Experiment-2: To implement a Multilayer Perceptron (MLP) using Keras with TensorFlow for regression problem (house price prediction).

Double-click (or enter) to edit

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
df = pd.read_csv("/content/data.csv")
print(df.shape)
```

```
df=df.drop('date',axis=1)
# Remove the last 4 columns
df = pd.DataFrame(df.drop(columns=df.columns[-4:]))
print(df.shape)
print(df.isna().sum())
```

```
x = df.drop('price',axis=1)
y = df['price']
from sklearn.preprocessing import StandardScaler
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)
x_train=pd.DataFrame(x_train)
y_train=pd.DataFrame(y_train)
s=StandardScaler()
x_train=s.fit_transform(x_train)
y_train=s.fit_transform(y_train)
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Dense, Dropout
from sklearn.metrics import r2_score
#With Using Batch Normalization
model1=Sequential()
```

```
model1.add(Dense(32,activation='relu',input_dim=12))
model1.add(BatchNormalization())
model1.add(Dense(16,activation='relu'))
model1.add(BatchNormalization())
model1.add(Dense(1,activation='linear'))
model1.summary()
```

```
#Without using batch normalization
model2=Sequential()
model2.add(Dense(32,activation='relu',input_dim=12))
model2.add(Dense(16,activation='relu'))
model2.add(Dense(1,activation='linear'))
model2.summary()
```

```
model1.compile(optimizer='adam', loss='mean_squared_error',
metrics=['mean_squared_error'])
```

```
model2.compile(optimizer='adam', loss='mean_squared_error',
metrics=['mean_squared_error'])
history1=model1.fit(x_train,y_train,epochs=100,validation_split=0.5)
```

```
history2=model2.fit(x_train,y_train,epochs=100,validation_split=0.5)
```

```
#To observe whether model has overfitted or not
plt.plot(history1.history["mean_squared_error"],color="blue")
plt.plot(history1.history["val_loss"],color="red")
plt.plot(history2.history["val_loss"],color="yellow")
plt.legend(["mean square error","value loss","value loss2"],loc="upper right")
```