Assignment on Ensemble Learning

Course name: Data Science

Course code: CSEL – 42---

Submitted By

Farhana Akter Suci

ID: B190305001

&

Rifah Sajida Deya

ID: B190305004

Submitted To

Dr. Md. Manowarul Islam

Associate Professor, Department of C S E, Jagannath University

5thth February , 2025

1) Load and Preprocess the Data:

```
import pandas as pd
```

```
data = pd.read_csv("Maternal Health Risk Data Set.csv")
```

```
data
```

	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
...	...	...	...	...	...	...	...
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

1014 rows × 7 columns

```
data.shape
```

```
(1014, 7)
```

```
data.isnull().sum().sum()
```

```
0
```

There is no null value in this dataet

```
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
```

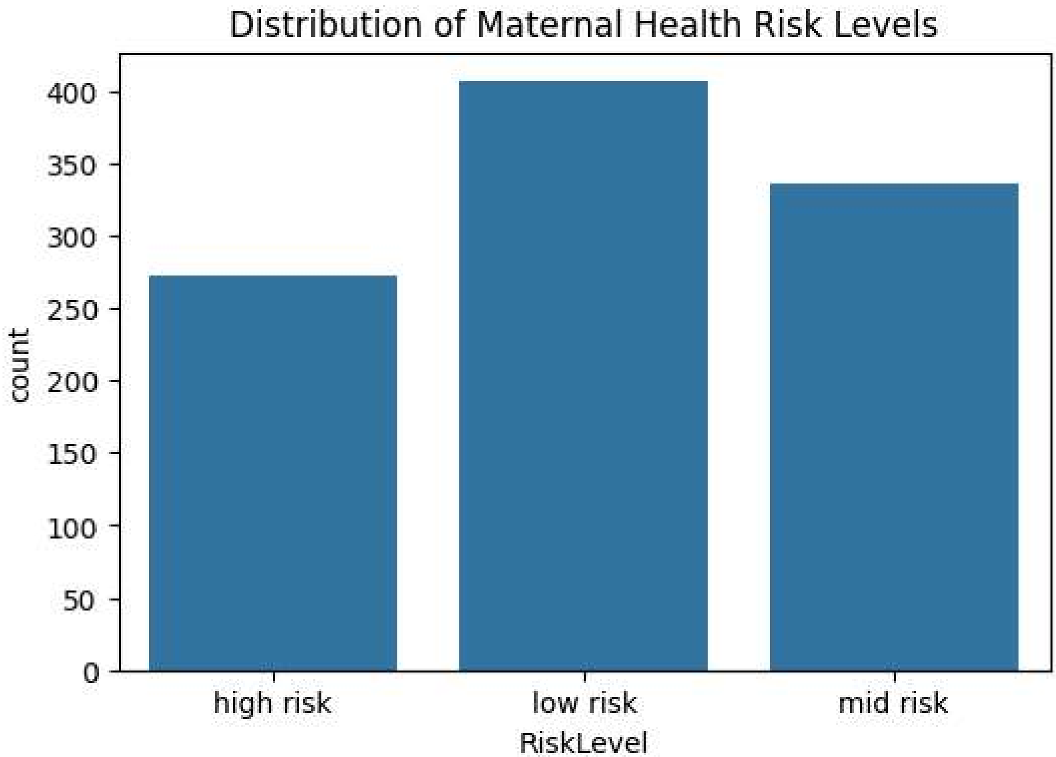
```
data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Age	1014.0	29.871795	13.474386	10.0	19.0	26.0	39.0	70.0
SystolicBP	1014.0	113.198225	18.403913	70.0	100.0	120.0	120.0	160.0
DiastolicBP	1014.0	76.460552	13.885796	49.0	65.0	80.0	90.0	100.0
BS	1014.0	8.725986	3.293532	6.0	6.9	7.5	8.0	19.0
BodyTemp	1014.0	98.665089	1.371384	98.0	98.0	98.0	98.0	103.0
HeartRate	1014.0	74.301775	8.088702	7.0	70.0	76.0	80.0	90.0

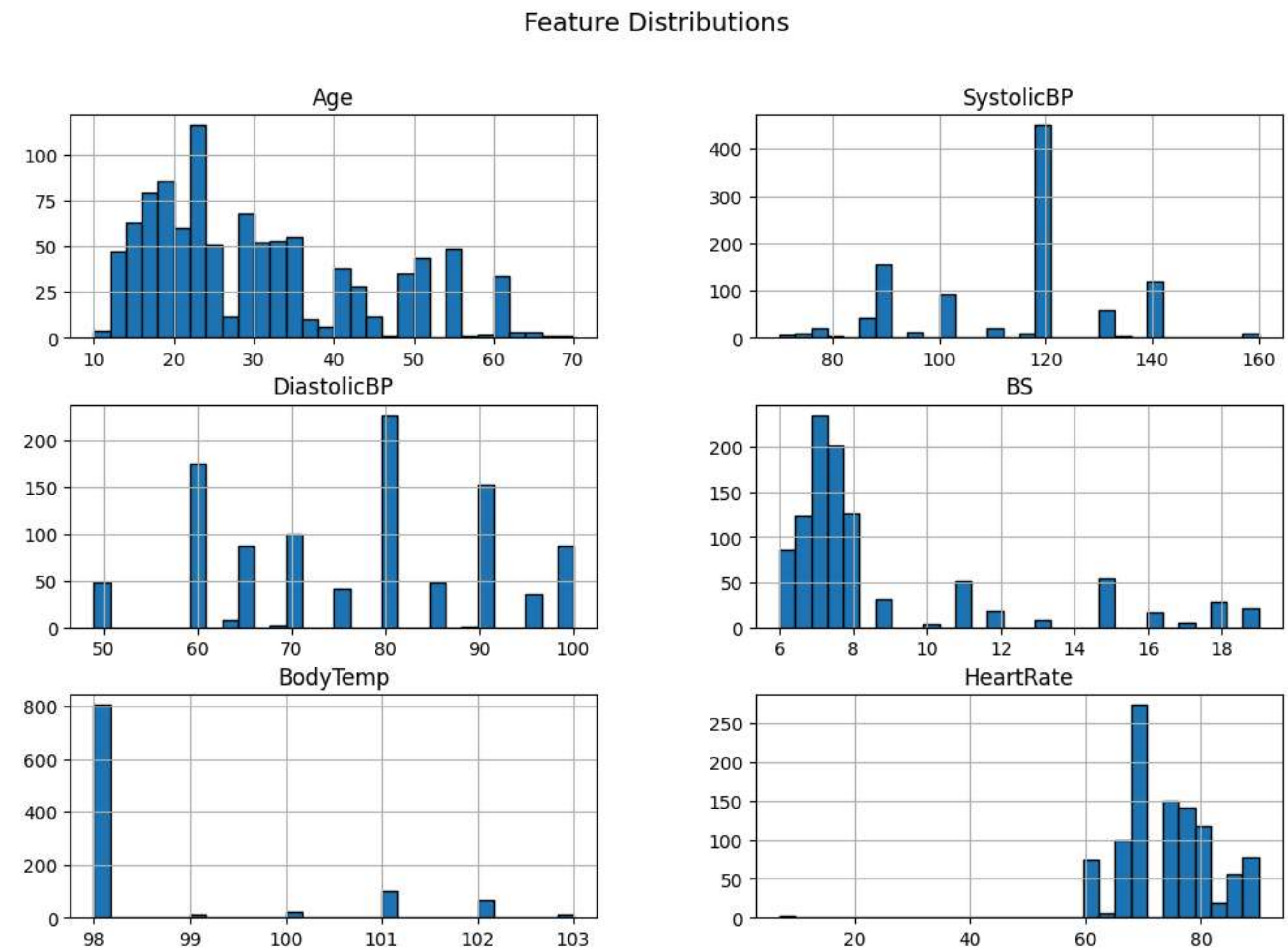
```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

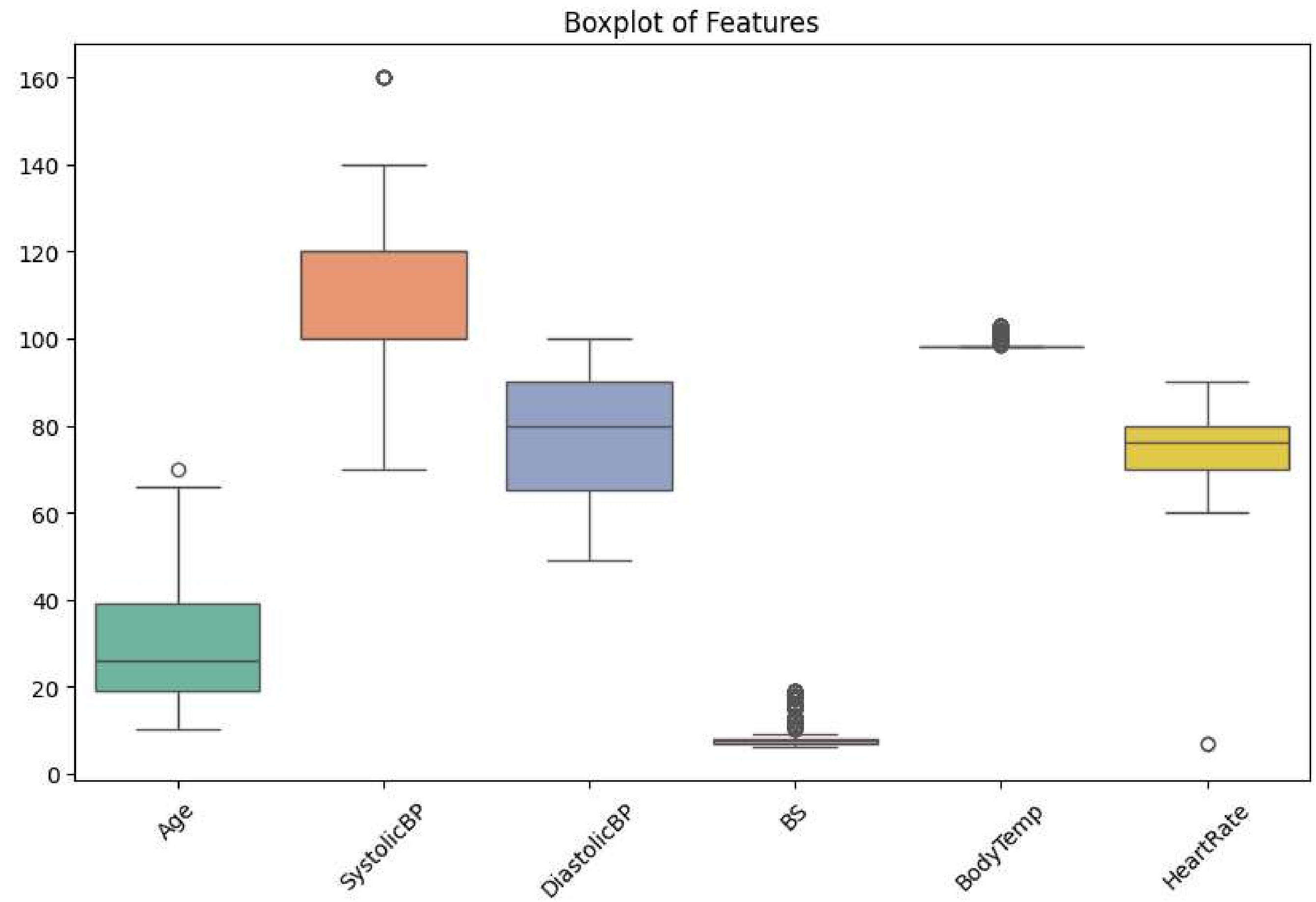
```
plt.figure(figsize=(6, 4))
sns.countplot(x="RiskLevel", data=data)
plt.title("Distribution of Maternal Health Risk Levels")
plt.show()
```




```
data.hist(figsize=(12, 8), bins=30, edgecolor='black')
plt.suptitle("Feature Distributions", fontsize=14)
plt.show()
```




```
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, palette="Set2")
plt.xticks(rotation=45)
plt.title("Boxplot of Features")
plt.show()
```



Label Encoding the target column

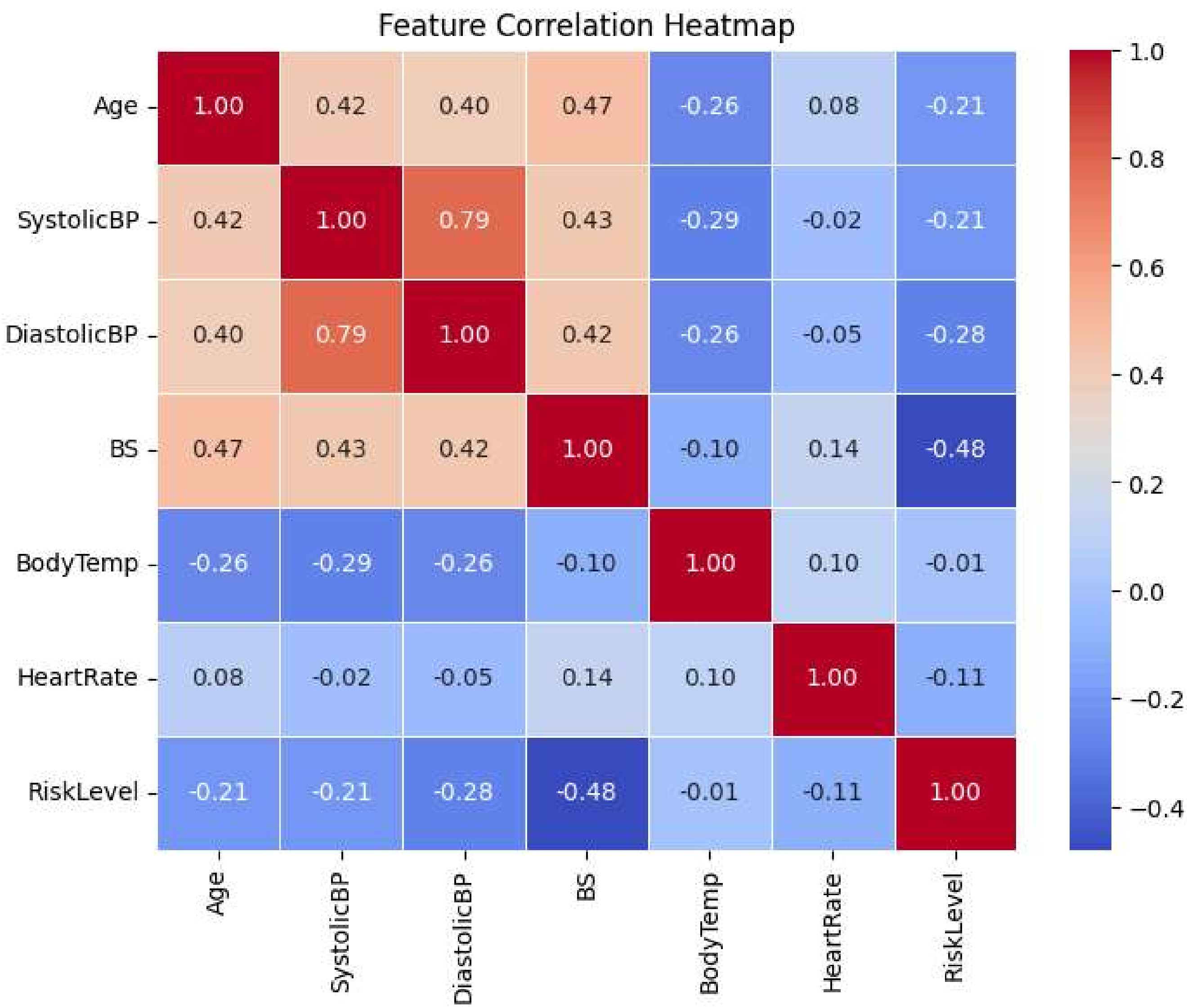
```
from sklearn.preprocessing import LabelEncoder
```

```
label_encoder=LabelEncoder()
```

```
data['RiskLevel']= label_encoder.fit_transform(data['RiskLevel'])
```

```
label_encoder = LabelEncoder()
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(data.corr(), annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.show()
```



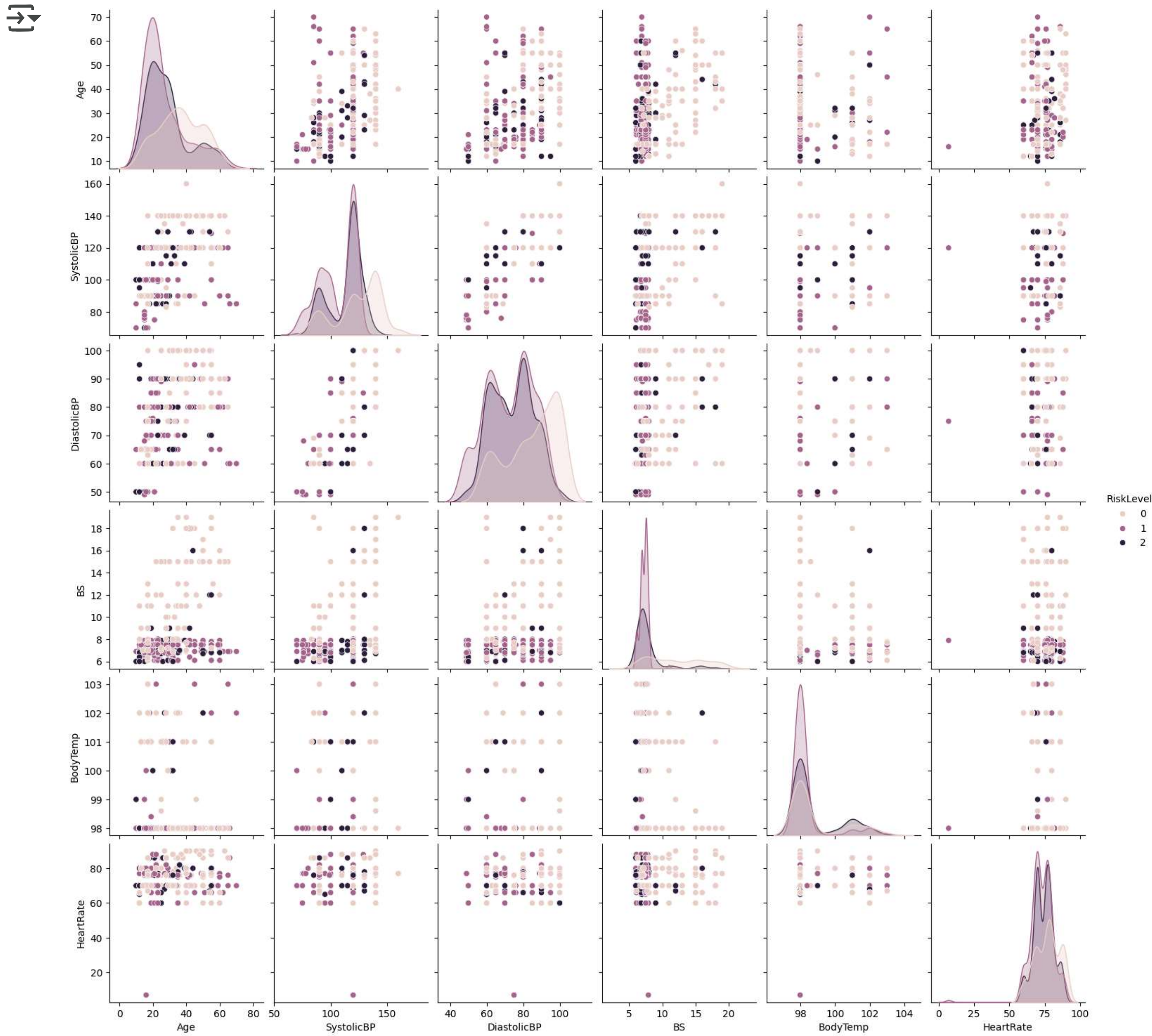
Skewness Analysis

```
from scipy.stats import skew

skewness = data.skew()
print("Feature Skewness:\n", skewness)
```

```
Feature Skewness:
 Age            0.783063
SystolicBP    -0.251189
DiastolicBP   -0.048441
BS             1.868203
BodyTemp       1.750988
HeartRate     -1.043525
RiskLevel     -0.108748
dtype: float64
```

```
sns.pairplot(data, hue="RiskLevel")
plt.show()
```



Age
SystolicBP
DiastolicBP
BS
BodyTemp
HeartRate
RiskLevel
0
1
2

Split the dataset

```
X = data.iloc[:, :-1]
```

```
X
```

	Age	SystolicBP	DiastolicBP	BS	BodyTemp	HeartRate
0	25	130	80	15.0	98.0	86
1	35	140	90	13.0	98.0	70
2	29	90	70	8.0	100.0	80
3	30	140	85	7.0	98.0	70
4	35	120	60	6.1	98.0	76
...	...	...	...	...	...	...
1009	22	120	60	15.0	98.0	80
1010	55	120	90	18.0	98.0	60
1011	35	85	60	19.0	98.0	86
1012	43	120	90	18.0	98.0	70
1013	32	120	65	6.0	101.0	76

1014 rows × 6 columns

```
y = data.iloc[:, -1]
```

```
y
```

```
0       0
1       0
2       0
3       0
4       1
       ..
1009    0
1010    0
1011    0
1012    0
1013    2
Name: RiskLevel, Length: 1014, dtype: int64
```

```
y.unique()
```

```
array([0, 1, 2])
```

```
y.nunique()
```

```
3
```

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
X = scaler.fit_transform(X)
```

Label Encoding the target column

```
y = data['RiskLevel']
```

```
y
```

```
0       0
1       0
2       0
3       0
4       1
       ..
1009    0
```

```
1010    0
1011    0
1012    0
1013    2
Name: RiskLevel, Length: 1014, dtype: int64
```

Split dataset: 80% training, 20% testing

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((811, 6), (203, 6), (811,), (203,))
```

Double-click (or enter) to edit

2) Bagging Implementation:

```
from sklearn.ensemble import BaggingClassifier
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
model1 = BaggingClassifier(estimator=DecisionTreeClassifier(), n_estimators=50, random_state
```

```
model1.fit(X_train, y_train)
```

```
▸ BaggingClassifier
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

```
test_predictions = model1.predict(X_test)
```

```
test_predictions
```

```
array([2, 0, 0, 1, 1, 2, 2, 2, 2, 1, 0, 2, 2, 0, 2, 2, 0, 1, 1, 2, 2, 2,
       0, 0, 1, 2, 0, 2, 1, 1, 2, 0, 1, 2, 1, 0, 0, 2, 1, 0, 0, 1, 0, 0,
       1, 2, 0, 2, 1, 1, 2, 1, 1, 2, 2, 2, 0, 1, 1, 2, 1, 2, 2, 2, 1, 0,
```

```
       2, 0, 2, 0, 0, 0, 2, 2, 2, 0, 1, 2, 0, 2, 1, 2, 1, 1, 2, 1, 1, 2,
       0, 2, 2, 0, 1, 0, 2, 1, 2, 1, 0, 2, 1, 2, 1, 1, 2, 2, 1, 2, 1, 2,
       2, 2, 1, 0, 2, 1, 1, 0, 2, 0, 2, 1, 1, 2, 0, 0, 2, 2, 2, 2, 1, 1,
       2, 2, 1, 1, 0, 2, 1, 2, 0, 2, 0, 0, 1, 1, 0, 1, 1, 2, 1, 2, 1, 2,
       0, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 2, 2, 1, 1, 0, 1, 2, 1, 2, 0,
       2, 1, 1, 1, 0, 2, 1, 2, 0, 1, 0, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 0,
       1, 2, 2, 2, 0])
```

```
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix
```

```
accuracy1=accuracy_score(y_test, test_predictions)
accuracy1
```

```
0.8078817733990148
```

```
precision1=precision_score(y_test, test_predictions, average='weighted')
precision1
```

```
0.814527289486854
```

```
recall1=recall_score(y_test, test_predictions, average='weighted')
recall1
```

```
0.8078817733990148
```

```
f11=f1_score(y_test, test_predictions, average='weighted')
f11
```

```
0.8084290104530727
```

Show accuracy matrix as a Dataframe

```
accuracy_matrix = pd.DataFrame({
    "Metric": ["Accuracy", "Precision", "Recall", "F1-Score"],
    "Score": [accuracy1, precision1, recall1, f11]
})
```

```
accuracy_matrix
```

	Metric	Score
0	Accuracy	0.807882
1	Precision	0.814527
2	Recall	0.807882
3	F1-Score	0.808429

Compute confusion matrix

```
conf_matrix1 = confusion_matrix(y_test, test_predictions)
```

```
conf_matrix1
```

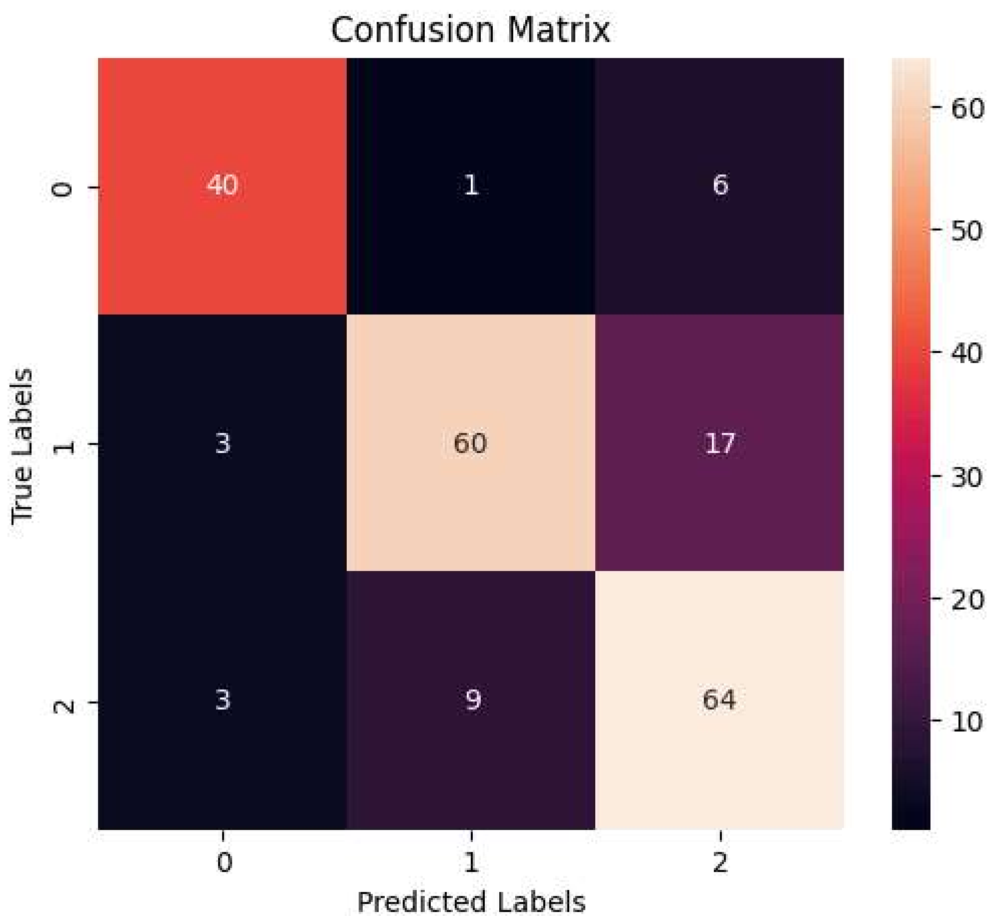
```
array([[40,  1,  6],
       [ 3, 60, 17],
       [ 3,  9, 64]])
```

Plot confusion matrix

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix1, annot=True)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```



3) Boosting Implementation:

```
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
```

```
from xgboost import XGBClassifier
```

```
model2=AdaBoostClassifier(n_estimators=50, random_state=42)
```

```
model3= GradientBoostingClassifier(n_estimators=50, random_state=42)
```

```
model4=XGBClassifier(n_estimators=50, eval_metric='mlogloss', random_state=42)
```

```
model2.fit(X_train, y_train)
```

```
▾ AdaBoostClassifier
AdaBoostClassifier(random_state=42)
```

```
model3.fit(X_train, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(n_estimators=50, random_state=42)
```

```
model4.fit(X_train, y_train)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='mlogloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=50,
              n_jobs=None, num_parallel_tree=None, objective='multi:softprob', ...)
```

AdaBoostClassifier Evaluation

```
test_predictions = model2.predict(X_test)
test_predictions
```

```
array([2, 0, 0, 1, 1, 2, 1, 0, 2, 1, 0, 2, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2,
       1, 2, 1, 1, 2, 1, 1, 1, 1, 0, 1, 2, 1, 0, 0, 1, 1, 0, 2, 2, 0, 0,
       1, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1, 1, 0,
       2, 1, 1, 0, 0, 0, 0, 2, 2, 0, 1, 2, 0, 2, 1, 1, 1, 1, 2, 1, 1, 0,
       0, 2, 2, 0, 1, 2, 2, 1, 2, 1, 0, 2, 1, 0, 1, 1, 2, 2, 1, 2, 1, 2,
       2, 1, 2, 2, 0, 1, 1, 0, 1, 0, 0, 1, 2, 1, 1, 0, 2, 2, 2, 2, 1, 1,
       2, 2, 1, 2, 0, 2, 2, 1, 0, 2, 2, 2, 1, 2, 0, 1, 1, 2, 2, 2, 2, 2,
       0, 2, 1, 1, 2, 0, 1, 1, 2, 2, 1, 2, 2, 1, 1, 2, 2, 1, 2, 1, 2, 0,
       2, 1, 1, 1, 0, 2, 2, 0, 0, 1, 0, 2, 1, 2, 2, 1, 1, 2, 1, 1, 2, 0,
       1, 1, 1, 2, 2])
```

```
accuracy2=accuracy_score(y_test, test_predictions)
accuracy2
```

```
0.6945812807881774
```

```
precision2=precision_score(y_test, test_predictions, average='weighted')
precision2
```

```
0.7036267138434321
```

```
recall2=recall_score(y_test, test_predictions, average='weighted')
recall2
```

```
0.6945812807881774
```

```
f12=f1_score(y_test, test_predictions, average='weighted')
f12
```

```
0.69643113617261
```

Show accuracy matrix as a Dataframe

```
accuracy_matrix = pd.DataFrame({
    "Metric": ["Accuracy", "Precision", "Recall", "F1-Score"],
    "Score": [accuracy2, precision2, recall2, f12]
})
```

```
accuracy_matrix
```

	Metric	Score
0	Accuracy	0.694581
1	Precision	0.703627
2	Recall	0.694581
3	F1-Score	0.696431

Compute confusion matrix¶

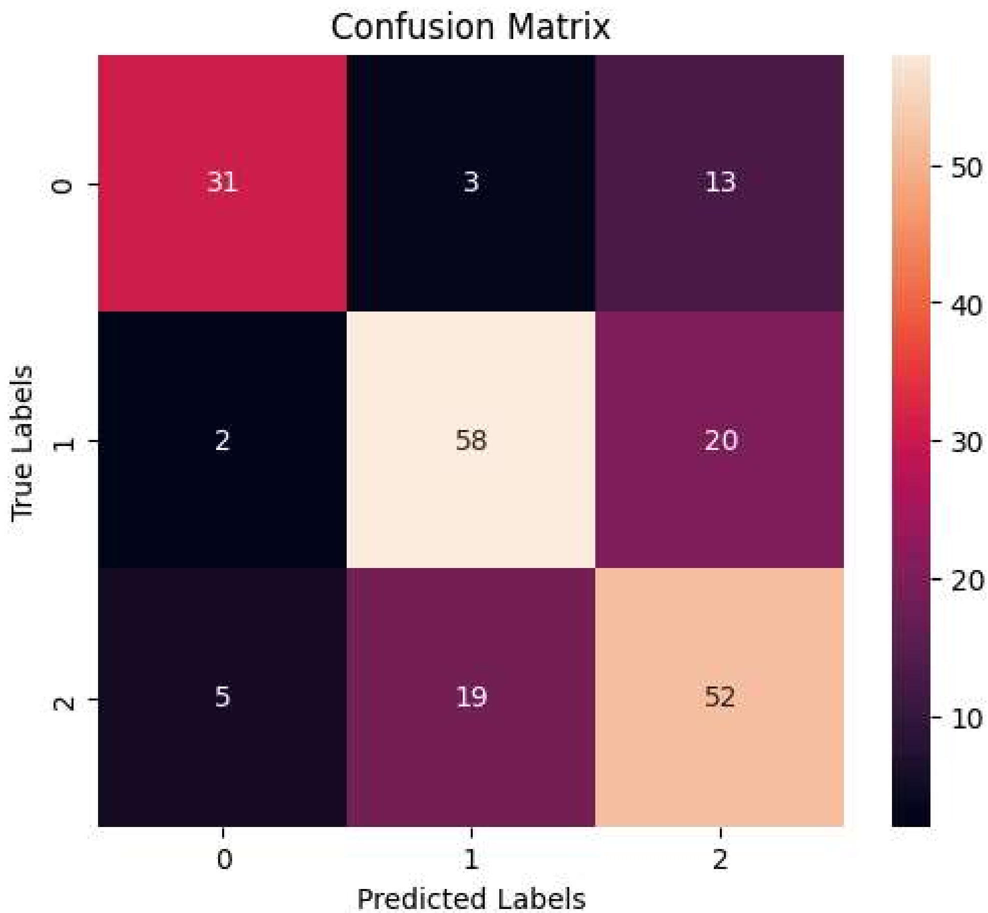
```
conf_matrix = confusion_matrix(y_test, test_predictions)
```

```
conf_matrix
```

```
array([[31,  3, 13],
       [ 2, 58, 20],
       [ 5, 19, 52]])
```

```
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
```

```
plt.show()
```



GradientBoostingClassifier Evaluation

```
test_predictions = model3.predict(X_test)
test_predictions
```

```
array([2, 0, 0, 1, 1, 1, 1, 2, 2, 1, 0, 1, 1, 1, 2, 1, 0, 1, 1, 2, 2, 2,
       2, 0, 1, 2, 0, 1, 1, 1, 1, 0, 1, 2, 1, 0, 0, 2, 1, 0, 0, 2, 0, 0,
       1, 2, 0, 2, 1, 1, 1, 1, 1, 2, 2, 2, 0, 1, 1, 1, 1, 1, 2, 1, 1, 0,
       2, 1, 2, 0, 0, 0, 2, 2, 2, 0, 1, 2, 0, 1, 1, 2, 1, 1, 2, 1, 1, 2,
       0, 2, 2, 2, 1, 0, 2, 1, 2, 1, 0, 2, 1, 0, 2, 1, 2, 2, 1, 2, 1, 2,
       2, 2, 1, 0, 2, 1, 1, 0, 2, 0, 2, 1, 1, 2, 1, 0, 2, 2, 1, 2, 1, 1,
       2, 2, 1, 1, 0, 2, 1, 2, 0, 2, 0, 0, 1, 1, 0, 1, 1, 2, 1, 2, 2, 2,
       0, 2, 1, 2, 1, 0, 1, 1, 1, 2, 1, 2, 0, 1, 1, 1, 0, 1, 2, 1, 1, 0,
       2, 1, 1, 1, 0, 1, 2, 0, 0, 1, 0, 1, 1, 1, 2, 2, 2, 2, 1, 1, 2, 0,
       1, 1, 2, 2, 0])
```

```
accuracy3=accuracy_score(y_test, test_predictions)
accuracy3
```

```
0.7487684729064039
```

```python
precision3=precision_score(y_test, test_predictions, average='weighted')
precision3
```

```
0.7495390214602038
```

```python
recall3=recall_score(y_test, test_predictions, average='weighted')
recall3
```

```
0.7487684729064039
```

```python
f13=f1_score(y_test, test_predictions, average='weighted')
f13
```

```
0.7479471055637963
```

Show accuracy matrix as a Dataframe

```python
accuracy_matrix = pd.DataFrame({
    "Metric": ["Accuracy", "Precision", "Recall", "F1-Score"],
    "Score": [accuracy3, precision3, recall3, f13]
})
```

```python
accuracy_matrix
```

	Metric	Score
0	Accuracy	0.748768
1	Precision	0.749539
2	Recall	0.748768
3	F1-Score	0.747947

Compute confusion matrix¶

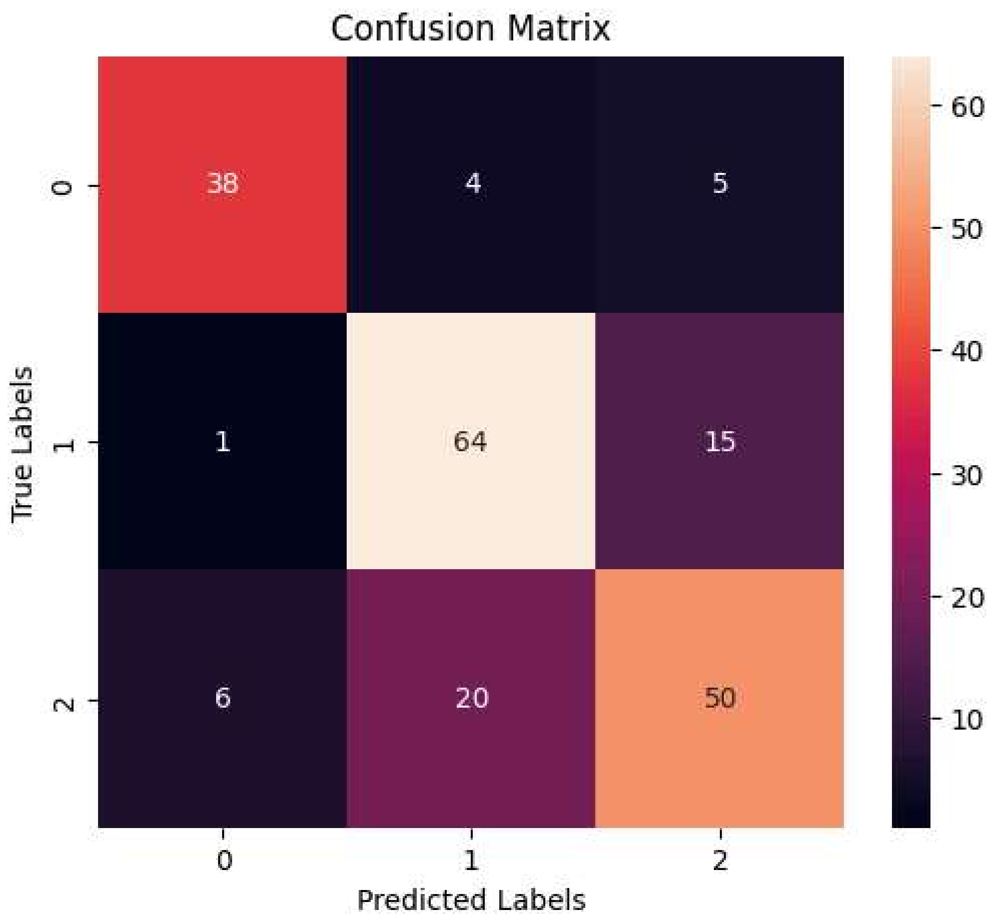
```python
conf_matrix = confusion_matrix(y_test, test_predictions)
```

```python
conf_matrix
```

```
array([[38,  4,  5],
       [ 1, 64, 15],
```

```
           [ 6, 20, 50]])
```

```
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```



XGBClassifier Evaluation

```
test_predictions = model4.predict(X_test)
test_predictions
```

```
array([2, 0, 0, 1, 1, 2, 2, 2, 2, 1, 0, 2, 1, 0, 2, 2, 0, 1, 1, 2, 2, 2,
       0, 0, 1, 2, 0, 1, 1, 1, 2, 0, 1, 2, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0,
       1, 2, 0, 2, 1, 1, 2, 1, 1, 2, 2, 2, 0, 1, 1, 2, 1, 2, 2, 1, 1, 0,
       2, 0, 2, 0, 0, 0, 1, 2, 2, 0, 1, 2, 0, 2, 1, 2, 1, 1, 2, 1, 1, 2,
       0, 2, 2, 0, 1, 0, 2, 1, 2, 1, 0, 2, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2,
       2, 2, 1, 0, 2, 1, 1, 0, 1, 0, 2, 1, 1, 2, 0, 0, 2, 2, 2, 2, 1, 1,
       2, 2, 1, 1, 0, 2, 1, 2, 0, 2, 0, 0, 1, 1, 0, 1, 1, 2, 1, 2, 1, 2,
       0, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 2, 1, 1, 1, 0, 1, 2, 1, 2, 0,
```

```
       2, 1, 1, 1, 0, 2, 1, 0, 0, 1, 0, 1, 1, 1, 2, 2, 2, 2, 1, 1, 2, 0,
       1, 2, 2, 2, 0])
```

```
accuracy4=accuracy_score(y_test, test_predictions)
accuracy4
```

```
0.8423645320197044
```

```
precision4=precision_score(y_test, test_predictions, average='weighted')
precision4
```

```
0.8427400801169274
```

```
recall4=recall_score(y_test, test_predictions, average='weighted')
recall4
```

```
0.8423645320197044
```

```
f14=f1_score(y_test, test_predictions, average='weighted')
f14
```

```
0.8423566439324042
```

Show accuracy matrix as a Dataframe

```
accuracy_matrix4 = pd.DataFrame({
    "Metric": ["Accuracy", "Precision", "Recall", "F1-Score"],
    "Score": [accuracy4, precision4, recall4, f14]
})
```

```
accuracy_matrix4
```

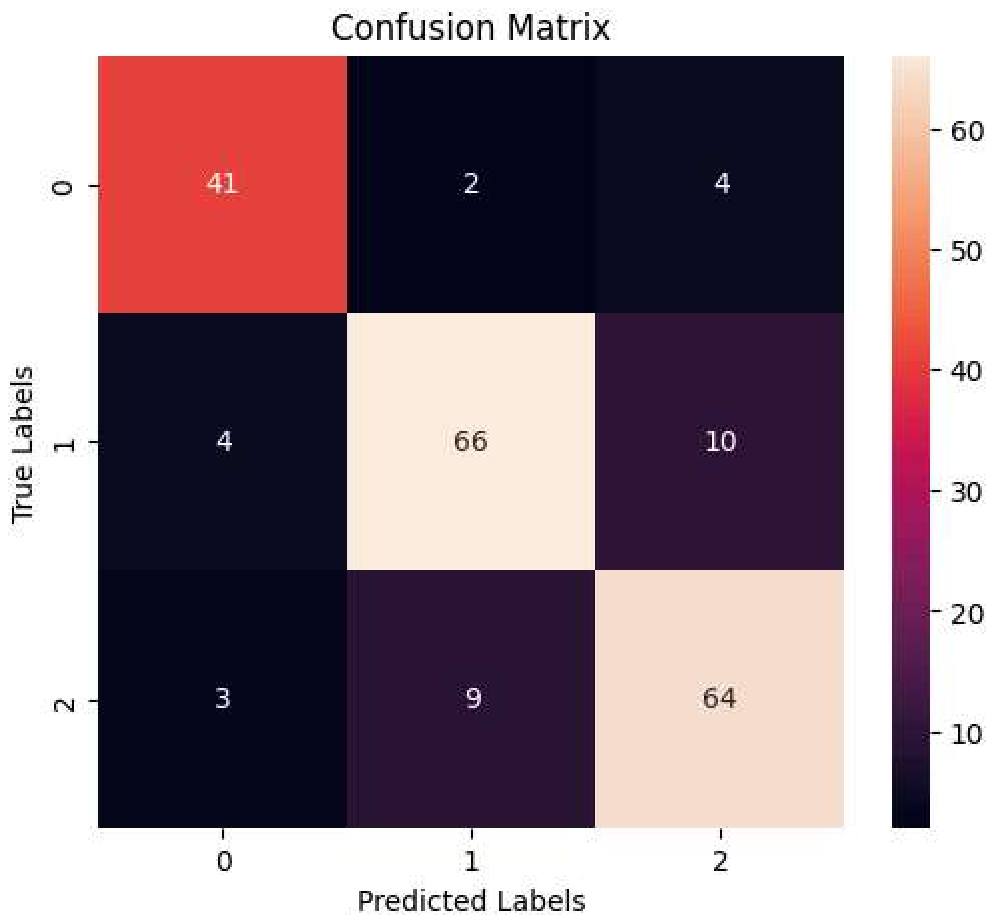
	Metric	Score
0	Accuracy	0.842365
1	Precision	0.842740
2	Recall	0.842365
3	F1-Score	0.842357

Compute confusion matrix

```
conf_matrix4 = confusion_matrix(y_test, test_predictions)
conf_matrix4
```

```
array([[41,  2,  4],
       [ 4, 66, 10],
       [ 3,  9, 64]])
```

```
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix4, annot=True)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```



Choose base models among AdaBoostClassifier GradientBoostingClassifier and XGBClassifier

```
models = {
    'Bagging':model1,
    'AdaBoost': model2,
    'GradientBoosting': model3,
```

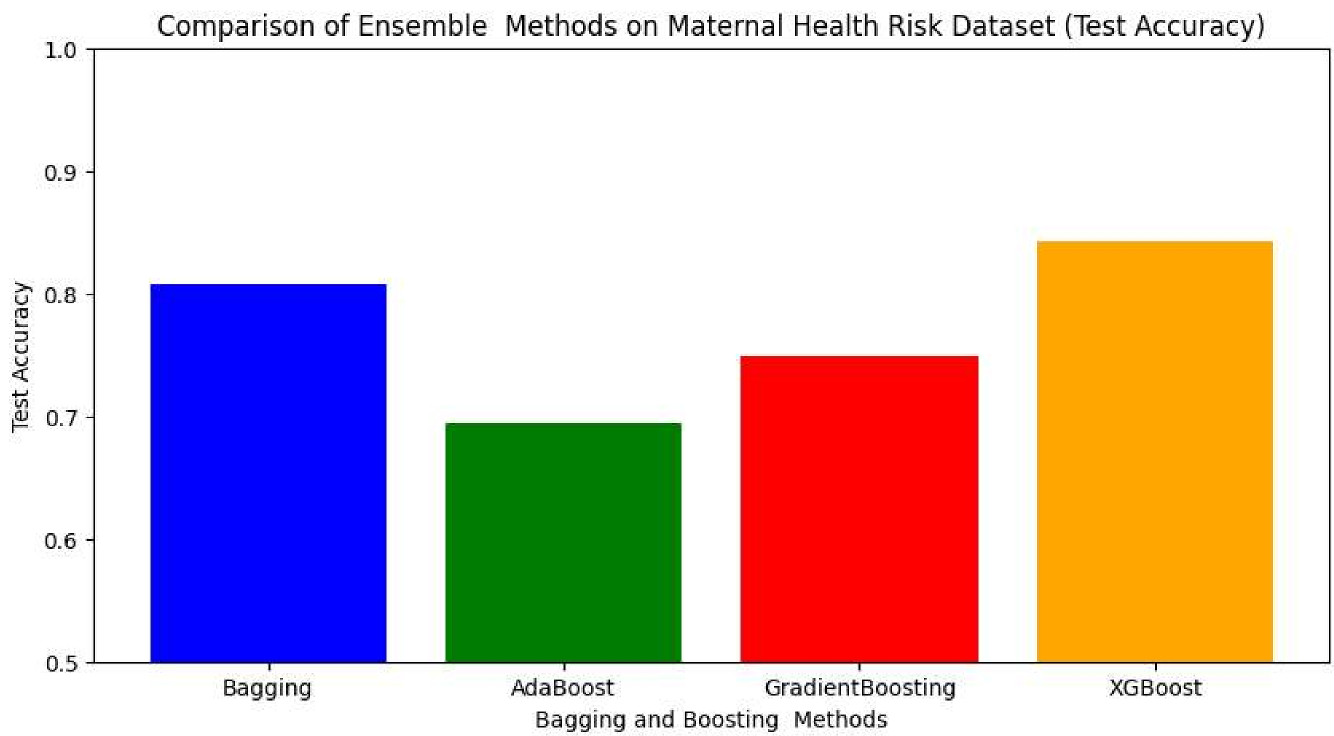
```
    'XGBoost': model4

}
```

```
test_accuracies = {}
```

```
for name, model in models.items():
    model.fit(X_train, y_train)
    test_predictions = model.predict(X_test)
    test_accuracies[name] = accuracy_score(y_test, test_predictions)
```

```
plt.figure(figsize=(10, 5))
plt.bar(test_accuracies.keys(), test_accuracies.values(), color=['blue', 'green', 'red','ora
plt.xlabel('Bagging and Boosting  Methods')
plt.ylabel('Test Accuracy')
plt.title('Comparison of Ensemble  Methods on Maternal Health Risk Dataset (Test Accuracy)')
plt.ylim([0.5, 1.0])  # Adjusted for dataset accuracy range
plt.show()
```




```
best_model_name = max(test_accuracies, key=test_accuracies.get)
```

```
best_model_name
```

```
'XGBoost'
```

```
print("\nTest Accuracy:")
for model, acc in test_accuracies.items():
    print(f"{model}: {acc:.4f}")
```

```
Test Accuracy:
Bagging: 0.8079
AdaBoost: 0.6946
GradientBoosting: 0.7488
XGBoost: 0.8424
```

```
print(f"\n✅ Best Model: **{best_model_name}**")
```

```
✅ Best Model: **XGBoost**
```

```
best_model = models[best_model_name]
```

```
best_model.fit(X_train, y_train)
best_test_predictions = best_model.predict(X_test)
```

Start coding or generate with AI.

```
best_test_accuracy = accuracy_score(y_test, best_test_predictions)
```

```
print(f"🎯 Best Model Test Accuracy: {best_test_accuracy:.4f}")
```

```
🎯 Best Model Test Accuracy: 0.8424
```

4) Stacking Implementation:

```
from sklearn.ensemble import  StackingClassifier
```

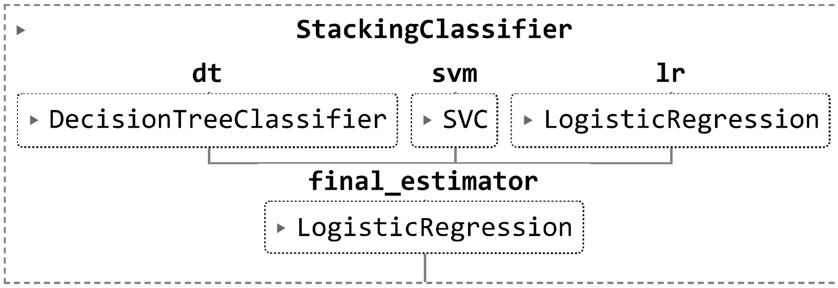
```
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.svm import SVC
```

```
from sklearn.linear_model import LogisticRegression
```

```
model5=StackingClassifier(
        estimators=[('dt', DecisionTreeClassifier()), ('svm', SVC(probability=True)), ('lr',
        final_estimator=LogisticRegression()
    )
```

```
model5.fit(X_train, y_train)
```




```
test_predictions = model5.predict(X_test)
test_predictions
```

```
array([2, 0, 0, 1, 1, 2, 2, 2, 2, 1, 0, 2, 2, 0, 2, 2, 0, 1, 1, 2, 2, 2,
       0, 0, 0, 2, 0, 1, 1, 1, 2, 0, 1, 2, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0,
       1, 2, 0, 2, 1, 1, 2, 1, 1, 2, 2, 2, 0, 1, 1, 2, 1, 2, 2, 1, 1, 0,
       2, 0, 2, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 2, 1, 2, 1, 1, 2, 1, 1, 2,
       0, 2, 2, 0, 1, 0, 2, 1, 2, 1, 0, 2, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2,
       2, 2, 1, 0, 2, 1, 1, 0, 1, 0, 2, 1, 1, 2, 0, 0, 2, 2, 2, 1, 1, 1,
       2, 2, 1, 1, 0, 2, 1, 2, 0, 2, 0, 0, 1, 1, 0, 1, 1, 2, 1, 2, 1, 2,
       0, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 2, 2, 1, 1, 0, 1, 2, 1, 2, 0,
       2, 1, 1, 1, 0, 2, 1, 2, 0, 1, 0, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 0,
       1, 2, 2, 2, 0])
```

```
accuracy5=accuracy_score(y_test, test_predictions)
accuracy5
```

```
0.8226600985221675
```

```
precision5=precision_score(y_test, test_predictions, average='weighted')
precision5
```

```
0.8252832151323424
```

```
recall5=recall_score(y_test, test_predictions, average='weighted')
recall5
```

```
0.8226600985221675
```

```
f15=f1_score(y_test, test_predictions, average='weighted')
f15
```

```
0.8225547363607623
```

Show accuracy matrix as a Dataframe

```
accuracy_matrix = pd.DataFrame({
    "Metric": ["Accuracy", "Precision", "Recall", "F1-Score"],
    "Score": [accuracy5, precision5, recall5, f15]
})
```

```
accuracy_matrix
```

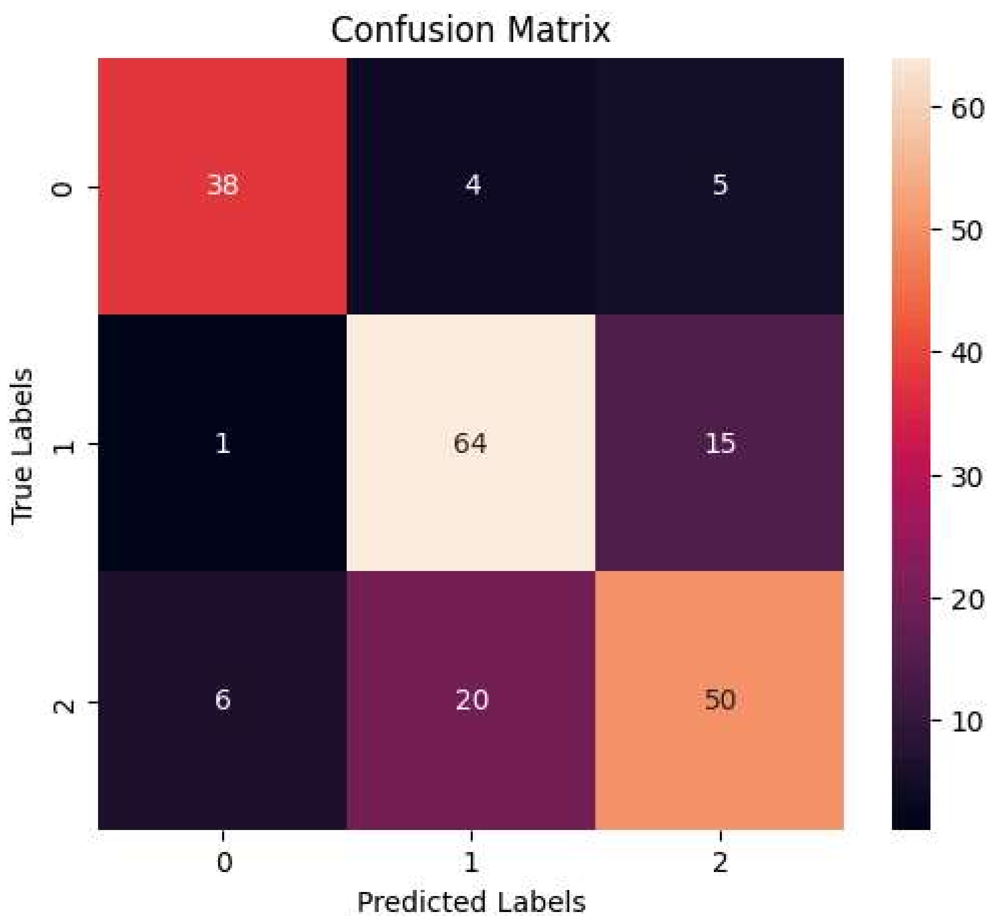
	Metric	Score
0	Accuracy	0.822660
1	Precision	0.825283
2	Recall	0.822660
3	F1-Score	0.822555

Compute confusion matrix

```
conf_matrix5 = confusion_matrix(y_test, test_predictions)
conf_matrix5
```

```
array([[41,  1,  5],
       [ 5, 62, 13],
       [ 3,  9, 64]])
```

```
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```



5) Compare the performance of Bagging, Boosting, and Stacking

```
performance={'Models':['Bagging','Boosting(XGB (best))','Stacking'],
              'Accuracy':[accuracy1,accuracy4,accuracy5],
               'Precision':[precision1,precision4,precision5],
              'Recall':[recall1,recall4,recall5],
              'F1-Score':[f11,f14,f15]


              }
```

```
pd.DataFrame(performance)
```

	Models	Accuracy	Precision	Recall	F1-Score
0	Bagging	0.807882	0.814527	0.807882	0.808429
1	Boosting(XGB (best))	0.842365	0.842740	0.842365	0.842357
2	Stacking	0.822660	0.825283	0.822660	0.822555

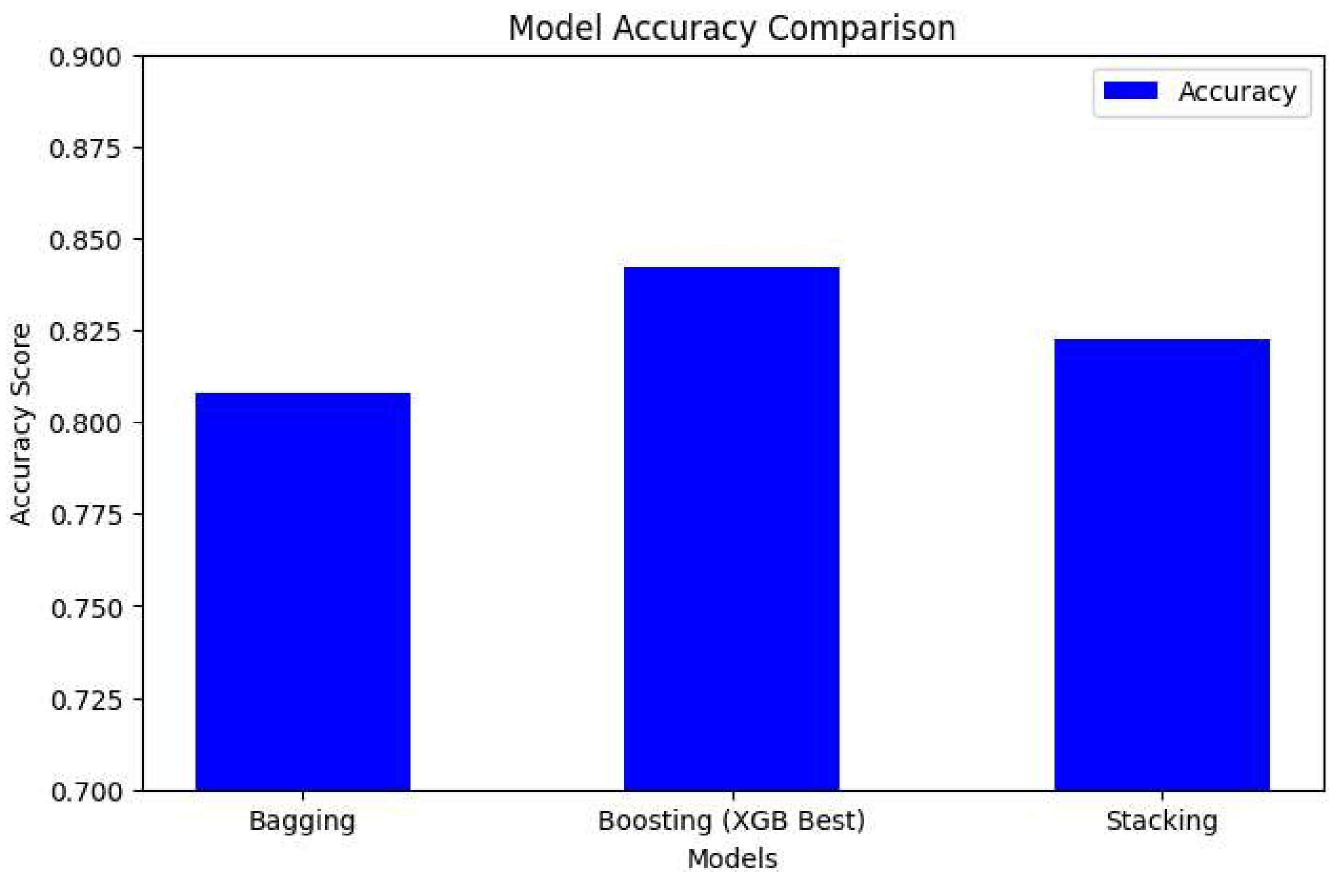
```
performance = {
    'Models': ['Bagging', 'Boosting (XGB Best)', 'Stacking'],
    'Accuracy': [accuracy1, accuracy4, accuracy5]  # Only accuracy values
}
```

```
models = performance['Models']
accuracy_values = performance['Accuracy']
```

```
import numpy as np
```

```
x = np.arange(len(models))
width = 0.5
```

```
fig, ax = plt.subplots(figsize=(8, 5))
ax.bar(x, accuracy_values, width, color='blue', label='Accuracy')
ax.set_xlabel('Models')
ax.set_ylabel('Accuracy Score')
ax.set_title('Model Accuracy Comparison')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()
plt.ylim(0.7, 0.9)
plt.show()
```



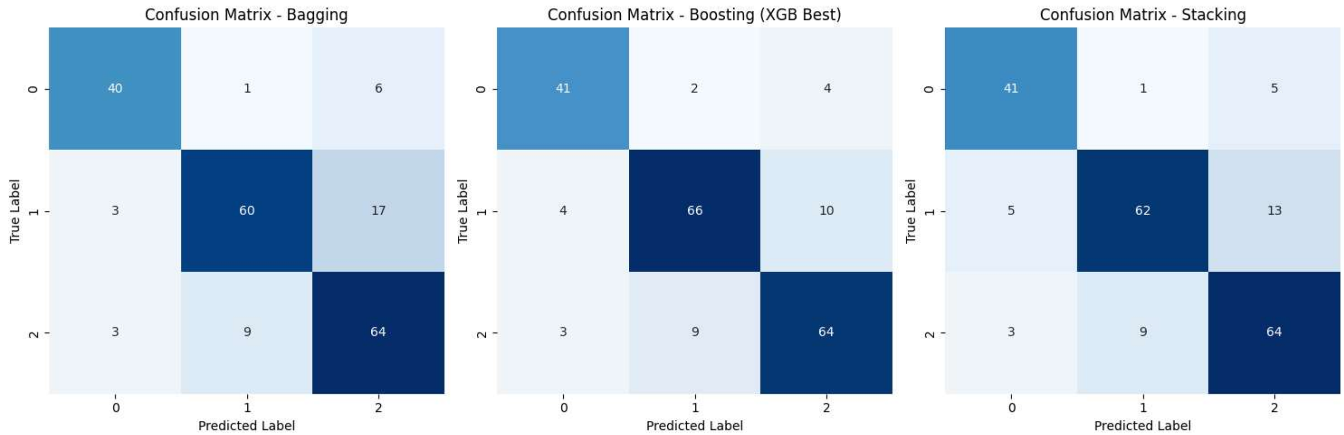

```
model_names = ['Bagging', 'Boosting (XGB Best)', 'Stacking']
```

```
conf_matrices = [conf_matrix1, conf_matrix4, conf_matrix5]
```

```
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
for i, ax in enumerate(axes):
    sns.heatmap(conf_matrices[i], annot=True, fmt="d", cmap="Blues", cbar=False, ax=ax)
    ax.set_title(f'Confusion Matrix - {model_names[i]}')
    ax.set_xlabel('Predicted Label')
    ax.set_ylabel('True Label')

plt.tight_layout()

plt.show()
```



6) Select the Model (XGB Classifier) Analysis

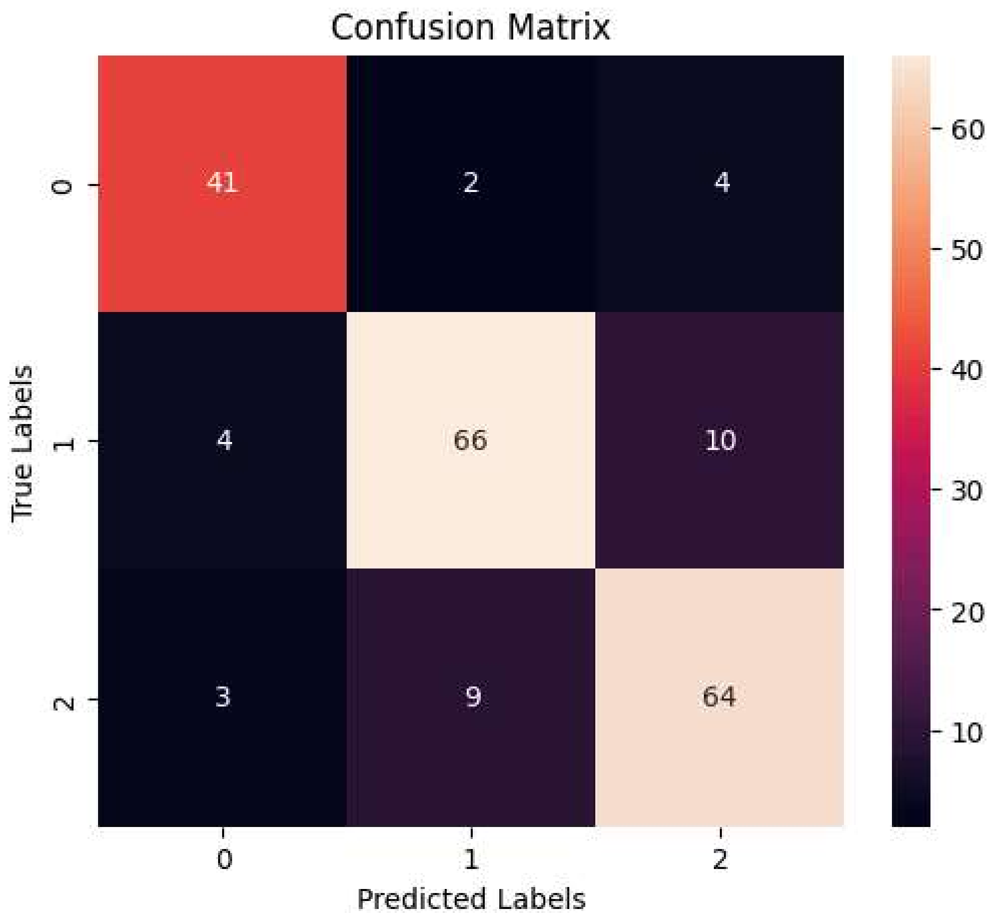
```
accuracy_matrix4 = pd.DataFrame({
    "Metric": ["Accuracy", "Precision", "Recall", "F1-Score"],
    "Score": [accuracy4, precision4, recall4, f14]
})
```

```
pd.DataFrame(accuracy_matrix4)
```

	Metric	Score
0	Accuracy	0.842365
1	Precision	0.842740
2	Recall	0.842365
3	F1-Score	0.842357

```
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix4, annot=True)
plt.xlabel("Predicted Labels")
```

```
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```



XGB Classifier (Best) with Validation

```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=
```

```
final_model=XGBClassifier(n_estimators=50, eval_metric='mlogloss', random_state=42)
```

```
final_model.fit(X_train, y_train)
```

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='mlogloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=50,
              n_jobs=None, num_parallel_tree=None, objective='multi:softprob', ...)
```

```
val_predictions = final_model.predict(X_val)
test_predictions = final_model.predict(X_test)
```

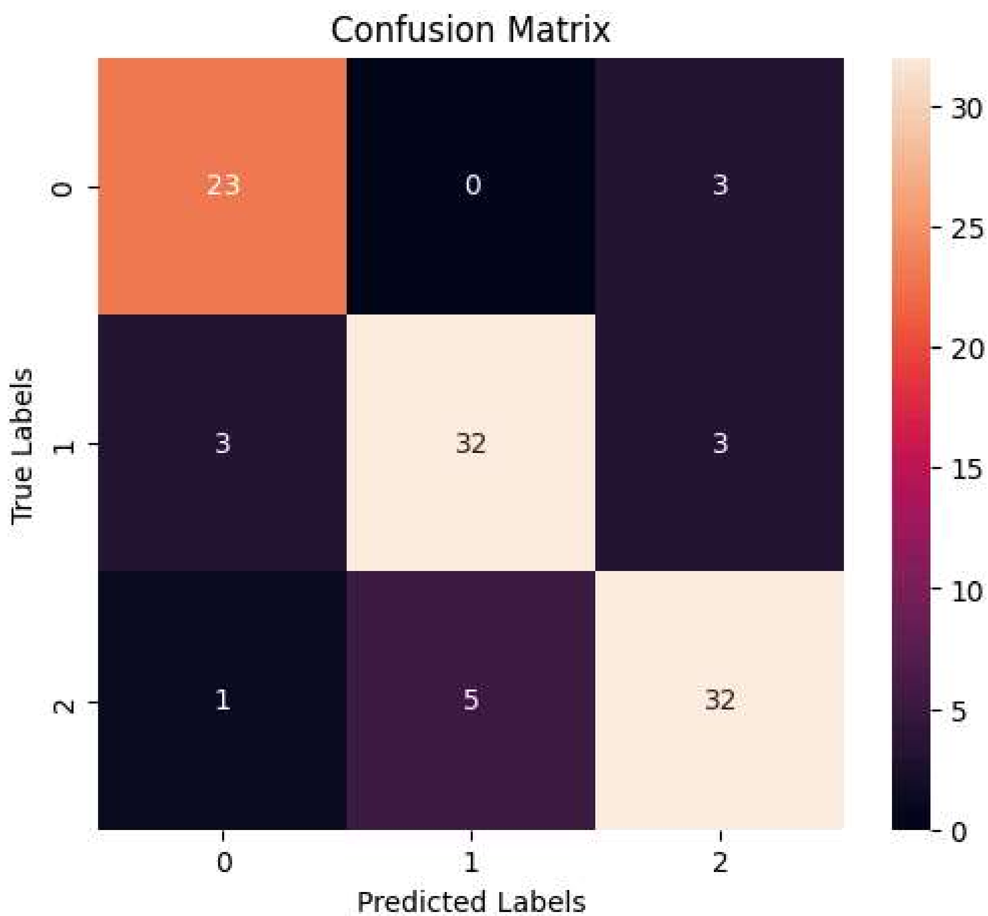
```
accuracy_score(y_test, test_predictions)
```

```
0.8529411764705882
```

```
conf_matrix = confusion_matrix(y_test, test_predictions)
conf_matrix
```

```
array([[23,  0,  3],
       [ 3, 32,  3],
       [ 1,  5, 32]])
```

```
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```




```
final_evaluation={'XGB':[accuracy4],
           'XGB(Validation)':[accuracy_score(y_test, test_predictions)]}
```

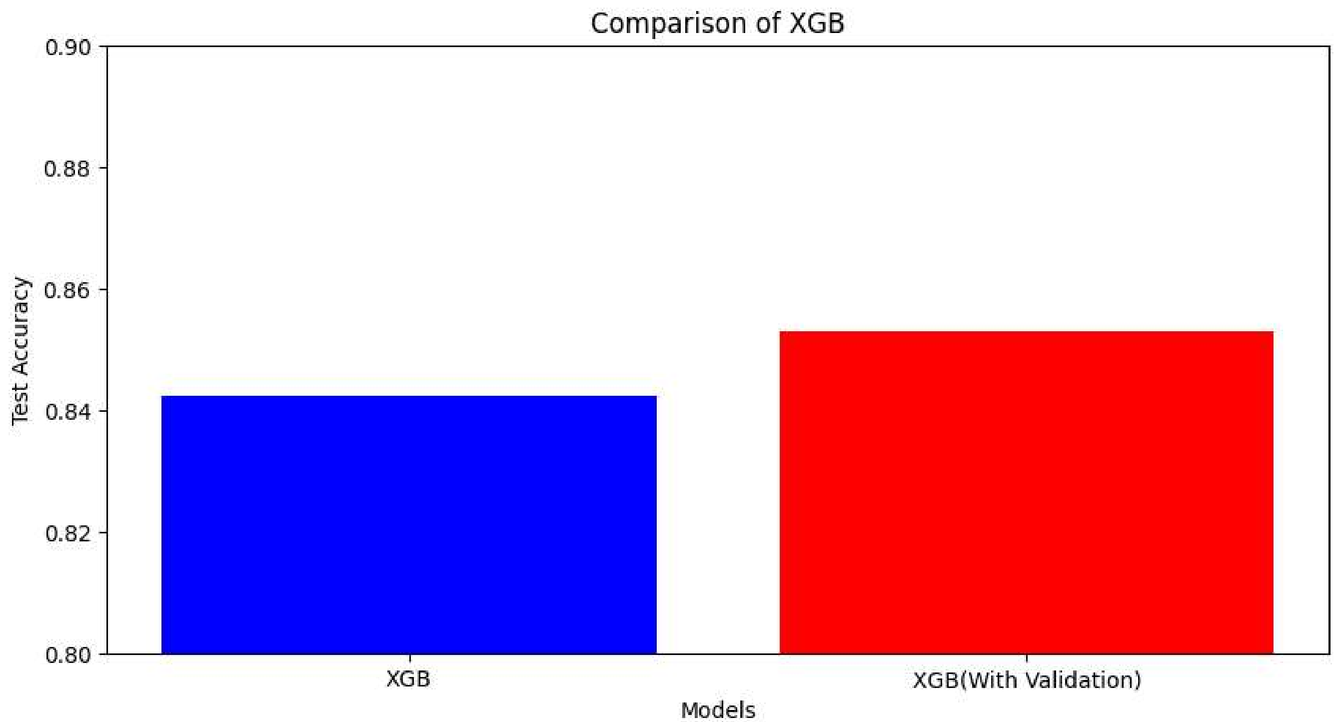
```
final_evaluation
```

{'XGB': [0.8423645320197044], 'XGB(Validation)': [0.8529411764705882]}

```
final_evaluation = {
    'XGB': [0.8423645320197044],
    'XGB(With Validation)': [0.8529411764705882]
}
```

```
models = list(final_evaluation.keys())
accuracies = [val[0] for val in final_evaluation.values()]
```

```
plt.figure(figsize=(10, 5))
plt.bar(models, accuracies, color=['blue', 'red'])
plt.xlabel('Models')
plt.ylabel('Test Accuracy')
plt.title('Comparison of XGB')
plt.ylim([0.8, 0.9])
plt.show()
```




```
pd.DataFrame(final_evaluation )
```

	XGB	XGB(With Validation)
0	0.842365	0.852941

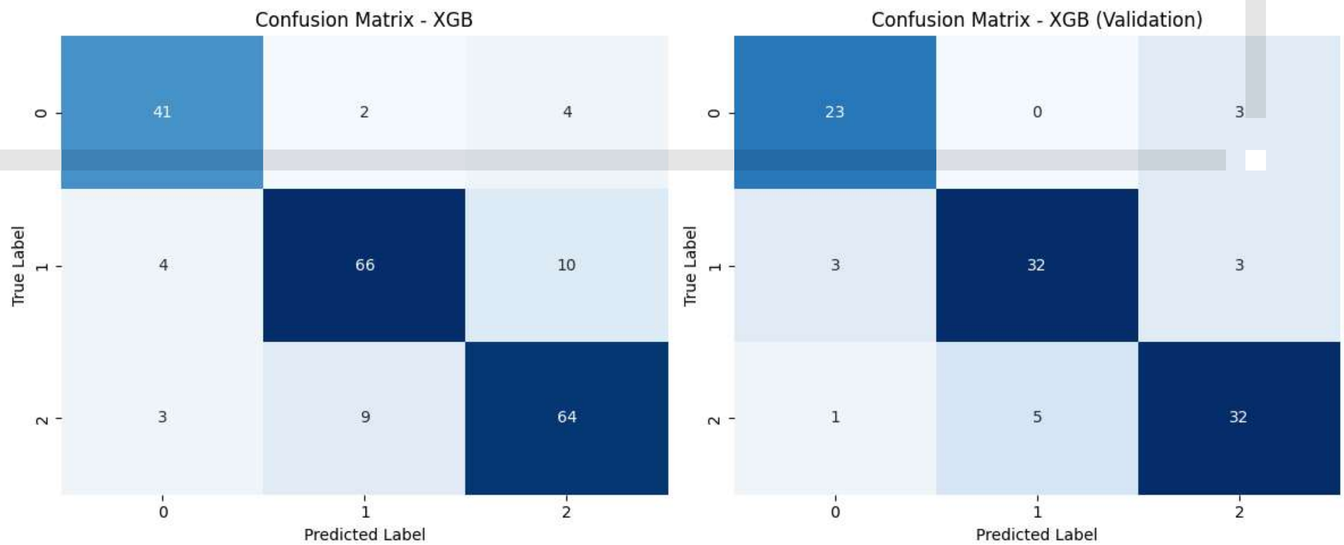
```
model_names = ['XGB', 'XGB (Validation)']
```

```
conf_matrices = [conf_matrix4, conf_matrix]
```

```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
for i, ax in enumerate(axes):
    sns.heatmap(conf_matrices[i], annot=True, fmt="d", cmap="Blues", cbar=False, ax=ax
    ax.set_title(f'Confusion Matrix - {model_names[i]}')
```

```
    ax.set_xlabel('Predicted Label')
    ax.set_ylabel('True Label')

plt.tight_layout()
```



End of Assignment

```
Start coding or generate with AI.
```