Python Tutorial and Homework 2

This Colab Notebook contains 2 sections. The first section is a Python Tutorial intended to make you familiar with Numpy and Colab functionalities. **This section will not be graded**.

The second section is a coding assignment (Homework 2). **This section will be graded**

1.0 Python Tutorial

This assignment section won't be graded but is intended as a tutorial to refresh the basics of python and its dependencies. It also allows one to get familiarized with Google Colab.

Double-click (or enter) to edit

Double-click (or enter) to edit

1.0.0 Array manipulation using numpy

Q1 - Matrix multiplication

```
import numpy as np

### Create two numpy arrays with the dimensions 3x2 and 2x3 respectively using  np.arange().
### The elements of the vector are
### Vector 1 elements = [ 2,  4,  6,  8, 10, 12];
### Vector 2 elements = [ 7, 10, 13, 16, 19, 22]

### Starting at 2, stepping by 2
vector1 = np.arange(2,14,2)
### Starting at 7, stepping by 3
vector2 =np.arange(7,24,3)

### Print vec
print(vector1, vector2)

### Take product of the two matricies (Matrix product)
prod = vector1 @ vector2

### Print
print(prod)
```

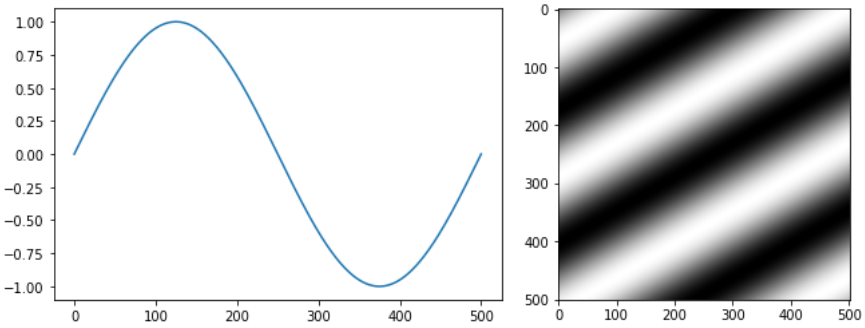
```
[ 2  4  6  8 10 12] [ 7 10 13 16 19 22]
714
```

Q2 - Diagonals

```
### Create two numpy arrays with the dimensions 10x10 using the function np.arange().
### Starting at 2, stepping by 3
vector1 = np.arange(2,302,3).reshape(10,10)

# vector1 = np.arange(2,30,3)
### Starting at 35, stepping by 9
vector2 = np.arange(35,935,9).reshape(10,10)

### Print vec
print(vector1,vector2)

### Obtain the diagonal matrix of each vector1 such that the start of the diagonal is from (3,0) and the end is (9,6)
### Reshape the the matrix such that it form a diagonal maritix of shape(7,7)
vector1_offset_diagonal = np.diag(np.diag(vector1, -3))


### Obtain a 7x7 matrix from the vector 2
### starting from (left top element) = (0,3)
### ending at (right bottom element) = (6,9)
vector2_offset_diagonal = np.diag(np.diag(vector2,3))

### Print diagonal matrix
print(vector1_offset_diagonal, vector2_offset_diagonal)


### Take product of the two diagonal matricies (Matrix product)
prod = vector1_offset_diagonal @ vector2_offset_diagonal

### Print
print(prod)
```

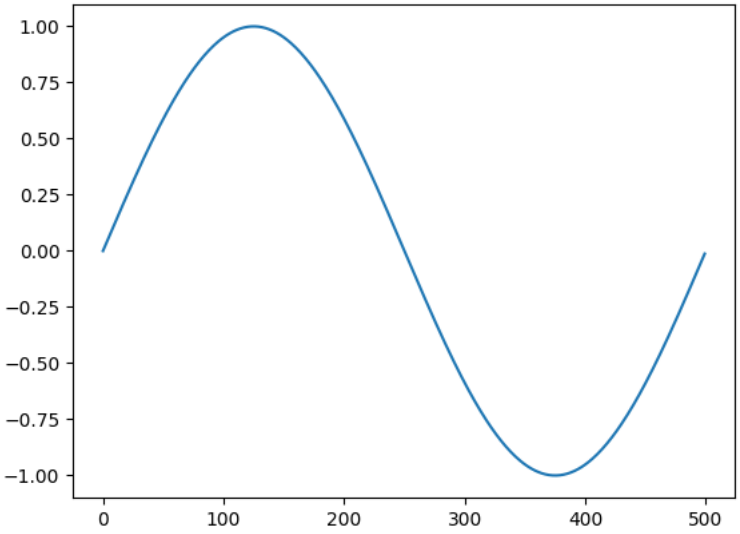
```
     [[  2   5   8  11  14  17  20  23  26  29]
      [ 32  35  38  41  44  47  50  53  56  59]
      [ 62  65  68  71  74  77  80  83  86  89]
      [ 92  95  98 101 104 107 110 113 116 119]
      [122 125 128 131 134 137 140 143 146 149]
      [152 155 158 161 164 167 170 173 176 179]
      [182 185 188 191 194 197 200 203 206 209]
      [212 215 218 221 224 227 230 233 236 239]
      [242 245 248 251 254 257 260 263 266 269]
      [272 275 278 281 284 287 290 293 296 299]] [[ 35  44  53  62  71  80  89  98 107 116]
      [125 134 143 152 161 170 179 188 197 206]
      [215 224 233 242 251 260 269 278 287 296]
      [305 314 323 332 341 350 359 368 377 386]
      [395 404 413 422 431 440 449 458 467 476]
      [485 494 503 512 521 530 539 548 557 566]
      [575 584 593 602 611 620 629 638 647 656]
      [665 674 683 692 701 710 719 728 737 746]
      [755 764 773 782 791 800 809 818 827 836]
      [845 854 863 872 881 890 899 908 917 926]]
     [[ 92   0   0   0   0   0   0]
      [  0 125   0   0   0   0   0]
      [  0   0 158   0   0   0   0]
      [  0   0   0 191   0   0   0]
      [  0   0   0   0 224   0   0]
      [  0   0   0   0   0 257   0]
      [  0   0   0   0   0   0 290]] [[ 62   0   0   0   0   0   0]
      [  0 161   0   0   0   0   0]
      [  0   0 260   0   0   0   0]
      [  0   0   0 359   0   0   0]
      [  0   0   0   0 458   0   0]
      [  0   0   0   0   0 557   0]
      [  0   0   0   0   0   0 656]]
     [[  5704      0      0      0      0      0      0]
      [     0  20125      0      0      0      0      0]
      [     0      0  41080      0      0      0      0]
      [     0      0      0  68569      0      0      0]
      [     0      0      0      0 102592      0      0]
      [     0      0      0      0      0 143149      0]
      [     0      0      0      0      0      0 190240]]
```

Q3 - Sin wave

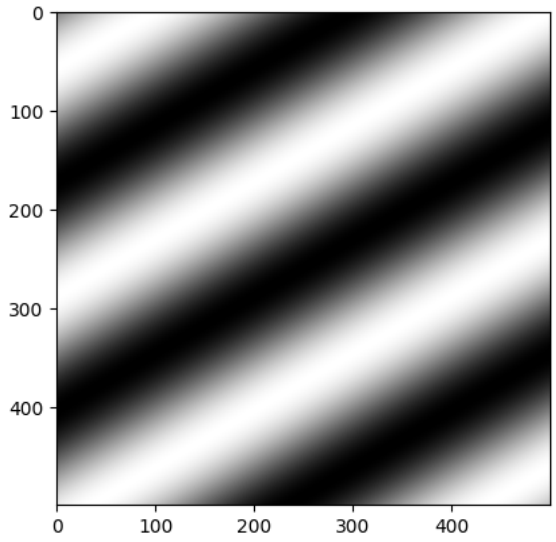


Sample outputs,

```
import matplotlib.pyplot as plt
### Create a time matrix that evenly samples a sine wave at a frequency of 1Hz
### Starting at time step T = 0
### End at time step T = 500
time = np.arange(0,500,1)

### Given wavelength of
wavelength = 500

### Construct a sin wave using the formula sin(2*pi*(time/wavelength))
y = np.sin(2*np.pi*(time/wavelength))

#### Plot the wave
plt.plot(time, y)
plt.show()
```




```
#### Given a 2D mesh grid
X, Y = np.meshgrid(time, time)
#### wavelength and angle of rotation(phi) of the sin wave in 2d. Imagine a 2D sine wave is being rotating about the Z axes
wavelength = 200
phi = np.pi / 3

#### Calculate the sin wave in 2d space using the formula sin(2*pi*(x'/wavelength)) where x'=Xcos(phi) + Ysin(phi)
x_prime = X*np.cos(phi) + Y*np.sin(phi)
grating = np.sin(2*np.pi*(x_prime/wavelength))

#### Plot the wave
#### Intuition, think of the white area as hills and the black areas as valleys
plt.set_cmap("gray")
plt.imshow(grating)
plt.show()
```



Q4 Car Brands

```
cars = ['Civic', 'Insight', 'Fit', 'Accord', 'Ridgeline', 'Avancier','Pilot', 'Legend', 'Beat', 'FR-V', 'HR-V', 'Shuttle']

#### Create a 3D array of cars of shape 2,3,2
cars = np.array(cars)
cars_3d = cars.reshape(2,3,2)

#### Extract the top layer of the matrix. Top layer of a matrix A of shape(2,3,2) will have the following structue A_top = [[A[0,0,0], A[0,(
#### HINT - Array slicing or splitting
cars_top_layer = cars_3d[0]

#### Similarly extract the bottom layer
#### HINT - Array slicing or splitting
cars_bottom_layer = cars_3d[1]

#### Print layers
print("\nTop Layer \n ",cars_top_layer,"\nBottom Layer\n", cars_bottom_layer)

#### Flatten the top layer
cars_top_flat = cars_top_layer.flatten()
#### Flatten the bottom layer
cars_bottom_flat = cars_bottom_layer.flatten()

#### Print layers
print("\nTop Flattened : ",cars_top_flat,"\nBottom Flattened : ",cars_bottom_flat)


new_car_list = np.empty((cars_top_layer.size + cars_bottom_layer.size,), dtype=object)
#### Interweave the to flattened lists and insert into new_car_list such that new_car_list=['Civic' 'Pilot' 'Fit' 'Beat' 'Ridgeline' 'HR-V'
#### Using only array slicing
new_car_list[0::2] = np.append(cars_top_flat[0::2] , cars_top_flat[1::2])
new_car_list[1::2] = np.append(cars_bottom_flat[0::2] ,cars_bottom_flat[1::2])

# print(new_car_list)

# new_car_list[0:6:2] = cars_top_flat[0::2]
# new_car_list[1:7:2] = cars_bottom_flat[0::2]
# new_car_list[6::2] = cars_top_flat[1::2]
# new_car_list[7::2] = cars_bottom_flat[1::2]

# print(new_car_list)


#### Concatenate and flatten the top and bottom layer such that the final list is of the form cat_flat = ['Civic' 'Insight' 'Pilot' 'Legend
cat_flat = np.concatenate((cars_top_layer,cars_bottom_layer), axis=1).flatten()

#### Print layers
print("\n\nInterwoven - ", new_car_list,"\nConcatenate and flatten - ", cat_flat)


     Top Layer
```

```
       [['Civic' 'Insight']
      ['Fit' 'Accord']
      ['Ridgeline' 'Avancier']]
     Bottom Layer
      [['Pilot' 'Legend']
      ['Beat' 'FR-V']
      ['HR-V' 'Shuttle']]

     Top Flattened :  ['Civic' 'Insight' 'Fit' 'Accord' 'Ridgeline' 'Avancier']
     Bottom Flattened :  ['Pilot' 'Legend' 'Beat' 'FR-V' 'HR-V' 'Shuttle']


     Interwoven -  ['Civic' 'Pilot' 'Fit' 'Beat' 'Ridgeline' 'HR-V' 'Insight' 'Legend'
      'Accord' 'FR-V' 'Avancier' 'Shuttle']
     Concatenate and flatten -  ['Civic' 'Insight' 'Pilot' 'Legend' 'Fit' 'Accord' 'Beat' 'FR-V'
      'Ridgeline' 'Avancier' 'HR-V' 'Shuttle']
```

1.0.1 Basics tensorflow

Helper functions

```
import tensorflow as tf
from keras.utils import to_categorical


def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[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'

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')
```

Q1 MNIST Classifier

```
## Import the MNIST dataset from keras
mnist = tf.keras.datasets.mnist
### Load the data
(x_train, y_train), (x_test, y_test) = mnist.load_data()

### Normalize the 8bit images with values in the range [0,255]
x_train, x_test =



## Create a model with the following architecture Flatten -> Dense(128, relu) -> Dense(64,relu) -> outpuLayer(size=10)
model =

### Complie the model with the adam optimizer and crossentropy loss
### HINT - No One hot encoding
model.compile()


### Train the model on the train data for 5 epochs
model.fit()
```

```
### Check the accuracy of the trained model
test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)
print('\nTest accuracy:', test_acc)


### Convert the above model to a probabilistic model with a softmax as the output layer
probability_model =

### Run the test data through the new model and get predictions
predictions =

### Plot a test output
i = 42 ### <---- Change this to some random number to see different predictions
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], y_test, x_test)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  y_test)
plt.show()
### Blue bars mean correct guess red bar means wrong guess!!
```

1.0.2 Basic Pytorch Tutorial

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms

# Set the random seed for reproducibility
torch.manual_seed(42)

# Define a simple feedforward neural network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 128)  # 28x28 input size (MNIST images are 28x28 pixels)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 10)   # 10 output classes (digits 0-9)

    def forward(self, x):
        x = x.view(-1, 784)  # Flatten the input image
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Load the MNIST dataset and apply transformations
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)

testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False)

# Initialize the neural network and optimizer
net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01)

# Training loop
num_epochs = 10
for epoch in range(num_epochs):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data

        optimizer.zero_grad()

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f'Epoch {epoch+1}, Loss: {running_loss / len(trainloader)}')

print('Finished Training')

# Evaluate the model on the test set
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy on test set: {100 * correct / total}%')
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9912422/9912422 [00:00<00:00, 314586192.83it/s]Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28881/28881 [00:00<00:00, 56237555.16it/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1648877/1648877 [00:00<00:00, 151028376.06it/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4542/4542 [00:00<00:00, 10084980.82it/s]Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw


Epoch 1, Loss: 0.7497772100065817
Epoch 2, Loss: 0.36693694127965837
Epoch 3, Loss: 0.32101490559068313
Epoch 4, Loss: 0.29383779380684977
Epoch 5, Loss: 0.273217272605183
Epoch 6, Loss: 0.2538067406571623
Epoch 7, Loss: 0.2366939038077969
Epoch 8, Loss: 0.22136161107816169
Epoch 9, Loss: 0.2073850411016232
Epoch 10, Loss: 0.19490794614275128
Finished Training
Accuracy on test set: 94.47%
```

2.0 **Homework 2**

90 points

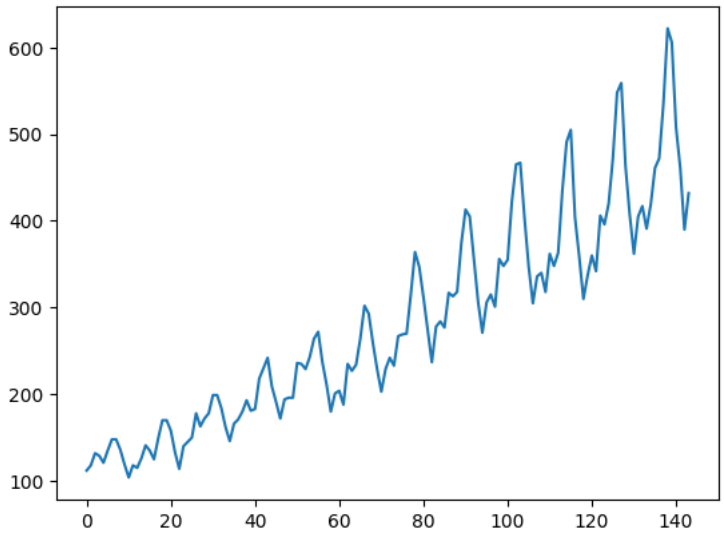
Note : This section will be graded and must be attemped using Pytorch only

Graded Section : Deep Learning Approach

Time-Series Prediction Time series and sequence prediction could be a really amazing to predict/estimate a robot's trajectory which requires temporal data at hand. In this assignemnt we will see how this could be done using Deep Learning.

Given a dataset link for airline passengers prediction problem. Predict the number of international airline passengers in units of 1,000 given a year and a month. Here is how the data looks like.

```
import matplotlib.pyplot as plt
import pandas as pd
import torch
import torchvision
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import math

file_name = '/content/airline-passengers.csv' # dataset path
# Reading data using pandas or csv
df = pd.read_csv(file_name)



# plotting the dataset
timeseries = df[["Passengers"]].values.astype('float32')
# plotting the dataset
plt.plot(timeseries)
plt.show()
```



1. Write the dataloader code to pre-process the data for pytorch tensors using any library of your choice. Here is a good resource for the dataloader Video link

Dataloader:

```
# Convert "Month" to datetime and extract year and month
df['Month'] = pd.to_datetime(df['Month'], format="%Y/%m")
df = df.set_index('Month')
dataset = df[["Passengers"]].values.astype('float32')

# Split into 70% train and #0% test
train_size = int(len(dataset)*0.70)
test_size = len(dataset) - train_size

train = dataset[0:train_size]
test  = dataset[train_size:]

train_data = df[:train_size].iloc[:,::2]
test_data = df[train_size:].iloc[:,::2]

# Scaling the data
scaler = MinMaxScaler(feature_range = (0,1))

train_scaled = scaler.fit_transform(train)
test_scaled = scaler.transform(test)

# print(*test_scaled[:5])

def create_dataset(dataset1, sequence_size = 1):
  X , y = [], []
  for i in range(len(dataset1)-sequence_size):
    feature = dataset1[i:i+sequence_size]
    target = dataset1[i+1:i+sequence_size+1]
    target = target[-1]
    X.append(feature)
    y.append(target)
  return torch.tensor(X, dtype=torch.float32), torch.tensor(y, dtype=torch.float32)

sequenceSize = 5
X_train, y_train = create_dataset(train_scaled, sequence_size=sequenceSize)
X_test, y_test = create_dataset(test_scaled, sequence_size=sequenceSize)

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1])

batch_size = 10
trainloader = torch.utils.data.DataLoader(torch.utils.data.TensorDataset(X_train, y_train), shuffle=True, batch_size=batch_size)
testloader = torch.utils.data.DataLoader(torch.utils.data.TensorDataset(X_test, y_test), shuffle=False, batch_size=batch_size)
```

```
<ipython-input-3-4b01bf38a06a>:32: UserWarning: Creating a tensor from a list of numpy.ndarrays is extremely slow. Please consider conv
  return torch.tensor(X, dtype=torch.float32), torch.tensor(y, dtype=torch.float32)
```

2. Create the model in pytorch here uinsg 1. Long-Short Term Memory (LSTM) and 2. Recurrent Neural Network (RNN). Here is a good resource for Custom model generation.

   Train using the two models. Here is the resource for the same Video link

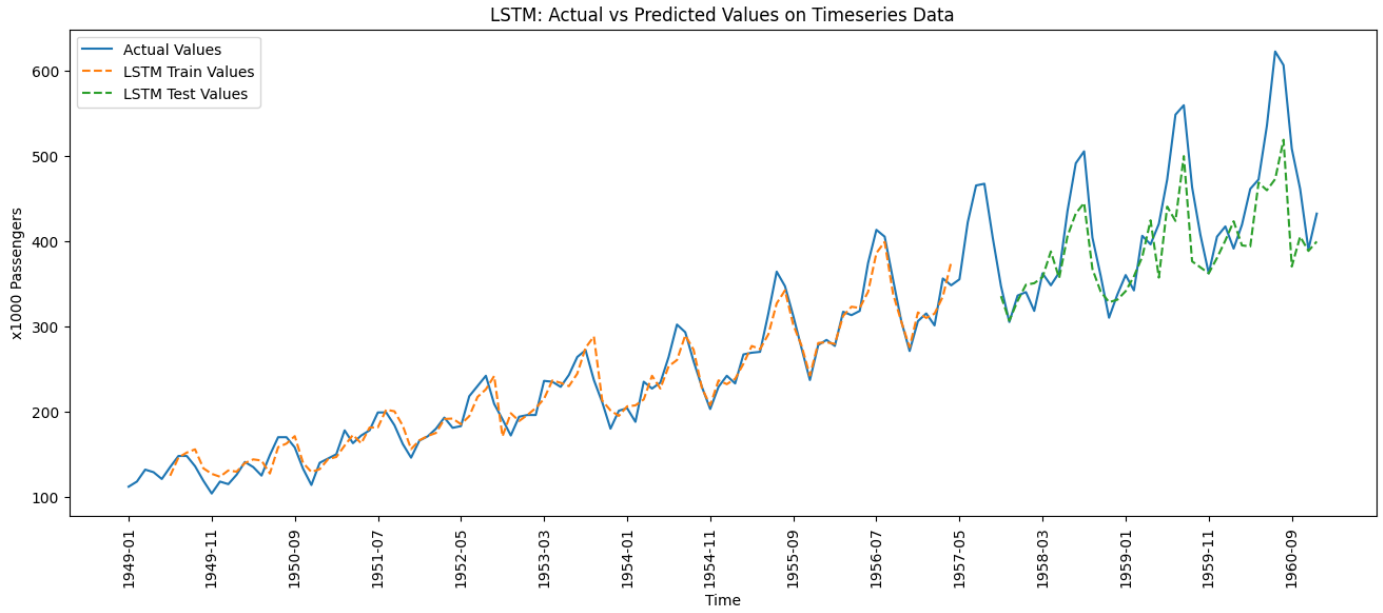
1. LSTM Model

```
# write your code here for the custom model creating for both the methods
import torch
import torch.nn as nn

# Model Prediction

class LSTM_model(nn.Module):
  def __init__(self, input_size, hidden_size, num_layers):
    super(LSTM_model, self).__init__()
    self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
    self.linear = nn.Linear(hidden_size, 1)

  def forward(self,x):
    x, _ = self.lstm(x)
    # x=x[-1]
```

```
    x = self.linear(x)
    return x

input_size = sequenceSize
hidden_size = 64
num_layers = 2
output_size = 1

lstm_model = LSTM_model(input_size, hidden_size, num_layers)

# Calculate Mean Squared Error (MSE)
mse_loss = nn.MSELoss(reduction='mean')
# Calculate Mean Absolute Error (MAE)
mae_loss = nn.L1Loss()
# Calculate Root Mean Squared Error (RMSE)
# rmse_loss = torch.sqrt(mse_loss)

optimizer = torch.optim.Adam(lstm_model.parameters(), lr=1e-3)
num_epochs = 2000

train_hist =[]
test_hist =[]
print(75*"*")
print('LSTM')
print(75*"*")
print('Training Model')
print(75*"*")

for epoch in range(num_epochs):
  lstm_model.train()
  total_loss = 0.0
  for inputs, labels in trainloader:
    outputs = lstm_model(inputs)
    mse_error = mse_loss(outputs, labels)
    mae_error = mae_loss(outputs, labels)
    rmse_error = torch.sqrt(mse_error)
    optimizer.zero_grad()
    mse_error.backward()
    optimizer.step()
    total_loss +=mse_error.item()
  average_loss = total_loss / len(trainloader)
  train_hist.append(average_loss)
  if (epoch+1)%100==0:
    # print(f'Epoch [{epoch+1}/{num_epochs}], Training Loss: {average_loss:.4f}')
    print(f"Epoch [{epoch+1}/{num_epochs}], Training MSE: {mse_error:.4f}, Training MAE: {mae_error:.4f}, Training RMSE: {rmse_error:.4f}")

print(75*"*")
print('Finished Training')
print(75*"*")
print('Evaluating Model')
print(75*"*")

lstm_model.eval()
with torch.no_grad():
  total_test_loss= 0.0
  for inputs, label_test in testloader:
    output_test = lstm_model(inputs)
    mse_error = mse_loss(output_test, label_test)
    mae_error = mae_loss(output_test, label_test)
    rmse_error = torch.sqrt(mse_error)
    total_test_loss += mse_error.item()
  average_test_loss = total_test_loss/len(testloader)
  test_hist.append(average_test_loss)
  print(f"Test MSE: {mse_error:.4f}, Test MAE: {mae_error:.4f}, Test RMSE: {rmse_error:.4f}")
  # print(f'Test Loss: {average_test_loss:.4f}')

  #Collecting Data for Plot
  train_plot = np.ones_like(timeseries) * np.nan
  train_plot[sequenceSize:len(train)] = scaler.inverse_transform(lstm_model(X_train))
  # shift test predictions for plotting
  test_plot = np.ones_like(timeseries) * np.nan
  test_plot[len(train)+sequenceSize:len(timeseries)] = scaler.inverse_transform(lstm_model(X_test))
# plot
print(75*"*")
plt.figure(figsize=(16, 6))
plt.plot(timeseries, label = 'Actual Values')
plt.xticks(np.arange(0,len(timeseries),10),labels = df.index[0::10].strftime('%Y-%m'),rotation='vertical')
plt.plot(train_plot, label= 'LSTM Train Values', linestyle='dashed')
```

```
plt.plot(test_plot, label= 'LSTM Test Values', linestyle='dashed')
plt.xlabel('Time')
plt.ylabel('x1000 Passengers')
plt.title('LSTM: Actual vs Predicted Values on Timeseries Data')
plt.legend()
plt.show()
```

```
**********************************************************************
LSTM
**********************************************************************
Training Model
**********************************************************************
Epoch [100/2000], Training MSE: 0.0077, Training MAE: 0.0726, Training RMSE: 0.0876
Epoch [200/2000], Training MSE: 0.0041, Training MAE: 0.0485, Training RMSE: 0.0637
Epoch [300/2000], Training MSE: 0.0048, Training MAE: 0.0587, Training RMSE: 0.0694
Epoch [400/2000], Training MSE: 0.0063, Training MAE: 0.0693, Training RMSE: 0.0791
Epoch [500/2000], Training MSE: 0.0052, Training MAE: 0.0582, Training RMSE: 0.0719
Epoch [600/2000], Training MSE: 0.0086, Training MAE: 0.0852, Training RMSE: 0.0929
Epoch [700/2000], Training MSE: 0.0128, Training MAE: 0.0906, Training RMSE: 0.1131
Epoch [800/2000], Training MSE: 0.0040, Training MAE: 0.0592, Training RMSE: 0.0630
Epoch [900/2000], Training MSE: 0.0033, Training MAE: 0.0469, Training RMSE: 0.0576
Epoch [1000/2000], Training MSE: 0.0027, Training MAE: 0.0371, Training RMSE: 0.0520
Epoch [1100/2000], Training MSE: 0.0052, Training MAE: 0.0592, Training RMSE: 0.0722
Epoch [1200/2000], Training MSE: 0.0055, Training MAE: 0.0625, Training RMSE: 0.0742
Epoch [1300/2000], Training MSE: 0.0048, Training MAE: 0.0467, Training RMSE: 0.0691
Epoch [1400/2000], Training MSE: 0.0020, Training MAE: 0.0335, Training RMSE: 0.0451
Epoch [1500/2000], Training MSE: 0.0014, Training MAE: 0.0316, Training RMSE: 0.0371
Epoch [1600/2000], Training MSE: 0.0015, Training MAE: 0.0298, Training RMSE: 0.0386
Epoch [1700/2000], Training MSE: 0.0023, Training MAE: 0.0395, Training RMSE: 0.0477
Epoch [1800/2000], Training MSE: 0.0027, Training MAE: 0.0484, Training RMSE: 0.0516
Epoch [1900/2000], Training MSE: 0.0016, Training MAE: 0.0375, Training RMSE: 0.0398
Epoch [2000/2000], Training MSE: 0.0017, Training MAE: 0.0325, Training RMSE: 0.0407
**********************************************************************
Finished Training
**********************************************************************
Evaluating Model
**********************************************************************
Test MSE: 0.0715, Test MAE: 0.2116, Test RMSE: 0.2674
**********************************************************************
```



2. RNN Model

```
# write your code here for the custom model creating for both the methods
import torch
import torch.nn as nn
```

```
# Model Prediction

class RNN_model(nn.Module):
  def __init__(self, input_size, hidden_size, num_layers):
    super(RNN_model, self).__init__()
    self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
    self.linear = nn.Linear(hidden_size, 1)

  def forward(self,x):
    x, _ = self.rnn(x)
    # x=x[-1]
    x = self.linear(x)
    return x

input_size = sequenceSize
hidden_size = 64
num_layers = 2
output_size = 1

rnn_model = RNN_model(input_size, hidden_size, num_layers)

# Calculate Mean Squared Error (MSE)
mse_loss = nn.MSELoss(reduction='mean')
# Calculate Mean Absolute Error (MAE)
mae_loss = nn.L1Loss()
# Calculate Root Mean Squared Error (RMSE)
# rmse_loss = torch.sqrt(mse_loss)

optimizer = torch.optim.Adam(rnn_model.parameters(), lr=1e-3)
num_epochs = 2000

train_hist =[]
test_hist =[]
print(75*"*")
print('RNN')
print(75*"*")
print('Training Model')
print(75*"*")

for epoch in range(num_epochs):
  rnn_model.train()
  total_loss = 0.0
  for inputs, labels in trainloader:
    outputs = rnn_model(inputs)
    mse_error = mse_loss(outputs, labels)
    mae_error = mae_loss(outputs, labels)
    rmse_error = torch.sqrt(mse_error)
    optimizer.zero_grad()
    mse_error.backward()
    optimizer.step()
    total_loss +=mse_error.item()
  average_loss = total_loss / len(trainloader)
  train_hist.append(average_loss)
  if (epoch+1)%100==0:
    # print(f'Epoch [{epoch+1}/{num_epochs}], Training Loss: {average_loss:.4f}')
    print(f"Epoch [{epoch+1}/{num_epochs}], Training MSE: {mse_error:.4f}, Training MAE: {mae_error:.4f}, Training RMSE: {rmse_error:.4f}")

print(75*"*")
print('Finished Training')
print(75*"*")
print('Evaluating Model')
print(75*"*")

rnn_model.eval()
with torch.no_grad():
  total_test_loss= 0.0
  for inputs, label_test in testloader:
    output_test = rnn_model(inputs)
    mse_error = mse_loss(output_test, label_test)
    mae_error = mae_loss(output_test, label_test)
    rmse_error = torch.sqrt(mse_error)
    total_test_loss += mse_error.item()
  average_test_loss = total_test_loss/len(testloader)
  test_hist.append(average_test_loss)
  print(f"Test MSE: {mse_error:.4f}, Test MAE: {mae_error:.4f}, Test RMSE: {rmse_error:.4f}")
  # print(f'Test Loss: {average_test_loss:.4f}')
```

```
  #Collecting Data for Plot
  train_plot = np.ones_like(timeseries) * np.nan
  train_plot[sequenceSize:len(train)] = scaler.inverse_transform(rnn_model(X_train))
  # shift test predictions for plotting
  test_plot = np.ones_like(timeseries) * np.nan
  test_plot[len(train)+sequenceSize:len(timeseries)] = scaler.inverse_transform(rnn_model(X_test))
# plot
print(75*"*")
plt.figure(figsize=(16, 6))
plt.plot(timeseries, label = 'Actual Values')
plt.xticks(np.arange(0,len(timeseries),10),labels = df.index[0::10].strftime('%Y-%m'),rotation='vertical')
plt.plot(train_plot, label= 'RNN Train Values', linestyle='dashed')
```