QTM 7571 – Project Group 4 Dilo, Kenneth, Javier, and Priyanka
Project Phase 2

Executive Summary

The goal of this project was to build a machine learning model that accurately predicts forest cover types based on environmental data from the Roosevelt National Forest in Northern Colorado. Using a real-world dataset from the U.S. Forest Service, we sought to identify patterns in elevation, slope, soil type, sunlight exposure, and proximity to roads or water to determine what kind of forest is likely to exist in a given area. This type of predictive modeling has practical applications in environmental planning, land use, and conservation.

Our approach began with extensive data preparation and visualization. We confirmed data quality, transformed complex categorical variables into interpretable formats, and used visual tools to understand the relationships between variables and forest types. The dataset was then split into a training set (60%) and a test set (40%) to evaluate how well each model could perform on unseen data, a critical step to avoid overfitting, where a model learns the training data too well and fails to generalize.

We built and evaluated four models:

- K-Nearest Neighbors (KNN), a simple model based on proximity in feature space.
- Logistic Regression, a statistical method that offered clear interpretability.
- Random Forest, a powerful ensemble of decision trees.
- A Stacking Ensemble, which used a neural network to combine predictions from the three base models.

The KNN and Logistic Regression models provided helpful interpretability, highlighting which features drove classification, though they were more limited in handling categorical variables and complex relationships. In contrast, Random Forest and the Stacking Ensemble achieved the strongest performance, with both error rates being 14.4%, a substantial improvement over the 86.5% baseline.

As we explore the trade-offs between interpretability and accuracy, this project illustrates how combining models and carefully preparing data can significantly enhance predictive power. The rest of this report dives into the preparation steps, each model's logic, performance comparisons, and insights from our results.

Data Preparation & Visualization

Data Understanding & Transformation

Our dataset consisted of 15,120 rows and 55 columns with a combination of numerical and categorical features. Table 1 shows the numerical and categorical columns of the data frame:

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)	Categorical
Vertical_Distance_To_Hydrology	Numerical		Soil_Type(1–40)	Categorical
Horizontal_Distance_To_Roadways	Numerical		Cover_Type (target)	Categorical
Hillshade_9am	Numerical			

Table 1: Summary of Dataset Columns

We began by confirming that the dataset was clean (i.e. there were no missing or null values across any columns), which saved us significant preprocessing time. We also ensured that each row of data had exactly one wilderness area and one soil type by checking the row-wise sums across the binary columns. This allowed us to transform the 40 binary soil type columns into a single column labeled 'Soil_Type.' Similarly, we combined the 4 binary wilderness area columns into one 'Wilderness_Area' column. These changes allowed us to treat these variables as categorical (factor) data rather than as many independent logical (true/false) variables, which simplifies the modeling process and makes the output easier to interpret.

Next, we dropped the 'ID' column, which is simply a row identifier and does not contain any predictive value. We also converted the 'Cover_Type' variable, which is the variable we are trying to predict (often called the Target Variable), into a factor, since we are working with a classification problem with seven distinct forest types.

Training and Testing the Models

To evaluate how well our models perform, we followed a standard and important practice in machine learning: splitting our dataset into two parts: a training set and a test set. Specifically, we used 60% of the data to train the models (help them learn patterns) and set aside the remaining 40% to test their performance on data they had never seen before.

This approach is critical because it allows us to simulate how well our models will work in real-world conditions, where they will encounter new and unseen data. A model that performs well only on the data it was trained on may not generalize well to new cases, this is known as overfitting.

Think of overfitting like studying for an exam by memorizing the exact questions from last year's test. You might do great if the same questions appear again, but you'll struggle if the questions change even slightly. A good model, like a well-prepared student, should understand the material well enough to handle different but related questions and not just repeat memorized answers.

To help guard against overfitting, we used several techniques across our models:

- Train-test split: Holding out data for testing helps us measure whether the model can generalize beyond the training data.

- Standardizing features (for KNN and Neural Networks): Ensures that variables are treated fairly and prevents some from dominating due to larger scales.
- Stepwise variable selection (for Logistic Regression): Removes less useful variables to keep the model focused and simple.
- Ensemble learning (Random Forest and Stacking): These methods combine multiple models or decision paths, which tends to balance out individual model biases and reduce overfitting.

Lastly, we chose not to fix a random seed when splitting the data. In machine learning, setting a seed (e.g., set.seed(1234)) ensures that the same random split is used every time you run the code. While this can help with reproducibility, our focus was on understanding overall model behavior and performance trends and not on getting the exact same numbers every time.

Visual Insights

To help us further understand this dataset, we decided to visualize the data in a number of different ways to give us some understanding of the data before we modeled. First, we observed that the distribution of forest cover types in the dataset was balanced across all seven categories, resulting in 2,160 instances of each cover type. This means that no single cover type dominated the data, which is hopefully helpful to our model, as a balanced dataset ensures that the model has enough examples from each category to learn from and helps prevent the model from being biased toward the most common class. We recognize that this balance may not reflect real-world conditions, where class imbalance is common, making it important to acknowledge this unique characteristic of our dataset. Figure 1 demonstrates how each cover type is represented in our dataset:

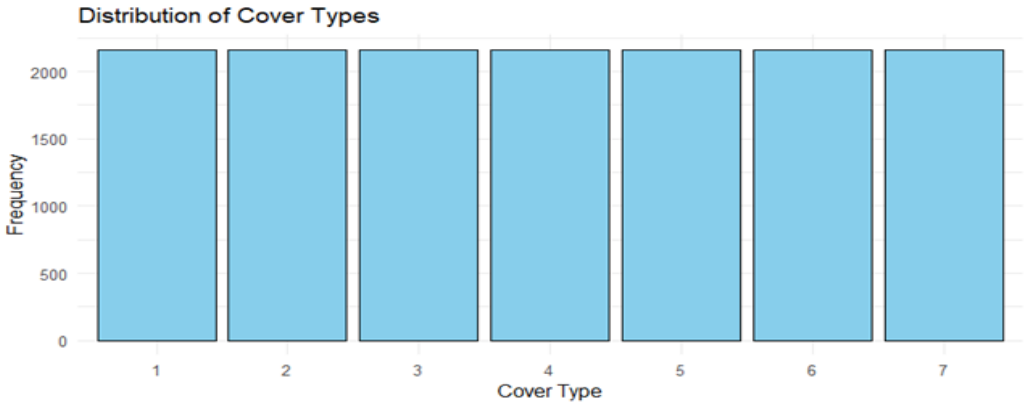

Figure 1: Distribution of Cover Types (target variable) in the dataset

We also explored histograms, which showed us the distribution of each continuous variable. For example, we saw elevation was somewhat normally distributed, while distances to hydrology or roadways (Figure 2) were skewed toward smaller values, indicating that most forest areas were closer to water sources and roads.

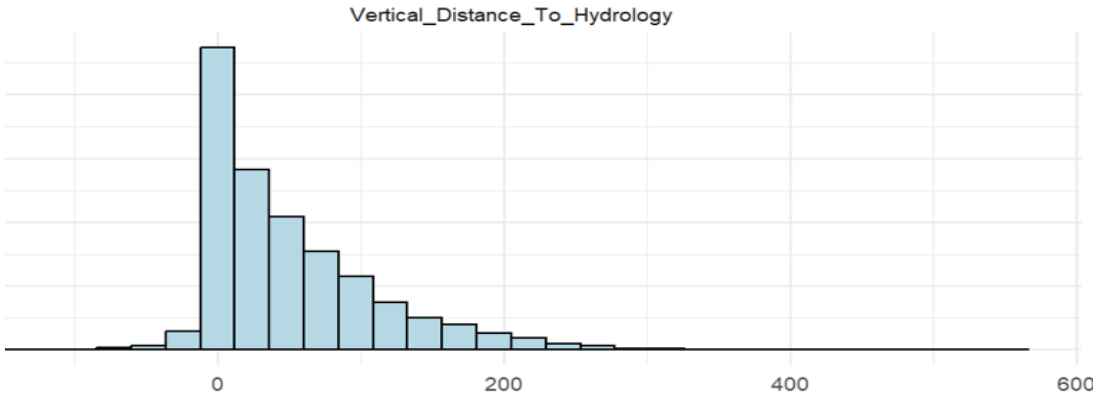

Figure 2: Histogram of Vertical Distance_to_Hydrology in the dataset

Moreover, box plots helped us see how these variables varied across different forest cover types. As seen in Figure 3, elevation appeared to be a strong differentiator between forest types, as some types were clearly found only at higher or lower elevations. On the other hand, variables like Aspect (the compass direction a slope faces) showed less variation across cover types, suggesting they might not be as informative for prediction.

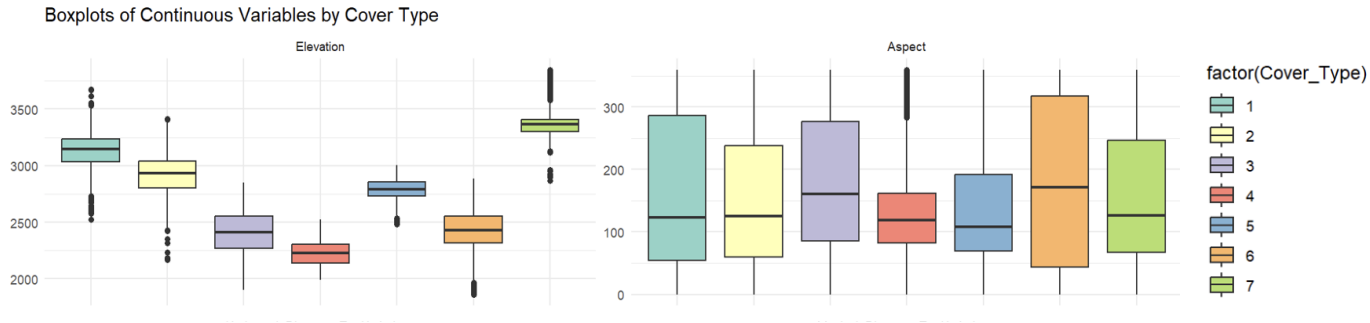

Figure 3: Box plots of Elevation and Aspect for each Cover Type in the dataset

These visualizations provided helpful early insights into which variables were most likely to be useful predictors, such as Elevation, and which ones might be less informative, such as Aspect.

Performance Measurement

After building our models, we needed a way to understand how well they performed on our Test data. We used performance metrics to evaluate how accurate and reliable each model's predictions were when tested on data it had not seen before.

To have something to compare to, we also established a baseline error rate. This represents the error rate we would get if we made a very simple prediction by always guessing the most common forest cover type in the training set for every observation in the test set. In our case, the baseline error rate was 86.5%, meaning this basic approach was wrong most of the time. Our goal was to build models that could improve upon this error rate and analyze the following metrics:

- Error Rate: This is the percentage of predictions that were incorrect. For example, an error rate of 30% means the model got 3 out of 10 predictions wrong. The lower the error rate, the better the model.
- Accuracy: This is the opposite of the error rate. It tells us how often the model was right. An accuracy of 70% means the model correctly predicted the cover type 7 out of 10 times.
- Confusion Matrix: This is a table that breaks down the model's predictions compared to the actual values. It shows which forest types the model tends to mix up. For example, if the model often predicts Type 2 when the true type is actually Type 3, the confusion matrix will highlight that.
- Precision: Precision tells us how often the model's prediction for a certain cover type is actually correct. For example, if the model predicts Type 4 twenty times, but only fifteen of those are right, the precision for Type 4 is 75%.
- Recall: Recall looks at it from the other direction than precision. Out of all the actual cases of a certain cover type, how many did the model catch correctly? If there were 100 real cases of Type 1 and the model correctly predicted 80 of them, the recall is 80%.

These metrics give us a more complete picture of model performance than just looking at accuracy alone. They help reveal not just whether the model is right, but also how it tends to be wrong.

Model 1: K-Nearest Neighbors (KNN)

The K-Nearest Neighbors (KNN) model is an intuitive classification algorithm. Instead of learning patterns during training, it stores the dataset and makes predictions based on similarity. When asked to predict the forest cover type of a new observation, it looks for the "K" most similar past cases and assigns the majority label. Think of it like asking a certain number of nearby forest plots what kind of cover they have and choosing the most common answer.

In our case, we followed the rule of thumb and set K to the square root of the training size, resulting in 123 as our default K. We standardized all numeric features so that variables measured on different scales (e.g., elevation in meters vs. slope in degrees) didn't unfairly influence the result.

However, one important limitation of KNN is that it cannot handle categorical data, so we had to exclude important variables like Soil Type and Wilderness Area.

Despite this, KNN still performed better than our benchmark model (which just guessed the most common type of every time), achieving a 40.4% error rate. Its strengths include simplicity and strong performance when data points cluster naturally. However, it tends to struggle with large datasets and cannot give insights into which features matter most.

Model 2: Logistic Regression

Logistic Regression is another classic machine learning method used for classification problems. It works by estimating the probability that a data point belongs to a specific category based on its features. Since our dataset includes seven possible forest cover types, we used multinomial logistic regression, which extends the method to handle more than two categories.

This model is valuable not just for making predictions, but also for interpreting which variables most influence the results. Each feature has a coefficient that shows how it increases or decreases the likelihood of belonging to a specific forest type. For example, Elevation had a strong negative effect for most forest types (Covers 2-6) compared to Cover Type 1, meaning higher elevation reduced the likelihood of those types. However, for Cover Type 7, Elevation had a strong positive coefficient (+8.78), suggesting this type is more likely to be found in high-elevation areas.

Other variables also played interesting roles:

- Slope had a strong positive coefficient for Cover Type 7, indicating steep terrain might be a distinguishing factor.
- Hillshade at 3pm was highly positive for Cover Type 6 and highly negative for Cover Type 4, suggesting sun exposure at different times of day could help distinguish between these types.

To simplify the model and reduce noise, we used stepwise selection, a technique that removes variables with little predictive value. The final model achieved an error rate of 29.3%, slightly better than KNN, and still a major improvement over the 86.5% benchmark.

One of Logistic Regression's strengths is its interpretability. It provides clear, measurable insights into how different features contribute to each prediction, a major advantage when transparency matters.

Model 3: Random Forest

Random Forest is a tree-based ensemble method that builds many decision trees and combines their predictions to improve accuracy and reduce overfitting. Each tree looks at a random sample

of the data and features, and the model makes its final prediction by aggregating the decisions of all trees, like taking a vote.

One of Random Forest's key advantages is its ability to handle both numerical and categorical variables without special preprocessing. In our model, we included all available features, including engineered variables like Total_Distance and Slope_Elevation, as well as important categorical inputs like Soil Type and Wilderness Area.

The Random Forest model delivered one of the best performances in our study, achieving an error rate of 14.4%, a substantial improvement over KNN and Logistic Regression.

We also explored the variable importance rankings, which revealed several meaningful insights:

- Elevation was the single most influential predictor, both in how much it improved accuracy and how often it was used across the decision trees.
- Soil Type ranked second in overall importance, showing how much ecological characteristics of the soil help determine cover type.
- Other highly important variables included Horizontal Distance to Roadways, Wilderness Area, and Hillshade at 9am, confirming the impact of accessibility and sunlight exposure on forest composition.

While Random Forest does not provide simple rules or coefficients for interpretation, it excels at capturing complex, nonlinear relationships in the data. Its ability to highlight which variables matter most makes it a powerful tool, even if its individual predictions are harder to unpack.

Ensemble Model: Stacking

After building and evaluating our three individual models, K-Nearest Neighbors, Logistic Regression, and Random Forest, we aimed to improve overall performance by combining them using a technique called stacking. Stacking is a form of ensemble learning, where the idea is to take the strengths of multiple models and blend them to make even better predictions.

To do this, we first took the predictions from each of our three base models and added them as new columns to our original dataset. This gave us a new dataset that not only included the original features (like elevation, slope, and soil type), but also what each model thought the cover type should be. We then trained a new model, a neural network, to learn from this combined information and make final predictions.

Neural networks are powerful but complex models that try to mimic how the human brain processes information. They are known for their ability to capture nonlinear patterns and interactions between variables that other models might miss. However, they function like a "black

box" as it is very difficult to understand how they arrive at a specific decision. This makes them less interpretable than simpler models like Logistic Regression.

We chose a neural network intentionally here. Since we were already combining models with different perspectives and performance patterns, trying to interpret exactly how each one influenced the final decision would have been very complicated. Instead, we opted for a flexible, high-performing method that could focus purely on maximizing prediction accuracy.

The stacked model using a neural network achieved an error rate of 14.4%, matching the Random Forest model. Our hypothesis is that Random Forest strongly influenced the decisions of the final neural net, since the predictions were quite similar. Nevertheless, the ensemble showed that by combining the perspectives of all three base models, we could build a final model that was strong in prediction types

Model Comparison

In this section, we compare the four models we built across key performance metrics. Our goal is to highlight how each model performed in terms of accuracy, interpretability, and practical use. We also identify the unique strengths and limitations of each approach, helping us understand not only which model was most effective, but also why.

Criteria	K-Nearest Neighbors (KNN)	Logistic Regression	Random Forest	Stacking: Neural Net
Error Rate	40.4% — highest of 3 models, better than benchmark	29.3% — Second overall, better than benchmark	14.4% - best overall performance tied with stacking	14.4% - best overall performance tied with Random Forest
Categorical Variable Handling	Could not use soil or wilderness variables (excluded from model)	Included all variables, including soil type and wilderness area	Included all variables – handled factors using internal splitting	Included all variables – handled categorical inputs via encoding or internally
Interpretability	Not easily interpretable — difficult to know which features influenced predictions	Highly interpretable — shows which variables contribute most to predictions	Less interpretable – difficult to explain specific decision logic of trees	Less interpretable – neural networks function like a "black box"
Overfitting Control	Used K = 123 and standardized all numeric features to prevent bias	Used stepwise selection to keep only the most useful variables	ensemble trees help prevent overfitting by averaging across splits	Utilization of only one hidden layer in the model
Preprocessing Required	Required additional steps — removed categorical variables and standardized data	Minimal preprocessing — able to use all features directly	Minimal preprocessing – can handle mixed data types natively	Moderate – required standardized inputs
Strengths	Strong accuracy; captures similarities in data effectively	Provides clear insights into which features matter most	High accuracy and robustness; identifies complex patterns and feature importance	Combines insights from multiple models; flexible; captures complex, nonlinear patterns
Limitations	Can't handle categorical variables directly;	Multicollinearity sensitivity — performance can suffer if input features are highly correlated with each other.	Less transparent; can be computationally intensive for large datasets	Hard to interpret; can be computationally expensive and sensitive

	performance drops on large datasets			to hyperparameter tuning

Table 2: Comparative Model Summary — KNN, Logistic Regression, Random Forest and Neural Net Stack

To further evaluate model performance, we compared their predictive accuracy using precision, recall, and accuracy for each forest cover type. These metrics help us understand not just how often the model was correct overall, but how reliable it was for each specific category. The table below summarizes the results for K-Nearest Neighbors (KNN), Logistic Regression and Random Forrest.

Cover Type	KNN			Logistic Regression			Random Forrest			Stacking		
	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy
1	0.55	0.43	0.43	0.58	0.65	0.65	0.78	0.76	0.76	0.79	0.76	0.76
2	0.55	0.34	0.34	0.47	0.56	0.56	0.78	0.68	0.68	0.77	0.69	0.69
3	0.55	0.35	0.35	0.52	0.43	0.43	0.85	0.81	0.81	0.84	0.80	0.80
4	0.71	0.86	0.86	0.79	0.68	0.68	0.92	0.96	0.96	0.93	0.97	0.97
5	0.57	0.74	0.74	0.74	0.70	0.70	0.89	0.95	0.95	0.90	0.94	0.94
6	0.52	0.63	0.63	0.61	0.52	0.52	0.83	0.86	0.86	0.81	0.86	0.86
7	0.67	0.82	0.82	0.88	0.75	0.75	0.93	0.98	0.98	0.93	0.98	0.98

Table 3: Criteria comparison between KNN and Logistic Regression models

Looking across the table, Random Forest and the Stacking Ensemble consistently outperformed the other models. Both showed very strong results for forest types 4, 5, and 7, with high precision and recall. This means not only did they frequently identify these types correctly, but when they predicted them, they were almost always right. These forest types likely have clear, consistent patterns in the data, making them easier to classify.

The Stacking Ensemble, which combines the strengths of multiple models, delivered particularly high performance on forest types 3 through 5. For example, it achieved 97% accuracy for type 4 and 94% for type 5, making it one of the most reliable models for those categories. In contrast, KNN performed the weakest overall, especially for forest types 1, 2, and 3, where precision and recall were considerably lower. Logistic Regression performed moderately well but lacked the flexibility of tree-based models to capture more complex patterns in the data.

Forest types 1 and 2 continue to be more challenging for all models, which may suggest that they share similar environmental features with other types, making them harder to distinguish even for more advanced models.

Summary & Future Work

This project provided hands-on experience in applying machine learning techniques to real-world environmental data. By building and comparing four different models, we were able to see the strengths and limitations of each approach. The KNN and Logistic Regression models helped establish interpretable baselines and revealed that even simpler models can produce meaningful

predictions when properly tuned. However, their limitations, especially KNN's inability to handle categorical data capped its performance.

Random Forest emerged as our strongest standalone model, able to capture complex patterns and interactions across both categorical and numerical variables. Its built-in feature importance scores also gave us insights into which variables (like elevation and soil type) mattered most in determining forest cover. The Stacking Ensemble, powered by a neural network, matched the Random Forest in performance, showing that it was probably the biggest factor in deciding which cover type to predict.

Still, challenges remain. Certain forest types (like Types 1 and 2) proved harder to classify, likely due to overlapping environmental characteristics. This highlights opportunities for deeper exploration, such as grouping features in new ways or enriching the dataset with additional context (e.g., climate or vegetation cover).

In future iterations, we plan to:

- Explore new model types (e.g., boosting, decision tree).
- Investigate more advanced feature engineering (e.g., clustering soil types or creating vegetation zones).
- Use cross-validation techniques to further validate generalizability.

Ultimately, this project sharpened our ability to build, evaluate, and interpret machine learning models — and emphasized the importance of balancing accuracy with transparency when building tools for real-world use.

```
MODEL PERFORMANCE COMPARISON
> cat("------------------------------------------------\n")
------------------------------------------------
> cat(sprintf("1. KNN (k = 5):              Error Rate = %.4f\n", knn_error))
1. KNN (k = 5):              Error Rate = 0.4036
> cat(sprintf("2. Logistic Regression:      Error Rate = %.4f\n", lr_error))
2. Logistic Regression:      Error Rate = 0.2937
> cat(sprintf("3. Random Forest (100 trees): Error Rate = %.4f\n", rf_error))
3. Random Forest (100 trees): Error Rate = 0.1437
> cat(sprintf("4. Stacked Model:            Error Rate = %.4f\n", stack_error))
4. Stacked Model:            Error Rate = 0.1440
> cat("------------------------------------------------\n")
------------------------------------------------
```

Figure 4: Final Model Error Rates