```
In [1]: import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from scipy.stats import f_oneway
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import matplotlib.pyplot as plt
```

```
In [2]: df = pd.read_csv("ob_extended.csv")
df.head()
```

```
C:\Users\local_CRauchman\Temp\9\ipykernel_18364\397843626.py:1: DtypeWarning: Columns (2,3) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("ob_extended.csv")
```

Out[2]:

	PatientEncounterCSNID	EpisodeID	CheckinTime	CheckoutTime	HospitalAdmissionTime	HospitalDischargeTime	LengthOfStay	IsInpatient	IsObservation	DeliveryDate	...	IsPrimaryDeliveryUnit	TotalLaborMinutes	GravidaCount	ParaCount	PretermCount	IsWeekend	IsHoliday	MonthName	Quarter
0	2103031973	5.542547e+18	NaN	NaN	2022-06-07 16:44:00.000	2022-06-11 13:48:00.000	3.836806	1	0	2022-06-08 00:00:00.000	...	1	NaN	NaN	NaN	NaN	0	0	June	
1	2103031973	5.542547e+18	NaN	NaN	2022-06-07 16:44:00.000	2022-06-11 13:48:00.000	3.836806	1	0	2022-06-08 00:00:00.000	...	1	NaN	NaN	NaN	NaN	0	0	June	
2	2103031973	5.542547e+18	NaN	NaN	2022-06-07 16:44:00.000	2022-06-11 13:48:00.000	3.836806	1	0	2022-06-08 00:00:00.000	...	1	NaN	NaN	NaN	NaN	0	0	June	
3	2103031973	5.542547e+18	NaN	NaN	2022-06-07 16:44:00.000	2022-06-11 13:48:00.000	3.836806	1	0	2022-06-08 00:00:00.000	...	1	NaN	NaN	NaN	NaN	0	0	June	
4	2103031973	5.542547e+18	NaN	NaN	2022-06-07 16:44:00.000	2022-06-11 13:48:00.000	3.836806	1	0	2022-06-08 00:00:00.000	...	1	NaN	NaN	NaN	NaN	0	0	June	

5 rows × 30 columns

dedup delivery encounters (adi's code)

```
In [3]: def deduplicate_encounters(df):
    print(f"Rows before deduplication: {len(df)}")
    df_deduped = df.drop_duplicates(subset=['PatientEncounterCSNID'], keep='first')
    print(f"Rows after deduplication: {len(df_deduped)}")
    #assert df_deduped['PatientEncounterCSNID'].value_counts().max() == 1, "Duplicates still exist!"
    return df_deduped
df_clean = deduplicate_encounters(df)
df_clean = df_clean.reset_index(drop=True)
```

```
Rows before deduplication: 177438
Rows after deduplication: 7386
```

define capacity as the median patients/day in a given week

limit dataset to 1/3/2022 - 9/1/2024

In [4]:

```
df_clean['AdmissionDateTime'] = pd.to_datetime(df['HospitalAdmissionTime'])
df_clean = df_clean.dropna(subset=['AdmissionDateTime', 'LengthOfStay'])
df_clean= df_clean[(df_clean['AdmissionDateTime']<datetime(2024,9,1)) &(df_clean['AdmissionDateTime']>=datetime(2022,1,3))]
# Expand data to represent daily occupancy based on admission day and LOS
expanded_data = []

for _, row in df_clean.iterrows():
    los_int = int(np.round(row['LengthOfStay'],0))
    for day in range(los_int):
        occupancy_date = row['AdmissionDateTime'] + pd.Timedelta(days=day)
        expanded_data.append(occupancy_date)

expanded_data = [i.date() for i in expanded_data]
occupancy_df = pd.DataFrame({'date': expanded_data})

weekly_capacity_i = occupancy_df.groupby('date').size()
weekly_capacity_i = weekly_capacity_i.reset_index(name='weekly_capacity')
weekly_capacity_i['date'] = [datetime(d.year, d.month, d.day) for d in weekly_capacity_i['date']]
weekly_capacity_i['week'] = weekly_capacity_i['date'].dt.to_period('W').apply(lambda r: r.start_time)
weekly_capacity = weekly_capacity_i.groupby('week').median()
```

```
C:\Users\local_CRauchman\Temp\9\ipykernel_18364\1043697256.py:20: FutureWarning: The default value of numeric_only in DataFrameGroupBy.median is deprecated. In a future version, numeric_only will default to False. Either specify numeric_only or select only c
olumns which should be valid for the function.
  weekly_capacity = weekly_capacity_i.groupby('week').median()
```

In [5]:

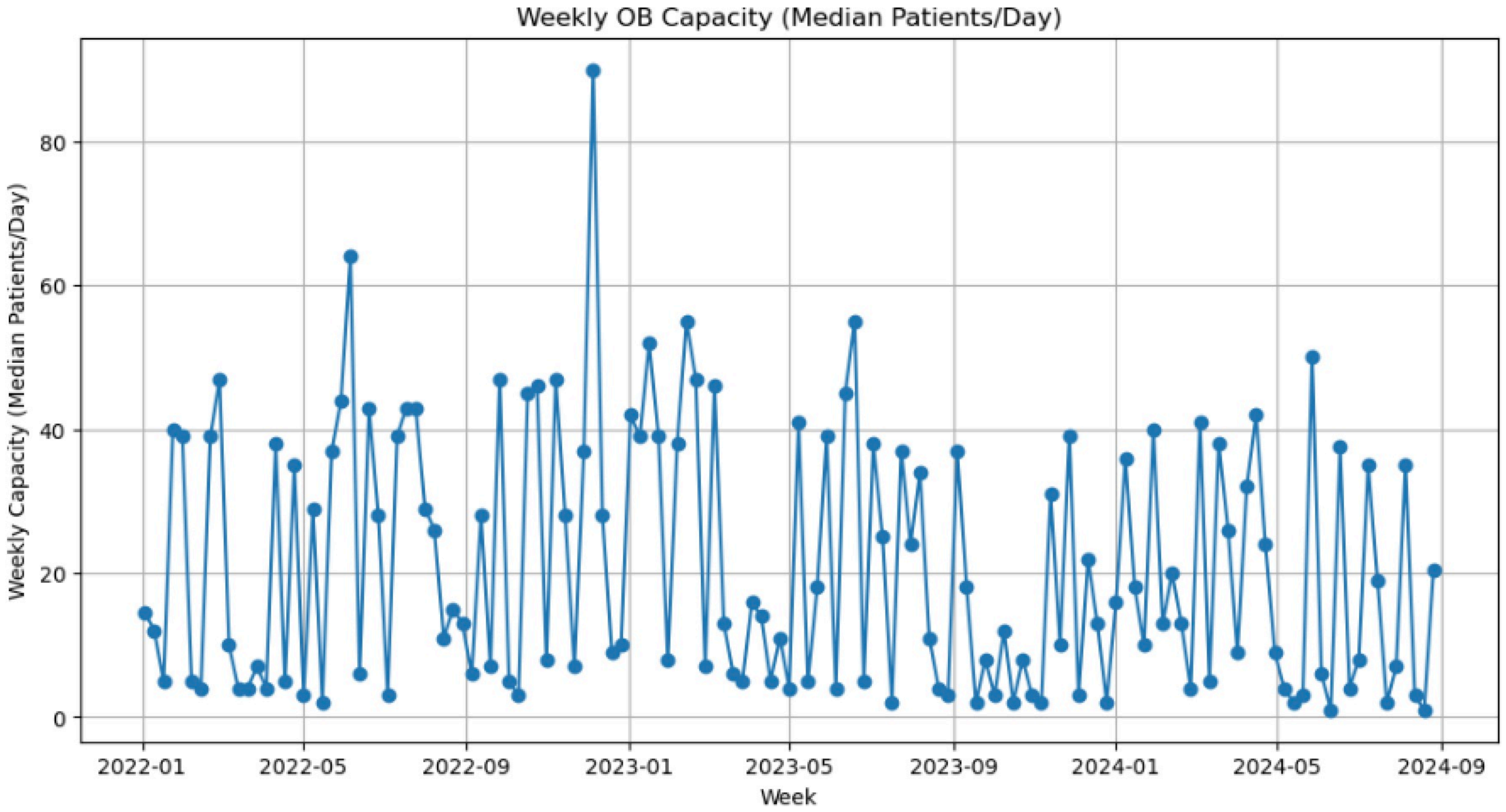
```
# Create a DataFrame of daily occupancy
# #only look at 2022 - 2023
# occupancy_df = pd.DataFrame({'date': expanded_data})
# occupancy_df= occupancy_df[(occupancy_df['date']<datetime(2024,1,1)) &(occupancy_df['date']>=datetime(2022,1,3))]

# # Group by week to calculate weekly capacity
#weekly_capacity_i['week'] = weekly_capacity_i['date'].dt.to_period('W').apply(lambda r: r.start_time)
# weekly_capacity = weekly_capacity_i.groupby('week').agg(np.median)
# weekly_capacity

#Convert to DataFrame for better presentation
weekly_capacity = weekly_capacity.reset_index()
print(weekly_capacity.head())

plt.figure(figsize=(12, 6))
plt.plot(weekly_capacity['week'], weekly_capacity['weekly_capacity'], marker='o')
plt.title("Weekly OB Capacity (Median Patients/Day)")
plt.xlabel("Week")
plt.ylabel("Weekly Capacity (Median Patients/Day)")
plt.grid()
plt.show()
```

```
        week  weekly_capacity
0 2022-01-03             14.5
1 2022-01-10             12.0
2 2022-01-17              5.0
3 2022-01-24             40.0
4 2022-01-31             39.0
```



In [6]:
```
weekly_capacity['WeekNum'] = [i.isocalendar()[1] for i in weekly_capacity['week']]
weekly_capacity['Year'] = [i.isocalendar()[0] for i in weekly_capacity['week']]

weekly_capacity.head()
```

Out[6]:

	week	weekly_capacity	WeekNum	Year
0	2022-01-03	14.5	1	2022
1	2022-01-10	12.0	2	2022
2	2022-01-17	5.0	3	2022
3	2022-01-24	40.0	4	2022
4	2022-01-31	39.0	5	2022

used an LSTM with a lookback of 2 weeks, relu activation,

In [7]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense


train = weekly_capacity[weekly_capacity['week']<datetime(2024,1,1)]
test = weekly_capacity[(weekly_capacity['week']>=datetime(2024,1,1)) & (weekly_capacity['week']<datetime(2024,9,1))]


# Assume train and test are loaded as DataFrames
# Columns: 'Week Date' (datetime) and 'Capacity'

# Convert 'Week Date' to index if not already done
train.set_index('week', inplace=True)
test.set_index('week', inplace=True)

# Scale the capacity values
scaler = MinMaxScaler(feature_range=(0, 1))
train_scaled = scaler.fit_transform(train[['weekly_capacity']])
test_scaled = scaler.transform(test[['weekly_capacity']])

# Create a function to build input-output pairs (sliding window)
def create_sequences(data, look_back):
    X, y = [], []
    for i in range(len(data) - look_back):
        X.append(data[i:i + look_back, 0])
        y.append(data[i + look_back, 0])
    return np.array(X), np.array(y)

# Use a look-back of 2 weeks (optimal - tried 1 through 5)
look_back = 2
X_train, y_train = create_sequences(train_scaled, look_back)
X_test, y_test = create_sequences(test_scaled, look_back)

# Reshape for LSTM: (samples, time steps, features)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

In [8]:
```
from tensorflow.keras.optimizers import Adam

# Define the LSTM model

#relu activation

model = Sequential([
    LSTM(50, activation='relu', input_shape=(look_back, 1), return_sequences=True),
    LSTM(50, activation='relu'),
    Dense(1)
])

# Compile the modelw adam optimizer, lr = 0.005 (optimized from range 0.001 - 0.01)
model.compile(optimizer=Adam(learning_rate=0.005), loss='mae')

# training
#batch size optimized from range (8 through 64)
history = model.fit(X_train, y_train, epochs=50, batch_size=8, validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/50
13/13 [==============================] - 3s 39ms/step - loss: 0.1912 - val_loss: 0.1505
Epoch 2/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1909 - val_loss: 0.1579
Epoch 3/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1850 - val_loss: 0.1437
Epoch 4/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1828 - val_loss: 0.1443
Epoch 5/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1829 - val_loss: 0.1455
Epoch 6/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1825 - val_loss: 0.1455
Epoch 7/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1834 - val_loss: 0.1435
Epoch 8/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1827 - val_loss: 0.1442
Epoch 9/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1822 - val_loss: 0.1458
Epoch 10/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1825 - val_loss: 0.1466
Epoch 11/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1827 - val_loss: 0.1449
Epoch 12/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1825 - val_loss: 0.1445
Epoch 13/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1830 - val_loss: 0.1448
Epoch 14/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1827 - val_loss: 0.1440
Epoch 15/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1826 - val_loss: 0.1458
Epoch 16/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1828 - val_loss: 0.1448
Epoch 17/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1845 - val_loss: 0.1479
Epoch 18/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1826 - val_loss: 0.1440
Epoch 19/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1827 - val_loss: 0.1459
Epoch 20/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1823 - val_loss: 0.1441
Epoch 21/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1817 - val_loss: 0.1467
Epoch 22/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1810 - val_loss: 0.1451
Epoch 23/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1821 - val_loss: 0.1470
Epoch 24/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1794 - val_loss: 0.1461
Epoch 25/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1797 - val_loss: 0.1456
Epoch 26/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1786 - val_loss: 0.1525
Epoch 27/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1827 - val_loss: 0.1472
Epoch 28/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1770 - val_loss: 0.1499
Epoch 29/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1759 - val_loss: 0.1505
Epoch 30/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1742 - val_loss: 0.1507
Epoch 31/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1754 - val_loss: 0.1553
Epoch 32/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1734 - val_loss: 0.1519
```

```
Epoch 33/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1704 - val_loss: 0.1576
Epoch 34/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1692 - val_loss: 0.1578
Epoch 35/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1694 - val_loss: 0.1540
Epoch 36/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1682 - val_loss: 0.1578
Epoch 37/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1667 - val_loss: 0.1588
Epoch 38/50
13/13 [==============================] - 0s 7ms/step - loss: 0.1704 - val_loss: 0.1591
Epoch 39/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1641 - val_loss: 0.1649
Epoch 40/50
13/13 [==============================] - 0s 7ms/step - loss: 0.1654 - val_loss: 0.1594
Epoch 41/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1604 - val_loss: 0.1617
Epoch 42/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1644 - val_loss: 0.1600
Epoch 43/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1591 - val_loss: 0.1725
Epoch 44/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1586 - val_loss: 0.1644
Epoch 45/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1713 - val_loss: 0.1626
Epoch 46/50
13/13 [==============================] - 0s 5ms/step - loss: 0.1563 - val_loss: 0.1645
Epoch 47/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1588 - val_loss: 0.1637
Epoch 48/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1573 - val_loss: 0.1668
Epoch 49/50
13/13 [==============================] - 0s 7ms/step - loss: 0.1595 - val_loss: 0.1697
Epoch 50/50
13/13 [==============================] - 0s 6ms/step - loss: 0.1535 - val_loss: 0.1672
```

In [9]:

```python
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error

#predict on test data
y_pred = model.predict(X_test)

#inverse transform predictions and true values to original scale
y_pred_rescaled = scaler.inverse_transform(y_pred)
y_test_rescaled = scaler.inverse_transform(y_test.reshape(-1, 1))

#performance
rmse = np.sqrt(mean_squared_error(y_test_rescaled, y_pred_rescaled))
mae = mean_absolute_error(y_test_rescaled, y_pred_rescaled)

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"Mean Absolute Error (MAE): {mae:.2f}")

plt.figure(figsize=(14, 7))
plt.plot(test.index[look_back:], y_test_rescaled, label="Actual Capacity", marker="o", color='blue')
plt.plot(test.index[look_back:], y_pred_rescaled, label="Predicted Capacity", linestyle="--", marker="x", color='orange')
plt.title("LSTM: Actual vs Predicted Weekly Capacity")
plt.xlabel("Week Date")
plt.ylabel("Capacity")
plt.legend()
plt.grid()
plt.show()
```

```
2/2 [==============================] - 0s 3ms/step
Root Mean Squared Error (RMSE): 17.69
Mean Absolute Error (MAE): 14.71
```