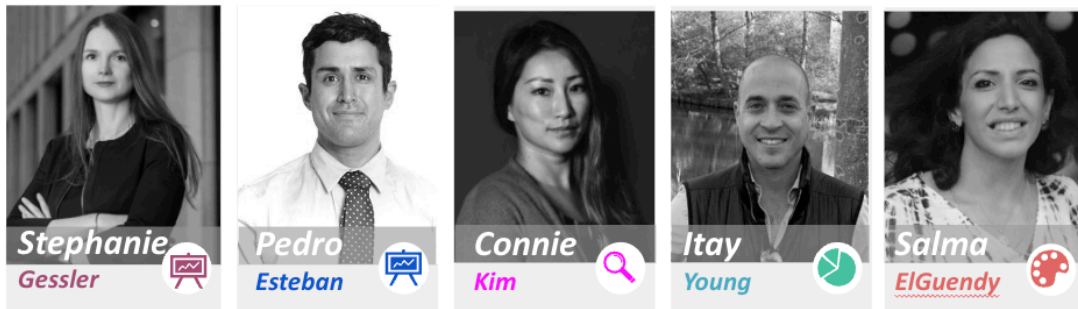


Machine Learning II Group Assignment



Group A:

MEET THE TEAM



GROUP A

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

Introduction

This is a continuation of forest_cover_type_detector_gr_a_Part1.

Above we import the files created in the previous notebook so that this notebook can run independently

Table of contents is an extension of the previous notebook

Table of contents

- [Libraries used](#)
- [0 Import Data](#)

5. Import Data

5. Feature Engineering

- 5.1 Check for Anomalies and Outliers
 - 5.1.1 Outlier Detection Treatment using Inter-Quartile Range rule Function
 - 5.1.2 Inter-Quartile Range rule: 4 IQR from Median
 - 5.1.3 Inter-Quartile Range rule: 3 IQR from Median
- 5.2 Feature Transformation and Building of new features
 - 5.2.1 Bivariate Combinations
 - 5.2.2 Polynominal
 - 5.2.3 ID
 - 5.2.4 Distance to Hydrology
 - 5.2.5 Horizontal Distance To Roadways
 - 5.2.6 Slope
 - 5.2.7 Horizontal Distance To Fire Points
 - 5.2.8 Hillshade
 - 5.2.8.1 Mean Hillshade
 - 5.2.8.2 Hillshade 9am
 - 5.2.8.3 Hillshade Noon
 - 5.2.8.4 Hillshade 3pm
 - 5.2.8.5 Hillshade Ratios
 - 5.2.8.6 Hillshades All Day
 - 5.2.9 Aspect
 - 5.2.9.2 Aspect Degrees
 - 5.2.10 Elevation
 - 5.2.11 Adding and subtracting same scale units
 - 5.2.12 Geoclimate
 - 5.2.12.1 Climatic Zone feature engineering to group soils
 - 5.2.12.2 Geological feature engineering to group soils
 - 5.2.13 Soil Features
 - 5.2.13.1 Soil Type Family
 - 5.2.13.2 Soil Complex
 - 5.2.13.3 Stonyness

6. Feature Selection

- 6.0 Prepare Data and Standardization
- 6.1 Single tree
- 6.2 Bagging
- 6.3 Random Forest
- 6.4 Extra Trees
 - 6.4.1 Feature Number Selecion
 - 6.4.2 Lasso Regularization
 - 6.4.3 Filter Methods
- 6.5 Recursive Feature Elimination
- 6.6 Tree Based Methodologies
 - 6.6.1 RandomForestClassifier
 - 6.6.2 XGBoost
 - 6.6.3 Extra trees Classifier
- 6.7 Score of all methods Together

Libraries used

```
In [1]: #!pip install squarify
#!pip install htmltabletomd
#!pip install GraphViz
#!pip install pygraphviz
```

```
In [2]: import pandas as pd
import numpy as np
```

```
import math
import seaborn as sns # Graphing
import matplotlib.pyplot as plt
import squarify #treemap
import matplotlib.pyplot as plt
import warnings
import plotly.graph_objects as go
import xgboost as xgb
import scipy.stats as stats
import htmltabletomd
import pydotplus
```

```
In [3]: from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import KBinsDiscretizer

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.linear_model import LogisticRegression

from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor

from sklearn import datasets
from sklearn import linear_model
```

```

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold

from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import RFE
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import f_classif

from sklearn.preprocessing import MinMaxScaler

from scipy.stats import norm

from yellowbrick.target import FeatureCorrelation
from yellowbrick.classifier import ROCAUC
from yellowbrick.model_selection import rfecv

from plotly.subplots import make_subplots
from IPython.display import Image
from io import StringIO

from itertools import combinations

from dtreeviz.trees import *

from numpy import percentile

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

```

0.Import the Data

Let's load the original Kaggle training and test data and create a data frame

```
In [4]: data_train = pd.read_csv("train.csv")
data_test = pd.read_csv("test.csv")
```

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

```
In [5]: df_original = data_train.copy()
```

5.Feature Engineering

5.1. Check for Anomalies and Outliers

Since the Z-score is sensitive to the mean and standard deviation and its assumption is a normal distribution, we cannot use the z-score for outlier handling because our data is skewed and failed to pass the normal test. Our data is not normally distributed or at least not just yet.

The disadvantage using percentiles is that it considers always elements at both sides of the spectrum of the lowest or highest value, which can potentially be mistaken as outliers.

As the number of observations increases, so does the number of observations considered outliers. After all, using a percentile based method will always flat-out and reject a certain percentage of our observations. Thus, we need to use the percentiles with caution.

5.1.1 Outlier Detection Treatment using Inter-Quartile Range rule Function

The IQR is the difference between the 75th and 25th percentile. The IQR is more resistant to outliers. The IQR by definition only covers the middle 50% of the data, so outliers are well outside this range and the presence of a small number of outliers is not likely to change this significantly.

Now we are testing different ranges for IQR, namely 2,3 and 4 to check for more extreme outlier values.

```
In [6]: def outlier_function(df, col_name,value_IQR):
        ''' This function detects first and third quartile and interquartile range for a given column
        of a dataframe. Then calculates upper and lower limits to determine outliers conservatively a
        returns the number of lower and uper limit and number of outliers respectively
        '''
        first_quartile = np.percentile(np.array(df[col_name].tolist()), 25)
        third_quartile = np.percentile(np.array(df[col_name].tolist()), 75)
        IQR = third_quartile - first_quartile

        upper_limit = third_quartile+(value_IQR*IQR)
        lower_limit = first_quartile-(value_IQR*IQR)
        outlier_count = 0

        for value in df[col_name].tolist():
            if (value < lower_limit) | (value > upper_limit):
                outlier_count +=1
        return lower_limit, upper_limit, outlier_count
```

5.1.2 Inter-Quartile Range rule: 4 IQR from Median

We loop through all columns to see if there are any outliers, for all values which are not only 0 and 1

```
In [7]: for column in ["Horizontal_Distance_To_Hydrology","Vertical_Distance_To_Hydrology","Horizontal_Di
        if outlier_function(data_train, column,4)[2] > 0:
            print("There are {} outliers in {}".format(outlier_function(data_train, column,4)[2], col
```

There are 13 outliers in Vertical_Distance_To_Hydrology
There are 1 outliers in Hillshade_9am

There is 1 record of Hillshade_9am with a zero value, which is a valid value as Hillshade can be zero. This is because there are parts in the mountain that never see the sunlight (blind spots). Hence we keep the value as it is.

Now we remove outliers and test again in our baseline model

```
In [8]: q25, q75 = percentile(data_train['Vertical_Distance_To_Hydrology'], 25), percentile(data_train['V
iqr = q75 - q25
# calculate the outlier cutoff
cut_off = iqr * 4
lower, upper = q25 - cut_off, q75 + cut_off
# remove outliers
data_train_vd_h = data_train[(data_train['Vertical_Distance_To_Hydrology'] > lower) & (data_train
```

Checking up to see if the model improves after removing vertical distance to hydrology

```
In [9]: X4=data_train_vd_h.drop(labels=['Id','Cover_Type'],axis=1)
        y4=data_train_vd_h['Cover_Type']
```

```
In [10]: X4_train, X4_val, y4_train, y4_val = train_test_split(
        X4, y4, test_size=0.20, random_state=42,stratify=y4)
```

```
In [11]: scaler = StandardScaler()
        scale_numerical=['Elevation','Aspect','Slope','Horizontal_Distance_To_Hydrology','Vertical_Distan
        'Horizontal_Distance_To_Roadways','Hillshade_9am','Hillshade_Noon','Hillshade_3pm',
        'Horizontal_Distance_To_Fire_Points']

        X4_train[scale_numerical] = scaler.fit_transform(X4_train[scale_numerical])
        X4_val[scale_numerical] = scaler.transform(X4_val[scale_numerical])
```

```
In [12]: y4.value_counts()
```

```
Out[12]: 5    2160
         3    2160
         6    2160
         4    2160
         1    2159
         2    2157
```

```
7      2151
Name: Cover_Type, dtype: int64
```

```
In [13]: forest_iqr4 = RandomForestClassifier(random_state=37)
model_forest_iqr4 = forest_iqr4.fit(X4_train,y4_train)
```

```
In [14]: # calculating accuracy score
model_forest_iqr4.score(X4_val,y4_val)
```

```
Out[14]: 0.8550628722700199
```

```
In [15]: forest = RandomForestClassifier(random_state=37)
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(model_forest_iqr4, X4_val, y4_val))))

Accuracy = 0.7783
```

Algorithm	Accuracy	CV Accuracy	Accuracy with IQR4	CV Accuracy with IQR4
Random Forest	0.87	0.78	0.86	0.79

Comparing the previous score with the new score after removing the outliers of vertical distance to Hydrology: Accuracy decreases slightly, hence better not to remove outliers.

Replacing with Median

Since removing outliers improved performance of our model, using median values to keep a balanced sample set seems to be a reasonable approach. Otherwise the data becomes unbalanced, for which, other tools have to be used

```
med = np.median(data_train['Vertical_Distance_To_Hydrology'])
for i in data_train['Vertical_Distance_To_Hydrology']:
    if i > upper or i < lower:
        data_train['Vertical_Distance_To_Hydrology_n'] = data_train['Vertical_Distance_To_Hydrology'].replace(i, med)
Xmed=data_train.drop(labels=['Id','Cover_Type','Vertical_Distance_To_Hydrology'],axis=1)
ymed=data_train['Cover_Type']
from sklearn.preprocessing import StandardScaler
scale_numerical = ['Elevation','Aspect','Slope','Horizontal_Distance_To_Hydrology','Vertical_Distance_To_Hydrology_n',
'Horizontal_Distance_To_Roadways','Hillshade_9am','Hillshade_Noon','Hillshade_3pm', 'Horizontal_Distance_To_Fire_Points']
scaler = StandardScaler()
Xmed[scale_numerical]=scaler.fit_transform(Xmed[scale_numerical])
ymed.value_counts()
Xmed_train,Xmed_val,ymed_train,ymed_val = train_test_split(X4,y4,random_state=37) #seed is 18!
Cannot use stratify because the dataset is unbalanced
forest = RandomForestClassifier(n_estimators=20)
model_forest = forest.fit(Xmed_train,ymed_train)
# calculating accuracy score
forest.score(Xmed_val,ymed_val)
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(model_forest, Xmed_val, ymed_val))))
```

5.1.3 Inter-Quartile Range rule: 3 IQR

First we are checking which variables will be removed using the 3 IQR dunction

Hillshade_9am,Hillshade_Noon,Hillshade_3pm has been excluded as these are valid outliers.

```
In [16]: for column in ["Horizontal_Distance_To_Hydrology", "Vertical_Distance_To_Hydrology", "Horizontal_Di
if outlier_function(data_train, column,3)[2] > 0:
print("There are {} outliers in {}".format(outlier_function(data_train, column,3)[2], col
```

```
There are 53 outliers in Horizontal_Distance_To_Hydrology
There are 49 outliers in Vertical_Distance_To_Hydrology
There are 3 outliers in Horizontal_Distance_To_Roadways
There are 132 outliers in Horizontal_Distance_To_Fire_Points
```

```
In [17]: cols = ["Horizontal_Distance_To_Hydrology", "Vertical_Distance_To_Hydrology", "Horizontal_Distance
Q1 = data_train[cols].quantile(0.25)
Q3 = data_train[cols].quantile(0.75)
IQR = Q3 - Q1

df = data_train[~((data_train[cols] < (Q1 - 3 * IQR)) | (data_train[cols] > (Q3 + 3 * IQR))).any(a
```

```
In [18]: df.shape
```

```
Out[18]: (14866, 56)
```

```
In [19]: x3=df.drop(labels=['Cover_Type'],axis=1)
y3=df['Cover_Type']
```

```
In [22]: X_train3,X_val3,y_train3,y_val3 = train_test_split (X3,y3,random_state=37) #seed is 37!

In [23]: scaler = StandardScaler()
scale_numerical=['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']

X_train3[scale_numerical] = scaler.fit_transform(X_train3[scale_numerical])
X_val3[scale_numerical] = scaler.transform(X_val3[scale_numerical])
```

Looking at the cover types you can see the values become unbalanced if we decided to remove them.

```
In [24]: y3.value_counts()

Out[24]: 4    2160
        6    2159
        3    2153
        5    2121
        1    2113
        7    2088
        2    2072
        Name: Cover_Type, dtype: int64

In [25]: forest_iqr3 = RandomForestClassifier(random_state=37)
model_forest_iqr3 = forest_iqr3.fit(X_train3,y_train3)

In [26]: model_forest_iqr3.score(X_val3,y_val3)

Out[26]: 0.8746300780199086

In [27]: forest_iqr3 = RandomForestClassifier(random_state=37)
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(model_forest_iqr3, X_val3, y_val3))))

Accuracy = 0.8041
```

Algorithm	Accuracy Baseline	CV Accuracy Baseline	Accuracy with IQR3	CV Accuracy with IQR3
Random Forest	0.87	0.78	0.87	0.80

Using the IQR3 rule to remove all outliers does not significantly improve the model hence we will not remove any outliers using IQR3. Plus some are valid outliers like Hillshade of zero.

5.1.3 Inter-Quartile Range rule: 1.5 IQR

First we are checking which variables will be removed using the 3 IQR dunction

Hillshade_9am,Hillshade_Noon,Hillshade_3pm has been excluded as these are valid outliers.

```
In [28]: for column in ["Horizontal_Distance_To_Hydrology","Vertical_Distance_To_Hydrology","Horizontal_Distance_To_Roadways","Horizontal_Distance_To_Fire_Points"]:
        if outlier_function(data_train, column,1.5)[2] > 0:
            print("There are {} outliers in {}".format(outlier_function(data_train, column,1.5)[2], column))

There are 512 outliers in Horizontal_Distance_To_Hydrology
There are 586 outliers in Vertical_Distance_To_Hydrology
There are 830 outliers in Horizontal_Distance_To_Roadways
There are 645 outliers in Horizontal_Distance_To_Fire_Points

In [29]: cols = ["Horizontal_Distance_To_Hydrology","Vertical_Distance_To_Hydrology","Horizontal_Distance_To_Roadways","Horizontal_Distance_To_Fire_Points"]

Q1 = data_train[cols].quantile(0.25)
Q3 = data_train[cols].quantile(0.75)
IQR = Q3 - Q1

df1 = data_train[~((data_train[cols] < (Q1 - 1.5 * IQR)) | (data_train[cols] > (Q3 + 1.5 * IQR)))].

In [30]: df1.shape

Out[30]: (12261, 56)

In [31]: x1=df1.drop(labels=['Cover_Type'],axis=1)
y1=df1['Cover_Type']

In [32]: X_train1,X_val1,y_train1,y_val1 = train_test_split (X1,y1,random_state=37) #seed is 37!
```

```
In [34]: scaler = StandardScaler()
scale_numerical=['Elevation','Aspect','Slope','Horizontal_Distance_To_Hydrology','Vertical_Distan
              'Horizontal_Distance_To_Roadways','Hillshade_9am','Hillshade_Noon','Hillshade_3pm',
              'Horizontal_Distance_To_Fire_Points']

X_train1[scale_numerical] = scaler.fit_transform(X_train1[scale_numerical])
X_val1[scale_numerical] = scaler.transform(X_val1[scale_numerical])
```

Looking at the cover types you can see the values become unbalanced if we decided to remove them.

```
In [35]: y1.value_counts()
```

```
Out[35]: 4    2055
        6    1912
        5    1814
        3    1780
        7    1606
        1    1555
        2    1539
        Name: Cover_Type, dtype: int64
```

```
In [36]: forest_iqr1 = RandomForestClassifier(random_state=37)
model_forest_iqr1 = forest_iqr1.fit(X_train1,y_train1)
```

```
In [37]: model_forest_iqr1.score(X_val1,y_val1)
```

```
Out[37]: 0.8692106979778212
```

```
In [38]: forest_iqr1 = RandomForestClassifier(random_state=37)
print("Accuracy = {0:.4f}".format(np.mean(cross_val_score(model_forest_iqr1, X_val1, y_val1))))
```

Accuracy = 0.8024

Algorithm	Accuracy Baseline	CV Accuracy Baseline	Accuracy with IQR3	CV Accuracy with IQR3
Random Forest	0.87	0.78	0.87	0.80

Using the IQR1.5 rule to remove all outliers does not significantly improve the model hence we will not remove any outliers.

5.2 Feature Transformation and Building of new features

5.2.1 Bivariate Combinations

During feature engineering, we want to try to create a wide variety of interactions between multiple variables in order to create new variables.

By manipulating them together, we create opportunities to have new and impactful features which could potentially impact our target variable, thus engineering our features.

For this argument, we will create as many bivariate combinations of our predicting variables using the 'combinations' method from itertools library.

We will not make interactions with the dummy variables as these are either 0 or 1 and we will not get any additional information from making the interaction this way.

Furthermore, it is not recommended to use standardization before bivariate combinations as we want to increase the signal.

Sources: <https://towardsdatascience.com/feature-engineering-combination-polynomial-features-3caa4c77a755>

<https://samchaaa.medium.com/preprocessing-why-you-should-generate-polynomial-features-first-before-standardizing-892b4326a91d>

In order to use the bivariate combinations we will use the itertools library to create the combinations of our features.

in order to use the bivariate combination we split the dataset for using it. Note this is not the split we will use later for testing the algorithm. This has only the purpose of testing all the combination and selecting the best once.

```
In [39]: # Identify and drop our target variable 'Cover_Type' from dataframe
X = data_train.drop('Cover_Type', axis = 1)

# Isolate our dependent variable as a feature
y = data_train['Cover_Type']
```

Train Test Split (80/20 size), drop duplicates and missing values

```
In [40]: X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = .2, random_state=37, stratify

X_train.drop_duplicates(inplace = True)
X_train.dropna(inplace = True)
```

Here, we create every possible bivariate combination to be tested for feature engineering, no dummies

We take categorical variables prior to our feature engineering

```
In [41]: column_list = X_train.columns
filtered_column_list = [column for column in column_list if 'Soil_Type' not in column and 'Wilder
interactions = list(combinations(filtered_column_list, 2))
interactions
```

```
Out[41]: [('Elevation', 'Aspect'),
('Elevation', 'Slope'),
('Elevation', 'Horizontal_Distance_To_Hydrology'),
('Elevation', 'Vertical_Distance_To_Hydrology'),
('Elevation', 'Horizontal_Distance_To_Roadways'),
('Elevation', 'Hillshade_9am'),
('Elevation', 'Hillshade_Noon'),
('Elevation', 'Hillshade_3pm'),
('Elevation', 'Horizontal_Distance_To_Fire_Points'),
('Aspect', 'Slope'),
('Aspect', 'Horizontal_Distance_To_Hydrology'),
('Aspect', 'Vertical_Distance_To_Hydrology'),
('Aspect', 'Horizontal_Distance_To_Roadways'),
('Aspect', 'Hillshade_9am'),
('Aspect', 'Hillshade_Noon'),
('Aspect', 'Hillshade_3pm'),
('Aspect', 'Horizontal_Distance_To_Fire_Points'),
('Slope', 'Horizontal_Distance_To_Hydrology'),
('Slope', 'Vertical_Distance_To_Hydrology'),
('Slope', 'Horizontal_Distance_To_Roadways'),
('Slope', 'Hillshade_9am'),
('Slope', 'Hillshade_Noon'),
('Slope', 'Hillshade_3pm'),
('Slope', 'Horizontal_Distance_To_Fire_Points'),
('Horizontal_Distance_To_Hydrology', 'Vertical_Distance_To_Hydrology'),
('Horizontal_Distance_To_Hydrology', 'Horizontal_Distance_To_Roadways'),
('Horizontal_Distance_To_Hydrology', 'Hillshade_9am'),
('Horizontal_Distance_To_Hydrology', 'Hillshade_Noon'),
('Horizontal_Distance_To_Hydrology', 'Hillshade_3pm'),
('Horizontal_Distance_To_Hydrology', 'Horizontal_Distance_To_Fire_Points'),
('Vertical_Distance_To_Hydrology', 'Horizontal_Distance_To_Roadways'),
('Vertical_Distance_To_Hydrology', 'Hillshade_9am'),
('Vertical_Distance_To_Hydrology', 'Hillshade_Noon'),
('Vertical_Distance_To_Hydrology', 'Hillshade_3pm'),
('Vertical_Distance_To_Hydrology', 'Horizontal_Distance_To_Fire_Points'),
('Horizontal_Distance_To_Roadways', 'Hillshade_9am'),
('Horizontal_Distance_To_Roadways', 'Hillshade_Noon'),
('Horizontal_Distance_To_Roadways', 'Hillshade_3pm'),
('Horizontal_Distance_To_Roadways', 'Horizontal_Distance_To_Fire_Points'),
('Hillshade_9am', 'Hillshade_Noon'),
('Hillshade_9am', 'Hillshade_3pm'),
('Hillshade_9am', 'Horizontal_Distance_To_Fire_Points'),
('Hillshade_Noon', 'Hillshade_3pm'),
('Hillshade_Noon', 'Horizontal_Distance_To_Fire_Points'),
('Hillshade_3pm', 'Horizontal_Distance_To_Fire_Points')]
```

Addition and division has been taken out as it created a lot of noise in the data. The division makes sense if it has a business meaning and the addition only if it is the same scale.

