Step 1: Data Loading

In this step, I load the CSV file into a Pandas Data Frame. This allows us to easily work with the data in Python. After loading, I checked the first few rows of the dataset to understand its structure.

[1]:
```
# Step 1: Data Loading
import pandas as pd

# Load the dataset
file_path = "C:\\Users\\mavin\\Downloads\\Healthcare-Diabetes.csv"
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
data.head()
```

[1]:

	Id	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1	6	148	72	35	0	33.6	0.627	50	1
1	2	1	85	66	29	0	26.6	0.351	31	0
2	3	8	183	64	0	0	23.3	0.672	32	1
3	4	1	89	66	23	94	28.1	0.167	21	0
4	5	0	137	40	35	168	43.1	2.288	33	1

We can see the various columns including `Pregnancies`, `Glucose`, `Blood Pressure`, and others that are relevant for diabetes prediction.

Step 2: Data Preprocessing

In this step, we focus on cleaning the dataset by handling invalid or missing values. Specifically, we replace any zero values in the columns that should not have zeros (such as `Insulin`, `Skin Thickness`, `Blood Pressure`, and `BMI`) with the median of their respective columns. This ensures that our data is more representative and prevents zeros from skewing the analysis.

[2]:
```
# Step 2: Data Preprocessing
# Replace 0 values in certain columns with the median value of that column
columns_to_replace = ['Insulin', 'SkinThickness', 'BloodPressure', 'BMI']

# For each column in the list, replace 0s with the median value of that column
for column in columns_to_replace:
    median_value = data[column].median()
    data[column] = data[column].replace(0, median_value)

# Display the dataset after preprocessing
data.head()
```

[2]:

	Id	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1	6	148	72	35	37	33.6	0.627	50	1
1	2	1	85	66	29	37	26.6	0.351	31	0
2	3	8	183	64	23	37	23.3	0.672	32	1
3	4	1	89	66	23	94	28.1	0.167	21	0
4	5	0	137	40	35	168	43.1	2.288	33	1

Changes Made:

- Other columns, like BMI and Blood Pressure, are unaffected as they did not contain zeros in the sample shown.

This preprocessing step ensures that the data is cleaned up and ready for modeling, which helps improve the accuracy of the predictions.

Step 3: Splitting Data into Features and Target Variables

In this step, we separate the dataset into two distinct parts: the feature set (X) and the target variable (y).

- **Features (X)**: These are the input variables that we will use to predict the target. In this case, we exclude the `Id` and `Outcome` columns, as well as the `Glucose` level that we are trying to predict. The features include:
  - `Pregnancies`
  - `Blood Pressure`
  - `Skin Thickness`
  - `Insulin`
  - `BMI`
  - `DiabetesPedigreeFunction`
  - `Age`
- **Target (y)**: This is the variable we want to predict, which in this case is the `Glucose` level.

After executing the code, the data is structured as follows:

Features (X):

[3]:
```
# Step 3: Splitting Data into Features and Target Variables
# Drop 'Id', 'Outcome', and 'Glucose' columns from the feature set
X = data.drop(['Id', 'Outcome', 'Glucose'], axis=1)

# Target variable (Glucose)
y = data['Glucose']

# Display feature set and target variable
X.head(), y.head()
```

[3]:
```
(   Pregnancies  BloodPressure  SkinThickness  Insulin   BMI  \
 0            6             72             35       37  33.6
 1            1             66             29       37  26.6
 2            8             64             23       37  23.3
 3            1             66             23       94  28.1
 4            0             40             35      168  43.1

    DiabetesPedigreeFunction  Age
 0                     0.627   50
 1                     0.351   31
 2                     0.672   32
 3                     0.167   21
 4                     2.288   33  ,
 0    148
 1     85
 2    183
 3     89
 4    137
 Name: Glucose, dtype: int64)
```

Summary:

- We prepare our dataset for model training by isolating the features and the target variable, making it easier to build and evaluate our predictive model in subsequent steps. This separation is crucial for machine learning tasks, as it allows us to train the model on the features and test it against the actual target values.

Step 4: Train-Test Split

In this step, we divide the dataset into two subsets: a training set and a testing set. This is a crucial step in machine learning to ensure that the model can generalize well to unseen data.

- **Training Set**: This is the portion of the data used to train the model. The model learns patterns from this data.

- **Testing Set**: This subset is used to evaluate the model's performance. It contains data that the model has not seen during training, allowing us to assess how well the model can predict outcomes on new data.

After executing the code, the output `(2214, 7)` and `(554, 7)` indicates the following:

- **(2214, 7)**: The training set contains 2,214 samples (or rows) and 7 features (or columns). This means we have a substantial amount of data to train our model effectively.
- **(554, 7)**: The testing set contains 554 samples and 7 features. This size is sufficient for evaluating the model's performance after training.

```
[4]: # Step 4: Train-Test Split
     from sklearn.model_selection import train_test_split

     # Split 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 sizes of the training and testing sets
     X_train.shape, X_test.shape
```

```
[4]: ((2214, 7), (554, 7))
```

Summary:

By splitting the data into training and testing sets, we ensure that we can effectively train our model while also validating its predictive capability, which helps prevent overfitting and ensures robust performance when making predictions on new, unseen data.

Step 5: Model Training and Coefficient Extraction

In this step, we train a regression model (such as Linear Regression) using the training dataset. The model learns the relationship between the features (input variables) and the target variable (Glucose levels) based on the data provided.

[5]:
```python
# Step 5: Model Training
from sklearn.linear_model import LinearRegression

# Initialize the linear regression model
model = LinearRegression()

# Train the model on the training set
model.fit(X_train, y_train)

# Display the model coefficients
print("Model Coefficients:", model.coef_)
```

```
Model Coefficients: [ 0.02649862  0.30780676 -0.11452361  0.11247878  0.51435239  4.53824217
  0.6392661 ]
```

Summary:

- The model coefficients provide insights into how each feature impacts the prediction of glucose levels. Positive coefficients indicate a positive relationship, while negative coefficients indicate a negative relationship. This information is essential for understanding the influence of different variables in predicting the target variable.

Step 6: Model Evaluation Using Mean Squared Error (MSE)

In this step, we evaluate the performance of the trained model using the Mean Squared Error (MSE) metric. MSE quantifies the average squared difference between the predicted values and the actual values of the target variable (Glucose levels).

]:
```python
# Step 6: Model Evaluation
from sklearn.metrics import mean_squared_error

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Display the MSE
print(f"Mean Squared Error (MSE): {mse}")
```

```
Mean Squared Error (MSE): 833.5194855859364
```

Interpretation of MSE:

- The MSE value indicates how well the model is performing. A lower MSE signifies that the predictions are closer to the actual values, while a higher MSE indicates larger errors in the predictions.
- In this case, an MSE of approximately **833.52** suggests that, on average, the model's predictions deviate from the actual glucose levels by about 29 (since the square root of 833.52 gives approximately 28.9, representing the RMSE).

Summary:

- Evaluating the model using MSE helps determine its accuracy in predicting the target variable. This step is crucial for assessing model performance and making necessary adjustments or improvements to enhance prediction capabilities.

Reflection on the Problem and Solution

Problem Overview: The task was to predict glucose levels in patients based on various health-related factors such as pregnancies, blood pressure, insulin levels, BMI, age, and other medical metrics. This is critical in understanding and potentially diagnosing diabetes-related conditions.

Challenges Faced:

1. **Handling Missing or Inconsistent Data**: Several columns, like `Insulin`, `Skin Thickness`, and `Blood Pressure`, had zeros, which are unrealistic in a medical context. Proper data preprocessing was necessary to replace these zero values with more meaningful ones, such as medians, ensuring that the dataset was clean for model training.
2. **Feature Selection**: Selecting the appropriate features for predicting glucose levels was another key consideration. We excluded the `Outcome` (which represents diabetes diagnosis) and focused only on continuous predictors related to health metrics.
3. **Model Training**: Linear regression was used as a basic model to predict glucose levels. This simple model works well for identifying linear relationships between input features and the target variable (glucose levels), but it may not capture more complex, non-linear patterns.

4. **Evaluation**: The Mean Squared Error (MSE) was used to evaluate the model's accuracy. The MSE of 833.52 is relatively high, indicating that the model's predictions have considerable error, suggesting room for improvement.

Solution Highlights:

5. **Data Preprocessing**: Replacing missing values with medians was an effective approach to deal with zeros in the dataset. This step was crucial to prevent skewed results or unrealistic predictions, especially in healthcare where precision is vital.
6. **Model Coefficients**: The coefficients from the linear regression model provided insight into how each feature influenced glucose levels. For example, a positive coefficient for `BMI` indicates that higher BMI tends to increase predicted glucose levels, which aligns with medical understanding.
7. **Model Performance**: The MSE of the model indicates the prediction error is relatively high, showing that a linear regression model may not be fully adequate for this type of problem. The model could likely be improved with more advanced algorithms like decision trees, random forests, or neural networks, which can capture complex, non-linear relationships between features.
8. **Feature Engineering**: Consider creating new features that might capture more relationships in the data (e.g., interaction terms between health metrics).
9. **Model Optimization**: Explore more sophisticated models such as Random Forests, Gradient Boosting, or Neural Networks, which could potentially reduce prediction errors by capturing more intricate patterns in the data.
10. **Cross-Validation**: Implement cross-validation to better assess model performance across different data splits, ensuring that the model generalizes well to unseen data.
11. **Hyperparameter Tuning**: Experiment with different model hyperparameters (e.g., for more complex models) to further optimize performance.

reflection:

The problem of predicting glucose levels based on medical factors is complex and requires careful data handling and thoughtful model selection. While the initial linear regression model provides a basic prediction framework, further refinement and the use of more sophisticated techniques are necessary to improve accuracy, which could ultimately support better healthcare decision-making for diabetes management.