```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Importing the dataset

```
from sklearn.datasets?
```

```
# importing datset
df = pd.read_csv('diabetes.csv')
df.head()
# null entries count

df.isnull().sum()
df.info()
df.duplicated()
df.duplicated().sum()
```

Extracting feature and target values

```
# list of features

features = [x for x in df.columns if x not in df.columns[-1]]
features
```

```
target = df.columns[-1]
target
# Extracting features

df_features = df.drop('Outcome', axis = 1)
df_features.head()
# Converting dataframe to numpy array

x = np.array(df_features)   # feature array
x[:5,:]


y = np.array(df['Outcome'])
y = np.expand_dims(y,axis=1) # reshaping to 2d array
y[:5]
```

Feature vs Target Plot

```
fig, ax = plt.subplots(4,2, figsize=(16,18))
ax = ax.flatten()
for i in range(len(features)):

    ax[i].scatter([x_val for x_val, y_val in zip(x[:,i], y) if y_val == 1],
            [y_val for y_val in y if y_val == 1],
            marker='o', color='blue', label='Diabetic')

    ax[i].scatter([x_val for x_val, y_val in zip(x[:,i], y) if y_val == 0],
            [y_val for y_val in y if y_val == 0],
            marker='x', color='red',  label='Non-Diabetic')

    ax[i].set_xlabel(features[i])
```

```
    ax[i].set_ylabel(target)
    ax[i].legend()

fig.suptitle('Feature vs target plot', fontsize=25)
```

Splitting dataset into train, validation and test set

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


x_train, x_temp, y_train, y_temp = train_test_split(x,y, test_size=0.4,
random_state = 1)
x_cv, x_test, y_cv, y_test = train_test_split(x_temp, y_temp,
test_size=0.4, random_state = 1)


scaler = StandardScaler()
x_train_norm = scaler.fit_transform(x_train)
x_cv_norm = scaler.transform(x_cv)
x_test_norm = scaler.transform(x_test)
joblib.dump(scaler, 'scaler.joblib')


import tensorflow as tf  # importing TensorFlow module
from tensorflow import keras  # importing Keras module from TensorFlow
from tensorflow.keras.layers import Dense  # importing Dense layer class
from Keras
from tensorflow.keras.models import Sequential  # importing Sequential
model class from Keras
from tensorflow.keras.regularizers import l2  # importing L2 regularization
from Keras regularizers
```

In [35]:

```
# function to build models with different number of layers and units

def build_models():
    model_1 = Sequential([
      Dense(units=20, activation='relu', kernel_regularizer = l2(0.01),
name='L1'), # using Ridge regularization
      Dense(units=10, activation='relu', kernel_regularizer = l2(0.01),
name='L2'),
      Dense(units=1, activation='linear', kernel_regularizer = l2(0.01),
name='L3')
    ], name = 'model_1')

    model_2 = Sequential([
      Dense(units=25, activation='relu', kernel_regularizer = l2(0.01),
name='L1'),
      Dense(units=15, activation='relu', kernel_regularizer = l2(0.01),
name='L2'),
      Dense(units=1, activation='linear', kernel_regularizer = l2(0.01),
name='L3')
    ], name = 'model_2')
```

```
    model_3 = Sequential([
        Dense(units=20, activation='relu', kernel_regularizer = l2(0.01),
name='L1'),
        Dense(units=12, activation='relu', kernel_regularizer = l2(0.01),
name='L2'),
        Dense(units=12, activation='relu', kernel_regularizer = l2(0.01),
name='L3'),
        Dense(units=1, activation='linear', kernel_regularizer = l2(0.01),
name='L4')
    ], name = 'model_3')

    model_4 = Sequential([
        Dense(units=32, activation='relu', kernel_regularizer = l2(0.01),
name='L1'),
        Dense(units=16, activation='relu', kernel_regularizer = l2(0.01),
name='L2'),
        Dense(units=8, activation='relu', kernel_regularizer = l2(0.01),
name='L3'),
        Dense(units=4, activation='relu', kernel_regularizer = l2(0.01),
name='L4'),
        Dense(units=12, activation='relu', kernel_regularizer = l2(0.01),
name='L5'),
        Dense(units=1, activation='linear', kernel_regularizer = l2(0.01),
name='L6')
    ], name = 'model_4')

    model_list=[model_1, model_2, model_3, model_4]

    return model_list


# building different models
models = build_models()

# Training error of each model
train_error_list= []

# Validation error of each model
val_error_list= []


# training each model
for model in models:
    model.compile(
        loss = tf.keras.losses.BinaryCrossentropy(from_logits=True),
       # sigmoid activation funciton is internally applied while
calculating BinaryCrossentopy loss
       # so that the roundoff error are minimized
        optimizer = tf.keras.optimizers.Adam(learning_rate = 0.01 )
    )

    print(f'training... {model.name}')
    model.fit(
        x_train, y_train,
        epochs=200,
```

```
        verbose=0  # not displaying training progress
    )

    # Computing training error
    yhat = model.predict(x_train)  # yhat is a linear function z = w.x + b
    yhat_sigmoid = tf.nn.sigmoid(yhat) # g(z) = 1/(1+e^(-z))

    threshold = 0.5

    yhat = np.where(yhat_sigmoid >= threshold, 1, 0)
    train_error = np.mean(yhat != y_train)
    train_error_list.append(train_error)

    # Computing validation error
    yhat = model.predict(x_cv)
    # getting probability values since the output layer is 'linear'
    yhat_sigmoid = tf.nn.sigmoid(yhat)

    yhat = np.where(yhat_sigmoid >= threshold, 1, 0)
    val_error = np.mean(yhat != y_cv)
    val_error_list.append(val_error)


print('Displaying Training  and Validation error for each models\n')
for i,j in enumerate(models):
    print(f'Training error for {j.name} = {train_error_list[i]}')
    print(f'Validation error for {j.name} = {val_error_list[i]}\n')

model_index = np.argmin(val_error_list)
models[model_index].name

# displaying best model

best_model = models[model_index]
best_model.summary()
```

Test Error OR Generalization

```
from sklearn.metrics import accuracy_score

yhat = best_model.predict(x_test)
yhat_sigmoid = tf.nn.sigmoid(yhat)
yhat = np.where(yhat_sigmoid >= threshold, 1, 0)
test_error = yhat != y_test
test_error = np.mean(test_error)
print(f'Test error = {test_error}')

accuracy_score(yhat, y_test)

tf.keras.models.save_model(best_model,'model.h5')

nn_model = tf.keras.models.load_model('model.h5')
saved_scaler = joblib.load('scaler.joblib')
```

```
print('Enter the following data:\n')
x_input = []
for i in range(len(features)):
    print(features[i])
    x_i = float(input())
    x_input.append(x_i)

x_input = np.array(x_input).reshape(1, x_train.shape[1])
x_input_norm = saved_scaler.transform(x_input)
pred = nn_model.predict(x_input_norm)
pred = tf.nn.sigmoid(pred)



if pred >= threshold:
    print('Sorry! You have a higher risk of developing diabetes based on
your medical data.')
else:
    print('Congratulations! You have a low risk of developing diabetes
based on your medical data.')
```