1. Data Exploration and Preprocessing

```
# Step 1: Install necessary libraries (only if required)
!pip install scikit-learn pandas matplotlib seaborn

# Step 2: Import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
import seaborn as sns

# Step 3: Load the dataset (assuming it's a CSV file)
df = pd.read_csv('/content/Breast Cancer Diagnosis Dataset with Tumor Characteristics.csv')

# Step 4: Exploratory Data Analysis (EDA)
print("First five rows of the dataset:")
print(df.head())

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

# Step 5: Convert categorical target (diagnosis) to numeric (Malignant = 1, Benign = 0)
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})

# Step 6: Define features (X) and target (y)
X = df.drop('diagnosis', axis=1)  # Features
y = df['diagnosis']  # Target

# Step 7: Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 8: Handle missing values with SimpleImputer (impute using mean)
imputer = SimpleImputer(strategy='mean')

# Apply imputation to training and testing sets
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Step 9: Normalize features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print("Data preprocessing complete.")
```

```
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

[5 rows x 33 columns]
```

```
    warnings.warn(
```

2. Model Development

```
# Step 10: Import required libraries for model development
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

# Step 11: Develop K-Nearest Neighbors (KNN) model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Step 12: Develop Logistic Regression model for comparison
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

print("Model development complete.")
```

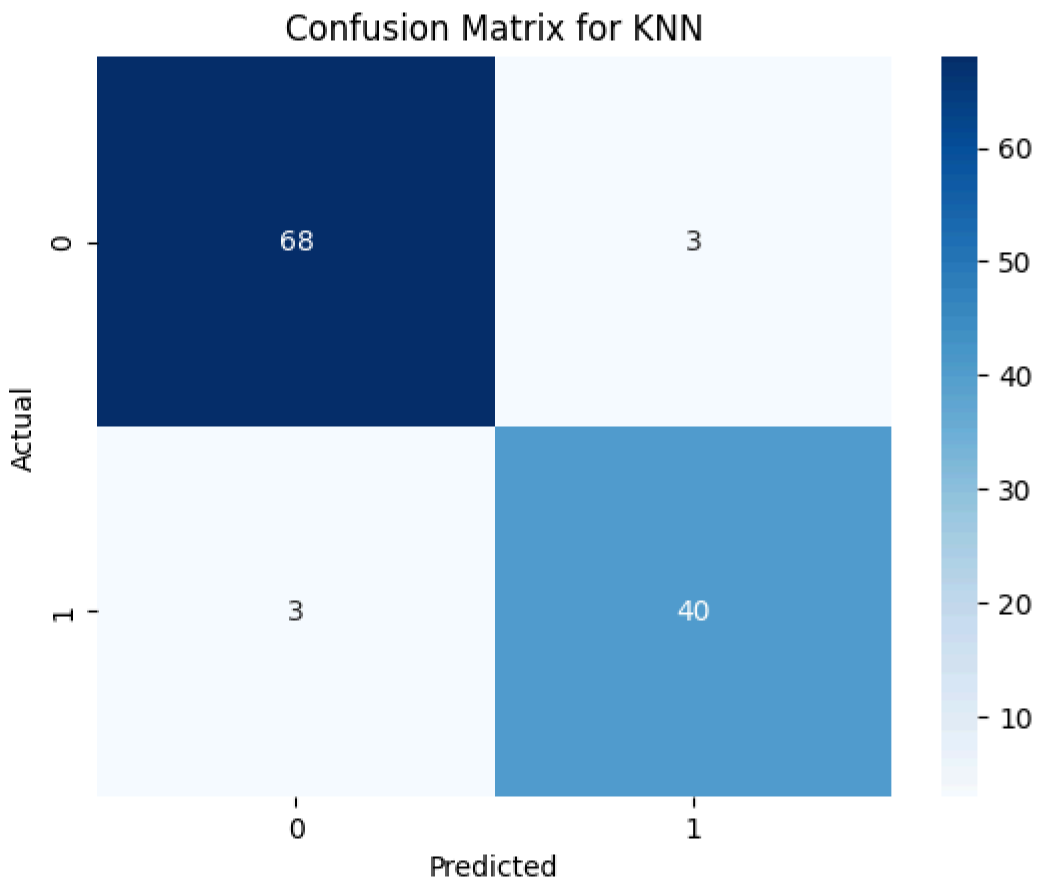
```
Model development complete.
```

3. Model Evaluation

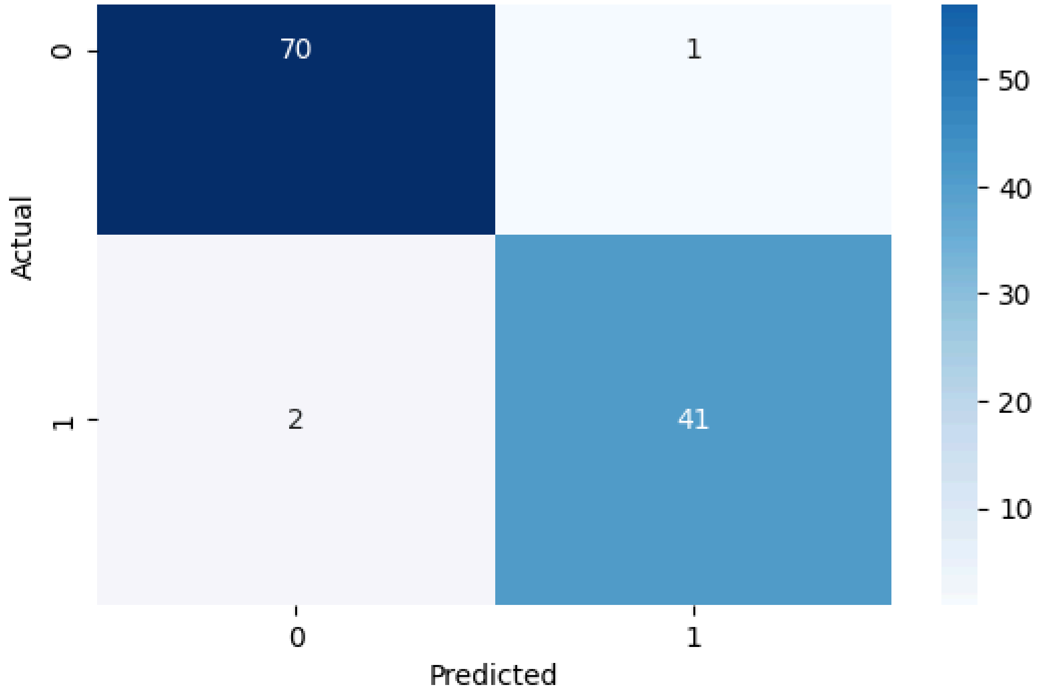
```
# Step 13: Import evaluation metrics
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Step 14: Evaluate KNN model
y_pred_knn = knn.predict(X_test)
print("KNN Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Classification Report:\n", classification_report(y_test, y_pred_knn))

# Step 15: Confusion matrix for KNN
conf_matrix_knn = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(conf_matrix_knn, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix for KNN')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Step 16: Evaluate Logistic Regression model
y_pred_log_reg = log_reg.predict(X_test)
print("Logistic Regression Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_log_reg))
print("Classification Report:\n", classification_report(y_test, y_pred_log_reg))

# Step 17: Confusion matrix for Logistic Regression
conf_matrix_log_reg = confusion_matrix(y_test, y_pred_log_reg)
```

```
sns.heatmap(conf_matrix_log_reg, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix for Logistic Regression')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print("Model evaluation complete.")
```

```
KNN Model Performance:
Accuracy: 0.9473684210526315
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        71
           1       0.93      0.93      0.93        43

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114
```



```
Logistic Regression Model Performance:
Accuracy: 0.9736842105263158
Classification Report:
              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
```



```
Model evaluation complete.
```

Logistic Regression is better than KNN for this task. It has an accuracy of 97.4%, while KNN is at 94.7%.

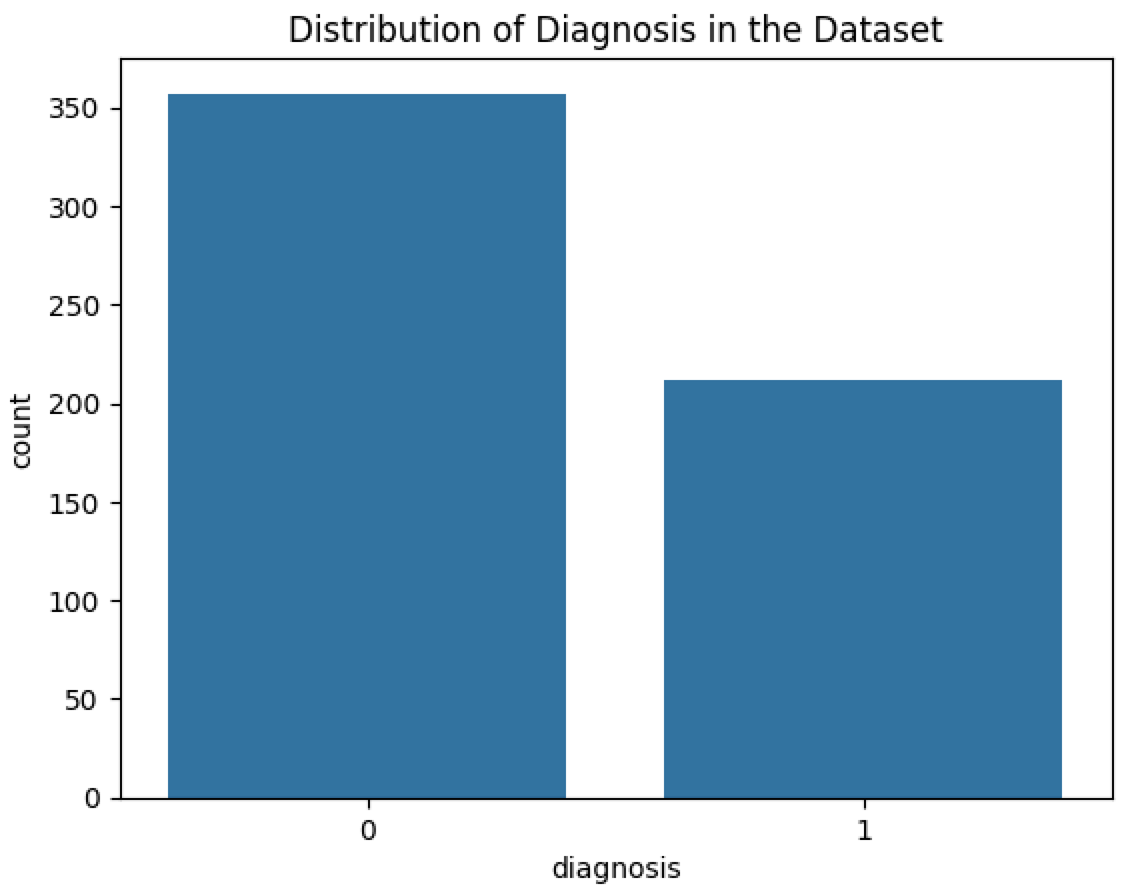
Logistic Regression is also more accurate in predicting malignant tumors, with 98% precision compared to KNN's 93%. Overall, Logistic Regression is the better choice.

4. Report and Visualizations

```
# Step 18: Compare models and report findings
print("Comparison of KNN and Logistic Regression Models:")

# KNN Results
knn_accuracy = accuracy_score(y_test, y_pred_knn)
print(f"KNN Accuracy: {knn_accuracy}")
print("KNN Classification Report:\n", classification_report(y_test, y_pred_knn))

# Logistic Regression Results
log_reg_accuracy = accuracy_score(y_test, y_pred_log_reg)
print(f"Logistic Regression Accuracy: {log_reg_accuracy}")
print("Logistic Regression Classification Report:\n", classification_report(y_test, y_pred_l

# Step 19: Optional Visualizations (decision boundaries if applicable, or data distribution)
# Example: Distribution of diagnosis values in the dataset
sns.countplot(x='diagnosis', data=df)
plt.title('Distribution of Diagnosis in the Dataset')
plt.show()
```

```
print("Report and visualizations complete.")
```

```
Comparison of KNN and Logistic Regression Models:
KNN Accuracy: 0.9473684210526315
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        71
           1       0.93      0.93      0.93        43

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114

Logistic Regression Accuracy: 0.9736842105263158
Logistic Regression Classification Report:
              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
```



```
Report and visualizations complete.
```

1. Data Preprocessing

- Loaded Dataset: Analyzed the data structure and checked for missing values.
- Handled Missing Values: Used mean imputation to replace missing values.
- Converted Target: Changed diagnosis labels ('M' for malignant and 'B' for benign) to 1 and 0.
- Split Data: Divided the dataset into training (80%) and testing (20%) sets.
- Normalized Features: Scaled the features for better model performance.

2. Model Development

- KNN: Used K-Nearest Neighbors with 3 neighbors.