Notebook Information

- **Name:** Geron Simon A. Javier
- **Y&S:** BSCS 3B IS
- **Course:** CSST 102 | Basic Machine Learning
- **Topic:** Topic 2: Supervised Learning Techniques
- **Due date:** N/A

Laboratory Exercise #2: Exercises for K-Nearest Neighbors (KNN) and Logistic Regression on Breast Cancer Diagnosis Dataset

Exercise 1: Data Exploration and Preprocessing

[ ]:
```python
# Load necessary libraries
import pandas as pd

# Load the dataset, ensure flexibility in file path handling
df = pd.read_csv('Breast Cancer Diagnosis Dataset with Tumor Characteristics.
↪csv')

# Check column names for consistency
print("Column Names:", df.columns)

# Display the first 10 rows
print("First 10 Rows:")
print(df.head(10))

# Check for missing values
print("Missing Values per Column:")
print(df.isnull().sum())

# Descriptive statistics
print("Descriptive Statistics:")
print(df.describe())
```

```
Column Names: Index(['id', 'diagnosis', 'radius_mean', 'texture_mean',
'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
First 10 Rows:
```

	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	
5	843786	M	12.45	15.70	82.57	477.1	
6	844359	M	18.25	19.98	119.60	1040.0	
7	84458202	M	13.71	20.83	90.20	577.9	
8	844981	M	13.00	21.82	87.50	519.8	
9	84501001	M	12.46	24.04	83.97	475.9	

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	\
0	0.11840	0.27760	0.30010	0.14710	
1	0.08474	0.07864	0.08690	0.07017	
2	0.10960	0.15990	0.19740	0.12790	
3	0.14250	0.28390	0.24140	0.10520	
4	0.10030	0.13280	0.19800	0.10430	
5	0.12780	0.17000	0.15780	0.08089	
6	0.09463	0.10900	0.11270	0.07400	
7	0.11890	0.16450	0.09366	0.05985	
8	0.12730	0.19320	0.18590	0.09353	
9	0.11860	0.23960	0.22730	0.08543	

	...	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	
5	...	23.75	103.40	741.6	0.1791	
6	...	27.66	153.20	1606.0	0.1442	
7	...	28.14	110.60	897.0	0.1654	
8	...	30.73	106.20	739.3	0.1703	
9	...	40.68	97.65	711.4	0.1853	

	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	
5	0.5249	0.5355	0.1741	0.3985	
6	0.2576	0.3784	0.1932	0.3063	
7	0.3682	0.2678	0.1556	0.3196	
8	0.5401	0.5390	0.2060	0.4378	
9	1.0580	1.1050	0.2210	0.4366	

```
   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
5                  0.12440          NaN
6                  0.08368          NaN
7                  0.11510          NaN
8                  0.10720          NaN
9                  0.20750          NaN

[10 rows x 33 columns]
Missing Values per Column:
id                           0
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
Unnamed: 32                569
dtype: int64
```

```
Descriptive Statistics:
                 id  radius_mean  texture_mean  perimeter_mean    area_mean  \
count  5.690000e+02   569.000000    569.000000      569.000000   569.000000
mean   3.037183e+07    14.127292     19.289649       91.969033   654.889104
std    1.250206e+08     3.524049      4.301036       24.298981   351.914129
min    8.670000e+03     6.981000      9.710000       43.790000   143.500000
25%    8.692180e+05    11.700000     16.170000       75.170000   420.300000
50%    9.060240e+05    13.370000     18.840000       86.240000   551.100000
75%    8.813129e+06    15.780000     21.800000      104.100000   782.700000
max    9.113205e+08    28.110000     39.280000      188.500000  2501.000000

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
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

       symmetry_mean  ...  texture_worst  perimeter_worst   area_worst  \
count     569.000000  ...     569.000000       569.000000   569.000000
mean        0.181162  ...      25.677223       107.261213   880.583128
std         0.027414  ...       6.146258        33.602542   569.356993
min         0.106000  ...      12.020000        50.410000   185.200000
25%         0.161900  ...      21.080000        84.110000   515.300000
50%         0.179200  ...      25.410000        97.660000   686.500000
75%         0.195700  ...      29.720000       125.400000  1084.000000
max         0.304000  ...      49.540000       251.200000  4254.000000

       smoothness_worst  compactness_worst  concavity_worst  \
count        569.000000         569.000000       569.000000
mean           0.132369           0.254265         0.272188
std            0.022832           0.157336         0.208624
min            0.071170           0.027290         0.000000
25%            0.116600           0.147200         0.114500
50%            0.131300           0.211900         0.226700
75%            0.146000           0.339100         0.382900
max            0.222600           1.058000         1.252000

       concave points_worst  symmetry_worst  fractal_dimension_worst  \
count            569.000000      569.000000               569.000000
mean               0.114606        0.290076                 0.083946
std                0.065732        0.061867                 0.018061
min                0.000000        0.156500                 0.055040
25%                0.064930        0.250400                 0.071460
50%                0.099930        0.282200                 0.080040
```

```
75%                0.161400        0.317900                  0.092080
max                0.291000        0.663800                  0.207500

       Unnamed: 32
count          0.0
mean           NaN
std            NaN
min            NaN
25%            NaN
50%            NaN
75%            NaN
max            NaN

[8 rows x 32 columns]
```

