Lab 11: Deep Learning for Network Performance Prediction using RNNs and LSTMs

This lab focuses on implementing Recurrent Neural Network architectures and LSTM for predicting ercentage of visitors using international roaming in Singapore.

Data Dictionary

Feature	Description	Data Type
ds	Timestamp indicating the month and year of the observation.	datetime
y	Percentage of visitors using international roaming services during the given month.	string

Lab Tasks Overview

1. **Data Loading and Initial Analysis**
    - Load and examine the dataset
    - Check for missing values
    - Analyze basic statistics
2. **Data Preprocessing**
    - Convert timestamps
    - Feature scaling
    - Sequence preparation
3. **Basic RNN Implementation**
    - Create sequences
    - Build simple RNN model
    - Train and evaluate
4. **LSTM Implementation**
    - Build LSTM architecture
    - Train and evaluate

Task: 1. **Data Loading and Initial Analysis**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from statsmodels.tsa.seasonal import seasonal_decompose

import tensorflow as tf
```

```
# Load the dataset
def load_and_analyze_data(file_path):
    # Read the data
    df = pd.read_csv(file_path)

    # Display basic information
    print("\nDataset Info:")
    print(df.info())

    # Display basic statistics
    print("\nBasic Statistics:")
    print(df.describe())

    # Check missing values
    print("\nMissing Values:")
    print(df.isnull().sum())

    return df
```

```
# Load the data
df = load_and_analyze_data('/content/InternationRoaming_singapore.csv')
```

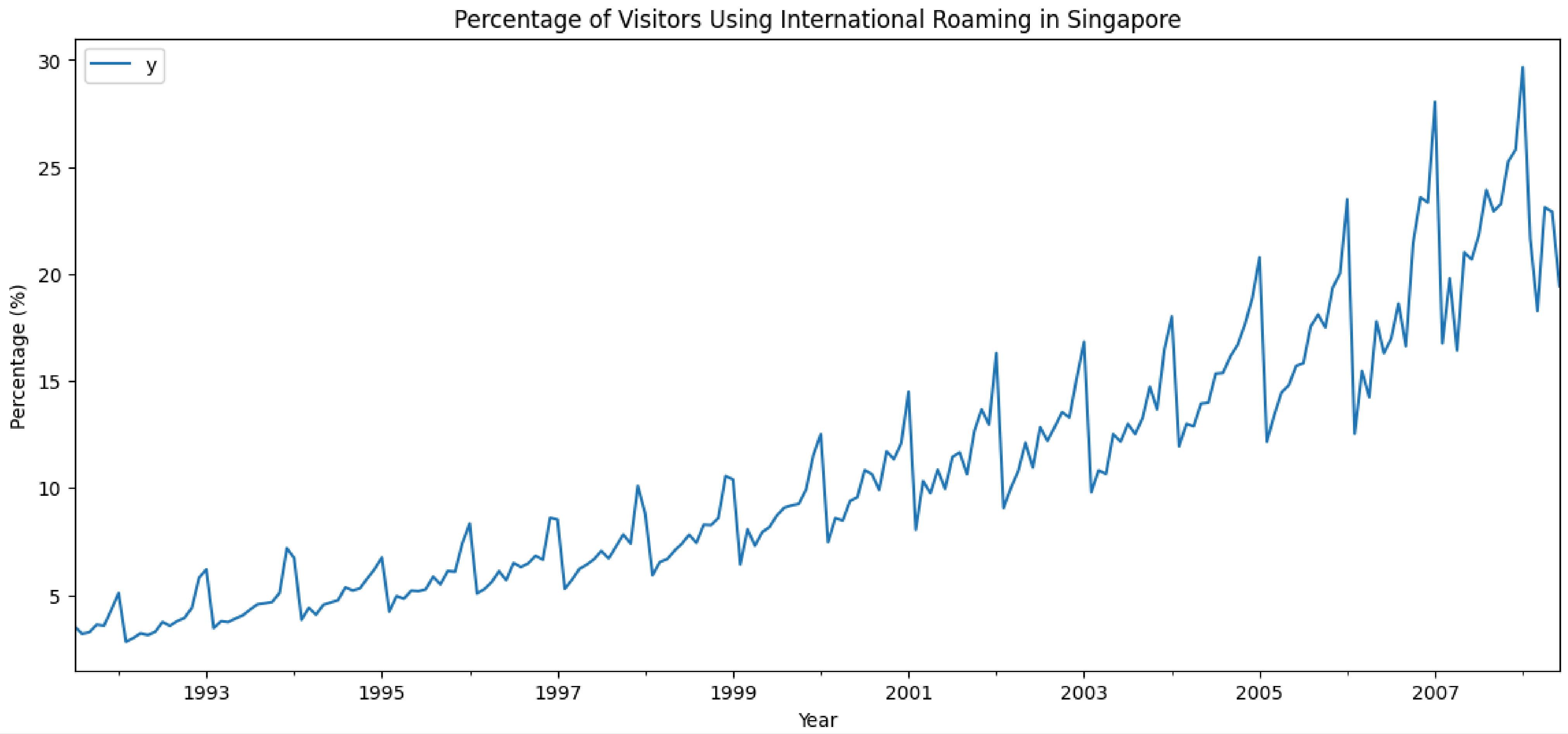
```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204 entries, 0 to 203
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   ds      204 non-null    object
 1   y       204 non-null    float64
dtypes: float64(1), object(1)
memory usage: 3.3+ KB
None

Basic Statistics:
                y
count  204.000000
mean    10.694430
std      5.956998
min      2.814520
25%      5.844095
50%      9.319345
75%     14.289964
max     29.665356

Missing Values:
ds    0
y     0
dtype: int64
```

```
# Convert the 'ds' column to datetime format
df['ds'] = pd.to_datetime(df['ds'])

# Set 'ds' as the index
df.set_index('ds', inplace=True)

# Display the first few rows of the dataset
df.head()
```

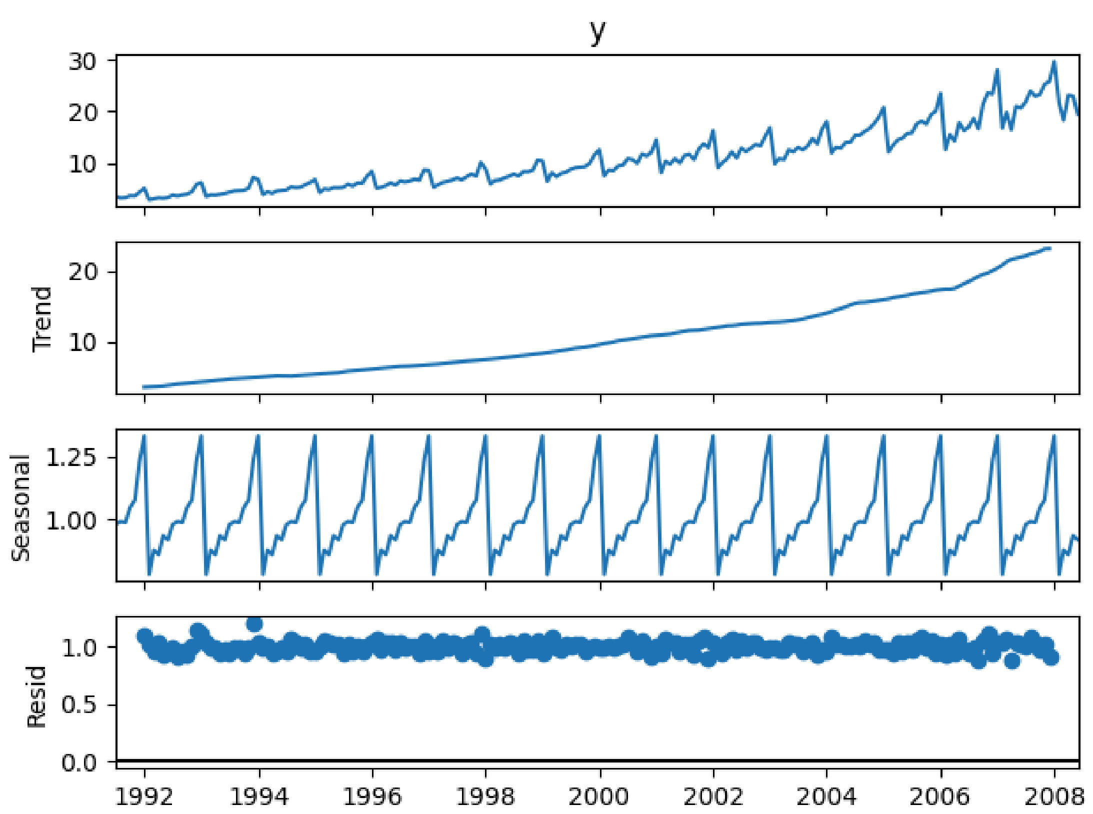
ds	y
1991-07-01	3.526591
1991-08-01	3.180891
1991-09-01	3.252221
1991-10-01	3.611003
1991-11-01	3.565869

Next steps: Generate code with df   View recommended plots   New interactive sheet

```
# 1. Plotting the Time Series
df.plot(figsize=(14, 6))
plt.title('Percentage of Visitors Using International Roaming in Singapore')
plt.xlabel('Year')
plt.ylabel('Percentage (%)')
plt.show()
```



```
# 2. Decomposing the Time Series
result = seasonal_decompose(df['y'], model='multiplicative', period=12)
result.plot()
plt.show()
```



Task 2. **Data Preprocessing**

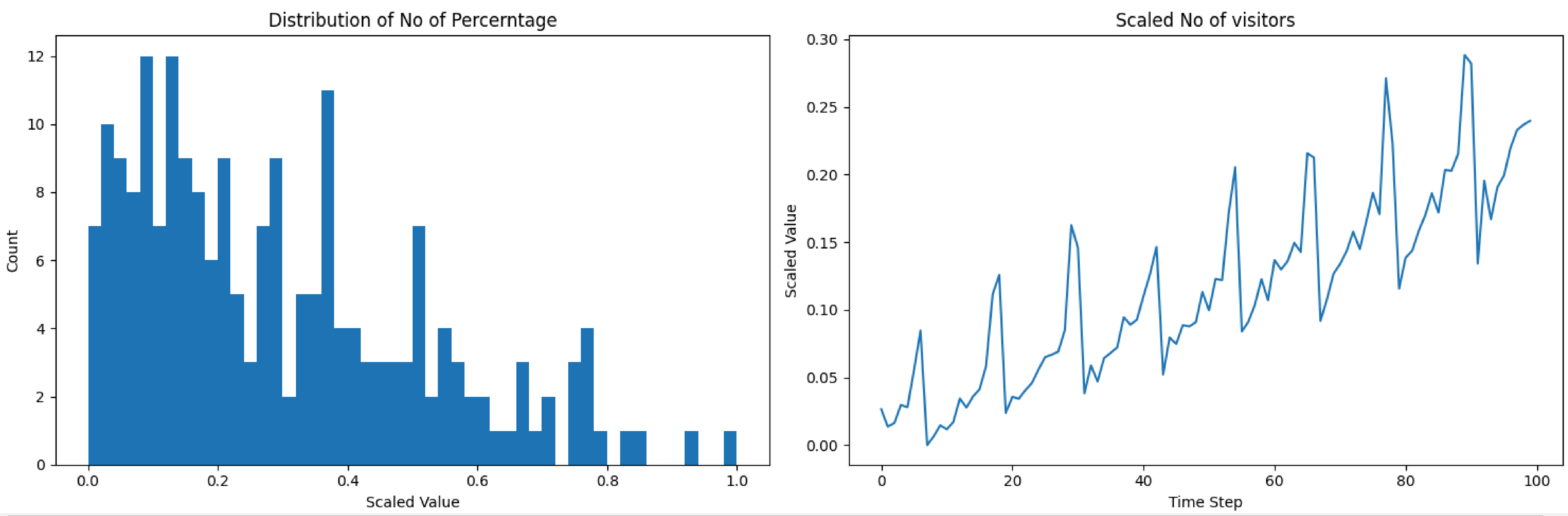
```
# Create sequences for RNN
def create_sequences(data, seq_length=12):
    """Create sequences for RNN input"""
    X, y = [], []
    data_array = data.values

    for i in range(len(data_array) - seq_length):
        X.append(data_array[i:(i + seq_length)])
        y.append(data_array[i + seq_length, 0])  # Signal Strength is first column

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

```
# Process the data
processed_data = df

# Scale the features
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(processed_data)
scaled_df = pd.DataFrame(scaled_data, columns=processed_data.columns)

# Create sequences
seq_length = 10  # We'll predict based on 10 previous measurements
X, y = create_sequences(scaled_df, seq_length)
```

```
# Print shapes and basic statistics
print("\nProcessed data shape:", processed_data.shape)
print("Sequence data shape (X):", X.shape)
print("Target data shape (y):", y.shape)
```

```
Processed data shape: (204, 1)
Sequence data shape (X): (194, 10, 1)
Target data shape (y): (194,)
```

```
# Plot scaled features distribution
plt.figure(figsize=(15, 5))
plt.subplot(1, 2, 1)
plt.hist(scaled_df['y'], bins=50)
plt.title('Distribution of No of Percerntage')
plt.xlabel('Scaled Value')
plt.ylabel('Count')

plt.subplot(1, 2, 2)
plt.plot(scaled_df['y'].iloc[:100])
plt.title('Scaled No of visitors')
plt.xlabel('Time Step')
plt.ylabel('Scaled Value')
plt.tight_layout()
plt.show()
```




```
# Display sample of the sequence data
print("\nSample sequence (first 5 time steps of first sequence):")
print(X[0][:5])
```

```
Sample sequence (first 5 time steps of first sequence):
[[0.02651951]
 [0.01364468]
 [0.01630121]
 [0.02966325]
 [0.02798233]]
```

Task 3: Basic RNN Implementation

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```
# 3.1 Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, shuffle=False
)

print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
```

```
Training set shape: (155, 10, 1)
Test set shape: (39, 10, 1)
```

```
# 3.2 Create and compile the basic RNN model
def create_simple_rnn():
    model = tf.keras.Sequential([
        # Input layer
        tf.keras.layers.SimpleRNN(32, input_shape=(10, 1), activation='tanh',
                                  return_sequences=True),
        tf.keras.layers.Dropout(0.2),

        # Second RNN layer
        tf.keras.layers.SimpleRNN(16, activation='tanh'),
        tf.keras.layers.Dropout(0.2),

        # Output layer
        tf.keras.layers.Dense(1)
    ])

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

    return model
```

```
# Create model
rnn_model = create_simple_rnn()
print("\nModel Summary:")
rnn_model.summary()
```

```
Model Summary:
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` objec
  super().__init__(**kwargs)
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
simple_rnn_6 (SimpleRNN)	(None, 10, 32)	1,088
dropout_8 (Dropout)	(None, 10, 32)	0
simple_rnn_7 (SimpleRNN)	(None, 16)	784
dropout_9 (Dropout)	(None, 16)	0
dense_5 (Dense)	(None, 1)	17

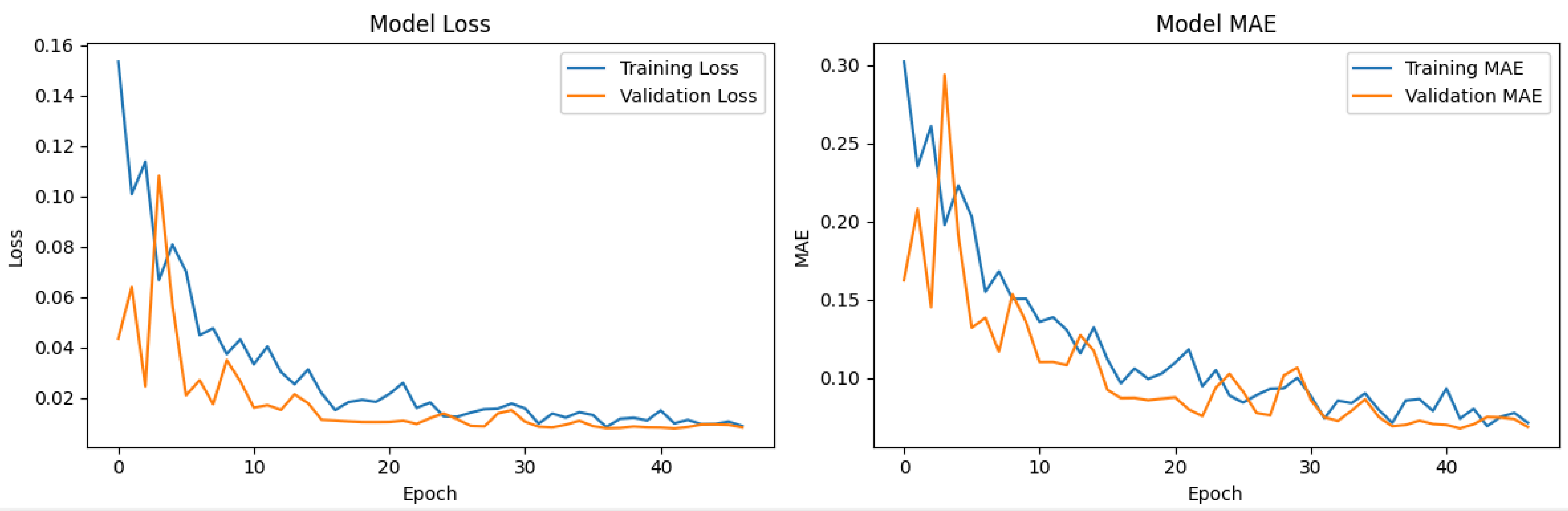
Total params: 1,889 (7.38 KB)
Trainable params: 1,889 (7.38 KB)
Non-trainable params: 0 (0.00 B)

```
# 3.3 Train the model
history = rnn_model.fit(
    X_train, y_train,
    epochs=50,
```

```
    batch_size=32,
    validation_split=0.2,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=5,
            restore_best_weights=True
        )
    ],
    verbose=1
)
```

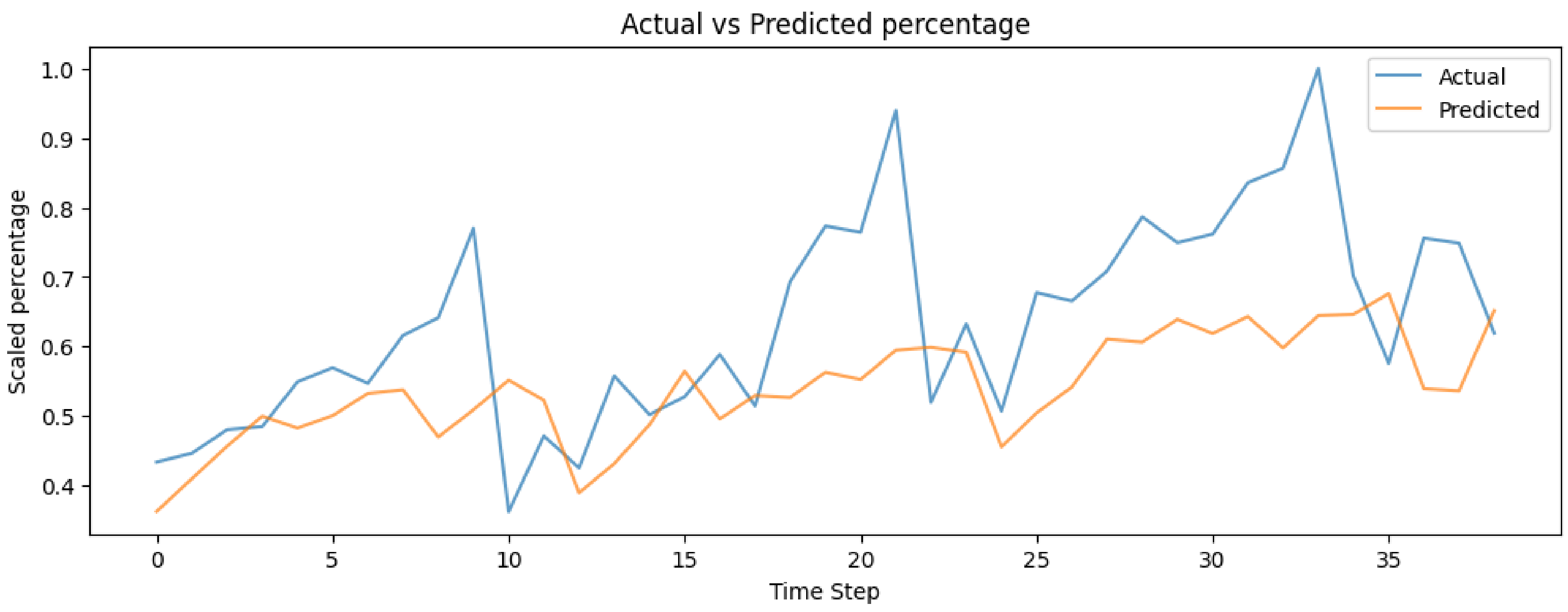
```
Epoch 1/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 131ms/step - loss: 0.1687 - mae: 0.3153 - val_loss: 0.0434 - val_mae: 0.1624
Epoch 2/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.1231 - mae: 0.2564 - val_loss: 0.0640 - val_mae: 0.2080
Epoch 3/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.1205 - mae: 0.2710 - val_loss: 0.0244 - val_mae: 0.1449
Epoch 4/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0719 - mae: 0.2070 - val_loss: 0.1080 - val_mae: 0.2937
Epoch 5/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0825 - mae: 0.2235 - val_loss: 0.0568 - val_mae: 0.1911
Epoch 6/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0606 - mae: 0.1862 - val_loss: 0.0208 - val_mae: 0.1319
Epoch 7/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0430 - mae: 0.1542 - val_loss: 0.0269 - val_mae: 0.1383
Epoch 8/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0484 - mae: 0.1734 - val_loss: 0.0174 - val_mae: 0.1167
Epoch 9/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0386 - mae: 0.1501 - val_loss: 0.0348 - val_mae: 0.1533
Epoch 10/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0390 - mae: 0.1463 - val_loss: 0.0265 - val_mae: 0.1355
Epoch 11/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - loss: 0.0366 - mae: 0.1456 - val_loss: 0.0159 - val_mae: 0.1099
Epoch 12/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0384 - mae: 0.1394 - val_loss: 0.0169 - val_mae: 0.1100
Epoch 13/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0296 - mae: 0.1287 - val_loss: 0.0151 - val_mae: 0.1080
Epoch 14/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0283 - mae: 0.1235 - val_loss: 0.0212 - val_mae: 0.1270
Epoch 15/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0294 - mae: 0.1279 - val_loss: 0.0177 - val_mae: 0.1171
Epoch 16/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0251 - mae: 0.1202 - val_loss: 0.0111 - val_mae: 0.0921
Epoch 17/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0153 - mae: 0.0979 - val_loss: 0.0108 - val_mae: 0.0868
Epoch 18/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0170 - mae: 0.1005 - val_loss: 0.0105 - val_mae: 0.0870
Epoch 19/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0187 - mae: 0.0950 - val_loss: 0.0102 - val_mae: 0.0856
Epoch 20/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0195 - mae: 0.1077 - val_loss: 0.0102 - val_mae: 0.0865
Epoch 21/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0208 - mae: 0.1075 - val_loss: 0.0102 - val_mae: 0.0873
Epoch 22/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0245 - mae: 0.1163 - val_loss: 0.0108 - val_mae: 0.0796
Epoch 23/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0148 - mae: 0.0895 - val_loss: 0.0095 - val_mae: 0.0753
Epoch 24/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0180 - mae: 0.1068 - val_loss: 0.0118 - val_mae: 0.0936
Epoch 25/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0129 - mae: 0.0887 - val_loss: 0.0136 - val_mae: 0.1024
Epoch 26/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0139 - mae: 0.0875 - val_loss: 0.0113 - val_mae: 0.0910
Epoch 27/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 0.0146 - mae: 0.0902 - val_loss: 0.0087 - val_mae: 0.0773
Epoch 28/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 0.0164 - mae: 0.0951 - val_loss: 0.0085 - val_mae: 0.0758
Epoch 29/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 0.0157 - mae: 0.0952 - val_loss: 0.0136 - val_mae: 0.1013
```

```
# 3.4 Plot training history
plt.figure(figsize=(12, 4))

# Loss plot
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# MAE plot
plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title('Model MAE')
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.legend()

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




