TASK 1

Given a dummy sales data of a well-known brand on Amazon ,building a time series forecasting model that predicts the number of units sold for each item ID.

EXPLORATORY DATA ANALYSIS (EDA)

In [2]:
```
import pandas as pd
import numpy as np
# Loading the dataset
data = pd.read_csv('train.csv')
# Converting 'date' column to datetime format
data['date'] = pd.to_datetime(data['date'], format='%Y-%m-%d')
# Extracting date-related features
data['year'] = data['date'].dt.year
data['month'] = data['date'].dt.month
data['day'] = data['date'].dt.day
data['dayofweek'] = data['date'].dt.dayofweek
# Sorting the data by date and item ID
data = data.sort_values(by=['Item Id', 'date'])
```

FEATURE ENGINEERING

In [3]:
```
# Dropping 'Item Id' from the features
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', 'ad_spend']
target = 'units'
# Creating lag features for the data
for lag in range(1, 8):
    data[f'lag_{lag}'] = data.groupby('Item Id')['units'].shift(lag)
# Creating rolling/window features for the data
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())
# Dropping rows with NaN values generated by lag and rolling features
data.dropna(inplace=True)
# Selecting features and target variable
X = data[features]
y = data[target]
# Loading the test dataset
test_data = pd.read_csv('test.csv')
# Converting 'date' column to datetime format
test_data['date'] = pd.to_datetime(test_data['date'], format='%Y-%m-%d')
# Extracting date-related features
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
# Ensuring all necessary columns exist in the test set
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

I've found that XGBoost is a powerful gradient boosting model that's particularly effective for regression tasks. I appreciate its ability to handle large datasets and capture complex relationships between features through its boosting techniques. Its flexibility in dealing with different types of data and the fine-tuning options available for hyperparameters make it a go-to choice for predictive analytics.

On the other hand, I've also worked with LightGBM, which is another gradient boosting framework that excels in speed and efficiency, especially with large datasets. I like that it uses histogram-based algorithms to speed up training and reduce memory usage. This makes it a great option for time series forecasting, where the volume of data can be quite substantial.

To leverage the strengths of both XGBoost and LightGBM, I use a Voting Regressor. By averaging predictions from both models, the Voting Regressor helps improve overall accuracy and stability. This ensemble approach allows me to capitalize on the strengths of each model while mitigating their weaknesses, often resulting in better generalization and more reliable predictions compared to using a single model.

In [4]:
```
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
# Splitting the 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=42)
# Training an XGBoost model
xgb_model = xgb.XGBRegressor(objective='reg:squarederror')
xgb_model.fit(X_train, y_train)
# Making predictions and evaluate the model
y_pred = xgb_model.predict(X_val)
mse = mean_squared_error(y_val, y_pred)
print(f'XGBoost Mean Squared Error: {mse}')
# Training a LightGBM model
lgb_model = lgb.LGBMRegressor()
lgb_model.fit(X_train, y_train)
# preparing an Ensemble model with Voting Regressor
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: 5841.064198659013
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001471 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 2606
[LightGBM] [Info] Number of data points in the train set: 41694, number of used features: 14
[LightGBM] [Info] Start training from score 13.272677
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.005350 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2606
[LightGBM] [Info] Number of data points in the train set: 41694, number of used features: 14
[LightGBM] [Info] Start training from score 13.272677
Voting Regressor Mean Squared Error: 4971.752984199898
```

HYPER PARAMETER TUNING

In [6]:
```
from sklearn.metrics import mean_squared_error

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}')

# Hyperparameter tuning for XGBoost
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)

# Making predictions and evaluating the final XGBoost model
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}')

# Making predictions on the test dataset
X_test = test_data[features]
test_data['TARGET'] = voting_model.predict(X_test)

# Creating the submission file
submission = test_data[['date', 'Item Id', 'TARGET']]
submission.to_csv('Task1_submission.csv', index=False)
```

```
Voting Regressor Mean Squared Error: 4971.752984199898
Final XGBoost Mean Squared Error: 6895.246074568306
```

```
In [ ]:
```