Configuration Manual

MSc Research Project
Master's in Data Analytics

Mitali Sopte
Student ID: X22198121

School of Computing
National College of Ireland

Supervisor: Jaswinder Singh

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Mitali Sopte		
Student ID:	X22198121		
Programme:	Master's in Data Analytics	**Year:**	2024
Module:	Msc in Research Project		
Lecturer:	Jaswinder Singh		
Submission Due Date:	12/08/2024		
Project Title:	Solar Sight Forecast: Deep Learning Approaches for Solar PV Power Prediction at Bui Solar Power Station Ghana		
Word Count:	**1182**	**Page Count: 13**	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Mitali Sopte

Date: 12/08/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Mitali Sopte
Student ID: X22198121

1 Introduction

This configuration manual gives us the detailed information of the environmental setup and configuration setup for "Solar Power Generation Prediction Using Machine Learning Techniques". This manual provides the requirement needed to carry out the research project successfully and also gives a step-by-step guidance.
The Configuration setup section and data gathering section gives information about the necessary software and hardware that were required in this project.

1.1 Project Objective

The main Objective of the project is to develop solar power generation prediction using deep learning methods for BUI Solar power station in Ghana. The dataset which consists of wide range of environmental parameters like humidity, wind speed and global irradiation aims to verify the generation of solar power with deep learning models like LSTM. The end goal of the project to offer insightful information that can help the energy management company to effective plan supply the power.

2 Configuration Setup

2.1 Hardware Configuration

The below are the specification of hardware which was configured for this project.

- Memory: 8 GB
- Processor: Apple M1
- System: macOS
- Operating system: 8 GB
- GPU: integrated GPU

2.2 Software Configuration

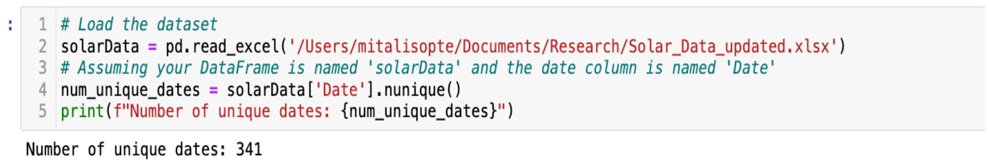
The research project's software setup uses Jupyter Notebook which offer an interactive environment for creating and testing deep leaning models. All the models are coded in Python version 3.11.4. Throughout the research several python libraries are used such as Keras version (from TensorFlow): 3.3.3 for high level neural network API and TensorFlow version: 2.16.1 for creating and training deep leaning models, Matplotlib (3.7.1) is used for visualisation of data. Other Libraries have also been used including NumPy for numerical calculation and pandas for data manipulation and scikit-learn for machine learning tool. Effective data processing and performance evaluation are made possible by using the combination of the above-mentioned tools.

3 Data Gathering

The gathering for this research id obtained from the previous researchers who focused on solar power generation at the BUI Solar Power Station in Ghana. This dataset is very significant as it includes a wide range of factors such as humidity, global irradiation and ambient temperature. With the time interval of 15-minute interval the data comprised of 32736 records and covers a 11 month period of April 25, 2021 till March 31, 2022. By working on the models like LSTM the present study seeks to improve the forecasting capacities of solar power generation by working on the preexisting dataset.

4 Data Preprocessing

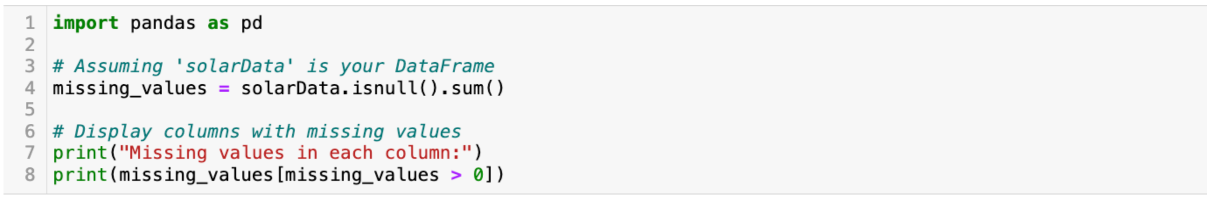
The data which is obtained in then imported in the juypter notebook as below and then the handling missing value and outliers' steps are being carried out. As shown in the Fig 2 missing values are being identified and then handled by time-based interpolation technique.



```
# Load the dataset
solarData = pd.read_excel('/Users/mitalisopte/Documents/Research/Solar_Data_updated.xlsx')
# Assuming your DataFrame is named 'solarData' and the date column is named 'Date'
num_unique_dates = solarData['Date'].nunique()
print(f"Number of unique dates: {num_unique_dates}")
```

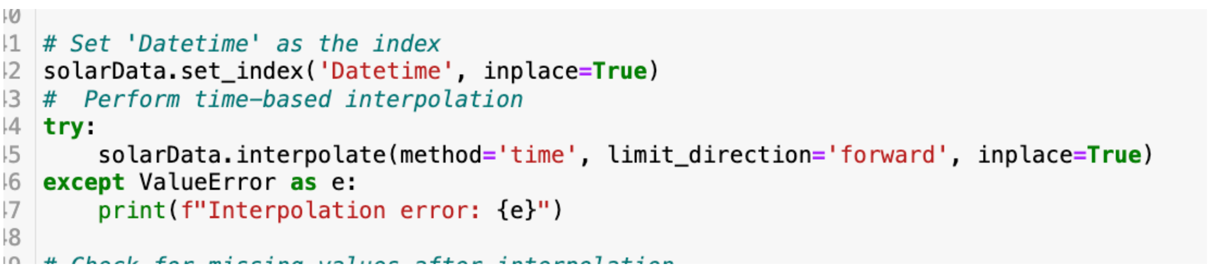
```
Number of unique dates: 341
```

Fig. 1 Importing the data



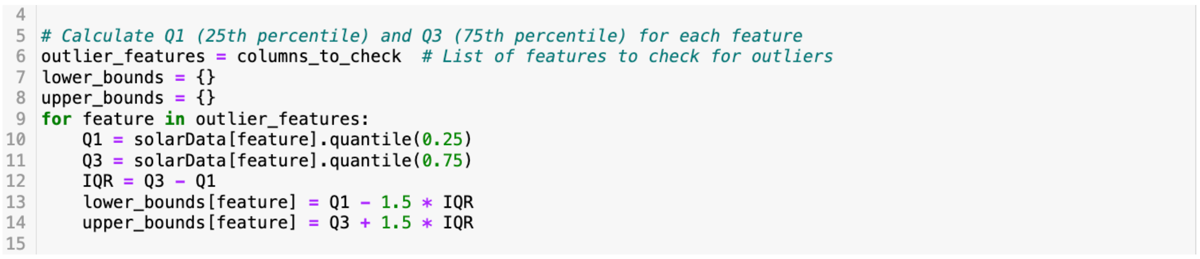
```
import pandas as pd

# Assuming 'solarData' is your DataFrame
missing_values = solarData.isnull().sum()

# Display columns with missing values
print("Missing values in each column:")
print(missing_values[missing_values > 0])
```

Fig. 2 Missing values



```

# Set 'Datetime' as the index
solarData.set_index('Datetime', inplace=True)
#  Perform time-based interpolation
try:
    solarData.interpolate(method='time', limit_direction='forward', inplace=True)
except ValueError as e:
    print(f"Interpolation error: {e}")

```

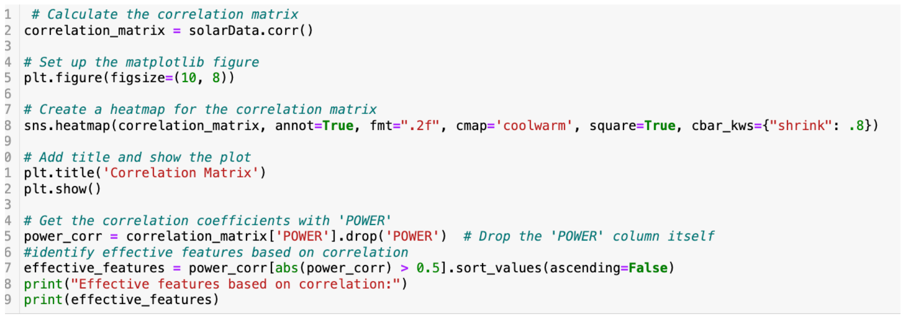
Fig. 3 Time Based Interpolation



```python

# Calculate Q1 (25th percentile) and Q3 (75th percentile) for each feature
outlier_features = columns_to_check  # List of features to check for outliers
lower_bounds = {}
upper_bounds = {}
for feature in outlier_features:
    Q1 = solarData[feature].quantile(0.25)
    Q3 = solarData[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bounds[feature] = Q1 - 1.5 * IQR
    upper_bounds[feature] = Q3 + 1.5 * IQR

```

Fig. 4 Interquartile Range Method

The above fig shows the method to handling the outliers. The outlier are those data points that can cause issue to the analysis and resulting in skewed outcome. So IQR method is applied before the data is applied to the model.

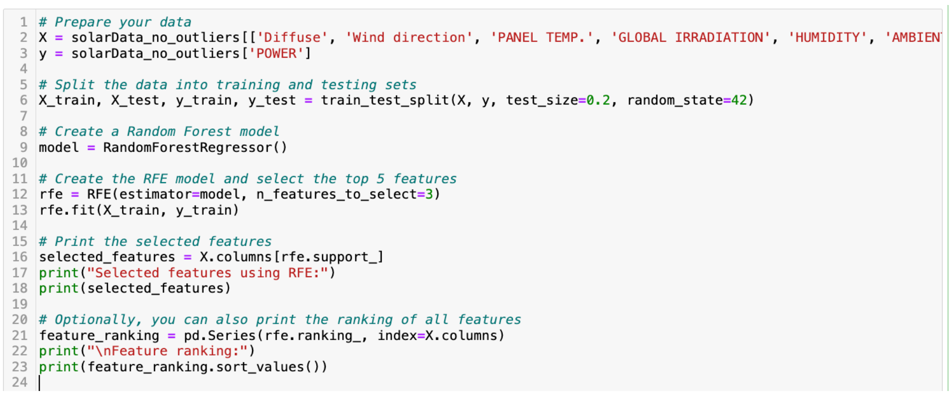


```python
 # Calculate the correlation matrix
correlation_matrix = solarData.corr()

# Set up the matplotlib figure
plt.figure(figsize=(10, 8))

# Create a heatmap for the correlation matrix
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True, cbar_kws={"shrink": .8})

# Add title and show the plot
plt.title('Correlation Matrix')
plt.show()

# Get the correlation coefficients with 'POWER'
power_corr = correlation_matrix['POWER'].drop('POWER')  # Drop the 'POWER' column itself
#identify effective features based on correlation
effective_features = power_corr[abs(power_corr) > 0.5].sort_values(ascending=False)
print("Effective features based on correlation:")
print(effective_features)
```

Fig. 5 Correlation Heat map

5 Feature Selection

The feature selection is done by the Recursive Feature Extraction Method this technique which uses a machine learning model like Random Forest to evaluate the importance of feature



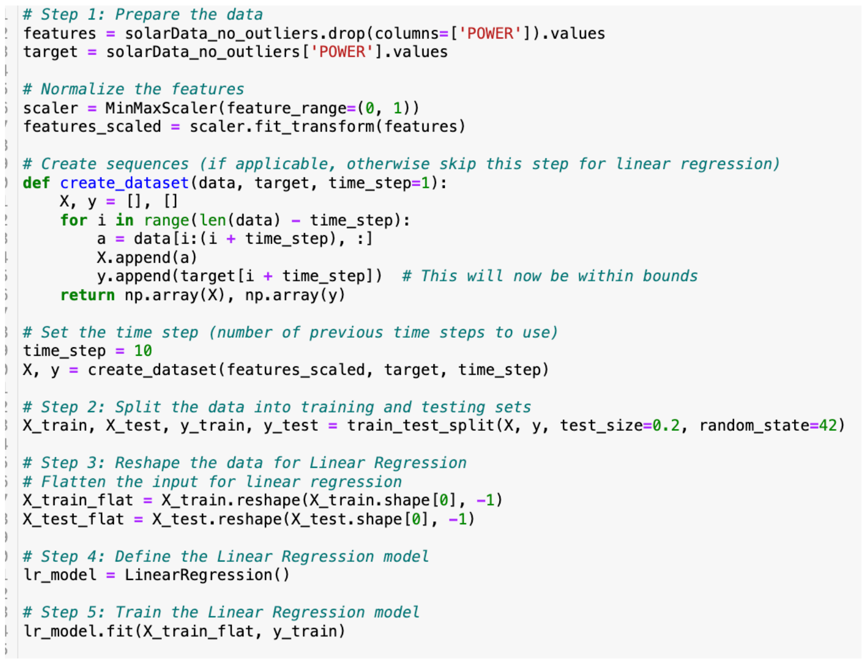
```python
# Prepare your data
X = solarData_no_outliers[['Diffuse', 'Wind direction', 'PANEL TEMP.', 'GLOBAL IRRADIATION', 'HUMIDITY', 'AMBIEN
y = solarData_no_outliers['POWER']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Random Forest model
model = RandomForestRegressor()

# Create the RFE model and select the top 5 features
rfe = RFE(estimator=model, n_features_to_select=3)
rfe.fit(X_train, y_train)

# Print the selected features
selected_features = X.columns[rfe.support_]
print("Selected features using RFE:")
print(selected_features)

# Optionally, you can also print the ranking of all features
feature_ranking = pd.Series(rfe.ranking_, index=X.columns)
print("\nFeature ranking:")
print(feature_ranking.sort_values())

