Prediction Of Forest Cover Type

STAT 841- Statistical Learning: Classification

University of Waterloo
Fall 2019

TEAM FOREST

Lavanya Sharma
Lipsa Mishra
Rudrani Bhadra

Contents

1 Introduction

Motivation

Prediction of the forest cover type helps in determining the information about the forest land. This information is highly required for ecosystem management. Ecosystem management is a process that aims to conserve major ecological services. It also aims to restore natural resources while meeting needs of current and future generations. The principle objective of ecosystem management is the efficient maintenance and ethical use of natural resources. In recent past due to climate change, management of natural resources like forests have become even more important. Prediction of forest cover type contributes highly in giving us information about natural forest which helps in ecosystem management. Therefore, prediction of forest cover is an important task.

Problem Statement

The problem is to predict the forest cover type using the cartographic variables like aspect, slope, soil type, wilderness area and so on. The goal is to build a model that predicts what types of trees grow in an area based on the surrounding characteristics. Also, by building such a model, we will be able to understand which tree types can grow in more diverse environments. We will also be able to tell if certain tree types are more sensitive to environmental factors like elevation or soil type.
The forest cover study area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. These areas represent forests with minimal human-caused disturbances. Therefore, the existing forest cover types are more a result of ecological processes rather than forest management practices. The inputs to our algorithm will be the 54 numeric features and the output (as this is a classification task) will be one of seven possible class labels that describe the predominant kind of tree cover. We will apply different machine learning algorithms to achieve this task.

2 Data

Introducing Data

This dataset contains tree observations from four areas of the Roosevelt National Forest in Colorado. All observations are cartographic variables (no remote sensing) from 30 meter x 30 meter sections of forest. The dataset of "forest cover type" contains 581012 observations and 54 attributes. All observations are cartographic variables. The data is in raw form (not scaled) and contains binary (0 or 1) columns of data for qualitative independent variables (wilderness areas and soil types). In total the data set contains 10 quantitative variables, 4 wilderness areas and 40 soil type both of binary type. The dataset do not have any null values. The information about all the attributes and the class types is summarized below.

S.No.	Name	Data Type	Mean	Std-Deviation
1	Elevation	quantitative	2959.365301	279.984734
2	Aspect	quantitative	155.656807	111.913721
3	Slope	quantitative	14.103704	7.488242
4	Horizontal Distance To Hydrology	quantitative	269.428217	212.549356
5	Vertical Distance To Hydrology	quantitative	46.418855	58.295232
6	Horizontal Distance To Roadways	quantitative	2350.146611	1559.254870
7	Hillshade 9am	quantitative	212.146049	26.769889
8	Hillshade Noon	quantitative	223.318716	19.768697
9	Hillshade 3pm	quantitative	142.528263	38.274529
10	Horizontal Distance To Fire Points	quantitative	1980.291226	1324.195210
11	Soil Type (40 columns)	qualitative	-	-
12	Wilderness Area (4 columns)	qualitative	-	-

Table 1: Attributes of Forest Cover Type Dataset

Class Label	Cover Type
1	Spruce/Fir
2	Lodgepole Pine
3	Ponderose Pine
4	Cottonwood/Willow
5	Aspen
6	Douglas-fir
7	Kurmmholz

Table 2: Class label of Forest Cover Type Dataset

Origin of Data

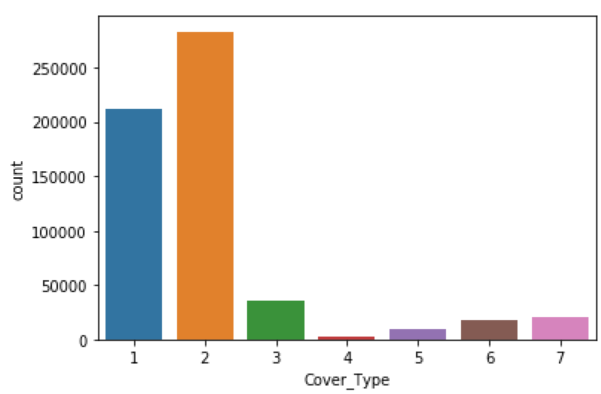
We found the Forest Cover type dataset in the UCI Machine Learning Repository that takes forestry data from four wilderness areas in Roosevelt National Forest in northern Colorado (https://archive.ics.uci.edu/ml/datasets/Covertype). This data is obtained from the US Geological Survey (USGS) and the US Forest Service (USFS) and provided by Machine Learning Laboratory of University of California Irvine. The original database owners are Jock A. Blackard, Dr. Denis J. Dean, and Dr. Charles W. Anderson of the Remote Sensing and GIS Program at Colorado State University. This problem is also part of a Kaggle competition.

3 Exploratory Data Analysis

We start with exploratory analysis of data to understand the structure of the data and to find the relationship between attributes and class labels. We use 10 most important attributes for our study. The following are the attributes that we will consider in your study:

1. Elevation
2. Aspect
3. Slope
4. Horizontal Distance To Hydrology
5. Vertical Distance To Hydrology
6. Horizontal Distance To Roadways
7. Hillshade 9am
8. Hillshade Noon
9. Hillshade 3pm
10. Horizontal Distance To Fire Points

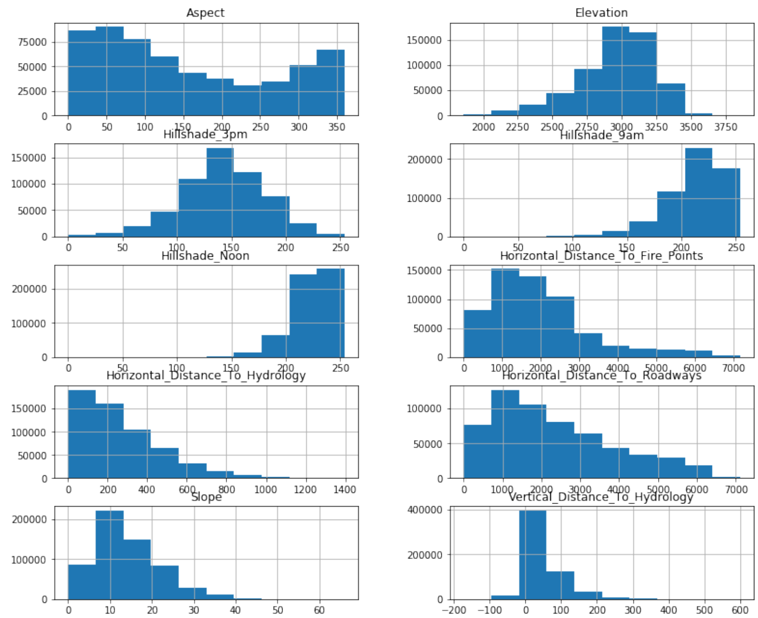
COUNT OF COVER TYPE



The cover types arranged in descending order of their count is as below –
Cover type 2 > Cover Type 1 > Cover Type 3 > Cover Type 7 > Cover Type 6 > Cover Type 5 > Cover Type 4.

This shows that count of Lodgepole Pine and Spruce/Fir cover type is highest. Cottonwood/Willow has the least count among all the cover types.

DISTRIBUTION OF FEATURES



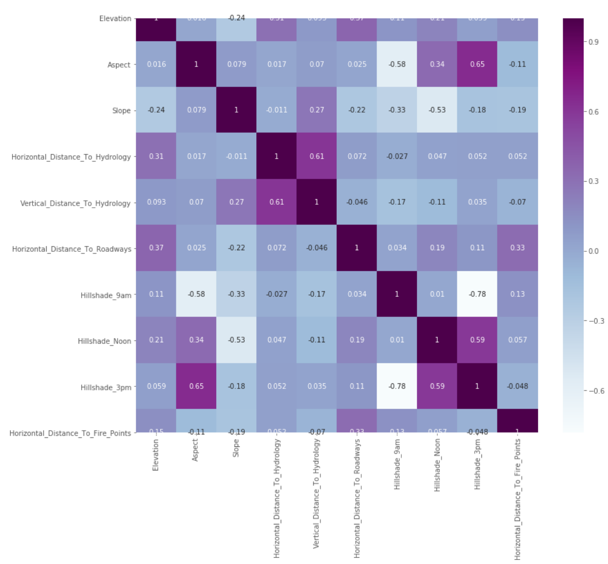
The plot shows the histograms of 10 numerical attributes. We notice that all attributes have different numerical ranges. This means that values are not scaled. Therefore, we need to scale and normalize the data to perform machine learning algorithms. We will use this data on different models to compare model accuracies. The detailed description of each feature is given below.

- Feature “Aspect” exhibits a double-peaked distribution. The first peak is close to 50 and the second peak is close to 350.
- Feature “Elevation” exhibits left skewed distribution, with a mean close to 3000.
- Feature “Hill_shade_3pm” exhibits roughly a normal distribution with a mean close to 150.
- Feature “Hill_shade_9am” exhibits a left skewed edge peaked distribution with a mean close to 225.
- Feature “Hill shade Noon” exhibits a left skewed edge peaked distribution with a mean close to 225. It appears to have a narrow spread with majority of points distributed between 200 and 250.
- Feature “Horizontal Distance to Fire Points” exhibits a right skewed distribution with mean close to 2000.
- Feature “Horizontal Distance to Hydrology” exhibits a right skewed edge peaked distribution with a mean close to 300.

- Feature “Horizontal Distance to Roadways” exhibits a right skewed distribution with a mean close to 2500.
- Feature “Slope” exhibits a right skewed distribution with a mean close to 15.
- Feature “Vertical Distance to Hydrology” exhibits a roughly normal distribution. It appears to have a narrow spread with majority of points distributed between -10 to 60.

HEAT MAP

We have used Heatmap to show the correlation between the attributes. Heatmap gives us the Pearson correlation for all attributes of our dataset. Pearson correlation returns coefficient values between -1 and 1. If the correlation between two features is 0, then this means that the features do not have any correlation between them. If the correlation between two features is greater than 0, then it means that the features have positive correlation. If the correlation between two features is less than 0 it means that the features have negative correlation. If the correlation coefficient value goes towards 0, the relationship between the two variables becomes weak. We can keep the the highly correlated variables and drop the others.



The above heat map exhibits strong correlation between the following features –

1. Hillshade_9am and Hillshade_3pm show a strong negative correlation with a correlation coefficient of -0.78.

2. Hillshade_3pm and Aspect exhibit a strong positive correlation with a correlation coefficient of 0.65.

3. Horizintal_Distance_To_Hydrology and Vertical_Distance_To_Hydrology exhibit a strong correlation with correlation coefficient of 0.61

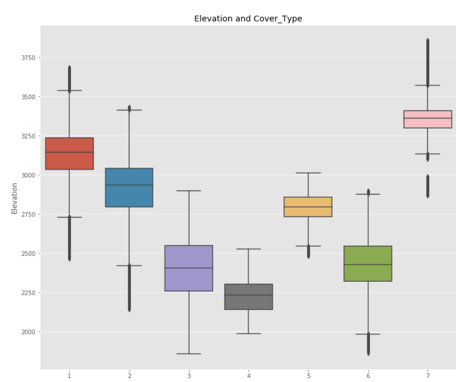
4. Hillshade_3pm and Hillshade_Noon exhibit a strong correlation with correlation coefficient of 0.59.

All other features appear to have weak to insignificant correlation amongst themselves with their coefficients ranging between -0.5 to 0.5.

BOXPLOTS

In order to understand the relationship between the attributes and class labels better, we have plotted them with different box plots. Below are the box plots for each attribute:

ELEVATION AND COVER TYPE



Median:

The cover types arranged in descending order of median elevation is as below – Cover type 7 > Cover Type 1 > Cover Type 2 > Cover Type 5 > Cover Type 6 > Cover Type 3 > Cover Type 4

Range:

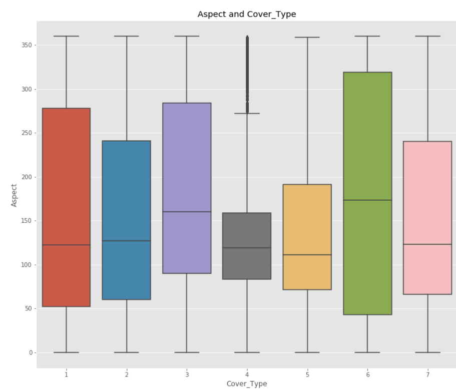
The cover types arranged in descending order of median elevation is as below – Cover type 3 > Cover Type 2 > Cover Type 6 > Cover Type 1 > Cover Type 4 > Cover Type 5 > Cover Type 7

Outliers:

Cover types 1, 2, 6, and 7 exhibit outliers on either extremes. While cover type 5 exhibit outliers on the lower extreme. Cover types 3 and 4 do not exhibit any outliers.

Cover types with top skew:	None
Cover types with bottom skew:	None
Cover types with insignificant skew:	1, 2, 3, 4, 5, 6, and 7

ASPECT AND COVER TYPE



Median:

The cover types arranged in descending order of median aspect is as below – Cover type 6 > Cover Type 3 > Cover Type 2 > Cover Type 1 > Cover Type 7 > Cover Type 4 > Cover Type 5

Range:

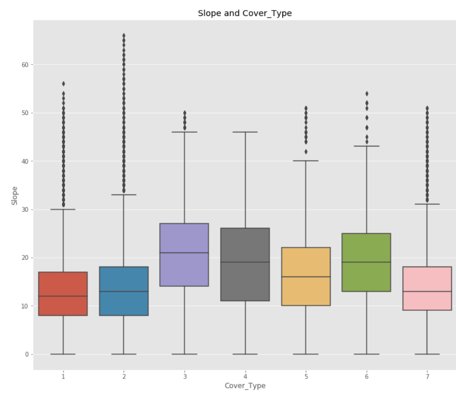
The cover types arranged in descending order of range of aspect is as below – Cover type 6 > Cover Type 1 > Cover Type 3 > Cover Type 2 > Cover Type 7 > Cover Type 5 > Cover Type 4

Outliers:

Only cover types 4 exhibit outliers on the upper extreme. While other cover types do not exhibit any outliers.

Cover types with top skew:	1, 2, 4, 5, and 7
Cover types with bottom skew:	3
Cover types with insignificant skew:	6

SLOPE AND COVER TYPE



Median:

The cover types arranged in descending order of median slope is as below – Cover type 3 > Cover Type 4 > Cover Type 6 > Cover Type 5 > Cover Type 2 > Cover Type 7 > Cover Type 1

Range:

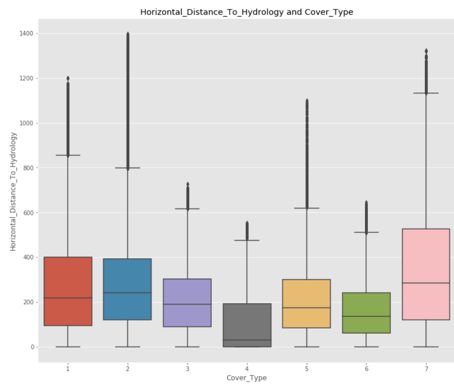
The cover types arranged in descending order of range of slope is as below – Cover type 4 > Cover Type 3> Cover Type 6 > Cover Type 5 > Cover Type 2 > Cover Type 7 > Cover Type 1

Outliers:

Only cover types 4 do not exhibit any outliers. While other cover types exhibit outliers on the upper extreme.

Cover types with top skew:	1, 2, 3, 4, 5, 6, and 7
Cover types with bottom skew:	None
Cover types with insignificant skew:	None

HORIZONTAL DISTANCE TO HYDROLOGY AND COVER TYPE



Median:

The cover types arranged in descending order of median Horizintal_Distance_To_Hydrology is as below – Cover type 7 > Cover Type 2 > Cover Type 1 > Cover Type 3 > Cover Type 5 > Cover Type 6 > Cover Type 4

Range:

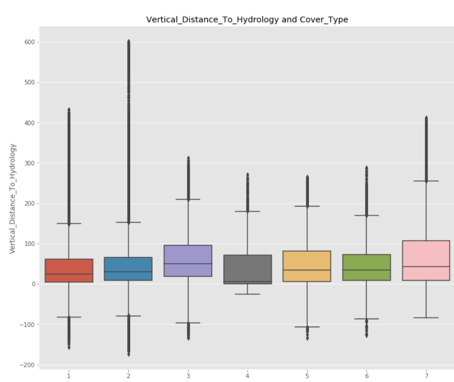
The cover types arranged in descending order of range of Horizintal_Distance_To_Hydrology is as below – Cover type 7 > Cover Type 1 > Cover Type 2 > Cover Type 5 > Cover Type 3 > Cover Type 6 > Cover Type 4

Outliers:

All the cover types exhibit an outlier in the upper extreme. 25% of the data for cover type 4 have zero distance to hydrology, so it doesn't have a bottom whisker.

Cover types with top skew:	1, 2, 3, 4, 5, 6, and 7
Cover types with bottom skew:	None
Cover types with insignificant skew:	None

VERTICAL DISTANCE TO HYDROLOGY AND COVER TYPE



Median:

The cover types arranged in descending order of median Vertical_Distance_To_Hydrology is as below – Cover type 3 > Cover Type 7 > Cover Type 6 > Cover Type 5 > Cover Type 2 > Cover Type 1 > Cover Type 4

Range:

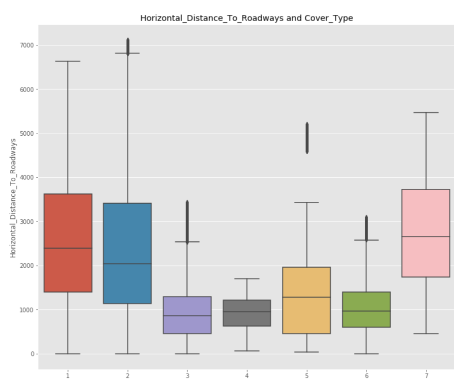
The cover types arranged in descending order of range of Vertical_Distance_To_Hydrology is as below – Cover type 7 > Cover Type 3 > Cover Type 5 > Cover Type 6 > Cover Type 2 > Cover Type 1 > Cover Type 4

Outliers:

Cover types 4 and 7 exhibit outliers on the upper extreme. While other cover types exhibit outliers on either extreme.

Cover types with top skew:	7
Cover types with bottom skew:	3
Cover types with insignificant skew:	1, 2, 4, 5, and 6

HORIZONTAL DISTANCE TO ROADWAYS AND COVER TYPE



Median:

The cover types arranged in descending order of median Horizontal_distance_to_Roadways is as below – Cover type 7 > Cover Type 1 > Cover Type 2 > Cover Type 5 > Cover Type 6 > Cover Type 4 > Cover Type 3

Range:

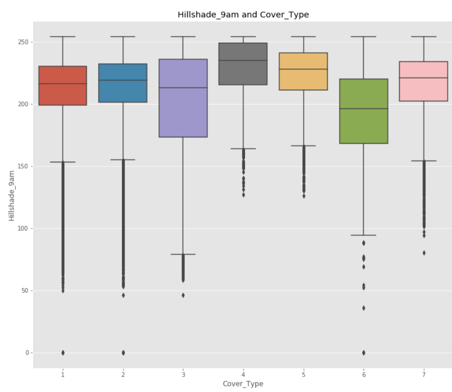
The cover types arranged in descending order of range of Horizontal_distance_to_Roadways is as below – Cover type 2 > Cover Type 1 > Cover Type 7 > Cover Type 5 > Cover Type 6 > Cover Type 3 > Cover Type 4

Outliers:

Cover types 2, 3, 5, and 6 exhibit outliers on the upper extreme. While the cover types 1, 4, and 7 do not exhibit any outliers.

Cover types with top skew:	1, 2, 3, 5, 6, and 7
Cover types with bottom skew:	None
Cover types with insignificant skew:	4

HILLSHADE 9AM AND COVER TYPE



Median:

The cover types arranged in descending order of median Hillshade_9am is as below – Cover type 4 > Cover Type 5 > Cover Type 7 > Cover Type 2 > Cover Type 1 > Cover Type 3 > Cover Type 6

Range:

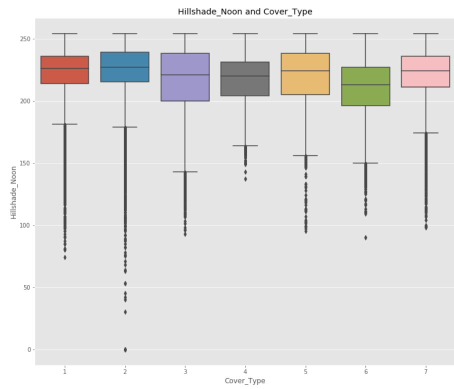
The cover types arranged in descending order of range of Hillshade_9am is as below – Cover type 3 > Cover Type 6 > Cover Type 1 > Cover Type 7 > Cover Type 2 > Cover Type 4 > Cover Type 5

Outliers:

All cover types exhibit outliers on the lower extreme.

Cover types with top skew:	None
Cover types with bottom skew:	1, 2, 3, 4, 5, 6, and 7
Cover types with insignificant skew:	None

HILLSHADE NOON AND COVER TYPE



Median:

The cover types arranged in descending order of median Hillshade_Noon is as below – Cover type 2 > Cover Type 1 > Cover Type 7 > Cover Type 5 > Cover Type 3 > Cover Type 4 > Cover Type 6

Range:

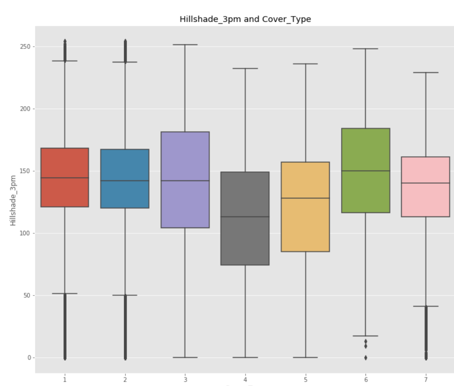
The cover types arranged in descending order of range of Hillshade_Noon is as below – Cover type 3 > Cover Type 6 > Cover Type 5 > Cover Type 4 > Cover Type 7 > Cover Type 2 > Cover Type 1

Outliers:

All cover types exhibit outliers on the lower extreme.

Cover types with top skew:	None
Cover types with bottom skew:	1, 2, 3, 4, 5, 6, and 7
Cover types with insignificant skew:	None

HILLSHADE 3 PM AND COVER TYPE



Median:

The cover types arranged in descending order of median Hillshade_3pm is as below – Cover type 6 > Cover Type 1 > Cover Type 2 > Cover Type 3 > Cover Type 7 > Cover Type 5 > Cover Type 4

Range:

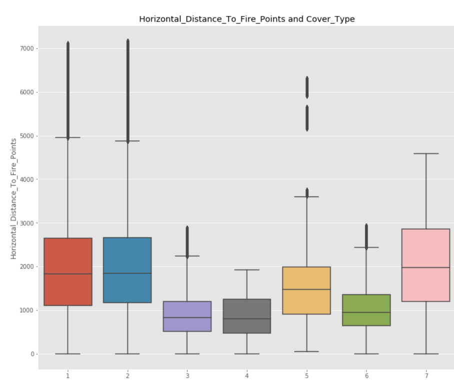
The cover types arranged in descending order of range of Hillshade_3pm is as below – Cover type 3 > Cover Type 5 > Cover Type 4 > Cover Type 6 > Cover Type 1 > Cover Type 2 > Cover Type 7

Outliers:

Cover types 1 and 2 exhibit outliers on both extreme. Cover types 6 and 7 exhibit outliers on the bottom extreme. While the cover types 3, 4, and 5 do not exhibit any outliers.

Cover types with top skew:	None
Cover types with bottom skew:	3, 5, and 6
Cover types with insignificant skew:	1, 2, 4, and 7

HORIZONTAL DISTANCE TO FIRE POINTS AND COVER TYPE



Median:

The cover types arranged in descending order of median Horizontal_Distance_Fire_Points is as below – Cover type 7 > Cover Type 2 > Cover Type 1 > Cover Type 5 > Cover Type 6 > Cover Type 3 > Cover Type 4

Range:

The cover types arranged in descending order of range of Horizontal_Distance_Fire_Points is as below – Cover type 1 > Cover Type 2 > Cover Type 7 > Cover Type 5 > Cover Type 6 > Cover Type 3 > Cover Type 4

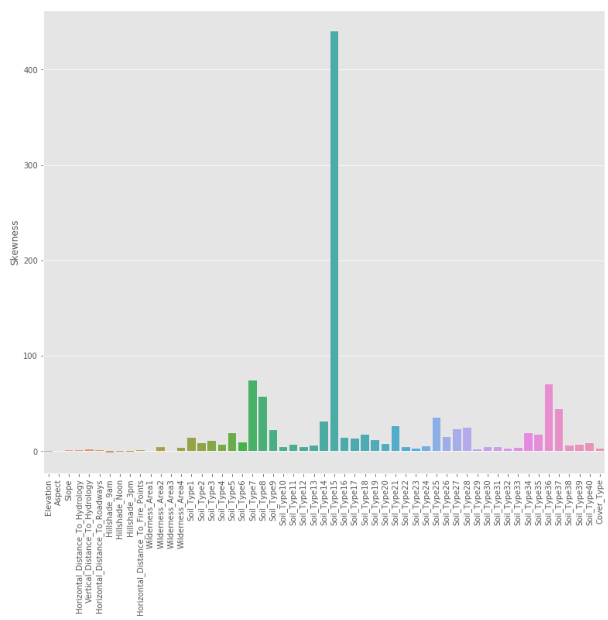
Outliers:

Cover types 1, 2, 3, 5, and 6 exhibit outliers on the upper extreme. While cover types 4 and 7 do not exhibit any outliers.

Cover types with top skew:	1,2, 3, 5, 6, and 7
Cover types with bottom skew:	None
Cover types with insignificant skew:	4

SKEWNESS

The skewness for a normal distribution is zero, and any symmetric data should have a skewness near zero. We have plotted the skewness for each attribute of the dataset:



In the above plot, we observe several attributes have skewness zero. Negative values for the skewness indicate data that those attributes are skewed left and positive values for the skewness indicate data that those attributes are skewed right.

4 Pre-processing

4.1 Standard Normalization

Our data is not scaled here. Hence, we need to standardize our data. Standardization of a dataset is very important here as our machine learning estimators

will behave badly if the individual features do not look like standard normally distributed data. It is especially important to scale data for estimators. If a feature has a variance that has orders of magnitude larger that others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.
We use 'StandardScalar' function from 'sklearn.preprocessing' library to scale all the values. It standardizes features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as:

$$z = \frac{(x-u)}{s}$$

Here, u is the mean of the training samples and s is the standard deviation of the training samples.
The function ensures that centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using the transform method.

4.2 Dimensionality Reduction - PCA

We tried dimensionality reduction technique (Principal Component Analysis) on the ten features after scaling. We noted the variance for each principal component. We observed that reducing the dimensions do not yield good result as the accuracy decreases. The reason for this is that attributes here are independent. Therefore, we decided to take all the ten features for applying various models to predict forest cover type. The following table shows the proportion of variance along the 10 principal component vectors. The tables suggests that variance is distributed along all the attributes.

Name	Standard Deviation	Proportion of Variance
PC1	1.62	0.24
PC2	1.55	0.22
PC3	1.32	0.16
PC4	1.12	0.11
PC5	0.87	0.07
PC6	0.78	0.05
PC7	0.68	0.04
PC8	0.68	0.04
PC9	0.59	0.03
PC10	0.55	0.03

Table 3: Variance distributed along each principal component vector

5 Methodology

Methods that we used in our analysis are :

1. Random Forest Classifier- (RFC)
2. Multi-class Logistic Regression - (LR)
3. Linear Discriminant analysis - (LDA)
4. Gaussian Naive Bayes - (GN)
5. AdaBoost - (AD)
6. Decision Tree - (DT)
7. Bagging - (BG)
8. Gradient Boosting - (GC)
9. K-Nearest Neighbors - (KNN)
10. Neural Network - (NN)

5.1 Random Forest

It is an ensembling technique which fits number of decision tree classifier on sub-samples of dataset. Random Forest works on idea that a large number of relatively uncorrelated models(trees) operating as a committee will outperform any of the individual constituent models. Random Forest uses bootstrap re-sampling. It introduces randomization at two stages. First, the number of attributes considered to build a weak learner and second, the number of features considered for split at a node. The main advantage of this technique is it avoids over-fitting.

Random Forest Algorithm

For iteration b = 1 to B:

1. Draw a bootstrap sample $\mathbb{Z}^*$ of size N from the training data.
2. Grow a random-forest tree T_b to the bootstrapped data, by re-cursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
   (a) Select m variables at random from the p variables.
   (b) Pick the best variable/split-point among the m.
   (c) Split the node into two daughter nodes.
3. Output the ensemble of trees T_b.
4. To make classification prediction at a new point x:
   Let $\hat{C_b}(x)$ be the class prediction of the b$^{\text{th}}$ random-forest tree. Take majority vote of all class predictions.

5.2 Logistic Regression

Logistic regression uses sigmoid and logistic activation function. Here, we use logistic regression extended for k classes. Multiple binary classifiers are trained for each class when solving multiple class classification problem. To arrive at the multinomial logit model, for K possible outcomes, we run K-1 independent binary logistic regression models, in which one outcome is chosen as a "pivot" and then the other K-1 outcomes are separately regressed against the pivot outcome.

1. We use multi-response logit link, comparing each class to baseline. Choosing last class as baseline and making $k1$ logit ratios

$$log \frac{p_k(x)}{p_K(x)} = \beta_{k0} + \beta_k'(x), k = 1, ...K-1$$

2. We get probability

$$P(Y = K \mid X = x) = \frac{1}{1 + \Sigma_{i=1}^{K-1} e^{\beta_{k0} + \beta_1' X}}$$

3. We can use this probabilty to find other probabilities

$$P(Y = k \mid X = x) = \frac{e^{\beta_{k0} + \beta_1' X}}{1 + \Sigma_{i=1}^{K-1} e^{\beta_{k0} + \beta_1' X}}$$

4. This results in $(p+1)(K-1)$ parameters and for convenience we call the parameters β.

5. Then $\hat{G}(x) = argmax_k \hat{p_k}(x)$

5.3 Linear Discriminant Analysis

In LDA, we need to find a new feature space to project the data in order to maximize classes separability. In order to maximize classes separability we maximize the distance between the mean of each class and minimize the spreading within the class itself. However, this formulation is only possible if we assume that the dataset has a Normal distribution. LDA uses Bayes' Theorem to estimate the probabilities. If the output class is (k) and the input is (x), here is how Bayes' theorem works to estimate the probability that the data belongs to each class:

$$P(Y = k \mid X = x) = \frac{f_k(x)\pi_k}{\Sigma_{i=1}^{K} f_1(k)\pi_1}$$

(i) π_k is the prior probability of the class k.

(ii) $f_k(x)$ is the density/mass on x given k called the likelihood of the data x given the parameter k.

(iii) $P(Y = k \mid X = x)$ is the posterior probability of class k.

5.4 Gaussian - Naive Bayes

Naive Bayes is a conditional probability model. It computes the conditional probability of a sample observation belonging to a particular class based on the observed data and the prior belief.It is called naive Bayes or idiot Bayes because the calculation of the probabilities for each hypothesis are simplified to make their calculation tractable. Using Bayes' theorem, the conditional probability can be decomposed as:

$$P(y = k \mid x) = \frac{p(y)p(x|y)}{p(x)}$$

When the classification is multivariate, we need to find the the class y with maximum probability.

$$y = argmax_y P(y) \pi_{i=1}^{n} P(x_i \mid y)$$

Naive Bayes can be extended to real-valued attributes assuming a Gaussian distribution. This extension of naive Bayes is called Gaussian Naive Bayes. The formula for conditional probability changes to

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2})$$

5.5 AdaBoost

AdaBoost focuses on classification problems and aims to convert a set of weak classifiers into a strong one. It performs extra tuning on weak classifiers and adjusts weight to yield best outcome. The AdaBoost algorithm begins by training a decision tree by assigning an equal weight to each observation. After evaluating the first tree, we increase the weight of those observations that are difficult to classify and lower the weights for those that are easy to classify. The second tree is therefore grown on this weighted data. Here, the idea is to improve upon the predictions of the first tree. Our new model is therefore Tree 1 + Tree 2. We then compute the classification error from this new 2-tree ensemble model and grow a third tree to predict the revised residuals. We repeat this process for a specified number of iterations. Subsequent trees help us to classify observations that are not well classified by the previous trees. Predictions of the final ensemble model is therefore the weighted sum of the predictions made by the previous tree models. The final equation for classification can be represented as

$$G(x) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$$

Here, G_m stands for the m$^{\text{th}}$ weak classifier. α_m is the corresponding weight which is computed by the boosting algorithm. AdaBoost is very fast and easy to implement. However, it is very sensitive to noise data, if the weak classifiers are too weak or too complex, then there is possibility of over-fitting the training data.

AdaBoost Algorithm

Consider a two-class problem, with the output variable coded as $Y \in \{-1, 1\}$. The weight for each data point is initialized as :

$$w_i = \frac{1}{N}, \ \ i = 1, 2, ..., N$$

For iteration m = 1 to M:

1. Fit weak classifier $G_m(x)$ to the dataset and select the one with the lowest weighted classification error:

$$err_m = \frac{\Sigma_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\Sigma_{i=1}^{N} w_i}$$

2. Calculate the weight for the mth weak classifier:

$$\alpha_m = \frac{log(1 - err_m)}{err_m}$$

3. Update the weight for each data point as:

$$w_i = w_i.exp[\alpha_m.I(y_i \neq G_m(x_i))]$$

   Observations misclassified by $G_m(x)$ have their weights scaled by a factor $exp(\alpha_m)$, increasing their relative influence for inducing the next classifier $G_{m+1}(x)$ in the sequence. After M iteration we get the final prediction by summing up the weighted prediction of each classifier.

5.6 Decision Tree

A decision tree is a flow-chart-like structure, where each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of a test, and each leaf (or terminal) node holds a class label. The topmost node in a tree is the root node. Decision tree partitions the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one. They are conceptually simple yet powerful.

Decision tree- Classification Algorithm

1. Fit piecewise-constant model to a feature space that has been partitioned into subsets.

2. Partitioning takes place recursively

   (i) Define loss function.

   (ii) Look for best splits within each variable such at loss is reduced.

   (iii) Repeat until some stopping criterion.

(iv) Prune tree back to 'optimal tree'.

(v) Assign a value to each terminal node.

Decision Tree trains very fast and works good with categorical variables. However since only one tree is fit and if the tree goes too deep, it might overfit the training data.

5.7 Bagging

Bagging is an ensemble technique which is used to reduce the variance of classifiers with high variability. It fits base classifiers each on random subsets of the original dataset and then aggregates their individual predictions (either by voting or by averaging) to form the final prediction.

Bagging Algorithm

1. Classifier generation:
   Let N be the size of the training set. For each of M iterations:
   (a) Sample N instances with replacement from the original training set.
   (b) Apply the learning algorithm to the sample.
   (c) Store the resulting classifier.
2. Classification:
   For each of the M classifiers:
   (a) Predict class of instance using classifier.
3. Return class that was predicted most often.

Bagging is most effective for highly non linear classifiers such as decision trees. It reduces overfitting (variance) by averaging or voting, however, this leads to an increase in bias and reduction in variance.

5.8 Gradient Boosting

Gradient Boosting trains many models in a gradual, additive and sequential manner. Gradient Boosting identifies the shortcomings of weak learners by using gradients in the loss function. The loss function is a measure of indicating how good are model's coefficients at fitting the underlying data.

Gradient Boosting Algorithm

Suppose we want to fit the forward stagewise additive model with a general loss function L with basis functions being trees.

1. Each tree can be expressed as $T(x;\Theta)$

2. Boosted tree model is a sum of such trees:

$$f_M(x) = \Sigma_{i=1}^{M} T(x;\Theta_m)$$

3. In the forward stagewise procedure we need to solve

$$\hat{\Theta}_m = \underset{\Theta}{argmin} \Sigma_{i=1}^{M} L(y_1, f_{m-1}(x) + T(x_i;\Theta_m))$$

Gradient boosting algorithm fits single decision tree at each iteration. Instead of averaging over all the trees, GB tries to find the best linear combination of fitted trees to explain the training data. As a result of this optimization, the Gradient Boosting model training is much slower and can overfit the training data. However, it can yield better results.

5.9 K- Nearest Neighbors

The k nearest neighbors algorithm is a clustering technique which assumes that similar things exist in close proximity. In other words, similar things are near to each other. It uses Euclidean distance to find the k nearest neighbors to every vector. Then, it finds the k nearest neighbors for every test vector during prediction and uses the majority vote to classify the test vector. Though implementing KNN is very simple, choosing the right value for k is critical. Also, KNN struggles in high dimensions.

K- Nearest Neighbors

(i) Given a point x_0, find the k training points x_m, r=1,...m closest in distance to x_0

(ii) Classify using majority vote among the m neighbors.

Despite its simplicity, k-nearest-neighbors has been successful in a large number of classification problems. It is often successful where each class has many possible prototypes, and the decision boundary is very irregular.

5.10 Neural Network

Neural network takes inspiration from the learning process occurring in human brains. They consists of an artificial network of functions, called parameters, which allows the computer to learn, and to fine tune itself, by analyzing new data. Each parameter, sometimes also referred to as neurons, is a function which produces an output, after receiving one or multiple inputs. Those outputs are then passed to the next layer of neurons, which use them as inputs of their own function, and produce further outputs. Those outputs are then passed on to the next layer of neurons, and so it continues until every layer of neurons have been considered, and the terminal neurons have received their input. Those terminal neurons then output the final result for the model.

Neural Network works good on data with non-linear relationship. It usually yields good result but it takes a long time to fit and it is very difficult to interpret the model. It requires a large number of training data in order to get good result and it estimates a large number of parameters.

6 Evaluation

Training and Testing data

We have split our data into training and testing datasets. We use the 'train_test_split' function to split the data into training and testing data. We specify the train size as 0.20 as we aim to put 20% of the data into our training set, and the rest of the data into the test set. The advantage of using this function is that the splitting do not follow the ascending order. The function by default ignores the original order of data. It randomly picks data to form the training and test set, which is what we needed.

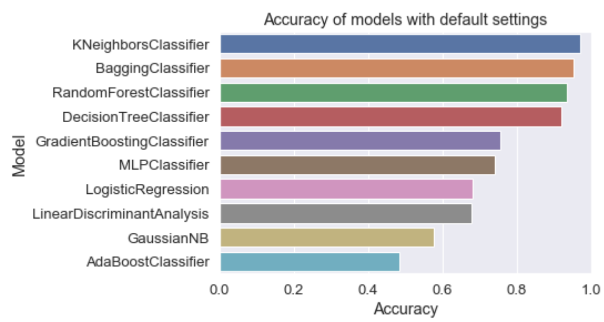
Model Comparison criterion

We train the dataset on the following models. Following are the accuracies obtained after using the models to perform prediction on the testing data. Table 4 shows the accuracies obtained with respect to default settings for each model.

S.No.	Model	Accuracy
1	K - Neighbor Classifier	0.969304
2	Bagging Classifier	0.953383
3	Random Forest Classifier	0.935535
4	Decision Tree Classifier	0.919486
5	Gradient Boosting Classifier	0.755015
6	MLP Classifier	0.741014
7	Logistic Regression	0.680189
8	Linear Discriminant Analysis	0.678976
9	Gaussian Naive Bayes	0.575803
10	Ada Boost Classifier	0.482500

Table 4: Accuracy for models

Below is the bar plot representation of the table.

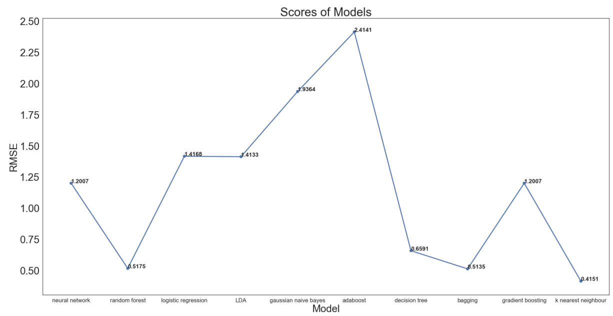


K-Fold Cross validation

Cross-validation is a resampling procedure. We use it to evaluate machine learning model. Initially, we choose a parameter k which is the number of groups that a given dataset has to be split into. The procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 is called 10-fold cross-validation.
The method is popular because it results in a less biased or less optimistic estimate of the model performance than other methods, such as a simple train/test split.The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups.
3. For each unique group:
   (i) Take the group as test data set.
   (ii) Take the remaining groups as a training data set.
   (iii) Fit the model on the training set and evaluate it on the test set.
   (iv) Retain the evaluation score and discard the model.
4. Summarize the performance of the model using the sample of model evaluation scores.

In brief Kfold splits the data into k folds, trains on the k-1 folds and tests on the the remaining group.
We use the Kfold function present in 'sklearn.model_selection' library to perform Kfold cross validation. We use Kfold cross validation to find out about model which has the least mean square error. We specify K=10 and get root mean square error for each iteration for each model and then take the mean. Following plot shows the 'Root mean square error' for each model.



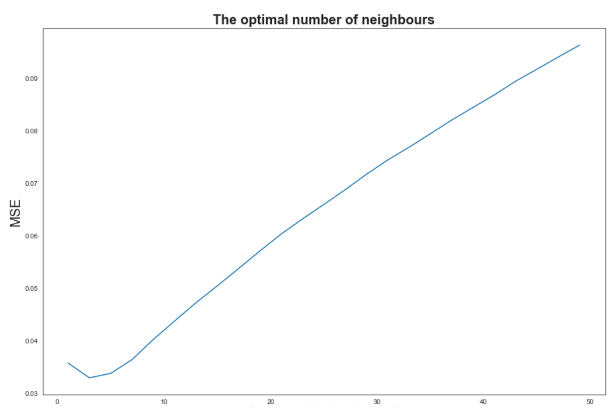
From the graph we can observe that KNN has least mean square error. So we choose KNN as our best model for classification and tune the parameters.

Tuning parameter selection

K refers to the nearest neighbors. The best K is the one that corresponds to the lowest test error rate, so repeated measurements are carried out of the test error for different values of K. Below is the procedure for finding out the optimal number of neighbors.

1. Initially consider a list L ranging from 1 to 50.
2. For each n in L:
   (i) Model the KNN classifier.
   (ii) Perform cross validation by taking K=10 and calculating score for each split.
   (iii) Obtain the accuracy by taking the mean of the K scores.
3. Calculate misclassification error.

We used cross validation in order to find number of neighbors which gives least mean square error when making predictions. From the graph we can see that optimal number of neighbors is 3. Hence, we train the KNN model taking n=3.



CONFUSION MATRIX OF KNN :

We get an accuracy of 96.96% for KNN classifer. We obtain the following confusion matrix after training and testing the KNN model.

From the confusion Matrix, it is evident that Class 1 is classified most accurately, followed by Class 0.

7 Conclusion

After analysing the data, we found that among 54 attributes only 10 of them were useful. After applying several models on the training set, we observe that best classifier for our dataset is KNN. We tested this result using Kfold cross validation. KNN outperformed all models by obtaining an accuracy of 96.96%. Hence, we conclude that KNN is the best model to predict forest cover type of a given region.

8 Future Work

1. Apply advanced classifiers as extremely Randomized tree.
2. Two-way classification approach to distinguish between majority class labels that have minority
3. Apply semi-supervised learning
4. Optimize the running time for Adaboost and bagging
5. Apply SVM. We tried applying SVM but it took a lot of time to run.

9 Contribution

1. Rudrani Bhadra: Data Analysis, Classification, Report writing
2. Lipsa Mishra: Data Analysis, Classification, Report writing
3. Lavanya Sharma: Data Analysis, Classification, Report writing

10 References

1. MachineLearningMastery (2019) k-Fold Cross-Validation , Retrieved on 21st November, 2019 from https://machinelearningmastery.com
2. Kevin's Blog (2016) A Complete Guide to K-Nearest-Neighbors with Applications in Python and R, Retrieved on 21st November, 2019 from https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/#parameter-tuning-with-cross-validation
3. TowardsDataScience (2017) Boosting algorithm: AdaBoost, Retrieved on 21st November, 2019 from https://towardsdatascience.com/boosting-algorithm-adaboost-b6737a9ee60c
4. Kaggle: The Home of Data Science. (2015). Description - Forest Cover Type Prediction, Kaggle. Retrieved on Dec 7th, 2015 from: https://www.kaggle.com/c/forest-cover-type-prediction
5. TowardsDataScience (2019) Random Forest: Understanding Random Forest, Retrieved on 21st November, 2019 from https://towardsdatascience.com/understanding-random-forest-58381e0602d2
6. Wikipedia (2019) Logistic Regression: Multinomial logistic regression, Retrieved on 21st November, 2019 from https://en.wikipedia.org/wiki/Multinomial_logistic_regression
7. TowardsDataScience (2019) Naive Bayes: naive-bayes-for-machine-learning, Retrieved on 21st November, 2019 from https://machinelearningmastery.com/naive-bayes-for-machine-learning/

11 Appendix

Data

Important Variables

S.No.	Name	Data Type	Mean	Std-Deviation
1	Elevation	quantitative	2959.365301	279.984734
2	Aspect	quantitative	155.656807	111.913721
3	Slope	quantitative	14.103704	7.488242
4	Horizontal Distance To Hydrology	quantitative	269.428217	212.549356
5	Vertical Distance To Hydrology	quantitative	46.418855	58.295232
6	Horizontal Distance To Roadways	quantitative	2350.146611	1559.254870
7	Hillshade 9am	quantitative	212.146049	26.769889
8	Hillshade Noon	quantitative	223.318716	19.768697
9	Hillshade 3pm	quantitative	142.528263	38.274529
10	Horizontal Distance To Fire Points	quantitative	1980.291226	1324.195210

Table 5: Attributes of Forest Cover Type Dataset

Origin of Data

We found the Forest Cover type dataset in the UCI Machine Learning Repository that takes forestry data from four wilderness areas in Roosevelt National Forest in northern Colorado (https://archive.ics.uci.edu/ml/datasets/Covertype). This data is obtained from the US Geological Survey (USGS) and the US Forest Service (USFS) and provided by Machine Learning Laboratory of University of California Irvine. The original database owners are Jock A. Blackard, Dr. Denis J. Dean, and Dr. Charles W. Anderson of the Remote Sensing and GIS Program at Colorado State University. This problem is also part of a Kaggle competition.

Literature Review

Forest Cover Type Dataset was a part of the Kaggle Competition, a few year ago. In this competition contestants asked to predict the forest cover type (the predominant kind of tree cover) from strictly cartographic variables (as opposed to remotely sensed data). The actual forest cover type for a given 30 x 30 meter cell was determined from US Forest Service (USFS) Region 2 Resource Information System data. Independent variables were then derived from data obtained from the US Geological Survey and USFS. The data is in raw form (not scaled) and contains binary columns of data for qualitative independent variables such as wilderness areas and soil type. From then, few people worked on this dataset, applying different classification models. The recent one that was on Kaggle was based on Random Forest Classifier with 96% accuracy.

Modeling details

We tried to use Quadratic Discriminant Analysis and SVM for the dataset but QDA gave extremely low accuracy (around 10%) and SVM took too long to run. The models which gave average accuracy are neural networks, gradient boosting classifier, logistic regression, linear discriminant analysis. Gaussian naive bayes and adaboost classifier performed relatively worse than the others

EDA

December 1, 2019

```
In [1]: import pandas as pd
        import numpy as np
        import scipy as sp
        import seaborn as sb
        import matplotlib.pyplot as plt

        cov = pd.read_csv('/home/lipsa/Desktop/Coursework Fall 2019/STAT 841/covtype.csv')
        cov.head()
```

```
Out[1]:    Elevation  Aspect  Slope  Horizontal_Distance_To_Hydrology  \
        0       2596      51      3                               258
        1       2590      56      2                               212
        2       2804     139      9                               268
        3       2785     155     18                               242
        4       2595      45      2                               153

           Vertical_Distance_To_Hydrology  Horizontal_Distance_To_Roadways  \
        0                               0                              510
        1                              -6                              390
        2                              65                             3180
        3                             118                             3090
        4                              -1                              391

           Hillshade_9am  Hillshade_Noon  Hillshade_3pm  \
        0            221             232            148
        1            220             235            151
        2            234             238            135
        3            238             238            122
        4            220             234            150

           Horizontal_Distance_To_Fire_Points  ...  Soil_Type32  Soil_Type33  \
        0                                6279  ...            0            0
        1                                6225  ...            0            0
        2                                6121  ...            0            0
        3                                6211  ...            0            0
        4                                6172  ...            0            0

           Soil_Type34  Soil_Type35  Soil_Type36  Soil_Type37  Soil_Type38  \
```

```
0          0          0          0          0          0
1          0          0          0          0          0
2          0          0          0          0          0
3          0          0          0          0          0
4          0          0          0          0          0

   Soil_Type39  Soil_Type40  Cover_Type
0            0            0           5
1            0            0           5
2            0            0           2
3            0            0           2
4            0            0           5

[5 rows x 55 columns]
```

In [2]:
```
print(cov.columns)
```

```
Index(['Elevation', 'Aspect', 'Slope', 'Horizontal_Distance_To_Hydrology',
       'Vertical_Distance_To_Hydrology', 'Horizontal_Distance_To_Roadways',
       'Hillshade_9am', 'Hillshade_Noon', 'Hillshade_3pm',
       'Horizontal_Distance_To_Fire_Points', 'Wilderness_Area1',
       'Wilderness_Area2', 'Wilderness_Area3', 'Wilderness_Area4',
       'Soil_Type1', 'Soil_Type2', 'Soil_Type3', 'Soil_Type4', 'Soil_Type5',
       'Soil_Type6', 'Soil_Type7', 'Soil_Type8', 'Soil_Type9', 'Soil_Type10',
       'Soil_Type11', 'Soil_Type12', 'Soil_Type13', 'Soil_Type14',
       'Soil_Type15', 'Soil_Type16', 'Soil_Type17', 'Soil_Type18',
       'Soil_Type19', 'Soil_Type20', 'Soil_Type21', 'Soil_Type22',
       'Soil_Type23', 'Soil_Type24', 'Soil_Type25', 'Soil_Type26',
       'Soil_Type27', 'Soil_Type28', 'Soil_Type29', 'Soil_Type30',
       'Soil_Type31', 'Soil_Type32', 'Soil_Type33', 'Soil_Type34',
       'Soil_Type35', 'Soil_Type36', 'Soil_Type37', 'Soil_Type38',
       'Soil_Type39', 'Soil_Type40', 'Cover_Type'],
      dtype='object')
```

In [3]:
```
#shape of data
cov.shape
```

Out[3]:
```
(581012, 55)
```

In [4]:
```
#check mising values
print(list(cov.isnull().any()))
```

```
[False, False, False, False, False, False, False, False, False, False, False, False, False, Fals
```

In [5]:
```
cov.describe()
```

Out[5]:
```
          Elevation         Aspect          Slope  \
count  581012.000000  581012.000000  581012.000000
```

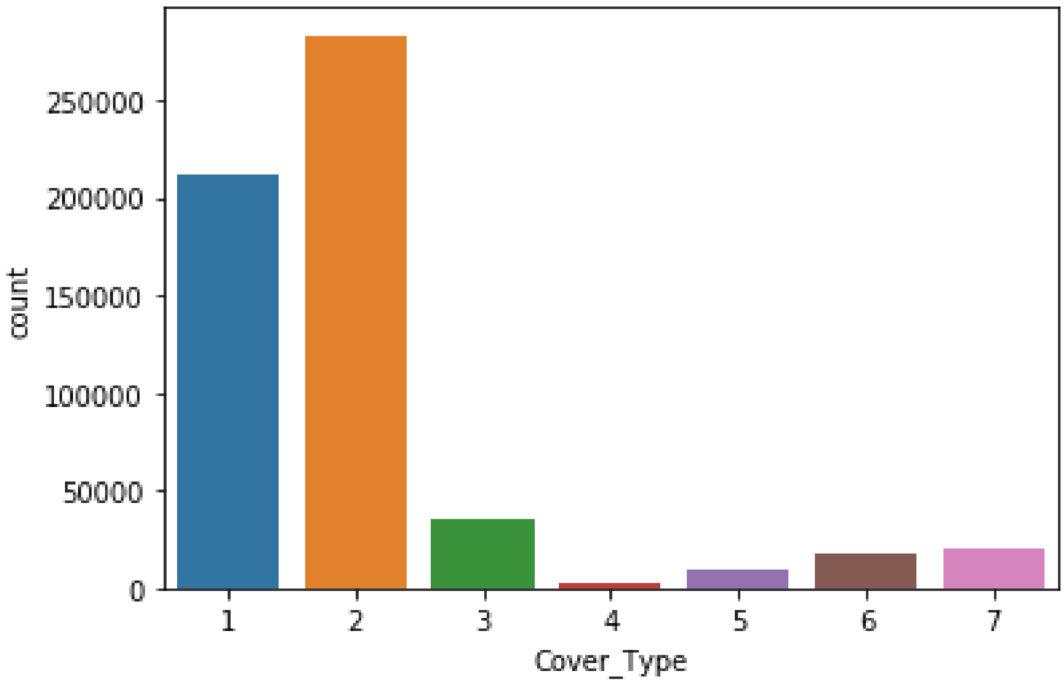
```
mean     2959.365301     155.656807      14.103704
std       279.984734     111.913721       7.488242
min      1859.000000       0.000000       0.000000
25%      2809.000000      58.000000       9.000000
50%      2996.000000     127.000000      13.000000
75%      3163.000000     260.000000      18.000000
max      3858.000000     360.000000      66.000000

       Horizontal_Distance_To_Hydrology  Vertical_Distance_To_Hydrology  \
count                     581012.000000                   581012.000000
mean                         269.428217                       46.418855
std                          212.549356                       58.295232
min                            0.000000                     -173.000000
25%                          108.000000                        7.000000
50%                          218.000000                       30.000000
75%                          384.000000                       69.000000
max                         1397.000000                      601.000000

       Horizontal_Distance_To_Roadways  Hillshade_9am  Hillshade_Noon  \
count                    581012.000000  581012.000000   581012.000000
mean                       2350.146611     212.146049      223.318716
std                        1559.254870      26.769889       19.768697
min                           0.000000       0.000000        0.000000
25%                        1106.000000     198.000000      213.000000
50%                        1997.000000     218.000000      226.000000
75%                        3328.000000     231.000000      237.000000
max                        7117.000000     254.000000      254.000000

       Hillshade_3pm  Horizontal_Distance_To_Fire_Points  ...    Soil_Type32  \
count  581012.000000                       581012.000000  ...  581012.000000
mean      142.528263                         1980.291226  ...       0.090392
std        38.274529                         1324.195210  ...       0.286743
min         0.000000                            0.000000  ...       0.000000
25%       119.000000                         1024.000000  ...       0.000000
50%       143.000000                         1710.000000  ...       0.000000
75%       168.000000                         2550.000000  ...       0.000000
max       254.000000                         7173.000000  ...       1.000000

         Soil_Type33    Soil_Type34    Soil_Type35    Soil_Type36  \
count  581012.000000  581012.000000  581012.000000  581012.000000
mean        0.077716       0.002773       0.003255       0.000205
std         0.267725       0.052584       0.056957       0.014310
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000       0.000000
75%         0.000000       0.000000       0.000000       0.000000
max         1.000000       1.000000       1.000000       1.000000
```

```
       Soil_Type37    Soil_Type38    Soil_Type39    Soil_Type40  \
count  581012.000000  581012.000000  581012.000000  581012.000000
mean        0.000513       0.026803       0.023762       0.015060
std         0.022641       0.161508       0.152307       0.121791
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000       0.000000
75%         0.000000       0.000000       0.000000       0.000000
max         1.000000       1.000000       1.000000       1.000000

          Cover_Type
count  581012.000000
mean        2.051471
std         1.396504
min         1.000000
25%         1.000000
50%         2.000000
75%         2.000000
max         7.000000

[8 rows x 55 columns]
```

In [6]:
```python
cov.Cover_Type.value_counts()
```

Out[6]:
```
2    283301
1    211840
3     35754
7     20510
6     17367
5      9493
4      2747
Name: Cover_Type, dtype: int64
```

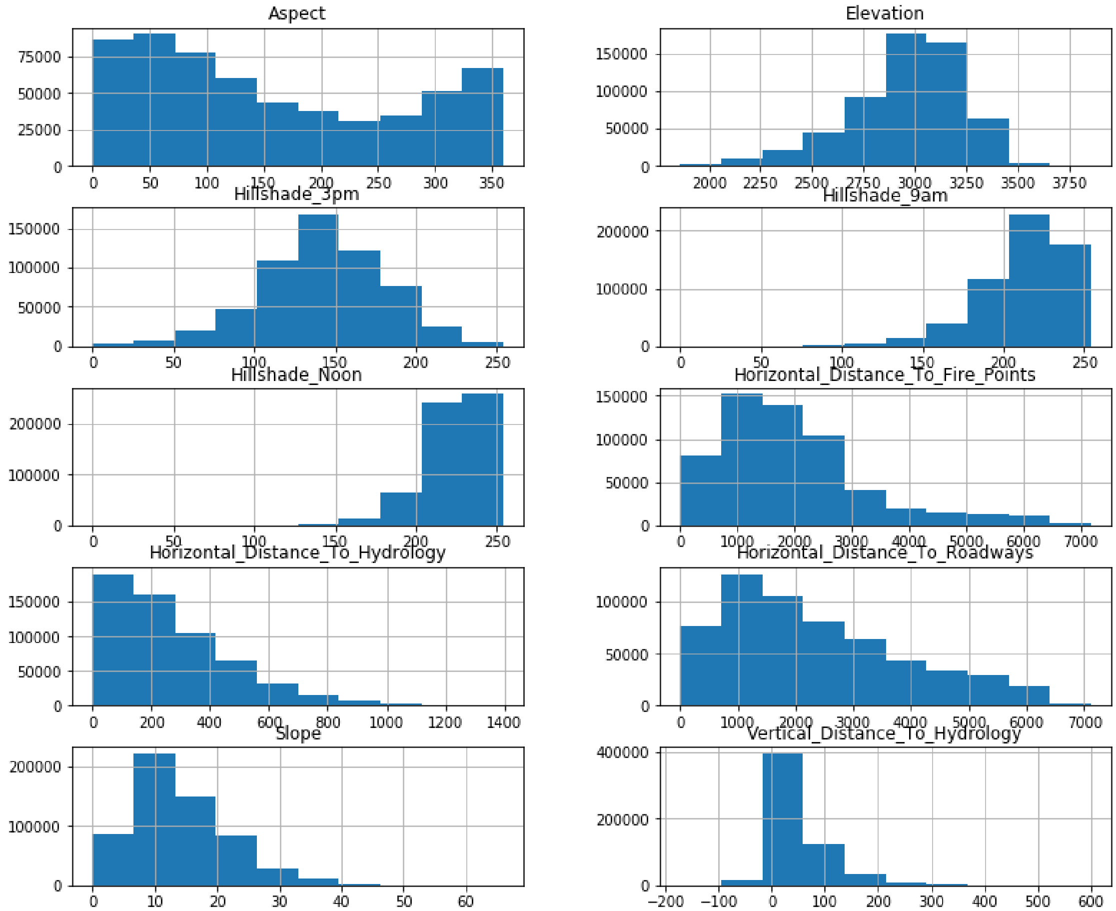
In [7]:
```python
#count plot of target
sb.countplot(x='Cover_Type', data=cov)
plt.show()
```



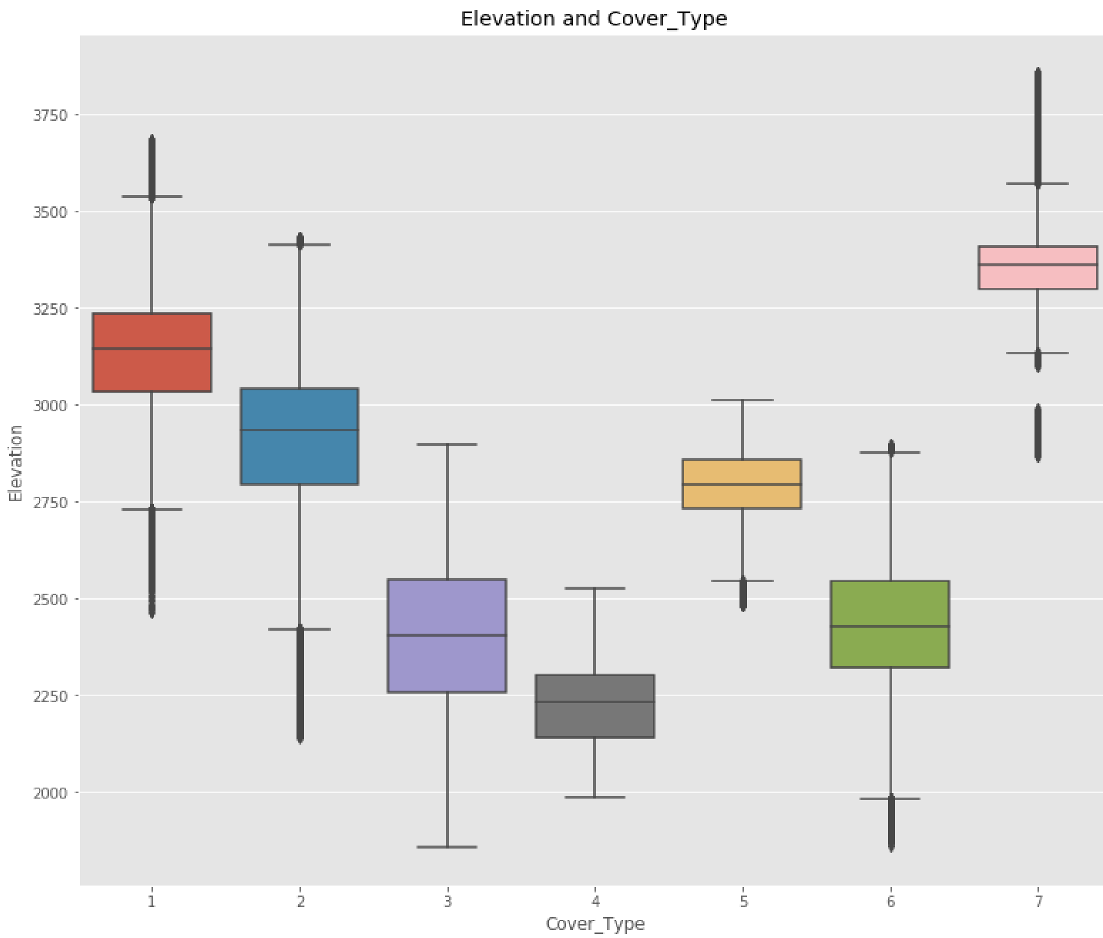

```
In [8]: #Consider the important columns
        col = ['Elevation', 'Aspect', 'Slope', 'Horizontal_Distance_To_Hydrology',
               'Vertical_Distance_To_Hydrology', 'Horizontal_Distance_To_Roadways',
               'Hillshade_9am', 'Hillshade_Noon', 'Hillshade_3pm',
               'Horizontal_Distance_To_Fire_Points']
```

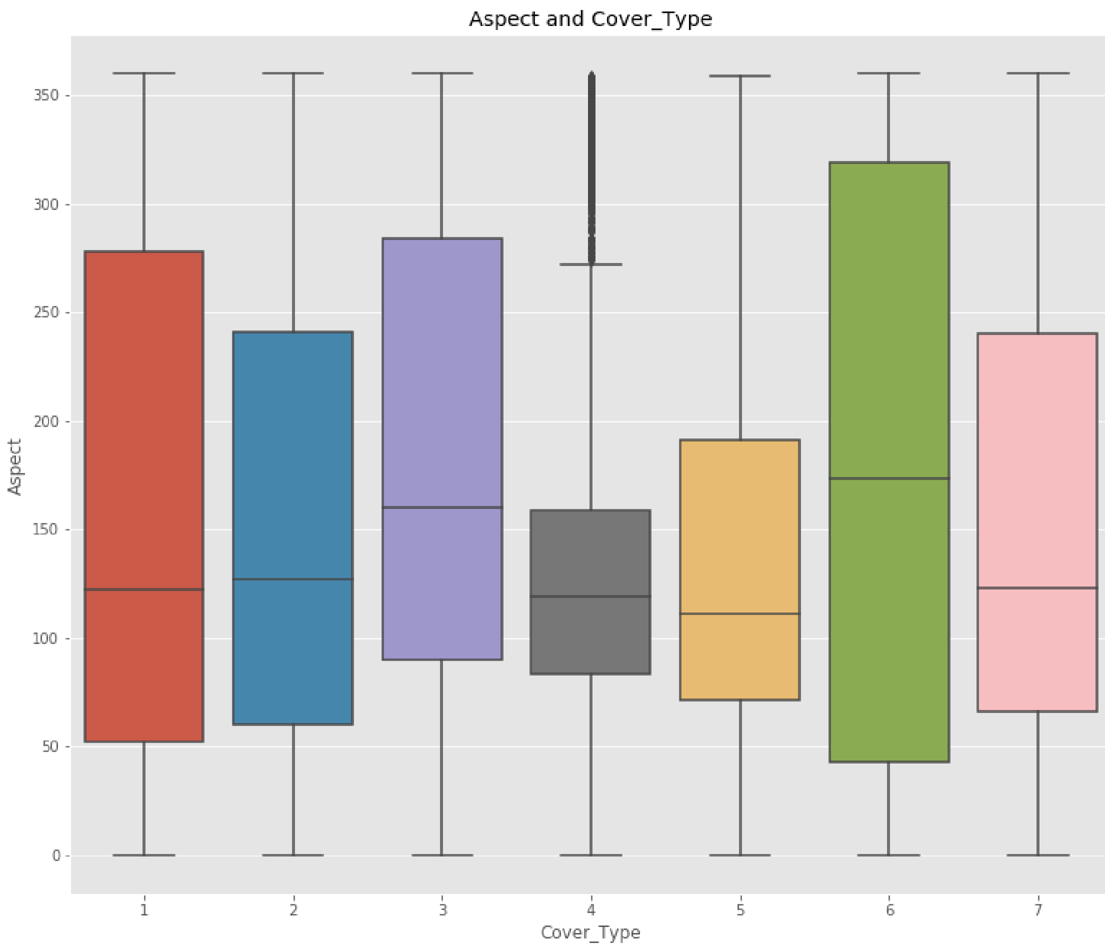
```
In [9]: train = cov[col]
```

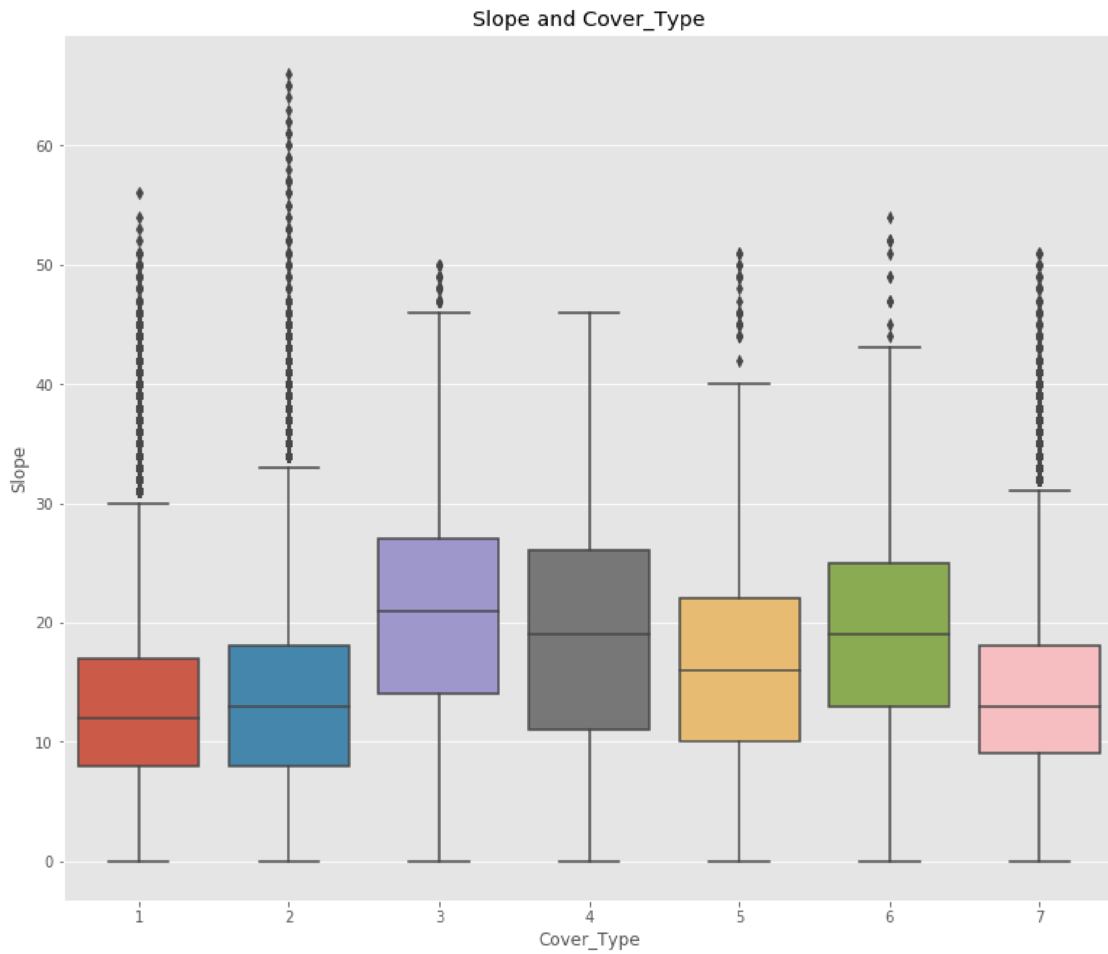
```
In [10]: train.hist(figsize=(13, 11),layout=(5,2))
         plt.show()
```

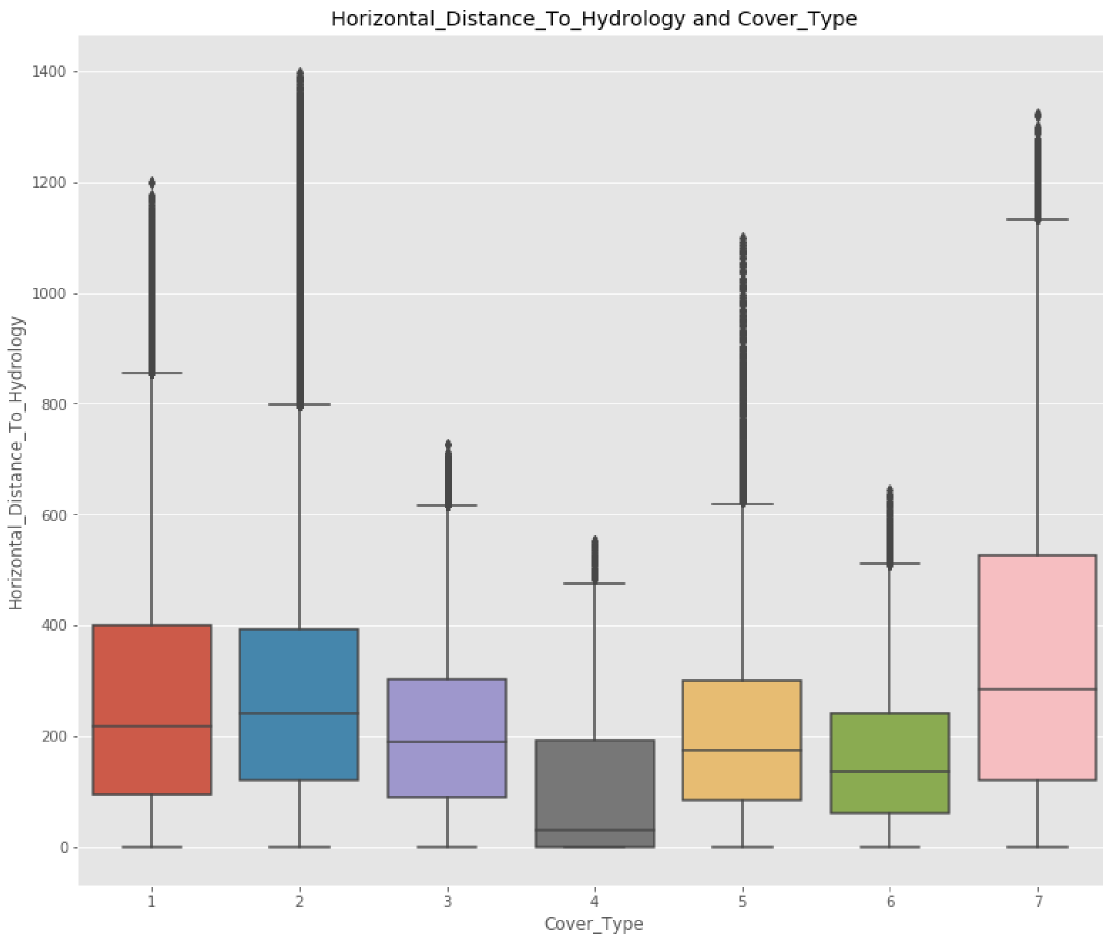


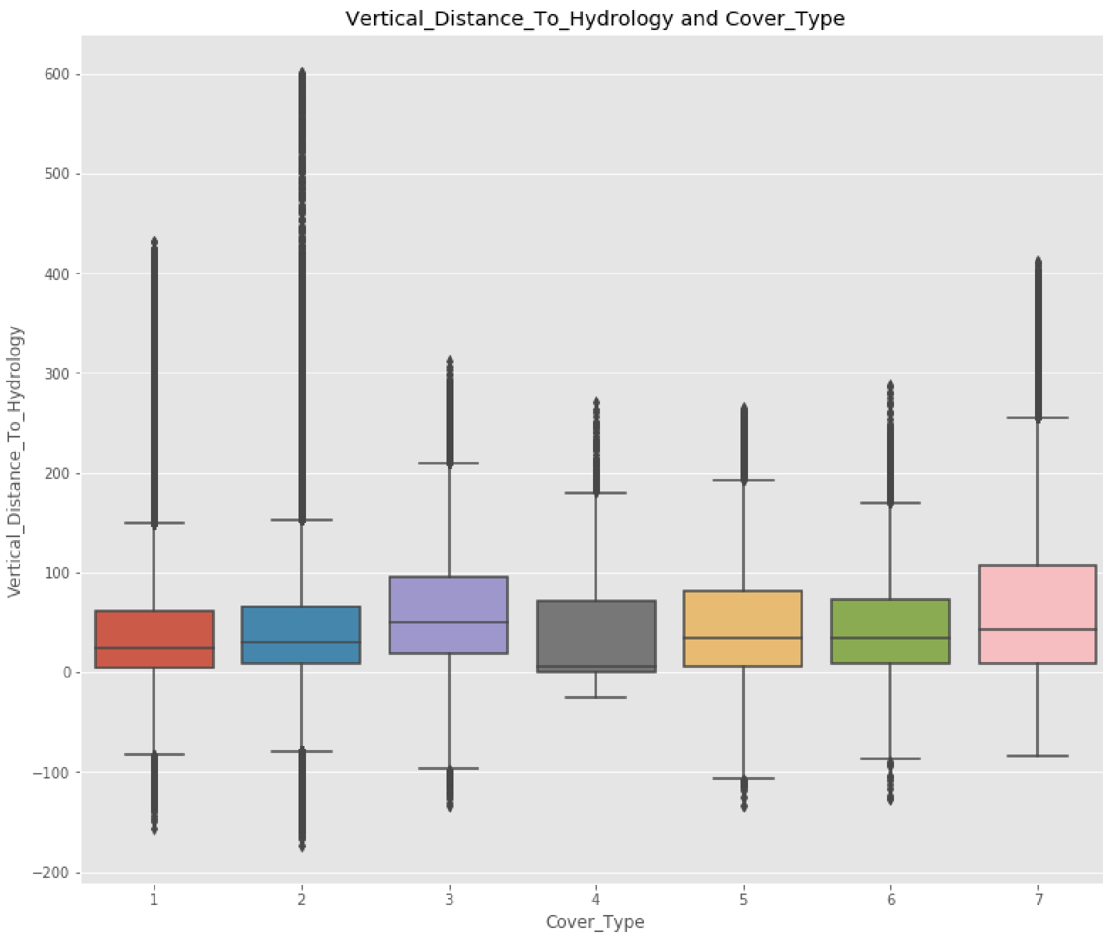
In [11]:
```
#Boxplot
plt.style.use('ggplot')
for i in col:
    plt.figure(figsize=(13, 11))
    plt.title(str(i) + " and " + str('Cover_Type'))
    sb.boxplot(x=cov.Cover_Type, y=train[i])
    plt.show()
```

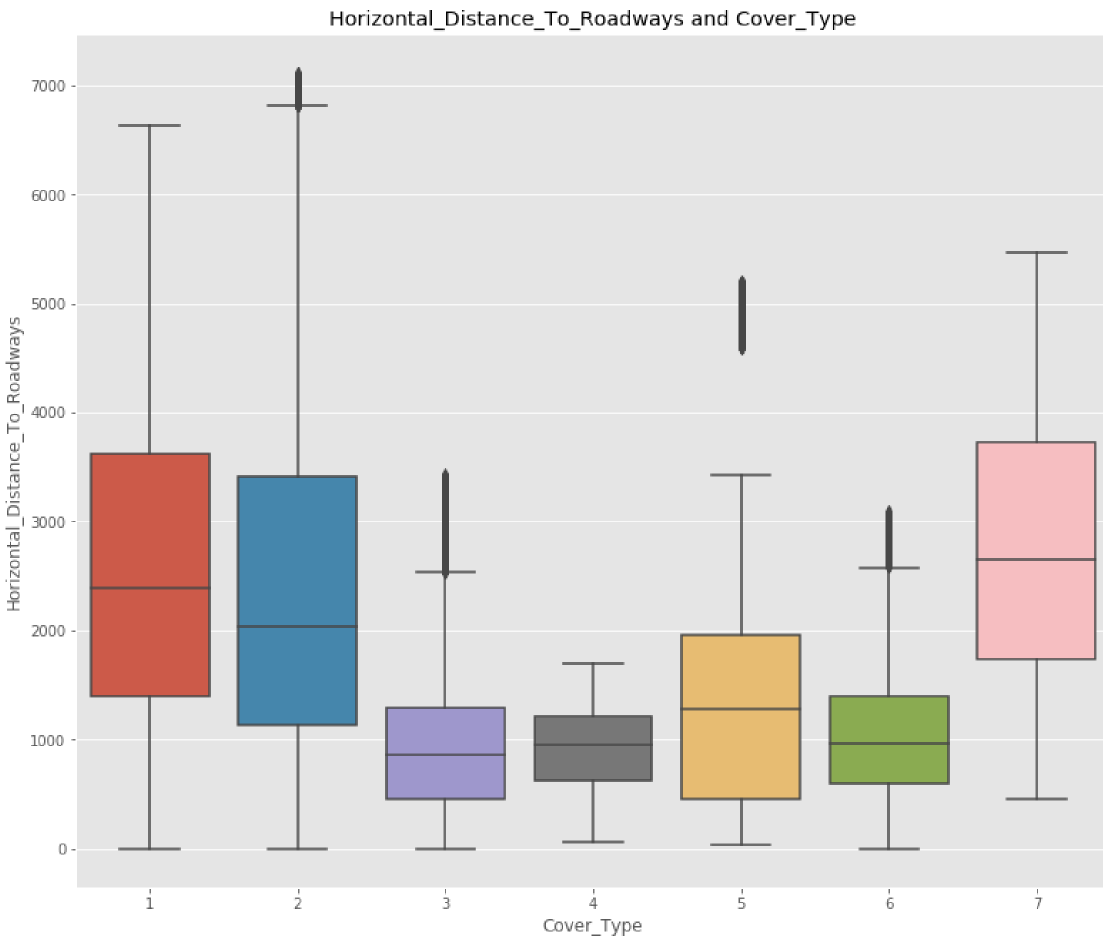

Elevation and Cover_Type
3750
3500
3250
3000
2750
2500
2250
2000
Elevation
1
2
3
4
5
6
7
Cover_Type

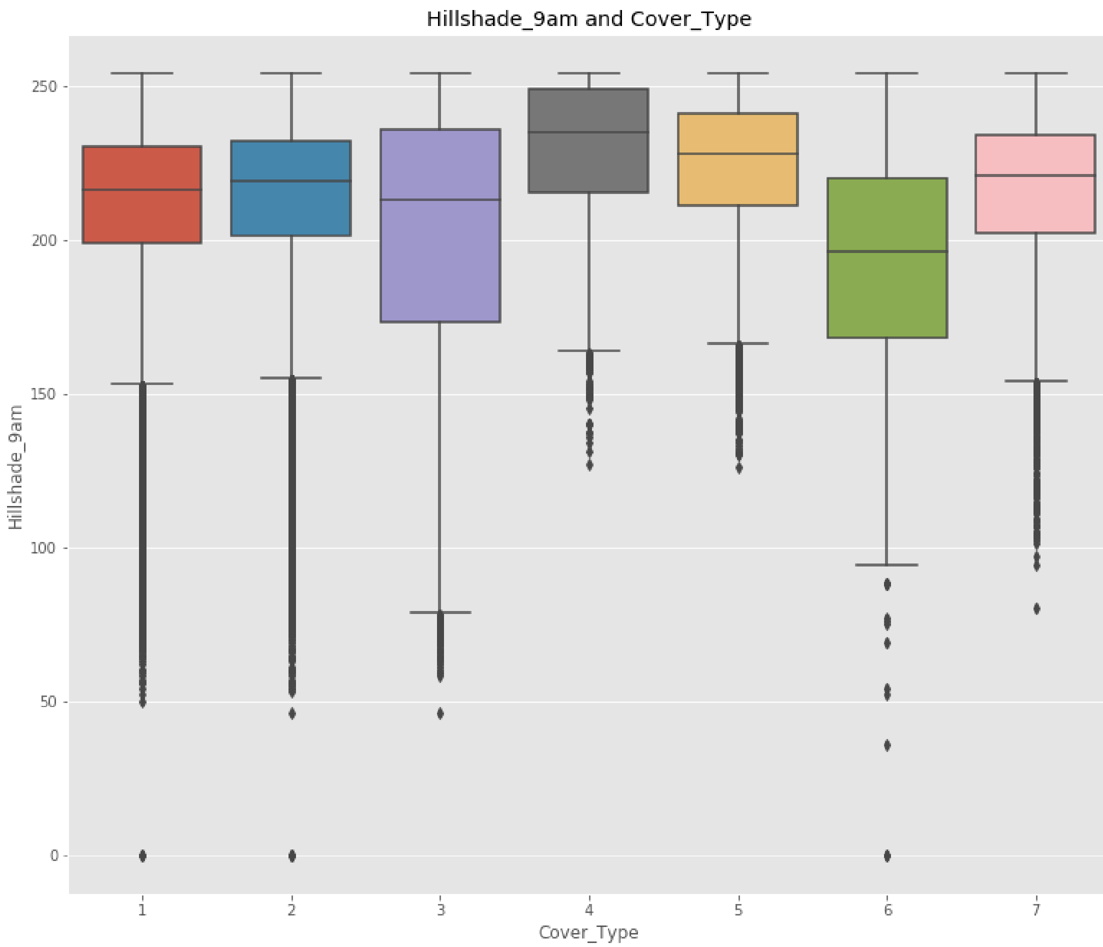

Aspect and Cover_Type
Aspect
350
300
250
200
150
100
50
0
1
2
3
4
5
6
7
Cover_Type

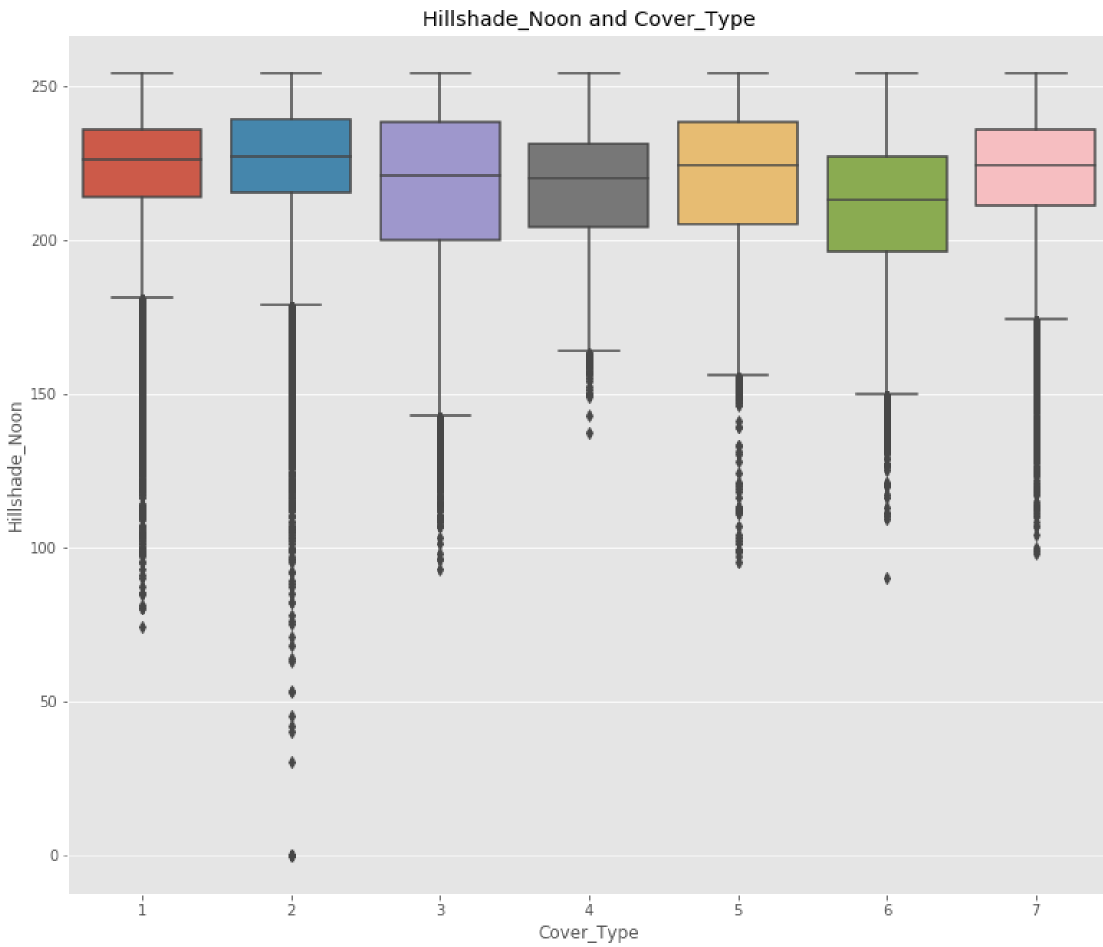

Slope and Cover_Type
Slope
0
10
20
30
40
50
60
1
2
3
4
5
6
7
Cover_Type


Horizontal_Distance_To_Hydrology and Cover_Type
Horizontal_Distance_To_Hydrology
0
200
400
600
800
1000
1200
1400
Cover_Type
1
2
3
4
5
6
7

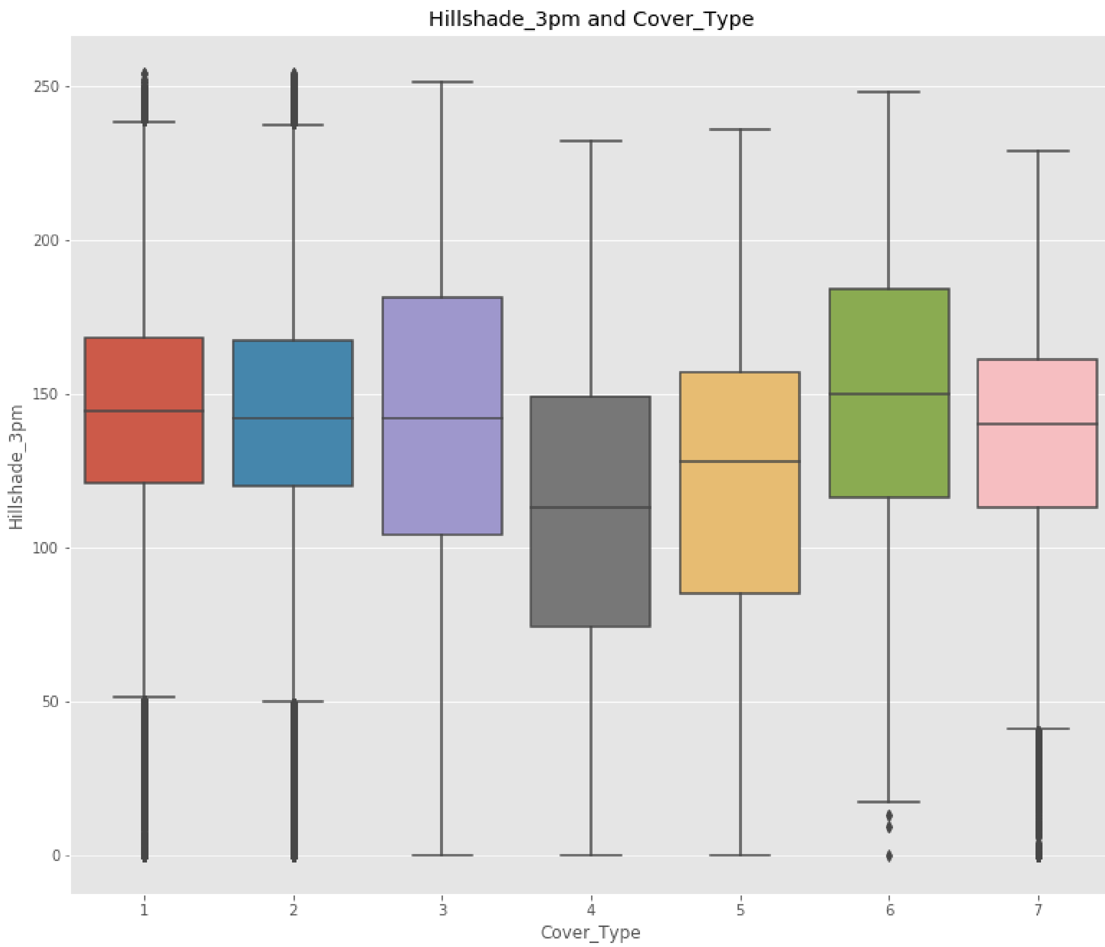

Vertical_Distance_To_Hydrology and Cover_Type
Vertical_Distance_To_Hydrology
600
500
400
300
200
100
0
−100
−200
1
2
3
4
5
6
7
Cover_Type

