Selecting best model in Pipeline

To select the best model when using multiple models in a pipeline, you can use techniques like cross-validation and evaluation metrics to compare their performance. Here's an example of how to accomplish this on the Titanic dataset:

In [1]:
```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score

# Load the Titanic dataset from Seaborn
titanic_data = sns.load_dataset('titanic')

# Select features and target variable
X = titanic_data[['pclass', 'sex', 'age', 'fare', 'embarked']]
y = titanic_data['survived']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a list of models to evaluate
models = [
    ('Random Forest', RandomForestClassifier(random_state=42)),
    ('Gradient Boosting', GradientBoostingClassifier(random_state=42))
]

best_model = None
best_accuracy = 0.0

# Iterate over the models and evaluate their performance
for name, model in models:
    # Create a pipeline for each model
    pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
```

```
        ('encoder', OneHotEncoder(handle_unknown='ignore')),
        ('model', model)
    ])

    # Perform cross-validation
    scores = cross_val_score(pipeline, X_train, y_train, cv=5)

    # Calculate mean accuracy
    mean_accuracy = scores.mean()

    # Fit the pipeline on the training data
    pipeline.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = pipeline.predict(X_test)

    # Calculate accuracy score
    accuracy = accuracy_score(y_test, y_pred)

    # Print the performance metrics
    print("Model:", name)
    print("Cross-validation Accuracy:", mean_accuracy)
    print("Test Accuracy:", accuracy)
    print()

    # Check if the current model has the best accuracy
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_model = pipeline

# Retrieve the best model
print("Best Model:", best_model)
```

```
Model: Random Forest
Cross-validation Accuracy: 0.7991529597163399
Test Accuracy: 0.8379888268156425

Model: Gradient Boosting
Cross-validation Accuracy: 0.8076135132473162
Test Accuracy: 0.7988826815642458

Best Model: Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent')),
                ('encoder', OneHotEncoder(handle_unknown='ignore')),
                ('model', RandomForestClassifier(random_state=42))])
```

we initialize the best_model and best_accuracy variables to track the best-performing model.

During the iteration over the models, after calculating the accuracy score for each model, we compare it with the current best_accuracy value. If the current model has a higher accuracy, we update best_accuracy and assign the pipeline object to best_model.

After the loop, we print the best model using print("Best Model:", best_model).

By comparing the accuracy scores of different models within the pipeline and selecting the one with the highest accuracy, you can retrieve the best-performing model for the given dataset.

Add more models in the same code

In [2]:
```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load the Titanic dataset from Seaborn
titanic_data = sns.load_dataset('titanic')
```

```
# Select features and target variable
X = titanic_data[['pclass', 'sex', 'age', 'fare', 'embarked']]
y = titanic_data['survived']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a list of models to evaluate
models = [
    ('Random Forest', RandomForestClassifier(random_state=42)),
    ('Gradient Boosting', GradientBoostingClassifier(random_state=42)),
    ('Support Vector Machine', SVC(random_state=42)),
    ('Logistic Regression', LogisticRegression(random_state=42))
]

best_model = None
best_accuracy = 0.0

# Iterate over the models and evaluate their performance
for name, model in models:
    # Create a pipeline for each model
    pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('encoder', OneHotEncoder(handle_unknown='ignore')),
        ('model', model)
    ])

    # Perform cross-validation
    scores = cross_val_score(pipeline, X_train, y_train, cv=5)

    # Calculate mean accuracy
    mean_accuracy = scores.mean()

    # Fit the pipeline on the training data
    pipeline.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = pipeline.predict(X_test)

    # Calculate accuracy score
    accuracy = accuracy_score(y_test, y_pred)
```

```
    # Print the performance metrics
    print("Model:", name)
    print("Cross-validation Accuracy:", mean_accuracy)
    print("Test Accuracy:", accuracy)
    print()

    # Check if the current model has the best accuracy
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_model = pipeline

# Retrieve the best model
print("Best Model:", best_model)
```

```
Model: Random Forest
Cross-validation Accuracy: 0.7991529597163399
Test Accuracy: 0.8379888268156425

Model: Gradient Boosting
Cross-validation Accuracy: 0.8076135132473162
Test Accuracy: 0.7988826815642458

Model: Support Vector Machine
Cross-validation Accuracy: 0.8160248202501723
Test Accuracy: 0.8044692737430168

Model: Logistic Regression
Cross-validation Accuracy: 0.7977839062346105
Test Accuracy: 0.8100558659217877

Best Model: Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent')),
                ('encoder', OneHotEncoder(handle_unknown='ignore')),
                ('model', RandomForestClassifier(random_state=42))])
```

In [ ]: