lcp5

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

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

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

+ Code + Text

New Section

```
[ ] 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
    import pandas as pd
    import numpy as np

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

    X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
                                                        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=8, activation='relu')) # hidden layer
    my_first_nn.add(Dense(15, activation='relu'))              # Second hidden layer
    my_first_nn.add(Dense(10, activation='relu'))
    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,
```

✓ 0s completed at 10:19 PM

```
    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))
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7668 - loss: 0.5062
Epoch 82/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7582 - loss: 0.4849
Epoch 83/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7476 - loss: 0.4957
Epoch 84/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7841 - loss: 0.5080
Epoch 85/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7146 - loss: 0.5566
Epoch 86/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7566 - loss: 0.4929
Epoch 87/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7369 - loss: 0.5305
Epoch 88/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7692 - loss: 0.4890
Epoch 89/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7652 - loss: 0.4755
Epoch 90/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7180 - loss: 0.5563
Epoch 91/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7642 - loss: 0.4784
Epoch 92/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7659 - loss: 0.4681
Epoch 93/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7600 - loss: 0.4929
Epoch 94/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7713 - loss: 0.4805
Epoch 95/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7663 - loss: 0.4813
Epoch 96/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7611 - loss: 0.4907
Epoch 97/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7532 - loss: 0.4990
Epoch 98/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7533 - loss: 0.4852
```

✓ 0s completed at 10:19 PM

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7642 - loss: 0.4784
Epoch 92/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7659 - loss: 0.4681
Epoch 93/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7600 - loss: 0.4929
Epoch 94/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7713 - loss: 0.4805
Epoch 95/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7663 - loss: 0.4813
Epoch 96/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7611 - loss: 0.4907
Epoch 97/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7532 - loss: 0.4990
Epoch 98/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7533 - loss: 0.4852
Epoch 99/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7606 - loss: 0.4915
Epoch 100/100
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 0.7639 - loss: 0.4719
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 3ms/step - acc: 0.7587 - loss: 0.5433
[0.5704227685928345, 0.734375]
```

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

✓ 0s completed at 10:19 PM

```
[15] 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
```

✓ 0s completed at 10:19 PM

```
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))
```

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 1.2691e-11
Epoch 83/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 1.5175e-11
Epoch 84/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 7.7172e-12
Epoch 85/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 8.5898e-12
Epoch 86/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 1.9796e-11
Epoch 87/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 3.4960e-11
Epoch 88/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 1.3339e-11
Epoch 89/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 1.8535e-11
Epoch 90/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 1.9618e-11
Epoch 91/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 6.4725e-12
Epoch 92/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 3.4795e-11
Epoch 93/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 8.8990e-12
Epoch 94/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 3.3189e-11
Epoch 95/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 2.3061e-11
Epoch 96/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 3.2589e-11
Epoch 97/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 1.8818e-11
Epoch 98/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 8.5919e-12
```

✓ 0s completed at 10:19 PM

[16]

```
Epoch 97/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 1.8818e-11
Epoch 98/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 8.5919e-12
Epoch 99/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 1.4931e-11
Epoch 100/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 1.4124e-11
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 20)	620
dense_6 (Dense)	(None, 30)	630
dense_7 (Dense)	(None, 40)	1,240
dense_8 (Dense)	(None, 50)	2,050
dense_9 (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 3ms/step - acc: 1.0000 - loss: 4.8367e-10
[9.311877380291378e-10, 1.0]
```

```
import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler  # Import StandardScaler
```

✓ 0s completed at 10:19 PM

```
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)

# 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=X_train.shape[1], 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))
```

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 6.2224e-06
Epoch 83/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 6.2256e-06
Epoch 84/100
```

✓ 0s completed at 10:19 PM

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 6.2224e-06
Epoch 83/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 6.2256e-06
Epoch 84/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 6.7300e-06
Epoch 85/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 5.9131e-06
Epoch 86/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 6.9172e-06
Epoch 87/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 6.1454e-06
Epoch 88/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 5.5569e-06
Epoch 89/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 5.6328e-06
Epoch 90/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 3.9726e-06
Epoch 91/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - acc: 1.0000 - loss: 5.6929e-06
Epoch 92/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 4.1221e-06
Epoch 93/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 5.0889e-06
Epoch 94/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 4.5934e-06
Epoch 95/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 5.9000e-06
Epoch 96/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 4.1617e-06
Epoch 97/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 4.4051e-06
Epoch 98/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 3.9066e-06
Epoch 99/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 3.2541e-06
Epoch 100/100
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - acc: 1.0000 - loss: 2.8692e-06
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #

✓ 0s completed at 10:19 PM

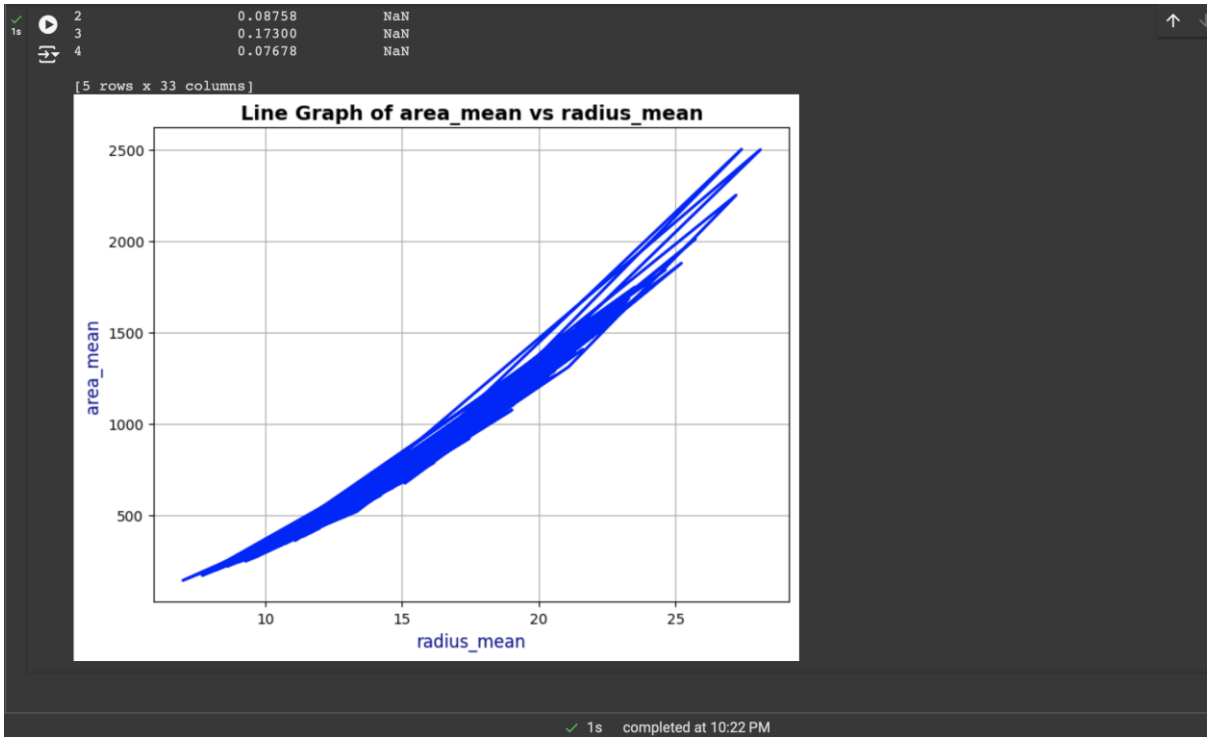
```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset (assuming the file is correctly located in Google Drive)
data = pd.read_csv('/content/gdrive/My Drive/breastcancer.csv')

# Print column names and first few rows to help with understanding the dataset structure
print("Columns in the dataset:", data.columns)
print(data.head())

# Assign actual numeric columns for X and Y axes
x_column = 'radius_mean'  # Replace with your actual X-axis column name
y_column = 'area_mean'    # Replace with your actual Y-axis column name

# Check if the columns exist in the data before proceeding
try:
    # Create the scatter plot with customized visuals
    plt.figure(figsize=(8, 6))
    plt.scatter(data[x_column], data[y_column], color='green', alpha=0.7, edgecolors='black')

    # Add labels and title with enhanced formatting
    plt.xlabel(x_column, fontsize=12, color='darkred')
    plt.ylabel(y_column, fontsize=12, color='darkred')
    plt.title(f'{y_column} vs {x_column}', fontsize=14, fontweight='bold')

    # Show grid and plot
    plt.grid(True)
    plt.show()

except KeyError as e:
    print(f"Error: Column {e} not found in the dataset.")
```

```
Columns in the dataset: 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',
```

✓ 0s completed at 10:19 PM

```
Columns in the dataset: 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')
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  ...          17.33           184.60      2019.0            0.1622
1  ...          23.41           158.80      1956.0            0.1238
2  ...          25.53           152.50      1709.0            0.1444
3  ...          26.50            98.87       567.7            0.2098
4  ...          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
```

✓ 0s completed at 10:19 PM



My GitHub link:https://github.com/nithin1086/BDA