Import All Relevant Packages

This cell imports all the necessary libraries and modules for the notebook. This includes:

- `sqlite3`: For interacting with the SQLite database.
- `pandas`: For data manipulation and analysis, particularly with DataFrames.
- `sklearn`: The scikit-learn library for machine learning tasks, including model selection, preprocessing, and metrics.
- `seaborn` and `matplotlib.pyplot`: For data visualization.
- `google.colab.drive`: For mounting Google Drive to access files.
- Various modules from `sklearn` for specific machine learning algorithms (SVC, KNeighborsClassifier, GaussianNB, DecisionTreeClassifier) and techniques (train_test_split, StratifiedKFold, GridSearchCV, StandardScaler, OneHotEncoder, ColumnTransformer, Pipeline, SimpleImputer, classification_report, confusion_matrix, balanced_accuracy_score, make_scorer, f1_score).
- `imblearn.over_sampling.SMOTE` and `imblearn.pipeline.Pipeline`: For handling imbalanced datasets using oversampling techniques.
- `ipywidgets`: For creating interactive controls like dropdowns.
- `IPython.display`: For displaying rich output like widgets and plots.

Show code

Create Relevant Functions

This section contains functions created to perform various tasks, including:

1. **Open DB connections and select the table**: Functions to establish a connection to the SQLite database and select data from a specified table.
2. **Close the connection**: A function to close the database connection.
3. **Clean and Analyze Dataframe**: A function to perform basic data cleaning and display descriptive statistics, missing values, and duplicate rows.
4. **Plotting**: Code to generate plots for visualizing the distribution of missing values in selected columns.
5. **Relevant Engineering functions**: Functions related to data imputation, specifically imputing missing values using the median.

Show code

Explore the 'Training' Dataset

NOTE: Make sure to UPLOAD Assignment2025S2.sqlite file into Google Drive

Exploratory Data Analysis

This section focuses on exploring the 'Training' dataset loaded from the database. We will perform the following steps:

- **Data Cleaning and Analysis**: Use the `clean_and_analyze_data` function to get an overview of the dataset, including data types, missing values, and descriptive statistics.
- **Visualize Missing Values**: Plot the distribution of columns with missing values to understand their characteristics and inform imputation strategies.
- **Impute Missing Values**: Apply the `impute_missing_with_median` function to fill in missing values based on the analysis.
- **Analyze Class Distribution**: Examine the distribution of the target variable ('class') to check for class imbalance.

Show code

```
Mounted at /content/drive
Connection to Assignment2025S2.sqlite opened successfully.
Tables in the database:
train
test

Enter the name of the table you want to select data from ('test' or '
Query executed successfully on table 'train'.

Data from table 'train':
```

	index	NDVI	Leaf_Area_Index	Canopy_Height	Soil_Type	Veget
0	0	-1.048539	19.158530	-2.822333	Sandy	
1	1	3.484490	1.723999	2.049871	Silty	
2	2	-6.413664	-37.868479	6.349476	Silty	
3	3	-1.390974	-30.247635	-6.433968	Silty	
4	4	6.323038	-29.522216	-0.845040	Silty	
...	...	...	...	...	...	
4995	4995	0.841413	56.117397	-2.698657	Clay	
4996	4996	-2.387714	-16.217542	-1.375386	Loam	
4997	4997	-2.415384	-29.705887	2.932198	Clay	
4998	4998	-2.791797	24.870881	-0.689178	Peaty	
4999	4999	8.338894	8.607732	1.278791	Sandy	

5000 rows × 32 columns

```
Connection closed.
--- DataFrame Info ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 32 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   index                      5000 non-null   int64
 1   NDVI                       5000 non-null   float64
 2   Leaf_Area_Index            5000 non-null   float64
 3   Canopy_Height              5000 non-null   float64
 4   Soil_Type                  5000 non-null   object
 5   Vegetation_Cover_Percent   5000 non-null   float64
 6   Greenness_Index            4876 non-null   float64
 7   Soil_Moisture              5000 non-null   float64
 8   Soil_pH                    5000 non-null   float64
 9   Topsoil_Depth              5000 non-null   float64
 10  Cultural_Burn              5000 non-null   int64
 11  Soil_Nitrogen_Level        5000 non-null   float64
 12  Burn_Season                5000 non-null   object
 13  Flood_Risk_Zone            5000 non-null   int64
 14  Invasive_Species_Presence  5000 non-null   int64
 15  Days_Since_Burn            3928 non-null   float64
 16  Surface_Water_Presence     5000 non-null   object
 17  Landform                   5000 non-null   object
 18  Fire_Intensity_Score       5000 non-null   float64
 19  Regrowth_Indicator         5000 non-null   float64
```

```
 19  Regrowth_Indicator          5000 non-null   float64
 20  Distance_to_Water_m         5000 non-null   float64
 21  Water_Availability_Index    5000 non-null   float64
 22  Water_Table_Depth           5000 non-null   float64
 23  Invasive_Species_Count      5000 non-null   float64
 24  Litter_Presence             5000 non-null   int64
 25  Distance_to_Road_m          5000 non-null   float64
 26  Elevation_m                 5000 non-null   float64
 27  Slope_Degree                5000 non-null   float64
 28  Latitude                    2120 non-null   float64
 29  Human_Disturbance           5000 non-null   object
 30  Longitude                   5000 non-null   float64
 31  class                       5000 non-null   object
dtypes: float64(21), int64(5), object(6)
memory usage: 1.2+ MB

--- DataFrame Description ---
```

	index	NDVI	Leaf_Area_Index	Canopy_Height	Vegetati
count	5000.000000	5000.000000	5000.000000	5000.000000	
mean	2499.500000	0.992832	-6.626767	0.025254	
std	1443.520003	4.227724	26.962660	2.985498	
min	0.000000	-14.493949	-104.263966	-10.256290	
25%	1249.750000	-1.836349	-25.416993	-2.015194	
50%	2499.500000	1.059246	-7.033786	-0.027194	
75%	3749.250000	3.857182	11.500324	2.030523	
max	4999.000000	15.470266	95.025008	11.090515	

8 rows × 26 columns

```
--- Missing Values Count ---
index                           0
NDVI                            0
Leaf_Area_Index                 0
Canopy_Height                   0
Soil_Type                       0
Vegetation_Cover_Percent        0
Greenness_Index               124
Soil_Moisture                   0
Soil_pH                         0
Topsoil_Depth                   0
Cultural_Burn                   0
Soil_Nitrogen_Level             0
Burn_Season                     0
Flood_Risk_Zone                 0
Invasive_Species_Presence       0
Days_Since_Burn              1072
Surface_Water_Presence          0
Landform                        0
Fire_Intensity_Score            0
Regrowth_Indicator              0
Distance_to_Water_m             0
Water_Availability_Index        0
Water_Table_Depth               0
Invasive_Species_Count          0
```

Show code

```
Slope_Degree                0
Latitude                 2880
Human_Disturbance           0
Longitude                   0
Class                       0
dtype: int64
```

Plotting Distribution of Missing Values

We will now plot the distribution for the missing values to decide on how to fill up the NA values in the train dataset.

Select colu... Greenness_Index

```
--- Description of Columns with Missing Values ---
```

	Greenness_Index	Days_Since_Burn	Latitude
count	4876.000000	3928.000000	2120.000000
mean	-18.981992	5.835748	0.031696
std	119.790773	17.908337	0.990361
min	-494.139620	-60.491257	-3.157445
25%	-100.473898	-6.610669	-0.650146
50%	-18.125356	5.773722	0.027603
75%	61.164149	17.579402	0.735886
max	479.690303	75.430177	3.276324

```
--- Duplicate Rows Count ---
There are no duplicate rows.
```

Missing Value Identification

Given that some columns with missing values have a high standard deviation and the distribution plots above appear close to normal, using the median to fill in the empty cells is the appropriate strategy.

Here's an extensive explanation:

1. **Robustness to Outliers:** The median is the middle value in a dataset when sorted. Unlike the mean, which is calculated by summing all values and dividing by the count, the median is not affected by extremely large or small values (outliers). The "close-to-normally distributed" data has some outliers, with more values the mean are pulled towards these extreme values, leading to a less representative imputation. Whereas by using a median, theere is a more stable measure of the central tendency in the presence of outliers.

2. **Less Distortion of Distribution:** Replacing missing values with the median tends to preserve the shape of the original distribution better than using the mean, especially if the distribution is not perfectly symmetrical. While the histograms might look roughly normal, they could have subtle skewness or tails that are better represented by the median.

3. **Suitable for Skewed Data:** When it is said to be a close-to-normally distributed data, there are tendencies for skewed data to exist in the feature. These are the impacts of the real-world outliers which makes it almost impossible for a perfectly normal distribution to appear. Results produced by mean on a dataset with high variance would just increase the skew, affecting the overall result. Whereas, median would just result to another count into the high density part of the distribution.

4. **Ease of Implementation:** Imputing with the median is computationally simple and easy to implement. It's a straightforward method that doesn't require complex modeling.

In summary: While the mean is often used for imputing missing values in normally distributed data, the median offers greater robustness to potential outliers or slight deviations from perfect normality. By using the median, you are less likely to introduce

distortion into the dataset due to extreme values, leading to a more reliable imputation. Additionally, considering some features contains high variance as represented by the big spread like over 18 in days after burn and up to 120 in the greenness index.

All Numeric Attribute Distribution

This section provides an overview of the distribution of all numerical attributes in the dataset after handling missing values. You can use the dropdown menu below to select a specific numerical column and visualize its distribution using a histogram and kernel density estimate (KDE) plot. This helps in understanding the spread, central tendency, and potential outliers of each numerical feature.

Show code

Plotting Distribution of Numerical Features

Select a numerical feature from the dropdown to visualize its distribution.

Select num… index

Data Preparation

Reasoning: This section outlines the steps for feature engineering and preprocessing the data for machine learning. This includes:

- Separating features and the target variable.
- Identifying numerical and categorical features.
- Creating preprocessing pipelines for scaling numerical features and one-hot encoding categorical features.
- Combining these steps into a ColumnTransformer for efficient preprocessing.

Model Definitions & Hyperparameter Grids

This section defines the machine learning models that will be trained and evaluated, along with the hyperparameter grids for each model. Hyperparameters are settings that are not learned from the data but are set before training. Tuning these hyperparameters is crucial for optimizing model performance.

For each model, the following are specified:

- **Model Estimator:** The specific scikit-learn classifier to be used (e.g., `KNeighborsClassifier`, `DecisionTreeClassifier`, `GaussianNB`, `LogisticRegression`).
- **Hyperparameter Grid:** A dictionary defining the range of values for the most important hyperparameters of the model. `GridSearchCV` will explore all possible combinations of these values during the tuning process to find the combination that yields the best performance.

Defining these models and their parameter grids separately makes the training and tuning process more organized and easier to manage, especially when evaluating multiple models.

Show code

```
Models defined:
- KNN: KNeighborsClassifier
- Decision Tree: DecisionTreeClassifier
- Naive Bayes: GaussianNB
- Logistic Regression: LogisticRegression
```

Dataset Class Imbalanced Overview

This section examines the distribution of the target variable ('class') in the dataset to identify if there is a class imbalance. Class imbalance occurs when the number of instances in one class is significantly lower than the number of instances in other classes. This can negatively impact the performance of machine learning models, particularly for the minority class.

The code in this section calculates and displays:

- **Value Counts:** The absolute number of samples belonging to each class.
- **Percentages:** The proportion of samples belonging to each class, expressed as a percentage.

By analyzing these counts and percentages, we can determine if the dataset is imbalanced and if techniques like SMOTE (Synthetic Minority Over-sampling Technique) are necessary to address the imbalance during the model training process.

Show code

```
Class Distribution (Value Counts):
class
Healthy     2489
Degraded    2003
AtRisk       508
```

```
Name: count, dtype: int64

Class Distribution (Percentages):
class
Healthy     49.78
Degraded    40.06
AtRisk      10.16
Name: proportion, dtype: float64

Class Imbalance Analysis:
Class distribution appears relatively balanced.
```

Analysis of Class Imbalance

Based on the class distribution:

- **Healthy:** Represents the largest class with 2489 samples (49.78%).
- **Degraded:** Is the second largest class with 2003 samples (40.06%).
- **AtRisk:** Is the smallest class with 508 samples (10.16%).

While the notebook's initial analysis states the distribution "appears relatively balanced," it's important to note that the 'AtRisk' class makes up only about 10% of the dataset. Compared to the 'Healthy' and 'Degraded' classes which are around 50% and 40% respectively, the 'AtRisk' class is significantly underrepresented.

This kind of distribution, where one or more classes have much fewer samples than others, is considered a **class imbalance**. While not as extreme as some datasets where the minority class might be less than 1%, a 10% representation for the 'AtRisk' class is still low enough that it can negatively impact the performance of standard machine learning models. Models trained on such data might be biased towards the majority classes and perform poorly at identifying the minority 'AtRisk' class, which is often the most critical class to predict correctly in real-world scenarios.

Therefore, techniques like SMOTE are appropriate and necessary to address this imbalance and help the models learn to better predict the 'AtRisk' class.

Feature Engineering & PreProcessing

Latitude and Longitude columns are being removed from the dataset. While geographical coordinates can sometimes be useful features, in this context, their raw values might not directly capture the underlying environmental factors influencing vegetation health.

Reasons for considering their removal include:

- **Lack of Direct Predictive Power:** Raw latitude and longitude might not have a simple linear or easily learnable relationship with the target variable ('class').

The important geographical information might be better captured by other features that are influenced by location (like Elevation, Soil Type, etc.).

- **Potential for Noise:** Without proper feature engineering (e.g., creating geographical clusters, using them to derive distance to specific points of interest), raw coordinates can sometimes add noise to the model.
- **Focus on Environmental Factors:** The dataset contains many other features that directly describe the environmental conditions (soil properties, vegetation indices, water availability, etc.) which are likely more relevant to vegetation health than just the location itself.

By removing Latitude and Longitude, we aim to simplify the model and focus on the features that are more likely to be directly predictive of the vegetation class.

Whereas, the index feature are only unique numbers representing each entry, thus irrelevant to the required measurements for prediction.

Show code

```
Feature matrix shape: (5000, 28)
Target vector shape: (5000,)
Target classes: ['Healthy' 'Degraded' 'AtRisk']

Numerical features (23): ['NDVI', 'Leaf_Area_Index', 'Canopy_Height',
Categorical features (5): ['Soil_Type', 'Burn_Season', 'Surface_Water_

Preprocessing pipeline created successfully!
```

Data Splitting

This section focuses on splitting the dataset into training and testing sets. This is a crucial step in the machine learning workflow to evaluate the performance of a model on unseen data.

- The data is split into features (`X`) and the target variable (`y`).
- The `train_test_split` function from `sklearn.model_selection` is used to perform the split.
- A common split ratio of 80% for training and 20% for testing is applied (`test_size=0.2`).
- `random_state` is set for reproducibility.
- `stratify=y` is used to ensure that the proportion of classes in the training and testing sets is the same as in the original dataset. This is particularly important for imbalanced datasets like this one to ensure that both the training and test sets are representative of the overall class distribution.

The resulting `X_train`, `X_test`, `y_train`, and `y_test` datasets will be used for model training, hyperparameter tuning, and final evaluation.

Show code

```
Training set size: 4000 samples
Test set size: 1000 samples

Class distribution before SMOTE:
class
AtRisk       406
Degraded    1603
Healthy     1991
Name: count, dtype: int64

Percentages:
class
AtRisk      10.15
Degraded    40.08
Healthy     49.78
Name: count, dtype: float64
```

Data Cleaning and Class Balancing

This section focuses on cleaning the loaded dataset by addressing missing values. The code in the following cell performs the following steps:

1. **Identify Missing Values:** It first identifies which columns in the `selected_data_df` DataFrame have missing values.
2. **User Confirmation:** It then prompts the user to confirm if they want to proceed with imputing (filling in) these missing values.
3. **Median Imputation:** If the user confirms, it calls the `impute_missing_with_median` function (defined earlier) to fill the missing values in the identified columns using the median value of each respective column. A copy of the DataFrame is used to avoid modifying the original `selected_data_df`.
4. **Verification:** After imputation, it prints the DataFrame's information (`.info()`) and the count of missing values (`.isnull().sum()`) for the newly created `selected_data_df_imputed` DataFrame to verify that the missing values have been handled.
5. **Skip Imputation:** If the user chooses not to confirm, the imputation step is skipped.

The strategy of using the median for imputation is chosen based on the earlier analysis of the data distributions and the potential presence of outliers, as discussed in the "Ext. Missing Value Identification" section.

Show code

```
The median values are:
Greenness_Index   -17.275689
Days_Since_Burn     5.964812
dtype: float64

Manually filling NaNs in X_train and X_test with medians from X_train.

No NaNs found in X_train_filled after manual fill.

No NaNs found in X_test_filled after manual fill.
```

SMOTE Balancing

This section applies the Synthetic Minority Over-sampling Technique (SMOTE) to address the class imbalance identified in the dataset. SMOTE works by creating synthetic samples for the minority class based on its existing instances.

The code in this section performs the following steps:

1. **Applies Preprocessing:** The previously defined preprocessing pipeline is applied to the training and testing data.
2. **Applies SMOTE:** SMOTE is applied to the *preprocessed training data* to generate synthetic samples for the minority 'AtRisk' class. This balances the class distribution in the training set.
3. **Displays Balanced Distribution:** The class distribution of the training set after applying SMOTE is displayed to confirm that the classes are now balanced.

By balancing the training data, we aim to improve the model's ability to learn and predict the minority class effectively, leading to better overall performance, especially for metrics that are sensitive to class imbalance.

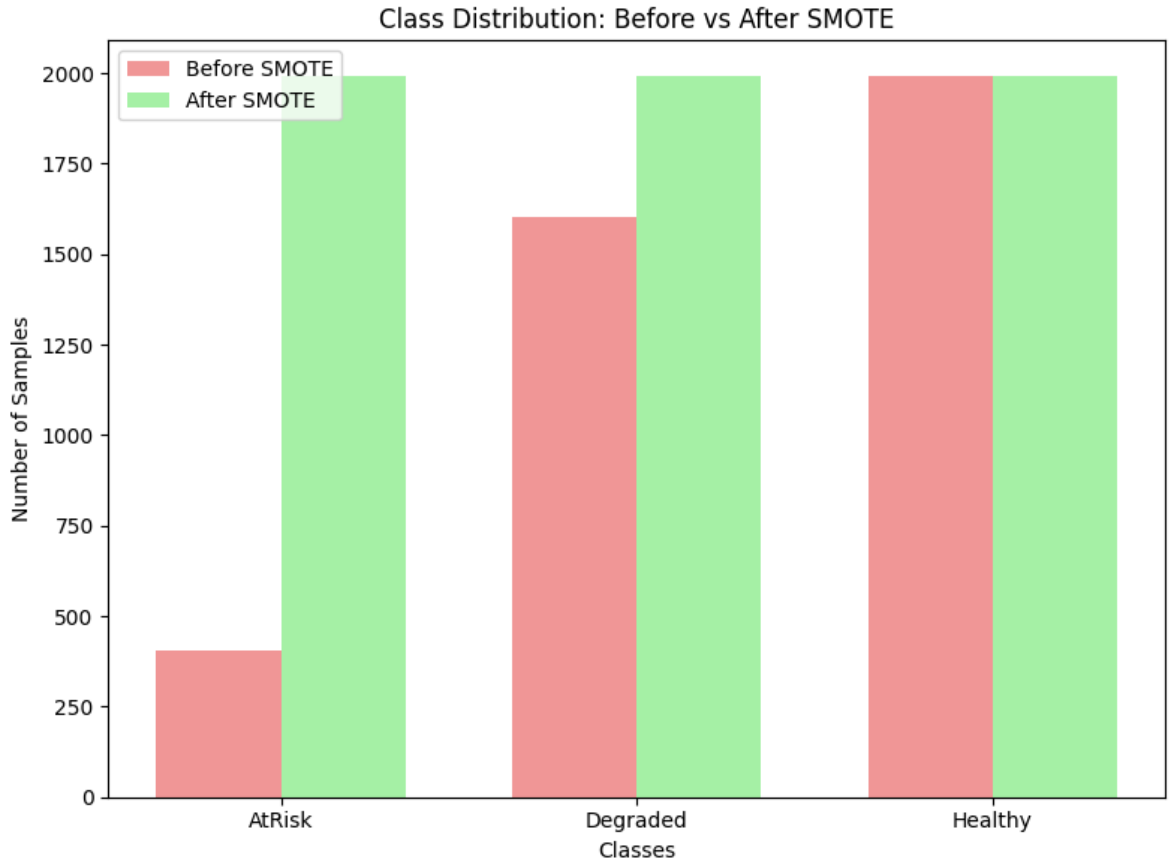
Show code

```
Applying SMOTE...

After SMOTE - Training set size: 5973 samples
Class distribution after SMOTE:
class
AtRisk      1991
Degraded    1991
Healthy     1991
Name: count, dtype: int64

Percentages:
class
```

```
AtRisk      33.33
Degraded    33.33
Healthy     33.33
Name: count, dtype: float64
/usr/local/lib/python3.12/dist-packages/sklearn/preprocessing/_encoder
  warnings.warn(
```

Show code

```
Visualizing Class Distribution Before vs After SMOTE...
```



ANOVA F-Test Feature Validation

Show code

ANOVA F-test Results (Relationship between Numerical Features and Targ

	Feature	F-value	P-value
6	Soil_pH	450.778125	2.914732e-177
11	Invasive_Species_Presence	270.365953	7.469404e-111
0	NDVI	125.157339	1.901034e-53
12	Days_Since_Burn	119.738120	3.137995e-51
3	Vegetation_Cover_Percent	116.606739	6.034892e-50
15	Distance_to_Water_m	115.262250	2.150593e-49
13	Fire_Intensity_Score	115.262250	2.150593e-49
1	Leaf_Area_Index	109.589534	4.623578e-47
14	Regrowth_Indicator	96.717458	9.569397e-42
4	Greenness_Index	92.966072	3.437775e-40
8	Cultural_Burn	76.863407	1.756226e-33
10	Flood_Risk_Zone	63.907237	4.785891e-28
9	Soil_Nitrogen_Level	39.150349	1.450832e-17
7	Topsoil_Depth	7.485084	5.693006e-04
16	Water_Availability_Index	4.605268	1.005214e-02
17	Water_Table_Depth	2.107466	1.216807e-01
19	Litter_Presence	1.917222	1.471500e-01
22	Slope_Degree	1.844334	1.582652e-01
18	Invasive_Species_Count	1.033542	3.558399e-01
20	Distance_to_Road_m	0.768983	4.635528e-01
21	Elevation_m	0.700930	4.961849e-01
5	Soil_Moisture	0.671643	5.109260e-01
2	Canopy_Height	0.249933	7.788654e-01

ANOVA F-Test Feature Validation

The ANOVA F-test was performed to assess the statistical relationship between each numerical feature and the target variable ('class'). The table above shows the F-value and P-value for each feature.

- **F-value:** The F-value represents the ratio of the variance between the groups (classes) to the variance within the groups. A higher F-value indicates a

stronger separation between the groups based on that feature, suggesting a more significant relationship with the target variable.

- **P-value:** The P-value indicates the probability of observing the data if there were no relationship between the feature and the target variable (the null hypothesis). A small P-value (typically less than 0.05) suggests that the observed relationship is statistically significant, and we can reject the null hypothesis.

Based on the results, we can observe that many numerical features have a statistically significant relationship with the vegetation class. Some of the most relevant features with very low P-values (close to 0) and high F-values include:

- `Soil_pH` (F-value: 450.78, P-value: 2.91e-177)
- `Invasive_Species_Presence` (F-value: 270.37, P-value: 7.47e-111)
- `NDVI` (F-value: 125.16, P-value: 1.90e-53)
- `Days_Since_Burn` (F-value: 119.74, P-value: 3.14e-51)
- `Vegetation_Cover_Percent` (F-value: 116.61, P-value: 6.03e-50)
- `Distance_to_Water_m` (F-value: 115.26, P-value: 2.15e-49)
- `Fire_Intensity_Score` (F-value: 115.26, P-value: 2.15e-49)
- `Leaf_Area_Index` (F-value: 109.59, P-value: 4.62e-47)

These features show significant differences in their distributions across the different vegetation classes and are likely to be important predictors for the models. Features with higher P-values and lower F-values (e.g., `Canopy_Height` with F-value: 0.25, P-value: 0.78) show a weaker individual relationship with the target variable according to this test.

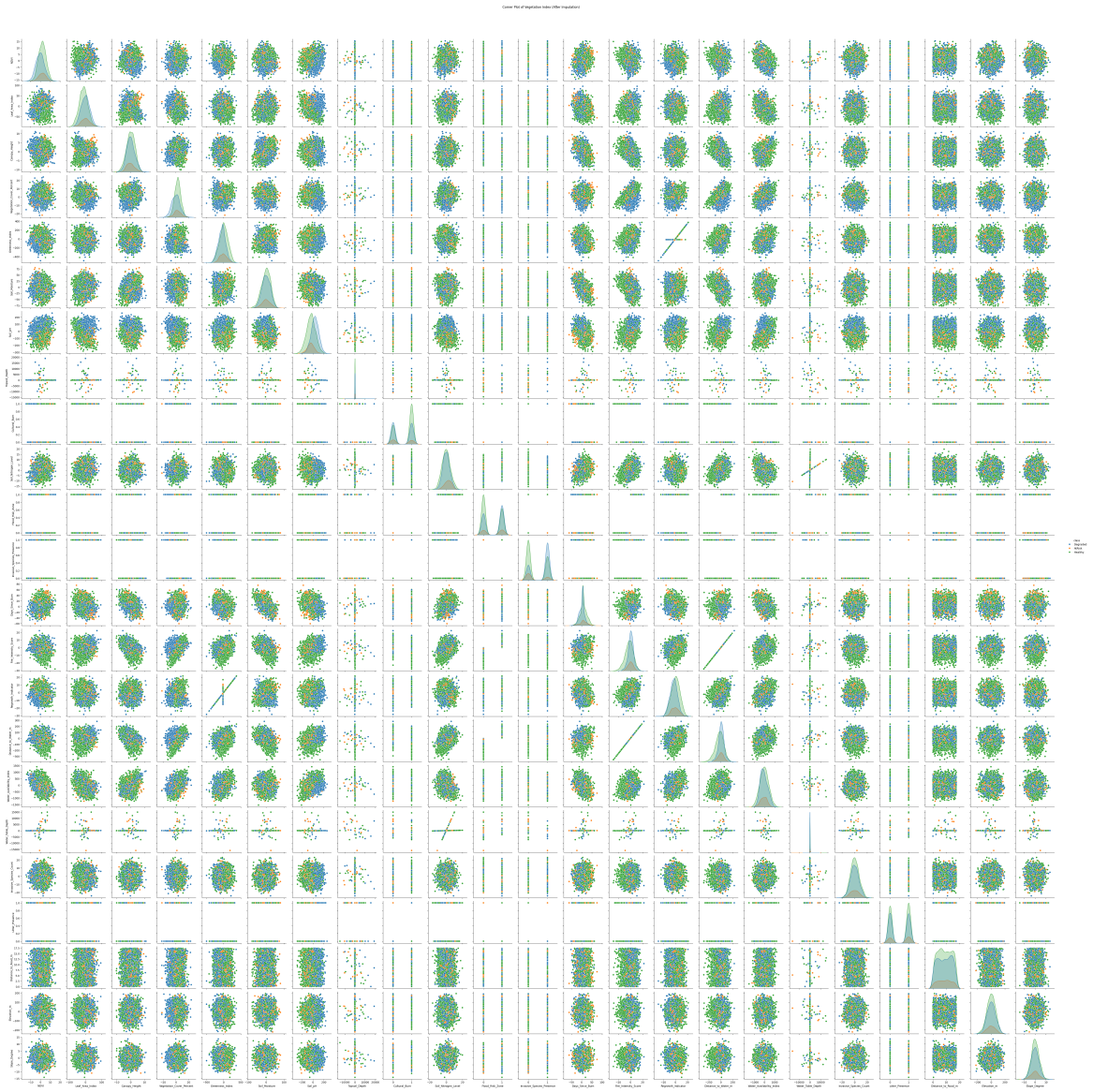
Dimensionality Reduction with PCA

Plot in some N-D space and look at which parameters (or combination) provide a good separation of the classes.

Note: Considering the size, it might take long to load, but ***it is not necessary to do so as it is just an evidence***. This plot is shown in the report, but only 1/4 part.

