depth = depth
        classifier = SVC(C = C)

        # Calculate the training and testing scores
        sizes, train_scores, test_scores = learning_curve(classifier, X, y, cv = cv

        # Find the mean and standard deviation for smoothing
        train_std = np.std(train_scores, axis = 1)
        train_mean = np.mean(train_scores, axis = 1)
        test_std = np.std(test_scores, axis = 1)
        test_mean = np.mean(test_scores, axis = 1)

        # Subplot the learning curve
        ax = fig.add_subplot(2, 2, k+1)
        ax.plot(sizes, train_mean, 'o-', color = 'r', label = 'Training Score')
        ax.plot(sizes, test_mean, 'o-', color = 'g', label = 'Testing Score')
        ax.fill_between(sizes, train_mean - train_std, \
            train_mean + train_std, alpha = 0.15, color = 'r')
        ax.fill_between(sizes, test_mean - test_std, \
            test_mean + test_std, alpha = 0.15, color = 'g')

        # Labels
        ax.set_title('C = %s'%(C))
        ax.set_xlabel('Number of Training Points')
        ax.set_ylabel('Score')
        ax.set_xlim([0, X.shape[0]*0.8])
        ax.set_ylim([-0.05, 1.05])

    # Visual aesthetics
    ax.legend(bbox_to_anchor=(1.05, 2.05), loc='lower left', borderaxespad = 0.)
    fig.suptitle('SVM Model Learning Performances', fontsize = 16, y = 1.03)
    fig.tight_layout()
    fig.show()


def ModelComplexitySVM(X, y, param_name, param_range):
    # Create 10 cross-validation sets for training and testing
    cv = ShuffleSplit(n_splits = 10, test_size = 0.2, random_state = 0)

    # Calculate the training and testing scores
    train_scores, test_scores = validation_curve(SVC(kernel="rbf", C=100, gamma=0.1
        param_name = param_name, param_range = param_range, cv = cv)

    # Find the mean and standard deviation for smoothing
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    # Plot the validation curve
    plt.figure(figsize=(7, 5))
    plt.title('SVM Model Complexity Performance')
    plt.plot(param_range, train_mean, 'o-', color = 'r', label = 'Training Score')
    plt.plot(param_range, test_mean, 'o-', color = 'g', label = 'Validation Score')
    plt.fill_between(param_range, train_mean - train_std, \
        train_mean + train_std, alpha = 0.15, color = 'r')
    plt.fill_between(param_range, test_mean - test_std, \
        test_mean + test_std, alpha = 0.15, color = 'g')

    # Visual aesthetics
    plt.legend(loc = 'lower right')
    plt.xlabel(param_name)
    plt.ylabel('Score')
    plt.ylim([-0.05,1.05])
    plt.show()
```

In [25]:

```
ModelLearningSVM(features, label)
```



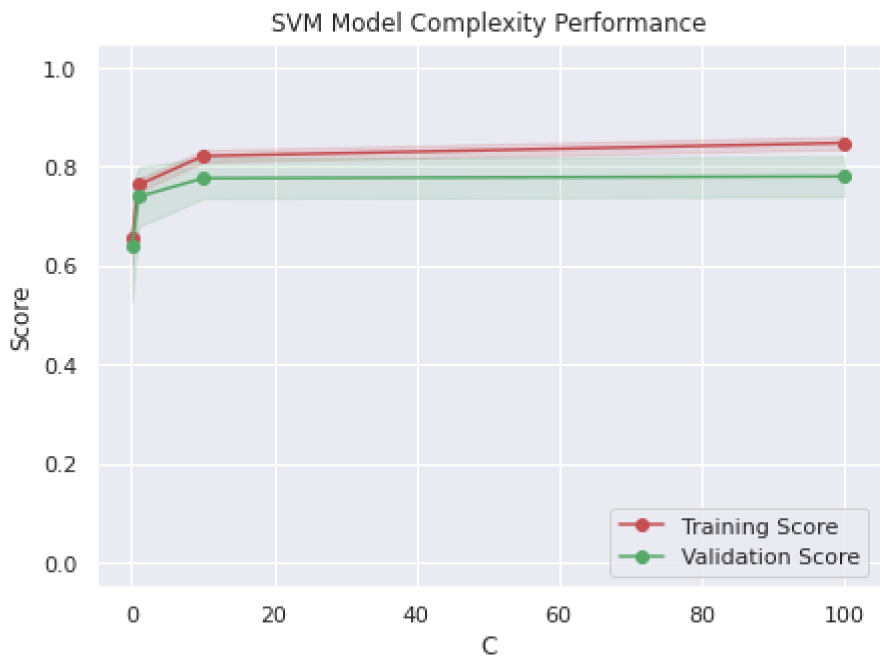
From Learning Curves we can see that **C = 1** and **C = 10** are perfroming better from other two options, and they both are giving approximatly same results so for our SVM model we can chose anyone of them, further we will see what results Model Complexity Graph will give us.

Model Complexity Graph for SVM

As we know that SVM rely on more the one hyper-parameters so for this reson we will plot two different model complexity graphs for SVM, first one will be for finding the best **C** value and the second one will be for finding the best **gamma** value

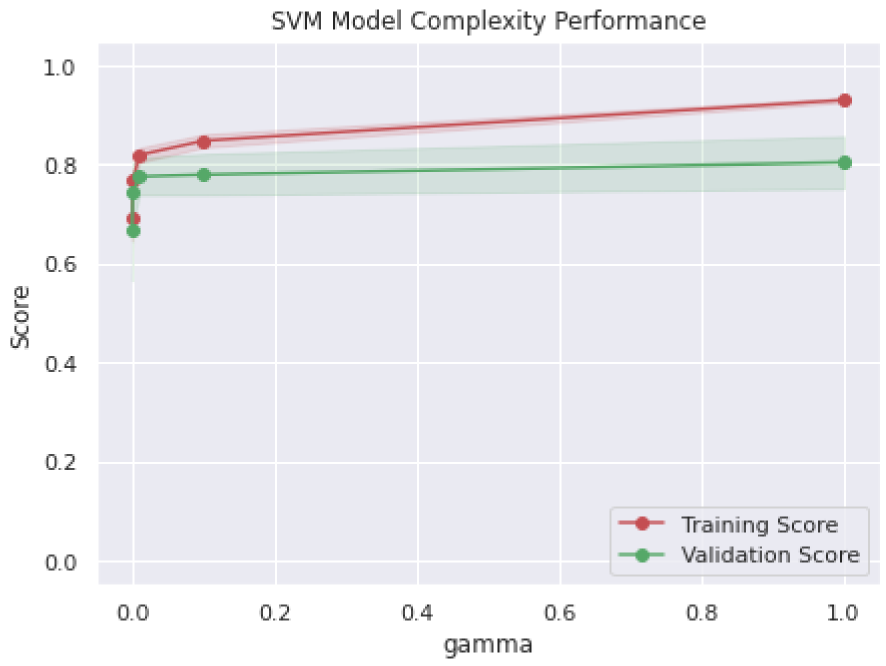
In [26]:

```
C = [0.1, 1, 10, 100]
ModelComplexitySVM(features, label, param_name="C", param_range=C)
```



In [27]:

```
gamma = [1, 0.1, 0.01, 0.001, 0.0001]
ModelComplexitySVM(features, label, param_name="gamma", param_range=gamma)
```



So from the above visualizations we can see that the best **C is 10** and the best **gamma is 0.1**, so now we will buid our SVM model using those parameters to get the best result

In [28]:

```
svm_model = SVC(kernel="rbf", C=10, gamma=0.1)
svm_model.fit(X_train, y_train)
svm_training_score = svm_model.score(X_train, y_train)

print("SVM model Training Score:", svm_training_score)
```

```
SVM model Training Score: 0.8287037037037037
```

In [29]:

```
svm_pred = svm_model.predict(X_test)
svm_testing_score = accuracy_score(svm_pred, y_test)

print("SVM model Testing Score:", svm_testing_score)
```

```
SVM model Testing Score: 0.7407407407407407
```

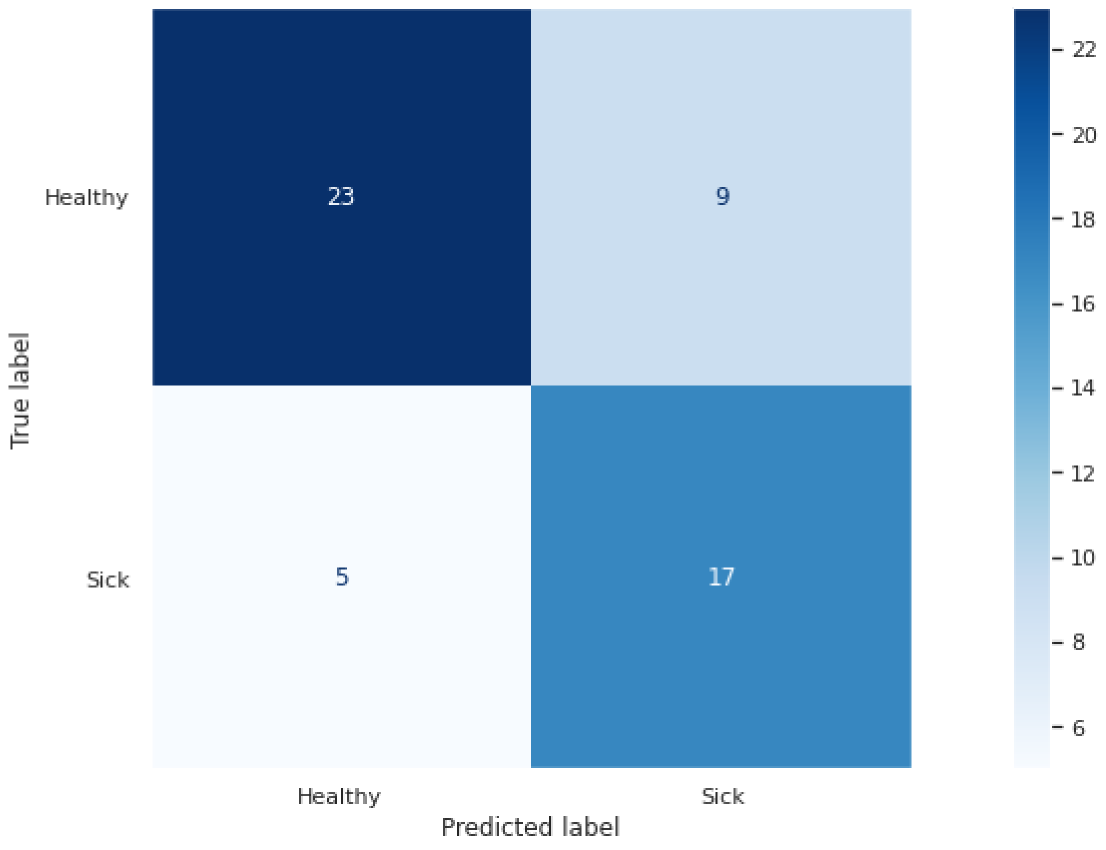
Confusion Matrix for our SVM Model

Now we will visualize the results by using confusion matrix.

In [30]:

```
print(classification_report(y_test, svm_pred))

plot_confusion_matrix(svm_model, X_test, y_test, cmap=plt.cm.Blues, display_labels=
plt.grid(None)
```

```
              precision    recall  f1-score   support

           1       0.82      0.72      0.77        32
           2       0.65      0.77      0.71        22

    accuracy                           0.74        54
   macro avg       0.74      0.75      0.74        54
weighted avg       0.75      0.74      0.74        54
```


Healthy
Sick
True label
Healthy
Sick
Predicted label