Elements OF AI/ML

Document File Containing the Working of Model

Name : Gaurav Srivastava

SAP : 500122467

ROLL : R2142230319

BATCH : 11

Working of the Flask App :

→ This is the home page of the flask app , i.e. , app.py which is linked with my Machine Learning Model , i.e. , Diabetic-Model.pkl.

→ Now the user enters real time data in the flask app for each features.

- First we enter that data that should predict Diabetes in the final predication.

Pregnancies : 2

Glucose : 120

Blood Pressure : 80

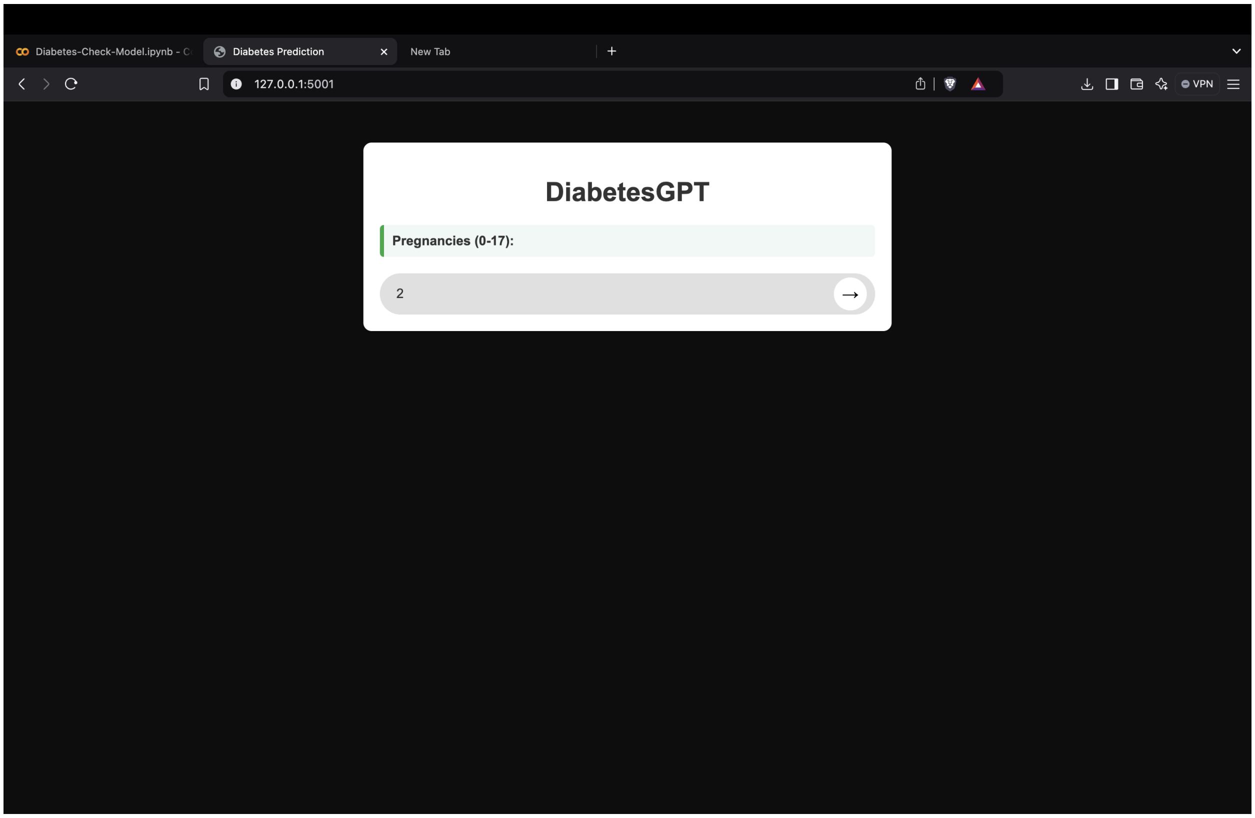
Skin Thickness : 28.5

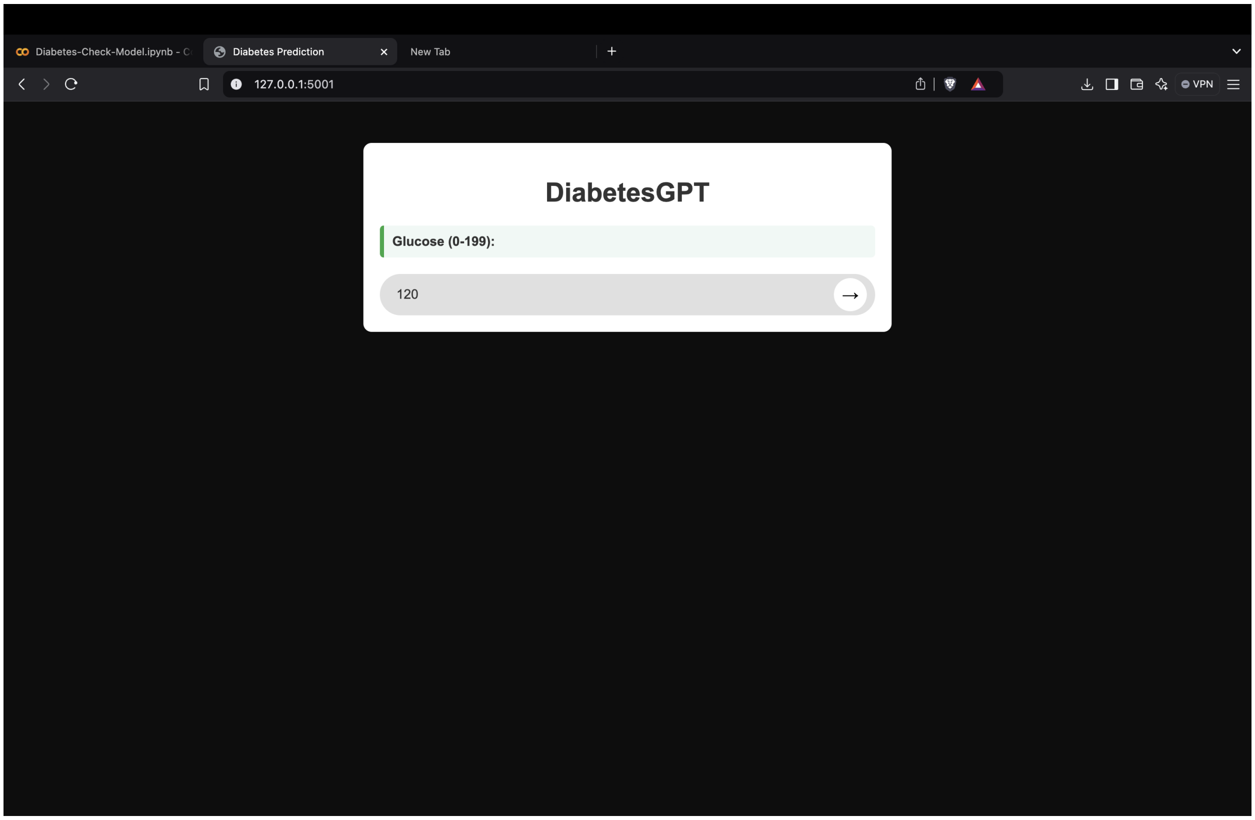
Insulin : 80

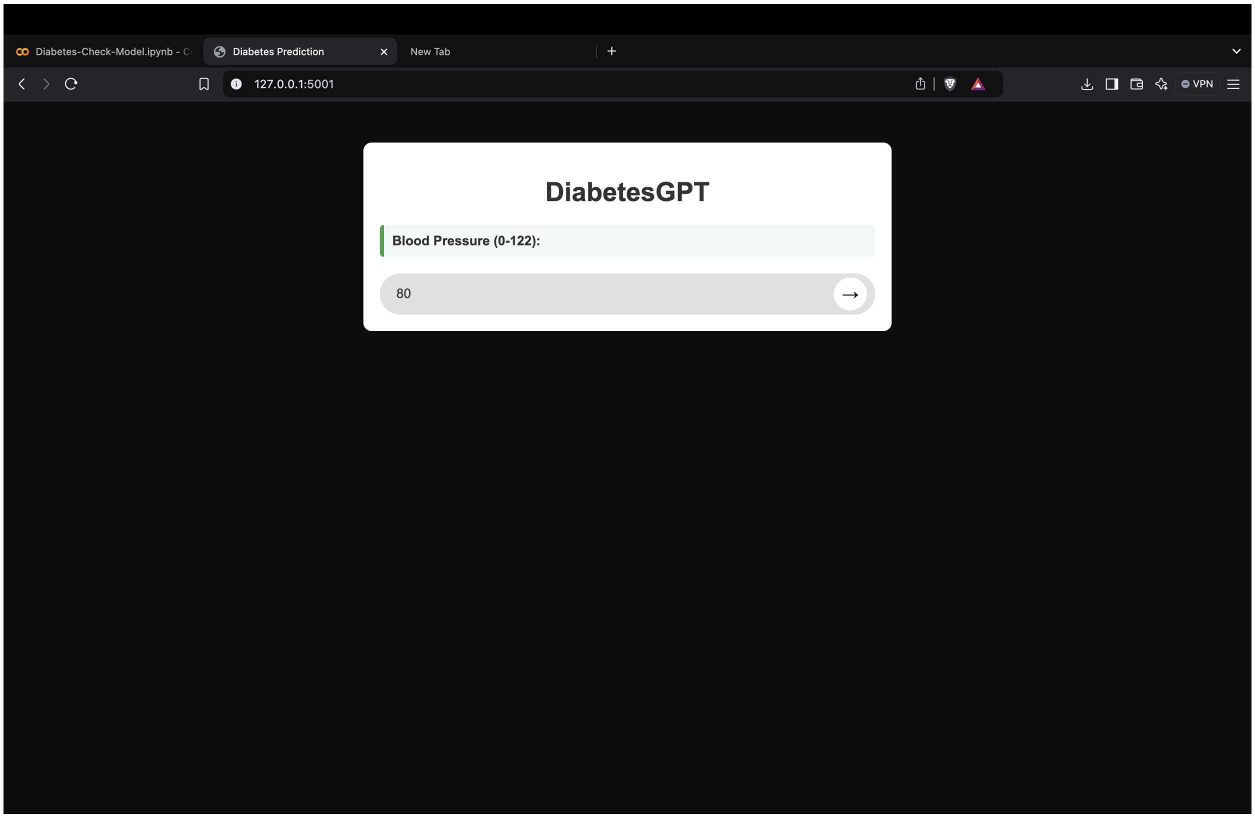
BMI : 28

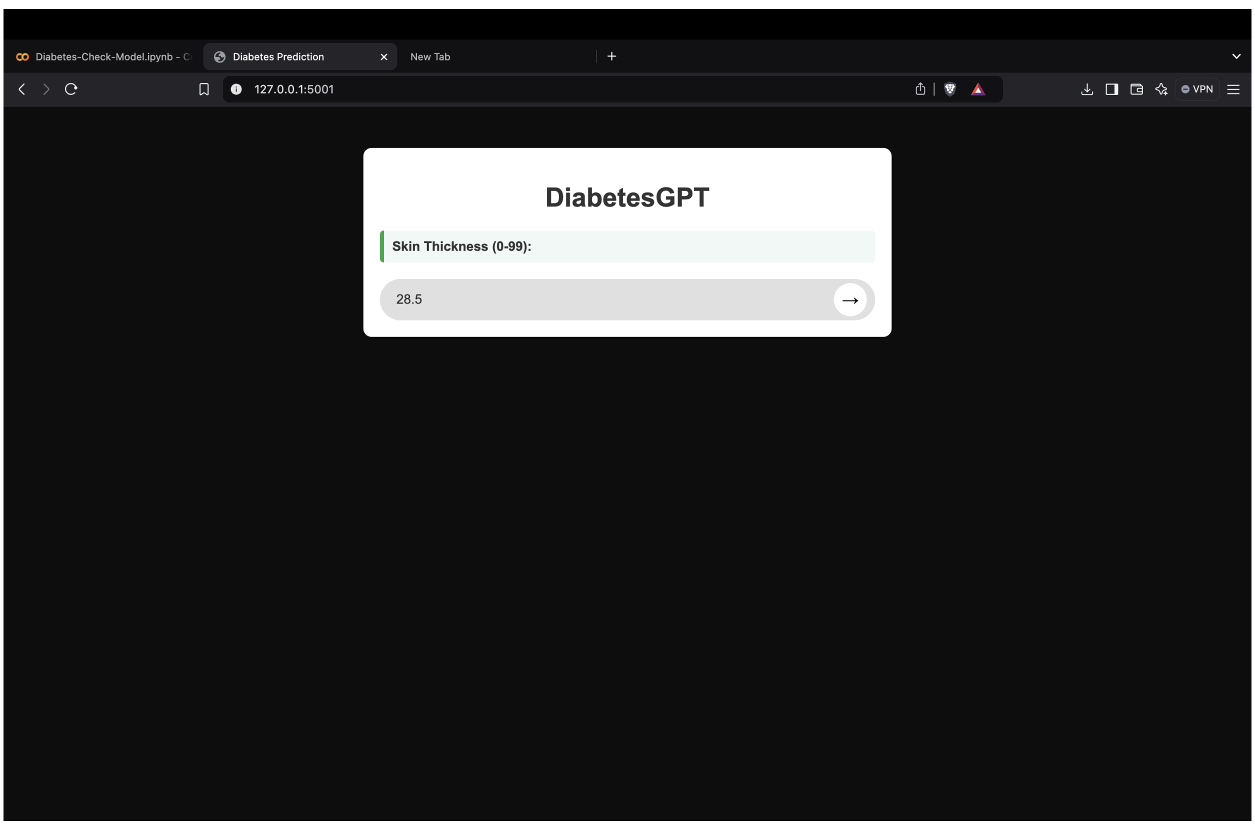
Diabetic Pedigree Function : 0.45

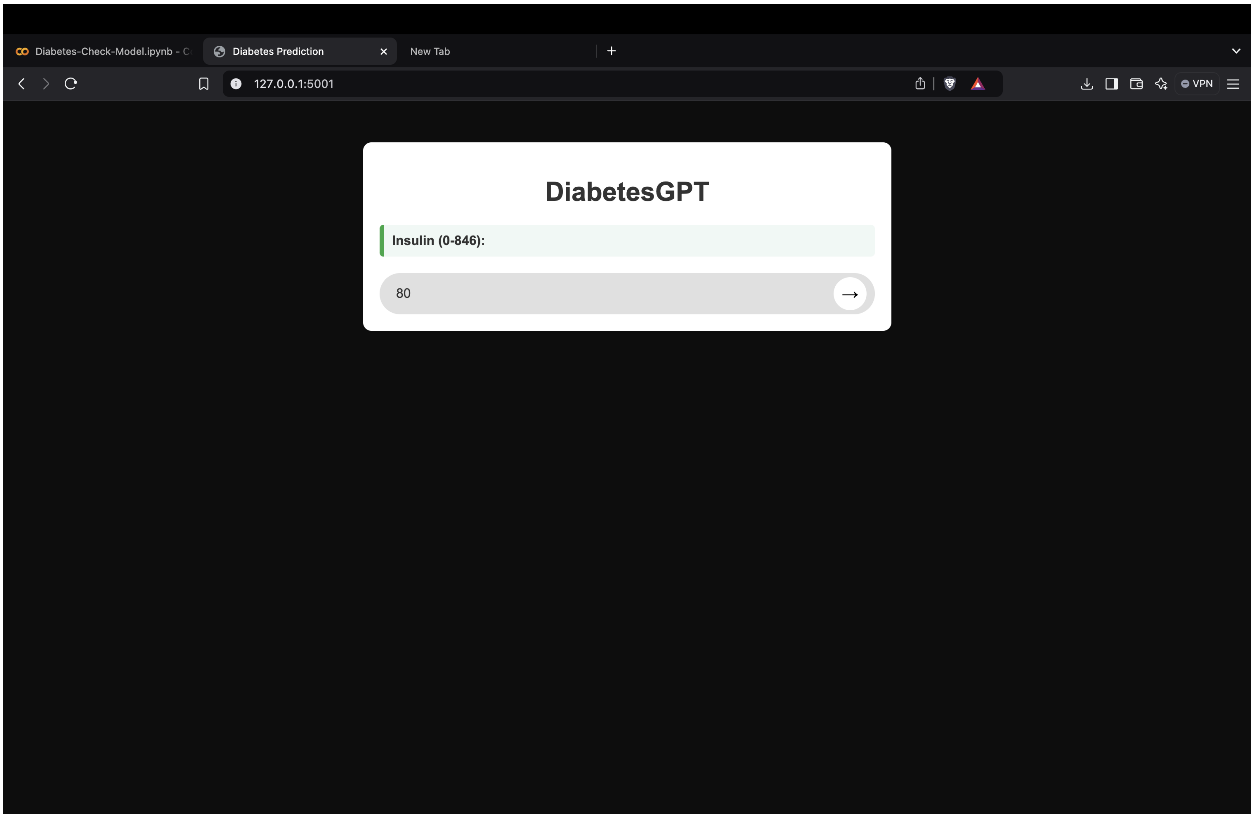
Age : 45

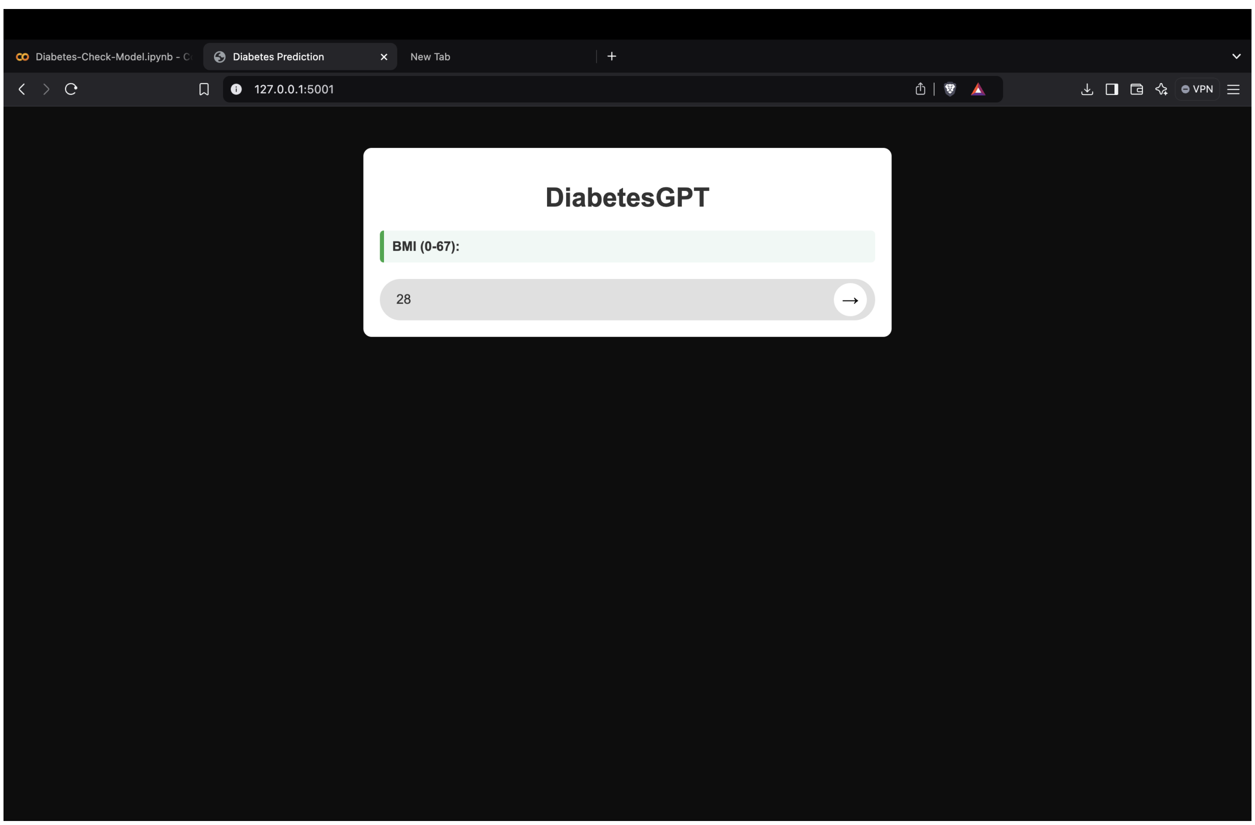

Diabetes-Check-Model.ipynb - C
Diabetes Prediction
New Tab
127.0.0.1:5001
VPN
DiabetesGPT
Pregnancies (0-17):
2

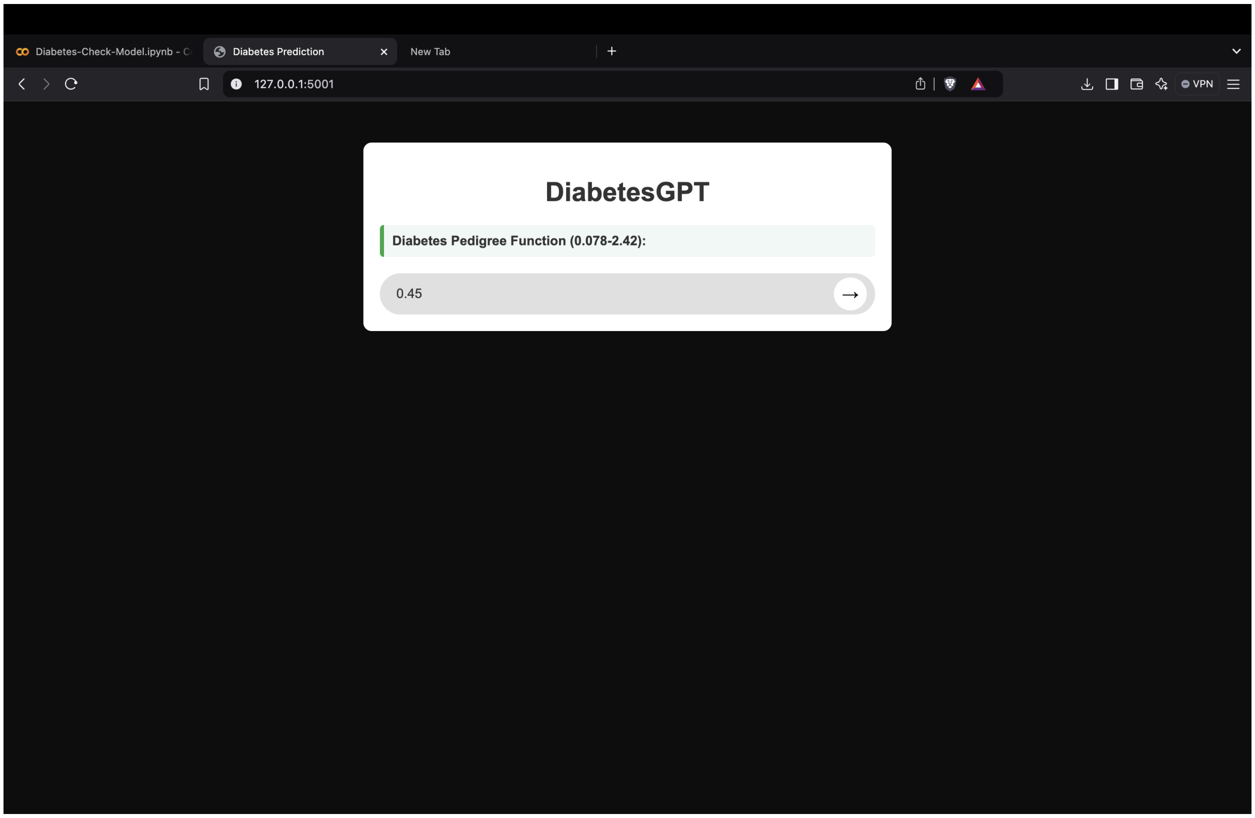

Diabetes-Check-Model.ipynb - C
Diabetes Prediction
New Tab
127.0.0.1:5001
VPN
DiabetesGPT
Glucose (0-199):
120

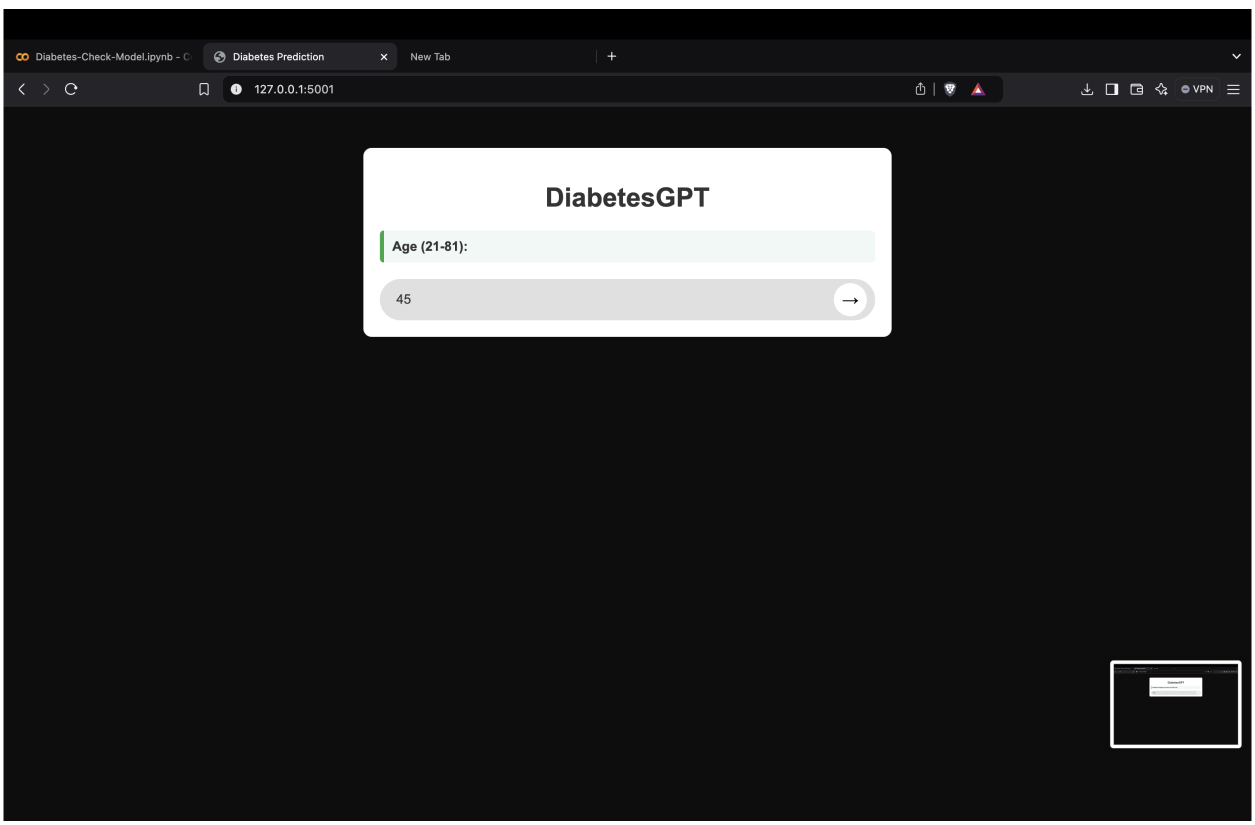

Diabetes-Check-Model.ipynb - C
Diabetes Prediction
New Tab
127.0.0.1:5001
VPN
DiabetesGPT
Blood Pressure (0-122):
80

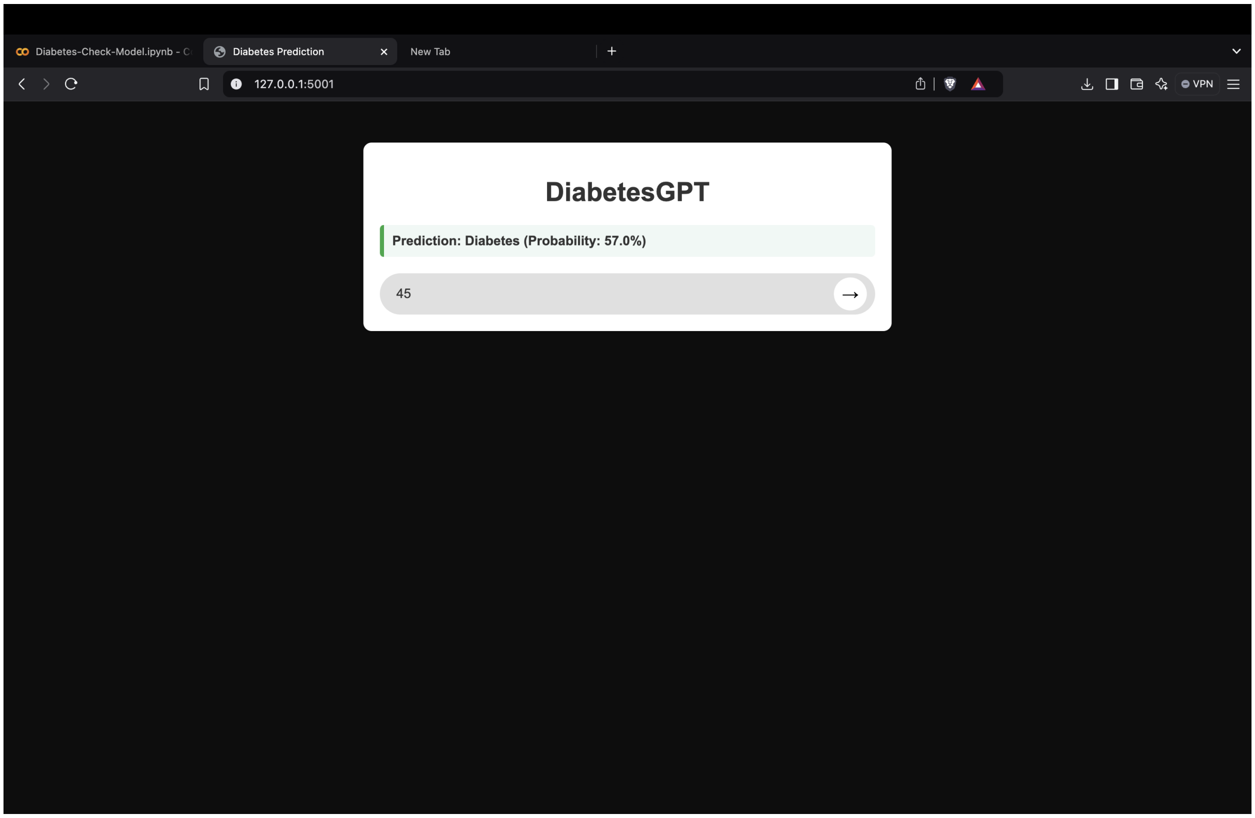

Diabetes-Check-Model.ipynb - C
Diabetes Prediction
New Tab
127.0.0.1:5001
VPN
DiabetesGPT
Skin Thickness (0-99):
28.5


Diabetes-Check-Model.ipynb - C
Diabetes Prediction
New Tab
127.0.0.1:5001
VPN
DiabetesGPT
Insulin (0-846):
80


Diabetes-Check-Model.ipynb - C
Diabetes Prediction
New Tab
127.0.0.1:5001
VPN
DiabetesGPT
BMI (0-67):
28

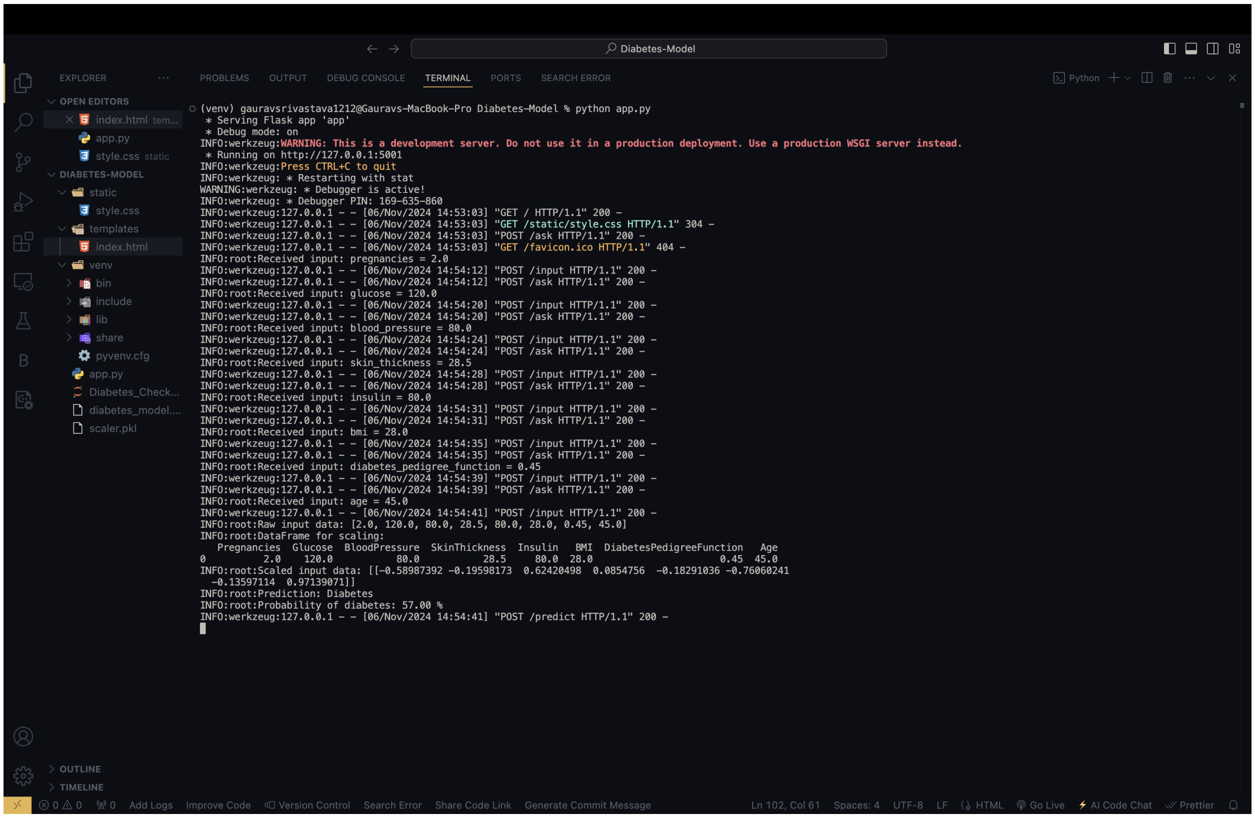

Diabetes-Check-Model.ipynb - C
Diabetes Prediction
New Tab
127.0.0.1:5001
VPN
DiabetesGPT
Diabetes Pedigree Function (0.078-2.42):
0.45


Diabetes-Check-Model.ipynb - C
Diabetes Prediction
New Tab
127.0.0.1:5001
VPN
DiabetesGPT
Age (21-81):
45



Hence it predicts : `57.0 %` ,i.e. , chances of this person getting diabetic !

At the backend , we also do console.log to see that user entered data is successfully getting extracted for the model prediction or not :



Now we look into our model working :

1. Install required libraries:

```
# Install required libraries for imbalanced learning and XGBoost
!pip install -q imbalanced-learn xgboost
```

- This cell installs two libraries:
  - `imbalanced-learn`: Used for handling imbalanced datasets, particularly with techniques like SMOTE (Synthetic Minority Over-sampling Technique).
  - `xgboost`: A library for the XGBoost model, which is often used in classification tasks due to its high performance with structured data.

The `-q` flag is used to suppress unnecessary output during the installation.

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold, cross_val_score, tra
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, rec
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
import warnings
import joblib
from google.colab import files
```

- Imports various libraries and modules:
    - `pandas` and `numpy`: Essential libraries for data handling and numerical operations.
    - `sklearn.model_selection`: Provides tools for splitting data (`train_test_split`), k-fold cross-validation (`KFold`), and evaluating models (`cross_val_score`).
    - `sklearn.linear_model`, `ensemble`, `svm`, `neighbors`, `neural_network`: Includes classifiers like Logistic Regression, Random Forest, SVM, K-Nearest Neighbors, and Neural Network, all used later in model comparisons.
    - `XGBClassifier`: XGBoost classifier from `xgboost`, a high-performing model commonly used in machine learning competitions.
    - `sklearn.metrics`: For evaluating model performance using metrics like accuracy, precision, recall, F1 score, and ROC AUC score.
    - `SMOTE`: Oversampling technique from `imbalanced-learn` to balance the class distribution.
    - `StandardScaler`: Used to scale features for improved model performance.

- `warnings`: To control warning messages.
- `joblib`: For saving trained models.
- `google.colab.files`: Allows downloading files when using Google Colab.

3. Suppress the specific UserWarning:

```
# Suppress the specific UserWarning from sklearn
warnings.filterwarnings("ignore", category=UserWarning, module=
```

Suppresses `UserWarning` from `sklearn.base`, ensuring these warnings do not clutter the output. This can be helpful when some warnings aren't relevant or may confuse the user.

4. Load Dataset:

```
# Load Dataset
url = 'https://raw.githubusercontent.com/plotly/datasets/master/
try:
    data = pd.read_csv(url)
except Exception as e:
    print("Error loading data:", e)

# Data Preprocessing
# Handle missing values and separate features/target
for column in ['Glucose', 'BloodPressure', 'SkinThickness', 'Ins
    data.loc[data[column] == 0, column] = data[column].median()

X = data.drop('Outcome', axis=1)
y = data['Outcome']

# Balance classes using SMOTE
smote = SMOTE(random_state=42)
X, y = smote.fit_resample(X, y)
```

```
# Scale features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

- **Data Loading**: Loads the dataset from a URL.
  - `pd.read_csv(url)`: Reads the CSV file from the given URL. If loading fails, an error message is printed.
- **Data Preprocessing**:
  - **Missing Value Handling**: For selected columns, any instance of `0` (which is likely erroneous for attributes like `Glucose` and `BMI`) is replaced with the median of the column.
  - **Feature/Target Split**: Splits the data into `X` (features) and `y` (target variable `Outcome`).
- **Class Balancing**:
  - Uses `SMOTE` to balance classes in `y`, which can help improve model performance on imbalanced datasets.
- **Feature Scaling**:
  - Scales features to standardize them (mean=0, variance=1), which helps models like logistic regression and neural networks.

5. Split Dataset for Training and Testing Purpose:

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_s

# Check the shapes of the training and testing sets
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

- **Train-Test Split**: Splits data into training and testing sets (80-20 split) with `train_test_split`.

- `stratify=y` ensures both sets have the same proportion of classes, preserving the class distribution.

- **Shape Check**: Outputs the shapes of the training and testing sets, verifying the split was done correctly.

6. Comparison Of Various Models for Comparison:

```
# Define Models for Comparison
models = {
    'Logistic Regression': LogisticRegression(max_iter=200),
    'Random Forest': RandomForestClassifier(),
    'Support Vector Machine': SVC(probability=True),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'XGBoost': XGBClassifier(use_label_encoder=False, eval_metr
    'Neural Network': MLPClassifier(max_iter=300)
}

# Model Training and Evaluation with K-Fold Cross Validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
results = {}

for model_name, model in models.items():
    with np.errstate(divide='ignore', invalid='ignore'):
        accuracy = cross_val_score(model, X_train, y_train, cv=k
        precision = cross_val_score(model, X_train, y_train, cv=
        results[model_name] = {'Accuracy': accuracy, 'Precision
```

- **Model Definitions**:
  - Initializes several models in a dictionary (`models`) to compare their performance on the data.
- **K-Fold Cross-Validation**:
  - `KFold` is set up to split data into 5 subsets for cross-validation.
- **Model Evaluation**:

- For each model, `cross_val_score` calculates average `accuracy` and `precision` across folds.
- Results are stored in `results`, associating each model with its performance metrics.

7. Displaying Precision and Accuracy Results For Each Model :

```
# Displaying results for each model
for model_name, metrics in results.items():
    print(f"{model_name}:")
    print(f"  Accuracy: {metrics['Accuracy']:.2f}")
    print(f"  Precision: {metrics['Precision']:.2f}")
```

- **Displaying Results**:
  - Iterates over the `results` dictionary.
  - For each model, it prints the `Accuracy` and `Precision` scores, formatted to two decimal places.
  - This helps in identifying the models that perform best on both metrics.

8. Train and Save the best performing model:

```
# Train and save the best-performing model
best_model = LogisticRegression(max_iter=200)
best_model.fit(X_train, y_train)

# Save the trained model
joblib.dump(best_model, "diabetes_model.pkl")
```

- **Selecting and Training Best Model**:
  - Chooses `LogisticRegression` as the best model based on previous results. This choice may be based on an assumption, or a selection process from prior cells could guide it.

- The model is trained using `X_train` and `y_train`.

- **Saving the Model**:
  - `joblib.dump` saves the trained model as `diabetes_model.pkl`.
  - This serialized model file can be loaded later for making predictions without retraining.

9. Test the prediction probability of the model:

```
# Test the Best Model on Sample Input Data
# Here, we select the best model based on accuracy and precisio
best_model = models[best_model_name]
best_model.fit(X_train, y_train)  # Train on the full training 

try:
    # Collect user inputs for each feature
    user_input = []
    user_input.append(float(input("Pregnancies (0-17): ")))
    user_input.append(float(input("Glucose (0-199): ")))
    user_input.append(float(input("Blood Pressure (0-122): ")))
    user_input.append(float(input("Skin Thickness (0-99): ")))
    user_input.append(float(input("Insulin (0-846): ")))
    user_input.append(float(input("BMI (0-67): ")))
    user_input.append(float(input("Diabetes Pedigree Function (
    user_input.append(float(input("Age (21-81): ")))

    # Convert to numpy array and scale the input
    user_input_scaled = scaler.transform(np.array([user_input])

    # Make prediction
    with np.errstate(divide='ignore', invalid='ignore'):
        prediction = best_model.predict(user_input_scaled)
        prediction_proba = best_model.predict_proba(user_input_
```

```
    # Output prediction and probability
    print(f"\nSample Prediction (0 = No Diabetes, 1 = Diabetes)
    print(f"Probability of Diabetes: {prediction_proba[0][1]:.2f

except ValueError:
    print("Invalid input. Please enter numeric values.")
```

- `best_model = models[best_model_name]`:
    - Selects the best-performing model based on previously calculated metrics (accuracy and precision).
    - `best_model_name` is likely the name of the model with the highest evaluation scores.
- `best_model.fit(X_train, y_train)`:
    - Retrains the best model on the entire training set to ensure it has the most complete information before making predictions.
    - This final training step is done after K-Fold validation to fine-tune the model.
- Initializes an empty list `user_input`.
- Each line prompts the user to enter a specific feature value for prediction:
    - **Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, Age**
    - Each feature input is converted to `float` to ensure numeric values, and all inputs are appended to the `user_input` list.
- Expected ranges are provided in parentheses to guide valid input values.
- `np.array([user_input])` converts the list of user inputs to a numpy array, compatible with the scaler.
- `scaler.transform(...)` applies scaling to the input, standardizing it according to the scaling parameters derived from the training data.
- `user_input_scaled` now contains the scaled values ready for prediction.

- **Suppressing Warnings**: The `np.errstate(divide='ignore', invalid='ignore')` context manager suppresses warnings (e.g., division by zero or invalid operations) during prediction.
- `best_model.predict(user_input_scaled)`:
  - Predicts the class (0 or 1) for diabetes.
- `best_model.predict_proba(user_input_scaled)`:
  - Calculates the probability for each class. `prediction_proba[0][1]` accesses the probability of the positive class (diabetes).
- Outputs the class prediction and the probability of diabetes.
- `prediction[0]`: Prints the result (0 for non-diabetic, 1 for diabetic).
- `prediction_proba[0][1]:.2f`: Prints the probability with two decimal places for readability.
- Catches any `ValueError` (e.g., if the user inputs a non-numeric value).
- Provides a helpful error message to prompt users for correct input.