K-Nearest Neighbors (KNN) and Logistic Regression

Exercise 1: Data Exploration and Preprocessing

1. Load and Explore the Data:

```
import pandas as pd

# Load the dataset
df = pd.read_csv('Breast Cancer Diagnosis Dataset with Tumor Characteristics.csv')

# Display the first 10 rows
print(df.head(10))

# Check for missing values
print(df.isnull().sum())

# Descriptive statistics
print(df.describe())
```

```
50%            NaN
75%            NaN
max            NaN

[8 rows x 32 columns]
```

Task: Summarize the Dataset:

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

# Missing values
print(df.isnull().sum())
```

```
Instances: 569, Features: 33
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
```

3. Preprocessing:

```
from sklearn.preprocessing import StandardScaler

# Drop irrelevant columns
df = df.drop(columns=['id', 'Unnamed: 32'], errors='ignore')

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

# Normalize features
scaler = StandardScaler()
features = df.drop(columns=['diagnosis'])
scaled_features = scaler.fit_transform(features)
```

4. Train-Test Split:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(scaled_features, df['diagnosis'], test_size=0.2, random_state=42)
```

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

1. Train the KNN Classifier:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

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}')

# Confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9473684210526315
[[68  3]
 [ 3 40]]
```

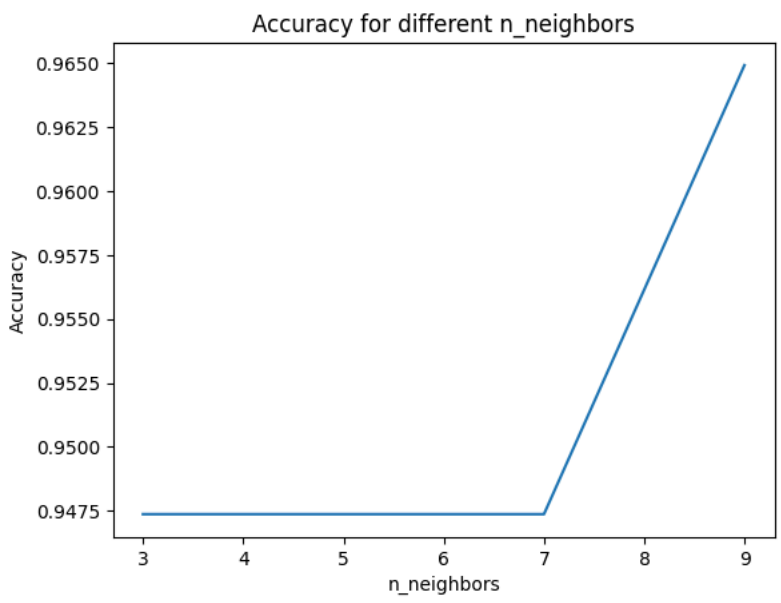
2. Experiment with Different n_neighbors:

```
import matplotlib.pyplot as plt

neighbors = [3, 5, 7, 9]
accuracies = []

for n in neighbors:
    knn = KNeighborsClassifier(n_neighbors=n)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracies.append(accuracy_score(y_test, y_pred))

# Plot
plt.plot(neighbors, accuracies)
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.title('Accuracy for different n_neighbors')
plt.show()
```



Exercise 3: Implementing Logistic Regression

1. Train Logistic Regression:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

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

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

# Accuracy and classification report
print(f'Accuracy: {accuracy_score(y_test, y_pred_lr)}')
print(confusion_matrix(y_test, y_pred_lr))
print(classification_report(y_test, y_pred_lr))
```

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

           0       0.97      0.99      0.98        71
           1       0.98      0.95      0.96        43

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

2. Comparison of KNN and Logistic Regression:

Compare their accuracy, precision, and F1-score based on the classification report.

Exercise 4: Hyperparameter Tuning and Cross-Validation

1. GridSearchCV for KNN:

```
from sklearn.model_selection import GridSearchCV

param_grid = {'n_neighbors': [3, 5, 7, 9], 'weights': ['uniform', 'distance'], 'p': [1, 2]}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Best parameters and accuracy
print(grid_search.best_params_)
print(grid_search.best_score_)
```

```
{'n_neighbors': 5, 'p': 1, 'weights': 'uniform'}
0.9648351648351647
```

2. Cross-Validation for Logistic Regression:

```
from sklearn.model_selection import cross_val_score

# k-fold cross-validation
cv_scores = cross_val_score(logreg, scaled_features, df['diagnosis'], cv=5)
print(f'Cross-validated accuracy: {cv_scores.mean()}')
```

```
Cross-validated accuracy: 0.9806862288464524
```

Exercise 5: Decision Boundary Visualization

1. Use PCA for Dimensionality Reduction:

```
from sklearn.decomposition import PCA

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

# KNN and Logistic regression with PCA data
knn_pca = KNeighborsClassifier(n_neighbors=5)
knn_pca.fit(X_pca, df['diagnosis'])

logreg_pca = LogisticRegression(max_iter=10000)
logreg_pca.fit(X_pca, df['diagnosis'])
```

```
LogisticRegression
LogisticRegression(max_iter=10000)
```

Task: Plot the Decision Boundary:**

You can use matplotlib or similar libraries to plot the decision boundaries.