# Final Project on Breast Cancer Wisconsin Dataset:

Breast cancer is one of the most common and life-impacting diseases worldwide, and early detection plays a crucial role in improving patient outcomes. Machine learning has become a powerful tool in medical diagnostics, offering the ability to identify patterns in clinical data that may not be immediately visible to the human eye. In this project, we apply supervised learning techniques to the Breast Cancer Wisconsin Diagnostic Dataset, a widely used benchmark dataset in machine learning and healthcare research.

The objective of this project is to build predictive models capable of distinguishing between malignant and benign breast tumors based on features computed from digitized images of fine-needle aspirates (FNA) of breast masses. These features describe various properties of cell nuclei, such as radius, texture, smoothness, and concavity.

This project follows a full machine-learning workflow:

Problem Definition: - I aim to create a classification model that predicts tumor type (malignant or benign) from the provided numeric features.

Exploratory Data Analysis (EDA): - I analyze the structure of the dataset, examine feature distributions, evaluate correlations, and visualize the data using pairplots and PCA.

Model Building & Training: - Several supervised learning models are trained, including Logistic Regression, Random Forest, Support Vector Machine (SVM) we studied at class.

Model Evaluation & Comparison: - Models are evaluated using accuracy, classification reports, and cross-validation. Hyperparameter tuning via GridSearchCV is used to optimize performance.

Conclusion & Insights: - I identify which model performs best and discuss why it is suitable for this dataset, along with key takeaways about dataset characteristics and model behavior.

This project demonstrates how classical machine learning models can be applied to a real-world medical classification problem, highlighting the importance of data preprocessing, model selection, and careful evaluation. Ultimately, the goal is to build an accurate, interpretable, and computationally efficient model for breast cancer diagnosis.

In [1]:
```
from sklearn.datasets import load_breast_cancer
import pandas as pd

# Load dataset
data = load_breast_cancer()

# Create DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

df.head()
```

Out[1]:

| | mean radius | mean texture | mean perimeter | mean area | mean smoothness | mean compactness | mean concavity | mean concave points | mean symmetry | di |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 | 0.27760 | 0.3001 | 0.14710 | 0.2419 | |
| **1** | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 | 0.07864 | 0.0869 | 0.07017 | 0.1812 | |
| **2** | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 | 0.15990 | 0.1974 | 0.12790 | 0.2069 | |
| **3** | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 | 0.28390 | 0.2414 | 0.10520 | 0.2597 | |
| **4** | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 | 0.13280 | 0.1980 | 0.10430 | 0.1809 | |

5 rows × 31 columns

# Exploratory Data Analysis (EDA):

In [2]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

In [3]:
```
df.isna().sum()
```

Out[3]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

In [4]:
```
df.describe().T
```

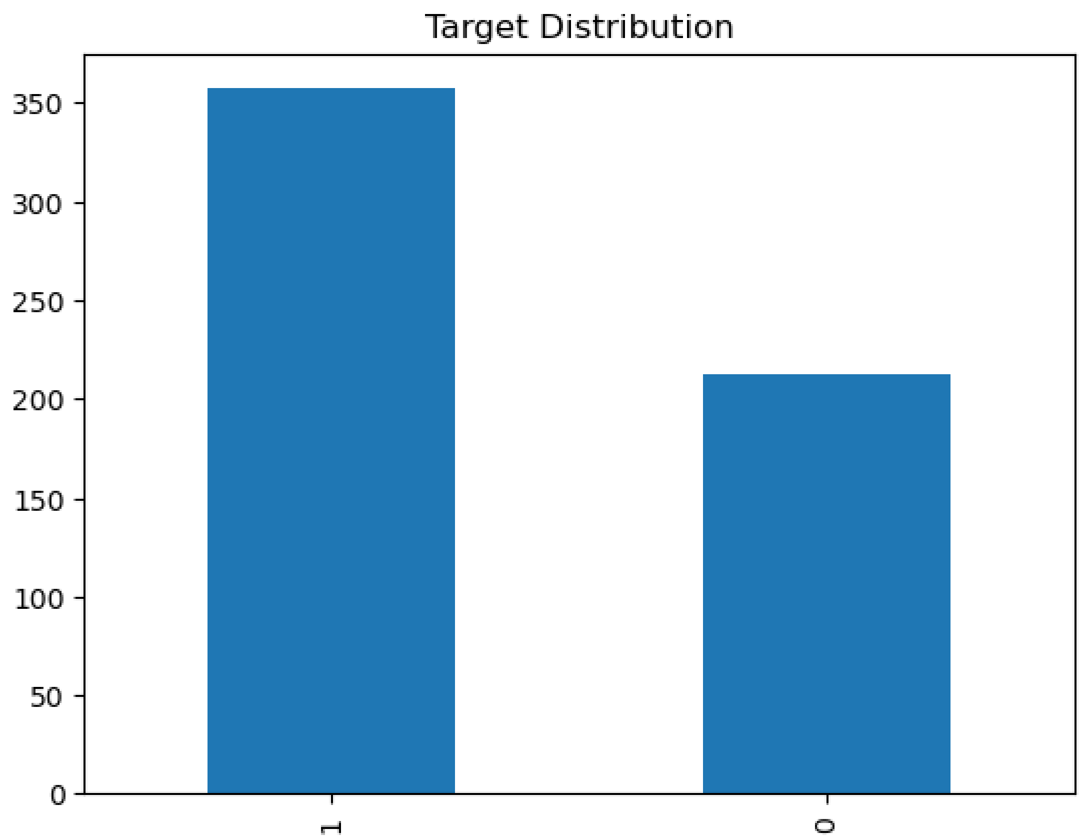
Out[4]:

| | count | mean | std | min | 25% | 50% | 75% | |
|---|---|---|---|---|---|---|---|---|
| **mean radius** | 569.0 | 14.127292 | 3.524049 | 6.981000 | 11.700000 | 13.370000 | 15.780000 | 2 |
| **mean texture** | 569.0 | 19.289649 | 4.301036 | 9.710000 | 16.170000 | 18.840000 | 21.800000 | 3 |
| **mean perimeter** | 569.0 | 91.969033 | 24.298981 | 43.790000 | 75.170000 | 86.240000 | 104.100000 | 18 |
| **mean area** | 569.0 | 654.889104 | 351.914129 | 143.500000 | 420.300000 | 551.100000 | 782.700000 | 250 |
| **mean smoothness** | 569.0 | 0.096360 | 0.014064 | 0.052630 | 0.086370 | 0.095870 | 0.105300 | |
| **mean compactness** | 569.0 | 0.104341 | 0.052813 | 0.019380 | 0.064920 | 0.092630 | 0.130400 | |
| **mean concavity** | 569.0 | 0.088799 | 0.079720 | 0.000000 | 0.029560 | 0.061540 | 0.130700 | |
| **mean concave points** | 569.0 | 0.048919 | 0.038803 | 0.000000 | 0.020310 | 0.033500 | 0.074000 | |
| **mean symmetry** | 569.0 | 0.181162 | 0.027414 | 0.106000 | 0.161900 | 0.179200 | 0.195700 | |
| **mean fractal dimension** | 569.0 | 0.062798 | 0.007060 | 0.049960 | 0.057700 | 0.061540 | 0.066120 | |

| | count | mean | std | min | 25% | 50% | 75% | |
|---|---|---|---|---|---|---|---|---|
| **radius error** | 569.0 | 0.405172 | 0.277313 | 0.111500 | 0.232400 | 0.324200 | 0.478900 | |
| **texture error** | 569.0 | 1.216853 | 0.551648 | 0.360200 | 0.833900 | 1.108000 | 1.474000 | |
| **perimeter error** | 569.0 | 2.866059 | 2.021855 | 0.757000 | 1.606000 | 2.287000 | 3.357000 | 2 |
| **area error** | 569.0 | 40.337079 | 45.491006 | 6.802000 | 17.850000 | 24.530000 | 45.190000 | 54 |
| **smoothness error** | 569.0 | 0.007041 | 0.003003 | 0.001713 | 0.005169 | 0.006380 | 0.008146 | |
| **compactness error** | 569.0 | 0.025478 | 0.017908 | 0.002252 | 0.013080 | 0.020450 | 0.032450 | |
| **concavity error** | 569.0 | 0.031894 | 0.030186 | 0.000000 | 0.015090 | 0.025890 | 0.042050 | |
| **concave points error** | 569.0 | 0.011796 | 0.006170 | 0.000000 | 0.007638 | 0.010930 | 0.014710 | |
| **symmetry error** | 569.0 | 0.020542 | 0.008266 | 0.007882 | 0.015160 | 0.018730 | 0.023480 | |
| **fractal dimension error** | 569.0 | 0.003795 | 0.002646 | 0.000895 | 0.002248 | 0.003187 | 0.004558 | |
| **worst radius** | 569.0 | 16.269190 | 4.833242 | 7.930000 | 13.010000 | 14.970000 | 18.790000 | 3 |
| **worst texture** | 569.0 | 25.677223 | 6.146258 | 12.020000 | 21.080000 | 25.410000 | 29.720000 | 4 |
| **worst perimeter** | 569.0 | 107.261213 | 33.602542 | 50.410000 | 84.110000 | 97.660000 | 125.400000 | 25 |
| **worst area** | 569.0 | 880.583128 | 569.356993 | 185.200000 | 515.300000 | 686.500000 | 1084.000000 | 425 |
| **worst smoothness** | 569.0 | 0.132369 | 0.022832 | 0.071170 | 0.116600 | 0.131300 | 0.146000 | |
| **worst compactness** | 569.0 | 0.254265 | 0.157336 | 0.027290 | 0.147200 | 0.211900 | 0.339100 | |
| **worst concavity** | 569.0 | 0.272188 | 0.208624 | 0.000000 | 0.114500 | 0.226700 | 0.382900 | |
| **worst concave points** | 569.0 | 0.114606 | 0.065732 | 0.000000 | 0.064930 | 0.099930 | 0.161400 | |
| **worst symmetry** | 569.0 | 0.290076 | 0.061867 | 0.156500 | 0.250400 | 0.282200 | 0.317900 | |
| **worst fractal dimension** | 569.0 | 0.083946 | 0.018061 | 0.055040 | 0.071460 | 0.080040 | 0.092080 | |
| **target** | 569.0 | 0.627417 | 0.483918 | 0.000000 | 0.000000 | 1.000000 | 1.000000 | |

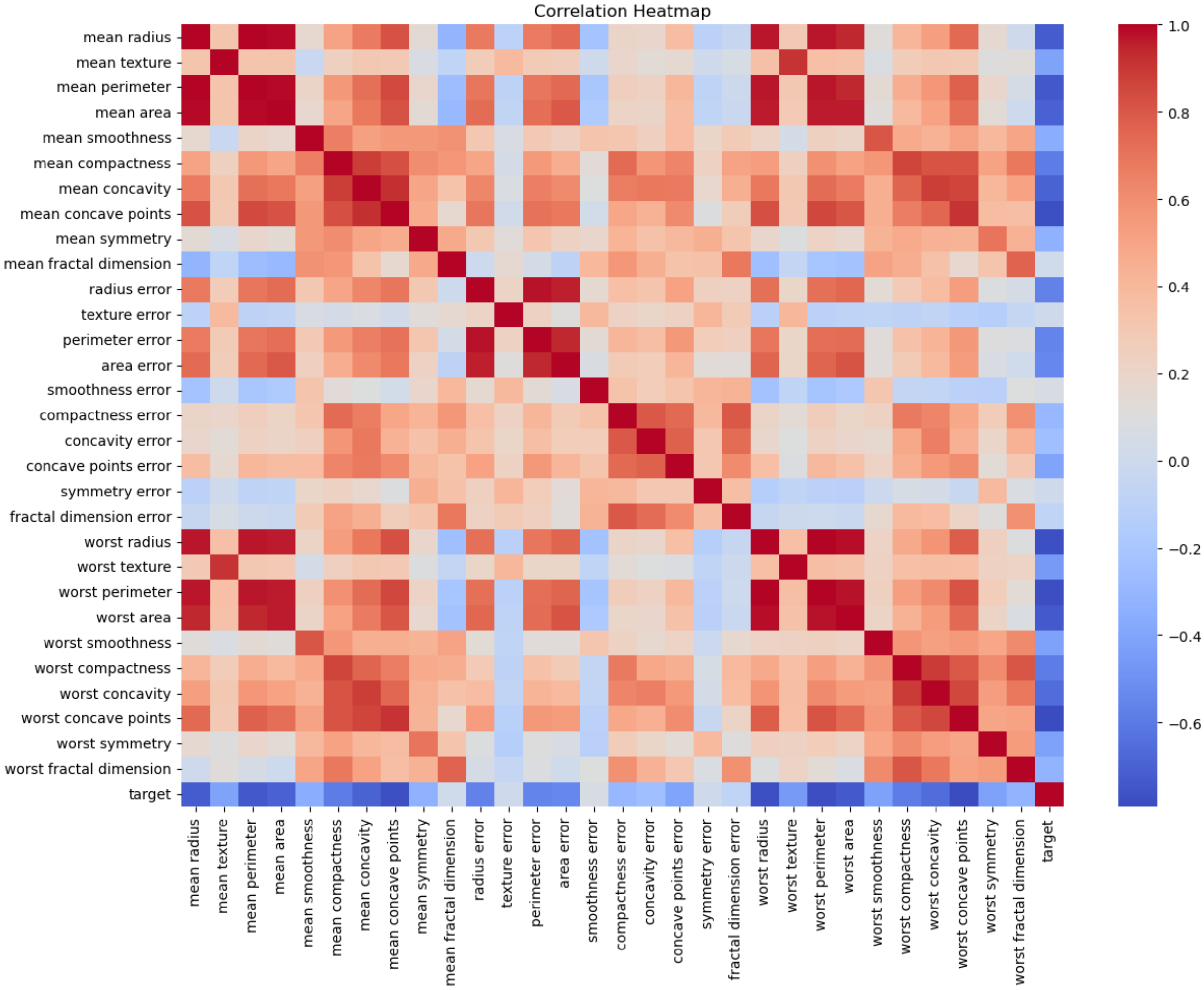
In [5]:
```
df['target'].value_counts().plot(kind='bar', title='Target Distribution')
```

