In [20]:
```
import numpy as np
import pandas as pd
data = pd.read_csv(r"C:\Users\vkrat\Documents\cancer.csv")
data.head()
```

Out[20]:

	Class	age	menopause	tumor-size	inv-nodes	node-caps	deg-malig	breast	breast-quad	irradiat
0	0	5	1	1	1	2	1	3	1	1
1	0	5	4	4	5	7	10	3	2	1
2	0	3	1	1	1	2	2	3	1	1
3	0	6	8	8	1	3	4	3	7	1
4	0	4	1	1	3	2	1	3	1	1

In [21]:
```
print(data.columns)
print(data["menopause"].tail())
```

```
Index(['Class', 'age', 'menopause', 'tumor-size', 'inv-nodes', 'node-caps',
       'deg-malig', 'breast', 'breast-quad', 'irradiat'],
      dtype='object')
678     1
679     1
680    10
681     8
682     8
Name: menopause, dtype: int64
```

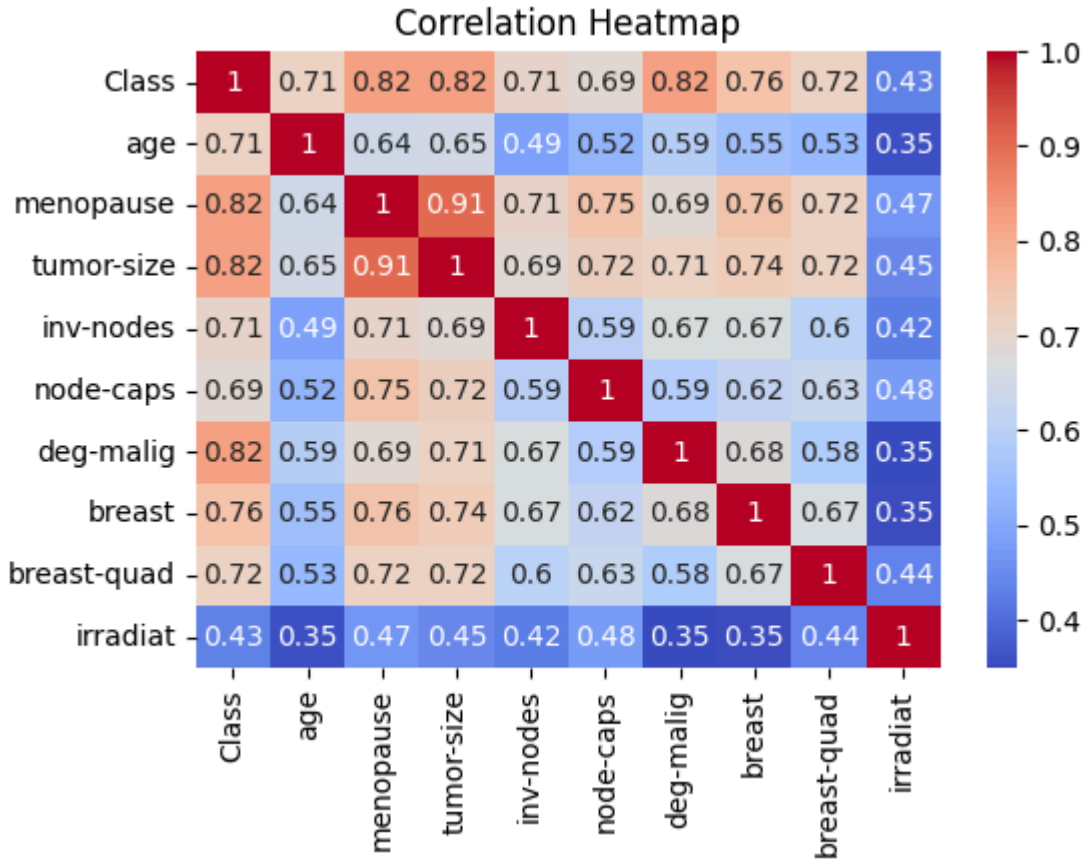
In [22]:
```
from sklearn.preprocessing import LabelEncoder
# Encode categorical variables
le = LabelEncoder()
categorical_cols = ["menopause", "node-caps", "breast", "breast-quad", "irradiat"]
for col in categorical_cols:
 data[col] = le.fit_transform(data[col])
# Map target class if it's categorical
if data["Class"].dtype == 'object':
 data["Class"] = le.fit_transform(data["Class"])
```

