

```

1 from google.colab import drive
2 drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5
6 from sklearn.model_selection import train_test_split, learning_curve
7 from sklearn.preprocessing import StandardScaler, LabelEncoder
8 from sklearn.metrics import (confusion_matrix, classification_report,
9                               precision_score, recall_score, f1_score,
10                             roc_curve, auc, ConfusionMatrixDisplay)
11
12 from sklearn.linear_model import LogisticRegression
13 from sklearn.neighbors import KNeighborsClassifier
14 from sklearn.tree import DecisionTreeClassifier
15 from sklearn.ensemble import RandomForestClassifier
16 from sklearn.svm import SVC
17
18 import warnings
19 warnings.filterwarnings('ignore')
20 sns.set(style="whitegrid")

```

```
1 t=pd.read_csv('/content/drive/MyDrive/Lab_performance/Titanic - Machine Learning from Disaster.csv')
```

```
1 t
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	
...	...	...	...	...	...	...	...	...	...	...	...	
413	1305	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 3236	8.0500	NaN	S	
414	1306	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000	C105	C	
415	1307	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500	NaN	S	
416	1308	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500	NaN	S	

Next steps: [Generate code with t](#) [New interactive sheet](#)

```

1 file_name = '/content/drive/MyDrive/Lab_performance/Titanic - Machine Learning from Disaster.csv'
2 df = pd.read_csv(file_name)

```

## Data Preprocessing

```

1 # --- Data Cleaning ---
2
3 # 1. Handle Missing Values:
4 # 'Age': Impute with median
5 df['Age'].fillna(df['Age'].median(), inplace=True)
6 # 'Fare': Impute with median
7 df['Fare'].fillna(df['Fare'].median(), inplace=True)
8 # 'Cabin': Too many missing values. Drop the column.
9 df.drop('Cabin', axis=1, inplace=True)
10

```

```

11 # 2. Remove Duplicate Rows:
12 df.drop_duplicates(inplace=True)
13
14 # 3. Drop Unnecessary Features:
15 df.drop(['Name', 'Ticket', 'PassengerId'], axis=1, inplace=True)
16
17 print("Data Cleaning Complete. Ready for EDA.")
18 print(df.head())

```

```

Data Cleaning Complete. Ready for EDA.
   Pclass      Sex   Age  SibSp  Parch     Fare Embarked
0      3    male  34.5      0      0   7.8292        Q
1      3  female  47.0      1      0   7.0000        S
2      2    male  62.0      0      0   9.6875        Q
3      3    male  27.0      0      0   8.6625        S
4      3  female  22.0      1      1  12.2875        S

```

## Encoding & Feature scaling

```

1 # Define features (X) and multi-class target (y)
2 X = df.drop('Embarked', axis=1)
3 y = df['Embarked'] # Last column is the target
4
5 # --- Target Encoding (Label Encoding for Multi-Class Target) ---
6 le = LabelEncoder()
7 y_encoded = le.fit_transform(y)
8 # Store the classes for reporting later: ['C', 'Q', 'S'] -> [0, 1, 2]
9 target_classes = le.classes_
10 print(f"Embarked classes encoded: {le.classes_}")
11
12 # --- Feature Encoding (One-Hot Encoding for X) ---
13 X_encoded = pd.get_dummies(X, columns=['Sex', 'Pclass'], drop_first=True)
14
15 # --- Feature Scaling (StandardScaler) ---
16 numerical_cols = ['Age', 'Fare', 'SibSp', 'Parch']
17 scaler = StandardScaler()
18 X_encoded[numerical_cols] = scaler.fit_transform(X_encoded[numerical_cols])
19
20 # --- Train-Test Split (80% Train / 20% Test) ---
21 X_train, X_test, y_train, y_test = train_test_split(
22     X_encoded, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
23 )
24
25 print(f"\nX_train shape: {X_train.shape}")
26 print(f"X_test shape: {X_test.shape}")

Embarked classes encoded: ['C' 'Q' 'S']

X_train shape: (334, 7)
X_test shape: (84, 7)

