In [2]:
```
#import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the datasets
train_df = pd.read_csv('/content/train.csv')
test_df = pd.read_csv('/content/test.csv')

# Convert 'date' columns to datetime
train_df['date'] = pd.to_datetime(train_df['date'])
test_df['date'] = pd.to_datetime(test_df['date'])

#basic information about train and test
print(train_df.info())
print(test_df.info())

# Checking missing values
print(train_df.isnull().sum())
print(test_df.isnull().sum())
```

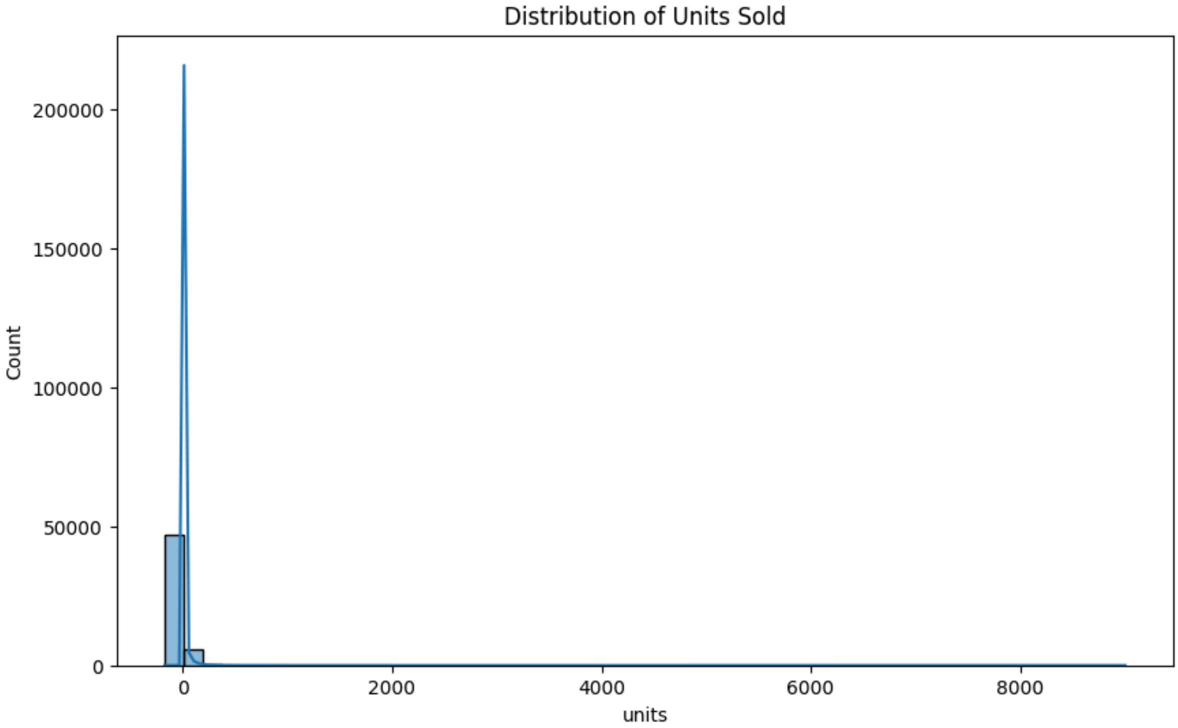
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56982 entries, 0 to 56981
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   ID          56982 non-null  object
 1   date        56982 non-null  datetime64[ns]
 2   Item Id     56982 non-null  object
 3   Item Name   56981 non-null  object
 4   ad_spend    37525 non-null  float64
 5   anarix_id   56981 non-null  object
 6   units       53068 non-null  float64
 7   unit_price  56981 non-null  float64
dtypes: datetime64[ns](1), float64(3), object(4)
memory usage: 3.5+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2833 entries, 0 to 2832
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   ID          2833 non-null   object
 1   date        2833 non-null   datetime64[ns]
 2   Item Id     2833 non-null   object
 3   Item Name   2489 non-null   object
 4   ad_spend    1382 non-null   float64
 5   anarix_id   2833 non-null   object
 6   unit_price  2833 non-null   float64
dtypes: datetime64[ns](1), float64(2), object(4)
memory usage: 155.1+ KB
None
ID                  0
date                0
Item Id             0
Item Name           1
ad_spend        19457
anarix_id           1
units            3914
unit_price          1
dtype: int64
ID                0
date              0
Item Id           0
Item Name       344
ad_spend       1451
anarix_id         0
unit_price        0
dtype: int64
```

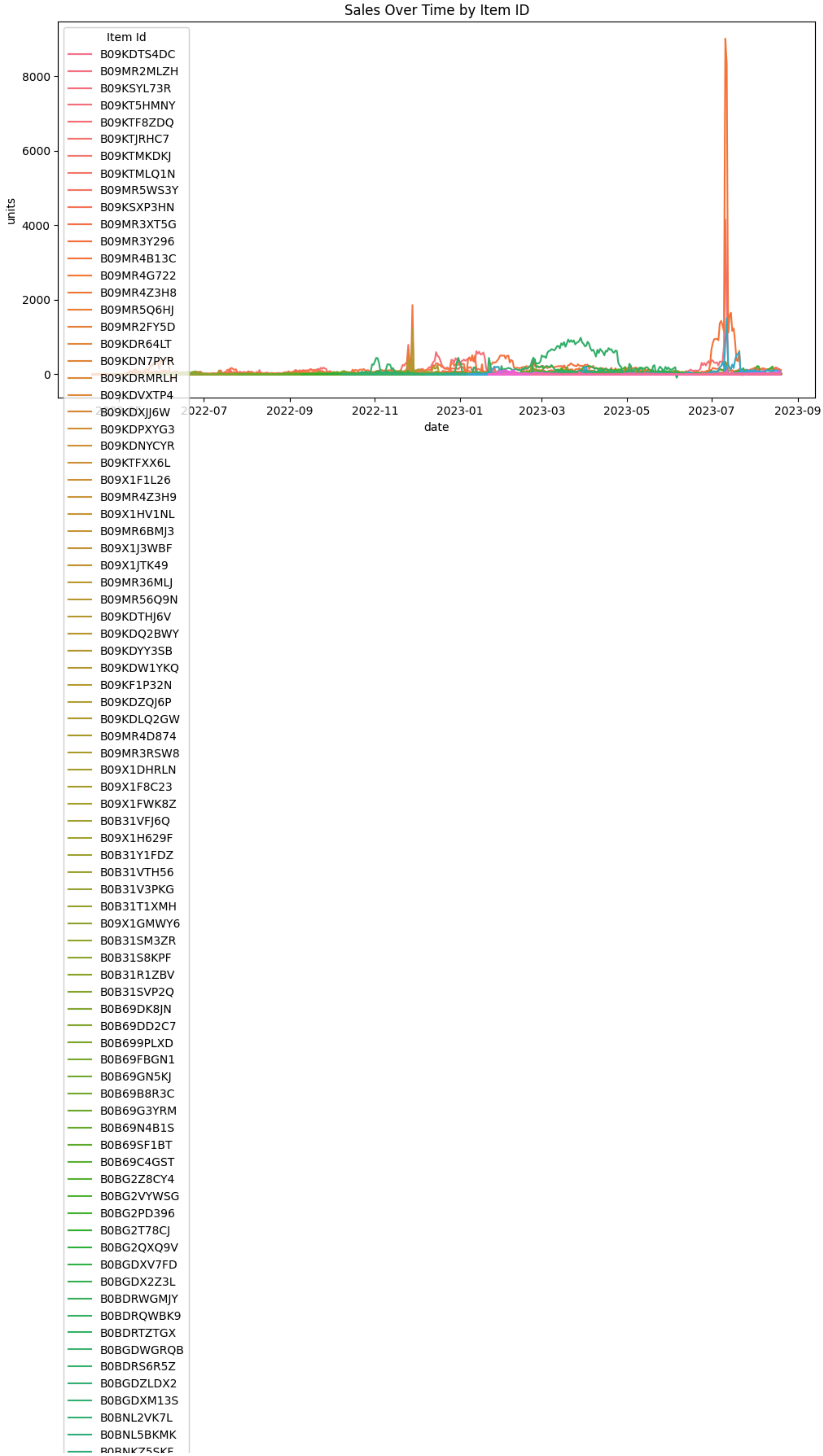
In [ ]:

In [4]:

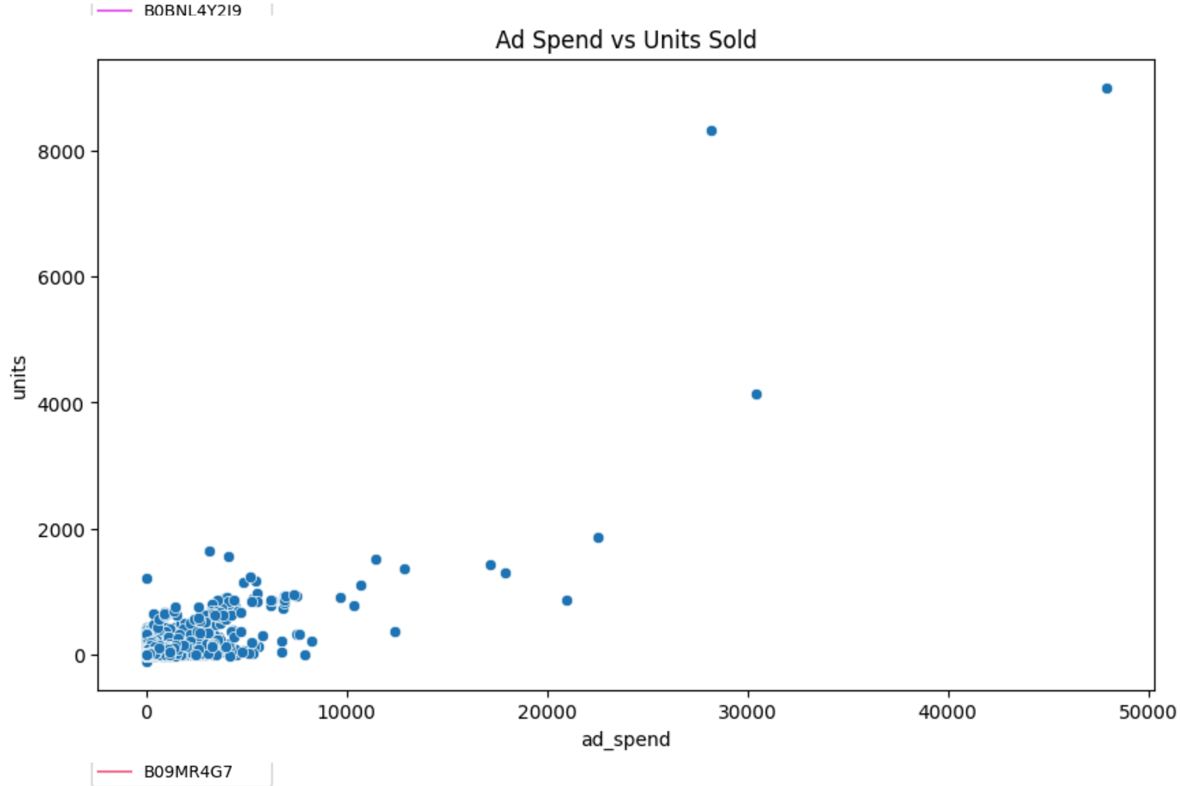
```
# EXPLORATORY DATA ANALYSIS
# Sales distribution
plt.figure(figsize=(10, 6))
sns.histplot(train_df['units'], bins=50, kde=True)
plt.title('Distribution of Units Sold')
plt.show()

# Sales over time
plt.figure(figsize=(12, 6))
sns.lineplot(data=train_df, x='date', y='units', hue='Item Id')
plt.title('Sales Over Time by Item ID')
plt.show()
```

```
# Ad Spend vs Units Sold
plt.figure(figsize=(10, 6))
sns.scatterplot(data=train_df, x='ad_spend', y='units')
plt.title('Ad Spend vs Units Sold')
plt.show()
```




Sales Over Time by Item ID
Item Id
B09KDTS4DC
B09MR2MLZH
B09KSYL73R
B09KT5HMNY
B09KTF8ZDQ
B09KTJRHC7
B09KTMKDKJ
B09KTMLQ1N
B09MR5WS3Y
B09KSXP3HN
B09MR3XT5G
B09MR3Y296
B09MR4B13C
B09MR4G722
B09MR4Z3H8
B09MR5Q6HJ
B09MR2FY5D
B09KDR64LT
B09KDN7PYR
B09KDRMRLH
B09KDVXTP4
B09KDXJJ6W
B09KDPXYG3
B09KDNYCYR
B09KTFXX6L
B09X1F1L26
B09MR4Z3H9
B09X1HV1NL
B09MR6BMJ3
B09X1J3WBF
B09X1JTK49
B09MR36MLJ
B09MR56Q9N
B09KDTHJ6V
B09KDQ2BWY
B09KDYY3SB
B09KDW1YKQ
B09KF1P32N
B09KDZQJ6P
B09KDLQ2GW
B09MR4D874
B09MR3RSW8
B09X1DHRLN
B09X1F8C23
B09X1FWK8Z
B0B31VFJ6Q
B09X1H629F
B0B31Y1FDZ
B0B31VTH56
B0B31V3PKG
B0B31T1XMH
B09X1GMWY6
B0B31SM3ZR
B0B31S8KPF
B0B31R1ZBV
B0B31SVP2Q
B0B69DK8JN
B0B69DD2C7
B0B699PLXD
B0B69FBGN1
B0B69GN5KJ
B0B69B8R3C
B0B69G3YRM
B0B69N4B1S
B0B69SF1BT
B0B69C4GST
B0BG2Z8CY4
B0BG2VYWSG
B0BG2PD396
B0BG2T78CJ
B0BG2QXQ9V
B0BGDXV7FD
B0BGDX2Z3L
B0BDRWGMJY
B0BDRQWBK9
B0BDRTZTGX
B0BGDWGRQB
B0BDRS6R5Z
B0BGDZLDX2
B0BGDXM13S
B0BNL2VK7L
B0BNL5BKMK
units
8000
6000
4000
2000
0
2022-07
2022-09
2022-11
2023-01
2023-03
2023-05
2023-07
2023-09
date


B0BNKZ9BJR
B0BNKZKGFG
B0BNKZYHRX
B0BRDCJWJC
B0BRDBKLVJ
B0BNL12NTT
B0BNL43XHH
B0BNL4HPH9
B0BNL4NX61
B0BRDBKB4D
B0BRDCBGFD
B0BRDC22VH
B0BNL4RPB5
B0BNL11QD8
B0BNKXC5H1
B0BNL117VT
B0BNKYPGZQ
B0BNKZFZVL
B0BNKZF51P
B0BNKZH2FW
B0BNKZ88Z5
B0BNKZ5BR1
B0BNKYZD2Z
B0BRCYJ33N
B0BNL5LTYH
B0BNL1J5XS
B0BNL1MQ48
B0BNL1QKLV
B0BNL3NZV7
B0BNL46Y7W
B0BNL1BVRR
B0BNKYW36G
B0BNL4S6MB
B0BNL4VP6Q
B0BNL4WZQV
B0BNL53HF3
B0BNL17RLC
B0BNL5N28Z
B0BRCYQNSW
B0BRCXXBG2
B0BRCXQMS5
B0BRCXDHBX
B0BRCXCXCV
B0BNL6H152
B0BRCYJ4SR
B0BNKY2YCQ
B0BNKZ3G6J
B0BNKZBVQ1
B0BNKZCWFL
B0BNKZFCFT
B0BNL23MTF
B0BNKZYZCC
B0BNL13PNY
B0BNL1RDVC
B0BNL1WL6R
B0BNL2DXZ8
B0BNKZ6N9L
B0BNL3ZHYM
B0BNL2Z6N7
B0BNL3L7LR
B0BNL3RDG4
B0BNL3Y4QM
B0BNL3YY1C
B0BNL3ZV7H
B0BNL49HMH
B0BNL4CB4X
B0BNL4L4K5
B0BNL4R8NS
B0BNL424DV
B0BNL3J36Z
B0BRCXP7VN
B0BNL5CJ8G
B0BNL5FVMF
B0BNL5HYLR
B0BNL5JR7V
B0BNL5LCM5
B0BNL63RCS
B0BNL68VZG
B0BNL71DN2
B0BNL7TJRT
B0BNLBQN9N
B0BNLS6YPG
B0BNL4S7GH
B0BRCXJRX6
B0BNL64PBG




In [5]:

```
submission_df=pd.read_csv('/content/sample_submission.csv')
submission_df.head()
```

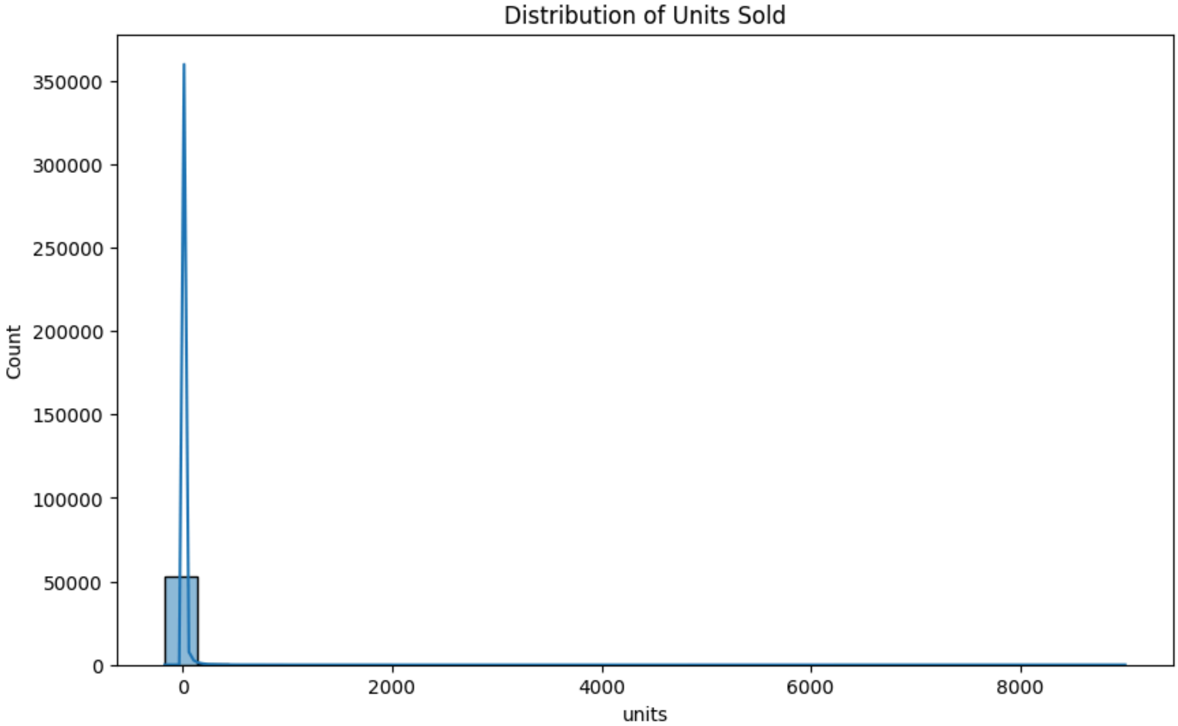
Out[5]:

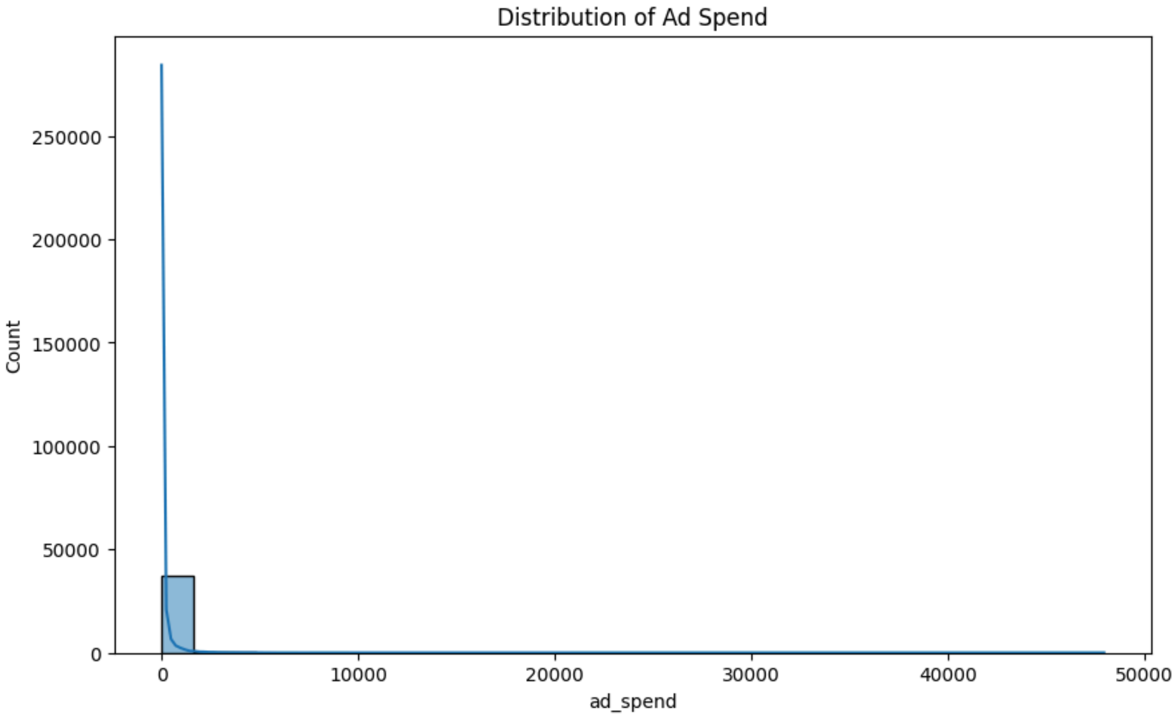
| | ID | TARGET |
|---|---|---|
| **0** | 2024-07-01_B09KDR64LT | 0 |
| **1** | 2024-07-01_B09KDTS4DC | 0 |
| **2** | 2024-07-01_B09KDTHJ6V | 0 |
| **3** | 2024-07-01_B09KDQ2BWY | 0 |
| **4** | 2024-07-01_B09KDYY3SB | 0 |

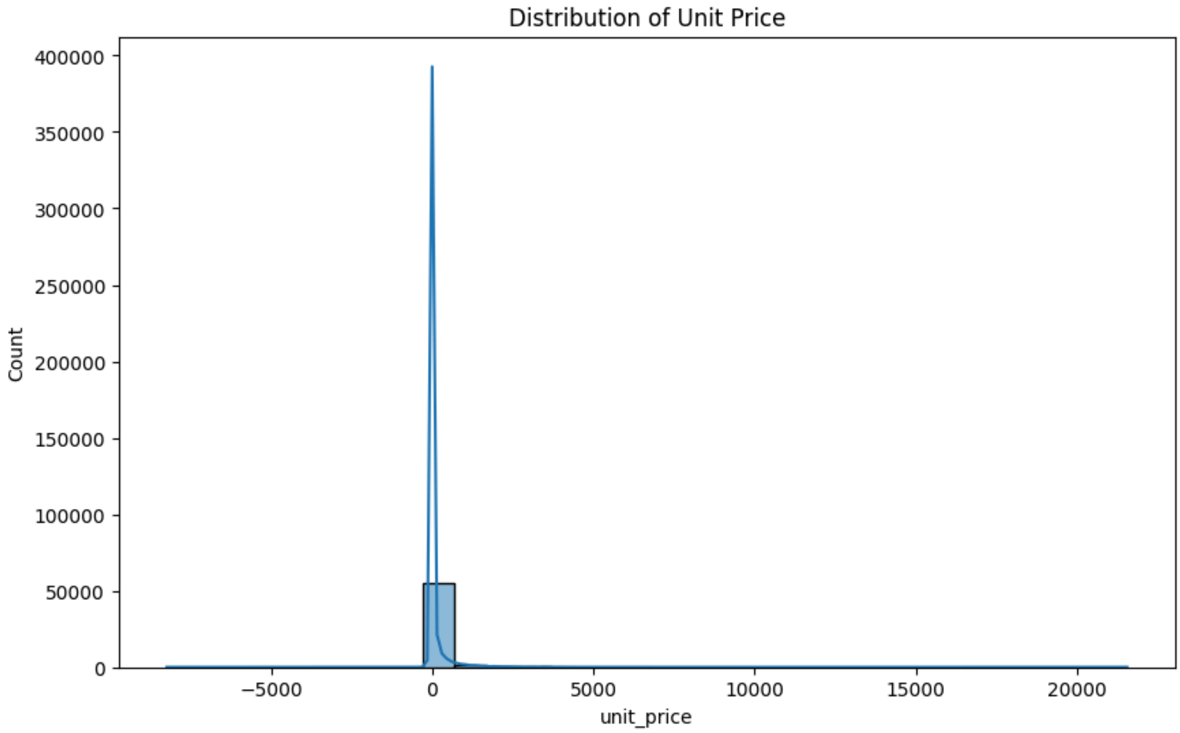
In [6]:

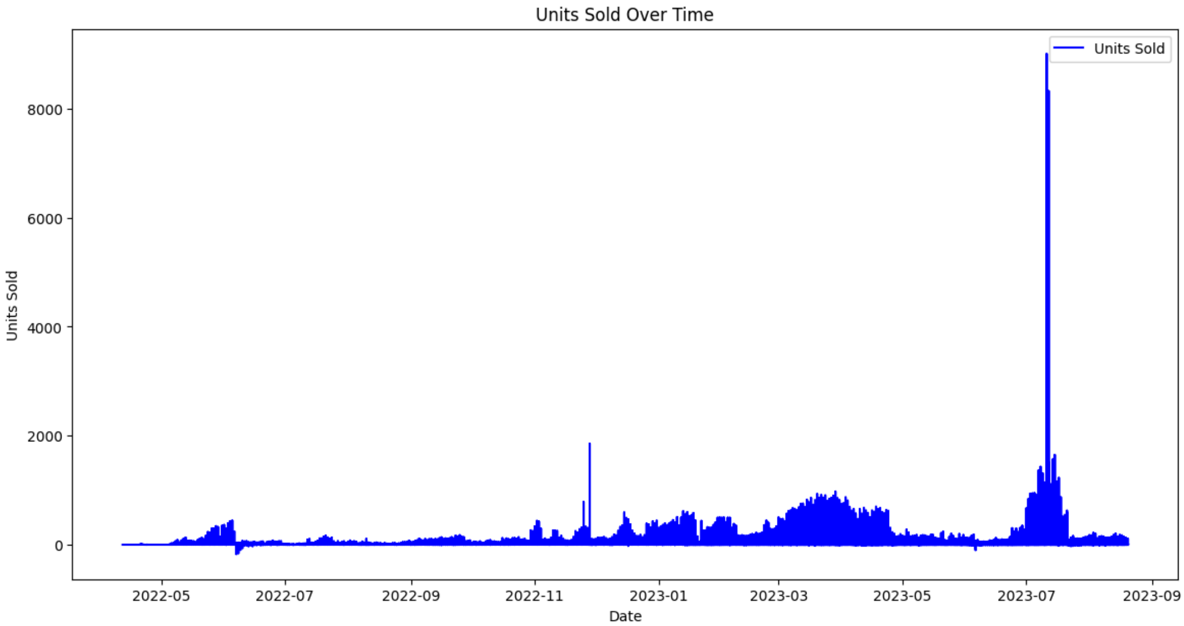
```
#EDA
# Plot distribution of 'units'
plt.figure(figsize=(10, 6))
sns.histplot(train_df['units'].dropna(), bins=30, kde=True)
plt.title('Distribution of Units Sold')
plt.show()

# Plot distribution of 'ad_spend'
plt.figure(figsize=(10, 6))
sns.histplot(train_df['ad_spend'].dropna(), bins=30, kde=True)
plt.title('Distribution of Ad Spend')
plt.show()

# Plot distribution of 'unit_price'
plt.figure(figsize=(10, 6))
sns.histplot(train_df['unit_price'].dropna(), bins=30, kde=True)
plt.title('Distribution of Unit Price')
plt.show()

# 3. Time Series Analysis
# Plot units sold over time
plt.figure(figsize=(14, 7))
plt.plot(train_df['date'], train_df['units'], label='Units Sold', color='blue')
plt.xlabel('Date')
```

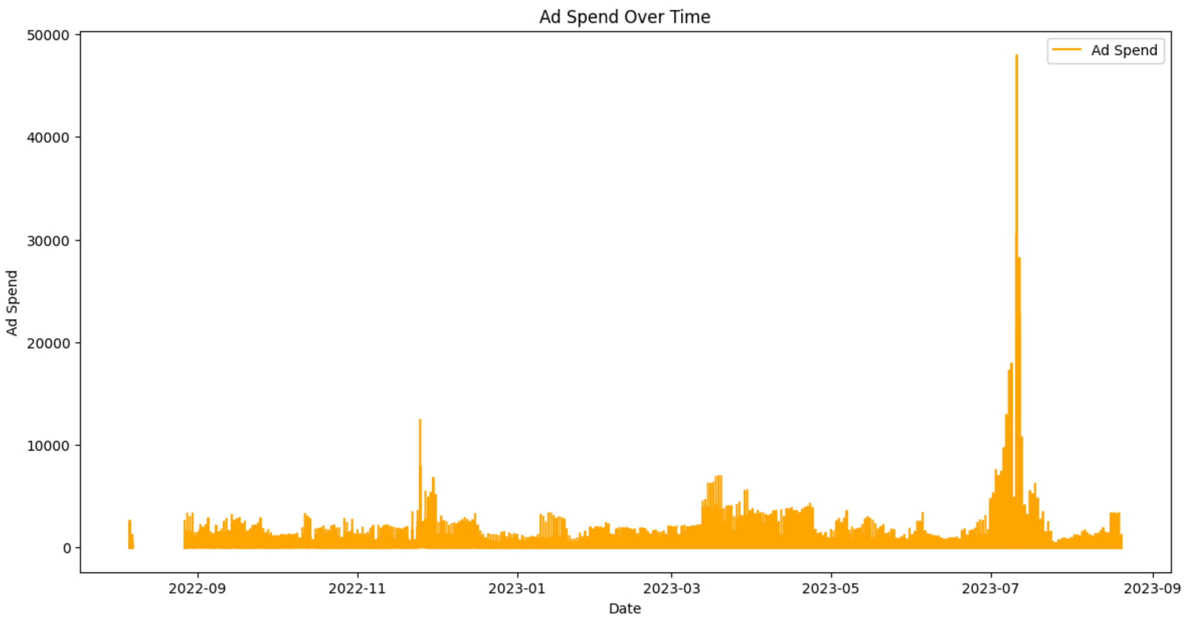
```
plt.ylabel('Units Sold')
plt.title('Units Sold Over Time')
plt.legend()
plt.show()

# Plot ad spend over time
plt.figure(figsize=(14, 7))
plt.plot(train_df['date'], train_df['ad_spend'], label='Ad Spend', color='orange')
plt.xlabel('Date')
plt.ylabel('Ad Spend')
plt.title('Ad Spend Over Time')
plt.legend()
plt.show()

# Select only numeric columns for correlation matrix
numeric_columns = train_df.select_dtypes(include=['float64', 'int64']).columns

# Checking correlation between features
corr_matrix = train_df[numeric_columns].corr()
plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()


# 5. Outlier Detection
# Boxplot for 'units'
plt.figure(figsize=(10, 6))
sns.boxplot(x=train_df['units'].dropna())
plt.title('Boxplot of Units Sold')
plt.show()

# Boxplot for 'ad_spend'
plt.figure(figsize=(10, 6))
sns.boxplot(x=train_df['ad_spend'].dropna())
plt.title('Boxplot of Ad Spend')
plt.show()

# 6. Feature Interactions
# Scatter plot of ad spend vs. units sold
plt.figure(figsize=(10, 6))
sns.scatterplot(x=train_df['ad_spend'], y=train_df['units'])
plt.xlabel('Ad Spend')
plt.ylabel('Units Sold')
plt.title('Ad Spend vs. Units Sold')
plt.show()

# Scatter plot of unit price vs. units sold
plt.figure(figsize=(10, 6))
sns.scatterplot(x=train_df['unit_price'], y=train_df['units'])
plt.xlabel('Unit Price')
plt.ylabel('Units Sold')
plt.title('Unit Price vs. Units Sold')
plt.show()
```

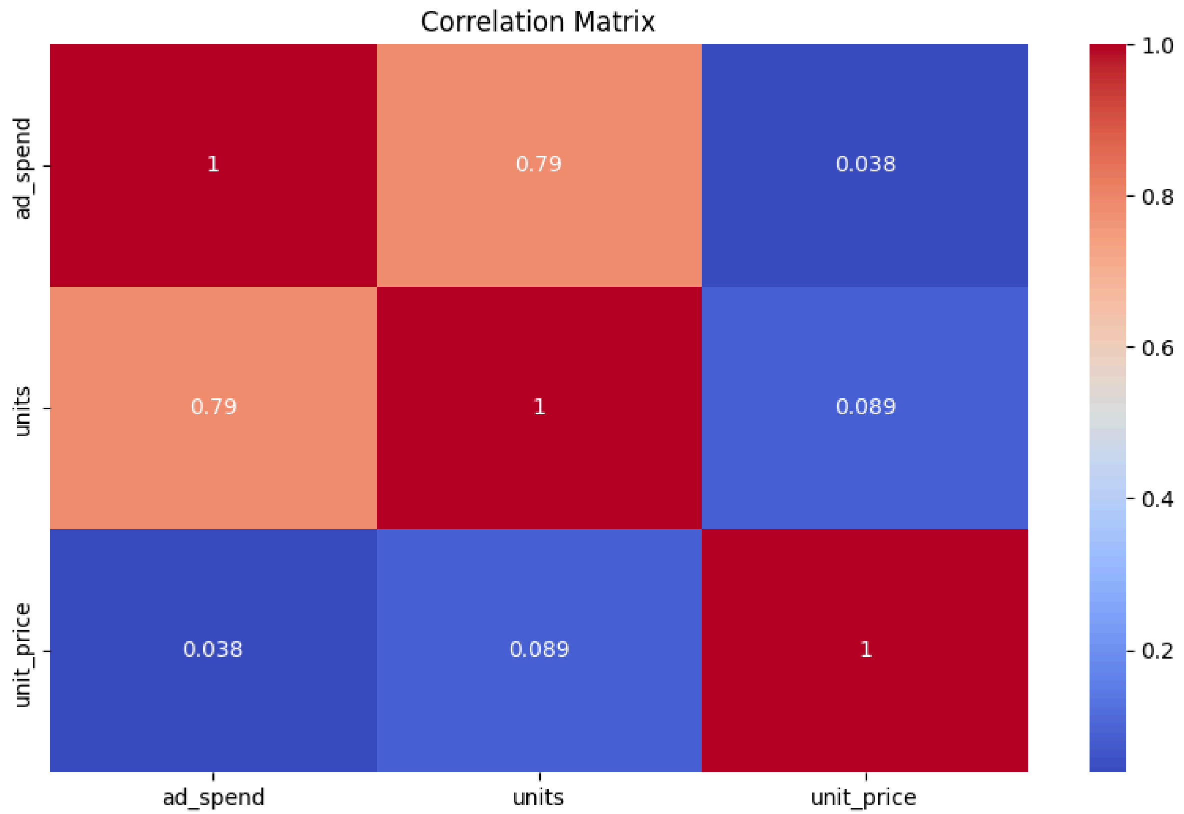

Distribution of Units Sold
350000
300000
250000
200000
150000
100000
50000
0
Count
0
2000
4000
6000
8000
units


Distribution of Ad Spend
250000
200000
150000
100000
50000
0
Count
0
10000
20000
30000
40000
50000
ad_spend

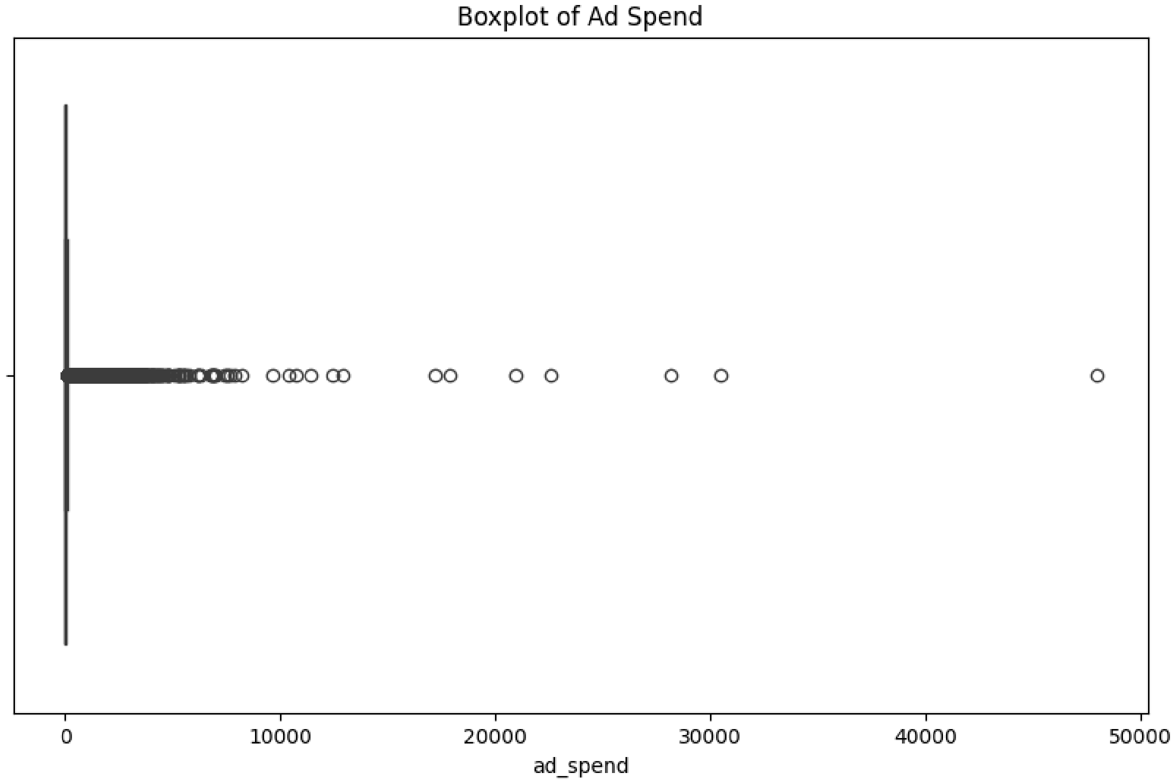

Distribution of Unit Price
Count
unit_price
400000
350000
300000
250000
200000
150000
100000
50000
0
−5000
0
5000
10000
15000
20000

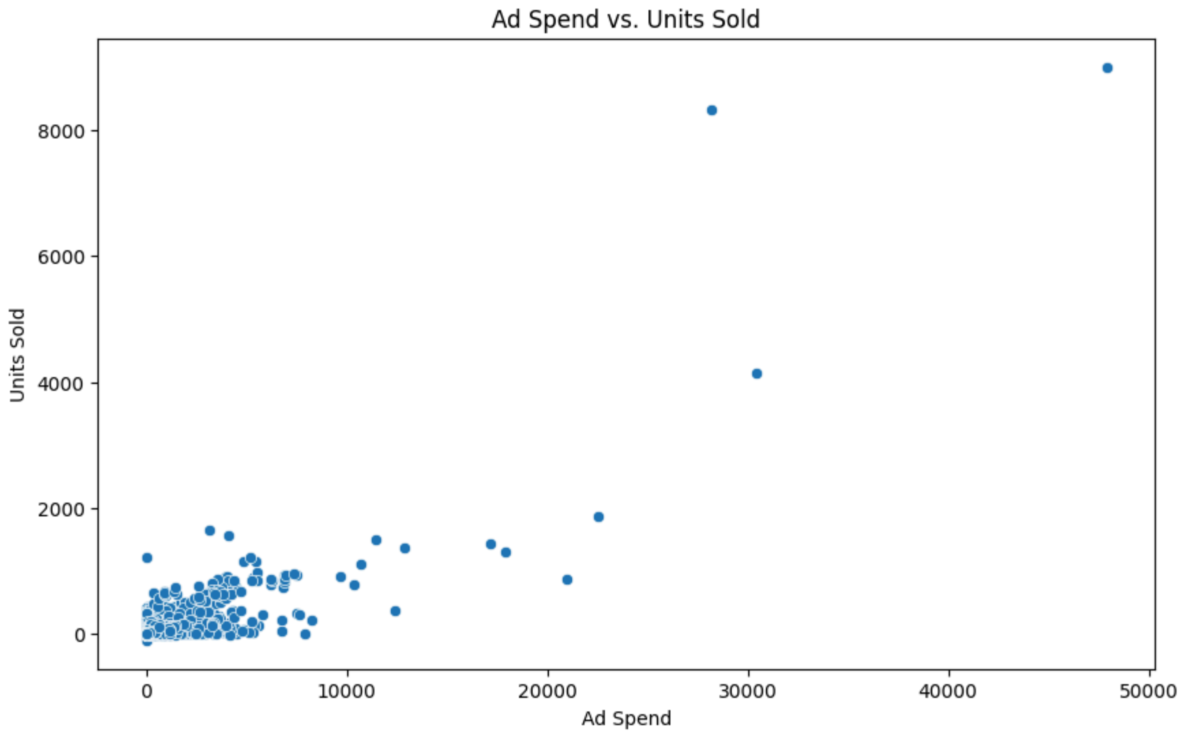

Units Sold Over Time
Units Sold
Units Sold
Date
8000
6000
4000
2000
0
2022-05
2022-07
2022-09
2022-11
2023-01
2023-03
2023-05
2023-07
2023-09

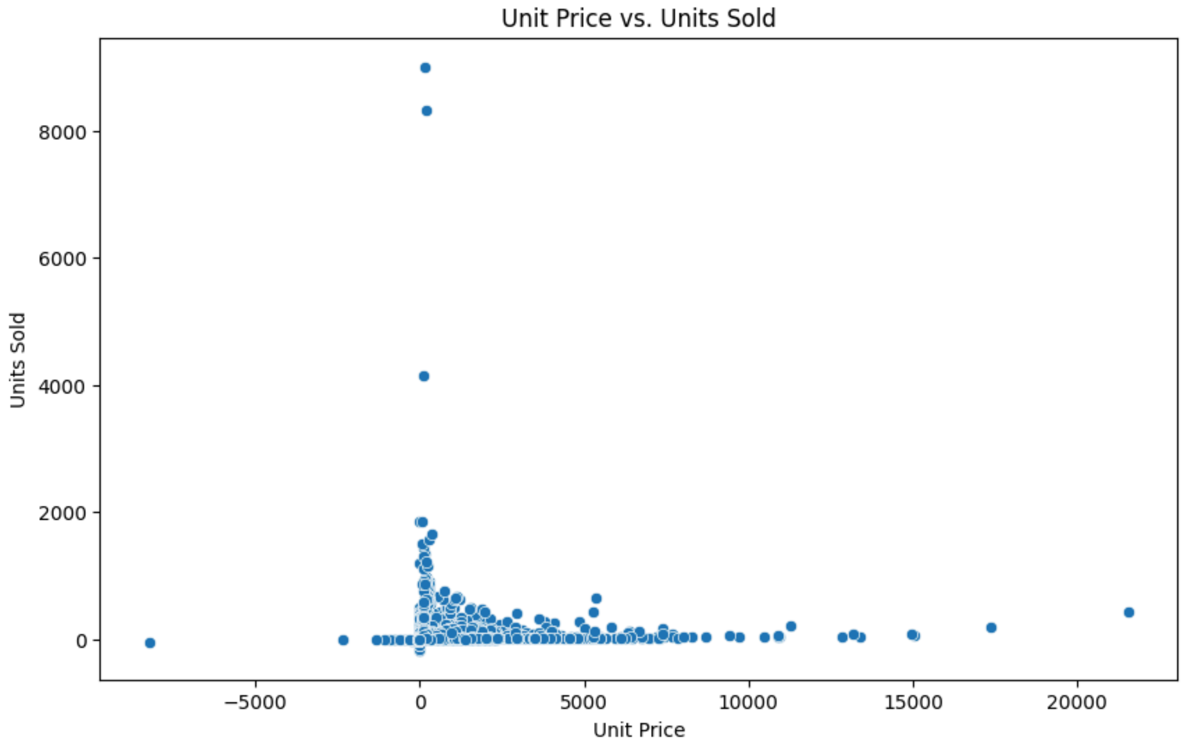

Ad Spend Over Time
Ad Spend
Ad Spend
Date
50000
40000
30000
20000
10000
0
2022-09
2022-11
2023-01
2023-03
2023-05
2023-07
2023-09


Correlation Matrix
ad_spend
units
unit_price
1
0.79
0.038
0.79
1
0.089
0.038
0.089
1
1.0
0.8
0.6
0.4
0.2


Boxplot of Units Sold
0
2000
4000
6000
8000
units


Boxplot of Ad Spend
0
10000
20000
30000
40000
50000
ad_spend


Ad Spend vs. Units Sold
8000
6000
4000
2000
0
Units Sold
0
10000
20000
30000
40000
50000
Ad Spend




In [7]:
```
# Missing values sum
print(train_df.isnull().sum())
print(test_df.isnull().sum())
```

```
ID                0
date              0
Item Id           0
Item Name         1
ad_spend      19457
anarix_id         1
units          3914
unit_price        1
dtype: int64
ID               0
date             0
Item Id          0
Item Name      344
ad_spend      1451
anarix_id        0
unit_price       0
dtype: int64
```

In [8]:
```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from xgboost import XGBRegressor
from sklearn.svm import SVR
import pandas as pd

# Check columns and data types
print(train_df.dtypes)

#preprocessing
print("Missing values in target variable (y):")
print(train_df['units'].isna().sum())
```

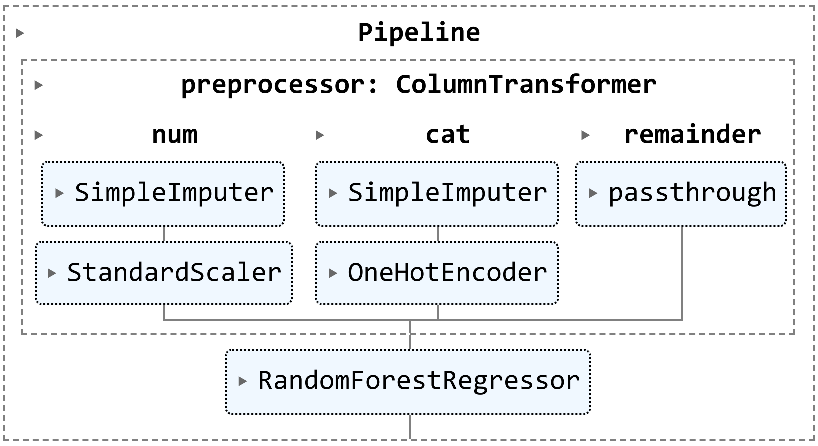
```
train_df_clean = train_df.dropna(subset=['units'])
X = train_df_clean.drop(columns=['units', 'date', 'ID'])
y = train_df_clean['units']

# Feature extraction from date
if 'month' not in X.columns:
    X['month'] = train_df_clean['date'].dt.month
    X['day_of_week'] = train_df_clean['date'].dt.dayofweek

# Define preprocessing for numerical and categorical data
num_features = ['ad_spend', 'unit_price', 'month', 'day_of_week']
cat_features = ['Item Name', 'Item Id', 'anarix_id']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='mean')),
            ('scaler', StandardScaler())
        ]), num_features),
        ('cat', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('onehot', OneHotEncoder(handle_unknown='ignore'))
        ]), cat_features)
    ],
    remainder='passthrough'
)

# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state

#train with three different models
rf_model = RandomForestRegressor(random_state=42)
xgb_model = XGBRegressor(random_state=42)
svr_model = SVR()

# Create pipelines
rf_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('model', rf_model)])

xgb_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('model', xgb_model)])

svr_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('model', svr_model)])

# Train all three  models
rf_pipeline.fit(X_train, y_train)
xgb_pipeline.fit(X_train, y_train)
svr_pipeline.fit(X_train, y_train)

# Evaluate three models which are trained
rf_pred = rf_pipeline.predict(X_val)
xgb_pred = xgb_pipeline.predict(X_val)
svr_pred = svr_pipeline.predict(X_val)

# Calculate MSE(mean square error for three models)
rf_mse = mean_squared_error(y_val, rf_pred)
xgb_mse = mean_squared_error(y_val, xgb_pred)
svr_mse = mean_squared_error(y_val, svr_pred)

print(f'Random Forest MSE: {rf_mse}')
print(f'XGBoost MSE: {xgb_mse}')
print(f'SVR MSE: {svr_mse}')
```

```
# Select the best model
best_model = rf_pipeline if rf_mse < xgb_mse and rf_mse < svr_mse else xgb_pipeline
```

```
ID                    object
date          datetime64[ns]
Item Id               object
Item Name             object
ad_spend             float64
anarix_id             object
units                float64
unit_price           float64
dtype: object
Missing values in target variable (y):
3914
Random Forest MSE: 275.543679321237
XGBoost MSE: 419.23154740512797
SVR MSE: 1304.9109254919879
```

In [9]:
```
best_model
```

Out[9]:




In [ ]:
```
#hyperparameter tuning
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for the RandomForestRegressor within the pipeline
param_grid = {
    'model__n_estimators': [100, 200, 500],
    'model__max_depth': [None, 10, 20, 30],
    'model__min_samples_split': [2, 5, 10],
    'model__min_samples_leaf': [1, 2, 4]
}

# Use GridSearchCV to search for the best hyperparameters
grid_search = GridSearchCV(estimator=rf_pipeline, param_grid=param_grid, cv=3, scor
grid_search.fit(X_train, y_train)

# Get the best model from the grid search
best_model = grid_search.best_estimator_

# Re-evaluate with the tuned model
best_pred = best_model.predict(X_val)
best_mse = mean_squared_error(y_val, best_pred)
print(f'Tuned Model MSE: {best_mse}')
```

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/process_executor.py:
752: UserWarning: A worker stopped while some jobs were given to the executor. Thi
s can be caused by a too short worker timeout or by a memory leak.
  warnings.warn(
```

In [12]:
```
test_df.head()
```

Out[12]:

| | ID | date | Item Id | Item Name | ad_spend | anarix_id | unit_price |
|---|---|---|---|---|---|---|---|
| **0** | 2024-07-01_B09KDR64LT | 2024-07-01 | B09KDR64LT | NapQueen Elizabeth 10" Gel Memory Foam Mattres... | NaN | NAPQUEEN | 0.0 |
| **1** | 2024-07-01_B09KDTS4DC | 2024-07-01 | B09KDTS4DC | NapQueen Elizabeth 8" Gel Memory Foam Mattress... | NaN | NAPQUEEN | 0.0 |
| **2** | 2024-07-01_B09KDTHJ6V | 2024-07-01 | B09KDTHJ6V | NapQueen Elizabeth 12" Gel Memory Foam Mattres... | NaN | NAPQUEEN | 0.0 |
| **3** | 2024-07-01_B09KDQ2BWY | 2024-07-01 | B09KDQ2BWY | NapQueen Elizabeth 12" Gel Memory Foam Mattres... | NaN | NAPQUEEN | 0.0 |
| **4** | 2024-07-01_B09KDYY3SB | 2024-07-01 | B09KDYY3SB | NapQueen Elizabeth 10" Gel Memory Foam Mattres... | 101.72 | NAPQUEEN | 1094.5 |

In [13]:
```
test_df.columns
```

Out[13]:
```
Index(['ID', 'date', 'Item Id', 'Item Name', 'ad_spend', 'anarix_id',
       'unit_price'],
      dtype='object')
```

In [ ]:
```
rf_model = RandomForestRegressor(random_state=42)

# Define the parameter grid for GridSearchCV
param_grid = {
    'model__n_estimators': [100, 200, 300],
    'model__max_depth': [None, 10, 20, 30],
    'model__min_samples_split': [2, 5, 10],
    'model__min_samples_leaf': [1, 2, 4]
}

# Create the pipeline
rf_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('model', rf_model)])

# Grid Search for hyperparameter tuning
grid_search = GridSearchCV(
    rf_pipeline,
    param_grid=param_grid,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=1
)

# Fit the grid search model
grid_search.fit(X_train, y_train)

# Get the best model
best_rf_pipeline = grid_search.best_estimator_
```

```
# Evaluate the best model
rf_pred = best_rf_pipeline.predict(X_val)
rf_mse = mean_squared_error(y_val, rf_pred)
print(f'Optimized Random Forest MSE: {rf_mse}')

# Cross-validation score
from sklearn.model_selection import cross_val_score

rf_scores = cross_val_score(best_rf_pipeline, X, y, cv=5, scoring='neg_mean_squarec
print(f'Random Forest Cross-Validation MSE: {-rf_scores.mean()}')
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
```

In [29]:

```
X_test = test_df.drop(columns=['date', 'ID'])

# Predict using the best model
test_df['units'] = best_model.predict(X_test)

submission_df['Target'] = test_df['units']
submission_df.to_csv('submission.csv', index=False)

print("Submission file created.")
```

```
Submission file created.
```

In [26]:

```
from google.colab import files

files.download('submission.csv')
```

In [ ]:

```
#EXPLANATION
In this given task, predicting the forecast of sales with time series data is our
primary goal. We began with the examination and investigation of the dataset by thi
we were able to find some trends and correlations in it that
corresponded to sales trends and interaction between ad spend and units sold.
This phase aims for handling missing  values and the extraction of time-related
features in the data preprocessing step, numeral variables will be standardized
and getting categorical ones will be one-hot encoded. Three models were trained:
Random Forest Regressor, XGBoost Regressor,
and Support Vector Regressor. The way their performance was assessed was using
the value mean squared error. The model  which was chosen as the best one was
"Random Forest" and then the hyperparameters were tuned with the help of GridSearch
which is a famous cross-validation technique. Then, the final model was used to pre
the sales of the test set and the results were saved for submission file. By seeing
trends of the data and by realizing that these models were capable
of explaining the intricate relationships between features in a robust way, which i
suitable approach for solving the
problem of forecasting the sales, we have chosen the method described.
```