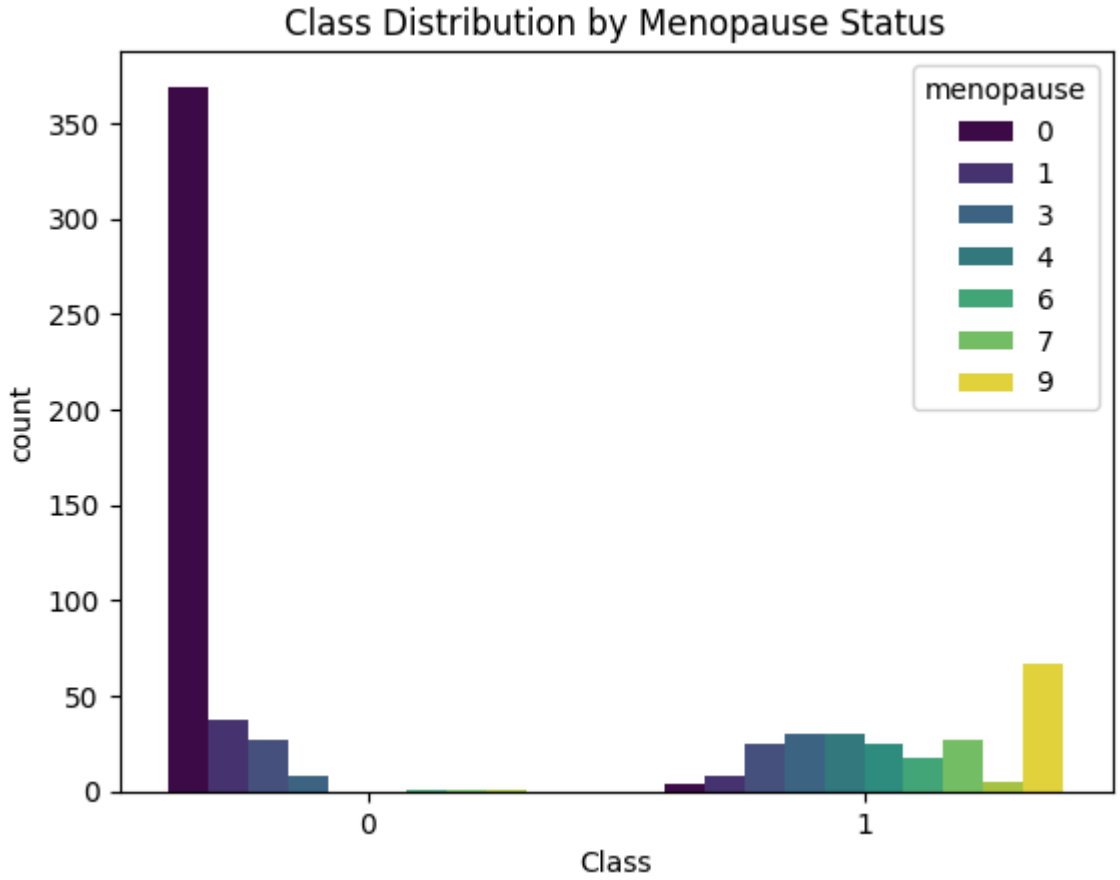
```
# Check for missing values
print(data.isnull().sum())
# Handle missing values if necessary
# For simplicity, we'll drop rows with missing values
data.dropna(inplace=True)
```

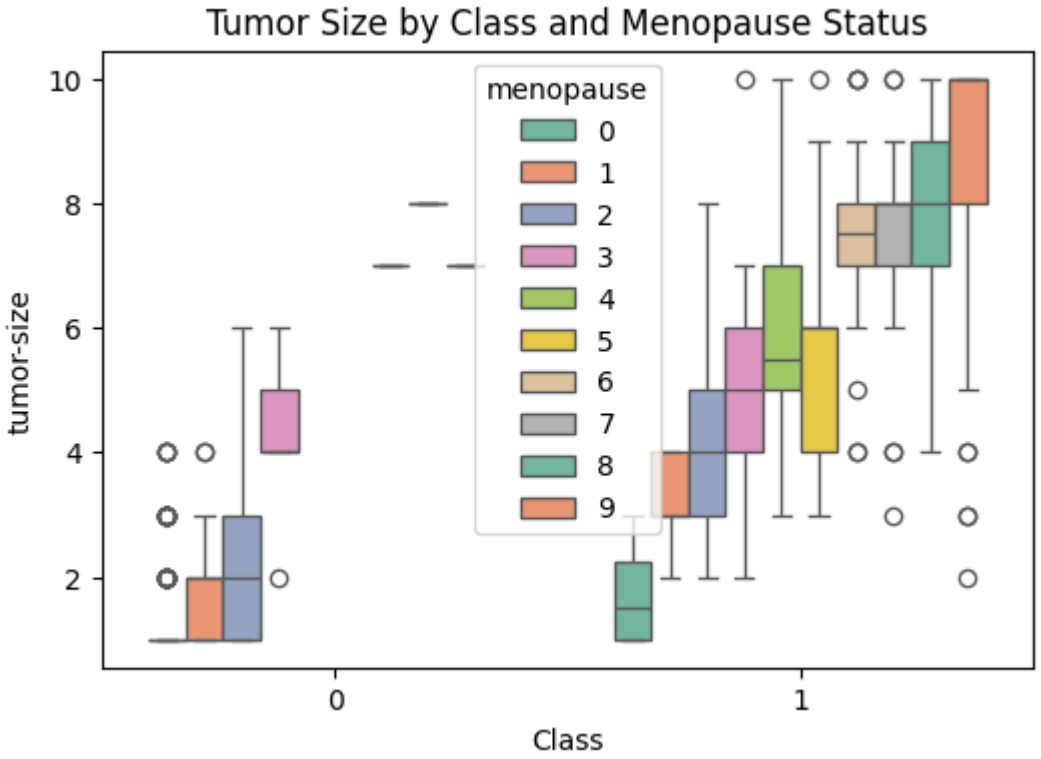
```
Class          0
age            0
menopause      0
tumor-size     0
inv-nodes      0
node-caps      0
deg-malig      0
breast         0
breast-quad    0
irradiat       0
dtype: int64
```

In [26]:

```
from sklearn.model_selection import train_test_split
# Define features (X) and target (y)
X = data.drop("Class", axis=1)
y = data["Class"]
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
import matplotlib.pyplot as plt
import seaborn as sns
#Correlation Heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(data.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
#Class Distribution
sns.countplot(data=data, x="Class", hue="menopause", palette="viridis")
plt.title("Class Distribution by Menopause Status")
plt.show()
#Box Plot for Tumor Size
plt.figure(figsize=(6, 4))
sns.boxplot(data=data, x="Class", y="tumor-size", hue="menopause", palette="Set2")
plt.title("Tumor Size by Class and Menopause Status")
plt.show()
```


Correlation Heatmap
Class 1 0.71 0.82 0.82 0.71 0.69 0.82 0.76 0.72 0.43
age 0.71 1 0.64 0.65 0.49 0.52 0.59 0.55 0.53 0.35
menopause 0.82 0.64 1 0.91 0.71 0.75 0.69 0.76 0.72 0.47
tumor-size 0.82 0.65 0.91 1 0.69 0.72 0.71 0.74 0.72 0.45
inv-nodes 0.71 0.49 0.71 0.69 1 0.59 0.67 0.67 0.6 0.42
node-caps 0.69 0.52 0.75 0.72 0.59 1 0.59 0.62 0.63 0.48
deg-malig 0.82 0.59 0.69 0.71 0.67 0.59 1 0.68 0.58 0.35
breast 0.76 0.55 0.76 0.74 0.67 0.62 0.68 1 0.67 0.35
breast-quad 0.72 0.53 0.72 0.72 0.6 0.63 0.58 0.67 1 0.44
irradiat 0.43 0.35 0.47 0.45 0.42 0.48 0.35 0.35 0.44 1
Class
age
menopause
tumor-size
inv-nodes
node-caps
deg-malig
breast
breast-quad
irradiat
1.0
0.9
0.8
0.7
0.6
0.5
0.4

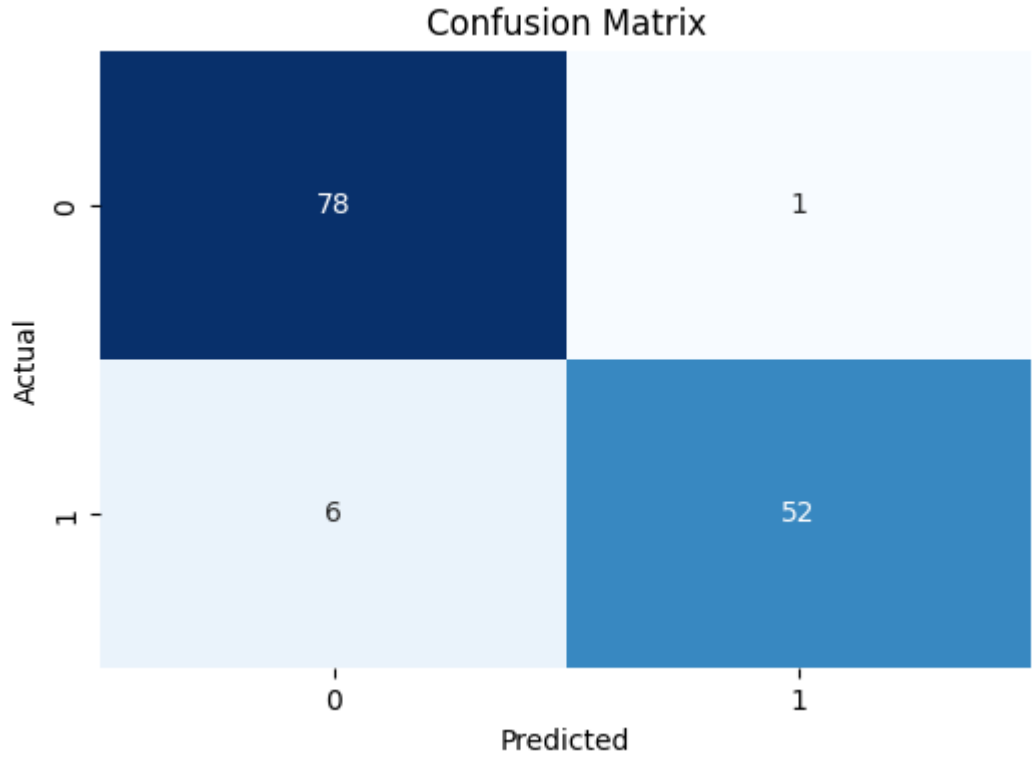

Class Distribution by Menopause Status
menopause
0
1
3
4
6
7
9
count
350
300
250
200
150
100
50
0
0
1
Class



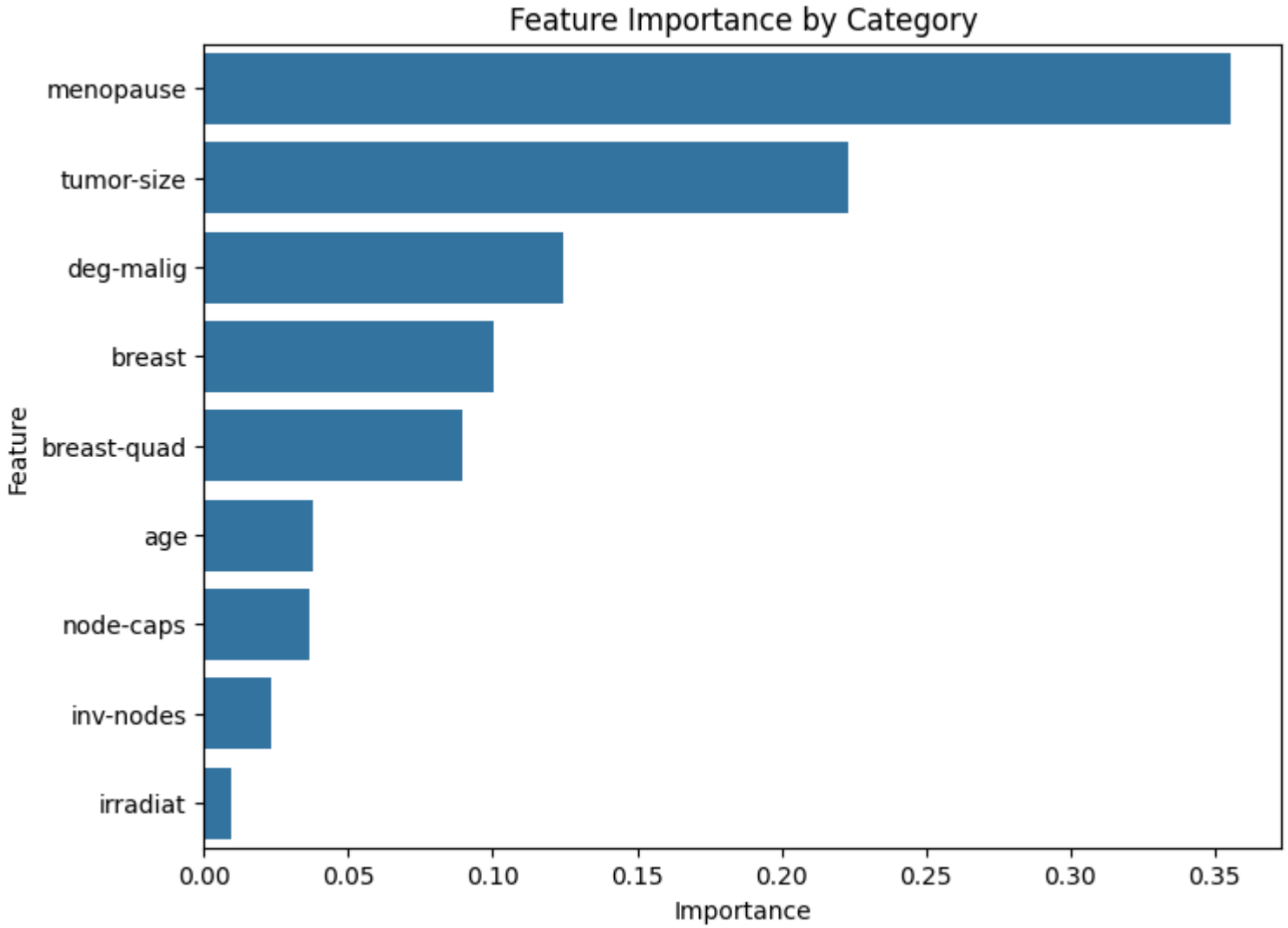
In [27]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
# Train a Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
# Predict on test set
y_pred = rf.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.99      0.96        79
           1       0.98      0.90      0.94        58

    accuracy                           0.95       137
   macro avg       0.95      0.94      0.95       137
weighted avg       0.95      0.95      0.95       137
```

In [30]:
```python
#Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [34]:
```
# Example: If you had a 'Category' column
importance = pd.DataFrame({"Feature": X.columns, "Importance": rf.feature_importances_})
importance = importance.sort_values("Importance", ascending=False)
plt.figure(figsize=(8, 6))
sns.barplot(data=importance, x="Importance", y="Feature")
plt.title("Feature Importance by Category")
plt.show()
```



In [35]:
```
import joblib
from sklearn.ensemble import RandomForestClassifier
model_filename = "cancer_survivability_model.pkl"
try:
 joblib.dump(rf, model_filename)
 print(f"Model saved as '{model_filename}'.")
```

```
except Exception as e:
 print(f"Error saving the model: {e}")
```

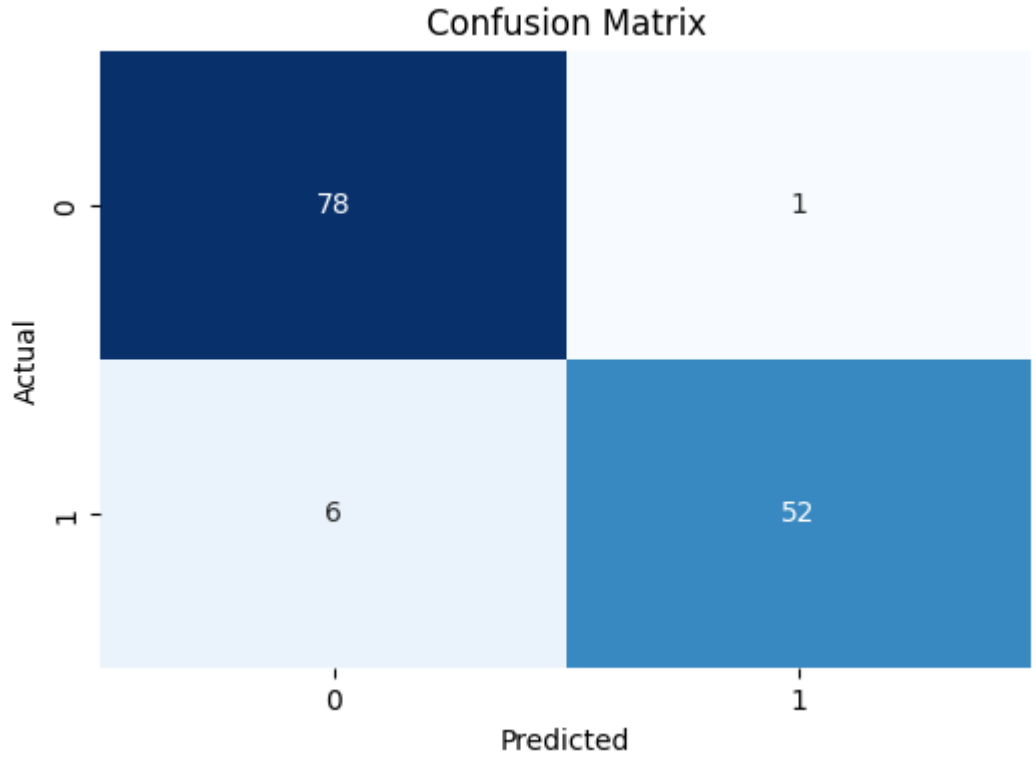
```
Model saved as 'cancer_survivability_model.pkl'.
```

In [36]:
```
# Loading the model (for future use)
# Load the saved model
loaded_model = joblib.load("cancer_survivability_model.pkl")
# Example: Predict on new data (X_test)
predictions = loaded_model.predict(X_test)
print(predictions)
```

```
[1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 0 0 1 1 0 1 0 1 1 0 0 0 1
 0 1 1 0 1 1 0 0 1 0 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0
 1 1 0 0 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 1
 1 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 1 1 1 0 0]
```

In [37]:
```
# Hyperparameter Tuning with GridSearchCV
import joblib
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score # Add accuracy_score import
from sklearn.model_selection import GridSearchCV, cross_val_score
# Accuracy Score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
# Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Accuracy: 0.9489
```

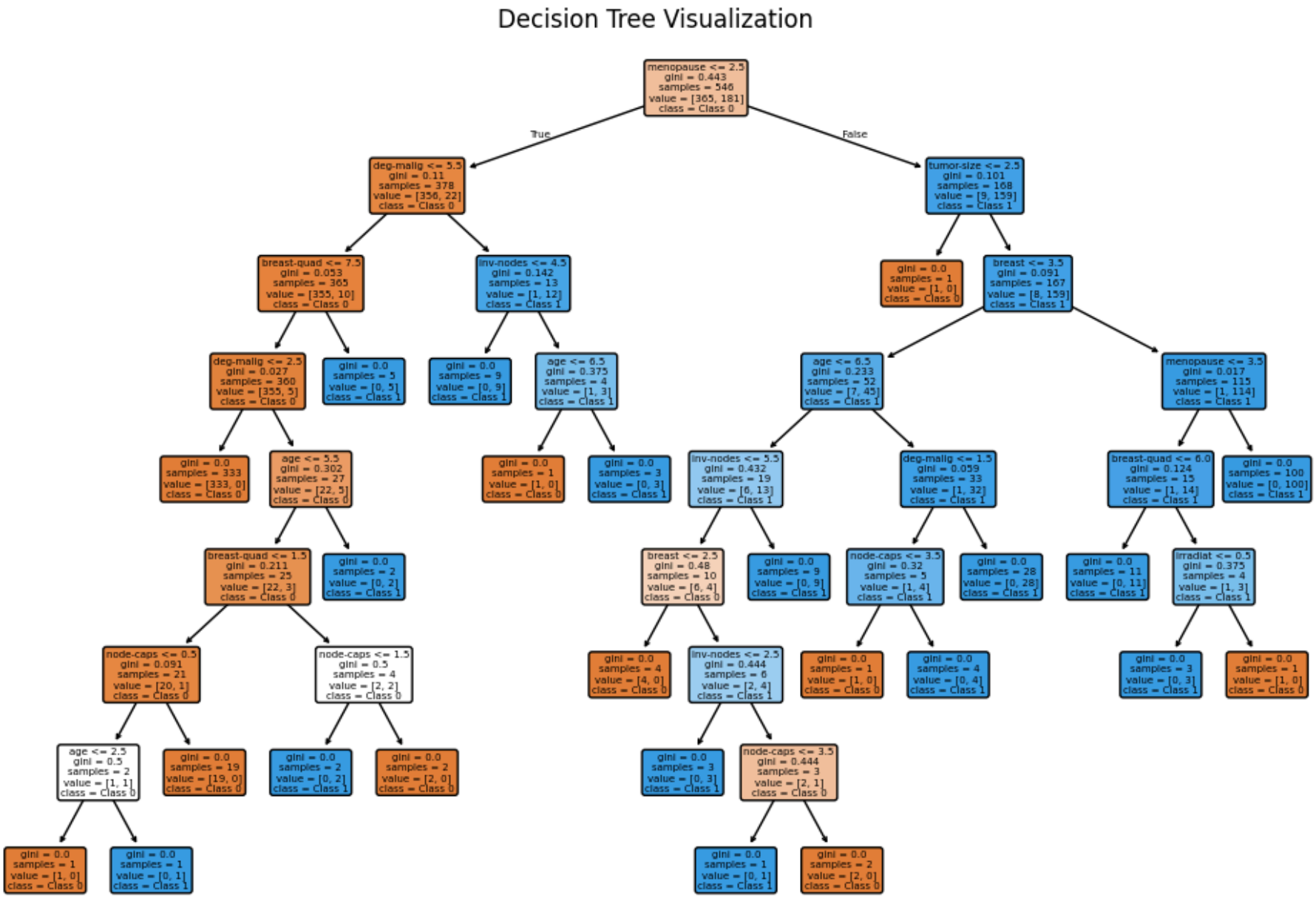


In [38]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
# Predictions and Evaluation
y_pred = dt.predict(X_test)
# Accuracy Score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 0.9343
```

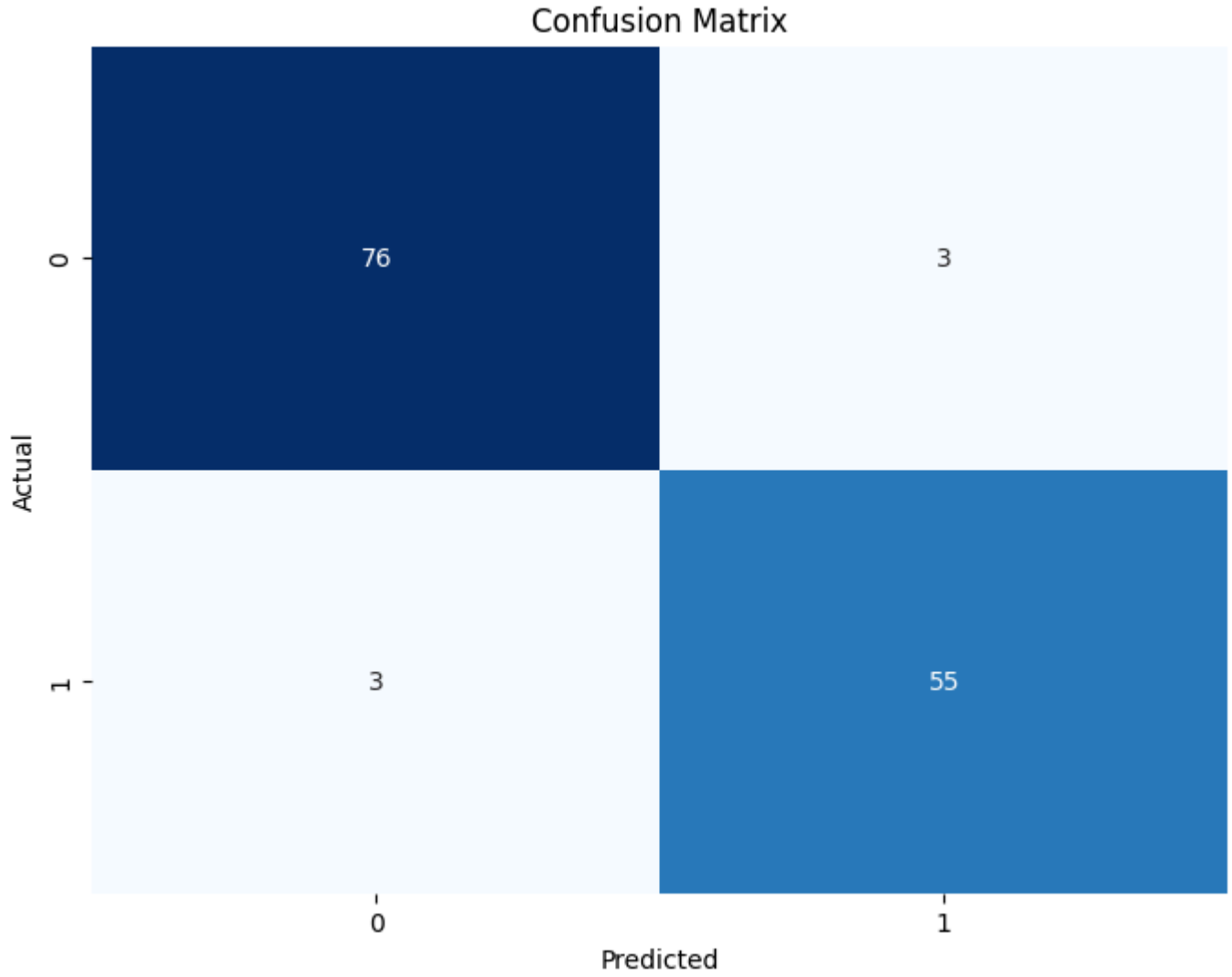
In [43]:
```
# Visualizing the Decision Tree
from sklearn.tree import plot_tree
plt.figure(figsize=(12, 8))
plot_tree(dt, feature_names=X.columns, class_names=['Class 0', 'Class 1'], filled=True, rounded=True)
```

```
plt.title("Decision Tree Visualization")
plt.show()
```

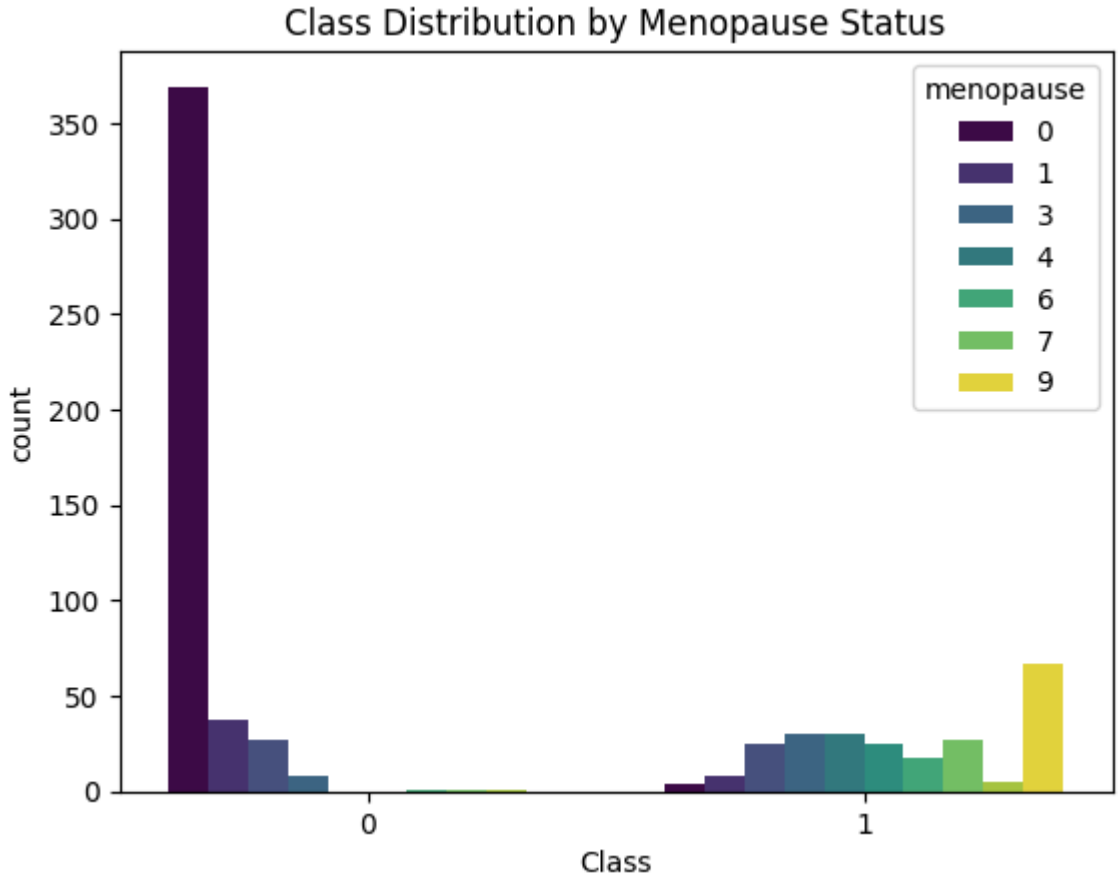


In [51]:
```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
# Train the Naive Bayes model on the training data
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)
# Accuracy Score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
# Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
Accuracy: 0.9562
```

```
Accuracy: 0.9562
```

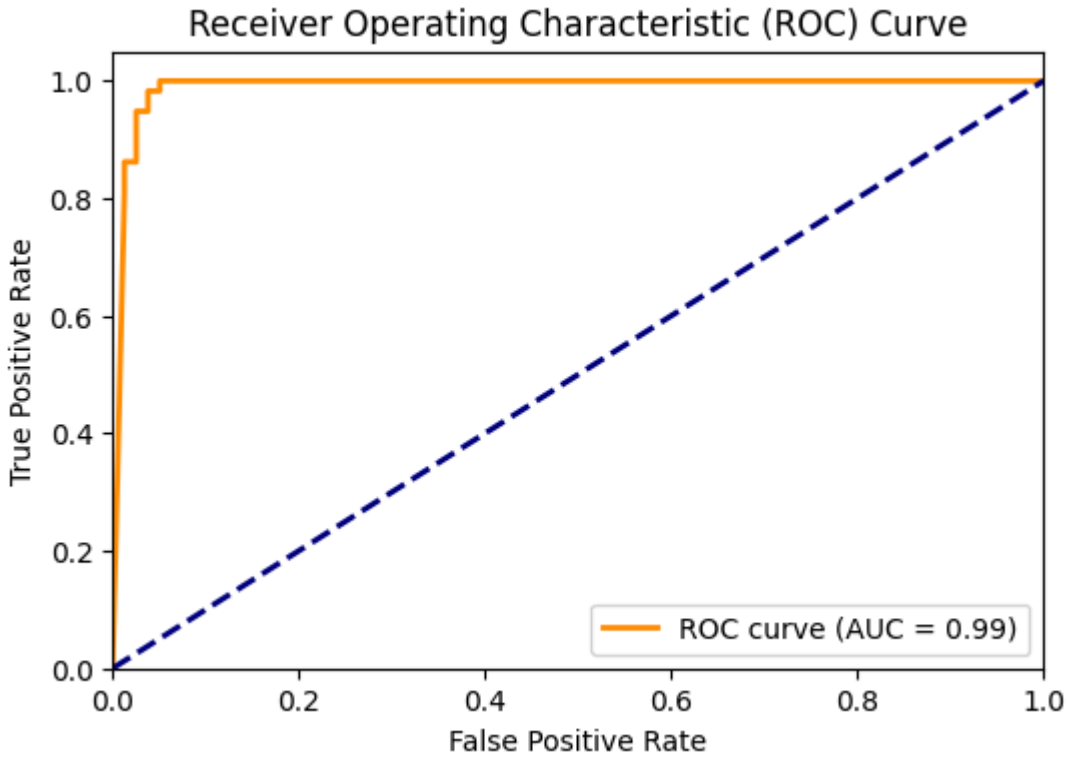


In [46]:
```
# Class Distribution Visualization
sns.countplot(data=data, x="Class", hue="menopause", palette="viridis")
plt.title("Class Distribution by Menopause Status")
plt.show()
```



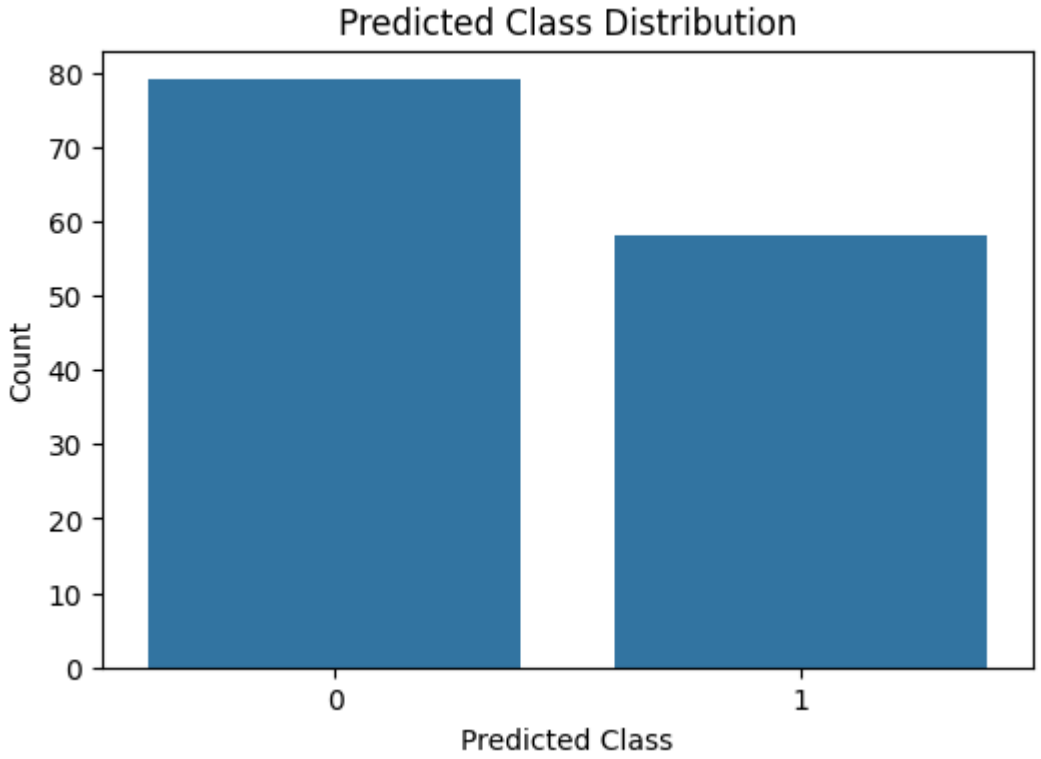
In [50]:
```
from sklearn.metrics import roc_curve, auc
# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, nb.predict_proba(X_test)[:, 1])
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
```

```
plt.legend(loc="lower right")
plt.show()
```



In [49]:
```
# Visualize the predicted class distribution
# Accuracy Score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
plt.figure(figsize=(6, 4))
sns.countplot(x=y_pred)
plt.title("Predicted Class Distribution")
plt.xlabel("Predicted Class")
plt.ylabel("Count")
plt.show()
```

```
Accuracy: 0.9562
```


Predicted Class Distribution
Count
80
70
60
50
40
30
20
10
0
0
1
Predicted Class

In [ ]: