Smart Soil Crop Recommender Dataset

AGENDA

ABSTRACT

INTRODUCTION

RESEARCH QUESTIONS

METHODOLOGY

DATASET

EXPLORATORY DATA ANALYSIS

MODEL EVALUATION

FUTURE ENHANCEMENT

REFERENCES

ABSTRACT

The "Smart Soil Crop Recommender Dataset" project is designed to assist in making data-driven agricultural decisions by recommending suitable crops based on soil characteristics. This project aims to empower farmers with accurate, region-specific recommendations to enhance crop yields and ensure efficient resource utilization. The presentation outlines the dataset's composition, the analytical methods employed, and the project's anticipated impact on modern agriculture.

INTRODUCTION

Agriculture is a cornerstone of human civilization, and its productivity is heavily influenced by soil quality and nutrient availability. The "Smart Soil Crop Recommender Dataset" project focuses on developing an intelligent system to support farmers in selecting the most suitable crops for their soil conditions. The dataset used includes 620 observations of soil samples, capturing critical features like macronutrients (N, P, K), pH levels, electrical conductivity (EC), sulphur (S), and trace elements (Cu, Fe, Mn, Zn, B). These data points are labeled with corresponding crops, such as pomegranate, ragi, and mulberry.

This project combines advanced data analysis and machine learning techniques to build a recommendation model that integrates agricultural science with modern technology. The primary objective is to provide accurate crop recommendations, improve farming efficiency, and minimize environmental impacts.

RESEARCH QUESTIONS

01 Which crops are best suited for different soil nutrient levels ?

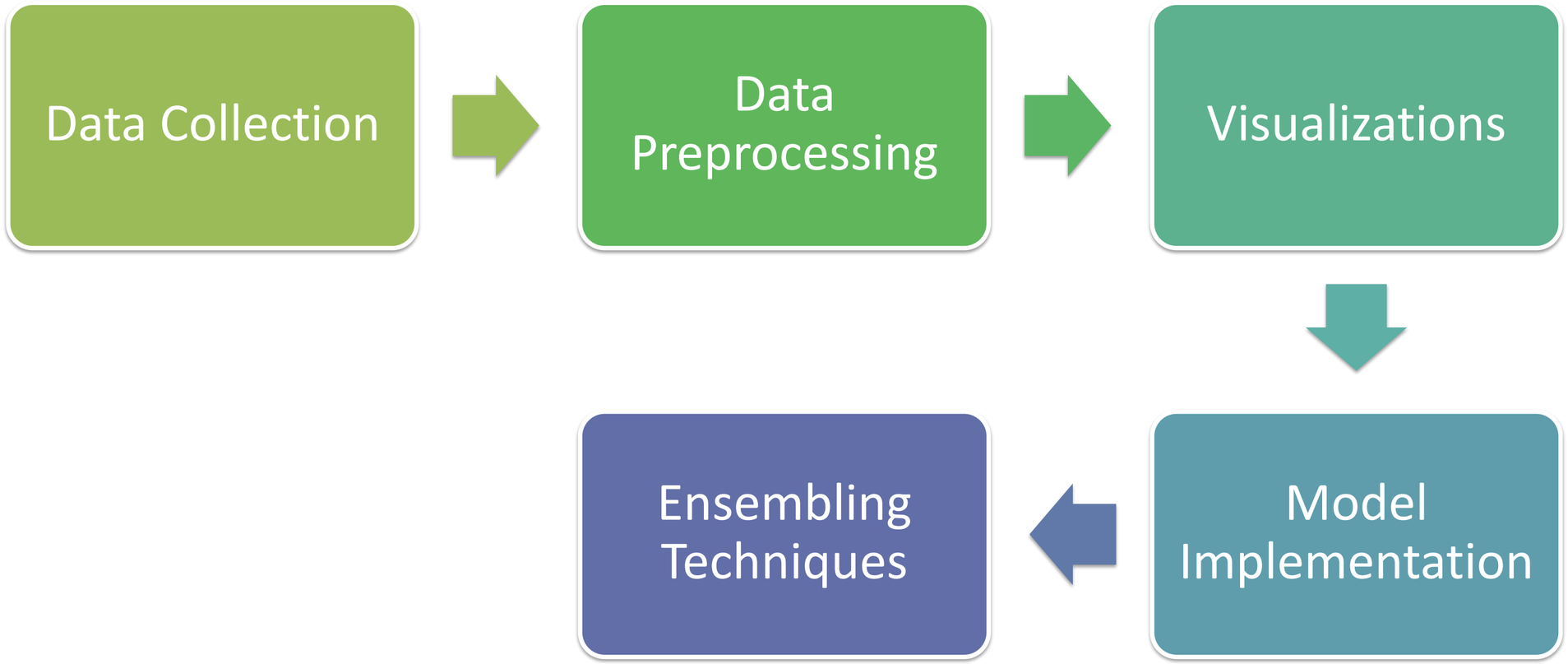
02 What is the relationship between nitrogen levels and different crop types ?

03 Which nutrients are most critical for specific crops ?

04 How can farmers improve their crop selection based on soil health ?

05 How can farmers improve their crop selection based on soil health ?

METHODOLOGY



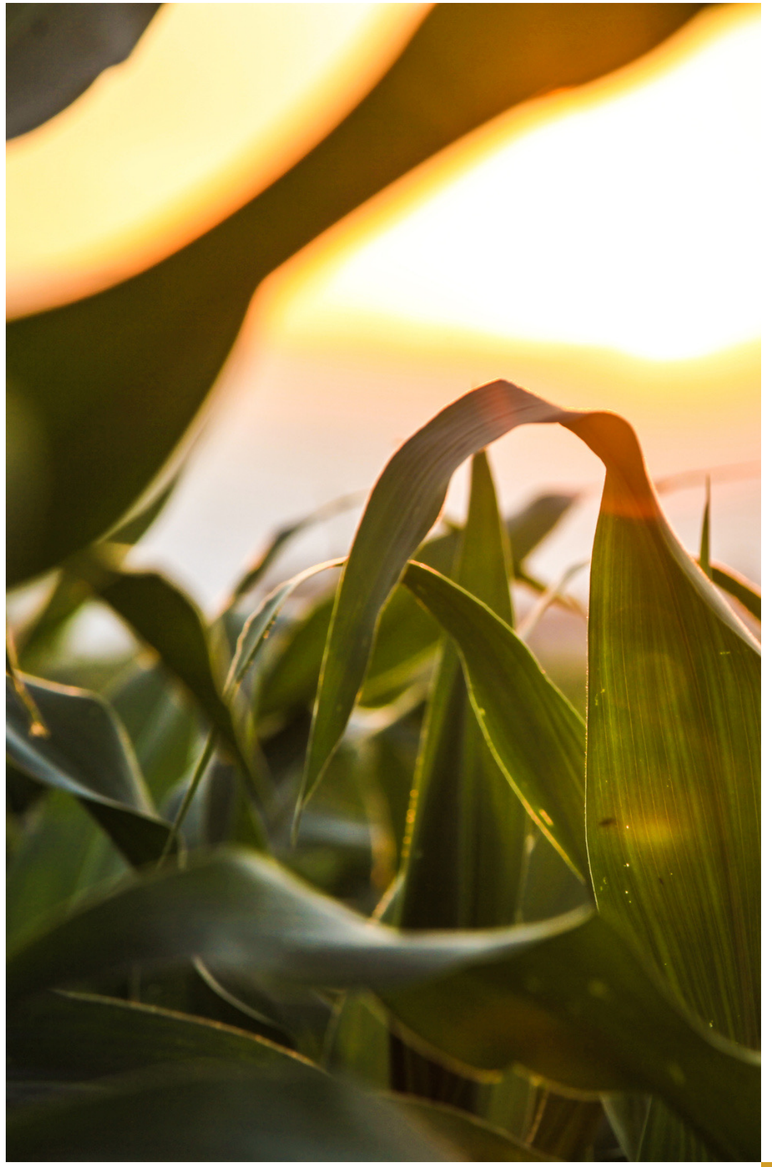
Dataset

Source of the Dataset: crop-recommendation-dataset

No.of Observations : 620 rows & 12 columns

Features Details : N (Nitrogen): Essential for plant growth, promoting leafy development and overall health**, P (Phosphorus):** Supports root development and flower/fruit production in plants**, K (Potassium):** Enhances drought resistance, disease resistance, and overall plant vigor, **ph:** Indicates soil acidity or alkalinity, affecting nutrient availability to plants, **EC (Electrical Conductivity):** Measures soil salinity and the ability to conduct electrical current, which influences nutrient absorption, **S (Sulfur):** Key for protein synthesis and chlorophyll formation in plants**, Cu (Copper):** Vital for enzymatic processes and the formation of chlorophyll, **Fe (Iron):** Critical for chlorophyll synthesis and enzymatic reactions in plants, **Mn (Manganese):** Important for photosynthesis and nitrogen metabolism in plants, **Zn (Zinc):** Supports plant growth by aiding in hormone production and enzyme activation, **B (Boron):** Essential for cell wall formation and reproductive growth in plants, **label** in the dataset represents the specific crop associated with the given soil properties, such as pomegranate, ragi, or mulberry.

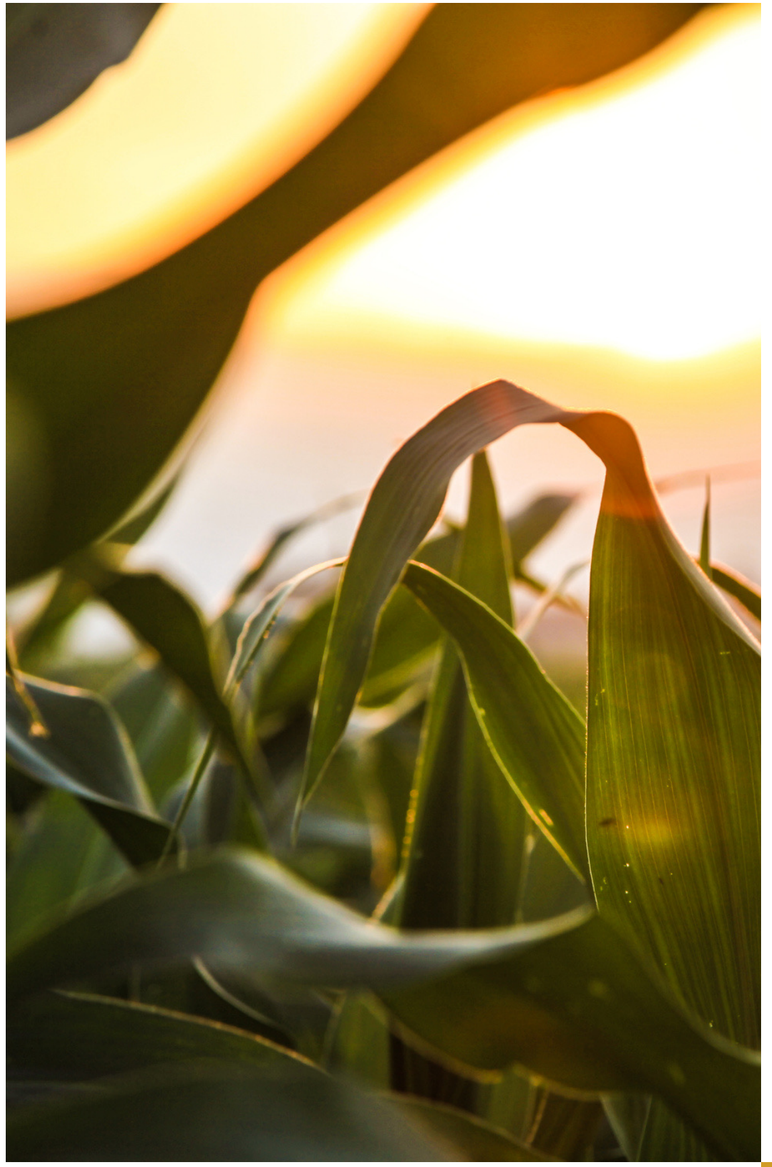
Details about the columns : Discrete & categorical



Screenshot of the Dataset

	N	P	K	ph	EC	S	Cu	Fe	Mn	Zn	B	label
0	143	69	217	5.9	0.58	0.23000	10.20	116.35	59.96	54.85	21.29	pomegranate
1	170	36	216	5.9	0.15	0.28000	15.69	114.20	56.87	31.28	28.62	pomegranate
2	158	66	219	6.8	0.34	0.20000	15.29	65.87	51.81	57.12	27.59	pomegranate
3	133	45	207	6.4	0.94	0.21000	8.48	103.10	43.81	68.50	47.29	pomegranate
4	132	48	218	6.7	0.54	0.19000	5.59	63.40	56.40	46.71	31.04	pomegranate
...	...	...	...	...	...	...	...	...	...	...	...	...
615	41	23	135	5.0	1.67	0.10655	26.00	39.20	206.89	31.09	20.64	potato
616	49	45	90	5.8	1.98	0.09229	19.00	40.20	91.12	32.68	14.91	potato
617	131	24	121	4.9	2.24	0.08775	22.00	40.00	94.34	24.93	23.74	potato
618	131	55	130	5.3	2.48	0.08983	15.00	41.00	92.58	45.73	21.48	potato
619	129	34	160	4.8	1.08	0.08869	25.00	39.00	259.93	33.49	14.16	potato

620 rows × 12 columns



EXPLORATORY DATA ANALYSIS

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 620 entries, 0 to 619
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   N       620 non-null    int64  
 1   P       620 non-null    int64  
 2   K       620 non-null    int64  
 3   ph      620 non-null    float64
 4   EC      620 non-null    float64
 5   S       620 non-null    float64
 6   Cu      620 non-null    float64
 7   Fe      620 non-null    float64
 8   Mn      620 non-null    float64
 9   Zn      620 non-null    float64
 10  B       620 non-null    float64
 11  label   620 non-null    object 
dtypes: float64(8), int64(3), object(1)
memory usage: 58.3+ KB
```

	N	P	K	ph	EC	S	Cu	Fe	Mn	Zn	B
count	620.00	620.00	620.00	620.00	620.00	620.00	620.00	620.00	620.00	620.00	620.00
mean	135.70	52.51	161.40	6.11	0.89	0.48	17.32	117.21	219.91	34.34	28.75
std	31.81	24.67	81.29	0.69	0.64	5.78	6.44	62.78	342.00	11.54	19.64
min	30.00	10.00	51.00	4.50	0.01	0.01	0.18	3.81	20.58	17.99	1.02
25%	116.00	30.00	89.00	5.70	0.30	0.09	12.59	68.31	58.78	24.73	13.44
50%	138.00	54.00	157.50	6.10	0.78	0.17	16.91	108.24	73.58	31.42	23.22
75%	160.00	75.00	206.00	6.50	1.39	0.24	21.10	155.62	163.67	43.54	42.77
max	198.00	100.00	369.00	8.00	2.48	113.20	35.00	276.72	1572.54	70.00	74.56

EXPLORATORY DATA ANALYSIS

```
df.isnull().sum()
```

```
N          0
P          0
K          0
ph         0
EC         0
S          0
Cu         0
Fe         0
Mn         0
Zn         0
B          0
label      0
dtype: int64
```

```
df[['N','P','K']]
```

	N	P	K
0	143	69	217
1	170	36	216
2	158	66	219
3	133	45	207
4	132	48	218
...	...	...	...
615	41	23	135
616	49	45	90
617	131	24	121
618	131	55	130
619	129	34	160

620 rows × 3 columns

```
df["label"].unique()
```

```
array(['pomegranate', 'mango', 'grapes', 'mulberry', 'ragi', 'potato'],
      dtype=object)
```

```
df['label'].value_counts()
```

```
label
pomegranate    104
mango          104
grapes         104
mulberry       104
ragi           104
potato         100
Name: count, dtype: int64
```

EXPLORATORY DATA ANALYSIS

```
df[['N', 'P', 'K']].mean()
```

```
N    135.701613
P     52.508065
K    161.401613
dtype: float64
```

```
# Average nutrient contribution for each Plantation
avg_nutrients=df.groupby(by='label')[['N', 'P', 'K']].agg("mean").round(2)
avg_nutrients
```

label	N	P	K
grapes	139.82	47.51	83.73
mango	114.74	78.16	157.46
mulberry	144.41	21.36	86.00
pomegranate	153.08	48.74	193.11
potato	108.88	44.27	152.22
ragi	152.25	74.69	295.54

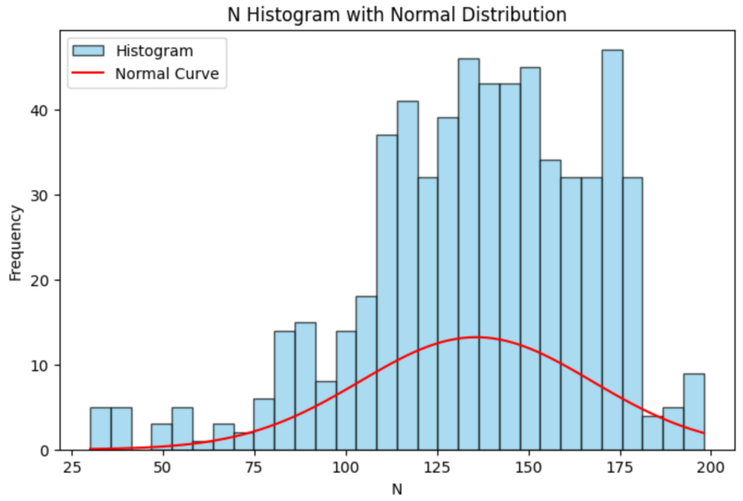
```
# Finding min & max values for PH & EC
df.groupby(by="label")[["ph","EC"]].agg(["min","max"]).round(2)
```

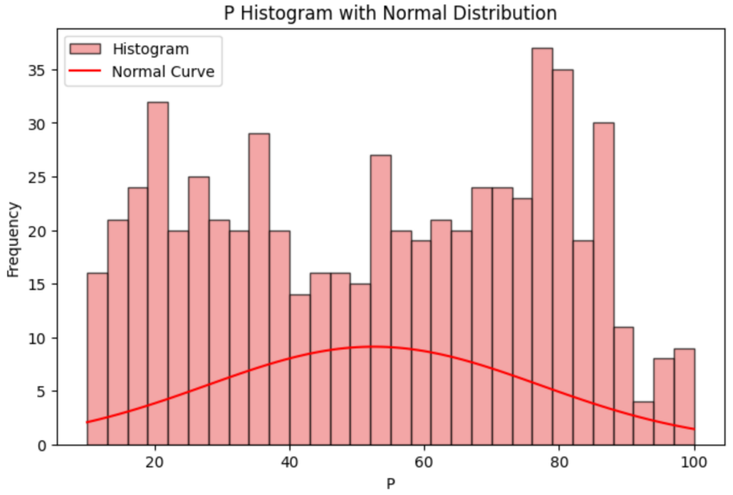
	ph		EC	
	min	max	min	max
label				
grapes	5.5	7.2	0.11	1.45
mango	4.5	7.0	0.23	1.50
mulberry	5.8	7.0	0.10	1.89
pomegranate	5.6	7.2	0.01	1.16
potato	4.6	6.0	1.08	2.48
ragi	4.9	8.0	0.14	2.45

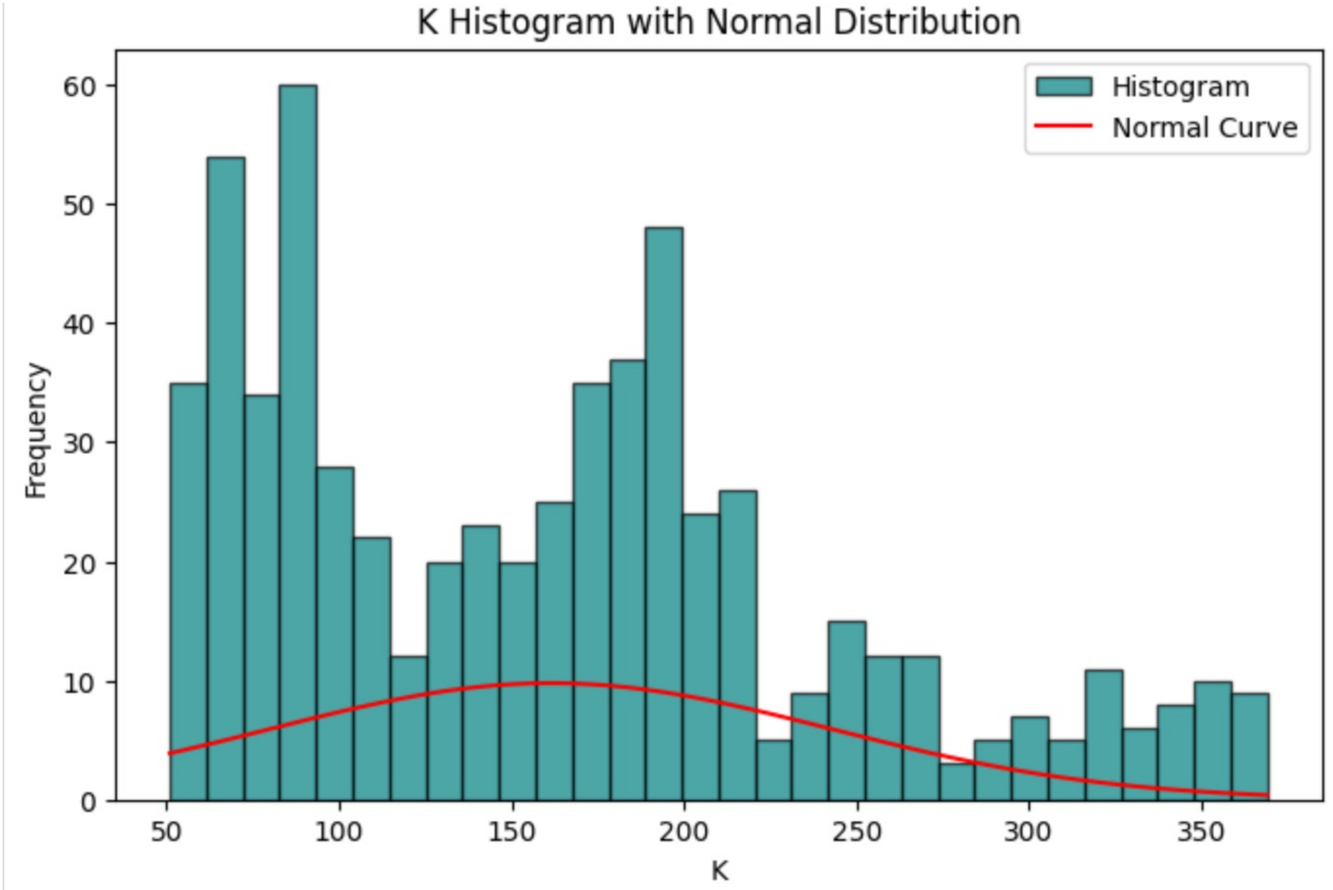
```
df['N_percent'] = (df['N'] / (df['N'] + df['P'] + df['K'])) * 100
df['P_percent'] = (df['P'] / (df['N'] + df['P'] + df['K'])) * 100
df['K_percent'] = (df['K'] / (df['N'] + df['P'] + df['K'])) * 100
df[['N_percent', 'P_percent', 'K_percent']].head()
```

	N_percent	P_percent	K_percent
0	33.333333	16.083916	50.582751
1	40.284360	8.530806	51.184834
2	35.665914	14.898420	49.435666
3	34.545455	11.688312	53.766234
4	33.165829	12.060302	54.773869

Visualizations

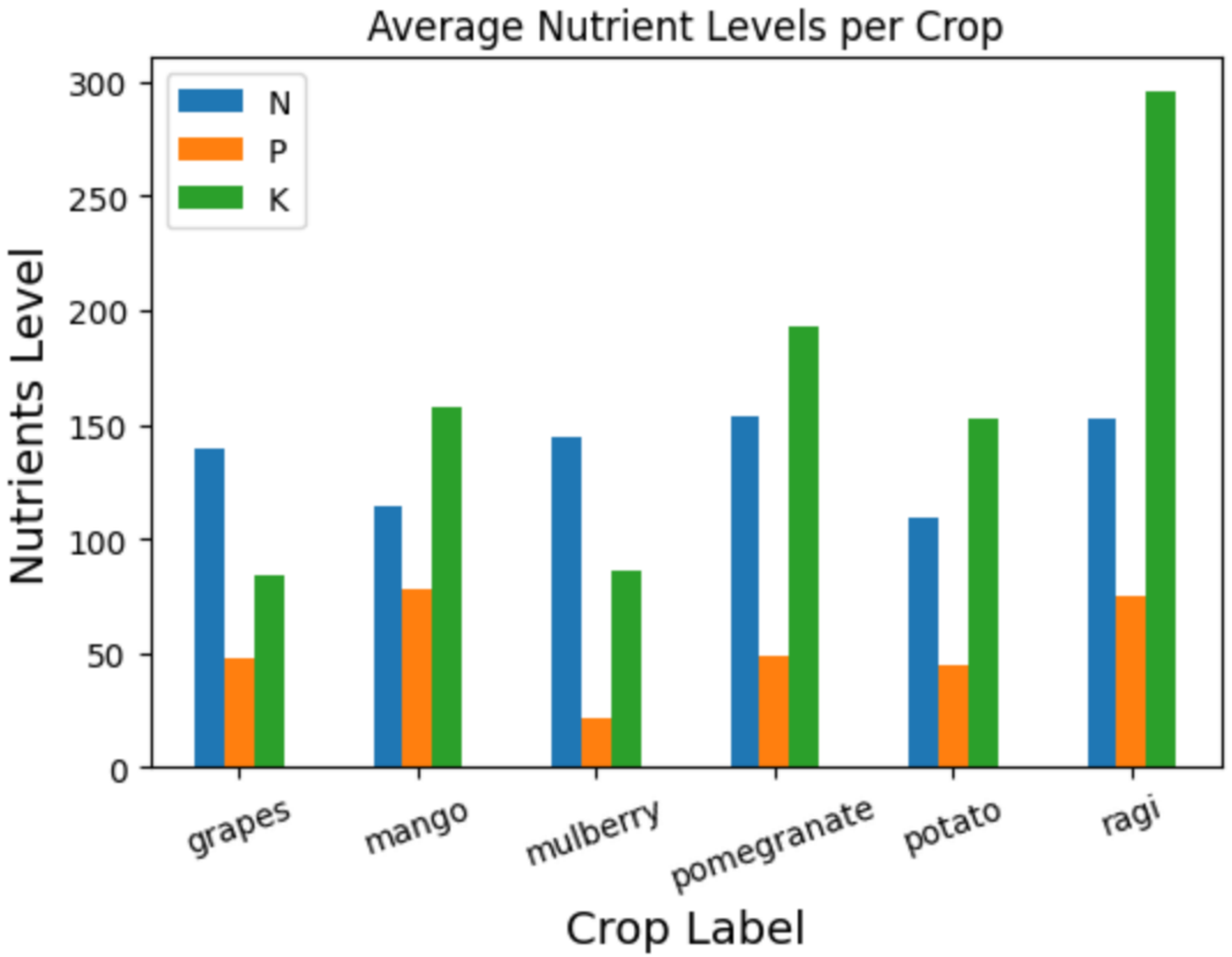




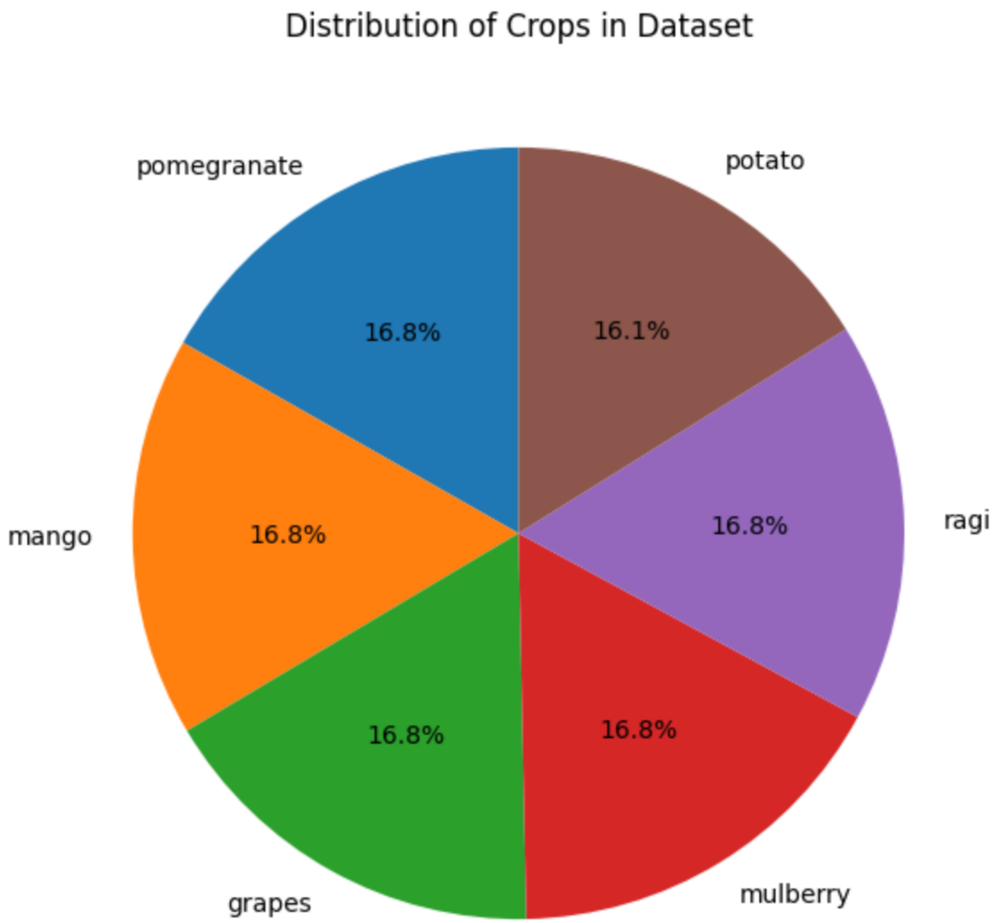


Visualizations

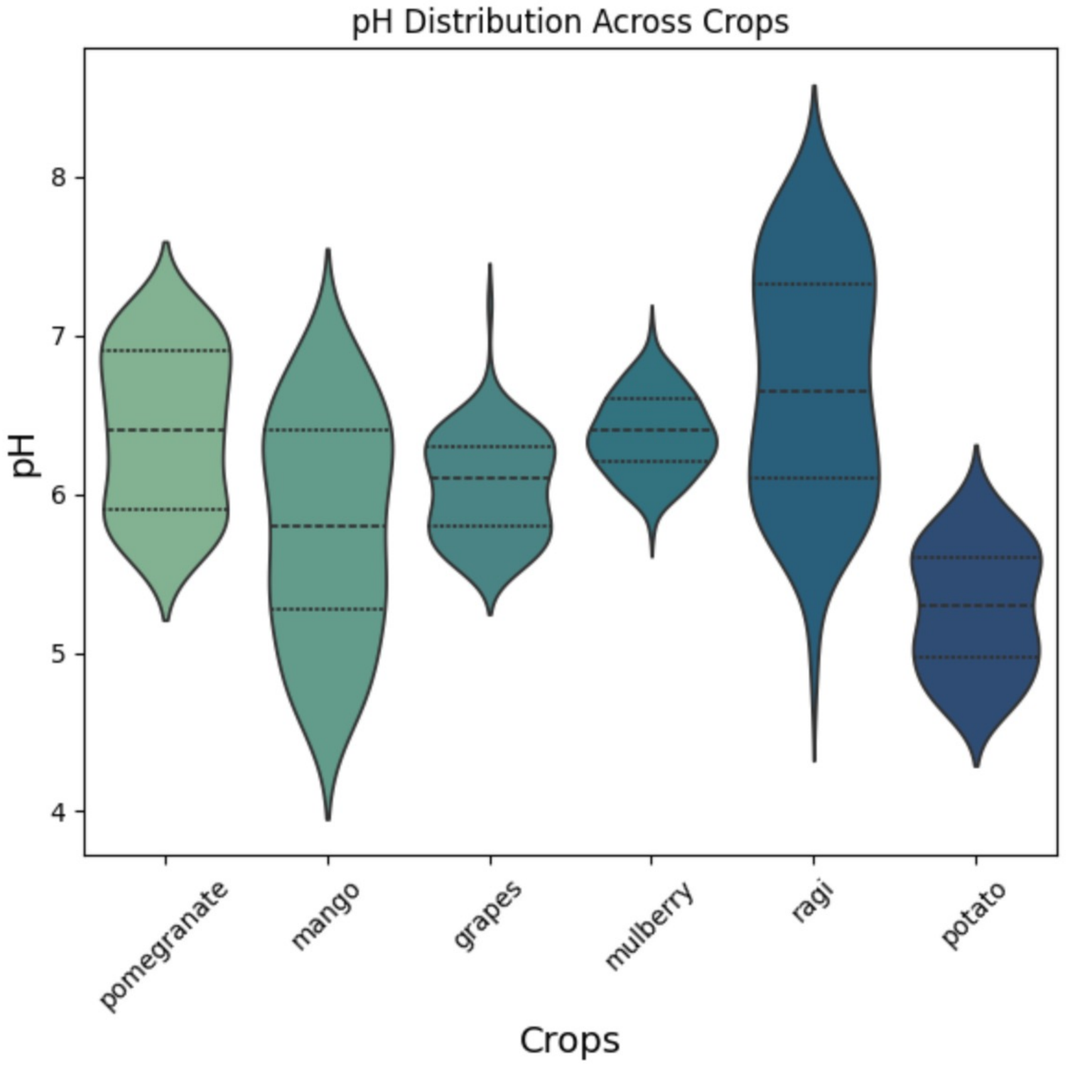
Visualizations



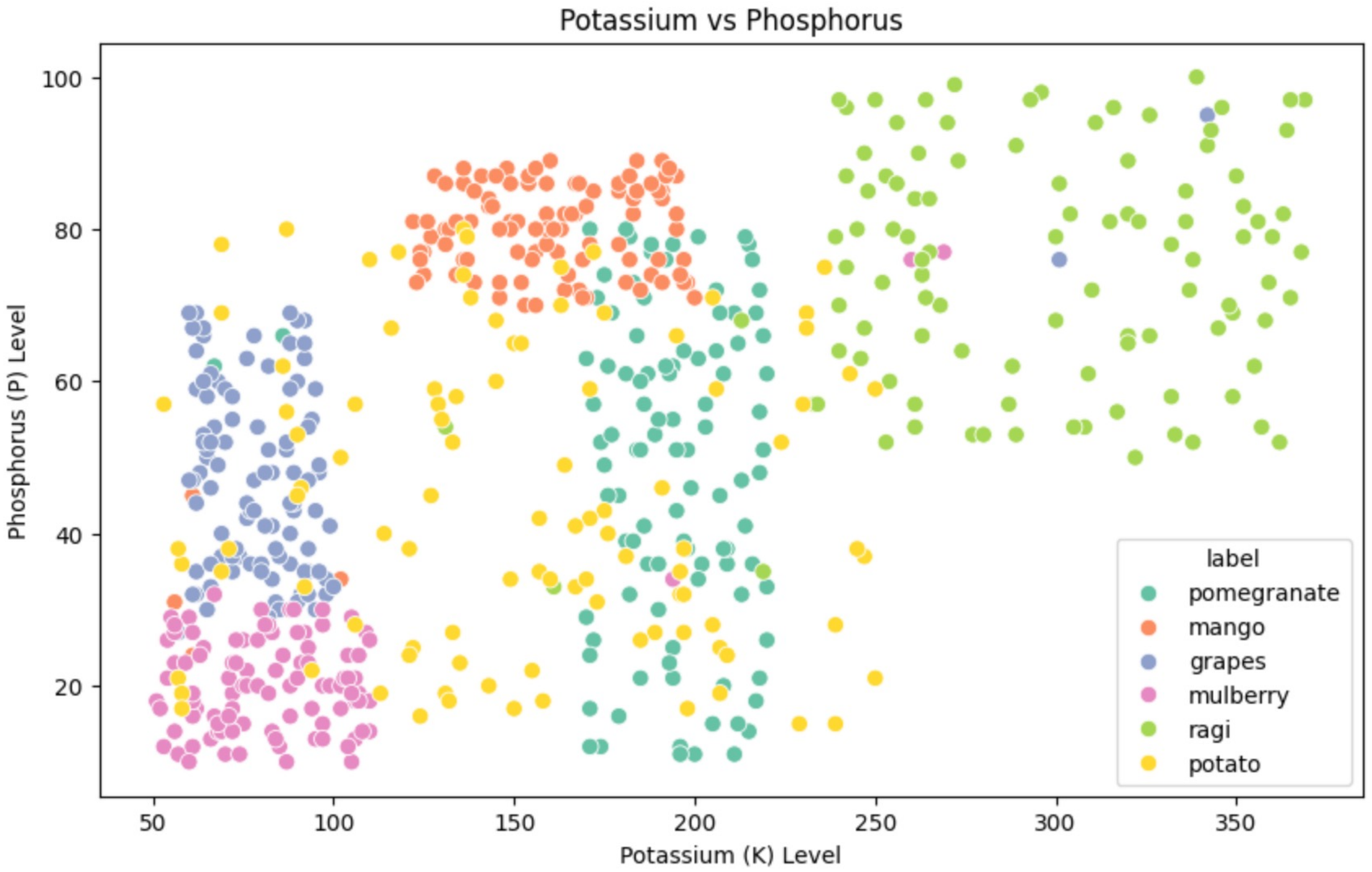
Visualizations



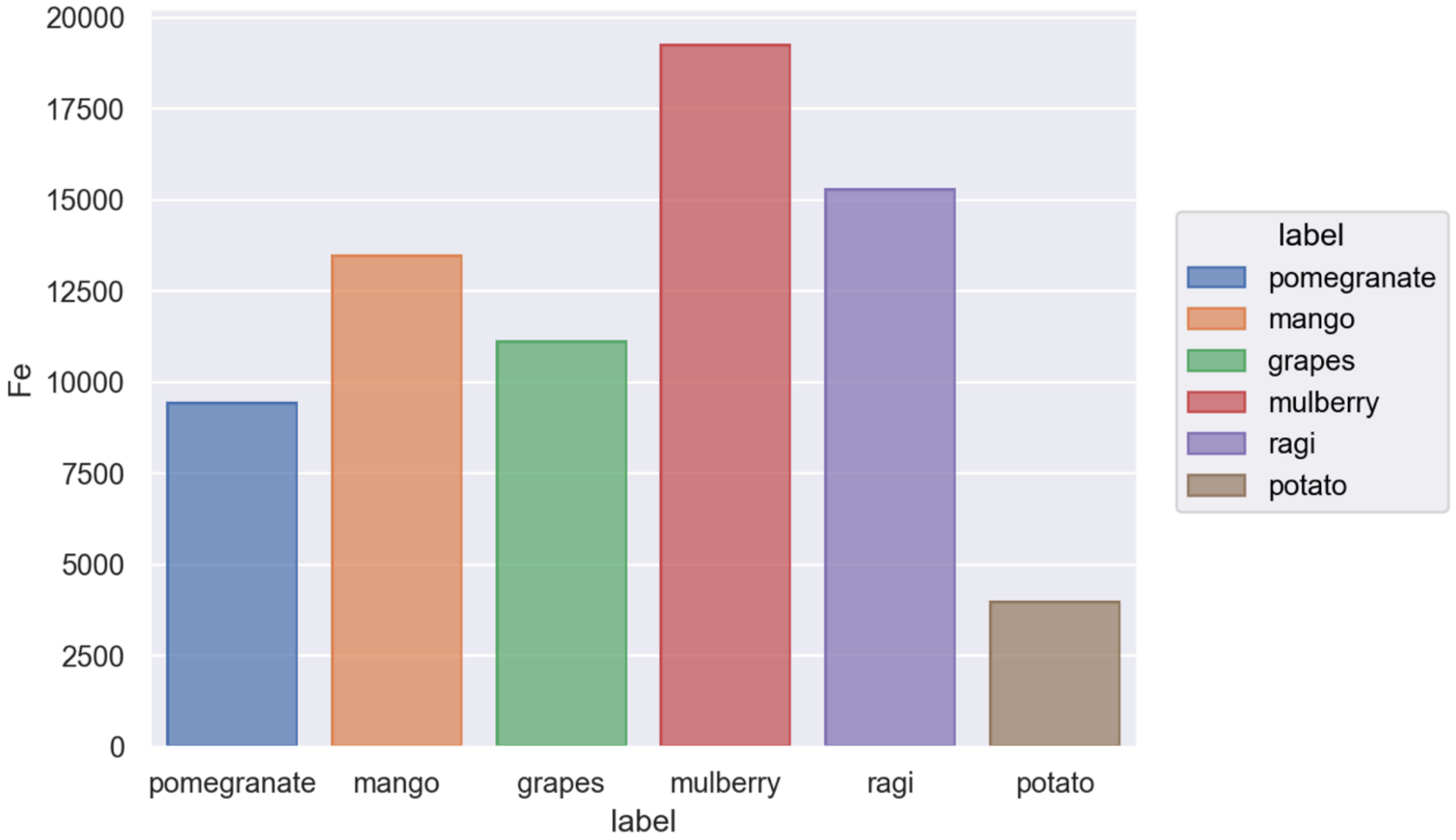
Visualizations



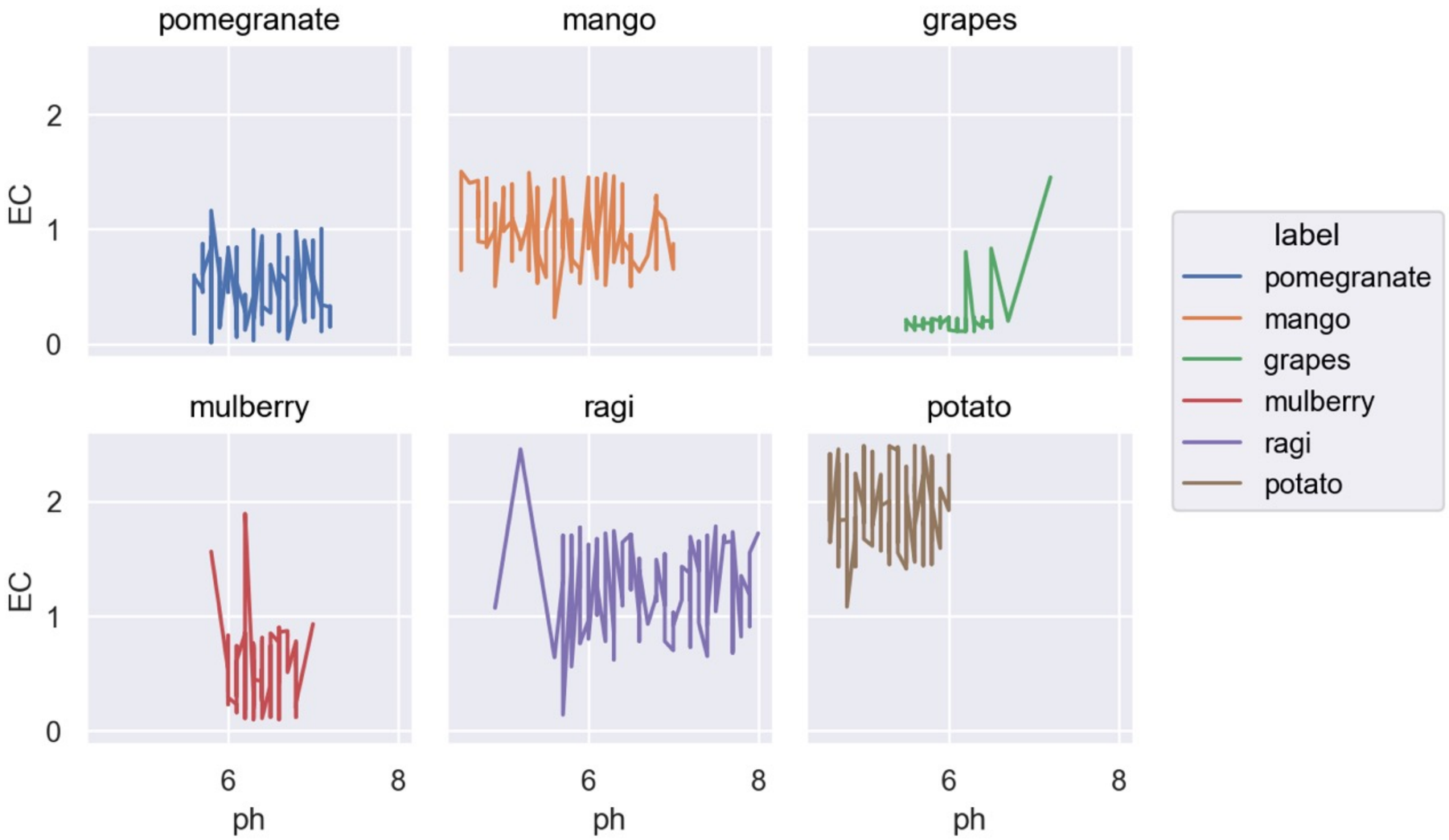
Visualizations



Visualizations



Visualizations



Label Encoding

```
from sklearn.preprocessing import LabelEncoder,StandardScaler
le = LabelEncoder()
df['label_encoded'] = le.fit_transform(df['label'])
print(df[['label', 'label_encoded']].head(10))
```

```
          label  label_encoded
0  pomegranate              3
1  pomegranate              3
2  pomegranate              3
3  pomegranate              3
4  pomegranate              3
5  pomegranate              3
6  pomegranate              3
7  pomegranate              3
8  pomegranate              3
9  pomegranate              3
```

Model Implementation

Logistic Regression ¶

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
X=df[['N', 'P', 'K', 'ph', 'Cu', 'Fe', 'Mn', 'Zn']]
y=df['label_encoded']
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_train,X_test,y_train,y_test = train_test_split(X_scaled,y,stratify=y,test_size=0.20,random_state=42)
y_train
```

```
56     3
149    1
377    2
458    5
571    4
      ..
433    5
491    5
135    1
257    0
185    1
Name: label_encoded, Length: 496, dtype: int64
```

Model Implementation

```
y_pred=clf.predict(X_test)
y_pred
```

```
array([3, 0, 3, 4, 4, 5, 5, 5, 5, 0, 0, 5, 2, 4, 0, 4, 3, 5, 0, 3, 0, 2,
       0, 0, 2, 5, 3, 5, 3, 0, 4, 1, 0, 2, 5, 2, 5, 4, 3, 0, 3, 5, 3, 1,
       3, 1, 5, 4, 5, 1, 2, 0, 3, 3, 3, 1, 4, 4, 2, 2, 2, 5, 0, 1, 1, 0,
       2, 1, 1, 4, 1, 4, 1, 4, 2, 3, 5, 2, 3, 2, 4, 0, 5, 2, 5, 3, 4, 2,
       4, 2, 1, 4, 0, 2, 0, 1, 0, 4, 1, 1, 3, 1, 2, 0, 1, 4, 5, 3, 4, 0,
       3, 5, 4, 5, 3, 2, 2, 5, 5, 3, 4, 0, 2, 5])
```

```
log = pd.DataFrame({'y_pred':y_pred,
                    'y_true':y_test})
log.head(10)
```

	y_pred	y_true
49	3	3
584	0	4
82	3	3
305	4	0
109	4	1
608	5	4
268	5	0
353	5	2
328	5	2
533	0	4

```
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{cm}")
```

```
Accuracy: 0.9354838709677419
Confusion Matrix:
[[20  0  1  0  0  0]
 [ 0 17  1  0  2  0]
 [ 0  0 19  0  0  2]
 [ 1  0  0 20  0  0]
 [ 0  0  0  1 19  0]
 [ 0  0  0  0  0 21]]
```

	Original Label	Encoded Value
0	grapes	0
1	mango	1
2	mulberry	2
3	pomegranate	3
4	potato	4
5	ragi	5

Model Implementation

```
tp=cm1[1,1]
tn=cm1[0,0]
fp=cm1[0,1]
fn=cm1[1,0]
fn
```

```
0
```

```
tp
```

```
17
```

```
tn
```

```
20
```

```
fp
```

```
0
```

```
# recall (sensitivity)
recall = tp/(tp+fn)
recall
```

```
1.0
```

```
precision=tp/(tp+fp)
precision
```

```
1.0
```

```
false_alarm_rate = fp/(fp+tn)
false_alarm_rate
```

```
0.0
```

```
print(metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.95      0.95        21
           1       1.00      0.85      0.92        20
           2       0.90      0.90      0.90        21
           3       0.95      0.95      0.95        21
           4       0.90      0.95      0.93        20
           5       0.91      1.00      0.95        21

    accuracy                           0.94       124
   macro avg       0.94      0.93      0.93       124
weighted avg       0.94      0.94      0.94       124
```

Model Implementation

```
y_prob = clf.predict_proba(X_test)
y_prob
```

```
array([[7.92653349e-02, 1.21031735e-03, 3.04181619e-01, 5.22032161e-01,
        3.25799422e-02, 6.07306262e-02],
       [8.04202693e-01, 5.91748454e-02, 8.30673503e-04, 1.33751602e-01,
        1.21238884e-03, 8.27797480e-04],
       [5.94022365e-02, 2.76634220e-03, 2.20210067e-04, 9.29238713e-01,
        5.50115720e-04, 7.82238285e-03],
       [1.05284943e-03, 1.18766130e-03, 1.62502166e-02, 9.06225293e-03,
        9.60574489e-01, 1.18725308e-02],
       [4.34262065e-03, 1.39661362e-02, 6.14900162e-03, 4.11472693e-03,
        9.67102937e-01, 4.32457722e-03],
       [3.24887906e-05, 1.09356791e-04, 3.73354010e-04, 1.21064816e-01,
        2.50061249e-04, 8.78169923e-01],
       [2.49567778e-07, 2.63123467e-03, 5.56360861e-07, 1.40331226e-03,
        3.46288860e-06, 9.95961184e-01],
       [3.61118541e-05, 9.36322118e-03, 2.23689196e-04, 1.93926412e-02,
        9.56250912e-08, 9.70984241e-01],
       [1.33692363e-06, 7.21724216e-04, 2.18659018e-06, 1.18313533e-02,
        3.14500161e-04. 9.87128899e-01].
```

Model Implementation

```
# Cross fold validation
from sklearn.model_selection import KFold, cross_val_score, RepeatedKFold
lr1 = LogisticRegression(solver='liblinear', max_iter=10000, random_state=42)
cv = RepeatedKFold(n_splits=5, random_state=456, n_repeats=10)
accuracy_results = cross_val_score(lr1, X, y, cv=cv, scoring='accuracy')
```

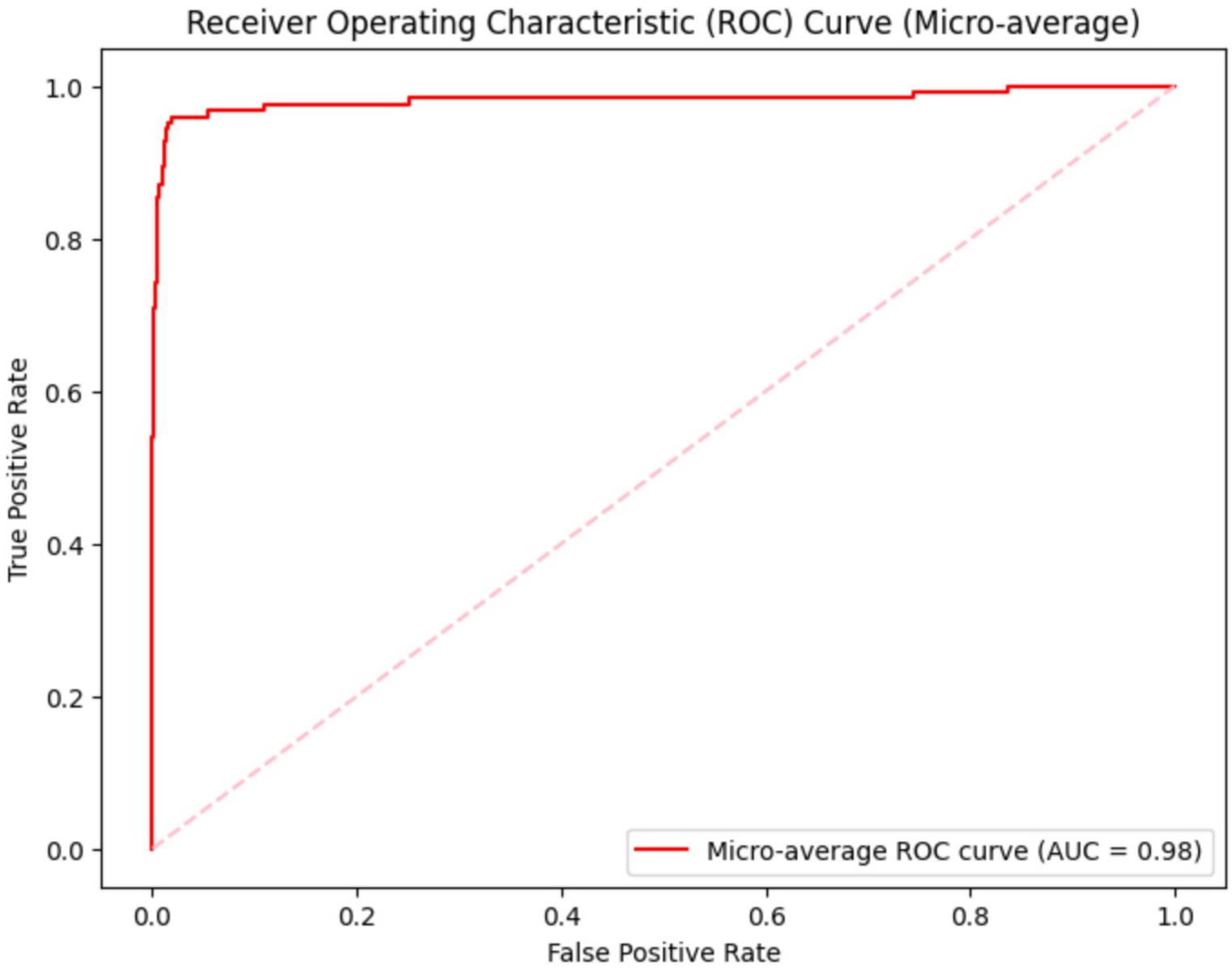
```
lr1
```

```
LogisticRegression
LogisticRegression(max_iter=10000, random_state=42, solver='liblinear')
```

```
# cross fold validation
mean_accuracy = accuracy_results.mean()
std_accuracy = accuracy_results.std()

print(f"Mean Accuracy: {mean_accuracy:.4f}")
print(f"Standard Deviation of Accuracy: {std_accuracy:.4f}")
```

```
Mean Accuracy: 0.9315
Standard Deviation of Accuracy: 0.0206
```

Model Implementation



Model Implementation

Ridge and Lasso Regression

Logistic Regression with L1 Regularization:

	precision	recall	f1-score	support
0	0.95	0.95	0.95	21
1	1.00	0.90	0.95	20
2	0.90	0.90	0.90	21
3	1.00	0.95	0.98	21
4	0.95	1.00	0.98	20
5	0.91	1.00	0.95	21
accuracy			0.95	124
macro avg	0.95	0.95	0.95	124
weighted avg	0.95	0.95	0.95	124

Logistic Regression with L2 Regularization:

	precision	recall	f1-score	support
0	0.95	0.95	0.95	21
1	1.00	0.85	0.92	20
2	0.90	0.90	0.90	21
3	0.95	0.95	0.95	21
4	0.90	0.95	0.93	20
5	0.91	1.00	0.95	21
accuracy			0.94	124
macro avg	0.94	0.93	0.93	124
weighted avg	0.94	0.94	0.94	124

Model Implementation

SMOTE (Synthetic Minority Oversampling Technique)

```
# Handle class imbalance using SMOTE
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

print("After applying SMOTE:")
print(f"Class distribution: {dict(zip(*np.unique(y_resampled, return_counts=True)))}")
```

```
After applying SMOTE:
Class distribution: {0: 84, 1: 84, 2: 84, 3: 84, 4: 84, 5: 84}
```

```
# Dimensionality reduction using PCA
from sklearn.decomposition import PCA

pca = PCA(n_components=0.95)  # Retain 95% of variance
X_pca = pca.fit_transform(X_train)

print(f"Original feature dimensions: {X_train.shape[1]}, Reduced dimensions: {X_pca.shape[1]}")
```

```
Original feature dimensions: 8, Reduced dimensions: 7
```

Model Implementation

Other Classifications and Ensembing Technique

```python
# Adding new classifiers and evaluating
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

# Initialize classifiers
rf_clf = RandomForestClassifier(random_state=42)
dt_clf = DecisionTreeClassifier(random_state=42)
svm_clf = SVC(probability=True, random_state=42)

# Fit and evaluate each classifier
for clf, name in [(rf_clf, "Random Forest"), (dt_clf, "Decision Tree"), (svm_clf, "SVM"),(log_reg_l1,"Logistic Regression(L1)")]:
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"{name} Classifier:")
    print(classification_report(y_test, y_pred))
```

Random Forest Classifier:

	precision	recall	f1-score	support
0	0.95	0.95	0.95	21
1	1.00	0.95	0.97	20
2	0.90	0.90	0.90	21
3	1.00	0.95	0.98	21
4	1.00	1.00	1.00	20
5	0.91	1.00	0.95	21
accuracy			0.96	124
macro avg	0.96	0.96	0.96	124
weighted avg	0.96	0.96	0.96	124

Decision Tree Classifier:

	precision	recall	f1-score	support
0	0.91	0.95	0.93	21
1	0.95	0.95	0.95	20
2	0.89	0.81	0.85	21
3	1.00	0.95	0.98	21
4	1.00	1.00	1.00	20
5	0.91	1.00	0.95	21
accuracy			0.94	124
macro avg	0.94	0.94	0.94	124
weighted avg	0.94	0.94	0.94	124

SVM Classifier:

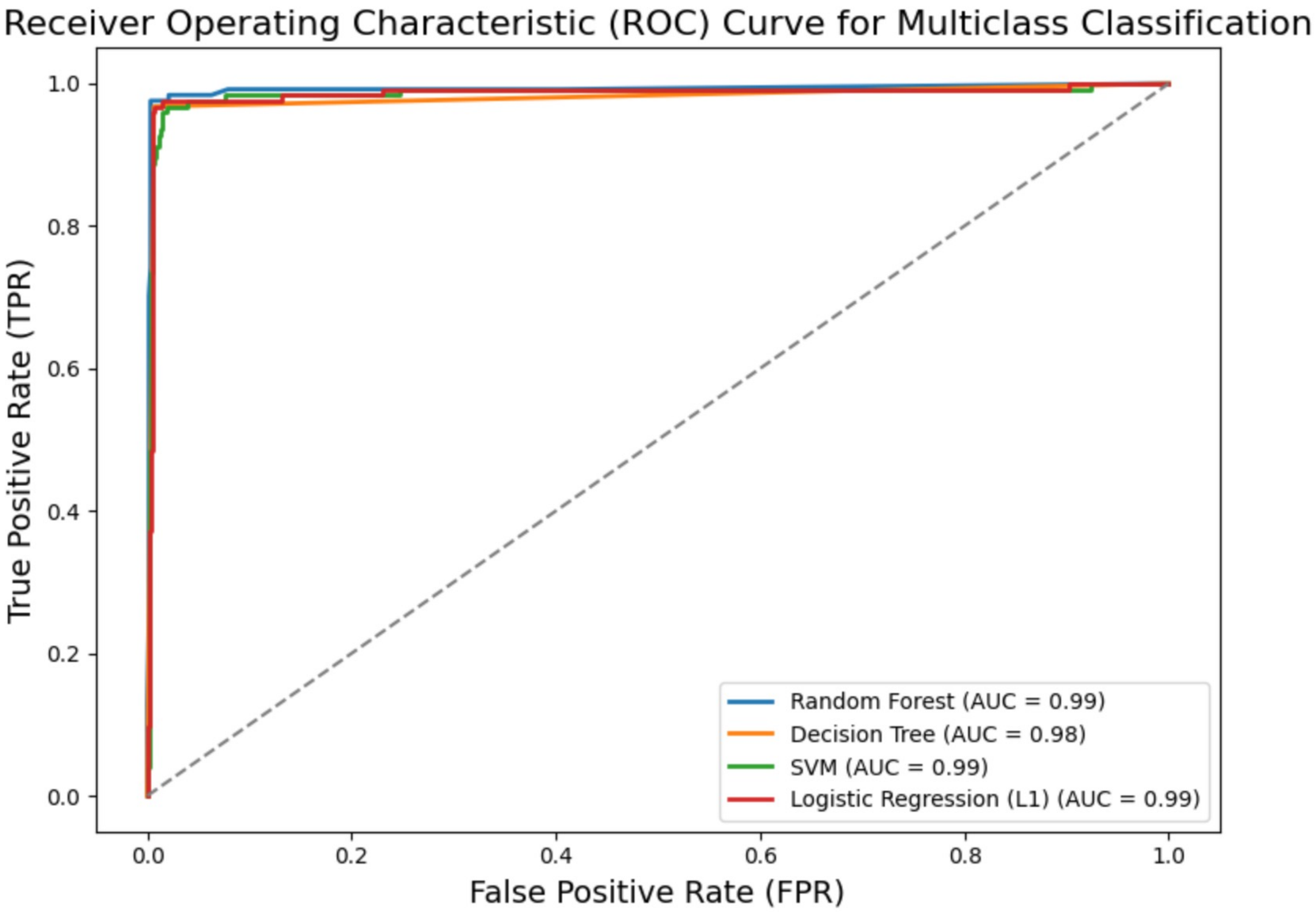
	precision	recall	f1-score	support
0	0.95	0.95	0.95	21
1	1.00	0.95	0.97	20
2	0.90	0.90	0.90	21
3	1.00	0.95	0.98	21
4	1.00	1.00	1.00	20
5	0.91	1.00	0.95	21
accuracy			0.96	124
macro avg	0.96	0.96	0.96	124
weighted avg	0.96	0.96	0.96	124

Model Implementation

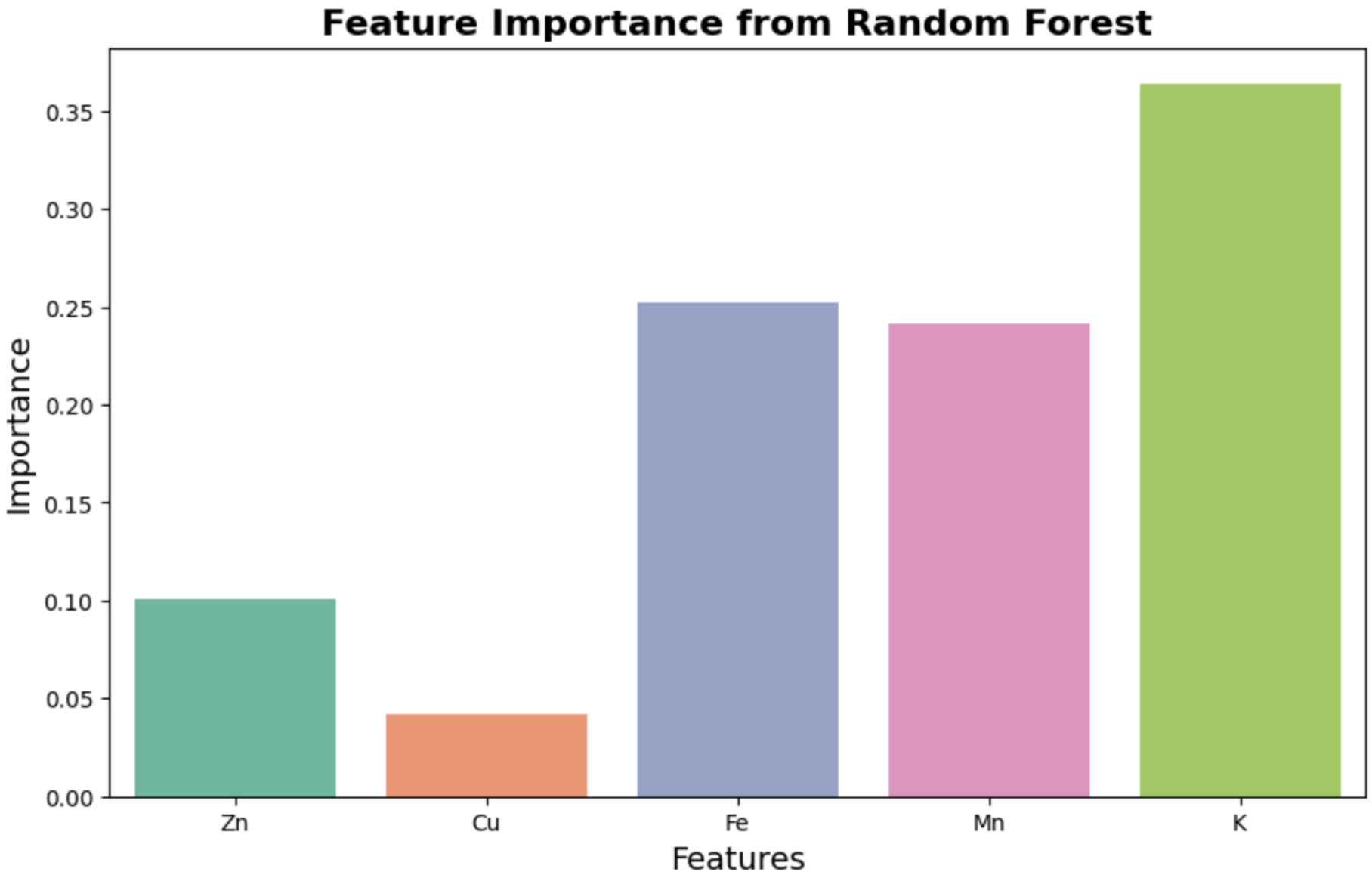
Comparison of Classifiers Based on Key Metrics

	Accuracy	Precision (Macro)	Recall (Macro)	F1-Score (Macro)
Random Forest	0.96	0.96	0.96	0.96
Decision Tree	0.94	0.94	0.94	0.94
SVM	0.96	0.96	0.96	0.96
Logistic Regression (L1)	0.95	0.95	0.95	0.95

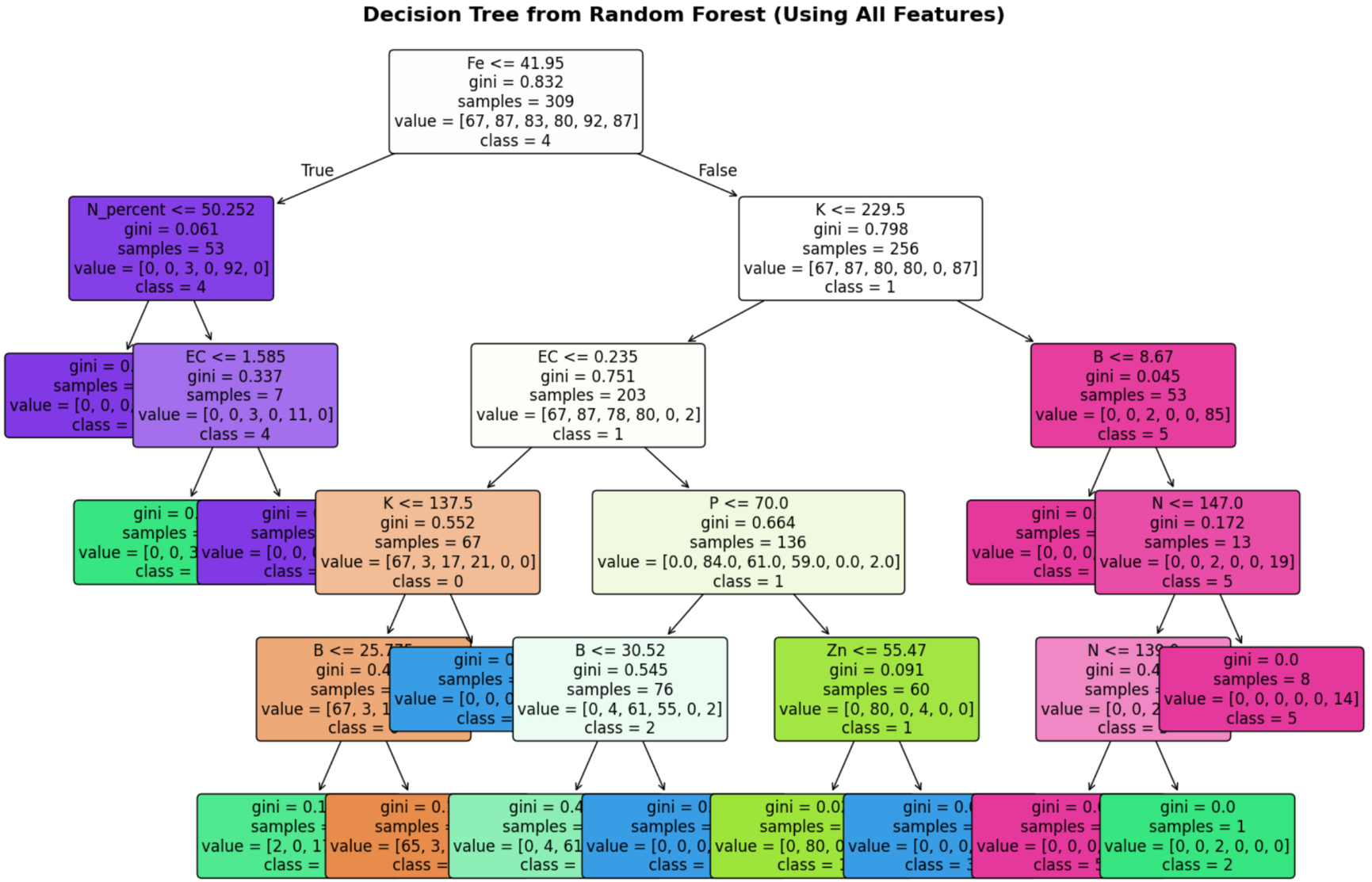
Model Implementation



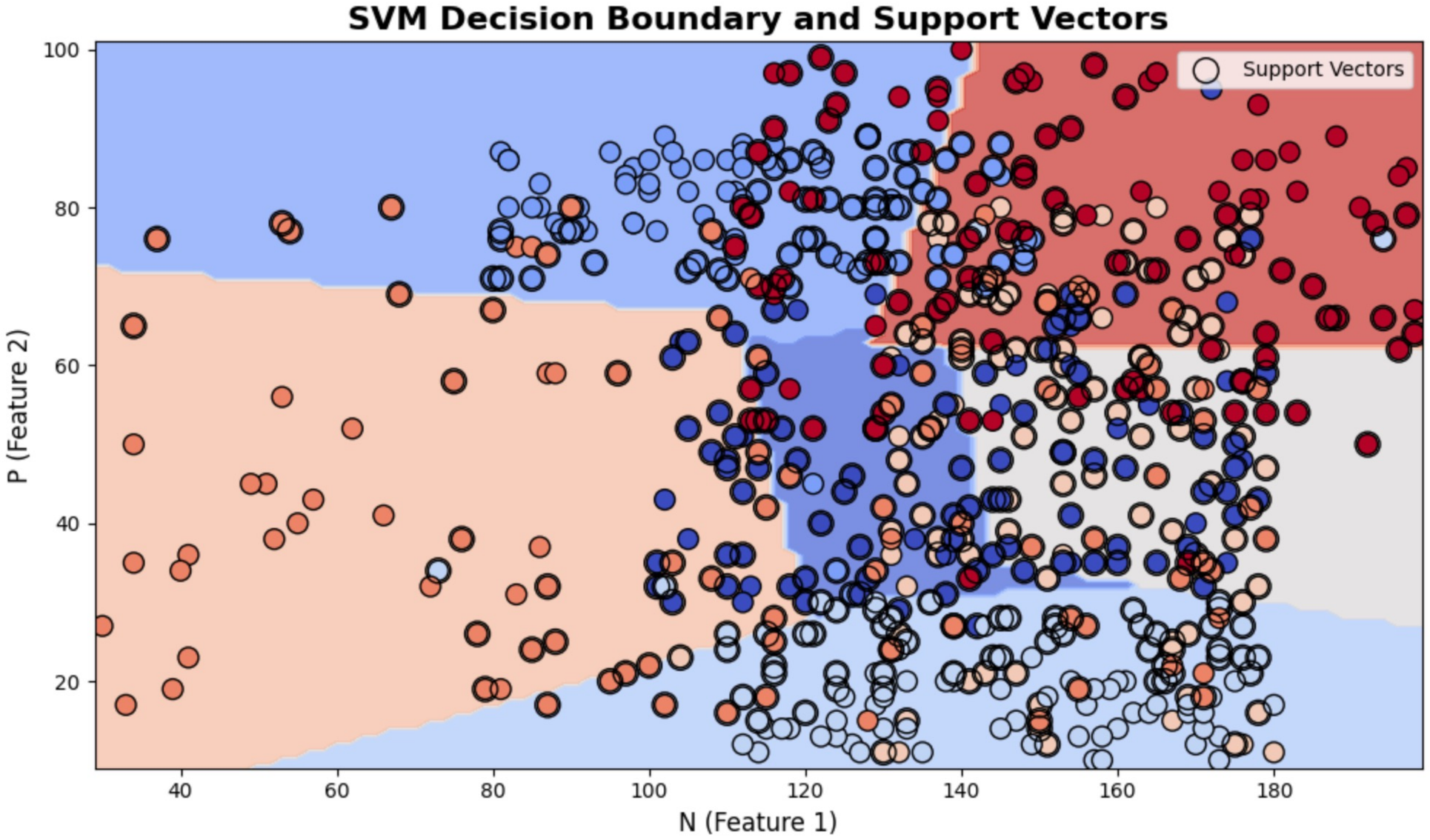
Model Implementation



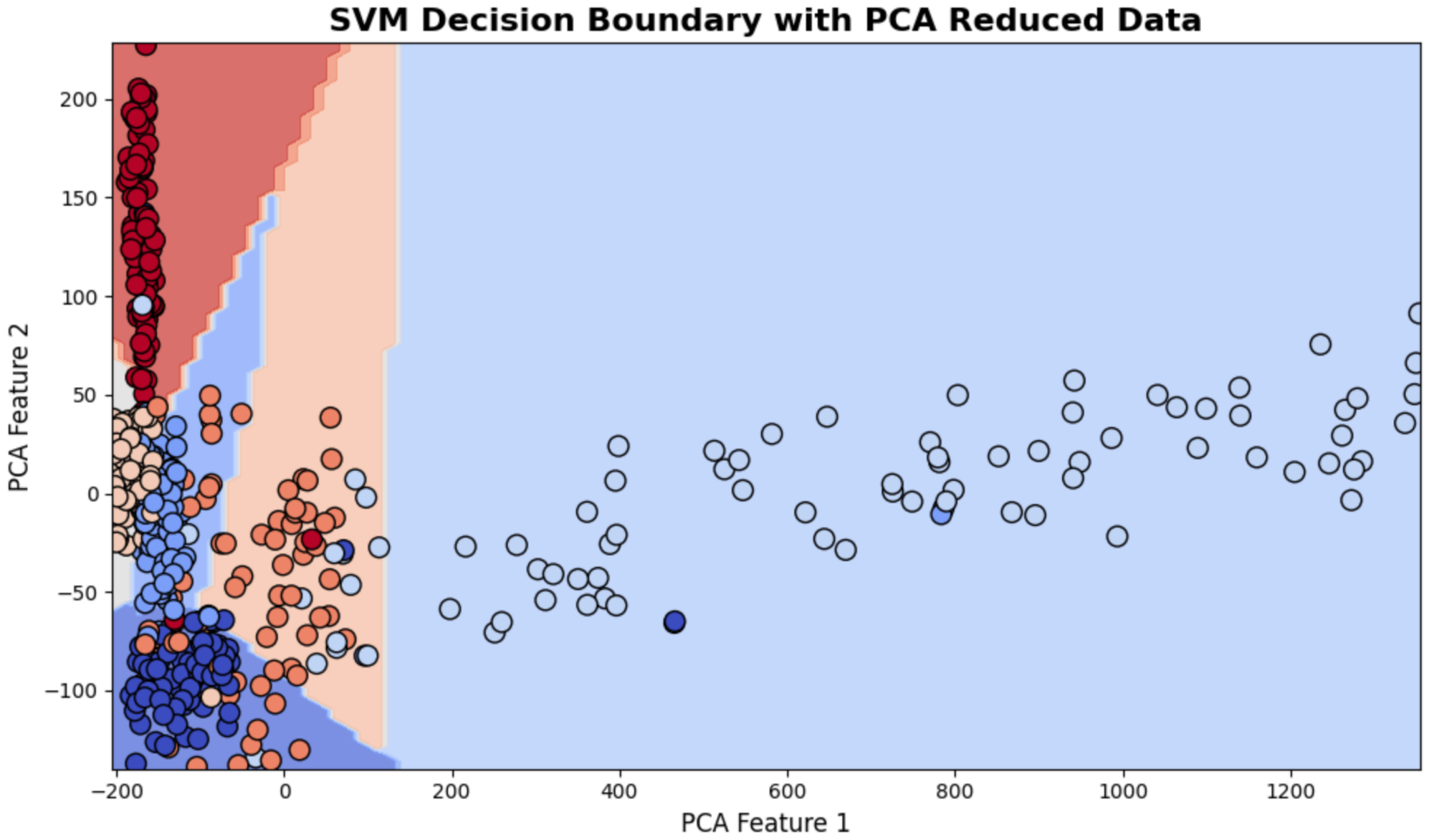
Model Implementation



Model Implementation



Model Implementation



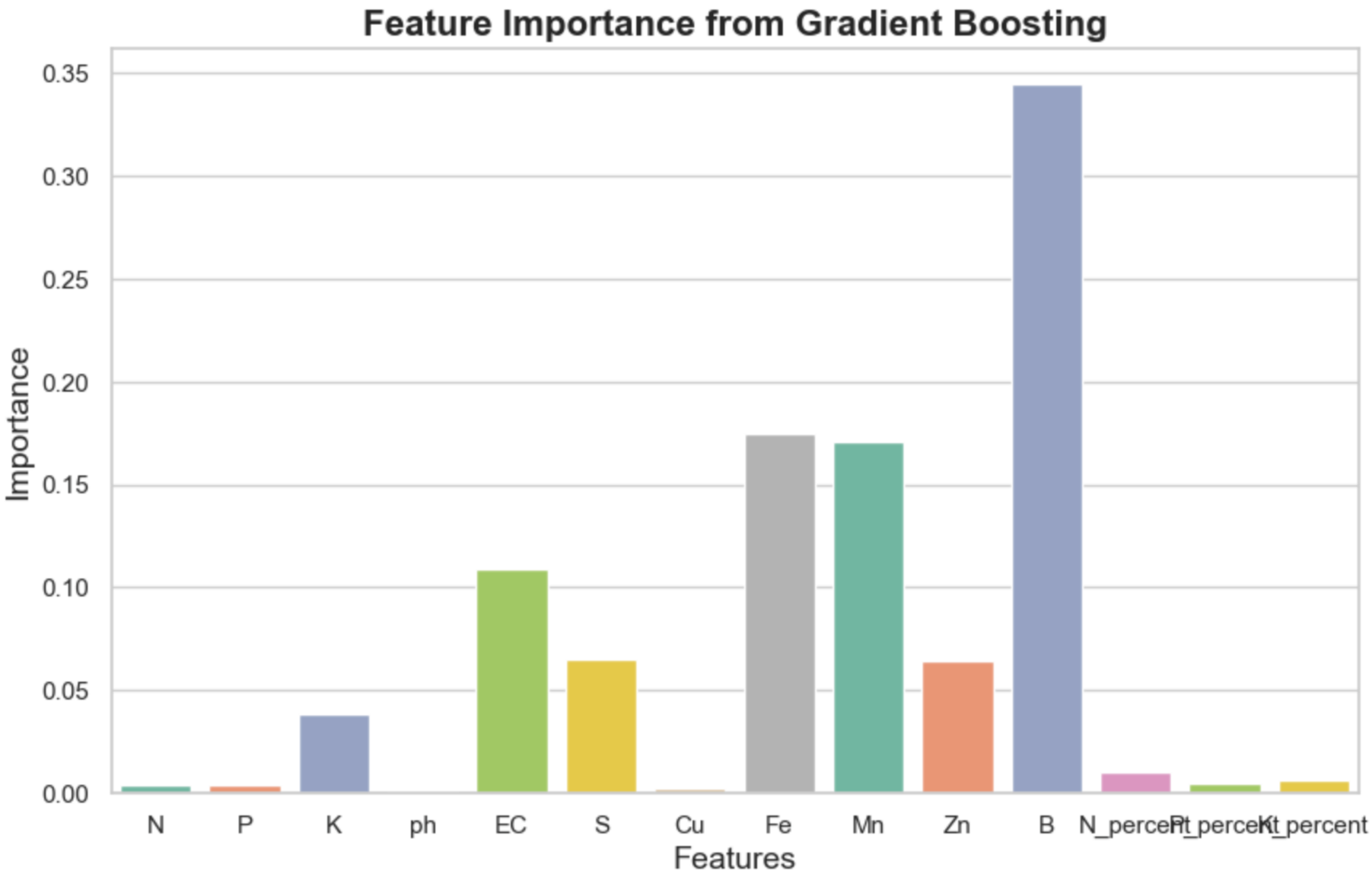
Model Implementation

Boosting Techniques

```python
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report

# Initialize the Gradient Boosting model
gb_clf = GradientBoostingClassifier(n_estimators=100, random_state=42)

# Fit the model to the training data
gb_clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred_gb = gb_clf.predict(X_test)

# Print classification report
print("Gradient Boosting Classifier:")
print(classification_report(y_test, y_pred_gb))
```

```
Gradient Boosting Classifier:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        24
           1       0.94      0.94      0.94        18
           2       1.00      1.00      1.00        21
           3       1.00      0.95      0.98        22
           4       1.00      1.00      1.00        23
           5       0.94      1.00      0.97        16

    accuracy                           0.98       124
   macro avg       0.97      0.98      0.97       124
weighted avg       0.98      0.98      0.98       124
```

Model Implementation

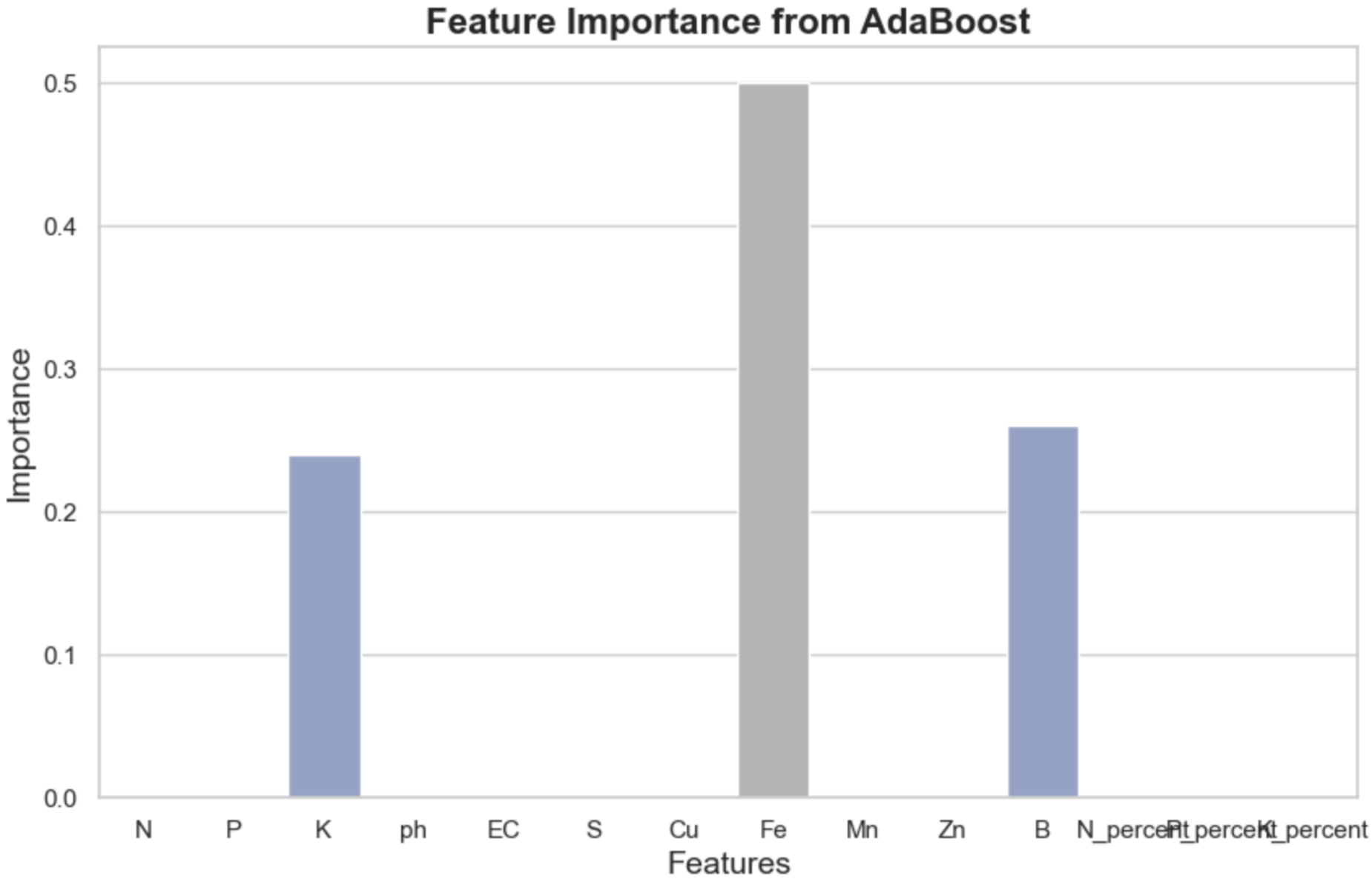


Model Implementation

```python
from sklearn.ensemble import AdaBoostClassifier

# Initialize the AdaBoost model with a base estimator
ada_clf = AdaBoostClassifier(n_estimators=50, random_state=42)

# Fit the AdaBoost model to the training data
ada_clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred_ada = ada_clf.predict(X_test)

# Print classification report
print("AdaBoost Classifier:")
print(classification_report(y_test, y_pred_ada))
```

```
AdaBoost Classifier:
              precision    recall  f1-score   support

           0       0.56      0.96      0.71        24
           1       1.00      0.06      0.11        18
           2       0.27      0.14      0.19        21
           3       0.40      1.00      0.57        22
           4       1.00      0.70      0.82        23
           5       0.00      0.00      0.00        16

    accuracy                           0.52       124
   macro avg       0.54      0.48      0.40       124
weighted avg       0.56      0.52      0.44       124
```

Model Implementation

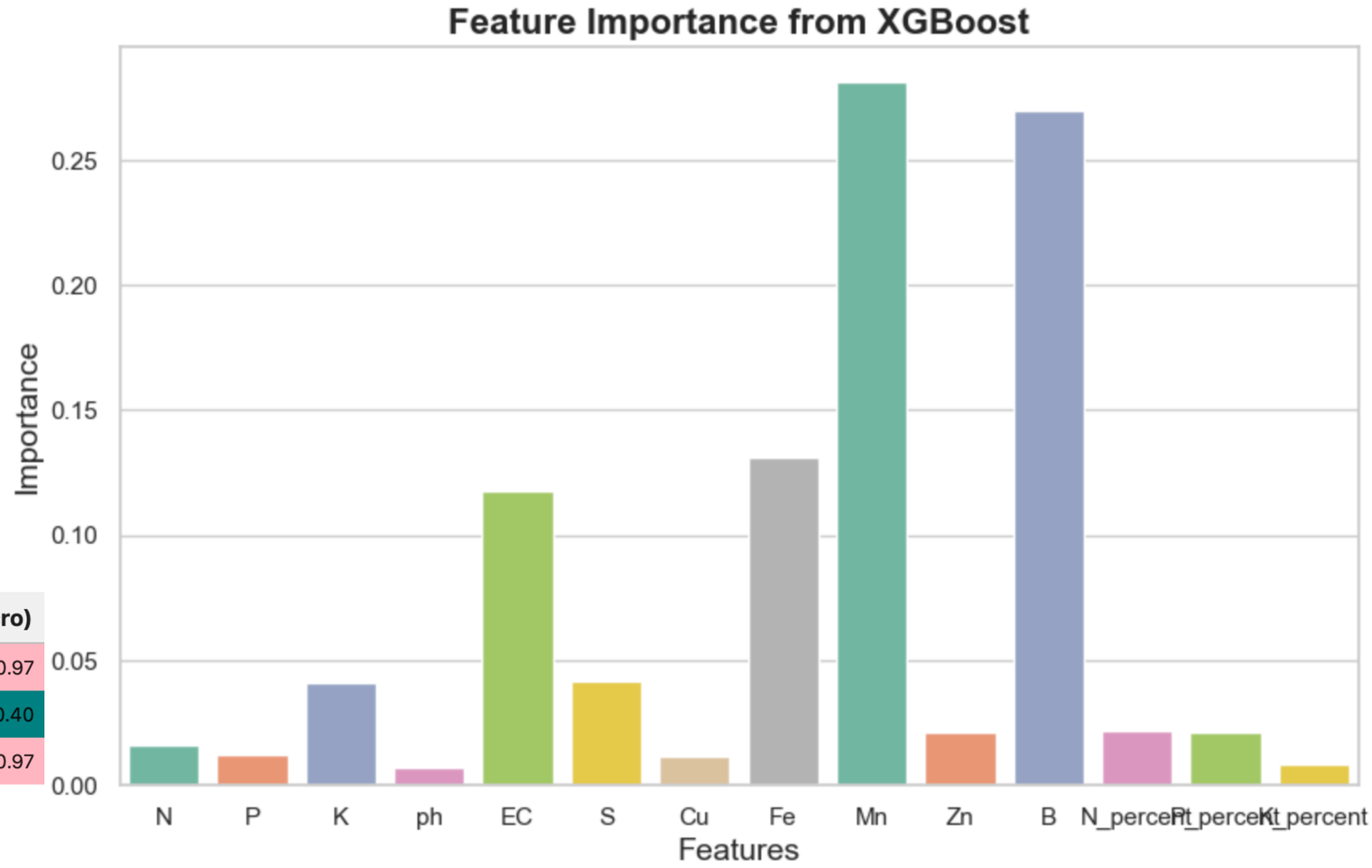


Model Implementation

```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        24
           1       0.94      0.94      0.94        18
           2       1.00      1.00      1.00        21
           3       1.00      0.95      0.98        22
           4       1.00      1.00      1.00        23
           5       0.94      1.00      0.97        16

    accuracy                           0.98       124
   macro avg       0.97      0.98      0.97       124
weighted avg       0.98      0.98      0.98       124
```

Comparison of Boosting Techniques Based on Key Metrics

	Accuracy	Precision (Macro)	Recall (Macro)	F1-Score (Macro)
Gradient Boosting	0.98	0.97	0.98	0.97
AdaBoost	0.52	0.54	0.48	0.40
XGBoost	0.98	0.97	0.98	0.97



Future Enhancement

01 Incorporating external data, such as weather forecasts or satellite imagery, to provide more comprehensive recommendations

02 User-Friendly Interface: Develop a mobile or web application for farmers to input soil data and get actionable recommendations easily.

03 Geospatial Analysis: Use GIS tools to analyze spatial variations in soil data and recommend region-specific crops.

04 Localized Models: Develop region-specific models tailored to local climate, soil, and crop patterns for better recommendations.


Thank You