ProjectDataAnalysis

March 9, 2025

1 Reading Data

Thyroid cancer data is retrived from Kaggle website: https://www.kaggle.com/datasets/ankushpanday1/thyroid-cancer-risk-prediction-dataset/data

[ ]:
```
#import necessary libraries
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import prince
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
from sklearn.model_selection import train_test_split,RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from scipy.stats import randint
from sklearn.decomposition import PCA
```

[42]:
```
#Mount google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[4]:
```
#read data from drive
thyroid_data = pd.read_csv('/content/drive/My Drive/COURSES/Winter_2025/CIND820␣
 ↪XJH - Big Data Analytics Project - W202/thyroid_cancer_risk_data.csv')
print(thyroid_data.head())
```

```
   Patient_ID  Age  Gender  Country  Ethnicity Family_History  \
0           1   66    Male   Russia  Caucasian             No
1           2   29    Male  Germany   Hispanic             No
```

```
2           3   86    Male  Nigeria  Caucasian             No
3           4   75  Female    India      Asian             No
4           5   35  Female  Germany    African            Yes

  Radiation_Exposure Iodine_Deficiency Smoking Obesity Diabetes  TSH_Level  \
0                Yes                No      No      No       No       9.37
1                Yes                No      No      No       No       1.83
2                 No                No      No      No       No       6.26
3                 No                No      No      No       No       4.10
4                Yes                No      No      No       No       9.10

   T3_Level  T4_Level  Nodule_Size Thyroid_Cancer_Risk Diagnosis
0      1.67      6.16         1.08                 Low    Benign
1      1.73     10.54         4.05                 Low    Benign
2      2.59     10.57         4.61                 Low    Benign
3      2.62     11.04         2.46              Medium    Benign
4      2.11     10.71         2.11                High    Benign
```

2 Preprocessing

Data is explored to know attributes name, shape of the data frame, data types, missing values and regional location of thyroid cancer patients.

3 Variable attributes, Data Types and Missing Values

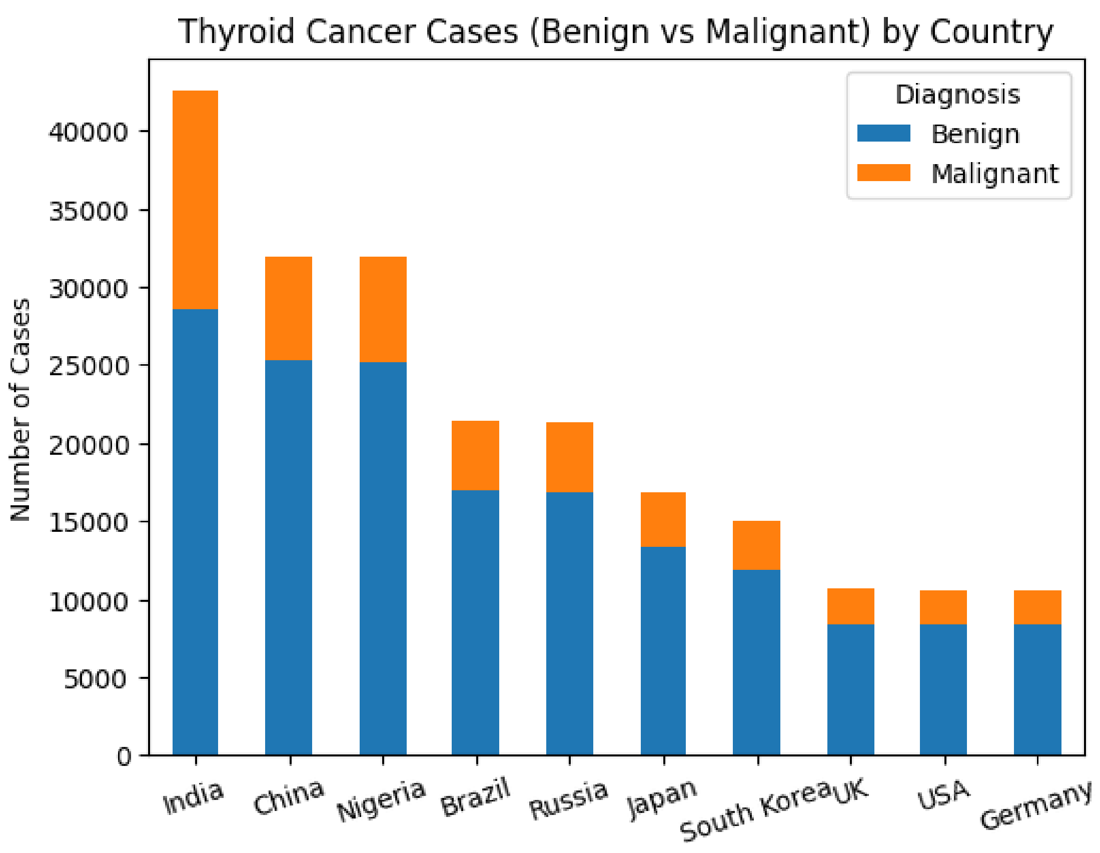
[6]:
```python
# Display the column names of the DataFrame
print("Column names in the DataFrame:")
print(thyroid_data.columns)

#Display the shape of the DataFrame
print("Shape of the DataFrame:")
print(thyroid_data.shape)

#Display the data types of the columns
print("Data types of the columns:")
print(thyroid_data.dtypes)

#Display the missing values in the DataFrame
missing_values = thyroid_data.isnull().sum()
print("Missing values in the DataFrame:", missing_values)

#Different countries in the dataset
countries = thyroid_data['Country'].unique()
print("Countries in the dataset:", countries)
```

```
Column names in the DataFrame:
Index(['Patient_ID', 'Age', 'Gender', 'Country', 'Ethnicity', 'Family_History',
       'Radiation_Exposure', 'Iodine_Deficiency', 'Smoking', 'Obesity',
       'Diabetes', 'TSH_Level', 'T3_Level', 'T4_Level', 'Nodule_Size',
       'Thyroid_Cancer_Risk', 'Diagnosis'],
      dtype='object')
Shape of the DataFrame:
(212691, 17)
Data types of the columns:
Patient_ID               int64
Age                      int64
Gender                  object
Country                 object
Ethnicity               object
Family_History          object
Radiation_Exposure      object
Iodine_Deficiency       object
Smoking                 object
Obesity                 object
Diabetes                object
TSH_Level              float64
T3_Level               float64
T4_Level               float64
Nodule_Size            float64
Thyroid_Cancer_Risk     object
Diagnosis               object
dtype: object
Missing values in the DataFrame: Patient_ID            0
Age                    0
Gender                 0
Country                0
Ethnicity              0
Family_History         0
Radiation_Exposure     0
Iodine_Deficiency      0
Smoking                0
Obesity                0
Diabetes               0
TSH_Level              0
T3_Level               0
T4_Level               0
Nodule_Size            0
Thyroid_Cancer_Risk    0
Diagnosis              0
dtype: int64
Countries in the dataset: ['Russia' 'Germany' 'Nigeria' 'India' 'UK' 'South
Korea' 'Brazil' 'China'
 'Japan' 'USA']
```

4 Thyroid Cancer Distribution by Region, Gender and Ethnicity

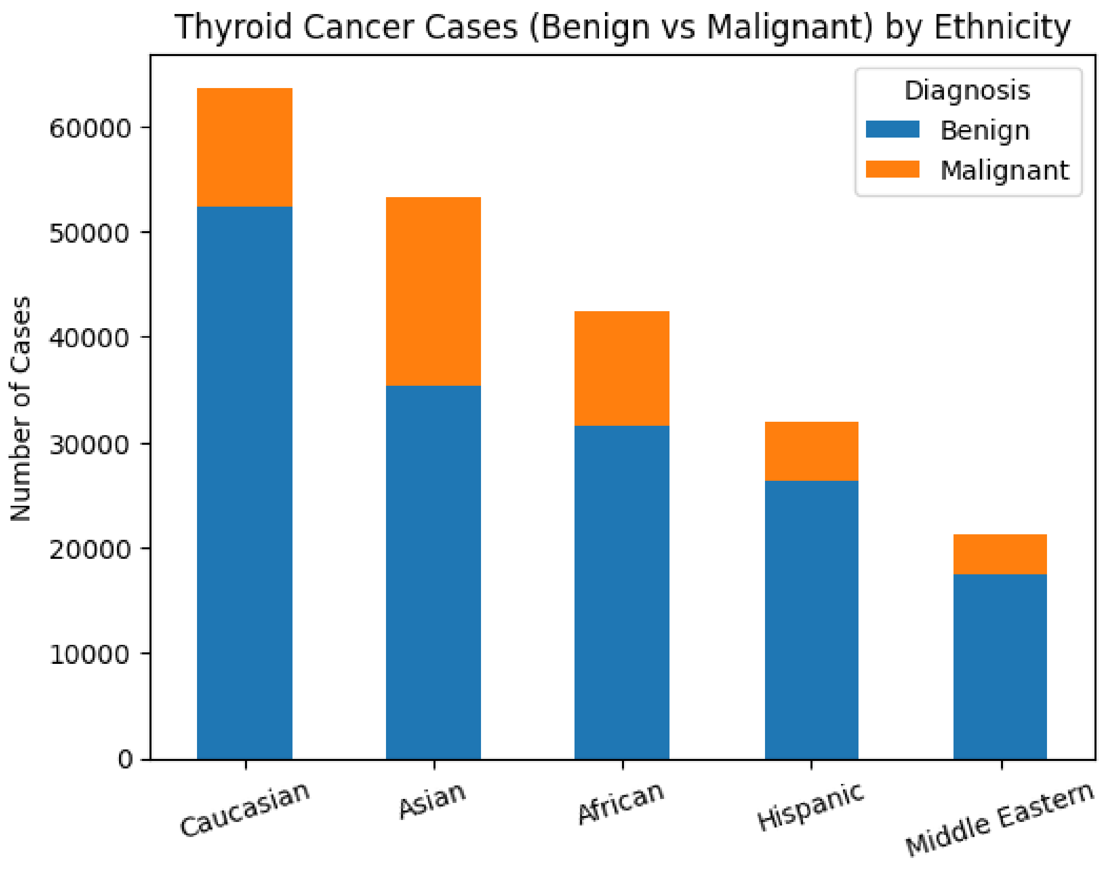
[41]:
```python
def myPlot(data, var):
    plt.figure(figsize=(6, 5))
    sorted_data = thyroid_data.groupby([var, 'Diagnosis']).size().unstack().
  ↪fillna(0)
    sorted_data['Total'] = sorted_data.sum(axis=1)
    sorted_data = sorted_data.sort_values(by='Total', ascending=False).
  ↪drop(columns='Total')
    sorted_data.plot(kind='bar', stacked=True)
    plt.title('Thyroid Cancer Cases (Benign vs Malignant) by '+var)
    plt.xlabel(' ')
    plt.ylabel('Number of Cases')
    plt.xticks(rotation=18)
    plt.legend(title='Diagnosis')
    plt.show()

myPlot(thyroid_data, 'Country')
myPlot(thyroid_data, 'Ethnicity')
myPlot(thyroid_data, 'Gender')
```

```
<Figure size 600x500 with 0 Axes>
```



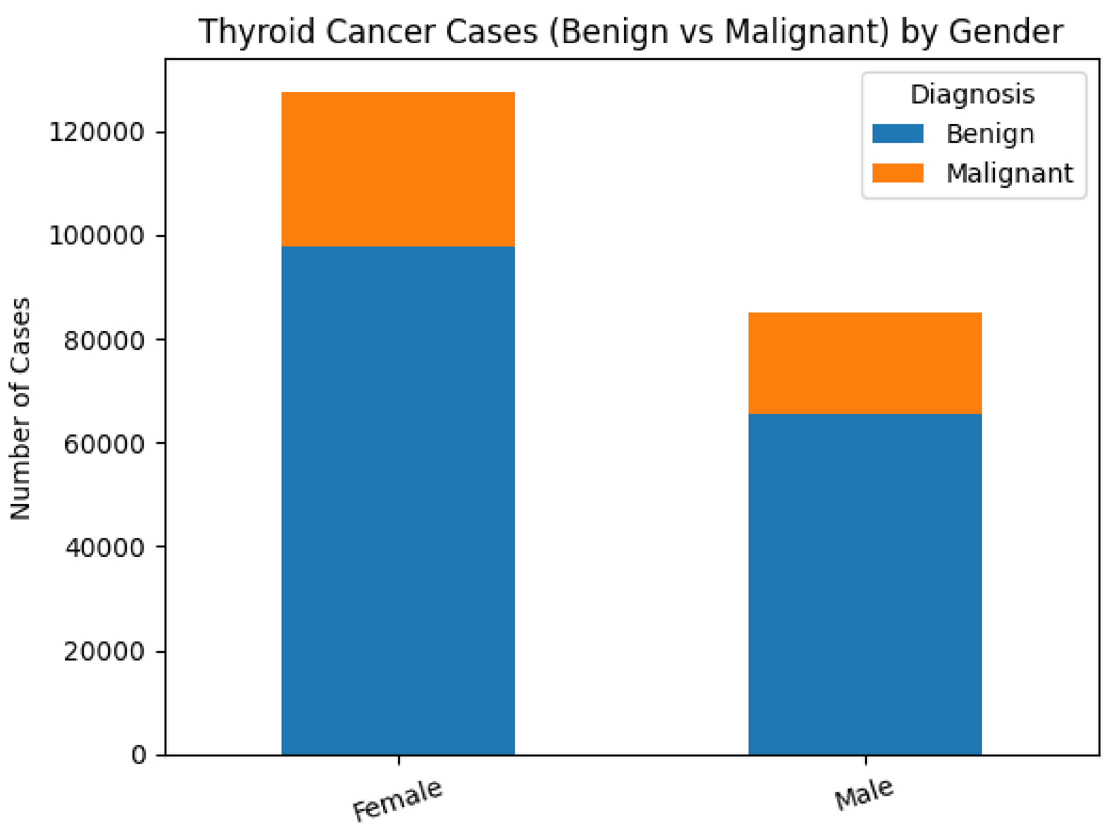
Thyroid Cancer Cases (Benign vs Malignant) by Country
Diagnosis
Benign
Malignant
Number of Cases
0
5000
10000
15000
20000
25000
30000
35000
40000
India
China
Nigeria
Brazil
Russia
Japan
South Korea
UK
USA
Germany

```
<Figure size 600x500 with 0 Axes>
```



Thyroid Cancer Cases (Benign vs Malignant) by Ethnicity
Number of Cases
0
10000
20000
30000
40000
50000
60000
Diagnosis
Benign
Malignant
Caucasian
Asian
African
Hispanic
Middle Eastern

```
<Figure size 600x500 with 0 Axes>
```



5 Summary Statistics

Proportion by categorical variables rounded to 2 decimal places

[10]:
```
print("Proportion by categorical variables (rounded to 2 decimal places):")
categorical_columns = thyroid_data.select_dtypes(include=['object']).columns
for column in categorical_columns:
    print(f"\n{column}:\n{thyroid_data[column].value_counts(normalize=True).
  ↪round(2)*100}")
```

```
Proportion by categorical variables (rounded to 2 decimal places):

Gender:
Gender
Female    60.0
Male      40.0
Name: proportion, dtype: float64

Country:
```

```
Country
India          20.0
China          15.0
Nigeria        15.0
Brazil         10.0
Russia         10.0
Japan           8.0
South Korea     7.0
UK              5.0
USA             5.0
Germany         5.0
Name: proportion, dtype: float64

Ethnicity:
Ethnicity
Caucasian         30.0
Asian             25.0
African           20.0
Hispanic          15.0
Middle Eastern    10.0
Name: proportion, dtype: float64

Family_History:
Family_History
No     70.0
Yes    30.0
Name: proportion, dtype: float64

Radiation_Exposure:
Radiation_Exposure
No     85.0
Yes    15.0
Name: proportion, dtype: float64

Iodine_Deficiency:
Iodine_Deficiency
No     75.0
Yes    25.0
Name: proportion, dtype: float64

Smoking:
Smoking
No     80.0
Yes    20.0
Name: proportion, dtype: float64

Obesity:
Obesity
```

```
No     70.0
Yes    30.0
Name: proportion, dtype: float64

Diabetes:
Diabetes
No     80.0
Yes    20.0
Name: proportion, dtype: float64

Thyroid_Cancer_Risk:
Thyroid_Cancer_Risk
Low       51.0
Medium    34.0
High      15.0
Name: proportion, dtype: float64

Diagnosis:
Diagnosis
Benign       77.0
Malignant    23.0
Name: proportion, dtype: float64
```

Summary statistics of all continuous variables rounded to 2 decimal places

[6]:
```
continuous_columns = thyroid_data.select_dtypes(include=['number']).columns
continuous_columns_except_id = continuous_columns.drop('Patient_ID',␣
↪errors='ignore')
summary_stats = thyroid_data[continuous_columns_except_id].describe().round(2)
print(summary_stats)
```

```
             Age  TSH_Level   T3_Level   T4_Level  Nodule_Size
count  212691.00  212691.00  212691.00  212691.00    212691.00
mean       51.92       5.05       2.00       8.25         2.50
std        21.63       2.86       0.87       2.16         1.44
min        15.00       0.10       0.50       4.50         0.00
25%        33.00       2.57       1.25       6.37         1.25
50%        52.00       5.04       2.00       8.24         2.51
75%        71.00       7.52       2.75      10.12         3.76
max        89.00      10.00       3.50      12.00         5.00
```

Scaling continuous variables

[7]:
```
# Initialize the scaler
scaler = MinMaxScaler()

# Fit and transform the continuous variables
scaled_df = scaler.fit_transform(thyroid_data[continuous_columns_except_id])
```

```
# Create a new DataFrame with the scaled data
scaled_data = pd.DataFrame(scaled_df, columns=continuous_columns_except_id)

# Print the scaled data
print(scaled_data.head())
```

```
        Age  TSH_Level  T3_Level  T4_Level  Nodule_Size
0  0.689189   0.936364  0.390000  0.221333        0.216
1  0.189189   0.174747  0.410000  0.805333        0.810
2  0.959459   0.622222  0.696667  0.809333        0.922
3  0.810811   0.404040  0.706667  0.872000        0.492
4  0.270270   0.909091  0.536667  0.828000        0.422
```

Merge the scaled data back into the original Thyroid data frame.

[8]:
```
thyroid_data = pd.concat([thyroid_data.
 ↪drop(columns=continuous_columns_except_id), scaled_data], axis=1)

# Print the head of the merged DataFrame
print(thyroid_data.head())
```

```
   Patient_ID  Gender  Country  Ethnicity Family_History Radiation_Exposure  \
0           1    Male   Russia  Caucasian             No                Yes
1           2    Male  Germany   Hispanic             No                Yes
2           3    Male  Nigeria  Caucasian             No                 No
3           4  Female    India      Asian             No                 No
4           5  Female  Germany    African            Yes                Yes

  Iodine_Deficiency Smoking Obesity Diabetes Thyroid_Cancer_Risk Diagnosis  \
0                No      No      No       No                 Low    Benign
1                No      No      No       No                 Low    Benign
2                No      No      No       No                 Low    Benign
3                No      No      No       No              Medium    Benign
4                No      No      No       No                High    Benign

        Age  TSH_Level  T3_Level  T4_Level  Nodule_Size
0  0.689189   0.936364  0.390000  0.221333        0.216
1  0.189189   0.174747  0.410000  0.805333        0.810
2  0.959459   0.622222  0.696667  0.809333        0.922
3  0.810811   0.404040  0.706667  0.872000        0.492
4  0.270270   0.909091  0.536667  0.828000        0.422
```

6 Predictive Analysis

The structure of predictive analysis - first we split the data as train and test, then choose a model, training the model, evaluate the model, and deploy the model for prediction.

7 Split the data into training and testing sets

[9]:
```
X = thyroid_data.drop('Diagnosis', axis=1)  # Features
y = thyroid_data['Diagnosis']  # Target variable

# Convert categorical features to numerical using one-hot encoding
X = pd.get_dummies(X, drop_first=True)

# random_state for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (170152, 28)
X_test shape: (42539, 28)
y_train shape: (170152,)
y_test shape: (42539,)
```

8 K-Nearest Neighbour

Grid search automates the process of testing multiple K values along with other hyperparameters. We can use GridSearchCV from scikit-learn libraries to search through different combinations of K values and evaluate the performance using cross-validation. The optimal value of k is found to be 5.

[34]:
```
# Use the optimal k found from GridSearchCV
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test,y_pred))
```

```
Accuracy: 0.7246526716660006
              precision    recall  f1-score   support

      Benign       0.77      0.92      0.84     32615
   Malignant       0.25      0.09      0.13      9924

    accuracy                           0.72     42539
   macro avg       0.51      0.50      0.48     42539
```

```
weighted avg       0.65      0.72      0.67     42539

[[29935  2680]
 [ 9033   891]]
```

9 Logistic Regression

[13]:
```python
# Initialize and train the logistic regression model
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)

# Make predictions on the test set
y_pred_logreg = logreg.predict(X_test)

# Evaluate the model
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
print(f"Logistic Regression Accuracy: {accuracy_logreg}")
print(classification_report(y_test, y_pred_logreg))
print(confusion_matrix(y_test, y_pred_logreg))
```

```
Logistic Regression Accuracy: 0.8250546557276852
              precision    recall  f1-score   support

      Benign       0.85      0.94      0.89     32615
   Malignant       0.69      0.45      0.54      9924

    accuracy                           0.83     42539
   macro avg       0.77      0.69      0.72     42539
weighted avg       0.81      0.83      0.81     42539

[[30657  1958]
 [ 5484  4440]]
```

10 Decision Tree

The optimization of decision tree classifier is performed by pre-pruning; maximum depth of the tree is used as a control variable for pre-pruning. Also, the attribute selection measure 'entropy' for the information gain is used as control parameter.

[16]:
```python
# Create Decision Tree classifer object
dtree = DecisionTreeClassifier(criterion="entropy", max_depth=5)

# Train Decision Tree Classifer
dtree.fit(X_train,y_train)

#Predict the response for test dataset
y_pred_dtree = dtree.predict(X_test)
```

```
# Evaluate the model
accuracy_dtree = accuracy_score(y_test, y_pred_dtree)
print(f"Decision Tree Accuracy: {accuracy_dtree}")
print(classification_report(y_test, y_pred_dtree))
print(confusion_matrix(y_test, y_pred_dtree))
```

```
Decision Tree Accuracy: 0.825007640047956
              precision    recall  f1-score   support

      Benign       0.85      0.94      0.89     32615
   Malignant       0.69      0.45      0.54      9924

    accuracy                           0.83     42539
   macro avg       0.77      0.69      0.72     42539
weighted avg       0.81      0.83      0.81     42539

[[30654  1961]
 [ 5483  4441]]
```

11 Random Forest

Create an instance of the Random forest model with the default parameters. Then fit the model to training data; pass both the features and the target variable so the model can learn to predict.

[17]:
```
# Train the Random Forest Classifier
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred_rf = rf_classifier.predict(X_test)

# Evaluate the model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Random Forest Accuracy: {accuracy_rf}")
print(classification_report(y_test, y_pred_rf))
print(confusion_matrix(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.8243024048520181
              precision    recall  f1-score   support

      Benign       0.85      0.94      0.89     32615
   Malignant       0.69      0.44      0.54      9924

    accuracy                           0.82     42539
   macro avg       0.77      0.69      0.72     42539
weighted avg       0.81      0.82      0.81     42539
```

```
[[30685  1930]
 [ 5544  4380]]
```

12 Principal Component Analysis

Since the dataset has both the continuous and categorical feature variables; the principal component analysis will be based on the FAMD() function from Python library Prince. The FAMD () function implements the principal component method dedicated to explore data with both continuous and categorical variables.

[27]:
```python
# prompt: PCA using prince library
pca = prince.PCA(n_components=2) # Reduce to 2 principal components
pca = pca.fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

# You can now use X_train_pca in your models
print(X_train_pca.head())
```

```
component         0         1
117332    -1.145959 -0.074020
63420      0.426001  1.437753
179948     0.572743  2.263643
187371    -1.081722  1.805283
103129     0.221810  0.270046
```

Prediction accuracy with KNN after PCA

[29]:
```python
# Example with KNN after PCA
knn_pca = KNeighborsClassifier(n_neighbors=5)
knn_pca.fit(X_train_pca, y_train)
y_pred_knn_pca = knn_pca.predict(X_test_pca)
accuracy_knn_pca = accuracy_score(y_test, y_pred_knn_pca)
print(f"KNN Accuracy with PCA: {accuracy_knn_pca}")
print(classification_report(y_test, y_pred_knn_pca))
print(confusion_matrix(y_test, y_pred_knn_pca))
```

```
KNN Accuracy with PCA: 0.7991020005171725
              precision    recall  f1-score   support

      Benign       0.83      0.92      0.88     32615
   Malignant       0.61      0.39      0.48      9924

    accuracy                           0.80     42539
   macro avg       0.72      0.66      0.68     42539
weighted avg       0.78      0.80      0.78     42539
```

```
[[30124  2491]
 [ 6055  3869]]
```

Prediction accuracy with decision Tree after PCA

[30]:
```
# Example with Decision Tree after PCA
dtree_pca = DecisionTreeClassifier(criterion="entropy", max_depth=5)
dtree_pca.fit(X_train_pca, y_train)
y_pred_dtree_pca = dtree_pca.predict(X_test_pca)
accuracy_dtree_pca = accuracy_score(y_test, y_pred_dtree_pca)
print(f"Decision Tree Accuracy with PCA: {accuracy_dtree_pca}")
print(classification_report(y_test, y_pred_dtree_pca))
print(confusion_matrix(y_test, y_pred_dtree_pca))
```

```
Decision Tree Accuracy with PCA: 0.8249371165283622
              precision    recall  f1-score   support

      Benign       0.85      0.94      0.89     32615
   Malignant       0.69      0.45      0.54      9924

    accuracy                           0.82     42539
   macro avg       0.77      0.69      0.72     42539
weighted avg       0.81      0.82      0.81     42539

[[30654  1961]
 [ 5486  4438]]
```

Prediction accuracy with Random Forest after PCA

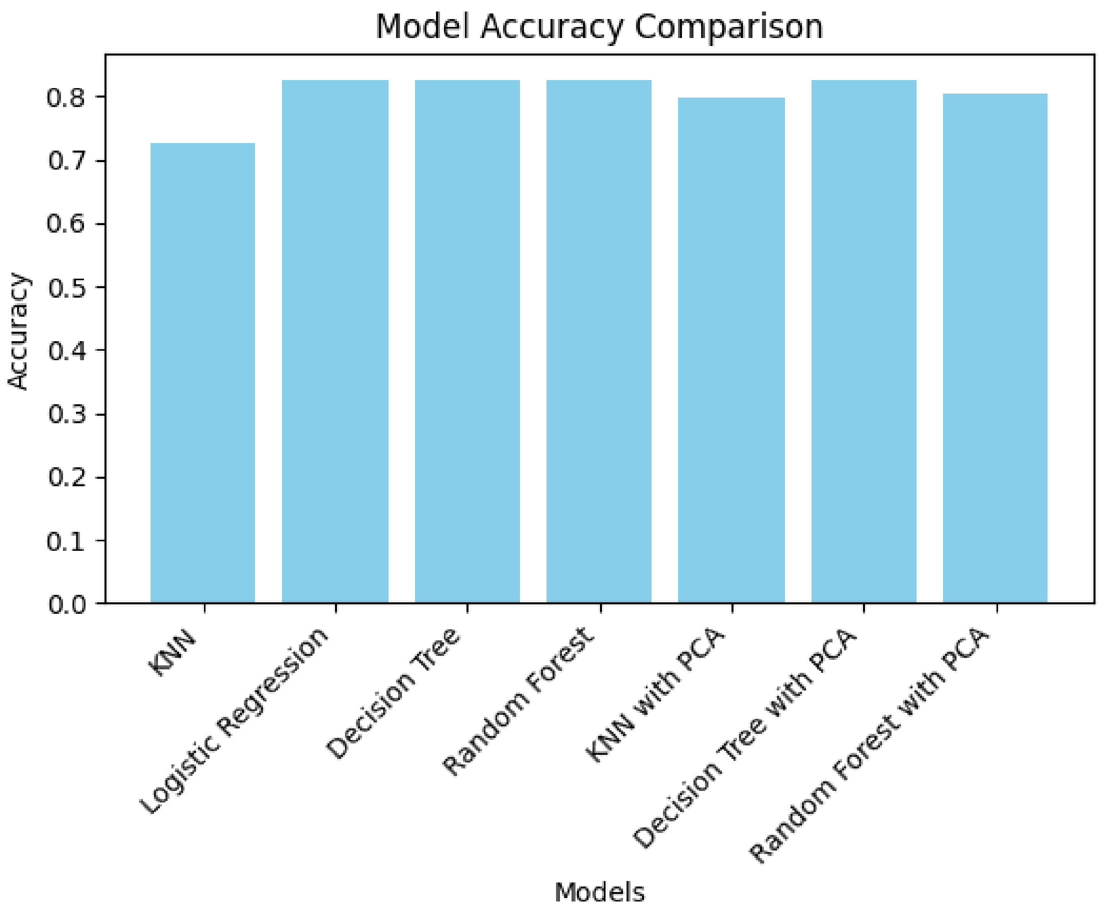
[31]:
```
# Example with Random Forest after PCA
rf_pca = RandomForestClassifier()
rf_pca.fit(X_train_pca, y_train)
y_pred_rf_pca = rf_pca.predict(X_test_pca)
accuracy_rf_pca = accuracy_score(y_test, y_pred_rf_pca)
print(f"Random Forest Accuracy with PCA: {accuracy_rf_pca}")
print(classification_report(y_test, y_pred_rf_pca))
print(confusion_matrix(y_test, y_pred_rf_pca))
```

```
Random Forest Accuracy with PCA: 0.8042737252873834
              precision    recall  f1-score   support

      Benign       0.83      0.93      0.88     32615
   Malignant       0.63      0.39      0.48      9924

    accuracy                           0.80     42539
   macro avg       0.73      0.66      0.68     42539
weighted avg       0.79      0.80      0.79     42539

[[30319  2296]
[ 6030  3894]]
```

13 Model Performance Comparison

Make a bar chart of all above accuracies found from different machine learning classifiers.

```
[39]: accuracies = {
          'KNN': accuracy,
          'Logistic Regression': accuracy_logreg,
          'Decision Tree': accuracy_dtree,
          'Random Forest': accuracy_rf,
          'KNN with PCA': accuracy_knn_pca,
          'Decision Tree with PCA': accuracy_dtree_pca,
          'Random Forest with PCA': accuracy_rf_pca
      }

      models = list(accuracies.keys())
      accuracy_values = list(accuracies.values())

      plt.figure(figsize=(6, 5))
      plt.bar(models, accuracy_values, color='skyblue')
      plt.xlabel("Models")
      plt.ylabel("Accuracy")
      plt.title("Model Accuracy Comparison")
      plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better␣
       ↪readability
      plt.tight_layout() # Adjust layout to prevent labels from overlapping
      plt.show()
```



Model accuracies in data frame view.

```
[40]: df = pd.DataFrame(list(accuracies.items()), columns=['Model', 'Accuracy'])
      df
```

```
[40]:                     Model  Accuracy
      0                     KNN  0.724653
      1     Logistic Regression  0.825055
      2           Decision Tree  0.825008
      3           Random Forest  0.824302
      4            KNN with PCA  0.799102
      5  Decision Tree with PCA  0.824937
      6  Random Forest with PCA  0.804274
```