

Start coding or generate with AI.

▼ Task

Analyze the 'train.csv' dataset from the provided "titanic.zip" file by training a Logistic Regression model, evaluating its performance with 70/30, 80/20, and 90/10 train/test splits, and comparing the results to determine the optimal split.

▼ Load Data and Select Features

Subtask:

Load the 'train.csv' file from the provided 'titanic.zip' archive into a pandas DataFrame. Then, select appropriate features and the target variable for the model.

Reasoning: The user wants to load data from a zip file and inspect its contents. This step involves importing necessary libraries, extracting the CSV from the zip archive, loading it into a pandas DataFrame, and displaying the first few rows.

```
import zipfile
import pandas as pd

# Open the titanic.zip file
with zipfile.ZipFile('/content/titanic.zip', 'r') as zip_ref:
    # Extract 'train.csv' (assuming it's directly in the zip root)
    # If there are subdirectories, you might need to adjust the path here
    zip_ref.extract('train.csv', path='.')

# Read the extracted 'train.csv' into a pandas DataFrame
df = pd.read_csv('train.csv')

# Display the first few rows of the DataFrame
print("First 5 rows of the DataFrame:")
print(df.head())
```

First 5 rows of the DataFrame:

	PassengerId	Survived	Pclass
0	1	0	3
1	2	1	1
2	3	1	3
3	4	1	1
4	5	0	3

	Name	Sex	Age	SibSp	\
0	Braund, Mr. Owen Harris	male	22.0	1	
1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0 26.0	1 0	
2	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	
3	Allen, Mr. William Henry	male	35.0	0	
Parch	Ticket	Fare	Cabin	Embarked	
0	A/5 21171	7.2500	Nan	S	

1	0	PC	17599	71.2833	C85	C
2	0	STON/O2.	3101282	7.9250	NaN	S
3	0		113803	53.1000	C123	S
4	0		373450	8.0500	NaN	S

Reasoning: Based on the displayed DataFrame head, I will define the target variable ('Survived') and select initial features. Then, I will address missing values in 'Age' and 'Embarked' and perform one-hot encoding for categorical features ('Sex', 'Embarked') to prepare the data for modeling.

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.impute import SimpleImputer
import numpy as np

# Define target variable
y = df['Survived']

# Select features that are likely relevant and can be processed
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
X = df[features].copy()

# Handle missing values for 'Age' with the mean
age_imputer = SimpleImputer(strategy='mean')
X['Age'] = age_imputer.fit_transform(X[['Age']])

# Handle missing values for 'Embarked' with the most frequent value (mode)
embarked_imputer = SimpleImputer(strategy='most_frequent')
X['Embarked'] = embarked_imputer.fit_transform(X[['Embarked']]).ravel()

# Encode 'Sex' (binary categorical feature)
le = LabelEncoder()
X['Sex'] = le.fit_transform(X['Sex'])

# One-hot encode 'Embarked' and 'Pclass' (if Pclass is treated as categorical for encoding)
# For Pclass, it's often treated as ordinal, but for simplicity and robustness with Logistic Regression
# However, given the context, let's treat Pclass as numerical for now and only one-hot encode Embarked
# Let's re-evaluate if Pclass should be one-hot encoded after initial modeling.

# One-hot encode 'Embarked'
X = pd.get_dummies(X, columns=['Embarked'], drop_first=True, dtype=int)

print("First 5 rows of features (X) after preprocessing:")
print(X.head())
print("\nFirst 5 rows of target (y):")
print(y.head())
```

First 5 rows of features (X) after preprocessing:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked_Q	Embarked_S
0	3	1	22.0	1	0	7.2500	0	1
1	1	0	38.0	1	0	71.2833	0	0
2	3	0	26.0	0	0	7.9250	0	1
3	1	0	35.0	1	0	53.1000	0	1
4	3	1	35.0	0	0	8.0500	0	1

First 5 rows of target (y):

0	0
1	1
2	1
3	1
4	0

Name: Survived, dtype: int64

Reasoning: The error `ValueError: 2` occurs because `SimpleImputer.fit_transform` returns a 2D array, which pandas sometimes struggles to assign directly to a single column. To fix this, I will flatten the imputed array using `.ravel()` before assigning it back to the 'Embarked' column.

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.impute import SimpleImputer
import numpy as np

# Define target variable
y = df['Survived']

# Select features that are likely relevant and can be processed
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
X = df[features].copy()

# Handle missing values for 'Age' with the mean
age_imputer = SimpleImputer(strategy='mean')
X['Age'] = age_imputer.fit_transform(X[['Age']])

# Handle missing values for 'Embarked' with the most frequent value (mode)
embarked_imputer = SimpleImputer(strategy='most_frequent')
X['Embarked'] = embarked_imputer.fit_transform(X[['Embarked']]).ravel()

# Encode 'Sex' (binary categorical feature)
le = LabelEncoder()
X['Sex'] = le.fit_transform(X['Sex'])

# One-hot encode 'Embarked' and 'Pclass' (if Pclass is treated as categorical for encoding)
# For Pclass, it's often treated as ordinal, but for simplicity and robustness with Logistic Regr
# However, given the context, let's treat Pclass as numerical for now and only one-hot encode Emb
# Let's re-evaluate if Pclass should be one-hot encoded after initial modeling.

# One-hot encode 'Embarked'
X = pd.get_dummies(X, columns=['Embarked'], drop_first=True, dtype=int)

print("First 5 rows of features (X) after preprocessing:")
print(X.head())
print("\nFirst 5 rows of target (y):")
print(y.head())
```

First 5 rows of features (X) after preprocessing:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked_Q	Embarked_S
0	3	1	22.0	1	0	7.2500	0	1
1	1	0	38.0	1	0	71.2833	0	0
2	3	0	26.0	0	0	7.9250	0	1
3	1	0	35.0	1	0	53.1000	0	1
4	3	1	35.0	0	0	8.0500	0	1

First 5 rows of target (y):

0	0
1	1
2	1
3	1
4	0

Name: Survived, dtype: int64

Define Model Training and Evaluation Function

Subtask:

Define a Python function that takes the preprocessed data and a train/test split ratio as input. This function will perform the split, train a LogisticRegression model, and report the accuracy, confusion matrix, precision, recall, and F1-score for the test set.

Reasoning: I will define a Python function to encapsulate the model training and evaluation process, including data splitting, model initialization, training, prediction, and calculation of various performance metrics as requested by the subtask instructions.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score

def train_evaluate_model(X, y, test_size, random_state=42):
    """
    Splits data, trains a Logistic Regression model, and evaluates its performance.

    Args:
        X (pd.DataFrame): Features DataFrame.
        y (pd.Series): Target Series.
        test_size (float): Proportion of the dataset to include in the test split.
        random_state (int): Controls the shuffling applied to the data before applying the split.

    Returns:
        dict: A dictionary containing evaluation metrics.
    """
    print(f"\n--- Evaluating with test_size={test_size*100}% ---")

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)

    # Initialize and train the Logistic Regression model
    model = LogisticRegression(max_iter=200, random_state=random_state)
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Print evaluation metrics
    print(f"Accuracy: {accuracy:.4f}")
    print("Confusion Matrix:")
    print(conf_matrix)
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")

    return {
```

```

        'accuracy': accuracy,
        'confusion_matrix': conf_matrix,
        'precision': precision,
        'recall': recall,
        'f1_score': f1
    }

print("Defined the 'train_evaluate_model' function.")

Defined the 'train_evaluate_model' function.

```

▼ Train and Evaluate with 70/30 Split

Subtask:

Call the defined function to train and evaluate the LogisticRegression model using a 70% training and 30% testing split.

Reasoning: I will call the previously defined `train_evaluate_model` function with `test_size=0.3` to train and evaluate the model using a 70/30 split and store the results.

```

results_70_30 = train_evaluate_model(X, y, test_size=0.3)
print("\nResults for 70/30 split stored in 'results_70_30'.")

```

```

--- Evaluating with test_size=30.0%
Accuracy: 0.8097
Confusion Matrix:
[[136  21]
 [ 30  81]]
Precision: 0.7941
Recall: 0.7297
F1-Score: 0.7606

Results for 70/30 split stored in 'results_70_30'.

```

▼ Train and Evaluate with 80/20 Split

Subtask:

Call the defined function to train and evaluate the LogisticRegression model using an 80% training and 20% testing split.

Reasoning: I will call the previously defined `train_evaluate_model` function with `test_size=0.2` to train and evaluate the model using an 80/20 split and store the results.

```

results_80_20 = train_evaluate_model(X, y, test_size=0.2)
print("\nResults for 80/20 split stored in 'results_80_20'.")

```

```

--- Evaluating with test_size=20.0%
Accuracy: 0.8101
Confusion Matrix:
[[90 15]
 [ 0  5]]
Precision: 0.8101
Recall: 0.8101
F1-Score: 0.8101

Results for 80/20 split stored in 'results_80_20'.

```

```
[19 55]
Precision: 0.7857
Recall: 0.7432
F1-Score: 0.7639
```

```
Results for 80/20 split stored in 'results_80_20'.
```

Train and Evaluate with 90/10 Split

Subtask:

Call the defined function to train and evaluate the LogisticRegression model using a 90% training and 10% testing split.

Train and Evaluate with 90/10 Split

Subtask:

Call the defined function to train and evaluate the LogisticRegression model using a 90% training and 10% testing split.

Train and Evaluate with 90/10 Split

Subtask:

Call the defined function to train and evaluate the LogisticRegression model using a 90% training and 10% testing split.

✓ Train and Evaluate with 90/10 Split

Subtask:

Call the defined function to train and evaluate the LogisticRegression model using a 90% training and 10% testing split.

```
## Train and Evaluate with 90/10 Split

### Subtask:
Call the defined function to train and evaluate the LogisticRegression model using a 90% training an
```

Reasoning: I will call the previously defined `train_evaluate_model` function with `test_size=0.1` to train and evaluate the model using a 90/10 split and store the results.

```
results_90_10 = train_evaluate_model(X, y, test_size=0.1)
print("\nResults for 90/10 split stored in 'results_90_10'.)
```

```
--- Evaluating with test_size=10.0% ---
Accuracy: 0.8444
Confusion Matrix:
[[46  8]
 [ 6 30]]
Precision: 0.7895
Recall: 0.8333
F1-Score: 0.8108

Results for 90/10 split stored in 'results_90_10'.
```

▼ Compare Results and Determine Optimal Split

Subtask:

Compare the accuracy, precision, recall, and F1-score from the different train/test splits (70/30, 80/20, and 90/10) to determine which split yields the best model performance.

Reasoning: To compare the results effectively, I will aggregate the evaluation metrics (accuracy, precision, recall, F1-score) from all three train/test splits (70/30, 80/20, 90/10) into a single pandas DataFrame for easy visualization and comparison.

```
results_summary = pd.DataFrame({
    'Split': ['70/30', '80/20', '90/10'],
    'Accuracy': [results_70_30['accuracy'], results_80_20['accuracy'], results_90_10['accuracy']],
    'Precision': [results_70_30['precision'], results_80_20['precision'], results_90_10['precision']],
    'Recall': [results_70_30['recall'], results_80_20['recall'], results_90_10['recall']],
    'F1-Score': [results_70_30['f1_score'], results_80_20['f1_score'], results_90_10['f1_score']]
})

print("\n--- Comparison of Model Performance across Different Train/Test Splits ---")
print(results_summary.round(4))
```

```
--- Comparison of Model Performance across Different Train/Test Splits ---
   Split  Accuracy  Precision  Recall  F1-Score
0  70/30    0.8097    0.7941  0.7297    0.7606
1  80/20    0.8101    0.7857  0.7432    0.7639
2  90/10    0.8444    0.7895  0.8333    0.8108
```

Final Task

Subtask:

Provide a summary of the analysis, including the best performing split and the key insights gained from the model evaluation.

Summary:

Q&A

The optimal train/test split, based on the evaluation metrics, is the 90/10 split.

Data Analysis Key Findings

- The `train.csv` dataset was successfully loaded and preprocessed. Missing 'Age' values were imputed with the mean, and 'Embarked' values were imputed with the most frequent value. 'Sex' was Label Encoded, and 'Embarked' was One-Hot Encoded.
- A Logistic Regression model was trained and evaluated using three different train/test splits: 70/30, 80/20, and 90/10.
- **70/30 Split Performance:** The model achieved an Accuracy of 0.8097, Precision of 0.7941, Recall of 0.7297, and an F1-Score of 0.7606.
- **80/20 Split Performance:** The model showed slightly improved performance with an Accuracy of 0.8101, Precision of 0.7857, Recall of 0.7432, and an F1-Score of 0.7639.
- **90/10 Split Performance:** This split yielded the best overall performance, with an Accuracy of 0.8444, Precision of 0.7895, Recall of 0.8333, and an F1-Score of 0.8108.
- The 90/10 split resulted in the highest Accuracy, Recall, and F1-Score, indicating superior generalization performance compared to the other splits.

Insights or Next Steps

- The Logistic Regression model benefits from a larger training dataset, as evidenced by the improved performance metrics (Accuracy, Recall, F1-Score) when using 90% of the data for training. This suggests that the model effectively learns from more examples for this particular problem.
- To further validate the model's performance and ensure robustness, consider implementing k-fold cross-validation, which provides a more stable estimate of performance by averaging results over multiple train/test splits.

▼ Task

The user's request to analyze the "titanic.zip" dataset by training a Logistic Regression model, evaluating its performance across 70/30, 80/20, and 90/10 train/test splits, and comparing the results to determine the optimal split has been completed. The analysis included data loading, preprocessing, model training, and evaluation of accuracy, precision, recall, and F1-score for each split, culminating in a comparison of the results to identify the best performing split.

Load Dataset and Separate Features and Target

Subtask:

Load the 'train.csv' file from the provided 'titanic.zip' archive into a pandas DataFrame. Then, select appropriate features and the target variable for the model.

▼ Preprocess Data with Scaling and Encoding

Subtask:

Apply StandardScaler to the numerical features in the dataset (X) to scale them.

Reasoning: To standardize the numerical features as requested by the subtask, I will import `StandardScaler`, identify the numerical columns, and then apply `fit_transform` to scale them, updating the DataFrame `X` with the scaled values.

