THEORY PROJECT

PROJECT REPORT

ARTIFICIAL INTELLIGENCE ANALYTICS

AREESHA RUBAB
(2021-BSE-041)

FAIZA SYED
(2021-BSE-046)

NOOR UL AIN
(2021-BSE-057)

TABLE OF CONTENTS

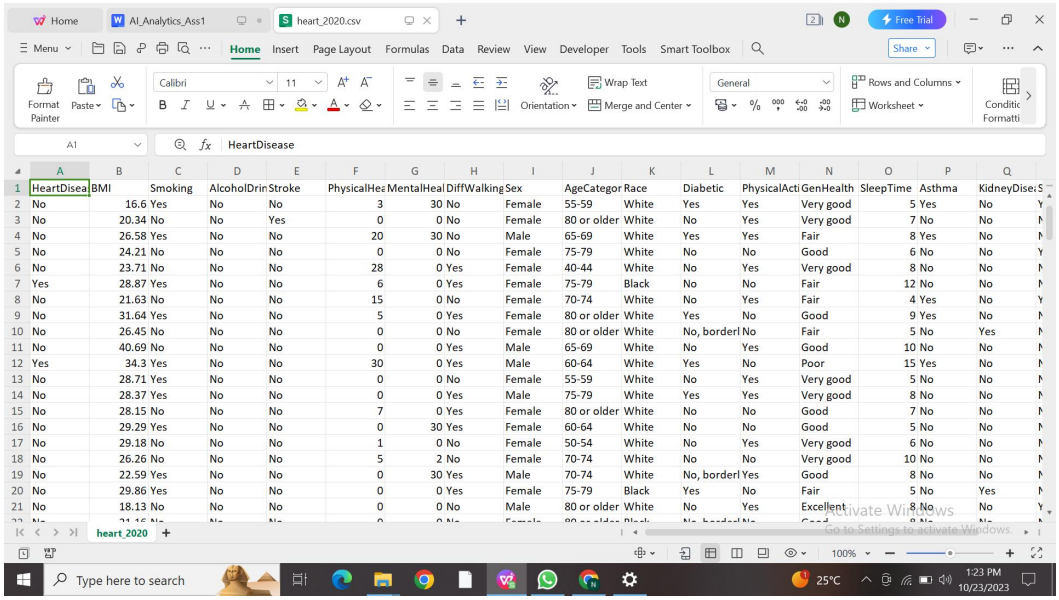
SEMESTER PROJECT

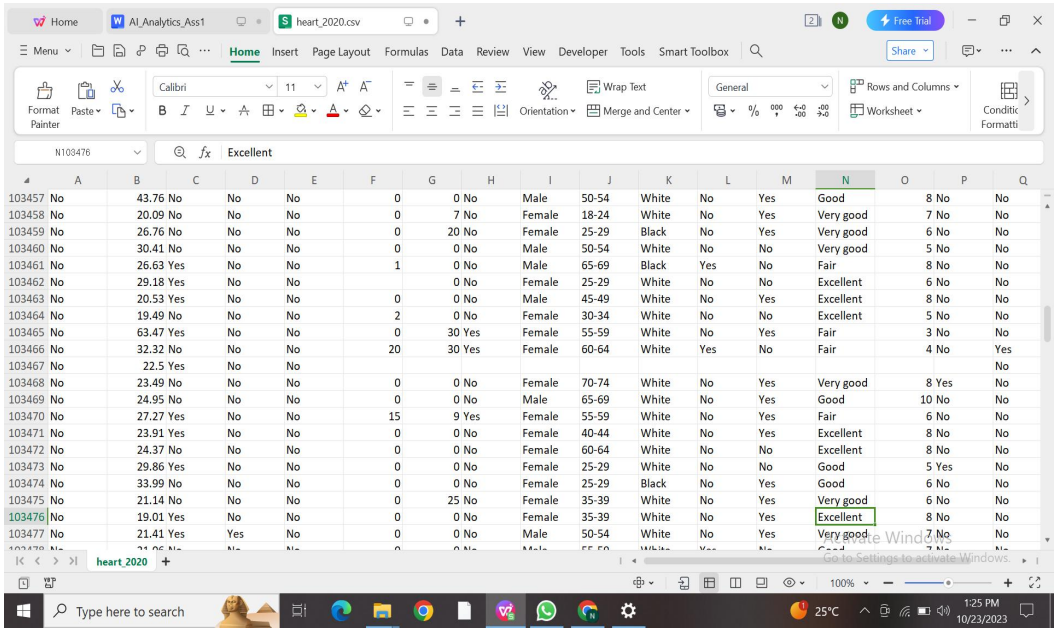
Apply
Data processing
Feature Selection
Classification
Performance metrics
Front using django

Dataset

“heart_2020_1.csv”

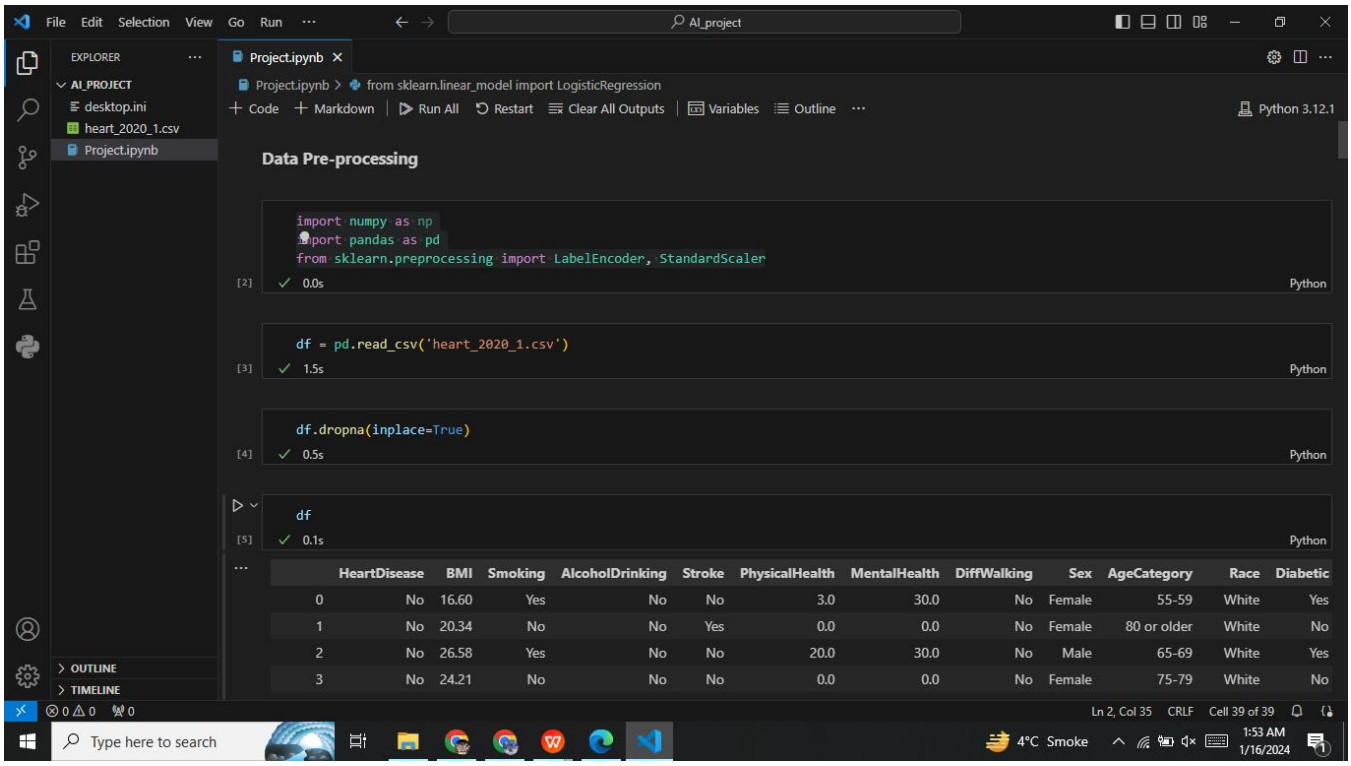
Our dataset, named "heart_2020_1.csv," encompasses details regarding the health and lifestyle factors of individuals. This dataset will train or test machine learning models that can predict certain health outcomes based on the variables. Comprising 18 variables, each representing a distinct aspect of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

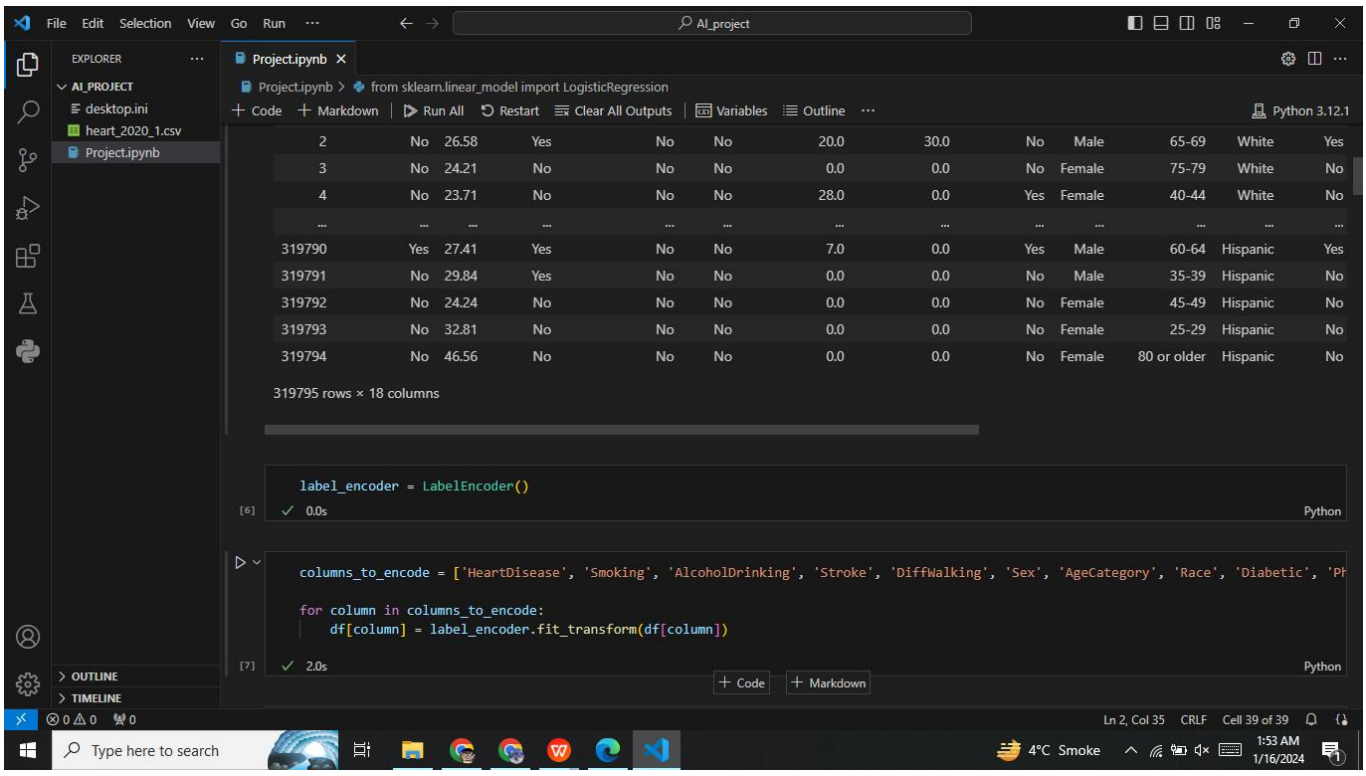




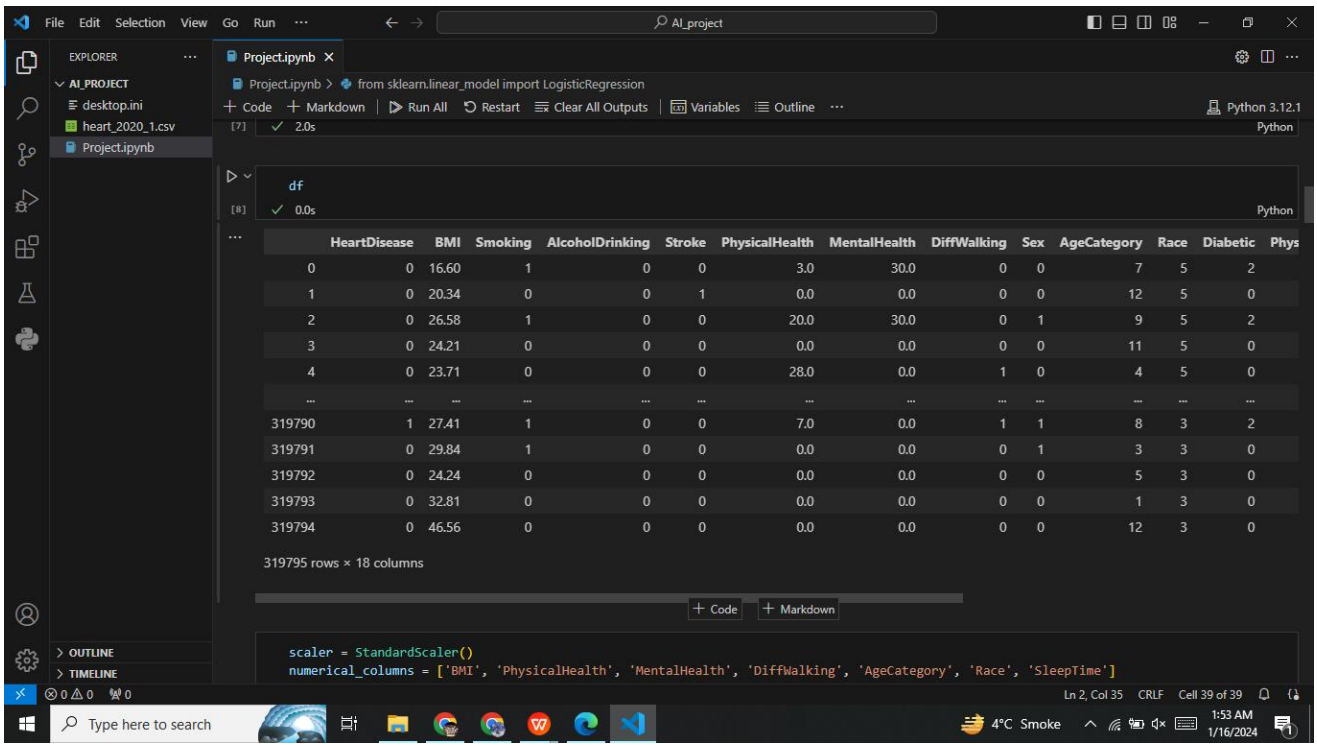
1. Data Pre-processing:

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.





This code uses a **LabelEncoder f**rom 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.



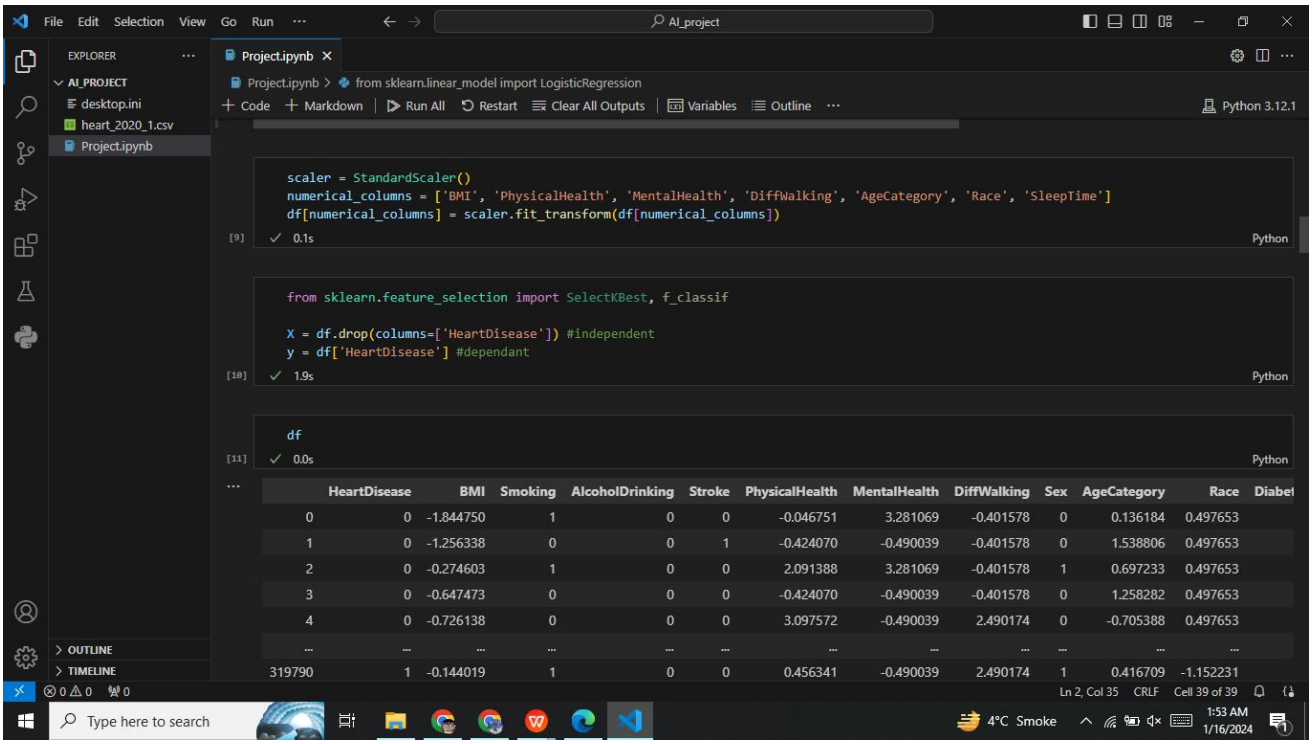
```
df
```

	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

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

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



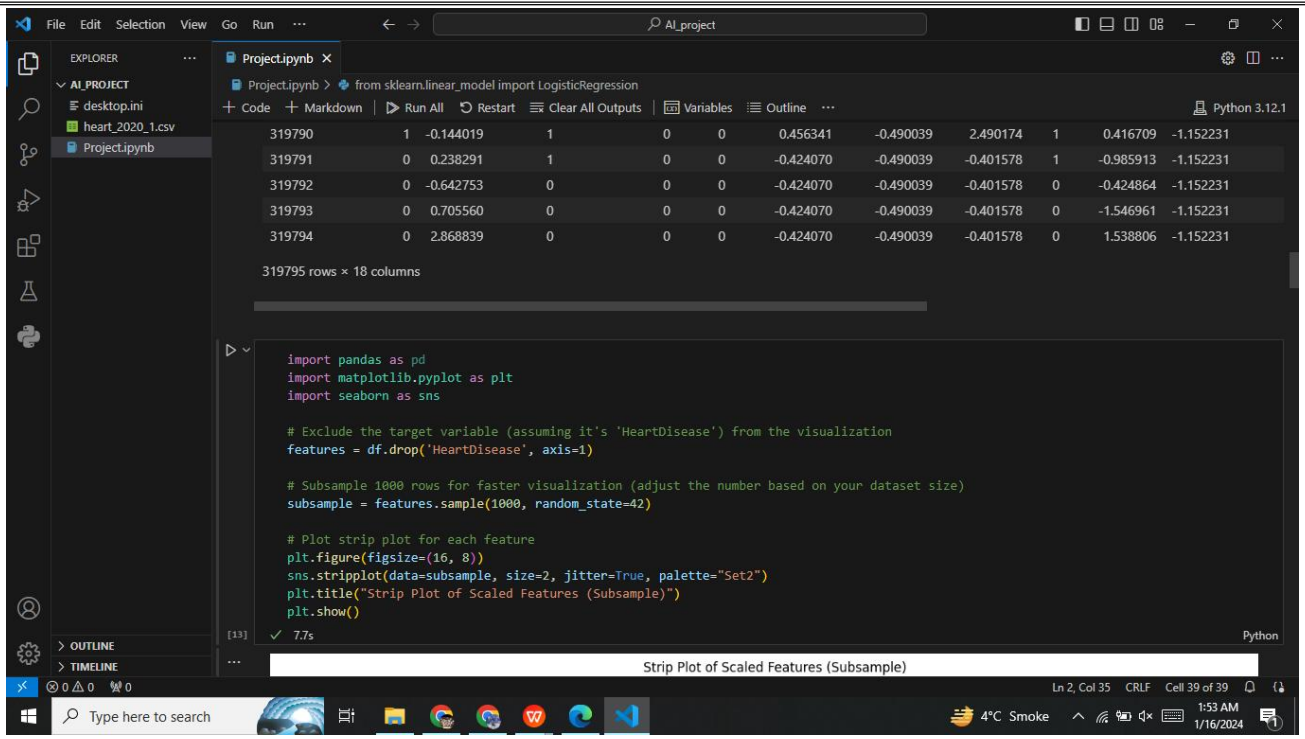
```
scaler = StandardScaler()
numerical_columns = ['BMI', 'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'AgeCategory', 'Race', 'SleepTime']
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
```

```
from sklearn.feature_selection import SelectKBest, f_classif

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

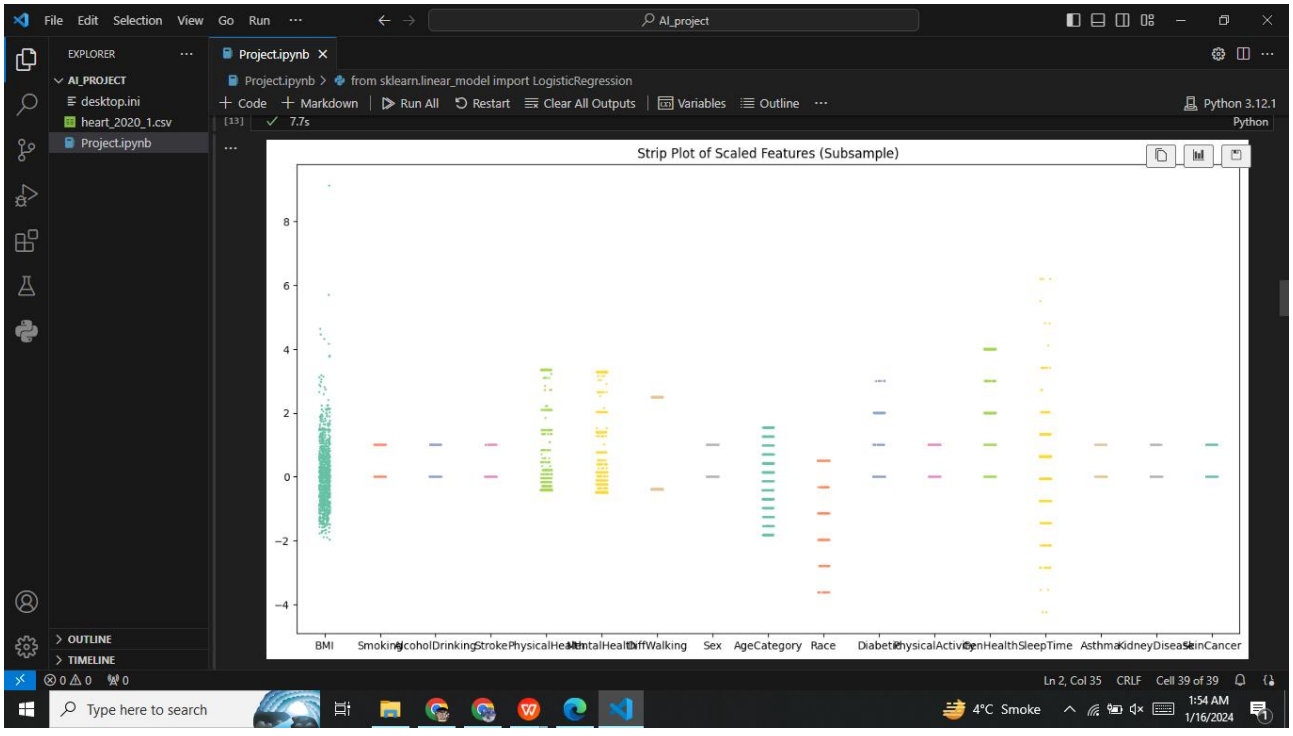
```
df
```

	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.



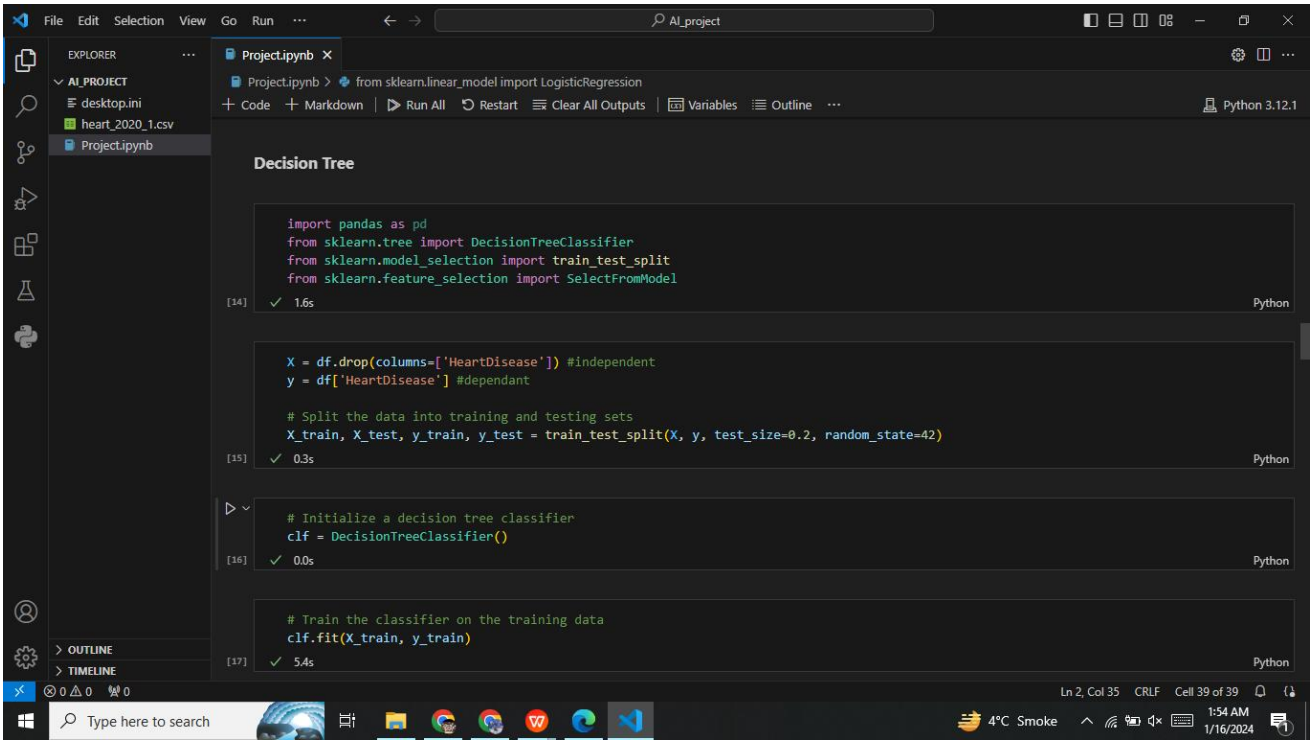
This Python code prepares a DataFrame by excluding the 'HeartDisease' target variable and then generates a strip plot to visualize the distribution of features in a random subsample of 1000 rows from the dataset. The strip plot displays the individual data points for each feature, aiding in the identification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



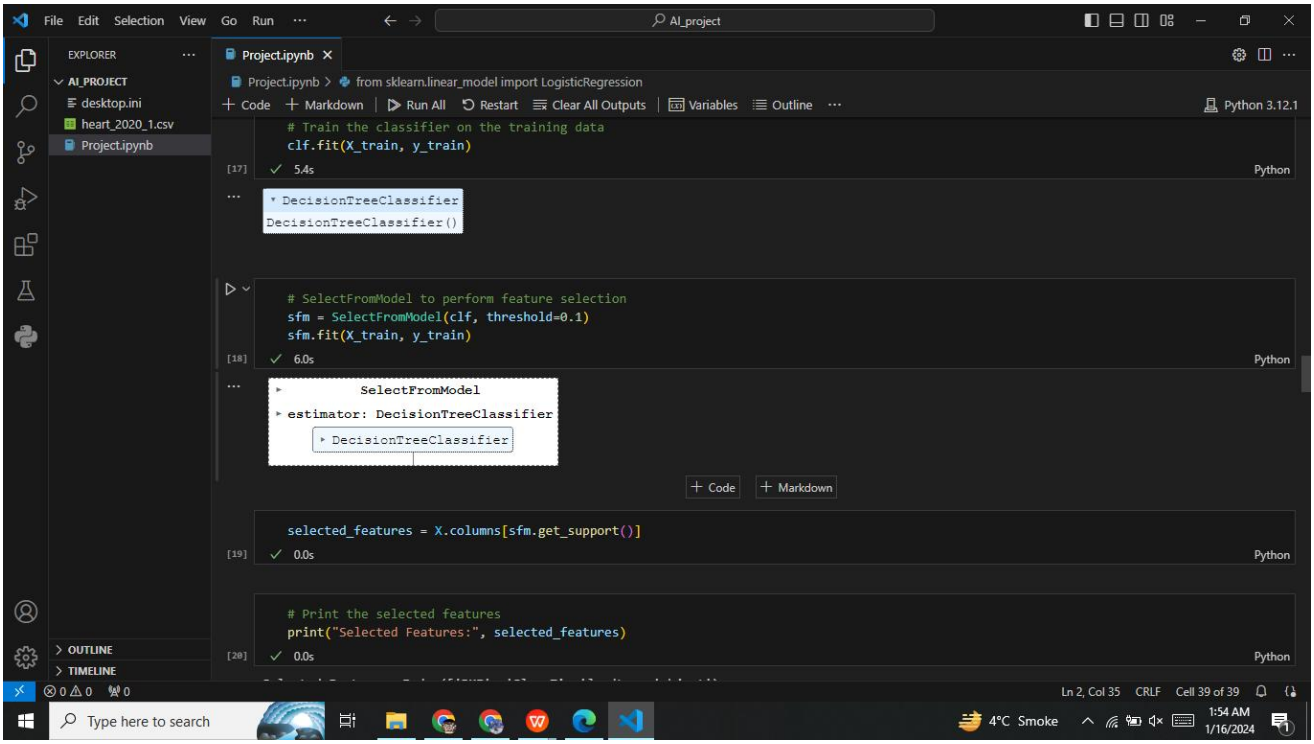
This shows distribution of features plotted in graph.

2. Feature Selection

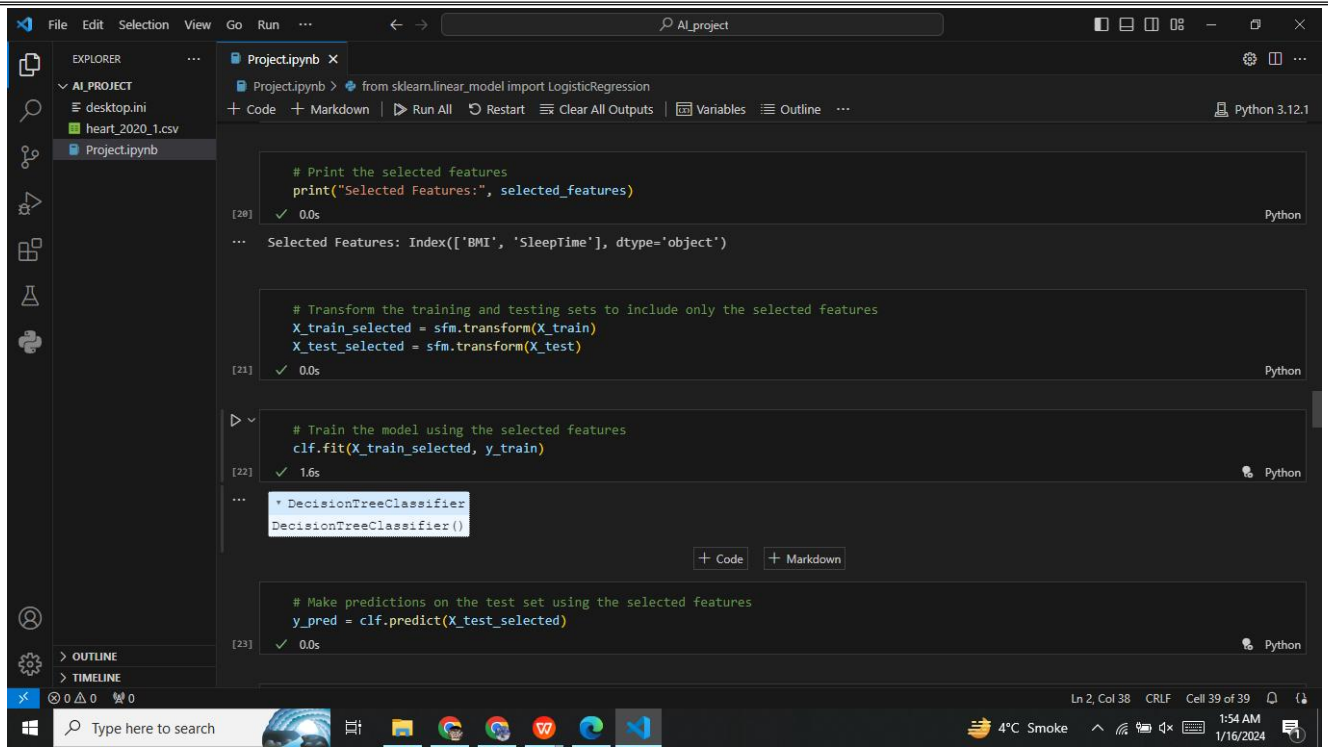
Decision Tree



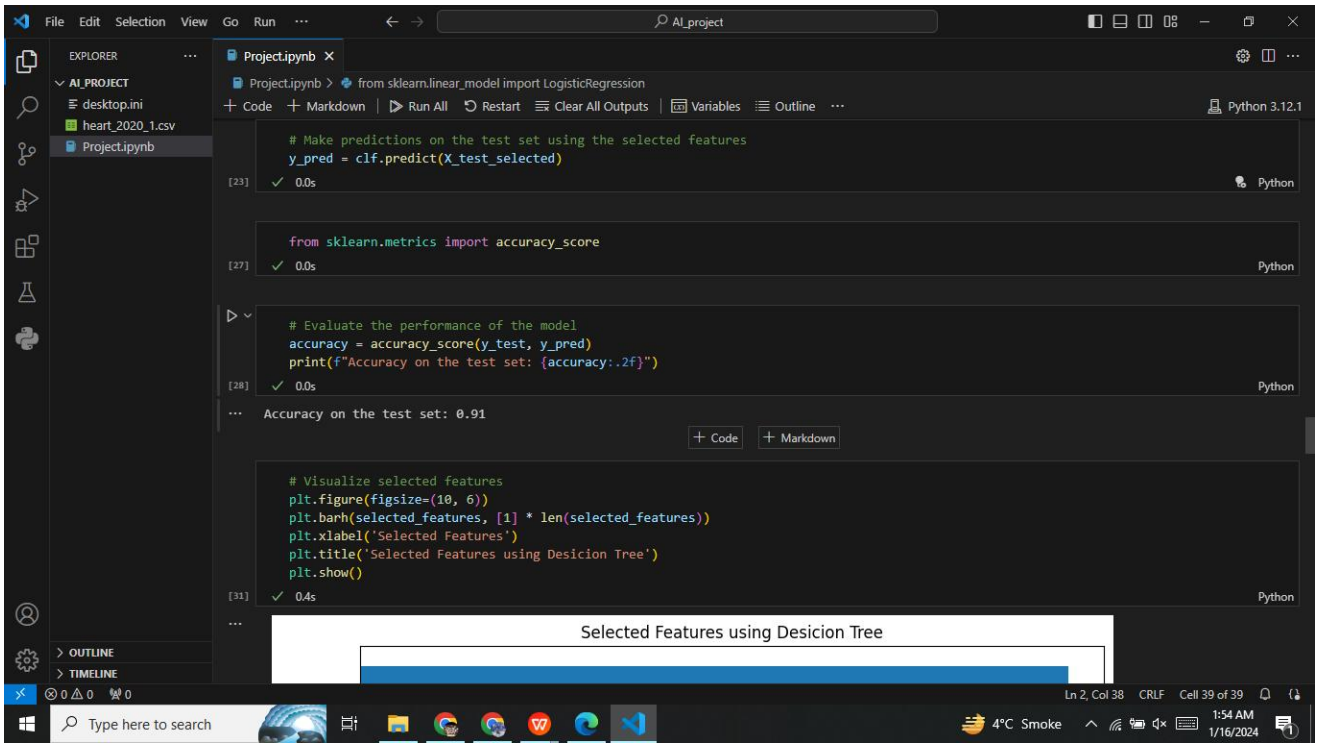
This Python code is implementing a **Decision Tree Classification Model** to predict the 'HeartDisease' variable based on the independent features in the DataFrame `df`. It splits the data into ***training*** and ***testing*** sets, then trains a **Decision Tree Classifier** using the training data.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



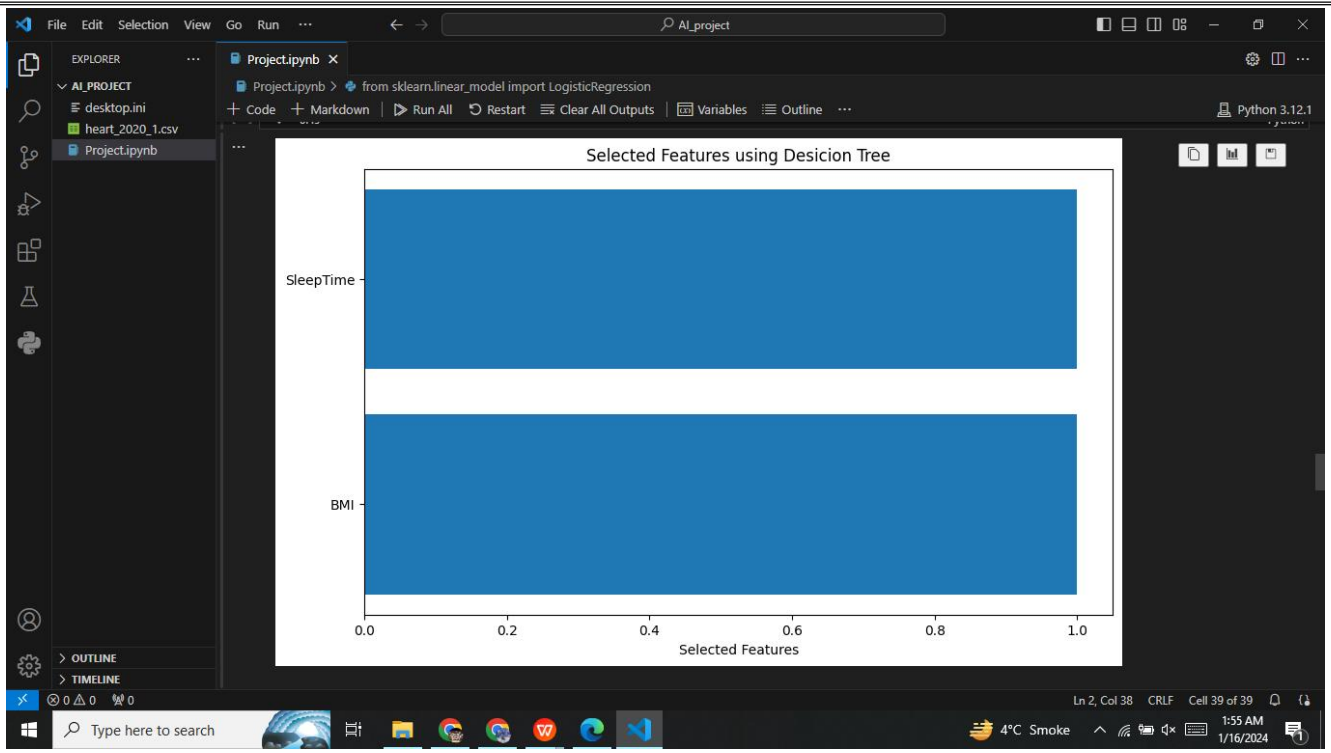
This code is using a pre-trained Decision Tree Classifier (`clf`) to identify important features for predicting 'HeartDisease' using the `**SelectFromModel**` method. It sets a threshold of 0.1, and the features surpassing this threshold are stored in `selected_features` for potential use in model training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



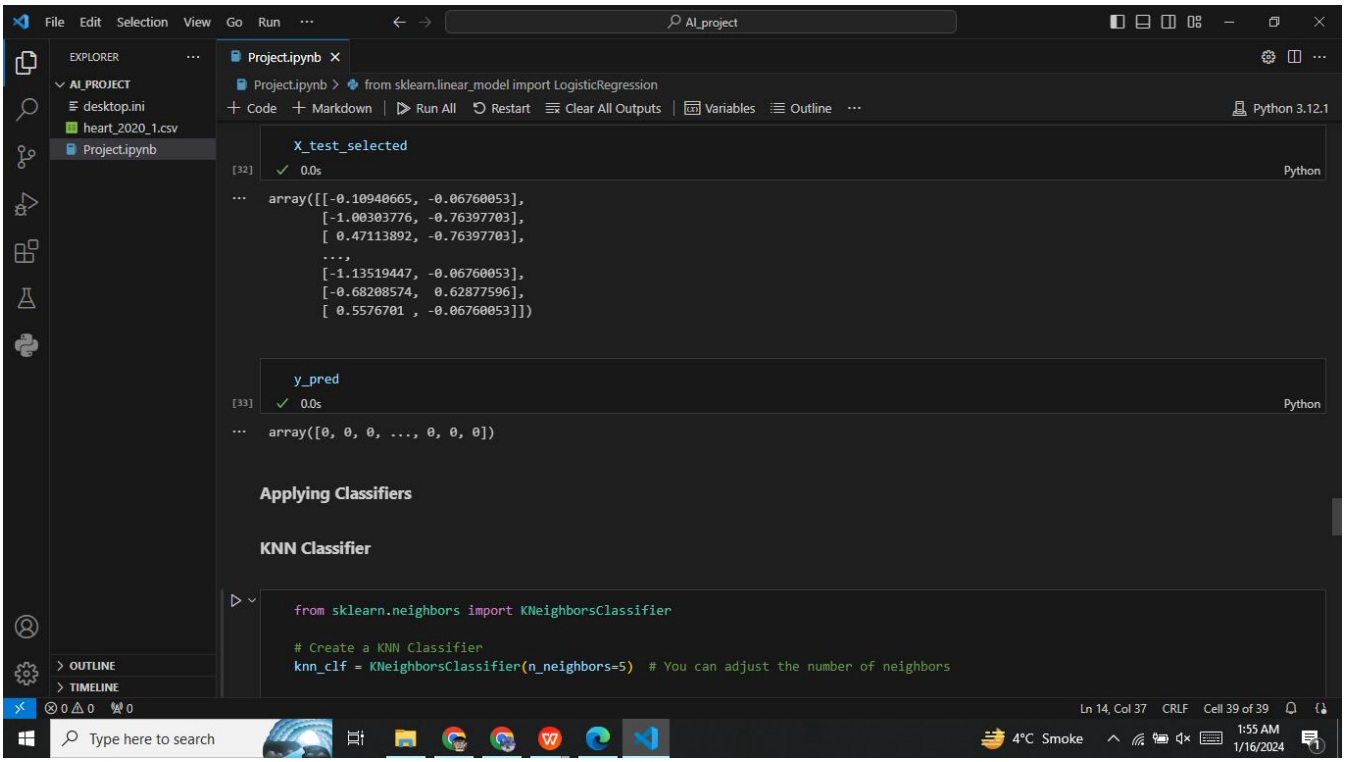
This code identifies and prints the features deemed important for predicting 'HeartDisease' using a pre-trained Decision Tree Classifier. It then retrains the classifier with only the selected features, streamlining the model to focus on the most influential predictors.



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.



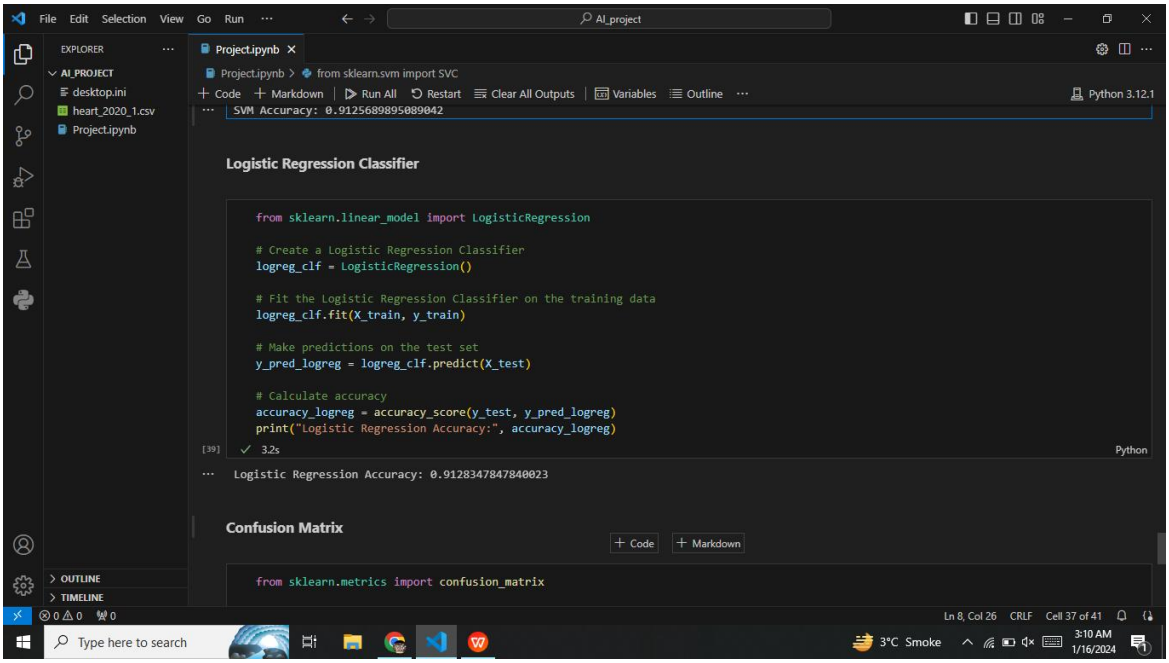
This graph is displaying selected features.



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

3. Classification

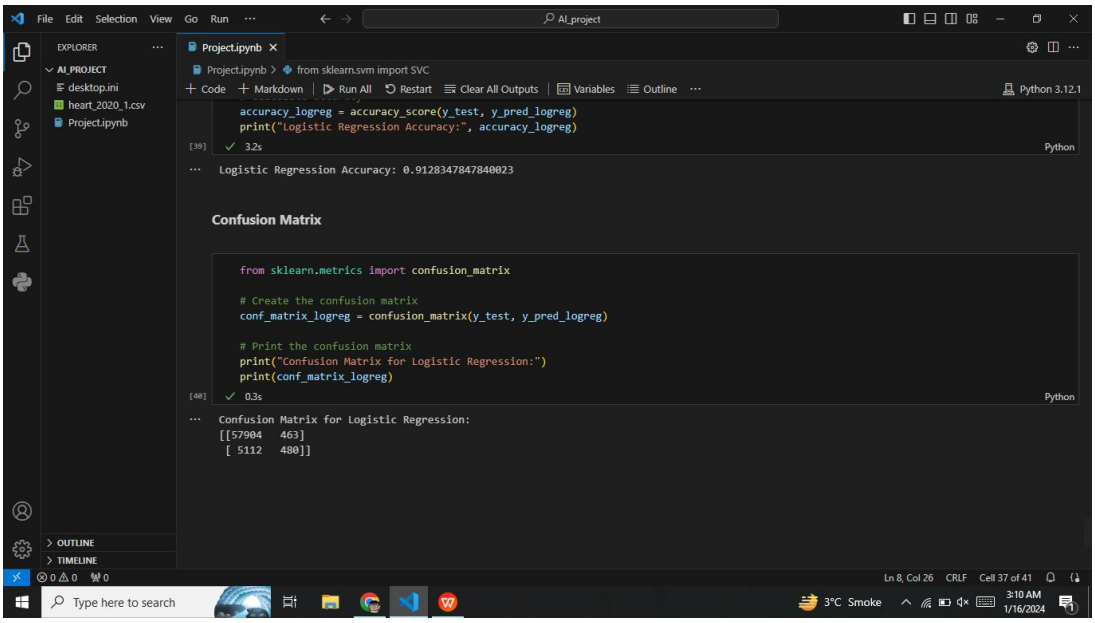
Logistic Regression Classifier



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.

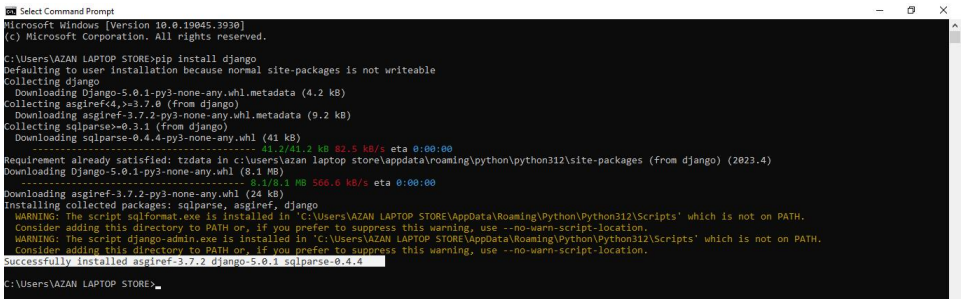
4. Performance metrics

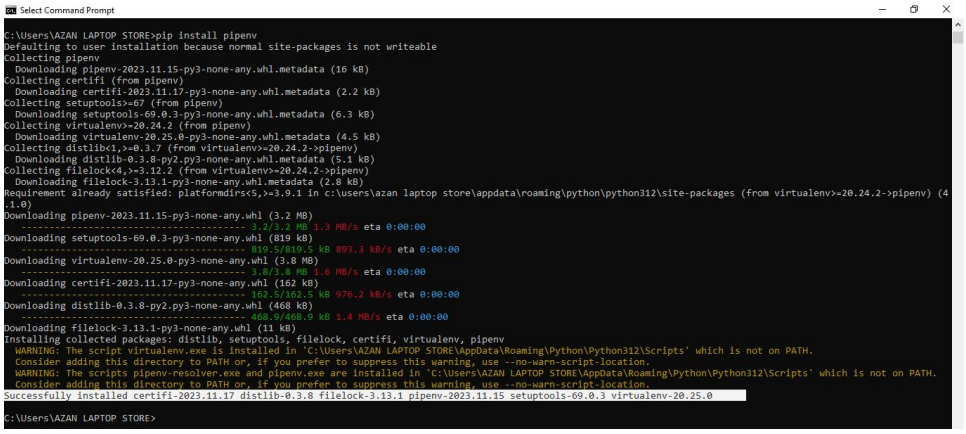
Confusion Matrix

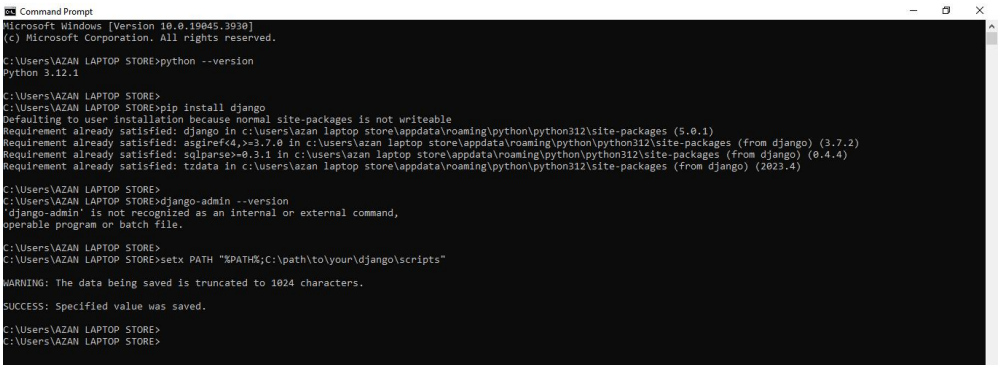


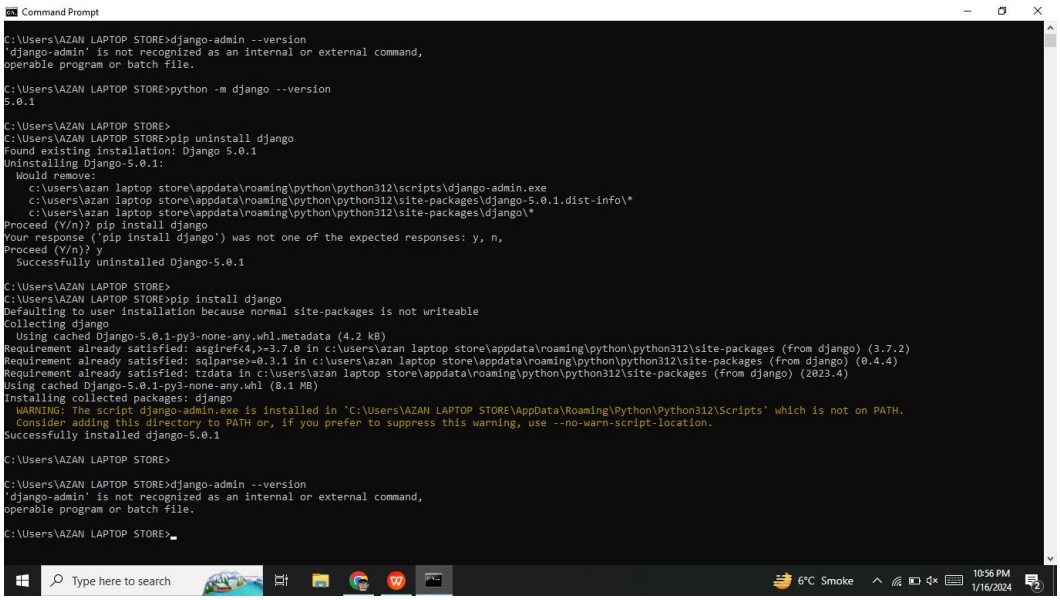
This code calculates and prints the confusion matrix for evaluating the performance of a logistic regression model on the test set. The **confusion matrix** provides a detailed breakdown of the model's predictions, showing the number of true positive, true negative, false positive, and false negative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

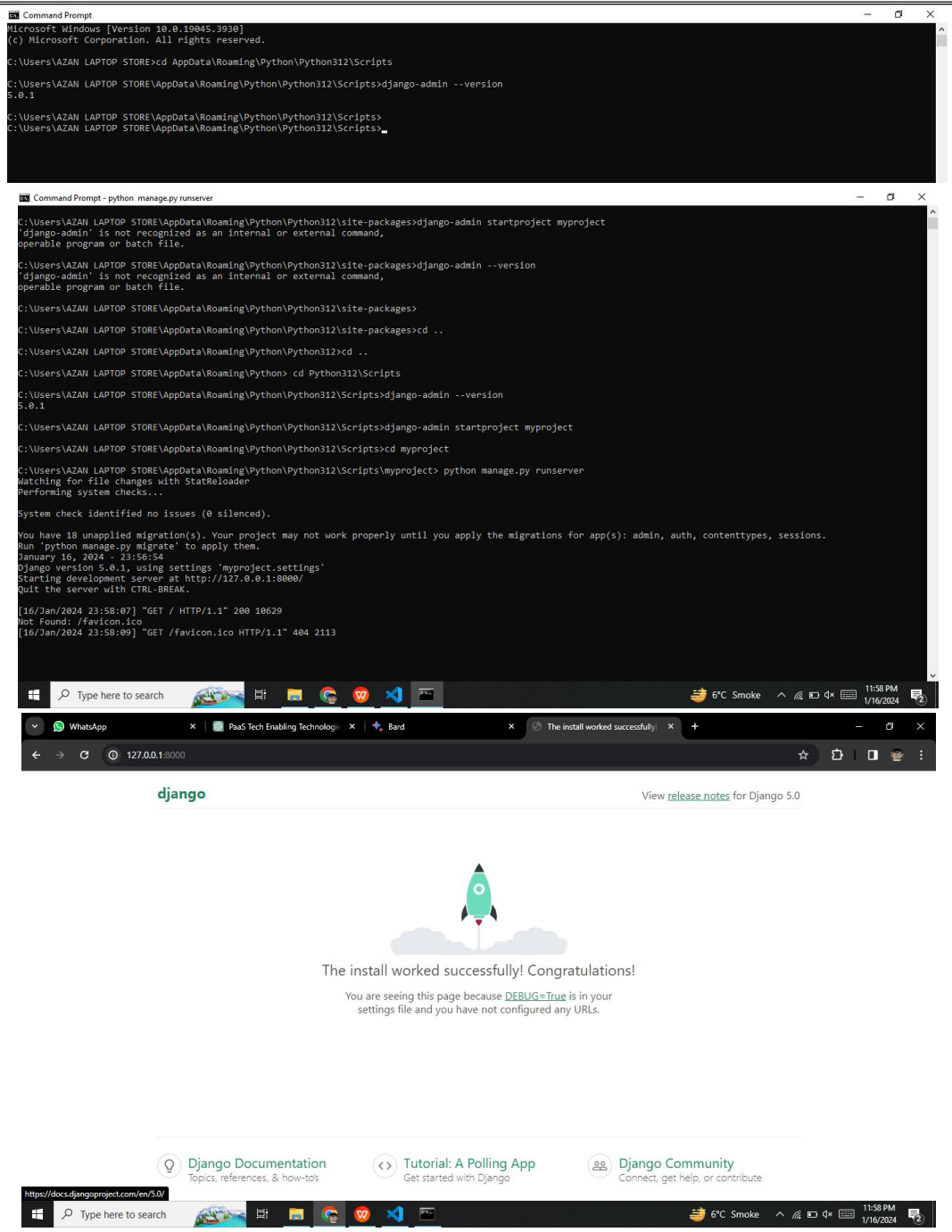
5. Front using django



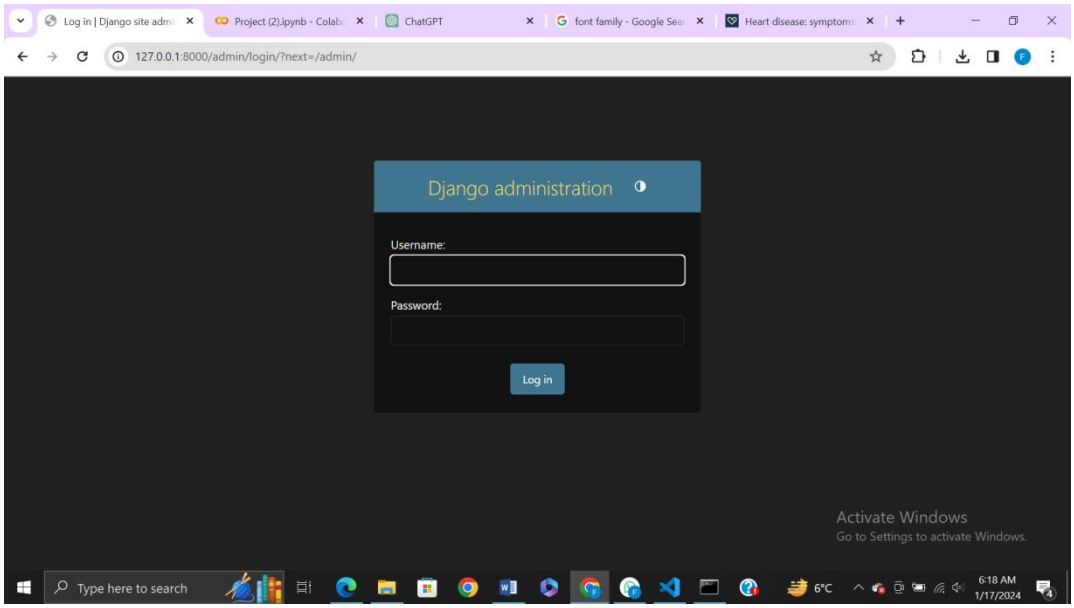








Making Interface for Disease Prediction



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-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
```

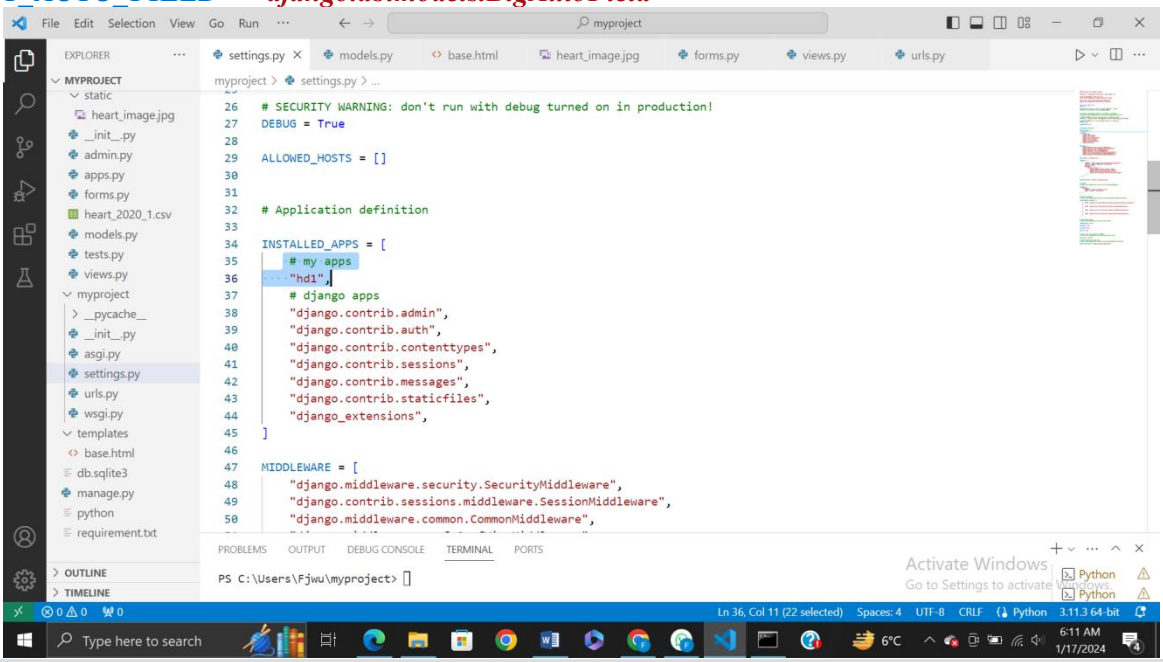
USE_TZ = True

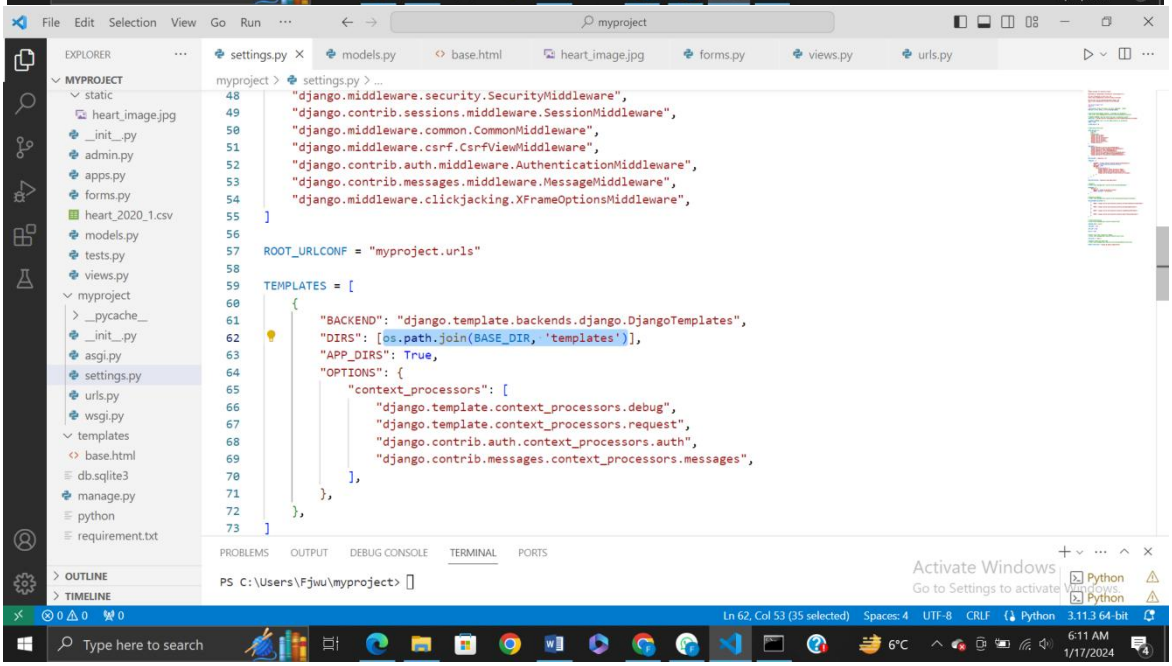
 # Static files (CSS, JavaScript, Images)
 # https://docs.djangoproject.com/en/5.0/howto/static-files/

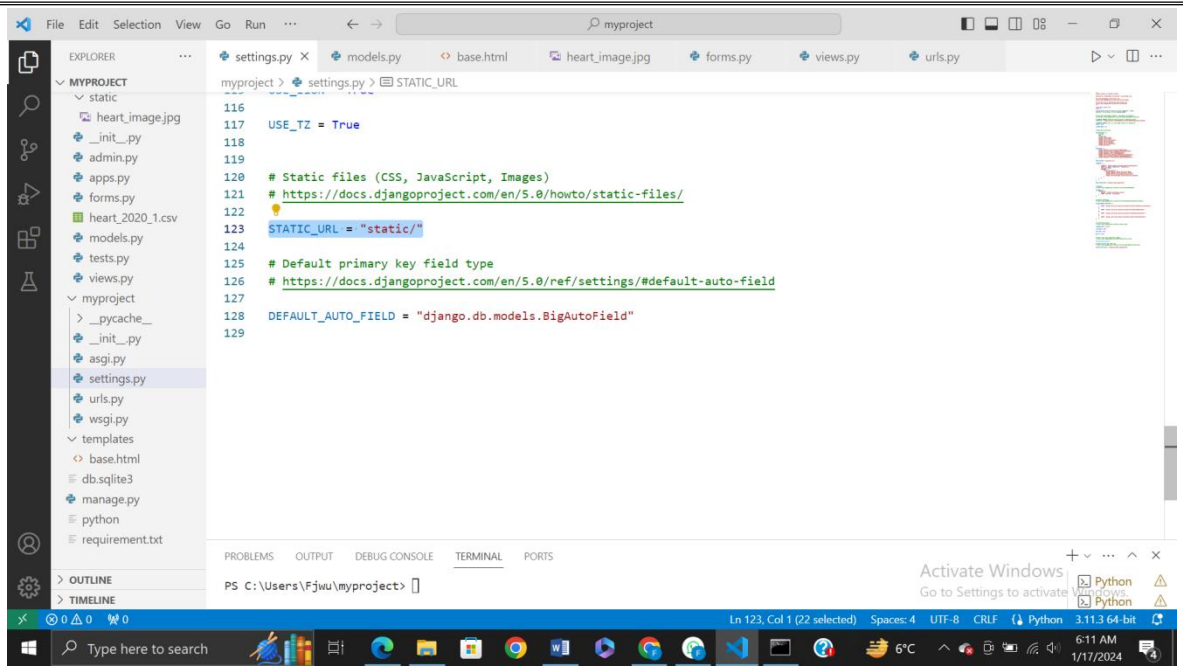
STATIC_URL = "static/"

 # Default primary key field type
 # https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"







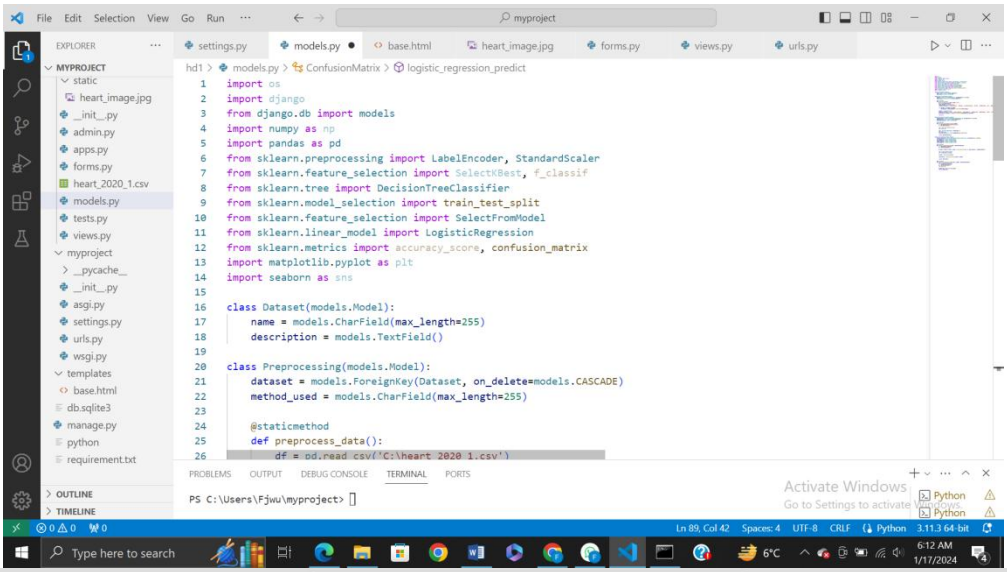
This code represents the Django project settings in a file named **"settings.py"** for a project called "myproject." It includes configurations for database settings, middleware, templates, static files, and other Django-specific parameters. Notably, it specifies a SQLite database, template directories,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

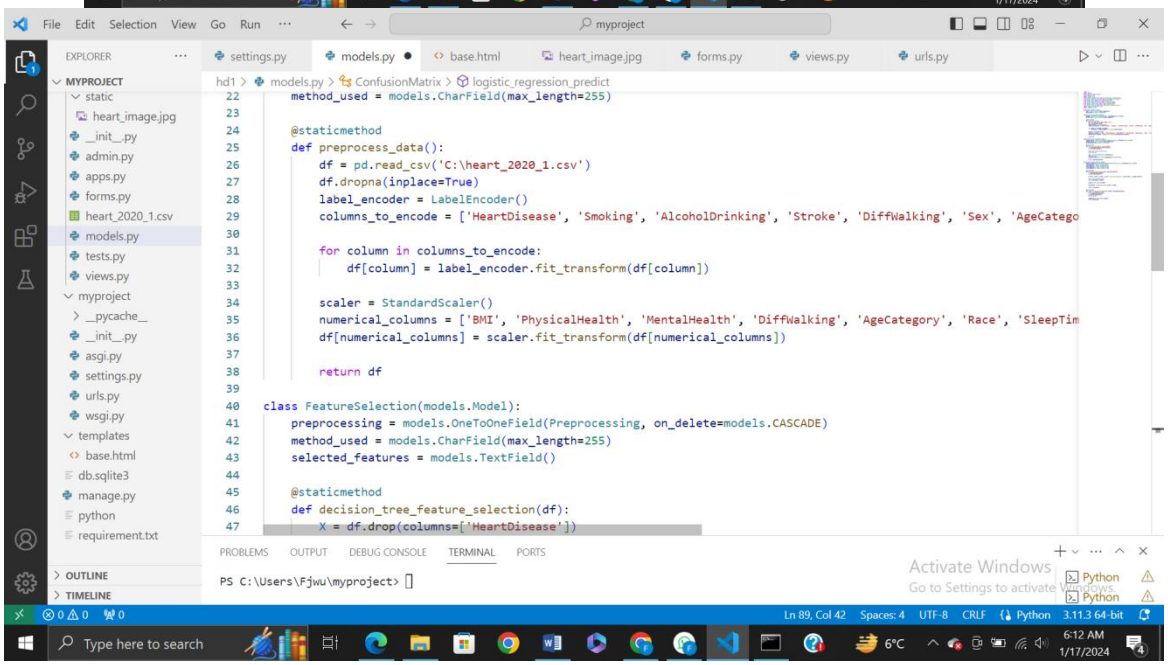
Models.py

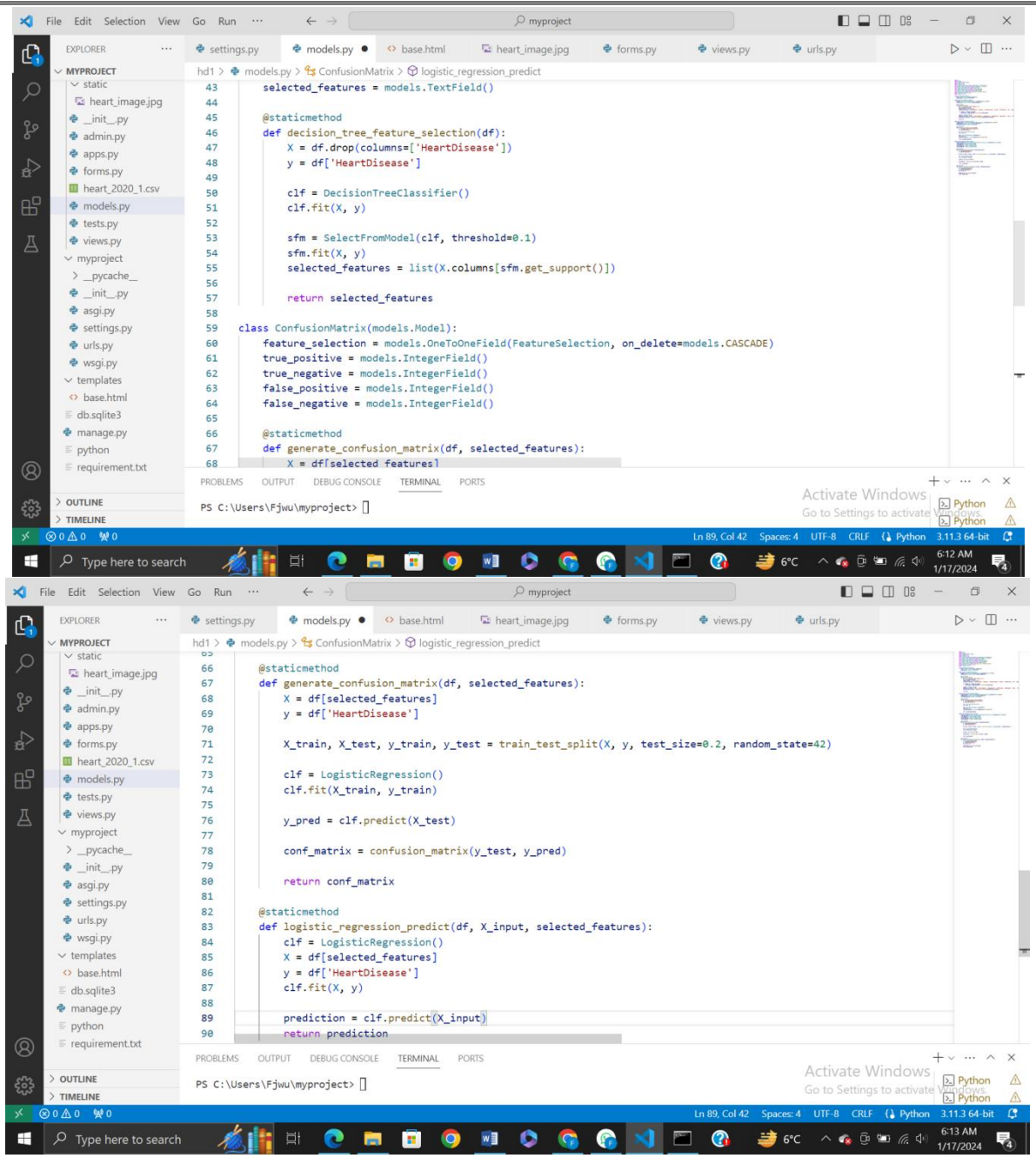
```
import os
import django
from django.db import models
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import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







```
    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]
```



```
    @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
```

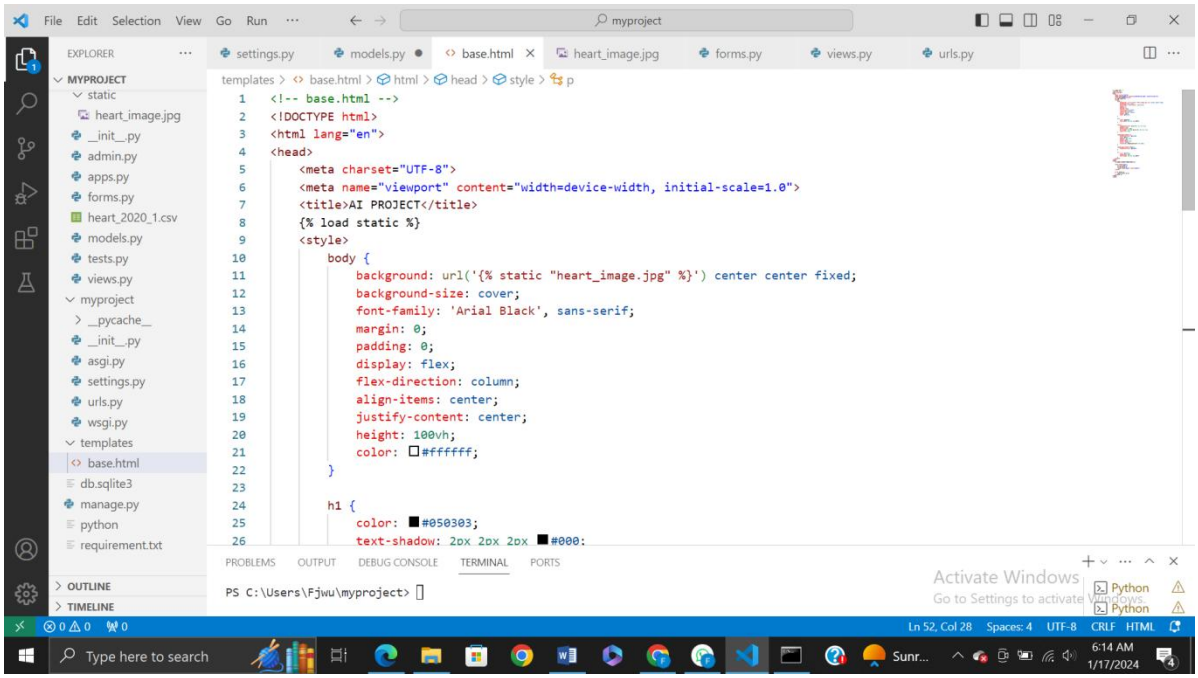
This Django models.py file defines three models: `Dataset`, `Preprocessing`, and `FeatureSelection`, along with methods for data preprocessing, decision tree-based feature selection, and logistic regression-based confusion matrix generation. It incorporates functionality to preprocess data, select relevant 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.

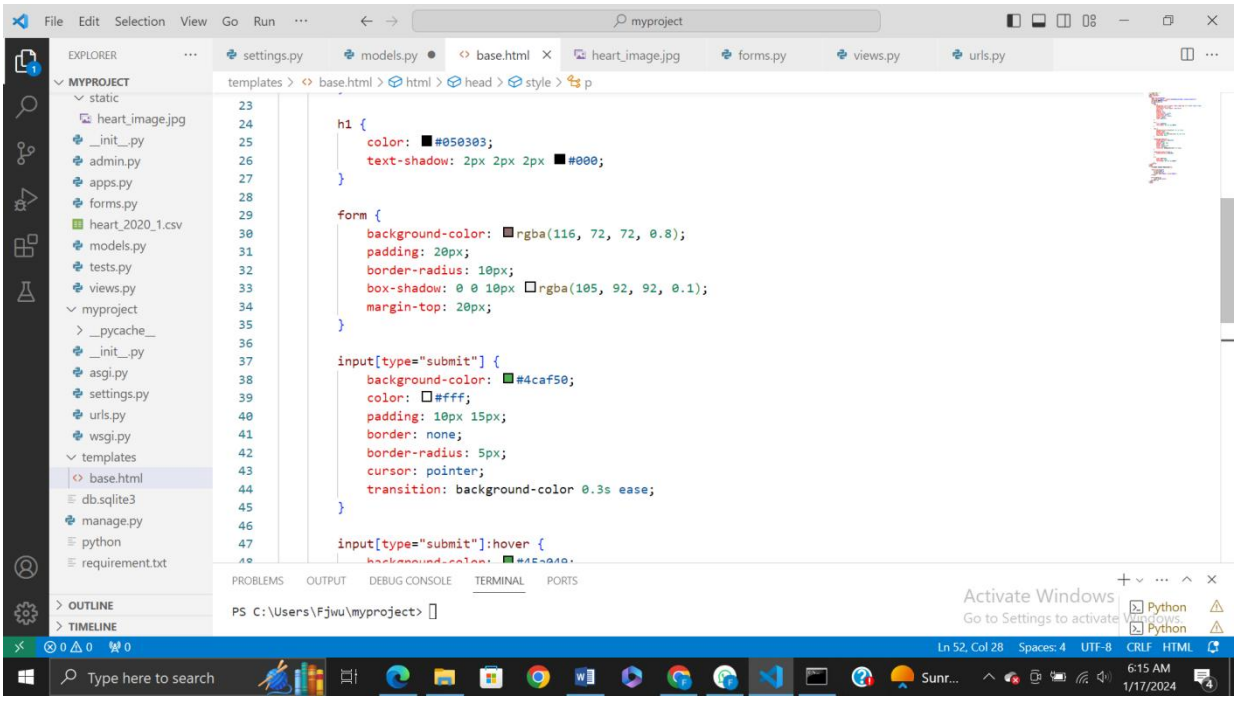
Base.html

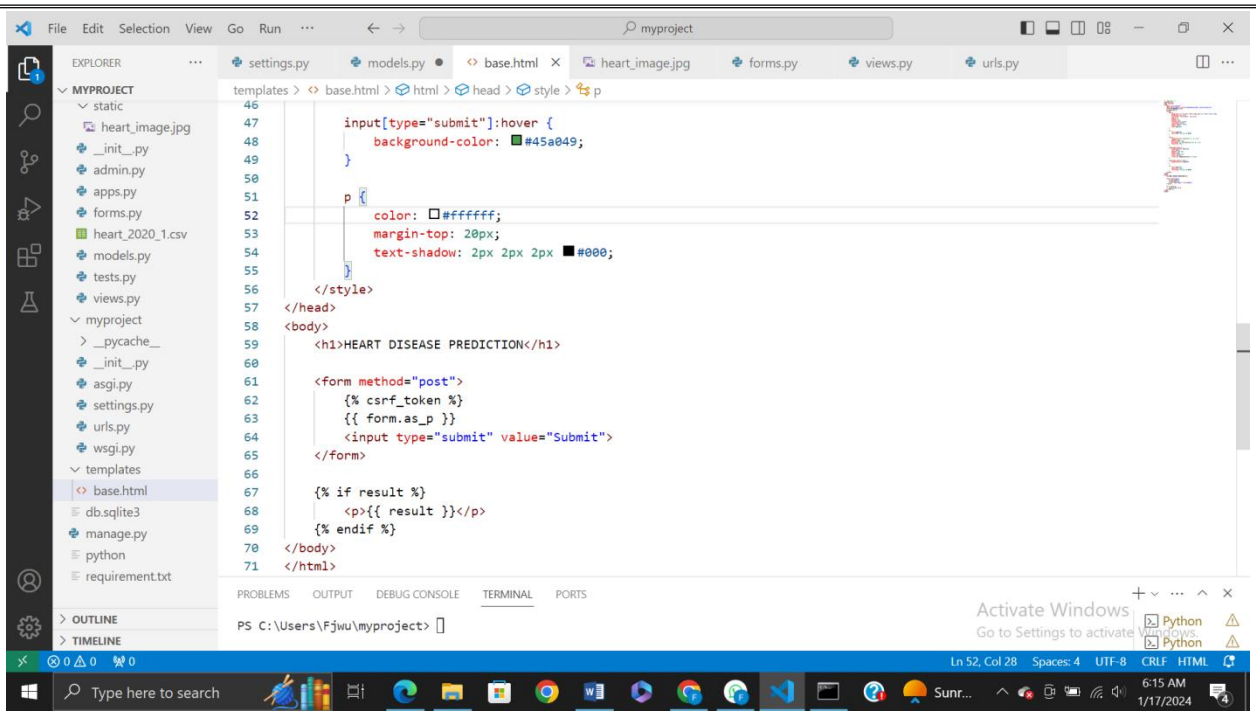
```
<!-- base.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <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>
```





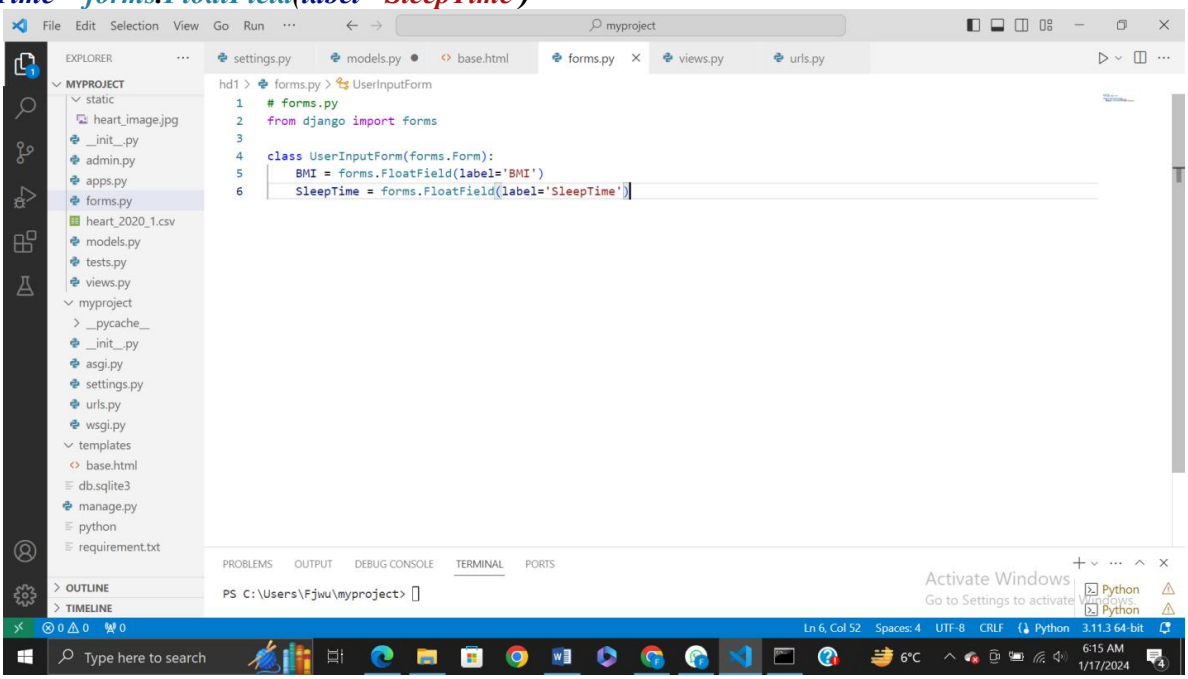


```
        input[type="submit"]:hover {
            background-color: #45a049;
        }

        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>
```

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.

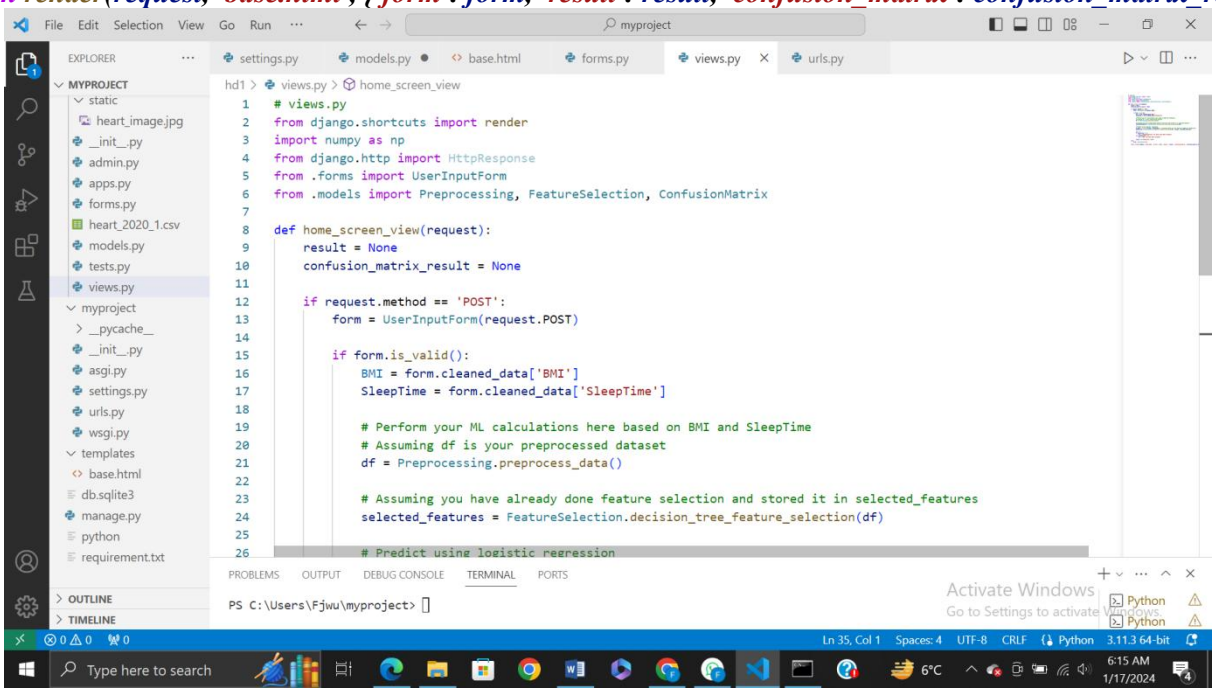
Forms.py:

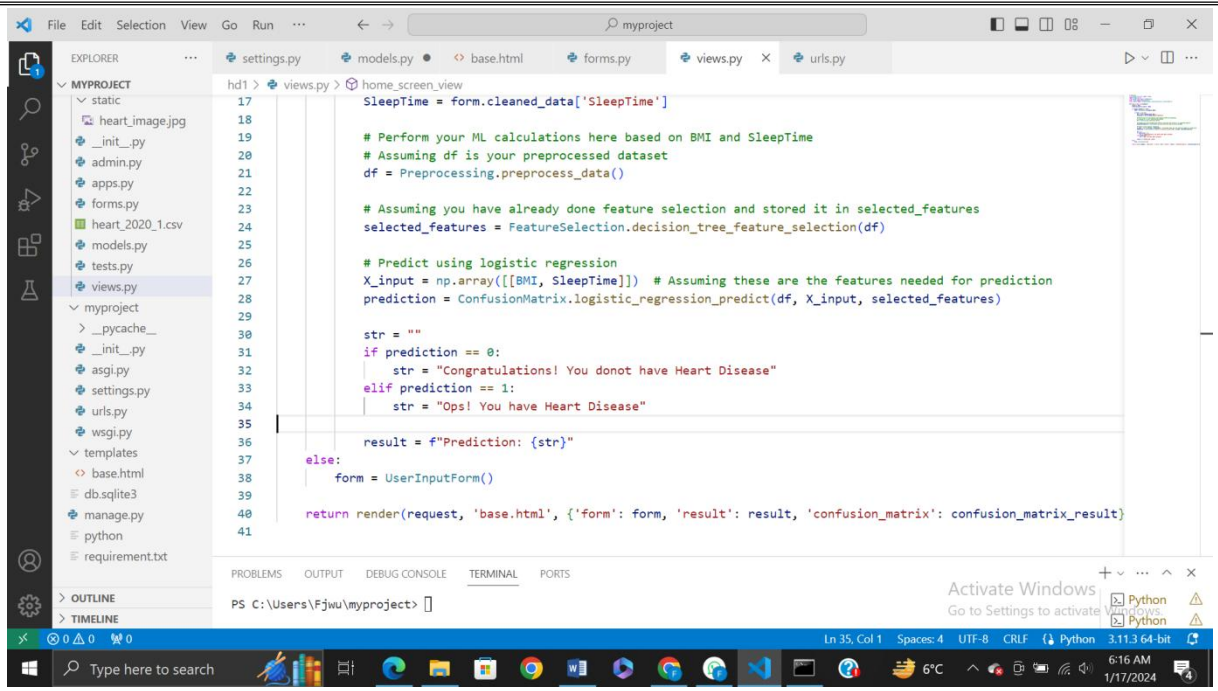
```
# forms.py
from django import forms

class UserInputForm(forms.Form):
    BMI = forms.FloatField(label='BMI')
    SleepTime = forms.FloatField(label='SleepTime')
```



Views.py:

```
# views.py
from django.shortcuts import render
import numpy as np
from django.http import HttpResponse
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 donot 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})
```



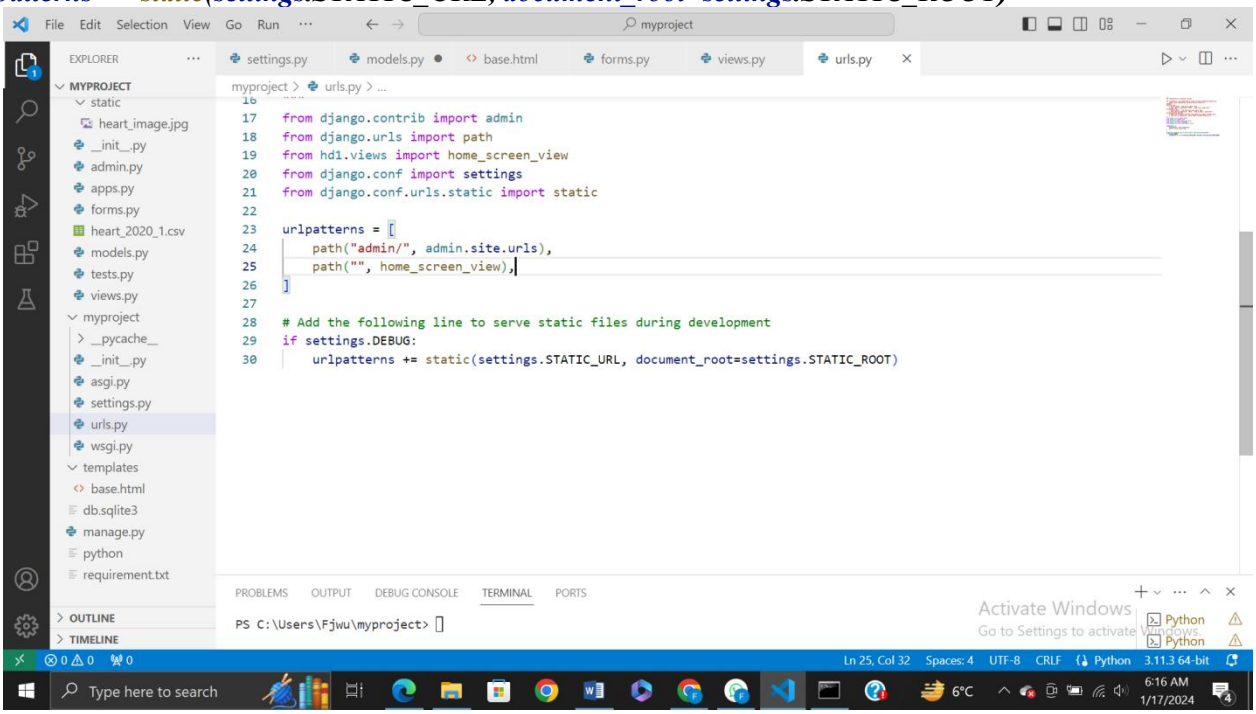


This Django view function processes user inputs for BMI and SleepTime, then uses a preprocessed dataset and decision tree feature selection to predict heart disease with logistic regression. The result, indicating the prediction outcome (either having or not having heart disease), is displayed on the 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

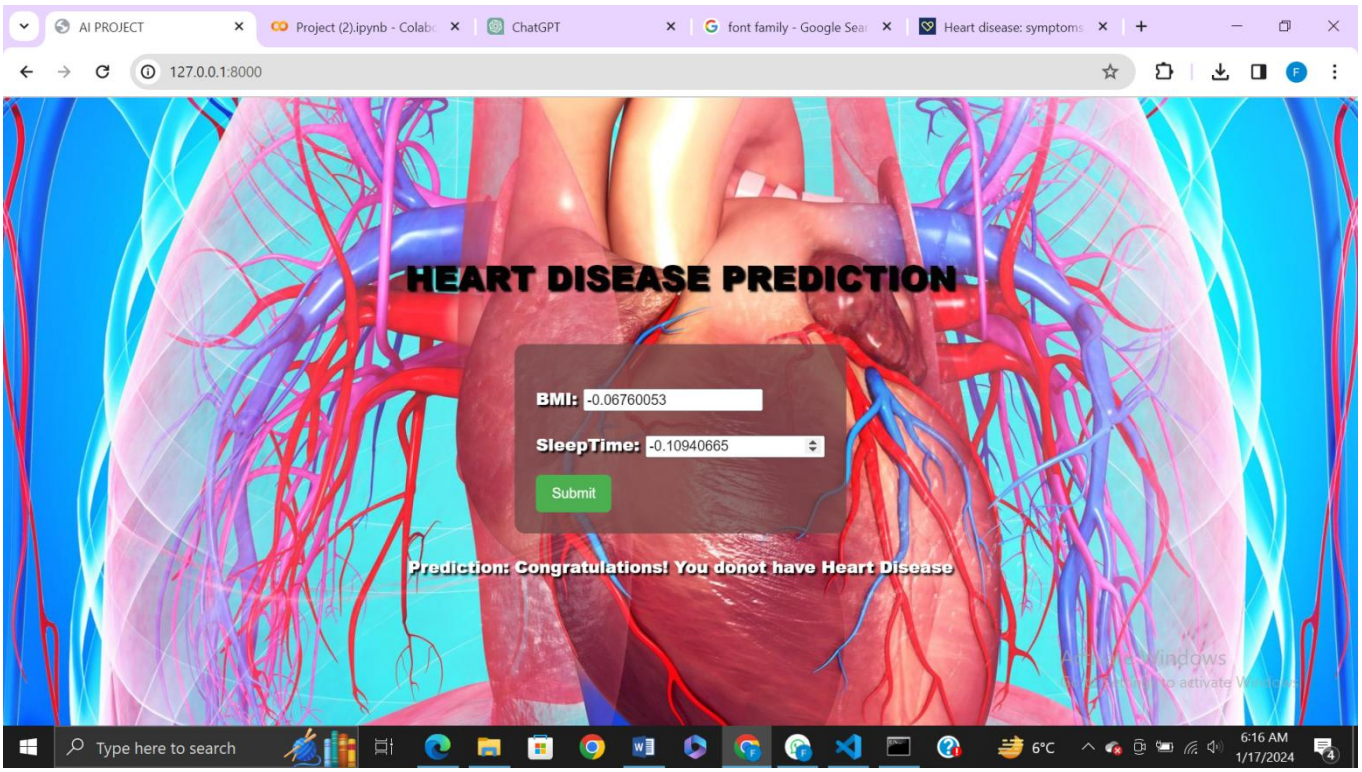
```
"""
URL configuration for myproject project.

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),
```

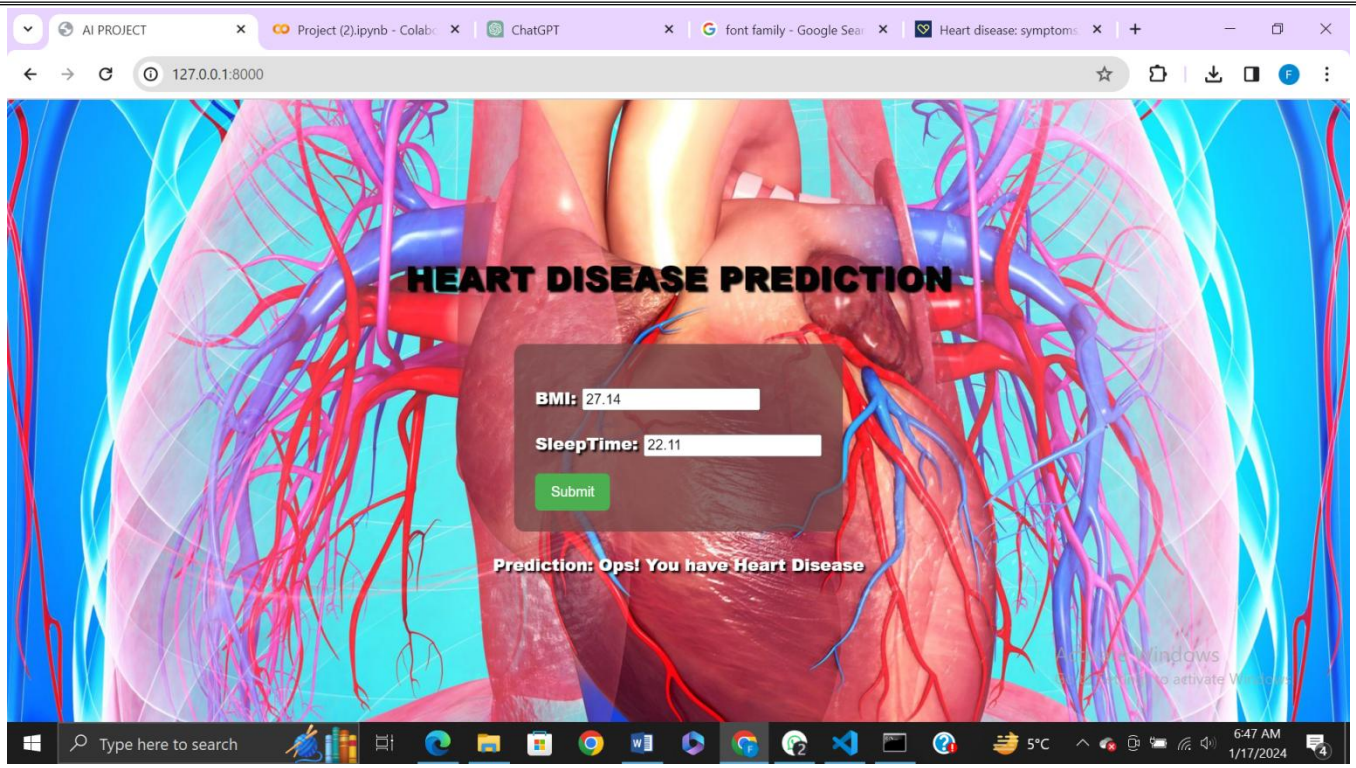
```
    path("", home_screen_view),
]

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



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:

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

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

$$\exists(x)\ SleepTime(x) \geq ThresholdSleepTime \land BMI(x) \leq ThresholdBMI \land HeartDisease(x)=0$$

For example:

If ThresholdSleepTime is 7 hours then:

$$\exists(x)\ SleepTime(x) \geq 7hrs \land BMI(x) \leq ThresholdBMI \land HeartDisease(x)=0$$

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: $\neg\exists xBMI(x) \geq ThresholdBMI$
- If no person has normal sleep or more: $\neg\exists xBMI(x) \geq ThresholdBMI$

$$\neg\exists(x)BMI(x) > ThresholdBMI \land SleepTime(x) \geq 7$$

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.
