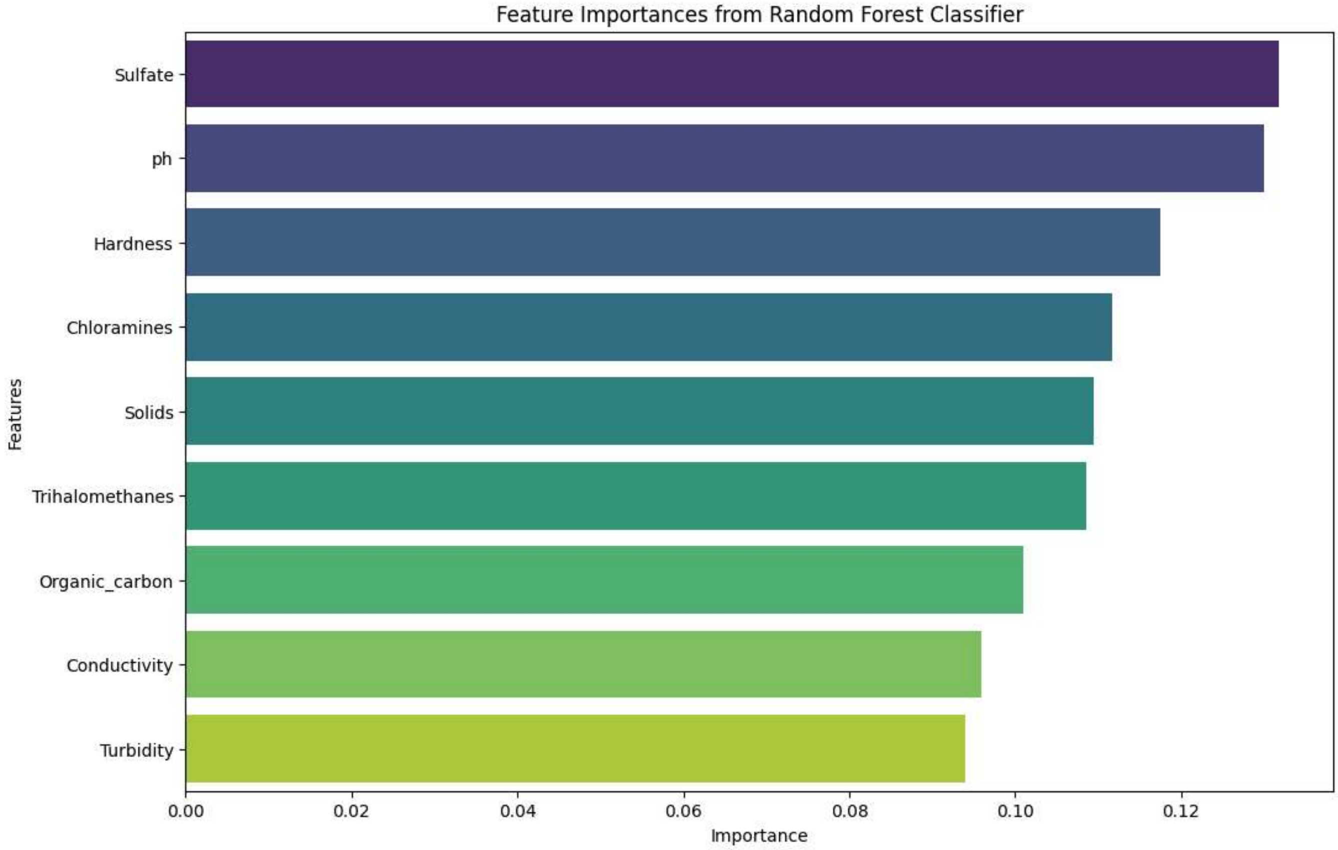
Finding important features from the original dataset

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings("ignore")


# Loading the original dataset
data = pd.read_csv('/content/water_potability (1).csv')

# Splitting the dataset into features and target variable
X = data.drop('Potability', axis=1)
y = data['Potability']

# Spliting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Creating a Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Getting feature importances
importances = model.feature_importances_

# Creating a DataFrame for feature importances
feature_importances = pd.DataFrame(importances, index=X.columns, columns=['importance']).sort_values('importance', ascending=False)

# Selecting the top 4 features
top_features = feature_importances.head(4).index.tolist()

# Creating a new DataFrame with the top features and the target column
top_features_with_target = data[top_features + ['Potability']]

# Saving the new DataFrame to a CSV file for further use
top_features_with_target.to_csv('top_4_features_with_target.csv', index=False)

print("Top 4 features along with the target column saved to 'top_4_features_with_target.csv'")
```

```
Top 4 features along with the target column saved to 'top_4_features_with_target.csv'
```

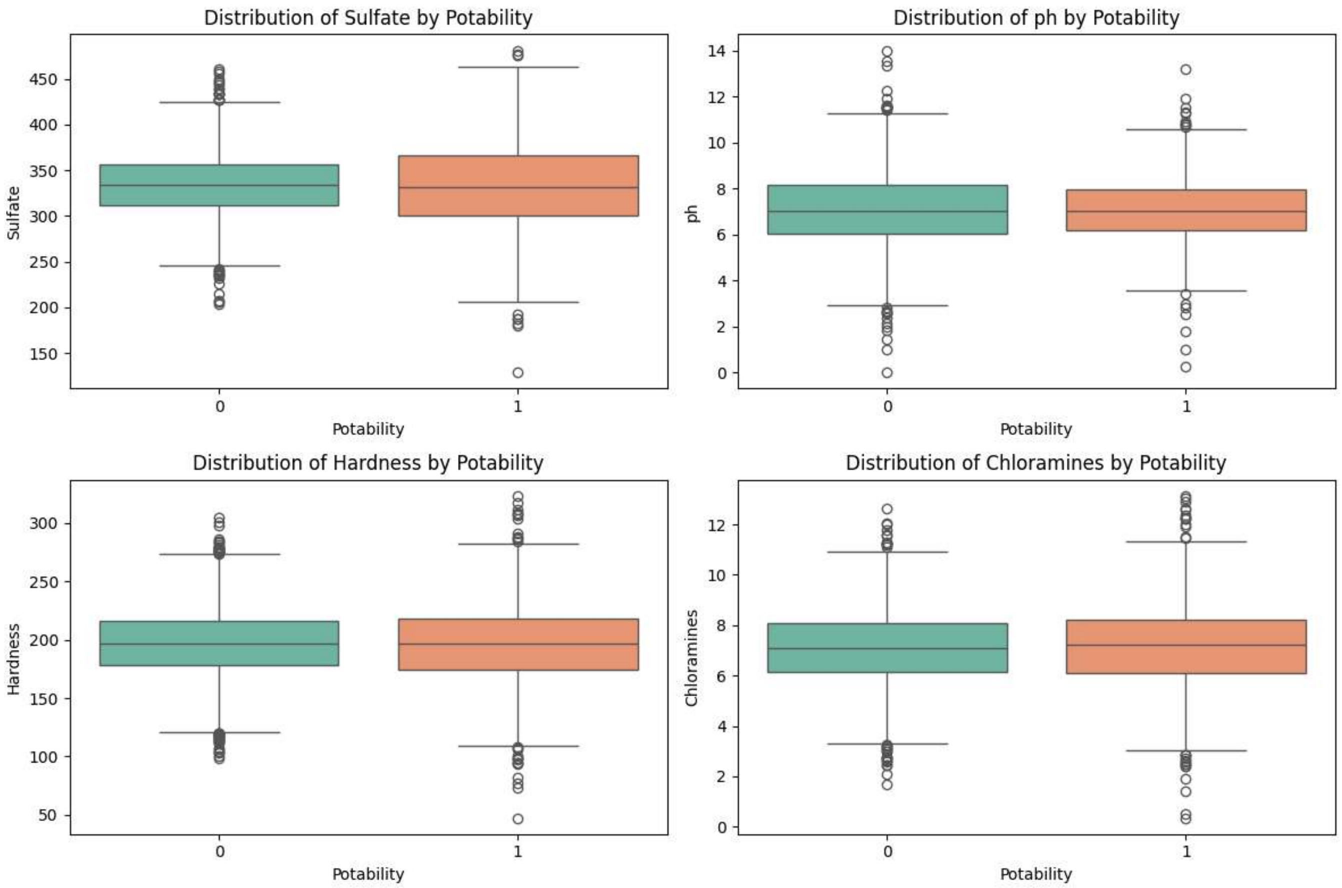
Visualization: Bar Plot of all feature importances

```
import matplotlib.pyplot as plt
import seaborn as sns
feature_importances = pd.DataFrame(importances, index=X.columns, columns=['importance']).sort_values('importance', ascending=False)
plt.figure(figsize=(12, 8))
sns.barplot(x=feature_importances['importance'], y=feature_importances.index, palette='viridis')
plt.title('Feature Importances from Random Forest Classifier')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.show()
```



Box Plot for each of the top features against the target variable

```
plt.figure(figsize=(12, 8))
for i, feature in enumerate(top_features):
    plt.subplot(2, 2, i + 1)  # Create a 2x2 grid of subplots
    sns.boxplot(x='Potability', y=feature, data=top_features_with_target, palette='Set2')
    plt.title(f'Distribution of {feature} by Potability')
    plt.xlabel('Potability')
    plt.ylabel(feature)
plt.tight_layout()
plt.show()
```



Importing necessary libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, recall_score, precision_score, classification_report
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
import pickle
```

Function to calculate binary features for the new dataset

```
def calculate_binary_features(df):
  #Defining threshold values
    thresholds = {
        'ph': (6.5, 8.5),
        'Hardness': 200.0,
        'Chloramines': 4.0,
        'Sulfate': 250.0
    }

    #Creating binary features based on thresholds
    for feature, threshold in thresholds.items():
        if feature in df.columns:
            if isinstance(threshold, tuple):
                df[f'is_{feature}_ok'] = ((df[feature] >= threshold[0]) & (df[feature] <= threshold[1])).astype(int)
            else:
                df[f'is_{feature}_ok'] = (df[feature] <= threshold).astype(int)
        else:
            print(f"Warning: Column '{feature}' not found in DataFrame.")

    return df
```

Loading the new dataset

```
data = pd.read_csv('/content/top_4_features_with_target.csv')
```

Previewing Data with Pandas

```
data.head()
```

	Sulfate	ph	Hardness	Chloramines	Potability
0	368.516441	NaN	204.890455	7.300212	0
1	NaN	3.716080	129.422921	6.635246	0
2	NaN	8.099124	224.236259	9.275884	0
3	356.886136	8.316766	214.373394	8.059332	0
4	310.135738	9.092223	181.101509	6.546600	0

Next steps: Generate code with data View recommended plots New interactive sheet

Understanding the data

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Sulfate      2495 non-null   float64
 1   ph           2785 non-null   float64
 2   Hardness     3276 non-null   float64
 3   Chloramines  3276 non-null   float64
 4   Potability   3276 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 128.1 KB
```

```
data.describe()
```

	Sulfate	ph	Hardness	Chloramines	Potability
count	2495.000000	2785.000000	3276.000000	3276.000000	3276.000000
mean	333.775777	7.080795	196.369496	7.122277	0.390110
std	41.416840	1.594320	32.879761	1.583085	0.487849
min	129.000000	0.000000	47.432000	0.352000	0.000000
25%	307.699498	6.093092	176.850538	6.127421	0.000000
50%	333.073546	7.036752	196.967627	7.130299	0.000000
75%	359.950170	8.062066	216.667456	8.114887	1.000000
max	481.030642	14.000000	323.124000	13.127000	1.000000

```
data['Potability'].value_counts()
```

	count
Potability	
0	1998
1	1278

dtype: int64

Filling missing values with column means

```
data.fillna(data.mean(),inplace=True)
```

Binary features calculation

```
data = calculate_binary_features(data)
print(data)
```

```
         Sulfate        ph    Hardness  Chloramines  Potability  is_ph_ok  \
0     368.516441  7.080795  204.890455     7.300212           0         1   
1     333.775777  3.716080  129.422921     6.635246           0         0   
2     333.775777  8.099124  224.236259     9.275884           0         1   
3     356.886136  8.316766  214.373394     8.059332           0         1   
```

```
4     310.135738  9.092223  181.101509     6.546600           0         0
...          ...       ...         ...          ...         ...       ...
3271  359.948574  4.668102  193.681735     7.166639           1         0
3272  333.775777  7.808856  193.553212     8.061362           1         1
3273  333.775777  9.419510  175.762646     7.350233           1         0
3274  333.775777  5.126763  230.603758     6.303357           1         0
3275  333.775777  7.874671  195.102299     7.509306           1         1

      is_Hardness_ok  is_Chloramines_ok  is_Sulfate_ok
0                  0                  0              0
1                  1                  0              0
2                  0                  0              0
3                  0                  0              0
4                  1                  0              0
...              ...                ...            ...
3271               1                  0              0
3272               1                  0              0
3273               1                  0              0
3274               0                  0              0
3275               1                  0              0

[3276 rows x 9 columns]
```

Preparation of features and target variable

```
upx = data.iloc[:, :-1].values
upy = data.iloc[:, -1].values
```

Splitting the data into training and testing sets

```
upx_train, upx_test, upy_train, upy_test = train_test_split(upx, upy, train_size=0.8, random_state=42)
```

Defining models

```
models = [
    LogisticRegression(C=5.0),
    GaussianNB(),
    svm.SVC(C=0.5),
    KNeighborsClassifier(n_neighbors=30)
]
```

```
models_names = ['LogisticRegression', 'GaussianNB', 'SVC', 'KNeighborsClassifier']

Recall = []
Specificity = []
Accuracy = []
Precision = []
F1_Score = []
```

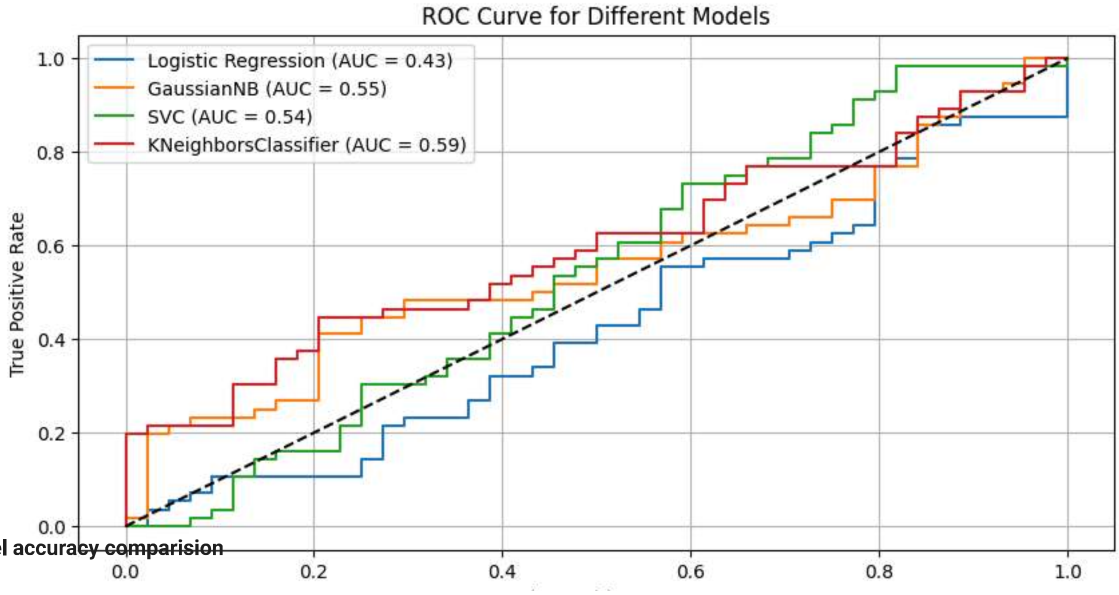
Training , evaluation of models and saving them to pickle files.

```
for z, model in enumerate(models):
    print(model)
    model.fit(upx_train, upy_train)

    pickle.dump(model, open(f"{models_names[z]}.pkl", 'wb'))

    upy_pred = model.predict(upx_test)
    print(classification_report(upy_test, upy_pred))

    cm_test = confusion_matrix(upy_test, upy_pred)
    print(".......")
    print("Recall on Test Data:", round(recall_score(upy_test, upy_pred), 4))
    Recall.append(round(recall_score(upy_test, upy_pred), 4))
    print("Specificity on Test Data:", round((cm_test[0, 0] / (cm_test[0, 0] + cm_test[0, 1])), 4))
    Specificity.append(round((cm_test[0, 0] / (cm_test[0, 0] + cm_test[0, 1])), 4))
    print("Accuracy on Test Data:", round((accuracy_score(upy_test, upy_pred) * 100), 4))
    Accuracy.append(round(accuracy_score(upy_test, upy_pred), 4))
    print("Precision on Test Data: ", round(precision_score(upy_test, upy_pred), 4))
    Precision.append(round(precision_score(upy_test, upy_pred), 4))
    print("F1 Score on Test Data: ", round(f1_score(upy_test, upy_pred), 4))
    F1_Score.append(round(f1_score(upy_test, upy_pred), 4))
    print(".......\n")
```

```
           1       0.39      0.64      0.48        11

    accuracy                           0.98       656
   macro avg       0.69      0.81      0.74       656
weighted avg       0.98      0.98      0.98       656

.......
Recall on Test Data: 0.6364
Specificity on Test Data: 0.9829
Accuracy on Test Data: 97.7134
Precision on Test Data:  0.3889
F1 Score on Test Data:  0.4828
.......

SVC(C=0.5)
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       645
           1       0.00      0.00      0.00        11

    accuracy                           0.98       656
   macro avg       0.49      0.50      0.50       656
weighted avg       0.97      0.98      0.97       656

.......
Recall on Test Data: 0.0
Specificity on Test Data: 1.0
Accuracy on Test Data: 98.3232
Precision on Test Data:  0.0
F1 Score on Test Data:  0.0
.......

KNeighborsClassifier(n_neighbors=30)
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       645
           1       1.00      0.36      0.53        11

    accuracy                           0.99       656
   macro avg       0.99      0.68      0.76       656
weighted avg       0.99      0.99      0.99       656

.......
Recall on Test Data: 0.3636
Specificity on Test Data: 1.0
Accuracy on Test Data: 98.9329
Precision on Test Data:  1.0
F1 Score on Test Data:  0.5333
.......
```

ROC -curve

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import roc_curve, auc

np.random.seed(0)
true_labels = np.random.randint(0, 2, size=100)  # Replace with actual true labels
predicted_probs_logistic = np.random.rand(100)  # Replace with actual predicted probabilities
predicted_probs_gaussian = np.random.rand(100)  # Replace with actual predicted probabilities
predicted_probs_svc = np.random.rand(100)  # Replace with actual predicted probabilities
predicted_probs_knn = np.random.rand(100)  # Replace with actual predicted probabilities

# Function to plot ROC curve
def plot_roc_curve(true_labels, predicted_probs, model_name):
    fpr, tpr, _ = roc_curve(true_labels, predicted_probs)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')

# Plotting ROC curves
plt.figure(figsize=(10, 5))
plot_roc_curve(true_labels, predicted_probs_logistic, 'Logistic Regression')
plot_roc_curve(true_labels, predicted_probs_gaussian, 'GaussianNB')
plot_roc_curve(true_labels, predicted_probs_svc, 'SVC')
plot_roc_curve(true_labels, predicted_probs_knn, 'KNeighborsClassifier')

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.title('ROC Curve for Different Models')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid()
plt.show()
```



Model accuracy comparision

```
import matplotlib.pyplot as plt
import numpy as np

# Model names
models_names = ['Logistic Regression', 'Gaussian Naive Bayes', 'SVM', 'K-Nearest Neighbors']

# Accuracy values for each model (replace these with your actual accuracy values)
accuracy_values = [98.78, 97.71, 98.32, 98.93]  # Example accuracy values

# Create a horizontal bar chart
plt.figure(figsize=(10, 6))
bars = plt.barh(models_names, accuracy_values, color=['blue', 'orange', 'green', 'red'], height=0.4)  # Adjust height for bar width

# Add accuracy values next to the bars
for bar in bars:
    plt.text(bar.get_width(), bar.get_y() + bar.get_height()/2, f'{bar.get_width():.2f}', va='center')

# Adding titles and labels
plt.title('Model Accuracy Comparison')
plt.xlabel('Accuracy (%)')
plt.xlim(0, 100)  # Set x-axis limit from 0 to 100
plt.grid(axis='x')

# Show the plot
plt.show()
```