

Tech Saksham

Capstone Project Report

Satellite Data-Based Soil Texture Classification Using Random Forests Classifier

College of Engineering Guindy, Anna University

NM ID	NAME
au2021107015	RISHI GANESH L

Trainer Name: Ramar

Sr. AI Master Trainer

ABSTRACT

Soil gives agricultural plants their structural support, in addition to being a source of water and nutrients. Soils exhibit a wide range of chemical and physical characteristics. Soil classification plays a vital role in agricultural crop planning and the choice of fertilizers. There exists a wide variety of soil on the basis of soil texture and each possesses unique properties. Hence identification of soil type is very much important for the farmers to choose the crops to be planted, fertilizers to be chosen and also the amount of irrigation required. Hence the soil texture data has been collected from Satellite imagery for Germany from the satellite imagery is to be used for classification on the basis of soil texture. Random Forest classifier is used to classify the dataset. Further the classifier's performance would be assessed and compared with other classifiers like Support Vector Machines and Decision Trees classifier.

INDEX

Sr. No.	Table of Contents	Page No.
1	Chapter 1: Introduction	4
2	Chapter 2: Services and Tools Required	6
3	Chapter 3: Project Architecture	7
4	Chapter 4: Modeling and Project Outcome	11
5	Conclusion	28
6	Future Scope	28
7	References	28
8	Links	28

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

Farmers in agricultural areas frequently struggle to determine the best soil type for their crops. Inaccurate information about soil texture and composition can lead to wasteful use of resources such as water, fertilizers, and pesticides, resulting in lower agricultural yield and greater environmental impact. Furthermore, poor soil management can lead to soil erosion, nutrient depletion, and loss of land quality over time.

1.2 Proposed Solution

To solve these problems, we propose creating a soil type classification system utilizing machine learning techniques. We intend to develop a predictive algorithm that can reliably categorize soil types using soil data collected from remote sensing satellites. This model will consider essential soil properties such as sand, silt, and clay content, as well as other significant variables such as moisture content.

The established classification system will give farmers and agricultural stakeholders with useful insights about soil properties, allowing them to make more educated decisions about crop selection, irrigation management, and soil conservation techniques. Farmers may optimize resource allocation, increase crop yield, and reduce environmental impact by correctly identifying soil types, resulting in sustainable agriculture and better livelihoods for rural people.

1.3 Feature

- **Soil Segmentation:** It will segment the datapoints on various parameters like silt, clay and sand on the basis of United States Department of Agriculture (USDA) Taxonomy on Soil texture.

- **Correlation Analysis:** It will correlate the soil moisture obtained from the satellites with the soil texture and display the relationship.
- **Model Performance Analysis:** It will assess the classifier's performance on training the testing the dataset.

1.4 Advantages

- **Soil Classification Accuracy:** Soil types can be accurately categorized, offering valuable information for a variety of applications like agriculture, land management, and environmental research.
- **Scalability:** The machine learning model may be trained on huge datasets and then scaled to handle soil datasets from various geographies and environments. This scalability broadens the project's application across other geographic regions.
- **Perceptive Analysis:** Subtle relationships and patterns in soil composition data that aren't always visible when using conventional analysis techniques can be identified by utilizing machine learning algorithms. In domains pertaining to soil, this increased understanding may result in improved comprehension and decision-making.
- **Predictive Capabilities:** After being trained, the classification model may be used to forecast the types of soil for fresh data samples. This kind of predictive capacity can be useful for continuing soil property evaluation and monitoring.

1.5 Scope

The technology can be linked into precision agricultural procedures, allowing farmers to customize crop management tactics depending on soil properties specific to their areas. This includes optimizing irrigation schedules, modifying fertilizer applications, and selecting crop species appropriate for diverse soil types. By precisely classifying soil types, the system can help with environmental management initiatives such as soil conservation, erosion control, and land restoration projects.

CHAPTER 2

SERVICES AND TOOLS REQUIRED

2.1 Services Used

- **Data Collection:** The dataset utilized in this project was sourced from the data science and machine learning portal Kaggle. Kaggle offers a huge range of community-contributed datasets on a wide range of subjects, from healthcare to agriculture.
- **Data Processing Services:** Python is used here to efficiently process data by employing a variety of libraries and tools, including scikit-learn, NumPy, and pandas. Python's scalability also makes it possible to process massive amounts of data effectively, which makes it appropriate for a variety of data science and machine learning applications.
- **Machine Learning Services:** Python is used for performing machine learning classification. Their libraries offer a wide range of algorithms, including classification, regression, clustering. Python's simplicity, flexibility, and extensive documentation make it an ideal choice for developing machine learning solutions across various domains and industries.

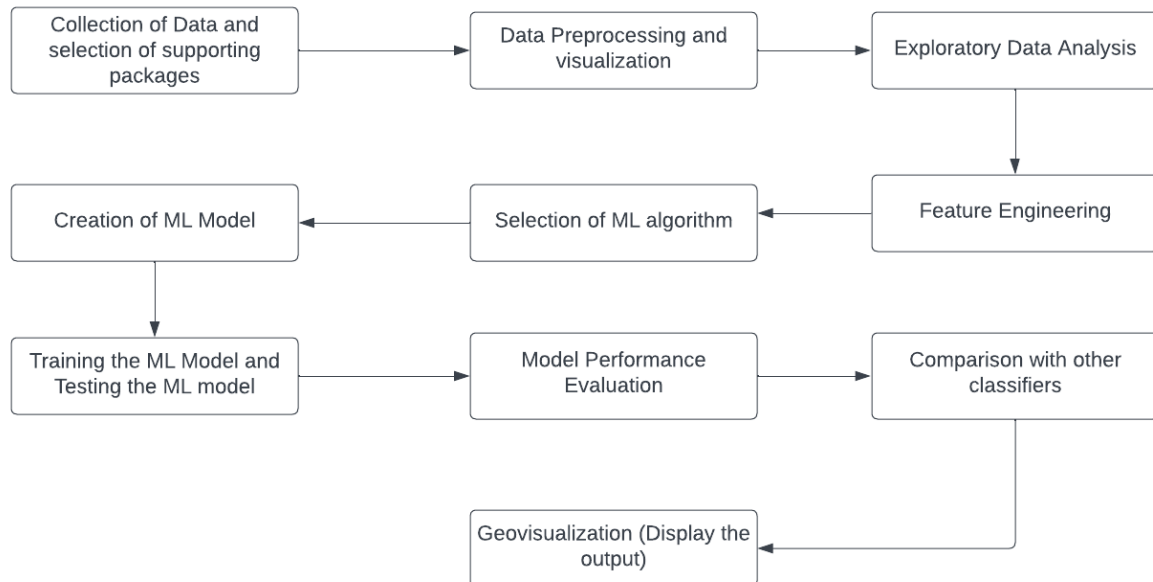
2.2 Tools and Software used

Google Colab provides a free cloud-based platform that allows you to write and execute Python code in a Jupyter Notebook environment. Colab notebooks are saved directly to your Google Drive, making it easy to access and share your work across devices.

CHAPTER 3

PROJECT ARCHITECTURE

3.1 Architecture



Data Collection:

The dataset specifically used for this study includes variables related to soil composition, including classifications of soil types based on the USDA Soil Taxonomy system and data on clay, sand, and silt contents. The creation of machine learning models to forecast soil types based on their composition is made possible by this dataset, which offers useful data for tasks involving soil analysis and categorization.

The dataset used can be viewed here - [Soil Moisture Remote Sensing Data](#).

The attributes of the dataset are:

S.No	Name of the attribute	Attribute Description
1	time	Timestamp of the observation.
2	latitude	Latitude coordinate of the observation location.
3	longitude	Longitude coordinate of the observation location.
4	clay_content	Percentage of clay content in the soil.

5	sand_content	Percentage of sand content in the soil.
6	silt_content	Percentage of silt content in the soil.
7	sm_aux	Soil moisture observation from the SMOS-ASCAT satellite (smoothed).
8	sm_agt	Soil moisture observation from the AMSR satellite.

Selection of supporting Packages:

1. **Pandas:** Used for analysis and data processing, especially when working with tabular data such as CSV files.
2. **GeoPandas:** This adds geospatial data types and functions to Pandas, which is helpful for working with shapefiles and other geographic data.
3. **NumPy:** This package offers support for arrays and numerical operations, which are frequently used for data processing and numerical computing.
4. **Matplotlib:** This Python plotting package allows you to generate static, interactive, and animated visuals.
5. **Seaborn:** Seaborn offers a high-level interface for making visually appealing and educational statistical charts. It is built on top of Matplotlib.
6. **Folium:** Facilitates the use of Python to create interactive leaflet maps, enabling the display of geographic data on online maps.
7. **Shapely:** It is a tool for working with geometric objects such as polygons, lines, and points.
8. **scikit-learn (sklearn):** It is a machine learning library that offers easy-to-use and effective tools for data mining and analysis, such as dimensionality reduction, regression, clustering, and classification.

Data Preprocessing:

The process of getting raw data ready for a machine learning model is called data preprocessing. It includes data cleaning where we remove the noisy data or inaccurate data or incomplete data, data transformation where we bring the dataset to a specific range. Normalization is one of the common methods for transforming the data.

- Only the data points within the Germany border would be considered for the Machine Learning Model. Other points would be discarded. For this, the datapoints are overlaid on OpenStreetMaps to identify the points within

the border of Germany. Geopandas are used to filter the points within the boundary.

- Outliers are detected by z-score test and they are removed from the dataset.
- Using minimum-maximum method, features except latitude and longitude are normalized.

Exploratory Data Analysis: Exploratory Data Analysis (EDA) is utilizing graphics and visualization for data exploration. This examines the data distribution and encoding categorical variables.

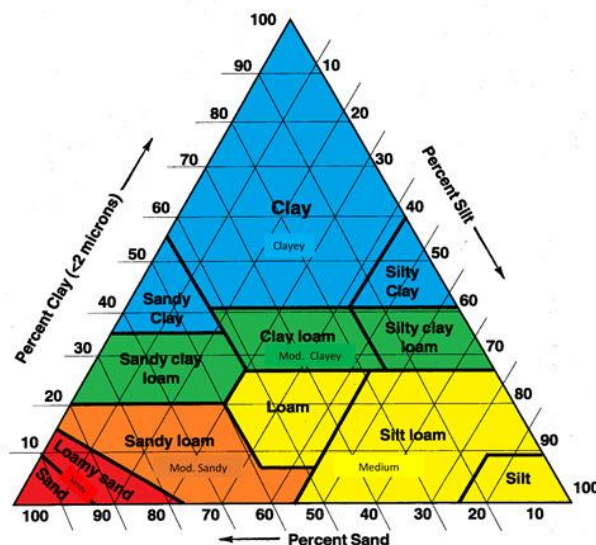
Numpy and Pandas are widely used Python packages for EDA. Numpy is helpful in dealing with arrays while Pandas deal with Dataframe and Series. Pandas library is helpful in cleaning, manipulating and analyzing the data.

- A histogram plot is drawn for soil moisture to identify the distribution.
- Correlation matrix is created between soil moisture and sand, silt and clay separately to identify the relationship between the variables.

Feature Selection and Engineering:

Since our focus is not on comparison of soil moisture between SMOS-ASCAT and AMSR, their values are taken mean and added as a new feature.

Since we don't deal with temporal analysis of data, we shall discard 'time' feature from the data frame. Based on United States Department of Agriculture (USDA), soil classification would be done on the basis of soil texture as follows:



Selection of ML Algorithm based on Output Data: Since our objective is to classify the soil on the basis of texture, classification algorithms are taken out of all other Machine learning algorithms. Random Forest classifier is chosen for this case because Random Forest provide a measure of feature importance, allowing us to identify which attribute (or feature) is more relevant for predicting the class.

Working of Random Forests: During training, the algorithm creates a large number of decision trees, from which it outputs the class that represents the mode of the categorization classes. A random selection of characteristics and a subset of the training data are used to build each decision tree in the random forest. This creates variation among the trees and increases the model's resilience against overfitting.

To produce these varied subsets, the random forest algorithm uses a method known as bagging (Bootstrap Aggregating). Each tree is constructed during the training phase by recursively dividing the data according to the characteristics. The algorithm chooses the best feature from the random subset at each split, maximizing either Gini impurity or information gain. The procedure keeps on until a predetermined end point is reached, like reaching a certain depth or having a certain number of samples in each leaf node.

Following training, each tree "votes" for a class, and the class with the greatest number of votes becomes the projected class for the input data. This allows the random forest to make predictions.

Model Training and Model Testing: Data is split into training data and testing data using scikit module.

Model Evaluation Performance: Random Forest's accuracy is evaluated by Accuracy constant, Contingency matrix and Accuracy Report that contains Precision, support, F1 score and recall

Model Comparison: Random Forest classifier's performance is compared with Support Vector Machines and Decision Trees Classifier.

Geovisualization: On Open Street map, the datapoints are overlaid and the classes of the datapoints are differentiated by colors.

CHAPTER 4

MODELING AND PROJECT OUTCOME

Importing the required modules:

Code snippet:

```
# importing the necessities
import pandas as pd
import geopandas as gpd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import folium
from folium.plugins import FloatImage
from shapely.geometry import Point
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

Data Overview:

Code Snippet – Shape and printing the first 5 rows:

```
# Load the data
df = pd.read_csv('/content/drive/MyDrive/Naan Mudhalvan/updated_data.csv')
# Data overview
print("Dataset Shape:", df.shape)
print("First 5 rows of the dataset:")
print(df.head())
```

Output:

Dataset Shape: (321584, 8)
First 5 rows of the dataset:

	time	latitude	longitude	clay_content	sand_content	silt_content	\
0	2013-01-03	54.875	9.125	5.0	86.0	9.0	
1	2013-01-05	54.875	9.125	5.0	86.0	9.0	
2	2013-01-07	54.875	9.125	5.0	86.0	9.0	
3	2013-01-08	54.875	9.125	5.0	86.0	9.0	
4	2013-01-09	54.875	9.125	5.0	86.0	9.0	

	sm_aux	sm_tgt
0	0.454120	0.53
1	0.437102	0.44
2	0.412978	0.42
3	0.378734	0.55
4	0.350740	0.46

Code Snippet – Descriptive Statistics:

```
# Descriptive statistics
print("Descriptive statistics of the dataset:")
print(df1.describe())
```

Output:

Descriptive statistics of the dataset:

	latitude	longitude	clay_content	sand_content	silt_content	\
count	669.000000	669.000000	669.000000	669.000000	669.000000	
mean	51.147795	10.294283	20.751868	40.844544	38.430493	
std	1.676589	2.128591	7.923361	21.421834	14.474426	
min	48.125000	6.125000	4.000000	11.000000	8.000000	
25%	49.875000	8.625000	13.000000	23.000000	24.000000	
50%	51.125000	10.125000	22.000000	32.000000	43.000000	
75%	52.625000	11.875000	27.000000	63.000000	50.000000	
max	54.875000	14.875000	46.000000	86.000000	67.000000	

	sm_aux	sm_tgt	sm_avg
count	669.000000	669.000000	669.000000
mean	0.304642	0.483528	0.394085
std	0.082946	0.074260	0.055032
min	0.007656	0.060000	0.164816
25%	0.274314	0.430000	0.366470
50%	0.316883	0.490000	0.395251
75%	0.344024	0.550000	0.431387
max	0.656782	0.600000	0.578391

Code Snippet – Visualizing the data points in the map:

```
#Data Overview - Visualizing the data points in a map
# Create a base map centered around Germany
m = folium.Map(location=[51.1657, 10.4515], zoom_start=5, tiles="OpenStreetMap")
# Get unique locations
unique_locations_ = df[['latitude', 'longitude']].drop_duplicates()
print("There are", len(unique_locations_), "unique locations in the dataset.")
```

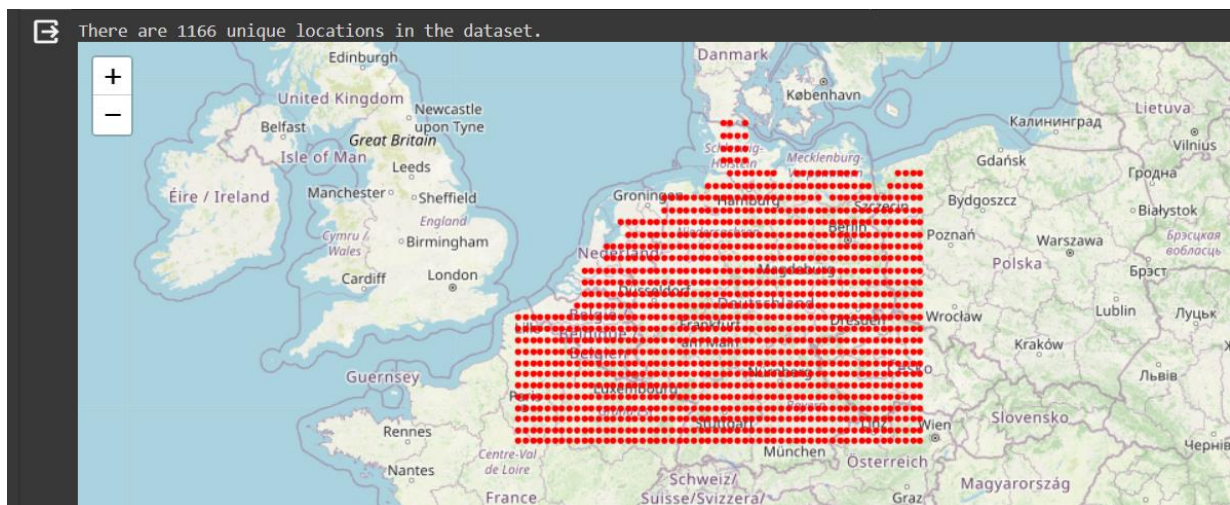
Add unique locations to the map

```
for idx, row in unique_locations_.iterrows():
    folium.CircleMarker(location=(row['latitude'], row['longitude']),
radius=1,color='red').add_to(m)
```

#display the map

```
m
```

Output:



Data Preprocessing:

Code Snippet - Removal of points that falls outside Germany:

Remove the points outside Germany border

Define a boundary for Germany

```
germany_boundary = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
germany_boundary = germany_boundary[germany_boundary.name == 'Germany']
```

Create a GeoDataFrame containing the points

```
geometry = []
```

Iterate over each row in the DataFrame

```
for index, row in unique_locations_.iterrows():
```

Extract longitude and latitude values from the current row

```
longitude = row['longitude']
```

```
latitude = row['latitude']

# Create a Point object using the longitude and latitude
point = Point(longitude, latitude)

# Append the Point object to the list
geometry.append(point)

points_gdf = gpd.GeoDataFrame(unique_locations_, geometry=geometry,
crs=germany_boundary.crs)

# Perform spatial join to filter points within Germany boundary
points_within_germany = gpd.sjoin(points_gdf, germany_boundary, how="inner",
op="within")

# Get the indices of points within Germany
indices_within_germany = points_within_germany.index

# Filter the original DataFrame based on the indices
df1 = df.loc[indices_within_germany]

# Display the filtered DataFrame
print("Filtered DataFrame with points only within Germany:")
print(df1.head())
```

Output:

```
Filtered DataFrame with points only within Germany:
   time    latitude  longitude  clay_content  sand_content  \
476  2013-06-01    54.875    9.875         18.0          52.0
478  2013-01-03    54.625    9.125         11.0          73.0
657  2013-01-03    54.625    9.375          6.0          80.0
901  2013-01-05    54.625    9.625         11.0          60.0
1088 2013-04-15    54.625    9.875         14.0          56.0

   silt_content  sm_aux  sm_tgt
476         30.0   0.056876   0.51
478         16.0   0.456073   0.53
657         14.0   0.451374   0.56
901         30.0   0.332698   0.52
1088        30.0   0.123108   0.55
```

Code Snippet - Overview of filtered dataset:

```
# Data overview of filtered dataset
print("Dataset Shape:", df1.shape)
print("First 5 rows of the dataset:")
print(df1.head())
```

Output:

```
Dataset Shape: (669, 8)
First 5 rows of the dataset:
   time  latitude  longitude  clay_content  sand_content  \
476  2013-06-01   54.875    9.875         18.0          52.0
478  2013-01-03   54.625    9.125         11.0          73.0
657  2013-01-03   54.625    9.375          6.0          80.0
901  2013-01-05   54.625    9.625         11.0          60.0
1088 2013-04-15   54.625    9.875         14.0          56.0

   silt_content  sm_aux  sm_tgt
476          30.0   0.056876   0.51
478          16.0   0.456073   0.53
657          14.0   0.451374   0.56
901          30.0   0.332698   0.52
1088         30.0   0.123108   0.55
```

Code Snippet - Handling of missing values and duplicates:

```
print("Missing values in each column:")
print(df1.isnull().sum())
# Handle missing values by either removing or imputing them
df1 = df1.dropna()
#Removal of duplicate records
df1.drop_duplicates()
```

Output:

```
Missing values in each column:
time          0
latitude      0
longitude     0
clay_content  0
sand_content  0
silt_content  0
sm_aux        0
sm_tgt        0
dtype: int64
```

Feature Engineering:

- 'time' feature is removed as we are not doing temporal analysis.
- Two satellites' soil moisture values are averaged.

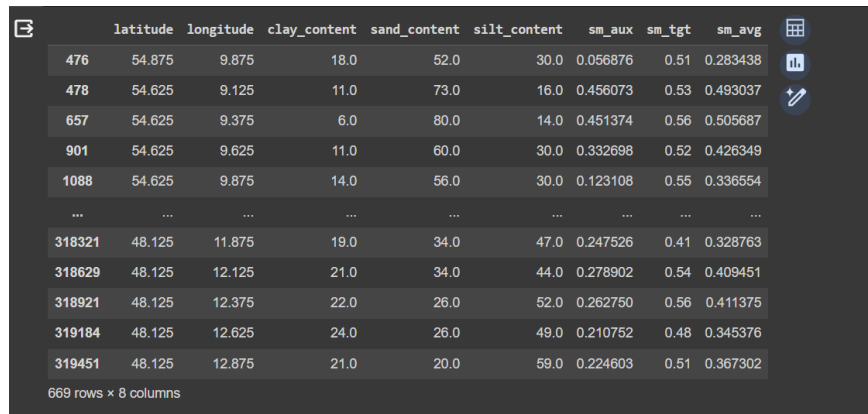
Code Snippet:

```
#Feature Engineering
# Remove the 'time' column from the DataFrame
df1.drop(columns=['time'], inplace=True)
```


#Averaging the satellite moisture values

```
df1['sm_avg'] = (df1['sm_aux'] + df1['sm_tgt']) / 2
df1
```

Output:



	latitude	longitude	clay_content	sand_content	silt_content	sm_aux	sm_tgt	sm_avg
476	54.875	9.875	18.0	52.0	30.0	0.056876	0.51	0.283438
478	54.625	9.125	11.0	73.0	16.0	0.456073	0.53	0.493037
657	54.625	9.375	6.0	80.0	14.0	0.451374	0.56	0.505687
901	54.625	9.625	11.0	60.0	30.0	0.332698	0.52	0.426349
1088	54.625	9.875	14.0	56.0	30.0	0.123108	0.55	0.336554
...
318321	48.125	11.875	19.0	34.0	47.0	0.247526	0.41	0.328763
318629	48.125	12.125	21.0	34.0	44.0	0.278902	0.54	0.409451
318921	48.125	12.375	22.0	26.0	52.0	0.262750	0.56	0.411375
319184	48.125	12.625	24.0	26.0	49.0	0.210752	0.48	0.345376
319451	48.125	12.875	21.0	20.0	59.0	0.224603	0.51	0.367302

669 rows x 8 columns

Outliers handling:

Code snippet:

```
# Detect outliers using z-scores
from scipy import stats
z_scores = np.abs(stats.zscore(df1))
threshold = 3
outliers = np.where(z_scores > threshold)

# Remove outliers from the DataFrame
df_no_outliers = df1[(z_scores < threshold).all(axis=1)]
print("Original DataFrame:")
print(df1)
print("\nDataFrame without outliers:")
print(df_no_outliers)
```


Output:

```
Original DataFrame:
  latitude longitude clay_content sand_content silt_content \
476    54.875    9.875      18.0        52.0        30.0
478    54.625    9.125      11.0        73.0        16.0
657    54.625    9.375       6.0        80.0        14.0
901    54.625    9.625      11.0        60.0        30.0
1088   54.625    9.875      14.0        56.0        30.0
...      ...      ...      ...      ...      ...
318321  48.125   11.875      19.0        34.0        47.0
318629  48.125   12.125      21.0        34.0        44.0
318921  48.125   12.375      22.0        26.0        52.0
319184  48.125   12.625      24.0        26.0        49.0
319451  48.125   12.875      21.0        20.0        59.0

  sm_aux sm_tgt sm_avg
476    0.056876  0.51  0.283438
478    0.456073  0.53  0.493037
657    0.451374  0.56  0.505687
901    0.332698  0.52  0.426349
1088   0.123108  0.55  0.336554
...      ...      ...      ...
318321  0.247526  0.41  0.328763
318629  0.278902  0.54  0.409451
318921  0.262750  0.56  0.411375
319184  0.210752  0.48  0.345376
319451  0.224603  0.51  0.367302

[669 rows x 8 columns]
```

```
DataFrame without outliers:
  latitude longitude clay_content sand_content silt_content \
476    54.875    9.875      18.0        52.0        30.0
478    54.625    9.125      11.0        73.0        16.0
657    54.625    9.375       6.0        80.0        14.0
901    54.625    9.625      11.0        60.0        30.0
1088   54.625    9.875      14.0        56.0        30.0
...      ...      ...      ...      ...      ...
318321  48.125   11.875      19.0        34.0        47.0
318629  48.125   12.125      21.0        34.0        44.0
318921  48.125   12.375      22.0        26.0        52.0
319184  48.125   12.625      24.0        26.0        49.0
319451  48.125   12.875      21.0        20.0        59.0

  sm_aux sm_tgt sm_avg
476    0.056876  0.51  0.283438
478    0.456073  0.53  0.493037
657    0.451374  0.56  0.505687
901    0.332698  0.52  0.426349
1088   0.123108  0.55  0.336554
...      ...      ...      ...
318321  0.247526  0.41  0.328763
318629  0.278902  0.54  0.409451
318921  0.262750  0.56  0.411375
319184  0.210752  0.48  0.345376
319451  0.224603  0.51  0.367302

[649 rows x 8 columns]
```

Data Normalization:

Code snippet:

#Data Normalization

```
from sklearn.preprocessing import MinMaxScaler
```

Define the columns you want to normalize

```
columns_to_normalize = ['clay_content', 'sand_content', 'silt_content', 'sm_aux', 'sm_tgt',
'sm_avg']
```

Initialize the MinMaxScaler

```
scaler = MinMaxScaler()
```

Normalize the selected columns

```
df_normalized = df_no_outliers.copy()
```

```
df_normalized[columns_to_normalize] =
```

```
scaler.fit_transform(df_normalized[columns_to_normalize])
```

```
print(df_normalized)
```

Output:

```

latitude longitude clay_content sand_content silt_content \
476 54.875 9.875 0.400000 0.546667 0.372881
478 54.625 9.125 0.200000 0.826667 0.135593
657 54.625 9.375 0.057143 0.920000 0.101695
901 54.625 9.625 0.200000 0.653333 0.372881
1088 54.625 9.875 0.285714 0.600000 0.372881
... ..
318321 48.125 11.875 0.428571 0.306667 0.661017
318629 48.125 12.125 0.485714 0.306667 0.610169
318921 48.125 12.375 0.514286 0.200000 0.745763
319184 48.125 12.625 0.571429 0.200000 0.694915
319451 48.125 12.875 0.485714 0.120000 0.864407

sm_aux sm_tgt sm_avg
476 0.000000 0.71875 0.175406
478 0.805553 0.78125 0.859142
657 0.796070 0.87500 0.900410
901 0.556590 0.75000 0.641599
1088 0.133652 0.84375 0.348677
... ..
318321 0.384719 0.40625 0.323262
318629 0.448034 0.81250 0.586476
318921 0.415439 0.87500 0.592752
319184 0.310510 0.62500 0.377455
319451 0.338462 0.71875 0.448979

[649 rows x 8 columns]
```

Exploratory Data Analysis:

Code Snippet – Correlation matrix:

Compute the correlation matrix

```
correlation_matrix = df_normalized.corr()
```

Plot the correlation matrix as a heatmap

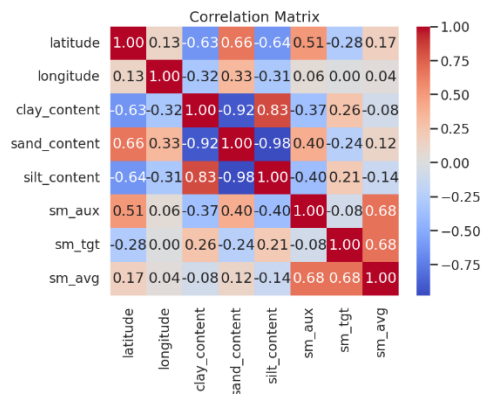
```
plt.figure(figsize=(8, 6))
```

```
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
```

```
plt.title('Correlation Matrix')
```

```
plt.show()
```

Output:

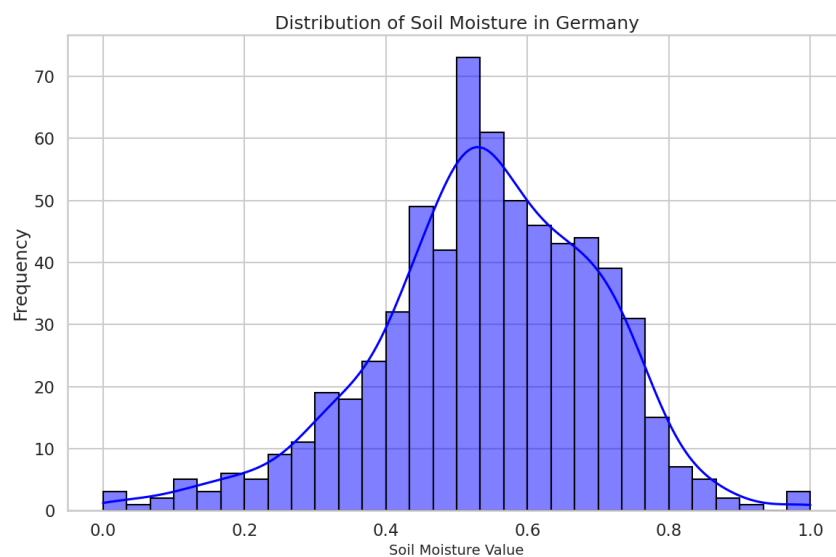


Code snippet – Data distribution using Histogram:

```
#Style setting
sns.set_style("whitegrid")
sns.set_context("talk")

# Create the distribution plot
plt.figure(figsize=(12, 8))
sns.histplot(df_normalized['sm_avg'], bins=30, kde=True, color='blue', edgecolor='black')
plt.title("Distribution of Soil Moisture in Germany")
plt.xlabel("Soil Moisture Value", fontsize=14)
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```

Output:



Inference:

Soil moisture values below 40% are present in a considerable amount of the land—more than 50%—highlighting possible areas of concern.

Code Snippet – Correlation analysis graph:

```
# Randomly sample 300 data points from the dataframe
sample_df = df_normalized.sample(n=300, random_state=42)

# Initialize a figure with three subplots side by side
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(20, 6))

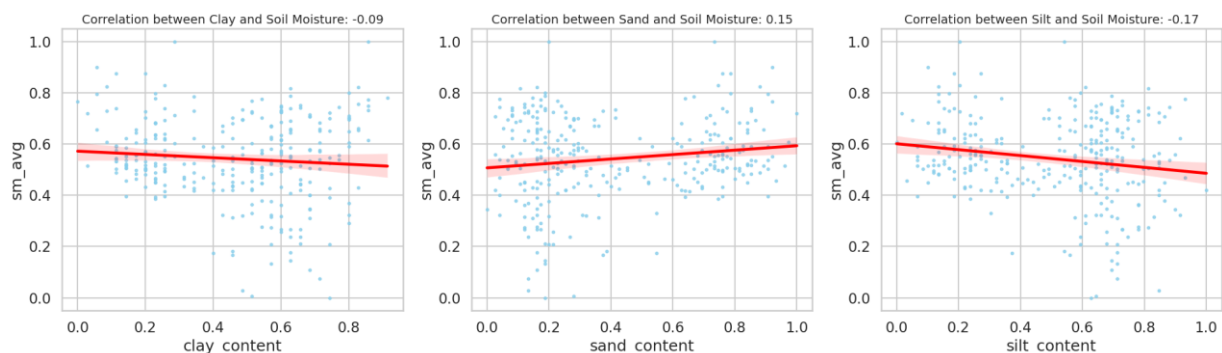
# List of soil components
components = ['clay_content', 'sand_content', 'silt_content']

# Plot scatter plots for each component
for i, component in enumerate(components):
    sns.regplot(x=component, y='sm_avg', data=sample_df, ax=axes[i], color='skyblue',
                scatter_kws={'s':10}, line_kws={'color':'red'})

    # Calculate correlation and annotate the plot with its value
    correlation = sample_df[component].corr(sample_df['sm_avg'])
    axes[i].set_title(f"Correlation between {component.split('_')[0].capitalize()} and Soil
    Moisture: {correlation:.2f}", fontsize=14)

# Adjust layout for better display
plt.tight_layout()
plt.show()
```

Output:



Inference:

- There are no clear correlations between soil moisture content and composition, according to the correlation plots.
- The data is inconsistent and lack a distinct directional trend, indicating that soil type may not be the only factor influencing soil moisture.
- The absence of a distinct trend may indicate the influence of artificial irrigation, given that the dataset originated in Europe, where automated irrigation systems are beneficial to many agricultural areas.

Feature Engineering:

Code snippet:

[The text in ***Bold + Italics*** should come as a single line but due to space constraints, it came as double line]

```
#Feature Selection and Engineering
#Define Labels based on USDA Soil Taxonomy
def classify_soil(row):
    if row['clay_content'] < 0.20 and row['sand_content'] < 0.28 and row['silt_content'] >= 0.52:
        return 'silt'
    elif row['clay_content'] < 0.20 and row['sand_content'] >= 0.52:
        return 'sand'
    elif row['clay_content'] >= 0.20 and row['sand_content'] < 0.45 and row['silt_content'] >= 0.40:
        return 'sandy clay'
    elif row['clay_content'] < 0.7 and row['sand_content'] < 0.15 and row['silt_content'] < 0.52:
        return 'sandy loam'
    elif row['clay_content'] >= 0.7 and row['sand_content'] < 0.15 and row['silt_content'] < 0.52:
        return 'loamy sand'
    elif row['clay_content'] >= 0.7 and row['clay_content'] < 0.20 and row['sand_content'] >= 0.15
and row['sand_content'] < 0.28 and row['silt_content'] >= 0.52:
        return 'silt loam'
    elif row['clay_content'] < 0.7 and row['sand_content'] >= 0.15 and row['sand_content'] < 0.28
and row['silt_content'] < 0.52:
        return 'sandy clay loam'
```

```

elif row['clay_content'] >= 0.7 and row['clay_content'] < 0.20 and row['sand_content'] < 0.15
and row['silt_content'] >= 0.52:
    return 'silty clay loam'

elif row['clay_content'] >= 0.7 and row['clay_content'] < 0.20 and row['sand_content'] >= 0.15
and row['sand_content'] < 0.28 and row['silt_content'] < 0.52:
    return 'sandy clay loam'

elif row['clay_content'] >= 0.20 and row['sand_content'] < 0.15 and row['silt_content'] >= 0.52:
    return 'clay loam'

elif row['clay_content'] >= 0.20 and row['sand_content'] >= 0.15 and row['sand_content'] <
0.28 and row['silt_content'] >= 0.52:
    return 'silty clay loam'

elif row['clay_content'] >= 0.20 and row['sand_content'] >= 0.28 and row['silt_content'] >= 0.52:
    return 'silty clay'

elif row['clay_content'] >= 0.20 and row['sand_content'] < 0.52 and row['silt_content'] < 0.52:
    return 'clay'

# Apply classification
df_normalized['soil_texture'] = df_normalized.apply(classify_soil, axis=1)
print(df_normalized)

```

Output:

```

latitude longitude clay_content sand_content silt_content \
476 54.875 9.875 0.400000 0.546667 0.372881
478 54.625 9.125 0.200000 0.826667 0.135593
657 54.625 9.375 0.057143 0.920000 0.101695
901 54.625 9.625 0.200000 0.653333 0.372881
1088 54.625 9.875 0.285714 0.600000 0.372881
... ..
318321 48.125 11.875 0.428571 0.306667 0.661017
318629 48.125 12.125 0.485714 0.306667 0.610169
318921 48.125 12.375 0.514286 0.200000 0.745763
319184 48.125 12.625 0.571429 0.200000 0.694915
319451 48.125 12.875 0.485714 0.120000 0.864407

sm_aux sm_tgt sm_avg soil_texture
476 0.000000 0.71875 0.175406 None
478 0.805553 0.78125 0.859142 None
657 0.796070 0.87500 0.900410 sand
901 0.556590 0.75000 0.641599 None
1088 0.133652 0.84375 0.348677 None
... ..
318321 0.384719 0.40625 0.323262 sandy clay
318629 0.448034 0.81250 0.586476 sandy clay
318921 0.415439 0.87500 0.592752 sandy clay
319184 0.310510 0.62500 0.377455 sandy clay
319451 0.338462 0.71875 0.448979 sandy clay

[649 rows x 9 columns]

```

ML Model:

1. Feature Selection:

Code Snippet:

```
data = df_normalized.copy()
```

```
data = data.dropna()

# Select features for training from the original dataframe
selected_features = ['silt_content', 'sand_content', 'clay_content', 'sm_avg']
X = data[selected_features] # Features (all columns except the target)
y = data['soil_texture'] # Target column
```

2. Splitting of Training Data and Testing Data:

Code Snippet:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

3. Feature Encoding:

Code Snippet:

```
from sklearn.preprocessing import LabelEncoder

# Initialize the label encoder
label_encoder = LabelEncoder()

# Encode the target variable
y_train_encoded = label_encoder.fit_transform(y_train)
```

4. Choosing training the model and decoding the output:

Code Snippet:

```
# Initialize and train the RandomForestClassifier
forest = RandomForestClassifier(n_estimators=50, random_state=0)
forest.fit(X_train, y_train_encoded)

# Predict on the test set
y_pred_encoded = forest.predict(X_test)

# Decode the predicted labels
y_pred = label_encoder.inverse_transform(y_pred_encoded)
```

5. Evaluation of model:

Code Snippet:

#Evaluation of Model

Print accuracy on training and testing subsets

```
print('Accuracy on the training subset: {:.3f}'.format(forest.score(X_train,
y_train_encoded)))
print('Accuracy on the testing subset: {:.3f}'.format(forest.score(X_test,
y_pred_encoded)))
```

Confusion matrix

```
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

Classification report

```
class_report = classification_report(y_test, y_pred)
print('Classification Report:')
print(class_report)
```

Output:

```
➡ Accuracy on the training subset: 1.000
Accuracy on the testing subset: 1.000
Confusion Matrix:
[[ 2  0  0]
 [ 0 25  0]
 [ 0  0 128]]
Classification Report:
              precision    recall  f1-score   support

      clay         1.00      1.00      1.00         2
      sand         1.00      1.00      1.00        25
    sandy clay         1.00      1.00      1.00       128

   accuracy              1.00              1.00      155
  macro avg              1.00              1.00      155
weighted avg              1.00              1.00      155
```

Comparison of Random Forest Classifier with SVM and Decision Trees:

Code Snippet:

#Comparison with SVM and Decision Trees

```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

# Train SVM model
svm_model = SVC(kernel='linear', random_state=0)
svm_model.fit(X_train, y_train)
```


Train Decision Trees model

```
dt_model = DecisionTreeClassifier(random_state=0)
dt_model.fit(X_train, y_train)
```

Predictions

```
svm_pred = svm_model.predict(X_test)
dt_pred = dt_model.predict(X_test)
```

Evaluate SVM model

```
svm_accuracy = accuracy_score(y_test, svm_pred)
svm_conf_matrix = confusion_matrix(y_test, svm_pred)
svm_report = classification_report(y_test, svm_pred)
```

Evaluate Decision Trees model

```
dt_accuracy = accuracy_score(y_test, dt_pred)
dt_conf_matrix = confusion_matrix(y_test, dt_pred)
dt_report = classification_report(y_test, dt_pred)
print("Report:\n", svm_report)
```

Print performance metrics

```
print("SVM Performance Metrics:")
print("Accuracy:", svm_accuracy)
print("Precision:", svm_precision)
print("Recall:", svm_recall)
print("F1-score:", svm_f1)
print("Confusion Matrix:\n", svm_conf_matrix)
print("\nDecision Trees Performance Metrics:")
print("Accuracy:", dt_accuracy)
print("Precision:", dt_precision)
print("Recall:", dt_recall)
print("F1-score:", dt_f1)
print("Confusion Matrix:\n", dt_conf_matrix)
print("Report:\n", dt_report)
```

Output:

SVM Performance Metrics:
 Accuracy: 0.9870967741935484
 Confusion Matrix:
 [[0 0 2]
 [0 25 0]
 [0 0 128]]
 Report:

	precision	recall	f1-score	support
clay	0.00	0.00	0.00	2
sand	1.00	1.00	1.00	25
sandy clay	0.98	1.00	0.99	128
accuracy			0.99	155
macro avg	0.66	0.67	0.66	155
weighted avg	0.97	0.99	0.98	155

Decision Trees Performance Metrics:
 Accuracy: 0.9935483870967742
 Confusion Matrix:
 [[1 0 0 1]
 [0 25 0 0]
 [0 0 128 0]
 [0 0 0 0]]
 Report:

	precision	recall	f1-score	support
clay	1.00	0.50	0.67	2
sand	1.00	1.00	1.00	25
sandy clay	1.00	1.00	1.00	128
silty clay	0.00	0.00	0.00	0
accuracy			0.99	155
macro avg	0.75	0.62	0.67	155
weighted avg	1.00	0.99	1.00	155

Geovisualization:

Code Snippet:

```
# Define colors for each class
class_colors = {
    'sand': 'red',
    'loamy sand': 'pink',
    'sandy loam': 'orange',
    'silt loam': 'lightgreen',
    'sandy clay loam': 'peach',
    'silty clay loam': 'yellow',
    'silty clay': 'blue',
    'clay loam': 'violet',
    'sandy clay': 'cyan',
    'clay': 'darkgreen',
    'silt': 'darkblue'}

# Get unique classes from the data
unique_classes = data['soil_texture'].unique()

# Create a map centered at the mean of latitude and longitude
m = folium.Map(location=[data['latitude'].mean(), data['longitude'].mean()], zoom_start=10)

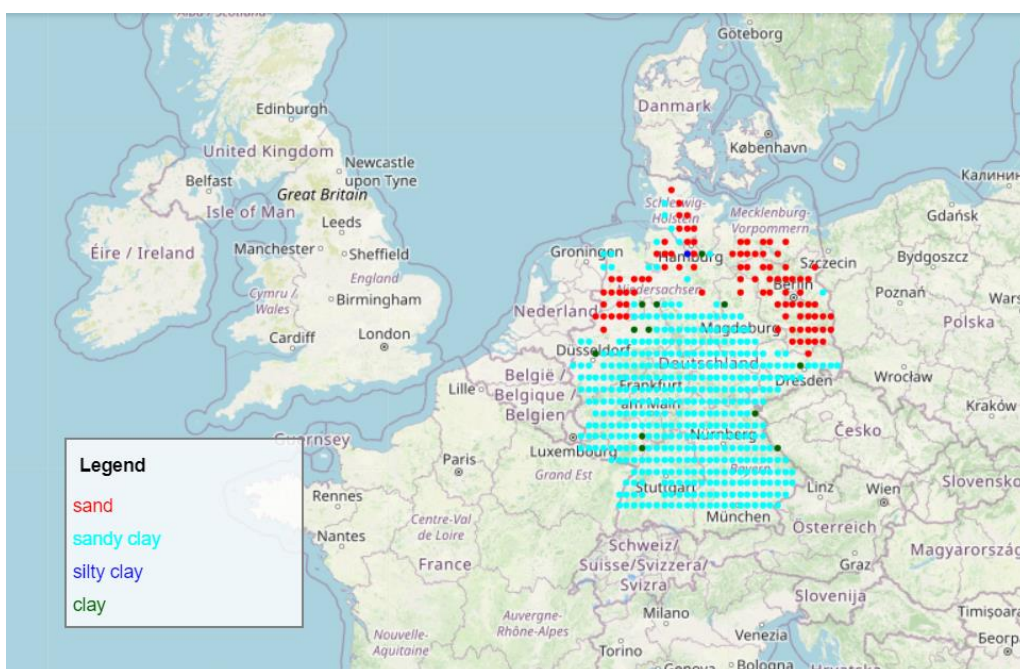
# Add markers for each data point with color corresponding to its class
for index, row in data.iterrows():
    class_color = class_colors.get(row['soil_texture'], 'black') # Default to black if class not
    found
```

```

folium.CircleMarker(location=[row['latitude'], row['longitude']], color=class_color, radius
=1, fill_color=class_color).add_to(m)
# Create legend only for existing points on the map - HTML
legend_html = """
<div style="position: fixed;
    bottom: 50px; left: 50px; width: 180px; height: auto;
    border: 2px solid grey; z-index: 9999; font-size: 14px;
    background-color: rgba(255, 255, 255, 0.8);
">
<p style="margin: 10px; color: black;"><strong>Legend</strong></p>
"""
for cls in unique_classes:
    class_color = class_colors.get(cls, 'black') # Default to black if class not found
    legend_html += f'<p style="margin: 5px; color: {class_color};">{cls}</p>'
legend_html += """
</div>
"""
m.get_root().html.add_child(folium.Element(legend_html))
# Display the map
m

```

Output:



CONCLUSION

Random Forest Classifier classified the soil on the basis of soil composition like clay, sand and silt with 100% accuracy which could not be brought by Support Vector Machine (98.7%) and Decision Tree classifier (99.3%). It has also been inferred that the datapoint's soil composition is not clearly dependent on soil moisture indicating that the soil could have been benefited by irrigation techniques. This ML classifier is helpful in assessing the unpredictable conditions and determine the importance of each feature in determining the output. Open Street Maps has been considered a powerful tool in visualizing the points and was very helpful in filtering the inner boundary points. The classification result has been displayed in Open Street Maps and HTML file has been used for displaying the legend. It helps us to identify that Germany is dominated by sandy clay soil.

FUTURE SCOPE

Using interpolation techniques like Kriging, soil type of whole Germany could be estimated and mapped visually. Temporal analysis of soil moisture can be done using Long Short Term Memory Neural Networks. And comparison of soil moistures obtained from two satellites can be compared and assessed. Outliers can be researched and the reason for occurrence can be found out from them. Also, training sites can be collected from the field and implemented in the algorithm to enhance the accuracy of the model.

REFERENCES

1) Daniel R. Gambill, Wade A. Wall, Andrew J. Fulton, Heidi R. Howard, 'Predicting USCS soil classification from soil property variables using Random Forest', Journal of Terramechanics, Volume 65, 2016, Pages 85-92, ISSN 0022-4898.

LINKS: (All the files are in this repository – [Click here](#))

- 1) PPT Link: [Powerpoint Presentation](#), [Input dataset in map](#), [Mapped Output](#)
- 2) Full code: [Click Here](#)
- 3) Video Explanation: [Link 1](#) (or) [Link 2](#)