

PROJECT REPORT

ARTIFICIAL
INTELLIGENCE
ANALYTICS



AREESHA RUBAB
(2021-BSE-041)

FAIZA SYED
(2021-BSE-046)

NOOR UL AIN
(2021-BSE-057)

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”

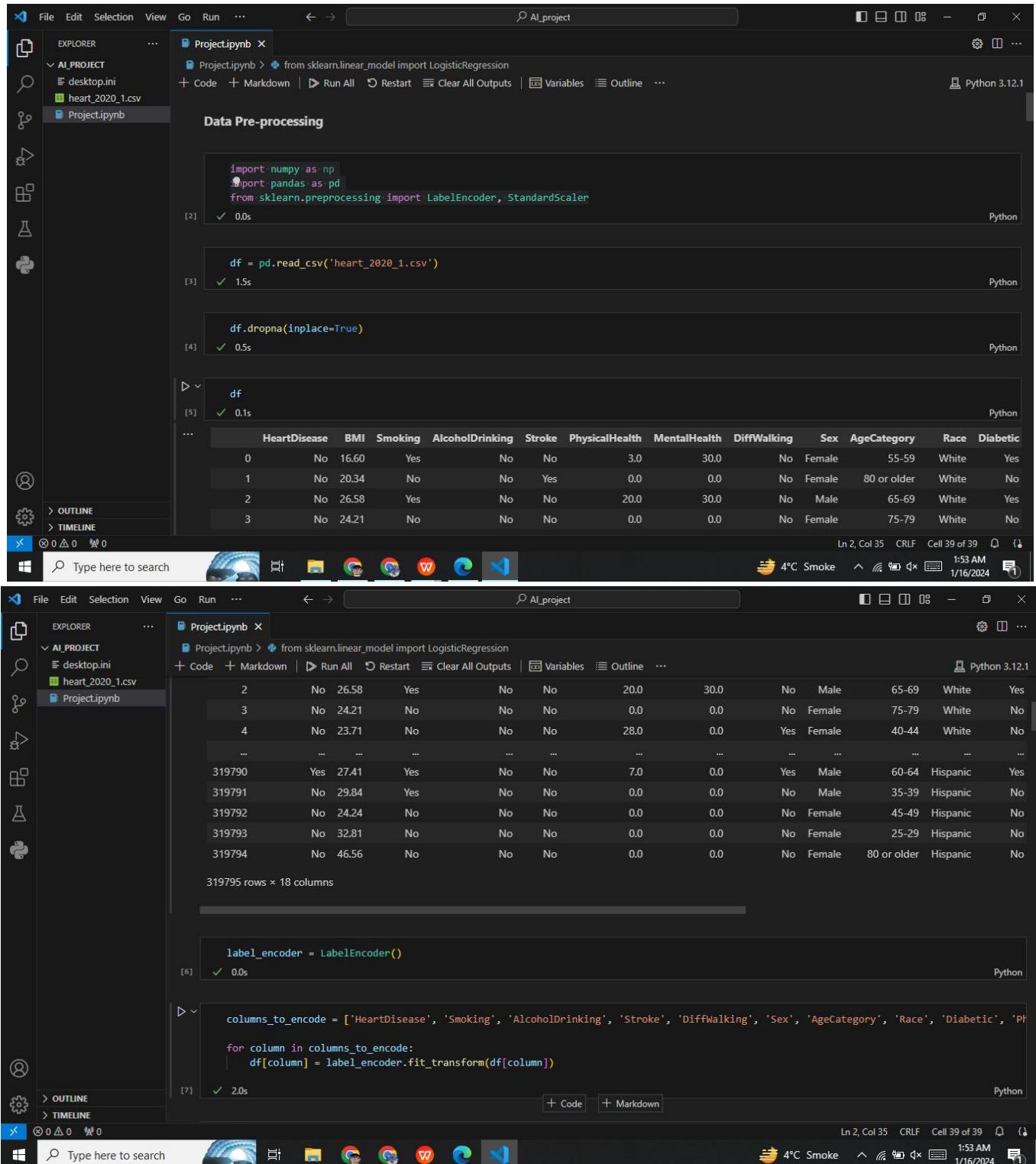
Our dataset, named "heart_2020_1.csv," encompasses details regarding the health and lifestyle factors of individuals. This dataset will train or test machine learning models that can predict certain health outcomes based on the variables. Comprising 18 variables, each representing a distinct aspect of health or behavior, the dataset is employed for training a model aimed at predicting whether a person is afflicted with heart disease or not.

HeartDisease	BMI	Smoking	AlcoholDrn	Stroke	PhysicalHeal	MentalHeal	DifWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActi	GenHealth	SleepTime	Asthma	KidneyDiseases	
No	16.6	Yes	No	No	3	30	No	Female	55-59	White	Yes	Yes	Very good	5	Yes	No	
No	20.34	No	No	Yes	0	0	No	Female	80 or older	White	No	Yes	Very good	7	No	No	
No	26.58	Yes	No	No	20	30	No	Male	65-69	White	Yes	Yes	Fair	8	Yes	No	
No	24.21	No	No	No	0	0	No	Female	75-79	White	No	No	Good	6	No	Yes	
No	23.71	No	No	No	28	0	Yes	Female	40-44	White	No	Yes	Very good	8	No	No	
Yes	28.87	Yes	No	No	6	0	Yes	Female	75-79	Black	No	No	Fair	12	No	No	
No	21.63	No	No	No	15	0	No	Female	70-74	White	No	Yes	Fair	4	Yes	No	
No	31.64	Yes	No	No	5	0	Yes	Female	80 or older	White	Yes	No	Good	9	Yes	No	
No	26.45	No	No	No	0	0	No	Female	80 or older	White	No	borderline	No	Fair	5	No	Yes
No	40.69	No	No	No	0	0	Yes	Male	65-69	White	No	Yes	Good	10	No	No	
Yes	34.3	Yes	No	No	30	0	Yes	Male	60-64	White	Yes	No	Poor	15	Yes	No	
No	28.71	Yes	No	No	0	0	No	Female	55-59	White	No	Yes	Very good	5	No	No	
No	28.37	Yes	No	No	0	0	Yes	Male	75-79	White	Yes	Yes	Very good	8	No	No	
No	28.15	No	No	No	7	0	Yes	Female	80 or older	White	No	No	Good	7	No	No	
No	29.29	Yes	No	No	0	30	Yes	Female	60-64	White	No	No	Good	5	No	No	
No	29.18	No	No	No	1	0	No	Female	50-54	White	No	Yes	Very good	6	No	No	
No	26.26	No	No	No	5	2	No	Female	70-74	White	No	No	Very good	10	No	No	
No	22.59	Yes	No	No	0	30	Yes	Male	70-74	White	No	borderline	Yes	Good	8	No	No
No	29.86	Yes	No	No	0	0	Yes	Female	75-79	Black	Yes	No	Fair	5	No	Yes	
No	18.13	No	No	No	0	0	No	Male	80 or older	White	No	Yes	Excellent	8	No	Yes	
No	21.16	No	No	No	0	0	No	Female	60-64	White	Yes	No	Good	7	No	No	

N103476	GenHealth	Asthma
103457	Good	No
103458	Very good	No
103459	Very good	No
103460	Very good	No
103461	Fair	No
103462	Excellent	No
103463	Excellent	No
103464	Excellent	No
103465	Fair	No
103466	Fair	Yes
103467	Fair	No
103468	Very good	No
103469	Good	No
103470	Fair	No
103471	Excellent	No
103472	Excellent	No
103473	Good	No
103474	Good	No
103475	Good	No
103476	Excellent	No
103477	Very good	No
103478	Very good	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 two Jupyter Notebook sessions. The top session displays the initial data loading and cleaning steps:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler

df = pd.read_csv('heart_2020_1.csv')
df.dropna(inplace=True)
```

The bottom session shows the application of a LabelEncoder to categorical columns:

```
label_encoder = LabelEncoder()

columns_to_encode = ['HeartDisease', 'Smoking', 'AlcoholDrinking', 'Stroke', 'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'Sex', 'AgeCategory', 'Race', 'Diabetic']

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

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.

The screenshot shows a Jupyter Notebook interface with a dark theme. In the top right cell, the code `df` is run, resulting in a DataFrame named 'df'. The output shows the first 10 rows of the dataset, which contains 319795 rows and 18 columns. The columns include HeartDisease, BMI, Smoking, AlcoholDrinking, Stroke, PhysicalHealth, MentalHealth, DiffWalking, Sex, AgeCategory, Race, Diabetic, and Phys. The data shows various numerical values for these features across different rows.

```

Project.ipynb x
Project.ipynb > from sklearn.linear_model import LogisticRegression
+ Code + Markdown | ▶ Run All ⚡ Restart ⚡ Clear All Outputs | Variables Outline ...
[7] ✓ 2.0s
df
[8]
...
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'.

The screenshot shows a Jupyter Notebook interface with a dark theme. In the top right cell, the code scales numerical columns using StandardScaler and then performs feature selection using SelectKBest and f_classif. The resulting DataFrame 'df' is displayed, showing scaled numerical values and the original categorical and binary columns. The data includes columns like HeartDisease, BMI, Smoking, AlcoholDrinking, Stroke, PhysicalHealth, MentalHealth, DiffWalking, Sex, AgeCategory, Race, and Diabetic.

```

Project.ipynb x
Project.ipynb > from sklearn.linear_model import LogisticRegression
+ Code + Markdown | ▶ Run All ⚡ Restart ⚡ Clear All Outputs | Variables Outline ...
[9] ✓ 0.1s
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
[10] ✓ 1.9s

```

