Deep Learning With Tensor Flow 1 (CSE 3793)

ASSIGNMENT-2: NEURAL NETWORK FOR REGRESSION AND CLASSIFICATION

Name:Tribhuwan Singh

Reg. No.: 2341019538

Section:23412C3

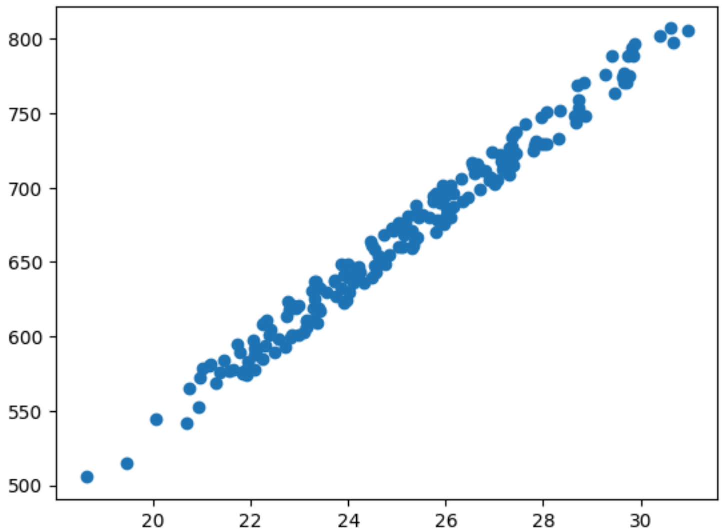
Serial No.: 46

Q1 Write a Python code to build a TensorFlow-Keras model (Sequential API) to implement a singlelayered perceptron (one Dense unit) that predicts house price from area

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

```
#genrate a random data
np.random.seed(0)
area = 2.5*np.random.randn(200)+25
price=25* area+5+np.random.randint(20,50, size = len(area))
```

```
#2 coverting the dataframe
data = np.array([area,price])
data = pd.DataFrame(data = data.T, columns = ['area','price'])
plt.scatter(data['area'],data['price'])
plt.show()
```




```
#3.Data normalization using min-max
data = (data-data.min())/(data.max()-data.min())#Normalise
```

```
#model bulding
model = K.Sequential([
    Dense(1,input_shape = [1,],activation=None)
])
model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1)	2

Total params: 2 (8.00 B)
Trainable params: 2 (8.00 B)
Non-trainable params: 0 (0.00 B)

