Some terminologies used:

Precision
Definition: The ratio of true positives to the total predicted positives.
Indicates: How many of the predicted positives are actually correct.
Example: If a model predicts 10 survivors and 8 are correct, precision is 80%.
Formula: Precision = (TP)/(TP+FP)

Recall
Definition: The ratio of true positives to the actual positives.
Indicates: How many actual positives are correctly identified.
Example: If there are 20 actual survivors and 15 are correctly predicted, recall is 75%.
Formula: Recall = (TP)/(TP+FN)

F1-Score
Definition: The harmonic mean of precision and recall.
Indicates: A balanced measure of the model's accuracy.
Example: Balances both precision and recall into a single metric.
Formula: 2 * (Precision*Recall)/(Precision+Recall)

True Positives (TP)
Definition: Correctly predicted positive instances.
Example: Actual survivors correctly identified by the model.

False Positives (FP)
Definition: Incorrectly predicted positive instances.
Example: Non-survivors incorrectly predicted as survivors.

True Negatives (TN)
Definition: Correctly predicted negative instances.
Example: Non-survivors correctly identified by the model.

False Negatives (FN)
Definition: Incorrectly predicted negative instances.
Example: Actual survivors incorrectly predicted as non-survivors.

These metrics help evaluate and understand a model's performance in classifying data.

TITANIC SURVIVAL PREDICTION

Five different supervised machine learning algorithms were evaluated to determine the most effective one for predicting survival outcomes on the Titanic.

Logistic Regression

Model Initialization:

```
#Initializa the model (Logistic Regression)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
model = LogisticRegression(max_iter=1000)
```

```
#Train the model
```

```
model.fit(X_train, y_train)
```

```
▾        LogisticRegression
LogisticRegression(max_iter=1000)
```

```
#Make predictions
```

```
y_pred = model.predict(X_test)
```

Model Evaluation:

```
#Evaluate the model
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print('Classification Report:')
print(classification_report(y_test, y_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.7972027972027972
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.91      0.83        80
           1       0.85      0.65      0.74        63

    accuracy                           0.80       143
   macro avg       0.81      0.78      0.79       143
weighted avg       0.81      0.80      0.79       143

Confusion Matrix:
[[73  7]
 [22 41]]
```

Results and Conclusion:

Logistic Regression Performance Summary

Accuracy: 79.72%
Indicates: The model correctly predicts about 80% of the instances overall.

Classification Report

Class 0 (Negative Class)

Precision: 0.77 - 77% of predicted class 0 are true class 0.
Recall: 0.91 - 91% of actual class 0 are correctly identified.
F1-Score: 0.83 - Balance between precision and recall.

Class 1 (Positive Class)

Precision: 0.85 - 85% of predicted class 1 are true class 1.
Recall: 0.65 - 65% of actual class 1 are correctly identified.
F1-Score: 0.74 - Lower due to missed class 1 instances.

Overall Averages

Macro Avg: Precision 0.81, Recall 0.78, F1-Score 0.79
Weighted Avg: Precision 0.81, Recall 0.80, F1-Score 0.79

True Negatives (73): Correctly identified as class 0.
False Positives (7): Incorrectly identified as class 1.
False Negatives (22): Missed class 1 instances, predicted as class 0.
True Positives (41): Correctly identified as class 1.

Key Insights

Strengths: High precision and recall for class 0; high precision for class 1.
Weaknesses: Lower recall for class 1 indicates missed positives.
Improvement Areas: Enhance recall for class 1 through threshold adjustment or balancing techniques.

Overall, the model performs well but needs better detection of class 1 to improve its predictive capabilities.

K Nearest Neighbors Classification

Model Initialization:

```
#KNN Classification
```

```
#Initialize the model (Knearest Neighbors Regression)
```

```
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
knn_class = KNeighborsClassifier()
```

```
#Split the data into training and test sets (for classification)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
#Initialize KNN
```

```
knn_class = KNeighborsClassifier(n_neighbors=5)
```

```
#Train the model (Classification)
```

```
knn_class.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
#Make predictions
```

```
y_pred_knn_class=knn_class.predict(X_test)
```

Model Evaluation:

```
#Evaluate the model
```

```
accuracy_knn_class = accuracy_score(y_test, y_pred_knn_class)
print(f'Accuracy of KNN Classification: {accuracy_knn_class}')
print('Classification Report:')
print(classification_report(y_test, y_pred_knn_class))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred_knn_class))
```

```
Accuracy of KNN Classification: 0.6853146853146853
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.68      0.71        80
           1       0.63      0.70      0.66        63

    accuracy                           0.69       143
   macro avg       0.68      0.69      0.68       143
weighted avg       0.69      0.69      0.69       143

Confusion Matrix:
[[54 26]
 [19 44]]
```

Results and Conclusion:

KNN Classification Performance Summary

Accuracy: 68.53%

The model correctly predicts about 68.53% of instances.

