Assignment 1

```
import numpy as np
import pandas as pd

data=pd.read_csv("E:/excel/boston.csv")

data.head()

data.shape #optional

data.isnull().sum()

data.dropna(inplace=True)

data.describe()  #optional

data.info()  #optional

import seaborn as sns
sns.histplot(data.PRICE)

correlation = data.corr()
correlation.loc['PRICE']

import matplotlib.pyplot as plt
fig,axes = plt.subplots(figsize=(15,12))
sns.heatmap(correlation,square = True,annot = True)
```

```
plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM', 'PTRATIO']
for i, col in enumerate(features):
    plt.subplot(1, len(features), i+1)
    x = data[col]
    y = data.PRICE
    plt.scatter(x, y, marker='o')
    plt.title("Variation in House prices")
    plt.xlabel(col)
    plt.ylabel('House prices in $1000')

plt.show()

X = data.iloc[:,:-1]
y= data.PRICE

from sklearn.model_selection import train_test_split
import numpy as np

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std

from sklearn.linear_model import LinearRegression
```

```
regressor = LinearRegression()

regressor.fit(X_train,y_train)

y_pred = regressor.predict(X_test)

from sklearn.metrics import mean_squared_error
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse)

from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2) #accuracy without deep learning

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

import tensorflow as tf
from keras.layers import Dense, Activation,Dropout
from keras.models import Sequential

model = Sequential()
model.add(Dense(128,activation = 'relu',input_dim =13))
model.add(Dense(64,activation = 'relu'))
model.add(Dense(32,activation = 'relu'))
model.add(Dense(16,activation = 'relu'))
model.add(Dense(1))
```

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

history = model.fit(X_train, y_train, epochs=100, validation_split=0.05)

from plotly.subplots import make_subplots
import plotly.graph_objects as go

y_pred = model.predict(X_test)
mse_nn, mae_nn = model.evaluate(X_test, y_test)
print('Mean squared error on test data: ', mse_nn)
print('Mean absolute error on test data: ', mae_nn)

from sklearn.metrics import mean_absolute_error
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
mse_lr = mean_squared_error(y_test, y_pred_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)
print('Mean squared error on test data: ', mse_lr)
print('Mean absolute error on test data: ', mae_lr)

from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2) #accuracy with deep learning

from sklearn.metrics import mean_squared_error
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse) #optional
```

```
import sklearn
new_data = sklearn.preprocessing.StandardScaler().fit_transform(([[0.1, 10.0,
5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]]))
prediction = model.predict(new_data)
print("Predicted house price:", prediction)
```

Assignment 2

```
from keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words = 10000)

word_index = imdb.get_word_index()

reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

decoded_review = ' '.join([reverse_word_index.get(i-3, '?') for i in train_data[0]])

decoded_review

import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))    # Creates an all zero matrix of shape
(len(sequences),10K)
    for i,sequence in enumerate(sequences):
        results[i,sequence] = 1                    # Sets specific indices of results[i] to 1s
    return results

X_train = vectorize_sequences(train_data)
```

```
X_test = vectorize_sequences(test_data)

X_train[0]

X_train.shape

y_train = np.asarray(train_labels).astype('float32')
y_test  = np.asarray(test_labels).astype('float32')

from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

from keras import optimizers
from keras import losses
from keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss = losses.binary_crossentropy,
              metrics = [metrics.binary_accuracy])

X_val = X_train[:10000]
partial_X_train = X_train[10000:]
```

```
y_val = y_train[:10000]
partial_y_train = y_train[10000:]

history = model.fit(partial_X_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(X_val, y_val))

history_dict = history.history
history_dict.keys()

import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')

loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, loss_values, 'bo', label="Training Loss")
plt.plot(epochs, val_loss_values, 'b', label="Validation Loss")

plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss Value')
plt.legend()

plt.show()
```

```
acc_values = history_dict['binary_accuracy']
val_acc_values = history_dict['val_binary_accuracy']

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, acc_values, 'ro', label="Training Accuracy")
plt.plot(epochs, val_acc_values, 'r', label="Validation Accuracy")

plt.title('Training and Validation Accuraccy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

model.fit(partial_X_train,
          partial_y_train,
          epochs=3,
          batch_size=512,
          validation_data=(X_val, y_val))

np.set_printoptions(suppress=True)
result = model.predict(X_test)
result

y_pred = np.zeros(len(result))
for i, score in enumerate(result):
    y_pred[i] = 1 if score > 0.5 else 0
```

```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_pred, y_test)
mae
```

Assignment 3

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
from sklearn import metrics

(x_train, y_train), (x_test, y_test) = mnist.load_data()

plt.imshow(x_train[0], cmap='gray')
plt.show()

print(x_train[0])

print("X_train shape", x_train.shape)
print("y_train shape", y_train.shape)
print("X_test shape", x_test.shape)
print("y_test shape", y_test.shape)

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
```

```
x_test = x_test.astype('float32')
x_train /= 255 # Each image has Intensity from 0 to 255
x_test /= 255

num_classes = 10
y_train = np.eye(num_classes)[y_train] # Return a 2-D array with ones on the diagonal and zeros elsewhere.
y_test = np.eye(num_classes)[y_test]

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))

model.add(Dropout(0.2))
model.add(Dense(512, activation='relu')) #returns a sequence of another vectors of dimension 512
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', # for a multi-class classification problem
optimizer=RMSprop(),
metrics=['accuracy'])

batch_size = 128 # batch_size argument is passed to the layer to define a batch size for the inputs.
epochs = 10
history = model.fit(x_train, y_train,
batch_size=batch_size,
epochs=epochs,
verbose=1, # verbose=1 will show you an animated progress bar eg. [==========]
```

```
validation_data=(x_test, y_test))


score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

(x_train, y_train), (x_test, y_test) = mnist.load_data()

plt.imshow(x_train[1], cmap='gray')
plt.show()
input_image = x_train[1].reshape(1, 784)
predictions = model.predict(input_image)
predicted_class = np.argmax(predictions[0])
print("Predicted class:", predicted_class)
```

Assignment 4

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

(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

plt.imshow(x_train[1])
plt.imshow(x_train[0])

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

```
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

x_train.shape
(60000, 28, 28)
x_test.shape
(10000, 28, 28, 1)
y_train.shape
(60000,)
y_test.shape
(10000,)

model = keras.Sequential([
keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
   keras.layers.MaxPooling2D((2,2)),
keras.layers.Dropout(0.25),
keras.layers.Conv2D(64, (3,3), activation='relu'),
keras.layers.MaxPooling2D((2,2)),
keras.layers.Dropout(0.25),
keras.layers.Conv2D(128, (3,3), activation='relu'),

keras.layers.Flatten(),
keras.layers.Dense(128, activation='relu'),

keras.layers.Dropout(0.25),
keras.layers.Dense(10, activation='softmax')])
model.summary()

Model: "sequential"
```

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

history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))

test_loss, test_acc = model.evaluate(x_test, y_test)

print('Test accuracy:', test_acc)
```

Assignment 5

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


import datetime

import math


from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean_squared_error


from keras.models import Sequential  # linear stack of layers

from keras.layers import Dense

from keras.layers import LSTM       # Long Short-Term Memory layer

from keras.layers import Dropout     # simple way to prevent overfitting


SHARE = 'AAPL'

SERVICE = 'fred' #'quandl' #'yahoo'


PREDICTORS = ['Open']  #['High', 'Low', 'Open']   # column names with prices

TARGET = 'Open'
```

```
TIMESTEP = 90  # the number of previous days used for prediction


START_DATE = datetime.datetime(2010, 1, 1)  # doesn't work for Kaggle Notebook, train set is used instead

END_DATE = datetime.datetime(2019, 9, 30)


START_DATE_TO_PREDICT = datetime.datetime(2019, 10, 1)  # doesn't work for Kaggle,test set is used instead

END_DATE_TO_PREDICT = datetime.datetime(2019, 10, 31)


N_EPOCHS = 100


df_train = pd.read_csv('E:/Github/DL_assignments/Google_Stock_Price_Train.csv')

df_test = pd.read_csv('E:/Github/DL_assignments/Google_Stock_Price_Test.csv')


df_test.tail()


df_train['Date'] = pd.to_datetime(df_train['Date'])

df_test['Date'] = pd.to_datetime(df_test['Date'])


df_train.set_index('Date', inplace=True)

df_test.set_index('Date', inplace=True)


df_train = df_train[PREDICTORS]

df_test = df_test[PREDICTORS]


training_set = df_train.values


sc = MinMaxScaler(feature_range = (0, 1))
```

```
training_set_scaled = sc.fit_transform(training_set)

X_train = []
y_train = []
target_col_index = df_train.columns.get_loc(TARGET)
for i in range(TIMESTEP, len(training_set)):
    X_train.append(training_set_scaled[i-TIMESTEP:i, :])      # X_train - list of Numpy arrays
    y_train.append(training_set_scaled[i, target_col_index])
X_train, y_train = np.array(X_train), np.array(y_train)       # convert list to Numpy array

regressor = Sequential()
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1],
X_train.shape[2])))
regressor.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation
regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

regressor.add(Dense(units = 1))

regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')

regressor.fit(X_train, y_train, epochs = N_EPOCHS, batch_size = 32)
```

```
# Extracting real prices
real_stock_price = df_test[TARGET].values

df_total = df_train.append(df_test)
inputs = df_total[len(df_total) - len(df_test) - TIMESTEP:][PREDICTORS]

inputs = sc.transform(inputs)



X_test = []
for i in range(TIMESTEP, TIMESTEP+len(df_test)):
   X_test.append(inputs[i-TIMESTEP:i, :])
X_test = np.array(X_test)

predicted_stock_price = regressor.predict(X_test)

temp_matrix = np.zeros((len(predicted_stock_price), len(PREDICTORS)))
temp_matrix[:,target_col_index:target_col_index+1] = predicted_stock_price  #
temp_matrix[:,[target_col_index]] = predicted_stock_price

predicted_stock_price = sc.inverse_transform(temp_matrix)[:,target_col_index]

df_test['Predicted price'] = predicted_stock_price
df_test[TARGET].plot(figsize=(16,4),legend=True)
df_test['Predicted price'].plot(figsize=(16,4),legend=True)
plt.legend(['Real price', 'Predicted price'])
plt.title('RNN - ' + SHARE + ' Stock Price Prediction')
plt.xlabel('Date')
plt.ylabel('Price')
```

```
plt.show()


rmse = math.sqrt(mean_squared_error(real_stock_price, predicted_stock_price))
print("The RMSE is {:.3f}.".format(rmse))
```