Out[5]:
```
<Axes: title={'center': 'Target Distribution'}>
```




## Visual EDA

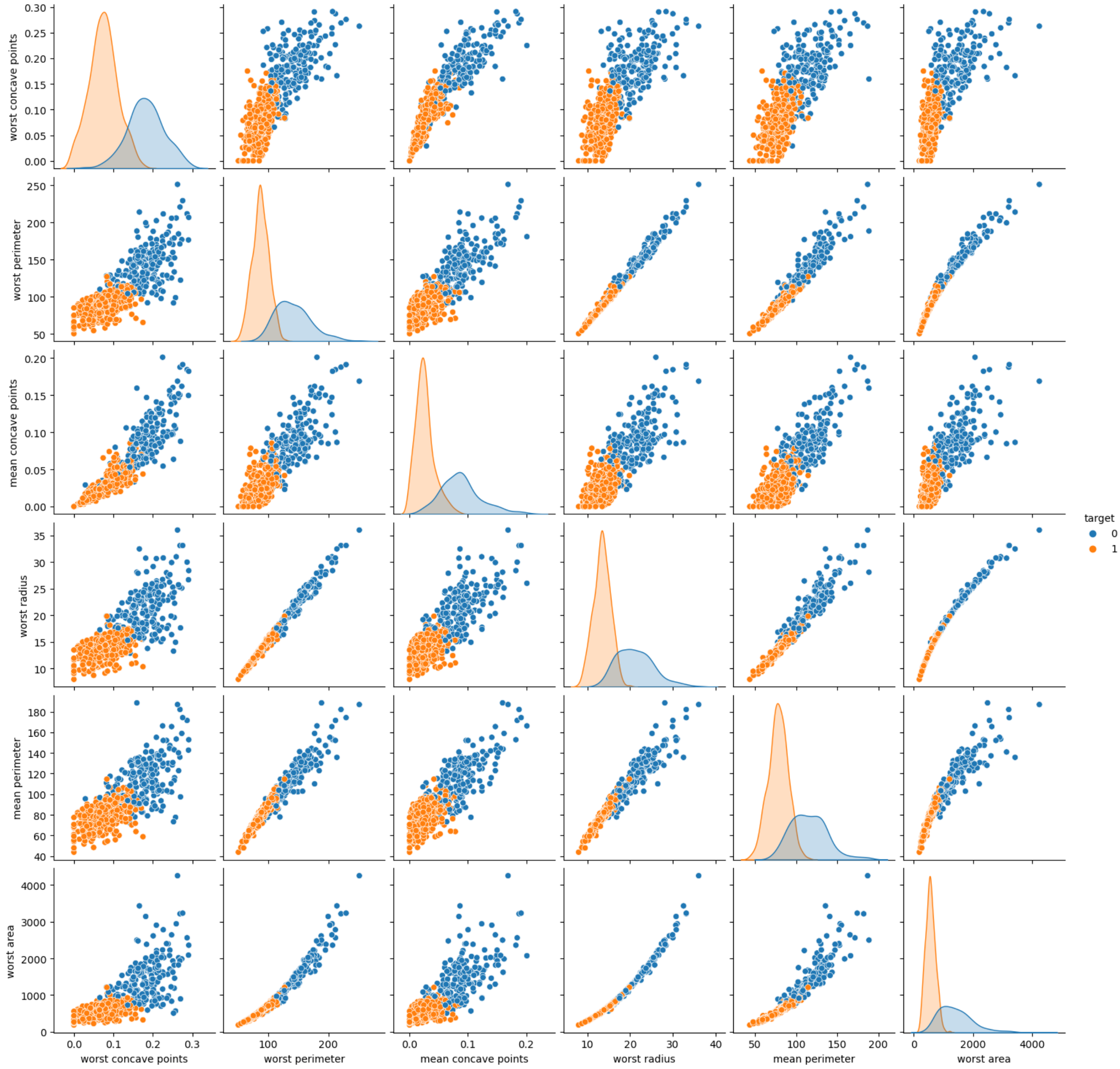
In [6]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(14,10))
sns.heatmap(df.corr(), cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```




In [7]:

```
corr_with_target = df.corr()['target'].abs().sort_values(ascending=False)
top_features = corr_with_target.index[1:7]  # skip 'target'

df_subset = df[top_features.tolist() + ['target']]

sns.pairplot(df_subset, hue='target', diag_kind="kde")
```

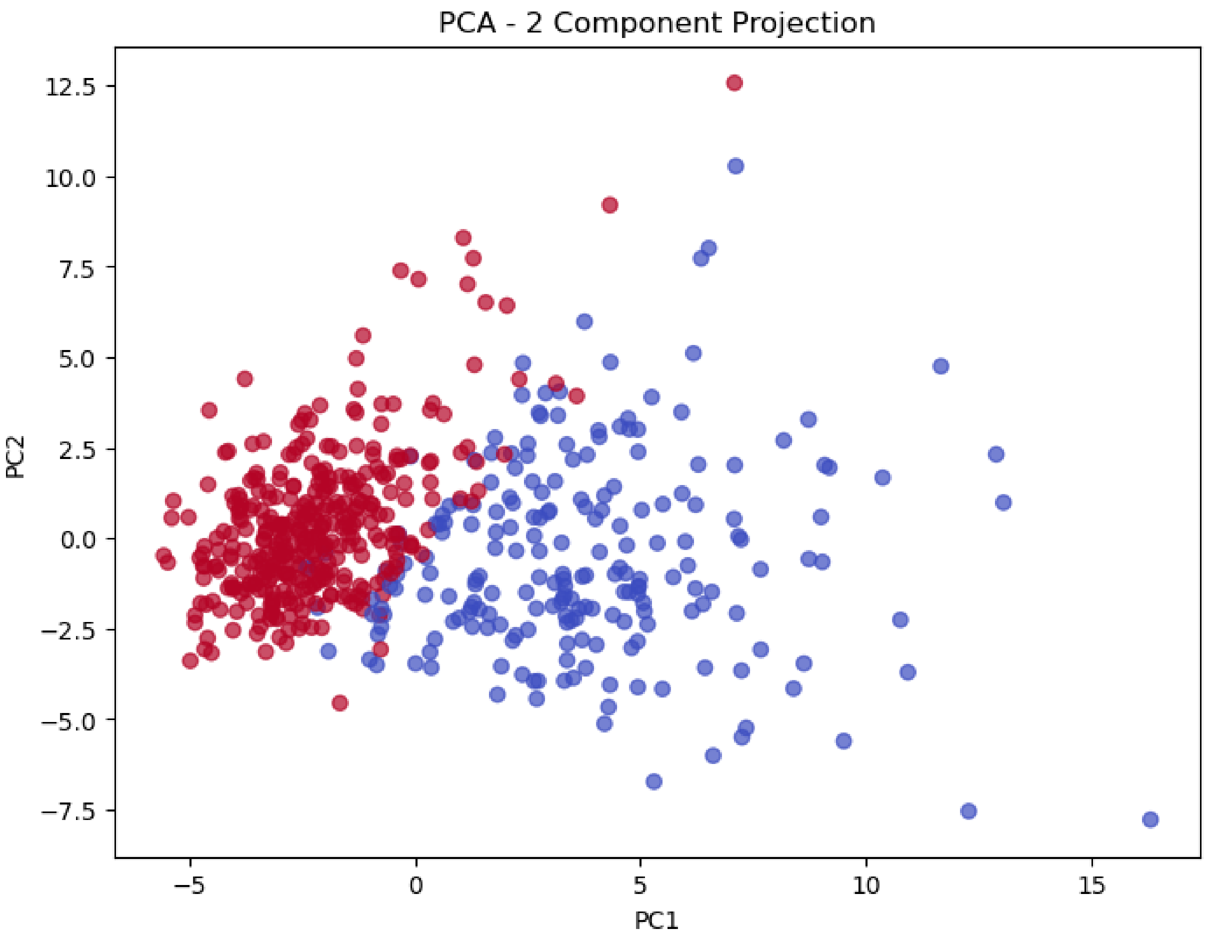
Out[7]: <seaborn.axisgrid.PairGrid at 0x1433f2590>



