



ECEN421: Introduction to Computer Networks

Energy Harvesting Prediction for Networks using ML/DL

Omar Samir Mohamed (211000372)

Mahmoud Reda Abdel Rauf (211000489)

Marwan Mohamed Zaki (211001365)

Abdullah El khateeb (211001730)

Nile University

Dr Mohamed Saeed Darweesh

ECEN424

Fall 2024

Abstract

Energy harvesting is gaining attention as a sustainable solution for reducing energy consumption in smart homes and networks. This paper presents a predictive model for energy harvesting using solar power to meet the energy demands of smart homes. We combine solar energy data with energy usage data from smart homes, including temperature and humidity variables, to predict the amount of energy harvested. Machine learning and deep learning techniques, including Random Forest, XGBoost, Catboost, Feedforward Neural Network(FNN) are employed to model and evaluate the relationship between weather conditions, energy usage, and solar energy. Our findings suggest that machine learning and deep learning models can significantly predict the potential for energy harvesting, providing valuable insights for optimizing energy consumption in smart homes.

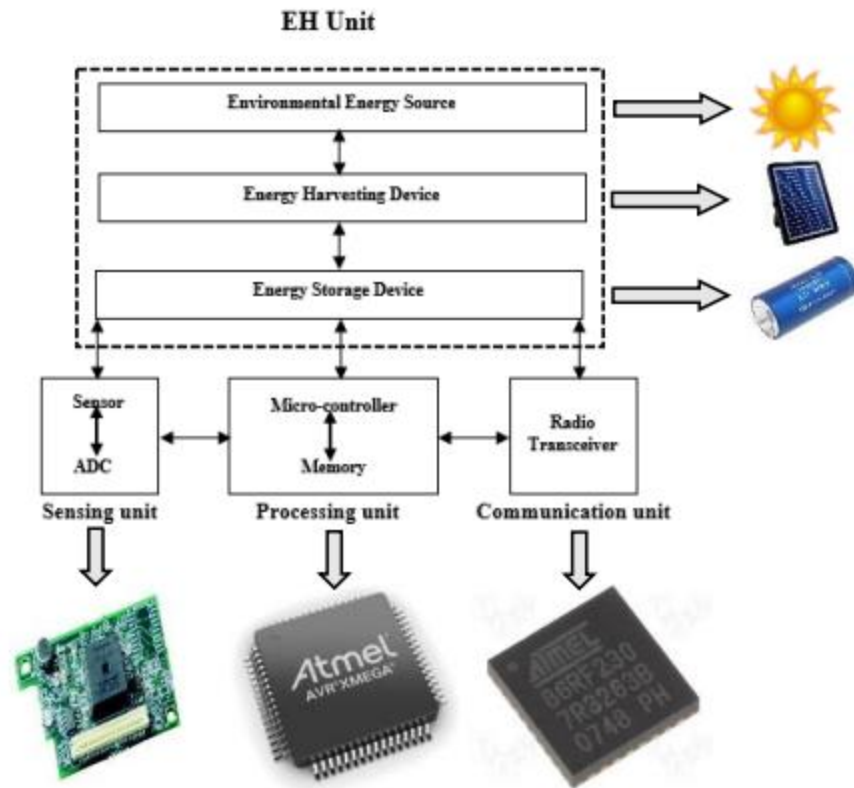
Introduction

The global shift toward renewable energy sources, particularly solar power, has made significant progress in recent years. In parallel, smart home technologies are becoming more widespread, providing improved control and optimization of energy consumption. However, one of the key challenges in smart home systems is managing the integration of renewable energy sources, such as solar energy, with the fluctuating energy demands of homes. Energy harvesting, the process of capturing and storing energy from renewable sources, has the potential to address this challenge by enabling homes to reduce dependency on the grid and make use of locally generated solar power.

Accurate prediction of energy harvesting potential is crucial for optimizing energy storage and consumption in smart homes. By combining solar energy data with factors such as temperature and humidity, machine learning and deep learning techniques offer a promising approach to forecast energy harvesting capacity and assess its effectiveness in meeting energy demands. This paper explores the application of predictive models to estimate energy harvesting potential from solar energy in smart homes. We aim to develop and evaluate models that can predict the amount of energy harvested, using historical energy usage data and environmental conditions as key inputs.

