GROUP -11

WATER QUALITY PREDICTION

BHAVAN'S VIVEKANANDA COLLEGE

K.Shiva Kumar Reddy

M. Sai Dheeraj

PV.Tilak

P. Manikethan Reddy

BACKGROUND -

A survey was conducted on different water samples to check their water quality with attributes such as ammonia,barium,lead,silver etc.

OBJECTIVE -

On the basis of these various factors, Our objective is to predict whether the water is safe or not

PATH -

Team followed a Standard machine learning algorithm development process

DATA SET

	aluminium	ammonia	arsenic	barium	cadmium	chloramine	chromium	copper	flouride	bacteria	...	lead	nitrates	nitrites	mercury	perchlorate	radium	selenium	silver	urani
0	1.65	9.08	0.04	2.85	0.007	0.35	0.83	0.17	0.05	0.20	...	0.054	16.08	1.13	0.007	37.75	6.78	0.08	0.34	0.
1	2.32	21.16	0.01	3.31	0.002	5.28	0.68	0.66	0.90	0.65	...	0.100	2.01	1.93	0.003	32.26	3.21	0.08	0.27	0.
2	1.01	14.02	0.04	0.58	0.008	4.24	0.53	0.02	0.99	0.05	...	0.078	14.16	1.11	0.006	50.28	7.07	0.07	0.44	0.
3	1.36	11.33	0.04	2.96	0.001	7.23	0.03	1.66	1.08	0.71	...	0.016	1.41	1.29	0.004	9.12	1.72	0.02	0.45	0.
4	0.92	24.33	0.03	0.20	0.006	2.67	0.69	0.57	0.61	0.13	...	0.117	6.74	1.11	0.003	16.90	2.41	0.02	0.06	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7991	0.05	7.78	0.00	1.95	0.040	0.10	0.03	0.03	1.37	0.00	...	0.197	14.29	1.00	0.005	3.57	2.13	0.09	0.06	0.
7992	0.05	24.22	0.02	0.59	0.010	0.45	0.02	0.02	1.48	0.00	...	0.031	10.27	1.00	0.001	1.48	1.11	0.09	0.10	0.
7993	0.09	6.85	0.00	0.61	0.030	0.05	0.05	0.02	0.91	0.00	...	0.182	15.92	1.00	0.000	1.35	4.84	0.00	0.04	0.
7994	0.01	10.00	0.01	2.00	0.000	2.00	0.00	0.09	0.00	0.00	...	0.000	0.00	0.00	0.000	0.00	0.00	0.00	0.00	0.
7995	0.04	6.85	0.01	0.70	0.030	0.05	0.01	0.03	1.00	0.00	...	0.182	15.92	1.00	0.000	1.35	4.84	0.00	0.04	0.

7996 rows × 21 columns

Rows -7996

Columns-21

Variable description

Target variable - **Is_safe**

0 - {Not safe}

1 - {Safe}

All attributes are numeric variables and they are listed bellow:

- aluminium - (dangerous if greater than 2.8)
- ammonia - (dangerous if greater than 32.5)
- arsenic - (dangerous if greater than 0.01)
- barium - (dangerous if greater than 2)
- cadmium - (dangerous if greater than 0.005)
- chloramine - (dangerous if greater than 4)
- chromium - (dangerous if greater than 0.1)
- copper - (dangerous if greater than 1.3)
- flouride - (dangerous if greater than 1.5)
- bacteria - (dangerous if greater than 0)
- viruses - (dangerous if greater than 0)
- lead - (dangerous if greater than 0.015)
- nitrates - (dangerous if greater than 10)
- nitrites - (dangerous if greater than 1)
- mercury - (dangerous if greater than 0.002)
- perchlorate - (dangerous if greater than 56)
- radium - (dangerous if greater than 5)
- selenium - (dangerous if greater than 0.5)
- silver - (dangerous if greater than 0.1)
- uranium - (dangerous if greater than 0.3)

is_safe - class attribute {0 - not safe, 1 - safe}

Data Cleaning

We found null values in

- Ammonia
- Is_safe

Before

```
np.sum(a.isna())

aluminium      0
ammonia        3
arsenic        0
barium         0
cadmium        0
chloramine     0
chromium       0
copper         0
flouride       0
bacteria       0
viruses        0
lead           0
nitrates       0
nitrites       0
mercury        0
perchlorate    0
radium         0
selenium       0
silver         0
uranium        0
is_safe        3
dtype: int64
```

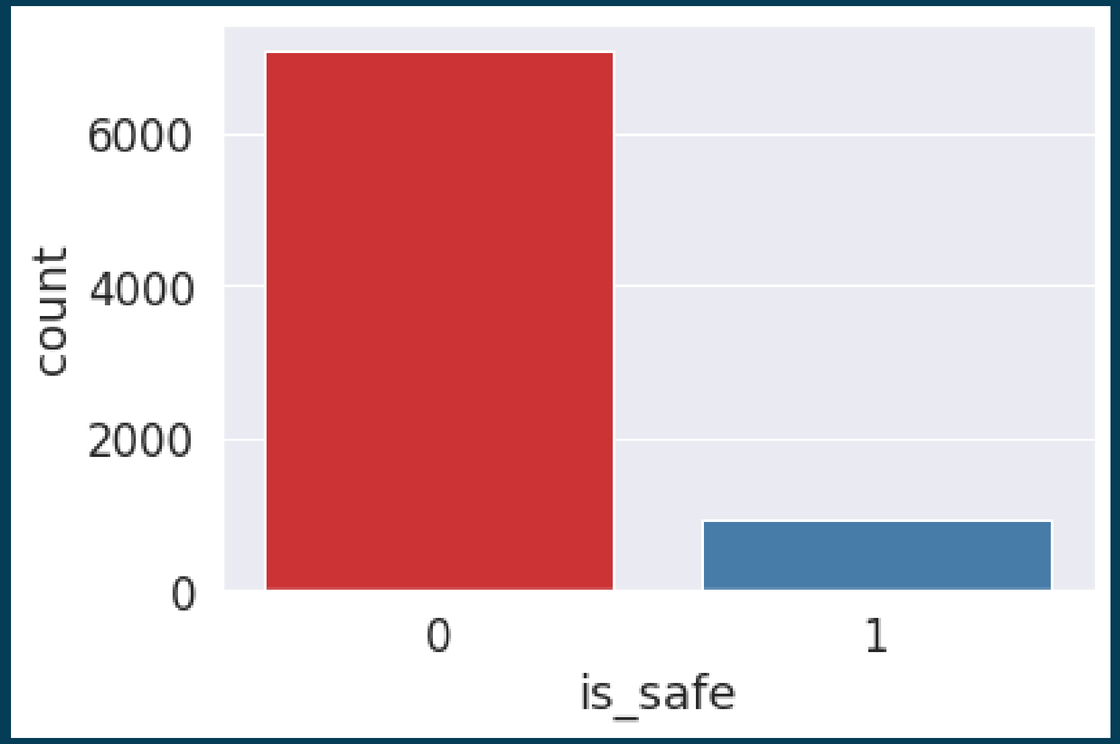
After

```
data.isnull().sum()

aluminium      0
ammonia        0
arsenic        0
barium         0
cadmium        0
chloramine     0
chromium       0
copper         0
flouride       0
bacteria       0
viruses        0
lead           0
nitrates       0
nitrites       0
mercury        0
perchlorate    0
radium         0
selenium       0
silver         0
uranium        0
is_safe        0
dtype: int64
```

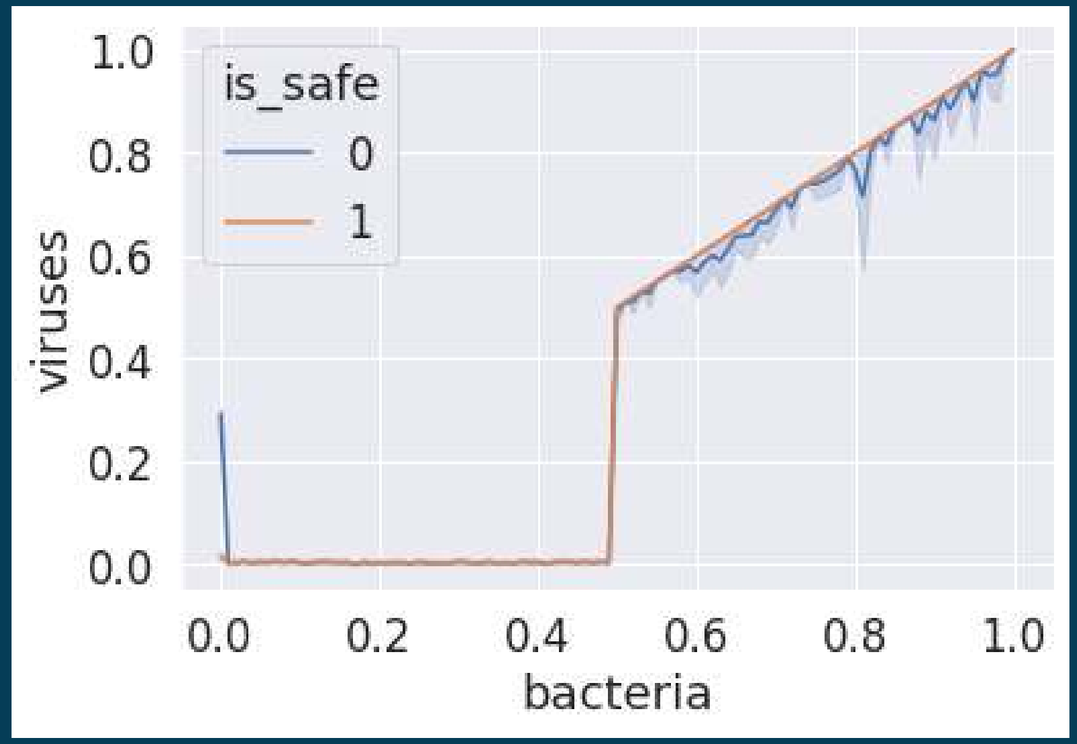
EXPLORATORY DATA ANALYSIS

Showing the count of water samples

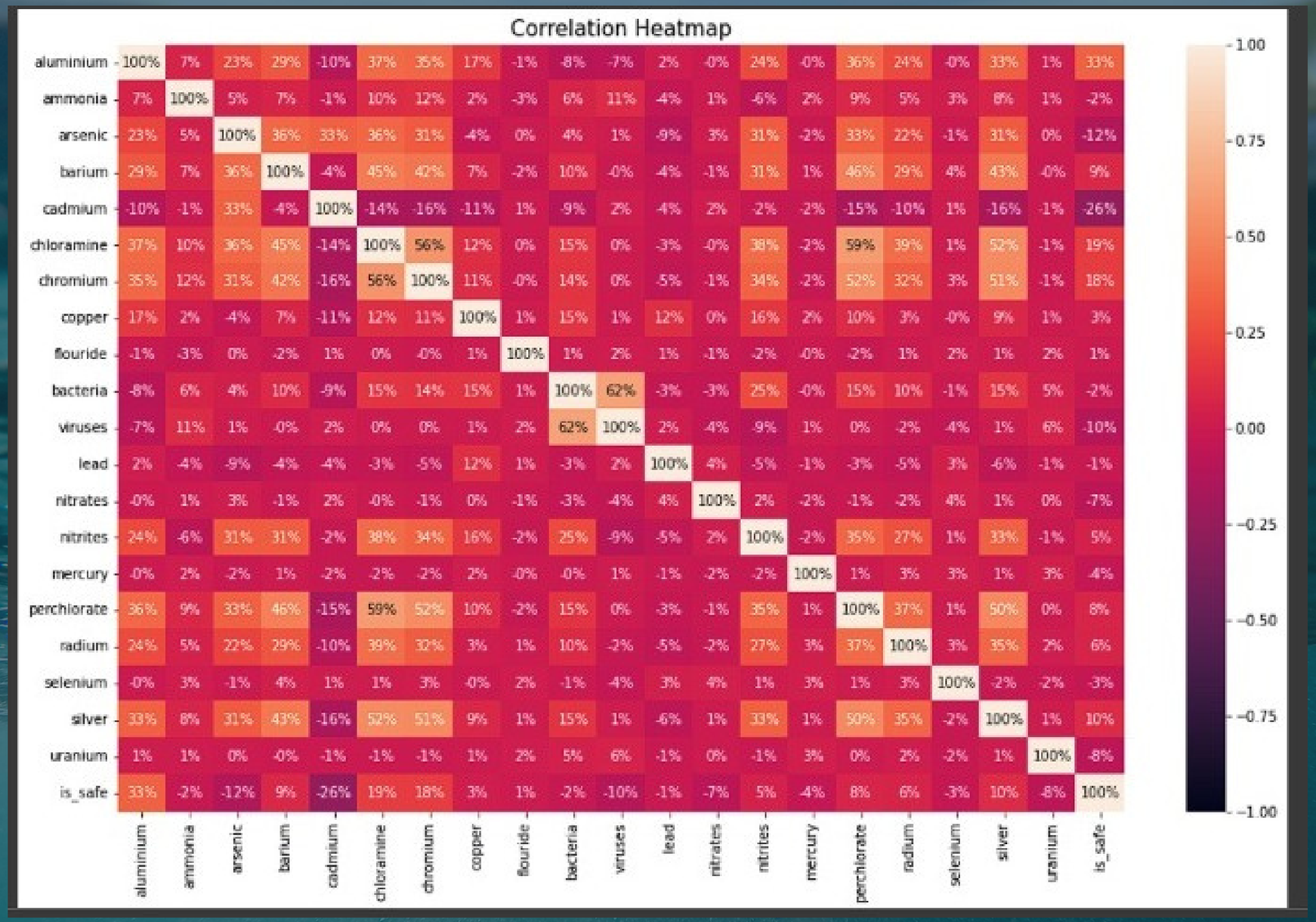


Majority of water samples is not safe

Finding the relationship between bacteria and viruses

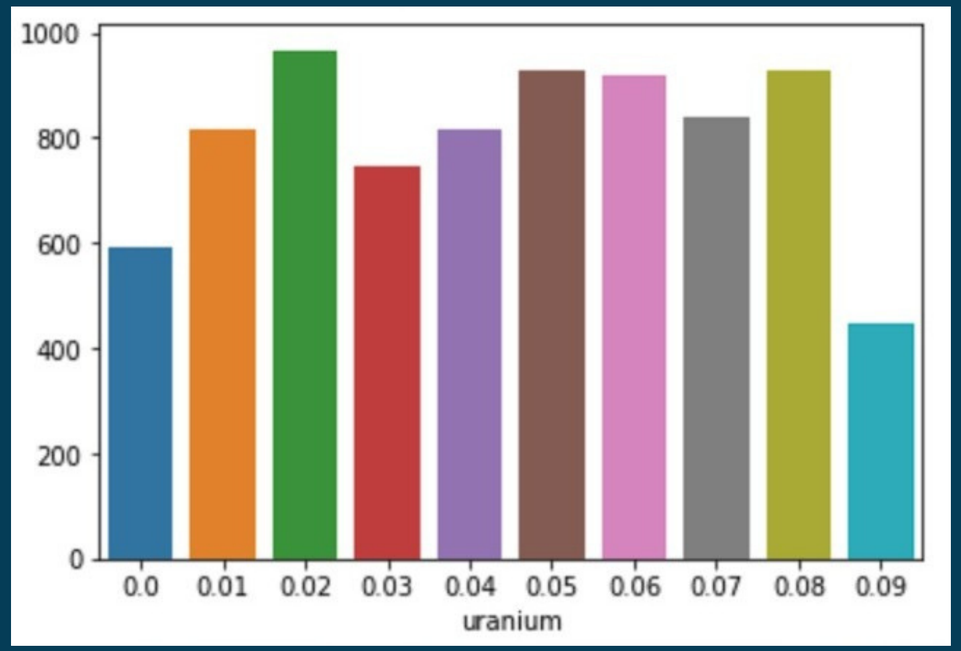


If any one of them increases it is not safe



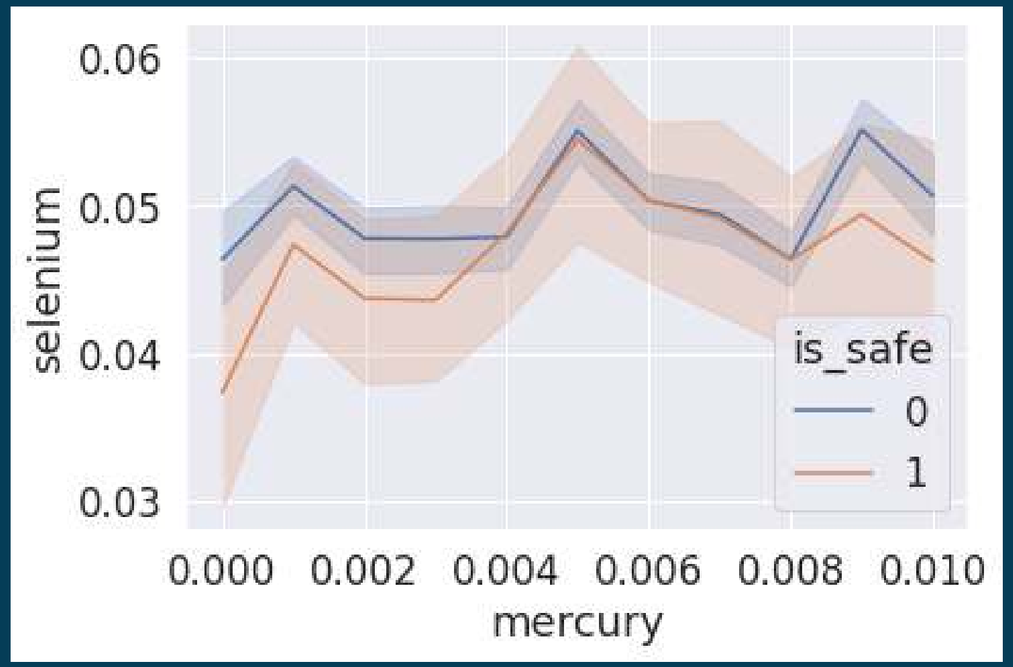
Chloromine, chromium, are highly correlated with target variable

Determining how Uranium is effecting the Is_safe



Maximum water samples of uranium are at 0.02

Finding the relationship between selenium and Mercury



When Selenium is constant and mercury level increases, the water tends to become not safe

LOGISTIC REGRESSION

MODEL	TRAIN-TEST RATIO	ACCURACY
1.	70-30	88.91%
2.	75-25	89.44%
3.	80-20	89.51%

LOGISTIC REGRESSION

After taking several training and testing ratio's, we got the best accuracy in 80-20 percentage.

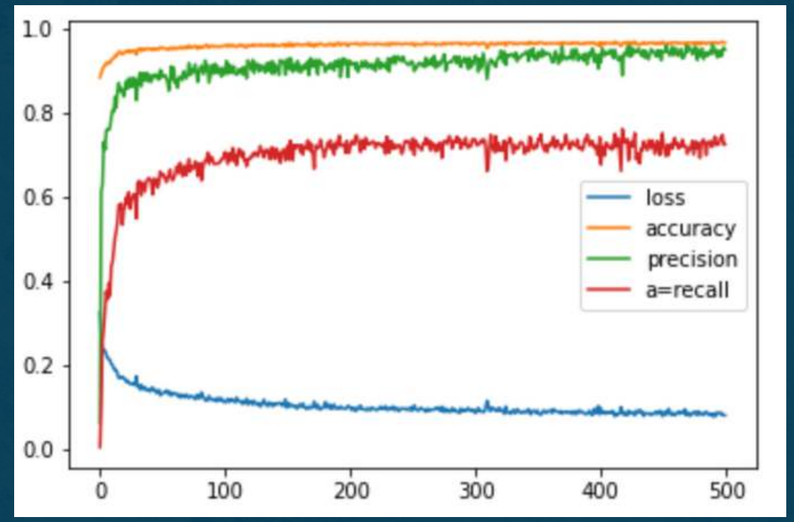
ACCURACY-**89.5%**

NEURAL NETWORK

train and test ratios	Architectures	Optimizer	Epochs	Accuracy	Loss	Precision	Recall
75-25	15-7-5--1	Adam					
			100	0.928	0.1862	0.7172	0.7
			500	0.9435	0.1485	0.9102	0.608
			1000	0.9505	0.1388	0.8832	0.696
75-25	18-7-5--1	Adam					
			100	0.944	0.1601	0.8632	0.656
			500	0.9455	0.1618	0.8219	0.72
			1000	0.95	0.1744	0.7841	0.828
75-25	16-7-5--1	Adam					
			100	0.9365	0.1643	0.7375	0.764
			500	0.956	0.1347	0.9133	0.716
			1000	0.948	0.1591	0.9294	0.632
75-25	12-8-6--1	Adam	50				
			100	0.941	0.169	0.9024	0.592
			500	0.9445	0.1961	0.8927	0.632
			1000	0.9365	0.2311	0.7943	0.664
75-25	14-10-8--1	Adam	50				
			100	0.9525	0.1349	0.8539	0.748
			500	0.9581	0.154	0.9346	0.715
			1000	0.952	0.2334	0.8407	0.76

NEURAL NETWORK

train and test ratios	Architectures	Optimizer	Epochs	Accuracy	Loss	Precision	Recall
80-20	15-7-5--1	Adam					
			100	0.9294	0.1733	0.9579	0.455
			500	0.9556	0.12	9.64E-01	6.70E-01
			1000	0.9556	0.13	0.8525	0.78
80-20	18-7-5--1	Adam					
			100	0.9481	0.1439	0.927	0.635
			500	0.9581	0.175	0.913	0.735
			1000	0.96	0.1444	0.8736	0.795
80-20	16-7-5--1	Adam					
			100	0.945	0.1532	0.9444	0.595
			500	0.9312	0.2424	0.9326	0.485
			1000	0.9531	0.2779	0.893	0.7099
80-20	12-8-6--1	Adam					
			100	0.9206	0.1716	0.8476	0.445
			500	0.9481	0.1671	0.8874	0.67
			1000	0.9525	0.1918	0.8333	0.7749
80-20	14-10-8--1	Adam					
			100	0.9569	0.1313	0.9018	0.735
			500	0.9581	0.154	0.9346	0.715
			1000	0.9569	0.1739	0.8358	0.8149



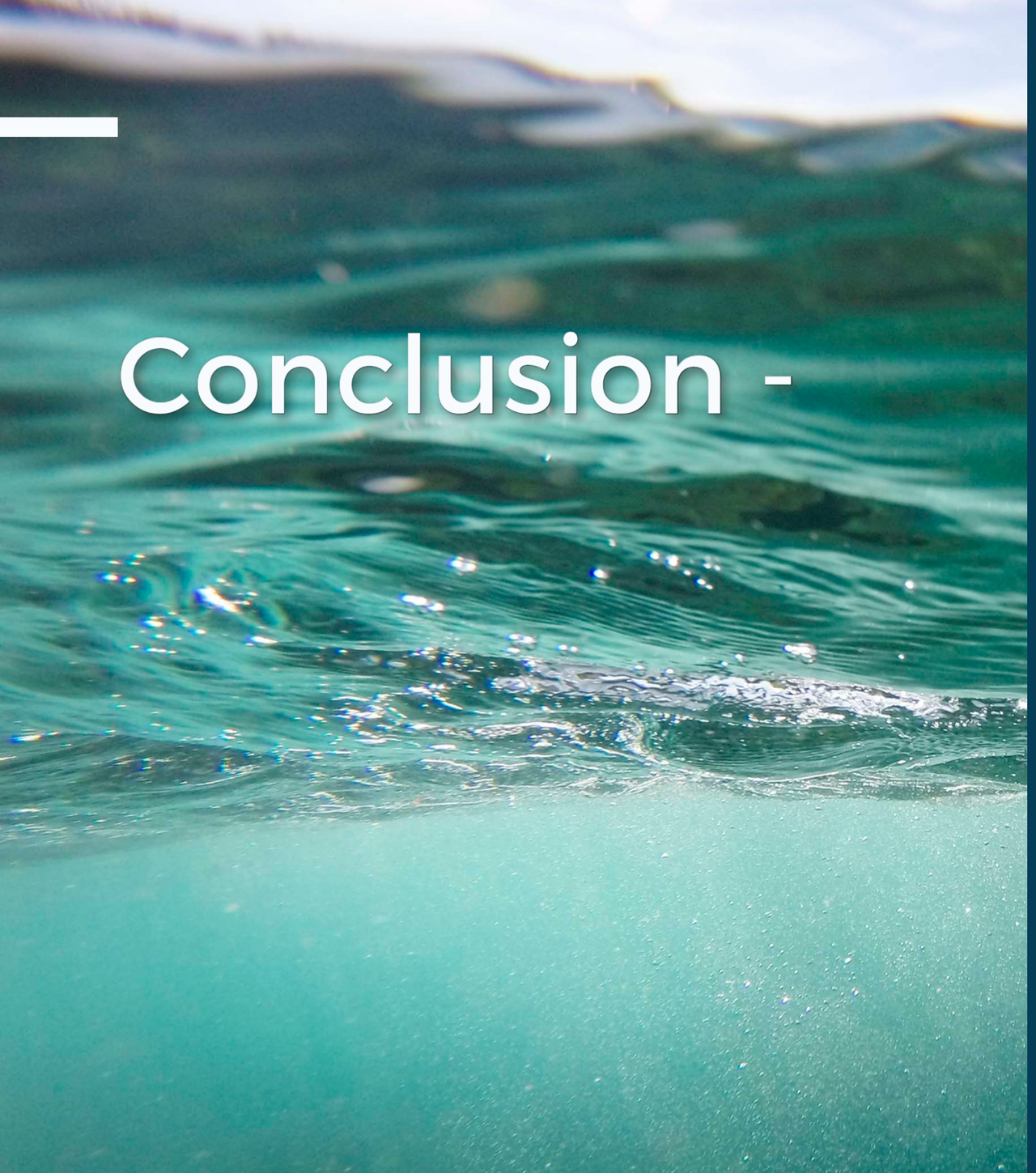
Architecture: 14-10-8-1 under ADAM Optimizer with 500 epochs gives the best accuracy.

BOOSTING & BAGGING

TYPES	TRAIN -TEST	ACCURACY
ADA BOOST	80-20	90.68%
GRADIENT BOOST	80-20	94.25%
EXTREME GRADIENT	80-20	94.25%
BAGGING	80-20	95.81%

SUMMARY

Logistic Regression	89.5%
Neural Network	95.81%
ADA Boost	90.68%
Gradient Boost	94.25%
Extreme Gradient	94.25%
Bagging	95.81%



Conclusion -

After comparing all the models at 80-20 ratio, we can conclude that Neural Network and Bagging are the best fit for our dataset.

THANK YOU

GROUP -11

K.Shiva Kumar Reddy

M. Sai Dheeraj

PV.Tilak

P. Manikethan Reddy

COLAB LINK- HTTPS://COLAB.RESEARCH.GOOGLE.COM/DRIVE/153SAHKHSIBUJR5KDRKLBIWSXLM9BUBNT?USP=SHARING

GITHUB - HTTPS://GITHUB.COM/PVTILAK/WATER-QUALITY.GIT

APPENDIX-

LOGISTIC REGRESSION

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

```
[ ]  x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0, train_size = .80)
```

```
[ ]  from sklearn.linear_model import LogisticRegression        #applying logistic regression

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

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

```
[ ]  from sklearn.metrics import r2_score
     r2_score(y_test,y_pred)
```

```
[ ]  from sklearn.metrics import accuracy_score
     accuracy_score(y_test,y_pred)
```

NEURAL NETWORK -

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

```
[ ]  from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.2, random_state=42)
     y_test
```

```
[ ]  import tensorflow as tf
```

```
[ ]  tf.random.set_seed(42)

     # STEP1: Creating the model

     model= tf.keras.Sequential([       tf.keras.layers.Dense(15, activation='relu'),
                                        tf.keras.layers.Dense(7, activation='relu'),
                                        tf.keras.layers.Dense(5, 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(learning_rate=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

     a= model.fit(x_train, y_train, epochs=1000,verbose=0)
```

```
[ ]  pd.DataFrame(a.history).plot()
```

BAGGING

```
[ ]  from sklearn.ensemble import BaggingClassifier
```

```
[ ]  bag_model = BaggingClassifier(
     base_estimator=BaggingClassifier(),
     n_estimators=1000,
     max_samples=0.8,
     bootstrap=True,
     oob_score=True,
     random_state=42
     )
```

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

```
[ ]  from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.2, random_state=42)
     y_test
```

```
[ ]  l=bag_model.fit(x_train, y_train)
```

```
[ ]  Classifier.score(x_test,y_test)
```

BOOSTING

```
[ ]  #gradientBoosting
     from sklearn import datasets, ensemble
     from sklearn.inspection import permutation_importance
     from sklearn.ensemble import GradientBoostingClassifier
     from sklearn.metrics import mean_squared_error as MSE
```

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

```
[ ]  from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.2, random_state=42)
     y_test
```

```
[ ]  GradientBoostingClassifier,
     Classifier = GradientBoostingClassifier(
          max_depth=2,
          n_estimators=1000,
          learning_rate=0.01,
          random_state=42
     )
     Classifier.fit(x_train, y_train)
```

```
[ ]  Classifier.score(x_test,y_test)
```

```
[ ]  best_Classifier = GradientBoostingClassifier(
          max_depth=2,
          n_estimators=1000,
          learning_rate=0.01,
          random_state=42

     )
     best_Classifier.fit(x_train, y_train)
```

```
[ ]  Classifier.score(x_test,y_test)
```

```
[ ]  #Ada Boosting
     from sklearn.ensemble import AdaBoostClassifier

     adaclf = AdaBoostClassifier(
          n_estimators=1000,
          learning_rate=0.01,
          random_state=42)
     adaclf.fit(x_train, y_train)
     y_pred_1 = adaclf.predict(x_test)
```

```
[ ]  print("Accuracy:",metrics.accuracy_score(y_test, y_pred_1))
```

```
[ ]  #ExtremeGradientBoosting
     from xgboost import XGBClassifier

     adaclf =XGBClassifier(
          n_estimators=1000,
          learning_rate=0.01,
          random_state=42)
     adaclf.fit(x_train, y_train)
     y_pred_1 = adaclf.predict(x_test)
```

```
[ ]  Classifier.score(x_test,y_test)
```