```

df
[11]
...
HeartDisease BMI Smoking AlcoholDrinking Stroke PhysicalHealth MentalHealth DiffWalking Sex AgeCategory Race Diabetic
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 -0.424070 -0.490039 -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 -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 corner, there's a progress bar indicating the status of a cell. The notebook has one open cell titled "Project.ipynb". The code in the cell uses pandas, matplotlib, and seaborn to load a dataset, drop the target variable, subsample 1000 rows, and create a strip plot of scaled features. The output of the cell is a "Strip Plot of Scaled Features (Subsample)" which is displayed below the code cell.

```

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

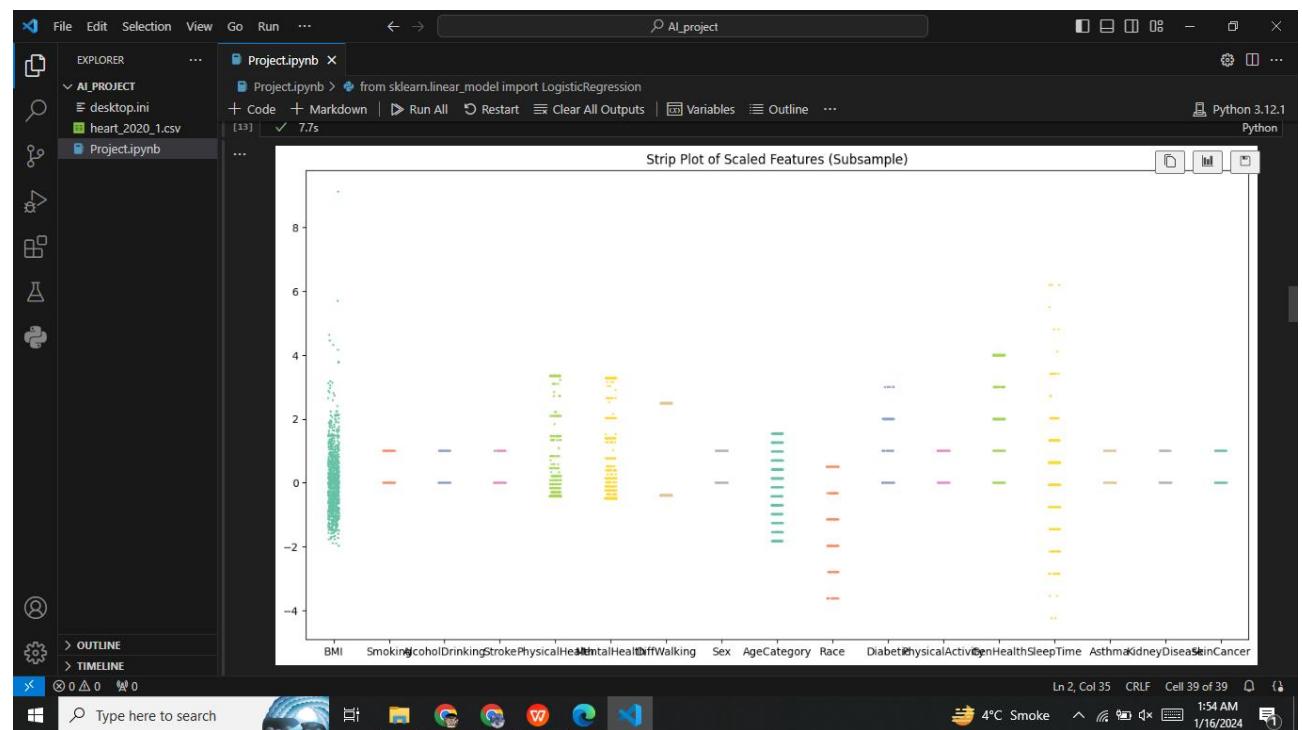
# Exclude the target variable (assuming it's 'HeartDisease') from the visualization
features = df.drop('HeartDisease', axis=1)

# 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()

```

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 cells numbered 14 through 17. Cell 14 takes 1.6s, cell 15 takes 0.3s, cell 16 takes 0.0s, and cell 17 takes 5.4s. The status bar at the bottom right shows 'Ln 2, Col 35 CRLF Cell 39 of 39' and the date '1/16/2024'.

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)

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

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

The code is run in cells numbered 17 through 20. Cell 17 takes 5.4s, cell 18 takes 6.0s, cell 19 takes 0.0s, and cell 20 takes 0.0s. The status bar at the bottom right shows 'Ln 2, Col 35 CRLF Cell 39 of 39' and the date '1/16/2024'.

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:

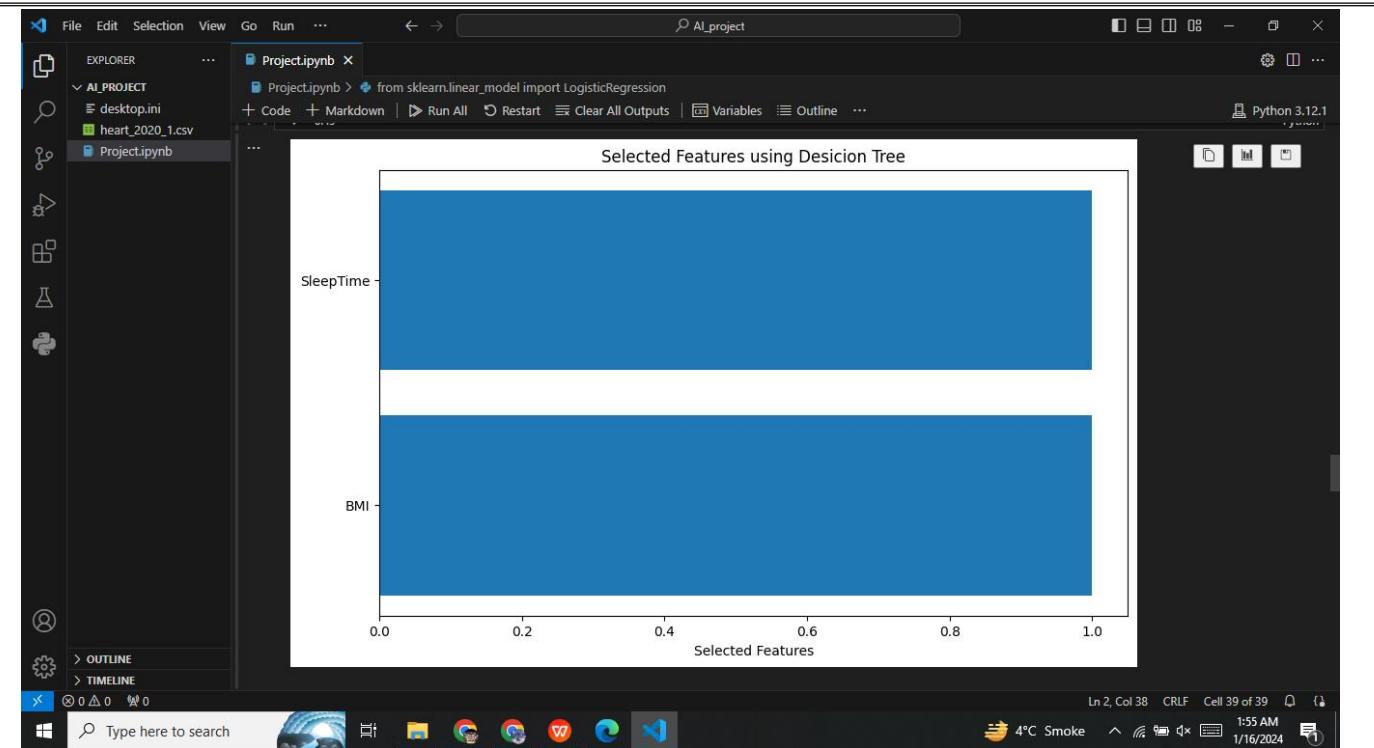
```
# Make predictions on the test set using the selected features
y_pred = clf.predict(X_test_selected)
[23] ✓ 0.0s
...
from sklearn.metrics import accuracy_score
[27] ✓ 0.0s

# Evaluate the performance of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on the test set: {accuracy:.2f}")
[28] ✓ 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()
[31] ✓ 0.4s
```

A horizontal bar chart titled "Selected Features using Desicion Tree" is displayed, showing two blue bars corresponding to the selected features "BMI" and "SleepTime". 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 x 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

```

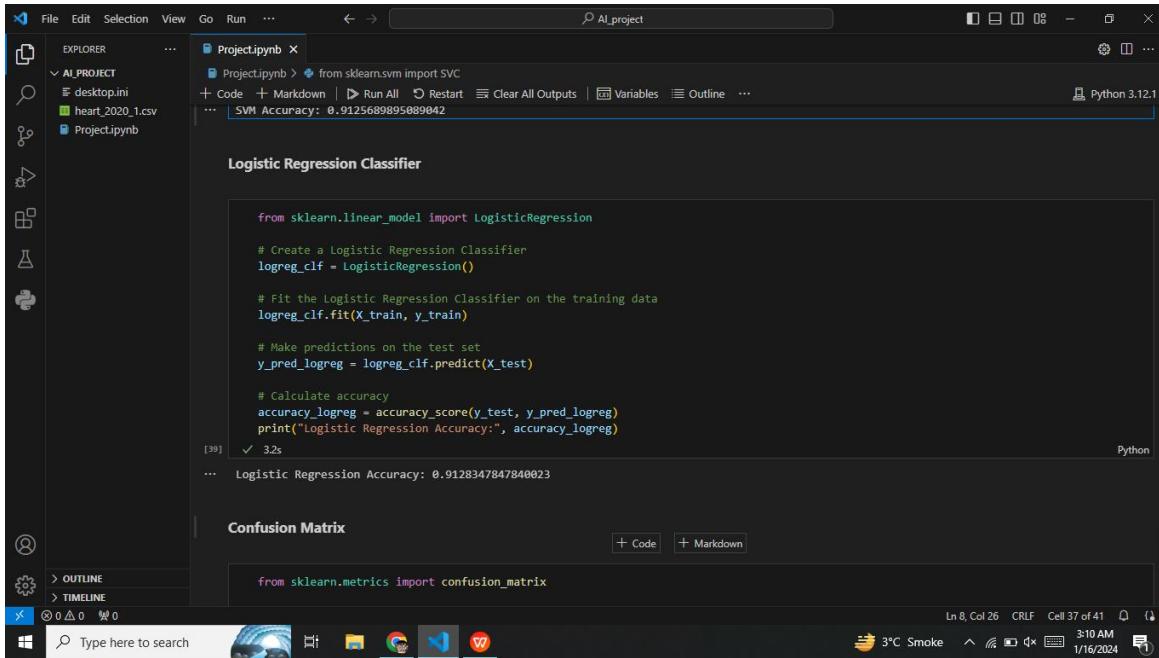
from sklearn.neighbors import KNeighborsClassifier
# Create a KNN Classifier
knn_clf = KNeighborsClassifier(n_neighbors=5) # You can adjust the number of neighbors

```

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 a dark theme. The left sidebar has an 'EXPLORER' section showing a project named 'AI_PROJECT' containing 'desktop.ini', 'heart_2020_1.csv', and 'Project.ipynb'. The main area displays Python code for a Logistic Regression Classifier. The code imports 'LogisticRegression' from 'sklearn.linear_model', creates a classifier, fits it to training data, makes predictions on a test set, calculates accuracy, and prints the result. The output shows an accuracy of 0.9128347847840023. Below the code cell is a 'Confusion Matrix' cell with placeholder code.

```
from sklearn.linear_model import LogisticRegression

# Create a Logistic Regression Classifier
logreg_clf = LogisticRegression()

# Fit the Logistic Regression Classifier on the training data
logreg_clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred_logreg = logreg_clf.predict(X_test)

# Calculate accuracy
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
print("Logistic Regression Accuracy:", accuracy_logreg)
```