The primary objectives of this project are as follows:

1. **Data Integration:** To merge solar energy data with energy usage data from smart homes, including environmental factors such as temperature and humidity.
2. **Predictive Modeling:** To apply machine learning and deep learning techniques to model and predict the relationship between weather conditions, energy consumption patterns, and solar energy generation.
3. **Model Evaluation:** To assess the performance of various models using appropriate evaluation metrics, such as RMSE, MAE, and R^2 , and compare their effectiveness in predicting energy harvesting potential.



Data Description

This study utilizes three distinct datasets with the same date and time [1]: **solar energy generation data**, **smart home energy usage data** and **weather data**. The solar energy dataset provides historical information on solar power generation, the smart home energy usage dataset contains energy consumption data for homes, and the weather data contains temperature and humidity.

1. Solar energy data

- **Date/Time:** start date = '2012-06-01' end date = '2013-05-31'
- **Solar Power Generation:** The amount of electricity from solar panels based on the solar irradiance. (AC and DC)

2. Smart Home Energy Usage Data

- **Date/Time:** start date = '2012-06-01' end date = '2013-05-31'
- **Energy Consumption:** The total amount of energy consumed by the smart home during a given time period.

3. The Weather Data

- **Date/Time:** start date = '2012-06-01' end date = '2013-05-31'
- **Temperature:** in Celsius
- **Humidity**
- **Weather:** (Sunny, Cloudy...)

Data Preprocessing

1. Weather Data Filtering and Merging

Weather data was filtered for the period from June 1, 2012, to May 31, 2013, and merged with the energy usage data based on the date and hour columns.

2. Merging Data

The solar energy and consumption data were merged on the date and hour columns, creating a combined dataset that includes both energy consumption and solar energy production.

3. Categorizing Energy and Weather

Energy consumption and solar outputs were categorized as Low, Medium, or High based on predefined thresholds. Weather was categorized as Sunny, Cloudy, or Rainy.

4. Defining Time of Day and Energy State

The hour column was used to define time periods: Morning, Afternoon, and Evening. A new state column was created combining the time of day, energy category, and weather category.

5. Feature Engineering and Encoding

Categorical features, such as weather, were one-hot encoded. Numerical features (temperature, humidity, hour) were scaled using StandardScaler for consistent input to models.

6. Normalization of Numerical Features

Numerical features were normalized to ensure they contributed equally to machine learning models.

Methodology

In this Project, four models were used to predict energy consumption and solar energy generation: Random Forest (RF), XGBoost, and Feedforward Neural Network (FNN). Each model has distinct characteristics and was chosen for its ability to handle different aspects of the energy prediction problem.

1. Random Forest (RF)

Random Forest is an ensemble learning method based on decision trees. It constructs multiple decision trees during training and merges their results to improve prediction accuracy and control overfitting. Random Forest is robust to outliers and can handle both regression and classification tasks. In this study, it was used to predict energy consumption and solar power output by learning from the features like weather, temperature, humidity, and time of day.

Advantages: Handles large datasets well, reduces overfitting by averaging multiple decision trees, and is interpretable.

Application: Used for both predicting energy consumption (energy_kWh) and solar power outputs (ac_output, dc_output).

2. XGBoost

XGBoost (Extreme Gradient Boosting) is a powerful implementation of gradient boosting, a machine learning technique that builds models sequentially. Each new model corrects errors made by previous models. XGBoost is known for its high performance and scalability, especially for large datasets and complex problems.

Advantages: High predictive accuracy, handles missing data well, and performs efficiently with large datasets.

Application: Applied to predict the energy consumption and solar power outputs, providing a strong baseline for comparison with other models.

3. Feedforward Neural Network (FNN)

A Feedforward Neural Network (FNN) is a type of artificial neural network where the information moves in one direction from input to output, without cycles. It consists of an input layer, one or more hidden layers, and an output layer. FNNs are capable of learning complex nonlinear relationships between features.

Advantages: Can model complex, nonlinear relationships and capture intricate patterns in the data.

Application: Used to predict energy consumption and solar output, with the ability to capture non-linear patterns in weather, time of day, and energy consumption data.

Model Implementation and Evaluation

Each model was trained on the same preprocessed dataset, with features such as weather, temperature, humidity, and time of day, and targets like energy consumption and solar outputs. The performance of these models was evaluated using metrics like **Mean Absolute Error (MAE)**, **Root Mean Squared Error (MSE)**, and **R-squared (R²)** to assess prediction accuracy.

1. Mean Absolute Error (MAE)

- **Definition:** MAE measures the average absolute difference between predicted values and actual values. It indicates how far predictions are, on average, from actual outcomes. The MAE value 0 represents perfect fit and the value of MAE is always greater than or equal to 0.
- **Equation:**

The diagram shows the MAE equation: $MAE = \frac{1}{n} \sum |y - \hat{y}|$. Annotations include: 'Divide by the total number of data points' pointing to $\frac{1}{n}$; 'Actual output value' pointing to y ; 'Predicted output value' pointing to \hat{y} ; 'Sum of' pointing to the summation symbol \sum ; and 'The absolute value of the residual' pointing to the absolute value bars $|y - \hat{y}|$.

2. Mean Squared Error (MSE)

- **Definition:** MSE calculates the average squared difference between predicted and actual values. Squaring makes larger errors, making MSE sensitive to outliers. . The MSE value 0 represents perfect fit and the value of MSE is always greater than or equal to 0
- **Equation:**

$$MSE = \frac{1}{n} \sum \underbrace{(y - \hat{y})^2}_{\text{The square of the difference between actual and predicted}}$$

3. Coefficient of Determination (R^2)

- **Definition:** R² evaluates how well the model explains the variability of the actual data. It's a normalized metric between 0 and 1, with higher values indicating a better fit.

- **Equation:** $R^2 = 1$: Perfect fit.

$$R^2 = 1 - \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y})^2}$$

These models offer complementary strengths: **Random Forest** and **XGBoost** excel in handling structured data and feature importance, while **FNN** is capable of capturing complex, non-linear, and temporal patterns.

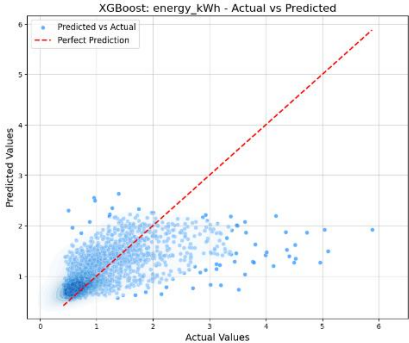
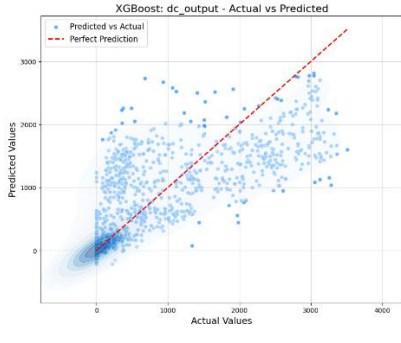
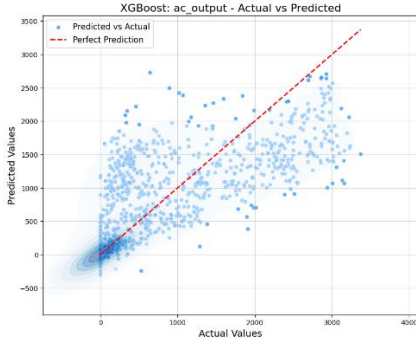
Results

In this section, we present the results of the four models: Random Forest (RF), XGBoost, and Feedforward Neural Network (FNN) .Each model's performance was evaluated based on the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R^2) metrics, which are commonly used to assess the accuracy and efficiency of regression models.

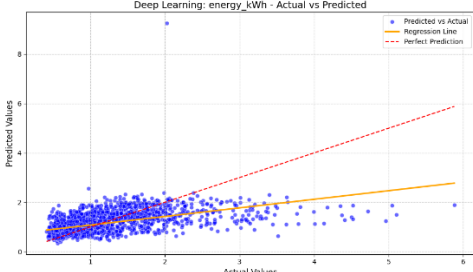
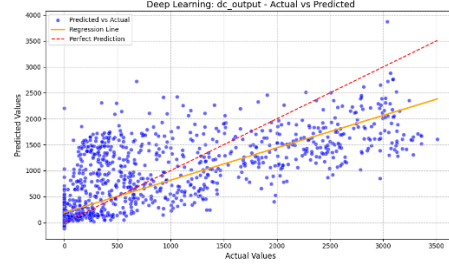
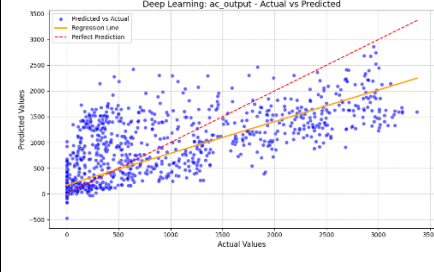
1. Random Forest (RF)