However we will add the variables which have the same metrics together in a second step but not in a for loop.

```
In [42]: for (key, value) in interactions:
          data_train[key + '_x_' + value] = data_train[key] * data_train[value]
          #data_train[key + '_+' + value] = data_train[key] + data_train[value]
          #data_train[key + '_divide_' + value] = data_train[key] / data_train[value]
```

```
In [43]: pd.set_option('display.max_columns', None) #to make all columns visible in dataframe now that we
          data_train
```

```
Out[43]:
```

	Id	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal
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	
...	
15115	15116	2607	243	23	258	7	
15116	15117	2603	121	19	633	195	
15117	15118	2492	134	25	365	117	
15118	15119	2487	167	28	218	101	
15119	15120	2475	197	34	319	78	

15120 rows x 101 columns

5.2.2 Polynomial Features

We have just seen how to make two variables interact together, but sometimes the relationship between dependent and independent variables are more complex and not linear.

Polynomials is another way to create new features! A very strong option for new features is increasing the power of a single variable.

For our purposes, we will try and see if all the existing variables, can improve our Baseline by being increased to the power.

Source: <https://towardsdatascience.com/feature-engineering-combination-polynomial-features-3caa4c77a755>

Here we select only the columns we are interested in, this is from column 2 to 9

```
In [44]: x_train_int_pf = data_train.iloc[:, 1:10]
          x_train_int_pf
```

```
Out[44]:
```

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distan
0	2596	51	3	258	0	
1	2590	56	2	212	-6	
2	2804	139	9	268	65	
3	2785	155	18	242	118	
4	2595	45	2	153	-1	
...	
15115	2607	243	23	258	7	
15116	2603	121	19	633	195	
15117	2492	134	25	365	117	
15118	2487	167	28	218	101	
15119	2475	197	34	319	78	

15120 rows x 9 columns

By default, when running polynomials we lose the name of our labels. This function gives us the ability to preserve it with the name + the transformation done

```
In [45]: def PolynomialFeatures_labeled(input_df,power):
'''Basically this is a cover for the sklearn preprocessing function.
The problem with that function is if you give it a labeled dataframe, it outputs an unlabeled
a whole bunch of unlabeled columns.

Inputs:
input_df = Your labeled pandas dataframe (list of x's not raised to any power)
power = what order polynomial you want variables up to. (use the same power as you want enter

Output:
Output: This function relies on the powers_ matrix which is one of the preprocessing function
outputs a labeled pandas dataframe
'''
poly = PolynomialFeatures(power)
output_nparray = poly.fit_transform(input_df)
powers_nparray = poly.powers_

input_feature_names = list(input_df.columns)
target_feature_names = ["Constant Term"]
for feature_distillation in powers_nparray[1:]:
    intermediary_label = ""
    final_label = ""
    for i in range(len(input_feature_names)):
        if feature_distillation[i] == 0:
            continue
        else:
            variable = input_feature_names[i]
            power = feature_distillation[i]
            intermediary_label = "%s^%d" % (variable,power)
            if final_label == "":
                #If the final label isn't yet specified
                final_label = intermediary_label
            else:
                final_label = final_label + " x " + intermediary_label
    target_feature_names.append(final_label)
output_df = pd.DataFrame(output_nparray, columns = target_feature_names)
return output_df
```

Polynomial features of degree two gives us 46 new features. Since we have already enough information, we will not go for Polynomial three to avoid dimensionality issues later on

```
In [46]: output_df_pw2 = PolynomialFeatures_labeled(X_train_int_pf,2)
pd.set_option('display.max_columns', None)
output_df_pw2.shape
```

Out[46]: (15120, 55)

There are some fields duplicated with respect to the original df. This is a side effect of the function as normal features are replicated to the power of one which is still the same value so we delete these

```
In [47]: column_list = output_df_pw2.columns
cols = [column for column in column_list if '^1' not in column]
output_df_pw2=output_df_pw2[cols]
output_df_pw2
```

```
Out[47]:
```

Constant Term	Elevation^2	Aspect^2	Slope^2	Horizontal_Distance_To_Hydrology^2	Vertical_Distance_To_Hydrolog
------------------	-------------	----------	---------	------------------------------------	-------------------------------

0	1.0	6739216.0	2601.0	9.0	66564.0	
1	1.0	6708100.0	3136.0	4.0	44944.0	
2	1.0	7862416.0	19321.0	81.0	71824.0	42
3	1.0	7756225.0	24025.0	324.0	58564.0	139
4	1.0	6734025.0	2025.0	4.0	23409.0	
...	
15115	1.0	6796449.0	59049.0	529.0	66564.0	
15116	1.0	6775609.0	14641.0	361.0	400689.0	380
15117	1.0	6210064.0	17956.0	625.0	133225.0	136
15118	1.0	6185169.0	27889.0	784.0	47524.0	102
15119	1.0	6125625.0	38809.0	1156.0	101761.0	60

15120 rows × 10 columns

In [48]: `x_train`

Out[48]:

	Id	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal
1439	1440	2828	72	45	228	163	
14821	14822	3516	320	14	323	22	
6922	6923	2702	178	11	85	10	
12884	12885	2200	152	8	0	0	
1589	1590	3296	356	3	270	31	
...	
3492	3493	2155	347	9	30	-3	
7127	7128	3405	47	18	150	10	
2291	2292	2076	297	32	90	39	
2281	2282	2875	202	15	150	-10	
8682	8683	3063	312	13	570	66	

12096 rows × 55 columns

In here, we concatenate our output to consolidate the ponynomials with the feature combinations

In [49]: `data_train = pd.concat([data_train,output_df_pw2], axis=1)`
`data_train`

Out[49]:

	Id	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal
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
...
15115	15116	2607	243	23	258	7
15116	15117	2603	121	19	633	195
15117	15118	2492	134	25	365	117
15118	15119	2487	167	28	218	101
15119	15120	2475	197	34	319	78

15120 rows × 111 columns

This results in a huge dataset. Just for curiosity we ran polynomials to the power of three and see that just the resulting dataframe is already bigger than the consolidated one above

```
In [50]: output_df_pw3 = PolynomialFeatures_labeled(X_train_int_pf,3)
output_df_pw3
```

Out[50]:

	Constant Term	Elevation^1	Aspect^1	Slope^1	Horizontal_Distance_To_Hydrology^1	Vertical_Distance_To_Hydrology
0	1.0	2596.0	51.0	3.0	258.0	
1	1.0	2590.0	56.0	2.0	212.0	-
2	1.0	2804.0	139.0	9.0	268.0	6
3	1.0	2785.0	155.0	18.0	242.0	11
4	1.0	2595.0	45.0	2.0	153.0	-
...
15115	1.0	2607.0	243.0	23.0	258.0	
15116	1.0	2603.0	121.0	19.0	633.0	19
15117	1.0	2492.0	134.0	25.0	365.0	11
15118	1.0	2487.0	167.0	28.0	218.0	10
15119	1.0	2475.0	197.0	34.0	319.0	7

15120 rows × 220 columns

5.2.3 ID

We agree for the test to not remove ID because the ID is the unique identifier to evaluate

For the train we will remove it as it doesn't add any value to the model

```
In [51]: data_train.drop('Id',axis = 1, inplace = True)
```

5.2.4 Distance To Hydrology

New Features

We combine Vertical distance to Hydrology and Horizontal distance to Hydrology since these two are highly correlated. This suggests to attempt a diagonal distance to hidrology using the Pythagoras theorem.

We will call this newly engineered feature, Distance_To_Hydrology

Source : <https://towardsdatascience.com/types-of-transformations-for-better-normal-distribution-61c22668d3b9>

```
In [52]: data_train['Distance_To_Hydrology'] = data_train['Horizontal_Distance_To_Hydrology']**2 + data_train['Vertical_Distance_To_Hydrology']**2
data_train['Distance_To_Hydrology'] = data_train['Distance_To_Hydrology']**0.5
```

```
data_train.head()
```

```
Out[52]:
```

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_1
0	2596	51	3	258	0	
1	2590	56	2	212	-6	
2	2804	139	9	268	65	
3	2785	155	18	242	118	
4	2595	45	2	153	-1	

Square root and logarithm Transformation

Now we are checking the distribution of the newly created variable and see if further transformation is needed.

The Distance to Hydrology inherits skewness from parent variables. It is positively skewed and has zero values.

In order to use log we will use $\log + 1$ in order to use logarithm with zero values.

Source: https://www.youtube.com/watch?v=_c3dVTRIK9c and

Source_2: <https://towardsdatascience.com/types-of-transformations-for-better-normal-distribution-61c22668d3b9>

As a rule of thumb, the skewness can be interpreted as follows:

As rule of thumb, skewness can be interpreted like this:

	Skewness
Fairly Symmetrical	-0.5 to 0.5
Moderate Skewed	-0.5 to -1.0 and 0.5 to 1.0
Highly Skewed	< -1.0 and > 1.0

Source: <https://www.marsja.se/transform-skewed-data-using-square-root-log-box-cox-methods-in-python/>

```
In [53]: print('\033[95m'+"Skew before transformation\n", data_train['Distance_To_Hydrology'].skew(),
          "\nmin\n", data_train['Distance_To_Hydrology'].min(),
          "\nmax\n", data_train['Distance_To_Hydrology'].max(),)
```

```
Skew before transformation
1.4420723247653018
min
0.0
max
1356.9395712411072
```

We do some transformations to minimize skewness

```
In [54]: #Using the log10+ 1 logarithm
data_train['log10_Distance_To_Hydrology'] = np.log10(data_train['Distance_To_Hydrology']+1)
```

```
In [55]: #Using the square root
data_train['sqr_Distance_To_Hydrology'] = data_train['Distance_To_Hydrology']**0.5
```

Results after logarithm and Square root Transformation

```
In [56]: print('\033[92m'+"Skew after Log transformation\n", data_train['log10_Distance_To_Hydrology'].skew(),
          "\nmin\n", data_train['log10_Distance_To_Hydrology'].min(),
          "\nmax\n", data_train['log10_Distance_To_Hydrology'].max(),)
```

```
Skew after Log transformation
-1.6236384008076947
min
0.0
max
3.1328804441267675
```

```
In [57]: print('\033[96m'+"Skew after Square Root Transformation\n", data_train['sqr_Distance_To_Hydrology'].skew(),
          "\nmin\n", data_train['sqr_Distance_To_Hydrology'].min(),
          "\nmax\n", data_train['sqr_Distance_To_Hydrology'].max(),)
```

```

max
data_train['sqr_Distance_To_Hydrology'].max(),)

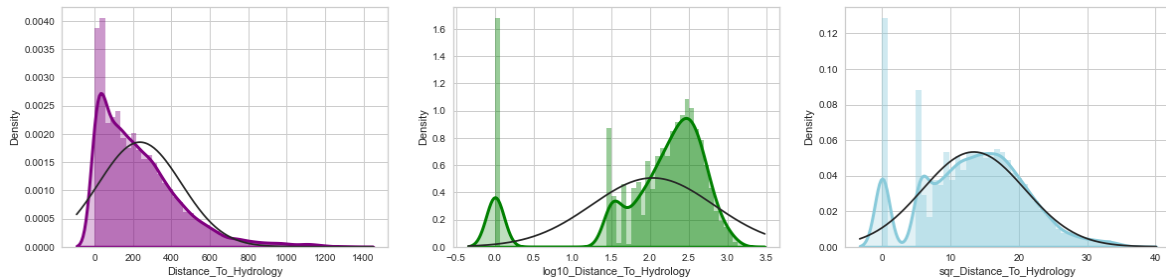
Skew after Square Root Transformation
0.0351310310927601
min
0.0
max
36.836660696120475

```

```

In [58]: def histPlot(first_feature,col):
          sns.distplot(first_feature,color=col,fit = norm,kde = True,kde_kws = {'shade': True, 'linewidth
          f = plt.figure(figsize=(20,15))
          f.add_subplot(331)
          histPlot(data_train['Distance_To_Hydrology'], 'purple')
          f.add_subplot(332)
          histPlot(data_train['log10_Distance_To_Hydrology'], 'green')
          f.add_subplot(333)
          histPlot(data_train['sqr_Distance_To_Hydrology'], 'c')

```



As you can see above, for distance to Hydrology the **square root** showed a better performance in terms of skewness and is closer to a normal bell shaped than the logarithm transformation. We will be using Square Root as a new feature in the dataset and will drop the others from the dataset.

For now we only remove log10 but later we will delete also originals

```

In [59]: data_train.drop(['log10_Distance_To_Hydrology'], axis=1,inplace=True)

```

5.2.5 Horizontal Distance To Roadways

Square root and logarithm Transformation

For log transformation there should be no zeros, negative values and the distribution should be positive skewed (bigger than 1 is positive) hence we are using the square root as you can see for logarithm transformation below the distribution did not improve!!!

```

In [60]: print('\033[95m'+ "Skew before Transformation\n", data_train['Horizontal_Distance_To_Roadways'].sk
          "\nmin before Transformation\n", data_train['Horizontal_Distance_To_Roadways'].min(),
          "\nmax before Transformation\n", data_train['Horizontal_Distance_To_Roadways'].max(),)

Skew before Transformation
1.247810678465482
min before Transformation
0
max before Transformation
6890

```

Results after logarithm and Square root Transformation

```

In [61]: # since we have null values we add plus 1 to avoid log of zero.We are using natural log and log10
data_train['Sqr_Horizontal_Distance_To_Roadways'] = data_train['Horizontal_Distance_To_Roadways'].sk
data_train['log_Horizontal_Distance_To_Roadways'] = np.log(data_train['Horizontal_Distance_To_Roadways'])
data_train['log10_Horizontal_Distance_To_Roadways'] = np.log10(data_train['Horizontal_Distance_To_Roadways'])

```

```

In [62]: print('\033[96m'+ "Skew after Square Root Transformation\n", data_train['Sqr_Horizontal_Distance_To_Roadways'].sk
          "\nmin \n", data_train['Sqr_Horizontal_Distance_To_Roadways'].min(),
          "\nmax \n", data_train['Sqr_Horizontal_Distance_To_Roadways'].max(),)

Skew after Square Root Transformation
0.47623044767204115

```

```
min
0.0
max
83.00602387778854
```

```
In [63]: print('\033[92m' + "Skew after log Transformation\n", data_train['log_Horizontal_Distance_To_Roadw
"\nmin\n", data_train['log_Horizontal_Distance_To_Roadways'].min(),
"\nmax\n", data_train['log_Horizontal_Distance_To_Roadways'].max(),)
```

```
Skew after log Transformation
-0.6892190307043193
min
0.0
max
8.837971491357209
```

```
In [64]: print('\033[92m' + "Skew after log10 transformation\n", data_train['log10_Horizontal_Distance_To_R
"\nmin\n", data_train['log10_Horizontal_Distance_To_Roadways'].min(),
"\nmax\n", data_train['log10_Horizontal_Distance_To_Roadways'].max(),)
```

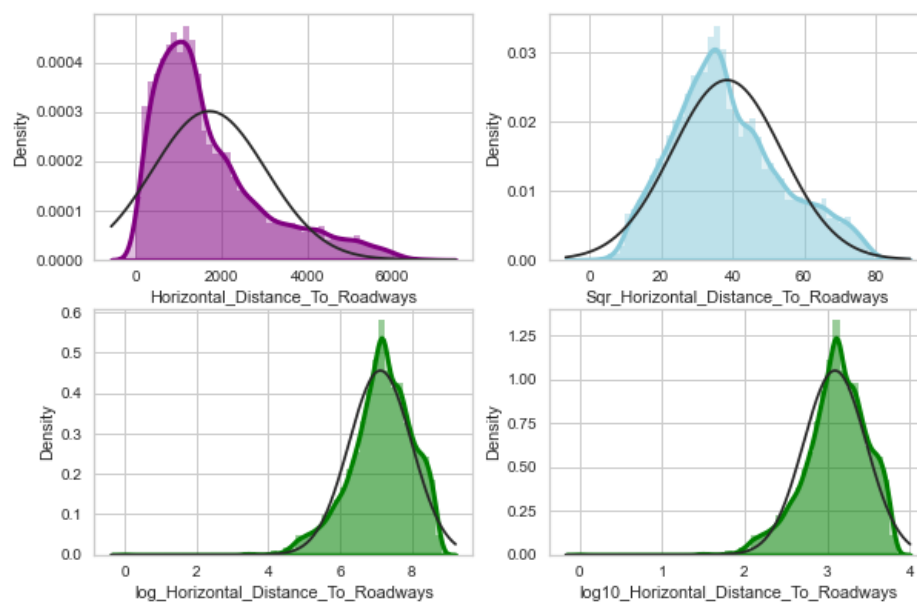
```
Skew after log10 transformation
-0.6892190307043152
min
0.0
max
3.8382822499146885
```

```
In [65]: # testing if the square root is normally distributed and it shows it is not, however it is less s
stats.normaltest(data_train['Sqr_Horizontal_Distance_To_Roadways'])
```

```
Out[65]: NormaltestResult(statistic=593.4024989110219, pvalue=1.3940700201238209e-129)
```

```
In [66]: def histPlot(first_feature,col):
sns.distplot(first_feature,color=col,fit = norm,kde = True,kde_kws = {'shade': True, 'linewidth

f = plt.figure(figsize=(15,10))
f.add_subplot(331)
histPlot(data_train['Horizontal_Distance_To_Roadways'], 'purple')
f.add_subplot(334)
histPlot(data_train['log_Horizontal_Distance_To_Roadways'], 'green')
f.add_subplot(335)
histPlot(data_train['log10_Horizontal_Distance_To_Roadways'], 'green')
f.add_subplot(332)
histPlot(data_train['Sqr_Horizontal_Distance_To_Roadways'], 'c')
```



We achieved the best result again using square root of the Horizontal Distance to Roadways. Similarly as before, we remove failed experiments

```
In [67]: data_train.drop(['log_Horizontal_Distance_To_Roadways', 'log10_Horizontal_Distance_To_Roadways'],
```

5.2.6 Slope

Square root and logarithm Transformation

```
In [68]: print('\033[95m' + "Skew before transformation\n", data_train['Slope'].skew(),
        "\nmin\n", data_train['Slope'].min(),
        "\nmax\n", data_train['Slope'].max(),)
```

```
Skew before transformation
0.5236583382936286
min
0
max
52
```

Results after logarithm and Square root Transformation

```
In [69]: # since we have null values we add plus 1 to avoid log of zero
data_train['logSlope'] = np.log(data_train['Slope']+1)
```

```
In [70]: print('\033[92m' + "Skew after log transformation\n", data_train['logSlope'].skew(),
        "\nmin\n", data_train['logSlope'].min(),
        "\nmax\n", data_train['logSlope'].max(),)
```

```
Skew after log transformation
-0.7530340055684758
min
0.0
max
3.970291913552122
```

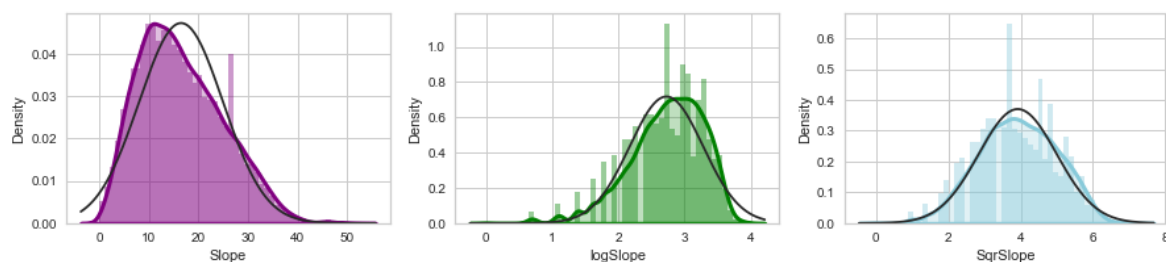
```
In [71]: data_train['SqrSlope'] = data_train['Slope']**0.5
```

```
In [72]: print('\033[96m' + "Skew after Square Root transformation\n", data_train['SqrSlope'].skew(),
        "\nmin\n", data_train['SqrSlope'].min(),
        "\nmax\n", data_train['SqrSlope'].max(),)
```

```
Skew after Square Root transformation
-0.09875824995364099
min
0.0
max
7.211102550927978
```

```
In [73]: def histPlot(first_feature,col):
        sns.distplot(first_feature,color=col,fit = norm,kde = True,kde_kws = {'shade': True, 'linewidth

f = plt.figure(figsize=(15,10))
f.add_subplot(331)
histPlot(data_train['Slope'], 'purple')
f.add_subplot(332)
histPlot(data_train['logSlope'], 'green')
f.add_subplot(333)
histPlot(data_train['SqrSlope'], 'c')
```



Since the skewness for the slope shows better performance when using the square root, we will transform the variable into square root as well

variable into square root as well.

```
In [74]: data_train.drop(['logSlope'], axis=1,inplace=True)
```

5.2.7 Horizontal Distance To Fire Points

Transformation

```
In [75]: print('\033[95m"+"Skew before transformation\n", data_train['Horizontal_Distance_To_Fire_Points']
        "\nmin\n", data_train['Horizontal_Distance_To_Fire_Points'].min(),
        "\nmax\n", data_train['Horizontal_Distance_To_Fire_Points'].max(),)
```

```
Skew before transformation
1.6170988738848289
min
0
max
6993
```

Results after logarithm and Sqrare root Transformation

```
In [76]: # since we have null values we add plus 1 to avoid log of zero
data_train['log_Horizontal_Distance_To_firepoints'] = np.log(data_train['Horizontal_Distance_To_F
```

```
In [77]: print('\033[92m"+"Skew after log transformation\n", data_train['log_Horizontal_Distance_To_firepo
        "\nmin\n", data_train['log_Horizontal_Distance_To_firepoints'].min(),
        "\nmax\n", data_train['log_Horizontal_Distance_To_firepoints'].max(),)
```

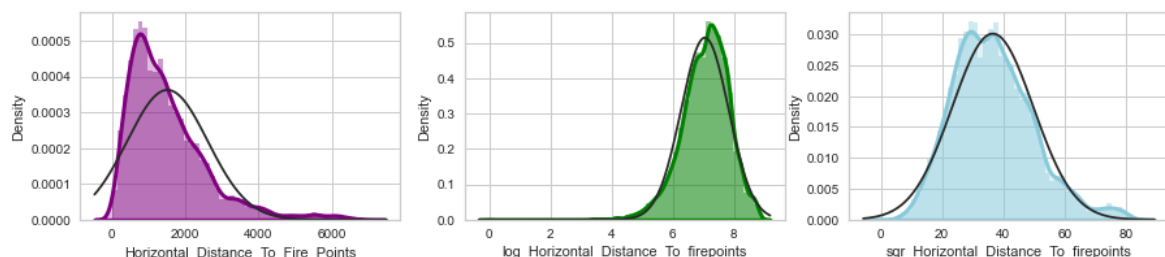
```
Skew after log transformation
-0.6708653924261154
min
0.0
max
8.852807917623322
```

```
In [78]: #Transform with square root
data_train['sqr_Horizontal_Distance_To_firepoints'] = data_train['Horizontal_Distance_To_Fire_Poi
```

```
In [79]: print('\033[96m"+"Skew after Square Root transformation\n", data_train['sqr_Horizontal_Distance_T
        "\nmin\n", data_train['sqr_Horizontal_Distance_To_firepoints'].min(),
        "\nmax\n", data_train['sqr_Horizontal_Distance_To_firepoints'].max(),)
```

```
Skew after Square Root transformation
0.5839887898581113
min
0.0
max
83.62415918859813
```

```
In [80]: def histPlot(first_feature,col):
        sns.distplot(first_feature,color=col,fit = norm,kde = True,kde_kws = {'shade': True, 'linewidth
        f = plt.figure(figsize=(15,10))
        f.add_subplot(331)
        histPlot(data_train['Horizontal_Distance_To_Fire_Points'], 'purple')
        f.add_subplot(332)
        histPlot(data_train['log_Horizontal_Distance_To_firepoints'], 'green')
        f.add_subplot(333)
        histPlot(data_train['sqr_Horizontal_Distance_To_firepoints'], 'c')
```



Since square root transformation gives the best result in skewness, we will also use sqr for the feature variable.

```
In [81]: data_train.drop(['log_Horizontal_Distance_To_firepoints'], axis=1,inplace=True)
```

5.2.8 Hillshades

5.2.8.1 Mean Hillshade

Creation of new Feature: Mean Hillshade

```
In [82]: # We take the average of Hillshades, which gives you the average light exposure of each cover type
data_train['Mean_Hillshade'] = (data_train['Hillshade_9am'] + data_train['Hillshade_Noon'] + data_train['Hillshade_3pm'])
```

```
In [83]: # Intensity of the Hillshade variables in 3 bin sizes with the bin discretizer
est = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')
data_train['Mean_Hillshade_bin'] = est.fit_transform(data_train[['Mean_Hillshade']])
```

```
In [84]: data_train[['Mean_Hillshade_bin', 'Mean_Hillshade']].describe()
```

```
Out[84]:
```

	Mean_Hillshade_bin	Mean_Hillshade
count	15120.000000	15120.000000
mean	1.768122	188.920635
std	0.460708	17.125035
min	0.000000	105.666667
25%	2.000000	180.333333
50%	2.000000	192.333333
75%	2.000000	201.333333
max	2.000000	213.666667

```
In [85]: print('\033[95m' + "Skew before transformation\n", data_train['Mean_Hillshade'].skew(),
          "\nmin\n", data_train['Mean_Hillshade'].min(),
          "\nmax\n", data_train['Mean_Hillshade'].max(),)
```

```
Skew before transformation
-1.0903510485555057
min
105.66666666666667
max
213.66666666666666
```

Results after logarithm Transformation, Square root Transformation and BoxCox Transformation

```
In [86]: data_train['log_Mean_Hillshade'] = np.log(data_train['Mean_Hillshade'])
```

```
In [87]: print('\033[92m' + "Skew after log transformation\n", data_train['log_Mean_Hillshade'].skew(),
          "\nmin\n", data_train['log_Mean_Hillshade'].min(),
          "\nmax\n", data_train['log_Mean_Hillshade'].max(),)
```

```
Skew after log transformation
-1.4322026123219795
min
4.660289485209171
max
5.36441716825256
```

```
In [88]: data_train['log10Mean_Hillshade'] = np.log10(data_train['Mean_Hillshade'])
```

```
In [89]: print('\033[92m' + "Skew after log10 transformation\n", data_train['log10Mean_Hillshade'].skew(),
          "\nmin\n", data_train['log10Mean_Hillshade'].min(),
          "\nmax\n", data_train['log10Mean_Hillshade'].max(),)
```

```
Skew after log10 transformation
-1.4322026123219602
min
2.023938007498089
max
2.329736774799155
```

```
In [90]: data_train['sqr_Mean_Hillshade'] = data_train['Mean_Hillshade']**0.5
```

```
In [91]: print('\033[96m'"Skew after Square Root transformation\n", data_train['sqr_Mean_Hillshade'].skew()
        "\nmin\n", data_train['sqr_Mean_Hillshade'].min(),
        "\nmax\n", data_train['sqr_Mean_Hillshade'].max(),)
```

```
Skew after Square Root transformation
-1.25173398314383
min
10.279429296739517
max
14.617341299520465
```

```
In [92]: #Now, the Box-Cox transformation also requires our data to only contain positive numbers
        # transform training data with Boxcox
        data_train['Mean_Hillshade_boxcox'], _ = stats.boxcox(data_train['Mean_Hillshade'])
```

```
In [93]: print('\033[93m'"Skew after Boxcox transformation\n", data_train['Mean_Hillshade_boxcox'].skew()
        "\nmin\n", data_train['Mean_Hillshade_boxcox'].min(),
        "\nmax\n", data_train['Mean_Hillshade_boxcox'].max(),)
```

```
Skew after Boxcox transformation
-0.14515719864079077
min
9274946583.191551
max
382374055990.05743
```

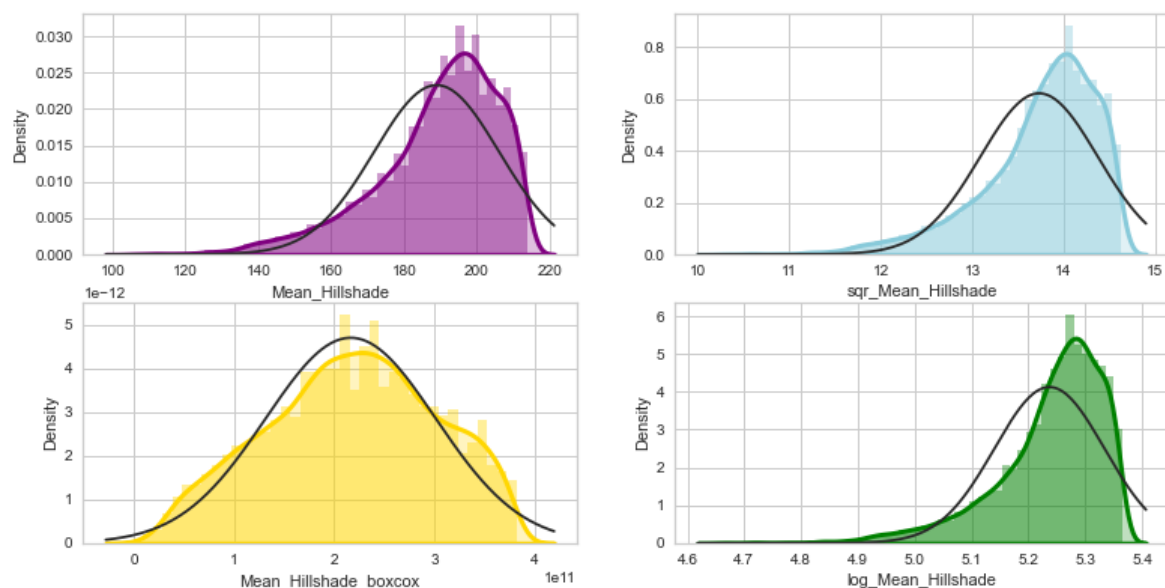
```
In [94]: stats.normaltest(data_train['Mean_Hillshade_boxcox'])
```

```
Out[94]: NormaltestResult(statistic=961.0422164343719, pvalue=2.052741503609043e-209)
```

```
In [95]: from scipy.stats import norm
        import scipy.stats as stats

        def histPlot(first_feature,col):
            sns.distplot(first_feature,color=col,fit = norm,kde = True,kde_kws = {'shade': True, 'linewidth': 2})

        f = plt.figure(figsize=(20,10))
        f.add_subplot(331)
        histPlot(data_train['Mean_Hillshade'], 'purple')
        f.add_subplot(335)
        histPlot(data_train['log_Mean_Hillshade'], 'green')
        f.add_subplot(334)
        histPlot(data_train['Mean_Hillshade_boxcox'], 'gold')
        f.add_subplot(332)
        histPlot(data_train['sqr_Mean_Hillshade'], 'c')
```



The distribution did not improve with Square Root and Logarithms Transformation. Hence we use BoxCox which improved the distribution substantially.

```
In [96]: data_train.drop(['log10Mean_Hillshade', 'log_Mean_Hillshade', 'sqr_Mean_Hillshade'], axis=1, inplace
```

5.2.8.2 Hillshade 9am

Transformation

```
In [97]: print('\033[95m'+ "Skew before transformation\n", data_train['Hillshade_9am'].skew(),
          "\nmin\n", data_train['Hillshade_9am'].min(),
          "\nmax\n", data_train['Hillshade_9am'].max(),)
```

```
Skew before transformation
-1.0936805605383073
min
0
max
254
```

Results after logarithm Transformation , Square root Transformation and BoxCox Transformation

```
In [98]: data_train['log_Hillshade_9am'] = np.log(data_train['Hillshade_9am']+1)
```

```
In [99]: print('\033[92m'+ "Skew after log transformation\n", data_train['log_Hillshade_9am'].skew(),
          "\nmin\n", data_train['log_Hillshade_9am'].min(),
          "\nmax\n", data_train['log_Hillshade_9am'].max(),)
```

```
Skew after log transformation
-3.708953816677303
min
0.0
max
5.541263545158426
```

```
In [100]: data_train['sqr_Hillshade_9am'] = data_train['Hillshade_9am']**0.5
```

```
In [101]: print('\033[96m'+ "Skew after Square Root transformation\n", data_train['sqr_Hillshade_9am'].skew(),
          "\nmin\n", data_train['sqr_Hillshade_9am'].min(),
          "\nmax\n", data_train['sqr_Hillshade_9am'].max(),)
```

```
Skew after Square Root transformation
-1.4741589700243982
min
0.0
max
15.937377450509228
```

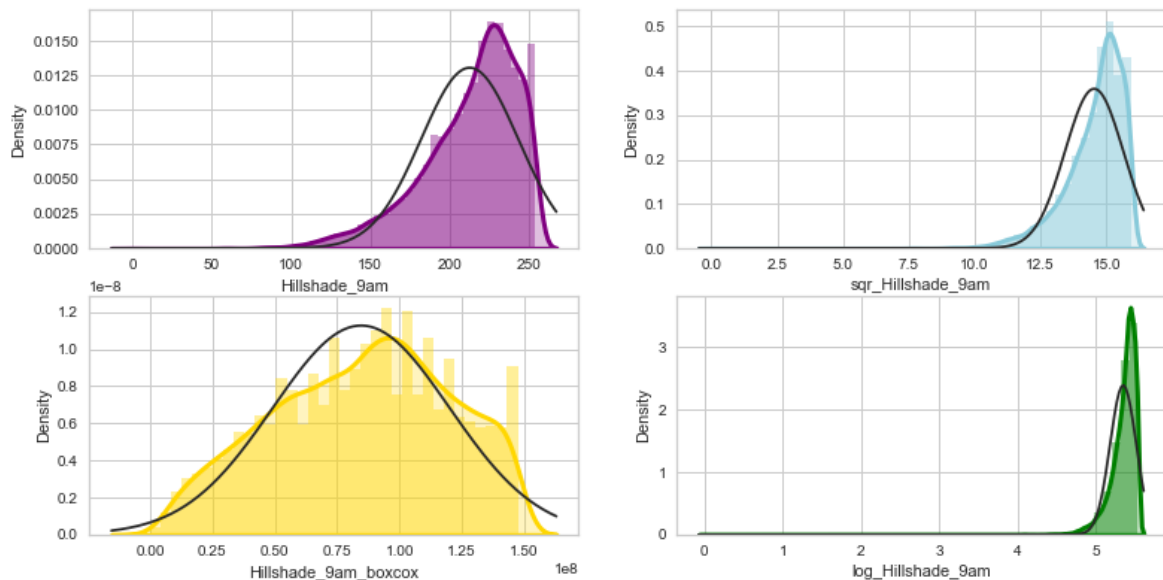
```
In [102]: #Now, the Box-Cox transformation also requires our data to only contain positive numbers, transfo
data_train['Hillshade_9am_boxcox'], lam = stats.boxcox(data_train['Hillshade_9am']+1)
#lam is the best lambda for the distribution
```

```
In [103]: print('\033[93m'+ "Skew after Boxcox transformation\n", data_train['Hillshade_9am_boxcox'].skew(),
          "\nmin\n", data_train['Hillshade_9am_boxcox'].min(),
          "\nmax\n", data_train['Hillshade_9am_boxcox'].max(),)
```

```
Skew after Boxcox transformation
-0.1964508631897843
min
0.0
max
147303558.7348688
```

```
In [104]: def histPlot(first_feature,col):
            sns.distplot(first_feature,color=col,fit = norm,kde = True,kde_kws = {'shade': True, 'linewidth

f = plt.figure(figsize=(20,10))
f.add_subplot(331)
histPlot(data_train['Hillshade_9am'], 'purple')
f.add_subplot(335)
histPlot(data_train['log_Hillshade_9am'], 'green')
f.add_subplot(334)
histPlot(data_train['Hillshade_9am_boxcox'], 'gold')
f.add_subplot(332)
histPlot(data_train['sqr_Hillshade_9am'], 'c')
```



BoxCox outperforms the other two for the Hillshade 9am

```
In [105... data_train.drop(['log_Hillshade_9am', 'sqr_Hillshade_9am'], axis=1, inplace=True)
```

5.2.8.3 Hillshade Noon

Transformation

```
In [106... print('\033[95m'+"Skew before transformation\n", data_train['Hillshade_Noon'].skew(),
        "\nmin\n", data_train['Hillshade_Noon'].min(),
        "\nmax\n", data_train['Hillshade_Noon'].max(),)
```

```
Skew before transformation
-0.9532317074981783
min
99
max
254
```

Results after logarithm Transformation , Square root Transformation and BoxCox Transformation

```
In [107... data_train['log_Hillshade_Noon'] = np.log(data_train['Hillshade_Noon']+1)
```

```
In [108... print('\033[92m'+"Skew after log transformation\n", data_train['log_Hillshade_Noon'].skew(),
        "\nmin\n", data_train['log_Hillshade_Noon'].min(),
        "\nmax\n", data_train['log_Hillshade_Noon'].max(),)
```

```
Skew after log transformation
-1.4151877543194478
min
4.605170185988092
max
5.541263545158426
```

```
In [109... data_train['sqr_Hillshade_Noon'] = data_train['Hillshade_Noon']**0.5
```

```
In [110... print('\033[96m'+"Skew after Square Root transformation\n", data_train['sqr_Hillshade_Noon'].skew(),
        "\nmin\n", data_train['sqr_Hillshade_Noon'].min(),
        "\nmax\n", data_train['sqr_Hillshade_Noon'].max(),)
```

```
Skew after Square Root transformation
-1.1654338788107177
min
9.9498743710662
max
15.937377450509228
```

