Bansilal RamnathAgarwal Charitable Trust's
VISHWAKARMA INSTITUTE OF TECHNOLOGY – PUNE
Department of Multidisciplinary Engineering

MD2201 Data Science
Course Project

S.No	Div	Batch No	Group No	Roll No	Gr.No	Name of Student
1	C	B2	10	29	12110599	Vedant Arvind Mude
2				32	12110525	Rutuj Naresh Nagrale
3				33	12110041	Hetan Hemant Nandre
4				38	12110699	Aarya Sunil Nirgude
5				39	12110007	Tejashri Sampat Nirmal

1. **Project Title:** Water Potability Prediction

2. **Data Set Name:** "water_potability.csv"

3. **Data set Link:** https://www.kaggle.com/datasets/adityakadiwal/water-potability

4. **Data Set Description:**
    - Total Rows – 3276, Total Columns – 10
    - Missing values are present in pH, Sulfate, Trihalomethanes

5. **Objective:** The objective of this data science project is to develop a predictive model that can accurately predict the potability of water based on various water quality parameters. The model should be able to classify water samples as either potable (1) or non-potable (0) with a high degree of accuracy, sensitivity, and specificity.

6. **Data Preprocessing (if any):**

a. Using KNN - KNN imputation works by using the KNN algorithm to find the K nearest neighbors of a data point with missing values. The missing values are then imputed with the average (for numerical data) or mode (for categorical data) of the corresponding feature values in the K nearest neighbors.

We have followed below mentioned steps for KNN imputation in our project –

- Identified the features in the dataset that have missing values.
- Split the dataset into a training set (with no missing values) and a test set (with missing values). In our dataset, Missing values are present in pH, Sulfate, Trihalomethanes
- Used the training set to train a KNN model.
- For each data point in the test set with missing values, used the KNN model to find the K nearest neighbors.
- Imputed the missing values with the average/mode of the corresponding feature values

in the K nearest neighbors.

- Used the imputed dataset for further analysis or modeling.

b. **RF Imputation -** RF impute works by treating the rows with missing values as the target variable, and the columns without missing values as the input variables. The RF model is trained to predict the missing values based on the other features in the dataset. One advantage of RF impute over other imputation techniques is that it can handle non-linear relationships between the features, which can be difficult to capture with simpler imputation methods like mean or median imputation.

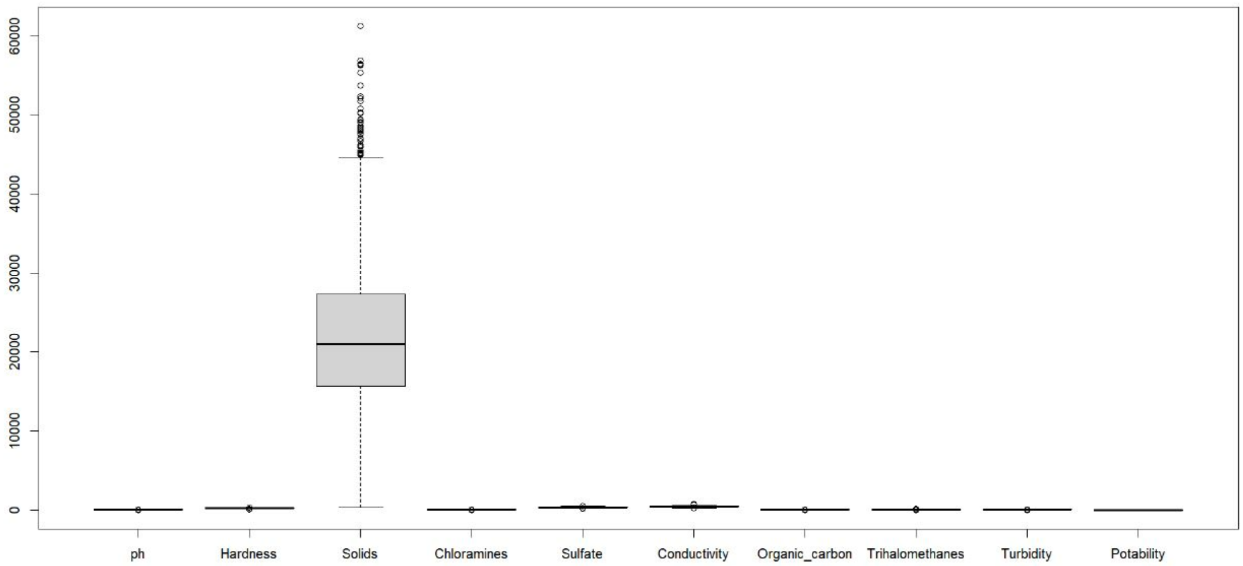

Figure 1 - Data visualization before removing outliers

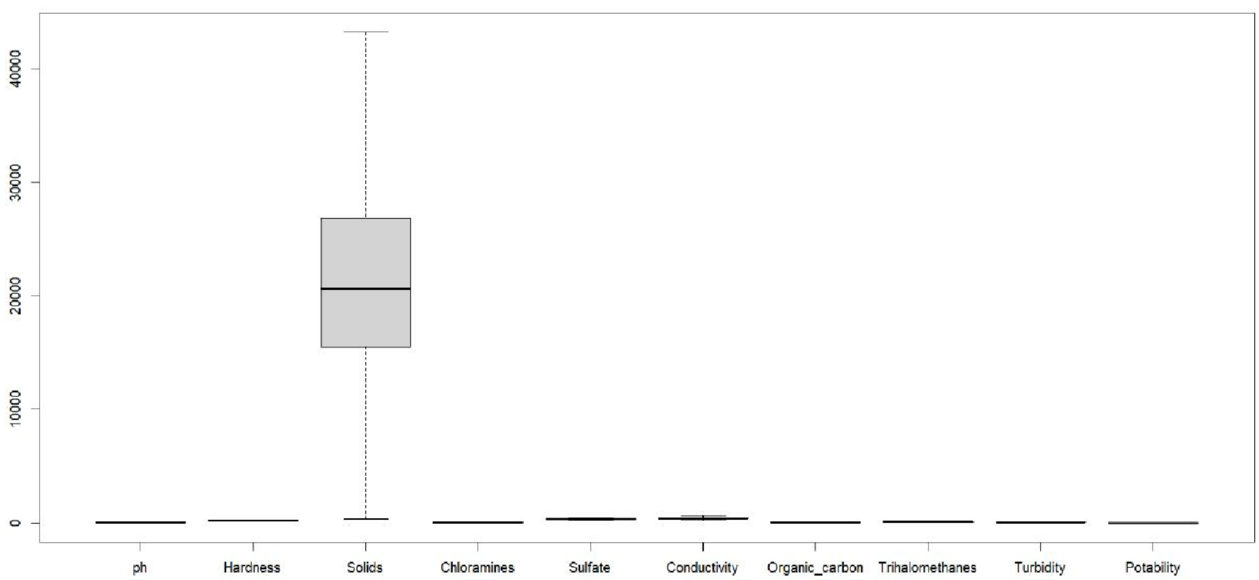

Figure 2 - Data visualization after removing outliers

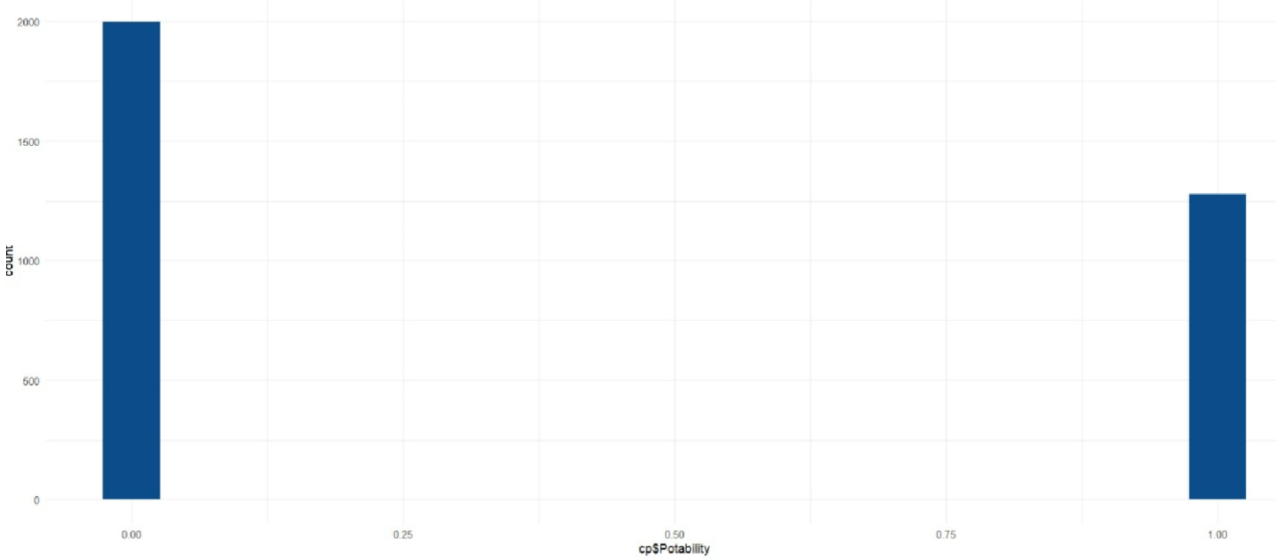

Figure 3 - Class Imbalance before Ovun Sampling

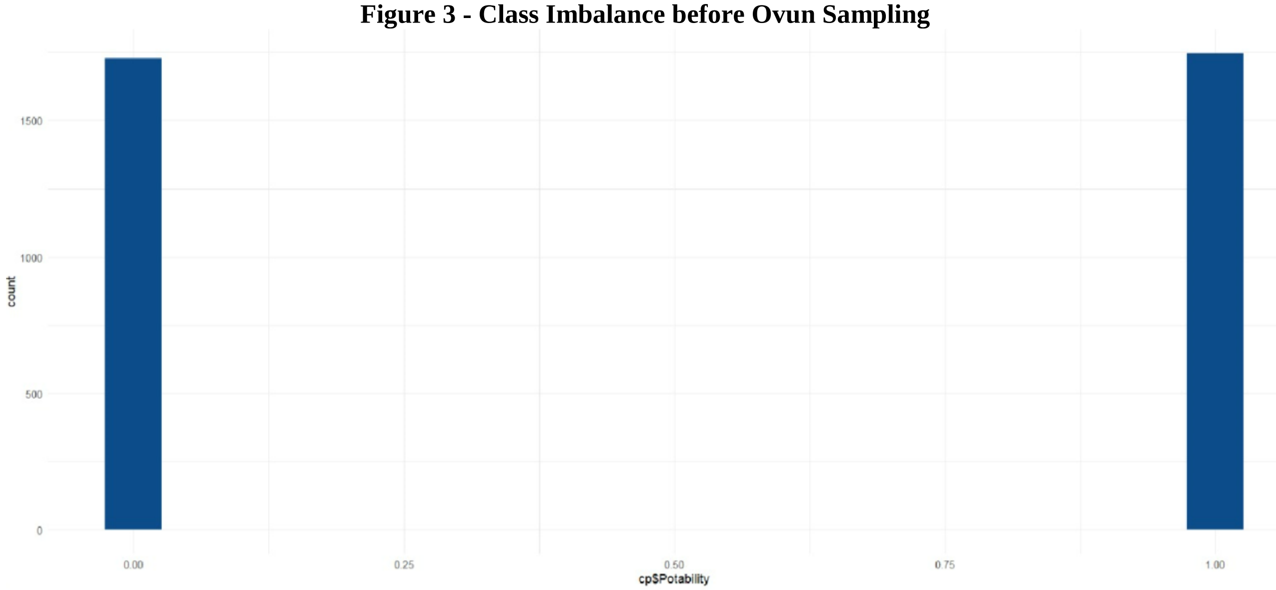

Figure 4 - Class Imbalance after Ovun Sampling

7. Feature Selection (if any):

Principal Component Analysis (PCA) - PCA is a technique that transforms a dataset into a new coordinate system by finding the principal components (or directions of highest variance) in the original data. These principal components can then be used as new features, which can potentially reduce the dimensionality of the dataset while retaining most of the information.

8. **Algorithms Implemented:** Explain algorithms with default arguments values or set hyperparameters value.

Decision Tree – The decision tree is implemented to train a model with "Potability" as the target variable and all other variables as predictors. The "set.seed(9999)" function sets a random seed value to ensure the reproducibility of the results. The "train" function trains the model using the specified hyperparameters, in this case using cross-validation with the "trControl" parameter and tuning the "cp" hyperparameter using the "tuneGrid" parameter. The resulting model is then used to predict the target variable ("Potability") for the dataset using the "predict" function, and a confusion matrix is created using the predicted values and the actual values of "Potability".

Random Forest – The Random Forest is implemented to train a model with "Potability" as the

target variable and all other variables as predictors. The "set.seed(9999)" function sets a random seed value to ensure the reproducibility of the results. The "train" function trains the model using the specified hyperparameters, in this case using cross-validation with the "trControl" parameter and tuning the "mtry" hyperparameter using the "tuneGrid" parameter. The resulting model is then used to predict the target variable ("Potability") for the "tst" dataset using the "predict" function, and a confusion matrix is created using the predicted values and the actual values of "Potability".

Xgb Linear – The gradient boosting algorithm is implemented using the "XgbLinear" method to train a model with "Potability" as the target variable and all other variables as predictors. The "set.seed(9999)" function sets a random seed value to ensure reproducibility of the results. The "train" function trains the model using the specified hyperparameters, in this case using cross-validation with the "trControl" parameter and tuning the "eta", "nrounds", "lambda", and "alpha" hyperparameters using the "tuneGrid" parameter. The resulting model is then used to predict the target variable ("Potability") for the dataset using the "predict" function, and a confusion matrix is created using the predicted values and the actual values of "Potability".

Xgb Tree – The gradient boosting algorithm is implemented using the "XgbTree" method to

train a model with "Potability" as the target variable and all other variables as predictors. The "set.seed(9999)" function sets a random seed value to ensure reproducibility of the results. The "expand.grid" function creates a grid of hyperparameters to be tested during the training process. In this case, the "xgb_grid_1" grid includes different values for "nrounds", "eta", "gamma", "max_depth", "min_child_weight", "subsample", and "colsample_bytree". The "train" function trains the model using the specified hyperparameters, in this case using cross-validation with the "trControl" parameter and tuning the hyperparameters using the "xgb_grid_1" parameter. The resulting model is then used to predict the target variable ("Potability") for the dataset using the "predict" function.

SVM Radial – Support Vector Machine algorithm is implemented using the "svmRinear" method to train a model with "Potability" as the target variable and all other variables as predictors. The "set.seed(9999)" function sets a random seed value to ensure reproducibility of the results. The "expand.grid" function creates a grid of hyperparameters to be tested during the training process. In this case, the "svm_r_grid" grid includes different values for "sigma" and "C". The "train" function trains the model using the specified hyperparameters, in this case using cross-validation with the "trControl" parameter and tuning the hyperparameters using the "svm_r_grid" parameter. The model is then used to predict the target variable ("Potability") for the dataset using the "predict" function.

SVM Polynomial – Support Vector Machine algorithm is implemented using the "svmPoly" method to train a model with "Potability" as the target variable and all other variables as predictors. The "set.seed(9999)" function sets a random seed value to ensure reproducibility of the results. The "expand.grid" function creates a grid of hyperparameters to be tested during the training process. In this case, the "svm_p_grid" grid includes different values for "degree", "scale", and "C". The "train" function trains the model using the specified hyperparameters, in this case using cross-validation with the "trControl" parameter and tuning the hyperparameters using the "svm_p_grid" parameter. The model is then used to predict the target variable ("Potability") for the dataset using the "predict" function.

AdaBoost – The AdaBoost algorithm is implemented using the "AdaBoost.M1" method to train a model on the "trn" dataset with "Potability" as the target variable and all other variables as predictors. The "set.seed(9999)" function sets a random seed value to ensure reproducibility of the results. The "expand.grid" function is used to create a grid of hyperparameters to be searched over during the training process. In this case, the grid specifies a fixed value of "mfinal" (the number of iterations) of 100, and a range of values for "coeflearn" (the coefficient for the learning rate), and "maxdepth" (the maximum depth of the decision trees). The "train" function trains the model using the specified hyperparameters, in this case using cross-validation with the "trControl" parameter.

Principal Component Analysis (PCA) –The PCA is implemented on the given dataset using the "**prcomp**" function in R. First, the code selects the numerical variables for the PCA analysis using "water[, 1:10]". Then, it applies PCA to the selected variables using "prcomp", with the parameters "center = TRUE" and "scale. = TRUE" indicating that the data should be centered and scaled before the analysis. The resulting PCA object is saved to "pca_result".

Next, the code uses the "predict" function to transform the original data ("mydata") into the principal component space defined by the PCA object ("pca_result"). The transformed data is saved to "mydata_pca".

The "**data.frame**" function is used to create a new data frame ("data_pca") from the

transformed data ("mydata_pca") and the "Potability" column of the original dataset. The column names of the new data frame correspond to the principal component scores (PC1-PC5).

Finally, the "**as.factor**" function is used to convert the "Potability" column to a factor variable.

9. CODE:

```
#COURSE PROJ----
cp<- read.csv("water_potability.csv")

#NA values
str(cp)
summary(cp)

#impute missing values using knn
library(DMwR2)
cp<-knnImputation(cp, k = 3, scale = TRUE, meth = "weighAvg",distData = NULL)

#impute missing values using rf
library(randomForest)
cp <- rfImpute(Potability ~ ., cp, ntree = 100,type="classification")
```

```
#remove outliers using boxplot
for (i in 2:10) {
  cp<-cp[!cp$ph %in% boxplot.stats(cp$ph)$out,]
  cp<-cp[!cp$Hardness %in% boxplot.stats(cp$Hardness)$out,]
  cp<-cp[!cp$Solids %in% boxplot.stats(cp$Solids)$out,]
  cp<-cp[!cp$Chloramines %in% boxplot.stats(cp$Chloramines)$out,]
  cp<-cp[!cp$Sulfate %in% boxplot.stats(cp$Sulfate)$out,]
  cp<-cp[!cp$Conductivity %in% boxplot.stats(cp$Conductivity)$out,]
  cp<-cp[!cp$Organic_carbon %in% boxplot.stats(cp$Organic_carbon)$out,]
  cp<-cp[!cp$Trihalomethanes %in% boxplot.stats(cp$Trihalomethanes)$out,]
  cp<-cp[!cp$Turbidity %in% boxplot.stats(cp$Turbidity)$out,]

}

#class imbalance
dim(cp)
head(cp)
table(cp$Potability)
prop.table(table(cp$Potability))

#smote
library(smotefamily)
smote_out=SMOTE(X=cp,target=cp$Potability,K=3,dup_size =1)
cp=smote_out$data
cp<-cp[,-11]

#ovun-over
library(ROSE)
library(smotefamily)
over=ovun.sample(Potability~.,cp,method="over")
cp=over$data

table(cp$Potability)
prop.table(table(cp$Potability))

cp$Potability=as.factor(cp$Potability)

#create training and testing data partitions
library(caret)
set.seed(9999)
cp<-cp[sample(1:nrow(cp)), ]
train <- createDataPartition(cp[,"Potability"],p=0.8,list=FALSE)
trn <- cp[train,]
tst <- cp[-train,]

#Algorithms applying
```

```
ctrl<-trainControl(method = "cv",number = 10)

#Decision Trees
set.seed(9999)
dec1<-train(Potability~.,data = trn,method="rpart",trControl=ctrl,tuneGrid =
expand.grid(cp = 0.001))#cp - hyperparameter
pred_1<-predict(dec1,tst)
confusionMatrix(table(tst[,"Potability"],pred_1))

#Random forest
set.seed(9999)
rand1<-train(Potability~.,data = trn,method="rf",trControl=ctrl,tuneGrid =
expand.grid(mtry = 3.16))#hyperparameter - mtry
pred_2<-predict(rand1,tst)
confusionMatrix(table(tst[,"Potability"],pred_2))

#Xgb linear
set.seed(9999)
xgb_lin<-
train(Potability~.,data=trn,method="xgbLinear",trControl=ctrl,tuneGrid=expand.grid
(eta = 0.3,nrounds=150,lambda=0.1,alpha=0.1))
pred_3<-predict(xgb_lin,tst)
confusionMatrix(table(tst[,"Potability"],pred_3))

#XGboost tree
set.seed(9999)
xgb_grid_1 = expand.grid(nrounds=150,eta=0.3, gamma=0.2, max_depth=6,
min_child_weight=1, subsample=1, colsample_bytree=1)
xgbTree1<-train(Potability~.,data=trn,method="xgbTree",trControl=ctrl,tuneGrid =
xgb_grid_1)
pred_4<-predict(xgbTree1,tst)
confusionMatrix(table(tst[,"Potability"],pred_4))

#SVM Radial
set.seed(9999)
svm_r_grid =expand.grid(sigma = c(0.01, 0.015, 0.2),C = c(0.75, 0.9, 1, 1.1,
1.25))
svm_r<-train(Potability~.,data=trn,method="svmRadial",trControl=ctrl,tuneGrid =
svm_r_grid)
pred_6<-predict(svm_r,tst)
confusionMatrix(table(tst[,"Potability"],pred_6))

#SVM Polynomial
set.seed(9999)
svm_p_grid =expand.grid(degree=2, scale=5, C=5)
```

```
svm_p<-train(Potability~.,data=trn,method="svmPoly",trControl=ctrl,tuneGrid =
svm_p_grid)
pred_7<-predict(svm_p,tst)
confusionMatrix(table(tst[,"Potability"],pred_7))

#Adaboost
set.seed(9999)
adagrid = expand.grid( mfinal = 100,coeflearn = c("Breiman", "Freund",
"Zhu"),maxdepth = 30)
ada<-train(Potability~.,data=trn,method="AdaBoost.M1",trControl=ctrl, tuneGrid =
adagrid)
pred_9<-predict(ada,tst)
confusionMatrix(table(tst[,"Potability"],pred_9))


#PRINCIPAL COMPONENT ANALYSIS----

# Load the water dataset
water<- read.csv("water_potability.csv")

# Prepare the data
# Remove rows with missing values by using na.omit
water <- na.omit(water)

# Removing missing values by kNN imputation
library(VIM)
water<-kNN(water,k=5)

# Removing missing values by rf imputation
library(randomForest)
na.roughfix(water)
water<-randomForest::rfImpute(Potability ~ .,ntree=200,iter=5,data=water)

boxplot(water)

#remove outliers using boxplot
for (i in 1:9) {
  water<-water[!water$ph %in% boxplot.stats(water$ph)$out,]
  water<-water[!water$Hardness %in% boxplot.stats(water$Hardness)$out,]
  water<-water[!water$Solids %in% boxplot.stats(water$Solids)$out,]
  water<-water[!water$Chloramines %in% boxplot.stats(water$Chloramines)$out,]
  water<-water[!water$Sulfate %in% boxplot.stats(water$Sulfate)$out,]
  water<-water[!water$Conductivity %in% boxplot.stats(water$Conductivity)$out,]
  water<-water[!water$Organic_carbon %in%
boxplot.stats(water$Organic_carbon)$out,]
```

```
  water<-water[!water$Trihalomethanes %in%
boxplot.stats(water$Trihalomethanes)$out,]
  water<-water[!water$Turbidity %in% boxplot.stats(water$Turbidity)$out,]
  }

boxplot(water)

#class imbalance
table(water$Potability)
prop.table(table(water$Potability))

#smote
library(smotefamily)
smote_out=SMOTE(X=water,target=water$Potability,K=3,dup_size =1)
water=smote_out$data

# ovun over sampling
over=ovun.sample(Potability~.,water,method="over")
water=over$data
table(water$Potability)
prop.table(table(water$Potability))
water$Potability=as.factor(water$Potability)

#pca
mydata <- water[, 1:10] # Select the numerical variables to apply PCA to
pca_result <- prcomp(mydata, center = TRUE, scale. = TRUE) # Apply PCA and save
the results to res.pca

mydata_pca <- predict(pca_result, newdata = mydata)

data_pca<-data.frame
(PC1=mydata_pca[,1],PC2=mydata_pca[,2],PC3=mydata_pca[,3],PC4=mydata_pca[,4],PC5=m
ydata_pca[,5],Potability=water$Potability)

data_pca$Potability<-as.factor(data_pca$Potability)

library(caret)
#create training and testing data partitions
set.seed(9999)
train <- createDataPartition(data_pca[,"Potability"],p=0.8,list=FALSE)
trn <- data_pca[train,]
tst <- data_pca[-train,]

#Algorithms applying
ctrl<-trainControl(method = "cv",number = 10)
```

```
#Decision Trees

set.seed(9999)
dec1<-train(Potability~.,data = trn,method="rpart",trControl=ctrl,tuneGrid =
expand.grid(cp = 0.001))#cp - hyperparameter
pred_1<-predict(dec1,tst)
confusionMatrix(table(tst[,"Potability"],pred_1))

#Random forest
set.seed(9999)
rand1<-train(Potability~.,data = trn,method="rf",trControl=ctrl,tuneGrid =
expand.grid(mtry = 2))#hyperparameter - mtry
pred_2<-predict(rand1,tst)
confusionMatrix(table(tst[,"Potability"],pred_2))

#Xgb linear
set.seed(9999)
xgb_lin<-
train(Potability~.,data=trn,method="xgbLinear",trControl=ctrl,tuneGrid=expand.grid
(eta = 0.1,nrounds=150,lambda=0.5,alpha=0.5))
pred_3<-predict(xgb_lin,tst)
confusionMatrix(table(tst[,"Potability"],pred_3))

#XGboost tree
set.seed(9999)
xgb_grid_1 = expand.grid(nrounds = 150,max_depth = 10,eta = 0.3,gamma =
5,colsample_bytree = 0.9,min_child_weight = 10,subsample = 0.8)
xgbTree1<-train(Potability~.,data=trn,method="xgbTree",trControl=ctrl,tuneGrid =
xgb_grid_1)
pred_4<-predict(xgbTree1,tst)
confusionMatrix(table(tst[,"Potability"],pred_4))

#SVM Radial
set.seed(9999)
svm_r_grid =expand.grid(sigma = c(0.01, 0.015, 0.2),C = c(0.75, 0.9, 1, 1.1,
1.25))
svm_r<-train(Potability~.,data=trn,method="svmRadial",trControl=ctrl,tuneGrid =
svm_r_grid)
pred_6<-predict(svm_r,tst)
confusionMatrix(table(tst[,"Potability"],pred_6))

#SVM Polynomial
set.seed(9999)
svm_p_grid =expand.grid(degree=2, scale=5, C=5)
svm_p<-train(Potability~.,data=trn,method="svmPoly",trControl=ctrl,tuneGrid =
svm_p_grid)
```

```
pred_7<-predict(svm_p,tst)
confusionMatrix(table(tst[,"Potability"],pred_7))

#Adaboost
set.seed(9999)
adagrid = expand.grid( mfinal = 100,coeflearn = c("Breiman", "Freund",
"Zhu"),maxdepth = 30)
ada<-train(Potability~.,data=trn,method="AdaBoost.M1",trControl=ctrl, tuneGrid =
adagrid)
pred_9<-predict(ada,tst)
confusionMatrix(table(tst[,"Potability"],pred_9))
```

10. **Evaluation Parameters used:** Accuracy, Specificity, Sensitivity, Precision
  a. **Accuracy –** Highest Accuracy – 83.76%(Without using PCA)
     Highest Accuracy – 96.25%(With PCA)
  b. **Specificity -** Highest Specificity – 85.75%(Without using PCA)
     Highest Specificity – 96.21%(With PCA)
  c. **Sensitivity -** Highest Sensitivity – 82.59%(Without using PCA)
     Highest Specificity – 96.63%(With PCA)
  d. **Precision -** Highest Precision – 85.32%(Without using PCA)
     Highest Precision – 96.21%(With PCA)

11. Results:

a. Imputation: KNN, Class Imbalance: SMOTE

Algorithm	Imputation	Class Imbalance	Training - Testing	Accuracy	Specificity	Sensitivity	Precision
Decision tree	KNN	SMOTE	80-20	64.34	66.44	61.49	57.39
Random forest	KNN	SMOTE	80-20	76.18	73.78	80.6	62.61
XGBoost Linear	KNN	SMOTE	80-20	74.21	74.39	73.95	66.67
XGBoost Tree	KNN	SMOTE	80-20	72.37	73.14	71.29	65.51
radial SVM	KNN	SMOTE	80-20	74.34	73.11	76.41	62.9
polynomial SVM	KNN	SMOTE	80-20	67.89	66.1	72.05	47.83
AdaBoost	KNN	SMOTE	80-20	76.58	76.33	73.86	70.43

b. Imputation: KNN, Class Imbalance: Ovun - Over

Algorithm	Imputation	Class Imbalance	Training - Testing	Accuracy	Specificity	Sensitivity	Precision
Decision tree	KNN	Ovun - Over	80-20	70.67	69.71	71.75	67.58
Random forest	KNN	Ovun - Over	80-20	81.61	81.25	81.99	80.73
XGBoost Linear	KNN	Ovun - Over	80-20	79.33	77.46	81.52	75.54
XGBoost Tree	KNN	Ovun - Over	80-20	79.79	77.97	81.91	76.15
radial SVM	KNN	Ovun - Over	80-20	71.88	71.99	71.78	71.56
polynomial SVM	KNN	Ovun - Over	80-20	66.01	67.2	64.93	68.5
Adaboost	KNN	Ovun - Over	80-20	83.76	85.05	82.54	85.32

c. Imputation: RF Impute, Class Imbalance: SMOTE

Algorithm	Imputation	Class Imbalance	Training - Testing	Accuracy	Specificity	Sensitivity	Precision
Decision tree	rf impute	SMOTE	80-20	60.47	62.01	58.24	51.35
Random forest	rf impute	SMOTE	80-20	71.41	69.68	74.46	58.11
XGBoost Linear	rf impute	SMOTE	80-20	65.94	66.84	64.66	58.11
XGBoost Tree	rf impute	SMOTE	80-20	65.94	67.03	64.44	58.78
radial SVM	rf impute	SMOTE	80-20	69.53	69.15	70.12	59.46
polynomial SVM	rf impute	SMOTE	80-20	63.75	61.57	70.51	37.16
Adaboost	rf impute	SMOTE	80-20	71.02	71.61	70.21	64.5

d. **Imputation:** RF Impute, **Class Imbalance:** Ovun - Over

Algorithm	Imputation	Class Imbalance	Training - Testing	Accuracy	Specificity	Sensitivity	Precision
Decision tree	rf impute	Ovun - Over	80-20	66.14	63.64	69.32	60
Random forest	rf impute	Ovun - Over	80-20	81.58	85.71	78.46	87.93
XGBoost Linear	rf impute	Ovun - Over	80-20	79.47	79.42	79.52	80.34
XGBoost Tree	rf impute	Ovun - Over	80-20	73.51	72.32	74.73	72.41
radial SVM	rf impute	Ovun - Over	80-20	70.88	70.96	70.81	72.76
polynomial SVM	rf impute	Ovun - Over	80-20	59.82	60.76	59.16	67.93
Adaboost	rf impute	Ovun - Over	80-20	83.45	84.3	82.59	84.03

e. PCA (All Variations)

Algorithm	Imputation	Class Imbalance	Training - Testing	Accuracy	Specificity	Sensitivity	Precision
Decision tree	na.omit	no oversampling	80-20	85.8	80.58	89.32	87.2
Random forest	outliers removed		columns pc1,2,3,4,5	92.17	90.84	92.99	94.31
XGBoost Linear				92.17	89.63	93.81	93.36
XGBoost Tree				91.88	88.97	93.78	92.89
radial SVM				92.46	92.19	92.63	95.26
polynomial SVM				92.75	91.6	93.46	94.79
Adaboost				92.17	89.05	94.23	92.89

Algorithm	Imputation	Class Imbalance	Training - Testing	Accuracy	Specificity	Sensitivity	Precision
Decision tree	na.omit	ovum oversampling	80-20	88.76	88.53	89	88.15
Random forest	outliers removed		columns pc1,2,3,4,5	96.25	95.45	97.1	95.26
XGBoost Linear				96.02	95.43	96.63	95.26
XGBoost Tree				92.51	92.2	92.28	91.94
radial SVM				95.08	96.21	93.98	96.21
polynomial SVM				94.38	94.86	93.9	94.79
Adaboost				95.78	85.41	96.17	95.26

Algorithm	Imputation	Class Imbalance	Training - Testing	Accuracy	Specificity	Sensitivity	Precision
Decision tree	na.omit	SMOTE	80-20	84.58	85.45	93.41	81.04
Random forest	outliers removed		columns pc1,2,3,4,5	87.92	87.81	88.06	83.89
XGBoost Linear				83.96	82.65	86.02	75.83
XGBoost Tree				85.62	85.21	86.22	80.09
radial SVM				87.71	88.04	87.25	84.36
polynomial SVM				86.46	85.29	87.24	81.04
Adaboost				87.08	86.32	88.21	81.52

Algorithm	Imputation	Class Imbalance	Training - Testing	Accuracy	Specificity	Sensitivity	Precision
Decision tree	KNN	no oversampling	80-20	77.51	71.96	80.45	84.46
Random forest	outliers removed		columns pc1,2,3,4,5	78.98	78.98	81.22	86.22
XGBoost Linear				77.15	71.43	80.17	84.16
XGBoost Tree				78.79	73.94	81.34	85.63
radial SVM				79.16	76.14	80.59	87.68
polynomial SVM				78.79	76.79	79.68	88.56
Adaboost				77.33	77.33	80.39	84.16

Algorithm	Imputation	Class Imbalance	Training - Testing	Accuracy	Specificity	Sensitivity	Precision
Decision tree	rf imput	no oversampling	80-20	76.91	68.39	81.88	81.61
Random forest	outliers removed		columns pc1,2,3,4,5	80.93	74.85	84.26	85.95
XGBoost Linear				77.75	69.1	82.99	81.61
XGBoost Tree				82.42	77.11	85.29	87.29
radial SVM				81.57	77.92	83.33	88.63
polynomial SVM				81.99	79.33	83.23	89.63
Adaboost				78.39	70.76	82.72	83.28

Algorithm	Imputation	Class Imbalance	Training - Testing	Accuracy	Specificity	Sensitivity	Precision
Decision tree	KNN	ovum oversampling	80-20	84.99	84.27	85.71	84.46
Random forest	outliers removed		columns pc1,2,3,4,5	89.9	87.08	93.06	86.51
XGBoost Linear				89.75	87.68	91.98	87.39
XGBoost Tree				87.52	86.05	89.06	85.92
radial SVM				84.7	84.18	85.21	84.46
polynomial SVM				84.84	84.64	85.04	85.04
Adaboost				89.75	87.46	92.24	87.1

Algorithm	Imputation	Class Imbalance	Training - Testing	Accuracy	Specificity	Sensitivity	Precision
Decision tree	rf	ovum oversampling	80-20	85.12	86.71	83.7	87.54
Random forest	outliers removed		columns pc1,2,3,4,5	91.9	91.97	91.83	92.13
XGBoost Linear				91.74	92.52	91	92.79
XGBoost Tree				87.44	88.89	86.12	89.51
radial SVM				85.45	87.86	83.38	88.85
polynomial SVM				85.45	89.55	82.2	90.82
Adaboost				91.9	92.54	91.29	92.79

Algorithm	Imputation	Class Imbalance	Training - Testing	Accuracy	Specificity	Sensitivity	Precision
Decision tree	knn	SMOTE	80-20	77.56	77.55	77.56	70.97
Random forest	outliers removed		columns pc1,2,3,4,5	83.27	82.06	85.02	76.54
XGBoost Linear				81.54	80.95	82.37	75.37
XGBoost Tree				82.07	81.99	82.19	77.13
radial SVM				81.81	81.18	82.69	75.66
polynomial SVM				81.41	80.77	82.32	75.07
Adaboost				83.67	83.07	84.49	78.3

Algorithm	Imputation	Class Imbalance	Training - Testing	Accuracy	Specificity	Sensitivity	Precision
Decision tree	rf	SMOTE	80-20	88.15	87.19	89.4	84.33
Random forest	outliers removed		columns pc1,2,3,4,5	89.38	88.08	91.1	85.33
XGBoost Linear				88.77	87.74	90.11	85
XGBoost Tree				88	87.57	88.54	85
radial SVM				89.85	87.97	92.39	85
polynomial SVM				90	87.6	93.36	84.33
Adaboost				90.62	89.91	91.64	87.67

12. **Conclusions:** This Water Potability prediction project involved preprocessing the data, exploring the data to gain insights, and selecting appropriate features for modeling. Various classification algorithms were tested, and the performance was evaluated using relevant metrics.

    The results showed that it is possible to accurately predict water potability using machine learning techniques.

    1. The best-performing model was AdaBoost without using PCA(Accuracy – 83.76%). Here, imputation is done using KNN and class imbalance is removed using Ovun-over.

2. The best-performing model was Random Forest using PCA(Accuracy – 96.25%) with feature columns 1,2,3,4, and 5. Here, imputation is done by removing outliers using na.omit() and class imbalance is removed using Ovun-oversampling.