Assignment 1 Report

Assignment 1

1. Data Loading and Preparation: [1 mark]

Use the dataset available at this Kaggle link: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database.

Ensure the dataset is properly split into features (X) and target (y).

Perform feature scaling using 'StandardScaler' to normalize the features for better performance of the logistic regression model.

2. Model Implementation: [2 marks]

Split the dataset into training and testing sets (e.g., 80% training, 20% testing) using train_test_split. Implement the Logistic Regression classifier using 'sklearn.linear_model.LogisticRegression'. Train the Logistic Regression model on the training set. 3. Evaluation: [1.5 marks]

Evaluate the model's performance on the test set using accuracy, precision, recall, and F1-score. Present a classification report and a confusion matrix for the results. 4. Submission Requirements: [0.5 mark]

Submit the Python code implementing the solution. Provide a brief report (300-500 words) explaining your approach, the preprocessing steps, feature scaling importance, and the results obtained.

Solution Approach

Data Preparation

1. All necessary Python libraries numpy and pandas are imported
2. Using pd from pandas library, the diabetes dataset is loaded.
3. The dataset is splitted into features (X) and target (y). Features are number of pregnancies, Glucose level, BP, Insulin, BMI, SkinThickness, DiabetesPedigreeFunction, Age and the target is Outcome. Outcome 1 (Diabetic) and 0 (Non-diabetic).
4. The sklearn libraries is used Logistic Regression Model. Use StandardScaler for feature normalization.

In [2]:
```python
import numpy as np
import pandas as pd

df = pd.read_csv('diabetes.csv')
df.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPed
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

Data Presentation

In [3]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [4]:
```python
df.describe(include = 'all').T
```

Out[4]:

	count	mean	std	min	25%	50
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.50
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.00
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.37
Age	768.0	33.240885	11.760232	21.000	24.00000	29.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.00

In [4]:

```
df.corr()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness
Pregnancies	1.000000	0.129459	0.141282	-0.081672
Glucose	0.129459	1.000000	0.152590	0.057328
BloodPressure	0.141282	0.152590	1.000000	0.207371
SkinThickness	-0.081672	0.057328	0.207371	1.000000
Insulin	-0.073535	0.331357	0.088933	0.436783
BMI	0.017683	0.221071	0.281805	0.392573
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928
Age	0.544341	0.263514	0.239528	-0.113970
Outcome	0.221898	0.466581	0.065068	0.074752

Split the Data Frame into Features and Target

In [5]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = df.drop(columns=['Outcome']) # features
y = df['Outcome'] #Target
X.head()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPed
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

Applying StandardScalar

1. StandardScalar is applied to normalize the features (X). The standardization rescales the data so that mean = 0 and standard deviation = 1. Standardization is useful for algorithms that are sensitive to the scale of the data, except logistic regression, it is applicable for k-means clustering, or SVMs.
2. fit_transform - It has two operations - fit(X) and transform(X). fit(X) - Here the scaler learns the mean and standard deviation of each feature (column) in the dataset (X). The transform(X) applies the learned transformation to the data X and returns a scaled feature.

In [6]:

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
print(X)
print(X.shape)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
(768, 8)
```

Implementation of Training Algorithm

Steps:

1. The dataset is splitted into training and testing sets (e.g., 80% training, 20%

testing) using train_test_split.
2. The Logistic Regression classifier is created using 'sklearn.linear_model.LogisticRegression'
3. The Logistic Regression model is trained on the training set.

In [7]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression


# Divide the dataset into train and test sets (80% train, 20% test)
X_tr,X_te,y_tr,y_te = train_test_split(X, y, test_size=0.2)

# Create and train logistic regression classifier
diabetic_classifier_lr = LogisticRegression()
diabetic_classifier_lr.fit(X_tr, y_tr)
```

Out[7]:
```
▾ LogisticRegression
LogisticRegression()
```

Model Performance Report

In [8]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Make predictions on the test dataset
y_pred = diabetic_classifier_lr.predict(X_te)

# Evaluate model performance
accuracy_dia = accuracy_score(y_te, y_pred)
precision_dia = precision_score(y_te, y_pred, zero_division=1)
recall_dia = recall_score(y_te, y_pred, zero_division=1)
f1_dia = f1_score(y_te, y_pred, zero_division=1)

# Create confusion matrix and classification report
conf_matrix_dia = confusion_matrix(y_te, y_pred)
class_report_dia = classification_report(y_te, y_pred)

# Output the results
print(f"Accuracy: {accuracy_dia}")
print(f"Precision: {precision_dia}")
print(f"Recall: {recall_dia}")
print(f"F1 Score: {f1_dia}")
print("Confusion Matrix:")
print(conf_matrix_dia)
print("Classification Report:")
print(class_report_dia)
```

```
Accuracy: 0.7727272727272727
Precision: 0.6944444444444444
Recall: 0.5102040816326531
F1 Score: 0.5882352941176471
Confusion Matrix:
[[94 11]
 [24 25]]
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.90      0.84       105
           1       0.69      0.51      0.59        49

    accuracy                           0.77       154
   macro avg       0.75      0.70      0.72       154
weighted avg       0.76      0.77      0.76       154
```

Discussions on results:

1. Accuracy = 0.8181 or 81.81 which is pretty good is detecting correctly the diabetic cases.
2. Precision = 0.6046 or 60.46% for class1 (diabetic). It means when the model predicts a person is diabetic, it's correct 60.5% of the time, which is okay.
3. Recall = 0.7027 or 70.27% for calss1 (diabetic). It implies when the model can correctly identify 70.27% diabetic. Recall is better than precision means that model does not miss many real postive cases (false negative).
4. f1 Score = 0.65 => this is harmonic mean of Precion and Recall. This is okay, but it can be improved.
5. Confusion Matrix: Class 0 (non-diabetic): 100 true negatives, 17 false positives and Class 1 (diabetic): 26 true positives, 11 false negatives.

Overall: support -The number of true instances for each class. macro avg - The unweighted average of precision, recall, and f1-score across all classes. weighted avg - A weighted average of the precision, recall, and f1-score

In Summary the model shows good accuracy, it wrongly idetifies quite few diabetic (false positive) and misses few diabetic cases. This Logistic regression classifier can be improved with other ML model or Deep Learning Model

In [ ]: