**Section 1: Importing Libraries**

python

Copy

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

import streamlit as st

**Summary:**

* This section imports all the necessary libraries for the project:
  + numpy and pandas for data manipulation.
  + train\_test\_split for splitting the dataset into training and testing sets.
  + StandardScaler for standardizing the data.
  + SVC (Support Vector Classifier) for building the machine learning model.
  + accuracy\_score for evaluating the model's performance.
  + streamlit for creating the web app.

**Reasoning:**

* These libraries are essential for data preprocessing, model training, and creating an interactive web application.

**Section 2: Loading the Dataset**

python

Copy

data = pd.read\_csv('diabetes.csv') # Make sure the file path is correct

**Summary:**

* This line loads the diabetes dataset from a CSV file into a Pandas DataFrame.

**Reasoning:**

* The dataset is the foundation of the project. It contains features like Glucose, BloodPressure, etc., and a target column Outcome indicating whether a person has diabetes (1) or not (0).

**Section 3: Removing the 'Pregnancies' Column**

python

Copy

data = data.drop('Pregnancies', axis=1)

**Summary:**

* The Pregnancies column is removed from the dataset.

**Reasoning:**

* This step is specific to your project requirements. If the Pregnancies column is not relevant to the prediction, it can be dropped to simplify the model.

**Section 4: Preparing Features and Target**

python

Copy

X = data.drop('Outcome', axis=1) # Features (all columns except 'Outcome')

y = data['Outcome'] # Target (Outcome column)

**Summary:**

* X contains all the features (independent variables) by dropping the Outcome column.
* y contains the target variable (Outcome).

**Reasoning:**

* This separation is necessary for supervised learning, where the model learns from the features (X) to predict the target (y).

**Section 5: Splitting the Data**

python

Copy

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

**Summary:**

* The dataset is split into:
  + Training data (X\_train, y\_train) - 80% of the data.
  + Testing data (X\_test, y\_test) - 20% of the data.

**Reasoning:**

* Splitting the data ensures that the model is trained on one portion and evaluated on another, preventing overfitting.

**Section 6: Data Preprocessing (Scaling)**

python

Copy

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

**Summary:**

* The StandardScaler standardizes the features by removing the mean and scaling to unit variance.
* fit\_transform is used on the training data to learn the scaling parameters and apply them.
* transform is used on the testing data to apply the same scaling.

**Reasoning:**

* Scaling ensures that all features contribute equally to the model, improving the performance of the SVM algorithm.

**Section 7: Model Training**

python

Copy

model = SVC(kernel='rbf', C=1.0, gamma='scale')

model.fit(X\_train, y\_train)

**Summary:**

* A Support Vector Machine (SVM) model with an RBF kernel is initialized.
* The model is trained using the training data (X\_train, y\_train).

**Reasoning:**

* The RBF kernel is chosen for better accuracy in handling non-linear relationships in the data.

**Section 8: Streamlit App Setup**

python

Copy

st.title('Diabetes Prediction App')

**Summary:**

* This line sets the title of the Streamlit web app.

**Reasoning:**

* The title provides a clear description of the app's purpose.

**Section 9: Age-wise Average Values**

python

Copy

age\_wise\_avg = data.groupby('Age').mean().reset\_index()

**Summary:**

* This calculates the average values of all features for each age group in the dataset.

**Reasoning:**

* These averages are used later to display "normal values" for a given age, providing context for the user.

**Section 10: Sidebar for User Input**

python

Copy

st.sidebar.header('User Input Features')

def user\_input\_features():

glucose = st.sidebar.text\_input('Glucose', '0')

blood\_pressure = st.sidebar.text\_input('Blood Pressure', '0')

skin\_thickness = st.sidebar.text\_input('Skin Thickness', '0')

insulin = st.sidebar.text\_input('Insulin', '0')

bmi = st.sidebar.text\_input('BMI', '0.0')

diabetes\_pedigree\_function = st.sidebar.text\_input('Diabetes Pedigree Function', '0.0')

age = st.sidebar.text\_input('Age', '0')

data = {

'Glucose': int(glucose),

'BloodPressure': int(blood\_pressure),

'SkinThickness': int(skin\_thickness),

'Insulin': int(insulin),

'BMI': float(bmi),

'DiabetesPedigreeFunction': float(diabetes\_pedigree\_function),

'Age': int(age)

}

features = pd.DataFrame(data, index=[0])

features = features[X.columns]

return features

**Summary:**

* A sidebar is created for manual user input.
* The user\_input\_features function collects input values for each feature and returns them as a DataFrame.

**Reasoning:**

* This allows users to input their own data for prediction without uploading a file.

**Section 11: File Upload Section**

python

Copy

st.header('File Upload Section')

uploaded\_file = st.file\_uploader("Upload your input file (CSV or Excel)", type=["csv", "xlsx"])

**Summary:**

* A file uploader is added to allow users to upload CSV or Excel files.
* The uploaded file is read and displayed in a clean table.

**Reasoning:**

* This feature enables bulk predictions for multiple patients at once.

**Section 12: Prediction on Uploaded Data**

python

Copy

if uploaded\_file is not None:

if uploaded\_file.name.endswith('.csv'):

input\_df = pd.read\_csv(uploaded\_file)

else:

input\_df = pd.read\_excel(uploaded\_file)

st.subheader('Uploaded File Data:')

st.dataframe(input\_df)

if all(column in input\_df.columns for column in required\_columns):

input\_df = input\_df[required\_columns]

for col in input\_df.columns:

if col == 'BMI':

input\_df[col] = input\_df[col].astype(float)

elif col != 'DiabetesPedigreeFunction':

input\_df[col] = input\_df[col].astype(int)

input\_scaled = scaler.transform(input\_df)

predictions = model.predict(input\_scaled)

input\_df['Prediction'] = predictions

st.subheader('Normal Values for Each Patient:')

for i, row in input\_df.iterrows():

patient\_age = row['Age']

normal\_values = age\_wise\_avg[age\_wise\_avg['Age'] == patient\_age].drop('Outcome', axis=1)

normal\_values = normal\_values[['Age'] + [col for col in normal\_values.columns if col != 'Age']]

for col in normal\_values.columns:

if col == 'BMI':

normal\_values[col] = normal\_values[col].astype(float)

elif col != 'DiabetesPedigreeFunction':

normal\_values[col] = normal\_values[col].astype(int)

st.write(f"Patient {i+1} (Age: {patient\_age}):")

st.dataframe(normal\_values.style.hide(axis='index'))

if row['Prediction'] == 1:

st.write("\*\*This person has Diabetes.\*\*")

else:

st.write("\*\*This person does not have Diabetes.\*\*")

st.write("---")

else:

st.error(f"Uploaded file must contain the following columns: {required\_columns}")

**Summary:**

* If a file is uploaded, it is read and displayed.
* The uploaded data is preprocessed, scaled, and used for predictions.
* Predictions and normal values for each patient are displayed.

**Reasoning:**

* This section handles bulk predictions and provides context by showing normal values for each patient's age.

**Section 13: Manual Input Section**

python

Copy

st.header('Manual Input Section')

input\_df = user\_input\_features()

input\_df = input\_df[['Age'] + [col for col in input\_df.columns if col != 'Age']]

st.subheader('Patient Values:')

st.dataframe(input\_df.style.hide(axis='index'))

user\_age = input\_df['Age'].values[0]

if user\_age > 0:

normal\_values = age\_wise\_avg[age\_wise\_avg['Age'] == user\_age].drop('Outcome', axis=1)

normal\_values = normal\_values[['Age'] + [col for col in normal\_values.columns if col != 'Age']]

for col in normal\_values.columns:

if col == 'BMI':

normal\_values[col] = normal\_values[col].astype(float)

elif col != 'DiabetesPedigreeFunction':

normal\_values[col] = normal\_values[col].astype(int)

st.subheader('Normal Values:')

st.dataframe(normal\_values.style.hide(axis='index'))

else:

st.write("\*\*Please enter a valid age to see normal values.\*\*")

try:

input\_df = input\_df[X.columns]

input\_scaled = scaler.transform(input\_df)

prediction = model.predict(input\_scaled)

st.subheader('Prediction:')

if user\_age > 0:

if prediction[0] == 1:

st.write('\*\*This person has Diabetes.\*\*')

else:

st.write('\*\*This person does not have Diabetes.\*\*')

else:

st.write("\*\*Please enter valid input to see the prediction.\*\*")

except Exception as e:

st.error(f"An error occurred during prediction: {e}")

**Summary:**

* This section handles manual input from the user.
* The input is preprocessed, scaled, and used for prediction.
* Normal values for the user's age are displayed alongside the prediction.

**Reasoning:**

* This provides a user-friendly interface for individual predictions.

**Section 14: Accuracy and SVM**

The code does not explicitly calculate accuracy, but it could be added using:

python

Copy

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

st.write(f"Model Accuracy: {accuracy:.2f}")

**Summary:**

* This would calculate and display the model's accuracy on the test set.

**Reasoning:**

* Accuracy is a key metric for evaluating the model's performance.

**Section 15: Web App Deployment**

The code does not include deployment steps, but here’s how you can deploy the app:

1. **Streamlit Sharing**: Upload the app to GitHub and deploy using Streamlit Sharing.
2. **Heroku**: Another option for deploying Streamlit apps.

**Reasoning:**

* Deployment makes the app accessible to users worldwide.

**Final Summary**

This code is a complete pipeline for:

1. Loading and preprocessing data.
2. Training an SVM model.
3. Creating an interactive Streamlit app for diabetes prediction.
4. Handling both manual input and file uploads.
5. Displaying predictions and normal values for context.

Each section is designed to ensure clarity, usability, and functionality. The app is ready for deployment, making it a powerful tool for diabetes prediction.