In [21]:

```
import pandas as pd
```

In [22]:

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

In [23]:

```
print(df.head())
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
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   

   DiabetesPedigreeFunction  Age  Outcome  
0                     0.627   50        1  
1                     0.351   31        0  
2                     0.672   32        1  
3                     0.167   21        0  
4                     2.288   33        1  
```

In [24]:

```
#1 Finding the range of values
print(df.describe())
```

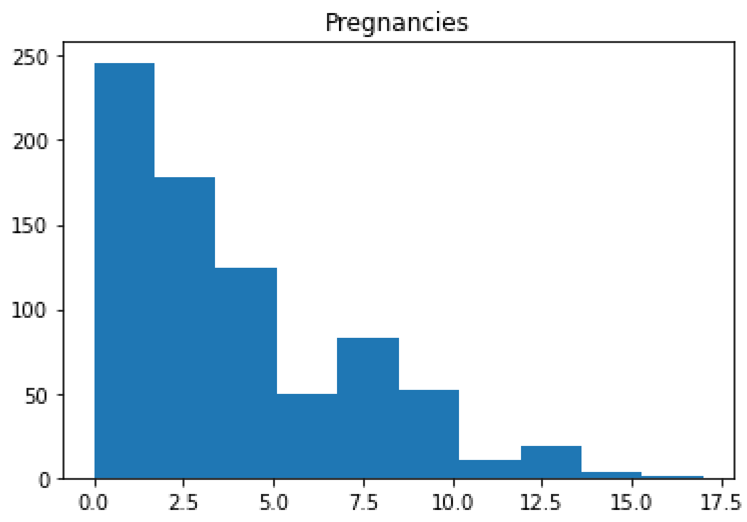
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  
\
count   768.000000  768.000000     768.000000     768.000000  768.000000   
mean      3.845052  120.894531      69.105469      20.536458   79.799479   
std       3.369578   31.972618      19.355807      15.952218  115.244002   
min       0.000000    0.000000       0.000000       0.000000    0.000000   
25%       1.000000   99.000000      62.000000       0.000000    0.000000   
50%       3.000000  117.000000      72.000000      23.000000   30.500000   
75%       6.000000  140.250000      80.000000      32.000000  127.250000   
max      17.000000  199.000000     122.000000      99.000000  846.000000   

              BMI  DiabetesPedigreeFunction         Age     Outcome  
count  768.000000                768.000000  768.000000  768.000000  
mean    31.992578                  0.471876   33.240885    0.348958  
std      7.884160                  0.331329   11.760232    0.476951  
min      0.000000                  0.078000   21.000000    0.000000  
25%     27.300000                  0.243750   24.000000    0.000000  
50%     32.000000                  0.372500   29.000000    0.000000  
75%     36.600000                  0.626250   41.000000    1.000000  
max     67.100000                  2.420000   81.000000    1.000000  
```

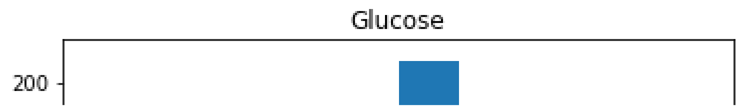
In [7]:

```
import matplotlib.pyplot as plt
```

In [10]:

```
#plot histogram
for col in df.columns:
    if df[col].dtype == 'float64' or df[col].dtype == 'int64':
        plt.hist(df[col], bins=10)
        plt.title(col)
        plt.show()
```





In [17]:

```
from sklearn.preprocessing import MinMaxScaler
# Load the diabetes dataset
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target
feature_names = diabetes.feature_names
dataset = pd.DataFrame(X, columns=feature_names)

# Normalize the features using min-max scaling
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Print the first 5 rows of the normalized features
print(X_scaled[:5, :])
```

```
---------------------------------------------------------------------------
--
NameError                                 Traceback (most recent call las
t)
Input In [17], in <cell line: 3>()
      1 from sklearn.preprocessing import MinMaxScaler
      2 # Load the diabetes dataset
----> 3 diabetes = load_diabetes()
      4 X, y = diabetes.data, diabetes.target
      5 feature_names = diabetes.feature_names

