Naive bayes model

In [1]:
```
# Import relevant libraries and modules.

import pandas as pd
from sklearn import naive_bayes
from sklearn import model_selection
from sklearn import metrics
```

In [2]:
```
data = pd.read_csv("_-dP7-fhQwOnT-_n4eMDvA_821463c49d9f4735877cc5f7b386acf
```

In [3]:
```
data.head(10)
```

Out[3]:

	fg	3p	ft	reb	ast	stl	blk	tov	target_5yrs	total_points	efficiency
0	34.7	25.0	69.9	4.1	1.9	0.4	0.4	1.3	0	266.4	9.722628
1	29.6	23.5	76.5	2.4	3.7	1.1	0.5	1.6	0	252.0	9.368030
2	42.2	24.4	67.0	2.2	1.0	0.5	0.3	1.0	0	384.8	25.150327
3	42.6	22.6	68.9	1.9	0.8	0.6	0.1	1.0	1	330.6	28.500000
4	52.4	0.0	67.4	2.5	0.3	0.3	0.4	0.8	1	216.0	18.782609
5	42.3	32.5	73.2	0.8	1.8	0.4	0.0	0.7	0	277.5	24.342105
6	43.5	50.0	81.1	2.0	0.6	0.2	0.1	0.7	1	409.2	37.541284
7	41.5	30.0	87.5	1.7	0.2	0.2	0.1	0.7	1	273.6	26.563107
8	39.2	23.3	71.4	0.8	2.3	0.3	0.0	1.1	0	156.0	15.757576
9	38.3	21.4	67.8	1.1	0.3	0.2	0.0	0.7	0	155.4	18.282353

In [4]:
```
# Define the y (target) variable.

y = data['target_5yrs']

# Define the X (predictor) variables.

X = data.drop('target_5yrs', axis = 1)
```

In [5]:
```
# Display the first 10 rows of your target data.

y.head(10)
```

Out[5]:
```
0    0
1    0
2    0
3    1
4    1
5    0
6    1
7    1
8    0
9    0
Name: target_5yrs, dtype: int64
```

In [6]:
```
# Display the first 10 rows of your predictor variables.

X.head(10)
```

Out[6]:

	fg	3p	ft	reb	ast	stl	blk	tov	total_points	efficiency
0	34.7	25.0	69.9	4.1	1.9	0.4	0.4	1.3	266.4	9.722628
1	29.6	23.5	76.5	2.4	3.7	1.1	0.5	1.6	252.0	9.368030
2	42.2	24.4	67.0	2.2	1.0	0.5	0.3	1.0	384.8	25.150327
3	42.6	22.6	68.9	1.9	0.8	0.6	0.1	1.0	330.6	28.500000
4	52.4	0.0	67.4	2.5	0.3	0.3	0.4	0.8	216.0	18.782609
5	42.3	32.5	73.2	0.8	1.8	0.4	0.0	0.7	277.5	24.342105
6	43.5	50.0	81.1	2.0	0.6	0.2	0.1	0.7	409.2	37.541284
7	41.5	30.0	87.5	1.7	0.2	0.2	0.1	0.7	273.6	26.563107
8	39.2	23.3	71.4	0.8	2.3	0.3	0.0	1.1	156.0	15.757576
9	38.3	21.4	67.8	1.1	0.3	0.2	0.0	0.7	155.4	18.282353

In [7]:
```
# Perform the split operation on your data.
# Assign the outputs as follows: X_train, X_test, y_train, y_test.

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
```

In [8]:
```
# Print the shape (rows, columns) of the output from the train-test split.

# Print the shape of X_train.
print(X_train.shape)

# Print the shape of X_test.

print(X_test.shape)

# Print the shape of y_train.


print(y_train.shape)

# Print the shape of y_test.

print(y_test.shape)
```

```
(1005, 10)
(335, 10)
(1005,)
(335,)
```

Model building

Which Naive Bayes algorithm should you use?

Using the assumption that your features are normally distributed and continuous, the Gaussian Naive Bayes algorithm is most appropriate for your data. While your data may not perfectly adhere to these assumptions, this model will still yield the most usable and accurate results.

In [10]:
```
# Assign `nb` to be the appropriate implementation of Naive Bayes.

nb = naive_bayes.GaussianNB()

# Fit the model on your training data.


nb.fit(X_train, y_train)

# Apply your model to predict on your test data. Call this "y_pred"

y_pred = nb.predict(X_test)
```

result and evaluation

In [11]:
```
# Print your accuracy score.

###http://localhost:8888/notebooks/naive%20bayes%20model.ipynb#result-and-

print('accuracy score:'), print(metrics.accuracy_score(y_test, y_pred))

# Print your precision score.

print('precision score:'), print(metrics.precision_score(y_test, y_pred))

# Print your recall score.

print('recall score:'), print(metrics.recall_score(y_test, y_pred))

# Print your f1 score.

print('f1 score:'), print(metrics.f1_score(y_test, y_pred))
```

```
accuracy score:
0.6985074626865672
precision score:
0.8211920529801324
recall score:
0.6262626262626263
f1 score:
0.7106017191977076
```

Out[11]: (None, None)

Question: What is the accuracy score for your model, and what does this tell you about the success of the model's performance?

The accuracy score for this model is 0.713, or 71.3% accurate.

Question: Can you evaluate the success of your model by using the accuracy score exclusively?

In classification problems, accuracy is useful to know but may not be the best metric by which to evaluate this model. While accuracy is often the most intuitive metric, it is a poor evaluation metric in some cases. In particular, if you have imbalanced classes, a model could appear accurate but be poor at balancing false positives and false negatives.

Question: What are the precision and recall scores for your model, and what do they mean? Is one of these scores more accurate than the other?

Precision and recall scores are both useful to evaluate the correct predictive capability of a model because they balance the false positives and false negatives inherent in prediction.

The model shows a precision score of 0.845, suggesting the model is quite good at predicting true positives—meaning the player will play longer than five years—while balancing false positives. The recall score of 0.6375 shows worse performance in predicting true negatives—where the player will not play for five years or more—while balancing false negatives.These two metrics combined can give a better assessment of model performance than accuracy does alone.

Question: What is the F1 score of your model, and what does this score mean?

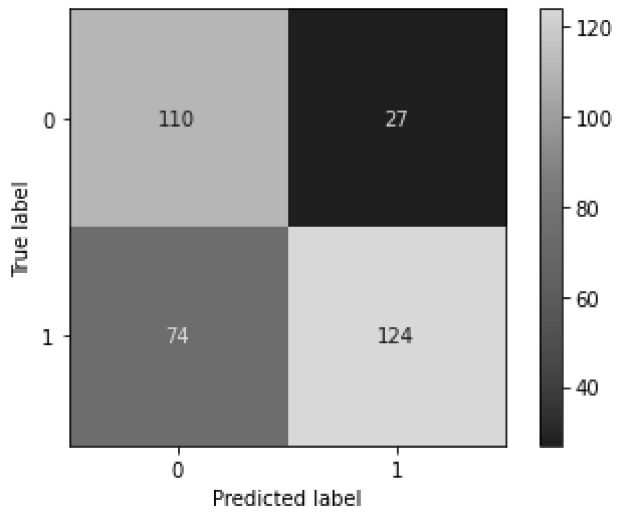
The F1 score balances the precision and recall performance to give a combined assessment of how well this model delivers predictions. In this case, the F1 score is 0.7268, which suggests reasonable predictive power in this model.

Gain clarity with the confusion matrix Recall that a confusion matrix is a graphic that shows your model's true and false positives and negatives. It helps to create a visual representation of the components feeding into the metrics.

Create a confusion matrix based on your predicted values for the test set

In [12]:
```
# Construct and display your confusion matrix.

# Construct the confusion matrix for your predicted and test values.

cm = metrics.confusion_matrix(y_test, y_pred)

# Create the display for your confusion matrix.
disp = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=

# Plot the visual in-line.

disp.plot()
```

Out[12]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x14719771c
10>
```



Question: What do you notice when observing your confusion matrix, and does this correlate to any of your other calculations?

The top left to bottom right diagonal in the confusion matrix represents the correct predictions, and the ratio of these squares showcases the accuracy.

The concentration of true positives stands out relative to false positives. This ratio is why the precision score is so high (0.845).

True negatives and false negatives are closer in number, which explains the worse recall score.

In [ ]:
```
1
```