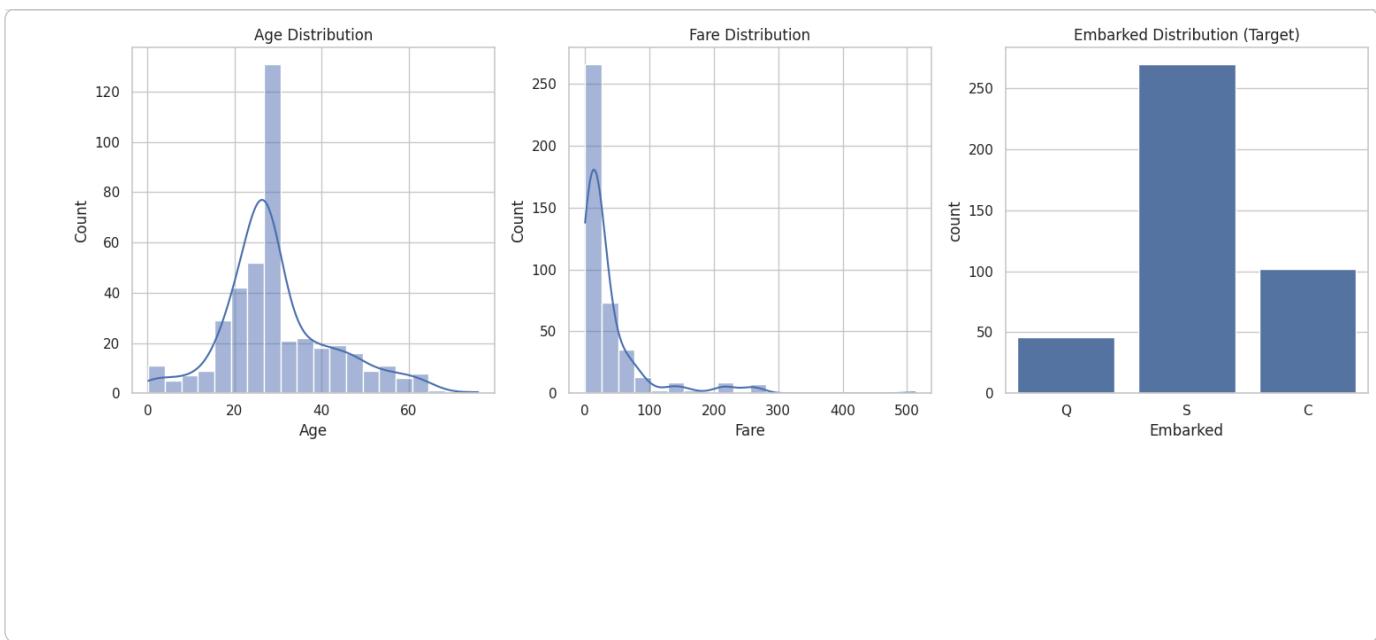
```

## Histogram

```

1 # Set up the visualization style
2 sns.set_style('whitegrid')
3 plt.rcParams['figure.dpi'] = 100
4
5 # 1. Histograms (Feature Distribution)
6 plt.figure(figsize=(15, 5))
7 plt.subplot(1, 3, 1)
8 sns.histplot(df['Age'], kde=True, bins=20)
9 plt.title('Age Distribution')
10 plt.subplot(1, 3, 2)
11 sns.histplot(df['Fare'], kde=True, bins=20)
12 plt.title('Fare Distribution')
13 plt.subplot(1, 3, 3)
14 # Distribution of the new target variable
15 sns.countplot(x='Embarked', data=df)
16 plt.title('Embarked Distribution (Target)')
17 plt.tight_layout()
18 plt.savefig('Histograms_EDA.png')
19 plt.show()

```

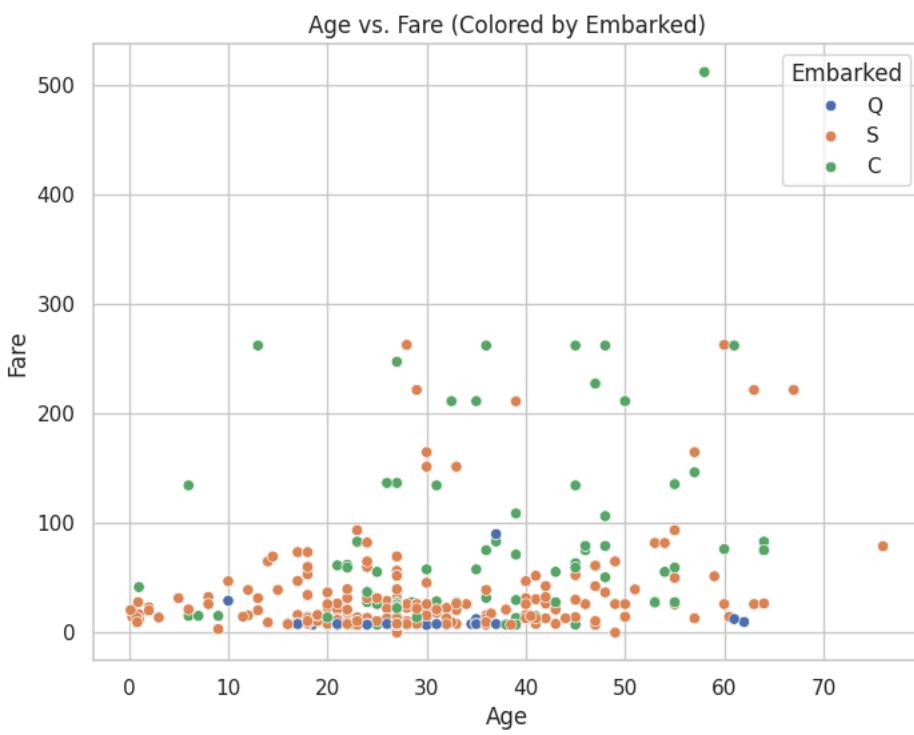


### Scatter Plot

```

1 # 2. Scatter Plots (Example: Age vs. Fare colored by Embarked)
2 plt.figure(figsize=(8, 6))
3 sns.scatterplot(x='Age', y='Fare', hue='Embarked', data=df)
4 plt.title('Age vs. Fare (Colored by Embarked)')
5 plt.savefig('ScatterPlot_Embarked.png')
6 plt.show()
7
8

```



### Correlation Heatmap

```

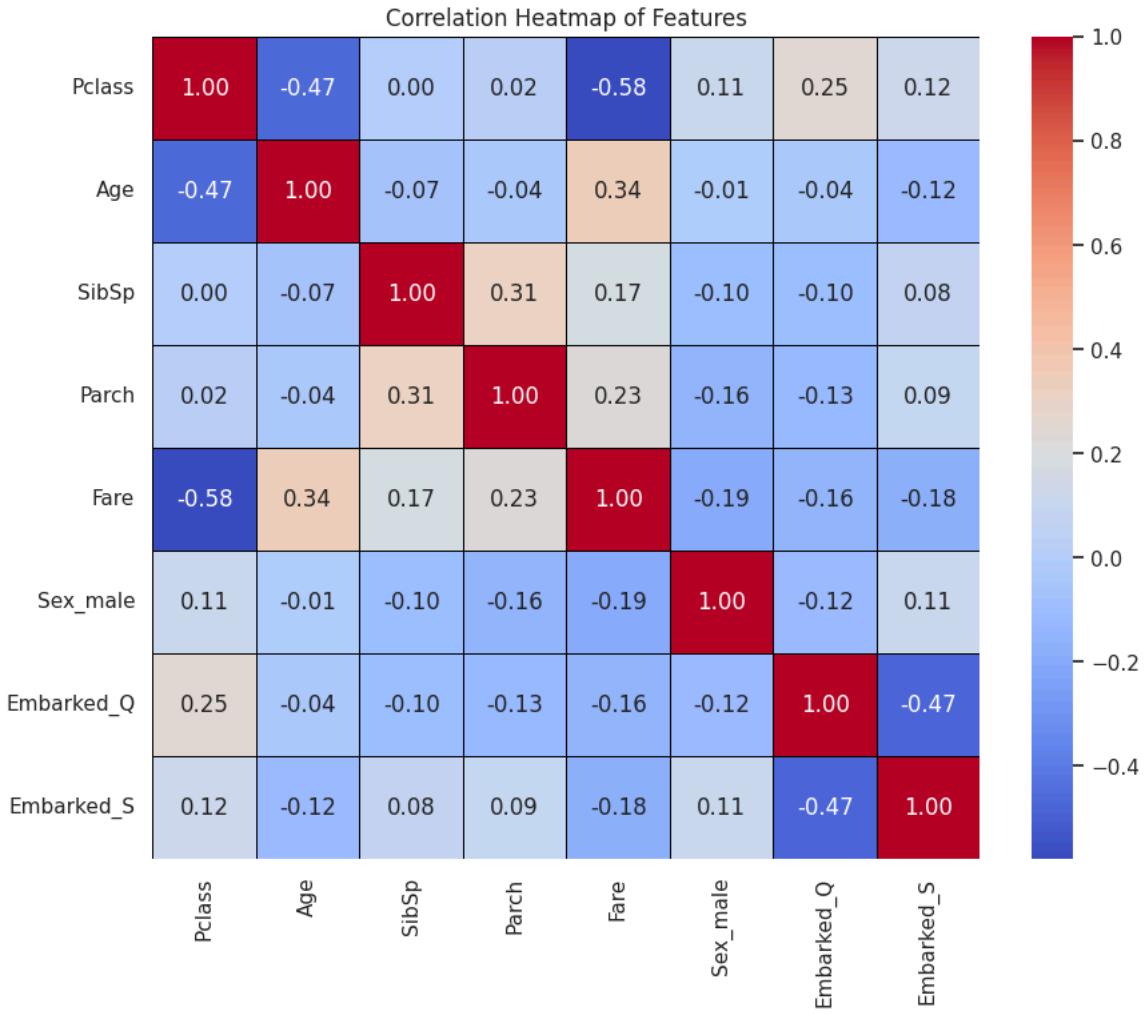
1 # 3. Correlation Heatmap
2 # Need to encode categorical features temporarily for correlation calculation
3 df_corr = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)
4 correlation_matrix = df_corr.corr()
5

```

```

6 plt.figure(figsize=(10, 8))
7 sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5, linecolor='black')
8 plt.title('Correlation Heatmap of Features')
9 plt.savefig('Correlation_Heatmap.png')
10 plt.show()

```



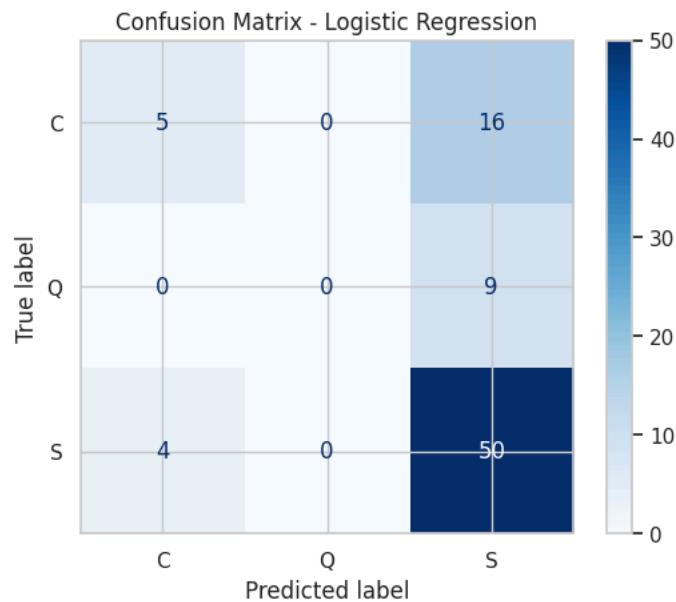
## Logistic Regression

