R Mini-Project: Predicting Forest Cover Type

Problem Statement:

This project aims at predicting the most probable forest cover type arising in regions of $30x30m$ regions using only cartographic variables. Cartographic Variables, also known as $VisualVariables$, refer to aspects of a graphical object that can visually differentiate it from other objects such as the size, orientation, color and so on. This dataset provides a set of variables that are remotely-sensed to train a model to make predictions. Various Machine Learning and Deep Learning Algorithms are used to perform multi-class classification problem and predict the forest cover type using only provided set of cartographic variables.

Forest Cover-Type Dataset:

Natural resource managers responsible for developing ecosystem management strategies require basic descriptive information including inventory data for forested lands to support their decision-making processes. However, managers generally do not have this type of data for in holdings or neighboring lands that are outside their immediate jurisdiction. One method of obtaining this information is through the use of predictive models.

The study area included four wilderness areas found in the Roosevelt National Forest of northern Colorado. A total of twelve cartographic measures were utilized as independent variables in the predictive models, while seven major forest cover types were used as dependent variables. Several subsets of these variables were examined to determine the best overall predictive model.

Relevant Information:

Predicting forest cover type from cartographic variables only (only remotely sensed data). The actual forest cover type for a given observation ($30x30$ $metercell$) was determined from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. Independent variables were derived from data originally obtained from US Geological Survey (USGS) and USFS data. Data is in raw form (not scaled) and contains binary (0 or 1) columns of data for qualitative independent variables (wilderness areas and soil types).

This study area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. These areas represent forests with minimal human-caused disturbances, so that existing forest cover types are more a result of ecological processes rather than forest management practices.

Some background information for these four wilderness areas:
$Neota$ (area 2) probably has the highest mean elevational value of the 4 wilderness areas. $Rawah$ (area 1) and $ComanchePeak$ (area 3) would have a lower mean elevational value, while $CachelaPoudre$ (area 4) would have the lowest mean elevational value.

As for primary major tree species in these areas, Neota would have $spruce/fir$ (type 1), while Rawah and Comanche Peak would probably have $lodgepolepine$ (type 2) as their primary species, followed by $spruce/fir$ and $aspen$ (type 5). Cache la Poudre would tend to have $Ponderosapine$ (type 3), $Douglas - fir$ (type 6), and $cottonwood/willow$ (type 4).

The Rawah and Comanche Peak areas would tend to be more typical of the overall dataset than either the Neota or Cache la Poudre, due to their assortment of tree species and range of predictive variable values (elevation, etc.) Cache la Poudre would probably be more unique than the others, due to its relatively low elevation range and species composition.

```r
# Import packages
library("keras")
use_condaenv("dlenv")
library("tfdatasets")
library("ggplot2")
library("Rtsne")
library("plotly")
library("ramify")
library("dplyr")
library("GGally")
library("datasets")
library("e1071")
library("tidyverse")    # data manipulation and visualization
library("kernlab")      # SVM methodology
library("ISLR")         # contains example data set "Khan"
library("RColorBrewer") # customized coloring of plots
library("class")        # KNN Implementation

# Setting the current working directory
setwd("C:\\RDemo\\R_Mini_Project\\Kaggle\\")

dataset_df = read.csv("covtype.csv", header = TRUE)

# Creating a data structure to hold classes
cov_type_classes_list = list(
  "Spruce/Fir" = 1,
  "Lodgepole Pine" = 2,
  "Ponderosa Pine" = 3,
  "Cottonwood/Willowr" = 4,
  "Aspen" = 5,
  "Douglas-fir" = 6,
  "Krummholz" = 7
)

print(cov_type_classes_list)
```

```
## $`Spruce/Fir`
## [1] 1
##
## $`Lodgepole Pine`
## [1] 2
##
## $`Ponderosa Pine`
## [1] 3
##
## $`Cottonwood/Willowr`
## [1] 4
##
## $Aspen
## [1] 5
##
## $`Douglas-fir`
## [1] 6
##
```

```
## $Krummholz
## [1] 7
```

```
# Specify the cover types
class = data.frame(dataset_df$Cover_Type)
colnames(class) <- c("cover_type_class")

# Give names -- convert class to categorical
class[class == 1] = "Spruce/Fir"
class[class == 2] = "Lodgepole Pine"
class[class == 3] = "Ponderosa Pine"
class[class == 4] = "Cottonwood/Willowr"
class[class == 5] = "Aspen"
class[class == 6] = "Douglas-fir"
class[class == 7] = "Krummholz"

# Check if it is done properly
for(i in 50:75){
  cat(dataset_df$Cover_Type[i], class$cover_type_class[i], "\n")
}
```

```
## 5 Aspen
## 5 Aspen
## 1 Spruce/Fir
## 1 Spruce/Fir
## 5 Aspen
## 5 Aspen
## 1 Spruce/Fir
## 5 Aspen
## 5 Aspen
## 5 Aspen
## 5 Aspen
## 5 Aspen
## 2 Lodgepole Pine
## 2 Lodgepole Pine
## 5 Aspen
## 5 Aspen
## 5 Aspen
## 5 Aspen
## 1 Spruce/Fir
## 2 Lodgepole Pine
## 2 Lodgepole Pine
## 2 Lodgepole Pine
## 2 Lodgepole Pine
## 2 Lodgepole Pine
## 2 Lodgepole Pine
## 2 Lodgepole Pine
```

```
# Create area column
area = data.frame(dataset_df$Cover_Type)
colnames(area) <- "Wilderness_Area"
area$Wilderness_Area[dataset_df$Wilderness_Area1 == 1] = "Rawah"
area$Wilderness_Area[dataset_df$Wilderness_Area2 == 1] = "Neota"
area$Wilderness_Area[dataset_df$Wilderness_Area3 == 1] = "Comanche Peak"
area$Wilderness_Area[dataset_df$Wilderness_Area4 == 1] = "Cache la Poudre"
```

```r
# Check
area$Wilderness_Area[400200:400250]
```

```
##  [1] "Comanche Peak" "Comanche Peak" "Comanche Peak" "Comanche Peak"
##  [5] "Comanche Peak" "Comanche Peak" "Comanche Peak" "Comanche Peak"
##  [9] "Comanche Peak" "Comanche Peak" "Comanche Peak" "Comanche Peak"
## [13] "Comanche Peak" "Comanche Peak" "Comanche Peak" "Comanche Peak"
## [17] "Comanche Peak" "Comanche Peak" "Comanche Peak" "Comanche Peak"
## [21] "Comanche Peak" "Comanche Peak" "Comanche Peak" "Comanche Peak"
## [25] "Comanche Peak" "Comanche Peak" "Comanche Peak" "Comanche Peak"
## [29] "Comanche Peak" "Comanche Peak" "Comanche Peak" "Comanche Peak"
## [33] "Comanche Peak" "Comanche Peak" "Comanche Peak" "Comanche Peak"
## [37] "Comanche Peak" "Rawah"         "Rawah"         "Rawah"
## [41] "Rawah"         "Rawah"         "Rawah"         "Rawah"
## [45] "Rawah"         "Rawah"         "Rawah"         "Rawah"
## [49] "Rawah"         "Rawah"         "Rawah"
```

```r
# Integer Encoding Areas.
# The encoded data structure will be a list
wilderness_area_enc = list(
  "Rawah" = 1,
  "Neota" = 2,
  "Comanche Peak" = 3,
  "Cache la Poudre" = 4
)
print(wilderness_area_enc)
```

```
## $Rawah
## [1] 1
##
## $Neota
## [1] 2
##
## $`Comanche Peak`
## [1] 3
##
## $`Cache la Poudre`
## [1] 4
```

```r
for (wild_area_enc in wilderness_area_enc){
  area$Integer_Wilderness_Area[area$Wilderness_Area == names(wilderness_area_enc)[[wild_area_enc]]] <- 
}

# Sanity Check
glimpse(area)
```

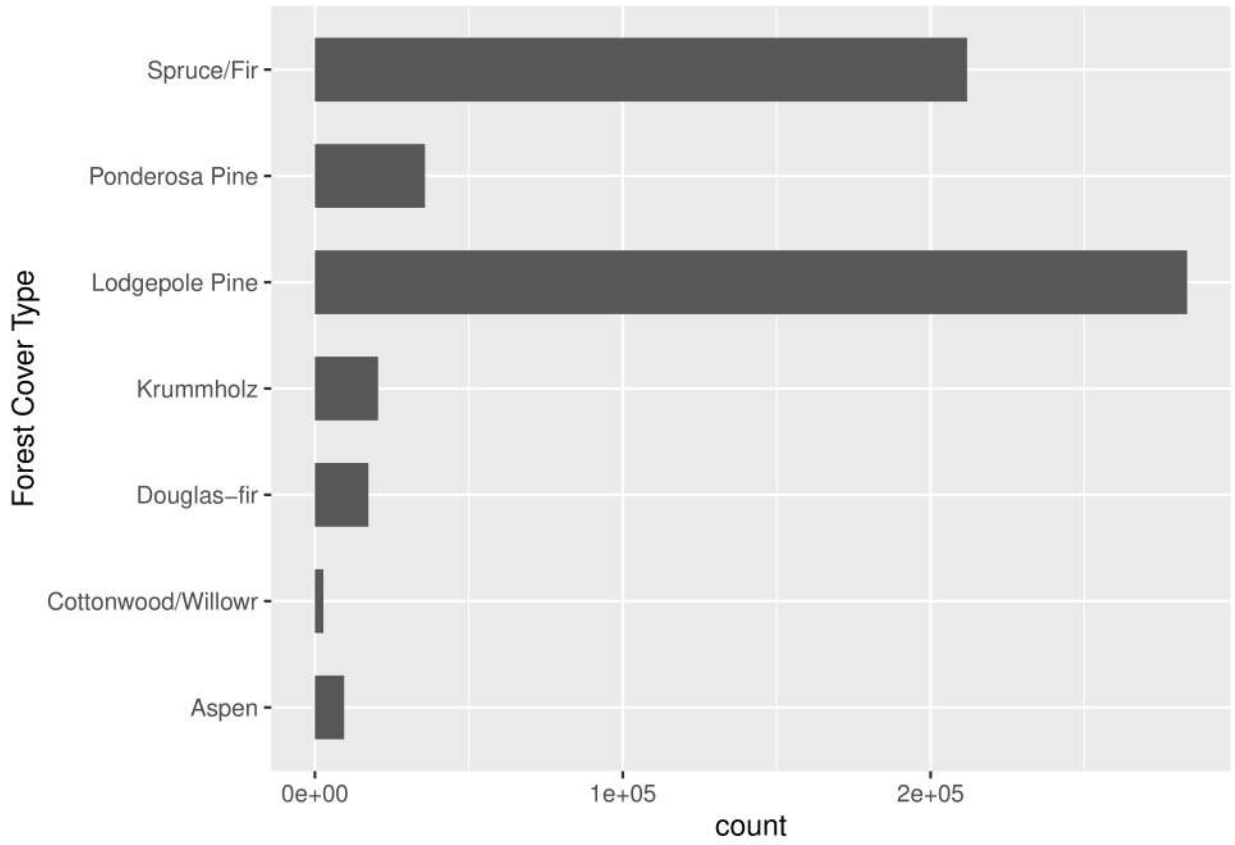
```
## Rows: 581,012
## Columns: 2
## $ Wilderness_Area         <chr> "Rawah", "Rawah", "Rawah", "Rawah", "Rawah", "~
## $ Integer_Wilderness_Area <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
```

```r
# Get some statistical analysis for all columns except
# the soil type, cover type and the wilderness area.

# 1 to 10 are all the columns that we need
```

```
summary(data.frame(dataset_df[ , 1:10]))
```

```
##    Elevation        Aspect          Slope      Horizontal_Distance_To_Hydrology
##  Min.   :1859   Min.   :  0.0   Min.   : 0.0   Min.   :   0.0
##  1st Qu.:2809   1st Qu.: 58.0   1st Qu.: 9.0   1st Qu.: 108.0
##  Median :2996   Median :127.0   Median :13.0   Median : 218.0
##  Mean   :2959   Mean   :155.7   Mean   :14.1   Mean   : 269.4
##  3rd Qu.:3163   3rd Qu.:260.0   3rd Qu.:18.0   3rd Qu.: 384.0
##  Max.   :3858   Max.   :360.0   Max.   :66.0   Max.   :1397.0
##  Vertical_Distance_To_Hydrology Horizontal_Distance_To_Roadways Hillshade_9am
##  Min.   :-173.00                Min.   :   0                    Min.   :  0.0
##  1st Qu.:   7.00                1st Qu.:1106                    1st Qu.:198.0
##  Median :  30.00                Median :1997                    Median :218.0
##  Mean   :  46.42                Mean   :2350                    Mean   :212.1
##  3rd Qu.:  69.00                3rd Qu.:3328                    3rd Qu.:231.0
##  Max.   : 601.00                Max.   :7117                    Max.   :254.0
##  Hillshade_Noon  Hillshade_3pm   Horizontal_Distance_To_Fire_Points
##  Min.   :  0.0   Min.   :  0.0   Min.   :   0
##  1st Qu.:213.0   1st Qu.:119.0   1st Qu.:1024
##  Median :226.0   Median :143.0   Median :1710
##  Mean   :223.3   Mean   :142.5   Mean   :1980
##  3rd Qu.:237.0   3rd Qu.:168.0   3rd Qu.:2550
##  Max.   :254.0   Max.   :254.0   Max.   :7173
```

```
# Total number of Entries per cover type
entries_per_cov_type = ggplot(class, aes(x=cover_type_class)) +
  geom_bar(width = 0.60) +
  scale_fill_grey(start = 0.00, end = 0.9) +
  theme(legend.position = "top") +
  coord_flip() +
  xlab("Forest Cover Type")

print(entries_per_cov_type)
```



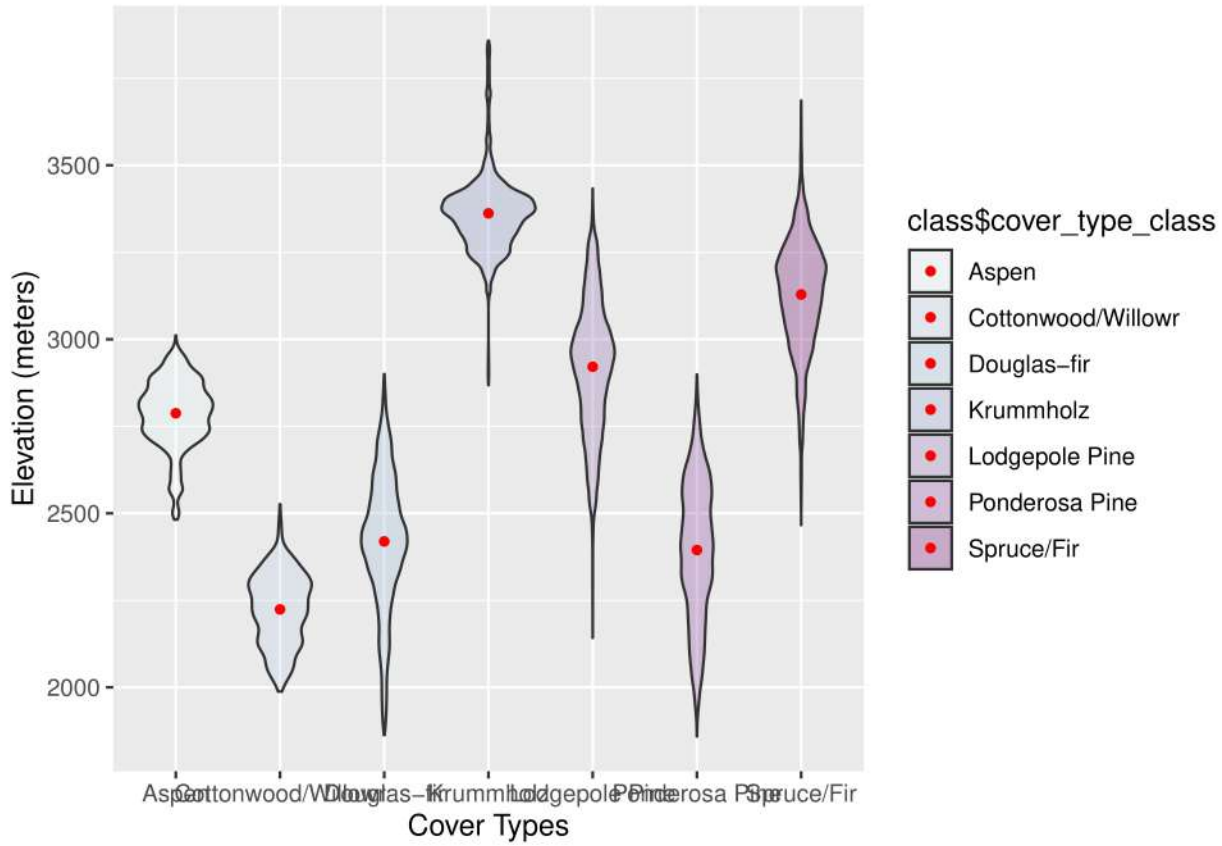
Exploratory Analysis:

Univariate Analysis:

Violin / Boxplots:

- Elevation attribute is extremely effective in separating the different Forest Cover Class types and can be decisive in making predictions.

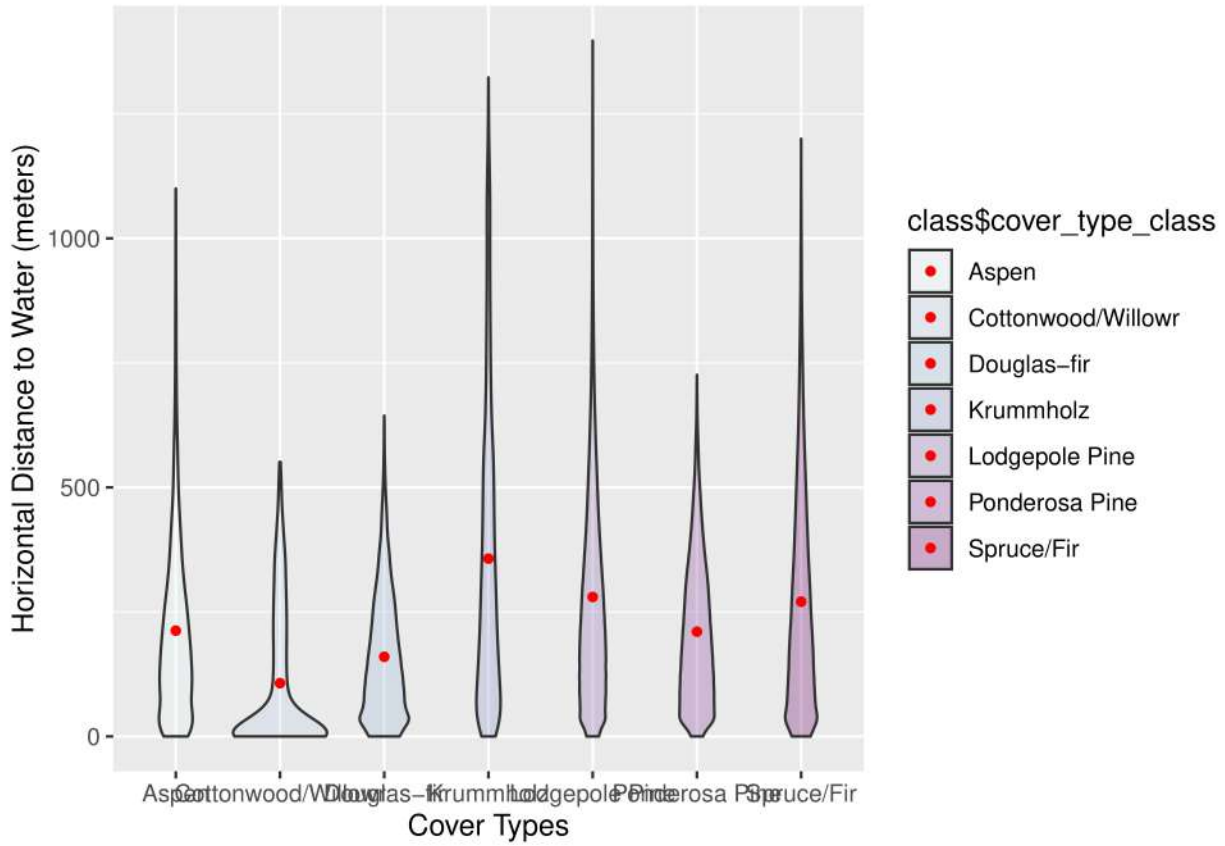
```
violin_elevation_all_types = ggplot(dataset_df, aes(x = class$cover_type_class, y = Elevation,
                                                    fill = class$cover_type_class)) +
  geom_violin(alpha = 0.3) +
  xlab("Cover Types") +
  ylab("Elevation (meters)") +
  scale_fill_brewer(palette="BuPu")+
  stat_summary(fun=mean, geom="point", shape=20, size=2, color="red")

print(violin_elevation_all_types)
```



* Slope and Azimuth do not separate Forest Cover Classes on their own. It is possible that these attributes combined with other can prove to be useful.

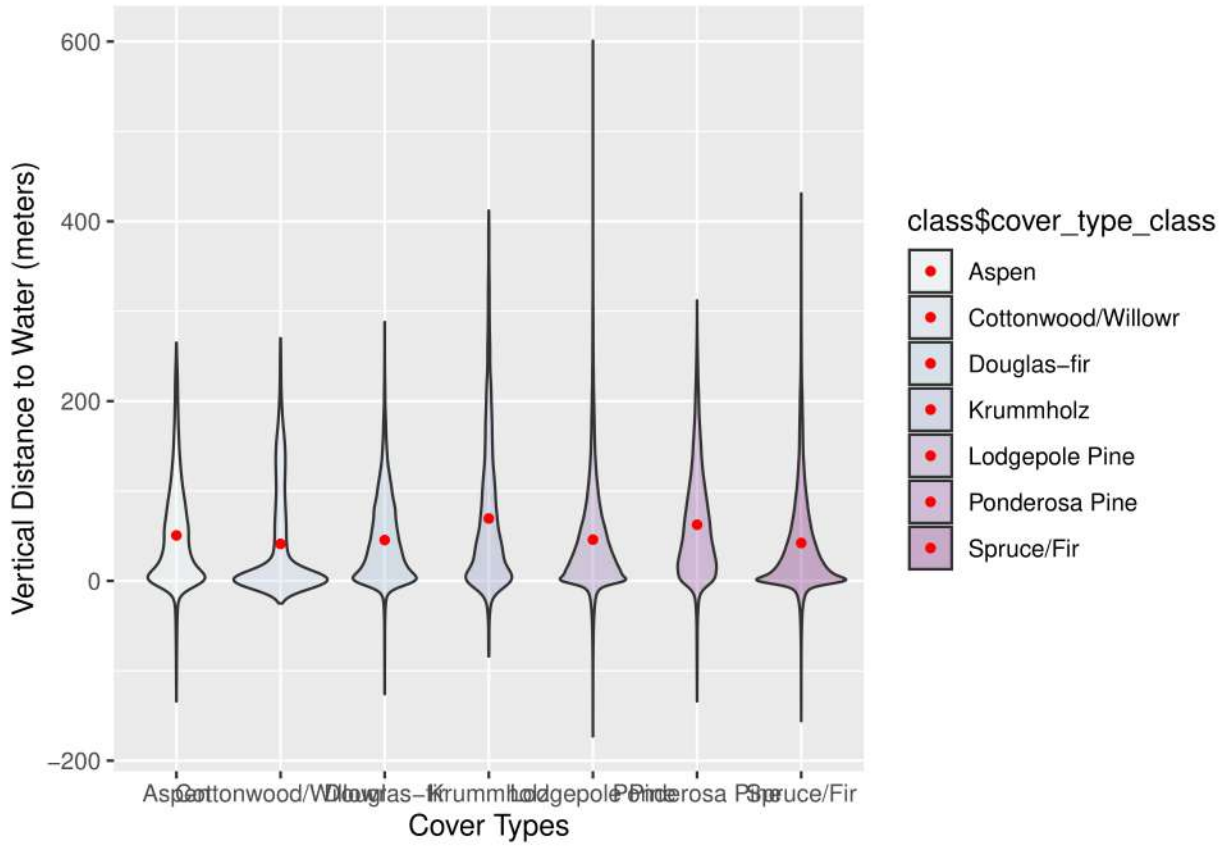
- From the Horizontal Distance to Hydrology Violin Plot we can determine that the cover types of *Krummholz*, *LodgepolePine* and *Spruce/Fir* are comfortable growing really close and far away from water bodies and can be useful in classifying these classes.

```
## Horizontal Distance to Hydrology
violin_hzh_all_types = ggplot(dataset_df, aes(x = class$cover_type_class,
                                              y = Horizontal_Distance_To_Hydrology,
                                              fill = class$cover_type_class)) +
  geom_violin(alpha = 0.3) +
  xlab("Cover Types") +
  ylab("Horizontal Distance to Water (meters)") +
  scale_fill_brewer(palette="BuPu")+
  stat_summary(fun=mean, geom="point", shape=20, size=2, color="red")

print(violin_hzh_all_types)
```



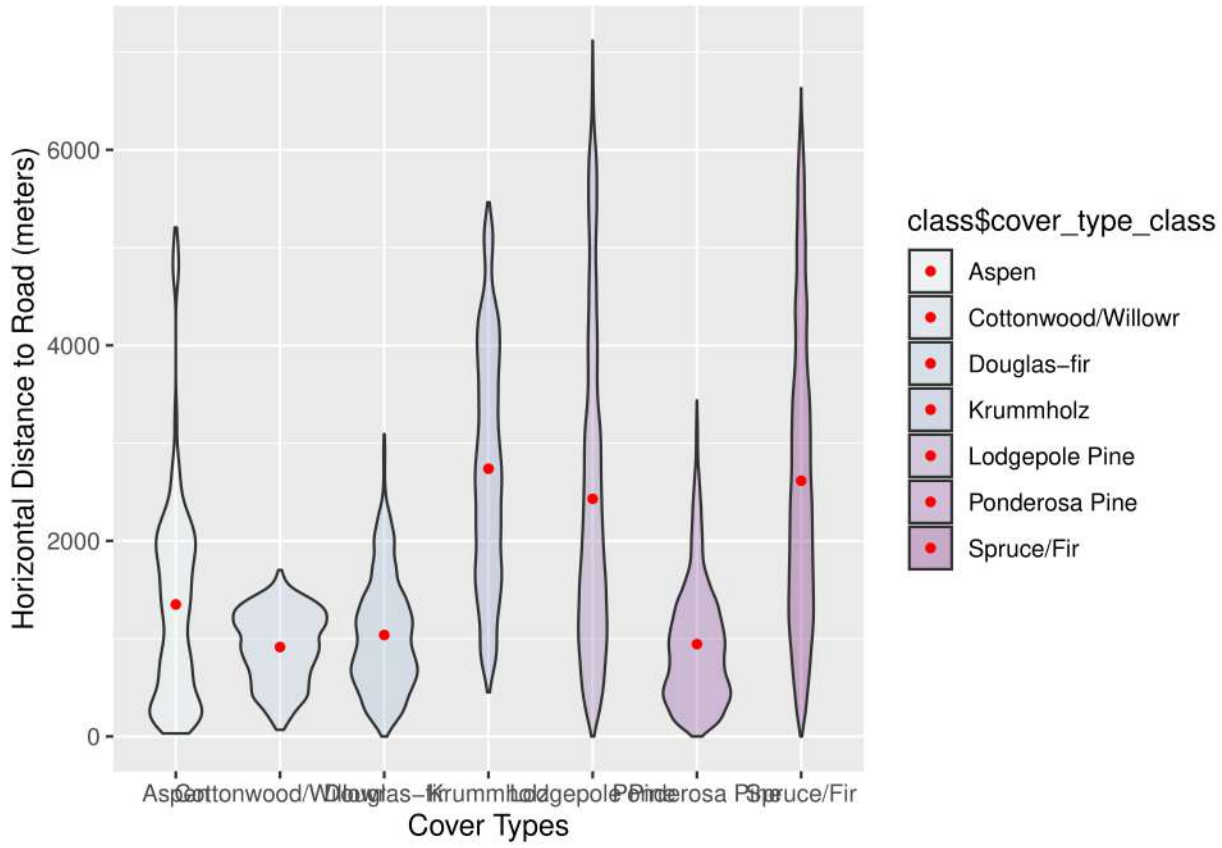
- Again, *Lodgepolepine* is comfortable growing far away from vertical water source and hence it can be a decisive factor in determining it.

```
## Vertical Distance to Hydrology
violin_vth_all_types = ggplot(dataset_df, aes(x = class$cover_type_class,
                                              y = Vertical_Distance_To_Hydrology,
                                              fill = class$cover_type_class)) +
  geom_violin(alpha = 0.3) +
  xlab("Cover Types") +
  ylab("Vertical Distance to Water (meters)") +
  scale_fill_brewer(palette="BuPu")+
  stat_summary(fun=mean, geom="point", shape=20, size=2, color="red")

print(violin_vth_all_types)
```



- From the Horizontal Distance to Hydrology Violin Plot we can determine that the cover types of $Krummholz$, $LodgepolePine$ and $Spruce/Fir$ are comfortable growing really close and far away from roadways. This has the potential to classify the aforementioned types and well as determine that the space under evaluation is not of other types.

```
violin_hzr_all_types = ggplot(dataset_df, aes(x = class$cover_type_class,
                                              y = Horizontal_Distance_To_Roadways,
                                              fill = class$cover_type_class)) +
  geom_violin(alpha = 0.3) +
  xlab("Cover Types") +
  ylab("Horizontal Distance to Road (meters)") +
  scale_fill_brewer(palette="BuPu")+
  stat_summary(fun=mean, geom="point", shape=20, size=2, color="red")

print(violin_hzr_all_types)
```

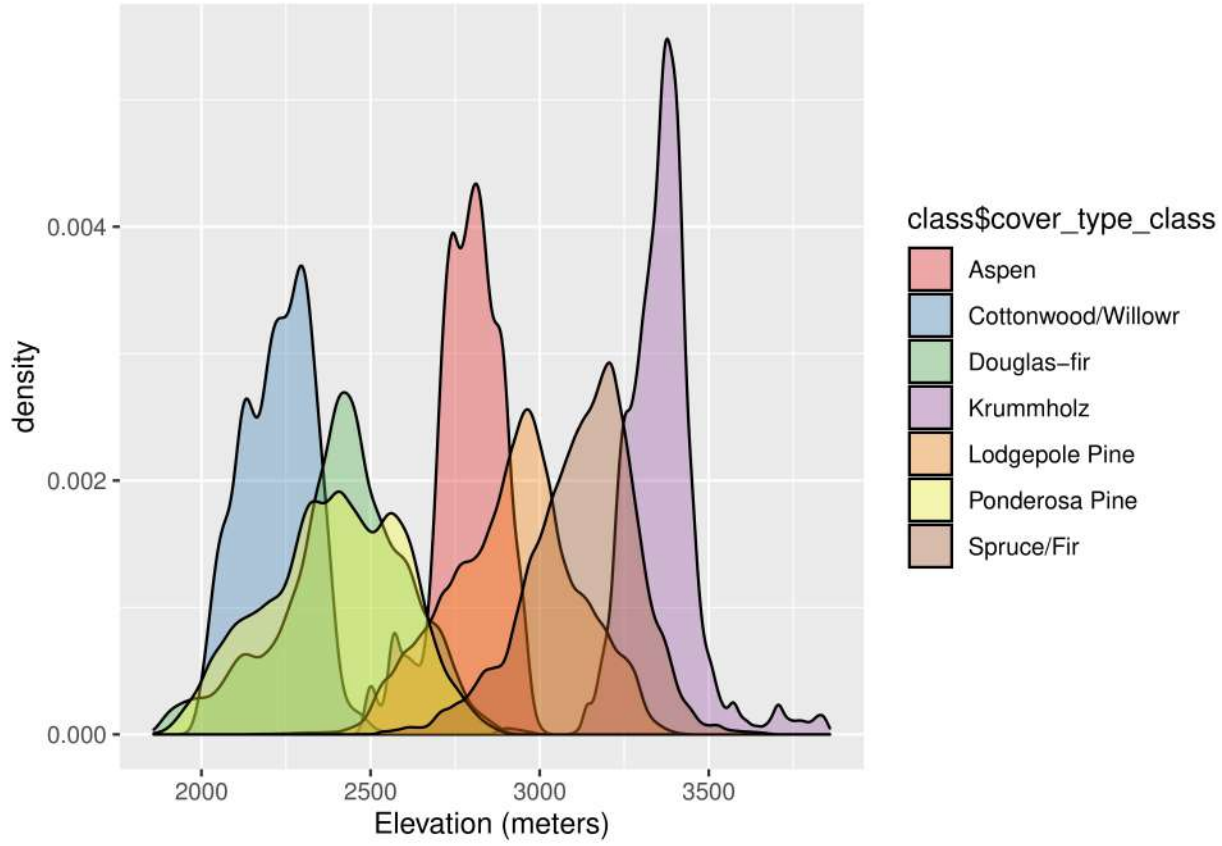


- Horizontal Distance to Fire-Points do not separate any specific class of cover type.

Histograms / Density Plots

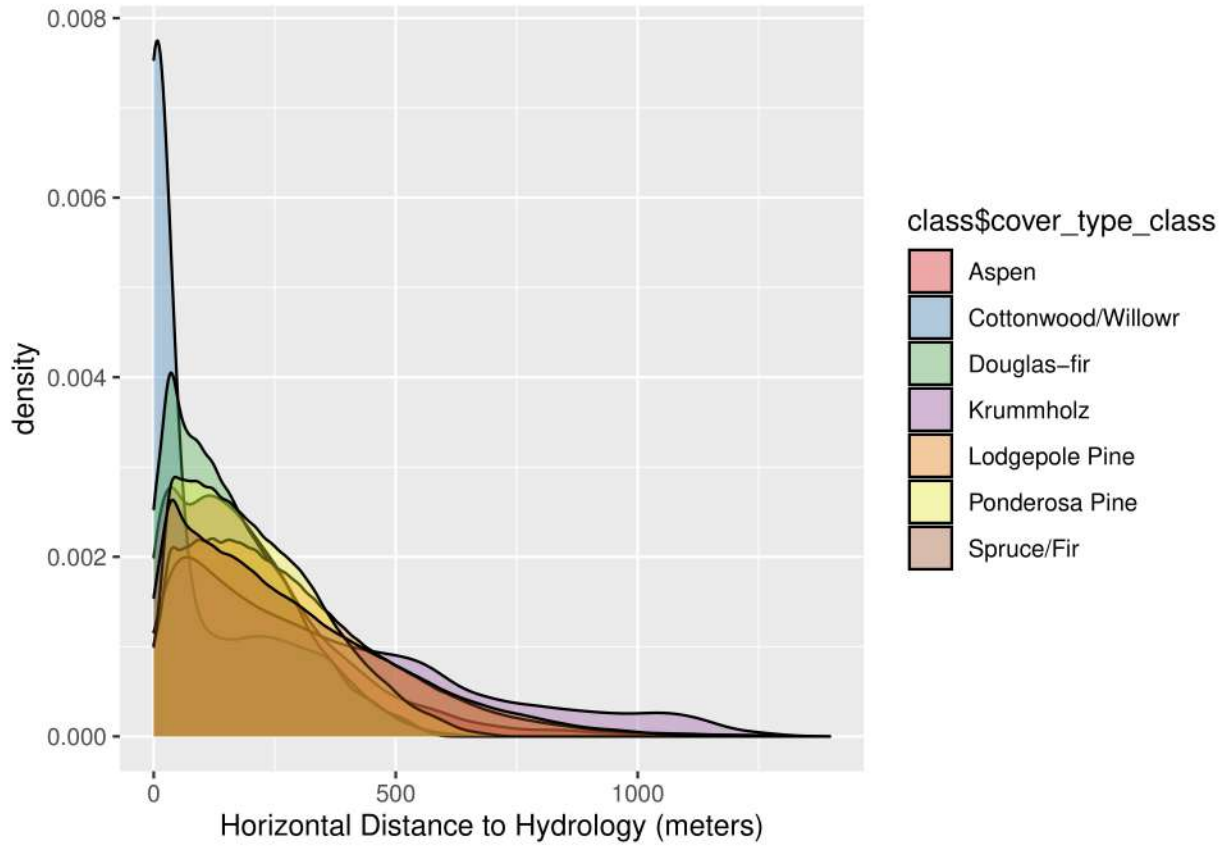
- It can be confirmed that elevation is an important attribute for classification of Forest Cover Types.

```
# Elevation
den_elevation_all = dataset_df %>%
  ggplot(aes(x = Elevation, fill = class$cover_type_class)) +
  geom_density(alpha = 0.35, position = "identity") +
  scale_fill_brewer(palette="Set1") +
  xlab("Elevation (meters)")

print(den_elevation_all)
```

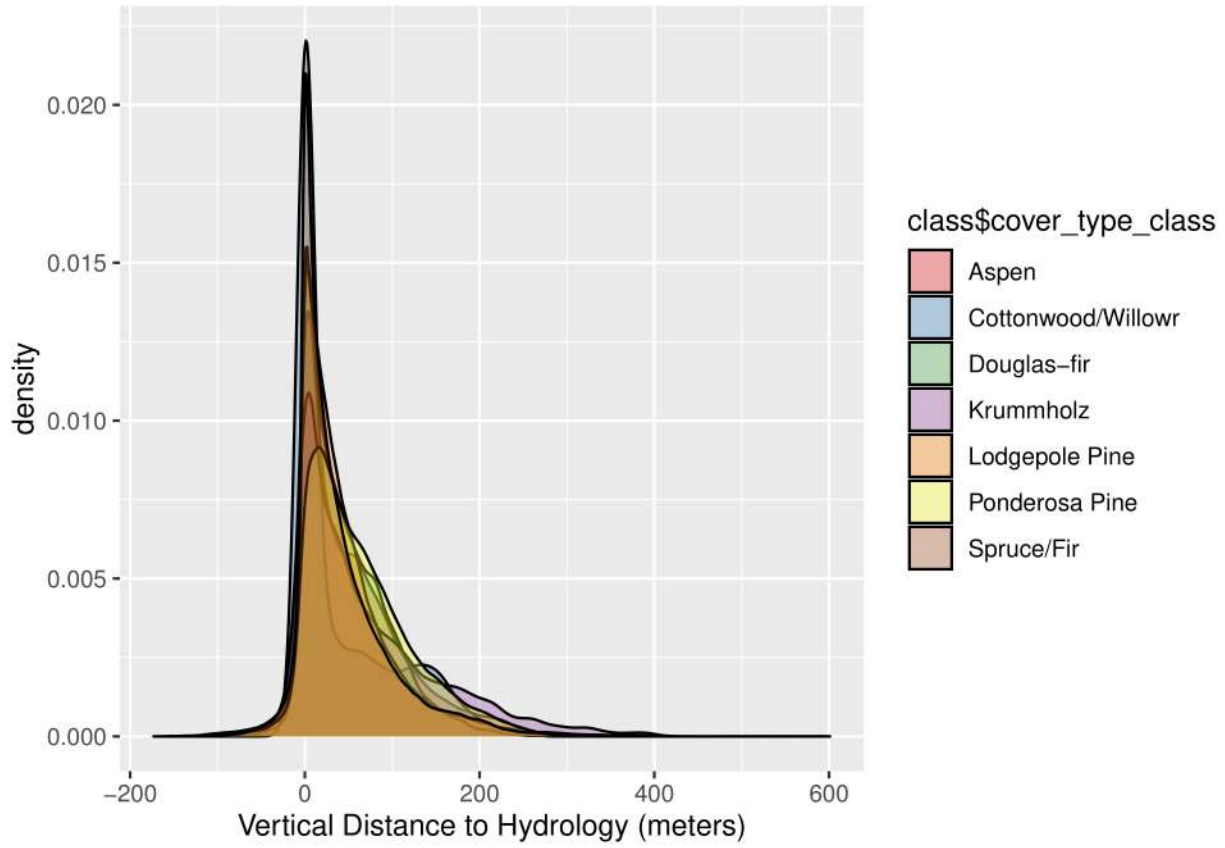


- Adding to our previous knowledge of relationships between the distance to water and type of forest cover, we can infer from the next plot that *Cottonwood/Willowr* grows close to water bodies at about roughly 500 meters.

```
## Horizontal Distance to water
den_hzh_all = dataset_df %>%
  ggplot(aes(x = Horizontal_Distance_To_Hydrology, fill = class$cover_type_class)) +
  geom_density(alpha = 0.35, position = "identity") +
  scale_fill_brewer(palette="Set1") +
  xlab("Horizontal Distance to Hydrology (meters)")

print(den_hzh_all)
```

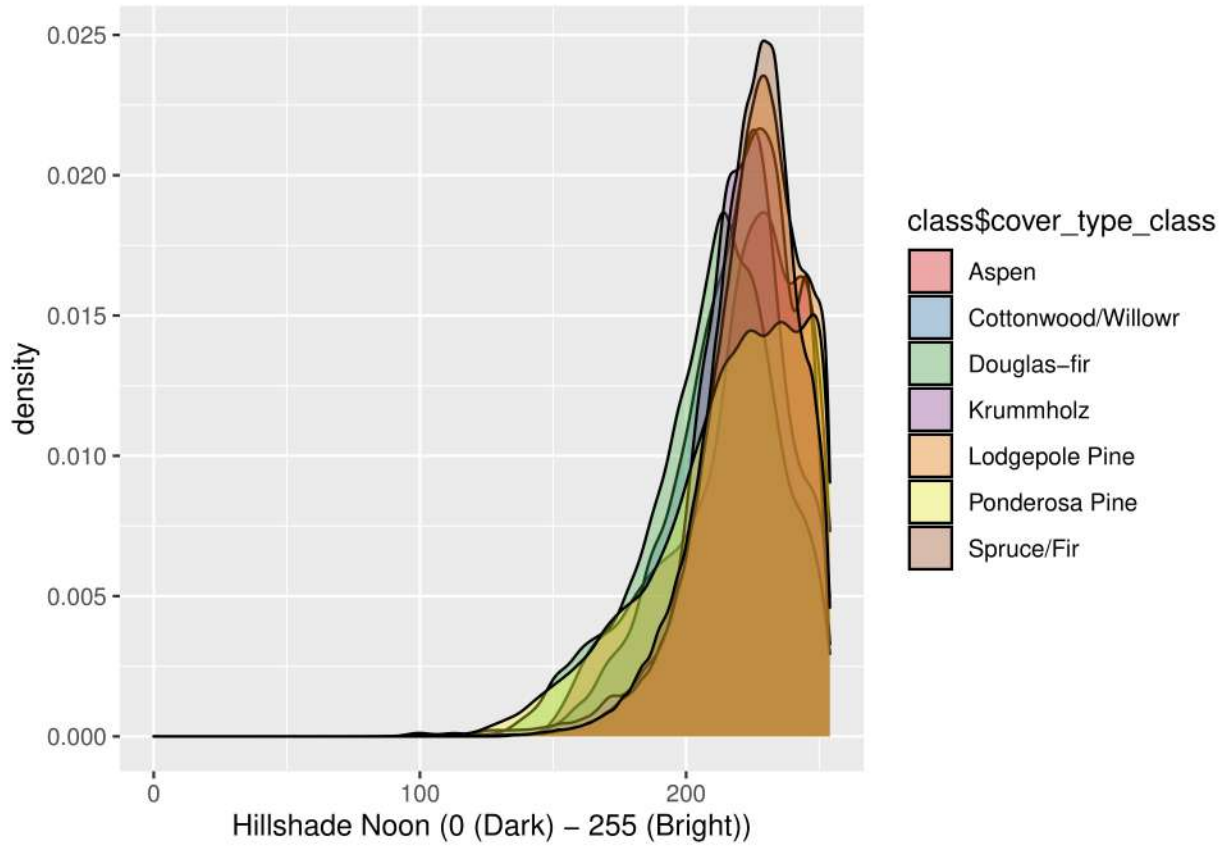



```
## Vertical Distance to water
den_vth_all = dataset_df %>%
  ggplot(aes(x = Vertical_Distance_To_Hydrology, fill = class$cover_type_class)) +
  geom_density(alpha = 0.35, position = "identity") +
  scale_fill_brewer(palette="Set1") +
  xlab("Vertical Distance to Hydrology (meters)")

print(den_vth_all)
```

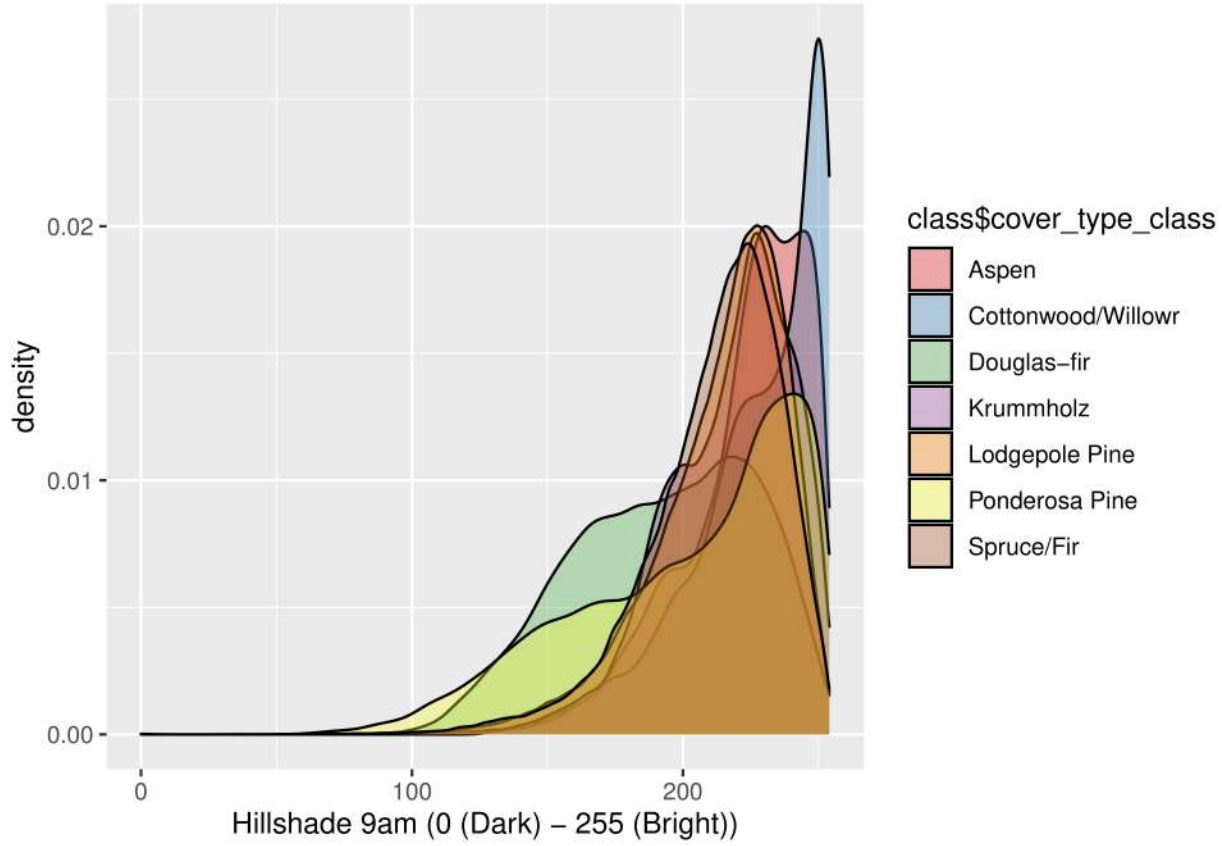


- From the plot below we can infer that all hillshade measurements are inconclusive in determining the type of forest.

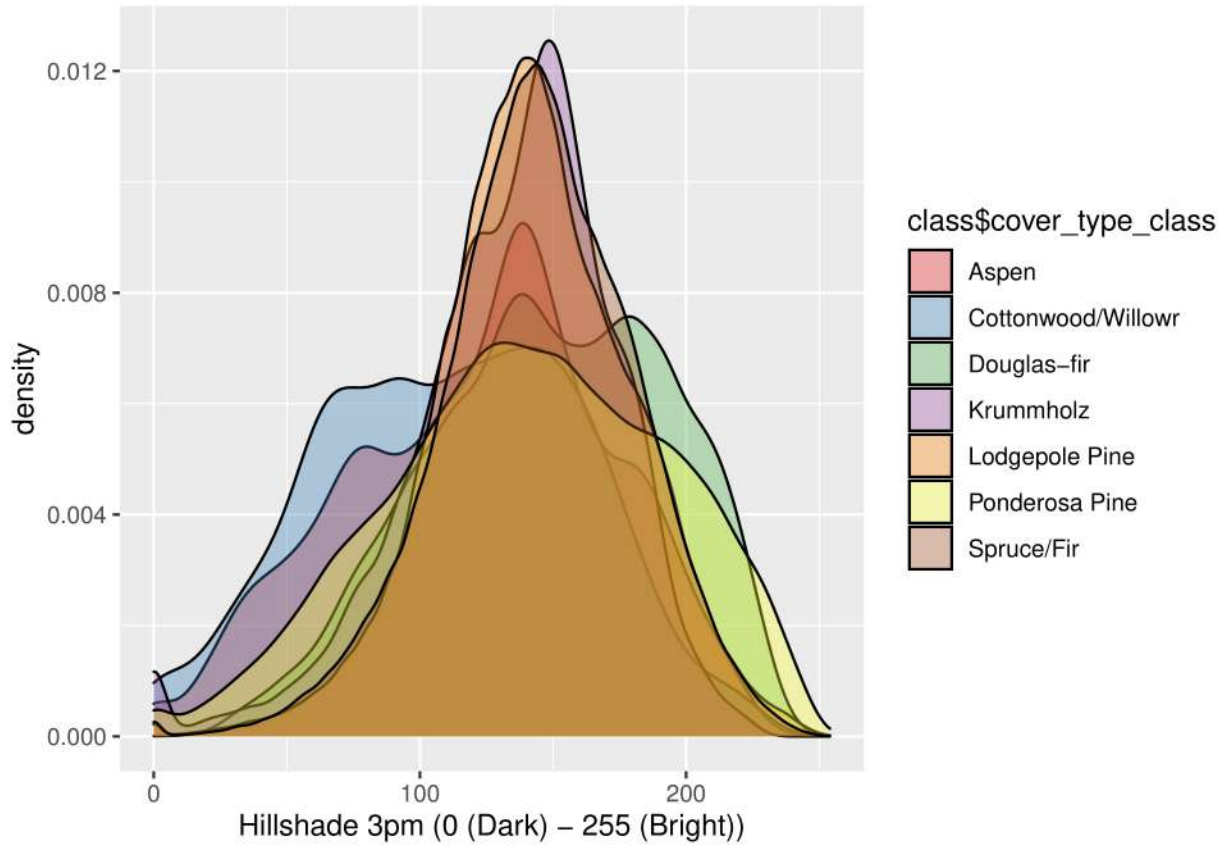
```
## Hillshade Noon

den_hsn_all = dataset_df %>%
  ggplot(aes(x = Hillshade_Noon, fill = class$cover_type_class)) +
  geom_density(alpha = 0.35, position = "identity") +
  scale_fill_brewer(palette="Set1") +
  xlab("Hillshade Noon (0 (Dark) - 255 (Bright))")

print(den_hsn_all)
```




```
## Hillshade 9am

den_hsn9_all = dataset_df %>%
  ggplot(aes(x = Hillshade_9am, fill = class$cover_type_class)) +
  geom_density(alpha = 0.35, position = "identity") +
  scale_fill_brewer(palette="Set1") +
  xlab("Hillshade 9am (0 (Dark) - 255 (Bright))")

print(den_hsn9_all)
```




```
## Hillshade 3pm
den_hsn3_all = dataset_df %>%
  ggplot(aes(x = Hillshade_3pm, fill = class$cover_type_class)) +
  geom_density(alpha = 0.35, position = "identity") +
  scale_fill_brewer(palette="Set1") +
  xlab("Hillshade 3pm (0 (Dark) - 255 (Bright))")

print(den_hsn3_all)
```

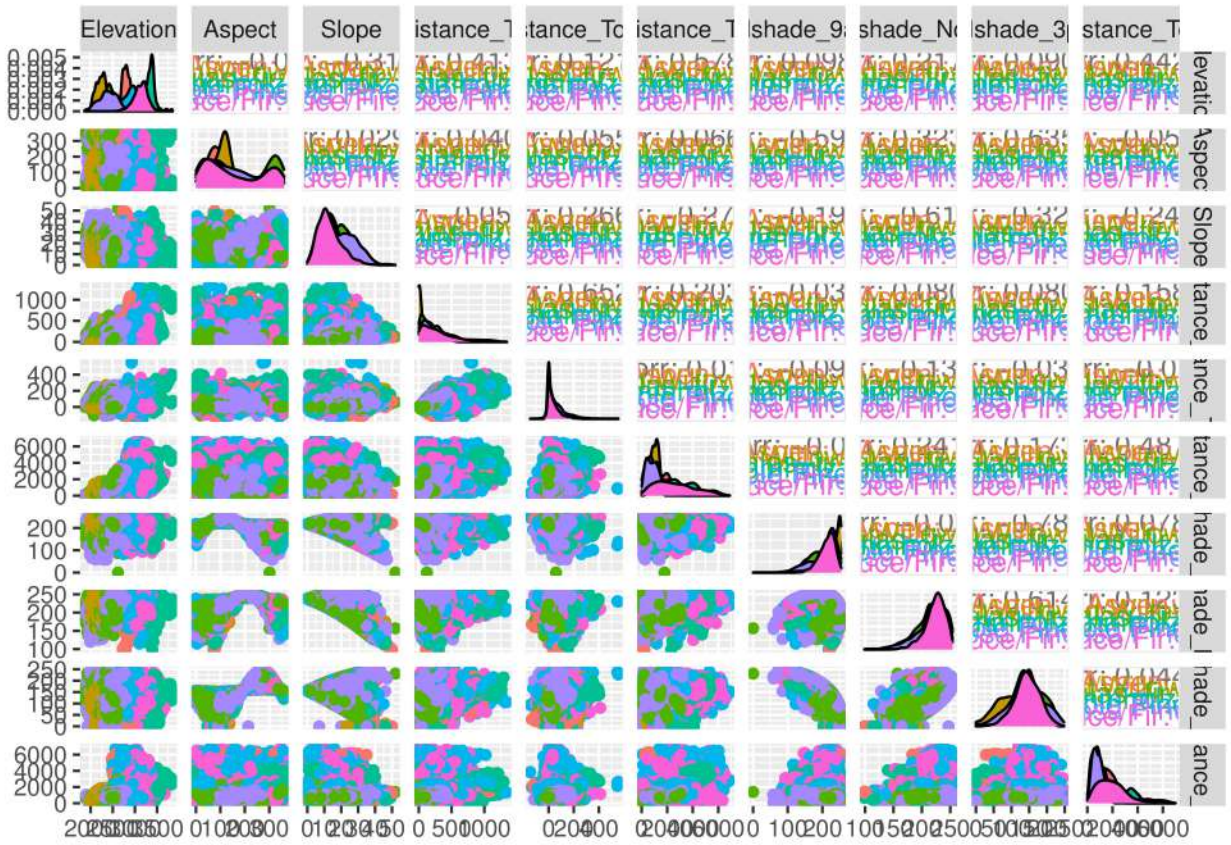


BiVariate Analysis:

Scatter Plots:

- From the scatterplot we can infer that Elevation with other attributes such as Aspect, Slope and various Hillshade measurements is able to separate the different cover types to a certain extent.
- Also, a negative correlation between slope and all different types of Hillshade measurements can be seen.
- Slighlty positive correlation can be seen between Hillshade 9am and Hillshade at 3pm and Noon and Horizontal Distance to Hydrology.
- Hillshade at noon also forms slight positive correlation with Hillshade at 3 pm and Horizontal distance to Hydrology attribute.

```
# Scatter Plots
subset_df = data.frame(dataset_df[1:15000, 1:10])
colnames(subset_df) <- colnames(dataset_df)[1:10]

scatter_matrix = ggpairs(subset_df, ggplot2::aes(colour=class$cover_type_class[1:15000]))
print(scatter_matrix)
```




```
# Generating Scatter matrix is incredibly compute-itensive
# task. Use only a subset of data.
```

Overview of Attributes: This dataset aims at predicting the most probable forest cover type arising in regions of $30x30m$ regions using only cartographic variables. Cartographic Variables, also known as $VisualVariables$, refer to aspects of a graphical object that can visually differentiate it from other objects such as the size, orientation, color and so on. This dataset provides a set of variables that are remotely-sensed to train a model to make predictions.

Following are the set of attributes available in the dataset:

Table 1: Overview of Attributes in Dataset.

Name	Data.Type	Measurement.Unit	Description
Elevation	Quantatitive	Meters	Elevation in meters
Aspect	Quantatitive	Azimuth Degrees	Aspect in degrees azimuth
Slope	Quantatitive	Degrees	Slope in degrees
Horizontal Distance to Hydrology	Quantatitive	Meters	Horz Dist to nearest surface water features
Vertical Distance to Hydrology	Quantatitive	Meters	Vert Dist to nearest surface water features
Horizontal Distance to Roadways	Quantatitive	Meters	Horz Dist to nearest roadway
Hillshade 9am	Quantatitive	0 to 255 Index	Hillshade index at 9am, summer solstice

Name	Data.Type	Measurement.Unit	Description
Hillshade Noon	Quantatitive	0 to 255 Index	Hillshade index at noon, summer soltice
Hillshade 3pm	Quantatitive	0 to 255 Index	Hillshade index at 3pm, summer solstice
Horizontal Distance to Fire Points	Quantatitive	Meters	Horz Dist to nearest wildfire ignition points
Wilderness Area	Qualitative	0 (absence) or 1 (presence)	Wilderness area designation
Soil Type	Qualitative	0 (absence) or 1 (presence)	Soil Type designation
Forest Cover Type	Integer	1 to 7	Forest Cover Type designation

Target Variable Classes: The task of predicting the forest cover type using cartographic variables that have been remotely measured is pivotal in modeling, using traditional Machine Learning and newer Deep-Learning approaches, to predict the type of forest that would emerge had there been no human interference at the site using only remotely-sensed cartographic values.

The following are the sets of regions from where the measurements were made:

Table 2: Wilderness Area Code Designations.

Index	Area
1	Rawah Wilderness Area
2	Neota Wilderness Area
3	Comanche Peak Wilderness Area
4	Cache La Poudre Wilderness Area

Approaches Used:

- **KNN - K Nearest Algorithm for Classification**

k-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor.

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor.The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point. A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric (or Hamming distance).

In our experiments, we tested a number of values for the parameter **k**. Smaller k leaves the model vulnerable to noise in data and might not allow it to generalize well. Larger values of k can lead to inconsistencies in predictions. From our experiments, we found the value of k being 31 led to highest accuracy in predictions.

Due to KNN being a computationally intensive algorithm with complexity being exponentially proportional to the number of samples n and the dimensionality of feature vector $|V|$, a maximum of 5000 samples from each target class were selected. Although some classes such as $Cottonwood/Willowr$ have extremely low number of recorded samples.

```
# Create new dataset
new_df <- rbind(dataset_df[dataset_df$Cover_Type == 1,][-(5001:nrow(dataset_df)),])

for (cov_type_idx in seq(2,7)){
  new_df <- rbind(
    new_df,
    dataset_df[dataset_df$Cover_Type == cov_type_idx,][-(5001:nrow(dataset_df)),]
  )
}

# Number of samples in new dataframe
nrow(new_df)
```

```
## [1] 32747
```

```
# Get the idea of sample types
table(new_df$Cover_Type)
```

```
##
##    1    2    3    4    5    6    7
## 5000 5000 5000 2747 5000 5000 5000
```

```
# Preparing train and test set of data
set.seed(123) # To make it reproducible

train_size = 0.7
test_size = 0.3

train_set_size = round(nrow(new_df) * train_size)
train_idxs <- sample(seq_len(nrow(new_df)), size = train_set_size)
```

Normalizing across all dimensions in feature vector in extremely necessary to avoid features with large range of values from overwhelming other features with a relatively small range and thus influencing the result. One such pair could have been *Elevation* and *Aspect*.

```
# Preparing data for KNN algorithm

# 1. Normalize the data

# Define a normalizing function

normalize_knn <- function(x) {
  numer <- x - min(x)
  denom <- max(x) - min(x)
  return(numer / denom)
}

# Apply the normalization:
```

```r
new_df[ , 1:10] = lapply(new_df[, 1:10], normalize_knn)

# Check
glimpse(new_df)
```

```
## Rows: 32,747
## Columns: 55
## $ Elevation                           <dbl> 0.4221106, 0.4422111, 0.4206030, 0.~
## $ Aspect                              <dbl> 0.96388889, 0.89722222, 0.20000000,~
## $ Slope                               <dbl> 0.04918033, 0.40983607, 0.03278689,~
## $ Horizontal_Distance_To_Hydrology    <dbl> 0.00000000, 0.06329114, 0.02233805,~
## $ Vertical_Distance_To_Hydrology      <dbl> 0.2085714, 0.2700000, 0.2085714, 0.~
## $ Horizontal_Distance_To_Roadways     <dbl> 0.3042090, 0.4525399, 0.4747460, 0.~
## $ Hillshade_9am                       <dbl> 0.8385827, 0.5866142, 0.8740157, 0.~
## $ Hillshade_Noon                      <dbl> 0.8780488, 0.7012195, 0.8780488, 0.~
## $ Hillshade_3pm                       <dbl> 0.6411290, 0.7741935, 0.6008065, 0.~
## $ Horizontal_Distance_To_Fire_Points  <dbl> 0.9553883, 0.8670013, 0.8463683, 0.~
## $ Wilderness_Area1                    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
## $ Wilderness_Area2                    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Wilderness_Area3                    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Wilderness_Area4                    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type1                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type2                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type3                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type4                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type5                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type6                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type7                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type8                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type9                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type10                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type11                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type12                         <int> 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,~
## $ Soil_Type13                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type14                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type15                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type16                         <int> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type17                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type18                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type19                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type20                         <int> 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,~
## $ Soil_Type21                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type22                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type23                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type24                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type25                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type26                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type27                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type28                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type29                         <int> 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,~
## $ Soil_Type30                         <int> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type31                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type32                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type33                         <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
```

```
## $ Soil_Type34                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type35                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type36                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type37                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type38                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type39                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Soil_Type40                          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
## $ Cover_Type                           <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
```

```
# Convert label to factor type
new_df$Cover_Type = as.factor(new_df$Cover_Type)

# Apply the KNN algorithm

# Create the sets for training and testing
train_set = new_df[train_idxs, ]
test_set = new_df[-train_idxs, ]

# See the number of examples in both
print(nrow(train_set)) #406708
```

```
## [1] 22923
```

```
print(nrow(test_set)) # 174304
```

```
## [1] 9824
```

Now we apply the KNN algorithm with the value of reference nearest sample points parameter, that is **k** being set to **31**.

```
# Specifying the k values

test_preds_knn = knn(train = train_set[,-55],
                     test = test_set[,-55],
                     cl = train_set[, 55],
                     k = 31)
```

Now viewing the ground truth label from the test set as well as the predicted result after training on a maximum of $5,000$ samples from each forest cover type and calcuating the accuracy of predictions made.

```
# Prediction matrix
pred_matrix_knn = table(true_labels = test_set[,55], predictions =  test_preds_knn)
print(pred_matrix_knn)
```

```
##            predictions
## true_labels    1    2    3    4    5    6    7
##           1 1088  197    2    0   87   21  114
##           2  230  993   22    1  189   64   26
##           3    0    2 1173  102   22  242    0
##           4    0    0   33  714    0   43    0
##           5   12   67   32    0 1326   54    1
##           6    1    1  179   52   22 1202    0
##           7   35    8    2    0    6    2 1457
```

```
# Accuracy
accuracy_knn = sum(diag(pred_matrix_knn)) / sum(pred_matrix_knn)

cat("The accuracy of KNN algorithm is", accuracy_knn)
```

```
## The accuracy of KNN algorithm is 0.809548
```

- **Support Vector Machines**

In machine learning, **support-vector machines (SVMs, also support-vector networks)** are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. Developed at AT&T Bell Laboratories by Vladimir Vapnik with colleagues (Boser et al., 1992, Guyon et al., 1993, Vapnik et al., 1997), SVMs are one of the most robust prediction methods, being based on statistical learning frameworks or VC theory proposed by Vapnik and Chervonenkis (1974) and Vapnik (1982, 1995). Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

Classifying data is a common task in machine learning. Suppose some given data points each belong to one of two classes, and the goal is to decide which class a new data point will be in. In the case of support-vector machines, a data point is viewed as a $p-dimensionalvector$ (a list of p numbers), and we want to know whether we can separate such points with a $(p-1)$-dimensional hyperplane. This is called a linear classifier. There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the maximum-margin hyperplane and the linear classifier it defines is known as a maximum-margin classifier; or equivalently, the perceptron of optimal stability.

In our experiments, we tried various types of kernel functions **K** and their parameters such as *gamma* and $misclassification\ cost\ regularizer$. We finally chose the *radial* kernel function with *gamma* set to **10** and *cost regularizer* set to **100**. In this case of multi-class classification using traditional binary classififer such as SVM, a technique called **one-to-all** was used to train multiple SVM classifiers for each forest cover class type. Hence, we have multiple support vectors pertaining to different models based on the class that we are currently trying to classify.

```
# Try a single parameter SVM
svmfit = svm(Cover_Type~., data = train_set, kernel = "radial",
               scale = FALSE,
               cost = 100,
               gamma = 10)
svm_preds = predict(svmfit, newdata = test_set[, 1:54])

# Prediction matrix
pred_matrix_svm = table(true_labels = test_set[,55], predictions =  svm_preds)
print(pred_matrix_svm)
```

```
##             predictions
## true_labels    1    2    3    4    5    6    7
##           1 1288  145    1    0   23    4   48
##           2  164 1211   14    0  102   24   10
##           3    3    8 1346   38   10  136    0
##           4    0    0   22  748    0   20    0
##           5    3   33    7    0 1447    2    0
##           6    1   13   73   16   11 1343    0
##           7   27    2    0    0    0    0 1481
```

```
# Accuracy
accuracy_svm = sum(diag(pred_matrix_svm)) / sum(pred_matrix_svm)
cat("The accuracy of SVM algorithm is", accuracy_svm)
```

```
## The accuracy of SVM algorithm is 0.9022801
```

- **Neural Networks with Tensorflow**

A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes. Thus a neural network is either a biological neural network, made up of real biological neurons, or an artificial neural network, for solving $artificialintelligence(AI)$ problems. The connections of the biological neuron are modeled as *weights*. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred to as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1.

These artificial networks may be used for predictive modeling, adaptive control and applications where they can be trained via a dataset. Self-learning resulting from experience can occur within networks, which can derive conclusions from a complex and seemingly unrelated set of information.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it. The "*signal*" at a connection is a real number, and the output of each neuron is computed by some **non-linear** function of the sum of its inputs. The connections are called *edges*. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

Neural networks learn (or are trained) by processing examples, each of which contains a known "*input*" and "*result*" forming probability-weighted associations between the two, which are stored within the data structure of the net itself. The training of a neural network from a given example is usually conducted by determining the difference between the processed output of the network (often a prediction) and a target output. This is the error. The network then adjusts its weighted associations according to a learning rule and using this error value. Successive adjustments will cause the neural network to produce output which is increasingly similar to the target output. After a sufficient number of these adjustments the training can be terminated based upon certain criteria. This is known as **supervised learning**.

Due to large number of soil types, we decided to encode various attributes of soil by using an **Embedding Layer** and reducing the dimensionality of *all* the soil types to about **16**. This has a positive effect on model because of reduced strain of modelling large number of sparse vectors. The dimensionality reduction of such Embeddings in in $2D$ or $3D$ space using algorithms such as **t-SNE** and **PCA** we can, with a certain level of error, visualize the similarity between different soil types.

Further, we created a dynamic framework to allow for a variable number of '*Dense*' layers along with varying number of units in each layer. **ReLu** activation function was used to impose non-linearity so as to make the network more expressive and increase its ability to form complex functional mappings from input to output. This is because, a multi-layer linearly activated network is still, mathematically, a linearly separating network which is limited in its expressive power.

The output layer consists of seven units because of seven output classes with $softmax$ activation to output a distribution of probabilities over the target classes instead of just activations. Further, an $argmax$ operation was performed over these probabilities to get the predicted forest cover class type.

The network can be changed dynamically just by changing the network parameters and then re-generating a new model and training on it.

```
set.seed(123) # To make it reproducible

train_size = 0.7
test_size = 0.3

train_set_size = round(nrow(dataset_df) * train_size)
train_idxs <- sample(seq_len(nrow(dataset_df)), size = train_set_size)

# Create the sets for training and testing
train_set_nn = dataset_df[train_idxs, ]
test_set_nn = dataset_df[-train_idxs, ]

# See the number of examples in both
print(nrow(train_set_nn)) # 406708
```

```
## [1] 406708
```

```
print(nrow(test_set_nn)) # 174304
```

```
## [1] 174304
```

```
# Soil Integer Encoding Dataframe
soil_df = data.frame("Soil_Type" = rep(0, nrow(dataset_df)))
count = 1
for (soil_type_idx in seq(15,54)){
  soil_df$Soil_Type[dataset_df[, soil_type_idx] == 1] = count
  count = count + 1
}

# Get the count of each soil type
table(soil_df$Soil_Type)
```

```
##
##     1     2     3     4     5     6     7     8     9    10    11
##  3031  7525  4823 12396  1597  6575   105   179  1147 32634 12410
##    12    13    14    15    16    17    18    19    20    21    22
## 29971 17431   599     3  2845  3422  1899  4021  9259   838 33373
##    23    24    25    26    27    28    29    30    31    32    33
## 57752 21278   474  2589  1086   946 115247 30170 25666 52519 45154
##    34    35    36    37    38    39    40
##  1611  1891   119   298 15573 13806  8750
```

Further we specify the data to be input into the network:

```
# Create a matrix of data
mat_x_other = as.matrix(train_set_nn[1:14])
mat_x_other_test = as.matrix(test_set_nn[1:14])

soil_x = array(soil_df$Soil_Type[train_idxs], dim = c(nrow(train_set_nn), 1))
soil_x_test = array(soil_df$Soil_Type[-train_idxs], dim = c(nrow(test_set_nn), 1))

y_train = array(dataset_df$Cover_Type[train_idxs] - 1, dim = c(nrow(train_set_nn), 1))
y_test = array(dataset_df$Cover_Type[-train_idxs] - 1, dim = c(nrow(test_set_nn), 1))

# Verify
```

```
nrow(mat_x_other)
```

```
## [1] 406708
```

```
nrow(mat_x_other_test)
```

```
## [1] 174304
```

```
length(soil_x)
```

```
## [1] 406708
```

```
length(soil_x_test)
```

```
## [1] 174304
```

```
length(y_train)
```

```
## [1] 406708
```

```
length(y_test)
```

```
## [1] 174304
```

Now, we specify all the parameters and other Hyper-Parameters to the Neural Network. Any required change in the network can be done via changing these specified parameters.

```
############### PARAM definition ###########################
NUM_SOIL_TYPES = 40
TOTAL_NUM_PARAM_EXCEPT_SOIL = 11
NUM_WILDERNESS_AREAS = 4
INTEGER_ENCODED_WILDERNESS_AREA = FALSE # set `FALSE` for OHE of Wilderness Area
CHECKPOINT_PATH = ".\\model_checkpoints"
MODEL_CHECKPOINT_FREQUENCY = 'epoch'
TENSORBOARD_LOG_DIR = ".\\tensorboard_log_dir"
EMBEDDING_SAVE_FREQ = 1
############################################################

############## Define HPARAMs ############################
BATCH_SIZE = 128
EPOCHS = 120
VERBOSE = 2
VALIDATION_SPLIT = 0.2
SHUFFLE_DATA = TRUE
SOIL_EMBEDDING_SIZE = 16
INIT_DENSE_LAYER_UNITS = 150
NUM_DENSE_LAYERS = 5 # First Dense Layer is added by default. (No Softmax)
USE_SAME_DENSE_UNITS = FALSE
NUM_DENSE_UNITS = 100
ALL_LAYER_DENSE_UNITS = c(100L, 80L, 70L, 65L, 50L)
NUM_FOREST_COV_TYPE_CLASSES = 7
LEARNING_RATE = 0.0001
DROPOUT_UNIT_RATE = 0.1 # To perform regularization
ACTIVATION_ALL_LAYERS = 'relu'
############################################################
```

Defining the Neural Network now:

```
create_forest_cov_type_model <- function(SOIL_EMBEDDING_SIZE,
                                INIT_DENSE_LAYER_UNITS,
                                NUM_DENSE_LAYERS,
                                USE_SAME_DENSE_UNITS,
                                NUM_DENSE_UNITS,
                                ALL_LAYER_DENSE_UNITS,
                                NUM_FOREST_COV_TYPE_CLASSES,
                                ACTIVATION_ALL_LAYERS,
                                NUM_SOIL_TYPES,
                                TOTAL_NUM_PARAM_EXCEPT_SOIL,
                                NUM_WILDERNESS_AREAS,
                                INTEGER_ENCODED_WILDERNESS_AREA){
  # Creating the model

  soil_type_ip <- layer_input(shape = c(1), dtype = "int32", name = "soil_type_ip")
  soil_emb_op <- soil_type_ip %>%
    layer_embedding(input_dim = (NUM_SOIL_TYPES + 1),
                    output_dim = SOIL_EMBEDDING_SIZE,
                    name = "soil_emb") %>%
    layer_reshape(target_shape = list(SOIL_EMBEDDING_SIZE))

  if(INTEGER_ENCODED_WILDERNESS_AREA){
    other_ip_layer <- layer_input(shape = c(TOTAL_NUM_PARAM_EXCEPT_SOIL),
                                  name = "other_ip")
  }else{
    other_ip_layer <- layer_input(shape = c(TOTAL_NUM_PARAM_EXCEPT_SOIL +
                                            NUM_WILDERNESS_AREAS - 1),
                                  name = "other_ip")
  }

  # Dense Layers

  dense_layer_op <- layer_concatenate(list(other_ip_layer, soil_emb_op)) %>%
    layer_dense(units = INIT_DENSE_LAYER_UNITS,
                activation = ACTIVATION_ALL_LAYERS, name = "init_dense")

  for (dense_idx in seq(1, NUM_DENSE_LAYERS)){
    # Some validation
    if(USE_SAME_DENSE_UNITS){
      dense_layer_op = dense_layer_op %>%
        layer_dense(units = NUM_DENSE_UNITS,
                    activation = ACTIVATION_ALL_LAYERS)
    }
    else{
      if(length(ALL_LAYER_DENSE_UNITS) != NUM_DENSE_LAYERS){
        stop("Require same number of dense units and dense layers.")
      }else if (class(ALL_LAYER_DENSE_UNITS) != "integer"){
        stop("Require Integer vector for specifying dense layer units.")
      }else{
        dense_layer_op = dense_layer_op %>%
          layer_dense(units = ALL_LAYER_DENSE_UNITS[dense_idx],
                      activation = ACTIVATION_ALL_LAYERS) %>%
          layer_dropout(rate = DROPOUT_UNIT_RATE)
```

```
        # Dropout will have a regularizing effect.
      }
    }
  }


  # Final Softmax Layer

  model_op_prob_dis = dense_layer_op %>%
    layer_dense(units = NUM_FOREST_COV_TYPE_CLASSES,
                activation = "softmax")


  # Define the model here
  model <- keras_model(
    inputs = c(soil_type_ip, other_ip_layer),
    outputs = c(model_op_prob_dis)
  )

  summary(model)
  return(model)
}
```

Creating an instance of the specified neural network:

```
model_forest = create_forest_cov_type_model(SOIL_EMBEDDING_SIZE,
                                INIT_DENSE_LAYER_UNITS,
                                NUM_DENSE_LAYERS,
                                USE_SAME_DENSE_UNITS,
                                NUM_DENSE_UNITS,
                                ALL_LAYER_DENSE_UNITS,
                                NUM_FOREST_COV_TYPE_CLASSES,
                                ACTIVATION_ALL_LAYERS,
                                NUM_SOIL_TYPES,
                                TOTAL_NUM_PARAM_EXCEPT_SOIL,
                                NUM_WILDERNESS_AREAS,
                                INTEGER_ENCODED_WILDERNESS_AREA)
```

```
## Model: "functional_1"
## ____________________________________________________________________________
## Layer (type)              Output Shape      Param #  Connected to
## ============================================================================
## soil_type_ip (InputLayer) [(None, 1)]       0
## ____________________________________________________________________________
## soil_emb (Embedding)      (None, 1, 16)     656      soil_type_ip[0][0]
## ____________________________________________________________________________
## other_ip (InputLayer)     [(None, 14)]      0
## ____________________________________________________________________________
## reshape (Reshape)         (None, 16)        0        soil_emb[0][0]
## ____________________________________________________________________________
## concatenate (Concatenate) (None, 30)        0        other_ip[0][0]
##                                                      reshape[0][0]
## ____________________________________________________________________________
## init_dense (Dense)        (None, 150)       4650     concatenate[0][0]
## ____________________________________________________________________________
```

```
## dense (Dense)              (None, 100)       15100     init_dense[0][0]
## ____________________________________________________________________________
## dropout (Dropout)          (None, 100)       0         dense[0][0]
## ____________________________________________________________________________
## dense_1 (Dense)            (None, 80)        8080      dropout[0][0]
## ____________________________________________________________________________
## dropout_1 (Dropout)        (None, 80)        0         dense_1[0][0]
## ____________________________________________________________________________
## dense_2 (Dense)            (None, 70)        5670      dropout_1[0][0]
## ____________________________________________________________________________
## dropout_2 (Dropout)        (None, 70)        0         dense_2[0][0]
## ____________________________________________________________________________
## dense_3 (Dense)            (None, 65)        4615      dropout_2[0][0]
## ____________________________________________________________________________
## dropout_3 (Dropout)        (None, 65)        0         dense_3[0][0]
## ____________________________________________________________________________
## dense_4 (Dense)            (None, 50)        3300      dropout_3[0][0]
## ____________________________________________________________________________
## dropout_4 (Dropout)        (None, 50)        0         dense_4[0][0]
## ____________________________________________________________________________
## dense_5 (Dense)            (None, 7)         357       dropout_4[0][0]
## ============================================================================
## Total params: 42,428
## Trainable params: 42,428
## Non-trainable params: 0
## ____________________________________________________________________________
```

We have also provided various callbacks to visualize the metrics using *Tensorboard* and save the model at regular checkpoints using *ModelCheckpoint* callback.

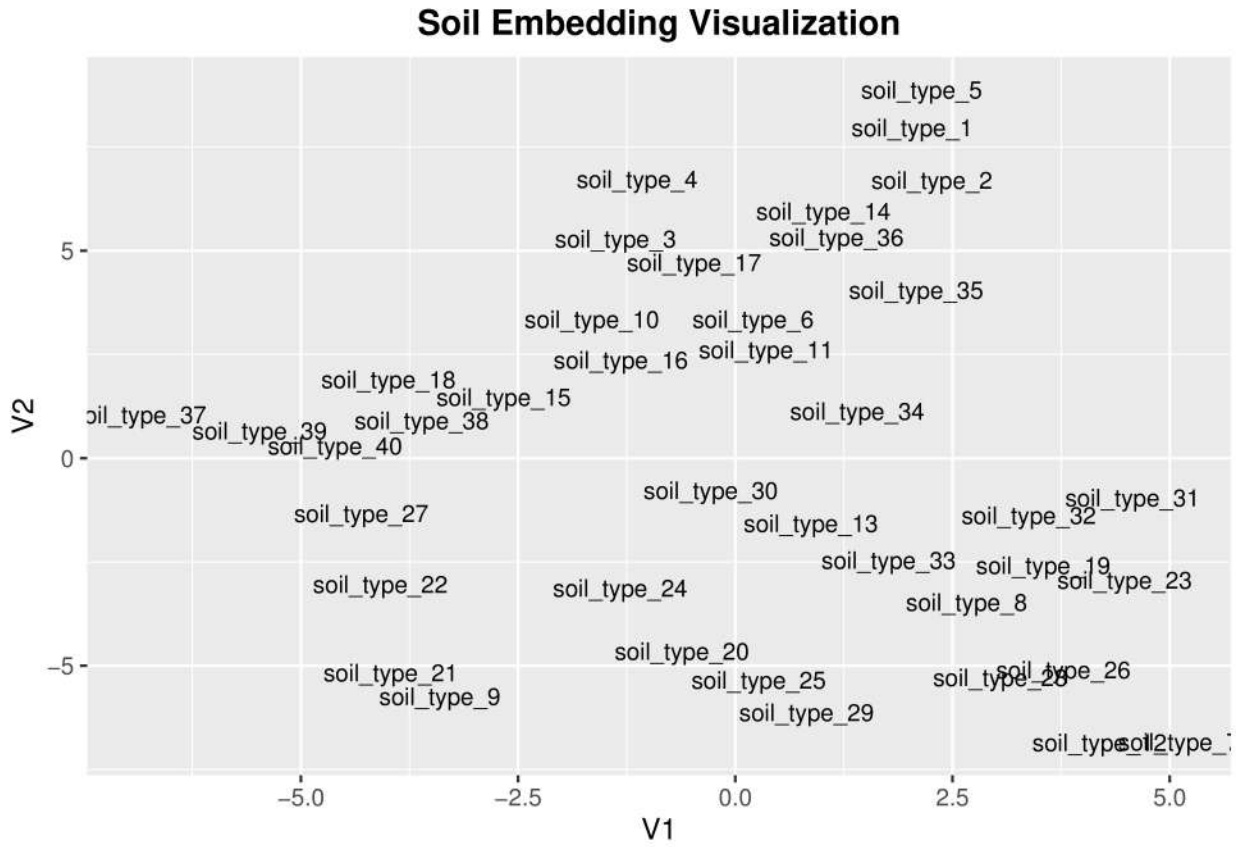
```
# Create callbacks to visualize model metrics
# and save the checkpointed model.
model_checkpoint_cb = callback_model_checkpoint(filepath = CHECKPOINT_PATH,
                                                save_freq = MODEL_CHECKPOINT_FREQUENCY)

# Create callback for Tensorboard plugin
model_tensorboard_cb = callback_tensorboard(log_dir = TENSORBOARD_LOG_DIR,
                                            embeddings_freq = EMBEDDING_SAVE_FREQ,
                                            update_freq = 'epoch',
                                            embeddings_layer_names = list("soil_emb"))
```

Now compiling the model with *ADAM* optimizer with learning rate set to 0.001.

```
model_forest %>% compile(
  optimizer = optimizer_adam(lr = LEARNING_RATE),
  loss = 'sparse_categorical_crossentropy'
)

model_forest %>% fit(x = list(soil_x, mat_x_other),
                     y = y_train,
                     epochs = EPOCHS,
                     callbacks = list(model_tensorboard_cb, model_checkpoint_cb),
                     validation_split = 0.10,
                     shuffle = SHUFFLE_DATA)
```

Creating Soil Type Embedding Visulization:

```r
generate_embedding_plot <- function(model){
  # Get the embedding weights
  embedding_matrix <- get_weights(model)[[1]]

  # Define the embedding metadata for each soil type
  soil_type_vec = c("soil_type_1")
  for (idx in 2:40){
    soil_type_vec = c(soil_type_vec,
                      paste("soil_type_", as.character(idx), sep = ""))
  }

  # Specify the names for the soil types
  row.names(embedding_matrix) <- c("UNK", soil_type_vec)

  # Perform Dimensionality Reduction from 16D to 2D
  soil_tsne = Rtsne(embedding_matrix[-1,], perplexity = 10, pca = FALSE)

  # Add new column in dataframe and plot it
  tsne_plot <- soil_tsne$Y %>%
    as.data.frame() %>%
    mutate(soil_type_name = row.names(embedding_matrix)[-1]) %>%
    ggplot(aes(x = V1, y = V2, label = soil_type_name)) +
    ggtitle("Soil Embedding Visualization") +
    theme(plot.title = element_text(color = "#000000", size = 13, hjust = 0.5,
                                    face = "bold")) +
    geom_text(size = 3)


  # Return the plot object
  return(tsne_plot)
}

# Embedding Visualization for different Soil Types
generate_embedding_plot(model_forest)
```



Now generating predictions and calculating accuracy:

```
# Predictions
preds_nn = model_forest %>% predict(x = list(soil_x_test,
                            mat_x_other_test))
pred_labels_nn = argmax(preds_nn, rows = TRUE)

pred_matrix_nn = table(true = as.factor(y_test + 1), predictions = as.factor(pred_labels_nn))
print(pred_matrix_nn)
```

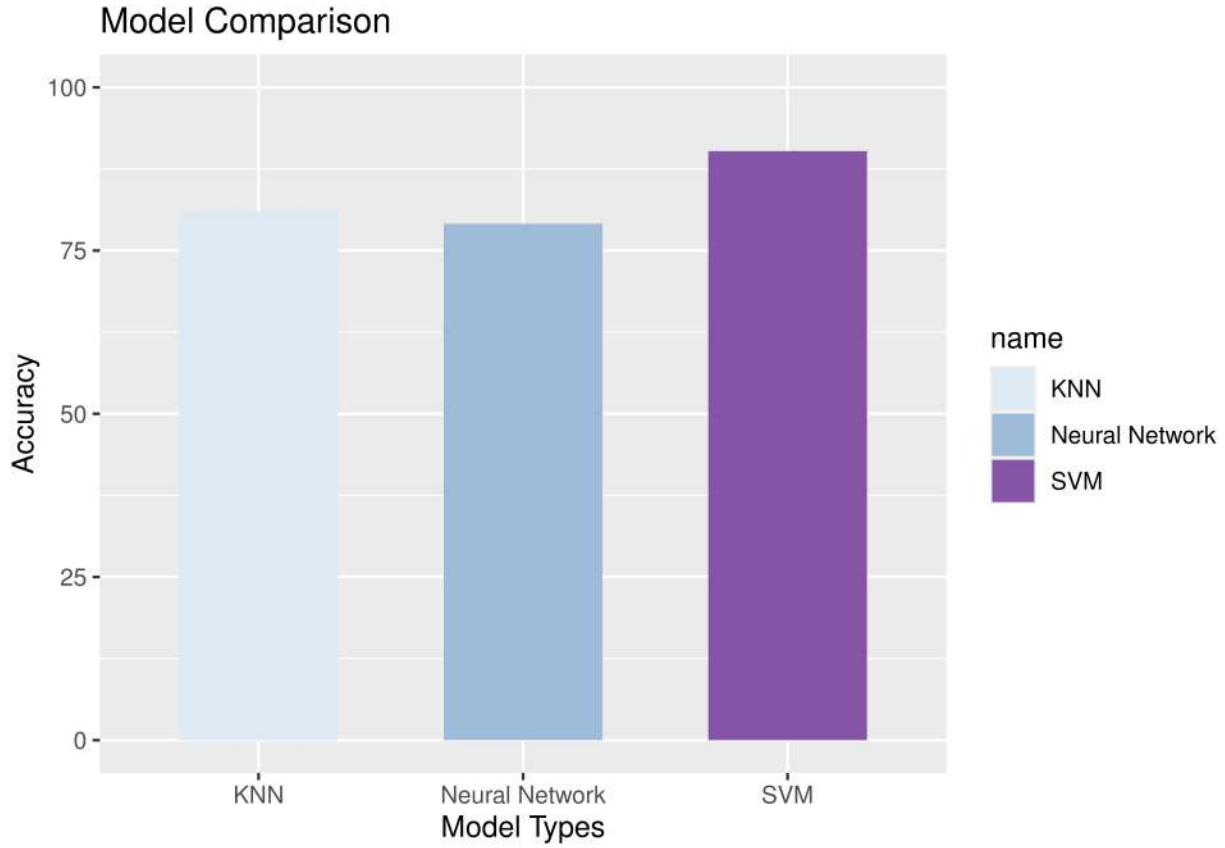
```
##     predictions
## true     1     2     3     4     5     6     7
##    1 40605 22812     0     0    17    17   127
##    2  3244 80868   221     0   130   520     9
##    3     0  1262  7893    21     0  1596     0
##    4     0     2   267   445     0   115     0
##    5     3  2072    43     0   715    13     0
##    6     0   773   652     1     0  3720     0
##    7  2353   151     0     0     2     0  3635
```

```
# Accuracy
accuracy_nn = sum(diag(pred_matrix_nn)) / sum(pred_matrix_nn)
cat("The accuracy of SVM algorithm is", accuracy_nn)
```

```
## The accuracy of SVM algorithm is 0.7910375
```

Visualizing the Results

```
accuracy_df = data.frame(
  name = c("KNN", "SVM", "Neural Network"),
  values = c(accuracy_knn * 100, accuracy_svm * 100, accuracy_nn * 100)
)

acc_plot = accuracy_df %>%
  ggplot(aes(x = name, y = values, fill = name)) +
  geom_bar(stat = "identity", width = 0.60) +
  scale_fill_brewer(palette="BuPu") +
  xlab("Model Types") +
  ylab("Accuracy") +
  ggtitle("Model Comparison") +
  ylim(0, 100)

print(acc_plot)
```



Conclusion

kNN and SVM algorithms, although not being trained on the entire data perform remarkably well. Neural network however, also performs well after being trained on the entire data. Correct approach for classifying the forest cover type would involve either SVM or Neural Networks since KNN requires to calculate the Euclidean Distance everytime while making a prediction with each sample in training set. SVM also requires storing the Kernel Function Dot Product Output for each pair of samples, and hence it is slightly more scalable than KNN but still cannot be used for training on huge amount of data. Neural Networks reach

the global optimum at around 70 epochs and can be used to classify even skewed data unlike SVM and KNN which were trained on non-skewed data and have high memory requirements. The similarity between different soil types can also be seen using the embeddings learned by neural network and closest types can be calculated using *cosinesimilarity*.