DL1

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras.datasets import boston_housing
```

```
(x_train, y_train), (x_test, y_test) = boston_housing.load_data()
```

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

```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
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, y_train, batch_size=32, epochs=20,
validation_data=(x_test, y_test))



y_pred = model.predict(x_test)


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


  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)}")
```

DL2

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds

from mlxtend.plotting import plot_confusion_matrix
%matplotlib inline
from tqdm.notebook import tqdm
import warnings
warnings.filterwarnings("ignore")
```

```
vocab_size = 10000

max_len = 200

(x_train, y_train), (x_test, y_test) =
tf.keras.datasets.imdb.load_data(num_words=vocab_size)
```

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

```
x_train = tf.keras.preprocessing.sequence.pad_sequences(x_train,
maxlen=max_len)

x_test = tf.keras.preprocessing.sequence.pad_sequences(x_test,
maxlen=max_len)
```

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

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, 128, input_length=max_len),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

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

```
model.summary()
```

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

```
history = model.fit(x_train, y_train, batch_size=128, epochs=5, validation_data=(x_test, y_test))
```

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

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

```
y_pred = y_pred.flatten()
```

```
y_pred = (y_pred > 0.5).astype(int)
```

```
print(metrics.classification_report(y_test, y_pred))
```

```
cm = metrics.confusion_matrix(y_test, y_pred)

plot_confusion_matrix(cm, class_names=['Negative', 'Positive'])

plt.title("Confusion Matrix")

plt.show()
```

DL3

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


from sklearn import metrics

from mlxtend.plotting import plot_confusion_matrix

import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
```

```
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 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(x_train[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[y_train[i]])
plt.show()
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu',
name='conv-layer-1', input_shape=(28, 28, 1)),
    tf.keras.layers.AveragePooling2D(pool_size=(2, 2), name='pooling-layer-
1'),
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
name='conv-layer-2'),
    tf.keras.layers.AveragePooling2D(pool_size=(2, 2), name='pooling-layer-
2'),
    tf.keras.layers.GlobalAveragePooling2D(name='pooling-layer-3'),
    tf.keras.layers.Dense(10, activation='softmax', name='output-layer')
])
```

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

```
model.summary()
```

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

```
epochs = 10
```

```
history = model.fit(x_train, y_train, epochs=epochs, validation_data=(x_test,
y_test))
```

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

```
model.evaluate(x_test, y_test)
```

```
predictions = model.predict(x_test)
predictions = tf.argmax(predictions, axis=1)
```

```
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()
```

DL4

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
```

```
import datetime
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from sklearn.metrics import r2_score
```

```
data = pd.read_csv('Google_Stock_Price_Train.csv',thousands=',')
data
```

```
ax1 = data.plot(x="Date", y=["Open", "High", "Low", "Close"],
figsize=(10,7),title='Open, High, Low, Close Stock Prices of Google Stocks')
ax1.set_ylabel("Stock Price")

ax2 = data.plot(x="Date", y=["Volume"],  figsize=(10,7))
ax2.set_ylabel("Stock Volume")
```

```
data.isna().sum()
```

```
data[['Open','High','Low','Close','Volume']].plot(kind='box',  subplots=True,
figsize=(14,7))

plt.show()
```

```
data.hist(figsize=(10,7))
plt.show()
```

```
scaler = MinMaxScaler()
data_without_date = data.drop("Date", axis=1)
scaled_data = pd.DataFrame(scaler.fit_transform(data_without_date))
```

```
scaled_data.hist(figsize=(10,7))
plt.show()
```

```
plt.figure(figsize=(10,7))
sns.heatmap(data.drop("Date", axis=1).corr(), annot =True)
plt.show()
```

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

```
def split_seq_multivariate(sequence, n_past, n_future):

    '''
    n_past ==> no of past observations
    n_future ==> no of future observations
    '''
    x = []
    y = []
    for window_start in range(len(sequence)):
        past_end = window_start + n_past
        future_end = past_end + n_future
        if future_end > len(sequence):
            break
        # slicing the past and future parts of the window (this indexing is
for 2 features vala data only)
        past = sequence[window_start:past_end, :]
        future = sequence[past_end:future_end, -1]
        x.append(past)
        y.append(future)

    return np.array(x), np.array(y)
```

```
n_steps = 60
```

```
scaled_data = scaled_data.to_numpy()
scaled_data.shape
```

```
x, y = split_seq_multivariate(scaled_data, n_steps, 1)
```

```
x.shape, y.shape
```

```
y = y[:, 0]
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()
model.add(LSTM(612, input_shape=(n_steps, 2)))
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(1))
```

```
model.summary()
```

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

```
history = model.fit(x_train, y_train, epochs=200, batch_size=32, verbose=2,
validation_data=(x_test, y_test))
```

```
plt.figure(figsize=(14, 10))

plt.subplot(2, 1, 1)
pd.DataFrame({"mae": history.history["mae"], "val_mae":
history.history["val_mae"]}).plot(ax=plt.gca())
plt.title("Metrics graph - MAE")


plt.subplot(2, 1, 2)
pd.DataFrame({"loss": history.history["loss"], "val_loss":
history.history["val_loss"]}).plot(ax=plt.gca())
plt.title("Metrics graph - Loss")

plt.tight_layout()
plt.show()
```

```
model.evaluate(x_test, y_test)
```

```
predictions = model.predict(x_test)
predictions.shape
```

```
plt.plot(y_test[:50], c='r')
plt.plot(predictions[:50], c='y')
plt.xlabel('Day')
plt.ylabel('Stock Price Volume')
plt.title('Stock Price Volume Prediction Graph using RNN (LSTM)')
```

```
plt.legend(['Actual', 'Predicted'], loc='lower right')
plt.show()
```