```

1 # Model 1 - Logistic Regression (with CM Plot)
2
3 print(" Model 1: Logistic Regression")
4 log_reg = LogisticRegression(random_state=42, solver='lbfgs', multi_class='multinomial')
5 log_reg.fit(X_train, y_train)
6 y_pred_log_reg = log_reg.predict(X_test)
7 y_prob_log_reg = log_reg.predict_proba(X_test)
8
9 # Confusion Matrix PLOT
10 cm_log_reg = confusion_matrix(y_test, y_pred_log_reg)
11 disp_log_reg = ConfusionMatrixDisplay(confusion_matrix=cm_log_reg, display_labels=target_classes)
12 plt.figure(figsize=(8, 8))
13 disp_log_reg.plot(cmap=plt.cm.Blues, values_format='d')
14 plt.title('Confusion Matrix - Logistic Regression')
15 plt.savefig('CM_LogisticRegression.png')
16 plt.show()
17
18 # Classification Report
19 print("\nClassification Report:")
20 print(classification_report(y_test, y_pred_log_reg, target_names=target_classes))
21
22

```

Model 1: Logistic Regression  
<Figure size 800x800 with 0 Axes>

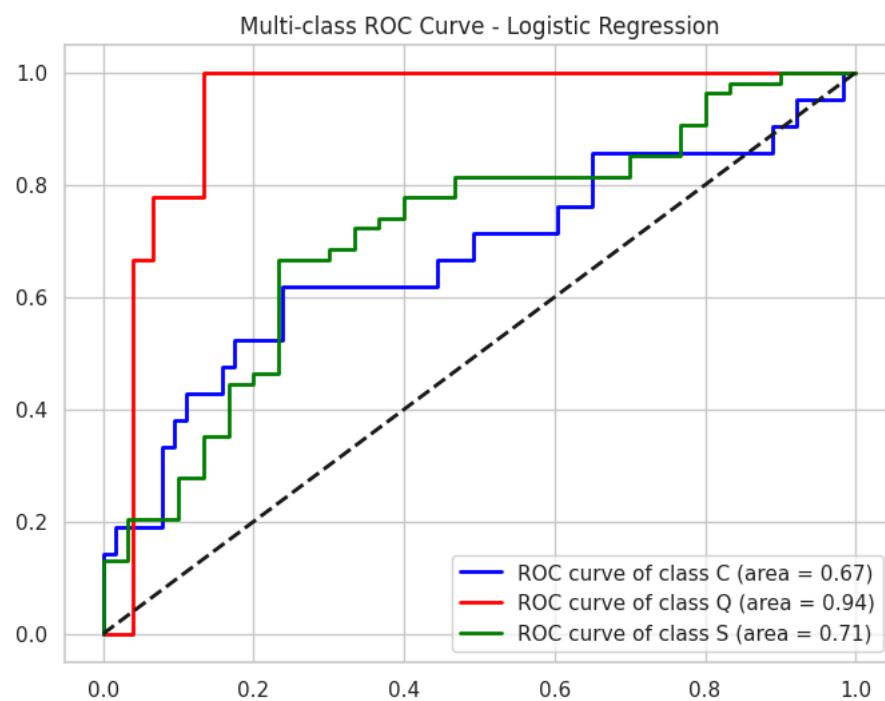


Classification Report:				
	precision	recall	f1-score	support
C	0.56	0.24	0.33	21
Q	0.00	0.00	0.00	9
S	0.67	0.93	0.78	54
accuracy			0.65	84
macro avg	0.41	0.39	0.37	84
weighted avg	0.57	0.65	0.58	84

```

1 # ROC Curve Plot (Multi-Class One-vs-Rest)
2 from itertools import cycle
3 fpr = dict()
4 tpr = dict()
5 roc_auc = dict()
6 n_classes = len(target_classes)
7 for i in range(n_classes):
8     fpr[i], tpr[i], _ = roc_curve(y_test == i, y_prob_log_reg[:, i])
9     roc_auc[i] = auc(fpr[i], tpr[i])
10 plt.figure(figsize=(8, 6))
11 colors = cycle(['blue', 'red', 'green'])
12 for i, color in zip(range(n_classes), colors):
13     plt.plot(fpr[i], tpr[i], color=color, lw=2,
14               label=f'ROC curve of class {target_classes[i]} (area = {roc_auc[i]:.2f})')
15 plt.plot([0, 1], [0, 1], 'k--', lw=2)
16 plt.title('Multi-class ROC Curve - Logistic Regression')
17 plt.legend(loc="lower right")
18 plt.savefig('ROC_Multiclass_LogisticRegression.png')
19 plt.show()

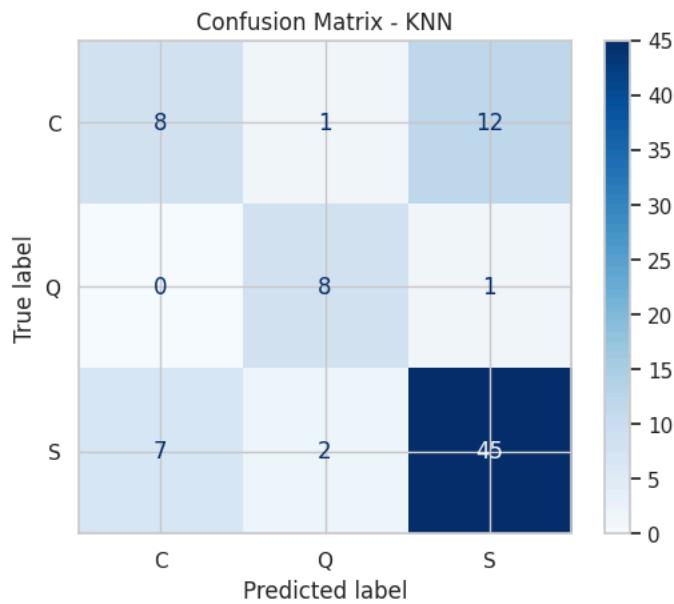
```



## K-Nearest Neighbors (KNN)

```
1 #Model 2 K-Nearest Neighbors (KNN) (with CM Plot)
2 print("Model 2: K-Nearest Neighbors (KNN)")
3 knn = KNeighborsClassifier(n_neighbors=5)
4 knn.fit(X_train, y_train)
5 y_pred_knn = knn.predict(X_test)
6 # Confusion Matrix PLOT
7 cm_knn = confusion_matrix(y_test, y_pred_knn)
8 disp_knn = ConfusionMatrixDisplay(confusion_matrix=cm_knn, display_labels=target_classes)
9 plt.figure(figsize=(8, 8))
10 disp_knn.plot(cmap=plt.cm.Blues, values_format='d')
11 plt.title('Confusion Matrix - KNN')
12 plt.savefig('CM_KNN.png')
13 plt.show()
14 # Classification Report
15 print("\nClassification Report:")
16 print(classification_report(y_test, y_pred_knn, target_names=target_classes))
```

Model 2: K-Nearest Neighbors (KNN)  
<Figure size 800x800 with 0 Axes>

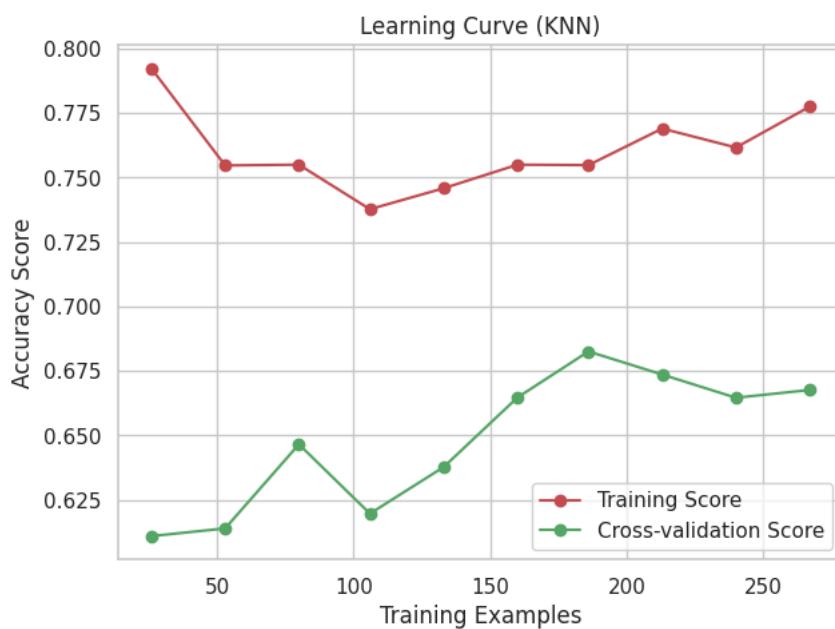


Classification Report:				
	precision	recall	f1-score	support
C	0.53	0.38	0.44	21
Q	0.73	0.89	0.80	9
S	0.78	0.83	0.80	54
accuracy			0.73	84
macro avg	0.68	0.70	0.68	84
weighted avg	0.71	0.73	0.71	84

```

1 # Learning Curve Plot
2 train_sizes, train_scores, test_scores = learning_curve(
3     knn, X_train, y_train, cv=5, n_jobs=-1,
4     train_sizes=np.linspace(0.1, 1.0, 10), scoring='accuracy'
5 )
6 train_scores_mean = np.mean(train_scores, axis=1)
7 test_scores_mean = np.mean(test_scores, axis=1)
8 plt.figure(figsize=(7, 5))
9 plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training Score")
10 plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation Score")
11 plt.xlabel("Training Examples")
12 plt.ylabel("Accuracy Score")
13 plt.title("Learning Curve (KNN)")
14 plt.legend(loc="best")
15 plt.savefig('Learning_Curve_KNN.png')
16 plt.show()
17

```



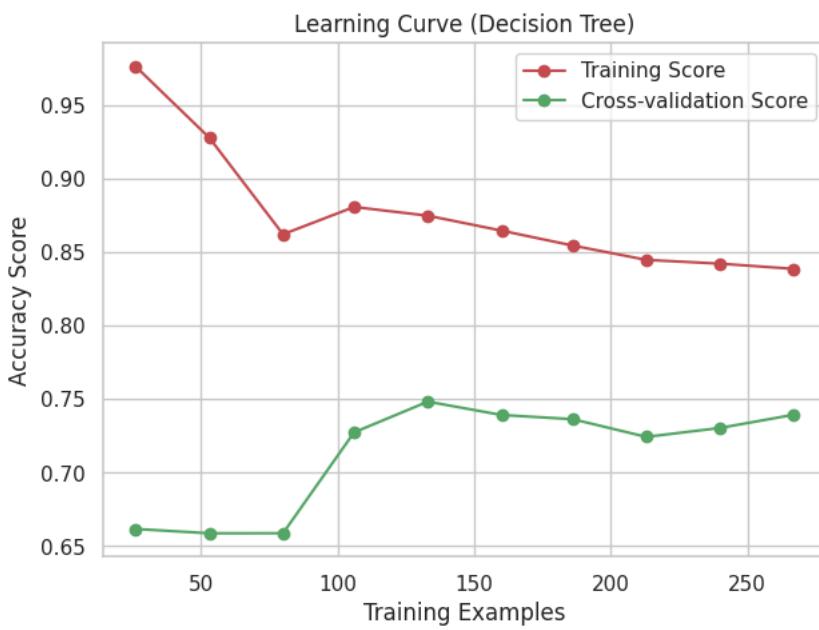
## Decision Tree

```
1 # Model 3 - Decision Tree Classifier (with CM Plot)
2 print("Model 3: Decision Tree Classifier")
3 dt_clf = DecisionTreeClassifier(max_depth=5, random_state=42)
4 dt_clf.fit(X_train, y_train)
5 y_pred_dt = dt_clf.predict(X_test)
6
7 # Confusion Matrix PLOT
8 cm_dt = confusion_matrix(y_test, y_pred_dt)
9 disp_dt = ConfusionMatrixDisplay(confusion_matrix=cm_dt, display_labels=target_classes)
10 plt.figure(figsize=(8, 8))
11 disp_dt.plot(cmap=plt.cm.Blues, values_format='d')
12 plt.title('Confusion Matrix - Decision Tree')
13 plt.savefig('CM_DecisionTree.png')
14 plt.show()
15 # Classification Report
16
17 print("\nClassification Report:")
18 print(classification_report(y_test, y_pred_dt, target_names=target_classes))
```

```
Model 3: Decision Tree Classifier
<Figure size 800x800 with 0 Axes>
```

Confusion Matrix - Decision Tree	
	C
C	13
	0
	8
	- 40

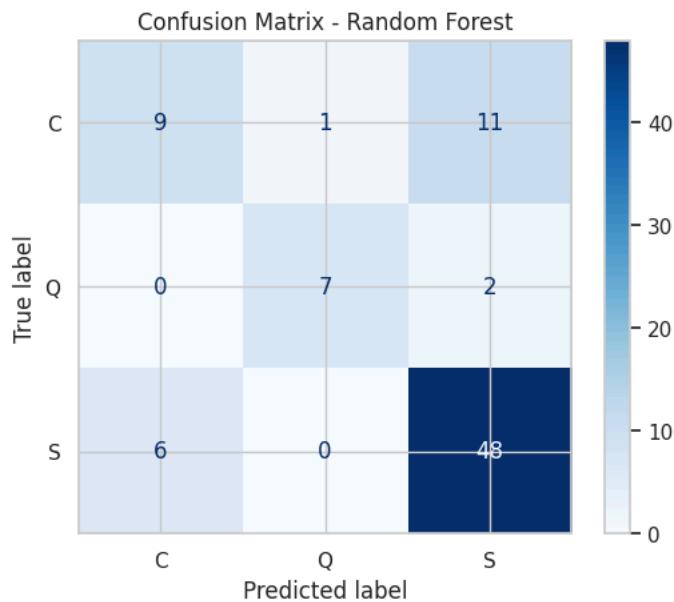
```
1 # Learning Curve Plot
2 train_sizes, train_scores, test_scores = learning_curve(
3     dt_clf, X_train, y_train, cv=5, n_jobs=-1,
4     train_sizes=np.linspace(0.1, 1.0, 10), scoring='accuracy'
5 )
6 train_scores_mean = np.mean(train_scores, axis=1)
7 test_scores_mean = np.mean(test_scores, axis=1)
8 plt.figure(figsize=(7, 5))
9 plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training Score")
10 plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation Score")
11 plt.xlabel("Training Examples")
12 plt.ylabel("Accuracy Score")
13 plt.title("Learning Curve (Decision Tree)")
14 plt.legend(loc="best")
15 plt.savefig('Learning_Curve_DecisionTree.png')
16 plt.show()
```



## Random Forest Classifier

```
1 # Model 4 - Random Forest Classifier (with CM Plot)
2 print(" Model 4: Random Forest Classifier ")
3 rf_clf = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)
4 rf_clf.fit(X_train, y_train)
5 y_pred_rf = rf_clf.predict(X_test)
6 # Confusion Matrix PLOT
7 cm_rf = confusion_matrix(y_test, y_pred_rf)
8 disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=target_classes)
9 plt.figure(figsize=(8, 8))
10 disp_rf.plot(cmap=plt.cm.Blues, values_format='d')
11 plt.title('Confusion Matrix - Random Forest')
12 plt.savefig('CM_RandomForest.png')
13 plt.show()
14 # Classification Report
15 print("\nClassification Report:")
16 print(classification_report(y_test, y_pred_rf, target_names=target_classes))
```

Model 4: Random Forest Classifier  
<Figure size 800x800 with 0 Axes>



Classification Report:

	precision	recall	f1-score	support
C	0.60	0.43	0.50	21
Q	0.88	0.78	0.82	9
S	0.79	0.89	0.83	54
accuracy			0.76	84
macro avg	0.75	0.70	0.72	84
weighted avg	0.75	0.76	0.75	84

## Support Vector Machine

```

1 # Model 5 - Support Vector Machine (SVM) (with CM Plot)
2
3 print("Model 5: Support Vector Machine (SVM)")
4 svm_clf = SVC(random_state=42, decision_function_shape='ovr')
5 svm_clf.fit(X_train, y_train)
6 y_pred_svm = svm_clf.predict(X_test)
7
8 # Confusion Matrix PLOT
9 cm_svm = confusion_matrix(y_test, y_pred_svm)
10 disp_svm = ConfusionMatrixDisplay(confusion_matrix=cm_svm, display_labels=target_classes)
11 plt.figure(figsize=(8, 8))
12 disp_svm.plot(cmap=plt.cm.Blues, values_format='d')
13 plt.title('Confusion Matrix - SVM')
14 plt.savefig('CM_SVM.png')
15 plt.show()
16
17 # Classification Report
18 print("\nClassification Report:")
19 print(classification_report(y_test, y_pred_svm, target_names=target_classes))

```

Model 5: Support Vector Machine (SVM)  
<Figure size 800x800 with 0 Axes>

