Self Project

Title: Anomaly Detection on Diabetes dataset.

Introduction:

AutoEncoders are used to learn the compressed representation of raw data. Auto-encoders can be used in computer vision, data-compression, image processing, and time-series forecasting.

AutoEncoders is comprised of three main parts

1. Encoder - It compresses the input into a latent space representation.
2. Bottleneck Layer - An encoder is mapping from input space into lower dimension latent space.
3. Decoder - decodes the encoded image back to the original image of the same dimension.

I have used AutoEncoders for anomaly detection in a diabetes dataset in this project. In a dataset of 253680 samples, Diabetes cases will be treated as anomalies.

Dataset used:

Dataset is available on kaggle(link).

No. of samples - 253680

No. of features - 22

Diabetes_012 feature is the target feature.
Diabetes_012 had three classes.

0 - No Diabetes, 1 - PreDiabetes, 2 - Diabetes

```
df.Diabetes_012.value_counts()
```

```
0.0    213703
2.0     35346
1.0      4631
Name: Diabetes_012, dtype: int64
```

Since the project is about anomaly detection, I made the data suitable for binary classification.
Therefore we are converting all the values labeled as 2 in the Diabetes_012 feature to 1.

```
df['Diabetes_012'].values[df['Diabetes_012'].values == 2] = 1
```

```
df.Diabetes_012.value_counts()
```

```
0.0    213703
1.0     39977
Name: Diabetes_012, dtype: int64
```

Exploratory Data Analysis:

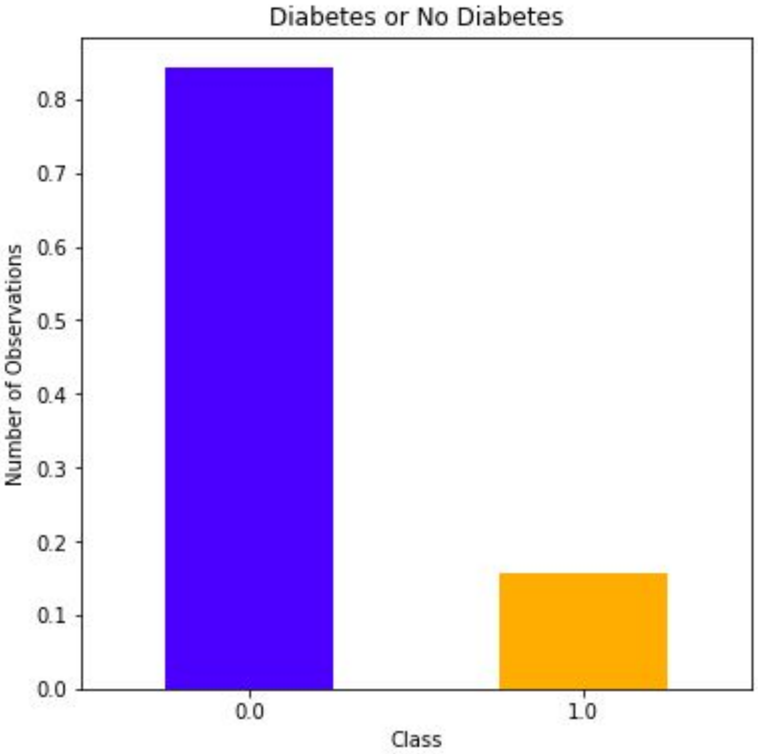
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253680 entries, 0 to 253679
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   Diabetes_012          253680 non-null  float64
 1   HighBP                253680 non-null  float64
 2   HighChol              253680 non-null  float64
 3   CholCheck             253680 non-null  float64
 4   BMI                   253680 non-null  float64
 5   Smoker                253680 non-null  float64
 6   Stroke                253680 non-null  float64
 7   HeartDiseaseorAttack  253680 non-null  float64
 8   PhysActivity          253680 non-null  float64
 9   Fruits                253680 non-null  float64
 10  Veggies               253680 non-null  float64
 11  HvyAlcoholConsump     253680 non-null  float64
 12  AnyHealthcare         253680 non-null  float64
 13  NoDocbcCost           253680 non-null  float64
 14  GenHlth               253680 non-null  float64
 15  MentHlth              253680 non-null  float64
 16  PhysHlth              253680 non-null  float64
 17  DiffWalk              253680 non-null  float64
 18  Sex                   253680 non-null  float64
 19  Age                   253680 non-null  float64
 20  Education             253680 non-null  float64
 21  Income                253680 non-null  float64
dtypes: float64(22)
memory usage: 42.6 MB
```

```
df.isnull().sum().sum()
```

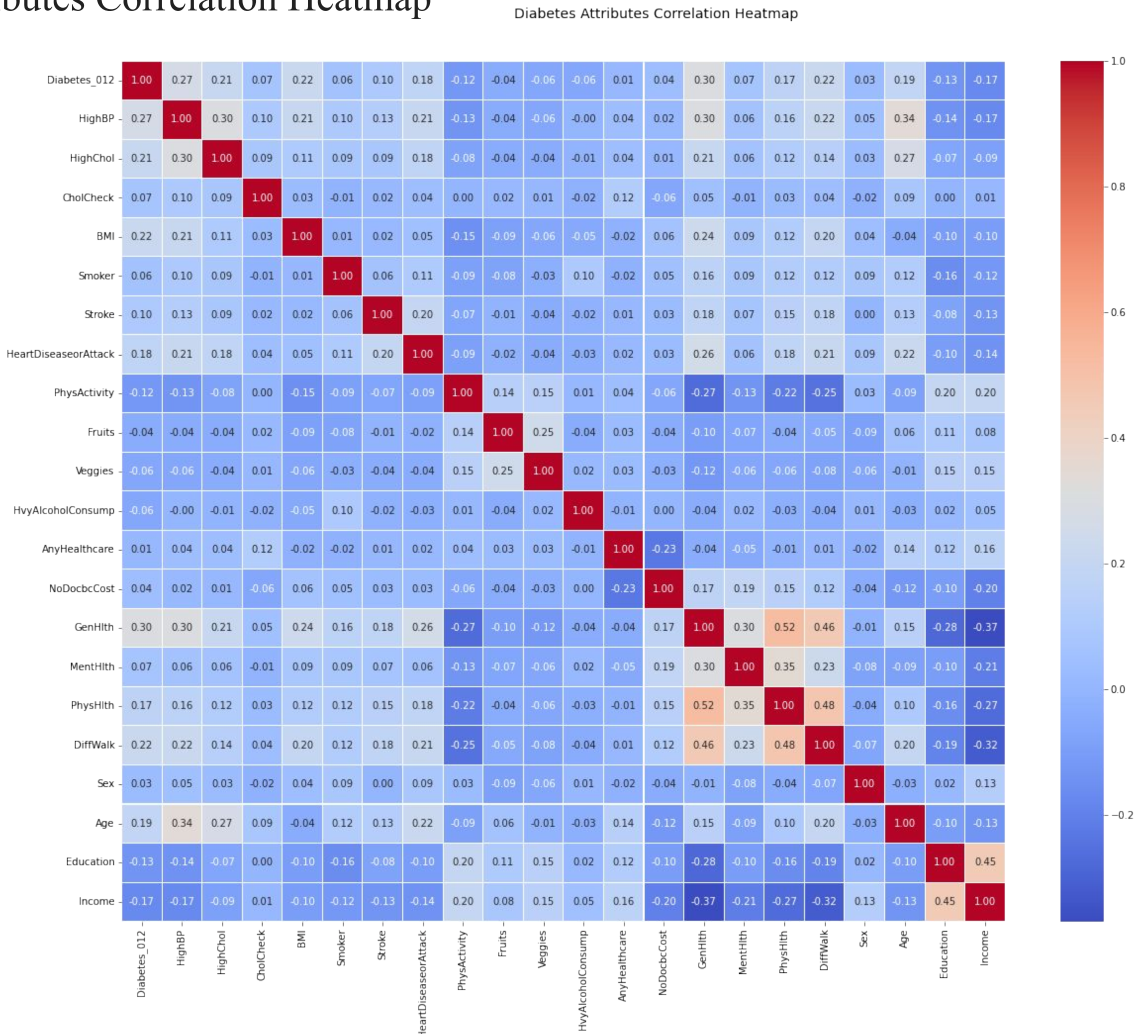
```
0
```

Dataset Visualization:



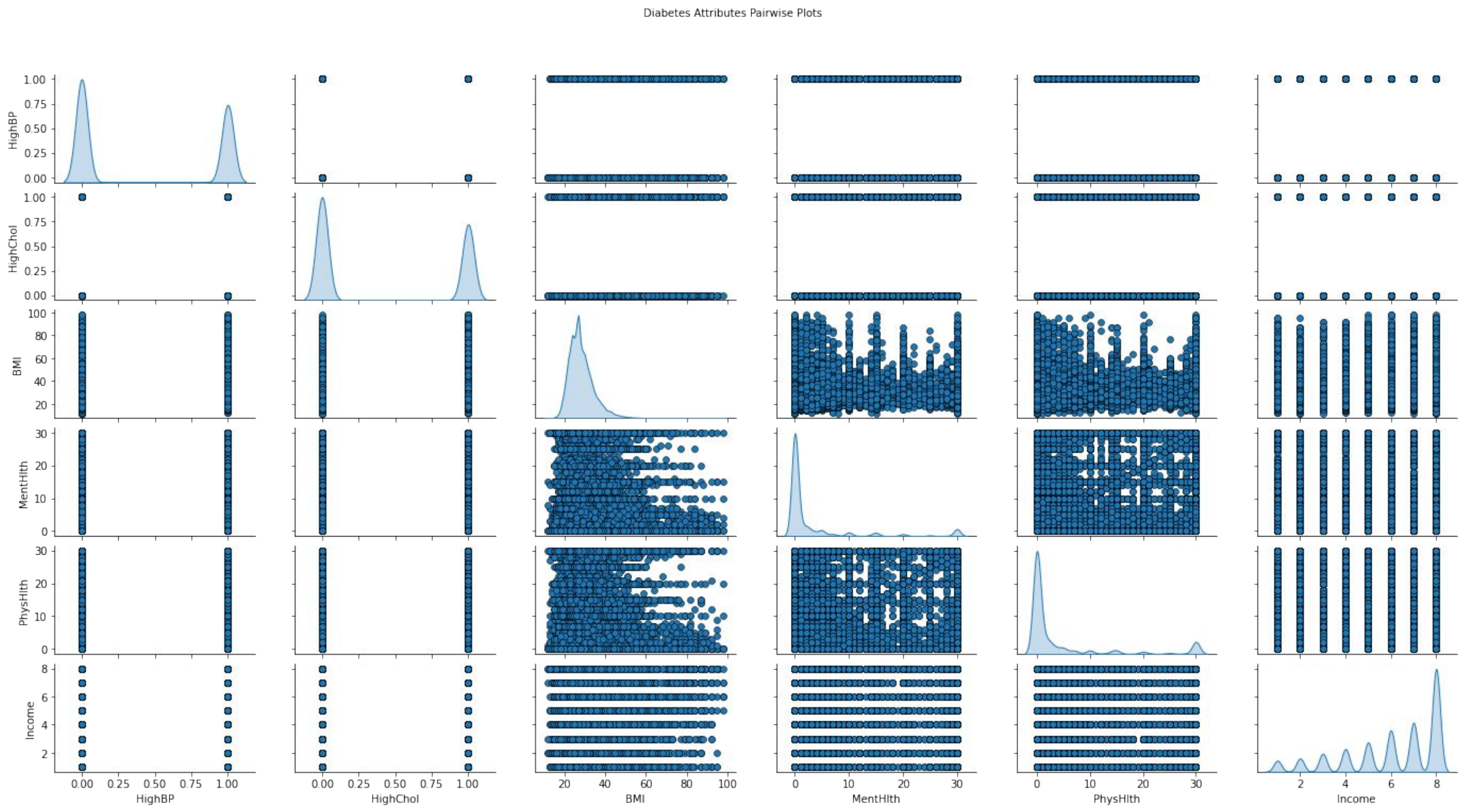
Class imbalance is suitable for outlier detection using autoencoders.

Diabetes Attributes Correlation Heatmap

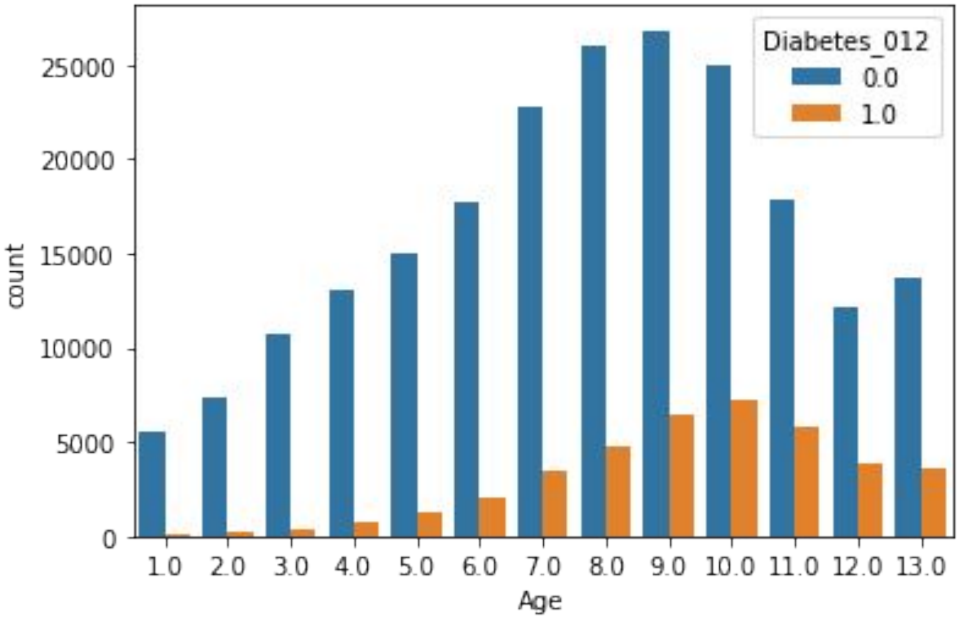


Diabetes Attributes Pairwise Plots

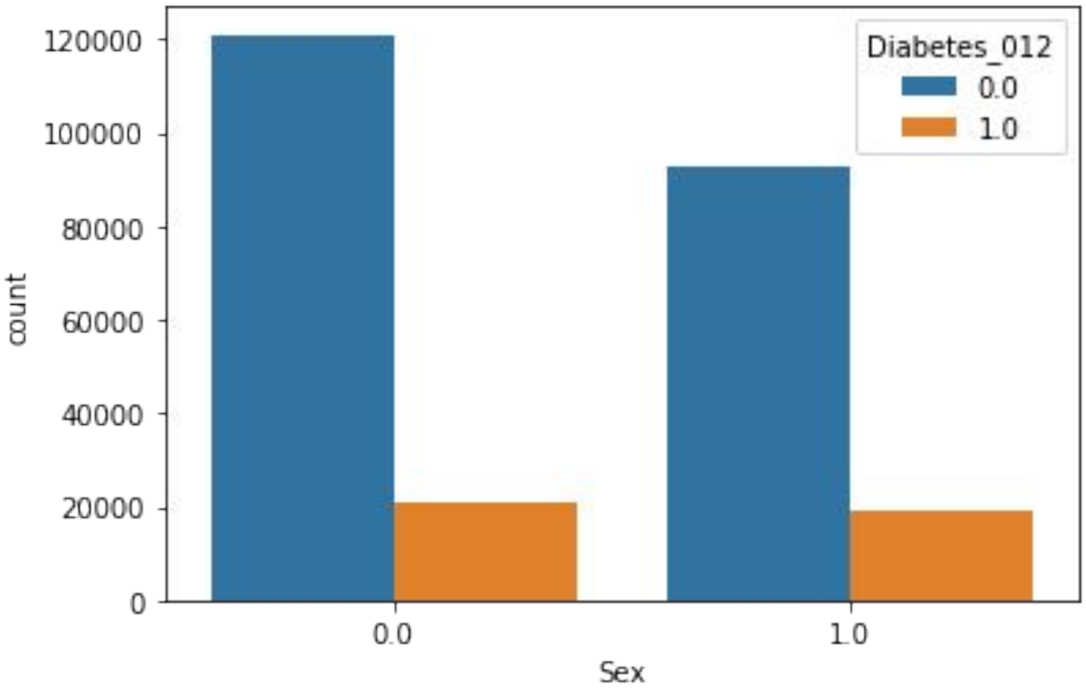
Included attributes - HighBP, HighChol, BMI, MentHlth, PhysHlth, Income



Plot of count of no. of samples against age(both classes)



Plot of count of no. of samples against gender(both classes)



Final data preparation -

1. Normalizing data

```
from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler()

df['BMI'] = sc.fit_transform(df['BMI'].values.reshape(-1, 1))
df['MentHlth'] = sc.fit_transform(df['MentHlth'].values.reshape(-1, 1))
df['PhysHlth'] = sc.fit_transform(df['PhysHlth'].values.reshape(-1, 1))
df['DiffWalk'] = sc.fit_transform(df['DiffWalk'].values.reshape(-1, 1))
df['Sex'] = sc.fit_transform(df['Sex'].values.reshape(-1, 1))
df['Age'] = sc.fit_transform(df['Age'].values.reshape(-1, 1))
df['Education'] = sc.fit_transform(df['Education'].values.reshape(-1, 1))
df['Income'] = sc.fit_transform(df['Income'].values.reshape(-1, 1))
```

2. Splitting data in 80:20 for training and testing.

Final data

```
labels_train = labels_train.astype(bool)
labels_test = labels_test.astype(bool)

#creating NoDiabetes and Diabetes datasets
NoDiabetes_data_train = data_train[~labels_train]
NoDiabetes_data_test = data_test[~labels_test]
Diabetes_data_train = data_train[labels_train]
Diabetes_data_test = data_test[labels_test]

print(" No. of records in Diabetes Train Data =",len(Diabetes_data_train))
print(" No. of records in No Diabetes Train data =",len(NoDiabetes_data_train))
print(" No. of records in Diabetes Test Data =",len(Diabetes_data_test))
print(" No. of records in No Diabetes Test data =",len(NoDiabetes_data_test))
```

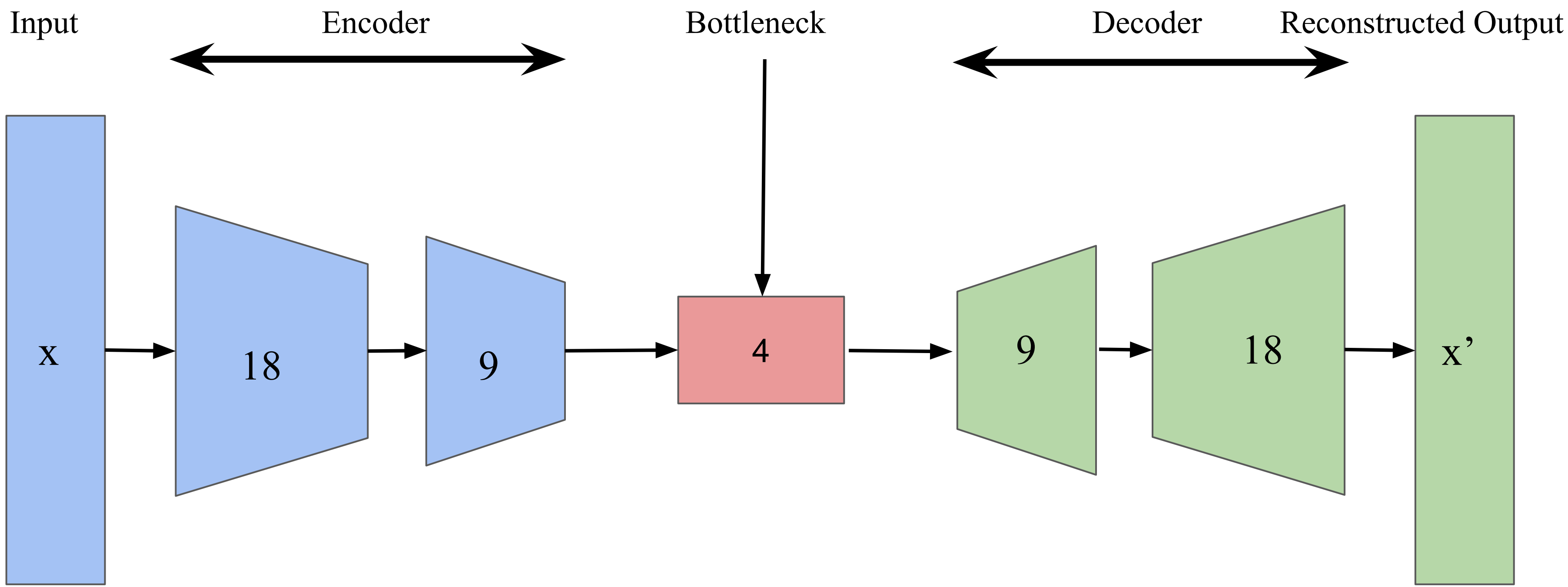
```
No. of records in Diabetes Train Data = 32100
No. of records in No Diabetes Train data = 170844
No. of records in Diabetes Test Data = 7877
No. of records in No Diabetes Test data = 42859
```

Building Model -

1. Setting parameters

Setting training parameter.

```
nb_epoch = 100
batch_size = 64
input_dim = NoDiabetes_data_train.shape[1]
encoding_dim = 18
hidden_dim_1 = int(encoding_dim / 2)
hidden_dim_2 = 4
learning_rate = 1e-7
```



Input
Encoder
Bottleneck
Decoder
Reconstructed Output
x
18
9
4
9
18
x’

2. Model Architecture

```
Model: "model_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 21)]              0

 dense_38 (Dense)            (None, 18)                396

 dropout_12 (Dropout)        (None, 18)                0

 dense_39 (Dense)            (None, 9)                 171

 dense_40 (Dense)            (None, 4)                 40

 dense_41 (Dense)            (None, 9)                 45

 dropout_13 (Dropout)        (None, 9)                 0

 dense_42 (Dense)            (None, 18)                180

 dense_43 (Dense)            (None, 21)                399

=================================================================
Total params: 1,231
Trainable params: 1,231
Non-trainable params: 0
_________________________________________________________________
```

3. Optimizer, Loss, etc.

```
cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_diabetes.h5", mode='min', monitor='val_loss', verbose=2, save_best_only=True

# define our early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.0001,
    patience=10,
    verbose=1,
    mode='min',
    restore_best_weights=True)
```

```
autoencoder.compile(metrics=['accuracy'], loss='mean_squared_error', optimizer='adam')
```

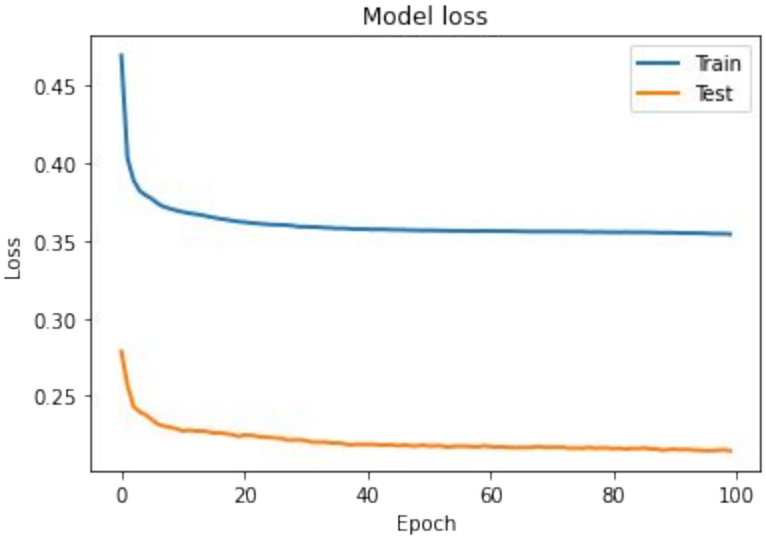
```
history = autoencoder.fit(Diabetes_data_train, Diabetes_data_train,
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    validation_data=(data_test, data_test),
                    verbose=1,
                    callbacks=[cp, early_stop]
                    ).history
```

Results -

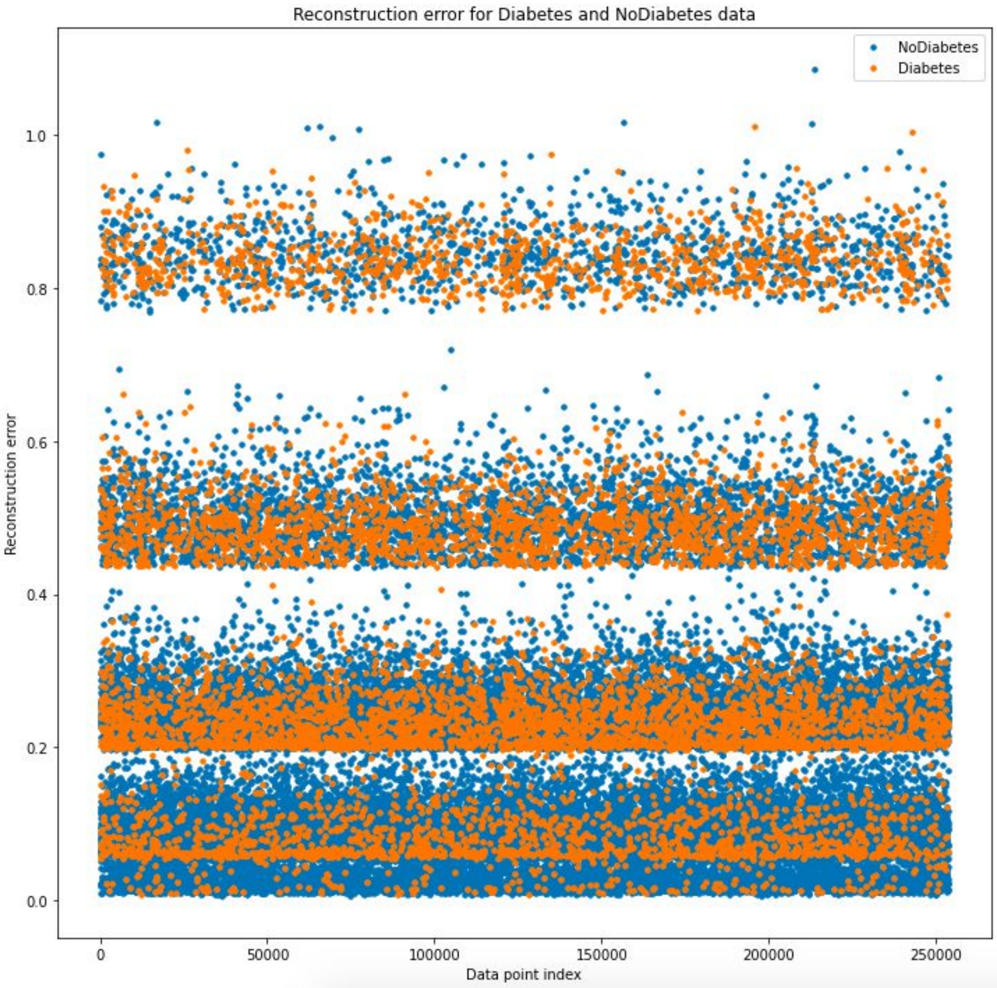
1. Results of training after 100 epochs.

```
Epoch 100/100
485/502 [===========================>..] - ETA: 0s - loss: 0.3544 - accuracy: 0.6083
Epoch 100: val_loss did not improve from 0.21462
502/502 [==============================] - 2s 5ms/step - loss: 0.3544 - accuracy: 0.6088 - val_loss: 0.2147 - val_accuracy: 0.5706
```

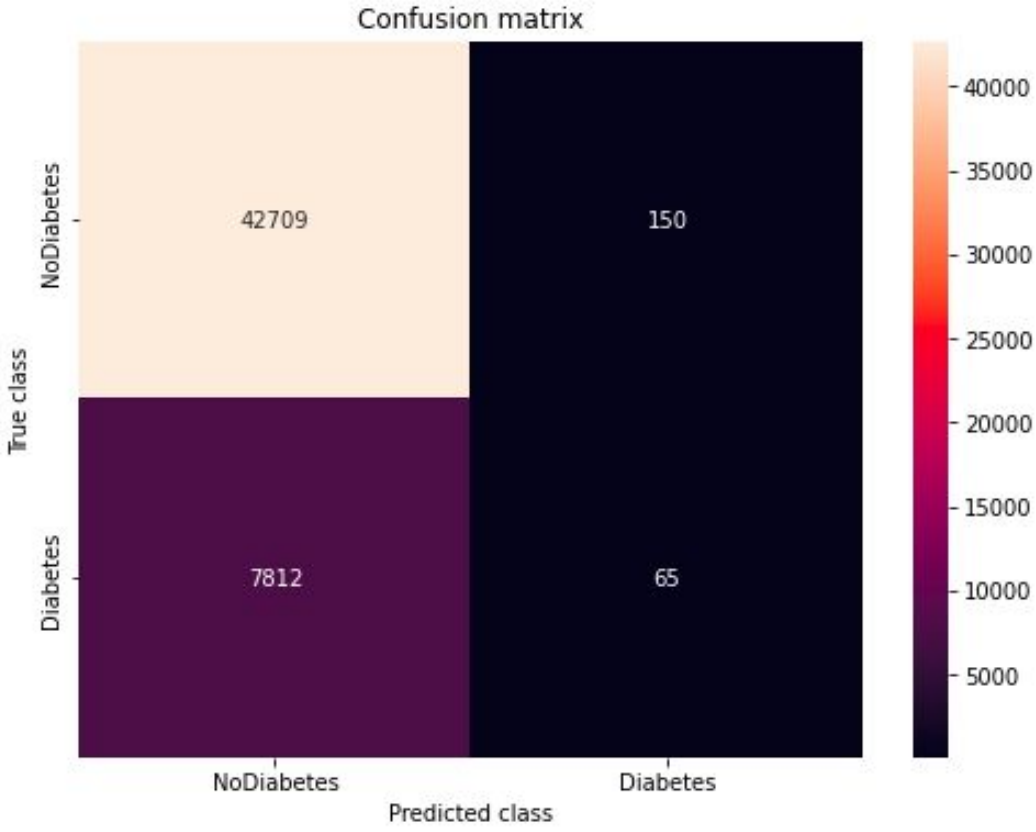
2. Loss in training and testing of model



3. Plot of reconstruction error against data point index for both classes.



4. Plot of confusion matrix and results of accuracy, precision and recall.




```
Accuracy:  0.843070009460738
Recall:  0.008251872540307223
Precision:  0.3023255813953488
```