|  |  |
| --- | --- |
|  | |
| Python Machine Learning Project  Tools for Data Analytics CA TWO – Report | |
|  | |
| Ciaran Finnegan / Radoslav Durina / Dermot Madsen  Student No : 10524150 / 10524378 / 10522567  18/10/2019 |  |
|  |  |

List of contents

1 Project Overview 3

1.1 High Level Description 3

1.2 Environment Assumptions 3

1.3 Project Execution Instructions 3

2 Machine Learning Process Flow Description 4

2.1 High Level Description of Machine Learning Workflow 4

3 Data Pre-Processing and Preparation 6

3.1 Python Libraries 6

3.2 What is data pre-processing? Why do we do it? 6

3.3 Load and Visualise Data 6

3.4 Types of Data and performing pre-processing 8

4 Model Evaluation Strategy 13

4.1 Benefit of Random Forest 13

4.2 Objective of Model evaluation 13

5 Model Building and Testing 15

5.1 Dividing the original dataset into train/validation/test sets 15

5.2 Tuning the RandomForest model for the Spruce dataset 16

5.3 Determine the subset of ‘significant features’ 18

6 Identifying The Best Model 20

6.1 Run RandomForest using ‘significant’ dataset 20

6.2 Final Classification Model Characteristics 21

7 Guidelines Generated by Prediction Model 25

7.1 Focus on a smaller Feature set of data 25

7.2 Random Forest Parameter 25

7.3 Check the ‘Missing Data’ 25

8 Principal Component Analysis (PCA) 26

8.1 Code and Data Preparation 26

8.2 PCA Implementation 27

9 K-Means Clustering 28

9.1 Elbow Plot Creation 28

9.2 K-Means Implementation 29

# Project Overview

## High Level Description

This document covers the design, implementation and observations on all parts of CA Two for the Tools for Data Analytics module.

This is a group report, containing the collaborative design/coding decisions and general observations, and reflections of each team member; Dermot, Radoslav, and Ciaran.

## Environment Assumptions

The IDEs used for development were a combination of Microsoft Visual Studio Community 2019, PyCharm Community 2016.3, and Jupyter Notebook (Anaconda 3). The code is submitted in one Python ***Surname\_Surname\_Surname.py*** file (*Madsen\_Durina\_Finnegan.py*), along with the source ‘Spruce’ csv file, and this report document (in .pdf format).

There were very minor code snippets added to the Python code base to improve graph presentations, depending on which IDE the individual contributors were using to develop and test the CA Two code and analysis.

## Project Execution Instructions

The procedure for the project solution is executed from the ‘*Madsen\_Durina\_Finnegan.py*’ file. For Visual Studio;

1. Add the *.py* file into a VS Project, include the *Spruce.csv* file in the same project folder.
2. Right click in the Solution Explorer list
3. Choose the option to ‘Set as Startup File’.
4. Run the file

The problem solution files follow the following structure;

* The main function is named with the format: *MainProg\_CATwo()*.
* The main program contains all the high level function calls to invoke the Machine Learning workflow steps. The code for all functions is included in the ‘*Madsen\_Durina\_Finnegan.py*’ file.

# Machine Learning Process Flow Description

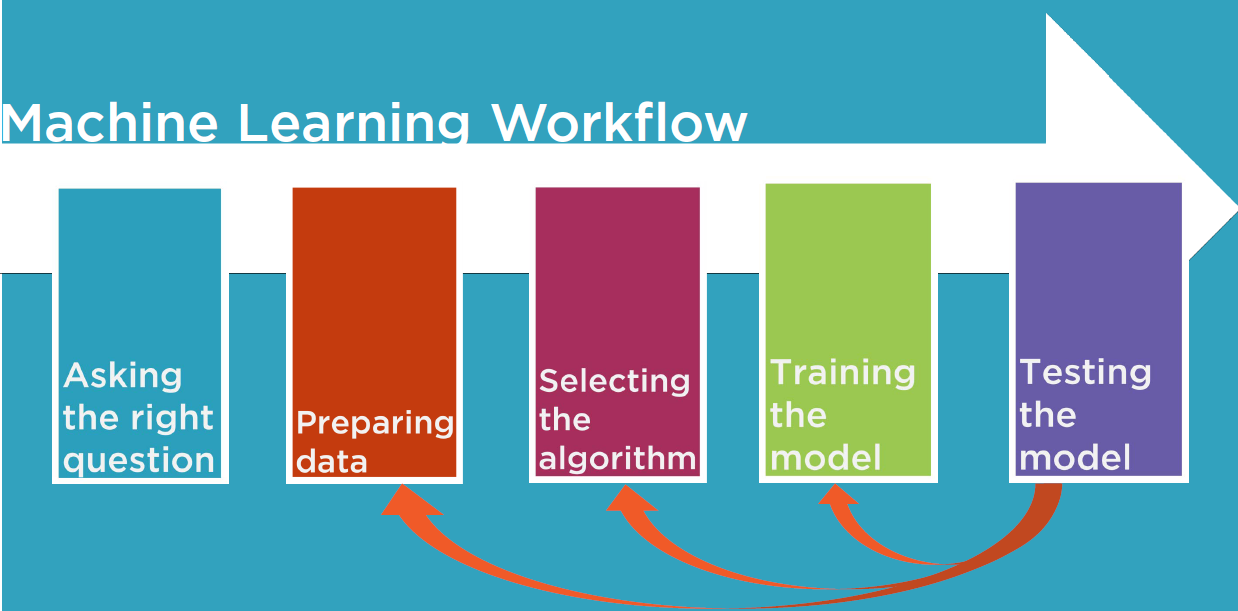
## High Level Description of Machine Learning Workflow

### Process Flow Diagram for Machine Learning

This continuous assessment is an exercise in the ability to analyse datasets using Python.

Research shows that this process, whether using Python or other tools, tends to follow orchestrated and repeatable patterns. This leads the analyst to systematically transform and process information to create prediction solutions.

A representation of this process workflow can be given as..

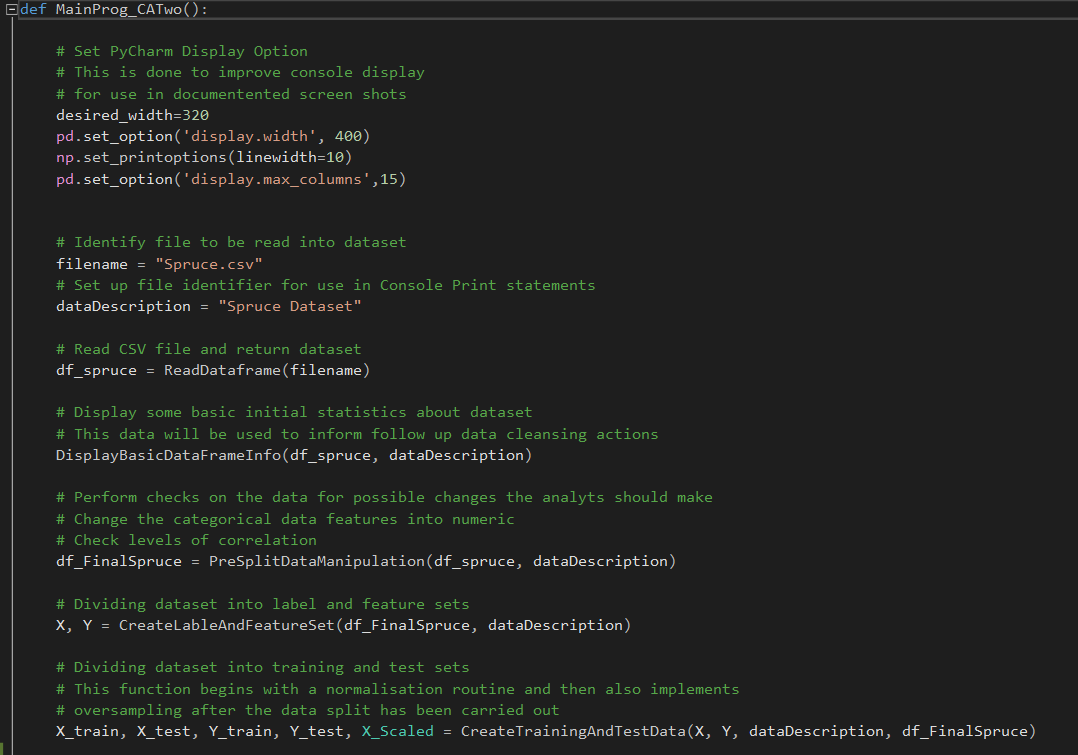


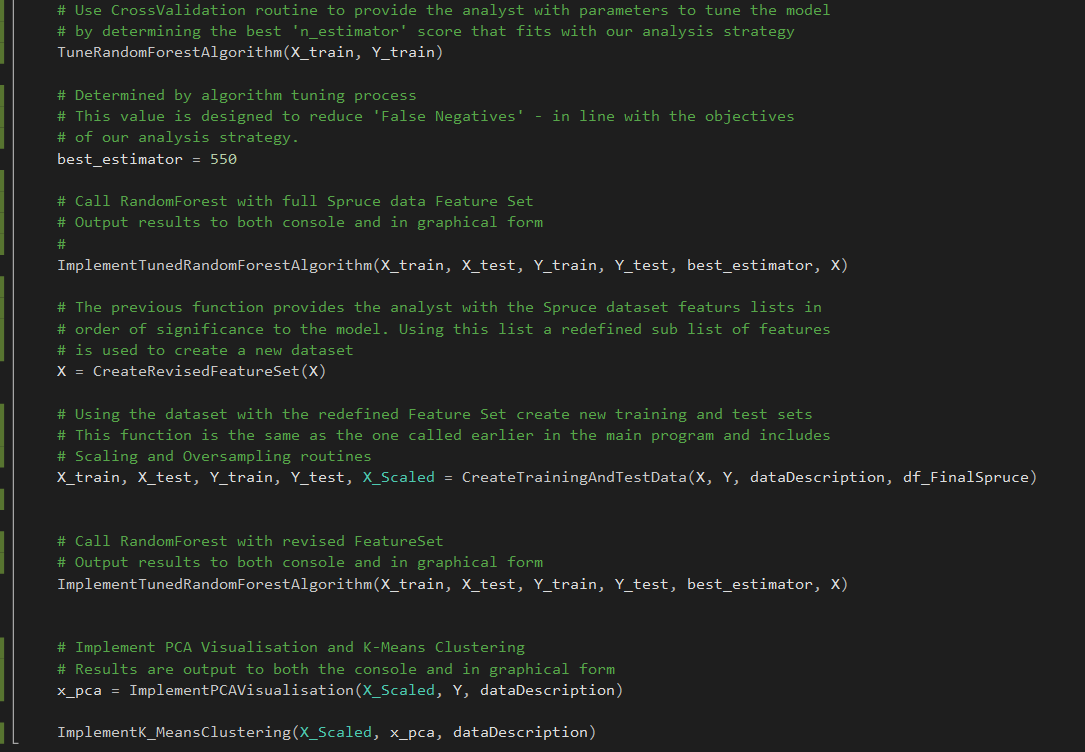
*\*credit to Jerry Kurata - Pluralsight*

* The Right Question: For the Spruce dataset analysis we were given the question - how to predict locations to plan Spruce trees for the Canadian Forest Department. We needed to supplement this by determining if we should focus on accuracy, or reducing certain types of false predictions for planting Spruce trees.
* Preparing the Data: What routines do we need to code in Python to manipulate the Spruce data? We need to make the data work effectively while constructing a model using the Random Forest algorithm in Python.
* Selecting the Algorithm: The Random Forest approach is a good fit for this problem. We expand on this statement with more detail later in this report.
* Training the Model: Python provides many libraries to split and train the model using the Spruce data. This is elaborated on in the report.
* Testing the Model: Python routines execute tests on the trained model and represent out both in the console and graphically.

The output from various Python functions we wrote helped inform decisions to refine parameters for Python functions and reiterate through the training the testing process.

### High Level Program Flow Sequence (code snippet from main function).



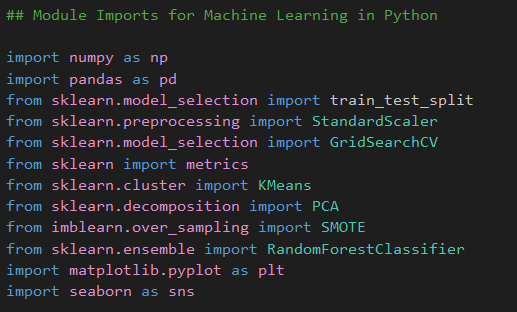


# Data Pre-Processing and Preparation

## Python Libraries

There are a number of libraries which can be imported into a Python application to provide access to pre-built Machine Learning functions.

The libraries imported into our CA applications are:



## What is data pre-processing? Why do we do it?

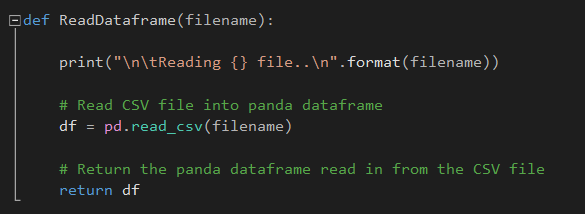
Data pre-processing is a process of cleaning the ‘raw’ data, that is the data is collected in the real world, and converting to a ‘clean’ data set.

The Spruce data in the ***Spruce.csv*** file is reasonably well formed but still requires certain steps to be executed to convert the data into a small clean data set.

Data pre-processing is required to achieve good results from the training and application of the subsequent model in our Spruce machine learning project.

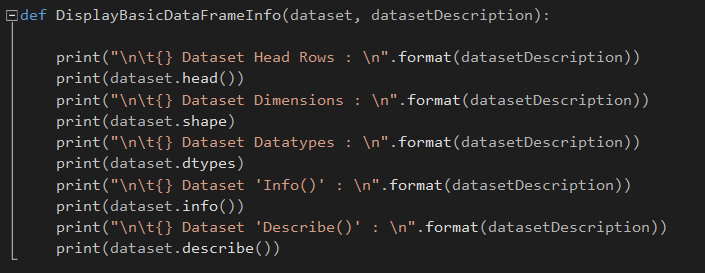
## Load and Visualise Data

The Python code used the Panda library to read the Spruce data into a Panda dataframe.

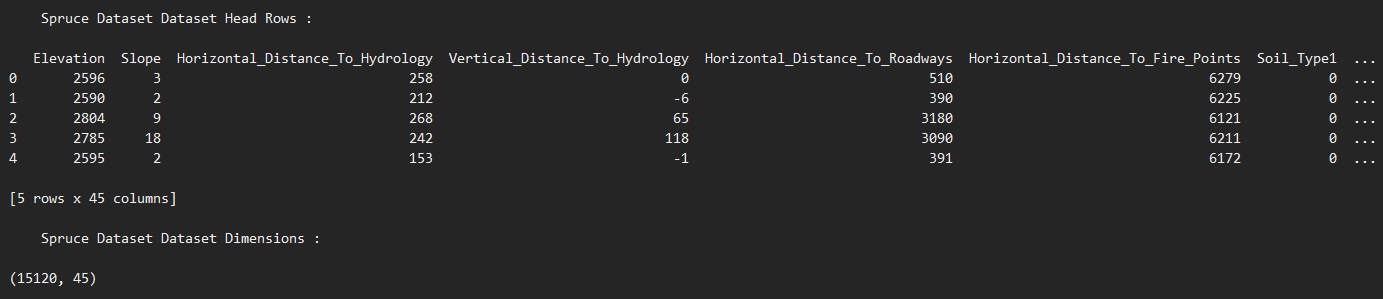


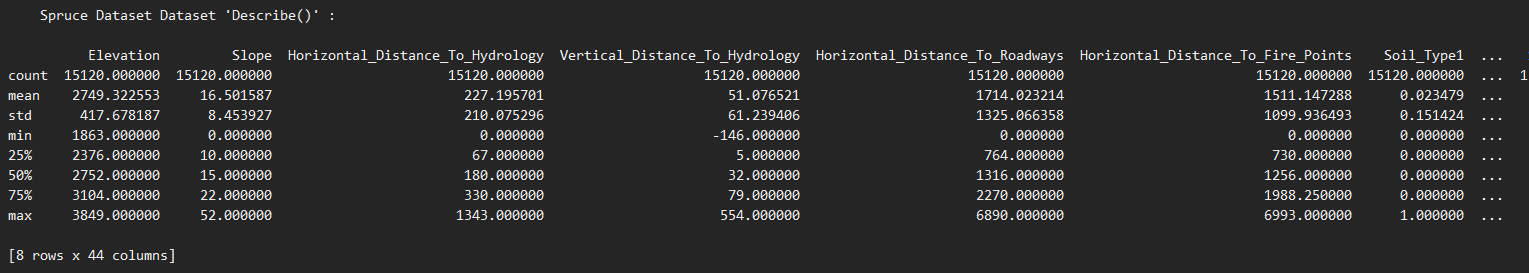
Once the data is in a dataframe format we can start to display and manipulate the data in our Spruce tree Python project.

We wrote the following function in Python to display basic information on the initial Panda dataset created from the direct read of the ***Spruce.csv*** file.



A partial view of the Visual Studio console output looks like this…





## Types of Data and performing pre-processing

Once our Python application loaded data from the CSV file into the Panda dataframe, we were able to proceed with further analysis to determine what ‘clean up’ was required.

### Three Types of Data

In general terms ‘real world’ data types we would be concerned with are:

1. Numeric e.g. income, age

2. Categorical e.g. gender, nationality

3. Ordinal e.g. low/medium/high

In the ***Spruce.csv*** file we only need to contend with numerical and categorical date.

### How can data pre-processing be performed?

These are some of the basic pre — processing techniques that can be used to convert raw data.

1. **Conversion of data:** As we know that Machine Learning models can only handle numeric features, hence categorical and ordinal data must be somehow converted into numeric features. For the ***Spruce.csv*** file, we only have one categorical column to consider: ‘Tree\_Type’.

2. **Ignoring the missing values:** Whenever we encounter missing data in the data set then we can remove the row or column of data depending on our need. This method is known to be efficient but it shouldn’t be performed if there are a lot of missing values in the dataset. There are no NULL values in the ***Spruce.csv*** file. There are a number of zero values that could potentially be evaluated as missing but we will choose to not take any actions for this data.

3. **Filling the missing values:** Whenever we encounter missing data in the data set then we can fill the missing data manually, most commonly the mean, median or highest frequency value is used. This could be an option for the ‘zero’ values in the Spruce dataset but we choose not to take any actions in this project.

4.**Machine learning:** If we have some missing data then we can predict what data shall be present at the empty position by using the existing data. As above, we take no specific actions to address the potentially ‘missing’ values – the zero entries in the Spruce dataset.

5. **Outliers detection:** There are some error data that might be present in our data set that deviates drastically from other observations in a data set. [Example: human weight = 800 Kg; due to mistyping of extra 0]. We did not check for outliers in the Spruce dataset.

### Python data processing code used on Spruce dataset

We coded the following analysis to check the Spruce data for clean-up actions:

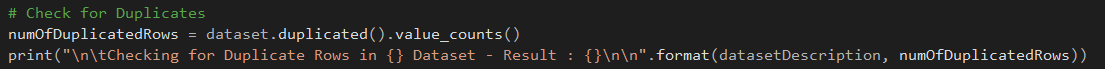
**Null Data**



The console output shows that the Spruce dataset has no null values. Hence, no clean up actions are required, such as removing rows from the dataset.



**Duplicate Data**



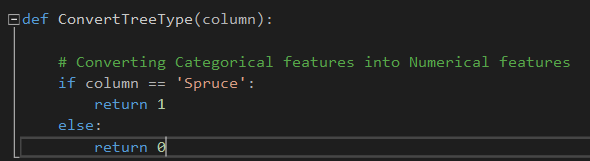
The console output shows that the Spruce dataset has no duplicate entries.



**Converting Categorical Types**



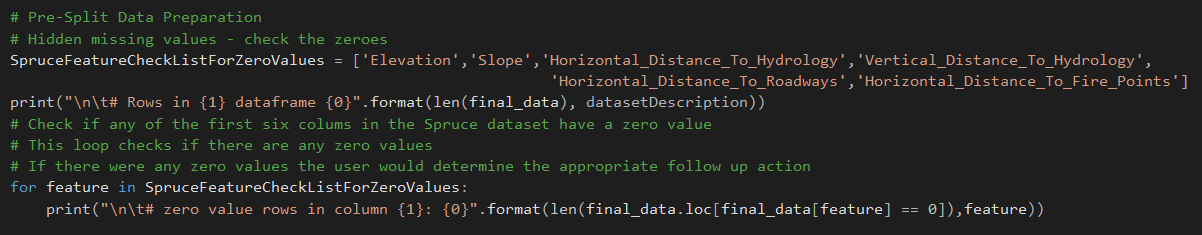
The ‘Tree\_Type’ feature contained the value ‘Other’ or ‘Spruce’. The following function was applied to convert this to a ‘0’ or ‘1’ numerical value.



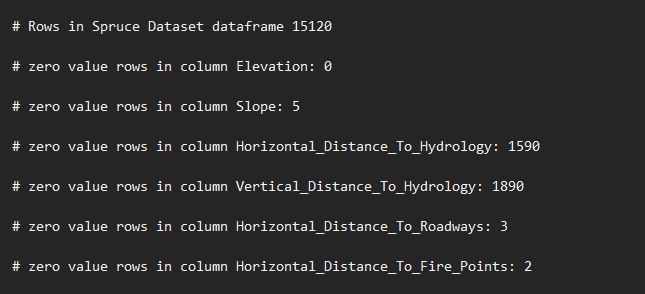
The ‘Tree\_Type’ is the target variable, which we are trying to predict. Converting the column to a number is necessary to allow the Random Forest algorithm process the dataset.

**Checking for Zero Values**

We added these lines of Python code to check the Spruce dataset for zero values.



It only seemed logical to check the first six features for zero values. The console out was as follows;



There are no zero entries for ‘Elevation’ but at least some of the other five columns show zero entries.

In the real world we could apply domain knowledge to assess if a zero value is a realistic number for any of these features.

To avoid unnecessary bias being introduced into the model, we could then decide to take action by either:

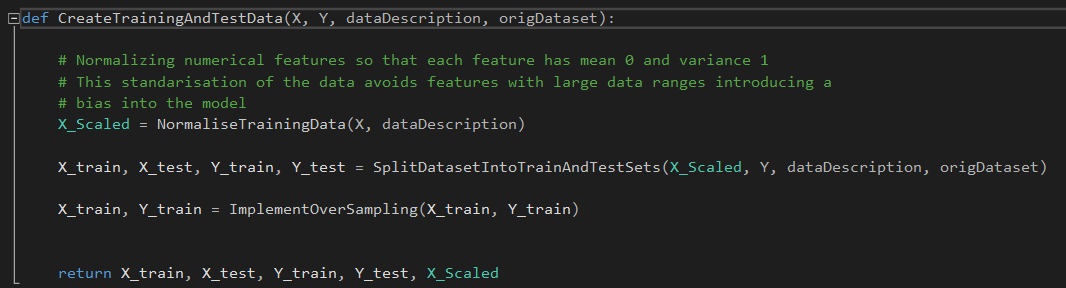
* Remove the rows with zero values
* ‘Impute’ and replace zero values, possibly with the mean values of the non zero entries in the feature.

In the Spruce dataset we choose to take no action, and assume that the zero values were reflective of real world data.

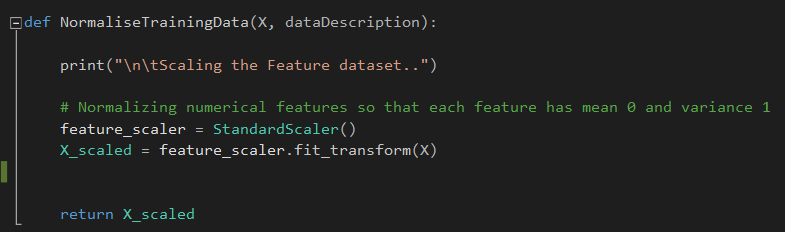
**Scaling and OverSampling The Data**

We created a Python function to call the routines to split the Spruce dataset, as described in Section 5 of this document.

The function ran additional data manipulation around the splitting of the Spruce dataset.



The Scaling routine in Python was as follows:

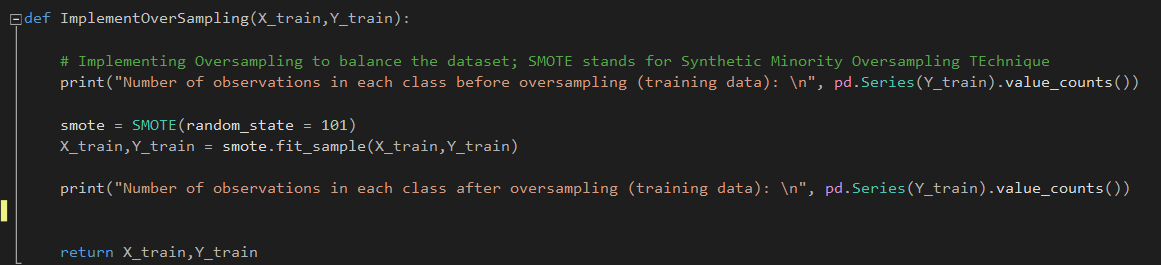


The purpose of this code is to adjust values in all the features of the dataset because a number of these features use a different range of actual numerical values.

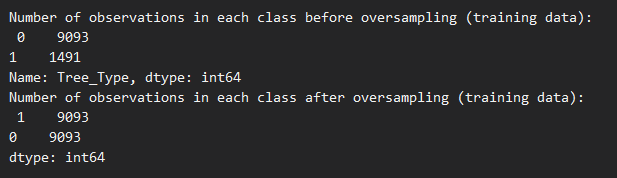
If we did not scale, or normalise, this data then the features with a large range of values would probably introduce bias into the model and effect accuracy.

By reducing all features to a range of values between -1 and 1, we prevent features with a larger variance in data values from introducing bias.

The OverSampling code is as follows:



A sample of the console output looks like:



The Spruce dataset has an approximate split of Spruce to Other Tree ratio of 15/85.

We can see this ratio replicated in the split of the Spruce dataset into Training and Test sets. In order to balance the data set, and improve model accuracy, a Python library (SMOTE) has been invoked.

# Model Evaluation Strategy

## Benefit of Random Forest

The dataset ***Spruce.csv*** contains cartographic data for observations made over different 30m × 30m patches in the forests of Alberta, Canada. This dataset has 15,120 observations, with 44 input variables (cartographic variables) and 1 target variable (Tree\_Type).

This Spruce dataset allows for Supervised Learning and is a ‘classification’ problem because we are required to advise Canada’s Forest Department whether they should plant Spruce or another tree type in a given 30m x 30m plot.

In Machine Learning, Random Forest is a commonly used and versatile algorithm to generate a model for this type of problem.

Given the characteristic (list of features) for a new forest plot we are aiming to generate a prediction on whether Spruce trees should be planted, based on previous real world data.

## Objective of Model evaluation

Section 5 of this document describes the process of ‘training’ a model to predict results.

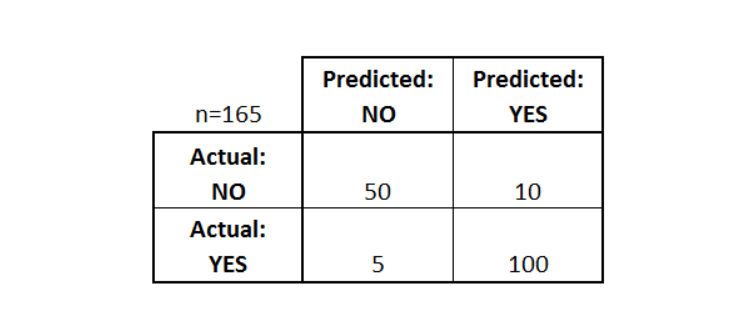
Once the model is trained we can use the same trained model to predict using the testing data i.e. the unseen data. Once this is done we can develop a confusion matrix, this tells us how well our model is trained. A confusion matrix has 4 parameters, which are ‘**True positives’**,**‘True Negatives’**,**‘False Positives’**and ‘**False Negatives’**. We prefer that we get more values in the True Negatives and True Positives to get a more accurate model. The size of the Confusion matrix completely depends upon the number of classes.

