LAB Assignment 5.1 - Univariate Time Series using LSTM

Objective - To forecast future values of a univariate time series using LSTM-based models

Name - Rohit Dahale

PRN - 202201070052

Dataset Link - https://www.kaggle.com/datasets/anirudhchauhan/retail-store-inventory-forecasting-dataset?resource=download

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

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 Dense, LSTM, Dropout,
Bidirectional
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

np.random.seed(42)

# Load dataset
df = pd.read_csv('/content/retail_store_inventory.csv')

# Display the basic information
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73100 entries, 0 to 73099
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Date                73100 non-null  object
 1   Store ID            73100 non-null  object
 2   Product ID          73100 non-null  object
 3   Category            73100 non-null  object
 4   Region              73100 non-null  object
 5   Inventory Level     73100 non-null  int64
 6   Units Sold          73100 non-null  int64
 7   Units Ordered       73100 non-null  int64
 8   Demand Forecast     73100 non-null  float64
 9   Price               73100 non-null  float64
```

```
 10  Discount            73100 non-null  int64
 11  Weather Condition   73100 non-null  object
 12  Holiday/Promotion   73100 non-null  int64
 13  Competitor Pricing  73100 non-null  float64
 14  Seasonality         73100 non-null  object
dtypes: float64(3), int64(5), object(7)
memory usage: 8.4+ MB
None
```

```
df.head(10)
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 73100,\n  \"fields\":
[\n    {\n      \"column\": \"Date\",\n      \"properties\": {\n
\"dtype\": \"object\",\n        \"num_unique_values\": 731,\n
\"samples\": [\n          \"2023-12-05\",\n          \"2022-02-03\",\n
\"2022-10-28\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Store ID\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"S002\",\n          \"S005\",\n          \"S003\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Product ID\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 20,\n        \"samples\": [\n
\"P0001\",\n          \"P0018\",\n          \"P0016\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Category\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
5,\n        \"samples\": [\n          \"Toys\",\n
\"Clothing\",\n          \"Electronics\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Region\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
4,\n        \"samples\": [\n          \"South\",\n          \"East\",\
n          \"North\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Inventory Level\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 129,\n        \"min\": 50,\n
\"max\": 500,\n        \"num_unique_values\": 451,\n
\"samples\": [\n          98,\n          375,\n          212\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Units Sold\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 108,\n        \"min\": 0,\n
\"max\": 499,\n        \"num_unique_values\": 498,\n
\"samples\": [\n          486,\n          176,\n          92\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Units Ordered\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 52,\n        \"min\": 20,\n
```

```
\"max\": 200,\n        \"num_unique_values\": 181,\n
\"samples\": [\n          144,\n          124,\n          87\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Demand Forecast\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 109.25407647039115,\n        \"min\": -
9.99,\n        \"max\": 518.55,\n        \"num_unique_values\":
31608,\n        \"samples\": [\n          276.33,\n          31.84,\n
141.35\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Price\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 26.021944625625636,\n        \"min\": 10.0,\n        \"max\":
100.0,\n        \"num_unique_values\": 8999,\n        \"samples\": [\n
61.69,\n          54.33,\n          33.89\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Discount\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 7,\n
\"min\": 0,\n        \"max\": 20,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          10,\n          15,\n          0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Weather Condition\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n
\"Sunny\",\n          \"Snowy\",\n          \"Rainy\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Holiday/Promotion\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Competitor Pricing\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 26.19140786909631,\n        \"min\":
5.03,\n        \"max\": 104.94,\n        \"num_unique_values\": 9751,\
n        \"samples\": [\n          96.16,\n          22.17\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Seasonality\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          \"Summer\",\n          \"Spring\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```

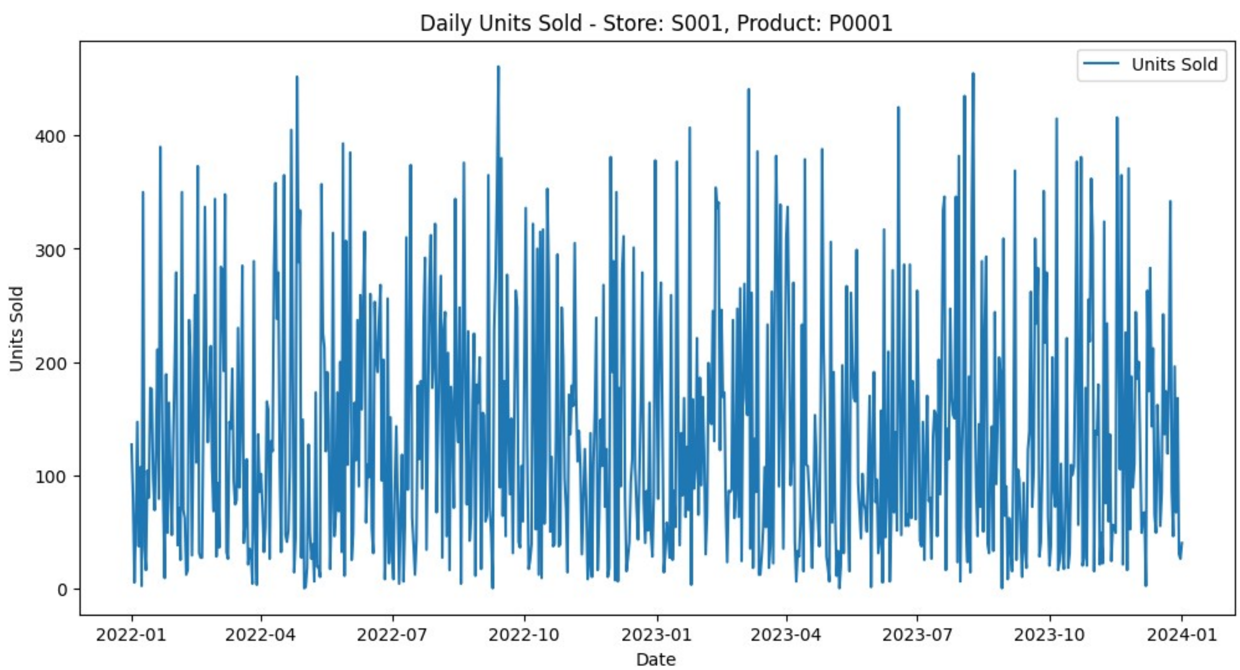

```
# Filter for a single store and product
store_id = "S001"
product_id = "P0001"

df_filtered = df[(df['Store ID'] == store_id) & (df['Product ID'] ==
product_id)].copy()
```

```python
# Convert date, sort and clean
df_filtered['Date'] = pd.to_datetime(df_filtered['Date'])
df_filtered.sort_values(by='Date', inplace=True)
df_filtered.dropna(inplace=True)

# Prepare time series
time_series = df_filtered[['Date', 'Units Sold']].copy()
time_series.set_index('Date', inplace=True)

# Plot
plt.figure(figsize=(12,6))
plt.plot(time_series.index, time_series['Units Sold'], label='Units
Sold')
plt.xlabel('Date')
plt.ylabel('Units Sold')
plt.title(f'Daily Units Sold - Store: {store_id}, Product:
{product_id}')
plt.legend()
plt.show()
```




```python
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(time_series)

scaled_data[:5]
```

```
array([[0.27548807],
       [0.17570499],
       [0.01084599],
```

```
       [0.12581345],
       [0.31887202]])
```

```
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(seq_length, len(data)):
        X.append(data[i-seq_length:i, 0])
        y.append(data[i, 0])
    return np.array(X), np.array(y)

sequence_length = 30  # 1 month of history
X, y = create_sequences(scaled_data, sequence_length)

X = X.reshape(X.shape[0], X.shape[1], 1)

split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

print(f'Train shape: {X_train.shape}, Test shape: {X_test.shape}')
```

```
Train shape: (560, 30, 1), Test shape: (141, 30, 1)
```

```
model = Sequential()
model.add(Bidirectional(LSTM(64, return_sequences=True),
input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(64)))
model.add(Dropout(0.2))
model.add(Dense(1))

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

# Callbacks
early_stop = EarlyStopping(monitor='val_loss', patience=5)
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss',
save_best_only=True)

model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/
bidirectional.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(**kwargs)
```

```
Model: "sequential"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━
━━━━━┓
```

```
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ bidirectional (Bidirectional)   │ (None, 30, 128)        │
33,792 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dropout (Dropout)               │ (None, 30, 128)        │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ bidirectional_1 (Bidirectional) │ (None, 128)            │
98,816 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dropout_1 (Dropout)             │ (None, 128)            │
0 │
├─────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dense (Dense)                   │ (None, 1)              │
129 │
└─────────────────────────────────┴────────────────────────┴──────────
─────┘

 Total params: 132,737 (518.50 KB)

 Trainable params: 132,737 (518.50 KB)

 Non-trainable params: 0 (0.00 B)
```

```
history = model.fit(
    X_train, y_train,
    epochs=20,
    batch_size=32,
    validation_data=(X_test, y_test),
    callbacks=[early_stop, checkpoint]
)
```

```
Epoch 1/20
17/18 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - loss: 0.0852

WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.

 18/18 ━━━━━━━━━━━━━━━━━━━━ 10s 113ms/step - loss: 0.0835 - val_loss:
0.0547
Epoch 2/20
```

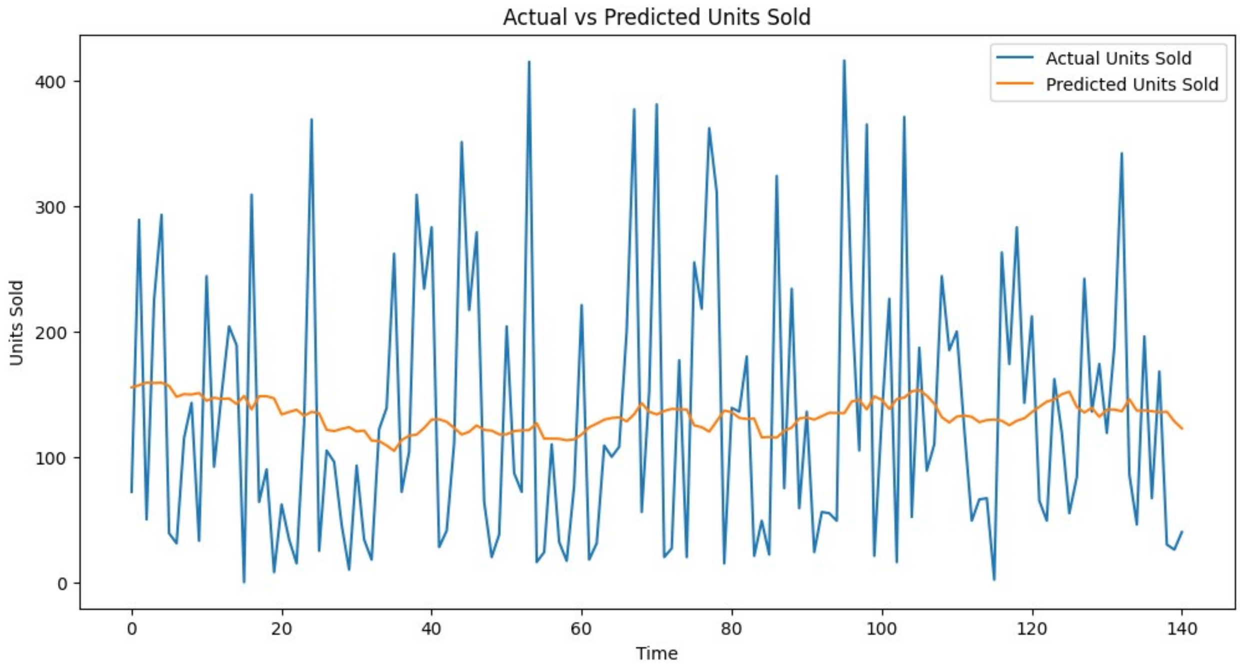
```
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 76ms/step - loss: 0.0579 - val_loss:
0.0550
Epoch 3/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 2s 64ms/step - loss: 0.0623 - val_loss:
0.0549
Epoch 4/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - loss: 0.0572 - val_loss:
0.0570
Epoch 5/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - loss: 0.0555 - val_loss:
0.0564
Epoch 6/20
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - loss: 0.0564 - val_loss:
0.0583
```

```
model.load_weights('best_model.h5')

predicted = model.predict(X_test)
predicted_values = scaler.inverse_transform(predicted)
actual_values = scaler.inverse_transform(y_test.reshape(-1, 1))
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 354ms/step
```

```
plt.figure(figsize=(12,6))
plt.plot(actual_values, label='Actual Units Sold')
plt.plot(predicted_values, label='Predicted Units Sold')
plt.title('Actual vs Predicted Units Sold')
plt.xlabel('Time')
plt.ylabel('Units Sold')
plt.legend()
plt.show()
```

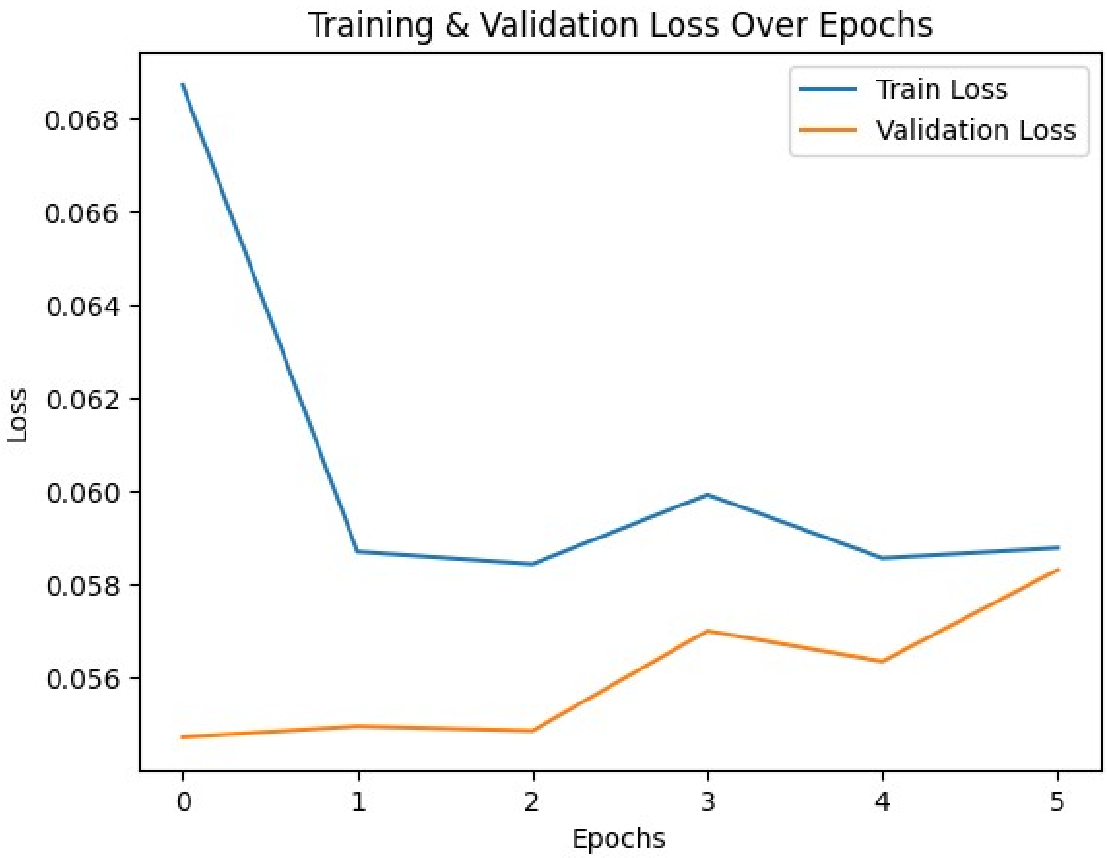



```
rmse = np.sqrt(mean_squared_error(actual_values, predicted_values))
mae = mean_absolute_error(actual_values, predicted_values)

print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
print(f'Mean Absolute Error (MAE): {mae:.2f}')
```

```
Root Mean Squared Error (RMSE): 107.85
Mean Absolute Error (MAE): 88.95
```

```
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training & Validation Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Training & Validation Loss Over Epochs
Train Loss
Validation Loss
0.068
0.066
0.064
0.062
0.060
0.058
0.056
Loss
0
1
2
3
4
5
Epochs