1st dl housing dataset

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from tqdm.notebook import tqdm
import warnings
warnings.filterwarnings("ignore")

boston = tf.keras.datasets.boston_housing

dir(boston)

boston_data = boston.load_data()

(x_train, y_train), (x_test, y_test) =
tf.keras.datasets.boston_housing.load_data(path='boston_housing.npz', test_split=0.2,
seed=42)

x_train.shape, y_train.shape, x_test.shape, y_test.shape

scaler = StandardScaler()

x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
y_train_scaled = scaler.fit_transform(y_train.reshape(-1, 1))
y_test_scaled = scaler.transform(y_test.reshape(-1, 1))

model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(13,), name='input-layer'),
    tf.keras.layers.Dense(100, name='hidden-layer-2'),
    tf.keras.layers.BatchNormalization(name='hidden-layer-3'),
    tf.keras.layers.Dense(50, name='hidden-layer-4'),
    tf.keras.layers.Dense(1, name='output-layer')
])

tf.keras.utils.plot_model(model, show_shapes=True)
```

```
model.summary()

model.compile(
    optimizer='adam',
    loss='mse',
    metrics=['mae']
)

history = model.fit(x_train_scaled, y_train_scaled, batch_size=32, epochs=20,
validation_data=(x_test, y_test))

pd.DataFrame(history.history).plot(figsize=(10,7))
plt.title("Metrics graph")
plt.show()

y_pred = model.predict(x_test)

sns.regplot(x=y_test, y=y_pred)
plt.title("Regression Line for Predicted values")
plt.show()

def regression_metrics_display(y_test, y_pred):
  print(f"MAE is {metrics.mean_absolute_error(y_test, y_pred)}")
  print(f"MSE is {metrics.mean_squared_error(y_test,y_pred)}")
  print(f"R2 score is {metrics.r2_score(y_test, y_pred)}")

regression_metrics_display(y_test, y_pred)
```

2nd pract

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
data_df = pd.read_csv('letter-recognition.csv')
data_df.head()

transposed_df = data_df.transpose()
transposed_df.head()

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
```

```
X = data_df.iloc[:, 1:].values
y = data_df['letter'].values
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_onehot = tf.keras.utils.to_categorical(y_encoded)
y_onehot

scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)
X_normalized

X_train, X_test, y_train, y_test = train_test_split(X_normalized, y_onehot, test_size=0.2,
random_state=42)
input_dim = X_train.shape[1]
output_dim = y_train.shape[1]
model = tf.keras.Sequential([
tf.keras.layers.InputLayer(input_shape=(input_dim,)),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.BatchNormalization(name='hidden-layer-3'),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(64, activation='relu'),
tf.keras.layers.Dense(output_dim, activation='softmax')
])
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])
model.summary()

history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.1)
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

y_pred = model.predict(X_test)
```

```
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)
results_df = pd.DataFrame({
'Predicted': label_encoder.inverse_transform(y_pred_classes),
'Actual': label_encoder.inverse_transform(y_true_classes)
})
print(results_df.head(30))

print(results_df.head(30))
```

DL 3

```
pip install mlxtend

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.utils import to_categorical
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from mlxtend.plotting import plot_confusion_matrix
from sklearn import metrics
import random
import warnings
warnings.filterwarnings("ignore")

pip install idx2numpy

import idx2numpy
trainX=idx2numpy.convert_from_file(r"C:\Users\Gites\Downloads\archive
(28)\train-images-idx3-ubyte")
testX=idx2numpy.convert_from_file(r"C:\Users\Gites\Downloads\archive
(28)\t10k-images-idx3-ubyte")
trainy=idx2numpy.convert_from_file(r"C:\Users\Gites\Downloads\archive
(28)\train-labels-idx1-ubyte")
testy=idx2numpy.convert_from_file(r"C:\Users\Gites\Downloads\archive
(28)\t10k-labels-idx1-ubyte")

trainX=trainX.reshape(trainX.shape[0],28,28,1)
testX=testX.reshape(testX.shape[0],28,28,1)
```

```
trainy_cat=to_categorical(trainy)
testy_cat=to_categorical(testy)

train_norm = trainX.astype('float32')
test_norm = testX.astype('float32')
train_norm = train_norm / 255.0
test_norm = test_norm / 255.0

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(trainX[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[trainy[i]])
plt.show()

model=tf.keras.models.Sequential([

tf.keras.layers.Conv2D(64,kernel_size=(3,3),input_shape=(28,28,1),activation='relu',padding='s
ame',name='conv-layer-1'),
    tf.keras.layers.AvgPool2D(pool_size=(2,2),name='pooling-layer-1'),
tf.keras.layers.Conv2D(32,kernel_size=(3,3),input_shape=(28,28,1),activation='relu',padding='s
ame',name='conv-layer-2'),
    tf.keras.layers.AvgPool2D(pool_size=(2,2),name='pooling-layer-2'),
    tf.keras.layers.GlobalAveragePooling2D(name='pooling-layer-3'),
    tf.keras.layers.Dense(len(class_names),activation='softmax',name='output-layer')
])

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history=model.fit(trainX,trainy_cat,epochs=10,validation_data=(testX,testy_cat))

model.summary()

pd.DataFrame(history.history).plot(figsize=(10,7))
plt.title("Metrics Graph")
plt.show()
```

```
model.evaluate(testX, testy_cat)

predictions = model.predict(testX)
predictions = tf.argmax(predictions, axis=1)
y_test = tf.argmax(testy_cat, axis=1)
y_test = tf.Variable(y_test)
print(metrics.accuracy_score(y_test, predictions))

print(metrics.classification_report(y_test, predictions))

cm = metrics.confusion_matrix(y_test, predictions)
plot_confusion_matrix(cm, figsize=(10,7), class_names=class_names)
plt.title("Confusion Matrix")
plt.show()

images = []
labels = []
random_indices = random.sample(range(len(testX)), 10)
for idx in random_indices:
    images.append(testX[idx])
    labels.append(testy_cat[idx])
images = np.array(images)
labels = np.array(labels)

fig = plt.figure(figsize=(20, 8))
rows = 2
cols = 5
x = 1
for image, label in zip(images, labels):
    fig.add_subplot(rows, cols, x)
    prediction = model.predict(tf.expand_dims(image, axis=0))
    prediction = class_names[tf.argmax(prediction.flatten())]
    label = class_names[tf.argmax(label)]
    plt.title(f"Label: {label}, Prediction: {prediction}")
    plt.imshow(image/255.)
    plt.axis("off")
    x += 1
```

DL4

```
import tensorflow as tf
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense
from tensorflow.keras import Sequential
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

df=pd.read_csv(r"GOOGL.csv",thousands=',')
df.head()

axis1=df.plot(x="Date",y=['Open','High','Low','Close'],figsize=(10,7),title="Open,High,Low,Close
Stock Prices Of of Google Stocks")
axis1.set_ylabel("Stock Prices")
axis2=df.plot(x="Date",y=['Volume'],figsize=(10,7))
axis2.set_ylabel("Stock Volume")

df.isnull().sum()

df[['Open','High','Low','Close','Volume']].plot(kind='box',layout=(1,5),subplots=True,sharex=False
,sharey=False,figsize=(10,7),color='red')

df.hist(figsize=(10,7))

scaler=MinMaxScaler()
data_without_date=df.drop("Date",axis=1)
scaled_df=scaler.fit_transform(data_without_date)
scaled_df=pd.DataFrame(scaled_df)
scaled_df.hist(figsize=(10,7))

plt.figure(figsize=(10,7))
sns.heatmap(df.drop("Date",axis=1).corr())
plt.show()

scaled_df=scaled_df.drop([0,2,3],axis=1)
scaled_df

def split_seq_multivariate(sequence,n_past,n_future):
    x=[]
    y=[]
    for window_start in range(len(sequence)):
        past_end=n_past+window_start
        future_end=past_end+n_future
        if future_end>len(sequence):
            break
        past=sequence.iloc[window_start:past_end,:]
        future=sequence.iloc[past_end:future_end,-1]
        x.append(past)
        y.append(future)
    return np.array(x),np.array(y)

n_steps=60
x,y=split_seq_multivariate(scaled_df,n_steps,1)
scaled_df.shape

x.shape,y.shape

X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
X_train.shape,X_test.shape,y_train.shape,y_test.shape

model=Sequential([
    LSTM(64,input_shape=(n_steps,3)),
    Dense(50,activation='relu'),
    Dense(50,activation='relu'),
    Dense(50,activation='relu'),
    Dense(1)
])

model.summary()

model.compile(optimizer='adam',loss='mse',metrics=['mae'])

history=model.fit(X_train,y_train,epochs=10,batch_size=32,verbose=2,validation_data=(X_test,y_test))

pd.DataFrame(history.history).plot(figsize=(10,7))

model.evaluate(X_test,y_test)

predictions=model.predict(X_test)
predictions.shape

plt.plot(y_test,c='r')
plt.plot(predictions,c='y')
plt.xlabel('Day')
plt.ylabel('Stock Price Volume')
plt.title('Stock Price Volume Prediction Graph using RNN(LSTM)')
plt.figure(figsize=(10,7))
plt.show()
```