Metric	energy_kWh	dc_output	ac_output
MAE	50.953%	30.837%	34.5%
MSE	45.55%	36.54%	35.87%
R^2	0.3379	0.6356	0.6204

2. XGBoost

Metric	energy_kWh	dc_output	ac_output
MAE	48.57%	37.90%	37.77%
MSE	40.45%	35.31%	34.49%
R ²	0.3602	0.6356	0.6315
			

3. Feedforward Neural Network (FNN)

Metric	energy_kWh	dc_output	ac_output
MAE	59.53%	37.88%	31.12%
MSE	49.91%	30.99%	29.007%
R ²	0.3379	0.6008	0.6257
			

Limitations

The predictions generated by the models were not highly accurate due to several data-related challenges:

1. **Limited Data Availability:** The dataset used for training and testing the models was relatively small, which restricted the model's ability to generalize and capture complex patterns in the data.
2. **Alignment of Dates Across Data Sources:** A significant challenge was finding datasets with overlapping and common dates. This limitation resulted in reduced compatibility between different datasets, affecting the quality of features used for modeling.
3. **Data Quality Issues:** The available data was not of optimal quality, with potential inconsistencies, missing values, or noise that hindered the model's performance.
4. **Limited Model Differentiation:** Due to the mentioned data challenges, the predictions of the different models were often close to each other, suggesting that the models struggled to capture distinct and nuanced patterns in the data.

Ineffective Use of Unmerged Data: I attempted to use the data without merging it, but the results were poor. The lack of cohesive features

```
➡ Training Random Forest model for energy_kWh...  
  
R2 Score for energy_kWh: 0.3402
```

These limitations underscore the need for larger, cleaner, and more cohesive datasets to improve prediction accuracy in future studies.

Conclusion

This study aimed to predict energy consumption and solar energy generation using advanced machine learning models, including Random Forest (RF), XGBoost, Feedforward Neural Network (FNN). By leveraging a comprehensive dataset containing features like weather conditions, temperature, humidity, and time of day, the models were trained and evaluated using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared (R^2).

However, **the predictions for certain cases were not accurate across all models.** The lack of precise and diverse data, such as finer-grained temporal details or additional features like appliance-level energy consumption and localized solar radiation patterns, limited the ability of the models to distinguish subtle patterns. This resulted in models producing similar outcomes due to insufficient variation in the training data.

Future Work

- **Better Data Collection:** Acquiring high-quality and diverse data, such as real-time IoT sensor readings, appliance-level usage data, and detailed historical weather records, is essential to improve predictions and enhance model differentiation.
- **Enhanced Features:** The inclusion of additional relevant features, such as solar panel specifications, cloud cover data, and electricity grid information, can improve prediction accuracy.
- **Hybrid Models:** Combining machine learning models, such as an ensemble of XGBoost and CNN, may leverage their complementary strengths for better predictions.

In conclusion, while the models demonstrated potential, their performance was constrained by data limitations. Future efforts should focus on obtaining richer and more granular datasets to unlock the full potential of machine learning for optimizing energy consumption and solar energy utilization.

Appendix

1. Importing Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import MinMaxScaler, StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import lightgbm as lgb
import xgboost as xgb
from catboost import CatBoostRegressor
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

2. Loading the Data

```
weather_data = pd.read_csv('Weather.csv')
energy_usage_data = pd.read_csv('energy.csv')
solar_energy_data = pd.read_csv('Solar.csv')
```

3. Data Preprocessing

```
[6] start_date = '2012-06-01'
    end_date = '2013-05-31'

    # Filter weather data based on the date range
    filtered_weather_data = weather_data[(weather_data['date'] >= start_date) & (weather_data['date'] <= end_date)]

    # Merge weather and energy usage data
    consumption_data = pd.merge(energy_usage_data, filtered_weather_data, on=['date', 'hour'], how='inner')
```

```
num_rows = len(solar_energy_data)
days_required = num_rows // 24 # Total number of days (rows divided by 24 hours per day)
new_dates = pd.date_range(start=start_date, periods=days_required + 1, freq='D')

# Repeat each date 24 times to match the hour count
date_column = [date.strftime('%Y-%m-%d') for date in new_dates for _ in range(24)][0:num_rows]

# Update the date column and save the modified dataset
solar_energy_data['date'] = date_column
solar_energy_data.to_csv('solar_energy_data_updated.csv', index=False)
```

```
[8] merged_data = pd.merge(consumption_data, solar_energy_data, on=['date', 'hour'], how='inner')
    merged_data
```

```
0s # Categorize energy levels
def categorize_energy(energy):
    if energy < 1.0:
        return 'Low'
    elif energy < 2.0:
        return 'Medium'
    else:
        return 'High'

# Categorize weather
def categorize_weather(weather):
    if 'Rain' in weather:
        return 'Rainy'
    elif 'Cloud' in weather:
        return 'Cloudy'
    else:
        return 'Sunny'

def define_state(row):
    if 6 <= row['hour'] < 12:
        time_of_day = 'Morning'
    elif 12 <= row['hour'] < 18:
        time_of_day = 'Afternoon'
    else:
        time_of_day = 'Evening'

    return f"{time_of_day}, {row['energy_category']}, {row['weather_category']}"

merged_data['energy_category'] = merged_data['energy_kWh'].apply(categorize_energy)
merged_data['ac_category'] = merged_data['ac_output'].apply(categorize_energy)
merged_data['dc_category'] = merged_data['dc_output'].apply(categorize_energy)
merged_data['weather_category'] = merged_data['weather'].apply(categorize_weather)
merged_data['state'] = merged_data.apply(define_state, axis=1)
actions = ['Low', 'Medium', 'High']
```

0a

```
# Define features and targets
categorical_columns = ['weather']
numerical_columns = ['temperature', 'humidity', 'hour']
targets = ['energy_kwh', 'ac_output', 'dc_output']

# Split features and targets
features = merged_data[categorical_columns + numerical_columns]
target_data = merged_data[targets]

# One-hot encode categorical features
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
encoded_cats = encoder.fit_transform(features[categorical_columns])
encoded_cats_df = pd.DataFrame(encoded_cats, columns=encoder.get_feature_names_out(categorical_columns))

# Drop original categorical columns and concatenate encoded features
features = features.drop(columns=categorical_columns)
features = pd.concat([features.reset_index(drop=True), encoded_cats_df.reset_index(drop=True)], axis=1)

# Normalize numerical features
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
```

7a

```
# Best Hyperparameters for Random Forest After testing different Hypertuning arameters
best_params = {
    'max_depth': 1000,
    'max_features': 'sqrt',
    'min_samples_leaf': 1,
    'min_samples_split': 10,
    'n_estimators': 200,
}

for target in targets:
    print(f"Training Random Forest model for {target}...")

    X_train, X_test, y_train, y_test = train_test_split(
        features_scaled, target_data[target], test_size=0.2, random_state=42
    )

    rf = RandomForestRegressor(
        max_depth=best_params['max_depth'],
        max_features=best_params['max_features'],
        min_samples_leaf=best_params['min_samples_leaf'],
        min_samples_split=best_params['min_samples_split'],
        n_estimators=best_params['n_estimators'],
        random_state=42
    )

    r2 = r2_score(y_test, rf_pred)
    mae = mean_absolute_error(y_test, rf_pred) # Calculate MAE
    mape = np.mean(np.abs((y_test - rf_pred) / y_test)) * 100
    rf.fit(X_train, y_train)
    rf_pred = rf.predict(X_test)
    r2 = r2_score(y_test, rf_pred)

    results[target] = {
        "Random Forest": {
            "R2": r2,
            "MAE": mae,
            "MAPE": mape,
            "predictions": rf_pred,
            "actuals": y_test
        }
    }

# Plotting predicted vs actual values
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=rf_pred, color='blue', label='Predicted vs Actual')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--', label='Perfect Prediction')

# Titles and labels
plt.title(f"Random Forest Predicted vs Actual for {target}")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.legend()
plt.grid(True)
plt.show()

# Display R² results
for target, metrics in results.items():
    print(f"R² Score for {target}: {metrics['Random Forest']['R2']:.4f}")
    print(f"Mean Absolute Error (MAE) for {target}: {metrics['Random Forest']['MAE']:.4f}")
    print(f"Mean Absolute Percentage Error (MAPE) for {target}: {metrics['Random Forest']['MAPE']:.4f}%")
```

```

results = {}

# Loop through each target and train an XGBoost model
for target in targets:
    print(f"Training XGBoost model for {target}...")

    # Split data into training and testing sets for each target
    X_train, X_test, y_train, y_test = train_test_split(
        features_scaled, target_data[target], test_size=0.2, random_state=42
    )

    # Create and train the XGBoost model
    model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, learning_rate=0.1, max_depth=5)
    model.fit(X_train, y_train)

    # Make predictions
    xgb_pred = model.predict(X_test)

    # Evaluate the model
    mae = mean_absolute_error(y_test, xgb_pred)
    mse = mean_squared_error(y_test, xgb_pred)
    r2 = r2_score(y_test, xgb_pred)

    # Store results
    results[target] = {
        "XGBoost": {
            "MAE": mae,
            "MSE": mse,
            "R2": r2
        }
    }

# Plot results (Actual vs Predicted) with improvements
plt.figure(figsize=(10, 8))

# Scatter plot with transparency to handle overlapping points
sns.scatterplot(x=y_test, y=xgb_pred, alpha=0.7, color='dodgerblue', label='Predicted vs Actual')

# Add density contours
sns.kdeplot(x=y_test, y=xgb_pred, cmap='Blues', fill=True, alpha=0.5, levels=10)

# Add a line for perfect prediction (y = x)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', linewidth=2, label='Perfect Prediction')

# Set plot labels and title
plt.xlabel('Actual Values', fontsize=14)
plt.ylabel('Predicted Values', fontsize=14)
plt.title(f"XGBoost: {target} - Actual vs Predicted", fontsize=16)
plt.legend(fontsize=12)
plt.grid(alpha=0.5)

# Show the plot
plt.show()

# Print results
print(f"Results for {target}:")
print(f" MAE: {mae:.4f}")
print(f" MSE: {mse:.4f}")
print(f" R²: {r2:.4f}")

```

✓
2m

```
results = {}

# Define the model
def build_model(input_dim):
    model = Sequential()
    model.add(Dense(64, input_dim=input_dim, activation='relu')) # First hidden layer
    model.add(Dense(32, activation='relu')) # Second hidden layer
    model.add(Dense(16, activation='relu')) # Third hidden layer
    model.add(Dense(1)) # Output layer (no activation for regression)
    model.compile(optimizer=Adam(), loss='mean_squared_error')
    return model

# Loop through each target and train a deep learning model
for target in targets:
    print(f"Training Deep Learning model for {target}...")

    # Split data into training and testing sets for each target
    X_train, X_test, y_train, y_test = train_test_split(
        features_scaled, target_data[target], test_size=0.2, random_state=42
    )

    # Build and train the model
    model = build_model(X_train.shape[1])
    model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

    # Make predictions
    dl_pred = model.predict(X_test)

    # Evaluate the model
    mae = mean_absolute_error(y_test, dl_pred)
    mse = mean_squared_error(y_test, dl_pred)
    r2 = r2_score(y_test, dl_pred)

    # Store results
    results[target] = {
        "Deep Learning": {
            "MAE": mae,
            "MSE": mse,
            "R2": r2
        }
    }

# Plot results (Actual vs Predicted)
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=dl_pred.flatten(), color='blue', label='Predicted vs Actual')

# Add a line for perfect prediction (y = x)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', label='Perfect Prediction')

# Set plot labels and title
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title(f"Deep Learning: {target} - Actual vs Predicted")
plt.legend()
plt.show()

# Print results
print(f"Results for {target}:")
print(f" MAE: {mae:.4f}")
print(f" MSE: {mse:.4f}")
print(f" R2: {r2:.4f}")
```

References

1. <https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/N3HGRN&version=5.0>
2. <https://www.mdpi.com/2079-9292/12/20/4304>
3. https://www.researchgate.net/publication/307913736_A_New_Energy_Prediction_Algorithm_for_Energy-Harvesting_Wireless_Sensor_Networks_With_Q-Learning
4. <http://sefidian.com/2022/08/18/a-guide-on-regression-error-metrics-with-python-code/>
5. <https://www.sciencedirect.com/topics/computer-science/mean-absolute-error>