Experiment No. 3

Design RNN or its variant including LSTM or GRU

a) Select a suitable time series dataset. Example – predict sentiments based on product reviews b) Apply for prediction

```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, GRU, Dense, Dropout
```

```
# 1. Load and Prepare the Dataset
# Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv"
data = pd.read_csv(url, usecols=[1], engine='python')
data = data.values.astype('float32')

# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)
```

```
# 2. Create Sequences for Time Series
def create_sequences(dataset, look_back=12):
    X, y = [], []
    for i in range(len(dataset) - look_back):
        X.append(dataset[i:i + look_back])
        y.append(dataset[i + look_back])
    return np.array(X), np.array(y)

# Sequence parameters
look_back = 12
X, y = create_sequences(data, look_back)
X = np.reshape(X, (X.shape[0], X.shape[1], 1))
```

```
# 3. Define LSTM Model
lstm_model = Sequential()
lstm_model.add(LSTM(100, return_sequences=False, input_shape=(X.shape[1], 1)))
lstm_model.add(Dropout(0.2))
lstm_model.add(Dense(1))
lstm_model.compile(optimizer='adam', loss='mse')
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argum
  super().__init__(**kwargs)
```

```
# 4. Define GRU Model
gru_model = Sequential()
gru_model.add(GRU(100, return_sequences=False, input_shape=(X.shape[1], 1)))
gru_model.add(Dropout(0.2))
gru_model.add(Dense(1))
gru_model.compile(optimizer='adam', loss='mse')
```

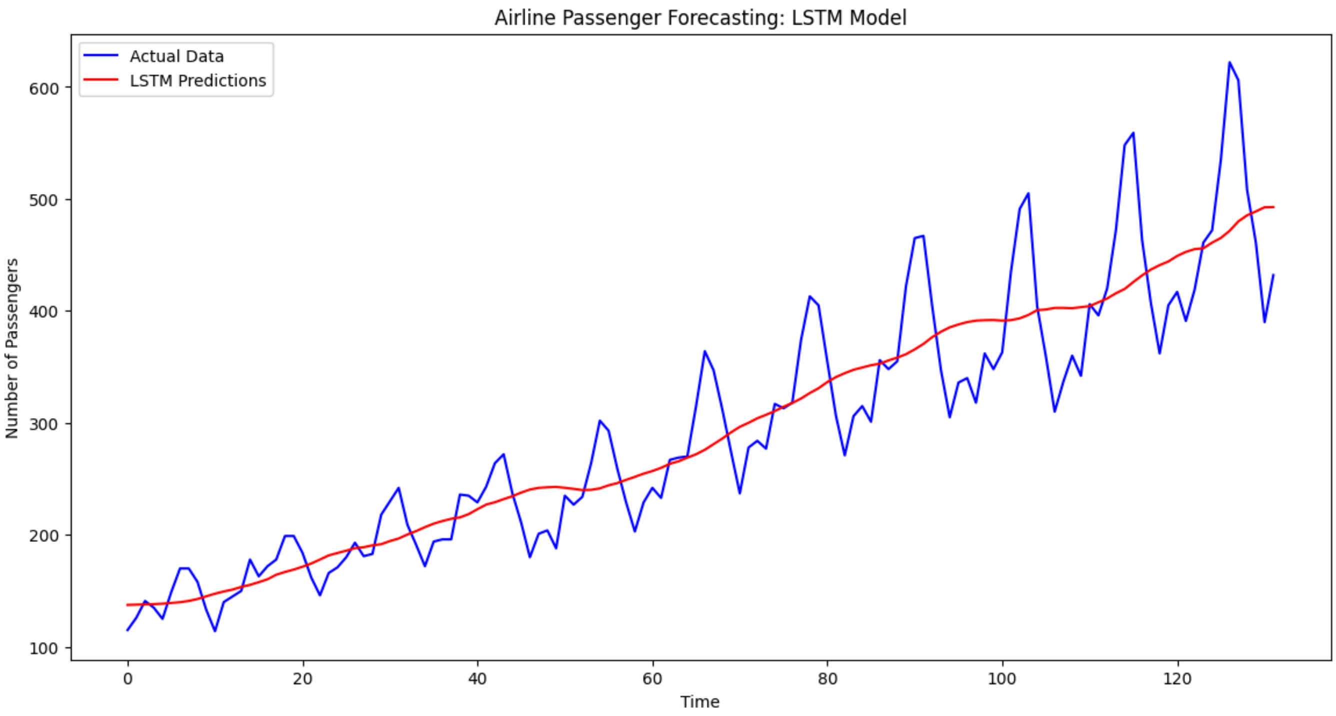
```
# 5. Train Both Models
epochs = 50
batch_size = 32

print("\nTraining LSTM Model...")
lstm_history = lstm_model.fit(X, y, epochs=epochs, batch_size=batch_size, validation_split=0.2, verbose=1)

print("\nTraining GRU Model...")
gru_history = gru_model.fit(X, y, epochs=epochs, batch_size=batch_size, validation_split=0.2, verbose=1)
```

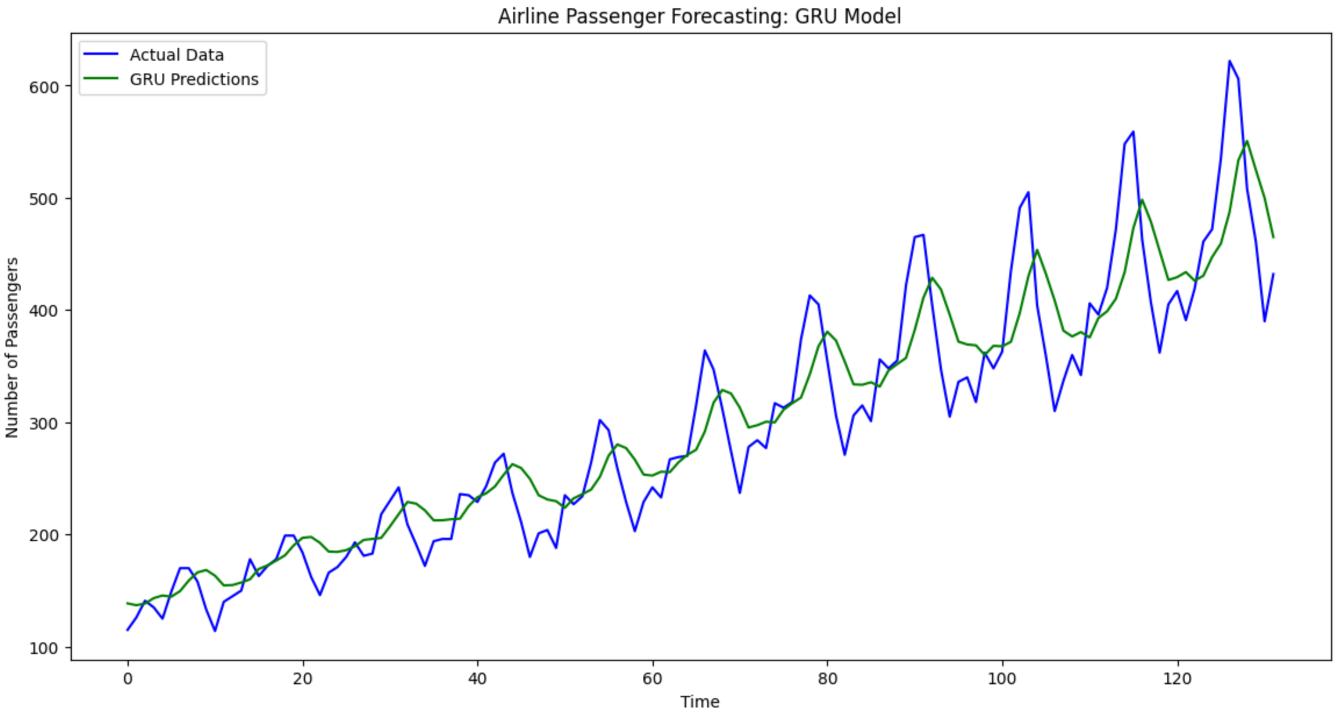
```
Epoch 28/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 0.0064 - val_loss: 0.0205
Epoch 29/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - loss: 0.0064 - val_loss: 0.0202
Epoch 30/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - loss: 0.0059 - val_loss: 0.0204
Epoch 31/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - loss: 0.0067 - val_loss: 0.0198
Epoch 32/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - loss: 0.0077 - val_loss: 0.0195
Epoch 33/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0065 - val_loss: 0.0193
Epoch 34/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 0.0061 - val_loss: 0.0189
Epoch 35/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0063 - val_loss: 0.0186
Epoch 36/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0069 - val_loss: 0.0183
Epoch 37/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - loss: 0.0063 - val_loss: 0.0183
Epoch 38/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - loss: 0.0058 - val_loss: 0.0180
Epoch 39/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 0.0063 - val_loss: 0.0177
Epoch 40/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0072 - val_loss: 0.0175
Epoch 41/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - loss: 0.0062 - val_loss: 0.0174
Epoch 42/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - loss: 0.0049 - val_loss: 0.0172
Epoch 43/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - loss: 0.0059 - val_loss: 0.0169
Epoch 44/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 0.0059 - val_loss: 0.0168
Epoch 45/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - loss: 0.0066 - val_loss: 0.0164
Epoch 46/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.0060 - val_loss: 0.0162
Epoch 47/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.0052 - val_loss: 0.0162
Epoch 48/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.0052 - val_loss: 0.0167
Epoch 49/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 0.0052 - val_loss: 0.0155
Epoch 50/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - loss: 0.0051 - val_loss: 0.0154
```

```
# 6. Prediction
# LSTM Predictions
lstm_pred = lstm_model.predict(X)
lstm_pred = scaler.inverse_transform(lstm_pred)

# GRU Predictions
gru_pred = gru_model.predict(X)
gru_pred = scaler.inverse_transform(gru_pred)

# Actual Values
actual = scaler.inverse_transform(y.reshape(-1, 1))
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
```

```
# 7. Separate Visualizations for LSTM and GRU
# LSTM Graph
plt.figure(figsize=(14, 7))
plt.plot(actual, label='Actual Data', color='blue')
plt.plot(lstm_pred, label='LSTM Predictions', color='red')
plt.xlabel('Time')
plt.ylabel('Number of Passengers')
plt.title('Airline Passenger Forecasting: LSTM Model')
plt.legend()
plt.show()

# GRU Graph
plt.figure(figsize=(14, 7))
plt.plot(actual, label='Actual Data', color='blue')
plt.plot(gru_pred, label='GRU Predictions', color='green')
plt.xlabel('Time')
plt.ylabel('Number of Passengers')
plt.title('Airline Passenger Forecasting: GRU Model')
plt.legend()
plt.show()
```





```
# 8. Evaluation Metrics
# LSTM Metrics
lstm_mse = mean_squared_error(actual, lstm_pred)
lstm_mae = mean_absolute_error(actual, lstm_pred)

# GRU Metrics
gru_mse = mean_squared_error(actual, gru_pred)
gru_mae = mean_absolute_error(actual, gru_pred)

# Display Metrics
print("\nLSTM Model Evaluation:")
print(f"Mean Squared Error: {lstm_mse}")
print(f"Mean Absolute Error: {lstm_mae}")

print("\nGRU Model Evaluation:")
print(f"Mean Squared Error: {gru_mse}")
print(f"Mean Absolute Error: {gru_mae}")
```

```
LSTM Model Evaluation:
Mean Squared Error: 2204.25048828125
Mean Absolute Error: 35.13486862182617

GRU Model Evaluation:
Mean Squared Error: 1892.03759765625
Mean Absolute Error: 33.32218551635742
```