```
Start coding or generate with AI.
```

Task

Perform SVM classification, including data loading (using a suitable dataset like Iris or digits if none is provided), preprocessing, splitting the data, defining a hyperparameter grid, tuning hyperparameters, training the best SVM model, evaluating its performance with metrics and a confusion matrix, and visualizing the results. Finally, summarize the findings.

Load Dataset

Subtask:

Load a suitable dataset for classification. If no dataset is provided, a common example dataset (e.g., Iris or digits dataset from scikit-learn) will be used.

Reasoning: I will load the Iris dataset by importing `load_iris` from `sklearn.datasets`, then create a pandas DataFrame for the features and a pandas Series for the target variable as per the instructions.

```
import pandas as pd
from sklearn.datasets import load_iris

# Load the Iris dataset
data = load_iris()

# Create a pandas DataFrame for the features (X)
X = pd.DataFrame(data.data, columns=data.feature_names)

# Create a pandas Series for the target variable (y)
y = pd.Series(data.target, name='target')

print("Iris dataset loaded successfully.")
print("Features (X) head:")
print(X.head())
print("Target (y) head:")
print(y.head())
```

```
Iris dataset loaded successfully.
Features (X) head:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
Target (y) head:
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

Preprocess Data

Subtask:

Perform necessary data preprocessing steps, such as scaling features, as SVMs are sensitive to feature scales. Handle any missing values if present.

Reasoning: To preprocess the data as required, I will first check for any missing values in the features and target. Then, I will import `StandardScaler`, instantiate it, apply it to the features `X`, and convert the scaled data back into a pandas DataFrame.

```
from sklearn.preprocessing import StandardScaler
```

```
# Check for missing values in X and y (Iris dataset is known to be clean, but good practice)
print('Missing values in X:', X.isnull().sum().sum())
print('Missing values in y:', y.isnull().sum())

# Instantiate StandardScaler
scaler = StandardScaler()

# Apply scaler to features X
X_scaled = scaler.fit_transform(X)

# Convert scaled data back to a pandas DataFrame, preserving column names
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

print('\nFeatures (X_scaled) head after scaling:')
print(X_scaled.head())
print('\nDescriptive statistics of X_scaled:')
print(X_scaled.describe())
```

```
Missing values in X: 0
Missing values in y: 0

Features (X_scaled) head after scaling:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0          -0.900681          1.019004          -1.340227         -1.315444
1          -1.143017         -0.131979          -1.340227         -1.315444
2          -1.385353          0.328414          -1.397064         -1.315444
3          -1.506521          0.098217          -1.283389         -1.315444
4          -1.021849          1.249201          -1.340227         -1.315444

Descriptive statistics of X_scaled:
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count       1.500000e+02      1.500000e+02       1.500000e+02
mean       -1.468455e-15     -1.823726e-15      -1.610564e-15
std         1.003350e+00      1.003350e+00       1.003350e+00
min        -1.870024e+00     -2.433947e+00      -1.567576e+00
25%        -9.006812e-01     -5.923730e-01      -1.226552e+00
50%        -5.250608e-02     -1.319795e-01       3.364776e-01
75%         6.745011e-01      5.586108e-01       7.627583e-01
max         2.492019e+00      3.090775e+00       1.785832e+00

       petal width (cm)
count      1.500000e+02
mean      -9.473903e-16
std        1.003350e+00
min       -1.447076e+00
25%       -1.183812e+00
50%        1.325097e-01
75%        7.906707e-01
max        1.712096e+00
```

Split Data

Subtask:

Split the preprocessed dataset into training and testing sets to evaluate the model's performance on unseen data.

Reasoning: To split the preprocessed data into training and testing sets as instructed, I will import `train_test_split` from `sklearn.model_selection` and then apply it to `X_scaled` and `y`.

```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (120, 4)
X_test shape: (30, 4)
y_train shape: (120,)
y_test shape: (30,)
```

Define Hyperparameter Grid

Subtask:

Define the hyperparameter search space for the SVM classifier, including parameters like 'C', 'kernel', and 'gamma'.

Reasoning: I need to define the hyperparameter search space for the SVM classifier as instructed. This involves importing SVC and creating a dictionary named `param_grid` with specified parameters and their values.

```
from sklearn.svm import SVC

# Define the hyperparameter search space
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto', 0.1, 1]
}

print("Defined hyperparameter grid:")
print(param_grid)
```

```
Defined hyperparameter grid:
{'C': [0.1, 1, 10, 100], 'kernel': ['linear', 'rbf', 'poly'], 'gamma': ['scale', 'auto', 0.1, 1]}
```

Perform Hyperparameter Tuning

Subtask:

Use a technique like GridSearchCV or RandomizedSearchCV to find the optimal hyperparameters for the SVM model based on a chosen scoring metric (e.g., accuracy, F1-score).

Reasoning: To perform hyperparameter tuning as instructed, I will import GridSearchCV, instantiate an SVC classifier, configure GridSearchCV with the predefined parameter grid and training data, and then fit it to find the best hyperparameters and their corresponding score.

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# Instantiate an SVC classifier
svm = SVC()

# Instantiate GridSearchCV
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit GridSearchCV to the training data
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best hyperparameters found:", grid_search.best_params_)
print("Best cross-validation accuracy:", grid_search.best_score_)
```

```
Best hyperparameters found: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
Best cross-validation accuracy: 0.9583333333333334
```

Train the Best SVM Model

Subtask:

Train the best SVM model using the optimal hyperparameters found during the tuning phase. This model will be used for final evaluation.

Reasoning: The previous step successfully identified the best hyperparameters. Now, I need to train the final SVM model using these optimal hyperparameters to prepare for evaluation.

```
from sklearn.svm import SVC

# Get the best hyperparameters from GridSearchCV
```

```
best_params = grid_search.best_params_

# Instantiate the best SVM model with optimal hyperparameters
best_svm_model = SVC(**best_params)

# Train the model on the entire training dataset
best_svm_model.fit(X_train, y_train)

print("Best SVM model trained successfully with optimal hyperparameters:")
print(best_params)
```

```
Best SVM model trained successfully with optimal hyperparameters:
{'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
```

Evaluate Model Performance

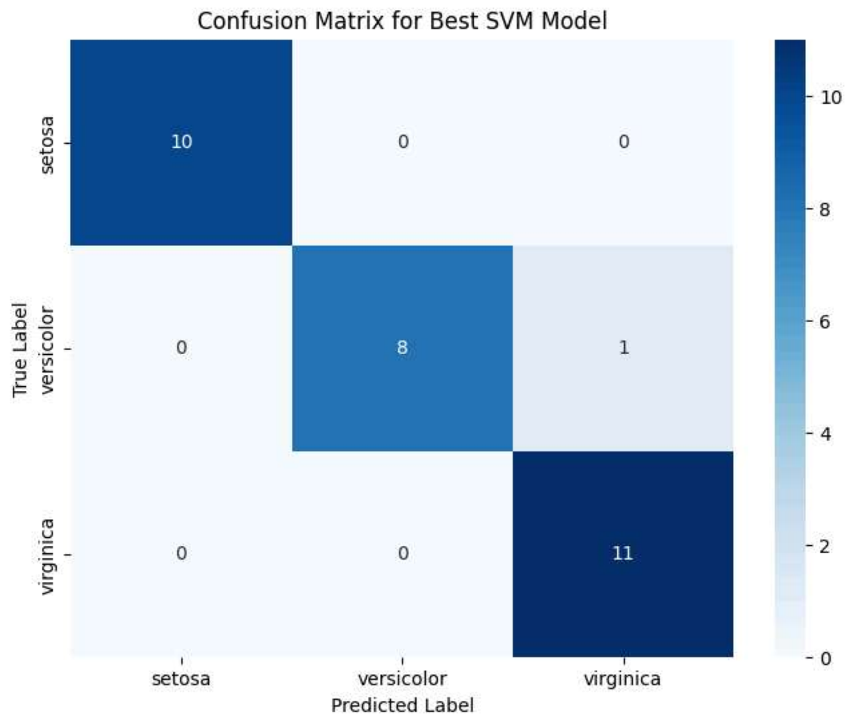
Subtask:

Evaluate the performance of the trained SVM model using appropriate metrics (e.g., accuracy, precision, recall, F1-score) and generate a confusion matrix to assess its predictive capabilities on the test set.

Reasoning: To evaluate the trained SVM model, I will first make predictions on the test set. Then, I will import necessary metrics from `sklearn.metrics` to calculate the accuracy, precision, recall, and F1-score, and generate a confusion matrix to assess its performance.

