ICP 5 REPORT

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
[ ] path_to_csv = '/content/gdrive/My Drive/diabetes.csv'
```

```
[ ] import keras
    import pandas as pd
    import numpy as np
    from keras.models import Sequential
    from keras.layers import Dense
    from sklearn.model_selection import train_test_split

    # Load dataset
    dataset = pd.read_csv(path_to_csv, header=None).values

    # Split the dataset into training and testing sets
    X_train, X_test, Y_train, Y_test = train_test_split(dataset[:, 0:8], dataset[:, 8], test_size=0.25, random_state=87)

    # Set random seed for reproducibility
    np.random.seed(155)

    # Create a Sequential model
    model = Sequential()

    # Add Dense layers with 'relu' activation for hidden layers
    model.add(Dense(20, input_dim=8, activation='relu'))  # First hidden layer
    model.add(Dense(15, activation='relu'))  # Second hidden layer
    model.add(Dense(10, activation='relu'))  # Third hidden layer
```

```python
# Add output layer with 'sigmoid' activation
model.add(Dense(1, activation='sigmoid'))

# Compile the model using binary crossentropy and adam optimizer
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])

# Train the model
model_fitted = model.fit(X_train, Y_train, epochs=100, initial_epoch=0)

# Print model summary and evaluate accuracy on the test set
print(model.summary())
print(model.evaluate(X_test,Y_test))
```

```
0s 3ms/step - acc: 0.7320 - loss: 0.5313
0s 3ms/step - acc: 0.7630 - loss: 0.5106
0s 3ms/step - acc: 0.7514 - loss: 0.5113
0s 3ms/step - acc: 0.7598 - loss: 0.4951
0s 3ms/step - acc: 0.7385 - loss: 0.5289
0s 4ms/step - acc: 0.7865 - loss: 0.4708
0s 2ms/step - acc: 0.7925 - loss: 0.5023
0s 3ms/step - acc: 0.7651 - loss: 0.5076
0s 3ms/step - acc: 0.7805 - loss: 0.4785
0s 4ms/step - acc: 0.7545 - loss: 0.5287
```

```
0s 3ms/step - acc: 0.7514 - loss: 0.5113
0s 3ms/step - acc: 0.7598 - loss: 0.4951
0s 3ms/step - acc: 0.7385 - loss: 0.5289
0s 4ms/step - acc: 0.7865 - loss: 0.4708
0s 2ms/step - acc: 0.7925 - loss: 0.5023
0s 3ms/step - acc: 0.7651 - loss: 0.5076
0s 3ms/step - acc: 0.7805 - loss: 0.4785
0s 4ms/step - acc: 0.7545 - loss: 0.5287
0s 2ms/step - acc: 0.7672 - loss: 0.4902
0s 3ms/step - acc: 0.7560 - loss: 0.4822
0s 5ms/step - acc: 0.7590 - loss: 0.5205
0s 3ms/step - acc: 0.7772 - loss: 0.4867
0s 4ms/step - acc: 0.7625 - loss: 0.5112
0s 3ms/step - acc: 0.7729 - loss: 0.4904
0s 2ms/step - acc: 0.7186 - loss: 0.5284
0s 2ms/step - acc: 0.7579 - loss: 0.5068
0s 2ms/step - acc: 0.7777 - loss: 0.4927
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 20)	180
dense_1 (Dense)	(None, 15)	315
dense_2 (Dense)	(None, 10)	160
dense_3 (Dense)	(None, 1)	11

```
Total params: 2,000 (7.82 KB)
Trainable params: 666 (2.60 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 1,334 (5.21 KB)
None
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 0.6987 - loss: 0.5817
[0.6005145907402039, 0.6875]
```

```
[ ] import keras
    import pandas as pd
    import numpy as np
    from keras.models import Sequential
    from keras.layers import Dense
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler

    # Load dataset
    dataset = pd.read_csv(path_to_csv, header=None).values

    # Split the dataset into features (X) and target (Y)
    X = dataset[:, 0:8]
    Y = dataset[:, 8]

    # Normalize the feature data
    sc = StandardScaler()
    X = sc.fit_transform(X)

    # Split the dataset into training and testing sets
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=87)

    # Set random seed for reproducibility
    np.random.seed(155)

    # Create a Sequential model
    model = Sequential()

    # Add Dense layers with 'relu' activation for hidden layers
    model.add(Dense(20, input_dim=8, activation='relu'))  # First hidden layer
    model.add(Dense(15, activation='relu'))  # Second hidden layer
    model.add(Dense(10, activation='relu'))  # Third hidden layer

    # Add output layer with 'sigmoid' activation
    model.add(Dense(1, activation='sigmoid'))

    # Compile the model using binary crossentropy and adam optimizer
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])

    # Train the model
    model_fitted = model.fit(X_train, Y_train, epochs=100, initial_epoch=0)

    # Print model summary and evaluate accuracy on the test set
```

```
print(model.summary())
print(model.evaluate(X_test,Y_test))
```

```
l00
──────────────────── 0s 2ms/step - acc: 0.8451 - loss: 0.3483
l00
──────────────────── 0s 2ms/step - acc: 0.8766 - loss: 0.3105
l00
──────────────────── 0s 2ms/step - acc: 0.8507 - loss: 0.3373
l00
──────────────────── 0s 2ms/step - acc: 0.8457 - loss: 0.3413
l00
──────────────────── 0s 2ms/step - acc: 0.8541 - loss: 0.3339
l00
──────────────────── 0s 2ms/step - acc: 0.8452 - loss: 0.3529
l00
──────────────────── 0s 2ms/step - acc: 0.8629 - loss: 0.3129
l00
──────────────────── 0s 2ms/step - acc: 0.8863 - loss: 0.2953
l00
──────────────────── 0s 2ms/step - acc: 0.8625 - loss: 0.3426
l00
──────────────────── 0s 2ms/step - acc: 0.8379 - loss: 0.3341
l00
──────────────────── 0s 2ms/step - acc: 0.8929 - loss: 0.2953
l00
──────────────────── 0s 2ms/step - acc: 0.8572 - loss: 0.3266
l00
──────────────────── 0s 2ms/step - acc: 0.8670 - loss: 0.3124
l00
──────────────────── 0s 3ms/step - acc: 0.8813 - loss: 0.3034
l00
──────────────────── 0s 2ms/step - acc: 0.8788 - loss: 0.3176
l00
```

```
l00
──────────────────── 0s 2ms/step - acc: 0.8670 - loss: 0.3124
l00
──────────────────── 0s 3ms/step - acc: 0.8813 - loss: 0.3034
l00
──────────────────── 0s 2ms/step - acc: 0.8788 - loss: 0.3176
l00
──────────────────── 0s 2ms/step - acc: 0.8844 - loss: 0.2940
l00
──────────────────── 0s 2ms/step - acc: 0.8860 - loss: 0.2951
l00
──────────────────── 0s 2ms/step - acc: 0.8716 - loss: 0.3050
'100
──────────────────── 0s 2ms/step - acc: 0.8935 - loss: 0.2897
equential_1"
```

ype)	Output Shape	Param #
(Dense)	(None, 20)	180
(Dense)	(None, 15)	315
(Dense)	(None, 10)	160
(Dense)	(None, 1)	11

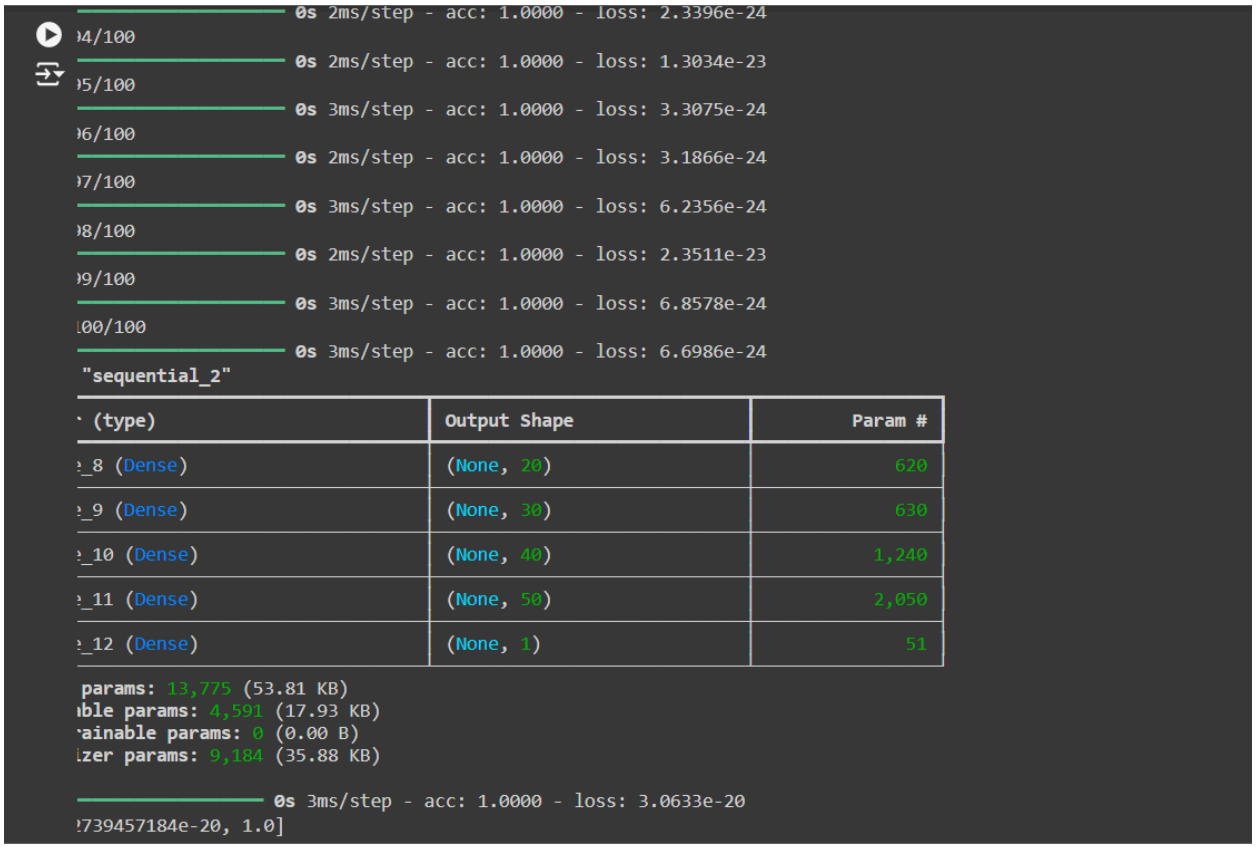
```
ams: 2,000 (7.82 KB)
 params: 666 (2.60 KB)
iable params: 0 (0.00 B)
 params: 1,334 (5.21 KB)

──────────────────── 0s 3ms/step - acc: 0.7612 - loss: 0.5396
)185638428, 0.7708333134651184]
```

```
[ ] path_to_csv1 = '/content/gdrive/My Drive/breastcancer.csv'
```

```
import keras
import pandas
from keras.models import Sequential
from keras.layers import Dense, Activation

# load dataset
from sklearn.model_selection import train_test_split
#from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

dataset = pd.read_csv(path_to_csv1, header=None).values

X = dataset[1:, 2:-1]  # Features
Y = dataset[1:, -1]   # Labels (M or B)

# Convert labels to binary format
Y = np.where(Y == 'M', 1, 0)  # M -> 1, B -> 0

#Convert to numeric
X = X.astype(np.float64) # Convert X to numeric

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                    test_size=0.25, random_state=87)


np.random.seed(155)
my_first_nn = Sequential() # create model
```

```
[ ] my_first_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer
    my_first_nn.add(Dense(30, activation='relu')) # hidden layer
    my_first_nn.add(Dense(40, activation='relu')) # hidden layer
    my_first_nn.add(Dense(50, activation='relu')) # hidden layer

    my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
    my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
    my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,
                                         initial_epoch=0)
    print(my_first_nn.summary())
    print(my_first_nn.evaluate(X_test,Y_test))
```

```
3/100
0s 2ms/step - acc: 1.0000 - loss: 5.5031e-24
4/100
0s 2ms/step - acc: 1.0000 - loss: 5.4153e-24
5/100
0s 2ms/step - acc: 1.0000 - loss: 2.2958e-24
6/100
0s 4ms/step - acc: 1.0000 - loss: 1.9675e-24
7/100
0s 4ms/step - acc: 1.0000 - loss: 7.5888e-24
8/100
0s 2ms/step - acc: 1.0000 - loss: 1.3166e-23
9/100
0s 2ms/step - acc: 1.0000 - loss: 7.6699e-24
0/100
0s 2ms/step - acc: 1.0000 - loss: 2.3326e-23
1/100
0s 3ms/step - acc: 1.0000 - loss: 2.4450e-24
2/100
0s 3ms/step - acc: 1.0000 - loss: 1.6542e-23
3/100
```


0s 2ms/step - acc: 1.0000 - loss: 2.3396e-24
4/100
0s 2ms/step - acc: 1.0000 - loss: 1.3034e-23
5/100
0s 3ms/step - acc: 1.0000 - loss: 3.3075e-24
6/100
0s 2ms/step - acc: 1.0000 - loss: 3.1866e-24
7/100
0s 3ms/step - acc: 1.0000 - loss: 6.2356e-24
8/100
0s 2ms/step - acc: 1.0000 - loss: 2.3511e-23
9/100
0s 3ms/step - acc: 1.0000 - loss: 6.8578e-24
00/100
0s 3ms/step - acc: 1.0000 - loss: 6.6986e-24
"sequential_2"
(type)
Output Shape
Param #
_8 (Dense)
(None, 20)
620
_9 (Dense)
(None, 30)
630
_10 (Dense)
(None, 40)
1,240
_11 (Dense)
(None, 50)
2,050
_12 (Dense)
(None, 1)
51
params: 13,775 (53.81 KB)
ble params: 4,591 (17.93 KB)
ainable params: 0 (0.00 B)
zer params: 9,184 (35.88 KB)
0s 3ms/step - acc: 1.0000 - loss: 3.0633e-20
739457184e-20, 1.0]

```
 Optimizer params: 9,184 (35.88 KB)
None
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 3.0633e-20
[5.8962739457184e-20, 1.0]
```

```python
import keras
import pandas
from keras.models import Sequential
from keras.layers import Dense, Activation

# load dataset
from sklearn.model_selection import train_test_split
#from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

dataset = pd.read_csv(path_to_csv1, header=None).values

X = dataset[1:, 2:-1]  # Features
Y = dataset[1:, -1]   # Labels (M or B)

# Convert labels to binary format
Y = np.where(Y == 'M', 1, 0)  # M -> 1, B -> 0

#Convert to numeric
X = X.astype(np.float64) # Convert X to numeric

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                    test_size=0.25, random_state=87)
```

```python
#Normalizing the data
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


np.random.seed(155)
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer
my_first_nn.add(Dense(30, activation='relu')) # hidden layer
my_first_nn.add(Dense(40, activation='relu')) # hidden layer
my_first_nn.add(Dense(50, activation='relu')) # hidden layer

my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,
                                     initial_epoch=0)
print(my_first_nn.summary())
print(my_first_nn.evaluate(X_test,Y_test))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, pr
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - acc: 0.8176 - loss: 0.5997
Epoch 2/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 0.2816
Epoch 3/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 0.0669
Epoch 4/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - acc: 1.0000 - loss: 0.0122
Epoch 5/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 0.0032
Epoch 6/100
```

```python
#Normalizing the data
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


np.random.seed(155)
my_first_nn = Sequential() # create model
my_first_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer
my_first_nn.add(Dense(30, activation='relu')) # hidden layer
my_first_nn.add(Dense(40, activation='relu')) # hidden layer
my_first_nn.add(Dense(50, activation='relu')) # hidden layer

my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,
                                     initial_epoch=0)
print(my_first_nn.summary())
print(my_first_nn.evaluate(X_test,Y_test))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, pr
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - acc: 0.8176 - loss: 0.5997
Epoch 2/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 0.2816
Epoch 3/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 0.0669
Epoch 4/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - acc: 1.0000 - loss: 0.0122
Epoch 5/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 0.0032
Epoch 6/100
```

```
Epoch 7/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 0.0016
Epoch 8/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - acc: 1.0000 - loss: 0.0012
Epoch 9/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 0.0010
Epoch 10/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 8.7806e-04
Epoch 11/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 6.3084e-04
Epoch 12/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - acc: 1.0000 - loss: 5.6421e-04
Epoch 13/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - acc: 1.0000 - loss: 5.2709e-04
Epoch 14/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - acc: 1.0000 - loss: 3.8428e-04
Epoch 15/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 2.3100e-04
Epoch 16/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 2.2936e-04
Epoch 17/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 2.1114e-04
Epoch 18/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 1.0000 - loss: 1.2757e-04
Epoch 19/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - acc: 1.0000 - loss: 1.5042e-04
Epoch 20/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 1.6276e-04
Epoch 21/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 1.0686e-04
Epoch 22/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 8.6160e-05
Epoch 23/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 8.4084e-05
```

```
Epoch 24/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - acc: 1.0000 - loss: 8.1183e-05
Epoch 25/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 7.0593e-05
Epoch 26/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 8.4707e-05
Epoch 27/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 6.0599e-05
Epoch 28/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 7.4843e-05
Epoch 29/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 1.0000 - loss: 5.2547e-05
Epoch 30/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 3.3969e-05
Epoch 31/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - acc: 1.0000 - loss: 5.7337e-05
Epoch 32/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 4.9170e-05
Epoch 33/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 2.9749e-05
Epoch 34/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 3.5653e-05
Epoch 35/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 3.1122e-05
Epoch 36/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 3.1609e-05
Epoch 37/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 3.5866e-05
Epoch 38/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 4.3771e-05
Epoch 39/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - acc: 1.0000 - loss: 2.8046e-05
Epoch 40/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - acc: 1.0000 - loss: 1.9907e-05
```

```
[ ]  Epoch 41/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 3.5813e-05
     Epoch 42/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - acc: 1.0000 - loss: 2.5293e-05
     Epoch 43/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 1.0000 - loss: 1.9264e-05
     Epoch 44/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 2.1780e-05
     Epoch 45/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - acc: 1.0000 - loss: 1.7480e-05
     Epoch 46/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - acc: 1.0000 - loss: 1.9936e-05
     Epoch 47/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 1.7947e-05
     Epoch 48/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 2.0831e-05
     Epoch 49/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 1.4722e-05
     Epoch 50/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 1.6388e-05
     Epoch 51/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 1.4409e-05
     Epoch 52/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 1.7394e-05
     Epoch 53/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 1.6754e-05
     Epoch 54/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 1.1243e-05
     Epoch 55/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 1.1512e-05
     Epoch 56/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 1.1118e-05
     Epoch 57/100
     14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 1.7493e-05
```

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 1.7493e-05
Epoch 58/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 1.5340e-05
Epoch 59/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 1.2474e-05
Epoch 60/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 1.0299e-05
Epoch 61/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 9.9197e-06
Epoch 62/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 1.3964e-05
Epoch 63/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 1.0329e-05
Epoch 64/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 1.4529e-05
Epoch 65/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - acc: 1.0000 - loss: 9.2856e-06
Epoch 66/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 6.5939e-06
Epoch 67/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 5.6795e-06
Epoch 68/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - acc: 1.0000 - loss: 7.3987e-06
Epoch 69/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 6.1561e-06
Epoch 70/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 7.7660e-06
Epoch 71/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 1.1675e-05
Epoch 72/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 1.2385e-05
Epoch 73/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 9.9271e-06
Epoch 74/100
```

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 9.8653e-06
Epoch 75/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 8.9551e-06
Epoch 76/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 5.6452e-06
Epoch 77/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 5.5384e-06
Epoch 78/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 5.2825e-06
Epoch 79/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 6.7665e-06
Epoch 80/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 5.2460e-06
Epoch 81/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 3.9280e-06
Epoch 82/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 5.6404e-06
Epoch 83/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 7.9074e-06
Epoch 84/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 7.4575e-06
Epoch 85/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 5.2450e-06
Epoch 86/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 5.1166e-06
Epoch 87/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 5.7406e-06
Epoch 88/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 4.1102e-06
Epoch 89/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 6.0938e-06
Epoch 90/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 3.7932e-06
Epoch 91/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 3.4277e-06
```

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 3.3689e-06
Epoch 93/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 4.6541e-06
Epoch 94/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 3.5622e-06
Epoch 95/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 5.1818e-06
Epoch 96/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 3.5201e-06
Epoch 97/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 3.9748e-06
Epoch 98/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 2.9670e-06
Epoch 99/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 2.5104e-06
Epoch 100/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 2.7179e-06
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
dense_13 (Dense)	(None, 20)	620
dense_14 (Dense)	(None, 30)	630
dense_15 (Dense)	(None, 40)	1,240
dense_16 (Dense)	(None, 50)	2,050
dense_17 (Dense)	(None, 1)	51

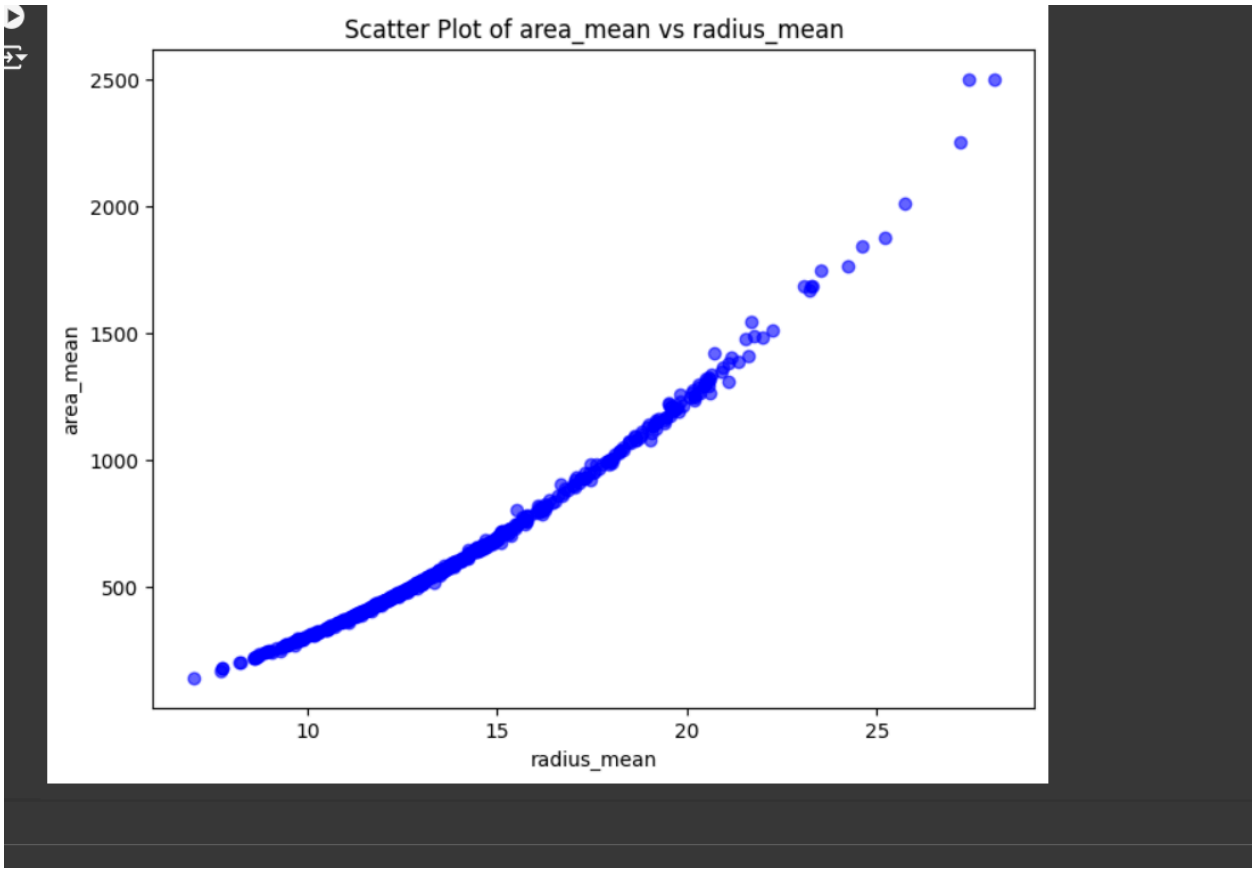
```
 Total params: 13,775 (53.81 KB)
 Trainable params: 4,591 (17.93 KB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 9,184 (35.88 KB)
None
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - acc: 1.0000 - loss: 9.2803e-06
[6.102526185713941e-06, 1.0]
```

```
path_to_csv1 = '/content/gdrive/My Drive/breastcancer.csv'
```

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

# Load the dataset
data = pd.read_csv('/content/gdrive/My Drive/breastcancer.csv')

# Print the column names to help you choose the correct columns
print(data.columns)

# Replace 'X_column' and 'Y_column' with actual numeric column names from your CSV file
x_column = 'radius_mean'  # Replace this with the actual column name for X-axis (e.g., 'Age')
y_column = 'area_mean'  # Replace this with the actual column name for Y-axis (e.g., 'Tumor_Size')

# Create a scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(data[x_column], data[y_column], color='blue', alpha=0.6)

# Add labels and title
plt.xlabel(x_column)
plt.ylabel(y_column)
plt.title(f'Scatter Plot of {y_column} vs {x_column}')

# Show the plot
plt.show()
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```



My Github link : -
https://github.com/Nitish300903/bda.git