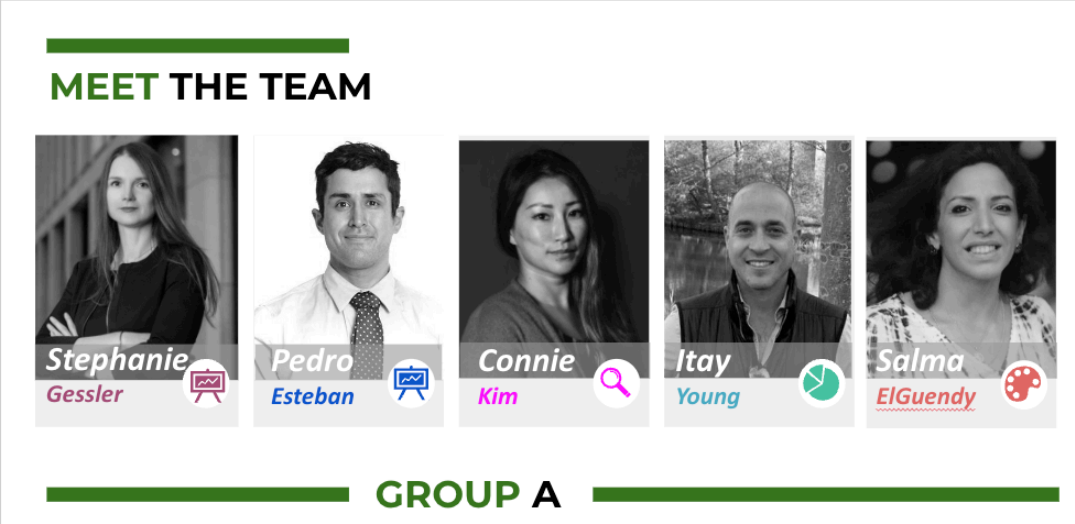
Machine Learning II
Group Assignment

Group A:



Stephanie Gessler, Pedro V. Esteban, Itay Young, Salma ElGuendy, Connie Kim

Introduction

The assignment is focused on solving the Forest Cover Type Prediction: https://www.kaggle.com/c/forest-cover-type-prediction/overview. This task proposes a classification problem: predict the forest cover type (the predominant kind of tree cover) from strictly cartographic variables (as opposed to remotely sensed data).

The study area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. Each observation is a 30m x 30m patch. You are asked to **predict an integer classification** for the forest cover type. The seven types are:

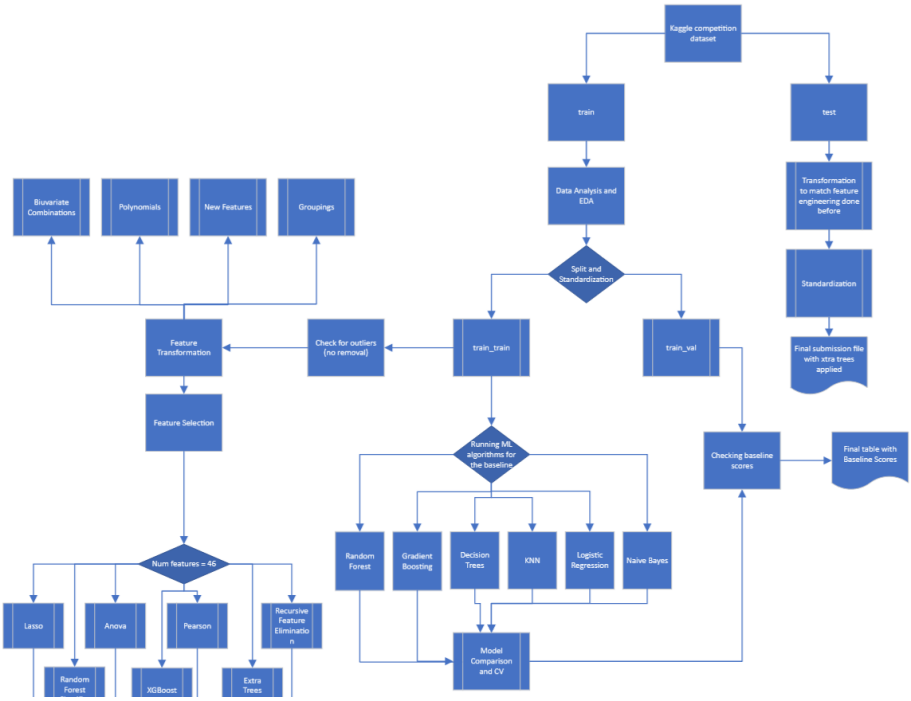
1. Spruce/Fir
2. Lodgepole Pine
3. Ponderosa Pine
4. Cottonwood/Willow

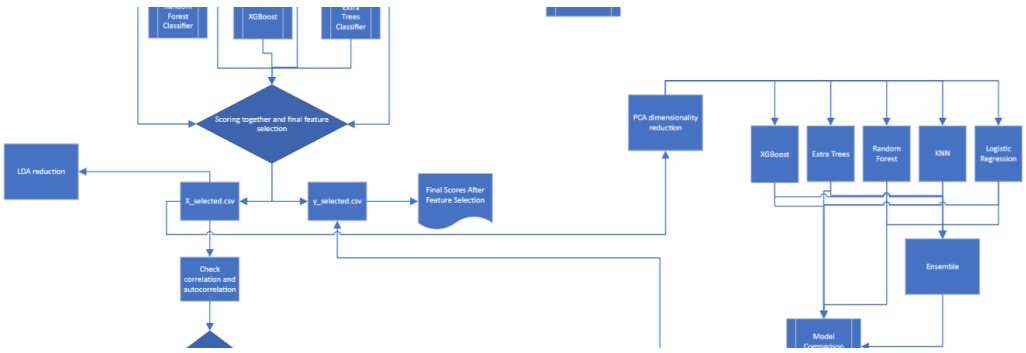
5. Aspen
6. Douglas-fir
7. Krummholz

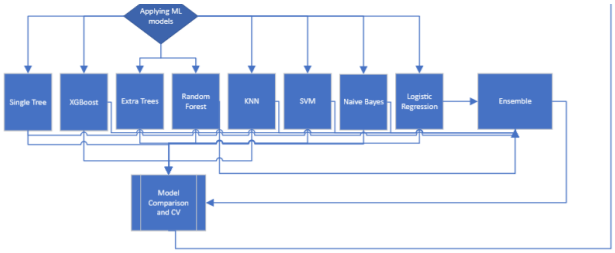
The training set (15120 observations) contains both features and the Cover_Type. The test set contains only the features.

You must predict the Cover_Type for every row in the test set (565892 observations).

Our workflow







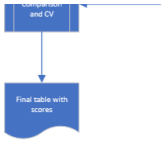


Table of contents

- 0. Libaries used
- 1. Import Data
- **2. Data analysis**
  - 2.1.Explanation of variables
  - 2.2.Label types
- **3. Exploratory Data Analysis**
  - 3.1 Analysis of the Dataset using EDA
  - 3.2 Normal Test
  - 3.3 Checking Variable Completeness

- 3.3 Checking Variable Completeness
- 3.4 Correlation Matrix
- 3.5 Paired density, scatterplot matrix and 3D Graphics
- 3.6 Categorial EDA
  - 3.6.1 Categorial Bar Diagrams
  - 3.6.2.Violinplot with Dependent Variable
  - 3.6.3.Treemap for categorial Data

- **4. Baseline Model**
  - 4.1 Prepare Data and Standardization
  - 4.2 Random Forest
  - 4.3 Gradient Boosting
  - 4.4 Decision Trees
  - 4.5 K-Nearest Neighbors (KNN)
  - 4.6 Logistic Regression
  - 4.7 Naive Bayes
  - 4.8 Model Comparison

0. Libaries used

In [99]:
```
#!pip install squarify
#!pip install yellowbrick
#!pip install plotly
#!pip install plotly richdem
```

In [100…
```
import pandas as pd
import numpy as np
import plotly.express as px
import seaborn as sns  # Graphing
import matplotlib.pyplot as plt
import plotly.io as pio
import squarify #treemap
import os
import matplotlib
import warnings
```

```
import richdem as rd
```

In [101…

```
from numpy import mean, std

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix, classificat
from sklearn.metrics import classification_report

from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.preprocessing import StandardScaler


from scipy.stats import normaltest

from pandas_profiling import ProfileReport

from yellowbrick.classifier import ROCAUC
from yellowbrick.classifier import ClassPredictionError
from yellowbrick.style.palettes import PALETTES, SEQUENCES, color_palette



warnings.simplefilter(action='ignore', category=FutureWarning)
```

1.Import the Data

Let’s load the training data and create data frame

In [102…

```
data_train = pd.read_csv("train.csv")
data_train.head()
```

Out[102]:

	Id	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To
0	1	2596	51	3	258	
1	2	2590	56	2	212	
2	3	2804	139	9	268	
3	4	2785	155	18	242	
4	5	2595	45	2	153	

5 rows × 56 columns

Let's keep the original dataset for later comparisons and make a backup copy

In [103…
```
df_original = data_train.copy()
```

Let's have a first feeling on the labels and rows

In [104…
```
data_train.tail()
```

Out[104]:

	Id	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Dist
15115	15116	2607	243	23	258	
15116	15117	2603	121	19	633	
15117	15118	2492	134	25	365	
15118	15119	2487	167	28	218	
15119	15120	2475	197	34	319	

5 rows × 56 columns

We load the test dataset to compare with the train

In [105…
```
data_test = pd.read_csv("test.csv")
data_test.head()
```

Out[105]:

	Id	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance
0	15121	2680	354	14	0	
1	15122	2683	0	13	0	
2	15123	2713	16	15	0	
3	15124	2709	24	17	0	
4	15125	2706	29	19	0	

5 rows × 55 columns

Inmediatly we see one less column, the predictor is gone

Let's get some insights regarding stats on the test dataset

In [106…
```
data_test.describe()
```

Out[106]:

	Id	Elevation	Aspect	Slope	Horizontal_Dist
count	565892.000000	565892.000000	565892.000000	565892.000000	
mean	298066.500000	2964.977407	155.629558	14.039635	
std	163359.093603	273.157030	111.962120	7.450155	
min	15121.000000	1859.000000	0.000000	0.000000	
25%	156593.750000	2818.000000	58.000000	9.000000	
50%	298066.500000	2999.000000	127.000000	13.000000	

50%	298066.500000	2999.000000	127.000000	13.000000
75%	439539.250000	3164.000000	260.000000	18.000000
max	581012.000000	3858.000000	360.000000	66.000000

8 rows × 55 columns

From comparing the train and test datasets we get the following conclusions:

- Difference between train and test datasets appear to be the last column with the cover type where prediction (y) has to be made
- Soil types are not consistent between train and test. Some soil types are unique of train and others only appear in test

2. Data analysis

To understand dataframe, let's look at the data types, the data shape and descriptive statistics. All features are integers.

__2.1.Explanation of variables__

Let's gain some domain knowledge by understanding more in detail what are all labels in the dataset for:

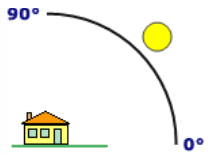
Elevation / quantitative /meters / Elevation in meters
Aspect / quantitative / azimuth / Aspect in degrees azimuth

The azimuth is the angular direction of the sun, measured from north in clockwise degrees from 0 to 360. An azimuth of 90 degrees is east. The default azimuth is 315 degrees (NW).

Default sun azimuth (direction) for hillshade is 315°

Slope / quantitative / degrees / Slope in degrees

The altitude is the slope or angle of the illumination source above the horizon. The units are in degrees, from 0 (on the horizon) to 90 (overhead). The default is 45 degrees.



Default sun altitude for hillshade is 45°

Horizontal_Distance_To_Hydrology / quantitative / meters / Horz Dist to nearest surface water features
Vertical_Distance_To_Hydrology / quantitative / meters / Vert Dist to nearest surface water features
Horizontal_Distance_To_Roadways / quantitative / meters / Horz Dist to nearest roadway
Hillshade_9am / quantitative / 0 to 255 index / Hillshade index at 9am, summer solstice
Hillshade_Noon / quantitative / 0 to 255 index / Hillshade index at noon, summer soltice
Hillshade_3pm / quantitative / 0 to 255 index / Hillshade index at 3pm, summer solstice

The algorithm for calculating the hillshade value is as follows:

```
(1)  Hillshade = 255.0 * ((cos(Zenith_rad) * cos(Slope_rad)) +
                (sin(Zenith_rad) * sin(Slope_rad) * cos(Azimuth_rad - Aspect_rad)))
```

Note that if the calculation of the hillshade value is < 0, the output cell value will be = 0.

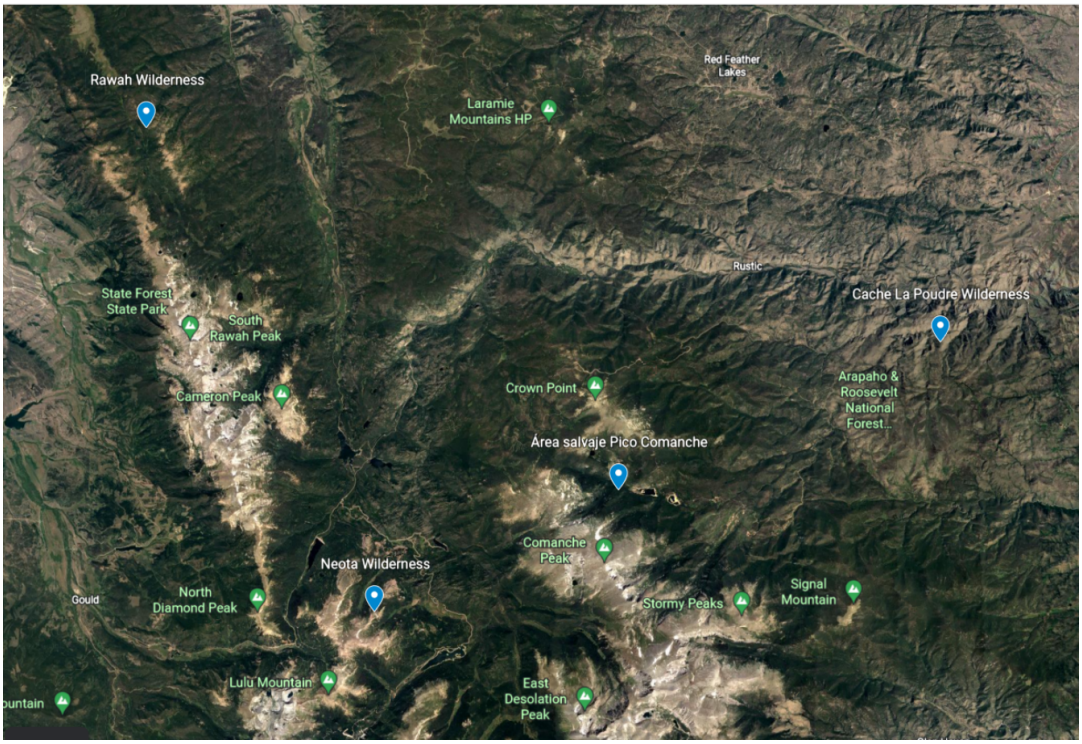
Horizontal_Distance_To_Fire_Points / quantitative / meters / Horz Dist to nearest wildfire ignition points
Wilderness_Area (4 binary columns) / qualitative / 0 (absence) or 1 (presence) / Wilderness area designation
Soil_Type (40 binary columns) / qualitative / 0 (absence) or 1 (presence) / Soil Type designation
Cover_Type (7 types) / integer / 1 to 7 / Forest Cover Type designation

Source: https://desktop.arcgis.com/en/arcmap/10.3/tools/spatial-analyst-toolbox/how-hillshade-works.htm

The wilderness areas are:

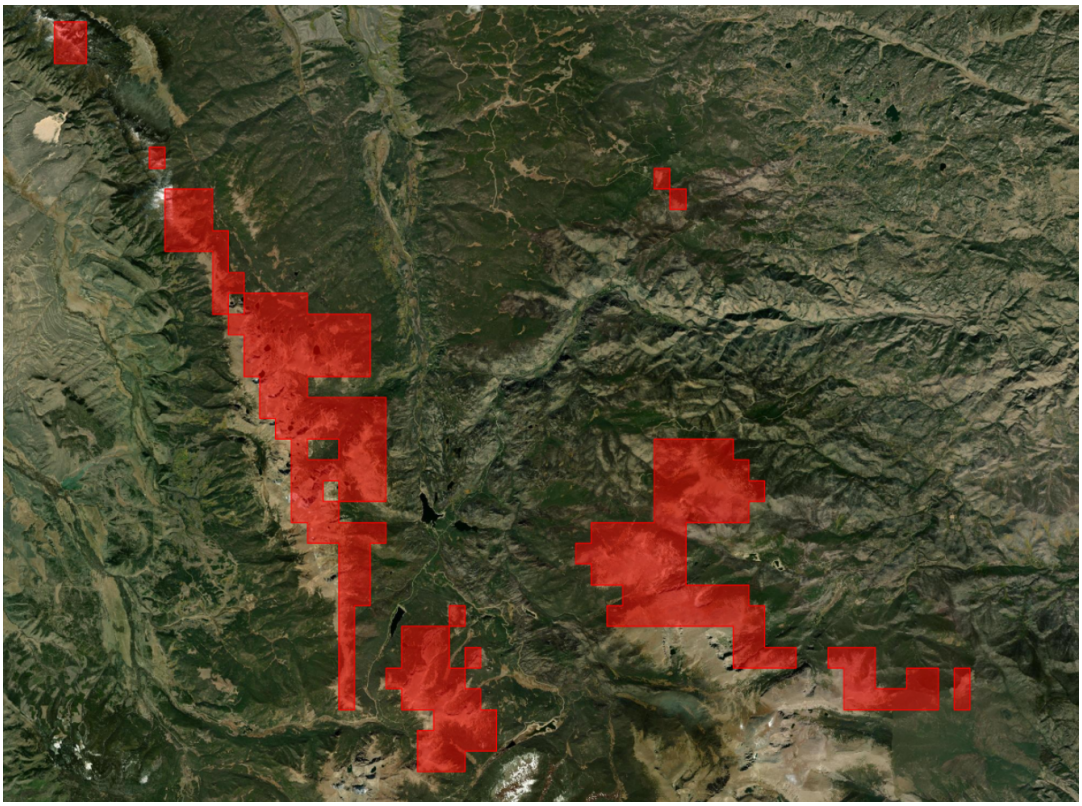
1 - Rawah Wilderness Area
2 - Neota Wilderness Area
3 - Comanche Peak Wilderness Area
4 - Cache la Poudre Wilderness Area

Here we can see a satellite view of the areas in question:



Source: Google My Maps, own made map based on wilderness areas

We can observe also one area, Cache La Poudre with an obvious difference in terrain type from the aerial



Example of Moran soil mapping, source:https://casoilresource.lawr.ucdavis.edu/sde/?series=moran#extent From this resource we can map soil types in the US forest parks. With the Moran families, we can see that this particular soil type is common in

parks. With the Moran families, we can see that this particular soil type is common in all of Roosevelt National park which might suggest data has been drilled down to the

most atomic level.

We can also use Google's Earth distance tool to understand how separated these areas are from one another. An estimated distance between areas to the centre of each is 25 km distance between nearest areas.

Based on the book "Statistical Methods and Applications in Forestry and Environmental Sciences" by Chandra et al. (2020), Chapter (Forest Cover-Type Prediction Using Model Averaging) DOI: 10.1007/978-981-15-1476-0, we group the soil types based on the USFS ecological land type units (ELUs), and supplementary provide the details of first digit for climatic zone and second digit for geologic zone of USFS ELU code. This goes inline with the atomicity problem laid down before for the soil types:

The soil types based on USFS ELUs code:

</tr>

</tr>

</tr>

Source: https://www.kaggle.com/competitions/forest-cover-type-prediction/data

First digit	Climatic zone	Second digit	Geologic zones
1.	Lower montane dry	1.	Alluvium
2.	Lower montane	2.	Glacial
3.	Montane dry	3.	Shale
4.	Montane	4.	Sandstone
5.	Montane dry and montane	5.	Mixed sedimentary
6.	Montane and subalpine	6.	Unspecified in the USFS ELU survey
7.	Subalpine	7.	Igneous and metamorphic
8.	Alpine	8.	Volcanic

__2.2 Label types__

Now, we check the types, and shape of the dataset to decide next steps in the pipeline

In [107…
```
data_train.dtypes
```

Out[107]:
```
Id                                    int64
Elevation                             int64
Aspect                                int64
Slope                                 int64
Horizontal_Distance_To_Hydrology      int64
Vertical_Distance_To_Hydrology        int64
Horizontal_Distance_To_Roadways       int64
```

```
Hillshade_9am                         int64
Hillshade_Noon                        int64
Hillshade_3pm                         int64
Horizontal_Distance_To_Fire_Points    int64
Wilderness_Area1                      int64
Wilderness_Area2                      int64
Wilderness_Area3                      int64
Wilderness_Area4                      int64
Soil_Type1                            int64
Soil_Type2                            int64
Soil_Type3                            int64
Soil_Type4                            int64
Soil_Type5                            int64
Soil_Type6                            int64
Soil_Type7                            int64
Soil_Type8                            int64
Soil_Type9                            int64
Soil_Type10                           int64
Soil_Type11                           int64
Soil_Type12                           int64
Soil_Type13                           int64
Soil_Type14                           int64
Soil_Type15                           int64
Soil_Type16                           int64
Soil_Type17                           int64
Soil_Type18                           int64
Soil_Type19                           int64
Soil_Type20                           int64
Soil_Type21                           int64
Soil_Type22                           int64
Soil_Type23                           int64
Soil_Type24                           int64
Soil_Type25                           int64
Soil_Type26                           int64
Soil_Type27                           int64
Soil_Type28                           int64
Soil_Type29                           int64
Soil_Type30                           int64
Soil_Type31                           int64
Soil_Type32                           int64
Soil_Type33                           int64
Soil_Type34                           int64
Soil_Type35                           int64
Soil_Type36                           int64
Soil_Type37                           int64
Soil_Type38                           int64
Soil_Type39                           int64
Soil_Type40                           int64
Cover_Type                            int64
dtype: object
```

We have 56 features with the test set consist of 15120 rows.

In [108…

```
data_train.shape
```

Out[108]:

```
(15120, 56)
```

We have 56 columns: the target variable Cover Type and 55 more features to work with. Let's analyze a little bit more each of them to check if we need to clean or preprocess them.

In [109…
```
data_train.columns
```

Out[109]:
```
Index(['Id', 'Elevation', 'Aspect', 'Slope',
       'Horizontal_Distance_To_Hydrology', 'Vertical_
Distance_To_Hydrology',
       'Horizontal_Distance_To_Roadways', 'Hillshade_
9am', 'Hillshade_Noon',
       'Hillshade_3pm', 'Horizontal_Distance_To_Fire_
Points',
       'Wilderness_Area1', 'Wilderness_Area2', 'Wilde
rness_Area3',
       'Wilderness_Area4', 'Soil_Type1', 'Soil_Type2'
, 'Soil_Type3',
       'Soil_Type4', 'Soil_Type5', 'Soil_Type6', 'Soi
l_Type7', 'Soil_Type8',
       'Soil_Type9', 'Soil_Type10', 'Soil_Type11', 'S
oil_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',
       'Cover_Type'],
      dtype='object')
```

Vertical_Distance_To_Hydrology seems to have positive and negative values. Checking the negative values we observed that few are lying on an average altitude of Denver, which is 1500 metres, these values are likely patches bellow a water deposit in the mountain.

3. Exploratory Data Analysis

3.1. Analysis of the dataset using EDA

The pandas profiling gives us the information that the data is formatted and clean:

- There are no null values and all features are numeric.
- We have 12 numeric values and 44 Categorial values.
- The soil and wilderness area are already one-hot-encoded columns.
- The alert show us that Soiltype 7 and Soiltype 15 are always zero.
- There are several highly correlated features as well.

In [110…

```
report = ProfileReport(data_train, minimal=False)
report
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/
s]
Generate report structure:   0%|          | 0/1 [00:00
<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	56

Number of observations	15120
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	6.5 MiB
Average record size in memory	448.0 B

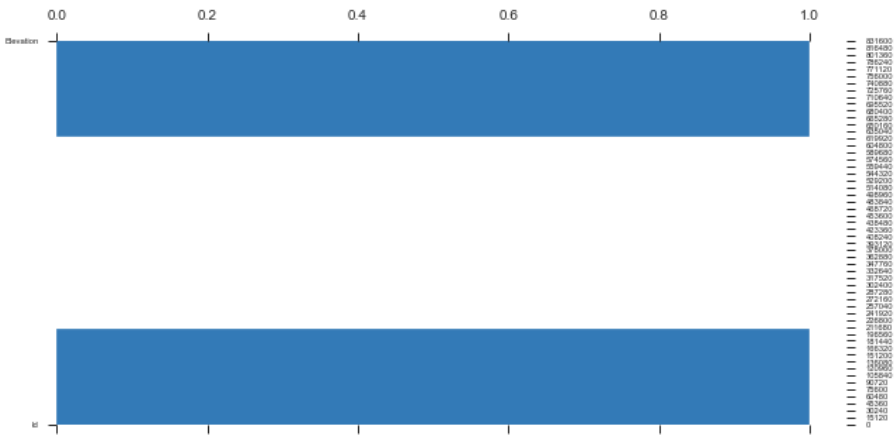
Variable types

Numeric	12
Categorical	44

Alerts

`Soil_Type7` has constant value "0"	**Constant**
`Soil_Type15` has constant value "0"	**Constant**
`Elevation` is highly correlated with `Horizontal_Distance_To_Roadways`	**High correlation**

Out[110]:



- **ID**: As unique values, consider dropping it from the analysis.
- **Elevation**: Has very few values above 3500meters, which makes sense as the higher the altitude the less trees grow.
- **Aspect**: Has high correlation with the Hillshade variables
- **Slope**: Has high correlation with the Hillshade variables, there seem to be values outside the norm around 25degrees

- **Cover_Type**: Cover Type are equal distributed, each Cover Type are represented in an equally.
- **Soil_Type15** and **Soil_Type7**: No values, consider removing them but first check if these are also exist in the testing data to confirm our thought.
- **Soil_Type28,36,25,9 and 8**: Very few values, seems to be rather scare soil types.

We name the Kaggle test dataset data_set and the train one data_train

In [111…
```
data_test = pd.read_csv("test.csv")
```

In [112…
```
data_test['Soil_Type15'].value_counts()
```

Out[112]:
```
0    565889
1         3
Name: Soil_Type15, dtype: int64
```

In [113…
```
data_test['Soil_Type7'].value_counts()
```

Out[113]:
```
0    565787
1       105
Name: Soil_Type7, dtype: int64
```

Checking if the Soil_Type 15 and Soil_Type7 should be removed based on the test data. Since both do exist with values in the test set, we cannot test these as these do not exist in the test set. As we are bucketizing soil types, these won't have any effect in the new categories engineered

The numerical variables seem to not be normal distributed. All variables though are not normally distributed.

To ascertain this, we will use the statistical D'Agostino and Pearson's test below. The following conclusions are made:

- All features are skewed to the right .
- The features that measure distance are positively skewed.
- Hillshade 9 am and Noon have a similar left-skewed distribution.
- None of the features follow a normal distribution, which is confirmed by the statistical Normal test for normality.

3.2. Normal Test

The numerical variables seem to not be normally distributed in the

The numerical variables seem to not be normally distributed in the Pandas Profiling Report. However we will test this with a normal test below and it confirms the initial analysis that the features do not follow a Gaussian Distribution at a significance level of 95%. All features are skewed.

In [114…

```
alpha = 0.05
for i in
(['Elevation','Aspect','Slope','Horizontal_Distance_To_Hydr

    print ()
    print (i)
    stat, p = normaltest(data_train[[i]])
    print('Statistics=%.3f, p=%.3f' % (stat, p))
    if p > alpha:
        print('The data follows Gaussian Distribution
(fail to reject H0)')
    else:
        print('The data does not follow Gaussian
Distribution  (reject H0)')
```

```
Elevation
Statistics=5560.372, p=0.000
The data does not follow Gaussian Distribution  (rejec
t H0)

Aspect
Statistics=9254.059, p=0.000
The data does not follow Gaussian Distribution  (rejec
t H0)

Slope
Statistics=659.561, p=0.000
The data does not follow Gaussian Distribution  (rejec
t H0)

Horizontal_Distance_To_Hydrology
Statistics=4118.478, p=0.000
The data does not follow Gaussian Distribution  (rejec
t H0)

Vertical_Distance_To_Hydrology
Statistics=4454.210, p=0.000
The data does not follow Gaussian Distribution  (rejec
t H0)

Horizontal_Distance_To_Roadways
Statistics=2766.766, p=0.000
The data does not follow Gaussian Distribution  (rejec
t H0)

Hillshade_9am
```

```
Statistics=2427.368, p=0.000
The data does not follow Gaussian Distribution  (rejec
t H0)

Hillshade_Noon
Statistics=2021.620, p=0.000
The data does not follow Gaussian Distribution  (rejec
t H0)

Hillshade_3pm
Statistics=282.570, p=0.000
The data does not follow Gaussian Distribution  (rejec
t H0)

Horizontal_Distance_To_Fire_Points
Statistics=4660.738, p=0.000
The data does not follow Gaussian Distribution  (rejec
t H0)
```

3.3.Checking Variable Completeness

Since Wilderness is One Hot Encoded, we want to check if all the values sum up to 15120 or if we have values which are entered as zero or even 1, meaning the values are entered twice in two different Wilderness.

Below, it shows that values are all together suming up to 15120, which shows us the completeness of the Wilderness area and uniqueness of the wilderness area.

In [115…
```
data_train.iloc[:,11:15].sum(axis=1).sum()
```

Out[115]:
```
15120
```

Same for the Soil, we check if the data is adding up to 15120 to see if there are values entered as zero and actually give us missing information.

In [116…
```
data_train.iloc[:,15:55].sum(axis=1).sum()
```

Out[116]:
```
15120
```

Similarly, the data is complete and there are no entered values with zero.

3.4.Correlation Matrix

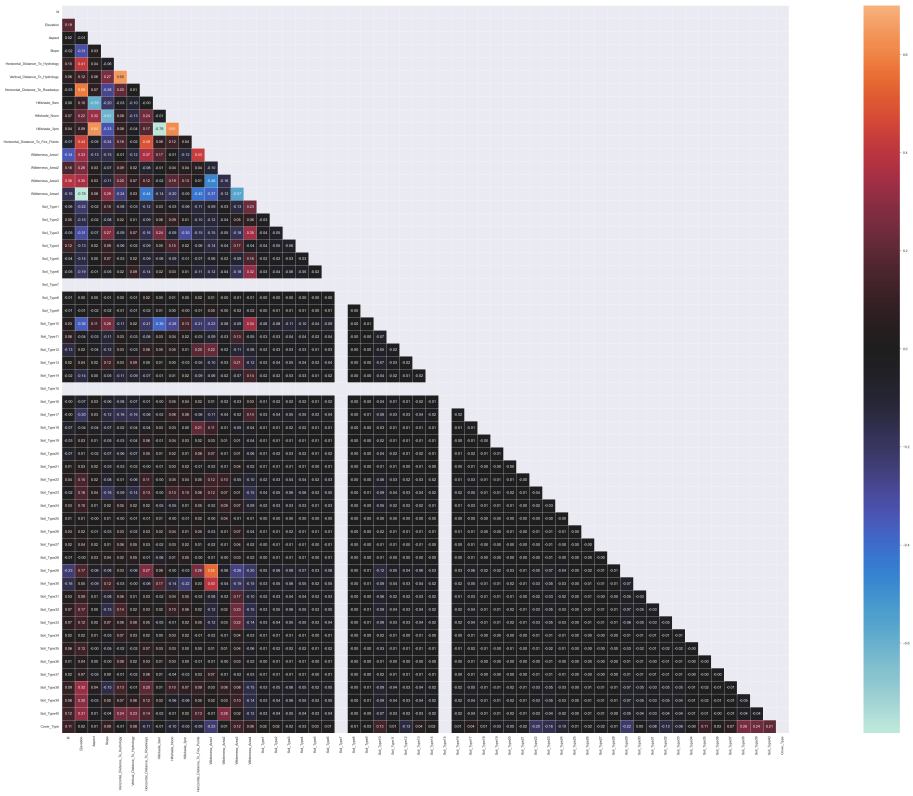
Looking at the features which are highly correlated with a threshold above 0.5:

above 0.5.

- **Vertical_Distance_To_Hydrology** positivly correlated to **Horizontal_Distance_To_Hydrology**
- **Horizontal_Distance_To_Roadways** positivly correlated to **Elevation**
- **Hillshade_9am** negativly correlated and **Hillshade_3pm** positivly correlated to **Aspect**
- **Hillshade_Noon** negativly correlated to **Slope**
- **Hillshade variables** are also correlated between themselves,but Hillshade 3pm and Hillshade 9am are negative correlated
- **Wilderness Area 4** is highly correlated with Elevation

In [117…

```
#Heatmap
matrix = data_train.corr()
mask = np.zeros_like(matrix)
mask[np.triu_indices_from(mask)] = True
fig, ax = plt.subplots(figsize=(120,60))
heatmap = sns.heatmap(matrix, center=0, fmt=".2f",
square=True, annot=True, linewidth=.9, mask =
mask,vmax=.7);
plt.show()
```



Correlated features beyond +-0.5 threshold:

- Horizontal Distance to roadways vs Elevation
- Vertical Distance to Hidrology vs Horizontal Distance to Hidrology
- Hillshade 9am with Aspect
- Hillshade Noon with Slope

- Hillshade Noon with Slope
- Hillshade 3pm with 9am and Noon
- Wilderness Area 4 and Aspect and Wilderness Area 3
- SoilType29 with Wilderness Area1

Performing reverse One Hot Encoding using Pandas idxmax. We transform Wilderness coded numbers and soil types back into categorical (text)

With this, we make later a dedicated EDA to understand better what Wilderness Area has which Soil type in a more direct fashion

In [118…

```
data_train['Wilderness'] = (data_train.iloc[:, 11:15] ==
1).idxmax(1)
data_train['Soil'] = (data_train.iloc[:, 15:56] ==
1).idxmax(1)
```

In [119…

```
data_train['Cover_Type1'] =
data_train['Cover_Type'].replace([1,2,3,4,5,6,7],
['Spruce/Fir','Lodgepole Pine','Ponderosa
Pine','Cottonwood/Willow','Aspen','Douglas-
fir','Krummholz'])
data_train['Wilderness'] =
data_train['Wilderness'].replace(['Wilderness_Area1','Wilde
['Rawah Wilderness Area ','Neota Wilderness
Area','Comanche Peak Wilderness Area','Cache la Poudre
Wilderness Area'])
data_train.head()
```

Out[119]:

Soil Type	USFS ELU Code	Description
1	**2702**	Cathedral family - Rock outcrop complex, extremely stony
2	***2703***	Vanet - Ratake families complex, very stony
3	**2704**	Haploborolis - Rock outcrop complex, rubbly
4	**2705**	Ratake family - Rock outcrop complex, rubbly
5	**2706**	Vanet family - Rock outcrop complex complex, rubbly
6	***2717***	Vanet - Wetmore families - Rock outcrop complex, stony
7	**3501**	Gothic family
8	**3502**	Supervisor - Limber families complex
9	**4201**	Troutville family, very stony
10	**4703**	Bullwark - Catamount families - Rock outcrop complex, rubbly
11	***4704***	Bullwark - Catamount families - Rock land complex, rubbly

12	**4744**	Legault family - Rock land complex, stony
13	**4758**	Catamount family - Rock land - Bullwark family complex, rubbly
14	**5101**	Pachic Argiborolis - Aquolis complex
15	**5151**	unspecified in the USFS Soil and ELU Survey
16	**6101**	Cryaquolis - Cryoborolis complex
17	***6102***	Gateview family - Cryaquolis complex
18	**6731**	Rogert family, very stony
19	**7101**	Typic Cryaquolis - Borohemists complex
20	**7102**	Typic Cryaquepts - Typic Cryaquolls complex
21	**7103**	Typic Cryaquolls - Leighcan family, till substratum complex
22	**7201**	Leighcan family, till substratum, extremely bouldery
23	***7202***	Leighcan family, till substratum - Typic Cryaquolls complex
24	**7700**	Leighcan family, extremely stony
25	**7701**	Leighcan family, warm, extremely stony
26	**7702**	Granile - Catamount families complex, very stony
27	**7709**	Leighcan family, warm - Rock outcrop complex, extremely stony
28	**7710**	Leighcan family - Rock outcrop complex, extremely stony
29	***7745***	Como - Legault families complex, extremely stony
30	**7746**	Como family - Rock land - Legault family complex, extremely stony
31	**7755**	Leighcan - Catamount families complex, extremely stony
32	**7756**	Catamount family - Rock outcrop - Leighcan family complex, extremely stony
33	**7757**	Leighcan - Catamount families - Rock outcrop complex, extremely stony
34	**7790**	Cryorthents - Rock land complex, extremely stony
35	***8703***	Cryumbrepts - Rock outcrop - Cryaquepts complex
36	**8707**	Bross family - Rock land - Cryumbrepts complex, extremely stony
37	**8708**	Rock outcrop - Cryumbrepts - Cryorthents complex, extremely stony
38	**8771**	Leighcan - Moran families - Cryaquolls complex, extremely stony
39	***8772***	Moran family - Cryorthents - Leighcan family complex, extremely stony
40	**8776**	Moran family - Cryorthents - Rock land complex, extremely stony

	Id	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To
0	1	2596	51	3	258	0
1	2	2590	56	2	212	-6

2	3	2804	139	9	268	65
3	4	2785	155	18	242	118
4	5	2595	45	2	153	-1

5 rows × 59 columns

In [120…

```
data_test['Wilderness'] = (data_test.iloc[:, 11:15] == 1).idxmax(1)
data_test['Soil'] = (data_test.iloc[:, 15:56] == 1).idxmax(1)
data_test['Wilderness'] = data_test['Wilderness'].replace(['Wilderness_Ar
```

3.5.Paired density, scatterplot matrix and 3D Graphics

In seaborn we can observe the following:

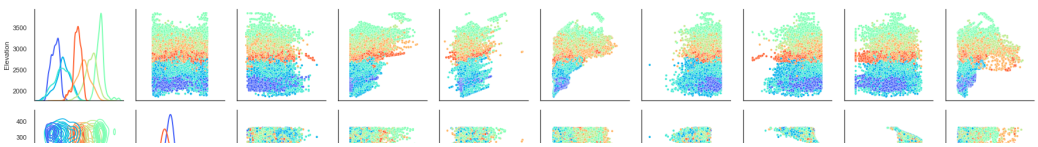
- Elevation differentiate the cover types very well
- Slope, almost all Cover Types overlap
- Very skewed distributions of Horizontal/Vertical Distance to Hydrology, Horizontal Distance to Roadways and Distance to Firepoints

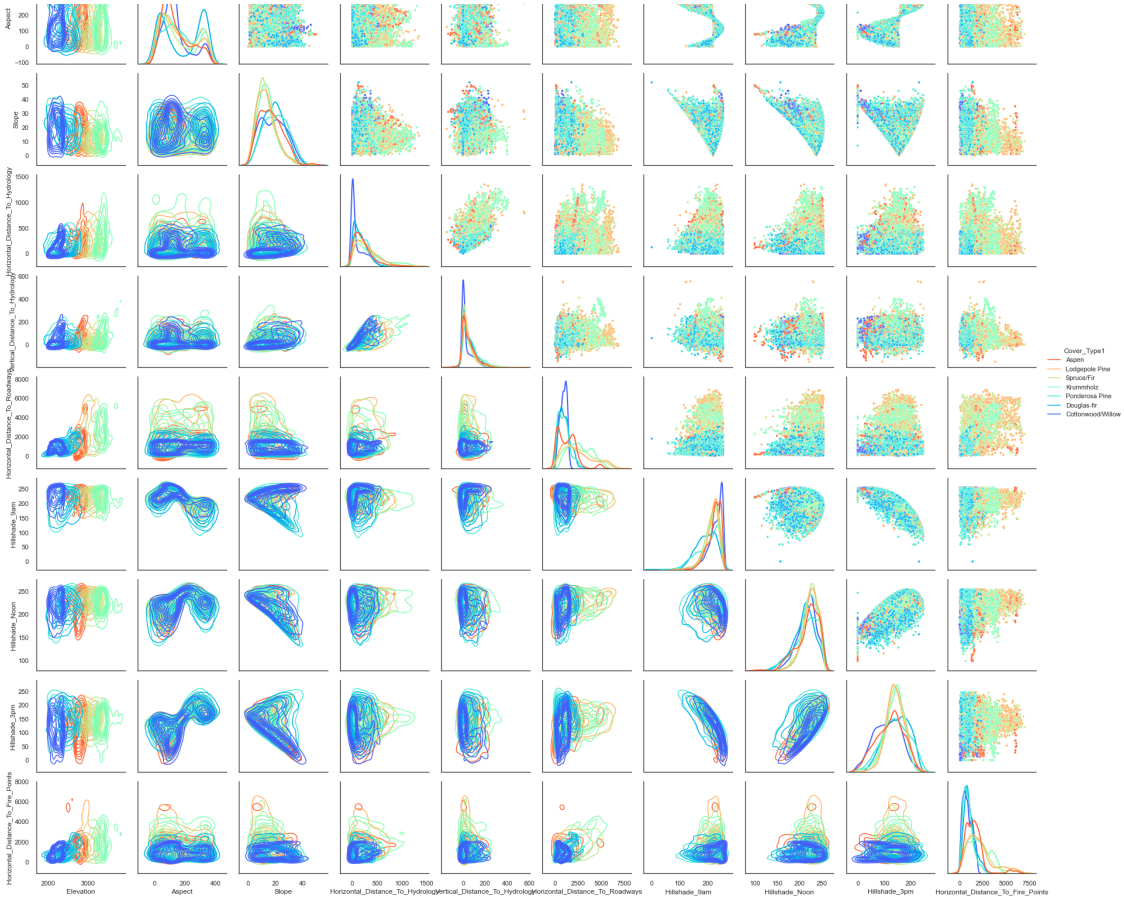
In [121…

```
sns.set_theme(style="white")

g = sns.PairGrid(data_train, vars = ['Elevation','Aspect','Slope','Horizo
g.map_upper(sns.scatterplot, s=15)
g.map_lower(sns.kdeplot,warn_singular=False)
g.map_diag(sns.kdeplot, lw=2,warn_singular=False)
g.add_legend()
```

Out[121]:

```
<seaborn.axisgrid.PairGrid at 0x7fc010a95a00>
```



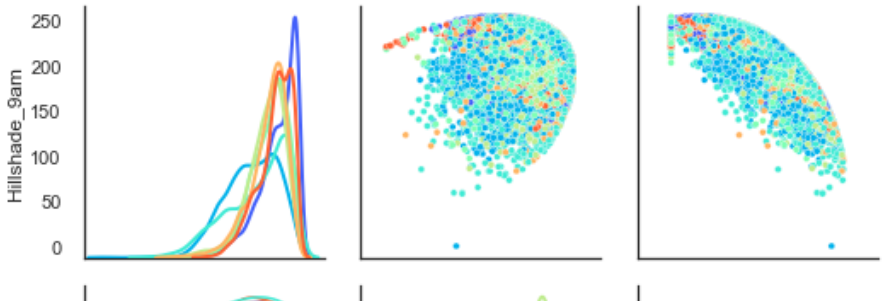


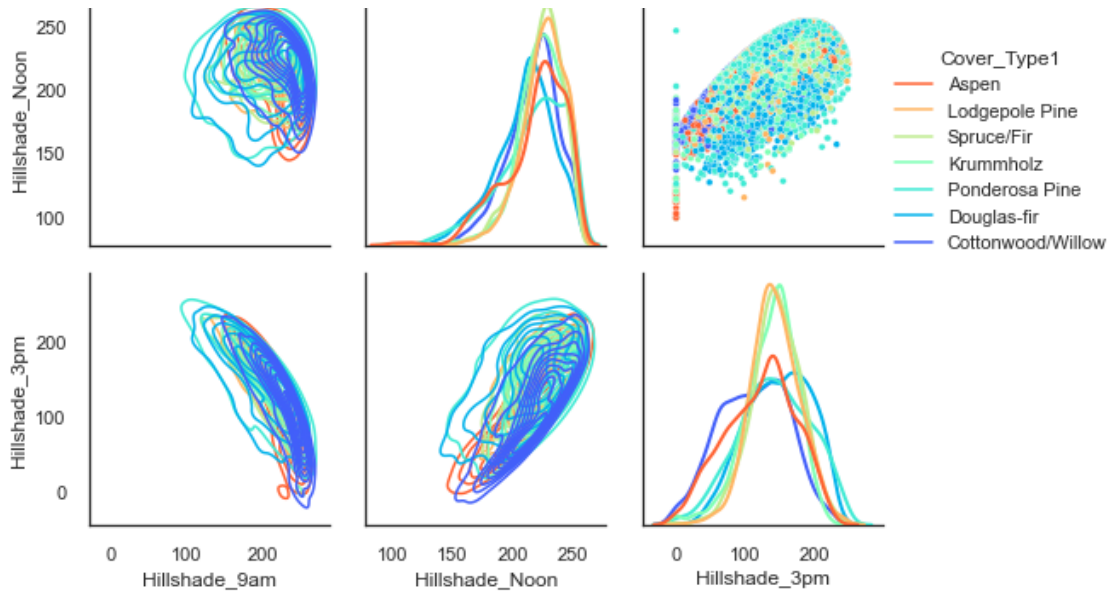
In [122…

```
sns.set_theme(style="white")

g = sns.PairGrid(data_train, vars = ['Hillshade_9am','Hillshade_Noon','Hi
g.map_upper(sns.scatterplot, s=15)
g.map_lower(sns.kdeplot,warn_singular=False)
g.map_diag(sns.kdeplot, lw=2,warn_singular=False)
g.add_legend()
```

Out[122]:

```
<seaborn.axisgrid.PairGrid at 0x7fc02641f370>
```



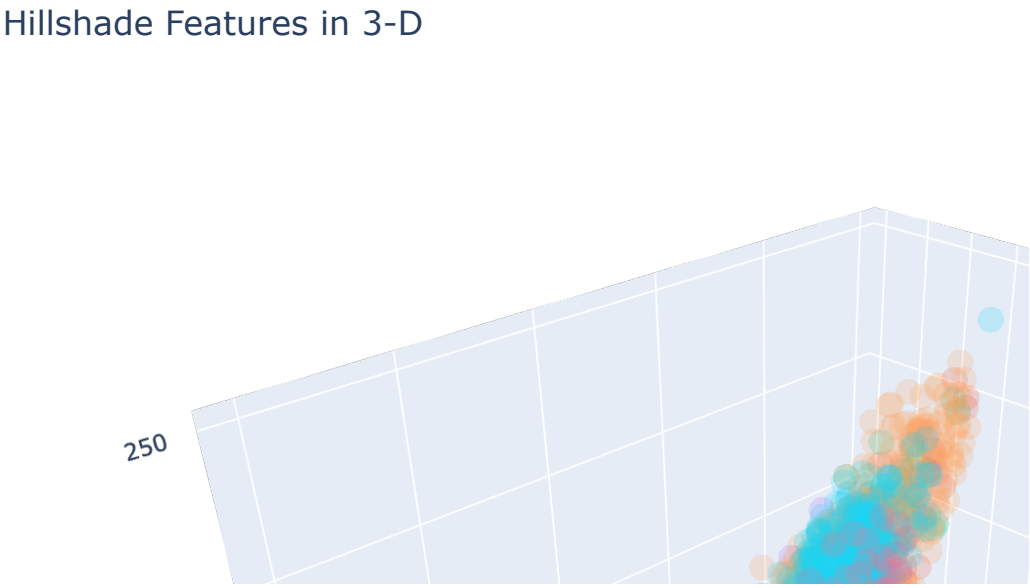


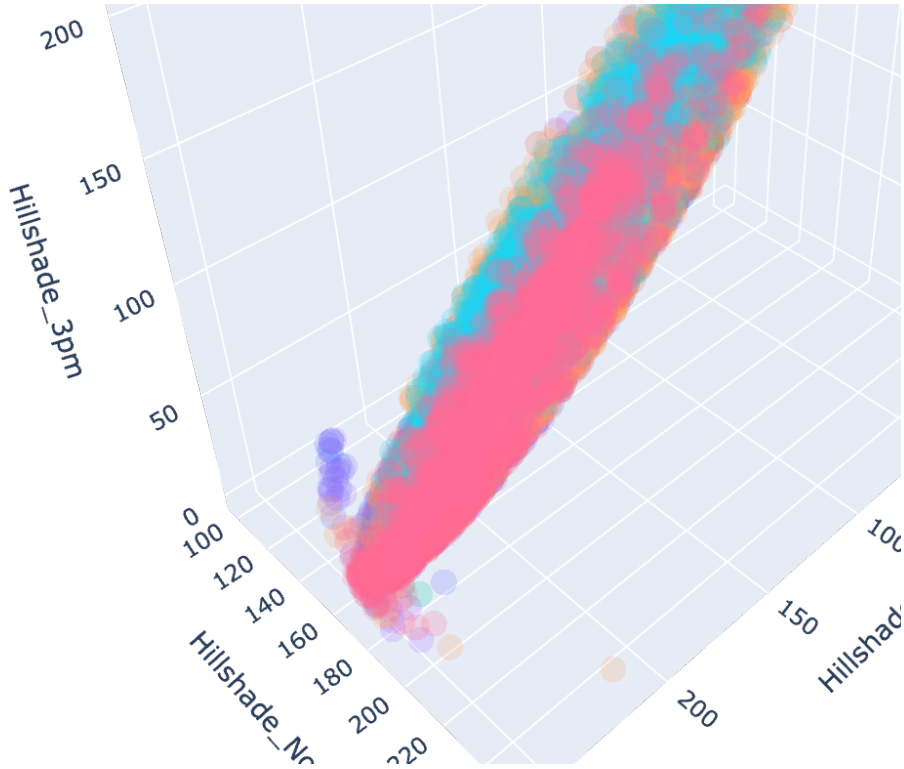
- Hillshade 3pm has some null values and looking at the graphical distribution it seems more these are missing values as these do not follow the pattern of the graph.
- But looking at the Hilshade noon and hilshade 9am it looks these are not nulls
- Since Hilshade 9am and Hilshade 3pm are negativly correlated, we can impute the values based on the correlation.

In [123…

```
label_names = ['Spruce/Fir', 'Lodgepole Pine', 'Ponderosa Pine', 'Cottonw
temp_df = data_train[['Hillshade_9am', 'Hillshade_Noon', 'Hillshade_3pm']
temp_df['Cover_Type'] = [label_names[i-1] for i in data_train['Cover_Type

fig = px.scatter_3d(
    temp_df,
    x='Hillshade_9am',
    y='Hillshade_Noon',
    z='Hillshade_3pm',
    color='Cover_Type',
    title='Hillshade Features in 3-D',
    opacity=0.2,
    color_discrete_sequence=px.colors.qualitative.Plotly[:7],
    width=1000, height=800
    )
```

```
    )

fig.show()
```

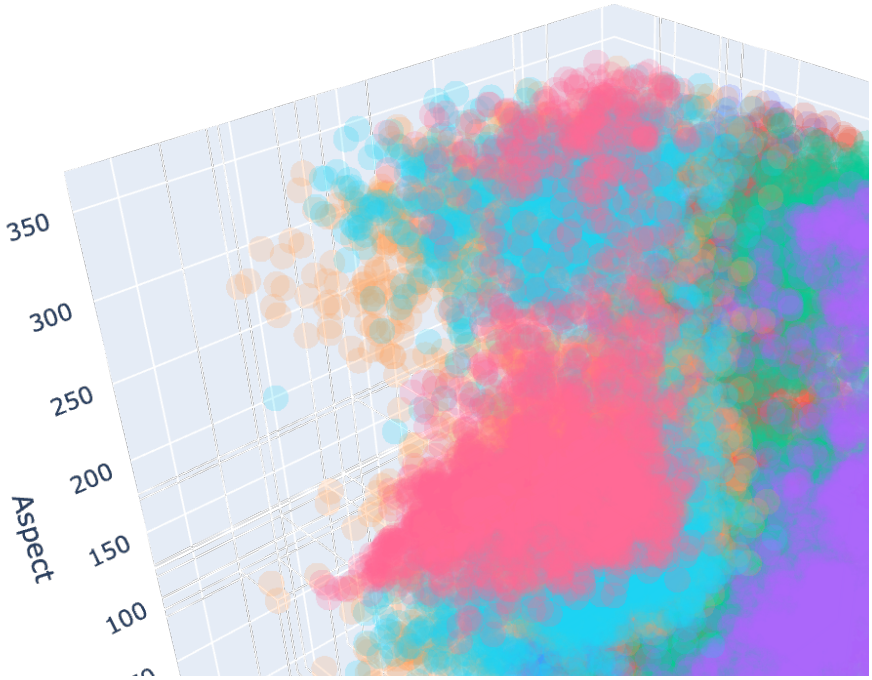


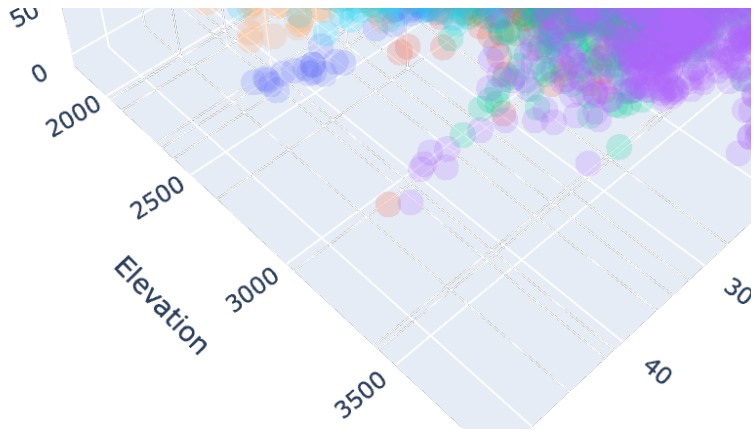


In [124…

```
label_names = ['Spruce/Fir', 'Lodgepole Pine', 'Ponderosa Pine', 'Cottonw
temp_df = data_train[['Slope', 'Elevation', 'Aspect']].copy()
temp_df['Cover_Type'] = [label_names[i-1] for i in data_train['Cover_Type

fig = px.scatter_3d(
    temp_df,
    x='Slope',
    y='Elevation',
    z='Aspect',
    color='Cover_Type',
    title='Hillshade Features in 3-D',
    opacity=0.2,
    color_discrete_sequence=px.colors.qualitative.Plotly[:7],
    width=1000, height=800
    )

fig.show()
```

Hillshade Features in 3-D

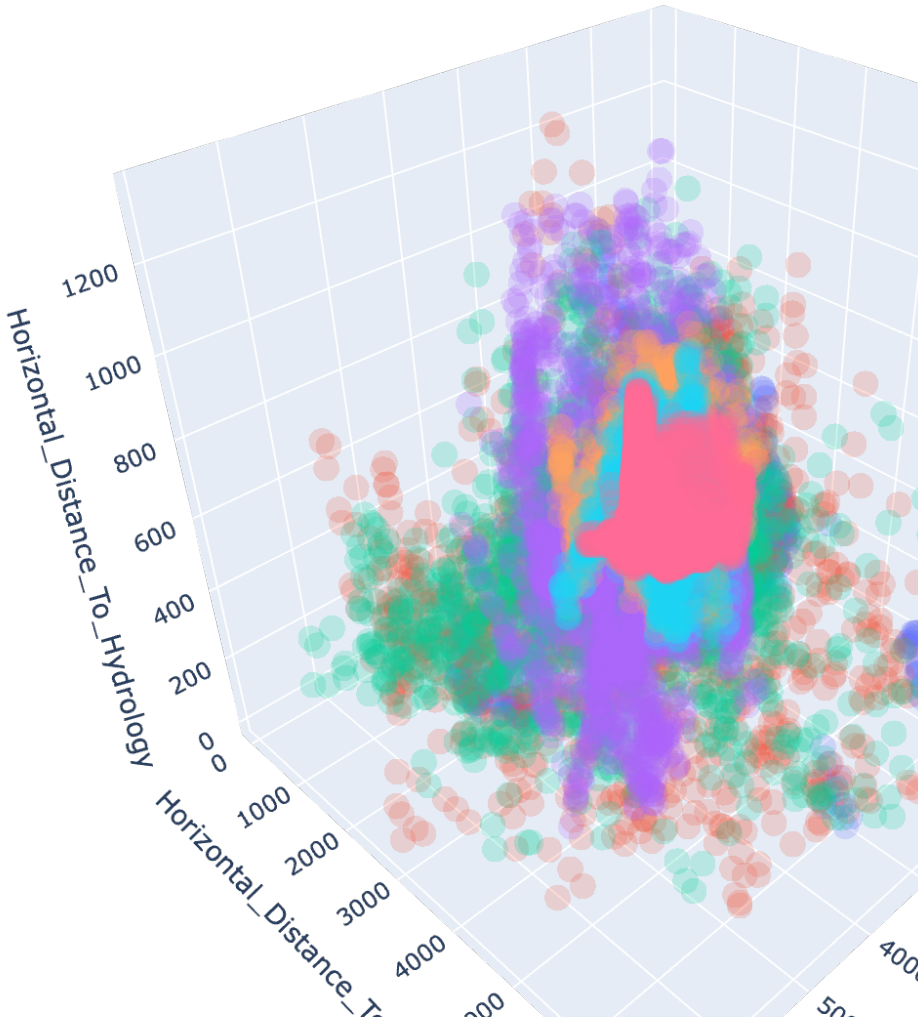




In [125…

```
label_names = ['Spruce/Fir', 'Lodgepole Pine', 'Ponderosa Pine', 'Cottonw
temp_df = data_train[['Horizontal_Distance_To_Roadways', 'Horizontal_Dist
temp_df['Cover_Type'] = [label_names[i-1] for i in data_train['Cover_Type

fig = px.scatter_3d(
    temp_df,
    x='Horizontal_Distance_To_Roadways',
    y='Horizontal_Distance_To_Fire_Points',
    z='Horizontal_Distance_To_Hydrology',
    color='Cover_Type',
    title='Hillshade Features in 3-D',
    opacity=0.2,
    color_discrete_sequence=px.colors.qualitative.Plotly[:7],
    width=1000, height=800
    )

fig.show()
```

Hillshade Features in 3-D



The 3D diagrams allow us to see common dimensions (like all three distances to hydrology and firepoints; the hillshades at different times; and the elevation, aspect and slope against the cover types

__3.6.Categorial EDA__

__3.6.1.Categorial Bar Diagrams__

- **Area1** has four dominant Cover types:

  - Lodgepole Pine
  - Spruce/Fir
  - Aspen
  - Krummholz.
- **Area2** has three Cover Types but are small in numbers:

  - Krummholz
  - Spruce/Fir
  - Lodgepole Pine.
- **Area3** has six Cover Types in decreasing order:

  - Krummholz
  - Aspen
  - Douglas-fir
  - Lodgepole Pine
  - Spruce/Fir
  - Ponderosa Pine
- **Area4** has four dominant Cover types in decreasing order:

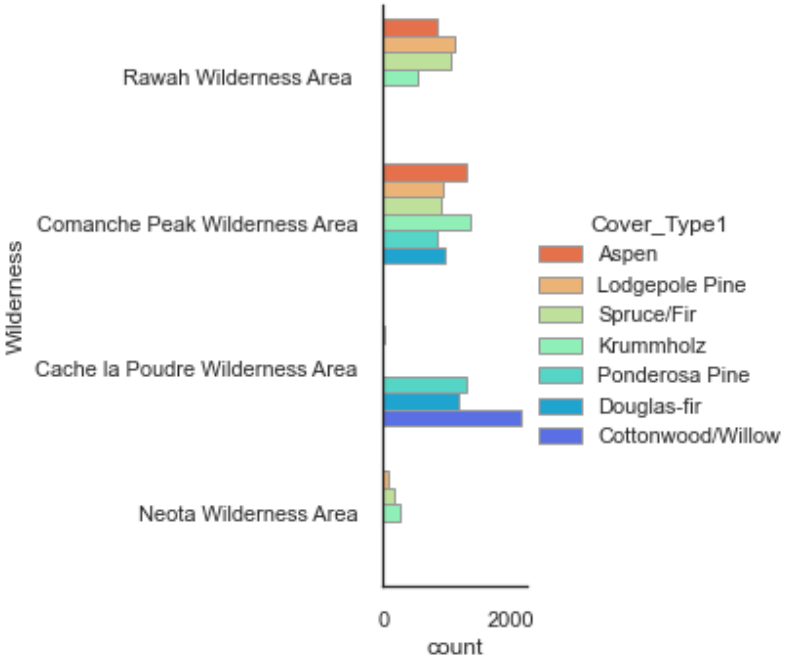
  - Cottonwood/Willow
  - Ponderosa Pine
  - Douglas-fir
  - Lodgepole Pine.

In [126…

```
sns.catplot(y="Wilderness", hue='Cover_Type1', kind="count",
            palette='rainbow_r', edgecolor=".6",
            data = data_train)
```

Out[126]:

```
<seaborn.axisgrid.FacetGrid at 0x7fc012dbaca0>
```


Rawah Wilderness Area
Comanche Peak Wilderness Area
Cache la Poudre Wilderness Area
Neota Wilderness Area
Wilderness
0
2000
count
Cover_Type1
Aspen
Lodgepole Pine
Spruce/Fir
Krummholz
Ponderosa Pine
Douglas-fir
Cottonwood/Willow

Highlights only: There are few Soil Types where there are few Cover Types or not existing cover Types, most of them are stony soil types like Soil Types 18,24,5,9,28,34,37,36,26,27 and 25. Few are from other soil types like 16,20,19,8 and 21.

Soil_Type3 (Haploborolis - Rock outcrop complex, rubbly) : Mainly Cover Type **Cottonwood**

Soil_Type4 (Ratake family - Rock outcrop complex, rubbly) :**Ponderosa Pine**

Soil_Type10 (Bullwark - Catamount families - Rock outcrop complex, rubbly): Mainly **Douglas-fir and Ponderosa Pine**

Soil_Type23 (Leighcan family, till substratum - Typic Cryaquolls complex): Mainly **Spruce/Fir**

Soil_Type29 (Como - Legault families complex, extremely stony) : **Lodgepole Pine, Spruce/Fir**

Soil_Type30 (Como family - Rock land - Legault family complex, extremely stony): **Aspen**

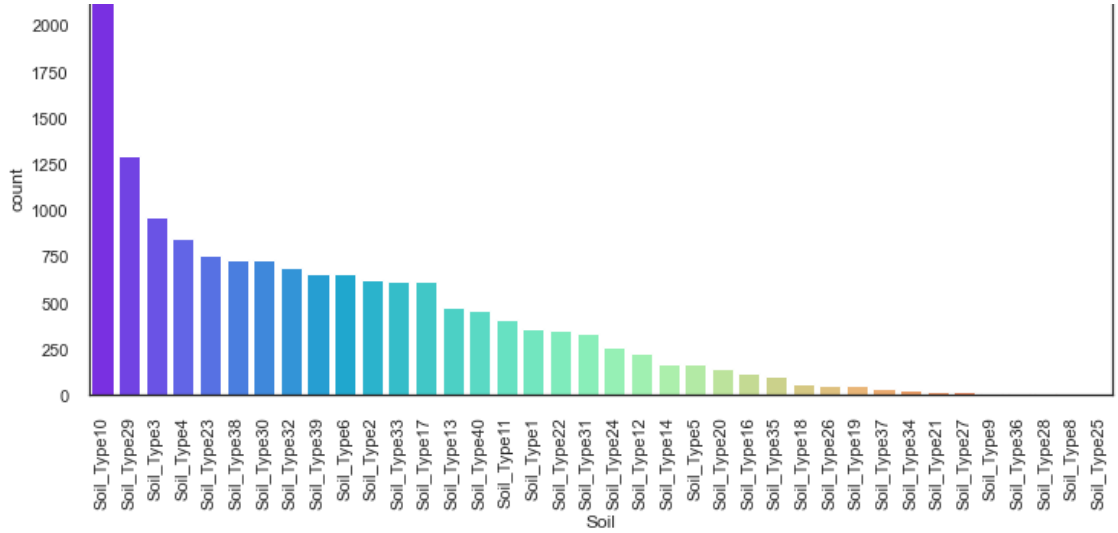
Soil_Type38 (Leighcan - Moran families - Cryaquolls complex, extremely stony): Mainly **Krummholz**

Soil_Type39 (Moran family - Cryorthents - Leighcan family complex, extremely stony): Mainly **Krummholz**

Soil_Type40 (Moran family - Cryorthents - Rock land complex, extremely stony): Mainly **Krummholz**

In [127…

