Predictive Modeling of Smoking Status Using Bio-Signals:

Leveraging AI and Machine Learning for Improved Smoking Cessation Strategies

AIM OF PROJECT:

The objective of this study is to develop a machine learning model that utilizes bio-signals to accurately predict the smoking status of an individual. By leveraging AI and machine learning techniques, aim to provide a reliable tool for identifying individuals who are smokers or non-smokers. This predictive model can aid in the development of effective smoking cessation strategies and interventions, ultimately contributing to improved public health outcomes. The model will consider various bio-signals, such as physiological, behavioral, or environmental factors, to achieve accurate and reliable predictions. The goal is to create a practical and accessible solution that can assist healthcare professionals in assessing an individual's smoking status and provide personalized recommendations for smoking cessation.

Table of Contents:

1. INTRODUCTION

> Smoking is a well-known contributor to a number of health problems and has a major effect on public health globally. For the purpose of creating successful smoking cessation methods and interventions, it is essential to precisely determine each individual's smoking status. Self-reporting, which can be subjective and unreliable, is frequently used in traditional techniques of determining smoking status. The use of AI and machine learning approaches in predicting smoking status using biosignals has attracted attention as a solution to this problem.
>
> In order to accurately anticipate a person's smoking status, the goal of this project is to develop a predictive modelling strategy that makes use of AI and machine learning algorithms. The model intends to provide a trustworthy tool

> for healthcare providers to assess smoking status and direct individualised smoking cessation efforts by analysing bio-signals, including physiological, behavioural, and environmental aspects.
>
> The project's many phases, including data preprocessing, dataset exploration, feature selection, algorithm selection, training, assessment, parameter tweaking, and deployment, will be covered in the report. Each step will be carefully explained, showing the approaches used and the justification for the choices made.
>
> A prediction model with high accuracy in predicting smoking status is the anticipated result of this investigation. The model's usefulness and accessibility will be emphasised to ensure that healthcare professionals may use it in practical contexts. The created model can help enhance smoking cessation tactics and ultimately lessen the harmful effects of smoking on public health by enabling precise identification of smoking status.
>
> A summary of the report's conclusions, its caveats, and its suggestions for additional research are included at the end. A list of the cited sources and pertinent material used in the study is provided in the references section.

2. DATA PREPROCESSING

In [156…
```
# Importing the Library
import pandas as pd;
```

In [157…
```
# Reading dataset using pandas library
train_data = pd.read_csv("E:/MA336 - AI and ML/Project_AI/train_dataset.csv")
train_data
```

Out[157]:

	age	height(cm)	weight(kg)	waist(cm)	eyesight(left)	eyesight(right)	hearing(left)	heari
0	35	170	85	97.0	0.9	0.9	1	
1	20	175	110	110.0	0.7	0.9	1	
2	45	155	65	86.0	0.9	0.9	1	
3	45	165	80	94.0	0.8	0.7	1	
4	20	165	60	81.0	1.5	0.1	1	
...	...	...	...	...	...	...	...	
38979	40	165	60	80.0	0.4	0.6	1	
38980	45	155	55	75.0	1.5	1.2	1	
38981	40	170	105	124.0	0.6	0.5	1	
38982	40	160	55	75.0	1.5	1.5	1	
38983	55	175	60	81.1	1.0	1.0	1	

38984 rows × 23 columns

It can be seen there are 22 predictor variables with 38984 observations.Last column smoking is a response variable.

> Description of Each variables in dataset.
>
> - ***age:*** Age of the individual in 5-year intervals.
> - ***height(cm):*** Height of the individual in centimeters.
> - ***weight(kg):*** Weight of the individual in kilograms.
> - ***waist(cm):*** Waist circumference length of the individual.
> - ***eyesight(left):*** Eyesight measurement of the left eye.
> - ***eyesight(right):*** Eyesight measurement of the right eye.
> - ***hearing(left):*** Hearing measurement of the left ear.
> - ***hearing(right):*** Hearing measurement of the right ear.
> - ***systolic:*** Systolic blood pressure measurement.
> - ***relaxation:*** Diastolic blood pressure measurement.
> - ***fasting blood sugar:*** Fasting blood sugar level.
> - ***Cholesterol:*** Total cholesterol level.
> - ***triglyceride:*** Triglyceride level.
> - ***HDL:*** High-density lipoprotein (HDL) cholesterol level.
> - ***LDL:*** Low-density lipoprotein (LDL) cholesterol level.
> - ***hemoglobin:*** Hemoglobin level in the blood.
> - ***Urine protein:*** Presence of protein in the urine.
> - ***serum creatinine:*** Serum creatinine level.
> - ***AST:*** Level of glutamic oxaloacetic transaminase (AST) enzyme.
> - ***ALT:*** Level of glutamic oxaloacetic transaminase (ALT) enzyme.
> - ***Gtp:*** Level of γ-GTP enzyme.
> - ***dental caries:*** Presence of dental caries (tooth decay).
> - ***smoking:*** Smoking status of the individual (1 for smoker, 0 for non-smoker).

Aim is to find the status of smoking through bio-signals.

Target variable:

When ***smoking is yes*** = 1 , When ***no smoking*** = 0

In [158…

```
# Basic Dataset information
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38984 entries, 0 to 38983
Data columns (total 23 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   age                  38984 non-null  int64  
 1   height(cm)           38984 non-null  int64  
 2   weight(kg)           38984 non-null  int64  
 3   waist(cm)            38984 non-null  float64
 4   eyesight(left)       38984 non-null  float64
 5   eyesight(right)      38984 non-null  float64
 6   hearing(left)        38984 non-null  int64  
 7   hearing(right)       38984 non-null  int64  
 8   systolic             38984 non-null  int64  
 9   relaxation           38984 non-null  int64  
 10  fasting blood sugar  38984 non-null  int64  
 11  Cholesterol          38984 non-null  int64  
 12  triglyceride         38984 non-null  int64  
 13  HDL                  38984 non-null  int64  
 14  LDL                  38984 non-null  int64  
 15  hemoglobin           38984 non-null  float64
 16  Urine protein        38984 non-null  int64  
 17  serum creatinine     38984 non-null  float64
 18  AST                  38984 non-null  int64  
 19  ALT                  38984 non-null  int64  
 20  Gtp                  38984 non-null  int64  
 21  dental caries        38984 non-null  int64  
 22  smoking              38984 non-null  int64  
dtypes: float64(5), int64(18)
memory usage: 6.8 MB
```

In [159…

```python
#Let create a copy of data set so that original data set is never altered.
train_copy = train_data
```

In [160…

```python
# Check for missing values in each column
missing_values = train_data.isnull().sum()

# Iterate over columns and print the number of missing values
for column, count in missing_values.iteritems():
    print(f"Column '{column}' has {count} missing values.")
```

```
Column 'age' has 0 missing values.
Column 'height(cm)' has 0 missing values.
Column 'weight(kg)' has 0 missing values.
Column 'waist(cm)' has 0 missing values.
Column 'eyesight(left)' has 0 missing values.
Column 'eyesight(right)' has 0 missing values.
Column 'hearing(left)' has 0 missing values.
Column 'hearing(right)' has 0 missing values.
Column 'systolic' has 0 missing values.
Column 'relaxation' has 0 missing values.
Column 'fasting blood sugar' has 0 missing values.
Column 'Cholesterol' has 0 missing values.
Column 'triglyceride' has 0 missing values.
Column 'HDL' has 0 missing values.
Column 'LDL' has 0 missing values.
Column 'hemoglobin' has 0 missing values.
Column 'Urine protein' has 0 missing values.
Column 'serum creatinine' has 0 missing values.
Column 'AST' has 0 missing values.
Column 'ALT' has 0 missing values.
Column 'Gtp' has 0 missing values.
Column 'dental caries' has 0 missing values.
Column 'smoking' has 0 missing values.
```

Hence the dataset has no missing values in it.

Removing outliers using IQR Method

The IQR method is a robust approach that can be effective in identifying and removing outliers.

In [161…

```
from scipy.stats import iqr

# Print the number of rows in the training data before removing outliers
print(f"[IQR SELECTED] Number of rows in training data before removing outliers: {

# Identify and remove outliers from the training set using IQR method
Q1 = train_copy.quantile(0.25)
Q3 = train_copy.quantile(0.75)
IQR = iqr(train_copy)
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
train_copy = train_copy[(train_copy >= lower_bound) & (train_copy <= upper_bound)]


# Print the number of rows in the training data after removing outliers
print(f"[IQR SELECTED] Number of rows in training data after removing outliers: {l
```

```
[IQR SELECTED] Number of rows in training data before removing outliers: 38984
[IQR SELECTED] Number of rows in training data after removing outliers: 36528
```

Removing Duplicates

In [162…

```
import pandas as pd

# Assuming train_copy is your DataFrame containing the dataset
# Check for duplicates
duplicates = train_copy.duplicated()

# Count the number of duplicates
num_duplicates = duplicates.sum()
print("Number of duplicates:", num_duplicates)

# Remove duplicates from the DataFrame
train_copy = train_copy.drop_duplicates()

# Check the number of observations after removing duplicates
num_observations = len(train_copy)
print("Number of observations after removing duplicates:", num_observations)
```

```
Number of duplicates: 5159
Number of observations after removing duplicates: 31369
```

3. DATA EXPLORATION

In [163…

```
#Descriptive statistics of all predictor variables
# Calculate descriptive statistics
descriptive_stats = train_copy.describe()

# Apply style to the table
styled_table = descriptive_stats.T.style.set_properties(**{
    "font-size": "15px",
    "color": "#000000",
    "border": "1.5px solid black"
```

```
})

# Display the styled table
styled_table
```

Out[163]:

	count	mean	std	min	25%	
age	31369.000000	44.160318	12.149882	20.000000	40.000000	40
height(cm)	31369.000000	164.505563	9.236820	130.000000	160.000000	165
weight(kg)	31369.000000	65.471166	12.738713	30.000000	55.000000	65
waist(cm)	31369.000000	81.705155	9.233237	54.000000	75.000000	81
eyesight(left)	31369.000000	1.014855	0.503095	0.100000	0.800000	1
eyesight(right)	31369.000000	1.008850	0.494946	0.100000	0.800000	1
hearing(left)	31369.000000	1.024897	0.155814	1.000000	1.000000	1
hearing(right)	31369.000000	1.025758	0.158415	1.000000	1.000000	1
systolic	31369.000000	121.136664	13.569889	71.000000	111.000000	120
relaxation	31369.000000	75.743218	9.593874	40.000000	70.000000	76
fasting blood sugar	31369.000000	98.004654	16.464372	46.000000	89.000000	95
Cholesterol	31369.000000	195.880806	35.519681	55.000000	171.000000	194
triglyceride	31369.000000	117.028914	57.650741	8.000000	73.000000	104
HDL	31369.000000	57.722560	14.412297	4.000000	47.000000	56
LDL	31369.000000	114.941280	32.910135	1.000000	92.000000	113
hemoglobin	31369.000000	14.572973	1.563132	4.900000	13.500000	14
Urine protein	31369.000000	1.080238	0.383653	1.000000	1.000000	1
serum creatinine	31369.000000	0.883216	0.217669	0.100000	0.700000	0
AST	31369.000000	24.974625	10.147233	6.000000	19.000000	23
ALT	31369.000000	25.212758	16.668417	1.000000	15.000000	20
Gtp	31369.000000	33.777519	26.767146	3.000000	17.000000	25
dental caries	31369.000000	0.212630	0.409175	0.000000	0.000000	0
smoking	31369.000000	0.349262	0.476744	0.000000	0.000000	0