Classification Report

Class 0 (Did Not Survive)

Precision: 0.74 - Accurate predictions for non-survivors.
Recall: 0.68 - Captures most non-survivors.
F1-Score: 0.71 - Balanced but misses some non-survivors.

Class 1 (Survived)

Precision: 0.63 - Accurate predictions for survivors.
Recall: 0.70 - Captures majority of survivors.
F1-Score: 0.66 - Moderate balance, some misses.

Confusion Matrix

54 True Negatives: Correct non-survivors.
26 False Positives: Non-survivors predicted as survivors.
19 False Negatives: Survivors predicted as non-survivors.
44 True Positives: Correct survivors.

Insights

Strengths: Good balance in capturing both classes.
Weaknesses: Needs improvement in reducing false positives and better identifying non-survivors.

The KNN model shows moderate accuracy with balanced but improvable precision and recall for both classes.

Support Vector Classification

Model Initialization:

```
#Support Vector Classification
```

```
from sklearn.svm import SVC
```

```
# Initialize the model
```

```
svc_class = SVC()
```

```
#Train the model (Classification)
```

```
svc_class.fit(X_train, y_train)
```

```
▾ SVC
SVC()
```

```
#Make predictions
```

```
y_pred_svc_class=svc_class.predict(X_test)
```

Model Evaluation:

```
#Evaluate the model
```

```
accuracy_svc_class = accuracy_score(y_test, y_pred_svc_class)
print(f'Accuracy of SVC Classification: {accuracy_svc_class}')
print('Classification Report:')
print(classification_report(y_test, y_pred_svc_class))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred_svc_class))
```

```
Accuracy of SVC Classification: 0.6363636363636364
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.84      0.72        80
           1       0.65      0.38      0.48        63

    accuracy                           0.64       143
   macro avg       0.64      0.61      0.60       143
weighted avg       0.64      0.64      0.61       143

Confusion Matrix:
[[67 13]
 [39 24]]
```

Results and Conclusion:

SVC Classification Performance Summary

Accuracy: 63.64%

Correctly predicts about 63.64% of instances.

Classification Report

Class 0 (Did Not Survive)
Precision: 0.63 - Reasonable accuracy for non-survivors.
Recall: 0.84 - High rate of detecting non-survivors.
F1-Score: 0.72 - Strong balance, but some false positives.

Class 1 (Survived)
Precision: 0.65 - Moderate accuracy for survivors.
Recall: 0.38 - Low rate of detecting survivors.
F1-Score: 0.48 - Struggles with survivor predictions.

Confusion Matrix
67 True Negatives: Correctly predicted non-survivors.
13 False Positives: Non-survivors predicted as survivors.
39 False Negatives: Survivors predicted as non-survivors.
24 True Positives: Correctly predicted survivors.

Insights
Strengths: High recall for class 0, accurately identifies most non-survivors.
Weaknesses: Poor recall for class 1, misses many survivors.

The SVC model shows decent performance for predicting non-survivors but struggles significantly in accurately predicting survivors.

Random Forest Classifier

Model Initialization:

```python
#Random forest classifier
```

```python
from sklearn.ensemble import RandomForestClassifier
```

```python
#Initialize the model
```

```python
rf_class = RandomForestClassifier()
```

```python
#Train the model (Classification)
```

```python
rf_class.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```python
#Make predictions
```

```python
y_pred_rf_class=rf_class.predict(X_test)
```

Model Evaluation:

```python
#Evaluate the model
```

```python
accuracy_rf_class = accuracy_score(y_test, y_pred_rf_class)
print(f'Accuracy of RF Classification: {accuracy_rf_class}')
print('Classification Report:')
print(classification_report(y_test, y_pred_rf_class))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred_rf_class))
```

```
Accuracy of RF Classification: 0.7692307692307693
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.84      0.80        80
           1       0.77      0.68      0.72        63

    accuracy                           0.77       143
   macro avg       0.77      0.76      0.76       143
weighted avg       0.77      0.77      0.77       143

Confusion Matrix:
[[67 13]
 [20 43]]
```

Results and Conclusion:

Random Forest Classification Performance Summary

Accuracy: 76.92%

Correctly predicts about 76.92% of instances.