```
soil_dummies = data_train.loc[:,data_train.columns.str.startswith('Soil_T
soil = soil_dummies.idxmax(axis=1)
soil.name = 'Soil'
data_train['Soil'] = soil
plt.figure(figsize=(12,5))
sns.countplot(x='Soil',data=data_train,palette='rainbow',order = data_tra
plt.xticks(rotation=90);
```

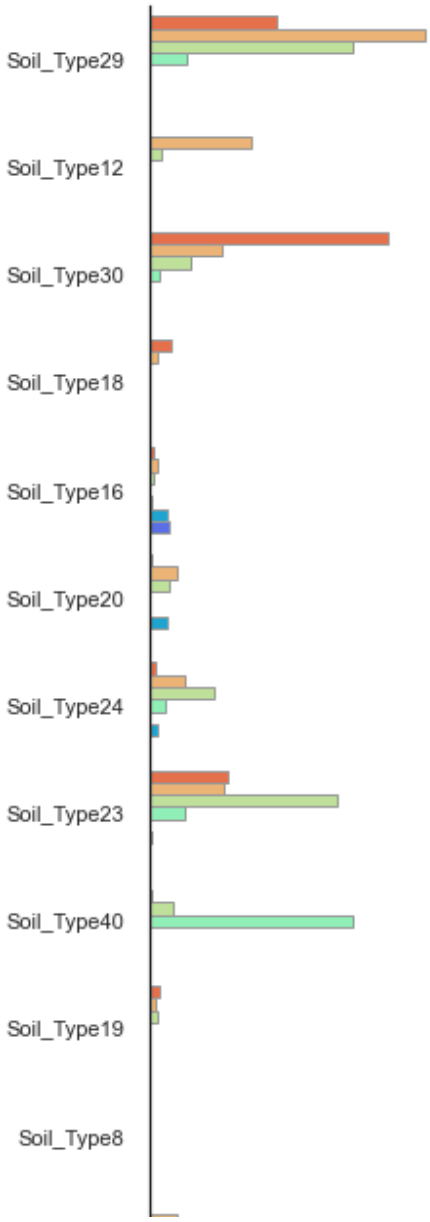


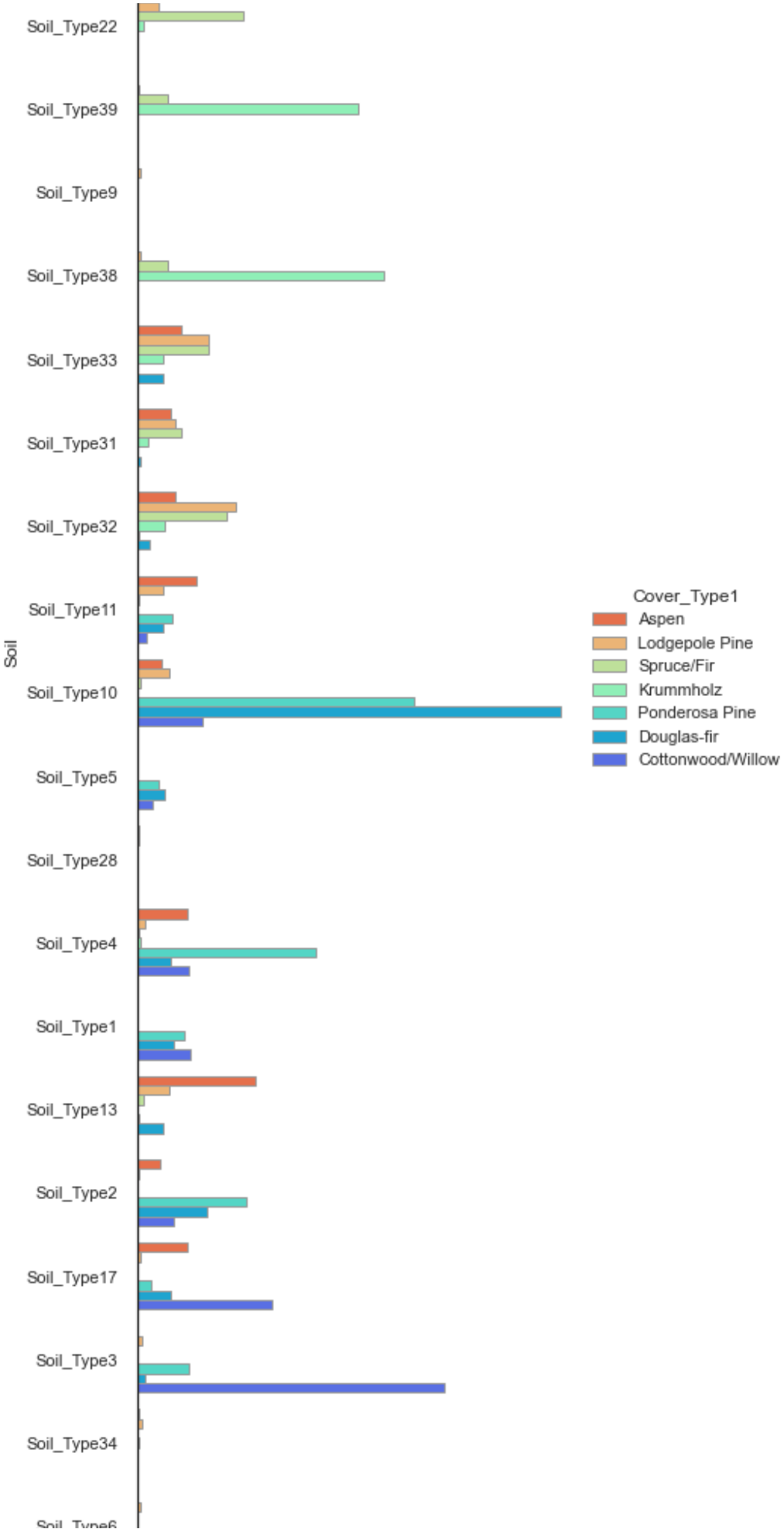
In [128…

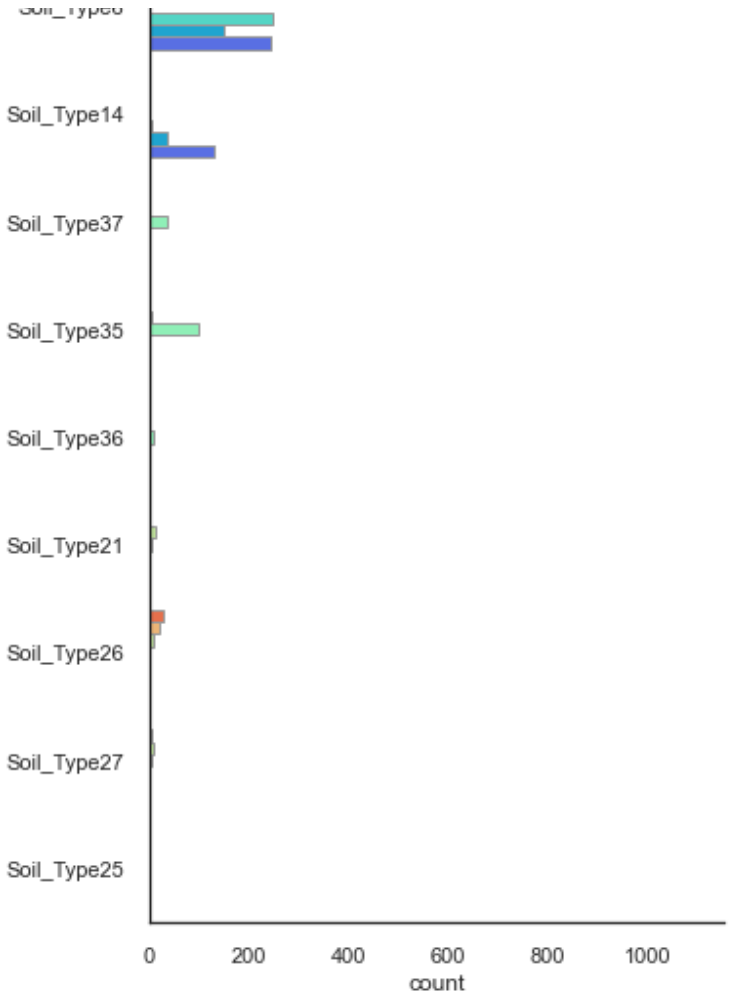
```
sns.catplot(y="Soil", hue='Cover_Type1', kind="count",
            palette='rainbow_r', edgecolor=".6",
            data=data_train,height=30, aspect=0.2)
```

Out[128]:

```
<seaborn.axisgrid.FacetGrid at 0x7fc0891232b0>
```




Soil_Type22
Soil_Type39
Soil_Type9
Soil_Type38
Soil_Type33
Soil_Type31
Soil_Type32
Soil_Type11
Soil_Type10
Soil_Type5
Soil_Type28
Soil_Type4
Soil_Type1
Soil_Type13
Soil_Type2
Soil_Type17
Soil_Type3
Soil_Type34
Soil_Type6
Soil
Cover_Type1
Aspen
Lodgepole Pine
Spruce/Fir
Krummholz
Ponderosa Pine
Douglas-fir
Cottonwood/Willow



__3.6.2.Violinplot with Dependent Variable__

Violinplot shows the distribution of the Cover_Types against the numerical variables:

1. **Spruce/Fir**:
   - Slope is lower

1. **Lodgepole Pine**:
   - Slope is lower

1. **Ponderosa Pine**:
   - Closer to the road than other cover types

1. **Cottonwood/Willow**:
   - Closer to the road than other cover types
   - Found on lower elevation

- Lowest horizontal distance to hydrology is lower and less spread than the other coverTypes
- Vertical distance to hydrology is more above the sea level than below, Hildshade 9am is high

1. **Aspen**:
   - Hildshade 9am is high

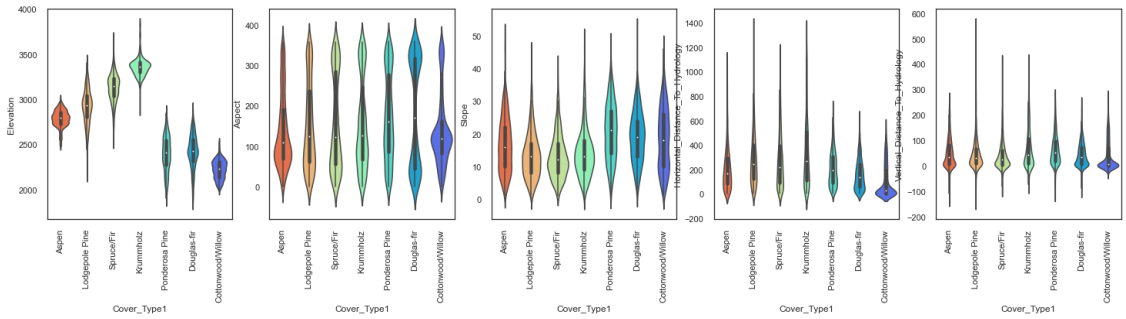
1. **Douglas-fir**:
   - Closer to the road than other cover types, Hildshade 9am is very spread

1. **Krummholz**:
   - Found on higher elevation, Slope is lower

- Elevation has the highest variance among the different Cover_Types
- Aspect on the other hand is fairly equally spread among the different Cover_Types

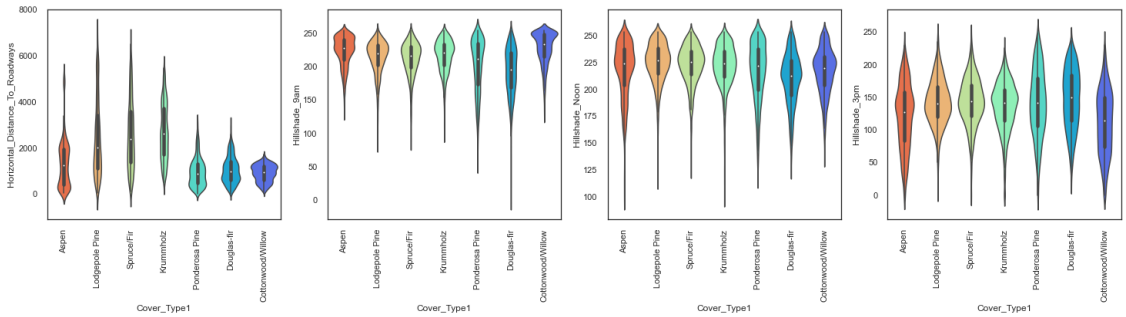
In [129…

```
vars= ['Elevation','Aspect','Slope','Horizontal_Distance_To_Hydrology','V
fig, axs = plt.subplots(1, len(vars), figsize=(25,5))
for i in range(0,len(vars)):
    sns.violinplot( y=data_train[vars[i]],x=data_train['Cover_Type1'], da
    for tick in axs[i].get_xticklabels():
        tick.set_rotation(90)
```



In [130…

```
vars= ['Horizontal_Distance_To_Roadways','Hillshade_9am','Hillshade_Noon'
fig, axs = plt.subplots(1, len(vars)-1, figsize=(25,5))
for i in range(0,len(vars)-1):
    sns.violinplot( y=data_train[vars[i]],x=data_train['Cover_Type1'], da
    for tick in axs[i].get_xticklabels():
        tick.set_rotation(90)
```



__3.6.3.Treemap for categorial Data__

In order to see the distribution of the treemap features we are checking which

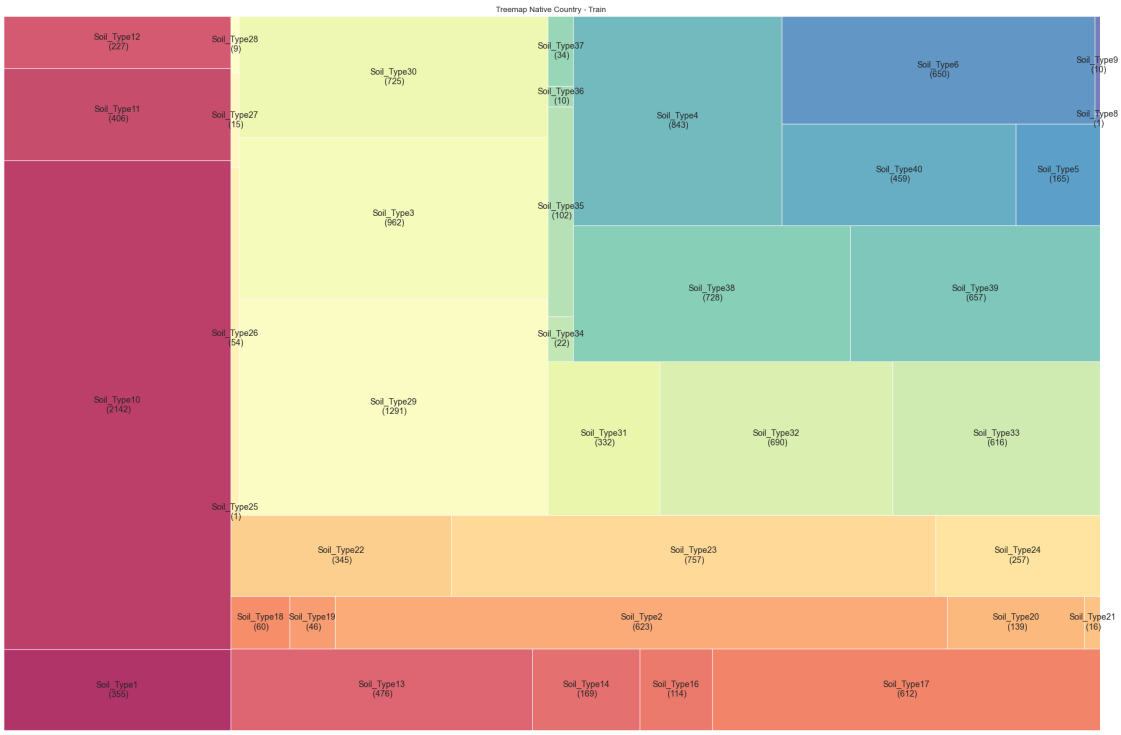
features are more represented than others in the train dataset.
You can see on the Soil Type that Soil Type 10 is represented the most, than Soil Type 29 and then Soil Type 3.

We can reach the conclusion that the topology of the terrain is composed for the most part of Bullwark - Catamount families - Rock outcrop complex, rubbly terrain despite any of the geological features like elevation, slope or azimuth

Treemap Graph by Soil Type on the Train Dataset

In [131…

```
# Prepare Data for Treemap graph

df = data_train.groupby('Soil').size().reset_index(name='counts')
labels = df.apply(lambda x: str(x[0]) + "\n (" + str(x[1]) + ")", axis=1)
sizes = df['counts'].values.tolist()
colors = [plt.cm.Spectral(i/float(len(labels))) for i in range(len(labels

# Draw Plot
plt.figure(figsize=(30,20), dpi= 80)
squarify.plot(sizes=sizes, label=labels, color=colors, alpha=.8,text_kwar

# Decorate
plt.title('Treemap Native Country - Train ')
plt.axis('off')
plt.show()
```

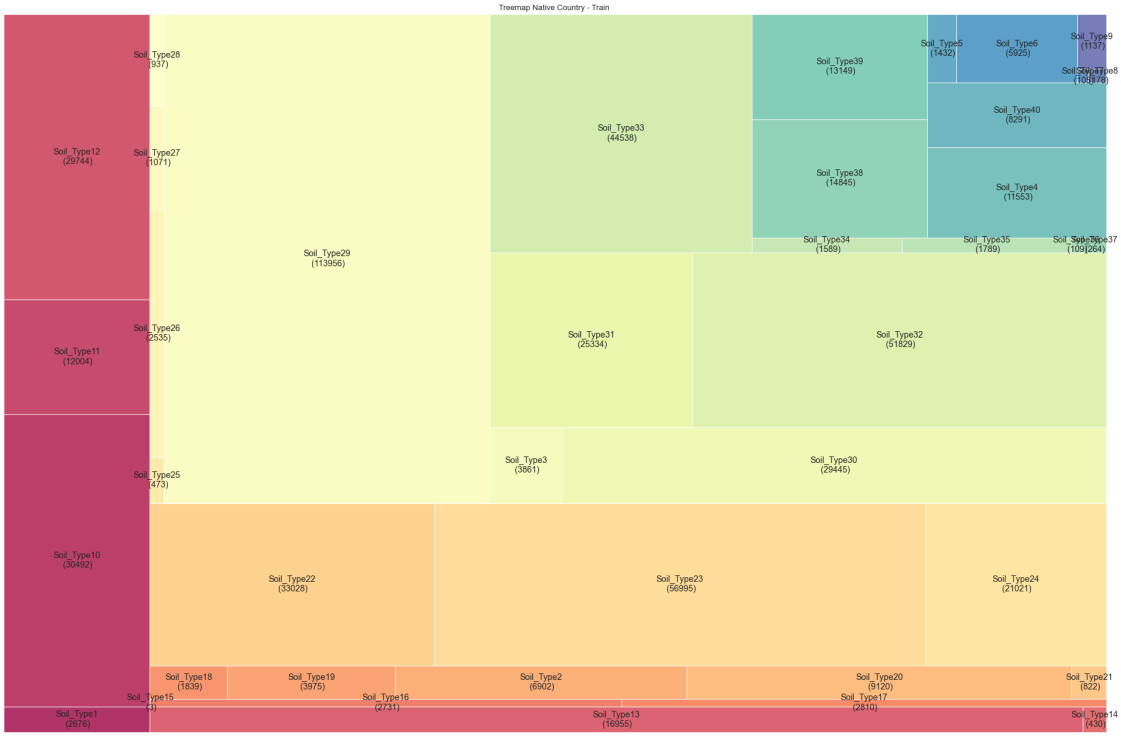


Treemap Graph by Soil Type on the Test Dataset

In [132…

```
df = data_test.groupby('Soil').size().reset_index(name='counts')
```

```
labels = df.apply(lambda x: str(x[0]) + "\n (" + str(x[1]) + ")", axis=1)
sizes = df['counts'].values.tolist()
colors = [plt.cm.Spectral(i/float(len(labels))) for i in range(len(labels

# Draw Plot
plt.figure(figsize=(30,20), dpi= 80)
squarify.plot(sizes=sizes, label=labels, color=colors, alpha=.8,text_kwar