```
#Model training by defining the loss function
model.compile(loss='mean_squared_error',optimizer="sgd")
model.fit(x=data['area'],y=data['price'],epochs=100,batch_size=32,verbose=1,validation_split=0.2)
```

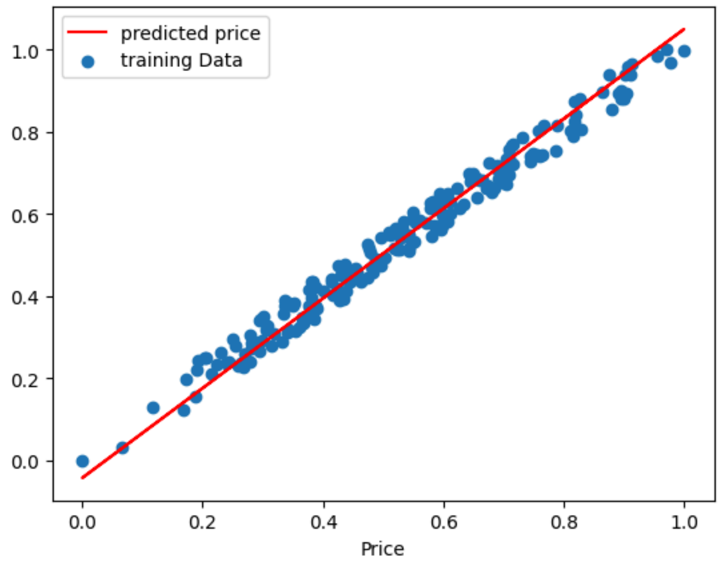
```
Epoch 1/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 84ms/step - loss: 1.4407 - val_loss: 1.0656
Epoch 2/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 1.1513 - val_loss: 0.8360
Epoch 3/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.8323 - val_loss: 0.6605
Epoch 4/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.6506 - val_loss: 0.5266
Epoch 5/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.5523 - val_loss: 0.4248
Epoch 6/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.4436 - val_loss: 0.3472
Epoch 7/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.3995 - val_loss: 0.2884
Epoch 8/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.3055 - val_loss: 0.2435
Epoch 9/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.2382 - val_loss: 0.2094
Epoch 10/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.2273 - val_loss: 0.1838
Epoch 11/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.2153 - val_loss: 0.1644
Epoch 12/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1778 - val_loss: 0.1497
Epoch 13/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1563 - val_loss: 0.1386
Epoch 14/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1585 - val_loss: 0.1302
Epoch 15/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1370 - val_loss: 0.1238
Epoch 16/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1366 - val_loss: 0.1189
Epoch 17/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1022 - val_loss: 0.1152
Epoch 18/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.1223 - val_loss: 0.1124
Epoch 19/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1212 - val_loss: 0.1103
Epoch 20/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.1131 - val_loss: 0.1086
Epoch 21/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.1155 - val_loss: 0.1073
Epoch 22/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1049 - val_loss: 0.1062
Epoch 23/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0927 - val_loss: 0.1053
Epoch 24/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1026 - val_loss: 0.1045
Epoch 25/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.1006 - val_loss: 0.1038
Epoch 26/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.1063 - val_loss: 0.1032
Epoch 27/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0961 - val_loss: 0.1027
Epoch 28/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0992 - val_loss: 0.1021
Epoch 29/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0981 - val_loss: 0.1016
```

```
#predicted output
y_pred = model.predict(data['area'])
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```

```
#6plot the actual and prediction value
plt.plot(data['area'],y_pred,color='red',label="predicted price")
plt.scatter(data['area'],data['price'],label="training Data")
plt.xlabel("Area")
plt.xlabel("Price")
```

```
plt.legend()
plt.show()
```




```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow.keras as K
from tensorflow.keras.layers import Dense,Normalization
import seaborn as sns
```

Q2 Write a Python code to build a regression model in TensorFlow–Keras to predict MPG (fuel efficiency) from the UCI Auto MPG dataset using: Keras Normalization layer for input standardization

(mean 0, std 1), a feed-forward network with two hidden layers (ReLU), and Adam optimizer and MSE loss.

```
!wget https://archive.ics.uci.edu/static/public/9/auto+mpg.zip
```

```
--2025-10-22 08:45:51--  https://archive.ics.uci.edu/static/public/9/auto+mpg.zip
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: ‘auto+mpg.zip’

auto+mpg.zip            [ <=>                ]  14.52K  --.-KB/s    in 0.05s

2025-10-22 08:45:52 (297 KB/s) - ‘auto+mpg.zip’ saved [14873]
```

```
import zipfile
with zipfile.ZipFile('auto+mpg.zip', 'r') as zip_ref:
    zip_ref.extractall('auto_mpg')
!ls auto_mpg
import pandas as pd
column_names = ['MPG','Cylinders','Displacement','Horsepower','Weight',
                'Acceleration', 'Model Year', 'Origin']
data = pd.read_csv('/content/auto-mpg.data', names=column_names, na_values = "?", comment='\t',sep=" ", skipinitialspace=Tı
data.head()
```

```
auto-mpg.data  auto-mpg.data-original  auto-mpg.names  Index
```

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Origin
0	18.0	8	307.0	130.0	3504.0	12.0	70	1
1	15.0	8	350.0	165.0	3693.0	11.5	70	1
2	18.0	8	318.0	150.0	3436.0	11.0	70	1
3	16.0	8	304.0	150.0	3433.0	12.0	70	1
4	17.0	8	302.0	140.0	3449.0	10.5	70	1

```
#Drop Column
data = data.drop('Origin',axis=1)
```

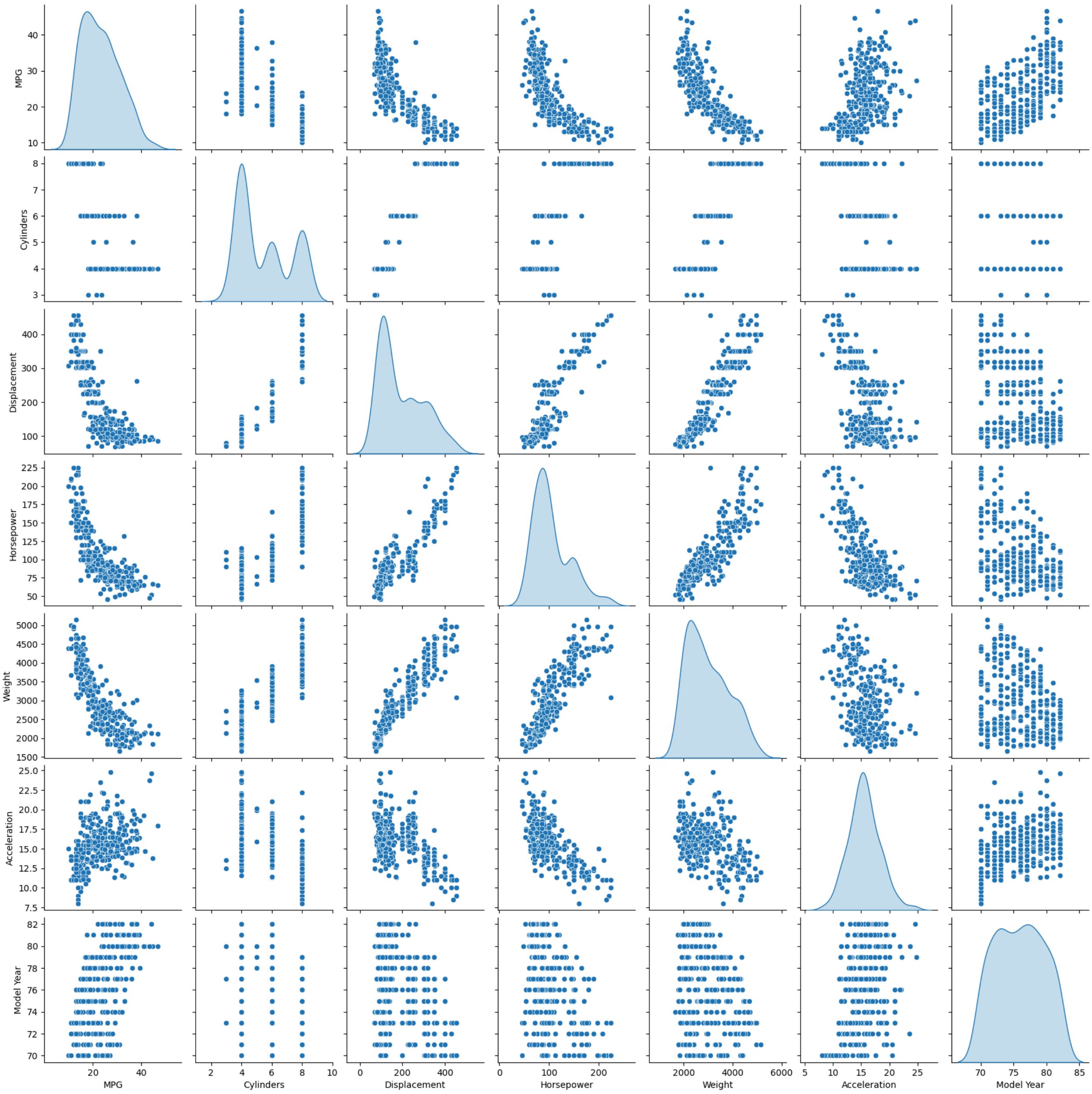
```
print(data.isna().sum())
data = data.dropna()
```

```
MPG             0
Cylinders       0
Displacement    0
Horsepower      6
Weight          0
Acceleration    0
Model Year      0
dtype: int64
```

```
train_dataset = data.sample(frac=0.8,random_state=0)
test_dataset= data.drop(train_dataset.index)
```

```
sns.pairplot(train_dataset[['MPG','Cylinders','Displacement','Horsepower','Weight','Acceleration','Model Year']],diag_kind=
```

```
<seaborn.axisgrid.PairGrid at 0x783bdccbbad0>
```




```
train_features= train_dataset.copy()
test_features=test_dataset.copy()
train_labels=train_features.pop('MPG')
test_labels=test_features.pop('MPG')
```

```
data_normalizer=Normalization(axis=-1)
data_normalizer.adapt(np.array(train_features))
```

```
#model bulding
model=K.Sequential([
```

```
    data_normalizer,
    Dense(64,activation='relu'),
    Dense(32,activation='relu'),
    Dense(1, activation=None)
])
model.summary()
```

Model: "sequential_7"

Layer (type)	Output Shape	Param #
normalization_2 (Normalization)	(314, 6)	13
dense_12 (Dense)	?	0 (unbuilt)
dense_13 (Dense)	?	0 (unbuilt)
dense_14 (Dense)	?	0 (unbuilt)

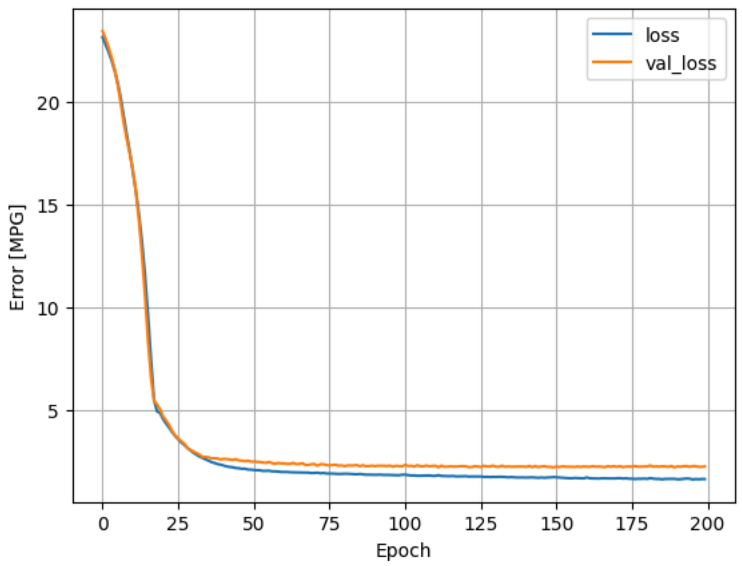
Total params: 13 (56.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 13 (56.00 B)

```
#model_training
model.compile(optimizer='adam',loss='mean_absolute_error')
history=model.fit(x=train_features,y=train_labels,epochs=200,batch_size=32,verbose=1,validation_split=0.2)
```

```
Epoch 1/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 22.8138 - val_loss: 23.4193
Epoch 2/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 22.7533 - val_loss: 23.0371
Epoch 3/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 22.6858 - val_loss: 22.6182
Epoch 4/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 22.6232 - val_loss: 22.1297
Epoch 5/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 21.6311 - val_loss: 21.5388
Epoch 6/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 21.8629 - val_loss: 20.8094
Epoch 7/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 20.4914 - val_loss: 19.9096
Epoch 8/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 20.0575 - val_loss: 18.9426
Epoch 9/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 18.6693 - val_loss: 18.1482
Epoch 10/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 18.3912 - val_loss: 17.4484
Epoch 11/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 16.1129 - val_loss: 16.6749
Epoch 12/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 15.6724 - val_loss: 15.6036
Epoch 13/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 14.3825 - val_loss: 14.2418
Epoch 14/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 13.2747 - val_loss: 12.5548
Epoch 15/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 12.4672 - val_loss: 10.5872
Epoch 16/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 9.7601 - val_loss: 8.3730
Epoch 17/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 7.8291 - val_loss: 6.5605
Epoch 18/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 5.7843 - val_loss: 5.4451
Epoch 19/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 4.9335 - val_loss: 5.2678
Epoch 20/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 5.1562 - val_loss: 5.0242
Epoch 21/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 4.6076 - val_loss: 4.6776
Epoch 22/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 4.1796 - val_loss: 4.4800
Epoch 23/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 4.2906 - val_loss: 4.2608
Epoch 24/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 3.9470 - val_loss: 3.9649
Epoch 25/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 3.9982 - val_loss: 3.7525
Epoch 26/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 3.5790 - val_loss: 3.6057
Epoch 27/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 3.2909 - val_loss: 3.4759
Epoch 28/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 3.3951 - val_loss: 3.3493
Epoch 29/200
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 3.2061 - val_loss: 3.1453
```

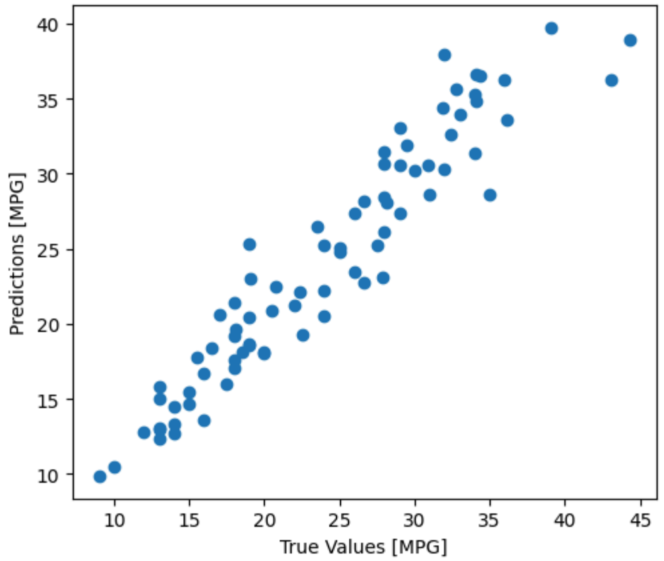
```
#plot
plt.plot(history.history['loss'],label='loss')
plt.plot(history.history['val_loss'],label='val_loss')
```

```
plt.xlabel('Epoch')
plt.ylabel('Error [MPG]')
plt.legend()
plt.grid(True)
```




```
#predict
y_pred=model.predict(test_features).flatten()
a=plt.axes(aspect='equal')
plt.scatter(test_labels,y_pred)
plt.xlabel('True Values [MPG]')
plt.ylabel('Predictions [MPG]')
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
Text(0, 0.5, 'Predictions [MPG]')
```



Q3 Write a Python code to build a logistic regression classifier for MNIST digits using TensorFlow–Keras.

```
((train_data, train_labels),(test_data, test_labels)) =tf.keras.datasets.mnist.load_data()
```

```
train_data = train_data/np.float32(255)
train_labels =train_labels.astype(np.int32)
test_data = test_data/np.float32(255)
test_labels = test_labels.astype(np.int32)
```

```
model = K.Sequential([
    K.layers.Flatten(input_shape=[28,28]),
    Dense(10,activation='softmax')
])
model.summary()
```

Model: "sequential_4"

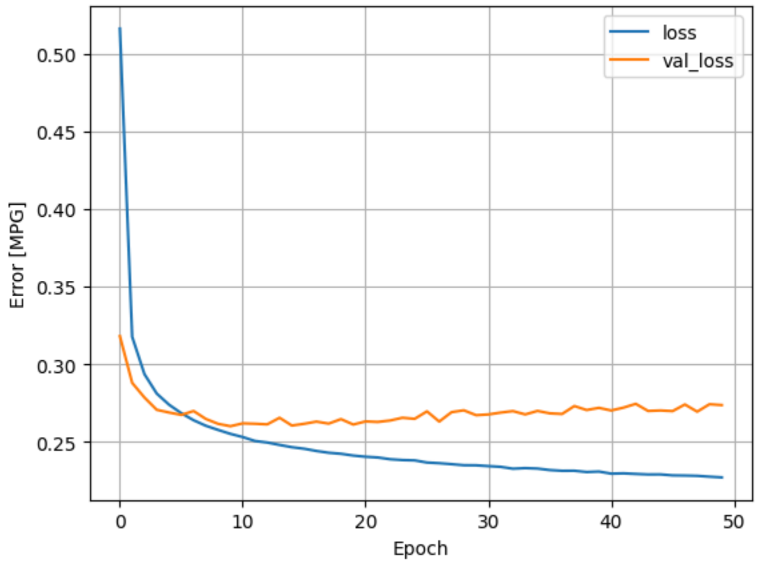
Layer (type)	Output Shape	Param #
flatten_3 (Flatten)	(None, 784)	0
dense_6 (Dense)	(None, 10)	7,850

Total params: 7,850 (30.66 KB)
Trainable params: 7,850 (30.66 KB)
Non-trainable params: 0 (0.00 B)

```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),metrics=['accuracy'])
history= model.fit(train_data,train_labels,epochs=50,verbose=1,validation_split =0.2)
```

```
Epoch 1/50
/usr/local/lib/python3.12/dist-packages/keras/src/backend/tensorflow/nn.py:717: UserWarning: "`sparse_categorical_crossent
  output, from_logits = _get_logits(
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.7919 - loss: 0.8051 - val_accuracy: 0.9126 - val_loss: 0.3181
Epoch 2/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9095 - loss: 0.3325 - val_accuracy: 0.9204 - val_loss: 0.2880
Epoch 3/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9182 - loss: 0.2955 - val_accuracy: 0.9215 - val_loss: 0.2785
Epoch 4/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9209 - loss: 0.2847 - val_accuracy: 0.9264 - val_loss: 0.2706
Epoch 5/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9229 - loss: 0.2739 - val_accuracy: 0.9254 - val_loss: 0.2689
Epoch 6/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9255 - loss: 0.2660 - val_accuracy: 0.9264 - val_loss: 0.2673
Epoch 7/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9270 - loss: 0.2621 - val_accuracy: 0.9256 - val_loss: 0.2698
Epoch 8/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9261 - loss: 0.2610 - val_accuracy: 0.9285 - val_loss: 0.2647
Epoch 9/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9297 - loss: 0.2482 - val_accuracy: 0.9308 - val_loss: 0.2616
Epoch 10/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9304 - loss: 0.2531 - val_accuracy: 0.9290 - val_loss: 0.2600
Epoch 11/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9293 - loss: 0.2531 - val_accuracy: 0.9307 - val_loss: 0.2619
Epoch 12/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9298 - loss: 0.2523 - val_accuracy: 0.9295 - val_loss: 0.2616
Epoch 13/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9302 - loss: 0.2494 - val_accuracy: 0.9296 - val_loss: 0.2612
Epoch 14/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9312 - loss: 0.2417 - val_accuracy: 0.9292 - val_loss: 0.2655
Epoch 15/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9316 - loss: 0.2456 - val_accuracy: 0.9308 - val_loss: 0.2605
Epoch 16/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9318 - loss: 0.2401 - val_accuracy: 0.9306 - val_loss: 0.2616
Epoch 17/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9326 - loss: 0.2430 - val_accuracy: 0.9287 - val_loss: 0.2630
Epoch 18/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9333 - loss: 0.2378 - val_accuracy: 0.9309 - val_loss: 0.2617
Epoch 19/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9329 - loss: 0.2423 - val_accuracy: 0.9289 - val_loss: 0.2646
Epoch 20/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9325 - loss: 0.2470 - val_accuracy: 0.9315 - val_loss: 0.2611
Epoch 21/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9331 - loss: 0.2355 - val_accuracy: 0.9298 - val_loss: 0.2631
Epoch 22/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9341 - loss: 0.2407 - val_accuracy: 0.9317 - val_loss: 0.2628
Epoch 23/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9357 - loss: 0.2388 - val_accuracy: 0.9293 - val_loss: 0.2637
Epoch 24/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9350 - loss: 0.2380 - val_accuracy: 0.9305 - val_loss: 0.2655
Epoch 25/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9365 - loss: 0.2266 - val_accuracy: 0.9298 - val_loss: 0.2648
Epoch 26/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9341 - loss: 0.2346 - val_accuracy: 0.9274 - val_loss: 0.2696
Epoch 27/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9355 - loss: 0.2289 - val_accuracy: 0.9312 - val_loss: 0.2630
Epoch 28/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9330 - loss: 0.2429 - val_accuracy: 0.9299 - val_loss: 0.2691
```

```
plt.plot(history.history['loss'],label='loss')
plt.plot(history.history['val_loss'],label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Error [MPG]')
plt.legend()
plt.grid(True)
```

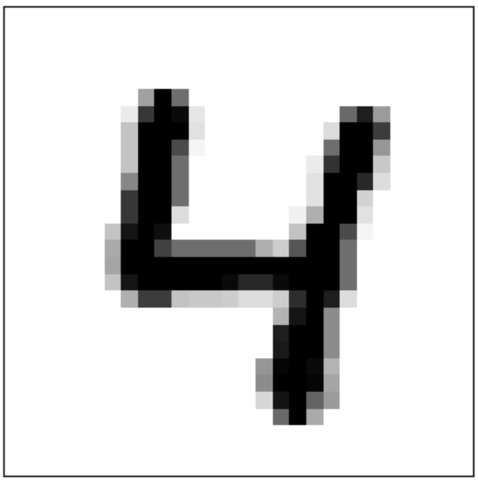


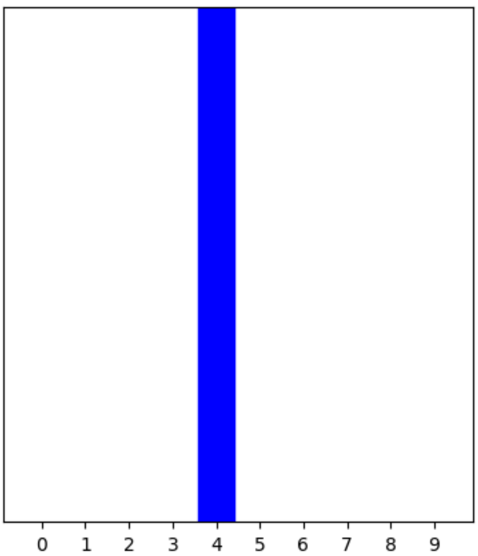

```
def plot_image(i, predictions_array, true_labels, images):
  true_label,img = true_labels[i], images[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)
  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'
    plt.xlabel("Pred {} Conf:{:2.0f}% True({})".format(predicted_label,100*np.max(predictions_array),true_label),color=colc
def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)
  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

```
predictions = model.predict(test_data)
i = 56
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plot_image(i , predictions[i], test_labels, test_data)
plt.subplot(1,2,2)
plot_value_array(i,predictions[i],test_labels)
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```





```
model = K.Sequential([
    K.layers.Flatten(input_shape=[28,28]),
    Dense(128,activation='relu'),
    Dense(10,activation='softmax')
])
model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/
  super().__init__(**kwargs)
Model: "sequential_5"
```

Layer (type)	Output Shape	Param #
flatten_4 (Flatten)	(None, 784)	0
dense_7 (Dense)	(None, 128)	100,480
dense_8 (Dense)	(None, 10)	1,290

```
 Total params: 101,770 (397.54 KB)
 Trainable params: 101,770 (397.54 KB)
 Non-trainable params: 0 (0.00 B)
```

```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),metrics=['accuracy'])
history= model.fit(train_data,train_labels,epochs=50,verbose=1,validation_split =0.2)
```

```
Epoch 1/50
/usr/local/lib/python3.12/dist-packages/keras/src/backend/tensorflow/nn.py:717: UserWarning: "`sparse_categorical_crossent
  output, from_logits = _get_logits(
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 9s 6ms/step - accuracy: 0.8716 - loss: 0.4684 - val_accuracy: 0.9538 - val_loss: 0.1663
Epoch 2/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9594 - loss: 0.1373 - val_accuracy: 0.9658 - val_loss: 0.1178
Epoch 3/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9744 - loss: 0.0897 - val_accuracy: 0.9677 - val_loss: 0.1087
Epoch 4/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9790 - loss: 0.0696 - val_accuracy: 0.9728 - val_loss: 0.0927
Epoch 5/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9838 - loss: 0.0521 - val_accuracy: 0.9743 - val_loss: 0.0895
Epoch 6/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9890 - loss: 0.0378 - val_accuracy: 0.9746 - val_loss: 0.0873
Epoch 7/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9924 - loss: 0.0271 - val_accuracy: 0.9741 - val_loss: 0.0882
Epoch 8/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9930 - loss: 0.0232 - val_accuracy: 0.9744 - val_loss: 0.0998
Epoch 9/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9938 - loss: 0.0192 - val_accuracy: 0.9751 - val_loss: 0.0857
Epoch 10/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9956 - loss: 0.0149 - val_accuracy: 0.9760 - val_loss: 0.0915
Epoch 11/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9971 - loss: 0.0115 - val_accuracy: 0.9778 - val_loss: 0.0932
Epoch 12/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9966 - loss: 0.0113 - val_accuracy: 0.9741 - val_loss: 0.1050
Epoch 13/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9977 - loss: 0.0085 - val_accuracy: 0.9781 - val_loss: 0.0914
Epoch 14/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9979 - loss: 0.0074 - val_accuracy: 0.9780 - val_loss: 0.0975
Epoch 15/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9978 - loss: 0.0076 - val_accuracy: 0.9762 - val_loss: 0.1028
Epoch 16/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9989 - loss: 0.0045 - val_accuracy: 0.9782 - val_loss: 0.1031
Epoch 17/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9988 - loss: 0.0046 - val_accuracy: 0.9759 - val_loss: 0.1089
Epoch 18/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9984 - loss: 0.0050 - val_accuracy: 0.9752 - val_loss: 0.1198
Epoch 19/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9983 - loss: 0.0060 - val_accuracy: 0.9739 - val_loss: 0.1299
Epoch 20/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9987 - loss: 0.0048 - val_accuracy: 0.9775 - val_loss: 0.1159
Epoch 21/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9990 - loss: 0.0039 - val_accuracy: 0.9774 - val_loss: 0.1197
Epoch 22/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9982 - loss: 0.0051 - val_accuracy: 0.9769 - val_loss: 0.1259
Epoch 23/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9994 - loss: 0.0024 - val_accuracy: 0.9778 - val_loss: 0.1235
Epoch 24/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.9987 - loss: 0.0043 - val_accuracy: 0.9768 - val_loss: 0.1345
Epoch 25/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9991 - loss: 0.0032 - val_accuracy: 0.9783 - val_loss: 0.1231
Epoch 26/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9996 - loss: 0.0017 - val_accuracy: 0.9747 - val_loss: 0.1424
Epoch 27/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9982 - loss: 0.0048 - val_accuracy: 0.9785 - val_loss: 0.1256
Epoch 28/50
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9993 - loss: 0.0021 - val_accuracy: 0.9744 - val_loss: 0.1621
```

```
plt.plot(history.history['loss'],label='loss')
plt.plot(history.history['val_loss'],label='val_loss')
```

```
plt.xlabel('Epoch')
plt.ylabel('Error [MPG]')
plt.legend()
plt.grid(True)
```