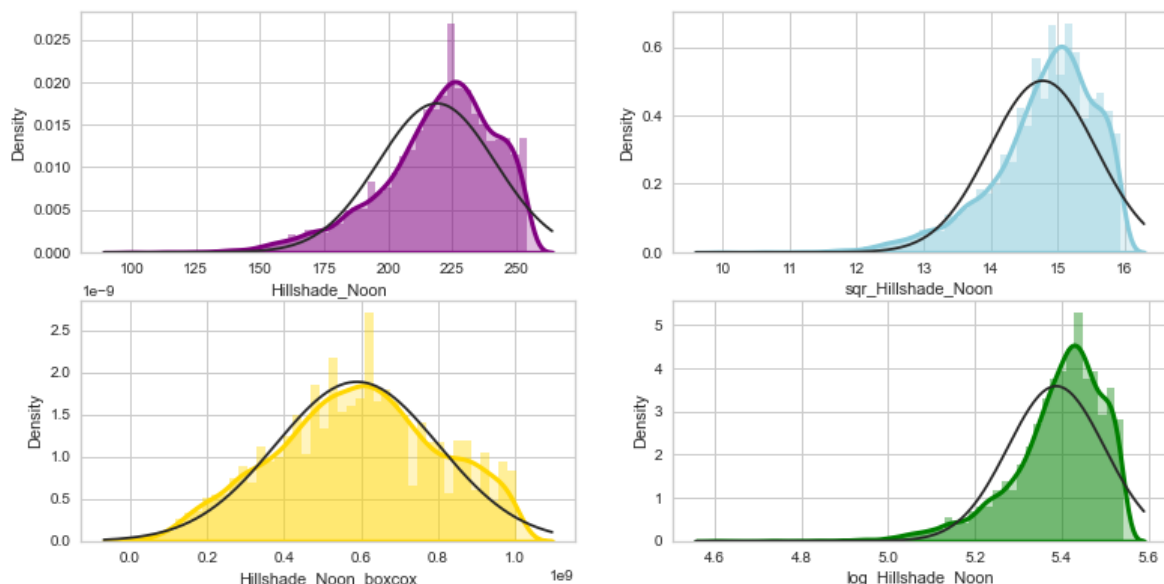
```
In [111... #Now, the Box-Cox transformation also requires our data to only contain positive numbers, transfo
data_train['Hillshade_Noon_boxcox'], lam = stats.boxcox(data_train['Hillshade_Noon'])
#lam is the best lambda for the distribution

In [112... print('\033[93m'"Skew after Boxcox transformation\n", data_train['Hillshade_Noon_boxcox'].skew()
"\nmin\n", data_train['Hillshade_Noon_boxcox'].min(),
"\nmax\n", data_train['Hillshade_Noon_boxcox'].max(),)

Skew after Boxcox transformation
-0.10142061186267355
min
23334179.80223645
max
1004797392.9966723
```

```
In [113... from scipy.stats import norm
import scipy.stats as stats

def histPlot(first_feature,col):
    sns.distplot(first_feature,color=col,fit = norm,kde = True,kde_kws = {'shade': True, 'linewidth
f = plt.figure(figsize=(20,10))
f.add_subplot(331)
histPlot(data_train['Hillshade_Noon'], 'purple')
f.add_subplot(335)
histPlot(data_train['log_Hillshade_Noon'], 'green')
f.add_subplot(334)
histPlot(data_train['Hillshade_Noon_boxcox'], 'gold')
f.add_subplot(332)
histPlot(data_train['sqr_Hillshade_Noon'], 'c')
```



Box Coc is outperforming the other transformations for Hillshade Noon

```
In [114... data_train.drop(['log_Hillshade_Noon', 'sqr_Hillshade_Noon'], axis=1,inplace=True)
```

5.2.8.4 Hillshade 3pm

Transformation

```
In [115... print('\033[95m'"Skew before transformation\n", data_train['Hillshade_3pm'].skew(),
"\nmin\n", data_train['Hillshade_3pm'].min(),
"\nmax\n", data_train['Hillshade_3pm'].max(),)

Skew before transformation
-0.34082723258478564
-:-
```

```

min
0
max
248

```

Results after logarithm Transformation , Square root Transformation and BoxCox Transformation

```
In [116... data_train['log_Hillshade_3pm'] = np.log(data_train['Hillshade_3pm']+1)
```

```
In [117... print('\033[92m"+"Skew after log transformation\n", data_train['log_Hillshade_3pm'].skew(),
        "\nmin\n", data_train['log_Hillshade_3pm'].min(),
        "\nmax\n", data_train['log_Hillshade_3pm'].max(),)
```

```

Skew after log transformation
-4.40025450334946
min
0.0
max
5.517452896464707

```

```
In [118... data_train['sqr_Hillshade_3pm'] = data_train['Hillshade_3pm']**0.5
```

```
In [119... print('\033[96m"+"Skew after Square Root transformation\n", data_train['sqr_Hillshade_3pm'].skew(
        "\nmin\n", data_train['sqr_Hillshade_3pm'].min(),
        "\nmax\n", data_train['sqr_Hillshade_3pm'].max(),)
```

```

Skew after Square Root transformation
-1.338669485611021
min
0.0
max
15.748015748023622

```

```
In [120... #Now, the Box-Cox transformation also requires our data to only contain positive numbers, transfo
data_train['Hillshade_3pm_boxcox'], lam = stats.boxcox(data_train['Hillshade_3pm']+1)
#lam is the best lambda for the distribution
```

```
In [121... print('\033[93m"+"Skew after Boxcox transformation\n", data_train['Hillshade_3pm_boxcox'].skew(),
        "\nmin\n", data_train['Hillshade_3pm_boxcox'].min(),
        "\nmax\n", data_train['Hillshade_3pm_boxcox'].max(),)
```

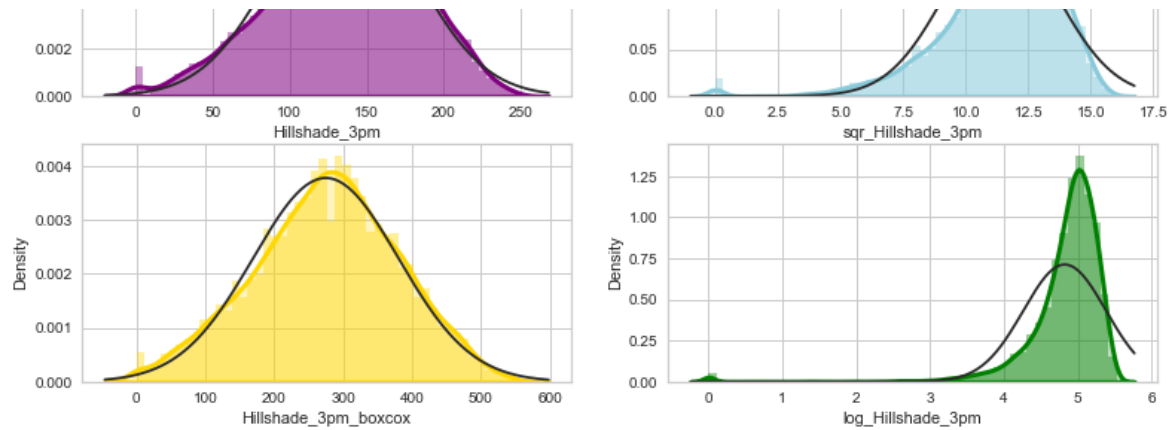
```

Skew after Boxcox transformation
-0.15018383894899204
min
0.0
max
552.0141106428457

```

```
In [122... def histPlot(first_feature,col):
    sns.distplot(first_feature,color=col,fit = norm,kde = True,kde_kws = {'shade': True, 'linewidth
f = plt.figure(figsize=(20,10))
f.add_subplot(331)
histPlot(data_train['Hillshade_3pm'], 'purple')
f.add_subplot(335)
histPlot(data_train['log_Hillshade_3pm'], 'green')
f.add_subplot(334)
histPlot(data_train['Hillshade_3pm_boxcox'], 'gold')
f.add_subplot(332)
histPlot(data_train['sqr_Hillshade_3pm'], 'c')
```





For the Hillshade 3pm the data was not highly skewed, we either keep the original or we can use boxcox as it improved the variables as well.

```
In [123... data_train.drop(['log_Hillshade_3pm', 'sqr_Hillshade_3pm'], axis=1, inplace=True)
```

5.2.8.5 Hillshades Ratios

```
In [124... data_train['ratio_Hillshade_3pm'] = data_train['Hillshade_3pm']/255
data_train['ratio_Hillshade_Noon'] = data_train['Hillshade_Noon']/255
data_train['ratio_Hillshade_9am'] = data_train['Hillshade_9am']/255
```

5.2.8.6 Hillshades All Day

This variable calculates the the point where there is a lot of sun and where there is less sun.

```
In [125... data_train['Hillshade_9am_bin'] = data_train['Hillshade_9am'] > 150
data_train['Hillshade_Noon_bin'] = data_train['Hillshade_Noon'] > 150
data_train['Hillshade_3pm_bin'] = data_train['Hillshade_3pm'] > 150
```

```
In [126... data_train['Hillshade_9am_bin'] = data_train['Hillshade_9am_bin'].astype(int)
data_train['Hillshade_Noon_bin'] = data_train['Hillshade_Noon_bin'].astype(int)
data_train['Hillshade_3pm_bin'] = data_train['Hillshade_3pm_bin'].astype(int)
```

```
In [127... data_train['Hillshade_allday_bin'] = data_train['Hillshade_9am_bin'] * data_train['Hillshade_Noon']
```

This variable calculates the point where there is a lot of sun and where there is less sun.

```
In [128... data_train['Hillshade_allday_bin'].value_counts()
```

```
Out[128]: 0    9996
          1    5124
          Name: Hillshade_allday_bin, dtype: int64
```

5.2.9 Aspect

Transformation

```
In [129... print('\033[95m'+"Skew before transformation\n", data_train['Aspect'].skew(),
      "\nmin\n", data_train['Aspect'].min(),
      "\nmax\n", data_train['Aspect'].max(),)
```

```
Skew before transformation
0.4509352940316223
min
0
max
360
```

Results after logarithm Transformation and Square root Transformation

```
In [130...] data_train['sqr_Aspect'] = data_train['Aspect']**0.5

In [131...] print('\033[96m'"Skew after Square Root transformation\n", data_train['sqr_Aspect'].skew(),
                "\nmin\n", data_train['sqr_Aspect'].min(),
                "\nmax\n", data_train['sqr_Aspect'].max(),)

Skew after Square Root transformation
-0.12653305947211343
min
0.0
max
18.973665961010276

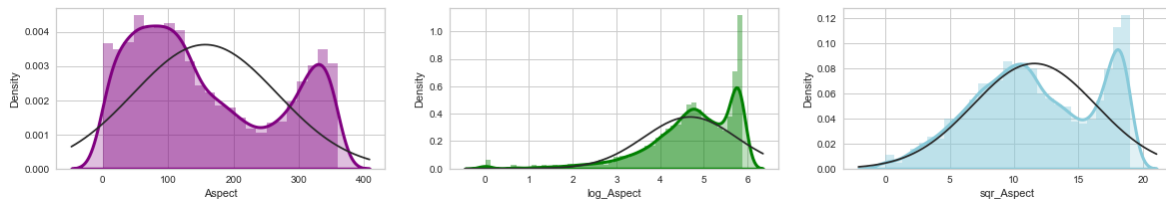
In [132...] data_train['log_Aspect'] = np.log(data_train['Aspect']+1)

In [133...] print('\033[92m'"Skew after log transformation\n", data_train['log_Aspect'].skew(),
                "\nmin\n", data_train['log_Aspect'].min(),
                "\nmax\n", data_train['log_Aspect'].max(),)

Skew after log transformation
-1.4584406899303897
min
0.0
max
5.8888779583328805

In [134...] def histPlot(first_feature,col):
    sns.distplot(first_feature,color=col,fit = norm,kde = True,kde_kws = {'shade': True, 'linewidth

f = plt.figure(figsize=(20,10))
f.add_subplot(331)
histPlot(data_train['Aspect'], 'purple')
f.add_subplot(332)
histPlot(data_train['log_Aspect'], 'green')
#f.add_subplot(334)
#histPlot(data_train['Hillshade_3pm_boxcox'], 'gold')
f.add_subplot(333)
histPlot(data_train['sqr_Aspect'], 'c')
```



For aspect square root turned out to be the best transformation in terms of skeweness.

Overall, the best transformations done here are square rt, and boxcox. log transformations did not proved to be beneficial

```
In [135...] data_train.drop(['log_Aspect'], axis=1,inplace=True)
```

In here we are transforming the ratios into a unit scale by dividing by its index. We do so because we think it is much easier to understand

```
In [136...] data_train['ratio_Hillshade_3pm'] = data_train['Hillshade_3pm']/255
data_train['ratio_Hillshade_Noon'] = data_train['Hillshade_Noon']/255
data_train['ratio_Hillshade_9am'] = data_train['Hillshade_9am']/255
```

5.2.9.2 Aspect in degrees

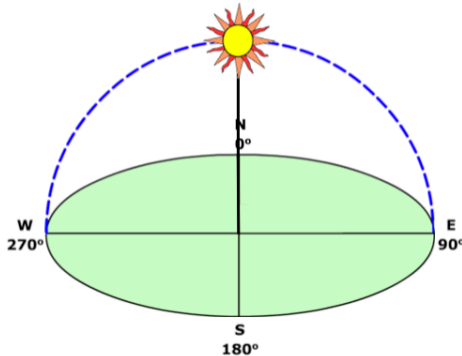
New Features

The azimuth is the angular direction of the sun, measured from north clockwise in degrees from 0 to 360. An Azimuth of 90 degrees is east The Out of values will be between for instance the middle of north and east

Azimuth of 90 degrees is east. The cut of values will be between for instance the middle of north and east.

We make a transformation so to get dummies for Aspect ordinal values: north, south, east and west

- Aspect_North: from 315 deg to 45 deg
- Aspect_East: from 45 deg to 135 deg
- Aspect_South: from 135 deg to 225 deg
- Aspect_West: from 225 deg to 315 deg



Source: <https://www.pveducation.org/pvcdrom/properties-of-sunlight/azimuth-angle>

```
In [137... #Grouping Aspect in the four directions
data_train['Aspect_North'] = np.where(((data_train['Aspect'] >= 0) & (data_train['Aspect'] < 45)) | ((X
data_train['Aspect_East'] = np.where((data_train['Aspect'] >= 45) & (data_train['Aspect'] < 135), 1, 0
data_train['Aspect_South'] = np.where((data_train['Aspect'] >= 135) & (data_train['Aspect'] < 225), 1
data_train['Aspect_West'] = np.where((data_train['Aspect'] >= 225) & (data_train['Aspect'] < 315), 1,
```

5.2.10 Elevation & Binned Elevation

No transformation is done as it is already very symmetric distributed

```
In [138... print('\033[95m'+ "Skew before transformation\n", data_train['Elevation'].skew(),
        "\nmin\n", data_train['Elevation'].min(),
        "\nmax\n", data_train['Elevation'].max(),)
```

```
Skew before transformation
0.07563970693591461
min
1863
max
3849
```

```
In [139... data_train['binned_elevation'] = [math.floor(v/50.0) for v in data_train['Elevation']]
```

5.2.11 Addition and subtraction of same sale units

We are making more features by summing and substracing different combinations similar in terms of units

Addition and Substraction on the same scale Using for loop was giving us a bad performance hence we are using the features on the same scale which to add or subtract

```
data_train['Road+Fire'] = data_train['Horizontal_Distance_To_Roadways'] + data_train['Horizontal_Distance_To_Fire_Points']
data_train['Road-Fire'] = abs(data_train['Horizontal_Distance_To_Roadways'] - data_train['Horizontal_Distance_To_Fire_Points'])
data_train['Road+Hydro'] = data_train['Horizontal_Distance_To_Roadways'] + data_train['Horizontal_Distance_To_Hydrology']
data_train['Road-Hydro'] = abs(data_train['Horizontal_Distance_To_Roadways'] - data_train['Horizontal_Distance_To_Hydrology'])
data_train['Hydro+Fire'] = data_train['Horizontal_Distance_To_Hydrology'] + data_train['Horizontal_Distance_To_Fire_Points']
data_train['Hydro-Fire'] = abs(data_train['Horizontal_Distance_To_Hydrology'] - data_train['Horizontal_Distance_To_Fire_Points'])
data_train['Road+Fire+Hydro'] = data_train['Horizontal_Distance_To_Roadways'] +
data_train['Horizontal_Distance_To_Fire_Points'] + data_train['Horizontal_Distance_To_Hydrology']
data_train['Ele+Road+Fire+Hydro'] = data_train['Elevation'] + data_train['Horizontal_Distance_To_Roadways'] +
data_train['Horizontal_Distance_To_Fire_Points'] + data_train['Horizontal_Distance_To_Hydrology'] data_train['Ele+road'] =
data_train['Elevation'] + data_train['Horizontal_Distance_To_Roadways'] data_train['Ele-road'] = abs(data_train['Elevation'] -
data_train['Horizontal_Distance_To_Roadways']) data_train['Ele+fire'] = data_train['Elevation'] +
data_train['Horizontal_Distance_To_Fire_Points'] data_train['Ele-fire'] = abs(data_train['Elevation'] -
```

```
data_train['Horizontal_Distance_To_Fire_Points']) data_train['Ele+hydro'] = data_train['Elevation'] +
data_train['Horizontal_Distance_To_Hydrology'] data_train['Ele-hydro'] = abs(data_train['Elevation'] -
data_train['Horizontal_Distance_To_Hydrology'])
```

```
In [140... data_train['sqr_Road+Fire'] = (data_train['Horizontal_Distance_To_Roadways'] + data_train['Horizo
data_train['sqr_Road-Fire'] = (abs(data_train['Horizontal_Distance_To_Roadways'] - data_train['Ho
data_train['sqr_Road+Hydro'] = (data_train['Horizontal_Distance_To_Roadways'] + data_train['Horiz
data_train['sqr_Road-Hydro'] = (abs(data_train['Horizontal_Distance_To_Roadways'] - data_train['H
data_train['sqr_Hydro+Fire'] = (data_train['Horizontal_Distance_To_Hydrology'] + data_train['Hori
data_train['sqr_Hydro-Fire'] = (abs(data_train['Horizontal_Distance_To_Hydrology'] - data_train['

data_train['sqr_Road+Fire+Hydro'] = (data_train['Horizontal_Distance_To_Roadways'] + data_train[

data_train['sqr_Ele+Road+Fire+Hydro'] = (data_train['Elevation'] + data_train['Horizontal_Distance

data_train['sqr_Ele+road'] = (data_train['Elevation'] + data_train['Horizontal_Distance_To_Roadwa
data_train['sqr_Ele-road'] = (abs(data_train['Elevation'] - data_train['Horizontal_Distance_To_Ro
data_train['sqr_Ele+fire'] = (data_train['Elevation'] + data_train['Horizontal_Distance_To_Fire_P
data_train['sqr_Ele-fire'] = (abs(data_train['Elevation'] - data_train['Horizontal_Distance_To_Fi
data_train['sqr_Ele+hydro'] = (data_train['Elevation'] + data_train['Horizontal_Distance_To_Hydro
data_train['sqr_Ele-hydro'] = (abs(data_train['Elevation'] - data_train['Horizontal_Distance_To_H
```

5.2.12 Geoclimate grouping

5.2.12 Climatic feature engineering to group soils

In the Kaggle competition, there is a reference to John A. Blackard which happened to be one geologist working

for the forest federal US agency. In a co-authored paper, he gives further insights on the soil families with a list of codes. These are digits categorizing the soils according to climate and geology. We decide to take this valuable insight and engineer features around this dynamic so to cut down the number of soils

From original database donated by John A. Blackard

Code Designations:

Wilderness Areas:

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

Soil Types: 1 to 40 : based on the USFS Ecological Landtype Units (ELUs) for this study area:

Study Code 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 - Borochemists 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.

Survey	Note:	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
		7. subalpine	7. igneous and metamorphic
		8. alpine	8. volcanic

The USFD, an American federal agency for forest service dependent on the department of agriculture has classified soil types according to **climatic zone (first digit)** and **geology (second digit)**. Because of this, we believe a similar classification can be artificially engineered grouping all similar soils in 7 categories for climate (there is no lower montane dry soils) and 4 for geology (we do not take into consideration shale, sandstone, volcanic or unspecified)

5.2.12.1 Climatic Zone feature engineering to group soils

```
In [141...] data_train["Lower_Montane_Climate"] = data_train.loc[:,data_train.columns.str.contains("^Soil_Typ
In [142...] data_train['Montane_Dry_Climate'] =data_train.loc[:,data_train.columns.str.contains("^Soil_Type[7
In [143...] data_train['Montane_Climate'] =data_train.loc[:,data_train.columns.str.contains("^Soil_Type[1][01
In [144...] data_train['Montane_Dry_and_Montane_Climate'] =data_train.loc[:,data_train.columns.str.contains("
In [145...] data_train['Montante_and_Subalpine_Climate'] =data_train.loc[:,data_train.columns.str.contains("^
In [146...] data_train['Subalpine_Climate'] =data_train.loc[:,data_train.columns.str.contains("^Soil_Type19$|
In [147...] data_train['Alpine_Climate'] =data_train.loc[:,data_train.columns.str.contains("^Soil_Type[3][567
```

5.2.12.2 Geological feature engineering to group soils

Note:	First digit: climatic zone	Second digit: geologic zones
	1. lower montane dry	1. alluvium

ELU Survey	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
	7. subalpine	7. igneous and metamorphic
	8. alpine	8. volcanic

```
In [148... data_train['Alluvium_Soil'] = data_train.loc[:,data_train.columns.str.contains("^Soil_Type[1][456
```

```
In [149... data_train['Glacial_Soil'] =data_train.loc[:,data_train.columns.str.contains("^Soil_Type[9]$|^Soi
```

```
In [150... data_train['Mixed_Sedimentary_Soil'] =data_train.loc[:,data_train.columns.str.contains("^Soil_Typ
```

```
In [151... data_train['Igneus_and_Metamorphic_Soil'] =data_train.loc[:,data_train.columns.str.contains("^Soi
```

```
In [152... data_train.head()
```

```
Out[152]:
```

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_
0	2596	51	3	258	0	
1	2590	56	2	212	-6	
2	2804	139	9	268	65	
3	2785	155	18	242	118	
4	2595	45	2	153	-1	

Based on the medium paper "Preprocessing: Why you should generate polynomial features first before standardizing" mention it is not good practice to standardize the variables before before PolynomialFeatures. This should be done after to not loss the signal of the variables.

```
In [153... # Identify and drop our target variable 'Cover_Type' from dataframe, isolating our independent va
X = data_train.drop('Cover_Type', axis = 1)

# Isolate our dependent variable as a feature
y = data_train['Cover_Type']
```

```
In [154... # Train Test Split (80/20 size), drop duplicates and missing values

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = .2, random_state = 37, strati

X_train.drop_duplicates(inplace = True)
X_train.dropna(inplace = True)
```

Advanced Factorization

The numerical values present a level of detail that may be much more fine-grained than we need. For instance, the soil level can be represented by different categories (soil family, complex or stony/rubberly). We aggregate the data up which can help to avoid overfitting when the data is more aggregate.

We played with these features but reached a final conclusion of not to considering any more families and complex type. But Stonyness, climate and geology. For purposes of the assignment we show the code as a RAWNBCONVERT

The only family grouping we do on soils are the ones above, on geological and climate grounds

5.2.13 Soil Features

5.2.13.1 Soil Type Family

Using RAWNBCONVERT to bin the soil variables to the four categories

using Discretization to bin the soil variable to the family type.

Cathedral

1 Cathedral family - Rock outcrop complex, extremely stony.

Ratake

2 Vanet - Ratake families complex, very stony.

4 Ratake family - Rock outcrop complex, rubbly.

Vanet

5 Vanet family - Rock outcrop complex complex, rubbly.

Wetmore

6 Vanet - Wetmore families - Rock outcrop complex, stony.

Gothic

7 Gothic family.

Limber

8 Supervisor - Limber families complex.

Troutville

9 Troutville family, very stony.

Legault

12 Legault family - Rock land complex, stony.

29 Como - Legault families complex, extremely stony.

Gateview

17 Gateview family - Cryaquolis complex.

Rogert

18 Rogert family, very stony.

Como

30 Como family - Rock land - Legault family complex, extremely stony.

Bross

36 Bross family - Rock land - Cryumbrepts complex, extremely stony.

Catamount

10 Bullwark - Catamount families - Rock outcrop complex, rubbly.

11 Bullwark - Catamount families - Rock land complex, rubbly.

13 Catamount family - Rock land - Bullwark family complex, rubbly.

26 Granile - Catamount families complex, very stony.

32 Catamount family - Rock outcrop - Leighcan family complex, extremely stony.

31 Leighcan - Catamount families complex, extremely stony.

33 Leighcan - Catamount families - Rock outcrop complex, extremely stony.

Leighcan

21 Typic Cryaquolls - Leighcan family, till substratum complex.

22 Leighcan family, till substratum, extremely bouldery.

23 Leighcan family, till substratum - Typic Cryaquolls complex.

24 Leighcan family, extremely stony.

25 Leighcan family, warm, extremely stony.

27 Leighcan family, warm - Rock outcrop complex, extremely stony.

28 Leighcan family - Rock outcrop complex, extremely stony.

Moran

38 Leighcan - Moran families - Cryaquolls complex, extremely stony.

39 Moran family - Cryorthents - Leighcan family complex, extremely stony.

40 Moran family - Cryorthents - Rock land complex, extremely stony.

Others

3 Haploborolis - Rock outcrop complex, rubbly.
15 unspecified in the USFS Soil and ELU Survey.

37 Rock outcrop - Cryumbrepts - Cryorthents complex, extremely stony.
34 Cryorthents - Rock land complex, extremely stony.
35 Cryumbrepts - Rock outcrop - Cryaquepts complex.
20 Typic Cryaquepts - Typic Cryaquolls complex.
14 Pachic Argiborolis - Aquolis complex.
16 Cryaquolis - Cryoborolis complex.
19 Typic Cryaquolis - Borohemists complex.

NOT USED

```
# Soil Type family_soil_types = { 'Family_Cathedral': ['Soil_Type1'], 'Family_Retake': ['Soil_Type2', 'Soil_Type4'], 'Family_Vanet':
['Soil_Type5'], 'Family_Wetmore': ['Soil_Type6'], 'Family_Gothic': ['Soil_Type7'], 'Family_Limber': ['Soil_Type8'], 'Family_Troutville_':
['Soil_Type9'], 'Family_Legault': ['Soil_Type12', 'Soil_Type29'], 'Family_Gateview': ['Soil_Type17'], 'Family_Rogert': ['Soil_Type18'],
'Family_Como': ['Soil_Type30'], 'Family_Bross': ['Soil_Type36'], 'Family_Catamount':
['Soil_Type10','Soil_Type11','Soil_Type13','Soil_Type26','Soil_Type32','Soil_Type31','Soil_Type33'], 'Family_Leighcan':
['Soil_Type21','Soil_Type22','Soil_Type23','Soil_Type24','Soil_Type25','Soil_Type27','Soil_Type28'], 'Family_Moran':
['Soil_Type38','Soil_Type39','Soil_Type40'], 'Family_Others':
['Soil_Type3','Soil_Type15','Soil_Type37','Soil_Type34','Soil_Type35','Soil_Type20','Soil_Type14','Soil_Type16','Soil_Type19'], } for
family in family_soil_types: data_train[family] = 0 soil_types = family_soil_types[family] for soil_type in soil_types:
data_train[family] += data_train[soil_type] data_train
```

We will group the soil types according to their family and according to the complex and stonyness

5.2.13.2 Soil Complex

Complex Group

Rock_outcrop_complex

1 Cathedral family - Rock outcrop complex, extremely stony.
2 Vanet - Ratake families complex, very stony.
3 Haploborolis - Rock outcrop complex, rubbly.
4 Ratake family - Rock outcrop complex, rubbly.
5 Vanet family - Rock outcrop complex complex, rubbly.
6 Vanet - Wetmore families - Rock outcrop complex, stony.
10 Bullwark - Catamount families - Rock outcrop complex, rubbly.
27 Leighcan family, warm - Rock outcrop complex, extremely stony.
28 Leighcan family - Rock outcrop complex, extremely stony.
33 Leighcan - Catamount families - Rock outcrop complex, extremely stony.

Ratake_families_complex

2 Vanet - Ratake families complex, very stony.

Limber families complex

8 Supervisor - Limber families complex.

rock land complex

11 Bullwark - Catamount families - Rock land complex, rubbly.
12 Legault family - Rock land complex, stony.
34 Cryorthents - Rock land complex, extremely stony.
40 Moran family - Cryorthents - Rock land complex, extremely stony.

Cryoborolis complex

16 Cryaquolis - Cryoborolis complex.
17 Gateview family - Cryaquolis complex.

Bullwark family complex

13 Catamount family - Rock land - Bullwark family complex, rubbly.

Aquolis complex

14 Pachic Argiborolis - Aquolis complex.

Borohemists complex

19 Typic Cryaquolis - Borohemists complex.

Cryaquolls complex

20 Typic Cryaquepts - Typic Cryaquolls complex.

23 Leighcan family, till substratum - Typic Cryaquolls complex.

38 Leighcan - Moran families - Cryaquolls complex, extremely stony.

till substratum complex

21 Typic Cryaquolls - Leighcan family, till substratum complex.

Catamount families complex

26 Granile - Catamount families complex, very stony.

1 Leighcan - Catamount families complex, extremely stony.

31 Leighcan - Catamount families complex, extremely stony.

Legault families complex

29 Como - Legault families complex, extremely stony.

30 Como family - Rock land - Legault family complex, extremely stony.

Leighcan family complex

32 Catamount family - Rock outcrop - Leighcan family complex, extremely stony.

39 Moran family - Cryorthents - Leighcan family complex, extremely stony.

Cryaquepts complex

35 Cryumbrepts - Rock outcrop - Cryaquepts complex.

Cryumbrepts complex

36 Bross family - Rock land - Cryumbrepts complex, extremely stony.

Cryorthents complex

37 Rock outcrop - Cryumbrepts - Cryorthents complex, extremely stony.

others

7 Gothic family.

9 Troutville family, very stony.

22 Leighcan family, till substratum, extremely bouldery.

24 Leighcan family, extremely stony.

25 Leighcan family, warm, extremely stony.

18 Rogert family, very stony.

15 unspecified in the USFS Soil and ELU Survey.

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

NOT USED

```
# Complex Type family_complex_types = { 'Rock_outcrop_complex':
['Soil_Type1','Soil_Type2','Soil_Type3','Soil_Type4','Soil_Type5','Soil_Type6','Soil_Type10','Soil_Type27','Soil_Type28','Soil_Type33'],
'Ratake_families_complex': ['Soil_Type2'], 'Limber_families_complex': ['Soil_Type8'], 'Rock_land_complex':
['Soil_Type11','Soil_Type12','Soil_Type34','Soil_Type40'], 'Cryoborolis_complex': ['Soil_Type16','Soil_Type17'],
'Bullwark_family_complex': ['Soil_Type13'], 'Aquolis_complex': ['Soil_Type14'], 'Borohemists_complex': ['Soil_Type19'],
'Cryaquolls_complex': ['Soil_Type20','Soil_Type23','Soil_Type38'], 'Till_substratum_complex': ['Soil_Type21'],
'Catamount_families_complex': ['Soil_Type26','Soil_Type1','Soil_Type31'], 'Legault_families_complex': ['Soil_Type39','Soil_Type30'],
'Leighcan_family_complex': ['Soil_Type32','Soil_Type39'], 'Cryaquepts_complex': ['Soil_Type35'], 'Cryumbrepts_complex':
['Soil_Type36'], 'Cryorthents_complex': ['Soil_Type37'], 'others_complex':
['Soil_Type7','Soil_Type9','Soil_Type22','Soil_Type24','Soil_Type25','Soil_Type18','Soil_Type15'], } for family in
```

```
family_complex_types: data_train[family] = 0 complex_types = family_complex_types[family] for complex_type in complex_types:
data_train[family] += data_train[complex_type] data_train
```

5.2.13.3 Stonyness**Stony**

1 Cathedral family - Rock outcrop complex. extremely stony.

2 Vanet - Ratake families complex, very stony.
 6 Vanet - Wetmore families - Rock outcrop complex, stony.
 9 Troutville family, very stony.
 12 Legault family - Rock land complex, stony.
 18 Rogert family, very stony.
 24 Leighcan family, extremely stony.
 25 Leighcan family, warm, extremely stony.
 26 Granile - Catamount families complex, very stony.
 27 Leighcan family, warm - Rock outcrop complex, extremely stony.
 28 Leighcan family - Rock outcrop complex, extremely stony.
 29 Como - Legault families complex, extremely stony.
 30 Como family - Rock land - Legault family complex, extremely stony.
 31 Leighcan - Catamount families complex, extremely stony.
 32 Catamount family - Rock outcrop - Leighcan family complex, extremely stony.
 33 Leighcan - Catamount families - Rock outcrop complex, extremely stony.
 34 Cryorthents - Rock land complex, extremely stony.
 36 Bross family - Rock land - Cryumbrepts complex, extremely stony.
 37 Rock outcrop - Cryumbrepts - Cryorthents complex, extremely stony.
 38 Leighcan - Moran families - Cryaquolls complex, extremely stony.
 39 Moran family - Cryorthents - Leighcan family complex, extremely stony.
 40 Moran family - Cryorthents - Rock land complex, extremely stony.

Rubbly

3 Haploborolis - Rock outcrop complex, rubbly.
 4 Ratake family - Rock outcrop complex, rubbly.
 5 Vanet family - Rock outcrop complex complex, rubbly.
 10 Bullwark - Catamount families - Rock outcrop complex, rubbly.
 11 Bullwark - Catamount families - Rock land complex, rubbly.
 13 Catamount family - Rock land - Bullwark family complex, rubbly.

others

7 Gothic family.
 8 Supervisor - Limber families complex.
 14 Pachic Argiborolis - Aquolis complex.
 15 unspecified in the USFS Soil and ELU Survey.
 16 Cryaquolis - Cryoborolis complex.
 17 Gateview family - Cryaquolis complex.
 19 Typic Cryaquolis - Borohemists complex.
 20 Typic Cryaquepts - Typic Cryaquolls complex.
 21 Typic Cryaquolls - Leighcan family, till substratum complex.
 22 Leighcan family, till substratum, extremely bouldery.
 23 Leighcan family, till substratum - Typic Cryaquolls complex.
 35 Cryumbrepts - Rock outcrop - Cryaquepts complex.

This has been included

```
# Soil Type family_types = { 'Type_Stony': ['Soil_Type1','Soil_Type2', 'Soil_Type6', 'Soil_Type9', 'Soil_Type12', 'Soil_Type18',
'Soil_Type24', 'Soil_Type25', 'Soil_Type26', 'Soil_Type27', 'Soil_Type28', 'Soil_Type29', 'Soil_Type30', 'Soil_Type31', 'Soil_Type32',
'Soil_Type33', 'Soil_Type34', 'Soil_Type36', 'Soil_Type37', 'Soil_Type38', 'Soil_Type39', 'Soil_Type40'], 'Type_Rubbly':
['Soil_Type3', 'Soil_Type4', 'Soil_Type5', 'Soil_Type10', 'Soil_Type11', 'Soil_Type13'], 'Type_Other': ['Soil_Type7','Soil_Type8',
'Soil_Type14', 'Soil_Type15', 'Soil_Type16', 'Soil_Type17', 'Soil_Type19', 'Soil_Type20', 'Soil_Type21', 'Soil_Type22', 'Soil_Type23',
'Soil_Type35']} for family in family_types: data_train[family] = 0 soil_types = family_types[family] for soil_type in soil_types:
data_train[family] += data_train[soil_type] data_train
```

Note: Soil type is a single variable which has been one-hot encoded presumably , so we will reverse engineer the soil type. We will eventually drop the original soil type columns which has the added effect of significantly reducing the total number of features.

```
In [155... # Original soil features
soil_features = [f'Soil_Type{i}' for i in range(1,41)]
```

```
In [156... # Drop original soil features
data_train.drop(columns = soil_features, inplace = True)
```

```
In [157... #test if elevation makes a difference to take out with the new interaction model improves
data_train = data_train.drop(['Elevation^2'], axis = 1)
data_train = data_train.drop(['Elevation'], axis = 1)
```

```
In [158... data_train
```

```
Out[158]:
```

	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways
--	--------	-------	----------------------------------	--------------------------------	---------------------------------

0	51	3	258	0
1	56	2	212	-6
2	139	9	268	65
3	155	18	242	118
4	45	2	153	-1
...
15115	243	23	258	7
15116	121	19	633	195
15117	134	25	365	117
15118	167	28	218	101
15119	197	34	319	78

15120 rows x 117 columns

Removing the original scaled variables did not improve nor worsen the model. Since it does not change much the score, we remove it as we have it double in the model with the scaled features.

```
In [159... data_train = data_train.drop(['Aspect', 'Slope', 'Horizontal_Distance_To_Roadways', 'Horizontal_Distance_To_Hydrology', 'Vertical_Distance_To_Hydrology'], axis = 1)
```

```
In [160... X_drop = data_train.drop(labels=["Cover_Type"], axis=1)
y_drop = data_train['Cover_Type']
```

```
In [161... X_train_drop, X_val_drop, y_train_drop, y_val_drop = train_test_split(X_drop, y_drop, random_state=37)
```

```
In [162... forest_dropped_variables = RandomForestClassifier(random_state=37)
model_dropped_variables = forest_dropped_variables.fit(X_train_drop, y_train_drop)
```

```
In [163... # calculating accuracy score
model_dropped_variables.score(X_val_drop, y_val_drop)
```

```
Out[163]: 0.8613756613756614
```

```
In [164... forest_dropped_variables = RandomForestClassifier(random_state=37)
print("Accuracy = {:.4f}".format(np.mean(cross_val_score(model_dropped_variables, X_val_drop, y_val_drop, cv=5))))
Accuracy = 0.7921
```

Model does not improve or worsen with respect to baseline's accuracy

5.10 Summary

Features	Transformation
ID	Drop
Distance To Hydrology	Square Root of the length of the side of horizontal and vertical
Horizontal Distance To Roadways	Square Root of horizontal Distance to Roadways
Slope	Square Root Slope
Horizontal_Distance To firepoints	Square Root Horizontal Distance to firepoints
Mean Hillshade	Box Cox Average of all Hillshades features

Hillshade 9am	Box Cox Hillshade 9am
Hillshade Noon	Box Cox Hillshade Noon
Hillshade 3pm	Box Cox Hillshade 3pm
Hillshade all day	Binned
Aspect	Square Root Aspect
Aspect North, East,South and West	Grouping Aspect
Geological Grouping	Grouping Soil Types
Climate Grouping	Grouping Soil Types
Soil Family	Grouping Soil Families
Soil Type Complex	Grouping Soil Complex
Soil Type Stonyness	Grouping by Soil stonyness

6.Feature Selection

We will try to use several feature selection algorithms where we use them in combination of all the different selection method and will take the best score of all the used common algorithms score.

Source: <https://towardsdatascience.com/the-5-feature-selection-algorithms-every-data-scientist-need-to-know-3a6b566efd2>

Locking all features in a csv

For the shake of efficiency, we create a csv file to reuse also later on in part III

```
In [165... data_train.shape
```

```
Out[165]: (15120, 108)
```

This code below have all features that we did in the feature engineering part and transport this to csv before we actually use the feature selection methods and select only few variables.

```
In [166... #Only X_Train replacement
data_train.to_csv('all_features_data_train.csv')
data_train.to_csv('all_features_data_train.csv') #this one is doubled
```

6.1. Standardization

We now proceed to do some standardization in the hopes it will help assist feature selection algorithms more efficiently.

