```python
#data exploration and preparation
#Importing necessary libraries
from sklearn import datasets
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV,
cross_val_score, KFold
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, classification_report

# Step 1: Loading the Iris dataset
iris = datasets.load_iris()

# Step 2: Converting the dataset into a pandas DataFrame for easier
manipulation
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Adding the target variable to the DataFrame
iris_df['species'] = iris.target

# Step 3: Data preprocessing
# Checking for missing values
print("Checking for missing values:")
print(iris_df.isnull().sum())

# Scaling the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(iris_df.drop('species',
axis=1))
```

```
Checking for missing values:
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
species              0
dtype: int64
```

```python
# Step 4 : Before implementing SVM i want to split my data set
# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(scaled_features,
iris_df['species'], test_size=0.2, random_state=42)

# Step 5: SVM Model Configuration
# Defining the SVM classifier
svm_classifier = SVC()

# Defining a dictionary of hyperparameters to search
```

```
param_grid = {
    'C': [0.1, 1, 10, 100],  # Regularization parameter
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],  # Kernel type
    'gamma': ['scale', 'auto'],  # Kernel coefficient for 'rbf',
'poly', and 'sigmoid'
    'degree': [2, 3, 4],  # Degree of the polynomial kernel
}

# Step 6: Cross validation procedure

from sklearn.model_selection import cross_val_score, KFold

# Define the number of folds for cross-validation
k = 5

# Define the KFold object with shuffling enabled
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Perform cross-validation
cv_scores = cross_val_score(svm_classifier, X_train, y_train, cv=kf)

# Print the cross-validation scores
print("Cross-validation scores:", cv_scores)

# Calculate and print the mean accuracy of cross-validation
mean_cv_accuracy = cv_scores.mean()
print("\nMean cross-validation accuracy:", mean_cv_accuracy)
```

```
Cross-validation scores: [0.91666667 1.         0.91666667 0.91666667
1.        ]

Mean cross-validation accuracy: 0.95
```

```
# Step 7: Model Evaluation
# Training the SVM classifier with the best parameters
grid_search = GridSearchCV(svm_classifier, param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_svm_classifier = grid_search.best_estimator_

# Making predictions on the test set using the best SVM model
y_pred = best_svm_classifier.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Printing evaluation metrics
print("Evaluation Metrics:")
print("Accuracy:", accuracy)
```

```
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Print the best parameters found
print("\nBest parameters:", grid_search.best_params_)
```

```
Evaluation Metrics:
Accuracy: 0.9666666666666667
Precision: 0.9694444444444444
Recall: 0.9666666666666667
F1-score: 0.9664109121909632

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


Best parameters: {'C': 10, 'degree': 2, 'gamma': 'scale', 'kernel':
'linear'}
```

Data Exploration and Preparation:

```
# Importing necessary libraries

from sklearn import datasets

import pandas as pd

from sklearn.preprocessing import StandardScaler

# Step 1: Loading the Iris dataset

iris = datasets.load_iris()

# Step 2: Converting the dataset into a pandas DataFrame for easier manipulation

iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Adding the target variable to the DataFrame

iris_df['species'] = iris.target

# Checking for missing values

print("Checking for missing values:")

print(iris_df.isnull().sum())

# Scaling the features

scaler = StandardScaler()

scaled_features = scaler.fit_transform(iris_df.drop('species', axis=1))

# Splitting the dataset into train and test sets

X_train, X_test, y_train, y_test = train_test_split(scaled_features, iris_df['species'], test_size=0.2,
random_state=42)
```

Observations and comments:

1.The Iris dataset is loaded using the datasets.load_iris() function from scikit-learn.
2. It includes characteristics pertaining to the target variable species as well as size of the petals and sepals.
3. The dataset has no missing values, guaranteeing the accuracy of the data.
4. To standardize features and improve model performance, StandardScaler() is used.
5. An 80-20 split ratio is used to separate the dataset into training and testing sets.

SVM Implementation:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
# Defining the SVM classifier
svm_classifier = SVC()
# Defining a dictionary of hyperparameters to search
param_grid = {
    'C': [0.1, 1, 10, 100],  # Regularization parameter
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],  # Kernel type
    'gamma': ['scale', 'auto'],  # Kernel coefficient for 'rbf', 'poly', and 'sigmoid'
    'degree': [2, 3, 4],  # Degree of the polynomial kernel
}
# Training the SVM classifier with the best parameters using grid search
grid_search = GridSearchCV(svm_classifier, param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_svm_classifier = grid_search.best_estimator_
```

Observations and Comments:
1. The scikit-learn SVC() function is used to implement the Support Vector Machine (SVM) classifier.
2. GridSearchCV is used to do a grid search to determine the ideal set of hyperparameters.
3. The hyperparameters consist of degree, gamma, kernel, and C.

```
from sklearn.model_selection import cross_val_score, KFold
```

Cross Validation:

```
# Define the number of folds for cross-validation
k = 5
# Define the KFold object with shuffling enabled
kf = KFold(n_splits=k, shuffle=True, random_state=42)
# Perform cross-validation
cv_scores = cross_val_score(svm_classifier, X_train, y_train, cv=kf)
# Print the cross-validation scores
print("Cross-validation scores:", cv_scores)
# Calculate and print the mean accuracy of cross-validation
mean_cv_accuracy = cv_scores.mean()
print("\nMean cross-validation accuracy:", mean_cv_accuracy)
```

Observations and Comments:

1. The training data is divided into 5 folds using the KFold object, with bias prevention enabled by shuffling.
2. Cross_val_score, which automates the cross-validation procedure, is used to train and assess the SVM classifier.

Evaluation Metrics:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report

# Making predictions on the test set using the best SVM model
y_pred = best_svm_classifier.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Printing evaluation metrics
print("Evaluation Metrics:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Print the best parameters found
print("\nBest parameters:", grid_search.best_params_)
```

Observations And Comments:

1. A variety of evaluation metrics, including accuracy, precision, recall, and F1-score, are used to assess the model's performance.
2. A thorough analysis of performance metrics for every class is given in the classification report.
3. To comprehend the ideal SVM setup, the optimum grid search parameters are printed.

Output Observations and Insights:

Data Exploration and Preparation:

1. The dataset is clean and ready for analysis because there are no missing values in it.

2. StandardScaler was used to scale the features in order to guarantee that each feature has an equal influence on the model's decision-making.

Svm Implementation:

1.{'C': 10, 'degree': 2, 'gamma':'scale', 'kernel': 'linear'} are the optimal parameters discovered by the grid search.

2.This indicates that the best results for this dataset were obtained using a linear kernel with a regularization parameter of 10 and scaled gamma.

K-fold Cross-Validation:

1. K-fold cross-validation with five folds was used to evaluate the SVM model's performance.

2. This makes it more likely that the model will perform well and won't be unduly reliant on a specific training-test split.

Evaluation Metrics:
1. The SVM model performed well on the test set:
Accuracy: 96.67%
Precision: 96.94%
Recall : 96.67%
F1 Score: 96.64%
2. These measures show how effectively the model works in different classification contexts.
3. The comprehensive classification report offers information on the model's accuracy, recall, and F1-score for every class.
4. Class 1 showed somewhat lesser recall (89%), whereas Class 0 and Class 2 had 100% precision and recall.