Why We Must Address Cardiovascular Health?

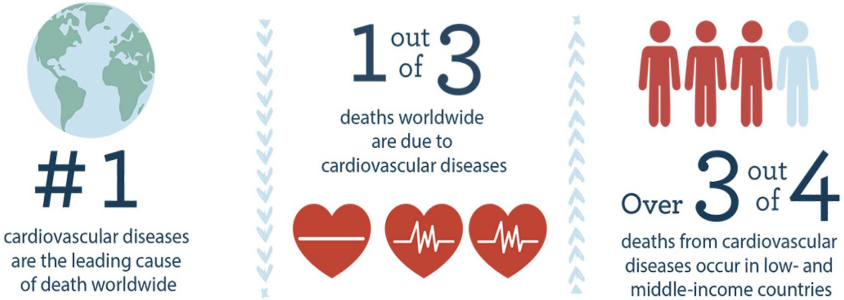
Cardiovascular disease are a group of blood and heart disorders that can lead to heart attack and stroke.

The Escalating Challenge of Global Health Inequities:
<u>Low and Middle income countries face:</u>

- Double burden of communicable and non communicable diseases.
- Limited access to effective and equitable health care services.
- Delayed detection and treatment for diseases.

<u>These conditions can lead to:</u>

- Overburdened, less resilient health systems.
- High productivity losses from premature death and disability.
- Strained economic development



Cardiovascular Health: Statistics and Projections

- 17.9 million people died from CVDs in 2019, representing 32% of all global deaths. Of these deaths, 85% were due to heart attack and stroke.
- Out of the 17 million premature deaths (under the age of 70) due to noncommunicable diseases in 2019, 38% were caused by CVDs.
- Key risk factors for CVDs include hypertension (Globally affecting 1.13 billion people), smoking (Globally 1 billion smokers), and obesity (Globally 650 million obese adults).
- Highest CVD deaths are found in Central and Eastern Europe, and the lowest in high-income Asian countries.
- An estimated $320 billion is spent on CVDs in the United States alone each year, not counting medical costs and lost productivity.

References: CDC[1], WHO [2]

Stroke Prediction Dataset

Dataset: The Stroke Prediction Dataset is taken from Kaggle. It contains 5000 patient records, each with 11 health related attributes. It includes a range of health related variables that are significant risk factors for stroke.

Dataset Attributes: Gender, Age, Hypertension, Heart Disease, Ever Married, Work Type, Residence Type, Average Glucose Level, BMI, Smoking Status, Stroke.

Objective: The dataset is used to train and test predictive machine learning models to predict stroke risk based on health indicators and lifestyle.

Database Creation

Parsing the Data - Code

```python
# Read and parse data
rows = []
with open(datafile, "r") as f:
    for line in f:
        rows.append(line.strip().split(","))
header, data = rows[0], rows[1:]


# Data to insert
patient_list = [(int(i[0]), i[2], i[1], i[5], i[6], i[7]) for i in data]
health_dets = [(int(i[0]), i[4], i[3], i[-4], i[-3], i[-2]) for i in data]
strokes = [(int(i[0]), int(i[-1])) for i in data]
```

Normalization

- Normalized the database by dividing into Patient Demographics, Health Details, and Stroke incidence.
- All are connected through PatientID.
- This type of dividing minimizes data overlap and maintains consistency.
- Scalable design ensures robustness.
- Enables fast and accurate queries, enhancing the predictive performance.

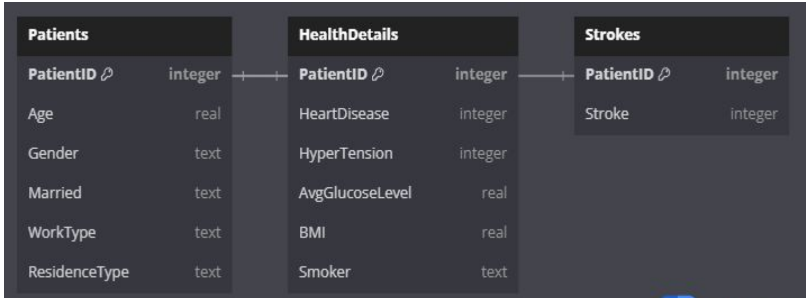


Table Creation - Code

```python
# Create tables
create_patients_table_sql = """
CREATE TABLE IF NOT EXISTS [Patients](
    [PatientID] INTEGER NOT NULL PRIMARY KEY,
    [Age] REAL,
    [Gender] TEXT,
    [Married] TEXTL,
    [WorkType] TEXT,
    [ResidenceType] TEXT);"""

create_healthdetails_table_sql = """
CREATE TABLE IF NOT EXISTS [HealthDetails](
    [PatientID] INTEGER NOT NULL PRIMARY KEY,
    [HeartDisease] INTEGER,
    [HyperTension] INTEGER,
    [AvgGlucoseLevel] REAL,
    [BMI] REAL,
    [Smoker] TEXT,
    FOREIGN KEY(PatientID) REFERENCES Patients(PatientID));
"""

create_strokes_table_sql = """
CREATE TABLE IF NOT EXISTS [Strokes](
    [PatientID] INTEGER NOT NULL PRIMARY KEY,
    [Stroke] INTEGER NOT NULL,
    FOREIGN KEY(PatientID) References Patients(PatientID));
"""
```

```python
# Insert statements
insert_patients = """
INSERT INTO Patients(
    PatientID,
    Age,
    Gender,
    Married,
    WorkType,
    ResidenceType) VALUES (?, ?, ?, ?, ?, ?)"""

insert_healthdetails = """
INSERT INTO HealthDetails(
    PatientID,
    HeartDisease,
    HyperTension,
    AvgGlucoseLevel,
    BMI,
    Smoker) VALUES (?, ?, ?, ?, ?, ?)"""

insert_strokes = "INSERT INTO Strokes(PatientID, Stroke) VALUES (?, ?)"

# Create tables and insert values
with conn:
    cur = conn.cursor()
    create_table(conn, create_patients_table_sql, drop_table_name = "Patients")
    create_table(conn, create_healthdetails_table_sql, drop_table_name = "HealthDetails")
    create_table(conn, create_strokes_table_sql, drop_table_name = "Strokes")

    cur.executemany(insert_patients, patient_list)
    cur.executemany(insert_healthdetails, health_dets)
    cur.executemany(insert_strokes, strokes)

conn.close()
```

Joining the Tables

- To create a comprehensive dataset for our stroke prediction model, we combined the three tables on PatientID.
- We did not select every column from all tables; chose to drop Married and WorkType. Felt like they wouldn’t be useful in the model.

```
# Join data into one dataframe
join_statement = """
SELECT
    p.PatientID, Age, Gender,
    ResidenceType, HeartDisease, HyperTension,
    AvgGlucoseLevel, BMI, Smoker,
    Stroke
FROM Patients AS p
INNER JOIN HealthDetails AS hd
ON p.PatientID = hd.PatientID
INNER JOIN Strokes AS s
ON s.PatientID = p.PatientID
"""

data = pd.read_sql_query(join_statement, conn)
```

Analysis of the Data

Preparing Data for Analysis

- "BMI" column had missing values, represented as the string "N/A"
  - Replaced these missing values with gender-specific average BMI, depending on the patient's gender.
- "Smoker" column had values of "Unknown"
  - Kept them as-is - could not reasonably drop these rows from the data and could not fill them in
- One entry whose gender was "Other"
  - Dropped from the dataset

Data Cleaning - Code

```python
data["BMI"] = data["BMI"].apply(lambda x: np.nan if x == "N/A" else x) # Convert "N/A" to NaN

bmi_female, bmi_male, bmi_other = data[["Gender", "BMI"]].groupby("Gender").mean()["BMI"] # Gender-specific averages
bmi_female, bmi_male, bmi_other
```

```python
# Replace BMI missing values with gender-specific BMI averages
data["BMI"] = np.where((np.isnan(data["BMI"])) & (data["Gender"] == "Female"), bmi_female, data["BMI"])
data["BMI"] = np.where((np.isnan(data["BMI"])) & (data["Gender"] == "Male"), bmi_male, data["BMI"])
```

```python
data.loc[data["Gender"] == "Other"]
```

	PatientID	Age	Gender	ResidenceType	HeartDisease	HyperTension	AvgGlucoseLevel	BMI	Smoker	Stroke
3926	56156	26.0	Other	Rural	0	0	143.33	22.4	formerly smoked	0

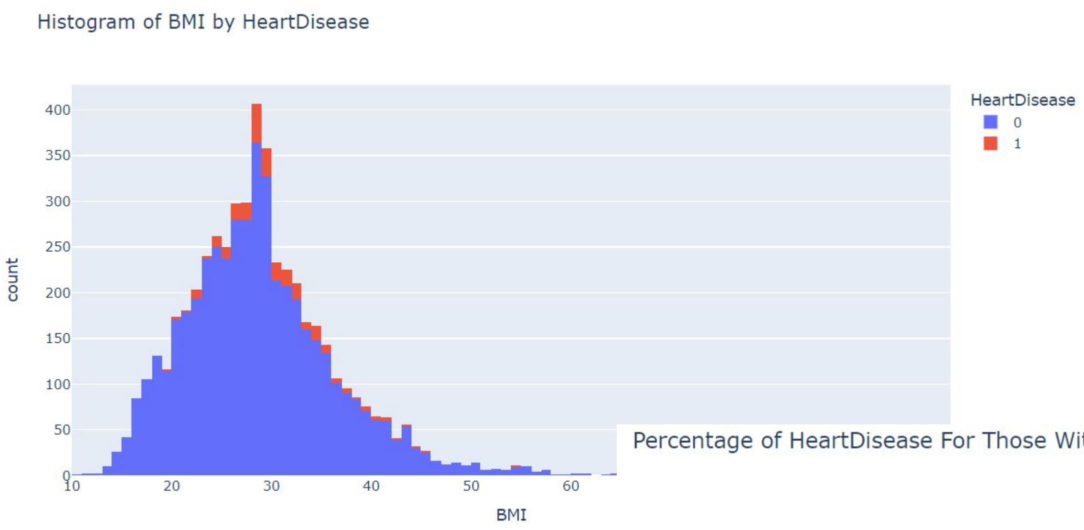
Analysis

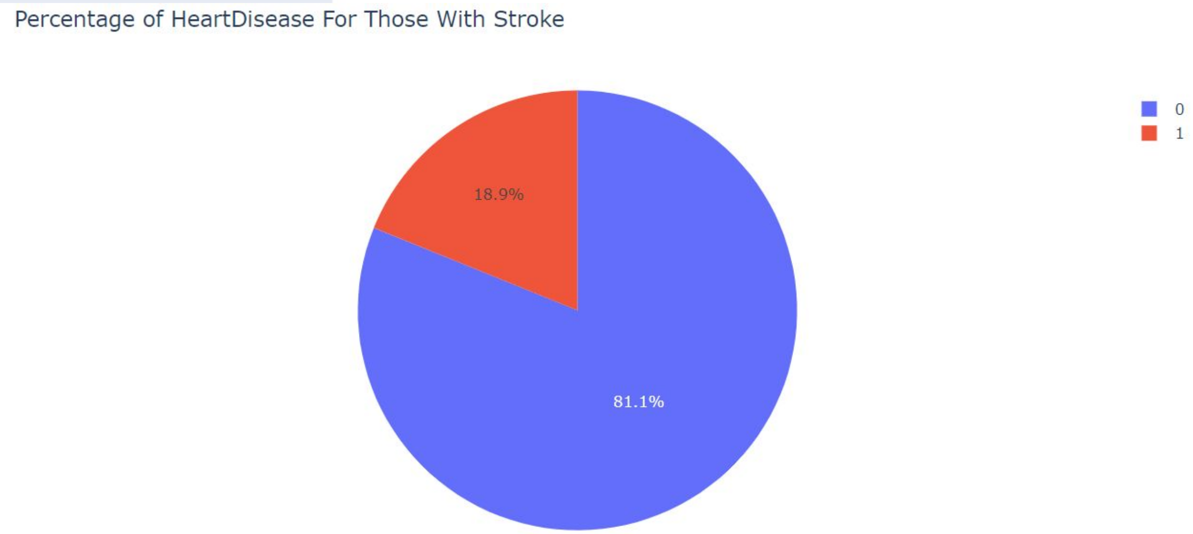
	Age	AvgGlucoseLevel	BMI
count	5109.000000	5109.000000	5109.000000
mean	43.229986	106.140399	28.892790
std	22.613575	45.285004	7.698351
min	0.080000	55.120000	10.300000
25%	25.000000	77.240000	23.800000
50%	45.000000	91.880000	28.400000
75%	61.000000	114.090000	32.800000
max	82.000000	271.740000	97.600000

	Stroke	Age
0	0	41.974831
1	1	67.728193

	Stroke	BMI
0	0	28.825118
1	1	30.213621

	Stroke	AvgGlucoseLevel
0	0	104.787584
1	1	132.544739


Histogram of BMI by HeartDisease
HeartDisease
0
1
count
400
350
300
250
200
150
100
50
0
10
20
30
40
50
60
BMI


Percentage of HeartDisease For Those With Stroke
0
1
18.9%
81.1%

Applying Machine Learning Algorithms

Data Preprocessing

- We encoded categorical variables as numeric variables (Gender, ResidenceType, Smoker)
- Use backward elimination method to select best 4 features for logistic regression model and k-NN classifier
- Split the data into train and test set (80-20 split) to balance between model training and validation capabilities.
- StandardScaler fitted to numerical variables on the train set and used to transform the test set for optimal model training.

Preprocessing - Code

```
# First need to encode categorical variables as numerical
enc = preprocessing.OrdinalEncoder()
enc.fit(data[["Gender", "ResidenceType", "Smoker"]])
transformed_categoricals = enc.transform(data[["Gender", "ResidenceType", "Smoker"]])

# Gender: 0 -> Female, 1 -> Male
# ResidenceType: 0 -> Rural, 1 -> Urban
# Smoker: 0 -> Unknown, 1 -> Formerly smoked, 2 -> Never smoked, 3 -> Smokes

data[["Gender", "ResidenceType", "Smoker"]] = transformed_categoricals
```

Encoding categorical variables

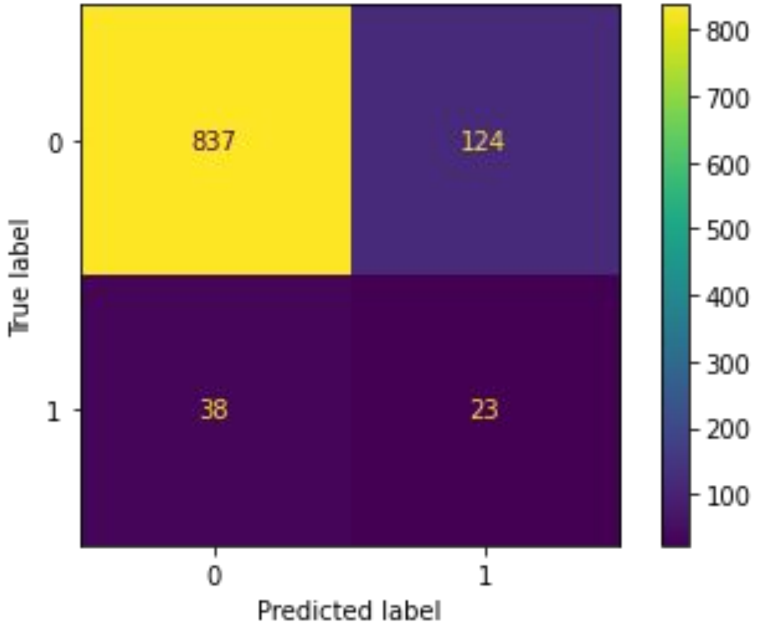
```
# Select features for a logistic regression model
log_reg = LogisticRegression(max_iter = 400, class_weight = "balanced") # use balanced class weight because imbalanced data
sfs_lr = SequentialFeatureSelector(log_reg, direction = "backward", n_features_to_select=4)
sfs_lr.fit(X, y)
```