3.2s

```
... Logistic Regression Accuracy: 0.9128347847840023
```

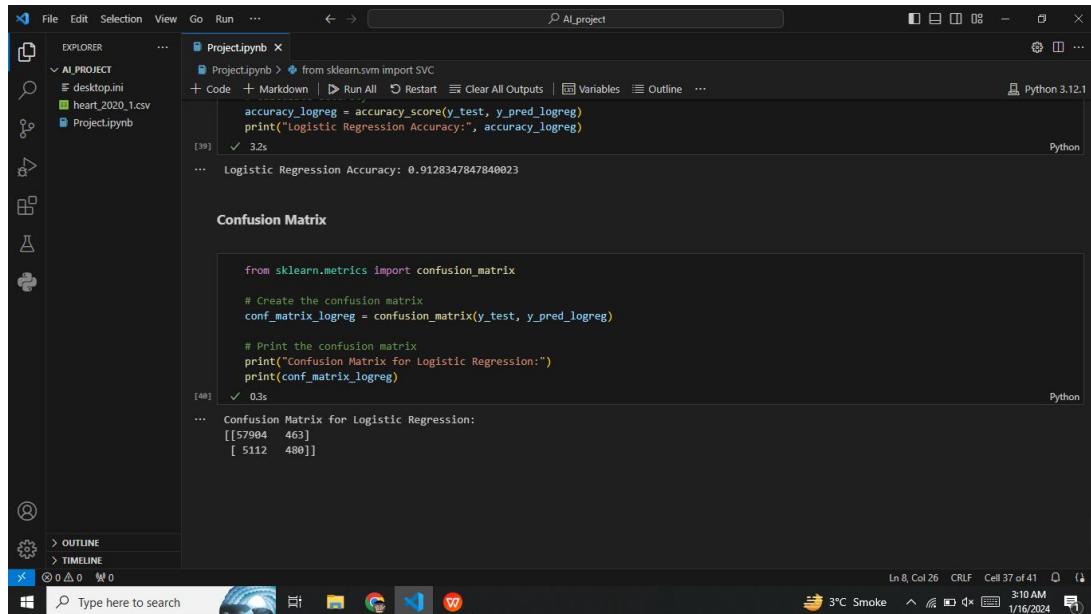
Confusion Matrix

+ Code + Markdown

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 a dark theme. The left sidebar has an 'EXPLORER' section showing a project named 'AI_PROJECT' containing 'desktop.ini', 'heart_2020_1.csv', and 'Project.ipynb'. The main area displays Python code for calculating a confusion matrix. It imports 'confusion_matrix' from 'sklearn.metrics', creates a confusion matrix for the test set, and prints it. The output shows a confusion matrix for Logistic Regression with values [[57904 463], [5112 486]].

```
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
print("Logistic Regression Accuracy:", accuracy_logreg)
```

3.2s

```
... Logistic Regression Accuracy: 0.9128347847840023
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix

# Create the confusion matrix
conf_matrix_logreg = confusion_matrix(y_test, y_pred_logreg)

# Print the confusion matrix
print("Confusion Matrix for Logistic Regression:")
print(conf_matrix_logreg)
```

0.3s

```
... Confusion Matrix for Logistic Regression:
[[57904 463]
 [ 5112 486]]
```

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 the Django framework.

- Screenshot 1:** Shows the initial command `C:\Users\AZAN LAPTOP STORE>python --version` which outputs "Python 3.12.1".
- Screenshot 2:** Shows the command `C:\Users\AZAN LAPTOP STORE>pip install django` being run. The output indicates that pip is trying to install Django because it's not writeable to the site-packages directory. It lists several dependencies being downloaded, including asgiref, sqlparse, and tzdata, along with their file sizes and download times.
- Screenshot 3:** Shows the command `C:\Users\AZAN LAPTOP STORE>pip install pipenv` being run. The output shows the installation of various pipenv dependencies like certifi, requests, and virtualenv, along with their file sizes and download times.
- Screenshot 4:** Shows the command `C:\Users\AZAN LAPTOP STORE>django-admin --version` being run. The output shows that 'django-admin' is not recognized as an internal or external command. It then shows the command `C:\Users\AZAN LAPTOP STORE>setx PATH "%PATH%;C:\path\to\your\django\scripts"` being run to set the environment variable. Finally, it shows the command `C:\Users\AZAN LAPTOP STORE>python -m django --version` being run, which successfully outputs "5.0.1".

```

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>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>cd ..
C:\Users\AZAN LAPTOP STORE>AppData\Roaming\Python\Python312>cd ..
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

```



[View release notes for Django 5.0](#)