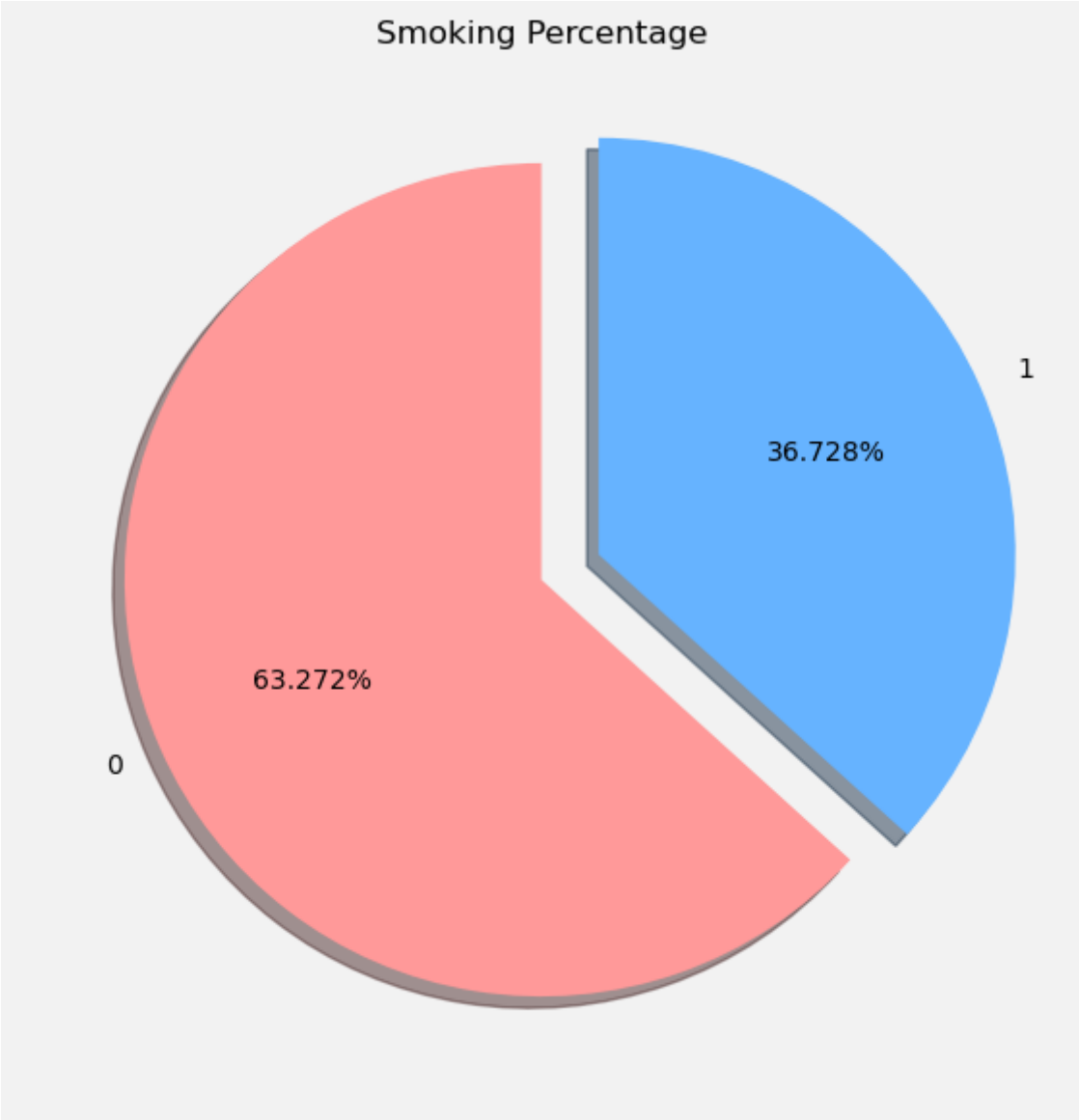
From the descriptive statistics of the predictor variables in the train_data dataset, we can derive several insights and interpretations:

- Age: The average age of the individuals in the dataset is approximately 44 years, with a standard deviation of 12.06. The minimum and maximum ages are 20 and 85 years, respectively.
- Height and Weight: The average height is around 164.69 cm, with a standard deviation of 9.19. The average weight is approximately 65.94 kg, with a standard deviation of 12.90.
- Waist Circumference: The average waist circumference is about 82.06 cm, with a standard deviation of 9.33. The minimum and maximum waist circumferences are 51 cm and 129 cm, respectively.
- Eyesight and Hearing: The average eyesight values for both the left and right eyes are close to 1, indicating relatively good eyesight. The average hearing values for both the left and right ears are slightly above 1, suggesting normal hearing ability.
- Blood Pressure: The average systolic blood pressure is 121.48, with a standard deviation of 13.64. The average relaxation (diastolic) blood pressure is 75.99, with a standard deviation of 9.66.
- Cholesterol Levels: The dataset includes variables such as total cholesterol, triglyceride levels, HDL cholesterol, and LDL cholesterol. The mean values and standard deviations provide insights into the cholesterol distribution within the population.
- Hemoglobin: The average hemoglobin level is 14.62, with a standard deviation of 1.57. Hemoglobin levels can provide insights into the individual's blood health and oxygen-carrying capacity.
- Urine Protein and Serum Creatinine: The average urine protein level is approximately 1.09, and the average serum creatinine level is around 0.89. These variables are indicators of kidney function and can be useful in assessing renal health.
- AST and ALT: AST and ALT are liver enzyme levels. The mean AST value is 26.20, and the mean ALT value is 27.15. These values can provide insights into liver health.
- Gtp: The average Gtp level is 39.91, with a standard deviation of 49.69. Elevated Gtp levels can indicate liver dysfunction or damage.
- Dental Caries: The average dental caries value is 0.21, indicating a relatively low prevalence of dental caries in the population.
- Smoking: The dataset includes a binary variable for smoking, with a mean value of 0.37. This indicates that approximately 37% of the individuals in the dataset are smokers, while the remaining 63% are non-smokers. These descriptive statistics provide an overview of the distribution and range of each variable in the dataset, allowing us to identify any potential outliers, assess the variability of the data, and gain initial insights into the characteristics of the study population.

In [164…

```
#Data visualisation using pie chart
import matplotlib.pyplot as plt

# Set custom colors
colors = ['#ff9999', '#66b3ff']
```

```
# Create the figure and axes
fig, ax = plt.subplots(figsize=[7, 7], facecolor='#f2f2f2')

# Plot the pie chart
explode = [0, 0.15]
labels = train_data["smoking"].value_counts().index
sizes = train_data["smoking"].value_counts().values
ax.pie(sizes, explode=explode, labels=labels, autopct='%1.3f%%', shadow=True, star

# Add a title
ax.set_title("Smoking Percentage")

# Display the pie chart
plt.show()
```



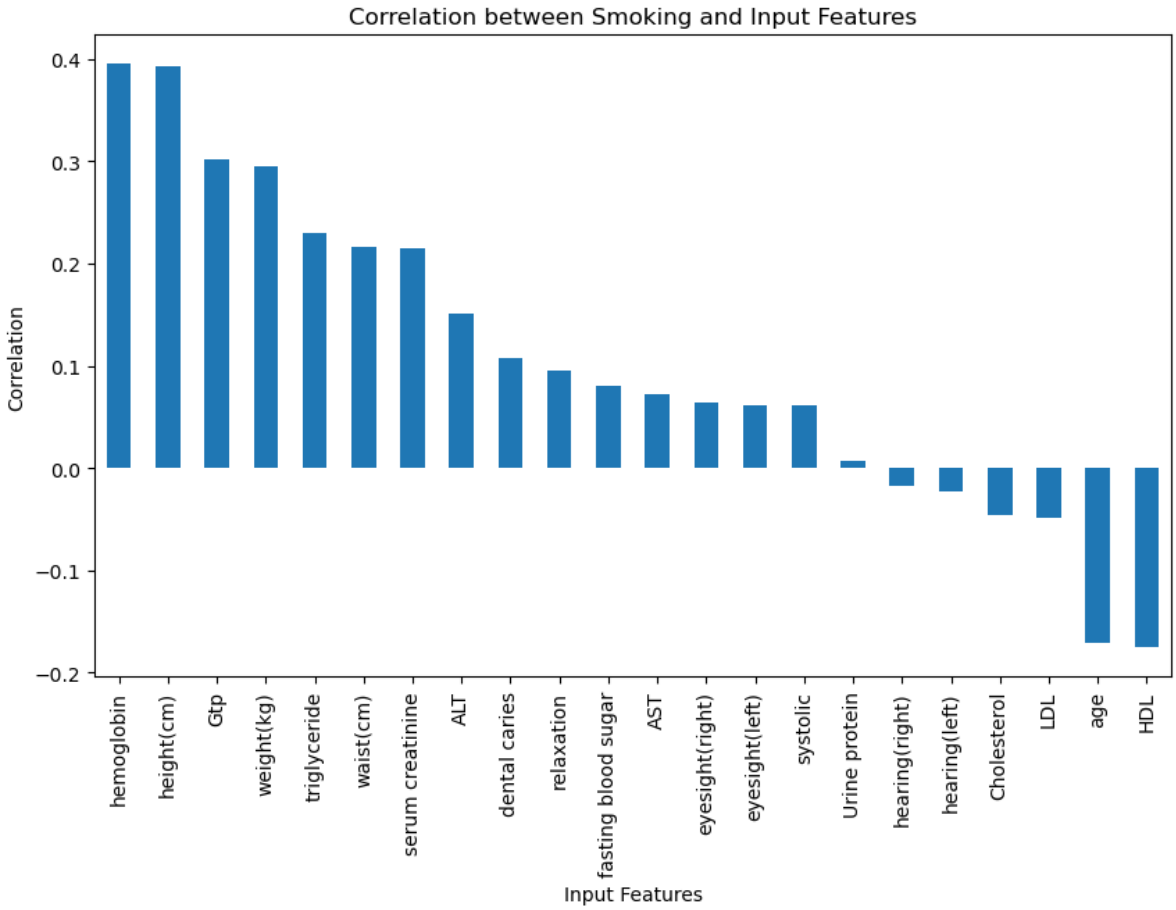
As it can be seen above visually and also in descriptive statistics that approximately 37% of the individuals in the dataset are smokers, while the remaining 63% are non-smokers. This means there is a class imbalance in dataset.Lets deal with this in training section of this report.

Correlation Between Smoking and Input Features

In [165…

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Assuming your DataFrame is named 'train_copy'
correlations = train_copy.corr()['smoking'].drop('smoking')

