National College of Ireland

Configuration Manual

MSc Research Project
Msc. Data Analytics in Data Science

Oyinkansola Shittu
Student ID: X20147406

School of Computing
National College of Ireland

Supervisor: Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Oyinkansola Shittu

Student ID: X2014740

Programme: Msc. Data Analytics in Data Science **Year:** 2023

Module: Research Project

Lecturer: Dr. Catherine Mulwa

Submission Due Date: August 14th 2023

Project Title: Soil Degradation Prediction and Classification using Digital Soil Maps: Boosting Nigerian Food Security

Word Count: ..... **Page Count:** 22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Oyinkansola shittu

Date: 14/08/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Oyinkansola Shittu
Student ID: X20147406

1 Introduction

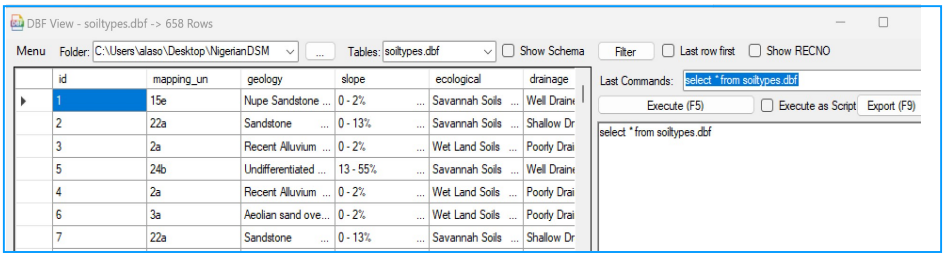
This project is split into three (4) broad categories:
Data acquisition and packages installation, data preprocessing, data modeling and feature interaction detection.
N.B: knowledge of R algorithm and SQL are pre-requisites

1.1 Data acquisition and package installation:

Nigerian digital soil map dataset freely downloaded from science direct website

1.2 Download an online database management package, DBF viewer was used for this project. This dbf viewer is required to access the downloaded dataset and a fair knowledge of SQL is required to generate report. This random report will be compared with same random report generated in Jupyter note book (R-Kernel) for data integrity check

1.3 This dataset is stored in a schema and a database management tool will be needed for access and will be referred to as NDSM hence forth**.** It contains five geospatial files (.shp, .shx, .sbx, .sbn, and .prj), choose the file ending with '.prj'. Once Dbf is connected you will see a screen like below.



- Select the directory the downloaded dataset from the 'folder' window.
- Choose the file from the 'tables' window.
- Click on 'schema—optional.
- Input the SQL -command to execute a query. As an example to get a report of all observations and features, write SELECT* from …..(input the file name).
- Click 'F5' to execute.
- Export (F9)the generated report as a .csv file and save on your local directory as 'soiltypesdbf_project.

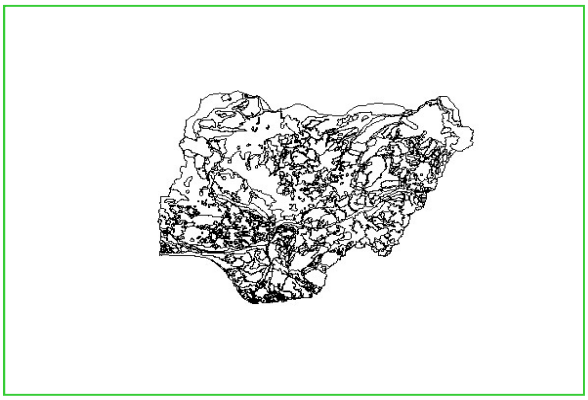
1.4 Install Jupyter notebook, anaconda or python and R

1.5 Extraction of the .shx and the .shp, files:

- R algorithms are used to extract these files in Rstudio
- The required packages and libraries are installed and called.

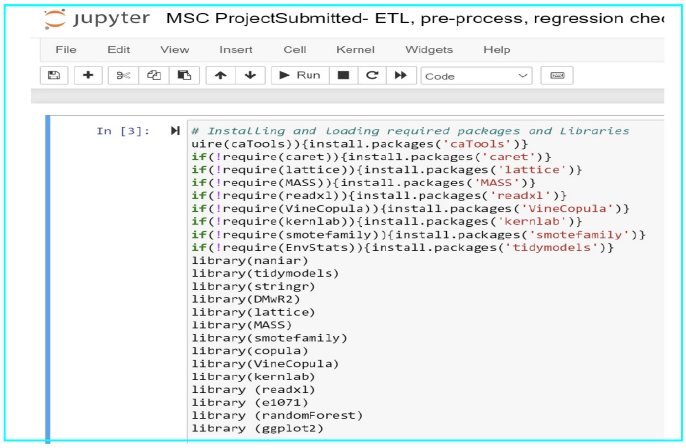
```
## Installing and calling required libraries
install.packages('foreign')
install.packages('sp')
install.packages('sf')
install.packages('rgdal')
install.packages('writexl')
library(foreign)
library(shapefiles)
library(raster)
library(foreign)
library(sp)
library(sf)
library(rgdal)
library(writexl)
```

- Load both the .shx and .shp files into Rstudio with st_read() and save as ‘soilSearch’ and ‘soilShape’ respectively. It shows 658 rows and 17 coulmns.
- Plot the files to convert the multipolygon to map. Both maps are the same with no spatial information (ie no location coordinates) so not useful for modeling. See figure below.



- Write both files onto local directory as .csv files

1.6 Install R packages needed for the loading and pre-processes in jupyter notebook. Below is a sample of command to install the packages and call the associated libraries.



2 Data Cleaning, Transformation and Pre-processing.

Here the data is checked for missing data, cleaning, understanding of the data in general.

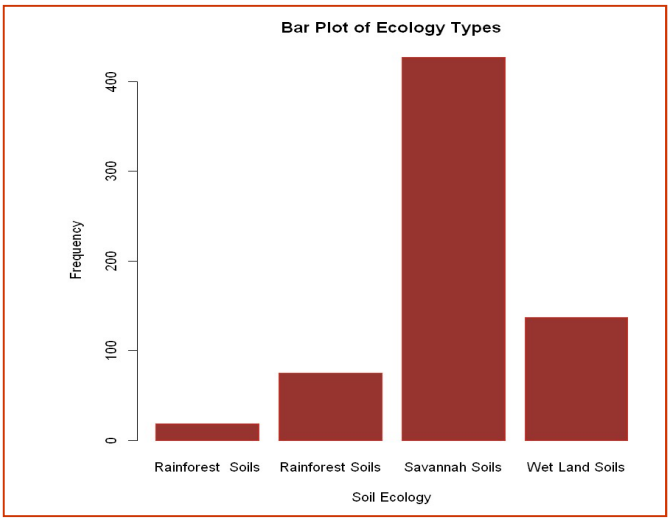
- Note only the .prj file is used for the project as it had the required data . Both the extracted .shp and .shx files are not going to be used since they have same data as the .prj files but no location coordinates.
- The ,sbn and .sbx files are the raster files for folder maintenance like index search speed not required in Jupyter notebook, so not used either.

2.1 Load the Dbf exported NDSM file (soiltypes.dbf_project) into Jupyter note book – R kernel, using the R command read.csv(), input 'string Asfactor =TRUE'

2.2 Get familiar with the data—summary, missing values, data types etc.

2.3 Check for data integrity: compare randomly generated query report in dbfviewer to same query report generated in jupyter notebook ensuring same output for data upload accuracy.

2.4 Visually explore the dataset to check for missing flaws for correction. Note variable 'ECOLOGICAL' needs to be corrected for the two mis-spaced label names 'Rainforest soils', as seen below



2.5 Remove irrelevant variables that are either descreet, ordinal or unnecessary ('Id','mapping_un','pH_describ','soil_class' and 'Percentage')

2.6 Transform the factor variables to integers

2.7 Check for regression assumptions

2.8 Use box plots to identify the variables with outliers

2.9 Remove identified outliers. This project used created three functions, first to change variables to numeric, second to detect the outliers and third to create a new dataframe of no-outliers , named soilNum2 and save in local directory.

2.10 Reload the newly created soilNum2 csv

2.11 Replace the existing variables (having outliers) with the new one (without outliers)

2.12 Correct variable non-normality with box-cox, log10, square root and cubic log transformations.

- Since these transformations did not work, it confirmed dataset to be non-normal and non-linear.
- The original dataset is now reloaded as a 'soilstr' and all previous pre-processes and transformation redone.

2.13 Synthesise 5000 more data points using copula package on this re pre-processed dataset. The copula family will enable generation of dependences similar to the original dataset

- To know and select the exact copula type to use (that fits the dataset), first create the Bicop bivariate objects (eCop) with the dataset for all 12 features.

```
set.seed(2)
conflicted::conflicts_prefer(copula::pobs)
eCop1<- pobs(as.matrix(soilstr))[,1]
eCop2<- pobs(as.matrix(soilstr))[,2]
eCop3<- pobs(as.matrix(soilstr))[,3]
eCop4<- pobs(as.matrix(soilstr))[,4]
eCop5<- pobs(as.matrix(soilstr))[,5]
eCop6<- pobs(as.matrix(soilstr))[,6]
eCop7<- pobs(as.matrix(soilstr))[,7]
eCop8<- pobs(as.matrix(soilstr))[,8]
eCop9<- pobs(as.matrix(soilstr))[,9]
eCop10<- pobs(as.matrix(soilstr))[,10]
eCop11<- pobs(as.matrix(soilstr))[,11]
eCop12<- pobs(as.matrix(soilstr))[,12]
```

- Select the created objects using BiCopSelect()function in pairs for Bivariate copula to generate the copula family. For example:

  selectCopula1<-BiCopSelect(eCop1,eCop2,familyset=NA)

- Merge the selectCopulas above using cbind to generate a matrix that gives information of the Family names and numbers, par1 and par2, all to be used later.
- Generate the 5,000 data points using above information with BiCopSim() .
- Bind the output with cbind() and as a dataframe- data.frame and save as 'simDatum'.
- Name the columns with the original dataset column names.
- Rename the columns of the simDatum to be exactly the same with the existing columns. Output should be as below extract.

A data.frame: 5000 × 12

GEOLOGY	SLOPE	ECOLOGICAL	DRAINAGE	SOIL_PH	SUITABILIT	SOIL_TEXTU	VEGETATION	DISTRIBUTI	SOIL_CLA_1	MAJOR_CROP
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
0.83359997	0.88669321	0.481552567	0.73476543	0.32868504	0.58518138	0.79186393	0.8741775760	0.77693160	0.29980782	0.19435906
0.97531416	0.89927560	0.193531110	0.91428219	0.87200081	0.76854053	0.47160293	0.9807316035	0.26971352	0.91201813	0.22341526
0.81227813	0.36381406	0.498285655	0.44654602	0.90462622	0.19820520	0.14539474	0.2404162837	0.28712601	0.70461511	0.36238606
0.51218190	0.83993839	0.936274827	0.29182360	0.12376392	0.20387697	0.90370487	0.7292369895	0.11295633	0.79290303	0.38477180
0.82883141	0.87899382	0.689566562	0.02230024	0.96054971	0.74950519	0.21217660	0.3265760183	0.10616752	0.76758835	0.44613357
0.28206092	0.70568652	0.963808967	0.09762768	0.28634021	0.94961055	0.98728940	0.0007812372	0.89685871	0.14062298	0.50084070
0.76470621	0.28157324	0.109797793	0.95940993	0.82876080	0.98064054	0.05881267	0.1035140059	0.10077906	0.49434106	0.54764168
0.73207440	0.40849911	0.202546175	0.78974652	0.16917898	0.90708829	0.63510158	0.2511310380	0.72638554	0.48900963	0.14045215
0.50837949	0.62980065	0.409267576	0.55550687	0.01410362	0.06515141	0.08033575	0.2695105991	0.30625890	0.90711767	0.94699282
0.60828433	0.35856047	0.116032289	0.27278478	0.21633766	0.98485120	0.06519950	0.1623746179	0.65018147	0.94542471	0.11286180
0.07814843	0.46006182	0.196761012	0.90304130	0.10552160	0.76550654	0.89550985	0.3723854136	0.43925000	0.06563119	0.44776114
0.03743085	0.21667351	0.202009404	0.59070794	0.78970513	0.10448286	0.50456221	0.0405371028	0.78111888	0.88285877	0.53492337
0.26774902	0.62269256	0.026139471	0.97292341	0.59572017	0.37902826	0.50925560	0.1623412461	0.40437063	0.67675572	0.41247659
0.69446482	0.88567271	0.076442906	0.57480236	0.43683717	0.05932444	0.67442786	0.2812166205	0.74721937	0.72183624	0.92240115
0.03980585	0.04832209	0.004568956	0.99793847	0.43795573	0.26871240	0.11552625	0.5853416836	0.78532358	0.93425609	0.04154902
0.06746978	0.01845225	0.772117136	0.22099277	0.09794679	0.91380485	0.37899732	0.2302426445	0.38524092	0.95127419	0.81993182
0.61759676	0.14958265	0.073994502	0.32116998	0.14146399	0.12139026	0.12328594	0.1670310052	0.98660010	0.39205596	0.98135497

2.14 Generate numeric estimates for the character variables of original dataset soilstr (568 observations) based on existing impact of each predictor factor level on Soil_Ph (dependent)

- Transform all variables to numeric
- #### Use the tidymodels package for the formula syntax below ###

- Apply the syntax below on soilstr data and save as 'soilData'. This command generates predictors numeric values equivalent to each character factor level and replacing the traditional dummy variables (0/1) with variance of soil_pH based on impact of each predictor factor-level.

```
soilData<- recipe(formula = SOIL_PH ~ . , data=soilstr) %>%
step_lencode_glm(GEOLOGY,SLOPE,ECOLOGICAL,DRAINAGE,SUITABILIT,
SOIL_TEXTU,VEGETATION, DISTRIBUTI,SOIL_CLA_1,MAJOR_CROP,Depth,
outcome = vars(SOIL_PH))%>%
prep(soilstr) %>%
bake(soilstr%>%select(GEOLOGY,SLOPE,ECOLOGICAL,DRAINAGE,SOIL_PH,
SUITABILIT,SOIL_TEXTU,VEGETATION,DISTRIBUTI,SOIL_CLA_1,MAJOR_CROP,
Depth))
soilData
```

2.15 Change the position of target variable Soil_pH in simDatum to match with the position in soilData (ie positioned after 'Depth' predictor

```
simDatum%>% relocate(SOIL_PH, .after= Depth)
```

2.16 Merge the soilData and the simDatum using rbind() and save as soilNew. This will ensure each row binds and values fitted for each column too.

- This gives the newly generated dataset of 5,658 observations and 12 features

2.17 Features selection using principal component analysis. Follow the steps below:

- Build an lm model

```
model= lm(SOIL_PH~ ., data=soilNew)
summary(model)

# Removing the target variable from the dataset
targetless<- soilNew[,-12]

# Principal Component Analysis (PCA)
targetless.pca<- prcomp(targetless, center=TRUE, scale=TRUE)

# Plotting the scree plot
plot(targetless.pca, type='l', main='Scree Plot of soilNew')
summary(targetless.pca)

# Doing the rotation matrix
print(targetless.pca$rotation)

# To see if variables can be discarded, Eigen values were calculated for the data
without PCA
eigen(cor(targetless))$values
diag(var(targetless.pca$x[,]))
```

```
#checking correlation between the target variable and the PCAs

soilNew.pca=cbind(soilNew[,12],data.frame(targetless.pca$x))
colnames(soilNew.pca)[1]<- 'SOIL_PH'
cor(soilNew.pca)[,1]

# Building a model to check statistical significance of the PCAs.
# All PCs were statistically significant, thus no variable removed.

signifPCA<- lm(SOIL_PH ~ ., data= soilNew.pca)
summary(signifPCA)
```

2.18 Recheck for outliers using the same three- function command and save the dataset of 'no-outliers) as SoilOut2 in local directory.

2.19 Treatment of data imbalances using Synthetic Minority Oversampling Technique.

- The steps below should be followed to apply the SMOTE() function used in correcting the data imbalances.

```
##To avoid clog and k being more than sample size, soilOut2 was split into
training(70%) and testing (30%) data using dependent variable (soil_ph) and the test
data used for imbalance correction. The test data size is 2,997 which is sufficient for
modeling.

set.seed(200)

#       Splitting dataset into training and testing samples

split3<-sample.split(soilOut2$SOIL_PH,SplitRatio = 0.7)# random selection
soil_training3 <- subset(soilOut2, split3== TRUE)
soil_test3 <- subset(soilOut2, split3==FALSE)

#       Applying the SMOTE function to the test data

soilSmote2<-soil_test3
testSmote2<- SMOTE(soilSmote2,soilSmote2[['SOIL_PH']]) # The ...$data is 2,997
observations and 13 features
tsetSmote$data

## Saving smoted-data as testSmote2$data and writing the data into local directory
## This dataset is loaded into R-studio for modeling implementation stage below

write.csv(testSmote2$data,'C:/Users/alaso/Desktop/testSmote2$data.csv',
row.names=FALSE)
```

3 Implementation

3.1 Prediction Models

3.1.1 Support Vector Machine for Regression

- Load the required packages

```
## Support Vector Machine for Regression Model ######

# Load required packages and libraries

if(!require(MASS)){install.packages('MASS')}
if(!require(e1071)){install.packages('e1071')}
if(!require(caret)){install.packages('caret')}
if(!require(lattice)){install.packages('lattice')}
if(!require(ggplot2)){install.packages('ggplot2')}
if(!require(kernlab)){install.packages('kernlab')}
library(MASS)
library(e1071)
library(caret)
library(lattice)
library(ggplot2)
library(kernlab)
```

- Read-in the soilSmote2$data as 'testSmoteSVR'
- Check which SVR kernel performs better between 'svmRadial' and 'svmPoly' based on lower RMSE value to select radial type to use:

```
## Using the svm-radial kernel##
set.seed(3)
model<- train(SOIL_PH~., data= testSmoteSVR, method='svmRadial')
model # smallest RMSE=0.05

## Using the polynomial kernel##
set.seed(2)
model1<- train(SOIL_PH~., data= testSmoteSVR, method='svmPoly')
model1    # smallest RMSE=0.01
```

- Polynomial kernel was chosen for the lower RMSE and would have been chosen if otherwise for the interaction effect advantage over radial kernel since check of interaction effect is one of the project's contribution.

- Split dataset into training testing and validate in ration 70:20:10 respectively.
- Set hyper-parameters to be used for model tuning:
- Set seed for reproducibility
    - Control = 10-fold cross validation
    - TuneGrid:

        Degree = 1,2,3, Scale = 0.01, 0.1, C = 0.25, 0.5,1

- **Modeling**
- Use the result of set parameters in above step for the modeling as below:

```
## Training with training data
set.seed(3)
model2a<-train(SOIL_PH~., data= svrTrain, method='svmPoly',
               trControl=contr,tuneGrid=tuneGrid)
model2a
```

- **Prediction on test and validate data**
- Follow the steps below;

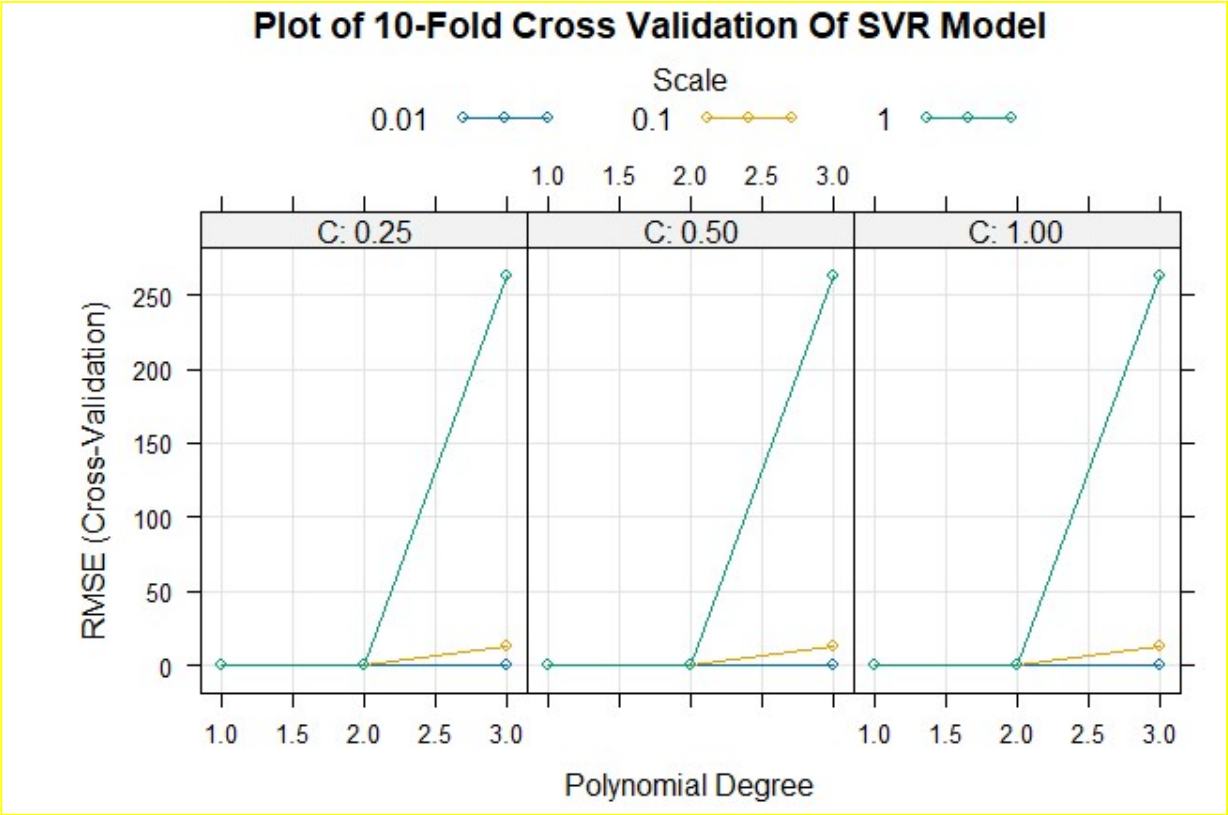
  svrTest= Test data and svrValid = validate data

```
### Predictions with test and validation data ###

## Subseting test and validation data

set.seed(1)
features.test= subset(svrTest, select= -c(SOIL_PH))
target.test = subset(svrTest, select= SOIL_PH)[,1]

set.seed(2)
features.Valid= subset(svrValid, select= -c(SOIL_PH))
target.Valid = subset(svrValid, select= SOIL_PH)[,1]

## Now for predictions ##
#Test prediction
set.seed(1)
predSVR = predict(model2a,newdata=features.test)

## validation prediction
set.seed(1)
predSVR2 = predict(model2a,newdata=features.Valid)
```

- **Evaluation of results**
- Use in-built R-commands for the evaluation metrics as below for test data:

```
#####    SVR Evaluation and Results   #####

## Evaluating Model2a test results ##

#RMSE:
sqrt(mean((target.test-predSVR)^2))        # [1] 0.0109

#R2:
cor(target.test,predSVR)^2                 # [1] 0.9999

#MAE:
MAE(target.test,predSVR)                   # [1] 0.0097
```

- Repeat same for valid date, replacing the ‘target.test’ for ‘target.valid’
- The 10-fold cross validation should look like this



3.1.2 Random Forest for Regression

- Load required packages and libraries

```
#### Random Forest for Regression Model  #####

#load Required packages and libraries

if(!require(caret)){install.packages('caret')}
if(!require(caTools)){install.packages('caTools')}
if(!require(randomForest)){install.packages('randomForest')}
library(caret)
library(caTools)
library(randomForest)
```

- Read in data (like SVR) as 'testSmoteRand'
- Split data into training and testing data in same ratio as SVR with train named randTrain, test is randTest and validate is randValid.
- Set hyper parameters:
  - contr = 10-fold cross validation
  - TuneGrid => .mtry = c(1:10)
- Now contr result is passed to 'trControl' parameter to retrain the model and .mtry of tuneGrid used to fine-tune the hyper parameters as below:

```
set.seed(3)
modelRf<-train(SOIL_PH~.,data=randTrain, method='rf', metric='RMSE',ntree=303,
               trControl=contr, tuneGrid=tuneGrid, importance=TRUE,nodesize=14)
modelRf
```

- Follow the steps below to select the best mtry and ntree to use in final model

```
# To know the best mtry and ntree to use
modelRf$bestTune$mtry      # [1] 10

## Tuning modelRf to know the best number of trees to use in final model

#creating a list of ntrees

safeTrees<-list()
for(ntree in c(250,400,450,550,600,800,850)) {
  set.seed(50)
  maxTrees<-train(SOIL_PH~.,data=randTrain, method='rf', metric='RMSE',
                  ntree=ntree,trControl=contr, tuneGrid=tuneGrid,
                  importance=TRUE,nodesize=14, maxnodes=24)
  key<-toString(ntree)
  safeTrees[[key]]<-maxTrees
}
tree_result<-resamples(safeTrees)
summary(tree_result)
# best ntree = 600 (with lowest error metrics), mtry=10 (from above)
```

- **Pre-modeling**
  - Fit random forest function
    set.seed(5)
    modelrand <- randomForest(formula = SOIL_PH ~ .,data = testSmoteRand)
  - Get results of the set hyper-parameters following steps below:

```
#display test MSE by number of trees
plot(modelrand, main='Plot of Errors by number of Trees')

#find number of trees that produce lowest test MSE
set.seed(4)
which.min(modelrand$mse)                        # [1] 498

#find RMSE of best model
sqrt(modelrand$mse[which.min(modelrand$mse)])     # [1] 0.2208

#produce variable importance plot
varImpPlot(modelrand, main=' Variable Importance Plot')
```

- **Modeling and predictions**
- Use the hyper parameter results above to fine-tune and train the final model, as such:
  tune_model2<-train(SOIL_PH~.,data=randTrain, mtryStart=4, trace=FALSE,
  ntreeTry=600,trControl=contr, importance=TRUE,
  improve=0.01, stepFactor=1.5)
- Subset test and valid data samples like SVR model
- Predict using both test and validate data

- **Evaluating results**
- Use in-built R algorithms to evaluate RMSE, COR and MSE like in SVR model
- Result should be like below for both test and validate

```
# Evaluating the randomForest testing results

#RMSE:
sqrt(mean((test_target-predRf)^2)) # [1] 0.0059

#R2:
cor(test_target,predRf)^2          # [1] 0.9994

#MAE:
MAE(test_target,predRf)            # [1] 0.0010

# MSE
(mean((test_target-predRf)^2))^2   # [1] 1.2088e-09
```

```
# Evaluating the randomForest validating results

#RMSE:
sqrt(mean((valid_target-predRf2)^2)) # [1] 0.0016

#R2:
cor(valid_target,predRf2)^2          # [1] 0.9999

#MAE:
MAE(valid_target,predRf2)            # [1] 0.0006

#MSE
(mean((valid_target-predRf2)^2))^2   # [1] 5.9346e-12
```

3.2 Classification Models

3.2.1 Non-Parametric Naïve Bayes Soil_pH Model

- Load the required packages and libraries

```
#### Non-parametric Naive Bayes Model #####

# Installing required packages and libraries
install.packages('caTools')
install.packages('e1071')
install.packages('caret')
install.packages('ggplot2')
library (e1071)
library(ggplot2)
library(caTools)
```

-Read- in the smote dataset as 'testSmoteNav'.

- Normalise testSmoteNav excluding the Soil_ph
- Split data into training and testing data in ratio 0.8n and 0.2 respectively.
- Set train and test labels
- Factorise target, Soil_pH, and convert to binary level (as a dataframe) like below syntax:

testSmoteNav$SOIL_PH <- as.factor(ifelse(testSmoteNav$SOIL_PH >= 0.50,2,1))

Modeling and prediction:
Tunning with kernel distribution:
-Set hyper-parameters: usekernel and poisson as TRUE and FALSE respectively makes the Naive Bayes model a non-parametric one
- using kernel density estimates and laplace =1 enables smoothening.

- Follow steps below to achieve this:

```
 ## Fitting Naive Bayes Model as a non-parametric model

modelNB <- naiveBayes(x=NavTrain, y=Train_label,usekernel=TRUE,
          poisson= FALSE,Bernoulli= TRUE, laplace = 1, bw=SJ)

modelNB
summary(modelNB)

## Predicting on test data
set.seed(2)
predNav <- predict(modelNB, newdata=NavTest,usekernel=usekernel)

# Confusion Matrix
confusionMatrix(predNav,Test_label)
```

Evaluation

- Function below used to get the result:

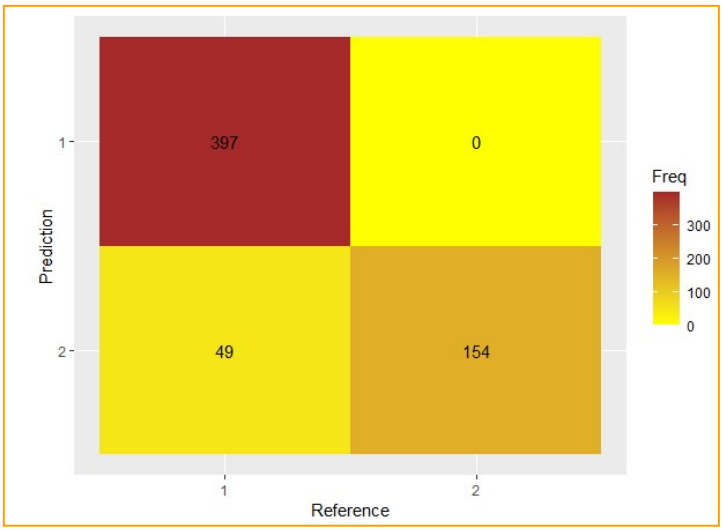
```
myStats<- function(Actual, Predicted){
  confusion.table<- table(Actual=Actual,Predicted=Predicted)
  output<- list(confusion.table= confusion.table)
  TN= confusion.table[1]
  FN= confusion.table[2]
  FP<-confusion.table[3]
  TP<-confusion.table[4]
  output$accuracy=(TP+TN)/sum(confusion.table)
  output$precission<- (TP)/ (TP+FP)
  output$sensitivity<-(TP)/(TP+FN)
  output$specificity<- (TN)/(TN+FP)
  output$FPR <- (FP)/(TN+FP)
  output$f1_score <-(2*((TP)/(TP+FP))*((TP)/(TP+FN))) / (((TP)/(TP+FP))+((TP)/(TP+FN)))


  return(output)
}

myStats(Actual=Test_label, Predicted=predNav)
```

Visualise confusion matrix

- Use code below:

```
Cf1<- confusionMatrix(as.factor(testSmoteNav[-ran,12]),as.factor(predNav),
                      dnn= c('Prediction','Reference'))

plt<-as.data.frame(Cf1$table)
plt$Prediction<- factor(plt$Prediction,levels= rev(levels(plt$Prediction)))

ggplot(plt,aes(Prediction, Reference, fill=Freq))+
  geom_tile()+geom_text(aes(label=Freq))+
  scale_fill_gradient(low='yellow', high='brown')+
  labs(x='Reference', y='Prediction')+
  scale_x_discrete(labels=c('1','2'))+
  scale_y_discrete(labels=c('2','1'))
```

- This should look like below:



3.2.2 Non-Parametric Naïve Bayes Soil_Texture Model

- Here, target is SOIL_TEXTU
- Repeat the same as above to factorisation step
- Use syntax below to factorise the soil_texture variable after grouping into four (4)

#grouped LoamyFineSand & Sandyloam; sandy, concretional & sandyClay;

testSmoteSoil$SOIL_TEXTU <- cut(testSmoteSoil$SOIL_TEXTU,4,
labels=c('ClayLoam', 'LoamyFineSand',' SandyClay',' SiltyClay'))

- Set training and testing data
- Set train and test labels

Modeling , prediction and confusion matrix

- Follow the steps below

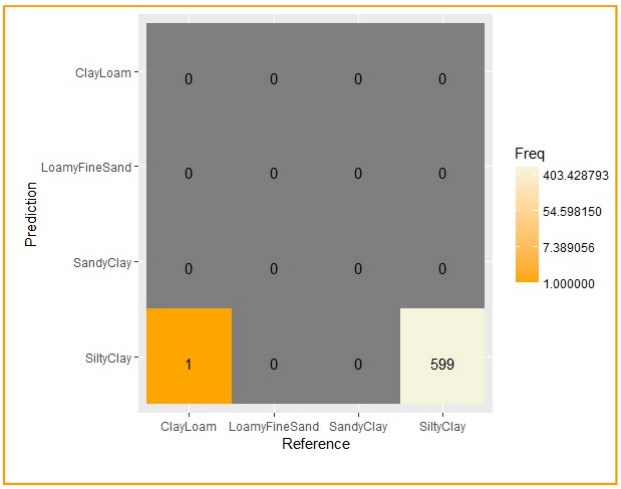
```
## Fitting non-parametric Naive Bayes model

soil_model <- naiveBayes(x=soilTrain, y=Train_soil,usekernel=TRUE,laplace =6)

soil_model
summary(soil_model)

## Predicting on test data
set.seed(2)
predSoil <- predict(soil_model, newdata=soilTest)


# Confusion Matrix
confusionMatrix(predSoil,Test_soil)
```

Evaluation

-Use same evaluation function like Soil_ph model

- Use same syntax for confusion matrix in above model and substitute with the test data and prediction names of this Soil_textu model

Soil_model confusion matrix should look like this:



3.2.3 K-Nearest Neighbour Soil_pH Model

- Target here is SOIL_PH variable
- Load required packages and libraries as below
- Read- in the data as ‘ testSmoteKNN’
- Normalise data without target variable
- Set training and testing data in ratio 0.8 and 0.2crespectively
- Factorise Soil_ph and convert to binary like in Naïve Bayes Soil_ph model
- Set training and testing labels

```
#### ###  K-Nearest Neighbour models

# (1) Install required packages and libraries
if(!require(dplyr)){install.packages('dplyr')}
if(!require(tidyr)){install.packages('tidyr')}
if(!require(ggplot2)){install.packages('ggplot2')}
if(!require(ROCR)){install.packages('ROCR')}
if(!require(pROC)){install.packages('pROC')}
if(!require(caret)){install.packages('caret')}
if(!require(caTools)){install.packages('caTools')}
if(!require(apaTables)){install.packages('apaTables')}
if(!require(reshape2)){install.packages('reshape2')}
if(!require(class)){install.packages('class')}
if(!require(SparseM)){install.packages('SparseM')}
if(!require(interactions)){install.packages('interactions')}
if(!require(psych)){install.packages('psych')}
if(!require(mgcv)){install.packages('mgcv')}
if(!require(DMwR2)){install.packages('DMwR2')}
if(!require(kernlab)){install.packages('kernlab')}
if(!require(e1071)){install.packages('e1071')}
if(!require(datarium)){install.packages('datarium')}
if(!require(readxl)){install.packages('readxl')}
library(mgcv)
library(SparseM)
library(class)
library (dplyr)
library (tidyr)
library (ggplot2)
library(readxl)
library(datarium)
library(interactions)
library(reshape2)
library(psych)
library(apaTables)
library(caTools)
library(caret)
library(pROC)
library(ROCR)
library(DMwR2)
library(kernlab)
library (e1071)
```

- **Modeling**
- Selecting the better k value to use for final model
- Follow steps below for the selection:

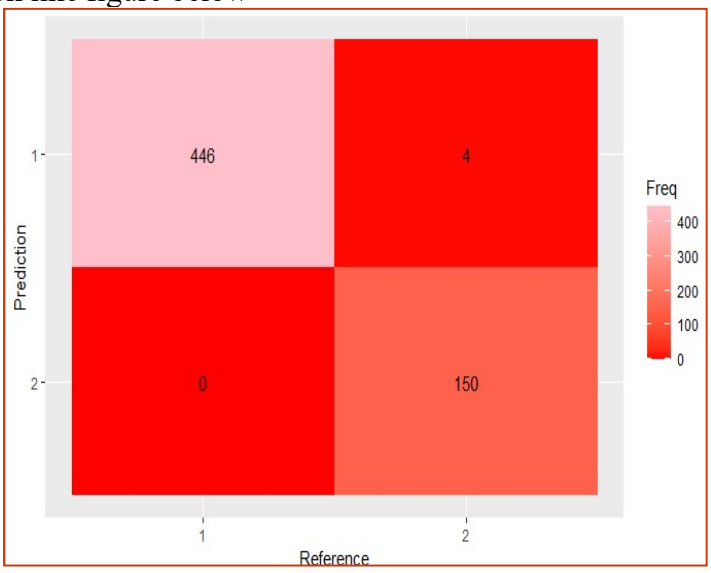
```
#####      Modeling and prediction of K-Nearest Neighbour    #####

# Since sqroot of 2397 observations =48.96 (k-value rule of thumb ),two models
# knn.48 and knn.49 (of k=48 and k=49 respectively) are built for comparison


# Modeling with training data for both k=48 and k=49

set.seed(4)

knn1.48<- knn(train=Train.knn1, test=Test.knn1,k=48, cl=Train.knn_label1)

knn1.49<- knn(train=Train.knn1, test=Test.knn1, cl=Train.knn_label1,k=49)


###   Models performance evaluation ###

# Accuracy: The formula is used here instead of the algorithm to avoid clog

acc1.48 <- 100*sum(Test.knn_label1==knn1.48)/NROW(Test.knn_label1) # [1]  99.33

acc1.49 <- 100*sum(Test.knn_label1==knn1.49)/NROW(Test.knn_label1) # [1]  99.33


# Tables
table(knn1.48, Test.knn_label1)

table(knn1.49, Test.knn_label1)

# Decision: k-value=48 selected as both have same prediction accuracy
 # and classification.
```

- **Prediction**

```
### Prediction on test data using kvalue=48

set.seed(10)

predknn1<- knn(train=Train.knn1, test=Test.knn1, cl=Train.knn_label1,k=48)
```

Evaluation , results and visualisation

- Set confusion matrix like Naïve Bayes soil_ph model, substituting objects for KNN test label and prediction ,predknn1
- Use the same evaluation function for Naïve Bayes, using knn values: Actual=Test.knn_label1, Predicted=predknn1
- Use same confusion matrix plot as in Naïve Bayes above, substituting for knn objects. and should look like figure below



3.2.4 K-Nearest Neighbour Model for Soil_Texture Model

- Here , the target is the soil texture variable
- Load data as ‘ soil_knn’
- Normalise without target variable
- Split data into training and testing in ration 80:20
- Use syntax below to convert target, SOIL_TEXTU into four groups:

```
# Converting target,Soil_texture to factors and 4 grouped factor levels

soil_knn$SOIL_TEXTU <- cut(soil_knn$SOIL_TEXTU,4, c('ClayLoam', 'LoamyFineSand',
                                                    'SandyClay','SiltyClay'))
```

- Set train and test labels

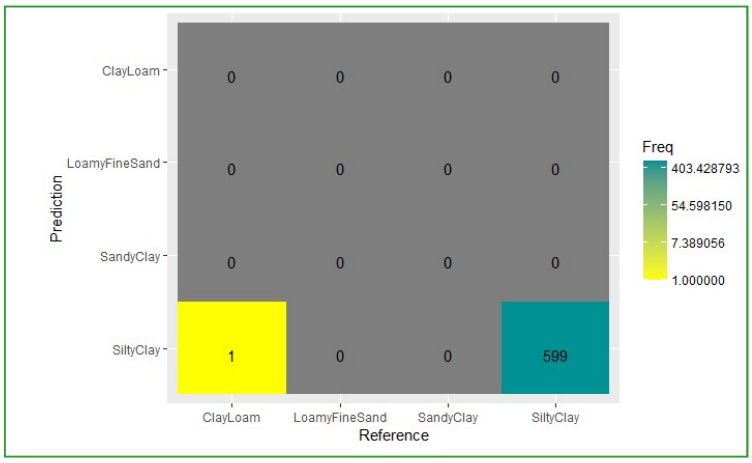
Modeling with kvalue of 48 and prediction

```
knn.text<- knn(train=Train_knn, test=Test_knn,cl=knn.Train_label,k=48)
```

```
set.seed(10)

predSoilknn<- knn(train=Train_knn, test=Test_knn, cl=knn.Train_label,k=48)
```

Evaluation, result and confusion matrix

- Use same evaluation function as Naïve Bayes soil texture model
- Use same confusion matrix syntax as Naïve Bayes soil texture model
- The confusion matrix should look like below:



3.2.5 Support Vector Machine Models

- Load packages and libraries

```
###  Support Vector Machine  model

# Install needed packages & call the libraries
install.packages('dplyr')
install.packages('tidyr')
install.packages('ggplot2')
install.packages('psych')
install.packages('reshape2')
install.packages('apaTables')
install.packages('caTools')
install.packages(('caret'))
install.packages('pROC')
install.packages('ROCR')
install.packages('class')
install.packages('SparseM')
library(SparseM)
library(class)
library (dplyr)
library (tidyr)
library (ggplot2)
library(readxl)
library(datarium)
library(reshape2)
library(psych)
library(apaTables)
library(caTools)
library(caret)
library(pROC)
library(ROCR)
library(DMwR2)
library(smotefamily)
library(kernlab)
library (e1071)
```

- Target here is SOIL_PH variable
- Load data as ‘testSmoteSVM’

- Normalise data without target variable
- Factorise SOIL_PH and transform to binary level like the other soil_ph models
- Split data into training and testing ratio 80: 20 like below:

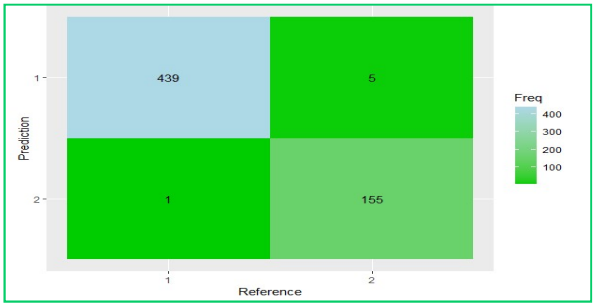
```
N <- nrow(testSmoteSVM)
set.seed(150)
# index splitting
svmTrain <- sample(1:N, size= N*0.8)
svmTrain <- sort(svmTrain)
svmTest <- setdiff(1:N,svmTrain)
```

Modeling and Prediction

- Here, three (3) separate models were developed with different kernels used for prediction on test data. This is to select the best of the kernels based on evaluating metrics accuracy, precision, recall and f1 score. The kernels are: 'Auto-selected', 'radial' and 'polynomial'.
- The steps below should be followed to develop and evaluate the three models:

```
### Auto-select Kernel Evaluation
set.seed(6)
matAut=confusionMatrix(testSmoteSVM$SOIL_PH[svmTest],predsvm[svmTest])
matAut

# Predictng with radial basis kernel for non-linearity

fitSVM1<-ksvm(SOIL_PH~., data=testSmoteSVM[svmTrain,], kernel='rbfdot')
predsvm1 <- predict(fitSVM1,testSmoteSVM)

# Radial Kernel Evaluation
set.seed(5)
matRadial<-confusionMatrix(testSmoteSVM$SOIL_PH[svmTest],predsvm1[svmTest])
matRadial

# Predictng with polynomial kernel for non-linearity and feature interaction effect

fitSVM2<-ksvm(SOIL_PH~., data=testSmoteSVM[svmTrain,], kernel='polydot')
predsvm2 <- predict(fitSVM2,testSmoteSVM)

#Polynomial Kernel Evaluation
matPolynomial<-confusionMatrix(testSmoteSVM$SOIL_PH[svmTest],predsvm2[svmTest])
matPolynomial
```

- The three models prediction results were marginally different in the evaluation matrices. Polynomial model was selected for the interaction effect advantage although not the optimal model.
- Same evaluation function used like for other models above
- Confusion matrix set like other models above, substituting for SVM values and should look like below:



3.2.6 Random Forest Model

- Target here is SOIL_PH variable.
- Load packages and libraries like random forest for regression model.
- Read-in data as 'testSmoteRand'.
- Factorise Soil_ph and transform to binary level.
- Set training and testing data in ration 80:20 respectively.

Modeling, prediction and confusion matrix

-Use steps below to achieve these:

```
## Fitting Random Forest model
set.seed(123)
RF_model <-randomForest(x=train[-12],
                        y=train$SOIL_PH,
                        ntree=600)  ## from regression model

RF_model

## Predicting test data

predRF<- predict(RF_model, newdata=test[-12])

# Confusion Matrix

RFconf<- table(test[,12],predRF)

RFconf
```

Evaluation and result

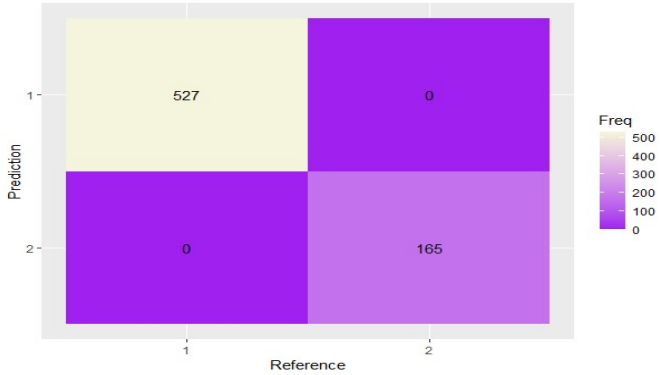
- Same evaluation function as in Naïve Bayes Soil_ph model, substituting for this random forest test[-12] and predRf.

Visualisations

- Below syntax are for visualising the model and the variables importance.

```
# Ploting model
plot(RF_model,col='red',main='Number of trees per error for random forest classifier')

# Vif plot
varImpPlot(RF_model,main='RF Classifier Variable Importance Plot', col='purple')
```

- Same confusion matrix plot syntax as in Naïve Bayes Soil_ph model, substituting for this random forest result parameters and should look like below figure.



4 Feature Interaction Detection

Here, the feature interactions are checked for the features that have interactions and although each possible interaction was handled during the modeling implementation stage with the use of polynomial kernels (SVR, SVM, NB), random forest with inbuilt handling and KNN indirectly through the distancing. The steps below should be followed to replicate work done on this.

- Load the required packages and libraries

```
# Install required packages and libraries

if(!require(dplyr)){install.packages('dplyr')}
if(!require(tidyr)){install.packages('tidyr')}
if(!require(caret)){install.packages('caret')}
if(!require(caTools)){install.packages('caTools')}
if(!require(reshape2)){install.packages('reshape2')}
if(!require(class)){install.packages('class')}
if(!require(SparseM)){install.packages('SparseM')}
if(!require(interactions)){install.packages('interactions')}
if(!require(kernlab)){install.packages('kernlab')}
if(!require(e1071)){install.packages('e1071')}
if(!require(readxl)){install.packages('readxl')}
if(!require(rms)){install.packages('rms')}
if(!require(interplot)){install.packages('interplot')}
library(interplot)
library(SparseM)
library(class)
library (dplyr)
library (tidyr)
library(readxl)
library(rms)
library(interactions)
library(reshape2)
library(apaTables)
library(caTools)
library(caret)
library(kernlab)
library (e1071)
```

- The steps below should be followed from dataset loading to applying anova

```
## Loading the dataset

data_interact<-read.csv('testSmote2$data.csv', sep=',', header=TRUE)
data_interact<-data_interact[,1:12]

# creating model for interaction analysis

model_intr<- lm(SOIL_PH~.+((I(GEOLOGY^2))+(I(SLOPE^2))+(I(ECOLOGICAL^2))+(I(DRAINAGE^2))+
(I(SUITABILIT^2))+(I(SOIL_TEXTU^2))+(I(VEGETATION^2))+(I(DISTRIBUTI^2))+
               (I(SOIL_CLA_1^2))+(I(MAJOR_CROP^2))+(I(Depth^2))),data=data_interact)

model_intr

summary(model_intr)

# Applying anova

anovaMod<- anova(model_intr)
anovaMod
```

- Anova on model_intr shows Geology, Slope, Vegetation and Major crop have significant interactions while drainage is marginal and others have no interaction.

```
> anovaMod<- anova(model_intr)
> anovaMod
Analysis of Variance Table

Response: SOIL_PH
                  Df  Sum Sq Mean Sq F value    Pr(>F)
GEOLOGY            1   1.842  1.8417 41.9169 1.110e-10 ***
SLOPE              1   0.147  0.1473  3.3527 0.0671943 .
ECOLOGICAL         1   0.626  0.6264 14.2563 0.0001626 ***
DRAINAGE           1   0.831  0.8310 18.9140 1.413e-05 ***
SUITABILIT         1   0.287  0.2875  6.5430 0.0105791 *
SOIL_TEXTU         1   0.030  0.0298  0.6783 0.4102557
VEGETATION         1   0.510  0.5100 11.6084 0.0006653 ***
DISTRIBUTI         1   0.172  0.1718  3.9109 0.0480671 *
SOIL_CLA_1         1   0.954  0.9542 21.7177 3.298e-06 ***
MAJOR_CROP         1   0.053  0.0528  1.2009 0.2732216
Depth              1   2.782  2.7818 63.3108 2.487e-15 ***
I(GEOLOGY^2)       1   0.967  0.9672 22.0135 2.830e-06 ***
I(SLOPE^2)         1   3.678  3.6781 83.7102 < 2.2e-16 ***
I(ECOLOGICAL^2)    1   0.158  0.1583  3.6034 0.0577576 .
I(DRAINAGE^2)      1   0.437  0.4367  9.9398 0.0016335 **
I(SUITABILIT^2)    1   0.091  0.0910  2.0714 0.1501918
I(SOIL_TEXTU^2)    1   0.005  0.0053  0.1205 0.7285128
I(VEGETATION^2)    1   1.278  1.2779 29.0838 7.474e-08 ***
I(DISTRIBUTI^2)    1   0.001  0.0015  0.0337 0.8543211
I(SOIL_CLA_1^2)    1   0.108  0.1078  2.4536 0.1173613
I(MAJOR_CROP^2)    1   0.557  0.5572 12.6816 0.0003751 ***
I(Depth^2)         1   0.006  0.0059  0.1352 0.7131174
Residuals       2974 130.672  0.0439
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

- Assess the interactions among the features with significant interaction with steps below and sample result

```
# Assessing the interaction among some of the significant variables

#This tests for interaction (shape differences across T, 3 d.f and global T,4 d.f)
#anova includes a test for nonlinear interaction

dd <- datadist(data_interact); options(datadist='dd')

# Checking interaction between geology and vegetation

Geov <- ols(SOIL_PH ~ rcs(GEOLOGY, 4) * VEGETATION, data=data_interact)
anova(Geov)

plot(Predict(Geov, GEOLOGY,VEGETATION)) # 2 estimated curves for 2 values of T


# Checking interaction in Geology and major crop

Geoc <- ols(SOIL_PH ~ rcs(GEOLOGY, 4) * MAJOR_CROP, data=data_interact)
anova(Geoc)
plot(Predict(Geoc, GEOLOGY,MAJOR_CROP))

## Checking interaction in vegetation and major crop
Vegc <- ols(SOIL_PH ~ rcs(VEGETATION, 4) * MAJOR_CROP, data=data_interact)
anova(Vegc)
plot(Predict(Vegc, VEGETATION,MAJOR_CROP))
```