* **True positives:** These are cases in which we predicted TRUE and our predicted output is correct.
* **True negatives:** We predicted FALSE and our predicted output is correct.
* **False positives:** We predicted TRUE, but the actual predicted output is FALSE.
* **False negatives:** We predicted FALSE, but the actual predicted output is TRUE.

We can also find out the accuracy of the model using the confusion matrix.

*Accuracy = (True Positives +True Negatives) / (Total number of classes)*

An example of a Confusion Matrix with random sample data can be shown as: \*



*\*Credit* [https://towardsdatascience.com](https://towardsdatascience.com/)

Accuracy in the above mode would be equal to (100+50) / 165 = 0.9090 (90.9% accuracy).

Note: the above is random example data. It is **not** based on Spruce dataset.

The objectives we considered were:

* Should we look for the highest possible classification accuracy?
* Should we look to minimise ‘false positives’?
* Should we look to minimise ‘false negatives’?
* Should we try and balance the level of false negatives and false positives?

Our decision for the analysis objective was to **reduce False Negatives**. We based our decision on the assumption that Canada’s forest Department would prefer not to miss an opportunity to plant Spruce on a given plot.

The Spruce tree is a valuable commodity so they want to avoid as much as possible planting a non-Spruce in a plot that would support spruce tree.

We therefore adjusted our code to search for the optimum parameter to reduce False Negatives.

# Model Building and Testing

## Dividing the original dataset into train/validation/test sets

For training a model we initially split the Spruce model into 3 three sections which are ‘**Training data**’,‘**Validation data**’ and ‘**Testing data**’.

We wrote Python code to train the classifier using a ‘**training data set**’, the tuned the parameters using ‘**validation set**’ , invoked by another Python function, and then tested the performance of our Spruce classifier on the unseen ‘**test data set**’.

During the training of the classifier only the Spruce training and/or validation set is used. The Spruce test data set was not used during the training of the classifier. The Spruce test set was only used to test the model accuracy.

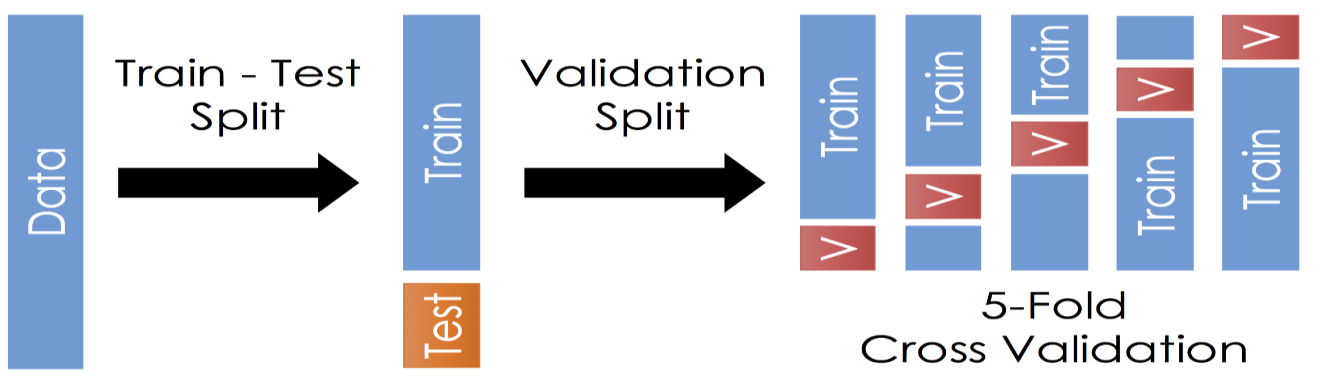
We used a 5-fold cross validation function in our code:

**Training set:** The training set is the material through which the computer learns how to process information. Machine learning uses algorithms to perform the training part. A set of data used for learning that is to fit the parameters of the classifier.

**Validation set:** Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. A set of unseen data is used from the training data to tune the parameters of a classifier.

**Test set:** A set of unseen data used only to assess the performance of a fully-specified classifier.

**This can be graphically represented like this:**



*\*Credit* [https://towardsdatascience.com](https://towardsdatascience.com/)

Once the data is divided into the 3 given segments we can start the training process.

## Tuning the RandomForest model for the Spruce dataset

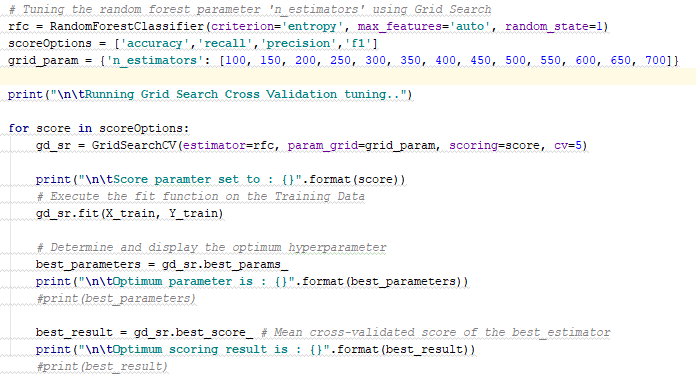
In our Python code – ***TuneRandomForestAlgorithm(X\_train, Y-train)*** - a CrossValidation routines was invoked on the training set to implement a 5-fold cross validation exercise.

The code snippet below shows the main body of our function providing a value for a hyperparameter to be applied to the model in order to optimise results.

The FOR LOOP runs through each of the following scoring options:

* Scoring = 'accuracy' when you want to maximize prediction accuracy
* Scoring = 'recall' when you want to minimize false negatives
* Scoring = 'precision' when you want to minimize false positives
* Scoring = 'f1' when you want to balance false positives and false negatives (place equal emphasis on minimizing both)

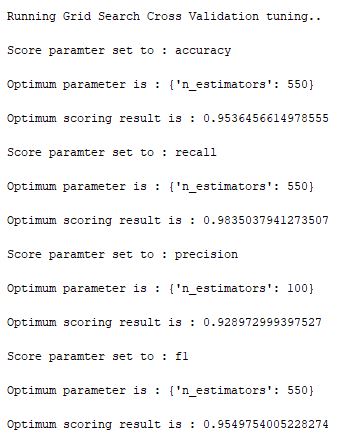
‘Scoring’ is a parameter option used in the *GridSearchCV* function and will return a hyperparameter value optimise the model for a particular objective goal.



The ‘n\_estimator’ value list is an incremental list reflecting the staggered list of option for the tree depth for the algorithm.

Each ‘scoring’ parameter is executed against this range of steps to provide and ‘n-estimator’ value that will optimise the model for the scoring objective.

This code took some time to run when executed and produced the following console output:



The ‘n\_estimator’ value will provide the max tree depth tuned for use in the Random Forest model when it is applied against the Spruce training dataset with a particular scoring objective.

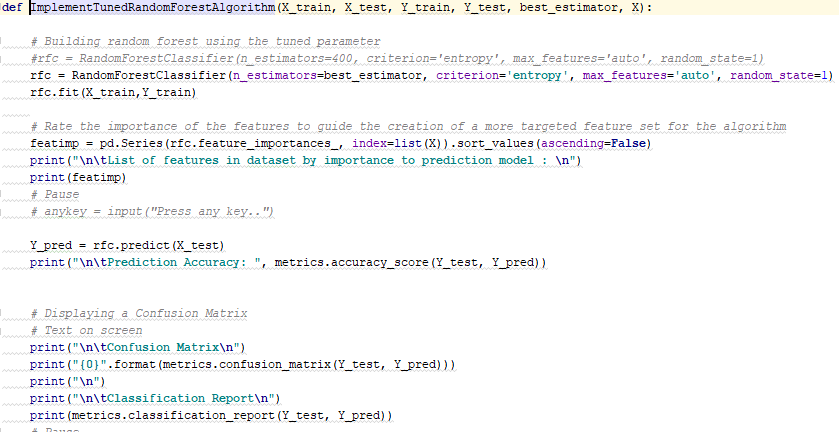
This use and purpose of this ‘hyperparameter’ is further described in Section 6.2 of this document.

## Determine the subset of ‘significant features’

### Run the RandomForest Algorithm against Full Spruce Training set

As our analysis strategy is to focus on the reduction of False Negatives, we passed the ‘*n\_estimator’* parameter of the RandomForestClassifier function a value of ‘**550**’

(We used the ‘*best\_estimator’* variable in our Python code to store this value).



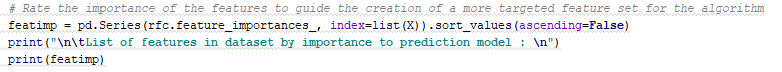
The first time we ran the Random Forest algorithm against the Spruce dataset we used the full list of features, as read in from the Spruce.csv file and subsequently ‘cleaned up’.

### Create Revised Featureset

Not every feature is going to have an impact on the accuracy of the model and some may have very effect on the eventual performance.

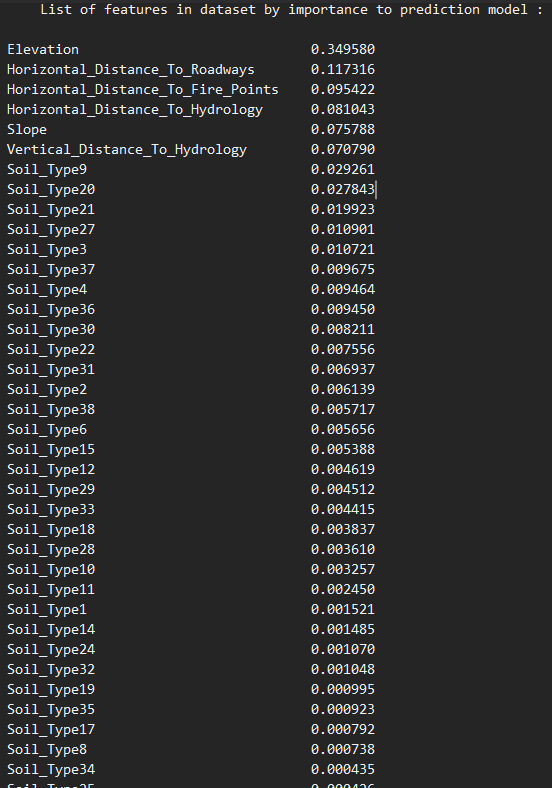
The Random Forest code invokes a function to generate a list of features, which is sorted by the relevance to the production model.

The code sniper below shows the function assessing the importance of each feature and providing a list for the analyst to review.



The screenshot below shows a partial list of features produced in the Visual Studio console. It represents a scoring of all features.

The features with most relevance are sorted at the top of the list, which assists the analyst in making an informed decision on which features are most ‘relevant’.



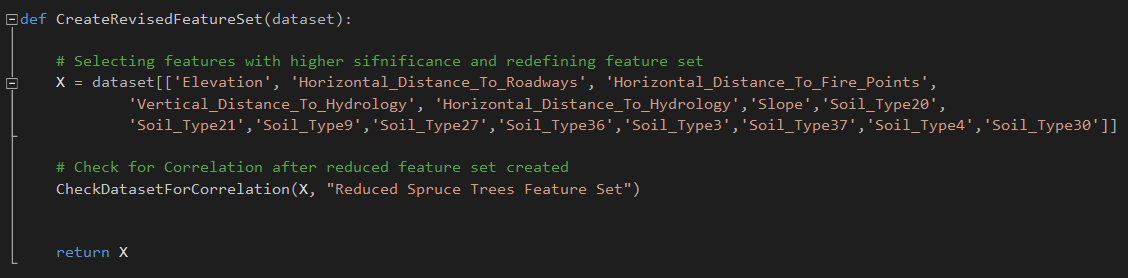
# Identifying The Best Model

## Run RandomForest using ‘significant’ dataset

We made an initial assumption that the features with a scare less than 0.01 could be ignored for the purposes of building a model.

However, we ran multiple iterations of the Random Forest model with a subset of the feature list, based on feature relevance, with a view to finding the smallest list that would meet our requirement of reducing False Negatives.

Based on these iterations we wrote the following function in Python to provide a reduced Feature set.



This will allow us to run a more performant analysis and we can compare the accuracy of the model predictions against test data to ensure that no significant accuracy has been lost by using this reduced dataset.

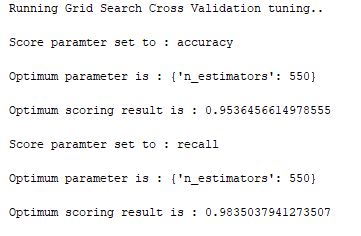
The benefit of a reduced feature set is that we can work with a smaller dataset, which is easier to manipulate, more performant to execute, and avoids less important features in future real world data introducing any bias.

## Final Classification Model Characteristics

### Features and Parameters

Using the Feature set described above in Section 6.1, we will also run the final Spruce classification model using the ‘n\_estimator’ value as described in the tuning process in Section 5.2 of this document.

In the console window, after the Grid Search Cross Validation tuning was executed, the follow value for the n\_estimator was displayed.



Our objective is to improve overall model accuracy and the reduction of false negatives (‘recall’).