[ ]:
```
#@title ## **Task: Summarize the Dataset:**

# Number of instances and features
print(f'Number of Instances: {df.shape[0]}')
print(f'Number of Features: {df.shape[1]}')

# Breakdown of target variable (diagnosis)
print("Diagnosis Breakdown (M = Malignant, B = Benign):")
print(df['diagnosis'].value_counts())

# Display missing values for further action
missing_values = df.isnull().sum()
print("Missing Values:")
print(missing_values[missing_values > 0])
```

```
Number of Instances: 569
Number of Features: 33
Diagnosis Breakdown (M = Malignant, B = Benign):
diagnosis
B    357
M    212
Name: count, dtype: int64
Missing Values:
Unnamed: 32    569
dtype: int64
```

[ ]:
```
#@title ## **3. Preprocessing:**

from sklearn.preprocessing import StandardScaler

# Drop irrelevant columns
if 'id' in df.columns:
    df = df.drop(columns=['id'])
```

```python
if 'Unnamed: 32' in df.columns:
    df = df.drop(columns=['Unnamed: 32'])

# Convert Diagnosis column to binary (M -> 1, B -> 0)
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})

# Check for any missing values before scaling
missing_values = df.isnull().sum()
if missing_values.any():
    print("There are still missing values. Consider handling them before scaling.
 ↪")
else:
    # Normalize features
    scaler = StandardScaler()
    features = df.drop(columns=['diagnosis'])
    scaled_features = scaler.fit_transform(features)
```

[ ]:
```python
#@title ## **4. Train-Test Split:**

from sklearn.model_selection import train_test_split

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    scaled_features, df['diagnosis'],
    test_size=0.2, random_state=42, stratify=df['diagnosis'])

# Print the sizes of training and testing sets
print(f'Training Set Size: {X_train.shape[0]} samples')
print(f'Testing Set Size: {X_test.shape[0]} samples')
```

```
Training Set Size: 455 samples
Testing Set Size: 114 samples
```

Exercise 2: Implementing K-Nearest Neighbors (KNN) Model

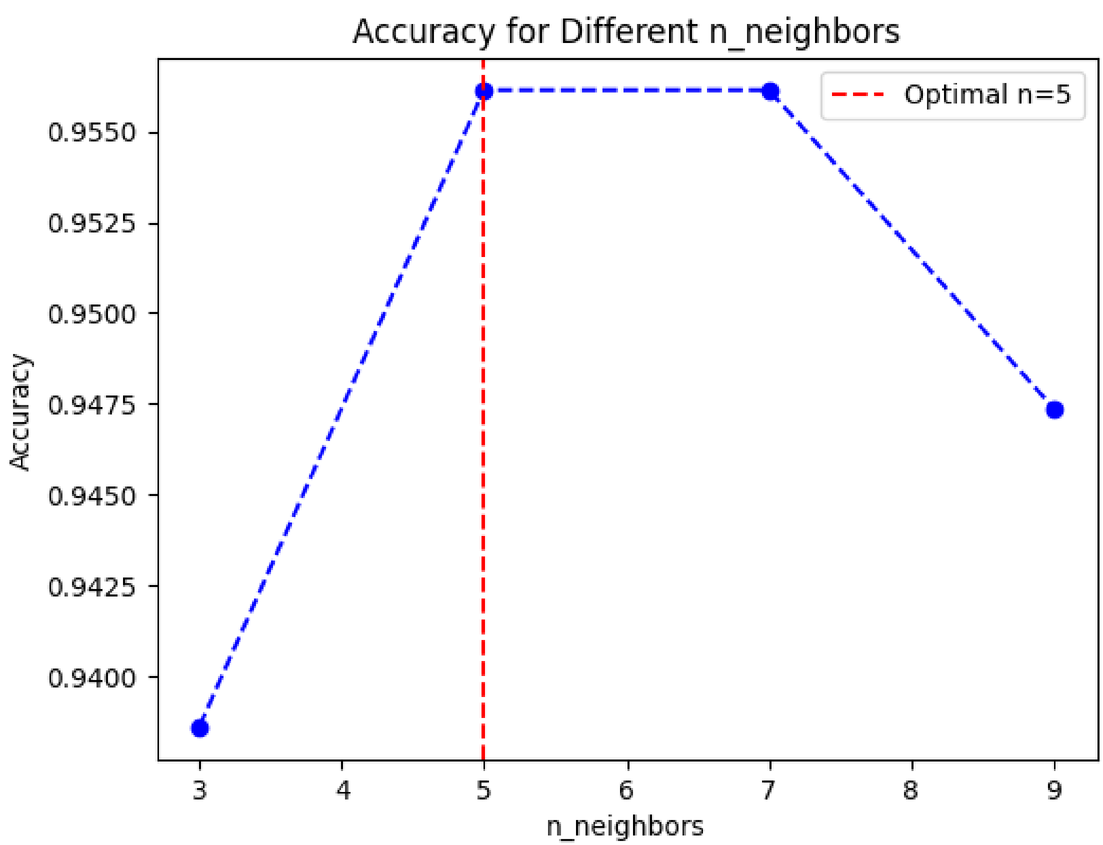
[ ]:
```python
#@title ## **1. Train the KNN Classifier:**

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# Initialize and train the KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predict the test set
y_pred = knn.predict(X_test)
```

```
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

# Classification report
print('Classification Report:')
print(classification_report(y_test, y_pred, target_names=['Benign',␣
↪'Malignant']))
```

```
Accuracy: 95.61%
Confusion Matrix:
[[71  1]
 [ 4 38]]
Classification Report:
              precision    recall  f1-score   support

      Benign       0.95      0.99      0.97        72
   Malignant       0.97      0.90      0.94        42

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

[ ]:
```
#@title ## **2. Experiment with Different n_neighbors:**

import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

# Define a list of different neighbor values to experiment with
neighbors = [3, 5, 7, 9]
accuracies = []

# Iterate over different n_neighbors values
for n in neighbors:
    knn = KNeighborsClassifier(n_neighbors=n)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)

    # Calculate accuracy for each model
    accuracy = accuracy_score(y_test, y_pred)
```

```
    accuracies.append(accuracy)

# Plot accuracy vs n_neighbors
plt.plot(neighbors, accuracies, marker='o', color='blue', linestyle='--')
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.title('Accuracy for Different n_neighbors')

# Highlight the optimal n_neighbors
optimal_n = neighbors[accuracies.index(max(accuracies))]
optimal_acc = max(accuracies)

plt.axvline(x=optimal_n, color='red', linestyle='--', label=f'Optimal␣
↪n={optimal_n}')
plt.legend()
plt.show()

# Print the best n_neighbors
print(f'The optimal n_neighbors is {optimal_n} with an accuracy of {optimal_acc␣
↪* 100:.2f}%.')
```



```
The optimal n_neighbors is 5 with an accuracy of 95.61%.
```

Exercise 3: Implementing Logistic Regression

[ ]:
```python
#@title ## **1. Train Logistic Regression:**

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score,␣
↪confusion_matrix

# Logistic Regression
logreg = LogisticRegression(max_iter=10000)
logreg.fit(X_train, y_train)

# Predict the test set
y_pred_lr = logreg.predict(X_test)

# Accuracy and classification report
accuracy_lr = accuracy_score(y_test, y_pred_lr)
print(f'Logistic Regression Accuracy: {accuracy_lr * 100:.2f}%')

# Confusion matrix
conf_matrix_lr = confusion_matrix(y_test, y_pred_lr)
print('Confusion Matrix (Logistic Regression):')
print(conf_matrix_lr)

# Classification report
print('Classification Report (Logistic Regression):')
print(classification_report(y_test, y_pred_lr, target_names=['Benign',␣
↪'Malignant']))
```

```
Logistic Regression Accuracy: 97.37%
Confusion Matrix (Logistic Regression):
[[71  1]
 [ 2 40]]
Classification Report (Logistic Regression):
              precision    recall  f1-score   support

      Benign       0.97      0.99      0.98        72
   Malignant       0.98      0.95      0.96        42

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

[ ]:

```python
#@title ## **2. Comparison of KNN and Logistic Regression:**

import pandas as pd

# Accuracy for KNN and Logistic Regression
accuracy_knn = accuracy_score(y_test, y_pred)
accuracy_lr = accuracy_score(y_test, y_pred_lr)

# Precision, Recall, F1-Score for both models
report_knn = classification_report(y_test, y_pred, target_names=['Benign',␣
↪'Malignant'], output_dict=True)
report_lr = classification_report(y_test, y_pred_lr, target_names=['Benign',␣
↪'Malignant'], output_dict=True)

# Create a comparison DataFrame
comparison_df = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression'],
    'Accuracy': [accuracy_knn * 100, accuracy_lr * 100],
    'Precision (Benign)': [report_knn['Benign']['precision'],␣
↪report_lr['Benign']['precision']],
    'Recall (Benign)': [report_knn['Benign']['recall'],␣
↪report_lr['Benign']['recall']],
    'F1-Score (Benign)': [report_knn['Benign']['f1-score'],␣
↪report_lr['Benign']['f1-score']],
    'Precision (Malignant)': [report_knn['Malignant']['precision'],␣
↪report_lr['Malignant']['precision']],
    'Recall (Malignant)': [report_knn['Malignant']['recall'],␣
↪report_lr['Malignant']['recall']],
    'F1-Score (Malignant)': [report_knn['Malignant']['f1-score'],␣
↪report_lr['Malignant']['f1-score']]
})

# Display the comparison
print(comparison_df)

# Determine which model performs better
if accuracy_knn > accuracy_lr:
    print("KNN performs better in terms of accuracy.")
elif accuracy_knn < accuracy_lr:
    print("Logistic Regression performs better in terms of accuracy.")
else:
    print("Both models have the same accuracy.")
```

```
                 Model   Accuracy  Precision (Benign)  Recall (Benign)  \
0                  KNN  94.736842            0.934211         0.986111
1  Logistic Regression  97.368421            0.972603         0.986111
```

```
   F1-Score (Benign)  Precision (Malignant)  Recall (Malignant)  \
0           0.959459               0.973684            0.880952
1           0.979310               0.975610            0.952381

   F1-Score (Malignant)
0              0.925000
1              0.963855
Logistic Regression performs better in terms of accuracy.
```

Exercise 4: Hyperparameter Tuning and Cross-Validation

[ ]:
```
#@title ## **1. GridSearchCV for KNN:**

from sklearn.model_selection import GridSearchCV

# Defining the parameter grid for KNN
param_grid = {'n_neighbors': [3, 5, 7, 9], 'weights': ['uniform', 'distance'],␣
↪'p': [1, 2]}

# Perform Grid Search with 5-fold cross-validation
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Output the best parameters and corresponding accuracy
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print(f'Best Parameters: {best_params}')
print(f'Best Cross-Validation Accuracy: {best_score * 100:.2f}%')
```

```
Best Parameters: {'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}
Best Cross-Validation Accuracy: 96.92%
```

[ ]:
```
#@title ## **2. Cross-Validation for Logistic Regression:**

from sklearn.model_selection import cross_val_score

# Perform 5-fold cross-validation for Logistic Regression
cv_scores = cross_val_score(logreg, scaled_features, df['diagnosis'], cv=5)

# Output the mean cross-validated accuracy
mean_cv_accuracy = cv_scores.mean()

print(f'Cross-Validated Accuracy (Logistic Regression): {mean_cv_accuracy * 100:.
↪2f}%')
```

```
Cross-Validated Accuracy (Logistic Regression): 98.07%
```

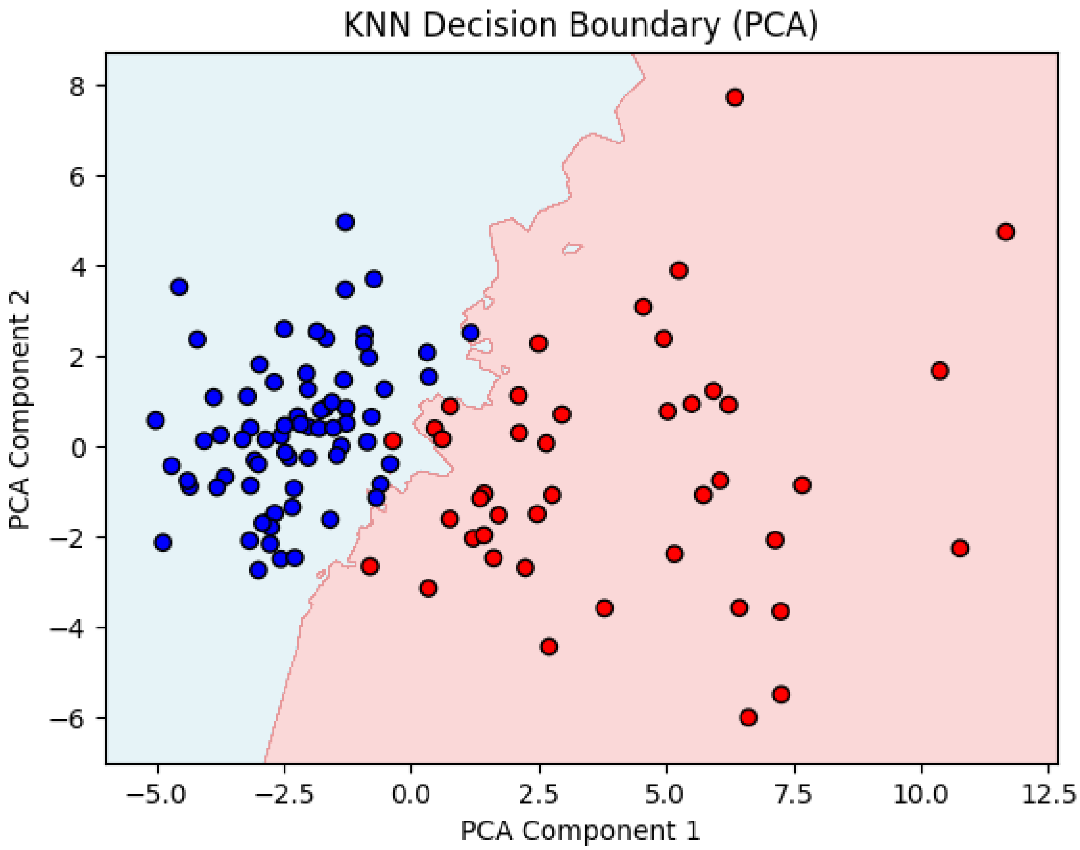
Exercise 5: Decision Boundary Visualization

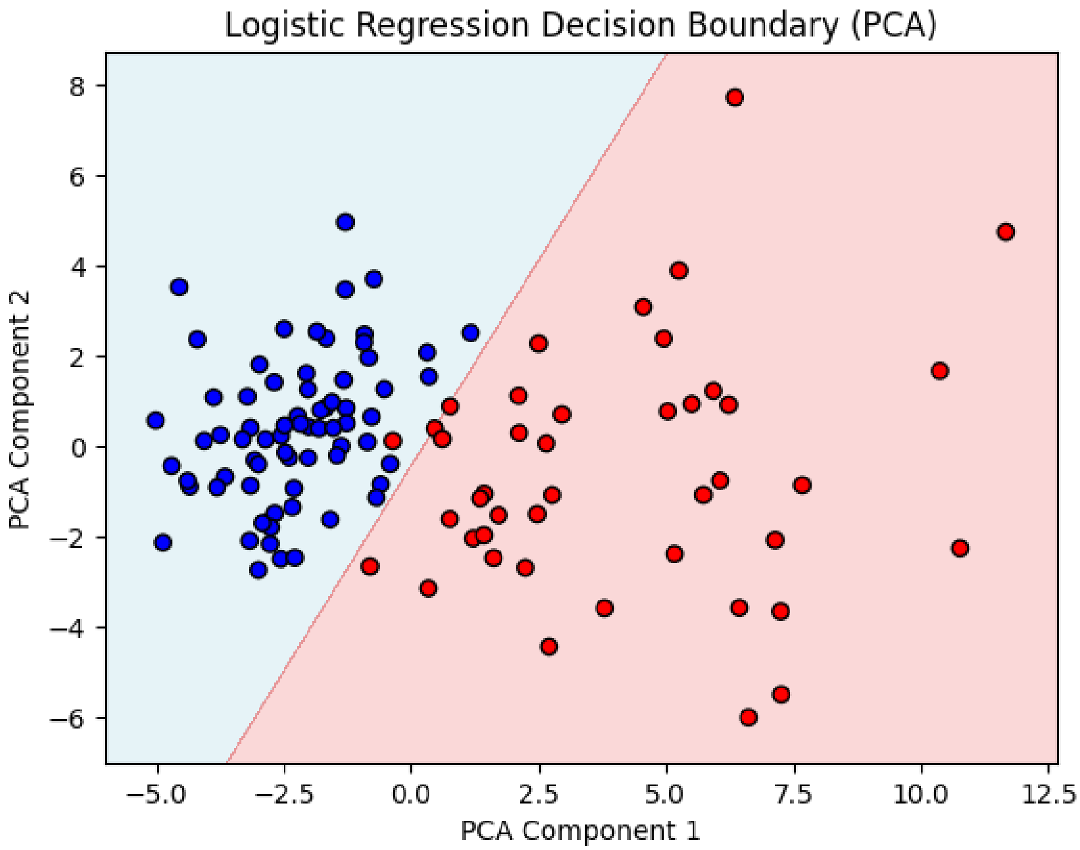
[ ]:
```python
#@title ## **1. Use PCA for Dimensionality Reduction:**

from sklearn.decomposition import PCA

# Apply PCA to reduce dimensionality to 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(scaled_features)

# Train-test split after PCA
X_pca_train, X_pca_test, y_train_pca, y_test_pca = train_test_split(X_pca,
 ↪df['diagnosis'], test_size=0.2, random_state=42)

# KNN with PCA data
knn_pca = KNeighborsClassifier(n_neighbors=5)
knn_pca.fit(X_pca_train, y_train_pca)

# Logistic Regression with PCA data
logreg_pca = LogisticRegression(max_iter=10000)
logreg_pca.fit(X_pca_train, y_train_pca)
```

[ ]: LogisticRegression(max_iter=10000)

[ ]:
```python
#@title ## **Task: Plot the Decision Boundary:**

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def plot_decision_boundary(model, X, y, title):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    # Predict on the mesh grid
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot decision boundary
    plt.contourf(xx, yy, Z, alpha=0.3, cmap=ListedColormap(('lightblue',
 ↪'lightcoral')))
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k',
 ↪cmap=ListedColormap(('blue', 'red')))
    plt.title(title)
    plt.xlabel('PCA Component 1')
```

```
    plt.ylabel('PCA Component 2')
    plt.show()

# Plot decision boundaries for KNN and Logistic Regression
plot_decision_boundary(knn_pca, X_pca_test, y_test_pca, title='KNN Decision␣
 ↪Boundary (PCA)')
plot_decision_boundary(logreg_pca, X_pca_test, y_test_pca, title='Logistic␣
 ↪Regression Decision Boundary (PCA)')
```





Logistic Regression Decision Boundary (PCA)
PCA Component 2
PCA Component 1
8
6
4
2
0
−2
−4
−6
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5