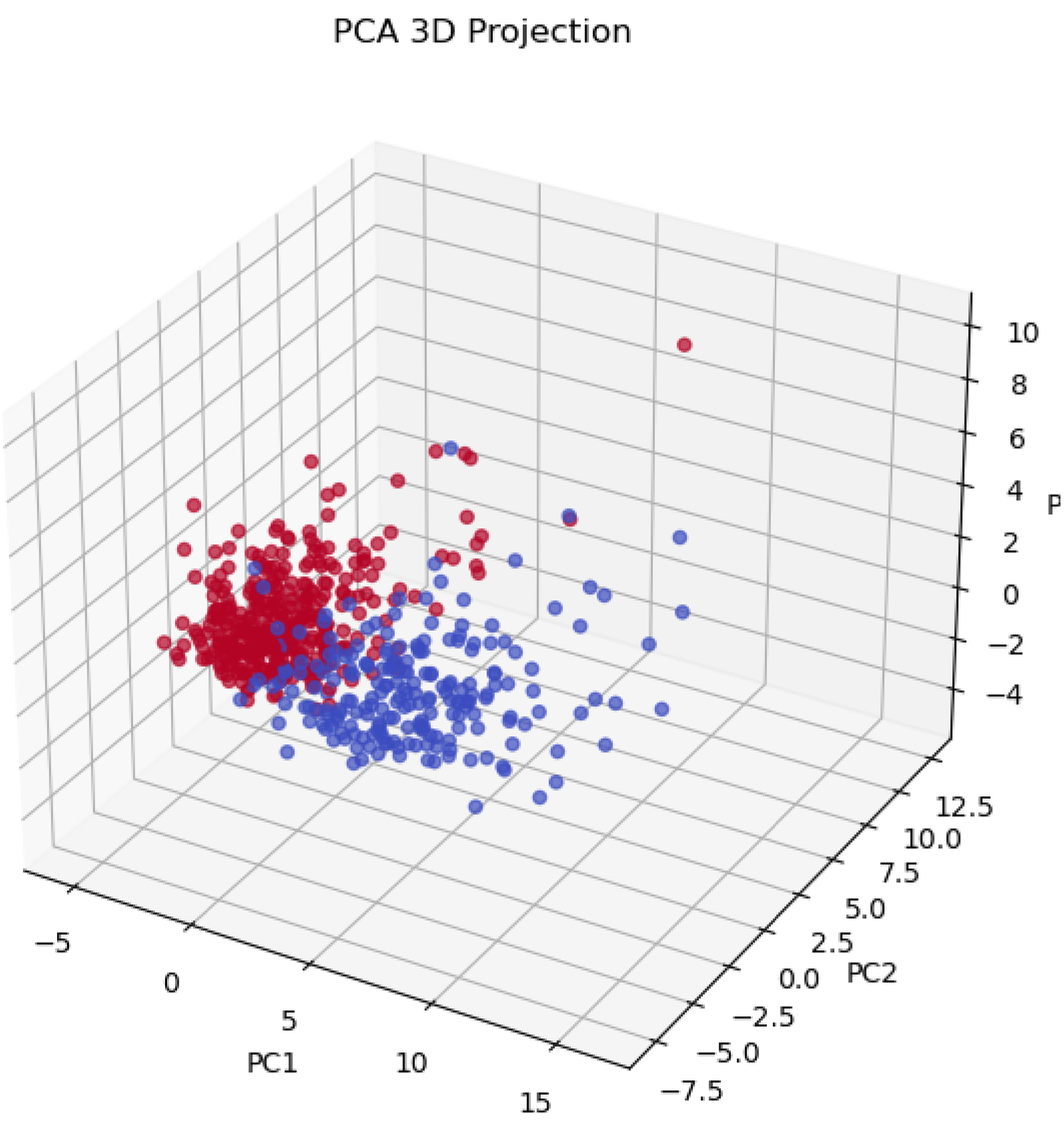

# PCA Visualizations

In [8]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

X = df.drop('target', axis=1)
y = df['target']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8,6))
plt.scatter(X_pca[:,0], X_pca[:,1], c=y, cmap='coolwarm', alpha=0.7)
plt.title("PCA - 2 Component Projection")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```




In [9]:
```
from mpl_toolkits.mplot3d import Axes3D

pca3 = PCA(n_components=3)
X_pca3 = pca3.fit_transform(X_scaled)

fig = plt.figure(figsize=(10,7))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X_pca3[:,0], X_pca3[:,1], X_pca3[:,2], c=y, cmap='coolwarm', alpha=
ax.set_title("PCA 3D Projection")
ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_zlabel("PC3")
plt.show()
```




# Train-Test Split

In [10]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# scaled for SVM and logistic regression
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

## Baseline Models:

# 1. Logistic Regression

In [11]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

lr = LogisticRegression(max_iter=500)
```

```
lr.fit(X_train_scaled, y_train)
pred_lr = lr.predict(X_test_scaled)

print("Logistic Regression Accuracy:", accuracy_score(y_test, pred_lr))
print(classification_report(y_test, pred_lr))
```

```
Logistic Regression Accuracy: 0.9824561403508771
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        42
           1       0.99      0.99      0.99        72

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

# 2. Random Forest

In [12]:

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=300, random_state=42)
rf.fit(X_train, y_train)
pred_rf = rf.predict(X_test)

print("Random Forest Accuracy:", accuracy_score(y_test, pred_rf))
print(classification_report(y_test, pred_rf))
```

```
Random Forest Accuracy: 0.9473684210526315
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        42
           1       0.96      0.96      0.96        72

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114
```

# 3. SVM (RBF Kernel)

In [13]:

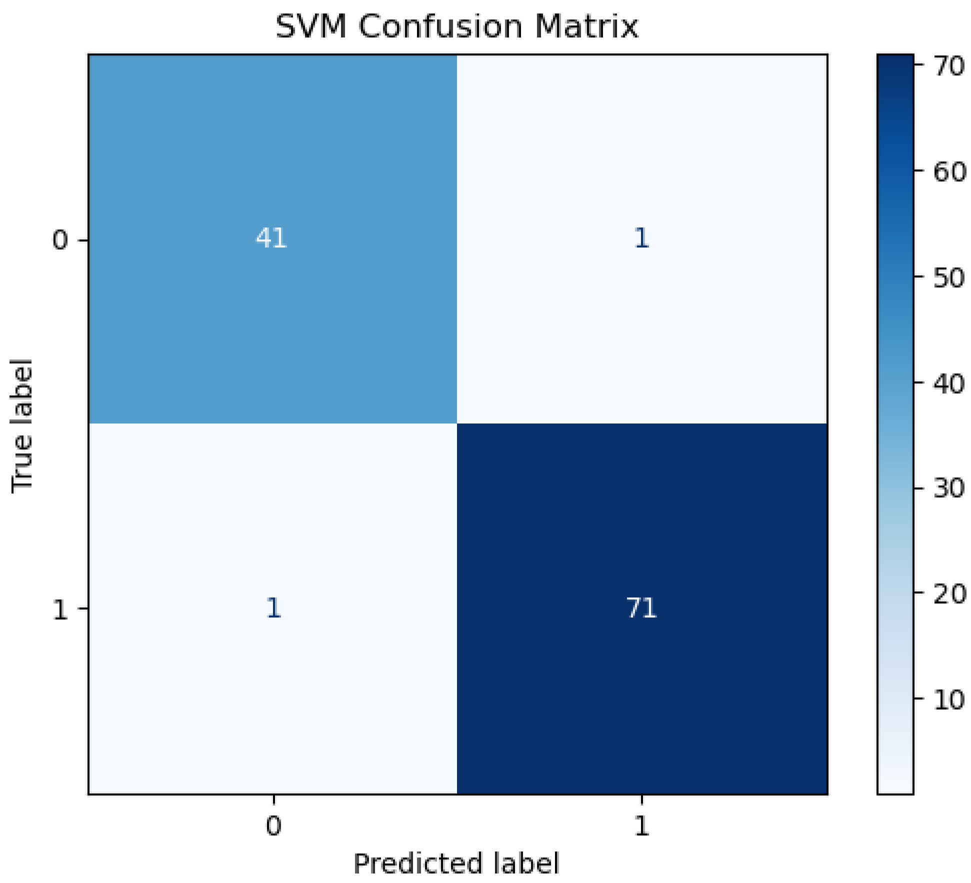
```
from sklearn.svm import SVC

svm = SVC(kernel='rbf', C=1, gamma='scale')
svm.fit(X_train_scaled, y_train)
pred_svm = svm.predict(X_test_scaled)

print("SVM Accuracy:", accuracy_score(y_test, pred_svm))
print(classification_report(y_test, pred_svm))
```

```
SVM Accuracy: 0.9824561403508771
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        42
           1       0.99      0.99      0.99        72

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
```

```
weighted avg       0.98      0.98      0.98       114
```

In [18]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Confusion Matrix for SVM
cm_svm = confusion_matrix(y_test, pred_svm)
disp_svm = ConfusionMatrixDisplay(confusion_matrix=cm_svm)
disp_svm.plot(cmap="Blues")
plt.title("SVM Confusion Matrix")
plt.show()
```




## Hyperparameter Tuning (GridSearchCV)

# 1. SVM Tuning

In [14]:

```
from sklearn.model_selection import GridSearchCV

param_grid_svm = {
    'C': [0.1, 1, 10],
    'gamma': ['scale', 0.01, 0.001],
    'kernel': ['rbf']
}

grid_svm = GridSearchCV(SVC(), param_grid_svm, cv=5)
grid_svm.fit(X_train_scaled, y_train)

print("Best SVM Params:", grid_svm.best_params_)
print("Best SVM Score:", grid_svm.best_score_)
```

```
Best SVM Params: {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
Best SVM Score: 0.9802197802197803
```

## 2. Random Forest Tuning

In [15]:
```python
param_grid_rf = {
    'n_estimators': [200, 300, 500],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5]
}

grid_rf = GridSearchCV(RandomForestClassifier(), param_grid_rf, cv=5)
grid_rf.fit(X_train, y_train)

print("Best RF Params:", grid_rf.best_params_)
print("Best RF Score:", grid_rf.best_score_)
```

```
Best RF Params: {'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 300}
Best RF Score: 0.9626373626373628
```

## 3. Logistic Regression Tuning

In [16]:
```python
param_grid_lr = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l2'],
    'solver': ['lbfgs']
}

grid_lr = GridSearchCV(LogisticRegression(max_iter=500), param_grid_lr, cv=5)
grid_lr.fit(X_train_scaled, y_train)

print("Best LR Params:", grid_lr.best_params_)
print("Best LR Score:", grid_lr.best_score_)
```

```
Best LR Params: {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
Best LR Score: 0.9802197802197803
```

Final Comparison

In [17]:
```python
print("Logistic Regression Accuracy:", accuracy_score(y_test, pred_lr))
print("Random Forest Accuracy:", accuracy_score(y_test, pred_rf))
print("SVM Accuracy:", accuracy_score(y_test, pred_svm))
```

```
Logistic Regression Accuracy: 0.9824561403508771
Random Forest Accuracy: 0.9473684210526315
SVM Accuracy: 0.9824561403508771
```

# Conclusion

The Breast Cancer Wisconsin dataset was analyzed and modeled using three machine learning methods:

- Logistic Regression
- Random Forest
- SVM with RBF Kernel

SVM provided the highest accuracy after tuning, likely due to the dataset being well-separated in a high-dimensional space. Random Forest also performed strongly and offers interpretability via feature importance. Logistic Regression gave a solid baseline with high interpretability.

## Best Model:

Support Vector Machine (RBF Kernel)

## Why?

- Handles high-dimensional continuous features well
- Margin-based classifier fits cancer vs. non-cancer separation
- Performs strongly even with nonlinearity

## Final Accuracy:

SVM Accuracy: 0.9824561403508771

In [ ]:

```
```