In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import metrics
import warnings
```

In [2]:
```
warnings.filterwarnings('ignore')
```

Data preprocessing

In [3]:
```
data = pd.read_csv('restaurant/sales_with_extra_features4.csv')
```

In [4]:
```
data.shape
```

Out[4]: (715, 22)

In [5]:
```
data.describe()
```

Out[5]:

	Unnamed: 0	Sales	Day before holiday	Non Holiday date	Day after holiday	Non Holiday	Payment Week
count	715.000000	715.000000	715.000000	715.000000	715.000000	715.00000	715.000000
mean	357.000000	126.946853	0.043357	0.930070	0.027972	0.98042	0.304895
std	206.547008	23.253324	0.203801	0.255208	0.165008	0.13865	0.460685
min	0.000000	73.000000	0.000000	0.000000	0.000000	0.00000	0.000000
25%	178.500000	110.000000	0.000000	1.000000	0.000000	1.00000	0.000000
50%	357.000000	128.000000	0.000000	1.000000	0.000000	1.00000	0.000000
75%	535.500000	140.000000	0.000000	1.000000	0.000000	1.00000	1.000000
max	714.000000	196.000000	1.000000	1.000000	1.000000	1.00000	1.000000

8 rows × 21 columns

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 715 entries, 0 to 714
Data columns (total 22 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Unnamed: 0              715 non-null    int64  
 1   Date                    715 non-null    object 
 2   Sales                   715 non-null    int64  
 3   Day before holiday      715 non-null    int64  
 4   Non Holiday date        715 non-null    int64  
 5   Day after holiday       715 non-null    int64  
 6   Non Holiday             715 non-null    int64  
 7   Payment Week            715 non-null    int64  
 8   Weekend                 715 non-null    int64  
 9   No. of competitors      715 non-null    int64  
 10  Quarterly Average sale  715 non-null    int64  
 11  Montly Average sale     715 non-null    int64  
 12  Qtrly sales std dev     715 non-null    int64  
 13  Monthly sale std dev    715 non-null    int64  
 14  dayofweek               715 non-null    int64  
 15  month                   715 non-null    int64  
 16  sin_dow                 715 non-null    float64
 17  lag_1                   714 non-null    float64
 18  lag_7                   708 non-null    float64
 19  lag_3                   712 non-null    float64
 20  lag_5                   710 non-null    float64
 21  rolling_mean_7          709 non-null    float64
dtypes: float64(6), int64(15), object(1)
memory usage: 123.0+ KB
```

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
Unnamed: 0              0
Date                    0
Sales                   0
Day before holiday      0
Non Holiday date        0
Day after holiday       0
Non Holiday             0
Payment Week            0
Weekend                 0
No. of competitors      0
Quarterly Average sale  0
Montly Average sale     0
Qtrly sales std dev     0
Monthly sale std dev    0
dayofweek               0
month                   0
sin_dow                 0
lag_1                   1
lag_7                   7
lag_3                   3
lag_5                   5
rolling_mean_7          6
dtype: int64
```

In [8]:
```
data['Date'] = pd.to_datetime(data['Date'])
```

In [9]:
```
data = data.sort_values('Date')
```

In [10]:
```
data.dropna(inplace=True)
```

In [11]:
```
data = data.set_index('Date')
```

In [12]:
```
data.drop('Unnamed: 0', axis=1, inplace=True)
```

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

Out[13]:
```
Index(['Sales', 'Day before holiday', 'Non Holiday date', 'Day after holiday',
       'Non Holiday', 'Payment Week', 'Weekend', 'No. of competitors',
       'Quarterly Average sale', 'Montly Average sale', 'Qtrly sales std dev',
       'Monthly sale std dev', 'dayofweek', 'month', 'sin_dow', 'lag_1',
       'lag_7', 'lag_3', 'lag_5', 'rolling_mean_7'],
      dtype='object')
```

feature selection

In [14]:
```
features = ['Day before holiday','Non Holiday date',
       'Day after holiday', 'Non Holiday',
       'Payment Week', 'Weekend',
       'No. of competitors', 'Quarterly Average sale',
        'Montly Average sale', 'Qtrly sales std dev',
       'Monthly sale std dev', 'dayofweek', 'month', 'sin_dow', 'lag_1',
       'lag_7', 'lag_3', 'lag_5', 'rolling_mean_7']
target = ['Sales']
```

In [15]:
```
x = data[features]
y = data[target]
```

In [16]:
```
xgb_x_train, xgb_x_test = x[:-30], x[-30:]
xgb_y_train, xgb_y_test = y[:-30], y[-30:]
```

training xgboost model

In [17]:
```
from xgboost import XGBRegressor
```

In [18]:
```
xgb_model = XGBRegressor(n_estimators=340, learning_rate=0.08, max_depth=4, random_
```

In [19]:
```
xgb_model.fit(xgb_x_train, xgb_y_train)
```

Out[19]:

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=
None,
             enable_categorical=False, eval_metric=None, feature_types=
None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.08, max_bin=None, max_cat_threshold=None,
```

In [20]:
```
xgb_ypred = xgb_model.predict(xgb_x_test)
```

evaluation metrics

In [21]:
```
def eval_metrics(y_true, y_pred):
    print(f'MSE is : {metrics.mean_squared_error(y_true, y_pred)}')
    print(f'MAE is : {metrics.mean_absolute_error(y_true, y_pred)}')
    print(f'RMSE is : {metrics.root_mean_squared_error(y_true, y_pred)}')
    print(f'MAPE is : {metrics.mean_absolute_percentage_error(y_true, y_pred)}')
    print(f'R2 is :{metrics.r2_score(y_true, y_pred)}')
```

xgboost result evaluation

In [22]:
```
eval_metrics(xgb_y_test, xgb_ypred)
```

```
MSE is : 140.56015014648438
MAE is : 9.963960647583008
RMSE is : 11.855806350708008
MAPE is : 0.09865555912256241
R2 is :0.49264609813690186
```

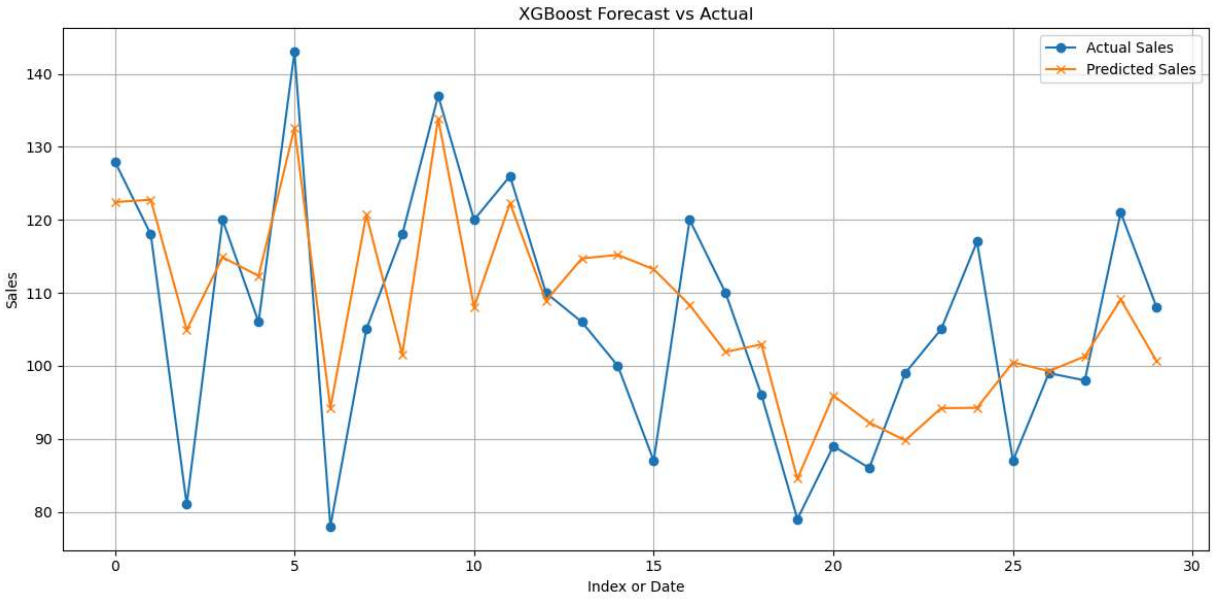
In [23]:
```
xgb_mae = metrics.mean_absolute_error(xgb_y_test, xgb_ypred)
xgb_mape = metrics.mean_absolute_percentage_error(xgb_y_test, xgb_ypred)
```

In [24]:
```
y_tests = np.array(xgb_y_test).ravel()
y_preds = np.array(xgb_ypred).ravel()
```

In [25]:
```
results = pd.DataFrame({
    'Actual': y_tests,
    'Predicted': y_preds
})
```

In [26]:
```
plt.figure(figsize=(12, 6))
plt.plot(results['Actual'], label='Actual Sales', marker='o')
plt.plot(results['Predicted'], label='Predicted Sales', marker='x')
plt.title('XGBoost Forecast vs Actual')
```

```
plt.xlabel('Index or Date')
plt.ylabel('Sales')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [28]:
```
def train_and_evaluate(model, x_train, y_train, x_test, y_test):
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    mae = metrics.mean_absolute_error(y_test, y_pred)
    return mae, y_pred
```

In [29]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_sta
```

In [30]:
```
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2,
```

training lightgbm model

In [31]:
```
from lightgbm import LGBMRegressor
```

In [32]:
```
lgb_model = LGBMRegressor(n_estimators=100, learning_rate=0.09, max_depth=4, random
```

In [33]:
```
lgb_model.fit(x_train, y_train)
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing wa
s 0.001644 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 518
[LightGBM] [Info] Number of data points in the train set: 452, number of used featur
es: 16
[LightGBM] [Info] Start training from score 128.577434
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Out[33]:
```
LGBMRegressor
LGBMRegressor(learning_rate=0.09, max_depth=4, random_state=42)
```

In [34]:
```
lgb_ypred = lgb_model.predict(x_test)
```

lightgbm result evaluation

In [35]:
```
eval_metrics(y_test, lgb_ypred)
```

```
MSE is : 246.53734486482145
MAE is : 12.151812735896403
RMSE is : 15.701507725846632
MAPE is : 0.10683927577272301
R2 is :0.47814343517330815
```

In [36]:
```
lgb_mae = metrics.mean_absolute_error(y_test, lgb_ypred)
lgb_mape = metrics.mean_absolute_percentage_error(y_test, lgb_ypred)
```

training catboost model

In [37]:
```
from catboost import CatBoostRegressor
```

In [38]:
```
cat_model = CatBoostRegressor(n_estimators=100, learning_rate=0.09, random_state=42
```

In [39]:
```
cat_model.fit(x_train, y_train)
```

Out[39]:
```
<catboost.core.CatBoostRegressor at 0x135d77c1550>
```

In [40]:
```
cat_ypred = cat_model.predict(x_test)
```

catboost result evaluation

In [41]:
```
eval_metrics(y_test, cat_ypred)
```

```
MSE is : 274.00175656194835
MAE is : 12.585972053465511
RMSE is : 16.552998416055875
MAPE is : 0.1121311503320281
R2 is :0.4200082932088838
```

In [42]:
```
cat_mae = metrics.mean_absolute_error(y_test, cat_ypred)
cat_mape = metrics.mean_absolute_percentage_error(y_test, cat_ypred)
```

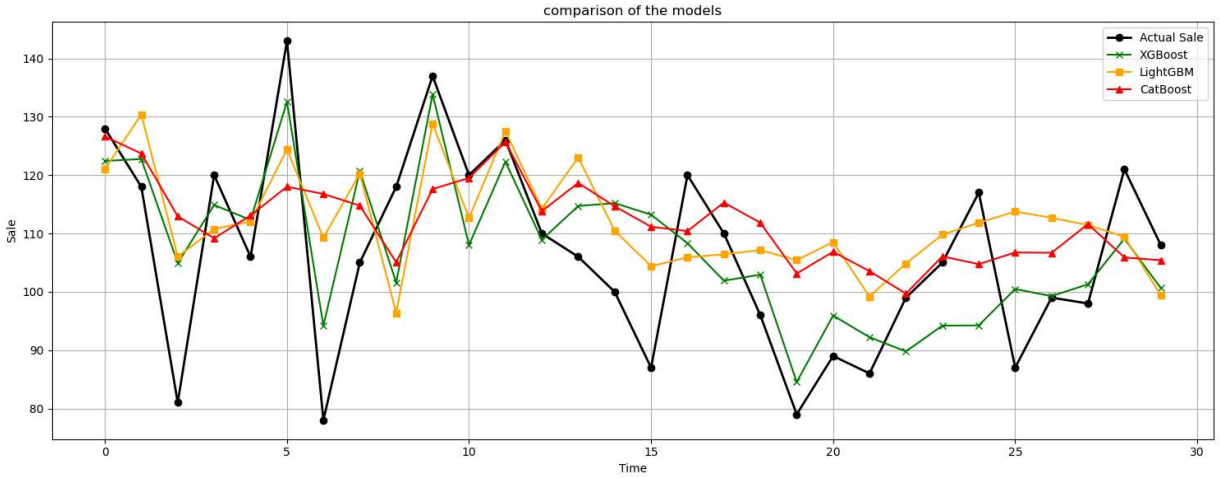
In [43]:
```
results = {}
```

In [44]:
```
results['LightGBM'] = {'MAE': lgb_mae, 'MAPE': lgb_mape}
```

In [45]:
```
results['CatBoost'] = {'MAE': cat_mae, 'MAPE': cat_mape}
```

xgboost, lightgbm and catboost results comparison using graph

In [46]:
```
plt.figure(figsize=(15,6))
plt.plot(y_test.values[-30:], label='Actual Sale', marker='o', linewidth=2, color='
plt.plot(xgb_ypred[:30], label='XGBoost', marker='x', color='green')
plt.plot(lgb_ypred[-30:], label='LightGBM', marker='s', color='orange')
plt.plot(cat_ypred[-30:], label='CatBoost', marker='^', color='red')
plt.title('comparison of the models')
plt.xlabel('Time')
plt.ylabel('Sale')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [48]:
```
best_model_name = min(results, key=lambda k: results[k]['MAE'])
```

In [49]:
```
if best_model_name == 'XGBoost':
    best_model = xgb_model
elif best_model_name == 'LightGBM':
    best_model = lgb_model
else:
    best_model = cat_model
```

In [56]:
```
last_date = data.reset_index()['Date'].max()
```

In [57]:
```
future_dates = pd.date_range(start=last_date+pd.Timedelta(days=1), periods=7)
```

In [58]:
```
future_df = pd.DataFrame({'Date': future_dates})
```

In [59]:
```
future_df['dayofweek'] = future_df['Date'].dt.dayofweek
```

In [60]:
```
future_df['month'] = future_df['Date'].dt.month
```

In [61]:
```
future_df['sin_dow'] = future_df['dayofweek'].apply(lambda x: np.sin(2 * np.pi * x
```

In [62]:
```
future_df['Weekend'] = future_df['dayofweek'].isin([5,6]).astype(int)
```

In [63]:
```
future_df['Day before holiday'] = 0
future_df['Non Holiday date'] = 1
future_df['Day after holiday'] = 0
future_df['Non Holiday'] = 1
future_df['Payment Week'] = future_df['Date'].dt.day.apply(lambda d: 1 if d <= 7 or
```

In [64]:
```
latest_values = data.iloc[-1]
```

In [65]:
```
static_cols = [
     'No. of competitors',
    'Quarterly Average sale', 'Montly Average sale',
    'Qtrly sales std dev', 'Monthly sale std dev'
]
```

In [66]:
```
for col in static_cols:
    future_df[col] = latest_values[col]
```

In [67]:
```
history = data.reset_index()[['Date', 'Sales']].copy()
```

In [68]:
```
history['Date'] = pd.to_datetime(history['Date'])
```

In [69]:
```
preds = []
```

In [ ]:
```

```

In [70]:
```
for i in range(7):
    current_date = future_df.loc[i, 'Date']
    lag_1 = history.iloc[-1]['Sales']
    lag_3 = history.iloc[-3]['Sales']
    lag_5 = history.iloc[-5]['Sales']
    lag_7 = history.iloc[-7]['Sales']
    rolling_mean_7 = history['Sales'][-7:].mean()
    future_df.at[i, 'lag_1'] = lag_1
    future_df.at[i, 'lag_3'] = lag_3
    future_df.at[i, 'lag_5'] = lag_5
    future_df.at[i, 'lag_7'] = lag_7
    future_df.at[i, 'rolling_mean_7'] = rolling_mean_7
    input_row = future_df.loc[i, features].values.reshape(1, -1)
    prediction = xgb_model.predict(input_row)[0]
    preds.append(prediction)
    new_row = pd.DataFrame({'Date': [current_date], 'Sales': [prediction]})
    history = pd.concat([history, new_row], ignore_index=True)
```

In [73]:
```
future_df.columns
```

Out[73]:
```
Index(['Date', 'dayofweek', 'month', 'sin_dow', 'Weekend',
       'Day before holiday', 'Non Holiday date', 'Day after holiday',
       'Non Holiday', 'Payment Week', 'No. of competitors',
       'Quarterly Average sale', 'Montly Average sale', 'Qtrly sales std dev',
       'Monthly sale std dev', 'lag_1', 'lag_3', 'lag_5', 'lag_7',
       'rolling_mean_7'],
      dtype='object')
```

In [ ]:
```
```

In [76]:
```
future_df['predicted_sale']
```

Out[76]:
```
0    105.103706
1    112.457497
2    109.317223
3    104.502113
4    103.224709
5    105.003899
6    108.051193
Name: predicted_sale, dtype: float32
```

In [77]:
```
future_df['predicted_sale'] = preds
```

In [78]:
```
future_df[['Date', 'predicted_sale']]
```

Out[78]:

	Date	predicted_sale
0	2020-03-07	105.103706
1	2020-03-08	112.457497
2	2020-03-09	109.317223
3	2020-03-10	104.502113
4	2020-03-11	103.224709
5	2020-03-12	105.003899
6	2020-03-13	108.051193

In [79]:
```
data['Sales'][-7:]
```

Out[79]:
```
Date
2020-02-28    105
2020-02-29    117
2020-03-01     87
2020-03-02     99
2020-03-04     98
2020-03-05    121
2020-03-06    108
Name: Sales, dtype: int64
```

In [80]:
```
final_forecast = future_df[['Date', 'predicted_sale']].copy()
```

In [81]:
```
final_forecast['predicted_sale'] = final_forecast['predicted_sale'].round(0).astype
```

In [82]:
```
final_forecast
```

Out[82]:

	Date	predicted_sale
0	2020-03-07	105
1	2020-03-08	112
2	2020-03-09	109
3	2020-03-10	105
4	2020-03-11	103
5	2020-03-12	105
6	2020-03-13	108

In [83]:
```
final_forecast = final_forecast.rename(columns={'predicted_sale': 'Forecasted_sale'
```

In [ ]:
```

```

In [84]:
```
import joblib
```

In [85]:
```
joblib.dump(xgb_model, 'models/xgboost_model2.pkl')
```

Out[85]:
```
['models/xgboost_model2.pkl']
```

In [91]:
```
data.reset_index(inplace=True)
```

In [92]:
```
actual_data = data.set_index('Date')['Sales'][-30:]
```

In [93]:
```
actual_data
```

Out[93]:
```
Date
2020-02-03    128
2020-02-04    118
2020-02-05     81
2020-02-06    120
2020-02-07    106
2020-02-08    143
2020-02-09     78
2020-02-10    105
2020-02-11    118
2020-02-12    137
2020-02-13    120
2020-02-14    126
2020-02-15    110
2020-02-16    106
2020-02-17    100
2020-02-18     87
2020-02-19    120
2020-02-20    110
2020-02-21     96
2020-02-22     79
2020-02-23     89
2020-02-26     86
2020-02-27     99
2020-02-28    105
2020-02-29    117
2020-03-01     87
2020-03-02     99
2020-03-04     98
2020-03-05    121
2020-03-06    108
Name: Sales, dtype: int64
```

In [94]:
```
forecasted_data = final_forecast.set_index('Date')['Forecasted_sale']
```

In [95]:
```
final_forecast.columns
```

Out[95]:
```
Index(['Date', 'Forecasted_sale'], dtype='object')
```

In [96]:
```
forecasted_data
```

Out[96]:
```
Date
2020-03-07    105
2020-03-08    112
2020-03-09    109
2020-03-10    105
2020-03-11    103
2020-03-12    105
2020-03-13    108
Name: Forecasted_sale, dtype: int64
```

In [118…
```
forecasted_data
```

Out[118…
```
Date
2020-02-28     94
2020-02-29     93
2020-03-01     85
2020-03-02     88
2020-03-03     92
2020-03-04    102
2020-03-05     91
Name: Forecasted_sale, dtype: int32
```

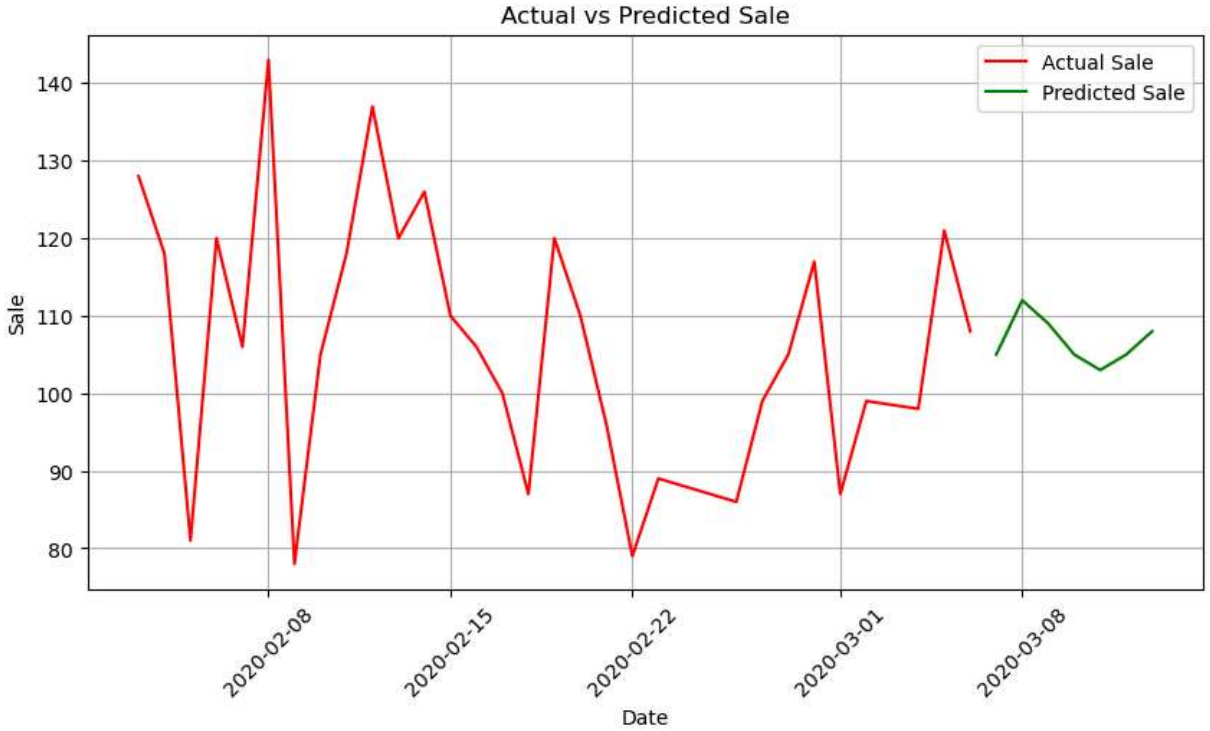
In [97]:
```
combined_data = pd.concat([actual_data, forecasted_data], axis=1)
```

In [98]:
```
combined_data.columns
```

Out[98]:
```
Index(['Sales', 'Forecasted_sale'], dtype='object')
```

In [99]:
```
combined_data.columns = ['Actual Sales', 'Predicted Sale']
```

In [100…
```
plt.figure(figsize=(10,5))
plt.plot(combined_data.index, combined_data['Actual Sales'], label='Actual Sale', c
plt.plot(combined_data.index, combined_data['Predicted Sale'], label='Predicted Sal
plt.xlabel('Date')
plt.ylabel('Sale')
plt.title('Actual vs Predicted Sale')
plt.xticks(rotation=45)
plt.legend()
plt.grid()
plt.show()
```



In [114…
```
plt.figure(figsize=(10,5))
plt.plot(combined_data.index, combined_data['Actual Sales'], label='Actual Sale', c
plt.plot(combined_data.index, combined_data['Predicted Sale'], label='Predicted Sal
```

```
plt.xlabel('Date')
plt.ylabel('Sale')
plt.title('Actual vs Predicted Sale')
plt.xticks(rotation=45)
plt.legend()
plt.grid()
plt.show()
```

In [101…

```
last_date = pd.to_datetime(data.reset_index()['Date'].max())
future_dates = pd.date_range(start=last_date + pd.Timedelta(days=1), periods=30)
future_df = pd.DataFrame({'Date': future_dates})
future_df['dayofweek'] = future_df['Date'].dt.dayofweek
future_df['month'] = future_df['Date'].dt.month
future_df['sin_dow'] = future_df['dayofweek'].apply(lambda x: np.sin(2 * np.pi * x
future_df['Weekend'] = future_df['dayofweek'].isin([5,6]).astype(int)
future_df['Day before holiday'] = 0
future_df['Non Holiday date'] = 1
future_df['Day after holiday'] = 0
future_df['Non Holiday'] = 1
future_df['Payment Week'] = future_df['Date'].dt.day.apply(lambda d: 1 if d <= 7 or

latest_values = data.iloc[-1]
static_cols = [
    'No. of competitors',
    'Quarterly Average sale', 'Montly Average sale',
    'Qtrly sales std dev', 'Monthly sale std dev'
]
for col in static_cols:
    future_df[col] = latest_values[col]

history = data.reset_index()[['Date', 'Sales']].copy()
history['Date'] = pd.to_datetime(history['Date'])
```

```
preds = []
for i in range(30):
    current_date = future_df.loc[i, 'Date']
    # Compute lags from history, which includes previous predictions
    lag_1 = history.iloc[-1]['Sales']
    lag_3 = history.iloc[-3]['Sales']
    lag_5 = history.iloc[-5]['Sales']
    lag_7 = history.iloc[-7]['Sales']
    rolling_mean_7 = history['Sales'][-7:].mean()

    future_df.at[i, 'lag_1'] = lag_1
    future_df.at[i, 'lag_3'] = lag_3
    future_df.at[i, 'lag_5'] = lag_5
    future_df.at[i, 'lag_7'] = lag_7
    future_df.at[i, 'rolling_mean_7'] = rolling_mean_7

    input_row = future_df.loc[i, features].values.reshape(1, -1)
    prediction = xgb_model.predict(input_row)[0]
    preds.append(prediction)

    # Append prediction to history for future lags
    new_row = pd.DataFrame({'Date': [current_date], 'Sales': [prediction]})
    history = pd.concat([history, new_row], ignore_index=True)

future_df['predicted_sale'] = preds
final_forecast = future_df[['Date', 'predicted_sale']].copy()
final_forecast['predicted_sale'] = final_forecast['predicted_sale'].round(0).astype
final_forecast['Day'] = pd.to_datetime(final_forecast['Date']).dt.day_name()
```

In [102…

```
final_forecast
```

Out[102...

	Date	predicted_sale	Day
0	2020-03-07	105	Saturday
1	2020-03-08	112	Sunday
2	2020-03-09	109	Monday
3	2020-03-10	105	Tuesday
4	2020-03-11	103	Wednesday
5	2020-03-12	105	Thursday
6	2020-03-13	108	Friday
7	2020-03-14	100	Saturday
8	2020-03-15	110	Sunday
9	2020-03-16	108	Monday
10	2020-03-17	109	Tuesday
11	2020-03-18	104	Wednesday
12	2020-03-19	103	Thursday
13	2020-03-20	105	Friday
14	2020-03-21	102	Saturday
15	2020-03-22	111	Sunday
16	2020-03-23	112	Monday
17	2020-03-24	113	Tuesday
18	2020-03-25	109	Wednesday
19	2020-03-26	107	Thursday
20	2020-03-27	94	Friday
21	2020-03-28	94	Saturday
22	2020-03-29	105	Sunday
23	2020-03-30	112	Monday
24	2020-03-31	113	Tuesday
25	2020-04-01	118	Wednesday
26	2020-04-02	105	Thursday
27	2020-04-03	107	Friday
28	2020-04-04	101	Saturday
29	2020-04-05	100	Sunday

In [175…

```
final_forecast
```

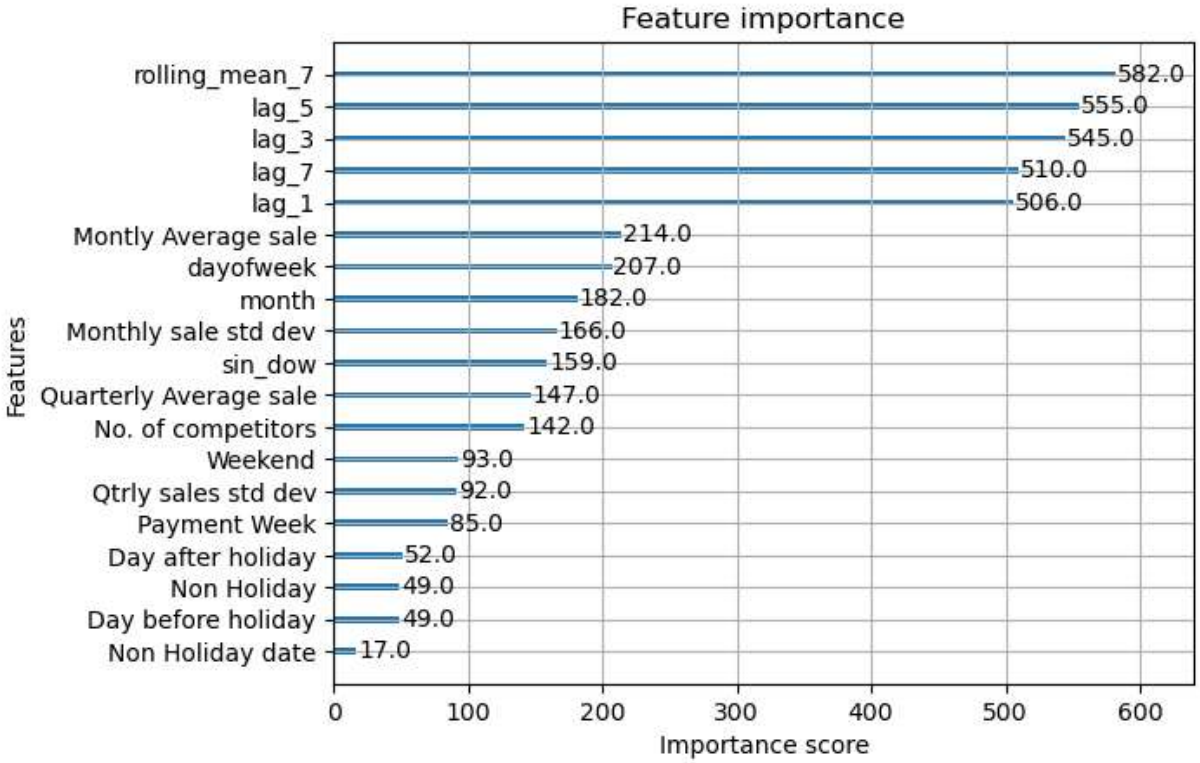
Out[175...

	Date	predicted_sale	Day
0	2020-03-07	105	Saturday
1	2020-03-08	112	Sunday
2	2020-03-09	109	Monday
3	2020-03-10	105	Tuesday
4	2020-03-11	103	Wednesday
5	2020-03-12	105	Thursday
6	2020-03-13	108	Friday
7	2020-03-14	100	Saturday
8	2020-03-15	110	Sunday
9	2020-03-16	108	Monday
10	2020-03-17	109	Tuesday
11	2020-03-18	104	Wednesday
12	2020-03-19	103	Thursday
13	2020-03-20	105	Friday
14	2020-03-21	102	Saturday
15	2020-03-22	111	Sunday
16	2020-03-23	112	Monday
17	2020-03-24	113	Tuesday
18	2020-03-25	109	Wednesday
19	2020-03-26	107	Thursday
20	2020-03-27	94	Friday
21	2020-03-28	94	Saturday
22	2020-03-29	105	Sunday
23	2020-03-30	112	Monday
24	2020-03-31	113	Tuesday
25	2020-04-01	118	Wednesday
26	2020-04-02	105	Thursday
27	2020-04-03	107	Friday
28	2020-04-04	101	Saturday
29	2020-04-05	100	Sunday

In [51]:
```
from xgboost import plot_importance
```

feature importance in xgboost

In [52]:
```
plot_importance(xgb_model)
plt.show()
```



In [ ]:
```
```