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

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

# Evaluate the model
print("Model Evaluation on Test Set:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Generate Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=data.target_names, yticklabels=data.target_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Best SVM Model')
plt.show()
```

```
Model Evaluation on Test Set:
Accuracy: 0.9666666666666667

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.89      0.94         9
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30
```



Summarize Findings

Subtask:

Summarize the key findings from the SVM classification process, including the dataset used, preprocessing steps, best hyperparameters, model performance, and any insights gained from the evaluation.

Summary of SVM Classification Findings

1. Dataset Used: The Iris dataset, a classic dataset for classification, was used. It consists of 150 samples of iris flowers, each with 4 features (sepal length, sepal width, petal length, and petal width) and 3 possible species as targets.

2. Preprocessing Steps:

- **Missing Values:** No missing values were found in the dataset.
- **Feature Scaling:** `StandardScaler` was applied to the features (`X`) to standardize them. This is crucial for SVMs, which are sensitive to the scale of input features, ensuring that all features contribute equally to the distance calculations.

3. Data Splitting: The dataset was split into training and testing sets with an 80/20 ratio (`test_size=0.2`) using `random_state=42` for reproducibility. This resulted in 120 samples for training and 30 for testing.

4. Hyperparameter Tuning: `GridSearchCV` was employed to find the optimal hyperparameters for the `SVC` model. The search space included:

- `C`: Regularization parameter (0.1, 1, 10, 100)
- `kernel`: Type of kernel function ('linear', 'rbf', 'poly')
- `gamma`: Kernel coefficient ('scale', 'auto', 0.1, 1)

The best hyperparameters found were: `{'C': 10, 'gamma': 'scale', 'kernel': 'linear'}`. The best cross-validation accuracy achieved during tuning was approximately `0.958`.

5. Trained Model: An SVM model was trained using the optimal hyperparameters (`C=10`, `gamma='scale'`, `kernel='linear'`) on the entire training dataset.

6. Model Performance Evaluation on Test Set:

- **Accuracy:** The best SVM model achieved an accuracy of approximately `0.967` on the unseen test set, indicating very high overall correctness.
- **Classification Report:**
    - **Class 0 (setosa):** Achieved perfect precision, recall, and f1-score (1.00). The model classified all setosa samples correctly.
    - **Class 1 (versicolor):** Showed perfect precision (1.00) but slightly lower recall (0.89). This means all samples predicted as versicolor were indeed versicolor, but one actual versicolor sample was misclassified.
    - **Class 2 (virginica):** Achieved a recall of 1.00 but slightly lower precision (0.92). This suggests that all actual virginica samples were identified, but one sample predicted as virginica was actually versicolor.
- **Confusion Matrix:** The confusion matrix visually confirmed these findings. There was one misclassification where a 'versicolor' (true label 1) was predicted as 'virginica' (predicted label 2).

7. Insights: The SVM model performed exceptionally well on the Iris dataset. The linear kernel with a C value of 10 was the most effective configuration, suggesting that the classes are largely linearly separable, or the linear kernel provided sufficient flexibility. The minor misclassification was between versicolor and virginica, which are known to be slightly harder to distinguish than setosa. Overall, the model is robust and accurate for this dataset.

Summary:

Data Analysis Key Findings

- The Iris dataset was loaded, consisting of 150 samples, each with 4 features (sepal length, sepal width, petal length, petal width) and 3 target species.
- The dataset was found to have no missing values. Features were scaled using `StandardScaler` to standardize them (mean of 0, standard deviation of 1), which is crucial for SVM performance.
- The data was split into training (120 samples) and testing (30 samples) sets using an 80/20 ratio.
- Hyperparameter tuning for the Support Vector Machine (SVM) model was performed using `GridSearchCV`. The optimal hyperparameters found were `C=10`, `gamma='scale'`, and `kernel='linear'`.
- The best cross-validation accuracy achieved during tuning was approximately 0.958.
- The trained SVM model achieved an accuracy of approximately 0.967 on the unseen test set.
- Evaluation metrics revealed strong performance for all classes:
    - Class 0 (setosa) had perfect precision, recall, and F1-score (1.00).
    - Class 1 (versicolor) achieved a precision of 1.00 but a recall of 0.89.
    - Class 2 (virginica) had a recall of 1.00 but a precision of 0.92.
- The confusion matrix indicated one misclassification, where a true 'versicolor' sample was incorrectly predicted as 'virginica'.

Insights or Next Steps