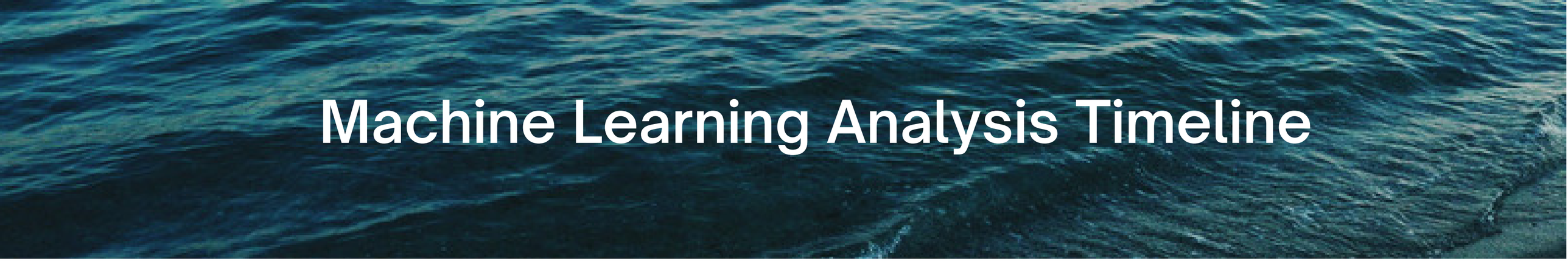
Analysis of Water Quality Dataset

Classification Method

Team-12
D. Sai Rizwana
S.R. Bhargavi
Shubham Singh
Rohit Menon
Sai Mohan

Bhavan's Vivekananda College | BSC Hons Data Science



Machine Learning Analysis Timeline

1. DATA PRE-PROCESSING
2. EXPLORATORY DATA ANALYSIS
3. TRAINING THE MODEL
4. DETERMINING MODEL ACCURACY
5. COMPARING RESULTS WITH OTHER MODELS
6. CONCLUSION AND INSIGHTS

About The Data

The data was aggregated from various sources to check the quality of the water based on different chemical compositions in it.

Objective

Water quality is measured by several factors, such as the concentration of dissolved oxygen, bacteria levels, the amount of salt, or the amount of material suspended in the water. So water testing is carried out to meet the regulatory requirementsand adhere to the safety procedures that are needed for pollutant-free water. This is a broad concept that involves several procedures to analyze and evaluate the quality of water.

The Path

Implementing multiple machine learning models to fit the best model for the dataset.

Data And Data Quality Check

- ## Data Introduction :

The data consists of 21 columns and 7999 observations in all attributes are numeric variables.

- ## Variables :

Barium	**dangerous if greater than 2**
aluminium	dangerous if greater than 2.8
chloramium	dangerous if greater than 4
lead	dangerous if greater than0.015
copper	dangerous if greater than 1.3
bacteria	dangerous if greater than 0
viruses	dangerous if greater than 0
nitrates	dangerous if greater than 1
mercury	dangerous if greater than 0.002

***Target Variable :**
is_safe

- **0 - not safe**
- **1 - safe**

Missing Values :

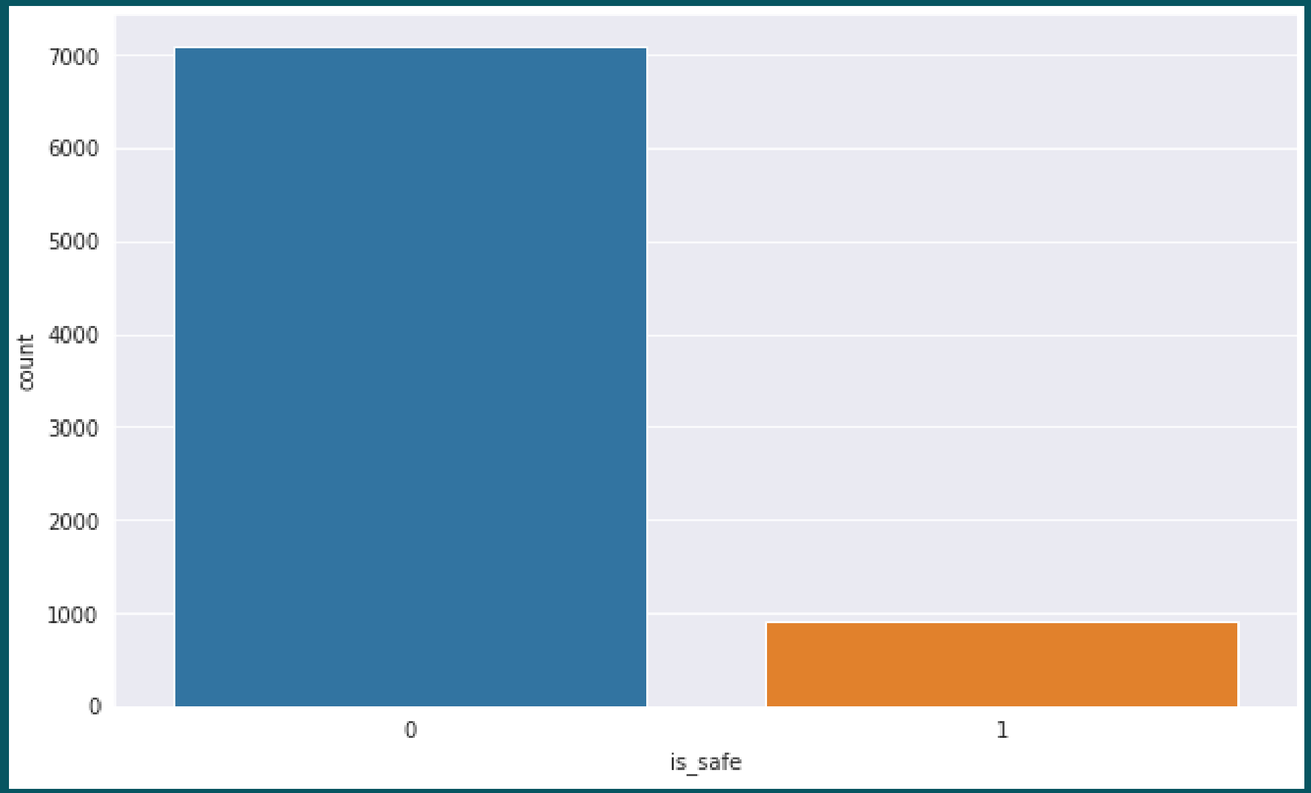
There were 21 Continuous variables. There were 2 variable columns that contained missing values, which were replaced by the mean and mode of the variables.

- is_safe - replaced by mode
- ammonia - replaced by mean

Exploratory Data Analysis

Our dataset consisted of 20 feature columns, these denoted the name of the impurities in our water samples. The countplot below shows us the amount of safe-to-drink water samples compared to the samples that are not safe-to-drink.

Safe to Drink Not Safe to Drink

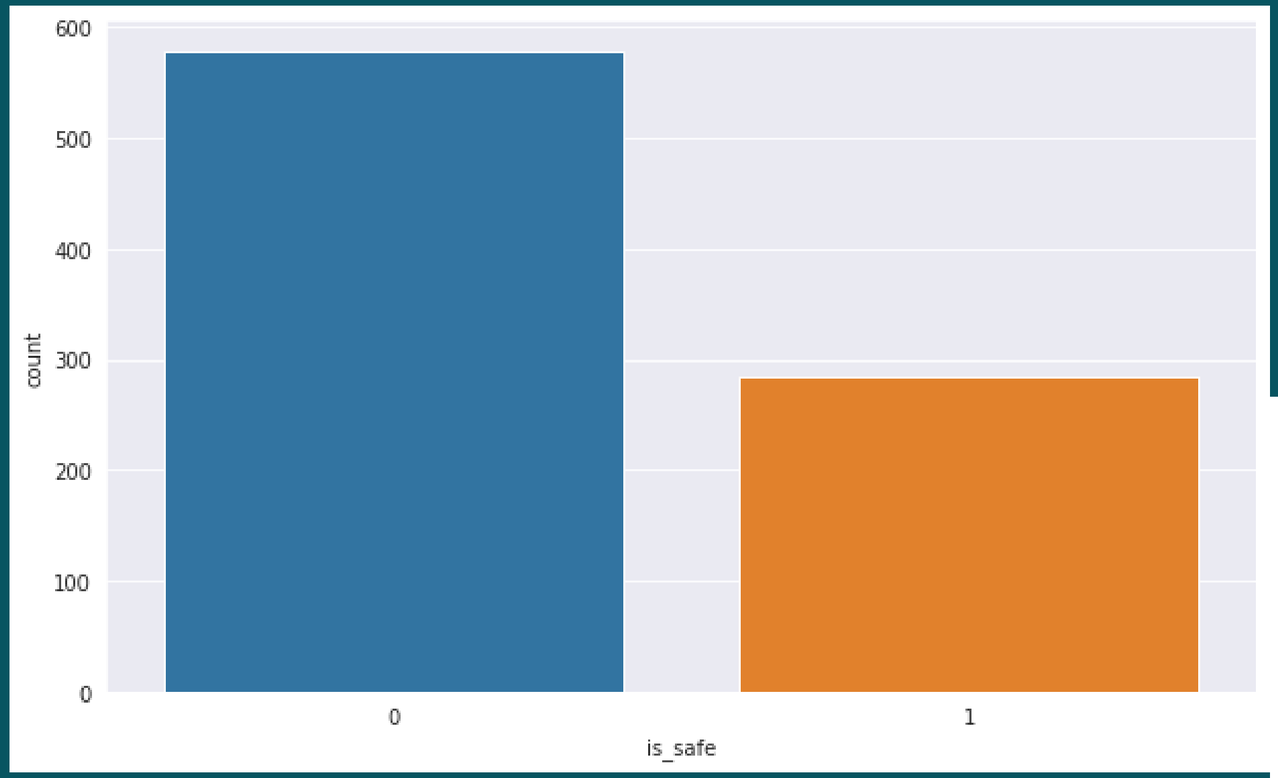


Heatmap


aluminium 1 0.068 0.23 0.29 -0.1 0.37 0.35 0.17 -0.0098 -0.078 -0.071 0.021 -0.0038 0.24 -0.0033 0.36 0.24 -0.0037 0.33 0.015 0.33
ammonia 0.068 1 0.047 0.07 -0.0065 0.11 0.13 0.016 -0.028 0.064 0.11 -0.038 0.0066 -0.064 0.02 0.091 0.05 0.03 0.076 0.015 -0.023
arsenic 0.23 0.047 1 0.36 0.33 0.36 0.31 -0.036 0.0038 0.036 0.012 -0.088 0.028 0.31 -0.015 0.33 0.22 -0.007 0.31 0.0015 -0.12
barium 0.29 0.07 0.36 1 -0.038 0.45 0.42 0.065 -0.019 0.1 -0.0023 -0.043 -0.011 0.31 0.006 0.46 0.29 0.035 0.43 -0.0024 0.091
cadmium -0.1 -0.0065 0.33 -0.038 1 -0.14 -0.16 -0.11 0.0049 -0.092 0.021 -0.035 0.02 -0.016 -0.016 -0.15 -0.099 0.01 -0.16 -0.0056 -0.26
chloramine 0.37 0.11 0.36 0.45 -0.14 1 0.56 0.12 0.0044 0.15 0.0037 -0.03 -0.0016 0.38 -0.021 0.59 0.39 0.011 0.52 -0.0077 0.19
chromium 0.35 0.13 0.31 0.42 -0.16 0.56 1 0.11 -0.0023 0.14 0.0024 -0.051 -0.013 0.34 -0.023 0.52 0.32 0.031 0.51 -0.0055 0.18
copper 0.17 0.016 -0.036 0.065 -0.11 0.12 0.11 1 0.012 0.15 0.0063 0.12 0.0023 0.16 0.018 0.1 0.026 -0.0033 0.089 0.007 0.03
flouride -0.0098 -0.028 0.0038 -0.019 0.0049 0.0044 -0.0023 0.012 1 0.014 0.018 0.012 -0.0081 -0.017 -0.0044 -0.016 0.0077 0.023 0.015 0.017 0.0066
bacteria -0.078 0.064 0.036 0.1 -0.092 0.15 0.14 0.15 0.014 1 0.62 -0.028 -0.034 0.25 -0.0045 0.15 0.099 -0.007 0.15 0.045 -0.022
viruses -0.071 0.11 0.012 -0.0023 0.021 0.0037 0.0024 0.0063 0.018 0.62 1 0.018 -0.045 -0.092 0.011 0.0032 -0.02 -0.035 0.012 0.058 -0.097
lead 0.021 -0.038 -0.088 -0.043 -0.035 -0.03 -0.051 0.12 0.012 -0.028 0.018 1 0.035 -0.052 -0.0078 -0.028 -0.049 0.032 -0.057 -0.0092 -0.01
nitrates -0.0038 0.0066 0.028 -0.011 0.02 -0.0016 -0.013 0.0023 -0.0081 -0.034 -0.045 0.035 1 0.017 -0.02 -0.014 -0.021 0.043 0.0052 0.00082 -0.072
nitrites 0.24 -0.064 0.31 0.31 -0.016 0.38 0.34 0.16 -0.017 0.25 -0.092 -0.052 0.017 1 -0.017 0.35 0.27 0.012 0.33 -0.0099 0.047
mercury -0.0033 0.02 -0.015 0.006 -0.016 -0.021 -0.023 0.018 -0.0044 -0.0045 0.011 -0.0078 -0.02 -0.017 1 0.0061 0.033 0.035 0.0061 0.032 -0.037
perchlorate 0.36 0.091 0.33 0.46 -0.15 0.59 0.52 0.1 -0.016 0.15 0.0032 -0.028 -0.014 0.35 0.0061 1 0.37 0.014 0.5 0.00092 0.076
radium 0.24 0.05 0.22 0.29 -0.099 0.39 0.32 0.026 0.0077 0.099 -0.02 -0.049 -0.021 0.27 0.033 0.37 1 0.028 0.35 0.02 0.065
selenium -0.0037 0.03 -0.007 0.035 0.01 0.011 0.031 -0.0033 0.023 -0.007 -0.035 0.032 0.043 0.012 0.035 0.014 0.028 1 -0.02 -0.022 -0.031
silver 0.33 0.076 0.31 0.43 -0.16 0.52 0.51 0.089 0.015 0.15 0.012 -0.057 0.0052 0.33 0.0061 0.5 0.35 -0.02 1 0.0082 0.1
uranium 0.015 0.015 0.0015 -0.0024 -0.0056 -0.0077 -0.0055 0.007 0.017 0.045 0.058 -0.0092 0.00082 -0.0099 0.032 0.00092 0.02 -0.022 0.0082 1 -0.076
is_safe 0.33 -0.023 -0.12 0.091 -0.26 0.19 0.18 0.03 0.0066 -0.022 -0.097 -0.01 -0.072 0.047 -0.037 0.076 0.065 -0.031 0.1 -0.076 1
aluminium ammonia arsenic barium cadmium chloramine chromium copper flouride bacteria viruses lead nitrates nitrites mercury perchlorate radium selenium silver uranium is_safe
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00

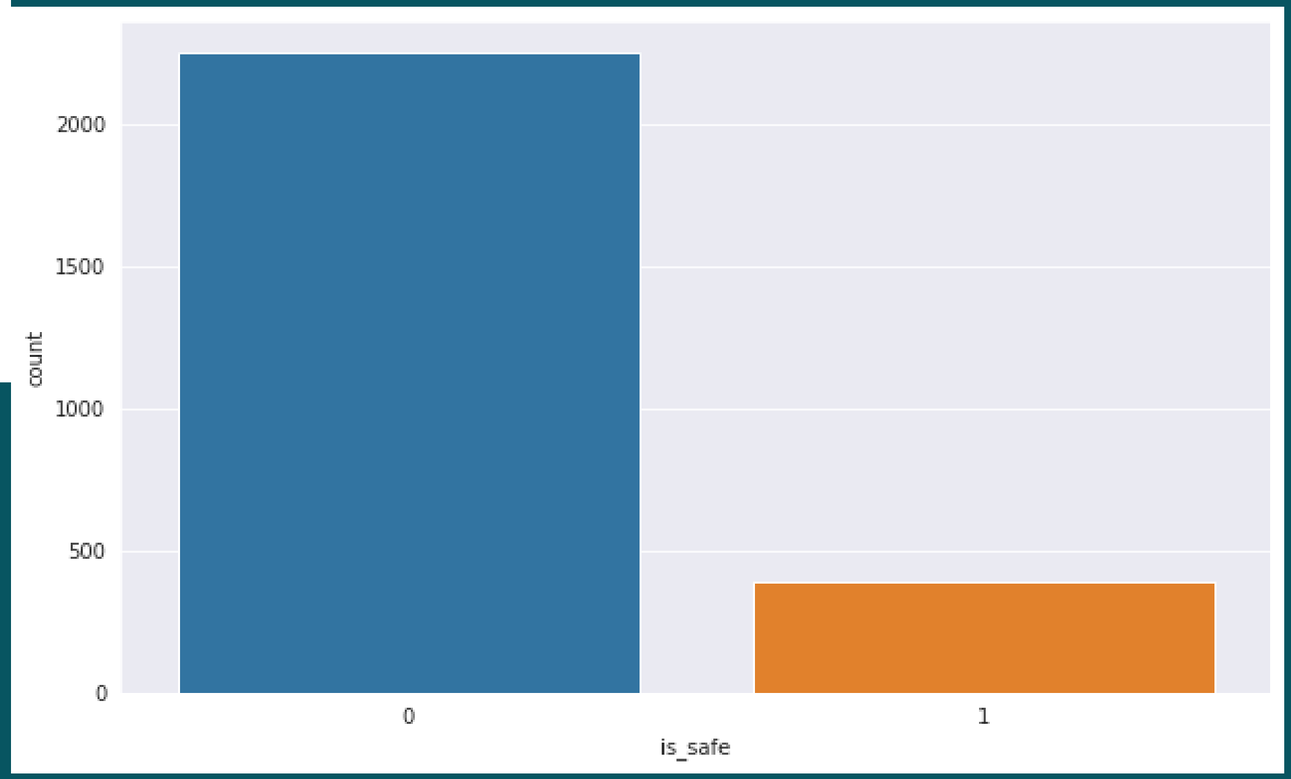
Threshold Countplots

Aluminium

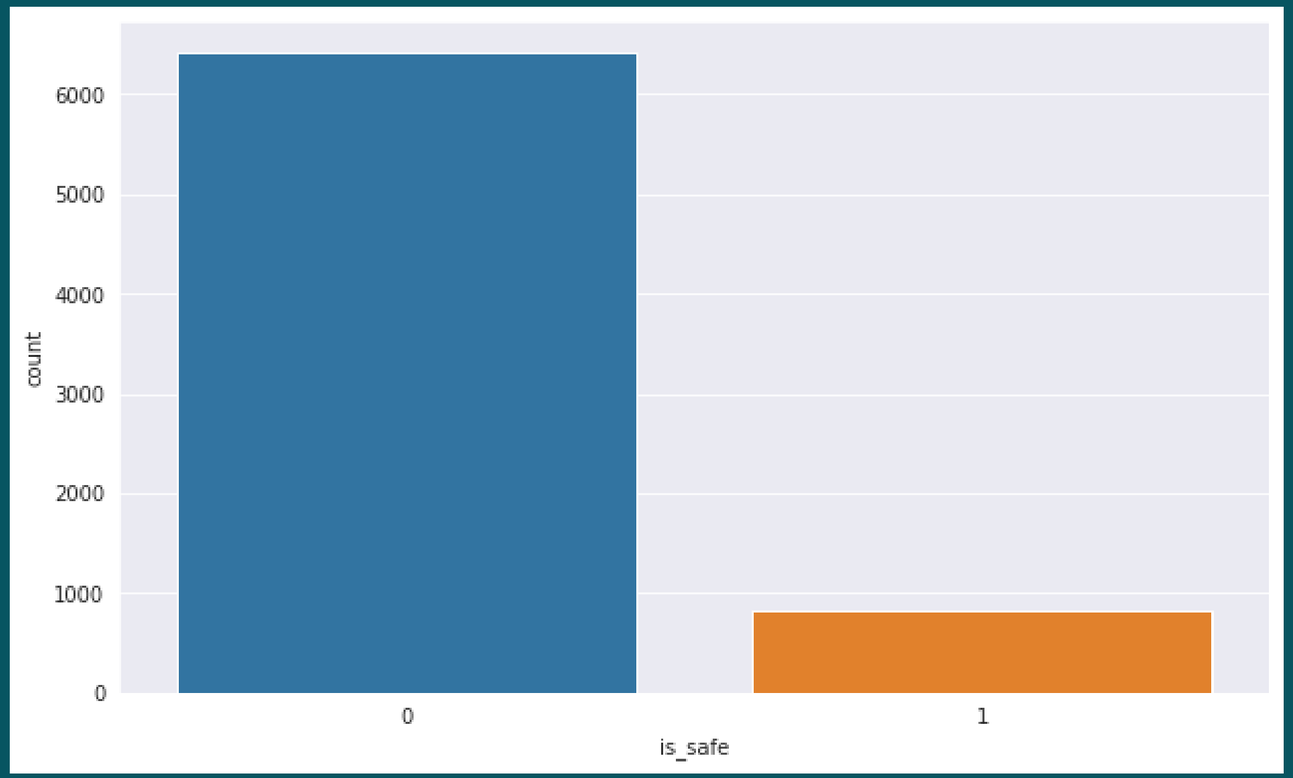


Here we can observe the amount of water samples that had dangerous levels of a particular impurant in more depth. For example, around 290 water samples had dangerous levels of aluminium that made them unsafe to drink.

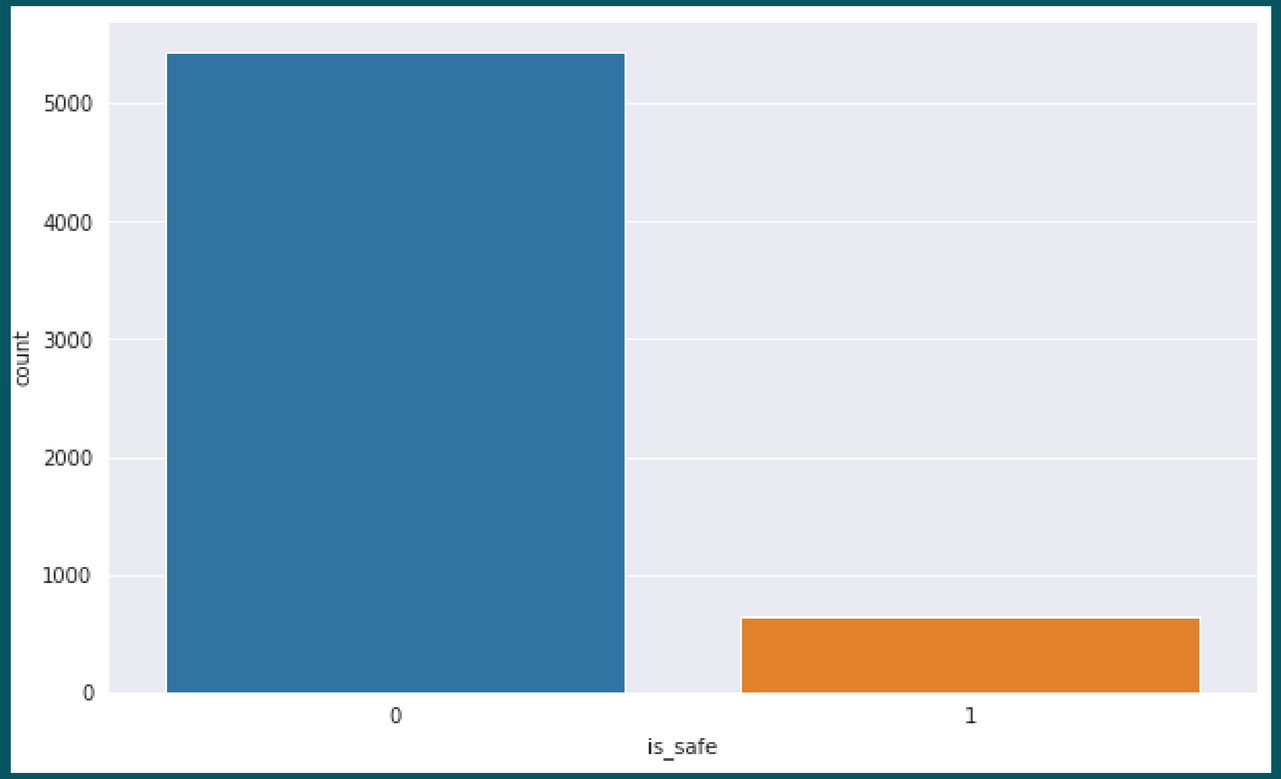
Barium



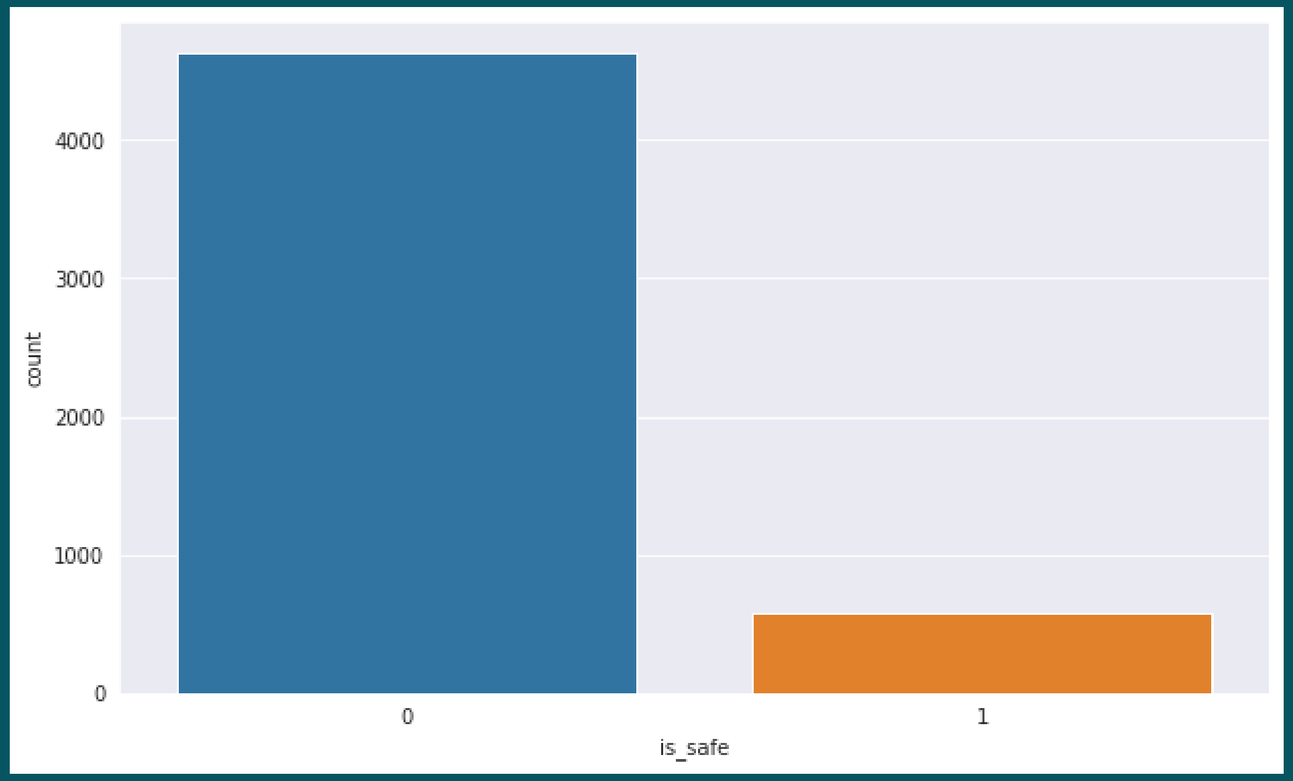
Lead



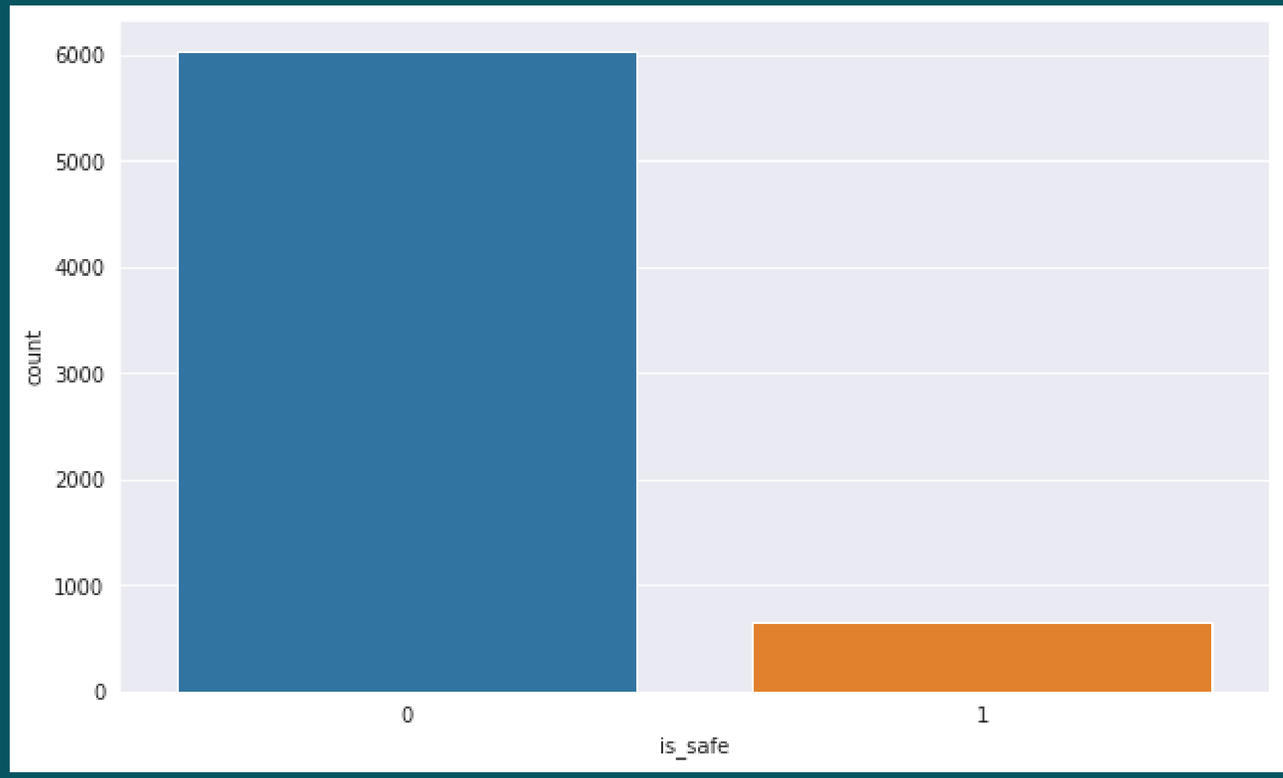
Mercury



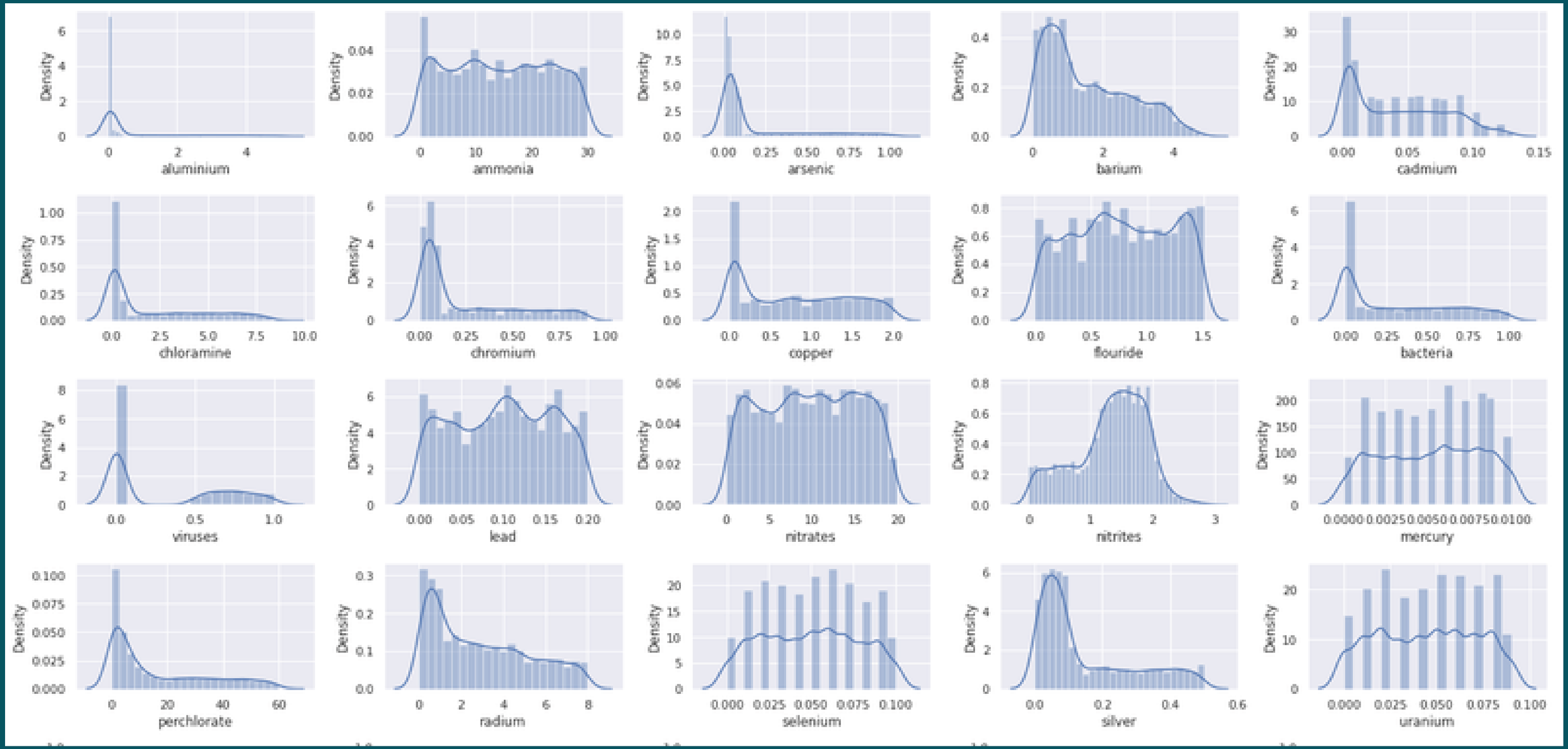
Bacteria



Viruses

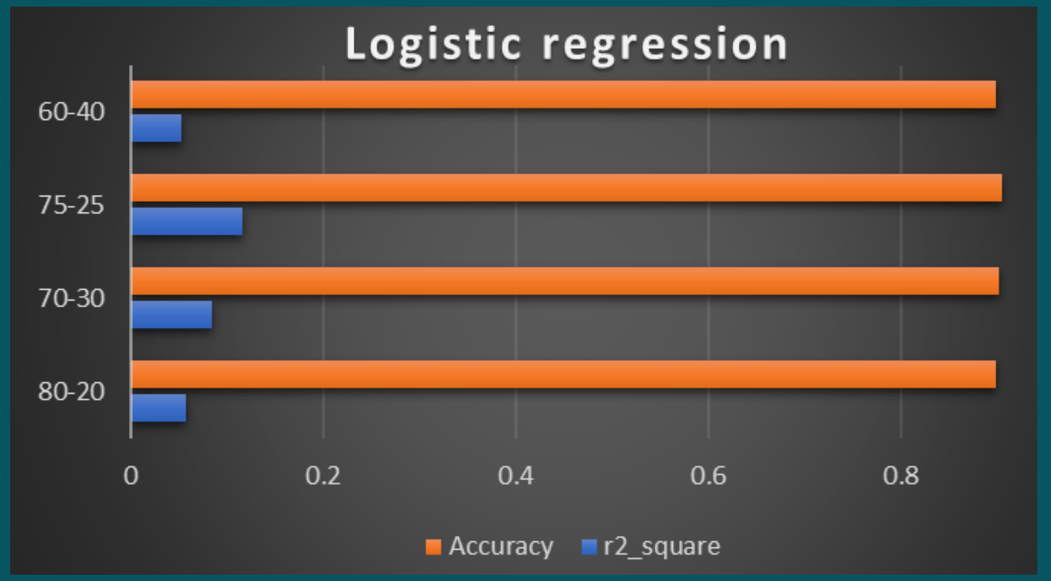


Impurity Distribution Plots



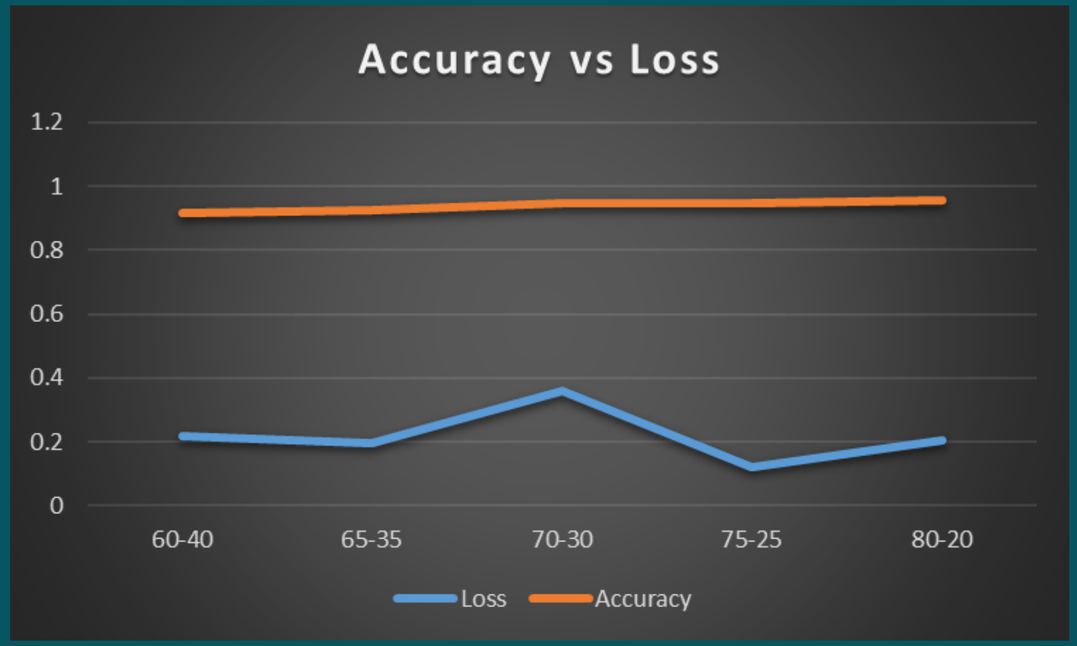
Logistic Regression

Train-Test	r2_square	Accuracy
80-20	0.057142	0.8986875
70-30	0.08372	0.902083
75-25	0.11622	0.905
60-40	0.05268	0.8975



Artificial Neural Network

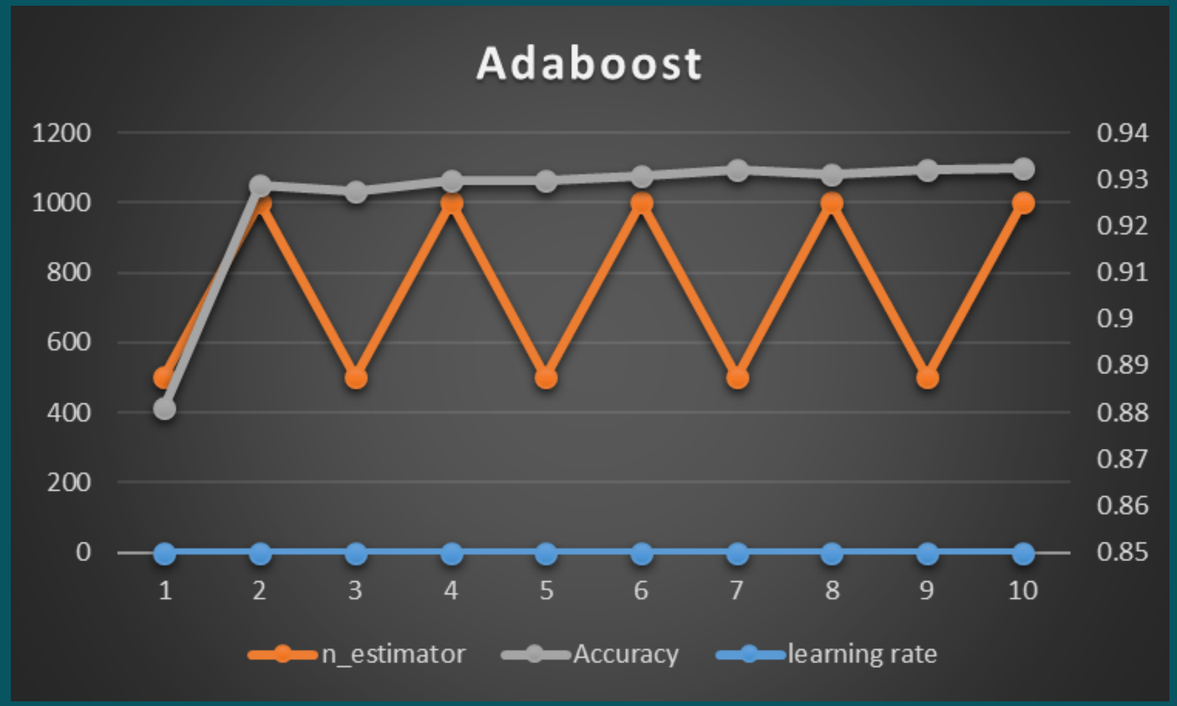
Train test Split	Architecture	Optimizer	Epochs	loss	Accuracy
60-40	21-12-6-4-2-1	SGD	400	0.2231	0.9114
60-40	21-12-6-4-2-1	Adam	200	0.342	0.8923
60-40	21-14-8-4-2-1	SGD	200	0.2169	0.9169
65-35	10-3-2-1	Adam	200	0.344	0.8913
65-35	21-17-13-9-4-1	SGD	200	0.2038	0.7929
65-35	21-14-8-4-1	SGD	200	0.197	0.924
70-30	20-12-8-4-2-1	Adam	200	0.2342	0.94
70-30	20-15-10-6-3-1	Adam	700	0.36	0.9479
70-30	20-12-8-4-2-1	Adam	1000	0.339	0.9438
75-25	21-12-6-4-2-1	Adam	200	0.3493	0.888
75-25	20-13-7-3-1	SGD	**550**	0.1521	0.9432
75-25	22-17-13-9-4-2-1	Adam	200	0.1207	0.9467
80-20	20-14-7-1	Adam	500	0.307	0.9419
80-20	20-12-6-3-1	Adam	700	0.3287	0.9406
80-20	20-16-12-8-4-2-1	Adam	500	0.2059	0.958125



Boosting :

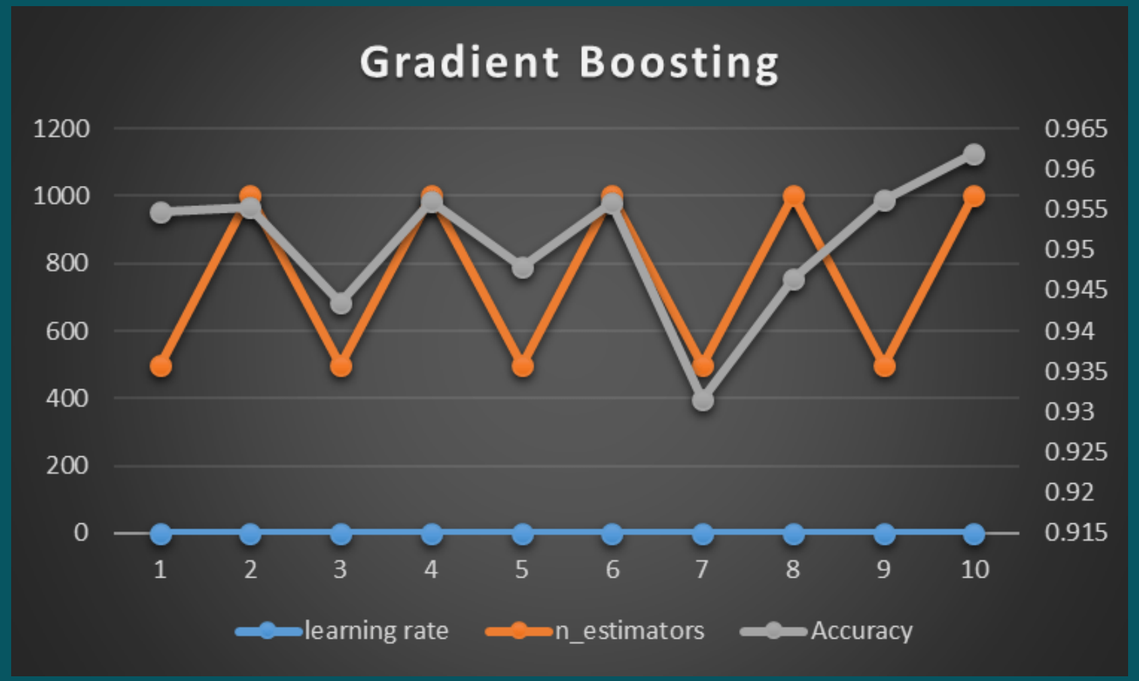
Adaboost :

Train Test	learning rate	n_estimator	Accuracy
60-40	0.01	500	0.880938
60-40	0.1	1000	0.92875
65-35	0.1	500	0.9275
65-35	0.3	1000	0.929642857
70-30	0.1	500	0.929583
70-30	0.3	1000	0.930833
75-25	0.1	500	0.932
75-25	0.3	1000	0.931
80-20	0.1	500	0.931875
80-20	0.3	1000	0.9325



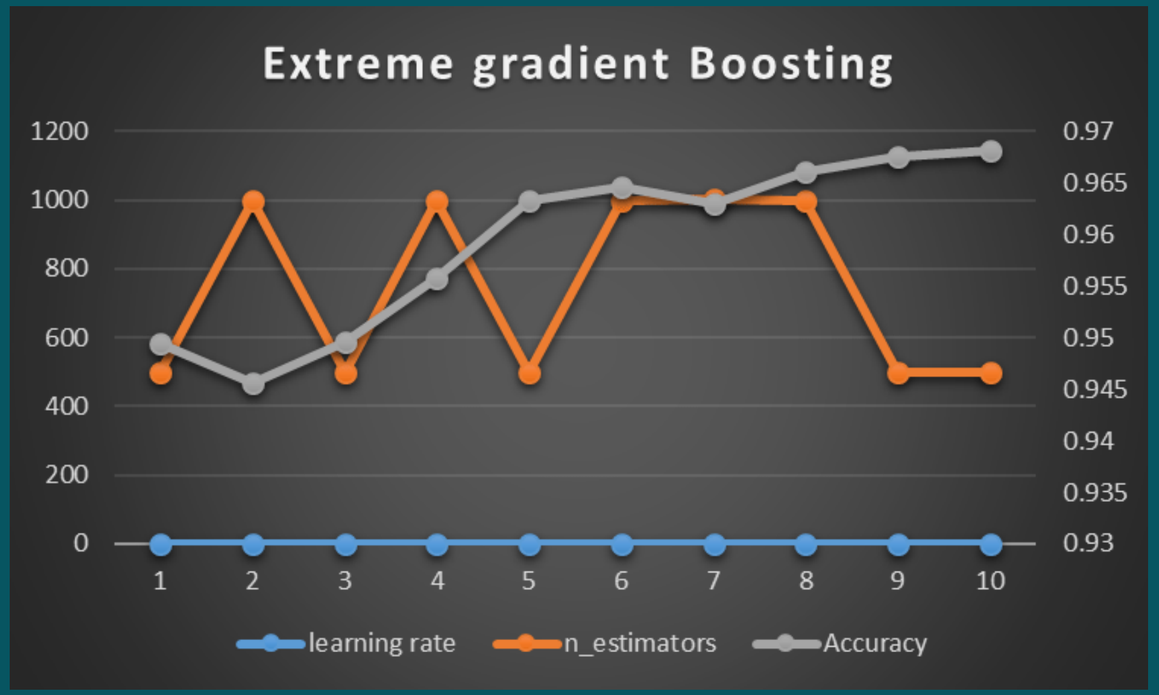
Gradient Boost :

Train Test	learning rate	n_estimators	Accuracy
60-40	0.1	500	0.954688
60-40	0.3	1000	0.955313
65-35	0.01	500	0.943571
65-35	0.3	1000	0.956071
70-30	0.01	500	0.947917
70-30	0.3	1000	0.955833
75-25	0.3	500	0.9315
75-25	0.01	1000	0.9465
80-20	0.1	500	0.95625
80-20	0.3	1000	0.961875



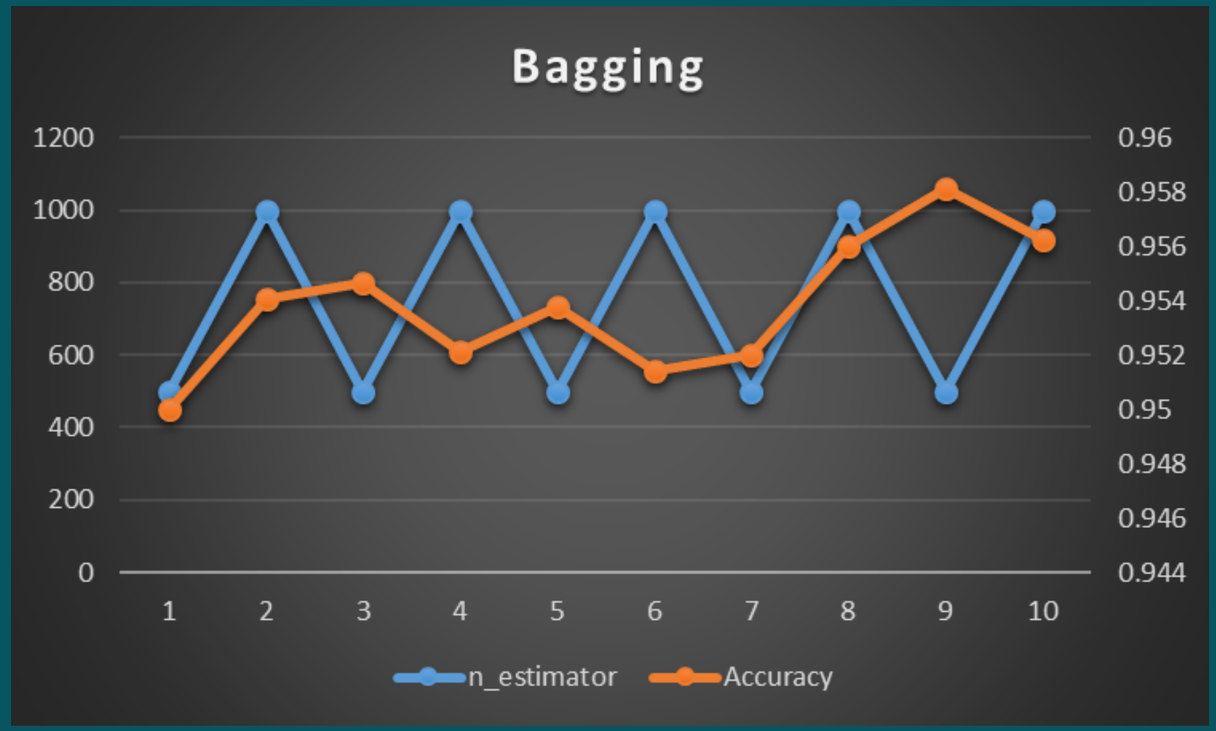
Extreme Gradient Boost(XGB) :

Train Test	learning rate	n_estimators	Accuracy
60-40	0.01	500	0.949375
60-40	0.01	1000	0.945625
65-35	0.01	500	0.949643
65-35	0.01	1000	0.955714
70-30	0.3	500	0.96333
70-30	0.1	1000	0.964583
75-25	0.3	1000	0.963
75-25	0.1	1000	0.966
80-20	0.1	500	0.9675
80-20	0.3	500	0.968125



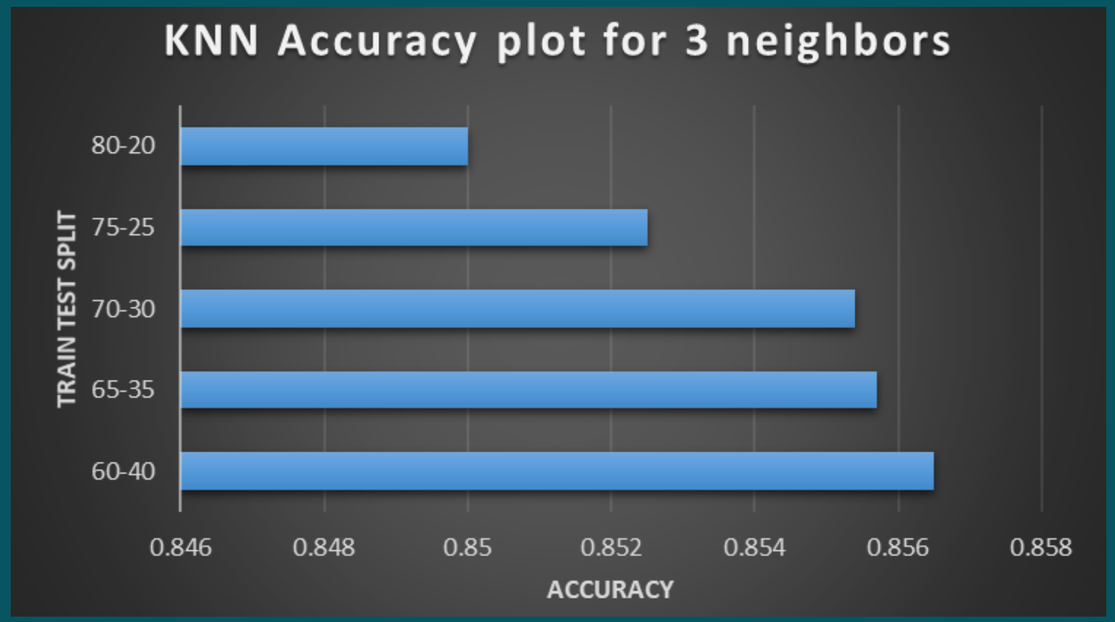
Bagging :

Train Test	n_estimator	Accuracy
60-40	500	0.95
60-40	1000	0.954063
65-35	500	0.954643
65-35	1000	0.952143
70-30	500	0.95375
70-30	1000	0.951429
75-25	500	0.952
75-25	1000	0.956
80-20	500	0.958125
80-20	1000	0.95625



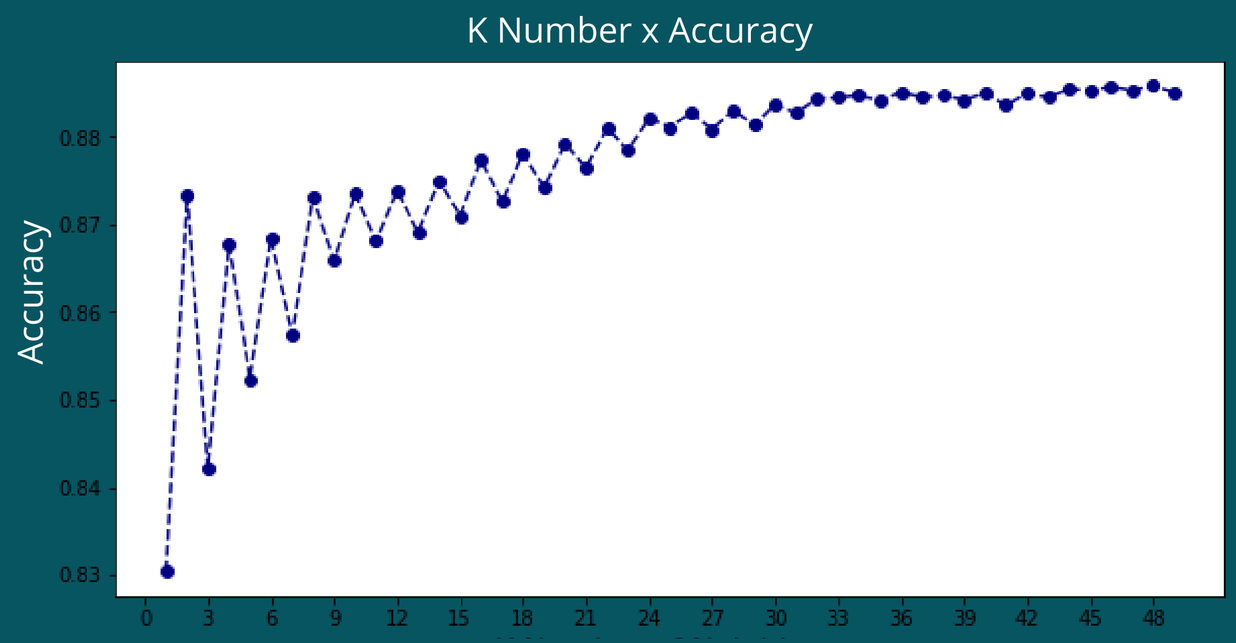
KNN

Train_test_split	Accuracy	Neighbors
60-40	0.8565	3
65-35	0.8557	3
70-30	0.8554	3
75-25	0.8525	3
80-20	0.85	3



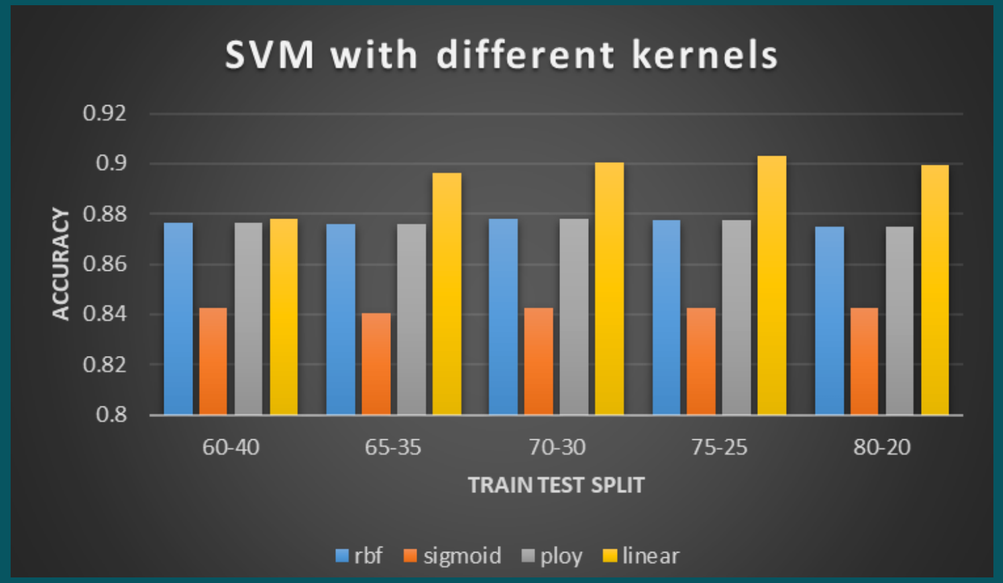
KNN using K Neighbors

- when we implemented KNN for K neighbors we got the same accuracy for all train test splits with best accuracy as **0.8858** and with K=48 for all train test splits



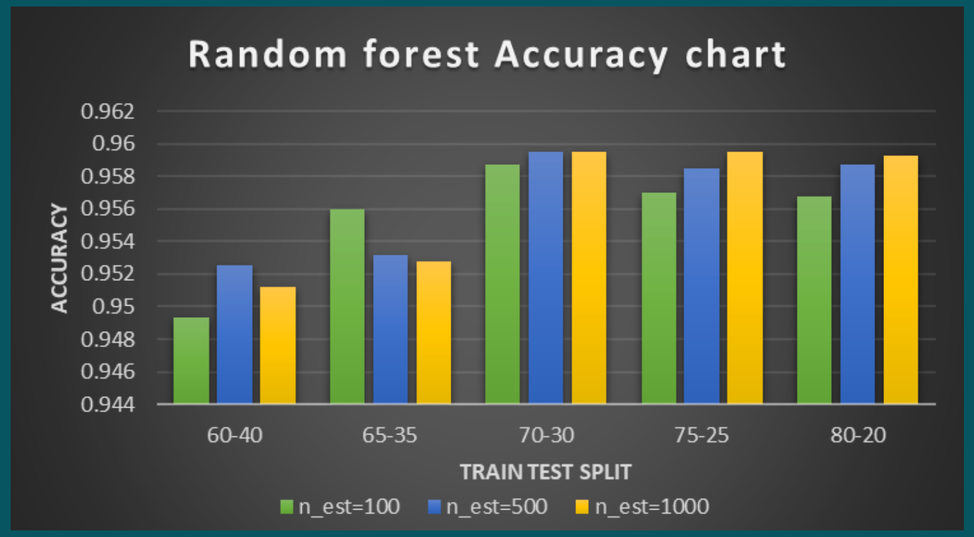
SVM

Kernel(accuracy)/ Train test split	rbf	sigmoid	poly	linear
60-40	0.8765	0.8428	0.8765	0.8778
65-35	0.876	0.8407	0.876	0.8964
70-30	0.8783	0.8425	0.8783	0.90041
75-25	0.8775	0.8425	0.8775	0.903
80-20	0.875	0.8425	0.875	0.8993



Random Forest

Accuracy/Train test split	n_est=100	n_est=500	n_est=1000
60-40	0.9493	0.9525	0.9512
65-35	0.956	0.9532	0.9528
70-30	0.9587	0.9595	0.9595
75-25	0.957	0.9585	0.9595
80-20	0.9568	0.9587	0.9593

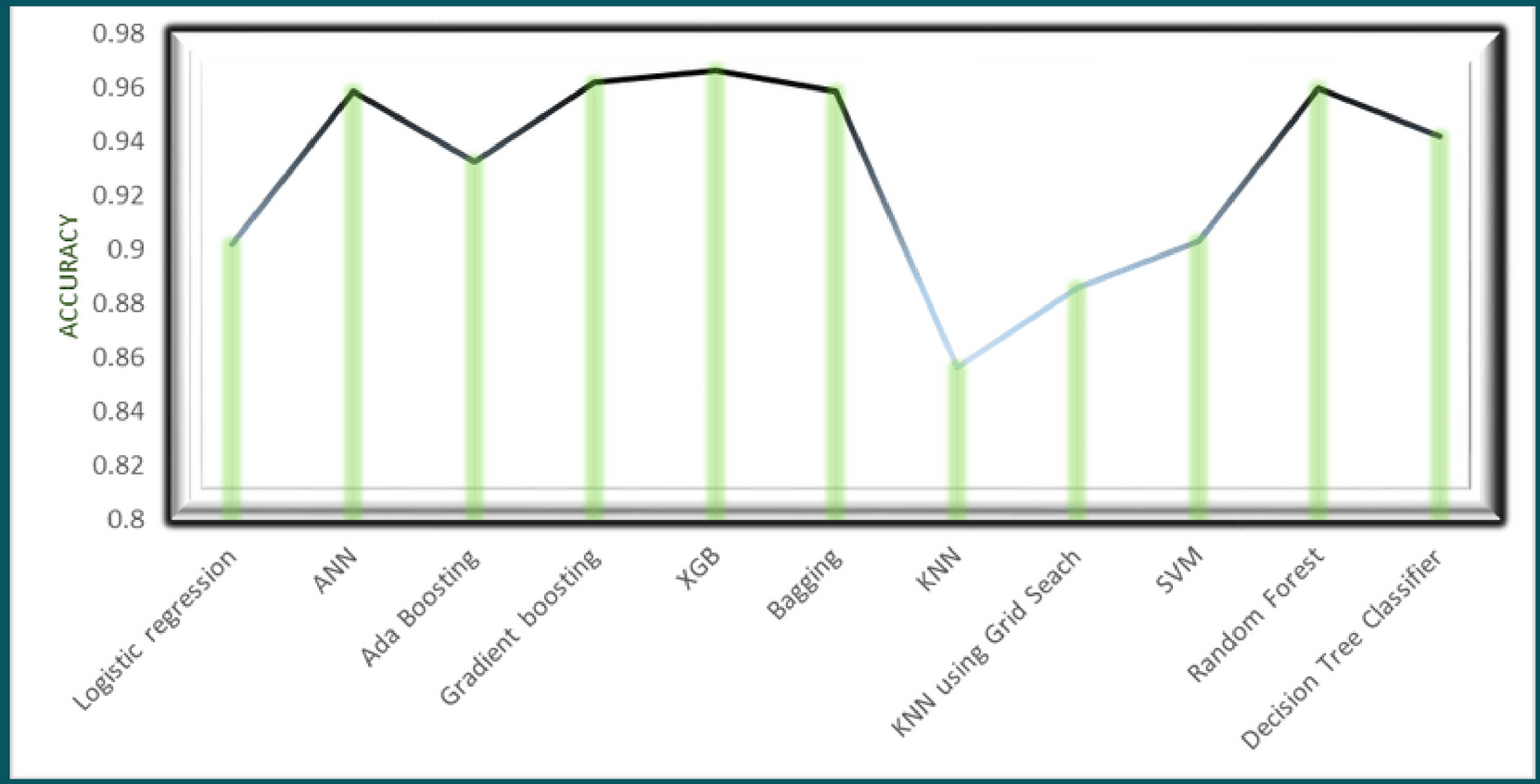


Decision Tree Classifier

The best accuracy we got was 0.9417 of train test split 80-20

Comparision of all implemented ML algorithms

Logistic regression	0.902
ANN	0.9581
Ada Boosting	0.9325
Gradient Boosting	0.9618
XGB	0.9661
Bagging	0.9581
KNN	0.8565
KNN using Grid Search	0.8858
SVM	0.903
Random Forest	0.9595
Decision Tree Classifier	0.9417



Conclusion

- Our EDA helped us determine that lead was the main impurity in most of the water samples that were unsafe to drink, so lead is the biggest contaminant of water.
- Uranium,Copper and Nitrates were below dangerous levels in all of the water samples.
- Viruses and Bacteria were among the top impurities that made water unsafe to drink.
- Aluminium was among the rarer impurities that made our samples unsafe to drink.
- The best accuracy we got in most of the algoritms was of 80-20 train test split.
- The best performing optimizer in ANN for our dataset is Adam.
- After implementation of all the ML algorithms we have got the best accuracy for XGB model.
- After implementing all the machine learning algorithms we got the best accuracy for XGB that is 0.9661.
- In Ensemble learning Algorithms we got better accuracy for all the train test split using learning rate as 0.3.
- In SVM the best choice of kernel for our dataset is linear.
- In KNN as the number of neighbors increased the accuracy too increased.

Conclusion

- Our EDA helped us determine that lead was the main impurity in most of the water samples that were unsafe to drink, so lead is the biggest contaminant of water.
- Uranium,Copper and Nitrates were below dangerous levels in all of the water samples.
- Viruses and Bacteria were among the top impurities that made water unsafe to drink.
- Aluminium was among the rarer impurities that made our samples unsafe to drink.
- The best accuracy we got in most of the algoritms was of 80-20 train test split.
- The best performing optimizer in ANN for our dataset is Adam.
- After implementation of all the ML algorithms we have got the best accuracy for XGB model.
- After implementing all the machine learning algorithms we got the best accuracy for XGB that is 0.9661.
- In Ensemble learning Algorithms we got better accuracy for all the train test split using learning rate as 0.3.
- In SVM the best choice of kernel for our dataset is linear.
- In KNN as the number of neighbors increased the accuracy too increased.

Presented by

D. Sai Rizwana

S.R. Bhargavi

Shubham Singh

Rohit Menon

Sai Mohan

Appendix

Training And Testing :

```
x= data.drop("is_safe", axis=1)
y=data["is_safe"]
x.head
```

```
y.head
```

```
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.2, random_state=42)
y_test
```

Logistic Regression :

```
logreg = LogisticRegression(C=1e9)
logreg.fit(x_train,y_train)
```

```
y_pred = logreg.predict(x_test)
y_pred
```

```
r2_score(y_test,y_pred)
```

```
accuracy_score(y_test,y_pred)
```

Neural Networks :

```
tf.random.set_seed(45)

# STEP1: Creating the model

model= tf.keras.Sequential([
                            tf.keras.layers.Dense(20, activation='relu'),
                            tf.keras.layers.Dense(13, activation='relu'),
                            tf.keras.layers.Dense(5, activation='relu'),
                            tf.keras.layers.Dense(3, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP2: Compiling the model

model.compile(loss= tf.keras.losses.binary_crossentropy,
              optimizer= tf.keras.optimizers.Adam(lr=0.01),
              metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                        tf.keras.metrics.Precision(name='precision'),
                        tf.keras.metrics.Recall(name='a=recall')
              ]
              )

# STEP1: Fit the model

history= model.fit(x_train, y_train, epochs= 550)

pd.DataFrame(history.history).plot();
```

Bagging :

```
#bagging
bag_model = BaggingClassifier(
base_estimator=BaggingClassifier(),
n_estimators=500,
max_samples=0.5,
bootstrap=True,
oob_score=True,
random_state=42
)
l=bag_model.fit(x_train, y_train)
l.score(x_test,y_test)
```

Ada Boosting :

```
#Ada Boosting
adaclf = AdaBoostClassifier(
    n_estimators=1000,
    learning_rate=0.1,
    random_state=42)
l=adaclf.fit(x_train, y_train)
y_pred_1 = adaclf.predict(x_test)
print("Accuracy:",adaclf.score(x_test,y_test))
```

Gradient Boosting :

```
#gradientBoosting
GradientBoostingClassifier,
Classifier = GradientBoostingClassifier(
    max_depth=2,
    n_estimators=1000,
    learning_rate=0.1,
    random_state=42
)
l=Classifier.fit(x_train, y_train)
l.score(x_test,y_test)
```

Extreme Gradient Boosting :

```
#ExtremeGradientBoosting
egb =XGBClassifier(
    n_estimators=1000,
    learning_rate=0.1,
    random_state=42)
l=egb.fit(x_train, y_train)
y_pred_1 = egb.predict(x_test)
l.score(x_test,y_test)
```

Random Forest :

```
from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier()
model.fit(x_train,y_train)
model=RandomForestClassifier(n_estimators=100)
c=model.fit(x_train,y_train)
y_pred=c.predict(x_test)
print('Accuracy of model is',metrics.accuracy_score(y_test,y_pred))
```

Decision Tree :

```
from sklearn.tree import DecisionTreeClassifier
# instantiate the DecisionTreeClassifier model with criterion gini index

clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)



# fit the model
clf_gini.fit(x_train, y_train)
y_pred_gini = clf_gini.predict(x_test)
from sklearn.metrics import accuracy_score

print('Model accuracy score with criterion gini index: {0:0.4f}'. format(accuracy_score(y_test, y_pred_gini)))
y_pred_train_gini = clf_gini.predict(x_train)

y_pred_train_gini
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train_gini)))
```

SVM :

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import svm
svc=svm.SVC(kernel='linear')
model=svc.fit(x_train,y_train)
y_pred=model.predict(x_test)
y_pred
print('Accuracy of model is',metrics.accuracy_score(y_test,y_pred))
```

X

KNN :

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train,y_train)

y_pred = knn.predict(x_test)
y_pred
print('Accuracy',metrics.accuracy_score(y_test,y_pred))
```

```
k_list = list(range(1,50))
k_values = dict(n_neighbors=k_list)
print(k_values.keys()),
print(k_values.values())
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(knn, k_values, cv=5, scoring='accuracy')
grid
grid.fit(data.drop('is_safe', axis=1), data.is_safe)
grid.cv_results_
for key in grid.cv_results_.keys():
  print(key)
grid_table = pd.DataFrame(grid.cv_results_)
grid_table.head()
grid_table_rank = grid_table[['params','mean_test_score','std_test_score','rank_test_score']].loc[grid_table['rank_test_score']==1].sort_values(by='std_test_score', ascending=True)
grid_table_rank
```

```
print("The best value of k = {} with {} of accuracy.".format(grid.best_params_,grid.best_score_))
print("The best classifier is: {}".format(grid.best_estimator_))
```

Graph using gridsearch with KNN :

```
graphic = grid.cv_results_['mean_test_score']
graphic

plt.figure(figsize=(10,5))
plt.plot(k_list,graphic,color='navy',linestyle='dashed',marker='o')
plt.xlabel('K Number of Neighbors', fontdict={'fontsize': 15})
plt.ylabel('Accuracy', fontdict={'fontsize': 15})
plt.title('K NUMBER X ACCURACY', fontdict={'fontsize': 30})
plt.xticks(range(0,50,3),)
#plt.xaxis.set_major_locator(MultipleLocator(3))
plt.show()
```