# Decorate
plt.title('Treemap Native Country - Train ')
plt.axis('off')
plt.show()
```



In [133…

```
# Prepare Data
df = data_train.groupby('Wilderness').size().reset_index(name='counts')
labels = df.apply(lambda x: str(x[0]) + "\n (" + str(x[1]) + ")", axis=1)
sizes = df['counts'].values.tolist()
colors = [plt.cm.Spectral(i/float(len(labels))) for i in range(len(labels

# Draw Plot
plt.figure(figsize=(30,20), dpi= 80)
squarify.plot(sizes=sizes, label=labels, color=colors, alpha=.8,text_kwar
```

```
# Decorate
plt.title('Treemap Native Country - Train ')
plt.axis('off')
plt.show()
```

In the train dataset: Wilderness Area 3 (Comanche) is highly represented (42%), then Area 4 (Cahce la Poudre) (31%) , then Area 1 (Rawah) (24%) and last Area 2 (Neota) (3%).

We now check the test data for the same wilderness as a comparision in terms of similar distributions

In [134…

```
# Prepare Data
df = data_test.groupby('Wilderness').size().reset_index(name='counts')
labels = df.apply(lambda x: str(x[0]) + "\n (" + str(x[1]) + ")", axis=1)
sizes = df['counts'].values.tolist()
colors = [plt.cm.Spectral(i/float(len(labels))) for i in range(len(labels
```

```
# Draw Plot
plt.figure(figsize=(30,20), dpi= 80)
squarify.plot(sizes=sizes, label=labels, color=colors, alpha=.8,text_kwar

# Decorate
plt.title('Treemap Native Country - Test ')
plt.axis('off')
plt.show()
```

From train to test, the wilderness areas are not equally distributed. In the train dataset Comanche and Cache la Poudre are the biggest areas. Weheras in the test, its Comanche and Rawah.

However, as already seen in the Pandas Profiling Report Cover Type is equally distributed accross the train dataset, meaning, the size of each cover type is relatively the same

In [135…

```
# Prepare Data
df = data_train.groupby('Cover_Type1').size().reset_index(name='counts')
labels = df.apply(lambda x: str(x[0]) + "\n (" + str(x[1]) + ")", axis=1)
sizes = df['counts'].values.tolist()
colors = [plt.cm.Spectral(i/float(len(labels))) for i in range(len(labels

# Draw Plot
plt.figure(figsize=(30,20), dpi= 80)
squarify.plot(sizes=sizes, label=labels, color=colors, alpha=.8,text_kwar
```

```
# Decorate
plt.title('Treemap Native Country - Train ')
plt.axis('off')
plt.show()
```

In [136…

```
pio.renderers.default='notebook'
```

It is interesting to see which covertypes are present in which wilderness area graphically. As seen before, Neota and Rawah are almost not present due its small size in the train dataset

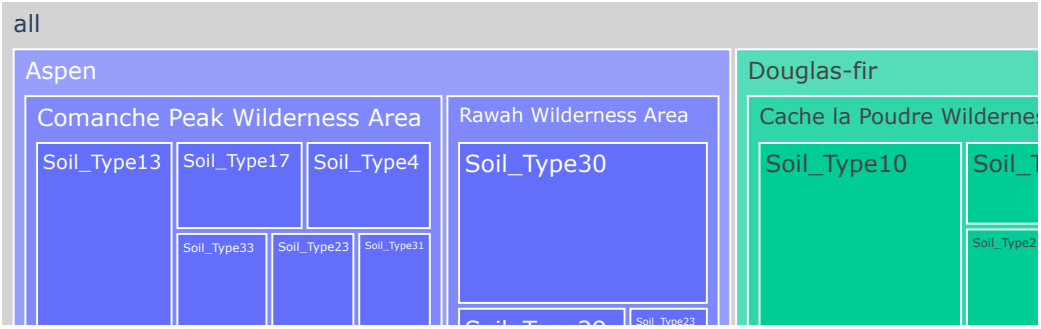
In [137…

```
fig = px.treemap(data_train, path=[px.Constant("all"), 'Cover_Type1', 'Wi
fig.update_traces(root_color="lightgrey")
fig.update_layout(margin = dict(t=50, l=25, r=25, b=25))
fig.show()
```

We can drill down even deeper and see which cover types are in which area, and at the same type understand which soil is present in the area

In [138…

```
fig = px.treemap(data_train, path=[px.Constant("all"), 'Cover_Type1', 'Wi
fig.update_traces(root_color="lightgrey")
fig.update_layout(margin = dict(t=50, l=25, r=25, b=25))
fig.show()
```



In [139…

```
matplotlib.rcParams['figure.figsize'] = (8, 5.5)
```

4.Baseline dataset

We will do firstly a basic benchmark without deploying any feature engineering.

The train dataset without any modifications (feature engineering) is used to use as a Baseline Model for future improvements.

In addition we use only basic Hyperparameters and will not attempt to fine tune these in the Baseline model either

4.1.Prepare Data and Standardization

We now assign y predictor (forest cover type) and drop irrelevant features for x.
These are the feature we create for EDA purposes (reverse dummy to categorical)

In [140…
```
X=data_train.drop(labels=['Id','Cover_Type','Wilderness','Cover_Type1','S
y=data_train['Cover_Type']
```

Machine Learning algorithms don't perform well when the input numerical attributes have very different scales. Since we have very skewed data, **standardization** is much less affected by outliers and it is the one we will choose **instead of normalization**.

The scale mean is zero and the standardization is 1.

It is also important we do not scale dummy variables, since these are 0 and 1, so we leave them out.

In [141…
```
scale_numerical =['Elevation','Aspect','Slope','Horizontal_Distance_To_Hy
            'Horizontal_Distance_To_Roadways','Hillshade_9am','Hillshade_
            'Horizontal_Distance_To_Fire_Points']
```

Now we do an additional split in our train dataset, resulting in train_train and train_val with an 80-20 split

We take for the Baseline model the exact same random state seed of 37.

Since the train set was not random and the cover types were equally spread, we stratify the split of the train into a train and validation set. Stratification will allow us to have a balanced dataset with respect to the target variable, so it does not happen the same we saw in the previous EDA

In [142…
```
X_train,X_val,y_train,y_val = train_test_split (X,y,random_state=37, test
```

In [143…
```
scaler = StandardScaler()
```

In [144…
```
X_train[scale_numerical] = scaler.fit_transform(X_train[scale_numerical])
X_val[scale_numerical] = scaler.transform(X_val[scale_numerical])
```

Testing the distribution of the cover types to ensure they are balanced among both train and validation set

In [145…
```
y_train.value_counts()
```

Out[145]:
```
5    1728
7    1728
3    1728
4    1728
1    1728
2    1728
6    1728
Name: Cover_Type, dtype: int64
```

In [146…
```
y_val.value_counts()
```

Out[146]:
```
6    432
4    432
1    432
5    432
2    432
3    432
7    432
Name: Cover_Type, dtype: int64
```

4.2.Random Forest

We start with the random forest classifier. We randomly tune the n_estimators hyperparameter to 20 and fit it in our train subsplit

In [147…
```
forest = RandomForestClassifier(random_state=37)

# fit the model with the training data
model_forest = forest.fit(X_train,y_train)

# predict the target on the train dataset
predict_train = model_forest.predict(X_train)
print('\nTarget on train data',predict_train)

# Accuracy Score on train dataset
accuracy_train = accuracy_score(y_train, predict_train)
print('\naccuracy_score on train dataset : ', accuracy_train)

# predict the target on the test dataset
predict_test = model_forest.predict(X_val)
print('\nTarget on test data',predict_test)

# Accuray Score on test dataset
accuracy_test = accuracy_score(y_val,predict_test)
print('\nAccuracy on test dataset : ', accuracy_test)
```

```
Target on train data [5 7 3 ... 3 5 1]

accuracy_score on train dataset :  1.0

Target on test data [6 4 4 ... 3 4 2]

Accuracy on test dataset :  0.8677248677248677
```

We get a very decent score of 86% without altering the original data_train_train subsplit even though it overfits on the train

In [148…
```
forest.score(X_val,y_val)
```

Out[148]:
```
0.8677248677248677
```

When we do the cross validation with the validation subset, accuracy decreases a bit to 78%

In [149…
```
forest = RandomForestClassifier(random_state=37)
```

```
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(model_forest, X
```

```
Accuracy = 0.7884
```

We are now interested in finding what would be the best hyperparameter for n_estimators and plug it back in the baseline

In [150…

```
from scipy.stats import uniform, randint
from sklearn.model_selection import RandomizedSearchCV

params = {
    "n_estimators": randint(0, 30), # default 100
}

forest_hyper = RandomizedSearchCV(forest, param_distributions=params, ran
forest_hyper.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 150 candidates, totalling 750 fits
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/model_selection/_validation.py:378: FitFailedWarning:


15 fits failed out of a total of 750.
The score on these train-test partitions for these parameters will be set
to nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------
-------
15 fits failed with the following error:
Traceback (most recent call last):
  File "/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages
/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages
/sklearn/ensemble/_forest.py", line 396, in fit
    self._validate_estimator()
  File "/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages
/sklearn/ensemble/_base.py", line 143, in _validate_estimator
    raise ValueError(
ValueError: n_estimators must be greater than zero, got 0.


/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/model_selection/_search.py:953: UserWarning:

One or more of the test scores are non-finite: [0.83258935 0.82382611 0.8
2655397 0.84193157 0.84242765 0.83730187
 0.83912097 0.84060847 0.76256533 0.76256533 0.82002303 0.83730187
 0.83920324 0.83730187 0.83845941 0.81092949 0.81092949 0.76256533
 0.83507002 0.82655397 0.83920324 0.82845572 0.82655397 0.84060847
 0.83589654 0.83920324 0.84060847 0.8383769  0.83920324 0.83779801
 0.83589654 0.82002303 0.76256533 0.69204712 0.8383769  0.84242765
 0.76256533 0.69204712 0.8398646  0.76256533 0.84060847 0.80861462
 0.82845572 0.83730187 0.82845572 0.83912097 0.70221432 0.82002303
 0.8383769  0.83779801 0.83589654 0.76256533 0.82382611 0.83258935
 0.77538054 0.83507002 0.77538054 0.84242765 0.8277114  0.83779801
 0.83920324 0.83779801 0.8277114  0.83912097 0.82382611 0.84242724
 0.83730187 0.83920324 0.8383769  0.79927297 0.8383769  0.84060847
 0.83589654 0.69204712 0.83258935 0.84242765 0.77538054 0.83258935
        nan 0.83845941 0.84060847 0.83258935 0.84242724 0.83920324
```

```
 0.82845572 0.83920324 0.70221432 0.80861462 0.83192847 0.80861462
 0.83845941 0.83920324 0.83779801 0.83507002 0.8398646  0.82002303
 0.77538054 0.83730187        nan 0.83845941 0.8277114  0.84242765
 0.84242724 0.82002303 0.80861462 0.77538054 0.83258935 0.84242724
 0.70221432 0.83845941 0.84242765 0.8398646  0.77538054 0.83258935
 0.69204712 0.77538054 0.81994048 0.83845941 0.83589654 0.8277114
 0.83589654 0.70221432 0.80861462 0.82655397        nan 0.83507002
 0.8398646  0.82382611 0.70221432 0.79927297 0.82382611 0.83912097
 0.8383769  0.82382611 0.83730187 0.84242724 0.83192847 0.82382611
 0.77538054 0.80861462 0.83845941 0.84242724 0.83258935 0.82655397
 0.83845941 0.81994048 0.83779801 0.81994048 0.83779801 0.84242724]

/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/model_selection/_search.py:953: UserWarning:

One or more of the train scores are non-finite: [0.99785052 0.99551506 0.
99648644 0.99952465 0.99958665 0.99915262
 0.99915262 0.99952465 0.95721725 0.95721725 0.99462631 0.99915262
 0.99927663 0.99915262 0.99911128 0.99117477 0.99117477 0.95721725
 0.99863592 0.99648644 0.99927663 0.99776785 0.99648644 0.99952465
 0.99840857 0.99927663 0.99952465 0.99894594 0.99927663 0.99900794
 0.99840857 0.99462631 0.95721725 0.88909557 0.99894594 0.99958665
 0.95721725 0.88909557 0.99933863 0.95721725 0.99952465 0.98871528
 0.99776785 0.99915262 0.99776785 0.99915262 0.889633   0.99462631
 0.99894594 0.99900794 0.99840857 0.95721725 0.99551506 0.99785052
 0.96591843 0.99863592 0.96591843 0.99958665 0.99685847 0.99900794
 0.99927663 0.99900794 0.99685847 0.99915262 0.99551506 0.99968998
 0.99915262 0.99927663 0.99894594 0.98412699 0.99894594 0.99952465
 0.99840857 0.88909557 0.99785052 0.99958665 0.96591843 0.99785052
        nan 0.99911128 0.99952465 0.99785052 0.99968998 0.99927663
 0.99776785 0.99927663 0.889633   0.98871528 0.99830522 0.98871528
 0.99911128 0.99927663 0.99900794 0.99863592 0.99933863 0.99462631
 0.96591843 0.99915262        nan 0.99911128 0.99685847 0.99958665
 0.99968998 0.99462631 0.98871528 0.96591843 0.99785052 0.99968998
 0.889633   0.99911128 0.99958665 0.99933863 0.96591843 0.99785052
 0.88909557 0.96591843 0.99293157 0.99911128 0.99840857 0.99685847
 0.99840857 0.889633   0.98871528 0.99648644        nan 0.99863592

 0.99933863 0.99551506 0.889633   0.98412699 0.99551506 0.99915262
 0.99894594 0.99551506 0.99915262 0.99968998 0.99830522 0.99551506
 0.96591843 0.98871528 0.99911128 0.99968998 0.99785052 0.99648644
 0.99911128 0.99293157 0.99900794 0.99293157 0.99900794 0.99968998]
```

Out[150]:

```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
  ▸ RandomForestClassifier
```

In [151…

```
print("Best parameters set found on development set:")
print()
print(forest_hyper.best_params_)
print()
print("Grid scores on development set:")
print()
means = forest_hyper.cv_results_['mean_test_score']
stds = forest_hyper.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, forest_hyper.cv_results_['param
    print("Accuracy = %0.3f (+/%0.03f) for %r" % (mean, std * 2, params))
```

```
Best parameters set found on development set:
```

```
{'n_estimators': 28}

Grid scores on development set:

Accuracy = 0.833 (+/0.007) for {'n_estimators': 15}
Accuracy = 0.824 (+/0.006) for {'n_estimators': 11}
Accuracy = 0.827 (+/0.006) for {'n_estimators': 12}
Accuracy = 0.842 (+/0.015) for {'n_estimators': 27}
Accuracy = 0.842 (+/0.012) for {'n_estimators': 28}
Accuracy = 0.837 (+/0.012) for {'n_estimators': 21}
Accuracy = 0.839 (+/0.016) for {'n_estimators': 22}
Accuracy = 0.841 (+/0.011) for {'n_estimators': 26}
Accuracy = 0.763 (+/0.011) for {'n_estimators': 3}
Accuracy = 0.763 (+/0.011) for {'n_estimators': 3}
Accuracy = 0.820 (+/0.006) for {'n_estimators': 10}
Accuracy = 0.837 (+/0.012) for {'n_estimators': 21}
Accuracy = 0.839 (+/0.012) for {'n_estimators': 24}
Accuracy = 0.837 (+/0.012) for {'n_estimators': 21}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 23}
Accuracy = 0.811 (+/0.010) for {'n_estimators': 8}
Accuracy = 0.811 (+/0.010) for {'n_estimators': 8}
Accuracy = 0.763 (+/0.011) for {'n_estimators': 3}
Accuracy = 0.835 (+/0.010) for {'n_estimators': 18}
Accuracy = 0.827 (+/0.006) for {'n_estimators': 12}
Accuracy = 0.839 (+/0.012) for {'n_estimators': 24}
Accuracy = 0.828 (+/0.011) for {'n_estimators': 14}
Accuracy = 0.827 (+/0.006) for {'n_estimators': 12}
Accuracy = 0.841 (+/0.011) for {'n_estimators': 26}
Accuracy = 0.836 (+/0.012) for {'n_estimators': 17}
Accuracy = 0.839 (+/0.012) for {'n_estimators': 24}
Accuracy = 0.841 (+/0.011) for {'n_estimators': 26}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 19}
Accuracy = 0.839 (+/0.012) for {'n_estimators': 24}
Accuracy = 0.838 (+/0.012) for {'n_estimators': 20}
Accuracy = 0.836 (+/0.012) for {'n_estimators': 17}
Accuracy = 0.820 (+/0.006) for {'n_estimators': 10}
Accuracy = 0.763 (+/0.011) for {'n_estimators': 3}
Accuracy = 0.692 (+/0.013) for {'n_estimators': 2}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 19}
Accuracy = 0.842 (+/0.012) for {'n_estimators': 28}
Accuracy = 0.763 (+/0.011) for {'n_estimators': 3}
Accuracy = 0.692 (+/0.013) for {'n_estimators': 2}
Accuracy = 0.840 (+/0.012) for {'n_estimators': 25}
Accuracy = 0.763 (+/0.011) for {'n_estimators': 3}
Accuracy = 0.841 (+/0.011) for {'n_estimators': 26}
Accuracy = 0.809 (+/0.006) for {'n_estimators': 7}
Accuracy = 0.828 (+/0.011) for {'n_estimators': 14}
Accuracy = 0.837 (+/0.012) for {'n_estimators': 21}
Accuracy = 0.828 (+/0.011) for {'n_estimators': 14}
Accuracy = 0.839 (+/0.016) for {'n_estimators': 22}
Accuracy = 0.702 (+/0.022) for {'n_estimators': 1}
Accuracy = 0.820 (+/0.006) for {'n_estimators': 10}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 19}
Accuracy = 0.838 (+/0.012) for {'n_estimators': 20}
Accuracy = 0.836 (+/0.012) for {'n_estimators': 17}
Accuracy = 0.763 (+/0.011) for {'n_estimators': 3}
Accuracy = 0.824 (+/0.006) for {'n_estimators': 11}
Accuracy = 0.833 (+/0.007) for {'n_estimators': 15}
Accuracy = 0.775 (+/0.008) for {'n_estimators': 4}
Accuracy = 0.835 (+/0.010) for {'n_estimators': 18}
Accuracy = 0.775 (+/0.008) for {'n_estimators': 4}
Accuracy = 0.842 (+/0.012) for {'n_estimators': 28}
Accuracy = 0.828 (+/0.010) for {'n_estimators': 13}
```

```
Accuracy = 0.828 (+/0.010) for {'n_estimators': 13}
Accuracy = 0.838 (+/0.012) for {'n_estimators': 20}
Accuracy = 0.839 (+/0.012) for {'n_estimators': 24}
Accuracy = 0.838 (+/0.012) for {'n_estimators': 20}
Accuracy = 0.828 (+/0.010) for {'n_estimators': 13}
Accuracy = 0.839 (+/0.016) for {'n_estimators': 22}
Accuracy = 0.824 (+/0.006) for {'n_estimators': 11}
Accuracy = 0.842 (+/0.014) for {'n_estimators': 29}
Accuracy = 0.837 (+/0.012) for {'n_estimators': 21}
Accuracy = 0.839 (+/0.012) for {'n_estimators': 24}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 19}
Accuracy = 0.799 (+/0.007) for {'n_estimators': 6}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 19}
Accuracy = 0.841 (+/0.011) for {'n_estimators': 26}
Accuracy = 0.836 (+/0.012) for {'n_estimators': 17}
Accuracy = 0.692 (+/0.013) for {'n_estimators': 2}
Accuracy = 0.833 (+/0.007) for {'n_estimators': 15}
Accuracy = 0.842 (+/0.012) for {'n_estimators': 28}
Accuracy = 0.775 (+/0.008) for {'n_estimators': 4}
Accuracy = 0.833 (+/0.007) for {'n_estimators': 15}
Accuracy = nan (+/nan) for {'n_estimators': 0}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 23}
Accuracy = 0.841 (+/0.011) for {'n_estimators': 26}
Accuracy = 0.833 (+/0.007) for {'n_estimators': 15}
Accuracy = 0.842 (+/0.014) for {'n_estimators': 29}
Accuracy = 0.839 (+/0.012) for {'n_estimators': 24}
Accuracy = 0.828 (+/0.011) for {'n_estimators': 14}
```

```
Accuracy = 0.839 (+/0.012) for {'n_estimators': 24}
Accuracy = 0.702 (+/0.022) for {'n_estimators': 1}
Accuracy = 0.809 (+/0.006) for {'n_estimators': 7}
Accuracy = 0.832 (+/0.011) for {'n_estimators': 16}
Accuracy = 0.809 (+/0.006) for {'n_estimators': 7}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 23}
Accuracy = 0.839 (+/0.012) for {'n_estimators': 24}
Accuracy = 0.838 (+/0.012) for {'n_estimators': 20}
Accuracy = 0.835 (+/0.010) for {'n_estimators': 18}
Accuracy = 0.840 (+/0.012) for {'n_estimators': 25}
Accuracy = 0.820 (+/0.006) for {'n_estimators': 10}
Accuracy = 0.775 (+/0.008) for {'n_estimators': 4}
Accuracy = 0.837 (+/0.012) for {'n_estimators': 21}
Accuracy = nan (+/nan) for {'n_estimators': 0}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 23}
Accuracy = 0.828 (+/0.010) for {'n_estimators': 13}
Accuracy = 0.842 (+/0.012) for {'n_estimators': 28}
Accuracy = 0.842 (+/0.014) for {'n_estimators': 29}
Accuracy = 0.820 (+/0.006) for {'n_estimators': 10}
Accuracy = 0.809 (+/0.006) for {'n_estimators': 7}
Accuracy = 0.775 (+/0.008) for {'n_estimators': 4}
Accuracy = 0.833 (+/0.007) for {'n_estimators': 15}
Accuracy = 0.842 (+/0.014) for {'n_estimators': 29}
Accuracy = 0.702 (+/0.022) for {'n_estimators': 1}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 23}
Accuracy = 0.842 (+/0.012) for {'n_estimators': 28}
Accuracy = 0.840 (+/0.012) for {'n_estimators': 25}
Accuracy = 0.775 (+/0.008) for {'n_estimators': 4}
Accuracy = 0.833 (+/0.007) for {'n_estimators': 15}
Accuracy = 0.692 (+/0.013) for {'n_estimators': 2}
Accuracy = 0.775 (+/0.008) for {'n_estimators': 4}
Accuracy = 0.820 (+/0.010) for {'n_estimators': 9}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 23}
Accuracy = 0.836 (+/0.012) for {'n_estimators': 17}
Accuracy = 0.828 (+/0.010) for {'n_estimators': 13}
Accuracy = 0.836 (+/0.012) for {'n_estimators': 17}
```

```
Accuracy = 0.702 (+/0.022) for {'n_estimators': 1}
Accuracy = 0.809 (+/0.006) for {'n_estimators': 7}
Accuracy = 0.827 (+/0.006) for {'n_estimators': 12}
Accuracy = nan (+/nan) for {'n_estimators': 0}
Accuracy = 0.835 (+/0.010) for {'n_estimators': 18}
Accuracy = 0.840 (+/0.012) for {'n_estimators': 25}
Accuracy = 0.824 (+/0.006) for {'n_estimators': 11}
Accuracy = 0.702 (+/0.022) for {'n_estimators': 1}
Accuracy = 0.799 (+/0.007) for {'n_estimators': 6}
Accuracy = 0.824 (+/0.006) for {'n_estimators': 11}
Accuracy = 0.839 (+/0.016) for {'n_estimators': 22}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 19}
Accuracy = 0.824 (+/0.006) for {'n_estimators': 11}
Accuracy = 0.837 (+/0.012) for {'n_estimators': 21}
Accuracy = 0.842 (+/0.014) for {'n_estimators': 29}
Accuracy = 0.832 (+/0.011) for {'n_estimators': 16}
Accuracy = 0.824 (+/0.006) for {'n_estimators': 11}
Accuracy = 0.775 (+/0.008) for {'n_estimators': 4}
Accuracy = 0.809 (+/0.006) for {'n_estimators': 7}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 23}
Accuracy = 0.842 (+/0.014) for {'n_estimators': 29}

Accuracy = 0.833 (+/0.007) for {'n_estimators': 15}
Accuracy = 0.827 (+/0.006) for {'n_estimators': 12}
Accuracy = 0.838 (+/0.011) for {'n_estimators': 23}
Accuracy = 0.820 (+/0.010) for {'n_estimators': 9}
Accuracy = 0.838 (+/0.012) for {'n_estimators': 20}
Accuracy = 0.820 (+/0.010) for {'n_estimators': 9}
Accuracy = 0.838 (+/0.012) for {'n_estimators': 20}
Accuracy = 0.842 (+/0.014) for {'n_estimators': 29}
```

The best n_estiamtors value is 29, let's plug it back in the original snippet to see if it surpasses 80.16% accuracy after cross-validation

In [152…

```
forest2 = RandomForestClassifier(n_estimators=29, random_state=37)

# fit the model with the training data
model_forest2 = forest.fit(X_train,y_train)

# predict the target on the train dataset
predict_train2 = model_forest.predict(X_train)
print('\nTarget on train data',predict_train2)

# Accuracy Score on train dataset
accuracy_train2 = accuracy_score(y_train, predict_train)
print('\naccuracy_score on train dataset : ', accuracy_train2)

# Predict the target on the test dataset
predict_test2 = model_forest2.predict(X_val)
print('\nTarget on test data',predict_test2)

# Accuracy Score on test dataset
accuracy_test2 = accuracy_score(y_val,predict_test)
print('\nAccuracy on test dataset : ', accuracy_test2)
```

```
Target on train data [5 7 3 ... 3 5 1]

accuracy_score on train dataset :  1.0

Target on test data [6 4 4 ... 3 4 2]

Accuracy on test dataset :  0.8677248677248677
```

Checking the cross validation

In [153…

```
forest2 = RandomForestClassifier(random_state=37)
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(model_forest2,
```

```
Accuracy = 0.8497
```

We get exactly the same accuracy when we were not looking for the best hyperparameters possible...

In [55]:

```
forest2 = RandomForestClassifier(random_state=37)
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(model_forest2,
```

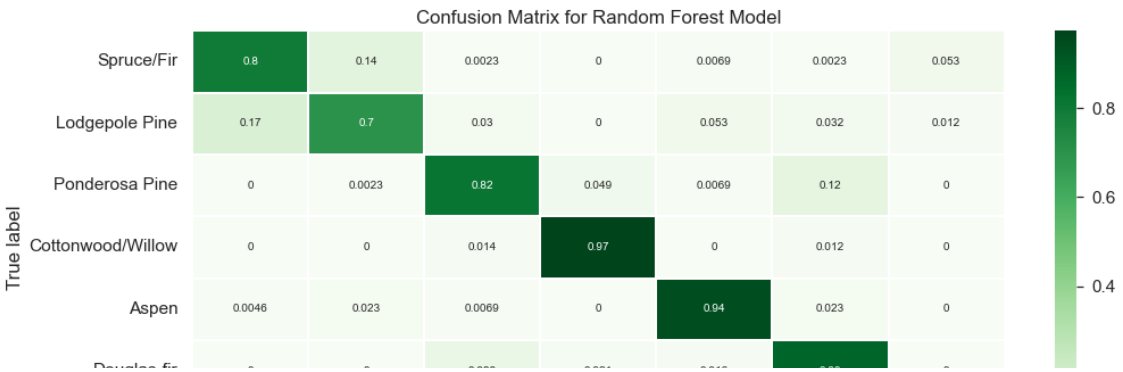
```
Accuracy = 0.7884
```

And the same accuracy when cross-validating.

In the confusion matrices below, the rows represent the true labels and the columns represent predicted labels. Values on the diagonal represent the % of times where the predicted label matches the true label. Values in all other cells represent instances where the classifier mislabeled an observation; the column tells us what has been predicted, and the row tells us the correct predicted labels, which you can see on the diagonal.

In [56]:

```
# Get and reshape confusion matrix data
matrix = confusion_matrix(y_val, predict_test2)
matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]

# Build the plot
plt.figure(figsize=(16,7))
sns.set(font_scale=1.4)
sns.heatmap(matrix, annot=True, annot_kws={'size':10},
            cmap=plt.cm.Greens, linewidths=0.2)

# Add labels to the plot
class_names = ['Spruce/Fir', 'Lodgepole Pine', 'Ponderosa Pine',
               'Cottonwood/Willow', 'Aspen', 'Douglas-fir',
               'Krummholz']
tick_marks = np.arange(len(class_names))
tick_marks2 = tick_marks + 0.5
plt.xticks(tick_marks, class_names, rotation=25)
plt.yticks(tick_marks2, class_names, rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for Random Forest Model')
plt.show()
```



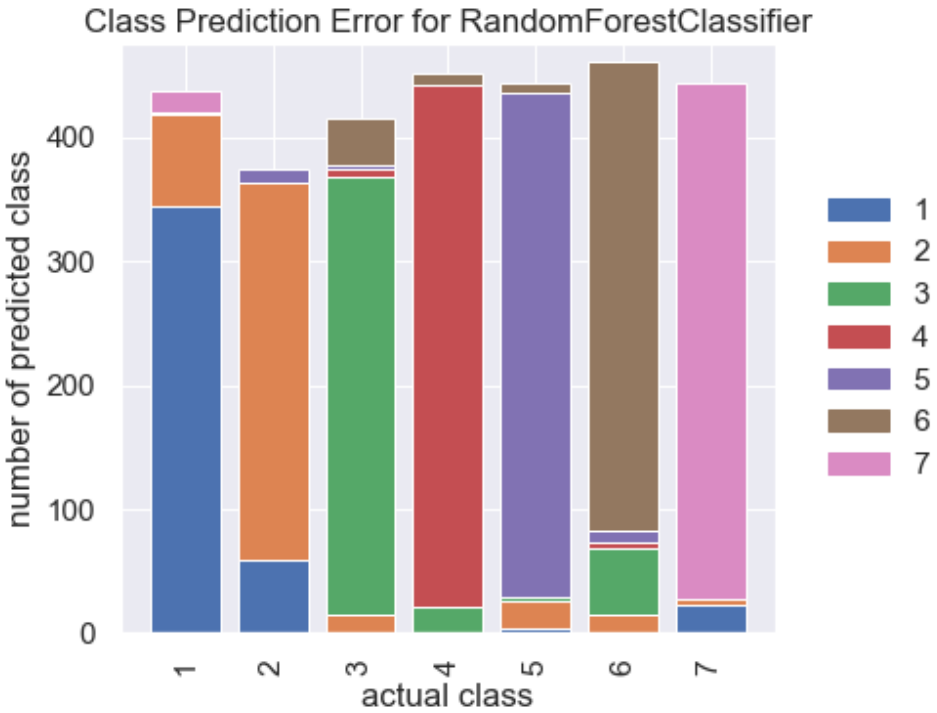
Precision:Correctly-identified number of classes divided by the model predicted classes. In case of Spur/Fir (1), the precision score is low. Meaning that the number of correctly -identified Spur/Fir divided by all correct or wrong predicted Spur/Fir (TP/(TP+FP))
Recall(Sensitivity,True Positive Rate): The ratio of correct positive predictions to the total positive examples (TP/(TP+FN)). For Spur/Fir the number of actual Spur/Fir that the prediction classified correctly as such.
F1-score: Measures of a model accuracy on a dataset. It combines precision and recall and can tell you if the model is good. The aim is to get close to 1. As you can see it is very low for Spur/Fir
Accuracy: Ratio of correctly predicted examples by the total examples. (TP+TN)/(TP+TN+FP+FN) which is 84%

In [57]:
```
print(classification_report(y_val, predict_test2))
```

```
              precision    recall  f1-score   support

           1       0.79      0.80      0.79       432
           2       0.81      0.70      0.75       432
           3       0.85      0.82      0.84       432
           4       0.93      0.97      0.95       432
           5       0.92      0.94      0.93       432
           6       0.82      0.88      0.85       432
           7       0.94      0.96      0.95       432

    accuracy                           0.87      3024
   macro avg       0.87      0.87      0.87      3024
weighted avg       0.87      0.87      0.87      3024
```

In [58]:
```
visualizer = ClassPredictionError(
    model_forest, palette = 'rainbow_r')
visualizer.fit(X_train, y_train)
visualizer.score(X_val, y_val)
visualizer.show();
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/base.py:450: UserWarning:

X does not have valid feature names, but RandomForestClassifier was fitte
d with feature names
```

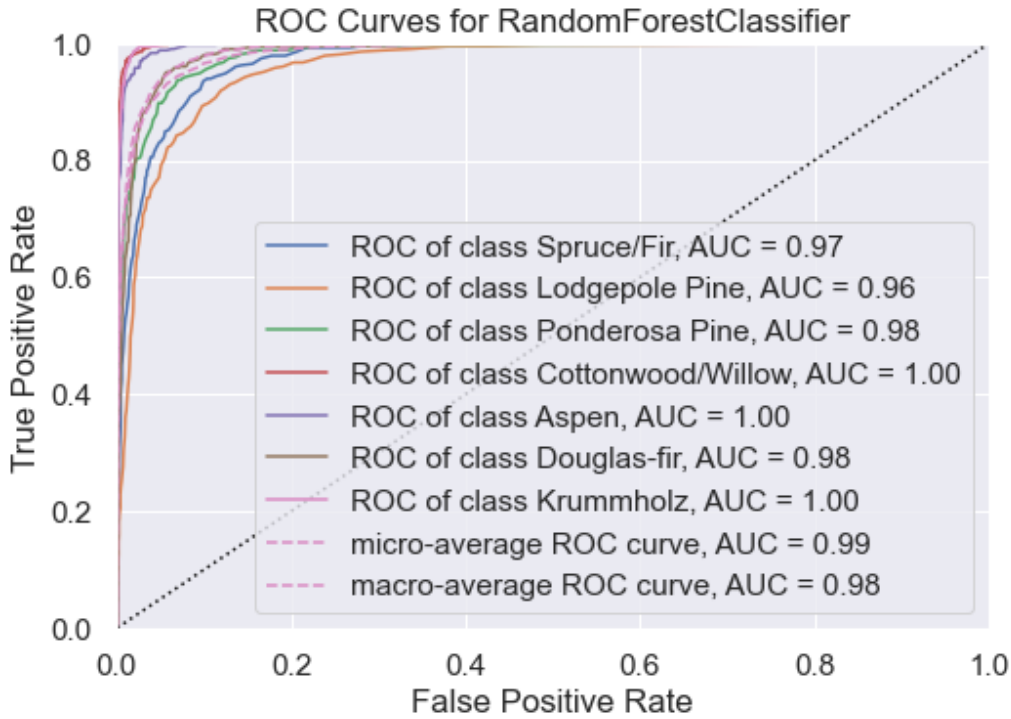


In [59]:
```
def plot_ROC_curve(model, X_train,y_train, X_val, y_val):

    # Creating visualization with the readable labels
    visualizer = ROCAUC(model, encoder={1: 'Spruce/Fir',
                                        2: 'Lodgepole Pine',
                                        3: 'Ponderosa Pine',
                                       4: 'Cottonwood/Willow',
                                       5: 'Aspen',
                                       6: 'Douglas-fir',
                                       7: 'Krummholz'})


    # Fitting to the training data first then scoring with the test data
    visualizer.fit(X_train,y_train)
    visualizer.score(X_val, y_val)
    visualizer.show()

    return visualizer
```

https://medium.com/swlh/how-to-create-an-auc-roc-plot-for-a-multiclass-model-9e13838dd3de

In [60]:
```
plot_ROC_curve(model_forest2, X_train,y_train, X_val, y_val)
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/base.py:450: UserWarning:

X does not have valid feature names, but RandomForestClassifier was fitte
d with feature names
```



Out[60]:

```
▸ ROCAUC
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

https://www.statology.org/plot-multiple-roc-curves-python/
https://afioto.medium.com/classification-visualizations-with-yellowbrick-d6f6150d7a32

Deploying model in test dataset (data_test)

The categorial variables of test that we added earlier for EDA needs to be removed as well

4.3.Gradient Boosting

We go through gradient boosting by preselecting a list of learning rates

In [61]:

```
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]

for learning_rate in lr_list:
    gb_clf = GradientBoostingClassifier(n_estimators=20, learning_rate=le
    gb_clf.fit(X_train, y_train)
```

```
    print("Learning rate: ", learning_rate)
    print("Accuracy score (training): {0:.3f}".format(gb_clf.score(X_trai
    print("Accuracy score (validation): {0:.3f}".format(gb_clf.score(X_va
```

```
Learning rate:  0.05
Accuracy score (training): 0.615
Accuracy score (validation): 0.599
Learning rate:  0.075
Accuracy score (training): 0.619
Accuracy score (validation): 0.611
Learning rate:  0.1
Accuracy score (training): 0.621
Accuracy score (validation): 0.612
Learning rate:  0.25
Accuracy score (training): 0.647
Accuracy score (validation): 0.649
Learning rate:  0.5
Accuracy score (training): 0.673
Accuracy score (validation): 0.664
Learning rate:  0.75
Accuracy score (training): 0.685
Accuracy score (validation): 0.677
Learning rate:  1
Accuracy score (training): 0.687
Accuracy score (validation): 0.659
```

The best learning rate for this algorithm is 0.67, we plug this back in below...

In [62]:
```
gb_clf = GradientBoostingClassifier(n_estimators=20, learning_rate=1, max
model_gb = gb_clf.fit(X_train, y_train)
y_pred_test_gb = gb_clf.predict(X_val)

print("Classification Report")
print(classification_report(y_val, y_pred_test_gb))
```

```
Classification Report
              precision    recall  f1-score   support

           1       0.60      0.65      0.62       432
           2       0.52      0.48      0.50       432
           3       0.55      0.57      0.56       432
           4       0.84      0.88      0.86       432
           5       0.69      0.69      0.69       432
           6       0.58      0.56      0.57       432
           7       0.84      0.78      0.81       432

    accuracy                           0.66      3024
   macro avg       0.66      0.66      0.66      3024
weighted avg       0.66      0.66      0.66      3024
```

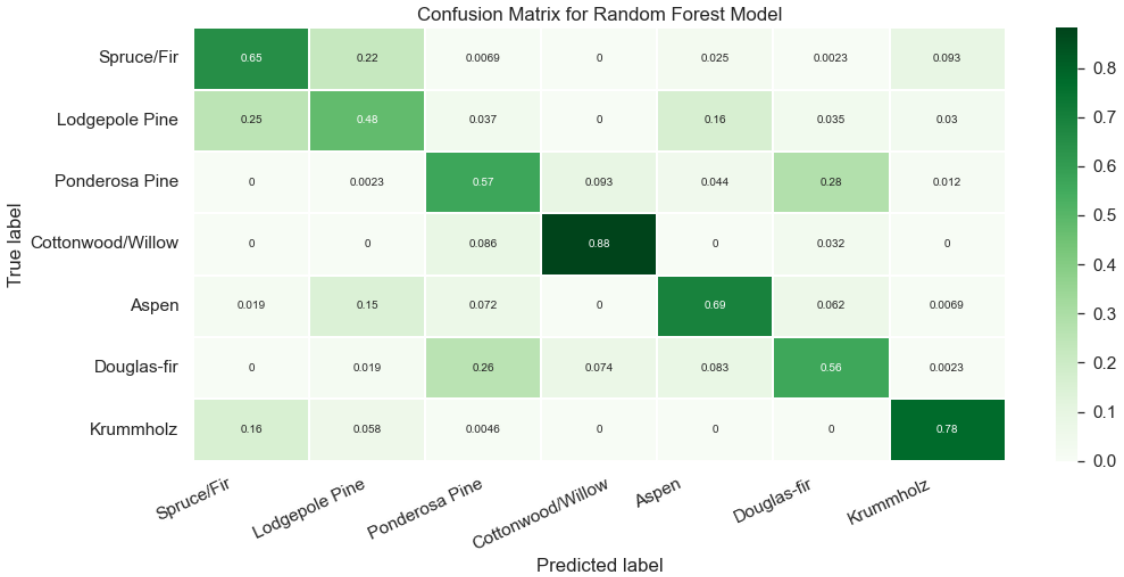
Accuracy is lower compared to Random Forest CV score of 75%

In [63]:
```
model_gb = GradientBoostingClassifier (random_state=37)
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(model_gb, X_val
```

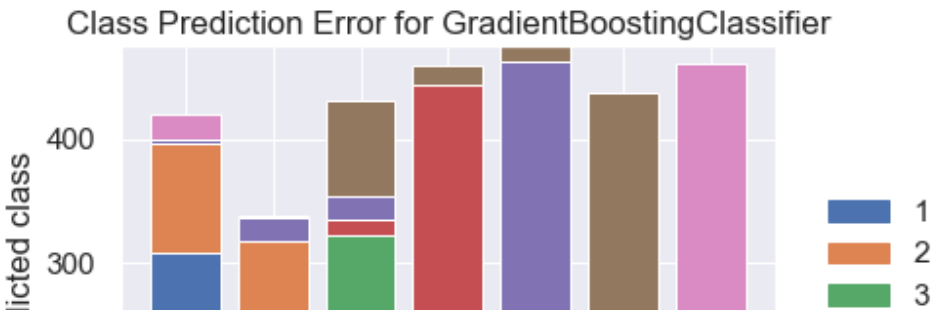
```
Accuracy = 0.7596
```

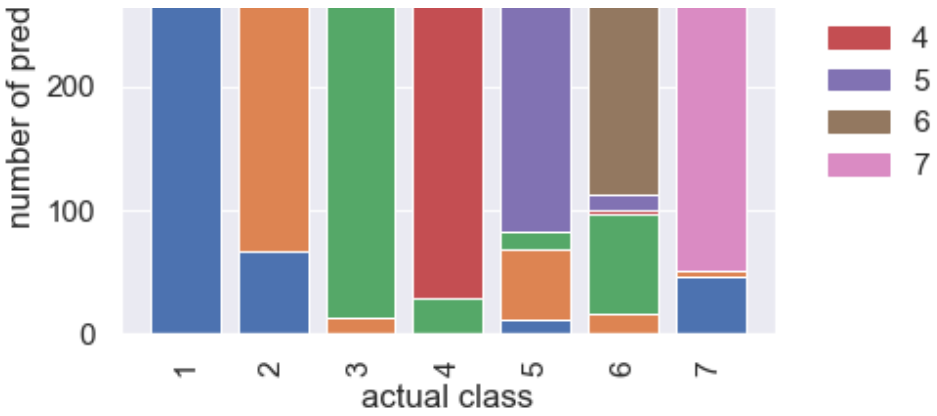
In comparison with the Random Forest, Gradient Boost actually increases accuracy when doing the cross validation with the train_val subsplit

