Garbage_Bin_Level_Preciction_and_Collection

March 1, 2024

1 Importing required libraries

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, confusion_matrix
from sklearn.metrics import mean_squared_error
```

2 Suppress warnings

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

3 Load data

[3]:
```
df=pd.read_csv("C:/Users/geeth/Downloads/trash data PS-2 (5).csv")
df
```

[3]:

	BIN ID	Date	TIME	WEEK NO	FILL LEVEL(IN LITRES)	\
0	BIN 1	10/1/2021	12:00:00 AM	1	5	
1	BIN 1	10/1/2021	01:00:00 AM	1	29	
2	BIN 1	10/1/2021	02:00:00 AM	1	53	
3	BIN 1	10/1/2021	03:00:00 AM	1	77	
4	BIN 1	10/1/2021	04:00:00 AM	1	101	
…	…	…	…	…	…	
11036	BIN 5	12/31/2021	07:00:00 PM	5	480	
11037	BIN 5	12/31/2021	08:00:00 PM	5	504	
11038	BIN 5	12/31/2021	09:00:00 PM	5	528	
11039	BIN 5	12/31/2021	10:00:00 PM	5	552	

```
11040  BIN 5  12/31/2021  11:00:00 PM        5                    576

       TOTAL(LITRES) FILL PERCENTAGE   LOCATION     LATITUDE   LONGITUDE  \
0                660              1%  MANAPAKKAM  13.0213° N  80.1832° E
1                660              4%  MANAPAKKAM  13.0213° N  80.1832° E
2                660              8%  MANAPAKKAM  13.0213° N  80.1832° E
3                660             12%  MANAPAKKAM  13.0213° N  80.1832° E
4                660             15%  MANAPAKKAM  13.0213° N  80.1832° E
…              …               …          …          …          …
11036            660             73%     T-NAGAR  13.0418° N  80.2341° E
11037            660             76%     T-NAGAR  13.0418° N  80.2341° E
11038            660             80%     T-NAGAR  13.0418° N  80.2341° E
11039            660             84%     T-NAGAR  13.0418° N  80.2341° E
11040            660             87%     T-NAGAR  13.0418° N  80.2341° E

      TEMPERATURE( IN  C) BATTERY LEVEL   FILL LEVEL INDICATOR(Above 550)
0                      24           100%                             False
1                    24.6           100%                             False
2                    24.6           100%                             False
3                    24.8           100%                             False
4                      25           100%                             False
…                    …            …                                …
11036                37.8            87%                             False
11037                37.5            87%                             False
11038                  37            87%                             False
11039                36.5            87%                              True
11040                  32            87%                              True

[11041 rows x 13 columns]
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11041 entries, 0 to 11040
Data columns (total 13 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   BIN ID                          11041 non-null  object
 1   Date                            11041 non-null  object
 2   TIME                            11041 non-null  object
 3   WEEK NO                         11041 non-null  int64
 4   FILL LEVEL(IN LITRES)           11041 non-null  int64
 5   TOTAL(LITRES)                   11041 non-null  int64
 6   FILL PERCENTAGE                 11041 non-null  object
 7   LOCATION                        11041 non-null  object
 8   LATITUDE                        11041 non-null  object
 9   LONGITUDE                       11041 non-null  object
 10  TEMPERATURE( IN  C)             11041 non-null  object
```

```
 11  BATTERY LEVEL                     11041 non-null  object
 12  FILL LEVEL INDICATOR(Above 550)   11041 non-null  bool
dtypes: bool(1), int64(3), object(9)
memory usage: 1.0+ MB
```

[5]:
```
df.isnull().sum()
```

[5]:
```
BIN ID                            0
Date                              0
TIME                              0
WEEK NO                           0
FILL LEVEL(IN LITRES)             0
TOTAL(LITRES)                     0
FILL PERCENTAGE                   0
LOCATION                          0
LATITUDE                          0
LONGITUDE                         0
TEMPERATURE( IN  C)               0
BATTERY LEVEL                     0
FILL LEVEL INDICATOR(Above 550)   0
dtype: int64
```

[6]:
```
df.nunique()
```

[6]:
```
BIN ID                              5
Date                               92
TIME                               24
WEEK NO                             5
FILL LEVEL(IN LITRES)             682
TOTAL(LITRES)                       1
FILL PERCENTAGE                   124
LOCATION                            5
LATITUDE                            5
LONGITUDE                           5
TEMPERATURE( IN  C)                51
BATTERY LEVEL                      14
FILL LEVEL INDICATOR(Above 550)     2
dtype: int64
```

4 Data Exploration

4.1 Check columns

[7]:
```
cat_cols=df.select_dtypes(include=['object']).columns

num_cols = df.select_dtypes (include=np.number).columns.tolist()

print("Categorical Variables:")
```

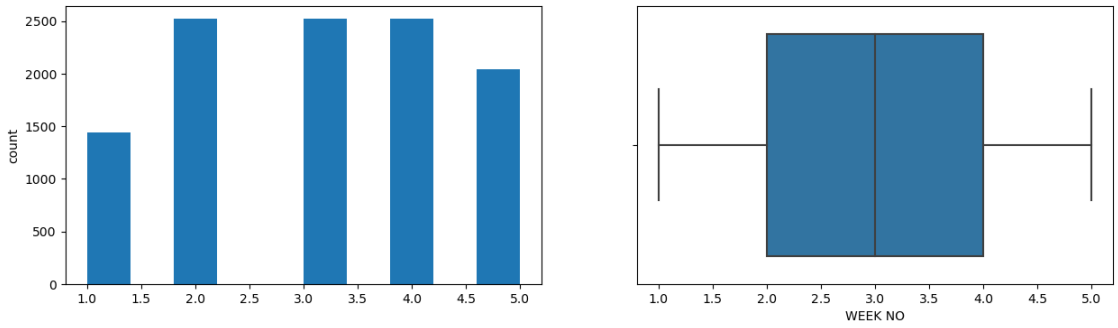
```
print(cat_cols)

print("Numerical Variables:")

print(num_cols)
```

```
Categorical Variables:
Index(['BIN ID', 'Date', 'TIME', 'FILL PERCENTAGE', 'LOCATION ', 'LATITUDE',
       'LONGITUDE', 'TEMPERATURE( IN  C)', 'BATTERY LEVEL '],
      dtype='object')
Numerical Variables:
['WEEK NO', 'FILL LEVEL(IN LITRES)', 'TOTAL(LITRES)']
```

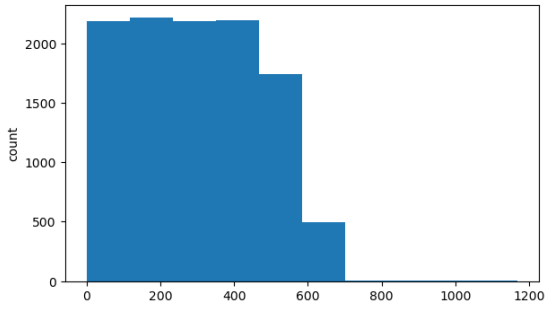
4.2 Explore numerical variables

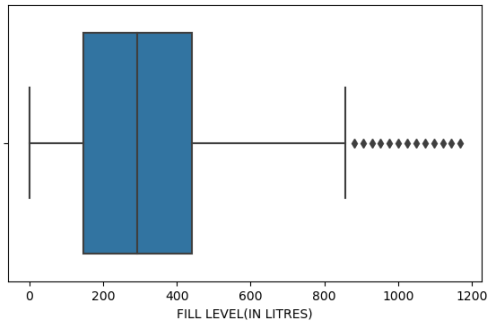
[8]:
```
for col in num_cols:
    print(col)
    print('Skew:', round(df[col].skew(), 2))
    plt.figure(figsize = (15, 4))
    plt.subplot(1, 2, 1)
    df[col].hist(grid=False)
    plt.ylabel('count')
    plt.subplot(1, 2, 2)
    sns.boxplot(x=df[col])
    plt.show()
```

```
WEEK NO
Skew: -0.06
```

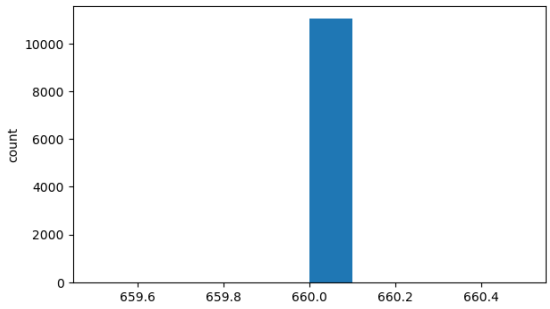


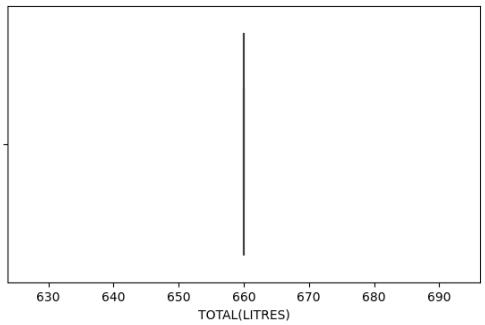

```
FILL LEVEL(IN LITRES)
Skew: 0.17
```





TOTAL(LITRES)
Skew: 0



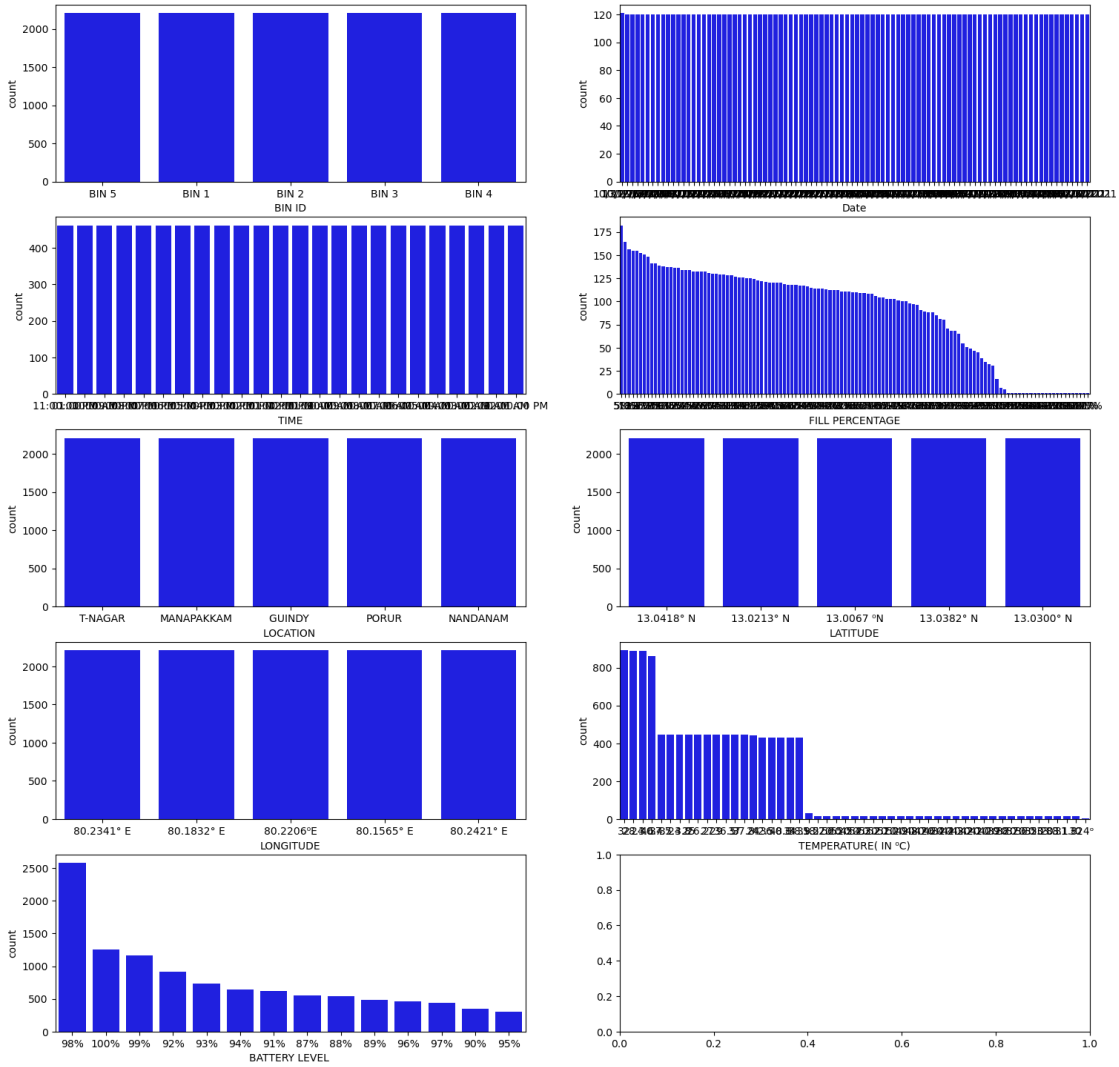


4.3 Explore categorical variables

[9]:
```
fig, axes = plt.subplots (5, 2, figsize = (18, 18))

fig.suptitle('Bar plot for all categorical variables in the dataset',␣
 ↪fontsize=24)
sns.countplot(ax = axes[0, 0], x = 'BIN ID', data = df, color = 'blue', order =␣
 ↪df['BIN ID'].value_counts().index);
sns.countplot(ax = axes[0, 1], x = 'Date', data = df, color = 'blue', order =␣
 ↪df['Date'].value_counts().index);
sns.countplot(ax = axes[1, 0], x = 'TIME', data = df, color = 'blue', order =␣
 ↪df['TIME'].value_counts().index);
sns.countplot(ax = axes[1, 1], x = 'FILL PERCENTAGE', data = df, color =␣
 ↪'blue', order = df['FILL PERCENTAGE'].value_counts().index);
sns.countplot(ax = axes[2, 0], x = 'LOCATION ', data = df, color = 'blue',␣
 ↪order = df['LOCATION '].value_counts().index);
sns.countplot(ax = axes[2, 1], x = 'LATITUDE', data = df, color = 'blue', order␣
 ↪= df['LATITUDE'].value_counts().index);
```

```
sns.countplot(ax = axes[3, 0], x = 'LONGITUDE', data = df, color = 'blue',␣
 ↪order = df['LONGITUDE'].value_counts().index);
sns.countplot(ax = axes[3, 1], x = 'TEMPERATURE( IN  C)' , data = df, color =␣
 ↪'blue', order = df['TEMPERATURE( IN  C)'].value_counts().index);
sns.countplot(ax = axes[4, 0], x = 'BATTERY LEVEL ', data = df, color = 'blue',␣
 ↪order = df['BATTERY LEVEL '].value_counts().index);
```

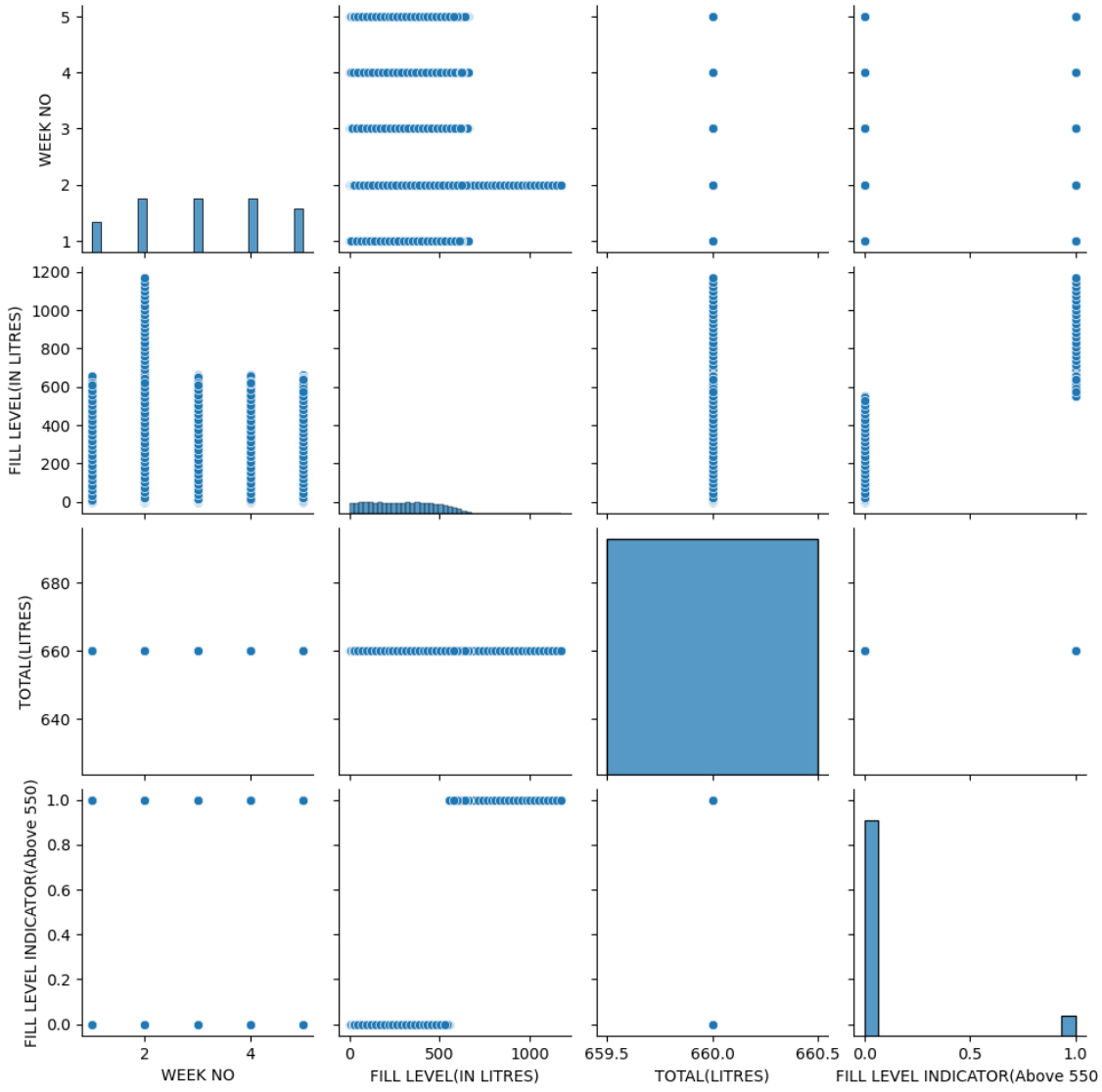
Bar plot for all categorical variables in the dataset



5 Pairplot

[10]:
```
plt.figure(figsize=(15,20))

sns.pairplot(df)

plt.show()
```

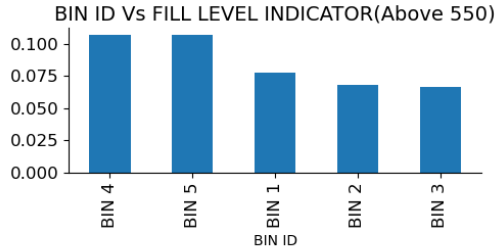
```
<Figure size 1500x2000 with 0 Axes>
```

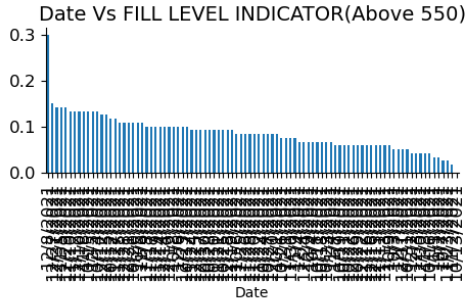


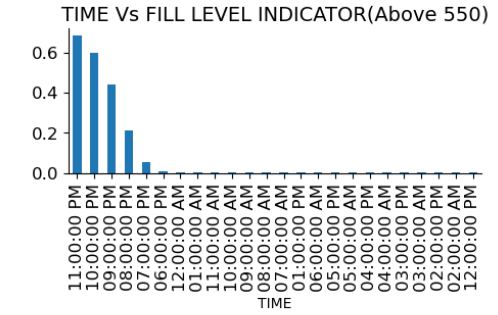
6 Grouped bar plots

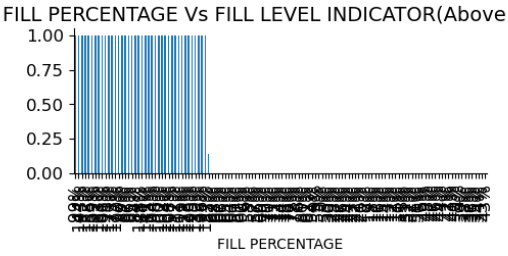
[11]:
```
fig, axarr= plt.subplots(5, 2, figsize=(12, 18))

df.groupby('BIN ID')['FILL LEVEL INDICATOR(Above 550)'].mean().
 ↪sort_values(ascending=False).plot.bar(ax=axarr[0][0], fontsize=12)
axarr[0][0].set_title("BIN ID Vs FILL LEVEL INDICATOR(Above 550)", fontsize=14)

df.groupby('Date')['FILL LEVEL INDICATOR(Above 550)'].mean().
 ↪sort_values(ascending=False).plot.bar(ax=axarr[0][1], fontsize=12)
axarr[0][1].set_title("Date Vs FILL LEVEL INDICATOR(Above 550)", fontsize=14)

df.groupby('TIME')['FILL LEVEL INDICATOR(Above 550)'].mean().
 ↪sort_values(ascending=False).plot.bar(ax=axarr[1][0], fontsize=12)
axarr[1][0].set_title("TIME Vs FILL LEVEL INDICATOR(Above 550)", fontsize=14)

df.groupby('FILL PERCENTAGE')['FILL LEVEL INDICATOR(Above 550)'].mean().
 ↪sort_values(ascending=False).plot.bar(ax=axarr[1][1], fontsize=12)
axarr[1][1].set_title("FILL PERCENTAGE Vs FILL LEVEL INDICATOR(Above 550)",␣
 ↪fontsize=14)

df.groupby('LOCATION ')['FILL LEVEL INDICATOR(Above 550)'].mean().
 ↪sort_values(ascending=False).head(10).plot.bar(ax=axarr[2][0], fontsize=12)
axarr[2][0].set_title("LOCATION Vs FILL LEVEL INDICATOR(Above 550)",␣
 ↪fontsize=14)

df.groupby('LATITUDE')['FILL LEVEL INDICATOR(Above 550)'].mean().
 ↪sort_values(ascending=False).plot.bar(ax=axarr[2][1], fontsize=12)
axarr[2][1].set_title("LATITUDE Vs FILL LEVEL INDICATOR(Above 550)",␣
 ↪fontsize=14)

df.groupby('LONGITUDE')['FILL LEVEL INDICATOR(Above 550)'].mean().
 ↪sort_values(ascending=False).plot.bar(ax=axarr[3][0], fontsize=12)
axarr[3][0].set_title("LONGITUDE Vs FILL LEVEL INDICATOR(Above 550)",␣
 ↪fontsize=14)

df.groupby('TEMPERATURE( IN  C)')['FILL LEVEL INDICATOR(Above 550)'].mean().
 ↪sort_values(ascending=False).plot.bar(ax=axarr[3][1], fontsize=12)
axarr[3][1].set_title("TEMPERATURE( IN  C) Vs FILL LEVEL INDICATOR(Above 550)",␣
 ↪fontsize=14)

df.groupby('BATTERY LEVEL ')['FILL LEVEL INDICATOR(Above 550)'].mean().
 ↪sort_values(ascending=False).plot.bar(ax=axarr[4][0], fontsize=12)
axarr[4][0].set_title("BATTERY LEVEL  Vs FILL LEVEL INDICATOR(Above 550)",␣
 ↪fontsize=14)
```

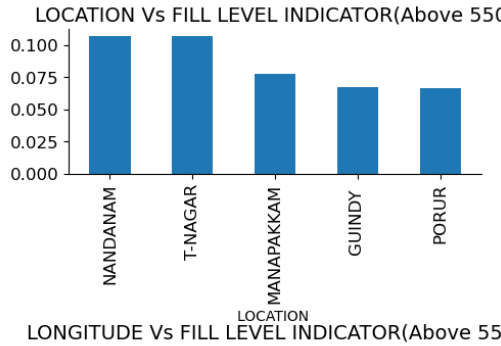
```
plt.subplots_adjust(hspace=1.2)

plt.subplots_adjust(wspace=.3)

sns.despine()
```

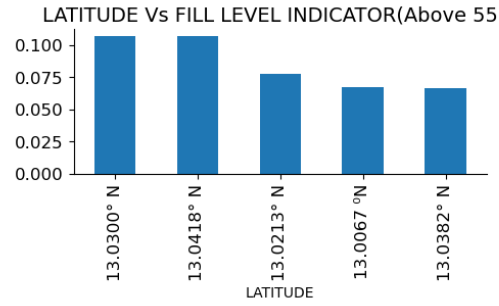

BIN ID Vs FILL LEVEL INDICATOR(Above 550)
0.100
0.075
0.050
0.025
0.000
BIN 4
BIN 5
BIN 1
BIN 2
BIN 3
BIN ID

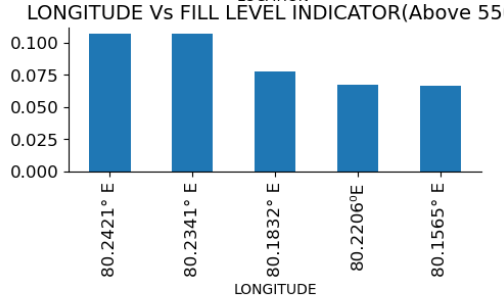

Date Vs FILL LEVEL INDICATOR(Above 550)
0.3
0.2
0.1
0.0
Date

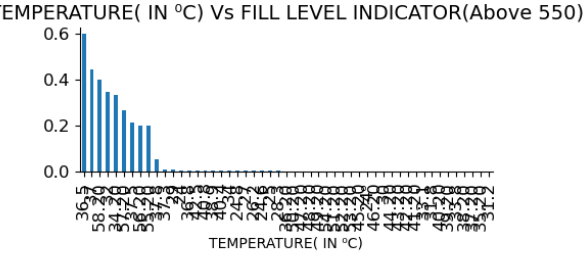

TIME Vs FILL LEVEL INDICATOR(Above 550)
0.6
0.4
0.2
0.0
11:00:00 PM
10:00:00 PM
09:00:00 PM
08:00:00 PM
07:00:00 PM
06:00:00 PM
12:00:00 AM
01:00:00 AM
11:00:00 AM
10:00:00 AM
09:00:00 AM
08:00:00 AM
07:00:00 AM
01:00:00 PM
06:00:00 AM
05:00:00 PM
05:00:00 AM
04:00:00 PM
04:00:00 AM
03:00:00 PM
03:00:00 AM
02:00:00 PM
02:00:00 AM
12:00:00 PM
TIME

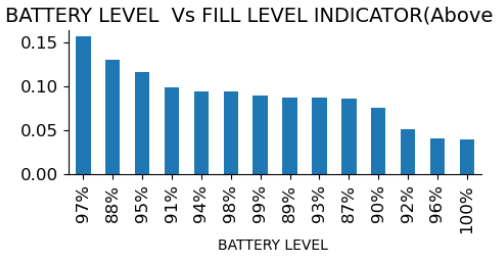

FILL PERCENTAGE Vs FILL LEVEL INDICATOR(Above 550)
1.00
0.75
0.50
0.25
0.00
FILL PERCENTAGE

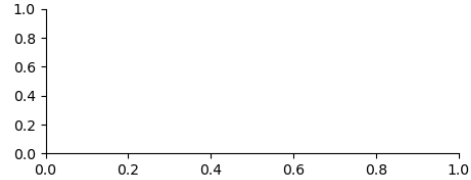

LOCATION Vs FILL LEVEL INDICATOR(Above 550)
0.100
0.075
0.050
0.025
0.000
NANDANAM
T-NAGAR
MANAPAKKAM
GUINDY
PORUR
LOCATION


LATITUDE Vs FILL LEVEL INDICATOR(Above 550)
0.100
0.075
0.050
0.025
0.000
13.0300° N
13.0418° N
13.0213° N
13.0067 °N
13.0382° N
LATITUDE


LONGITUDE Vs FILL LEVEL INDICATOR(Above 550)
0.100
0.075
0.050
0.025
0.000
80.2421° E
80.2341° E
80.1832° E
80.2206°E
80.1565° E
LONGITUDE


TEMPERATURE(IN °C) Vs FILL LEVEL INDICATOR(Above 550)
0.6
0.4
0.2
0.0
TEMPERATURE(IN °C)


BATTERY LEVEL Vs FILL LEVEL INDICATOR(Above 550)
0.15
0.10
0.05
0.00
97%
88%
95%
91%
94%
98%
99%
89%
93%
87%
90%
92%
96%
100%
BATTERY LEVEL


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

7 Encode categorical variables

[12]:
```
# Define the columns to encode
columns = ['BIN ID','Date','TIME', 'LOCATION ', 'LATITUDE', 'LONGITUDE','FILL␣
 ↪LEVEL INDICATOR(Above 550)','FILL PERCENTAGE','BATTERY LEVEL ','TEMPERATURE(␣
 ↪IN  C)']

# Initialize the LabelEncoder
encoder = LabelEncoder()

# Encode each column separately
for column in columns:
    df[column] = encoder.fit_transform(df[column])

# Now df contains the encoded values for each column
df
```

[12]:

	BIN ID	Date	TIME	WEEK NO	FILL LEVEL(IN LITRES)	TOTAL(LITRES)	\
0	0	0	22	1	5	660	
1	0	0	0	1	29	660	
2	0	0	2	1	53	660	
3	0	0	4	1	77	660	
4	0	0	6	1	101	660	
…	…	…	…	…	…	…	
11036	4	85	13	5	480	660	
11037	4	85	15	5	504	660	
11038	4	85	17	5	528	660	
11039	4	85	19	5	552	660	
11040	4	85	21	5	576	660	

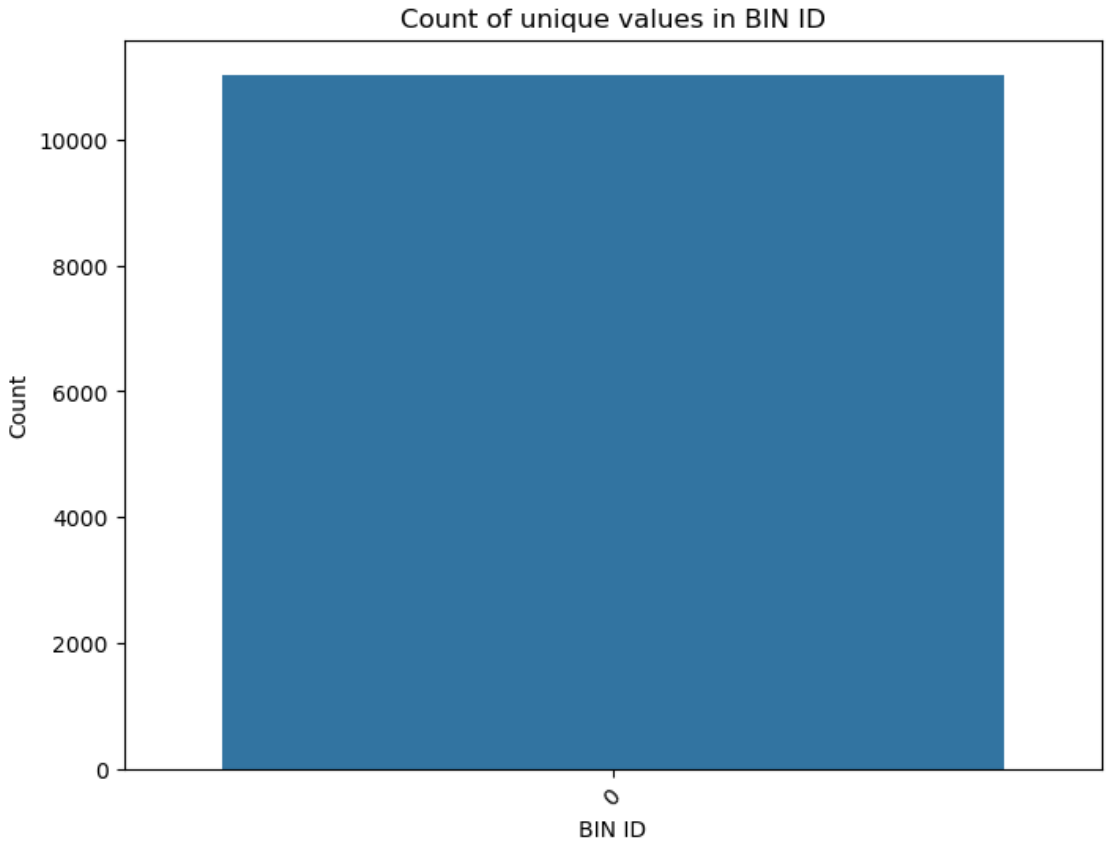
	FILL PERCENTAGE	LOCATION	LATITUDE	LONGITUDE	TEMPERATURE(IN C)	\
0	1	1	1	1	0	
1	58	1	1	1	1	
2	102	1	1	1	1	
3	12	1	1	1	2	
4	23	1	1	1	4	
…	…	…	…	…	…	
11036	95	4	4	3	25	
11037	98	4	4	3	24	
11038	103	4	4	3	21	
11039	107	4	4	3	19	
11040	110	4	4	3	12	

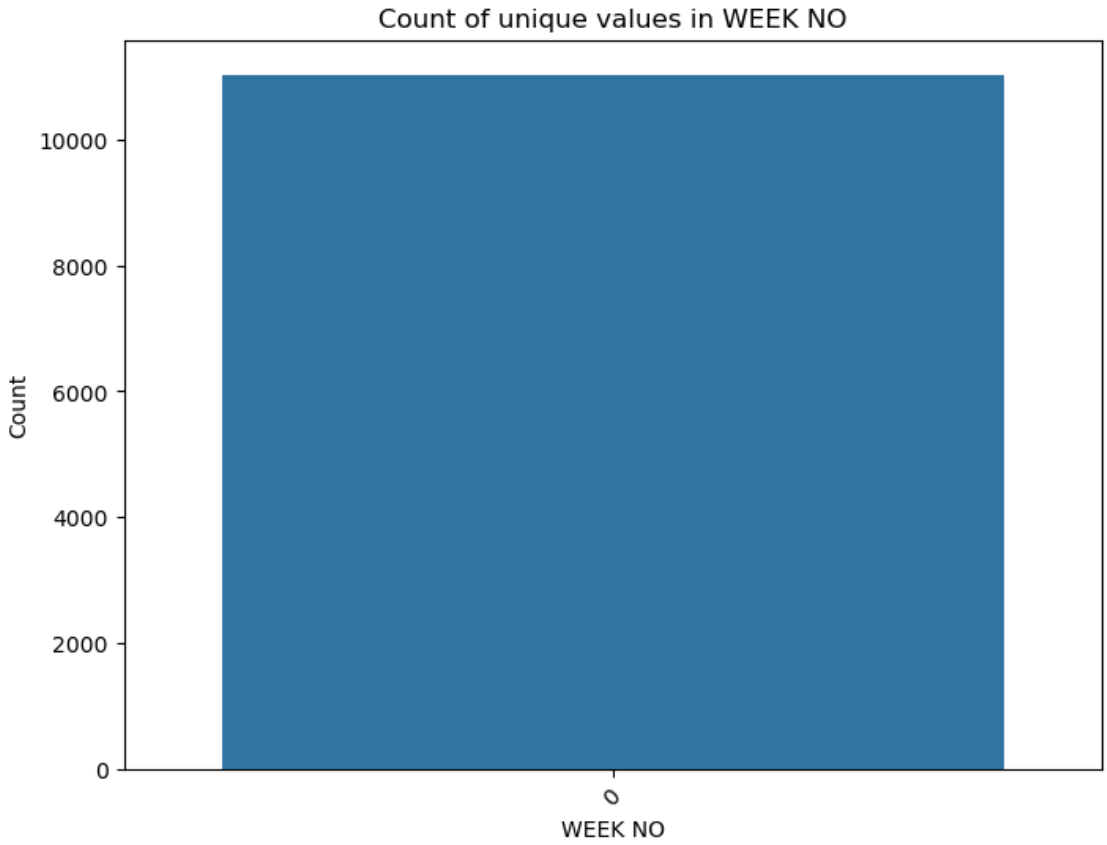
	BATTERY LEVEL	FILL LEVEL INDICATOR(Above 550)
0	0	0
1	0	0
2	0	0

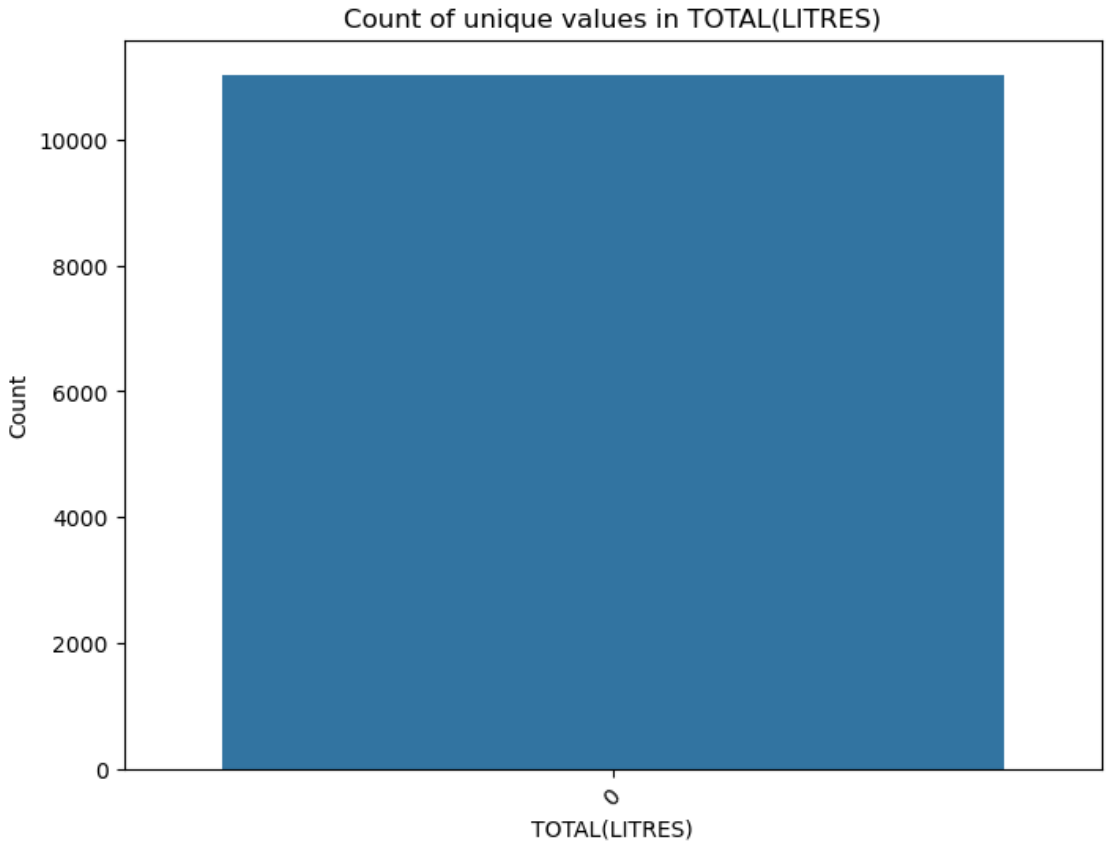
```
3                    0                                    0
4                    0                                    0
…                    …                                    …
11036                1                                    0
11037                1                                    0
11038                1                                    0
11039                1                                    1
11040                1                                    1

[11041 rows x 13 columns]
```

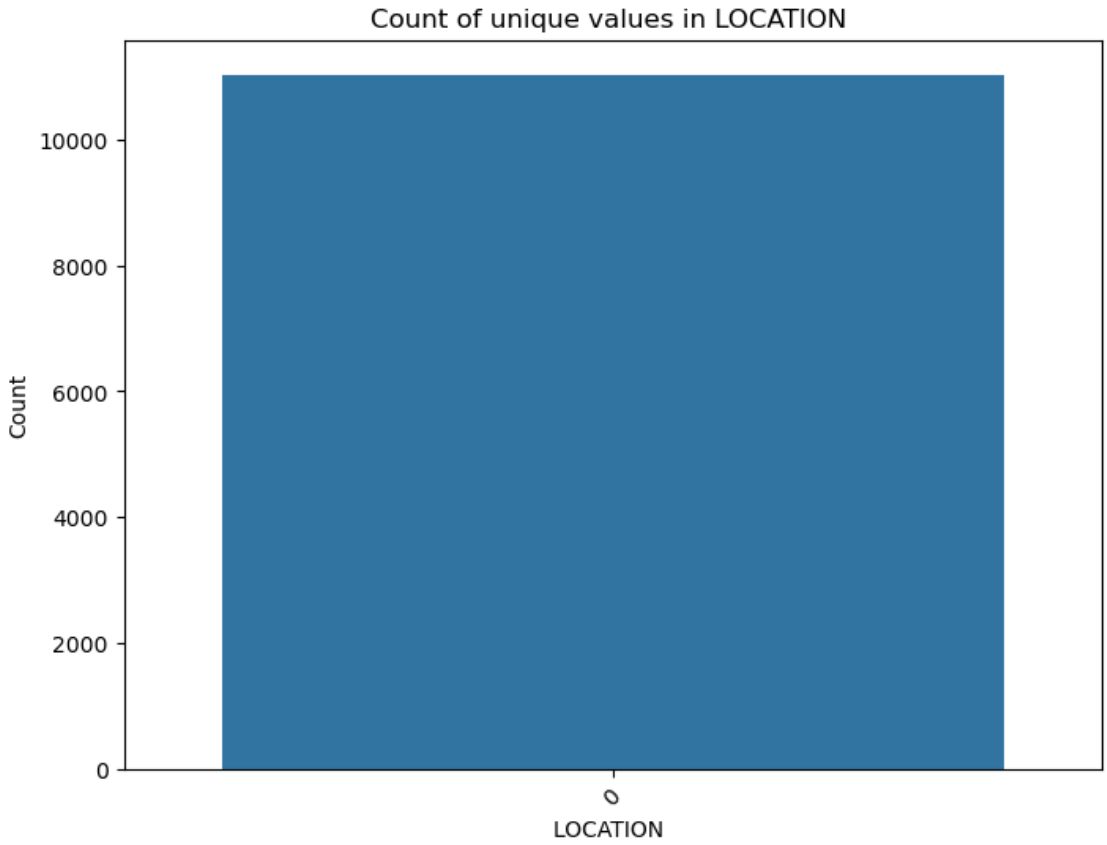
8 Count of unique values for categorical columns

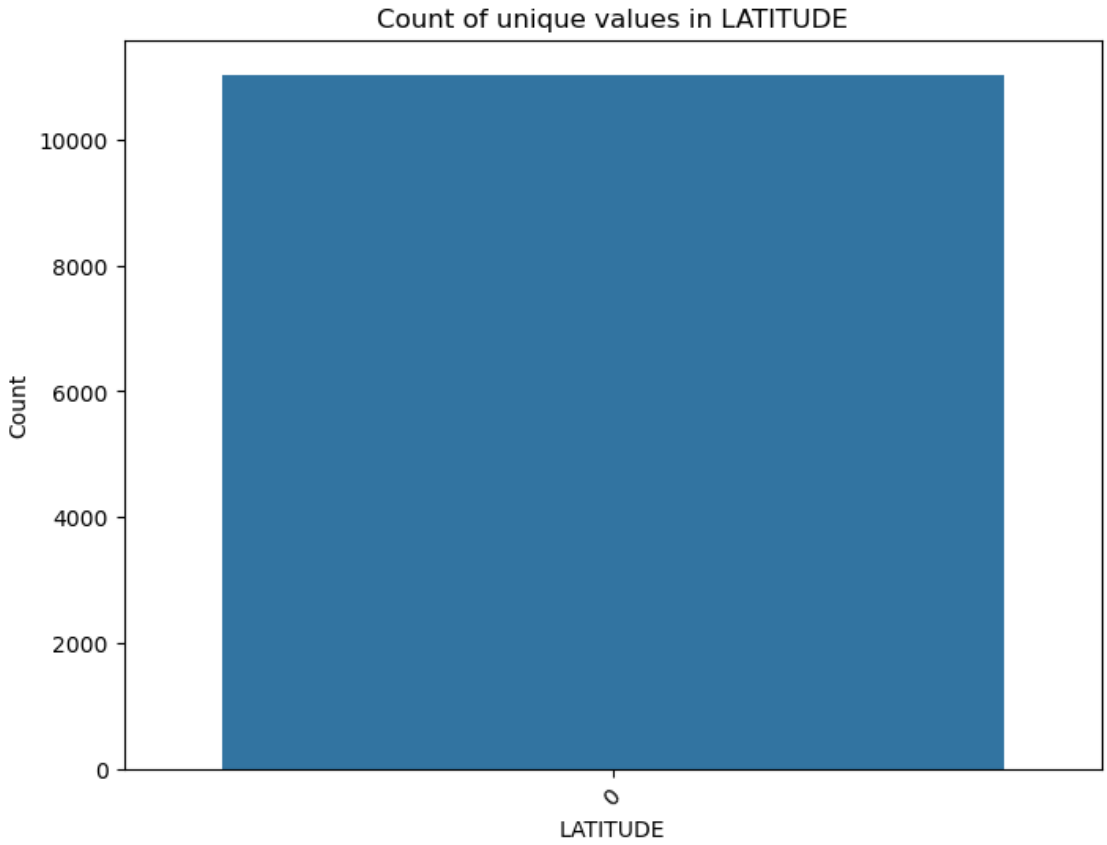
[13]:
```
categorical_columns = ['BIN ID','WEEK NO','TOTAL(LITRES)','LOCATION␣
 ↪','LATITUDE','LONGITUDE','FILL LEVEL INDICATOR(Above 550)']
for column in categorical_columns:
    plt.figure(figsize=(8, 6))
    sns.countplot(df[column])
    plt.title(f'Count of unique values in {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=45)
    plt.show()
```

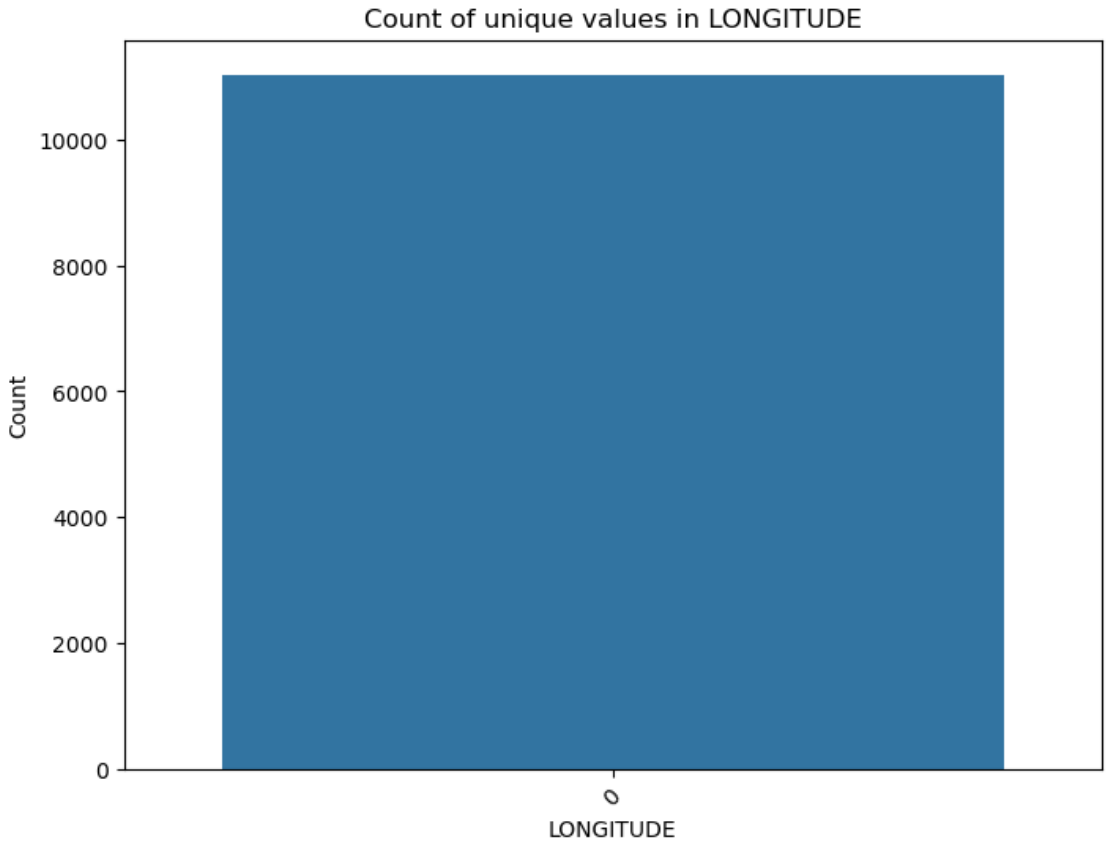

Count of unique values in BIN ID
10000
8000
6000
4000
2000
0
Count
0
BIN ID

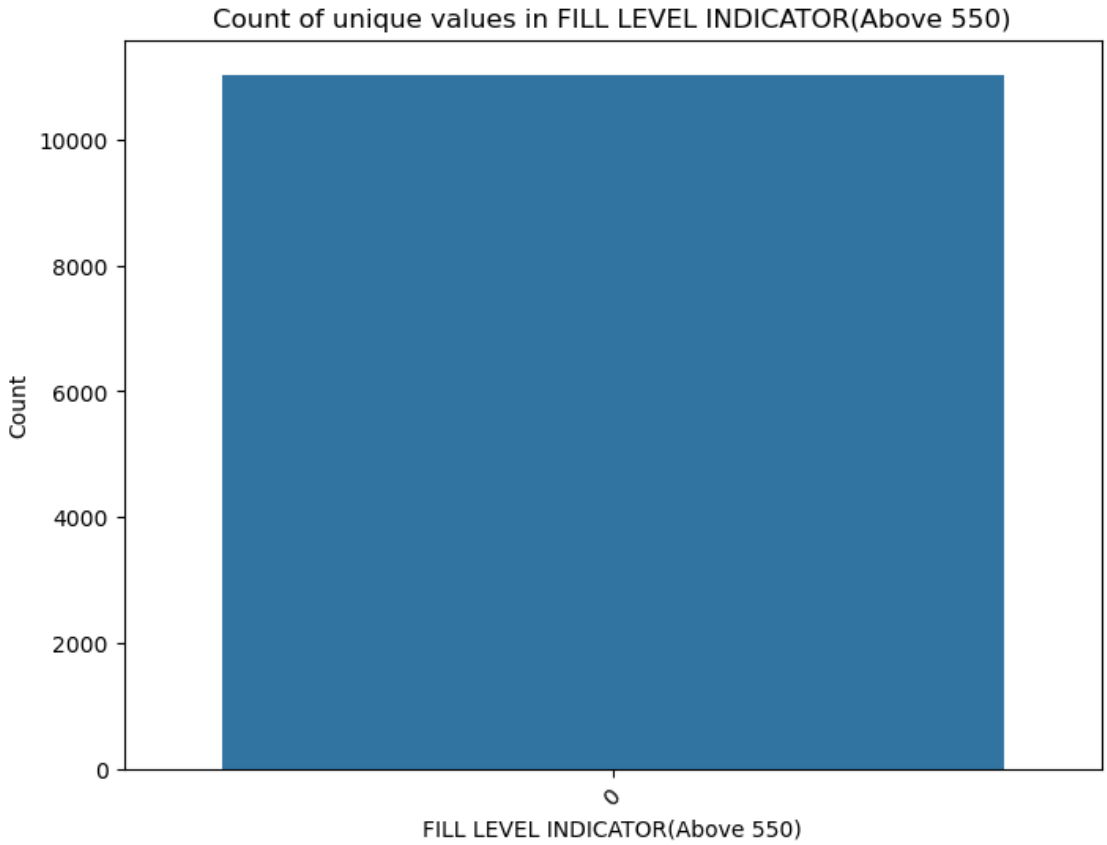

Count of unique values in WEEK NO
10000
8000
6000
4000
2000
0
Count
0
WEEK NO


Count of unique values in TOTAL(LITRES)
10000
8000
6000
4000
2000
0
Count
0
TOTAL(LITRES)


Count of unique values in LOCATION
10000
8000
6000
4000
2000
0
Count
0
LOCATION


Count of unique values in LATITUDE
10000
8000
6000
4000
2000
0
Count
0
LATITUDE


Count of unique values in LONGITUDE
10000
8000
6000
4000
2000
0
Count
0
LONGITUDE



[14]:
```
df.describe().T
```

[14]:

	count	mean	std	min	\
BIN ID	11041.0	2.000181	1.414342	0.0	
Date	11041.0	45.498053	26.557332	0.0	
TIME	11041.0	11.500860	6.922777	0.0	
WEEK NO	11041.0	3.109229	1.306071	1.0	
FILL LEVEL(IN LITRES)	11041.0	297.267458	176.612948	0.0	
TOTAL(LITRES)	11041.0	660.000000	0.000000	660.0	
FILL PERCENTAGE	11041.0	67.190653	29.388646	0.0	
LOCATION	11041.0	2.000181	1.414342	0.0	
LATITUDE	11041.0	2.000181	1.414342	0.0	
LONGITUDE	11041.0	2.000091	1.414246	0.0	
TEMPERATURE(IN C)	11041.0	15.733720	11.397134	0.0	
BATTERY LEVEL	11041.0	7.393533	4.537516	0.0	
FILL LEVEL INDICATOR(Above 550)	11041.0	0.085047	0.278964	0.0	

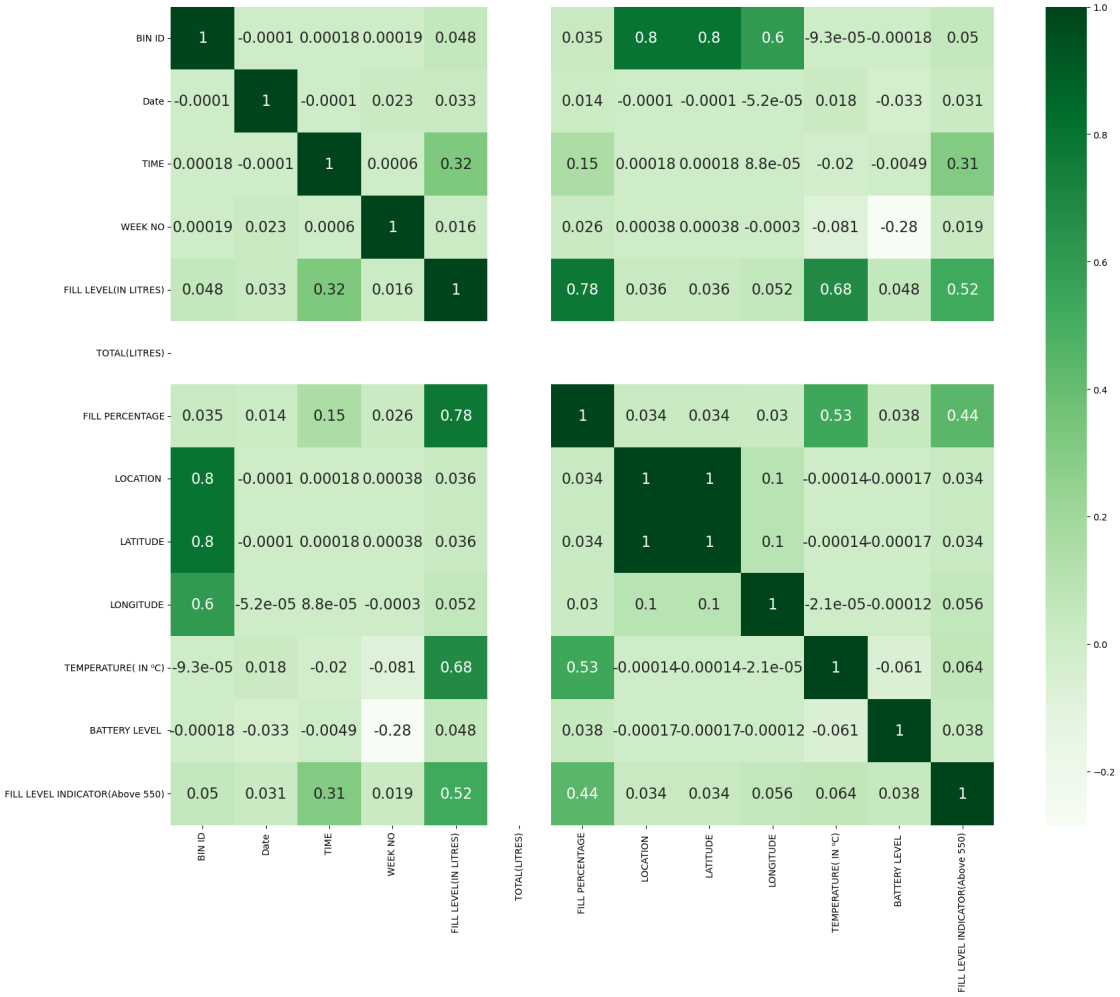
	25%	50%	75%	max
BIN ID	1.0	2.0	3.0	4.0
Date	23.0	45.0	68.0	91.0

TIME	6.0	12.0	18.0	23.0
WEEK NO	2.0	3.0	4.0	5.0
FILL LEVEL(IN LITRES)	147.0	293.0	440.0	1169.0
TOTAL(LITRES)	660.0	660.0	660.0	660.0
FILL PERCENTAGE	46.0	69.0	91.0	123.0
LOCATION	1.0	2.0	3.0	4.0
LATITUDE	1.0	2.0	3.0	4.0
LONGITUDE	1.0	2.0	3.0	4.0
TEMPERATURE(IN C)	6.0	15.0	25.0	50.0
BATTERY LEVEL	3.0	8.0	12.0	13.0
FILL LEVEL INDICATOR(Above 550)	0.0	0.0	0.0	1.0

9 Heatmap of correlations

[15]:
```
fig, ax = plt.subplots(figsize=(20,16))
sns.heatmap(df.corr(), annot=True, cmap='Greens',annot_kws={"fontsize":16})
```

[15]: <Axes: >



10 Modeling

10.1 Split data

[16]:
```
x=df.drop(['BIN ID','Date','TIME','WEEK NO','TOTAL(LITRES)','BATTERY LEVEL␣
 ↪','FILL LEVEL INDICATOR(Above 550)'],axis=1)
y=df['FILL LEVEL INDICATOR(Above 550)']
x
```

[16]:

	FILL LEVEL(IN LITRES)	FILL PERCENTAGE	LOCATION	LATITUDE	LONGITUDE	\
0	5	1	1	1	1	
1	29	58	1	1	1	
2	53	102	1	1	1	
3	77	12	1	1	1	
4	101	23	1	1	1	
…	…	…	…	…	…	
11036	480	95	4	4	3	
11037	504	98	4	4	3	
11038	528	103	4	4	3	
11039	552	107	4	4	3	
11040	576	110	4	4	3	

	TEMPERATURE(IN C)
0	0
1	1
2	1
3	2
4	4
…	…
11036	25
11037	24
11038	21
11039	19
11040	12

[11041 rows x 6 columns]

[17]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=0,test_size=0.2)
```

10.2 Logistic Regression

[18]:
```
lr=LogisticRegression(max_iter=1000)
lr.fit(x_train,y_train)
pred_1=lr.predict(x_test)
acc_1=accuracy_score(y_test,pred_1)
mse_lr = mean_squared_error(y_test, pred_1)
print("Accuracy:", acc_1)
print("Mean Squared Error:", mse_lr)
```

```
Accuracy: 1.0
Mean Squared Error: 0.0
```

10.3 Random Forest

[19]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
pred_2=rfc.predict(x_test)
acc_2=accuracy_score(y_test,pred_2)
mse_rfc = mean_squared_error(y_test, pred_2)
print("Accuracy:", acc_2)
print("Mean Squared Error:", mse_rfc)
```

```
Accuracy: 1.0
Mean Squared Error: 0.0
```

10.4 KNN

[20]:
```
best_k = None
best_acc_knn = 0
for i in range(1,21):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    preds=knn.predict(x_test)
    acc_3=accuracy_score(y_test,preds)
    mse_knn = mean_squared_error(y_test,preds)
    if acc_3 > best_acc_knn:
        best_acc_knn = acc_3
        best_k = i
    print("\nK-Nearest Neighbors (k =", i, "):")
    print("Accuracy:", acc_3)
    print("Mean Squared Error:", mse_knn)
```

```
K-Nearest Neighbors (k = 1 ):
Accuracy: 0.9990946129470348
Mean Squared Error: 0.0009053870529651426

K-Nearest Neighbors (k = 2 ):
```

```
Accuracy: 0.9995473064735174
Mean Squared Error: 0.0004526935264825713

K-Nearest Neighbors (k = 3 ):
Accuracy: 0.9995473064735174
Mean Squared Error: 0.0004526935264825713

K-Nearest Neighbors (k = 4 ):
Accuracy: 1.0
Mean Squared Error: 0.0

K-Nearest Neighbors (k = 5 ):
Accuracy: 1.0
Mean Squared Error: 0.0

K-Nearest Neighbors (k = 6 ):
Accuracy: 0.9990946129470348
Mean Squared Error: 0.0009053870529651426

K-Nearest Neighbors (k = 7 ):
Accuracy: 0.9986419194205522
Mean Squared Error: 0.001358080579447714

K-Nearest Neighbors (k = 8 ):
Accuracy: 0.9986419194205522
Mean Squared Error: 0.001358080579447714

K-Nearest Neighbors (k = 9 ):
Accuracy: 0.9990946129470348
Mean Squared Error: 0.0009053870529651426

K-Nearest Neighbors (k = 10 ):
Accuracy: 0.9995473064735174
Mean Squared Error: 0.0004526935264825713

K-Nearest Neighbors (k = 11 ):
Accuracy: 0.9986419194205522
Mean Squared Error: 0.001358080579447714

K-Nearest Neighbors (k = 12 ):
Accuracy: 1.0
Mean Squared Error: 0.0

K-Nearest Neighbors (k = 13 ):
Accuracy: 0.9990946129470348
Mean Squared Error: 0.0009053870529651426

K-Nearest Neighbors (k = 14 ):
```

```
Accuracy: 0.9990946129470348
Mean Squared Error: 0.0009053870529651426

K-Nearest Neighbors (k = 15 ):
Accuracy: 0.9990946129470348
Mean Squared Error: 0.0009053870529651426

K-Nearest Neighbors (k = 16 ):
Accuracy: 1.0
Mean Squared Error: 0.0

K-Nearest Neighbors (k = 17 ):
Accuracy: 1.0
Mean Squared Error: 0.0

K-Nearest Neighbors (k = 18 ):
Accuracy: 1.0
Mean Squared Error: 0.0

K-Nearest Neighbors (k = 19 ):
Accuracy: 1.0
Mean Squared Error: 0.0

K-Nearest Neighbors (k = 20 ):
Accuracy: 1.0
Mean Squared Error: 0.0
```

10.5 Custom Linear Regression

[21]:
```python
import numpy as np
from sklearn.metrics import mean_squared_error

class CustomLinearRegression:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.theta = None

    def fit(self, x, y):
        # Add bias term to input features
        ones = np.ones((x.shape[0], 1))
        x = np.concatenate((ones, x), axis=1)

        # Initialize parameters randomly
        np.random.seed(0)
        self.theta = np.random.rand(x.shape[1])
```

```
        # Gradient descent
        for _ in range(self.num_iterations):
            # Compute gradients
            gradients = np.dot(x.T, np.dot(x, self.theta) - y) / x.shape[0]

            # Update parameters
            self.theta -= self.learning_rate * gradients

    def predict(self, x):
        # Add bias term to input features
        ones = np.ones((x.shape[0], 1))
        x = np.concatenate((ones, x), axis=1)

        # Predict target variable
        return np.dot(x, self.theta)
    def score(self, X, y):
        # Calculate R^2 score or any other metric you want to use for scoring
        predictions = self.predict(X)
        return your_custom_scoring_function(y, predictions)
    def get_params(self, deep=True):
        return {'learning_rate': self.learning_rate, 'num_iterations': self.
 ↪num_iterations}

# Example usage:
# Assuming x_train, y_train, x_test, y_test are your training and testing data
custom_lr = CustomLinearRegression(learning_rate=0.01, num_iterations=1000)
custom_lr.fit(x_train, y_train)
predictions = custom_lr.predict(x_test)

# Handle NaN values
predictions = np.nan_to_num(predictions)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error (MSE):", mse)

# Calculate Accuracy Score
threshold = 0.5  # Example threshold
predicted_classes = (predictions >= threshold).astype(int)
true_classes = (y_test >= threshold).astype(int)
accuracy = (predicted_classes == true_classes).mean()
print("Accuracy Score:", accuracy)
```

```
Mean Squared Error (MSE): 0.08782254413761884
Accuracy Score: 0.9121774558623812
```

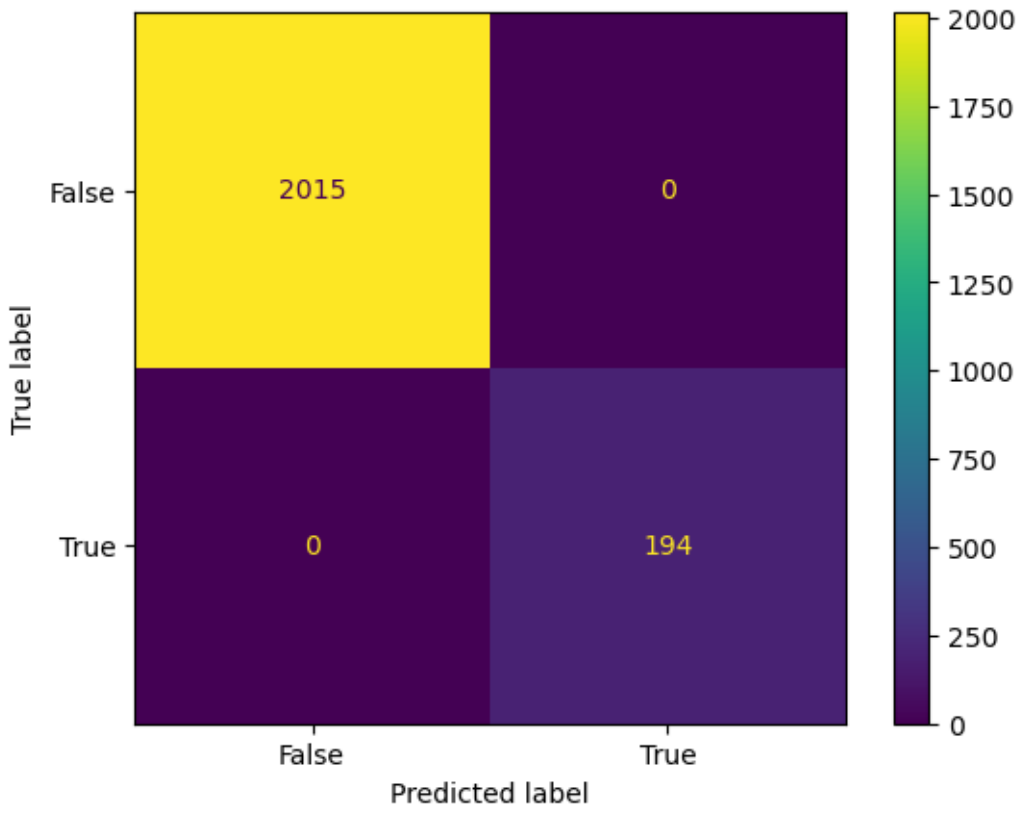
11 Cross-validation

[22]:
```
from sklearn.model_selection import cross_val_score
lr_cv_scores = cross_val_score(lr, x, y, cv=5)
print("Logistic Regression Cross-Validation Accuracy:", np.mean(lr_cv_scores))

rfc_cv_scores = cross_val_score(rfc, x, y, cv=5)
print("Random Forest Cross-Validation Accuracy:", np.mean(rfc_cv_scores))

knn_cv_scores = cross_val_score(knn, x, y, cv=5)
print(f"KNN Cross-Validation Accuracy with {best_k} neighbors:", np.
 ↪mean(knn_cv_scores))

# Cross-validation for Custom Linear Regression
custom_lr_cv_scores = cross_val_score(custom_lr, x, y, cv=5)
print("Custom Linear Regression Cross-Validation Accuracy:", np.
 ↪mean(custom_lr_cv_scores))
```

```
Logistic Regression Cross-Validation Accuracy: 1.0
Random Forest Cross-Validation Accuracy: 1.0
KNN Cross-Validation Accuracy with 4 neighbors: 0.9999094612947035
Custom Linear Regression Cross-Validation Accuracy: nan
```

12 Metrics and Evaluation

[23]:
```
from sklearn import metrics
confusion_matrix = metrics.confusion_matrix(y_test, preds)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =␣
 ↪confusion_matrix, display_labels = [False, True])
cm_display.plot()
plt.show()
```



[24]:
```
f1 = f1_score(y_test, preds)
print("F1 score:", f1)
```

```
F1 score: 1.0
```

[25]:
```
f1 = f1_score(y_test, preds)
print("F1 score:", f1)
```

```
F1 score: 1.0
```

[26]:
```
recall = recall_score(y_test, preds)
print("Recall: {:.2f}".format(recall))
```

```
Recall: 1.00
```

[27]:
```
new_bin_data = [[5, 58, 1, 1, 1, 25]]
prediction = knn.predict(new_bin_data)
if prediction[0]== 0:
    print("The bin didn't fill yet")
else:
    print("The bin is full")
```

```
The bin didn't fill yet
```