Abstract

This report represents phase 1 of a two phase project to demonstrate an end-to-end Machine Learning project. The end result of the two phase project is a set of models based on various machine learning classification algorithms to predict forest cover type based on cartographic data collected from four wilderness areas in Roosevelt National Forest, Colorado. Phase 1 of project covers data loading, initial examination, wrangling, detailed data exploration using visualization. In phase 2, the focus will be on training different models on the datasets prepared in phase 1 and comparing their performances on different evaluation metrics. In phase 1, we will derive multiple datasets from the original data so that these datasets can be directly fed to classification algorithms in phase 2. We will also explore underlaying relationships among the features to find out information that can be used to tune model parameters in the next phase.

Introduction

Objective

Objective of the project is to predict forest cover type based on the cartographic features provided. To fulfil the objective, through the course of this project we will develop machine learning prediction models to predict what kind of trees cover a particular observation area in a forest. Such models, if mature enough, can be used to predict the cover type of unexplored remote forest areas, based on which a comparatively more precise estimation of a state's natural resources can be made. These estimations in turn can be used for resource planning as well as calculating biomass and carbon footprint. We will divide the entire project life cycle in two phases. In phase 1, we will start off by loading data and making intial examinations, followed by cleaning the data and later on exploring the data using univariate and multivariate visualizations. We will wrangle and partition the data to render multiple offspring train-test dataset pairs from original data, which will be tailored to be used with a particular set of classification algorithms in phase 2.

Report is compiled by utilizing awesome functionalities provided by 'Jupyter Notebook' 1 . Report contains textual content as well as 'Python 3.6' code to carry out data pre-processing and exploratory activities.

Data Set

The dataset is hosted on "UCI Machine Learning Library"**2** and is put together by "Jock A. Blackard"**3**, "Dr. Denis J. Dean"**4** & "Dr. Charles W. Anderson"**5**. Original database is owned by "Remote Sensing and GIS Program, Colorado State University"**6**. Cover type data (target feature) is collected and provided by "US Forest Service**7** Inventory Information". Cartographic data (descriptive features) is accumlated from US Geological Survey (USGS)**8**.

Dataset contains 581012 observations, each of which, refer to 30 * 30 meter area in one of the four wilderness area upon which the study was conducted. Data contains 12 distinct measurements, and 54 descriptive features of data consisting 10 quantitative variables, 4 binary wilderness areas and 40 binary soil type variables. 44 out of 54 the original features are result of "one hot encoding" and were originally contained in two categorical features, which can be assumed to be 'wilderness area' & 'soil type' each having 4 & 40 levels respectively.

Target Feature

The response feature is "cover_type" which is a categorical variable with 7 distinct levels. Each level represents a tree species which is dominant in the 30 * 30 meter area of that particular wilderness. Values of target feature are integer encoded. Following are the categories with repective names.

- 1 -> Spruce/Fir
- 2 -> Lodgepole Pine
- 3 -> Ponderosa Pine
- 4 -> Cottonwood/Willow
- 5 -> Aspen
- 6 -> Douglas-fir
- 7 -> Krummholz

Descriptive Features

The dataset contains set of nominal and continuous features that describe cartographic data about the 30 * 30 meter observation area.

- **`Elevation`** : *Continuous - meters* - Elevation of observation area
- **`Aspect`** : *Continuous - azimuth* - Aspect in azimuth of observation area
- **`Slope`** : *Continuous - degrees* - Slope of the observation area in degrees
- **`Horizontal_Distance_To_Hydrology`** : *Continuous - meters* - Horizontal Distance from nearest surface water source
- **`Vertical_Distance_To_Hydrology`** : *Continuous - meters* - Vertical Distance from nearest surface water source
- **`Horizontal_Distance_To_Roadways`** : *Continuous - meters* - Horizontal Distance from nearest built up roadway
- **`Hillshade_9am`** : *0 to 255 index* - Hillshade index on observation area at 9am, summer solstice
- **`Hillshade_Noon`** : *0 to 255 index* - Hillshade index on observation area at noon, summer soltice
- **`Hillshade_3pm`** : *0 to 255 index* - Hillshade index on observation area at 3pm, summer solstice
- **`Horizontal_Distance_To_Fire_Points`** : *Continuous - meters* - Horizontal Distance of observation area from nearest wildfire ignition points

- **`Wilderness_Area`** : *4 Binary features* - Wilderness area designation - 0 (absence) or 1 (presence) 1 -> Rawah Wilderness Area 2 -> Neota Wilderness Area 3 -> Comanche Peak Wilderness Area 4 -> Cache la Poudre Wilderness Area
- **`Soil_Type`** : *40 Binary features* - Soil Type designation - 0 (absence) or 1 (presence) Soil_Type 1 to 40 : based on the USFS Ecological Landtype Units (ELUs) for study area
  - USFS Codes for Soil Types - '2702', '2703', '2704', '2705', '2706', '2717', '3501', '3502', '4201', '4703', '4704', '4744', '4758', '5101', '5151', '6101', '6102', '6731', '7101', '7102', '7103', '7201', '7202', '7700', '7701', '7702', '7709', '7710', '7745', '7746', '7755', '7756','7757', '7790', '8703', '8707', '8708', '8771', '8772', '8776'

Data Pre-processing

Preliminaries

We start off by loading all the relevant packages for this phase of project. we will load Matplotlib*9*, Pandas*10*, Numpy*11*, Seaborn*12*, Sklearn*13*, Scipy*14*, Imbalance-learn*15* packages for our project.

We get the data in form of a zip file from https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/covtype.data.gz (https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/covtype.data.gz). After loading the data into a pandas dataframe named 'data_original', We rename the features with appropriate title. Notice that each level of 'wilderness area' & 'soil type' has a dedicated feature to it. we will name these features with names of their respective levels e.g. 'rawah', 'neota' etc for wilderness area and '2073', '2074' etc for soil type. These numeric soil type codes are designated codes by USFC.

lastly, We check if data is loaded correctly by inspecting the first few instances of data and checking the dimension of the dataframe.

In [1]:
```
# To display output of all the statements and not only last statement
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# Packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from IPython.display import display, HTML
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

In [2]:
```
filePathShort = "new 1.DATA"

filePathFull = "covtype.data"

features = ['elevation', 'aspect', 'slope', 'horizontal_distance_to_hydrology'
,
            'vertical_distance_to_hydrology', 'horizontal_distance_to_roadway
s',
            'hillshade_9am', 'hillshade_noon', 'hillshade_3pm', 'horizontal_di
stance_to_fire_points',
            'rawah','neota','comanche_peak','cache_la_poudre',
            '2702', '2703', '2704', '2705', '2706', '2717', '3501', '3502', '4
201', '4703',
            '4704', '4744', '4758', '5101', '5151', '6101', '6102', '6731', '7
101', '7102',
            '7103', '7201', '7202', '7700', '7701', '7702', '7709', '7710', '7
745', '7746',
            '7755', '7756','7757', '7790', '8703', '8707', '8708', '8771', '87
72', '8776',
            'cover_type']

continuous_feature_names = ['elevation', 'aspect', 'slope', 'horizontal_distan
ce_to_hydrology',
                            'vertical_distance_to_hydrology', 'horizontal_dist
ance_to_roadways',
                            'hillshade_9am', 'hillshade_noon', 'hillshade_3pm'
,
                            'horizontal_distance_to_fire_points']

binary_feature_names = ['rawah','neota','comanche_peak','cache_la_poudre',
            '2702', '2703', '2704', '2705', '2706', '2717', '3501', '3502', '4
201', '4703',
            '4704', '4744', '4758', '5101', '5151', '6101', '6102', '6731', '7
101', '7102',
            '7103', '7201', '7202', '7700', '7701', '7702', '7709', '7710', '7
745', '7746',
            '7755', '7756','7757', '7790', '8703', '8707', '8708', '8771', '87
72', '8776']


# print(features)
Data = pd.read_csv(filePathFull, names = features)

print("Confirming proper data loading")
print("Loaded Data has " + str(Data.shape[0]) +" instances & " + str(Data.shap
e[1]) \
      + " features")
Data.head()
```

```
Confirming proper data loading
Loaded Data has 581012 instances & 55 features
```

Out[2]:

	elevation	aspect	slope	horizontal_distance_to_hydrology	vertical_distance_to_hydrology	hor
0	2596	51	3	258	0	
1	2590	56	2	212	-6	
2	2804	139	9	268	65	
3	2785	155	18	242	118	
4	2595	45	2	153	-1	

5 rows × 55 columns

Data Cleaning and Transformation

Data type check and casting

From the table below, it is evident that all the data is of integer type. We will be adding 2 categorical features for 'soil_type' and 'wilderness_area' in further sections.

In [3]:
```
typesBefore = Data.dtypes
indices = list(typesBefore.index)
Data[Data.select_dtypes(['object']).columns] = \
Data.select_dtypes(['object']).apply(lambda x: x.astype('category'))

k = 0
print(f"Data Types are: ")
typesAfter = Data.dtypes
print("Feature Names\t\t\t\t\t  Original type")
for i,j in zip(typesBefore, typesAfter):
    print ('%-55s%-20s' % (indices[k],str(i)))
    #print( indices[k] +""+ str(i) + " >>> " + str(j))
    k += 1
```

```
Data Types are:
Feature Names                                      Original type
elevation                                               int64
aspect                                                  int64
slope                                                   int64
horizontal_distance_to_hydrology                        int64
vertical_distance_to_hydrology                          int64
horizontal_distance_to_roadways                         int64
hillshade_9am                                           int64
hillshade_noon                                          int64
hillshade_3pm                                           int64
horizontal_distance_to_fire_points                      int64
rawah                                                   int64
neota                                                   int64
comanche_peak                                           int64
cache_la_poudre                                         int64
2702                                                    int64
2703                                                    int64
2704                                                    int64
2705                                                    int64
2706                                                    int64
2717                                                    int64
3501                                                    int64
3502                                                    int64
4201                                                    int64
4703                                                    int64
4704                                                    int64
4744                                                    int64
4758                                                    int64
5101                                                    int64
5151                                                    int64
6101                                                    int64
6102                                                    int64
6731                                                    int64
7101                                                    int64
7102                                                    int64
7103                                                    int64
7201                                                    int64
7202                                                    int64
7700                                                    int64
7701                                                    int64
7702                                                    int64
7709                                                    int64
7710                                                    int64
7745                                                    int64
7746                                                    int64
7755                                                    int64
7756                                                    int64
7757                                                    int64
7790                                                    int64
8703                                                    int64
8707                                                    int64
8708                                                    int64
8771                                                    int64
8772                                                    int64
8776                                                    int64
cover_type                                              int64
```

Null values check

A quick check reveals that there are no missing values in dataset. However, we will make further sanity checks to see if missing values are encoded with some special values and if so, we will make the necessary imputations.

In [4]:
```
print("Missing value check")

print("Number of null values in entire dataset = " + str(Data.isnull().sum().sum()))
```

```
Missing value check
Number of null values in entire dataset = 0
```

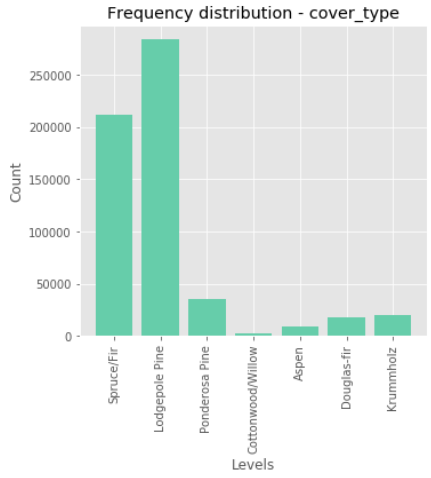
Target feature's distribution over the levels

The dataset has comparatively large number of instances (581012) based on the 7 levels of target feature. In such cases, it is always interesting to see how the values of target feature are distributed. Our target feature being a categorical one, we will use a barplot to checkout frequency distribution.

We have written a small function to help plot a barplot for categorical feature and generate a frequency table to better read the data from the plot.

In [5]:
```
def inspect_freq_distribution(feature, labels):
    plt.style.use('ggplot')
    freq_dist = (feature.value_counts() ).sort_index()
    df = pd.DataFrame((feature.value_counts() ).sort_index())
    bars = labels
    fig = plt.figure(figsize=(12,5))

    ax1 = fig.add_subplot(121)
    y_pos = np.arange(len(bars))
    _ = plt.bar(freq_dist.index, freq_dist , color = 'mediumaquamarine');
    _ = plt.xticks(freq_dist.index, bars, rotation = 90);
    _ = plt.ylabel("Count");
    _ = plt.xlabel("Levels");
    _ = plt.title("Frequency distribution - " + str(feature.name));

    ax2 = fig.add_subplot(122)
    font_size=14
    bbox=[1,0, 1,1]
    ax2.axis('off')
    mpl_table = ax2.table(cellText = df.values, rowLabels = labels, bbox=bbox,
colLabels=df.columns)
    mpl_table.auto_set_font_size(False)
    mpl_table.set_fontsize(font_size)

    plt.show();
```

In [6]:
```
inspect_freq_distribution(Data['cover_type'],['Spruce/Fir', 'Lodgepole Pine',
'Ponderosa Pine', 'Cottonwood/Willow', 'Aspen', 'Douglas-fir', 'Krummholz'])
```



	cover_type
Spruce/Fir	211840
Lodgepole Pine	283301
Ponderosa Pine	35754
Cottonwood/Willow	2747
Aspen	9493
Douglas-fir	17367
Krummholz	20510

Resampling - Under sampling

From the bar plot above and the frequency table, it is observable that the target feature distribution is highly uneven. Hence the dataset is **'imbalanced'**. Imbalanced datasets have significantly higher instance count for some target levels than rest of the target levels. In our dataset, 'Cottonwood/Willow' has only 2747 instances, while 'Spruce/Fir' and 'Lodgepole Pine' levels have over 200,000 instances.

This kind of radically uneven distribution can adversly affect the performance of most of the machine learning algorithms including KNN, Naive Bayes etc.

We can deal with imbalanced datasets by resampling the dataset from the imbalanced dataset. There are two commonly used techiniques by which we can resample the dataset.

1. Under-sampling
    - It balances the dataset by reducing size of levels which are dominant in the distribution.
    - We keep a certain number of instances of minority levels, and then randomly select same number of instances from the other more dominant target levels.
    - This method is useful when we have sufficient number of observations in the dataset to make a workable resampled dataset.
2. Over-sampling
    - It balances the data by increasing the size of minority levels.
    - Here, new samples of minority level are added using various techinques like repeatation, SMOTE (Synthetic Minority Oversampling Technique) or bootstraping
    - This method is suitable when the number of instances in the dataset are insufficient.

Comparing both of these techniques, we find that undersampling is more suitable to our dataset, as our dataset offers a sufficient number instances of minority level.(Cottonwood/Willow - 2747).

However, While applying undersampling it is important to remember that we don't use all the 2747 instances of minority level. Doing so will result in a testing set, where no unforseen query instances of this class will exist. This will certainly not be good for our models' performance.

In [7]:
```
X = Data[['elevation', 'aspect', 'slope', 'horizontal_distance_to_hydrology',
            'vertical_distance_to_hydrology', 'horizontal_distance_to_roadway
s',
            'hillshade_9am', 'hillshade_noon', 'hillshade_3pm', 'horizontal_di
stance_to_fire_points',
            'rawah','neota','comanche_peak','cache_la_poudre',
            '2702', '2703', '2704', '2705', '2706', '2717', '3501', '3502', '4
201', '4703',
            '4704', '4744', '4758', '5101', '5151', '6101', '6102', '6731', '7
101', '7102',
            '7103', '7201', '7202', '7700', '7701', '7702', '7709', '7710', '7
745', '7746',
            '7755', '7756','7757', '7790', '8703', '8707', '8708', '8771', '87
72', '8776']]

Y = Data[['cover_type']]

from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=0)
rus.fit(X, Y)
X_resampled, y_resampled = rus.fit_resample(X, Y)


balanced_data = pd.DataFrame(X_resampled)
balanced_data['cover_type'] = y_resampled
balanced_data.columns = features
```

Out[7]:
```
RandomUnderSampler(random_state=0, ratio=None, replacement=False,
          return_indices=False, sampling_strategy='auto')
```

We will examine the results of undersampling in next sections.

Brief on categorical feature encoding

There are several ML Algorithms, which can directly deal with categorical data, such as decision trees and random forests. However, several other algorithms like K Nearest Neighbors can not understand categorical data directly, and hence needs the data to be encoded into a numerical format. There are several methods to perform the encoding, some of them being 'Integer encoding', 'one-hot encoding' and 'binary encoding' etc.

Integer encoding assigns a random or ordered integer value to each level of a categorical feature and is most simplest encoding technique and is easy to reverse it to interprete the results. However, it induces a false sense of order or comparability into the data, for example, if we encode Argentina -> 1 & Spain -> 2 algorithm may use 1 & 2 as absolute number and draw conclusions based on the fact that 1 < 2. Which can affect the model accuracy adversly.

One-hot encoding is useful when we have categorical features comparatively small number of levels than instances in the dataset. It creates sparse matrix, with all the levels added as new features, and marking 1 for presence of the that level for that observation & 0 otherwise. This technique is most commonly used encoding technique and is easy to reverse.

Binary encoding is useful when the number of levels in feature is very high, as it can represent 2^n levels with n features. for example, if there are 100 levels in a categorical feature, one-hot encoding the feature will result in a set 100 new features, while binary encoding can represent same information in 7 features (2^7 = 128 distinct values). However, it takes slightly more work to reverse it.

One-hot encoding in the data

While this encoding makes the data more compaitable to classification algorithms we will be using in phase-II, understanding and visualizing the encoded data is rather complex. To make the further process of this report easier, we will reverse the encoding and append the original features derived from this process back to the data frame.

We have 54 features in total excluding target feature. It is also evident that 44 of these features are binary features, meaning that they have only two possible levels, 1 and 0, where 1 represents presence and 0 represents abscence.

Out of these 44 binary features, first 4. namely 'rawah','neota','comanche_peak' and 'cache_la_poudre' represent which to which wilderness the observation cell belongs. It is inferable that these 4 features are result of a 'one-hot' encoding on a original categorical feature 'Wilderness Area'.

Simillarly, rest of the 40 binary features '2702', '2703' etc. are result of applying 'one-hot' encoding to original feature 'Soil Type'.

Notice the simple function we used to reverse the one-hot encoding. Also notice the updated dimensions of the new dataset with two additional features 'soil_type' & 'wilderness_area' appended to the end of dataframe.

In [8]:
```
def reverse_one_hot_encoding(row, x):
    for c in balanced_data[x].columns:
        if row[c]==1:
            return c

