

SMRUTI DAWALE
B21BB007
LAB 4

Que 1)

- A) I have done the steps of preprocessing i.e removing outliers , standard scaling and checking and removing null values.
Then i trained the model using linear regression.

The screenshot shows a Google Colab notebook titled "Untitled10.ipynb". The code cell contains Python code for removing outliers using the Interquartile Range (IQR) method and visualizing the data. The output of the code shows two plots: a density plot of 'Pregnancies' and a boxplot of 'Pregnancies'. The density plot shows a sharp peak at 0 and a long tail extending to approximately 15. The boxplot shows the median at 5, with whiskers extending from 0 to about 10.

```
#Removing the Outliers using IQR
colum = np.array(data.columns)
new_data = pd.DataFrame(columns = data.columns)

for i in colum:
    percentile25 = data[i].quantile(0.25)
    percentile75 = data[i].quantile(0.75)
    iqr = percentile75 - percentile25
    upperlimit = percentile75 + 1.5*iqr
    lowerlimit = percentile25 - 1.5*iqr

    new_data[i] = data[(data[i] < upperlimit) & (data[i] > lowerlimit)][i]

column1 = np.array(new_data.columns)

for j in column1:
    plt.figure()
    plt.subplot(2,2,1)
    sns.distplot(new_data[j])

    plt.subplot(2,2,2)
    sns.boxplot(new_data[j])
```

The screenshot shows a Google Colab notebook titled "Untitled10.ipynb". The code cell contains Python code for cleaning the dataset by dropping rows with missing values. The output of the code shows a table of the cleaned dataset with 764 rows and 8 columns. The columns are labeled: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age, and Outcome.

```
df_data.columns = [ "Pregnancies" , "Glucose" , "BloodPressure" , "SkinThickness" , "Insulin" , "BMI" , "DiabetesPedigreeFunction" , "Age" , ]
df_data["Outcome"] = new_data["Outcome"]
df_data = df_data.dropna(axis=0)
df_data
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.675509	0.866981	-0.018981	0.930948	-0.785317	0.219044	0.789739	1.553513	1.0
1	-0.850478	-1.199743	-0.540537	0.549059	-0.785317	-0.859222	-0.316713	-0.157439	0.0
2	1.285904	2.015161	-0.714388	-1.296737	-0.785317	-1.367547	0.970139	-0.067388	1.0
3	-0.850478	-1.068523	-0.540537	0.167170	0.403198	-0.628165	-1.054347	-1.057940	0.0
5	0.370312	-0.182784	0.154871	-1.296737	-0.785317	-1.013260	-0.918045	-0.247489	0.0
...
759	1.896299	-0.674861	0.328723	1.758374	1.490562	0.111217	-1.038311	2.724165	1.0
760	-0.545281	0.014047	-0.192833	0.421763	-0.785317	0.711965	-0.360810	-0.517639	0.0
761	0.370312	-0.018758	-0.018981	0.167170	0.630786	-0.920837	-0.741654	-0.247489	1.0

Untitled10.ipynb

```
# normalizing feature @ 1) Standard Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X = new_data.iloc[:,8:]
scaler.fit(X)
datascaled = scaler.transform(X)

df_data = pd.DataFrame(datascaled)
df_data
```

	0	1	2	3	4	5	6	7
0	0.675509	0.866981	-0.018981	0.930948	-0.785317	0.219044	0.789739	1.553513
1	-0.850478	-1.199743	-0.540537	0.549059	-0.785317	-0.859222	-0.316713	-0.157439
2	1.285904	2.015161	-0.714388	-1.298737	-0.785317	-1.387547	0.970139	-0.067388
3	-0.850478	-1.068523	-0.540537	0.167170	0.403198	-0.628165	-1.054347	-1.057940
4	-1.155676	0.506124	-2.800610	0.930948	1.338837	1.682404	NaN	0.022662
...
759	1.896299	-0.674861	0.328723	1.758374	1.490562	0.111217	-1.038311	2.724165
760	-0.545281	0.014047	-0.192833	0.421763	-0.785317	0.711965	-0.360810	-0.517639
761	0.370312	-0.018758	-0.018981	0.167170	0.630786	-0.920837	-0.741654	-0.247489
762	-0.850478	0.145268	-1.062092	-1.296737	-0.785317	-0.320089	-0.324730	1.283363
763	-0.850478	-0.937302	-0.192833	0.676355	-0.785317	-0.273877	-0.461032	-0.877839

764 rows × 8 columns

```
df_data.columns = [ "Pregnancies" , "Glucose" , "BloodPressure" , "SkinThickness" , "Insulin" , "BMI" , "DiabetesPedigreeFunction" , "Age" , ]
```

Untitled10.ipynb

```
[ ] 763 -0.850478 -0.937302 -0.192833 0.676355 -0.785317 -0.273877 -0.461032 -0.877839 0.0
636 rows × 9 columns

[x]
[ ] x= np.array(df_data["Glucose"]).reshape(-1 ,1)
y= np.array(df_data["Age"]).reshape(-1 ,1)

from sklearn.model_selection import train_test_split

x_train ,x_test , y_train , y_test = train_test_split( x , y , test_size = 0.3 )

[ ] #performing linear regression
from sklearn.linear_model import LinearRegression

lr = LinearRegression()

lr.fit(x_train , y_train)

LinearRegression()

[ ] #normalizing feature @ 2) method Minmax scaling
from sklearn.preprocessing import MinMaxScaler

[ ] norm = MinMaxScaler().fit(x_train)
X_train_norm = norm.transform(x_train)
X_test_norm = norm.transform(x_test)

[ ] X_train_norm

[ ] from sklearn.metrics import mean_squared_error, r2_score
```

B) I have made a class for this question named LineR() , where I have made functions for training and predicting the x_train .

Rms error using code : RMSE is 0.9500411120764171

Rms error using sklearn: RMSE is 0.9932804774182604

Que 1 part B) implementing linear Regression using numpy from scratch

```
class LineR():
    def __init__(self):
        self.m = None
        self.b = None

    def fit(self , x_train , y_train) :
        num =0
        den =0

        for i in range(len(x_train)):
            num = num + ((x_train[i] - x_train.mean())*(y_train[i] - y_train.mean()))
            den = den + ((x_train[i] - x_train.mean())*(x_train[i] - x_train.mean()))

        self.m = num/den
        self.b = y_train.mean() - (self.m * x_train.mean())
        print(self.m)
        print(self.b)

    def predict(self,x_test):
        print(x_test)

        return self.m * x_test + self.b
```

```
[15]
print("R2 score is ", r2)
Performance on Test Set -->
RMSE is  0.9727685629907123
R2 score is  0.05978149857810888

[36]
x= np.array(df_data["Glucose"]).reshape(-1 ,1)
y= np.array(df_data["Age"]).reshape(-1 ,1)

from sklearn.model_selection import train_test_split

x_train ,x_test , y_train , y_test  = train_test_split( x , y , test_size = 0.3 )

y_pred = lr.predict(x_test)
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
r2 = r2_score(y_test, y_pred)

print("Performance on Test Set -->")
print('RMSE is ', rmse)
print('R2 score is ', r2)
```

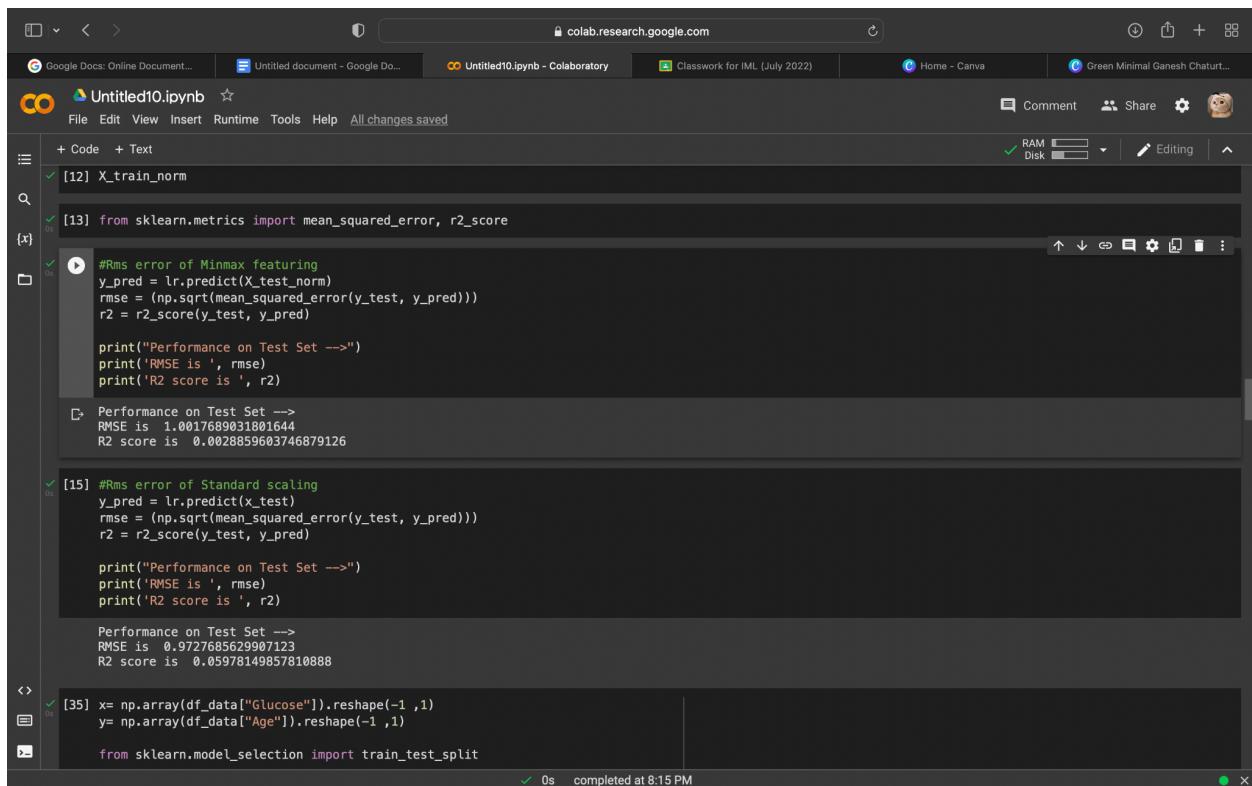
Que 2)

A) I have used Standard scaling and Minmax Scaling

The rms values using

Standard scaling - RMSE is 0.9727685629907123

Minmax Scaling - RMSE is 1.0017689031801644



The screenshot shows a Google Colab notebook titled "Untitled10.ipynb". The code cell [13] contains the following Python code:

```
[13] X_train_norm
      from sklearn.metrics import mean_squared_error, r2_score
      #Rms error of Minmax featuring
      y_pred = lr.predict(X_test_norm)
      rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
      r2 = r2_score(y_test, y_pred)

      print("Performance on Test Set -->")
      print('RMSE is ', rmse)
      print('R2 score is ', r2)

  Performance on Test Set -->
  RMSE is 1.0017689031801644
  R2 score is 0.0028859603746879126

[15] #Rms error of Standard scaling
      y_pred = lr.predict(x_test)
      rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
      r2 = r2_score(y_test, y_pred)

      print("Performance on Test Set -->")
      print('RMSE is ', rmse)
      print('R2 score is ', r2)

  Performance on Test Set -->
  RMSE is 0.9727685629907123
  R2 score is 0.05978149857810888

[35] x= np.array(df_data["Glucose"]).reshape(-1 ,1)
      y= np.array(df_data["Age"]).reshape(-1 ,1)
      from sklearn.model_selection import train_test_split
```

The output of the code shows the RMSE and R2 score for both standard and minmax scaling on the test set. The notebook also includes code for feature selection and splitting the data.

B) Ratio

test = 0.3 , train = 0.7 RMSE is 0.9932804774182604

test = 0.2 , train = 0.8 RMSE is 0.9755082114907181

Test = 0.5 , tain = 0.5 RMSE is 0.957391806335918

The screenshot shows a Google Colab notebook titled "Untitled10.ipynb". The code cell [36] contains Python code for linear regression. It imports numpy and sklearn, reshapes the data, splits it into training and testing sets, performs predictions, calculates RMSE and R2 score, and prints the results. The output shows two runs with different test set sizes (0.3 and 0.5) and their corresponding performance metrics.

```
x= np.array(df_data["Glucose"]).reshape(-1 ,1)
y= np.array(df_data["Age"]).reshape(-1 ,1)

from sklearn.model_selection import train_test_split
x_train ,x_test , y_train , y_test = train_test_split( x , y , test_size = 0.3 )

y_pred = lr.predict(x_test)
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
r2 = r2_score(y_test, y_pred)

print("Performance on Test Set -->")
print('RMSE is ', rmse)
print('R2 score is ', r2)

[36] x= np.array(df_data["Glucose"]).reshape(-1 ,1)
y= np.array(df_data["Age"]).reshape(-1 ,1)

from sklearn.model_selection import train_test_split

x_train ,x_test , y_train , y_test = train_test_split( x , y , test_size = 0.5 )

y_pred = lr.predict(x_test)
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
r2 = r2_score(y_test, y_pred)

print("Performance on Test Set -->")
print('RMSE is ', rmse)
print('R2 score is ', r2)

Performance on Test Set -->
RMSE is  0.9932804774182604
R2 score is  0.01353490963539894
```

The screenshot shows a Google Colab notebook titled "Untitled10.ipynb". The code cell [36] contains Python code for linear regression, similar to the one above, but with a different test size (0.2). The output shows the performance metrics. The code cell [19] contains code to fit the model and plot a scatter plot of Glucose vs Age. The output shows the scatter plot with a red regression line.

```
print("Performance on Test Set -->")
print('RMSE is ', rmse)
print('R2 score is ', r2)

[36] x= np.array(df_data["Glucose"]).reshape(-1 ,1)
y= np.array(df_data["Age"]).reshape(-1 ,1)

from sklearn.model_selection import train_test_split

x_train ,x_test , y_train , y_test = train_test_split( x , y , test_size = 0.2 )

y_pred = lr.predict(x_test)
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
r2 = r2_score(y_test, y_pred)

print("Performance on Test Set -->")
print('RMSE is ', rmse)
print('R2 score is ', r2)

Performance on Test Set -->
RMSE is  0.957391806335918
R2 score is  0.03339093417596761
```

```
[19] lr.fit(x_test , y_test)
plt.scatter(df_data["Glucose"] , df_data["Age"])

plt.xlabel("Glucose")
plt.ylabel("Age")
plt.plot(x_test , lr.predict(x_test) , color = "red")
```

C) Graphs

Untitled10.ipynb

```
plt.scatter(df_data["Glucose"] , df_data["Age"])
plt.xlabel("Glucose")
plt.ylabel("Age")
plt.plot(x_train , lr.predict(x_train) , color = "red")
```

A scatter plot showing the relationship between Glucose levels (x-axis, ranging from -2 to 2) and Age (y-axis, ranging from -1.0 to 3.0). The data points are blue dots, and a red line represents the linear regression fit.

```
x= np.array(df_data["BMI"]).reshape(-1 ,1)
y= np.array(df_data["Age"]).reshape(-1 ,1)

from sklearn.model_selection import train_test_split

x_train_ , x_test_ , y_train_ , y_test_ = train_test_split( x , y , test_size = 0.3 )
```

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

Untitled10.ipynb

```
print("R2 score is ", r2)
```

```
Performance on Test Set -->
RMSE is  0.9755082114907181
R2 score is  0.04398761461956713
```

```
lr.fit(x_test , y_test)
plt.scatter(df_data["Glucose"] , df_data["Age"])

plt.xlabel("Glucose")
plt.ylabel("Age")
plt.plot(x_test , lr.predict(x_test) , color = "red")
```

A scatter plot showing the relationship between Glucose levels (x-axis, ranging from -2 to 2) and Age (y-axis, ranging from -1.0 to 3.0). The data points are blue dots, and a red line represents the linear regression fit. The plot title indicates it is for the test set, with RMSE and R2 scores displayed above the plot.

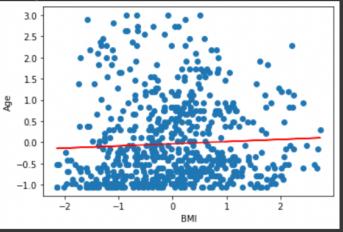
```
plt.scatter(df_data["Glucose"] , df_data["Age"])
plt.xlabel("Glucose")
plt.ylabel("Age")
plt.plot(x_train , lr.predict(x_train) , color = "red")
```

Untitled10.ipynb

```
[ ] lr.fit(x_train_ , y_train_)

LinearRegression()

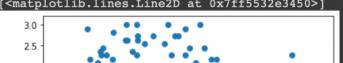
[ ] plt.scatter(df_data["BMI"] , df_data["Age"])
plt.xlabel("BMI")
plt.ylabel("Age")
plt.plot(x_train_ , lr.predict(x_train_ ) , color = "red")

[<matplotlib.lines.Line2D at 0x7ff5532f7510>]


```

```
[ ] lr.fit(x_test_ , y_test_)

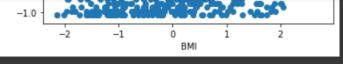
plt.scatter(df_data["BMI"] , df_data["Age"])
plt.xlabel("BMI")
plt.ylabel("Age")
plt.plot(x_test_ , lr.predict(x_test_ ) , color = "red")

[<matplotlib.lines.Line2D at 0x7ff5532e3450>]


```

0s completed at 8:15 PM

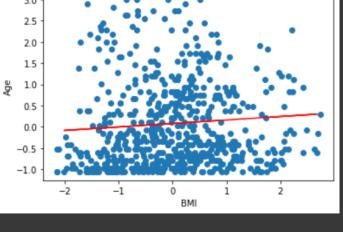
Untitled10.ipynb

```
[ ] -1.0


```

```
[ ] lr.fit(x_test_ , y_test_)

plt.scatter(df_data["BMI"] , df_data["Age"])
plt.xlabel("BMI")
plt.ylabel("Age")
plt.plot(x_test_ , lr.predict(x_test_ ) , color = "red")

[<matplotlib.lines.Line2D at 0x7ff5532e3450>]


```

Que 1 part B) implementing linear Regression using numpy from scratch

```
[ ] class LineR():
    def __init__(self):
        self.m = None
        self.b = None

    def fit(self , x_train , y_train) :
```

0s completed at 8:15 PM

