In [1]:
```
#한글깨짐
import matplotlib.pyplot as plt
from matplotlib import rc
%matplotlib inline
from matplotlib import font_manager
f_path = "C:/windows/Fonts/malgun.ttf"
font_manager.FontProperties(fname=f_path).get_name()
rc('font', family='Malgun Gothic')
```

In [10]:
```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
from keras.models import load_model
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
from scipy import stats
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

In [63]:
```
# 데이터 불러오기
df = pd.read_csv('total_data.csv')
print(df.info())
# 날짜 열을 datetime 형식으로 변환
df['be_date'] = pd.to_datetime(df['be_date'])

# 날짜를 인덱스로 설정
df.set_index('be_date', inplace=True)

# 'be_total_energy'가 같은 시간대 데이터를 합산하고 10분 간격으로 리샘플링
df = df.groupby(pd.Grouper(freq='1T')).agg({
    'be_ac_energy': 'sum',
    'be_light_energy': 'sum',
    'be_plug_energy': 'sum',
    'be_total_energy': 'sum',
    'be_floor': 'sum'
})

print(df.info())
```

```
 0   be_date          object
 1   be_ac_energy     float64
 2   be_light_energy  float64
 3   be_plug_energy   float64
 4   be_total_energy  float64
 5   be_floor         int64
dtypes: float64(4), int64(1), object(1)
memory usage: 83.4+ MB
None
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 264960 entries, 2018-07-01 00:00:00 to 2018-12-31 23:59:00
Freq: T
Data columns (total 5 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   be_ac_energy     264960 non-null  float64
 1   be_light_energy  264960 non-null  float64
 2   be_plug_energy   264960 non-null  float64
 3   be_total_energy  264960 non-null  float64
 4   be_floor         264960 non-null  int64
```

In [64]:
```
# 날짜를 기준으로 필터링 (9월 1일 00시 00분부터 2주간의 데이터)

start_date = '2018-09-01 00:00:00'
end_date = '2018-09-14 23:59:59'
df = df[start_date:end_date]
```

In [65]:
```
# be_date를 기준으로 그룹화하고 합계 계산
#df = df.groupby(pd.Grouper(freq='D')).sum()

# Z-점수를 이용한 이상치 제거
z_scores = stats.zscore(df[['be_ac_energy', 'be_light_energy', 'be_plug_energy']]
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3).all(axis=1)
df_no_outliers = df[filtered_entries]

# 이상치 제거 후 데이터를 확인해보세요
print(df_no_outliers.head())
```

```
                     be_ac_energy  be_light_energy  be_plug_energy  ₩
be_date
2018-09-01 00:00:00         53.18            60.05           41.34
2018-09-01 00:01:00         51.99            60.10           41.25
2018-09-01 00:02:00         53.10            60.21           41.35
2018-09-01 00:03:00         53.12            60.18           41.26
2018-09-01 00:04:00         53.19            60.63           41.31

                     be_total_energy  be_floor
be_date
2018-09-01 00:00:00           154.57        28
2018-09-01 00:01:00           153.34        28
2018-09-01 00:02:00           154.66        28
2018-09-01 00:03:00           154.56        28
2018-09-01 00:04:00           155.13        28
```

In [66]:
```
# 데이터 스케일링
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df_no_outliers['be_total_energy'].values.resha
```

In [67]:
```
# # 훈련 및 테스트 데이터 분할
# train_data = result[:row_cnt, :]
# x_train = train_data[:, :seq_len]
# y_train = train_data[:, seq_len:]
# x_train_reshape = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

# 데이터 슬라이싱을 통해 학습용 데이터와 테스트용 데이터 분리
train_data = scaled_data[:-288]  # 마지막 2일(288개 데이터)을 제외한 데이터를 학
test_data = scaled_data[-288:]  # 마지막 2일(288개 데이터)을 테스트용으로 사용
```

In [72]:
```
# 시퀀스 데이터 생성 (10분 간격으로 2주치 데이터를 학습)
seq_len = 12  # 시퀀스 길이 (하루에 144개의 10분 간격 데이터가 있음)
future_period = 30  # 미래 예측 기간 (하루, 144개의 10분 간격 데이터를 예측)
result = []
for idx in range(len(scaled_data) - seq_len - future_period):
    seq_x = scaled_data[idx: idx + seq_len]
    seq_y = scaled_data[idx + seq_len: idx + seq_len + future_period]
    result.append(np.append(seq_x, seq_y))

result = np.array(result)
#row_cnt = int(round(result.shape[0] * 0.8))
```

In [73]:
```
# 훈련 및 테스트 데이터 분할
row_cnt = int(round(result.shape[0] * 0.8))
train_data = result[:row_cnt, :]
x_train = train_data[:, :seq_len]
y_train = train_data[:, seq_len:]
x_train_reshape = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
test_data = result[row_cnt:, :]
x_test = test_data[:, :seq_len]
y_test = test_data[:, seq_len:]
x_test_reshape = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

In [74]:
```
# LSTM 모델 정의
model = Sequential()
model.add(LSTM(64, return_sequences=True, input_shape=(seq_len, 1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(future_period, activation='linear'))
model.compile(loss='mse', optimizer='adam')

# 모델 훈련
model.fit(x_train_reshape, y_train, validation_data=(x_test_reshape, y_test), bat
# 예측
pred = model.predict(x_test_reshape)

# 예측값 역 스케일링
y_test_original = scaler.inverse_transform(y_test)
pred_original = scaler.inverse_transform(pred)
```

```
Epoch 1/20
63/63 [==============================] - 9s 105ms/step - loss: 0.0331 - val_los
s: 0.0160
Epoch 2/20
63/63 [==============================] - 6s 95ms/step - loss: 0.0104 - val_los
s: 0.0158
Epoch 3/20
63/63 [==============================] - 6s 95ms/step - loss: 0.0102 - val_los
s: 0.0152
Epoch 4/20
63/63 [==============================] - 6s 97ms/step - loss: 0.0098 - val_los
s: 0.0148
Epoch 5/20
63/63 [==============================] - 6s 98ms/step - loss: 0.0095 - val_los
s: 0.0143
Epoch 6/20
63/63 [==============================] - 6s 98ms/step - loss: 0.0093 - val_los
s: 0.0138
Epoch 7/20
63/63 [==============================] - 6s 99ms/step - loss: 0.0090 - val_los
s: 0.0136
Epoch 8/20
63/63 [==============================] - 6s 98ms/step - loss: 0.0089 - val_los
s: 0.0139
Epoch 9/20
63/63 [==============================] - 6s 98ms/step - loss: 0.0088 - val_los
s: 0.0135
Epoch 10/20
63/63 [==============================] - 6s 99ms/step - loss: 0.0087 - val_los
s: 0.0134
Epoch 11/20
63/63 [==============================] - 6s 98ms/step - loss: 0.0085 - val_los
s: 0.0131
Epoch 12/20
63/63 [==============================] - 6s 99ms/step - loss: 0.0084 - val_los
s: 0.0128
Epoch 13/20
63/63 [==============================] - 6s 98ms/step - loss: 0.0084 - val_los
s: 0.0127
Epoch 14/20
63/63 [==============================] - 6s 99ms/step - loss: 0.0082 - val_los
s: 0.0129
Epoch 15/20
63/63 [==============================] - 6s 98ms/step - loss: 0.0081 - val_los
s: 0.0125
Epoch 16/20
63/63 [==============================] - 6s 98ms/step - loss: 0.0081 - val_los
s: 0.0124
Epoch 17/20
63/63 [==============================] - 6s 98ms/step - loss: 0.0081 - val_los
s: 0.0122
Epoch 18/20
63/63 [==============================] - 6s 98ms/step - loss: 0.0080 - val_los
s: 0.0121
Epoch 19/20
63/63 [==============================] - 6s 98ms/step - loss: 0.0080 - val_los
s: 0.0123
Epoch 20/20
63/63 [==============================] - 6s 99ms/step - loss: 0.0079 - val_los
```

```
s: 0.0121
126/126 [==============================] - 1s 5ms/step
```

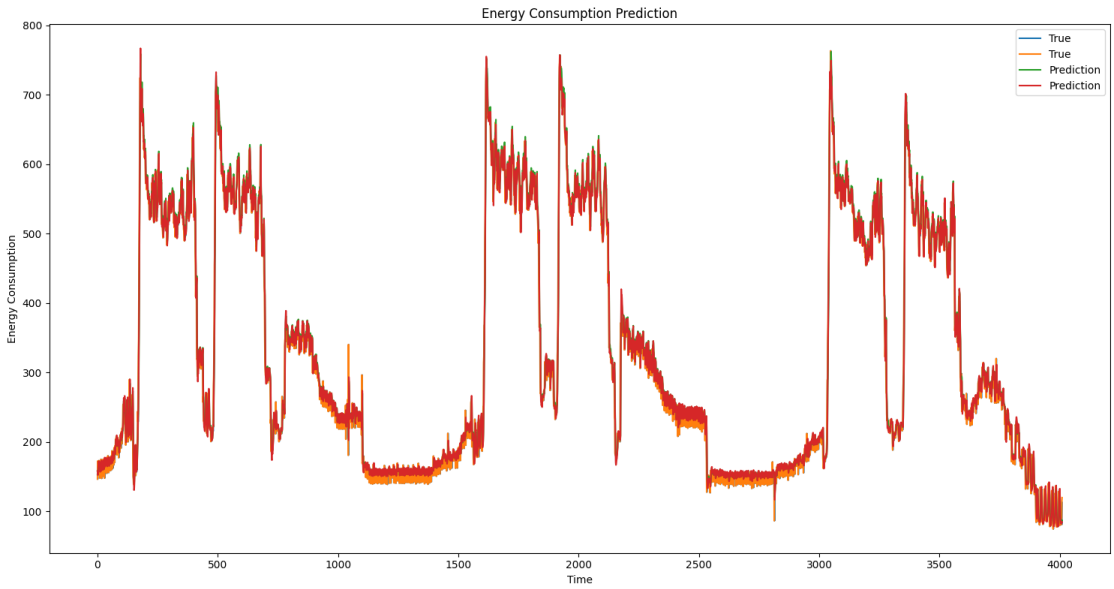
In [71]:
```
# 평가
# MSE 계산
mse = mean_squared_error(y_test_original, pred_original)
# MAE 계산
mae = mean_absolute_error(y_test_original, pred_original)

# RMSE 계산
rmse = np.sqrt(mean_squared_error(y_test_original, pred_original))
#r2
r2 = r2_score(y_test_original, pred_original)

print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
print(f"R-squared (R2) Score: {r2:.4f}")

# 시각화
plt.figure(figsize=(15, 8))
plt.plot(y_test_original, label='True')
plt.plot(pred_original, label='Prediction')
plt.title("Energy Consumption Prediction")
plt.xlabel("Time")
plt.ylabel("Energy Consumption")
plt.legend()
plt.tight_layout()
plt.show()
```

```
Mean Squared Error (MSE): 351.6322
Mean Absolute Error (MAE): 12.4040
Root Mean Squared Error (RMSE): 18.7519
R-squared (R2) Score: 0.9881
```



In [ ]: