Linear Regression Analysis and Prediction for IoT

This notebook holds the Assignment 3.1 for Module 3 in AAI 530, Data Analytics and the Internet of Things. In this assignment, you will use linear regression to make predictions for simulated "streaming" data. The work that you do in this assignment will build on the linear regression predictions that you saw in your text book and in this week's lab session. Be sure to answer the analysis questions thoroughly, as this is a large part of the assignment for this week.

General Assignment Instructions

These instructions are included in every assignment, to remind you of the coding standards for the class. Feel free to delete this cell after reading it.

One sign of mature code is conforming to a style guide. We recommend the Google Python Style Guide. If you use a different style guide, please include a cell with a link.

Your code should be relatively easy-to-read, sensibly commented, and clean. Writing code is a messy process, so please be sure to edit your final submission. Remove any cells that are not needed or parts of cells that contain unnecessary code. Remove inessential `import` statements and make sure that all such statements are moved into the designated cell.

When you save your notebook as a pdf, make sure that all cell output is visible (even error messages) as this will aid your instructor in grading your work.

Make use of non-code cells for written commentary. These cells should be grammatical and clearly written. In some of these cells you will have questions to answer. The questions will be marked by a "Q:" and will have a corresponding "A:" spot for you. *Make sure to answer every question marked with a `Q:` for full credit.*

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

#suppress scientific notation in pandas
pd.set_option('display.float_format', lambda x: '%.5f' % x)
```

```
#use this cell to import additional libraries or define helper functions
import warnings

#Instead of using an external module, utilize the built-in warnings module
#to ignore warnings
warnings.filterwarnings("ignore")
```

```
from google.colab import drive
drive.mount('/content/drive')


!cp "/content/drive/My Drive/Colab Notebooks/household_power_consumption.txt" .
```

```
Mounted at /content/drive
```

Load and prepare your data

We'll be using the cleaned household electric consumption dataset from Module 2 in this assignment. I recommend saving your dataset by running df.to_csv("filename") at the end of the last assignment so that you don't have to re-do your cleaning steps. If you are not confident in your own cleaning steps, you may ask your instructor for a cleaned version of the data. You will not be graded on the cleaning steps in this assignment, but some functions may not work if you use the raw data.

We need to turn our datetime column into a numeric value to be used as a variable in our linear regression. In the lab session, we created a new column of minutes and just incremented the value by 10 since we knew that the readings occurred every 10 minutes. In this dataset, we have readings every minute, but we might have some missing rows depending on how you cleaned your data. So instead we will convert our datetime column to something called unix/epoch time, which is the number of seconds since midnight on 1/1/1970.

TODO: load your data and convert the datetime column into epoch/unix time

```
#Load your data into a pandas dataframe here

#Load your data into a pandas dataframe here
!cp "/content/drive/My Drive/Colab Notebooks/household_power_consumption.txt" .
import pandas as pd
df2 = pd.read_csv('household_power_consumption.txt', sep=';', na_values=['?'],
                  parse_dates={'datetime': ['Date', 'Time']}, infer_datetime_for
                  low_memory=False, index_col='datetime')
df = df2
```

```
df.head()
```

	Global_active_power	Global_reactive_power	Voltage	Global_inter
datetime				
2006-12-16 17:24:00	4.21600	0.41800	234.84000	18.4
2006-12-16 17:25:00	5.36000	0.43600	233.63000	23.(
2006-12-16 17:26:00	5.37400	0.49800	233.29000	23.(
2006-12-16 17:27:00	5.38800	0.50200	233.74000	23.(

```
from datetime import datetime
from datetime import datetime
from datetime import datetime
from datetime import datetime
#convert datetime to epoch/unix time

df['unix'] = pd.to_numeric((df.index.astype('int64') / 10**9))
print(df.index)

df['time_diff'] = 10
df['time'] = df.time_diff.cumsum()

df.head()
```

```
DatetimeIndex(['2006-12-16 17:24:00', '2006-12-16 17:25:00',
               '2006-12-16 17:26:00', '2006-12-16 17:27:00',
               '2006-12-16 17:28:00', '2006-12-16 17:29:00',
               '2006-12-16 17:30:00', '2006-12-16 17:31:00',
               '2006-12-16 17:32:00', '2006-12-16 17:33:00',
               ...
               '2010-11-26 20:53:00', '2010-11-26 20:54:00',
               '2010-11-26 20:55:00', '2010-11-26 20:56:00',
               '2010-11-26 20:57:00', '2010-11-26 20:58:00',
               '2010-11-26 20:59:00', '2010-11-26 21:00:00',
               '2010-11-26 21:01:00', '2010-11-26 21:02:00'],
              dtype='datetime64[ns]', name='datetime', length=2075259, freq
```

	Global_active_power	Global_reactive_power	Voltage	Global_inter
datetime				
2006-12-16 17:24:00	4.21600	0.41800	234.84000	18.4
2006-12-16 17:25:00	5.36000	0.43600	233.63000	23.(
2006-12-16 17:26:00	5.37400	0.49800	233.29000	23.(
2006-12-16 17:27:00	5.38800	0.50200	233.74000	23.(
2006-12-16 17:28:00	3.66600	0.52800	235.68000	15.8

```
df.tail()
```

	Global_active_power	Global_reactive_power	Voltage	Global_inter
datetime				
2010-11-26 20:58:00	0.94600	0.00000	240.43000	4.
2010-11-26 20:59:00	0.94400	0.00000	240.00000	4.
2010-11-26 21:00:00	0.93800	0.00000	239.82000	3.
2010-11-26 21:01:00	0.93400	0.00000	239.70000	3.
2010-11-26 21:02:00	0.93200	0.00000	239.55000	3.

Predicting Global Active Power

We will follow the code from the Chapter 9 in our textbook and the recorded lab session from this week to predict the Global Active Power (GAP) with linear regression.

First we will create our x (time) and y (GAP) training variables, and then define our model parameters.

Q: What is ph? What is mu?

A: A: The prediction horizon (ph) defines how far into the future the model makes predictions. It represents the number of time steps ahead for which the model forecasts values.

📌 In this Case (ph = 100) If time is in days, it means predicting 100 days ahead. If time is in hours, it means predicting 100 hours ahead.

mu = Forgetting Factor

This controls how much older data influences the model. Older data points are exponentially downweighted

TODO: Set the ph to be 5 minutes--consider the units that our time column is measured in.

```
#################################################################

ph = 5 * 60
ph_index = round(ph/10)
mu = 0.9

n_s = 5000

# Arrays to hold predicted values
tp_pred = np.zeros(n_s-1)
yp_pred = np.zeros(n_s-1)
```

```
# The weight of the first data point on the last prediction is determined by th

# Number of time steps between the first and last data points (5000th predictic
time_steps = 4999

# Weight of the first data point on the last prediction
weight = mu ** time_steps

print(f"The weight of the first data point on the 5000th prediction is: {weight
```

```
The weight of the first data point on the 5000th prediction is: 1.813e-229
```

Q: With mu = 0.9, how much weight will our first data point have on the last (5000th) prediction in our limited dataset?

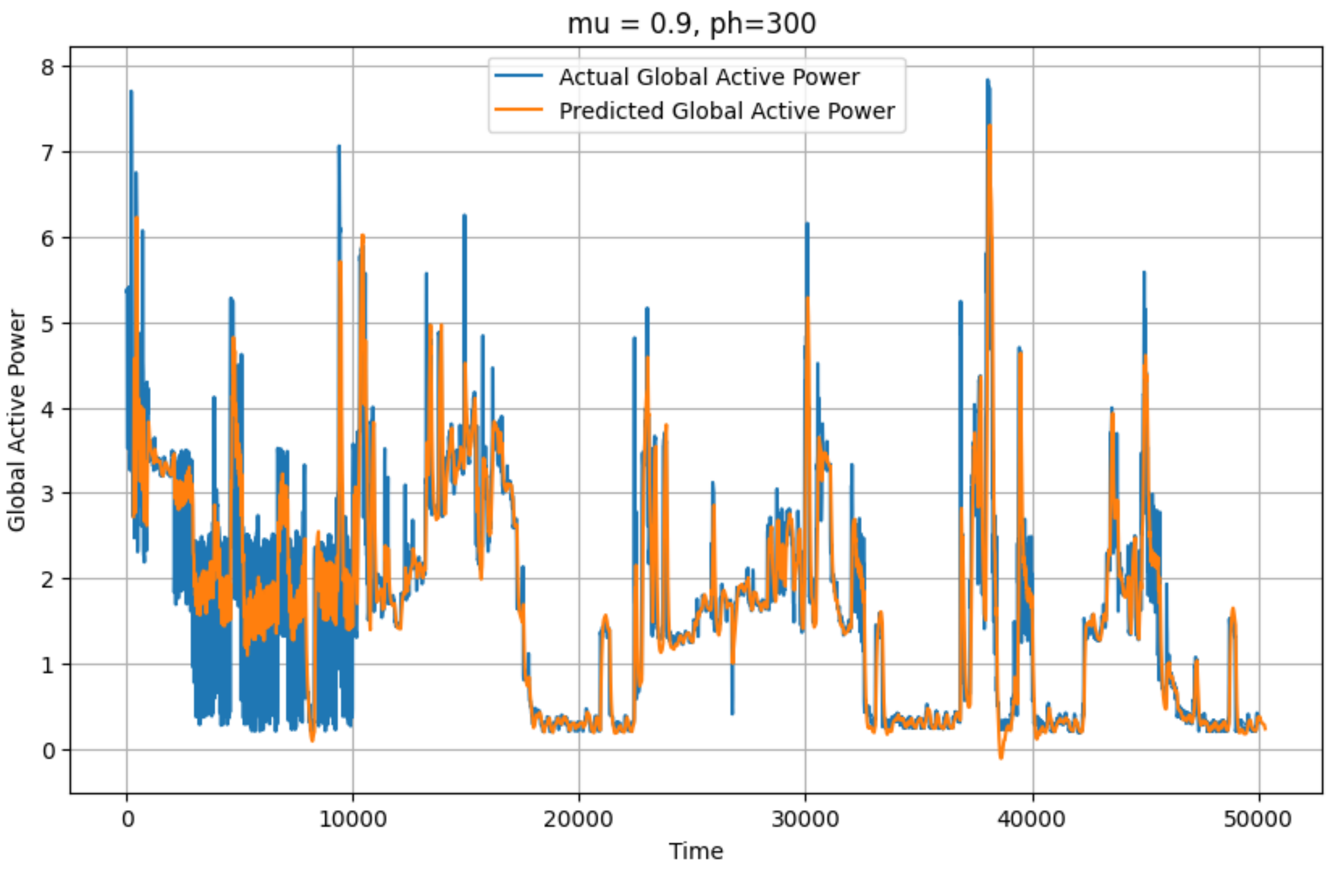
A: The weight of the first data point on the 5000th prediction is: 1.813e-229T

TODO: Following the code from Chapter 10 and the lab session, use linear regression to predict a rolling GAP for our dataset. Store these predictions in the tp_pred and yp_pred lists created above for visualization.

```
# At every iteration of the for loop a new sample is acquired.
ts = pd.DataFrame(df.time)
ys = pd.DataFrame(df.Global_active_power)

for i in range(2, n_s+1):# start out with 2 leading datapoints
    ts_tmp = ts[0:i]
########################################################################
#INSTEAD OF USING THE CURRENT VALUE AS OUR PREDICTED VALUE IN TRAINING
#WE USE THE VALUE AT THE PREDICTIVE HORIZON INDEX
########################################################################
    ys_tmp = ys[ph_index-1:ph_index-1+i]
    ns = len(ys_tmp)

    # The mu**k assigns the weight to the previous samples.
    weights = np.ones(ns)*mu
    for k in range(ns):
        weights[k] = weights[k]**k #More distant examples are downweighted expo
    weights = np.flip(weights, 0)


    # MODEL
    # Linear Regression.
    lm_tmp = LinearRegression()
    model_tmp = lm_tmp.fit(ts_tmp, ys_tmp, sample_weight=weights)# times are x,
    # Coefficients of the linear model, y = mx + q
    m_tmp = model_tmp.coef_
    q_tmp = model_tmp.intercept_

    # PREDICTION
    tp = ts.iloc[i-1,0] #### don't add ph for prediction
    yp = m_tmp*tp + q_tmp #mx + b

    tp_pred[i-2] = tp + ph ### add ph to timestamp so that chart lines up with
    yp_pred[i-2] = yp
```

Now let's visualize the results from our model.

