About Dataset:

Context: This dataset is created for prediction of Graduate Admissions from an Indian perspective.

Content: The dataset contains several parameters which are considered important during the application for Masters Programs.

The parameters included are :

1. GRE Scores (out of 340)
2. TOEFL Scores (out of 120)
3. University Rating (out of 5)
4. Statement of Purpose and Letter of Recommendation Strength (out of 5)
5. Undergraduate GPA (out of 10)
6. Research Experience (either 0 or 1)
7. Chance of Admit (ranging from 0 to 1)

```
import numpy as np
import pandas as pd
```

```
df = pd.read_csv("Admission_Predict.csv")
df.head()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.shape
```

```
(400, 9)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         400 non-null    int64  
 1   GRE Score          400 non-null    int64  
 2   TOEFL Score        400 non-null    int64  
 3   University Rating  400 non-null    int64  
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64  
 8   Chance of Admit    400 non-null    float64
```

```
dtypes: float64(4), int64(5)
memory usage: 28.3 KB
```

```
df.duplicated().sum()
```

```
np.int64(0)
```

```
X= df.iloc[:, 0:7]
X
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	337	118	4	4.5	4.5	9.65	1
1	324	107	4	4.0	4.5	8.87	1
2	316	104	3	3.0	3.5	8.00	1
3	322	110	3	3.5	2.5	8.67	1
4	314	103	2	2.0	3.0	8.21	0
...	...	...	...	...	...	...	...
395	324	110	3	3.5	3.5	9.04	1
396	325	107	3	3.0	3.5	9.11	1
397	330	116	4	5.0	4.5	9.45	1
398	312	103	3	3.5	4.0	8.78	0
399	333	117	4	5.0	4.0	9.66	1

400 rows × 7 columns

Next steps: Generate code with X | View recommended plots | New interactive sheet

```
y = df.iloc[:, -1]
y
```

	Chance of Admit
0	0.92
1	0.76
2	0.72
3	0.80
4	0.65
...	...
395	0.82
396	0.84
397	0.91
398	0.67
399	0.95

400 rows × 1 columns

dtype: float64

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_state = 42)
```

```
#Use min-max scaling here
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
X_train_scaled
```

```
array([[0.64      , 0.64285714, 0.5       , ..., 0.375     , 0.59935897,
        1.        ],
       [0.56      , 0.64285714, 0.5       , ..., 0.5       , 0.64102564,
        0.        ],
       [1.        , 1.        , 1.        , ..., 0.875     , 0.99679487,
        1.        ],
       ...,
       [0.32      , 0.46428571, 0.25      , ..., 0.5       , 0.45512821,
        1.        ],
       [0.24      , 0.25      , 0.        , ..., 0.25      , 0.14423077,
        0.        ],
       [0.48      , 0.5       , 0.25      , ..., 0.625     , 0.46474359,
        0.        ]])
```

```
import tensorflow
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense, Input
```

```
model = Sequential()
model.add(Input(shape=(7,)))
model.add(Dense(7, activation = 'relu'))
model.add(Dense(7, activation = 'relu'))
```

```
model.add(Dense(1, activation = 'linear'))
#For a regression problem, the activation should be linear and no. of perceptrons = 1
```

```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 7)	56
dense_3 (Dense)	(None, 7)	56
dense_4 (Dense)	(None, 1)	8

Total params: 120 (480.00 B)
Trainable params: 120 (480.00 B)
Non-trainable params: 0 (0.00 B)

```
model.compile(loss = 'mean_squared_error', optimizer = 'Adam')
```

```
history = model.fit(X_train_scaled, y_train, epochs = 100, validation_split = 0.2)
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0053 - val_loss: 0.0043
Epoch 93/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0050 - val_loss: 0.0043
Epoch 94/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.0047 - val_loss: 0.0043
Epoch 95/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0050 - val_loss: 0.0043
Epoch 96/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 0.0057 - val_loss: 0.0043
Epoch 97/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0049 - val_loss: 0.0042
Epoch 98/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 0.0055 - val_loss: 0.0042
Epoch 99/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0056 - val_loss: 0.0043
Epoch 100/100
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 0.0050 - val_loss: 0.0042
```

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

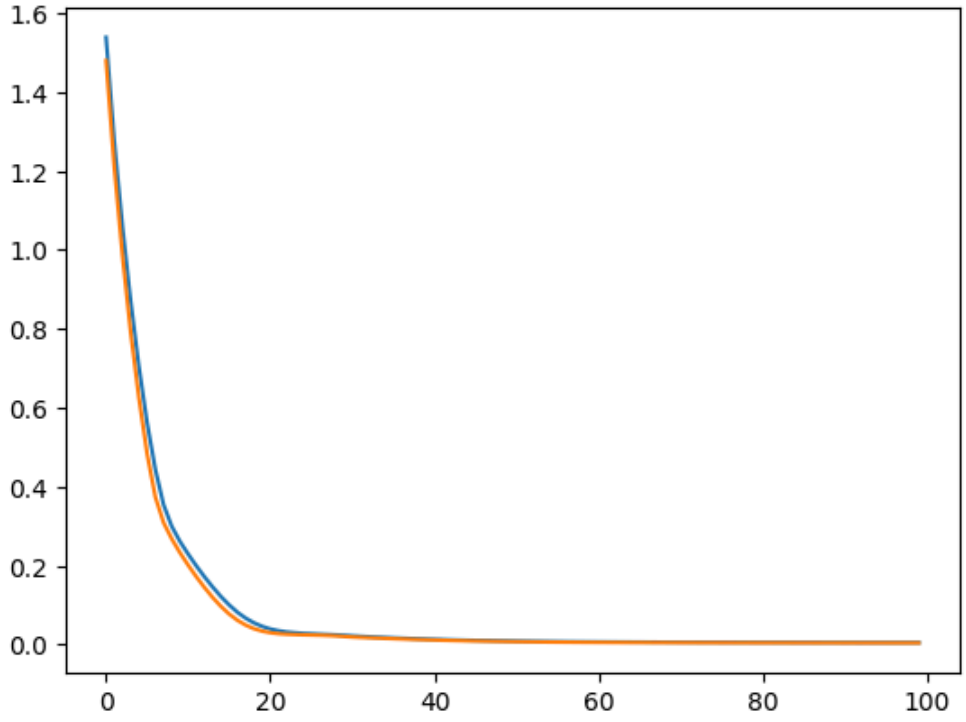
```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```

```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

```
0.7866705060948727
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x7d7414af7f90>]
```



Start coding or generate with AI