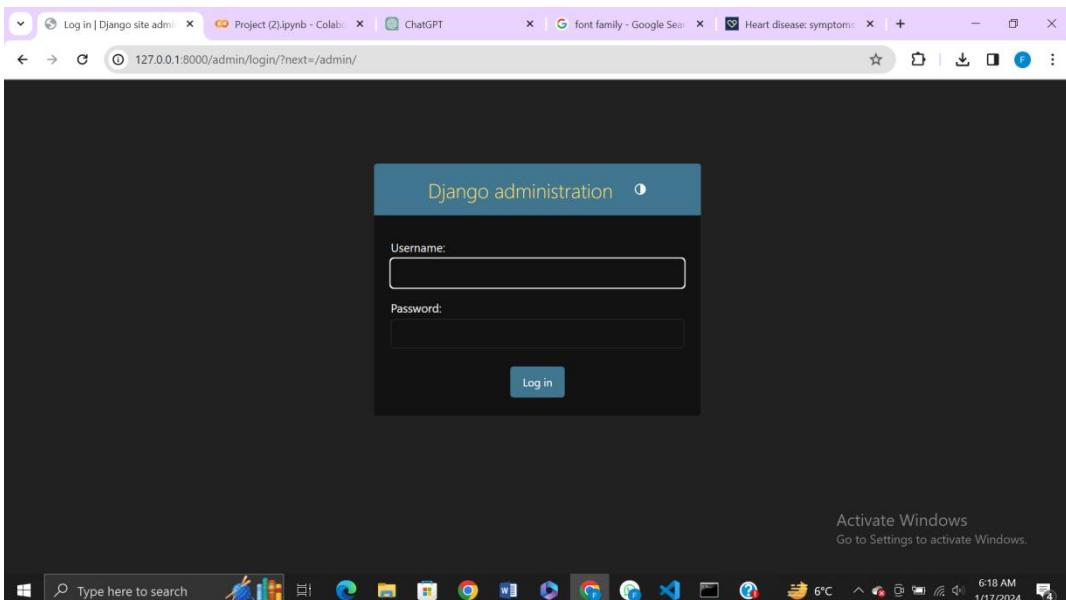


The install worked successfully! Congratulations!

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

"hd1",

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-passwordValidators

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"

The screenshot shows a code editor with the file `settings.py` open. The code defines various settings for a Django application, including middleware, URLs, and template configurations. The code editor has a sidebar with project files like `models.py`, `base.html`, `heart_image.jpg`, `forms.py`, `views.py`, and `urls.py`. The bottom status bar shows the file path `PS C:\Users\Fjju\myproject>` and the Python version `Python 3.11.3 64-bit`.

```
File Edit Selection View Go Run ... ← → myproject  
EXPLORER ... settings.py x models.py base.html heart_image.jpg forms.py views.py urls.py D ...  
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  
48 "django.middleware.security.SecurityMiddleware",  
49 "django.contrib.sessions.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 ]  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
Activate Windows Go to Settings to activate Windows Python 3.11.3 64-bit  
OUTLINE TIMELINE PS C:\Users\Fjju\myproject> 6:11 AM 1473 2024  
Type here to search
```

```

File Edit Selection View Go Run ... ← → myproject
EXPLORER ... settings.py X 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

```

```

116 USE_TZ = True
117
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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Fjwu\myproject> []

Activate Windows Python Go to Settings to activate Windows Python

Ln 123, Col 1 (22 selected) Spaces: 4 UTF-8 CRLF Python 3.11.3 64-bit 6:11 AM 1/17/2024

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]

```

$y = df['HeartDisease']$
 $clf.fit(X, y)$

$prediction = clf.predict(X_input)$
 $return prediction$

```

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
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Fjwu\myproject> Activate Windows
Go to Settings to activate Windows Python
In 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
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Fjwu\myproject> Activate Windows
Go to Settings to activate Windows Python
In 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 ...
PROJECT
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
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]

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Fjwu\myproject> []

Activate Windows Go to Settings to activate Windows Python

In 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 ...
PROJECT
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
@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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Fjwu\myproject> []

Activate Windows Go to Settings to activate Windows Python

In 89, Col 42 Spaces: 4 UTF-8 CRLF Python 3.11.3 64-bit 6:13 AM 1/17/2024

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'. The 'base.html' tab contains the HTML structure for a page, including a header with a background image and a large red 'h1' title. The 'style' tab contains the corresponding CSS code, which defines styles for the 'body' and 'h1' elements, including colors and shadows.

```
<!-- 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: #fffff;
            }
            h1 {
                color: #050303;
                text-shadow: 2px 2px 2px #000;
            }
        </style>
    </head>
    <body>
        <h1>AI PROJECT</h1>
        <form method="post">
            {% csrf_token %}
            {{ form.as_p }}
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

The screenshot shows a code editor interface with two tabs open: 'base.html' and 'style'. The 'base.html' tab contains the HTML structure for a page, including a header with a background image and a large red 'h1' title. The 'style' tab contains the corresponding CSS code, which defines styles for the 'body', 'h1', and 'input[type="submit"]' elements, including colors, shadows, and transitions.

```
<!-- 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-color: #1a1a1a;
                color: #fff;
                font-family: 'Arial Black', sans-serif;
                margin: 0;
                padding: 0;
                display: flex;
                flex-direction: column;
                align-items: center;
                justify-content: center;
                height: 100vh;
                background-image: url('{% static "heart_image.jpg" %}');
                background-size: cover;
            }
            h1 {
                color: #050303;
                text-shadow: 2px 2px 2px #000;
            }
            form {
                background-color: #1a1a1a;
                padding: 20px;
                border-radius: 10px;
                box-shadow: 0 0 10px #1a1a1a;
                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: #3e8a4d;
            }
        </style>
    </head>
    <body>
        <h1>AI PROJECT</h1>
        <form method="post">
            {% csrf_token %}
            {{ form.as_p }}
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

The screenshot shows a code editor interface with the file `base.html` open. The code defines a basic HTML structure with a header containing a background image, a title, and a form for user input. It also includes Django template tags for handling form submission and displaying results.

```

<head>
    <input type="submit":hover {
        background-color: #45a649;
    }

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

```

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 of a new form class, `UserInputForm`, which inherits from `forms.Form`. This form includes fields for `BMI` and `SleepTime`.

```

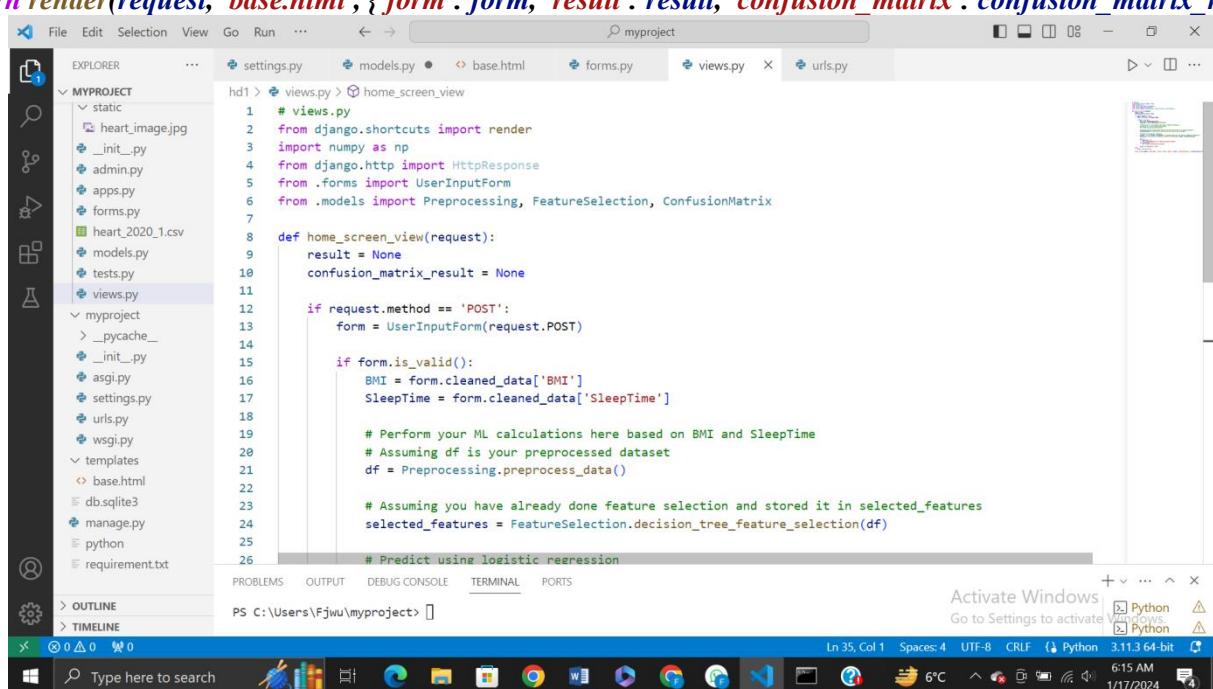
# forms.py
from django import forms

class UserInputForm(forms.Form):
    BMI = forms.FloatField(label='BMI')
    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 also shows the project structure with files like 'settings.py', 'models.py', 'base.html', and 'urls.py'.

```

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)

```

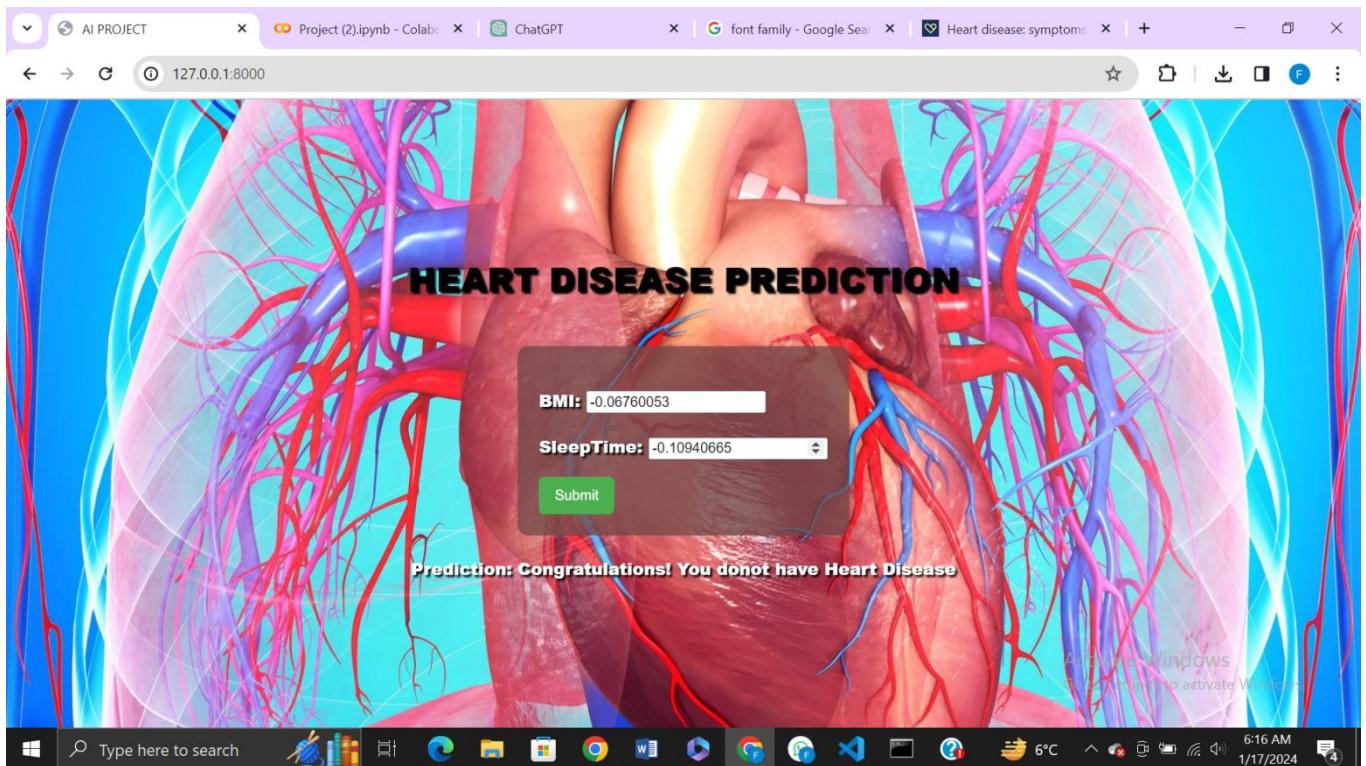
```

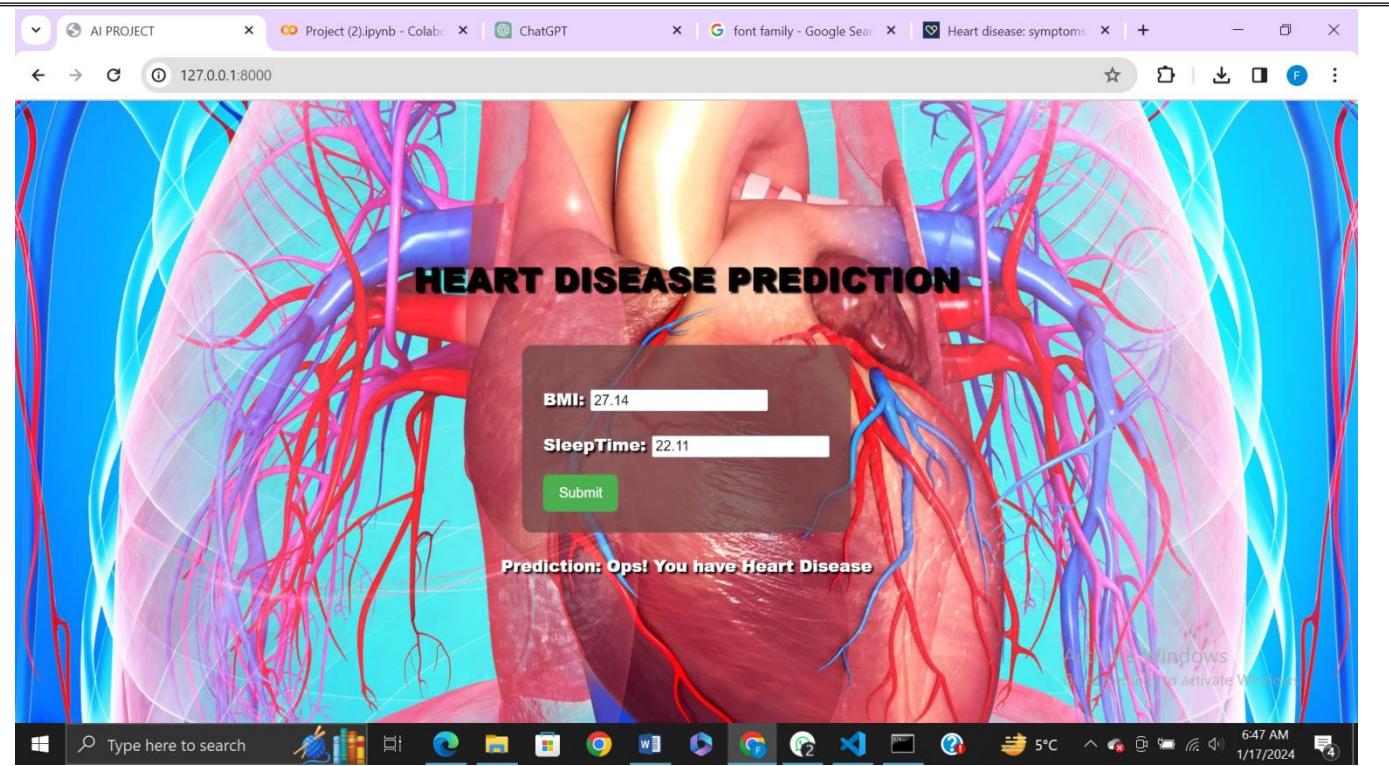
File Edit Selection View Go Run ... myproject
EXPLORER settings.py models.py base.html forms.py views.py
MYPY PROJECT static heart_image.jpg __init__.py admin.py apps.py forms.py heart_2020_1.csv models.py tests.py views.py
myproject _pycache_ __init__.py asgi.py settings.py urls.py wsgi.py
templates 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: $SleepTime(x) < ThresholdSleepTime$
- A person has high BMI than Threshold BMI: $BMI(x) > ThresholdBMI$
- He is more likely to have heart disease: $HeartDisease(x) = 1$

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

For example:

If ThresholdSleepTime is 7 hours then:

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

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

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

For example:

If ThresholdSleepTime is 7 hours then:

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

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

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

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.