```
# 3.5 Evaluate model performance
# Make predictions
train_predictions = rnn_model.predict(X_train)
test_predictions = rnn_model.predict(X_test)

# Plot actual vs predicted for test set
plt.figure(figsize=(12, 4))
plt.plot(y_test[:100], label='Actual', alpha=0.7)
plt.plot(test_predictions[:100], label='Predicted', alpha=0.7)
plt.title('Actual vs Predicted percentage')
plt.xlabel('Time Step')
plt.ylabel('Scaled percentage')
plt.legend()
plt.show()
```

```
WARNING:tensorflow:5 out of the last 10 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x7abc6832acb0> triggered tf.function retracing. Tracing is expensive and
1/5 ━━━━━━━━━━━━━━━━━━━━ 0s 220ms/stepWARNING:tensorflow:6 out of the last 14 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x7abc6832acb0> triggered tf.functio
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```



Task 4: LSTM Implementation

```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```
# Step 1: Data Preparation
def create_sequences(data, sequence_length):
    sequences = []
    targets = []
    for i in range(len(data) - sequence_length):
        seq = data[i:i + sequence_length]
        label = data[i + sequence_length]
        sequences.append(seq)
        targets.append(label)
    return np.array(sequences), np.array(targets)
```

```
# Load dataset
data = pd.read_csv('InternationRoaming_singapore.csv')
data.head()
```

	ds	y
0	1991-07-01	3.526591
1	1991-08-01	3.180891
2	1991-09-01	3.252221
3	1991-10-01	3.611003
4	1991-11-01	3.565869

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data['ds'] = pd.to_datetime(data['ds'])
values = data['y'].values.reshape(-1, 1)  # Percentage of visitors

# Normalize data
scaler = MinMaxScaler()
values = scaler.fit_transform(values)
```

```
# Create sequences
sequence_length = 12  # Using past 12 months for prediction
X, y = create_sequences(values, sequence_length)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Convert to PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32)
```

```
# Step 2: Model Design
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        output, _ = self.lstm(x)
        output = self.fc(output[:, -1, :])  # Only the last output
        return output
```

```
# Hyperparameters
input_size = 1
hidden_size = 64
num_layers = 2
output_size = 1
learning_rate = 0.001
epochs = 50

model = LSTMModel(input_size, hidden_size, num_layers, output_size)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

```
# X_train
```

```
# Step 3: Training
for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch + 1}/{epochs}], Loss: {loss.item():.4f}')
```

```
Epoch [10/50], Loss: 0.0512
Epoch [20/50], Loss: 0.0450
Epoch [30/50], Loss: 0.0394
Epoch [40/50], Loss: 0.0257
Epoch [50/50], Loss: 0.0093
```

```
# Step 4: Evaluation
model.eval()
with torch.no_grad():
    predictions = model(X_test)
    predictions = scaler.inverse_transform(predictions.numpy())
    actuals = scaler.inverse_transform(y_test.numpy().reshape(-1, 1))
```

```
# Step 5: Visualization
plt.figure(figsize=(10, 6))
plt.plot(actuals, label='Actual Values')
plt.plot(predictions, label='Predicted Values')
plt.legend()
plt.xlabel('Time Steps')
plt.ylabel('Percentage of Visitors')
plt.title('International Roaming Usage Forecasting')
plt.show()
```