Classification Report

Class 0 (Did Not Survive)

Precision: 0.77 - Accurate in identifying non-survivors.
Recall: 0.84 - Captures most non-survivors.
F1-Score: 0.80 - Strong performance overall.

Class 1 (Survived)

Precision: 0.77 - Good at predicting survivors.
Recall: 0.68 - Misses some survivors.
F1-Score: 0.72 - Balanced but needs improvement.

Confusion Matrix

67 True Negatives: Correctly predicted non-survivors.
13 False Positives: Non-survivors predicted as survivors.
20 False Negatives: Survivors predicted as non-survivors.
43 True Positives: Correctly predicted survivors.

Insights

Strengths: High recall for non-survivors and balanced performance across both classes.
Weaknesses: Moderate recall for survivors; room for improvement in identifying all survivors.

The Random Forest model provides strong and balanced performance, effectively identifying non-survivors while maintaining good predictions for survivors.

Gradient Boosting Classifier

Model Initialization:

```
#Gradient Boosting Classifier
```

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
#Initialize the model
```

```
gb_class = GradientBoostingClassifier()
```

```
#Train the model (Classification)
```

```
gb_class.fit(X_train, y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

```
#Make prediction
```

```
y_pred_gb_class=gb_class.predict(X_test)
```

Model Evaluation:

```
#Evaluate the model
```

```
accuracy_gb_class = accuracy_score(y_test, y_pred_gb_class)
print(f'Accuracy of GB Classification: {accuracy_gb_class}')
print('Classification Report:')
print(classification_report(y_test, y_pred_gb_class))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred_gb_class))
```

```
Accuracy of GB Classification: 0.7692307692307693
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.85      0.80        80
           1       0.78      0.67      0.72        63

    accuracy                           0.77       143
   macro avg       0.77      0.76      0.76       143
weighted avg       0.77      0.77      0.77       143

Confusion Matrix:
[[68 12]
 [21 42]]
```

Results and Conclusion:

Gradient Boosting Classification Performance Summary

Accuracy: 76.92%
Correctly predicts about 76.92% of instances.

Classification Report

Class 0 (Did Not Survive)

Precision: 0.76 - Good at predicting non-survivors.
Recall: 0.85 - High rate of capturing non-survivors.
F1-Score: 0.80 - Strong overall performance.

Class 1 (Survived)
Precision: 0.78 - Accurate in predicting survivors.
Recall: 0.67 - Misses some survivors.
F1-Score: 0.72 - Balanced but can improve in recall.

Confusion Matrix
68 True Negatives: Correctly predicted non-survivors.
12 False Positives: Non-survivors predicted as survivors.
21 False Negatives: Survivors predicted as non-survivors.
42 True Positives: Correctly predicted survivors.

Insights
Strengths: High recall for non-survivors and balanced performance across classes.
Weaknesses: Moderate recall for survivors; some missed cases.

Gradient Boosting offers robust performance, particularly for non-survivors, while maintaining good but slightly lower effectiveness in predicting survivors.

Best Model Selection: Logistic Regression
Reasons:

Highest Accuracy: Logistic Regression has the highest accuracy at 79.72%.
Balanced Metrics: It provides the best balance of precision, recall, and F1-score for both classes, indicating robust performance in both identifying survivors and non-survivors.

Lower False Positives and Negatives: It has lower false positives and a good balance between true positives and false negatives, leading to reliable predictions.

Interpretability: Logistic Regression models are easier to interpret, which is valuable for understanding the factors affecting survival.

Runner-up:
Gradient Boosting and Random Forest: Both offer strong and balanced performance but slightly lower than Logistic Regression. They are good alternatives due to their high recall for non-survivors and balanced precision and recall.

In conclusion, Logistic Regression is the best choice for predicting Titanic survival based on its superior accuracy, balanced performance metrics, and interpretability.