**Diabetes Risk Assessment**

**DSAI PROJECT**

**CSE, 6TH SEMESTER**

**Group Members:**

|  |  |
| --- | --- |
| Akankshya Nayak | B121005 |
| Minakshi Sahoo | B121028 |
| Priyambada Onkar | B121040 |

**Faculty:**

**Mr. Sanjay Saxena**



**Department of Computer Science and Engineering**

**International Institute of Information Technology**

**Bhubaneswar – 751003**

Date: 15 th April 2024

**Contents**

1. Objectives
2. Introduction
3. Data
4. Methodology
5. Flow of work
6. Results
7. Conclusion

**Objective of the project**

The project aims to develop a predictive model to identify individuals at high risk of developing diabetes based on various health indicators. The dataset used for this project is the Pima Indians Diabetes Database obtained from Kaggle. The primary objective is to build an accurate predictive model using machine learning techniques to assist healthcare professionals in early detection and intervention strategies for diabetes prevention.

**Introduction**

This project aims to develop a predictive model for diabetes risk assessment using the PIMA Indians Diabetes dataset. Leveraging Convolutional Neural Networks (CNNs), the model will analyze clinical parameters such as glucose levels, blood pressure, and BMI to predict the likelihood of diabetes development. The provided code demonstrates the implementation of the CNN model and its training process, while the PIMA dataset serves as the foundation for model development and evaluation. By accurately predicting diabetes risk, this project aims to facilitate early intervention and improve patient outcomes in diabetes management.

1. **Data Complexity**: Medical datasets like the Pima Indians Diabetes dataset often contain complex relationships between input features and the target variable. CNNs excel at capturing spatial patterns in data, making them suitable for tasks where feature locality matters, such as analyzing medical images or sequential data.
2. **Automatic Feature Learning**: CNNs automatically learn hierarchical representations of features from raw data during training. This eliminates the need for manual feature engineering

## **Methodology**

Data Loading: The code starts by loading the dataset. In this example, the Pima Indians Diabetes dataset is used. This dataset contains information about diagnostic measurements for diabetes, such as glucose level, blood pressure, etc.

Data Preprocessing:

The dataset is split into features (X) and the target variable (y).

The data is split into training and testing sets using the train\_test\_split function from scikit-learn.

Features are standardized using StandardScaler to ensure that all features have the same scale.

Data Reshaping: Since CNNs expect input data in a specific format, the features are reshaped to add a third dimension. For a 1D CNN, this third dimension represents the number of channels (which is 1 in this case).

Model Definition:

A sequential Keras model is created.

The model architecture consists of a 1D convolutional layer followed by max-pooling, flattening, dense layers with ReLU activation, dropout regularization, and a final dense layer with a sigmoid activation function.

Model Compilation: The model is compiled with the Adam optimizer and binary cross-entropy loss function. The accuracy metric is used to evaluate the model during training.

Model Training: The model is trained using the fit method. The training data, along with validation data, is provided. Training is performed over a specified number of epochs with a defined batch size.

Model Evaluation and Visualization:

The training history (accuracy and loss) is stored during model training.

Matplotlib is used to visualize the accuracy and loss over epochs.

Two plots are generated: one showing accuracy vs. epochs and the other showing loss vs. epochs. These plots help in understanding the model's performance and training progress.

Results Analysis:

The accuracy plot shows how the accuracy of the model changes with each epoch during training. It helps in understanding whether the model is improving or overfitting.

The loss plot shows how the loss (error) of the model changes with each epoch. A decreasing loss indicates that the model is learning effectively.

**WORKFLOW**

Importing Libraries: Necessary libraries are imported including NumPy, Pandas, Matplotlib, scikit-learn, and TensorFlow.

Loading the Dataset: The Pima Indians Diabetes dataset is loaded from the provided URL and column names are assigned.

Data Preprocessing: Features (X) and the target variable (y) are separated. The data is split into training and testing sets using a 80-20 split. Features are standardized using StandardScaler. Additionally, the data is reshaped to add a third dimension required for CNN input.

Model Definition: A CNN model is defined using the Sequential API in Keras. It consists of a 1D convolutional layer, max-pooling layer, flattening layer, dense hidden layer with ReLU activation, dropout layer for regularization, and a final dense output layer with a sigmoid activation function.

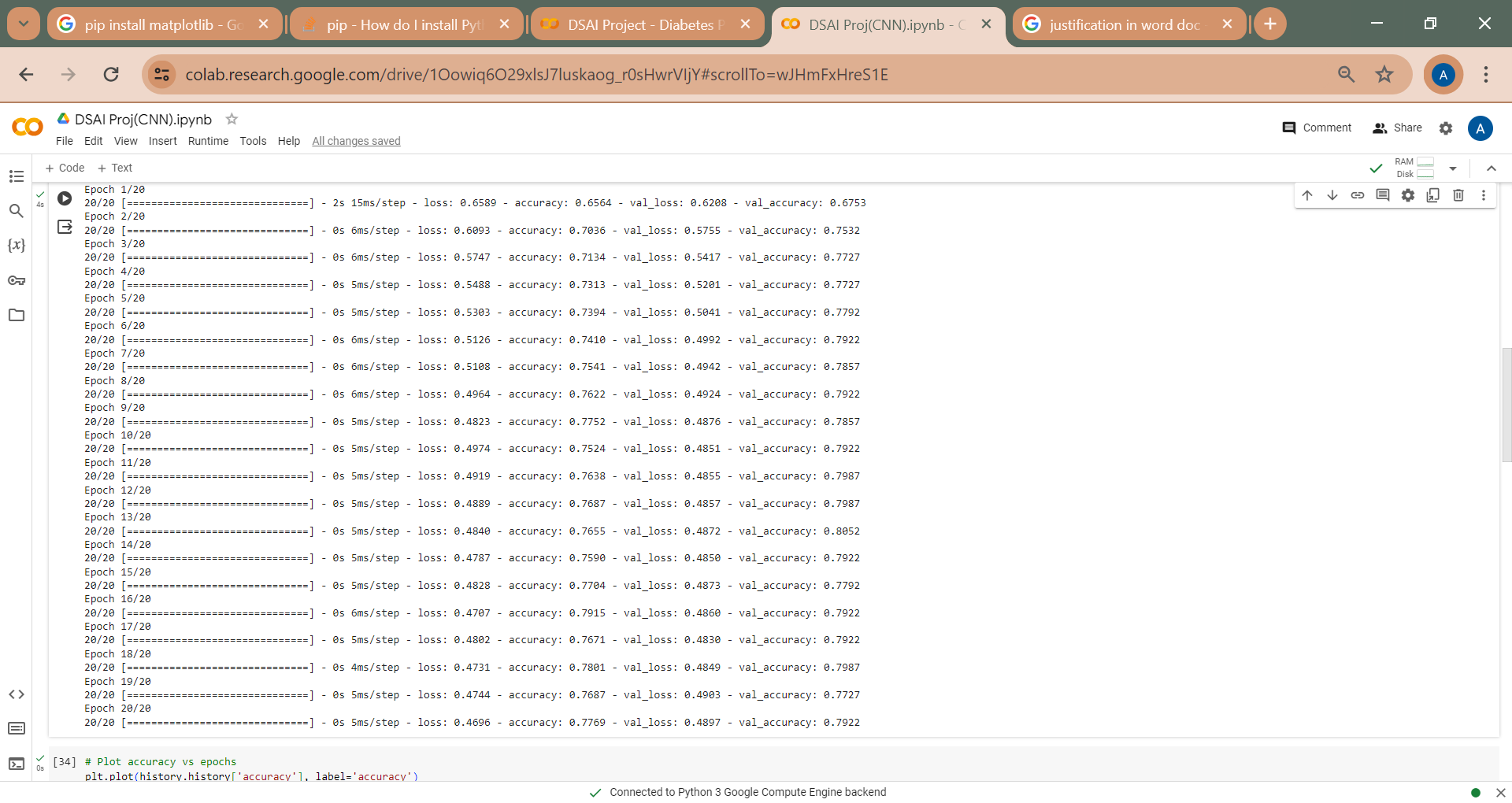
Model Compilation: The model is compiled with the Adam optimizer, binary crossentropy loss function (suitable for binary classification), and accuracy metric for evaluation.

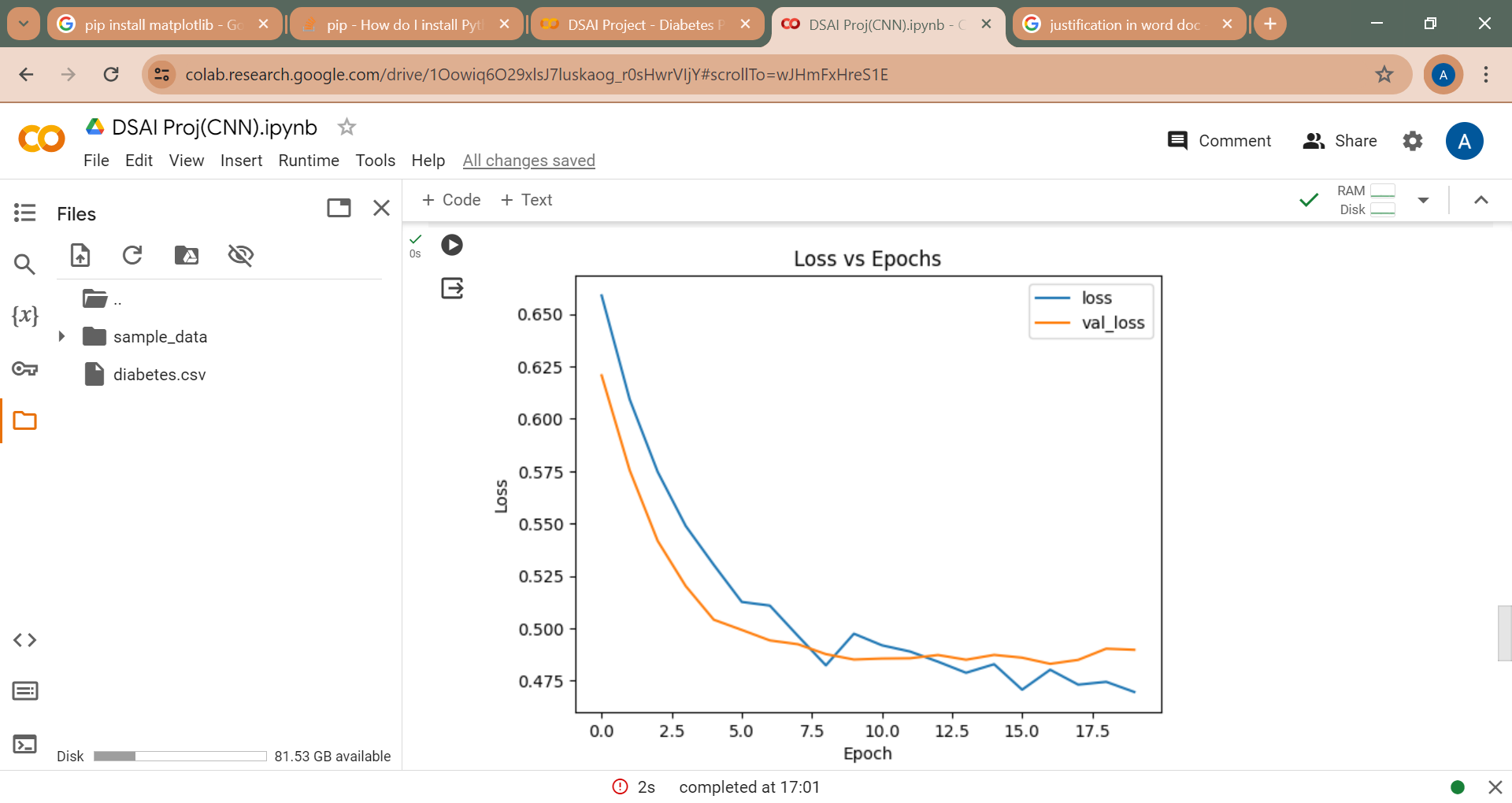
Model Training: The model is trained using the training data for 20 epochs with a batch size of 32. Validation data is provided to monitor performance on unseen data during training.

Plotting Accuracy vs Epochs: The training and validation accuracy values are plotted against epochs to visualize how the accuracy changes during training.

Plotting Loss vs Epochs: The training and validation loss values are plotted against epochs to visualize how the loss changes during training.

**Pictures Of Training Process and Graphs**





# **Analysis of Results**

## Performance Metrics

The accuracy\_score function is used to calculate the accuracy of the model on both the training and testing data.

The training accuracy represents how well the model performs on the data it was trained on.

The testing accuracy represents how well the model generalizes to unseen data, which is crucial for assessing the model's real-world performance.

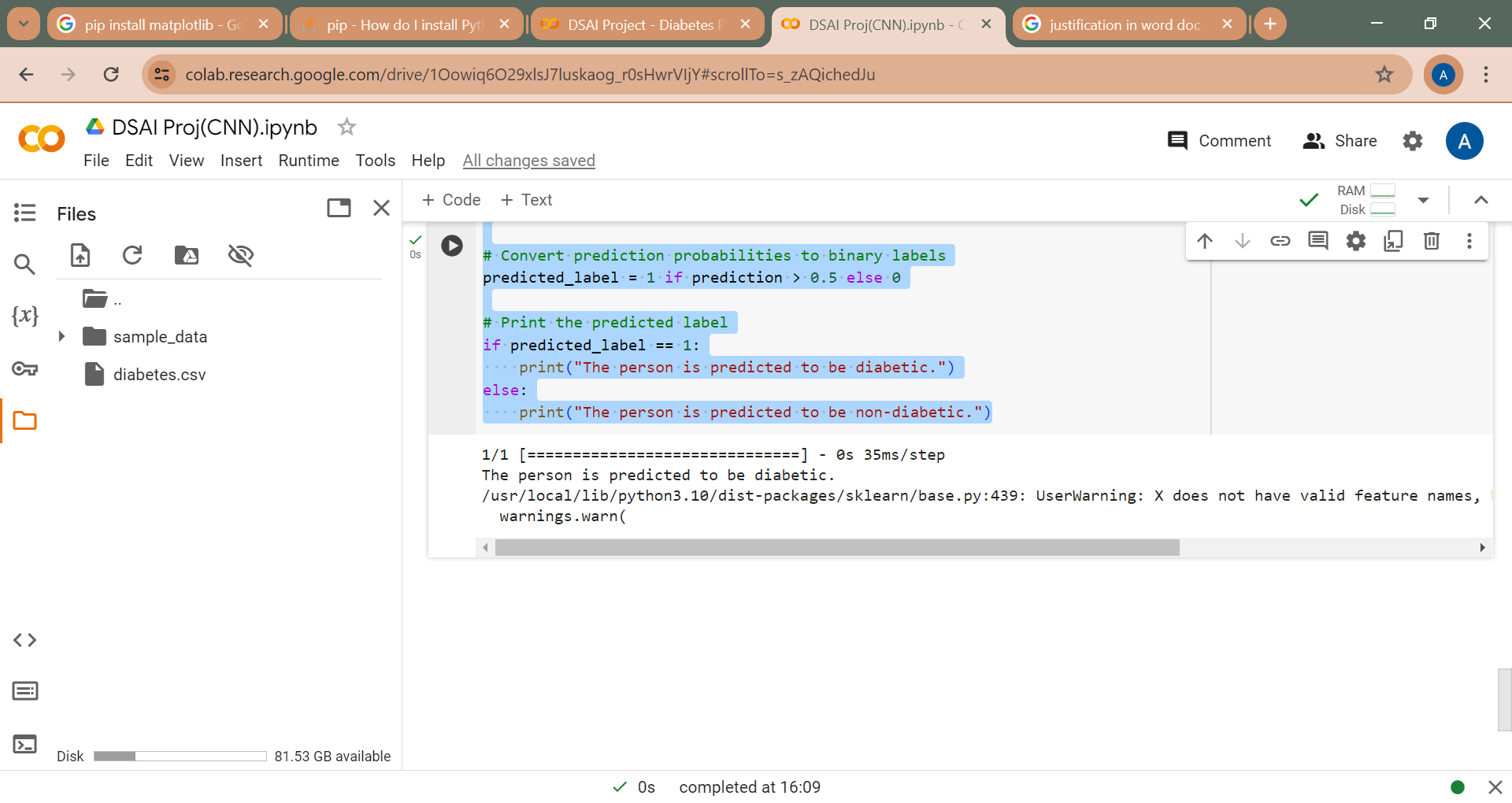
#### Model Performance:

* **Training Accuracy:** 0.7769
* **Testing Accuracy:** 0.7922

The CNN model trained on the PIMA Indians Diabetes dataset achieved a training accuracy of 77.69% and a testing accuracy of 79.22%. These accuracy scores indicate the model's ability to effectively learn from the training data and generalize to unseen data.

## Testing with a sample

input\_data = [5, 166, 72, 19, 175, 25.8, 0.587, 51]



# **Conclusion**

The CNN model demonstrates promising performance in predicting diabetes risk based on the PIMA dataset. With a testing accuracy of 79.22%, the model shows potential for assisting in early detection and intervention for individuals at risk of diabetes.

# References

- Kaggle: Pima Indians Diabetes Database

- Scikit-learn

# Implemented Code

# Import necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import StandardScaler

import tensorflow as tf

import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout

# Load the dataset (example using the Pima Indians Diabetes dataset)

url = ‘/content/diabetes.csv

data = pd.read\_csv(url)

# Separate features and target variable

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

y = data['Outcome']

# Split 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)

# Standardize features

scaler = StandardScaler()

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

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

# Reshape data for CNN input (add 3rd dimension)

X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)

X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)

# Define the CNN model

model = Sequential([

Conv1D(filters=32, kernel\_size=3, activation='relu', input\_shape=(X\_train.shape[1], 1)),

MaxPooling1D(pool\_size=2),

Flatten(),

Dense(64, activation='relu'),

Dropout(0.5),

Dense(1, activation='sigmoid')

])

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_data=(X\_test, y\_test))

# Plot accuracy vs epochs

plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val\_accuracy'], label='val\_accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Accuracy vs Epochs')

plt.show()

# Plot loss vs epochs

plt.plot(history.history['loss'], label='loss')

plt.plot(history.history['val\_loss'], label='val\_loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.title('Loss vs Epochs')

plt.show()

input\_data = np.array([5, 166, 72, 19, 175, 25.8, 0.587, 51]) # Input data

input\_data = scaler.transform(input\_data.reshape(1, -1)).reshape(1, input\_data.shape[0], 1)

# Make predictions

prediction = model.predict(input\_data)

# Convert prediction probabilities to binary labels

predicted\_label = 1 if prediction > 0.5 else 0

# Print the predicted label

if predicted\_label == 1:

print("The person is predicted to be diabetic.")

else:

print("The person is predicted to be non-diabetic.")