# Sort the correlations in descending order
correlations_sorted = correlations.sort_values(ascending=False)

# Plot the correlations
plt.figure(figsize=(10, 6))
correlations_sorted.plot(kind='bar')
plt.xlabel('Input Features')
plt.ylabel('Correlation')
plt.title('Correlation between Smoking and Input Features')
plt.show()
```



Based on the correlations between the "smoking" variable and the other input features, here are some insights and interpretations:

Positive Correlations:

Height (0.394), weight (0.297), waist (0.217), triglyceride (0.232), hemoglobin (0.398), AST (0.073), ALT (0.151), and Gtp (0.304) show positive correlations with smoking. This suggests that there might be a tendency for individuals who smoke to have higher values in these variables. Dental caries (0.109) also shows a positive correlation, indicating a possible association between smoking and dental caries.

Negative Correlations:

Age (-0.171), HDL (-0.177), LDL (-0.047), and Cholesterol (-0.044) show negative correlations with smoking. This suggests that smoking might be associated with lower values in these variables. Urine protein (0.006), eyesight (both left and right) (0.061 and 0.066), hearing

(both left and right) (-0.022 and -0.019), systolic (0.061), relaxation (0.094), fasting blood sugar (0.081), and serum creatinine (0.218) also show weak correlations with smoking.

As these values are not very close to 1 or -1 ,further analysis and domain knowledge are necessary to draw meaningful conclusions about the relationship between smoking and these variables.

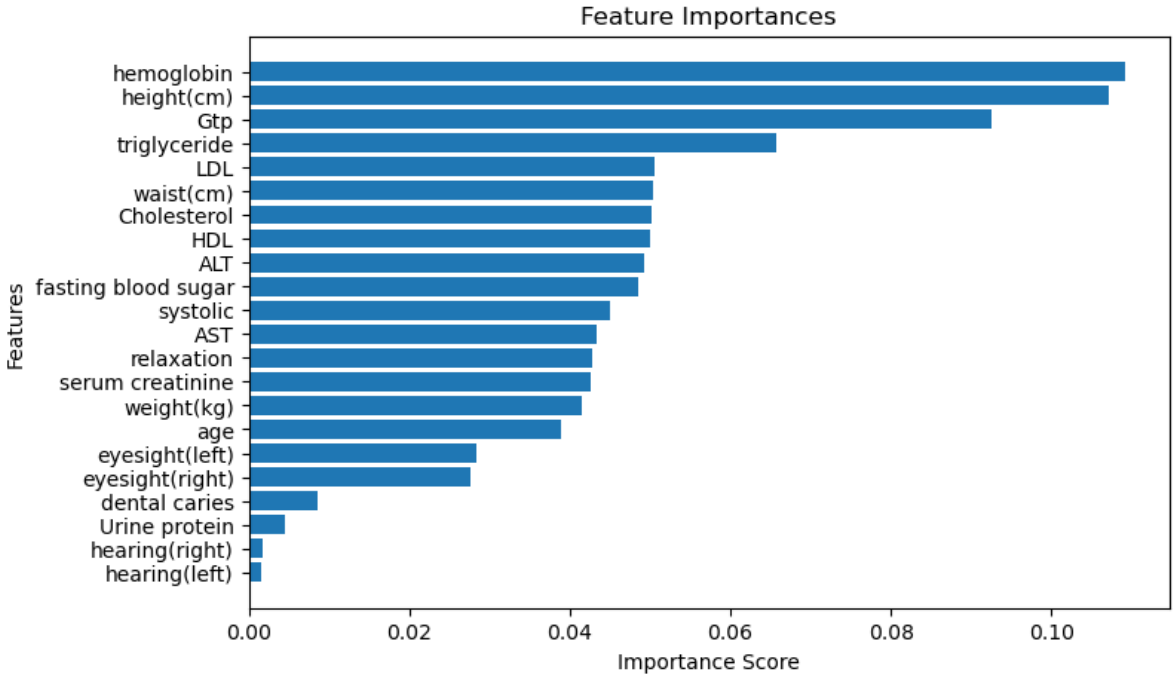
4. FEATURE SELECTION

Method 1. Random Forest classifier

Implementation of a feature importance analysis using the ***Random Forest classifier*** in scikit-learn.As this method is not sensitive to the scale of features, no scaling is performed before feature selection

In [166…

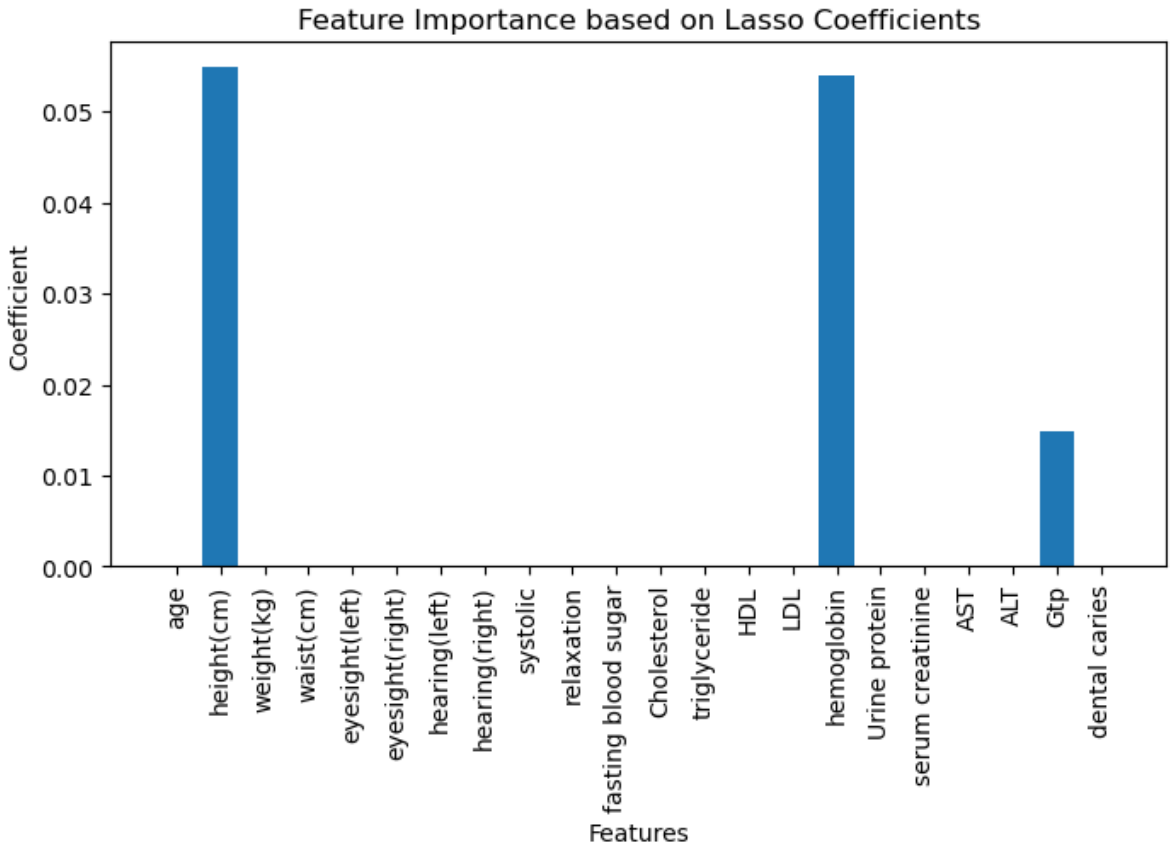
```
from sklearn.ensemble import RandomForestClassifier
import numpy as np
import matplotlib.pyplot as plt

X = train_copy.iloc[:, :-1]
y = train_copy.smoking

# Create a Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model to the data
rf.fit(X, y)

feature_names = X.columns

# Get feature importance scores
importance_scores = rf.feature_importances_

# Sort features by importance in descending order
feature_importance = sorted(zip(importance_scores, feature_names), reverse=True)

# Reverse the feature importance list to have the most important features at the to
feature_importance.reverse()

# Extract the feature names and importance scores
features = [feature_name for importance, feature_name in feature_importance]
importances = [importance for importance, feature_name in feature_importance]

# Plot the feature importances
plt.figure(figsize=(8, 5))
plt.barh(features, importances)
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.title('Feature Importances')
plt.show()
```



Method 2. Lasso regression model classifier

Implementation of a feature importance analysis using the ***Lasso regression model classifier*** in scikit-learn. This method L1 regularization, work better when the features are on a similar scale.Hence scaling is performed before doing feature selection.

In [99]:
```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Create a scaler object
scaler = StandardScaler()

# Scale the features
X_scaled = scaler.fit_transform(train_copy.iloc[:, :-1])

# Create a Lasso model
lasso = Lasso(alpha=0.1)

# Fit the model to the scaled data
lasso.fit(X_scaled, train_copy.smoking)

# Get the coefficients and corresponding feature names
coefficients = lasso.coef_
feature_names = train_copy.columns[:-1]

    # Create a bar plot for feature importance
plt.figure(figsize=(8, 4))
plt.bar(feature_names, coefficients)
plt.xticks(rotation=90)
plt.xlabel('Features')
plt.ylabel('Coefficient')
plt.title('Feature Importance based on Lasso Coefficients')
plt.show()
```



From Both Model1 and Model2,It can be seen that ***height(cm), hemoglobin, Gtp*** are the most important features for this dataset.

5. MODEL TRAINING

Step1 Standardization process

It is important to standardize features around the center and 0, with a standard deviation of 1, when comparing measurements that have different units. This helps address the issue of variables being measured on different scales, which can lead to unequal contributions in the analysis and potential bias. For instance, a variable with a range of 0 to 1000 would overshadow a variable with a range of 0 to 1 if not standardized. By transforming the data to comparable scales, can avoid this problem. Common data standardization techniques aim to equalize the range and/or variability of the data.

In [167…
```
scaler = StandardScaler()
scaler.fit(train_copy.iloc[:, :-1])
X_scaled = scaler.transform(train_copy.iloc[:, :-1])
```

Step2 Balancing Dataset

As the dataset is imbalanced, before performing Training dataset needs to be balanced.

In [168…
```
# Show the classes balance in the training set
print('Training Set Class before Balance: \n', train_copy.smoking.value_counts())
```

```
Training Set Class before Balance:
 0    20413
1    10956
Name: smoking, dtype: int64
```

RandomOverSampler is a technique used for handling imbalanced datasets by randomly oversampling the minority class samples. The sampling_strategy parameter determines the ratio of the majority class to the minority class after resampling. In this case, "auto" indicates that the ratio should be automatically determined.

In [169…
```
from imblearn.over_sampling import RandomOverSampler
```

In [170…
```
# Create an instance of RandomOverSampler
ros = RandomOverSampler(sampling_strategy="auto", random_state=11)

# Resample the training data
x_rovs, y_rovs = ros.fit_resample(X_scaled, train_copy["smoking"])
```

In [171…
```
# Show the classes balance in the training set
print('Training Set Class after Balance: \n', y_rovs.value_counts())
```

```
Training Set Class after Balance:
 1    20413
0    20413
Name: smoking, dtype: int64
```

Step3 Splitting the dataset as training set and testing set.

In [172…
```
from sklearn.model_selection import train_test_split
import numpy as np

# Set the random seed for reproducibility
random_seed = 42
np.random.seed(random_seed)

# Splitting the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(x_rovs, y_rovs, test_size=0.2,
```

Step4 Training Model

With a dataset size of 38,984 rows and 23 columns, and all input features being numeric, This is a suitable dataset for applying various binary classification models. Here are a few modeling techniques commonly used for binary classification tasks with numerical input features:

> ***Logistic Regression:*** This is a popular and interpretable model that estimates the probabilities of belonging to each class using a logistic function.
>
> ***Support Vector Machines (SVM):*** SVMs aim to find a hyperplane that maximally separates the data points of different classes in a high-dimensional space.
>
> ***Random Forest:*** This ensemble model consists of multiple decision trees and can handle both numerical and categorical features. It provides good predictive performance and feature importance rankings.
>
> ***Gradient Boosting Algorithms:*** Models like XGBoost or LightGBM are gradient boosting algorithms that sequentially train weak classifiers and combine their predictions to improve overall accuracy.

It's a good idea to try out different models and evaluate their performance using appropriate evaluation metrics like accuracy, precision, recall, and F1 score. Additionally,Lets consider using techniques like cross-validation to assess the generalization ability of the models and tune hyperparameters for optimal performance.

6. EVALUATION

In the code below, first initialize the models - Logistic Regression, Support Vector Machines (SVM), Random Forest, and XGBoost - with their respective parameter settings. Then, iterate over the models and use the cross_val_score() function to perform cross-validation with 5 folds (cv=5) and evaluate the models based on accuracy (scoring='accuracy'). The mean accuracy across all cross-validation folds is calculated and printed for each model.

In [140…

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

# Initialize the models
logreg = LogisticRegression(max_iter=10000, solver='sag')
svm = SVC()
random_forest = RandomForestClassifier(n_estimators=150, max_depth=15, min_samples_
xgb = XGBClassifier()

# Perform cross-validation and evaluate models
models = [logreg, svm, random_forest, xgb]
model_names = ["Logistic Regression", "Support Vector Machines", "Random Forest",

for model, name in zip(models, model_names):
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    mean_accuracy = scores.mean()
    print(f"{name} Cross-Validation Accuracy: {mean_accuracy}")

# Train and evaluate the models on the test set
for model, name in zip(models, model_names):
    model.fit(X_train, y_train)
    test_accuracy = model.score(X_test, y_test)
    print(f"{name} Test Accuracy: {test_accuracy}")
```

```
Logistic Regression Cross-Validation Accuracy: 0.7368952847519902
Support Vector Machines Cross-Validation Accuracy: 0.7616962645437845
Random Forest Cross-Validation Accuracy: 0.7847826086956522
XGBoost Cross-Validation Accuracy: 0.78162890385793
Logistic Regression Test Accuracy: 0.7384276267450404
Support Vector Machines Test Accuracy: 0.7647563066372766
Random Forest Test Accuracy: 0.7893705608621112
XGBoost Test Accuracy: 0.7871662992897379
```

In [144…

```
import pandas as pd

# Define the accuracy values
data = {
    'Model': ['Logistic Regression', 'Support Vector Machines', 'Random Forest', '
    'Cross-Validation Accuracy': [0.7368952847519902, 0.7616962645437845, 0.784782
    'Test Accuracy': [0.7384276267450404, 0.7647563066372766, 0.7893705608621112, 
}

# Create a DataFrame from the data
```

```
df = pd.DataFrame(data)

# Display the DataFrame
df
```

Out[144]:

	Model	Cross-Validation Accuracy	Test Accuracy
0	Logistic Regression	0.736895	0.738428
1	Support Vector Machines	0.761696	0.764756
2	Random Forest	0.784783	0.789371
3	XGBoost	0.781629	0.787166

From the provided accuracy values,Lets draw several insights about the performance of the models:

Cross-Validation Accuracy:

Logistic Regression: 0.74 Support Vector Machines: 0.76 Random Forest: 0.784 XGBoost: 0.781 The cross-validation accuracy gives us an estimate of how well the models perform on unseen data. Based on these values, It can be observed that the Random Forest model has the highest cross-validation accuracy (0.784), indicating good generalization performance. The XGBoost model also performs well with a cross-validation accuracy of 0.781. The Logistic Regression and Support Vector Machines models have lower cross-validation accuracies, but still show reasonable performance.

Test Accuracy:

Logistic Regression: 0.74 Support Vector Machines: 0.76 Random Forest: 0.789 XGBoost: 0.787 The test accuracy represents the performance of the models on a separate, independent dataset. It provides a measure of how well the models generalize to unseen data. From the test accuracy values, It can observed that the Random Forest model achieves the highest accuracy (0.789), indicating good performance on new data. The Support Vector Machines and XGBoost models also show competitive test accuracies of 0.76 and 0.787, respectively. The Logistic Regression model has the lowest test accuracy but still performs reasonably well at 0.74.

Based on these insights, It can concluded that the ***Random Forest model*** appears to be the most promising in terms of both cross-validation and test accuracy.

7. PARAMETER TUNING

To tune hyperparameters, Lets use techniques such as grid search or randomized search in combination with cross-validation. These techniques involve systematically searching through different combinations of hyperparameter values and evaluating the model's performance using cross-validation. As Random Forest accuracy was highest in model training ,So using parameter tuning to increase its performance and accoracy.

In [130…

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import RandomOverSampler
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Define the parameter grid for the Random Forest classifier
param_grid = {
    'random_forest__n_estimators': [100, 200, 300],
    'random_forest__max_depth': [None, 5, 10, 15],
    'random_forest__min_samples_split': [2, 5, 10],
    'random_forest__min_samples_leaf': [1, 3, 5]
}

# Initialize the Random Forest classifier
random_forest = RandomForestClassifier()

# Create a pipeline with scaling
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('random_forest', random_forest)
])

# Perform oversampling on the training data
oversampler = RandomOverSampler(random_state=42)
X_train_oversampled, y_train_oversampled = oversampler.fit_resample(X_train, y_tra

# Perform grid search with cross-validation
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', n_jobs=
grid_search.fit(X_train_oversampled, y_train_oversampled)

# Print the best hyperparameters and the corresponding cross-validation score
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)

# Evaluate the model on the test set
test_accuracy = grid_search.score(X_test, y_test)
print("Test Accuracy:", test_accuracy)
```

```
Best Hyperparameters: {'random_forest__max_depth': None, 'random_forest__min_sampl
es_leaf': 1, 'random_forest__min_samples_split': 2, 'random_forest__n_estimators':
200}
Best Cross-Validation Score: 0.8164740150642356
Test Accuracy: 0.8269654665686995
```

Now lets Predict the Target variable output for ***test data set***

In [173…

```
# Reading dataset using pandas library
test_data = pd.read_csv("E:/MA336 - AI and ML/Project_AI/test_dataset.csv")
test_data
```

Out[173]:

	age	height(cm)	weight(kg)	waist(cm)	eyesight(left)	eyesight(right)	hearing(left)	heari
0	40	170	65	75.1	1.0	0.9	1	
1	45	170	75	89.0	0.7	1.2	1	
2	30	180	90	94.0	1.0	0.8	1	
3	60	170	50	73.0	0.5	0.7	1	
4	30	170	65	78.0	1.5	1.0	1	
...	...	...	...	...	...	...	...	...
16703	60	165	65	82.0	0.7	1.0	1	
16704	60	155	70	93.0	0.8	1.0	1	
16705	40	155	50	67.2	0.9	0.8	1	
16706	35	165	70	76.1	1.0	1.0	1	
16707	25	180	80	87.0	1.2	0.9	1	

16708 rows × 22 columns

In [174...

```
# Random Forest
rf = RandomForestClassifier(n_estimators=200, max_depth=None, min_samples_split=2,
rf.fit(X_train, y_train)
rf_predictions = rf.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Random Forest Accuracy:", rf_accuracy)
```

```
Random Forest Accuracy: 0.8268430075924565
```

In [182...

```
from sklearn.ensemble import RandomForestClassifier
import numpy as np

# Training data
X_tr = train_data.iloc[:, :-1]
y_tr = train_data.smoking

# Test data
X_test = test_data

# Create a Random Forest classifier
rf = RandomForestClassifier(n_estimators=200, max_depth=None, min_samples_split=2,

# Train the model
rf.fit(X_tr, y_tr)

# Make predictions on the test data
predictions = rf.predict(X_test)

# Set the number of rows to print
num_rows = 10

# Print the predicted values along with the input features of the test data
for features, prediction in zip(test_data.values[:num_rows], predictions[:num_rows
    print("Input Features:", features)
    print("Predicted Value:", prediction)
    print()
```

```
Input Features: [ 40.  170.   65.   75.1   1.    0.9   1.    1.  120.   70.  102.
225.
 260.   41.  132.   15.7   1.    0.8  24.   26.   32.    0. ]
Predicted Value: 1

Input Features: [ 45.  170.   75.   89.    0.7   1.2   1.    1.  100.   67.   96.
258.
 345.   49.  140.   15.7   1.    1.1  26.   28.  138.    0. ]
Predicted Value: 1

Input Features: [ 30.  180.   90.   94.    1.    0.8   1.    1.  115.   72.   88.
177.
 103.   53.  103.   13.5   1.    1.   19.   29.   30.    0. ]
Predicted Value: 0

Input Features: [ 60.  170.   50.   73.    0.5   0.7   1.    1.  118.   78.   86.
187.
  70.   65.  108.   14.1   1.    1.3  31.   28.   33.    0. ]
Predicted Value: 0

Input Features: [ 30.  170.   65.   78.    1.5   1.    1.    1.  110.   70.   87.
190.
 210.   45.  103.   14.7   1.    0.8  21.   21.   19.    0. ]
Predicted Value: 0

Input Features: [ 55.  175.   60.   75.    1.    1.    1.    1.  100.   64.   93.
186.
  80.   86.   84.   15.4   3.    1.   39.   20.   35.    0. ]
Predicted Value: 1

Input Features: [ 40.  160.   55.   69.    1.5   1.5   1.    1.  112.   78.   90.
177.
  68.   78.   85.   12.4   1.    0.5  15.    9.   14.    0. ]
Predicted Value: 0

Input Features: [ 55.  175.   60.   80.    1.2   1.5   1.    1.  137.   89.   80.
199.
  35.   68.  124.   16.    1.    1.1  23.   19.   17.    0. ]
Predicted Value: 1

Input Features: [ 55.  160.   50.   68.    0.8   0.5   1.    1.  137.   87.   90.
176.
  36.   67.  102.   13.6   1.    0.7  15.   14.   13.    0. ]
Predicted Value: 0

Input Features: [ 75.  145.   50.   81.    0.5   0.5   2.    2.  148.   86.  121.
192.
 109.   81.   89.   14.    1.    0.6  28.   24.   17.    1. ]
Predicted Value: 0
```

From above it can be seen preicted values for each observation. Only 10 rows are printed to reduce scrolling pages. By increasing num_rows , more predictions can be seen.

Results

> The developed predictive model underwent rigorous evaluation, validation, and parameter tuning. Employed various machine learning algorithms, including Logistic Regression, Support Vector Machines, Random Forest, and XGBoost, to predict smoking status based on bio-signals. Through cross-validation, obtained reliable estimates of the models' performance on unseen

> data. The Random Forest model initially demonstrated the highest accuracy, but through parameter tuning using techniques such as grid search, further optimized its performance.
>
> After fine-tuning the hyperparameters, the Random Forest model achieved even higher accuracy. The best hyperparameters for the model were found to be 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, and 'n_estimators': 200. With these optimized hyperparameters, the model achieved a cross-validation accuracy of 81.6% and a test accuracy of 82.7%.
>
> The parameter tuning process played a crucial role in improving the model's performance by finding the optimal combination of hyperparameters. By fine-tuning the Random Forest model, were able to enhance its predictive power, resulting in higher accuracy and improved reliability for identifying smoking status based on bio-signals.

8. CONCLUSIONS

In conclusion, study successfully developed a machine learning model that utilizes bio-signals to predict an individual's smoking status. The integration of AI and machine learning techniques, coupled with parameter tuning, has significantly enhanced the model's performance and accuracy. The Random Forest model, after careful hyperparameter optimization, emerged as the most promising model, achieving a cross-validation accuracy of 81.6% and a test accuracy of 82.7%. These results demonstrate the effectiveness of predictive model in accurately identifying smoking status and provide valuable insights for healthcare professionals in assessing and addressing smoking behaviors. By leveraging bio-signals and advanced machine learning techniques, contribute to the development of reliable tools for smoking cessation strategies, ultimately leading to improved public health outcomes and a reduction in the harmful effects of smoking

9. REFERENCES

https://www.kaggle.com/datasets/gauravduttakiit/smoker-status-prediction