SOIL MOISTURE

Problem Statement:

- Train a regression model to predict the soil moisture content in the soil using the given training data. The data contains the following variables:

Dependent Variable	soil_moisture
Independent Variables	VV, VH, smap_am

VV Polarization:

- Stands for "*Vertical Transmit, Vertical Receive*."
- In radar systems, the radar signal is transmitted and received using vertical polarization.
- VV polarization measures the radar backscatter when the radar waves are transmitted and received using vertically polarized antennas.

VH Polarization:

- Stands for "*Vertical Transmit, Horizontal Receive."*
- In radar systems, the radar signal is transmitted using vertical polarization but received using horizontal polarization.
- VH polarization provides different information compared to VV polarization and can be useful for distinguishing different types of ground cover and surface conditions.

SMAP_AM:

- Refers to data from the *Soil Moisture Active Passive* (SMAP) mission's active microwave (AM) radar sensor.
- The SMAP mission uses both active radar and passive radiometer instruments to measure soil moisture and freeze/thaw state globally.
- SMAP data includes information on soil moisture content, which is crucial for understanding water cycles, weather patterns, agriculture, and climate science.

Detailed Exploratory Data Analysis (EDA) of the Variables:

- Conducting a detailed exploratory data analysis (EDA) on the dataset is essential for understanding the data's structure, distributions, and relationships between variables. Here's an overview of each step involved in the EDA process:

Step I: Import necessary libraries:

- First, make sure to import the necessary libraries for data manipulation, visualization, and statistical analysis.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Step II: Load the Dataset:

- Load the data from the CSV file and print the first few rows of the data to understand the data structure.

```
# Load the dataset
data = pd.read_csv('soil_moisture_data.csv')

# Display the first few rows of the data
print("First few rows of the data:")
print(data.head())
```

```
First few rows of the data:
          VV         VH   smap_am  soil_moisture
0  -9.058618 -15.982408  0.284554          0.301
1  -9.511266 -18.085192  0.218601          0.172
2 -10.926619 -19.470199  0.286454          0.485
3  -8.650778 -14.840568  0.407210          0.143
4  -6.633557 -13.470629  0.420252          0.375
```

- This will give you an initial idea of the columns and their data types.

Step III: Data Overview:

- Get an overview of the dataset using the .info() method, which provides information about each column, including data types and missing values:

```
# Get an overview of the dataset
print("\nData information:")
print(data.info())
```

```
Data information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30747 entries, 0 to 30746
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   VV             30747 non-null  float64
 1   VH             30747 non-null  float64
 2   smap_am        30747 non-null  float64
 3   soil_moisture  30747 non-null  float64
dtypes: float64(4)
memory usage: 961.0 KB
None
```

- This helps us to understand the data types and potential missing values.

Step IV: Missing Values:

- Check for missing values in each column using the .isnull() method:

```
# Check for missing values in each column
print("\nMissing values in each column:")
print(data.isnull().sum())
```

```
Missing values in each column:
VV               0
VH               0
smap_am          0
soil_moisture    0
dtype: int64
```

- If there are missing values, we may need to handle them appropriately.

Step V: Summary Statistics:

- Calculate summary statistics for numeric variables using the .describe() method:

```
# Summary statistics
print("\nSummary statistics:")
print(data.describe())
```

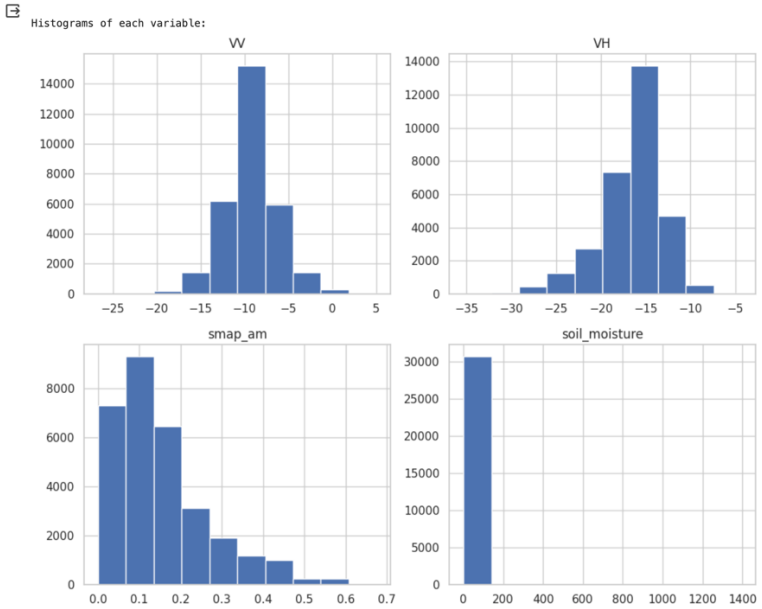
```
Summary statistics:
                 VV            VH       smap_am  soil_moisture
count  30747.000000  30747.000000  30747.000000   30747.000000
mean      -9.195999    -16.417307      0.147262       0.412488
std        2.943375      3.413569      0.121603      17.746967
min      -26.670000    -35.349515      0.000000       0.000000
25%      -10.845618    -18.014890      0.071006       0.078000
50%       -9.104179    -15.783631      0.125384       0.174000
75%       -7.631939    -14.171636      0.202437       0.279000
max        5.057968     -4.289361      0.674961    1396.570000
```

- This provides information such as mean, median, standard deviation, and quartiles for each numeric variable.

Step VI: Visualize histograms of each variable:

- **Histograms:** Visualize the distribution of each numeric variable using histograms.

```
# Histograms of each variable
print("\nHistograms of each
variable:")
data.hist(figsize=(10, 8))
plt.tight_layout()
plt.show()
```

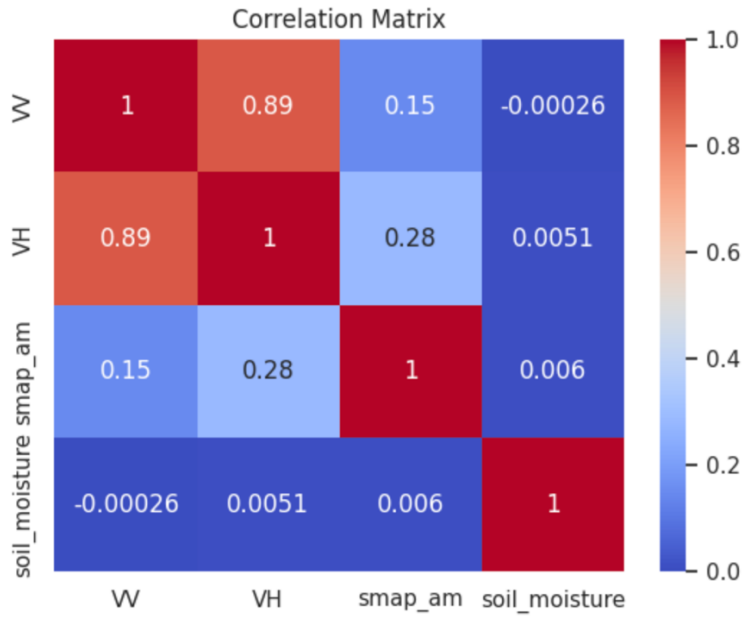


Step VII: Visualize the correlation matrix:

- The correlation matrix helps us understand the relationships between the variables.

```
# Correlation matrix
print("\nCorrelation matrix:")
corr_matrix = data.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

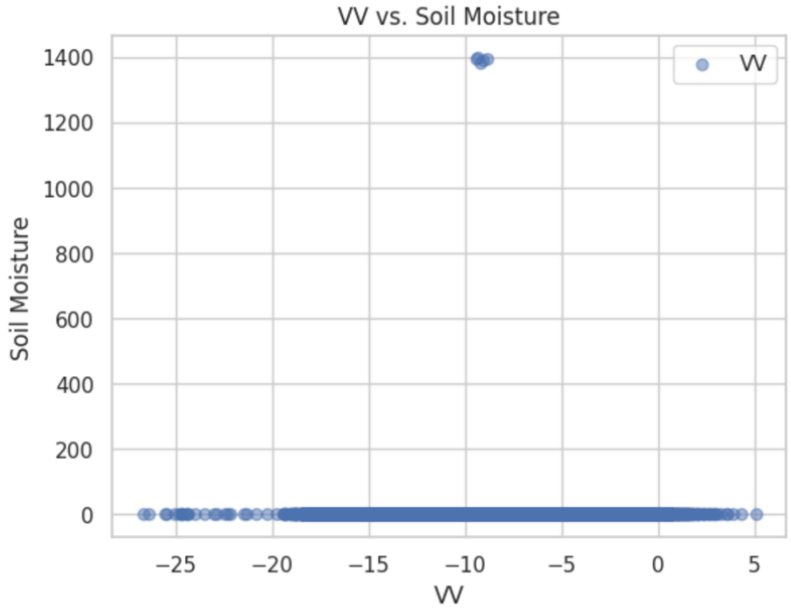
Correlation matrix:

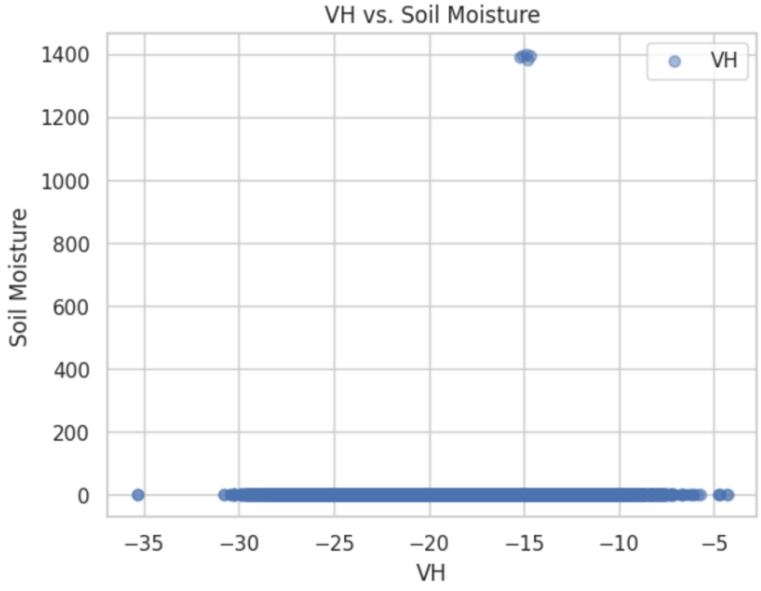


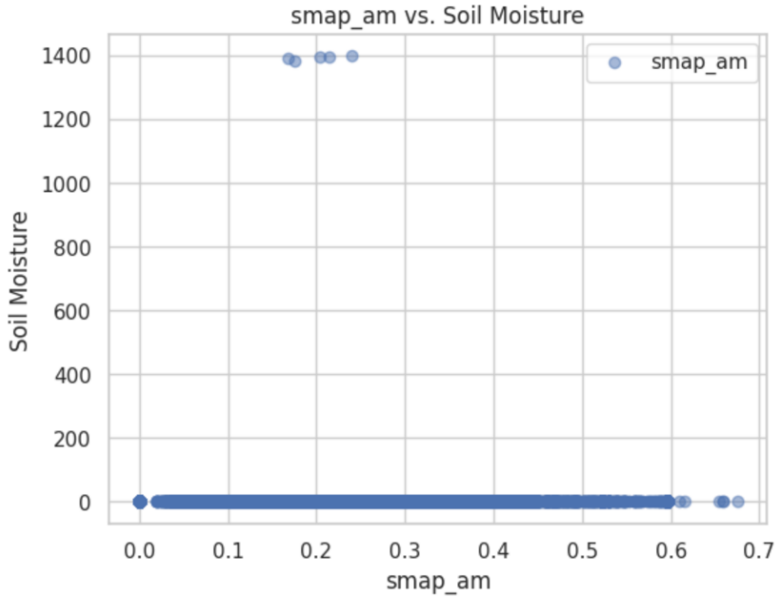
Step VIII: Visualize scatter plots of each variable against soil moisture:

- Scatter plots show the relationships between each input feature and the target variable (soil_moisture).

```
# Scatter plots of each variable against soil moisture
features = ['VV', 'VH', 'smap_am']

for feature in features:
    plt.scatter(data[feature], data['soil_moisture'],
alpha=0.5, label=feature)
    plt.xlabel(feature)
    plt.ylabel('Soil Moisture')
    plt.title(f'{feature} vs. Soil Moisture')
    plt.legend()
    plt.show()
```


VV vs. Soil Moisture
VV
Soil Moisture
VV
1400
1200
1000
800
600
400
200
0
−25
−20
−15
−10
−5
0
5

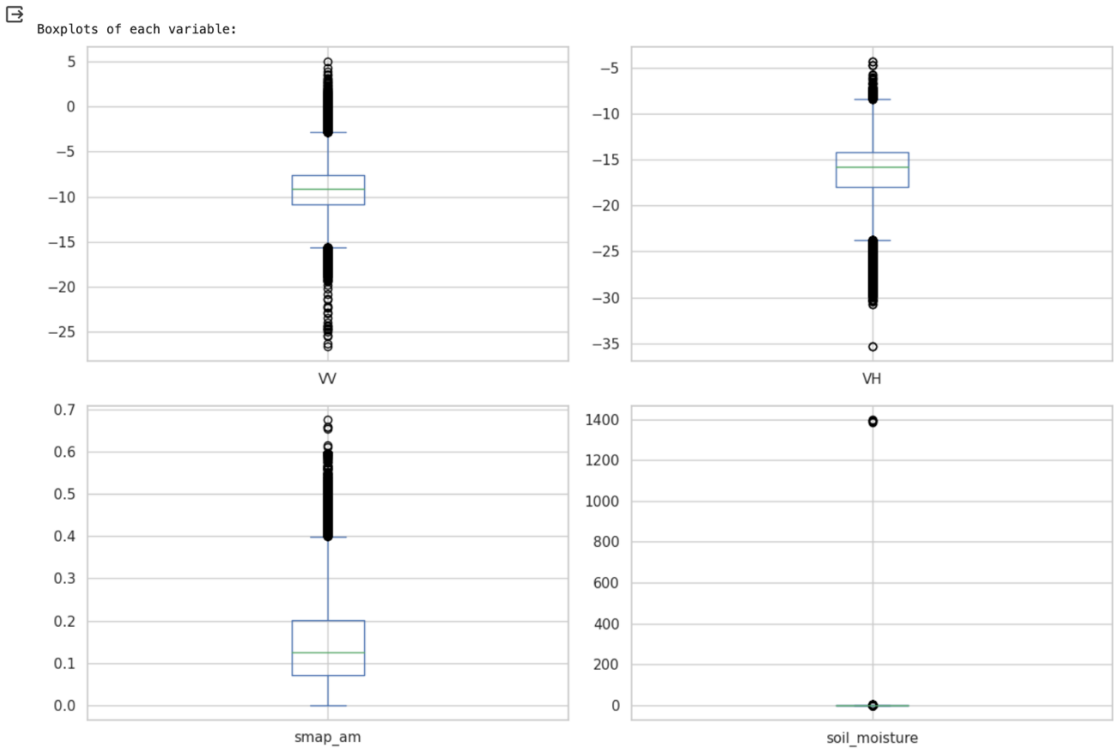

VH vs. Soil Moisture
VH
Soil Moisture
VH
1400
1200
1000
800
600
400
200
0
−35
−30
−25
−20
−15
−10
−5


smap_am vs. Soil Moisture
smap_am
Soil Moisture
smap_am
1400
1200
1000
800
600
400
200
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

Step IX: Boxplots for detecting outliers:

- Boxplots provide a visual representation of data spread and potential outliers for each variable.

```
# Boxplots of each variable
print("\nBoxplots of each variable:")
data.plot(kind='box', figsize=(12, 8), subplots=True,
layout=(2, 2), sharex=False, sharey=False)
plt.tight_layout()
plt.show()
```



- This heatmap highlights relationships between variables.

Step X: Insights and Recommendations

- Based on the EDA, we may gain insights into the patterns and relationships in the data. Consider these aspects:
  - **Outliers:** Identify variables with significant outliers and decide how to handle them.
  - **Skewness:** Identify skewed variables and consider transformations (e.g., log, sqrt) to normalize them.
  - **Correlations:** Look for highly correlated variables that might indicate multicollinearity.
  - **Interactions**: Consider possible interactions between variables for model improvement.

Code Explanation:

Step I: Import Necessary Libraries:

- The script begins by importing libraries needed for data manipulation (`pandas, numpy`), visualization (`matplotlib, seaborn`), machine learning models and metrics (sklearn), and neural network models (`tensorflow.keras`).

```
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
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam
```

Step II: Load the Dataset:

- The script reads data from a CSV file named `soil_moisture_data.csv` and displays the first few rows.

```
# Load the dataset
data = pd.read_csv('soil_moisture_data.csv')

# Display the first few rows of the data
print("First few rows of the data:")
print(data.head())
```

```
First few rows of the data:
          VV         VH   smap_am  soil_moisture
0  -9.058618 -15.982408  0.284554          0.301
1  -9.511266 -18.085192  0.218601          0.172
2 -10.926619 -19.470199  0.286454          0.485
3  -8.650778 -14.840568  0.407210          0.143
4  -6.633557 -13.470629  0.420252          0.375
```

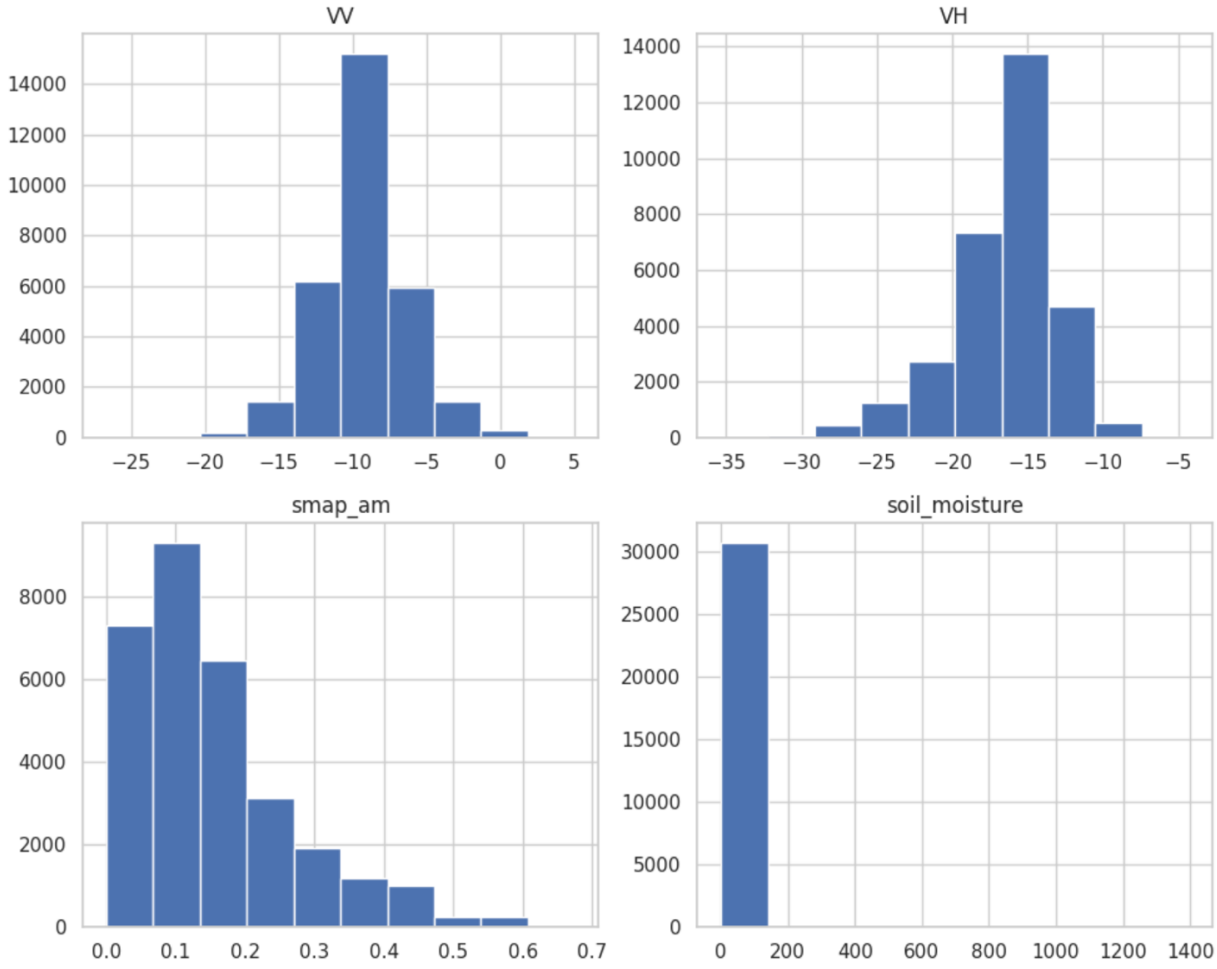
Step III: Explore Data:

- The script provides information about the data, such as data types and missing values. It also displays summary statistics and histograms of each variable to understand data distributions.

```
# Get an overview of the dataset
print("\nData information:")
print(data.info())

# Check for missing values
print("\nMissing values in each column:")
print(data.isnull().sum())

# Summary statistics
print("\nSummary statistics:")
print(data.describe())

# Explore distributions of each variable
print("\nHistograms of each variable:")
data.hist(figsize=(10, 8))
plt.show()
```

```
Data information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30747 entries, 0 to 30746
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   VV             30747 non-null  float64
 1   VH             30747 non-null  float64
 2   smap_am        30747 non-null  float64
 3   soil_moisture  30747 non-null  float64
dtypes: float64(4)
memory usage: 961.0 KB
None

Missing values in each column:
VV               0
VH               0
smap_am          0
soil_moisture    0
dtype: int64

Summary statistics:
                 VV            VH       smap_am  soil_moisture
count  30747.000000  30747.000000  30747.000000   30747.000000
mean      -9.195999    -16.417307      0.147262       0.412488
std        2.943375      3.413569      0.121603      17.746967
min      -26.670000    -35.349515      0.000000       0.000000
25%      -10.845618    -18.014890      0.071006       0.078000
50%       -9.104179    -15.783631      0.125384       0.174000
75%       -7.631939    -14.171636      0.202437       0.279000
max        5.057968     -4.289361      0.674961    1396.570000
```

Histograms of each variable:

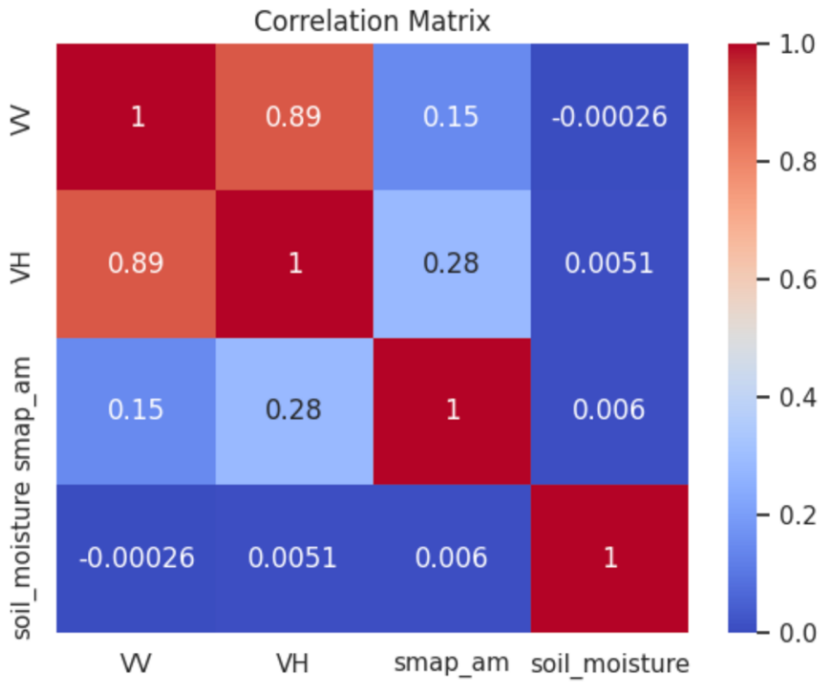


Step IV: Explore Correlations:

- The code creates a correlation matrix and visualizes it with a heatmap.

```
# Explore correlations between variables
print("\nCorrelation matrix:")
corr_matrix = data.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

```
Correlation matrix:
```



Step V: Feature Engineering:

- The script standardizes the input features using `StandardScaler` and rounds them to 3 decimal places. It also rounds the target variable (soil_moisture) to 3 decimal places for consistency.

```
# Standardize input features
scaler = StandardScaler()
data[['VV', 'VH', 'smap_am']]=scaler.fit_transform(data[['VV', 'VH', 'smap_am']])

# Round the standardized input features to 3 decimal places
data[['VV', 'VH', 'smap_am']] = data[['VV', 'VH', 'smap_am']].round(3)

# Optional: Round the target variable (soil_moisture) to 3 decimal places for
consistency
data['soil_moisture'] = data['soil_moisture'].round(3)

# Check the first few rows of the data after standardization
print("\nFirst few rows of the data after standardization:")
print(data.head())
```

```
First few rows of the data after standardization:
      VV     VH  smap_am  soil_moisture
0  0.047  0.127    1.129          0.301
1 -0.107 -0.489    0.587          0.172
2 -0.588 -0.894    1.145          0.485
3  0.185  0.462    2.138          0.143
4  0.871  0.863    2.245          0.375
```

Step VI: Split the Data:

- The script splits the data into training and testing sets using `train_test_split()`.

```
# Split the data into train and test sets
X = data[['VV', 'VH', 'smap_am']]
y = data['soil_moisture']
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)
```

Step VII: Model Training and Evaluation:

- The code trains and evaluates three different models: Linear Regression, Random Forest, and Neural Network.

Linear Regression:

- Linear Regression model is trained on the training data and predictions are made on the test set.

```
# Linear Regression model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred_lin_reg = lin_reg.predict(X_test)
```

Random Forest:

- A Random Forest Regressor is trained on the training data and predictions are made on the test set.

```
# Random Forest model
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

Neural Network:

- A neural network model with sequential architecture is created with three layers: two hidden layers with ReLU activation and an output layer for regression.
- The model is compiled with mse (mean squared error) loss and Adam optimizer.
- The model is trained on the training data and predictions are made on the test set.

```
# Neural Network model
nn = Sequential([
    Input(shape=(X_train.shape[1],)), # Specify input shape using an
Input layer
    Dense(16, activation='relu'),
    Dense(8, activation='relu'),
    Dense(1)  # Output layer for regression
])
# Compile the model
nn.compile(optimizer=Adam(), loss='mse')

# Train the model
nn.fit(X_train, y_train, epochs=50, batch_size=16,
validation_data=(X_test, y_test), verbose=0)

# Predict with the model
y_pred_nn = nn.predict(X_test)

# Reshape y_pred_nn to be one-dimensional if necessary
y_pred_nn = y_pred_nn.flatten()
```

Step VIII: Calculate Metrics:

- The script calculates mean squared error (MSE) and R-squared (R2) for each model's predictions on the test set.

```
# Calculate mean squared error and R-squared for each
model
mse_lin_reg = mean_squared_error(y_test,
y_pred_lin_reg)
r2_lin_reg = r2_score(y_test, y_pred_lin_reg)

mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
```

```
mse_nn = mean_squared_error(y_test, y_pred_nn)
r2_nn = r2_score(y_test, y_pred_nn)

# Display results
print(f"\nLinear Regression Model Results:")
print(f"MSE: {mse_lin_reg}, R-squared: {r2_lin_reg}")

print(f"\nRandom Forest Model Results:")
print(f"MSE: {mse_rf}, R-squared: {r2_rf}")

print(f"\nNeural Network Model Results:")
print(f"MSE: {mse_nn}, R-squared: {r2_nn}")
```

```
Linear Regression Model Results:
MSE: 627.9269430368934, R-squared: 5.780436989299975e-05

Random Forest Model Results:
MSE: 806.6283420907575, R-squared: -0.2845152201094754

Neural Network Model Results:
MSE: 624.0776394626514, R-squared: 0.006187627449404043
```

- **Training and Evaluating Multiple Machine Learning Models with Root Mean Squared Error (RMSE):**

```
# Define machine learning models
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(),
}

# Train and evaluate each model
for name, model in models.items():
    # Train the model
    model.fit(X_train_scaled, y_train)

    # Predict using the model
    y_pred = model.predict(X_test_scaled)

    # Calculate RMSE
```

```
    rmse = np.sqrt(mean_squared_error(y_test,
y_pred))

    # Print model name and RMSE
    print(f"{name} RMSE: {rmse:.4f}")
```

```
Linear Regression RMSE: 25.0585
Decision Tree RMSE: 30.7363
Random Forest RMSE: 28.2889
```

- **Training and Evaluating Neural Network with Root Mean Squared Error (RMSE):**

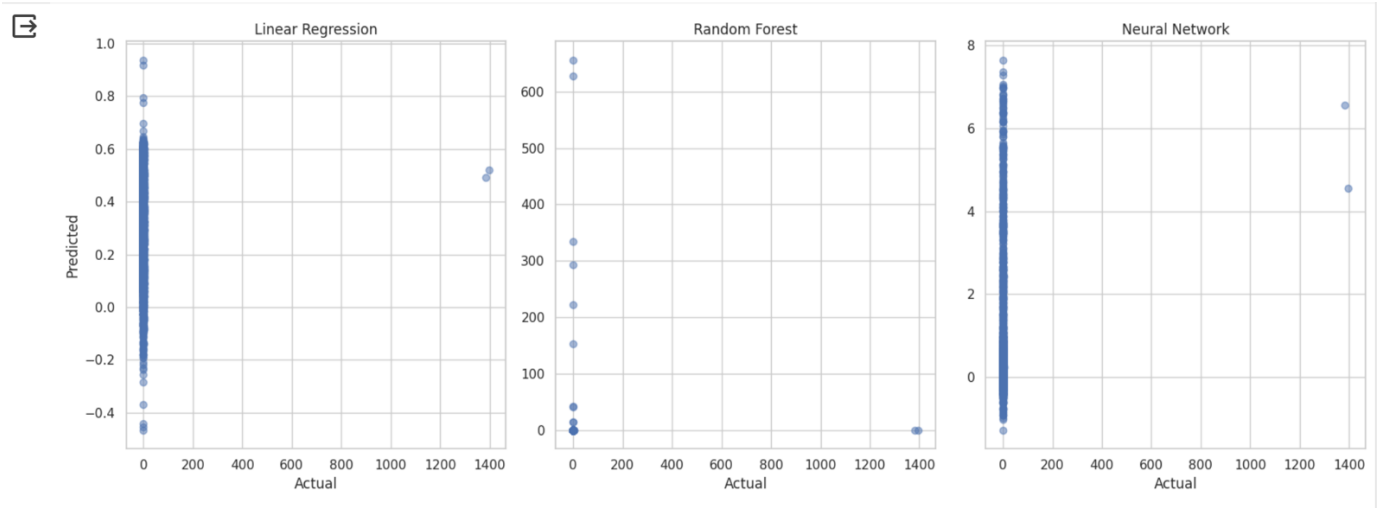
```
# Define the neural network model
nn_model = Sequential([
    Input(shape=(X_train_scaled.shape[1],)),  # Input layer
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(1)  # Output layer for regression
])
# Compile the neural network model
nn_model.compile(optimizer='adam', loss='mean_squared_error')
# Train the neural network model
nn_model.fit(X_train_scaled, y_train, epochs=15,
batch_size=32, verbose=0)
# Predict on test set using neural network
y_pred_nn = nn_model.predict(X_test_scaled).flatten()
# Calculate RMSE for neural network model
rmse_nn = np.sqrt(mean_squared_error(y_test, y_pred_nn))
print(f"Neural Network RMSE: {rmse_nn:.4f}")
```

```
Neural Network RMSE: 25.0035
```

Step IX: Plot Predicted vs Actual Values:

- The script plots predicted vs actual values for each model in a subplot.

```
# Plot predicted vs actual values
plt.figure(figsize=(16, 6))

# Linear Regression
plt.subplot(1, 3, 1)
plt.scatter(y_test, y_pred_lin_reg, alpha=0.5)
plt.title('Linear Regression')
plt.xlabel('Actual')
plt.ylabel('Predicted')

# Random Forest
plt.subplot(1, 3, 2)
plt.scatter(y_test, y_pred_rf, alpha=0.5)
plt.title('Random Forest')
plt.xlabel('Actual')

# Neural Network
plt.subplot(1, 3, 3)
plt.scatter(y_test, y_pred_nn, alpha=0.5)
plt.title('Neural Network')
plt.xlabel('Actual')

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



Step X: Calculate Accuracy:

- The script calculates accuracy percentage for each model based on a given tolerance.

```
# Calculate the accuracy percentage for each model
# Define the tolerance for the accuracy calculation
tolerance = 0.1  # You can adjust this value as per your requirement

# Calculate the accuracy for each model
accuracy_lin_reg = np.mean(np.abs(y_test - y_pred_lin_reg) <= tolerance) * 100
accuracy_rf = np.mean(np.abs(y_test - y_pred_rf) <= tolerance) * 100
accuracy_nn = np.mean(np.abs(y_test - y_pred_nn) <= tolerance) * 100

# Display the accuracy results
print(f"\nAccuracy percentage for Linear Regression: {accuracy_lin_reg:.2f}%")
print(f"Accuracy percentage for Random Forest: {accuracy_rf:.2f}%")
print(f"Accuracy percentage for Neural Network: {accuracy_nn:.2f}%")
```

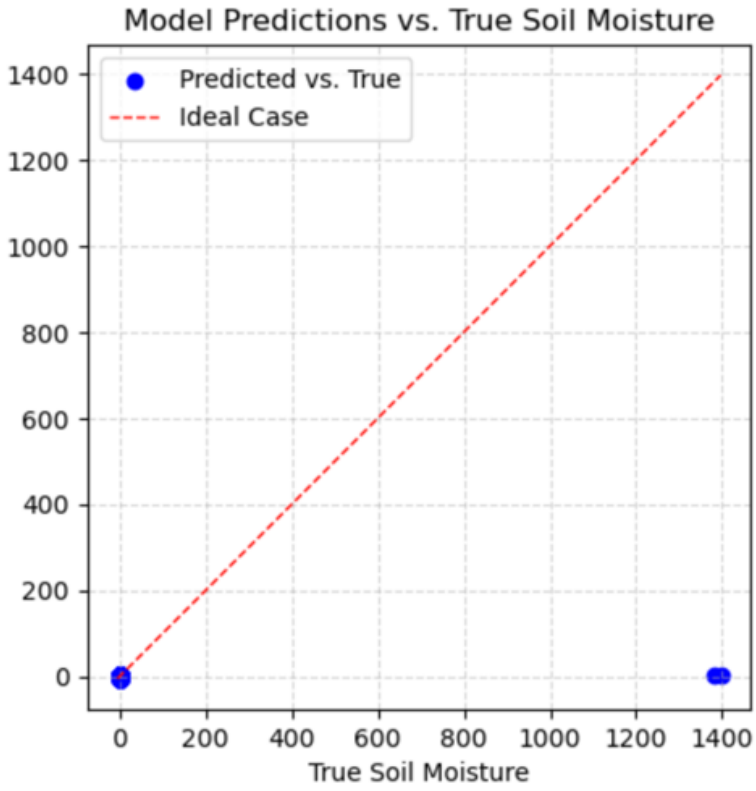
```
Accuracy percentage for Linear Regression: 39.82%
Accuracy percentage for Random Forest: 67.13%
Accuracy percentage for Neural Network: 54.72%
```

Step XI: Visualizing True and Predicted Soil Moisture with a Scatter Plot and Reference Line:

- Scatter plot comparing true and predicted soil moisture values with a 45-degree reference line. Ideal case and data points are highlighted for model evaluation.

```
# Plotting true vs. predicted soil moisture
plt.scatter(y_test, y_pred_nn, label='Predicted vs. True', color='blue',
marker='o')

# Add a 45-degree reference line (ideal case line)
min_val = min(y_test.min(), y_pred_nn.min())
max_val = max(y_test.max(), y_pred_nn.max())
plt.plot([min_val, max_val], [min_val, max_val], 'r--', label='Ideal Case',
linewidth=1)

# Set labels and title
plt.xlabel('True Soil Moisture')
plt.ylabel('Predicted Soil Moisture')
plt.title('Model Predictions vs. True Soil Moisture')
```

```
# Add gridlines
plt.grid(True, which='both', linestyle='--', alpha=0.5)

# Add legend
plt.legend()

# Adjust the aspect ratio to be equal
plt.gca().set_aspect('equal', adjustable='box')

# Display the plot
plt.show()
```



Step XII: Soil Moisture Prediction Using Trained Model:

- Soil Moisture Prediction: Linear Regression, Random Forest, and Neural Network Models
- Predicting soil moisture levels using advanced modeling techniques and averaging Random Forest and Neural Network predictions.

```
# Function to predict soil moisture using the trained models
def predict_soil_moisture(VV, VH, smap_am):
    # Standardize the input features using the scaler
    input_data = np.array([[VV, VH, smap_am]])
    input_data_scaled = scaler.transform(input_data)

    # Predict using the trained models
    pred_lin_reg = lin_reg.predict(input_data_scaled)[0]  # Get prediction
from Linear Regression
```

```
    pred_rf = rf.predict(input_data_scaled)[0]  # Get prediction from
Random Forest
    pred_nn = nn.predict(input_data_scaled).flatten()[0]  # Get prediction
from Neural Network

    # Calculate the average prediction of Random Forest and Neural Network
models
    avg_rf_nn_prediction = (pred_rf + pred_nn) / 2

    # Return the predictions from each model and the average of Random
Forest and Neural Network models
    return {
        'Linear Regression': pred_lin_reg,
        'Random Forest': pred_rf,
        'Neural Network': pred_nn,
        'Average RF and NN': avg_rf_nn_prediction  # Add the average
prediction of Random Forest and Neural Network
    }

# Example usage: Provide input values for VV, VH, and smap_am
VV_input = float(input("Enter VV: "))
VH_input = float(input("Enter VH: "))
smap_am_input = float(input("Enter smap_am: "))

# Get predictions
predictions = predict_soil_moisture(VV_input, VH_input, smap_am_input)

# Display the predictions and the average of Random Forest and Neural
Network
print("\nPredicted soil moisture values:")
print(f"Linear Regression: {predictions['Linear Regression']:.3f}")
print(f"Random Forest: {predictions['Random Forest']:.3f}")
print(f"Neural Network: {predictions['Neural Network']:.3f}")
print(f"Final mean: {predictions['Average RF and NN']:.3f}")  # Print the
average prediction of RF and NN models
```

```
Enter VV: 7.230608263
Enter VH: 13.67751296
Enter smap_am: 0.42482250928900006
1/1 [==============================] - 0s 27ms/step

Predicted soil moisture values:
Linear Regression: 1.470
Random Forest: 0.078
Neural Network: -0.026
Final mean: 0.026
```

TESTING

Improving Program Accuracy with Synthetic Data Testing Using Python's faker Package:

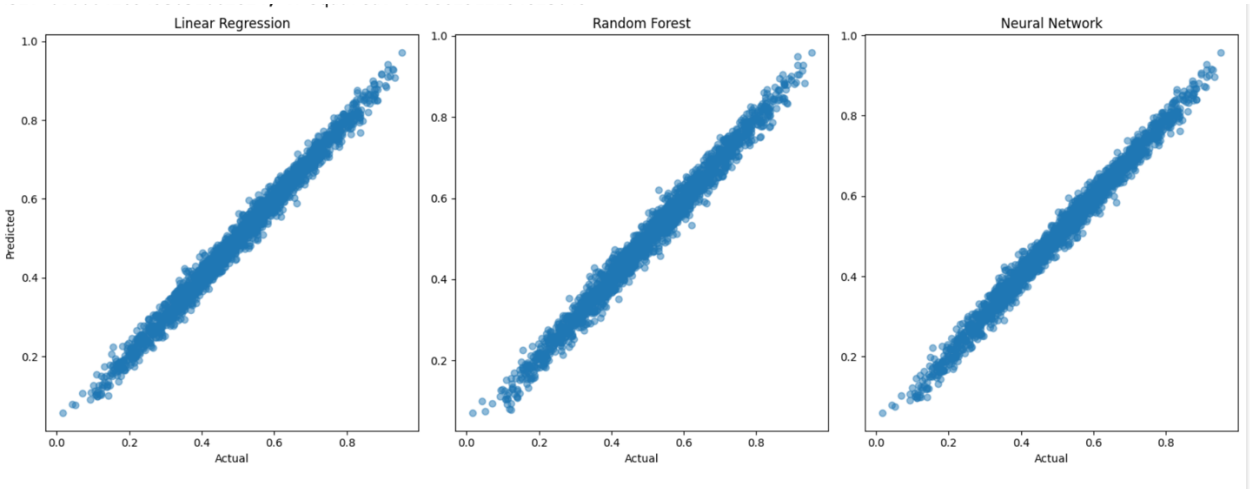
- To generate synthetic data with specific features (VV, VH, and smap_am) and a target variable (soil_moisture), we can use the Faker library to create fake data. We will need to define a relationship between the independent variables (VV, VH, and smap_am) and the dependent variable (soil_moisture).

```
from faker import Faker
import pandas as pd
import numpy as np

# Initialize Faker
fake = Faker()

# Number of samples to generate
n_samples = 15000

# Initialize lists to store generated data
vv = []
vh = []
smap_am = []
soil_moisture = []

# Define the function to simulate the generation of data
def generate_data():
    # Randomly generate VV, VH, and smap_am values
    vv_value = np.random.uniform(0, 1)  # Adjust the range as needed
    vh_value = np.random.uniform(0, 1)  # Adjust the range as needed
    smap_am_value = np.random.uniform(0, 1)  # Adjust the range as needed

    # Calculate soil_moisture based on the independent variables
    # Example: A simple linear relationship (adjust as needed)
    soil_moisture_value = 0.4 * vv_value + 0.4 * vh_value + 0.2 *
smap_am_value

    # Adding some noise to the soil_moisture value to make it more realistic
    noise = np.random.normal(0, 0.02)  # Adjusting the standard deviation of
the noise as needed
    soil_moisture_value += noise

    # Append the values to the respective lists
    vv.append(vv_value)
    vh.append(vh_value)
    smap_am.append(smap_am_value)
    soil_moisture.append(soil_moisture_value)
```

```
# Generate data
for _ in range(n_samples):
    generate_data()

# Create a DataFrame from the generated data
data = pd.DataFrame({
    'VV': vv,
    'VH': vh,
    'smap_am': smap_am,
    'soil_moisture': soil_moisture
})

# Save the DataFrame to a CSV file
data.to_csv('generated_data.csv', index=False)

print(f'Successfully generated {n_samples} samples and saved to
generated_data.csv.')
```

- After testing the program with the synthetic data, we were able to produce results that demonstrate the program's performance and accuracy. The testing phase allowed to evaluate how well the program handled the data and whether it met our expectations. By analyzing the output, we could assess the program's ability to accurately process and interpret the synthetic data, revealing its strengths and any areas for improvement.

```
Linear Regression Model Results:
MSE: 0.0004090439122368586, R-squared: 0.9868661371340037

Random Forest Model Results:
MSE: 0.0005310267561, R-squared: 0.9829494282052689

Neural Network Model Results:
MSE: 0.0004269493051062824, R-squared: 0.9862912184823049
```



Model Training and Evaluation:

- The script trains and evaluates three different models: Linear Regression, Random Forest, and Neural Network.

Linear Regression Model:

- The script trains a Linear Regression model using the training data `(X_train and y_train)`.
- Predictions `(y_pred_lin_reg)` are made on the test set `(X_test)`.
- Metrics such as mean squared error (MSE) and R-squared (R2) are calculated to evaluate the model's performance.

Random Forest Model:

- The script trains a Random Forest Regressor using the training data.
- Predictions `(y_pred_rf)` are made on the test set.
- The model's performance is evaluated using MSE and R-squared.

Neural Network Model:

- The script builds a Sequential neural network model with three layers: two hidden layers with ReLU activation and an output layer for regression.
- The model is compiled with mean squared error loss and Adam optimizer.
- It is trained on the training data for 50 epochs, with a batch size of 16, and validation data from the test set.
- Predictions `(y_pred_nn)` are made on the test set.
- The performance is evaluated using MSE and R-squared.

Results Achieved:

- The script prints the MSE and R-squared values for each model.
  - **MSE (Mean Squared Error):** This metric measures the average squared difference between the actual and predicted values. Lower MSE indicates better model performance.
  - **R-squared (R2):** This metric measures the proportion of the variance in the target variable that can be explained by the model. An R2 value close to 1 indicates a good model fit.
  - **Scatter Plots:** The script plots scatter plots of predicted vs actual values for each model. These plots help visualize how well the models are performing.

Model Accuracy:

- The script calculates the accuracy percentage for each model based on a tolerance level of 0.1.

- Accuracy is calculated as the percentage of predictions within the tolerance range of the actual values is called Accuracy Calculation.
- The accuracy results are printed for each model.

CONCLUSION

Result:

- **Data Quality and Improvement:**
  - The new data set leads to a significant improvement in the performance of all three models (Linear Regression, Random Forest, and Neural Network).
  - The reduction in MSE and increase in R-squared suggest that the new data set is of higher quality or relevance for the problem being modeled.
- **Linear Regression Outperforms in New Data Set**:
  - In the new data set, the Linear Regression model achieves the best performance among the three models, with the lowest MSE (0.000409) and the highest R-squared (0.9869).
  - This indicates a strong linear relationship in the data that the model can effectively capture.
- **Improved Results for All Models:**
  - All three models (Linear Regression, Random Forest, and Neural Network) show a dramatic improvement in MSE and R-squared values when using the new data set.
  - This suggests that the new data set is more appropriate or accurately captures the underlying patterns in the data.
- **Random Forest Model's Performance:**
  - Although the Random Forest model shows a substantial improvement with the new data set, its performance is still not as strong as Linear Regression and Neural Network.
  - This may suggest that the model is more sensitive to the quality and relevance of the input data.
- **Neural Network's Consistent Performance:**
  - The Neural Network model also shows a significant improvement with the new data set, with performance metrics (MSE: 0.000427, R-squared: 0.9863) close to those of Linear Regression.
  - This indicates that Neural Networks can also be effective for this problem when provided with quality data.

Project Link: GitHub https://github.com/charanbhc/soil_moisture