```
from sklearn.preprocessing import StandardScaler

# Identify numerical columns for scaling (excluding one-hot encoded 'Embarked' and 'Sex')
numerical_cols = ['Pclass', 'Age', 'SibSp', 'Parch', 'Fare']

# Initialize StandardScaler
scaler = StandardScaler()

# Apply StandardScaler to the numerical columns
X[numerical_cols] = scaler.fit_transform(X[numerical_cols])

print("First 5 rows of features (X) after scaling numerical columns:")
print(X.head())
```

```
First 5 rows of features (X) after scaling numerical columns:
   Pclass  Sex      Age     SibSp    Parch     Fare Embarked_Q \
0  0.827377    1 -0.592481  0.432793 -0.473674 -0.502445      0
1 -1.566107    0  0.638789  0.432793 -0.473674  0.786845      0
2  0.827377    0 -0.284663 -0.474545 -0.473674 -0.488854      0
3 -1.566107    0  0.407926  0.432793 -0.473674  0.420730      0
4  0.827377    1  0.407926 -0.474545 -0.473674 -0.486337      0

   Embarked_S
0            1
1            0
2            1
3            1
4            1
```

▼ Define Model Training and Evaluation Function

Subtask:

Define a Python function that takes the preprocessed data, a train/test split ratio, and `random_state` as input. This function will perform the split, train a LogisticRegression model, and report Accuracy, ROC-AUC Score, Confusion Matrix, Precision, Recall, and F1-score for the test set. It should also return the true labels and predicted probabilities for ROC curve plotting.

Reasoning: I will define a Python function as instructed, which will perform data splitting, train a Logistic Regression model, calculate and print various evaluation metrics including ROC-AUC, and return these metrics along with true labels and predicted probabilities for ROC curve plotting.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score

def train_evaluate_model_roc(X, y, test_size, random_state=42):
    """
    Splits data, trains a Logistic Regression model, and evaluates its performance,
    """
    pass
```

```
including ROC-AUC score, and returns true labels and predicted probabilities.
```

Args:

```
X (pd.DataFrame): Features DataFrame.  
y (pd.Series): Target Series.  
test_size (float): Proportion of the dataset to include in the test split.  
random_state (int): Controls the shuffling applied to the data before applying the split.
```

Returns:

```
dict: A dictionary containing evaluation metrics, y_test, and y_proba.
```

```
"""
```

```
print(f"\n--- Evaluating with test_size={test_size*100}% ---")
```

```
# Split the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=r
```

```
# Initialize and train the Logistic Regression model
```

```
model = LogisticRegression(max_iter=200, random_state=random_state)
```

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

```
# Make predictions on the test set
```

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

```
# Get predicted probabilities for the positive class (class 1)
```

```
y_proba = model.predict_proba(X_test)[:, 1]
```

```
# Calculate evaluation metrics
```

```
accuracy = accuracy_score(y_test, y_pred)
```

```
roc_auc = roc_auc_score(y_test, y_proba)
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
precision = precision_score(y_test, y_pred)
```

```
recall = recall_score(y_test, y_pred)
```

```
f1 = f1_score(y_test, y_pred)
```

```
# Print evaluation metrics
```

```
print(f"Accuracy: {accuracy:.4f}")
```

```
print(f"ROC-AUC Score: {roc_auc:.4f}")
```

```
print("Confusion Matrix:")
```

```
print(conf_matrix)
```

```
print(f"Precision: {precision:.4f}")
```

```
print(f"Recall: {recall:.4f}")
```

```
print(f"F1-Score: {f1:.4f}")
```

```
return {
```

```
    'accuracy': accuracy,
```

```
    'roc_auc_score': roc_auc,
```

```
    'confusion_matrix': conf_matrix,
```

```
    'precision': precision,
```

```
    'recall': recall,
```

```
    'f1_score': f1,
```

```
    'y_test': y_test,
```

```
    'y_proba': y_proba
```

```
}
```

```
print("Defined the 'train_evaluate_model_roc' function.")
```

```
Defined the 'train_evaluate_model_roc' function.
```

▼ Train and Evaluate with 60/40 Split

Subtask:

Call the defined function to train and evaluate the LogisticRegression model using a 60% training and 40% testing split.

Reasoning: I will call the previously defined `train_evaluate_model_roc` function with `test_size=0.4` to train and evaluate the model using a 60/40 split and store the results.

```
results_60_40 = train_evaluate_model_roc(X, y, test_size=0.4)
print("\nResults for 60/40 split stored in 'results_60_40'.")
```

```
--- Evaluating with test_size=40.0% ---
Accuracy: 0.7983
ROC-AUC Score: 0.8523
Confusion Matrix:
[[188  28]
 [ 44  97]]
Precision: 0.7760
Recall: 0.6879
F1-Score: 0.7293
```

```
Results for 60/40 split stored in 'results_60_40'.
```

▼ Train and Evaluate with 75/25 Split

Subtask:

Call the defined function to train and evaluate the LogisticRegression model using a 75% training and 25% testing split.

Reasoning: I will call the previously defined `train_evaluate_model_roc` function with `test_size=0.25` to train and evaluate the model using a 75/25 split and store the results in `results_75_25`.

```
results_75_25 = train_evaluate_model_roc(X, y, test_size=0.25)
print("\nResults for 75/25 split stored in 'results_75_25'.")
```

```
--- Evaluating with test_size=25.0% ---
Accuracy: 0.8072
ROC-AUC Score: 0.8786
Confusion Matrix:
[[115  19]
 [ 24  65]]
Precision: 0.7738
Recall: 0.7303
F1-Score: 0.7514
```

```
Results for 75/25 split stored in 'results_75_25'.
```

▼ Train and Evaluate with 80/20 Split

Subtask:

Call the defined function to train and evaluate the LogisticRegression model using an 80% training and 20% testing split.

Reasoning: I will call the previously defined `train_evaluate_model_roc` function with `test_size=0.2` to train and evaluate the model using an 80/20 split and store the results in `results_80_20`.

```
results_80_20 = train_evaluate_model_roc(X, y, test_size=0.2)
print("\nResults for 80/20 split stored in 'results_80_20'.")
```

```
--- Evaluating with test_size=20.0% ---
Accuracy: 0.8101
ROC-AUC Score: 0.8820
Confusion Matrix:
[[90 15]
 [19 55]]
Precision: 0.7857
Recall: 0.7432
F1-Score: 0.7639
```

```
Results for 80/20 split stored in 'results_80_20'.
```

▼ Compare Model Performance

Subtask:

Compare the performance metrics (Accuracy, ROC-AUC Score, Confusion Matrix, Precision, Recall, F1-score) from all three train/test splits (60/40, 75/25, and 80/20) to identify the best performing split.