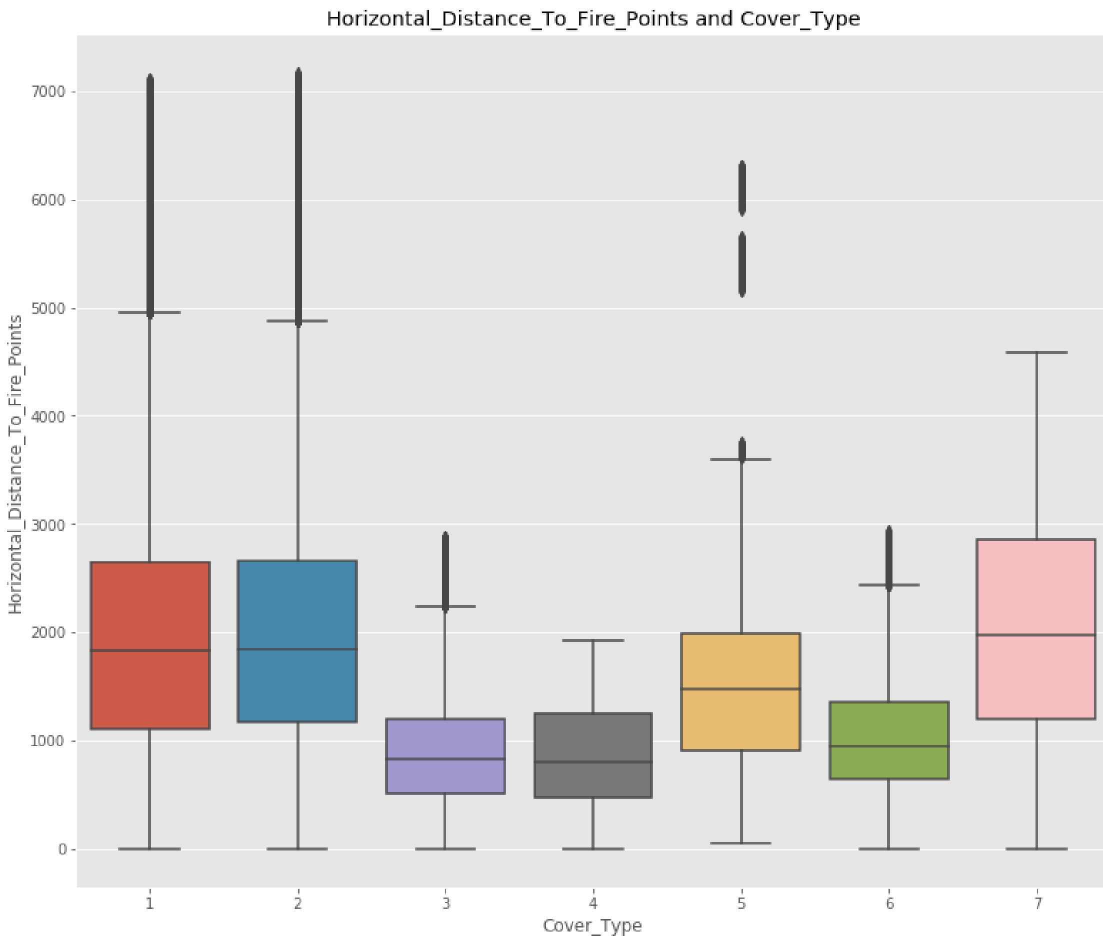

Horizontal_Distance_To_Roadways and Cover_Type
Horizontal_Distance_To_Roadways
0
1000
2000
3000
4000
5000
6000
7000
Cover_Type
1
2
3
4
5
6
7


Hillshade_9am and Cover_Type
Hillshade_9am
Cover_Type
250
200
150
100
50
0
1
2
3
4
5
6
7

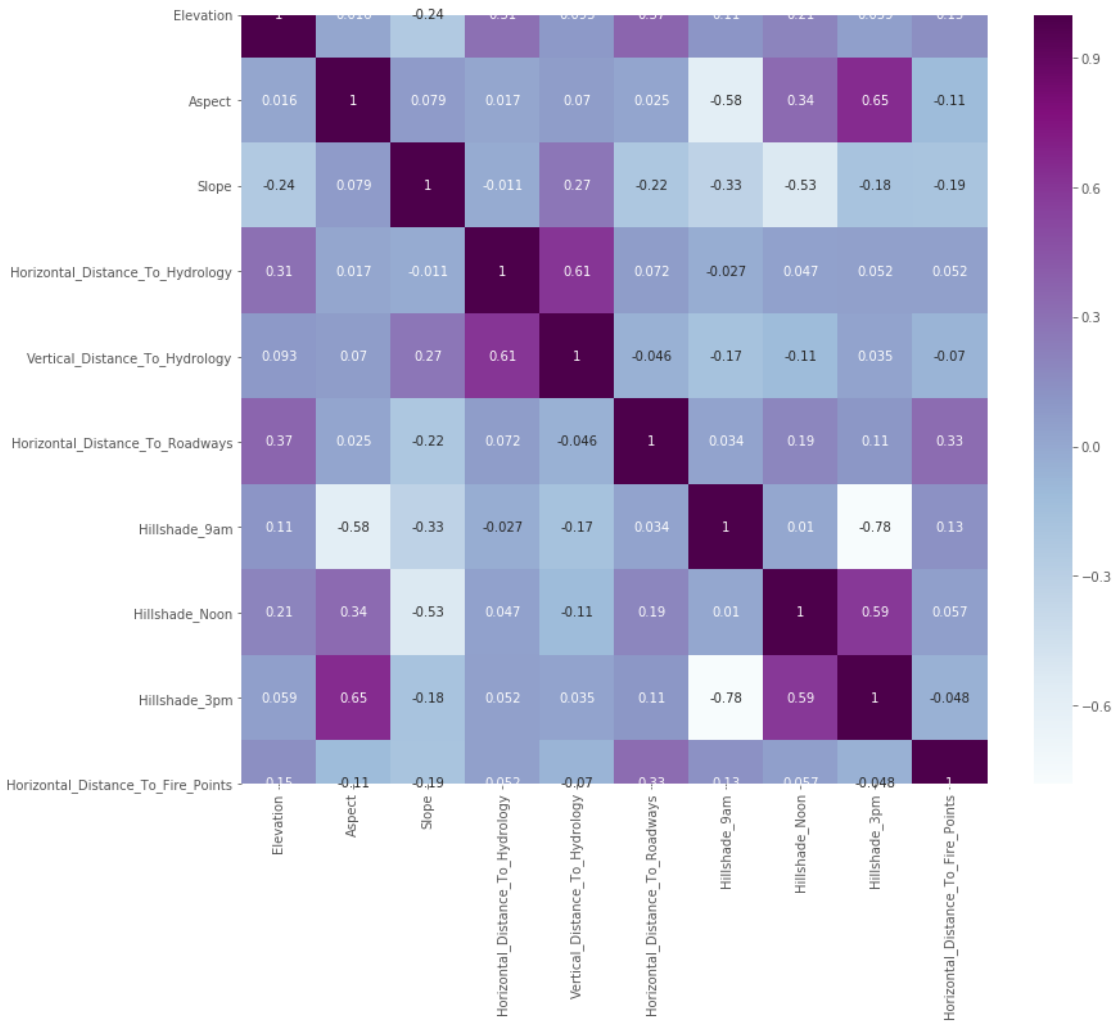


Hillshade_Noon and Cover_Type
Hillshade_Noon
Cover_Type
0
50
100
150
200
250
1
2
3
4
5
6
7


Hillshade_3pm and Cover_Type
Hillshade_3pm
0
50
100
150
200
250
Cover_Type
1
2
3
4
5
6
7



In [12]:
```
#Correlation inbetween the features using seaborn
plt.figure(figsize=(13,11))
corr = train.corr()
sb.heatmap(corr, annot=True, cmap = "BuPu")
plt.show()
```



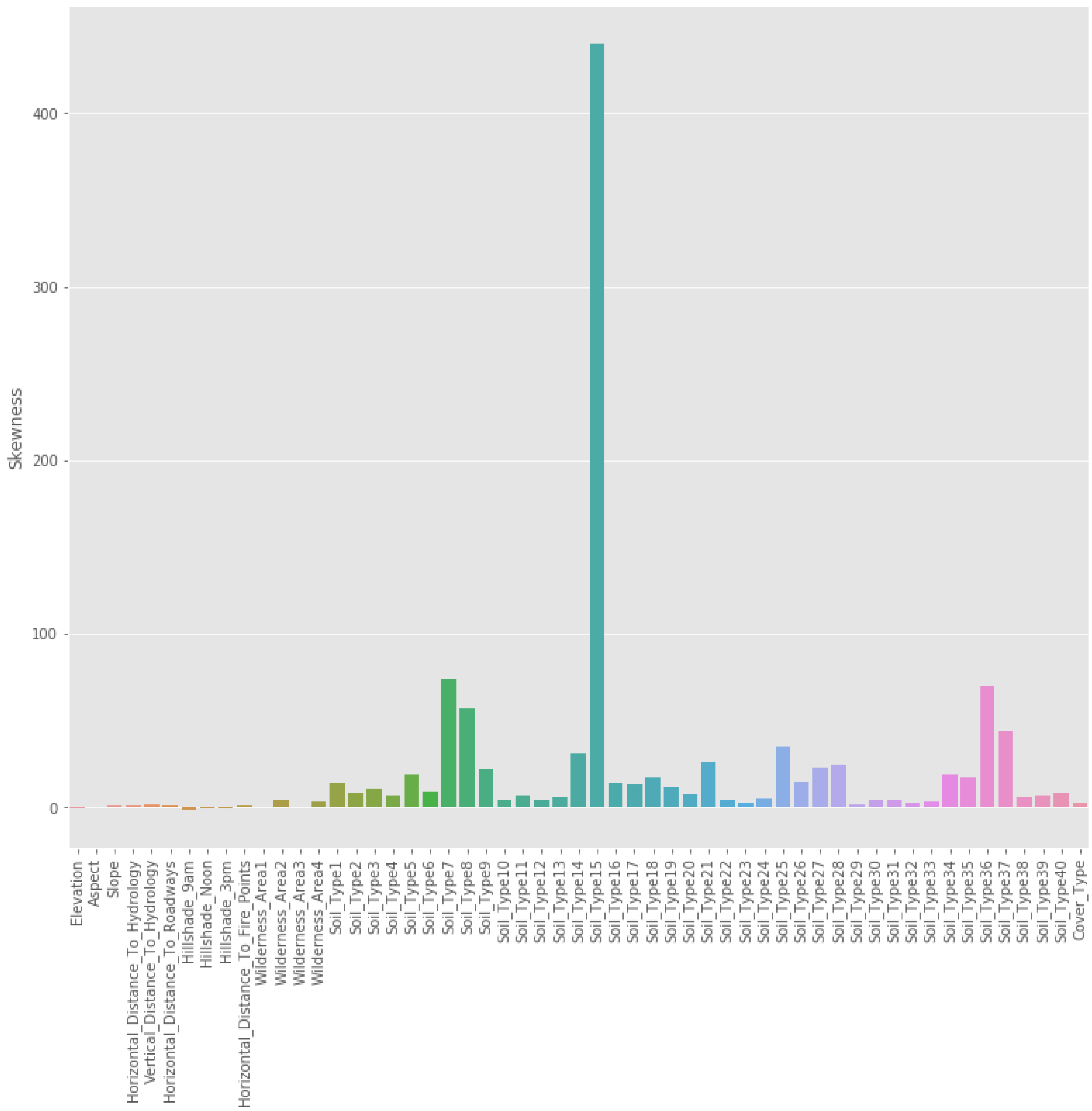
In [13]:
```python
#Skewness:
skewness = cov.skew()
print('Skewness of the below features:')
print(skewness)
```

```
Skewness of the below features:
Elevation                            -0.817596
Aspect                                0.402628
Slope                                 0.789273
Horizontal_Distance_To_Hydrology      1.140437
Vertical_Distance_To_Hydrology        1.790250
Horizontal_Distance_To_Roadways       0.713679
Hillshade_9am                        -1.181147
Hillshade_Noon                       -1.063056
Hillshade_3pm                        -0.277053
Horizontal_Distance_To_Fire_Points    1.288644
```

```
Wilderness_Area1                      0.205618
Wilderness_Area2                      4.061595
Wilderness_Area3                      0.257822
Wilderness_Area4                      3.575561
Soil_Type1                           13.736670
Soil_Type2                            8.615358
Soil_Type3                           10.838630
Soil_Type4                            6.625176
Soil_Type5                           18.995243
Soil_Type6                            9.240061
Soil_Type7                           74.367173
Soil_Type8                           56.946415
Soil_Type9                           22.440005
Soil_Type10                           3.855317
Soil_Type11                           6.621186
Soil_Type12                           4.054662
Soil_Type13                           5.510281
Soil_Type14                          31.096237
Soil_Type15                         440.078023
Soil_Type16                          14.185489
Soil_Type17                          12.914877
Soil_Type18                          17.405794
Soil_Type19                          11.895466
Soil_Type20                           7.730948
Soil_Type21                          26.274260
Soil_Type22                           3.804032
Soil_Type23                           2.677848
Soil_Type24                           4.933954
Soil_Type25                          34.968140
Soil_Type26                          14.880229
Soil_Type27                          23.065265
Soil_Type28                          24.722103
Soil_Type29                           1.512910
Soil_Type30                           4.038910
Soil_Type31                           4.436636
Soil_Type32                           2.856975
Soil_Type33                           3.154625
Soil_Type34                          18.911839
Soil_Type35                          17.442936
Soil_Type36                          69.853269
Soil_Type37                          44.121596
Soil_Type38                           5.859748
Soil_Type39                           6.253684
Soil_Type40                           7.963478
Cover_Type                            2.276574
dtype: float64
```

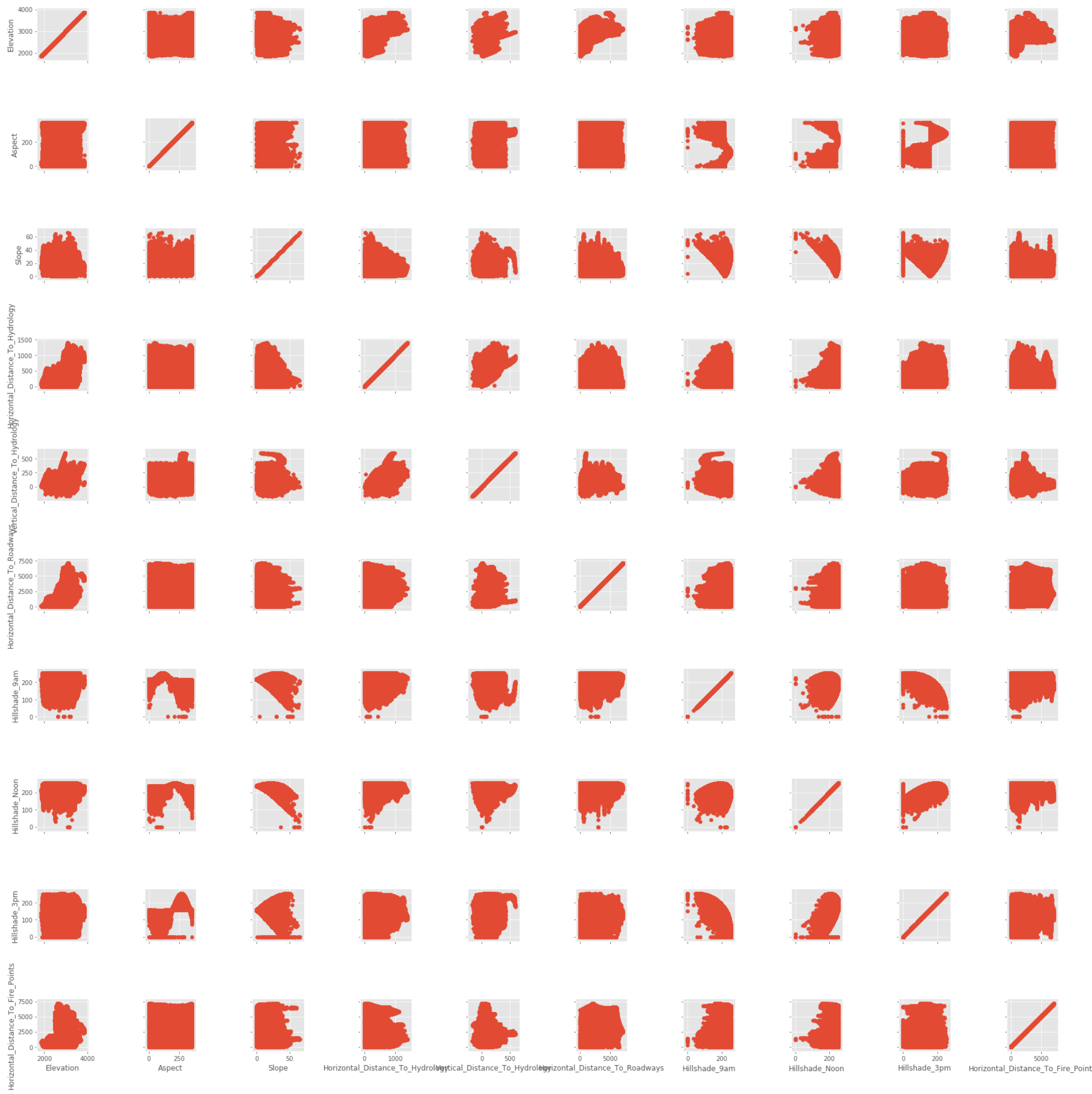
In [14]:
```
df_skew=pd.DataFrame(skewness,columns=['Skewness'])
plt.figure(figsize=(13,11))
sb.barplot(x=df_skew.index,y='Skewness',data=df_skew)
plt.xticks(rotation=90)
```

Out[14]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54]), <a list of 55 Text xticklabel objects>)
```



In [15]:
```
pg = sb.PairGrid(train)
pg.map(plt.scatter)
```

Out[15]: <seaborn.axisgrid.PairGrid at 0x7fc5e70ba1d0>



In [ ]:

In [ ]:

In [ ]:

stat 841 project

December 1, 2019

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#import warnings
#warnings.filterwarnings('ignore')
```

[2]:
```
data=pd.read_csv("/Users/rudranibhadra/Downloads/covtype.csv")
data.head()
```

[2]:
```
   Elevation  Aspect  Slope  Horizontal_Distance_To_Hydrology  \
0       2596      51      3                               258
1       2590      56      2                               212
2       2804     139      9                               268
3       2785     155     18                               242
4       2595      45      2                               153

   Vertical_Distance_To_Hydrology  Horizontal_Distance_To_Roadways  \
0                               0                              510
1                              -6                              390
2                              65                             3180
3                             118                             3090
4                              -1                              391

   Hillshade_9am  Hillshade_Noon  Hillshade_3pm  \
0            221             232            148
1            220             235            151
2            234             238            135
3            238             238            122
4            220             234            150

   Horizontal_Distance_To_Fire_Points  ...  Soil_Type32  Soil_Type33  \
0                                6279  ...            0            0
1                                6225  ...            0            0
2                                6121  ...            0            0
3                                6211  ...            0            0
4                                6172  ...            0            0
```

```
   Soil_Type34  Soil_Type35  Soil_Type36  Soil_Type37  Soil_Type38  \
0            0            0            0            0            0
1            0            0            0            0            0
2            0            0            0            0            0
3            0            0            0            0            0
4            0            0            0            0            0

   Soil_Type39  Soil_Type40  Cover_Type
0            0            0           5
1            0            0           5
2            0            0           2
3            0            0           2
4            0            0           5

[5 rows x 55 columns]
```

[3]:
```
data.isnull().sum()
#no null values
```

[3]:
```
Elevation                             0
Aspect                                0
Slope                                 0
Horizontal_Distance_To_Hydrology      0
Vertical_Distance_To_Hydrology        0
Horizontal_Distance_To_Roadways       0
Hillshade_9am                         0
Hillshade_Noon                        0
Hillshade_3pm                         0
Horizontal_Distance_To_Fire_Points    0
Wilderness_Area1                      0
Wilderness_Area2                      0
Wilderness_Area3                      0
Wilderness_Area4                      0
Soil_Type1                            0
Soil_Type2                            0
Soil_Type3                            0
Soil_Type4                            0
Soil_Type5                            0
Soil_Type6                            0
Soil_Type7                            0
Soil_Type8                            0
Soil_Type9                            0
Soil_Type10                           0
Soil_Type11                           0
Soil_Type12                           0
Soil_Type13                           0
Soil_Type14                           0
```

```
Soil_Type15                           0
Soil_Type16                           0
Soil_Type17                           0
Soil_Type18                           0
Soil_Type19                           0
Soil_Type20                           0
Soil_Type21                           0
Soil_Type22                           0
Soil_Type23                           0
Soil_Type24                           0
Soil_Type25                           0
Soil_Type26                           0
Soil_Type27                           0
Soil_Type28                           0
Soil_Type29                           0
Soil_Type30                           0
Soil_Type31                           0
Soil_Type32                           0
Soil_Type33                           0
Soil_Type34                           0
Soil_Type35                           0
Soil_Type36                           0
Soil_Type37                           0
Soil_Type38                           0
Soil_Type39                           0
Soil_Type40                           0
Cover_Type                            0
dtype: int64
```

[4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 581012 entries, 0 to 581011
Data columns (total 55 columns):
Elevation                             581012 non-null int64
Aspect                                581012 non-null int64
Slope                                 581012 non-null int64
Horizontal_Distance_To_Hydrology      581012 non-null int64
Vertical_Distance_To_Hydrology        581012 non-null int64
Horizontal_Distance_To_Roadways       581012 non-null int64
Hillshade_9am                         581012 non-null int64
Hillshade_Noon                        581012 non-null int64
Hillshade_3pm                         581012 non-null int64
Horizontal_Distance_To_Fire_Points    581012 non-null int64
Wilderness_Area1                      581012 non-null int64
Wilderness_Area2                      581012 non-null int64
Wilderness_Area3                      581012 non-null int64
Wilderness_Area4                      581012 non-null int64
```

```
Soil_Type1                            581012 non-null int64
Soil_Type2                            581012 non-null int64
Soil_Type3                            581012 non-null int64
Soil_Type4                            581012 non-null int64
Soil_Type5                            581012 non-null int64
Soil_Type6                            581012 non-null int64
Soil_Type7                            581012 non-null int64
Soil_Type8                            581012 non-null int64
Soil_Type9                            581012 non-null int64
Soil_Type10                           581012 non-null int64
Soil_Type11                           581012 non-null int64
Soil_Type12                           581012 non-null int64
Soil_Type13                           581012 non-null int64
Soil_Type14                           581012 non-null int64
Soil_Type15                           581012 non-null int64
Soil_Type16                           581012 non-null int64
Soil_Type17                           581012 non-null int64
Soil_Type18                           581012 non-null int64
Soil_Type19                           581012 non-null int64
Soil_Type20                           581012 non-null int64
Soil_Type21                           581012 non-null int64
Soil_Type22                           581012 non-null int64
Soil_Type23                           581012 non-null int64
Soil_Type24                           581012 non-null int64
Soil_Type25                           581012 non-null int64
Soil_Type26                           581012 non-null int64
Soil_Type27                           581012 non-null int64
Soil_Type28                           581012 non-null int64
Soil_Type29                           581012 non-null int64
Soil_Type30                           581012 non-null int64
Soil_Type31                           581012 non-null int64
Soil_Type32                           581012 non-null int64
Soil_Type33                           581012 non-null int64
Soil_Type34                           581012 non-null int64
Soil_Type35                           581012 non-null int64
Soil_Type36                           581012 non-null int64
Soil_Type37                           581012 non-null int64
Soil_Type38                           581012 non-null int64
Soil_Type39                           581012 non-null int64
Soil_Type40                           581012 non-null int64
Cover_Type                            581012 non-null int64
dtypes: int64(55)
memory usage: 243.8 MB
```

[5]:
```
data.describe()
#data description
```

[5]:

```
          Elevation         Aspect          Slope  \
count  581012.000000  581012.000000  581012.000000
mean     2959.365301     155.656807      14.103704
std       279.984734     111.913721       7.488242
min      1859.000000       0.000000       0.000000
25%      2809.000000      58.000000       9.000000
50%      2996.000000     127.000000      13.000000
75%      3163.000000     260.000000      18.000000
max      3858.000000     360.000000      66.000000

       Horizontal_Distance_To_Hydrology  Vertical_Distance_To_Hydrology  \
count                     581012.000000                   581012.000000
mean                         269.428217                       46.418855
std                          212.549356                       58.295232
min                            0.000000                     -173.000000
25%                          108.000000                        7.000000
50%                          218.000000                       30.000000
75%                          384.000000                       69.000000
max                         1397.000000                      601.000000

       Horizontal_Distance_To_Roadways  Hillshade_9am  Hillshade_Noon  \
count                    581012.000000  581012.000000   581012.000000
mean                       2350.146611     212.146049      223.318716
std                        1559.254870      26.769889       19.768697
min                           0.000000       0.000000        0.000000
25%                        1106.000000     198.000000      213.000000
50%                        1997.000000     218.000000      226.000000
75%                        3328.000000     231.000000      237.000000
max                        7117.000000     254.000000      254.000000

       Hillshade_3pm  Horizontal_Distance_To_Fire_Points  ...    Soil_Type32  \
count  581012.000000                       581012.000000  ...  581012.000000
mean      142.528263                         1980.291226  ...       0.090392
std        38.274529                         1324.195210  ...       0.286743
min         0.000000                            0.000000  ...       0.000000
25%       119.000000                         1024.000000  ...       0.000000
50%       143.000000                         1710.000000  ...       0.000000
75%       168.000000                         2550.000000  ...       0.000000
max       254.000000                         7173.000000  ...       1.000000

         Soil_Type33    Soil_Type34    Soil_Type35    Soil_Type36  \
count  581012.000000  581012.000000  581012.000000  581012.000000
mean        0.077716       0.002773       0.003255       0.000205
std         0.267725       0.052584       0.056957       0.014310
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000       0.000000
```

```
75%          0.000000       0.000000       0.000000       0.000000
max          1.000000       1.000000       1.000000       1.000000

         Soil_Type37    Soil_Type38    Soil_Type39    Soil_Type40  \
count  581012.000000  581012.000000  581012.000000  581012.000000
mean        0.000513       0.026803       0.023762       0.015060
std         0.022641       0.161508       0.152307       0.121791
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000       0.000000
75%         0.000000       0.000000       0.000000       0.000000
max         1.000000       1.000000       1.000000       1.000000

          Cover_Type
count  581012.000000
mean        2.051471
std         1.396504
min         1.000000
25%         1.000000
50%         2.000000
75%         2.000000
max         7.000000

[8 rows x 55 columns]
```

[6]:

```
#PCA
from sklearn.preprocessing import StandardScaler

x=data.iloc[:,0:11].values
y=data.iloc[:,54].values
x=StandardScaler().fit_transform(x)
```

[7]:

```
from sklearn.decomposition import PCA
pca=PCA(n_components=10)
pcas=pca.fit_transform(x)
pcd=pd.
 ↪DataFrame(data=pcas,columns=['pca1','pca2','pca3','pca4','pca5','pca6','pca7','pca8','pca9','
```

[8]:

```
pcd
```

[8]:

```
            pca1      pca2      pca3      pca4      pca5      pca6      pca7  \
0      -1.328257 -1.522925 -1.020452  0.385712 -3.395181  1.049953  0.686125
1      -1.238281 -1.617650 -1.293663  0.284398 -3.397491  1.053170  0.719246
2      -1.113183 -2.070857  0.247315  1.011635 -2.388311  0.644410 -0.141083
3      -1.130966 -1.288993  0.867068  1.511532 -2.398574  0.549895 -0.219431
4      -1.325174 -1.556609 -1.402988  0.321403 -3.322718  1.041629  0.677189
5      -1.148023 -1.327152 -1.137919  0.216257 -3.303262  1.036613  1.050123
6      -1.567631 -1.108353 -0.770696  0.738716 -3.189349  1.100642  0.649822
7      -1.423965 -1.394523 -0.965833  0.480217 -3.294332  1.031880  0.657770
```

8	-1.585136	-0.746025	-0.245099	0.936153	-3.221086	0.930936	0.604887
9	-1.877311	-0.741727	-0.653583	0.964767	-3.003213	1.144912	0.704661
10	-0.162627	-1.720519	-0.889630	0.521527	-3.409522	0.798785	0.835947
11	-0.860649	-2.295917	0.314004	0.888684	-0.777497	-0.318708	-0.811650
12	-2.172315	-0.718308	0.280407	1.934745	-1.880967	0.784330	-0.277810
13	0.172452	-1.661102	-1.050811	0.733109	-3.324234	0.850109	0.778904
14	-0.763895	-1.436559	-1.831297	0.390744	-3.094414	0.610337	0.667123
15	-1.638856	-0.825727	-1.680873	0.776080	-2.886565	0.700149	0.425063
16	-0.084550	-1.917247	-1.709724	0.506307	-3.140789	1.023516	1.133179
17	-1.622491	-0.860135	-1.502975	0.629721	-2.935757	0.720637	0.573498
18	-1.289194	-1.175042	-1.598467	0.494729	-3.138385	0.615885	0.305839
19	-1.448487	-1.007130	-1.522783	0.598886	-3.027610	0.611864	0.406800
20	-1.811250	-0.630760	-1.489209	0.841979	-2.796260	0.688969	0.484639
21	0.693730	-2.264406	-0.309570	1.585863	-0.836503	-0.039994	-0.804136
22	-2.777564	-0.251272	0.761423	2.154101	-1.648962	0.764047	-0.228279
23	-1.742960	-0.675703	-1.389803	0.778573	-2.872022	0.745019	0.552648
24	-1.475219	-0.569175	-1.258717	0.982325	-2.903383	0.679129	0.397000
25	-1.149478	-1.231428	-1.984874	0.540116	-2.877936	0.591940	0.504445
26	-0.883864	-1.149191	-1.938365	0.641450	-2.836035	0.613927	0.508740
27	-1.209642	-1.964216	0.405447	1.139201	0.063995	-0.482471	-1.025329
28	-0.714550	-2.718958	-0.464881	0.774451	-2.115804	0.473323	-0.263098
29	-2.872394	-0.255941	0.231919	2.251003	-1.566066	0.921851	-0.246590
...	...	...	...	...	...	...	...
580982	0.670276	0.786224	-1.334094	-1.845159	-1.399898	-0.650840	-0.205844
580983	1.780934	0.959115	-1.353572	-1.318784	-1.336788	-0.579619	-0.229754
580984	1.339256	1.028427	-1.230025	-1.458423	-1.318704	-0.617764	-0.256851
580985	1.072705	1.207486	-1.143053	-1.414049	-1.244727	-0.606792	-0.287197
580986	1.592463	1.412386	-1.115138	-1.046197	-1.172572	-0.552265	-0.317724
580987	1.870239	1.877520	-0.857894	-0.563041	-1.049958	-0.514587	-0.372532
580988	1.230426	2.370519	-0.622247	-0.347944	-0.847154	-0.460069	-0.379338
580989	0.740640	2.784225	-0.524342	-0.112728	-0.655752	-0.387377	-0.369051
580990	-0.201906	3.171256	-0.389724	-0.062730	-0.467634	-0.342513	-0.322591
580991	-0.693467	3.119926	-0.507615	-0.256975	-0.448236	-0.348522	-0.306615
580992	0.275336	2.325164	-0.898015	-0.677364	-0.753291	-0.472848	-0.371034
580993	0.320078	2.139258	-1.001257	-0.837909	-0.816211	-0.508899	-0.371572
580994	0.209199	1.961889	-1.082097	-1.048825	-0.891012	-0.560816	-0.352065
580995	0.411477	1.811558	-1.178843	-1.126349	-0.946295	-0.583235	-0.358257
580996	0.276306	1.847917	-1.227829	-1.130553	-0.915121	-0.570553	-0.357252
580997	0.078322	2.003474	-1.235090	-1.041191	-0.838194	-0.532250	-0.346026
580998	0.006367	2.119902	-1.194659	-0.976880	-0.816530	-0.536120	-0.357403
580999	0.068520	2.090511	-1.253958	-0.978406	-0.821595	-0.528022	-0.358807
581000	0.091682	2.087668	-1.343192	-0.949820	-0.803342	-0.509770	-0.362141
581001	0.227203	2.116970	-1.426436	-0.859870	-0.775539	-0.472440	-0.369743
581002	0.024448	2.178595	-1.464565	-0.857264	-0.734331	-0.453363	-0.363541
581003	-0.325114	2.263054	-1.493123	-0.880752	-0.679172	-0.430946	-0.357488
581004	-0.427702	2.210478	-1.569247	-0.941823	-0.687013	-0.425038	-0.351447
581005	-0.346259	2.019003	-1.684105	-1.075970	-0.756662	-0.443831	-0.348077

581006	-0.392378	1.945133	-1.743665	-1.145671	-0.780608	-0.445247	-0.345004
581007	-0.481969	1.916159	-1.747127	-1.210517	-0.819003	-0.468985	-0.342827
581008	-0.500964	1.845254	-1.879404	-1.252548	-0.829222	-0.454572	-0.339208
581009	-0.209508	1.588947	-2.055250	-1.369317	-0.922281	-0.472151	-0.340793
581010	0.178002	1.332214	-2.185209	-1.458746	-1.026378	-0.496722	-0.331352
581011	0.075827	1.239121	-2.214034	-1.576062	-1.075651	-0.509484	-0.306315

	pca8	pca9	pca10
0	0.690597	-0.060757	0.136138
1	0.697519	-0.144915	0.071531
2	-0.743064	-0.305380	-0.212413
3	-1.584411	-1.001050	-0.148930
4	0.723029	-0.334267	-0.103503
5	-0.109102	0.234920	0.520968
6	0.542177	-0.064696	0.262315
7	0.607600	-0.153468	0.048564
8	0.316249	-0.481553	-0.199357
9	0.173629	-0.045610	0.206849
10	-0.388321	-0.337650	-0.510407
11	-0.699195	0.380692	0.490537
12	-1.799886	-0.625707	0.098247
13	-0.516342	-0.566602	-0.258709
14	-0.215023	-0.215554	-0.290825
15	0.349027	-0.405793	-0.203733
16	-0.369398	0.319107	-0.586697
17	0.124885	-0.313658	-0.152376
18	1.025837	-0.628310	-0.057218
19	0.608046	-0.434309	-0.226196
20	-0.020272	-0.322171	-0.170367
21	-0.793707	-0.551661	0.724684
22	-1.728486	-0.339107	-0.131338
23	0.325862	-0.169085	-0.002131
24	0.716483	-0.474196	0.001876
25	-0.336747	-0.367136	-0.291676
26	-0.745063	-0.466455	0.158014
27	-1.080155	0.270416	0.649323
28	0.121234	0.114547	-0.568872
29	-1.808056	-0.315490	0.067301
...	...	...	...
580982	-0.255081	0.373196	0.402813
580983	-0.289759	-0.065764	0.883653
580984	-0.446053	-0.089087	0.837528
580985	-0.630917	-0.155390	0.959066
580986	-0.611422	-0.410792	1.249219
580987	-0.746426	-0.776809	1.513587
580988	-1.110986	-0.792419	1.600239
580989	-1.348198	-0.758211	1.626330

```
580990 -1.652622 -0.528081  1.431883
580991 -1.722745 -0.317395  1.279675
580992 -1.357293 -0.545493  1.257533
580993 -1.284310 -0.538657  1.142637
580994 -1.196987 -0.456700  0.893219
580995 -1.126080 -0.524506  0.859530
580996 -1.165113 -0.515291  0.825106
580997 -1.242207 -0.489996  0.823775
580998 -1.319483 -0.552650  0.821047
580999 -1.307993 -0.559325  0.859420
581000 -1.291385 -0.573641  0.849689
581001 -1.283231 -0.640004  0.953091
581002 -1.335480 -0.606617  0.896367
581003 -1.434413 -0.520283  0.845679
581004 -1.390160 -0.434234  0.794764
581005 -1.272707 -0.365825  0.739279
581006 -1.223576 -0.298569  0.708493
581007 -1.189832 -0.268048  0.598234
581008 -1.129339 -0.236902  0.535450
581009 -0.972717 -0.236910  0.500838
581010 -0.789218 -0.239629  0.449401
581011 -0.681305 -0.139069  0.304766

[581012 rows x 10 columns]
```

[9]:
```python
finalp=pd.concat([pcd,data[['Cover_Type']]],axis=1)
```

[10]:
```python
finalp
```

[10]:
```
            pca1      pca2      pca3      pca4      pca5      pca6      pca7  \
0      -1.328257 -1.522925 -1.020452  0.385712 -3.395181  1.049953  0.686125
1      -1.238281 -1.617650 -1.293663  0.284398 -3.397491  1.053170  0.719246
2      -1.113183 -2.070857  0.247315  1.011635 -2.388311  0.644410 -0.141083
3      -1.130966 -1.288993  0.867068  1.511532 -2.398574  0.549895 -0.219431
4      -1.325174 -1.556609 -1.402988  0.321403 -3.322718  1.041629  0.677189
5      -1.148023 -1.327152 -1.137919  0.216257 -3.303262  1.036613  1.050123
6      -1.567631 -1.108353 -0.770696  0.738716 -3.189349  1.100642  0.649822
7      -1.423965 -1.394523 -0.965833  0.480217 -3.294332  1.031880  0.657770
8      -1.585136 -0.746025 -0.245099  0.936153 -3.221086  0.930936  0.604887
9      -1.877311 -0.741727 -0.653583  0.964767 -3.003213  1.144912  0.704661
10     -0.162627 -1.720519 -0.889630  0.521527 -3.409522  0.798785  0.835947
11     -0.860649 -2.295917  0.314004  0.888684 -0.777497 -0.318708 -0.811650
12     -2.172315 -0.718308  0.280407  1.934745 -1.880967  0.784330 -0.277810
13      0.172452 -1.661102 -1.050811  0.733109 -3.324234  0.850109  0.778904
14     -0.763895 -1.436559 -1.831297  0.390744 -3.094414  0.610337  0.667123
15     -1.638856 -0.825727 -1.680873  0.776080 -2.886565  0.700149  0.425063
16     -0.084550 -1.917247 -1.709724  0.506307 -3.140789  1.023516  1.133179
17     -1.622491 -0.860135 -1.502975  0.629721 -2.935757  0.720637  0.573498
```

18	-1.289194	-1.175042	-1.598467	0.494729	-3.138385	0.615885	0.305839
19	-1.448487	-1.007130	-1.522783	0.598886	-3.027610	0.611864	0.406800
20	-1.811250	-0.630760	-1.489209	0.841979	-2.796260	0.688969	0.484639
21	0.693730	-2.264406	-0.309570	1.585863	-0.836503	-0.039994	-0.804136
22	-2.777564	-0.251272	0.761423	2.154101	-1.648962	0.764047	-0.228279
23	-1.742960	-0.675703	-1.389803	0.778573	-2.872022	0.745019	0.552648
24	-1.475219	-0.569175	-1.258717	0.982325	-2.903383	0.679129	0.397000
25	-1.149478	-1.231428	-1.984874	0.540116	-2.877936	0.591940	0.504445
26	-0.883864	-1.149191	-1.938365	0.641450	-2.836035	0.613927	0.508740
27	-1.209642	-1.964216	0.405447	1.139201	0.063995	-0.482471	-1.025329
28	-0.714550	-2.718958	-0.464881	0.774451	-2.115804	0.473323	-0.263098
29	-2.872394	-0.255941	0.231919	2.251003	-1.566066	0.921851	-0.246590
...	...	...	...	...	...	...	...
580982	0.670276	0.786224	-1.334094	-1.845159	-1.399898	-0.650840	-0.205844
580983	1.780934	0.959115	-1.353572	-1.318784	-1.336788	-0.579619	-0.229754
580984	1.339256	1.028427	-1.230025	-1.458423	-1.318704	-0.617764	-0.256851
580985	1.072705	1.207486	-1.143053	-1.414049	-1.244727	-0.606792	-0.287197
580986	1.592463	1.412386	-1.115138	-1.046197	-1.172572	-0.552265	-0.317724
580987	1.870239	1.877520	-0.857894	-0.563041	-1.049958	-0.514587	-0.372532
580988	1.230426	2.370519	-0.622247	-0.347944	-0.847154	-0.460069	-0.379338
580989	0.740640	2.784225	-0.524342	-0.112728	-0.655752	-0.387377	-0.369051
580990	-0.201906	3.171256	-0.389724	-0.062730	-0.467634	-0.342513	-0.322591
580991	-0.693467	3.119926	-0.507615	-0.256975	-0.448236	-0.348522	-0.306615
580992	0.275336	2.325164	-0.898015	-0.677364	-0.753291	-0.472848	-0.371034
580993	0.320078	2.139258	-1.001257	-0.837909	-0.816211	-0.508899	-0.371572
580994	0.209199	1.961889	-1.082097	-1.048825	-0.891012	-0.560816	-0.352065
580995	0.411477	1.811558	-1.178843	-1.126349	-0.946295	-0.583235	-0.358257
580996	0.276306	1.847917	-1.227829	-1.130553	-0.915121	-0.570553	-0.357252
580997	0.078322	2.003474	-1.235090	-1.041191	-0.838194	-0.532250	-0.346026
580998	0.006367	2.119902	-1.194659	-0.976880	-0.816530	-0.536120	-0.357403
580999	0.068520	2.090511	-1.253958	-0.978406	-0.821595	-0.528022	-0.358807
581000	0.091682	2.087668	-1.343192	-0.949820	-0.803342	-0.509770	-0.362141
581001	0.227203	2.116970	-1.426436	-0.859870	-0.775539	-0.472440	-0.369743
581002	0.024448	2.178595	-1.464565	-0.857264	-0.734331	-0.453363	-0.363541
581003	-0.325114	2.263054	-1.493123	-0.880752	-0.679172	-0.430946	-0.357488
581004	-0.427702	2.210478	-1.569247	-0.941823	-0.687013	-0.425038	-0.351447
581005	-0.346259	2.019003	-1.684105	-1.075970	-0.756662	-0.443831	-0.348077
581006	-0.392378	1.945133	-1.743665	-1.145671	-0.780608	-0.445247	-0.345004
581007	-0.481969	1.916159	-1.747127	-1.210517	-0.819003	-0.468985	-0.342827
581008	-0.500964	1.845254	-1.879404	-1.252548	-0.829222	-0.454572	-0.339208
581009	-0.209508	1.588947	-2.055250	-1.369317	-0.922281	-0.472151	-0.340793
581010	0.178002	1.332214	-2.185209	-1.458746	-1.026378	-0.496722	-0.331352
581011	0.075827	1.239121	-2.214034	-1.576062	-1.075651	-0.509484	-0.306315

	pca8	pca9	pca10	Cover_Type
0	0.690597	-0.060757	0.136138	5
1	0.697519	-0.144915	0.071531	5

```
2      -0.743064 -0.305380 -0.212413           2
3      -1.584411 -1.001050 -0.148930           2
4       0.723029 -0.334267 -0.103503           5
5      -0.109102  0.234920  0.520968           2
6       0.542177 -0.064696  0.262315           5
7       0.607600 -0.153468  0.048564           5
8       0.316249 -0.481553 -0.199357           5
9       0.173629 -0.045610  0.206849           5
10     -0.388321 -0.337650 -0.510407           5
11     -0.699195  0.380692  0.490537           2
12     -1.799886 -0.625707  0.098247           2
13     -0.516342 -0.566602 -0.258709           5
14     -0.215023 -0.215554 -0.290825           5
15      0.349027 -0.405793 -0.203733           5
16     -0.369398  0.319107 -0.586697           5
17      0.124885 -0.313658 -0.152376           5
18      1.025837 -0.628310 -0.057218           5
19      0.608046 -0.434309 -0.226196           5
20     -0.020272 -0.322171 -0.170367           5
21     -0.793707 -0.551661  0.724684           2
22     -1.728486 -0.339107 -0.131338           5
23      0.325862 -0.169085 -0.002131           5
24      0.716483 -0.474196  0.001876           5
25     -0.336747 -0.367136 -0.291676           5
26     -0.745063 -0.466455  0.158014           5
27     -1.080155  0.270416  0.649323           2
28      0.121234  0.114547 -0.568872           2
29     -1.808056 -0.315490  0.067301           5
...          ...       ...       ...         ...
580982 -0.255081  0.373196  0.402813           3
580983 -0.289759 -0.065764  0.883653           3
580984 -0.446053 -0.089087  0.837528           3
580985 -0.630917 -0.155390  0.959066           3
580986 -0.611422 -0.410792  1.249219           3
580987 -0.746426 -0.776809  1.513587           3
580988 -1.110986 -0.792419  1.600239           3
580989 -1.348198 -0.758211  1.626330           3
580990 -1.652622 -0.528081  1.431883           3
580991 -1.722745 -0.317395  1.279675           3
580992 -1.357293 -0.545493  1.257533           3
580993 -1.284310 -0.538657  1.142637           3
580994 -1.196987 -0.456700  0.893219           3
580995 -1.126080 -0.524506  0.859530           3
580996 -1.165113 -0.515291  0.825106           3
580997 -1.242207 -0.489996  0.823775           3
580998 -1.319483 -0.552650  0.821047           3
580999 -1.307993 -0.559325  0.859420           3
```

```
581000 -1.291385 -0.573641  0.849689          3
581001 -1.283231 -0.640004  0.953091          3
581002 -1.335480 -0.606617  0.896367          3
581003 -1.434413 -0.520283  0.845679          3
581004 -1.390160 -0.434234  0.794764          3
581005 -1.272707 -0.365825  0.739279          3
581006 -1.223576 -0.298569  0.708493          3
581007 -1.189832 -0.268048  0.598234          3
581008 -1.129339 -0.236902  0.535450          3
581009 -0.972717 -0.236910  0.500838          3
581010 -0.789218 -0.239629  0.449401          3
581011 -0.681305 -0.139069  0.304766          3

[581012 rows x 11 columns]
```

[11]:
```
a1=pcd.std(axis=0)
```

[12]:
```
a1
```

[12]:
```
pca1     1.620338
pca2     1.550967
pca3     1.319215
pca4     1.119629
pca5     0.878327
pca6     0.786359
pca7     0.681971
pca8     0.681358
pca9     0.592279
pca10    0.549343
dtype: float64
```

[13]:
```
a2=pcd.var(axis=0)
```

[14]:
```
pca.explained_variance_ratio_
```

[14]:
```
array([0.23868098, 0.21868141, 0.15821146, 0.11396071, 0.07013249,
       0.05621451, 0.0422803 , 0.04220441, 0.03189036, 0.02743433])
```

[15]:
```
#feature selection
from sklearn.ensemble import ExtraTreesClassifier
#from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel

x=data.iloc[:,0:54]
y=data.iloc[:,54]

ec=ExtraTreesClassifier()
ec=ec.fit(x,y)

m= SelectFromModel(ec, prefit=True)
```

```
X = m.transform(x)
```

```
/Users/rudranibhadra/anaconda3/lib/python3.7/site-
packages/sklearn/ensemble/forest.py:245: FutureWarning: The default value of
n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

[16]:
```
X.shape
```

[16]:
```
(581012, 12)
```

[17]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
#X=pd.get_dummies(X, drop_first=True)
#y=data['C']
#y=pd.get_dummies(y, drop_first=True)
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
 ↪20,random_state=10)
#X_train2,X_val,y_train2,y_val=train_test_split(X_train,y_train,size=0.
 ↪30,random_state=10)
#scaler=StandardScaler()
#X_train=scaler.fit_transform(X_train)
#X_test=scaler.transform(X_test)
```

[18]:
```
from sklearn.ensemble import AdaBoostClassifier, BaggingClassifier,␣
 ↪GradientBoostingClassifier, RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression, LassoCV,␣
 ↪LogisticRegressionCV, RidgeClassifier, RidgeClassifierCV
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis,␣
 ↪LinearDiscriminantAnalysis
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
from sklearn.neural_network import MLPClassifier
```

[131]:
```
# classification_model=[LogisticRegression(),LogisticRegressionCV(),LassoCV(),␣
 ↪RidgeClassifier(),
#            RidgeClassifierCV(),␣
 ↪KNeighborsClassifier(),DecisionTreeClassifier(), GaussianNB(),
#            AdaBoostClassifier(), BaggingClassifier(),ExtraTreesClassifier(),
#           ␣
 ↪GradientBoostingClassifier(),RandomForestClassifier(),LinearDiscriminantAnalysis(),
#            QuadraticDiscriminantAnalysis(),SVC()]
```

[92]:
```
# acc_df = pd.DataFrame(list(modelaccuracy.items()), columns=['Model',␣
 ↪'Accuracy']).sort_values('Accuracy', ascending=False).reset_index(drop=True)
```

```
# acc_df.index=acc_df.index+1
# sns.barplot(data=acc_df,y='Model',x='Accuracy')
# plt.xlim(0,1)
# plt.title('Accuracy of models with default settings')
# plt.xticks(rotation=45)
# plt.show()

# acc_df
```

[1]:
```
# #QDA
# from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
# QDA = QuadraticDiscriminantAnalysis()
# QDA.fit(X_train, y_train)

# #prediction
# y_pred = QDA.predict(X_test)

# #score
# print("Accuracy -- ", QDA.score(X_test, y_test)*100)

# #confusion
# cm = confusion_matrix(y_pred, y_test)
# plt.figure(figsize=(10, 8))
# sns.set(font_scale=1.2)
# sns.heatmap(cm, annot=True, fmt='g',cmap="BuPu")
# plt.show()
```

[19]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error
```

[20]:
```
import warnings
warnings.filterwarnings(action="ignore")
```

[21]:
```
kf=KFold(n_splits=10, random_state=40, shuffle=True)

def cv_rmse(model):
    rmse = np.sqrt(-cross_val_score(model, X_train,␣
 ↪y_train,scoring="neg_mean_squared_error",cv=10))
    return (rmse)
```

[25]:
```
RFC = RandomForestClassifier(n_estimators=100)
LR = LogisticRegression()
LDA = LinearDiscriminantAnalysis()
GN = GaussianNB()
AD = AdaBoostClassifier()
DT=DecisionTreeClassifier()
BG=BaggingClassifier()
SV=SVC()
```

```
GC=GradientBoostingClassifier()
KN=KNeighborsClassifier()
NN=MLPClassifier()





#cross_val_score(LR, X_train, y_train)
#print(X.shape)
#print(y_train.shape)
```

[26]:
```
scores={}
```

[27]:
```
s=cv_rmse(NN)
print("NN: {:.4f} ({:.4f})".format(s.mean(), s.std()))
scores['neural network'] = (s.mean(), s.std())
```

```
NN: 1.2161 (0.0145)
```

[28]:
```
s=cv_rmse(RFC)
print("RFC: {:.4f} ({:.4f})".format(s.mean(), s.std()))
scores['random forest'] = (s.mean(), s.std())
```

```
RFC: 0.5175 (0.0148)
```

[29]:
```
s=cv_rmse(LR)
print("LR: {:.4f} ({:.4f})".format(s.mean(), s.std()))
scores['logistic regression'] = (s.mean(), s.std())
```

```
LR: 1.4168 (0.0032)
```

[30]:
```
s=cv_rmse(LDA)
print("LDA: {:.4f} ({:.4f})".format(s.mean(), s.std()))
scores['LDA'] = (s.mean(), s.std())
```

```
LDA: 1.4133 (0.0056)
```

[31]:
```
s=cv_rmse(GN)
print("GN: {:.4f} ({:.4f})".format(s.mean(), s.std()))
scores['gaussian naive bayes'] = (s.mean(), s.std())
```

```
GN: 1.9364 (0.0055)
```

[32]:
```
s=cv_rmse(AD)
print("AD: {:.4f} ({:.4f})".format(s.mean(), s.std()))
scores['adaboost'] = (s.mean(), s.std())
```

```
AD: 2.4141 (0.3989)
```

[33]:
```
s=cv_rmse(DT)
print("DT: {:.4f} ({:.4f})".format(s.mean(), s.std()))
scores['decision tree'] = (s.mean(), s.std())
```

```
DT: 0.6602 (0.0125)
```

[34]:
```
s=cv_rmse(BG)
print("BG: {:.4f} ({:.4f})".format(s.mean(), s.std()))
scores['bagging'] = (s.mean(), s.std())
```

```
BG: 0.5125 (0.0147)
```

[ ]:
```
# s=cv_rmse(SV)
# print("SV: {:.4f} ({:.4f})".format(s.mean(), s.std()))
# scores['sv'] = (s.mean(), s.std())
```

[27]:
```
s=cv_rmse(GC)
print("GC: {:.4f} ({:.4f})".format(s.mean(), s.std()))
scores['gradient boosting'] = (s.mean(), s.std())
```

```
GC: 1.1552 (0.0083)
```

[28]:
```
s=cv_rmse(KN)
print("KN: {:.4f} ({:.4f})".format(s.mean(), s.std()))
scores['k nearest neighbour'] = (s.mean(), s.std())
```

```
KN: 0.4151 (0.0136)
```

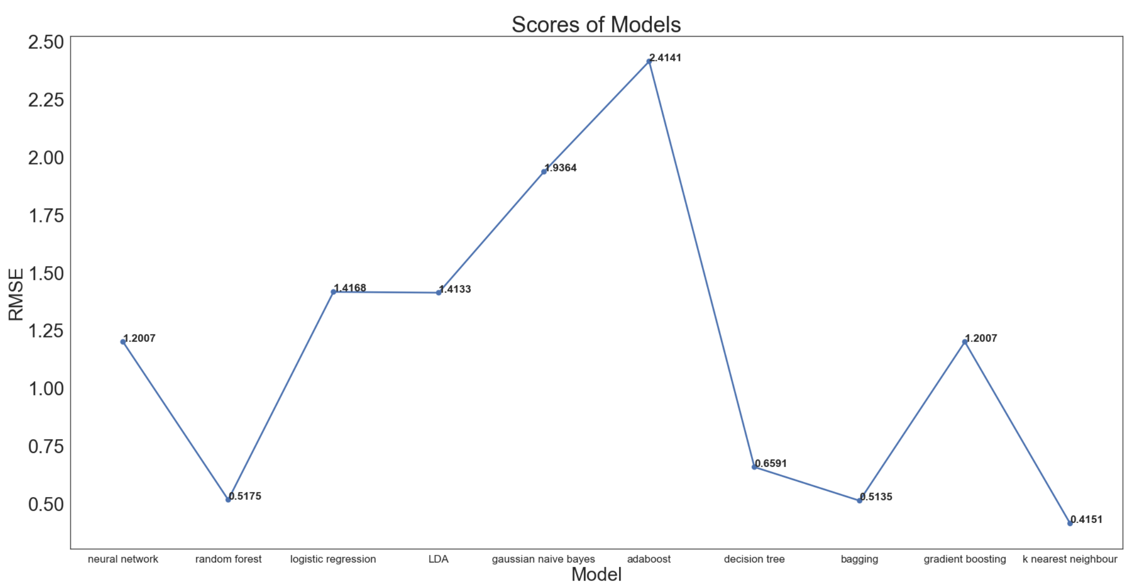
[35]:
```
# scores['neural network'] = (1.2007,0.0236)
# scores['random forest'] = (0.5175,0.0172)
# scores['logistic regression'] = (1.4168,0.0032)
# scores['LDA'] = (1.4133,0.0056)
# scores['gaussian naive bayes'] = (1.9364,0.0055)
# scores['adaboost'] = (2.4141,0.3989)
# scores['decision tree'] = (0.6591,0.0111)
# scores['bagging'] = (0.5135,0.0122)
# scores['gradient boosting'] = (1.2007,0.0236)
# scores['k nearest neighbour'] = (0.4151,0.0136)
```

[28]:
```
scores
```

[28]:
```
{'neural network': (1.2007, 0.0236),
 'random forest': (0.5175, 0.0172),
 'logistic regression': (1.4168, 0.0032),
 'LDA': (1.4133, 0.0056),
 'gaussian naive bayes': (1.9364, 0.0055),
 'adaboost': (2.4141, 0.3989),
```

```
 'decision tree': (0.6591, 0.0111),
 'bagging': (0.5135, 0.0122),
 'gradient boosting': (1.2007, 0.0236),
 'k nearest neighbour': (0.4151, 0.0136)}
```

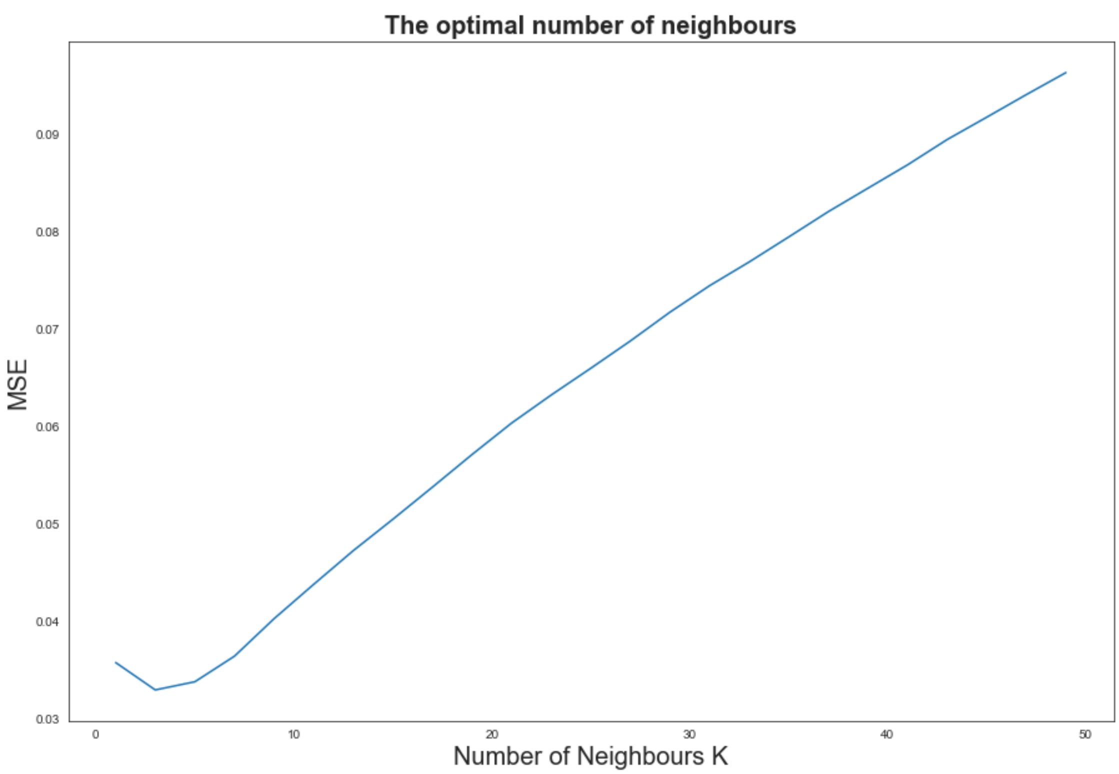
[41]:
```
sns.set_style("white")
fig=plt.figure(figsize=(30,15))

ax=sns.pointplot(x=list(scores.keys()),y=[score for score, _ in scores.
 ↪values()], markers=['o'], linestyles=['-'])
for i,score in enumerate(scores.values()):
    ax.text(i,score[0]+0.002,'{:.4f}'.
 ↪format(score[0]),size='large',weight='semibold')

plt.ylabel('RMSE',size=30)
plt.xlabel('Model',size=30)
plt.title('Scores of Models',size=35)
plt.tick_params(axis='x', labelsize=30)
plt.tick_params(axis='y', labelsize=30)
plt.xticks(size=17)
plt.show()
```



[37]:
```
#using cv for parameter tuning

kl=list(range(1,50,2))
cv_scores=[]

for k in kl:
```

```
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())

# calculating misclassification error
MSE = [1 - x for x in cv_scores]

plt.figure()
plt.figure(figsize=(15,10))
plt.title('The optimal number of neighbours', fontsize=20, fontweight='bold')
plt.xlabel('Number of Neighbours K', fontsize=20)
plt.ylabel('MSE', fontsize=20)
sns.set_style("whitegrid")
plt.plot(kl, MSE)

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



[38]:

```
#findng optimal number of neighbours

bestk = kl[MSE.index(min(MSE))]
```

```python
print("The optimal number of neighbours = %d." % bestk)
```

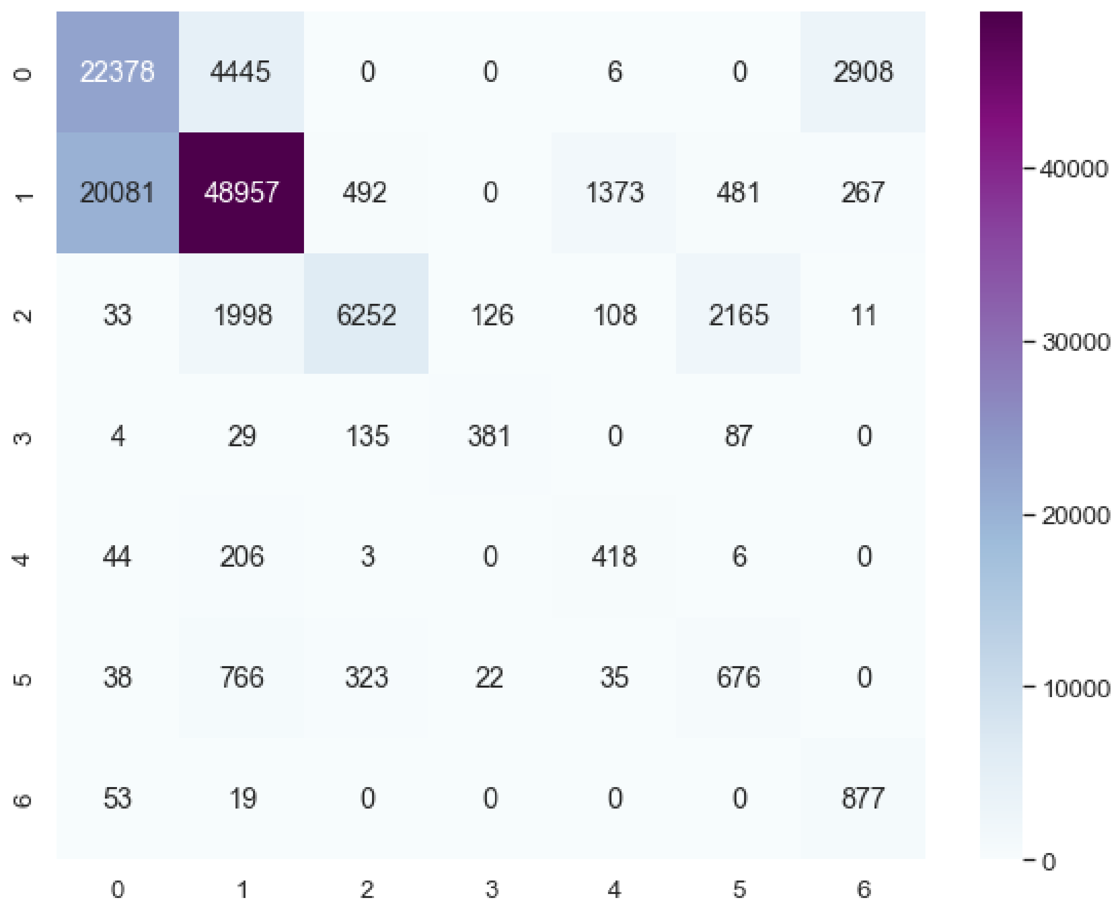
```
The optimal number of neighbours = 3.
```

[46]:
```python
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

#prediction
y_pred = knn.predict(X_test)

#score
print("Accuracy -- ", knn.score(X_test, y_test)*100)

#confusion
cm = confusion_matrix(y_pred, y_test)
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.2)
sns.heatmap(cm, annot=True, fmt='g',cmap="BuPu")
plt.show()
```

```
Accuracy --  96.95790986463344
```

[13]:
```
#neural nw
NN=MLPClassifier()
NN.fit(X_train, y_train)

#prediction
y_pred = NN.predict(X_test)

#score
print("Accuracy -- ", NN.score(X_test, y_test)*100)

#confusion
cm = confusion_matrix(y_pred, y_test)
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.2)
sns.heatmap(cm, annot=True, fmt='g',cmap="BuPu")
plt.show()
```

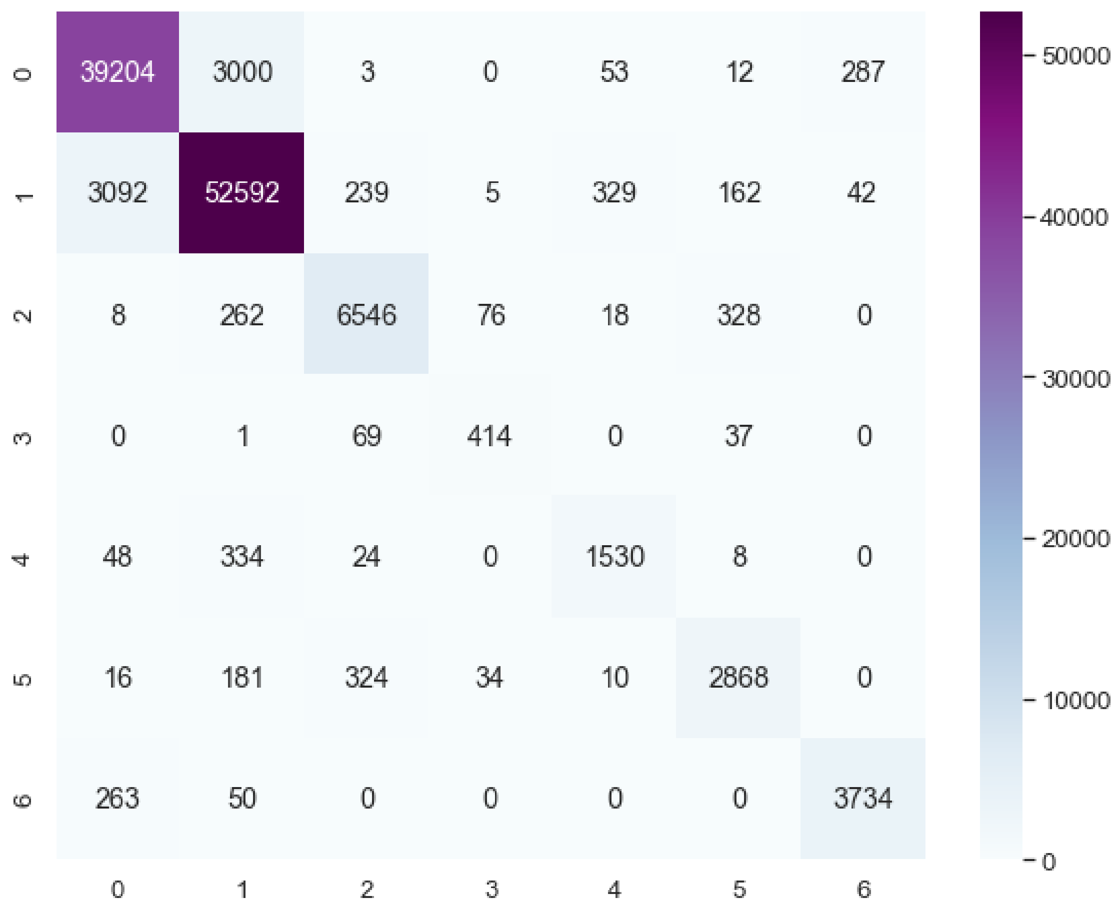
```
Accuracy --  68.79254408233867
```



[14]:

```
#decision tree
DT=DecisionTreeClassifier()
DT.fit(X_train, y_train)

#prediction
y_pred = DT.predict(X_test)

#score
print("Accuracy -- ", DT.score(X_test, y_test)*100)

#confusion
cm = confusion_matrix(y_pred, y_test)
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.2)
sns.heatmap(cm, annot=True, fmt='g',cmap="BuPu")
plt.show()
```

```
Accuracy --  91.98385583848953
```



[15]:

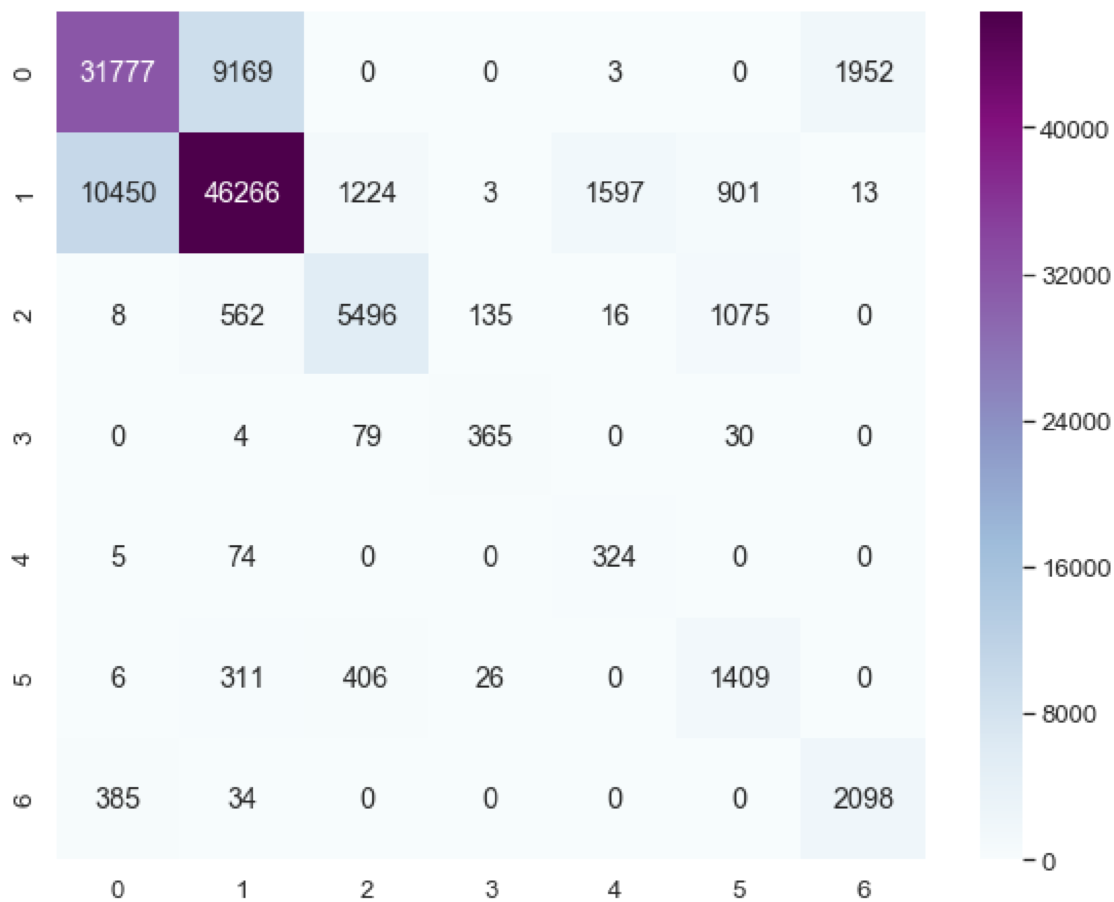
```
#bagging
BC=BaggingClassifier()
BC.fit(X_train, y_train)

#prediction
y_pred = BC.predict(X_test)

#score
print("Accuracy -- ", BC.score(X_test, y_test)*100)

#confusion
cm = confusion_matrix(y_pred, y_test)
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.2)
sns.heatmap(cm, annot=True, fmt='g',cmap="BuPu")
plt.show()
```

```
Accuracy --  95.27636980112389
```

[16]:

```
#gradient boosting
GC=GradientBoostingClassifier()
GC.fit(X_train, y_train)

#prediction
y_pred = GC.predict(X_test)

#score
print("Accuracy -- ", GC.score(X_test, y_test)*100)

#confusion
cm = confusion_matrix(y_pred, y_test)
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.2)
sns.heatmap(cm, annot=True, fmt='g',cmap="BuPu")
plt.show()
```

```
Accuracy --  75.50149307677081
```



[36]:

```
#algo
#X.head()
from sklearn.ensemble import RandomForestClassifier
RFC = RandomForestClassifier(n_estimators=100)
RFC.fit(X_train, y_train)

#prediction
y_pred = RFC.predict(X_test)

#score
print("Accuracy -- ", RFC.score(X_test, y_test)*100)

#confusion
cm = confusion_matrix(y_pred, y_test)
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.2)
sns.heatmap(cm, annot=True, fmt='g',cmap="BuPu")
plt.show()
```

```
Accuracy --  95.01906147001368
```

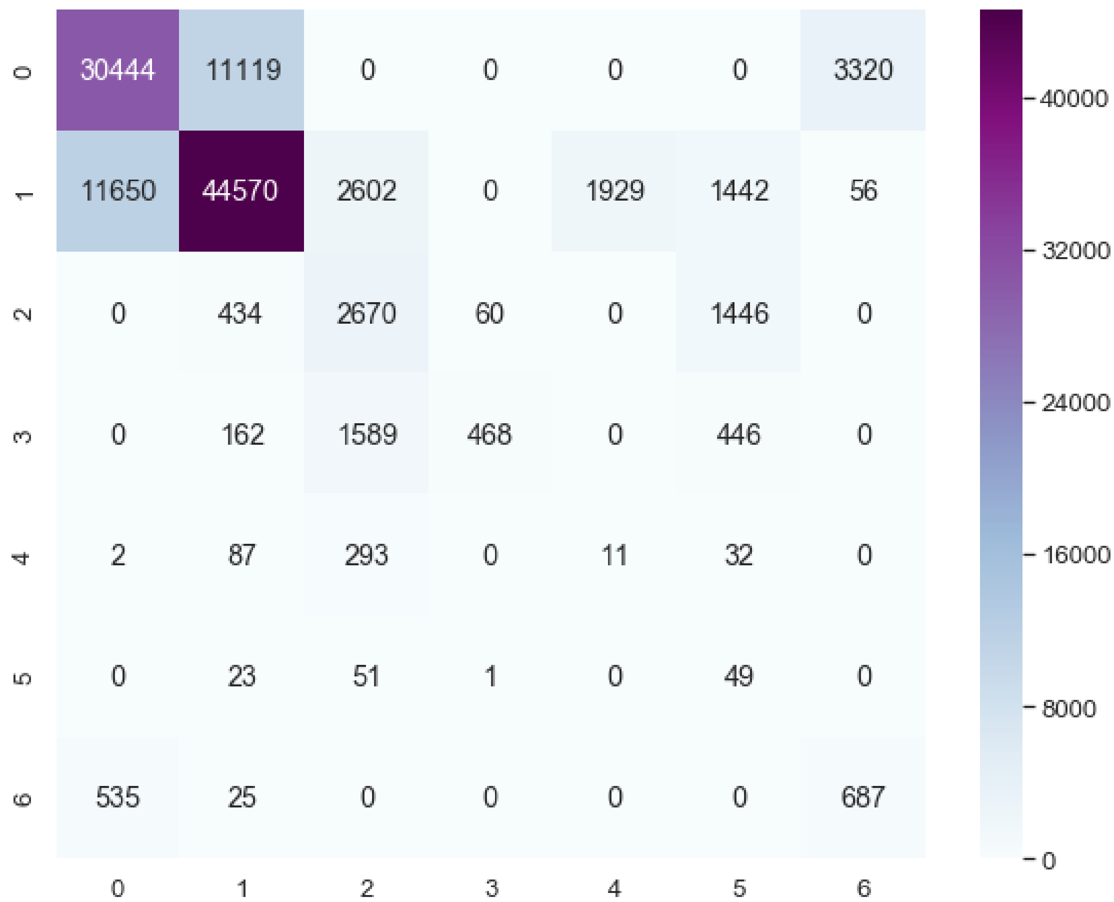
[37]:

```
#algo
#X.head()
#LogisticRegression
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression()
LR.fit(X_train, y_train)

#prediction
y_pred = LR.predict(X_test)

#score
print("Accuracy -- ", LR.score(X_test, y_test)*100)

#confusion
cm = confusion_matrix(y_pred, y_test)
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.2)
sns.heatmap(cm, annot=True, fmt='g',cmap="BuPu")
plt.show()
```

```
Accuracy --  68.01889796304744
```

[38]:
```python
#LDA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
LDA = LinearDiscriminantAnalysis()
LDA.fit(X_train, y_train)

#prediction
y_pred = LDA.predict(X_test)

#score
print("Accuracy -- ", LDA.score(X_test, y_test)*100)

#confusion
cm = confusion_matrix(y_pred, y_test)
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.2)
sns.heatmap(cm, annot=True, fmt='g',cmap="BuPu")
plt.show()
```

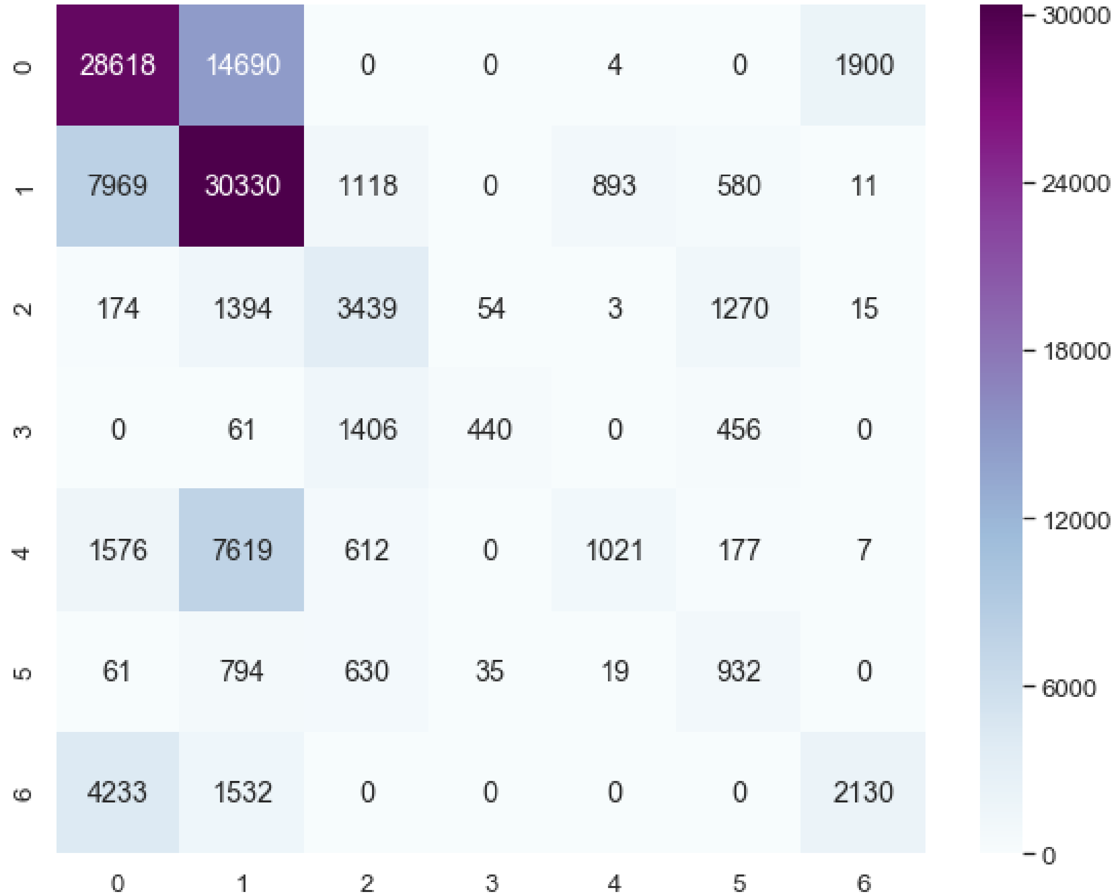
```
Accuracy --  67.89755858282489
```



[39]:

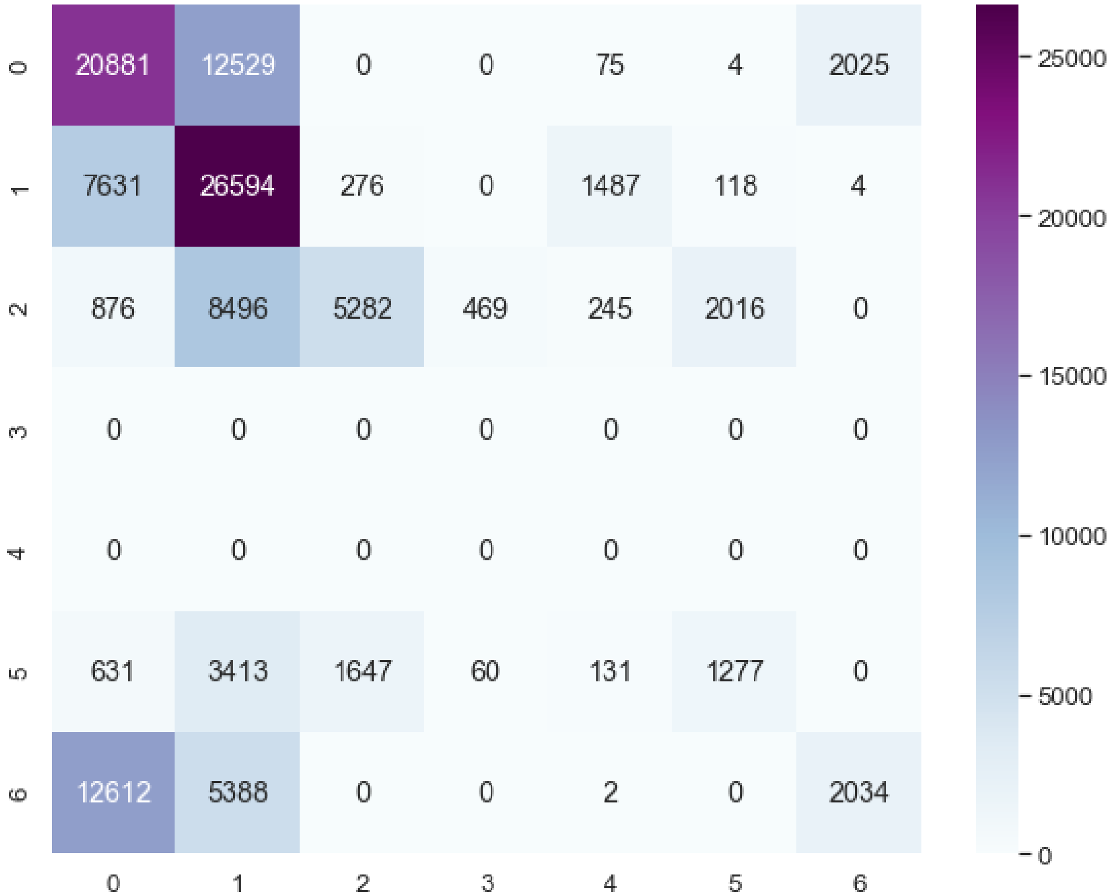
```python
#Gaussian NB
from sklearn.naive_bayes import GaussianNB
GN = GaussianNB()
GN.fit(X_train, y_train)

#prediction
y_pred = GN.predict(X_test)

#score
print("Accuracy -- ", GN.score(X_test, y_test)*100)

#confusion
cm = confusion_matrix(y_pred, y_test)
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.2)
sns.heatmap(cm, annot=True, fmt='g',cmap="BuPu")
plt.show()
```

```
Accuracy --  57.58026901198764
```



[40]:
```python
#AD
from sklearn.ensemble import AdaBoostClassifier
AD = AdaBoostClassifier()
AD.fit(X_train, y_train)

#prediction
y_pred = AD.predict(X_test)

#score
print("Accuracy -- ", AD.score(X_test, y_test)*100)

#confusion
cm = confusion_matrix(y_pred, y_test)
plt.figure(figsize=(10, 8))
sns.set(font_scale=1.2)
sns.heatmap(cm, annot=True, fmt='g',cmap="BuPu")
plt.show()
```

```
Accuracy --  48.250045179556466
```

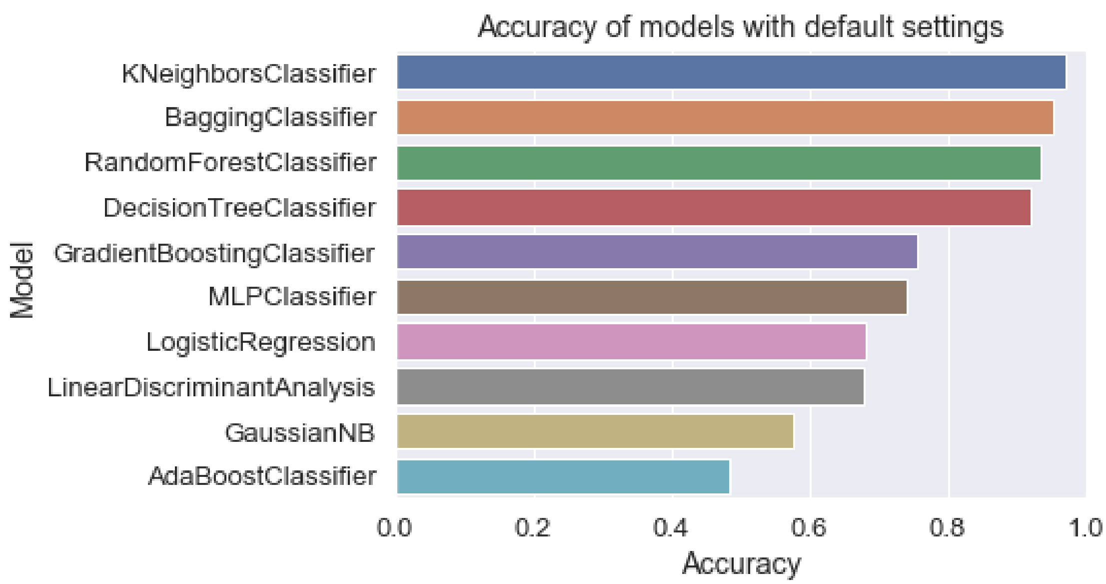


[21]:
```python
models = [LogisticRegression(),
          KNeighborsClassifier(),
          DecisionTreeClassifier(),
          AdaBoostClassifier(),
          BaggingClassifier(),
          GradientBoostingClassifier(),
          RandomForestClassifier(),
          GaussianNB(),
          LinearDiscriminantAnalysis(),
          MLPClassifier()]

# Gather metrics here
accuracy_by_model={}

# Train then evaluate each model
for model in models:
    model.fit(X_train, y_train)
```

```
    y_pred = model.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    # Fill metrics dictionary
    model_name = model.__class__.__name__
    accuracy_by_model[model_name]=score
```

```
/Users/rudranibhadra/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/rudranibhadra/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/logistic.py:469: FutureWarning: Default
multi_class will be changed to 'auto' in 0.22. Specify the multi_class option to
silence this warning.
  "this warning.", FutureWarning)
/Users/rudranibhadra/anaconda3/lib/python3.7/site-
packages/sklearn/ensemble/forest.py:245: FutureWarning: The default value of
n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

[25]:
```
# Draw accuracy by model chart
acc_df = pd.DataFrame(list(accuracy_by_model.items()), columns=['Model',␣
 ↪'Accuracy']).sort_values('Accuracy', ascending=False).reset_index(drop=True)
acc_df.index=acc_df.index+1
sns.barplot(data=acc_df,y='Model',x='Accuracy')
plt.xlim(0,1)
plt.title('Accuracy of models with default settings')
#plt.xticks(rotation=45)
plt.show()

# Print table
acc_df
```



[25]:

	Model	Accuracy
1	KNeighborsClassifier	0.969304
2	BaggingClassifier	0.953383
3	RandomForestClassifier	0.935535
4	DecisionTreeClassifier	0.919486
5	GradientBoostingClassifier	0.755015
6	MLPClassifier	0.741014
7	LogisticRegression	0.680189
8	LinearDiscriminantAnalysis	0.678976
9	GaussianNB	0.575803
10	AdaBoostClassifier	0.482500

[ ]: