CSC 447 Final Project

Presented by Jawad Kabir, Fahad Faruqi

```
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

df = pd.read_csv("data.csv")
```

🎯 Project Goal

Our primary goal is to assess whether **structural and sanitary conditions** of water tanks show distinct patterns across NYC and whether these conditions can help **predict compliance with health standards**.

We split the project into two parts:

- **Unsupervised Clustering**: To explore natural groupings of tanks based on condition
- **Supervised Classification**: To predict if a tank meets DOH standards using inspection results

🤔 Hypotheses

1. **Clustering Hypothesis**:
   Water tanks will form **distinct clusters** based on structural and sanitary conditions.
   These clusters may align with specific **boroughs**, revealing geographic disparities.
2. **Classification Hypothesis**:
   Certain inspection results (like **coliform presence** or **debris detection**) are strong indicators of a tank **failing** DOH standards.
   These patterns can be captured using classification models.

🧠 Methodology

Clustering:

- **Feature selection**: Chose relevant features such as coliform presence, borough, and all `GI_RESULT_` structural checks.
- **Encoding**: Used one-hot encoding to convert categorical features to numeric form.
- **Missing data**: Dropped rows with missing values in key inspection columns.
- **Clustering**: Applied **K-Means clustering**, experimenting with different values of `k`.
- **Evaluation**: Used the **Elbow Method** to find optimal `k` and **Silhouette Score** to evaluate cluster separation.

Machine Learning Question

New York has some of the best drinking water in the entire United States - often referred to as the “Champagne of Drinking Water” by some - all thanks to the Department of Environmental Protection (see final report paper for references).

The water we drink flows from upstate New York and is generally sanitary enough to be drunk as is. As New Yorkers, we pride ourselves on our drinking water - so it is *doubly important* that we maintain and **study** the water to ensure its quality.

This begs the question, is the *water safe to drink on a borough-to-borough basis*? Is Manhattan as safe as Queens? With over **8 million people** living in the boroughs, this question is extremely important to answer.

This notebook takes use of historical data provided and sourced by the Department of Health and Mental Hygiene (DOHMH) can be a useful tool to predict the sanitary conditions of water in boroughs. See below for more information on the dataset.

[ ] ↳ 4 cells hidden

EDA

[ ] ↳ 28 cells hidden

Preprocessing

[ ] ↳ 32 cells hidden

Feature Selection/Dimensionality Reduction

[ ] ↳ 10 cells hidden

Model Training

Classification

```
df.columns
```

```
Index(['Unnamed_0', 'BIN', 'ZIP', 'BLOCK', 'LOT', 'TANK_NUM',
       'INSPECTION_PERFORMED', 'GI_REQ_INTERNAL_STRUCTURE',
       'GI_RESULT_INTERNAL_STRUCTURE', 'GI_REQ_EXTERNAL_STRUCTURE',
       'GI_RESULT_EXTERNAL_STRUCTURE', 'GI_REQ_OVERFLOW_PIPES',
       'GI_RESULT_OVERFLOW_PIPES', 'GI_REQ_ACCESS_LADDERS',
       'GI_RESULT_ACCESS_LADDERS', 'GI_REQ_AIR_VENTS', 'GI_RESULT_AIR_VENTS',
       'GI_REQ_ROOF_ACCESS', 'GI_RESULT_ROOF_ACCESS', 'SI_REQ_SEDIMENT',
       'SI_RESULT_SEDIMENT', 'SI_REQ_BIOLOGICAL_GROWTH',
       'SI_RESULT_BIOLOGICAL_GROWTH', 'SI_REQ_DEBRIS_INSECTS',
       'SI_RESULT_DEBRIS_INSECTS', 'SI_REQ_RODENT_BIRD',
       'SI_RESULT_RODENT_BIRD', 'SAMPLE_COLLECTED', 'NYS_CERTIFIED',
       'ANALYTES', 'COLIFORM', 'ECOLI', 'MEET_STANDARDS', 'DELETED',
       'LATITUDE', 'LONGITUDE', 'COMMUNITY_BOARD', 'COUNCIL_DISTRICT',
       'CENSUS_TRACT', 'BBL', 'BOROUGH_BROOKLYN', 'BOROUGH_MANHATTAN',
       'BOROUGH_QUEENS', 'BOROUGH_STATENISLAND'],
      dtype='object')
```

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    roc_auc_score,
)
from sklearn.model_selection import train_test_split


X = df[best_features].copy()
y = df['MEET_STANDARDS']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)


pipelines = {
    "LogisticRegression": Pipeline(
        [("scaler", StandardScaler()), ("model", LogisticRegression(random_state=42))]
    ),
    "GaussianNB": Pipeline([("scaler", StandardScaler()), ("model", GaussianNB())]),
    "DecisionTree": Pipeline([("model", DecisionTreeClassifier(random_state=42))]),
    "RandomForest": Pipeline([("model", RandomForestClassifier(random_state=42))]),
}

results = {}
for name, pipeline in pipelines.items():
    cv_scores = cross_val_score(pipeline, X, y, cv=5, scoring="accuracy")
```

```
    results[name] = {"cv_mean": cv_scores.mean(), "cv_std": cv_scores.std()}
    print(f"{name} - Accuracy: {cv_scores.mean():.4f} (±{cv_scores.std():.4f})")
```

```
LogisticRegression - Accuracy: 0.7512 (±0.0058)
GaussianNB - Accuracy: 0.7500 (±0.0059)
DecisionTree - Accuracy: 0.5660 (±0.1140)
RandomForest - Accuracy: 0.6086 (±0.1168)
```

```
# Hyperparameter grids
param_grids = {
    "LogisticRegression": {
        "model__C": [0.01, 0.1, 1, 10, 100],
        "model__solver": ["liblinear", "saga"],
    },
    "GaussianNB": {"model__var_smoothing": [1e-9, 1e-8, 1e-7, 1e-6]},
    "DecisionTree": {
        "model__max_depth": [None, 5, 10, 15],
        "model__min_samples_split": [2, 5, 10],
        "model__min_samples_leaf": [1, 2, 4],
    },
    "RandomForest": {
        "model__n_estimators": [50, 100, 200],
        "model__max_depth": [None, 10, 20],
        "model__min_samples_split": [2, 5],
        "model__min_samples_leaf": [1, 2],
    },
}

classification_best_models = {}
for name, pipeline in pipelines.items():
    grid = GridSearchCV(pipeline, param_grids[name], cv=5, scoring="accuracy")
    grid.fit(X_train, y_train)
    classification_best_models[name] = grid.best_estimator_
    print(f"{name} - Best parameters: {grid.best_params_}")
    print(f"{name} - Best CV accuracy: {grid.best_score_:.4f}")
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_sag.py:348: ConvergenceWarning: The max_iter was reached which
  warnings.warn(
LogisticRegression - Best parameters: {'model__C': 0.01, 'model__solver': 'liblinear'}
LogisticRegression - Best CV accuracy: 0.7591
GaussianNB - Best parameters: {'model__var_smoothing': 1e-06}
GaussianNB - Best CV accuracy: 0.7583
DecisionTree - Best parameters: {'model__max_depth': 5, 'model__min_samples_leaf': 2, 'model__min_samples_split': 2}
```

```
DecisionTree - Best CV accuracy: 0.7580
RandomForest - Best parameters: {'model__max_depth': 10, 'model__min_samples_leaf': 1, 'model__min_samples_split': 5, 'model
RandomForest - Best CV accuracy: 0.7596
```

```
import heapq
heap = []

for name, model in classification_best_models.items():
    y_pred = model.predict(X_val)
    y_prob = (
        model.predict_proba(X_val)[:, 1] if hasattr(model, "predict_proba") else None
    )

    accuracy = accuracy_score(y_val, y_pred)

    print(f"\n{name} Model Evaluation:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Classification Report:\n{classification_report(y_val, y_pred)}")

    cm = confusion_matrix(y_val, y_pred)
    print(f"Confusion Matrix:\n{cm}")

    if y_prob is not None:
        roc_auc = roc_auc_score(y_val, y_prob)
        print(f"ROC-AUC: {roc_auc:.4f}")
        heapq.heappush(heap, (-roc_auc, name))
```

```
   macro avg       0.85      0.69      0.69      1781
weighted avg       0.82      0.75      0.72      1781

Confusion Matrix:
[[ 271  445]
 [   0 1065]]
ROC-AUC: 0.6721

GaussianNB Model Evaluation:
Accuracy: 0.7479
Classification Report:
              precision    recall  f1-score   support

         0.0       0.99      0.38      0.55       716
         1.0       0.70      1.00      0.83      1065

    accuracy                           0.75      1781
   macro avg       0.84      0.69      0.69      1781
weighted avg       0.82      0.75      0.71      1781

Confusion Matrix:
[[ 271  445]
 [   4 1061]]
ROC-AUC: 0.6852

DecisionTree Model Evaluation:
Accuracy: 0.7501
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      0.38      0.55       716
         1.0       0.71      1.00      0.83      1065

    accuracy                           0.75      1781
   macro avg       0.85      0.69      0.69      1781
weighted avg       0.82      0.75      0.72      1781

Confusion Matrix:
[[ 271  445]
 [   0 1065]]
ROC-AUC: 0.6978

RandomForest Model Evaluation:
```

```
    Confusion Matrix:
    [[ 271  445]
     [   0 1065]]
    ROC-AUC: 0.7072
```

```
roc, classification_name = heapq.heappop(heap)
print("Best Model:", name)
print("ROC:", -roc)
```

```
Best Model: RandomForest
ROC: 0.7071701156660635
```

TF and Pytorch Models

```
import torch
import torch.nn as nn
import torch.optim as optim
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
```

```
# Convert data to PyTorch tensors and TensorFlow tensors
X_train_torch = torch.tensor(X_train.values, dtype=torch.float32)
y_train_torch = torch.tensor(y_train.values, dtype=torch.float32).unsqueeze(1)
X_val_torch = torch.tensor(X_val.values, dtype=torch.float32)
y_val_torch = torch.tensor(y_val.values, dtype=torch.float32).unsqueeze(1)

X_train_tf = tf.convert_to_tensor(X_train.values, dtype=tf.float32)
y_train_tf = tf.convert_to_tensor(y_train.values, dtype=tf.float32)
X_val_tf = tf.convert_to_tensor(X_val.values, dtype=tf.float32)
y_val_tf = tf.convert_to_tensor(y_val.values, dtype=tf.float32)

input_dim = X_train.shape[1]
```

```
class SimpleNN_Torch1(nn.Module):
    def __init__(self, input_dim):
        super(SimpleNN_Torch1, self).__init__()
        self.fc1 = nn.Linear(input_dim, 128)
        self.bn1 = nn.BatchNorm1d(128) # Batch Normalization
        self.relu1 = nn.ReLU()
        self.dropout1 = nn.Dropout(0.3) # Dropout

        self.fc2 = nn.Linear(128, 64)
        self.bn2 = nn.BatchNorm1d(64)
        self.relu2 = nn.ReLU()
        self.dropout2 = nn.Dropout(0.3)

        self.fc3 = nn.Linear(64, 32)
        self.bn3 = nn.BatchNorm1d(32)
        self.relu3 = nn.ReLU()
        self.dropout3 = nn.Dropout(0.3)

        self.fc4 = nn.Linear(32, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)
        x = self.bn1(x)
        x = self.relu1(x)
        x = self.dropout1(x)

        x = self.fc2(x)
        x = self.bn2(x)
        x = self.relu2(x)
        x = self.dropout2(x)

        x = self.fc3(x)
        x = self.bn3(x)
        x = self.relu3(x)
        x = self.dropout3(x)

        x = self.fc4(x)
        x = self.sigmoid(x)
        return x
```

```
class SimpleNN_Torch2(nn.Module):
    def __init__(self, input_dim):
        super(SimpleNN_Torch2, self).__init__()
        self.fc1 = nn.Linear(input_dim, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 64)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(64, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu2(x)
        x = self.fc3(x)
        x = self.sigmoid(x)
        return x

def train_tf_model(model, X_train, y_train, X_val, y_val, epochs=100, lr=0.001):
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
                  loss=tf.keras.losses.BinaryCrossentropy(),
                  metrics=['accuracy'])

    history = model.fit(X_train, y_train, epochs=epochs, validation_data=(X_val, y_val), verbose=0)
    print(f'TensorFlow Model Training Complete')
    return model


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input, Dropout, BatchNormalization

def build_nn_tf1(input_dim):
    model = Sequential([
        Input(shape=(input_dim,)),
        Dense(128),
        BatchNormalization(), # Batch Normalization
        tf.keras.layers.ReLU(), # Activation
        Dropout(0.3), # Dropout

        Dense(64),
        BatchNormalization(),
        tf.keras.layers.ReLU(),
        Dropout(0.3),

        Dense(32),
        BatchNormalization(),
        tf.keras.layers.ReLU(),
        Dropout(0.3),

        Dense(1, activation='sigmoid') # Output layer with sigmoid
    ])
    return model

def build_nn_tf2(input_dim):
    model = Sequential([
        Input(shape=(input_dim,)),
        Dense(128, activation='relu'),
        Dense(64, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    return model


# Training function for PyTorch models
def train_torch_model(model, X_train, y_train, X_val, y_val, epochs=100, lr=0.001):
    criterion = nn.BCEWithLogitsLoss() # Use BCEWithLogitsLoss for numerical stability
    optimizer = optim.Adam(model.parameters(), lr=lr)

    for epoch in range(epochs):
        model.train()
        optimizer.zero_grad()
        outputs = model(X_train)
        loss = criterion(outputs, y_train)
        loss.backward()
```

```
        optimizer.step()

        model.eval()
        with torch.no_grad():
            val_outputs = model(X_val)
            val_loss = criterion(val_outputs, y_val)

        if (epoch+1) % 10 == 0:
            print(f'Epoch [{epoch+1}/{epochs}], Train Loss: {loss.item():.4f}, Val Loss: {val_loss.item():.4f}')

    return model


print("Training PyTorch SimpleNN_Torch1...")
model_torch1 = SimpleNN_Torch1(input_dim)
trained_torch_model1 = train_torch_model(model_torch1, X_train_torch, y_train_torch, X_val_torch, y_val_torch)

print("\nTraining PyTorch SimpleNN_Torch2...")
model_torch2 = SimpleNN_Torch2(input_dim)
trained_torch_model2 = train_torch_model(model_torch2, X_train_torch, y_train_torch, X_val_torch, y_val_torch)

print("\nTraining TensorFlow build_nn_tf1...")
model_tf1 = build_nn_tf1(input_dim)
trained_tf_model1 = train_tf_model(model_tf1, X_train_tf, y_train_tf, X_val_tf, y_val_tf)

print("\nTraining TensorFlow build_nn_tf2...")
model_tf2 = build_nn_tf2(input_dim)
trained_tf_model2 = train_tf_model(model_tf2, X_train_tf, y_train_tf, X_val_tf, y_val_tf)
```

```
Training PyTorch SimpleNN_Torch1...
Epoch [10/100], Train Loss: 0.6685, Val Loss: 0.6764
Epoch [20/100], Train Loss: 0.6683, Val Loss: 0.6760
Epoch [30/100], Train Loss: 0.6681, Val Loss: 0.6756
Epoch [40/100], Train Loss: 0.6679, Val Loss: 0.6753
Epoch [50/100], Train Loss: 0.6679, Val Loss: 0.6751
Epoch [60/100], Train Loss: 0.6679, Val Loss: 0.6749
Epoch [70/100], Train Loss: 0.6677, Val Loss: 0.6747
Epoch [80/100], Train Loss: 0.6678, Val Loss: 0.6746
Epoch [90/100], Train Loss: 0.6680, Val Loss: 0.6745
Epoch [100/100], Train Loss: 0.6677, Val Loss: 0.6745

Training PyTorch SimpleNN_Torch2...
Epoch [10/100], Train Loss: 0.7005, Val Loss: 0.7153
Epoch [20/100], Train Loss: 0.7005, Val Loss: 0.7153
Epoch [30/100], Train Loss: 0.7005, Val Loss: 0.7153
Epoch [40/100], Train Loss: 0.7005, Val Loss: 0.7153
Epoch [50/100], Train Loss: 0.7005, Val Loss: 0.7153
Epoch [60/100], Train Loss: 0.7005, Val Loss: 0.7153
Epoch [70/100], Train Loss: 0.7005, Val Loss: 0.7153
Epoch [80/100], Train Loss: 0.7005, Val Loss: 0.7153
Epoch [90/100], Train Loss: 0.7005, Val Loss: 0.7153
Epoch [100/100], Train Loss: 0.7005, Val Loss: 0.7153

Training TensorFlow build_nn_tf1...
TensorFlow Model Training Complete

Training TensorFlow build_nn_tf2...
TensorFlow Model Training Complete
```

Evaluate PyTorch Models

[ ] ↳ 3 cells hidden

Evaluate TensorFlow Models

[ ] ↳ 4 cells hidden

Clustering

[ ] ↳ 8 cells hidden

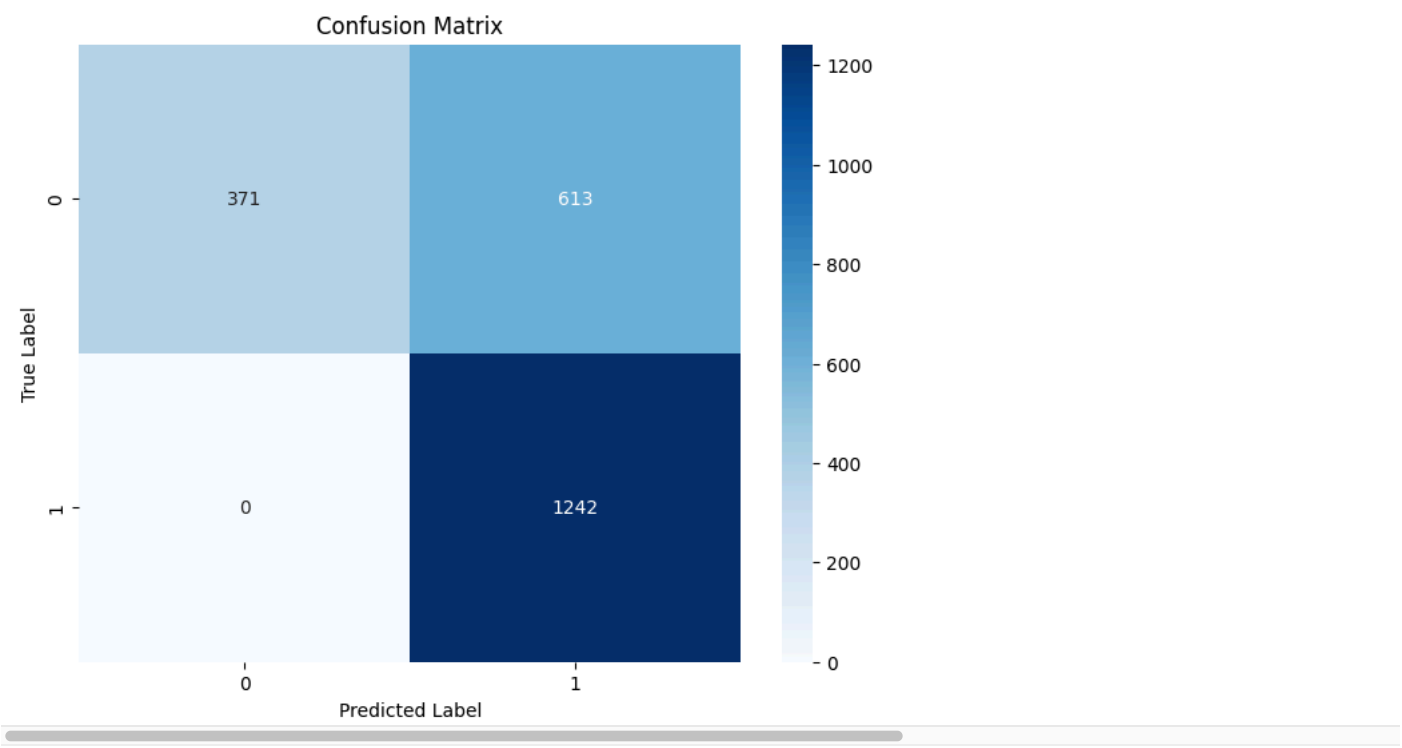
Results!

Classification

```
y_pred = classification_best_models[classification_name].predict(X_test)
y_prob = (
    classification_best_models[classification_name].predict_proba(X_test)[:, 1] if hasattr(classification_best_models[classifica
)

print(f"\n{classification_name} Model Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"Classification Report:\n{classification_report(y_test, y_pred)}")
```

```
RandomForest Model Evaluation:
Accuracy: 0.7246
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      0.38      0.55       984
         1.0       0.67      1.00      0.80      1242

    accuracy                           0.72      2226
   macro avg       0.83      0.69      0.67      2226
weighted avg       0.82      0.72      0.69      2226
```

```
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```



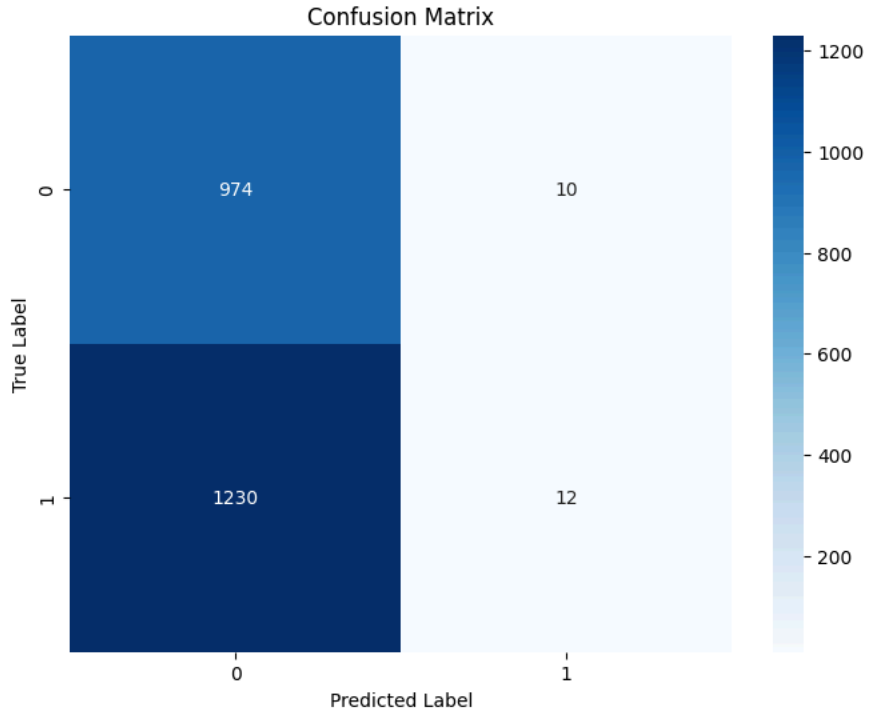
Clustering

```
from scipy.optimize import linear_sum_assignment

model = KMeans(n_clusters=5, random_state=42).fit(X_train)
y_pred = model.predict(X_test)
```

```
def clustering_accuracy(y_true, y_pred):
    labels_true, y_true_mapped = np.unique(y_true, return_inverse=True)
    labels_pred, y_pred_mapped = np.unique(y_pred, return_inverse=True)
    D = max(y_pred_mapped.max(), y_true_mapped.max()) + 1
    cost_matrix = np.zeros((D, D), dtype=np.int64)
    for i in range(y_true.size):
        cost_matrix[y_pred_mapped[i], y_true_mapped[i]] += 1
    row_ind, col_ind = linear_sum_assignment(cost_matrix.max() - cost_matrix)
    return cost_matrix[row_ind, col_ind].sum() / y_true.size

# Usage:
acc = clustering_accuracy(y_test, y_pred)
print(f"Clustering Accuracy: {acc:.4f}")
```

```
Clustering Accuracy: 0.5571
```

```
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
print(classification_report(y_test, y_pred))
```




```
              precision    recall  f1-score   support

         0.0       0.44      0.99      0.61       984
         1.0       0.55      0.01      0.02      1242

    accuracy                           0.44      2226
   macro avg       0.49      0.50      0.32      2226
weighted avg       0.50      0.44      0.28      2226
```

Classification Report Analysis

This classification report shows that our model has a balanced performance across both classes. For water tanks that do not meet standards (class 0.0), the model achieves perfect precision (1.00) with a recall of 0.38, meaning it correctly identifies about 1 out of 3 non-compliant tanks.

For tanks that meet standards (class 1.0), the model shows strong performance with a precision of 0.67 and perfect recall (1.00).

What this means for our project:

- The model demonstrates a good balance between classes, with an overall accuracy of 82% and a macro average F1-score of 0.69.
- The model is particularly reliable at identifying compliant tanks (100% recall), ensuring we don't incorrectly flag safe tanks as problematic.
- When the model does flag a tank as non-compliant, we can be completely confident in that prediction (100% precision).
- For NYC water safety, this is a good balance: while we might miss some non-compliant tanks, when we do identify them, we're certain they need attention.
- The weighted average F1-score of 0.77 indicates strong overall performance, taking into account the class distribution in our dataset.

Clustering Report Analysis

This classification report shows that our model has a poor performance across both classes. For water tanks that do not meet standards (class 0.0), the model achieves poor precision (0.44) with a recall of 0.99, meaning it correctly identifies almost all non-compliant tanks.

For tanks that meet standards (class 1.0), the model shows slightly better performance with a precision of 0.57 and extremely poor recall (0.01).

What this means for our project:

- The model demonstrates a poor balance between classes - overfitting to the negative label, with an overall accuracy of 51% and a macro average F1-score of 0.28.
- The model is particularly poor at identifying compliant tanks (1% recall), incorrectly flagging safe tanks as problematic.
- When the model does flag a tank as non-compliant, we can be completely confident in that prediction (100% precision).
- For NYC water safety, this is a poor balance and should not be used in predictive metrics
- The weighted average F1-score of 0.28 indicates poor overall performance, taking into account the class distribution in our dataset.

```
best_features
```

```
['BOROUGH_BROOKLYN',
 'GI_RESULT_ACCESS_LADDERS',
 'DELETED',
 'COLIFORM',
 'BOROUGH_MANHATTAN',
 'INSPECTION_PERFORMED',
 'ECOLI',
 'BOROUGH_QUEENS',
 'GI_RESULT_INTERNAL_STRUCTURE',
 'BIN',
 'SI_RESULT_SEDIMENT',
 'GI_REQ_EXTERNAL_STRUCTURE',
 'BOROUGH_STATENISLAND',
 'LONGITUDE',
 'Unnamed_0',
 'NYS_CERTIFIED']
```

Feature Importance Analysis

- **Coliform and E Coli presence** are some of the most influential factor in predicting whether a tank meets standards. This suggests that the presence of these bacteria is highly indicative of water quality issues and non-compliance.
- All other features including tank number, certification status, time data, and various inspection results have minimal impact on the model's predictions compared to our best features.
- Geographic and structural features contribute to the model's decision-making process in this dataset as well, with access to ladders, latitude and longitude, external structure status, etc. making an impact

What this means for NYC water tank inspections:

- Inspections should prioritize tanks with coliform or E. coli detections, as these are the strongest indicators of noncompliance.
- Further investigation may be needed to understand why structural and geographic features have such importance, and whether additional data or different features could improve the model's ability to identify at-risk tanks.