Name: Angela Hartono
Student ID: 2602059582

Google Collab Link
https://colab.research.google.com/drive/1-VRMHSp7DF7Z18uVE5iBiDsBHQBw5NKH?usp=sharing

First, import the necessary libraries. Next, obtain the dataset from the Google Drive link, or import the dataset into Google Colab so that it can be accessed and used.

```
[216] # Import the libraries
      import pandas as pd
      import numpy as np
      from datetime import datetime
```

```
[217] # Get the dataset from the given link
      !wget "https://drive.google.com/uc?export=download&id=1abSLIIPVkgmZW8Cx3vohIlHk16gJmuvO" -O healthcare_dataset.csv

      --2023-12-03 17:44:46--  https://drive.google.com/uc?export=download&id=1abSLIIPVkgmZW8Cx3vohIlHk16gJmuvO
      Resolving drive.google.com (drive.google.com)... 172.217.0.78, 2607:f8b0:4025:815::200e
      Connecting to drive.google.com (drive.google.com)|172.217.0.78|:443... connected.
      HTTP request sent, awaiting response... 303 See Other
      Location: https://doc-10-5o-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7mbp1/4erf1oh69v5haq5qpdpji92nmbi91ovn/1701625425000/01442500710412781851/*/1abSLIIPVkgmZW8Cx3vohIlHk16gJmuvO?
      Warning: wildcards not supported in HTTP.
      --2023-12-03 17:44:47--  https://doc-10-5o-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7mbp1/4erf1oh69v5haq5qpdpji92nmbi91ovn/1701625425000/01442500710412781851/*/1abSLIIPVkgmZW8Cx3v
      Resolving doc-10-5o-docs.googleusercontent.com (doc-10-5o-docs.googleusercontent.com)... 172.217.12.1, 2607:f8b0:4025:815::2001
      Connecting to doc-10-5o-docs.googleusercontent.com (doc-10-5o-docs.googleusercontent.com)|172.217.12.1|:443... connected.
      HTTP request sent, awaiting response... 200 OK
      Length: 1512362 (1.4M) [text/csv]
      Saving to: 'healthcare_dataset.csv'

      healthcare_dataset. 100%[===================>]   1.44M  --.-KB/s    in 0.08s

      2023-12-03 17:44:47 (17.3 MB/s) - 'healthcare_dataset.csv' saved [1512362/1512362]
```

```
# Get the dataset and load it into 'df'
df=pd.read_csv('healthcare_dataset.csv')
```

Then, I want to explore the data by displaying the top 4 rows, checking the number of rows and columns, and displaying the column names in the dataset.

Data Exploration

```
# Display the first few rows of the dataset
df.head()
```

	Name	Age	Gender	Blood Type	Medical Condition	Date of Admission	Doctor	Hospital	Insurance Provider	Billing Amount	Room Number	Admission Type	Discharge Date	Medication	Test Results
0	Tiffany Ramirez	81	Female	O-	Diabetes	2022-11-17	Patrick Parker	Wallace-Hamilton	Medicare	37490.983364	146	Elective	2022-12-01	Aspirin	Inconclusive
1	Ruben Burns	35	Male	O+	Asthma	2023-06-01	Diane Jackson	Burke, Griffin and Cooper	UnitedHealthcare	47304.064845	404	Emergency	2023-06-15	Lipitor	Normal
2	Chad Byrd	61	Male	B-	Obesity	2019-01-09	Paul Baker	Walton LLC	Medicare	36874.896997	292	Emergency	2019-02-08	Lipitor	Normal
3	Antonio Frederick	49	Male	B-	Asthma	2020-05-02	Brian Chandler	Garcia Ltd	Medicare	23303.322092	480	Urgent	2020-05-03	Penicillin	Abnormal
4	Mrs. Brandy Flowers	51	Male	O-	Arthritis	2021-07-09	Dustin Griffin	Jones, Brown and Murray	UnitedHealthcare	18086.344184	477	Urgent	2021-08-02	Paracetamol	Normal

```
[220] # Check the number of rows and columns of the dataset
      df.shape

      (10000, 15)
```

```
[221] # Display the column names in the dataset
      df.columns

      Index(['Name', 'Age', 'Gender', 'Blood Type', 'Medical Condition',
             'Date of Admission', 'Doctor', 'Hospital', 'Insurance Provider',
             'Billing Amount', 'Room Number', 'Admission Type', 'Discharge Date',
             'Medication', 'Test Results'],
            dtype='object')
```

For a more comprehensive understanding of the dataset, I used df.info and df.info() to display the information about the dataframe. After that, I calculated the count of each value in the test result column as it represents the target variable I'll be using.

```
[63] # Display concise summary information about the DataFrame
df.info
```

```
<bound method DataFrame.info of                          Name  Age  Gender Blood Type Medical Condition  \
0         Tiffany Ramirez   81  Female        O-          Diabetes
1             Ruben Burns   35    Male        O+            Asthma
2               Chad Byrd   61    Male        B-           Obesity
3       Antonio Frederick   49    Male        B-            Asthma
4      Mrs. Brandy Flowers   51    Male        O-         Arthritis
...                   ...  ...     ...       ...               ...
9995           James Hood   83    Male        A+           Obesity
9996      Stephanie Evans   47  Female       AB+         Arthritis
9997  Christopher Martinez   54    Male        B-         Arthritis
9998          Amanda Duke   84    Male        A+         Arthritis
9999             Eric King   20    Male        B-         Arthritis

     Date of Admission            Doctor                    Hospital  \
0           2022-11-17    Patrick Parker            Wallace-Hamilton
1           2023-06-01     Diane Jackson  Burke, Griffin and Cooper
2           2019-01-09        Paul Baker                 Walton LLC
3           2020-05-02    Brian Chandler                 Garcia Ltd
4           2021-07-09    Dustin Griffin     Jones, Brown and Murray
...                ...               ...                         ...
9995        2022-07-29      Samuel Moody    Wood, Martin and Simmons
9996        2022-01-06 Christopher Yates                Nash-Krueger
9997        2022-07-01  Robert Nicholson             Larson and Sons
9998        2020-02-06       Jamie Lewis                Wilson-Lyons
9999        2023-03-22       Tasha Avila  Torres, Young and Stewart

     Insurance Provider  Billing Amount  Room Number Admission Type  \
0              Medicare    37490.983364          146       Elective
1      UnitedHealthcare    47304.064845          404      Emergency
2              Medicare    36874.896997          292      Emergency
3              Medicare    23303.322092          480         Urgent
4      UnitedHealthcare    18086.344184          477         Urgent
...                 ...             ...          ...            ...
9995   UnitedHealthcare    39606.840083          110       Elective
9996         Blue Cross     5995.717488          244      Emergency
9997         Blue Cross    49559.202905          312       Elective
9998   UnitedHealthcare    25236.344761          420         Urgent
9999              Aetna    37223.965865          290      Emergency

     Discharge Date   Medication  Test Results
0        2022-12-01      Aspirin  Inconclusive
1        2023-06-15      Lipitor        Normal
2        2019-02-08      Lipitor        Normal
3        2020-05-03   Penicillin      Abnormal
4        2021-08-02  Paracetamol        Normal
...             ...          ...           ...
9995     2022-08-02    Ibuprofen      Abnormal
9996     2022-01-29    Ibuprofen        Normal
9997     2022-07-15    Ibuprofen        Normal
9998     2020-02-26   Penicillin        Normal
9999     2023-04-15   Penicillin      Abnormal

[10000 rows x 15 columns]>
```

```
[222] # Information about the columns, their data types, and missing values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Name                10000 non-null  object
 1   Age                 10000 non-null  int64
 2   Gender              10000 non-null  object
 3   Blood Type          10000 non-null  object
 4   Medical Condition   10000 non-null  object
 5   Date of Admission   10000 non-null  object
 6   Doctor              10000 non-null  object
 7   Hospital            10000 non-null  object
 8   Insurance Provider  10000 non-null  object
 9   Billing Amount      10000 non-null  float64
 10  Room Number         10000 non-null  int64
 11  Admission Type      10000 non-null  object
 12  Discharge Date      10000 non-null  object
 13  Medication          10000 non-null  object
 14  Test Results        10000 non-null  object
dtypes: float64(1), int64(2), object(12)
memory usage: 1.1+ MB
```

Dapat dilihat bahwa tidak ada missing values

```
[223] # Count the occurrences of each value in the 'Test Results' column, which is the target variable
df['Test Results'].value_counts()
```

```
Abnormal        3456
Inconclusive    3277
Normal          3267
Name: Test Results, dtype: int64
```

I performed feature extraction with the aim of optimizing and enhancing the machine learning process. This involved adding or modifying a feature to assist the machine learning model in better understanding the data. Specifically, I created a new feature called 'Days Gap' derived from the 'Date of Admission' and 'Discharge Date' features. Subsequently, I dropped these two features ('Date of Admission' and 'Discharge Date') as they did not impact the target variable, 'Test Result'. The addition of the 'Days Gap' feature was motivated by its relevance to the target variable because generally, when discussing someone's test results, the duration of their hospital stay is often longer for more severe illnesses.

Feature Extraction

```
[ ] # Convert date columns to datetime format if needed
df['Date of Admission'] = pd.to_datetime(df['Date of Admission'])
df['Discharge Date'] = pd.to_datetime(df['Discharge Date'])

# Calculate the difference in days between the two columns
df['Days Gap'] = (df['Discharge Date'] - df['Date of Admission']).dt.days

# Display data with the new 'selisih_hari' column
df_1 = df.drop(['Date of Admission', 'Discharge Date'], axis=1)

df_1.head(5)
```

	Name	Age	Gender	Blood Type	Medical Condition	Doctor	Hospital	Insurance Provider	Billing Amount	Room Number	Admission Type	Medication	Test Results	Days Gap
0	Tiffany Ramirez	81	Female	O-	Diabetes	Patrick Parker	Wallace-Hamilton	Medicare	37490.983364	146	Elective	Aspirin	Inconclusive	14
1	Ruben Burns	35	Male	O+	Asthma	Diane Jackson	Burke, Griffin and Cooper	UnitedHealthcare	47304.064845	404	Emergency	Lipitor	Normal	14
2	Chad Byrd	61	Male	B-	Obesity	Paul Baker	Walton LLC	Medicare	36874.896997	292	Emergency	Lipitor	Normal	30
3	Antonio Frederick	49	Male	B-	Asthma	Brian Chandler	Garcia Ltd	Medicare	23303.322092	480	Urgent	Penicillin	Abnormal	1
4	Mrs. Brandy Flowers	51	Male	O-	Arthritis	Dustin Griffin	Jones, Brown and Murray	UnitedHealthcare	18086.344184	477	Urgent	Paracetamol	Normal	24

I checked the correlation between variables to know if the correlation was strong or not. When the correlation is strong where it is close to 1 or -1, it's usually beneficial to retain those columns in the dataset. However, in this case, the correlation among variables is weak. Despite considering dropping columns due to their limited correlation, none were removed because the correlations were low and there were few features. Consequently, no actions were taken in this regard.

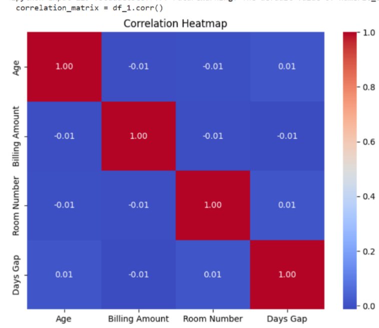
Correlation

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate correlation matrix
correlation_matrix = df_1.corr()

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```

```
<ipython-input-225-f38aaaec3e8f>:7: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  correlation_matrix = df_1.corr()
```



No correlation among the numeric data

Next, I performed data preprocessing by separating the target variable (y) from the other features (X). Then, I split the data into training and testing sets for both the X and y variables.

Data Preprocessing

Separate Target

```python
X = df_1.drop(['Test Results'], axis=1) #the other features beside target
y = df_1['Test Results'] #target
```

Split Train and Test

```python
[16] 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=123)
```

Next, I checked for missing values and outliers. It appears that there are no missing values or outliers in the dataset, so there's no need for further handling as the data is clean.

Check Missing Values in Data Training and Testing

```
[17] # Check missing values in training data
     missing_values_train = X_train.isnull().sum()

     # Check missing values in testing data
     missing_values_test = X_test.isnull().sum()

     print("Missing values in training data:")
     print(missing_values_train)

     print("\nMissing values in testing data:")
     print(missing_values_test)
```

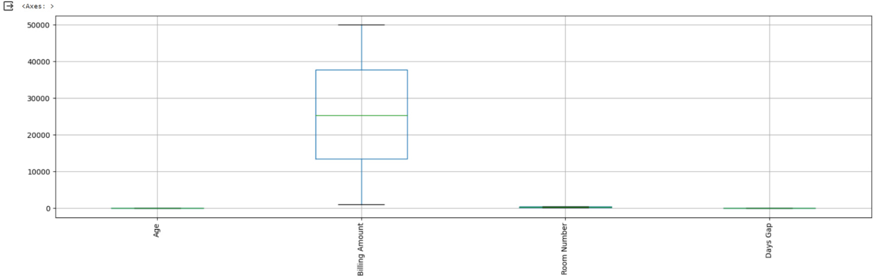
```
Missing values in training data:
Name                  0
Age                   0
Gender                0
Blood Type            0
Medical Condition     0
Doctor                0
Hospital              0
Insurance Provider    0
Billing Amount        0
Room Number           0
Admission Type        0
Medication            0
Days Gap              0
dtype: int64

Missing values in testing data:
Name                  0
Age                   0
Gender                0
Blood Type            0
Medical Condition     0
Doctor                0
Hospital              0
Insurance Provider    0
Billing Amount        0
Room Number           0
Admission Type        0
Medication            0
Days Gap              0
dtype: int64
```

Check Outliers in Data Training and Testing using Boxplot

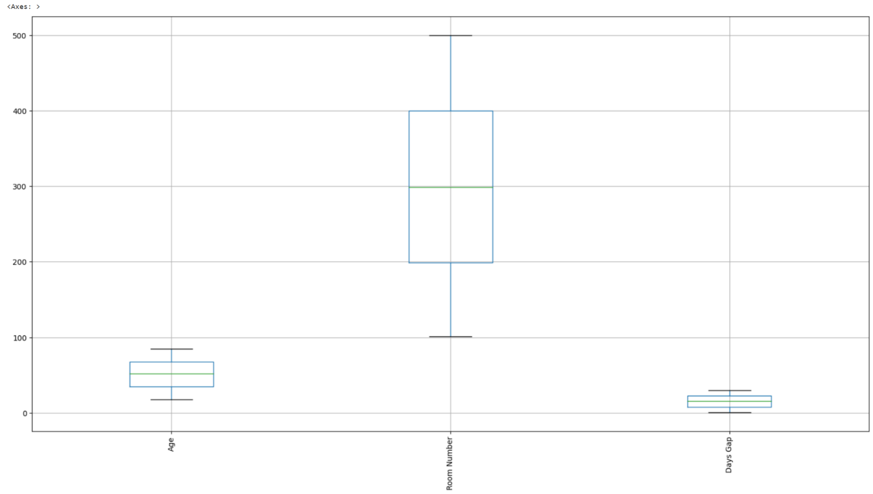
```
df.boxplot(figsize=(20,5), rot=90)
```

```
<Axes: >
```




```
[19] df.drop('Billing Amount', axis = 1).boxplot(figsize=(20,10), rot=90)
```

```
<Axes: >
```



Afterward, I performed encoding to process all numeric data. I used label encoding for the name column and one-hot encoding for other categorical columns. Additionally, I applied standard scaling to normalize the numeric data in machine learning. This ensures that all numeric features have a mean of 0 and a standard deviation of 1. It helps maintain consistency across different feature scales, preventing certain features from dominating the model due to their larger magnitudes. This process aids algorithms using distance-based calculations or gradient-based optimization for more effective convergence and better outcomes.

Encoding

Label Encoding

```
from sklearn.preprocessing import LabelEncoder

# Convert all values in the 'Name' column to string data type
X_train['Name'] = X_train['Name'].astype(str)
X_test['Name'] = X_test['Name'].astype(str)

# Create a LabelEncoder for the 'Name' column using combined data from training and testing
label_encoder = LabelEncoder()

# Combine training and testing data
combined_data = pd.concat([X_train['Name'], X_test['Name']], axis=0)
label_encoder.fit(combined_data)

# Encode the training and testing data for the 'Name' column using the same encoder
X_train['Name'] = label_encoder.transform(X_train['Name'])
X_test['Name'] = label_encoder.transform(X_test['Name'])
```

```
# Display the first few rows of the encoded data
X_train.head()
```

	Name	Age	Gender	Blood Type	Medical Condition	Doctor	Hospital	Insurance Provider	Billing Amount	Room Number	Admission Type	Medication	Days Gap
9216	5415	44	Female	B-	Asthma	Nancy Davidson	Atkinson-Johnson	Aetna	28900.615874	292	Elective	Penicillin	8
7324	3799	30	Male	O-	Cancer	Mary Wilson	Schultz-Jones	Medicare	40461.419916	346	Urgent	Lipitor	8
918	5521	74	Male	B+	Hypertension	Ronald Taylor	Perez, Ray and Combs	Blue Cross	2936.565609	172	Emergency	Ibuprofen	18
5902	3612	83	Female	A+	Hypertension	Tyler Foster	Williamson-Hernandez	Aetna	22777.106392	477	Elective	Lipitor	9
2807	3302	46	Female	O+	Diabetes	James Brandt	Hughes LLC	UnitedHealthcare	11962.537331	227	Elective	Lipitor	12

```
[73] # Display the first few rows of the encoded data
X_test.head()
```

	Name	Age	Gender	Blood Type	Medical Condition	Doctor	Hospital	Insurance Provider	Billing Amount	Room Number	Admission Type	Medication	Days Gap
2656	6038	32	Female	A+	Cancer	Stephen Cook	Monroe, Allen and Welch	Aetna	49243.317363	155	Urgent	Paracetamol	21
445	2247	54	Male	AB+	Obesity	Jennifer Carlson	Cook, White and Davis	Blue Cross	1926.881278	485	Urgent	Penicillin	24
9505	3279	51	Male	O-	Hypertension	Amy Daniels	Alvarez-Taylor	Blue Cross	28162.980371	395	Elective	Paracetamol	13
332	912	83	Female	O+	Hypertension	Cory Fletcher	Hernandez-Mendoza	UnitedHealthcare	37744.299585	471	Elective	Aspirin	1
4168	1150	85	Male	O-	Obesity	Taylor Gardner	Burke-Mendoza	Cigna	1889.902251	201	Elective	Paracetamol	16

One Hot Encoding

```
[20] # List of columns to be one-hot encoded
one_hot_cols = ['Gender', 'Blood Type', 'Medical Condition', 'Doctor', 'Hospital', 'Insurance Provider', 'Admission Type', 'Medication']

# Combining training and testing data
combined_data = pd.concat([X_train, X_test], axis=0)

# Performing one-hot encoding on the combined data
combined_data_encoded = pd.get_dummies(combined_data, columns=one_hot_cols)

# Splitting the encoded data back into training and testing datasets
X_train_encoded = combined_data_encoded.loc[X_train.index, :]
X_test_encoded = combined_data_encoded.loc[X_test.index, :]
```

```
# Display the first few rows of the encoded data
X_train_encoded.head()
```

	Name	Age	Billing Amount	Room Number	Days Gap	Gender_Female	Gender_Male	Blood Type_A+	Blood Type_A-	Blood Type_AB+	...	Insurance Provider_Medicare	Insurance Provider_UnitedHealthcare	Admission Type_Elective	Admission Type_Emergency	Admission Type_Urgent	Medication_Aspirin	Medication_Ibuprofen
9216	5872	44	28900.615874	292	8	1	0	0	0	0	...	0	0	1	0	0	0	0
7324	4417	30	40461.419916	346	8	0	1	0	0	0	...	1	0	0	0	1	0	0
918	5968	74	2936.565609	172	18	0	1	0	0	0	...	0	0	0	1	0	0	1
5902	4249	83	22777.106392	477	9	1	0	1	0	0	...	0	0	1	0	0	0	0
2807	3970	46	11962.537331	227	12	1	0	0	0	0	...	0	1	1	0	0	0	0

5 rows × 18089 columns

```
[45] # Display the first few rows of the encoded data
X_test_encoded.head()
```

	Name	Age	Billing Amount	Room Number	Days Gap	Gender_Female	Gender_Male	Blood Type_A+	Blood Type_A-	Blood Type_AB+	...	Insurance Provider_Medicare	Insurance Provider_UnitedHealthcare	Admission Type_Elective	Admission Type_Emergency	Admission Type_Urgent	Medication_Aspirin	Medication_Ibuprofen
2656	6432	32	49243.317363	155	21	1	0	1	0	0	...	0	0	0	0	1	0	0
445	302	54	1926.881278	485	24	0	1	0	0	1	...	0	0	0	0	1	0	0
9505	395	51	28162.980371	395	13	0	1	0	0	0	...	0	0	1	0	0	0	0
332	1819	83	37744.299585	471	1	1	0	0	0	0	...	0	1	1	0	0	1	0
4168	2032	85	1889.902251	201	16	0	1	0	0	0	...	0	0	1	0	0	0	0

5 rows × 18089 columns

Standar Scaler for Numerical Data

```
[46] from sklearn.preprocessing import StandardScaler

# List of numeric columns for scaling
numeric_cols = ['Age', 'Billing Amount', 'Room Number', 'Days Gap']

# Initializing the StandardScaler
scaler = StandardScaler()

# Scaling the training data
X_train_encoded[numeric_cols] = scaler.fit_transform(X_train_encoded[numeric_cols])

# Scaling the testing data
X_test_encoded[numeric_cols] = scaler.transform(X_test_encoded[numeric_cols])
```

Then, I performed a classification modeling utilizing an ensemble, starting with a random forest as part of the bagging model. Here, I aim to compare the random forest model in its default state with the tuned version. The first evaluation (by default), demonstrated moderate accuracy in identifying Abnormal cases but faced challenges in achieving overall accuracy across all categories. In the second evaluation (after tuning), while it improved in identifying Abnormal instances, it didn't do well with Inconclusive and Normal classes, resulting in low recall and F1-scores. Overall, there was a bit of progress in identifying Abnormal cases, but the model needs substantial improvement to work effectively across all categories.

The default accuracy of the random forest model is 0.33, and after tuning, the accuracy improved to 0.34. It can be concluded that the tuned random forest model is more accurate compared to the default one.

Random Forest

By Default

```
[23] from sklearn.ensemble import RandomForestClassifier
     from sklearn.model_selection import GridSearchCV

     # Initialize the Random Forest classifier with default parameters
     clf_default = RandomForestClassifier(random_state=123)

     # Fit the Random Forest classifier with encoded training data
     clf_default.fit(X_train_encoded, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=123)
```

```
[24] from sklearn.metrics import classification_report

     # Predict using the encoded testing data
     y_pred = clf_default.predict(X_test_encoded)

     # Evaluate the model before tuning
     print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

    Abnormal       0.35      0.49      0.41       691
Inconclusive       0.31      0.27      0.29       637
      Normal       0.30      0.22      0.25       672

    accuracy                           0.33      2000
   macro avg       0.32      0.33      0.32      2000
weighted avg       0.32      0.33      0.32      2000
```

Tunning

```
[25] from sklearn.ensemble import RandomForestClassifier
     from sklearn.model_selection import GridSearchCV

     # Define the parameters for GridSearchCV
     param_grid = {
         'n_estimators': [50, 150],
         'max_depth': [None, 5, 10],
         'min_samples_split': [1, 5],
         'min_samples_leaf': [1, 2]
     }

     # Initializing the Random Forest model
     clf = RandomForestClassifier(random_state=123)

     # Initialize GridSearchCV
     grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

     # Fit GridSearchCV with encoded training data
     grid_search.fit(X_train_encoded, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
60 fits failed out of a total of 120.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
60 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py", line 340, in fit
    self._validate_params()
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 600, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'min_samples_split' parameter of RandomForestClassifier must be an int in the range [2, inf) or a float in the range (0.0, 1.0]. Got 1 instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: UserWarning: One or more of the test scores are non-finite: [      nan      nan 0.331625 0.33925       nan      nan 0.34625  0.34525
      nan      nan 0.346125 0.3455        nan      nan 0.345375 0.345
      nan      nan 0.343875 0.34525       nan      nan 0.345125 0.344875]
  warnings.warn(
```

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

```
[26] from sklearn.metrics import classification_report

     # Get the best Gradient Random Forest classifier model
     best_rf = grid_search.best_estimator_

     # Predict using the best Stacking Classifier model and encoded testing data
     y_pred = best_rf.predict(X_test_encoded)

     print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

    Abnormal       0.34      0.97      0.51       691
Inconclusive       0.24      0.01      0.02       637
      Normal       0.21      0.00      0.01       672

    accuracy                           0.34      2000
   macro avg       0.27      0.33      0.18      2000
weighted avg       0.27      0.34      0.19      2000
```

Secondly, the bagging model itself. Here, I aim to compare the bagging model in its default state with the tuned version. From the evaluation of both bagging models, it appears that their performance is quite balanced, showing relatively similar performance between the two models. However, both struggle in predicting the Inconclusive and Normal classes, evident from their low recall values for these classes. Though there's a slight improvement in the recall score for the Abnormal class in the second model, overall, their performance remains relatively consistent with each other. Further enhancements are needed to improve its precision, recall, and overall performance across different classes.

The default accuracy of the bagging model is 0.34, and after tuning, the accuracy decreased to 0.32. It can be concluded that the default bagging model is more accurate compared to the tuned one.

Bagging

By Default

```
[27] from sklearn.ensemble import BaggingClassifier
     from sklearn.metrics import accuracy_score

     # Initialize the Bagging classifier with default parameters
     bagging_default = BaggingClassifier(random_state=123)

     # Fit the Bagging classifier with encoded training data
     bagging_default.fit(X_train_encoded, y_train)
```

```
BaggingClassifier
BaggingClassifier(random_state=123)
```

```
[28] from sklearn.metrics import classification_report

     # Predict using the encoded testing data
     y_pred = bagging_default.predict(X_test_encoded)

     # Evaluate the model before tuning
     print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

    Abnormal       0.36      0.45      0.40       691
Inconclusive       0.29      0.29      0.29       637
      Normal       0.36      0.26      0.30       672

    accuracy                           0.34      2000
   macro avg       0.33      0.33      0.33      2000
weighted avg       0.34      0.34      0.33      2000
```

Tuning

```
[29] from sklearn.ensemble import BaggingClassifier
     from sklearn.model_selection import GridSearchCV

     # Define the parameters for GridSearchCV
     param_grid = {
         'n_estimators': [5, 20],
         'max_samples': [0.5, 2.0],
         'max_features': [0.5, 2.0],
         'bootstrap_features': [True, False]
     }

     # Initialize the Bagging classifier
     bagging_tuned = BaggingClassifier(random_state=123)

     # Initialize GridSearchCV
     grid_search = GridSearchCV(estimator=bagging_tuned, param_grid=param_grid, scoring='accuracy', cv=5)

     # Fit GridSearchCV with encoded training data
     grid_search.fit(X_train_encoded, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
60 fits failed out of a total of 80.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
20 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_bagging.py", line 326, in fit
    self._validate_params()
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 600, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_samples' parameter of BaggingClassifier must be an int in the range [1, inf) or a float in the range (0, 1]. Got 2.0 instead.

--------------------------------------------------------------------------------
40 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_bagging.py", line 326, in fit
    self._validate_params()
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 600, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of BaggingClassifier must be an int in the range [1, inf) or a float in the range (0, 1]. Got 2.0 instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: UserWarning: One or more of the test scores are non-finite: [0.33625  0.332625      nan      nan      nan      nan      nan      nan
 0.3375   0.323625      nan      nan      nan      nan      nan      nan]
  warnings.warn(
```

```
GridSearchCV
estimator: BaggingClassifier
BaggingClassifier
```

```
from sklearn.metrics import classification_report

# Get the best Gradient Bagging classifier model
best_bagging = grid_search.best_estimator_

# Predict using the best Stacking Classifier model and encoded testing data
y_pred = best_bagging.predict(X_test_encoded)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

    Abnormal       0.33      0.47      0.39       691
Inconclusive       0.29      0.28      0.29       637
      Normal       0.34      0.20      0.25       672

    accuracy                           0.32      2000
   macro avg       0.32      0.32      0.31      2000
weighted avg       0.32      0.32      0.31      2000
```

Thirdly, the boosting model. Here, I aim to compare the boosting model in its default state with the tuned version. From the evaluation of both boosting models are doing well in identifying the Abnormal class, showing better recall. However, they consistently struggle in recalling instances for Inconclusive and Normal classes, indicating the need for improvement. Despite fine-tuning, these models face challenges in accurately predicting these less common classes, even after enhancing Abnormal class recall. This emphasizes the need for more accurate adjustments or improvements in features to deal with these classification challenges. It's crucial to significantly improve recalling Inconclusive and Normal instances for overall performance enhancement and balanced predictions, while also ensuring consistent precision for Abnormal cases.

The default accuracy of the boosting model is 0.34, and after tuning, the accuracy improved to 0.35. It can be concluded that the tuned boosting model is slightly more accurate compared to the default one.

Boosting

By Default

```
[31] from sklearn.ensemble import GradientBoostingClassifier
     from sklearn.metrics import accuracy_score

     # Initialize the Gradient Boosting classifier with default parameters
     gb_default = GradientBoostingClassifier(random_state=123)

     # Fit the Gradient Boosting classifier with encoded training data
     gb_default.fit(X_train_encoded, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=123)
```

```
[32] from sklearn.metrics import classification_report

     # Predict using the encoded testing data
     y_pred = gb_default.predict(X_test_encoded)

     # Evaluate the model before tuning
     print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

    Abnormal       0.34      0.81      0.48       691
Inconclusive       0.34      0.10      0.16       637
      Normal       0.34      0.08      0.13       672

    accuracy                           0.34      2000
   macro avg       0.34      0.33      0.26      2000
weighted avg       0.34      0.34      0.26      2000
```

Tuning

```
[50] from sklearn.ensemble import GradientBoostingClassifier
     from sklearn.model_selection import GridSearchCV

     # Define the parameters for GridSearchCV
     param_grid = {
         'n_estimators': [50, 100, 200],
         'learning_rate': [0.01, 0.5],
         'max_depth': [2, 5]
     }

     # Initialize the Gradient Boosting classifier
     gb_tuned = GradientBoostingClassifier(random_state=123)

     # Initialize GridSearchCV
     grid_search = GridSearchCV(estimator=gb_tuned, param_grid=param_grid, scoring='accuracy', cv=5)

     # Fit GridSearchCV with encoded training data
     grid_search.fit(X_train_encoded, y_train)
```

```
GridSearchCV
estimator: GradientBoostingClassifier
GradientBoostingClassifier
```

```
[51] from sklearn.metrics import classification_report

     # Get the best Gradient Boosting classifier model
     best_gb = grid_search.best_estimator_

     # Predict using the best Stacking Classifier model and encoded testing data
     y_pred = best_gb.predict(X_test_encoded)

     print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

    Abnormal       0.35      0.86      0.49       691
Inconclusive       0.34      0.05      0.09       637
      Normal       0.35      0.09      0.15       672

    accuracy                           0.35      2000
   macro avg       0.35      0.34      0.24      2000
weighted avg       0.35      0.35      0.25      2000
```

Fourthly, the stacking model. Here, I aim to compare the stacking model in its default state with the tuned version. From the evaluation results of both stacking models, there are facing challenges in performance, particularly in predicting Inconclusive and Normal classes. Despite showing decent precision for Abnormal cases, the model struggles with low recall for these less frequent classes. Even after fine-tuning, both models continue to have difficulties in accurately predicting Inconclusive and Normal classes. To improve overall performance (accuracy) and achieve a more balanced prediction, significant enhancements are needed, especially in recalling instances from these classes, while maintaining the model's consistency in predicting Abnormal cases.

The default accuracy of the stacking model is 0.32, and after tuning, the accuracy improved to 0.33. It can be concluded that the tuned stacking model is slightly more accurate compared to the default one.

Stacking

By Default

```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Define the base estimators
estimators = [
    ('lr', LogisticRegression(random_state=123)),
    ('dt', DecisionTreeClassifier(random_state=123)),
    ('rf', RandomForestClassifier(random_state=123))
]

# Initialize the Stacking Classifier with default parameters
stack_default = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())

# Fit the Stacking Classifier with encoded training data
stack_default.fit(X_train_encoded, y_train)
```

```
StackingClassifier
lr                  dt                      rf
LogisticRegression  DecisionTreeClassifier  RandomForestClassifier
final_estimator
LogisticRegression
```

```
from sklearn.metrics import classification_report

# Predict using the encoded testing data
y_pred = stack_default.predict(X_test_encoded)

# Evaluate the model before tuning
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

    Abnormal       0.34      0.69      0.45       691
Inconclusive       0.30      0.24      0.27       637
      Normal       0.26      0.03      0.06       672

    accuracy                           0.32      2000
   macro avg       0.30      0.32      0.26      2000
weighted avg       0.30      0.32      0.26      2000
```

Tuning

```
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameters for GridSearchCV for the final estimator
param_grid = {
    'final_estimator__C': [0.1, 1, 10]
}

# Initialize the Stacking Classifier
stack_tuned = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=stack_tuned, param_grid=param_grid, scoring='accuracy', cv=5)

# Fit GridSearchCV with encoded training data
grid_search.fit(X_train_encoded, y_train)
```

```
GridSearchCV
estimator: StackingClassifier
lr                  dt                      rf
LogisticRegression  DecisionTreeClassifier  RandomForestClassifier
final_estimator
LogisticRegression
```

```
[45] from sklearn.metrics import classification_report

# Get the best Stacking Classifier model
best_stack = grid_search.best_estimator_

# Predict using the best Stacking Classifier model and encoded testing data
y_pred = best_stack.predict(X_test_encoded)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

    Abnormal       0.34      0.72      0.46       691
Inconclusive       0.31      0.23      0.27       637
      Normal       0.26      0.02      0.04       672

    accuracy                           0.33      2000
   macro avg       0.30      0.32      0.25      2000
weighted avg       0.30      0.33      0.26      2000
```

From all these models, it can be concluded that the boosting model provides the most accurate predictions compared to other models I've tried because it has the highest accuracy by default of 0.34 and 0.35 after tuning.