导入数据库

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc, roc_auc_score
from sklearn.preprocessing import StandardScaler
```

数据加载及处理

In [2]:
```
# 1. 数据加载与预处理
# 载入乳腺癌数据集，该数据集包含 569 个样本，每个样本有 30 个特征和二分类标签（良性/恶性）
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

# 输出数据集基本信息
print("数据集特征形状: ", X.shape)
print("数据集标签分布:\n", y.value_counts())

# 数据预处理: 标准化特征数据
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
数据集特征形状:  (569, 30)
数据集标签分布:
 1    357
0    212
Name: count, dtype: int64
```

可视化结果

In [3]:
```
# 将标准化后的数据转换为DataFrame便于后续处理和可视化
X_scaled = pd.DataFrame(X_scaled, columns=data.feature_names)

# 2. 数据探索性分析（Exploratory Data Analysis, EDA）
# 分析数据集的基本统计量、相关性和分布情况，帮助我们更好地理解数据
print("\n数据集描述统计信息:\n", X.describe())

# 可视化数据特征间的相关性热力图
plt.figure(figsize=(14, 10))
corr = X.corr()
sns.heatmap(corr, annot=True, fmt=".2f", cmap="RdBu_r")
plt.title("Feature Correlation Heatmap", fontsize=16)
plt.xlabel("X Axis", fontsize=14)
plt.ylabel("Y Axis", fontsize=14)
plt.tight_layout()
plt.show()
```

数据集描述统计信息:

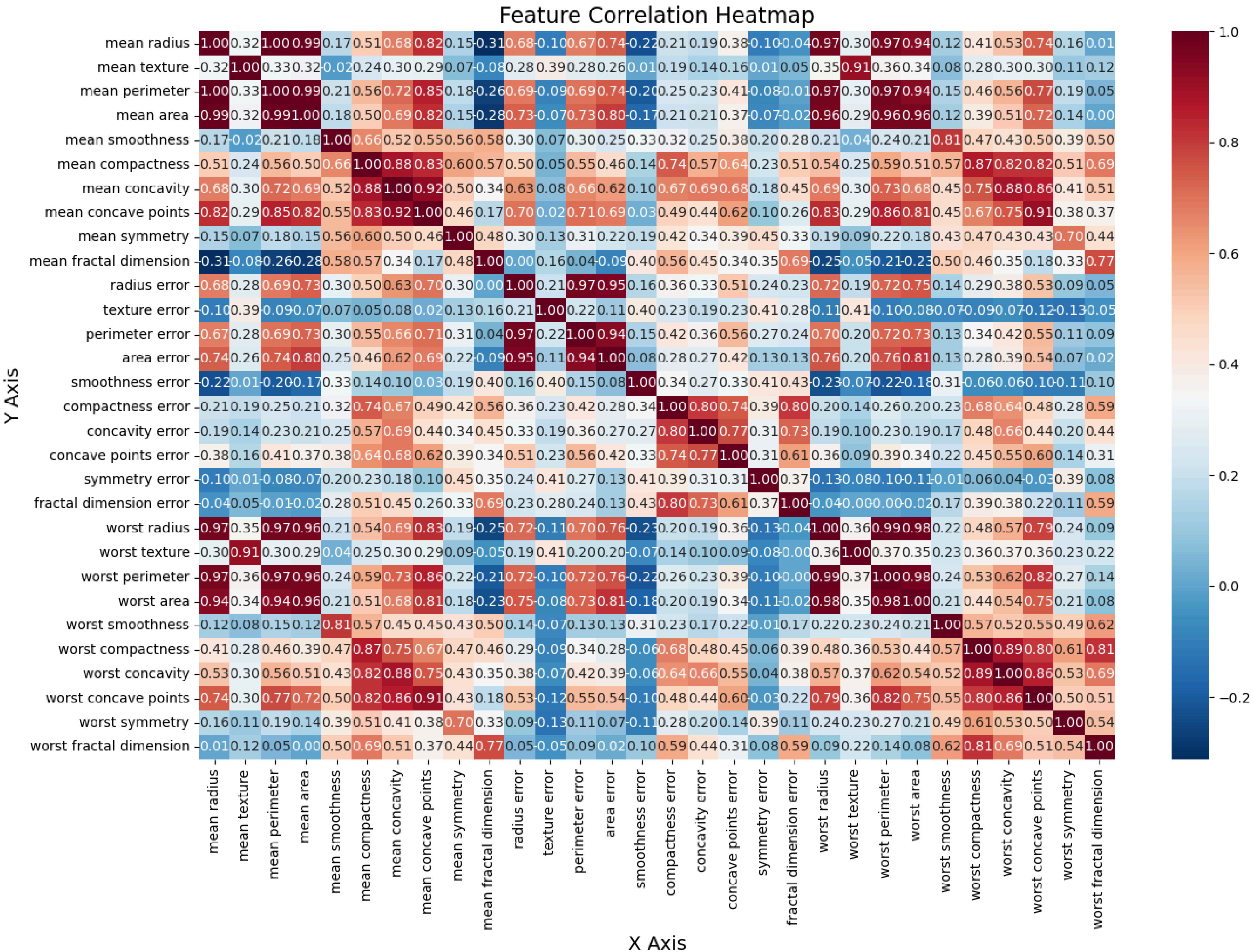
	mean radius	mean texture	mean perimeter	mean area
count	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104
std	3.524049	4.301036	24.298981	351.914129
min	6.981000	9.710000	43.790000	143.500000
25%	11.700000	16.170000	75.170000	420.300000
50%	13.370000	18.840000	86.240000	551.100000
75%	15.780000	21.800000	104.100000	782.700000
max	28.110000	39.280000	188.500000	2501.000000

	mean smoothness	mean compactness	mean concavity	mean concave points
count	569.000000	569.000000	569.000000	569.000000
mean	0.096360	0.104341	0.088799	0.048919
std	0.014064	0.052813	0.079720	0.038803
min	0.052630	0.019380	0.000000	0.000000
25%	0.086370	0.064920	0.029560	0.020310
50%	0.095870	0.092630	0.061540	0.033500
75%	0.105300	0.130400	0.130700	0.074000
max	0.163400	0.345400	0.426800	0.201200

	mean symmetry	mean fractal dimension	...	worst radius
count	569.000000	569.000000	...	569.000000
mean	0.181162	0.062798	...	16.269190
std	0.027414	0.007060	...	4.833242
min	0.106000	0.049960	...	7.930000
25%	0.161900	0.057700	...	13.010000
50%	0.179200	0.061540	...	14.970000
75%	0.195700	0.066120	...	18.790000
max	0.304000	0.097440	...	36.040000

	worst texture	worst perimeter	worst area	worst smoothness
count	569.000000	569.000000	569.000000	569.000000
mean	25.677223	107.261213	880.583128	0.132369
std	6.146258	33.602542	569.356993	0.022832
min	12.020000	50.410000	185.200000	0.071170
25%	21.080000	84.110000	515.300000	0.116600
50%	25.410000	97.660000	686.500000	0.131300
75%	29.720000	125.400000	1084.000000	0.146000
max	49.540000	251.200000	4254.000000	0.222600

```
       worst compactness  worst concavity  worst concave points  \
count         569.000000       569.000000            569.000000
mean            0.254265         0.272188              0.114606
std             0.157336         0.208624              0.065732
min             0.027290         0.000000              0.000000
25%             0.147200         0.114500              0.064930
50%             0.211900         0.226700              0.099930
75%             0.339100         0.382900              0.161400
max             1.058000         1.252000              0.291000

       worst symmetry  worst fractal dimension
count      569.000000               569.000000
mean         0.290076                 0.083946
std          0.061867                 0.018061
min          0.156500                 0.055040
25%          0.250400                 0.071460
50%          0.282200                 0.080040
75%          0.317900                 0.092080
max          0.663800                 0.207500

[8 rows x 30 columns]
```

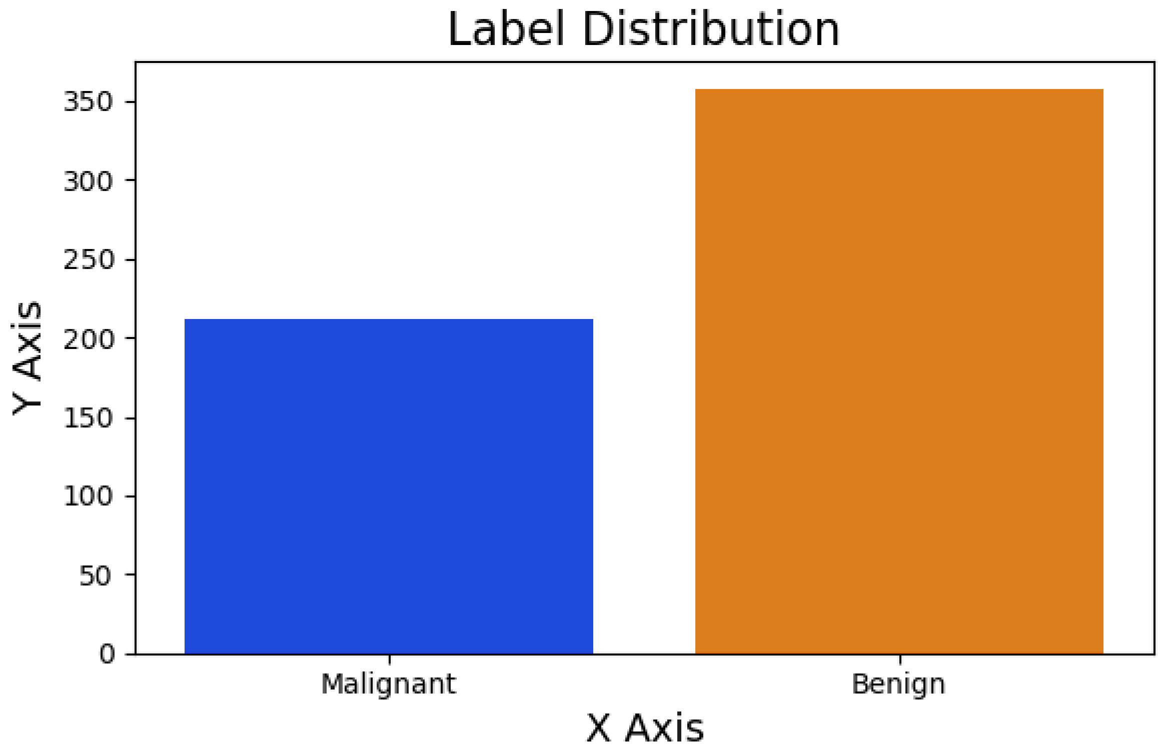

Feature Correlation Heatmap
Y Axis
X Axis
mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

数据驯化及可视化

In [4]:

```
# 可视化标签分布情况：良性与恶性样本比例
plt.figure(figsize=(6, 4))
sns.countplot(x=y, palette="bright")
plt.title("Label Distribution", fontsize=16)
plt.xlabel("X Axis", fontsize=14)
plt.ylabel("Y Axis", fontsize=14)
plt.xticks([0, 1], ['Malignant', 'Benign'])
plt.tight_layout()
plt.show()
```

```
C:\Users\86187\AppData\Local\Temp\ipykernel_17632\1426562402.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and se
t `legend=False` for the same effect.

  sns.countplot(x=y, palette="bright")
```



划分数据集，构建随机森林，输出结果。

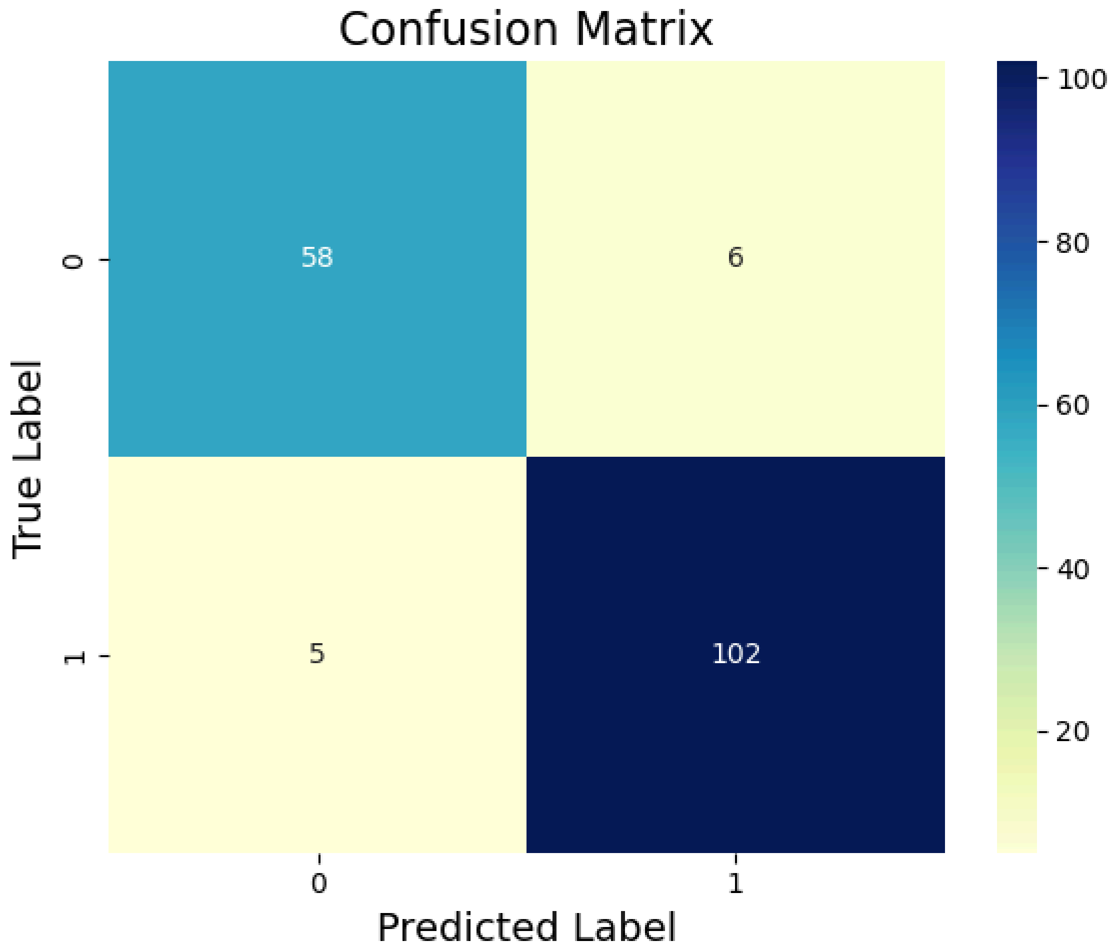
In [5]:
```
# 3. 划分训练集和测试集
# 为了评估模型的泛化能力，将数据集随机分为训练集和测试集，其中测试集占比 30%
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42, stratify=y)
print("\n训练集样本数: ", X_train.shape[0])
print("测试集样本数: ", X_test.shape[0])

# 4. 构建基础随机森林分类器
# 初步构建随机森林模型，并进行训练和预测
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

# 输出分类报告和混淆矩阵
print("\nInitial Model Classification Report:\n", classification_report(y_test, y_pred))
```

```
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
```

```
训练集样本数： 398
测试集样本数： 171

Initial Model Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.91      0.91        64
           1       0.94      0.95      0.95       107

    accuracy                           0.94       171
   macro avg       0.93      0.93      0.93       171
weighted avg       0.94      0.94      0.94       171

Confusion Matrix:
 [[ 58   6]
 [  5 102]]
```

可视化

In [6]:
```
# 可视化混淆矩阵
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="YlGnBu", cbar=True)
plt.title("Confusion Matrix", fontsize=16)
plt.xlabel("Predicted Label", fontsize=14)
plt.ylabel("True Label", fontsize=14)
plt.tight_layout()
plt.show()
```



调参以及训练模型

In [7]:
```
# 5. 随机森林超参数调优
# 通过 GridSearchCV 网格搜索方法寻找最佳超参数组合
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
```

```
}

# 采用 5 折交叉验证
grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state=42),
                           param_grid=param_grid,
                           cv=5,
                           n_jobs=-1,
                           verbose=1,
                           scoring='accuracy')

# 训练调优
grid_search.fit(X_train, y_train)

# 输出最佳超参数及最佳得分
print("\nBest Parameters:", grid_search.best_params_)
print("Best CV Accuracy: {:.4f}".format(grid_search.best_score_))
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits

Best Parameters: {'bootstrap': True, 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Best CV Accuracy: 0.9725
```

优化

In [8]:

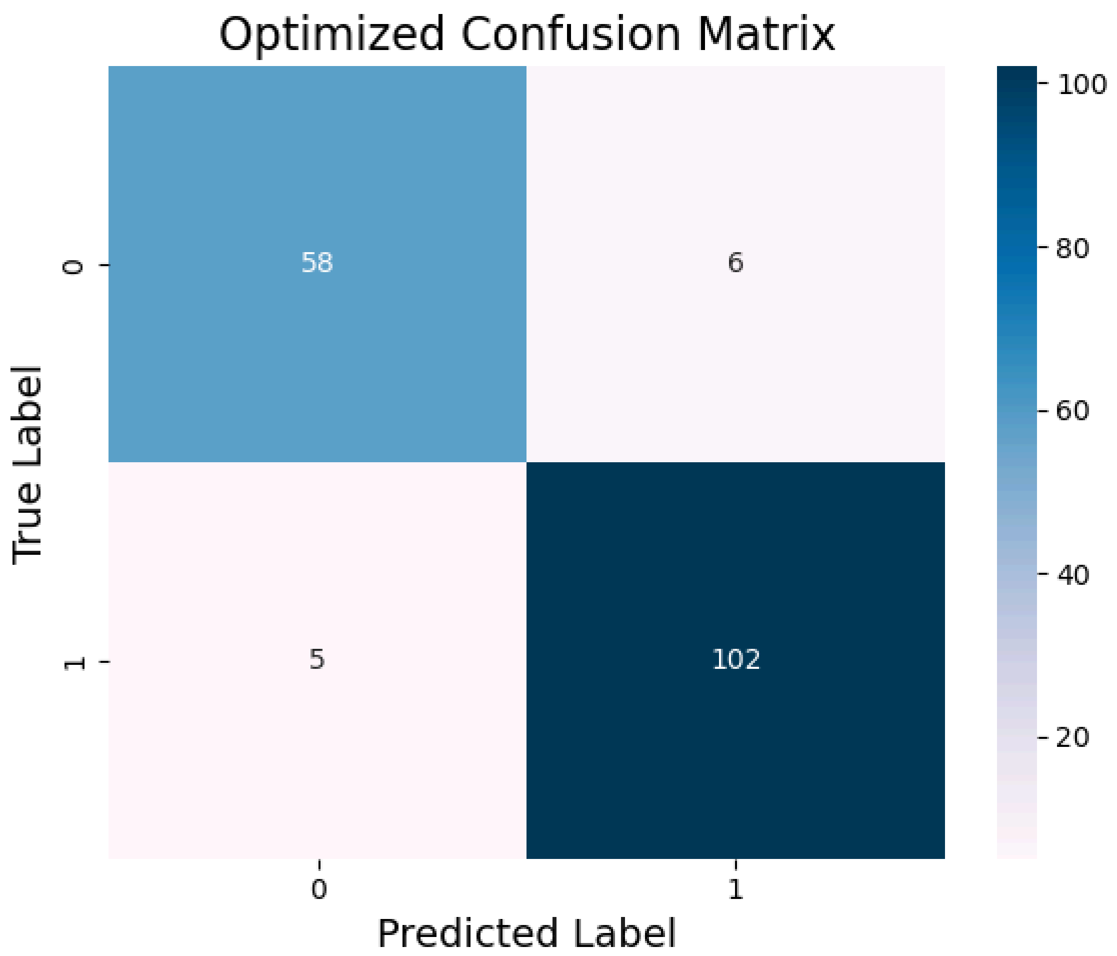
```
# 6. 使用最优参数构建优化后的随机森林模型
best_rf = grid_search.best_estimator_
best_rf.fit(X_train, y_train)
y_pred_best = best_rf.predict(X_test)

# 输出优化后模型的分类报告和混淆矩阵
print("\nOptimized Model Classification Report:\n", classification_report(y_test, y_pred_best))
cm_best = confusion_matrix(y_test, y_pred_best)
print("Optimized Confusion Matrix:\n", cm_best)
```

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

           0       0.92      0.91      0.91        64
           1       0.94      0.95      0.95       107

    accuracy                           0.94       171
   macro avg       0.93      0.93      0.93       171
weighted avg       0.94      0.94      0.94       171

Optimized Confusion Matrix:
 [[ 58   6]
 [  5 102]]
```

可视化

In [9]:

```
# 可视化优化后模型的混淆矩阵
plt.figure(figsize=(6, 5))
sns.heatmap(cm_best, annot=True, fmt="d", cmap="PuBu", cbar=True)
plt.title("Optimized Confusion Matrix", fontsize=16)
plt.xlabel("Predicted Label", fontsize=14)
plt.ylabel("True Label", fontsize=14)
plt.tight_layout()
plt.show()
```

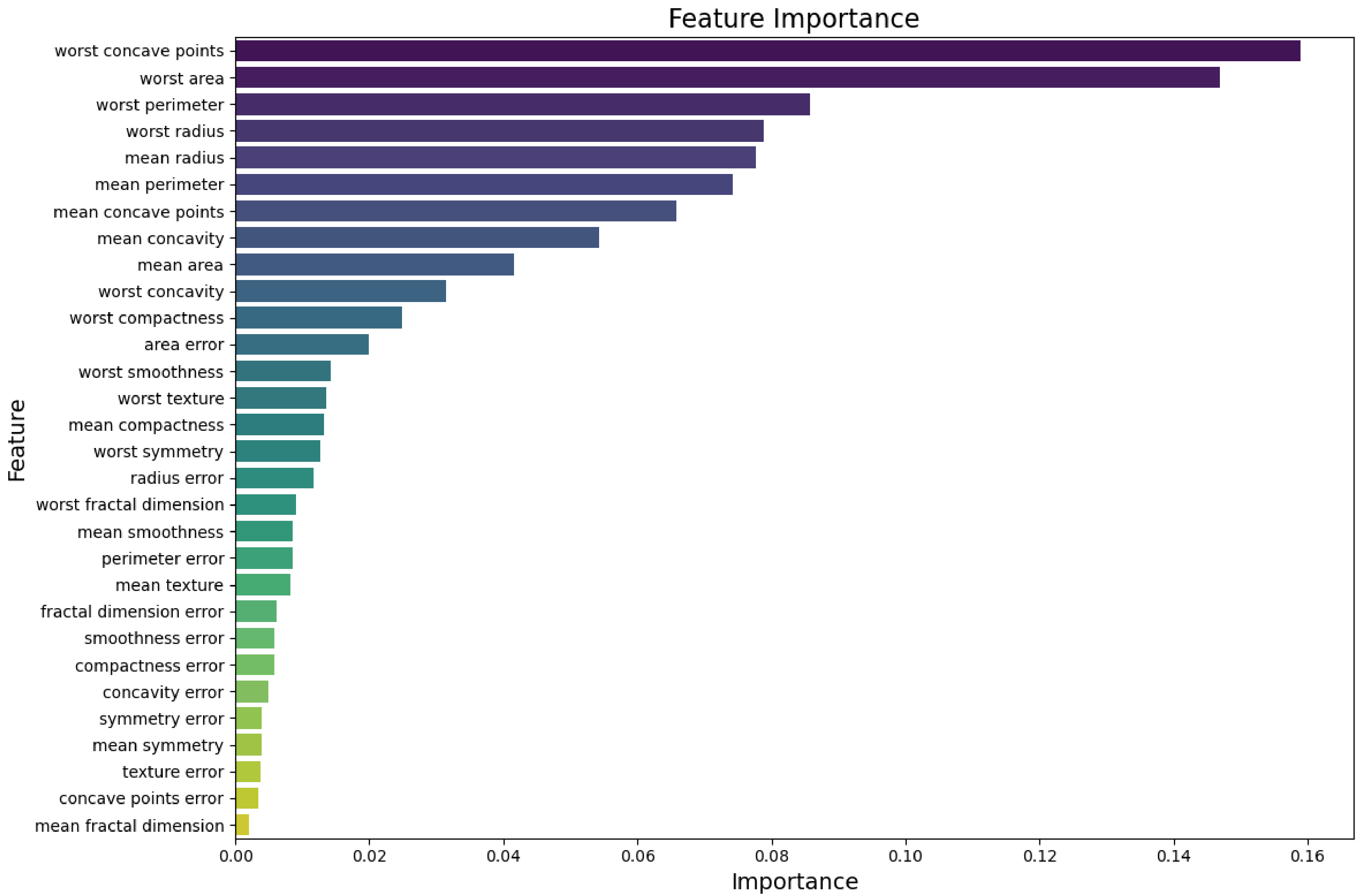


可视化

In [10]:
```
# 7. 模型的重要性分析（Feature Importance）
# 随机森林模型可以计算各个特征的重要性，下面绘制特征重要性图，帮助理解哪些特征对分类任务贡献最大
importances = best_rf.feature_importances_
indices = np.argsort(importances)[::-1]
features = X.columns

# 可视化特征重要性
plt.figure(figsize=(12, 8))
```

```
sns.barplot(x=importances[indices], y=features[indices], palette="viridis")
plt.title("Feature Importance", fontsize=16)
plt.xlabel("Importance", fontsize=14)
plt.ylabel("Feature", fontsize=14)
plt.tight_layout()
plt.show()
```

```
C:\Users\86187\AppData\Local\Temp\ipykernel_17632\3564549565.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and se
t `legend=False` for the same effect.

  sns.barplot(x=importances[indices], y=features[indices], palette="viridis")
```

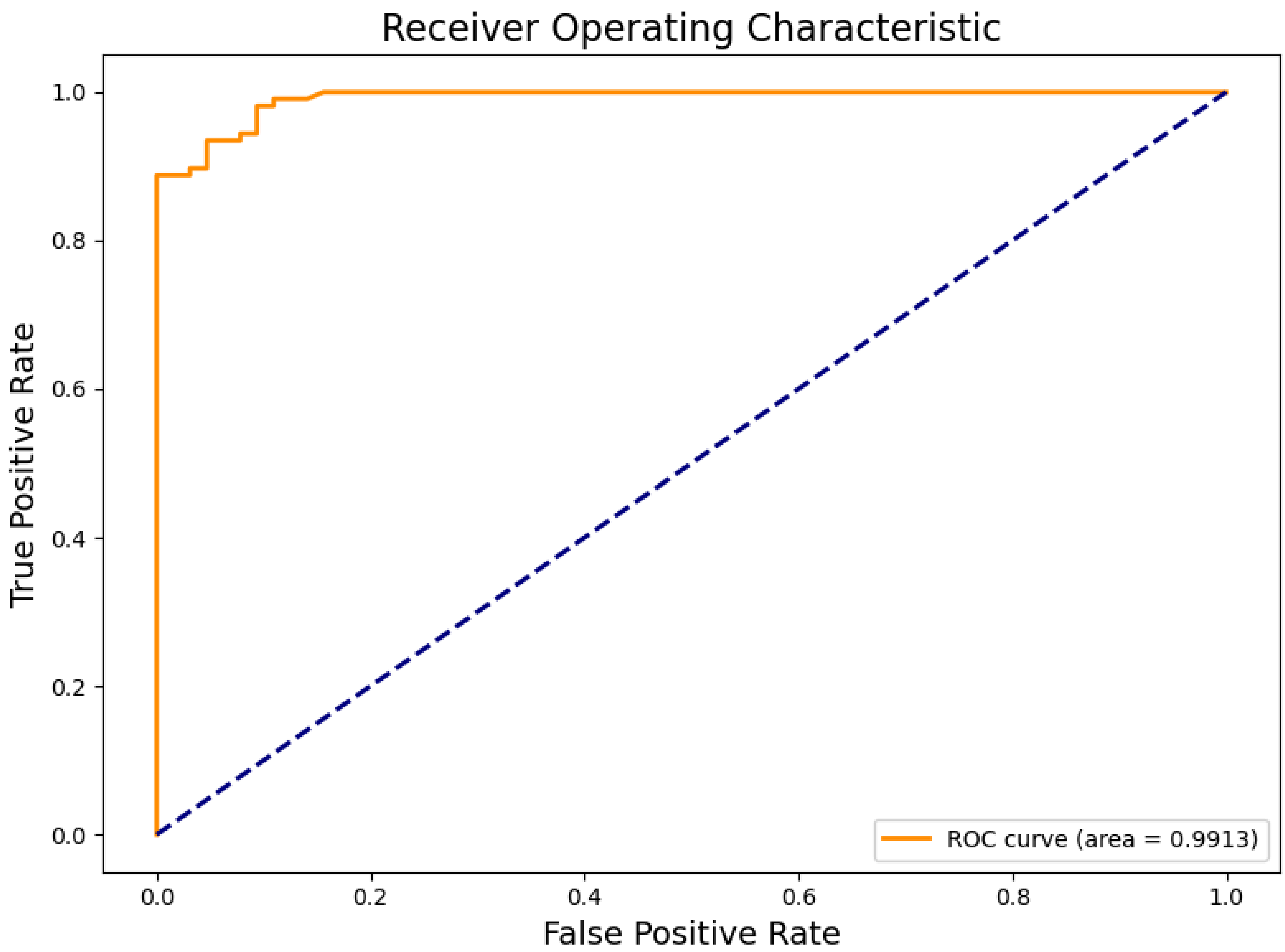

Feature Importance
worst concave points
worst area
worst perimeter
worst radius
mean radius
mean perimeter
mean concave points
mean concavity
mean area
worst concavity
worst compactness
area error
worst smoothness
worst texture
mean compactness
worst symmetry
radius error
worst fractal dimension
mean smoothness
perimeter error
mean texture
fractal dimension error
smoothness error
compactness error
concavity error
symmetry error
mean symmetry
texture error
concave points error
mean fractal dimension
Feature
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
Importance

分数评估

In [11]:

```
# 8. ROC 曲线与 AUC 分数评估
# 计算 ROC 曲线，并绘制 ROC 曲线图，评估模型分类效果
y_proba = best_rf.predict_proba(X_test)[:, 1]  # 获取正例概率
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)
print("\nOptimized Model ROC AUC: {:.4f}".format(roc_auc))

# 可视化 ROC 曲线
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.title("Receiver Operating Characteristic", fontsize=16)
plt.xlabel("False Positive Rate", fontsize=14)
plt.ylabel("True Positive Rate", fontsize=14)
plt.legend(loc="lower right")
plt.tight_layout()
plt.show()
```

```
Optimized Model ROC AUC: 0.9913
```

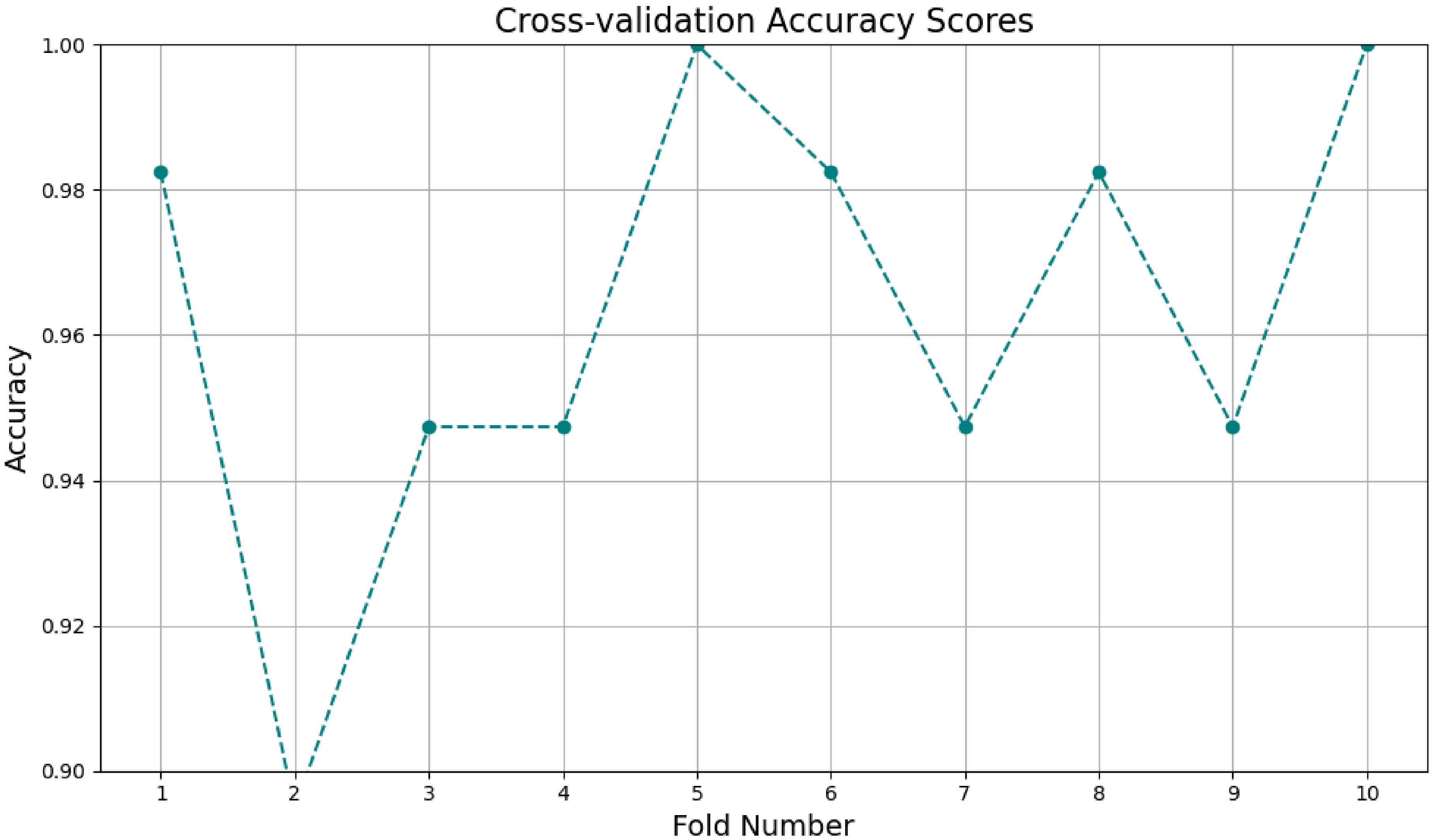


最后分析

In [12]:
```
# 9. 交叉验证与模型稳定性评估
# 采用交叉验证进一步评估模型的稳定性和鲁棒性
```

```
cv_scores = cross_val_score(best_rf, X_scaled, y, cv=10, scoring='accuracy')
print("\nCross-validation Accuracy Scores:\n", cv_scores)
print("Mean CV Accuracy: {:.4f}".format(np.mean(cv_scores)))

# 可视化交叉验证结果
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), cv_scores, marker='o', linestyle='--', color='teal')
plt.title("Cross-validation Accuracy Scores", fontsize=16)
plt.xlabel("Fold Number", fontsize=14)
plt.ylabel("Accuracy", fontsize=14)
plt.xticks(range(1, 11))
plt.ylim(0.90, 1.00)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Cross-validation Accuracy Scores:
 [0.98245614 0.89473684 0.94736842 0.94736842 1.         0.98245614
 0.94736842 0.98245614 0.94736842 1.        ]
Mean CV Accuracy: 0.9632
```



相关代码分析都会放在machine-learning-code文件夹中进行。也不可否认，模型的模拟结果确实不错。