This is my 15. Health Risk Prediction project

Import Required Libraries

In [1]:
```
# Data handling
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Preprocessing & splitting
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline,Pipeline
from sklearn.impute import SimpleImputer
from sklearn.utils import resample

# Evaluation metrics
from sklearn.metrics import (accuracy_score,precision_score,recall_score,f1_scor
    classification_report,confusion_matrix,auc,roc_auc_score, roc_curve
)

# Classifier Models
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,GradientB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

# XGBoost
from xgboost import XGBClassifier
# Warnings
import warnings
warnings.filterwarnings("ignore")

# Save the model
import joblib
```

Load Dataset

In [2]:
```
data = pd.read_csv('Maternal Health Risk Data Set.csv')
data.head()
```

Out[2]:

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25	130	80	15.0	98.0	86	high risk
1	35	140	90	13.0	98.0	70	high risk
2	29	90	70	8.0	100.0	80	high risk
3	30	140	85	7.0	98.0	70	high risk
4	35	120	60	6.1	98.0	76	low risk

EDA

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1014 entries, 0 to 1013
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          1014 non-null   int64  
 1   SystolicBP   1014 non-null   int64  
 2   DiastolicBP  1014 non-null   int64  
 3   BS           1014 non-null   float64
 4   BodyTemp     1014 non-null   float64
 5   HeartRate    1014 non-null   int64  
 6   RiskLevel    1014 non-null   object 
dtypes: float64(2), int64(4), object(1)
memory usage: 55.6+ KB
```

Checking for null values

In [4]:
```
data.isna().sum()
```

Out[4]:
```
Age            0
SystolicBP     0
DiastolicBP    0
BS             0
BodyTemp       0
HeartRate      0
RiskLevel      0
dtype: int64
```

In [5]:
```
for i in data.columns:
    print(f"{i}")
    print(f"{data[i].nunique()}")
```

```
Age
50
SystolicBP
19
DiastolicBP
16
BS
29
BodyTemp
8
HeartRate
16
RiskLevel
3
```

Check for duplicate rows

In [6]:
```
duplicate_rows = data[data.duplicated()]
```

Display the duplicate rows

In [7]:
```
print("Duplicate Rows:")
print(duplicate_rows)
```

```
Duplicate Rows:
      Age  SystolicBP  DiastolicBP    BS  BodyTemp  HeartRate  RiskLevel
67     19         120           80   7.0      98.0         70   mid risk
72     19         120           80   7.0      98.0         70   mid risk
97     19         120           80   7.0      98.0         70   mid risk
106    50         140           90  15.0      98.0         90  high risk
107    25         140          100   6.8      98.0         80  high risk
...   ...         ...          ...   ...       ...        ...        ...
1009   22         120           60  15.0      98.0         80  high risk
1010   55         120           90  18.0      98.0         60  high risk
1011   35          85           60  19.0      98.0         86  high risk
1012   43         120           90  18.0      98.0         70  high risk
1013   32         120           65   6.0     101.0         76   mid risk

[562 rows x 7 columns]
```

Show the number of duplicates

In [8]:
```
print("\nNumber of duplicate rows:", duplicate_rows.shape[0])
```

```
Number of duplicate rows: 562
```

Remove duplicate rows

In [9]:
```
data_cleaned = data.drop_duplicates()
```

Reset index after removing duplicates

In [10]:
```
data_cleaned = data_cleaned.reset_index(drop=True)
```

Check new shape

In [11]:
```
print("New dataset shape after removing duplicates:", data_cleaned.shape)
```

```
New dataset shape after removing duplicates: (452, 7)
```

Save the cleaned dataset to a new CSV file

In [12]:
```
data_cleaned.to_csv("Maternal_Health_Cleaned.csv", index=False)

data_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 452 entries, 0 to 451
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          452 non-null    int64  
 1   SystolicBP   452 non-null    int64  
 2   DiastolicBP  452 non-null    int64  
 3   BS           452 non-null    float64
 4   BodyTemp     452 non-null    float64
 5   HeartRate    452 non-null    int64  
 6   RiskLevel    452 non-null    object 
dtypes: float64(2), int64(4), object(1)
memory usage: 24.8+ KB
```

Check class distribution before

In [13]:
```
print("Before balancing:")
print(data_cleaned['RiskLevel'].value_counts())
```

```
Before balancing:
RiskLevel
low risk     234
high risk    112
mid risk     106
Name: count, dtype: int64
```

Find the maximum class count

In [14]:
```
max_count = data_cleaned['RiskLevel'].value_counts().max()
max_count
```

Out[14]: 234

Create a list to store upsampled DataFrames

In [15]:
```
upsampled_list = []
```

Loop through each class and upsample

In [16]:
```
for cls in data_cleaned['RiskLevel'].unique():
    cls_data = data_cleaned[data_cleaned['RiskLevel'] == cls]
    cls_upsampled = resample(
        cls_data,
        replace=True,
        n_samples=max_count,
        random_state=42
    )
    upsampled_list.append(cls_upsampled)
```

Combine all balanced class data

In [17]:
```
data_balanced = pd.concat(upsampled_list)
```

Shuffle the final dataset

In [18]:
```
data_balanced = data_balanced.sample(frac=1, random_state=42).reset_index(drop=T
```

Check class distribution after balancing

In [19]:
```
print("\nAfter balancing:")
print(data_balanced['RiskLevel'].value_counts())
```

```
After balancing:
RiskLevel
mid risk     234
high risk    234
low risk     234
Name: count, dtype: int64
```

Swap variable

In [20]:
```
data = data_balanced
data['RiskLevel'].value_counts()
```

Out[20]:
```
RiskLevel
mid risk     234
high risk    234
low risk     234
Name: count, dtype: int64
```

Encoding

In [21]:
```
le = LabelEncoder()
data['RiskLevel'] = le.fit_transform(data['RiskLevel'])
```

In [22]:
```
data.head(2)
```

Out[22]:

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate	RiskLevel
0	25	120	100	6.8	98.0	60	2
1	29	90	70	8.0	100.0	80	0

Float column covert to int

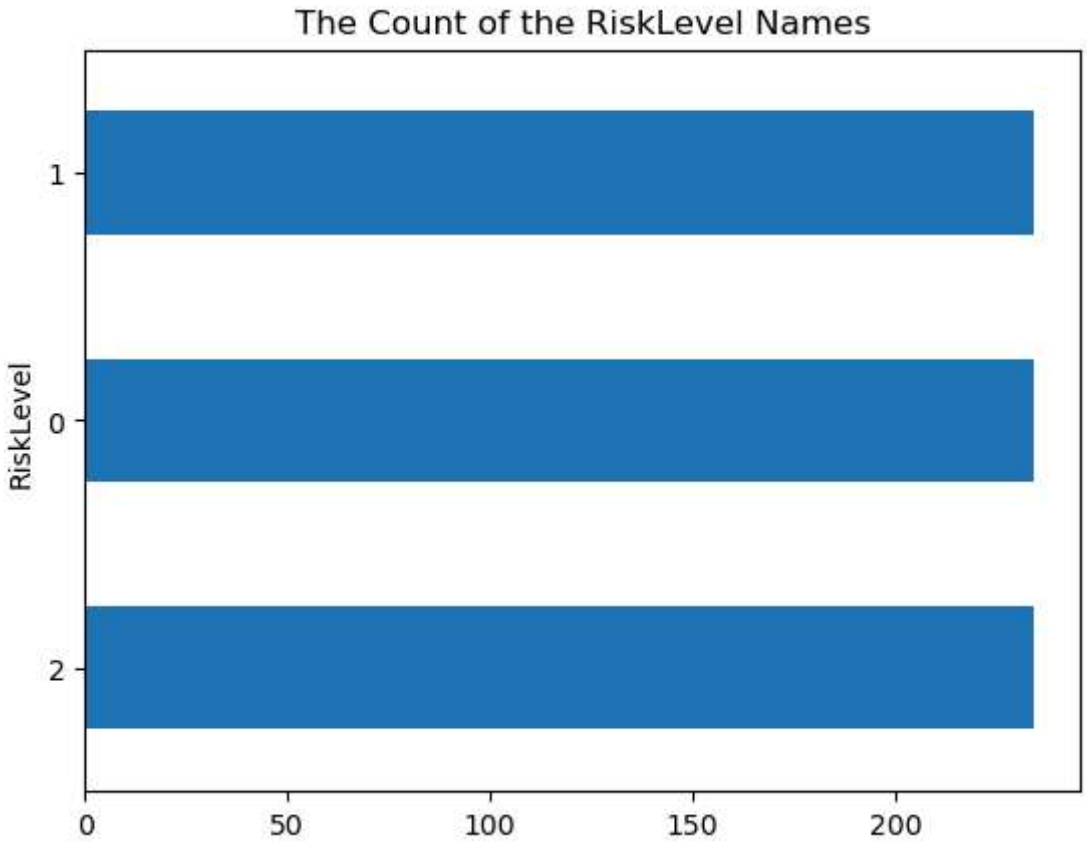
In [23]:
```
for i in [['BodyTemp','BS']]:
    data[i] = data[i].astype('int64')

data.dtypes
```

Out[23]:
```
Age            int64
SystolicBP     int64
DiastolicBP    int64
BS             int64
BodyTemp       int64
HeartRate      int64
RiskLevel      int32
dtype: object
```

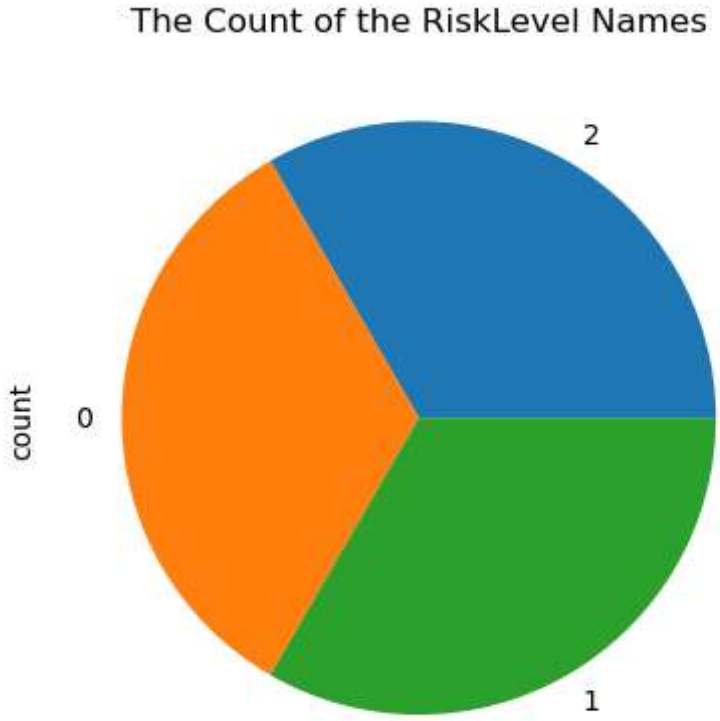
In [24]:
```
RiskLevel_vc  = data["RiskLevel"].value_counts()
RiskLevel_vc
```

Out[24]:
```
RiskLevel
2    234
0    234
1    234
Name: count, dtype: int64
```

In [25]:
```
RiskLevel_vc.plot(kind='barh',title='The Count of the RiskLevel Names')
plt.show()
```



In [26]:
```
RiskLevel_vc.plot(kind='pie',title='The Count of the RiskLevel Names')
plt.show()
```

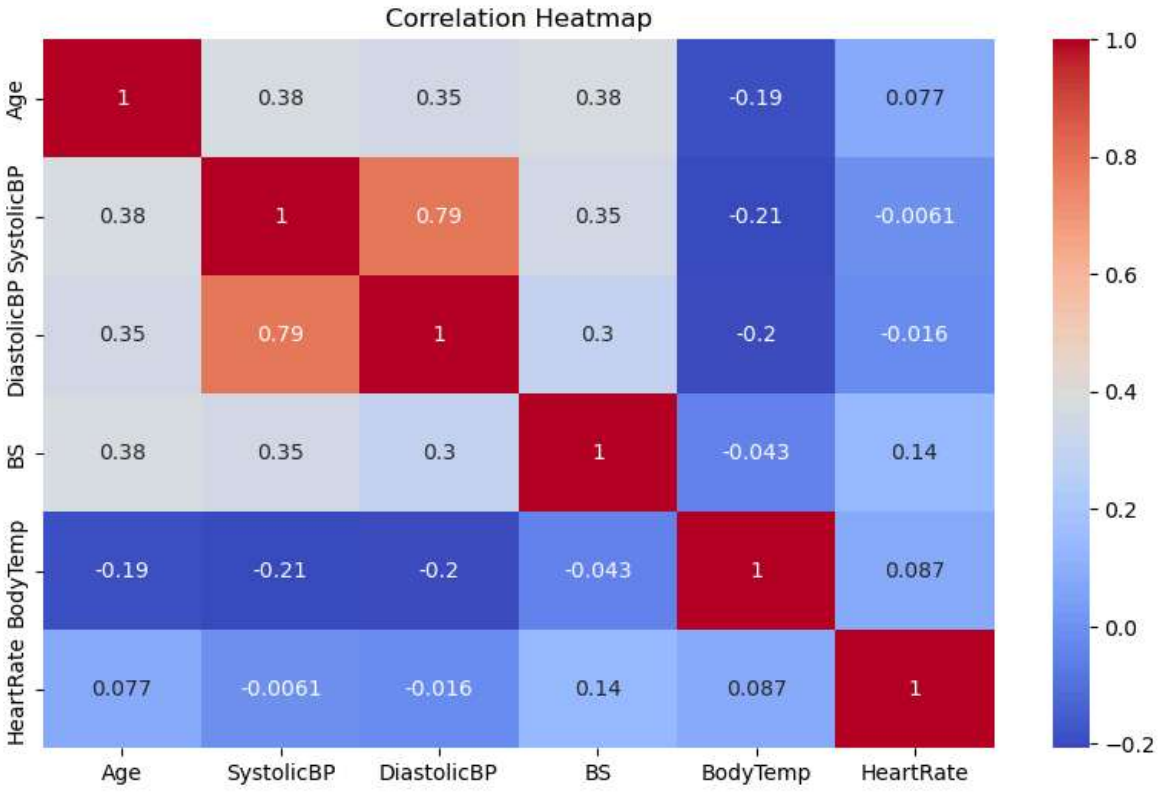


Splitting for Target and Feature into Train Test Split

In [27]:
```
X = data.drop(['RiskLevel'],axis=1)
y = data['RiskLevel']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

Heatmap

In [28]:
```
plt.figure(figsize=(10,6))
sns.heatmap(data_cleaned.corr(numeric_only=True), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```



Preprocessing pipeline

In [29]:
```
preprocessor = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('scaler', StandardScaler())
])
```

Define models & parameter grids for classification

In [30]:
```
models = {
    # ------------------------------
    "Naive Bayes": {
        "model": MultinomialNB(),
        "param_grid": {
            "alpha": [0.1, 0.5, 1.0]
```

```
        }
    },
    # ------------------------------
    "Logistic Regression": {
        "model": LogisticRegression(solver='lbfgs', max_iter=500, random_state=4
        "param_grid": {
            "C": [0.1, 1, 10],
            "penalty": ['l2']
        }
    },
    # ------------------------------
    "Decision Tree": {
        "model": DecisionTreeClassifier(random_state=42),
        "param_grid": {
            "max_depth": [3, 5, 10, None],
            "min_samples_split": [2, 5, 10],
            "min_samples_leaf": [1, 2, 4]
        }
    },
    # ------------------------------
    "Random Forest": {
        "model": RandomForestClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5],
            "min_samples_leaf": [1, 2]
        }
    },
    # ------------------------------
    "Gradient Boosting": {
        "model": GradientBoostingClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "learning_rate": [0.01, 0.1, 0.2],
            "max_depth": [3, 5]
        }
    },
    # ------------------------------
    "AdaBoost": {
        "model": AdaBoostClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [50, 100, 200],
            "learning_rate": [0.01, 0.1, 1.0]
        }
    },
    # ------------------------------
    "Extra Trees": {
        "model": ExtraTreesClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5],
            "min_samples_leaf": [1, 2]
        }
    },
    # ------------------------------
    "K-Nearest Neighbors": {
        "model": KNeighborsClassifier(),
        "param_grid": {
```

```
            "n_neighbors": [3, 5, 7, 9],
            "weights": ["uniform", "distance"],
            "p": [1, 2]  # Manhattan (L1) or Euclidean (L2)
        }
    },
    # ------------------------------
    "MLP Classifier": {
        "model": MLPClassifier(random_state=42),
        "param_grid": {
            "hidden_layer_sizes": [(50,), (100,)],
            "activation": ["relu", "tanh"],
            "solver": ["adam"],
            "max_iter": [200]
        }
    },
    # ------------------------------
    "XGBoost": {
        "model": XGBClassifier(use_label_encoder=False, eval_metric="logloss", r
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [3, 5, 7],
            "learning_rate": [0.01, 0.1, 0.2],
            "subsample": [0.8, 1.0],
            "colsample_bytree": [0.8, 1.0]
        }
    }
}

print(f" Total Models Loaded: {len(models)}")
```

```
Total Models Loaded: 10
```

Run GridSearchCV for each classification model

In [31]:

```
results = []
best_score = -float('inf')
best_model = None
best_name = None

# Run GridSearchCV for each classification model
for name, mp in models.items():
    print(f"\n Running GridSearchCV for {name}...")
    grid = GridSearchCV(mp['model'], param_grid=mp['param_grid'], cv=3, scoring=
    grid.fit(X_train, y_train)

    y_pred = grid.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted')
    rec = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    results.append({
        'Model': name,
        'Best Params': grid.best_params_,
        'Accuracy': acc,
        'Precision': prec,
```

```
        'Recall': rec,
        'F1 Score': f1
    })

    print(f"{name} Best CV Score (Accuracy): {grid.best_score_:.4f}")
    print(f"{name} Best Params: {grid.best_params_}")
    print(f"{name} Classification Report:\n{classification_report(y_test, y_pred

    if grid.best_score_ > best_score:
        best_score = grid.best_score_
        best_model = grid.best_estimator_
        best_name = name

# Output summary
results_df = pd.DataFrame(results)
print("\n Summary Results:")
print(results_df)

print(f"\n Best Model: {best_name}")
print(f" Best CV Score (Accuracy): {best_score:.4f}")
```

```
 Running GridSearchCV for Naive Bayes...
Naive Bayes Best CV Score (Accuracy): 0.4920
Naive Bayes Best Params: {'alpha': 0.1}
Naive Bayes Classification Report:
              precision    recall  f1-score   support

           0       0.57      0.59      0.58        46
           1       0.54      0.40      0.46        50
           2       0.49      0.62      0.55        45

    accuracy                           0.53       141
   macro avg       0.54      0.54      0.53       141
weighted avg       0.54      0.53      0.53       141


 Running GridSearchCV for Logistic Regression...
Logistic Regression Best CV Score (Accuracy): 0.5704
Logistic Regression Best Params: {'C': 10, 'penalty': 'l2'}
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.87      0.77        46
           1       0.70      0.52      0.60        50
           2       0.41      0.42      0.42        45

    accuracy                           0.60       141
   macro avg       0.60      0.60      0.59       141
weighted avg       0.61      0.60      0.60       141


 Running GridSearchCV for Decision Tree...
Decision Tree Best CV Score (Accuracy): 0.7790
Decision Tree Best Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_sample
s_split': 2}
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.83      0.83        46
           1       0.81      0.70      0.75        50
           2       0.63      0.73      0.68        45

    accuracy                           0.75       141
   macro avg       0.76      0.75      0.75       141
weighted avg       0.76      0.75      0.75       141


 Running GridSearchCV for Random Forest...
Random Forest Best CV Score (Accuracy): 0.7825
Random Forest Best Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_sample
s_split': 2, 'n_estimators': 100}
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.89      0.87        46
           1       0.88      0.74      0.80        50
           2       0.71      0.80      0.75        45

    accuracy                           0.81       141
   macro avg       0.81      0.81      0.81       141
weighted avg       0.82      0.81      0.81       141
```

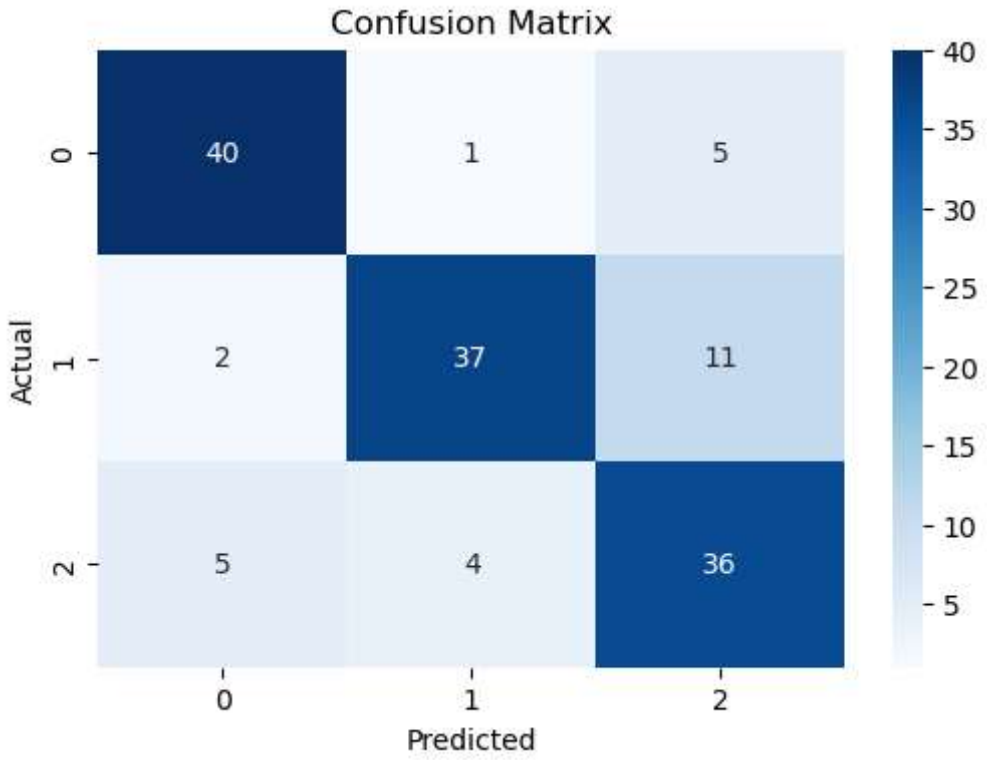
```
 Running GridSearchCV for Gradient Boosting...
Gradient Boosting Best CV Score (Accuracy): 0.7897
Gradient Boosting Best Params: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimato
rs': 200}
Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.89      0.87        46
           1       0.93      0.78      0.85        50
           2       0.73      0.82      0.77        45

    accuracy                           0.83       141
   macro avg       0.84      0.83      0.83       141
weighted avg       0.84      0.83      0.83       141


 Running GridSearchCV for AdaBoost...
AdaBoost Best CV Score (Accuracy): 0.6078
AdaBoost Best Params: {'learning_rate': 0.1, 'n_estimators': 200}
AdaBoost Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.76      0.79        46
           1       0.79      0.68      0.73        50
           2       0.53      0.64      0.58        45

    accuracy                           0.70       141
   macro avg       0.71      0.70      0.70       141
weighted avg       0.71      0.70      0.70       141


 Running GridSearchCV for Extra Trees...
Extra Trees Best CV Score (Accuracy): 0.7861
Extra Trees Best Params: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_sp
lit': 2, 'n_estimators': 200}
Extra Trees Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.89      0.87        46
           1       0.84      0.76      0.80        50
           2       0.71      0.76      0.73        45

    accuracy                           0.80       141
   macro avg       0.80      0.80      0.80       141
weighted avg       0.80      0.80      0.80       141


 Running GridSearchCV for K-Nearest Neighbors...
K-Nearest Neighbors Best CV Score (Accuracy): 0.7736
K-Nearest Neighbors Best Params: {'n_neighbors': 9, 'p': 2, 'weights': 'distanc
e'}
K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.85      0.84        46
           1       0.84      0.72      0.77        50
           2       0.65      0.73      0.69        45
```

```
    accuracy                           0.77       141
   macro avg       0.77      0.77      0.77       141
weighted avg       0.77      0.77      0.77       141


 Running GridSearchCV for MLP Classifier...
MLP Classifier Best CV Score (Accuracy): 0.6114
MLP Classifier Best Params: {'activation': 'tanh', 'hidden_layer_sizes': (100,),
'max_iter': 200, 'solver': 'adam'}
MLP Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.74      0.77        46
           1       0.71      0.60      0.65        50
           2       0.53      0.67      0.59        45

    accuracy                           0.67       141
   macro avg       0.68      0.67      0.67       141
weighted avg       0.69      0.67      0.67       141


 Running GridSearchCV for XGBoost...
XGBoost Best CV Score (Accuracy): 0.7897
XGBoost Best Params: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth':
7, 'n_estimators': 200, 'subsample': 0.8}
XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.87      0.86        46
           1       0.88      0.74      0.80        50
           2       0.69      0.80      0.74        45

    accuracy                           0.80       141
   macro avg       0.81      0.80      0.80       141
weighted avg       0.81      0.80      0.80       141


 Summary Results:
                 Model                                        Best Params  \
0          Naive Bayes                                    {'alpha': 0.1}
1  Logistic Regression                          {'C': 10, 'penalty': 'l2'}
2        Decision Tree  {'max_depth': None, 'min_samples_leaf': 1, 'mi...
3        Random Forest  {'max_depth': None, 'min_samples_leaf': 1, 'mi...
4    Gradient Boosting  {'learning_rate': 0.1, 'max_depth': 5, 'n_esti...
5             AdaBoost        {'learning_rate': 0.1, 'n_estimators': 200}
6          Extra Trees  {'max_depth': 20, 'min_samples_leaf': 1, 'min_...
7  K-Nearest Neighbors  {'n_neighbors': 9, 'p': 2, 'weights': 'distance'}
8       MLP Classifier  {'activation': 'tanh', 'hidden_layer_sizes': (...
9              XGBoost  {'colsample_bytree': 1.0, 'learning_rate': 0.1...

   Accuracy  Precision    Recall  F1 Score
0  0.531915   0.535871  0.531915  0.527688
1  0.602837   0.606002  0.602837  0.596176
2  0.751773   0.760676  0.751773  0.753567
3  0.808511   0.816340  0.808511  0.809185
4  0.829787   0.839484  0.829787  0.831252
5  0.695035   0.714213  0.695035  0.700985
6  0.801418   0.804177  0.801418  0.801637
7  0.765957   0.774102  0.765957  0.767573
8  0.666667   0.685366  0.666667  0.671097
```

```
9  0.801418   0.810996  0.801418  0.802761

Best Model: Gradient Boosting
Best CV Score (Accuracy): 0.7897
```

Confusion Matrix

In [32]:
```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=set(y_test), ytic
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



Saving the model

In [35]:
```
feature_columns = X.columns.to_list()
joblib.dump(feature_columns,'15.Health Risk Prediction.joblib')
```

Out[35]:
```
['15.Health Risk Prediction.joblib']
```

In [36]:
```
joblib.dump(best_model,'bestmodel 15.Health Risk Prediction.joblib')
```

Out[36]:
```
['bestmodel 15.Health Risk Prediction.joblib']
```

If you have any suggestions, please DM me.¶

Even a small message from you can make a big impact on my career

I am arun kumar

In [ ]: