bhargav-assignment-solution

May 23, 2025

1 Bar Inventory Forecasting and Recommendation System

1.1 1. Import Libraries

[12]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
import warnings
warnings.filterwarnings("ignore")
```

2 2. Load and Inspect Data

[13]:
```
df=pd.read_excel('Dataset.xlsx')
df
```

[13]:
```
      Date Time Served        Bar Name Alcohol Type      Brand Name  \
0       1/1/2023 19:35     Smith's Bar          Rum  Captain Morgan
1       1/1/2023 10:07     Smith's Bar         Wine     Yellow Tail
2       1/1/2023 11:26   Johnson's Bar        Vodka      Grey Goose
3       1/1/2023 13:53   Johnson's Bar         Beer           Coors
4       1/1/2023 22:28   Johnson's Bar         Wine     Yellow Tail
…                    …               …           …               …
6570    1/1/2024 21:03  Anderson's Bar         Beer           Coors
6571    1/1/2024 21:15  Anderson's Bar          Rum          Malibu
6572    1/1/2024 18:34  Anderson's Bar      Whiskey    Jack Daniels
6573    1/1/2024 22:46    Thomas's Bar        Vodka         Absolut
6574    1/1/2024 21:26    Thomas's Bar          Rum         Bacardi

      Opening Balance (ml)  Purchase (ml)  Consumed (ml)  Closing Balance (ml)
0             2.555040e+03        1824.84           0.00               4379.88
```

```
1           1.344370e+03       0.00         0.00              1344.37
2           1.034280e+03       0.00         0.00              1034.28
3           2.194530e+03       0.00         0.00              2194.53
4           1.020900e+03       0.00         0.00              1020.90
…                    …           …            …                    …
6570        2.467080e+03       0.00       321.06              2146.02
6571        8.530000e-14    1743.64       175.05              1568.59
6572        4.192660e+03       0.00       197.60              3995.06
6573        2.424950e+03       0.00       128.52              2296.43
6574        1.778360e+03    1195.45       572.60              2401.21

[6575 rows x 8 columns]
```

3 3. Data Cleaning & Preprocessing

[14]:
```python
df['Date Time Served'] = pd.to_datetime(df['Date Time Served'])
df['Date'] = df['Date Time Served'].dt.date
```

4 3.1 information & Description of dataset of numerical columns

[15]:
```python
print("Inforamtion of dataset")
print("     ")
print(df.info())
print("Description of dataset")
df.describe()
```

```
Inforamtion of dataset

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6575 entries, 0 to 6574
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Date Time Served      6575 non-null   datetime64[ns]
 1   Bar Name              6575 non-null   object
 2   Alcohol Type          6575 non-null   object
 3   Brand Name            6575 non-null   object
 4   Opening Balance (ml)  6575 non-null   float64
 5   Purchase (ml)         6575 non-null   float64
 6   Consumed (ml)         6575 non-null   float64
 7   Closing Balance (ml)  6575 non-null   float64
 8   Date                  6575 non-null   object
dtypes: datetime64[ns](1), float64(4), object(4)
memory usage: 462.4+ KB
None
```

Description of dataset

[15]:

```
                      Date Time Served  Opening Balance (ml)  Purchase (ml)  \
count                             6575           6575.000000    6575.000000
mean   2023-07-01 09:09:52.033459968           2468.397180     315.841757
min              2023-01-01 10:07:00              0.000000       0.000000
25%              2023-03-30 19:35:00            619.135000       0.000000
50%              2023-06-30 18:08:00           1848.440000       0.000000
75%              2023-10-01 13:59:30           3853.020000     526.345000
max              2024-01-01 22:46:00          11862.520000    1999.840000
std                                NaN           2284.552895     582.120264

       Consumed (ml)  Closing Balance (ml)
count    6575.000000           6575.000000
mean      299.419264           2484.811748
min         0.000000              0.000000
25%       156.640000            611.000000
50%       300.390000           1849.840000
75%       450.870000           3906.050000
max      1180.580000          11862.520000
std       191.903874           2302.363298
```

5 3.2 Datatypes of dataframe(df)

[16]:

```python
print(df.dtypes)
```

```
Date Time Served        datetime64[ns]
Bar Name                        object
Alcohol Type                    object
Brand Name                      object
Opening Balance (ml)           float64
Purchase (ml)                  float64
Consumed (ml)                  float64
Closing Balance (ml)           float64
Date                            object
dtype: object
```

6 3.3 checking missing values in dataframe(df)

[17]:

```python
print(df.isnull().sum())
```

```
Date Time Served        0
Bar Name                0
Alcohol Type            0
Brand Name              0
Opening Balance (ml)    0
```

```
Purchase (ml)          0
Consumed (ml)          0
Closing Balance (ml)   0
Date                   0
dtype: int64
```

7 3.3 checking duplicate sum values in dataframe(df)

[18]:
```
df.duplicated().sum()
```

[18]:
```
np.int64(0)
```

8 4. Data Labeling & Standardization

[19]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['Bar Name']=le.fit_transform(df['Bar Name'])
df['Alcohol Type']=le.fit_transform(df['Alcohol Type'])
df['Brand Name']=le.fit_transform(df['Brand Name'])
df['Brand Name']=le.fit_transform(df['Brand Name'])
df
```

[19]:
```
        Date Time Served  Bar Name  Alcohol Type  Brand Name  \
0    2023-01-01 19:35:00         3             1           4
1    2023-01-01 10:07:00         3             4          15
2    2023-01-01 11:26:00         2             2           6
3    2023-01-01 13:53:00         2             0           5
4    2023-01-01 22:28:00         2             4          15
…                    …         …             …           …
6570 2024-01-01 21:03:00         0             0           5
6571 2024-01-01 21:15:00         0             1          11
6572 2024-01-01 18:34:00         0             3           8
6573 2024-01-01 22:46:00         5             2           0
6574 2024-01-01 21:26:00         5             1           1

      Opening Balance (ml)  Purchase (ml)  Consumed (ml)  \
0             2.555040e+03        1824.84           0.00
1             1.344370e+03           0.00           0.00
2             1.034280e+03           0.00           0.00
3             2.194530e+03           0.00           0.00
4             1.020900e+03           0.00           0.00
…                      …            …            …
6570          2.467080e+03           0.00         321.06
6571          8.530000e-14        1743.64         175.05
6572          4.192660e+03           0.00         197.60
```

```
6573          2.424950e+03          0.00          128.52
6574          1.778360e+03       1195.45          572.60

      Closing Balance (ml)        Date
0                  4379.88  2023-01-01
1                  1344.37  2023-01-01
2                  1034.28  2023-01-01
3                  2194.53  2023-01-01
4                  1020.90  2023-01-01
…                      …         …
6570               2146.02  2024-01-01
6571               1568.59  2024-01-01
6572               3995.06  2024-01-01
6573               2296.43  2024-01-01
6574               2401.21  2024-01-01

[6575 rows x 9 columns]
```

[20]:

```python
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
df['Opening Balance (ml)']=scaler.fit_transform(df[['Opening Balance (ml)']])
df['Closing Balance (ml)']=scaler.fit_transform(df[['Closing Balance (ml)']])
df['Purchase Quantity (ml)']=scaler.fit_transform(df[['Purchase (ml)']])
df['Sale Quantity (ml)']=scaler.fit_transform(df[['Consumed (ml)']])
df
```

[20]:

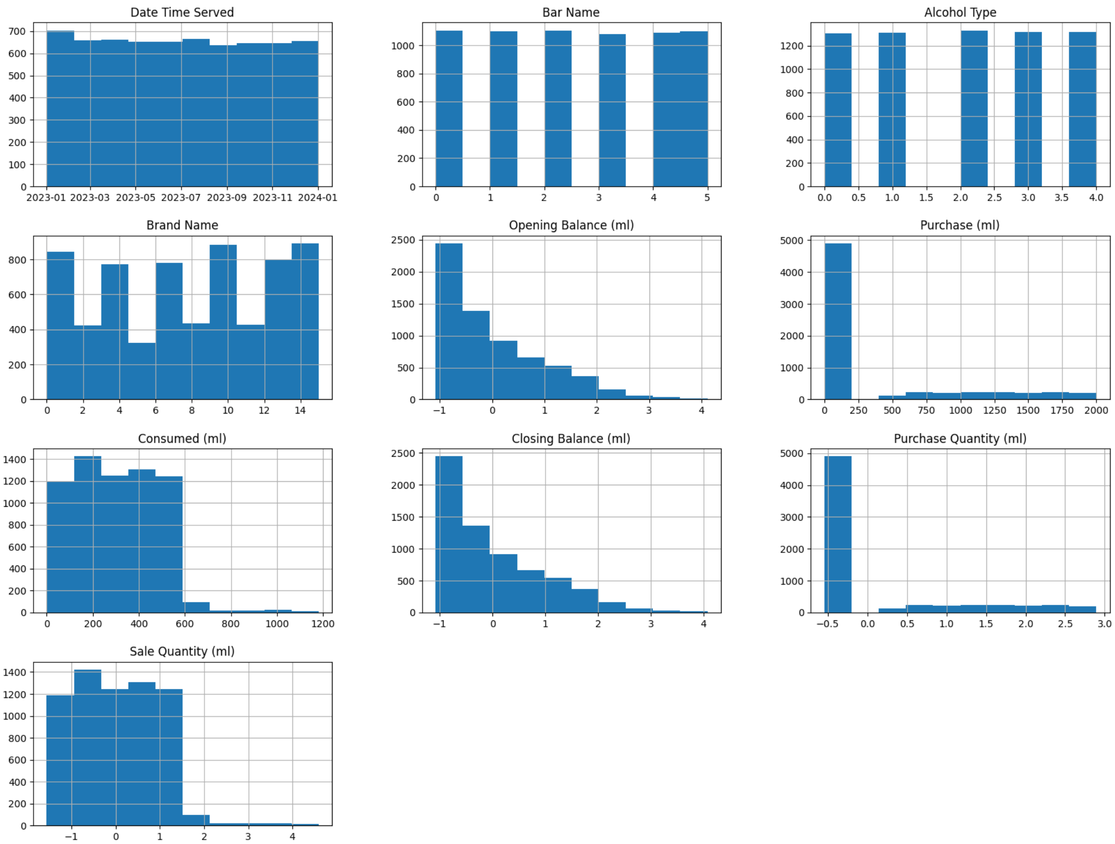
```
         Date Time Served  Bar Name  Alcohol Type  Brand Name  \
0    2023-01-01 19:35:00         3             1           4
1    2023-01-01 10:07:00         3             4          15
2    2023-01-01 11:26:00         2             2           6
3    2023-01-01 13:53:00         2             0           5
4    2023-01-01 22:28:00         2             4          15
…                    …        …            …          …
6570 2024-01-01 21:03:00         0             0           5
6571 2024-01-01 21:15:00         0             1          11
6572 2024-01-01 18:34:00         0             3           8
6573 2024-01-01 22:46:00         5             2           0
6574 2024-01-01 21:26:00         5             1           1

      Opening Balance (ml)  Purchase (ml)  Consumed (ml)  \
0                 0.037928        1824.84           0.00
1                -0.492049           0.00           0.00
2                -0.627793           0.00           0.00
3                -0.119887           0.00           0.00
4                -0.633650           0.00           0.00
…                      …            …            …
6570             -0.000577           0.00         321.06
```

```
6571             -1.080555       1743.64        175.05
6572              0.754806          0.00        197.60
6573             -0.019019          0.00        128.52
6574             -0.302068       1195.45        572.60

      Closing Balance (ml)        Date  Purchase Quantity (ml)  \
0                 0.823160  2023-01-01                2.592442
1                -0.495373  2023-01-01               -0.542613
2                -0.630067  2023-01-01               -0.542613
3                -0.126089  2023-01-01               -0.542613
4                -0.635878  2023-01-01               -0.542613
…                      …         …                     …
6570             -0.147161  2024-01-01               -0.542613
6571             -0.397979  2024-01-01                2.452941
6572              0.656006  2024-01-01               -0.542613
6573             -0.081827  2024-01-01               -0.542613
6574             -0.036314  2024-01-01                1.511157

      Sale Quantity (ml)
0              -1.560375
1              -1.560375
2              -1.560375
3              -1.560375
4              -1.560375
…                    …
6570            0.112777
6571           -0.648130
6572           -0.530615
6573           -0.890614
6574            1.423637

[6575 rows x 11 columns]
```

9 5. Exploratory Data Analysis (EDA)

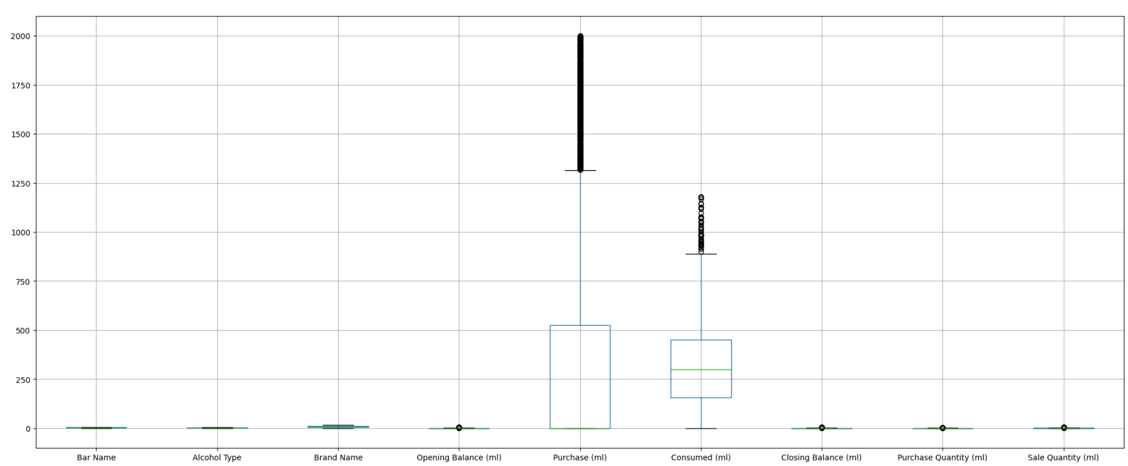
#5.1 Histogram plot of each column

[21]:
```
df.hist(figsize=(20,15))
plt.show()
```



#5.2 boxplot of each column

[22]:
```
df.boxplot(figsize=(25,10))
plt.show()
```



10 5.3 correlation matrix of each column to column

[23]:
```
df_name=df.drop('Date Time Served',axis=1)
df_name=df_name.drop('Date',axis=1)
corr=df_name.corr()
corr
```

[23]:

	Bar Name	Alcohol Type	Brand Name
Bar Name	1.000000	-0.000115	0.000775
Alcohol Type	-0.000115	1.000000	0.325360
Brand Name	0.000775	0.325360	1.000000
Opening Balance (ml)	-0.083647	0.110881	-0.004314
Purchase (ml)	-0.002936	0.000823	-0.020959
Consumed (ml)	-0.026739	-0.033068	-0.038064
Closing Balance (ml)	-0.081517	0.112988	-0.006405
Purchase Quantity (ml)	-0.002936	0.000823	-0.020959
Sale Quantity (ml)	-0.026739	-0.033068	-0.038064

	Opening Balance (ml)	Purchase (ml)	Consumed (ml)
Bar Name	-0.083647	-0.002936	-0.026739
Alcohol Type	0.110881	0.000823	-0.033068
Brand Name	-0.004314	-0.020959	-0.038064
Opening Balance (ml)	1.000000	-0.015837	0.258866
Purchase (ml)	-0.015837	1.000000	0.111480
Consumed (ml)	0.258866	0.111480	1.000000
Closing Balance (ml)	0.966686	0.227826	0.201699
Purchase Quantity (ml)	-0.015837	1.000000	0.111480
Sale Quantity (ml)	0.258866	0.111480	1.000000

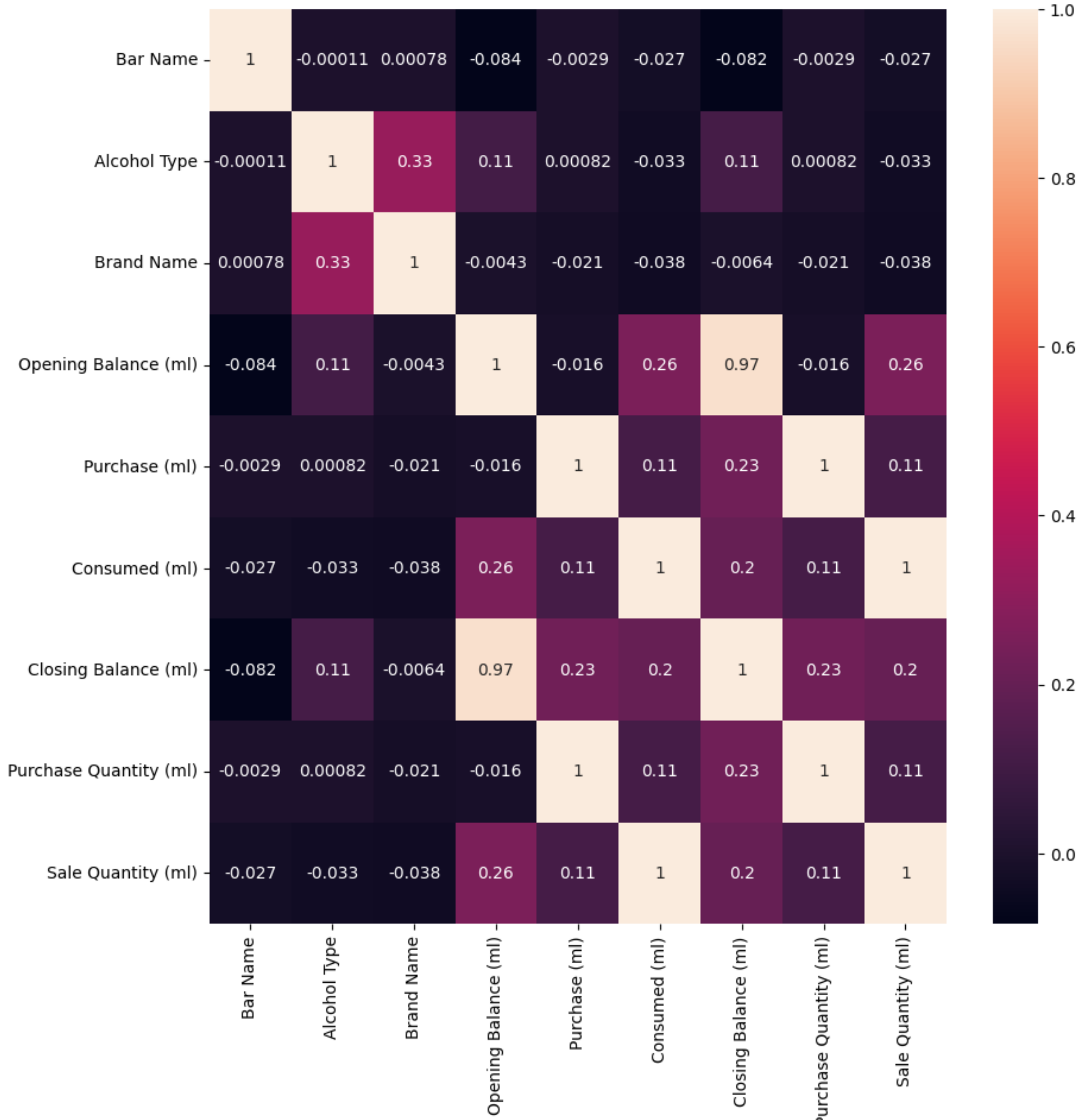
	Closing Balance (ml)	Purchase Quantity (ml)
Bar Name	-0.081517	-0.002936
Alcohol Type	0.112988	0.000823
Brand Name	-0.006405	-0.020959
Opening Balance (ml)	0.966686	-0.015837
Purchase (ml)	0.227826	1.000000
Consumed (ml)	0.201699	0.111480
Closing Balance (ml)	1.000000	0.227826
Purchase Quantity (ml)	0.227826	1.000000
Sale Quantity (ml)	0.201699	0.111480

	Sale Quantity (ml)
Bar Name	-0.026739
Alcohol Type	-0.033068
Brand Name	-0.038064
Opening Balance (ml)	0.258866
Purchase (ml)	0.111480
Consumed (ml)	1.000000

```
Closing Balance (ml)              0.201699
Purchase Quantity (ml)            0.111480
Sale Quantity (ml)                1.000000
```

11 5.4 Heatmap of correlation matrix

[24]:
```python
plt.figure(figsize=(10,10))
sns.heatmap(corr,annot=True)
plt.show()
```

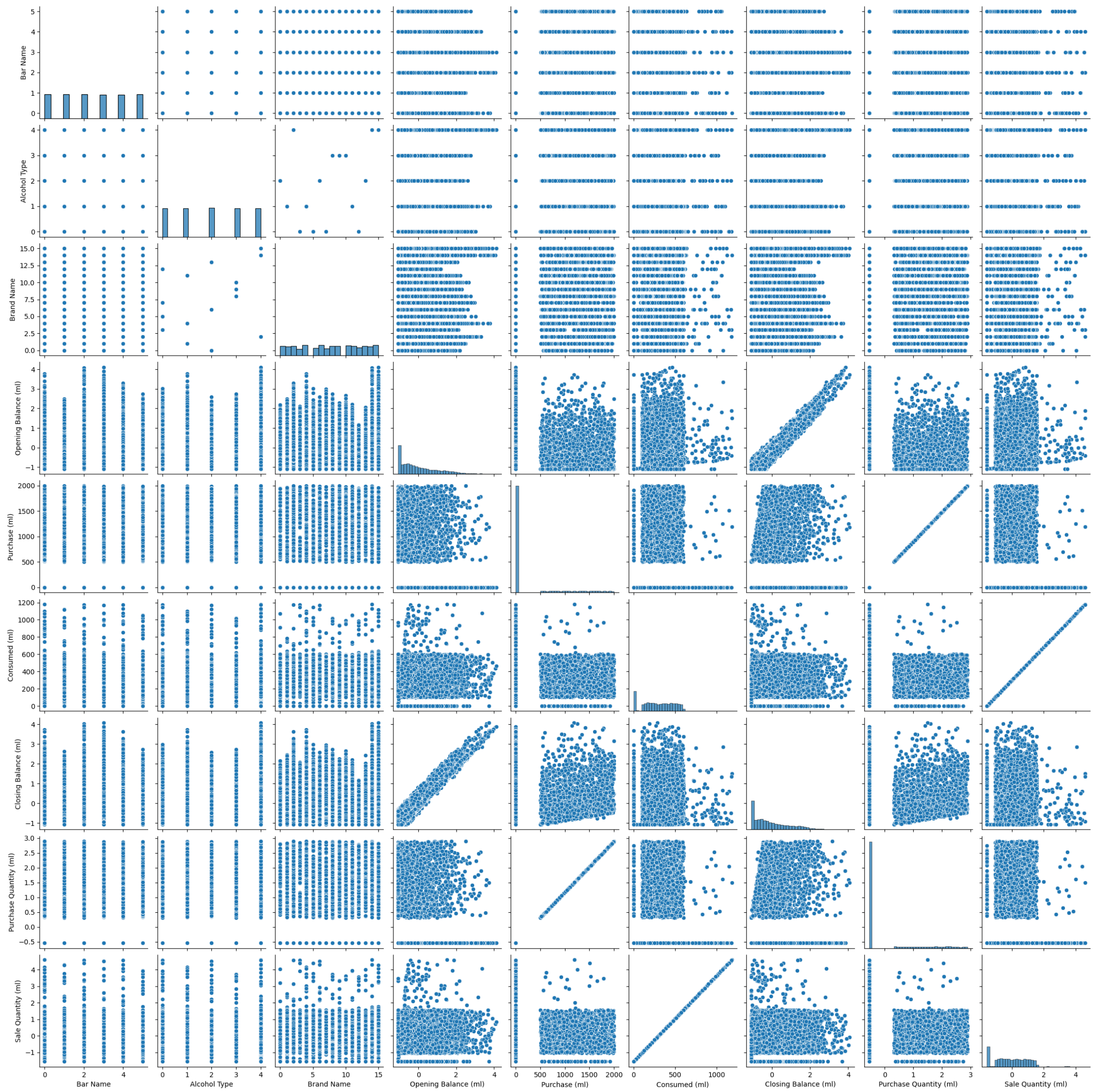


	Bar Name	Alcohol Type	Brand Name	Opening Balance (ml)	Purchase (ml)	Consumed (ml)	Closing Balance (ml)	Purchase Quantity (ml)	Sale Quantity (ml)
Bar Name	1	-0.00011	0.00078	-0.084	-0.0029	-0.027	-0.082	-0.0029	-0.027
Alcohol Type	-0.00011	1	0.33	0.11	0.00082	-0.033	0.11	0.00082	-0.033
Brand Name	0.00078	0.33	1	-0.0043	-0.021	-0.038	-0.0064	-0.021	-0.038
Opening Balance (ml)	-0.084	0.11	-0.0043	1	-0.016	0.26	0.97	-0.016	0.26
Purchase (ml)	-0.0029	0.00082	-0.021	-0.016	1	0.11	0.23	1	0.11
Consumed (ml)	-0.027	-0.033	-0.038	0.26	0.11	1	0.2	0.11	1
Closing Balance (ml)	-0.082	0.11	-0.0064	0.97	0.23	0.2	1	0.23	0.2
Purchase Quantity (ml)	-0.0029	0.00082	-0.021	-0.016	1	0.11	0.23	1	0.11
Sale Quantity (ml)	-0.027	-0.033	-0.038	0.26	0.11	1	0.2	0.11	1

#5.5 Pair plot of each column to column

[25]:
```
plt.figure(figsize=(10,10))
sns.pairplot(df_name)
plt.show()
```

```
<Figure size 1000x1000 with 0 Axes>
```



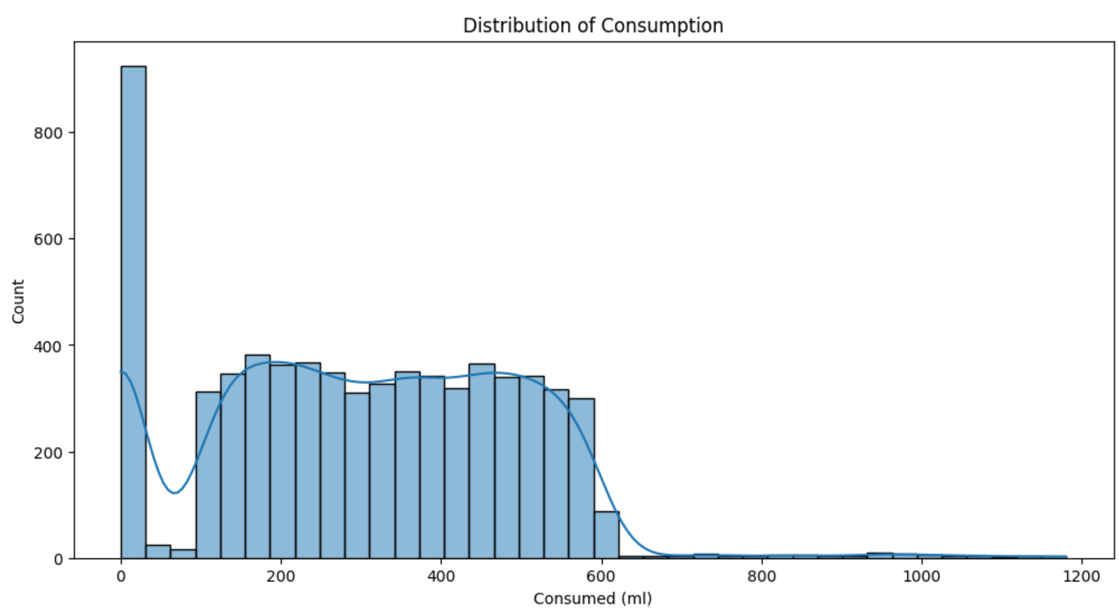
#5.6 graph between opening balance vs closing balance

[26]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,5))
plt.plot(df_name['Closing Balance (ml)'], df_name['Opening Balance (ml)'])
plt.xlabel('Closing Balance')
plt.ylabel('Opening Balance')
```

```
plt.title('Opening Balance vs. Closing Balance ')
plt.show()
```

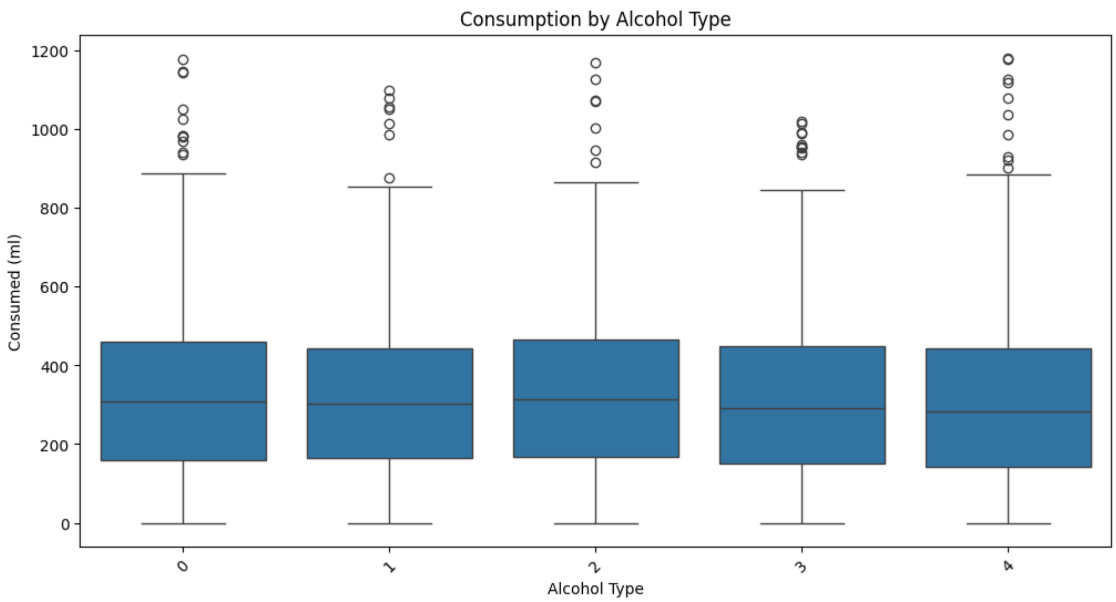
#5.7 Distribution of Consumption

[27]:
```
plt.figure(figsize=(12, 6))
sns.histplot(df['Consumed (ml)'], kde=True)
plt.title('Distribution of Consumption')
plt.show()
```



#5.8 Consumption by Alcohol Type

[28]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(x='Alcohol Type', y='Consumed (ml)', data=df)
plt.title('Consumption by Alcohol Type')
plt.xticks(rotation=45)
plt.show()
```



12 6. Aggregate Data Daily per Brand

13 6.1 Features and target

14 Feature Engineering

[29]:
```
# Convert 'Date Time Served' to datetime if not already
df['Date Time Served'] = pd.to_datetime(df['Date Time Served'])

# Extract the date part into a new column, keeping it as a datetime-like object
# Using .dt.floor('D') effectively removes the time component while keeping it␣
↪a datetime object
df['Date'] = df['Date Time Served'].dt.floor('D')

# Feature Engineering
df['Brand'] = df['Brand Name'].astype('category').cat.codes
```

```
df['Bar'] = df['Bar Name'].astype('category').cat.codes
df['DayOfYear'] = df['Date'].dt.dayofyear
```

[30]:
```
daily_df = df.groupby(['Date', 'Bar Name', 'Brand Name'])['Consumed (ml)'].
↪sum().reset_index()
daily_df['Date'] = pd.to_datetime(daily_df['Date'])
daily_df
```

[30]:
```
           Date  Bar Name  Brand Name  Consumed (ml)
0     2023-01-01         0           1           0.00
1     2023-01-01         0          10           0.00
2     2023-01-01         0          12           0.00
3     2023-01-01         0          14           0.00
4     2023-01-01         1           4           0.00
…            …        …           …              …
6570 2024-01-01         4           3         558.72
6571 2024-01-01         4           4         440.76
6572 2024-01-01         4          10         253.33
6573 2024-01-01         5           0         128.52
6574 2024-01-01         5           1         572.60

[6575 rows x 4 columns]
```

[31]:
```
daily_df['Brand_Code'] = daily_df['Brand Name'].astype('category').cat.codes
daily_df['Bar_Code'] = daily_df['Bar Name'].astype('category').cat.codes

X = daily_df[['Brand_Code', 'Bar_Code']]
X['DayOfYear'] = daily_df['Date'].dt.dayofyear
y = daily_df['Consumed (ml)']
```

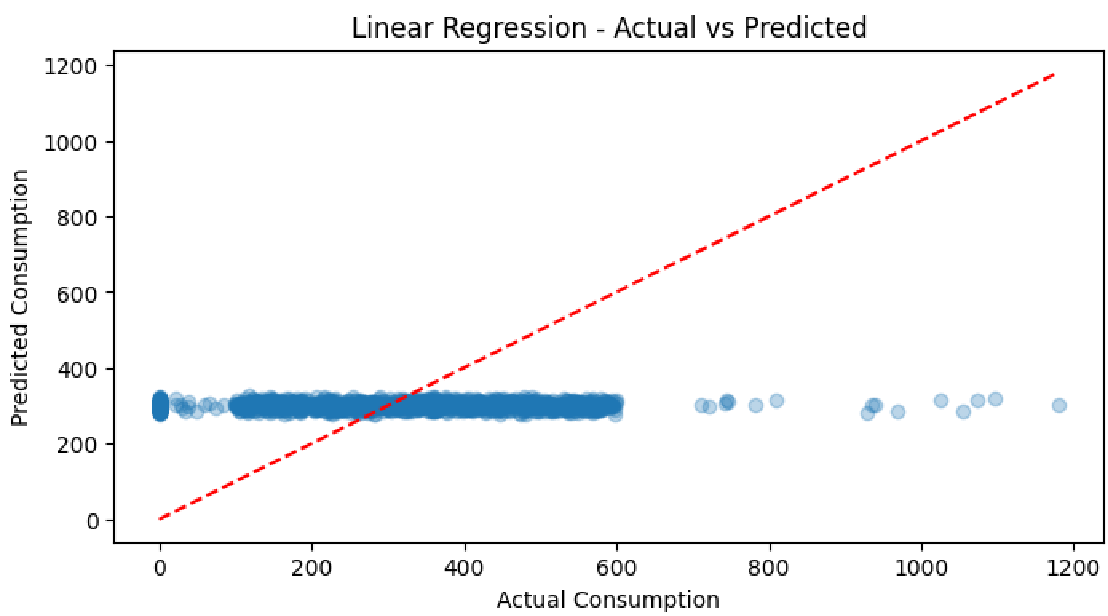
15 7. Train/Test Split

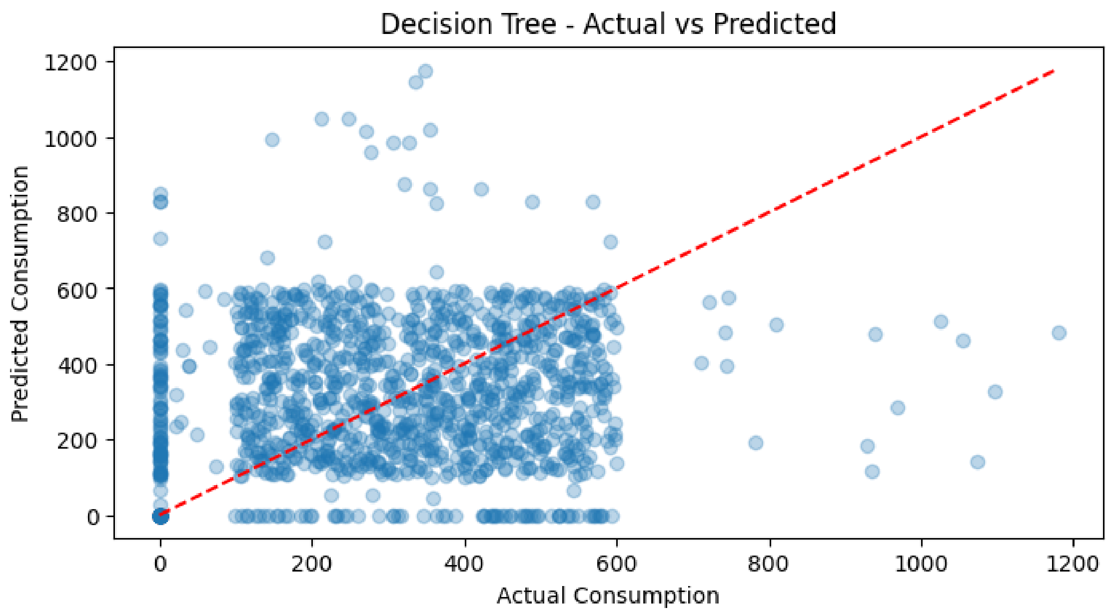
[32]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

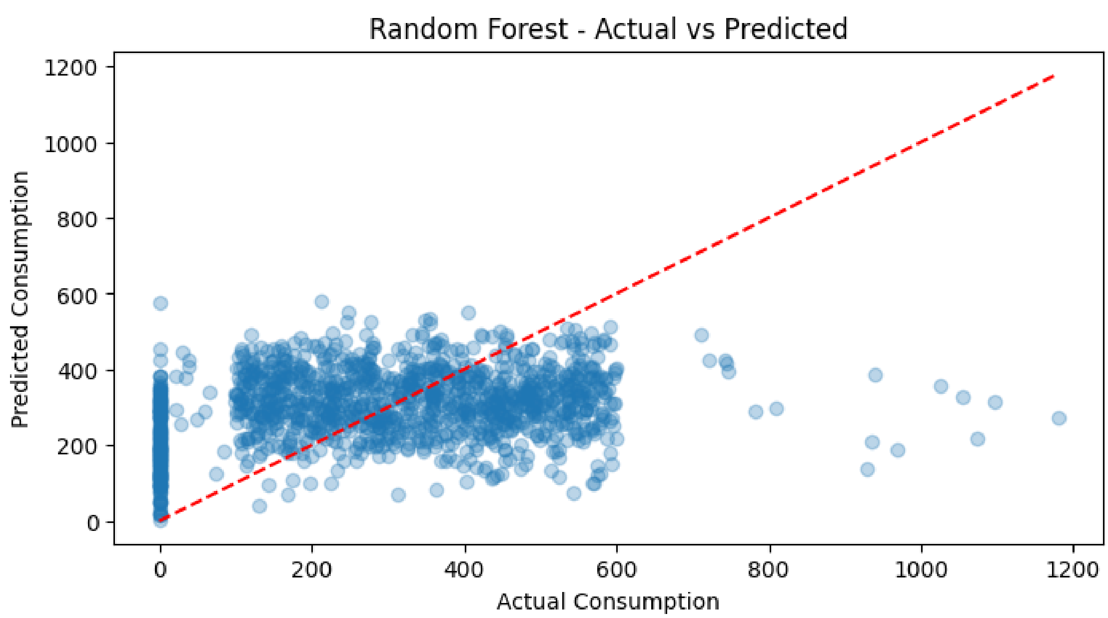
16 8. Model Building

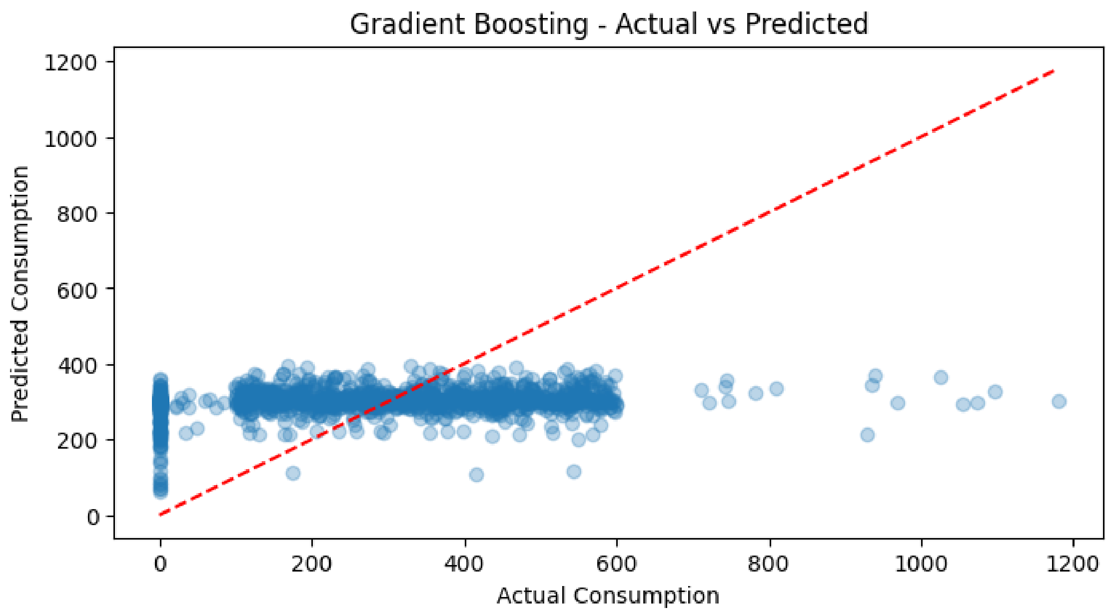
[33]:
```
models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest": RandomForestRegressor(n_estimators=100),
    "Gradient Boosting": GradientBoostingRegressor(n_estimators=100),
    "XGBoost": XGBRegressor(n_estimators=100)
}
```

```python
results = []

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)
    results.append({"Model": name, "MAE": mae, "RMSE": rmse, "R2 Score": r2})

    # Plot actual vs predicted
    plt.figure(figsize=(8, 4))
    plt.scatter(y_test, y_pred, alpha=0.3)
    plt.xlabel('Actual Consumption')
    plt.ylabel('Predicted Consumption')
    plt.title(f'{name} - Actual vs Predicted')
    plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
    plt.show()
```

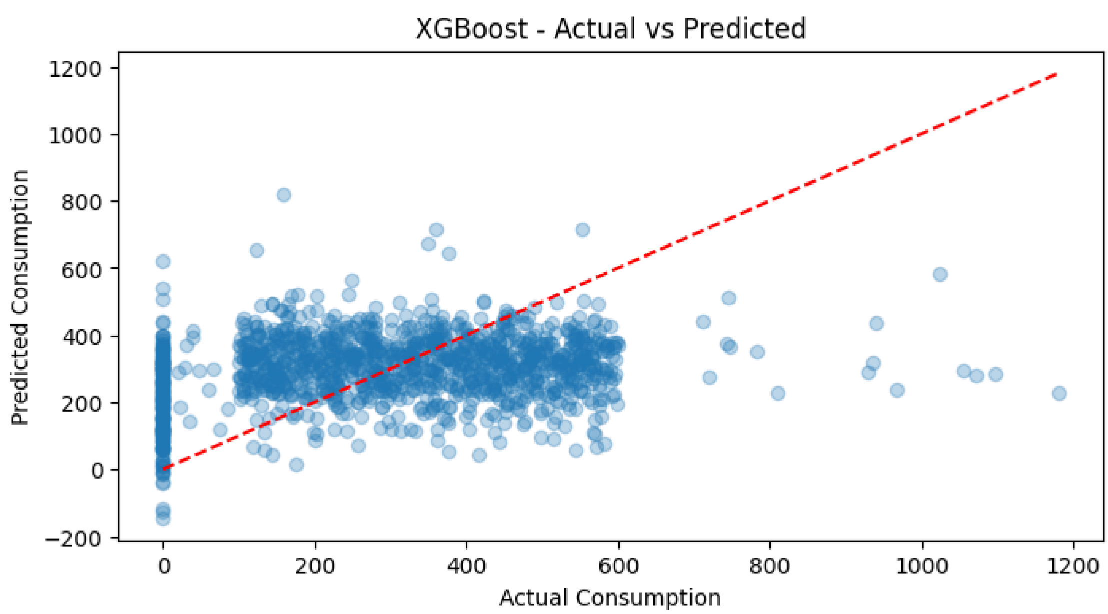



Decision Tree - Actual vs Predicted
Predicted Consumption
Actual Consumption
0
200
400
600
800
1000
1200


Random Forest - Actual vs Predicted
Predicted Consumption
Actual Consumption
0
200
400
600
800
1000
1200



Gradient Boosting - Actual vs Predicted
Predicted Consumption
Actual Consumption
0
200
400
600
800
1000
1200



XGBoost - Actual vs Predicted
Predicted Consumption
Actual Consumption
−200
0
200
400
600
800
1000
1200

17 9. Model Evaluation Summary Table

[34]:
```python
results_df = pd.DataFrame(results)
print("\nModel Evaluation Results:")
print(results_df)

# Save results to CSV (optional)
results_df.to_csv("model_evaluation_results.csv", index=False) # Optional
```

```
Model Evaluation Results:
               Model         MAE        RMSE  R2 Score
0  Linear Regression  158.742100  191.348988 -0.000652
1      Decision Tree  187.024437  247.305217 -0.671465
2      Random Forest  156.007551  190.446527  0.008765
3  Gradient Boosting  154.187473  185.943524  0.055085
4            XGBoost  158.768805  195.240908 -0.041771
```

18 10. Few more models

[35]:
```python
!pip install catboost
from sklearn.ensemble import GradientBoostingRegressor
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor
```

```
Collecting catboost
  Downloading catboost-1.2.8-cp311-cp311-manylinux2014_x86_64.whl.metadata (1.2
kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-
packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-
packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in
/usr/local/lib/python3.11/dist-packages (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.11/dist-
packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
(from catboost) (1.15.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages
(from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages
(from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas>=0.24->catboost) (2025.2)
```

```
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-
packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (4.58.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.4.8)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-
packages (from matplotlib->catboost) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (3.2.3)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.11/dist-packages (from plotly->catboost) (9.1.2)
Downloading catboost-1.2.8-cp311-cp311-manylinux2014_x86_64.whl (99.2 MB)
                          99.2/99.2 MB
8.5 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.8
```

[36]:
```
gb_model = GradientBoostingRegressor(random_state=42)
gb_model.fit(X_train, y_train)
gb_preds = gb_model.predict(X_test)

catboost_model = CatBoostRegressor(verbose=0, iterations=100, random_state=42)
catboost_model.fit(X_train, y_train)
catboost_preds = catboost_model.predict(X_test)

lgbm_model = LGBMRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
lgbm_model.fit(X_train, y_train)
lgbm_preds = lgbm_model.predict(X_test)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000501 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 262
[LightGBM] [Info] Number of data points in the train set: 5260, number of used
features: 3
[LightGBM] [Info] Start training from score 300.552523
```

19 Ensemble

[37]:
```
ensemble_preds = (gb_preds + catboost_preds + lgbm_preds) / 3
```

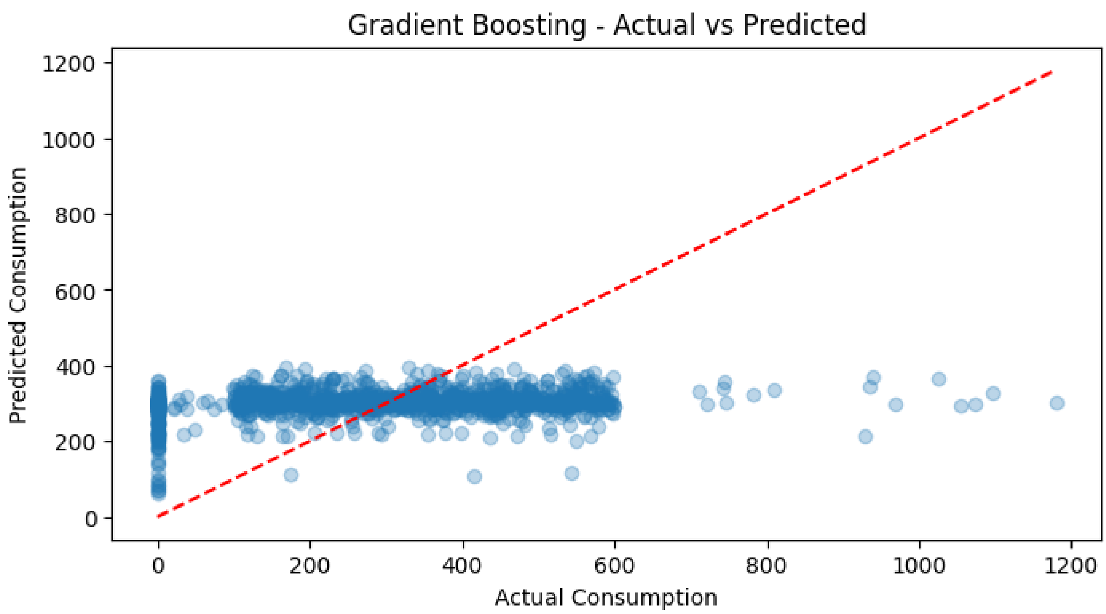
#11. Evaluation of few model

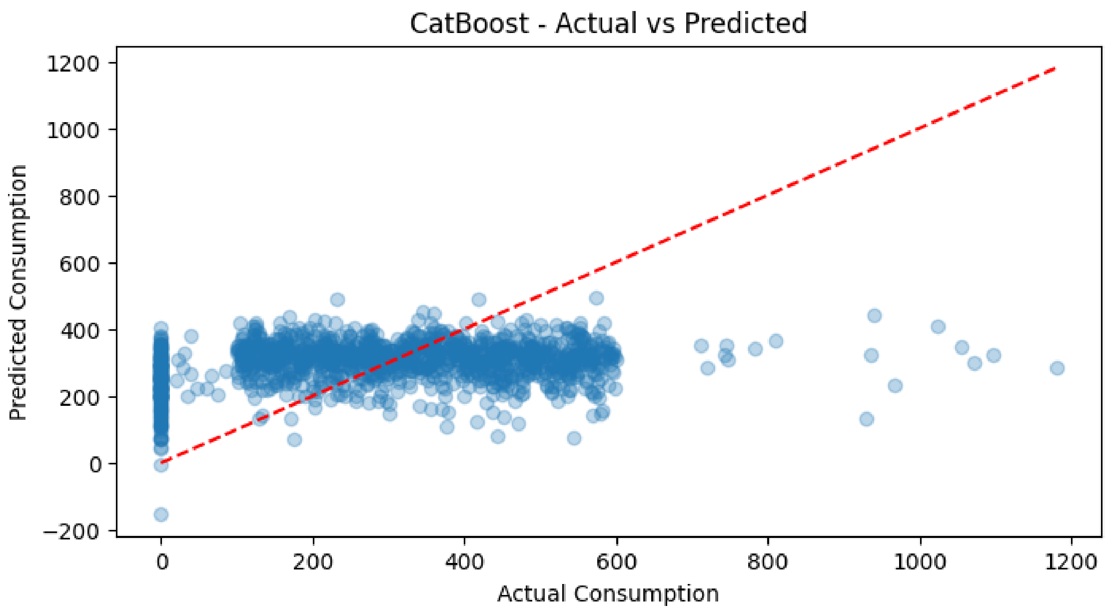
[38]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

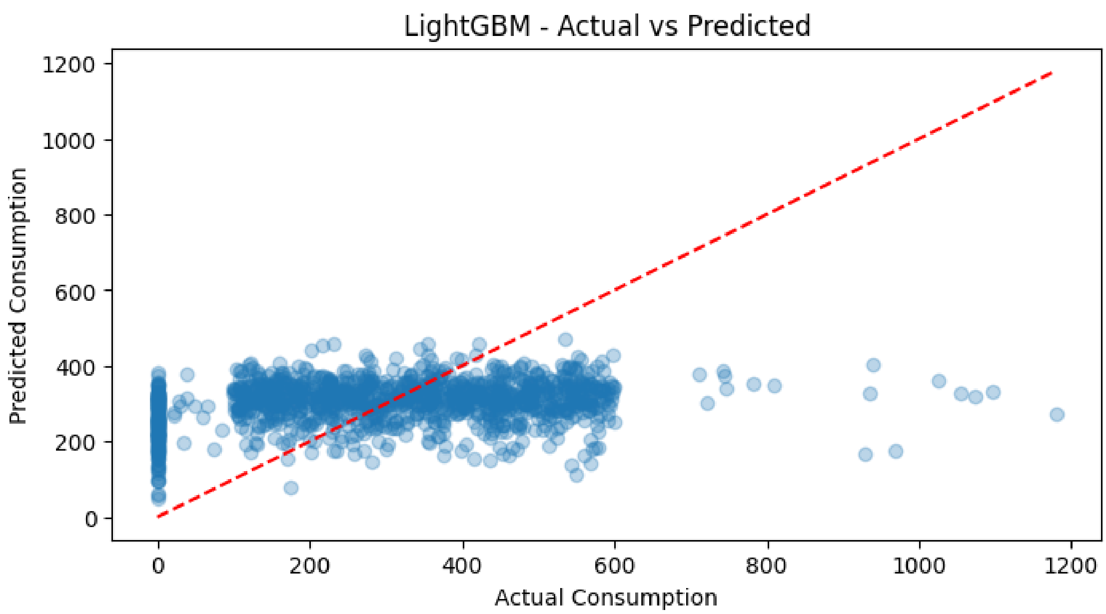
[39]:
```
model_names = ['Gradient Boosting', 'CatBoost', 'LightGBM', 'Ensemble (GB +␣
↪CatBoost + LGBM)']
mae_scores = [
    mean_absolute_error(y_test, gb_preds),
    mean_absolute_error(y_test, catboost_preds),
    mean_absolute_error(y_test, lgbm_preds),
    mean_absolute_error(y_test, ensemble_preds)
]
rmse_scores = [
    mean_squared_error(y_test, gb_preds)**0.5,
    mean_squared_error(y_test, catboost_preds)**0.5,
    mean_squared_error(y_test, lgbm_preds)**0.5,
    mean_squared_error(y_test, ensemble_preds)**0.5
]

r2_scores = [
    r2_score(y_test, gb_preds),
    r2_score(y_test, catboost_preds),
    r2_score(y_test, lgbm_preds),
    r2_score(y_test, ensemble_preds)
]
```

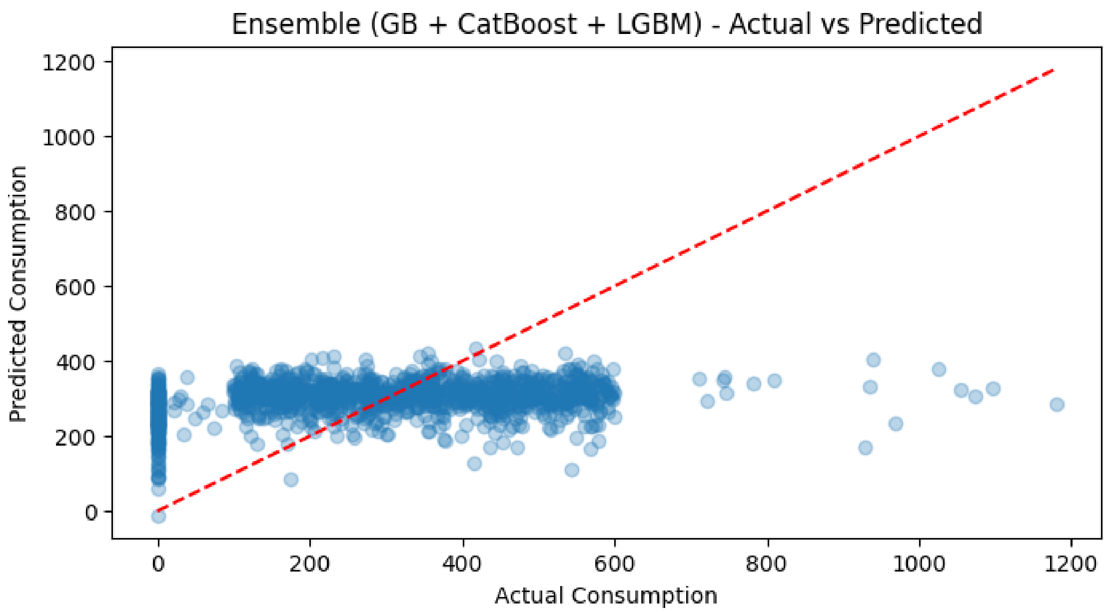
[40]:
```
plt.figure(figsize=(8, 4))
plt.scatter(y_test, gb_preds, alpha=0.3)
plt.xlabel('Actual Consumption')
plt.ylabel('Predicted Consumption')
plt.title('Gradient Boosting - Actual vs Predicted')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.show()

plt.figure(figsize=(8, 4))
plt.scatter(y_test, catboost_preds, alpha=0.3)
plt.xlabel('Actual Consumption')
plt.ylabel('Predicted Consumption')
plt.title('CatBoost - Actual vs Predicted')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.show()

plt.figure(figsize=(8, 4))
```

```
plt.scatter(y_test, lgbm_preds, alpha=0.3)
plt.xlabel('Actual Consumption')
plt.ylabel('Predicted Consumption')
plt.title('LightGBM - Actual vs Predicted')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.show()

plt.figure(figsize=(8, 4))
plt.scatter(y_test, ensemble_preds, alpha=0.3)
plt.xlabel('Actual Consumption')
plt.ylabel('Predicted Consumption')
plt.title('Ensemble (GB + CatBoost + LGBM) - Actual vs Predicted')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.show()
```




CatBoost - Actual vs Predicted
Predicted Consumption
Actual Consumption
1200
1000
800
600
400
200
0
−200
0
200
400
600
800
1000
1200


LightGBM - Actual vs Predicted
Predicted Consumption
Actual Consumption
1200
1000
800
600
400
200
0
0
200
400
600
800
1000
1200



[41]:
```
# Final results table
evaluation_df = pd.DataFrame({
    'Model': model_names,
    'MAE': mae_scores,
    'RMSE': rmse_scores,
    'R2 Score': r2_scores
})

print("\nModel Evaluation Results:")
print(evaluation_df.sort_values(by='RMSE'))
```

```
Model Evaluation Results:
                              Model         MAE        RMSE  R2 Score
3  Ensemble (GB + CatBoost + LGBM)  152.990501  184.265086  0.072067
2                          LightGBM  152.947842  184.576734  0.068925
0                 Gradient Boosting  154.187473  185.943524  0.055085
1                          CatBoost  154.855334  187.246744  0.041793
```

[42]:
```
import pandas as pd
final_evaluation_df = pd.concat([results_df, evaluation_df], ignore_index=True)
print("\nCombined Model Evaluation Results:")
print(final_evaluation_df.sort_values(by='RMSE'))
```

```
Combined Model Evaluation Results:
                              Model         MAE        RMSE  R2 Score
```

```
8  Ensemble (GB + CatBoost + LGBM)  152.990501  184.265086  0.072067
7                         LightGBM  152.947842  184.576734  0.068925
5                Gradient Boosting  154.187473  185.943524  0.055085
3                Gradient Boosting  154.187473  185.943524  0.055085
6                         CatBoost  154.855334  187.246744  0.041793
2                    Random Forest  156.007551  190.446527  0.008765
0                Linear Regression  158.742100  191.348988 -0.000652
4                          XGBoost  158.768805  195.240908 -0.041771
1                    Decision Tree  187.024437  247.305217 -0.671465
```

Observations:

- Best performing model by all metrics is the ensemble of Gradient Boosting + CatBoost + LightGBM.
- LightGBM alone is a close second, with nearly identical MAE and RMSE but slightly lower R^2.
- The ensemble improved performance marginally, showing that combining these models helped, but the improvement is slight.
- Traditional Linear Regression and Decision Tree models perform the worst, with negative R^2 scores indicating poor fits.
- XGBoost, surprisingly, performed worse than some others here.

---

20 Forecasting on dataset

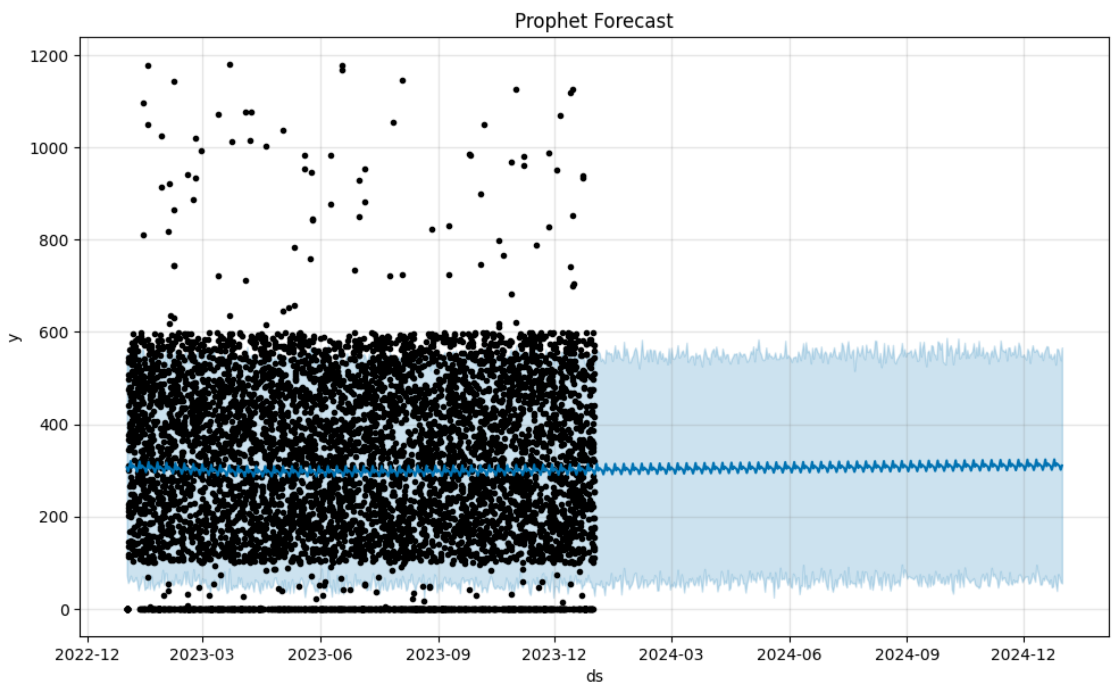
21 Prophet Forecasting

[43]:
```python
import matplotlib.pyplot as plt
from prophet import Prophet
import warnings
warnings.filterwarnings("ignore")

# Prepare data for Prophet (requires 'ds' for datetime and 'y' for the value)
prophet_df = daily_df[['Date', 'Consumed (ml)']].rename(columns={'Date': 'ds',␣
↪'Consumed (ml)': 'y'})

# Initialize and fit Prophet model
prophet_model = Prophet()
prophet_model.fit(prophet_df)

# Create a future dataframe for forecasting
future = prophet_model.make_future_dataframe(periods=365) # Forecast for the␣
↪next 365 days

# Make predictions
```

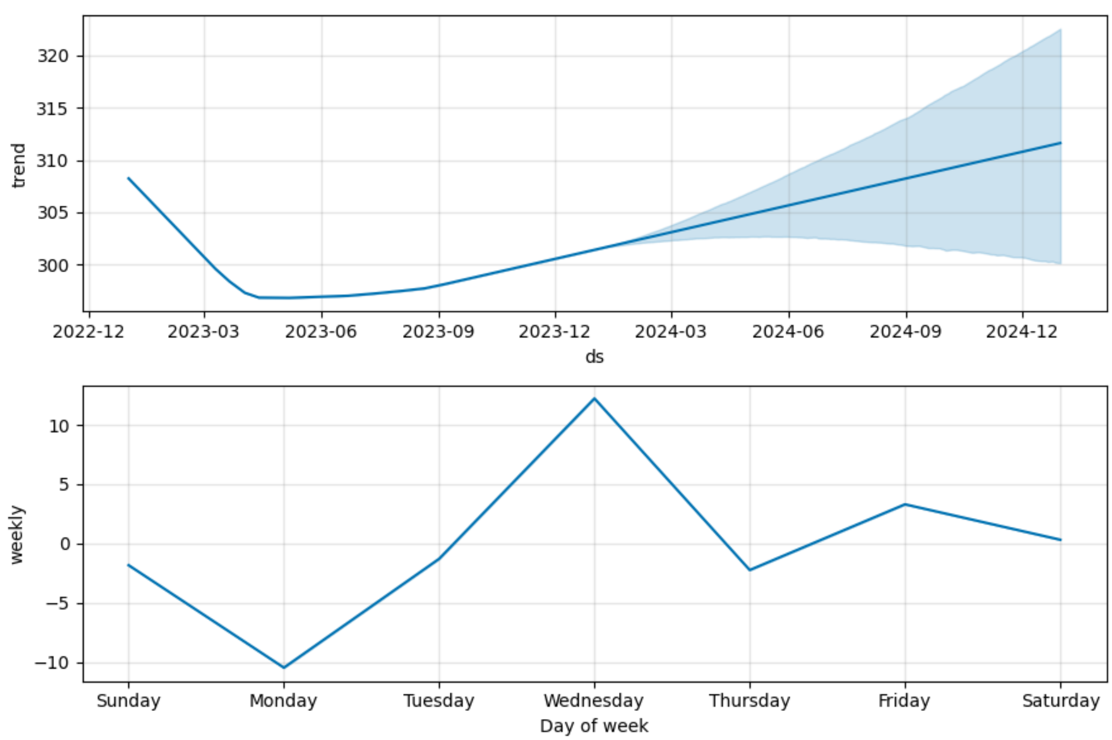
```python
forecast = prophet_model.predict(future)

# Display the forecast (contains columns like 'ds', 'yhat', 'yhat_lower',␣
↪'yhat_upper')
print("\nProphet Forecast:")
print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail())
```

```
INFO:prophet:Disabling yearly seasonality. Run prophet with
yearly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp2js9m6_v/tuzkexc0.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp2js9m6_v/xcnnelpc.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=67517', 'data',
'file=/tmp/tmp2js9m6_v/tuzkexc0.json', 'init=/tmp/tmp2js9m6_v/xcnnelpc.json',
'output',
'file=/tmp/tmp2js9m6_v/prophet_modelh3rm_y1t/prophet_model-20250523121910.csv',
'method=optimize', 'algorithm=lbfgs', 'iter=10000']
12:19:10 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
12:19:11 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing


Prophet Forecast:
            ds        yhat  yhat_lower  yhat_upper
726 2024-12-27  314.799830   38.727789  563.629178
727 2024-12-28  311.843940   75.114068  561.752552
728 2024-12-29  309.724985   68.024845  550.413998
729 2024-12-30  301.115703   57.366955  538.673779
730 2024-12-31  310.322624   55.027775  566.064085
```

[44]:
```python
# Plot the forecast
fig1 = prophet_model.plot(forecast)
plt.title("Prophet Forecast")
plt.show()
```



[45]:
```python
# Plot the components of the forecast (trend, weekly, yearly seasonality)
fig2 = prophet_model.plot_components(forecast)
plt.show()
```



#Recommend Inventory (Par Level)

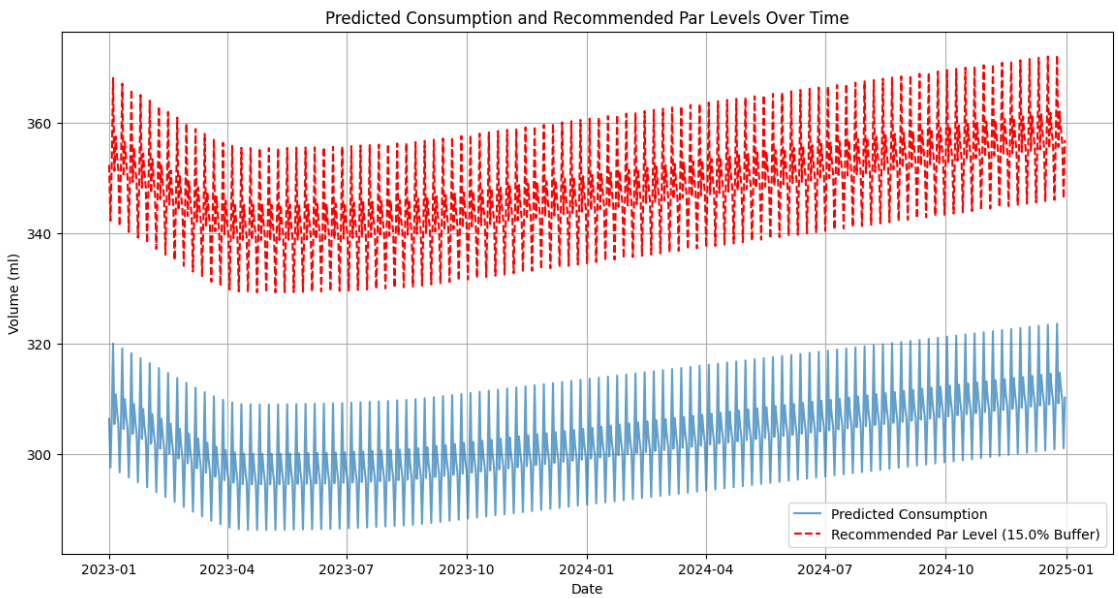
[46]:
```python
import pandas as pd
import matplotlib.pyplot as plt

# Define a buffer or safety stock percentage
safety_stock_buffer = 0.15  # 15% of predicted consumption

def recommend_par_level(forecast_df, buffer_percentage=0.15):

  recommendation_df = forecast_df[['ds', 'yhat']].copy()
  recommendation_df.rename(columns={'ds': 'Date', 'yhat': 'Predicted␣
↪Consumption'}, inplace=True)

  # Calculate recommended par level: Predicted Consumption + Safety Stock
  # Ensure par level is not negative
  recommendation_df['Recommended Par Level'] = (recommendation_df['Predicted␣
↪Consumption'] * (1 + buffer_percentage)).apply(lambda x: max(0, x))

  return recommendation_df

# Generate inventory recommendations using the Prophet forecast
inventory_recommendations = recommend_par_level(forecast, safety_stock_buffer)

print("\nInventory Recommendations (Par Level):")
print(inventory_recommendations.head())
```

```
Inventory Recommendations (Par Level):
        Date  Predicted Consumption  Recommended Par Level
0 2023-01-01             306.407061             352.368120
1 2023-01-02             297.642749             342.289161
2 2023-01-03             306.694641             352.698838
3 2023-01-04             320.081329             368.093529
4 2023-01-05             305.491577             351.315314
```

22 Predicted Consumption and Recommended Par Levels Over Time

[47]:
```python
plt.figure(figsize=(14, 7))
plt.plot(inventory_recommendations['Date'],␣
↪inventory_recommendations['Predicted Consumption'], label='Predicted␣
↪Consumption', alpha=0.7)
```

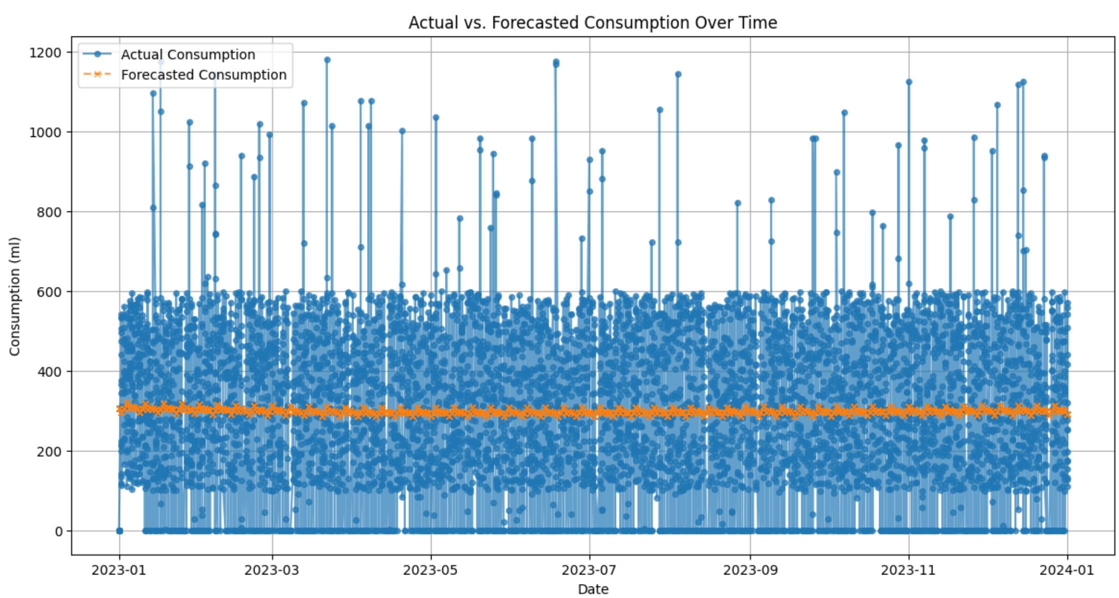
```
plt.plot(inventory_recommendations['Date'],␣
 ↪inventory_recommendations['Recommended Par Level'], label=f'Recommended Par␣
 ↪Level ({safety_stock_buffer*100}% Buffer)', linestyle='--', color='red')
plt.title('Predicted Consumption and Recommended Par Levels Over Time')
plt.xlabel('Date')
plt.ylabel('Volume (ml)')
plt.legend()
plt.grid(True)
plt.show()
```



22.1 Actual vs. Forecasted Consumption Over Time

[48]:
```
import pandas as pd
import matplotlib.pyplot as plt
# Combine actuals and forecast for plotting
# We need to merge the original data with the forecast for overlapping dates

# Rename 'ds' in forecast to 'Date' for merging
forecast_merged = forecast[['ds', 'yhat']].rename(columns={'ds': 'Date', 'yhat':
 ↪ 'Forecasted Consumption'})

# Ensure 'Date' in daily_df is datetime for merging
daily_df['Date'] = pd.to_datetime(daily_df['Date'])

# Merge actuals and forecast on Date
comparison_df = pd.merge(daily_df[['Date', 'Consumed (ml)']], forecast_merged,␣
 ↪on='Date', how='left')
```

```
# Rename 'Consumed (ml)' to 'Actual Consumption' for clarity
comparison_df.rename(columns={'Consumed (ml)': 'Actual Consumption'},␣
↪inplace=True)

# Sort by date
comparison_df = comparison_df.sort_values(by='Date')

# Plotting
plt.figure(figsize=(14, 7))
plt.plot(comparison_df['Date'], comparison_df['Actual Consumption'],␣
↪label='Actual Consumption', marker='o', linestyle='-', markersize=4, alpha=0.
↪7)
plt.plot(comparison_df['Date'], comparison_df['Forecasted Consumption'],␣
↪label='Forecasted Consumption', marker='x', linestyle='--', markersize=4,␣
↪alpha=0.7)

plt.title('Actual vs. Forecasted Consumption Over Time')
plt.xlabel('Date')
plt.ylabel('Consumption (ml)')
plt.legend()
plt.grid(True)
plt.show()
```



23 Actual vs. Forecasted Consumption (Overlap Period)

[49]:
```
comparison_df_valid = comparison_df.dropna(subset=['Forecasted Consumption'])

plt.figure(figsize=(14, 7))
plt.plot(comparison_df_valid['Date'], comparison_df_valid['Actual␣
↪Consumption'], label='Actual Consumption', marker='o', linestyle='-',␣
↪markersize=4, alpha=0.7)
plt.plot(comparison_df_valid['Date'], comparison_df_valid['Forecasted␣
↪Consumption'], label='Forecasted Consumption', marker='x', linestyle='--',␣
↪markersize=4, alpha=0.7)

plt.title('Actual vs. Forecasted Consumption (Overlap Period)')
plt.xlabel('Date')
plt.ylabel('Consumption (ml)')
plt.legend()
plt.grid(True)
plt.show()
```