Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.decomposition import PCA
```

Exercise 1: Data Exploration and Preprocessing

```
# 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())
```

```
         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
```

```
# 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
```

```
# 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)
```

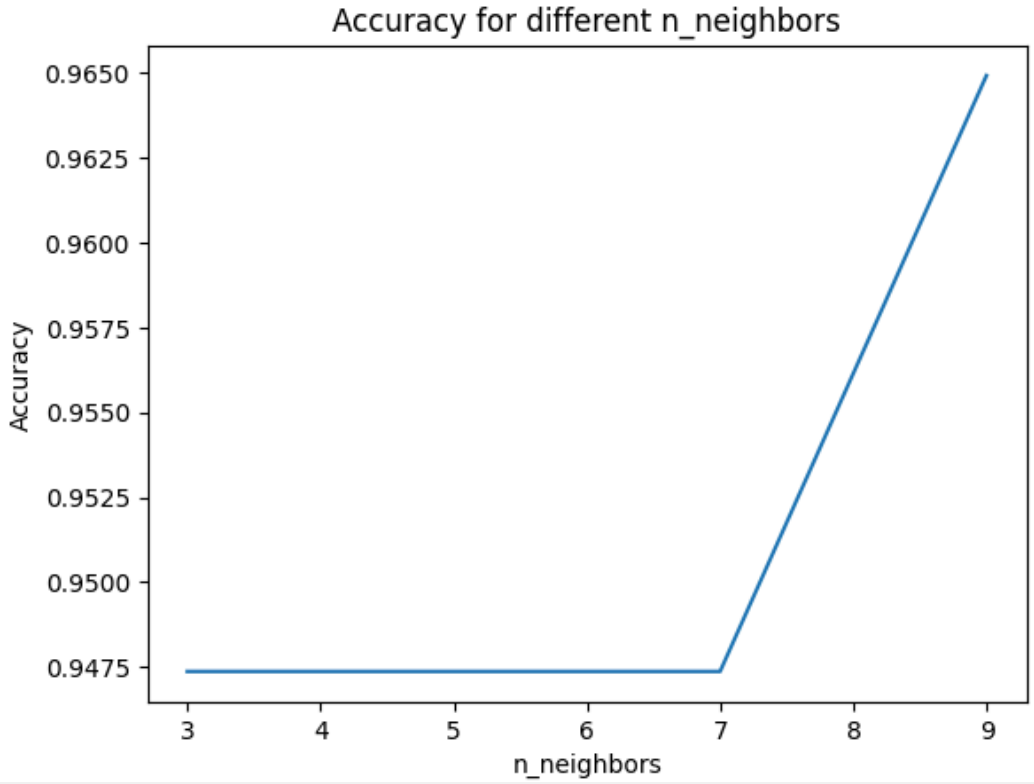
```
X_train, X_test, y_train, y_test = train_test_split(scaled_features, df['diagnosis'], test_size=0.2, ran
```

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

```
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]]
```

```
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

```
# 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
```

Exercise 4: Hyperparameter Tuning and Cross-Validation

```
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
```

```
# 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

```
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)
```

```
# Create a meshgrid for the PCA features
x_min, x_max = X_pca[:, 0].min() - 1, X_pca[:, 0].max() + 1
y_min, y_max = X_pca[:, 1].min() - 1, X_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                     np.arange(y_min, y_max, 0.01))

# Predict the class using the KNN model
Z_knn = knn_pca.predict(np.c_[xx.ravel(), yy.ravel()])
Z_knn = Z_knn.reshape(xx.shape)

# Predict the class using the Logistic Regression model
Z_logreg = logreg_pca.predict(np.c_[xx.ravel(), yy.ravel()])
Z_logreg = Z_logreg.reshape(xx.shape)

# Create a simple plot
plt.figure(figsize=(12, 5))

# KNN Decision Boundary
plt.subplot(1, 2, 1)
plt.contourf(xx, yy, Z_knn, alpha=0.4)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['diagnosis'], edgecolor='k')
plt.title('KNN Decision Boundary')

# Logistic Regression Decision Boundary
```

```
plt.subplot(1, 2, 2)
plt.contourf(xx, yy, Z_logreg, alpha=0.4)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['diagnosis'], edgecolor='k')
plt.title('Logistic Regression Decision Boundary')

plt.show()
```