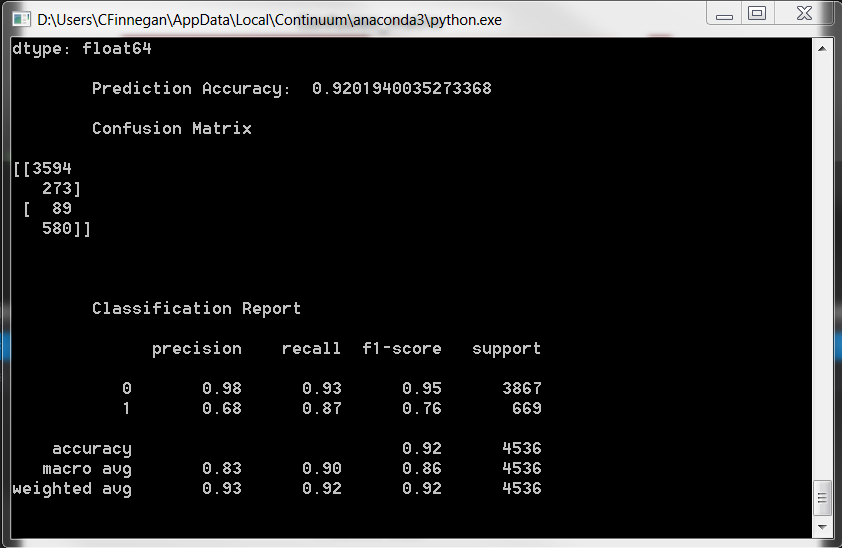
The ‘n\_estimators’ value of 550, as taken from our incremental list of options from 100 – 700 (increasing in increments of 50), will provide the optimum parameter value for both accuracy and recall.

This value represents the optimum tree depth to be used for the RandomForest algorithm. This hyperparameter will improve the accuracy of the model run against the test data (or any ‘real world’ data).

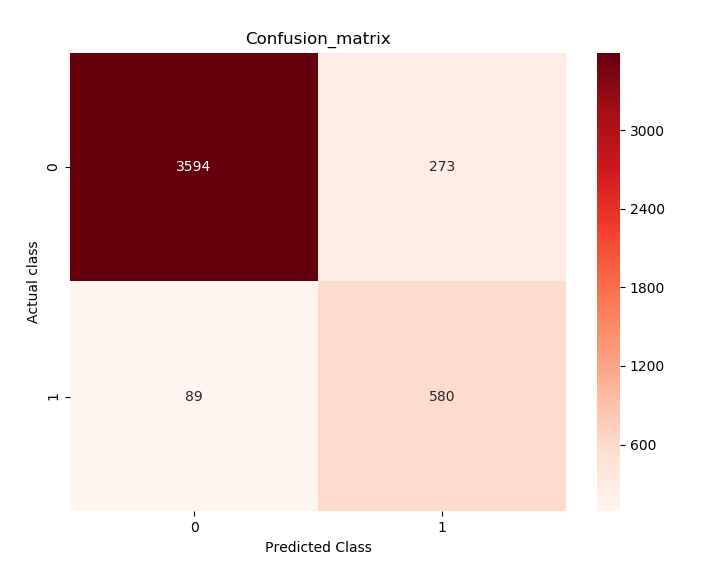
This hyperparameter should prevent ‘overfitting’ of the data by the Random Forest algorithm (the concept will to many ML algorithms). Thus we are attempting to avoid a situation that the model fits the training data very well but is a poor predictor against the test data, and hence other real world examples.

### Performance Results on Test Set

The performance results from the tuned model are output to the Visual Studio console as follows;



The Graph results were presented by our program like this:



The Model prediction scores 92.02% against the test data, with an 87% success on minimising false negatives.

The confusion matrix seen in the graph shows the following figures:

**Confusion matrix:**

**[[3594 273]**

**[ 89 580]]**

**TP: 3594**

**TN: 580**

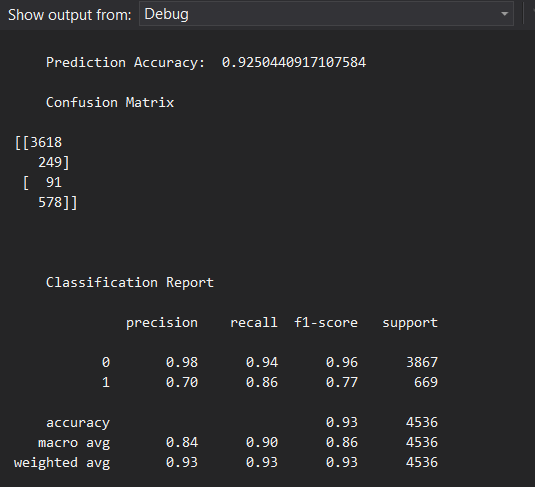
**FP: 89**

**FN: 273**

When we compare the FN value against a full dataset we can see that we have taken steps in our code to reduce the number of False Negatives.

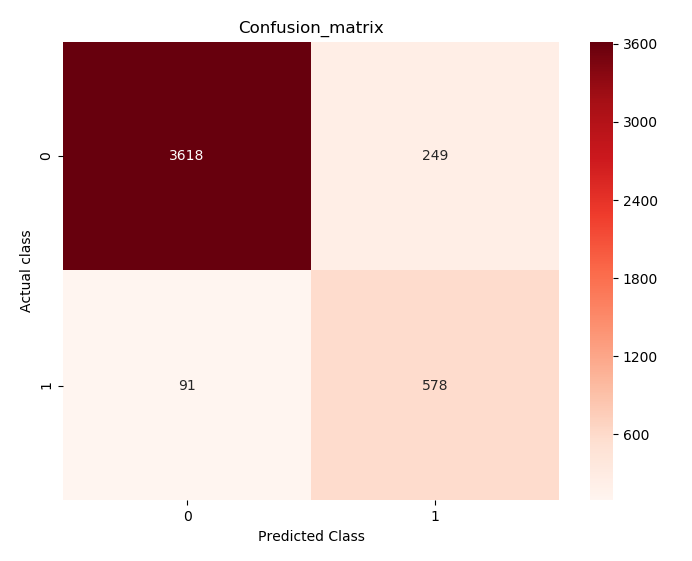
### Comparison Against Full Dataset

An interesting comparison can be made with the model performance when the full data set is used:



Model accuracy is 92.5% but the false negative performance is slightly worse.

The graph of the Confusion Matrix for this run looks like this:



**Confusion matrix:**

**[[3618 249]**

**[ 91 578]]**

**TP: 578**

**TN: 3618**

**FP: 249**

**FN: 91**

Our revisions to the Feature set, by selecting of sub list of the most significant features, have allowed us to follow through on our objective of reducing False Negatives.

In addition, we found that lower values of ‘n\_estimators’ reduced the false positive values, but that was not our objective.

# Guidelines Generated by Prediction Model

## Focus on a smaller Feature set of data

Our analysis of the Spruce dataset, through our Python application showed that we could achieve our objective of building a model to predict on which plot to plant Spruce trees, and do so with an approach that reduces false negative results.

The original dataset contained 44 input variables, mainly information of the soil types in Canadian forests.

A guideline we would offer the Forestry Services is that it could focus on a smaller sub set of soil types and the resultant dataset would still be able to generate a meaningful prediction model. Rather than run tests on the soil type to determine which of the 38 types to record for each plot, the forest department could limit their checks to determining if the new plot was one of just of the following:

* Soil Type 20
* Soil Type 21
* Soil Type 9
* Soil Type 27
* Soil Type 36
* Soil Type 3
* Soil Type 37
* Soil Type 4
* Soil Type 30

Any other soil type could just be marked as ‘other’. Presumably this would speed up the process of assessing each individual plot for future data gathering.

Working with a smaller dataset should also improve processing time if the dataset grew to be much larger.

## Random Forest Parameter

In setting the ‘n\_estimator’ parameter to the Random Forest function we focused on not overfitting the data and reducing False Negatives, which minimises the chance of missing a plot that is good for Spruce.

Our analysis showed that we could reduce the value of ‘n\_estimator’ if we wanted to focus on False Positives, and train the model to avoid planting Spruce in an unsuitable plot. Reducing False Positives was not our strategy in this instance but it is useful information to have on hand to assist the forestry prediction.

## Check the ‘Missing Data’

A number of the rows in the Spruce dataset have zero values. Other advice for the Canadian Forestry Department would be have a forest ranger, or other expect, apply some domain knowledge and determine if those zero values are legitimate or are genuine ‘missing data’. This could inform future data clean up processes and help with new model accuracy.

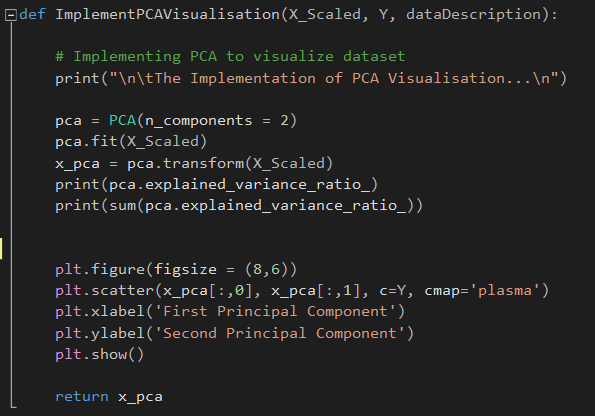
# Principal Component Analysis (PCA)

## Code and Data Preparation

PCA is a technique for dimensionality reduction that is used to visualise datasets with more than 2 or 3 dimensions, which is the case with the Spruce dataset.

The PCA visualisation for the Spruce dataset will attempt to represent on two axes as much variance as possible contained in all the captured features.

In our code set, the PCA function is invoked using the reduced Feature set. This feature set has already been reduced to the 15 features as described in Section 6.1 of this document.



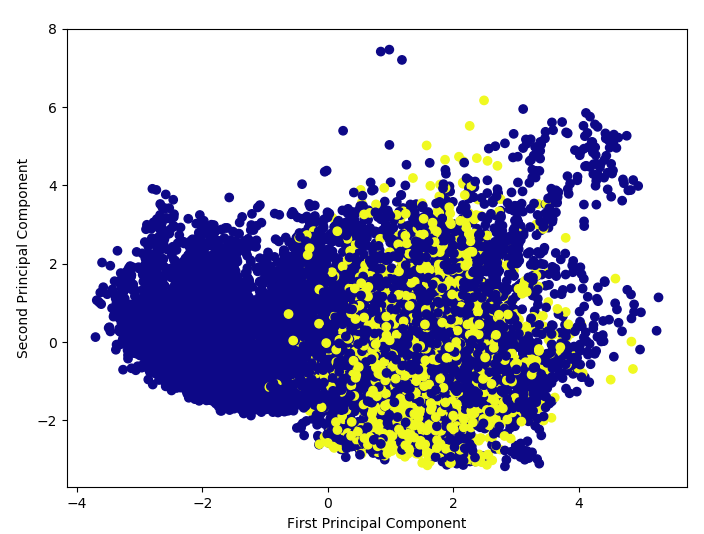
The data has also been normalised so that features with different ranges are adjusted to allow them be compared much more effectively by the Machine Learning algorithms, and by the PCA visualisation. The result is that the dataset based to the PCA Visualisation routine contains feature values that are all mostly between -1 and 1 in magnitude.

This ‘scaled’ data set is the observations of the 44 input variables for the 15,120 forest plots in the ***Spruce.csv*** file, represented by the ‘X\_Scaled’ variable in the code snippet above.

The ‘Y’ variable contains the associated target variable for ‘Spruce/Other’, converted to a numerical value.

## PCA Implementation

Visualisation of the dataset, as output by our Python code:



How much variance is explained by the first two principal components? Is the resultant plot a good representation of data?

A sizeable amount of the Spruce data is represented by the first two principal components, as visualised in the above diagram.

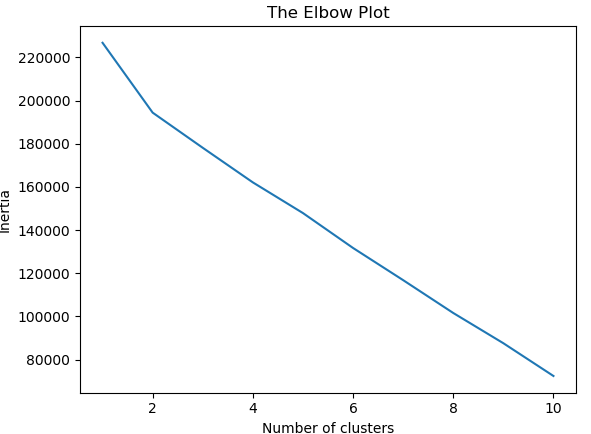
This analysis does make it easy for us to visualise the Spruce data, which would not be feasible if we tried to assess any more than 2 or 3 dimension.

The PCA graph is still a little cluttered and some of the data is obscured but it gives us a sense of the ‘shape’ of the data and will then feed into K-Mean analysis.

# K-Means Clustering

## Elbow Plot Creation

The image below is our output of the Elbow Plot creation from our code analysing the Spruce tree data:



The ‘Inertia’ value on the axis represents the within cluster sun of squared errors (WCSS) for a set of K values. We invoke the KMeans function in one of our Python libraries to generate the above graph.

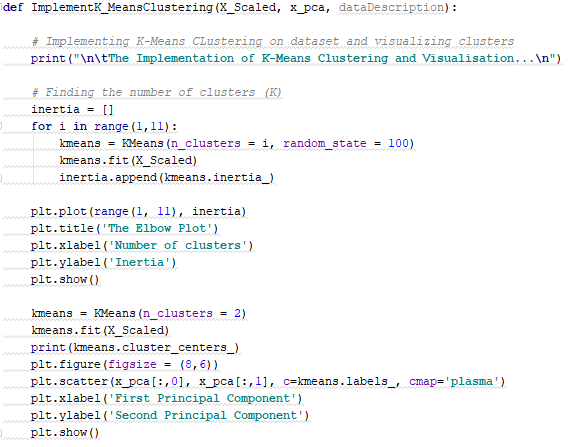
In the above elbow plot we can see that inertia drops as the number of clusters increases.

The drop in the value of inertia is less significant after number of clusters = 2. A cluster value of 2 clearly represents the ‘bend’ in the elbow.

## K-Means Implementation

We know the make-up of the Spruce dataset that there can only really be two meaningful clusters – to plant a Spruce tree **or** to plant another type of tree.

The K-Means code, including the Elbow Plot, as implemented by our group, is as follows:



The elbow plot confirms our assumption that ‘2’ is the optimum cluster value. This would have been understood from the nature of the dataset, even without the validation of the elbow plot.

The PCA visualisation matrix is passed to the K-Means function and allows the following cluster graph to be created. The PCA matrix allows a meaningful representation of the clustering of Spruce vs Other Tree types.

Our K-Means visualisation from the code is as follows:

