Data

For our project, we have decided to create a model on the *Forest Cover Type Prediction* dataset in Kaggle. This is a multi-class classification problem where we will be looking to predict the type of forest cover for a given 30×30-meter cell in the Roosevelt National Forest located in northern Colorado. This dataset contains several numeric features and many binary indicators with the goal to build a model that accurately classifies the column "Cover_Type." We decided to go with this problem because we were interested in working with a dataset that has both numerical and categorical values. Additionally, we are excited to work on a machine learning problem that would have real-world applications in environmental science and land management.

Dataset

Below is an example table that captures each column, along with the preferred data type for modeling. For brevity, the 4 Wilderness_Area columns and 40 Soil_Type columns are grouped together, but in the actual dataset these will be 44 binary columns (with values 0 or 1) only, indicating the wilderness area and soil type of the 30x30 meter land (1 = yes; 0 = no).

Column Name	Data Type		Column Name	Data Type
Elevation	Numerical		Hillshade_Noon	Numerical
Aspect	Numerical		Hillshade_3pm	Numerical
Slope	Numerical		Horizontal_Distance_To_Fire_Points	Numerical
Horizontal_Distance_To_Hydrology	Numerical		Wilderness_Area(1–4)	Cat (logical)
Vertical_Distance_To_Hydrology	Numerical		Soil_Type(1–40)	Cat (logical)
Horizontal_Distance_To_Roadways	Numerical		Cover_Type (target)	Cat (factor)
Hillshade_9am	Numerical			

*Note: Data types are not immediately read in R as needed, so we would need to prepare the data as numerical/categorical accordingly
**For the full data dictionary, please refer to the dataset description here: https://www.kaggle.com/c/forest-cover-type-prediction/data

Target and KPIs

Our goal is to predict which forest cover type (from 7 different types) each piece of 30x30 meter land belongs to. To do this, we will build a machine learning model using the "train" dataset from Kaggle, making use of all (or some) of the available columns to learn which factors best predict each cover type. We will split the training data into two parts: one portion (the training set) will teach the model how different factors relate to cover type. Once the model is trained, we will then use the remaining portion (the test set) to see how well the model performs on unseen data. By comparing the predicted cover types to the actual ones in this test set, we can evaluate how accurate our model is.

To measure how well our model does, we use error rate, which is simply the percentage of incorrect predictions from our model using the train dataset. Since we do not currently have a trained model, we will have to create a "benchmark error rate" to assess how efficient our model is. To achieve this, we will first create a simple benchmark model that always guesses the most frequent cover type, no matter what. This basic approach is called predicting the mode (the most common value). We will measure how often this guess is wrong which will then become our baseline or benchmark error. By comparing our (hopefully) more advanced model to this simple guesser, we can tell if our modeling efforts are improving performance.

Additionally, we will be using other metrics to evaluate our model's efficiency. When working with multiple categories (for example, predicting one of seven different forest cover types), precision and recall are helpful, and we calculate them for each class individually:

Precision

- "Out of all the predictions the model made for this class, how many were correct?"
- For instance, if the model predicts 100 areas as 'Spruce/Fir' but only 80 of them truly are 'Spruce/Fir,' the precision for that class is 80/100 = 80%.

Recall

- "Out of all the real instances of this class, how many did the model actually catch?"
- For instance, if there are 100 actual 'Spruce/Fir' areas, but the model only identifies 20 of them as 'Spruce/Fir,' the recall for that class is 20/100 = 20%.

For our project, we need a classification model because our target variable is categorical. Although we can use both KNN and Logistic Regression, we are leaning toward Logistic Regression since it handles both categorical and numerical features, which we believe will both be important for our predictions based on the data definitions. Though we currently only have these two classification models in our toolbox, we will keep an eye open for more potential models which we can use.

Test Data on Kaggle

The data set "test' on Kaggle is not useful in this project because this dataset does not contain our target column (Cover_Type). Our target column is not in the dataset because this dataset is used to judge the effectiveness of the models. We will need to split our "training" dataset into test/train to be able to have a test dataset that allows us to use our KPIs to assess model performance.

Challenges

One of the biggest challenges with this data set is the large number of "yes-or-no" columns for wilderness areas and soil types. These binary columns indicate whether a specific soil type or wilderness area is present (1) or not (0). Because there are so many of these columns, they can create a situation where most entries in a single row are 0s. We think that having an overwhelming number of binary columns could potentially make model building complicated and our model could struggle to generate patterns from mostly empty fields.

Additionally, some of the numerical columns, such as elevation, slope, and aspect, can be closely related with each other and other columns such as Soil_Type. When two or more variables provide overlapping information, it becomes harder to distinguish which variable is causing or explaining the differences in forest cover. For example, a high elevation might usually imply a certain soil type, or a specific slope could always correspond to a particular vegetation.

While splitting our training data into separate training and test/validation sets allows us to test our model, it also means that the model has fewer examples to learn from. This could increase the risk of underfitting (i.e., the model is too simple to capture data complexity). This is especially true since the Kaggle test dataset is much larger (565,892 observations) than the training set (15,120 observations) and there is a risk that our model may make inaccurate predictions.