```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import xgboost as xgb
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')

# Load data
train_df = pd.read_csv('/content/drive/MyDrive/train.csv')
test_df = pd.read_csv('/content/drive/MyDrive/test.csv')

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

# Feature Engineering: Create lag features and rolling statistics
without ad spend data
for lag in range(1, 8):
    train_df[f'lag_{lag}'] = train_df['units'].shift(lag)

train_df['rolling_mean_7'] =
train_df['units'].rolling(window=7).mean()
train_df['rolling_std_7'] = train_df['units'].rolling(window=7).std()

# Drop rows with NaN values created by lag features
train_df.dropna(inplace=True)

# For test data, we'll need to handle it carefully since there's no
'units' column
test_lag_features = pd.concat([train_df[['date',
'units']].tail(7).shift(i) for i in range(1, 8)], axis=1)
# Generate 14 column names because we concatenated 7 DataFrames with 2
columns each
test_lag_features.columns = [f'lag_{i}_{j}' for i in range(1, 8) for j
in range(2)]
test_lag_features = test_lag_features.iloc[7:]
test_df = pd.concat([test_df.reset_index(drop=True),
test_lag_features.reset_index(drop=True)], axis=1)

test_df['rolling_mean_7'] =
train_df['units'].rolling(window=7).mean().iloc[-1]
test_df['rolling_std_7'] =
train_df['units'].rolling(window=7).std().iloc[-1]
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```python
!pip install --upgrade scikit-learn

# Convert 'Item Id' and 'anarix_id' to numerical representations
# Assuming they are categorical, you can use Label Encoding or One-Hot Encoding

# Example using Label Encoding:
from sklearn.preprocessing import LabelEncoder

label_encoder_item = LabelEncoder() # Create a separate encoder for 'Item Id'
label_encoder_anarix = LabelEncoder() # Create a separate encoder for 'anarix_id'

# Fit and transform on the training data
X_train['Item Id'] = label_encoder_item.fit_transform(X_train['Item Id'])
X_train['anarix_id'] = label_encoder_anarix.fit_transform(X_train['anarix_id'])

# For handling unknown labels in the validation set, we'll manually map them to a new value
X_valid['Item Id'] = X_valid['Item Id'].map(lambda s: '<unknown>' if s not in label_encoder_item.classes_ else s)
X_valid['anarix_id'] = X_valid['anarix_id'].map(lambda s: '<unknown>' if s not in label_encoder_anarix.classes_ else s)

# Add the '<unknown>' label to the encoders
import bisect
le_classes_item = label_encoder_item.classes_.tolist()
# Convert all elements in le_classes_item to strings to ensure consistency
le_classes_item = [str(item) for item in le_classes_item]
bisect.insort_left(le_classes_item, '<unknown>')
label_encoder_item.classes_ = np.array(le_classes_item) # Convert the list back to a NumPy array

le_classes_anarix = label_encoder_anarix.classes_.tolist()
# Convert all elements in le_classes_anarix to strings to ensure consistency
le_classes_anarix = [str(item) for item in le_classes_anarix]
bisect.insort_left(le_classes_anarix, '<unknown>')
label_encoder_anarix.classes_ = np.array(le_classes_anarix) # Convert the list back to a NumPy array

# Now transform the validation data
X_valid['Item Id'] = label_encoder_item.transform(X_valid['Item Id'])
X_valid['anarix_id'] = label_encoder_anarix.transform(X_valid['anarix_id'])
```

```python
# Now try fitting the model again using the encoded data
model.fit(X_train, y_train,
          eval_set=[(X_valid, y_valid)],
          callbacks=[early_stopping(stopping_rounds=50,
verbose=True)])
```

```
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.10/dist-packages (1.5.1)
Requirement already satisfied: numpy>=1.19.5 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.005308 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2747
[LightGBM] [Info] Number of data points in the train set: 27104,
number of used features: 11
[LightGBM] [Warning] Found whitespace in feature_names, replace with
underlines
[LightGBM] [Info] Start training from score 12.661194
Training until validation scores don't improve for 50 rounds
Early stopping, best iteration is:
[225] valid_0's l2: 4047.99
```

```
LGBMRegressor(learning_rate=0.05, n_estimators=1000)
```

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
import xgboost as xgb

# Load test data
test_df = pd.read_csv('/content/drive/MyDrive/test.csv')

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

# Feature Engineering on test data without 'units'
# Here, we will use the last available values from the train data for
rolling features and lags
last_train_date = train_df['date'].max()

# Create lag features and rolling statistics based on the last
```

```
available data from the training set
last_week_data = train_df[train_df['date'] > last_train_date -
pd.Timedelta(days=7)]
for lag in range(1, 8):
    test_df[f'lag_{lag}'] = last_week_data['units'].values[-lag]

# Rolling mean and std based on the last week of the training data
test_df['rolling_mean_7'] = last_week_data['units'].mean()
test_df['rolling_std_7'] = last_week_data['units'].std()

# Convert 'Item Id' and 'anarix_id' to numerical representations using
Label Encoders created during training
test_df['Item Id'] = test_df['Item Id'].map(lambda s: '<unknown>' if s
not in label_encoder_item.classes_ else s)
test_df['anarix_id'] = test_df['anarix_id'].map(lambda s: '<unknown>'
if s not in label_encoder_anarix.classes_ else s)

# Add the '<unknown>' label to the encoders
import bisect
le_classes_item = label_encoder_item.classes_.tolist()
le_classes_item = [str(item) for item in le_classes_item]  # Ensure
all elements are strings
bisect.insort_left(le_classes_item, '<unknown>')
label_encoder_item.classes_ = np.array(le_classes_item)  # Convert the
list back to a NumPy array

le_classes_anarix = label_encoder_anarix.classes_.tolist()
le_classes_anarix = [str(item) for item in le_classes_anarix]  #
Ensure all elements are strings
bisect.insort_left(le_classes_anarix, '<unknown>')
label_encoder_anarix.classes_ = np.array(le_classes_anarix)  # Convert
the list back to a NumPy array

# Now transform the test data
test_df['Item Id'] = label_encoder_item.transform(test_df['Item Id'])
test_df['anarix_id'] =
label_encoder_anarix.transform(test_df['anarix_id'])

# Features for prediction
features = [col for col in test_df.columns if col not in ['date',
'units', 'ID', 'ad_spend', 'Item Name']]

# Generate predictions for the test set
test_preds = model.predict(test_df[features])
test_df['units'] = test_preds

# Ensure the predictions include the ID column from the test set
submission = test_df[['ID', 'units']]

# Save the predictions to CSV
```

```
submission.to_csv('/content/drive/MyDrive/sample_submission.csv',
index=False)
```