Backward elimination

```
# Scale data
scaler = preprocessing.StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

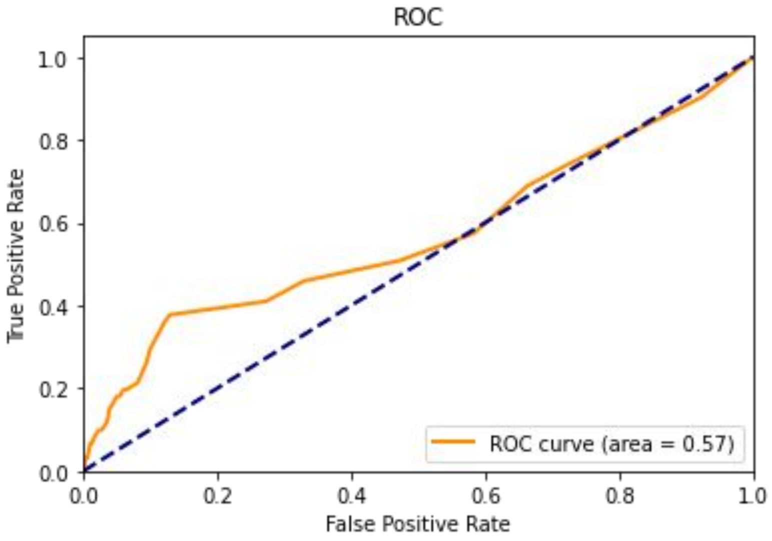
Logistic Regression Model: Building and Evaluating the Model

- Features chosen were Gender, HeartDisease, HyperTension, and Smoker.
- Used class_weight = “balanced” because dataset is unbalanced.
- F1 score of about **0.22**

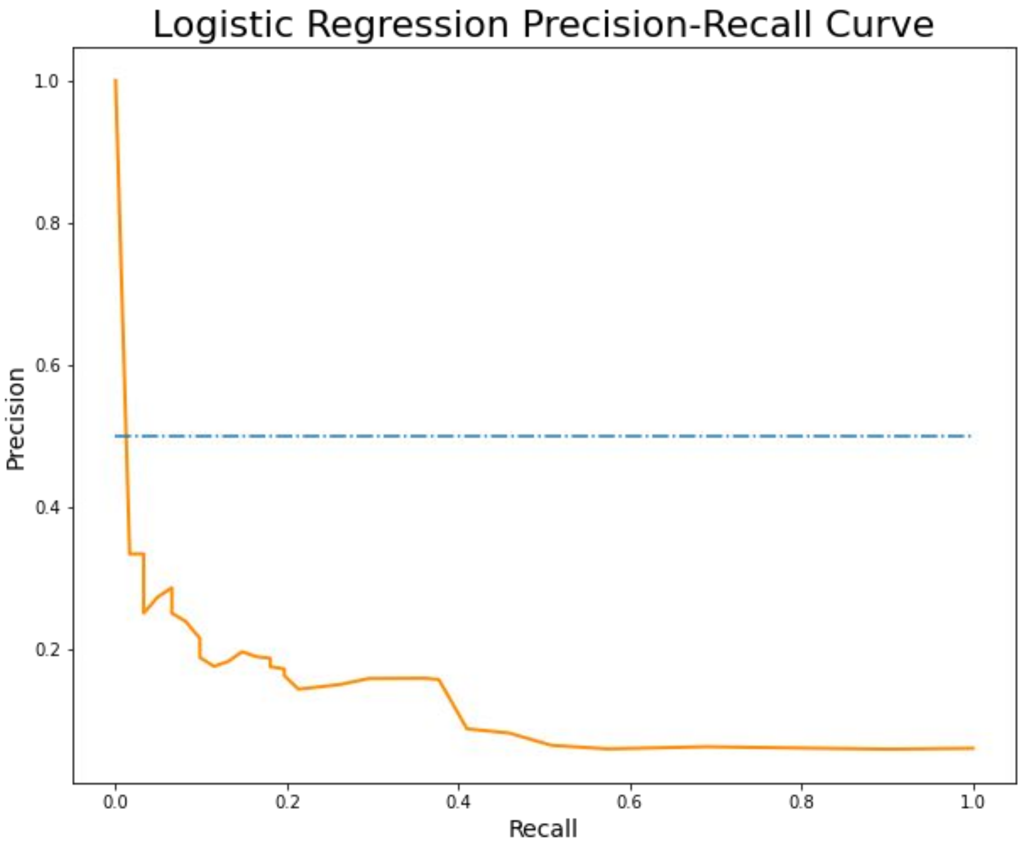


	precision	recall	f1-score	support
0	0.96	0.87	0.91	961
1	0.16	0.38	0.22	61
accuracy			0.84	1022
macro avg	0.56	0.62	0.57	1022
weighted avg	0.91	0.84	0.87	1022

Our logistic regression model has high accuracy, but poor performance on predicting a stroke (1).

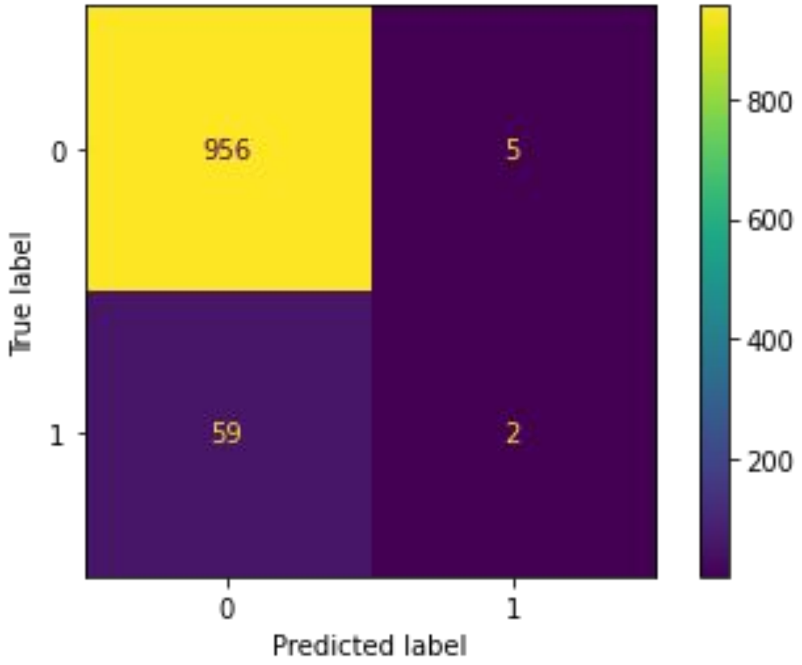


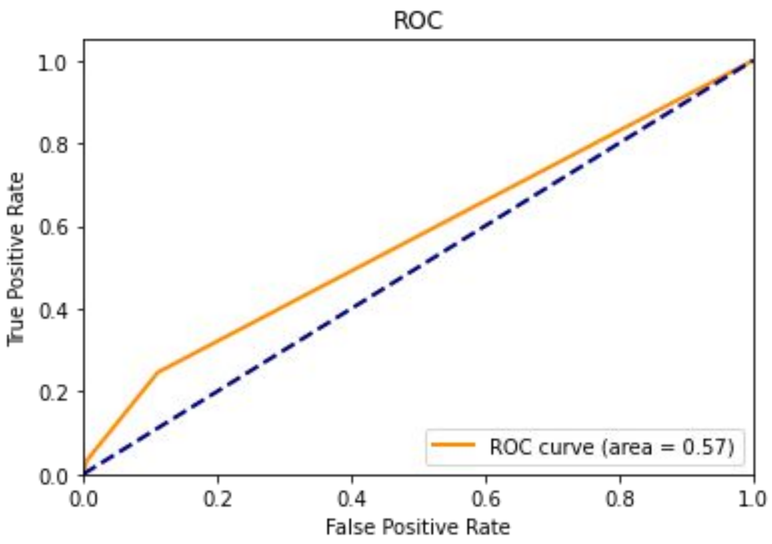
- Area under ROC curve is about **0.57**.
- Area under Precision-Recall curve is about **0.11**.

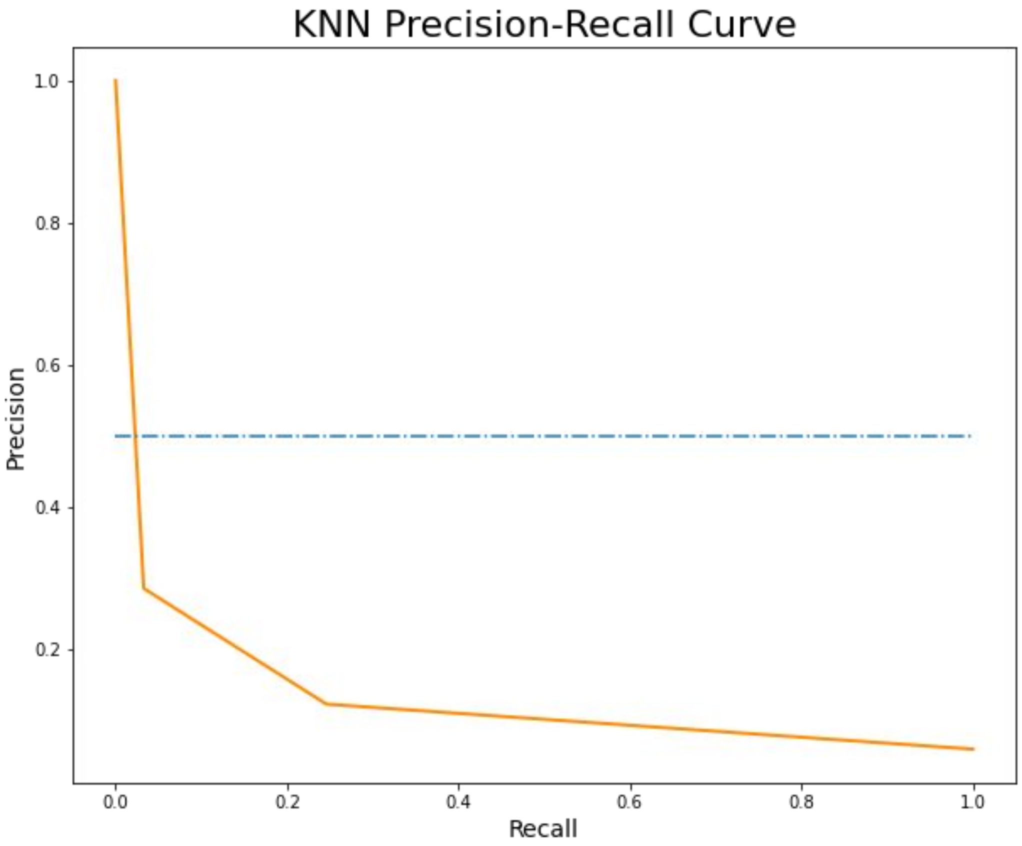


k-Nearest Neighbors Classifier: Building and Evaluating the Model

- Features chosen by were Gender, HyperTension, AvgGlucoseLevel, and Smoker.
- Used k = **3** neighbors.
- F1 score is about **0.06**







- Area under ROC curve is about **0.57** again.
- Area under Precision-Recall curve is about **0.08**.

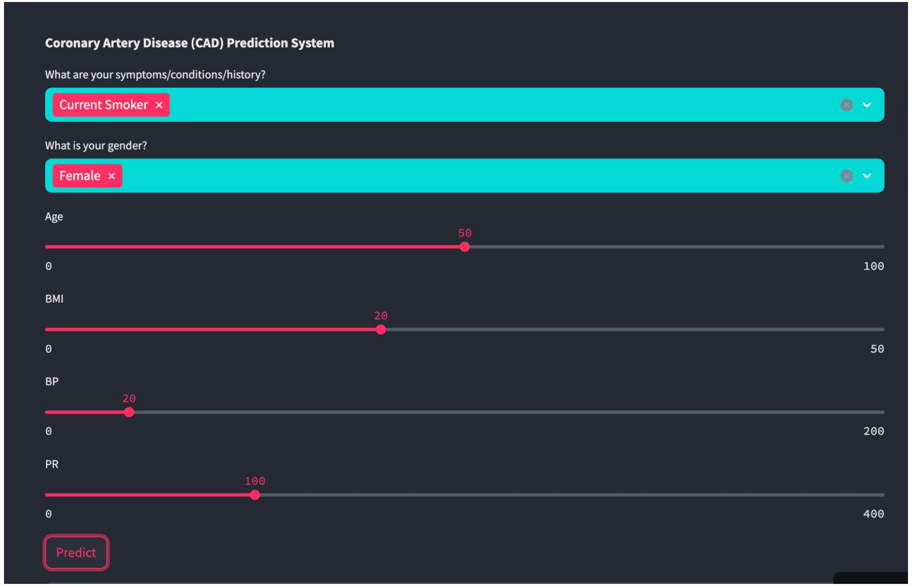
Implementation of Web App

Web App - CAD and Stroke Prediction System

The web app's clear, actionable results were designed to be user-friendly. They are built on our robust models, ensuring high accuracy predictions that will output users with immediate insights.

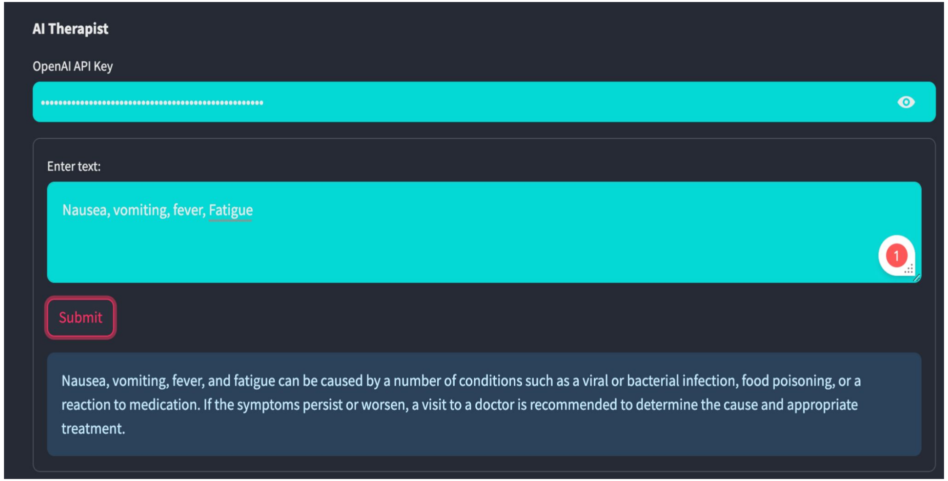
Our web app features two key predictive systems:

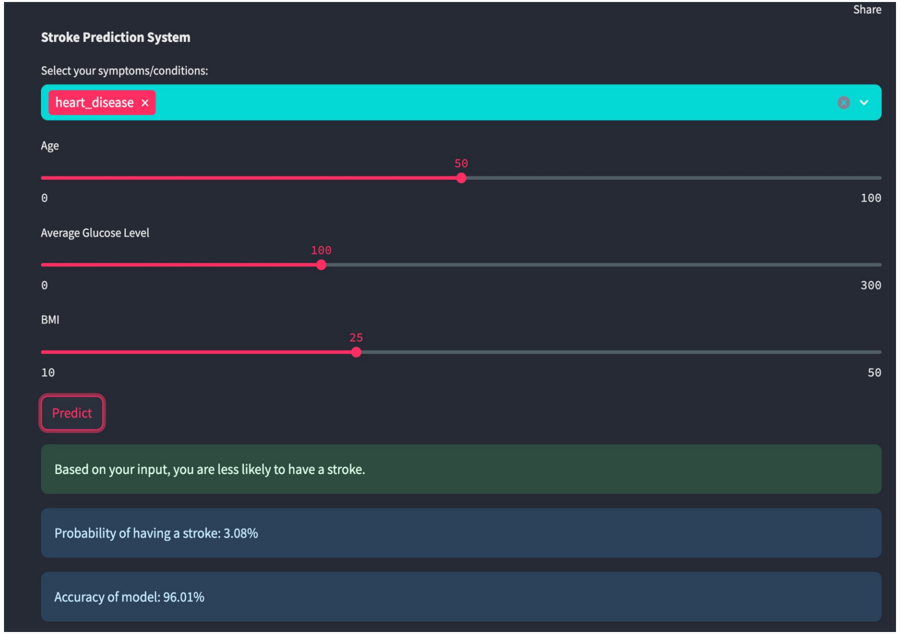
1. **CAD Prediction System:**
   - User-provided health data includes gender, age, body mass index (BMI), blood pressure (BP), pulse rate (PR), and smoking status.
2. **Stroke Prediction System:**
   - User-provided health data includes symptoms/conditions, age, average glucose level, and BMI.
   - Upon prediction, app outputs the probability of having a stroke, along with model's accuracy.



TRY THE APP HERE

Web App - User Interface





[TRY THE APP HERE]

Conclusions

- We were able to achieve notable predictive accuracy, indicating the potential for real world application in early detection.
- A bigger and more varied dataset is required to minimize any bias and enhance the generalizability of the models. We also want to address the issues of class imbalance since that is one of the challenges faced by our current model.
- Growing our database to include a larger range of patient demographics, implementing cutting-edge machine learning model, and integrating a more comprehensive range of health indicators into out app. Also, we plan to refine our model with user feedback to improve and broaden our app’s predictive capabilities.

THANK YOU!
If any questions, please ask :)