NameError: name 'load_diabetes' is not defined
```

In [18]:

```
from sklearn.model_selection import train_test_split

# Split the normalized data into training and testing subsets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_s

# Print the shapes of the training and testing subsets
print("Training subset shape:", X_train.shape, y_train.shape)
print("Testing subset shape:", X_test.shape, y_test.shape)
```

```
---------------------------------------------------------------------------
--
NameError                                 Traceback (most recent call las
t)
Input In [18], in <cell line: 4>()
      1 from sklearn.model_selection import train_test_split
      3 # Split the normalized data into training and testing subsets
----> 4 X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
      6 # Print the shapes of the training and testing subsets
      7 print("Training subset shape:", X_train.shape, y_train.shape)

NameError: name 'X_scaled' is not defined
```

In [26]:

```
# 2 Normalizing the dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes
names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', '
df = pd.read_csv(url, names=names)

# Split the dataset into features and label
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# Normalize the features using MinMaxScaler
scaler = MinMaxScaler()
X_norm = scaler.fit_transform(X)

# Print the first 5 rows of normalized features
print(X_norm[:5])
```

```
[[0.35294118 0.74371859 0.59016393 0.35353535 0.         0.50074516
  0.23441503 0.48333333]
 [0.05882353 0.42713568 0.54098361 0.29292929 0.         0.39642325
  0.11656704 0.16666667]
 [0.47058824 0.91959799 0.52459016 0.         0.         0.34724292
  0.25362938 0.18333333]
 [0.05882353 0.44723618 0.54098361 0.23232323 0.11111111 0.41877794
  0.03800171 0.        ]
 [0.         0.68844221 0.32786885 0.35353535 0.19858156 0.64232489
  0.94363792 0.2       ]]
```

In [27]:

```
# Spliting the data into training and testing subsets
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=42)
print("X_train shape: ", X_train.shape)
print("y_train shape: ", y_train.shape)
print("X_test shape: ", X_test.shape)
print("y_test shape: ", y_test.shape)
```

```
X_train shape:  (614, 8)
y_train shape:  (614,)
X_test shape:  (154, 8)
y_test shape:  (154,)
```

In [36]:

```
from sklearn.decomposition import PCA
# Apply PCA to reduce the dimensionality
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Print the explained variance ratio of the first two principal components
print("Explained variance ratio:", pca.explained_variance_ratio_)

# Print the first 5 rows of the reduced dataset
print(X_pca[:5])
```

```
Explained variance ratio: [0.88854663 0.06159078]
[[-75.71465491 -35.95078264]
 [-82.3582676   28.90821322]
 [-74.63064344 -67.90649647]
 [ 11.07742273  34.89848586]
 [ 89.74378806  -2.74693708]]
```

In [45]:

```
from keras.models import Sequential
from keras.layers import Dense

# Define the model architecture
# This model contains 16 neurons with 1 hidden layer
# The output layer has 1 neuron
model = Sequential()
model.add(Dense(16, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
X_train_pca = X_train.iloc[:, :2]

# Train the model
# Training the model using fit method
# validation set to monitor the performance of the model
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test)
```

