```
import tkinter


from tkinter import *
from tkinter import filedialog
import numpy as np
from PIL import ImageTk
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import numpy as np
import pandas as pd
from string import punctuation
import os
import pandas as pd
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler as ss
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category = FutureWarning)
from sklearn.model_selection import train_test_split
# import warnings filter
from warnings import simplefilter
# libraries that are used for analysis and visualization
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# %matplotlib inline


# Impoting data preprocessing libraries
```

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Importing model selection libraries.
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV

# Importing metrics for model evaluation.
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score, roc_curve
from sklearn.metrics import classification_report

# Importing machine learning models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

# Importing SMOTE for handling class imbalance.


# Importing warnings library. Would help to throw away warnings caused.
import warnings
warnings.filterwarnings('ignore')

simplefilter(action='ignore', category = FutureWarning)

main = tkinter.Tk()
main.title("Cardiovascular Disease Prediction using Machine Learning")  # designing main screen
main.geometry("1300x1200")
```

```
global filename

def upload():
    global filename
    filename = filedialog.askopenfilename(initialdir="dataset",
                                          title="Select a File",
                                          filetypes=(("Text files",
                                                      ".csv"),
                                                     ("all files",
                                                      ".")))




    text.delete('1.0', END)
    text.insert(END, " \n");
    heart = pd.read_csv(filename)
    text.insert(END, heart);
    text.insert(END, " \n");
    text.insert(END, filename + " Dataset is loaded\n");

def runMachineLearning():
    """### Dataset Loading"""
    text.delete('1.0', END)
    text.insert(END, " \n");
    # load the CARDIOVASCULAR RISK DATA from the drive
    risk_df = pd.read_csv('dataset/data_cardiovascular_risk.csv', index_col='id')

    """### Dataset First View"""

    # Viewing the top 5 rows to take a glimpse of the data
    text.insert(END,risk_df.head());
```

```
text.insert(END, " \n");

# Viewing the last 5 rows

text.insert(END, risk_df.tail());
text.insert(END, " \n");
"""### Dataset Rows & Columns Count"""

# Dataset Rows & Columns

text.insert(END, risk_df.shape);
text.insert(END, " \n");



"""### Dataset Information"""

# Dataset Info

text.insert(END, risk_df);
text.insert(END, " \n");

"""### Duplicate Values

*How important is it to get rid of duplicate records in my data?*

The mere presence of repeated data in the dataset is referred to as "duplication." This could be caused by incorrect data entry or procedures for collecting data. We can save time and money by not sending the same data to the machine learning model multiple times by removing duplicate data from our set.
"""
```

```
# Checking Duplicate Values
value = len(risk_df[risk_df.duplicated()])



text.insert(END, " \n");
text.insert(END, "The number of duplicate values in the data set is = ");
text.insert(END, " \n");
text.insert(END, value);



"""We found that there were no duplicate entries in the above data.


### Missing Values/Null Values


*Why dealing with missing values is necessary?*


There are frequently a lot of missing values in the actual data. Corrupted or missing data may result in missing values. Since many machine-learning algorithms do not support missing values, missing data must be handled during the dataset's pre-processing. Therefore, we begin by looking for values that are missing.
"""


# Missing Values/Null Values Count
text.insert(END, " \n");
text.insert(END, " Missing Values/Null Values Count");
text.insert(END, risk_df.isnull().sum());
text.insert(END, " \n");



"""We can see there are null values present in totChol, BMI, glucose, BPMeds, cigsPerDay, education has some null values and heartRate has 1 null value."""
```

```
# Missing Values Percentage
round(risk_df.isna().sum() / len(risk_df) * 100, 2)

# Visualizing the missing values using Heatmap
plt.figure(figsize=(12, 4))
sns.heatmap(risk_df.isna(), cmap='coolwarm')

"""### What did you know about your dataset?
```

The data comes from an ongoing cardiovascular study of Framingham, Massachusetts, residents. The purpose of the classification is to determine whether the patient is at risk for coronary heart disease (CHD) in the ten years to come. The data about the patients is provided by the dataset. It incorporates more than 4,000 records and 15 ascribes.

A classification algorithm is a method of supervised learning that divides data into various classes by utilizing data training. Data or observations are used to train classification predictive modeling, and new observations are categorized into classes or groups.

* There are 3390 rows and 16 columns in the dataset. In the 'education', 'cigsPerDay', 'BPMeds', 'totChol', 'BMI', 'heartRate', and 'glucose', there are missing values. The dataset does not contain any duplicate values.

* A potential risk factor exists for each attribute. Demographic, behavioral, and medical risk factors are these characteristics.

* First, we should try to understand the dataset through EDA, and then we can deal with null values later.

```
## *3. Understanding Your Variables*

###  Columns
"""

# Dataset Columns
```

risk_df.columns

"""### Statistical Summary"""

Dataset Describe (used to get statistics of numerical columns)

risk_df.describe().T

"""As can be seen in the statistical summary, there is skewness and outliers present in the continuous features 'cigsperday', 'totchol', 'sysbp', 'diebp', 'BMI', 'heartrate', and 'glucose' because there is such a large difference between the 75% percentile value and the maximum value.

Variables Description

Demographic

* *age :* Age of the patient (Continuous - Although the recorded ages have been truncated to whole numbers, the concept of age is continuous)

* *education :* level of education from 1 to 4 (Ordinal Variable)

* *sex :* male or female ("M" or "F")

Behavioral

* *is_smoking :* whether or not the patient is a current smoker ("YES" or "NO")

* *cigsPerDay :* the number of cigarettes that the person smoked on average in one day (can be considered continuous as one can have any number of cigarettes, even half a cigarette.)

Medical(history)

* *BPMeds :* whether or not the patient was on blood pressure medication (Nominal)

* *prevalentStroke :* whether or not the patient had previously had a stroke (Nominal)

* *prevalentHyp :* whether or not the patient was hypertensive (Nominal)

* *diabetes :* whether or not the patient had diabetes (Nominal)

Medical(current)

* *totChol :* total cholesterol level (Continuous)

* *sysBP :* systolic blood pressure (Continuous)

* *diaBP :* diastolic blood pressure (Continuous)

* *BMI :* Body Mass Index (Continuous)

* *heartRate :* heart rate (Continuous - In medical research, variables such as heart rate though in fact discrete, yet are considered continuous because of large number of possible values.)

* *glucose :* glucose level (Continuous)

Predict variable (desired target)

* *TenYearCHD :* (binary: â1â, means âYesâ, â0â means âNoâ)

Unique Values

"""

Check Unique Values for each variable.

for i in risk_df.columns.tolist():

  print("No. of unique values in", i, "is", risk_df[i].nunique())

"""## *4. EDA*

What is EDA?

* EDA stands for Exploratory Data Analysis. It is a process of analyzing and understanding the data, which is an essential step in the data science process. The goal of EDA is to gain insights into the data, identify patterns, and discover relationships and trends. It is an iterative process that helps to identify outliers, missing values, and any other issues that may affect the analysis and modeling of the data.

4.1 Numeric and Categorical features

"""

numeric_features = []

```
categorical_features = []

# splitting features into numeric and categoric.
'''
If feature has more than 10 categories we will consider it
as numerical_features, remaining features will be added to categorical_features.
'''
for col in risk_df.columns:
    if risk_df[col].nunique() > 10:
        numeric_features.append(col)
    else:
        categorical_features.append(col)
text.insert(END, " \n");
text.insert(END,f'numeric feature : {numeric_features}')
text.insert(END, " \n");
text.insert(END,f'category feature : {categorical_features}')
text.insert(END, " \n");
"""*Observation:*

* In our case, most of the categorical features are of the binary type, so the values are 0 and 1 (with a few exceptions). In terms of the numerical features, each one has a unique set of continuous and range values.

#### *4.2 Univariate Analysis*

##### *4.2.1 Data Distribution of Numeric features*
"""

# figsize
plt.figure(figsize=(15, 5))
# title
plt.suptitle('Data Distibution of Numeric Features', fontsize=20, fontweight='bold', y=1.02)
```

```
for i, col in enumerate(numeric_features):
    plt.subplot(2, 4, i + 1)  # subplots 2 rows, 4 columns

    # dist plots
    sns.distplot(risk_df[col])
    # x-axis label
    plt.xlabel(col)
    plt.tight_layout()

"""*Observation:*
```

* For numerical features, we can see that the majority of distributions are right-skewed. The distributions of totChol (total cholesterol) and BMI are roughly comparable. The distribution of glucose is highly skewed to the right. It demonstrates that glucose has many outliers.

```
##### *4.2.2 Outlier Analysis of Numeric features*
"""

# figsize
plt.figure(figsize=(15, 5))
# title
plt.suptitle('Outlier Analysis of Numeric Features', fontsize=20, fontweight='bold', y=1.02)

for i, col in enumerate(numeric_features):
    plt.subplot(2, 4, i + 1)  # subplots 2 rows, 4 columns

    # boxplots
    sns.boxplot(risk_df[col])
    # x-axis label
    plt.xlabel(col)
    plt.tight_layout()
```

"""*Observation:*

* Outliers are visible in the 'cigsPerDay', 'totChol', 'sysBP', 'diaBP', 'BMI', 'heartRate', and 'glucose' columns.

4.3 Bivariate and Multivariate Analysis

4.3.1 Scatter plot between target varible wrt to age and heartRate

"""

plotting graph to analyze age with respect to heartrate which are having Disease or No Disease

figsize

plt.figure(figsize=(10, 5))

scatterplot

sns.scatterplot(x='age', y='heartRate', hue='TenYearCHD', data=risk_df)

title

plt.title('Heart Disease wrt Age and heartRate')

plt.legend(['Disease', 'No Disease'])

"""*Observation:*

* This is a multi variate analysis between age, TenYearCHD and heartRate.

* There is a clear relation between age and Heart Disease, with the increase in age the chance of heart disease increases. There is no significant relationship between heart disease and heart rate

4.3.2 Regression plot between target variable and numerical features

"""

Checking Linearity of all numerical features with our target variable

```
# figsize
plt.figure(figsize=(15, 5))
# title
plt.suptitle('Bivariate Analysis of Numerical features', fontsize=20, fontweight='bold', y=1.02)

for i, col in enumerate(numeric_features):
    plt.subplot(2, 4, i + 1)  # subplots of 2 rows and 4 columns

    # regression plots
    sns.regplot(x=risk_df[col], y='TenYearCHD', data=risk_df)
    # x-axis lable
    plt.xlabel(col)
    plt.tight_layout()

"""*Observation:*
* Numerous Independent numerical variables are linked to our Target variable and have a positive relationship with our target variable.

##### *4.3.3 Bivariate Analysis of Categorical Features*
"""

# Counting number of category present in each feature with respect to target feature

# figsize
plt.figure(figsize=(15, 5))
# title
plt.suptitle('Bivariate Analysis of Categorical Features', fontsize=20, fontweight='bold', y=1.02)

for i, col in enumerate(
        categorical_features[:-1]):  # taking all features in categoric column except target feature(TenYearCHD)
    plt.subplot(2, 4, i + 1)  # subplots of 2 rows and 4 columns
```

```
    a = risk_df.groupby(col)[['TenYearCHD']].count().reset_index()


    # barplot
    sns.barplot(x=a[col], y=a['TenYearCHD'])
    # x-axis label
    plt.xlabel(col)
    plt.tight_layout()


"""## *5. Data Cleaning*


#### *What is data cleaning?*

* Data cleaning is the process of identifying and correcting or removing inaccuracies, inconsistencies, and missing values in a dataset. It is an important step in the data preparation process that ensures that the data is accurate, complete, and in a format that can be easily analyzed. Data cleaning may include tasks such as removing duplicate records, filling in missing values, correcting errors, and standardizing data formats. The goal of data cleaning is to improve the quality of the data and make it suitable for further analysis and modeling.


### *5.1 Duplicate Values*
"""


text.insert(END, " \n");
text.insert(END, " counting duplicate values");
text.insert(END, " \n");
text.insert(END, risk_df.duplicated().sum());
text.insert(END, " \n");


# counting duplicate values



"""There is no duplicate record in the dataset


### *5.2 Missing Values*
```

```
"""
text.insert(END, " \n");
text.insert(END, " Missing Values/Null Values Count");
text.insert(END, " \n");
text.insert(END, risk_df.isnull().sum());
text.insert(END, " \n");
# Missing Values/Null Values Count



# Missing Values Percentage
round(risk_df.isna().sum() / len(risk_df) * 100, 2)


# checking the shape of the data before missing value treatment
risk_df.shape


"""*Handling Missing Values*
```

* Typically, we use other records to replace these null values. However, the entries in this dataset are person-specific. The values vary from person to person, and the dataset is related to the medical field in this particular instance. Consequently, removing rows with any null value is the most logical choice we have for dealing with such values.

* We cannot take any risks with this prediction, so if we attempt to impute null values using advanced methods, it may affect the outcome because the values will be incorrect.

* *In the healthcare industry, every piece of data is crucial. Because of this, we came up with a solution by setting a threshold value. If a feature has less than 5% null values, we decide to drop those rows, and the remaining rows are imputing, which will affect prediction but not significantly.*

```
"""


# features which has less than 5%  null values present.
nan_columns = ['education', 'cigsPerDay', 'BPMeds', 'totChol', 'BMI', 'heartRate']


# dropping null values
```

```
risk_df.dropna(subset=nan_columns, inplace=True)


# glucose level are continuous in nature.
# Outlier are not treating yet thats why imputimg NaN values with median value.


risk_df['glucose'] = risk_df.glucose.fillna(risk_df.glucose.median())


# checking for null values after treating them.
risk_df.isna().sum()


# checking the shape of the data after missing values treatment
risk_df.shape


"""### *5.3 Skewness*"""


# statistical summary
risk_df.describe().T


"""As can be seen in the statistical summary for numerical features, there is a significant difference between the 75% percentile and maximum value, indicating that the dataset contains skewness and outliers."""


# figsize
plt.figure(figsize=(15, 5))
# title
plt.suptitle('Data distibution in numerical columns', fontsize=20, fontweight='bold', y=1.02)


for i, col in enumerate(numeric_features):
    plt.subplot(2, 4, i + 1)  # subplots of 2 rows and 4 columns


    # distplot
    sns.distplot(risk_df[col])
```

```
    # mean line
    plt.axvline(risk_df[col].mean(), color='blue', linestyle='dashed', linewidth=2)
    # median line
    plt.axvline(risk_df[col].median(), color='red', linestyle='dashed', linewidth=2)
    # x-axis label
    plt.xlabel(col)
    plt.tight_layout()

"""*Observation:*
```

* For numerical features, we can see that the majority of distributions are right-skewed. The distributions of totChol (total cholesterol) and BMI are roughly comparable. The distribution of glucose is highly skewed to the right. It demonstrates that glucose has many outliers.

* Some of the variables can get a normal distribution when outliers are removed. As a result, it appears that outliers should be removed before the transformation. First, we get rid of outliers, and then we check to see if we need to use the transformation technique again.

```
### *5.4 Treating Outliers*
"""

# figsize
plt.figure(figsize=(15, 5))
# boxplot of numerical features
sns.boxplot(data=risk_df[numeric_features])
plt.show()

"""* Since we have limited datapoint hence we are not simply removing the outlier instead of that we are using the clipping method.
```

Clipping Method: In this method, we set a cap on our outliers data, which means that if a value is higher than or lower than a certain threshold, all values will be considered outliers. This method replaces values that fall outside of a specified range with either the minimum or maximum value within that range.

```
"""
```

```
# we are going to replace the datapoints with upper and lower bound of all the outliers


def clip_outliers(risk_df):
    for col in risk_df[numeric_features]:
        # using IQR method to define range of upper and lower limit.
        q1 = risk_df[col].quantile(0.25)
        q3 = risk_df[col].quantile(0.75)
        iqr = q3 - q1
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr


        # replacing the outliers with upper and lower bound
        risk_df[col] = risk_df[col].clip(lower_bound, upper_bound)
    return risk_df

# using the function to treat outliers
risk_df = clip_outliers(risk_df)


# checking the boxplot after outlier treatment


# figsize


# boxplot of numerical features


# checking for distribution after treating outliers.


"""* *We can also observe some shifts in the distribution of the data after treating outliers. Some of the data were skewed before handling outliers, but after doing so, the features almost follow the normal distribution. Therefore, we are not utilizing the numerical feature transformation technique.*


## *6. Feature Engineering*
```

* Feature engineering is the process of creating new features from existing ones to improve the performance of a machine learning model. This involves transforming raw data into a more useful and informative form, by either creating new features from the existing data, or selecting only the most relevant features from the raw data.

* The goal of feature engineering is to extract relevant information from the raw data and represent it in a way that can be easily understood by the machine learning model. The success of a machine learning model depends heavily on the quality of the features used as inputs, so feature engineering plays an important role in model performance.

6.1 Encoding

Encoding is a technique in feature engineering that is used to convert categorical variables into numerical values that can be used by machine learning algorithms.

There are several encoding techniques, including:

* One-hot encoding: creates a binary column for each unique category, with a value of 1 indicating the presence of the category and 0 indicating the absence.

* Label encoding: assigns a unique integer value to each category.

* Ordinal encoding: assigns an ordered integer value to each category based on the natural ordering of the categories.

* Count encoding: replaces a categorical value with the number of times it appears in the dataset.

Except for the 'sex' and 'is_smoking' columns, almost all of the categories in the dataset are already represented numerically (ordinal). Therefore, we are encoding these two columns.

"""

Label Encoding

risk_df['sex'] = risk_df['sex'].map({'M': 1, 'F': 0})

risk_df['is_smoking'] = risk_df['is_smoking'].map({'YES': 1, 'NO': 0})

"""#####Checking if one hot encoding is required in any feature or not

#####*Every categorical feature except 'education' is binary and already encoded with 1 and 0 values hence we do not need to use One Hot Encoding for any column but we can use One Hot Encoding for education.*

We're using an algorithm that treats the categorical variables as unordered, such as decision trees or random forests, hence one-hot encoding can be more effective in representing the categorical variables.

"""

```
# Check Unique Values for each categorical variable.
for i in categorical_features:
    print("No. of unique values in", i, "is", risk_df[i].nunique())

# dropping our target variable from categorical features list
categorical_features.pop(-1)

# check the datatypes of each column in the DataFrame
risk_df.dtypes

# Cast values in the categorical columns as type str.
risk_df[categorical_features] = risk_df[categorical_features].astype(str)

# checking the result
risk_df.dtypes

# one-hot encode the 'education' feature
education_onehot = pd.get_dummies(risk_df['education'], prefix='education')

# drop the original education feature
risk_df.drop('education', axis=1, inplace=True)
```

```
# concatenate the one-hot encoded education feature with the rest of the data
risk_df = pd.concat([risk_df, education_onehot], axis=1)
risk_df.head(3)

"""### *6.2 Feature Selection*

Feature selection is a technique in machine learning where you select a subset of the most important features from a larger set of features to use as inputs for a model. The goal of feature selection is to reduce the number of features used in the model, while retaining the most important and relevant information from the data.

#### *6.2.1 Correlation Coefficient and Heatmap*

* The correlation coefficient is a numerical measure of the strength and direction of a linear relationship between two variables. In other words, it measures the extent to which changes in one variable are associated with changes in the other variable. The correlation coefficient ranges from -1 to 1, with -1 indicating a perfect negative correlation, 1 indicating a perfect positive correlation, and 0 indicating no correlation.

* The correlation coefficient is an important tool in data analysis and machine learning, as it can help to identify relationships between variables and can be used in feature selection techniques to remove highly correlated features, which can reduce overfitting and improve the performance of the model.
"""

# Plotting correlation heatmap
plt.figure(figsize=(15, 5))
sns.heatmap(risk_df.corr(), annot=True)

# find and remove correlated features

def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated features
    corr_matrix = dataset.corr()
```

```
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold:  # we are interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr


# checking the highly correlated features
correlation(risk_df, 0.7)  # setting threshold of 0.7


"""#### *6.2.2 Feature Manipulation*


##### *The readings of blood pressure are shown in two numbers.*
* The highest number in your systolic blood pressure reading.
* The lowest number on your diastolic blood pressure readings.

Pulse Pressure is the sum of the top number (systolic) and the bottom number (diastolic). For instance, a pulse pressure of 40 is considered to be healthy if the resting blood pressure is 120/80 millimeters of mercury (mm Hg). A pulse pressure greater than 40 mm Hg is typically considered unhealthy.

The risk of a heart event, such as a heart attack or stroke, can be predicted by measuring pulse pressure. High pulse pressure, especially in older people, is considered a risk factor for cardiovascular disease.
"""


# adding new column PulsePressure
risk_df['pulse_pressure'] = risk_df['sysBP'] - risk_df['diaBP']


# dropping the sysBP and diaBP columns
risk_df.drop(columns=['sysBP', 'diaBP'], inplace=True)
```

"""#####If a person smokes (is_smoking=='yes'), but the number of cigarettes smoked per day is 0, or cigsPerDay is 0. Then it may develop into a conflicting case, we must treat those records."""

checking data, weather the provide information is correct or not

risk_df[(risk_df.is_smoking == 'YES') & (risk_df.cigsPerDay == 0)]

"""since the is_smoking and cigsPerDay columns do not contain any conflict cases. It is sufficient to provide information regarding is_smoking in the cigsPerDay column."""

droping is_smoking column due to multi-collinearity

risk_df.drop('is_smoking', axis=1, inplace=True)

"""### *6.3 Extra Trees Classifier*

Extra_Tree_Classifier is a tree-based strategy that naturally ranks according to how well they decrease the Gini impurity (the purity of the node) across all trees. Notes with the least amount of impurity are found at the ends of trees, while nodes at the beginning of trees have the greatest decrease in impurity. As a result, we can select a subset of the most important features by pruning trees below a particular node.

"""

X = risk_df.drop('TenYearCHD', axis=1)

y = risk_df['TenYearCHD']

importing libarary

from sklearn.ensemble import ExtraTreesClassifier

model fitting

model = ExtraTreesClassifier()

model.fit(X, y)

ranking feature based on importance

```
ranked_features = pd.Series(model.feature_importances_, index=X.columns)
print(ranked_features.sort_values(ascending=False))

"""### *6.4 Chi-square Test*

In feature selection, the chi-square test can be used to determine if a variable is related to target variable. If the p-value of the test is low, it indicates that there is a significant relationship between the two variables, and the variable can be selected as an important feature for the model.
"""

# importing libarary
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# model fitting
ordered_rank_features = SelectKBest(score_func=chi2, k='all')
model = ordered_rank_features.fit(X, y)

# ranking feature based on importance
feature_imp = pd.Series(model.scores_, index=X.columns)
feature_imp.sort_values(ascending=False)

"""### *6.5 Information Gain*

Information gain is used in decision trees to select the attribute with the highest information gain as the root node or the top level of the tree. Information gain measures the difference between the entropy of the original dataset and the entropy of the subsets created by splitting the data on an attribute.
"""

# importing libarary
from sklearn.feature_selection import mutual_info_classif
```

```
# model fitting
mutual_info = mutual_info_classif(X, y)

# ranking feature based on importance.
mutual_data = pd.Series(mutual_info, index=X.columns)
mutual_data.sort_values(ascending=False)

"""### *6.6 Feature Importance*"""

# plotting graph ---> Feature Importance
fig, axs = plt.subplots(1, 3, figsize=(15, 4))

ranked_features.sort_values(ascending=False).plot(kind='barh', title='ExtraTreesClassifier',
ax=axs[0])
feature_imp.sort_values(ascending=False).plot(kind='barh', title='Chi Square test', ax=axs[1])
mutual_data.sort_values(ascending=False).plot(kind='barh', title='Information Gain', ax=axs[2])

plt.suptitle('Feature Importance', fontsize=20, fontweight='bold', y=1.1)
plt.tight_layout()

"""*From these graphs, we can say that the two most important features are 'age and
'pulse_pressure' to predict the target variable*

As we discussed earlier, in the healthcare industry, every piece of data is crucial for analyzing or
forecasting the target variable. The entries in this dataset are person-specific, the values vary
between individuals, and all of the features are very important.

That is why I am using all features, except multi-collinear features, to train the model.
"""

# plotting correlation heatmap to check multicollinearity.
plt.figure(figsize=(15, 4))
```

```
sns.heatmap(risk_df.drop(columns='TenYearCHD').corr(), annot=True)

correlation(risk_df, 0.7)  # 0.7 is the thresold value for multicollinearity

"""Within the independent variables, there is no high multicollinearity.

### *6.7 Dependent and Independent Variable Assignment*
"""

# copying the data to save the work done till now
model_df = risk_df.copy()
model_df.head(3)

X = model_df.drop(columns='TenYearCHD')  # independent features
y = model_df['TenYearCHD']  # dependent features

"""### *6.8 Handling Imbalance Target Variable*

Checking if data is balanced or not
"""

# Dependant Column Value Counts
print(model_df.TenYearCHD.value_counts())
print(" ")

# Dependant Variable Column Visualization
fig, ax = plt.subplots(1, 2, figsize=(15, 6))

# pie chart for percentage
model_df['TenYearCHD'].value_counts().plot(kind='pie', autopct="%1.1f%%", startangle=90,
ax=ax[0])
```

```
# bar chart for count
model_df['TenYearCHD'].value_counts().plot(kind='bar', ax=ax[1])
plt.show()

"""When there are significantly more instances of certain classes than others, the issue of class imbalance typically arises. Class imbalance in the target class is a problem for machine learning models because it can result in biased predictions. That is why we need to balance the target class.

*The data set differs significantly. Our data, therefore, lack balance. We will use the Synthetic Minority Oversampling Technique (SMOTE) to resolve this issue.*

* SMOTE (Synthetic Minority Oversampling Technique) works by randomly selecting a minority class point and calculating its k-nearest neighbors. Between the selected point and its neighbors, the synthetic points are added. Continue with the steps until the data is balanced.
"""

## Handling target class imbalance using SMOTE
from collections import Counter

print(f'Before Handling Imbalanced class {Counter(y)}')

# Resampling the minority class

# fit predictor and target variable

print(f'After Handling Imbalanced class {Counter(y)}')

"""We have successfully balanced the target variable

## *7. Model Building*

### *7.1 Train Test Split*
"""
```

```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=33)



text.insert(END, " \n");
text.insert(END, " Train Dataset Shape:\n")
text.insert(END, " \n");
text.insert(END,X_train.shape);
text.insert(END, " \n");
text.insert(END, " Test Dataset Shape:\n")
text.insert(END, " \n");
text.insert(END, X_test.shape);
text.insert(END, " \n");
"""### *7.2 Scaling Data*"""

# Scaling Data
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

"""#### *Evaluation Metrics Used -*
```

* Accuracy: Simply put, accuracy is the percentage of times that the classifier correctly predicts. The ratio of the total number of predictions to the number of correct predictions is how accuracy is defined. If a model has a 99 percent accuracy rate, you might think it is doing very well. However, this is not always the case and can be misleading in some situations.

When the target class is well-balanced, accuracy is useful, but it is not a good choice for unbalanced classes.

* Confusion Matrix: The Confusion Matrix is a performance measurement for classification problems in machine learning in which there can be two or more classes output. It is a table with actual and predicted value combinations. The table that is frequently used to describe the

performance of a classification model on a set of test data for which the true values are known is referred to as a confusion matrix. It is extremely helpful for determining the AUC-ROC curves, precision, recall, and accuracy.

* Precision: Precision explains the percentage of correctly predicted cases that were actually successful. When False Positives are more of a concern than False Negatives, precision can be useful.

A label's precision is calculated by dividing the number of predicted positives by the number of true positives.

* Recall: Recall describes the proportion of actual positive cases that our model correctly predicted. When False Negative is more important than False Positive, this metric is helpful. In medical cases, it matters whether we raise a false alarm or not, but the actual positive cases should not go unnoticed. The number of true positives divided by the total number of actual positives is the definition of recall for a label.

* F1 Score: This score incorporates both Precision and Recall metrics. When Precision and Recall are equal, it reaches its peak.

The harmonic mean of recall and precision is the F1 Score.

* AUC-ROC: The Receiver Operator Characteristic (ROC) is a probability curve that separates the "signal" from the "noise" by plotting the TPR (True Positive Rate) against the FPR (False Positive Rate) at various threshold values. The measure of a classifier's ability to differentiate between classes is the Area Under the Curve (AUC). This simply indicates that the classifier is able to precisely differentiate between all Positive and Negative class points when AUC is equal to 1. The classifier would correctly identify all negatives as positives when the AUC was zero.

7.3 Model Training

"""

empty list for appending performance metric score

model_result = []

def predict(ml_model, model_name):

'''

Pass the model and predict value.

Function will calculate all the eveluation metrics and appending those metrics score on model_result table.

Plotting confusion_matrix and roc_curve for test data.

'''

```
text.insert(END, " \n");

text.insert(END, model_name)

text.insert(END, " \n");

text.insert(END, "------------------------------------------------------------");

# model fitting

model = ml_model.fit(X_train, y_train)


# predicting value and probability

y_train_pred = model.predict(X_train)

y_test_pred = model.predict(X_test)

y_train_prob = model.predict_proba(X_train)[:, 1]

y_test_prob = model.predict_proba(X_test)[:, 1]


''' Performance Metrics '''

# accuracy score  ---->  (TP+TN)/(TP+FP+TN+FN)

train_accuracy = accuracy_score(y_train, y_train_pred)

test_accuracy = accuracy_score(y_test, y_test_pred)

text.insert(END, " \n");

text.insert(END,f'train accuracy : {round(train_accuracy, 3)}')

text.insert(END, " \n");

text.insert(END,f'test accuracy : {round(test_accuracy, 3)}')

text.insert(END, " \n");
```

```
# precision score  ----->  TP/(TP+FP)
train_precision = precision_score(y_train, y_train_pred)
test_precision = precision_score(y_test, y_test_pred)

text.insert(END,f'train precision : {round(train_precision, 3)}')
text.insert(END, " \n");
text.insert(END,f'test precision : {round(test_precision, 3)}')
text.insert(END, " \n");
# recall score  ----->  TP/(TP+FN)
train_recall = recall_score(y_train, y_train_pred)
test_recall = recall_score(y_test, y_test_pred)
text.insert(END,f'train recall : {round(train_recall, 3)}')
text.insert(END, " \n");
text.insert(END,f'test recall : {round(test_recall, 3)}')
text.insert(END, " \n");

# f1 score  ----->  Harmonic Mean of Precision and Recall
train_f1 = f1_score(y_train, y_train_pred)
test_f1 = f1_score(y_test, y_test_pred)
text.insert(END,f'train f1 : {round(train_f1, 3)}')
text.insert(END, " \n");
text.insert(END,f'test f1 : {round(test_f1, 3)}')
text.insert(END, " \n");
# roc_auc score  ----->  It shows how well the model can differentiate between classes.
train_roc_auc = roc_auc_score(y_train, y_train_prob)
test_roc_auc = roc_auc_score(y_test, y_test_prob)
text.insert(END,f'train roc_auc : {round(train_roc_auc, 3)}')
text.insert(END, " \n");
text.insert(END,f'test roc_auc : {round(test_roc_auc, 3)}')
text.insert(END, " \n");
```

```
text.insert(END,'-' * 80)
text.insert(END, " \n");

# classification report
text.insert(END,f'classification report for test data \n{classification_report(y_test, y_test_pred)}')
text.insert(END, " \n");
text.insert(END,'-' * 80)
text.insert(END, " \n");
''' plotting ROC curve '''
fpr, tpr, threshold = roc_curve(y_test, y_test_prob)
plt.plot(fpr, tpr, label=f'ROC - {model_name}')
plt.plot([0, 1], [0, 1], '--')
plt.title('ROC curve on Test data', weight='bold')
plt.xlabel('False Positive Rate----->')
plt.ylabel('True Positive Rate----->')
plt.legend(loc=4)

''' actual value vs predicted value on test data'''
d = {'y_actual': y_test, 'y_predict': y_test_pred}
print(pd.DataFrame(data=d).head(10).T)  # constructing a dataframe with both actual and predicted values
print('-' * 80)

# using the score from the performance metrics to create the final model_result.
model_result.append({'model': model_name,
                     'train_accuracy': train_accuracy,
                     'test_accuracy': test_accuracy,
                     'train_precision': train_precision,
                     'test_precision': test_precision,
                     'train_recall': train_recall,
                     'test_recall': test_recall,
```

```
                'train_f1': train_f1,
                'test_f1': test_f1,
                'train_roc_auc': train_roc_auc,
                'test_roc_auc': test_roc_auc})

"""## *8. Model Implementation*

### *8.1 Logistic Regression*

Logistic regression is one of the simplest algorithms for estimating the relationship between independent variables and a single dependent binary variable and determining the likelihood of an event occurring.

The regulation parameter C controls the trade-off between keeping the model simple (underfitting) and increasing its complexity (overfitting). With increasing values of C, the model becomes more complicated and the power of regulation decreases, resulting in an overfitting of the data.
"""

predict(LogisticRegression(), 'LogisticRegression')

"""### *8.2 SVM (Support Vector Machine)*

Classification is carried out by a Support Vector Machine (SVM) by locating the hyperplane with the greatest margin between the two classes. The support vectors are the vectors (cases) that define the hyperplane. Finding a hyperplane in an N-dimensional space that clearly classifies the data points is the goal of the SVM algorithm.

The number of features determines the hyperplane's dimension. The hyperplane is just a line if there are two input features. The hyperplane transforms into a two-dimensional plane when there are three input features. When the number of features is greater than three, it becomes difficult to imagine.
"""

predict(SVC(probability=True), 'SVM')
```

"""### *8.3 KNN (K-Nearest Neighbours)*

A supervised machine learning algorithm known as KNN or K-nearest neighbor can be used to solve classification and regression problems. K is not a non-parametric nearest neighbor, i.e. It makes no assumptions regarding the assumptions that underlie the data. An input or unseen data set is categorized here by the algorithm based on the characteristics shared by the closest data points. The distance between two points determines these closest neighbors. The distance metric methods that are utilized can be Euclidean Distance, Manhattan Distance, Minkowski, Cosine Similarity Measure etc)

"""

```
# Checking the optimum value of the k:
accuracy = []

# Iteratig for the optimum values of k
for i in range(1, 15):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    accuracy.append(knn.score(X_test, y_test))

# plotting the k-value vs accuracy
plt.title('k-NN Varying number of neighbors')
plt.plot(range(1, 15), accuracy)
plt.xlabel('number of neighbours')
plt.ylabel('Accuracy')
plt.show()
```

"""The best accuracy is at K=1. So we will concentrate on low values of k; k=3 is superior to k=2. For binary classification, k is typically an odd number (to prevent ties) of at least three."""

```
predict(KNeighborsClassifier(n_neighbors=1), 'KNN')
```

"""### *8.4 Decision Tree*

A decision tree is a tree-like model used in machine learning to make predictions or decisions by breaking down a set of rules or conditions into smaller and smaller sub-conditions, based on the values of the input features.

Each node in the tree represents a test on a feature, and each branch represents the outcome of the test. The final branches of the tree, called the leaves, represent the class predictions or decisions. The tree is built recursively by finding the best feature to split the data based on the information gain or decrease in impurity at each node.

"""

predict(DecisionTreeClassifier(), 'DecisionTree')

"""### *8.5 Random Forest*

Random Forest is an ensemble machine learning algorithm that builds multiple decision trees and combines their predictions to make a final classification or regression prediction. In contrast to a single decision tree, Random Forest reduces the risk of overfitting by combining the results of many trees, each built on a different subset of the data.

Hyperparameter Tunning using RandomizedSearchCV

"""

rf_params = {'n_estimators': [500, 600, 800], # number of trees in the ensemble

'max_features': ["log2", "sqrt"], # maximum number of features considered when splitting a node.

'max_depth': [35, 40, 45, 50], # maximum number of levels allowed in each tree.

'min_samples_split': [7, 9, 12],

minimum number of samples necessary in a node to cause node splitting.

'min_samples_leaf': [4, 7, 10]} # minimum number of samples which can be stored in a tree leaf.

performing Hyperparameter Tunning using RandomizedSearchCV

```
rf = RandomForestClassifier()
rf_randomsearch = RandomizedSearchCV(estimator=rf, param_distributions=rf_params, n_iter=2, cv=5, verbose=2)

# model fitting
rf_randomsearch.fit(X_train, y_train)

optimal_model = rf_randomsearch.best_estimator_
optimal_model

predict(optimal_model, 'RandomForest')

# importance feature
importances = optimal_model.feature_importances_

# Creating a dictonary
importance_dict = {'Feature': list(X.columns),
                   'Feature Importance': importances}

# Creating the dataframe
importance = pd.DataFrame(importance_dict)
sorting_features = importance.sort_values(by=['Feature Importance'], ascending=False)
sorting_features

# plotting feature importance graph
plt.figure(figsize=(15, 5))
sns.barplot(x='Feature Importance', y='Feature', data=sorting_features, color='blue')
plt.show()

"""### *8.6 AdaBoost*
```

```
    AdaBoost (Adaptive Boosting) is an ensemble machine learning algorithm that combines multiple
weak models to form a stronger model. It works by assigning weights to the data points in a dataset
and iteratively building weak models that try to correctly classify or predict the target variable. After
each iteration, the weights of the misclassified or mispredicted data points are increased, making it
more likely that the next weak model will focus on these points.
    """

    predict(AdaBoostClassifier(), 'AdaBoost')



def predictDisease():
    import GUI

font = ('times',16, 'bold')
title = Label(main, text='CARDIOVASCULAR DISEASE PREDICTION USING MACHINE LEARNING')
title.config(bg='white', fg='purple')
title.config(font=font)
title.config(height=3, width=120)
title.place(x=0, y=5)

font1 = ('times', 14, 'bold')
uploadButton = Button(main, text="Upload Heart Disease Dataset", command=upload)
uploadButton.place(x=50, y=100)
uploadButton.config(font=font1)

readButton = Button(main, text="Run Machine Learning Algorithms",
command=runMachineLearning)
readButton.place(x=350, y=100)
readButton.config(font=font1)

cleanButton = Button(main, text="Predict Heart Disease Risk", command=predictDisease)
cleanButton.place(x=760, y=100)
cleanButton.config(font=font1)
```

```
font1 = ('times', 12, 'bold')
text = Text(main, height=25, width=150)
scroll = Scrollbar(text)
text.configure(yscrollcommand=scroll.set)
text.place(x=10, y=200)
text.config(font=font1)

main.config(bg='purple')
main.mainloop()
```