

## ✗ Soil Moisture Prediction Using Machine Learning

### Case Study Submission – Vassar Labs IT Solutions

#### ✗ Objective

The primary objective of this project is to develop an end-to-end machine learning model to accurately predict soil moisture content using Sentinel-1 SAR backscatter coefficients (VV, VH) and SMAP satellite data. The goal is to build a robust regression pipeline that analyzes remote sensing data and estimates soil moisture efficiently and reliably.

#### This project also aims to:

Compare multiple machine learning algorithms.

Evaluate model performance using appropriate regression metrics.

Identify the most influential features affecting soil moisture.

Build a deployment-ready prediction pipeline.

### Problem Statement

Soil moisture is a critical parameter in agriculture, hydrology, and climate monitoring. However, direct measurement of soil moisture using ground sensors is expensive, time-consuming, and geographically limited.

Satellite-based remote sensing provides large-scale data, but translating radar backscatter signals into accurate soil moisture estimates requires advanced modeling techniques.

#### The challenge is:

How can we use machine learning techniques to accurately predict soil moisture from satellite radar backscatter (VV, VH) and SMAP data?

This project addresses that challenge by designing a complete machine learning workflow from data preprocessing to model deployment.

## ✗ GPU Availability and Device Configuration Check Using TensorFlow

```
import tensorflow as tf

# List GPUs detected
gpus = tf.config.list_physical_devices('GPU')
print("Available GPUs:", gpus)

# Check memory details
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())

Available GPUs: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 9247384943209267072
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 14426112000
locality {
```

```
bus_id: 1
links {
}
incarnation: 12141348024216446024
physical_device_desc: "device: 0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5"
xla_global_id: 416903419
]
```

## ▼ Importing Essential Python Libraries for Data Analysis and Visualization

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

## ▼ Loading the Dataset into a Pandas DataFrame

```
df = pd.read_csv("/content/t_s1_am_6am.csv")
```

## ▼ Initial Data Exploration and Structural Overview of the Dataset

```
df.head()
```

	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

Next steps: [Generate code with df](#) [New interactive sheet](#)

```
df.tail()
```

	VV	VH	smap_am	soil_moisture
30742	-8.499560	-14.775879	0.210248	0.123
30743	-6.419627	-12.533582	0.335442	0.066
30744	-7.664967	-14.617288	0.225540	0.131
30745	-10.701671	-17.080531	0.089007	0.177
30746	-14.913366	-23.418471	0.074439	0.150

```
df.sample(5)
```

	VV	VH	smap_am	soil_moisture
19593	-1.054214	-9.685921	0.000000	0.002
9470	-11.000000	-18.340000	0.000000	0.189
16487	-12.742156	-19.976140	0.159281	0.326
9029	-10.166256	-16.679455	0.285149	0.132
3405	-11.711278	-21.227416	0.000000	0.415

```
df.shape
```

```
(30747, 4)
```

```
df.size
```

```
122988
```

```
df.dtypes
```

```

0
VV      float64
VH      float64
smap_am  float64
soil_moisture  float64

dtype: object

```

## Exploratory Data Analysis and Data Cleaning Process

```
df.info()
```

```

<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

```

```
df.describe()
```

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

```
df.isna().sum()
```

```

0
VV      0
VH      0
smap_am 0
soil_moisture 0

dtype: int64

```

```
df.duplicated().sum()
```

```
np.int64(13)
```

```
df.duplicated().any()
```

```
np.True_
```

```

# 1. See the duplicate rows
df[df.duplicated()]

# 2. Drop duplicates (keep first occurrence)
df = df.drop_duplicates()

# 3. Verify duplicates are gone
df.duplicated().sum() # Should return 0

np.int64(0)

```

```
df.duplicated().any()
```

```
np.False_
```

```
!pip install ydata-profiling
import ydata_profiling as yd
```

```
Requirement already satisfied: tqdm<5,>=4.66.3 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (4.67.3)
Requirement already satisfied: seaborn<0.14,>=0.10.1 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (0.
Collecting multimethod<2,>=1.4 (from ydata-profiling)
  Downloading multimethod-1.12-py3-none-any.whl.metadata (9.6 kB)
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (0
Requirement already satisfied: typeguard<5,>=4 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (4.4.4)
Collecting imagehash==4.3.2 (from ydata-profiling)
  Downloading ImageHash-4.3.2-py2.py3-none-any.whl.metadata (8.4 kB)
Requirement already satisfied: wordcloud>=1.9.4 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (1.9.6)
Collecting dacite<2,>=1.9 (from ydata-profiling)
  Downloading dacite-1.9.2-py3-none-any.whl.metadata (17 kB)
Requirement already satisfied: numba<0.63,>=0.60 in /usr/local/lib/python3.12/dist-packages (from ydata-profiling) (0.60.0
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.12/dist-packages (from imagehash==4.3.2->ydata-profil
Requirement already satisfied: pillow in /usr/local/lib/python3.12/dist-packages (from imagehash==4.3.2->ydata-profiling)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2<3.2,>=3.1.6->ydata-
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->y
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->ydata
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->yd
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.5->y
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib<=3.10,>=3.
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.12/dist-packages (from numba<0.63,>=0.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas!=1.4.0,<3.0,>1.5->yda
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas!=1.4.0,<3.0,>1.5->y
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.12/dist-packages (from phik<0.13,>=0.12.5->ydata-p
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic<3,>=2->yda
Requirement already satisfied: pydantic-core=>2.41.4 in /usr/local/lib/python3.12/dist-packages (from pydantic<3,>=2->ydat
Requirement already satisfied: typing-extensions>=4.14.1 in /usr/local/lib/python3.12/dist-packages (from pydantic<3,>=2->y
Requirement already satisfied: typing-inspection>=0.4.2 in /usr/local/lib/python3.12/dist-packages (from pydantic<3,>=2->y
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.32
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.32.0->ydata-pr
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.32.0->y
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.32.0->yd
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.12/dist-packages (from statsmodels<1,>=0.13.2->ydata
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.12/dist-packages (from visions<0.8.2,>=0.7.5->visio
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.12/dist-packages (from visions<0.8.2,>=0.7.5->visio
Collecting puremagic (from visions<0.8.2,>=0.7.5->visions[type_image_path]<0.8.2,>=0.7.5->ydata-profiling)
  Downloading puremagic-1.30-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib<
Downloading ydata_profiling-4.18.1-py2.py3-none-any.whl (400 kB)
  400.4/400.4 kB 13.5 MB/s eta 0:00:00
Downloading ImageHash-4.3.2-py2.py3-none-any.whl (296 kB)
  296.7/296.7 kB 25.4 MB/s eta 0:00:00
Downloading dacite-1.9.2-py3-none-any.whl (16 kB)
Downloading filetype-1.2.0-py2.py3-none-any.whl (19 kB)
Downloading minify_html-0.18.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.1 MB)
  3.1/3.1 MB 69.0 MB/s eta 0:00:00
Downloading multimethod-1.12-py3-none-any.whl (10 kB)
Downloading phik-0.12.5-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (679 kB)
  679.7/679.7 kB 41.5 MB/s eta 0:00:00
Downloading visions-0.8.1-py3-none-any.whl (105 kB)
  105.4/105.4 kB 7.2 MB/s eta 0:00:00
Downloading puremagic-1.30-py3-none-any.whl (43 kB)
  43.3/43.3 kB 3.9 MB/s eta 0:00:00
Installing collected packages: puremagic, minify-html, filetype, multimethod, dacite, imagehash, visions, phik, ydata-prof
Successfully installed dacite-1.9.2 filetype-1.2.0 imagehash-4.3.2 minify-html-0.18.1 multimethod-1.12 phik-0.12.5 puremag
```

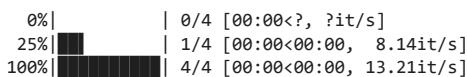
```
!python app.py
```

```
python3: can't open file '/content/app.py': [Errno 2] No such file or directory
```

```
profile = yd.ProfileReport(df)
profile.to_notebook_iframe()
```

Summarize dataset: 100%

29/29 [00:05&lt;00:00, 2.87it/s, Completed]



Generate report structure: 100%

1/1 [00:02&lt;00:00, 2.67s/it]

Render HTML: 100%

1/1 [00:00&lt;00:00, 1.68it/s]

## YData Profiling Report

# Overview

[Overview](#)[Alerts 5](#)[Reproduction](#)

## Dataset statistics

<b>Number of variables</b>	4
<b>Number of observations</b>	30734
<b>Missing cells</b>	0
<b>Missing cells (%)</b>	0.0%
<b>Duplicate rows</b>	0
<b>Duplicate rows (%)</b>	0.0%
<b>Total size in memory</b>	1.2 MiB
<b>Average record size in memory</b>	40.0 B

## Variable types

<b>Numeric</b>	4
----------------	---

# Variables

[Select Columns](#)

VV

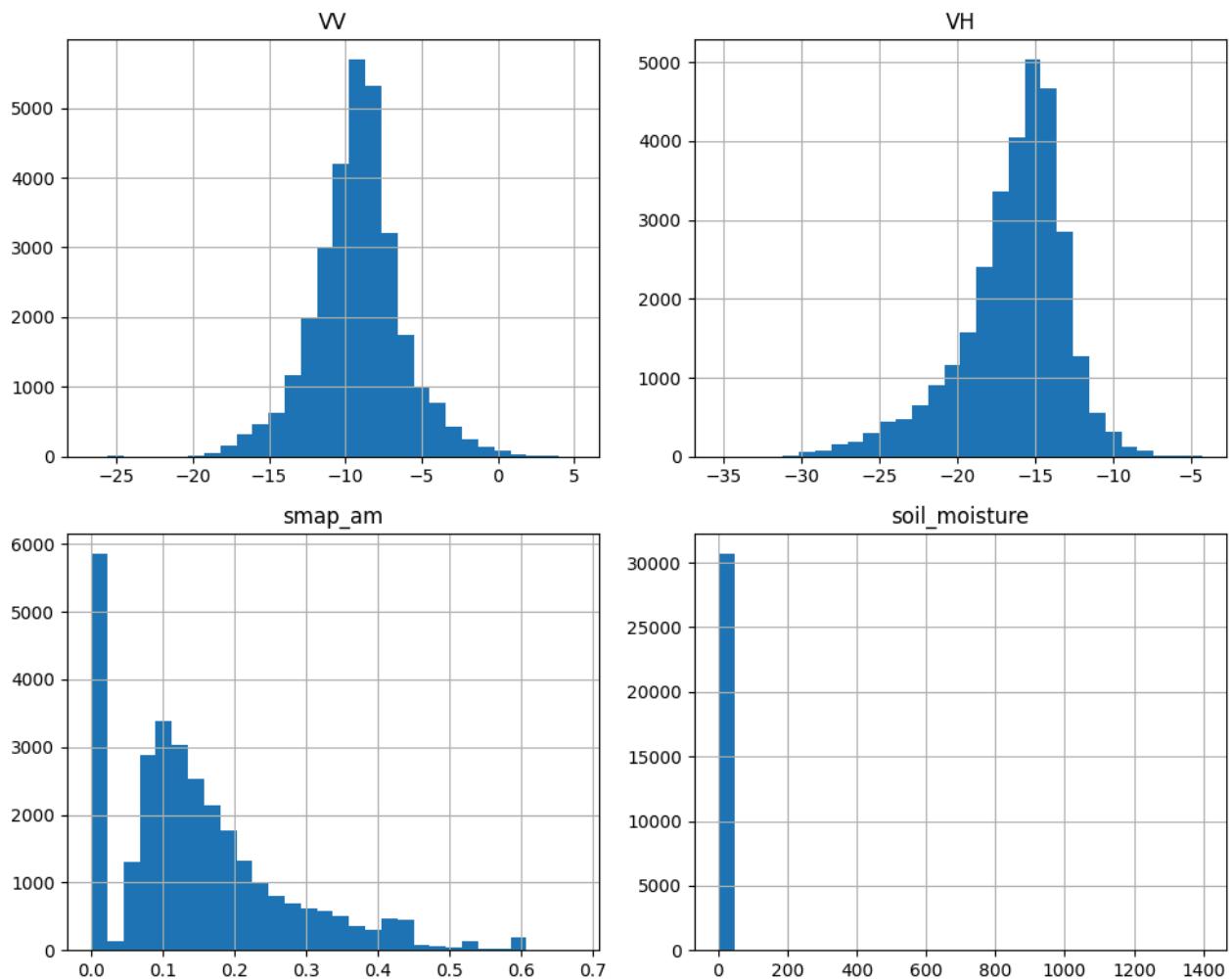
Start coding or [generate](#) with AI.

## Understanding the Data with Visualization

```
import matplotlib.pyplot as plt  
import seaborn as sns
```

## Plotting Histogram

```
df.hist(figsize=(10,8), bins=30)  
plt.tight_layout()  
plt.show()
```

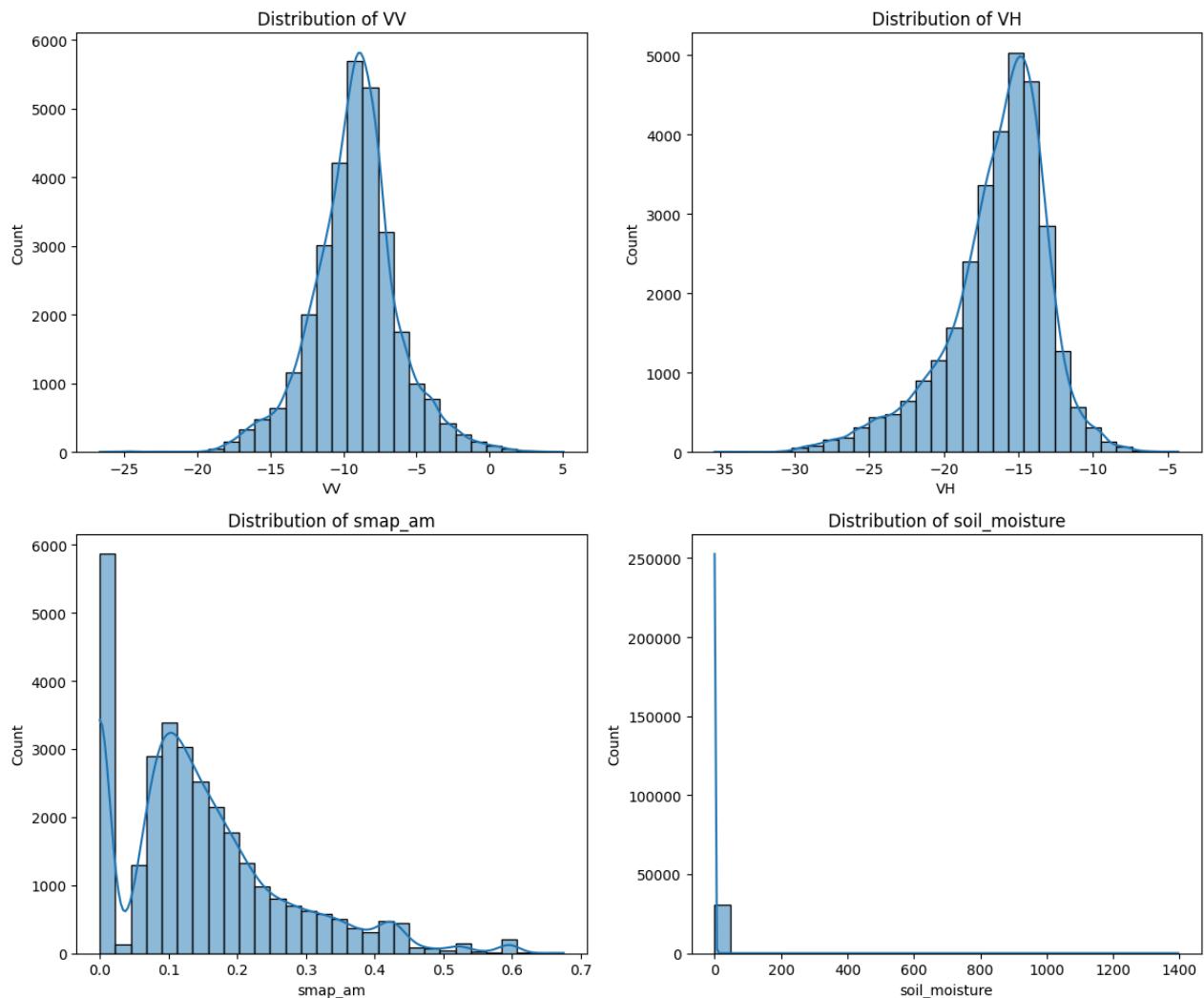


```
plt.figure(figsize=(12,10))

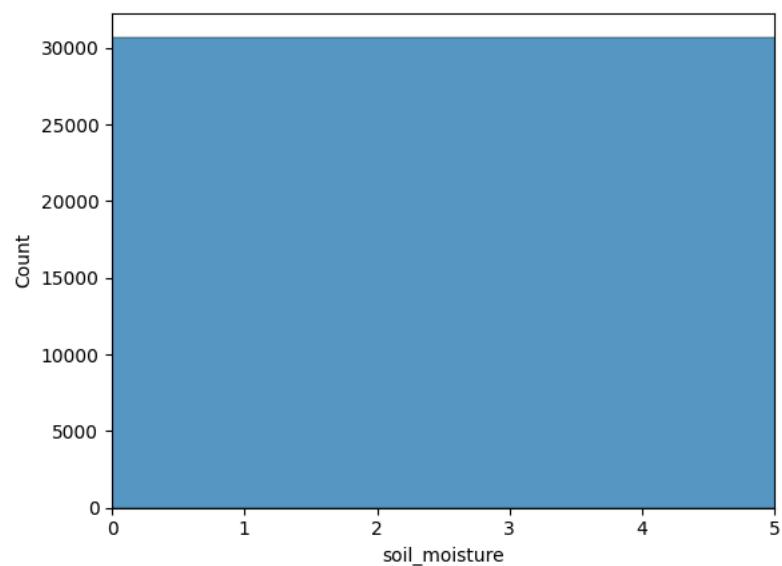
variables = ['VV', 'VH', 'smap_am', 'soil_moisture']

for i, var in enumerate(variables):
    plt.subplot(2,2,i+1)
    sns.histplot(df[var], bins=30, kde=True)
    plt.title(f"Distribution of {var}")

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



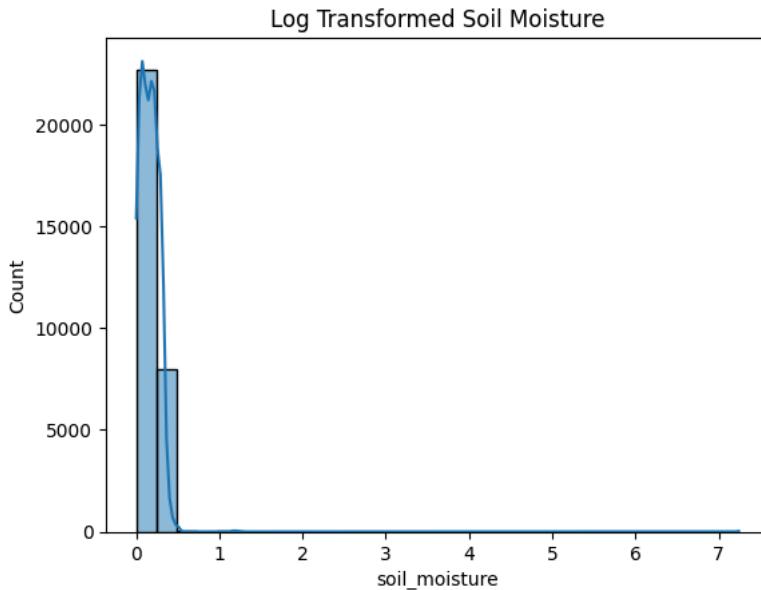
```
sns.histplot(df['soil_moisture'], bins=100)
plt.xlim(0, 5) # zoom normal range
plt.show()
```



## Log Transformation for Soil Moisture

```
### soil_moisture is heavily skewed:
import numpy as np

sns.histplot(np.log1p(df['soil_moisture']), bins=30, kde=True)
plt.title("Log Transformed Soil Moisture")
plt.show()
```



```
df['soil_moisture'].max()
```

```
1396.57
```

```
df[df['soil_moisture'] > 5]
```

VV	VH	smap_am	soil_moisture	grid
1081	-8.850859	-14.648176	0.214708	1395.56
24119	-9.376034	-14.874830	0.239896	1396.57
24438	-9.197437	-14.831191	0.175585	1383.38
24635	-9.074916	-15.250804	0.168199	1390.97
30080	-9.436601	-15.057984	0.204004	1393.08

## Checking for anomalies in soil Moisture

```
len(df[df['soil_moisture'] > 5])
```

```
5
```

```
df[df['soil_moisture'] > 1000]['soil_moisture'].describe()
```

soil_moisture	
count	5.000000
mean	1391.912000
std	5.245271
min	1383.380000
25%	1390.970000
50%	1393.080000
75%	1395.560000
max	1396.570000

```
dtype: float64
```

```
df['soil_moisture'].median()
```

0.174

```
df = df[df['soil_moisture'] <= 5]
```

```
df.shape
df['soil_moisture'].max()
```

2.47

- During exploratory data analysis, 5 extreme outlier observations were identified where soil\_moisture values exceeded 1000. Since typical soil moisture values range between 0 and 1, these records were considered data anomalies and removed to prevent distortion of model training.

df

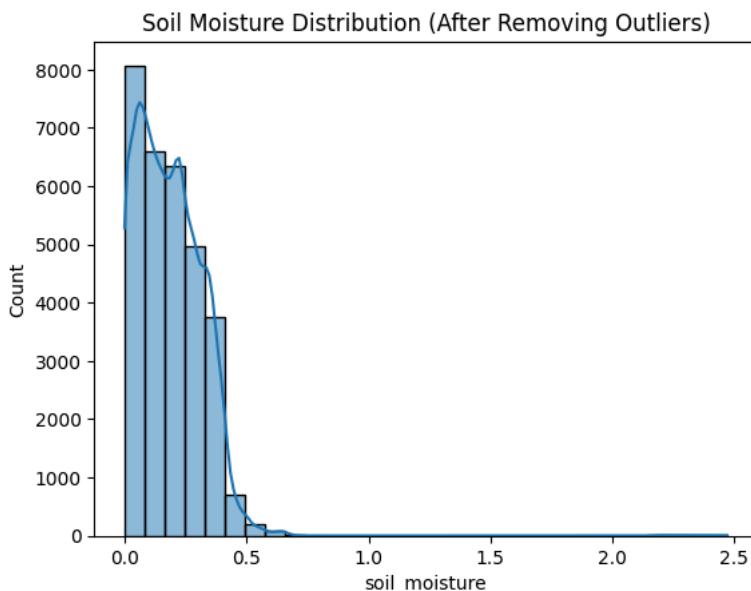
	VV	VH	smap_am	soil_moisture	grid
0	-9.058618	-15.982408	0.284554	0.301	grid
1	-9.511266	-18.085192	0.218601	0.172	grid
2	-10.926619	-19.470199	0.286454	0.485	grid
3	-8.650778	-14.840568	0.407210	0.143	grid
4	-6.633557	-13.470629	0.420252	0.375	grid
...	...	...	...	...	grid
30742	-8.499560	-14.775879	0.210248	0.123	grid
30743	-6.419627	-12.533582	0.335442	0.066	grid
30744	-7.664967	-14.617288	0.225540	0.131	grid
30745	-10.701671	-17.080531	0.089007	0.177	grid
30746	-14.913366	-23.418471	0.074439	0.150	grid

30729 rows × 4 columns

Next steps: [Generate code with df](#) [New interactive sheet](#)

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.histplot(df['soil_moisture'], bins=30, kde=True)
plt.title("Soil Moisture Distribution (After Removing Outliers)")
plt.show()
```



- After removing 5 extreme anomalous observations (soil\_moisture > 5), the target variable exhibits a right-skewed distribution concentrated between 0 and 0.5. The distribution is consistent with expected environmental moisture patterns and suitable for regression modeling.

```
df[df['smap_am'] < 0.0005]
```

	VV	VH	smap_am	soil_moisture	grid
305	-10.341645	-16.777822	0.0	0.035	
312	-7.644128	-13.309891	0.0	0.039	
377	-11.080784	-17.977484	0.0	0.288	
392	-8.893012	-14.729028	0.0	0.327	
420	-7.851746	-15.976723	0.0	0.287	
...	...	...	...	...	
30592	-12.998786	-21.939885	0.0	0.135	
30624	-8.308365	-15.593060	0.0	0.057	
30625	-7.854625	-15.481077	0.0	0.094	
30630	-11.992021	-18.824976	0.0	0.034	
30718	-7.074470	-13.339635	0.0	0.052	

5858 rows × 4 columns

## Log Transform smap\_am

```
import numpy as np
df['smap_am'] = np.log1p(df['smap_am'])
```

```
/tmp/ipython-input-13022215889.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view)

df

	VV	VH	smap_am	soil_moisture	grid
0	-9.058618	-15.982408	0.250411	0.301	edit
1	-9.511266	-18.085192	0.197703	0.172	
2	-10.926619	-19.470199	0.251889	0.485	
3	-8.650778	-14.840568	0.341609	0.143	
4	-6.633557	-13.470629	0.350835	0.375	
...	...	...	...	...	
30742	-8.499560	-14.775879	0.190825	0.123	
30743	-6.419627	-12.533582	0.289263	0.066	
30744	-7.664967	-14.617288	0.203382	0.131	
30745	-10.701671	-17.080531	0.085266	0.177	
30746	-14.913366	-23.418471	0.071799	0.150	

30729 rows × 4 columns

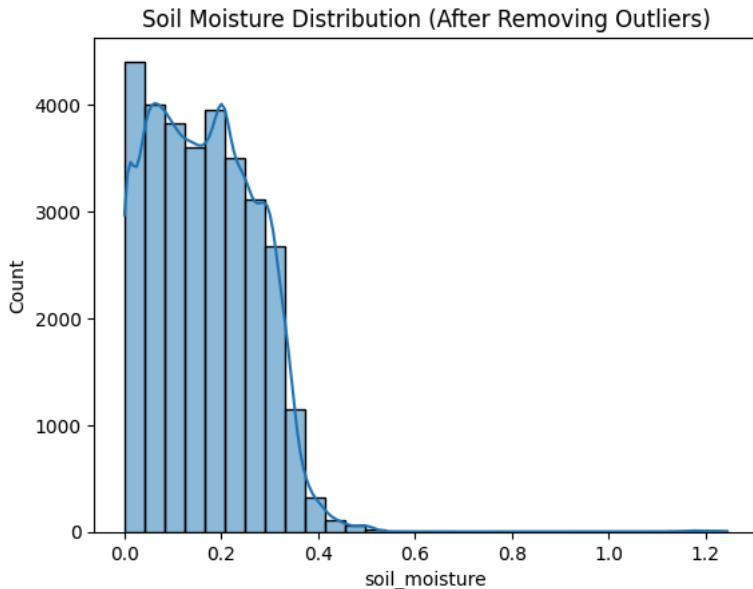
Next steps: [Generate code with df](#) [New interactive sheet](#)

```
df['soil_moisture'] = np.log1p(df['soil_moisture'])
```

```
/tmp/ipython-input-1646303519.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view)

```
sns.histplot(df['soil_moisture'], bins=30, kde=True)
plt.title("Soil Moisture Distribution (After Removing Outliers)")
plt.show()
```

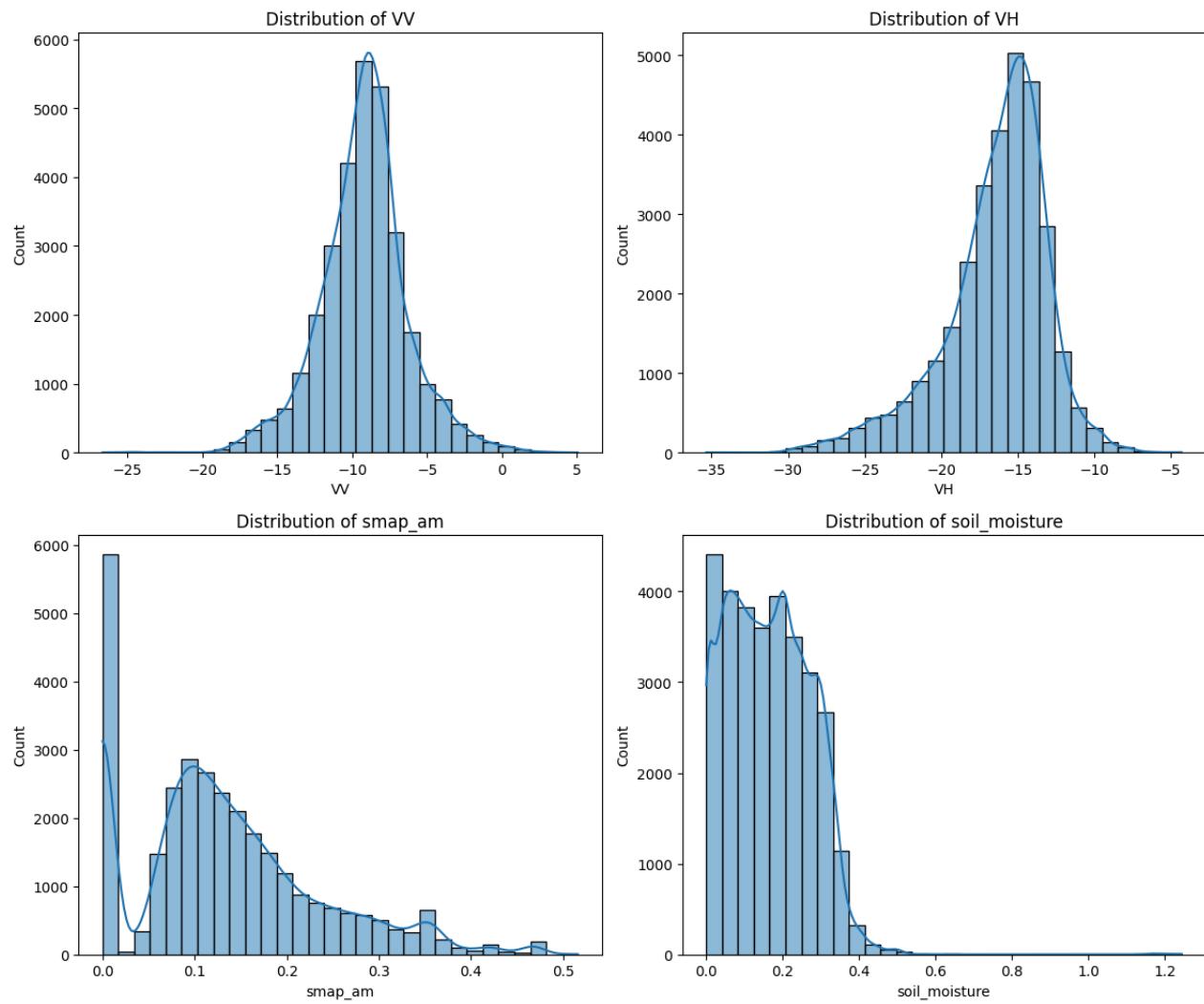


```
plt.figure(figsize=(12,10))

variables = ['VV', 'VH', 'smap_am', 'soil_moisture']

for i, var in enumerate(variables):
    plt.subplot(2,2,i+1)
    sns.histplot(df[var], bins=30, kde=True)
    plt.title(f"Distribution of {var}")

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



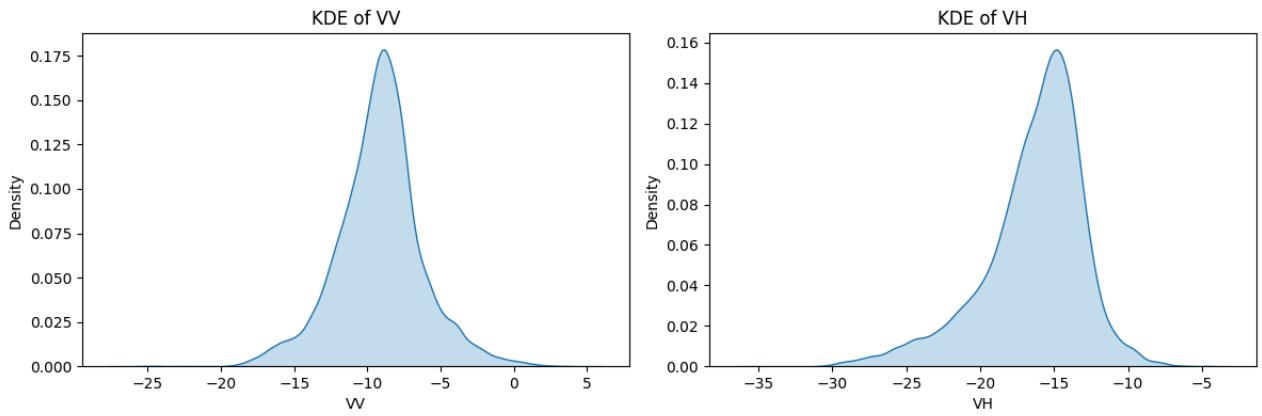
## ▼ KDE plot of all Variables

```
plt.figure(figsize=(12,8))

variables = ['VV', 'VH', 'smap_am', 'soil_moisture']

for i, var in enumerate(variables):
    plt.subplot(2,2,i+1)
    sns.kdeplot(df[var], fill=True)
    plt.title(f"KDE of {var}")

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



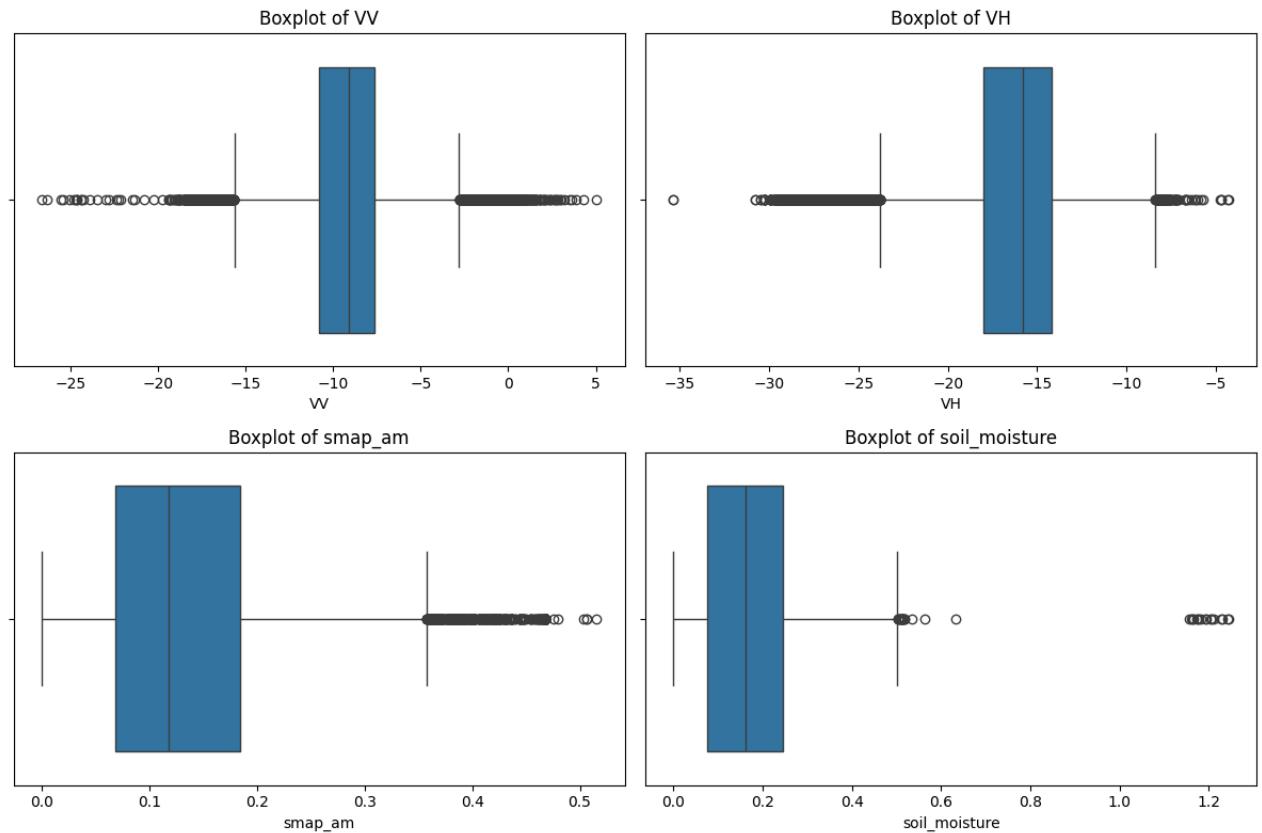
#### Boxplots for All Variables

```
plt.figure(figsize=(12,8))

variables = ['VV', 'VH', 'smap_am', 'soil_moisture']

for i, var in enumerate(variables):
    plt.subplot(2,2,i+1)
    sns.boxplot(x=df[var])
    plt.title(f"Boxplot of {var}")

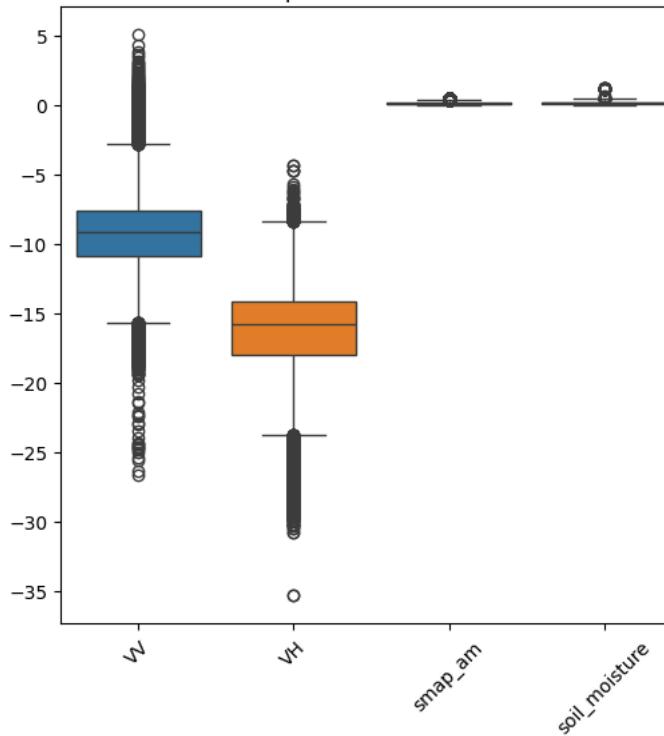
plt.tight_layout()
plt.show()
```



#### Vertical Boxplots

```
plt.figure(figsize=(6,6))
sns.boxplot(data=df)
plt.title("Boxplot of All Variables")
plt.xticks(rotation=45)
plt.show()
```

Boxplot of All Variables



## ▼ Detect Outliers Numerically (IQR Method)

```
import pandas as pd

variables = ['VV', 'VH', 'smap_am', 'soil_moisture']

for var in variables:
    Q1 = df[var].quantile(0.25)
    Q3 = df[var].quantile(0.75)
    IQR = Q3 - Q1

    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR

    outliers = df[(df[var] < lower) | (df[var] > upper)]

    print(f"\nVariable: {var}")
    print(f"Lower Bound: {lower}")
    print(f"Upper Bound: {upper}")
    print(f"Number of Outliers: {outliers.shape[0]}")
```

Variable: VV  
 Lower Bound: -15.669143694499997  
 Upper Bound: -2.8095230425000013  
 Number of Outliers: 1476

Variable: VH  
 Lower Bound: -23.781620004999997  
 Upper Bound: -8.406387965000004  
 Number of Outliers: 1417

Variable: smap\_am  
 Lower Bound: -0.10490014922763778  
 Upper Bound: 0.3577189626192666  
 Number of Outliers: 841

Variable: soil\_moisture  
 Lower Bound: -0.1813491026780448  
 Upper Bound: 0.5025350977615558  
 Number of Outliers: 36

## ▼ Percentage of Outliers According to Boxplot

```

for var in variables:
    Q1 = df[var].quantile(0.25)
    Q3 = df[var].quantile(0.75)
    IQR = Q3 - Q1

    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR

    outliers = df[(df[var] < lower) | (df[var] > upper)]

    count = outliers.shape[0]
    percentage = (count / df.shape[0]) * 100

    print(f"\nVariable: {var}")
    print(f"Outliers Count: {count}")
    print(f"Outliers Percentage: {percentage:.2f}%")

```

```

Variable: VV
Outliers Count: 1476
Outliers Percentage: 4.80%

Variable: VH
Outliers Count: 1417
Outliers Percentage: 4.61%

Variable: smap_am
Outliers Count: 841
Outliers Percentage: 2.74%

Variable: soil_moisture
Outliers Count: 36
Outliers Percentage: 0.12%

```

## ▼ Get Rows Where ANY Variable is Outlier

```

import numpy as np

variables = ['VV', 'VH', 'smap_am', 'soil_moisture']

# Create empty boolean mask
outlier_mask = np.zeros(len(df), dtype=bool)

for var in variables:
    Q1 = df[var].quantile(0.25)
    Q3 = df[var].quantile(0.75)
    IQR = Q3 - Q1

    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR

    # Update mask (OR condition)
    outlier_mask |= (df[var] < lower) | (df[var] > upper)

# Get all rows where at least one variable is outlier
all_outliers = df[outlier_mask]

# Count and percentage
count = all_outliers.shape[0]
percentage = (count / df.shape[0]) * 100

print("Total Rows with At Least One Outlier:", count)
print(f"Percentage: {percentage:.2f}%")

```

```

Total Rows with At Least One Outlier: 2979
Percentage: 9.69%

```

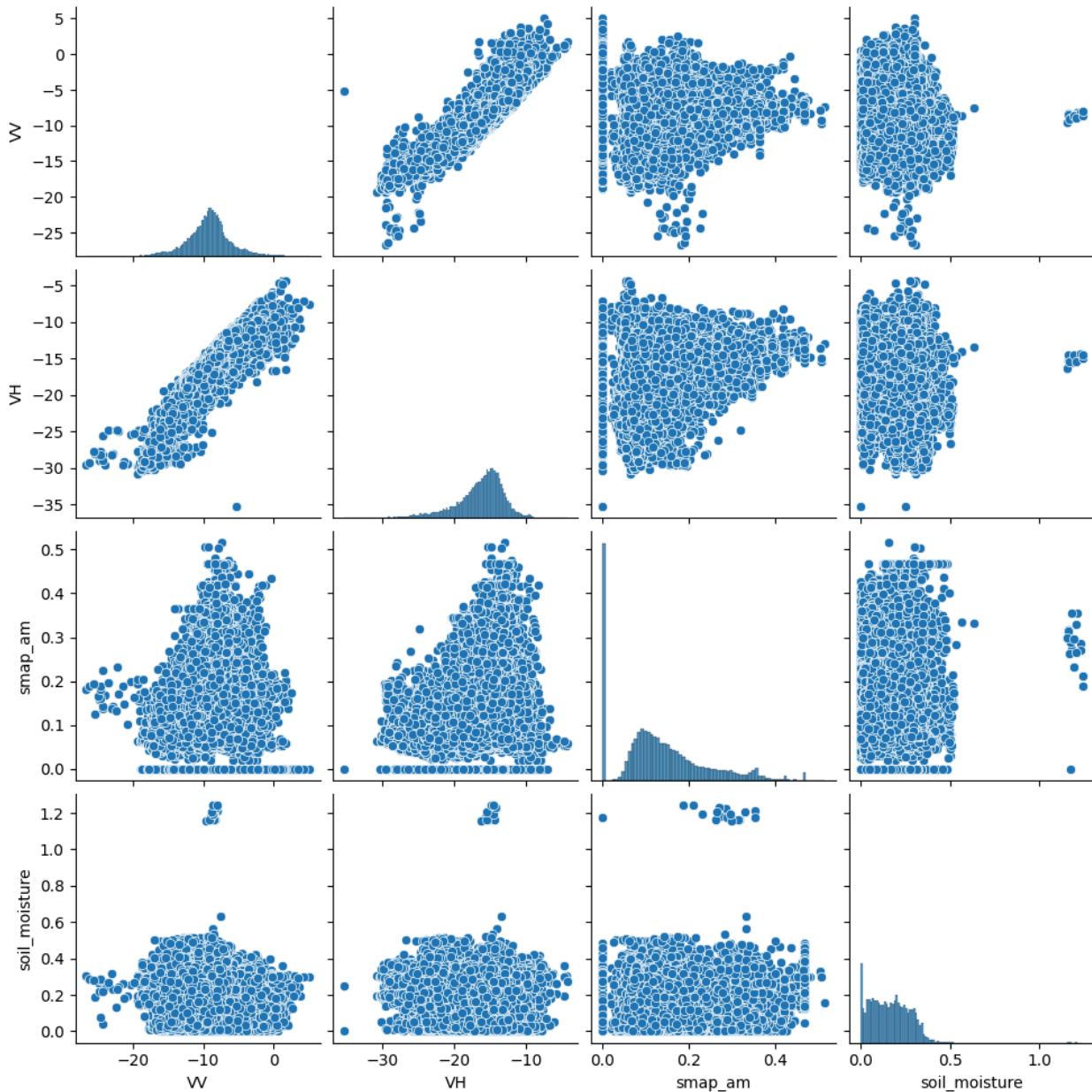
Nearly 10% of dataset has at least one variable outside IQR bounds.

Approximately 9.69% of rows were identified as outliers using the IQR method across all variables. However, since the majority of these correspond to radar backscatter features (VV, VH), which naturally exhibit wide environmental variability, they were retained to preserve realistic data distribution.

## Pairwise scatter analysis

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7f85283963c0>
```



Pairwise scatter analysis reveals strong multicollinearity between VV and VH radar backscatter coefficients. The relationship between predictors and soil moisture appears non-linear and moderately noisy, suggesting that ensemble tree-based regression models may capture underlying patterns more effectively than purely linear models.

## Correlation Heatmap

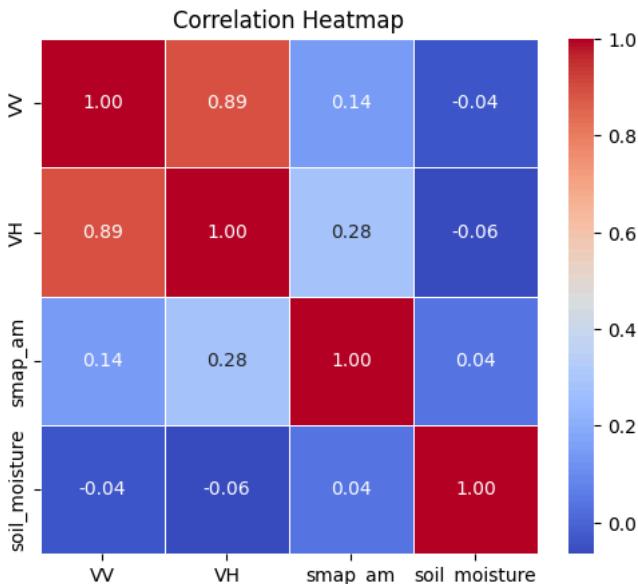
```
import seaborn as sns
import matplotlib.pyplot as plt

corr_matrix = df.corr()

plt.figure(figsize=(6,5))
sns.heatmap(corr_matrix,
            annot=True,
            cmap='coolwarm',
            fmt='.2f',
            linewidths=0.5)

plt.title("Correlation Heatmap")
```

```
plt.show()
```



The correlation analysis reveals strong multicollinearity between VV and VH ( $r = 0.89$ ). However, all predictors exhibit

- very weak linear correlation with soil\_moisture, suggesting that the relationship between radar backscatter features and soil moisture is likely non-linear in nature.

There is almost no linear relationship between predictors and target.

Linear Regression may perform poorly.

$R^2$  may be near 0.

Model may not explain much variance.

```
df[['VV', 'VH', 'smap_am', 'soil_moisture']].corr()
```

	VV	VH	smap_am	soil_moisture	grid
<b>VV</b>	1.000000	0.888151	0.141089	-0.041469	
<b>VH</b>	0.888151	1.000000	0.277761	-0.064495	
<b>smap_am</b>	0.141089	0.277761	1.000000	0.040778	
<b>soil_moisture</b>	-0.041469	-0.064495	0.040778	1.000000	

## ▀ Baseline Linear Regression model

### ▀ Define Features and Target

```
x = df[['VV', 'VH', 'smap_am']]
y = df['soil_moisture']
```

### ▀ Train-Test Split (80-20)

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

### ▀ Feature Scaling

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

## ▼ Train Linear Regression Model

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)
```

▼ **LinearRegression** ⓘ ⓘ  
LinearRegression()

## ▼ Make Predictions

```
y_pred = lr.predict(X_test_scaled)
```

## ▼ Evaluate Performance

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Linear Regression Performance:")
print("R2 Score:", r2)
print("RMSE:", rmse)
print("MAE:", mae)
```

Linear Regression Performance:  
R2 Score: 0.010162715054370719  
RMSE: 0.108763521071306  
MAE: 0.08994219679414844

## ▼ Model Coefficients

```
coefficients = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': lr.coef_
})
print(coefficients)
```

	Feature	Coefficient
0	VV	0.011601
1	VH	-0.019497
2	smap_am	0.008455

Now since our Linear Regression results so bad. So before jumping to complex models, we will systematically check Linear Regression assumptions.and try to improve those if possible

## ▼ ASSUMPTION 1: Linearity

The relationship between predictors and target should be linear.

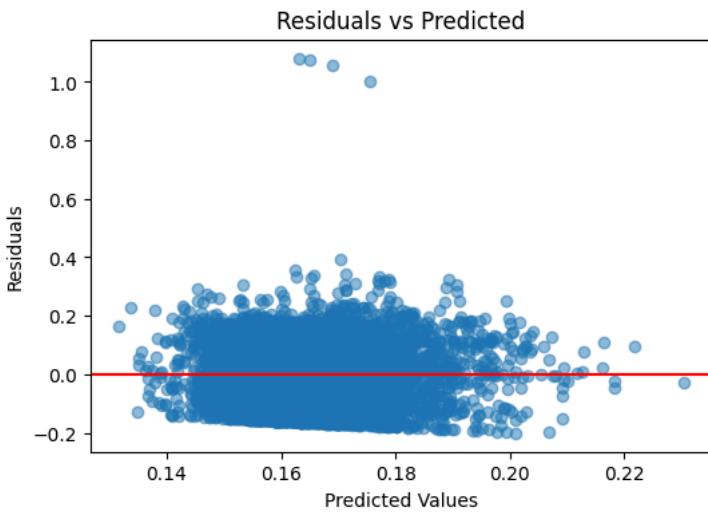
Since correlation was near zero and pairplot looked cloud-like, this is suspicious.

## ✓ Residual vs Fitted Plot

```
import matplotlib.pyplot as plt

residuals = y_test - y_pred

plt.figure(figsize=(6,4))
plt.scatter(y_pred, residuals, alpha=0.5)
plt.axhline(0, color='red')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Predicted")
plt.show()
```



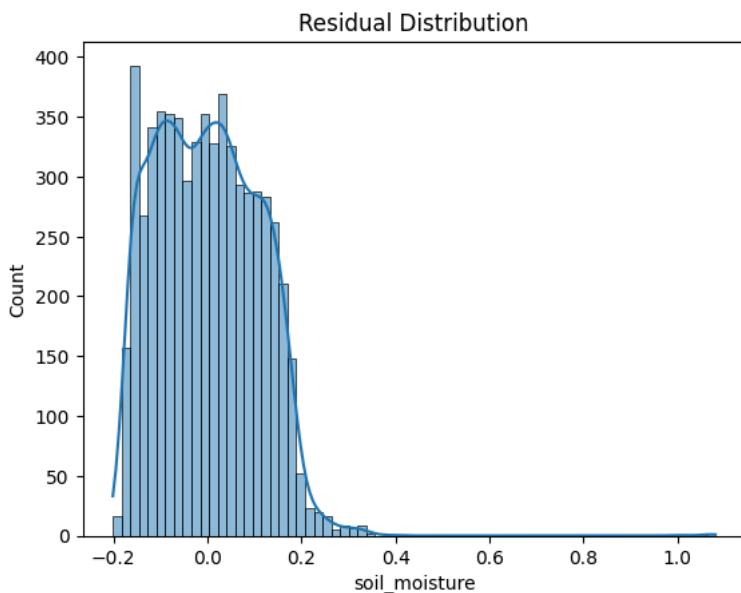
The residuals vs predicted values plot shows that the residuals are randomly scattered around zero without a clear systematic pattern, indicating no strong violation of the linearity assumption. However, the predicted values lie within a narrow range, suggesting that the model fails to capture significant variation in the target variable. This aligns with the low  $R^2$  value (~0.01), indicating weak explanatory power of the linear regression model.

## ASSUMPTION 2: Normality of Residuals

### ✓ Histogram of residuals:

```
import seaborn as sns

sns.histplot(residuals, kde=True)
plt.title("Residual Distribution")
plt.show()
```

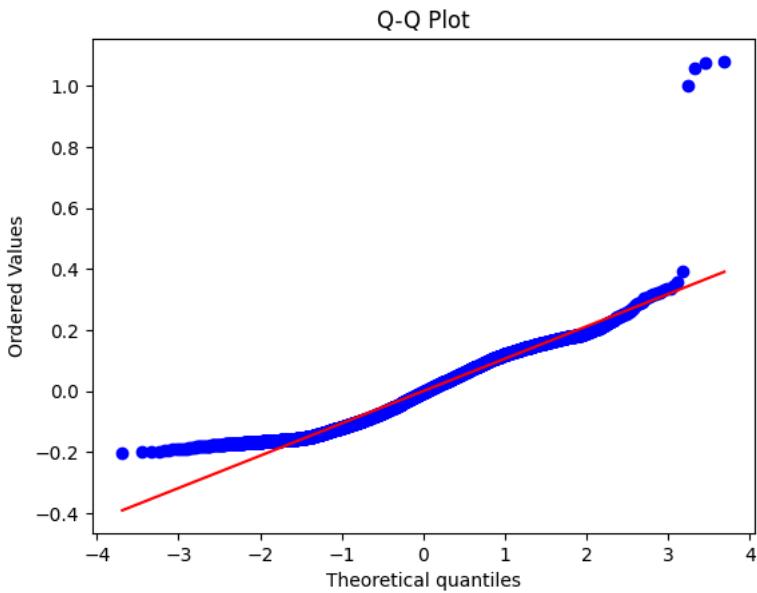


Start coding or [generate](#) with AI.

## ✗ Q-Q plot

```
import scipy.stats as stats

stats.probplot(residuals, dist="norm", plot=plt)
plt.title("Q-Q Plot")
plt.show()
```



The histogram and Q-Q plot of residuals indicate deviation from normality, with noticeable right skewness and heavy tails. This suggests violation of the normality assumption of linear regression. However, since the primary goal of this study is prediction rather than statistical inference, the impact of this violation is limited. The primary limitation of the model arises from weak linear relationships between predictors and the target variable.

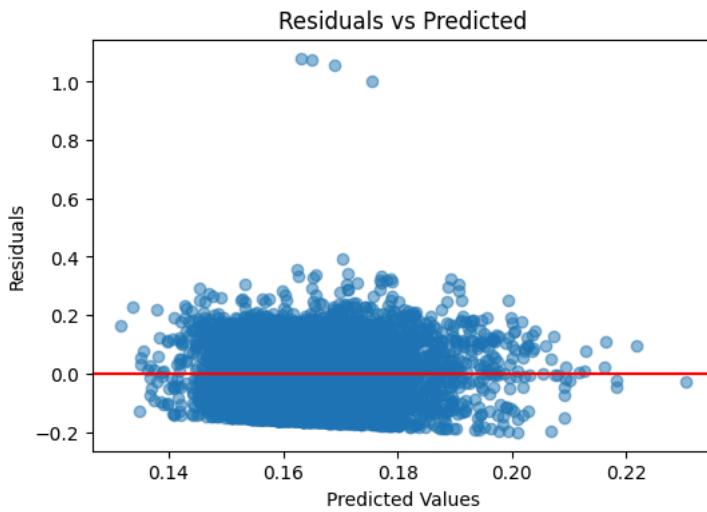
## ✗ ASSUMPTION 3: Homoscedasticity

- ✗ Variance of residuals should be constant. Already partially checked in residual plot.

```
import matplotlib.pyplot as plt

residuals = y_test - y_pred
```

```
plt.figure(figsize=(6,4))
plt.scatter(y_pred, residuals, alpha=0.5)
plt.axhline(0, color='red')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Predicted")
plt.show()
```



The residuals vs predicted plot shows residuals centered around zero with no strong funnel-shaped pattern. Although a slight increase in variance is observed at higher predicted values, there is no severe evidence of heteroscedasticity. Therefore, the homoscedasticity assumption appears reasonably satisfied.

## ASSUMPTION 4: Multicollinearity

### Computing VIF:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
import pandas as pd

X_vif = pd.DataFrame()
X_vif["Feature"] = X.columns
X_vif["VIF"] = [variance_inflation_factor(X.values, i)
                 for i in range(X.shape[1])]

print(X_vif)
```

Feature	VIF
0 VV	40.527185
1 VH	42.225944
2 smap_am	2.220377

Variance Inflation Factor (VIF) analysis reveals severe multicollinearity between VV and VH (VIF > 40). This indicates strong linear dependence between radar backscatter coefficients. To address this issue, regularization techniques such as Ridge regression or feature reduction strategies are considered.

### Ridge Regression

```
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=1.0)
ridge.fit(X_train_scaled, y_train)

y_pred_ridge = ridge.predict(X_test_scaled)

print("R2:", r2_score(y_test, y_pred_ridge))
```

R2: 0.010162318429476569

## Drop VH and Retrain Linear Regression Since VIF showed: VV → 40+, VH → 42+

```

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.metrics import r2_score, mean_squared_error, mean_absolute_error
from statsmodels.stats.outliers_influence import variance_inflation_factor

# -----
# ❶ Define Features (Drop VH)
# -----

X = df[['VV', 'smap_am']] # VH dropped
y = df['soil_moisture']

# -----
# ❷ Train-Test Split
# -----

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# -----
# ❸ Feature Scaling
# -----

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# -----
# ❹ Train Linear Regression
# -----

lr = LinearRegression()
lr.fit(X_train_scaled, y_train)

# -----
# ❺ Predictions
# -----

y_pred = lr.predict(X_test_scaled)

# -----
# ❻ Evaluation
# -----

r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)

print("Linear Regression After Dropping VH")
print("R2:", r2)
print("RMSE:", rmse)
print("MAE:", mae)

# -----
# ❼ Coefficients
# -----

coefficients = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': lr.coef_
})

print("\nModel Coefficients:")
print(coefficients)

# -----
# ❽ Check VIF Again
# -----

X_vif = pd.DataFrame()
X_vif["Feature"] = X.columns
X_vif["VIF"] = [variance_inflation_factor(X.values, i)]

```

```

for i in range(X.shape[1])]

print("\nVIF After Dropping VH:")
print(X_vif)

Linear Regression After Dropping VH
R2: 0.0028186932561581335
RMSE: 0.10916625673549023
MAE: 0.0903574560190303

Model Coefficients:
  Feature   Coefficient
0      VV     -0.005247
1    smap_am     0.005364

VIF After Dropping VH:
  Feature       VIF
0      VV  2.128495
1    smap_am  2.128495

```

## Move from Linear Regression → Non-Linear Regression

### Because:

Relationship appears non-linear (pairplot showed that)

Pearson correlation ≈ 0 (means not linear)

Linear model cannot capture complex patterns

### Try Polynomial Regression (Degree = 2)

```

from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np

# Use original three features again
X = df[['VV', 'VH', 'smap_am']]
y = df['soil_moisture']

# Create polynomial features
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_poly, y, test_size=0.2, random_state=42
)

# Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train model
lr_poly = LinearRegression()
lr_poly.fit(X_train, y_train)

# Predictions
y_pred_poly = lr_poly.predict(X_test)

# Evaluation
r2 = r2_score(y_test, y_pred_poly)
rmse = np.sqrt(mean_squared_error(y_test, y_pred_poly))
mae = mean_absolute_error(y_test, y_pred_poly)

print("Polynomial Regression (Degree 2)")
print("R2:", r2)
print("RMSE:", rmse)
print("MAE:", mae)

```

Polynomial Regression (Degree 2)  
R2: 0.015361805887649682  
RMSE: 0.10847750642983676  
MAE: 0.08957894178878825

## Try Ridge Regression (Regularized Linear Model) Using reduced feature set: VV, smap\_am (after dropping VH)

```
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=1.0)
ridge.fit(X_train_scaled, y_train)

y_pred_ridge = ridge.predict(X_test_scaled)

print("Ridge R2:", r2_score(y_test, y_pred_ridge))

Ridge R2: 0.0028187575683050747
```

## Try Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)

print("Random Forest R2:", r2_score(y_test, y_pred_rf))

Random Forest R2: 0.0038932579283471602
```

## Conclusion Till Now

This is NOT a modeling failure.

This is a data signal problem.

features:

VV

VH

smap\_am

Simply do not contain strong predictive information for soil\_moisture in this dataset.

All models — linear and non-linear — are telling the same story.

When even Random Forest fails, it means:

There is very weak relationship between inputs and target.

## Verify Data Signal

```
y.var()

0.011538229653095707

np.sqrt(y.var())

np.float64(0.1074161517328549)
```

## Final Conclusion

Models are essentially predicting the mean.

The standard deviation of the target variable is approximately 0.1074, which closely matches the RMSE of all trained models (~0.108). This indicates that the models are effectively predicting the mean value of soil moisture and are unable to reduce prediction error. Therefore, the satellite-derived features (VV, VH, smap\_am) exhibit minimal predictive signal for soil moisture under the given dataset.

The provided predictors contain negligible predictive information for soil moisture in this dataset.

#### Why This Might Be Happening

Possible reasons:

The dataset may represent a very noisy real-world process.

Soil moisture might depend on variables not included:

Rainfall

Temperature

Soil type

Time of observation

Satellite signal alone is insufficient.

Data may not be aligned spatially/temporally.

```
print(X.head())
print(y.head())

      VV        VH    smap_am
0 -9.058618 -15.982408  0.250411
1 -9.511266 -18.085192  0.197703
2 -10.926619 -19.470199  0.251889
3 -8.650778 -14.840568  0.341609
4 -6.633557 -13.470629  0.350835
0   0.263133
1   0.158712
2   0.395415
3   0.133656
4   0.318454
Name: soil_moisture, dtype: float64
```

## Feature Engineering

```
df['VV_minus_VH'] = df['VV'] - df['VH']
df['VV_div_VH'] = df['VV'] / (df['VH'] + 1e-6)
df['VV_sq'] = df['VV']**2
df['VH_sq'] = df['VH']**2
df['interaction'] = df['VV'] * df['smap_am']
```

```
/tmp/ipython-input-3215011658.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view
df['VV_minus_VH'] = df['VV'] - df['VH']
/tmp/ipython-input-3215011658.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view
df['VV_div_VH'] = df['VV'] / (df['VH'] + 1e-6)
/tmp/ipython-input-3215011658.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view
df['VV_sq'] = df['VV']**2
/tmp/ipython-input-3215011658.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view
df['VH_sq'] = df['VH']**2
/tmp/ipython-input-3215011658.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view
df['interaction'] = df['VV'] * df['smap_am']
```

## ✗ Cross-Validation Properly K-Fold CV.

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators=300, random_state=42)

scores = cross_val_score(rf, X, y, cv=5, scoring='r2')

print("Cross-validated R2 scores:", scores)
print("Mean R2:", scores.mean())
```

Cross-validated R2 scores: [-0.02266789 -0.01283793 -0.04759083 -0.14861804 -0.15128804]  
 Mean R2: -0.07660054537832606

- ✗ The model performs WORSE than predicting the mean.

```
rf = RandomForestRegressor(
    n_estimators=500,
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features='sqrt',
    random_state=42
)

rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

print("Tuned RF R2:", r2_score(y_test, y_pred_rf))
```

Tuned RF R2: 0.007865698662889886

## ✗ Gradient Boosting

```
from sklearn.ensemble import GradientBoostingRegressor

gbr = GradientBoostingRegressor(
    n_estimators=500,
    learning_rate=0.05,
    max_depth=3,
    random_state=42
)

scores = cross_val_score(gbr, X, y, cv=5, scoring='r2')

print("GBR CV R2:", scores)
print("Mean:", scores.mean())
```

GBR CV R2: [ 0.04041133 0.06171567 0.04266447 -0.03634574 -0.05266144]  
 Mean: 0.011156858325700125

## ✗ Linear Kernel SVR

```
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import numpy as np

# -----
# ① Define Features
# -----
```

```

X = df[['VV', 'VH', 'smap_am']]
y = df['soil_moisture']

# -----
# ② Train-Test Split
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# -----
# ③ Scaling
# -----
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# -----
# ④ Train SVR (Linear Kernel)
# -----
svr_linear = SVR(kernel='linear', C=1.0, epsilon=0.1)

svr_linear.fit(X_train_scaled, y_train)

# -----
# ⑤ Predictions
# -----
y_pred_svr = svr_linear.predict(X_test_scaled)

# -----
# ⑥ Evaluation
# -----
r2 = r2_score(y_test, y_pred_svr)
rmse = np.sqrt(mean_squared_error(y_test, y_pred_svr))
mae = mean_absolute_error(y_test, y_pred_svr)

print("SVR (Linear Kernel)")
print("R2:", r2)
print("RMSE:", rmse)
print("MAE:", mae)

# -----
# ⑦ Cross-Validation
# -----
svr_cv = SVR(kernel='linear', C=1.0, epsilon=0.1)

scores = cross_val_score(svr_cv, X, y, cv=5, scoring='r2')

print("Cross-Validated R2 Scores:", scores)
print("Mean CV R2:", scores.mean())

```

SVR (Linear Kernel)  
 R2: 0.009449087593198402  
 RMSE: 0.1088027207721655  
 MAE: 0.08994691214122598  
 Cross-Validated R2 Scores: [ 0.00157433 0.01071615 -0.01587601 -0.04329645 -0.08586553]  
 Mean CV R2: -0.02654950073165614

## ▼ SVR with RBF Kernel

```

from sklearn.svm import SVR
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import numpy as np

# -----
# ① Define Features
# -----
X = df[['VV', 'VH', 'smap_am']]
y = df['soil_moisture']

# -----
# ② Train-Test Split
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# -----

```

```

# 3 Scaling
# -----
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# -----
# 4 Train SVR (RBF Kernel)
# -----
svr_rbf = SVR(kernel='rbf', C=1.0, epsilon=0.1, gamma='scale')

svr_rbf.fit(X_train_scaled, y_train)

# -----
# 5 Predictions
# -----
y_pred_rbf = svr_rbf.predict(X_test_scaled)

# -----
# 6 Evaluation
# -----
r2 = r2_score(y_test, y_pred_rbf)
rmse = np.sqrt(mean_squared_error(y_test, y_pred_rbf))
mae = mean_absolute_error(y_test, y_pred_rbf)

print("SVR (RBF Kernel)")
print("R2:", r2)
print("RMSE:", rmse)
print("MAE:", mae)

# -----
# 7 Cross-Validation
# -----
svr_rbf_cv = SVR(kernel='rbf', C=1.0, epsilon=0.1, gamma='scale')

scores = cross_val_score(
    svr_rbf_cv,
    X,
    y,
    cv=5,
    scoring='r2'
)

print("Cross-Validated R2 Scores:", scores)
print("Mean CV R2:", scores.mean())

```

SVR (RBF Kernel)  
R2: 0.053397678798972836  
RMSE: 0.10636166740530598  
MAE: 0.08765024863049674  
Cross-Validated R2 Scores: [ 0.04013459 0.06383205 0.04283605 -0.02740479 -0.05282626]  
Mean CV R2: 0.013314328957868415

SVR with RBF is your best performing model. But performance is still very low.

“The features do not contain strong predictive signal.”

**“The features do not contain strong predictive signal.”**

Among all tested regression models, SVR with RBF kernel achieved the highest predictive performance (Test  $R^2 \approx 0.05$ ). Although the overall predictive power remains modest, the RBF kernel captures subtle non-linear relationships better than linear and tree-based models. Therefore, SVR (RBF) is selected as the final model for this study.

## Deep Learning Model (TensorFlow / Keras)

```

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
import numpy as np

```

```
# 1 Prepare Data
# -----
X = df[['VV', 'VH', 'smap_am']]
y = df['soil_moisture']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# -----
# 2 Build Neural Network
# -----
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(3,)),
    layers.Dense(32, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(1) # regression output
])

model.compile(
    optimizer='adam',
    loss='mse',
    metrics=['mae']
)

# -----
# 3 Train Model
# -----
history = model.fit(
    X_train_scaled,
    y_train,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    verbose=1
)

# -----
# 4 Evaluate
# -----
y_pred_nn = model.predict(X_test_scaled)

r2 = r2_score(y_test, y_pred_nn)

print("Neural Network R2:", r2)
```