```
Epoch 1/100
20/20 [==============================] - 0s 8ms/step - loss: 7.4521 -
accuracy: 0.4202 - val_loss: 6.0188 - val_accuracy: 0.4351
Epoch 2/100
20/20 [==============================] - 0s 3ms/step - loss: 4.2080 -
accuracy: 0.5423 - val_loss: 3.2250 - val_accuracy: 0.4610
Epoch 3/100
20/20 [==============================] - 0s 2ms/step - loss: 2.7712 -
accuracy: 0.5456 - val_loss: 2.5086 - val_accuracy: 0.5325
Epoch 4/100
20/20 [==============================] - 0s 3ms/step - loss: 2.3498 -
accuracy: 0.5586 - val_loss: 2.1931 - val_accuracy: 0.5455
Epoch 5/100
20/20 [==============================] - 0s 2ms/step - loss: 2.0341 -
accuracy: 0.5570 - val_loss: 1.9826 - val_accuracy: 0.5584
Epoch 6/100
20/20 [==============================] - 0s 3ms/step - loss: 1.7961 -
accuracy: 0.5814 - val_loss: 1.7918 - val_accuracy: 0.5714
Epoch 7/100
```

IMPLEMENTING THE SAME CODE WITH DIFFERENT NO OF NEURONS

In [46]:

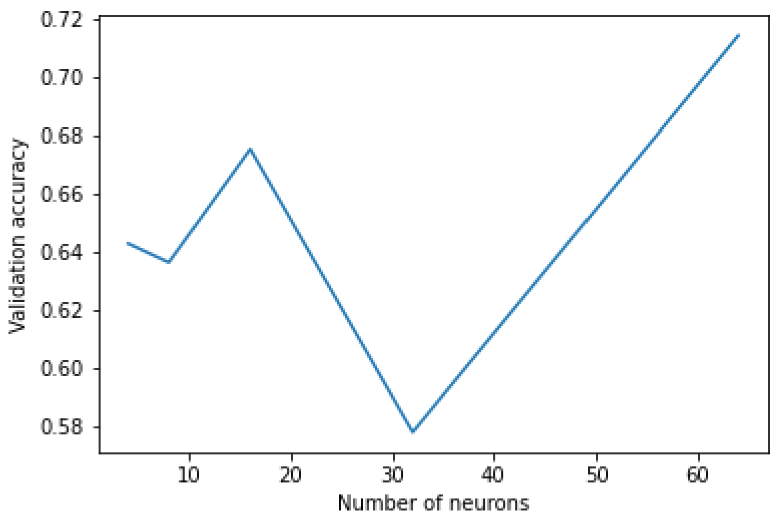
```python
# By increasing the no of neurons the model will be able to identify complex patterns.

from keras.models import Sequential
from keras.layers import Dense

# Define the model architecture
# This model contains 16 neurons with 1 hidden layer
# The output layer has 1 neuron
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
X_train_pca = X_train.iloc[:, :2]

# Train the model
# Training the model using fit method
# validation set to monitor the performance of the model
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test))
```

```
Epoch 1/100
20/20 [==============================] - 1s 10ms/step - loss: 8.4687 -
accuracy: 0.3730 - val_loss: 6.3064 - val_accuracy: 0.3506
Epoch 2/100
20/20 [==============================] - 0s 3ms/step - loss: 3.6688 -
accuracy: 0.4609 - val_loss: 2.2537 - val_accuracy: 0.4545
Epoch 3/100
20/20 [==============================] - 0s 3ms/step - loss: 1.6702 -
accuracy: 0.5049 - val_loss: 1.2872 - val_accuracy: 0.6429
Epoch 4/100
20/20 [==============================] - 0s 3ms/step - loss: 1.3683 -
accuracy: 0.5603 - val_loss: 1.2114 - val_accuracy: 0.6299
Epoch 5/100
20/20 [==============================] - 0s 3ms/step - loss: 1.2386 -
accuracy: 0.5407 - val_loss: 1.1242 - val_accuracy: 0.6104
Epoch 6/100
20/20 [==============================] - 0s 3ms/step - loss: 1.1236 -
accuracy: 0.5440 - val_loss: 1.0364 - val_accuracy: 0.6039
Epoch 7/100
```

ACCURACY WITH DIFFERENT NUMBER OF NEURONS

In [49]:

```
neurons = [4, 8, 16, 32, 64]
accuracies = []

for n in neurons:
    # create model with n neurons in hidden layer
    model = Sequential()
    model.add(Dense(n, activation='relu', input_dim=X_train.shape[1]))
    model.add(Dense(1, activation='sigmoid'))

    # compile and fit model
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, b

    # record validation accuracy
    accuracies.append(history.history['val_accuracy'][-1])

# plot accuracy vs number of neurons
import matplotlib.pyplot as plt

plt.plot(neurons, accuracies)
plt.xlabel('Number of neurons')
plt.ylabel('Validation accuracy')
plt.show()
```



CONFUSION MATRIX AND CLASSIFICATION REPORT

In [54]:

```
from sklearn.metrics import confusion_matrix
y_pred = model.predict(X_test)
y_pred_binary = [1 if p >= 0.5 else 0 for p in y_pred]
cm = confusion_matrix(y_test, y_pred)
cr = classification_report(y_test, y_pred)

print('Confusion matrix:\n', cm)
print('Classification report:\n', cr)
```

```
5/5 [==============================] - 0s 1ms/step

------------------------------------------------------------------------
--
ValueError                                Traceback (most recent call las
t)
Input In [54], in <cell line: 4>()
      2 y_pred = model.predict(X_test)
      3 y_pred_binary = [1 if p >= 0.5 else 0 for p in y_pred]
----> 4 cm = confusion_matrix(y_test, y_pred)
      5 cr = classification_report(y_test, y_pred)
      7 print('Confusion matrix:\n', cm)

File ~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:30
7, in confusion_matrix(y_true, y_pred, labels, sample_weight, normalize)
    222 def confusion_matrix(
    223     y_true, y_pred, *, labels=None, sample_weight=None, normalize
=None
    224 ):
    225     """Compute confusion matrix to evaluate the accuracy of a cla
ssification.
    226
    227     By definition a confusion matrix :math:`C` is such that :mat
h:`C_{i, j}`
   (...)
    305     (0, 2, 1, 1)
    306     """
--> 307     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    308     if y_type not in ("binary", "multiclass"):
    309         raise ValueError("%s is not supported" % y_type)

File ~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:93,
in _check_targets(y_true, y_pred)
     90     y_type = {"multiclass"}
     92 if len(y_type) > 1:
---> 93     raise ValueError(
     94         "Classification metrics can't handle a mix of {0} and {1}
targets".format(
     95             type_true, type_pred
     96         )
     97     )
     99 # We can't have more than one value on y_type => The set is no mo
re needed
    100 y_type = y_type.pop()

ValueError: Classification metrics can't handle a mix of binary and conti
nuous targets
```

MLP WITH 2 HIDDEN LAYERS

In [55]:

```
from sklearn.neural_network import MLPClassifier

# create MLP model with two hidden layers
model = MLPClassifier(hidden_layer_sizes=(64, 32))

# fit the model to the training data
model.fit(X_train, y_train)
```

Out[55]:

```
MLPClassifier(hidden_layer_sizes=(64, 32))
```

In [56]:

```
y_pred = model.predict(X_test)
```

In [57]:

```
print(y_pred)
```

```
[1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 1 1 0 1
0
 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0
0
 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0
0
 0 0 1 0 1 1 0 0 1 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0
0
 0 0 0 0 1 0]
```

MLP WITH 3 HIDDEN LAYERS

In [61]:

```
from sklearn.neural_network import MLPClassifier

# define the model with three hidden layers
model = MLPClassifier(hidden_layer_sizes=(20, 10, 5))

# train the model on the training data
model.fit(X_train, y_train)
```

Out[61]:

```
MLPClassifier(hidden_layer_sizes=(20, 10, 5))
```

In [62]:

```
y_pred = model.predict(X_test)
```

In [63]:

```
print(y_pred)
```

```
[1 0 0 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 1
1
 0 1 1 0 0 0 1 0 1 1 0 0 1 0 1 1 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 1 0 0
0
 0 0 0 1 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 0 1 1 0 0
0
 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 1 0 0 0 1 1 0 0 0
0
 0 1 0 0 1 0]
```

USING LOGISTIC REGRESSION FOR THIS MODEL

In [68]:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

# train the model on the training data
model.fit(X_train, y_train)

# make predictions on the test data
y_pred = model.predict(X_test)
```

```
C:\Users\pavan\anaconda3\lib\site-packages\sklearn\linear_model\_logisti
c.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://s
cikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression (https://scikit-learn.org/stable/modules/linear_model.html#logis
tic-regression)
  n_iter_i = _check_optimize_result(
```

In [1]:

```
print(y_pred = model.predict(X_test))
```

```
-------------------------------------------------------------------------
--
NameError                                 Traceback (most recent call las
t)
Input In [1], in <cell line: 1>()
----> 1 print(y_pred = model.predict(X_test))

NameError: name 'model' is not defined
```

In [ ]:

```

```