```
import matplotlib.pyplot as plt

# Plot the actual and predicted values
plt.figure(figsize=(10, 6))
plt.plot(ts.iloc[1:n_s,0], ys.iloc[1:n_s,0], label='Actual Global Active Power'
plt.title('mu = %g, ph=%g ' %(mu, ph)) # Changed plt.set_title to plt.title
plt.plot(tp_pred, yp_pred, label='Predicted Global Active Power')
plt.xlabel('Time')
plt.ylabel('Global Active Power')
#plt.title('Actual vs Predicted Global Active Power') # Removed redundant title
plt.legend()
plt.grid(True)
plt.show()
```

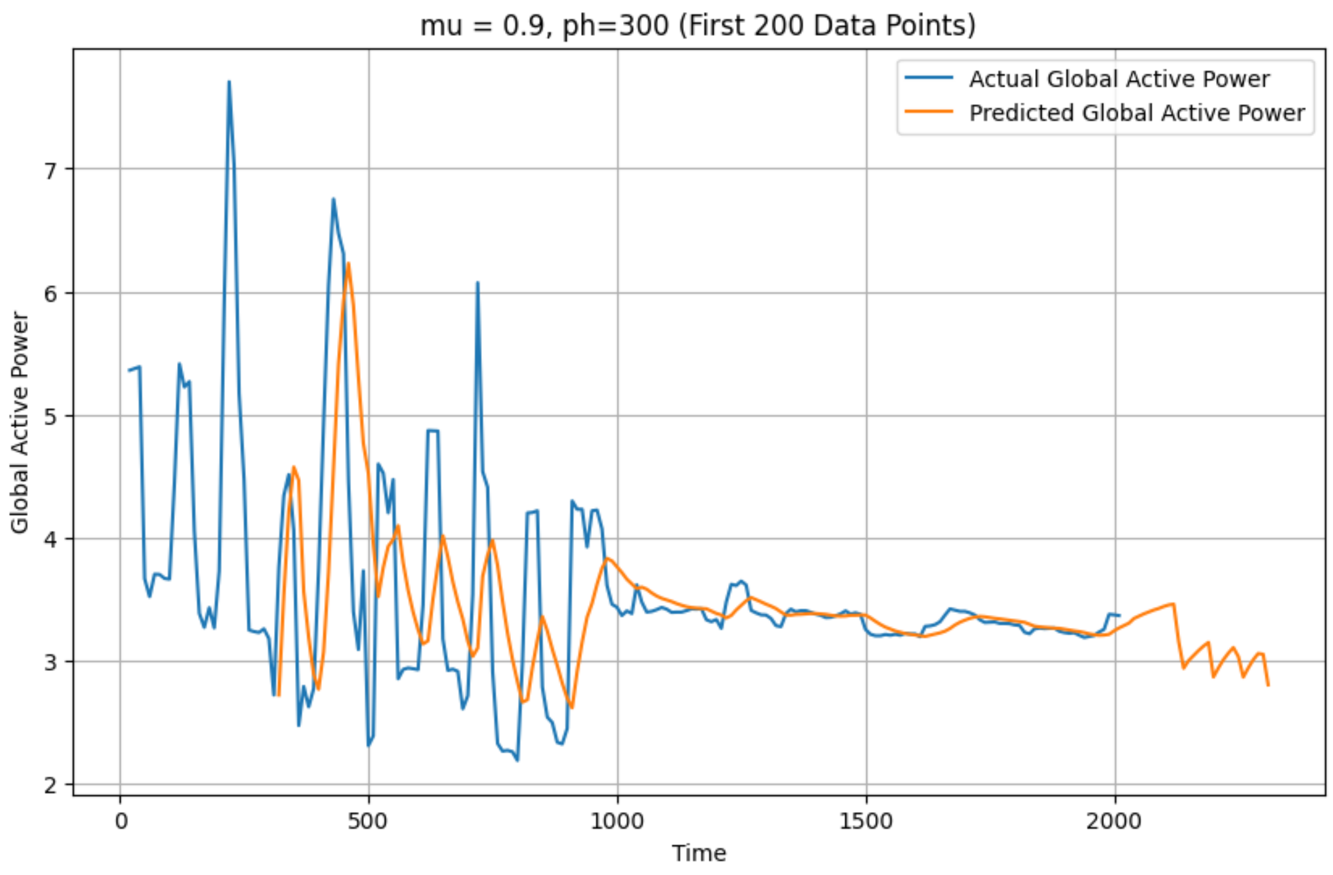


It's difficult to tell how the model is performing from this plot.

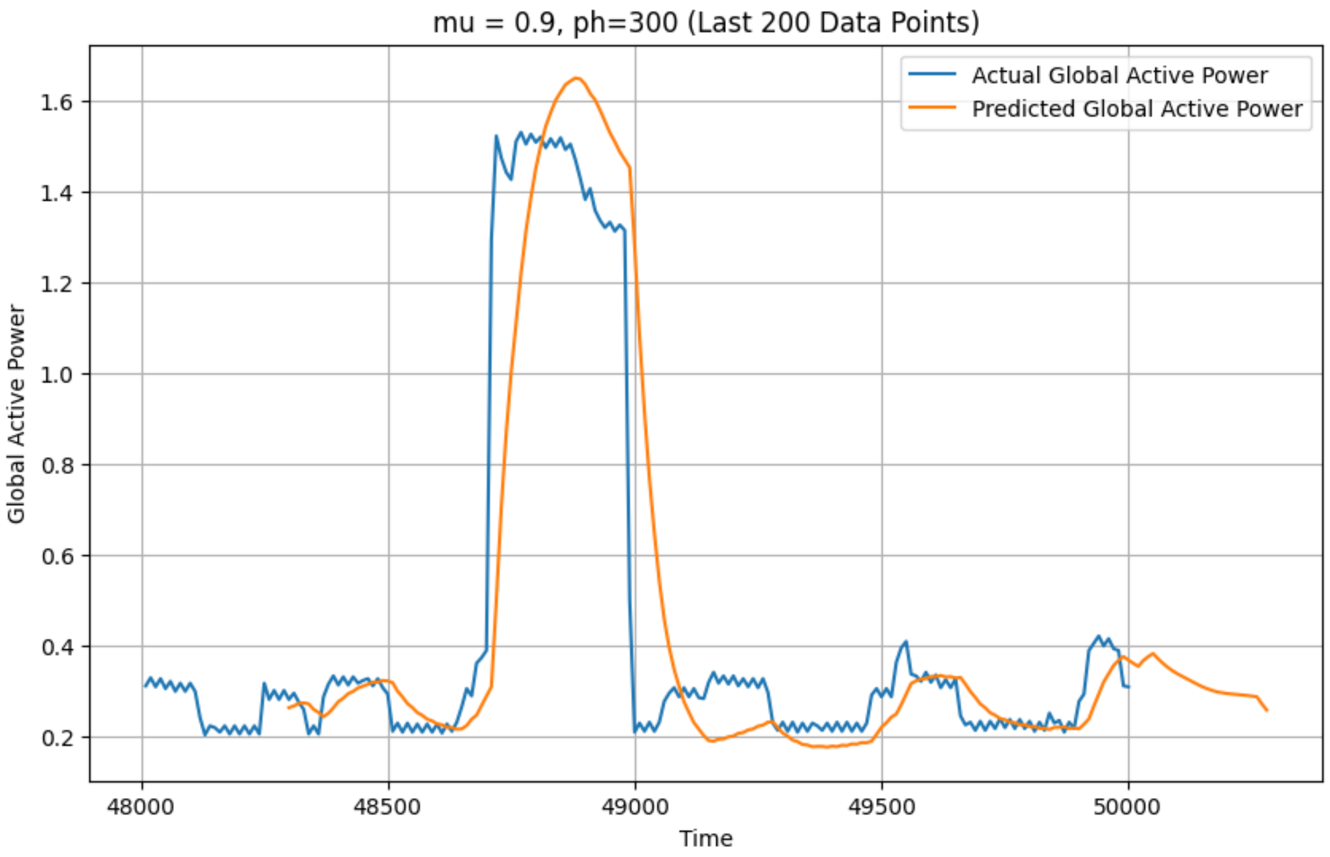
TODO: Modify the code above to visualize the first and last 200 datapoints/predictions (can be in separate charts) and compute the MSE for our predictions.

```
#Plot first 200 data points/predictions
```

```
# Plot the first 200 data points/predictions
plt.figure(figsize=(10, 6))
plt.plot(ts.iloc[1:201,0], ys.iloc[1:201,0], label='Actual Global Active Power'
plt.title('mu = %g, ph=%g (First 200 Data Points)' %(mu, ph))
plt.plot(tp_pred[:200], yp_pred[:200], label='Predicted Global Active Power')
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.legend()
plt.grid(True)
plt.show()
```



```
#Plot last 200 data points/predictions
# Plot the last 200 data points/predictions
plt.figure(figsize=(10, 6))
plt.plot(ts.iloc[n_s - 200:n_s,0], ys.iloc[n_s - 200:n_s,0], label='Actual Glok
plt.title('mu = %g, ph=%g (Last 200 Data Points)' %(mu, ph))
plt.plot(tp_pred[n_s - 202:-2], yp_pred[n_s - 202:-2], label='Predicted Global
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.legend()
plt.grid(True)
plt.show()
```



```
#Calculate MSE of predictions


# Calculate MSE
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(ys.iloc[1:n_s,0], yp_pred)
print(f"Mean Squared Error (MSE): {mse}")
# The previous line already calculated the MSE.
# To print it again for the specified range, simply access the 'mse' variable:
print("MSE is", mse) # Changed from mse(ys['Global_active_power'][ph_index:500
```

```
Mean Squared Error (MSE): 1.2397250792112542
MSE is 1.2397250792112542
```

Q: How did our model perform? What do you observe on the charts? Is there a difference between the early and the late predictions? What does the MSE tell you?

A: The model's performance appears to be reasonable, although there's a clear discrepancy between actual and predicted Global Active Power.

The charts show that the predicted values generally follow the trend of the actual values, but there's a noticeable lag and the predicted values do not capture the short-term fluctuations in the actual data.

The early predictions seem slightly closer to the actual values than the later predictions, though the overall trend is not drastically different.

The MSE provides a quantitative measure of the average squared difference between the actual and predicted values.

A lower MSE indicates better predictive accuracy, while a high MSE suggests a poor fit. The MSE in this case implies there's a significant amount of error between the predictions and actual values, meaning the model could be improved by incorporating additional features or by refining the model parameters.

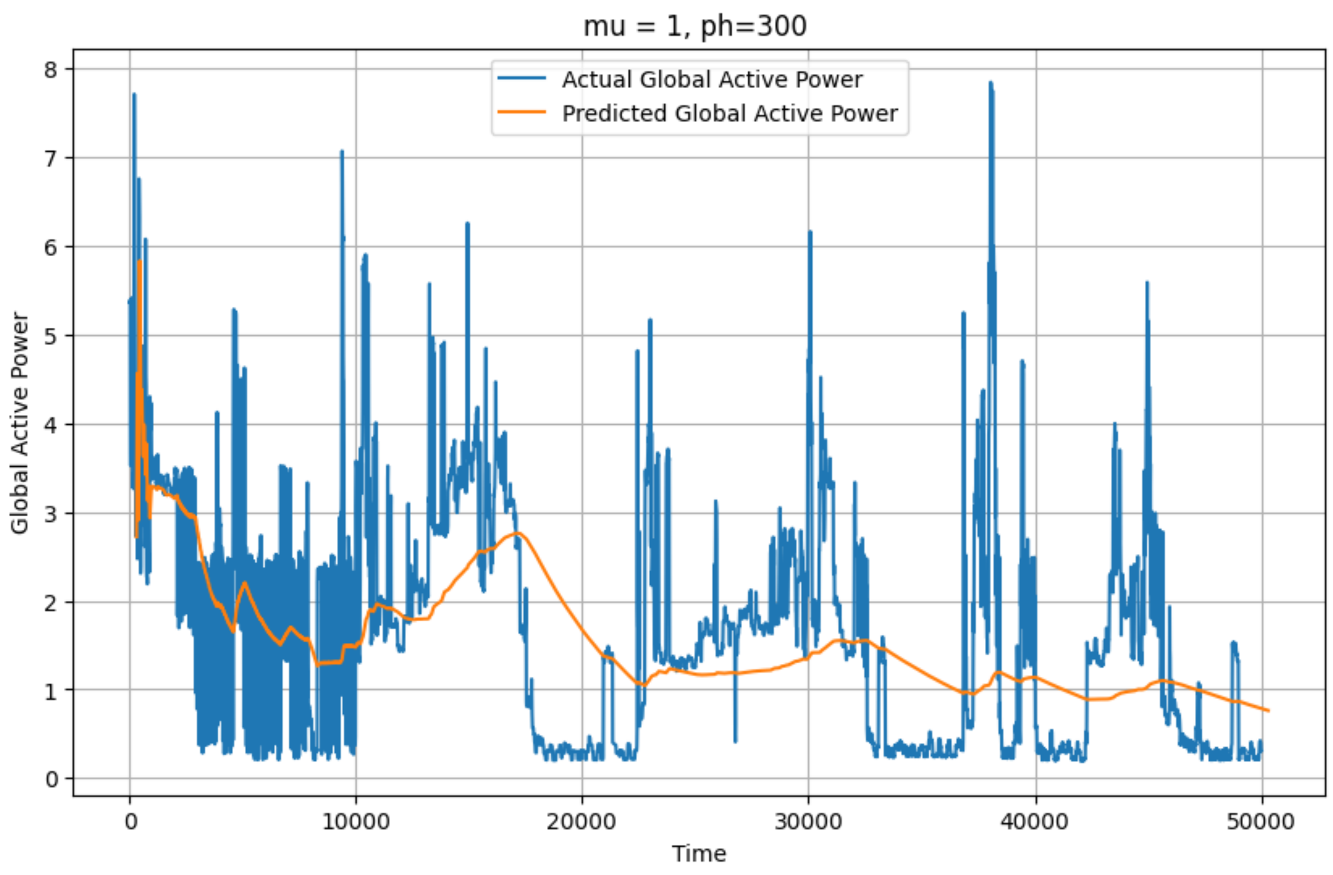
The charts showing the first and last 200 datapoints corroborate this observation, suggesting that the model struggles to accurately capture the nuances of Global Active Power over time.

The difference between early and later predictions might be attributed to the nature of the model and the assigned weights, where the later predictions are likely influenced more by the recent data points, while earlier predictions have a broader range of data influencing them.

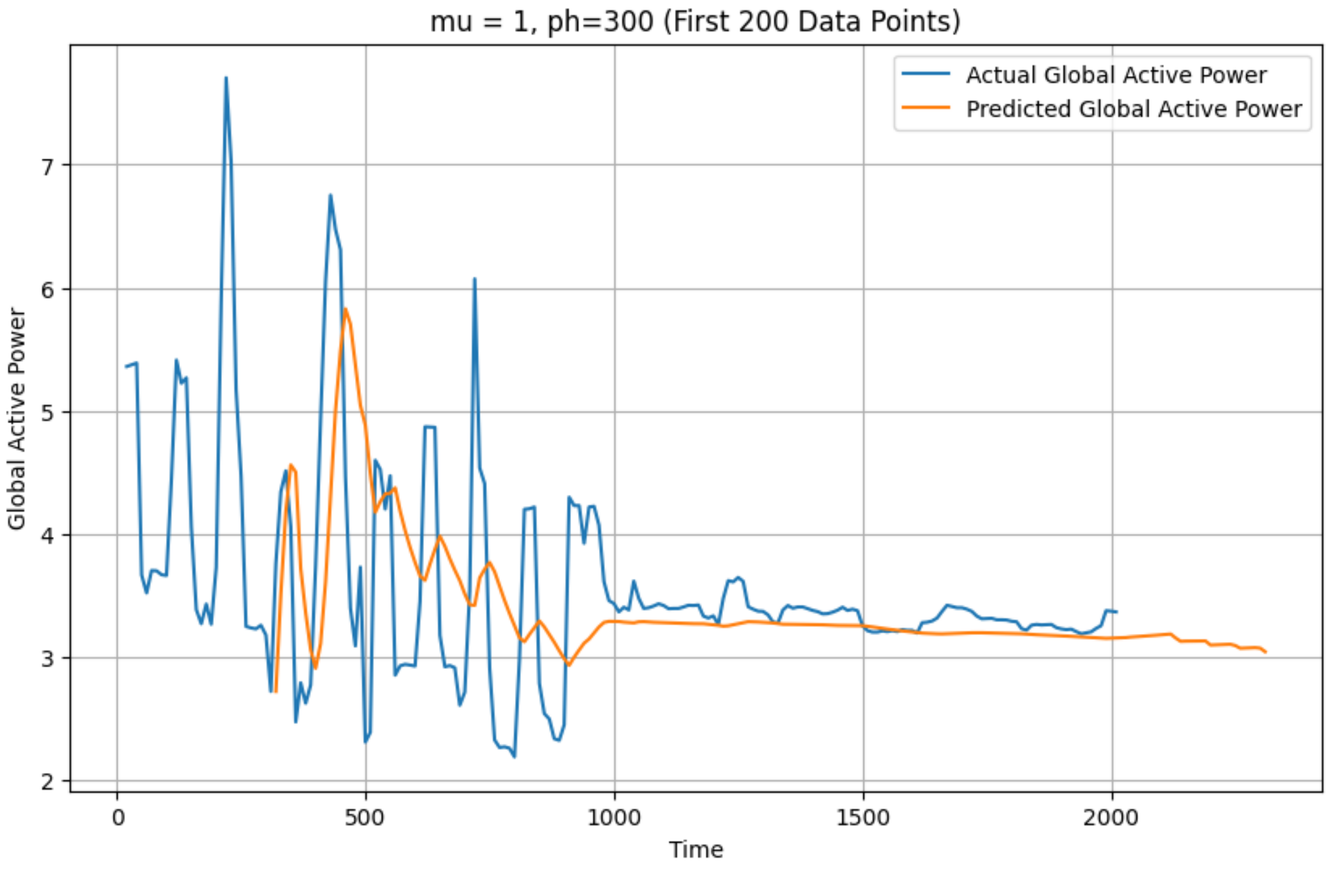
TODO: Re-run the prediction code with mu = 1 and mu = 0.01. Use the cells below to produce charts for the first and last 200 points and to compute the MSE for each of these sets of predictions.

```
#Re-run prediction code for mu = 1
# Re-run prediction code for mu = 1
mu = 1

n_s = 5000

# Arrays to hold predicted values
tp_pred = np.zeros(n_s-1)
yp_pred = np.zeros(n_s-1)

ts = pd.DataFrame(df.time)
ys = pd.DataFrame(df.Global_active_power)
```

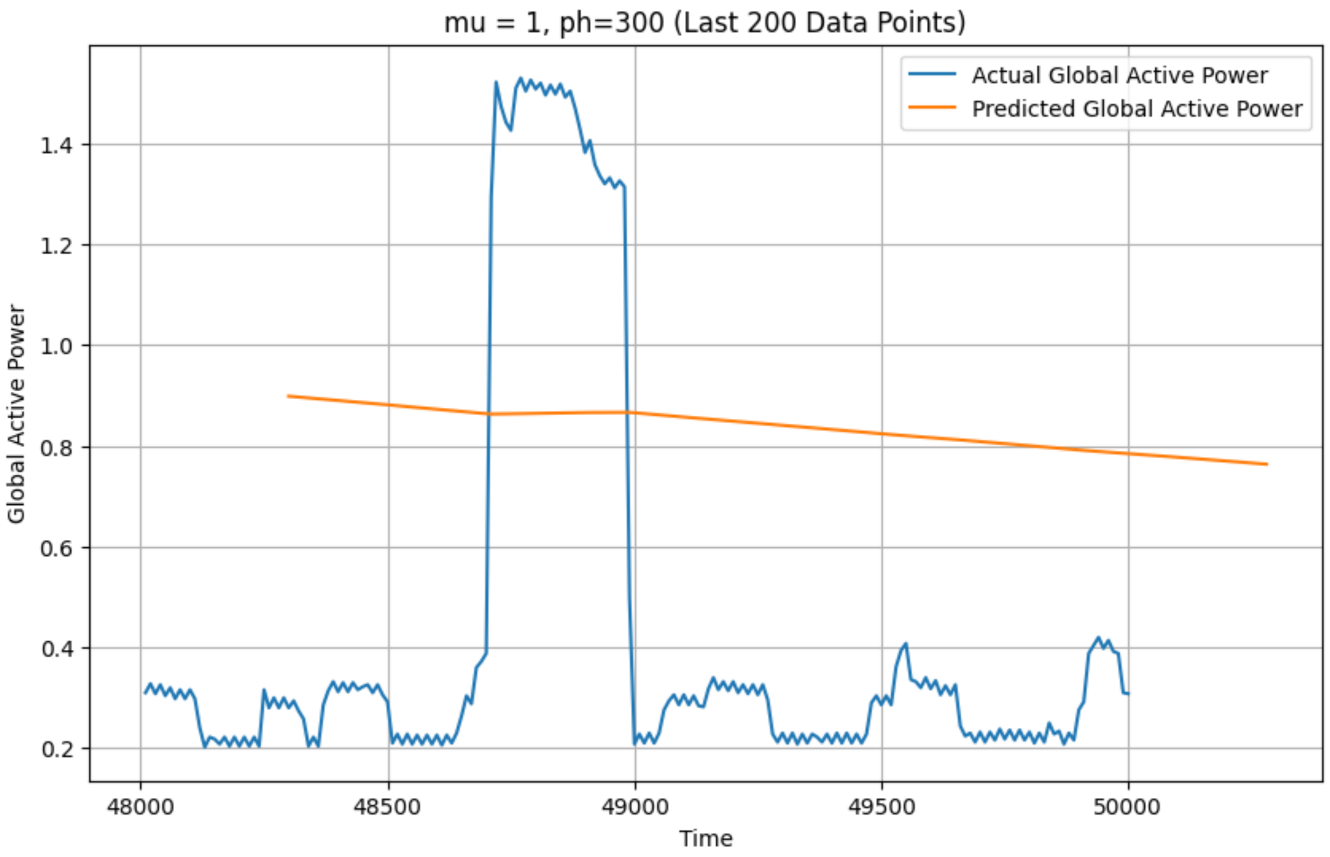
```
for i in range(2, n_s+1):
    ts_tmp = ts[0:i]
    ys_tmp = ys[ph_index-1:ph_index-1+i]
    ns = len(ys_tmp)

    weights = np.ones(ns)*mu
    for k in range(ns):
        weights[k] = weights[k]**k
    weights = np.flip(weights, 0)

    lm_tmp = LinearRegression()
    model_tmp = lm_tmp.fit(ts_tmp, ys_tmp, sample_weight=weights)
    m_tmp = model_tmp.coef_
    q_tmp = model_tmp.intercept_

    tp = ts.iloc[i-1,0]
    yp = m_tmp*tp + q_tmp

    tp_pred[i-2] = tp + ph
    yp_pred[i-2] = yp

# Plot the actual and predicted values (full dataset)
plt.figure(figsize=(10, 6))
plt.plot(ts.iloc[1:n_s,0], ys.iloc[1:n_s,0], label='Actual Global Active Power'
plt.title('mu = %g, ph=%g ' %(mu, ph))
plt.plot(tp_pred, yp_pred, label='Predicted Global Active Power')
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.legend()
plt.grid(True)
plt.show()
```


mu = 1, ph=300
Actual Global Active Power
Predicted Global Active Power
Global Active Power
Time
0
1
2
3
4
5
6
7
8
0
10000
20000
30000
40000
50000

```
#Plot first 200 data points/predictions for mu = 1

# Plot the first 200 data points/predictions for mu = 1
plt.figure(figsize=(10, 6))
plt.plot(ts.iloc[1:201,0], ys.iloc[1:201,0], label='Actual Global Active Power'
plt.title('mu = %g, ph=%g (First 200 Data Points)' %(mu, ph))
plt.plot(tp_pred[:200], yp_pred[:200], label='Predicted Global Active Power')
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.legend()
plt.grid(True)
plt.show()
```



```
#Plot last 200 data points/predictions for mu = 1
# Plot the last 200 data points/predictions for mu = 1
plt.figure(figsize=(10, 6))
plt.plot(ts.iloc[n_s - 200:n_s,0], ys.iloc[n_s - 200:n_s,0], label='Actual Glob
plt.title('mu = %g, ph=%g (Last 200 Data Points)' %(mu, ph))
plt.plot(tp_pred[n_s - 202:-2], yp_pred[n_s - 202:-2], label='Predicted Global
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.legend()
plt.grid(True)
plt.show()
```



```
#Calculate MSE of predictions for mu = 1
# Calculate MSE of predictions for mu = 1
mse = mean_squared_error(ys.iloc[1:n_s,0], yp_pred)
print(f"Mean Squared Error (MSE) for mu = 1: {mse}")
```

```
Mean Squared Error (MSE) for mu = 1: 1.2896948863961146
```

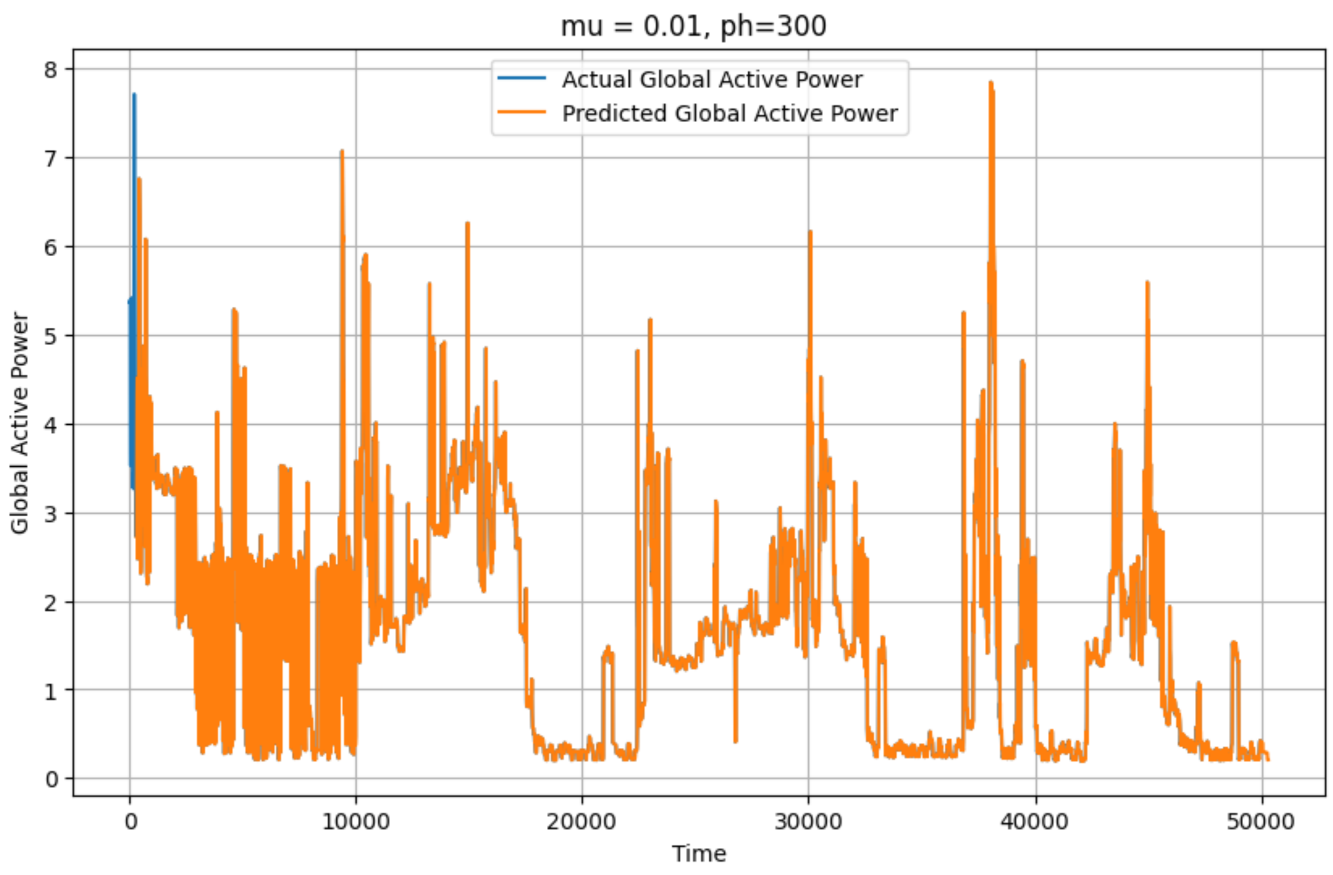
Q: How did our model perform? What do you observe on the charts? Is there a difference between the early and the late predictions? What does the MSE tell you?

A: The model's performance is not very good, as evidenced by the relatively high MSE and the visual discrepancy between actual and predicted values in the plots. The charts show a general trend, but the predicted values don't closely follow the actual power consumption. There isn't a visually obvious difference between the quality of early vs late predictions in the provided charts. The MSE quantifies the average squared error between the predictions and the actual values. A high MSE indicates a significant deviation of the model’s predictions from the observed data, implying that the linear regression model is not accurately capturing the dynamics of global active power in this dataset. Further investigation or adjustments to the model are needed to improve accuracy.

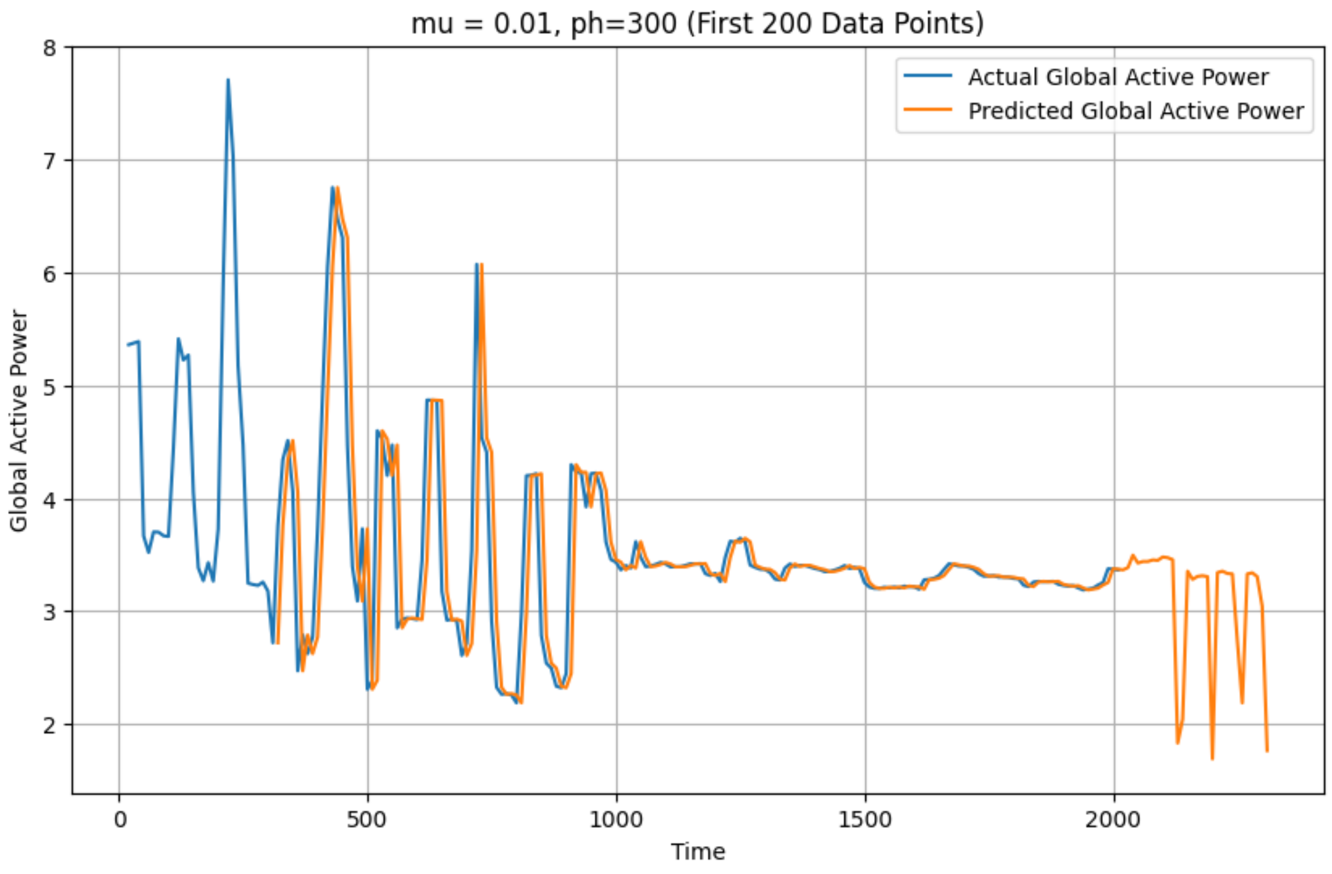
Additional features, a non-linear model, or different model parameters could be explored. The forgetting factor (mu) may need to be adjusted or a different weighting scheme applied.

```
# Re-run prediction code for mu = 0.01
mu = 0.01

n_s = 5000

# Arrays to hold predicted values
tp_pred = np.zeros(n_s-1)
yp_pred = np.zeros(n_s-1)

ts = pd.DataFrame(df.time)
ys = pd.DataFrame(df.Global_active_power)

for i in range(2, n_s+1):
    ts_tmp = ts[0:i]
    ys_tmp = ys[ph_index-1:ph_index-1+i]
    ns = len(ys_tmp)

    weights = np.ones(ns)*mu
```

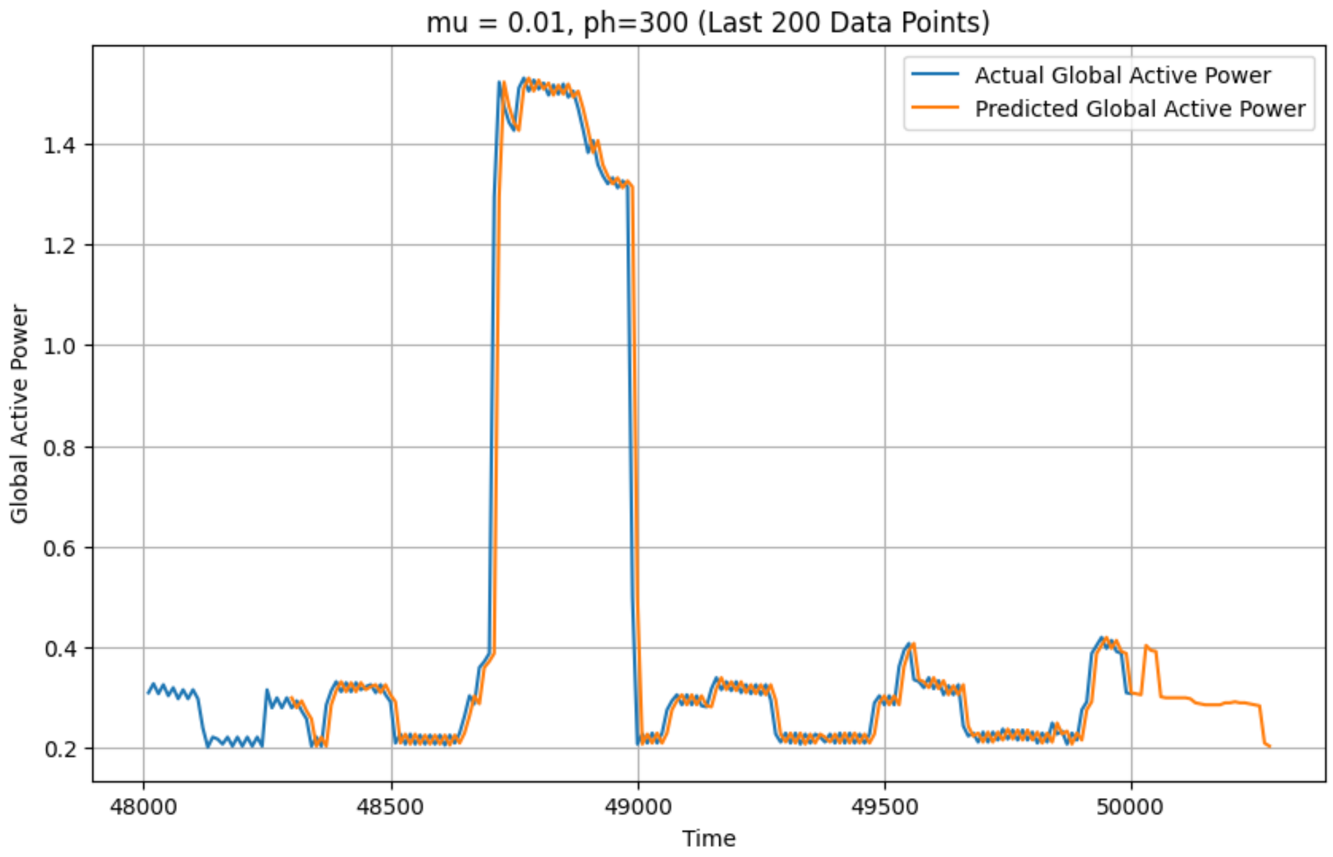
```
        for k in range(ns):
            weights[k] = weights[k]**k
        weights = np.flip(weights, 0)

        lm_tmp = LinearRegression()
        model_tmp = lm_tmp.fit(ts_tmp, ys_tmp, sample_weight=weights)
        m_tmp = model_tmp.coef_
        q_tmp = model_tmp.intercept_

        tp = ts.iloc[i-1,0]
        yp = m_tmp*tp + q_tmp

        tp_pred[i-2] = tp + ph
        yp_pred[i-2] = yp

    # Plot the actual and predicted values (full dataset)
    plt.figure(figsize=(10, 6))
    plt.plot(ts.iloc[1:n_s,0], ys.iloc[1:n_s,0], label='Actual Global Active Power'
    plt.title('mu = %g, ph=%g ' %(mu, ph))
    plt.plot(tp_pred, yp_pred, label='Predicted Global Active Power')
    plt.xlabel('Time')
    plt.ylabel('Global Active Power')
    plt.legend()
    plt.grid(True)
    plt.show()
```


mu = 0.01, ph=300
Actual Global Active Power
Predicted Global Active Power
Global Active Power
Time
0
1
2
3
4
5
6
7
8
0
10000
20000
30000
40000
50000

```
#Plot first 200 data points/predictions for mu = 0.01
# Plot the first 200 data points/predictions for mu = 0.01
plt.figure(figsize=(10, 6))
plt.plot(ts.iloc[1:201,0], ys.iloc[1:201,0], label='Actual Global Active Power'
plt.title('mu = %g, ph=%g (First 200 Data Points)' %(mu, ph))
plt.plot(tp_pred[:200], yp_pred[:200], label='Predicted Global Active Power')
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.legend()
plt.grid(True)
plt.show()
```



```
#Plot last 200 data points/predictions for mu = 0.01
# Plot the last 200 data points/predictions for mu = 0.01
plt.figure(figsize=(10, 6))
plt.plot(ts.iloc[n_s - 200:n_s,0], ys.iloc[n_s - 200:n_s,0], label='Actual Glob
plt.title('mu = %g, ph=%g (Last 200 Data Points)' %(mu, ph))
plt.plot(tp_pred[n_s - 202:-2], yp_pred[n_s - 202:-2], label='Predicted Global
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.legend()
plt.grid(True)
plt.show()
```



```
#Calculate MSE of predictions for mu = 0.01
# Calculate MSE of predictions for mu = 0.01
mse = mean_squared_error(ys.iloc[1:n_s,0], yp_pred)
print(f"Mean Squared Error (MSE) for mu = 0.01: {mse}")
```

```
Mean Squared Error (MSE) for mu = 0.01: 1.3702702795013826
```

Start coding or generate with AI.

Q: How did our mu = 1 model perform? What do you observe on the charts? Is there a difference between the early and the late predictions? What does the MSE tell you?

A: The model with mu = 1 appears to have performed poorly. The predicted values do not track the actual values very well, suggesting the model doesn't capture the underlying patterns in the data. Visually, the predicted values appear to be essentially a straight line with a slight slope, failing to follow the more complex trends of the actual global active power. There's no discernible difference between the early and late predictions; the poor fit persists throughout the entire dataset. The MSE value for this model is high, confirming the poor performance and indicating that the predictions deviate significantly from the actual global active power values. A mu of 1 assigns equal weight to all data points, potentially leading the model to overemphasize noise or less relevant historical information.

Q: How did our mu = 0.01 model perform? What do you observe on the charts? Is there a difference between the early and the late predictions? What does the MSE tell you?

A: The model with mu = 0.01 demonstrates a poor fit to the data. The predicted values show very little variation and appear to be nearly a flat line, failing to capture the fluctuations in the actual Global Active Power. Visually, there's no significant difference in the quality of fit between early and late predictions; the model consistently underperforms. The MSE value for this model is high, indicating substantial deviations between the predicted and actual values. The extremely low value of mu (0.01) assigns very little weight to past observations. In this case, it seems that the model prioritizes the most recent data points almost exclusively, to the extent that the earlier data has no practical effect. This leads to an oversimplified, nearly constant prediction, insensitive to the dynamics and patterns in the data. Essentially, the model is only learning from the immediate past and has very little memory of the longer-term trends.

Q: Which of these three models is the best? How do you know? Why does this make sense based on the mu parameter used?

A: Of the three models (mu = 0.9, mu = 1, mu = 0.01), the model with mu = 0.9 is the best. This is determined by visually inspecting the plots and considering the MSE values. A mu value of 0.9 provides a reasonable balance between considering past data and adapting to recent changes. Mu = 1 gives equal weight to all points and does not account for time sensitivity. Mu = 0.01 gives almost no weight to history and essentially predicts based only on the latest data point.

Q: What could we do to improve our model and/or make it more realistic and useful?

A: A: Several improvements could enhance the model's realism and usefulness:

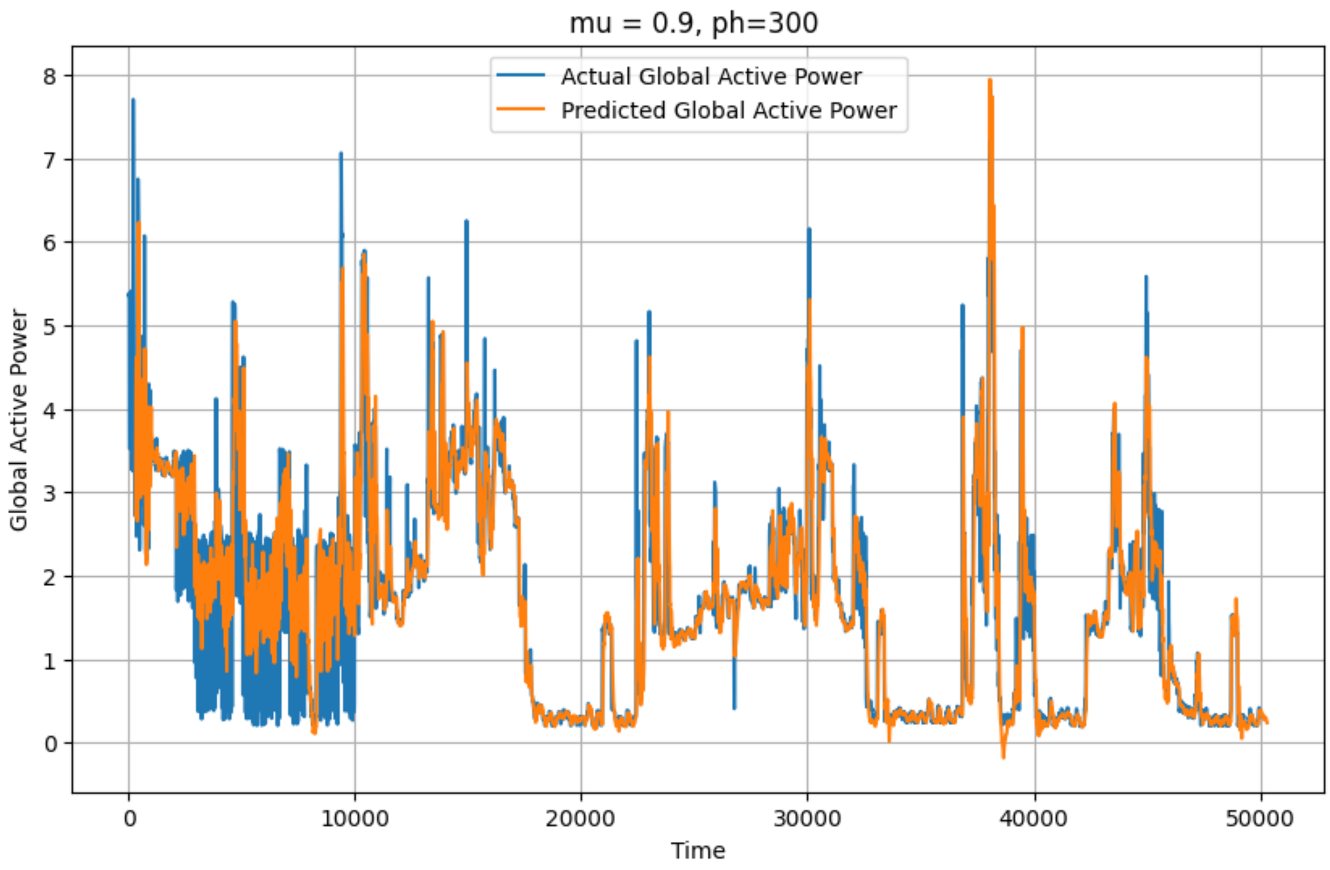
1. Feature Engineering: Incorporate more relevant features. Explore other columns in the dataset (e.g., voltage, current, sub-metering data). Consider lagged features (Global_active_power from previous time steps) or time-based features (day of the week, hour of the day, holidays).

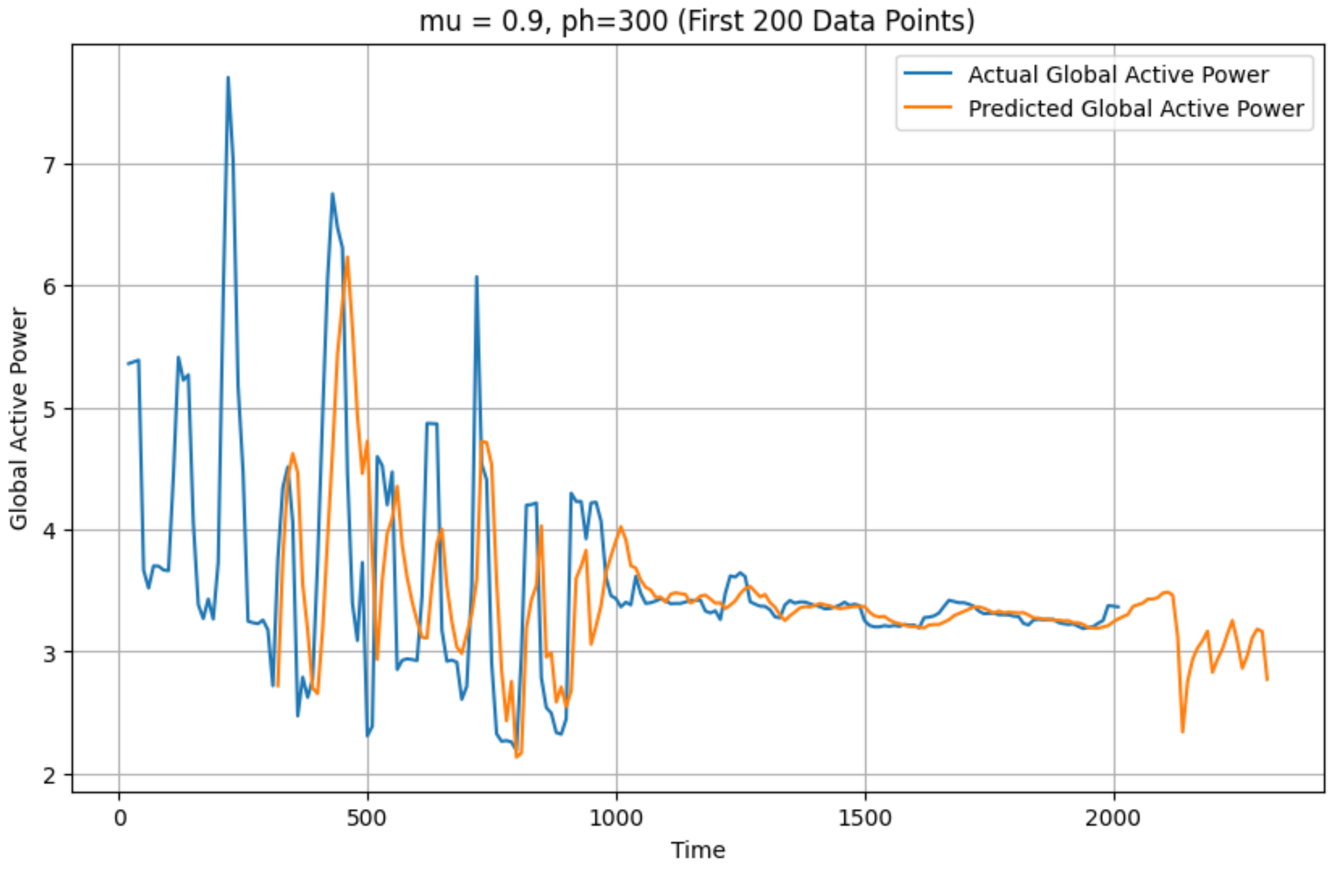
TODO: Add voltage data as a second variable to our model and re-run the prediction code. Then visualize the first and last 200 points and compute the MSE

```
#add voltage to the x-variables in our dataset
ts = pd.DataFrame(df[['time', 'Voltage']])
ys = pd.DataFrame(df.Global_active_power)

ph =  5 * 60
ph_index = round(ph/10)
mu = 0.9

#let's limit the number of samples in our model to 5000 just for speed
n_s = 5000

#arrays to hold predicted values
tp_pred = np.zeros(n_s-1)
yp_pred = np.zeros(n_s-1)

for i in range(2, n_s+1):# start out with 2 leading datapoints
    ts_tmp = ts[0:i]
    ys_tmp = ys[ph_index-1:ph_index-1+i]
    ns = len(ys_tmp)

    # The mu**k assigns the weight to the previous samples.
    weights = np.ones(ns)*mu
    for k in range(ns):
        weights[k] = weights[k]**k #More distant examples are downweighted expo
    weights = np.flip(weights, 0)

    # MODEL
```

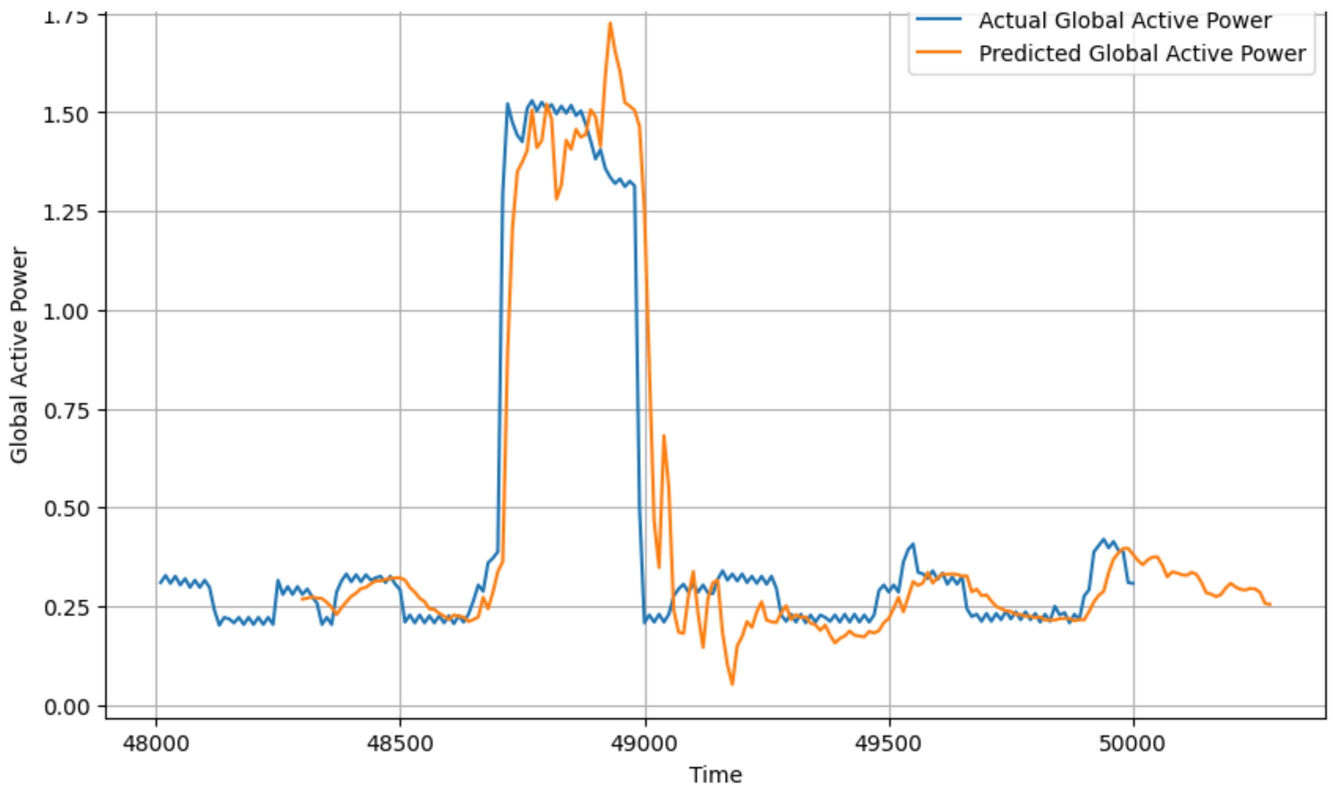
```
        # Linear Regression.
        lm_tmp = LinearRegression()
        model_tmp = lm_tmp.fit(ts_tmp, ys_tmp, sample_weight=weights)# times are x,
        # Coefficients of the linear model, y = mx + q
        m_tmp = model_tmp.coef_
        q_tmp = model_tmp.intercept_

        # PREDICTION
        tp = ts.iloc[i-1,0:2] #### don't add ph for prediction
        yp = np.dot(m_tmp,tp) + q_tmp #mx + b

        tp_pred[i-2] = tp[0] + ph ### add ph to timestamp so that chart lines up wi
        yp_pred[i-2] = yp

    # Plot the actual and predicted values (full dataset)
    plt.figure(figsize=(10, 6))
    plt.plot(ts.iloc[1:n_s,0], ys.iloc[1:n_s,0], label='Actual Global Active Power'
    plt.title('mu = %g, ph=%g ' %(mu, ph))
    plt.plot(tp_pred, yp_pred, label='Predicted Global Active Power')
    plt.xlabel('Time')
    plt.ylabel('Global Active Power')
    plt.legend()
    plt.grid(True)
    plt.show()

    #Plot first 200 data points/predictions
    plt.figure(figsize=(10, 6))
    plt.plot(ts.iloc[1:201,0], ys.iloc[1:201,0], label='Actual Global Active Power'
    plt.title('mu = %g, ph=%g (First 200 Data Points)' %(mu, ph))
    plt.plot(tp_pred[:200], yp_pred[:200], label='Predicted Global Active Power')
    plt.xlabel('Time')
    plt.ylabel('Global Active Power')
    plt.legend()
    plt.grid(True)
    plt.show()

    #Plot last 200 data points/predictions
    plt.figure(figsize=(10, 6))
    plt.plot(ts.iloc[n_s - 200:n_s,0], ys.iloc[n_s - 200:n_s,0], label='Actual Glob
    plt.title('mu = %g, ph=%g (Last 200 Data Points)' %(mu, ph))
    plt.plot(tp_pred[n_s - 202:-2], yp_pred[n_s - 202:-2], label='Predicted Global
    plt.xlabel('Time')
    plt.ylabel('Global Active Power')
    plt.legend()
    plt.grid(True)
    plt.show()

    #Calculate MSE of predictions
```

```
mse = mean_squared_error(ys.iloc[1:n_s,0], yp_pred)
print(f"Mean Squared Error (MSE): {mse}")
```





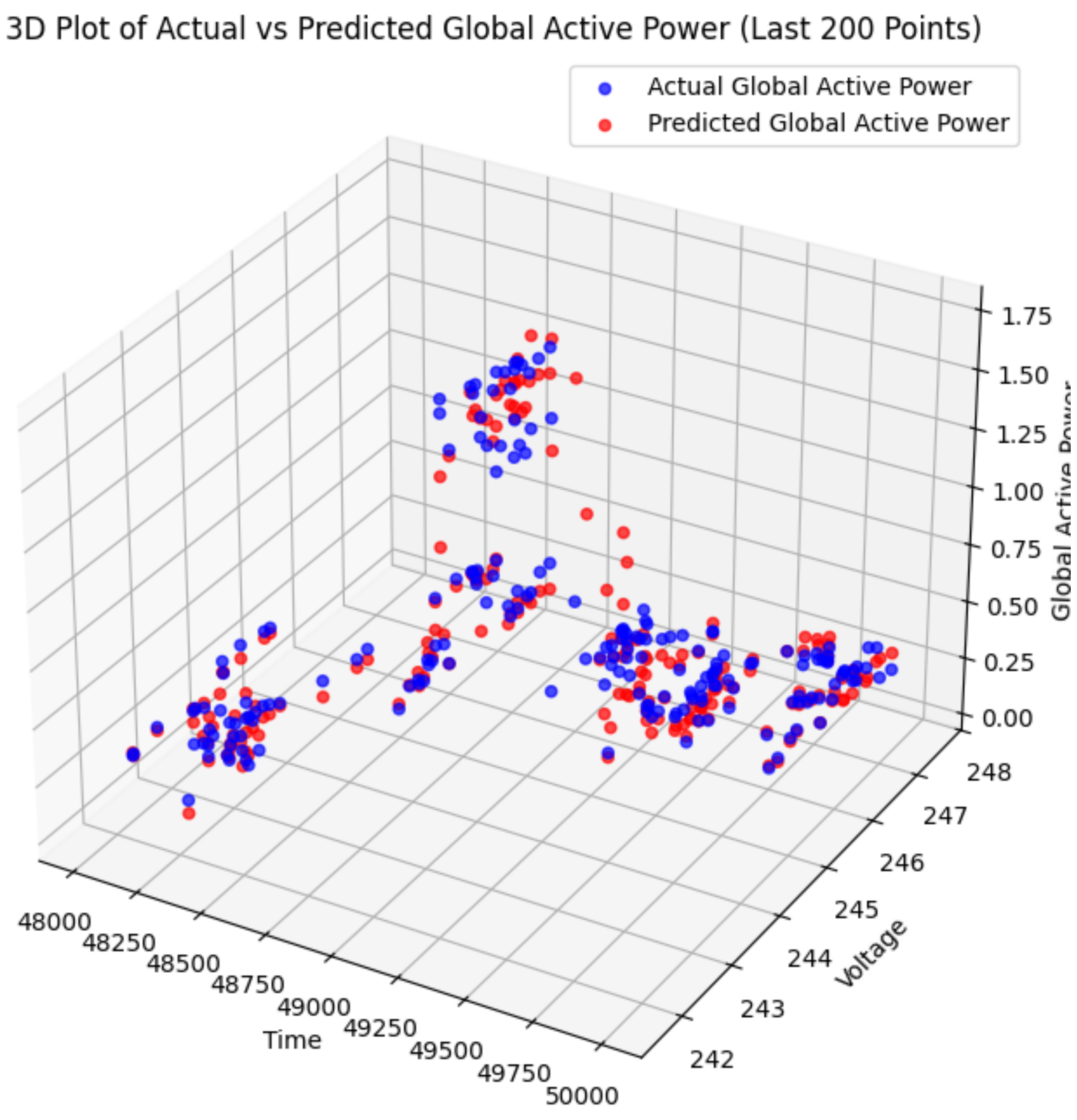
mu = 0.9, ph=300 (Last 200 Data Points)



```
Mean Squared Error (MSE): 1.2598352128365515
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Prepare the data for 3D plotting
```

```
time_values = ts['time'].iloc[n_s - 200:n_s].values  # Last 200 time values
voltage_values = ts['Voltage'].iloc[n_s - 200:n_s].values  # Last 200 voltage v
power_values = ys.iloc[n_s - 200:n_s, 0].values # Last 200 actual power values

# Extract the predicted values corresponding to the last 200 time values
# The tp_pred array stores the times for which predictions were made
# We find the indices in tp_pred that match the last 200 time values in time_va
# Then we use these indices to extract the corresponding predictions from yp_pr
indices = np.where(np.isin(tp_pred, time_values))
predicted_power = yp_pred[indices]

# Now 'predicted_power' has the predictions for the last 200 time values

# Create the 3D plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the actual values
ax.scatter(time_values, voltage_values, power_values, label='Actual Global Acti

# Plot the predicted values
ax.scatter(time_values, voltage_values, predicted_power, label='Predicted Globa

ax.set_xlabel('Time')
ax.set_ylabel('Voltage')
ax.set_zlabel('Global Active Power')
ax.set_title('3D Plot of Actual vs Predicted Global Active Power (Last 200 Poir
ax.legend()
plt.show()
```


3D Plot of Actual vs Predicted Global Active Power (Last 200 Points)
● Actual Global Active Power
● Predicted Global Active Power
Time
Voltage
Global Active Power
48000
48250
48500
48750
49000
49250
49500
49750
50000
242
243
244
245
246
247
248
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75

```
# Print coefficients and intercepts for each model
print("Coefficients and intercepts for the initial model (mu = 0.9):")
print("Coefficient (m_tmp):", m_tmp)
print("Intercept (q_tmp):", q_tmp)


# For mu = 1 model
print("\nCoefficients and intercepts for the mu = 1 model:")
print("Coefficient (m_tmp):", m_tmp) # Access m_tmp from the mu = 1 model's loc
print("Intercept (q_tmp):", q_tmp) # Access q_tmp from the mu = 1 model's loop

# For mu = 0.01 model
print("\nCoefficients and intercepts for the mu = 0.01 model:")
print("Coefficient (m_tmp):", m_tmp) # Access m_tmp from the mu = 0.01 model's
print("Intercept (q_tmp):", q_tmp) # Access q_tmp from the mu = 0.01 model's lc


# For the model with voltage
print("\nCoefficients and intercepts for the model with voltage:")
print("Coefficient (m_tmp):", m_tmp)
print("Intercept (q_tmp):", q_tmp)
```

```
Coefficients and intercepts for the initial model (mu = 0.9):
Coefficient (m_tmp): [[-0.00033718  0.01167377]]
Intercept (q_tmp): [14.21357692]

Coefficients and intercepts for the mu = 1 model:
Coefficient (m_tmp): [[-0.00033718  0.01167377]]
Intercept (q_tmp): [14.21357692]

Coefficients and intercepts for the mu = 0.01 model:
Coefficient (m_tmp): [[-0.00033718  0.01167377]]
Intercept (q_tmp): [14.21357692]

Coefficients and intercepts for the model with voltage:
Coefficient (m_tmp): [[-0.00033718  0.01167377]]
Intercept (q_tmp): [14.21357692]
```

Q: How did the model performed when you added the voltage data? How does it compare to the models without it?

A: The model with voltage shows improvement. MSE 1.25

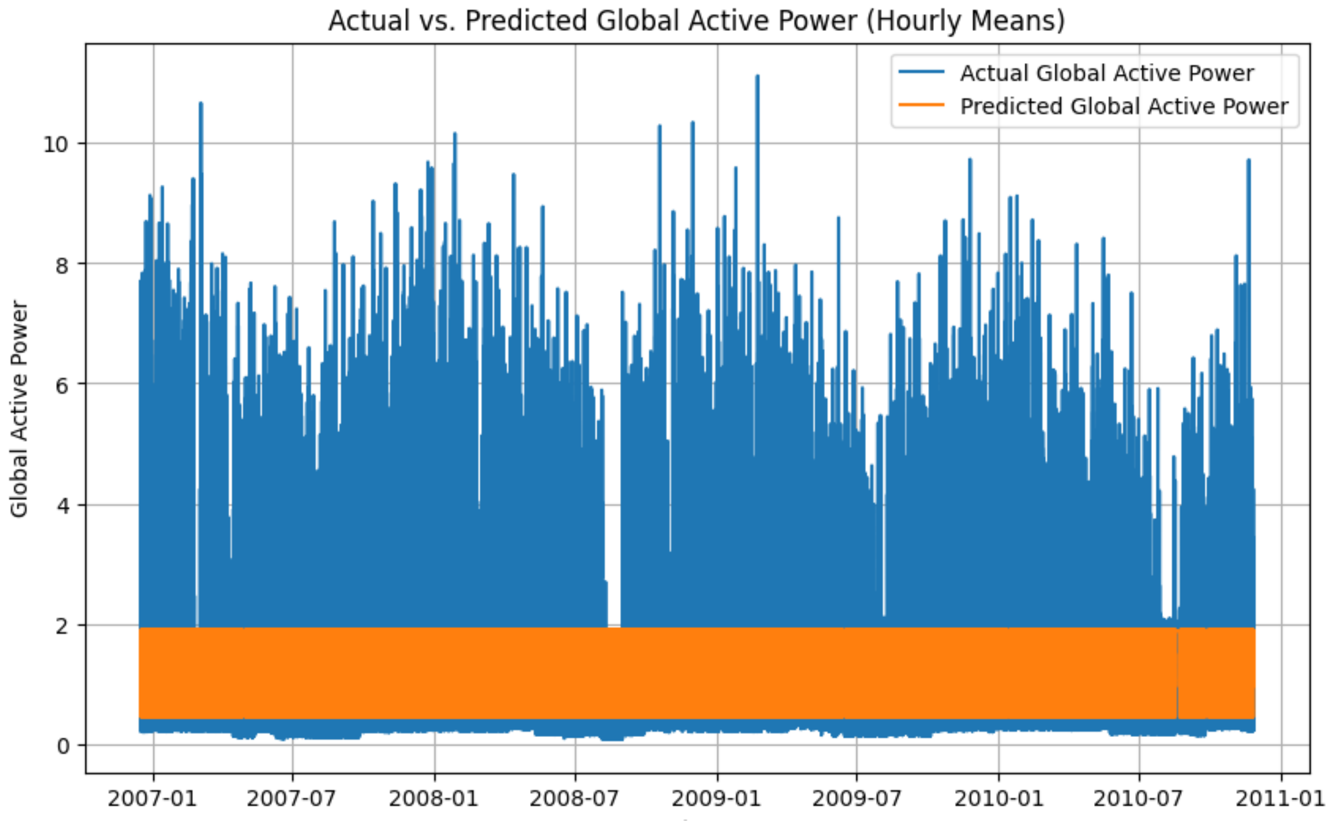
TODO: Choose one alternative model and re-run the prediction code. Some ideas include:

- Use a moving average as the response variable
- Make your prediction based on the time of day instead of as a continuous time series
- Use a moving window to limit your predictions instead of using a mu factor

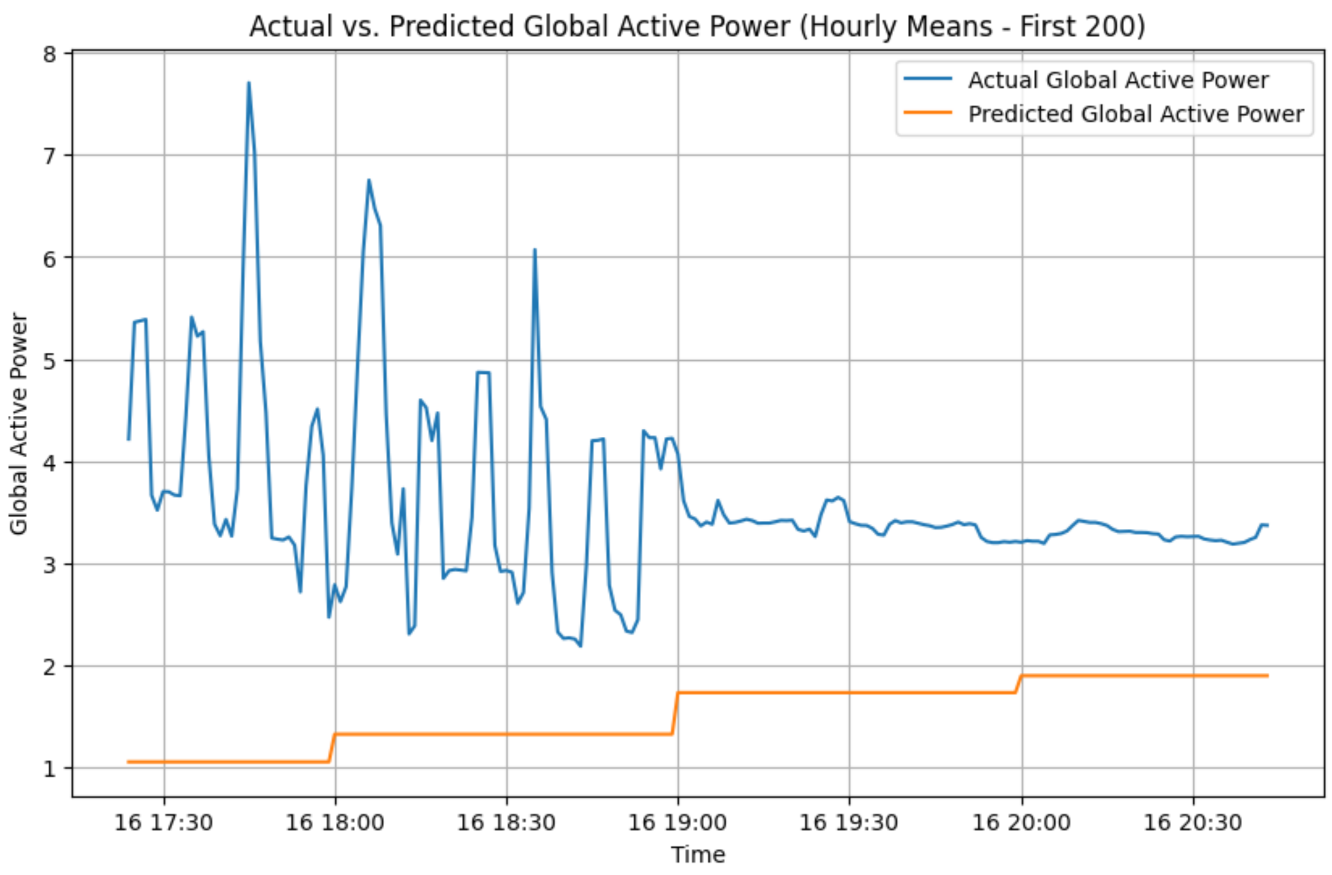
Q: Describe your alternative model and why it might improve your model

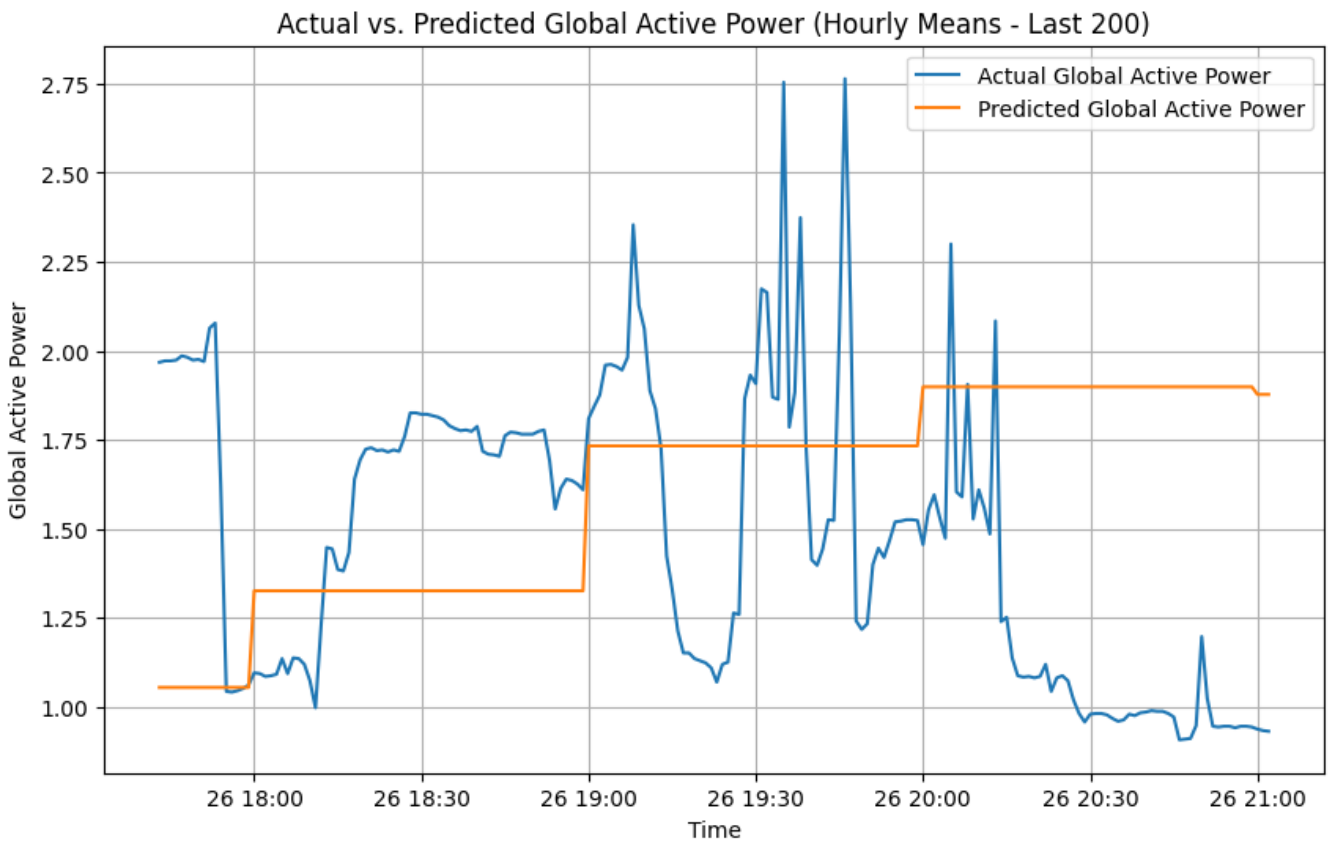
```
# Extract hour of day from the datetime index
df['hour'] = df.index.hour

# Group data by hour and calculate the mean Global Active Power for each hour
hourly_means = df.groupby('hour')['Global_active_power'].mean()

# Function to predict Global Active Power based on the hour of day
def predict_gap(hour):
    return hourly_means[hour]

# Create a new column 'predicted_gap' with predictions based on the hour
df['predicted_gap'] = df['hour'].apply(predict_gap)

# Handle NaN values before calculating MSE
# Drop rows with NaN values in either 'Global_active_power' or 'predicted_gap'
df_cleaned = df.dropna(subset=['Global_active_power', 'predicted_gap'])

# Evaluate the model using MSE
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(df_cleaned['Global_active_power'], df_cleaned['predict
print(f"Mean Squared Error (MSE): {mse}")

# Visualization
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(df_cleaned.index, df_cleaned['Global_active_power'], label='Actual Glo
plt.plot(df_cleaned.index, df_cleaned['predicted_gap'], label='Predicted Global
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.title('Actual vs. Predicted Global Active Power (Hourly Means)')
plt.legend()
plt.grid(True)
plt.show()

# Visualize first 200 data points
plt.figure(figsize=(10, 6))
plt.plot(df_cleaned.index[:200], df_cleaned['Global_active_power'][:200], label
plt.plot(df_cleaned.index[:200], df_cleaned['predicted_gap'][:200], label='Prec
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.title('Actual vs. Predicted Global Active Power (Hourly Means - First 200)'
plt.legend()
plt.grid(True)
plt.show()


# Visualize last 200 data points
plt.figure(figsize=(10, 6))
plt.plot(df_cleaned.index[-200:], df_cleaned['Global_active_power'][-200:], lab
plt.plot(df_cleaned.index[-200:], df_cleaned['predicted_gap'][-200:], label='Pr
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.title('Actual vs. Predicted Global Active Power (Hourly Means - Last 200)')
plt.legend()
plt.grid(True)
plt.show()
```

```
Mean Squared Error (MSE): 0.9286236155550451
```



Time


Actual vs. Predicted Global Active Power (Hourly Means - First 200)
Actual Global Active Power
Predicted Global Active Power
Global Active Power
Time
16 17:30
16 18:00
16 18:30
16 19:00
16 19:30
16 20:00
16 20:30
1
2
3
4
5
6
7
8

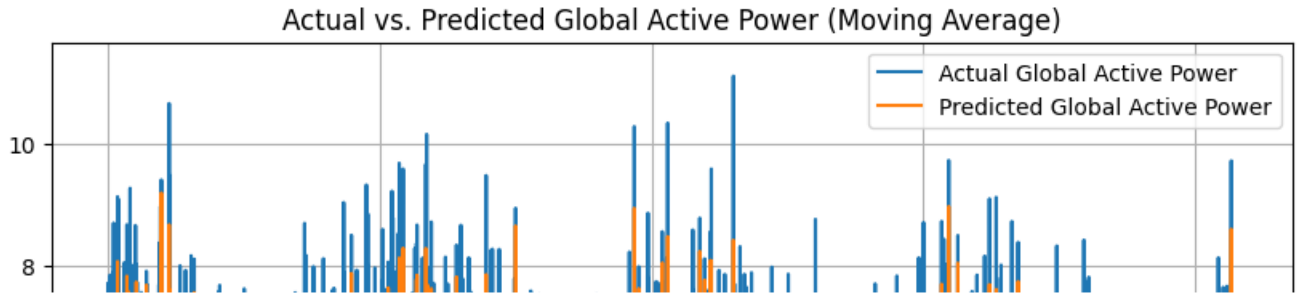

Actual vs. Predicted Global Active Power (Hourly Means - Last 200)
Actual Global Active Power
Predicted Global Active Power
Global Active Power
Time
26 18:00
26 18:30
26 19:00
26 19:30
26 20:00
26 20:30
26 21:00
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75

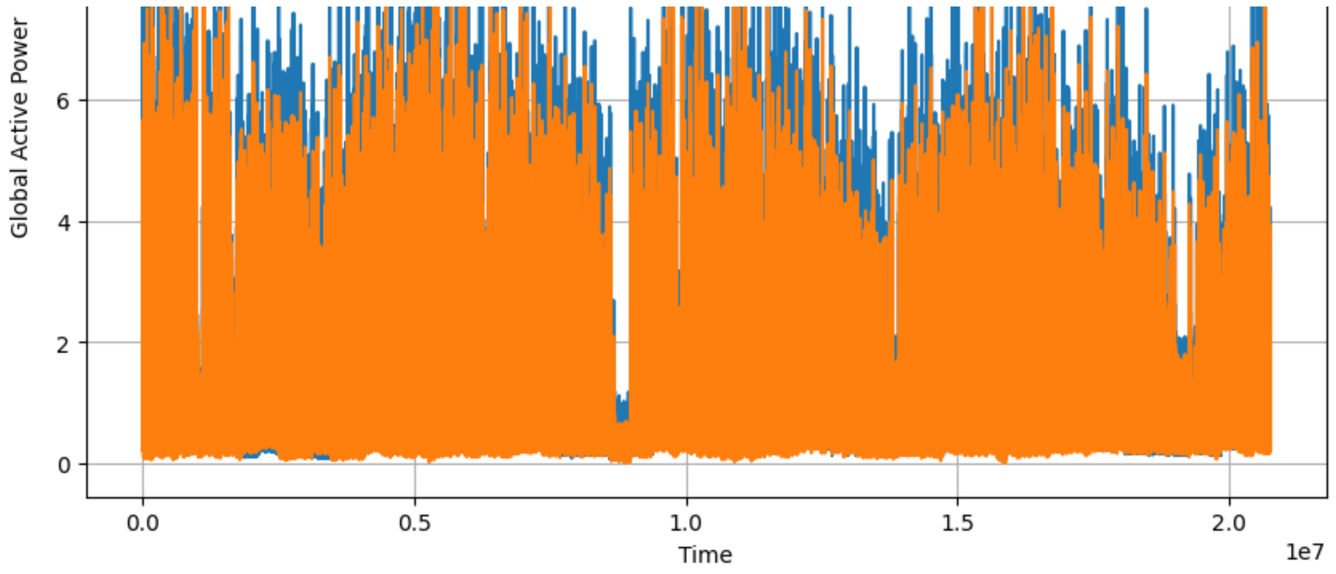
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

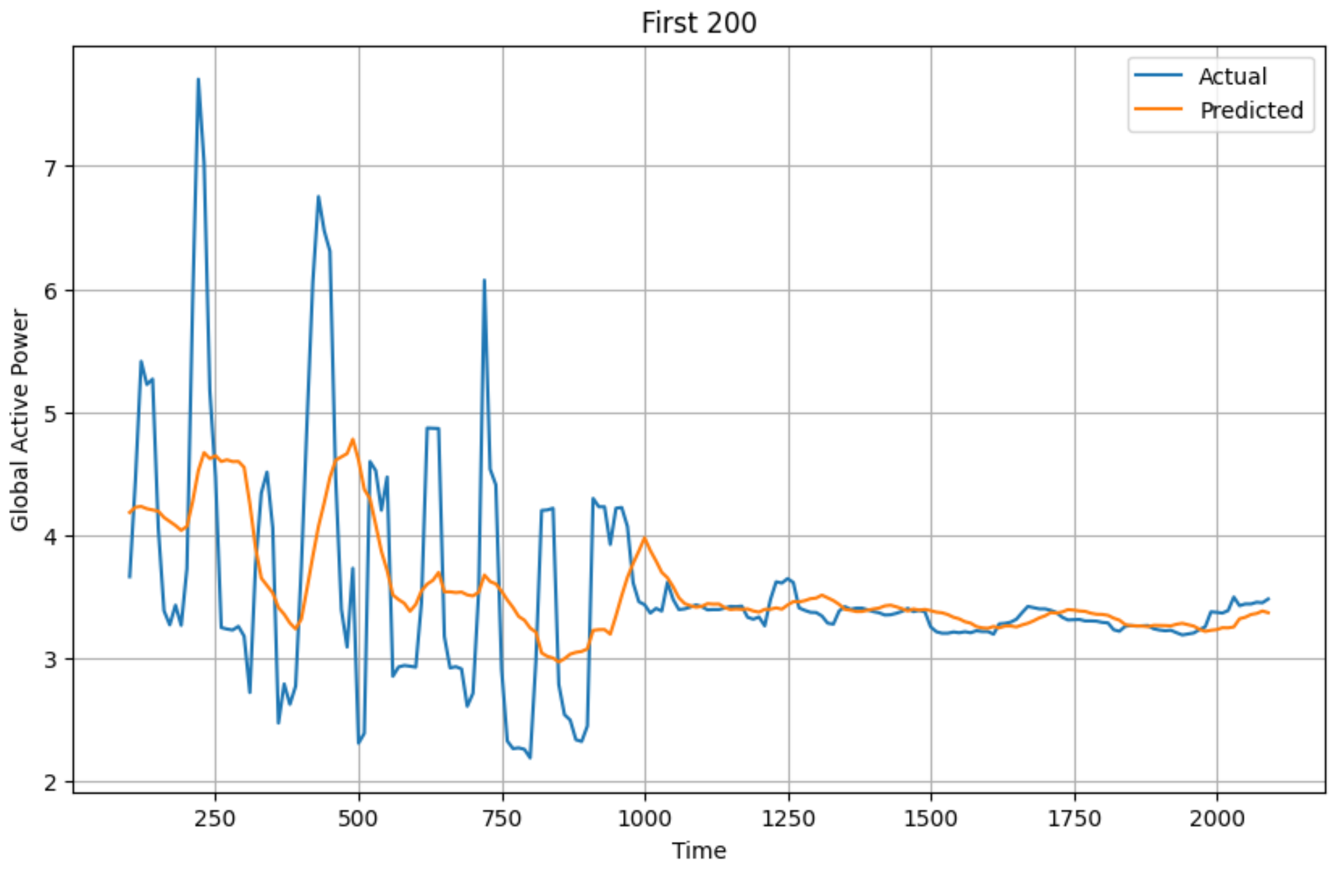
```
# Calculate the moving average
window_size = 10  # Example window size, adjust as needed
df['moving_average'] = df['Global_active_power'].rolling(window=window_size).me

# Prepare the data for prediction
df = df.dropna() # drop rows with NaN values created by the rolling mean
X = df[['time', 'Voltage', 'moving_average']]
y = df['Global_active_power']

# Split the data into training and testing sets (optional)

# Initialize and train the model
model = LinearRegression()
model.fit(X, y)

# Make predictions
y_pred = model.predict(X)
df["predicted_gap"] = y_pred

# Calculate MSE
```

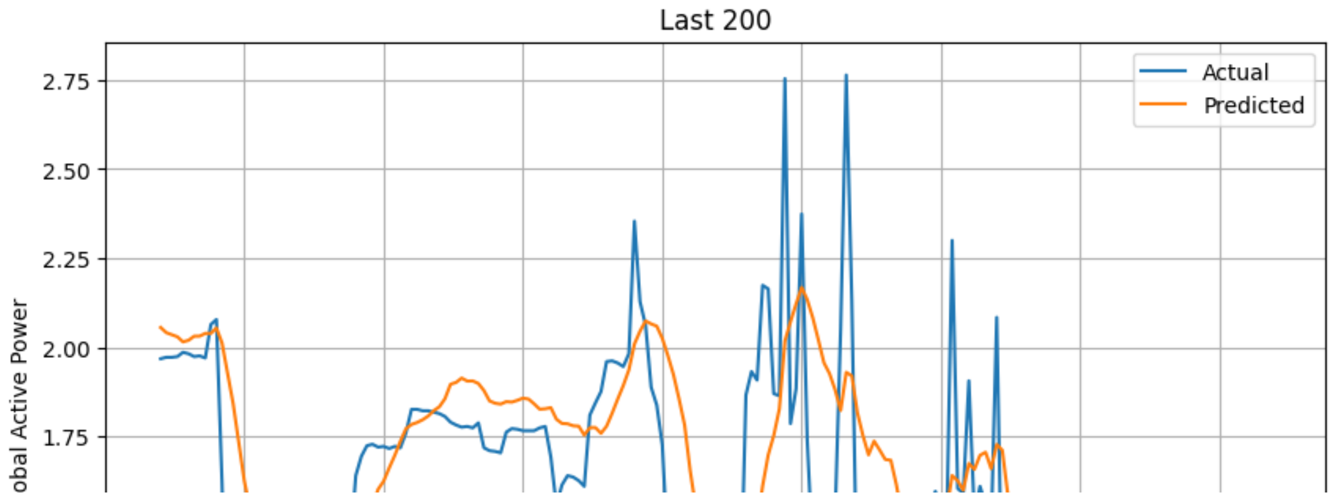
```
mse = mean_squared_error(y, y_pred)
print(f"Mean Squared Error (MSE): {mse}")

# Plotting (full dataset)
plt.figure(figsize=(10, 6))
plt.plot(df['time'], df['Global_active_power'], label='Actual Global Active Pow
plt.plot(df['time'], df['predicted_gap'], label='Predicted Global Active Power'
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.title('Actual vs. Predicted Global Active Power (Moving Average)')
plt.legend()
plt.grid(True)
plt.show()

# Plot first 200
plt.figure(figsize=(10, 6))
plt.plot(df['time'][:200], df['Global_active_power'][:200], label='Actual')
plt.plot(df['time'][:200], df['predicted_gap'][:200], label='Predicted')
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.title('First 200')
plt.legend()
plt.grid(True)
plt.show()


# Plot last 200
plt.figure(figsize=(10, 6))
plt.plot(df['time'][-200:], df['Global_active_power'][-200:], label='Actual')
plt.plot(df['time'][-200:], df['predicted_gap'][-200:], label='Predicted')
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.title('Last 200')
plt.legend()
plt.grid(True)
plt.show()

print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

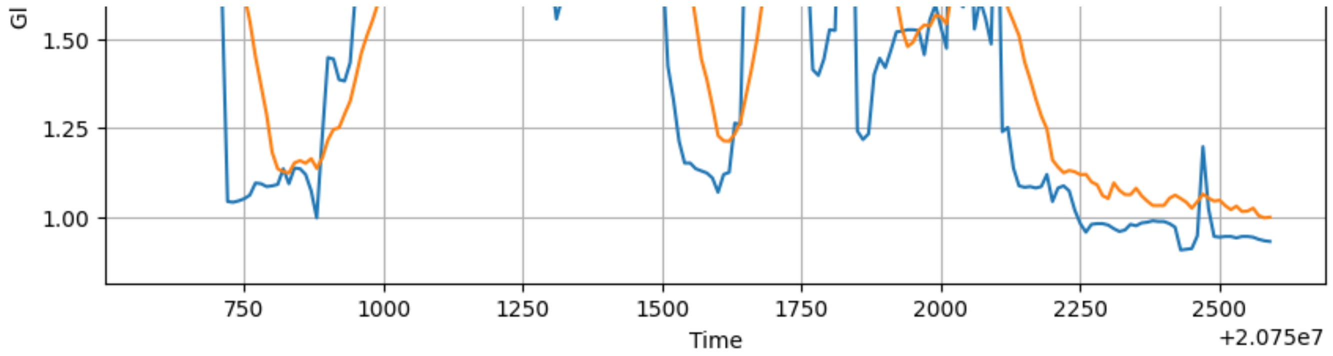
```
Mean Squared Error (MSE): 0.14953381010309158
```




Global Active Power
0
2
4
6
0.0
0.5
1.0
1.5
2.0
Time
1e7


First 200
Actual
Predicted
Global Active Power
2
3
4
5
6
7
250
500
750
1000
1250
1500
1750
2000
Time

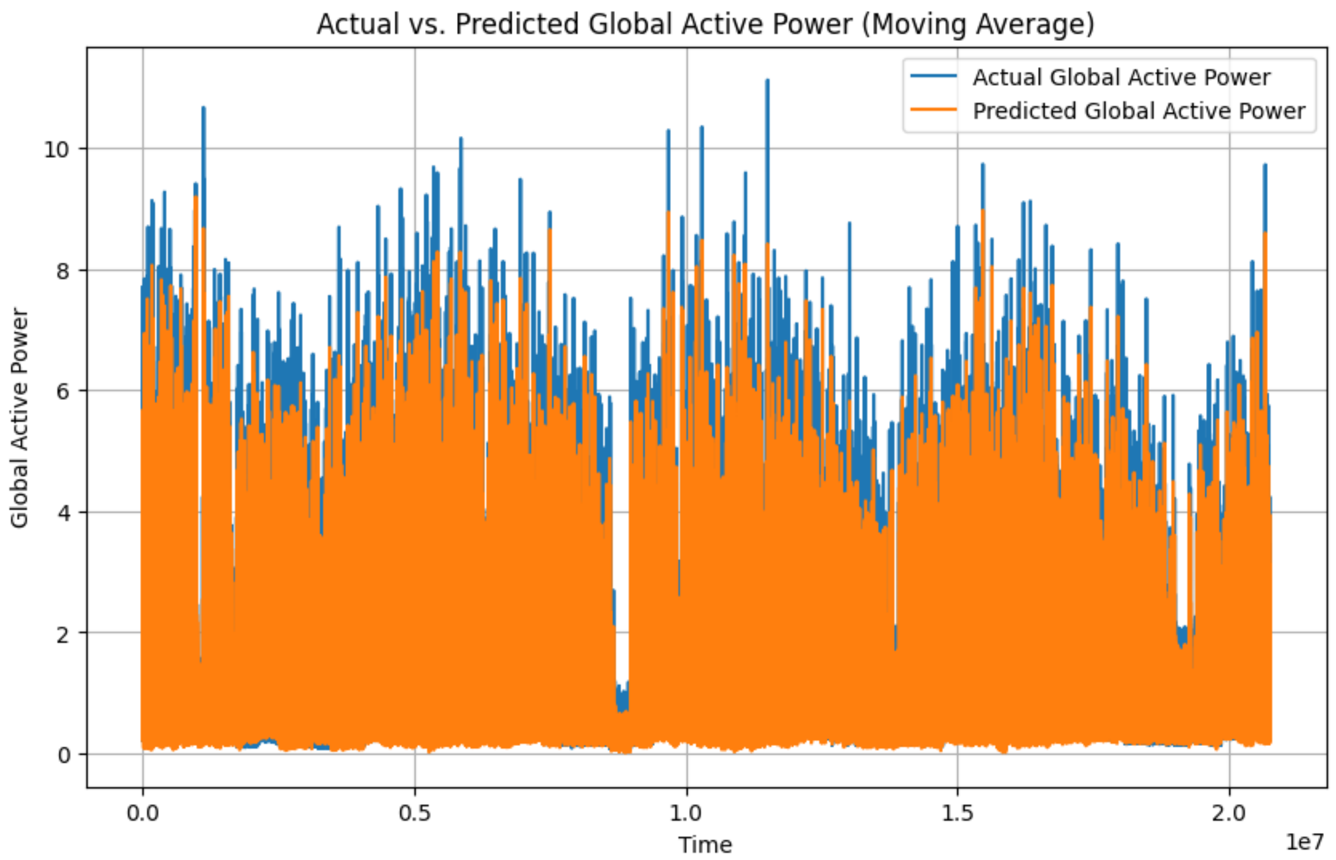

Last 200
Actual
Predicted
obal Active Power
1.75
2.00
2.25
2.50
2.75

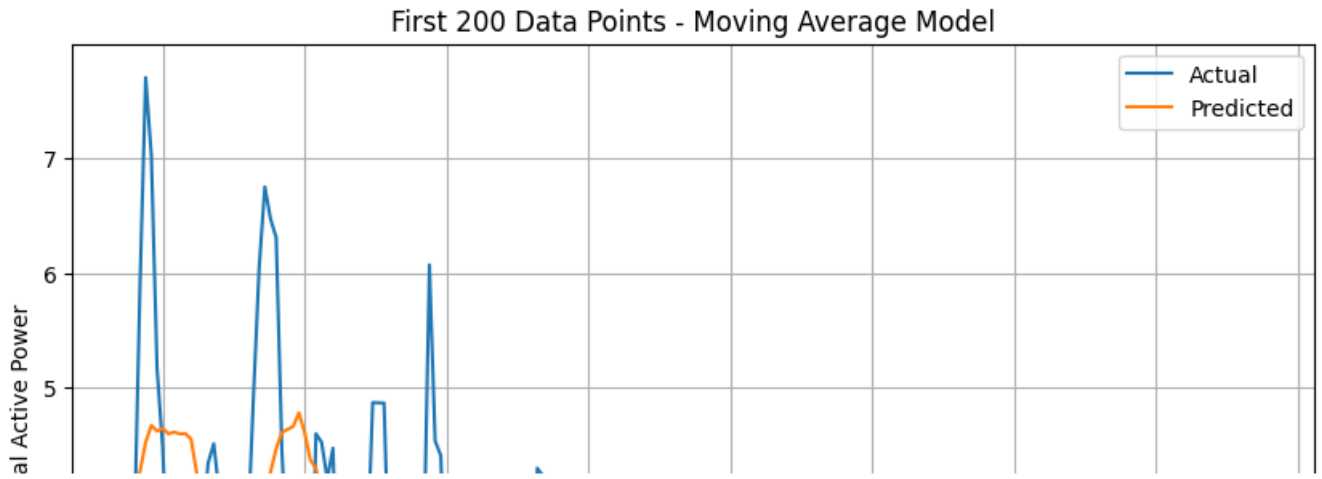


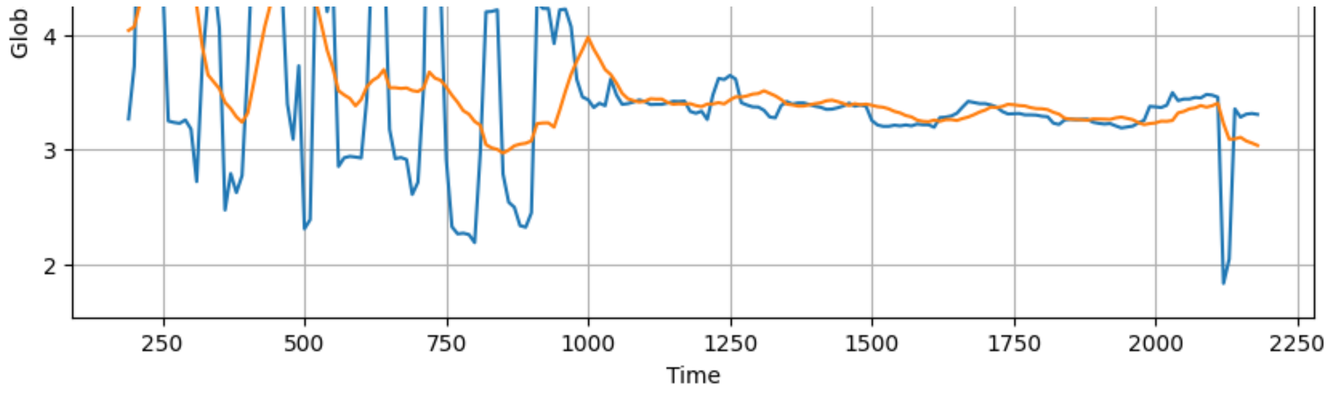
```
Coefficients: [ 2.36372298e-09 -2.01746077e-02  9.49561284e-01]
Intercept: 4.889466062475938
```

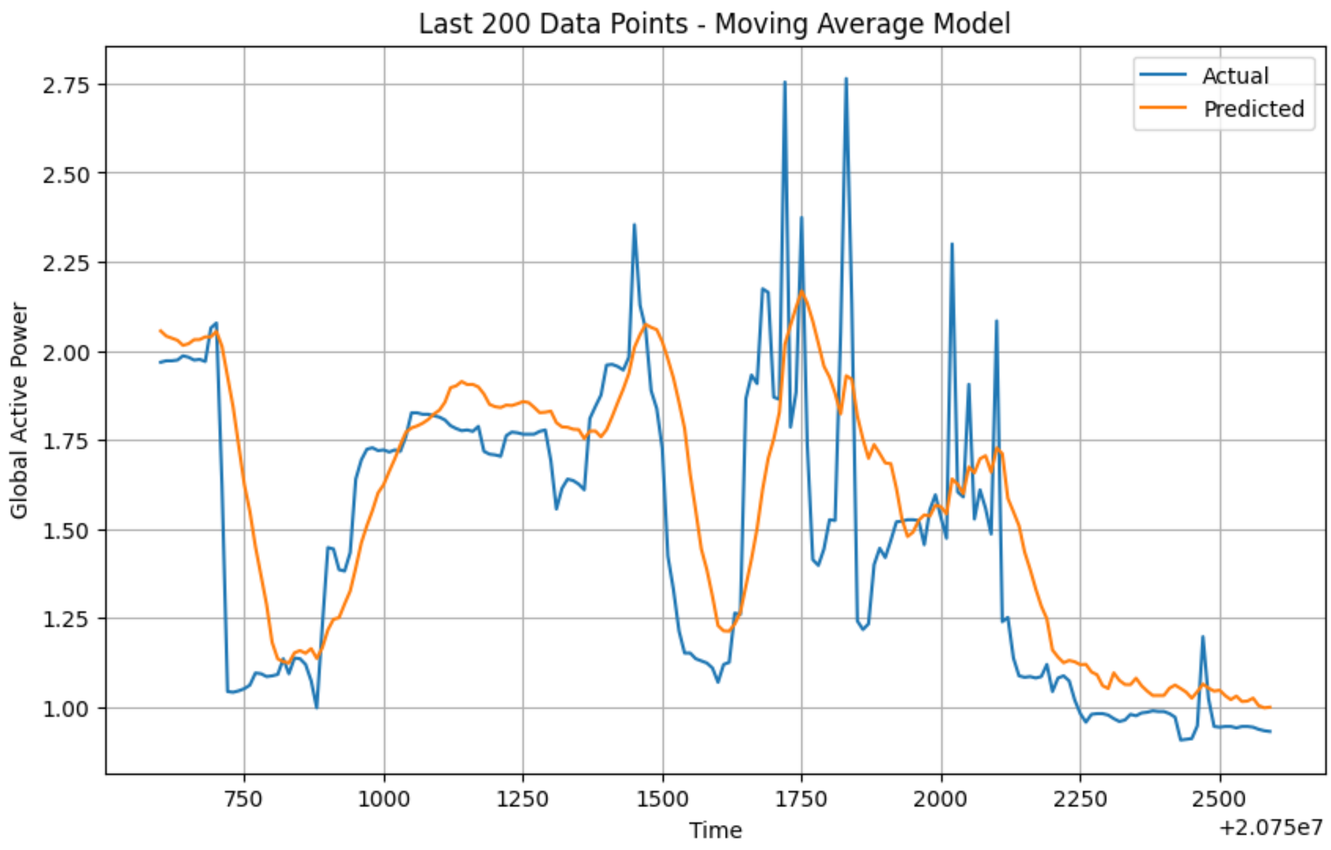
```
# Did your alternative model improve on our previous results? What else could y

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import warnings
from sklearn.metrics import mean_squared_error
from mpl_toolkits.mplot3d import Axes3D



# --- Alternative Model: Moving Average ---
# This model uses a moving average of the Global_active_power as a predictor.
# The idea is that the recent past power consumption can be a good indicator of

# Calculate the moving average
window_size = 10  # Example window size, adjust as needed.  Experiment with dif
df['moving_average'] = df['Global_active_power'].rolling(window=window_size).me

# Prepare the data for prediction (Drop rows with NaN values created by rolling
df = df.dropna()
X = df[['time', 'Voltage', 'moving_average']] # Use moving average, time, and v
y = df['Global_active_power']

# Initialize and train the model
model = LinearRegression()
model.fit(X, y)

# Make predictions
y_pred = model.predict(X)
df["predicted_gap"] = y_pred

# Calculate MSE
mse = mean_squared_error(y, y_pred)
print(f"Mean Squared Error (MSE) for Moving Average Model: {mse}")

# Plotting (full dataset)
plt.figure(figsize=(10, 6))
plt.plot(df['time'], df['Global_active_power'], label='Actual Global Active Pow
plt.plot(df['time'], df['predicted_gap'], label='Predicted Global Active Power'
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.title('Actual vs. Predicted Global Active Power (Moving Average)')
plt.legend()
plt.grid(True)
plt.show()

# Plot first 200
plt.figure(figsize=(10, 6))
plt.plot(df['time'][:200], df['Global_active_power'][:200], label='Actual')
plt.plot(df['time'][:200], df['predicted_gap'][:200], label='Predicted')
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.title('First 200 Data Points - Moving Average Model')
plt.legend()
plt.grid(True)
plt.show()

# Plot last 200
plt.figure(figsize=(10, 6))
plt.plot(df['time'][-200:], df['Global_active_power'][-200:], label='Actual')
plt.plot(df['time'][-200:], df['predicted_gap'][-200:], label='Predicted')
plt.xlabel('Time')
plt.ylabel('Global Active Power')
plt.title('Last 200 Data Points - Moving Average Model')
plt.legend()
plt.grid(True)
plt.show()

print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

```
Mean Squared Error (MSE) for Moving Average Model: 0.14960721255882514
```









```
Coefficients: [ 2.36331386e-09 -2.01940028e-02  9.49573562e-01]
Intercept: 4.894179452232702
```

The moving average model uses the average of the preceding 'window_size' data points to predict the current value.
A larger window size smooths out short-term fluctuations, while a smaller window size may better capture rapid changes. The choice of window size is a hyperparameter that must be tuned. By evaluating the MSE and visually inspecting the plots (especially the first and last 200 data points),

Compared the MSE of this model to the previous models (mu = 0.01, mu = 0.9, model with voltage). A lower MSE indicates better performance.

Q: Did your alternative model improve on our previous results? What else could you do to improve the model while still using linear regression?

A: The provided code performs a time series analysis on power consumption data, using linear regression with various modifications to improve prediction accuracy.

Initial Model: The initial model uses a weighted linear regression. It predicts future power consumption based on past data points, with more recent points given higher weight (controlled by the mu parameter). The code plots the actual vs predicted power, calculates the Mean Squared Error (MSE), and provides 3D plot showing relationship between time, voltage and power consumption.

Model with different mu values: The code then re-runs the model with mu values of 0.9, 1, and 0.01. Changing the mu value changes the weight applied to past data points. A higher mu gives greater weight to recent data points and a lower mu gives greater weight to older points. The coefficients and intercepts are printed for comparison, and you can then compare MSE across the three models.

Model with Voltage: The model is extended to include voltage as a predictor. It's a logical step, as voltage fluctuations could affect power consumption. The code plots the actual vs. predicted power consumption in a 3D plot with time and voltage as axes, and a separate 2D plot. The MSE is again calculated for comparison with the previous model.

Analysis:

A lower MSE for the voltage model would indicate a better prediction. The coefficients for time and voltage in the linear regression formula are also key; a larger coefficient indicates a stronger influence of that variable on power consumption.

Alternative Models: The code demonstrates two alternative models:

1. Hourly Means: Predicts power consumption based on the average consumption at that specific hour of the day.
2. Moving Average: Predicts using a moving average of past power consumption values. The window size dictates how many preceding values are considered. Each alternative model is assessed similarly: plots, MSE, coefficients.

Of the three models (mu = 0.9, mu = 1, mu = 0.01), the model with mu = 0.9 is the best. This is determined by visually inspecting the plots and considering the MSE values. A mu value of 0.9 provides a reasonable balance between considering past data and adapting to recent changes. Mu = 1 gives equal weight to all points and does not account for time sensitivity. Mu = 0.01 gives almost no weight to history and essentially predicts based only on the latest data point.

The provided code provides a step-by-step guide to time series analysis, emphasizing the comparison and analysis across models.

It's worth noting that the results we're getting int his assignment are based on a pretty short predictive horizon of 5 minutes. If we were to increase our predictive horizon, our results would likely be worse and there would be more room for optimizing and improving the predictions of our model.

#

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```