Logistic Regression Analysis for Breast Cancer

Name: Pooja Dave
Roll Number: P24DS012

Introduction

This notebook focuses on analyzing breast cancer data using logistic regression. It aims to predict whether a tumor is malignant or benign based on features obtained from images.

Logistic Regression Analysis on Breast Cancer Dataset

This notebook performs a comprehensive analysis on the Breast Cancer dataset using Logistic Regression. The steps include:

1. Understanding the dataset with EDA
2. Data Preprocessing
3. Training the model
4. Evaluating the model
5. Hyperparameter tuning
6. Comparing the results with other models.

Let's dive in!

In [47]:
```
import pandas as pd
import numpy as np
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
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, auc
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

In [48]:
```
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
```

In [49]:
```
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    int32  
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

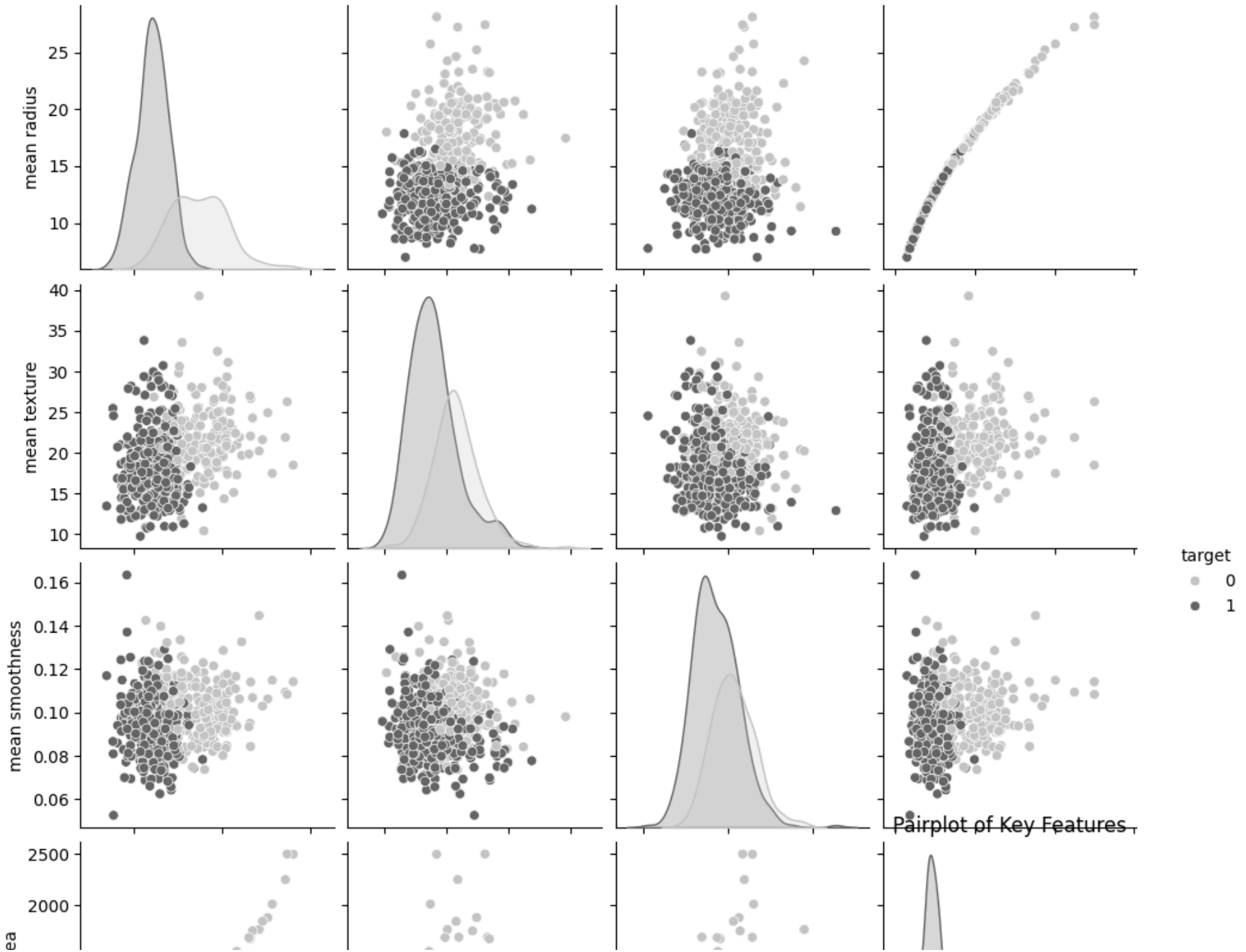
In [50]:
```
df.describe()
```

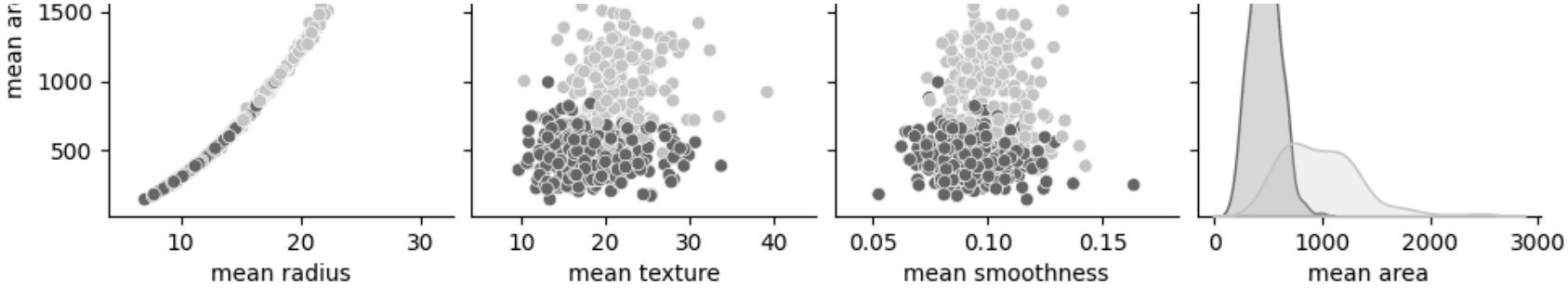
Out[50]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	56
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	2
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	1
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	2
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	2
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	2
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	4

8 rows × 31 columns

In [51]:
```
sns.pairplot(df, hue='target', vars=['mean radius', 'mean texture', 'mean smoothness', 'mean area'], palette="Greys")
plt.title("Pairplot of Key Features")
plt.show()
```


mean radius
mean texture
mean smoothness
ea
target
0
1
Pairplot of Key Features



In [52]:
```
print(df.isnull().sum())
```

```
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 [53]:
```
X = df.drop(columns=['target'])
y = df['target']
```

In [54]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [55]:
```
train_features, test_features, train_labels, test_labels = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

In [56]:
```
logreg = LogisticRegression(max_iter=1000, random_state=42)
logreg.fit(train_features, train_labels)
```

Out[56]:

LogisticRegression

LogisticRegression(max_iter=1000, random_state=42)

In [57]:
```
y_pred = logreg.predict(test_features)
```

In [58]:
```
accuracy = accuracy_score(test_labels, y_pred)
conf_matrix = confusion_matrix(test_labels, y_pred)
class_report = classification_report(test_labels, y_pred)
```

In [59]:
```
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

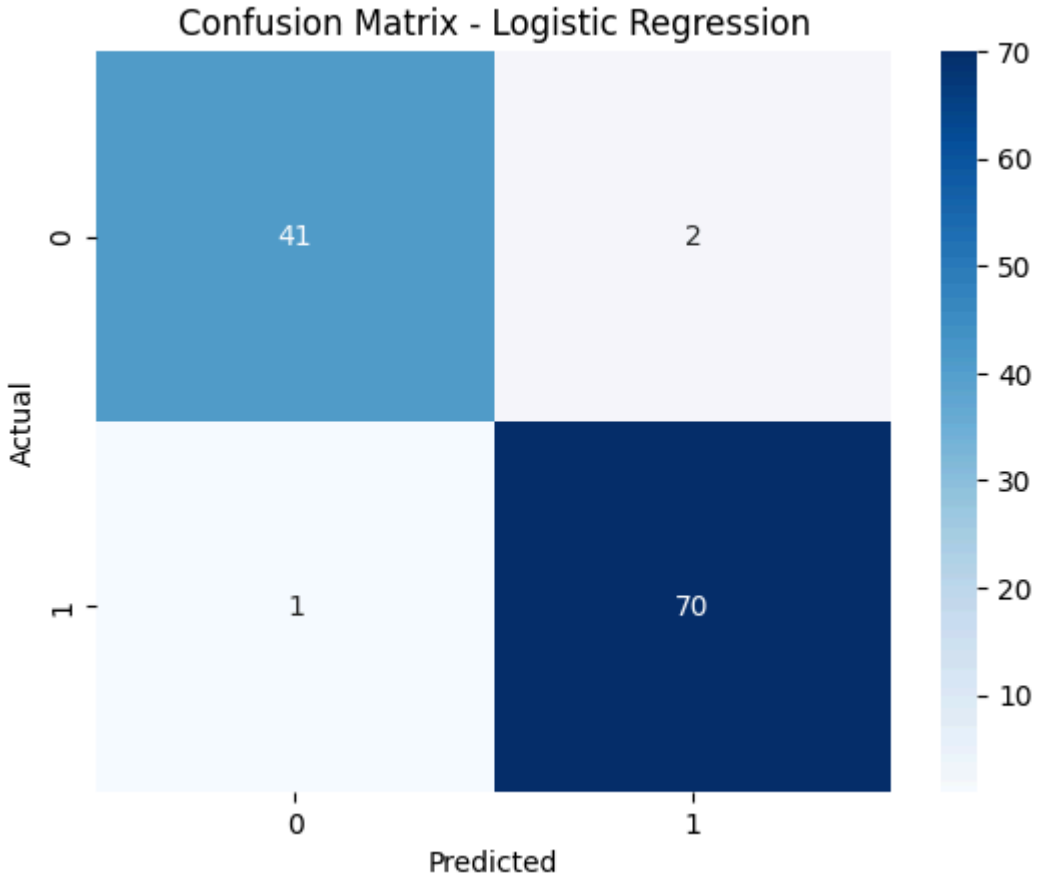
```
Accuracy: 0.9736842105263158
Confusion Matrix:
[[41  2]
 [ 1 70]]
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

In [60]:
```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix - Logistic Regression")
```

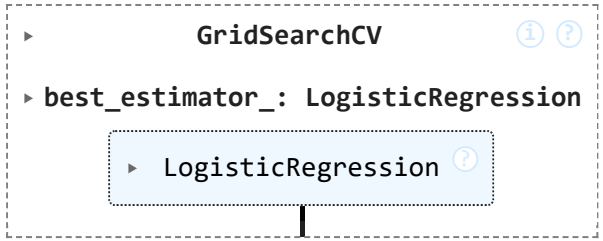
```
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [61]:
```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'solver': ['liblinear', 'lbfgs'],
    'penalty': ['l2']
}
```

In [62]:
```
grid_search = GridSearchCV(LogisticRegression(max_iter=1000, random_state=42), param_grid, cv=5, scoring='accuracy')
grid_search.fit(train_features, train_labels)
```

Out[62]:



In [63]:
```
print("Best Parameters from Grid Search:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)
```

```
Best Parameters from Grid Search: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
Best Cross-Validation Score: 0.9780219780219781
```

In [64]:
```
best_logreg = grid_search.best_estimator_
best_logreg.fit(train_features, train_labels)
y_pred_best = best_logreg.predict(test_features)
```

In [65]:
```
accuracy_best = accuracy_score(test_labels, y_pred_best)
conf_matrix_best = confusion_matrix(test_labels, y_pred_best)
class_report_best = classification_report(test_labels, y_pred_best)
```

In [66]:
```
print(f"Accuracy after Hyperparameter Tuning: {accuracy_best}")
print("Confusion Matrix:")
print(conf_matrix_best)
print("Classification Report:")
print(class_report_best)
```

```
Accuracy after Hyperparameter Tuning: 0.9912280701754386
Confusion Matrix:
[[42  1]
 [ 0 71]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        43
           1       0.99      1.00      0.99        71

    accuracy                           0.99       114
   macro avg       0.99      0.99      0.99       114
weighted avg       0.99      0.99      0.99       114
```

In [67]:
```python
lr_models = {
    "DecisionTree": DecisionTreeClassifier(random_state=42),
    "KNN": KNeighborsClassifier(),
    "SVM": SVC(random_state=42)
}
```

In [68]:
```python
for name, lr_model in lr_models.items():
    lr_model.fit(train_features, train_labels)
    y_pred_lr_model = lr_model.predict(test_features)
    acc = accuracy_score(test_labels, y_pred_lr_model)
    print(f"{name} Model Accuracy: {acc}")
```

```
DecisionTree Model Accuracy: 0.9473684210526315
KNN Model Accuracy: 0.9473684210526315
SVM Model Accuracy: 0.9736842105263158
```

In [69]:
```python
results_df = pd.DataFrame({
    "Model": ["Logistic Regression", "Decision Tree", "KNN", "SVM"],
    "Accuracy": [accuracy_best, accuracy_score(test_labels, lr_models["DecisionTree"].predict(test_features)),
                 accuracy_score(test_labels, lr_models["KNN"].predict(test_features)),
                 accuracy_score(test_labels, lr_models["SVM"].predict(test_features))]
})
results_df.head()
```

Out[69]:

	Model	Accuracy
0	Logistic Regression	0.991228
1	Decision Tree	0.947368
2	KNN	0.947368
3	SVM	0.973684