Forest Cover Type Prediction

By Venkat, Pawan, Varshitha, Ruchitha, Harsha

Business Problem

- Effective forest management and conservation require accurate predictions of forest cover types. Despite advancements in technology and data availability, the lack of precise predictive models poses a significant challenge.

- Developing robust machine learning algorithms capable of reliably predicting forest cover types based on diverse environmental data sources remains a critical need.

- This business problem statement aims to address the imperative for accurate, scalable, and interpretable predictive models that support informed decision-making in forestry practices, biodiversity conservation, risk mitigation, and sustainable land use planning.

OBJECTIVES:

- Develop an AI-driven system capable of accurately predicting forest cover types using machine learning algorithms. This system should prioritize interpretability, aiming to provide insights and explanations for predictions, enabling better-informed forest management decisions.
- The objective includes enhancing the model's ability to interpret and communicate the rationale behind predictions, thereby aiding forest management practices, conservation efforts, risk assessment, and land use planning within the forestry domain.

Solution Approach

- Machine Learning – Classification (We have implemented almost 14 Algorithms)

- Logistic Regression
- Decision Tree
- Random Forest
- Gradient Boosting
- Support Vector Machine
- K-Nearest neighbors
- Naïve Bayes
- AdaBoost
- Bagging Classifier
- Extra Trees
- Stochastic Gradient Descent
- Quadratic Discriminant Analysis
- Artificial Neural Network (ANN).

Scope

- **The scope of addressing the business problem statement related to accurate forest cover type prediction for effective forest management and conservation involves several key aspects:**

1. **Algorithm Development:** Create precise machine learning models for forest cover type prediction.
2. **Data Collection and Integration:** Gather diverse environmental data sources, like satellite imagery and climate data, for model training.
3. **Data Preprocessing and Feature Engineering:** Clean and process collected data, identifying key features for prediction.
4. **Model Training and Validation:** Train models, optimize parameters and rigorously validate performance.
5. **Interpretability and Explainability:** Ensure models offer understandable insights into their predictions.
6. **Scalability and Generalizability:** Ensure models can handle diverse ecosystems and regions.
7. **Implementation and Deployment:** Apply models to practical forest management systems. Continuously assess and update models for better accuracy.
8. **Evaluation and Continuous Improvement:**

- **Time Line: 15 days**
- **1. Collecting and Integrating data – 2 days**
- **2. Data preparation – 2 days**
- **3. Model training and validation – 4 days**
- **4. Scaling & Generalization – 2 days**
- **5. Implementing and Deployment – 2 days**
- **6. Evaluation and improvising – 3 days**

Agile Method

- Plan: Break down tasks for data collection, preprocessing, and model development into manageable items. Rank tasks based on dependencies, focusing on high-value features early on.
- Code: Implement data preprocessing and model building incrementally, emphasizing team collaboration. Using Google colab to manage the code versions and facilitate collaboration.
- Train: Collect diverse environmental datasets, extracting relevant features for accurate predictions. Develop machine learning models progressively, starting simple and refining based on feedback.
- Test: Thoroughly test individual components (preprocessing steps, model algorithms) for functionality.
- Deploy: Prepare models for practical deployment in forest management systems. Ensure required servers, databases, and interfaces are ready to support deployment.

Data Sources & Data Understanding

- 15k data is trained. In this, we have included geological variables like elevation, slope, aspect, soil type, Wilderness Area, Cover type, hill shades (time to time), Horizontal and Vertical distances to Hydrology, etc. We ensure that the data utilized for model development is suitable, comprehensive, and aligned with the objectives of forest cover type prediction.

01

Data Preprocessing

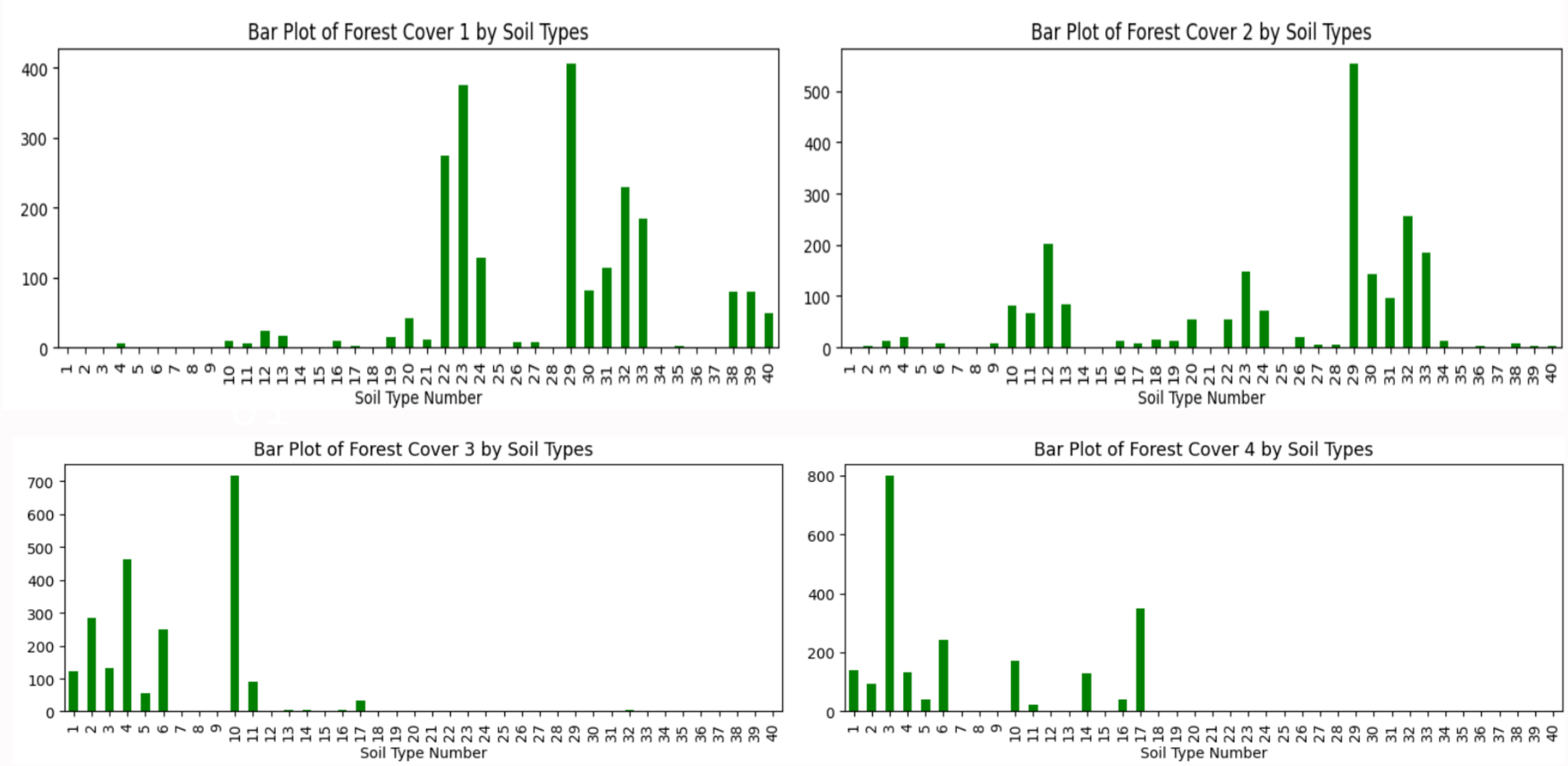
Cleaning data, removing duplicates, and outliers from the data, dealing with null values. i.e.,

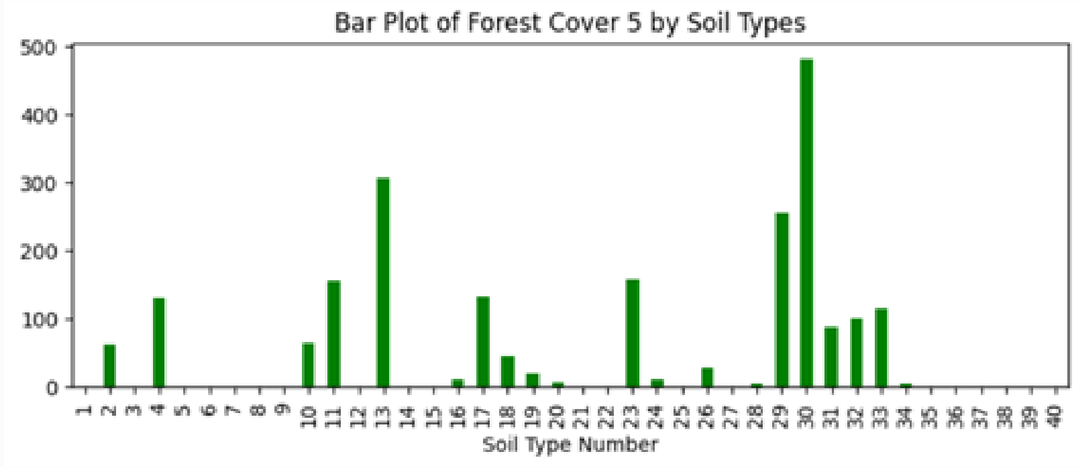
- Remove Duplicates: Ensuring data integrity by removing duplicate entries.
- Address Missing Values: Imputing missing data using appropriate methods.
- Outlier Treatment: Detecting and handle outliers to prevent skewing the models.

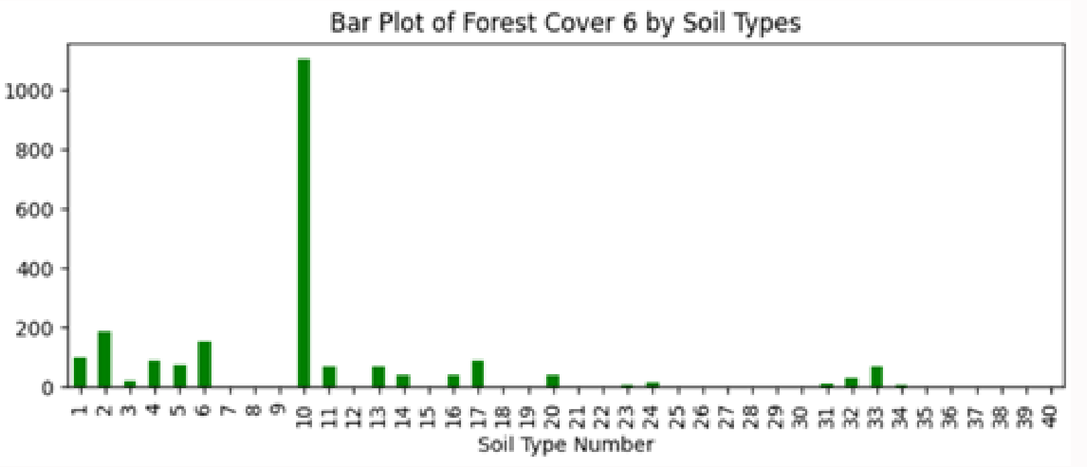
Data Visualization

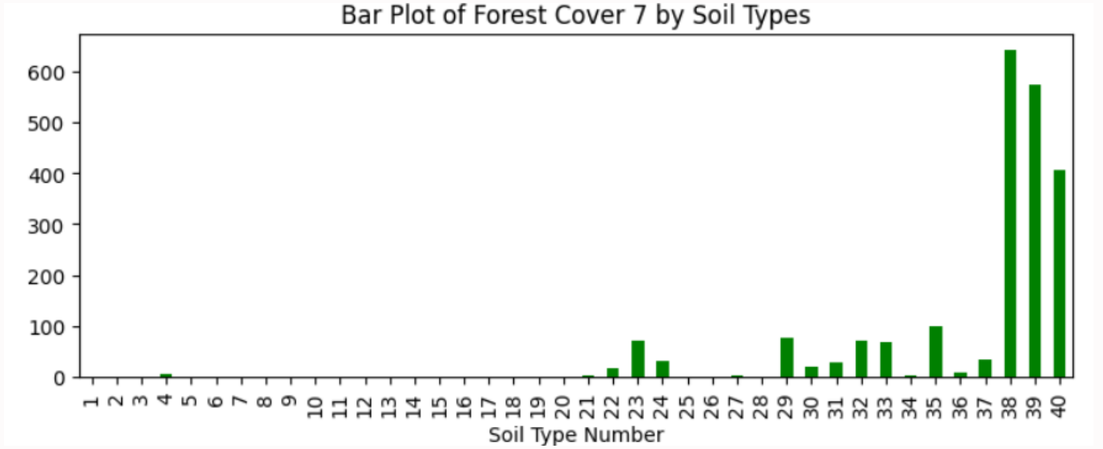
Visualizing forest cover type data in Python can provide insights into the distribution and relationships within the dataset.
data visualization could serve various purposes:
Exploratory Data Analysis (EDA): Understanding the distribution and relationships within the dataset.

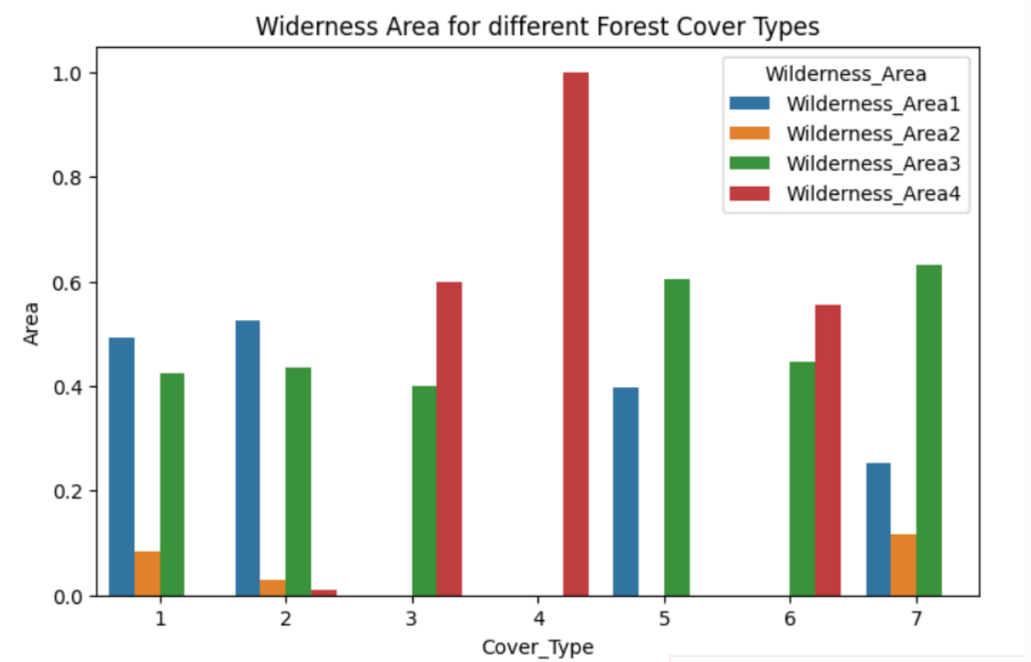
- Common visualizations here might include:
- Histograms: For each attribute/feature to understand their distributions.
- Scatterplots: To observe relationships between different attributes.




Bar Plot of Forest Cover 5 by Soil Types
Soil Type Number


Bar Plot of Forest Cover 6 by Soil Types
Soil Type Number


Bar Plot of Forest Cover 7 by Soil Types
Soil Type Number


Widerness Area for different Forest Cover Types
Wilderness_Area
Wilderness_Area1
Wilderness_Area2
Wilderness_Area3
Wilderness_Area4
Area
Cover_Type

AUTO EDA

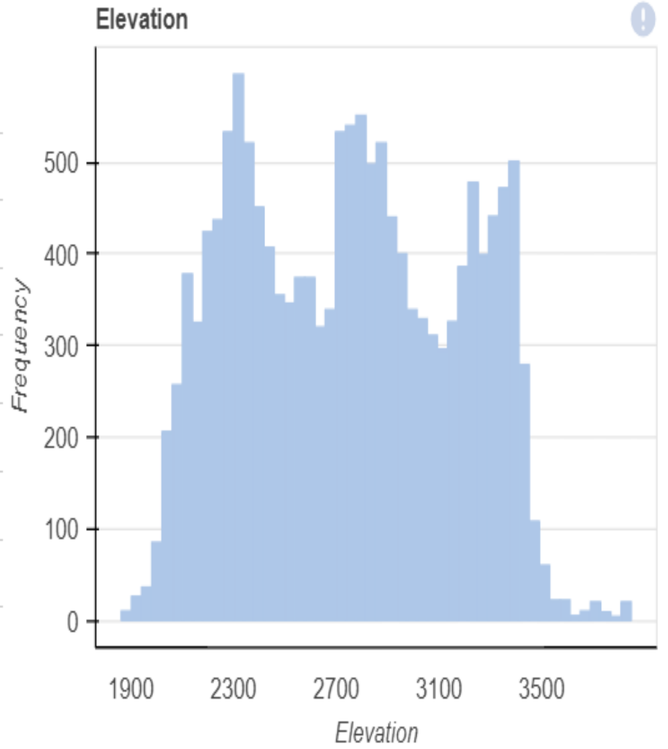
Auto Exploratory Data Analysis (Auto EDA) involves automating the process of exploring and understanding a dataset without extensive manual intervention. For forest cover type prediction, here's a general workflow for auto EDA and data preparation:

Elevation Column Auto EDA:

Elevation
numerical

Show Details

Approximate Distinct Count	1665	Mean	2749.3226
Approximate Unique (%)	11.0%	Minimum	1863
Missing	0	Maximum	3849
Missing (%)	0.0%	Zeros	0
Infinite	0	Zeros (%)	0.0%
Infinite (%)	0.0%	Negatives	0
Memory Size	241920	Negatives (%)	0.0%

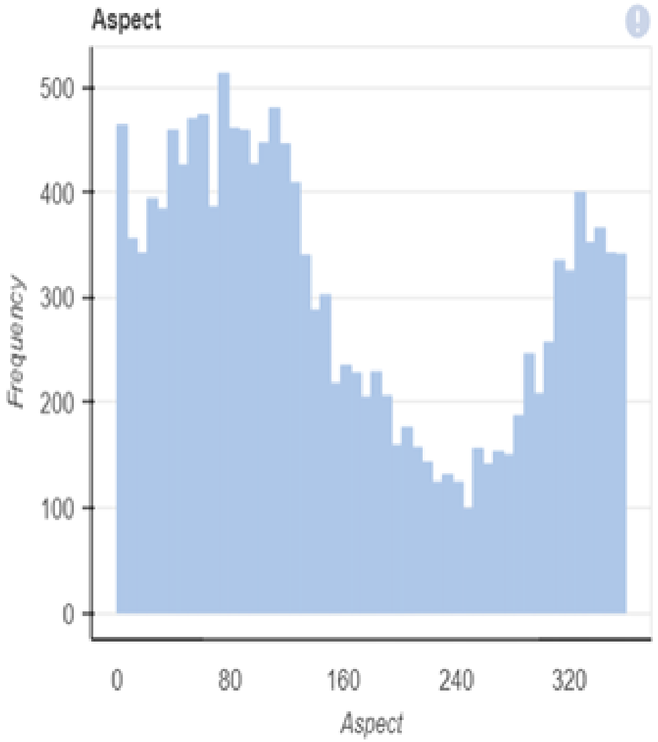


Aspect Column Auto EDA:

Aspect
numerical

Show Details

Approximate Distinct Count	361	Mean	156.6767
Approximate Unique (%)	2.4%	Minimum	0
Missing	0	Maximum	360
Missing (%)	0.0%	Zeros	110
Infinite	0	Zeros (%)	0.7%
Infinite (%)	0.0%	Negatives	0
Memory Size	241920	Negatives (%)	0.0%

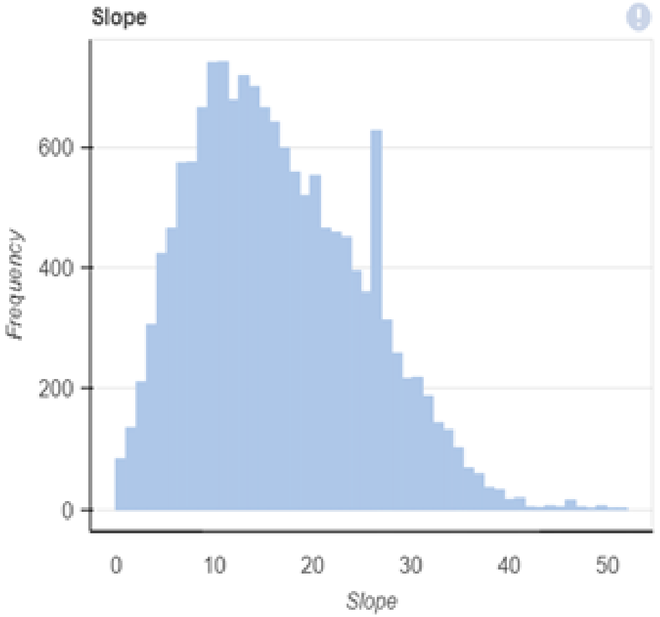


Slope Column Auto EDA:

Slope
numerical

Show Details

Approximate Distinct Count	52	Mean	16.5016
Approximate Unique (%)	0.3%	Minimum	0
Missing	0	Maximum	52
Missing (%)	0.0%	Zeros	5
Infinite	0	Zeros (%)	0.0%
Infinite (%)	0.0%	Negatives	0
Memory Size	241920	Negatives (%)	0.0%



Data Loading:

- Load the forest cover dataset into a programming environment (Python, R, etc.). Ensure the data is in a format compatible with the chosen analysis tools.

Basic Data Overview:

- Utilize automated functions or libraries (such as Pandas Profiling, AutoViz, or SweetViz in Python) to generate a summary report of the dataset.
- Explore the dataset's structure, basic statistics (mean, median, min, max), data types, missing values, and distributions of features.

Visualization:

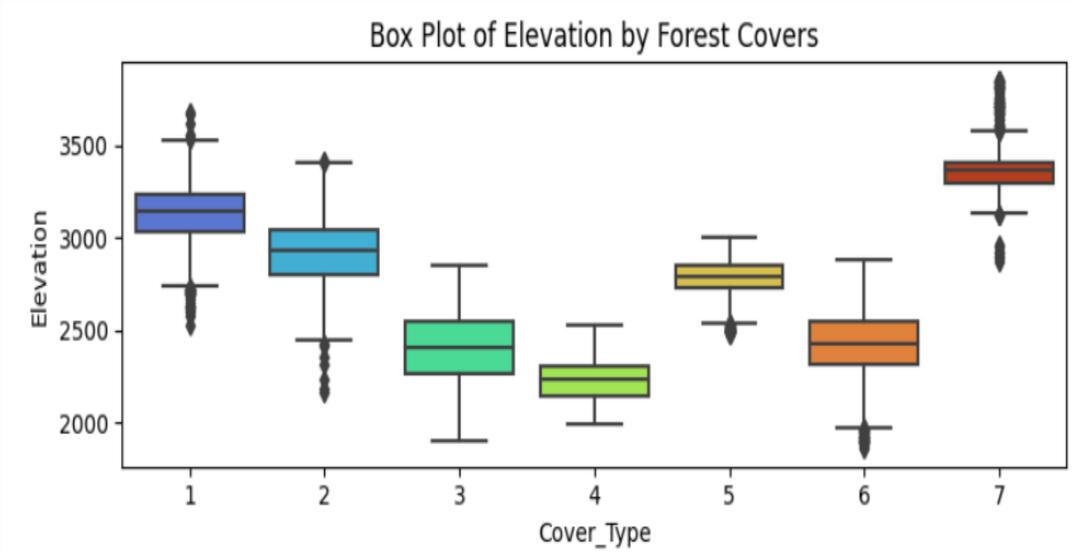
- Generate visualizations automatically to understand relationships between variables. This includes histograms, scatter plots, box plots, correlation matrices, etc., to reveal patterns and potential insights.
- For forest cover type prediction, visualize features like elevation, soil type, wilderness area, and their distributions across different cover types.

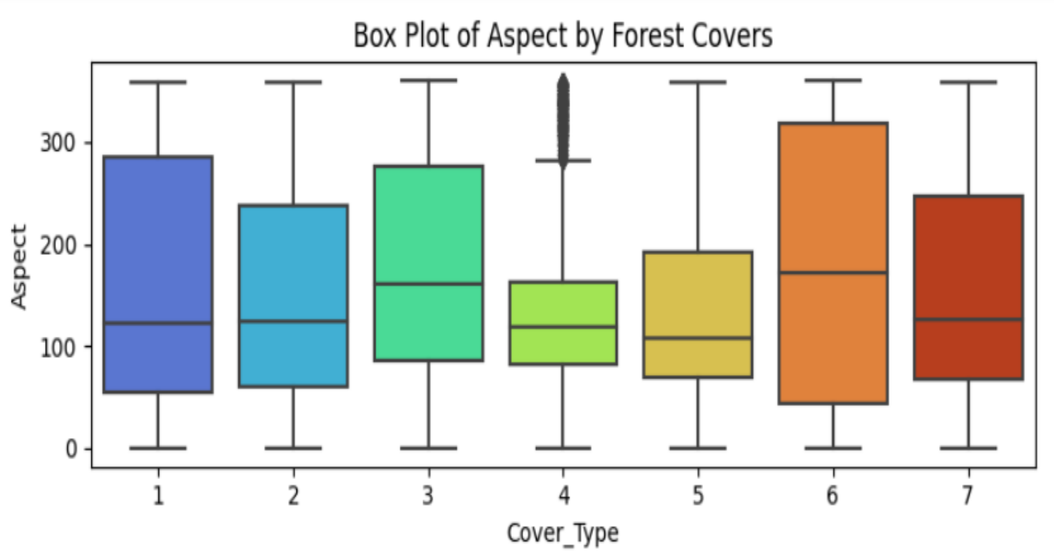
Feature Engineering:

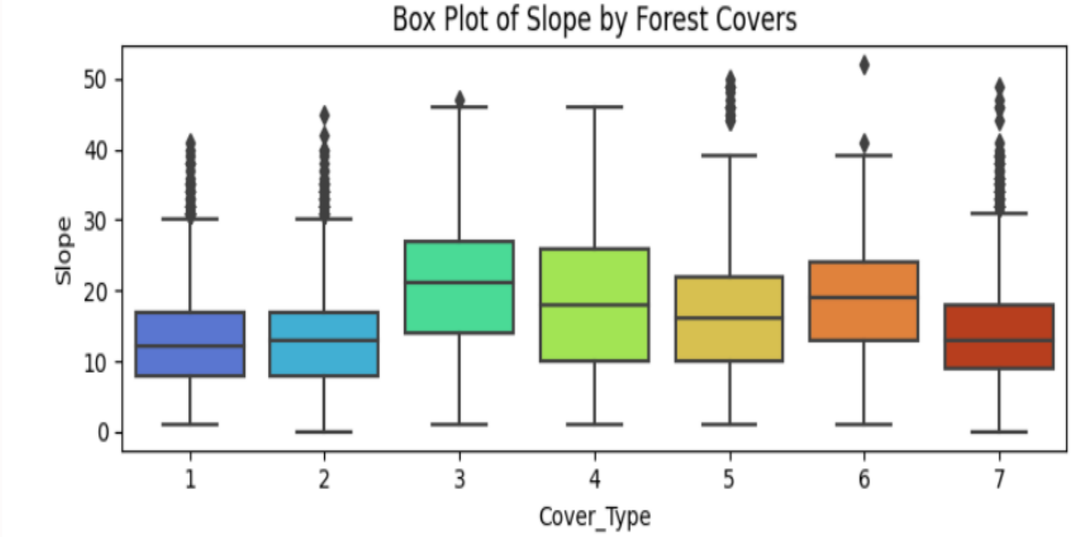
- Automatically derive or generate new features based on existing ones. For instance, combining or transforming variables to create more informative features that could enhance the predictive power of the model.
- Identify categorical variables and encode them appropriately (e.g., one-hot encoding) for machine learning algorithms to interpret them correctly.

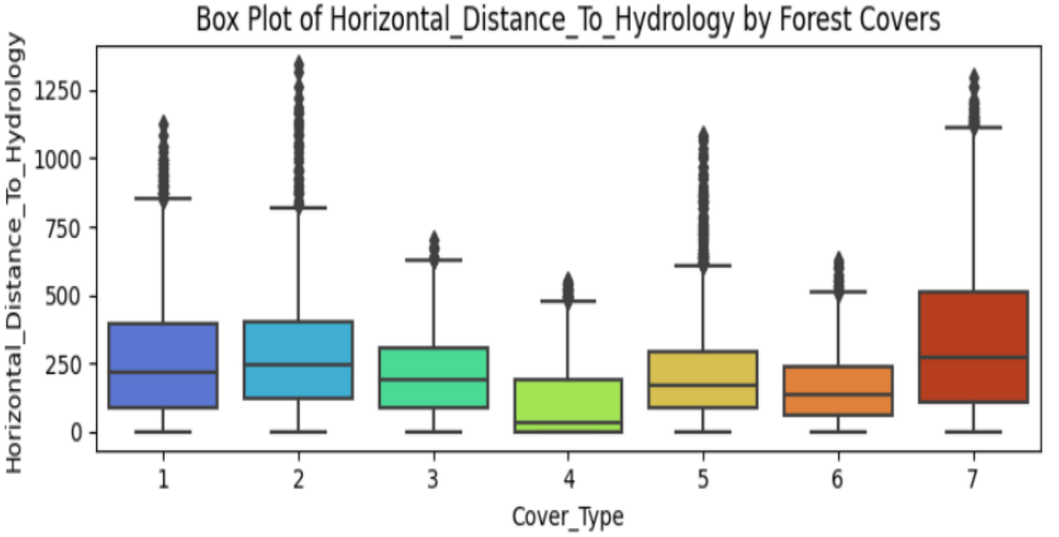
Handling Missing Values and Outliers:

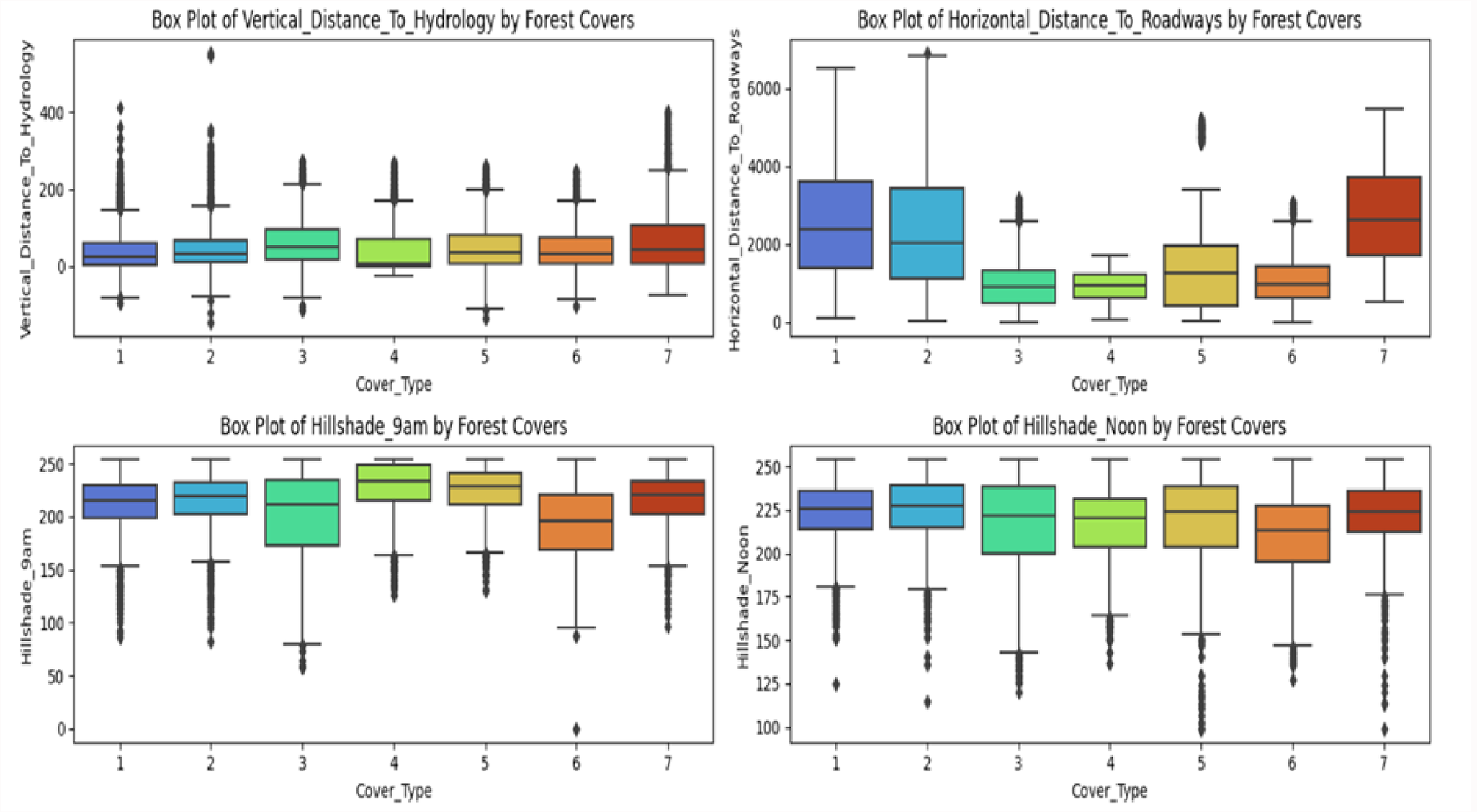
- Auto EDA tools can assist in identifying missing values and suggesting strategies to handle them, such as imputation techniques or considering removal based on impact. Detect outliers and consider automated methods to address them, either by capping/extending ranges or utilizing algorithms robust to outliers.


Box Plot of Elevation by Forest Covers
Elevation
3500
3000
2500
2000
1
2
3
4
5
6
7
Cover_Type


Box Plot of Aspect by Forest Covers
Aspect
300
200
100
0
1
2
3
4
5
6
7
Cover_Type

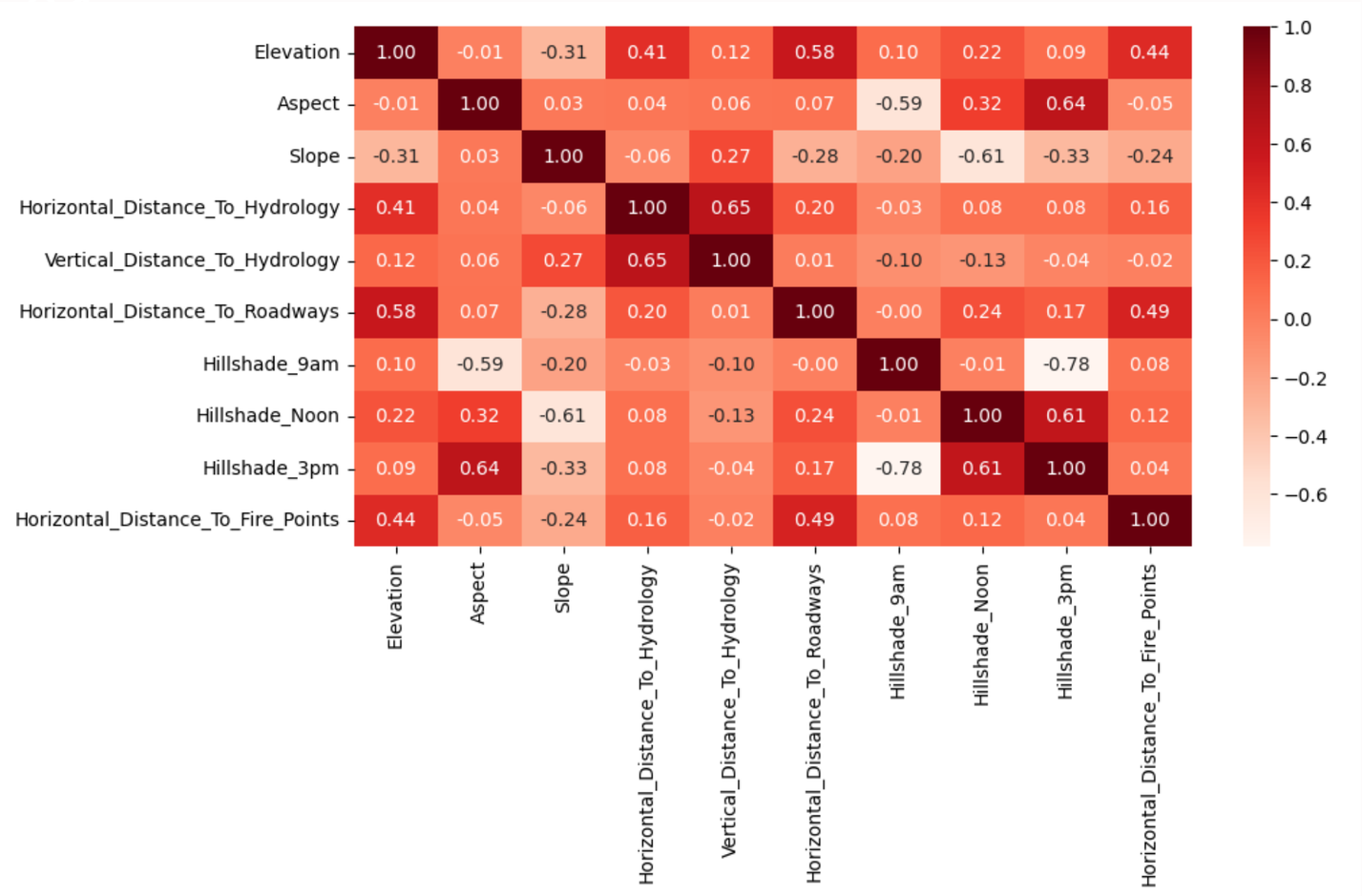

Box Plot of Slope by Forest Covers
Slope
50
40
30
20
10
0
1
2
3
4
5
6
7
Cover_Type


Box Plot of Horizontal_Distance_To_Hydrology by Forest Covers
Horizontal_Distance_To_Hydrology
1250
1000
750
500
250
0
1
2
3
4
5
6
7
Cover_Type


Box Plot of Vertical_Distance_To_Hydrology by Forest Covers
Vertical_Distance_To_Hydrology
Cover_Type
Box Plot of Horizontal_Distance_To_Roadways by Forest Covers
Horizontal_Distance_To_Roadways
Cover_Type
Box Plot of Hillshade_9am by Forest Covers
Hillshade_9am
Cover_Type
Box Plot of Hillshade_Noon by Forest Covers
Hillshade_Noon
Cover_Type

Correlation Analysis:

- Automatically identify correlations between different features and the target variable (forest cover type). This analysis helps understand which variables might have a stronger influence on predicting the cover type.



Train-Test Split:

- Split the dataset into training and testing sets automatically to ensure the model's evaluation is done on unseen data.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state=100)
```

01

Data Preprocessing:

- Normalize or scale features if required by the chosen machine learning algorithm. Auto EDA tools may suggest suitable preprocessing steps based on the characteristics of the dataset.

```
train_df.drop(columns = 'Id', inplace = True) #Dropping ID column
train_df.isnull().sum() #Checking Null values
```

```
Elevation                           0
Aspect                              0
Slope                               0
Horizontal_Distance_To_Hydrology    0
Vertical_Distance_To_Hydrology      0
Horizontal_Distance_To_Roadways     0
Hillshade_9am                       0
Hillshade_Noon                      0
Hillshade_3pm                       0
Horizontal_Distance_To_Fire_Points  0
Wilderness_Area1                    0
Wilderness_Area2                    0
Wilderness_Area3                    0
Wilderness_Area4                    0
Soil_Type1                          0
Soil_Type2                          0
Soil_Type3                          0
Soil_Type4                          0
Soil_Type5                          0
Soil_Type6                          0
Soil_Type7                          0
Soil_Type8                          0
Soil_Type9                          0
Soil_Type10                         0
Soil_Type11                         0
Soil_Type12                         0
Soil_Type13                         0
Soil_Type14                         0
Soil_Type15                         0
Soil_Type16                         0
Soil_Type17                         0
Soil_Type18                         0
Soil_Type19                         0
Soil_Type20                         0
```

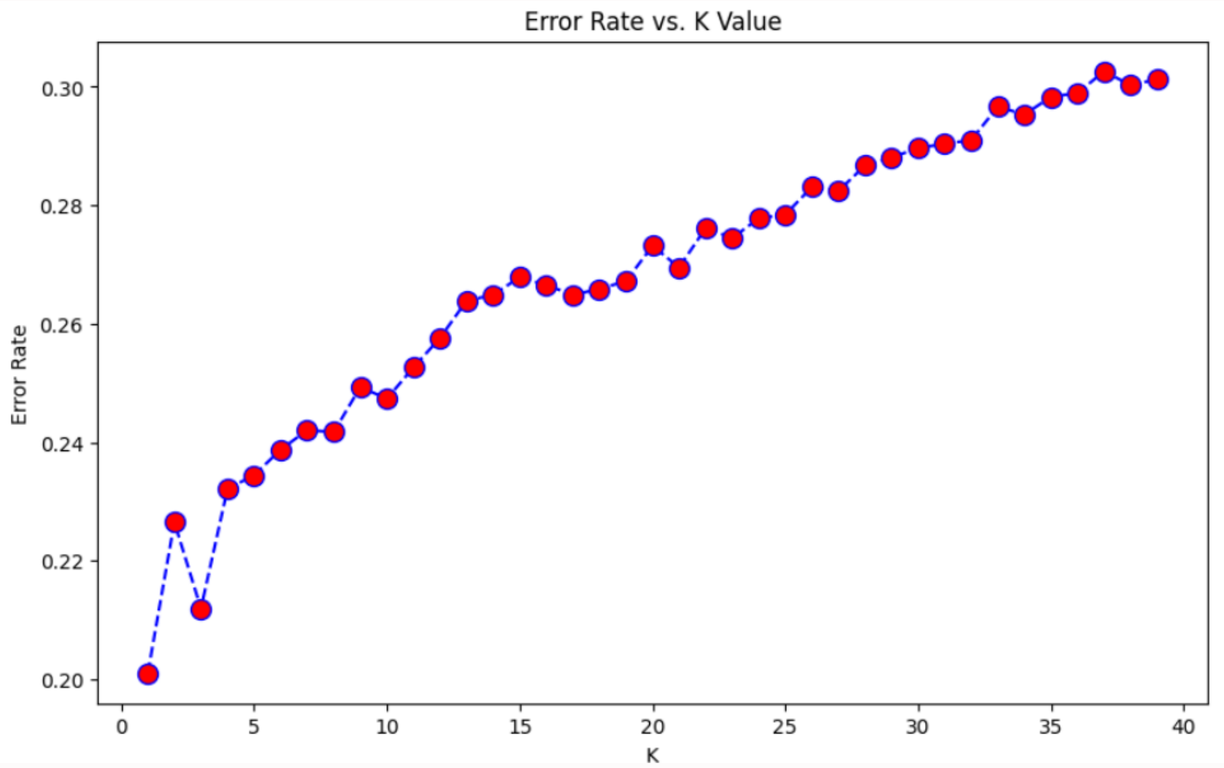
Model Training and Model Testing:

```
#K-Nearest Neighbour
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(x_train,y_train)
pred_knn = knn.predict(x_test)
accuracy = accuracy_score(y_test, pred_knn)
print("KNN : " ,accuracy)
print(classification_report(y_test,pred_knn))
```

```
KNN :  0.798941798941799
              precision    recall  f1-score   support

           1       0.72      0.69      0.70       446
           2       0.73      0.63      0.67       474
           3       0.77      0.75      0.76       393
           4       0.88      0.92      0.90       399
           5       0.83      0.91      0.87       438
           6       0.76      0.78      0.77       445
           7       0.90      0.95      0.92       429

    accuracy                           0.80      3024
   macro avg       0.80      0.80      0.80      3024
weighted avg       0.80      0.80      0.80      3024
```



Random Forest Classifier Model New value Prediction:

New Value Prediction

```
[ ] continuous_input = [2699,347,3,0,0,2096,213,234,159,6853]

    df_continuous_input = pd.DataFrame([continuous_input], columns=['Elevation', 'Aspect', 'Slope', 'Horizontal_Distance_To_Hydrology', 'Vertical_Distance_To_Hydrology', 'Horizontal_Dista

    df_continuous_input
```

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade_9am	Hillshade_Noon	Hillshade_3pm	Horizontal
0	2699	347	3	0	0	2096	213	234	159	

```
[ ] from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    scaler.fit(df_continuous_input)
    scaled_continuous_features = scaler.transform(df_continuous_input)
    scaled_continuous_features
    df_scaled_continuous_features = pd.DataFrame(scaled_continuous_features, columns=['Elevation', 'Aspect', 'Slope', 'Horizontal_Distance_To_Hydrology', 'Vertical_Distance_To_Hydrology',
```

```
    binary_input = [0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    df_binary_input = pd.DataFrame([binary_input], columns=['Soil_Type1', 'Soil_Type2', 'Soil_Type3',
           'Soil_Type4', 'Soil_Type5', 'Soil_Type6', 'Soil_Type7', 'Soil_Type8',
           'Soil_Type9', 'Soil_Type10', 'Soil_Type11', 'Soil_Type12',
           'Soil_Type13', 'Soil_Type14', 'Soil_Type15', 'Soil_Type16',
           'Soil_Type17', 'Soil_Type18', 'Soil_Type19', 'Soil_Type20',
           'Soil_Type21', 'Soil_Type22', 'Soil_Type23', 'Soil_Type24',
           'Soil_Type25', 'Soil_Type26', 'Soil_Type27', 'Soil_Type28',
           'Soil_Type29', 'Soil_Type30', 'Soil_Type31', 'Soil_Type32',
           'Soil_Type33', 'Soil_Type34', 'Soil_Type35', 'Soil_Type36',
           'Soil_Type37', 'Soil_Type38', 'Soil_Type39'-, 'Soil_Type40'])
```

```
[ ] new_pred_df =  pd.concat([df_scaled_continuous_features, df_binary_input], axis = 1)
    new_pred_df
    new_pred_df.shape
```

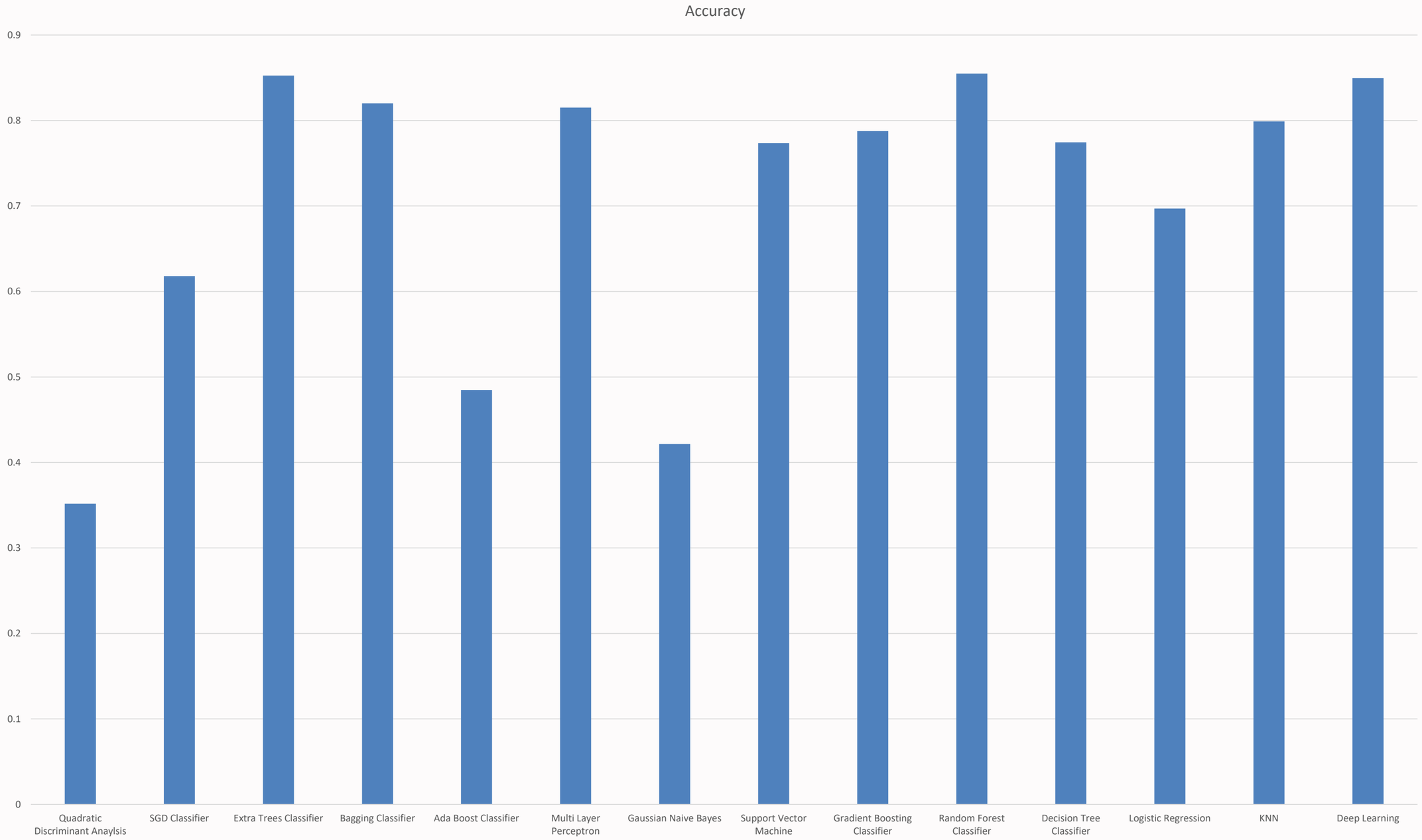
```
    (1, 50)
```

```
[ ] rfc.predict(new_pred_df)
```

```
    array([5])
```

Graph on Model accuracy comparison:

Model	Accuracy
Quadratic Discriminant Analysis	0.35185185185185186
SGD Classifier	0.6180555555555556
Extra Trees Classifier	0.8525132275132276
Bagging Classifier	0.8201058201058201
Ada Boost Classifier	0.48478835978835977
Multi Layer Perceptron	0.8151455026455027
Gaussian Naive Bayes	0.42162698412698413
Support Vector Machine	0.7734788359788359
Gradient Boosting Classifier	0.7876984126984127
Random Forest Classifier	0.8548280423280423
Decision Tree Classifier	0.7744708994708994
Logistic Regression	0.6970899470899471
KNN	0.798941798941799
Deep Learning	0.8495370149612427


Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Quadratic Discriminant Anaylsis
SGD Classifier
Extra Trees Classifier
Bagging Classifier
Ada Boost Classifier
Multi Layer Perceptron
Gaussian Naive Bayes
Support Vector Machine
Gradient Boosting Classifier
Random Forest Classifier
Decision Tree Classifier
Logistic Regression
KNN
Deep Learning

Finalized Model Details:

- Random Forest is a robust and versatile ensemble learning algorithm widely used for classification tasks, especially when handling complex datasets like predicting forest cover types. This algorithm combines the strength of multiple decision trees, mitigating overfitting and enhancing predictive accuracy. By aggregating predictions from numerous decision trees trained on different subsets of the data and features, Random Forest minimizes the risk of individual tree biases and variance, resulting in a more stable and accurate model.

- In the context of forest cover type prediction, the Random Forest algorithm excels in capturing intricate relationships between various environmental features—such as elevation, soil type, and vegetation indices—to classify different forest cover types effectively. Its ability to handle large datasets with numerous input variables and maintain predictive power makes it a preferred choice. Notably, in my experimentation with various models for this task, the Random Forest algorithm consistently outperformed others, yielding an impressive accuracy of over 85.48%. This high accuracy demonstrates the algorithm's effectiveness in accurately predicting the diverse cover types within forest regions.

- The success of Random Forest in surpassing other models underscores its capability to handle complex data relationships, mitigate overfitting, and generalize well to unseen data. Its adaptability to different feature types and robustness against noise make it a formidable tool for forest cover type prediction. The superior accuracy attained by the Random Forest model reinforces its significance and utility in addressing the intricacies of predicting forest cover types, thereby contributing to informed land management and ecological studies.

```python
# Random Forest

rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)
pred_rfc = rfc.predict(x_test)
accuracy = accuracy_score(y_test, pred_rfc)
print("Random Forest Classifier : ",accuracy)
print(classification_report(y_test,pred_rfc))
```

```
Random Forest Classifier :  0.8548280423280423
              precision    recall  f1-score   support

           1       0.81      0.75      0.78       446
           2       0.80      0.70      0.75       474
           3       0.80      0.84      0.82       393
           4       0.93      0.96      0.94       399
           5       0.87      0.95      0.91       438
           6       0.85      0.83      0.84       445
           7       0.92      0.97      0.95       429

    accuracy                           0.85      3024
   macro avg       0.85      0.86      0.85      3024
weighted avg       0.85      0.85      0.85      3024
```

Deploy:

Flask App:

Flask is a lightweight web framework for Python that's used to build web applications. It's known for its simplicity and flexibility, making it a popular choice for developing web applications and APIs. Here are some key aspects of Flask:

Key Features:

Routing: Flask uses decorators to define routes and map them to specific functions (view functions) that handle the requests.

HTTP Request Handling: It supports various HTTP methods like GET, POST, PUT, DELETE, etc., allowing you to handle different types of requests.

Templates: Flask integrates Jinja2 templates, enabling the creation of dynamic HTML content by rendering data within templates.

Extensions: Flask has a modular design and provides a wide range of extensions for tasks like form validation, database integration (SQLAlchemy), user authentication, and more.

Development Server: It includes a built-in development server, which is convenient for testing and development.

```python
#pip install flask
from flask import Flask, render_template, request
import pickle
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler


# Loading the mlr model
model = pickle.load(open('model.pkl', 'rb'))

# Flask is used for creating your application
# render template is used for rendering the HTML page
app = Flask(__name__) # your application

@app.route('/') # default route
def home():
    return render_template('index.html') # rendering your home page.


@app.route('/pred', methods=['POST']) # prediction route
def predict1():
    '''
    For rendering results on HTML
    '''

    Elevation = request.form.get('elevation')
    Aspect = request.form.get('aspect')
    Slope = int(request.form.get('slope'))
    Horizontal_Distance_To_Hydrology = int(request.form.get('hordt-hyd'))
    Vertical_Distance_To_Hydrology = int(request.form.get('verdt-hyd'))
    Horizontal_Distance_To_Roadways = int(request.form.get('hordt-road'))
    Hillshade_9am = int(request.form.get('hillshade9'))
    Hillshade_Noon = int(request.form.get('hillshade12'))
```

```
    Hillshade_9am = int(request.form.get('hillshade9'))
    Hillshade_Noon = int(request.form.get('hillshade12'))
    Hillshade_3pm = int(request.form.get('hillshade3'))
    Horizontal_Distance_To_Fire_Points = int(request.form.get('hordt-fire'))
    soil_type = request.form.get('soilType')

    # Create a DataFrame from the form data
    scaler = StandardScaler()
    numerical_features = [Elevation, Aspect, Slope, Horizontal_Distance_To_Hydrology, Vertical_Distance_To_Hydrology, Horizo
                          Hillshade_3pm, Horizontal_Distance_To_Fire_Points]
    numerical_features_scaled = scaler.fit_transform([numerical_features])

        # Convert soil_type to a dataframe with 40 columns
    soil_type_list = list(soil_type)
    soil_type_df = pd.DataFrame([soil_type_list], columns=[f'Soil_Type{i}' for i in range(1, 41)])

        # Concatenate the numerical features and soil_type dataframes
    input_df = pd.DataFrame(np.concatenate([numerical_features_scaled, soil_type_df], axis=1))

        # Make prediction using the pre-trained model
    prediction = model.predict(input_df)
    print(prediction)

    outputs = ["Spruce/Fir","Lodgepole Pine","Ponderosa Pine","Cottonwood/Willow","Aspen","Douglas-fir","Krummholz"]

    return render_template("index.html", result="The predicted cover is " + outputs[prediction[0]-1]+"!")


# running your application
if __name__ == "__main__":
    app.run()
```

THANK YOU