We attempt also to run a few correlation matrices and take out heavily correlated values but this did not improved baselines' accuracy so it is marked as RAWNBCONVERT

NOT USED

```
#obtain the correlations of each features in dataset corrmatrix = data_train.corr() top_corr_features = corrmatrix.index plt.figure(figsize=(20,20)) #plot heat map g=sns.heatmap(data_train[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

We can barely see anything but I'd remove all green blocks (polynomial features)

NOT USED

```
#if run twice, error will appear as labels are already deleted! data_train =  
data_train.drop(["Horizontal_Distance_To_Hydrology_x_Vertical_Distance_To_Hydrology",  
"Horizontal_Distance_To_Hydrology_x_Horizontal_Distance_To_Roadways",  
"Vertical_Distance_To_Hydrology_x_Horizontal_Distance_To_Hydrology", "Vertical_Distance_To_Hydrology_x_Vertica
```

```

"Horizontal_Distance_to_Hydrology_x_Hillshade_9am", "Horizontal_Distance_to_Hydrology_x_Hillshade_Noon",
"Horizontal_Distance_to_Hydrology_x_Hillshade_3pm",
"Horizontal_Distance_to_Hydrology_x_Horizontal_Distance_to_Fire_Points",
"Vertical_Distance_to_Hydrology_x_Horizontal_Distance_to_Roadways"], axis=1) data_train#obtain the correlations of each
features in dataset corrmatrix = data_train.corr() top_corr_features = corrmatrix.index plt.figure(figsize=(20,20)) #plot heat map
g=sns.heatmap(data_train[top_corr_features].corr(),annot=True,cmap="RdYlGn")

```

Green correlations are still there, I do another drop

NOT USED

```

#If run twice, error will appear as labels are already deleted! data_train =
data_train.drop(["Horizontal_Distance_to_Roadways_x_Hillshade_9am", "Horizontal_Distance_to_Roadways_x_Hillshade_Noon",
"Horizontal_Distance_to_Roadways_x_Hillshade_3pm",
"Horizontal_Distance_to_Roadways_x_Horizontal_Distance_to_Fire_Points", "Vertical_Distance_to_Hydrology_x_Hillshade_9am",
"Vertical_Distance_to_Hydrology_x_Hillshade_Noon", "Vertical_Distance_to_Hydrology_x_Hillshade_3pm"],
axis=1) data_train#obtain the correlations of each features in dataset corrmatrix = data_train.corr() top_corr_features = corrmatrix.index
plt.figure(figsize=(20,20)) #plot heat map g=sns.heatmap(data_train[top_corr_features].corr(),annot=True,cmap="RdYlGn")

```

Doing an extra round of drops

NOT USED

```

data_train = data_train.drop(["Aspect_x_Hillshade_9am", "Aspect_x_Hillshade_Noon", "Aspect_x_Hillshade_3pm",
"ratio_Hillshade_3pm"], axis=1) data_train#obtain the correlations of each features in dataset corrmatrix = data_train.corr()
top_corr_features = corrmatrix.index plt.figure(figsize=(20,20)) #plot heat map
g=sns.heatmap(data_train[top_corr_features].corr(),annot=True,cmap="RdYlGn")

```

Now removing perfect colinearity 1s

NOT USED

```

data_train = data_train.drop(["binned_elevation", "Mixed_Sedimentary_Soil", "Elevation^2"], axis=1) data_train.head()#obtain the
correlations of each features in dataset corrmatrix = data_train.corr() top_corr_features = corrmatrix.index plt.figure(figsize=(20,20))
#plot heat map g=sns.heatmap(data_train[top_corr_features].corr(),annot=True,cmap="RdYlGn") data_train =
data_train.drop(["Type_Other"], axis=1) data_train = data_train.drop(["Hillshade_Noon^2"], axis=1)

```

We **split** the dataset to train and validation set, in order to test our models. We use stratify to have a balanced dataset with regards to y predictor

```

In [167]: X = data_train.drop(['Cover_Type'], axis=1)
y = data_train['Cover_Type']
column_list = X.columns

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.20, random_state=37, stratify=y)
print("The shape of validation data: {} and {}".format(X_val.shape, y_val.shape))
print("The shape of training data: {} and {}".format(X_train.shape, y_train.shape))

```

The shape of validation data: (3024, 107) and (3024,)
The shape of training data: (12096, 107) and (12096,)

For the standardization we need only numerical values, since these have been already encoded we use the names to filter out the dummy variables

```

In [168]: scale_numerical = [column for column in column_list if 'Soil' not in column and 'Wilderness_Area'
scale_categorial = [column for column in column_list if column not in scale_numerical ]

```

We want to see only the dummy variables filtered

```

In [169]: numerical_train = data_train.filter(items=scale_numerical)

```

```

In [170]: categorial_train = data_train.filter(items=scale_categorial)

```

We proceed now to standardize using the Standard Scaler after trying MinMax, which didn't bring good results

```

In [171]: scaler = StandardScaler()
#scaler = MinMaxScaler()

```

```

In [172]: X_train[scale_numerical] = scaler.fit_transform(X_train[scale_numerical])
X_val[scale_numerical] = scaler.transform(X_val[scale_numerical])

```

6.4.1 Number of feature selection

We will start with 46 maximum features as a point of start.

```
In [173... num_feats=46

In [174... def get_feature_importance(clf, feature_names):
    """
    Function to print the most important features of a logreg classifier
    based on the coefficient values
    """
    return pd.DataFrame(
        {
            'variable': feature_names, # Feature names
            'coefficient': clf.coef_[0] # Feature Coefficients
        }
    ) \
    .round(decimals=2) \
    .sort_values('coefficient', ascending=False) \
    .style.bar(color=['red', 'green'], align='zero')
```

Embedded Method

6.4.2 Lasso Regularization

When applying regularization to a Machine Learning model, we add a penalty to the model parameters to avoid that our model tries to resemble too closely our input data. In this way, we can make our model less complex and we can avoid overfitting (making learn to our model, not just the key data characteristics but also it's intrinsic noise). One of the possible Regularization Methods is Lasso (L1) Regression. When using Lasso Regression, the coefficients of the inputs features gets shrunk if they are not positively contributing to our Machine Learning model training. In this way, some of the features might get automatically discarded assigning them coefficients equal to zero.

```
In [175... lasso_mod = linear_model.LogisticRegression(penalty='l1', solver='saga', multi_class='multinomial')
print("Accuracy = {:.4}".format(np.mean(cross_val_score(lasso_mod, X_train, y_train, cv=5))))

/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:35
0: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:35
0: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:35
0: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:35
0: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
Accuracy = 0.7321

/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:35
0: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(

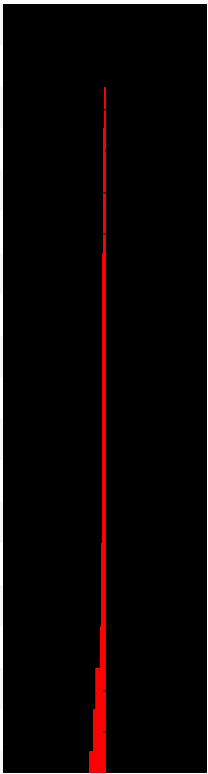
In [176... get_feature_importance(lasso_mod.fit(X_train,y_train), X_train.columns.get_level_values(0).tolist()

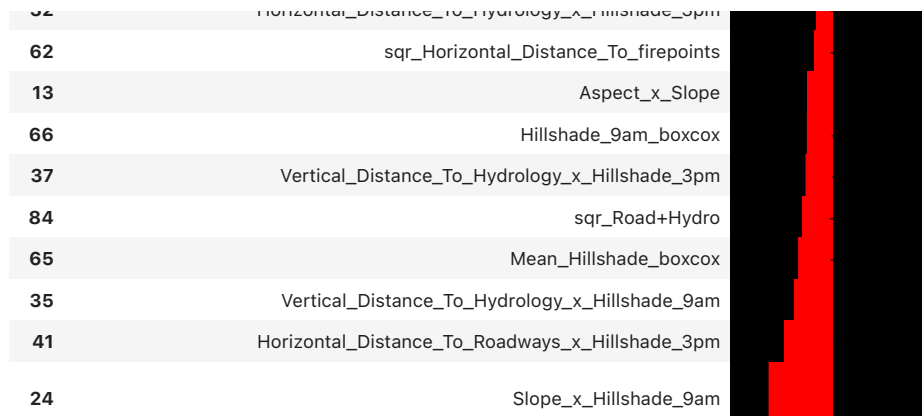
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:35
0: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
```

```
Out [176]:
```

	variable	coefficient
95	sqr_Ele-hydro	2.820000
81	binned_elevation	1.8300
5	Elevation_x_Slope	1.550

6	Elevation_x_Horizontal_Distance_To_Hydrology	1.470
10	Elevation_x_Hillshade_Noon	0.99
0	Wilderness_Area1	0.9
7	Elevation_x_Vertical_Distance_To_Hydrology	0.9
101	Subalpine_Climate	0.1
9	Elevation_x_Hillshade_9am	0.1
57	Hillshade_3pm^2	0.1
104	Glacial_Soil	0.1
73	Hillshade_Noon_bin	0.1
50	Aspect^2	0.1
70	ratio_Hillshade_Noon	0.1
45	Hillshade_9am_x_Horizontal_Distance_To_Fire_Points	0.1
60	Sqr_Horizontal_Distance_To_Roadways	0.1
29	Horizontal_Distance_To_Hydrology_x_Horizontal_Distance_To_Roadways	0.1
93	sqr_Ele-fire	0.1
92	sqr_Ele+fire	0.1
54	Horizontal_Distance_To_Roadways^2	0.1
8	Elevation_x_Horizontal_Distance_To_Roadways	0.1
89	sqr_Ele+Road+Fire+Hydro	0.1
38	Vertical_Distance_To_Hydrology_x_Horizontal_Distance_To_Fire_Points	0.1
77	Aspect_North	0.1
103	Alluvium_Soil	0.1
27	Slope_x_Horizontal_Distance_To_Fire_Points	0.1
33	Horizontal_Distance_To_Hydrology_x_Horizontal_Distance_To_Fire_Points	0.1
28	Horizontal_Distance_To_Hydrology_x_Vertical_Distance_To_Hydrology	0.1
102	Alpine_Climate	0.1
15	Aspect_x_Vertical_Distance_To_Hydrology	0.1
46	Hillshade_Noon_x_Hillshade_3pm	0.1
4	Elevation_x_Aspect	0.1
12	Elevation_x_Horizontal_Distance_To_Fire_Points	0.1
83	sqr_Road-Fire	0.1
94	sqr_Ele+hydro	0.1
68	Hillshade_3pm_boxcox	0.1
76	sqr_Aspect	0.1
1	Wilderness_Area2	0.1
63	Mean_Hillshade	0.1
59	sqr_Distance_To_Hydrology	0.1
86	sqr_Hydro+Fire	0.1
53	Vertical_Distance_To_Hydrology^2	0.1
74	Hillshade_3pm_bin	0.000000
25	Slope_x_Hillshade_Noon	0.000000
97	Montane_Dry_Climate	0.000000
88	sqr_Road+Fire+Hydro	0.000000
17	Aspect_x_Hillshade_9am	0.000000
30	Horizontal_Distance_To_Hydrology_x_Hillshade_9am	0.000000
69	ratio_Hillshade_3pm	0.000000
26	Slope_x_Hillshade_3pm	0.000000
72	ratio_Hillshade_3pm	0.000000

1	ratio_Hillshade_9am	0.000000
18	Aspect_x_Hillshade_Noon	0.000000
98	Montane_Climate	0.000000
23	Slope_x_Horizontal_Distance_To_Roadways	-0.000000
78	Aspect_East	0.000000
80	Aspect_West	0.000000
82	sqr_Road+Fire	0.000000
90	sqr_Ele+road	0.000000
19	Aspect_x_Hillshade_3pm	0.000000
85	sqr_Road-Hydro	0.000000
31	Horizontal_Distance_To_Hydrology_x_Hillshade_Noon	0.000000
87	sqr_Hydro-Fire	0.000000
61	SqrSlope	0.000000
105	Mixed_Sedimentary_Soil	0.000000
47	Hillshade_Noon_x_Horizontal_Distance_To_Fire_Points	-0.000000
49	Constant Term	0.000000
39	Horizontal_Distance_To_Roadways_x_Hillshade_9am	0.000000
99	Montane_Dry_and_Montane_Climate	0.000000
40	Horizontal_Distance_To_Roadways_x_Hillshade_Noon	0.000000
52	Horizontal_Distance_To_Hydrology^2	0.000000
3	Wilderness_Area4	0.000000
36	Vertical_Distance_To_Hydrology_x_Hillshade_Noon	0.000000
55	Hillshade_9am^2	0.000000
56	Hillshade_Noon^2	0.000000
11	Elevation_x_Hillshade_3pm	0.000000
44	Hillshade_9am_x_Hillshade_3pm	0.000000
58	Distance_To_Hydrology	0.000000
100	Montane_and_Subalpine_Climate	0.000000
48	Hillshade_3pm_x_Horizontal_Distance_To_Fire_Points	0.000000
91	sqr_Ele-road	
51	Slope^2	
96	Lower_Montane_Climate	
75	Hillshade_allday_bin	
79	Aspect_South	
72	Hillshade_9am_bin	
43	Hillshade_9am_x_Hillshade_Noon	
20	Aspect_x_Horizontal_Distance_To_Fire_Points	
16	Aspect_x_Horizontal_Distance_To_Roadways	
42	Horizontal_Distance_To_Roadways_x_Horizontal_Distance_To_Fire_Points	
21	Slope_x_Horizontal_Distance_To_Hydrology	
22	Slope_x_Vertical_Distance_To_Hydrology	
34	Vertical_Distance_To_Hydrology_x_Horizontal_Distance_To_Roadways	
106	Igneous_and_Metamorphic_Soil	
64	Mean_Hillshade_bin	
14	Aspect_x_Horizontal_Distance_To_Hydrology	
67	Hillshade_Noon_boxcox	
2	Wilderness_Area3	
22	Horizontal_Distance_To_Hydrology_x_Hillshade_3pm	



Similar performance w.r.t the un-regularized models. However, you can see how the feature coefficients are smaller than the original ones, due to the regularization.

Both methods give quite a lot of importance to elevation related variables

Let's look at how the coefficient weights and accuracy scores change along with the different regularization values. To that end, I have implemented the following piece of code. Do not be overwhelmed by it. It basically defines a list of regularization values to test and train a new Logistic Regression model for one of these regularization values. We keep track of the coefficient values and the accuracy of each of these models to plot them according to the defined regularization parameters.

```
#Watch out, takes 10 min to run on an i7 processor
lasso_mod =
linear_model.LogisticRegression(penalty='l1',solver='saga',max_iter=5000) alphas = 10**np.linspace(-1,-3,100) coefs_ = [] scores_
= [] for a in alphas: lasso_mod.set_params(C=a) scores_.append(np.mean(cross_val_score(lasso_mod, X_train, y_train, cv=5))) #
Appends the accuracy of the model coefs_.append(lasso_mod.fit(X_train, y_train).coef_.ravel().copy()) # Appends the coefficient
of the model coefs_ = np.array(coefs_) scores_ = np.array(scores_) fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))
fig.suptitle('Logistic Regression Path', fontsize=20) # Coeff Weights Plot ax1.plot(alphas, coefs_, marker='o') ymin, ymax = plt.ylim()
ax1.set_ylabel('Coefficient Weights', fontsize = 15) ax1.set_xlabel('log(C)', fontsize = 15) ax1.axis('tight') # Accuracy Plot
ax2.plot(alphas, scores_, marker='o') ymin, ymax = plt.ylim() ax2.set_ylabel('Accuracy Score', fontsize = 15) ax2.set_xlabel('log(C)',
fontsize = 15) ax2.axis('tight') plt.show()
```

As you can see in the left figure, the smaller the alpha value (alpha), the larger the regularization and, consequently, the smaller the weights of the coefficients. This is because, if we check the sklearn documentation, we will see that this value is the: "Inverse of regularization strength."

When regularization is large enough (i.e., alpha is small), the values of the coefficients are close to 0 (i.e., null model).

As there is a trade-off between variance (i.e., less over-fitted model --> more regularization) and bias (i.e., learning more from the training set --> less regularization), You must find the optimal alpha value.

As you can see in the right figure, this value is achieved with small alpha values (i.e., more regularization).

```
In [177]: embedded_lr_selector = SelectFromModel(LogisticRegression(C=0.04,penalty='l1',solver='saga',multi
embedded_lr_selector.fit(X_train, y_train))
```

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:35
0: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
warnings.warn(
```

```
Out[177]: ► SelectFromModel
► estimator: LogisticRegression
► LogisticRegression
```

```
In [178]: embedded_lr_support = embedded_lr_selector.get_support()
embedded_lr_feature = X_train.loc[:,embedded_lr_support].columns.tolist()
print(str(len(embedded_lr_feature)), 'selected features')
```

46 selected features

```
In [179]: embedded_lr_feature
```

```
Out[179]: ['Wilderness_Areal',
'Wilderness_Area3',
'Wilderness_Area4',
'Elevation_x_Hillshade_9am',
'Elevation_x_Hillshade_Noon',
'Elevation_x_Hillshade_3pm',
'Aspect_x_Horizontal_Distance_To_Roadways',
'Slope_x_Vertical_Distance_To_Hydrology',
'Slope_x_Horizontal_Distance_To_Roadways',
'Slope_x_Hillshade_9am',
'Slope_x_Hillshade_3pm',
'Slope_x_Horizontal_Distance_To_Fire_Points',
'Horizontal_Distance_To_Hydrology_x_Horizontal_Distance_To_Roadways',
'Horizontal_Distance_To_Hydrology_x_Horizontal_Distance_To_Fire_Points',
'Vertical_Distance_To_Hydrology_x_Hillshade_9am',
'Vertical_Distance_To_Hydrology_x_Hillshade_3pm',
'Horizontal_Distance_To_Roadways_x_Horizontal_Distance_To_Fire_Points',
'Hillshade_9am_x_Hillshade_Noon',
'Hillshade_Noon_x_Horizontal_Distance_To_Fire_Points',
'Slope^2',
'Horizontal_Distance_To_Hydrology^2',
'Vertical_Distance_To_Hydrology^2',
'Hillshade_3pm^2',
'sqr_Distance_To_Hydrology',
'SqrSlope',
'Mean_Hillshade_boxcox',
'Hillshade_9am_boxcox',
'Hillshade_Noon_boxcox',
'Hillshade_Noon_bin',
'Hillshade_allday_bin',
'Aspect_South',
'Aspect_West',
'binned_elevation',
'sqr_Road-Fire',
'sqr_Road-Hydro',
'sqr_Ele-road',
'sqr_Ele+fire',
'sqr_Ele-fire',
'sqr_Ele+hydro',
'sqr_Ele-hydro',
'Lower_Montane_Climate',
'Montane_Climate',
'Montane_and_Subalpine_Climate',
'Subalpine_Climate',
'Alpine_Climate',
'Igneus_and_Metamorphic_Soil']
```

6.4.3 Filter Method

6.4.3.1 Anova F-value

Chi-Square does not work because it needs non - negative values. For that reason we will use Anova. It is a univariate filter method that uses variance to find out the separability of the individual features between classes. It applies to multi-class endpoints. The SelectKBest class just scores the features using a function `f_classif` and then "removes all but the k highest scoring features."

f_classif = ANOVA F-value between label/feature for classification tasks.

k is the prior selected numbers of chosen features we want to select for further selection.

```
In [180... #Code from class Forum, select the best features
anov_selector = SelectKBest(f_classif, k=num_feats)
anov_selector.fit(X_train, y_train)
```

```

/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn/feature_selection/_univariate_selection.py:110: UserWarning: Features [49] are constant.
  warnings.warn("Features %s are constant." % constant_features_idx, UserWarning)
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/sklearn/feature_selection/_univariate_selection.py:111: RuntimeWarning: invalid value encountered in true_divide
  f = msb / msd

```

```

Out[180]:
▼ SelectKBest
SelectKBest(k=46)

```

```

In [181]: anov_support = anov_selector.get_support()
# Get columns from original dataframe
anov_feature = X_train.iloc[:, anov_support].columns.tolist()
print(str(len(anov_feature)), 'selected features')

46 selected features

```

6.4.3.2. Pearson correlation

We get another set of features based on correlation to the target, this time giving more importance to the alpine climate

```

In [182]: # Create a list of the feature names
fig, ax = plt.subplots(figsize=(10,40))
# Instantiate the visualizer
visualizer = FeatureCorrelation(labels=None, sort=True)

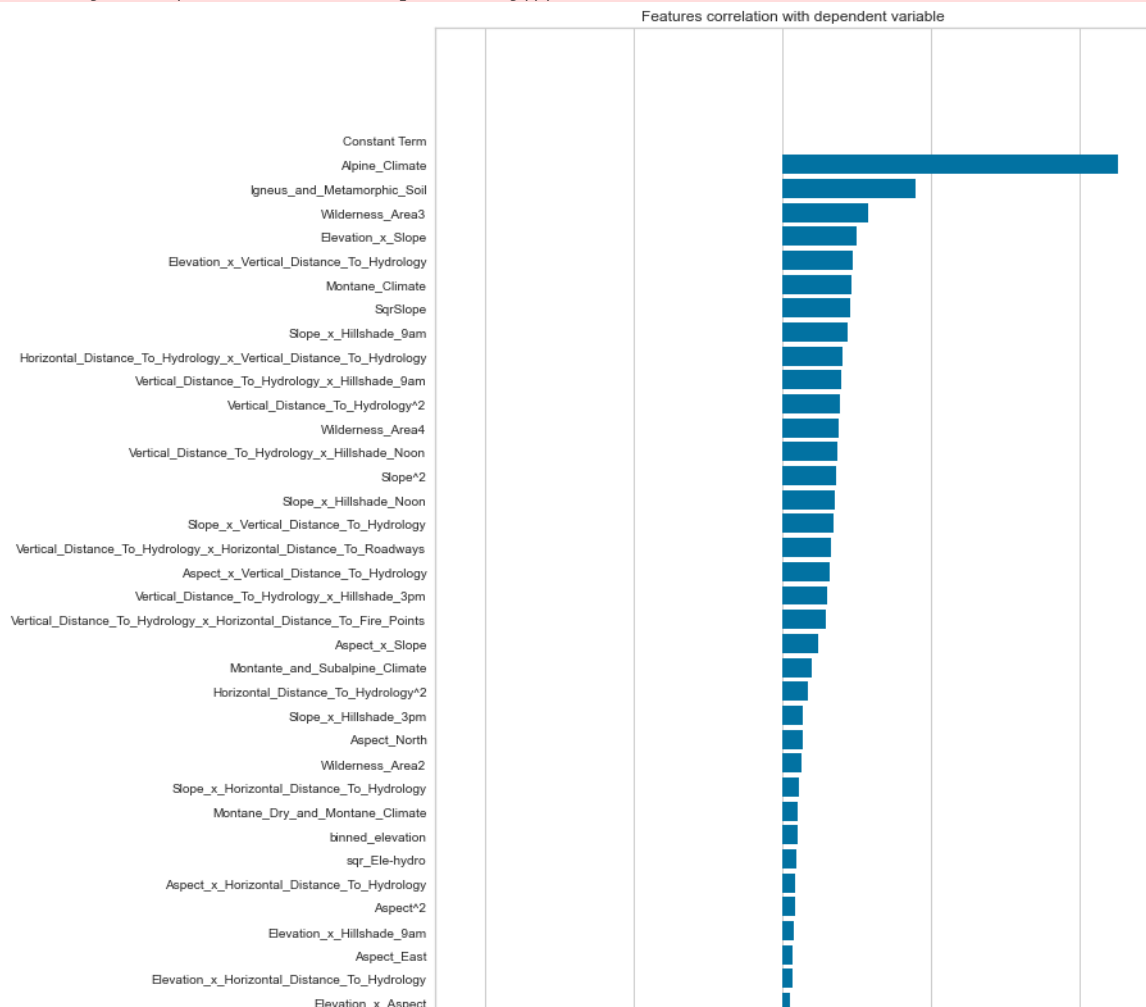
ax = visualizer.fit(X_train, y_train) # Fit the data to the visualizer
visualizer.show() # Finalize and render the figure

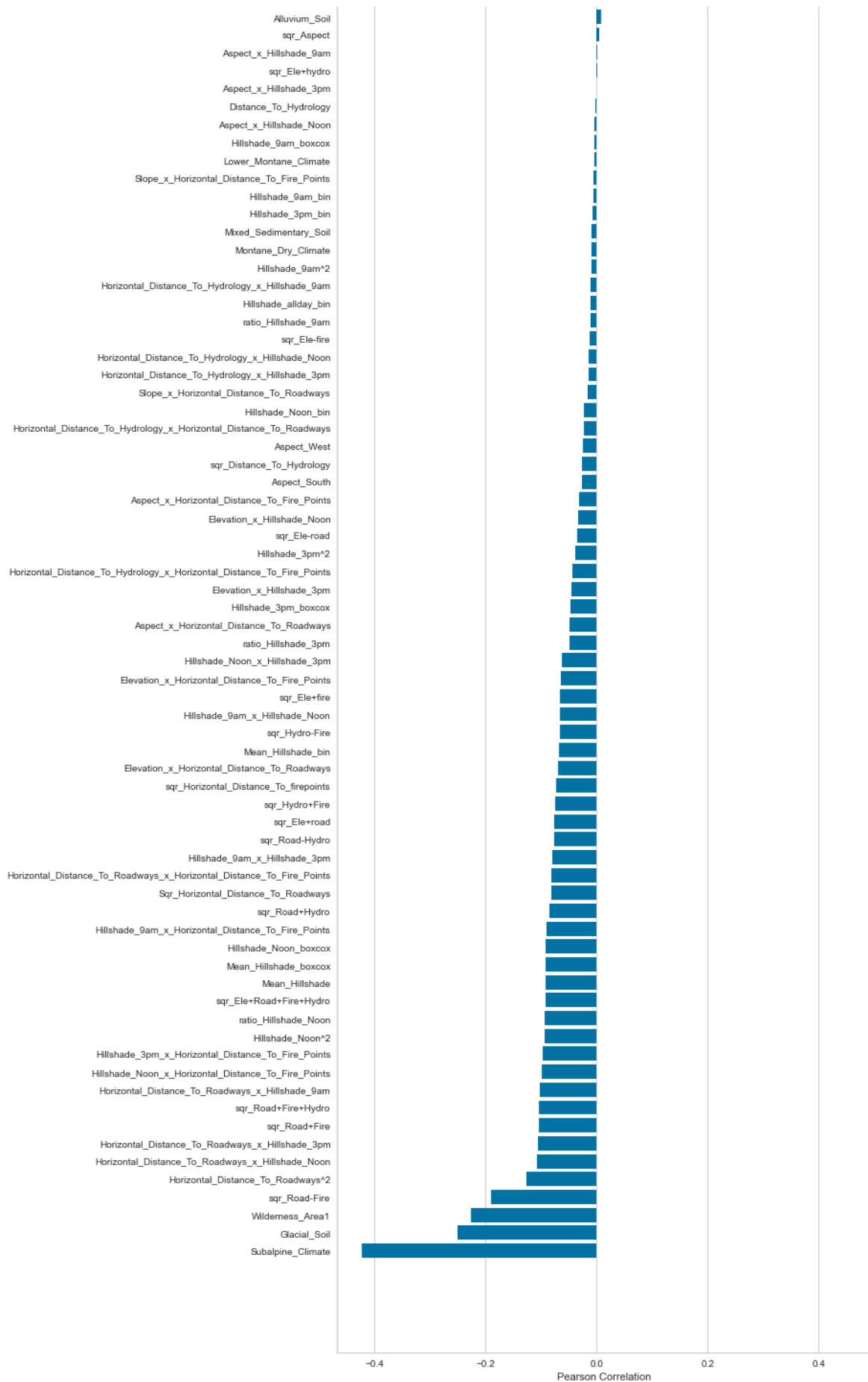
```

```

/Users/stephaniegessler/.local/lib/python3.9/site-packages/scipy/stats/_stats_py.py:4068: PearsonRConstantInputWarning: An input array is constant; the correlation coefficient is not defined.
  warnings.warn(PearsonRConstantInputWarning())

```





```
Out[182]: <AxesSubplot:title={'center': 'Features correlation with dependent variable'}, xlabel='Pearson Co
relation'>
```

```
In [183... pd.set_option('display.max_columns', None)
```

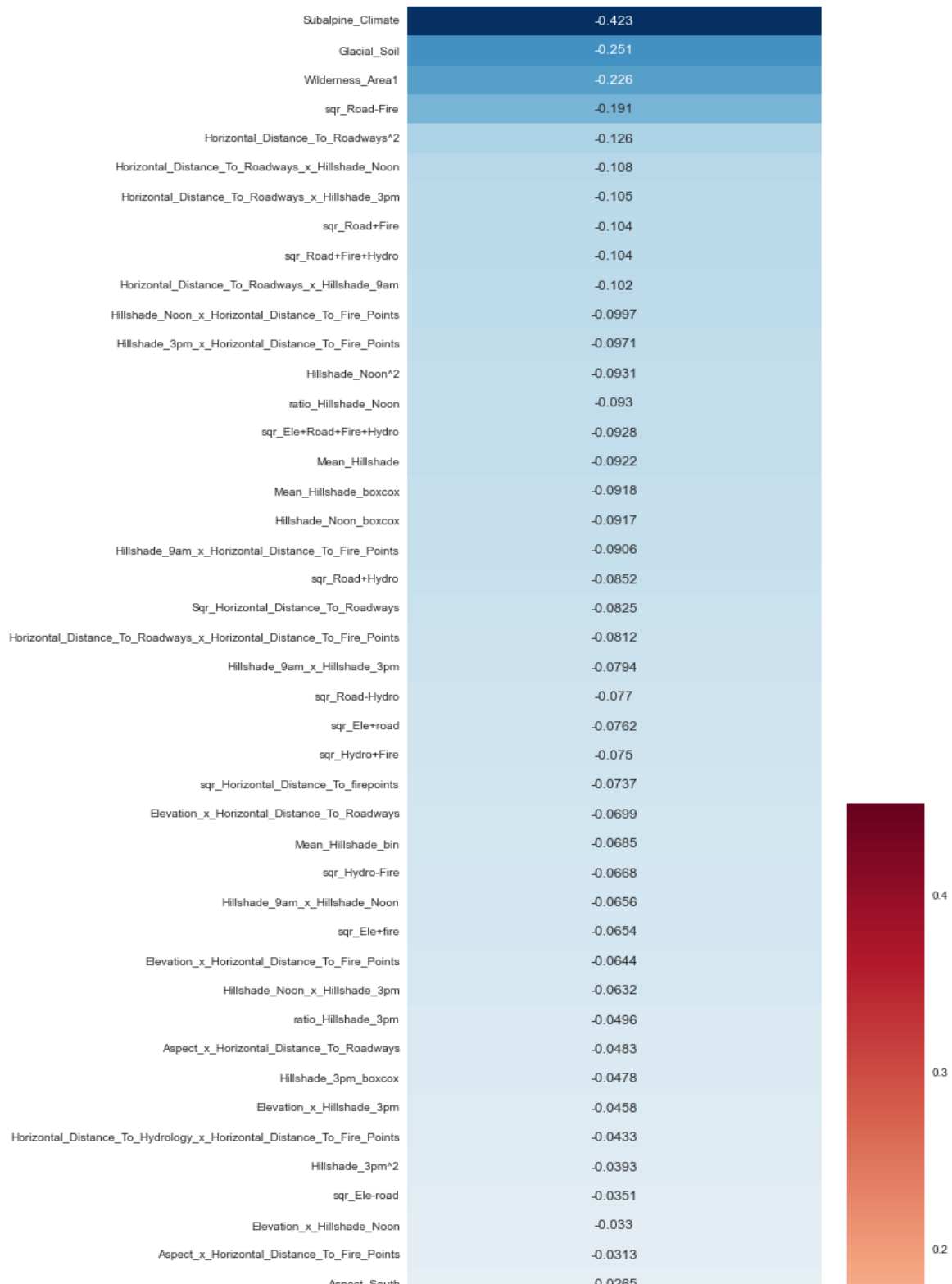
```
In [184... corr = data_train.corrwith(data_train["Cover_Type"])
X_y = data_train.copy()
X_y['Cover_Type'] = y_train
```

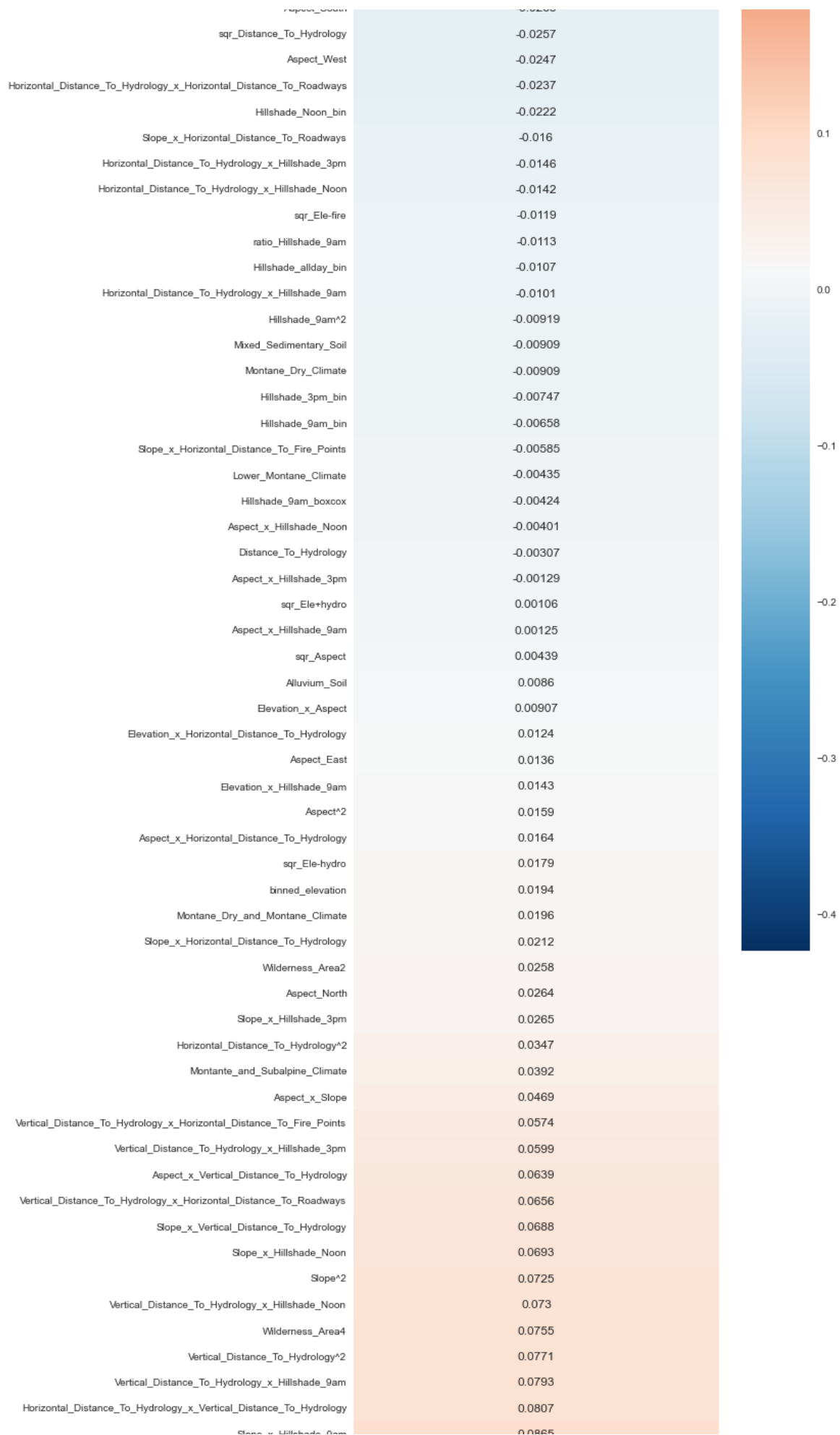
This plot will show the correlation to the Cover_type from each individual feature.

```
In [185... fig, ax = plt.subplots(figsize=(8,50))

corr_matrix = X_y.corr()

corr_target = corr_matrix[['Cover_Type']].drop(labels=['Cover_Type'])
corr_target_sorted = corr_target.sort_values(by = 'Cover_Type')
sns.heatmap(corr_target_sorted, annot=True, fmt='.3', cmap='RdBu_r', ax=ax)
plt.show()
```





ridge_A_Temperature_2am	0.0000
SqrSlope	0.0906
Montane_Climate	0.0928
Elevation_x_Vertical_Distance_To_Hydrology	0.0948
Elevation_x_Slope	0.0987
Wilderness_Area3	0.114
Igneous_and_Metamorphic_Soil	0.179
Alpine_Climate	0.451
Constant Term	

Cover_Type

The function below helps to select the features that we specified in the beginning.

```
In [186... def cor_selector(X, y,num_feats):
    cor_list = []
    feature_name = X.columns.tolist()
    # calculate the correlation with y for each feature
    for i in X.columns.tolist():
        cor = np.corrcoef(X[i], y)[0, 1]
        cor_list.append(cor)
    # replace NaN with 0
    cor_list = [0 if np.isnan(i) else i for i in cor_list]
    # feature name
    cor_feature = X.iloc[:,np.argsort(np.abs(cor_list))[-num_feats:]].columns.tolist()
    # feature selection? 0 for not select, 1 for select
    cor_support = [True if i in cor_feature else False for i in feature_name]
    return cor_support, cor_feature
cor_support, cor_feature = cor_selector(X_train, y_train,num_feats)
print(str(len(cor_feature)), 'selected features')
```

46 selected features

```
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/numpy/lib/function_base.py:2691
: RuntimeWarning: invalid value encountered in true_divide
  c /= stddev[:, None]
/Users/stephaniegessler/opt/anaconda3/lib/python3.9/site-packages/numpy/lib/function_base.py:2692
: RuntimeWarning: invalid value encountered in true_divide
  c /= stddev[None, :]
```

```
In [187... cor_feature
```

```
Out[187]: ['sqr_Hydro-Fire',
'Mean_Hillshade_bin',
'Slope_x_Vertical_Distance_To_Hydrology',
'Slope_x_Hillshade_Noon',
'Elevation_x_Horizontal_Distance_To_Roadways',
'Slope^2',
'Vertical_Distance_To_Hydrology_x_Hillshade_Noon',
'sqr_Horizontal_Distance_To_firepoints',
'sqr_Hydro+Fire',
'Wilderness_Area4',
'sqr_Ele+road',
'sqr_Road-Hydro',
'Vertical_Distance_To_Hydrology^2',
'Vertical_Distance_To_Hydrology_x_Hillshade_9am',
'Hillshade_9am_x_Hillshade_3pm',
'Horizontal_Distance_To_Hydrology_x_Vertical_Distance_To_Hydrology',
'Horizontal_Distance_To_Roadways_x_Horizontal_Distance_To_Fire_Points',
'Sqr_Horizontal_Distance_To_Roadways',
'sqr_Road+Hydro',
'Slope_x_Hillshade_9am',
'SqrSlope',
'Hillshade_9am_x_Horizontal_Distance_To_Fire_Points',
'Hillshade_Noon_boxcox',
'Mean_Hillshade_boxcox',
'Mean_Hillshade',
'Montane_Climate',
'sqr_Ele+Road+Fire+Hydro',
'ratio_Hillshade_Noon',
'Hillshade_Noon^2',
'Elevation_x_Vertical_Distance_To_Hydrology',
'Hillshade_3pm_x_Horizontal_Distance_To_Fire_Points',
'Elevation_x_Slope',
'Hillshade_Noon_x_Horizontal_Distance_To_Fire_Points',
...]
```

```
'Horizontal_Distance_To_Roadways_x_Hillshade_9am',
'sqr_Road+Fire+Hydro',
'sqr_Road+Fire',
'Horizontal_Distance_To_Roadways_x_Hillshade_3pm',
'Horizontal_Distance_To_Roadways_x_Hillshade_Noon',
'Wilderness_Area3',
'Horizontal_Distance_To_Roadways^2',
'Igneus_and_Metamorphic_Soil',
'sqr_Road-Fire',
'Wilderness_Areal',
'Glacial_Soil',
'Subalpine_Climate',
'Alpine_Climate']
```

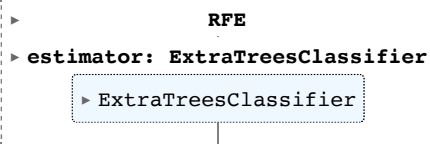
6.5. Recursive Feature Elimination

The goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

```
In [188... rfe_selector = RFE(estimator=ExtraTreesClassifier(), n_features_to_select=num_feats, step=10, ver
rfe_selector.fit(X_train, y_train)
#stimator=LogisticRegression(max_iter=3000)
```

```
Fitting estimator with 107 features.
Fitting estimator with 97 features.
Fitting estimator with 87 features.
Fitting estimator with 77 features.
Fitting estimator with 67 features.
Fitting estimator with 57 features.
Fitting estimator with 47 features.
```

```
Out[188]:
```



```
In [189... rfe_support = rfe_selector.get_support()
rfe_feature = X_train.loc[:,rfe_support].columns.tolist()
print(str(len(rfe_feature)), 'selected features')
```

```
46 selected features
```

```
In [190... rfe_feature
```

```
Out[190]: ['Wilderness_Areal',
'Wilderness_Area3',
'Wilderness_Area4',
'Elevation_x_Horizontal_Distance_To_Hydrology',
'Elevation_x_Horizontal_Distance_To_Roadways',
'Elevation_x_Hillshade_9am',
'Elevation_x_Hillshade_Noon',
'Elevation_x_Horizontal_Distance_To_Fire_Points',
'Horizontal_Distance_To_Hydrology_x_Hillshade_Noon',
'Horizontal_Distance_To_Hydrology_x_Hillshade_3pm',
'Horizontal_Distance_To_Roadways_x_Hillshade_9am',
'Horizontal_Distance_To_Roadways_x_Hillshade_Noon',
'Horizontal_Distance_To_Roadways_x_Hillshade_3pm',
'Horizontal_Distance_To_Roadways_x_Horizontal_Distance_To_Fire_Points',
'Hillshade_9am_x_Hillshade_Noon',
'Hillshade_9am_x_Horizontal_Distance_To_Fire_Points',
'Hillshade_Noon_x_Horizontal_Distance_To_Fire_Points',
'Horizontal_Distance_To_Roadways^2',
'Hillshade_9am^2',
'Hillshade_Noon^2',
'Distance_To_Hydrology',
'sqr_Distance_To_Hydrology',
'Sqr_Horizontal_Distance_To_Roadways',
'sqr_Horizontal_Distance_To_firepoints',
'Hillshade 9am boxcox',
```



```
'ratio_Hillshade_9am',
'binmed_elevation',
'sqr_Road+Fire',
'sqr_Road-Fire',
'sqr_Road+Hydro',
'sqr_Road-Hydro',
'sqr_Hydro+Fire',
'sqr_Hydro-Fire',
'sqr_Road+Fire+Hydro',
'sqr_Ele+Road+Fire+Hydro',
'sqr_Ele+road',
'sqr_Ele-road',
'sqr_Ele+fire',
'sqr_Ele-fire',
'sqr_Ele+hydro',
'sqr_Ele-hydro',
'Lower_Montane_Climate',
'Montane_Climate',
'Subalpine_Climate',
'Alpine_Climate',
'Igneus_and_Metamorphic_Soil']
```

6.6. Tree-based: SelectFromModel

6.6.1 RandomForestClassifier

Embedded methods use algorithms that have built-in feature selection methods. We can also use RandomForest to select features based on feature importance. We calculate feature importance using node impurities in each decision tree. In Random forest, the final feature importance is the average of all decision tree feature importance.

```
In [191...] embedded_rf_selector = SelectFromModel(RandomForestClassifier(n_estimators=100), max_features=num_
embedded_rf_selector.fit(X_train, y_train)
```

```
Out[191]: ▸ SelectFromModel
          ▸ estimator: RandomForestClassifier
            ▸ RandomForestClassifier
```

```
In [192...] embedded_rf_support = embedded_rf_selector.get_support()
embedded_rf_feature = X_train.loc[:,embedded_rf_support].columns.tolist()
print(str(len(embedded_rf_feature)), 'selected features')
```

31 selected features

```
In [193...] embedded_rf_feature
```

```
Out[193]: ['Wilderness_Area4',
'Elevation_x_Horizontal_Distance_To_Hydrology',
'Elevation_x_Horizontal_Distance_To_Roadways',
'Elevation_x_Hillshade_9am',
'Elevation_x_Hillshade_Noon',
'Elevation_x_Horizontal_Distance_To_Fire_Points',
'Horizontal_Distance_To_Hydrology_x_Hillshade_3pm',
'Horizontal_Distance_To_Roadways_x_Hillshade_9am',
'Horizontal_Distance_To_Roadways_x_Hillshade_Noon',
'Horizontal_Distance_To_Roadways_x_Horizontal_Distance_To_Fire_Points',
'Hillshade_9am_x_Hillshade_Noon',
'Horizontal_Distance_To_Roadways^2',
'Sqr_Horizontal_Distance_To_Roadways',
'Hillshade_9am_boxcox',
'ratio_Hillshade_9am',
'binmed_elevation']
```

```

'-----',
'sqr_Road+Fire',
'sqr_Road-Fire',
'sqr_Road+Hydro',
'sqr_Road-Hydro',
'sqr_Road+Fire+Hydro',
'sqr_Ele+Road+Fire+Hydro',
'sqr_Ele+road',
'sqr_Ele-road',
'sqr_Ele+fire',
'sqr_Ele-fire',
'sqr_Ele+hydro',
'sqr_Ele-hydro',
'Lower_Montane_Climate',
'Subalpine_Climate',
'Alpine_Climate']

```

6.6.2 XgBoost

XGBoost is relatively straightforward to retrieve importance scores for each attribute.

Generally, importance provides a score that indicates how useful or valuable each feature was in the construction of the boosted decision trees within the model. The more an attribute is used to make key decisions with decision trees, the higher its relative importance. This importance is calculated explicitly for each attribute in the dataset, allowing attributes to be ranked and compared to each other.

```

In [194... #y_train needs to be transformed from 1,2,3,4,5,6,7 to 0 1 2 3 4 5,6
le = LabelEncoder()
y_train1 = le.fit_transform(y_train)
model=xgb.XGBClassifier(n_jobs=-1)
embedded_xgb_selector = SelectFromModel(model, max_features=num_feats)
embedded_xgb_selector.fit(X_train, y_train1)

```

```

Out[194]:
└─ SelectFromModel
  └─ estimator: XGBClassifier
    └─ XGBClassifier

```

```

In [195... embedded_xgb_support = embedded_xgb_selector.get_support()
embedded_xgb_feature = X_train.loc[:,embedded_xgb_support].columns.tolist()
print(str(len(embedded_xgb_feature)), 'selected features')

17 selected features

```

```

In [196... embedded_xgb_feature

```

```

Out[196]: ['Wilderness_Areal',
'Wilderness_Area3',
'Wilderness_Area4',
'Elevation_x_Horizontal_Distance_To_Hydrology',
'Horizontal_Distance_To_Hydrology_x_Hillshade_3pm',
'Hillshade_9am_x_Hillshade_Noon',
'Aspect_North',
'Aspect_West',
'binned_elevation',
'sqr_Road+Fire',
'Lower_Montane_Climate',
'Montane_Climate',
'Montane_and_Subalpine_Climate',
'Subalpine_Climate',
'Alpine_Climate',
'Alluvium_Soil',
'Igneus_and_Metamorphic_Soil']

```

6.6.3 ExtraTreesClassifier

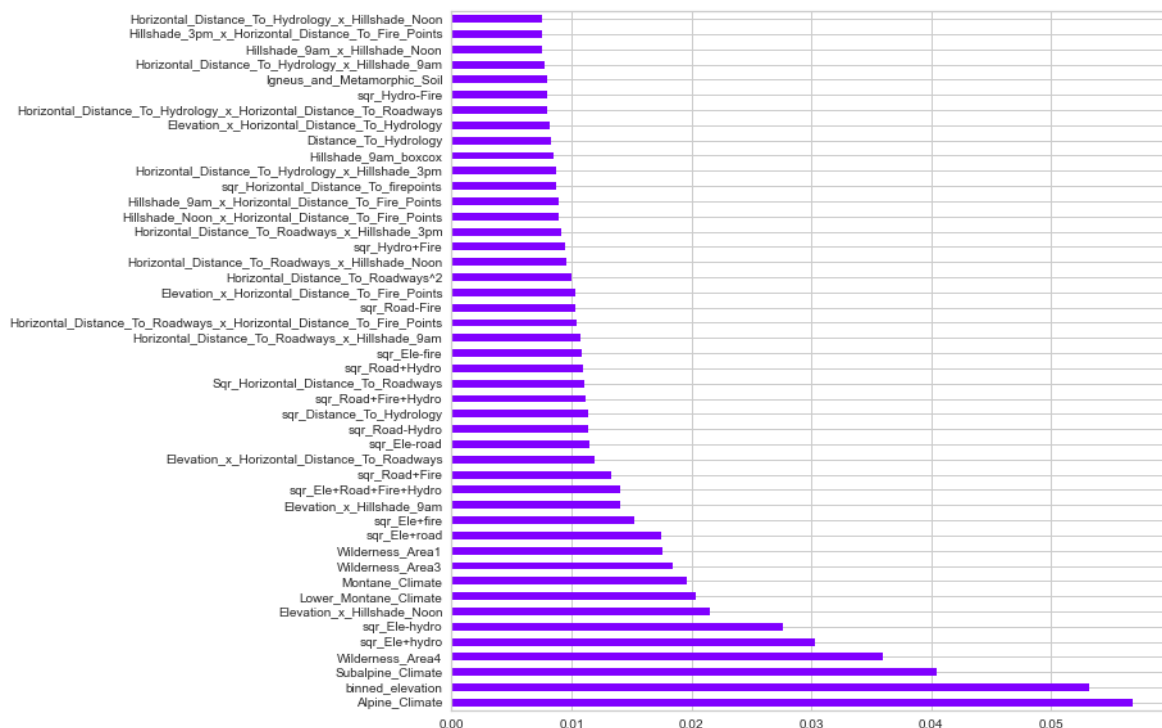
Each Decision Tree in the Extra Trees Forest is constructed from the original training sample. Then, at each test node, Each tree is provided with a random sample of k features from the feature-set from which each decision tree must select the best feature to split the data based on some mathematical criteria (typically the Gini Index). This random sample of features leads to the creation of multiple de-correlated decision trees. To perform feature selection using the above forest structure, during the construction of the forest, for each feature, the normalized total reduction in the mathematical criteria used in the decision of feature of split (Gini Index if the Gini Index is used in the construction of the forest) is computed. This value is called the Gini Importance of the feature.

```
In [197... extra_tree_model= ExtraTreesClassifier()
# Training the model
extra_tree_forest_selector = SelectFromModel(ExtraTreesClassifier(max_features=num_feats,criterio
extra_tree_forest_selector.fit(X_train, y_train)
```

```
Out[197]: ▸ SelectFromModel
▸ estimator: ExtraTreesClassifier
▸ ExtraTreesClassifier
```

```
In [198... fig, ax = plt.subplots(figsize=(10, 10))
extra_tree_model.fit(X_train,y_train)
extra_tree_model.feature_importances_

#plot graph of feature importances for better visualization
feat_importances = pd.Series(extra_tree_model.feature_importances_, index=X_train.columns)
ax = feat_importances.nlargest(num_feats).plot(kind='barh', colormap = 'rainbow')
plt.show()
```



```
In [199... extra_tree_forest_support = extra_tree_forest_selector.get_support()
extra_tree_forest_feature = X_train.loc[:,extra_tree_forest_support].columns.tolist()
print(str(len(extra_tree_forest_feature)), 'selected features')
```

25 selected features

```
In [200... extra_tree_forest_feature
```

```
Out[200]: ['Wilderness_Area1',
           'Wilderness_Area3',
           'Wilderness_Area4',
           'Elevation_x_Hillshade_9am',
           'Elevation_x_Hillshade_Noon',
           'Horizontal_Distance_To_Roadways_x_Horizontal_Distance_To_Fire_Points',
           'Horizontal_Distance_To_Roadways^2',
           'sqr_Distance_To_Hydrology',
           'Hillshade_9am_boxcox',
           'binned_elevation',
           'sqr_Road+Fire',
           'sqr_Road-Fire',
           'sqr_Road-Hydro',
           'sqr_Road+Fire+Hydro',
           'sqr_Ele+Road+Fire+Hydro',
           'sqr_Ele+road',
           'sqr_Ele-road',
           'sqr_Ele+fire',
           'sqr_Ele-fire',
           'sqr_Ele+hydro',
           'sqr_Ele-hydro',
           'Lower_Montane_Climate',
           'Montane_Climate',
           'Subalpine_Climate',
           'Alpine_Climate']
```

```
import pandas as pd
import numpy as np
import sklearn
chi2_score = pd.DataFrame(list(zip(X_train.columns,
extra_tree_forest_support.scores_,extra_tree_forest_selector.pvalues_)),
columns = ['feature','score','pvalue'])
chi2_score.sort_values('score', ascending = False)
chi2_score.style.set_precision(2)
```

6.7. Score of all methods together

In the table below you see the score which of the features has been ranked as the best based on the feature selection we specified. True means that this feature has been selected as one of the best predictor for the given model. Since we used different method, we put them together and select the best features above a threshold of 4. Meaning one feature has been ranked more than four times as one of the best predictors by the given feature selection method like ExtraTree, Pearson.

```
In [201]: feature_name = list(X_train.columns)
```

```
In [202]: pd.set_option('display.max_rows', None)
# put all selection together
feature_selection_df = pd.DataFrame({'Feature':feature_name, 'ExtraTree':extra_tree_forest_support
                                     'Random Forest':emdeded_rf_support, 'XGB':emdeded_xgb_support
# count the selected times for each feature
feature_selection_df['Total'] = np.sum(feature_selection_df, axis=1)
# display the top 100
feature_selection_df = feature_selection_df.sort_values(['Total','Feature'], ascending=False)
feature_selection_df.index = range(1, len(feature_selection_df)+1)
feature_selection_df.head(num_feats)
```

```
Out[202]:
```

	Feature	ExtraTree	Pearson	RFE	Anova	Logistics	Random Forest	XGB	To
1	Wilderness_Area4	True	True	True	True	True	True	True	
2	Subalpine_Climate	True	True	True	True	True	True	True	
3	Alpine_Climate	True	True	True	True	True	True	True	
4	sqr_Road-Hydro	True	True	True	True	True	True	False	
5	sqr_Road-Fire	True	True	True	True	True	True	False	
6	sqr_Road+Fire	True	True	True	True	False	True	True	

7	binned_elevation	True	False	True	True	True	True	True
8	Wilderness_Area3	True	True	True	True	True	False	True
9	Wilderness_Area1	True	True	True	True	True	False	True
10	Montane_Climate	True	True	True	True	True	False	True
11	Lower_Montane_Climate	True	False	True	True	True	True	True
12	Horizontal_Distance_To_Roadways_x_Horizontal_D...	True	True	True	True	True	True	False
13	sqr_Road+Fire+Hydro	True	True	True	True	False	True	False
14	sqr_Ele-hydro	True	False	True	True	True	True	False
15	sqr_Ele+road	True	True	True	True	False	True	False
16	sqr_Ele+hydro	True	False	True	True	True	True	False
17	sqr_Ele+fire	True	False	True	True	True	True	False
18	sqr_Ele+Road+Fire+Hydro	True	True	True	True	False	True	False
19	Horizontal_Distance_To_Roadways^2	True	True	True	True	False	True	False
20	Hillshade_9am_x_Hillshade_Noon	False	False	True	True	True	True	True
21	Elevation_x_Hillshade_Noon	True	False	True	True	True	True	False
22	Elevation_x_Hillshade_9am	True	False	True	True	True	True	False
23	sqr_Road+Hydro	False	True	True	True	False	True	False
24	sqr_Ele-road	True	False	True	False	True	True	False
25	sqr_Ele-fire	True	False	True	False	True	True	False
26	sqr_Distance_To_Hydrology	True	False	True	True	True	False	False
27	Sqr_Horizontal_Distance_To_Roadways	False	True	True	True	False	True	False
28	Igneus_and_Metamorphic_Soil	False	True	True	False	True	False	True
29	Horizontal_Distance_To_Roadways_x_Hillshade_Noon	False	True	True	True	False	True	False
30	Horizontal_Distance_To_Roadways_x_Hillshade_9am	False	True	True	True	False	True	False
31	Horizontal_Distance_To_Hydrology_x_Hillshade_3pm	False	False	True	True	False	True	True
32	Hillshade_Noon_x_Horizontal_Distance_To_Fire_P...	False	True	True	True	True	False	False
33	Hillshade_9am_boxcox	True	False	True	False	True	True	False
34	Elevation_x_Horizontal_Distance_To_Roadways	False	True	True	True	False	True	False
35	Elevation_x_Horizontal_Distance_To_Hydrology	False	False	True	True	False	True	True
36	sqr_Hydro-Fire	False	True	True	True	False	False	False
37	sqr_Hydro+Fire	False	True	True	True	False	False	False
38	sqr_Horizontal_Distance_To_firepoints	False	True	True	True	False	False	False
39	ratio_Hillshade_9am	False	False	True	True	False	True	False
40	Horizontal_Distance_To_Roadways_x_Hillshade_3pm	False	True	True	True	False	False	False
41	Hillshade_9am_x_Horizontal_Distance_To_Fire_Po...	False	True	True	True	False	False	False
42	Elevation_x_Horizontal_Distance_To_Fire_Points	False	False	True	True	False	True	False
43	Vertical_Distance_To_Hydrology_x_Hillshade_9am	False	True	False	False	True	False	False
44	Vertical_Distance_To_Hydrology^2	False	True	False	False	True	False	False
45	SqrSlope	False	True	False	False	True	False	False
46	Slope_x_Vertical_Distance_To_Hydrology	False	True	False	False	True	False	False

This code selects only the features above a threshold of four.

```
In [204... conditions = [(feature_selection_df['Total']>3)]
new_df = feature_selection_df.loc[feature_selection_df['Total'] >3]
```

```
In [205... feature_list = new_df['Feature'].to_list()
```

```
feature_list
```

```
Out[205]: ['Wilderness_Area4',
           'Subalpine_Climate',
           'Alpine_Climate',
           'sqr_Road-Hydro',
           'sqr_Road-Fire',
           'sqr_Road+Fire',
           'binned_elevation',
           'Wilderness_Area3',
           'Wilderness_Area1',
           'Montane_Climate',
           'Lower_Montane_Climate',
           'Horizontal_Distance_To_Roadways_x_Horizontal_Distance_To_Fire_Points',
           'sqr_Road+Fire+Hydro',
           'sqr_Ele-hydro',
           'sqr_Ele+road',
           'sqr_Ele+hydro',
           'sqr_Ele+fire',
           'sqr_Ele+Road+Fire+Hydro',
           'Horizontal_Distance_To_Roadways^2',
           'Hillshade_9am_x_Hillshade_Noon',
           'Elevation_x_Hillshade_Noon',
           'Elevation_x_Hillshade_9am',
           'sqr_Road+Hydro',
           'sqr_Ele-road',
           'sqr_Ele-fire',
           'sqr_Distance_To_Hydrology',
           'Sqr_Horizontal_Distance_To_Roadways',
           'Igneus_and_Metamorphic_Soil',
           'Horizontal_Distance_To_Roadways_x_Hillshade_Noon',
           'Horizontal_Distance_To_Roadways_x_Hillshade_9am',
           'Horizontal_Distance_To_Hydrology_x_Hillshade_3pm',
           'Hillshade_Noon_x_Horizontal_Distance_To_Fire_Points',
           'Hillshade_9am_boxcox',
           'Elevation_x_Horizontal_Distance_To_Roadways',
           'Elevation_x_Horizontal_Distance_To_Hydrology']
```

```
In [206... X_selected = data_train[feature_list]
            y_selected = data_train['Cover_Type']
```

```
In [207... #Covert the new features in a CSV which will be used by the model later on.
X_selected.to_csv('X_selected.csv')
y_selected.to_csv('y_selected.csv')
```

```
In [208... X_selected1 = pd.read_csv("X_selected.csv")
y_selected1 = pd.read_csv("y_selected.csv")
```

Check on the dataset if both have the same row length.

```
In [209... print(X_selected.shape)
print(y_selected.shape)

if X.shape[0] != y.shape[0]:
    print("X and y rows are mismatched, check dataset again")

(15120, 35)
(15120,)
```

Machine Learning II
Group Assignment