# Reversing one-hot encoding and appending the newly generated features back t
o original data
f1 = [['2702', '2703', '2704', '2705', '2706', '2717', '3501', '3502', '4201',
'4703',
            '4704', '4744', '4758', '5101', '5151', '6101', '6102', '6731', '7
101', '7102',
            '7103', '7201', '7202', '7700', '7701', '7702', '7709', '7710', '7
745', '7746',
            '7755', '7756','7757', '7790', '8703', '8707', '8708', '8771', '87
72', '8776']]
f2 = [['rawah','neota','comanche_peak','cache_la_poudre']]

balanced_data['soil_type'] = balanced_data.apply(reverse_one_hot_encoding, arg
s=(f1), axis = 1)
balanced_data['wilderness_area'] = balanced_data.apply(reverse_one_hot_encodin
g, args=(f2), axis = 1)
balanced_data.head(3)
```

Out[8]:

	elevation	aspect	slope	horizontal_distance_to_hydrology	vertical_distance_to_hydrology	hor
0	3067	5	15	108	16	
1	3152	32	10	660	135	
2	2977	345	5	120	6	

3 rows × 57 columns

In [9]:
```
balanced_data.shape
```

Out[9]:
```
(19229, 57)
```

Summary Statistics & sanity checks

We will check the data for abnormal values. We will do so by checking the summary statistics and identify if there are any impossible value in the features. Consequently, we can say that there are no encoded null values as well.

From both tables below, it is evident that there are no abnormal values, some outliers may be present, but they are not errornous.

In [10]:
```
display(HTML('<b>Table 1: Summary of continuous features</b>'))
display(balanced_data[continuous_feature_names].describe())

display(HTML('<b>Table 2: Summary of categorical binary features</b>'))
display(balanced_data[binary_feature_names + ['soil_type','wilderness_area','cover_type']].apply(lambda x: x.astype('category')).describe())
```

Table 1: Summary of continuous features

	elevation	aspect	slope	horizontal_distance_to_hydrology	vertical_dista
count	19229.000000	19229.000000	19229.000000	19229.000000	
mean	2749.982734	155.864111	16.511259	227.743772	
std	417.554014	109.797225	8.488489	208.740273	
min	1871.000000	0.000000	0.000000	0.000000	
25%	2380.000000	65.000000	10.000000	67.000000	
50%	2753.000000	125.000000	15.000000	180.000000	
75%	3108.000000	257.000000	22.000000	330.000000	
max	3850.000000	360.000000	51.000000	1321.000000	

Table 2: Summary of categorical binary features

	rawah	neota	comanche_peak	cache_la_poudre	2702	2703	2704	2705	2706	
count	19229	19229	19229	19229	19229	19229	19229	19229	19229	1
unique	2	2	2	2	2	2	2	2	2	
top	0	0	0	0	0	0	0	0	0	
freq	14732	18594	10951	13410	18786	18413	17986	18132	19036	1

4 rows × 47 columns

Data Exploration

We will perform Data Exploration using visualisations and will perform,

- Univariate visualizations to understand general distribution of Data in individual descriptive features.
- Multivariate visualizations to better understand relationships among features.

As we have reversed the one-hot encoding, it will be easier to visualize the features using the newly added feature.

Univariate visualizations

We will divide the Univariate visualizations portion in 2 parts,

- Exploration of categorical features
- Exploration of continuous features

To visualize categorical features, we will use bar plots, which is the general preference to explore data characteristics in nominal or small scale ordinal data. Barplots reflect the proportion of observation falling into particular categories.

To visualize continuous features, we will use histogram with overlayed density plots. These plots are preferred to examine general distribution of values in feature as well as skewness of distribution.

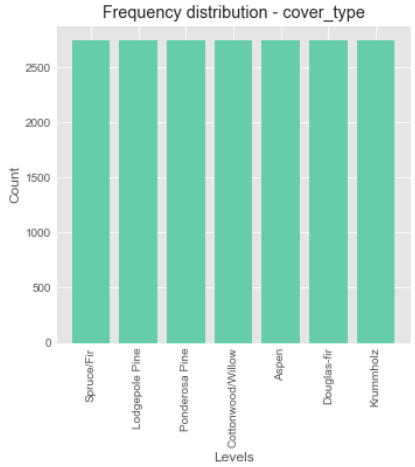
In [11]:
```
# General graph styling for consistent layout across graphs
import seaborn as sns

plt.style.use('seaborn-white')
plt.rcParams['axes.edgecolor']='#FFFFFF'
plt.rcParams['axes.linewidth']=0
plt.rcParams['xtick.color']='#333F4B'
plt.rcParams['ytick.color']='#333F4B'
plt.rcParams['figure.figsize'] = 4.5, 3
```

Exploration of categorical features

Cover Type

As a result of resampling (undersampling) the dataset over target feature cover type, we can see that it has a flat distribution. Frequency table shows that all target levels have equal instance count of 2747 instances, making our dataset balanced over target feature level distribution.

In [12]:
```
inspect_freq_distribution(balanced_data['cover_type'],['Spruce/Fir', 'Lodgepole Pine', 'Ponderosa Pine', 'Cottonwood/Willow', 'Aspen', 'Douglas-fir', 'Krummholz'])
```

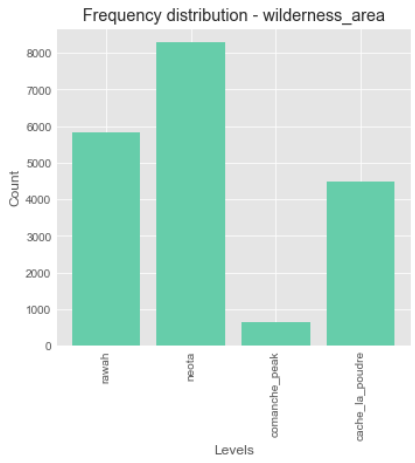


	cover_type
Spruce/Fir	2747
Lodgepole Pine	2747
Ponderosa Pine	2747
Cottonwood/Willow	2747
Aspen	2747
Douglas-fir	2747
Krummholz	2747

Wilderness Area

The bar plot below shows the frequency distribution of the 4 levels of feature 'wilderness_area'. As can be infered from the plot and frequency table, we have 8278 instances from 'neota' area, which occupies highest proportion of the dataset, followed by 'rawah' with 5819 instances and 'cache la poudre' area with 4497 instances. 'Comanche peak' has the least number of occurances out of all 4 levels.

In [13]:
```
inspect_freq_distribution(balanced_data['wilderness_area'],['rawah','neota','comanche_peak','cache_la_poudre'])
```



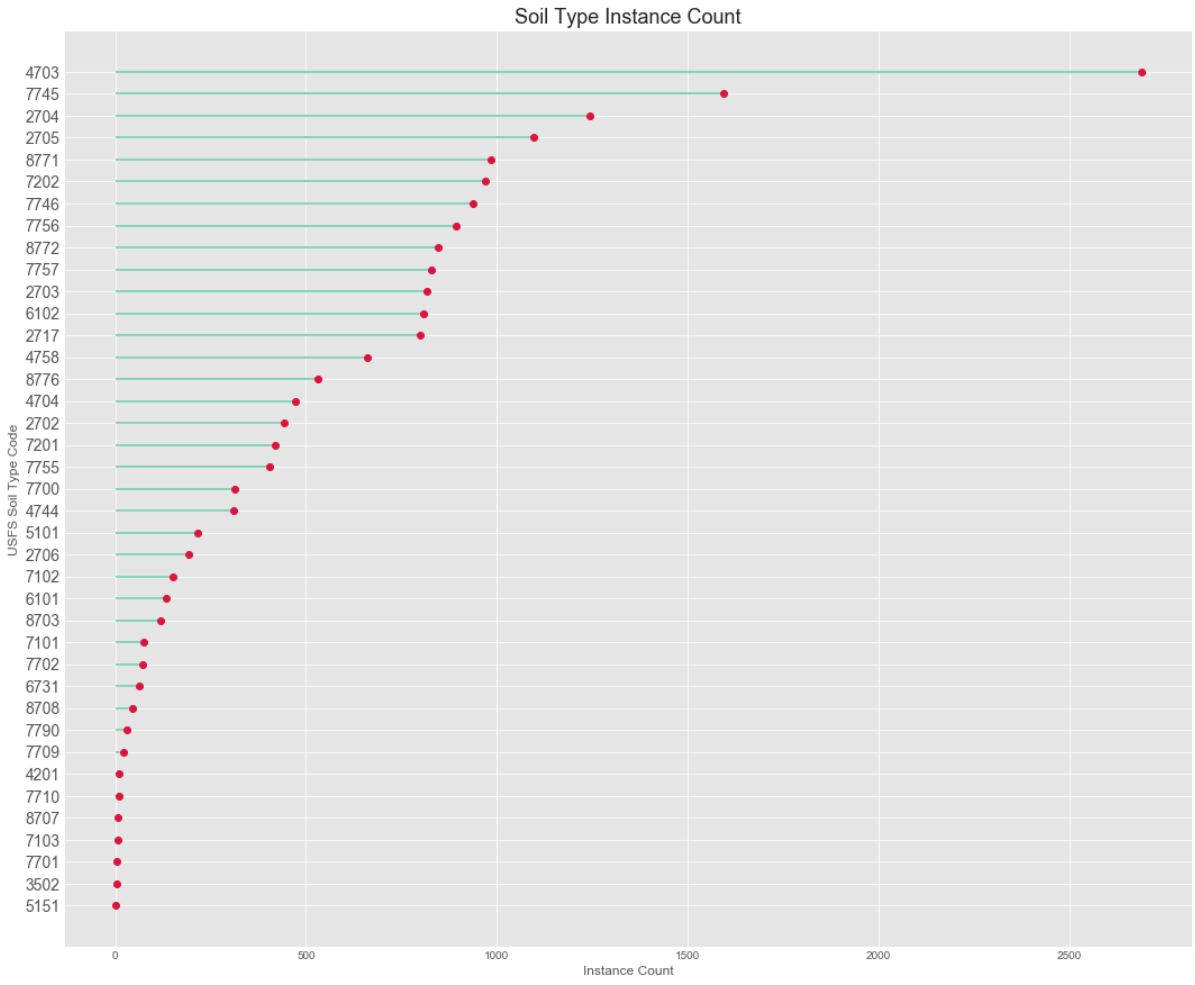
	wilderness_area
rawah	5819
neota	8278
comanche_peak	635
cache_la_poudre	4497

Soil Type Count

The horizontal lolipop plot below shows the frequency distribution of levels of feature 'soil_type'. The labels on Y axis are the USFS (United States Forest Services) soil type codes, latest version of these codes and their associated soil type can be found at - https://www.fs.fed.us/nrm/documents/fsveg/cse_user_guides/R8FG_appendix.pdf (https://www.fs.fed.us/nrm/documents/fsveg/cse_user_guides/R8FG_appendix.pdf).

From the graph we can see a non uniform distribution, with some levels like '4703', '7745' & '2704' occupying significantly more number of observations. Other levels at the bottom of the Y axis like '5151' & '3502' have very low frequency.

In [14]:
```
ordered_df = pd.DataFrame(balanced_data['soil_type'].value_counts().sort_values())
ordered_df.head(39)

my_range=range(1,len(ordered_df.index)+1)

plt.figure(figsize=(18,15))
plt.hlines(y=my_range, xmin=0, xmax=ordered_df['soil_type'], color='mediumaquamarine')
plt.plot(ordered_df['soil_type'], my_range, "o", color = 'crimson')
# Add titles and axis names
plt.yticks(my_range, ordered_df.index,  fontsize = 14)
plt.title("Soil Type Instance Count", fontdict={'fontsize': 18, 'fontweight': 'medium'})
plt.xlabel('Instance Count')
plt.ylabel('USFS Soil Type Code')
plt.show();
```



Exploration of continuous features

We have plotted side by side plots for continuous features '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'.

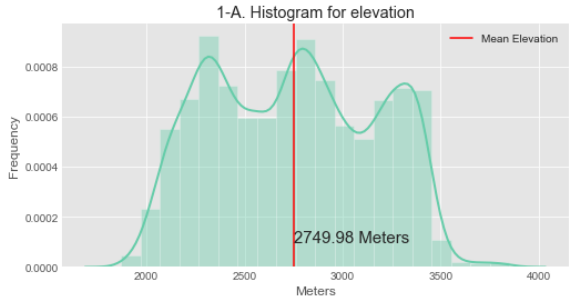
Most of the features have a highly skewed distribution. 'Elevation' has a multi-modal distribution, while aspect density plot forms a valley. 'Horizontal_distance_from roadways','Horizontal_distance_from_hydrology','Vertical_distance_from_hydrology' & 'Horizontal_distance_from_firepoints' have a high left-skew. The left skew in the density plot indicates that there are relatively few instances with great distances from hydrology, roadways and firepoints in general.

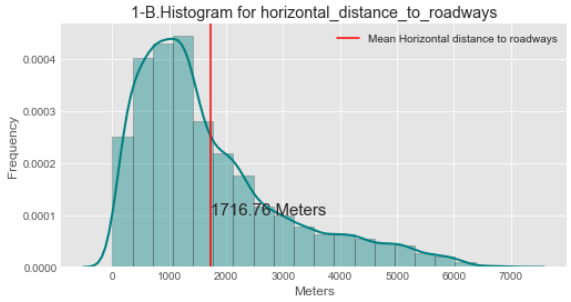
Features representing 'Hillshade' at different time of days are right-skewed, except 'hillshade_3pm', which is roughly normally distributed. It is interesting to note that hillshade index' mean value increases from 9am to noon and than dramatically decreases.

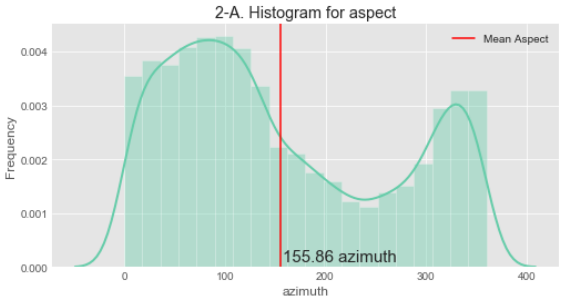
We will examine relationships among these feature in a more detailed manner in next section.

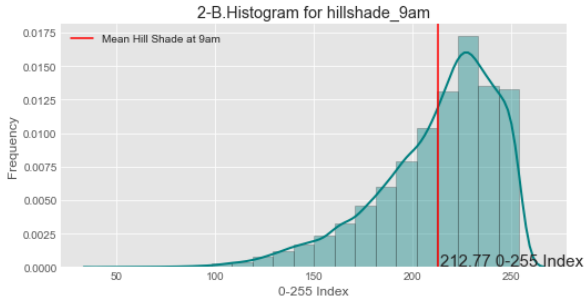
In [15]:
```
plot1 = ['elevation', 'aspect', 'slope', 'horizontal_distance_to_hydrology',
            'vertical_distance_to_hydrology']
plot2 = ['horizontal_distance_to_roadways', 'hillshade_9am', 'hillshade_noon',
         'hillshade_3pm', 'horizontal_distance_to_fire_points']

exLabelsA = ["Elevation", "Aspect", "Slope", \
             "Horizontal distance to hydrology","Vertical  distance to hydrolo
gy" ]
xLabelsA = ["Meters", "azimuth", "Degree", "Meters","Meters"]
exLabelsB = ["Horizontal distance to roadways", "Hill Shade at 9am", \
             "Hill Shade at noon", "Hill Shade at 3pm","Horizontal distance to
fire points", ]
xLabelsB = ["Meters", "0-255 Index", "0-255 Index", "0-255 Index","Meters"]

fig, axs = plt.subplots(nrows = 5, ncols=2, figsize=(18,25))

i=0
sns.despine(left = True)

for f1,f2 in zip(plot1,plot2):

    mean1 = round(balanced_data[f1].mean(), 2)
    mean2 = round(balanced_data[f2].mean(), 2)

    sns.distplot(balanced_data[f1], hist=True, kde=True, bins=int(20), color =
\
                 'mediumaquamarine', ax = axs[i][0], hist_kws={'edgecolor':'wh
ite'},\
                 kde_kws={'linewidth': 2})

    l1 = axs[i][0].axvline(x= mean1, label='Mean ' + exLabelsA[i], c='r')

    axs[i][0].text(x=  mean1 + 1.2, y = 0.0001, s=str(mean1)+" "+ xLabelsA[i],
\
                   fontsize = 15 )
    axs[i][0].set_title(str(i+1)+"-A. Histogram for " + f1 )
    axs[i][0].set_ylabel("Frequency")
    axs[i][0].set_xlabel(xLabelsA[i])
    axs[i][0].legend(loc = "best")

    sns.distplot(balanced_data[f2], hist=True, kde=True,  bins=int(20), color
= \
                 'teal', ax = axs[i][1], hist_kws={'edgecolor':'black'}, \
                 kde_kws={'linewidth': 2})

    l2 = axs[i][1].axvline(x= mean2, label='Mean ' + exLabelsB[i], c='r')
    axs[i][1].text(x= mean2 + 1.2, y = 0.0001, s=str(mean2)+" "+ xLabelsB[i],\
                   fontsize = 15)
    axs[i][1].set_title(str(i+1)+"-B.Histogram for " + f2)
    axs[i][1].set_ylabel("Frequency")
    axs[i][1].set_xlabel(xLabelsB[i])
    axs[i][1].legend(loc = "best")

    i+=1
plt.subplots_adjust(hspace = 0.3)
plt.show();
```

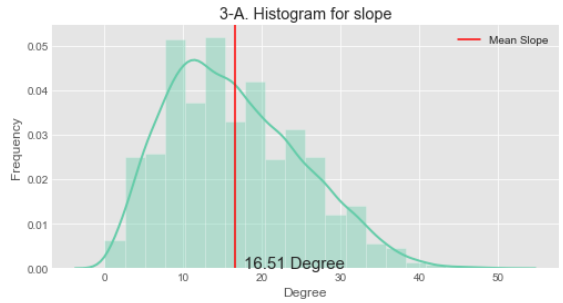
```
C:\Users\jigar\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureW
arning: Using a non-tuple sequence for multidimensional indexing is deprecate
d; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be in
terpreted as an array index, `arr[np.array(seq)]`, which will result either i
n an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

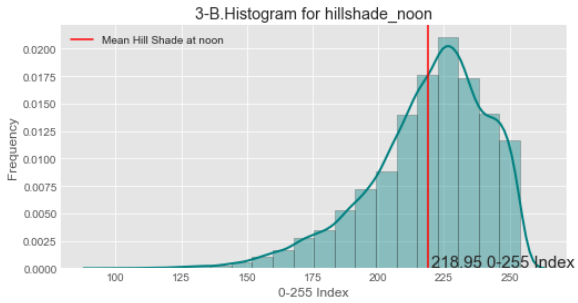


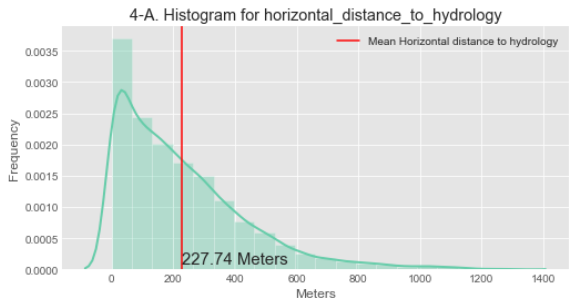


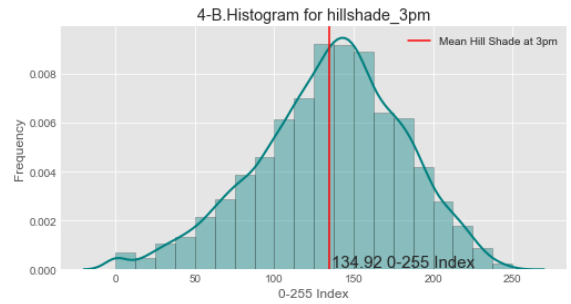


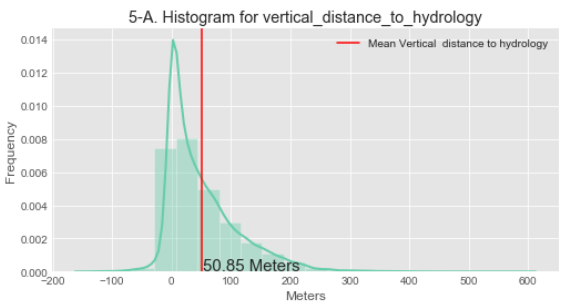


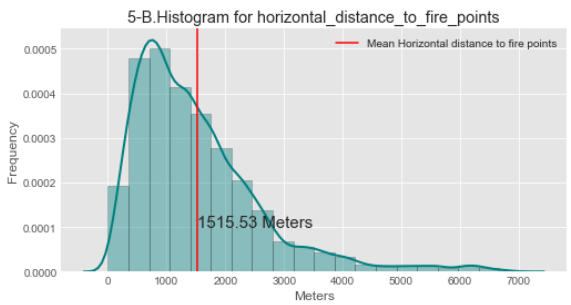












Multivariate Visualization

Categorized box plots

In this section, we will inspect underlaying relationships between various continuous variables with respect to their corresponding target feature level. We will use box plots grouped by cover_type. We can state some

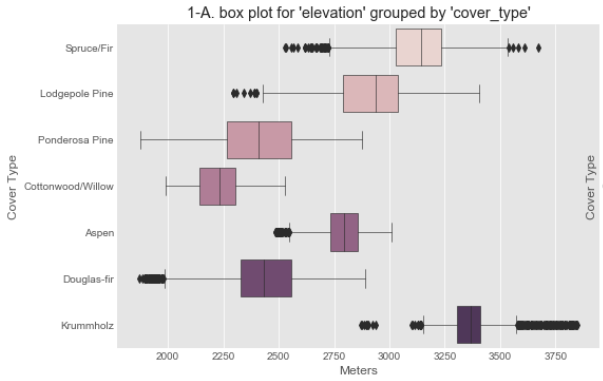
Elevation by Cover Type

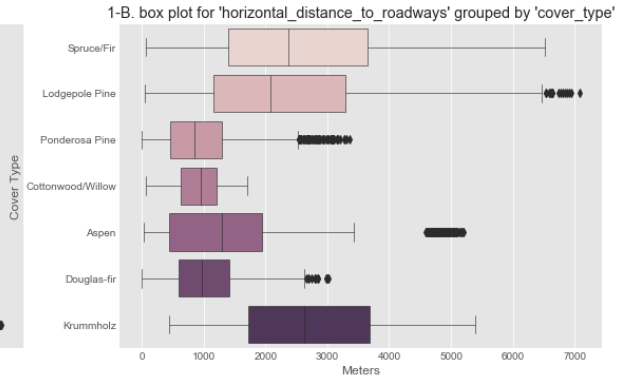
Following box plot shows some significant insights. It is rather understandable that some tree species are more suitable to high altitude and some are more commonly found at lower altitude. By plotting the elevation grouped by cover type, we can have a rough idea on which forest cover type designation is commonly found at a particular range of elevation.

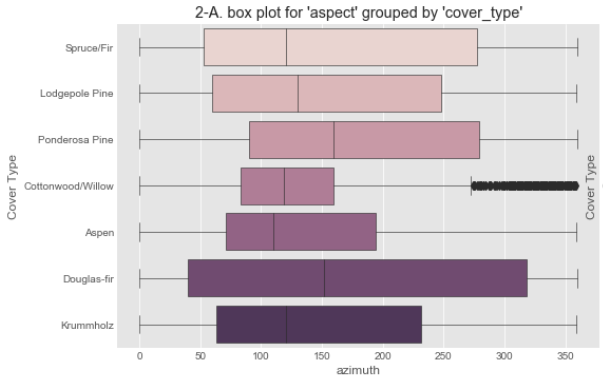
Considering the plots 1-A, We can observe following insights

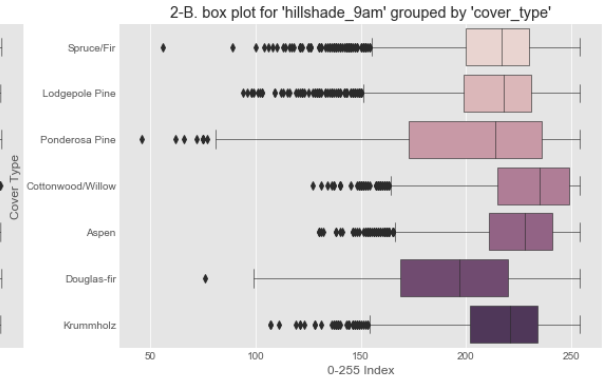
- 'Krummholz' forests have heavy presence at higher altitude(more than 3300 meters).
- 'Spruce-Fir' forests are also found at high altitudes which is slightly lower than 'Krummholz' forests.
- 'Cottonwillow' cover type is generally found at the bottom of valleys near surface water sources or in shallow waters around banks of lakes and rivers. Which confirms with it having lowest elevation, as water sources will be in the valley with low elevation.
- Elevation has a significant impact on cover type, as different levels of cover types have a rather significant difference in their elevation ranges.
- From 1-B, we can see that 'Krummholz' & 'Spruce-Fir' are relatively farther from roadways, while 'Aspen', 'Douglas-fir' & 'Ponderosa Pine' are comparatively near to roadways.
- From 4-A & 5-A, we can see 'Krummholz' is relatively far from hydrology (Water sources), which makes sense considering high elevation. This observation can be used to propose a hypothesis - 'as elevation increases, distance from the hydrology increases.' We will confirm the hypothesis in a later section.
- From 1-A & 5-B, it is observable that the areas at higher altitudes are farther away from the fire starting points. They are also farther away from roadways and hydrology. It will be worthwhile to check if these features are correlated or not.
- There are no meaningful insights which can be gathered from hillshade features and aspect feature. Wide and overlapping ranges among groups do point towards high shannon's entropy values however.

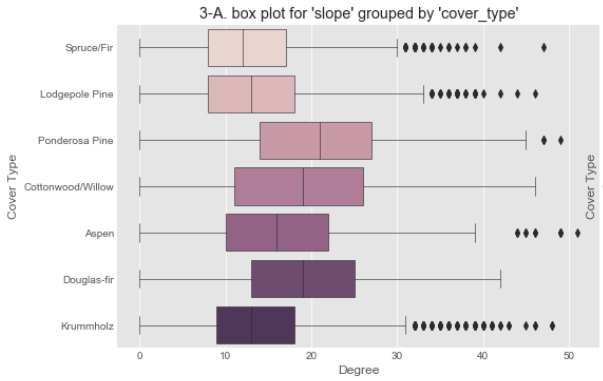
In [16]:
```
dic = {'cover_type': {1:'Spruce/Fir', 2:'Lodgepole Pine', 3:'Ponderosa Pine',
4:'Cottonwood/Willow', 5:'Aspen', 6:'Douglas-fir', 7:'Krummholz'}}
temp_data = balanced_data.replace(dic)


plot1 = ['elevation', 'aspect', 'slope', 'horizontal_distance_to_hydrology', \
            'vertical_distance_to_hydrology']
plot2 = ['horizontal_distance_to_roadways', 'hillshade_9am', 'hillshade_noon',
\
         'hillshade_3pm', 'horizontal_distance_to_fire_points']


exLabelsA = ["Elevation", "Aspect", "Slope", "Horizontal distance to hydrolog
y",\
             "Vertical  distance to hydrology", ]
xLabelsA = ["Meters", "azimuth", "Degree", "Meters","Meters"]
exLabelsB = ["Horizontal distance to roadways", "Hill Shade at 9am", \
             "Hill Shade at noon", "Hill Shade at 3pm",\
             "Horizontal distance to fire points", ]
xLabelsB = ["Meters", "0-255 Index", "0-255 Index", "0-255 Index","Meters"]

fig, axs = plt.subplots(nrows = 5, ncols=2, figsize=(18,35))

i=0
sns.despine(left = True)

for f1,f2 in zip(plot1,plot2):
    sns.boxplot(x=temp_data[f1],y = temp_data['cover_type'], orient="h", palet
te=sns.cubehelix_palette(8), ax = axs[i][0],linewidth=0.6)
    axs[i][0].set_title(str(i+1)+"-A. box plot for '" + str(f1) + "' grouped b
y 'cover_type'" )
    axs[i][0].set_ylabel("Cover Type")
    axs[i][0].set_xlabel(xLabelsA[i])
    sns.boxplot(x=temp_data[f2],y = temp_data['cover_type'], orient="h", palet
te=sns.cubehelix_palette(8),ax = axs[i][1], linewidth=0.6)
    axs[i][1].set_title(str(i+1)+"-B. box plot for '" + str(f2) + "' grouped b
y 'cover_type'")
    axs[i][1].set_ylabel("Cover Type")
    axs[i][1].set_xlabel(xLabelsB[i])
    i+=1
plt.subplots_adjust(hspace = 0.3)
plt.show();
```

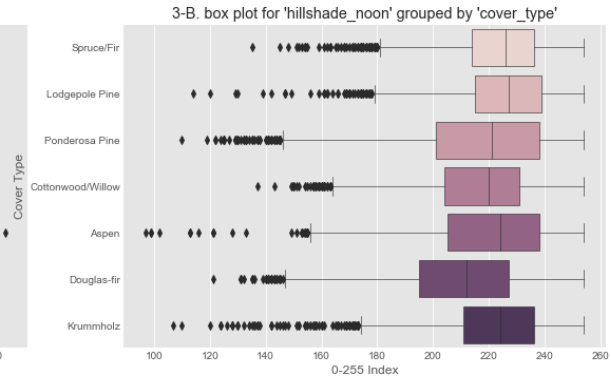

1-A. box plot for 'elevation' grouped by 'cover_type'
Spruce/Fir
Lodgepole Pine
Ponderosa Pine
Cottonwood/Willow
Aspen
Douglas-fir
Krummholz
Cover Type
2000
2250
2500
2750
3000
3250
3500
3750
Meters

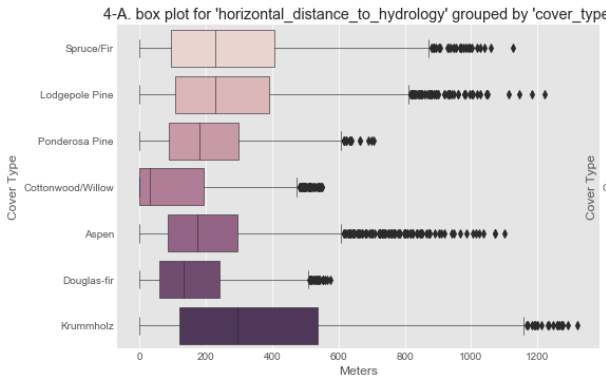

1-B. box plot for 'horizontal_distance_to_roadways' grouped by 'cover_type'
Spruce/Fir
Lodgepole Pine
Ponderosa Pine
Cottonwood/Willow
Aspen
Douglas-fir
Krummholz
Cover Type
0
1000
2000
3000
4000
5000
6000
7000
Meters

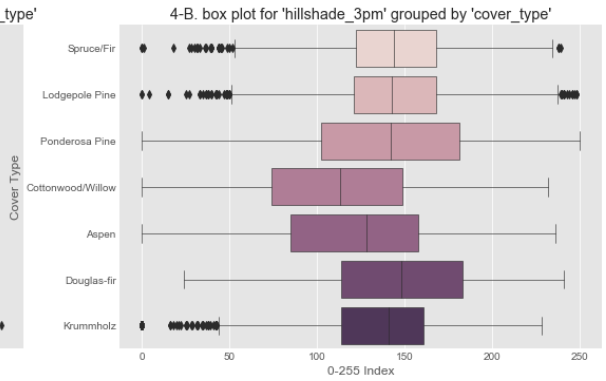

2-A. box plot for 'aspect' grouped by 'cover_type'
Spruce/Fir
Lodgepole Pine
Ponderosa Pine
Cottonwood/Willow
Aspen
Douglas-fir
Krummholz
Cover Type
0
50
100
150
200
250
300
350
azimuth

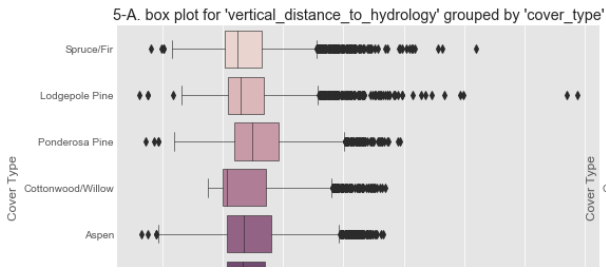

2-B. box plot for 'hillshade_9am' grouped by 'cover_type'
Spruce/Fir
Lodgepole Pine
Ponderosa Pine
Cottonwood/Willow
Aspen
Douglas-fir
Krummholz
Cover Type
50
100
150
200
250
0-255 Index

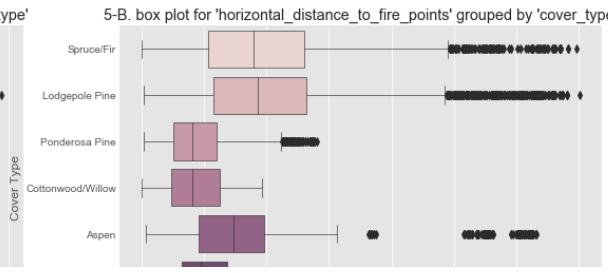

3-A. box plot for 'slope' grouped by 'cover_type'
Spruce/Fir
Lodgepole Pine
Ponderosa Pine
Cottonwood/Willow
Aspen
Douglas-fir
Krummholz
Cover Type
0
10
20
30
40
50
Degree

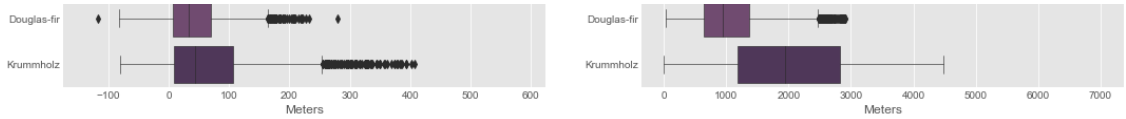

3-B. box plot for 'hillshade_noon' grouped by 'cover_type'
Spruce/Fir
Lodgepole Pine
Ponderosa Pine
Cottonwood/Willow
Aspen
Douglas-fir
Krummholz
Cover Type
100
120
140
160
180
200
220
240
260
0-255 Index


4-A. box plot for 'horizontal_distance_to_hydrology' grouped by 'cover_type'
Spruce/Fir
Lodgepole Pine
Ponderosa Pine
Cottonwood/Willow
Aspen
Douglas-fir
Krummholz
Cover Type
0
200
400
600
800
1000
1200
Meters


4-B. box plot for 'hillshade_3pm' grouped by 'cover_type'
Spruce/Fir
Lodgepole Pine
Ponderosa Pine
Cottonwood/Willow
Aspen
Douglas-fir
Krummholz
Cover Type
0
50
100
150
200
250
0-255 Index


5-A. box plot for 'vertical_distance_to_hydrology' grouped by 'cover_type'
Spruce/Fir
Lodgepole Pine
Ponderosa Pine
Cottonwood/Willow
Aspen
Cover Type


5-B. box plot for 'horizontal_distance_to_fire_points' grouped by 'cover_type'
Spruce/Fir
Lodgepole Pine
Ponderosa Pine
Cottonwood/Willow
Aspen
Cover Type