In [64]:
```
# Get and reshape confusion matrix data
matrix = confusion_matrix(y_val, y_pred_test_gb)
matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]

# Build the plot
plt.figure(figsize=(16,7))
sns.set(font_scale=1.4)
sns.heatmap(matrix, annot=True, annot_kws={'size':10},
            cmap=plt.cm.Greens, linewidths=0.2)

# Add labels to the plot
class_names = ['Spruce/Fir', 'Lodgepole Pine', 'Ponderosa Pine',
               'Cottonwood/Willow', 'Aspen', 'Douglas-fir',
               'Krummholz']
tick_marks = np.arange(len(class_names))
tick_marks2 = tick_marks + 0.5
plt.xticks(tick_marks, class_names, rotation=25)
plt.yticks(tick_marks2, class_names, rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for Random Forest Model')
plt.show()
```



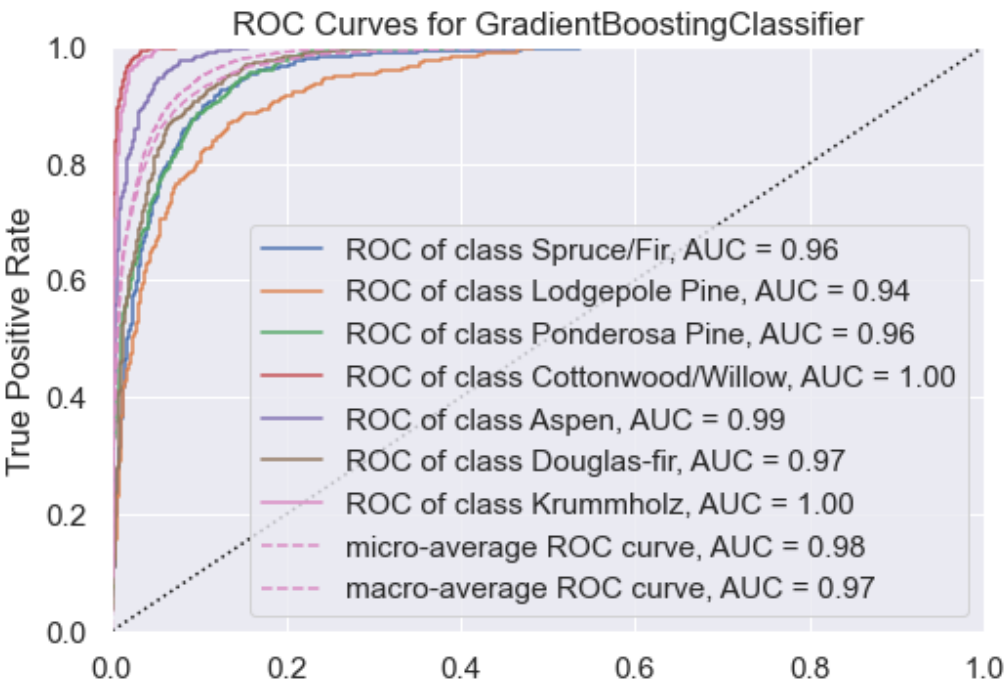
In [65]:
```
visualizer = ClassPredictionError(
    model_gb, palette = 'rainbow_r')
visualizer.fit(X_train, y_train)
visualizer.score(X_val, y_val)
visualizer.show();
```





In [66]:
```
plot_ROC_curve(model_gb, X_train,y_train, X_val, y_val)
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/base.py:450: UserWarning:

X does not have valid feature names, but GradientBoostingClassifier was f
itted with feature names
```



Out[66]:

```
▸ ROCAUC
▸ estimator: GradientBoostingClassifier
    ▸ GradientBoostingClassifier
```

4.4.Decision Trees

In [67]:

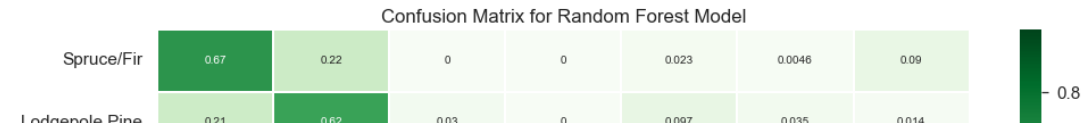
```
tree = DecisionTreeClassifier(random_state=18)
model_tree = tree.fit(X_train, y_train)
```

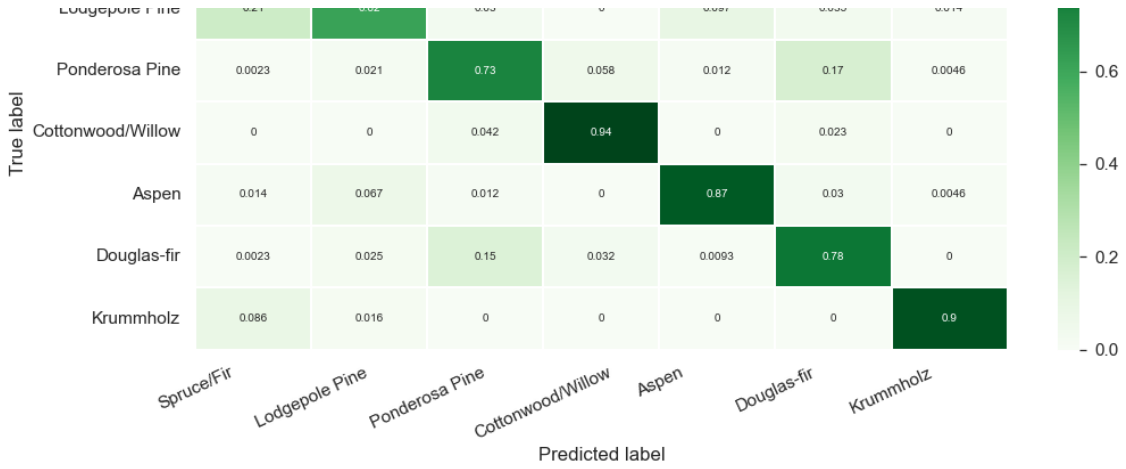
In [68]:

```
tree_predict_train = tree.predict(X_train)
tree_predict_val = tree.predict(X_val)
```

In [69]:

```
# Get and reshape confusion matrix data
matrix = confusion_matrix(y_val, tree_predict_val)
matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]

# Build the plot
plt.figure(figsize=(16,7))
sns.set(font_scale=1.4)
sns.heatmap(matrix, annot=True, annot_kws={'size':10},
            cmap=plt.cm.Greens, linewidths=0.2)

# Add labels to the plot
class_names = ['Spruce/Fir', 'Lodgepole Pine', 'Ponderosa Pine',
               'Cottonwood/Willow', 'Aspen', 'Douglas-fir',
               'Krummholz']
tick_marks = np.arange(len(class_names))
tick_marks2 = tick_marks + 0.5
plt.xticks(tick_marks, class_names, rotation=25)
plt.yticks(tick_marks2, class_names, rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for Random Forest Model')
plt.show()
```





In [70]:
```
print("Training Results:\n")
print(classification_report(y_train, tree_predict_train))
print("\nTesting Results:\n")
print(classification_report(y_val, tree_predict_val))
```

```
Training Results:

              precision    recall  f1-score   support

           1       1.00      1.00      1.00      1728
           2       1.00      1.00      1.00      1728
           3       1.00      1.00      1.00      1728
           4       1.00      1.00      1.00      1728
           5       1.00      1.00      1.00      1728
           6       1.00      1.00      1.00      1728
           7       1.00      1.00      1.00      1728

    accuracy                           1.00     12096
   macro avg       1.00      1.00      1.00     12096
weighted avg       1.00      1.00      1.00     12096


Testing Results:

              precision    recall  f1-score   support

           1       0.68      0.67      0.67       432
           2       0.64      0.62      0.63       432
           3       0.76      0.73      0.74       432
           4       0.91      0.94      0.92       432
           5       0.86      0.87      0.87       432
           6       0.75      0.78      0.76       432
           7       0.89      0.90      0.89       432

    accuracy                           0.79      3024
   macro avg       0.78      0.79      0.78      3024
weighted avg       0.78      0.79      0.78      3024
```

Similar to the Random Forest, we get a prediction of 1 in the train but does not generalize well in the train.

Compared to the Random Forest, it is 2% less accurate, let's perform some cross validation

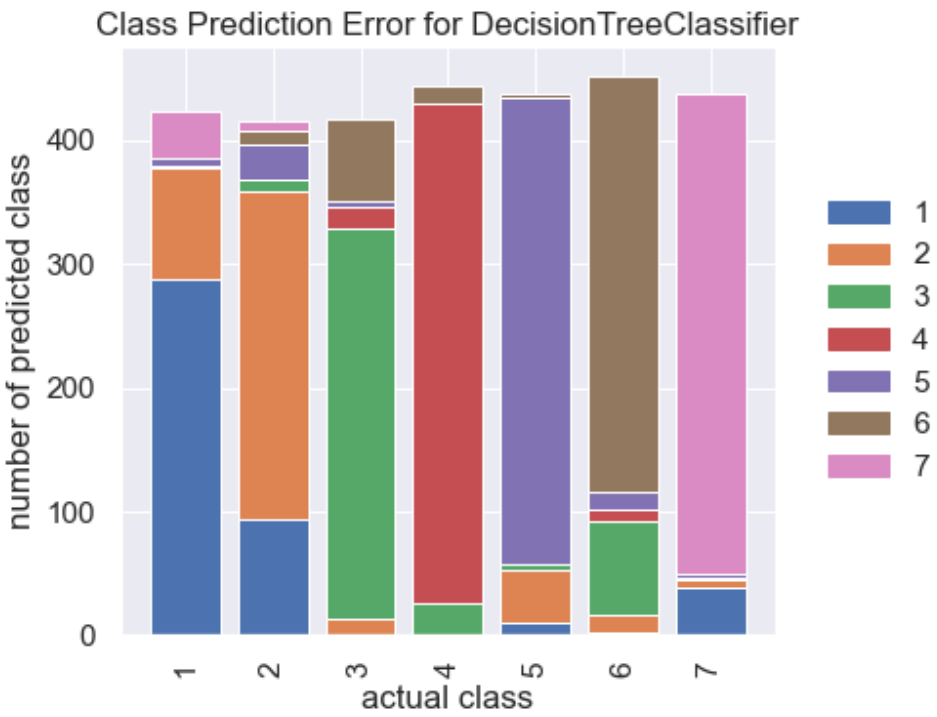
In [71]:
```
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(model_tree, X_v
```

```
Accuracy = 0.6868
```

When cross validation is done, it is 10% less effective than our Random Forest baseline

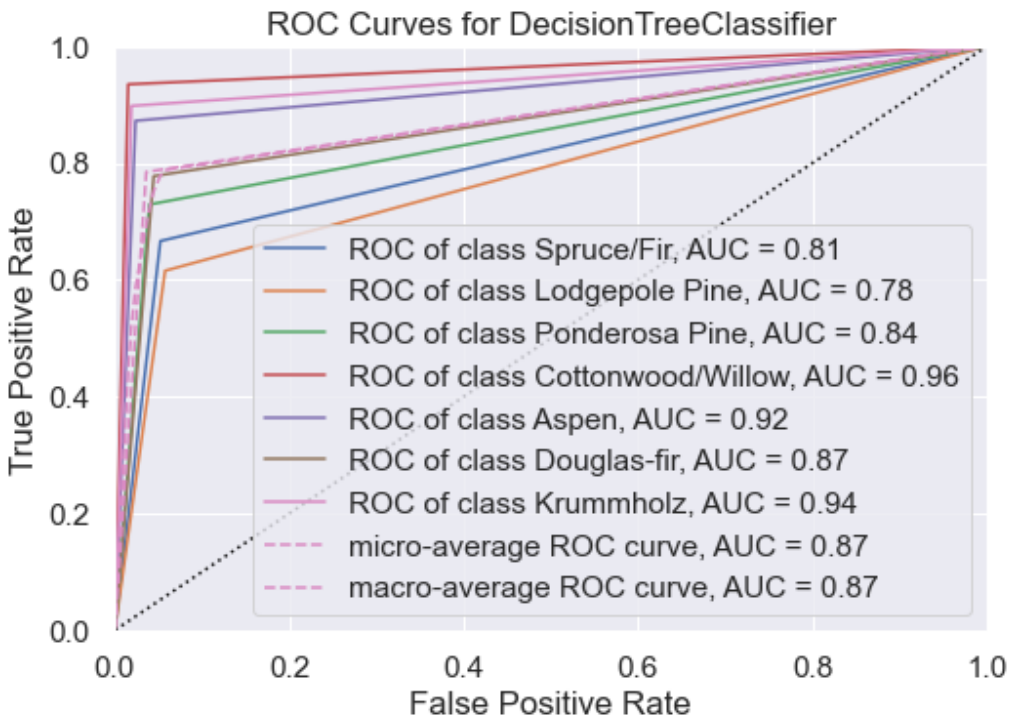
In [72]:
```
visualizer = ClassPredictionError(
    model_tree, palette = 'rainbow_r')
visualizer.fit(X_train, y_train)
visualizer.score(X_val, y_val)
visualizer.show();
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/base.py:450: UserWarning:

X does not have valid feature names, but DecisionTreeClassifier was fitte
d with feature names
```



In [73]:
```
plot_ROC_curve(model_tree, X_train,y_train, X_val, y_val)
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/base.py:450: UserWarning:

X does not have valid feature names, but DecisionTreeClassifier was fitte
d with feature names
```



Out[73]:

```
▸ ROCAUC
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

4.5.K-Nearest Neighbors (KNN)

Now we proceed with KNN setting the hyperparameter n_neighbors to an initial 2

In [74]:

```
knn = KNeighborsClassifier(n_neighbors=2)
model_knn = knn.fit(X_train, y_train)
```

In [75]:

```
knn_predict_train = knn.predict(X_train)
```

```
knn_predict_test = knn.predict(X_val)
```

In [76]:
```
print(knn.score(X_val,y_val))
```

```
0.7824074074074074
```

We get a score of 66%, let's cross validate this result

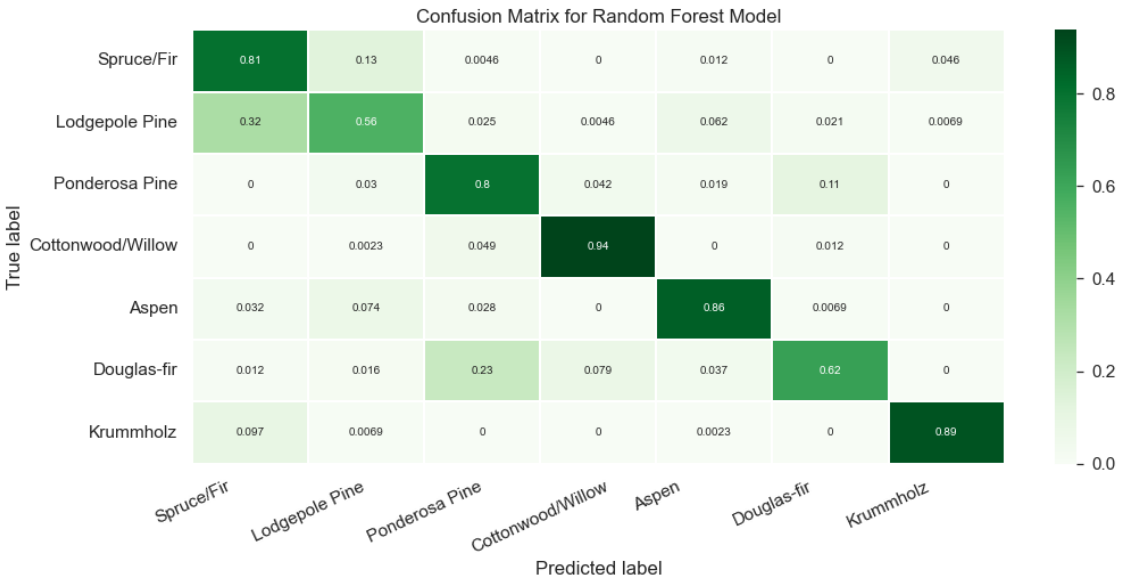
In [77]:
```
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(model_knn, X_va
```

```
Accuracy = 0.6680
```

We get a score of 69%, which decreases the score almost 9% with respect to the initial result.

Our lead baseline is still the CV Random forest at 80%

In [78]:
```
# Get and reshape confusion matrix data
matrix = confusion_matrix(y_val, knn_predict_test)
matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]

# Build the plot
plt.figure(figsize=(16,7))
sns.set(font_scale=1.4)
sns.heatmap(matrix, annot=True, annot_kws={'size':10},
            cmap=plt.cm.Greens, linewidths=0.2)

# Add labels to the plot
class_names = ['Spruce/Fir', 'Lodgepole Pine', 'Ponderosa Pine',
               'Cottonwood/Willow', 'Aspen', 'Douglas-fir',
               'Krummholz']
tick_marks = np.arange(len(class_names))
tick_marks2 = tick_marks + 0.5
plt.xticks(tick_marks, class_names, rotation=25)
plt.yticks(tick_marks2, class_names, rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for Random Forest Model')
plt.show()
```

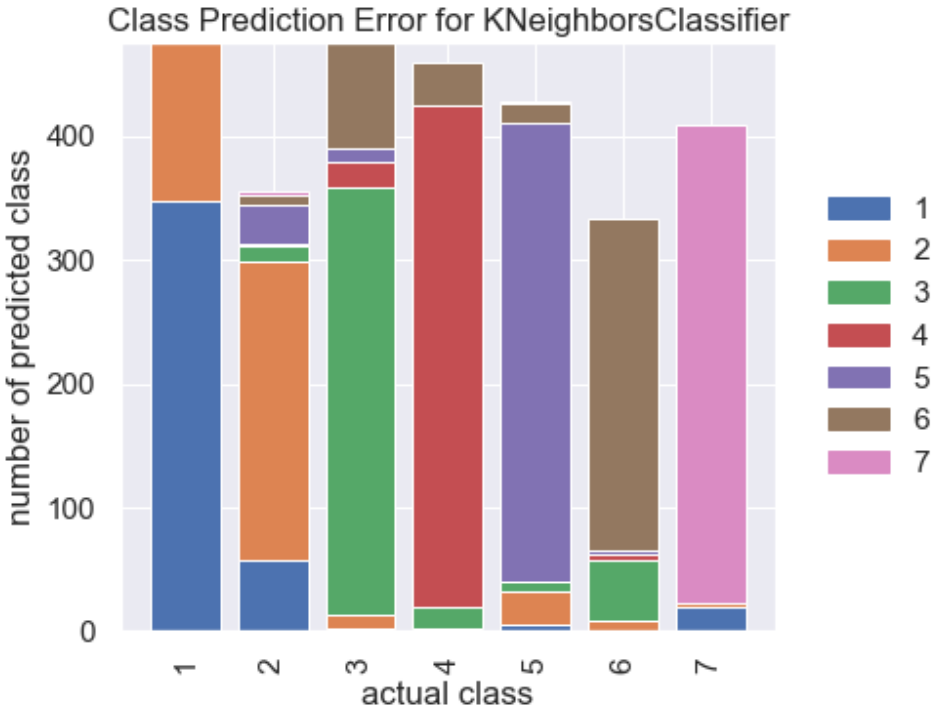


In [79]:
```
print("Training Results:\n")
print(classification_report(y_train, knn_predict_train))
print("\nTesting Results:\n")
print(classification_report(y_val, knn_predict_test))
```

```
Training Results:

              precision    recall  f1-score   support

           1       0.80      1.00      0.89      1728
           2       0.92      0.80      0.85      1728
           3       0.85      0.98      0.91      1728
           4       0.96      0.97      0.96      1728
           5       0.99      0.94      0.96      1728
           6       1.00      0.81      0.89      1728
           7       1.00      0.96      0.98      1728

    accuracy                           0.92     12096
   macro avg       0.93      0.92      0.92     12096
weighted avg       0.93      0.92      0.92     12096


Testing Results:

              precision    recall  f1-score   support

           1       0.64      0.81      0.71       432
           2       0.68      0.56      0.61       432
           3       0.70      0.80      0.75       432
           4       0.88      0.94      0.91       432
           5       0.87      0.86      0.86       432
           6       0.81      0.62      0.70       432
           7       0.94      0.89      0.92       432

    accuracy                           0.78      3024
   macro avg       0.79      0.78      0.78      3024
weighted avg       0.79      0.78      0.78      3024
```

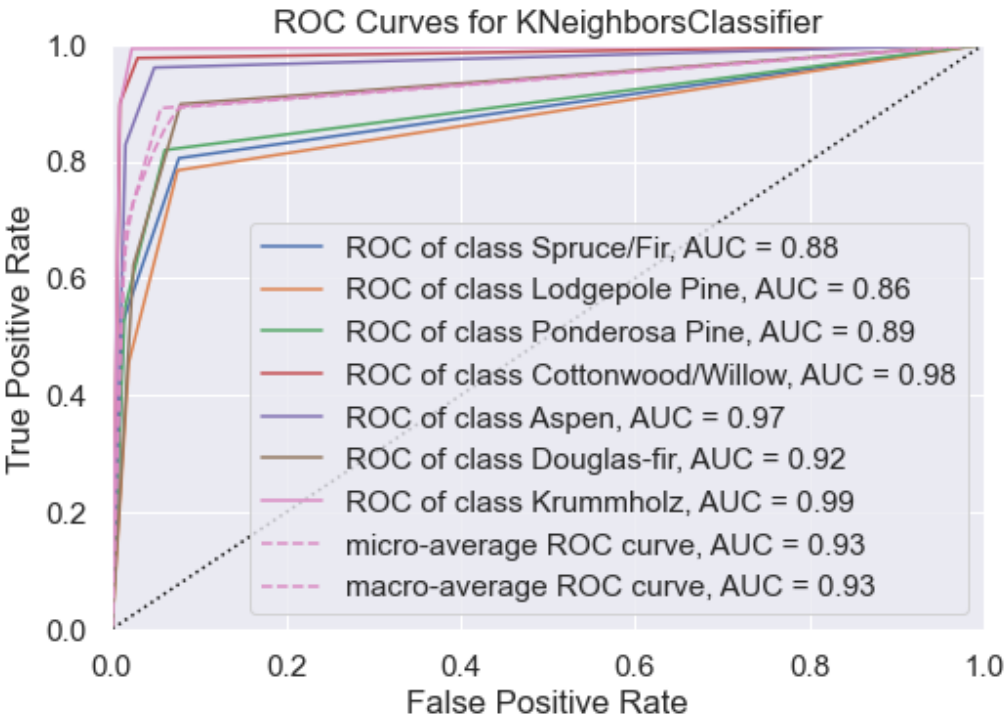
In [80]:
```
visualizer = ClassPredictionError(
    model_knn, palette = 'rainbow_r')
visualizer.fit(X_train, y_train)
visualizer.score(X_val, y_val)
visualizer.show();
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/base.py:450: UserWarning:

X does not have valid feature names, but KNeighborsClassifier was fitted
with feature names
```



In [81]:
```
plot_ROC_curve(model_knn, X_train,y_train, X_val, y_val)
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/base.py:450: UserWarning:

X does not have valid feature names, but KNeighborsClassifier was fitted
with feature names
```



Out[81]:

ROCAUC

```
▸ estimator: KNeighborsClassifier
    ▸ KNeighborsClassifier
```

4.6.Logistic Regression

Lastly, we use logistic regression setting up the multiclass and a basic lbfgs solver for our baseline

In [82]:
```
# define the multinomial logistic regression model
logit = LogisticRegression(multi_class='multinomial', solver='lbfgs')

# define the model evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
```

In [83]:
```
model_logit = logit.fit(X_train, y_train)
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/linear_model/_logistic.py:444: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression
```

In [84]:
```
logit.score(X_val,y_val)
```

Out[84]:
```
0.7192460317460317
```

We get an initial score of 72%, let's see with cross validation

In [85]:
```
# report the model performance
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(model_logit, X_
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/linear_model/_logistic.py:444: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
```

```
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression

/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/linear_model/_logistic.py:444: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression

/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/linear_model/_logistic.py:444: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression

/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/linear_model/_logistic.py:444: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression
```

```
Accuracy = 0.6987
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/linear_model/_logistic.py:444: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression
```

Score decreases 2%

Random Forest still leads.

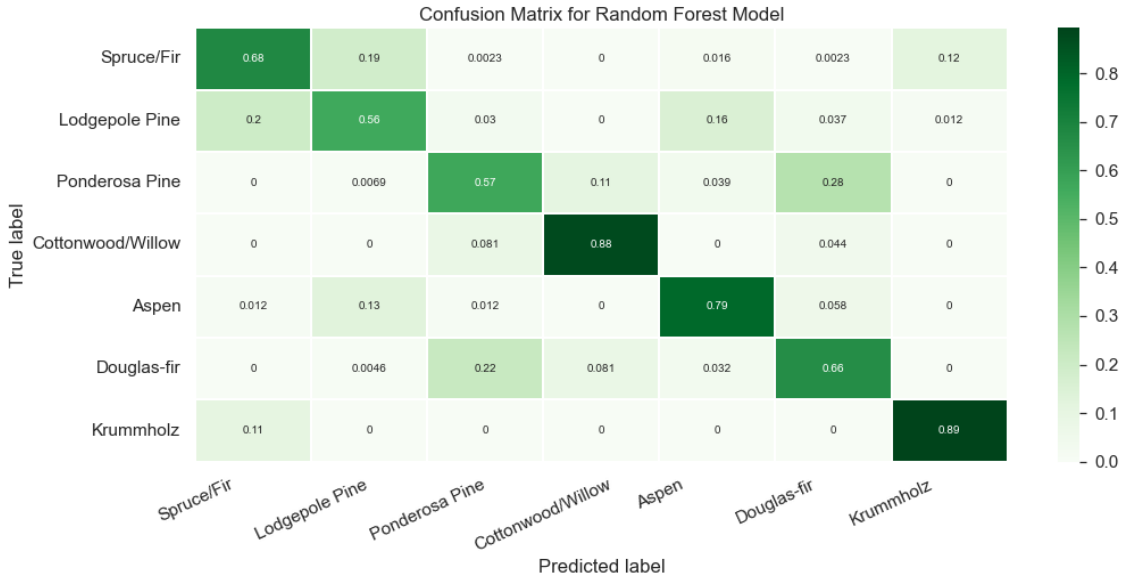
In [86]:
```
logit_fit_train = logit.predict(X_train)
logit_fit_val = logit.predict(X_val)
```

In [87]:
```
print(classification_report(y_val, logit_fit_val))
```

```
              precision    recall  f1-score   support

           1       0.68      0.68      0.68       432
           2       0.64      0.56      0.60       432
           3       0.62      0.57      0.59       432
           4       0.82      0.88      0.85       432
           5       0.77      0.79      0.78       432
           6       0.61      0.66      0.64       432
           7       0.88      0.89      0.88       432

    accuracy                           0.72      3024
   macro avg       0.72      0.72      0.72      3024
weighted avg       0.72      0.72      0.72      3024
```

In [88]:
```
# Get and reshape confusion matrix data
matrix = confusion_matrix(y_val, logit_fit_val)
matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]

# Build the plot
plt.figure(figsize=(16,7))
sns.set(font_scale=1.4)
sns.heatmap(matrix, annot=True, annot_kws={'size':10},
            cmap=plt.cm.Greens, linewidths=0.2)

# Add labels to the plot
class_names = ['Spruce/Fir', 'Lodgepole Pine', 'Ponderosa Pine',
               'Cottonwood/Willow', 'Aspen', 'Douglas-fir',
               'Krummholz']
tick_marks = np.arange(len(class_names))
tick_marks2 = tick_marks + 0.5
plt.xticks(tick_marks, class_names, rotation=25)
plt.yticks(tick_marks2, class_names, rotation=0)
```

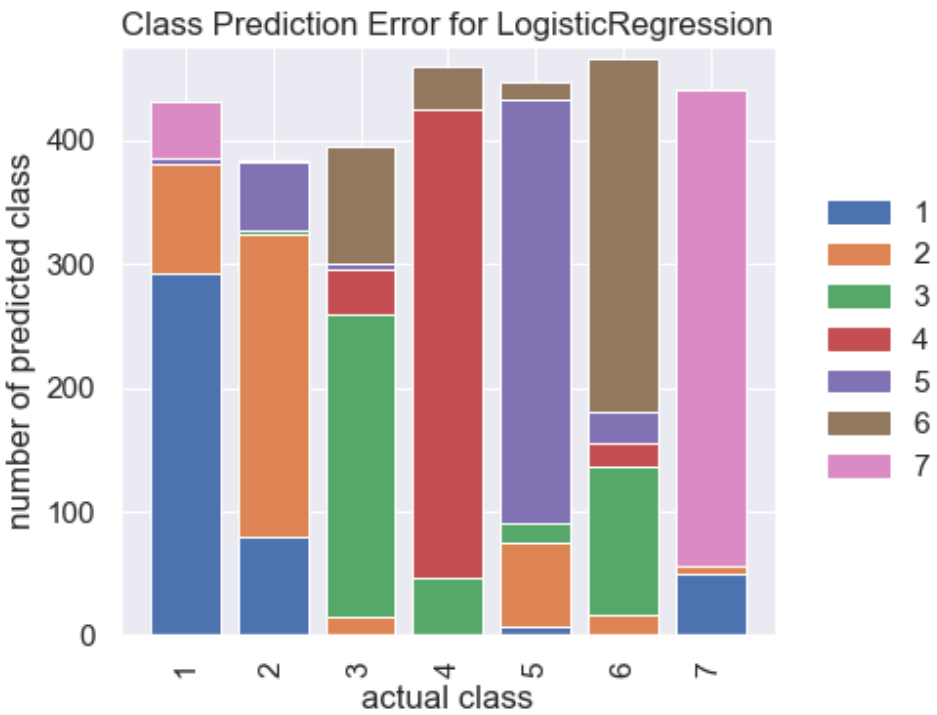
```
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for Random Forest Model')
plt.show()
```



In [89]:

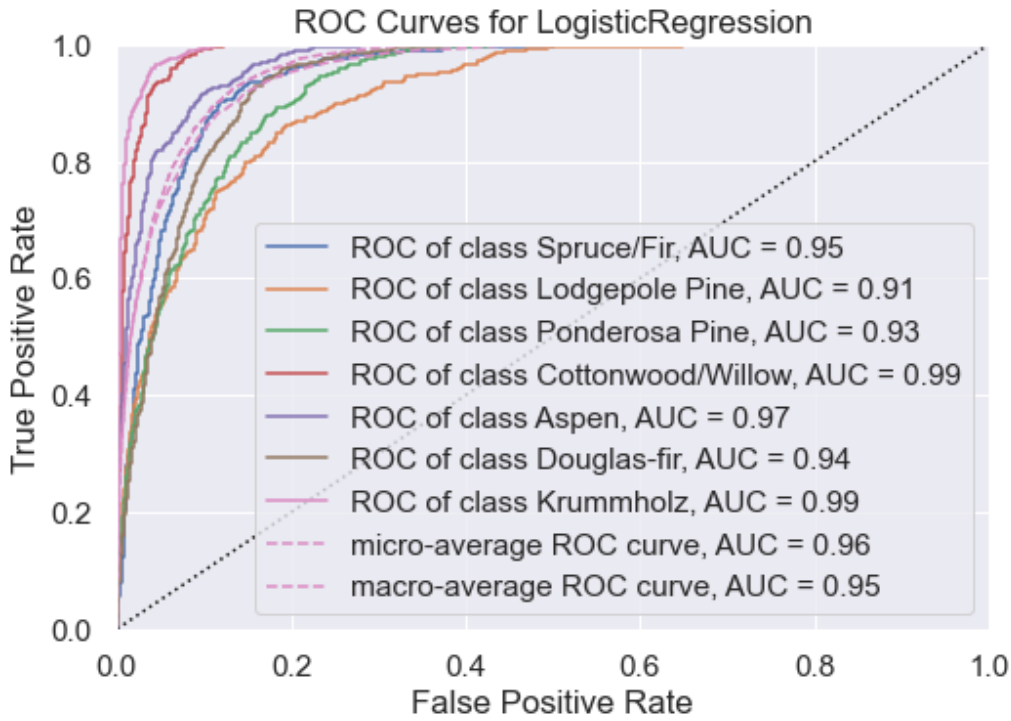
```
visualizer = ClassPredictionError(
    model_logit, palette = 'rainbow_r')
visualizer.fit(X_train, y_train)
visualizer.score(X_val, y_val)
visualizer.show();
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/base.py:450: UserWarning:

X does not have valid feature names, but LogisticRegression was fitted wi
th feature names
```



In [90]:
```
plot_ROC_curve(model_logit, X_train,y_train, X_val, y_val)
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/base.py:450: UserWarning:

X does not have valid feature names, but LogisticRegression was fitted wi
th feature names
```



Out[90]:
```
▸ ROCAUC
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

4.7.Naive Bayes

Naive Bayes models are a group of extremely fast and simple classification algorithms that are often suitable for very high-dimensional datasets. Because they are so fast and have so few tunable parameters, they end up being very useful as a quick-and-dirty baseline for a classification problem.

In [91]:
```
bayes = GaussianNB()
model_bayes = bayes.fit(X_train, y_train)
```

In [92]:
```
bayes.score(X_val,y_val)
```

Out[92]:
```
0.47685185185185186
```

This is the worst score by **far**

In [93]:
```
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(model_bayes, X_
```

```
Accuracy = 0.4517
```

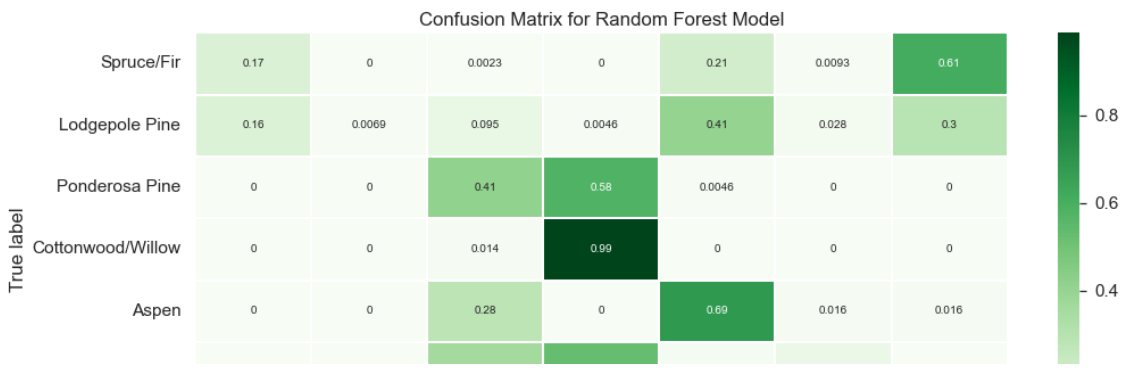
It does not get better...

In [94]:
```
bayes_fit_train = bayes.predict(X_train)
bayes_fit_val = bayes.predict(X_val)
```

In [95]:
```
print(classification_report(y_val, bayes_fit_val))
```

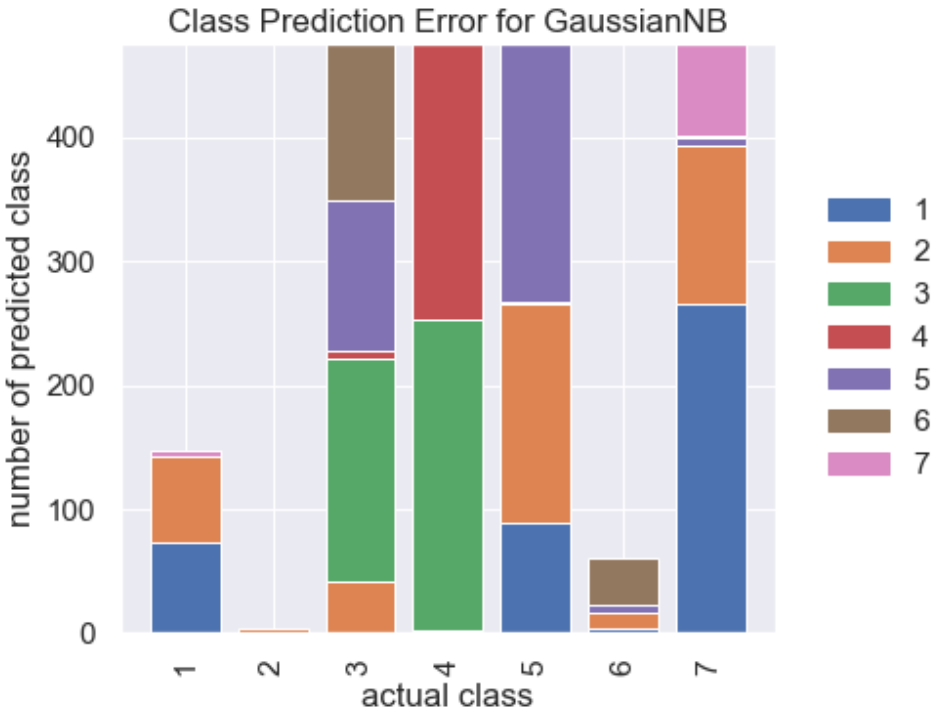
```
              precision    recall  f1-score   support

           1       0.50      0.17      0.25       432
           2       1.00      0.01      0.01       432
           3       0.35      0.41      0.38       432
           4       0.47      0.99      0.64       432
           5       0.52      0.69      0.59       432
           6       0.62      0.09      0.15       432
           7       0.52      0.99      0.68       432

    accuracy                           0.48      3024
   macro avg       0.57      0.48      0.39      3024
weighted avg       0.57      0.48      0.39      3024
```

In [96]:
```
# Get and reshape confusion matrix data
matrix = confusion_matrix(y_val, bayes_fit_val)
matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]

# Build the plot
plt.figure(figsize=(16,7))
sns.set(font_scale=1.4)
sns.heatmap(matrix, annot=True, annot_kws={'size':10},
            cmap=plt.cm.Greens, linewidths=0.2)

# Add labels to the plot
class_names = ['Spruce/Fir', 'Lodgepole Pine', 'Ponderosa Pine',
               'Cottonwood/Willow', 'Aspen', 'Douglas-fir',
               'Krummholz']
tick_marks = np.arange(len(class_names))
tick_marks2 = tick_marks + 0.5
plt.xticks(tick_marks, class_names, rotation=25)
plt.yticks(tick_marks2, class_names, rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for Random Forest Model')
plt.show()
```



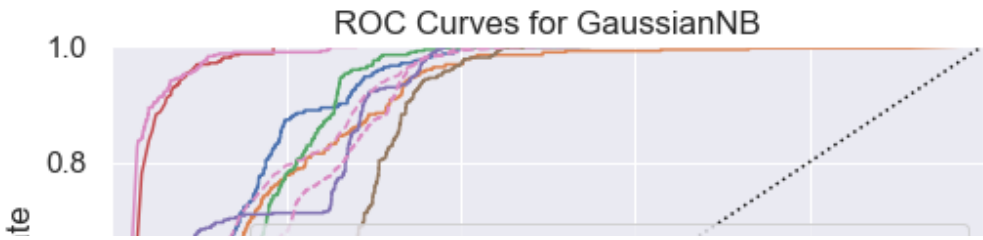
In [97]:
```
visualizer = ClassPredictionError(
    model_bayes, palette = 'rainbow_r')
visualizer.fit(X_train, y_train)
visualizer.score(X_val, y_val)
visualizer.show();
```

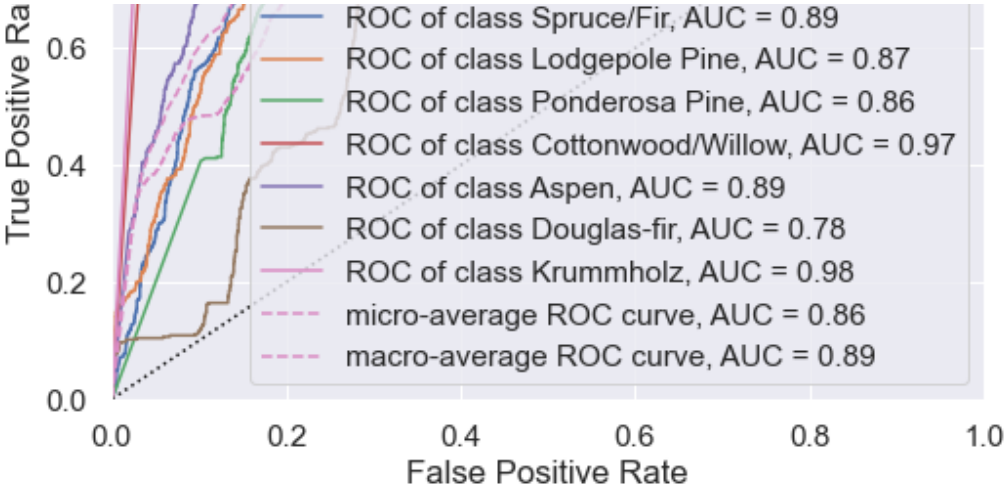
```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/base.py:450: UserWarning:

X does not have valid feature names, but GaussianNB was fitted with featu
re names
```



In [98]:
```
plot_ROC_curve(model_bayes, X_train,y_train, X_val, y_val)
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn
/base.py:450: UserWarning:

X does not have valid feature names, but GaussianNB was fitted with featu
re names
```





Out[98]:

```
▸ ROCAUC
▸ estimator: GaussianNB
  ▸ GaussianNB
```

4.8.Model Comparison

A macro-average will compute the metric independently for each class and then take the average (hence treating all classes equally), whereas a micro-average will aggregate the contributions of all classes to compute the average metric. In a multi-class classification setup, micro-average is preferable if you suspect there might be class imbalance (i.e you may have many more examples of one class than of other classes). Since we do not have a inbalance of classes and all classes are evently distributed we can use the macro-average, which are equal to the micro-average of all algorithms.

Work it out in Python

Algorithms	Accuracy	CV Accuracy	Precision	Recall	F1
Random Forest	**0.87**	**0.78**	0.87	0.87	0.87
Gadient Boosting	***0.75***	**0.66**	0.66	0.66	0.66
Decision Trees	**0.79**	**0.68**	0.78	0.79	0.78
K- Nearest Neighbors(KNN)	**0.78**	**0.66**	0.79	0.78	0.78
Logistic Regression	**0.72**	**0.69**	0.72	0.72	0.72
Naive Bayes	**0.48**	**0.45**	0.57	0.48	0.39

Our best model is **Random Forest (80%)**, so this will be our baseline model to compare with.

Machine Learning II
Group Assignment