1. \*\*Title:\*\* Improving Smart Building Energy Efficiency with ANN-Based Predictions

- \*\*P - Problem:\*\* Existing energy consumption prediction methods in smart buildings lack precision, impacting resource utilization and sustainability goals.

- \*\*I - Intervention:\*\* Implementing Artificial Neural Networks (ANN) for enhanced energy consumption predictions.

- \*\*C - Comparison:\*\* Conducting a comparative analysis between ANN and Convolutional Long Short-Term Memory (ConvLSTM) algorithms in the context of smart building energy prediction.

- \*\*O - Outcome:\*\* Evaluating the impact of ANN on prediction accuracy and aiming for a significant improvement over ConvLSTM-based approaches.

2. \*\*Title:\*\* Optimizing Appliance Scheduling in Smart Buildings: ANN vs. SVM Algorithms Comparison

- \*\*P - Problem:\*\* Inefficient scheduling of appliances contributes to increased energy consumption and costs in smart buildings.

- \*\*I - Intervention:\*\* Employing Artificial Neural Networks (ANN) to optimize the start and end times of appliances, aiming for energy cost reduction.

- \*\*C - Comparison:\*\* Comparing the performance of ANN-based scheduling against Support Vector Machine (SVM) algorithms and traditional scheduling approaches.

- \*\*O - Outcome:\*\* Assessing the effectiveness of ANN-based scheduling in achieving energy cost reduction and optimal appliance usage within smart buildings.

3. \*\*Title:\*\* Time-Series Energy Forecasting in Smart Buildings: ANN vs. RNN Algorithms

- \*\*P - Problem:\*\* Existing time-series models for predicting smart building energy consumption face challenges in capturing complex patterns and dependencies.

- \*\*I - Intervention:\*\* Exploring Artificial Neural Networks (ANN) for enhancing the accuracy of time-series energy consumption forecasts.

- \*\*C - Comparison:\*\* Comparing the performance of ANN against Recurrent Neural Network (RNN) algorithms in the context of smart building energy prediction.

- \*\*O - Outcome:\*\* Evaluating the predictive power and robustness of ANN in capturing intricate patterns and dependencies within smart building energy consumption data.

4. \*\*Title:\*\* Smart Building Energy Efficiency: ANN vs. Random Forest Algorithms

- \*\*P - Problem:\*\* Current energy efficiency initiatives in smart buildings are hindered by inaccurate predictions and suboptimal resource utilization.

- \*\*I - Intervention:\*\* Utilizing Artificial Neural Networks (ANN) as a main algorithm for energy consumption prediction.

- \*\*C - Comparison:\*\* Contrasting the performance of ANN against Random Forest algorithms and other standalone models to identify the most effective approach.

- \*\*O - Outcome:\*\* Assessing the impact of ANN on energy efficiency, aiming for significant improvements in accuracy and resource utilization within smart buildings.

# Import necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestRegressor

from sklearn.svm import SVR

from sklearn.neighbors import KNeighborsRegressor

from sklearn.metrics import mean\_squared\_error

import xgboost as xgb

from google.colab import files

# Upload your CSV file

uploaded = files.upload()

# Load the data

data = pd.read\_csv(next(iter(uploaded)))  # Use the first uploaded file

# Print the column names

print(data.columns)

# Replace 'Evaluated Annual Electric Savings (kWh)' with the actual target variable name

target\_variable = 'Evaluated Annual Electric Savings (kWh)'

# Assuming your dataset has features and a target variable

# Exclude non-numeric columns from standardization

non\_numeric\_columns = ['Project ID', 'Project County', 'Project City', 'Project Completion Date', 'Customer Type', 'Location 1']

X = data.drop([target\_variable] + non\_numeric\_columns, axis=1)

y = data[target\_variable]

# Convert categorical variables to numeric if any

X = pd.get\_dummies(X)

# Fill missing values with the mean

X = X.fillna(X.mean())

# 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)

# Standardize the numeric features

numeric\_columns = X.select\_dtypes(include=['float64', 'int64']).columns

scaler = StandardScaler()

X\_train[numeric\_columns] = scaler.fit\_transform(X\_train[numeric\_columns])

X\_test[numeric\_columns] = scaler.transform(X\_test[numeric\_columns])

# Initialize models

rf\_model = RandomForestRegressor(random\_state=42)

svm\_model = SVR()

knn\_model = KNeighborsRegressor()

xgb\_model = xgb.XGBRegressor(random\_state=42)

# Train models

rf\_model.fit(X\_train, y\_train)

svm\_model.fit(X\_train, y\_train)

knn\_model.fit(X\_train, y\_train)

xgb\_model.fit(X\_train, y\_train)

# Make predictions

rf\_pred = rf\_model.predict(X\_test)

svm\_pred = svm\_model.predict(X\_test)

knn\_pred = knn\_model.predict(X\_test)

xgb\_pred = xgb\_model.predict(X\_test)

# Evaluate models

rf\_accuracy = mean\_squared\_error(y\_test, rf\_pred)

svm\_accuracy = mean\_squared\_error(y\_test, svm\_pred)

knn\_accuracy = mean\_squared\_error(y\_test, knn\_pred)

xgb\_accuracy = mean\_squared\_error(y\_test, xgb\_pred)

# Print the test accuracies

print(f"Random Forest Test Accuracy: {rf\_accuracy}")

print(f"SVM Test Accuracy: {svm\_accuracy}")

print(f"KNN Test Accuracy: {knn\_accuracy}")

print(f"XGBoost Test Accuracy: {xgb\_accuracy}")

# Print the R-squared value for each model

print(f"Random Forest R-squared: {rf\_model.score(X\_test, y\_test)}")

print(f"SVM R-squared: {svm\_model.score(X\_test, y\_test)}")

print(f"KNN R-squared: {knn\_model.score(X\_test, y\_test)}")

print(f"XGBoost R-squared: {xgb\_model.score(X\_test, y\_test)}")

# Import necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.svm import SVR

from google.colab import files

# Upload your CSV file

uploaded = files.upload()

# Load the data

data = pd.read\_csv(next(iter(uploaded)))

# Replace 'Evaluated Annual Electric Savings (kWh)' with the actual target variable name

target\_variable = 'Evaluated Annual Electric Savings (kWh)'

# Exclude non-numeric columns from standardization

non\_numeric\_columns = ['Project ID', 'Project County', 'Project City', 'Project Completion Date', 'Customer Type', 'Location 1']

X = data.drop([target\_variable] + non\_numeric\_columns, axis=1)

y = data[target\_variable]

# Convert categorical variables to numeric if any

X = pd.get\_dummies(X)

# Fill missing values with the mean or another appropriate strategy

X = X.fillna(X.mean())

# Convert data types

X = X.astype('float32')

y = y.astype('float32')

# 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)

# Standardize the numeric features for ANN

numeric\_columns = X.select\_dtypes(include=['float32']).columns

scaler\_ann = StandardScaler()

X\_train[numeric\_columns] = scaler\_ann.fit\_transform(X\_train[numeric\_columns])

X\_test[numeric\_columns] = scaler\_ann.transform(X\_test[numeric\_columns])

# Fill NaN values in the target variable for training set for ANN

y\_train = y\_train.fillna(y\_train.mean())

# Fill NaN values in the target variable for testing set for ANN

y\_test = y\_test.fillna(y\_test.mean())

# Initialize the ANN model

ann\_model = Sequential()

ann\_model.add(Dense(64, input\_dim=X\_train.shape[1], activation='relu'))

ann\_model.add(Dense(32, activation='relu'))

ann\_model.add(Dense(1))

ann\_model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the ANN model

ann\_model.fit(X\_train, y\_train, epochs=50, batch\_size=32, verbose=1)

# Make predictions using the ANN model

ann\_pred = ann\_model.predict(X\_test)

# Evaluate the ANN model

ann\_accuracy = mean\_squared\_error(y\_test, ann\_pred)

print(f"ANN Test Mean Squared Error: {ann\_accuracy}")

# Calculate percentage error for ANN

mean\_value = y\_test.mean()

ann\_percentage\_error = (ann\_accuracy / mean\_value) \* 100

print(f"ANN Test Mean Squared Error (Percentage): {ann\_percentage\_error}%")

# Standardize the features for SVM

scaler\_svm = StandardScaler()

X\_train\_svm = scaler\_svm.fit\_transform(X\_train)

X\_test\_svm = scaler\_svm.transform(X\_test)

# Fill NaN values in the target variable for training set for SVM

y\_train\_svm = y\_train.fillna(y\_train.mean())

# Fill NaN values in the target variable for testing set for SVM

y\_test\_svm = y\_test.fillna(y\_test.mean())

# Initialize the SVM model with hyperparameter tuning

param\_grid = {'C': [0.1, 1, 10, 100], 'gamma': [0.01, 0.1, 1, 'auto']}

svm\_model\_tuned = GridSearchCV(SVR(), param\_grid, cv=3)

svm\_model\_tuned.fit(X\_train\_svm, y\_train\_svm)

# Get the best hyperparameters

best\_params = svm\_model\_tuned.best\_params\_

print(f"Best Hyperparameters: {best\_params}")

# Make predictions using the tuned SVM model

svm\_pred\_tuned = svm\_model\_tuned.predict(X\_test\_svm)

# Evaluate the tuned SVM model

svm\_accuracy\_tuned = mean\_squared\_error(y\_test\_svm, svm\_pred\_tuned)

print(f"Tuned SVM Test Mean Squared Error: {svm\_accuracy\_tuned}")

# Calculate percentage error for the tuned SVM

svm\_percentage\_error\_tuned = (svm\_accuracy\_tuned / mean\_value) \* 100

print(f"Tuned SVM Test Mean Squared Error (Percentage): {svm\_percentage\_error\_tuned}%")

# Import necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, SimpleRNN

from google.colab import files

# Upload your CSV file

uploaded = files.upload()

# Load the data

data = pd.read\_csv(next(iter(uploaded)))

# Replace 'Evaluated Annual Electric Savings (kWh)' with the actual target variable name

target\_variable = 'Evaluated Annual Electric Savings (kWh)'

# Exclude non-numeric columns from standardization

non\_numeric\_columns = ['Project ID', 'Project County', 'Project City', 'Project Completion Date', 'Customer Type', 'Location 1']

X = data.drop([target\_variable] + non\_numeric\_columns, axis=1)

y = data[target\_variable]

# Convert categorical variables to numeric if any

X = pd.get\_dummies(X)

# Fill missing values with the mean or another appropriate strategy

X = X.fillna(X.mean())

# Convert data types

X = X.astype('float32')

y = y.astype('float32')

# 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)

# Standardize the numeric features for ANN

numeric\_columns = X.select\_dtypes(include=['float32']).columns

scaler\_ann = StandardScaler()

X\_train[numeric\_columns] = scaler\_ann.fit\_transform(X\_train[numeric\_columns])

X\_test[numeric\_columns] = scaler\_ann.transform(X\_test[numeric\_columns])

# Fill NaN values in the target variable for training set for ANN

y\_train = y\_train.fillna(y\_train.mean())

# Fill NaN values in the target variable for testing set for ANN

y\_test = y\_test.fillna(y\_test.mean())

# Initialize the ANN model

ann\_model = Sequential()

ann\_model.add(Dense(64, input\_dim=X\_train.shape[1], activation='relu'))

ann\_model.add(Dense(32, activation='relu'))

ann\_model.add(Dense(1))

ann\_model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the ANN model

ann\_model.fit(X\_train, y\_train, epochs=50, batch\_size=32, verbose=1)

# Make predictions using the ANN model

ann\_pred = ann\_model.predict(X\_test)

# Evaluate the ANN model

ann\_accuracy = mean\_squared\_error(y\_test, ann\_pred)

print(f"ANN Test Mean Squared Error: {ann\_accuracy}")

# Calculate percentage error for ANN

mean\_value = y\_test.mean()

ann\_percentage\_error = (ann\_accuracy / mean\_value) \* 100

print(f"ANN Test Mean Squared Error (Percentage): {ann\_percentage\_error}%")

# Standardize the numeric features for RNN

scaler\_rnn = StandardScaler()

X\_train[numeric\_columns] = scaler\_rnn.fit\_transform(X\_train[numeric\_columns])

X\_test[numeric\_columns] = scaler\_rnn.transform(X\_test[numeric\_columns])

# Fill NaN values in the target variable for training set for RNN

y\_train = y\_train.fillna(y\_train.mean())

# Fill NaN values in the target variable for testing set for RNN

y\_test = y\_test.fillna(y\_test.mean())

# Reshape data for RNN

X\_train\_rnn = X\_train.values.reshape((X\_train.shape[0], 1, X\_train.shape[1]))

X\_test\_rnn = X\_test.values.reshape((X\_test.shape[0], 1, X\_test.shape[1]))

# Initialize the RNN model

rnn\_model = Sequential()

rnn\_model.add(SimpleRNN(64, input\_shape=(X\_train\_rnn.shape[1], X\_train\_rnn.shape[2]), activation='relu'))

rnn\_model.add(Dense(32, activation='relu'))

rnn\_model.add(Dense(1))

rnn\_model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the RNN model

rnn\_model.fit(X\_train\_rnn, y\_train, epochs=50, batch\_size=32, verbose=1)

# Make predictions using the RNN model

rnn\_pred = rnn\_model.predict(X\_test\_rnn)

# Evaluate the RNN model

rnn\_accuracy = mean\_squared\_error(y\_test, rnn\_pred)

print(f"RNN Test Mean Squared Error: {rnn\_accuracy}")

# Calculate percentage error for RNN

rnn\_percentage\_error = (rnn\_accuracy / mean\_value) \* 100

print(f"RNN Test Mean Squared Error (Percentage): {rnn\_percentage\_error}%")

# Import necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from google.colab import files

# Upload your CSV file

uploaded = files.upload()

# Load the data

data = pd.read\_csv(next(iter(uploaded)))

# Replace 'Evaluated Annual Electric Savings (kWh)' with the actual target variable name

target\_variable = 'Evaluated Annual Electric Savings (kWh)'

# Exclude non-numeric columns from standardization

non\_numeric\_columns = ['Project ID', 'Project County', 'Project City', 'Project Completion Date', 'Customer Type', 'Location 1']

X = data.drop([target\_variable] + non\_numeric\_columns, axis=1)

y = data[target\_variable]

# Convert categorical variables to numeric if any

X = pd.get\_dummies(X)

# Fill missing values with the mean or another appropriate strategy

X = X.fillna(X.mean())

# Convert data types

X = X.astype('float32')

y = y.astype('float32')

# 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)

# Standardize the numeric features for ANN

numeric\_columns = X.select\_dtypes(include=['float32']).columns

scaler\_ann = StandardScaler()

X\_train[numeric\_columns] = scaler\_ann.fit\_transform(X\_train[numeric\_columns])

X\_test[numeric\_columns] = scaler\_ann.transform(X\_test[numeric\_columns])

# Fill NaN values in the target variable for training set for ANN

y\_train = y\_train.fillna(y\_train.mean())

# Fill NaN values in the target variable for testing set for ANN

y\_test = y\_test.fillna(y\_test.mean())

# Initialize the ANN model

ann\_model = Sequential()

ann\_model.add(Dense(64, input\_dim=X\_train.shape[1], activation='relu'))

ann\_model.add(Dense(32, activation='relu'))

ann\_model.add(Dense(1))

ann\_model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the ANN model

ann\_model.fit(X\_train, y\_train, epochs=50, batch\_size=32, verbose=1)

# Make predictions using the ANN model

ann\_pred = ann\_model.predict(X\_test)

# Evaluate the ANN model

ann\_accuracy = mean\_squared\_error(y\_test, ann\_pred)

print(f"ANN Test Mean Squared Error: {ann\_accuracy}")

# Calculate percentage error for ANN

mean\_value = y\_test.mean()

ann\_percentage\_error = (ann\_accuracy / mean\_value) \* 100

print(f"ANN Test Mean Squared Error (Percentage): {ann\_percentage\_error}%")

# Standardize the numeric features for Random Forest

scaler\_rf = StandardScaler()

X\_train[numeric\_columns] = scaler\_rf.fit\_transform(X\_train[numeric\_columns])

X\_test[numeric\_columns] = scaler\_rf.transform(X\_test[numeric\_columns])

# Fill NaN values in the target variable for training set for Random Forest

y\_train\_rf = y\_train.fillna(y\_train.mean())

# Fill NaN values in the target variable for testing set for Random Forest

y\_test\_rf = y\_test.fillna(y\_test.mean())

# Initialize the Random Forest model

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

# Train the Random Forest model

rf\_model.fit(X\_train, y\_train\_rf)

# Make predictions using the Random Forest model

rf\_pred = rf\_model.predict(X\_test)

# Evaluate the Random Forest model

rf\_accuracy = mean\_squared\_error(y\_test\_rf, rf\_pred)

print(f"Random Forest Test Mean Squared Error: {rf\_accuracy}")

# Calculate percentage error for Random Forest

rf\_percentage\_error = (rf\_accuracy / mean\_value) \* 100

print(f"Random Forest Test Mean Squared Error (Percentage): {rf\_percentage\_error}%")

# Import necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Conv1D, LSTM, Flatten

from google.colab import files

# Upload your CSV file

uploaded = files.upload()

# Load the data

data = pd.read\_csv(next(iter(uploaded)))

# Replace 'Evaluated Annual Electric Savings (kWh)' with the actual target variable name

target\_variable = 'Evaluated Annual Electric Savings (kWh)'

# Exclude non-numeric columns from standardization

non\_numeric\_columns = ['Project ID', 'Project County', 'Project City', 'Project Completion Date', 'Customer Type', 'Location 1']

X = data.drop([target\_variable] + non\_numeric\_columns, axis=1)

y = data[target\_variable]

# Convert categorical variables to numeric if any

X = pd.get\_dummies(X)

# Fill missing values with the mean or another appropriate strategy

X = X.fillna(X.mean())

# Convert data types

X = X.astype('float32')

y = y.astype('float32')

# 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)

# Standardize the numeric features for ANN

numeric\_columns = X.select\_dtypes(include=['float32']).columns

scaler\_ann = StandardScaler()

X\_train[numeric\_columns] = scaler\_ann.fit\_transform(X\_train[numeric\_columns])

X\_test[numeric\_columns] = scaler\_ann.transform(X\_test[numeric\_columns])

# Fill NaN values in the target variable for training set for ANN

y\_train = y\_train.fillna(y\_train.mean())

# Fill NaN values in the target variable for testing set for ANN

y\_test = y\_test.fillna(y\_test.mean())

# Initialize the ANN model

ann\_model = Sequential()

ann\_model.add(Dense(64, input\_dim=X\_train.shape[1], activation='relu'))

ann\_model.add(Dense(32, activation='relu'))

ann\_model.add(Dense(1))

ann\_model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the ANN model

ann\_model.fit(X\_train, y\_train, epochs=50, batch\_size=32, verbose=1)

# Make predictions using the ANN model

ann\_pred = ann\_model.predict(X\_test)

# Evaluate the ANN model

ann\_accuracy = mean\_squared\_error(y\_test, ann\_pred)

print(f"ANN Test Mean Squared Error: {ann\_accuracy}")

# Calculate percentage error for ANN

mean\_value = y\_test.mean()

ann\_percentage\_error = (ann\_accuracy / mean\_value) \* 100

print(f"ANN Test Mean Squared Error (Percentage): {ann\_percentage\_error}%")

# Reshape data for ConvLSTM

X\_train\_cnn\_lstm = X\_train.values.reshape((X\_train.shape[0], 1, X\_train.shape[1]))

X\_test\_cnn\_lstm = X\_test.values.reshape((X\_test.shape[0], 1, X\_test.shape[1]))

# Initialize the ConvLSTM model

cnn\_lstm\_model = Sequential()

cnn\_lstm\_model.add(Conv1D(filters=64, kernel\_size=1, activation='relu', input\_shape=(1, X\_train.shape[1])))

cnn\_lstm\_model.add(LSTM(64, activation='relu'))

cnn\_lstm\_model.add(Dense(32, activation='relu'))

cnn\_lstm\_model.add(Dense(1))

cnn\_lstm\_model.compile(optimizer='adam', loss='mean\_squared\_error')

# Train the ConvLSTM model

cnn\_lstm\_model.fit(X\_train\_cnn\_lstm, y\_train, epochs=50, batch\_size=32, verbose=1)

# Make predictions using the ConvLSTM model

cnn\_lstm\_pred = cnn\_lstm\_model.predict(X\_test\_cnn\_lstm)

# Evaluate the ConvLSTM model

cnn\_lstm\_accuracy = mean\_squared\_error(y\_test, cnn\_lstm\_pred)

print(f"ConvLSTM Test Mean Squared Error: {cnn\_lstm\_accuracy}")

# Calculate percentage error for ConvLSTM

cnn\_lstm\_percentage\_error = (cnn\_lstm\_accuracy / mean\_value) \* 100

print(f"ConvLSTM Test Mean Squared Error (Percentage): {cnn\_lstm\_percentage\_error}%")