



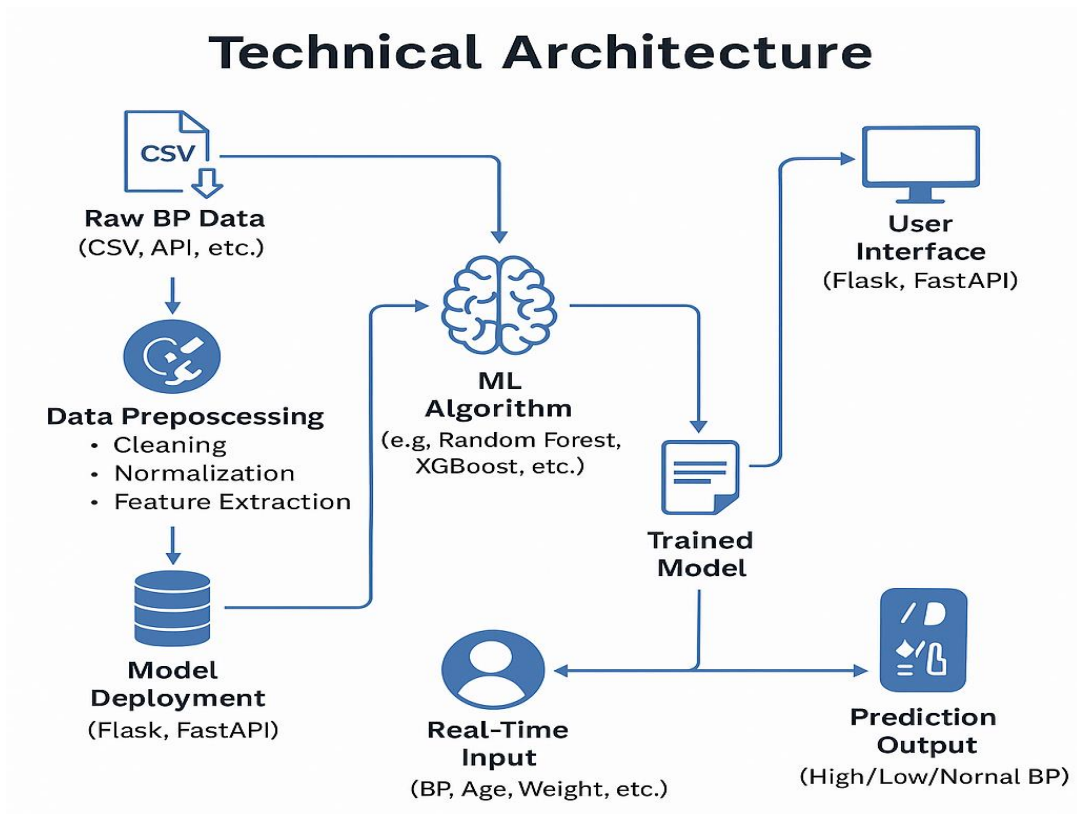
## Predictive Pulse: Harnessing Machine Learning for Blood Pressure Analysis

Name	Enrollment no	Email-id
Ujjval Patel	220090116051	Ujjvalpatel271@gmail.com
Deep Patel	220090116039	Pdeep0650@gmail.com
Harshil Patel	220090116043	Hp2954752@gmail.com

## Predictive Pulse: Harnessing Machine Learning for Blood Pressure Analysis:

Machine learning is transforming blood pressure analysis by enabling more accurate, real-time predictions and personalized insights from complex health data. By identifying subtle patterns and correlations in vast datasets, these advanced algorithms can aid early detection of hypertension, optimize treatment strategies, and support proactive patient care. This innovative approach holds the potential to revolutionize how clinicians interpret and manage blood pressure, offering a new level of precision and foresight in cardiovascular health management.

### Technical Architecture:



## Project Flow:

1. User Interaction & Input:
  - **User Input:** The user enters their blood pressure and relevant health data through the web interface.
2. Model Integration & Prediction:
  - **Data Sending:** The web interface sends the entered data to the integrated machine learning (ML) model.
  - **Prediction Process:** The model processes the input data and returns the prediction to the UI.
  - **Display Prediction:** The predicted blood pressure category is shown to the user in real-time.
1. Data Collection & Preparation:
  - **Data Gathering:** Collect blood pressure data and related health metrics from reliable sources, such as CSV files, APIs, or medical records.
  - **Data Cleaning and Transformation:** Clean the raw data by removing errors, handling missing values, and normalizing the features. Engineer necessary variables to make the data suitable for the ML model.
2. Exploratory Data Analysis (EDA):
  - **Descriptive Statistics:** Compute basic statistical measures (mean, median, range) to understand data distribution and quality.
  - **Visual Analysis:** Generate visualizations (histograms, boxplots, scatterplots) to spot trends, correlations, and outliers in the data.
3. Model Building:
  - **Algorithm Selection:** Train the model using different algorithms like Random Forest, XGBoost, SVM, etc., to detect patterns between features and blood pressure categories.
  - **Model Evaluation:** Evaluate the models' performance using validation data.
4. Performance Testing & Hyperparameter Tuning:
  - **Testing with Metrics:** Evaluate the models using metrics like accuracy, precision, recall, F1-score, and ROC-AUC.
  - **Hyperparameter Optimization:** Use grid search or random search to fine-tune the model's hyperparameters. Compare the model's performance before and after tuning.
5. Model Deployment:
  - **Saving the Best Model:** Serialize the best-performing model (e.g., using pickle or joblib) for future use.
  - **Integration with Web Framework:** Deploy the model using frameworks like Flask or FastAPI, enabling real-time predictions via a user-friendly web interface.

## Prerequisites:

To complete this project, you must require the following software, concepts, and packages

- Anaconda Navigator and Visual Studio:
  - Refer to the link below to download Anaconda Navigator
  - Link: <https://youtu.be/1ra4zH2G4o0>
- Python packages:
  - Open anaconda prompt as administrator
  - Type "pip install numpy" and click enter.
  - Type "pip install pandas" and click enter.
  - Type "pip install scikit-learn" and click enter.
  - Type "pip install matplotlib" and click enter.
  - Type "pip install scipy" and click enter.
  - Type "pip install pickle-mixin" and click enter.
  - Type "pip install seaborn" and click enter.
  - Type "pip install Flask" and click enter.

## Prior Knowledge:

ML Concepts:

- Supervised Learning: [Learn more about Supervised Machine Learning](#)
- Unsupervised Learning: [Learn more about Unsupervised Machine Learning](#)

Algorithms:

- Logistic Regression: [Explore Logistic Regression in Machine Learning](#)
- Decision Tree: [Learn about Decision Tree Regression in Python](#)
- Random Forest: [Understand the Random Forest Algorithm](#)
- Naive Bayes: [Study the Naive Bayes Classifier](#)

Evaluation Metrics:

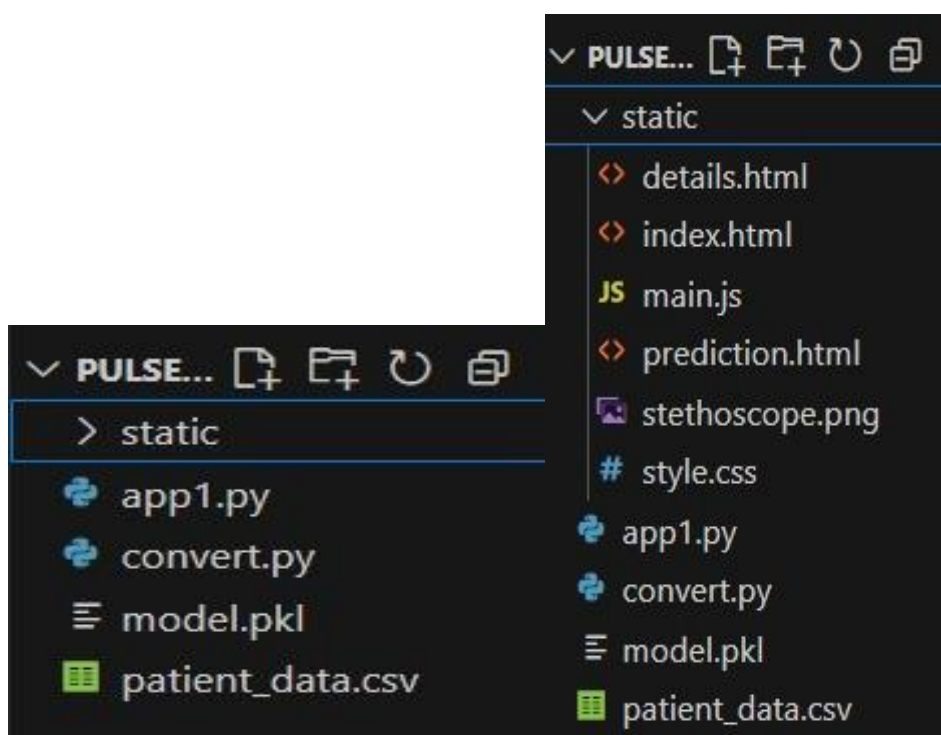
- Important Model Evaluation and Error Metrics: [Read about Model Evaluation and Error Metrics](#)

Web Framework:

- Flask Basics: [Watch Flask Basics Tutorial on YouTube](#)

## Project Structure:

Create the Project folder which contains files as shown below



- **Python files:** app1.py for backend functionality and convert.py for data conversion.
- **Model file:** model.pkl, which likely contains the trained machine learning model.
- **HTML files:** index.html, details.html, and prediction.html for user interface display.
- **JavaScript:** main.js for interactive elements in the frontend.
- **CSS and assets:** style.css for styling and an image stethoscope.png.

## **Milestone : Define Problem / Problem Understanding**

### **Activity 1: Specify the business problem**

There is a need for a predictive system that can analyze an individual's physiological, lifestyle, and historical blood pressure data to forecast potential blood pressure spikes or drops. This can enable early intervention, prevent emergencies, and improve long-term health outcomes.

The **Predictive Pulse** project aims to use **machine learning** to build an intelligent model capable of predicting future blood pressure readings and detecting abnormal trends. The model will assist healthcare providers, wellness companies, and even wearable device manufacturers in offering proactive, personalized health recommendations.

### **Activity 2: Business requirements**

- ☐ **Accurate and Reliable Predictions**
  - The ML model should be trained on recent, high-quality datasets containing blood pressure readings and related physiological/lifestyle data.
  - Accuracy and precision in detecting abnormal patterns are essential.
- ☐ **Real-Time and Batch Prediction Capability**
  - Should support live data streaming from wearable devices (e.g., smartwatches) as well as batch analysis for historical datasets.
- ☐ **Adaptability & Scalability**
  - The system should be able to adapt to new patient data and be scalable to handle millions of users.
- ☐ **Integration with Existing Health Systems**
  - APIs and interoperability features for integration with Electronic Health Records (EHR), telehealth platforms, and IoT devices.
- ☐ **Compliance and Data Privacy**
  - Must follow HIPAA (in the US), GDPR (in the EU), and other applicable data privacy regulations to ensure patient confidentiality.
- ☐ **User-Friendly Interface**
  - Dashboard for healthcare professionals with visualization tools to track patient trends.
  - Simple mobile app interface for patients to understand their health status.
- ☐ **Alert and Notification System**
  - Automatic alerts for abnormal predictions so that immediate action can be taken.

### **Activity 3: Literature Survey (Student Will Write)**

- ☐ **Existing Approaches to Blood Pressure Prediction**
  - Review research papers on ML models for cardiovascular risk prediction, such as logistic regression, random forests, gradient boosting, and deep learning models.
  - Analysis of wearable technology research for continuous blood pressure monitoring.
- ☐ **Key Datasets and Parameters**
  - Studies utilizing datasets like MIMIC-III, NHANES, or wearable device datasets containing systolic and diastolic BP, heart rate, BMI, physical activity levels, and dietary habits.
- ☐ **Strengths and Limitations of Current Models**
  - Many models perform well on static datasets but lack real-time adaptability.
  - Challenges in handling missing data, irregular intervals, and device measurement errors.
- ☐ **Gaps in Existing Research**

- Lack of personalized prediction models that adjust to individual physiological baselines.
- Limited integration with mobile/wearable platforms for mass deployment.

#### □ **Relevant Techniques**

- Feature engineering approaches for time-series health data.
- Application of ensemble models and recurrent neural networks (RNNs) for sequential prediction.
- Explainable AI (XAI) to interpret predictions for medical professionals.

### **Activity 4: Social or Business Impact.**

#### Social Impact

- **Preventative Healthcare:** Early warning system for blood pressure changes could reduce the incidence of heart attacks and strokes.
- **Improved Quality of Life:** Patients can make lifestyle adjustments proactively instead of reacting after a health crisis.
- **Empowering Patients:** Access to personal health analytics increases patient awareness and engagement in their own healthcare.

#### Business Model / Impact

- **Integration with Wearable Tech Companies:** Sell or license the predictive engine to smartwatch and fitness tracker manufacturers.
- **Telemedicine Support:** Offer hospitals and clinics subscription-based access to the model for patient monitoring.
- **Insurance Industry Value:** Insurance companies can use the system to promote preventive care and reduce claims costs.
- **Data-Driven Drug & Therapy Development:** Analysis of aggregated blood pressure trends can assist in designing targeted interventions or new medication regimens.

## **Milestone 1: Data Collection & Preparation**

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

### **Activity 1.1: Collect the dataset**

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

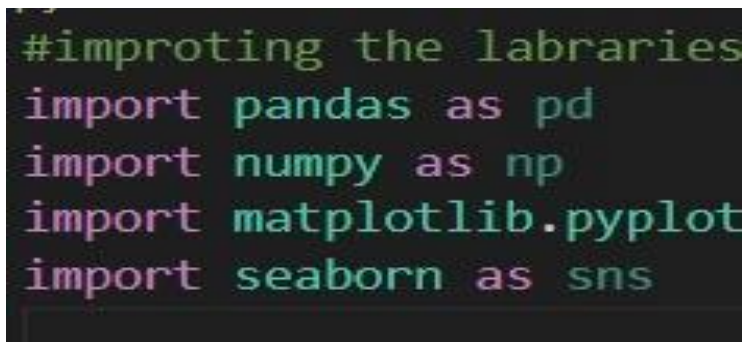
Link: <https://www.kaggle.com/datasets/bunttyshah/auto-insurance-claims-data>

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

**Note:** There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

### **Activity 1.2: Importing the libraries**

Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.



```
#improting the labraries
import pandas as pd
import numpy as np
import matplotlib.pyplot
import seaborn as sns
```

### **Activity 1.3: Read the Dataset**

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read\_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

- For checking the null values, `df.isna().any()` function is used. To sum those null values we use `.sum()` function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

`df.head()`

	C	Age	History	Patient	TakeMedication	Severity	BreathShortness	VisualChanges	NoseBleeding	Whendiagnoused	Systolic	Diastolic	Con
0	Male	18-34	Yes	No	No	Mild	No	No	No	<1 Year	111 - 120	81 - 90	
1	Female	18-34	Yes	No	No	Mild	No	No	No	<1 Year	111 - 120	81 - 90	
2	Male	35-50	Yes	No	No	Mild	No	No	No	<1 Year	111 - 120	81 - 90	
3	Female	35-50	Yes	No	No	Mild	No	No	No	<1 Year	111 - 120	81 - 90	
4	Male	51-64	Yes	No	No	Mild	No	No	No	<1 Year	111 - 120	81 - 90	

## Activity 1.4: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling Outliers

## Activity 1.5: Handling missing values

- For checking the null values, `df.isna().any()` function is used. To sum those null values we use `.sum()` function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

`df.info()`

Choose files patient\_data.csv

• patient\_data.csv(text/csv) - 166455 bytes, last modified: 30/07/2025 - 100% done  
 Saving patient\_data.csv to patient\_data.csv  
 User uploaded file "patient\_data.csv" with length 166455 bytes  
 <class 'pandas.core.frame.DataFrame'>  
 RangeIndex: 1825 entries, 0 to 1824  
 Data columns (total 14 columns):  
 # Column Non-Null Count Dtype  
 ---  
 0 Gender 1825 non-null object  
 1 Age 1825 non-null object  
 2 History 1825 non-null object  
 3 Patient 1825 non-null object  
 4 TakeMedication 1825 non-null object  
 5 Severity 1825 non-null object  
 6 BreathShortness 1825 non-null object  
 7 VisualChanges 1825 non-null object  
 8 NoseBleeding 1825 non-null object  
 9 Whendiagnoused 1825 non-null object  
 10 Systolic 1825 non-null object  
 11 Diastolic 1825 non-null object  
 12 ControlledDiet 1825 non-null object  
 13 Stages 1825 non-null object  
 dtypes: object(14)  
 memory usage: 199.7+ KB

`df.shape`  
`df.isnull().sum()`

Gender	0
Age	0
History	0
Patient	0
TakeMedication	0
Severity	0
BreathShortness	0
VisualChanges	0
NoseBleeding	0
Whendiagnoused	0
Systolic	0
Diastolic	0
ControlledDiet	0
Stages	0



## Activity 1.6: Handling Categorical Values

We have mostly all columns in our dataset, which have categorical values in it. So, we convert these categorical values into numerical values for the model to understand. There are multiple encoding techniques, we use one of them as a Label Encoder as shown below.

```
#converting categorical into numerical value
from sklearn.preprocessing import LabelEncoder

columns = ['Gender', 'Severity', 'History', 'Patient', 'TakeMedication', 'BreathShortness',
           'VisualChanges', 'NoseBleeding', 'ControlledDiet', 'Stages']

label_encoder = LabelEncoder()
for col in columns:
    df[col] = label_encoder.fit_transform(df[col])
```

Looking upon the target Stages column, there are some spelling corrections in the stages and so there are multiple stages created, so combining them into one replaces it.

```
array(['HYPERTENSION (Stage-1)', 'HYPERTENSION (Stage-2)',
       'HYPERTENSIVE CRISIS', 'HYPERTENSION (Stage-2).',
       'HYPERTENSIVE CRISI', 'NORMAL'], dtype=object)

df['Stages'].replace({'HYPERTENSIVE CRISI': 'HYPERTENSIVE CRISIS',
                     'HYPERTENSION (Stage-2).': 'HYPERTENSION (Stage-2)'},
                    inplace=True)
```

## Milestone 2: Exploratory Data Analysis

### Activity 2.1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features.

df.describe()												
	Gender	Age	History	Patient	TakeMedication	Severity	BreathShortness	VisualChanges	NoseBleeding	Whendiagnosed	Systolic	Diastol
count	1825	1825	1825	1825	1825	1825	1825	1825	1825	1825	1825	1825
unique	2	4	2	2	3	3	2	2	3	3	5	5
top	Female	51-64	Yes	No	No	Moderate	No	No	No	<1 Year	111 - 120	81 - 90
freq	913	475	1657	984	744	697	976	940	984	625	1008	744

### Activity 2.2: Visual analysis

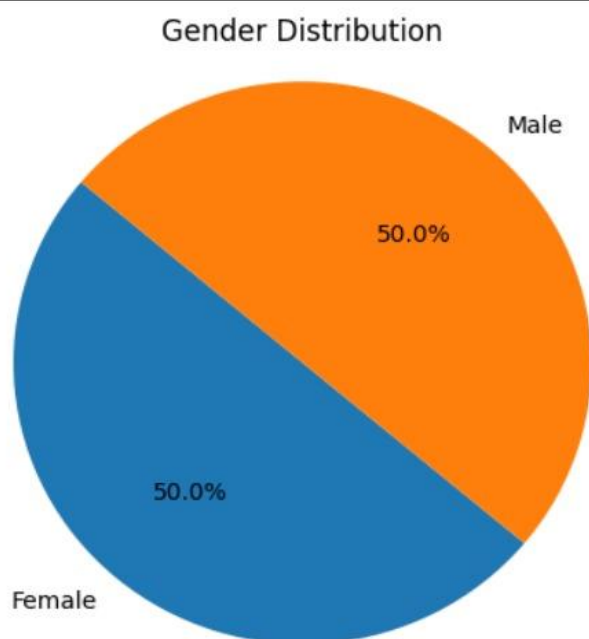
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

### Activity 2.3: Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as Piechart and countplot.

Seaborn package provides a wonderful function countplot. It is more useful for categorical features. With the help of countplot, we can Number of unique values in the feature. From the countplot we can say that there are only 247 fraud cases reported in 1000 insurance claims.

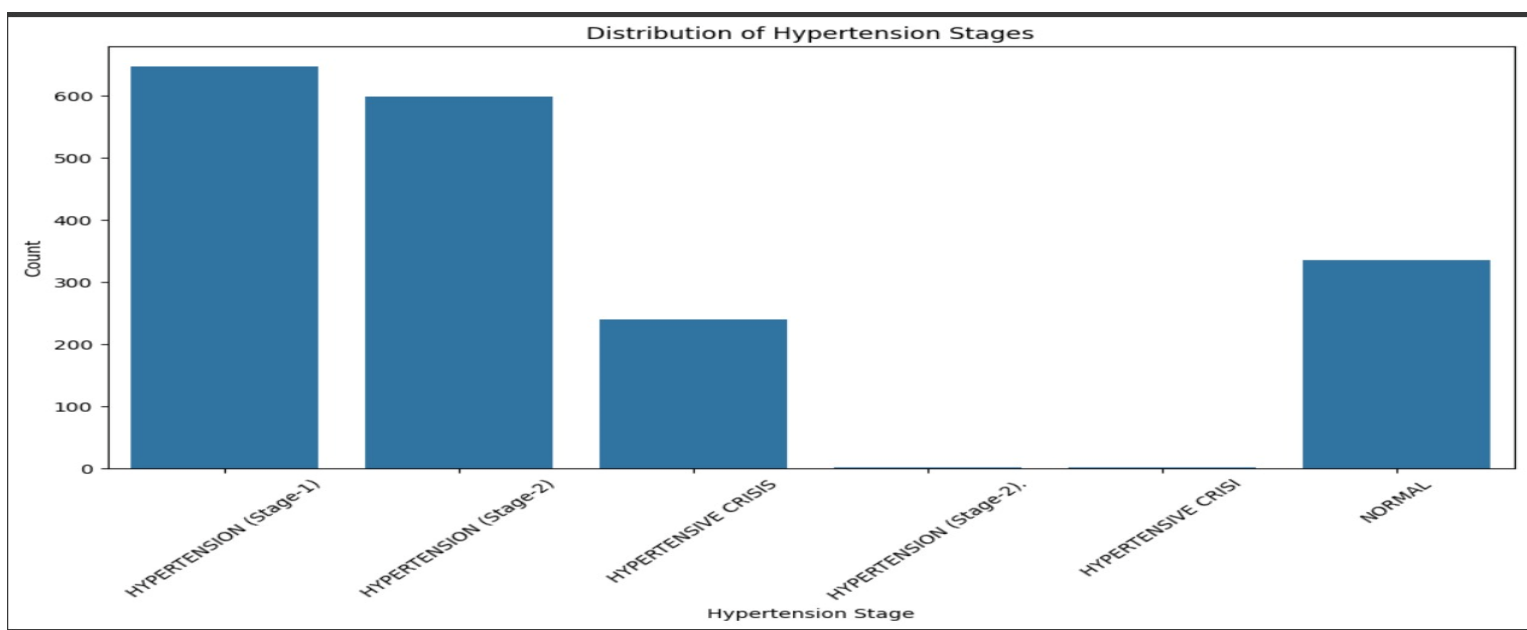
```
gender_counts = df['Gender'].value_counts()
plt.pie(gender_counts, labels=gender_counts.index, autopct='%1.1f%%', startangle=140)
plt.title('Gender Distribution')
plt.axis('equal')
plt.show()
```



The pie chart visualizes the **gender distribution** in the dataset using matplotlib. It shows an equal split:

- **50% Male**
- **50% Female**

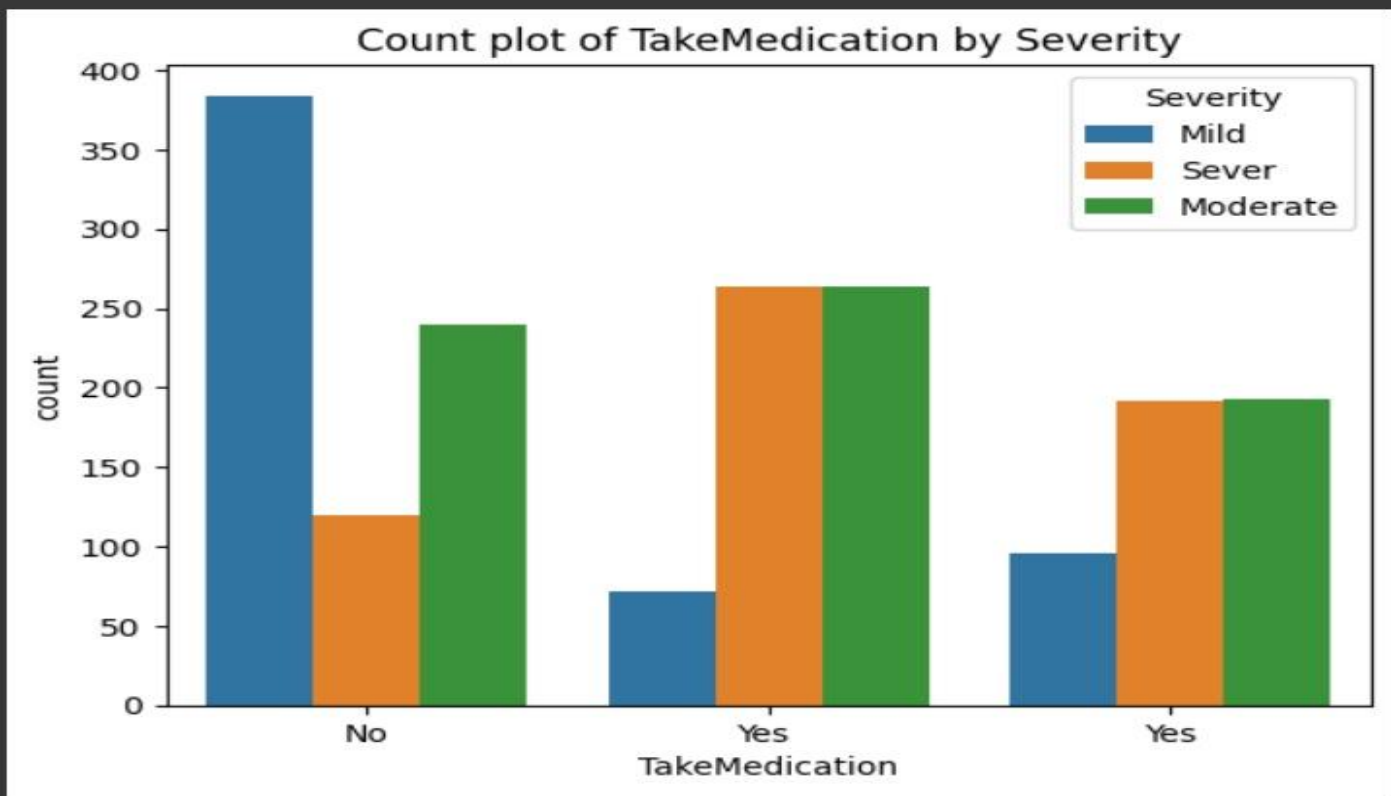
The Python code uses `value_counts()` to count occurrences of each gender, then plots those using `plt.pie()` with percentage labels. This helps in understanding the gender balance in the data, which is crucial for building unbiased machine learning models.



## Activity 2.4: Bivariate analysis

To find the relation between two features we use bivariate analysis. Here we can use barplot.

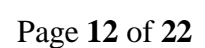
```
sns.countplot(x='TakeMedication',hue='Severity',data=df)
plt.title('Count plot of TakeMedication by Severity')
plt.show()
```



The relationship between 'Take Medication' and 'Severity' suggests a potential correlation between medication adherence and the severity of the medical condition. Analyzing the data might reveal whether patients who take their medication regularly tend to experience lower severity levels compared to those who do not. Understanding this relationship could inform healthcare professionals about the effectiveness of prescribed medications in managing the condition and the importance of adherence to treatment protocols for better outcomes.

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used heatmap from seaborn package.

✓ 1m 31.9s



## Splitting data into train and test

Now let's split the Dataset into train and test sets. First, split the dataset into x and y and then split the data set. Here x and y variables are created. On the x variable, df is passed by dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using the train\_test\_split() function from sklearn. As parameters, we are passing x, y, test\_size, random\_state.

```
#Splitting the data into X and Y
X = df.drop('Stages' , axis = 1)
X
```

Python

	Gender	Age	Patient	Severity	BreathShortness	VisualChanges	NoseBleeding	Whendiagnoused	Systolic	Diastolic
0	1	0	0	0	0	0	1	0	0.0	0.000077
1	0	0	0	0	0	0	1	0	0.0	0.000077
2	1	1	0	0	0	0	1	0	0.0	0.000077
3	0	1	0	0	0	0	1	0	0.0	0.000077
4	1	2	0	0	0	0	1	0	0.0	0.000077
...	...	...	...	...	...	...	...	...	...	...
1820	0	1	0	2	0	0	1	6	0.0	0.000000
1821	1	2	0	2	0	0	1	6	0.0	0.000000
1822	0	2	0	2	0	0	1	6	0.0	0.000000
1823	1	3	0	2	0	0	1	6	0.0	0.000000
1824	0	3	0	2	0	0	1	6	0.0	0.000000

1825 rows × 11 columns

```
Y = df['Stages']
Y
```

```
0      0
1      0
2      0
3      0
4      0
..
1820    3
1821    3
1822    3
1823    3
1824    3
Name: Stages, Length: 1825, dtype: int32
```

## Milestone 3: Model Building

### Activity 3.1: Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying five classification algorithms. The best model is saved based on its performance.

### Activity 3.2: Logistic Regression model

Logistic Regression Model is imported from sklearn Library then Logistic Regression algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
import pandas as pd
df = pd.read_csv('/patient_data.csv')
x = df.drop('Stages', axis=1)
y = df['Stages']
x = pd.get_dummies(x, drop_first=True)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=30)
lr_model = LogisticRegression()
lr_model.fit(x_train, y_train)
y_pred = lr_model.predict(x_test)
acc_lr = accuracy_score(y_test, y_pred)
c_lr = classification_report(y_test, y_pred)
print("Accuracy:", acc_lr)
print(c_lr)
```

Accuracy: 1.0

	precision	recall	f1-score	support
HYPERTENSION (Stage-1)	1.00	1.00	1.00	132
HYPERTENSION (Stage-2)	1.00	1.00	1.00	133
HYPERTENSIVE CRISIS	1.00	1.00	1.00	34
NORMAL	1.00	1.00	1.00	66
accuracy			1.00	365
macro avg	1.00	1.00	1.00	365
weighted avg	1.00	1.00	1.00	365

### Activity 3.3: Random forest model

First Random Forest Model is imported from sklearn Library then RandomForestClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. We can find the Train and Test accuracy by X\_train and X\_test.



```

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Fit Random Forest model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict and evaluate
predictions = rf.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
report = classification_report(y_test, predictions, target_names=le.classes_)

print("Random Forest Model Accuracy:", accuracy)
print("Classification Report:\n", report)

```

	precision	recall	f1-score	support
HYPERTENSION (Stage-1)	1.00	1.00	1.00	170
HYPERTENSION (Stage-2)	1.00	1.00	1.00	151
HYPERTENSIVE CRISIS	1.00	1.00	1.00	59
NORMAL	1.00	1.00	1.00	77
accuracy			1.00	457
macro avg	1.00	1.00	1.00	457
weighted avg	1.00	1.00	1.00	457

### Activity 3.4: Decision Tree Model

A function named `decision_tree_model` is created and train and test data are passed as the parameters. Inside the function, the Decision Classifier algorithm is initialized and training data is passed to the model with the `.fit()` function. Test data is predicted with the `.predict()` function and saved in a new variable. For evaluating the model, an accuracy score and classification report are used.

```

from sklearn.tree import DecisionTreeClassifier
Decision_tree_model = DecisionTreeClassifier()
Decision_tree_model.fit(x_train, y_train)
y_pred = Decision_tree_model.predict(x_test)
acc_dt = accuracy_score(y_test, y_pred)
c_dt = classification_report(y_test, y_pred)
print("Accuracy:", acc_dt)
print(c_dt)

```

```

Accuracy: 1.0

```

	precision	recall	f1-score	support
HYPERTENSION (Stage-1)	1.00	1.00	1.00	132
HYPERTENSION (Stage-2)	1.00	1.00	1.00	133
HYPERTENSIVE CRISIS	1.00	1.00	1.00	34
NORMAL	1.00	1.00	1.00	66
accuracy			1.00	365
macro avg	1.00	1.00	1.00	365
weighted avg	1.00	1.00	1.00	365



### Activity 3.5: Gaussian Navies Bayes

A variable named NB is created and train and test data are passed as the parameters. Inside the function, the Gaussian Navies Bayes algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in the new variable. For evaluating the model, accuracy score and classification report are used.

```
from sklearn.naive_bayes import GaussianNB
NB = GaussianNB()
NB.fit(x_train, y_train)
y_pred = NB.predict(x_test)
acc_nb = accuracy_score(y_test, y_pred)
c_nb = classification_report(y_test, y_pred)
print("Accuracy:", acc_nb)
print(c_nb)
```

Accuracy: 1.0

	precision	recall	f1-score	support
HYPERTENSION (Stage-1)	1.00	1.00	1.00	132
HYPERTENSION (Stage-2)	1.00	1.00	1.00	133
HYPERTENSIVE CRISIS	1.00	1.00	1.00	34
NORMAL	1.00	1.00	1.00	66
accuracy			1.00	365
macro avg	1.00	1.00	1.00	365
weighted avg	1.00	1.00	1.00	365

### Activity 3.6: Multinomial Navies Bayes

A function named ada\_model is created and train and test data are passed as the parameters. Inside the function, the Multinomial Navies Bayes algorithm is initialized and training data is passed to the model with the fit() function. Test data is predicted with the .predict() function and saved in a new variable. For evaluating the model, an accuracy score and classification report are used.

```
from sklearn.naive_bayes import MultinomialNB
mNB = MultinomialNB()
mNB.fit(x_train, y_train)
y_pred = mNB.predict(x_test)
acc_mnb = accuracy_score(y_test, y_pred)
c_mnb = classification_report(y_test, y_pred)
print("Accuracy:", acc_mnb)
print(c_mnb)
```

Accuracy: 0.9972602739726028

	precision	recall	f1-score	support
HYPERTENSION (Stage-1)	1.00	0.99	1.00	132
HYPERTENSION (Stage-2)	1.00	1.00	1.00	133
HYPERTENSIVE CRISIS	1.00	1.00	1.00	34
NORMAL	0.99	1.00	0.99	66
accuracy			1.00	365
macro avg	1.00	1.00	1.00	365
weighted avg	1.00	1.00	1.00	365

### Activity 3.7: Testing the model

Here we have tested with Decision Tree algorithm. You can test with all algorithm. With the help of predict() function.

```
prediction = Decision_tree_model.predict(x_test)
print(prediction)
```

```
[ 'HYPERTENSION (Stage-1)' 'HYPERTENSION (Stage-2)'
  'HYPERTENSION (Stage-1)' 'HYPERTENSION (Stage-2)' 'HYPERTENSIVE CRISIS'
  'HYPERTENSION (Stage-2)' 'HYPERTENSION (Stage-2)'
  'HYPERTENSION (Stage-2)' 'NORMAL' 'NORMAL' 'HYPERTENSION (Stage-1)'
  'HYPERTENSION (Stage-2)' 'HYPERTENSION (Stage-2)'
  'HYPERTENSION (Stage-2)' 'HYPERTENSIVE CRISIS' 'NORMAL'
  'HYPERTENSION (Stage-1)' 'HYPERTENSION (Stage-2)'
  'HYPERTENSION (Stage-1)' 'HYPERTENSION (Stage-1)' 'NORMAL'
  'HYPERTENSION (Stage-1)' 'HYPERTENSION (Stage-2)'
  'HYPERTENSION (Stage-2)' 'HYPERTENSION (Stage-1)'
  'HYPERTENSION (Stage-2)' 'NORMAL' 'HYPERTENSION (Stage-1)'
  'HYPERTENSION (Stage-2)' 'HYPERTENSIVE CRISIS' 'HYPERTENSION (Stage-1)'
  'HYPERTENSION (Stage-1)' 'NORMAL' 'NORMAL' 'HYPERTENSIVE CRISIS' 'NORMAL'
  'HYPERTENSION (Stage-2)' 'NORMAL' 'HYPERTENSION (Stage-1)'
  'HYPERTENSION (Stage-1)' 'HYPERTENSION (Stage-1)'
  'HYPERTENSION (Stage-2)' 'HYPERTENSION (Stage-2)'
  'HYPERTENSION (Stage-2)' 'HYPERTENSION (Stage-2)'
  'HYPERTENSION (Stage-1)' 'HYPERTENSION (Stage-2)'
  'HYPERTENSION (Stage-2)' 'HYPERTENSION (Stage-2)' 'NORMAL'
  'HYPERTENSION (Stage-1)' 'HYPERTENSION (Stage-2)'
  'HYPERTENSION (Stage-2)' 'HYPERTENSIVE CRISIS' 'HYPERTENSION (Stage-1)'
  'NORMAL' 'HYPERTENSION (Stage-1)' 'HYPERTENSION (Stage-2)'
  'HYPERTENSION (Stage-2)' 'HYPERTENSION (Stage-1)' 'NORMAL'
  'HYPERTENSION (Stage-1)' 'HYPERTENSION (Stage-1)'
  'HYPERTENSION (Stage-2)' 'HYPERTENSION (Stage-1)'
  'HYPERTENSION (Stage-2)' 'NORMAL' 'HYPERTENSION (Stage-1)'
```

## Milestone 4: Performance Testing & Hyperparameter Tuning

### Activity 4.1: Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for regression tasks including Accuracy scores and classification reports.

### Compare the model

For comparing the above four models, the compareModel function is defined.

```
rf_model = RandomForestClassifier()
rf_model.fit(x_train, y_train)
y_pred_rf = rf_model.predict(x_test)
acc_rf = accuracy_score(y_test, y_pred_rf)
c_rf = classification_report(y_test, y_pred_rf)
# Update the model comparison DataFrame
model = pd.DataFrame({
    'Model': [
        'Linear Regression',
        'Decision Tree Classifier',
        'RandomForest Classifier',
        'Gaussian Navies Bayes',
        'Multinomial Navies Bayes'
    ],
    'Score': [
        acc_lr,
        acc_dt,
        acc_rf,
        acc_nb,
        acc_mnb
    ],
})
print(model)
model = pd.DataFrame({'Model':['Linear Regression','Decision Tree Classifier','RandomForest Classifier',
                                'Gaussian Navies Bayes','Multinomial Navies Bayes'],
                      'Score':[acc_lr,acc_dt,acc_rf,acc_nb,acc_mnb],
                      })
```

✓ 0.1s

	Model	Score
0	Linear Regression	1.00000
1	Decision Tree Classifier	1.00000
2	RandomForest Classifier	1.00000
3	Gaussian Navies Bayes	1.00000
4	Multinomial Navies Bayes	0.99726

## **Milestone 5: Model Deployment**

### **Activity 5.1: Save the best model**

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

### **Activity 5.2: Integrate with Web Framework**

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the user where he has to enter the values for predictions. The entered values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the web application

#### **Activity 5.2: Building Html Page:**

For this project create HTML file namely

- index.html
- details.html
- prediction.html

and save them in the static folder. Refer this [link](#) for statics.

#### **Activity 5.3: Build Python code:**

Import the libraries

```
import pickle
import warnings
pickle.dump(random_forest, open("model.pkl", "wb"))
```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module ( name ) as argument.

- -



```

import numpy as np
import pandas as pd
from flask import Flask, request, jsonify, send_from_directory
import pickle

app = Flask(__name__, static_folder='static', static_url_path='/static')

# Load your pickled model and label encoder (model.pkl contains a tuple: (model, label_encoder))
model, label_encoder = pickle.load(open('model.pkl', 'rb'))

```

Render HTML page:

```

@app.route('/')
def home():
    return send_from_directory('static', 'index.html')

@app.route('/details')
def details():
    return send_from_directory('static', 'details.html')

@app.route('/prediction')
def prediction():
    return send_from_directory('static', 'prediction.html')

@app.route('/api/predict', methods=['POST'])
def api_predict():

```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```

# Extract and convert inputs
Gender = int(data.get("Gender"))
Age = float(data.get("Age"))
Patient = int(data.get("Patient"))
Severity = int(data.get("Severity"))
BreathShortness = int(data.get("BreathShortness"))
VisualChanges = int(data.get("VisualChanges"))
NoseBleeding = int(data.get("NoseBleeding"))
Whendiagnosed = int(data.get("Whendiagnosed"))
Systolic = float(data.get("Systolic"))
Diastolic = float(data.get("Diastolic"))
ControlledDiet = int(data.get("ControlledDiet"))

# Arrange data into dataframe with proper columns
features = np.array([[Gender, Age, Patient, Severity, BreathShortness,
                        VisualChanges, NoseBleeding, Whendiagnosed,
                        Systolic, Diastolic, ControlledDiet]])

df = pd.DataFrame(features, columns=[
    'Gender', 'Age', 'Patient', 'Severity',
    'BreathShortness', 'VisualChanges', 'NoseBleeding',
    'Whendiagnosed', 'Systolic', 'Diastolic', 'ControlledDiet'
])
df.rename(columns={'Whendiagnosed': 'Whendiagnoused'}, inplace=True)

# Predict class
pred_class = model.predict(df)[0]
pred_label = label_encoder.inverse_transform([pred_class])[0]

return jsonify({'prediction': f"Your Blood Pressure is {pred_label}"})

except Exception as e:
    return jsonify({'error': str(e)}), 400

```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

```
if __name__ == '__main__':  
    # Run on localhost port 5500  
    app.run(debug=True, host='127.0.0.1', port=5500)
```

#### Activity .5.4: Run the web application

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
PS C:\Users\pc\OneDrive\Desktop\pulse-prediction system> python app1.py  
* Serving Flask app 'app1'  
* Debug mode: on  
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
* Running on http://127.0.0.1:5500  
Press CTRL+C to quit  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 707-293-310
```

Now, Go the web browser and write the localhost url (<http://127.0.0.1:5500>) to get the below result

## PREDICTIVE PULSE

[Home](#) [About](#)

[Check your Blood Pressure](#)

# HEALTHCARE

### Welcome to Predictive Pulse

Our platform helps you monitor blood pressure and understand the associated stages and risks. Discover your status and proactive steps toward better health.

INSERT TEXT HERE

### INFORMATION.

Blood pressure (BP) is the force of blood pushing against artery walls. Maintaining healthy BP levels is crucial for cardiovascular health, and helps prevent diseases like heart attack and stroke.

### INFORMATION.

Blood pressure (BP) is the force of blood pushing against artery walls. Maintaining healthy BP levels is crucial for cardiovascular health, and helps prevent diseases like heart attack and stroke.

#### Stages

- Stage 1: **Normal** – Systolic < 120 mm Hg, Diastolic < 80 mm Hg
- Stage 2: **Elevated** – Systolic 120–129 mm Hg, Diastolic < 80 mm Hg
- Stage 3: **Hypertension Stage 1** – Systolic 130–139 mm Hg, Diastolic 80–89 mm Hg
- Stage 4: **Hypertension Stage 2** – Systolic ≥ 140 mm Hg, Diastolic ≥ 90 mm Hg

#### Risk Factors:

- Family history
- High salt, low potassium diets
- Lack of physical activity
- Excessive alcohol consumption
- Chronic stress
- Increases with age



# Predict Blood Pressure

**Gender (0 = Female, 1 = Male):**

**Age:**

**Patient History (0 = No, 1 = Yes):**

**Severity (1 = Mild, 2 = Moderate, 3 = Severe):**

**Breath Shortness (0 = No, 1 = Yes):**

**Visual Changes (0 = No, 1 = Yes):**

**Nose Bleeding (0 = No, 1 = Yes):**

**When Diagnosed (years ago):**

**Systolic (mm Hg):**

**Diastolic (mm Hg):**

**Controlled Diet (0 = No, 1 = Yes):**

**Predict**

**Your Blood Pressure is HYPERTENSION (Stage-1)**

**Go Home**

**Model Details**

## How Does the Model Work?

Our system analyzes your blood pressure and related health indicators using a robust machine learning model. You provide your measurements and symptoms, and the model classifies your blood pressure into recognized clinical stages.

- **Inputs considered:** Gender, age, patient history, symptom severity, breath shortness, visual changes, nose bleeding, diagnosis period, systolic & diastolic values, controlled diet.
- **Stages:**
  - 0 – Normal
  - 1 – Hypertension (Stage 1)
  - 2 – Hypertension (Stage 2)
  - 3 – Hypertension Crisis
- **Disclaimer:** This is a screening tool. For a medical diagnosis, consult your doctor.

[Go Home](#)[Blood Pressure Prediction](#)

