

## **TABLE OF CONTENTS**

Dataset.....	1
1. Data Pre-processing: .....	2
2. Feature Selection.....	5
Decision Tree.....	5
3. Classification .....	8
Logistic Regression Classifier .....	8
4. Performance metrics .....	8
Confusion Matrix .....	8
5. Front using django .....	9
Setting.py:.....	11
Models.py .....	14
Base.html .....	17
Forms.py:.....	20
Views.py:.....	21
Urls.py .....	22
6. Predicate Logic .....	24
General: .....	24
Predicate Logic Rules/Statements to Predict Heart Disease .....	24

# SEMESTER PROJECT

Apply

Data processing

Feature Selection

Classification

Performance metrics

Front using django

Dataset

## “heart\_2020\_1.csv”

Our dataset, named "heart\_2020\_1.csv," encompasses details regarding the health and lifestyle factors of individuals. This dataset will train or test machine learning models that can predict certain health outcomes based on the variables. Comprising 18 variables, each representing a distinct aspect of health or behavior, the dataset is employed for training a model aimed at predicting whether a person is afflicted with heart disease or not.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	HeartDisease	BMI	Smoking	AlcoholDrin	Stroke	PhysicalHes	MentalHes	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActi	GenHealth	SleepTime	Asthma	KidneyDiseas
2	No	16.6	Yes	No	No	3	30	No	Female	55-59	White	Yes	Yes	Very good	5 Yes	No	Y
3	No	20.34	No	No	Yes	0	0	No	Female	80 or older	White	No	Yes	Very good	7 No	No	N
4	No	26.58	Yes	No	No	20	30	No	Male	65-69	White	Yes	Yes	Fair	8 Yes	No	N
5	No	24.21	No	No	No	0	0	No	Female	75-79	White	No	No	Good	6 No	No	Y
6	No	23.71	No	No	No	28	0	Yes	Female	40-44	White	No	Yes	Very good	8 No	No	N
7	Yes	28.87	Yes	No	No	6	0	Yes	Female	75-79	Black	No	No	Fair	12 No	No	N
8	No	21.63	No	No	No	15	0	No	Female	70-74	White	No	Yes	Fair	4 Yes	No	Y
9	No	31.64	Yes	No	No	5	0	Yes	Female	80 or older	White	Yes	No	Good	9 Yes	No	N
10	No	26.45	No	No	No	0	0	No	Female	80 or older	White	No	borderNo	Fair	5 No	Yes	N
11	No	40.69	No	No	No	0	0	Yes	Male	65-69	White	No	Yes	Good	10 No	No	N
12	Yes	34.3	Yes	No	No	30	0	Yes	Male	60-64	White	Yes	No	Poor	15 Yes	No	N
13	No	28.71	Yes	No	No	0	0	No	Female	55-59	White	No	Yes	Very good	5 No	No	N
14	No	28.37	Yes	No	No	0	0	Yes	Male	75-79	White	Yes	Yes	Very good	8 No	No	N
15	No	28.15	No	No	No	7	0	Yes	Female	80 or older	White	No	No	Good	7 No	No	N
16	No	29.29	Yes	No	No	0	30	Yes	Female	60-64	White	No	Yes	Good	5 No	No	N
17	No	29.18	No	No	No	1	0	No	Female	50-54	White	No	Yes	Very good	6 No	No	N
18	No	26.26	No	No	No	5	2	No	Female	70-74	White	No	No	Very good	10 No	No	N
19	No	22.59	Yes	No	No	0	30	Yes	Male	70-74	White	No	borderYes	Good	8 No	No	N
20	No	29.86	Yes	No	No	0	0	Yes	Female	75-79	Black	Yes	No	Fair	5 No	Yes	Y
21	No	18.13	No	No	No	0	0	No	Male	80 or older	White	No	Yes	Excellent	8 No	No	Y

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
103457	No	43.76	No	No	No	0	0	No	Male	50-54	White	No	Yes	Good	8 No	No	No
103458	No	20.09	No	No	No	0	7	No	Female	18-24	White	No	Yes	Very good	7 No	No	No
103459	No	26.76	No	No	No	0	20	No	Female	25-29	Black	No	Yes	Very good	6 No	No	No
103460	No	30.41	No	No	No	0	0	No	Male	50-54	White	No	No	Very good	5 No	No	No
103461	No	26.63	Yes	No	No	1	0	No	Male	65-69	Black	Yes	No	Fair	8 No	No	No
103462	No	29.18	Yes	No	No	0	0	No	Female	25-29	White	No	No	Excellent	6 No	No	No
103463	No	20.53	Yes	No	No	0	0	No	Male	45-49	White	No	Yes	Excellent	8 No	No	No
103464	No	19.49	No	No	No	2	0	No	Female	30-34	White	No	No	Excellent	5 No	No	No
103465	No	63.47	Yes	No	No	0	30	Yes	Female	55-59	White	No	Yes	Fair	3 No	No	No
103466	No	32.32	No	No	No	20	30	Yes	Female	60-64	White	Yes	No	Fair	4 No	Yes	No
103467	No	22.5	Yes	No	No	0	0	No	Female	70-74	White	No	Yes	Very good	8 Yes	No	No
103468	No	23.49	No	No	No	0	0	No	Female	65-69	White	No	Yes	Good	10 No	No	No
103469	No	24.95	No	No	No	0	0	No	Male	25-29	White	No	Yes	Fair	6 No	No	No
103470	No	27.27	Yes	No	No	15	9	Yes	Female	55-59	White	No	Yes	Excellent	8 No	No	No
103471	No	23.91	Yes	No	No	0	0	No	Female	40-44	White	No	Yes	Excellent	8 No	No	No
103472	No	24.37	No	No	No	0	0	No	Female	60-64	White	No	No	Excellent	8 No	No	No
103473	No	29.86	Yes	No	No	0	0	No	Female	25-29	White	No	No	Good	5 Yes	No	No
103474	No	33.99	No	No	No	0	0	No	Female	25-29	Black	No	Yes	Good	6 No	No	No
103475	No	21.14	No	No	No	0	25	No	Female	35-39	White	No	Yes	Very good	6 No	No	No
103476	No	19.01	Yes	No	No	0	0	No	Female	35-39	White	No	Yes	Excellent	8 No	No	No
103477	No	21.41	Yes	No	No	0	0	No	Male	50-54	White	No	Yes	Very good	7 No	No	No

## **1. Data Pre-processing:**

This code uses **NumPy** and **Pandas** to handle data and employs scikit-learn's **LabelEncoder** and **StandardScaler** for preprocessing, indicating a common data preparation pipeline for machine learning tasks. It facilitates the transformation of input data, encoding categorical labels and standardizing numerical features.

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Title Bar:** AI\_project
- Left Sidebar:** Explorer, AI\_PROJECT, desktop.ini, heart\_2020\_1.csv, Project.ipynb.
- Code Cells:**
  - [2] import numpy as np  
import pandas as pd  
from sklearn.preprocessing import LabelEncoder, StandardScaler  
0.0s Python
  - [3] df = pd.read\_csv('heart\_2020\_1.csv')  
1.5s Python
  - [4] df.dropna(inplace=True)  
0.5s Python
- Output Cell [5]:** df (0.1s)
  - Content: A table showing the first few rows of the dataset.
  - Columns: HeartDisease, BMI, Smoking, AlcoholDrinking, Stroke, PhysicalHealth, MentalHealth, DiffWalking, Sex, AgeCategory, Race, Diabetic.
  - Data:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic
0	No	16.60	Yes	No	No	3.0	30.0	No	Female	55-59	White	Yes
1	No	20.34	No	No	Yes	0.0	0.0	No	Female	80 or older	White	No
2	No	26.58	Yes	No	No	20.0	30.0	No	Male	65-69	White	Yes
3	No	24.21	No	No	No	0.0	0.0	No	Female	75-79	White	No
- Bottom Status Bar:** In 2, Col 35, CRLF, Cell 39 of 39, 4°C, Smoke, 11/16/2024

File Edit Selection View Go Run ... AI\_project

EXPLORER AL\_PROJECT desktop.ini heart\_2020\_1.csv Project.ipynb

Project.ipynb > from sklearn.linear\_model import LogisticRegression

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ... Python 3.12.1

	2	No	26.58	Yes	No	No	20.0	30.0	No	Male	65-69	White	Yes
3	No	24.21	No	No	No	0.0	0.0	No	Female	75-79	White	No	
4	No	23.71	No	No	No	28.0	0.0	Yes	Female	40-44	White	No	
...	...	...	...	...	...	...	...	...	...	...	...	...	
319790	Yes	27.41	Yes	No	No	7.0	0.0	Yes	Male	60-64	Hispanic	Yes	
319791	No	29.84	Yes	No	No	0.0	0.0	No	Male	35-39	Hispanic	No	
319792	No	24.24	No	No	No	0.0	0.0	No	Female	45-49	Hispanic	No	
319793	No	32.81	No	No	No	0.0	0.0	No	Female	25-29	Hispanic	No	
319794	No	46.56	No	No	No	0.0	0.0	No	Female	80 or older	Hispanic	No	

319795 rows × 18 columns

```
[6] label_encoder = LabelEncoder()
    ✓ 0.0s Python
```

```
[7] columns_to_encode = ['HeartDisease', 'Smoking', 'AlcoholDrinking', 'Stroke', 'DiffWalking', 'Sex', 'AgeCategory', 'Race', 'Diabetic', 'Ph
    for column in columns_to_encode:
        df[column] = label_encoder.fit_transform(df[column])
    ✓ 2.0s Python
```

> OUTLINE > TIMELINE 0 0 0 0 0 Type here to search 4°C Smoke 11:53 AM Cell 39 of 39 1/16/2024

This code uses a **LabelEncoder** from scikit-learn to transform categorical columns in a DataFrame named 'df,' specified in 'columns\_to\_encode.' It efficiently converts categorical data into numerical format for machine learning applications.

This screenshot shows a Jupyter Notebook interface with a dark theme. The code cell at the top imports LogisticRegression and displays a DataFrame named df. The DataFrame contains 319795 rows and 18 columns, including categorical and numerical features. The output cell below shows the first few rows of the scaled DataFrame.

```
df
```

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	Phys
0	0	16.60	1	0	0	3.0	30.0	0	0	7	5	2	
1	0	20.34	0	0	1	0.0	0.0	0	0	12	5	0	
2	0	26.58	1	0	0	20.0	30.0	0	1	9	5	2	
3	0	24.21	0	0	0	0.0	0.0	0	0	11	5	0	
4	0	23.71	0	0	0	28.0	0.0	1	0	4	5	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
319790	1	27.41	1	0	0	7.0	0.0	1	1	8	3	2	
319791	0	29.84	1	0	0	0.0	0.0	0	1	3	3	0	
319792	0	24.24	0	0	0	0.0	0.0	0	0	5	3	0	
319793	0	32.81	0	0	0	0.0	0.0	0	0	1	3	0	
319794	0	46.56	0	0	0	0.0	0.0	0	0	12	3	0	

319795 rows × 18 columns

This code displays encoded columns in DataFrame named 'df'.

This screenshot shows a Jupyter Notebook interface with a dark theme. The code cell imports StandardScaler and SelectKBest, then scales the numerical columns of the DataFrame df and selects the most important features. The resulting scaled DataFrame is displayed in the output cell.

```
scaler = StandardScaler()
numerical_columns = ['BMI', 'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'AgeCategory', 'Race', 'SleepTime']
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
```

```
from sklearn.feature_selection import SelectKBest, f_classif
```

```
X = df.drop(columns=['HeartDisease']) #independent
y = df['HeartDisease'] #dependant
```

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabet
0	0	-1.844750	1	0	0	-0.046751	3.281069	-0.401578	0	0.136184	0.497653	
1	0	-1.256338	0	0	1	-0.424070	-0.490039	-0.401578	0	1.538806	0.497653	
2	0	-0.274603	1	0	0	2.091388	3.281069	-0.401578	1	0.697233	0.497653	
3	0	-0.647473	0	0	0	-0.424070	-0.490039	-0.401578	0	1.258282	0.497653	
4	0	-0.726138	0	0	0	3.097572	-0.490039	2.490174	0	-0.705388	0.497653	
...	...	...	...	...	...	...	...	...	...	...	...	...
319790	1	-0.144019	1	0	0	0.456341	-0.490039	2.490174	1	0.416709	-1.152231	

The code scales numerical features like **BMI**, **PhysicalHealth**, and **SleepTime** using **StandardScaler** and selects the most relevant features for predicting heart disease with **SelectKBest** and **f\_classif**. The resulting data, represented by independent variables (X) and the dependent variable (y), is prepared for training a machine learning model and displayed.

The screenshot shows a Jupyter Notebook interface with a dark theme. In the top right, the title bar says "AI\_project". The left sidebar has sections for "EXPLORER", "AL\_PROJECT", "desktop.ini", and "heart\_2020\_1.csv". A file named "Project.ipynb" is currently selected. The main area shows a table of data with 319795 rows and 18 columns, followed by a code cell:

```

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Exclude the target variable (assuming it's 'HeartDisease' from the visualization
features = df.drop('HeartDisease', axis=1)

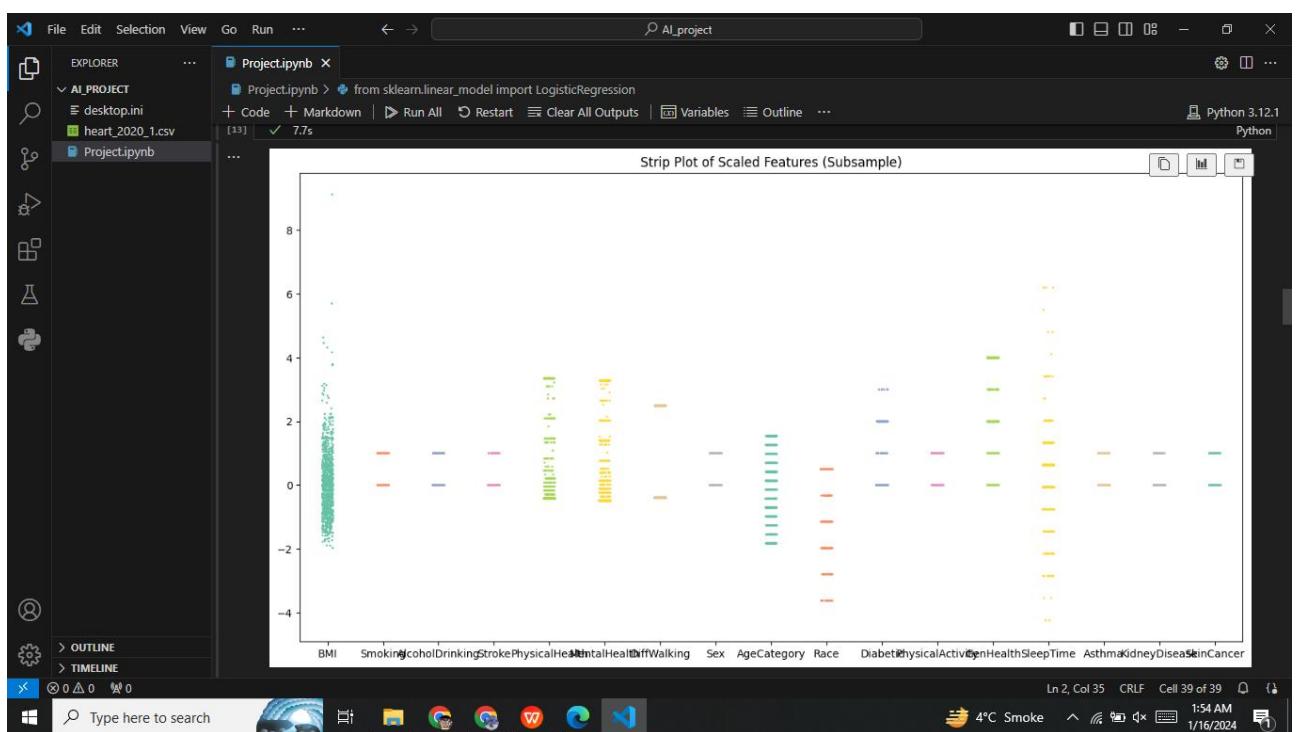
# Subsample 1000 rows for faster visualization (adjust the number based on your dataset size)
subsample = features.sample(1000, random_state=42)

# Plot strip plot for each feature
plt.figure(figsize=(16, 8))
sns.stripplot(data=subsample, size=2, jitter=True, palette="Set2")
plt.title("Strip Plot of Scaled Features (Subsample)")
plt.show()

```

The code cell has a status bar indicating "7.7s" and "Python". Below the code cell is the resulting "Strip Plot of Scaled Features (Subsample)" visualization.

This Python code prepares a DataFrame by excluding the 'HeartDisease' target variable and then generates a strip plot to visualize the distribution of features in a random subsample of 1000 rows from the dataset. The strip plot displays the individual data points for each feature, aiding in the identification of patterns and outliers.



This shows distribution of features plotted in graph.

## 2. Feature Selection

### Decision Tree

The screenshot shows a Jupyter Notebook interface with a dark theme. The left sidebar has an 'EXPLORER' section showing a project structure with 'AI\_PROJECT', 'desktop.ini', and 'heart\_2020\_1.csv'. The main area displays the following Python code:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel

X = df.drop(columns=['HeartDisease']) #independent
y = df['HeartDisease'] #dependant

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize a decision tree classifier
clf = DecisionTreeClassifier()

# Train the classifier on the training data
clf.fit(X_train, y_train)
```

The code is run in three cells, with execution times of 1.6s, 0.3s, and 0.0s respectively. The status bar at the bottom right indicates 'Ln 2, Col 35 CRLF Cell 39 of 39'.

This Python code is implementing a **Decision Tree Classification Model** to predict the 'HeartDisease' variable based on the independent features in the DataFrame `df`. It splits the data into **training** and **testing** sets, then trains a **Decision Tree Classifier** using the training data. The model is stored in the variable `clf`, ready for evaluation on the test set or making predictions on new data.

The screenshot shows a Jupyter Notebook interface with a dark theme. The left sidebar has an 'EXPLORER' section showing a project structure with 'AI\_PROJECT', 'desktop.ini', and 'heart\_2020\_1.csv'. The main area displays the following Python code:

```
# Train the classifier on the training data
clf.fit(X_train, y_train)

... * DecisionTreeClassifier
DecisionTreeClassifier()

# SelectFromModel to perform feature selection
sfm = SelectFromModel(clf, threshold=0.1)
sfm.fit(X_train, y_train)

... > SelectFromModel
> estimator: DecisionTreeClassifier
    > DecisionTreeClassifier

selected_features = X.columns[sfm.get_support()]

# Print the selected features
print('Selected Features:', selected_features)
```

The code is run in four cells, with execution times of 5.4s, 6.0s, 0.0s, and 0.0s respectively. The status bar at the bottom right indicates 'Ln 2, Col 35 CRLF Cell 39 of 39'.

This code is using a pre-trained Decision Tree Classifier (`clf`) to identify important features for predicting 'HeartDisease' using the **'SelectFromModel'** method. It sets a threshold of 0.1, and the features surpassing this threshold are stored in `selected\_features` for potential use in model training or analysis.

The screenshot shows a Jupyter Notebook interface with a dark theme. The left sidebar includes icons for file operations, search, and project management. The main area displays a single code cell:

```
# Print the selected features
print("Selected Features:", selected_features)
[20] ✓ 0.0s
...
Selected Features: Index(['BMI', 'SleepTime'], dtype='object')

# Transform the training and testing sets to include only the selected features
X_train_selected = sfm.transform(X_train)
X_test_selected = sfm.transform(X_test)
[21] ✓ 0.0s

# Train the model using the selected features
clf.fit(X_train_selected, y_train)
[22] ✓ 1.6s
...
* DecisionTreeClassifier
DecisionTreeClassifier()

# Make predictions on the test set using the selected features
y_pred = clf.predict(X_test_selected)
[23] ✓ 0.0s
```

The status bar at the bottom indicates "Ln 2, Col 38 CRLF Cell 39 of 39".

This code identifies and prints the features deemed important for predicting 'HeartDisease' using a pre-trained Decision Tree Classifier. It then retrains the classifier with only the selected features, streamlining the model to focus on the most influential predictors.

The screenshot shows a Jupyter Notebook interface with a dark theme. The left sidebar includes icons for file operations, search, and project management. The main area displays a single code cell:

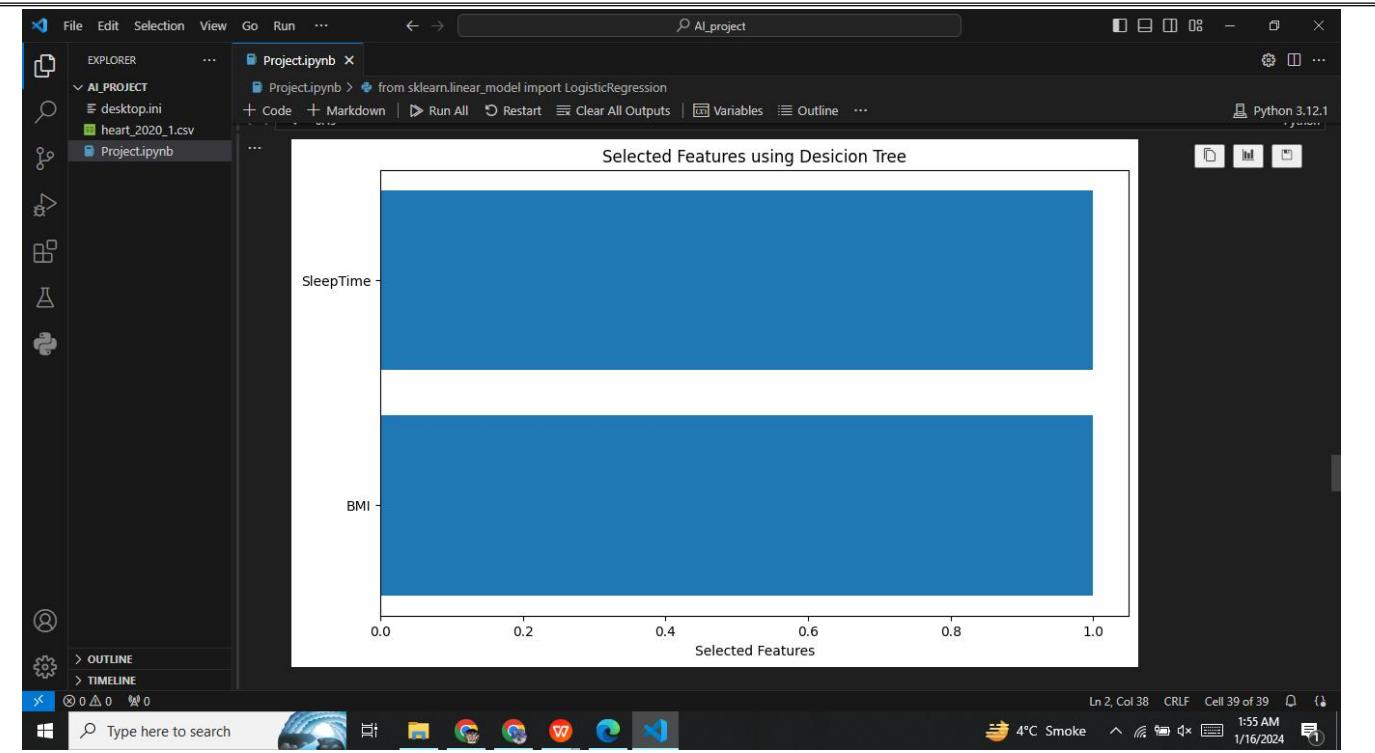
```
# Make predictions on the test set using the selected features
y_pred = clf.predict(X_test_selected)
[24] ✓ 0.0s
...
from sklearn.metrics import accuracy_score
[25] ✓ 0.0s

# Evaluate the performance of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on the test set: {accuracy:.2f}")
[26] ✓ 0.0s
...
Accuracy on the test set: 0.91

# Visualize selected features
plt.figure(figsize=(10, 6))
plt.bar(selected_features, [1] * len(selected_features))
plt.xlabel('Selected Features')
plt.title('Selected Features using Desicion Tree')
plt.show()
[27] ✓ 0.4s
...
Selected Features using Desicion Tree
```

The status bar at the bottom indicates "Ln 2, Col 38 CRLF Cell 39 of 39".

This code predicts 'HeartDisease' on the test set using the trained Decision Tree Classifier with the selected features and assesses the model's accuracy. Additionally, it visualizes the chosen features through a horizontal bar chart for better interpretability.



This graph is displaying selected features.

```

File Edit Selection View Go Run ...
Project.ipynb > from sklearn.linear_model import LogisticRegression
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...
Python 3.12.1

X_test_selected
[32] ✓ 0.0s
...
array([[-0.10940665, -0.06760053],
       [-1.00303776, -0.76397703],
       [ 0.47113892, -0.76397703],
       ...,
       [-1.13519447, -0.06760053],
       [-0.68208574,  0.62877596],
       [ 0.5576701 , -0.06760053]]]

y_pred
[33] ✓ 0.0s
...
array([0, 0, 0, ..., 0, 0, 0])

Applying Classifiers

KNN Classifier

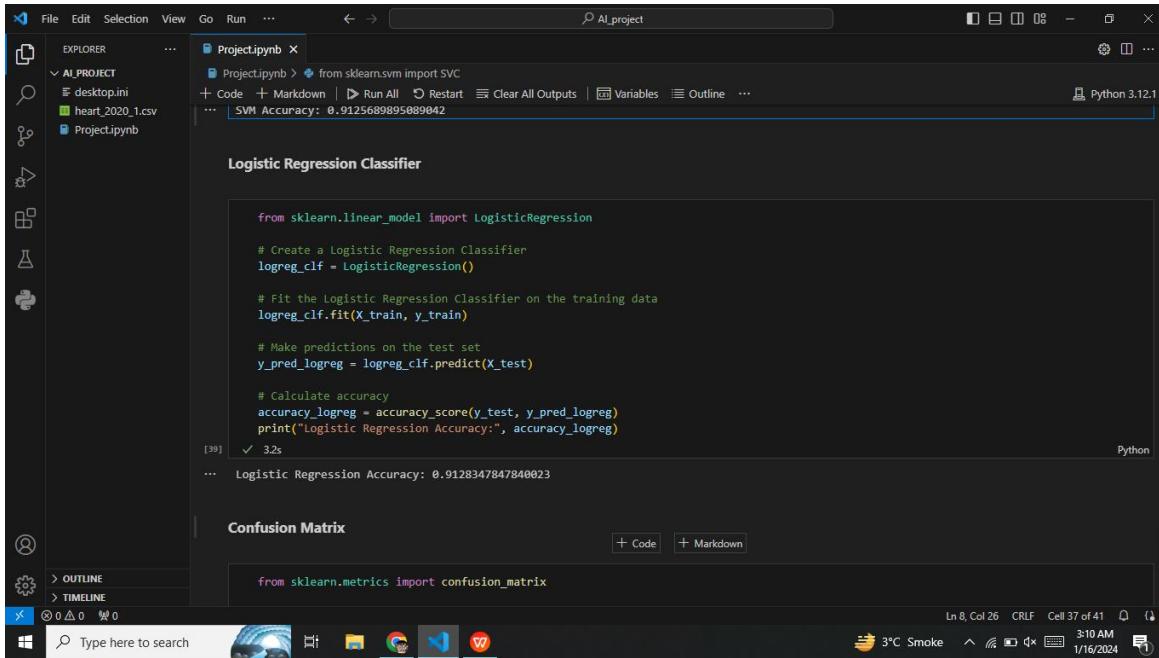
from sklearn.neighbors import KNeighborsClassifier
# Create a KNN Classifier
knn_clf = KNeighborsClassifier(n_neighbors=5) # You can adjust the number of neighbors

```

The predicted values are stored in `y_pred`, representing the model's output for the target variable.

### 3. Classification

#### Logistic Regression Classifier



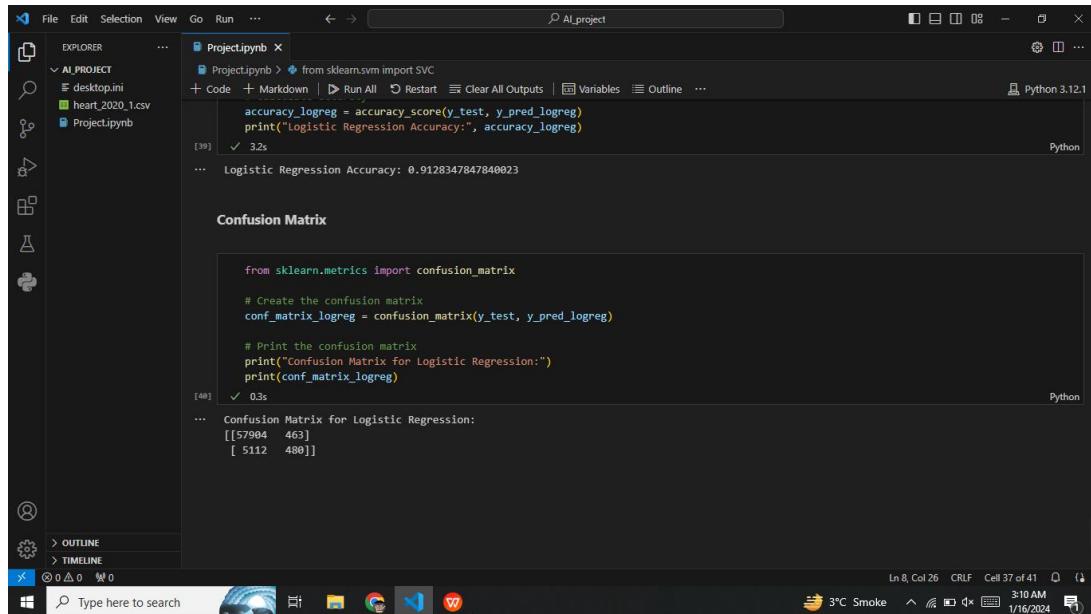
The screenshot shows a Jupyter Notebook interface with the following details:

- File Explorer:** Shows a project named "AI\_PROJECT" containing "desktop.ini", "heart\_2020\_1.csv", and "Project.ipynb".
- Code Cell:** Displays Python code for training a Logistic Regression classifier and calculating accuracy. The output shows an accuracy of 0.9125689895089042.
- Output Cell:** Shows the resulting confusion matrix for the logistic regression model.
- System Bar:** Includes icons for file operations, search, and system status (3°C, 3:10 AM, 1/16/2024).

This code trains a **Logistic Regression Classifier** using the training data ('X\_train' and 'y\_train') and then uses it to predict 'HeartDisease' on the test set ('X\_test'). The resulting accuracy of the model on the test data is calculated and printed as a performance metric.

### 4. Performance metrics

#### Confusion Matrix



The screenshot shows a Jupyter Notebook interface with the following details:

- File Explorer:** Shows a project named "AI\_PROJECT" containing "desktop.ini", "heart\_2020\_1.csv", and "Project.ipynb".
- Code Cell:** Displays Python code for calculating the confusion matrix and printing it. The output shows a confusion matrix for Logistic Regression with values [[57904 463], [ 5112 488]].
- System Bar:** Includes icons for file operations, search, and system status (3°C, 3:10 AM, 1/16/2024).

This code calculates and prints the confusion matrix for evaluating the performance of a logistic regression model on the test set. The **confusion matrix** provides a detailed breakdown of the model's predictions, showing the number of true positive, true negative, false positive, and false negative instances.

## 5. Front using django

The image consists of four vertically stacked screenshots of a Microsoft Windows Command Prompt window. Each screenshot shows a different step in the process of installing Django and django-admin.

- Screenshot 1:** Shows the command `python --version` being run, displaying Python 3.12.1.
- Screenshot 2:** Shows the command `pip install django` being run. It lists several dependencies being downloaded, including Django 5.0.1, aspref 4.3.7.2, and sqlparse 0.4.4. A warning message at the bottom states: "WARNING: The script sqlformat.exe is installed in 'C:\Users\AZAN LAPTOP STORE\AppData\Roaming\Python\Python312\Scripts' which is not on PATH. Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location."
- Screenshot 3:** Shows the command `pip install pipenv` being run. It lists dependencies like certifi, filelock, and virtualenv, along with their versions and download links. A warning message at the bottom states: "WARNING: The script virtualenv.exe is installed in 'C:\Users\AZAN LAPTOP STORE\AppData\Roaming\Python\Python312\Scripts' which is not on PATH. Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location."
- Screenshot 4:** Shows the command `django-admin --version` being run. It displays the error message: "'django-admin' is not recognized as an internal or external command, operable program or batch file."

```

C:\ Command Prompt
Microsoft Windows [Version 10.0.19045.3938]
(c) Microsoft Corporation. All rights reserved.

C:\Users\AZAN LAPTOP STORE>cd AppData\Roaming\Python\Python312\Scripts
C:\Users\AZAN LAPTOP STORE>AppData\Roaming\Python\Python312\Scripts>django-admin --version
5.0.1
C:\Users\AZAN LAPTOP STORE>AppData\Roaming\Python\Python312\Scripts>
C:\Users\AZAN LAPTOP STORE>

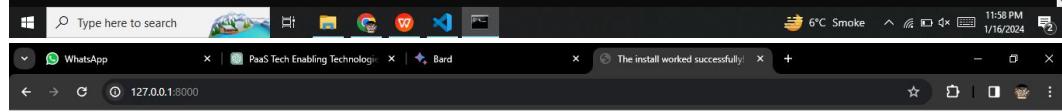
C:\ Command Prompt - python manage.py runserver
C:\Users\AZAN LAPTOP STORE>AppData\Roaming\Python\Python312\site-packages>django-admin startproject myproject
'django-admin' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\AZAN LAPTOP STORE>AppData\Roaming\Python\Python312\site-packages>
C:\Users\AZAN LAPTOP STORE>AppData\Roaming\Python\Python312\site-packages>cd ..
C:\Users\AZAN LAPTOP STORE>AppData\Roaming\Python\Python312>d ..
C:\Users\AZAN LAPTOP STORE>AppData\Roaming\Python> cd Python312\Scripts
C:\Users\AZAN LAPTOP STORE>AppData\Roaming\Python\Python312\Scripts>django-admin --version
5.0.1
C:\Users\AZAN LAPTOP STORE>AppData\Roaming\Python\Python312\Scripts>django-admin startproject myproject
C:\Users\AZAN LAPTOP STORE>AppData\Roaming\Python\Python312\Scripts>cd myproject
C:\Users\AZAN LAPTOP STORE>AppData\Roaming\Python\Python312\Scripts>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
January 16, 2024 - 23:56:54
Django version 5.0.1, using settings 'myproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[16/Jan/2024 23:58:07] "GET / HTTP/1.1" 200 10629
Not Found: /favicon.ico
[16/Jan/2024 23:58:09] "GET /favicon.ico HTTP/1.1" 404 2113

```



djongo View release notes for Django 5.0

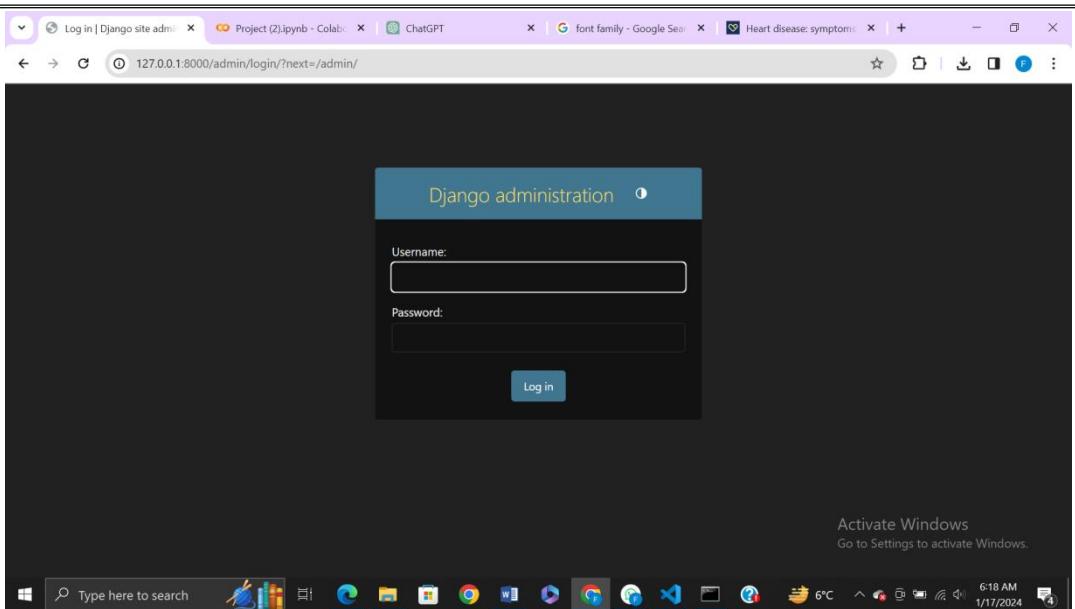


The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.



## Making Interface for Disease Prediction



## Setting.py:

""""

*Django settings for myproject project.*

*Generated by 'django-admin startproject' using Django 5.0.1.*

*For more information on this file, see*

*<https://docs.djangoproject.com/en/5.0/topics/settings/>*

*For the full list of settings and their values, see*

*<https://docs.djangoproject.com/en/5.0/ref/settings/>*

""""

```
from pathlib import Path
import os
```

*# Build paths inside the project like this: BASE\_DIR / 'subdir'.*

```
BASE_DIR = Path(__file__).resolve().parent.parent
```

*# Quick-start development settings - unsuitable for production*

*# See <https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/>*

*# SECURITY WARNING: keep the secret key used in production secret!*

```
SECRET_KEY = "django-insecure-^63xwady&k#icv2=j+skerh^t&do#1*%&wq=s6k-pu!vw%psd6"
```

*# SECURITY WARNING: don't run with debug turned on in production!*

```
DEBUG = True
```

```
ALLOWED_HOSTS = []
```

*# Application definition*

```
INSTALLED_APPS = [
```

*# my apps*

*"hdI",*

*# django apps*

```

"django.contrib.admin",
"django.contrib.auth",
"django.contrib.contenttypes",
"django.contrib.sessions",
"django.contrib.messages",
"django.contrib.staticfiles",
"django_extensions",
]

MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
]

ROOT_URLCONF = "myproject.urls"

TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS": [os.path.join(BASE_DIR, 'templates')],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.messages",
            ],
        },
    },
]

WSGI_APPLICATION = "myproject.wsgi.application"

# Database
# https://docs.djangoproject.com/en/5.0/ref/settings/#databases

DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.sqlite3",
        "NAME": BASE_DIR / "db.sqlite3",
    }
}

# Password validation
# https://docs.djangoproject.com/en/5.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME": "django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
    },
]

```

```

    "NAME": "django.contrib.auth.password_validation.MinimumLengthValidator",
},
{
    "NAME": "django.contrib.auth.password_validation.CommonPasswordValidator",
},
{
    "NAME": "django.contrib.auth.password_validation.NumericPasswordValidator",
},
]

```

**# Internationalization**

**# <https://docs.djangoproject.com/en/5.0/topics/i18n/>**

**LANGUAGE\_CODE = "en-us"**

**TIME\_ZONE = "UTC"**

**USE\_I18N = True**

**USE\_TZ = True**

**# Static files (CSS, JavaScript, Images)**

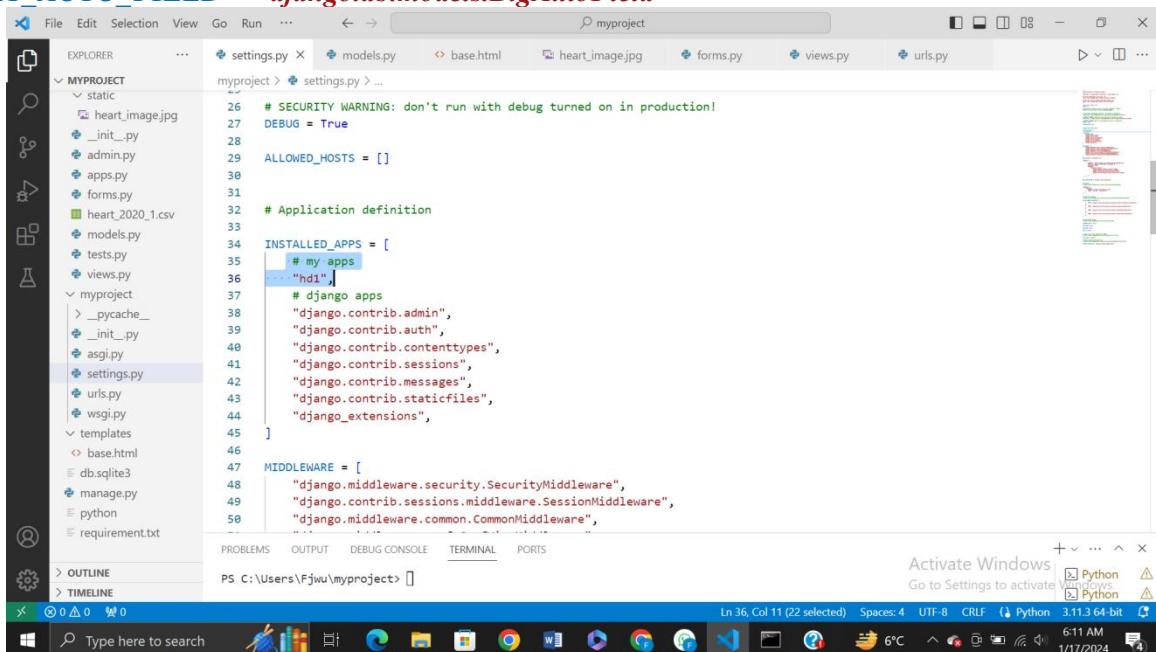
**# <https://docs.djangoproject.com/en/5.0/howto/static-files/>**

**STATIC\_URL = "static/"**

**# Default primary key field type**

**# <https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field>**

**DEFAULT\_AUTO\_FIELD = "django.db.models.BigAutoField"**



```

    myproject > settings.py > ...
    48     "django.middleware.security.SecurityMiddleware",
    49     "django.contrib.sessions.middleware.SessionMiddleware",
    50     "django.middleware.common.CommonMiddleware",
    51     "django.middleware.csrf.CsrfViewMiddleware",
    52     "django.contrib.auth.middleware.AuthenticationMiddleware",
    53     "django.contrib.messages.middleware.MessageMiddleware",
    54     "django.middleware.clickjacking.XFrameOptionsMiddleware",
    55   ]
    56
    57 ROOT_URLCONF = "myproject.urls"
    58
    59 TEMPLATES = [
    60   {
    61     "BACKEND": "django.template.backends.django.DjangoTemplates",
    62     "DIRS": [os.path.join(BASE_DIR, "templates")],
    63     "APP_DIRS": True,
    64     "OPTIONS": {
    65       "context_processors": [
    66         "django.template.context_processors.debug",
    67         "django.template.context_processors.request",
    68         "django.contrib.auth.context_processors.auth",
    69         "django.contrib.messages.context_processors.messages",
    70       ],
    71     },
    72   },
    73 ]
  
```

```

    myproject > settings.py > STATIC_URL
    116
    117     USE_TZ = True
    118
    119
    120     # Static files (CSS, JavaScript, Images)
    121     # https://docs.djangoproject.com/en/5.0/howto/static-files/
    122
    123     STATIC_URL = "static/"
    124
    125     # Default primary key field type
    126     # https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field
    127
    128     DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"
    129
  
```

This code represents the Django project settings in a file named "settings.py" for a project called "myproject." It includes configurations for database settings, middleware, templates, static files, and other Django-specific parameters. Notably, it specifies a SQLite database, template directories, and other settings essential for the proper functioning of a Django web application.

## Models.py

```

import os
import django
from django.db import models
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
  
```

```

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

class Dataset(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField()

class Preprocessing(models.Model):
    dataset = models.ForeignKey(Dataset, on_delete=models.CASCADE)
    method_used = models.CharField(max_length=255)

    @staticmethod
    def preprocess_data():
        df = pd.read_csv('C:\heart_2020_1.csv')
        df.dropna(inplace=True)
        label_encoder = LabelEncoder()
        columns_to_encode = ['HeartDisease', 'Smoking', 'AlcoholDrinking', 'Stroke', 'DiffWalking', 'Sex',
        'AgeCategory', 'Race', 'Diabetic', 'PhysicalActivity', 'GenHealth', 'Asthma', 'KidneyDisease', 'SkinCancer']

        for column in columns_to_encode:
            df[column] = label_encoder.fit_transform(df[column])

        scaler = StandardScaler()
        numerical_columns = ['BMI', 'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'AgeCategory', 'Race',
        'SleepTime']
        df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

        return df

class FeatureSelection(models.Model):
    preprocessing = models.OneToOneField(Preprocessing, on_delete=models.CASCADE)
    method_used = models.CharField(max_length=255)
    selected_features = models.TextField()

    @staticmethod
    def decision_tree_feature_selection(df):
        X = df.drop(columns=['HeartDisease'])
        y = df['HeartDisease']

        clf = DecisionTreeClassifier()
        clf.fit(X, y)

        sfm = SelectFromModel(clf, threshold=0.1)
        sfm.fit(X, y)
        selected_features = list(X.columns[sfm.get_support()])

        return selected_features

class ConfusionMatrix(models.Model):
    feature_selection = models.OneToOneField(FeatureSelection, on_delete=models.CASCADE)
    true_positive = models.IntegerField()
    true_negative = models.IntegerField()
    false_positive = models.IntegerField()
    false_negative = models.IntegerField()

    @staticmethod
    def generate_confusion_matrix(df, selected_features):

```

*X = df['selected\_features']*

*y = df['HeartDisease']*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

*clf = LogisticRegression()*

*clf.fit(X\_train, y\_train)*

*y\_pred = clf.predict(X\_test)*

*conf\_matrix = confusion\_matrix(y\_test, y\_pred)*

*return conf\_matrix*

*@staticmethod*

*def logistic\_regression\_predict(df, X\_input, selected\_features):*

*clf = LogisticRegression()*

*X = df[selected\_features]*

*y = df['HeartDisease']*

*clf.fit(X, y)*

*prediction = clf.predict(X\_input)*

*return prediction*

```
File Edit Selection View Go Run ... ← → myproject
EXPLORER ... settings.py models.py base.html heart_image.jpg forms.py views.py urls.py
MYPROJECT
  static
    heart_image.jpg
    __init__.py
    admin.py
    apps.py
    forms.py
    heart_2020_1.csv
    models.py
    tests.py
    views.py
  myproject
    > __pycache__
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
    templates
      base.html
    db.sqlite3
    manage.py
    python
    requirement.txt
hd1 > models.py > ConfusionMatrix > logistic_regression_predict
1 import os
2 import django
3 from django.db import models
4 import numpy as np
5 import pandas as pd
6 from sklearn.preprocessing import LabelEncoder, StandardScaler
7 from sklearn.feature_selection import SelectKBest, f_classif
8 from sklearn.tree import DecisionTreeClassifier
9 from sklearn.model_selection import train_test_split
10 from sklearn.feature_selection import SelectFromModel
11 from sklearn.linear_model import LogisticRegression
12 from sklearn.metrics import accuracy_score, confusion_matrix
13 import matplotlib.pyplot as plt
14 import seaborn as sns
15
16 class Dataset(models.Model):
17     name = models.CharField(max_length=255)
18     description = models.TextField()
19
20 class Preprocessing(models.Model):
21     dataset = models.ForeignKey(Dataset, on_delete=models.CASCADE)
22     method_used = models.CharField(max_length=255)
23
24 @staticmethod
25 def preprocess_data():
26     df = pd.read_csv('C:/heart_2020_1.csv')
27
28     method_used = models.CharField(max_length=255)
29
30     @staticmethod
31     def preprocess_data():
32         df = pd.read_csv('C:/heart_2020_1.csv')
33         df.dropna(inplace=True)
34         label_encoder = LabelEncoder()
35         columns_to_encode = ['HeartDisease', 'Smoking', 'AlcoholDrinking', 'Stroke', 'DiffWalking', 'Sex', 'AgeCategory']
36
37         for column in columns_to_encode:
38             df[column] = label_encoder.fit_transform(df[column])
39
40         scaler = StandardScaler()
41         numerical_columns = ['BMI', 'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'AgeCategory', 'Race', 'SleepTime']
42         df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
43
44         return df
45
46 class FeatureSelection(models.Model):
47     preprocessing = models.OneToOneField(Preprocessing, on_delete=models.CASCADE)
48     method_used = models.CharField(max_length=255)
49     selected_features = models.TextField()
50
51     @staticmethod
52     def decision_tree_feature_selection(df):
53         X = df.drop(columns=['HeartDisease'])
54
55     def __str__(self):
56         return self.method_used
57
58     def __repr__(self):
59         return self.method_used
60
61     def __eq__(self, other):
62         if isinstance(other, FeatureSelection):
63             return self.method_used == other.method_used
64         return False
65
66     def __hash__(self):
67         return hash(self.method_used)
68
69     def __lt__(self, other):
70         if isinstance(other, FeatureSelection):
71             return self.method_used < other.method_used
72         return False
73
74     def __gt__(self, other):
75         if isinstance(other, FeatureSelection):
76             return self.method_used > other.method_used
77         return False
78
79     def __le__(self, other):
80         if isinstance(other, FeatureSelection):
81             return self.method_used <= other.method_used
82         return False
83
84     def __ge__(self, other):
85         if isinstance(other, FeatureSelection):
86             return self.method_used >= other.method_used
87         return False
88
89     def __ne__(self, other):
90         if isinstance(other, FeatureSelection):
91             return self.method_used != other.method_used
92         return False
93
94     def __int__(self):
95         return int(self.method_used)
96
97     def __float__(self):
98         return float(self.method_used)
99
100    def __complex__(self):
101        return complex(self.method_used)
102
103    def __bool__(self):
104        return bool(self.method_used)
105
106    def __str__(self):
107        return str(self.method_used)
108
109    def __repr__(self):
110        return repr(self.method_used)
111
112    def __hash__(self):
113        return hash(self.method_used)
114
115    def __lt__(self, other):
116        if isinstance(other, FeatureSelection):
117            return self.method_used < other.method_used
118        return False
119
120    def __gt__(self, other):
121        if isinstance(other, FeatureSelection):
122            return self.method_used > other.method_used
123        return False
124
125    def __le__(self, other):
126        if isinstance(other, FeatureSelection):
127            return self.method_used <= other.method_used
128        return False
129
130    def __ge__(self, other):
131        if isinstance(other, FeatureSelection):
132            return self.method_used >= other.method_used
133        return False
134
135    def __ne__(self, other):
136        if isinstance(other, FeatureSelection):
137            return self.method_used != other.method_used
138        return False
139
140    def __int__(self):
141        return int(self.method_used)
142
143    def __float__(self):
144        return float(self.method_used)
145
146    def __complex__(self):
147        return complex(self.method_used)
148
149    def __bool__(self):
150        return bool(self.method_used)
151
152    def __str__(self):
153        return str(self.method_used)
154
155    def __repr__(self):
156        return repr(self.method_used)
157
158    def __hash__(self):
159        return hash(self.method_used)
160
161    def __lt__(self, other):
162        if isinstance(other, FeatureSelection):
163            return self.method_used < other.method_used
164        return False
165
166    def __gt__(self, other):
167        if isinstance(other, FeatureSelection):
168            return self.method_used > other.method_used
169        return False
170
171    def __le__(self, other):
172        if isinstance(other, FeatureSelection):
173            return self.method_used <= other.method_used
174        return False
175
176    def __ge__(self, other):
177        if isinstance(other, FeatureSelection):
178            return self.method_used >= other.method_used
179        return False
180
181    def __ne__(self, other):
182        if isinstance(other, FeatureSelection):
183            return self.method_used != other.method_used
184        return False
185
186    def __int__(self):
187        return int(self.method_used)
188
189    def __float__(self):
190        return float(self.method_used)
191
192    def __complex__(self):
193        return complex(self.method_used)
194
195    def __bool__(self):
196        return bool(self.method_used)
197
198    def __str__(self):
199        return str(self.method_used)
200
201    def __repr__(self):
202        return repr(self.method_used)
203
204    def __hash__(self):
205        return hash(self.method_used)
206
207    def __lt__(self, other):
208        if isinstance(other, FeatureSelection):
209            return self.method_used < other.method_used
210        return False
211
212    def __gt__(self, other):
213        if isinstance(other, FeatureSelection):
214            return self.method_used > other.method_used
215        return False
216
217    def __le__(self, other):
218        if isinstance(other, FeatureSelection):
219            return self.method_used <= other.method_used
220        return False
221
222    def __ge__(self, other):
223        if isinstance(other, FeatureSelection):
224            return self.method_used >= other.method_used
225        return False
226
227    def __ne__(self, other):
228        if isinstance(other, FeatureSelection):
229            return self.method_used != other.method_used
230        return False
231
232    def __int__(self):
233        return int(self.method_used)
234
235    def __float__(self):
236        return float(self.method_used)
237
238    def __complex__(self):
239        return complex(self.method_used)
240
241    def __bool__(self):
242        return bool(self.method_used)
243
244    def __str__(self):
245        return str(self.method_used)
246
247    def __repr__(self):
248        return repr(self.method_used)
249
250    def __hash__(self):
251        return hash(self.method_used)
252
253    def __lt__(self, other):
254        if isinstance(other, FeatureSelection):
255            return self.method_used < other.method_used
256        return False
257
258    def __gt__(self, other):
259        if isinstance(other, FeatureSelection):
260            return self.method_used > other.method_used
261        return False
262
263    def __le__(self, other):
264        if isinstance(other, FeatureSelection):
265            return self.method_used <= other.method_used
266        return False
267
268    def __ge__(self, other):
269        if isinstance(other, FeatureSelection):
270            return self.method_used >= other.method_used
271        return False
272
273    def __ne__(self, other):
274        if isinstance(other, FeatureSelection):
275            return self.method_used != other.method_used
276        return False
277
278    def __int__(self):
279        return int(self.method_used)
280
281    def __float__(self):
282        return float(self.method_used)
283
284    def __complex__(self):
285        return complex(self.method_used)
286
287    def __bool__(self):
288        return bool(self.method_used)
289
290    def __str__(self):
291        return str(self.method_used)
292
293    def __repr__(self):
294        return repr(self.method_used)
295
296    def __hash__(self):
297        return hash(self.method_used)
298
299    def __lt__(self, other):
300        if isinstance(other, FeatureSelection):
301            return self.method_used < other.method_used
302        return False
303
304    def __gt__(self, other):
305        if isinstance(other, FeatureSelection):
306            return self.method_used > other.method_used
307        return False
308
309    def __le__(self, other):
310        if isinstance(other, FeatureSelection):
311            return self.method_used <= other.method_used
312        return False
313
314    def __ge__(self, other):
315        if isinstance(other, FeatureSelection):
316            return self.method_used >= other.method_used
317        return False
318
319    def __ne__(self, other):
320        if isinstance(other, FeatureSelection):
321            return self.method_used != other.method_used
322        return False
323
324    def __int__(self):
325        return int(self.method_used)
326
327    def __float__(self):
328        return float(self.method_used)
329
330    def __complex__(self):
331        return complex(self.method_used)
332
333    def __bool__(self):
334        return bool(self.method_used)
335
336    def __str__(self):
337        return str(self.method_used)
338
339    def __repr__(self):
340        return repr(self.method_used)
341
342    def __hash__(self):
343        return hash(self.method_used)
344
345    def __lt__(self, other):
346        if isinstance(other, FeatureSelection):
347            return self.method_used < other.method_used
348        return False
349
350    def __gt__(self, other):
351        if isinstance(other, FeatureSelection):
352            return self.method_used > other.method_used
353        return False
354
355    def __le__(self, other):
356        if isinstance(other, FeatureSelection):
357            return self.method_used <= other.method_used
358        return False
359
360    def __ge__(self, other):
361        if isinstance(other, FeatureSelection):
362            return self.method_used >= other.method_used
363        return False
364
365    def __ne__(self, other):
366        if isinstance(other, FeatureSelection):
367            return self.method_used != other.method_used
368        return False
369
370    def __int__(self):
371        return int(self.method_used)
372
373    def __float__(self):
374        return float(self.method_used)
375
376    def __complex__(self):
377        return complex(self.method_used)
378
379    def __bool__(self):
380        return bool(self.method_used)
381
382    def __str__(self):
383        return str(self.method_used)
384
385    def __repr__(self):
386        return repr(self.method_used)
387
388    def __hash__(self):
389        return hash(self.method_used)
390
391    def __lt__(self, other):
392        if isinstance(other, FeatureSelection):
393            return self.method_used < other.method_used
394        return False
395
396    def __gt__(self, other):
397        if isinstance(other, FeatureSelection):
398            return self.method_used > other.method_used
399        return False
399
400    def __le__(self, other):
401        if isinstance(other, FeatureSelection):
402            return self.method_used <= other.method_used
403        return False
404
405    def __ge__(self, other):
406        if isinstance(other, FeatureSelection):
407            return self.method_used >= other.method_used
408        return False
409
410    def __ne__(self, other):
411        if isinstance(other, FeatureSelection):
412            return self.method_used != other.method_used
413        return False
414
415    def __int__(self):
416        return int(self.method_used)
417
418    def __float__(self):
419        return float(self.method_used)
420
421    def __complex__(self):
422        return complex(self.method_used)
423
424    def __bool__(self):
425        return bool(self.method_used)
426
427    def __str__(self):
428        return str(self.method_used)
429
430    def __repr__(self):
431        return repr(self.method_used)
432
433    def __hash__(self):
434        return hash(self.method_used)
435
436    def __lt__(self, other):
437        if isinstance(other, FeatureSelection):
438            return self.method_used < other.method_used
439        return False
440
441    def __gt__(self, other):
442        if isinstance(other, FeatureSelection):
443            return self.method_used > other.method_used
444        return False
445
446    def __le__(self, other):
447        if isinstance(other, FeatureSelection):
448            return self.method_used <= other.method_used
449        return False
450
451    def __ge__(self, other):
452        if isinstance(other, FeatureSelection):
453            return self.method_used >= other.method_used
454        return False
455
456    def __ne__(self, other):
457        if isinstance(other, FeatureSelection):
458            return self.method_used != other.method_used
459        return False
460
461    def __int__(self):
462        return int(self.method_used)
463
464    def __float__(self):
465        return float(self.method_used)
466
467    def __complex__(self):
468        return complex(self.method_used)
469
470    def __bool__(self):
471        return bool(self.method_used)
472
473    def __str__(self):
474        return str(self.method_used)
475
476    def __repr__(self):
477        return repr(self.method_used)
478
479    def __hash__(self):
480        return hash(self.method_used)
481
482    def __lt__(self, other):
483        if isinstance(other, FeatureSelection):
484            return self.method_used < other.method_used
485        return False
486
487    def __gt__(self, other):
488        if isinstance(other, FeatureSelection):
489            return self.method_used > other.method_used
490        return False
491
492    def __le__(self, other):
493        if isinstance(other, FeatureSelection):
494            return self.method_used <= other.method_used
495        return False
496
497    def __ge__(self, other):
498        if isinstance(other, FeatureSelection):
499            return self.method_used >= other.method_used
500        return False
501
502    def __ne__(self, other):
503        if isinstance(other, FeatureSelection):
504            return self.method_used != other.method_used
505        return False
506
507    def __int__(self):
508        return int(self.method_used)
509
510    def __float__(self):
511        return float(self.method_used)
512
513    def __complex__(self):
514        return complex(self.method_used)
515
516    def __bool__(self):
517        return bool(self.method_used)
518
519    def __str__(self):
520        return str(self.method_used)
521
522    def __repr__(self):
523        return repr(self.method_used)
524
525    def __hash__(self):
526        return hash(self.method_used)
527
528    def __lt__(self, other):
529        if isinstance(other, FeatureSelection):
530            return self.method_used < other.method_used
531        return False
532
533    def __gt__(self, other):
534        if isinstance(other, FeatureSelection):
535            return self.method_used > other.method_used
536        return False
537
538    def __le__(self, other):
539        if isinstance(other, FeatureSelection):
540            return self.method_used <= other.method_used
541        return False
542
543    def __ge__(self, other):
544        if isinstance(other, FeatureSelection):
545            return self.method_used >= other.method_used
546        return False
547
548    def __ne__(self, other):
549        if isinstance(other, FeatureSelection):
550            return self.method_used != other.method_used
551        return False
552
553    def __int__(self):
554        return int(self.method_used)
555
556    def __float__(self):
557        return float(self.method_used)
558
559    def __complex__(self):
560        return complex(self.method_used)
561
562    def __bool__(self):
563        return bool(self.method_used)
564
565    def __str__(self):
566        return str(self.method_used)
567
568    def __repr__(self):
569        return repr(self.method_used)
570
571    def __hash__(self):
572        return hash(self.method_used)
573
574    def __lt__(self, other):
575        if isinstance(other, FeatureSelection):
576            return self.method_used < other.method_used
577        return False
578
579    def __gt__(self, other):
580        if isinstance(other, FeatureSelection):
581            return self.method_used > other.method_used
582        return False
583
584    def __le__(self, other):
585        if isinstance(other, FeatureSelection):
586            return self.method_used <= other.method_used
587        return False
588
589    def __ge__(self, other):
590        if isinstance(other, FeatureSelection):
591            return self.method_used >= other.method_used
592        return False
593
594    def __ne__(self, other):
595        if isinstance(other, FeatureSelection):
596            return self.method_used != other.method_used
597        return False
598
599    def __int__(self):
600        return int(self.method_used)
601
602    def __float__(self):
603        return float(self.method_used)
604
605    def __complex__(self):
606        return complex(self.method_used)
607
608    def __bool__(self):
609        return bool(self.method_used)
610
611    def __str__(self):
612        return str(self.method_used)
613
614    def __repr__(self):
615        return repr(self.method_used)
616
617    def __hash__(self):
618        return hash(self.method_used)
619
620    def __lt__(self, other):
621        if isinstance(other, FeatureSelection):
622            return self.method_used < other.method_used
623        return False
624
625    def __gt__(self, other):
626        if isinstance(other, FeatureSelection):
627            return self.method_used > other.method_used
628        return False
629
630    def __le__(self, other):
631        if isinstance(other, FeatureSelection):
632            return self.method_used <= other.method_used
633        return False
634
635    def __ge__(self, other):
636        if isinstance(other, FeatureSelection):
637            return self.method_used >= other.method_used
638        return False
639
640    def __ne__(self, other):
641        if isinstance(other, FeatureSelection):
642            return self.method_used != other.method_used
643        return False
644
645    def __int__(self):
646        return int(self.method_used)
647
648    def __float__(self):
649        return float(self.method_used)
650
651    def __complex__(self):
652        return complex(self.method_used)
653
654    def __bool__(self):
655        return bool(self.method_used)
656
657    def __str__(self):
658        return str(self.method_used)
659
660    def __repr__(self):
661        return repr(self.method_used)
662
663    def __hash__(self):
664        return hash(self.method_used)
665
666    def __lt__(self, other):
667        if isinstance(other, FeatureSelection):
668            return self.method_used < other.method_used
669        return False
670
671    def __gt__(self, other):
672        if isinstance(other, FeatureSelection):
673            return self.method_used > other.method_used
674        return False
675
676    def __le__(self, other):
677        if isinstance(other, FeatureSelection):
678            return self.method_used <= other.method_used
679        return False
680
681    def __ge__(self, other):
682        if isinstance(other, FeatureSelection):
683            return self.method_used >= other.method_used
684        return False
685
686    def __ne__(self, other):
687        if isinstance(other, FeatureSelection):
688            return self.method_used != other.method_used
689        return False
690
691    def __int__(self):
692        return int(self.method_used)
693
694    def __float__(self):
695        return float(self.method_used)
696
697    def __complex__(self):
698        return complex(self.method_used)
699
700    def __bool__(self):
701        return bool(self.method_used)
702
703    def __str__(self):
704        return str(self.method_used)
705
706    def __repr__(self):
707        return repr(self.method_used)
708
709    def __hash__(self):
710        return hash(self.method_used)
711
712    def __lt__(self, other):
713        if isinstance(other, FeatureSelection):
714            return self.method_used < other.method_used
715        return False
716
717    def __gt__(self, other):
718        if isinstance(other, FeatureSelection):
719            return self.method_used > other.method_used
720        return False
721
722    def __le__(self, other):
723        if isinstance(other, FeatureSelection):
724            return self.method_used <= other.method_used
725        return False
726
727    def __ge__(self, other):
728        if isinstance(other, FeatureSelection):
729            return self.method_used >= other.method_used
730        return False
731
732    def __ne__(self, other):
733        if isinstance(other, FeatureSelection):
734            return self.method_used != other.method_used
735        return False
736
737    def __int__(self):
738        return int(self.method_used)
739
740    def __float__(self):
741        return float(self.method_used)
742
743    def __complex__(self):
744        return complex(self.method_used)
745
746    def __bool__(self):
747        return bool(self.method_used)
748
749    def __str__(self):
750        return str(self.method_used)
751
752    def __repr__(self):
753        return repr(self.method_used)
754
755    def __hash__(self):
756        return hash(self.method_used)
757
758    def __lt__(self, other):
759        if isinstance(other, FeatureSelection):
760            return self.method_used < other.method_used
761        return False
762
763    def __gt__(self, other):
764        if isinstance(other, FeatureSelection):
765            return self.method_used > other.method_used
766        return False
767
768    def __le__(self, other):
769        if isinstance(other, FeatureSelection):
770            return self.method_used <= other.method_used
771        return False
772
773    def __ge__(self, other):
774        if isinstance(other, FeatureSelection):
775            return self.method_used >= other.method_used
776        return False
777
778    def __ne__(self, other):
779        if isinstance(other, FeatureSelection):
780            return self.method_used != other.method_used
781        return False
782
783    def __int__(self):
784        return int(self.method_used)
785
786    def __float__(self):
787        return float(self.method_used)
788
789    def __complex__(self):
790        return complex(self.method_used)
791
792    def __bool__(self):
793        return bool(self.method_used)
794
795    def __str__(self):
796        return str(self.method_used)
797
798    def __repr__(self):
799        return repr(self.method_used)
800
801    def __hash__(self):
802        return hash(self.method_used)
803
804    def __lt__(self, other):
805        if isinstance(other, FeatureSelection):
806            return self.method_used < other.method_used
807        return False
808
809    def __gt__(self, other):
810        if isinstance(other, FeatureSelection):
811            return self.method_used > other.method_used
812        return False
813
814    def __le__(self, other):
815        if isinstance(other, FeatureSelection):
816            return self.method_used <= other.method_used
817        return False
818
819    def __ge__(self, other):
820        if isinstance(other, FeatureSelection):
821            return self.method_used >= other.method_used
822        return False
823
824    def __ne__(self, other):
825        if isinstance(other, FeatureSelection):
826            return self.method_used != other.method_used
827        return False
828
829    def __int__(self):
830        return int(self.method_used)
831
832    def __float__(self):
833        return float(self.method_used)
834
835    def __complex__(self):
836        return complex(self.method_used)
837
838    def __bool__(self):
839        return bool(self.method_used)
840
841    def __str__(self):
842        return str(self.method_used)
843
844    def __repr__(self):
845        return repr(self.method_used)
846
847    def __hash__(self):
848        return hash(self.method_used)
849
850    def __lt__(self, other):
851        if isinstance(other, FeatureSelection):
852            return self.method_used < other.method_used
853        return False
854
855    def __gt__(self, other):
856        if isinstance(other, FeatureSelection):
857            return self.method_used > other.method_used
858        return False
859
860    def __le__(self, other):
861        if isinstance(other, FeatureSelection):
862            return self.method_used <= other.method_used
863        return False
864
865    def __ge__(self, other):
866        if isinstance(other, FeatureSelection):
867            return self.method_used >= other.method_used
868        return False
869
870    def __ne__(self, other):
871        if isinstance(other, FeatureSelection):
872            return self.method_used != other.method_used
873        return False
874
875    def __int__(self):
876        return int(self.method_used)
877
878    def __float__(self):
879        return float(self.method_used)
880
881    def __complex__(self):
882        return complex(self.method_used)
883
884    def __bool__(self):
885        return bool(self.method_used)
886
887    def __str__(self):
888        return str(self.method_used)
889
889    def __repr__(self):
890        return repr(self.method_used)
891
892    def __hash__(self):
893        return hash(self.method_used)
894
895    def __lt__(self, other):
896        if isinstance(other, FeatureSelection):
897            return self.method_used < other.method_used
898        return False
899
900    def __gt__(self, other):
901        if isinstance(other, FeatureSelection):
902            return self.method_used > other.method_used
903        return False
904
905    def __le__(self, other):
906        if isinstance(other, FeatureSelection):
907            return self.method_used <= other.method_used
908        return False
909
910    def __ge__(self, other):
911        if isinstance(other, FeatureSelection):
912            return self.method_used >= other.method_used
913        return False
914
915    def __ne__(self, other):
916        if isinstance(other, FeatureSelection):
917            return self.method_used != other.method_used
918        return False
919
920    def __int__(self):
921        return int(self.method_used)
922
923    def __float__(self):
924        return float(self.method_used)
925
926    def __complex__(self):
927        return complex(self.method_used)
928
929    def __bool__(self):
930        return bool(self.method_used)
931
932    def __str__(self):
933        return str(self.method_used)
934
935    def __repr__(self):
936        return repr(self.method_used)
937
938    def __hash__(self):
939        return hash(self.method_used)
940
941    def __lt__(self, other):
942        if isinstance(other, FeatureSelection):
943            return self.method_used < other.method_used
944        return False
945
946    def __gt__(self, other):
947        if isinstance(other, FeatureSelection):
948            return self.method_used > other.method_used
949        return False
950
951    def __le__(self, other):
952        if isinstance(other, FeatureSelection):
953            return self.method_used <= other.method_used
954        return False
955
956    def __ge__(self, other):
957        if isinstance(other, FeatureSelection):
958            return self.method_used >= other.method_used
959        return False
960
961    def __ne__(self, other):
962        if isinstance(other, FeatureSelection):
963            return self.method_used != other.method_used
964        return False
965
966    def __int__(self):
967        return int(self.method_used)
968
969    def __float__(self):
970        return float(self.method_used)
971
972    def __complex__(self):
973        return complex(self.method_used)
974
975    def __bool__(self):
976        return bool(self.method_used)
977
978    def __str__(self):
979        return str(self.method_used)
980
980    def __repr__(self):
981        return repr(self.method_used)
982
982    def __hash__(self):
983        return hash(self.method_used)
984
985    def __lt__(self, other):
986        if isinstance(other, FeatureSelection):
987            return self.method_used < other.method_used
988        return False
989
990    def __gt__(self, other):
991        if isinstance(other, FeatureSelection):
992            return self.method_used > other.method_used
993        return False
994
995    def __le__(self, other):
996        if isinstance(other, FeatureSelection):
997            return self.method_used <= other.method_used
998        return False
999
1000   def __ge__(self, other):
1001      if isinstance(other, FeatureSelection):
1002          return self.method_used >= other.method_used
1003      return False
1004
1005  def __ne__(self, other):
1006      if isinstance(other, FeatureSelection):
1007          return self.method_used != other.method_used
1008      return False
1009
1010  def __int__(self):
1011      return int(self.method_used)
1012
1013  def __float__(self):
1014      return float(self.method_used)
1015
1016  def __complex__(self):
1017      return complex(self.method_used)
1018
1019  def __bool__(self):
1020      return bool(self.method_used)
1021
1022  def __str__(self):
1023      return str(self.method_used)
1024
1025  def __repr__(self):
1026      return repr(self.method_used)
1027
1027  def __hash__(self):
1028      return hash(self.method_used)
1029
1030  def __lt__(self, other):
1031      if isinstance(other, FeatureSelection):
1032          return self.method_used < other.method_used
1033      return False
1034
1035  def __gt__(self, other):
1036      if isinstance(other, FeatureSelection):
1037          return self.method_used > other.method_used
1038      return False
1039
1040  def __le__(self, other):
1041      if isinstance(other, FeatureSelection):
1042          return self.method_used <= other.method_used
1043      return False
1044
1045  def __ge__(self, other):
1046      if isinstance(other, FeatureSelection):
1047          return self.method_used >= other.method_used
1048      return False
1049
1050  def __ne__(self, other):
1051      if isinstance(other, FeatureSelection):
1052          return self.method_used != other.method_used
1053      return False
1054
1055  def __int__(self):
1056      return int(self.method_used)
1057
1058  def __float__(self):
1059      return float(self.method_used)
1060
1061  def __complex__(self):
1062      return complex(self.method_used)
1063
1064  def __bool__(self):
1065      return bool(self.method_used)
1066
1067  def __str__(self):
1068      return str(self.method_used)
1069
1069  def __repr__(self):
1070      return repr(self.method_used)
1071
1071  def __hash__(self):
1072      return hash(self.method_used)
1073
1074  def __lt__(self, other):
1075      if isinstance(other, FeatureSelection):
1076          return self.method_used < other.method_used
1077      return False
1078
1079  def __gt__(self, other):
1080      if isinstance(other, FeatureSelection):
1081          return self.method_used > other.method_used
1082      return False
1083
1084  def __le__(self, other):
1085      if isinstance(other, FeatureSelection):
1086          return self.method_used <= other.method_used
1087      return False
1088
1089  def __ge__(self, other):
1090      if isinstance(other, FeatureSelection):
1091          return self.method_used >= other.method_used
1092      return False
1093
1094  def __ne__(self, other):
1095      if isinstance(other, FeatureSelection):
1096          return self.method_used != other.method_used
1097      return False
1098
1099  def __int__(self):
1100      return int(self.method_used)
1101
1102  def __float__(self):
1103      return float(self.method_used)
1104
1105  def __complex__(self):
1106      return complex(self.method_used)
1107
1108  def __bool__(self):
1109      return bool(self.method_used)
1110
1111  def __str__(self):
1112      return str(self.method_used)
1113
1113  def __repr__(self):
1114      return repr(self.method_used)
1115
1115  def __hash__(self):
1116      return hash(self.method_used)
1117
1117  def __lt__(self, other):
1118      if isinstance(other, FeatureSelection):
1119          return self.method_used < other.method_used
1120      return False
1121
1122  def __gt__(self, other):
112
```

```

File Edit Selection View Go Run ... < > myproject
EXPLORER settings.py models.py base.html heart_image.jpg forms.py views.py urls.py ...
MYPROJECT
static
  heart_image.jpg
  __init__.py
  admin.py
  apps.py
  forms.py
  heart_2020_1.csv
  models.py
  tests.py
  views.py
myproject
  > __pycache__
  __init__.py
  asgi.py
  settings.py
  urls.py
  wsgi.py
templates
  base.html
  db.sqlite3
  manage.py
  python
  requirement.txt
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Fjwu\myproject> []
Activate Windows Go to Settings to activate Windows Python
Ln 89, Col 42 Spaces: 4 UTF-8 CRLF Python 3.11.3 64-bit 6:12 AM 1/17/2024

File Edit Selection View Go Run ... < > myproject
EXPLORER settings.py models.py base.html heart_image.jpg forms.py views.py urls.py ...
MYPROJECT
static
  heart_image.jpg
  __init__.py
  admin.py
  apps.py
  forms.py
  heart_2020_1.csv
  models.py
  tests.py
  views.py
myproject
  > __pycache__
  __init__.py
  asgi.py
  settings.py
  urls.py
  wsgi.py
templates
  base.html
  db.sqlite3
  manage.py
  python
  requirement.txt
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Fjwu\myproject> []
Activate Windows Go to Settings to activate Windows Python
Ln 89, Col 42 Spaces: 4 UTF-8 CRLF Python 3.11.3 64-bit 6:13 AM 1/17/2024

```

The code in the top window is:

```

hd1 > models.py > ConfusionMatrix > logistic_regression_predict
selected_features = models.TextField()

@staticmethod
def decision_tree_feature_selection(df):
    X = df.drop(columns=['HeartDisease'])
    y = df['HeartDisease']

    clf = DecisionTreeClassifier()
    clf.fit(X, y)

    sfm = SelectFromModel(clf, threshold=0.1)
    sfm.fit(X, y)
    selected_features = list(X.columns[sfm.get_support()])

    return selected_features

class ConfusionMatrix(models.Model):
    feature_selection = models.OneToOneField(FeatureSelection, on_delete=models.CASCADE)
    true_positive = models.IntegerField()
    true_negative = models.IntegerField()
    false_positive = models.IntegerField()
    false_negative = models.IntegerField()

    @staticmethod
    def generate_confusion_matrix(df, selected_features):
        X = df[selected_features]

```

The code in the bottom window is:

```

hd1 > models.py > ConfusionMatrix > logistic_regression_predict
@staticmethod
def generate_confusion_matrix(df, selected_features):
    X = df[selected_features]
    y = df['HeartDisease']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    clf = LogisticRegression()
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    conf_matrix = confusion_matrix(y_test, y_pred)

    return conf_matrix

@staticmethod
def logistic_regression_predict(df, X_input, selected_features):
    clf = LogisticRegression()
    X = df[selected_features]
    y = df['HeartDisease']
    clf.fit(X, y)

    prediction = clf.predict(X_input)
    return prediction

```

This Django models.py file defines three models: 'Dataset', 'Preprocessing', and 'FeatureSelection', along with methods for data preprocessing, decision tree-based feature selection, and logistic regression-based confusion matrix generation. It incorporates functionality to preprocess data, select relevant features using a decision tree, and evaluate logistic regression performance with a confusion matrix. These models serve as components for managing datasets, preprocessing steps, feature selection methods, and evaluation metrics in a Django web application.

## Base.html

```

<!-- base.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">

```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>AI PROJECT</title>
{% load static %}
<style>
body {
    background: url('{% static "heart_image.jpg" %}') center center fixed;
    background-size: cover;
    font-family: 'Arial Black', sans-serif;
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    height: 100vh;
    color: #ffffff;
}

h1 {
    color: #050303;
    text-shadow: 2px 2px 2px #000;
}

form {
    background-color: rgba(116, 72, 72, 0.8);
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(105, 92, 92, 0.1);
    margin-top: 20px;
}

input[type="submit"] {
    background-color: #4caf50;
    color: #fff;
    padding: 10px 15px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

input[type="submit"]:hover {
    background-color: #45a049;
}

p {
    color: #ffffff;
    margin-top: 20px;
    text-shadow: 2px 2px 2px #000;
}

</style>
</head>
<body>

```

## <h1>HEART DISEASE PREDICTION</h1>

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Submit">
</form>

{% if result %}
    <p>{{ result }}</p>
{% endif %}
</body>
</html>
```

The screenshot shows a code editor interface with two tabs open: 'base.html' and 'style.css'. The 'base.html' tab contains the following code:

```
<!-- base.html -->
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>AI PROJECT</title>
        {% load static %}<br/>
        <style>
            body {
                background: url('{% static "heart_image.jpg" %}') center center fixed;
                background-size: cover;
                font-family: 'Arial Black', sans-serif;
                margin: 0;
                padding: 0;
                display: flex;
                flex-direction: column;
                align-items: center;
                justify-content: center;
                height: 100vh;
                color: #ffffff;
            }
            h1 {
                color: #050303;
                text-shadow: 2px 2px 2px #000;
            }
        </style>
    </head>
    <body>
        <h1>HEART DISEASE PREDICTION</h1>
        <form method="post">
            {% csrf_token %}
            {{ form.as_p }}
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

The 'style.css' tab contains the following CSS code:

```
h1 {
    color: #050303;
    text-shadow: 2px 2px 2px #000;
}
```

The screenshot shows a code editor interface with two tabs open: 'base.html' and 'style.css'. The 'base.html' tab contains the following code:

```
<!-- base.html -->
<!DOCTYPE html>
<html lang="en">
    <head>
        <link href="style.css" rel="stylesheet">
    </head>
    <body>
        <h1>HEART DISEASE PREDICTION</h1>
        <form method="post">
            {% csrf_token %}
            {{ form.as_p }}
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

The 'style.css' tab contains the following CSS code:

```
h1 {
    color: #050303;
    text-shadow: 2px 2px 2px #000;
}

form {
    background-color: #111111;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 0 10px #111111;
    margin-top: 20px;
}

input[type="submit"] {
    background-color: #4caf50;
    color: #fff;
    padding: 10px 15px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

input[type="submit"]:hover {
    background-color: #3e517d;
}
```

The screenshot shows a code editor interface with the file `base.html` open. The code defines a web page template with a background image, form styling, and dynamic content for heart disease prediction.

```
46 input[type="submit"]:hover {  
47     background-color: #45a049;  
48 }  
49  
50 p {  
51     color: #ffffff;  
52     margin-top: 20px;  
53     text-shadow: 2px 2px 2px #000;  
54 }  
55  
56 //style  
57 //head  
58 </head>  
59 <body>  
60     <h1>HEART DISEASE PREDICTION</h1>  
61     <form method="post">  
62         {% csrf_token %}  
63         {{ form.as_p }}  
64         <input type="submit" value="Submit">  
65     </form>  
66  
67     {% if result %}  
68         <p>{{ result }}</p>  
69     {% endif %}  
70 </body>  
71 </html>
```

This HTML code defines a web page template named "base.html" for a Heart Disease Prediction application. The page includes a background image, styling for form elements, and dynamic content using Django template tags. It features a form for user input, a submission button, and displays the prediction result if available. The styling enhances the visual appeal, and the form facilitates user interaction with the underlying machine learning model for heart disease prediction.

## Forms.py:

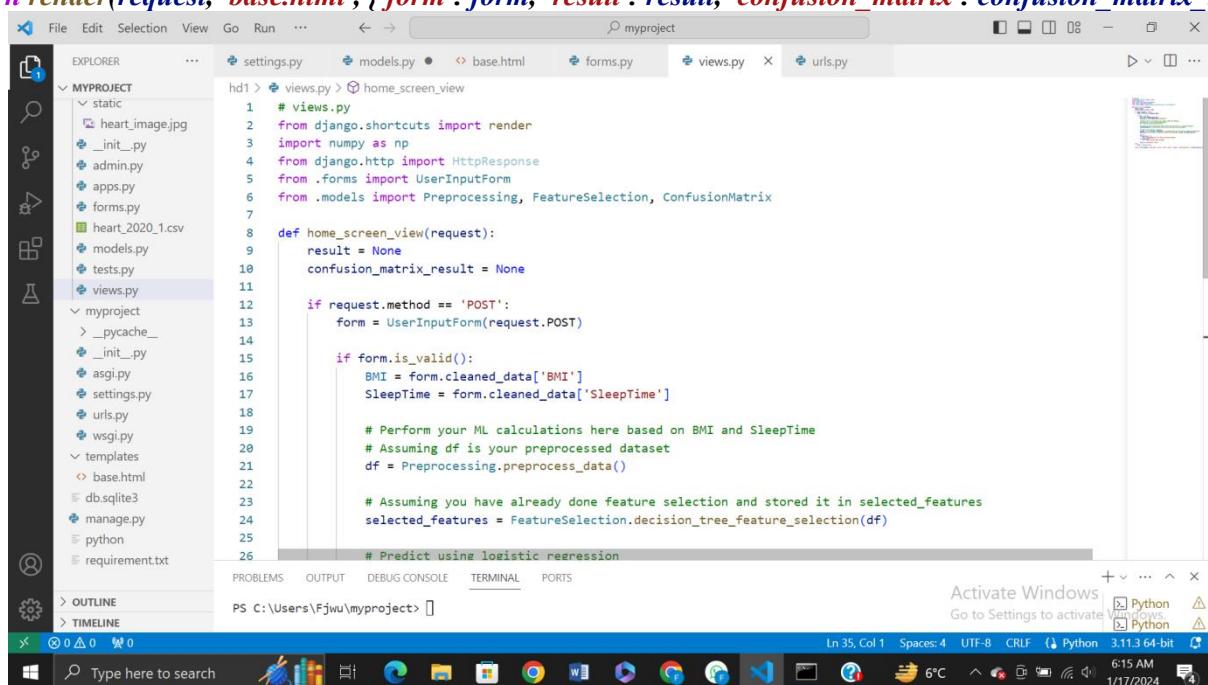
```
# forms.py  
from django import forms  
  
class UserInputForm(forms.Form):  
    BMI = forms.FloatField(label='BMI')  
    SleepTime = forms.FloatField(label='SleepTime')
```

The screenshot shows a code editor interface with the file `forms.py` open. It contains the definition for the `UserInputForm` class, which includes fields for BMI and SleepTime.

```
1 # forms.py  
2 from django import forms  
3  
4 class UserInputForm(forms.Form):  
5     BMI = forms.FloatField(label='BMI')  
6     SleepTime = forms.FloatField(label='SleepTime')
```

## Views.py:

```
# views.py
from django.shortcuts import render
import numpy as np
from django.http import HttpResponseRedirect
from .forms import UserInputForm
from .models import Preprocessing, FeatureSelection, ConfusionMatrix
def home_screen_view(request):
    result = None
    confusion_matrix_result = None
    if request.method == 'POST':
        form = UserInputForm(request.POST)
        if form.is_valid():
            BMI = form.cleaned_data['BMI']
            SleepTime = form.cleaned_data['SleepTime']
            # Perform your ML calculations here based on BMI and SleepTime
            # Assuming df is your preprocessed dataset
            df = Preprocessing.preprocess_data()
            # Assuming you have already done feature selection and stored it in selected_features
            selected_features = FeatureSelection.decision_tree_feature_selection(df)
            # Predict using logistic regression
            X_input = np.array([[BMI, SleepTime]]) # Assuming these are the features needed for prediction
            prediction = ConfusionMatrix.logistic_regression_predict(df, X_input, selected_features)
            str = ""
            if prediction == 0:
                str = "Congratulations! You do not have Heart Disease"
            elif prediction == 1:
                str = "Ops! You have Heart Disease"
            result = f"Prediction: {str}"
        else:
            form = UserInputForm()
    return render(request, 'base.html', {'form': form, 'result': result, 'confusion_matrix': confusion_matrix_result})
```



The screenshot shows a code editor with the 'views.py' file open. The file contains Python code for a Django application. The code defines a view function 'home\_screen\_view' that handles POST requests. It uses a user input form to get BMI and SleepTime values, performs ML calculations, preprocesses the data, selects features, and uses logistic regression to predict heart disease. The prediction result is then displayed to the user. The code editor has a sidebar with project files like 'settings.py', 'models.py', 'base.html', and 'urls.py'. The bottom status bar shows the file path 'PS C:\Users\Fjwu\myproject>', line 35, column 1, and other system information.

```

hd1 > views.py > home_screen_view
17
18     SleepTime = form.cleaned_data['SleepTime']
19
20     # Perform your ML calculations here based on BMI and SleepTime
21     # Assuming df is your preprocessed dataset
22     df = Preprocessing.preprocessing_data()
23
24     # Assuming you have already done feature selection and stored it in selected_features
25     selected_features = FeatureSelection.decision_tree_feature_selection(df)
26
27     # Predict using logistic regression
28     X_input = np.array([[BMI, SleepTime]]) # Assuming these are the features needed for prediction
29     prediction = ConfusionMatrix.logistic_regression_predict(df, X_input, selected_features)
30
31     str = ""
32     if prediction == 0:
33         str = "Congratulations! You do not have Heart Disease"
34     elif prediction == 1:
35         str = "Ops! You have Heart Disease"
36
37     result = f"Prediction: {str}"
38
39     else:
40         form = UserInputForm()
41
42     return render(request, 'base.html', {'form': form, 'result': result, 'confusion_matrix': confusion_matrix_result})

```

This Django view function processes user inputs for BMI and SleepTime, then uses a preprocessed dataset and decision tree feature selection to predict heart disease with logistic regression. The result, indicating the prediction outcome (either having or not having heart disease), is displayed on the web page along with the corresponding confusion matrix. The code is part of a Django web application, providing a user-friendly interface for heart disease prediction.

## Urls.py

""""

*URL configuration for myproject project.*

*The `urlpatterns` list routes URLs to views. For more information please see:*

<https://docs.djangoproject.com/en/5.0/topics/http/urls/>

*Examples:*

*Function views*

1. Add an import: `from my_app import views`
2. Add a URL to urlpatterns: `path('', views.home, name='home')`

*Class-based views*

1. Add an import: `from other_app.views import Home`
2. Add a URL to urlpatterns: `path('', Home.as_view(), name='home')`

*Including another URLconf*

1. Import the `include()` function: `from django.urls import include, path`
2. Add a URL to urlpatterns: `path('blog/', include('blog.urls'))`

""""

```

from django.contrib import admin
from django.urls import path
from hd1.views import home_screen_view
from django.conf import settings
from django.conf.urls.static import static

```

```

urlpatterns = [
    path("admin/", admin.site.urls),
]

```

```

path("", home_screen_view),
]

# Add the following line to serve static files during development
if settings.DEBUG:
    urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)

```

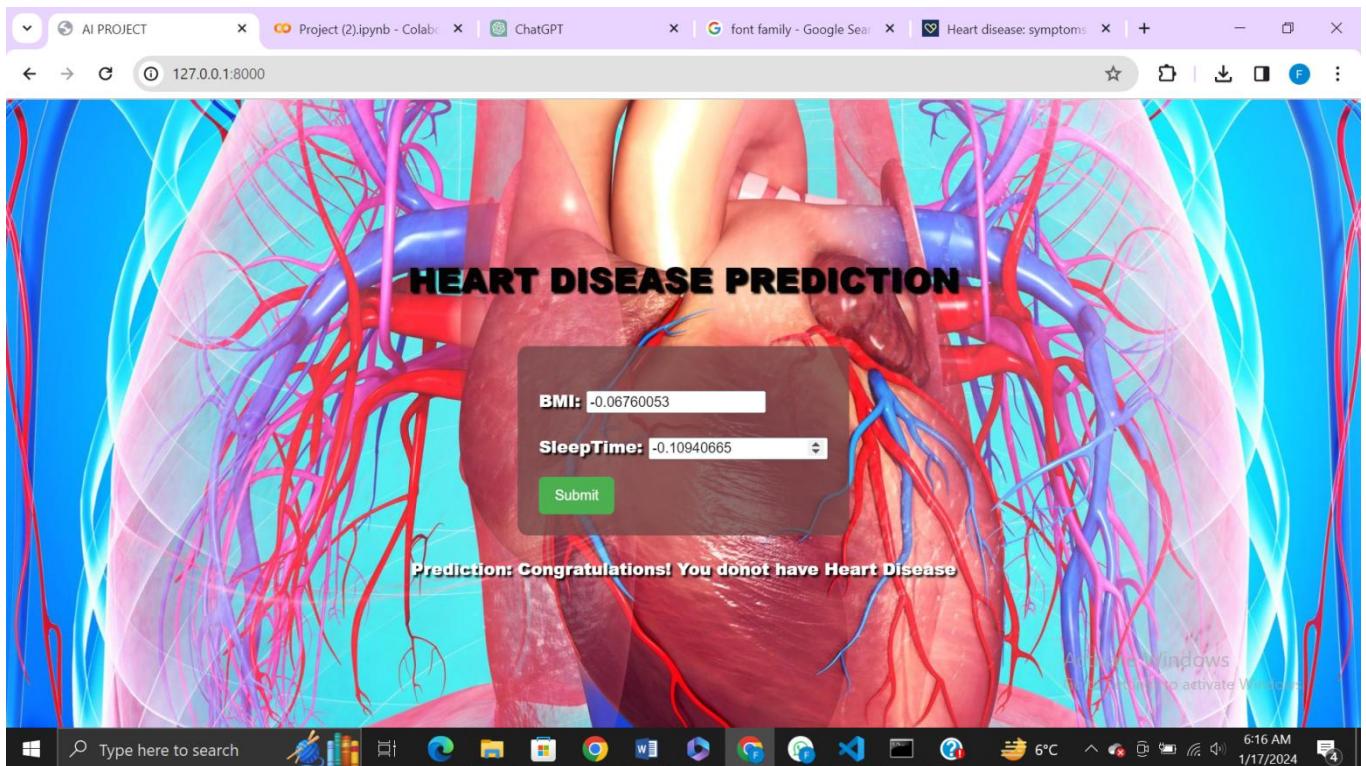
```

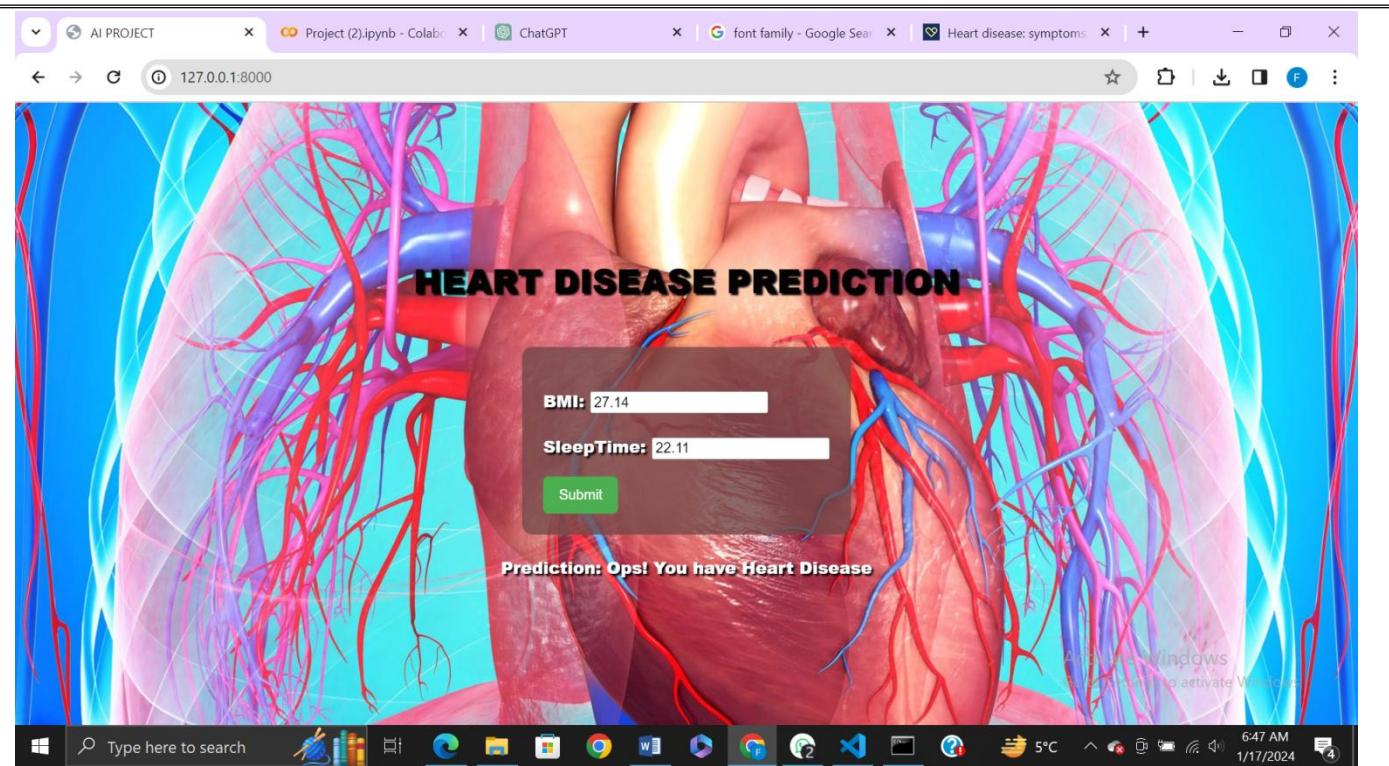
File Edit Selection View Go Run ... myproject
EXPLORER settings.py models.py base.html forms.py views.py
MYPYJECT static heart_image.jpg __init__.py admin.py apps.py forms.py heart_2020_1.csv models.py tests.py views.py
myproject _pycache_ __init__.py asgi.py settings.py urls.py wsgi.py
templates base.html db.sqlite3 manage.py python requirement.txt
OUTLINE TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Fjju\myproject> + ... x
Activate Windows Go to Settings to activate Windows Python
In 25, Col 32 Spaces: 4 UTF-8 CRLF Python 3.11.3 64-bit 6:16 AM 1/17/2024
Type here to search

```

This Django 'urls.py' configuration file defines URL patterns for a project named "myproject." The specified `urlpatterns` list includes a path for the admin interface and a default path leading to a view named 'home\_screen\_view'. Additionally, if the project is in debug mode, it appends a line to serve static files, enabling the handling of static content during development.

## Interface and Predicting through django





## 6. Predicate Logic

Predicate logic uses ***predicates and quantifiers*** to express relationships between variables. Let's denote the following predicates:

### General:

**Sleep(x)** : x represents the amount of sleep a person gets.

**BMI(x)** : x represents the Body Mass Index of a person.

**HeartDisease(x)**: x represents the presence of heart disease.

The statement is as follows:

$$\forall(x) \text{SleepTime}(x) \wedge \text{BMI}(x) \rightarrow \text{HeartDisease}(x)$$

This can be read statement reads:

For all individuals x, if x has a certain amount of sleep ***SleepTime(x)*** and a specific BMI ***BMI(x)***, then x will have heart disease ***HeartDisease(x)***.

### Predicate Logic Rules/Statements to Predict Heart Disease

Let's create some predicate rules based on the provided data for predicting heart disease using sleep and BMI. We'll use predicates **SleepTime(x)**, **BMI(x)**, and **HeartDisease(x)** to represent the sleep time, BMI, and presence of heart disease for an individual x, respectively. We can make some simple rules using predicates:

**1. For all persons who sleeps less than a certain amount of SleepTime and has a high BMI, then they are more likely to have heart disease.**

- A person sleeps less than a certain amount of Sleep Time:  $SleepTime(x) < \text{ThresholdSleepTime}$
- A person has high BMI than Threshold BMI:  $BMI(x) > \text{ThresholdBMI}$
- He is more likely to have heart disease:  $HeartDisease(x) = 1$

$$\forall(x) SleepTime(x) < \text{ThresholdSleepTime} \wedge BMI(x) > \text{ThresholdBMI} \rightarrow HeartDisease(x) = 1$$

For example:

If ThresholdSleepTime is 7 hours then:

$$\forall(x) SleepTime(x) < 7hrs \wedge BMI(x) > \text{ThresholdBMI} \rightarrow HeartDisease(x) = 1$$

**2. For all persons who sleeps a sufficient amount of SleepTime is greater than or equal to ThresholdSleepTime and has a normal BMI is less than or equal to ThresholdBMI, then they are less likely to have heart disease.**

- If a person SleepTime greater than required:  $SleepTime(x) \geq \text{ThresholdSleepTime}$
- If a person has normal BMI less than ThresholdBMI:  $BMI(x) \leq \text{ThresholdBMI}$
- They are less likely to have heart disease:  $HeartDisease(x) = 0$

$$\forall(x) SleepTime(x) \geq \text{ThresholdSleepTime} \wedge BMI \leq \text{ThresholdBMI} \rightarrow HeartDisease(x) = 0$$

For example:

If ThresholdSleepTime is 7 hours then:

$$\forall x (SleepTime(x) \geq 7hrs \wedge BMI(x) \leq \text{ThresholdBMI}) \rightarrow HeartDisease(x) = 0$$

**3. There exists an individual who sleeps less than required and has a high BMI, indicating a higher likelihood of heart disease.**

- If a person SleepTime is less than required:  $SleepTime(x) < \text{ThresholdSleepTime}$
- If a person has high BMI than required:  $BMI(x) > \text{ThresholdBMI}$
- They are more likely to have heart disease:  $HeartDisease(x) = 1$

$$\exists(x) SleepTime(x) < \text{ThresholdSleepTime} \wedge BMI(x) > \text{ThresholdBMI} \rightarrow HeartDisease(x) = 1$$

**For example:**

If ThresholdSleepTime is 7 hours then:

$$\exists(x) \text{SleepTime}(x) < 7\text{hrs} \wedge \text{BMI}(x) > \text{ThresholdBMI} \rightarrow \text{HeartDisease}(x) = 1$$

**4. There exists an individual who sleeps 7 hours or more and has a normal BMI, indicating a lower likelihood of heart disease.**

- If a person has Sleeps required or more than required:  $\text{SleepTime}(x) \geq \text{ThresholdSleepTime}$
- If a person has normal BMI than required:  $\text{BMI}(x) > \text{ThresholdBMI}$
- They are more likely to have heart disease:  $\text{HeartDisease}(x) = 0$

$$\exists(x) \text{SleepTime}(x) \geq \text{ThresholdSleepTime} \wedge \text{BMI}(x) \leq \text{ThresholdBMI} \wedge \text{HeartDisease}(x) = 0$$

**For example:**

If ThresholdSleepTime is 7 hours then:

$$\exists(x) \text{SleepTime}(x) \geq 7\text{hrs} \wedge \text{BMI}(x) \leq \text{ThresholdBMI} \wedge \text{HeartDisease}(x) = 0$$

**5. Not an individual exists who has a BMI greater than a thresholdBMI and sleeps 7 hours or more.**

- If no person has BMI greater than required:  $\neg \exists x \text{BMI}(x) \geq \text{ThresholdBMI}$
- If no person has normal sleep or more:  $\neg \exists x \text{SleepTime}(x) \geq 7$

$$\neg \exists(x) \text{BMI}(x) > \text{ThresholdBMI} \wedge \text{SleepTime}(x) \geq 7$$

This statement asserts that there is no individual in the dataset who both has a BMI above a certain threshold and sleeps 7 hours or more. In other words, it expresses a negation, indicating the absence of individuals meeting these specific conditions.

---