```

Fig.6 Recursive Feature Extraction

6 Implementation

6.1 Linear Regression Base Model

The linear regression model serves as the base model for the further execution. Fig 8 shows the execution results.



```python
# Step 1: Prepare the data
features = solarData_no_outliers.drop(columns=['POWER']).values
target = solarData_no_outliers['POWER'].values

# Normalize the features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features)

# Create sequences (if applicable, otherwise skip this step for linear regression)
def create_dataset(data, target, time_step=1):
    X, y = [], []
    for i in range(len(data) - time_step):
        a = data[i:(i + time_step), :]
        X.append(a)
        y.append(target[i + time_step])  # This will now be within bounds
    return np.array(X), np.array(y)

# Set the time step (number of previous time steps to use)
time_step = 10
X, y = create_dataset(features_scaled, target, time_step)

# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Reshape the data for Linear Regression
# Flatten the input for linear regression
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)

# Step 4: Define the Linear Regression model
lr_model = LinearRegression()

# Step 5: Train the Linear Regression model
lr_model.fit(X_train_flat, y_train)
```

Fig.7 Linear Regression Model

```
Linear Regression RMSE: 4.33, R²: 0.87
```

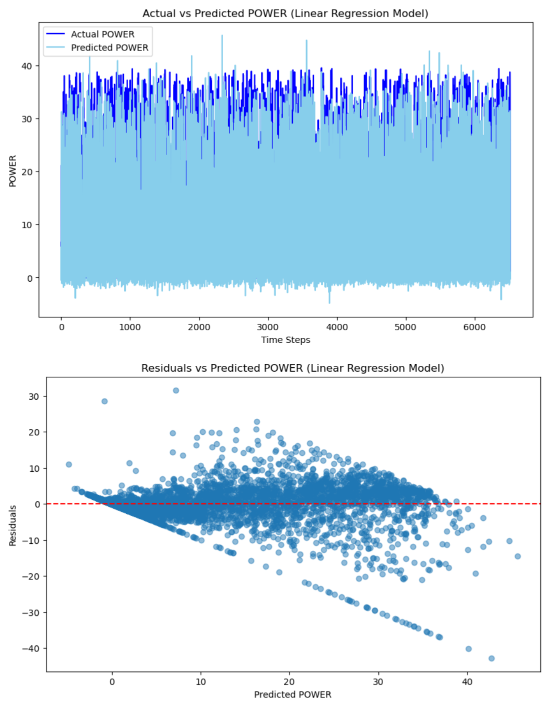
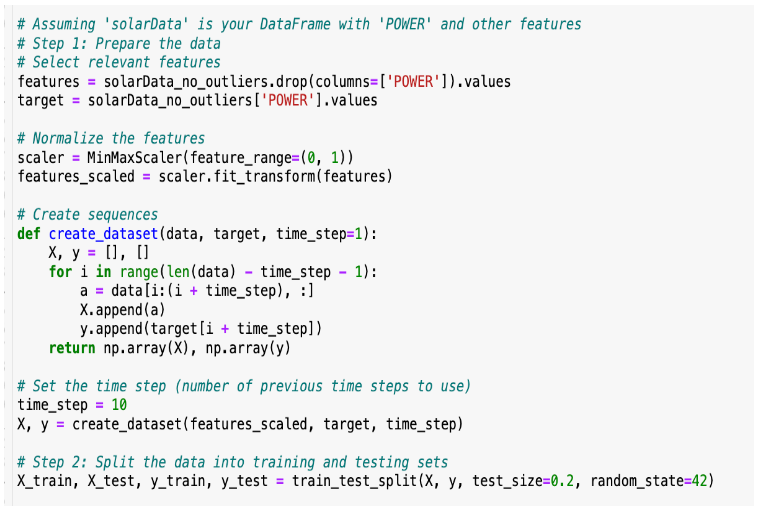


Fig.8 Linear Regression Results

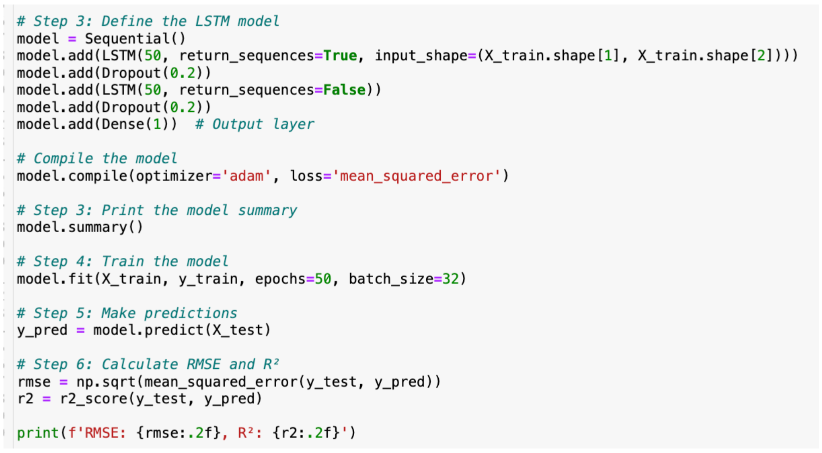
6.2 Models with All Features

The below fig shows the code for the Models like LSMT , CNN- LSTM and Stacked LSTM when all the features are applied. Fig 13 shows the result of stacked LSTM which has lower RMSE and high R2



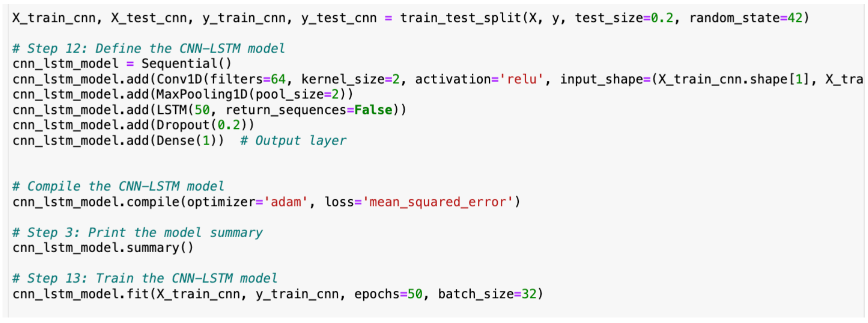
```
# Assuming 'solarData' is your DataFrame with 'POWER' and other features
# Step 1: Prepare the data
# Select relevant features
features = solarData_no_outliers.drop(columns=['POWER']).values
target = solarData_no_outliers['POWER'].values

# Normalize the features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features)

# Create sequences
def create_dataset(data, target, time_step=1):
    X, y = [], []
    for i in range(len(data) - time_step - 1):
        a = data[i:(i + time_step), :]
        X.append(a)
        y.append(target[i + time_step])
    return np.array(X), np.array(y)

# Set the time step (number of previous time steps to use)
time_step = 10
X, y = create_dataset(features_scaled, target, time_step)

# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Fig.9 Sequence creation for All feature dataset



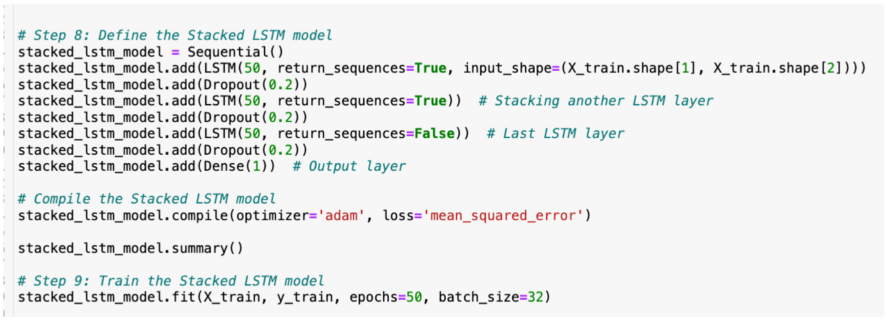
```
# Step 3: Define the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))  # Output layer

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

# Step 3: Print the model summary
model.summary()

# Step 4: Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32)

# Step 5: Make predictions
y_pred = model.predict(X_test)

# Step 6: Calculate RMSE and R²
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f'RMSE: {rmse:.2f}, R²: {r2:.2f}')
```

Fig.10 LSTM Model



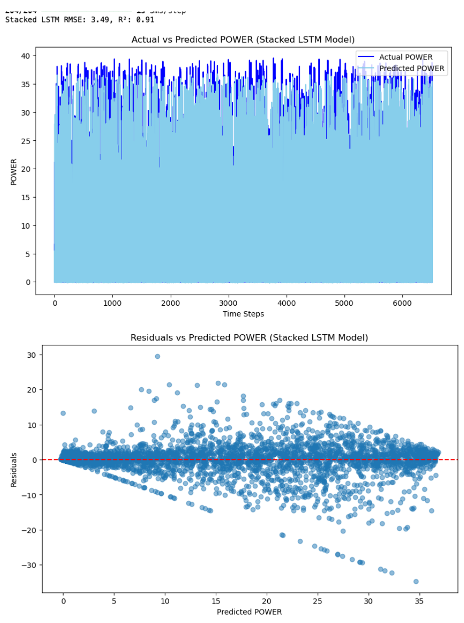
```
X_train_cnn, X_test_cnn, y_train_cnn, y_test_cnn = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 12: Define the CNN-LSTM model
cnn_lstm_model = Sequential()
cnn_lstm_model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(X_train_cnn.shape[1], X_tra
cnn_lstm_model.add(MaxPooling1D(pool_size=2))
cnn_lstm_model.add(LSTM(50, return_sequences=False))
cnn_lstm_model.add(Dropout(0.2))
cnn_lstm_model.add(Dense(1))  # Output layer


# Compile the CNN-LSTM model
cnn_lstm_model.compile(optimizer='adam', loss='mean_squared_error')

# Step 3: Print the model summary
cnn_lstm_model.summary()

# Step 13: Train the CNN-LSTM model
cnn_lstm_model.fit(X_train_cnn, y_train_cnn, epochs=50, batch_size=32)
```

Fig.11 CNN LSTM Model



```
# Step 8: Define the Stacked LSTM model
stacked_lstm_model = Sequential()
stacked_lstm_model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
stacked_lstm_model.add(Dropout(0.2))
stacked_lstm_model.add(LSTM(50, return_sequences=True))  # Stacking another LSTM layer
stacked_lstm_model.add(Dropout(0.2))
stacked_lstm_model.add(LSTM(50, return_sequences=False))  # Last LSTM layer
stacked_lstm_model.add(Dropout(0.2))
stacked_lstm_model.add(Dense(1))  # Output layer

# Compile the Stacked LSTM model
stacked_lstm_model.compile(optimizer='adam', loss='mean_squared_error')

stacked_lstm_model.summary()

# Step 9: Train the Stacked LSTM model
stacked_lstm_model.fit(X_train, y_train, epochs=50, batch_size=32)
```

Fig.12 Stacked LSTM Model

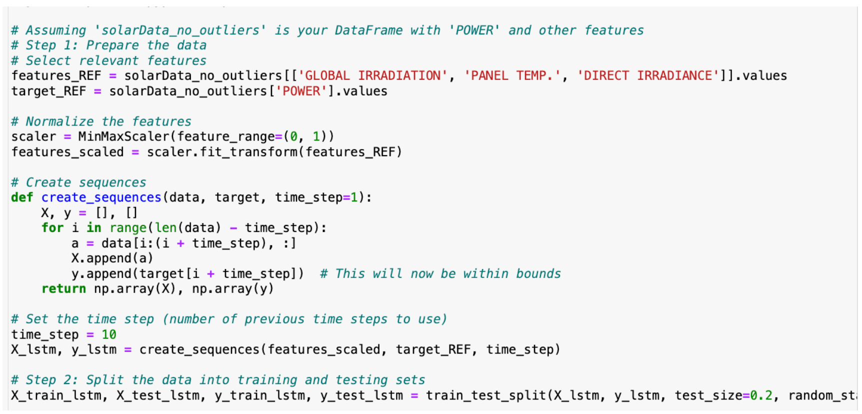


```
Stacked LSTM RMSE: 3.49, R²: 0.91
```

Fig.13 Stacked LSTM Model Results

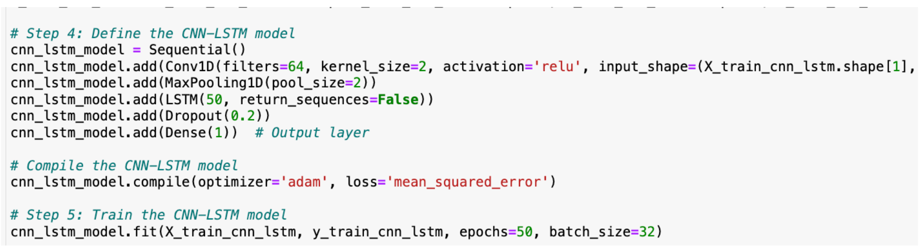
6.3 Model with 3 Features

The below fig shows the code for the Models like LSMT , CNN- LSTM and Stacked LSTM when only 3 of the features are applied. Fig 17 shows the result of stacked LSTM which has lower RMSE and high R2



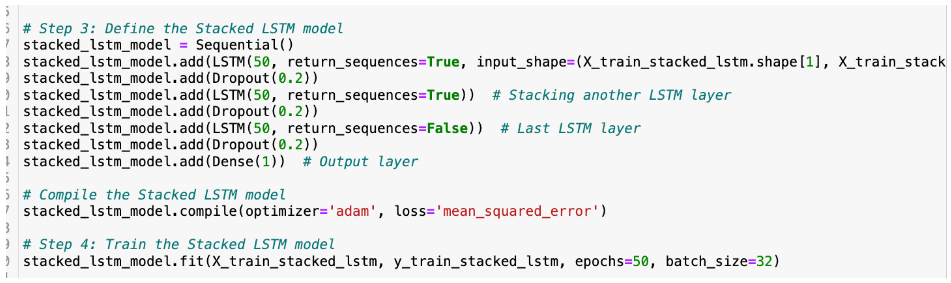
```
# Assuming 'solarData_no_outliers' is your DataFrame with 'POWER' and other features
# Step 1: Prepare the data
# Select relevant features
features_REF = solarData_no_outliers[['GLOBAL IRRADIATION', 'PANEL TEMP.', 'DIRECT IRRADIANCE']].values
target_REF = solarData_no_outliers['POWER'].values

# Normalize the features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features_REF)

# Create sequences
def create_sequences(data, target, time_step=1):
    X, y = [], []
    for i in range(len(data) - time_step):
        a = data[i:(i + time_step), :]
        X.append(a)
        y.append(target[i + time_step])  # This will now be within bounds
    return np.array(X), np.array(y)

# Set the time step (number of previous time steps to use)
time_step = 10
X_lstm, y_lstm = create_sequences(features_scaled, target_REF, time_step)

# Step 2: Split the data into training and testing sets
X_train_lstm, X_test_lstm, y_train_lstm, y_test_lstm = train_test_split(X_lstm, y_lstm, test_size=0.2, random_st
```

Fig.14 LSTM Model



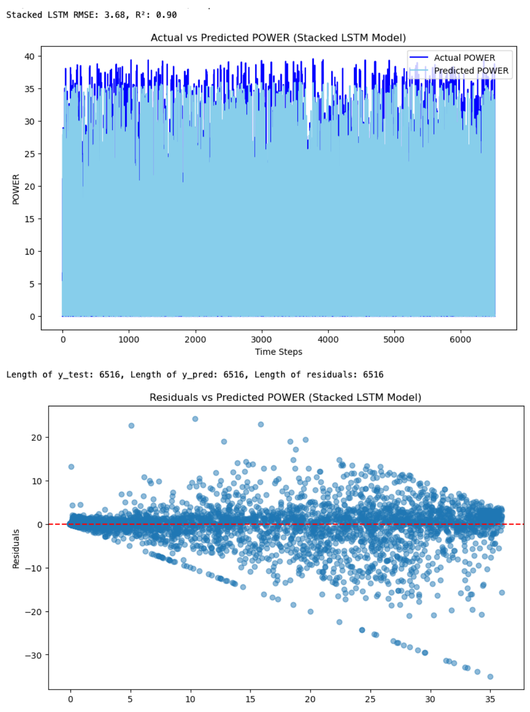
```
# Step 4: Define the CNN-LSTM model
cnn_lstm_model = Sequential()
cnn_lstm_model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(X_train_cnn_lstm.shape[1],
cnn_lstm_model.add(MaxPooling1D(pool_size=2))
cnn_lstm_model.add(LSTM(50, return_sequences=False))
cnn_lstm_model.add(Dropout(0.2))
cnn_lstm_model.add(Dense(1))  # Output layer

# Compile the CNN-LSTM model
cnn_lstm_model.compile(optimizer='adam', loss='mean_squared_error')

# Step 5: Train the CNN-LSTM model
cnn_lstm_model.fit(X_train_cnn_lstm, y_train_cnn_lstm, epochs=50, batch_size=32)
```

Fig.15 LSTM Model



```
# Step 3: Define the Stacked LSTM model
stacked_lstm_model = Sequential()
stacked_lstm_model.add(LSTM(50, return_sequences=True, input_shape=(X_train_stacked_lstm.shape[1], X_train_stack
stacked_lstm_model.add(Dropout(0.2))
stacked_lstm_model.add(LSTM(50, return_sequences=True))  # Stacking another LSTM layer
stacked_lstm_model.add(Dropout(0.2))
stacked_lstm_model.add(LSTM(50, return_sequences=False))  # Last LSTM layer
stacked_lstm_model.add(Dropout(0.2))
stacked_lstm_model.add(Dense(1))  # Output layer

# Compile the Stacked LSTM model
stacked_lstm_model.compile(optimizer='adam', loss='mean_squared_error')

# Step 4: Train the Stacked LSTM model
stacked_lstm_model.fit(X_train_stacked_lstm, y_train_stacked_lstm, epochs=50, batch_size=32)
```

Fig.16 Stacked LSTM Model



```
Stacked LSTM RMSE: 3.68, R²: 0.90
```

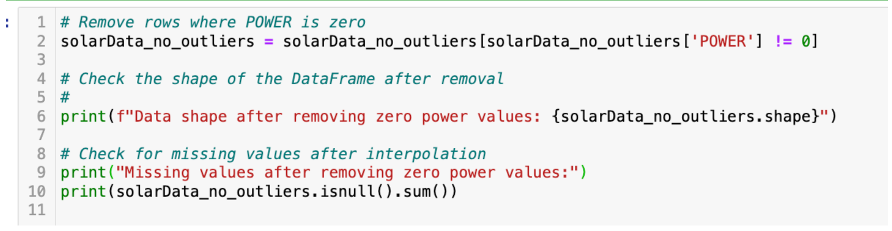
```
Length of y_test: 6516, Length of y_pred: 6516, Length of residuals: 6516
```

Fig.17 Stacked LSTM Model Results

6.4 Model with no 0 data values with All Feature

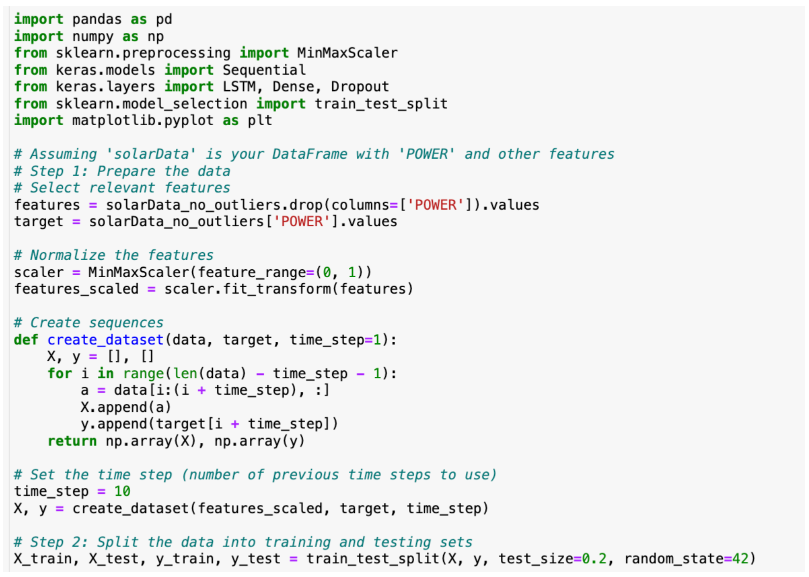
The below fig shows the code for the Models like LSMT , CNN- LSTM and Stacked LSTM when 0 data values form the power are removed and all the features are applied. Fig 23 shows the result of stacked LSTM which has lower RMSE and high R2

Power removing 0 with all feature



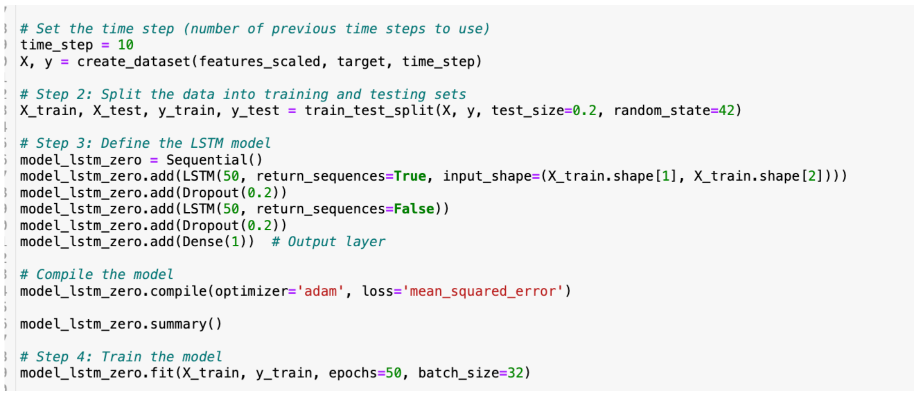
```
# Remove rows where POWER is zero
solarData_no_outliers = solarData_no_outliers[solarData_no_outliers['POWER'] != 0]

# Check the shape of the DataFrame after removal
#
print(f"Data shape after removing zero power values: {solarData_no_outliers.shape}")

# Check for missing values after interpolation
print("Missing values after removing zero power values:")
print(solarData_no_outliers.isnull().sum())

```

Fig.18 Removing 0 data form power



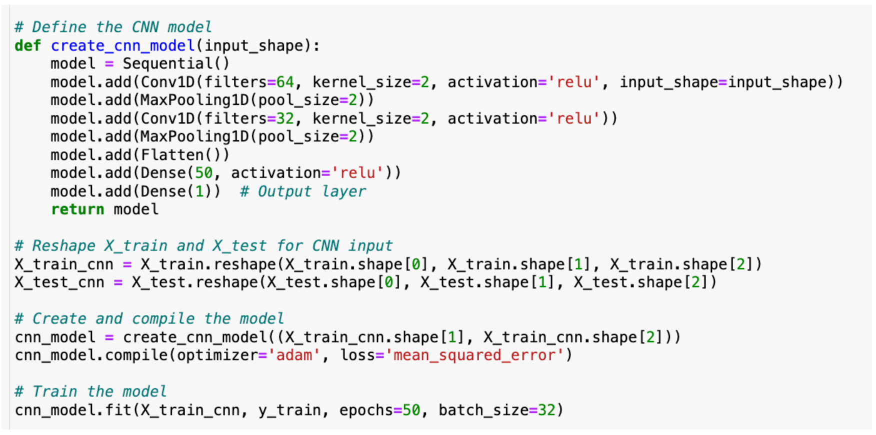
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Assuming 'solarData' is your DataFrame with 'POWER' and other features
# Step 1: Prepare the data
# Select relevant features
features = solarData_no_outliers.drop(columns=['POWER']).values
target = solarData_no_outliers['POWER'].values

# Normalize the features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features)

# Create sequences
def create_dataset(data, target, time_step=1):
    X, y = [], []
    for i in range(len(data) - time_step - 1):
        a = data[i:(i + time_step), :]
        X.append(a)
        y.append(target[i + time_step])
    return np.array(X), np.array(y)

# Set the time step (number of previous time steps to use)
time_step = 10
X, y = create_dataset(features_scaled, target, time_step)

# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Fig.19 Sequence creation for All feature dataset



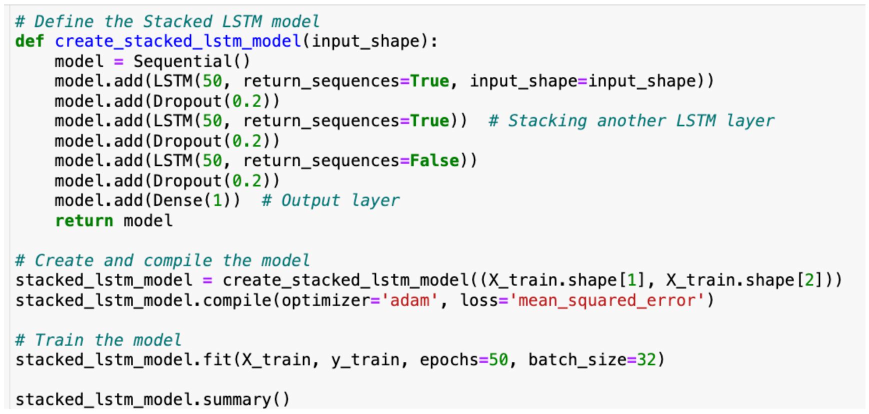
```
# Set the time step (number of previous time steps to use)
time_step = 10
X, y = create_dataset(features_scaled, target, time_step)

# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Define the LSTM model
model_lstm_zero = Sequential()
model_lstm_zero.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model_lstm_zero.add(Dropout(0.2))
model_lstm_zero.add(LSTM(50, return_sequences=False))
model_lstm_zero.add(Dropout(0.2))
model_lstm_zero.add(Dense(1))  # Output layer

# Compile the model
model_lstm_zero.compile(optimizer='adam', loss='mean_squared_error')

model_lstm_zero.summary()

# Step 4: Train the model
model_lstm_zero.fit(X_train, y_train, epochs=50, batch_size=32)
```

Fig.20 LSTM Model



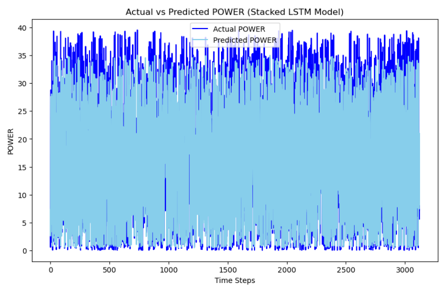
```python
# Define the CNN model
def create_cnn_model(input_shape):
    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=input_shape))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Conv1D(filters=32, kernel_size=2, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(50, activation='relu'))
    model.add(Dense(1))  # Output layer
    return model

# Reshape X_train and X_test for CNN input
X_train_cnn = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2])
X_test_cnn = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2])

# Create and compile the model
cnn_model = create_cnn_model((X_train_cnn.shape[1], X_train_cnn.shape[2]))
cnn_model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
cnn_model.fit(X_train_cnn, y_train, epochs=50, batch_size=32)
```

Fig.21 CNN-LSTM Model



```python
# Define the Stacked LSTM model
def create_stacked_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(50, return_sequences=True, input_shape=input_shape))
    model.add(Dropout(0.2))
    model.add(LSTM(50, return_sequences=True))  # Stacking another LSTM layer
    model.add(Dropout(0.2))
    model.add(LSTM(50, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(1))  # Output layer
    return model

# Create and compile the model
stacked_lstm_model = create_stacked_lstm_model((X_train.shape[1], X_train.shape[2]))
stacked_lstm_model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
stacked_lstm_model.fit(X_train, y_train, epochs=50, batch_size=32)

stacked_lstm_model.summary()
```

Fig.22 Stacked LSTM Model



```
Stacked LSTM RMSE: 4.37, R²: 0.87
Length of y_test: 3123, Length of y_pred: 3123, Length of residuals: 3123
```

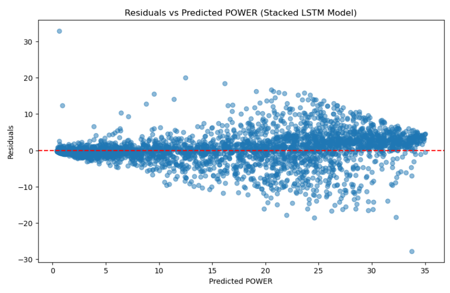
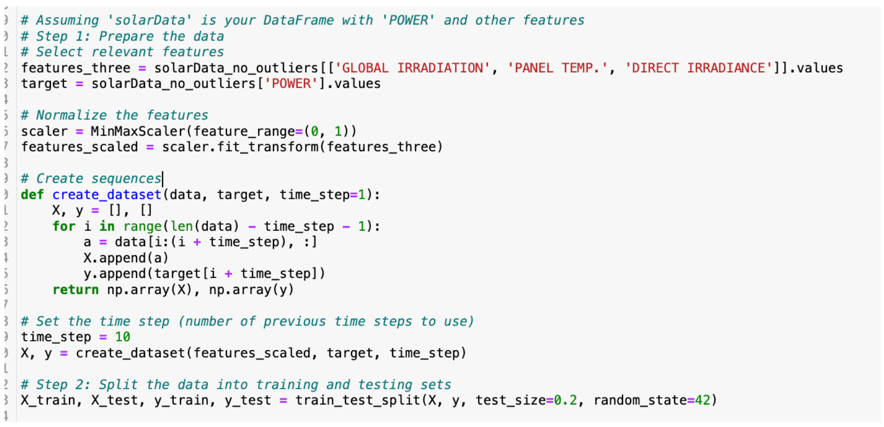


Fig.23 Stacked LSTM Model Results

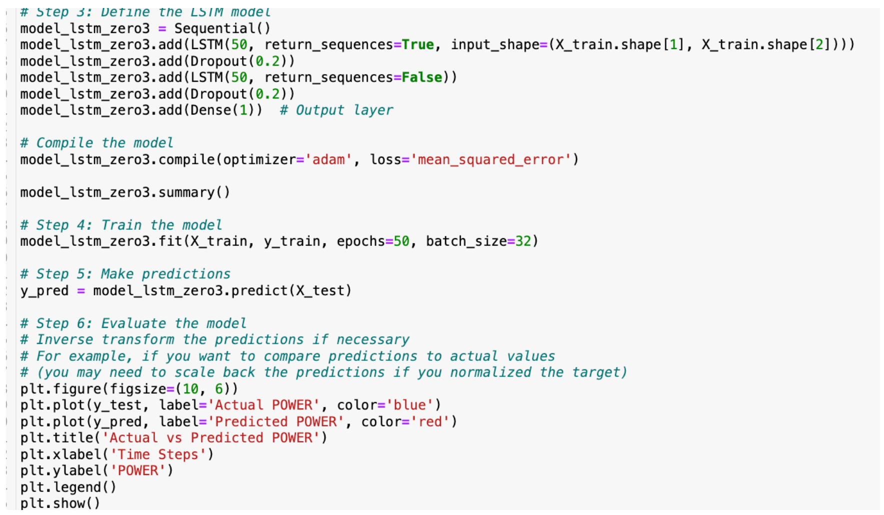
6.5 Model with no 0 data values with 3 Features

The below fig shows the code for the Models like LSMT , CNN- LSTM and Stacked LSTM when 0 data values form the power are removed and only 3 of the features are applied. Fig 28 shows the result of stacked LSTM which has lower RMSE and high R2



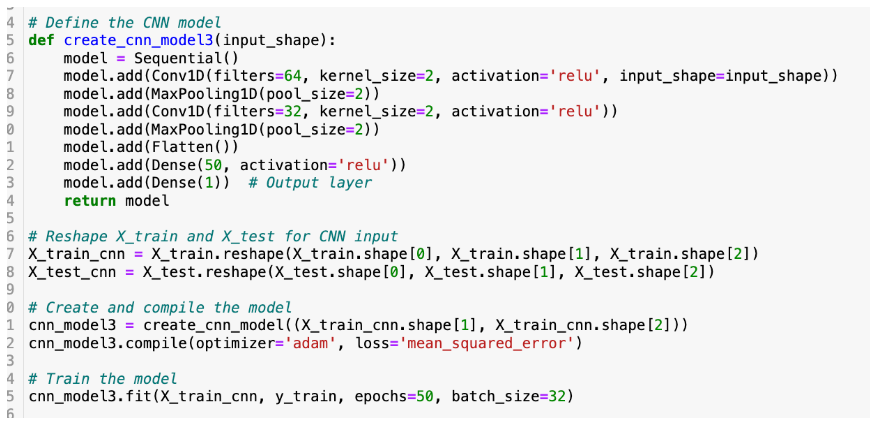
```python
# Assuming 'solarData' is your DataFrame with 'POWER' and other features
# Step 1: Prepare the data
# Select relevant features
features_three = solarData_no_outliers[['GLOBAL IRRADIATION', 'PANEL TEMP.', 'DIRECT IRRADIANCE']].values
target = solarData_no_outliers['POWER'].values

# Normalize the features
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features_three)

# Create sequences
def create_dataset(data, target, time_step=1):
    X, y = [], []
    for i in range(len(data) - time_step - 1):
        a = data[i:(i + time_step), :]
        X.append(a)
        y.append(target[i + time_step])
    return np.array(X), np.array(y)

# Set the time step (number of previous time steps to use)
time_step = 10
X, y = create_dataset(features_scaled, target, time_step)

# Step 2: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Fig.24 Sequence creation for 3 feature dataset



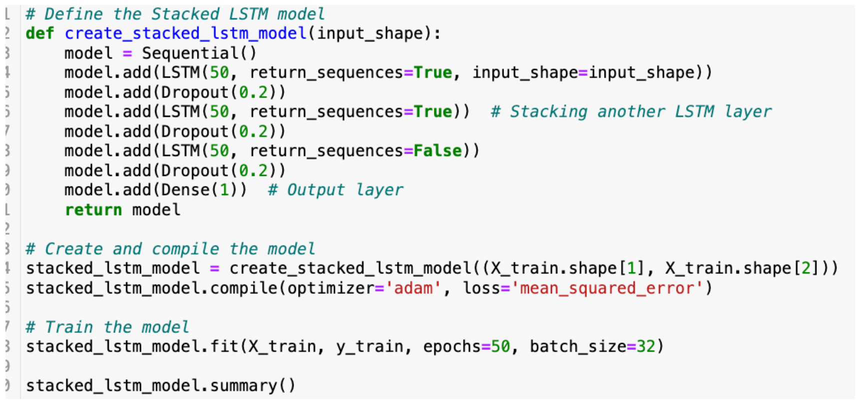
```python
# Step 3: Define the LSTM model
model_lstm_zero3 = Sequential()
model_lstm_zero3.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model_lstm_zero3.add(Dropout(0.2))
model_lstm_zero3.add(LSTM(50, return_sequences=False))
model_lstm_zero3.add(Dropout(0.2))
model_lstm_zero3.add(Dense(1))  # Output layer

# Compile the model
model_lstm_zero3.compile(optimizer='adam', loss='mean_squared_error')

model_lstm_zero3.summary()

# Step 4: Train the model
model_lstm_zero3.fit(X_train, y_train, epochs=50, batch_size=32)

# Step 5: Make predictions
y_pred = model_lstm_zero3.predict(X_test)

# Step 6: Evaluate the model
# Inverse transform the predictions if necessary
# For example, if you want to compare predictions to actual values
# (you may need to scale back the predictions if you normalized the target)
plt.figure(figsize=(10, 6))
plt.plot(y_test, label='Actual POWER', color='blue')
plt.plot(y_pred, label='Predicted POWER', color='red')
plt.title('Actual vs Predicted POWER')
plt.xlabel('Time Steps')
plt.ylabel('POWER')
plt.legend()
plt.show()
```

Fig.25 LSTM Model



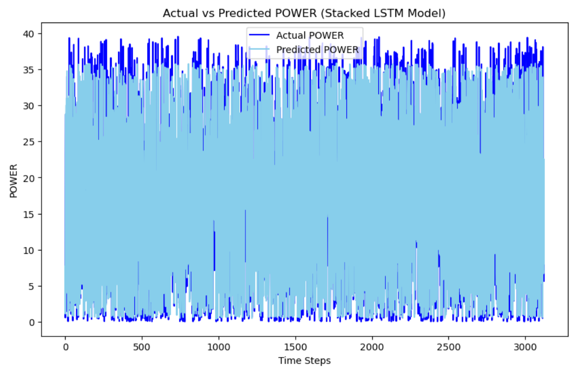
```python
# Define the CNN model
def create_cnn_model3(input_shape):
    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=input_shape))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Conv1D(filters=32, kernel_size=2, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(50, activation='relu'))
    model.add(Dense(1))  # Output layer
    return model

# Reshape X_train and X_test for CNN input
X_train_cnn = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2])
X_test_cnn = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2])

# Create and compile the model
cnn_model3 = create_cnn_model((X_train_cnn.shape[1], X_train_cnn.shape[2]))
cnn_model3.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
cnn_model3.fit(X_train_cnn, y_train, epochs=50, batch_size=32)
```

Fig.26 CNN LSTM Model

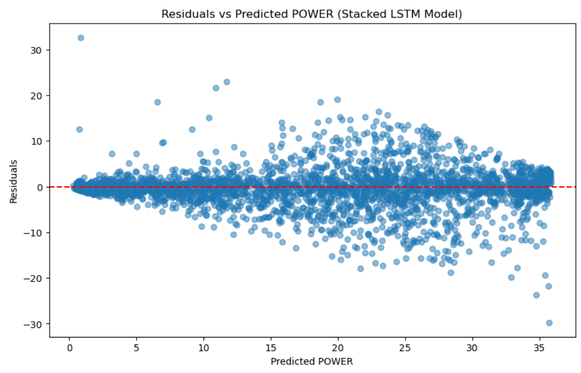


```python
# Define the Stacked LSTM model
def create_stacked_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(50, return_sequences=True, input_shape=input_shape))
    model.add(Dropout(0.2))
    model.add(LSTM(50, return_sequences=True))  # Stacking another LSTM layer
    model.add(Dropout(0.2))
    model.add(LSTM(50, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(1))  # Output layer
    return model

# Create and compile the model
stacked_lstm_model = create_stacked_lstm_model((X_train.shape[1], X_train.shape[2]))
stacked_lstm_model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
stacked_lstm_model.fit(X_train, y_train, epochs=50, batch_size=32)

stacked_lstm_model.summary()
```

Fig.27 Stacked LSTM Model



```
Length of y_test: 3123, Length of y_pred: 3123, Length of residuals: 3123
```



F

Fig.28 Stacked LSTM Model Results

References

K. Pun, S. M. Basnet and W. Jewell, "Solar Power Prediction in Different Forecasting Horizons Using Machine Learning and Time Series Techniques," 2021 IEEE Conference on Technologies for Sustainability (SusTech), Irvine, CA, USA, 2021, pp. 1-7, doi: 10.1109/SusTech51236.2021.9467464. keywords: {Renewable energy sources;Time series analysis;Machine learning;Solar energy;Predictive models;Performance analysis;Solar power generation;Renewable Energy Resources;Forecasting;Machine Learning;Time Series},