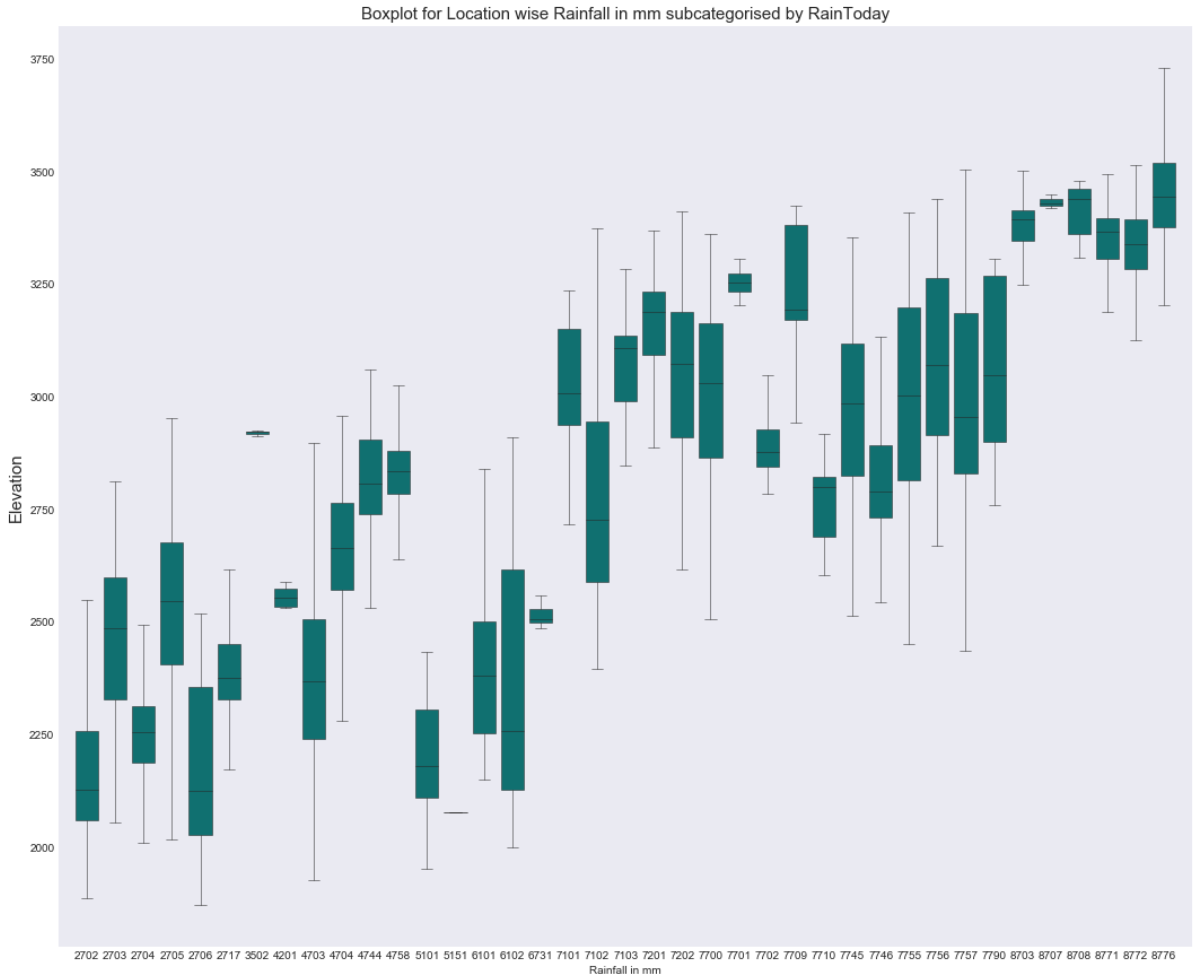


Boxplot for 'Soil Type'-wise Elevation

We have already seen that certain cover types like 'Spruce-fir' & 'Krummholz' are more likely to be found at higher altitudes. It will be interesting to see if same applies to soil types.

After examining the plot below, it does becomes clearer that certain soil types, the ones on right end side of plot, are more likely to be present at higher altitude. Soil Types like '8701','8702', '8771', '8772' & '8776' are only found at higher elevation. Can these soil types and the cover types 'Spruce-fir' & 'Krummholz' have a direct correlation? We will examine this kind of relationships among 'cover_type', 'soil_type' & 'elevation' in the next section.

In [17]:
```
#figure(num=None, figsize=(15, 20), dpi=80, facecolor='w', edgecolor='k')
plt.figure(figsize=(18,15))
plt.style.use("seaborn-dark")
sns.boxplot(x="soil_type", y="elevation", data=temp_data,  color = "teal", lin
ewidth = 0.5, showfliers = False)
plt.title("Boxplot for Location wise Rainfall in mm \
subcategorised by RainToday",fontsize=15 )
plt.ylabel("Elevation",fontsize=15)
plt.xlabel("Rainfall in mm",fontsize=10)
plt.xlim(-1,39)


plt.show();
```



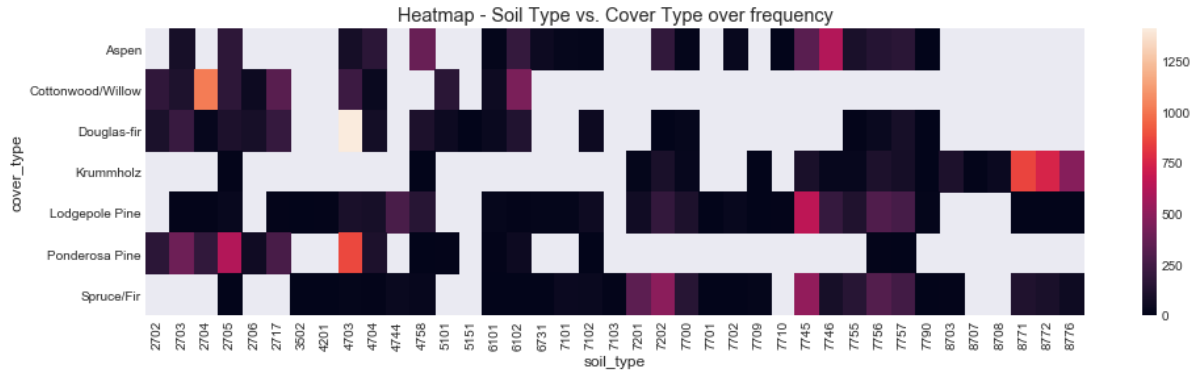
In the following plot, we have two subplots, a heatmap to examine the how target cover types are distributed over soil type. To aggregate the data, we have appended a temparory column 'count'. Basically we are counting how many instances falls in a particular (cover_type level, soil_type level) pair. Second subplot plots mean elevation grouped by soil_type. From second subplot, we can have rough estimate which soil types are more prone to be found at higher elevations.

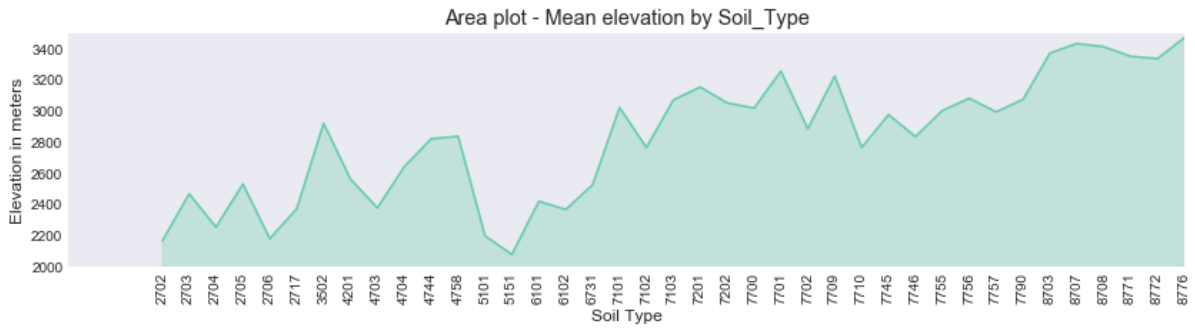
Combining information from these two plots, following observations can be made

- 'Krummholz' forms a dense cluster near Soil types '8771', '8772' & '8776', all of which are found at high altitudes.
- 'Lodgepole Pine' is pretty versatile forest type, it has presence from shallowest elevation of dataset to some of the highest elevations and can be found with most of all observed soil types.
- 'Spruce-fir' & 'Lodgepole Pine' both form a dense cluster on soil types 7745, 7746, 7755, 7756, 7757. These soil types are found at sub alpine elevations, which makes 'Spruce-fir' & 'Lodgepole Pine' a more occupying cover type at sub-alpine heights.
- Aspen also forms a cluster at roughly same height and same soil_types, but in general, the cluster is bit left shifted.
- 'Cottonwood/willow' & 'Ponderosa pine' have a dense occurance on the left side of heatmap, which contains soil types which are prone to appear at lower altitudes.
- Soil type '5151' has exceptionally low elevation and suprisingly low instances count, with only 'some instances of 'Douglas-Fir' occupying 0-250 instances. However, roughlt more than half of 'Douglas-fir' instances (out of 2747), occurs in '4703' soil type.

Some of these observations can be a result of undersampling. but in and all, plots combined do show some intersting insights.

In [18]:
```
temp_data['count'] = 1

df_count_covertype_by_soiltype = temp_data[['soil_type','cover_type','count']]
\
.pivot_table(values='count',index='cover_type',columns='soil_type',aggfunc=np.
sum)
df_mean_elevation_soiltype = temp_data[['soil_type','elevation']]\
.pivot_table(values='elevation',columns='soil_type',aggfunc=np.mean)

x = range(1,len(list(df_mean_elevation_soiltype.columns))+1)
y = df_mean_elevation_soiltype.values[0]

fig, axs = plt.subplots(nrows = 1, ncols=1, figsize=(16,4))

sns.heatmap(df_count_covertype_by_soiltype, annot=False)
plt.title("Heatmap - Soil Type vs. Cover Type over frequency")
plt.show();

plt.figure(figsize=(14,3))
plt.fill_between(x, y, color = "mediumaquamarine", alpha = 0.3)
plt.plot(x, y, color="mediumaquamarine", alpha=1)
plt.xticks(ticks = x,labels = df_mean_elevation_soiltype.columns, rotation = 9
0)
plt.xlim(-2.5,39)
plt.ylim(2000,3500)
plt.ylabel("Elevation in meters")
plt.xlabel("Soil Type")
plt.title("Area plot - Mean elevation by Soil_Type")
plt.show();
```

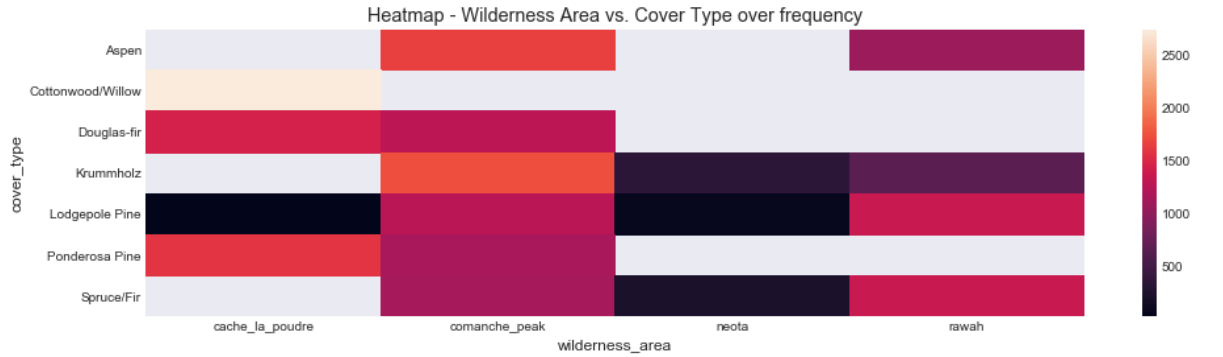




We plotted similar heatmap of 'wilderness area' and 'covertype'. We can make following inferences from the plot,

- 'Cottonwood/Willow' is present only in 'couche la poudre' wilderness. It can mean that 'couche al poudre' has comparatively lower mean elevation.
- 'Lodgepole pine' is present in all 4 areas. which can be attributed to it's greater frequency in dataset.
- 'Aspen' is present in 'commanche peak' and 'rawah' areas.

In [19]:
```
df_count_covertype_by_soiltype = temp_data[['wilderness_area','cover_type','co
unt']]\
.pivot_table(values='count',index='cover_type',columns='wilderness_area',aggfu
nc=np.sum)

x = range(1,len(list(df_mean_elevation_soiltype.columns))+1)
y = df_mean_elevation_soiltype.values[0]


fig, axs = plt.subplots(figsize=(16,4))

sns.heatmap(df_count_covertype_by_soiltype, annot=False)
plt.title("Heatmap - Wilderness Area vs. Cover Type over frequency")
plt.show();
```



Data partition and export

We have partitioned data into training and testing datasets and export them to csv files, which we will be using in the next phase of the project to feed to the ML Algorithms. We have used 70% : 30% ratio for train test pairs. We have made 2 pairs of the dataset, one with reversed one-hot encoded features 'soil_type' & 'wilderness_area' and the other pair with 44 one-hot encoded features. The main reasons for doing that is that Algorithms like decision trees and random forests can directly deal with the categorical features. Adding 44 extra features instead of 2 categorical features will create unnecessary overhead for the algorithm. However, we will use both pairs of dataset to see how encoding affects alogrithm's running time and performance.

We have done integer encoding on the dataset with categorical features 'soil_type' and 'wilderness_areas'.

We will also standardize the data with centering and scaling. We are choosing not to normalize because we have not removed outliers from the data, as they were of significance to us. We will keep both sets of data, one with standardization applied and one without it, to see the effects of stadardization on the dataset.

In [20]:
```
categorical_data = balanced_data[['elevation', 'aspect', 'slope', 'horizontal_
distance_to_hydrology',
            'vertical_distance_to_hydrology', 'horizontal_distance_to_roadway
s',
            'hillshade_9am', 'hillshade_noon', 'hillshade_3pm', 'horizontal_di
stance_to_fire_points',
            'wilderness_area','soil_type','cover_type']]

onehot_data = balanced_data[['elevation', 'aspect', 'slope', 'horizontal_dista
nce_to_hydrology',
            'vertical_distance_to_hydrology', 'horizontal_distance_to_roadway
s',
            'hillshade_9am', 'hillshade_noon', 'hillshade_3pm', 'horizontal_di
stance_to_fire_points',
            'rawah','neota','comanche_peak','cache_la_poudre',
            '2702', '2703', '2704', '2705', '2706', '2717', '3501', '3502', '4
201', '4703',
            '4704', '4744', '4758', '5101', '5151', '6101', '6102', '6731', '7
101', '7102',
            '7103', '7201', '7202', '7700', '7701', '7702', '7709', '7710', '7
745', '7746',
            '7755', '7756','7757', '7790', '8703', '8707', '8708', '8771', '87
72', '8776',
            'cover_type']]

# non onehot encoded data

# encoding the lebels
lb_make = LabelEncoder()
categorical_data['soil_type'] = lb_make.fit_transform(categorical_data['soil_t
ype'])
categorical_data['wilderness_area'] = lb_make.fit_transform(categorical_data[
'wilderness_area'])

categorical_x = categorical_data[['elevation', 'aspect', 'slope', 'horizontal_
distance_to_hydrology',
            'vertical_distance_to_hydrology', 'horizontal_distance_to_roadway
s',
            'hillshade_9am', 'hillshade_noon', 'hillshade_3pm', 'horizontal_di
stance_to_fire_points',
            'wilderness_area','soil_type']]
categorical_x.shape
categorical_y = categorical_data['cover_type']
categorical_scaled_x = preprocessing.MinMaxScaler().fit_transform(categorical_
x)
categorical_scaled_x.shape
categorical_train_data, categorical_test_data, categorical_train_labels, categ
orical_test_labels =\
train_test_split(categorical_scaled_x, categorical_y, test_size=0.30, random_s
tate=42)

# onehot encoded data
onehot_data_x = balanced_data[['elevation', 'aspect', 'slope', 'horizontal_dis
tance_to_hydrology',
            'vertical_distance_to_hydrology', 'horizontal_distance_to_roadway
s',
```

```
            'hillshade_9am', 'hillshade_noon', 'hillshade_3pm', 'horizontal_di
stance_to_fire_points',
            'rawah','neota','comanche_peak','cache_la_poudre',
            '2702', '2703', '2704', '2705', '2706', '2717', '3501', '3502', '4
201', '4703',
            '4704', '4744', '4758', '5101', '5151', '6101', '6102', '6731', '7
101', '7102',
            '7103', '7201', '7202', '7700', '7701', '7702', '7709', '7710', '7
745', '7746',
            '7755', '7756','7757', '7790', '8703', '8707', '8708', '8771', '87
72', '8776']]
onehot_data_y = balanced_data['cover_type']
onehot_data_scaled_x = preprocessing.MinMaxScaler().fit_transform(onehot_data_
x)
onehot_train_data, onehot_test_data, onehot_train_labels, onehot_test_labels =
\
train_test_split(onehot_data_scaled_x, onehot_data_y, test_size=0.30, random_s
tate=42)


pd.DataFrame(categorical_train_data).to_csv(r'categorical_train_data.csv',head
er=None, index=None)
pd.DataFrame(categorical_test_data).to_csv(r'categorical_test_data.csv',header
=None, index=None)
pd.DataFrame(categorical_train_labels).to_csv(r'categorical_train_labels.csv',
header=None, index=None)
pd.DataFrame(categorical_test_labels).to_csv(r'categorical_test_labels.csv',he
ader=None, index=None)


pd.DataFrame(onehot_train_data).to_csv(r'onehot_train_data.csv',header=None, i
ndex=None)
pd.DataFrame(onehot_test_data).to_csv(r'onehot_test_data.csv',header=None, ind
ex=None)
pd.DataFrame(onehot_train_labels).to_csv(r'onehot_train_labels.csv',header=Non
e, index=None)
pd.DataFrame(onehot_test_labels).to_csv(r'onehot_test_labels.csv',header=None,
index=None)
```

```
C:\Users\jigar\Anaconda3\lib\site-packages\ipykernel_launcher.py:20: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
C:\Users\jigar\Anaconda3\lib\site-packages\ipykernel_launcher.py:21: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
```

Out[20]:
```
(19229, 12)
```

```
C:\Users\jigar\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:323:
DataConversionWarning: Data with input dtype int32, int64 were all converted
to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

Out[20]:
```
(19229, 12)
```

```
C:\Users\jigar\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:323:
DataConversionWarning: Data with input dtype int64 were all converted to floa
t64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

Summary

In Phase 1, We have imputed Null values in 'RainToday' feature with mode and in other continuous feature with mean. We have dropped instances with nulls in either 'WindGustDir','WindDir9am','WindDir3pm' features. We have not removed or imputed outliers due to their statistical significance. We have dropped "Date" and 'Risk_MM' features as they were not required for purpose of this project. We have dropped "Cloud9am", "Cloud3pm", "Sunshine" & "Evaporation" features because of high (>35%) presence of nulls among them. From exploration section we found out that 9am & 3pm feature pairs are strongly correlation and we will have to consider this correlation while fitting models. We also normalized feature 'Rainfall' using MinMax normalization. we discovered how features are distributed and how this distribution serves to our target feature 'RainTomorrow'.

Significant Continuous Features - "MinTemp", "MaxTemp", "Temp9am", "Temp3pm", "Pressure9am", "Pressure3pm", "Humidity9am", "Humidity3pm", "WindSpeed9am", "WindSpeed3pm", "WindGustSpeed", "Rainfall"

Significant Categorical Features - "WindGustDir", "WindDir9am", "WindDir3pm", "RainToday"