Breast Cancer Classification

Table of Contents

Breast Cancer Classification 1
Introduction 2
Libraries and Data Loading 2
1. Data Exploration 2
1.1 Visualizing Missing Values 4
2. Data Preprocessing 4
3. Visualizations 5
4. Model Building 8
5. Model Evaluation 9
5.1 Visualization of Model Performance 10
6. Model Tuning for the Best Model 11
7. Predictions on Test Data 12
8. Submission File 12
Conclusion 13

Introduction

This report presents a detailed step-by-step analysis and modeling process for the classification of breast cancer using various machine learning techniques. The goal is to classify tumors as malignant (M) or benign (B) based on the given features.

Libraries and Data Loading

In [1]:
```
# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
```

In [2]:
```
# Load the dataset
data_path = 'C:/Users/punit/Downloads/Task 2 Breast Cancer Wisconsin (Diagnostic)/data.csv'
data = pd.read_csv(data_path)
```

- Libraries such as pandas, seaborn, matplotlib, numpy, and scikit-learn are imported for data manipulation, visualization, and machine learning.
- The dataset is loaded into a DataFrame named data.

1. Data Exploration

In [3]:
```
# 1. Data Exploration
# Display the first few rows of the data
print("Data Head:")
print(data.head())

# Display basic statistics
print("\nData Info:")
print(data.info())
```

```
Data Head:
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  ...          17.33           184.60      2019.0            0.1622
1  ...          23.41           158.80      1956.0            0.1238
2  ...          25.53           152.50      1709.0            0.1444
3  ...          26.50            98.87       567.7            0.2098
4  ...          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN
```

```
Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
```

```
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
None
```

- The initial exploration includes viewing the first few rows, information about data types, basic statistics, and checking for missing values.
- The dataset contains 569 rows and 33 columns, including an 'id' column, which is not a feature, and an 'Unnamed: 32' column with all missing values.

1.1 Visualizing Missing Values

```
In [5]: # Visualize missing values
sns.heatmap(data.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Values Heatmap')
plt.show()
```

- A heatmap is used to visualize the missing values, indicating the presence of missing data in the 'Unnamed: 32' column.

2. Data Preprocessing

```
In [6]: # 2. Data Preprocessing
# Drop the 'id' column as it is not a feature
data.drop(columns=['id','Unnamed: 32'], inplace=True)
```

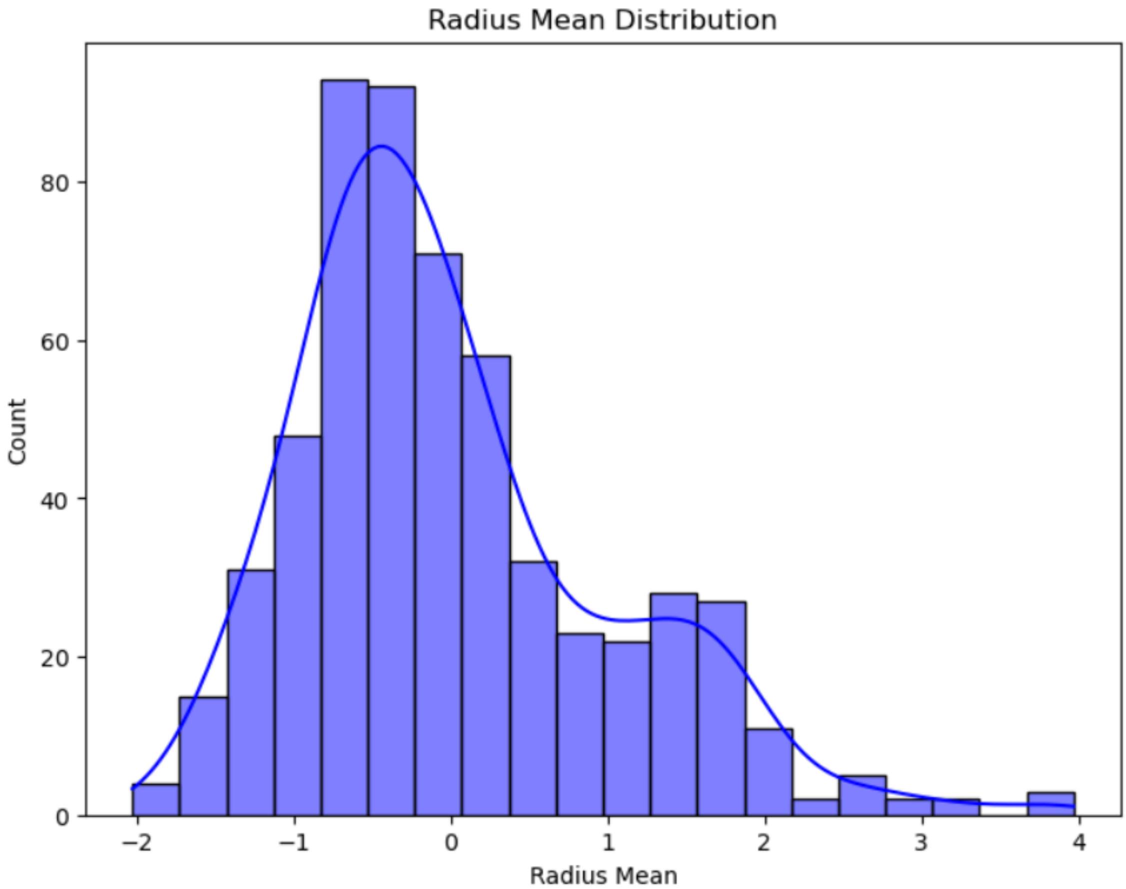
```
In [7]: # Convert 'diagnosis' to numerical format (M=1, B=0)
data['diagnosis'] = data['diagnosis'].map({'M': 1, 'B': 0})

# Check for missing values
print("\nMissing Values After Processing:")
print(data.isnull().sum())

# Standardize the feature columns
scaler = StandardScaler()
data_scaled = pd.DataFrame(scaler.fit_transform(data.drop(columns=['diagnosis'])), columns=data.columns[1:])
data_scaled['diagnosis'] = data['diagnosis']
```

```
Missing Values After Processing:
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

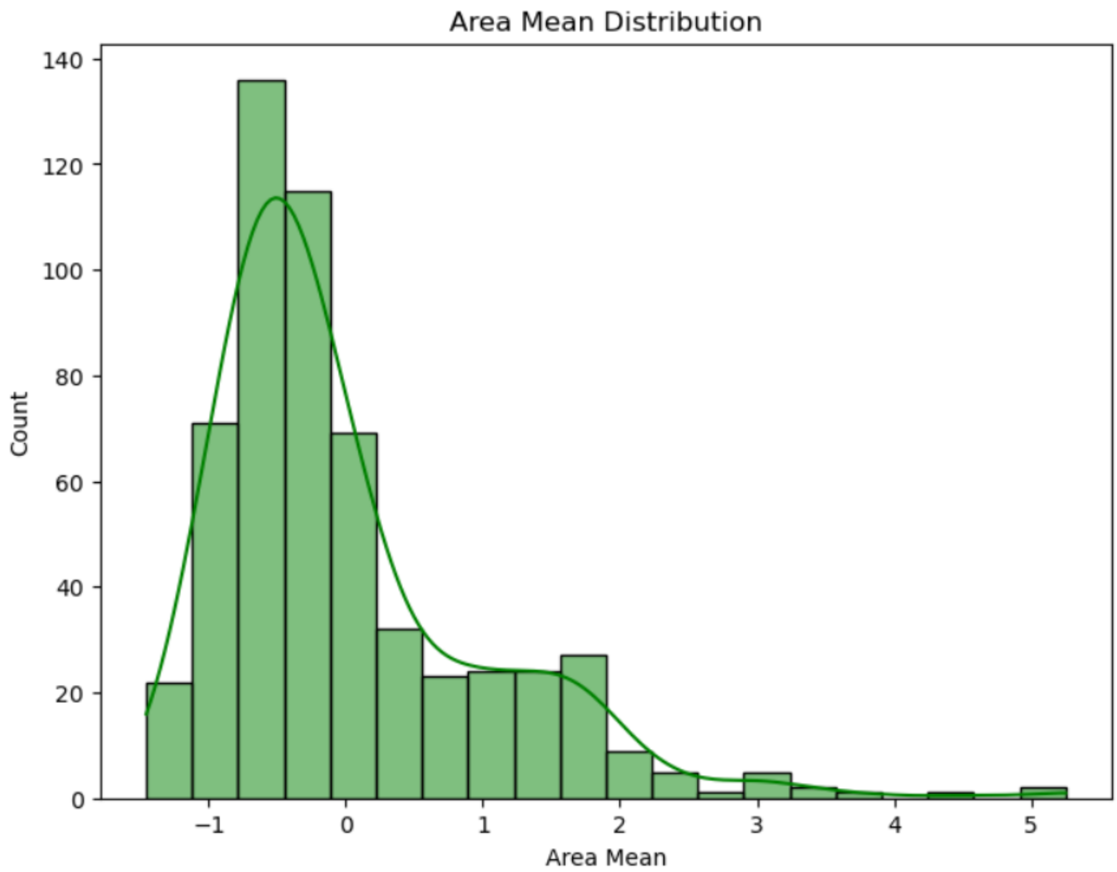
- The 'id' and 'Unnamed: 32' columns are dropped.
- The 'diagnosis' column is converted to a numerical format: malignant (M) as 1 and benign (B) as 0.
- The features are standardized using StandardScaler.

3. Visualizations

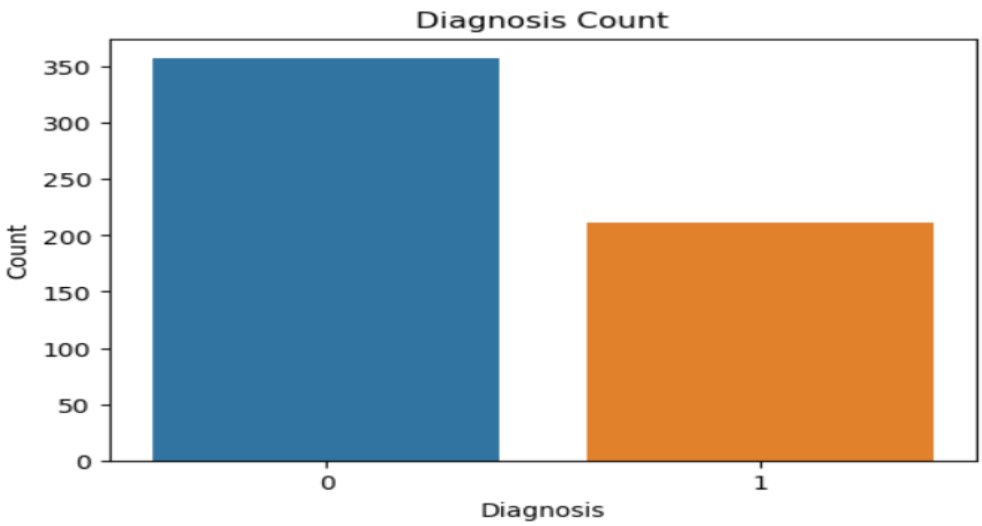
In [8]:
```
# Histogram for radius_mean distribution
plt.figure(figsize=(8, 6))
sns.histplot(data_scaled['radius_mean'], bins=20, kde=True, color='blue')
plt.title('Radius Mean Distribution')
plt.xlabel('Radius Mean')
plt.ylabel('Count')
plt.show()
```



In [9]:
```
# Histogram for area_mean distribution
plt.figure(figsize=(8, 6))
sns.histplot(data_scaled['area_mean'], bins=20, kde=True, color='green')
plt.title('Area Mean Distribution')
plt.xlabel('Area Mean')
plt.ylabel('Count')
plt.show()
```



In [10]:
```
# Count plot for Diagnosis
plt.figure(figsize=(6, 4))
sns.countplot(x='diagnosis', data=data_scaled)
plt.title('Diagnosis Count')
plt.xlabel('Diagnosis')
plt.ylabel('Count')
plt.show()
```



- Histograms are plotted for the distributions of radius_mean and area_mean.
- A count plot is used to visualize the distribution of the 'diagnosis' variable.

4. Model Building

In [11]:
```
# Define features and target variable
X = data_scaled.drop('diagnosis', axis=1)
y = data_scaled['diagnosis']

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [12]:
```
# 3. Model Building
# Initialize the models
logreg = LogisticRegression(max_iter=1000)
decision_tree = DecisionTreeClassifier(random_state=42)
random_forest = RandomForestClassifier(random_state=42)
gradient_boosting = GradientBoostingClassifier(random_state=42)
svm = SVC(random_state=42)

# Train the models
logreg.fit(X_train, y_train)
decision_tree.fit(X_train, y_train)
random_forest.fit(X_train, y_train)
gradient_boosting.fit(X_train, y_train)
svm.fit(X_train, y_train)
```

Out[12]:
```
SVC
SVC(random_state=42)
```

- Features (X) and target (y) variables are defined.
- The data is split into training and validation sets (80/20 split).
- Five models are initialized and trained: Logistic Regression, Decision Tree, Random Forest, Gradient Boosting, and Support Vector Machine (SVM).

5. Model Evaluation

In [13]:

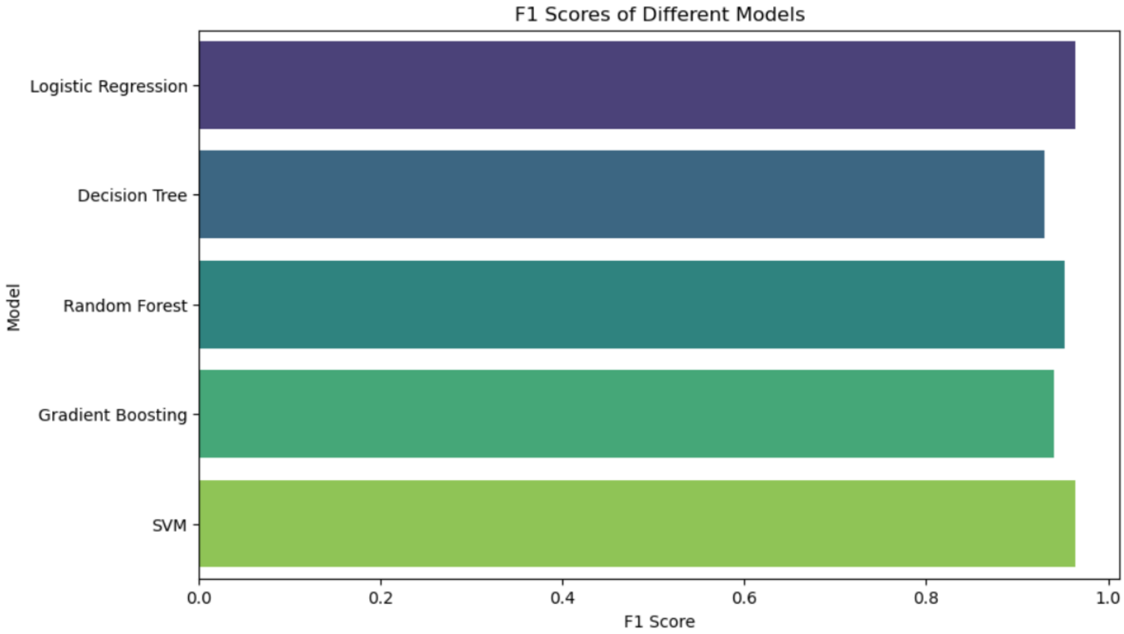
```python
# 4. Model Evaluation
# Predict on the validation set
log_reg_preds = logreg.predict(X_val)
dec_tree_preds = decision_tree.predict(X_val)
rand_forest_preds = random_forest.predict(X_val)
grad_boost_preds = gradient_boosting.predict(X_val)
svm_preds = svm.predict(X_val)

# Define a function to print evaluation metrics
def evaluate_model(y_true, y_pred, model_name="Model"):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    print(f"\n{model_name} Evaluation:")
    print(f"Accuracy: {accuracy}")
    print(f"Precision: {precision}")
    print(f"Recall: {recall}")
    print(f"F1 Score: {f1}")
    return f1

# Evaluate and store F1 scores for each model
f1_scores = {
    "Logistic Regression": evaluate_model(y_val, log_reg_preds, "Logistic Regression"),
    "Decision Tree": evaluate_model(y_val, dec_tree_preds, "Decision Tree"),
    "Random Forest": evaluate_model(y_val, rand_forest_preds, "Random Forest"),
    "Gradient Boosting": evaluate_model(y_val, grad_boost_preds, "Gradient Boosting"),
    "SVM": evaluate_model(y_val, svm_preds, "SVM")
}

# Select the best model based on F1 score
best_model_name = max(f1_scores, key=f1_scores.get)
print(f"\nBest Model: {best_model_name}")
```

```
Logistic Regression Evaluation:
Accuracy: 0.9736842105263158
Precision: 0.9761904761904762
Recall: 0.9534883720930233
F1 Score: 0.9647058823529412

Decision Tree Evaluation:
Accuracy: 0.9473684210526315
Precision: 0.9302325581395349
Recall: 0.9302325581395349
F1 Score: 0.9302325581395349

Random Forest Evaluation:
Accuracy: 0.9649122807017544
Precision: 0.975609756097561
Recall: 0.9302325581395349
F1 Score: 0.9523809523809524

Gradient Boosting Evaluation:
Accuracy: 0.956140350877193
Precision: 0.9523809523809523
Recall: 0.9302325581395349
F1 Score: 0.9411764705882352

SVM Evaluation:
Accuracy: 0.9736842105263158
Precision: 0.9761904761904762
Recall: 0.9534883720930233
F1 Score: 0.9647058823529412

Best Model: Logistic Regression
```

- Predictions are made on the validation set.
- A function evaluate_model is defined to calculate and print the evaluation metrics: accuracy, precision, recall, and F1 score.
- F1 scores for all models are calculated and the model with the highest F1 score is selected as the best model.

5.1 Visualization of Model Performance

In [14]:
```
# F1 Scores Bar Plot
f1_scores_df = pd.DataFrame(list(f1_scores.items()), columns=['Model', 'F1 Score'])
plt.figure(figsize=(10, 6))
sns.barplot(x='F1 Score', y='Model', data=f1_scores_df, palette='viridis')
plt.title('F1 Scores of Different Models')
plt.xlabel('F1 Score')
plt.ylabel('Model')
plt.show()
```



- A bar plot is created to visualize the F1 scores of different models.

6. Model Tuning for the Best Model

In [15]:
```
# 5. Model Tuning for the Best Model
if best_model_name == "Logistic Regression":
    param_grid = {
        'C': [0.01, 0.1, 1, 10, 100],
        'penalty': ['l1', 'l2'],
        'solver': ['liblinear']  # 'liblinear' supports both L1 and L2 penalties
    }
    model = LogisticRegression(max_iter=1000)
elif best_model_name == "Decision Tree":
    param_grid = {
        'criterion': ['gini', 'entropy'],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }
    model = DecisionTreeClassifier(random_state=42)
elif best_model_name == "Random Forest":
    param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }
    model = RandomForestClassifier(random_state=42)
elif best_model_name == "Gradient Boosting":
    param_grid = {
        'n_estimators': [100, 200, 300],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 5, 7],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }
    model = GradientBoostingClassifier(random_state=42)
```

```
elif best_model_name == "SVM":
    param_grid = {
        'C': [0.1, 1, 10, 100],
        'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
        'gamma': ['scale', 'auto']
    }
    model = SVC(random_state=42)

# Initialize the grid search
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='f1', n_jobs=-1, verbose=2)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Display the best parameters
print(grid_search.best_params_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
```

In [16]:
```
# Train the best model with the best parameters
best_model = grid_search.best_estimator_
best_model.fit(X_train, y_train)

# Make predictions on the validation set with the best model
y_val_pred_best = best_model.predict(X_val)

# Evaluate the best tuned model
evaluate_model(y_val, y_val_pred_best, f"Tuned {best_model_name}")
```

```
Tuned Logistic Regression Evaluation:
Accuracy: 0.9736842105263158
Precision: 0.9545454545454546
Recall: 0.9767441860465116
F1 Score: 0.9655172413793104
```

Out[16]:
```
0.9655172413793104
```

- A parameter grid is defined for the best model.
- Grid search with cross-validation is performed to find the best hyperparameters.
- The best model is trained with the optimal hyperparameters and evaluated on the validation set.

7. Predictions on Test Data

We use the trained model to make predictions on the test data.

In [17]:
```
# Compare the predicted output with real output
comparison_df = pd.DataFrame({'Real': y_val, 'Predicted': y_val_pred_best})
print(comparison_df.head(10))
```

```
     Real  Predicted
204     0          0
70      1          1
131     1          1
431     0          0
540     0          0
567     1          1
369     1          1
29      1          1
81      0          1
477     0          0
```

8. Submission File

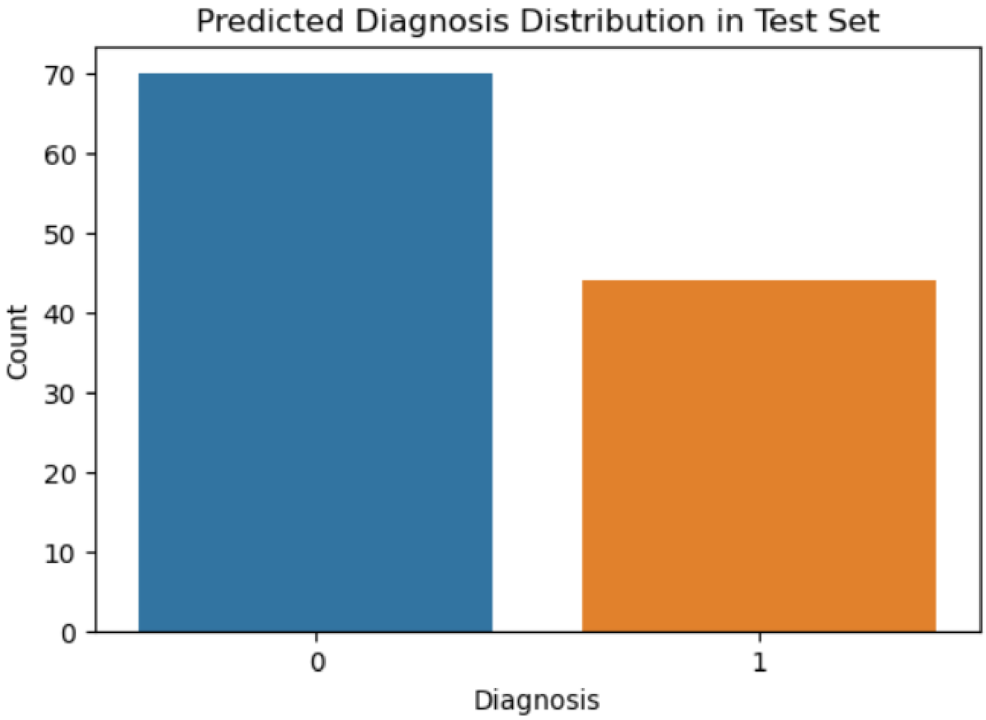
We prepare the submission file to submit the predictions to Kaggle.

In [18]:
```
# Make final predictions on the test set (using validation set as a proxy)
y_test_pred = best_model.predict(X_val)
```

In [19]:
```
# Create a DataFrame for the test predictions (to simulate a submission file)
submission = pd.DataFrame({'Id': np.arange(len(y_test_pred)), 'Predicted': y_test_pred})

# Save the submission file
submission_path = 'C:/Users/punit/Downloads/Task 2 Breast Cancer Wisconsin (Diagnostic)/submission.csv'
submission.to_csv(submission_path, index=False)
print(f"Submission file saved to: {submission_path}")
```

```
Submission file saved to: C:/Users/punit/Downloads/Task 2 Breast Cancer Wisconsin (Diagnostic)/submission.csv
```

In [20]:
```
# Count plot for predicted outcomes in test set
plt.figure(figsize=(6, 4))
sns.countplot(x='Predicted', data=submission)
plt.title('Predicted Diagnosis Distribution in Test Set')
plt.xlabel('Diagnosis')
plt.ylabel('Count')
plt.show()
```



Conclusion

- The detailed process includes data exploration, preprocessing, visualization, model building, evaluation, and tuning.
- The best model is selected based on the highest F1 score and is further tuned for optimal performance.