```
import pandas as pd

# Load the dataset
file_path = 'kamrup_rice_cleaned_data.csv'
rice_data = pd.read_csv(file_path)

# Display the first few rows of the dataset
rice_data.head(), rice_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23 entries, 0 to 22
Data columns (total 2 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Year                   23 non-null     object 
 1   Yield (Tonnes/Hectare)  23 non-null     float64
dtypes: float64(1), object(1)
memory usage: 496.0+ bytes
(         Year  Yield (Tonnes/Hectare)
 0  1997-01-01                0.980326
 1  1998-01-01                1.295593
 2  1999-01-01                1.419327
 3  2000-01-01                1.505253
 4  2001-01-01                1.536712,
 None)
```

```
# Data Preprocessing

# Convert the "Year" column to datetime format
rice_data['Year'] = pd.to_datetime(rice_data['Year'])

# Set the "Year" column as the index
rice_data.set_index('Year', inplace=True)

# Normalize the yield data
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
rice_data['Yield (Tonnes/Hectare)'] = scaler.fit_transform(rice_data[['Yield (Tonnes/Hectare)']])

# Display the first few rows of the preprocessed data
rice_data.head()
```

| | Yield (Tonnes/Hectare) |
|---|---|
| **Year** | |
| **1997-01-01** | 0.000000 |
| **1998-01-01** | 0.189116 |
| **1999-01-01** | 0.263338 |
| **2000-01-01** | 0.314881 |
| **2001-01-01** | 0.333752 |

```
import numpy as np

# Create sequences
def create_sequences(data, seq_length):
    xs = []
    ys = []
    for i in range(len(data)-seq_length):
        x = data[i:i+seq_length]
        y = data[i+seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

# Define sequence length
seq_length = 3

# Convert the dataframe to a numpy array
data = rice_data['Yield (Tonnes/Hectare)'].values

# Create sequences
X, y = create_sequences(data, seq_length)

# Split the data into training and testing sets (70% train, 30% test)
split_idx = int(0.8 * len(X))
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((16, 3), (4, 3), (16,), (4,))
```

```
from keras.models import Sequential
from keras.layers import LSTM, Dense

# Define the LSTM model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(seq_length, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

# Reshape the data to fit the model's input shape
X_train_reshaped = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_reshaped = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Train the model
history = model.fit(X_train_reshaped, y_train, epochs=200, validation_data=(X_test_reshaped, y_test), verbose=1)

# Summarize the model
model.summary()
```

```
Epoch 195/200
1/1 [==============================] - 0s 37ms/step - loss: 0.0089 - val_loss: 0.0054
Epoch 196/200
1/1 [==============================] - 0s 39ms/step - loss: 0.0089 - val_loss: 0.0055
Epoch 197/200
1/1 [==============================] - 0s 40ms/step - loss: 0.0089 - val_loss: 0.0056
Epoch 198/200
1/1 [==============================] - 0s 42ms/step - loss: 0.0089 - val_loss: 0.0057
Epoch 199/200
1/1 [==============================] - 0s 61ms/step - loss: 0.0088 - val_loss: 0.0058
Epoch 200/200
1/1 [==============================] - 0s 54ms/step - loss: 0.0088 - val_loss: 0.0059
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50)                10400

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 10451 (40.82 KB)
Trainable params: 10451 (40.82 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
y_pred = model.predict(X_test_reshaped)
```

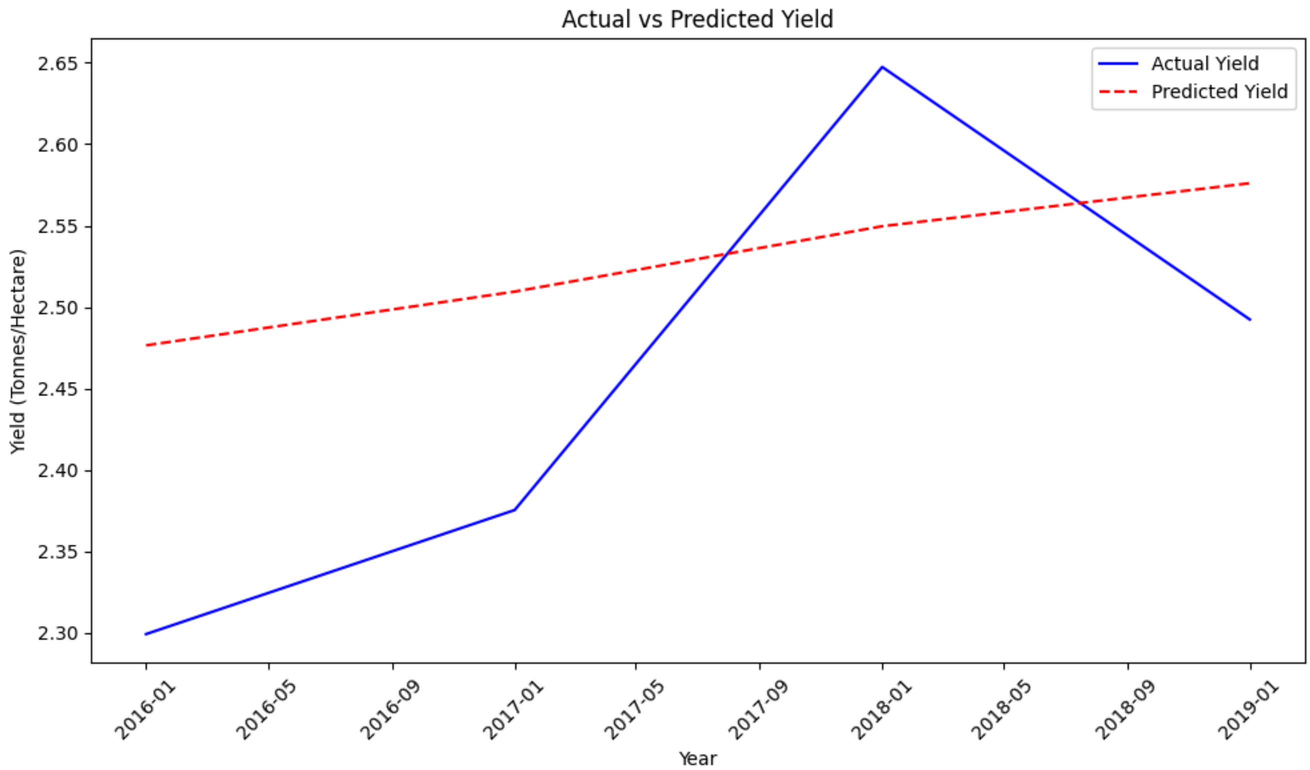
```
1/1 [==============================] - 0s 336ms/step
```

```
y_pred_inverse = scaler.inverse_transform(y_pred)
y_test_inverse = scaler.inverse_transform(y_test.reshape(-1, 1))
```

```
from sklearn.metrics import mean_squared_error

mse = mean_squared_error(y_test_inverse, y_pred_inverse)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 0.016492609036593355
```

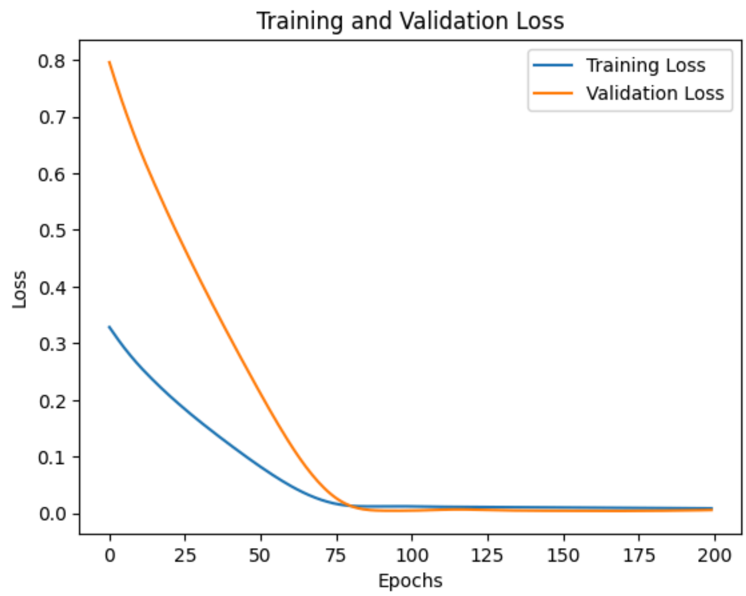
```
# Get the years corresponding to the test dataset
import matplotlib.pyplot as plt
test_years = rice_data.index[split_idx+seq_length:]

# Plot the results based on years
plt.figure(figsize=(10, 6))
plt.plot(test_years, y_test_inverse, color='blue', label='Actual Yield')
plt.plot(test_years, y_pred_inverse, color='red', linestyle='dashed', label='Predicted Yield')
plt.xlabel('Year')
plt.ylabel('Yield (Tonnes/Hectare)')
plt.title('Actual vs Predicted Yield')
plt.legend()
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()
```




```
# Save the model
model.save('rice_yield_lstm_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an
  saving_api.save_model(
```

```
# Plot training history
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50)                10400

 dense (Dense)               (None, 1)                 51
```

```
=================================================================
Total params: 10451 (40.82 KB)
Trainable params: 10451 (40.82 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

## Transfer Learning Nagaon

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from keras.models import load_model
import matplotlib.pyplot as plt
```

```
# Load the saved model
saved_model_path = 'rice_yield_lstm_model.h5'
model1 = load_model(saved_model_path)
```

```
# Load your new dataset
new_file_path = 'Nagaon_rice_cleaned_data.csv'
new_data = pd.read_csv(new_file_path)
```

```
# Data Preprocessing
new_data['Year'] = pd.to_datetime(new_data['Year'])
new_data.set_index('Year', inplace=True)
scaler = MinMaxScaler(feature_range=(0, 1))
new_data['Yield (Tonnes/Hectare)'] = scaler.fit_transform(new_data[['Yield (Tonnes/Hectare)']])
```

```
new_data
```

| | Yield (Tonnes/Hectare) |
|---|---|
| **Year** | |
| **1997-01-01** | 0.000000 |
| **1998-01-01** | 0.367798 |
| **1999-01-01** | 0.534952 |
| **2000-01-01** | 0.500367 |
| **2001-01-01** | 0.494395 |
| **2002-01-01** | 0.446928 |
| **2003-01-01** | 0.500155 |
| **2004-01-01** | 0.473107 |
| **2005-01-01** | 0.363544 |
| **2006-01-01** | 0.376288 |
| **2007-01-01** | 0.402433 |
| **2008-01-01** | 0.378269 |
| **2009-01-01** | 0.697636 |
| **2010-01-01** | 0.580383 |
| **2011-01-01** | 0.430779 |
| **2012-01-01** | 0.682824 |
| **2013-01-01** | 0.778547 |
| **2014-01-01** | 0.658777 |
| **2015-01-01** | 0.835853 |
| **2016-01-01** | 0.857179 |
| **2017-01-01** | 0.962274 |
| **2018-01-01** | 1.000000 |
| **2019-01-01** | 0.775903 |

```
# Create sequences
def create_sequences(data, seq_length):
    xs = []
    ys = []
    for i in range(len(data)-seq_length):
        x = data[i:i+seq_length]
        y = data[i+seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

seq_length = 3
data = new_data['Yield (Tonnes/Hectare)'].values
X, y = create_sequences(data, seq_length)

split_idx = int(0.7 * len(X))
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]
```

```
# Reshape the data
X_train_reshaped = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_reshaped = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

```
# Continue training the model (fine-tuning)
history = model1.fit(X_train_reshaped, y_train, epochs=30, validation_data=(X_test_reshaped, y_test), verbose=1)
```

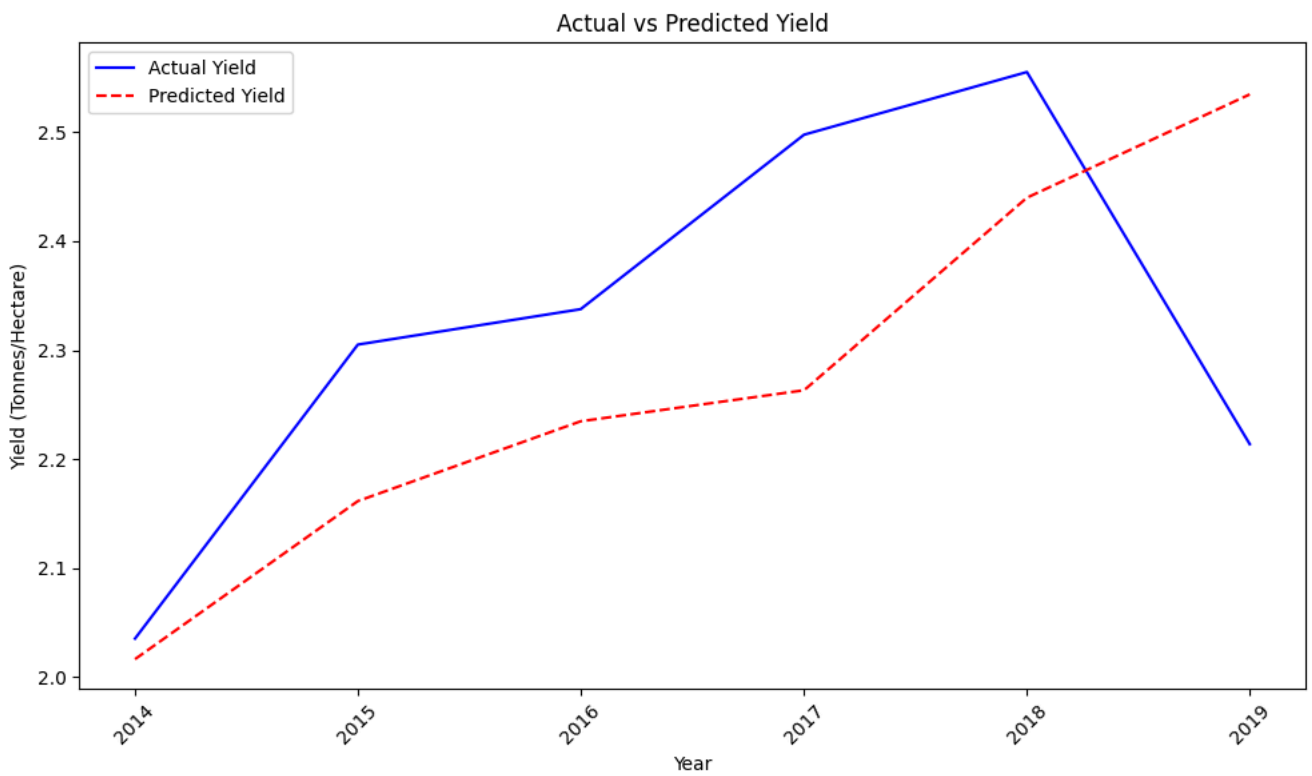
```
Epoch 2/30
1/1 [==============================] - 0s 36ms/step - loss: 0.0161 - val_loss: 0.0142
Epoch 3/30
1/1 [==============================] - 0s 36ms/step - loss: 0.0159 - val_loss: 0.0141
Epoch 4/30
1/1 [==============================] - 0s 36ms/step - loss: 0.0156 - val_loss: 0.0141
Epoch 5/30
1/1 [==============================] - 0s 36ms/step - loss: 0.0154 - val_loss: 0.0141
Epoch 6/30
1/1 [==============================] - 0s 42ms/step - loss: 0.0152 - val_loss: 0.0142
Epoch 7/30
1/1 [==============================] - 0s 35ms/step - loss: 0.0150 - val_loss: 0.0145
Epoch 8/30
1/1 [==============================] - 0s 35ms/step - loss: 0.0149 - val_loss: 0.0148
Epoch 9/30
1/1 [==============================] - 0s 37ms/step - loss: 0.0148 - val_loss: 0.0152
Epoch 10/30
1/1 [==============================] - 0s 36ms/step - loss: 0.0148 - val_loss: 0.0156
Epoch 11/30
1/1 [==============================] - 0s 39ms/step - loss: 0.0149 - val_loss: 0.0161
Epoch 12/30
1/1 [==============================] - 0s 38ms/step - loss: 0.0149 - val_loss: 0.0164
Epoch 13/30
1/1 [==============================] - 0s 38ms/step - loss: 0.0150 - val_loss: 0.0167
Epoch 14/30
1/1 [==============================] - 0s 44ms/step - loss: 0.0150 - val_loss: 0.0169
Epoch 15/30
1/1 [==============================] - 0s 37ms/step - loss: 0.0150 - val_loss: 0.0169
Epoch 16/30
1/1 [==============================] - 0s 37ms/step - loss: 0.0150 - val_loss: 0.0168
Epoch 17/30
1/1 [==============================] - 0s 37ms/step - loss: 0.0150 - val_loss: 0.0167
Epoch 18/30
1/1 [==============================] - 0s 38ms/step - loss: 0.0150 - val_loss: 0.0165
Epoch 19/30
1/1 [==============================] - 0s 35ms/step - loss: 0.0149 - val_loss: 0.0162
Epoch 20/30
1/1 [==============================] - 0s 43ms/step - loss: 0.0149 - val_loss: 0.0160
Epoch 21/30
1/1 [==============================] - 0s 36ms/step - loss: 0.0148 - val_loss: 0.0157
Epoch 22/30
1/1 [==============================] - 0s 39ms/step - loss: 0.0148 - val_loss: 0.0155
Epoch 23/30
1/1 [==============================] - 0s 38ms/step - loss: 0.0148 - val_loss: 0.0152
Epoch 24/30
1/1 [==============================] - 0s 37ms/step - loss: 0.0147 - val_loss: 0.0150
Epoch 25/30
1/1 [==============================] - 0s 38ms/step - loss: 0.0147 - val_loss: 0.0149
Epoch 26/30
1/1 [==============================] - 0s 34ms/step - loss: 0.0147 - val_loss: 0.0148
Epoch 27/30
1/1 [==============================] - 0s 41ms/step - loss: 0.0147 - val_loss: 0.0147
Epoch 28/30
1/1 [==============================] - 0s 35ms/step - loss: 0.0147 - val_loss: 0.0146
Epoch 29/30
1/1 [==============================] - 0s 35ms/step - loss: 0.0147 - val_loss: 0.0146
Epoch 30/30
1/1 [==============================] - 0s 36ms/step - loss: 0.0147 - val_loss: 0.0146
```

```
# Make predictions
y_pred = model1.predict(X_test_reshaped)
```

```
1/1 [==============================] - 0s 189ms/step
```

```
# Inverse transform the predictions and the true values
y_pred_inverse = scaler.inverse_transform(y_pred)
y_test_inverse = scaler.inverse_transform(y_test.reshape(-1, 1))
```

```
# Plot the results based on years
test_years = new_data.index[split_idx+seq_length:]

plt.figure(figsize=(10, 6))
plt.plot(test_years, y_test_inverse, color='blue', label='Actual Yield')
plt.plot(test_years, y_pred_inverse, color='red', linestyle='dashed', label='Predicted Yield')
plt.xlabel('Year')
plt.ylabel('Yield (Tonnes/Hectare)')
plt.title('Actual vs Predicted Yield')
plt.legend()
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()
```



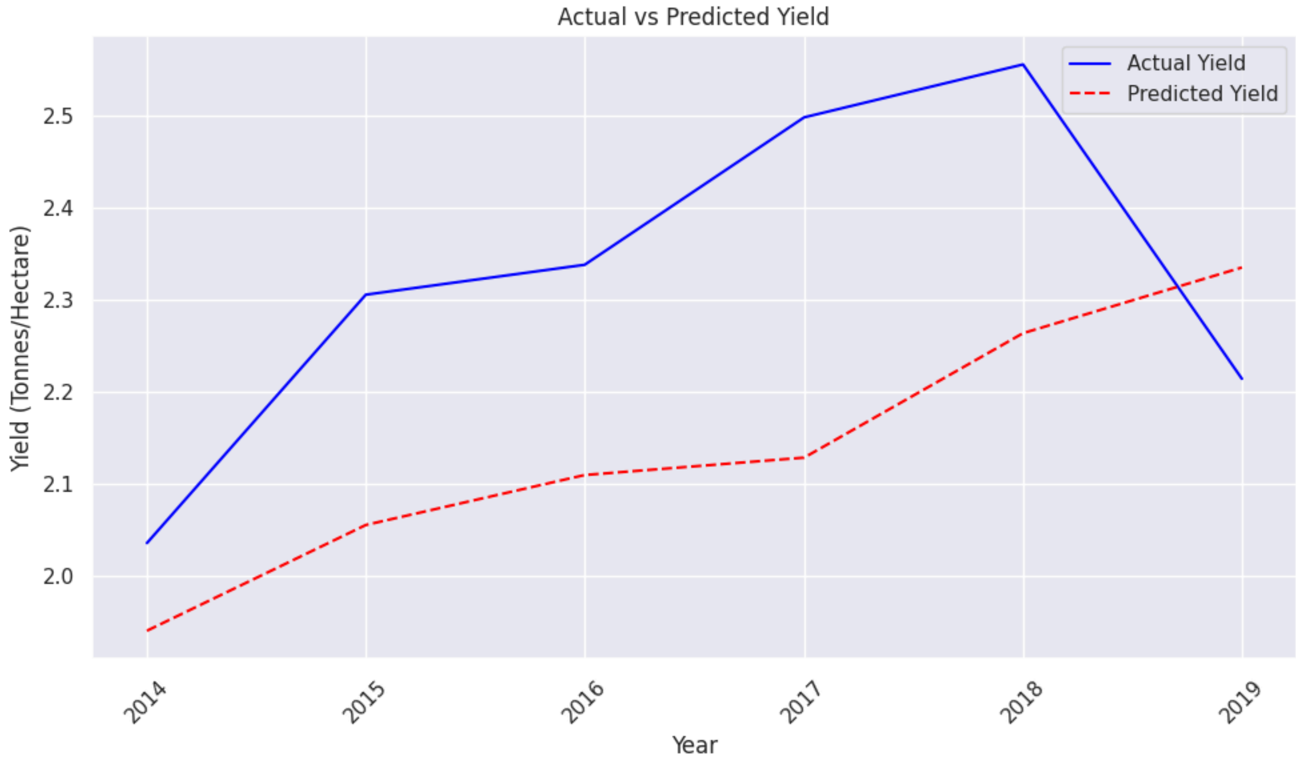

## Transfer learning 2

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from keras.models import load_model, Sequential
from keras.layers import LSTM, Dense
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

# Load the saved model
saved_model_path = 'rice_yield_lstm_model.h5'
base_model = load_model(saved_model_path)

# Freeze all layers except the last one
for layer in base_model.layers[:]:
    layer.trainable = True

# Create a new model and add layers on top of the base model
model_2 = Sequential()
model_2.add(base_model)
model_2.add(Dense(200, activation='relu'))
model_2.add(Dense(200, activation='relu'))
model_2.add(Dense(1))

# Compile the new model
model_2.compile(optimizer='adam', loss='mse')

# Load your new dataset
new_file_path = 'Nagaon_rice_cleaned_data.csv'  # Replace with your new dataset file path
new_data = pd.read_csv(new_file_path)

# Data Preprocessing
new_data['Year'] = pd.to_datetime(new_data['Year'])
new_data.set_index('Year', inplace=True)
scaler = MinMaxScaler(feature_range=(0, 1))
new_data['Yield (Tonnes/Hectare)'] = scaler.fit_transform(new_data[['Yield (Tonnes/Hectare)']])

# Create sequences
def create_sequences(data, seq_length):
    xs = []
    ys = []
    for i in range(len(data)-seq_length):
        x = data[i:i+seq_length]
        y = data[i+seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

seq_length = 3
data = new_data['Yield (Tonnes/Hectare)'].values
X, y = create_sequences(data, seq_length)

split_idx = int(0.7 * len(X))
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]

# Reshape the data
X_train_reshaped = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_reshaped = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Train the new model (fine-tuning)
history = model_2.fit(X_train_reshaped, y_train, epochs=50, validation_data=(X_test_reshaped, y_test), verbose=1)

# Make predictions
y_pred = model_2.predict(X_test_reshaped)

# Inverse transform the predictions and the true values
y_pred_inverse = scaler.inverse_transform(y_pred)
y_test_inverse = scaler.inverse_transform(y_test.reshape(-1, 1))

# Plot the results based on years
test_years = new_data.index[split_idx+seq_length:]

plt.figure(figsize=(10, 6))
plt.plot(test_years, y_test_inverse, color='blue', label='Actual Yield')
plt.plot(test_years, y_pred_inverse, color='red', linestyle='dashed', label='Predicted Yield')
plt.xlabel('Year')
plt.ylabel('Yield (Tonnes/Hectare)')
plt.title('Actual vs Predicted Yield')
plt.legend()
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent clipping of labels
```

```
plt.show()
```

```
Epoch 1/50
1/1 [==============================] - 2s 2s/step - loss: 0.2345 - val_loss: 0.5049
Epoch 2/50
1/1 [==============================] - 0s 39ms/step - loss: 0.1838 - val_loss: 0.3947
Epoch 3/50
1/1 [==============================] - 0s 40ms/step - loss: 0.1402 - val_loss: 0.3002
Epoch 4/50
1/1 [==============================] - 0s 41ms/step - loss: 0.1040 - val_loss: 0.2202
Epoch 5/50
1/1 [==============================] - 0s 40ms/step - loss: 0.0744 - val_loss: 0.1532
Epoch 6/50
1/1 [==============================] - 0s 44ms/step - loss: 0.0507 - val_loss: 0.0994
Epoch 7/50
1/1 [==============================] - 0s 39ms/step - loss: 0.0328 - val_loss: 0.0578
Epoch 8/50
1/1 [==============================] - 0s 40ms/step - loss: 0.0208 - val_loss: 0.0297
Epoch 9/50
1/1 [==============================] - 0s 39ms/step - loss: 0.0150 - val_loss: 0.0152
Epoch 10/50
1/1 [==============================] - 0s 38ms/step - loss: 0.0152 - val_loss: 0.0124
Epoch 11/50
1/1 [==============================] - 0s 38ms/step - loss: 0.0202 - val_loss: 0.0168
Epoch 12/50
1/1 [==============================] - 0s 44ms/step - loss: 0.0273 - val_loss: 0.0225
Epoch 13/50
1/1 [==============================] - 0s 40ms/step - loss: 0.0331 - val_loss: 0.0255
Epoch 14/50
1/1 [==============================] - 0s 39ms/step - loss: 0.0358 - val_loss: 0.0247
Epoch 15/50
1/1 [==============================] - 0s 39ms/step - loss: 0.0351 - val_loss: 0.0211
Epoch 16/50
1/1 [==============================] - 0s 42ms/step - loss: 0.0318 - val_loss: 0.0167
Epoch 17/50
1/1 [==============================] - 0s 41ms/step - loss: 0.0272 - val_loss: 0.0134
Epoch 18/50
1/1 [==============================] - 0s 42ms/step - loss: 0.0226 - val_loss: 0.0122
Epoch 19/50
1/1 [==============================] - 0s 39ms/step - loss: 0.0187 - val_loss: 0.0137
Epoch 20/50
1/1 [==============================] - 0s 38ms/step - loss: 0.0160 - val_loss: 0.0176
Epoch 21/50
1/1 [==============================] - 0s 38ms/step - loss: 0.0146 - val_loss: 0.0233
Epoch 22/50
1/1 [==============================] - 0s 42ms/step - loss: 0.0144 - val_loss: 0.0299
Epoch 23/50
1/1 [==============================] - 0s 39ms/step - loss: 0.0150 - val_loss: 0.0365
Epoch 24/50
1/1 [==============================] - 0s 43ms/step - loss: 0.0160 - val_loss: 0.0424
Epoch 25/50
1/1 [==============================] - 0s 36ms/step - loss: 0.0171 - val_loss: 0.0471
Epoch 26/50
1/1 [==============================] - 0s 40ms/step - loss: 0.0181 - val_loss: 0.0501
Epoch 27/50
1/1 [==============================] - 0s 38ms/step - loss: 0.0188 - val_loss: 0.0515
Epoch 28/50
1/1 [==============================] - 0s 36ms/step - loss: 0.0191 - val_loss: 0.0510
Epoch 29/50
1/1 [==============================] - 0s 39ms/step - loss: 0.0189 - val_loss: 0.0490
Epoch 30/50
1/1 [==============================] - 0s 40ms/step - loss: 0.0184 - val_loss: 0.0458
Epoch 31/50
1/1 [==============================] - 0s 41ms/step - loss: 0.0177 - val_loss: 0.0418
Epoch 32/50
1/1 [==============================] - 0s 39ms/step - loss: 0.0168 - val_loss: 0.0373
Epoch 33/50
1/1 [==============================] - 0s 37ms/step - loss: 0.0159 - val_loss: 0.0327
Epoch 34/50
1/1 [==============================] - 0s 38ms/step - loss: 0.0152 - val_loss: 0.0283
Epoch 35/50
1/1 [==============================] - 0s 40ms/step - loss: 0.0146 - val_loss: 0.0245
Epoch 36/50
1/1 [==============================] - 0s 41ms/step - loss: 0.0144 - val_loss: 0.0214
Epoch 37/50
1/1 [==============================] - 0s 40ms/step - loss: 0.0143 - val_loss: 0.0189
Epoch 38/50
1/1 [==============================] - 0s 39ms/step - loss: 0.0145 - val_loss: 0.0171
Epoch 39/50
1/1 [==============================] - 0s 39ms/step - loss: 0.0148 - val_loss: 0.0160
Epoch 40/50
1/1 [==============================] - 0s 40ms/step - loss: 0.0151 - val_loss: 0.0153
Epoch 41/50
1/1 [==============================] - 0s 50ms/step - loss: 0.0154 - val_loss: 0.0151
Epoch 42/50
1/1 [==============================] - 0s 39ms/step - loss: 0.0155 - val_loss: 0.0152
Epoch 43/50
1/1 [==============================] - 0s 37ms/step - loss: 0.0154 - val_loss: 0.0157
Epoch 44/50
1/1 [==============================] - 0s 40ms/step - loss: 0.0152 - val_loss: 0.0165
Epoch 45/50
1/1 [==============================] - 0s 42ms/step - loss: 0.0150 - val_loss: 0.0176
```

```
Epoch 46/50
1/1 [==============================] - 0s 39ms/step - loss: 0.0147 - val_loss: 0.0190
Epoch 47/50
1/1 [==============================] - 0s 39ms/step - loss: 0.0145 - val_loss: 0.0207
Epoch 48/50
1/1 [==============================] - 0s 38ms/step - loss: 0.0144 - val_loss: 0.0224
Epoch 49/50
1/1 [==============================] - 0s 37ms/step - loss: 0.0143 - val_loss: 0.0242
Epoch 50/50
1/1 [==============================] - 0s 40ms/step - loss: 0.0143 - val_loss: 0.0259
1/1 [==============================] - 0s 200ms/step
```




```
model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 1)                 10451

 dense_1 (Dense)             (None, 200)               400

 dense_2 (Dense)             (None, 200)               40200

 dense_3 (Dense)             (None, 1)                 201

=================================================================
Total params: 51252 (200.20 KB)
Trainable params: 51252 (200.20 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50)                10400

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 10451 (40.82 KB)
Trainable params: 10451 (40.82 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from keras.models import load_model, Model, Sequential
from keras.layers import Dense, Input
import matplotlib.pyplot as plt

# Load the saved model
saved_model_path = 'rice_yield_lstm_model.h5'
base_model = load_model(saved_model_path)

# Remove the output layer of the base model
base_model_output = base_model.layers[-2].output

# Create a new model by adding new layers on top of the base model (excluding the last layer)
x = Dense(200, activation='relu')(base_model_output)
x = Dense(200, activation='relu')(x)
output = Dense(1)(x)
model_2 = Model(inputs=base_model.input, outputs=output)

# Freeze all layers in the base model except the second-to-last layer
for layer in base_model.layers[:-2]:
    layer.trainable = False

# Compile the new model
model_2.compile(optimizer='adam', loss='mse')

# Load your new dataset
new_file_path = 'Nagaon_rice_cleaned_data.csv'  # Replace with your new dataset file path
new_data = pd.read_csv(new_file_path)

# Data Preprocessing
new_data['Year'] = pd.to_datetime(new_data['Year'])
new_data.set_index('Year', inplace=True)
scaler = MinMaxScaler(feature_range=(0, 1))
new_data['Yield (Tonnes/Hectare)'] = scaler.fit_transform(new_data[['Yield (Tonnes/Hectare)']])

# Create sequences
def create_sequences(data, seq_length):
    xs = []
    ys = []
    for i in range(len(data)-seq_length):
        x = data[i:i+seq_length]
        y = data[i+seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

seq_length = 3
data = new_data['Yield (Tonnes/Hectare)'].values
X, y = create_sequences(data, seq_length)

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

# Reshape the data
X_train_reshaped = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_reshaped = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Train the new model (fine-tuning)
history = model_2.fit(X_train_reshaped, y_train, epochs=100, validation_data=(X_test_reshaped, y_test), verbose=1)

# Make predictions
y_pred = model_2.predict(X_test_reshaped)

# Inverse transform the predictions and the true values
y_pred_inverse = scaler.inverse_transform(y_pred)
y_test_inverse = scaler.inverse_transform(y_test.reshape(-1, 1))

# Plot the results based on years
test_years = new_data.index[split_idx+seq_length:]

plt.figure(figsize=(10, 6))
plt.plot(test_years, y_test_inverse, color='blue', label='Actual Yield')
plt.plot(test_years, y_pred_inverse, color='red', linestyle='dashed', label='Predicted Yield')
plt.xlabel('Year')
plt.ylabel('Yield (Tonnes/Hectare)')
plt.title('Actual vs Predicted Yield')
plt.legend()
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent clipping of labels
```

```
plt.show()
```

```
Epoch 1/100
1/1 [==============================] - 2s 2s/step - loss: 0.2602 - val_loss:
Epoch 2/100
1/1 [==============================] - 0s 67ms/step - loss: 0.1933 - val_loss
Epoch 3/100
1/1 [==============================] - 0s 62ms/step - loss: 0.1389 - val_loss
Epoch 4/100
1/1 [==============================] - 0s 82ms/step - loss: 0.0957 - val_loss
Epoch 5/100
1/1 [==============================] - 0s 85ms/step - loss: 0.0617 - val_loss
Epoch 6/100
1/1 [==============================] - 0s 79ms/step - loss: 0.0364 - val_loss
Epoch 7/100
1/1 [==============================] - 0s 87ms/step - loss: 0.0204 - val_loss
Epoch 8/100
1/1 [==============================] - 0s 84ms/step - loss: 0.0141 - val_loss
Epoch 9/100
1/1 [==============================] - 0s 74ms/step - loss: 0.0162 - val_loss
Epoch 10/100
1/1 [==============================] - 0s 77ms/step - loss: 0.0240 - val_loss
Epoch 11/100
1/1 [==============================] - 0s 70ms/step - loss: 0.0324 - val_loss
Epoch 12/100
1/1 [==============================] - 0s 61ms/step - loss: 0.0375 - val_loss
Epoch 13/100
1/1 [==============================] - 0s 96ms/step - loss: 0.0380 - val_loss
Epoch 14/100
1/1 [==============================] - 0s 80ms/step - loss: 0.0345 - val_loss
Epoch 15/100
1/1 [==============================] - 0s 72ms/step - loss: 0.0291 - val_loss
Epoch 16/100
1/1 [==============================] - 0s 68ms/step - loss: 0.0233 - val_loss
Epoch 17/100
1/1 [==============================] - 0s 65ms/step - loss: 0.0185 - val_loss
Epoch 18/100
1/1 [==============================] - 0s 91ms/step - loss: 0.0153 - val_loss
Epoch 19/100
1/1 [==============================] - 0s 61ms/step - loss: 0.0139 - val_loss
Epoch 20/100
1/1 [==============================] - 0s 76ms/step - loss: 0.0140 - val_loss
Epoch 21/100
1/1 [==============================] - 0s 79ms/step - loss: 0.0151 - val_loss
Epoch 22/100
1/1 [==============================] - 0s 79ms/step - loss: 0.0166 - val_loss
Epoch 23/100
1/1 [==============================] - 0s 74ms/step - loss: 0.0180 - val_loss
Epoch 24/100
1/1 [==============================] - 0s 59ms/step - loss: 0.0191 - val_loss
Epoch 25/100
1/1 [==============================] - 0s 79ms/step - loss: 0.0195 - val_loss
Epoch 26/100
1/1 [==============================] - 0s 68ms/step - loss: 0.0194 - val_loss
Epoch 27/100
1/1 [==============================] - 0s 69ms/step - loss: 0.0188 - val_loss
Epoch 28/100
1/1 [==============================] - 0s 71ms/step - loss: 0.0178 - val_loss
Epoch 29/100
```

```
model_2.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_input (InputLayer)     [(None, 3, 1)]            0

 lstm (LSTM)                 (None, 50)                10400

 dense_4 (Dense)             (None, 200)               10200

 dense_5 (Dense)             (None, 200)               40200

 dense_6 (Dense)             (None, 1)                 201

=================================================================
Total params: 61001 (238.29 KB)
Trainable params: 61001 (238.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model_2.save('rice_yield_lstm_model_2.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an
  saving_api.save_model(
```

```
# Compute MAE on the test set
mae = np.mean(np.abs(y_test_inverse - y_pred_inverse))
print(f'Mean Absolute Error on the test set: {mae:.4f}')
```

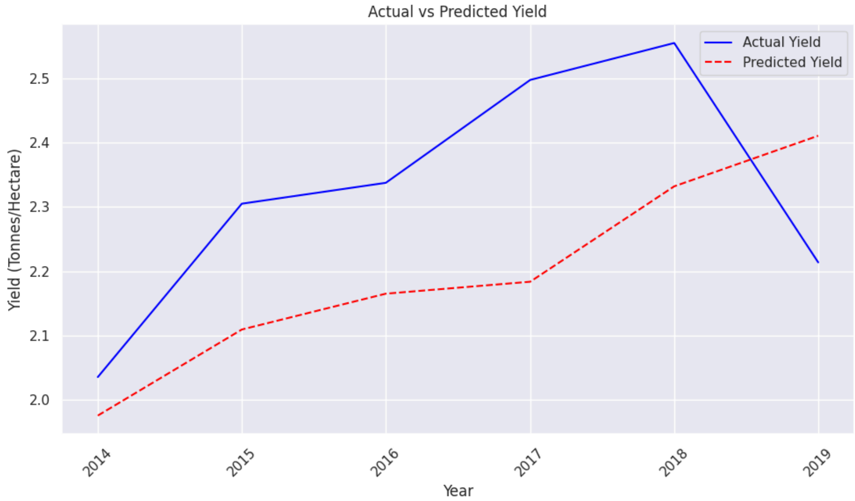
```
Mean Absolute Error on the test set: 0.2067
```

## Transfer Learning Nagaon Original

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from keras.models import load_model, Model
from keras.layers import Dense
import matplotlib.pyplot as plt
from keras.metrics import MeanAbsoluteError

# Load the saved model
saved_model_path = 'rice_yield_lstm_model.h5'
base_model = load_model(saved_model_path)

# Remove the output layer of the base model
base_model_output = base_model.layers[-2].output

# Create a new model by adding new layers on top of the base model (excluding the last layer)
x = Dense(200, activation='relu')(base_model_output)
x = Dense(200, activation='relu')(x)
output = Dense(1)(x)
model_2 = Model(inputs=base_model.input, outputs=output)

# Freeze all layers in the base model except the second-to-last layer
for layer in base_model.layers[:-2]:
    layer.trainable = False

# Compile the new model with MAE metric
model_2.compile(optimizer='adam', loss='mse', metrics=[MeanAbsoluteError()])

# Load your new dataset
new_file_path = 'Nagaon_rice_cleaned_data.csv'  # Replace with your new dataset file path
new_data = pd.read_csv(new_file_path)

# Data Preprocessing
new_data['Year'] = pd.to_datetime(new_data['Year'])
new_data.set_index('Year', inplace=True)
scaler = MinMaxScaler(feature_range=(0, 1))
new_data['Yield (Tonnes/Hectare)'] = scaler.fit_transform(new_data[['Yield (Tonnes/Hectare)']])

# Create sequences
def create_sequences(data, seq_length):
    xs = []
    ys = []
    for i in range(len(data)-seq_length):
        x = data[i:i+seq_length]
        y = data[i+seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

seq_length = 3
data = new_data['Yield (Tonnes/Hectare)'].values
X, y = create_sequences(data, seq_length)

split_idx = int(0.7 * len(X))
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]

# Reshape the data
X_train_reshaped = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_reshaped = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Train the new model (fine-tuning)
history = model_2.fit(X_train_reshaped, y_train, epochs=8, validation_data=(X_test_reshaped, y_test), verbose=1)

# Make predictions
y_pred = model_2.predict(X_test_reshaped)

# Inverse transform the predictions and the true values
y_pred_inverse = scaler.inverse_transform(y_pred)
y_test_inverse = scaler.inverse_transform(y_test.reshape(-1, 1))

# Compute MAE on the test set
mae = np.mean(np.abs(y_test_inverse - y_pred_inverse))
print(f'Mean Absolute Error on the test set: {mae:.4f}')

# Plot the results based on years
test_years = new_data.index[split_idx+seq_length:]

plt.figure(figsize=(10, 6))
plt.plot(test_years, y_test_inverse, color='blue', label='Actual Yield')
plt.plot(test_years, y_pred_inverse, color='red', linestyle='dashed', label='Predicted Yield')
plt.xlabel('Year')
```

```
plt.ylabel('Yield (Tonnes/Hectare)')
plt.title('Actual vs Predicted Yield')
plt.legend()
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()

# Plot training & validation loss and MAE
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 12))

# Loss plot
ax1.plot(history.history['loss'], label='Training Loss')
ax1.plot(history.history['val_loss'], label='Validation Loss')
ax1.set_title('Training and Validation Loss')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss (MSE)')
ax1.legend()

# MAE plot
ax2.plot(history.history['mean_absolute_error'], label='Training MAE')
ax2.plot(history.history['val_mean_absolute_error'], label='Validation MAE')
ax2.set_title('Training and Validation MAE')
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Mean Absolute Error')
ax2.legend()

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

```
Epoch 1/8
1/1 [==============================] - 3s 3s/step - loss: 0.2641 - mean_absol
Epoch 2/8
1/1 [==============================] - 0s 38ms/step - loss: 0.1979 - mean_abs
Epoch 3/8
1/1 [==============================] - 0s 39ms/step - loss: 0.1479 - mean_abs
Epoch 4/8
1/1 [==============================] - 0s 43ms/step - loss: 0.1047 - mean_abs
Epoch 5/8
1/1 [==============================] - 0s 43ms/step - loss: 0.0701 - mean_abs
Epoch 6/8
1/1 [==============================] - 0s 56ms/step - loss: 0.0444 - mean_abs
Epoch 7/8
1/1 [==============================] - 0s 56ms/step - loss: 0.0268 - mean_abs
Epoch 8/8
1/1 [==============================] - 0s 65ms/step - loss: 0.0169 - mean_abs
1/1 [==============================] - 0s 189ms/step
Mean Absolute Error on the test set: 0.1938
```




```
model_2.save('model_3.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an
  saving_api.save_model(
```

## Preprocessing

```
# load the data
raw_df = pd.read_excel("udaipur_rice.xls")
raw_df.to_csv('udaipur.csv', index=False)
raw_df
```

```
WARNING *** file size (16320) not 512 + multiple of sector size (512)
```

| | Crop Production Statistics | Unnamed: 1 | Unnamed: 2 | Unnamed: 3 | Unnamed: 4 | Unnamed: 5 |
|---|---|---|---|---|---|---|
| **0** | NaN | NaN | NaN | NaN | NaN | NaN |
| **1** | State/Crop/District | Year | Season | Area (Hectare) | Production (Tonnes) | Yield (Tonnes/Hectare) |
| **2** | Rajasthan | NaN | NaN | NaN | NaN | NaN |
| **3** | Rice | NaN | NaN | NaN | NaN | NaN |
| **4** | 1.UDAIPUR | 1997-98 | Kharif | 10200 | 8700 | 0.852941 |
| **5** | NaN | 1998-99 | Kharif | 8444 | 5257 | 0.622572 |
| **6** | NaN | 1999-00 | Kharif | 6452 | 2877 | 0.445908 |
| **7** | NaN | 2000-01 | Kharif | 5879 | 1663 | 0.282871 |
| **8** | NaN | 2001-02 | Kharif | 6272 | 7038 | 1.12213 |
| **9** | NaN | 2002-03 | Kharif | 3683 | 880 | 0.238936 |
| **10** | NaN | 2003-04 | Kharif | 5141 | 6070 | 1.180704 |
| **11** | NaN | 2004-05 | Kharif | 5564 | 4165 | 0.748562 |
| **12** | NaN | 2005-06 | Kharif | 5785 | 4477 | 0.773898 |
| **13** | NaN | 2006-07 | Kharif | 6175 | 5725 | 0.927126 |
| **14** | NaN | 2007-08 | Kharif | 5759 | 4256 | 0.739017 |
| **15** | NaN | 2008-09 | Kharif | 3689 | 2013 | 0.545676 |
| **16** | NaN | 2009-10 | Kharif | 4253 | 2594 | 0.609922 |
| **17** | NaN | 2010-11 | Kharif | 4881 | 5269 | 1.079492 |
| **18** | NaN | 2011-12 | Kharif | 5436 | 5676 | 1.04415 |
| **19** | NaN | 2012-13 | Kharif | 4710 | 4876 | 1.035244 |
| **20** | NaN | 2013-14 | Kharif | 4881 | 5596 | 1.146486 |
| **21** | NaN | 2014-15 | Kharif | 4855 | 5221 | 1.075386 |
| **22** | NaN | 2015-16 | Kharif | 4868 | 4667 | 0.95871 |
| **23** | NaN | 2016-17 | Kharif | 4864 | 5488 | 1.128289 |
| **24** | NaN | 2017-18 | Kharif | 5202 | 5937 | 1.141292 |
| **25** | NaN | 2018-19 | Kharif | 4641 | 4629 | 0.997414 |
| **26** | NaN | 2019-20 | Kharif | 4692 | 4721 | 1.006181 |

```
raw_df.columns = raw_df.iloc[1]
raw_df
```

| 1 | State/Crop/District | Year | Season | Area (Hectare) | Production (Tonnes) | Yield (Tonnes/Hectare) |
|---|---|---|---|---|---|---|
| 0 | NaN | NaN | NaN | NaN | NaN | NaN |
| 1 | State/Crop/District | Year | Season | Area (Hectare) | Production (Tonnes) | Yield (Tonnes/Hectare) |
| 2 | Rajasthan | NaN | NaN | NaN | NaN | NaN |
| 3 | Rice | NaN | NaN | NaN | NaN | NaN |
| 4 | 1.UDAIPUR | 1997-98 | Kharif | 10200 | 8700 | 0.852941 |
| 5 | NaN | 1998-99 | Kharif | 8444 | 5257 | 0.622572 |
| 6 | NaN | 1999-00 | Kharif | 6452 | 2877 | 0.445908 |
| 7 | NaN | 2000-01 | Kharif | 5879 | 1663 | 0.282871 |
| 8 | NaN | 2001-02 | Kharif | 6272 | 7038 | 1.12213 |
| 9 | NaN | 2002-03 | Kharif | 3683 | 880 | 0.238936 |
| 10 | NaN | 2003-04 | Kharif | 5141 | 6070 | 1.180704 |
| 11 | NaN | 2004-05 | Kharif | 5564 | 4165 | 0.748562 |
| 12 | NaN | 2005-06 | Kharif | 5785 | 4477 | 0.773898 |
| 13 | NaN | 2006-07 | Kharif | 6175 | 5725 | 0.927126 |
| 14 | NaN | 2007-08 | Kharif | 5759 | 4256 | 0.739017 |
| 15 | NaN | 2008-09 | Kharif | 3689 | 2013 | 0.545676 |
| 16 | NaN | 2009-10 | Kharif | 4253 | 2594 | 0.609922 |
| 17 | NaN | 2010-11 | Kharif | 4881 | 5269 | 1.079492 |
| 18 | NaN | 2011-12 | Kharif | 5436 | 5676 | 1.04415 |

```
raw_df.dropna(subset=['Yield (Tonnes/Hectare)'], inplace=True)
raw_df
```

| 1 | State/Crop/District | Year | Season | Area (Hectare) | Production (Tonnes) | Yield (Tonnes/Hectare) |
|---|---|---|---|---|---|---|
| 1 | State/Crop/District | Year | Season | Area (Hectare) | Production (Tonnes) | Yield (Tonnes/Hectare) |
| 4 | 1.UDAIPUR | 1997-98 | Kharif | 10200 | 8700 | 0.852941 |
| 5 | NaN | 1998-99 | Kharif | 8444 | 5257 | 0.622572 |
| 6 | NaN | 1999-00 | Kharif | 6452 | 2877 | 0.445908 |
| 7 | NaN | 2000-01 | Kharif | 5879 | 1663 | 0.282871 |
| 8 | NaN | 2001-02 | Kharif | 6272 | 7038 | 1.12213 |
| 9 | NaN | 2002-03 | Kharif | 3683 | 880 | 0.238936 |
| 10 | NaN | 2003-04 | Kharif | 5141 | 6070 | 1.180704 |
| 11 | NaN | 2004-05 | Kharif | 5564 | 4165 | 0.748562 |
| 12 | NaN | 2005-06 | Kharif | 5785 | 4477 | 0.773898 |
| 13 | NaN | 2006-07 | Kharif | 6175 | 5725 | 0.927126 |
| 14 | NaN | 2007-08 | Kharif | 5759 | 4256 | 0.739017 |
| 15 | NaN | 2008-09 | Kharif | 3689 | 2013 | 0.545676 |
| 16 | NaN | 2009-10 | Kharif | 4253 | 2594 | 0.609922 |
| 17 | NaN | 2010-11 | Kharif | 4881 | 5269 | 1.079492 |
| 18 | NaN | 2011-12 | Kharif | 5436 | 5676 | 1.04415 |

```
df = raw_df.drop(index = 1)
df
```

| 1 | State/Crop/District | Year | Season | Area (Hectare) | Production (Tonnes) | Yield (Tonnes/Hectare) |
|---|---|---|---|---|---|---|
| 4 | 1.UDAIPUR | 1997-98 | Kharif | 10200 | 8700 | 0.852941 |
| 5 | NaN | 1998-99 | Kharif | 8444 | 5257 | 0.622572 |
| 6 | NaN | 1999-00 | Kharif | 6452 | 2877 | 0.445908 |
| 7 | NaN | 2000-01 | Kharif | 5879 | 1663 | 0.282871 |
| 8 | NaN | 2001-02 | Kharif | 6272 | 7038 | 1.12213 |
| 9 | NaN | 2002-03 | Kharif | 3683 | 880 | 0.238936 |
| 10 | NaN | 2003-04 | Kharif | 5141 | 6070 | 1.180704 |
| 11 | NaN | 2004-05 | Kharif | 5564 | 4165 | 0.748562 |
| 12 | NaN | 2005-06 | Kharif | 5785 | 4477 | 0.773898 |
| 13 | NaN | 2006-07 | Kharif | 6175 | 5725 | 0.927126 |
| 14 | NaN | 2007-08 | Kharif | 5759 | 4256 | 0.739017 |
| 15 | NaN | 2008-09 | Kharif | 3689 | 2013 | 0.545676 |
| 16 | NaN | 2009-10 | Kharif | 4253 | 2594 | 0.609922 |
| 17 | NaN | 2010-11 | Kharif | 4881 | 5269 | 1.079492 |
| 18 | NaN | 2011-12 | Kharif | 5436 | 5676 | 1.04415 |

```
df = df.drop(index = 27)
df
```

| 1 | State/Crop/District | Year | Season | Area (Hectare) | Production (Tonnes) | Yield (Tonnes/Hectare) |
|---|---|---|---|---|---|---|
| 4 | 1.UDAIPUR | 1997-98 | Kharif | 10200 | 8700 | 0.852941 |
| 5 | NaN | 1998-99 | Kharif | 8444 | 5257 | 0.622572 |
| 6 | NaN | 1999-00 | Kharif | 6452 | 2877 | 0.445908 |
| 7 | NaN | 2000-01 | Kharif | 5879 | 1663 | 0.282871 |
| 8 | NaN | 2001-02 | Kharif | 6272 | 7038 | 1.12213 |
| 9 | NaN | 2002-03 | Kharif | 3683 | 880 | 0.238936 |
| 10 | NaN | 2003-04 | Kharif | 5141 | 6070 | 1.180704 |
| 11 | NaN | 2004-05 | Kharif | 5564 | 4165 | 0.748562 |
| 12 | NaN | 2005-06 | Kharif | 5785 | 4477 | 0.773898 |
| 13 | NaN | 2006-07 | Kharif | 6175 | 5725 | 0.927126 |
| 14 | NaN | 2007-08 | Kharif | 5759 | 4256 | 0.739017 |
| 15 | NaN | 2008-09 | Kharif | 3689 | 2013 | 0.545676 |
| 16 | NaN | 2009-10 | Kharif | 4253 | 2594 | 0.609922 |
| 17 | NaN | 2010-11 | Kharif | 4881 | 5269 | 1.079492 |
| 18 | NaN | 2011-12 | Kharif | 5436 | 5676 | 1.04415 |

```
df['Year'] = df['Year'].fillna(method='ffill')
df
```

| 1 | State/Crop/District | Year | Season | Area (Hectare) | Production (Tonnes) | Yielc (Tonnes/Hectare) |
|---|---|---|---|---|---|---|
| 4 | 1.VILLUPURAM | 1998-99 | Kharif | 156618 | 539236 | 3.443001 |
| 5 | NaN | 1999-00 | Kharif | 142737 | 488497 | 3.422357 |
| 6 | NaN | 2000-01 | Kharif | 141767 | 498417 | 3.515748 |
| 7 | NaN | 2001-02 | Kharif | 155555 | 561130 | 3.607277 |
| 8 | NaN | 2002-03 | Kharif | 80630 | 241287 | 2.992521 |
| 9 | NaN | 2003-04 | Kharif | 75279 | 234750 | 3.1184 |
| 10 | NaN | 2004-05 | Kharif | 163696 | 525850 | 3.212357 |
| 11 | NaN | 2005-06 | Kharif | 168435 | 511931 | 3.039339 |
| 12 | NaN | 2006-07 | Kharif | 144783 | 496113 | 3.426597 |
| 13 | NaN | 2007-08 | Kharif | 145403 | 480329 | 3.303433 |
| 14 | NaN | 2008-09 | Kharif | 146641 | 451139 | 3.076486 |
| 15 | NaN | 2009-10 | Kharif | 148454 | 467831 | 3.151353 |
| 16 | NaN | 2010-11 | Kharif | 149929 | 512013 | 3.415036 |
| 17 | NaN | 2011-12 | Kharif | 129858 | 471615 | 3.631775 |
| 18 | NaN | 2012-13 | Kharif | 115591 | 396530 | 3.430457 |
| 19 | NaN | 2013-14 | Kharif | 170443 | 796608 | 4.67375 |
| 20 | NaN | 2014-15 | Kharif | 171478 | 832585 | 4.855346 |
| 21 | NaN | 2015-16 | Kharif | 182303 | 773313 | 4.24191 |
| 22 | NaN | 2016-17 | Kharif | 95560 | 354198 | 3.706551 |
|  |  | 2017- |  |  |  |  |

```
df = df.drop(index = 23)
df = df.drop(index = 24)
df = df.drop(index = 25)
df = df.drop(index = 27)
df = df.drop(index = 28)
df = df.drop(index = 29)
df = df.drop(index = 31)
df = df.drop(index = 32)
df = df.drop(index = 33)
df
```

| 1 | State/Crop/District | Year | Season | Area (Hectare) | Production (Tonnes) | Yield (Tonnes/Hectare) |
|---|---|---|---|---|---|---|
| 4 | 1.VILLUPURAM | 1998-99 | Kharif | 156618 | 539236 | 3.443001 |
| 5 | NaN | 1999-00 | Kharif | 142737 | 488497 | 3.422357 |
| 6 | NaN | 2000-01 | Kharif | 141767 | 498417 | 3.515748 |
| 7 | NaN | 2001-02 | Kharif | 155555 | 561130 | 3.607277 |
| 8 | NaN | 2002-03 | Kharif | 80630 | 241287 | 2.992521 |
| 9 | NaN | 2003-04 | Kharif | 75279 | 234750 | 3.1184 |
| 10 | NaN | 2004-05 | Kharif | 163696 | 525850 | 3.212357 |
| 11 | NaN | 2005-06 | Kharif | 168435 | 511931 | 3.039339 |
| 12 | NaN | 2006-07 | Kharif | 144783 | 496113 | 3.426597 |
| 13 | NaN | 2007-08 | Kharif | 145403 | 480329 | 3.303433 |
| 14 | NaN | 2008-09 | Kharif | 146641 | 451139 | 3.076486 |
| 15 | NaN | 2009-10 | Kharif | 148454 | 467831 | 3.151353 |
| 16 | NaN | 2010-11 | Kharif | 149929 | 512013 | 3.415036 |
| 17 | NaN | 2011-12 | Kharif | 129858 | 471615 | 3.631775 |

```
newdf = df.reset_index()
newdf
```

| 1 | index | State/Crop/District | Year | Season | Area (Hectare) | Production (Tonnes) | (Tonnes/H |
|---|---|---|---|---|---|---|---|
| 0 | 4 | 1.UDAIPUR | 1997-98 | Kharif | 10200 | 8700 | |
| 1 | 5 | NaN | 1998-99 | Kharif | 8444 | 5257 | |
| 2 | 6 | NaN | 1999-00 | Kharif | 6452 | 2877 | |
| 3 | 7 | NaN | 2000-01 | Kharif | 5879 | 1663 | |
| 4 | 8 | NaN | 2001-02 | Kharif | 6272 | 7038 | |
| 5 | 9 | NaN | 2002-03 | Kharif | 3683 | 880 | |
| 6 | 10 | NaN | 2003-04 | Kharif | 5141 | 6070 | |
| 7 | 11 | NaN | 2004-05 | Kharif | 5564 | 4165 | |
| 8 | 12 | NaN | 2005-06 | Kharif | 5785 | 4477 | |
| 9 | 13 | NaN | 2006-07 | Kharif | 6175 | 5725 | |
| 10 | 14 | NaN | 2007-08 | Kharif | 5759 | 4256 | |
| 11 | 15 | NaN | 2008-09 | Kharif | 3689 | 2013 | |
| 12 | 16 | NaN | 2009-10 | Kharif | 4253 | 2594 | |
| 13 | 17 | NaN | 2010-11 | Kharif | 4881 | 5269 | |
| 14 | 18 | NaN | 2011-12 | Kharif | 5436 | 5676 | |

```
df = newdf.drop(['index', 'Season', 'Area (Hectare)', 'Production (Tonnes)','State/Crop/District'], axis=1)  # df.columns i
df
```

| 1 | Year | Yield (Tonnes/Hectare) |
|---|---|---|
| 0 | 1997-98 | 0.852941 |
| 1 | 1998-99 | 0.622572 |
| 2 | 1999-00 | 0.445908 |
| 3 | 2000-01 | 0.282871 |
| 4 | 2001-02 | 1.12213 |
| 5 | 2002-03 | 0.238936 |
| 6 | 2003-04 | 1.180704 |
| 7 | 2004-05 | 0.748562 |
| 8 | 2005-06 | 0.773898 |
| 9 | 2006-07 | 0.927126 |
| 10 | 2007-08 | 0.739017 |
| 11 | 2008-09 | 0.545676 |
| 12 | 2009-10 | 0.609922 |
| 13 | 2010-11 | 1.079492 |
| 14 | 2011-12 | 1.04415 |
| 15 | 2012-13 | 1.035244 |
| 16 | 2013-14 | 1.146486 |
| 17 | 2014-15 | 1.075386 |
| 18 | 2015-16 | 0.95871 |
| 19 | 2016-17 | 1.128289 |
| 20 | 2017-18 | 1.141292 |
| 21 | 2018-19 | 0.997414 |
| 22 | 2019-20 | 1.006181 |

```
df['Year'] = (pd.to_datetime(df['Year'].str.split('-', expand=True)[0])).dt.year
df.set_index('Year', inplace=True)
df = df.astype('float32')

df
```

```
<ipython-input-132-e51455692f2a>:1: UserWarning: Could not infer format, so e
  df['Year'] = (pd.to_datetime(df['Year'].str.split('-', expand=True)[0])).dt
```

| 1 | Yield (Tonnes/Hectare) |
|---|---|
| **Year** | |
| **1997** | 0.852941 |
| **1998** | 0.622572 |
| **1999** | 0.445908 |
| **2000** | 0.282871 |
| **2001** | 1.122130 |
| **2002** | 0.238936 |
| **2003** | 1.180704 |
| **2004** | 0.748562 |
| **2005** | 0.773898 |
| **2006** | 0.927126 |
| **2007** | 0.739017 |
| **2008** | 0.545676 |
| **2009** | 0.609922 |
| **2010** | 1.079492 |
| **2011** | 1.044150 |
| **2012** | 1.035244 |
| **2013** | 1.146486 |
| **2014** | 1.075386 |
| **2015** | 0.958710 |
| **2016** | 1.128289 |
| **2017** | 1.141292 |
| **2018** | 0.997414 |
| **2019** | 1.006181 |

```
# preprocessing
raw_df.columns = raw_df.iloc[1]
raw_df.dropna(subset=['Yield (Tonnes/Hectare)'], inplace=True)
raw_df['Year'] = raw_df['Year'].fillna(method='ffill')
df = raw_df.drop(index = 1)
df = df.drop(index = 35)
filtered_df = df[df['Season'].str.contains('Total')]
newdf = filtered_df.reset_index()
df = newdf.drop(['index', 'Season', 'Area (Hectare)', 'Production (Tonnes)'], axis=1)  # df.columns is zero-based pd.Index
df['Year'] = (pd.to_datetime(df['Year'].str.split('-', expand=True)[0])).dt.year
df.set_index('Year', inplace=True)
df = df.astype('float32')
df = df.drop(['State/Crop/District'], axis = 1)
df
```

```
<ipython-input-77-03c743d1bb2b>:10: UserWarning: Could not infer format, so e
  df['Year'] = (pd.to_datetime(df['Year'].str.split('-', expand=True)[0])).dt
```

| 1 | Yield (Tonnes/Hectare) |
|---|---|
| **Year** | |
| **2017** | 4.297967 |
| **2018** | 4.489400 |
| **2019** | 4.175825 |

```
df.plot(figsize=(12, 6))
```

<Axes: xlabel='Year'>

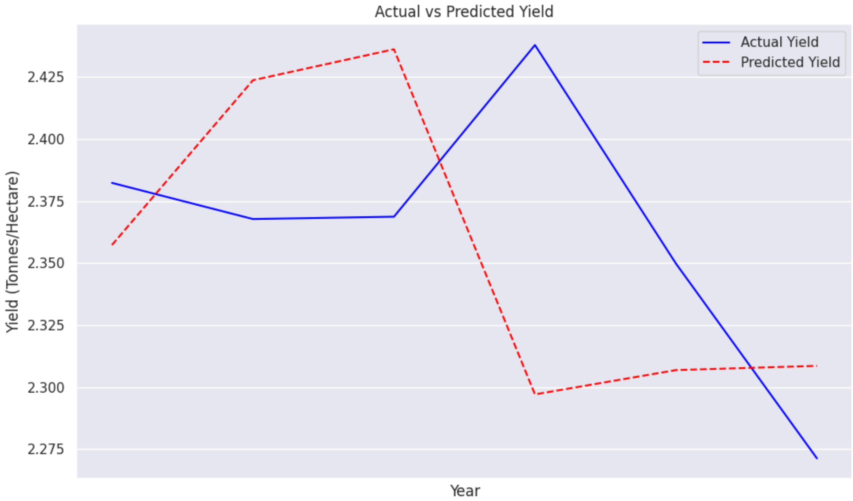

```
df.to_csv('udaipur_rice_cleaned_data.csv')
```

## Transfer Learnng for Marigaon

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from keras.models import load_model, Model
from keras.layers import Dense
import matplotlib.pyplot as plt
from keras.metrics import MeanAbsoluteError

# Load the saved model
saved_model_path = 'rice_yield_lstm_model.h5'
base_model = load_model(saved_model_path)

# Remove the output layer of the base model
base_model_output = base_model.layers[-2].output

# Create a new model by adding new layers on top of the base model (excluding the last layer)
x = Dense(200, activation='relu')(base_model_output)
x = Dense(200, activation='relu')(x)
output = Dense(1)(x)
model_2 = Model(inputs=base_model.input, outputs=output)

# Freeze all layers in the base model except the second-to-last layer
for layer in base_model.layers[:-2]:
    layer.trainable = False

# Compile the new model with MAE metric
model_2.compile(optimizer='adam', loss='mse', metrics=[MeanAbsoluteError()])

# Load your new dataset
new_file_path = 'Marigaon_rice_cleaned_data.csv'  # Replace with your new dataset file path
new_data = pd.read_csv(new_file_path)

# Data Preprocessing
new_data['Year'] = pd.to_datetime(new_data['Year'])
new_data.set_index('Year', inplace=True)
scaler = MinMaxScaler(feature_range=(0, 1))
new_data['Yield (Tonnes/Hectare)'] = scaler.fit_transform(new_data[['Yield (Tonnes/Hectare)']])

# Create sequences
def create_sequences(data, seq_length):
    xs = []
    ys = []
    for i in range(len(data)-seq_length):
        x = data[i:i+seq_length]
        y = data[i+seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

seq_length = 3
data = new_data['Yield (Tonnes/Hectare)'].values
X, y = create_sequences(data, seq_length)

split_idx = int(0.7 * len(X))
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]

# Reshape the data
X_train_reshaped = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_reshaped = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Train the new model (fine-tuning)
history = model_2.fit(X_train_reshaped, y_train, epochs=8, validation_data=(X_test_reshaped, y_test), verbose=1)

# Make predictions
y_pred = model_2.predict(X_test_reshaped)

# Inverse transform the predictions and the true values
y_pred_inverse = scaler.inverse_transform(y_pred)
y_test_inverse = scaler.inverse_transform(y_test.reshape(-1, 1))

# Compute MAE on the test set
mae = np.mean(np.abs(y_test_inverse - y_pred_inverse))
print(f'Mean Absolute Error on the test set: {mae:.4f}')

# Plot the results based on years
test_years = new_data.index[split_idx+seq_length:]

plt.figure(figsize=(10, 6))
plt.plot(test_years, y_test_inverse, color='blue', label='Actual Yield')
plt.plot(test_years, y_pred_inverse, color='red', linestyle='dashed', label='Predicted Yield')
plt.xlabel('Year')
```

```
plt.ylabel('Yield (Tonnes/Hectare)')
plt.title('Actual vs Predicted Yield')
plt.legend()
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()

# Plot training & validation loss and MAE
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 12))

# Loss plot
ax1.plot(history.history['loss'], label='Training Loss')
ax1.plot(history.history['val_loss'], label='Validation Loss')
ax1.set_title('Training and Validation Loss')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss (MSE)')
ax1.legend()

# MAE plot
ax2.plot(history.history['mean_absolute_error'], label='Training MAE')
ax2.plot(history.history['val_mean_absolute_error'], label='Validation MAE')
ax2.set_title('Training and Validation MAE')
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Mean Absolute Error')
ax2.legend()

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

```
Epoch 1/8
1/1 [==============================] - 8s 8s/step - loss: 0.3164 - mean_absol
Epoch 2/8
1/1 [==============================] - 0s 58ms/step - loss: 0.2462 - mean_abs
Epoch 3/8
1/1 [==============================] - 0s 40ms/step - loss: 0.1873 - mean_abs
Epoch 4/8
1/1 [==============================] - 0s 56ms/step - loss: 0.1401 - mean_abs
Epoch 5/8
1/1 [==============================] - 0s 39ms/step - loss: 0.1033 - mean_abs
Epoch 6/8
1/1 [==============================] - 0s 42ms/step - loss: 0.0755 - mean_abs
Epoch 7/8
1/1 [==============================] - 0s 38ms/step - loss: 0.0549 - mean_abs
Epoch 8/8
1/1 [==============================] - 0s 39ms/step - loss: 0.0410 - mean_abs
1/1 [==============================] - 0s 187ms/step
Mean Absolute Error on the test set: 0.0616
```




## Transfer Learning for Jorhat

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from keras.models import load_model, Model
from keras.layers import Dense
import matplotlib.pyplot as plt
from keras.metrics import MeanAbsoluteError

# Load the saved model
saved_model_path = 'rice_yield_lstm_model.h5'
base_model = load_model(saved_model_path)

# Remove the output layer of the base model
base_model_output = base_model.layers[-2].output

# Create a new model by adding new layers on top of the base model (excluding the last layer)
x = Dense(200, activation='relu')(base_model_output)
x = Dense(200, activation='relu')(x)
output = Dense(1)(x)
model_2 = Model(inputs=base_model.input, outputs=output)

# Freeze all layers in the base model except the second-to-last layer
for layer in base_model.layers[:-2]:
    layer.trainable = False

# Compile the new model with MAE metric
model_2.compile(optimizer='adam', loss='mse', metrics=[MeanAbsoluteError()])

# Load your new dataset
new_file_path = 'Jorhat_rice_cleaned_data.csv'  # Replace with your new dataset file path
new_data = pd.read_csv(new_file_path)

# Data Preprocessing
new_data['Year'] = pd.to_datetime(new_data['Year'])
new_data.set_index('Year', inplace=True)
scaler = MinMaxScaler(feature_range=(0, 1))
new_data['Yield (Tonnes/Hectare)'] = scaler.fit_transform(new_data[['Yield (Tonnes/Hectare)']])

# Create sequences
def create_sequences(data, seq_length):
    xs = []
    ys = []
    for i in range(len(data)-seq_length):
        x = data[i:i+seq_length]
        y = data[i+seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

seq_length = 3
data = new_data['Yield (Tonnes/Hectare)'].values
X, y = create_sequences(data, seq_length)

split_idx = int(0.7 * len(X))
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]

# Reshape the data
X_train_reshaped = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_reshaped = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Train the new model (fine-tuning)
history = model_2.fit(X_train_reshaped, y_train, epochs=8, validation_data=(X_test_reshaped, y_test), verbose=1)

# Make predictions
y_pred = model_2.predict(X_test_reshaped)

# Inverse transform the predictions and the true values
y_pred_inverse = scaler.inverse_transform(y_pred)
y_test_inverse = scaler.inverse_transform(y_test.reshape(-1, 1))

# Compute MAE on the test set
mae = np.mean(np.abs(y_test_inverse - y_pred_inverse))
print(f'Mean Absolute Error on the test set: {mae:.4f}')

# Plot the results based on years
test_years = new_data.index[split_idx+seq_length:]

plt.figure(figsize=(10, 6))
plt.plot(test_years, y_test_inverse, color='blue', label='Actual Yield')
plt.plot(test_years, y_pred_inverse, color='red', linestyle='dashed', label='Predicted Yield')
plt.xlabel('Year')
```

```
plt.ylabel('Yield (Tonnes/Hectare)')
plt.title('Actual vs Predicted Yield')
plt.legend()
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()

# Plot training & validation loss and MAE
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 12))

# Loss plot
ax1.plot(history.history['loss'], label='Training Loss')
ax1.plot(history.history['val_loss'], label='Validation Loss')
ax1.set_title('Training and Validation Loss')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss (MSE)')
ax1.legend()

# MAE plot
ax2.plot(history.history['mean_absolute_error'], label='Training MAE')
ax2.plot(history.history['val_mean_absolute_error'], label='Validation MAE')
ax2.set_title('Training and Validation MAE')
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Mean Absolute Error')
ax2.legend()

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

```
Epoch 1/8
1/1 [==============================] - 4s 4s/step - loss: 0.2377 - mean_absol
Epoch 2/8
1/1 [==============================] - 0s 60ms/step - loss: 0.1783 - mean_abs
Epoch 3/8
1/1 [==============================] - 0s 59ms/step - loss: 0.1334 - mean_abs
Epoch 4/8
1/1 [==============================] - 0s 56ms/step - loss: 0.0980 - mean_abs
Epoch 5/8
1/1 [==============================] - 0s 52ms/step - loss: 0.0697 - mean_abs
Epoch 6/8
1/1 [==============================] - 0s 73ms/step - loss: 0.0485 - mean_abs
Epoch 7/8
1/1 [==============================] - 0s 67ms/step - loss: 0.0349 - mean_abs
Epoch 8/8
1/1 [==============================] - 0s 53ms/step - loss: 0.0294 - mean_abs
1/1 [==============================] - 0s 294ms/step
Mean Absolute Error on the test set: 0.1427
```