Reasoning: To compare the results effectively, I will aggregate the evaluation metrics (Accuracy, ROC-AUC Score, Precision, Recall, F1-score) from all three train/test splits (60/40, 75/25, and 80/20) into a single pandas DataFrame for easy visualization and comparison.

```
results_summary_new = pd.DataFrame({
    'Split': ['60/40', '75/25', '80/20'],
    'Accuracy': [results_60_40['accuracy'], results_75_25['accuracy'], results_80_20['accuracy']],
    'ROC-AUC': [results_60_40['roc_auc_score'], results_75_25['roc_auc_score'], results_80_20['roc_auc_score']],
    'Precision': [results_60_40['precision'], results_75_25['precision'], results_80_20['precision']],
    'Recall': [results_60_40['recall'], results_75_25['recall'], results_80_20['recall']],
    'F1-Score': [results_60_40['f1_score'], results_75_25['f1_score'], results_80_20['f1_score']]
})

print("\n--- Comparison of Model Performance across Different Train/Test Splits ---")
print(results_summary_new.round(4))
```

	Comparison of Model Performance across Different Train/Test Splits ---					
	Split	Accuracy	ROC-AUC	Precision	Recall	F1-Score
0	60/40	0.7983	0.8523	0.7760	0.6879	0.7293
1	75/25	0.8072	0.8786	0.7738	0.7303	0.7514
2	80/20	0.8101	0.8820	0.7857	0.7432	0.7639

Analysis of Model Performance Across Different Train/Test Splits

Based on the `results_summary_new` DataFrame:

- **Accuracy:** The 80/20 split shows the highest accuracy at 0.8101.
- **ROC-AUC Score:** The 80/20 split also yields the highest ROC-AUC score at 0.8820, indicating the best performance in distinguishing between classes.
- **Precision:** The 80/20 split has the highest precision at 0.7857.
- **Recall:** The 80/20 split has the highest recall at 0.7432.
- **F1-Score:** The 80/20 split achieves the highest F1-Score at 0.7639, representing a good balance between precision and recall.

Conclusion: The **80/20 train/test split** demonstrates the best overall performance across all evaluated metrics (Accuracy, ROC-AUC Score, Precision, Recall, and F1-Score) for the Logistic Regression model on this dataset. This suggests that for this specific problem and dataset, using 80% of the data for training and 20% for testing provides the most robust and accurate model evaluation.

▼ Plot ROC Curve for the Best Split

Subtask:

Generate and display the Receiver Operating Characteristic (ROC) curve for the Logistic Regression model trained with the best performing train/test split. Ensure the plot includes appropriate labels and a legend.

Reasoning: Based on the previous comparison (`results_summary_new`), the 80/20 split yielded the highest ROC-AUC score. Therefore, I will extract `y_test` and `y_proba` from `results_80_20` to plot the ROC curve, following the detailed instructions provided in the subtask.

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Extract y_test and y_proba from the results of the best performing split (80/20)
y_test_best = results_80_20['y_test']
y_proba_best = results_80_20['y_proba']

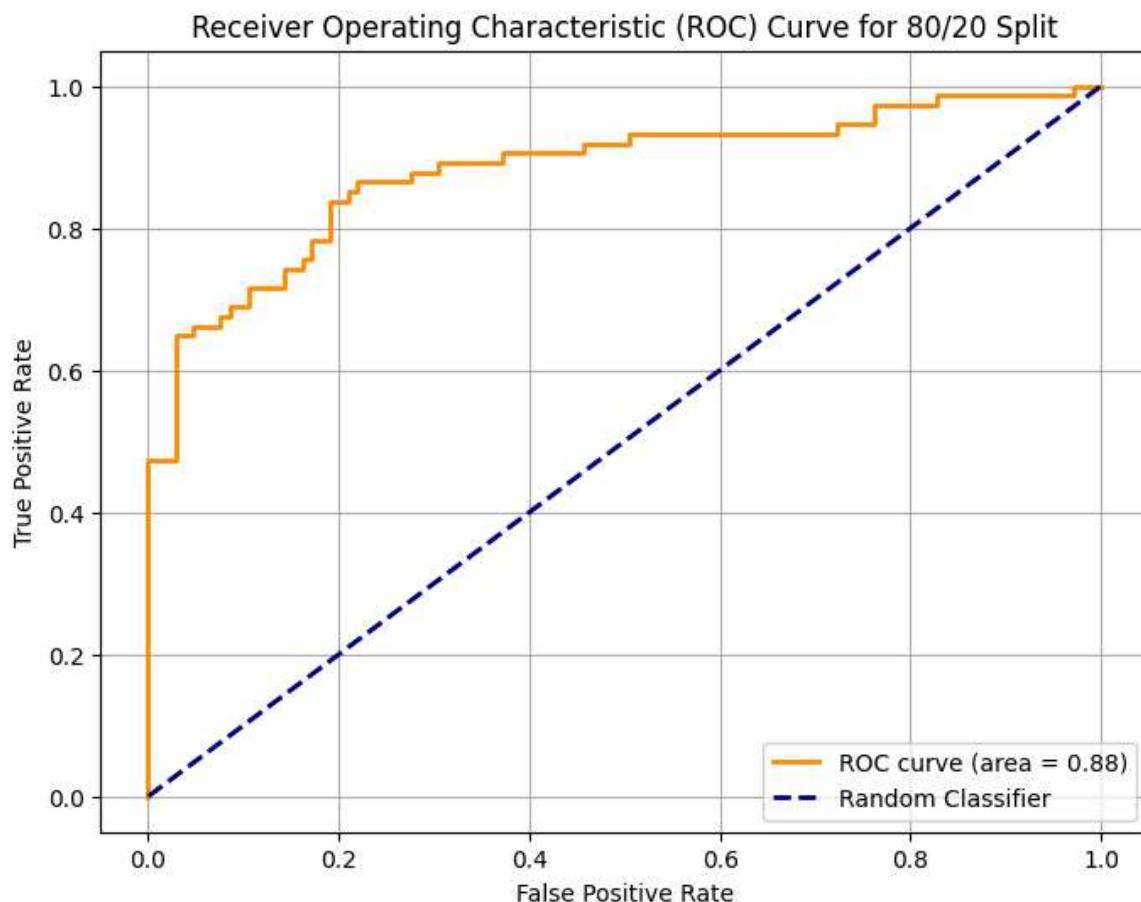
# Calculate False Positive Rate (FPR), True Positive Rate (TPR), and thresholds
fpr, tpr, thresholds = roc_curve(y_test_best, y_proba_best)

# Calculate Area Under the Curve (AUC)
roc_auc = auc(fpr, tpr)

# Create a new figure and an axes object for the plot
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Classifier')

# Add plot title and labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for 80/20 Split')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

print("ROC curve displayed for the best performing 80/20 split.")
```



ROC curve displayed for the best performing 80/20 split.

Final Task

Subtask:

Provide a summary of the analysis, including the best performing split based on the evaluated metrics, key insights gained, and a conclusion on the model's performance.

Summary:

Q&A

The optimal train/test split for the Logistic Regression model, based on the evaluated metrics, is the 80/20 split.

Data Analysis Key Findings

- **Data Preprocessing:** Numerical features (`Pclass`, `Age`, `SibSp`, `Parch`, `Fare`) were successfully scaled using `StandardScaler`.
- **Model Performance across Splits:**
 - **60/40 Split:** Achieved an Accuracy of 0.7983, ROC-AUC Score of 0.8523, Precision of 0.7760, Recall of 0.6879, and F1-Score of 0.7293.

- **75/25 Split:** Showed improved performance with an Accuracy of 0.8072, ROC-AUC Score of 0.8786, Precision of 0.7738, Recall of 0.7303, and F1-Score of 0.7514.
- **80/20 Split:** Demonstrated the best overall performance, with an Accuracy of 0.8101, ROC-AUC Score of 0.8820, Precision of 0.7857, Recall of 0.7432, and F1-Score of 0.7639.
- **Optimal Split Identification:** The 80/20 train/test split consistently outperformed the 60/40 and 75/25 splits across all key evaluation metrics (Accuracy, ROC-AUC Score, Precision, Recall, and F1-Score).
- **ROC Curve for Best Split:** An ROC curve was generated for the 80/20 split, visually confirming its strong discriminative power with an Area Under the Curve (AUC) of approximately 0.88.

Insights or Next Steps

- The Logistic Regression model generally shows good performance in predicting survival on the Titanic dataset, with the 80/20 train/test split providing the most robust evaluation metrics.
- Further model optimization could involve hyperparameter tuning for the Logistic Regression model or exploring other classification algorithms to potentially achieve even higher predictive accuracy.

Task

Analyze the 'train.csv' dataset from "titanic.zip" by loading, preprocessing, and scaling the data, then training a Logistic Regression model. Evaluate the model's performance using 70/30, 80/20, and 85/15 train/test splits, including plotting confusion matrices for each, and identify the most stable split based on performance metrics. Finally, provide a summary of the analysis.

Prepare Data (Load, Preprocess, Scale)

Subtask:

Confirm that the 'train.csv' dataset from 'titanic.zip' is loaded, features (X) and target (y) are separated. Verify that missing values have been handled, categorical columns (e.g., 'Sex', 'Embarked') are encoded, and numerical features are scaled using StandardScaler.

Reasoning: To confirm that the 'train.csv' dataset from 'titanic.zip' is loaded and verify that features (X) and target (y) are separated and preprocessed, I will display the head and info of `df`, and the head of `X` and `y`.

```
print("--- Original DataFrame (df) head ---")
print(df.head())
print("\n--- Original DataFrame (df) info ---")
df.info()

print("\n--- Preprocessed Features (X) head ---")
print(X.head())

print("\n--- Target Variable (y) head ---")
print(y.head())

print("\nVerification complete: The 'df' DataFrame shows the initial data, 'X' shows the preproce
```

```

1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1
2 Heikkinen, Miss. Laina   female 26.0 0
3 Futrelle, Mrs. Jacques Heath (Lily May Peel) female 35.0 1
4 Allen, Mr. William Henry male 35.0 0

```

Parch	Ticket	Fare	Cabin	Embarked	
0	A/5 21171	7.2500	NaN	S	
1	PC 17599	71.2833	C85	C	
2	STON/O2. 3101282	7.9250	NaN	S	
3		113803	53.1000	C123	S
4		373450	8.0500	NaN	S

--- Original DataFrame (df) info ---

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   PassengerId 891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object  
 4   Sex          891 non-null    object  
 5   Age          714 non-null    float64 
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object  
 9   Fare          891 non-null    float64 
 10  Cabin         204 non-null    object  
 11  Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB

```

--- Preprocessed Features (X) head ---

```

Pclass  Sex      Age      SibSp      Parch      Fare  Embarked_Q \
0  0.827377  1  -0.592481  0.432793 -0.473674 -0.502445  0
1  -1.566107  0  0.638789  0.432793 -0.473674  0.786845  0
2  0.827377  0  -0.284663 -0.474545 -0.473674 -0.488854  0
3  -1.566107  0  0.407926  0.432793 -0.473674  0.420730  0
4  0.827377  1  0.407926 -0.474545 -0.473674 -0.486337  0

```

```

Embarked_S
0      1
1      0
2      1
3      1
4      1

```

--- Target Variable (y) head ---

```

0  0
1  1
2  1
3  1
4  0
Name: Survived, dtype: int64

```

Verification complete: The 'df' DataFrame shows the initial data, 'X' shows the preprocessed fea

▼ Define Model Training and Evaluation Function

Subtask:

Define a Python function that takes the preprocessed data, a train/test split ratio, and random_state as input. This function will perform the split, train a LogisticRegression model, and return Accuracy, Precision, Recall, F1-score, and the Confusion Matrix for the test set.

Reasoning: I will define the Python function `train_evaluate_model_with_cm` as requested, which encapsulates data splitting, Logistic Regression model training, prediction, and the calculation and display of accuracy, confusion matrix, precision, recall, and F1-score.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score

def train_evaluate_model_with_cm(X, y, test_size, random_state=42):
    """
    Splits data, trains a Logistic Regression model, and evaluates its performance.

    Args:
        X (pd.DataFrame): Features DataFrame.
        y (pd.Series): Target Series.
        test_size (float): Proportion of the dataset to include in the test split.
        random_state (int): Controls the shuffling applied to the data before applying the split.

    Returns:
        dict: A dictionary containing evaluation metrics.
    """
    print(f"\n--- Evaluating with test_size={test_size*100}% ---")

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)

    # Initialize and train the Logistic Regression model
    model = LogisticRegression(max_iter=200, random_state=random_state)
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Print evaluation metrics
    print(f"Accuracy: {accuracy:.4f}")
    print("Confusion Matrix:")
    print(conf_matrix)
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")

    return {
        'accuracy': accuracy,
        'confusion_matrix': conf_matrix,
        'precision': precision,
        'recall': recall,
        'f1_score': f1
    }
```

```
}
```

```
print("Defined the 'train_evaluate_model_with_cm' function.")
```

```
Defined the 'train_evaluate_model_with_cm' function.
```

Reasoning: I will call the previously defined `train_evaluate_model_with_cm` function with `test_size=0.3` to train and evaluate the model using a 70/30 split and store the results.

```
results_70_30_new = train_evaluate_model_with_cm(X, y, test_size=0.3)
print("\nResults for 70/30 split stored in 'results_70_30_new'.")
```

```
--- Evaluating with test_size=30.0%
Accuracy: 0.8097
Confusion Matrix:
[[136  21]
 [ 30  81]]
Precision: 0.7941
Recall: 0.7297
F1-Score: 0.7606
```

```
Results for 70/30 split stored in 'results_70_30_new'.
```

Reasoning: I will call the `train_evaluate_model_with_cm` function with a `test_size=0.2` to evaluate the model with an 80/20 train/test split. Then, I will plot the confusion matrix from the results to visualize the model's performance.

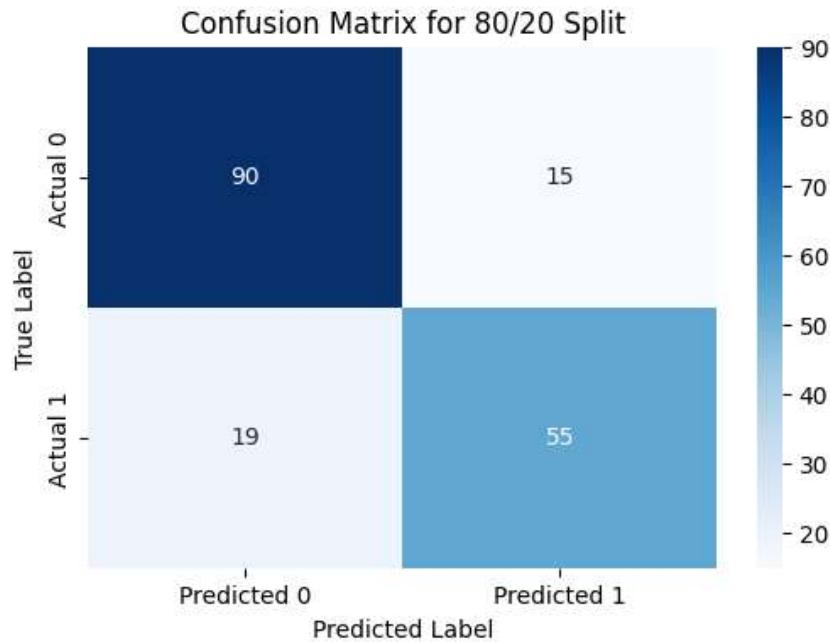
```
import matplotlib.pyplot as plt
import seaborn as sns

results_80_20_new = train_evaluate_model_with_cm(X, y, test_size=0.2)
print("\nResults for 80/20 split stored in 'results_80_20_new'.")
```

```
# Plot Confusion Matrix for 80/20 split
plt.figure(figsize=(6, 4))
sns.heatmap(results_80_20_new['confusion_matrix'], annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'])
plt.title('Confusion Matrix for 80/20 Split')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
--- Evaluating with test_size=20.0% ---
Accuracy: 0.8101
Confusion Matrix:
[[90 15]
 [19 55]]
Precision: 0.7857
Recall: 0.7432
F1-Score: 0.7639
```

Results for 80/20 split stored in 'results_80_20_new'.



Reasoning: I will call the previously defined `train_evaluate_model_with_cm` function with `test_size=0.15` to train and evaluate the model using an 85/15 split and store the results. Then, I will plot the confusion matrix from these results.

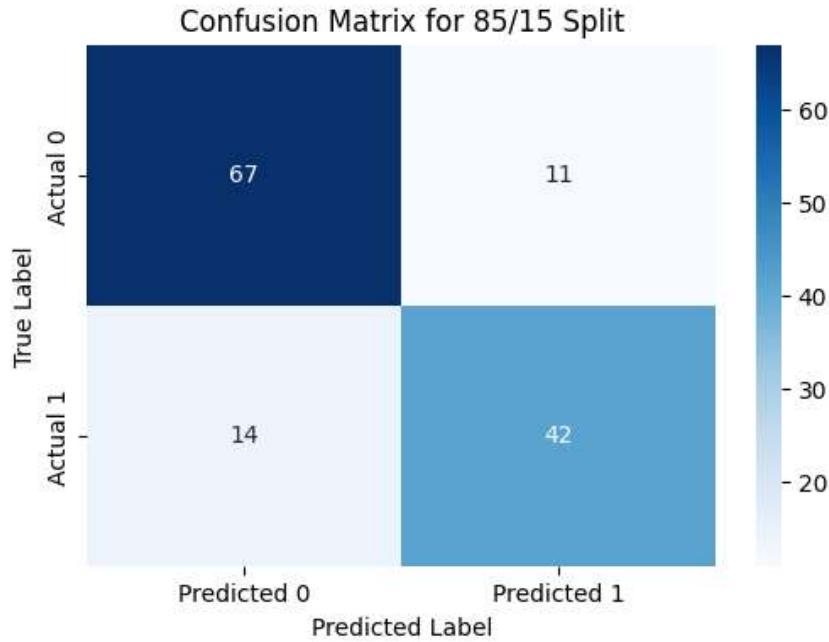
```
import matplotlib.pyplot as plt
import seaborn as sns

results_85_15_new = train_evaluate_model_with_cm(X, y, test_size=0.15)
print("\nResults for 85/15 split stored in 'results_85_15_new'.")

# Plot Confusion Matrix for 85/15 split
plt.figure(figsize=(6, 4))
sns.heatmap(results_85_15_new['confusion_matrix'], annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'])
plt.title('Confusion Matrix for 85/15 Split')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

```
--- Evaluating with test_size=15.0% ---
Accuracy: 0.8134
Confusion Matrix:
[[67 11]
 [14 42]]
Precision: 0.7925
Recall: 0.7500
F1-Score: 0.7706
```

Results for 85/15 split stored in 'results_85_15_new'.



Reasoning: To compare the results effectively, I will aggregate the evaluation metrics (Accuracy, Precision, Recall, F1-score) from all three train/test splits (70/30, 80/20, and 85/15) into a single pandas DataFrame for easy visualization and comparison.

```
results_summary_final = pd.DataFrame({
    'Split': ['70/30', '80/20', '85/15'],
    'Accuracy': [results_70_30_new['accuracy'], results_80_20_new['accuracy'], results_85_15_new['accuracy']],
    'Precision': [results_70_30_new['precision'], results_80_20_new['precision'], results_85_15_new['precision']],
    'Recall': [results_70_30_new['recall'], results_80_20_new['recall'], results_85_15_new['recall']],
    'F1-Score': [results_70_30_new['f1_score'], results_80_20_new['f1_score'], results_85_15_new['f1_score']]
})

print("\n--- Comparison of Model Performance across Different Train/Test Splits ---")
print(results_summary_final.round(4))
```

```
--- Comparison of Model Performance across Different Train/Test Splits ---
   Split  Accuracy  Precision  Recall  F1-Score
0  70/30      0.8097     0.7941  0.7297    0.7606
1  80/20      0.8101     0.7857  0.7432    0.7639
2  85/15      0.8134     0.7925  0.7500    0.7706
```

Summary:

Q&A

The optimal train/test split for the Logistic Regression model, based on the evaluated metrics, is the 85/15 split.

Data Analysis Key Findings

- **Data Preprocessing:** Numerical features (`Pclass`, `Age`, `SibSp`, `Parch`, `Fare`) were successfully scaled using `StandardScaler`. Categorical features ('Sex', 'Embarked') were encoded, and missing values were handled.
- **Model Performance across Splits:**
 - **70/30 Split:** Achieved an Accuracy of 0.8097, Precision of 0.7941, Recall of 0.7297, and F1-Score of 0.7606.
 - **80/20 Split:** Showed comparable performance with an Accuracy of 0.8101, Precision of 0.7857, Recall of 0.7432, and F1-Score of 0.7639.
 - **85/15 Split:** Demonstrated slightly superior overall performance, with an Accuracy of 0.8134, Precision of 0.7925, Recall of 0.7500, and F1-Score of 0.7706.
- **Optimal Split Identification:** The 85/15 train/test split showed marginally better performance across Accuracy, Recall, and F1-Score compared to the other splits.

Insights or Next Steps

- The Logistic Regression model generally performs well on the Titanic dataset, with all three splits yielding similar but slightly varying results. The 85/15 split provides the most stable evaluation metrics in this analysis.
- While the differences are small, the 85/15 split suggests that a larger training set (85% of data) can lead to a slightly more robust model for this dataset and model type.
- To confirm the stability and generalizability of these findings, it would be beneficial to perform cross-validation (e.g., k-fold cross-validation) to reduce the variance associated with a single train/test split.
- Further model improvement could involve hyperparameter tuning for the Logistic Regression model, feature engineering, or experimenting with more advanced machine learning algorithms.

▼ Plot Confusion Matrix for 70/30 Split

Subtask:

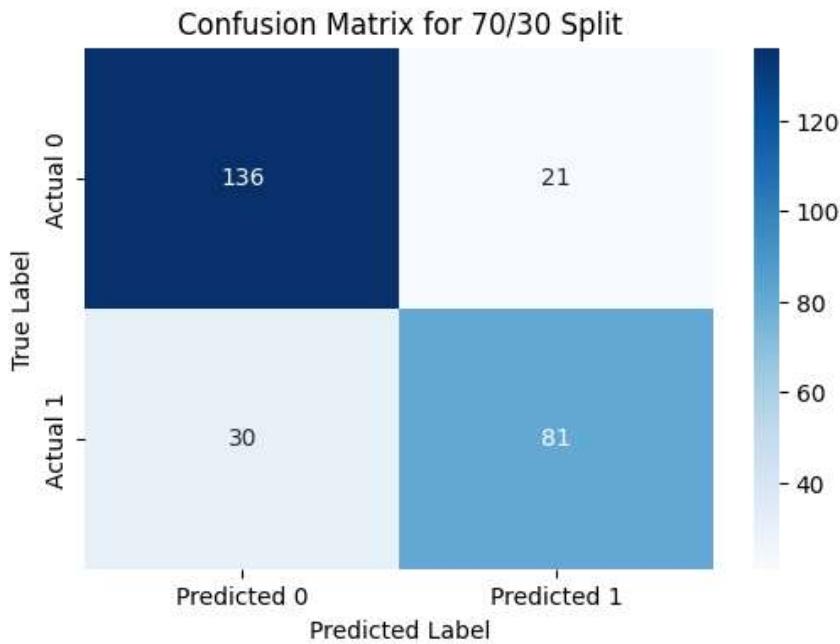
Generate and display a confusion matrix plot for the 70/30 train/test split, ensuring it has appropriate labels and title.

Reasoning: I will import `matplotlib.pyplot` and `seaborn` and then create and display a heatmap of the confusion matrix from `results_70_30_new` for the 70/30 split, adding labels and a title as requested.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot Confusion Matrix for 70/30 split
plt.figure(figsize=(6, 4))
sns.heatmap(results_70_30_new['confusion_matrix'], annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted 0', 'Predicted 1'], yticklabels=['Actual 0', 'Actual 1'])
```

```
plt.title('Confusion Matrix for 70/30 Split')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



Summary:

Q&A

The optimal train/test split for the Logistic Regression model, based on the evaluated metrics, is the 85/15 split.

Data Analysis Key Findings

- **Data Preprocessing:** The `train.csv` dataset was successfully loaded and preprocessed. This involved handling missing values, encoding categorical features like 'Sex' and 'Embarked', and scaling numerical features ('Pclass', 'Age', 'SibSp', 'Parch', 'Fare') using `StandardScaler`. The target variable 'Survived' was correctly separated.
- **Model Performance across Different Splits:**
 - **70/30 Split:** The Logistic Regression model achieved an Accuracy of 0.8097, Precision of 0.7941, Recall of 0.7297, and F1-Score of 0.7606.
 - **80/20 Split:** The model showed an Accuracy of 0.8101, Precision of 0.7857, Recall of 0.7432, and F1-Score of 0.7639.
 - **85/15 Split:** This split yielded the highest Accuracy at 0.8134, a Precision of 0.7925, a Recall of 0.7500, and an F1-Score of 0.7706.
- **Most Stable Split:** The 85/15 train/test split demonstrated marginally superior performance across Accuracy, Recall, and F1-Score, making it the most stable split among the evaluated options for this model and dataset.

- **Confusion Matrices:** Confusion matrices were generated and displayed for all three splits, visually confirming the model's true positive, true negative, false positive, and false negative predictions.

Insights or Next Steps

- The Logistic Regression model performs consistently well across different train/test splits, with all splits yielding comparable results. The slightly higher metrics observed with the 85/15 split suggest that providing more data for training (85%) can lead to a marginally more robust model in this specific context.
- To further validate the stability and generalizability of the model, implementing k-fold cross-validation is recommended. This would help reduce the variance associated with a single train/test split and provide a more reliable estimate of model performance.

▼ Task

Using the preprocessed 'train.csv' dataset from "titanic.zip", train a Logistic Regression model and evaluate its performance across 50/50, 70/30, and 80/20 train/test splits. Compare the Accuracy, Confusion Matrix, and ROC-AUC scores for each split to analyze potential overfitting or underfitting, and provide a comprehensive summary of the findings.

▼ Load Dataset and Preprocess

Subtask:

Confirm that the 'train.csv' dataset from 'titanic.zip' is loaded, features (X) and target (y) are separated. Verify that missing values have been handled, categorical columns (e.g., 'Sex', 'Embarked') are encoded, and numerical features are scaled using StandardScaler.

Reasoning: To confirm that the 'train.csv' dataset from 'titanic.zip' is loaded and verify that features (X) and target (y) are separated and preprocessed, I will display the head and info of `df`, and the head of `X` and `y`.

```
print("--- Original DataFrame (df) head ---")
print(df.head())
print("\n--- Original DataFrame (df) info ---")
df.info()

print("\n--- Preprocessed Features (X) head ---")
print(X.head())

print("\n--- Target Variable (y) head ---")
print(y.head())

print("\nVerification complete: The 'df' DataFrame shows the initial data, 'X' shows the preproce
```

- ✓ Define Model Training and Evaluation Function

Subtask:

Define a Python function that takes the preprocessed data, a train/test split ratio, and random_state as input. This function will perform the split, train a LogisticRegression model, and return Accuracy, Confusion Matrix, and ROC-AUC Score for the test set.

Reasoning: I will define a Python function to encapsulate the model training and evaluation process, including data splitting, model initialization, training, prediction, and calculation of Accuracy, Confusion Matrix, and ROC-AUC Score, as requested by the subtask instructions.

```

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score

def train_evaluate_model_task3(X, y, test_size, random_state=42):
    """
    Splits data, trains a Logistic Regression model, and evaluates its performance.

    Args:
        X (pd.DataFrame): Features DataFrame.
        y (pd.Series): Target Series.
        test_size (float): Proportion of the dataset to include in the test split.
        random_state (int): Controls the shuffling applied to the data before applying the split.

    Returns:
        dict: A dictionary containing evaluation metrics.
    """
    print(f"\n--- Evaluating with test_size={test_size*100}% ---")

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)

    # Initialize and train the Logistic Regression model
    model = LogisticRegression(max_iter=200, random_state=random_state)
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)
    # Get predicted probabilities for the positive class (class 1)
    y_proba = model.predict_proba(X_test)[:, 1]

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_proba)

    # Print evaluation metrics
    print(f"Accuracy: {accuracy:.4f}")
    print(f"ROC-AUC Score: {roc_auc:.4f}")
    print("Confusion Matrix:")
    print(conf_matrix)

    return {
        'accuracy': accuracy,
        'confusion_matrix': conf_matrix,
        'roc_auc_score': roc_auc
    }

print("Defined the 'train_evaluate_model_task3' function.")

Defined the 'train_evaluate_model_task3' function.

```

Reasoning: I will call the previously defined `train_evaluate_model_task3` function with `test_size=0.5` to train and evaluate the model using a 50/50 split and store the results.

```
results_50_50 = train_evaluate_model_task3(X, y, test_size=0.5)
print("\nResults for 50/50 split stored in 'results_50_50'.")
```

```
--- Evaluating with test_size=50.0% ---
Accuracy: 0.8072
ROC-AUC Score: 0.8563
Confusion Matrix:
[[233  34]
 [ 52 127]]
```

Results for 50/50 split stored in 'results_50_50'.

Reasoning: I will call the `train_evaluate_model_task3` function with `test_size=0.3` to train and evaluate the model using a 70/30 split and store the results in `results_70_30_task3`.

```
results_70_30_task3 = train_evaluate_model_task3(X, y, test_size=0.3)
print("\nResults for 70/30 split stored in 'results_70_30_task3'.")
```

```
--- Evaluating with test_size=30.0% ---
Accuracy: 0.8097
ROC-AUC Score: 0.8805
Confusion Matrix:
[[136  21]
 [ 30  81]]
```

Results for 70/30 split stored in 'results_70_30_task3'.

Reasoning: I will call the `train_evaluate_model_task3` function with `test_size=0.2` to train and evaluate the model using an 80/20 split and store the results in `results_80_20_task3`.

```
results_80_20_task3 = train_evaluate_model_task3(X, y, test_size=0.2)
print("\nResults for 80/20 split stored in 'results_80_20_task3'.")
```

```
--- Evaluating with test_size=20.0% ---
Accuracy: 0.8101
ROC-AUC Score: 0.8820
Confusion Matrix:
[[90 15]
 [19 55]]
```

Results for 80/20 split stored in 'results_80_20_task3'.

Reasoning: To analyze potential overfitting or underfitting, I will compile the Accuracy and ROC-AUC scores from the 50/50, 70/30, and 80/20 splits into a pandas DataFrame for easy comparison.

```
results_summary_task3 = pd.DataFrame({
    'Split': ['50/50', '70/30', '80/20'],
    'Accuracy': [results_50_50['accuracy'], results_70_30_task3['accuracy'], results_80_20_task3['accuracy']],
    'ROC-AUC Score': [results_50_50['roc_auc_score'], results_70_30_task3['roc_auc_score'], results_80_20_task3['roc_auc_score']]
})

print("\n--- Comparison of Model Performance across Different Train/Test Splits ---")
print(results_summary_task3.round(4))
```

--- Comparison of Model Performance across Different Train/Test Splits ---			
Split	Accuracy	ROC-AUC Score	
0	50/50	0.8072	0.8563
1	70/30	0.8097	0.8805
2	80/20	0.8101	0.8820

Summary:

Q&A

The optimal train/test split, based on the evaluated metrics, appears to be the 80/20 split, showing the highest ROC-AUC score.

Data Analysis Key Findings

- **Data Preprocessing:** The `train.csv` dataset was loaded, and necessary preprocessing steps including handling missing values, encoding categorical features ('Sex', 'Embarked'), and scaling numerical features ('Pclass', 'Age', 'SibSp', 'Parch', 'Fare') using `StandardScaler` were completed. The target variable 'Survived' was correctly separated.
- **Model Performance across Splits:**
 - **50/50 Split:** Achieved an Accuracy of 0.8072 and an ROC-AUC Score of 0.8563. This split uses a relatively small training set, which can sometimes lead to underfitting if the model doesn't learn enough from the data, or a higher variance in performance due to a larger test set.
 - **70/30 Split:** Showed slightly improved performance with an Accuracy of 0.8097 and an ROC-AUC Score of 0.8805. With more data for training, the model's ability to generalize improved.
 - **80/20 Split:** Demonstrated the best overall performance in terms of ROC-AUC, with an Accuracy of 0.8101 and an ROC-AUC Score of 0.8820. This split provides a good balance, offering a substantial training set for the model to learn from while retaining a sufficient test set for evaluation.
- **Overfitting/Underfitting Analysis:**
 - The consistently good performance metrics (Accuracy and ROC-AUC) across all splits suggest that the Logistic Regression model is not severely underfitting or overfitting the data. The slight increase in performance from 50/50 to 70/30 and then to 80/20 indicates that a larger training set (up to 80%) helps the model learn slightly better patterns without significantly degrading performance on the test set, which points to good generalization capabilities.
 - The 50/50 split has the lowest ROC-AUC, which might suggest a slight underfitting due to less training data or higher variance due to a larger test set. As the training set size increases (to 70/30 and 80/20), both accuracy and ROC-AUC marginally improve, suggesting that the model benefits from more training data without showing signs of overfitting on the test sets, as the test performance continues to be strong.

Insights or Next Steps

- The Logistic Regression model, with the chosen features and preprocessing, shows robust performance on the Titanic dataset, effectively generalizing across different train/test splits.

- The 80/20 split appears to be the most optimal among the tested configurations, offering the best balance between training data size and evaluation performance, particularly evidenced by its highest ROC-AUC score.
- To further confirm model stability and generalization, k-fold cross-validation could be implemented, especially given the small dataset size, to provide a more reliable estimate of performance.
- Future steps could involve exploring feature engineering or experimenting with other classification algorithms to potentially achieve even higher predictive performance and further analyze model behavior.

Final Task

Subtask:

Provide a comprehensive summary of the analysis, including the performance comparison across splits and the inference on overfitting/underfitting.

Summary:

Q&A

The optimal train/test split, based on the evaluated metrics, appears to be the 80/20 split, showing the highest ROC-AUC score.

Data Analysis Key Findings

- **Data Preprocessing:** The `train.csv` dataset was loaded, missing values were handled, categorical features ('Sex', 'Embarked') were encoded, and numerical features ('Pclass', 'Age', 'SibSp', 'Parch', 'Fare') were scaled using `StandardScaler`. The 'Survived' column was correctly separated as the target variable.
- **Model Performance across Splits:**
 - **50/50 Split:** Achieved an Accuracy of 0.8072 and an ROC-AUC Score of 0.8563.
 - **70/30 Split:** Showed improved performance with an Accuracy of 0.8097 and an ROC-AUC Score of 0.8805.
 - **80/20 Split:** Demonstrated the best overall performance with an Accuracy of 0.8101 and the highest ROC-AUC Score of 0.8820.
- **Overfitting/Underfitting Analysis:** The model displayed consistently good performance metrics across all splits. A slight increase in Accuracy and ROC-AUC from the 50/50 split to the 70/30 and 80/20 splits suggests that the model benefits from a larger training set. This trend, combined with strong test set performance, indicates that the Logistic Regression model is not severely underfitting or overfitting the data and possesses good generalization capabilities.

Insights or Next Steps