TASK 2

predicting unit sales without using ad spend data.

Exploratory Data Analysis (EDA)

In [1]:
```
import pandas as pd
import numpy as np

data = pd.read_csv('train.csv')

data['date'] = pd.to_datetime(data['date'], format='%Y-%m-%d')
data['year'] = data['date'].dt.year
data['month'] = data['date'].dt.month
data['day'] = data['date'].dt.day
data['dayofweek'] = data['date'].dt.dayofweek

data = data.sort_values(by=['Item Id', 'date'])
```

Feature Engineering

In [2]:
```
features = ['year', 'month', 'day', 'dayofweek', 'lag_1', 'lag_2', 'lag_3', 'lag_4', 'lag_5', 'lag_6', 'lag_7', 'rolling_mean_7', 'rolling_sum_7']
target = 'units'

for lag in range(1, 8):
    data[f'lag_{lag}'] = data.groupby('Item Id')['units'].shift(lag)

data['rolling_mean_7'] = data.groupby('Item Id')['units'].transform(lambda x: x.rolling(window=7).mean())
data['rolling_sum_7'] = data.groupby('Item Id')['units'].transform(lambda x: x.rolling(window=7).sum())

data.dropna(inplace=True)

X = data[features]
y = data[target]

test_data = pd.read_csv('test.csv')

test_data['date'] = pd.to_datetime(test_data['date'], format='%Y-%m-%d')

test_data['year'] = test_data['date'].dt.year
test_data['month'] = test_data['date'].dt.month
test_data['day'] = test_data['date'].dt.day
test_data['dayofweek'] = test_data['date'].dt.dayofweek

for lag in range(1, 8):
    test_data[f'lag_{lag}'] = np.nan

test_data['rolling_mean_7'] = np.nan
test_data['rolling_sum_7'] = np.nan
```

Model Selection

In [3]:
```
import xgboost as xgb
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import VotingRegressor

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

xgb_model = xgb.XGBRegressor(objective='reg:squarederror')
xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_val)
mse = mean_squared_error(y_val, y_pred)
print(f'XGBoost Mean Squared Error: {mse}')

lgb_model = lgb.LGBMRegressor()
lgb_model.fit(X_train, y_train)

voting_model = VotingRegressor(estimators=[
    ('xgb', xgb_model),
    ('lgb', lgb_model)
])

voting_model.fit(X_train, y_train)

y_pred_voting = voting_model.predict(X_val)
mse_voting = mean_squared_error(y_val, y_pred_voting)
print(f'Voting Regressor Mean Squared Error: {mse_voting}')
```

```
XGBoost Mean Squared Error: 18919.183501297714
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000655 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 2351
[LightGBM] [Info] Number of data points in the train set: 41694, number of used features: 13
[LightGBM] [Info] Start training from score 13.272677
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.006973 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2351
[LightGBM] [Info] Number of data points in the train set: 41694, number of used features: 13
[LightGBM] [Info] Start training from score 13.272677
Voting Regressor Mean Squared Error: 9002.894698495329
```

Hyperparameter Tuning

In [4]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1]
}

grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=3, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
final_xgb_model = xgb.XGBRegressor(objective='reg:squarederror', **best_params)
final_xgb_model.fit(X_train, y_train)

y_pred_final = final_xgb_model.predict(X_val)
mse_final = mean_squared_error(y_val, y_pred_final)
print(f'Final XGBoost Mean Squared Error: {mse_final}')
print(f'Voting Regressor Mean Squared Error: {mse_voting}')


X_test = test_data[features]
test_data['TARGET'] = voting_model.predict(X_test)

submission = test_data[['date', 'Item Id', 'TARGET']]
submission.to_csv('Task2_submission.csv', index=False)
```

```
Final XGBoost Mean Squared Error: 8585.214285165122
Voting Regressor Mean Squared Error: 9002.894698495329
```

When predicting without using ad spend data, I've noticed that the MSE of the Voting Regressor is higher than that of the final XGBoost model. This suggests that, in this specific scenario, the Voting Regressor may not be leveraging its ensemble advantage effectively. The individual models might not be complementing each other well or might not be capturing the data's patterns as effectively as XGBoost on its own. This could imply that XGBoost is more adept at handling the data in the absence of ad spend information, possibly due to its specialized boosting capabilities and hyperparameter tuning.

In [ ]:

```

```