NAME: KINGSTON ENGINEERING COLLEGE

COLLEGE CODE:5113

DOMAIN : ARTIFICIAL INTELLIGENCE

PROJECT TITLE:AI BASED DIABETES PREDICTION SYSTEM COLLEGE

PROJECT MEMBERS: NAAN MUDHALVAN ID:

SATHYA P (TEAM LEADER ) 511321104089

SAKTHI S 511321104084

SWETHA P 511321104100

PRIYADHARSHINI B 511321104074

INTRODUCTION:

IN THE TECHNOLOGY OF ARTIFICIAL INTELLIGENCE TO CONTINUE BUILDING THE DIABETES

PREDICTION PROJECT, WE&#39;LL GO THROUGH THE FOLLOWING STEPS:

SELECTING A MACHINE LEARNING ALGORITHM:

For a binary classification problem like diabetes prediction, several algorithms can be effective.

Common choices include logistic regression, support vector machines (svm), random forest, gradient

boosting, neural networks, etc.

Since you mentioned using artificial intelligence, we could consider using a deep learning model like

a convolutional neural network (cnn) or a recurrent neural network (rnn).

DATA PREPROCESSING:

Load and preprocess the dataset. This includes handling missing values, normalizing/standardizing

features, and splitting the data into training and testing sets.

FEATURE SELECTION/ENGINEERING:

Identify and select relevant features. You might also want to consider feature engineering to create

new features or transform existing ones.

MODEL TRAINING:

Train the selected machine learning algorithm on the pre-processed data using the training set.

MODEL EVALUATION:

Evaluate the model using appropriate metrics. For a binary classification problem like diabetes

prediction, common metrics include accuracy, precision, recall, f1-score, etc.

FINE-TUNING AND HYPERPARAMETER OPTIMIZATION:

Depending on the algorithm chosen, there may be hyperparameters that need to be tuned to

achieve better performance. Techniques like grid search or random search can be used for this

purpose.

CROSS-VALIDATION:

Perform cross-validation to assess the model&#39;s generalization performance.

MODEL INTERPRETABILITY (OPTIONAL BUT RECOMMENDED):

Depending on the model used, it might be helpful to understand which features are driving the

predictions. Techniques like shape values, lime, or feature importance plots can be used.

DEPLOYMENT:

Once you have a satisfactory model, deploy it for practical use. This can be done using various

methods like creating a web application, deploying , or integrating into an existing system.

MONITORING AND MAINTENANCE:

Continuously monitor the model&#39;s performance in the real-world setting. If necessary, retrain the

model with new data or update it with improved versions.

CONCLUSION:

Therefore, in this technology we built our project by selecting a machine learning algorithm, training

the model, and evaluated its performance by using the given dataset.

PROGRAM:

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

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

from sklearn.metrics import accuracy\_score, confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

In [2]:

#read the data file

data = pd.read\_csv(&quot;/kaggle/input/diabetes-data-set/diabetes.csv&quot;)

data.head()

Out[2]:

Pregna

ncies

Gluc

ose

BloodPr

essure

SkinThic

kness

Insu

lin

B

M

I

DiabetesPedigre

eFunction

A

ge

Outc

ome

0 6 148 72 35 0 33.

6 0.627 50 1

1 1 85 66 29 0 26.

6 0.351 31 0

2 8 183 64 0 0 23.

3 0.672 32 1

3 1 89 66 23 94 28.

1 0.167 21 0

4 0 137 40 35 168 43.

1 2.288 33 1

In [3]:

data.describe()

Out[3]:

Pregn

ancie

s

Gluc

ose

BloodP

ressure

SkinT

hickne

ss

Insul

in BMI

DiabetesPedi

greeFunctio

n

Age Outc

ome

co

un

t

768.0

00000

768.0

0000

0

768.00

0000

768.00

0000

768.0

0000

0

768.0

0000

0

768.000000

768.0

0000

0

768.0

0000

0

m 3.845 120.8 69.105 20.536 79.79 31.99 0.471876 33.24 0.348

ea

n

052 9453

1

469 458 9479 2578 0885 958

st

d

3.369

578

31.97

2618

19.355

807

15.952

218

115.2

4400

2

7.884

160 0.331329 11.76

0232

0.476

951

mi

n

0.000

000

0.000

000

0.0000

00

0.0000

00

0.000

000

0.000

000 0.078000 21.00

0000

0.000

000

25

%

1.000

000

99.00

0000

62.000

000

0.0000

00

0.000

000

27.30

0000 0.243750 24.00

0000

0.000

000

50

%

3.000

000

117.0

0000

0

72.000

000

23.000

000

30.50

0000

32.00

0000 0.372500 29.00

0000

0.000

000

75

%

6.000

000

140.2

5000

0

80.000

000

32.000

000

127.2

5000

0

36.60

0000 0.626250 41.00

0000

1.000

000

m

ax

17.00

0000

199.0

0000

0

122.00

0000

99.000

000

846.0

0000

0

67.10

0000 2.420000 81.00

0000

1.000

000

In [4]:

data.isnull().sum()

Out[4]:

Pregnancies 0

Glucose 0

BloodPressure 0

SkinThickness 0

Insulin 0

BMI 0

DiabetesPedigreeFunction 0

Age 0

Outcome 0

dtype: int64

We can see there few data for columns Glucose , Insulin, skin thickenss, BMI and Blood

Pressure which have value as 0. That&#39;s not possible,right? you can do a quick search to see that

one cannot have 0 values for these. Let&#39;s deal with that. we can either remove such data or

simply replace it with their respective mean values. Let&#39;s do the latter.

In [5]:

#here few misconception is there lke BMI can not be zero, BP can&#39;t be zero,

glucose, insuline can&#39;t be zero so lets try to fix it

# now replacing zero values with the mean of the column

data[&#39;BMI&#39;] = data[&#39;BMI&#39;].replace(0,data[&#39;BMI&#39;].mean())

data[&#39;BloodPressure&#39;] =

data[&#39;BloodPressure&#39;].replace(0,data[&#39;BloodPressure&#39;].mean())

data[&#39;Glucose&#39;] = data[&#39;Glucose&#39;].replace(0,data[&#39;Glucose&#39;].mean())

data[&#39;Insulin&#39;] = data[&#39;Insulin&#39;].replace(0,data[&#39;Insulin&#39;].mean())

data[&#39;SkinThickness&#39;] =

data[&#39;SkinThickness&#39;].replace(0,data[&#39;SkinThickness&#39;].mean())

In [6]:

#now we have dealt with the 0 values and data looks better. But, there

still are outliers present in some columns.lets visualize it

fig, ax = plt.subplots(figsize=(15,10))

sns.boxplot(data=data, width= 0.5,ax=ax, fliersize=3)

Out[6]:

&lt;Axes: &gt;

In [7]:

data.head()

Out[7]:

Pregna

ncies

Gluc

ose

BloodPr

essure

SkinThi

ckness

Insuli

n

B

M

I

DiabetesPedigr

eeFunction

A

ge

Outc

ome

0 6 148.

0 72.0 35.00000

0

79.799

479

33

.6 0.627 50 1

1 1 85.0 66.0 29.00000

0

79.799

479

26

.6 0.351 31 0

2 8 183.

0 64.0 20.53645

8

79.799

479

23

.3 0.672 32 1

3 1 89.0 66.0 23.00000

0

94.000

000

28

.1 0.167 21 0

4 0 137.

0 40.0 35.00000

0

168.00

0000

43

.1 2.288 33 1

In [8]:

#segregate the dependent and independent variable

X = data.drop(columns = [&#39;Outcome&#39;])

y = data[&#39;Outcome&#39;]

In [9]:

# separate dataset into train and test

X\_train, X\_test, y\_train, y\_test =

train\_test\_split(X,y,test\_size=0.25,random\_state=0)

X\_train.shape, X\_test.shape

Out[9]:

((576, 8), (192, 8))

In [10]:

import pickle

##standard Scaling- Standardization

def scaler\_standard(X\_train, X\_test):

#scaling the data

scaler = StandardScaler()

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

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

#saving the model

file = open(&#39;standardScalar.pkl&#39;,&#39;wb&#39;)

pickle.dump(scaler,file)

file.close()

return X\_train\_scaled, X\_test\_scaled

In [11]:

X\_train\_scaled, X\_test\_scaled = scaler\_standard(X\_train, X\_test)

In [12]:

X\_train\_scaled

Out[12]:

array([[ 1.50755225, -1.09947934, -0.89942504, ..., -1.45561965,

-0.98325882, -0.04863985],

[-0.82986389, -0.1331471 , -1.23618124, ..., 0.09272955,

-0.62493647, -0.88246592],

[-1.12204091, -1.03283573, 0.61597784, ..., -0.03629955,

0.39884168, -0.5489355 ],

...,

[ 0.04666716, -0.93287033, -0.64685789, ..., -1.14021518,

-0.96519215, -1.04923114],

[ 2.09190629, -1.23276654, 0.11084355, ..., -0.36604058,

-0.5075031 , 0.11812536],

[ 0.33884418, 0.46664532, 0.78435594, ..., -0.09470985,

0.51627505, 2.953134 ]])

In [13]:

log\_reg = LogisticRegression()

log\_reg.fit(X\_train\_scaled,y\_train)

Out[13]:

LogisticRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust

the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with

nbviewer.org.

In [14]:

## Hyperparameter Tuning

## GridSearch CV

from sklearn.model\_selection import GridSearchCV

import numpy as np

import warnings

warnings.filterwarnings(&#39;ignore&#39;)

# parameter grid

parameters = {

&#39;penalty&#39; : [&#39;l1&#39;,&#39;l2&#39;],

&#39;C&#39; : np.logspace(-3,3,7),

&#39;solver&#39; : [&#39;newton-cg&#39;, &#39;lbfgs&#39;, &#39;liblinear&#39;],

}

In [15]:

logreg = LogisticRegression()

clf = GridSearchCV(logreg, # model

param\_grid = parameters, # hyperparameters

scoring=&#39;accuracy&#39;, # metric for scoring

cv=10) # number of folds

clf.fit(X\_train\_scaled,y\_train)

Out[15]:

GridSearchCV(cv=10, estimator=LogisticRegression(),

param\_grid={&#39;C&#39;: array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03]),

&#39;penalty&#39;: [&#39;l1&#39;, &#39;l2&#39;],

&#39;solver&#39;: [&#39;newton-cg&#39;, &#39;lbfgs&#39;, &#39;liblinear&#39;]},

scoring=&#39;accuracy&#39;)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust

the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with

nbviewer.org.

In [16]:

clf.best\_params\_

Out[16]:

{&#39;C&#39;: 1.0, &#39;penalty&#39;: &#39;l2&#39;, &#39;solver&#39;: &#39;liblinear&#39;}

In [17]:

clf.best\_score\_

Out[17]:

0.763793103448276

let&#39;s see how well our model performs on the test data set.

In [18]:

y\_pred = clf.predict(X\_test\_scaled)

accuracy = accuracy\_score(y\_test,y\_pred) accuracy

In [19]:

conf\_mat = confusion\_matrix(y\_test,y\_pred)

conf\_mat

Out[19]:

array([[117, 13],

[ 26, 36]])

In [20]:

true\_positive = conf\_mat[0][0]

false\_positive = conf\_mat[0][1]

false\_negative = conf\_mat[1][0]

true\_negative = conf\_mat[1][1]

In [21]:

Accuracy = (true\_positive + true\_negative) / (true\_positive +false\_positive

+ false\_negative + true\_negative)

Accuracy

Out[21]:

0.796875

In [22]:

Precision = true\_positive/(true\_positive+false\_positive)

Precision

Out[22]:

0.9

In [23]:

Recall = true\_positive/(true\_positive+false\_negative)

Recall

Out[23]:

0.8181818181818182

In [24]:

F1\_Score = 2\*(Recall \* Precision) / (Recall + Precision)

F1\_Score

Out[24]:

0.8571428571428572

Ln[25]:

import pickle

file = open(&#39;modelForPrediction.pkl&pickle.dump(log\_reg,file)

file.close()#39;,&#39;wb&#39;)