```
import pandas as pd
import numpy as np

generation2021 = pd.read_csv("ASI - 2021.csv")
generation2021 = generation2021.drop(columns = [
    " Energy BOARD 1 3MW",
    " Energy BOARD 1 5MW",
    " Energy BOARD 10 3MW",
    " Energy BOARD 10 5MW",
    " Energy BOARD 11 3MW",
    " Energy BOARD 11 5MW",
    " Energy BOARD 12 5MW",
    " Energy BOARD 13 5MW",
    " Energy BOARD 14 5MW",
    " Energy BOARD 15 5MW",
    " Energy BOARD 16 5MW",
    " Energy BOARD 17 5MW",
    " Energy BOARD 18 5MW",
    " Energy BOARD 19 5MW",
    " Energy BOARD 2 3MW",
    " Energy BOARD 2 5MW",
    " Energy BOARD 3 3MW",
    " Energy BOARD 3 5MW",
    " Energy BOARD 4 3MW",
    " Energy BOARD 4 5MW",
    " Energy BOARD 5 3MW",
    " Energy BOARD 5 5MW",
    " Energy BOARD 6 3MW",
    " Energy BOARD 6 5MW",
    " Energy BOARD 7 3MW",
    " Energy BOARD 7 5MW",
    " Energy BOARD 8 3MW",
    " Energy BOARD 8 5MW",
    " Energy BOARD 9 3MW",
    " Energy BOARD 9 5MW",
    " Energy MSB  5MW 3200A",
    " Energy MSB 3MW",
    " Energy MSB 5MW 6300A"
]
)

generation2021["Time"] = pd.to_datetime(generation2021["Time"], format='mixed')
generation2021.set_index("Time", inplace = True)

daily_sum = generation2021.resample('10T').sum()

daily_sum['24 Hour sum solar power from solar panel (MW)'] = daily_sum.sum(axis=1)

data2021 = daily_sum[['24 Hour sum solar power from solar panel (MW)']].reset_index()

print(data2021)
```

```
                     Time  24 Hour sum solar power from solar panel (MW)
0     2021-01-01 00:00:00                                              0
1     2021-01-01 00:10:00                                              0
2     2021-01-01 00:20:00                                              0
3     2021-01-01 00:30:00                                              0
4     2021-01-01 00:40:00                                              0
...                   ...                                            ...
52555 2021-12-31 23:10:00                                              0
52556 2021-12-31 23:20:00                                              0
52557 2021-12-31 23:30:00                                              0
52558 2021-12-31 23:40:00                                              0
```

```
52559 2021-12-31 23:50:00                                            0

[52560 rows x 2 columns]
C:\Users\Hrth\AppData\Local\Temp\ipykernel_16196\630098888.py:45: FutureWarning: 'T' is deprecated
  daily_sum = generation2021.resample('10T').sum()
```

```
import pandas as pd
import numpy as np

generation2022 = pd.read_csv("ASI - 2022.csv")
generation2022 = generation2022.drop(columns = [
    " Energy BOARD 1 3MW",
    " Energy BOARD 1 5MW",
    " Energy BOARD 10 3MW",
    " Energy BOARD 10 5MW",
    " Energy BOARD 11 3MW",
    " Energy BOARD 11 5MW",
    " Energy BOARD 12 5MW",
    " Energy BOARD 13 5MW",
    " Energy BOARD 14 5MW",
    " Energy BOARD 15 5MW",
    " Energy BOARD 16 5MW",
    " Energy BOARD 17 5MW",
    " Energy BOARD 18 5MW",
    " Energy BOARD 19 5MW",
    " Energy BOARD 2 3MW",
    " Energy BOARD 2 5MW",
    " Energy BOARD 3 3MW",
    " Energy BOARD 3 5MW",
    " Energy BOARD 4 3MW",
    " Energy BOARD 4 5MW",
    " Energy BOARD 5 3MW",
    " Energy BOARD 5 5MW",
    " Energy BOARD 6 3MW",
    " Energy BOARD 6 5MW",
    " Energy BOARD 7 3MW",
    " Energy BOARD 7 5MW",
    " Energy BOARD 8 3MW",
    " Energy BOARD 8 5MW",
    " Energy BOARD 9 3MW",
    " Energy BOARD 9 5MW",
    " Energy MSB  5MW 3200A",
    " Energy MSB 3MW",
    " Energy MSB 5MW 6300A"
]
)

generation2022["Time"] = pd.to_datetime(generation2022["Time"], format='mixed')
generation2022.set_index("Time", inplace = True)

daily_sum = generation2022.resample('10T').sum()

daily_sum['24 Hour sum solar power from solar panel (MW)'] = daily_sum.sum(axis=1)

data2022 = daily_sum[['24 Hour sum solar power from solar panel (MW)']].reset_index()

print(data2022)
# print(data2022.to_string())
```

```
                        Time  24 Hour sum solar power from solar panel (MW)
0     2022-01-01 00:00:00                                            0.0
1     2022-01-01 00:10:00                                            0.0
```

```
2     2022-01-01 00:20:00                                             0.0
3     2022-01-01 00:30:00                                             0.0
4     2022-01-01 00:40:00                                             0.0
...                   ...                                             ...
52555 2022-12-31 23:10:00                                             0.0
52556 2022-12-31 23:20:00                                             0.0
52557 2022-12-31 23:30:00                                             0.0
52558 2022-12-31 23:40:00                                             0.0
52559 2022-12-31 23:50:00                                             0.0

[52560 rows x 2 columns]
C:\Users\Hrth\AppData\Local\Temp\ipykernel_16196\847790081.py:45: FutureWarning: 'T' is deprecated
  daily_sum = generation2022.resample('10T').sum()
```

```
import pandas as pd
import numpy as np

generation2023 = pd.read_csv("Generation - 2023 (January - August).csv")
generation2023 = generation2023.drop(columns = [
    " Energy BOARD 1 3MW",
    " Energy BOARD 1 5MW",
    " Energy BOARD 10 3MW",
    " Energy BOARD 10 5MW",
    " Energy BOARD 11 3MW",
    " Energy BOARD 11 5MW",
    " Energy BOARD 12 5MW",
    " Energy BOARD 13 5MW",
    " Energy BOARD 14 5MW",
    " Energy BOARD 15 5MW",
    " Energy BOARD 16 5MW",
    " Energy BOARD 17 5MW",
    " Energy BOARD 18 5MW",
    " Energy BOARD 19 5MW",
    " Energy BOARD 2 3MW",
    " Energy BOARD 2 5MW",
    " Energy BOARD 3 3MW",
    " Energy BOARD 3 5MW",
    " Energy BOARD 4 3MW",
    " Energy BOARD 4 5MW",
    " Energy BOARD 5 3MW",
    " Energy BOARD 5 5MW",
    " Energy BOARD 6 3MW",
    " Energy BOARD 6 5MW",
    " Energy BOARD 7 3MW",
    " Energy BOARD 7 5MW",
    " Energy BOARD 8 3MW",
    " Energy BOARD 8 5MW",
    " Energy BOARD 9 3MW",
    " Energy BOARD 9 5MW",
    " Energy MSB  5MW 3200A",
    " Energy MSB 3MW",
    " Energy MSB 5MW 6300A"
]
)

generation2023["Time"] = pd.to_datetime(generation2023["Time"], format='mixed')
generation2023.set_index("Time", inplace = True)

daily_sum = generation2023.resample('10T').sum()

daily_sum['24 Hour sum solar power from solar panel (MW)'] = daily_sum.sum(axis=1)

data2023 = daily_sum[['24 Hour sum solar power from solar panel (MW)']].reset_index()
```

```
print(data2023)
```

```
                     Time  24 Hour sum solar power from solar panel (MW)
0     2023-01-01 00:00:00                                              0
1     2023-01-01 00:10:00                                              0
2     2023-01-01 00:20:00                                              0
3     2023-01-01 00:30:00                                              0
4     2023-01-01 00:40:00                                              0
...                   ...                                            ...
34987 2023-08-31 23:10:00                                              0
34988 2023-08-31 23:20:00                                              0
34989 2023-08-31 23:30:00                                              0
34990 2023-08-31 23:40:00                                              0
34991 2023-08-31 23:50:00                                              0

[34992 rows x 2 columns]
C:\Users\Hrth\AppData\Local\Temp\ipykernel_16196\1146002310.py:45: FutureWarning: 'T' is deprecated
  daily_sum = generation2023.resample('10T').sum()
```

```
data = pd.concat([data2021, data2022, data2023], ignore_index=True)
data.describe()
```

	Time	24 Hour sum solar power from solar panel (MW)
count	140112	140112.000000
mean	2022-05-02 11:54:59.999999488	5093.091541
min	2021-01-01 00:00:00	0.000000
25%	2021-09-01 05:57:30	0.000000
50%	2022-05-02 11:55:00	29.000000
75%	2022-12-31 17:52:30	9599.250000
max	2023-08-31 23:50:00	28509.000000
std	NaN	7231.336095

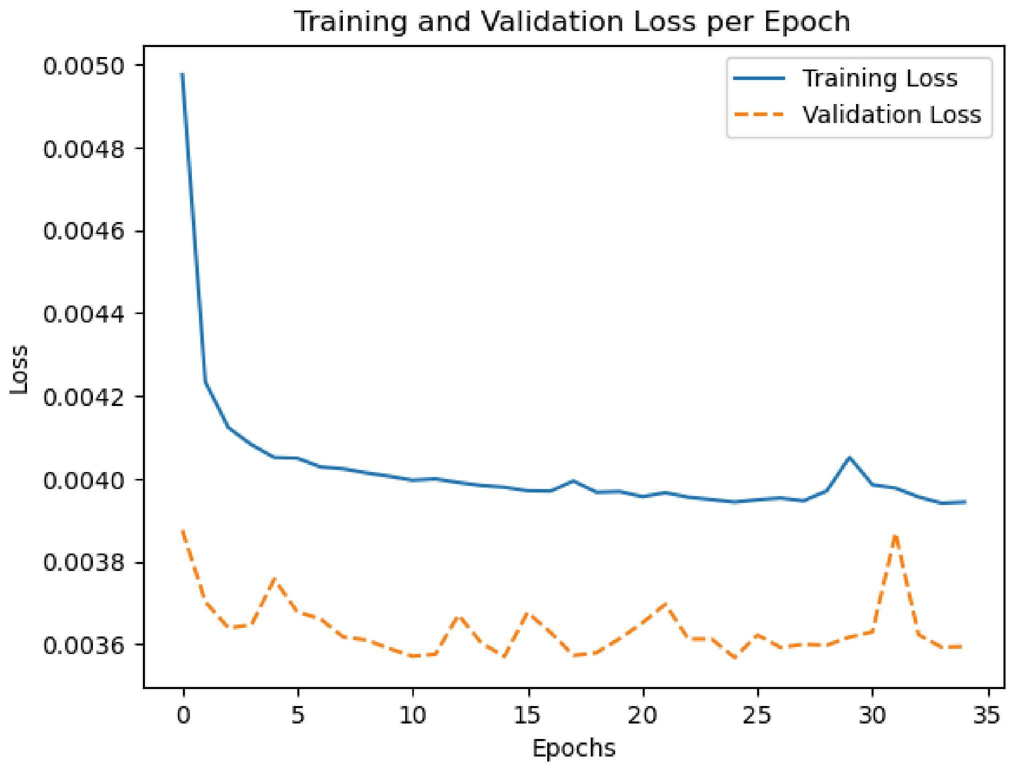
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.callbacks import EarlyStopping

# Select relevant columns
selected_columns = [
    '24 Hour sum solar power from solar panel (MW)'
    # '24 Hour mean solar power from solar panel (MW)'

]
data_selected = data[selected_columns]
split_ratio = 0.8

train_size = int(len(data_selected) * split_ratio)

train_data = data_selected[:train_size]
test_data = data_selected[train_size:]

# Normalize the data
scaler = MinMaxScaler()
train_scaled = scaler.fit_transform(train_data)
test_scaled = scaler.transform(test_data)
```

```python
# Prepare the data for LSTM
def create_dataset(X, y, time_steps=1):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        Xs.append(X[i:(i + time_steps)])
        ys.append(y[i + time_steps])
    return np.array(Xs), np.array(ys)

time_steps = 144 # 6: Short Term, 12: Medium Term, 144: Daily cycle
X_train, y_train = create_dataset(train_scaled, train_scaled[:, 0], time_steps)
X_test, y_test = create_dataset(test_scaled, test_scaled[:, 0], time_steps)

# Define the LSTM model architecture
model = Sequential([
    LSTM(units=64, input_shape=(X_train.shape[1], X_train.shape[2])),
    Dense(units=1)
])
model.compile(optimizer='adam', loss='mean_squared_error')

# Define the EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the LSTM model
history = model.fit(X_train, y_train, epochs=200, batch_size=32, validation_split=0.1, verbose=1, callb

# Evaluate the model performance
model.evaluate(X_test, y_test)

model.save("lstm_model.h5")

# Make predictions
predictions = model.predict(X_test)
```

```
Epoch 1/200
c:\Users\Hrth\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pa
  super().__init__(**kwargs)
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 72s 22ms/step - loss: 0.0064 - val_loss: 0.0039
Epoch 2/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 71s 22ms/step - loss: 0.0043 - val_loss: 0.0037
Epoch 3/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 69s 22ms/step - loss: 0.0040 - val_loss: 0.0036
Epoch 4/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 66s 21ms/step - loss: 0.0041 - val_loss: 0.0036
Epoch 5/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 69s 22ms/step - loss: 0.0041 - val_loss: 0.0038
Epoch 6/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 68s 22ms/step - loss: 0.0041 - val_loss: 0.0037
Epoch 7/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 70s 22ms/step - loss: 0.0040 - val_loss: 0.0037
Epoch 8/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 71s 23ms/step - loss: 0.0040 - val_loss: 0.0036
Epoch 9/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 70s 22ms/step - loss: 0.0041 - val_loss: 0.0036
Epoch 10/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 70s 22ms/step - loss: 0.0041 - val_loss: 0.0036
Epoch 11/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 69s 22ms/step - loss: 0.0039 - val_loss: 0.0036
Epoch 12/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 71s 22ms/step - loss: 0.0040 - val_loss: 0.0036
Epoch 13/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 70s 22ms/step - loss: 0.0040 - val_loss: 0.0037
Epoch 14/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 72s 23ms/step - loss: 0.0040 - val_loss: 0.0036
Epoch 15/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 70s 22ms/step - loss: 0.0039 - val_loss: 0.0036
```

```
Epoch 16/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 70s 22ms/step - loss: 0.0040 - val_loss: 0.0037
Epoch 17/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 68s 22ms/step - loss: 0.0040 - val_loss: 0.0036
Epoch 18/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 69s 22ms/step - loss: 0.0040 - val_loss: 0.0036
Epoch 19/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 69s 22ms/step - loss: 0.0040 - val_loss: 0.0036
Epoch 20/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 70s 22ms/step - loss: 0.0040 - val_loss: 0.0036
Epoch 21/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 69s 22ms/step - loss: 0.0039 - val_loss: 0.0037
Epoch 22/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 69s 22ms/step - loss: 0.0039 - val_loss: 0.0037
Epoch 23/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 67s 21ms/step - loss: 0.0039 - val_loss: 0.0036
Epoch 24/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 71s 22ms/step - loss: 0.0039 - val_loss: 0.0036
Epoch 25/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 71s 23ms/step - loss: 0.0040 - val_loss: 0.0036
Epoch 26/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 71s 23ms/step - loss: 0.0040 - val_loss: 0.0036
Epoch 27/200
3149/3149 ━━━━━━━━━━━━━━━━━━━━ 71s 23ms/step - loss: 0.0041 - val_loss: 0.0036
Epoch 28/200
```

```python
import matplotlib.pyplot as plt

# Plotting training and validation loss per epoch
plt.plot(history.history['loss'], label='Training Loss', linestyle='-')
plt.plot(history.history['val_loss'], label='Validation Loss', linestyle='--')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss per Epoch')
plt.show()
```




```python
y_pred = model.predict(X_test)
y_true = y_test
```

```
# Now can print y_true and y_pred
print("Actual values (y_true):", y_true)
print("Predicted values (y_pred):", y_pred)
```

```
872/872 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step
Actual values (y_true): [0.41200516 0.45729346 0.49099971 ... 0.         0.         0.        ]
Predicted values (y_pred): [[0.39354044]
 [0.42716295]
 [0.46544203]
 ...
 [0.00386974]
 [0.00385851]
 [0.00383478]]
```

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_true, y_pred)
print("Mean Squared Error (MSE):", mse)
```

```
Mean Squared Error (MSE): 0.004714859956599381
```

```
import numpy as np
import pandas as pd

# Flatten the X_test array
X_test_flat = X_test.reshape(X_test.shape[0], -1)

# Convert y_test and y_pred to 1D arrays
y_test_flat = y_test.flatten()
y_pred_flat = y_pred.flatten()

# Convert X_test_flat to DataFrame
X_test_df = pd.DataFrame(X_test_flat, columns=[f'Feature_{i}' for i in range(X_test_flat.shape[1])])

# Create DataFrame for y_test and y_pred
y_test_df = pd.DataFrame({'Actual values (y_true)': y_test_flat})
y_pred_df = pd.DataFrame({'Predicted values (y_pred)': y_pred_flat})

# Concatenate X_test_df, y_test_df, and y_pred_df along columns
result_df = pd.concat([X_test_df, y_test_df, y_pred_df], axis=1)

# Print the result DataFrame
result_df.head(10)
```

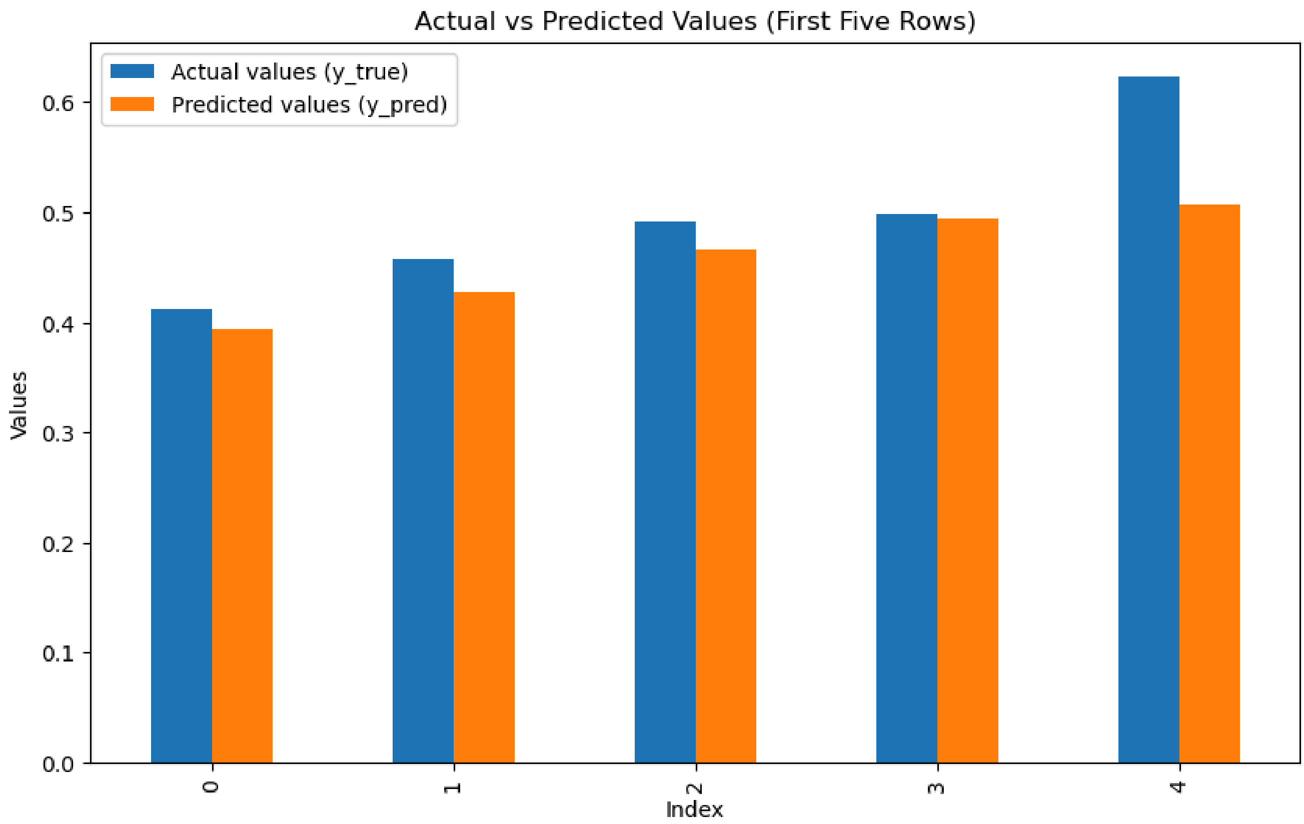
	Feature_0	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Feature_8
0	0.415304	0.454640	0.501004	0.539013	0.578600	0.609366	0.644937	0.673049	0.688791
1	0.454640	0.501004	0.539013	0.578600	0.609366	0.644937	0.673049	0.688791	0.744012
2	0.501004	0.539013	0.578600	0.609366	0.644937	0.673049	0.688791	0.744012	0.736804
3	0.539013	0.578600	0.609366	0.644937	0.673049	0.688791	0.744012	0.736804	0.670970
4	0.578600	0.609366	0.644937	0.673049	0.688791	0.744012	0.736804	0.670970	0.612665
5	0.609366	0.644937	0.673049	0.688791	0.744012	0.736804	0.670970	0.612665	0.705680
6	0.644937	0.673049	0.688791	0.744012	0.736804	0.670970	0.612665	0.705680	0.838282
7	0.673049	0.688791	0.744012	0.736804	0.670970	0.612665	0.705680	0.838282	0.834696
8	0.688791	0.744012	0.736804	0.670970	0.612665	0.705680	0.838282	0.834696	0.842477
9	0.744012	0.736804	0.670970	0.612665	0.705680	0.838282	0.834696	0.842477	0.884323

10 rows × 146 columns

```
result_df = result_df.iloc[:, -2:]
result_df.head()
```

	Actual values (y_true)	Predicted values (y_pred)
0	0.412005	0.393540
1	0.457293	0.427163
2	0.491000	0.465442
3	0.498924	0.494093
4	0.623530	0.506671

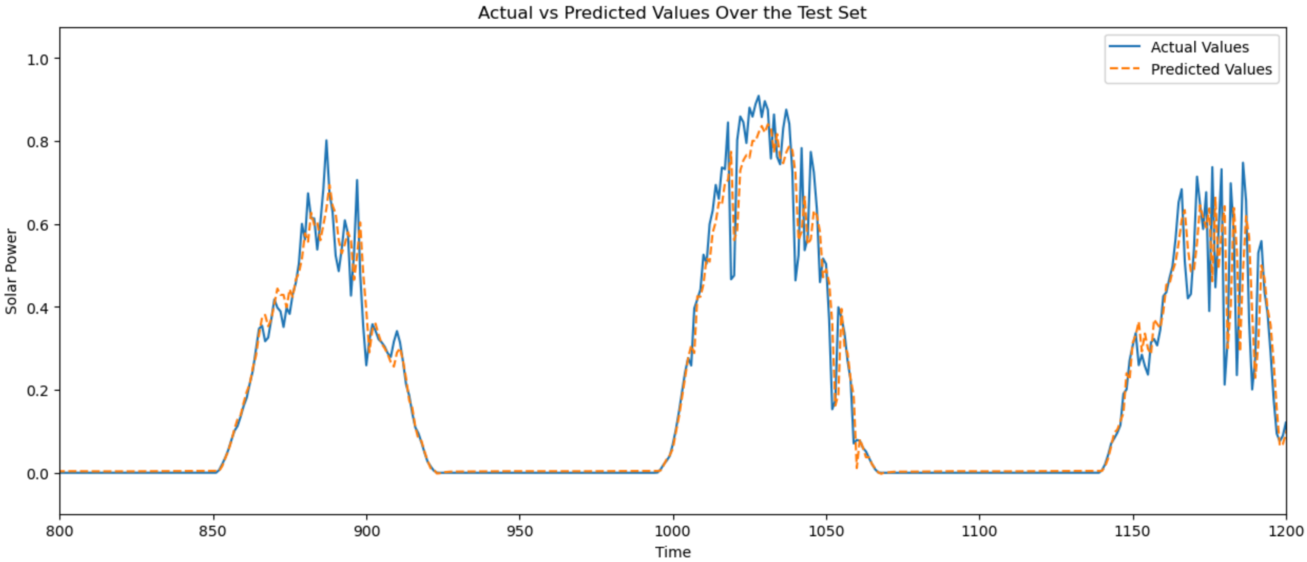
```
import matplotlib.pyplot as plt

# Selecting only the first five rows
result_df_first_five = result_df.iloc[:5]

# Plotting the actual vs predicted values for the first five rows
result_df_first_five.plot(kind='bar', figsize=(10, 6))
plt.title('Actual vs Predicted Values (First Five Rows)')
plt.xlabel('Index')
plt.ylabel('Values')
plt.show()
```




```
from sklearn.metrics import mean_absolute_error, mean_squared_error

y_true = result_df['Actual values (y_true)']
y_pred = result_df['Predicted values (y_pred)']

mae = mean_absolute_error(y_true, y_pred)
mse = mean_squared_error(y_true, y_pred)
rmse = mean_squared_error(y_true, y_pred, squared=False)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
```

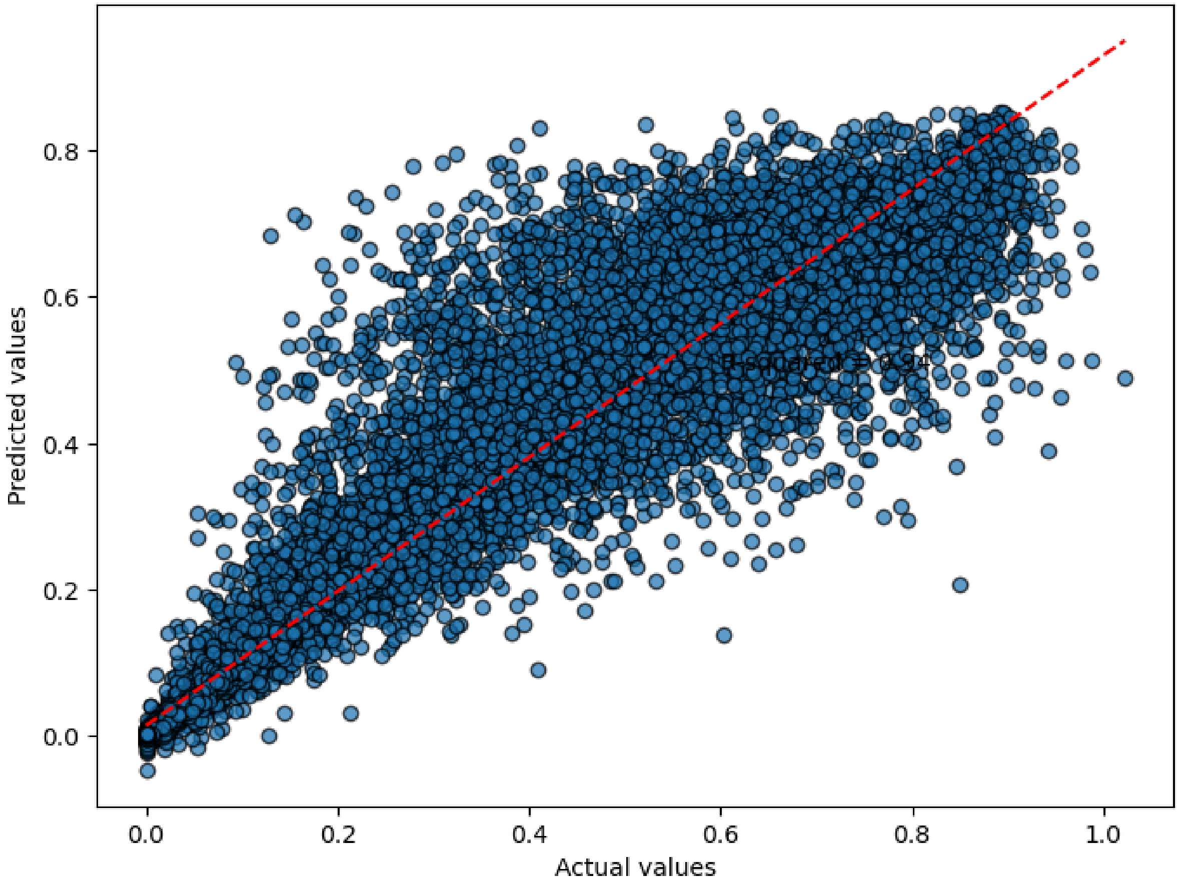
```
Mean Absolute Error: 0.032608578409018575
Mean Squared Error: 0.004714859956599381
Root Mean Squared Error: 0.0686648378473246
c:\Users\Hrth\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:483: FutureWarning: 'squar
  warnings.warn(
```

```
import matplotlib.pyplot as plt

# Plotting the actual vs predicted values over the entire test set
plt.figure(figsize=(15, 6))
plt.plot(result_df['Actual values (y_true)'], label='Actual Values')
plt.plot(result_df['Predicted values (y_pred)'], label='Predicted Values', linestyle='dashed')
plt.xlim(800,1200)
plt.title('Actual vs Predicted Values Over the Test Set')
plt.xlabel('Time')
plt.ylabel('Solar Power')
plt.legend()
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

r_squared = r2_score(y_true, y_pred)
plt.figure(figsize=(8, 6))
plt.scatter(y_true,y_pred, cmap='viridis', edgecolor='k', alpha=0.7)
plt.xlabel('Actual values')
plt.ylabel('Predicted values')

plt.plot(np.unique(y_true), np.poly1d(np.polyfit(y_true, y_pred, 1))(np.unique(y_true)), 'r--')

plt.text(0.6, 0.5, 'R-squared = %0.2f' % r_squared)
plt.show()
```

```
C:\Users\Hrth\AppData\Local\Temp\ipykernel_16196\702279148.py:7: UserWarning: No data for colormapp
  plt.scatter(y_true,y_pred, cmap='viridis', edgecolor='k', alpha=0.7)
```



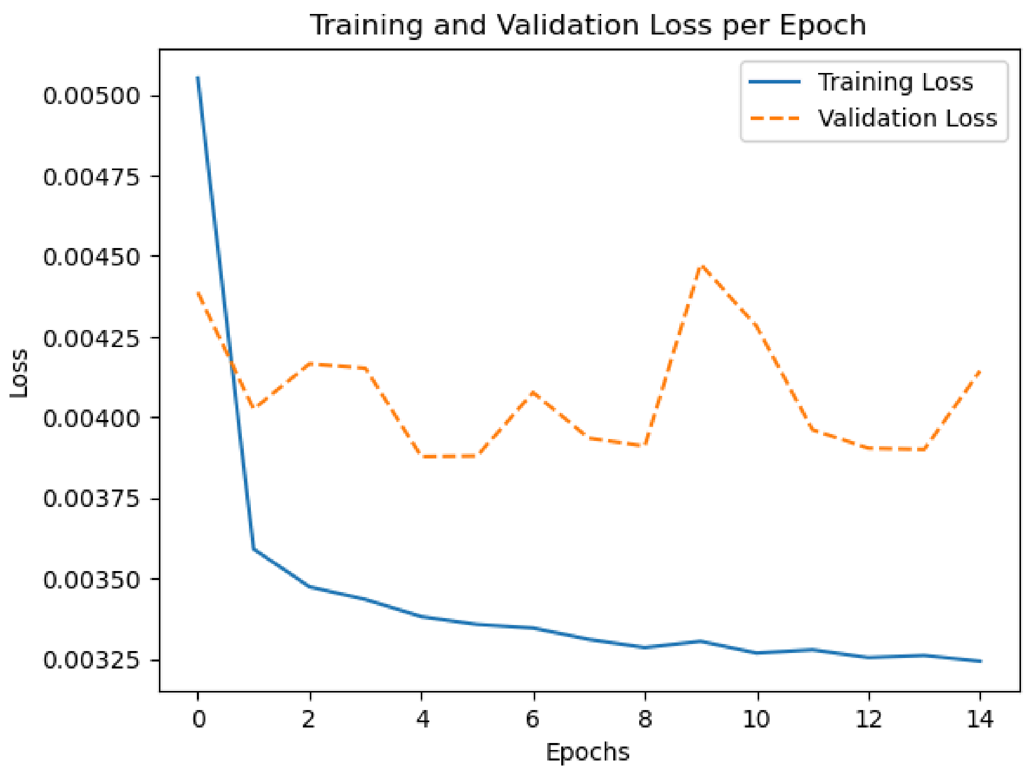
UNIVARIATE USING ASI GENERATION 2021 TO COMPARE WITH MULTIVARIATE ASI GENERATION 2021

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.callbacks import EarlyStopping

# Select relevant columns
selected_columns = [
    '24 Hour sum solar power from solar panel (MW)'
    # '24 Hour mean solar power from solar panel (MW)'

]
data_selected = data2021[selected_columns]
split_ratio = 0.8

train_size = int(len(data_selected) * split_ratio)

train_data = data_selected[:train_size]
test_data = data_selected[train_size:]

# Normalize the data
scaler = MinMaxScaler()
train_scaled = scaler.fit_transform(train_data)
test_scaled = scaler.transform(test_data)

# Prepare the data for LSTM
def create_dataset(X, y, time_steps=1):
    Xs, ys = [], []
```

```
    for i in range(len(X) - time_steps):
        Xs.append(X[i:(i + time_steps)])
        ys.append(y[i + time_steps])
    return np.array(Xs), np.array(ys)

time_steps = 144 # 6: Short Term, 12: Medium Term, 144: Daily cycle
X_train, y_train = create_dataset(train_scaled, train_scaled[:, 0], time_steps)
X_test, y_test = create_dataset(test_scaled, test_scaled[:, 0], time_steps)

# Define the LSTM model architecture
model = Sequential([
    LSTM(units=64, input_shape=(X_train.shape[1], X_train.shape[2])),
    Dense(units=1)
])
model.compile(optimizer='adam', loss='mean_squared_error')

# Define the EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the LSTM model
history = model.fit(X_train, y_train, epochs=200, batch_size=32, validation_split=0.1, verbose=1, callb

# Evaluate the model performance
model.evaluate(X_test, y_test)

model.save("lstm_univariate_model.h5")

# Make predictions
predictions = model.predict(X_test)
```

```
Epoch 1/200
c:\Users\Hrth\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass
  super().__init__(**kwargs)
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 28s 22ms/step - loss: 0.0076 - val_loss: 0.0044
Epoch 2/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 26s 22ms/step - loss: 0.0036 - val_loss: 0.0040
Epoch 3/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 27s 22ms/step - loss: 0.0035 - val_loss: 0.0042
Epoch 4/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 26s 22ms/step - loss: 0.0034 - val_loss: 0.0042
Epoch 5/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 26s 22ms/step - loss: 0.0034 - val_loss: 0.0039
Epoch 6/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 26s 22ms/step - loss: 0.0034 - val_loss: 0.0039
Epoch 7/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 27s 22ms/step - loss: 0.0033 - val_loss: 0.0041
Epoch 8/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 27s 23ms/step - loss: 0.0033 - val_loss: 0.0039
Epoch 9/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 27s 23ms/step - loss: 0.0034 - val_loss: 0.0039
Epoch 10/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 26s 22ms/step - loss: 0.0032 - val_loss: 0.0045
Epoch 11/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 26s 22ms/step - loss: 0.0032 - val_loss: 0.0043
Epoch 12/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 28s 24ms/step - loss: 0.0033 - val_loss: 0.0040
Epoch 13/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 27s 23ms/step - loss: 0.0031 - val_loss: 0.0039
Epoch 14/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 26s 22ms/step - loss: 0.0033 - val_loss: 0.0039
Epoch 15/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 26s 22ms/step - loss: 0.0032 - val_loss: 0.0041
324/324 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0053
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_mod
324/324 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step
```

```
import matplotlib.pyplot as plt

# Plotting training and validation loss per epoch
plt.plot(history.history['loss'], label='Training Loss', linestyle='-')
plt.plot(history.history['val_loss'], label='Validation Loss', linestyle='--')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss per Epoch')
plt.show()
```




```
y_pred = model.predict(X_test)
y_true = y_test

# Now can print y_true and y_pred
print("Actual values (y_true):", y_true)
print("Predicted values (y_pred):", y_pred)
```

```
324/324 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
Actual values (y_true): [0. 0. 0. ... 0. 0. 0.]
Predicted values (y_pred): [[0.00673043]
 [0.00678476]
 [0.00682513]
 ...
 [0.00645103]
 [0.00633337]
 [0.00622264]]
```

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_true, y_pred)
print("Mean Squared Error (MSE):", mse)
```

```
Mean Squared Error (MSE): 0.004530084933743686
```

```
import numpy as np
import pandas as pd
```

```
# Flatten the X_test array
X_test_flat = X_test.reshape(X_test.shape[0], -1)

# Convert y_test and y_pred to 1D arrays
y_test_flat = y_test.flatten()
y_pred_flat = y_pred.flatten()

# Convert X_test_flat to DataFrame
X_test_df = pd.DataFrame(X_test_flat, columns=[f'Feature_{i}' for i in range(X_test_flat.shape[1])])

# Create DataFrame for y_test and y_pred
y_test_df = pd.DataFrame({'Actual values (y_true)': y_test_flat})
y_pred_df = pd.DataFrame({'Predicted values (y_pred)': y_pred_flat})

# Concatenate X_test_df, y_test_df, and y_pred_df along columns
result_df = pd.concat([X_test_df, y_test_df, y_pred_df], axis=1)

# Print the result DataFrame
result_df.head(10)
```

	Feature_0	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Feature_8
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.(
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.(
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.(
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.(
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.(
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.(
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.(
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.(
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.(
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.(

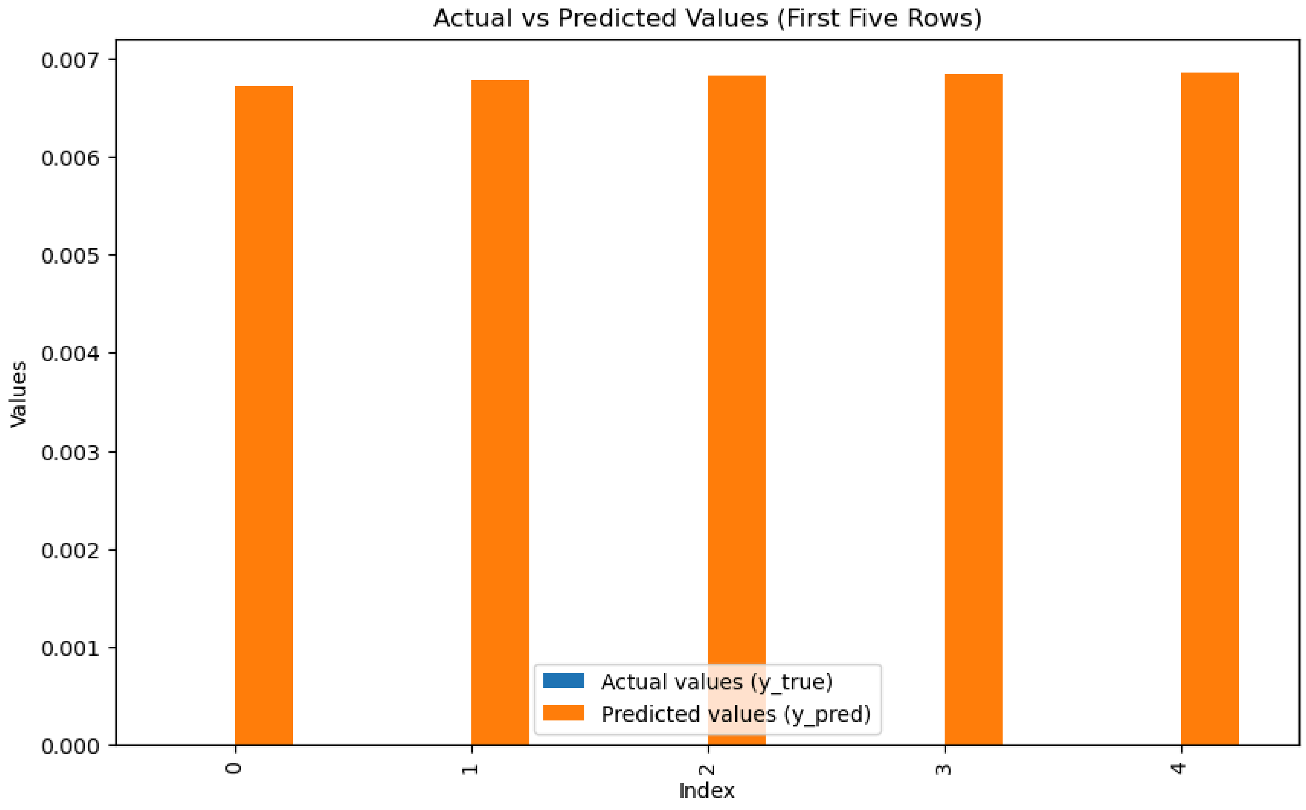
10 rows × 146 columns

```
result_df = result_df.iloc[:, -2:]
result_df.head()
```

	Actual values (y_true)	Predicted values (y_pred)
0	0.0	0.006730
1	0.0	0.006785
2	0.0	0.006825
3	0.0	0.006852
4	0.0	0.006866

```
import matplotlib.pyplot as plt
```

```
# Selecting only the first five rows
result_df_first_five = result_df.iloc[:5]

# Plotting the actual vs predicted values for the first five rows
result_df_first_five.plot(kind='bar', figsize=(10, 6))
plt.title('Actual vs Predicted Values (First Five Rows)')
plt.xlabel('Index')
plt.ylabel('Values')
plt.show()
```

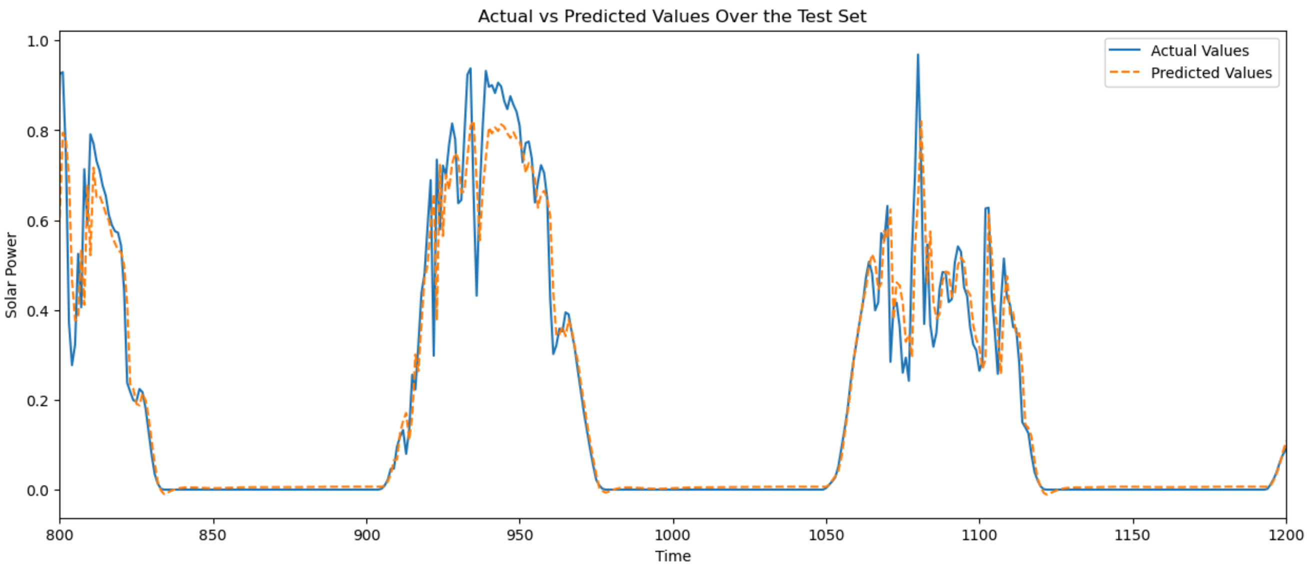



```
from sklearn.metrics import mean_absolute_error, mean_squared_error

y_true = result_df['Actual values (y_true)']
y_pred = result_df['Predicted values (y_pred)']

mae = mean_absolute_error(y_true, y_pred)
mse = mean_squared_error(y_true, y_pred)
rmse = mean_squared_error(y_true, y_pred, squared=False)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
```

```
Mean Absolute Error: 0.03281725517127589
Mean Squared Error: 0.004530084933743686
Root Mean Squared Error: 0.06730590563794299
c:\Users\Hrth\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:483: FutureWarning: 'squar
  warnings.warn(
```

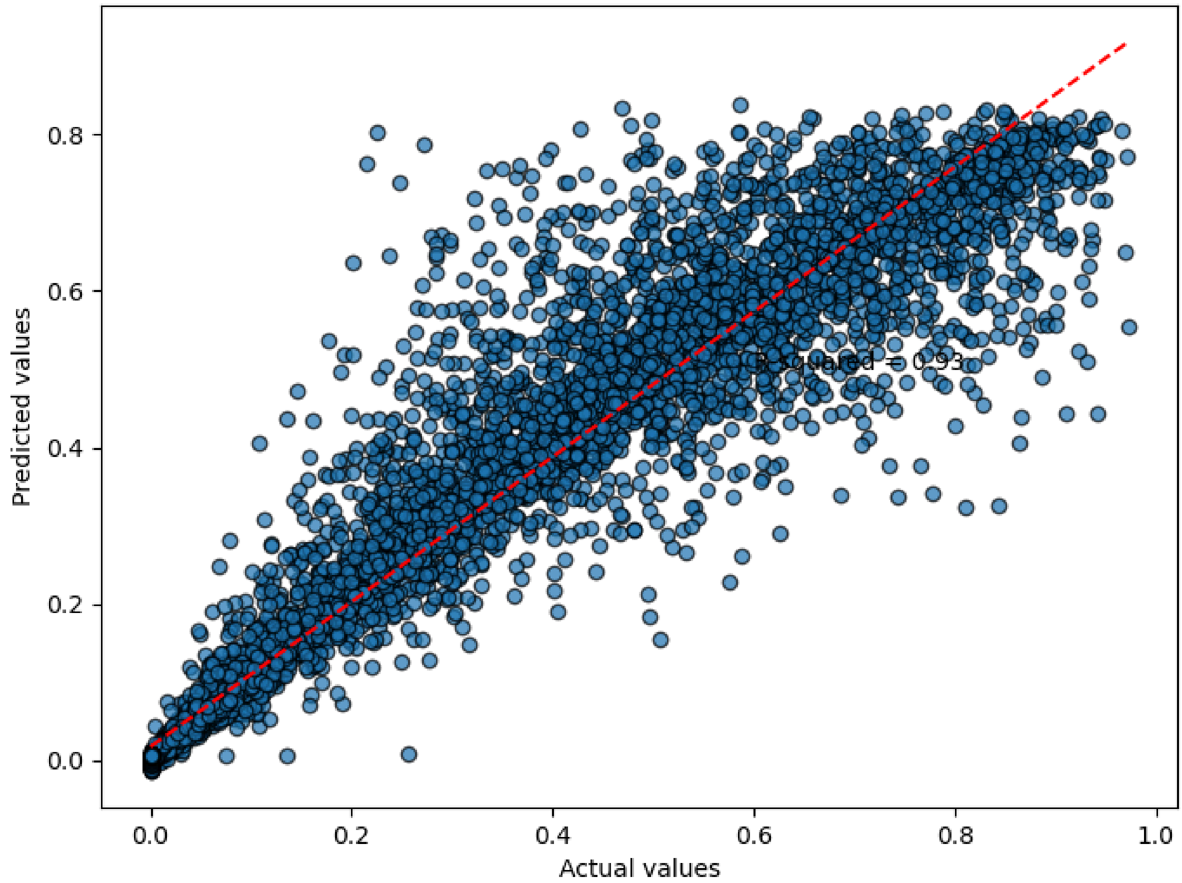
```
import matplotlib.pyplot as plt

# Plotting the actual vs predicted values over the entire test set
```

```
plt.figure(figsize=(15, 6))
plt.plot(result_df['Actual values (y_true)'], label='Actual Values')
plt.plot(result_df['Predicted values (y_pred)'], label='Predicted Values', linestyle='dashed')
plt.xlim(800,1200)
plt.title('Actual vs Predicted Values Over the Test Set')
plt.xlabel('Time')
plt.ylabel('Solar Power')
plt.legend()
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

r_squared = r2_score(y_true, y_pred)
plt.figure(figsize=(8, 6))
plt.scatter(y_true,y_pred, cmap='viridis', edgecolor='k', alpha=0.7)
plt.xlabel('Actual values')
plt.ylabel('Predicted values')

plt.plot(np.unique(y_true), np.poly1d(np.polyfit(y_true, y_pred, 1))(np.unique(y_true)), 'r--')

plt.text(0.6, 0.5, 'R-squared = %0.2f' % r_squared)
plt.show()
```

```
C:\Users\Hrth\AppData\Local\Temp\ipykernel_16196\702279148.py:7: UserWarning: No data for colormapp
  plt.scatter(y_true,y_pred, cmap='viridis', edgecolor='k', alpha=0.7)
```



MULTIVARIATE USING ASI GENERATION 2021 + WEATHER 2021 TO COMPARE WITH UNIVARIATE ASI GENERATION 2021

```
import pandas as pd
import numpy as np

environment2021 = pd.read_csv("Weather sensor - 2021.csv", encoding='unicode_escape')
environment2021 = environment2021[['Time', 'W/m^2', '°C']]

environment2021["Time"] = pd.to_datetime(environment2021["Time"], format='mixed')
environment2021.set_index("Time", inplace=True)

environment2021_resampled = environment2021.resample('10T').mean()

weatherdata2021 = environment2021_resampled[['W/m^2', '°C']].reset_index()

weatherdata2021.rename(columns={'W/m^2': '10-min mean W/m^2', '°C': '10-min mean °C'}, inplace=True)

print(weatherdata2021)
```

```
C:\Users\Hrth\AppData\Local\Temp\ipykernel_16196\3430815374.py:4: DtypeWarning: Columns (3,5) have
  environment2021 = pd.read_csv("Weather sensor - 2021.csv", encoding='unicode_escape')
                     Time  10-min mean W/m^2  10-min mean °C
0     2021-01-01 00:00:00                0.0          24.665
1     2021-01-01 00:10:00                0.0          24.520
2     2021-01-01 00:20:00                0.0          24.395
3     2021-01-01 00:30:00                0.0          24.350
4     2021-01-01 00:40:00                0.0          24.235
...                   ...                ...             ...
```

```
52555 2021-12-31 23:10:00               0.0          22.750
52556 2021-12-31 23:20:00               0.0          22.600
52557 2021-12-31 23:30:00               0.0          22.600
52558 2021-12-31 23:40:00               0.0          22.650
52559 2021-12-31 23:50:00               0.0          22.650

[52560 rows x 3 columns]
C:\Users\Hrth\AppData\Local\Temp\ipykernel_16196\3430815374.py:10: FutureWarning: 'T' is deprecated
  environment2021_resampled = environment2021.resample('10T').mean()
```

```
weatherdata2021 = weatherdata2021.drop(columns = ['Time'])
combined_data2021 = pd.concat([data2021, weatherdata2021], axis=1, join='inner')

print(combined_data2021)
```

```
                      Time  24 Hour sum solar power from solar panel (MW)  \
0     2021-01-01 00:00:00                                              0
1     2021-01-01 00:10:00                                              0
2     2021-01-01 00:20:00                                              0
3     2021-01-01 00:30:00                                              0
4     2021-01-01 00:40:00                                              0
...                   ...                                            ...
52555 2021-12-31 23:10:00                                              0
52556 2021-12-31 23:20:00                                              0
52557 2021-12-31 23:30:00                                              0
52558 2021-12-31 23:40:00                                              0
52559 2021-12-31 23:50:00                                              0

       10-min mean W/m^2  10-min mean °C
0                    0.0          24.665
1                    0.0          24.520
2                    0.0          24.395
3                    0.0          24.350
4                    0.0          24.235
...                  ...             ...
52555                0.0          22.750
52556                0.0          22.600
52557                0.0          22.600
52558                0.0          22.650
52559                0.0          22.650

[52560 rows x 4 columns]
```

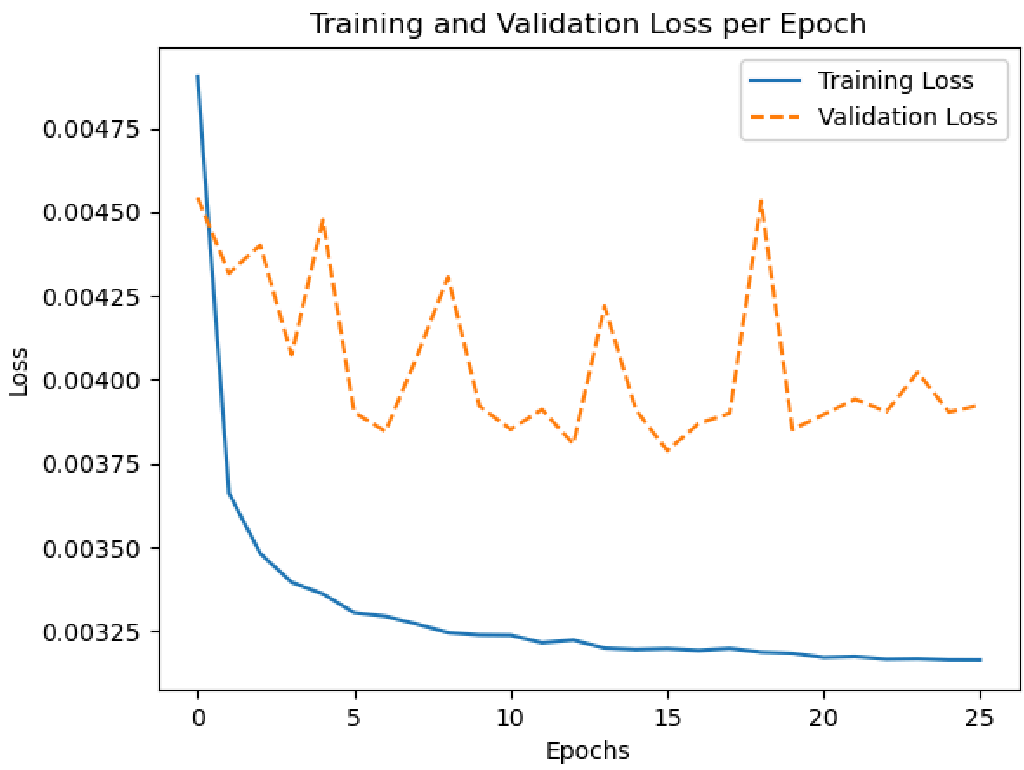
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.callbacks import EarlyStopping

# Select relevant columns
selected_columns = [
    '24 Hour sum solar power from solar panel (MW)',
    '10-min mean W/m^2',
    '10-min mean °C'

]
data_selected = combined_data2021[selected_columns]
split_ratio = 0.8

train_size = int(len(data_selected) * split_ratio)

train_data = data_selected[:train_size]
test_data = data_selected[train_size:]
```

```
# Normalize the data
scaler = MinMaxScaler()
train_scaled = scaler.fit_transform(train_data)
test_scaled = scaler.transform(test_data)

# Prepare the data for LSTM
def create_dataset(X, y, time_steps=1):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        Xs.append(X[i:(i + time_steps)])
        ys.append(y[i + time_steps])
    return np.array(Xs), np.array(ys)

time_steps = 144 # 6: Short Term, 12: Medium Term, 144: Daily cycle
X_train, y_train = create_dataset(train_scaled, train_scaled[:, 0], time_steps)
X_test, y_test = create_dataset(test_scaled, test_scaled[:, 0], time_steps)

# Define the LSTM model architecture
model = Sequential([
    LSTM(units=64, input_shape=(X_train.shape[1], X_train.shape[2])),
    Dense(units=1)
])
model.compile(optimizer='adam', loss='mean_squared_error')

# Define the EarlyStopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the LSTM model
history = model.fit(X_train, y_train, epochs=200, batch_size=32, validation_split=0.1, verbose=1, callb

# Evaluate the model performance
model.evaluate(X_test, y_test)

model.save("lstm_multivariate_model.h5")

# Make predictions
predictions = model.predict(X_test)
```

```
c:\Users\Hrth\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pa
  super().__init__(**kwargs)
Epoch 1/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 27s 21ms/step - loss: 0.0070 - val_loss: 0.0045
Epoch 2/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 21ms/step - loss: 0.0038 - val_loss: 0.0043
Epoch 3/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 21ms/step - loss: 0.0035 - val_loss: 0.0044
Epoch 4/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 21ms/step - loss: 0.0034 - val_loss: 0.0041
Epoch 5/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 21ms/step - loss: 0.0034 - val_loss: 0.0045
Epoch 6/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 21ms/step - loss: 0.0033 - val_loss: 0.0039
Epoch 7/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 21ms/step - loss: 0.0034 - val_loss: 0.0038
Epoch 8/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 21ms/step - loss: 0.0033 - val_loss: 0.0041
Epoch 9/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 24s 20ms/step - loss: 0.0033 - val_loss: 0.0043
Epoch 10/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 21ms/step - loss: 0.0032 - val_loss: 0.0039
Epoch 11/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 21ms/step - loss: 0.0033 - val_loss: 0.0039
Epoch 12/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 21ms/step - loss: 0.0032 - val_loss: 0.0039
Epoch 13/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 21ms/step - loss: 0.0033 - val_loss: 0.0038
```

```
Epoch 14/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 24s 20ms/step - loss: 0.0032 - val_loss: 0.0042
Epoch 15/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 24s 20ms/step - loss: 0.0032 - val_loss: 0.0039
Epoch 16/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 24s 20ms/step - loss: 0.0031 - val_loss: 0.0038
Epoch 17/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 24s 21ms/step - loss: 0.0031 - val_loss: 0.0039
Epoch 18/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 21ms/step - loss: 0.0030 - val_loss: 0.0039
Epoch 19/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 22ms/step - loss: 0.0031 - val_loss: 0.0045
Epoch 20/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 21ms/step - loss: 0.0032 - val_loss: 0.0039
Epoch 21/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 25s 22ms/step - loss: 0.0031 - val_loss: 0.0039
Epoch 22/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 26s 22ms/step - loss: 0.0032 - val_loss: 0.0039
Epoch 23/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 26s 22ms/step - loss: 0.0032 - val_loss: 0.0039
Epoch 24/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 26s 22ms/step - loss: 0.0032 - val_loss: 0.0040
Epoch 25/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 26s 22ms/step - loss: 0.0030 - val_loss: 0.0039
Epoch 26/200
1179/1179 ━━━━━━━━━━━━━━━━━━━━ 26s 22ms/step - loss: 0.0032 - val_loss: 0.0039
324/324 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0052
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_m
324/324 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step
```

```
import matplotlib.pyplot as plt

# Plotting training and validation loss per epoch
plt.plot(history.history['loss'], label='Training Loss', linestyle='-')
plt.plot(history.history['val_loss'], label='Validation Loss', linestyle='--')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss per Epoch')
plt.show()
```




```
y_pred = model.predict(X_test)
y_true = y_test

# Now can print y_true and y_pred
print("Actual values (y_true):", y_true)
print("Predicted values (y_pred):", y_pred)
```

```
324/324 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step
Actual values (y_true): [0. 0. 0. ... 0. 0. 0.]
Predicted values (y_pred): [[0.00237881]
 [0.002421  ]
 [0.00251113]
 ...
 [0.00121984]
 [0.00099874]
 [0.00099327]]
```

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_true, y_pred)
print("Mean Squared Error (MSE):", mse)
```

```
Mean Squared Error (MSE): 0.004495965415773541
```

```
import numpy as np
import pandas as pd

# Flatten the X_test array
X_test_flat = X_test.reshape(X_test.shape[0], -1)

# Convert y_test and y_pred to 1D arrays
y_test_flat = y_test.flatten()
y_pred_flat = y_pred.flatten()

# Convert X_test_flat to DataFrame
X_test_df = pd.DataFrame(X_test_flat, columns=[f'Feature_{i}' for i in range(X_test_flat.shape[1])])

# Create DataFrame for y_test and y_pred
```

```
y_test_df = pd.DataFrame({'Actual values (y_true)': y_test_flat})
y_pred_df = pd.DataFrame({'Predicted values (y_pred)': y_pred_flat})

# Concatenate X_test_df, y_test_df, and y_pred_df along columns
result_df = pd.concat([X_test_df, y_test_df, y_pred_df], axis=1)

# Print the result DataFrame
result_df.head(10)
```

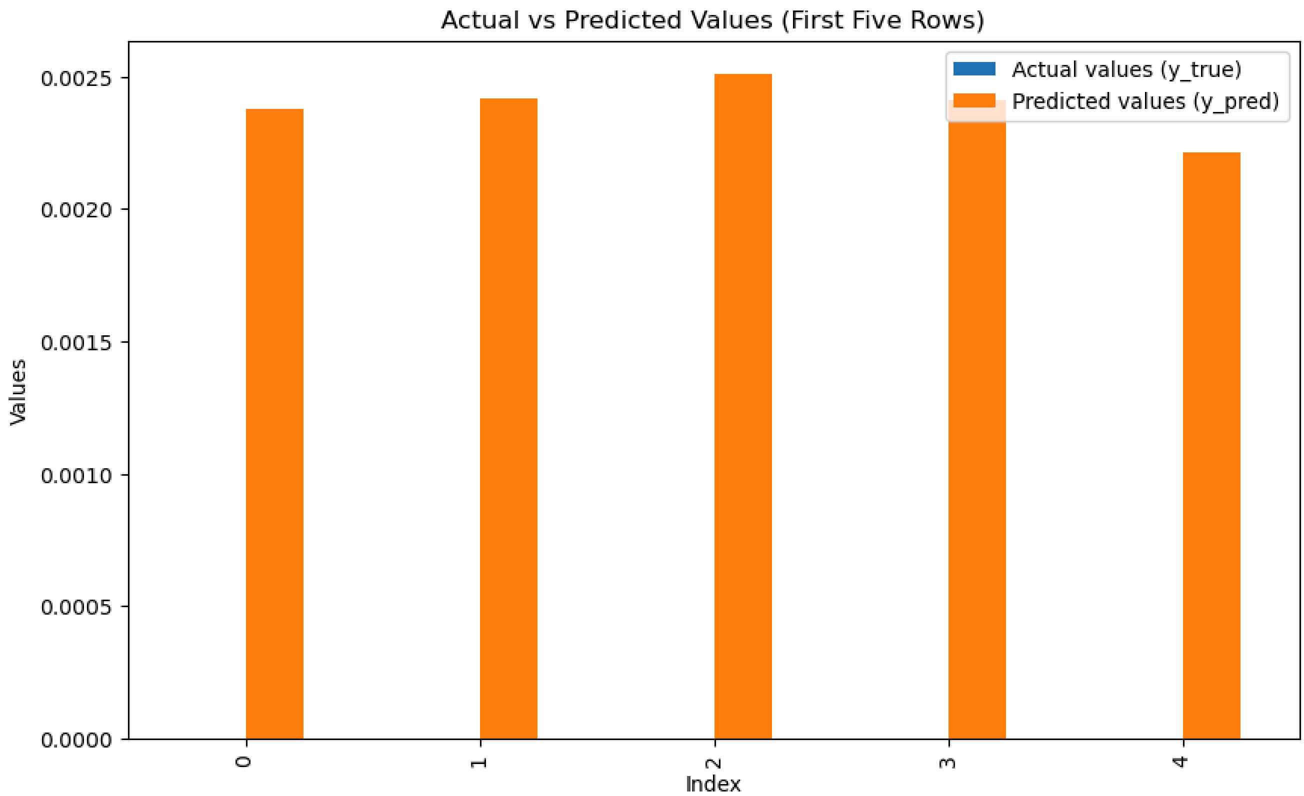
	Feature_0	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Feature_8
0	0.0	0.0	0.153933	0.0	0.0	0.153933	0.0	0.0	0.15487
1	0.0	0.0	0.153933	0.0	0.0	0.154878	0.0	0.0	0.15676
2	0.0	0.0	0.154878	0.0	0.0	0.156766	0.0	0.0	0.15676
3	0.0	0.0	0.156766	0.0	0.0	0.156766	0.0	0.0	0.15487
4	0.0	0.0	0.156766	0.0	0.0	0.154878	0.0	0.0	0.15487
5	0.0	0.0	0.154878	0.0	0.0	0.154878	0.0	0.0	0.15487
6	0.0	0.0	0.154878	0.0	0.0	0.154878	0.0	0.0	0.15676
7	0.0	0.0	0.154878	0.0	0.0	0.156766	0.0	0.0	0.15960
8	0.0	0.0	0.156766	0.0	0.0	0.159600	0.0	0.0	0.16054
9	0.0	0.0	0.159600	0.0	0.0	0.160544	0.0	0.0	0.15960

10 rows × 434 columns

```
result_df = result_df.iloc[:, -2:]
result_df.head()
```

	Actual values (y_true)	Predicted values (y_pred)
0	0.0	0.002379
1	0.0	0.002421
2	0.0	0.002511
3	0.0	0.002414
4	0.0	0.002216

```
import matplotlib.pyplot as plt

# Selecting only the first five rows
result_df_first_five = result_df.iloc[:5]

# Plotting the actual vs predicted values for the first five rows
result_df_first_five.plot(kind='bar', figsize=(10, 6))
plt.title('Actual vs Predicted Values (First Five Rows)')
plt.xlabel('Index')
plt.ylabel('Values')
plt.show()
```




```
from sklearn.metrics import mean_absolute_error, mean_squared_error

y_true = result_df['Actual values (y_true)']
y_pred = result_df['Predicted values (y_pred)']

mae = mean_absolute_error(y_true, y_pred)
mse = mean_squared_error(y_true, y_pred)
rmse = mean_squared_error(y_true, y_pred, squared=False)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
```

```
Mean Absolute Error: 0.031108629943393983
```