Arka Shine

Innovation in agriculture

INTERNSHIP REPORT

BUDDULA VANSHIKA

1BM22AI034

DEPARTMENT OF MACHINE LEARNING

B.M.S COLLEGE OF ENGINEERING

(An Autonomous Institution Affiliated to VTU, Belagavi)

Post Box No.: 1908, Bull Temple Road, Bengaluru – 560 019

Arka Shine

Innovation in agriculture

DECLARATION

I, Buddula Vanshika (1BM22AI034) students of 5th Semester, B.E, Department of Artificial Intelligence and Machine Learning Engineering, BMS College of Engineering, Bangalore, hereby declare that, this internship has been carried out during the academic semester August - November 2024. I also declare that to the best of my knowledge and belief, the report is not from part of any other report by any other students.

Signature of the Candidate

BUDDULA VANSHIKA (1BM22AI034)

Table of Contents

Chapter 1: Introduction

Agriculture remains a cornerstone of global food security, economic development, and environmental sustainability. Among the many factors influencing agricultural productivity, soil quality stands out as a critical determinant of crop yield, resilience, and long-term land viability. Traditional soil testing methods, although scientifically accurate—are often expensive, time-consuming, and require specialized equipment and expertise, limiting their widespread adoption, especially among smallholder farmers in resource-constrained regions.

This project proposes an intelligent, data-driven approach to soil health assessment through machine learning (ML) and spectral analysis, offering a scalable, fast, and cost-effective alternative to conventional soil testing techniques. Leveraging hyperspectral and multispectral data, the model predicts key soil nutrient levels, including pH, organic carbon, nitrogen, phosphorus, potassium, and trace micronutrients like zinc, iron, manganese, and copper. The use of spectral data ensures non-invasive analysis and rapid processing, ideal for real-time decision-making.

The project involves the creation of a multi-output regression model trained on a rich dataset with diverse agro-climatic attributes. To enhance accuracy and interpretability, advanced feature selection techniques such as K Nearest Neighbors(KNN), Cars Algorithm and Mutual Information are employed to identify the most impactful spectral bands and soil parameters. Multiple ensemble learning algorithms, including Random Forest, Gaussian Process Regression (GPR), K Nearest Neighbors(KNN), and Partial Least Square Regression (PLSR), are benchmarked to determine the optimal predictive framework. Model performance is rigorously validated using metrics such as R^2 score, RMSE, and MAE, alongside cross-validation for generalization.

To ensure practical usability, the system includes an interactive and user-friendly web interface or mobile application, allowing farmers, agronomists, and policymakers to input spectral data (via portable sensors or satellite imaging) and instantly receive soil

nutrient predictions along with actionable recommendations for fertilizer usage, crop rotation strategies, and soil treatment plans.

Visualizations such as feature importance plots and help users understand how input features influence predictions, fostering trust and facilitating adoption in real-world agricultural workflows.

By aligning with the goals of precision agriculture and climate-smart farming, this solution enhances productivity, reduces excessive fertilizer application, and promotes environmental conservation. The broader vision is to empower farmers especially in underserved and developing regions with affordable, data-driven tools for sustainable land management and food security.

Soil spectroscopy, particularly in the visible and near-infrared (vis-NIR) range, has emerged as a transformative tool in modern agriculture, offering a rapid and non-invasive means of analyzing soil composition. By measuring the reflectance of electromagnetic radiation across various wavelengths, this technique enables the identification of key soil properties that influence fertility and crop productivity. The spectral regions commonly used include the visible range (400–780 nm), near-infrared (780–2500 nm), and short-wave infrared (SWIR) (1000–2500 nm). Each region provides specific information: visible light is absorbed mainly due to electronic transitions influenced by pigments and mineral content. Machine Learning (ML) techniques play a crucial role in preprocessing, analyzing, and modeling soil spectral data. ML algorithms excel at handling large datasets and identifying non-linear relationships between spectral features and soil attributes. Moreover, the ability to conduct non-invasive analysis supports sustainable farming by reducing the need for chemical testing and minimizing environmental disruption.

As we continue to refine spectral modeling and feature selection techniques, the potential for real-time, accurate, and affordable soil analysis becomes increasingly attainable marking a significant step forward in the future of smart farming.

Problem Statement

Modern agriculture faces the critical challenge of ensuring optimal soil health to maximize crop yield and sustainability. One of the most important factors influencing soil productivity is the accurate assessment of nutrient concentrations such as nitrogen, phosphorus, potassium, and micronutrients like zinc, copper, and iron. Traditionally, analyzing these nutrient levels involves labor-intensive, time-consuming laboratory procedures that may delay actionable decisions for farmers. In response to this challenge, there is a growing need for a rapid, cost-effective, and scalable method to determine soil nutrient concentrations. Hyperspectral and spectroscopic analysis of soil samples presents a promising non-destructive alternative, as different nutrients exhibit unique reflectance patterns at specific wavelengths of light. However, interpreting this spectral data to accurately predict nutrient values requires sophisticated modeling and technical expertise, often unavailable to end-users such as farmers and agronomists.

This project aims to develop an intelligent soil analysis system that leverages machine learning algorithms trained on spectral reflectance data to predict nutrient concentrations with high accuracy. The system utilizes various regression models including K-Nearest Neighbors, Random Forest, Gaussian Process Regressor, and Partial Least Squares Regression to learn the complex relationships between light wavelengths and soil nutrients. A Streamlit-based web interface is a user-friendly application that allows users to input spectral data and receive instant predictions of key soil parameters such as pH, electrical conductivity, organic carbon, and macro/micronutrients. By providing real-time analysis and interpretability, this system empowers farmers and soil experts to make informed decisions about fertilization, crop selection, and land management.

In summary, this project addresses the gap between advanced soil analysis techniques and their practical application in agriculture by offering an automated, real-time, and easy-to-use platform for precision farming.

Chapter 2: Dataset Summary

The dataset used in this study comprises 100 rows, each representing an individual soil sample collected under standardized conditions. These samples contain continuous numerical data for both the predictor variables (spectral measurements) and the target variables (soil nutrient concentrations), making it well-suited for regression-based machine learning models.

Each row includes 18 spectral features, labeled from A(410) to T(730), corresponding to reflectance values at specific wavelengths in the visible and near-infrared (vis-NIR) region of the electromagnetic spectrum. These bands capture critical information about the physical and chemical properties of soil, such as organic matter content, mineral composition, and moisture levels. The dataset also contains 13 target variables, representing key soil properties and macro/micronutrient levels. These include:

- pH (acidity/alkalinity)
- EC (Electrical Conductivity) in dS/m
- OC (Organic Carbon) in %
- Macronutrients: Phosphorus (P), Potassium (K) in kg/ha
- Secondary nutrients: Calcium (Ca), Magnesium (Mg) in meq/100g, Sulfur (S) in ppm
- Micronutrients: Iron (Fe), Manganese (Mn), Copper (Cu), Zinc (Zn), and Boron (B) in ppm

These parameters are critical for evaluating soil fertility and guiding decisions related to crop management and fertilization.

Preliminary exploratory data analysis revealed likely strong correlations between certain spectral bands and nutrient levels, suggesting the potential for predictive modeling. Given the presence of multiple dependent variables, the dataset is

particularly suitable for multi-output regression, where a model predicts several nutrient values simultaneously based on a common set of spectral inputs.

Data Interpolation to Augment the Dataset

To enhance the size and resolution of the dataset—and improve the generalization capability of machine learning models **data interpolation** techniques were applied. Interpolation is a mathematical approach used to estimate unknown values that fall between known data points. In data science, it plays a crucial role in:

- Filling in missing or incomplete data
- Increasing the resolution of datasets (data augmentation),
- Smoothing noisy observations
- Preparing datasets for continuous modeling tasks.

For this project, **linear interpolation** was selected as the method of choice due to its simplicity, computational efficiency, and effectiveness in generating smooth transitions between data points. Linear interpolation estimates intermediate values by drawing a straight line between two known data points and computing values along that line. This technique assumes a consistent rate of change between adjacent values, which is often reasonable for reflectance and nutrient variation in natural soil samples.

Applying linear interpolation allowed for the generation of new, intermediate synthetic data points that preserved the trends and integrity of the original dataset. This not only **increased the dataset size**, which is particularly valuable when training machine learning models, but also helped reduce overfitting by exposing the model to a finer and more continuous representation of the underlying data distribution.

By resampling the data and introducing more resolution through interpolation, the models could better learn the relationship between spectral bands and nutrient levels. It also helped in **smoothing out noise** and bridging any small gaps in the measurements, thus making the predictive modeling more robust and reliable.

Initially we had a 100 rows dataset. On implementing a data interpolation algorithm we increased the dataset to 1000 rows. The metrics turned out to be much better with respect to the new dataset. The model performance increased. All the models are trained on the new dataset.

Significance in Model Development

The interpolation-enhanced dataset formed a strong foundation for developing multi-target machine learning models aimed at predicting soil health metrics. This process, combined with feature selection and ensemble learning algorithms like Random Forest, Gradient Boosting, and AdaBoost, ensured that the models could generalize well, even when the original dataset was small. Ultimately, this preprocessing step supports the larger goal of the project: delivering a scalable, rapid, and accurate soil nutrient assessment tool that leverages spectroscopy and machine learning to empower sustainable agricultural practices.

	A(410)	B(435)	C(460)	D(485)	E(510)	F(535)	G(560)	H(585)	R(610)	I(645
0	1945.583073	992.445549	2051.158343	707.815855	822.497556	1371.591959	318.743582	315.808545	1305.441288	159.151127
1	1943.544121	992.629960	2055.462126	709.027790	824.405453	1376.891259	319.765813	317.053994	1306.199401	159.776543
2	1941.592882	993.025711	2059.605570	710.169829	826.305536	1382.204525	320.839546	318.173784	1307.039845	160.540736
3	1939.801584	993.313118	2064.002916	711.341535	828.096294	1387.523483	321.732475	319.413991	1307.831078	161.041621
4	1937.841025	993.582409	2068.295197	712.622823	829.978290	1392.976353	322.992158	320.462005	1308.672093	161.642165

S(680)	J(705)	U(760)	V(810)	W(860)	K(900)	L(940)	T(730)
329.435021	49.883583	75.742013	237.625746	305.198939	60.100183	36.558138	83.336164
329.346105	50.126325	75.726853	237.613553	305.017837	60.312984	36.546880	83.273791
329.137451	50.253398	75.642033	237.571633	304.623520	60.688095	36.753995	83.253864
329.162111	50.351358	75.505536	237.408290	304.469558	60.840350	36.787122	83.265827
328.961156	50.565332	75.626609	237.353305	304.143067	61.206304	36.976748	83.252022

Fig 2.1 represents the dataset showing the attributes. There are 18 spectral features.

	pH	EC (dS/m)	OC (%)	P (kg/ha)	K (kg/ha)	Ca (meq/100g)	Mg (meq/100g)
0	6.721300	0.078143	0.976578	26.111221	443.993002	6.181086	2.265456
1	6.764217	0.161826	1.056002	26.209108	442.239771	6.109986	2.295709
2	6.790044	0.125657	1.107562	26.119729	440.458318	6.163074	2.330365
3	6.791516	0.052417	1.064160	26.372220	438.631156	5.993081	2.449625
4	6.749717	0.173584	0.954701	26.432852	436.879476	6.055142	2.322788

S (ppm)	Fe (ppm)	Mn (ppm)	Cu (ppm)	Zn (ppm)	B (ppm)
11.311470	3.084245	14.170898	2.158530	0.774614	1.207778
11.344523	3.167145	13.962777	2.249895	0.787312	1.189985
11.433120	3.165903	13.966927	2.148322	0.862102	1.269603
11.579233	3.179318	13.839813	2.159101	0.903875	1.194604
11.619573	3.179172	13.821768	2.168172	0.842110	1.273132

Fig 2.2 represents the dataset showing the 13 target variables, representing key soil properties and macro/micronutrient levels from pH to B(ppm).

```
x.shape
```

```
(1000, 18)
```

```
y.shape
```

```
(1000, 13)
```

```
import numpy as np

# Number of new rows to add
num_new_rows = 900

# Function to generate new rows by adding small random perturbations
def augment_data(df, num_new_rows):
    new_rows = df.sample(n=num_new_rows, replace=True).apply(
        lambda x: x + np.random.normal(loc=0, scale=0.05 * x.std(), size=len(x))
    )
    return pd.concat([df, new_rows], ignore_index=True)

# Augment both datasets
augmented_soil = augment_data(df_soil, num_new_rows)
augmented_soil_target = augment_data(df_soil_target, num_new_rows)

# Save the augmented datasets
augmented_soil.to_csv('augmented_soil.csv', index=False)
augmented_soil_target.to_csv('augmented_soil_target.csv', index=False)

print("Augmentation completed and saved to 'augmented_soil.csv' and 'augmented_soil_target.csv'")
```

```
Augmentation completed and saved to 'augmented_soil.csv' and 'augmented_soil_target.csv'
```

Fig 2.3 represents the shapes of the dataset- attributes and the target values after data interpolation.

Chapter 3: Software Requirements

- Programming Language: Python
- Libraries and Frameworks:
    - Pandas, NumPy: Data manipulation and analysis
    - Scikit-learn: Machine learning algorithms and utilities
    - Matplotlib, Seaborn: Data visualization
    - Streamlit: User interface development
    - Pickle: Model serialization
- Development Environment:
    - Jupyter Notebook, VS Code

Chapter 4: Architecture

The pipeline ensures a smooth flow from raw data to a real-world usable application. The use of interpolation, feature engineering, and multiple model comparisons boosts accuracy and robustness. The Streamlit UI brings everything together, providing a user-friendly, deployable solution for precision agriculture.

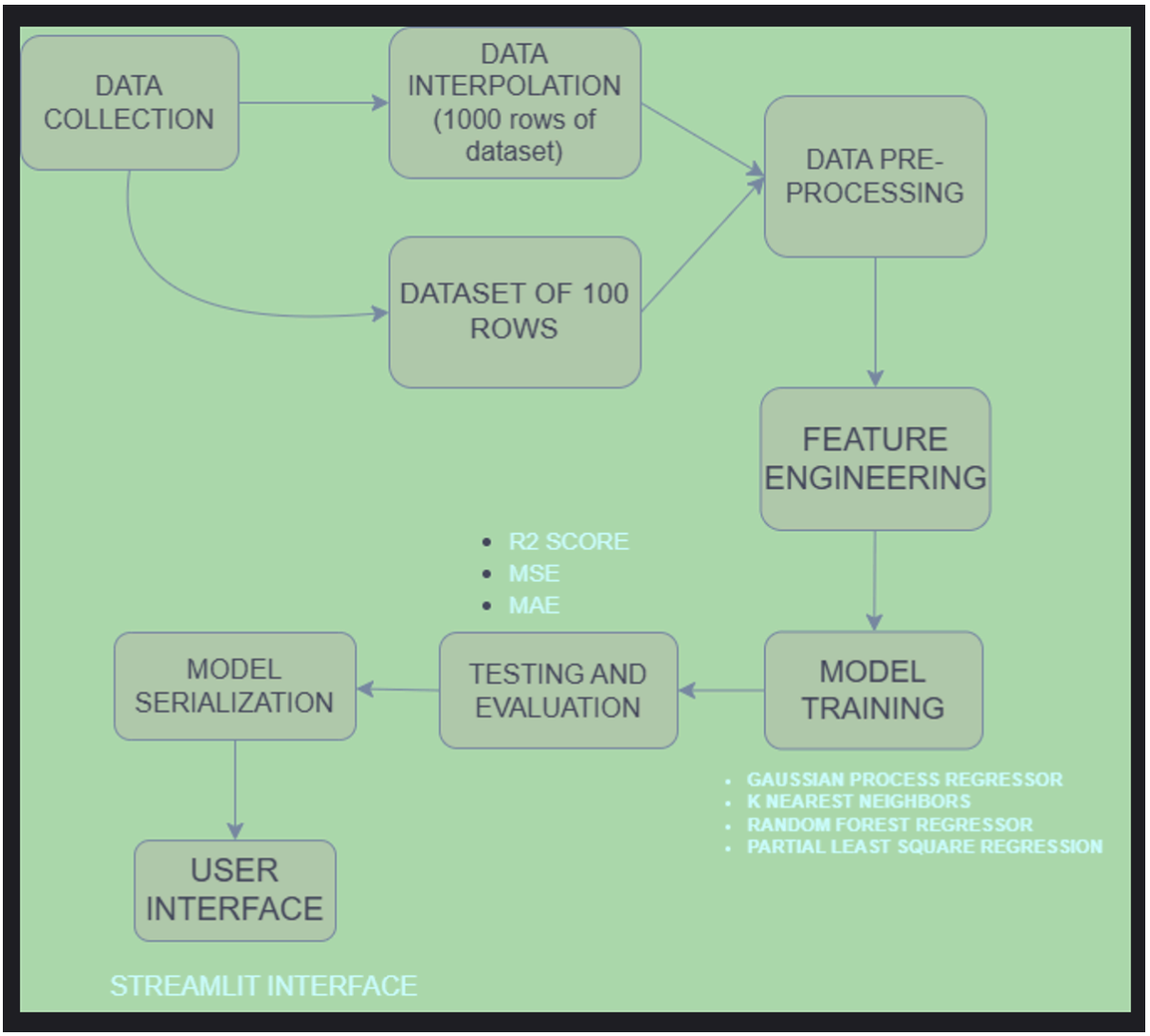


Fig 4.1 Architecture

1. **DATA COLLECTION**- This is the initial step where raw soil spectral data and nutrient values are gathered. Each row represents one soil sample with reflectance values across various wavelengths.

2. **DATA INTERPOLATION** (1000 rows of dataset)- Since the original dataset had only 100 rows, interpolation is used to synthetically expand it to 1000 rows. This helps in training more robust models by estimating intermediate values, simulating

more data points.

3. **DATASET OF 100 ROWS-** The original raw data (100 samples) is preserved. Both original and interpolated datasets can be used depending on the training need or testing strategy.

4. **DATA PRE-PROCESSING**

- Handling missing values
- Scaling the data using techniques like StandardScaler, MinMaxScaler, or RobustScaler.

5. **FEATURE ENGINEERING**

- Extracts or selects the most informative spectral bands.
- Could include techniques like permutation importance, or PCA
- Helps reduce dimensionality and improve model interpretability.

6. **MODEL TRAINING-** Several ML models are trained to learn from the spectral data:

- Gaussian Process Regressor (GPR)
- K Nearest Neighbors (KNN)
- Random Forest Regressor (RF)
- Partial Least Squares Regression (PLSR)
- Each model learns to predict multiple soil nutrient values (pH, K, Zn, etc.).

7. **TESTING AND EVALUATION**- Models are evaluated on unseen data using metrics:

- R^2 Score (goodness of fit)
- MSE (Mean Squared Error)
- MAE (Mean Absolute Error)

- This helps compare and choose the best-performing model.

8. **MODEL SERIALIZATION**

- The best-trained model is saved using tools like joblib or pickle.
- This serialized model can later be loaded without retraining.

9. **USER INTERFACE** (Streamlit Interface)- A Streamlit app is developed that:

    - Takes spectral input from the user.
    - Loads the serialized model.
    - Predicts nutrient levels instantly.
- Makes the model accessible to non-technical users like farmers or agronomists.

Chapter 5: Implementation and Results

- Data preprocessing is a crucial step in any data science or machine learning pipeline, involving the preparation and transformation of raw data into a clean, organized, and usable format. Real-world data is often incomplete, inconsistent, or noisy, which can negatively impact the performance and accuracy of machine learning models.
- Preprocessing addresses these issues by applying various techniques such as data cleaning, normalization, feature selection, and handling missing values. Data cleaning involves removing duplicates, correcting errors, and dealing with missing or null values through imputation or deletion.
- Normalization or standardization ensures that all features are on a similar scale, which is particularly important for algorithms sensitive to the magnitude of input values. Feature selection helps identify and retain only the most relevant variables, reducing dimensionality and improving model efficiency.
- Outlier detection and removal may also be performed to eliminate extreme values that could skew the model's learning process. In the context of this soil nutrient prediction project, data preprocessing includes cleaning the spectral dataset, selecting important wavelengths, and applying interpolation techniques to augment the dataset.
- These steps collectively enhance the quality and robustness of the data, allowing the machine learning model to learn more effectively and make accurate predictions about soil properties and nutrient levels.

1. Handling Missing Values

- Since all the predictors and target variables are numerical, missing values were replaced using the **mean** of the respective columns. This approach ensures minimal impact on the overall data distribution while retaining sufficient variability for analysis.

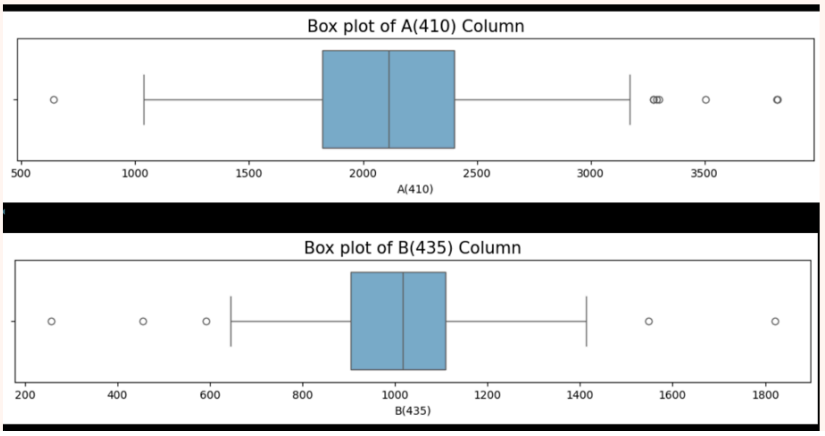
```
#treating the null values
#Since all are numerical data, we can use mean as the measure the replace all the misisng values
if df.isnull().values.any():
    # Calculate mean for numeric columns only
    numeric_cols = df.select_dtypes(include='number').columns
    df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())

# Change non-numeric values to numeric values
df = df.apply(pd.to_numeric, errors='coerce')
```

```
# Replace null values in columns with the column mean
df_filled_soil = df_soil.fillna(df_soil.mean())
df_filled_soil_target = df_soil_target.fillna(df_soil_target.mean())

# Save the updated datasets
df_filled_soil.to_csv('filled_soil.csv', index=False)
df_filled_soil_target.to_csv('filled_soil_target.csv', index=False)

print("Null values replaced with column mean and saved to 'filled_soil.csv' and 'filled_soil_target.csv'")
```

Fig 5.1 shows that since all are numerical data, we can use mean as the measure to replace all the missing values.

2. Outlier Detection and Treatment

- Outliers were identified using statistical measures such as interquartile ranges (IQR) and visualization tools like box plots.
- Fig 5.2 shows that, given the small number of outliers, no data points were removed. Instead, they were retained to preserve the integrity of the dataset and avoid potential bias introduced by manual removal.




```
Column 'A(410)' has 8 outliers.
Column 'B(435)' has 5 outliers.
Column 'C(460)' has 2 outliers.
Column 'D(485)' has 3 outliers.
Column 'E(510)' has 3 outliers.
Column 'F(535)' has 0 outliers.
Column 'G(560)' has 2 outliers.
Column 'H(585)' has 4 outliers.
Column 'R(610)' has 2 outliers.
Column 'I(645)' has 6 outliers.
Column 'S(680)' has 3 outliers.
Column 'J(705)' has 2 outliers.
Column 'U(760)' has 4 outliers.
Column 'V(810)' has 0 outliers.
Column 'W(860)' has 2 outliers.
Column 'K(900)' has 2 outliers.
Column 'L(940)' has 1 outliers.
Column 'T(730)' has 4 outliers.
Column 'pH' has 0 outliers.
Column 'EC  (dS/m)' has 7 outliers.
Column 'OC (%)' has 1 outliers.
```

Fig 5.2 represents the box plots showing outliers in the data.

3. Feature Scaling

- In many machine learning models (like KNN, SVM, Gradient Descent-based

models, etc.), features with larger ranges dominate the learning process, leading to biased or suboptimal performance. Scaling transforms features to a common scale, so each feature contributes equally.

- StandardScalar-To give features zero mean and unit variance so that they are centered and normalized. Subtract the mean and divide by standard deviation to scale the data. Transforms data to follow a standard normal distribution (bell-shaped curve). Helps models converge faster and perform better.
- MinMax Scalar-o scale all features between a fixed range like [0, 1]. Especially useful for models sensitive to magnitude like neural networks. Rescale based on min and max of the data. Compresses all features into the same scale while preserving relationships between values.
- RobustScalar- To handle data with outliers better by using the median and interquartile range (IQR) instead of mean and std. A feature with an outlier at 500 won't drastically affect scaling; values stay closer to the median-centered distribution.

```
# Standardisation
#importing libraries for data transformation
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
xts = scaler.fit_transform(X) #X train and test scaled
```

```
# Preprocess the data
from sklearn.preprocessing import RobustScaler
scaler_X = RobustScaler()
X_scaled = scaler_X.fit_transform(X_imputed)
```

```
from sklearn.preprocessing import MinMaxScaler

# Initialize Min-Max Scaler (for normalization)
scaler_norm = MinMaxScaler()

# Fit and transform the scaled data
X_scaled_norm = scaler_norm.fit_transform(X_scaled)
```

Fig 5.3 Data Scaling

4. Splitting the Dataset

- The dataset was divided into training (80%) and testing (20%) subsets using **train_test_split** with a fixed random state for reproducibility.
- This ensures that the model is trained on one portion of the data and evaluated on an unseen subset to assess generalization performance.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(xts, y, test_size=0.2, random_state=42)
```

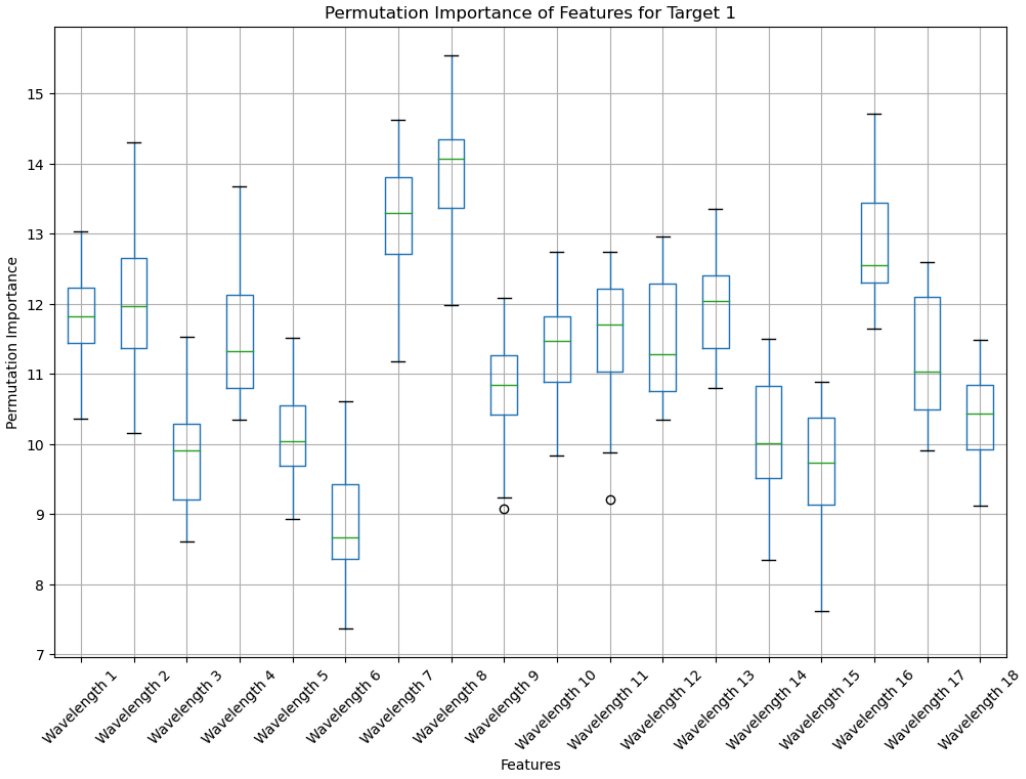
Fig 5.4 Splitting the dataset into train and test sets prior to training.

5. Feature Importance

- Permutation Importance is a model-agnostic technique used to determine the importance of each feature in a trained machine learning model by evaluating how the model's performance changes when each feature is randomly shuffled.
- If a feature is important, shuffling its values should break the relationship between that feature and the target variable. As a result, the model's performance will drop significantly. If the performance does not change much, it suggests that the feature is not crucial for the model's predictions.
- The plot_feature_importances function uses permutation_importance from scikit-learn to compute feature importances.
    - It does this over 30 repetitions to ensure stable results.
    - The output is visualized as a horizontal boxplot, where each box represents a feature (a wavelength).
    - The length of the box shows the variability in importance across the 30 repetitions.
    - Longer boxes with higher values mean the model's performance dropped more when that feature was shuffled hence it is an important feature.
    - Short or flat boxes near zero suggest the feature had little to no impact on the model's accuracy hence it is a less important feature. The same is represented in the figure

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.inspection import permutation_importance

# Function to compute and plot feature importances using permutation importance
def plot_feature_importances(model, X_test, y_test, feature_names):
    # Compute permutation importance
    perm_importance = permutation_importance(model, X_test, y_test, n_repeats=30, random_state=42)
    importances = perm_importance.importances.T  # Transpose to have shape (n_repeats, n_features)

    # Create a DataFrame for plotting
    importance_df = pd.DataFrame(importances, columns=feature_names)

    # Plot box plots
    plt.figure(figsize=(12, 8))
    sns.boxplot(data=importance_df, orient='h')
    plt.xlabel('Permutation Importance')
    plt.ylabel('Feature (Wavelength)')
    plt.title('Permutation Importances of Wavelength Features using GPR Model')
    plt.show()

# Assuming `X_test` is your test set features and `feature_names` is the list of feature names
feature_names = [f'Wavelength {i+1}' for i in range(X_test.shape[1])]

# Plot feature importances for knn model
plot_feature_importances(best_knn_model, X_test, y_test, feature_names)
```

Fig 5.5 represents the code snippet finding the permutation importance of each feature i.e the wavelengths. Permutation importance is found wrt to every trained model.



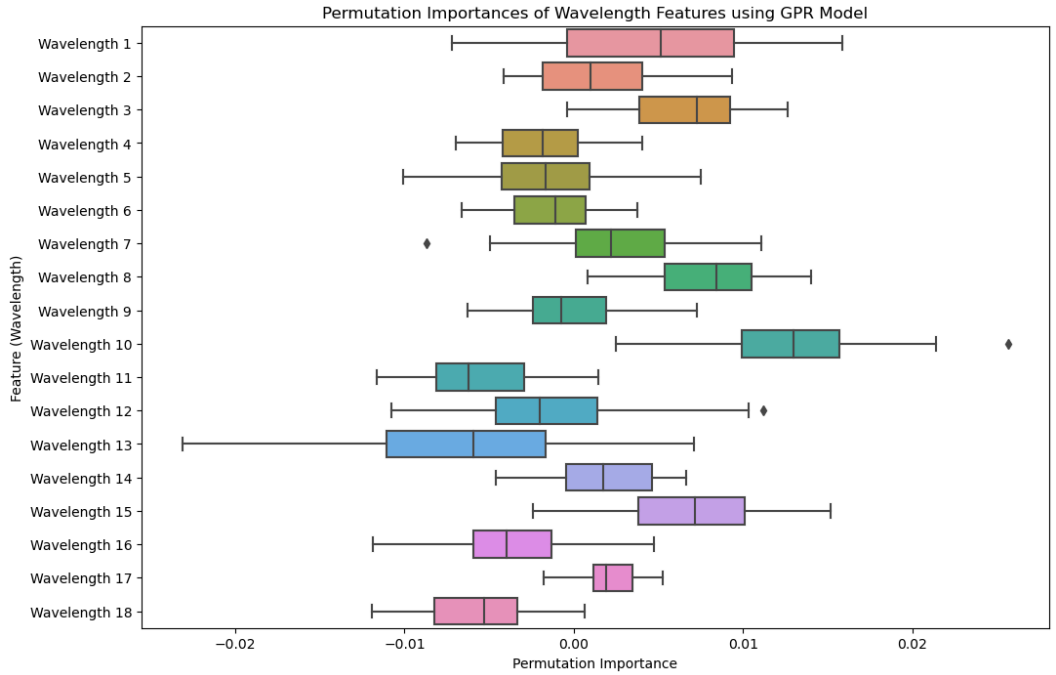


Fig 5.6 represents the code snippet of feature importance using permutation_importance. Box plots of the importance of all the features wrt each target and wrt to all targets considered together is plotted.

Models Implemented

1. K Nearest Neighbors Regressor- K-Nearest Neighbors (KNN) is typically associated with classification problems, but it can also be used for regression. In KNN regression, instead of predicting a class, the algorithm predicts a continuous value. The idea is simple: for a given input, find the kkk closest neighbors in the training set, and then average their values (or take a weighted average) to predict the output.

1. **Data Point Location**: For a given test input, the algorithm identifies the kkk closest data points in the training set using a distance metric (e.g., Euclidean distance).
2. **Neighbor Values**: It looks up the output (target) values of these kkk neighbors.
3. **Prediction**: The prediction for the test input is the average of these kkk neighbors' values.

Steps for KNN Regression:

1. **Choose kkk**: Select the number of nearest neighbors, kkk.
2. **Calculate Distance**: Compute the distance between the input query point and all other points in the training set.
3. **Identify Neighbors**: Select the kkk-nearest data points.
4. **Average Output**: The predicted value is the mean of the target values of these kkk-nearest points. In some cases, a weighted average can be used, where closer neighbors are given more weight.

Selecting the optimal K value: Selecting the best value of k in K-Nearest Neighbors (KNN) is crucial because it directly influences the model's performance and prediction accuracy. The k value determines the number of neighbors considered when making predictions. If k is too small (e.g., k=1), the model becomes highly sensitive to noise and outliers, as it only considers the closest neighbor, which might not represent the broader pattern in the data. This can lead to overfitting, where the model memorizes specific points but fails to generalize to new data. On the other hand, if k is too large (e.g k=50), the model may become too simplistic by averaging too many neighbors, including potentially irrelevant or distant ones. This can result in underfitting, where the model fails to capture important local variations. Therefore, selecting the optimal k value ensures a good balance between bias and variance, leading to more accurate and generalizable predictions.

Procedure:

- Given the characteristics of your dataset (1000 rows, 18 features, and hyperspectral data), KD-trees work well for datasets with moderate dimensions (like 18 features), and with 1000 rows, this method should provide a fast and efficient search.
- The task involves building a K-Nearest Neighbors (KNN) regression model to predict soil organic matter using a dataset with 1000 rows, 18 features, and 13 target values derived from hyperspectral data.
- The goal is to optimize the model by selecting the best value of k (number of

neighbors) and the appropriate distance metric through GridSearchCV.

- Hyperparameter tuning can be done after using a K Nearest Neighbors Regressor. In fact, it's a common practice to optimize the performance of the model.f Fine-tuning the hyperparameters can lead to better model performance, especially for complex datasets.
- By defining a parameter grid, which includes different values of k and metrics (such as Euclidean, Manhattan, and Cosine distances), a cross-validated search is performed to identify the parameter combination that minimizes the mean squared error (MSE).
- The result is a model that balances sensitivity to local data patterns without overfitting or underfitting, ensuring accurate predictions of soil organic matter.
- The plots of actual and predicted values of the nutrients are represented visually in the graph. Graphs for 5 individual samples are plotted.

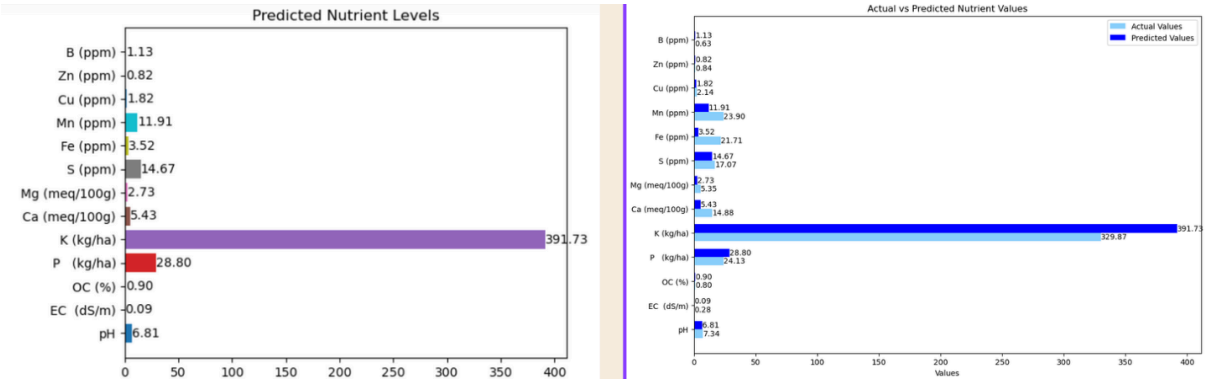
```
# Model training with hyperparameter tuning for KNeighborsRegressor
knn_param_grid = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

knn = KNeighborsRegressor()
knn_grid_search = GridSearchCV(estimator=knn, param_grid=knn_param_grid, cv=5, n_jobs=-1, scoring='neg_mean_squared_error')
knn_grid_search.fit(X_train, y_train)

best_knn_model = knn_grid_search.best_estimator_
```

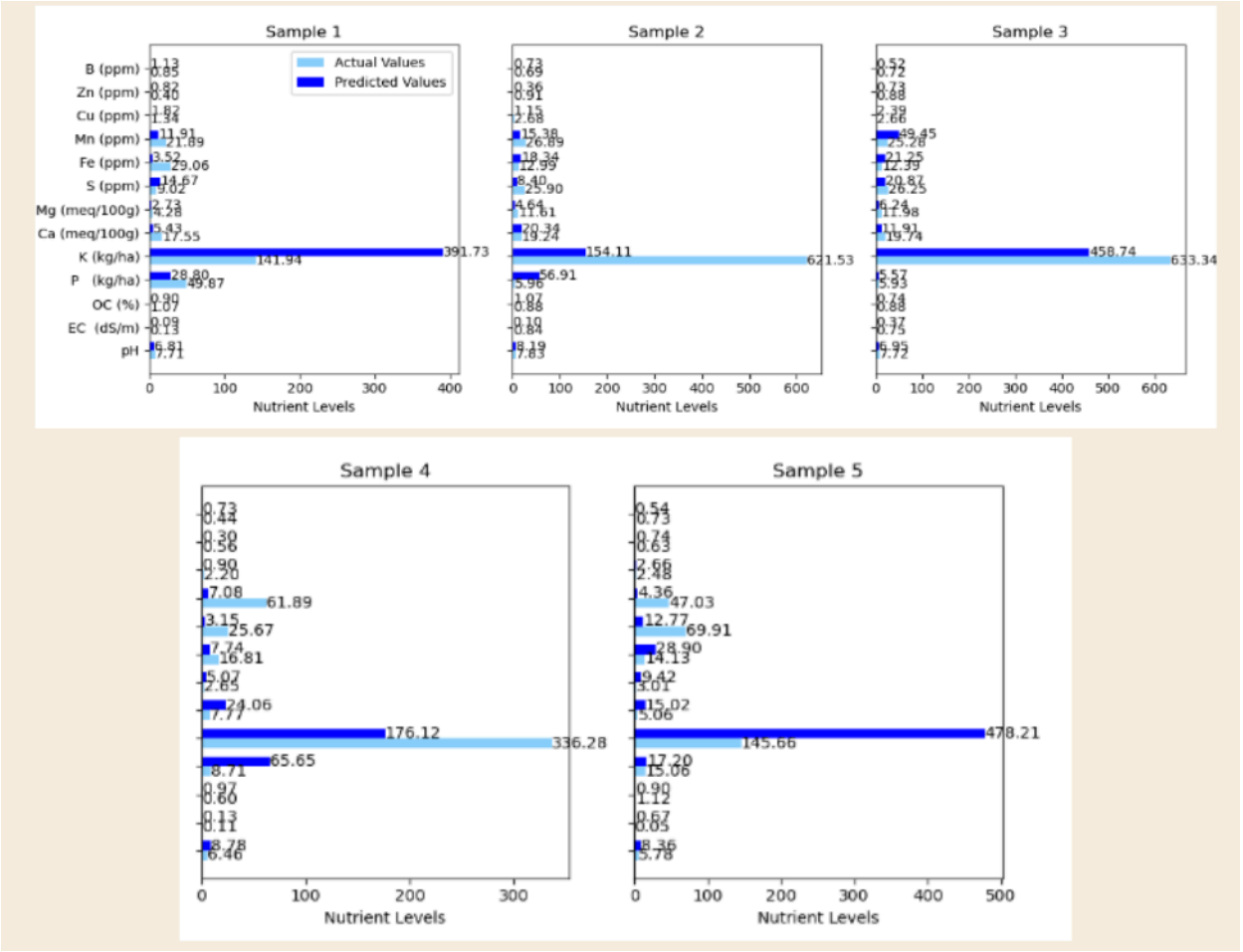
```
print("Best parameters found:", gridcv.best_params_)
print("Best score (negative MSE):", gridcv.best_score_)
```

```
Best parameters found: {'metric': 'euclidean', 'n_neighbors': 5}
Best score (negative MSE): -1.6780338456724682
```

```
KNN Model Evaluation:
train data
Mean Squared Error (MSE): 0.7390565621680552
R-squared (R2) Score: 0.9885681629971624
Mean Absolute Error (MAE): 0.19440475370052568
Root Mean Squared Error (RMSE): 0.859683989712531
test data
Mean Squared Error (MSE): 1.6700254134424766
R-squared (R2) Score: 0.9821528954571678
Mean Absolute Error (MAE): 0.26055225733948356
Root Mean Squared Error (RMSE): 1.2922946310507046
```

Fig 5.7 KNN model training evaluation





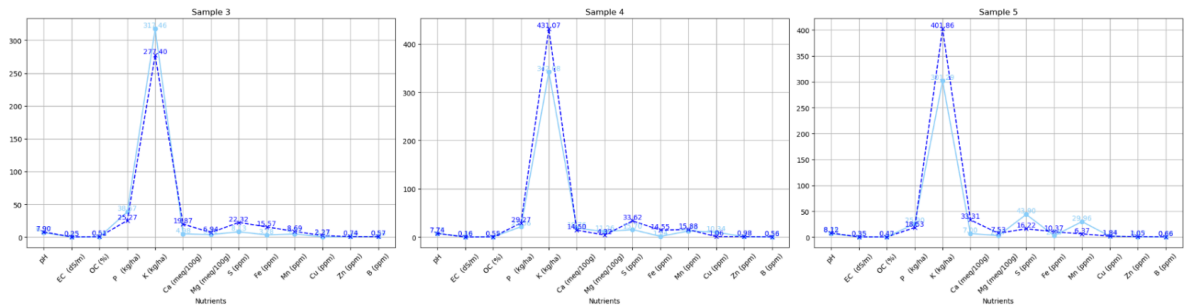


Fig 5.8 represents the actual vs predicted values of the nutrients for KNN model.

Fig 5.9 represents the feature importance for each target variable

2. Random Forest Regressor- When monitoring land surface information using remote sensing, RF is an algorithm that can quantitatively analyze ground-based information. RF, which integrates the advantages of a decision tree and bagging set, introduces randomly selected features into model training.

- Using RF to train the models not only effectively avoids overfitting, but also restrains the negative effects of noise. This improves the accuracy and stability

of model classification and prediction.

- In addition, RF is able to process and analyze high-dimensional information features, and statistically analyze the importance of multi-dimensional features, which is conducive to the comprehensive use of the hyperspectral characteristics of ground based objects. Since the number of decision trees has a great influence on a model, the number of decision trees was set as 100 in this study to guarantee the stability of each model.
- Random Forest is an ensemble learning method primarily used for regression and classification tasks. It builds multiple decision trees during training and outputs the average of the predictions from these trees (in the case of regression). It is a powerful and flexible model that works well in various scenarios, especially when the relationship between features and the target variable is complex and non-linear

Key Concepts of Random Forest Regressor

- Decision Trees: Each decision tree is constructed by recursively splitting the dataset into subsets based on feature values that maximize the variance reduction at each node (for regression). Decision trees, however, tend to overfit the data.
- Bagging (Bootstrap Aggregating): Random Forest improves the decision tree's performance through bagging, where multiple trees are built on random subsets of the data, and the final prediction is the average of all the trees' predictions. This reduces overfitting and increases the model's generalization ability.
- Feature Sampling: In addition to bagging, Random Forest introduces another layer of randomness by selecting a random subset of features at each split in the tree construction process. This prevents any single feature from dominating the model and enhances diversity among the trees.

Procedure:

- Data Preprocessing for Random Forest: For optimal performance, data preprocessing is critical. In this case, we will use MinMaxScaler to normalize the feature set. Normalization helps in scaling the feature values within a

specific range, typically [0, 1].

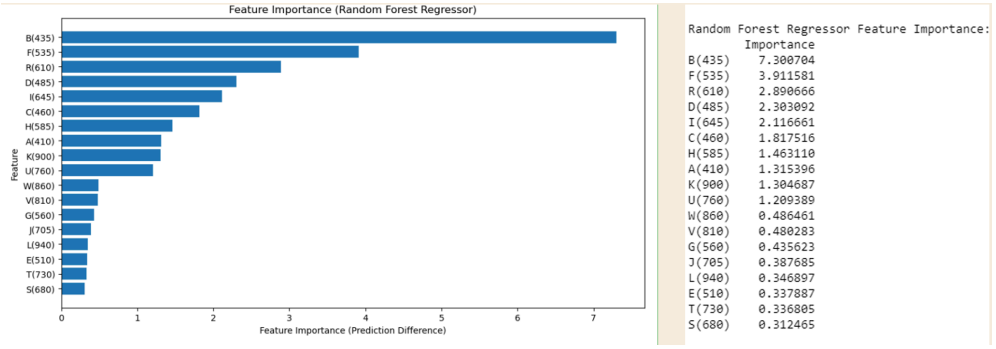
- Hyperparameter Tuning for Random Forest: The performance of Random Forest can be improved by tuning its hyperparameters. Some key hyperparameters include:
    - n_estimators: The number of trees in the forest. Increasing this typically improves performance but also increases computational cost.
    - max_depth: The maximum depth of each tree. Limiting depth helps prevent overfitting.
    - min_samples_split: The minimum number of samples required to split a node. Higher values can prevent trees from learning overly specific patterns in the data.
    - Using GridSearchCV helps find the best combination of these hyperparameters by evaluating different models with cross-validation.
- Feature Importance in Random Forest: One of the significant advantages of Random Forest is that it computes feature importance. It estimates how important each feature is for predicting the target variable by calculating how much each feature reduces the error across all the trees.
- Model Training and Prediction: After identifying the best hyperparameters through GridSearchCV, the model is retrained on the full training dataset. Once trained, the model can be used to make predictions on the test set.
- Evaluation Metrics: For regression tasks, the following evaluation metrics are commonly used:
    - Mean Squared Error (MSE): Measures the average of the squares of the errors or deviations from the actual target values.
    - R-squared (R^2): Measures the proportion of variance in the target variable that is predictable from the features. An R^2 value closer to 1 indicates better performance.
- Conclusion: Random Forest is a powerful and flexible model for regression tasks. By leveraging an ensemble of decision trees, it reduces overfitting, captures complex relationships, and provides useful insights into feature importance. Combining MinMaxScaler for data normalization, GridSearchCV

for hyperparameter tuning, and k-fold cross-validation for model evaluation ensures that the model is robust and generalizes well to new data. Finally, calculating key evaluation metrics like MSE and R-squared helps assess the model's performance on unseen data.

Training Process Overview (100 vs. 1000 Rows)

- Dataset Overview:
    - 100 Rows: Small dataset, challenging for generalization.
    - 1000 Rows: Larger dataset improves learning and reduces model variance.
- Evaluation Metrics and Results
    - 100 Rows: MSE (Mean Squared Error): Higher due to limited training data. R^2 Score: Lower, indicating reduced ability to explain variance in the target.
    - 1000 Rows: MSE: Significantly lower, reflecting improved predictions. R^2 Score: Higher, showing better variance explanation.
- Trade-Offs:
    - Larger datasets require more computational power and time for training. However, they enhance model generalization, reducing overfitting and increasing reliability.
- Observations and Conclusions:Increasing dataset size substantially improved performance metrics. Feature selection helped focus on critical variables, streamlining the model.
- Trade-Off Analysis:
    - Small Dataset (100 Rows): Quick to train but prone to overfitting and unreliable predictions.
    - Larger Dataset (1000 Rows): Requires more resources but yields accurate and robust models.
- Conclusion:
    - Training RFR on a well-preprocessed and sufficiently large dataset leads to a balanced and high-performing model.

- The combination of robust preprocessing, feature selection, and larger datasets is key to effective model training.




```
Random Forest Regressor Feature Importance:
      Importance
B(435)    7.300704
F(535)    3.911581
R(610)    2.890666
D(485)    2.303092
I(645)    2.116661
C(460)    1.817516
H(585)    1.463110
A(410)    1.315396
K(900)    1.304687
U(760)    1.209389
W(860)    0.486461
V(810)    0.480283
G(560)    0.435623
J(705)    0.387685
L(940)    0.346897
E(510)    0.337887
T(730)    0.336805
S(680)    0.312465
```

```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_imputed, test_size=0.2, random_state=42)

# Initialize the Random Forest Regressor
rf_regressor = RandomForestRegressor(random_state=42)

# Define the parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [900, 800, 400],
    'max_depth': [10, 20, 9],
    'min_samples_split': [70, 50, 90, 30]
}

# Define the number of folds for cross-validation
k_fold = KFold(n_splits=5, shuffle=True, random_state=42)

# Initialize GridSearchCV with Random Forest Regressor and cross-validation
grid_search = GridSearchCV(estimator=rf_regressor, param_grid=param_grid, cv=k_fold, scoring='r2')

# Perform grid search to find the best model
grid_search.fit(X_train, y_train)

# Print the best parameters and best score from GridSearchCV
print("Best Parameters found:")
print(grid_search.best_params_)
```

Fig 5.10 Hyperparameter tuning for RFRegressor

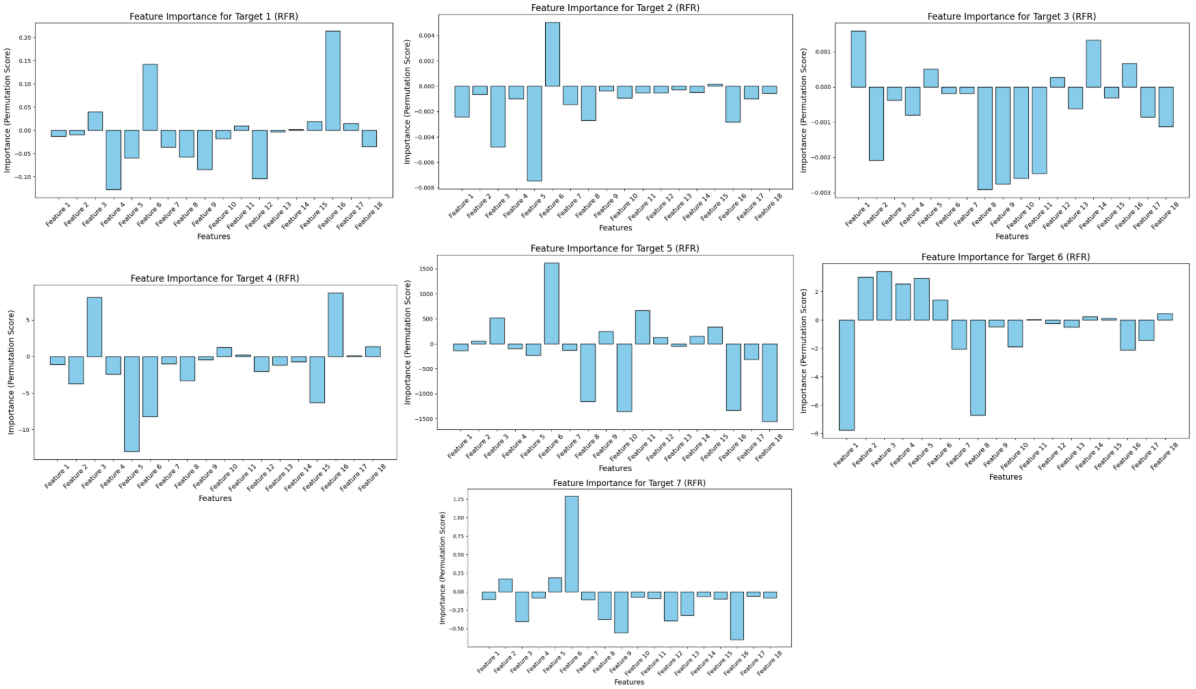
MODEL EVALUATION METRICS

```
Random Forest Model Evaluation:
Training Metrics:
Mean Squared Error (MSE): 18.776175558000197
R-squared (R2) Score: 0.9557172184286251
Mean Absolute Error (MAE): 1.3789963623366088
Root Mean Squared Error (RMSE): 4.3331484578768125

Testing Metrics:
Mean Squared Error (MSE): 17.967600219670974
R-squared (R2) Score: 0.952748818995738
Mean Absolute Error (MAE): 1.399770774977758
```

HYPERPARAMETER TUNING

```
Best Parameters found:
{'max_depth': 20, 'min_samples_split': 30, 'n_estimators': 800}
Best R-squared (R2) score found:
0.9293170598408972
```

Fig 5.11 Model evaluation results



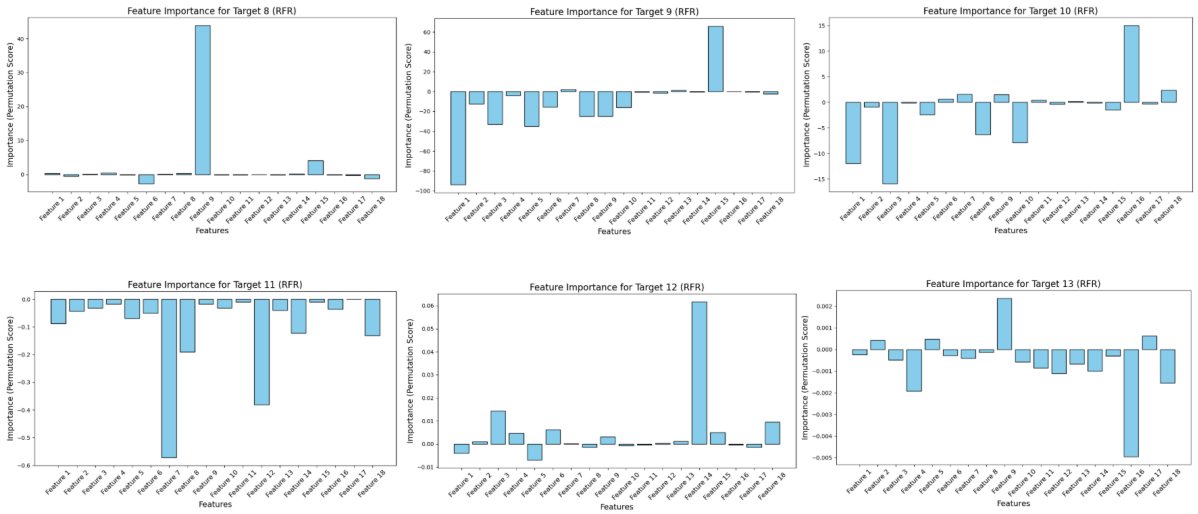


Fig 5.12 Feature Importance for each target variable.

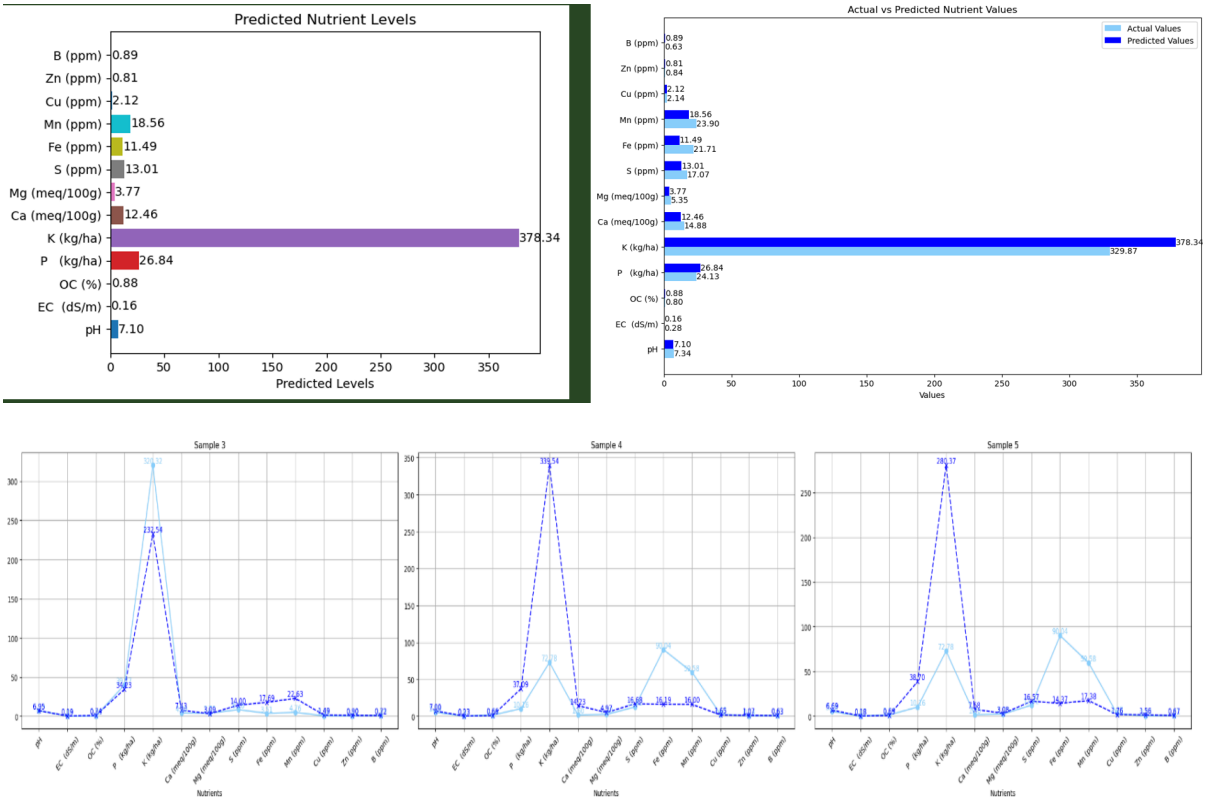


Fig 5.13 represents Random Forest Regressor's prediction and the actual values.

3. Gaussian Process Regressor- Gaussian Process Regression (GPR) is a flexible, nonparametric regression method that offers an approach to making predictions and quantifying uncertainty in those predictions. Unlike parametric models like linear

regression, which assume a specific form for the function being modeled, GPR allows for more flexibility by placing a Gaussian distribution over possible functions, which can better adapt to the data's complexity. Gaussian Process Regression is a powerful tool for regression tasks, particularly for smaller datasets with complex relationships. With hyperparameter tuning and cross-validation, GPR can achieve high accuracy while providing uncertainty estimates for predictions, which are essential for many real-world applications.

Advantages of GPR

- Non-parametric Nature: Unlike traditional regression techniques that fit fixed-form functions (e.g., linear regression), GPR adapts the model complexity to the data.
- Uncertainty Quantification: GPR gives uncertainty (variance) along with the predictions, which is helpful in many real-world problems where confidence in predictions is important.
- Flexibility: By choosing appropriate kernel functions, GPR can model a wide range of data patterns, including smooth, periodic, and more complex relationships.
- Works Well for Small Datasets: GPR typically performs well on small to medium-sized datasets where it can capture the underlying data structure effectively.

Procedure:

- By following these steps, Gaussian Process Regression can be effectively applied to your spectroscopic dataset, using robust feature selection and hyperparameter optimization techniques.
    - Data Pre-processing.
    - Feature selection using importance scores.
    - GPR model training and hyperparameter training.
    - Training on important features.
    - Making predictions and evaluating the model.
- RobustScaler: Since spectroscopic data often contains outliers, a RobustScaler

is applied to scale the data. Unlike standard scaling, which uses mean and variance, the RobustScaler uses the median and interquartile range, making it more resistant to outliers.

- Hyperparameter tuning: is crucial to optimize model performance. We use GridSearchCV to explore a range of kernel parameters to find the best-performing GPR model. Cross-validation with k folds ensures the model's generalization to unseen data.
- Best Model: After using GridSearchCV and k-fold cross-validation, the best GPR model with optimized hyperparameters is selected for prediction.
- Predictions: The GPR model provides predicted values for the target variable based on the selected features (importance_df).
- Evaluation Metrics: The final evaluation metrics such as MSE and R^2 are used to assess the model's performance.
    - Mean Squared Error (MSE): Measures the average of the squares of the errors between actual and predicted values.
    - R-squared (R^2): Measures how well the regression model fits the data. A higher R^2 indicates a better fit.
- Making Predictions: Use the trained model to make predictions on the test data (scaled and selected features).

Trade-offs When Using 1000 Rows

1.Advantages of Larger Dataset:

•Improved Generalization: GPR can learn more robust patterns, reducing overfitting.

•Better Metrics: MSE and R^2 scores improve due to the availability of diverse samples.

2.Challenges with Larger Dataset:

•Computational Complexity: GPR scales poorly with dataset size ($O(n^3)$ complexity), making training slow and resource-intensive.

•Memory Usage: Large datasets can strain memory resources, requiring approximations like sparse GPR.

3.Trade-off Example:

- 100 Rows: Fast training, but the model might underperform on unseen data due to insufficient variability.
- 1000 Rows: Better generalization and performance metrics (lower MSE, higher R^2), but at the cost of computational efficiency.

```python
import numpy as np
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
from sklearn.model_selection import GridSearchCV, train_test_split, KFold
from sklearn.metrics import make_scorer, r2_score, mean_squared_error, mean_absolute_error

# Assuming X_scaled and y_imputed are your preprocessed data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_imputed, test_size=0.2, random_state=42)

# Define an RBF kernel with an initial length scale
kernel = C(1.0, (1e-2, 1e2)) * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))

# Initialize Gaussian Process Regressor
gpr = GaussianProcessRegressor(kernel=kernel, random_state=42)

# Define parameters for Grid Search
param_grid = {
    'alpha': [1e-2, 1e-1, 1.0, 10.0, 100.0],  # Regularization parameter
    'kernel__k2__length_scale': [0.1, 1.0, 10.0]  # Length scale parameter for the RBF kernel
}
```

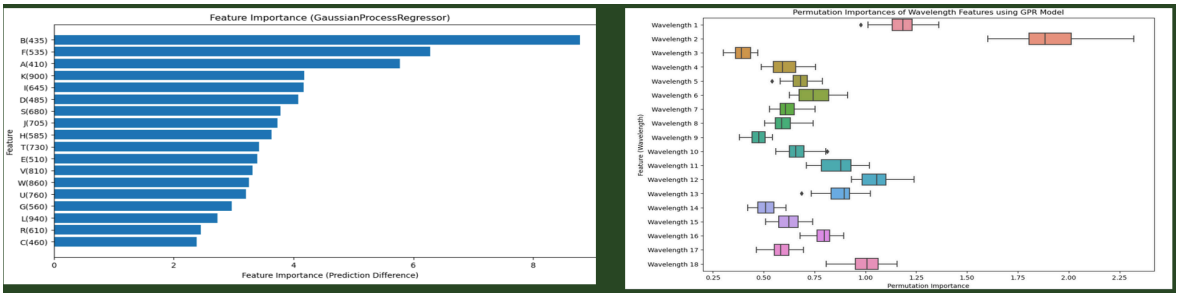
```
Best Parameters found:
{'alpha': 1.0, 'kernel__k2__length_scale': 0.1}

Best R2 Score found:
0.9851797814822321

Test R2 Score: 0.9847325350167885
Test MSE: 0.09895904039455533
Test MAE: 0.10804407518186308
```

HYPER PARAMETER TUNING TO SELECT THE BEST MODEL

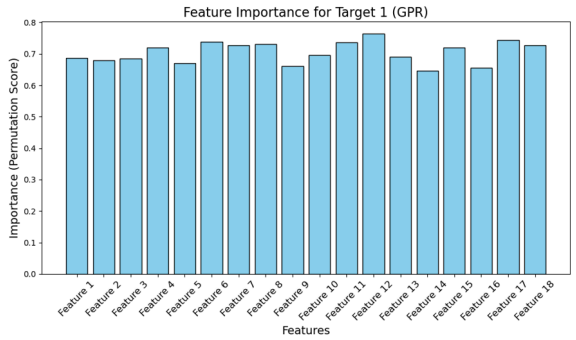
```
Gaussian Process Model Evaluation:
Mean Squared Error (MSE): 0.1102259699937934
R-squared (R2) Score: 0.987741945409682
R-squared (R2) Score on test data: 0.9847325350167885
Mean Absolute Error (MAE): 0.10689958371574965
Root Mean Squared Error (RMSE): 0.3320029668448663
```

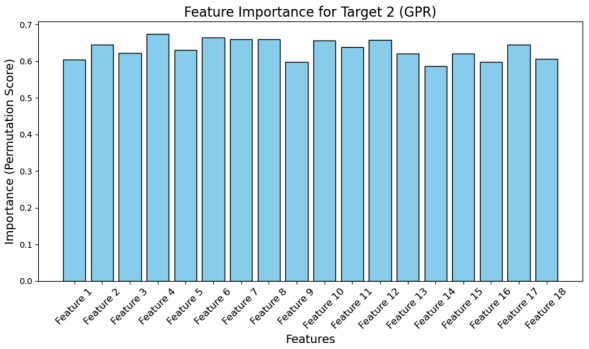
MODEL EVALUATION METRICS

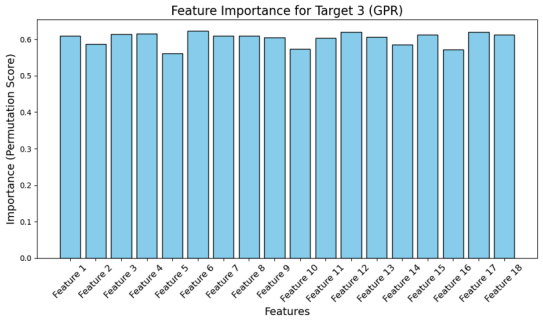
Fig 5.14 represents the GPR parameters and the evaluation metric values.



FEATURE IMPORTANCE WITH RESPECT TO GPR MODEL







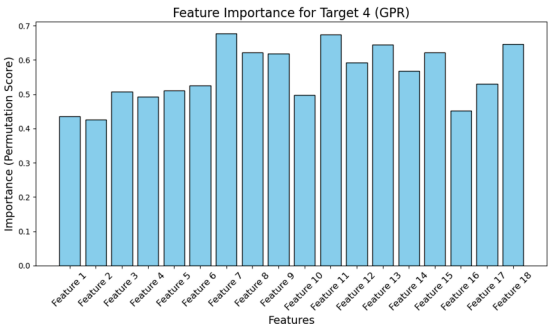
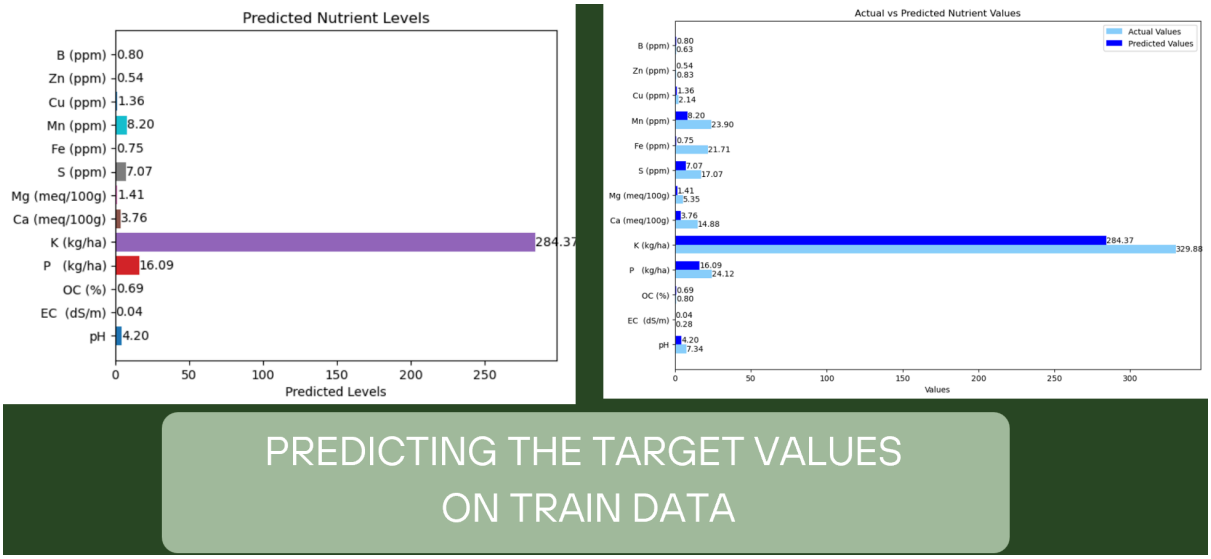


Fig 5.15 Feature Importance for each target variable wrt GPR model.



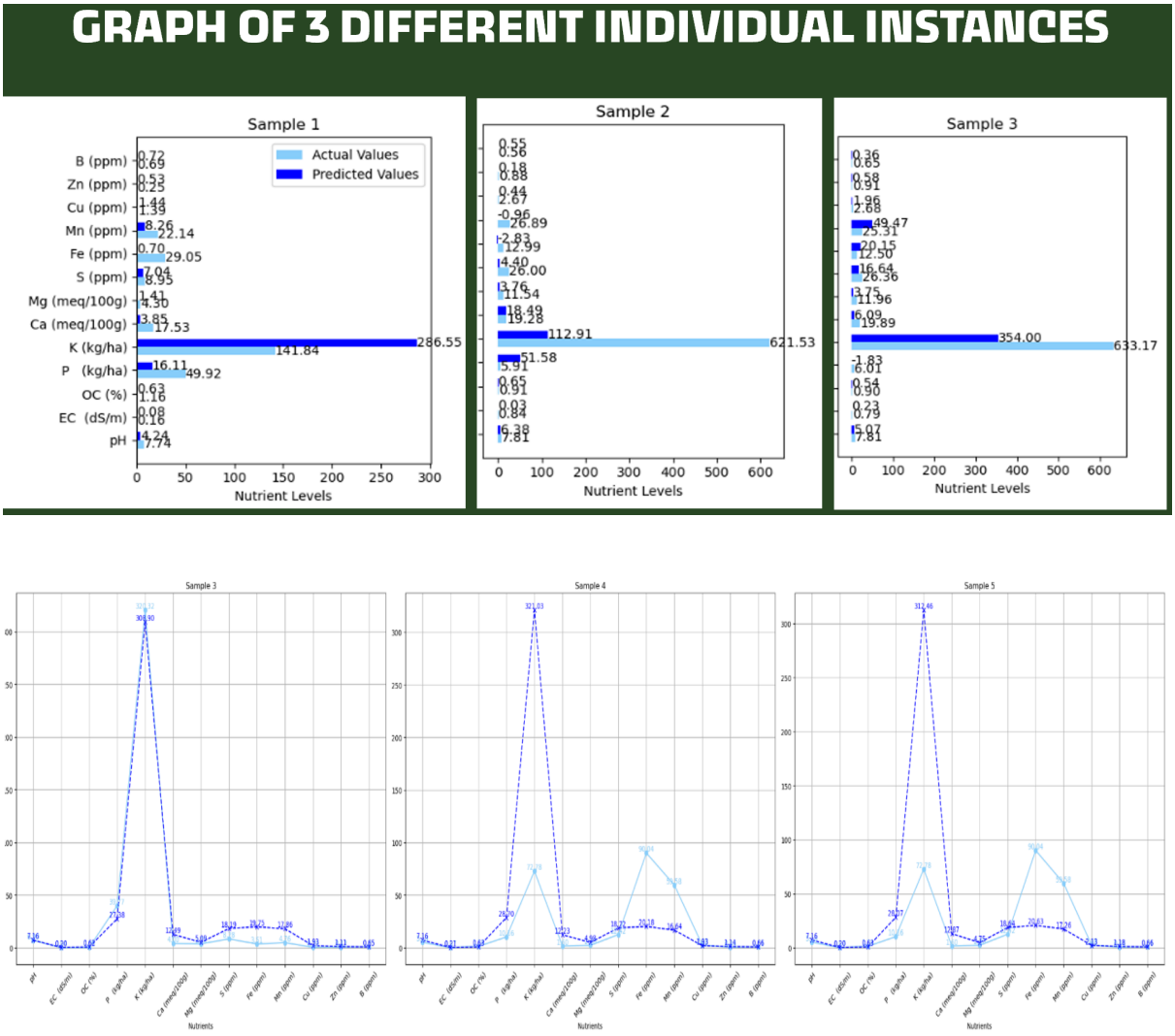


Fig 5.16 represents the actual and the predicted values of the GPR model.

4. Partial Least Square Regression- Partial least squares regression. With the goal of using regression to model the data relationships that involve multiple dependent and independent variables, PLSR is applicable to the processing of abundant sample data . Moreover, when combined with the advantages of PCA, PLSR can eliminate some of the variation in the data and extract the information in a way that is most effective in analyzing the data. In addition, PLSR has been extensively used in spectral analysis modeling to establish a linear regression model between predictive and observation variables.

- Partial Least Squares Regression (PLSR) is a statistical method that is particularly useful when you have highly collinear, noisy, or numerous predictor variables (features) relative to observations. Unlike traditional regression methods, PLSR works by finding components that maximize the covariance between the predictors and the response variable(s), helping to model the relationship more effectively.
- PLSR is a supervised learning method, as it takes into account the target variable (dependent variable) while creating latent components. This differentiates it from techniques like Principal Component Analysis (PCA), which is unsupervised and only focuses on the input features without considering the target.

Why PLSR is significant:

- Multicollinearity: Spectroscopic data often has highly collinear variables, which PLSR handles well.
- Dimensionality Reduction: PLSR reduces the dimensionality by creating latent variables, improving model performance and interpretability.
- Efficiency: By selecting the most important features (with CARS) and using a small number of components in PLSR, we avoid overfitting and improve predictive accuracy.

- PLS regression often requires scaling both the input features and the target variables to ensure that they are on a comparable scale. We are using StandardScaler to normalize your data, ensuring that all features and target variables have mean 0 and variance 1. After scaling back the predicted values to their original scale, we are visualizing the actual vs. predicted nutrient levels to evaluate your model's performance.
- PLSR is highly beneficial in situations where we have many predictor variables but a relatively small sample size—which is a typical scenario in spectroscopic analysis. Traditional regression methods struggle with datasets where the number of features exceeds the number of samples, as the model becomes prone to overfitting. PLSR, by reducing the dimensionality and focusing on the covariance between X and Y, is much more robust in these cases.
- In this case, PLSR allows us to build a model that can generalize well even when the number of features is large and the number of training samples is limited.
- Another advantage of PLSR is that the latent components it extracts can be interpreted in terms of their contribution to the prediction of the response variables. By examining the loadings (coefficients) of each predictor variable on the latent components, we can gain insights into which features are driving the predictions.
- In this case, this might help us understand which spectroscopic features (wavelengths or frequencies) are most important for predicting specific nutrient levels.
- For evaluating PLSR models, we typically look at metrics such as R-squared, Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). These metrics indicate how well the model's predictions match the actual values.
- In this project, we use PLSR to predict nutrient levels based on spectroscopic features. After applying CARS to reduce the dimensionality, the selected features are used to train a PLSR model. We scale both the features and target variables, perform the regression, and then scale the predictions back to the original units for interpretation.

In summary, PLSR allows us to model the relationship between spectroscopic features and nutrient levels efficiently, handling the challenges of multicollinearity and high-dimensional data. By scaling your data appropriately and selecting relevant features, we enhance the model's performance and achieve meaningful predictions.

Feature Importance and Cars Algorithm:

Competitive Adaptive Reweighted Sampling (CARS) is a feature selection algorithm commonly used in machine learning, particularly in scenarios involving high-dimensional data. It is mainly employed in regression problems such as Partial Least Squares (PLS) regression. The goal of CARS is to identify the most informative subset of features from a large set, optimizing model performance while reducing computational complexity.

Key Concepts of CARS:

1. **Adaptive Sampling**:
    - CARS uses adaptive reweighting to prioritize more informative features and gradually eliminate less relevant ones during each iteration.
    - The algorithm iteratively "samples" features using a competitive mechanism, ensuring only the most significant variables are retained.
2. **Exponential Decay Function**: CARS implements an exponential decay function to progressively reduce the number of variables considered, ensuring that the feature subset size decreases across iterations.
3. **Monte Carlo Sampling**: It applies Monte Carlo sampling to the dataset, generating different feature subsets in each iteration. These subsets compete to determine which features are most important for model performance.
4. **PLS Regression**: CARS is often paired with PLS regression. In each iteration, a PLS model is built using a random subset of features, and the importance of each feature is evaluated based on its contribution to the model.
5. **Feature Elimination**: Features with low significance are eliminated based on their regression coefficients. After a series of iterations, only the most relevant features remain.

6. **Final Selection**: After several iterations, CARS selects the optimal set of features that contribute the most to model accuracy. This reduces overfitting and enhances model generalizability.

Applications of CARS

Spectroscopic Data Analysis CARS is frequently used in chemometrics to analyze spectroscopic data, where datasets often have thousands of variables (such as wavelengths), and only a few are relevant for predicting chemical properties.

Steps to Apply CARS for Feature Selection:

1. **Data Preprocessing**: Normalize or standardize your spectroscopic data to ensure that all features are on the same scale.
2. **Set Up a PLS Model**: Since CARS typically works with PLS (Partial Least Squares) regression, you'll need to define a PLS model to evaluate feature importance during each iteration.
3. **Apply CARS**: Run the CARS algorithm, which will iteratively sample and evaluate the features based on their contribution to the PLS model. In each iteration, CARS will apply an exponential decay function to reduce the number of features, eliminating those with lower significance.
4. **Feature Elimination**: After several iterations, the algorithm will provide a subset of the most important features that have the highest predictive power.
5. **Validate the Model**: Validate the model using the selected features to ensure that model performance improves with fewer features, reducing the risk of overfitting.

```python
from sklearn.cross_decomposition import PLSRegression

# Initialize the PLSRegression model with a suitable number of components
pls = PLSRegression(n_components=2)

# Fit the model on the training data
pls.fit(X_train, y_train)

# Predict on training and testing sets
y_train_pred = pls.predict(X_train)
y_test_pred = pls.predict(X_test)

# Check the model performance (R^2 score)
train_score = pls.score(X_train, y_train)
test_score = pls.score(X_test, y_test)
print(f'PLS Training R^2 Score: {train_score}')
print(f'PLS Testing R^2 Score: {test_score}')
```

```python
import numpy as np
from sklearn.utils import resample

def competitive_adaptive_reweighted_sampling(X_train, y_train, pls, iterations=50):
    n_features = X_train.shape[1]
    selected_features = list(range(n_features))

    for i in range(iterations):
        # Train the PLS model on the current set of features
        pls.fit(X_train[:, selected_features], y_train)

        # Get regression coefficients (absolute values for ranking)
        coefficients = np.abs(pls.coef_).flatten()

        # Sort features by importance (lower coefficients will be eliminated)
        ranked_features = np.argsort(coefficients)

        # Eliminate a certain percentage of less important features in each iteration
        elimination_rate = 0.1  # Eliminate 10% of the least important features
        n_to_eliminate = max(1, int(elimination_rate * len(selected_features)))
        selected_features = [selected_features[j] for j in ranked_features[:-n_to_eliminate]]

        print(f'Iteration {i + 1}: Number of selected features: {len(selected_features)}')

    return selected_features

# Apply CARS for feature selection
selected_features = competitive_adaptive_reweighted_sampling(X_train.values, y_train.values, pls)
print(f'Selected Features after CARS: {selected_features}')
```

Fig 5.17 represents the model training of PLS model and the implementation of CARS algorithm.

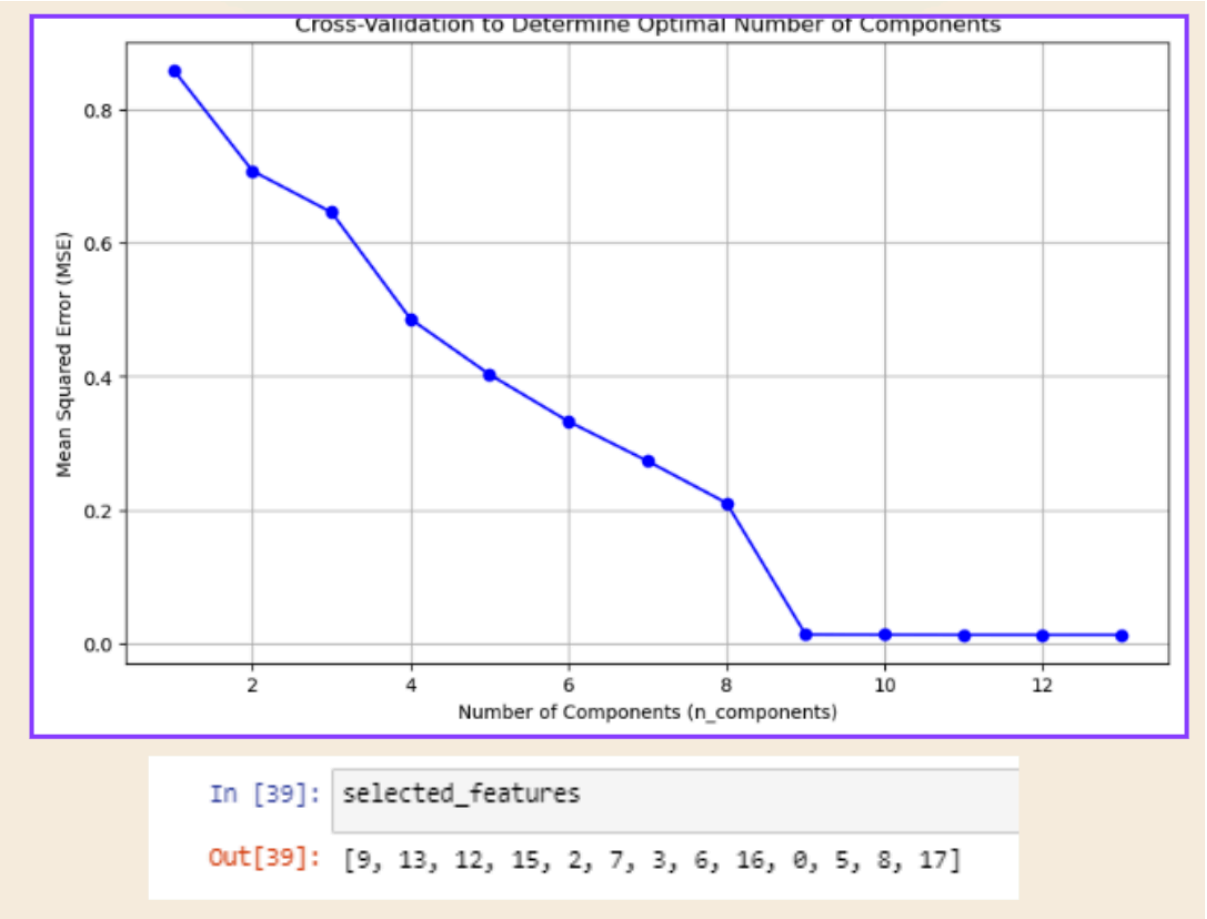
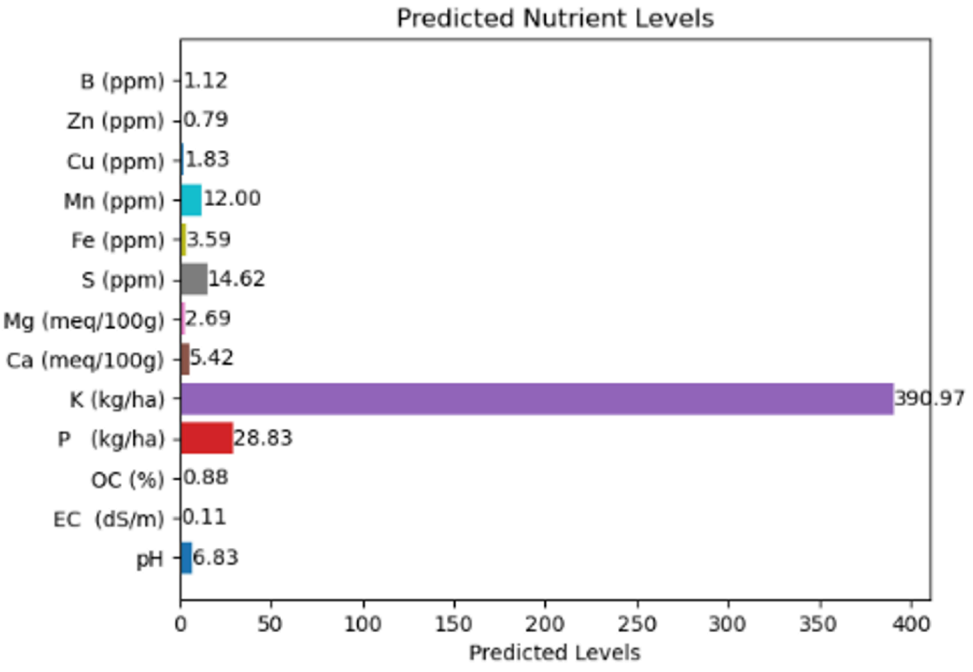


Fig 5.18 represents the selected features based on the CARS algorithm.



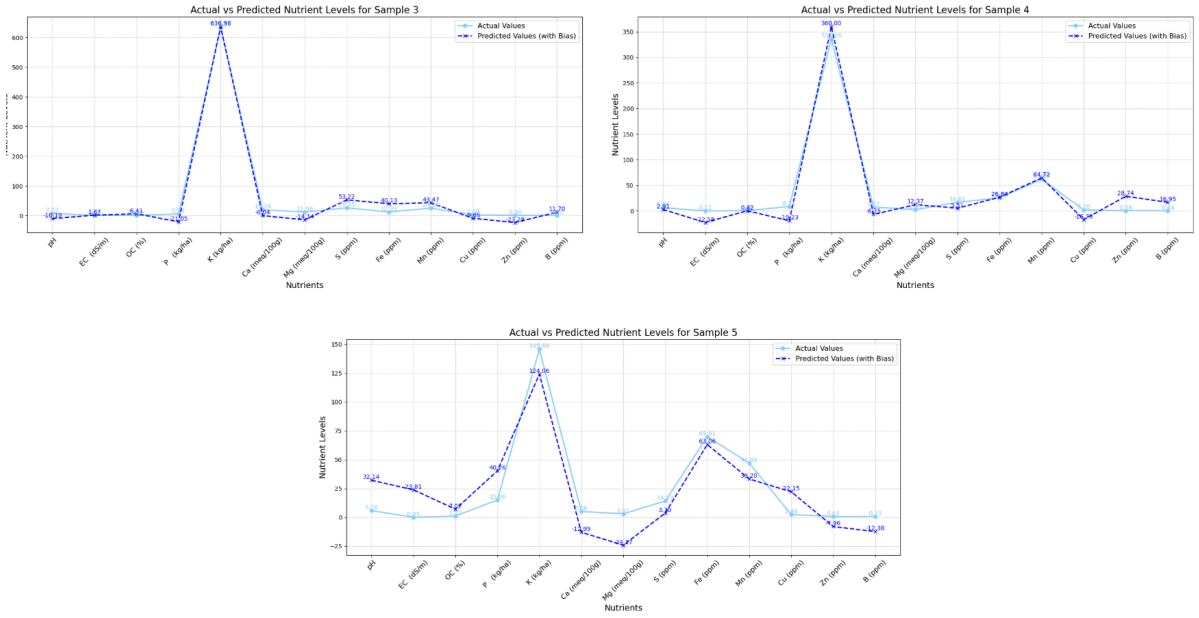


Fig 5.19 represents the actual and the predicted values for three sample inputs.

Chapter 6: Results and UI Design

A user-friendly web application that predicts soil nutrient levels based on hyperspectral reflectance values. The application is powered by a trained K-Nearest Neighbors (KNN) regression model and provides visual insights into the predicted nutrient content to aid in precision agriculture and sustainable farming. The application can be integrated with any of the trained models.

Model Serialization and Pickle

Pickle is a Python module used for serializing and deserializing Python objects. Serialization means converting a Python object (like a trained machine learning model or a list) into a byte stream that can be saved to a file. Deserialization means loading that byte stream back into the original Python object.

In machine learning projects, once we train a model, we often want to save it so we don't need to retrain it every time. pickle lets us save:

- Trained models (best_knn_model.pkl)
- Scalers (scaler.pkl)
- Lists like feature or target column names (feature_columns.pkl, target_columns.pkl)

These files can then be loaded directly in a different Python script or application, like a Streamlit app, for real-time prediction.

Streamlit

Streamlit is an open-source Python library that allows you to build interactive web applications for data science and machine learning without requiring HTML, CSS, or JavaScript.

Streamlit turns any Python script into a web app by running it with: streamlit run app.py

Why We Used Streamlit in This Project:

- To create a user interface for non-technical users (like farmers or agronomists).
- To take manual input of soil reflectance values.
- To show predicted nutrient values in a readable and visual way.
- To integrate ML model prediction in real-time.

Key Concepts

1. Hyperspectral Data Input

- Soil samples are analyzed across 18 specific wavelengths.
- Each wavelength corresponds to a feature like A(410), B(435), etc.
- These features represent reflectance values of the soil at particular light frequencies.

2. Model and Preprocessing

- The trained model is a KNN Regressor, chosen for its ability to learn from patterns in reflectance values and predict multiple continuous nutrient levels.
- Before prediction:
    - Input values are scaled using the same MinMaxScaler or StandardScaler used during training.
    - The model outputs 13 nutrient levels, such as Nitrogen, Phosphorus, Potassium, etc.

3. Web Interface with Streamlit

- Streamlit is used to build an intuitive interface for non-technical users, like farmers or agronomists.
- The interface provides:
    - Title and instruction area.
    - 18 interactive number input fields.
    - A "Predict" button to trigger the model.
    - Display of predicted nutrient values.

- A visual bar chart showing all predictions for comparison.

4. Deployment

- Streamlit app can be deployed easily using platforms like Streamlit Cloud or packaged in Docker containers for more advanced deployments.

Workflow of the Application

Step 1: User Input

- The user enters the reflectance values for all 18 wavelengths manually into the interface.
- Default values are pre-filled to guide users.

Step 2: Preprocessing

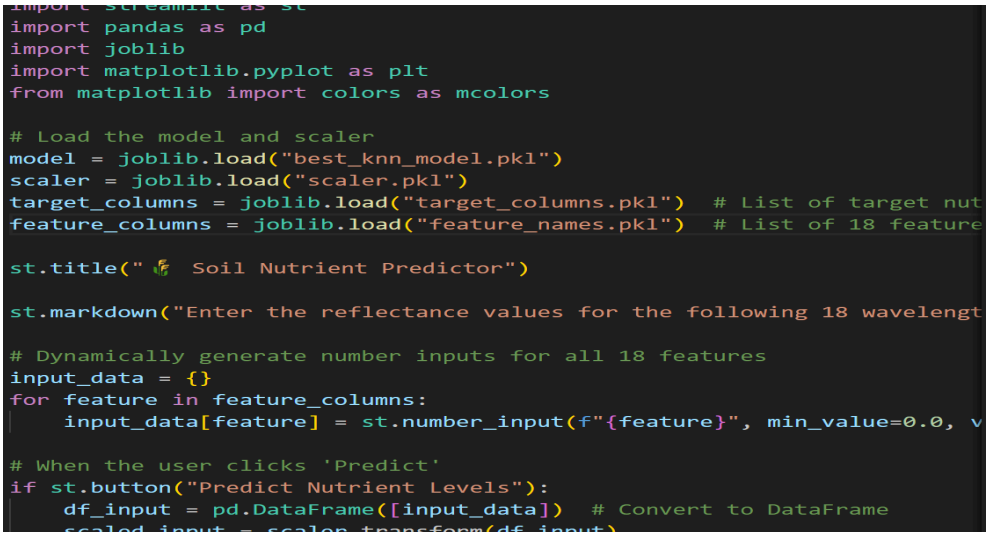
- Inputs are gathered into a Pandas DataFrame.
- The DataFrame is scaled using a preloaded scaler.pkl to match the training phase.

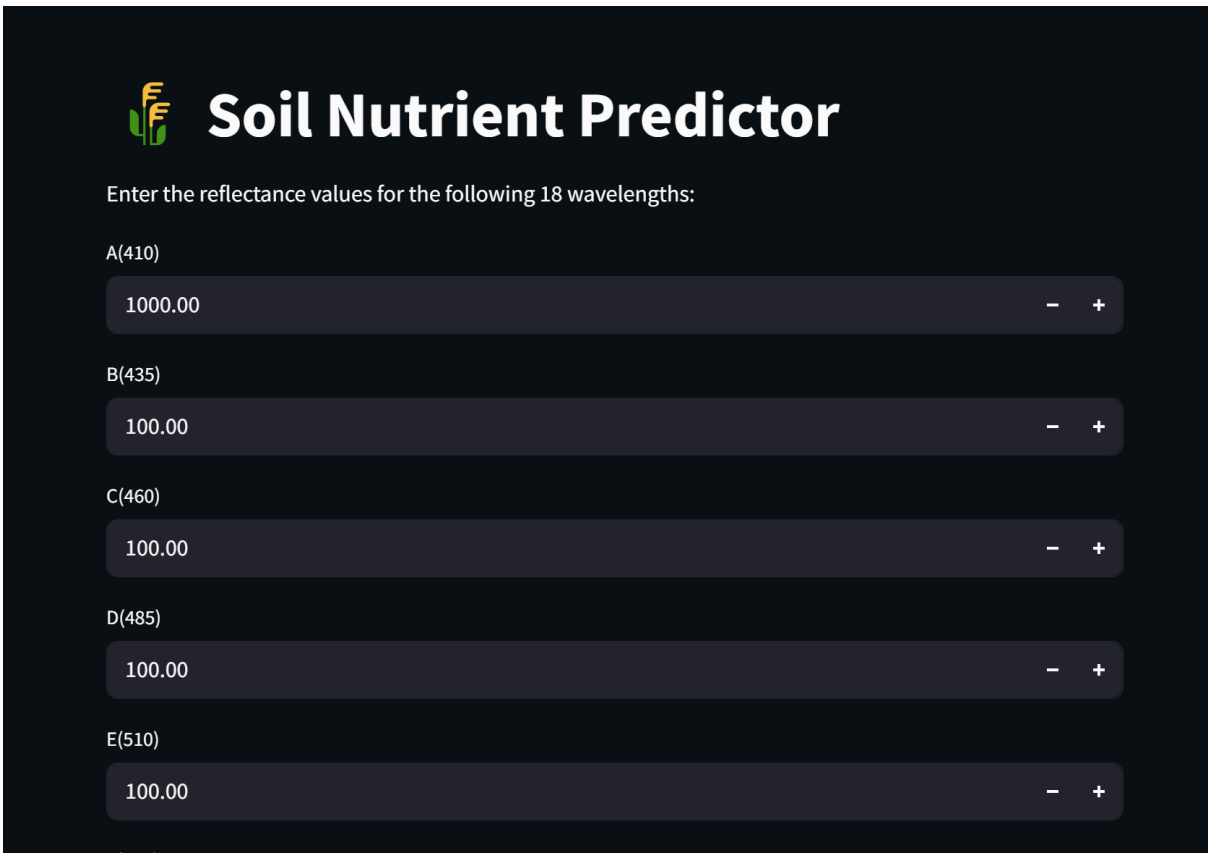
Step 3: Prediction

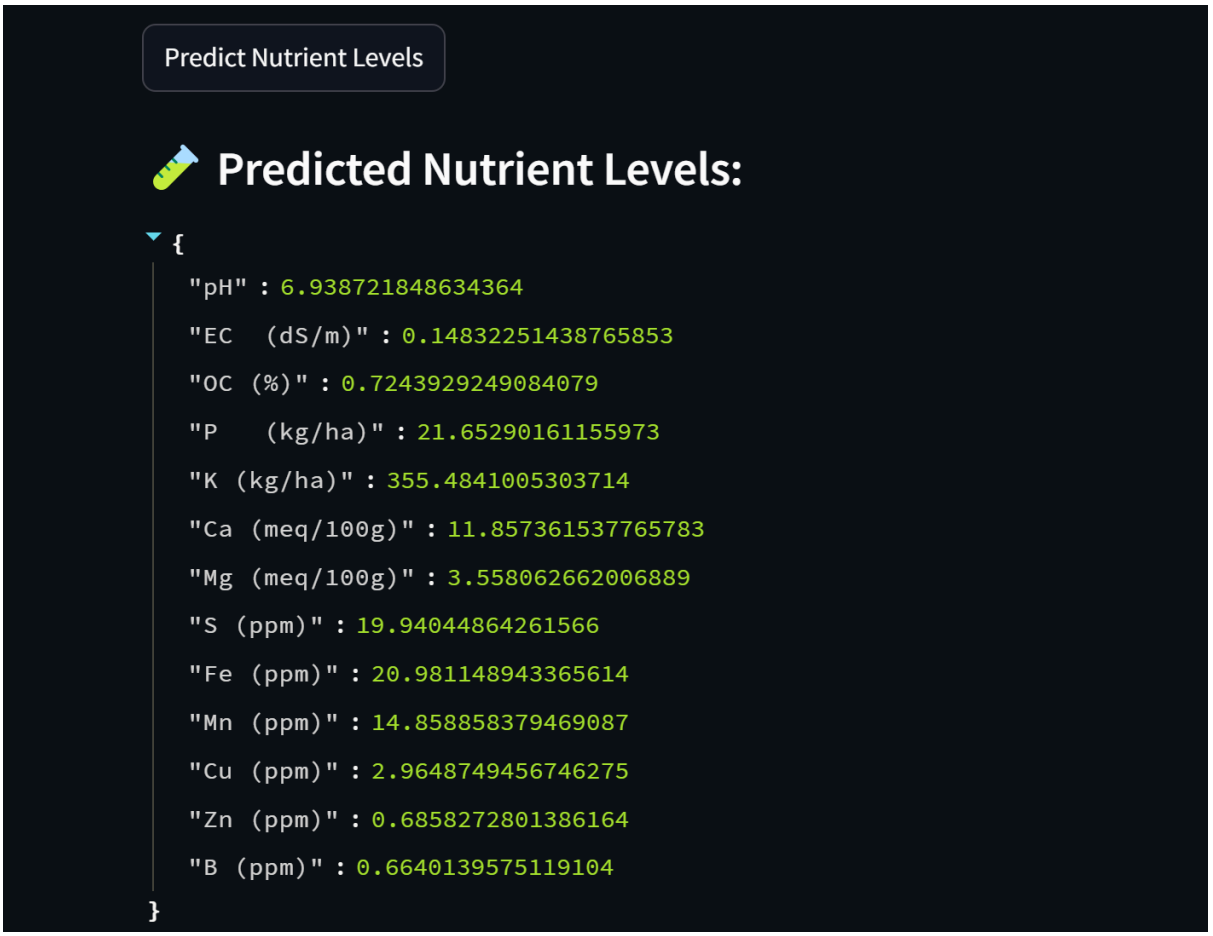
- The processed input is fed into the trained best_knn_model.pkl.
- The model outputs predicted values for each nutrient.

Step 4: Result Display

- The predicted nutrient values are shown in tabular format for clarity..A horizontal bar chart visualizes the levels, helping users easily compare the




Soil Nutrient Predictor
Enter the reflectance values for the following 18 wavelengths:
A(410)
1000.00
B(435)
100.00
C(460)
100.00
D(485)
100.00
E(510)
100.00


Predict Nutrient Levels
Predicted Nutrient Levels:
{
"pH" : 6.938721848634364
"EC (dS/m)" : 0.14832251438765853
"OC (%)" : 0.7243929249084079
"P (kg/ha)" : 21.65290161155973
"K (kg/ha)" : 355.4841005303714
"Ca (meq/100g)" : 11.857361537765783
"Mg (meq/100g)" : 3.558062662006889
"S (ppm)" : 19.94044864261566
"Fe (ppm)" : 20.981148943365614
"Mn (ppm)" : 14.858858379469087
"Cu (ppm)" : 2.9648749456746275
"Zn (ppm)" : 0.6858272801386164
"B (ppm)" : 0.6640139575119104
}

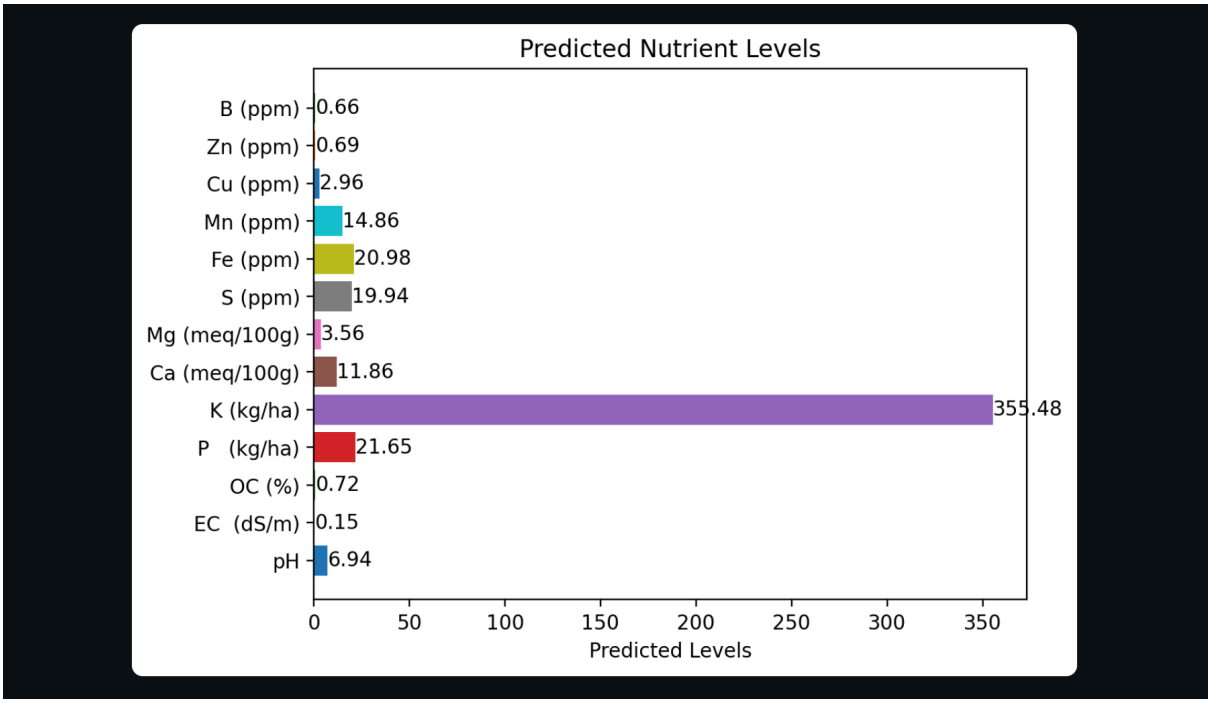


Fig 6.1 represents the code snapshot and the streamlit interface. The values of the features can be entered and on clicking the Predict button, we get the values of the soil nutrients predicted by the model. These predicted values are represented visually with a horizontal bar graph.

The Streamlit application was deployed using Streamlit Community Cloud. To ensure smooth deployment, a requirements.txt file was created and cleaned to include only the necessary libraries required by the app. This helps in reducing load time and avoiding dependency conflicts.

Once deployed, the app can be accessed by simply sharing the public URL. No additional installation is required by the end-user — the application runs directly in the browser.

App Link: https://arkashinevanshika-jmsz5jfljmfofkbhfr5gzw.streamlit.app/

Chapter 7: Conclusion

This project successfully demonstrates the potential of integrating spectroscopic data analysis with machine learning to revolutionize soil nutrient assessment. By analyzing visible and near-infrared spectral data, we were able to non-invasively capture detailed information about soil composition. Using this data, we trained and evaluated multiple regression models—including KNN, Random Forest, GPR and PLSR to accurately predict critical soil properties such as pH, Potassium (K), and other macro- and micronutrient levels.

To enhance usability and accessibility, the trained models were seamlessly deployed within an interactive Streamlit application. This interface allows users—farmers, agronomists, and researchers—to input spectral readings and receive instant predictions of soil health indicators without needing complex laboratory tests.

By offering a scalable, cost-effective, and real-time solution for soil analysis, this project represents a significant step toward achieving precision agriculture. It empowers stakeholders to make informed decisions about fertilization, crop planning, and sustainable land use, ultimately contributing to improved agricultural productivity, resource optimization, and environmental conservation.

In essence, this project bridges the gap between cutting-edge technology and practical agricultural applications, reinforcing the vital role of AI-driven tools in building a more sustainable and food-secure future.

Project Repository Available at
https://github.com/buddulavanshika/Vanshika_ArkaShine.git