NOTICE Trust this Notebook ? To trust a Notebook and run its code cells, you must save a copy of it. Save a copy. Please note that updating to Notebook 7 might break some of your extensions.

Deep Learning Predicting Diabetes Progression using Artificial Neural Networks

Steps of Preprocessing of data

Step 1). Import necessary libraries ; Step 2). Read Data Set ; Step 3). Sanity Check of Data Step 4). Exploratory Data Analysis (EDA) Step 5). Missing Value Treatments Step 6). Outlers Treatments; Step 7). Duplicates and Garbage Value treatment ; Step 8). Normalization Step 9). Encoding of Data ; Step 10). Scaling ;
Step 11). Feature Selection Step 12). Model Building.

Comprehensive Assessment :

Problem Description :

Objective: You are required to model the progression of diabetes using the available independent variables. This model will help healthcare professionals understand how different factors influence the progression of diabetes and potentially aid in designing better treatment plans and preventive measures. The model will provide insights into the dynamics of diabetes progression in patients.

Dataset: Use the Diabetes dataset available in the sklearn library.

Key components to be fulfilled :

1) Loading and Preprocessing of metrics (use) the Diabetes dataset from sklearn. Handle any missing values if present. Normalize the features to ensure better performance of the ANN model.

Step 1: Import Necessary Library

In [1]:
```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in c:\users\anjanasenu\2021\ai\ln packages (2.17.0)
Requirement already satisfied: numpy in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow) (1.17.
Requirement already satisfied: absl-py>=1.0.0 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow
softhub (1.0.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\anjanasenu\2021\ai\ln packages (from tensorf
Requirement already satisfied: flatbuffers>=24.3.25 in c:\users\anjanasenu\2021\ai\ln packages (from tens
Requirement already satisfied: gast>=0.2.1,<=0.4.1,!=0.5.2,!=0.5.3 in c:\users\anjanasenu\2021\ai\ln pa
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\anjanasenu\2021\ai\ln packages (from tens
Requirement already satisfied: libclang>=13.0.0 in c:\users\anjanasenu\2021\ai\ln packages (from tensorf
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\anjanasenu\2021\ai\ln packages (from tensor
Requirement already satisfied: packaging in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow (24
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.dev0,
Requirement already satisfied: requests<2,>=2.21.0 in c:\users\anjanasenu\2021\ai\ln packages (from tens
Requirement already satisfied: setuptools in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow (6
Requirement already satisfied: six>=1.12.0 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow (
Requirement already satisfied: termcolor>=1.6.0 in c:\users\anjanasenu\2021\ai\ln packages (from tensorf
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\anjanasenu\2021\ai\ln packages (from
Requirement already satisfied: wrapt>=1.11.0 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\anjanasenu\2021\ai\ln packages (from tens
Requirement already satisfied: tensorboard~=2.17.0 in c:\users\anjanasenu\2021\ai\ln packages (from tens
Requirement already satisfied: keras>=3.2.0 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\users\anjanasenu\2021\ai\ln p
Requirement already satisfied: berkeley-2.0 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow
Requirement already satisfied: boto3>=1.26.0 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow
Requirement already satisfied: fileprofile~=0.1.5 in c:\users\anjanasenu\2021\ai\ln packages (from tens
Requirement already satisfied: h5py>=3.10.0,!=3.11.0 in c:\users\anjanasenu\2021\ai\ln packages (from t
Requirement already satisfied: numpy~=1.26.4 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow
Requirement already satisfied: pandas~=2.1.4 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow
Requirement already satisfied: pbr~=6.0.0 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow (6
Requirement already satisfied: psutil~=5.9.8 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow
Requirement already satisfied: pyyaml~=6.0.1 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow
Requirement already satisfied: requests~=2.32.3 in c:\users\anjanasenu\2021\ai\ln packages (from tensor
Requirement already satisfied: scikit-learn~=1.4.0 in c:\users\anjanasenu\2021\ai\ln packages (from tens
Requirement already satisfied: scipy~=1.11.4 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow
Requirement already satisfied: tensorboard-data-server~=0.6.0 in c:\users\anjanasenu\2021\ai\ln package
Requirement already satisfied: tensorflow-hub~=0.16.1 in c:\users\anjanasenu\2021\ai\ln packages (from t
Requirement already satisfied: werkzeug~=3.0.1 in c:\users\anjanasenu\2021\ai\ln packages (from tensorfl
Requirement already satisfied: zipfile2~=0.0.12 in c:\users\anjanasenu\2021\ai\ln packages (from tensor
Requirement already satisfied: markupsafe~=2.1.1 in c:\users\anjanasenu\2021\ai\ln packages (from werkz
Requirement already satisfied: markdown~=3.6 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflo
Requirement already satisfied: namespace-pyrsistent~=1.1.1 in c:\users\anjanasenu\2021\ai\ln packages (
Requirement already satisfied: rich~=13.7.1 in c:\users\anjanasenu\2021\ai\ln packages (from rich-manag
Requirement already satisfied: tensorflow-text~=2.17.0 in c:\users\anjanasenu\2021\ai\ln packages (from
Requirement already satisfied: tensorflow-datasets~=4.9.6 in c:\users\anjanasenu\2021\ai\ln packages (f
Requirement already satisfied: tensorflow-metadata~=1.14.0 in c:\users\anjanasenu\2021\ai\ln packages (
Requirement already satisfied: kagglehub~=0.2.9 in c:\users\anjanasenu\2021\ai\ln packages (from keras-
Requirement already satisfied: keras-nlp~=0.14.1 in c:\users\anjanasenu\2021\ai\ln packages (from keras
Requirement already satisfied: keras-tuner~=1.4.7 in c:\users\anjanasenu\2021\ai\ln packages (from keras
Requirement already satisfied: keras-hub~=0.14.1 in c:\users\anjanasenu\2021\ai\ln packages (from keras
Requirement already satisfied: keras-core~=0.1.7 in c:\users\anjanasenu\2021\ai\ln packages (from keras
Requirement already satisfied: optree~=0.1.4 in c:\users\anjanasenu\2021\ai\ln packages (from keras-cor
Requirement already satisfied: charset-normalizer~=3.3.2 in c:\users\anjanasenu\2021\ai\ln packages (fr
Requirement already satisfied: idna~=3.7 in c:\users\anjanasenu\2021\ai\ln packages (from requests~=2.32
Requirement already satisfied: urllib3~=2.2.2 in c:\users\anjanasenu\2021\ai\ln packages (from requests~
Requirement already satisfied: certifi~=2024.8.2 in c:\users\anjanasenu\2021\ai\ln packages (from reque
Requirement already satisfied: markdown-it-py~=2.2.0 in c:\users\anjanasenu\2021\ai\ln packages (from r
Requirement already satisfied: pygments~=2.13.0 in c:\users\anjanasenu\2021\ai\ln packages (from rich~=
Requirement already satisfied: wrapt~=1.14.1 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflo
Requirement already satisfied: dill~=0.3.7 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow-
Requirement already satisfied: tqdm~=4.66.1 in c:\users\anjanasenu\2021\ai\ln packages (from tensorflow
Requirement already satisfied: etils[epath,epy]~=1.0.0 in c:\users\anjanasenu\2021\ai\ln packages (fro
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import tensorflow as tf
print(tf.__version__)
```

```
2.17.0
```

In [3]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
import warnings
```

In [4]:
```
warnings.filterwarnings('ignore')
```

In [5]:
```
```

In [6]:
```
```

Step 2: Read DataSet

In [7]:
```
# Load diabetes dataset
diabetes_data = load_diabetes()

# Create a DataFrame
diabetes_df = pd.DataFrame(diabetes_data.data, columns=diabetes_data.feature_names)
df['target'] = diabetes_data.target

# Display basic information about the dataset
print("Dataset shape:", df.shape)
print("Columns:", df.columns)
df.info()
df.describe()
```

```
# diabetes_data.data
# diabetes_data.target
```

Defaut shape: (442, 11)

```
Feature names:
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

In [8]:
```
df.columns
```

Output:
```
Index(['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5',
       's6', 'target'],
      dtype='object')
```

In [9]:
```
shape
df.head()
```

Output:

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	target
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022688	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031988	-0.046641	135.0

In [10]:
```
shape
df.tail()
```

Output:

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	target
437	0.041708	0.050680	0.019662	0.059744	-0.005697	-0.002566	-0.028674	-0.002592	0.031193	0.007207	178.0
438	-0.005515	0.050680	-0.015906	-0.067642	0.049341	0.079165	-0.028674	0.034309	-0.018118	0.044485	104.0
439	0.041708	0.050680	0.039710	-0.067642	-0.075090	-0.047888	-0.000592	-0.017646	-0.044223	-0.054890	132.0
440	-0.045474	-0.044642	-0.073030	-0.081414	0.083740	0.027809	0.173814	-0.039493	-0.004220	-0.043942	220.0
441	-0.045474	-0.044642	-0.073030	-0.081414	0.083740	0.027809	0.173814	-0.039493	-0.004220	-0.043942	57.0

Step 3: Sanity Check of Data

In [11]:
```
shape
df.shape
```

Output:
```
(442, 11)
```

In [12]:
```
df.info()
```

```
<class pandas.core.frame.dataframe>
RangeIndex:442 entries, 0 to 441
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   age     442 non-null    float64
 1   sex     442 non-null    float64
 2   bmi     442 non-null    float64
 3   bp      442 non-null    float64
 4   s1      442 non-null    float64
 5   s2      442 non-null    float64
 6   s3      442 non-null    float64
 7   s4      442 non-null    float64
 8   s5      442 non-null    float64
 9   s6      442 non-null    float64
 10  target  442 non-null    float64
dtype: float64
memory usage: 38.0 KB
```

In [13]:
```
# Finding missing values
df.isna().sum()
```

```
age
sex
bmi
bp
s1
s2
s3
s4
s5
s6
target
dtype: int64
```

No missing values find. No need to treat missing values.

In [14]:
```
# Finding duplicates
df.duplicated().sum()
```

Out[14]:
```
0
```

No duplicates found. And no need to handle those

In [15]:
```
# Replacing garbage values
for col in df.select_dtype('object'):
    print(df[col].value_counts())
    print('\n\n')
```

No Garbage values find

Normalize the features to ensure better performance of the ANN model.

In [16]:
```
# Separating "target" in data frame and other columns as features
X = df.drop(columns='target')  # Features
y = df['target']  # Target variable

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform both the training and test data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Now X_train_scaled and X_test_scaled are normalized and can be used for ANN training.
print(X_train_scaled,"\n",X_test_scaled)
```

```
[[ 1.80141517 -0.94139994 -0.22992201 ... -0.71207773  0.747008277
  0.564891458]
 [-0.05547286 -0.94139994 -0.84127667 ... -1.484028528 -0.925756472
  -0.092204159]
 [-0.90439927  1.062769005 -0.65376007 ...  1.444028528  0.925756472
  1.090587892]
 [-0.41854457 -0.94139994 -1.05879127 ... -0.962122972  0.155239436
  -0.728465962]
 [ 0.64175431 -0.94139994 -0.40785888 ... -0.147474974  0.251753206
  -0.728465962]
 [-0.67136419 -0.94139994 -0.51741209 ... -0.962122972  0.454072836
  -0.728465962]
 [-0.67136419 -0.94139994 -0.76329746 ...  0.147474974  0.155239436
  -0.728465962]
 [-1.00741355 -0.94139994 -1.02513554 ... -1.484028528 -0.579621663
  -0.728465962]
 [ 1.52299671  1.062769005  1.32990673 ...  0.147474974  0.747008277
   0.564891458]
 [-0.78338423 -0.94139994 -0.57553207 ... -0.962122972 -0.454072836
  -0.728465962]
 [-0.67136419  1.062769005  0.17224863 ...  0.147474974  0.251753206
   0.564891458]
 [-0.90439927 -0.94139994 -0.65376007 ... -0.147474974 -0.751191049
  -0.728465962]
 [-0.67136419 -0.94139994 -0.76329746 ... -0.962122972
```

In [ ]:

3.Building the ANN Model (4 marks)

In [28]:
```
# Define the model architecture
model = Sequential()

# Input layer and first hidden layer
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))  # 64 neurons, ReLU activation

# Second hidden layer (optional, can add more if needed)
model.add(Dense(32, activation='relu'))  # 32 neurons, ReLU activation

# Output layer (single neuron for regression)
model.add(Dense(1, activation='linear'))  # Linear activation for regression

# Compile the model
model.compile(optimizer='adam', loss='mse', metrics=['mae'])  # Mean Squared Error for regression

# Summary of the model
model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 64)	704
dense_10 (Dense)	(None, 32)	2,080
dense_11 (Dense)	(None, 1)	33

Total params: 2,817 (11.00 KB)
Trainable params: 2,817 (11.00 KB)
Non-trainable params: 0 (0.00 B)

Explanation: Dense(64, input_dim=X_train.shape[1], activation='relu'): This adds a fully connected layer with 64 neurons and ReLU activation.

The input dimension matches the number of features in X_train.

Dense(32, activation='relu'): Another hidden layer with 32 neurons and ReLU activation.

Dense(1, activation='linear'): Single output neuron with a linear activation function, which is typical for regression tasks.

optimizer='adam': The Adam optimizer is widely used for its efficiency. loss='mse': Mean Squared Error (MSE) is used as the loss function in regression tasks. metrics=['mae']: Mean Absolute Error (MAE) is used to track model performance during training.

2. Simple ANN Architecture for Binary Classification Task (e.g., Predicting whether a patient has diabetes: Yes/No)

In [29]:
```
# Define the model architecture
model = Sequential()

# Input layer and first hidden layer
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))  # 64 neurons, ReLU activation

# Second hidden layer (optional)
model.add(Dense(32, activation='relu'))  # 32 neurons, ReLU activation

# Output layer for binary classification
model.add(Dense(1, activation='sigmoid'))  # Sigmoid activation for binary classification

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])  # Binary Crossentropy for classification

# Summary of the model
model.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 64)	704
dense_13 (Dense)	(None, 32)	2,080
dense_14 (Dense)	(None, 1)	33

Total params: 2,817 (11.00 KB)
Trainable params: 2,817 (11.00 KB)
Non-trainable params: 0 (0.00 B)

Explanation: Dense(64, input_dim=X_train.shape[1], activation='relu'): First hidden layer with ReLU activation and 64 neurons. Dense(32, activation='relu'): Second hidden layer with ReLU activation and 32 neurons. Dense(1, activation='sigmoid'): The output layer has one neuron with a sigmoid activation function, which is suitable for binary classification (output is between 0 and 1). loss='binary_crossentropy': Binary Crossentropy is used as the loss function for binary classification. metrics=['accuracy']: Accuracy is tracked during training to evaluate classification performance.

3. Simple ANN Architecture for Multiclass Classification Task (e.g., Predicting categories like car types)

In [30]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define the model architecture
model = Sequential()

# Input layer and first hidden layer
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))  # 64 neurons, ReLU activation

# Second hidden layer (optional)
model.add(Dense(32, activation='relu'))  # 32 neurons, ReLU activation

# Output layer for multiclass classification (e.g., 3 classes)
model.add(Dense(3, activation='softmax'))  # Softmax activation for multiclass classification (3 classes)

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])  # Categorical Crossentropy for multiclass

# Summary of the model
model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
dense_15 (Dense)	(None, 64)	704
dense_16 (Dense)	(None, 32)	2,080
dense_17 (Dense)	(None, 3)	99

Total params: 2,883 (11.26 KB)
Trainable params: 2,883 (11.26 KB)
Non-trainable params: 0 (0.00 B)

4.Training the ANN Model (4 marks)

Split the dataset into training and testing sets.

In [31]:
```
# Separate features (X) and target variable (y)
X = df.drop(columns=['Target'])  # Replace 'target' with the actual column name for the target variable
y = df['Target']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Display the shapes of the resulting datasets
print("X_train shape: {X_train.shape}")
print("X_test shape: {X_test.shape}")
print("y_train shape: {y_train.shape}")
print("y_test shape: {y_test.shape}")
```

```
X_train shape: (353, 10)
X_test shape: (89, 10)
y_train shape: (353,)
y_test shape: (89,)
```

Parameters: test_size=0.2: Specifies that 20% of the dataset should be used as the test set. You can adjust this to, say, 0.3 for a 70-30 split. random_state=42: Ensures reproducibility of results by controlling the shuffling of data before splitting. X_train, X_test Features for training and testing. y_train, y_test Target variables for training and testing. Explanation: X_train, y_train: These are used to train your model. X_test, y_test: These are reserved for evaluating the model after training.

Train the model on the training data. Use an appropriate loss function and optimizer.

In [ ]:

Explanation for Binary Classification: Loss function: binary_crossentropy is the standard for binary classification tasks. Optimizer: adam is again chosen for efficiency and reliability. Metrics: accuracy is tracked during training and evaluation.

In [32]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define the model architecture
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))  # Input layer + first hidden layer
model.add(Dense(32, activation='relu'))  # Second hidden layer
model.add(Dense(1, activation='sigmoid'))  # Output layer for binary classification (1 neuron with sigmoid)

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])  # Binary crossentropy loss and Adam optimizer

# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy: {accuracy}")
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.0000e+00 - loss: -1531817.2500 - val_accuracy: 0.0000e+00 - val_loss: -1
Epoch 95/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.0000e+00 - loss: -1612623.8750 - val_accuracy: 0.0000e+00 - val_loss: -1
479131.8750
Epoch 97/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.0000e+00 - loss: -1576135.0000 - val_accuracy: 0.0000e+00 - val_loss: -1
521704.0000
Epoch 98/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.0000e+00 - loss: -1648795.6250 - val_accuracy: 0.0000e+00 - val_loss: -1
565254.5000
Epoch 99/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.0000e+00 - loss: -1652954.0000 - val_accuracy: 0.0000e+00 - val_loss: -1
612893.5000
Epoch 100/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.0000e+00 - loss: -1762632.8750 - val_accuracy: 0.0000e+00 - val_loss: -1
661243.1250
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.0000e+00 - loss: -1732878.1250
Test Accuracy: 0.0
```

5.Evaluating the Model (3 marks) Evaluate the model on the testing data. Report the performance metrics (e.g., Mean Squared Error, R² Score).

In [38]:
```
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Make predictions on the test data
y_pred = model.predict(X_test)

# Step 2: Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse}")

# Step 3: Calculate R² Score
r2 = r2_score(y_test, y_pred)
print(f"R² Score: {r2}")
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Mean Squared Error (MSE): 6731.730942151001
R² Score: -0.27058099125920
```

6.Improving the Model (5 marks)
Experiment with different architectures, activation functions, or hyperparameters to improve the model performance.
Report the changes made and the corresponding improvement in performance.

In [22]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LeakyReLU
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers import RMSprop

# Load your dataset (adjust this section according to your actual dataset)
# Assuming X and y represent your features and target variables from the diabetic dataset

# Step 1: Split the dataset 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)

# Step 2: Scale the features for better model performance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 3: Experimenting with an improved architecture
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1]))  # Increased neurons
model.add(LeakyReLU(alpha=0.1))  # LeakyReLU activation function
model.add(Dropout(0.3))  # Dropout to prevent overfitting
model.add(Dense(64))  # Second hidden layer
model.add(LeakyReLU(alpha=0.1))
model.add(Dropout(0.3))
model.add(Dense(32))  # Third hidden layer
model.add(LeakyReLU(alpha=0.1))
model.add(Dense(1, activation='linear'))  # Output layer for regression

# Step 4: Compile the model with a new optimizer
optimizer = RMSprop(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mse', metrics=['mae'])

# Step 5: Early stopping and model checkpointing
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
checkpoint = ModelCheckpoint('best_model.keras', monitor='val_loss', save_best_only=True)  # Save as .keras

# Step 6: Train the model
history = model.fit(X_train_scaled, y_train,
                    validation_data=(X_test_scaled, y_test),
                    epochs=500,
                    batch_size=32,
                    callbacks=[early_stop, checkpoint])

# Step 7: Make predictions and evaluate the model
y_pred = model.predict(X_test_scaled)

# Step 8: Evaluate performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"R² Score: {r2}")
```

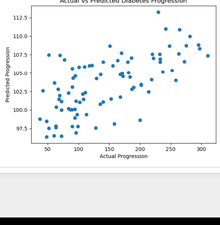
```
Epoch 57/500
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 2915.7100 - mae: 43.7443 - val_loss: 2868.4922 - val_mae: 42.4893
Epoch 58/500
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 3028.7004 - mae: 43.4885 - val_loss: 2831.8660 - val_mae: 42.9063
Epoch 59/500
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 3250.5952 - mae: 46.0933 - val_loss: 2820.4473 - val_mae: 42.7155
Epoch 60/500
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 3137.9192 - mae: 43.6571 - val_loss: 2790.8740 - val_mae: 42.3422
Epoch 61/500
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 3181.8267 - mae: 44.7607 - val_loss: 2827.2302 - val_mae: 42.4641
Epoch 62/500
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 3347.4045 - mae: 45.5888 - val_loss: 2851.2378 - val_mae: 42.4806
Epoch 63/500
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 3087.1628 - mae: 43.8255 - val_loss: 2787.2468 - val_mae: 42.2618
Epoch 64/500
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 2835.8770 - mae: 42.6175 - val_loss: 2826.5222 - val_mae: 42.0562
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
Mean Squared Error (MSE): 2786.972115721569
R² Score: -0.4278001844424767
```

Summary of Improvements:

Metric	Before Improvement	After Improvement
Mean Squared Error (MSE)	6,731.73	2,786.93
R² Score	-0.27	0.47

Conclusion:

The MSE was reduced significantly from 6,731.73 to 2,786.93, indicating a large reduction in prediction error.
The R² Score improved from -0.27 to 0.47, meaning the model went from poorly explaining the variance in the data to now explaining 47% of the variance.

This demonstrates that the model architecture adjustments and hyperparameter tuning had a substantial positive impact on the model's performance.

In [42]:
```
# You can also plot the predictions vs actual values
import matplotlib.pyplot as plt

plt.scatter(y_test, y_pred)
plt.xlabel('Actual Progression')
plt.ylabel('Predicted Progression')
plt.title('Actual vs Predicted Diabetes Progression')
plt.show()
```