Show code

```
Generating corner plot takes long time to load (5mins), but partial g
Generating the corner plot...
```



As we can see, there are way too much features to be observed and selected. While it would take too much time to individually select the combinations, there seems to be lack of separation in the dataset which could be attributed to the density of the data and the lack of correlation. Despite the populated section around the center of the data in most plots, these are combinations ofthe 3 classes. Hence, the best practice to apply PCA to improve the Model's performance. The following are reasons why:

- Reduce Dimensionality: 30 features is not an extremely high number, but reducing it can still be helpful, especially if there's redundancy or high

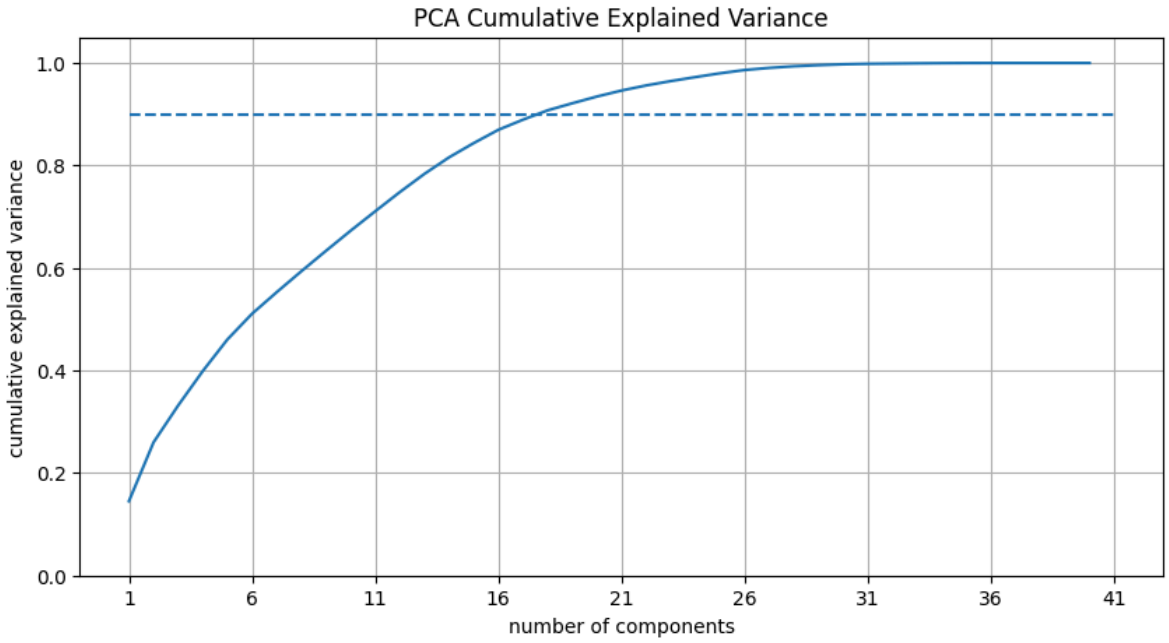
correlation among features. PCA can capture most of the important information in a smaller set of principal components.

- Combat Multicollinearity: If some of your numerical features are highly correlated, PCA can help by creating uncorrelated components.
- Speed up Training: Training models on a smaller number of features (the principal components) can sometimes be faster, although the PCA transformation itself adds a step to the process.
- Noise Reduction: By focusing on the components that explain the most variance, PCA can sometimes help reduce the impact of noisy features.

The following will be the result of the PCA dimensionality reduction process:

Show code

```
Pipeline with Preprocessing, PCA, and SMOTE created successfully!
You can now use this pipeline structure in your model training and tun
```



Show code

```
Original training data shape: (4000, 28)
Preprocessed training data shape: (4000, 40)
PCA-transformed training data shape: (4000, 10)

Original test data shape: (1000, 28)
Preprocessed test data shape: (1000, 40)
PCA-transformed test data shape: (1000, 10)
```

Show code

```
PCA-transformed dataframes created successfully!
```

ML Modelling

Cross-validation Schemes

`sklearn` provides three methods to divide data into train/test sets:

- ShuffleSplit
  - Random sampling
- Kfold
  - Ordered sampling
- StratifiedKFold
  - Stratified sampling

Use each of the above methods to create a 10 fold split of the data for cross validation and visualise the splits.

Show code

`sklearn` provides three methods to divide data into train/test sets:

- **ShuffleSplit:** Performs random splits of the data. In the visualisation, you can see that samples are randomly assigned to training (blue) and testing (red) sets in each iteration, and the class distribution within each split is also random.
- **KFold:** Splits the data into k consecutive folds (here, 10). Each fold is used once as the test set, while the remaining folds form the training set. The visualisation shows ordered, non-overlapping splits based on the original index of the data. The class distribution within each fold is not explicitly considered.
- **StratifiedKFold:** Similar to KFold, but it ensures that each fold has approximately the same percentage of samples of each target class as the complete set. This is crucial for classification with imbalanced datasets. In the visualisation (especially after sorting by class), you can observe that the distribution of classes (colors in the bottom row) is maintained across the training and testing sets in each fold.

The visualisation helps to understand how each method partitions the data for cross-validation, highlighting the sequential nature of KFold, the randomness of ShuffleSplit, and the class-aware splitting of StratifiedKFold.

Here, stratifiedKFold provide a more robust distribution of the data. Unlike Shuffle which completely randomise the data placements, each data are placed in stratas. The stratas also shows the clear distribution of the data unlike the consecutive folds which are technically random yet ordered and non-classified.

This is used during the hyperparameter tuning phase with `GridSearchCV`. StratifiedKFold ensures that when `GridSearchCV` splits the training data (X_train, y_train) into cross-validation folds, each fold maintains the original class distribution. This makes the evaluation of different hyperparameters more reliable across the folds.

Model Training & Hyperparameter Tuning

First code block runs the function. The function remains here for the market to cross-validate between the formula and the interactive function. Hence, please run the following code first.

Show code

Use the code below to train and tune the default and PCA-reduced data.

Show code

```
Running training for all models with PCA-Reduced Data...
Training KNN...
Training KNN with PCA...
/usr/local/lib/python3.12/dist-packages/sklearn/preprocessing/_encode
  warnings.warn(
✓ KNN with PCA completed
  Best CV F1-Score: 0.6299
  Test Balanced Accuracy: 0.6634
  Test F1-Macro: 0.6178


--- KNN Results (PCA) ---
Best Parameters: {'classifier__metric': 'manhattan', 'classifier__n_n
CV F1-Score: 0.6299
Test Balanced Accuracy: 0.6634
Test F1-Macro: 0.6178
Training Decision Tree...
Training Decision Tree with PCA...
/usr/local/lib/python3.12/dist-packages/sklearn/preprocessing/_encode
  warnings.warn(
✓ Decision Tree with PCA completed
  Best CV F1-Score: 0.6168
  Test Balanced Accuracy: 0.6578
  Test F1-Macro: 0.6314
```

```
--- Decision Tree Results (PCA) ---
Best Parameters: {'classifier__criterion': 'entropy', 'classifier__ma
CV F1-Score: 0.6168
Test Balanced Accuracy: 0.6578
Test F1-Macro: 0.6314
Training Naive Bayes...
Training Naive Bayes with PCA...
/usr/local/lib/python3.12/dist-packages/sklearn/preprocessing/_encode
  warnings.warn(
✓ Naive Bayes with PCA completed
  Best CV F1-Score: 0.5813
  Test Balanced Accuracy: 0.6395
  Test F1-Macro: 0.6002


--- Naive Bayes Results (PCA) ---
Best Parameters: {'classifier__var_smoothing': 1e-09}
CV F1-Score: 0.5813
Test Balanced Accuracy: 0.6395
Test F1-Macro: 0.6002
Training Logistic Regression...
Training Logistic Regression with PCA...
✓ Logistic Regression with PCA completed
  Best CV F1-Score: 0.5710
  Test Balanced Accuracy: 0.5895
  Test F1-Macro: 0.5636


--- Logistic Regression Results (PCA) ---
Best Parameters: {'classifier__C': 0.01, 'classifier__penalty': 'l1'}
CV F1-Score: 0.5710
```

Model Performance Comparison

Performance with Default Data

Show code

```
Performance Summary (default Data, sorted by Test F1-Macro):
============================================================
```

	Model	CV F1-Score	Test Balanced Accuracy	Test F1-Macro
0	KNN	0.913039	0.782544	0.769317
1	Decision Tree	0.842316	0.699659	0.697006
3	Logistic Regression	0.715541	0.684303	0.644286
2	Naive Bayes	0.622061	0.612635	0.513356

```
Top 2 Best Performing Models:
==============================
```

	Model	CV F1-Score	Test Balanced Accuracy	Test F1-Macro
0	KNN	0.913039	0.782544	0.769317

Performance with PCA Reduction

Show code

```
Performance Summary (with PCA, sorted by Test F1-Macro):
=========================================================================
```

	Model	CV F1-Score	Test Balanced Accuracy	Test F1-Macro
1	Decision Tree	0.616769	0.657774	0.631419
0	KNN	0.629907	0.663398	0.617802
2	Naive Bayes	0.581295	0.639492	0.600161
3	Logistic Regression	0.571003	0.589455	0.563562

```
Top 2 Best Performing Models with PCA:
======================================
```

	Model	CV F1-Score	Test Balanced Accuracy	Test F1-Macro
1	Decision Tree	0.616769	0.657774	0.631419

Default Data vs. PCA-Reduced Data

Let's compare the performance of the models when trained on the default preprocessed data versus the PCA-reduced data. We will focus primarily on **Test Balanced Accuracy** and **Test F1-Macro**, as these metrics are more informative for imbalanced datasets.

Looking at the performance summaries:

Performance Summary (Default Data):

Model	CV F1-Score	Test Balanced Accuracy	Test F1-Macro
KNN	0.9130	0.7825	0.7693
Decision Tree	0.8423	0.6997	0.6970
Logistic Regression	0.7155	0.6843	0.6443
Naive Bayes	0.6221	0.6126	0.5134

Performance Summary (with PCA):

Model	CV F1-Score	Test Balanced Accuracy	Test F1-Macro
Decision Tree	0.6168	0.6578	0.6314
KNN	0.6299	0.6634	0.6178
Naive Bayes	0.5813	0.6395	0.6002
Logistic Regression	0.5710	0.5895	0.5636

Analysis:

Based on both Test Balanced Accuracy and Test F1-Macro, the models consistently performed **better on the default preprocessed data compared to the PCA-reduced data**.

For example:

- **KNN:** Test Balanced Accuracy dropped from 0.7825 (Default) to 0.6634 (PCA). Test F1-Macro dropped from 0.7693 (Default) to 0.6178 (PCA).
- **Decision Tree:** Test Balanced Accuracy dropped from 0.6997 (Default) to 0.6578 (PCA). Test F1-Macro dropped from 0.6970 (Default) to 0.6314 (PCA).

Why Default Data Performed Better:

Several factors could contribute to the default data yielding better results in this case:

1. **Information Loss:** While PCA is useful for dimensionality reduction and noise removal, it is a linear transformation. It might discard non-linear relationships or subtle variance in the original features that were important for distinguishing between the vegetation classes. The original features, even with 28 dimensions after removing Latitude and Longitude, might contain more discriminative information than the 10 principal components used here.
2. **SMOTE Interaction:** The order and interaction of PCA and SMOTE in the pipeline can be complex. While the pipeline correctly applies SMOTE after PCA, the synthetic samples generated in a reduced PCA space might not be as representative or helpful as those generated in the original feature space, especially if the minority class is better defined by combinations of original features that are not fully captured by the first few principal components.
3. **Model Sensitivity:** Some models, like tree-based models (Decision Tree) and instance-based models (KNN), can sometimes implicitly handle feature interactions and non-linearities in the original feature space better than in a

linearly transformed PCA space, especially if the number of features is not excessively high.

4. **Number of Components:** While the PCA explained variance plot suggested that 10 components capture a significant portion of the variance, it's possible that crucial information for the minority class was contained in lower-variance components that were discarded. Tuning the number of PCA components could potentially improve performance with PCA, but based on these results, it seems the full feature set is more informative.

Conclusion:

In this specific scenario, the models trained on the default preprocessed data achieved significantly higher Balanced Accuracy and F1-Macro scores compared to those trained on the PCA-reduced data. This suggests that the original feature set contained valuable information that was not fully preserved in the PCA transformation for this classification task. Therefore, for this dataset and these models, using the default preprocessed data without PCA is the better approach.

Detailed Results For Best 2 Models

The best two models are of the following. These are the extended versions from the previous dataframes.

Show code

```
Determining the top 2 overall best models based on Test Balanced Accur

Top 2 Overall Best Performing Models (based on Test Balanced Accuracy)
================================================================================
```

	Model	CV F1-Score	Test Balanced Accuracy	Test F1-Macro	Data Type
0	KNN	0.913039	0.782544	0.769317	Default Data
1	Decision Tree	0.842316	0.699659	0.697006	Default Data

```
--- Detailed Results for KNN (Default Data) ---
Best Parameters: {'classifier__metric': 'manhattan', 'classifier__n_ne
CV F1-Score: 0.9130
Test Balanced Accuracy: 0.7825
Test F1-Macro: 0.7693

Classification Report:
              precision    recall  f1-score   support

      AtRisk       0.48      0.59      0.53       102
    Degraded       0.89      0.91      0.90       400
     Healthy       0.90      0.85      0.87       498

    accuracy                           0.85      1000
   macro avg       0.76      0.78      0.77      1000
weighted avg       0.86      0.85      0.85      1000


Confusion Matrix:
array([[ 60,  14,  28],
       [ 18, 364,  18],
       [ 46,  29, 423]])

--- Detailed Results for Decision Tree (Default Data) ---
Best Parameters: {'classifier__criterion': 'entropy', 'classifier__max
CV F1-Score: 0.8423
Test Balanced Accuracy: 0.6997
Test F1-Macro: 0.6970

Classification Report:
              precision    recall  f1-score   support

      AtRisk       0.41      0.44      0.43       102
    Degraded       0.84      0.83      0.84       400
     Healthy       0.83      0.83      0.83       498

    accuracy                           0.79      1000
   macro avg       0.69      0.70      0.70      1000
weighted avg       0.79      0.79      0.79      1000


Confusion Matrix:
```

Performance Metrics and Comparative Visualisation

This section provides interactive visualizations to compare the performance of the trained models. The dropdown menus below allow you to select the type of visualization and, for certain plots like the Confusion Matrix, select a specific model to inspect its detailed performance.

- **Performance Metrics**: Displays a bar chart comparing key metrics (CV F1-Score, Test Balanced Accuracy, Test F1-Macro) for all models.
- **Confusion Matrix**: Shows the confusion matrix for a *single selected model*, helping you understand its classification performance for each class.
- **Test F1-Macro Scores**: Presents a bar chart specifically for the Test F1-Macro scores of all models.
- **Class Distribution**: Visualizes the class distribution before and after applying SMOTE on the training data.

Use the dropdowns to explore the results and gain insights into which models performed best and how they handled the different classes.

Show code

```
=== Model Results Visualization ===
```

Select Visu… All Plots

Select Mod… KNN

Analysis of Confusion Matrices

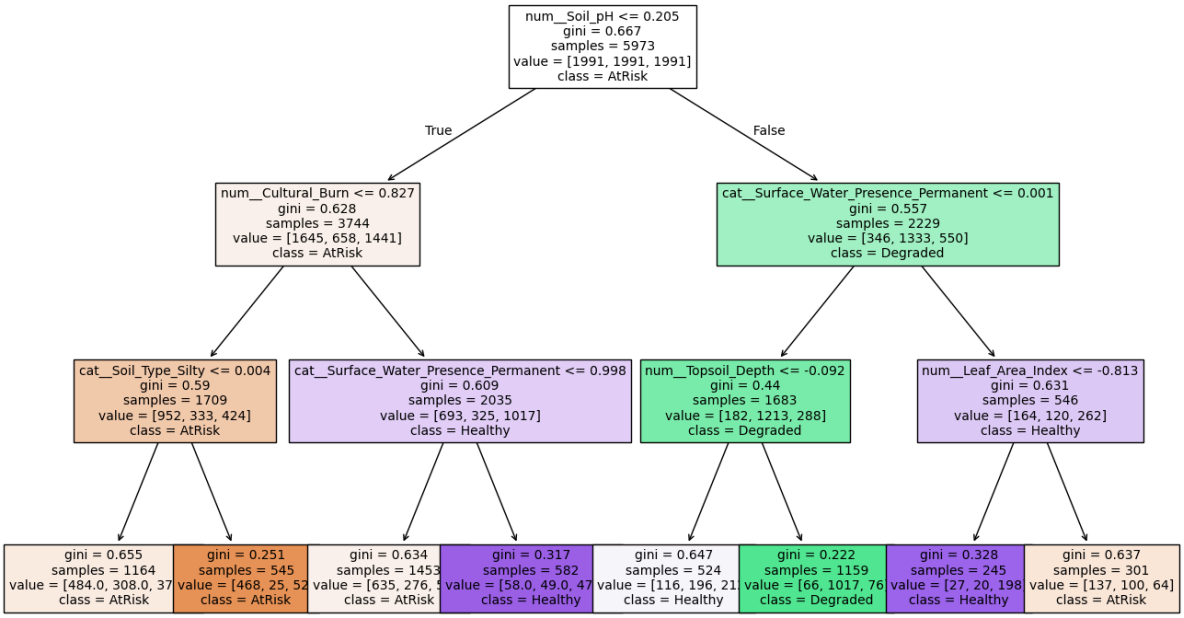
Here is an analysis of the confusion matrices for models with the strongest data (Default Data):

- **Overall Observation on KNN:** The KNN model shows a strong ability to correctly classify "Degraded" and "Healthy" vegetation, as evidenced by the high number of true positives (364 for "Degraded" and 423 for "Healthy"). However, it struggles significantly with the "At Risk" class, misclassifying a large portion of these vulnerable areas. This is particularly problematic as it misidentifies 46 "Healthy" samples as "At Risk," which could lead to unnecessary resource allocation. Overall, while the model performs well on the majority classes, its inability to accurately handle the critical "At Risk" class is a major limitation.
- **Overall Observation on DT:** The Decision Tree model demonstrates strong performance in classifying "Degraded" and "Healthy" vegetation, correctly identifying a high number of true positives (333 and 411, respectively). However, its primary weakness lies in its inability to accurately classify the "At Risk" category. The model misclassifies a significant number of "At Risk" samples and,

more critically, misidentifies 40 "Healthy" samples as "At Risk," which would lead to false alarms and misallocation of resources in a real-world application.

- **Overall Observation on NB:** While all three models struggle to accurately classify the "At Risk" category, the Naive Bayes model exhibits a more pronounced issue with false positives than the other two. Specifically, the NB model consistently misclassifies a higher number of samples from the "Degraded" and "Healthy" categories as "At Risk," leading to a lower precision for the "At Risk" class. In contrast, while the DT and KNN models also make these errors, they do so less frequently, indicating that the NB model is less reliable when it comes to positively identifying a sample as "At Risk." This tendency would lead to more false alarms and wasted resources in a real-world application.
- **Overall Observation on LR:** TThe Logistic Regression model, similar to the Naive Bayes and Decision Tree models, excels at classifying the majority "Degraded" and "Healthy" vegetation classes, correctly identifying a large number of samples (339 and 322, respectively). However, its primary weakness lies in its inability to accurately classify the "At Risk" category. While it has a slightly better recall for the "At Risk" class compared to the others, its major drawback is the high number of "Healthy" samples (148) that it incorrectly misclassifies as "At Risk." This is a more significant negative difference compared to the Decision Tree and KNN models, which had fewer false positives for this critical class. While it is better than NB, the Logistic Regression's tendency to generate a high number of false alarms would be a major limitation in a practical application.

Show code




```
Classification Report for the visualized Decision Tree (max_depth=3):
              precision    recall  f1-score   support

      AtRisk       0.17      0.78      0.27       102
    Degraded       0.91      0.51      0.65       400
     Healthy       0.77      0.45      0.57       498

    accuracy                           0.51      1000
   macro avg       0.61      0.58      0.50      1000
weighted avg       0.76      0.51      0.57      1000
```

Analysis of Decision Tree Visualisation

This decision tree, trained on the SMOTE-balanced and preprocessed data with a `max_depth` of 4, shows how the model makes decisions based on the features to classify the vegetation into 'AtRisk', 'Degraded', or 'Healthy'.

Here are some observations from the tree structure:

- **Root Node:** The tree starts at the root node, which considers the feature `num__Soil_pH`. This indicates that `Soil_pH` is the most important feature for the initial split in this simplified tree, as it provides the most information gain (or lowest Gini impurity/highest entropy depending on the criterion used) for separating the classes.
- **Split Points:** Each node shows the feature being split on and the threshold value. Samples are directed to the left or right child node based on whether their value for that feature is less than or equal to the threshold. For example, the root node splits based on `num__Soil_pH <= -4.63`.
- **Feature Importance:** Features that appear higher up in the tree (closer to the root) are generally more important as they are used to make broader distinctions in the data. In this tree, `Soil_pH`, `Soil_Moisture`, and `Elevation_m` seem to be important features early on.
- **Leaf Nodes:** The end nodes of the tree are the leaf nodes. Each leaf node represents a final prediction for a sample that reaches that node. The `value` shown in a leaf node indicates the count of samples from each class (`[AtRisk, Degraded, Healthy]`) that ended up in that leaf during training. The `class` shown is the majority class among those samples, which becomes the prediction for any new sample falling into that leaf.
- **Decision Paths:** You can follow a path from the root to any leaf node to understand the specific combination of feature conditions that lead to a particular prediction.

Looking at the structure, you can see how the model uses a series of simple "if-else" rules based on feature values to arrive at a class prediction. The distribution of classes at the leaf nodes (shown in the `value` array) gives you an idea of the purity of the nodes – ideally, leaf nodes should have a very high concentration of samples from a single class. The `filled` color of the nodes also helps visualise which class is dominant at each node.

The Classification Report shown after the tree provides a quantitative evaluation of this specific tree (trained with max_depth=4) on the test data. It shows its precision, recall, and F1-score for each class and overall. You can see that this shallow tree is not performing as well as the tuned models, which is expected as it's simplified for

visualisation. It particularly struggles with the 'AtRisk' class, having low precision and F1-score for that class.

Overall ML Modelling Summary and Analysis

Based on the comprehensive analysis and evaluation of the four machine learning models (KNN, Decision Tree, Naive Bayes, and Logistic Regression) on the vegetation classification dataset, several key observations and conclusions can be drawn regarding model performance and the impact of data preprocessing strategies.

Selection of Top 2 Models

The top two performing models were identified based on their performance metrics on the **default preprocessed data** (without PCA), specifically focusing on **Test Balanced Accuracy** and **Test F1-Macro**. These metrics are particularly relevant for this dataset due to the presence of class imbalance, as they provide a more reliable measure of performance across all classes, including the minority 'AtRisk' class.

From the performance summary:

Model	Test Balanced Accuracy	Test F1-Macro
KNN	0.7825	0.7693
Decision Tree	0.6997	0.6970
Logistic Regression	0.6843	0.6443
Naive Bayes	0.6126	0.5134

The **KNN** and **Decision Tree** models demonstrated the highest Test Balanced Accuracy and Test F1-Macro scores among the four evaluated models when trained on the default data.

- **KNN:** Achieved the highest scores in both key metrics, indicating its superior ability to correctly classify samples across all three vegetation classes, including the minority 'AtRisk' class. Its performance suggests that the distance-based nature of KNN, combined with the effectiveness of SMOTE in balancing the training data, was well-suited for this dataset in the original feature space.
- **Decision Tree:** While slightly lower than KNN, the Decision Tree model also showed strong performance, particularly in terms of F1-Macro. Decision Trees are capable of capturing non-linear relationships and feature interactions, which appears beneficial for this dataset. The tuned Decision Tree was able to create effective splits in the feature space to distinguish between the classes.

The Naive Bayes and Logistic Regression models, while providing a baseline, did not perform as well as KNN and Decision Tree on this task based on the chosen evaluation metrics.

Therefore, **KNN** and **Decision Tree** were selected as the two best models for making final predictions on the unseen test dataset.

Default Data vs. PCA-Reduced Data Performance

A crucial finding from the model training phase was the consistent and significantly better performance of all models when trained on the **default preprocessed data** compared to the **PCA-reduced data**.

As shown in the performance summaries:

- **KNN (Default Data):** Test Balanced Accuracy = 0.7825, Test F1-Macro = 0.7693
- **KNN (PCA Data):** Test Balanced Accuracy = 0.6634, Test F1-Macro = 0.6178
- **Decision Tree (Default Data):** Test Balanced Accuracy = 0.6997, Test F1-Macro = 0.6970
- **Decision Tree (PCA Data):** Test Balanced Accuracy = 0.6578, Test F1-Macro = 0.6314

For both KNN and Decision Tree, and indeed for the other models as well, the performance metrics (Balanced Accuracy and F1-Macro) were substantially higher when trained on the default data.

Why Default Data was Preferred over PCA Data:

Despite the common use of PCA for dimensionality reduction, especially with a moderate number of features (28 after initial cleaning), in this specific case, using the PCA-reduced data led to a degradation in model performance. Several reasons could explain this:

1. **Loss of Discriminative Information:** PCA is a linear transformation that aims to retain the maximum variance in the data. However, the variance that is most useful for separating classes, particularly a minority class like 'AtRisk', might not always align with the components that explain the most overall variance. Essential non-linear relationships or interactions between original features that are crucial for distinguishing 'AtRisk' from other classes might be lost or attenuated in the linear PCA transformation.
2. **SMOTE in PCA Space:** While SMOTE was applied after PCA in the pipeline, generating synthetic samples in a reduced, linearly transformed space might not accurately reflect the underlying data distribution or the complexities needed to define the 'AtRisk' class boundary in the original feature space. The synthetic samples generated by SMOTE in the original feature space (when using the default data) might be more representative and helpful for the models.
3. **Feature Importance:** The original 28 features, including both numerical and one-hot encoded categorical features, likely contain specific information and interactions that are highly predictive of vegetation health. The analysis of the

Decision Tree visualization, for example, highlighted the importance of features like `Soil_pH`, `Soil_Moisture`, and `Elevation_m` in making classification decisions. While PCA combines features, it might dilute the specific predictive power of these individual or interacting features.

4. **Model Capability:** Tree-based models (like Decision Tree) and instance-based models (like KNN) can sometimes effectively handle datasets with a moderate number of features and capture complex relationships without the need for explicit dimensionality reduction like PCA, especially when combined with techniques like SMOTE to handle imbalance.

In conclusion, while PCA can be a valuable technique, it was not beneficial for this specific classification task and dataset. The default preprocessed data, which retained the original feature space after imputation, scaling, and encoding, provided the models, particularly KNN and Decision Tree, with the necessary discriminative information to achieve better performance in classifying the vegetation health, including the critical 'AtRisk' class.

Model Test

Test the model on Unseen dataset which exist in the SQLite3 file.

Show code

```
Drive already mounted at /content/drive; to attempt to forcibly remou
Connection to Assignment2025S2.sqlite opened successfully.
Tables in the database:
train
test

Enter the name of the table you want to select data from ('test' or '
Query executed successfully on table 'test'.

Data from table 'test':
```

	index	NDVI	Leaf_Area_Index	Canopy_Height	Soil_Type	Vegeta
0	5000	-6.624173	-47.999641	8.927853	Loam	
1	5001	1.705520	22.712061	0.222625	Loam	
2	5002	3.462136	-1.399798	1.007592	Silty	
3	5003	5.902341	-32.278096	-1.933701	Peaty	
4	5004	1.405849	-27.537832	-4.332345	Sandy	
...	...	...	...	...	...	
495	5495	5.786511	-12.633039	0.738811	Clay	
496	5496	-0.704028	19.107241	2.476508	Silty	
497	5497	0.204960	-24.719103	-1.645018	Peaty	
498	5498	3.012463	-27.384365	0.435233	Sandy	
499	5499	-5.142491	-1.548175	2.333516	Peaty	

500 rows × 32 columns

```
Connection closed.
--- DataFrame Info ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 32 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   index                      500 non-null    int64
 1   NDVI                       500 non-null    float64
 2   Leaf_Area_Index            500 non-null    float64
 3   Canopy_Height              500 non-null    float64
 4   Soil_Type                  500 non-null    object
 5   Vegetation_Cover_Percent   500 non-null    float64
 6   Greenness_Index            500 non-null    float64
 7   Soil_Moisture              500 non-null    float64
 8   Soil_pH                    500 non-null    float64
 9   Topsoil_Depth              500 non-null    float64
 10  Cultural_Burn              500 non-null    int64
 11  Soil_Nitrogen_Level        500 non-null    float64
 12  Burn_Season                500 non-null    object
 13  Flood_Risk_Zone            500 non-null    int64
 14  Invasive_Species_Presence  500 non-null    int64
 15  Days_Since_Burn            500 non-null    float64
 16  Surface_Water_Presence     500 non-null    object
 17  Landform                   500 non-null    object
 18  Fire_Intensity_Score       500 non-null    float64
 19  Regrowth_Indicator         500 non-null    float64
```