

# **Question 3: Propose and carry out additional analysis using the dataset given.**

## **Quick summary of what you would expect in this report.**

In this report, we want to predict a household's future gas consumption based on that household's past consumption. We first broke down the problem into two subtasks.

The first subtask was finding a good model that could accurately forecast household gas consumption. We already had a forecast model that we created in question 2. However, we were not totally delighted with its accuracy performance. So we came up with two ideas. The first idea was adding more useful attributes to the dataset with the belief that this would improve the accuracy of the model. More specifically, we added some weather attributes to the dataset, created a new linear regression and neural network model. There was not much improvement to the accuracy. Section 2 will discuss this in detail. Our second idea was to create a Long Short-Term Memory (LSTM) network model. There was noticeable increase in accuracy. Section 3 will talk about this in much detail.

The second subtask was to create a tool that can retrieve the gas forecast from our chosen model and provide this forecast to the respective households. We didn't wanted to simply stop at building a good forecast model and wanted to take a step further to demonstate what a gas provider can then do with the gas forecast model that can benefit both the gas provider and the households. So, we have created a telegram bot from scratch, just for this project, that can provide various services to residents such as sending households their respective gas consumption forecast, from our chosen model. Details about the bot will be discussed in section 4 and in the video presentation. In fact, our video presentation will focus more about the telegram bot. In addition, we have attach the code used to create the telegram bot under the "Python Telegram Bot" zip file.

We then finally ended the report with some remarks, reflections and conclusion. This can be seen in section 5.

## **Table Of Contents**

- Section 1: Introduction
- Section 2: Idea 1 - Adding new features for better accuracy
- Section 3: Idea 2 - Using LSTM models for better accracy
- Section 4: Details about our telegram bot that provides the gas forecast to the respective households
- Section 5: Quick summary, closing remarks and reflections

## **Section 1: Introduction**

This section will cover the problem we plan to address, why it is important, the analysis we have conducted and lastly the app that we have created to fix the problem.

## **Section 1.1: Problem description**

The problem that we would be addressing in this report is the prediction of gas consumption for each household based on their past consumption.

## **Section 1.2: Significance of the problem**

Consumers: For consumers, being able to get accurate prediction of their future gas consumption from the comfort of their homes is advantageous in many ways. Firstly, it allows them to monitor their gas consumption regularly and plan their future consumption accordingly. Overtime, consumers will become more conscious of their own gas expenditure which in turn would lead to a decrease in the usage of gas and subsequently their gas bill. Secondly, consumers will also be able to check for any sudden unexpected surge which could be due to faulty meters or gas leaks. In such cases, this allows them to quickly notify the gas companies which would save costs and time.

Gas Provider: For gas providers, by being able to forecast consumers' gas consumption, they are able to adjust the supply of the gas with respect to the demand. This would greatly benefit them as this would allow them to reduce the load on the meters when there is less demand and better utilize the resources accordingly. In addition, using the insights obtained from the forecast, gas providers would be able to perform demand response measures to change consumer patterns to better balance the demand for gas throughout the day. For instance, if the forecast projects that more people are consuming gas during the peak hours, a gas provider might come up with campaigns such as discounts and flexible pricing to encourage more people to consume gas during off-peak hours, which would help to even out gas consumption throughout the day. With this, gas providers will not have to drastically change the amount of gas they are producing throughout the day, which would help them save costs, reduce energy and emissions and better protect the equipments which used to fail or require frequent maintenance in the past due to haphazard production levels.

## **Section 1.3: Our proposed idea**

We will first aim to create a good forecasting model that would be able to accurately predict gas consumption for different households. Then, we will aim to implement a tool that would be able to use this model to effectively provide future gas consumption forecast to the respective households in a convenient manner.

## **Section 1.4: Breakdown of problem**

As can be seen easily, the problem that we intend to solve can be broken down into 2 subtasks. They are: building a good model that can accurately predict household gas consumption and creating a tool that can then utilize the built model to provide the forecast to the respective households. Below, we will briefly discuss these subtasks in detail.

## **Section 1.5: Subtask 1 - Model that can accurately predict household gas consumption**

It is very important to have a good model as without that, we will not be able to accurately predict future gas consumptions and there would be no point providing the inaccurate forecast

to customers.

Now to quickly recap, in question 2, we built a linear regression and support vector machine forecast model for each household that utilized additional features that we created such as the whether the date is a holiday and etc. Both the models had decent accuracy. Here, we will aim to do more analysis and explore more models to try to come out with a model that achieves better accuracy. Specifically, we will be exploring two ideas. The first idea is to include more features that influence household gas consumption in our model with the belief that this will help us find a better correlation between the attributes and household gas consumption. We will discuss this further in section 2. The second idea is to use Long Short-Term Memory network (LSTM) for our gas consumption prediction. LSTM models are commonly regarded suitable for learning and forecasting time series data and hence are suitable for our problem. We will discuss more about this in section 3.

## **Section 1.6: Subtask 2 - Tool to provide the forecast to the respective households**

After much research and discussion, we have decided to build a telegram bot from scratch that will send the gas consumption forecast to the respective households. Our bot will make use of the information obtained from the chosen model for a particular household to provide gas consumption forecast for various time periods for that respective household. More information about this will be discussed in section 4.

---

## **Section 2: Building a better forecast model - Adding new features for better accuracy**

In this section, we will explore our first idea to create a better model, which is adding new features. Specifically, we will be discussing how we added two new features, daily average temperature and precipitation, in the data to aid in our prediction.

To briefly summarize, we first built a linear regression model based on the two new attributes in addition to the other attributes that we used in question 2. We then also built a neural network with the same attributes used in the linear regression model to compare both model's accuracy. At the end of the section we made some remarks based on our observations.

### **Section 2.1: Motivation for finding new attributes**

The use of suitable attribute(s) can undoubtedly improve the accuracy of the model which would in turn make the model more robust. That being said, having too many attributes can increase the training time or even lead to overfitting. Worst, if redundant attributes are used, it can result in a model with bad performance. Hence, choosing the right attributes is important.

With that in mind, we first researched and made a list of factors that we thought, greatly influence household gas consumption. From that list, we shortlisted a few such as family size and weather. We then browsed through the internet to find suitable data for those attributes. Unfortunately, we were either unable to find such data or the data we found was incomplete or inaccurate. The only suitable data we were able to find was weather data. This weather data was obtained from one of the weather stations in Texas (near to Pecan Street) from the [U.S. National](#)

Centers for Environmental Information website. It contains various weather details like minimum, maximum temperature, precipitation and etc.

## Section 2.2: Weather's influence on gas consumption

In the US, natural gas is commonly used to heat houses. And it is common for gas consumption to go up in the winter when it is very cold outside and there is a need to warm houses. The same can also be said whenever there is strong rain as rain causes temperature to drop as well. In addition, during these periods, many households may decide to stay at home for a prolonged period which might also increase gas consumption. And lastly, more energy would be required to heat up to a certain temperature during these cold periods as compared to other times, which would also drive consumption up. Hence, we felt that average temperature and precipitation can be two new features that can be added to the dataset for better prediction.

## Section 2.3: Algorithm approach

Below is a quick summary of the steps involved.

1. Process raw weather data and obtain the daily average temperature and precipitation attributes.
2. Add average temperature and precipitation attributes to the dataset.
3. Build a linear regression model and evaluate its accuracy.
4. Build a neural network and evaluate its accuracy.

## Section 2.4: Code

In [2]:

```
# Libraries used
import os
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import datetime as dt
import numpy as np

from datetime import datetime
from functools import reduce
from sklearn.metrics import mean_squared_error as mse
from tqdm.auto import tqdm
from tqdm import tqdm
from copy import deepcopy
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout

plt.rcParams.update({'figure.max_open_warning': 0})
```

```
pd.options.mode.chained_assignment = None # default='warn'
pd.set_option('display.max_rows', 157)
```

## 1. Process raw weather data and obtain the daily average temperature and precipitation attributes

In [3]:

```
# Load the dataset (that we used for question 2)
df = pd.read_csv('Question_2_dataset.csv', index_col='localminute', parse_dates=True)
df
```

Out[3]:

	dataid	meter_value	hours_passed	weekday	is_holiday	is_weekday	is_office_ho
localminute							
2015-10-01 05:00:00+00:00	739.0	88858.000000	0	3	0	1	
2015-10-01 06:00:00+00:00	739.0	88858.000000	1	3	0	1	
2015-10-01 07:00:00+00:00	739.0	88860.000000	2	3	0	1	
2015-10-01 08:00:00+00:00	739.0	88860.000000	3	3	0	1	
2015-10-01 09:00:00+00:00	739.0	88860.000000	4	3	0	1	
...	...	...	...	...	...	...	...
2016-03-08 12:00:00+00:00	4874.0	314452.311622	779	1	0	1	
2016-03-08 13:00:00+00:00	4874.0	314461.233716	780	1	0	1	
2016-03-08 14:00:00+00:00	4874.0	314470.155811	781	1	0	1	
2016-03-08 15:00:00+00:00	4874.0	314479.077905	782	1	0	1	
2016-03-08 16:00:00+00:00	4874.0	314488.000000	783	1	0	1	

610632 rows × 12 columns

In [4]:

```
# Load the raw weather dataset
rawWeatherData = pd.read_csv('WeatherData.csv')
rawWeatherData.head()
```

Out[4]:

	STATION	NAME	LATITUDE	LONGITUDE	ELEVATION	DATE	DAPR	MDPR	PRCP	S
0	US1TXTV0059	TANGLEWOOD FOREST 0.6 NE, TX US	30.1807	-97.8315	241.1	2015-10-01	NaN	NaN	0.0	
1	US1TXTV0059	TANGLEWOOD FOREST 0.6 NE, TX US	30.1807	-97.8315	241.1	2015-10-02	NaN	NaN	0.0	

	STATION	NAME	LATITUDE	LONGITUDE	ELEVATION	DATE	DAPR	MDPR	PRCP	\$
<b>2</b>	US1TXTV0059	TANGLEWOOD FOREST 0.6 NE, TX US	30.1807	-97.8315	241.1	2015-10-03	NaN	NaN	0.0	
<b>3</b>	US1TXTV0059	TANGLEWOOD FOREST 0.6 NE, TX US	30.1807	-97.8315	241.1	2015-10-04	NaN	NaN	0.0	
<b>4</b>	US1TXTV0059	TANGLEWOOD FOREST 0.6 NE, TX US	30.1807	-97.8315	241.1	2015-10-05	NaN	NaN	0.0	

In [5]:

```
# obtain data from closest weather station to pecan street
rawWeatherData = rawWeatherData[rawWeatherData.NAME=='AUSTIN CAMP MABRY, TX US']

# only interested in obtaining temperature and precipitation attributes
rawWeatherData = rawWeatherData[['DATE','PRCP','SNOW','TMAX','TMIN']]

# compute average temperature from minimum and maximum temperature
rawWeatherData['TAVG']=(rawWeatherData['TMAX']+rawWeatherData['TMIN'])/2

rawWeatherData
```

Out[5]:

	DATE	PRCP	SNOW	TMAX	TMIN	TAVG
<b>6666</b>	2015-10-01	0.0	0.0	34.4	20.6	27.50
<b>6667</b>	2015-10-02	0.0	0.0	31.7	19.4	25.55
<b>6668</b>	2015-10-03	0.0	0.0	30.6	13.3	21.95
<b>6669</b>	2015-10-04	0.0	0.0	30.6	15.0	22.80
<b>6670</b>	2015-10-05	0.0	0.0	29.4	19.4	24.40
...	...	...	...	...	...	...
<b>6844</b>	2016-03-27	0.0	0.0	23.3	13.3	18.30
<b>6845</b>	2016-03-28	0.0	0.0	23.9	8.9	16.40
<b>6846</b>	2016-03-29	0.5	0.0	20.6	14.4	17.50
<b>6847</b>	2016-03-30	0.8	0.0	25.0	20.0	22.50
<b>6848</b>	2016-03-31	0.0	0.0	30.0	18.9	24.45

183 rows × 6 columns

Note that the weather data has already been filtered to be between October 2015 & Mar 2016. Since we will be using the hourly meter readings that we used in question 2 to build the models, we will need hourly average temperature and precipitation attributes. However, we don't have them. Hence, for each hour in a day, we will simply use the obtained daily average temperature and precipitation for that day.

In [9]:

```
# weather data has already been filtered to be between October 2015 & Mar 2016
# but need to create entries for each hour
filteredWeatherData = pd.DataFrame(np.repeat(rawWeatherData.values,24, axis=0))
```

```
filteredWeatherData.columns = ["DateTime","Precipitation","Snowfall","Min temperature",
                               "Max temperature","Daily average temperature"]
filteredWeatherData.head()
```

Out[9]:

	<b>DateTime</b>	<b>Precipitation</b>	<b>Snowfall</b>	<b>Min temperature</b>	<b>Max temperature</b>	<b>Daily average temperature</b>
<b>0</b>	2015-10-01	0.0	0.0	34.4	20.6	27.5
<b>1</b>	2015-10-01	0.0	0.0	34.4	20.6	27.5
<b>2</b>	2015-10-01	0.0	0.0	34.4	20.6	27.5
<b>3</b>	2015-10-01	0.0	0.0	34.4	20.6	27.5
<b>4</b>	2015-10-01	0.0	0.0	34.4	20.6	27.5

In [12]:

```
# obtain hour entries column
x = df[df.dataid==35.0]
xdtimes = x.index

# add it to the weather data
filteredWeatherData['DateTime'] = xdtimes
filteredWeatherData
```

Out[12]:

	<b>DateTime</b>	<b>Precipitation</b>	<b>Snowfall</b>	<b>Min temperature</b>	<b>Max temperature</b>	<b>Daily average temperature</b>
<b>0</b>	2015-10-01 05:00:00+00:00	0.0	0.0	34.4	20.6	27.5
<b>1</b>	2015-10-01 06:00:00+00:00	0.0	0.0	34.4	20.6	27.5
<b>2</b>	2015-10-01 07:00:00+00:00	0.0	0.0	34.4	20.6	27.5
<b>3</b>	2015-10-01 08:00:00+00:00	0.0	0.0	34.4	20.6	27.5
<b>4</b>	2015-10-01 09:00:00+00:00	0.0	0.0	34.4	20.6	27.5
...	...	...	...	...	...	...
<b>4387</b>	2016-04-01 00:00:00+00:00	0.0	0.0	30.0	18.9	24.45
<b>4388</b>	2016-04-01 01:00:00+00:00	0.0	0.0	30.0	18.9	24.45
<b>4389</b>	2016-04-01 02:00:00+00:00	0.0	0.0	30.0	18.9	24.45
<b>4390</b>	2016-04-01 03:00:00+00:00	0.0	0.0	30.0	18.9	24.45
<b>4391</b>	2016-04-01 04:00:00+00:00	0.0	0.0	30.0	18.9	24.45

4392 rows × 6 columns

In [13]:

```
# save it as csv  
filteredWeatherData.to_csv('processed_climate_data.csv', index=False)
```

In [14]:

```
weather_df = filteredWeatherData  
weather_df['localminute'] = pd.to_datetime(weather_df.DateTime, format = '%Y-%m-%d %H:%M')  
weather_df
```

Out[14]:

	DateTime	Precipitation	Snowfall	Min temperature	Max temperature	Daily average temperature	localminute
0	2015-10-01 05:00:00+00:00	0.0	0.0	34.4	20.6	27.5	2015-10-01 05:00:00+00:00
1	2015-10-01 06:00:00+00:00	0.0	0.0	34.4	20.6	27.5	2015-10-01 06:00:00+00:00
2	2015-10-01 07:00:00+00:00	0.0	0.0	34.4	20.6	27.5	2015-10-01 07:00:00+00:00
3	2015-10-01 08:00:00+00:00	0.0	0.0	34.4	20.6	27.5	2015-10-01 08:00:00+00:00
4	2015-10-01 09:00:00+00:00	0.0	0.0	34.4	20.6	27.5	2015-10-01 09:00:00+00:00
...	...	...	...	...	...	...	...
4387	2016-04-01 00:00:00+00:00	0.0	0.0	30.0	18.9	24.45	2016-04-01 00:00:00+00:00
4388	2016-04-01 01:00:00+00:00	0.0	0.0	30.0	18.9	24.45	2016-04-01 01:00:00+00:00
4389	2016-04-01 02:00:00+00:00	0.0	0.0	30.0	18.9	24.45	2016-04-01 02:00:00+00:00
4390	2016-04-01 03:00:00+00:00	0.0	0.0	30.0	18.9	24.45	2016-04-01 03:00:00+00:00
4391	2016-04-01 04:00:00+00:00	0.0	0.0	30.0	18.9	24.45	2016-04-01 04:00:00+00:00

4392 rows × 7 columns

In [15]:

```
# set Localminute as index so that can later merge with hourly meter readings  
weather_df.set_index('localminute', inplace=True)  
weather_df
```

Out[15]:

localminute	DateTime	Precipitation	Snowfall	Min temperature	Max temperature	Daily average temperature
2015-10-01 05:00:00+00:00	2015-10-01 05:00:00+00:00	0.0	0.0	34.4	20.6	27.5
2015-10-01 06:00:00+00:00	2015-10-01 06:00:00+00:00	0.0	0.0	34.4	20.6	27.5

	<b>DateTime</b>	<b>Precipitation</b>	<b>Snowfall</b>	<b>Min temperature</b>	<b>Max temperature</b>	<b>Daily average temperature</b>
<b>localminute</b>						
<b>2015-10-01 07:00:00+00:00</b>	2015-10-01 07:00:00+00:00	0.0	0.0	34.4	20.6	27.5
<b>2015-10-01 08:00:00+00:00</b>	2015-10-01 08:00:00+00:00	0.0	0.0	34.4	20.6	27.5
<b>2015-10-01 09:00:00+00:00</b>	2015-10-01 09:00:00+00:00	0.0	0.0	34.4	20.6	27.5
...	...	...	...	...	...	...
<b>2016-04-01 00:00:00+00:00</b>	2016-04-01 00:00:00+00:00	0.0	0.0	30.0	18.9	24.45
<b>2016-04-01 01:00:00+00:00</b>	2016-04-01 01:00:00+00:00	0.0	0.0	30.0	18.9	24.45
<b>2016-04-01 02:00:00+00:00</b>	2016-04-01 02:00:00+00:00	0.0	0.0	30.0	18.9	24.45
<b>2016-04-01 03:00:00+00:00</b>	2016-04-01 03:00:00+00:00	0.0	0.0	30.0	18.9	24.45
<b>2016-04-01 04:00:00+00:00</b>	2016-04-01 04:00:00+00:00	0.0	0.0	30.0	18.9	24.45

4392 rows × 6 columns

## 2. Add average temperature and precipitation attributes to the dataset.

In [25]:

```
# merge weather dataframe to dataset
df_with_weather = pd.merge(df, weather_df[['Precipitation', 'Daily average temperature',
                                             on='localminute']])
df_with_weather.head()
```

Out[25]:

	<b>dataid</b>	<b>meter_value</b>	<b>hours_passed</b>	<b>weekday</b>	<b>is_holiday</b>	<b>is_weekday</b>	<b>is_office_hour</b>
<b>localminute</b>							
<b>2015-10-01 05:00:00+00:00</b>	739.0	88858.0	0	3	0	1	0
<b>2015-10-01 05:00:00+00:00</b>	8890.0	197164.0	0	3	0	1	0
<b>2015-10-01 05:00:00+00:00</b>	6910.0	179118.0	0	3	0	1	0
<b>2015-10-01 05:00:00+00:00</b>	3635.0	151330.0	0	3	0	1	0
<b>2015-10-01 05:00:00+00:00</b>	1507.0	390354.0	0	3	0	1	0

In [ ]:

```
df_with_weather.to_csv('dfweather.csv', index=True)
```

```
In [3]: df = pd.read_csv('dfweather.csv', index_col='localminute', parse_dates=True, date_parse=df
```

Out[3]:

	dataid	meter_value	hours_passed	weekday	is_holiday	is_weekday	is_office_hour
<b>localminute</b>							
2015-10-01 05:00:00+00:00	739.0	88858.0	0	3	0	1	0
2015-10-01 05:00:00+00:00	8890.0	197164.0	0	3	0	1	0
2015-10-01 05:00:00+00:00	6910.0	179118.0	0	3	0	1	0
2015-10-01 05:00:00+00:00	3635.0	151330.0	0	3	0	1	0
2015-10-01 05:00:00+00:00	1507.0	390354.0	0	3	0	1	0
...	...	...	...	...	...	...	...
2016-04-01 04:00:00+00:00	2945.0	161232.0	3767	4	0	1	0
2016-04-01 04:00:00+00:00	7016.0	313488.0	3767	4	0	1	0
2016-04-01 04:00:00+00:00	8967.0	189188.0	3767	4	0	1	0
2016-04-01 04:00:00+00:00	3310.0	405514.0	3767	4	0	1	0
2016-04-01 04:00:00+00:00	6673.0	85362.0	1829	4	0	1	0

610632 rows × 14 columns

### 3. Build a linear regression model and evaluate its accuracy.

Now that we have added the respective weather attributes to the dataset, we will build models and evaluate their performance. We will first build a simple linear regression model followed by a neural network and compare their accuracies.

For both the models, the attributes that we will be using are: 'is\_holiday', 'is\_weekday' and 'hours\_passed' from our question 2 (as they seemed to have correlation with the meter readings) and 'Precipitation' and 'Daily average temperature' attributes that we just obtained.

Also, we will be using the mean squared error (MSE) as our metric. This is because we want to check how close the forecasts are to actual values and penalize large outliers more heavily as compared to small ones.

These are the functions that we will be using to build both the linear regression and the neural network model.

`plot_overall_timeseries` creates a timeseries plot with the actual meter values we used for training (denoted in blue), the actual meter values we used for testing (denoted in red) as well as the predicted meter values we derived from our model (denoted in orange). In each plot, we also calculated the Mean Squared Error derived from the testing dataset for that particular household, and noted it down in the title of each plot.

`plot_testing_scatter` creates a scatter plot with the actual meter values we used for testing versus the predicted meter values we derived from our model (denoted in red) as well as the line for goodness of fit that indicates a perfect prediction (denoted in green). In each plot, we also calculated the Mean Squared Error derived from the testing dataset for that particular household, and noted it down in the title of each plot.

`train_each_house_lr` creates the linear regression model and trains the model for each individual household based on the features provided. After training, the plots for each household are displayed over their respective time period.

`train_each_house_nn` creates the neural network model and trains the model for each individual household based on the features provided. This function also scales and normalizes the values before fitting them to the model for training. After training, the predicted values are rescaled back to their original form and the plots for each household are displayed over their respective time period.

#### **Additional Details about attributes used from question 2 (extracted from our question 2 for reference)**

The three attributes that we will be using from question 2 are 'is\_holiday', 'is\_weekday' and 'hours\_passed'. We used these 3 features as we observed a non-linear trend in the meter value for most households. We incorporated a boolean variable denoting if the particular day is a holiday and another boolean variable denoting if the particular day is a weekday. Our rationale is that households are likely to be at home if it is a weekend or possibly a holiday whereas most people would not be at home during working hours on a weekday.

#### **Additional Details about training (extracted from our question 2 for reference)**

For our training, we took the month of January as the dataset. We used the first 26 days of January as the training dataset and the last 5 days to test the model. We observed that this smaller dataset provided a higher level of accuracy compared to training on the overall dataset. This is because we wanted to predict the next hours' consumption and including datapoints from older consumptions led to more noise during training. Hence the gas company should predict the hourly readings using more relevant readings from a month ago till now.

**note :** There are two houses with data ID 4874 and 8967, which have incomplete data for this time period and hence we do not include them in our plots. Removing these two houses from the unique\_households is done below.

In [4]:

```
unique_households = df['dataid'].unique()
unique_households.sort()
unique_households = np.delete(unique_households, [83,140])
unique_households
```

Out[4]: array([ 35., 44., 77., 94., 114., 187., 222., 252., 370.,
 483., 484., 661., 739., 744., 871., 1042., 1086., 1103.,
1185., 1283., 1403., 1415., 1507., 1556., 1589., 1619., 1697.,

```

1714., 1718., 1790., 1791., 1792., 1800., 1801., 2018., 2034.,
2072., 2094., 2129., 2233., 2335., 2378., 2449., 2461., 2470.,
2575., 2638., 2645., 2755., 2814., 2818., 2945., 2946., 2965.,
2980., 3036., 3039., 3134., 3310., 3367., 3527., 3544., 3577.,
3635., 3723., 3778., 3849., 3893., 3918., 4029., 4031., 4193.,
4228., 4296., 4352., 4356., 4373., 4421., 4447., 4514., 4671.,
4732., 4767., 4998., 5129., 5131., 5193., 5275., 5317., 5395.,
5403., 5439., 5484., 5545., 5636., 5658., 5785., 5810., 5814.,
5892., 5972., 6101., 6412., 6505., 6578., 6673., 6685., 6830.,
6836., 6863., 6910., 7016., 7017., 7030., 7117., 7287., 7429.,
7460., 7566., 7674., 7682., 7739., 7741., 7794., 7900., 7919.,
7965., 7989., 8059., 8084., 8086., 8155., 8156., 8244., 8386.,
8467., 8703., 8829., 8890., 9052., 9121., 9134., 9160., 9278.,
9295., 9474., 9600., 9620., 9631., 9639., 9729., 9766., 9849.,
9956., 9982.])

```

In [108..

```

# for details about what the respective function do, please look above.

features = ['is_holiday', 'is_weekday', 'hours_passed', 'Precipitation', 'Daily aver
featureNmeter = ['is_holiday', 'is_weekday', 'hours_passed', 'Precipitation', 'Daily
hours_to_predict = 120
mse_lr = []
mse_nn = []

def plot_overall_timeseries(house_df, train_df, test_df, house):
    mean_squared_error = mse(test_df['predicted_meter_value'], test_df['meter_value']
    plt.figure(figsize=(12, 8))
    plt.scatter(train_df['localminute'], train_df['meter_value'], c='blue', label =
    plt.scatter(test_df['localminute'], test_df['meter_value'], c='red' , label =
    plt.scatter(test_df['localminute'], test_df['predicted_meter_value'], c='orange'
    plt.xlabel('Date')
    plt.ylabel('Gas Consumption')
    plt.title(f'Time Series Plot - Household {house} Testing MSE : {mean_squared_err
    plt.legend()
    return plt

def plot_testing_scatter(test_df, house):
    mean_squared_error = mse(test_df['predicted_meter_value'], test_df['meter_value']
    plt.figure(figsize=(12, 8))
    plt.plot(test_df['meter_value'], test_df['predicted_meter_value'], 'o' ,c='red'
    plt.plot(test_df['meter_value'], test_df['meter_value'], c='green', label = 'lin
    plt.xlabel('Actual Meter Value')
    plt.ylabel('Predicted Meter Value')
    plt.title(f' Scatter Plot - Household {house} Testing MSE : {mean_squared_error}
    plt.legend()
    return plt

def train_each_house_lr(hours_to_predict):
    for house in unique_households:
        # obtain house
        house_df = df.loc[df['dataid'] == house]
        house_df = house_df.reset_index()

        # we simply interpolate the hours with no data since meter_value is the cumu
        house_df = house_df[(house_df['localminute'] > "2016-01-01 00:00:00+00:00")]
        hourly_df_train = house_df[:hours_to_predict]
        hourly_df_test = house_df[-hours_to_predict:]

        # create the Linear regression model
        temp_model = make_pipeline(StandardScaler(), LinearRegression())
        temp_model.fit(hourly_df_train[features], hourly_df_train['meter_value'])

        # obtained predicted meter reading, compute mse and plot
        hourly_df_train['predicted_meter_value'] = temp_model.predict(hourly_df_trai
        hourly_df_test['predicted_meter_value'] = temp_model.predict(hourly_df_test[


```

```

house_df['predicted_meter_value'] = temp_model.predict(house_df[features])
mean_squared_error = mse(hourly_df_test['predicted_meter_value'], hourly_df_
mse_lr.append(mean_squared_error)
plot1 = plot_overall_timeseries(house_df, hourly_df_train, hourly_df_test, h
plot2 = plot_testing_scatter(hourly_df_test, house)
plt.show()

def train_each_house_nn(hours_to_predict):
    for house in unique_households:
        # obtain house
        house_df = df.loc[df['dataid'] == house]
        house_df = house_df.reset_index()

        # we simply interpolate the hours with no data since meter_value is the cumu
        house_df = house_df[(house_df['localminute'] > "2016-01-01 00:00:00+00:00")]
        hourly_df_train = house_df[:-hours_to_predict]
        hourly_df_test = house_df[-hours_to_predict:]

        # scale and normalize
        scaler = MinMaxScaler().fit(hourly_df_train[features])
        hourly_df_train[features] = scaler.transform(hourly_df_train[features])
        hourly_df_test[features] = scaler.transform(hourly_df_test[features])

        meter_train_df = hourly_df_train[['meter_value']]
        meter_test_df = hourly_df_test[['meter_value']]
        scaler1 = MinMaxScaler().fit(meter_train_df)
        meter_train_df = scaler1.transform(meter_train_df)
        meter_test_df = scaler1.transform(meter_test_df)

        # create the neural network model
        temp_model = Sequential()
        temp_model.add(Dense(40, input_dim=5, activation='relu'))
        temp_model.add(Dropout(0.2))
        temp_model.add(Dense(25, activation='relu'))
        temp_model.add(Dense(1, activation='linear'))

        # compile the keras model
        temp_model.compile(loss='mean_squared_error', optimizer='adam', metrics=['me

        # train the model
        temp_model.fit(hourly_df_train[features], meter_train_df, epochs=150, batch_)

        # obtained predicted meter reading, compute mse and plot

        val = temp_model.predict(hourly_df_train[features])
        hourly_df_train['predicted_meter_value'] = scaler1.inverse_transform(val)

        val1 = temp_model.predict(hourly_df_test[features])
        hourly_df_test['predicted_meter_value'] = scaler1.inverse_transform(val1)

        house_df['predicted_meter_value'] = temp_model.predict(house_df[features])

        mean_squared_error = mse(hourly_df_test['predicted_meter_value'], hourly_df_
mse_nn.append(mean_squared_error)
plot1 = plot_overall_timeseries(house_df, hourly_df_train, hourly_df_test, h
plot2 = plot_testing_scatter(hourly_df_test, house)
plt.show()

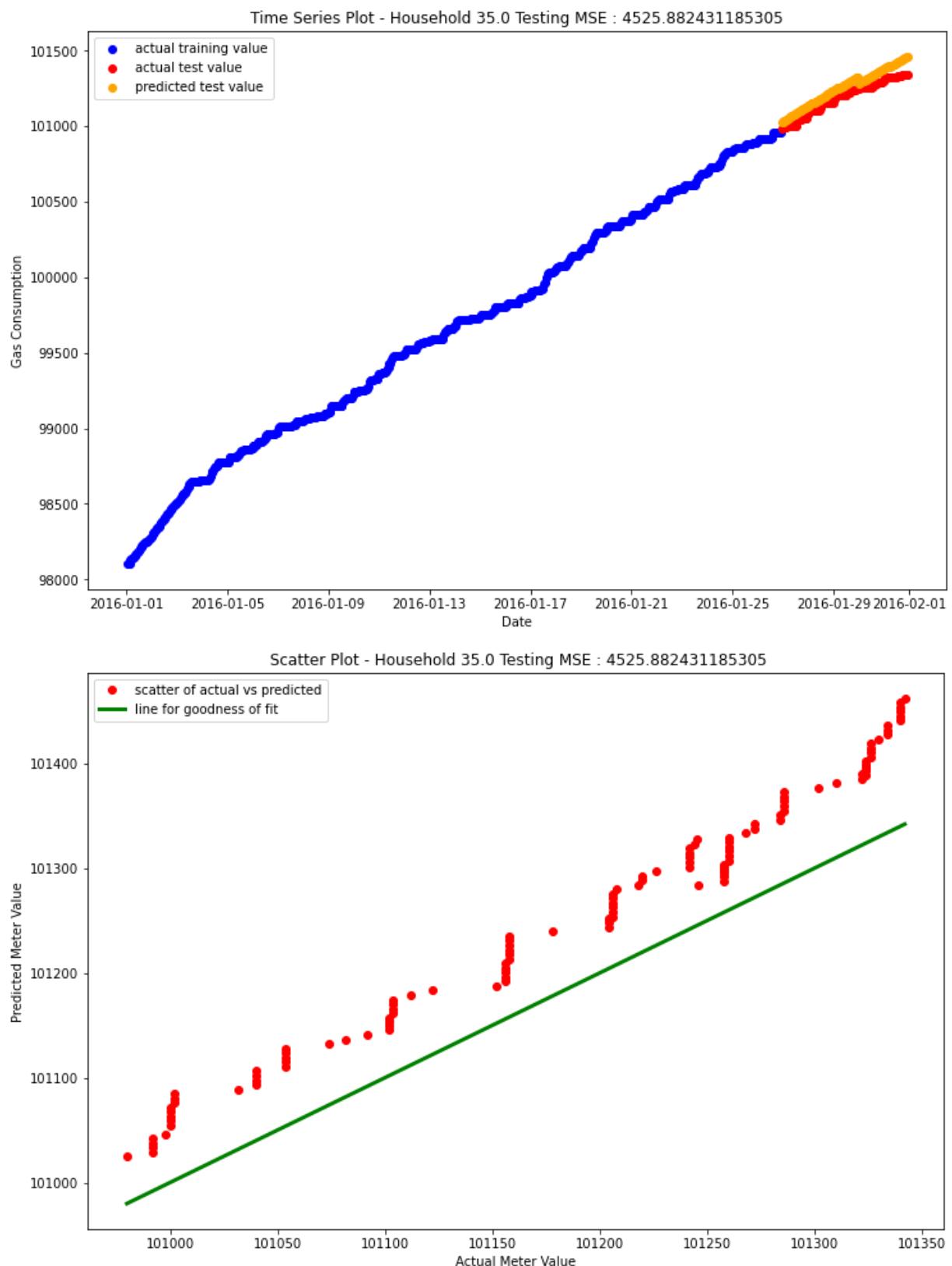
```

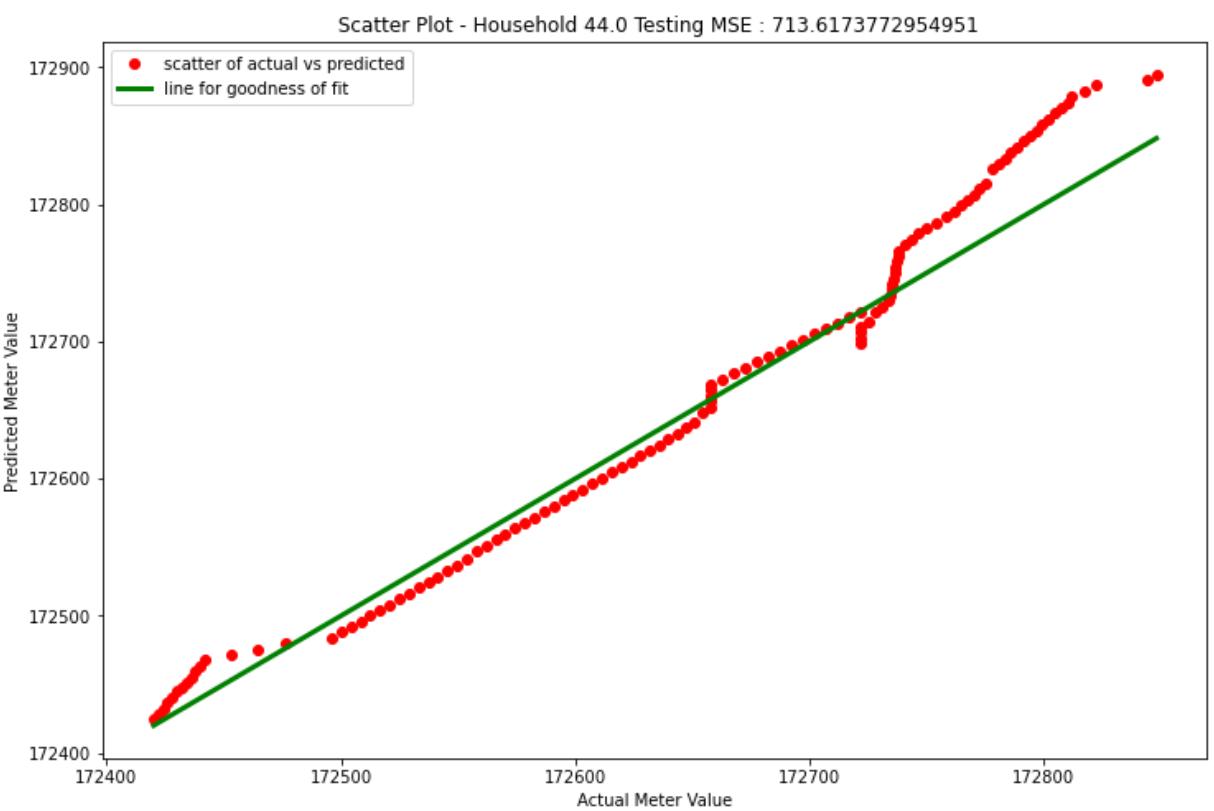
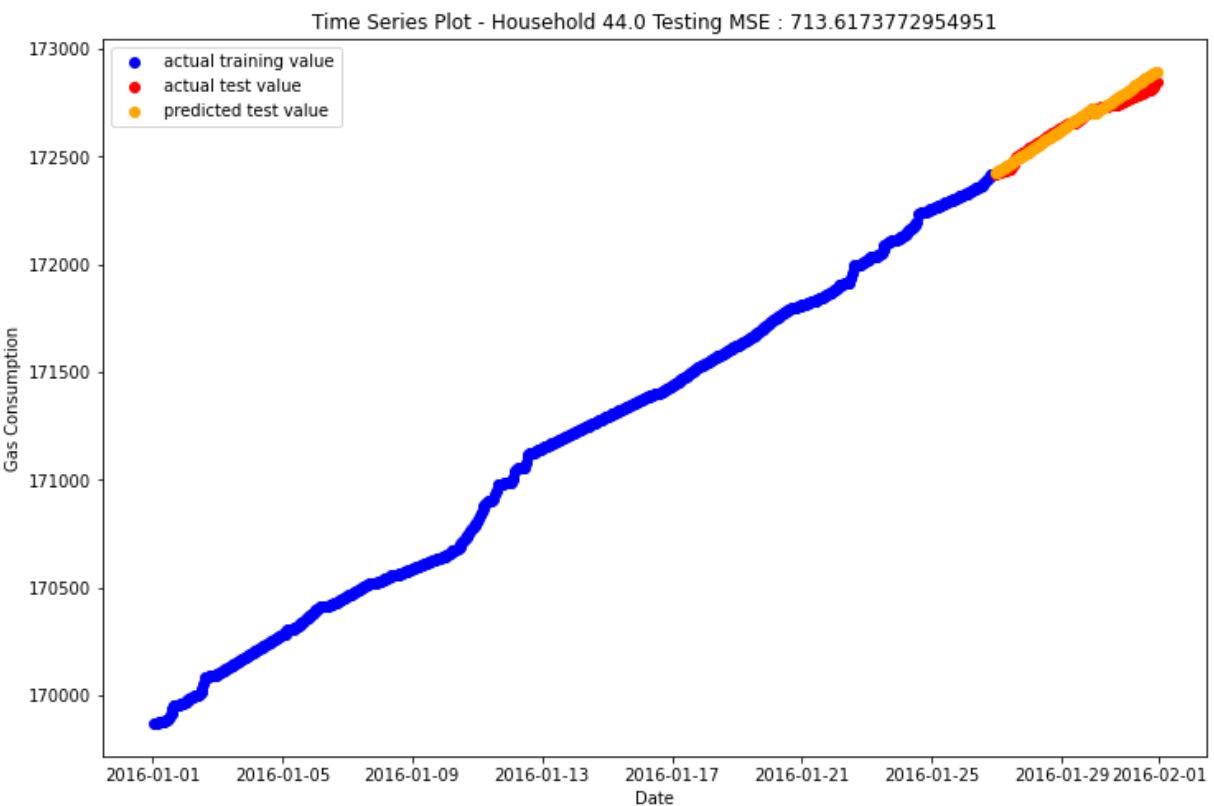
Now, we will plot the linear regression model for each household.

In [106...]

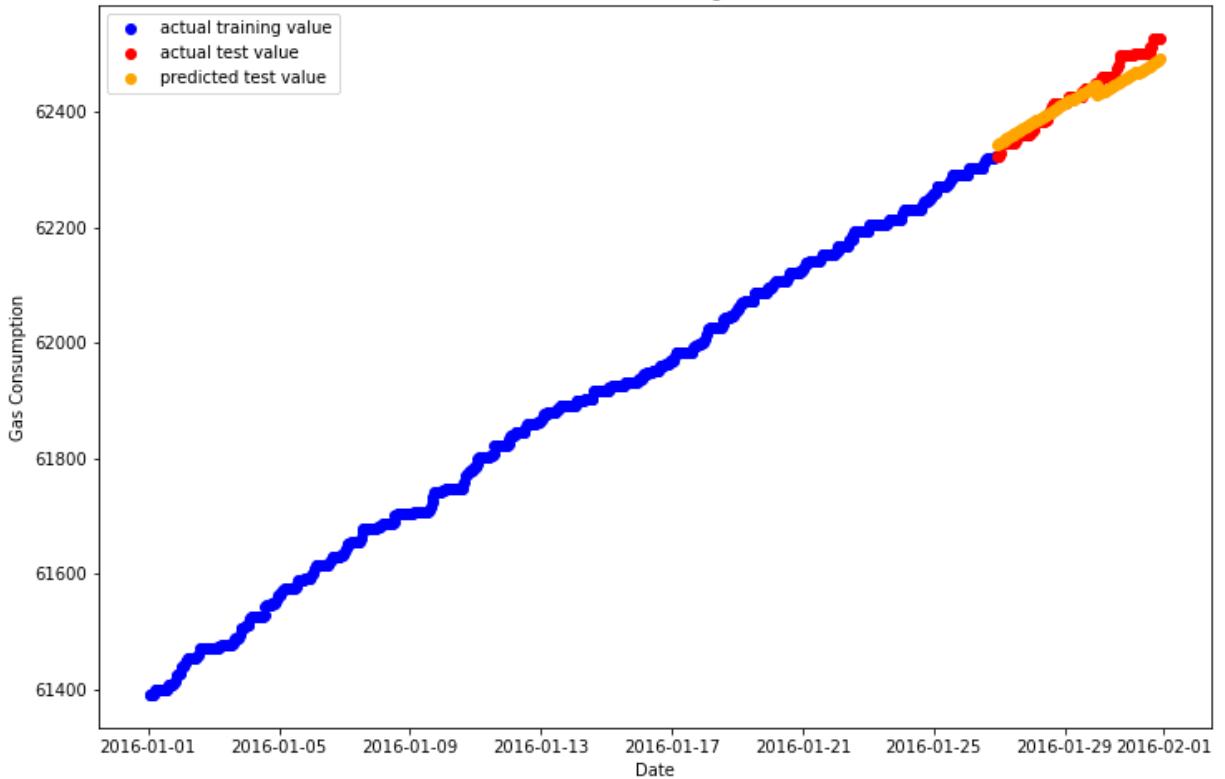
```
# build the Linear regression model
```

```
train_each_house_lr(hours_to_predict)
```

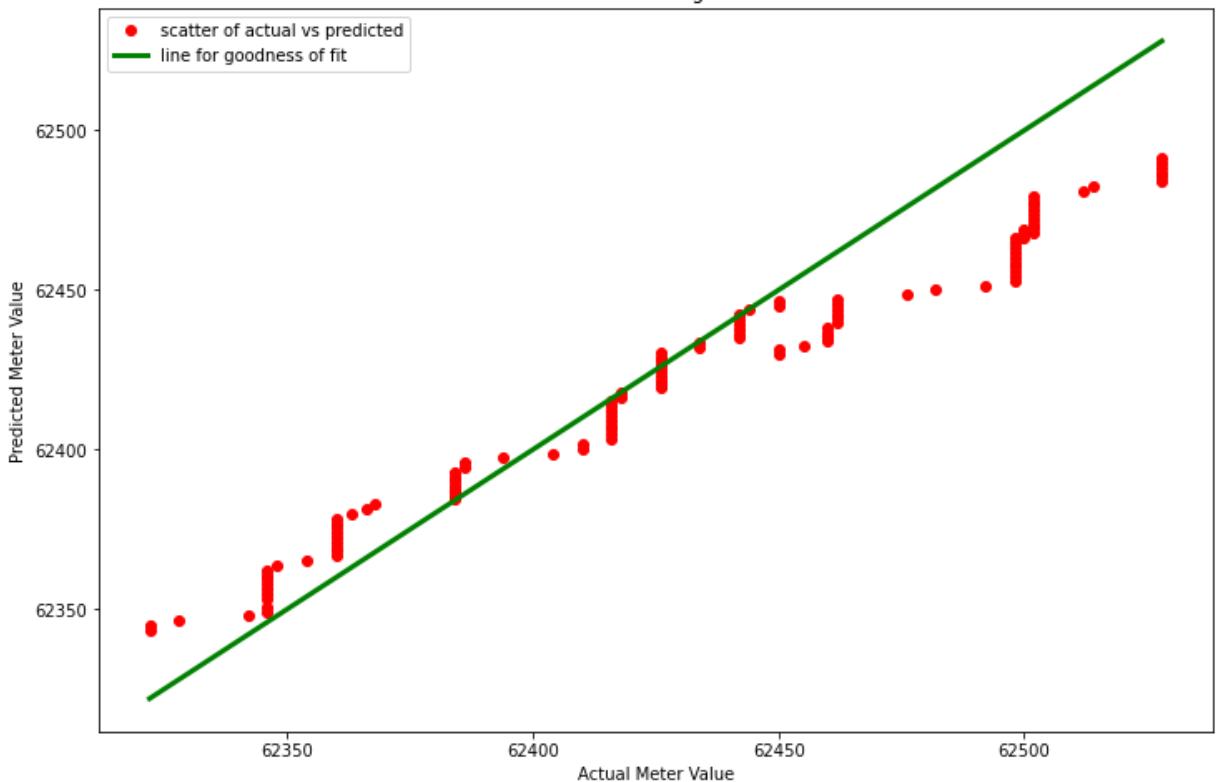




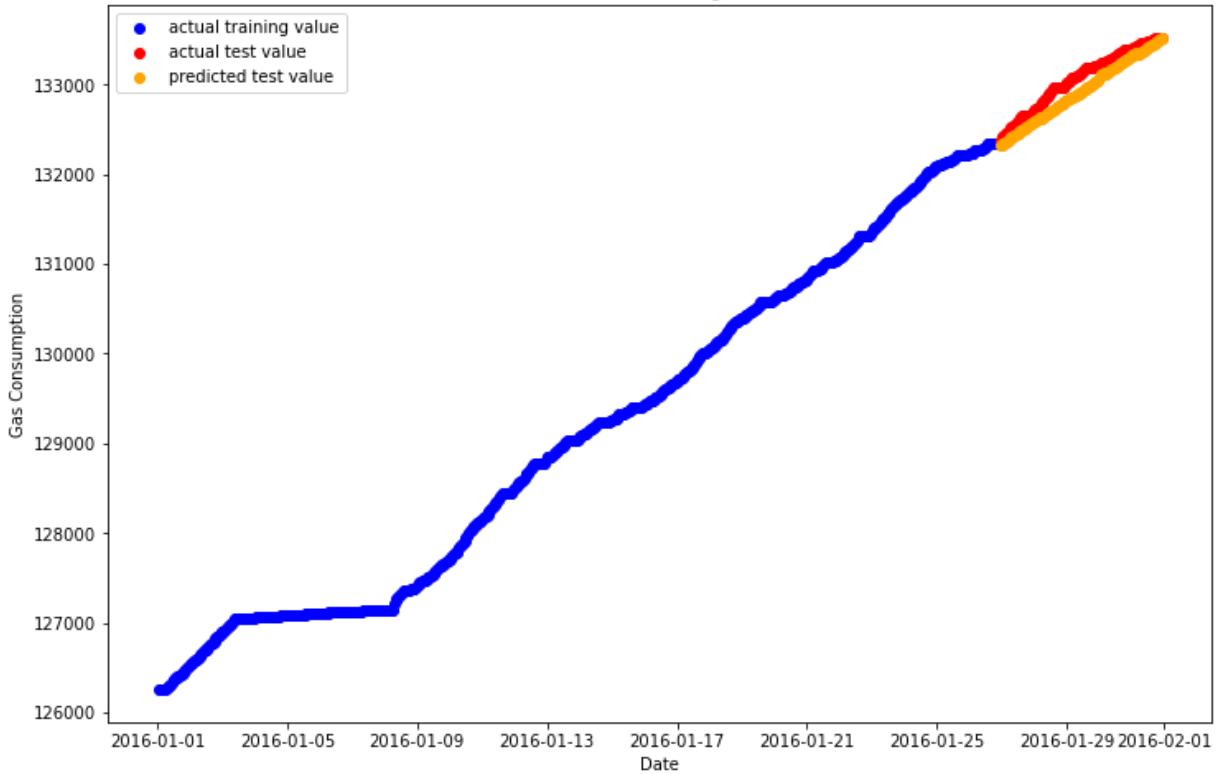
Time Series Plot - Household 77.0 Testing MSE : 445.9034574251401



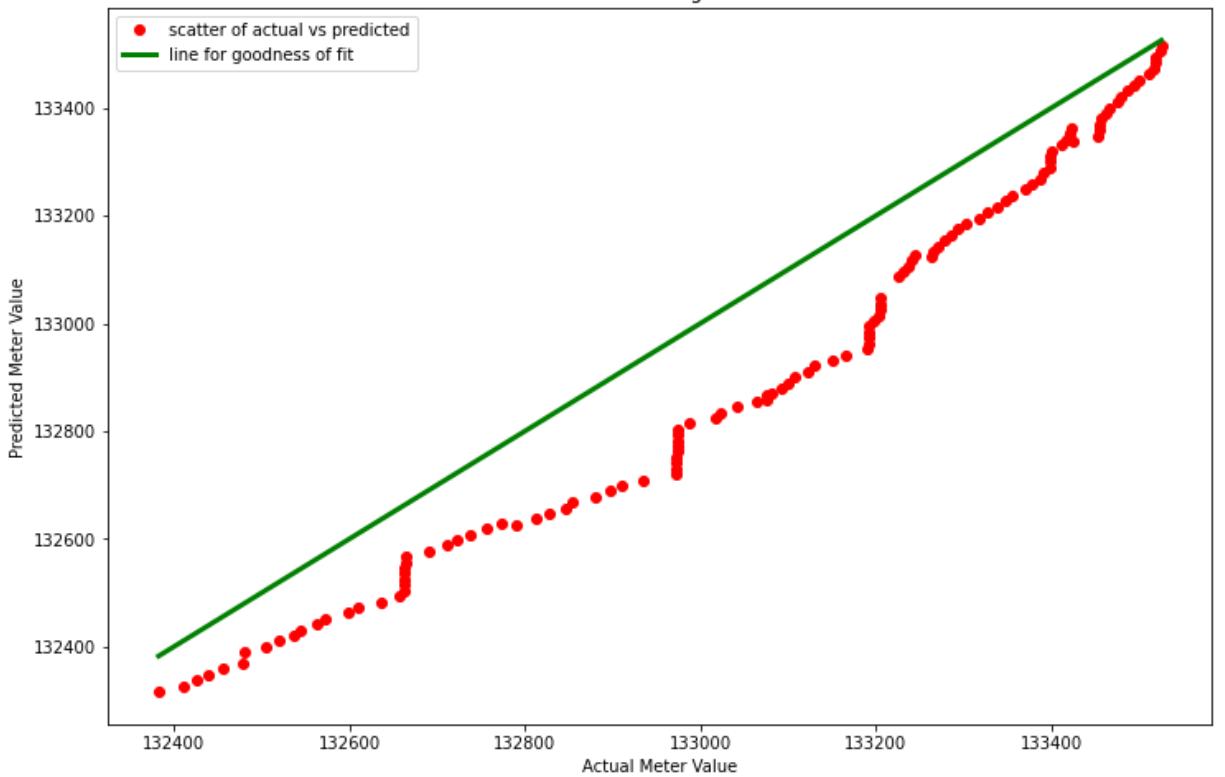
Scatter Plot - Household 77.0 Testing MSE : 445.9034574251401

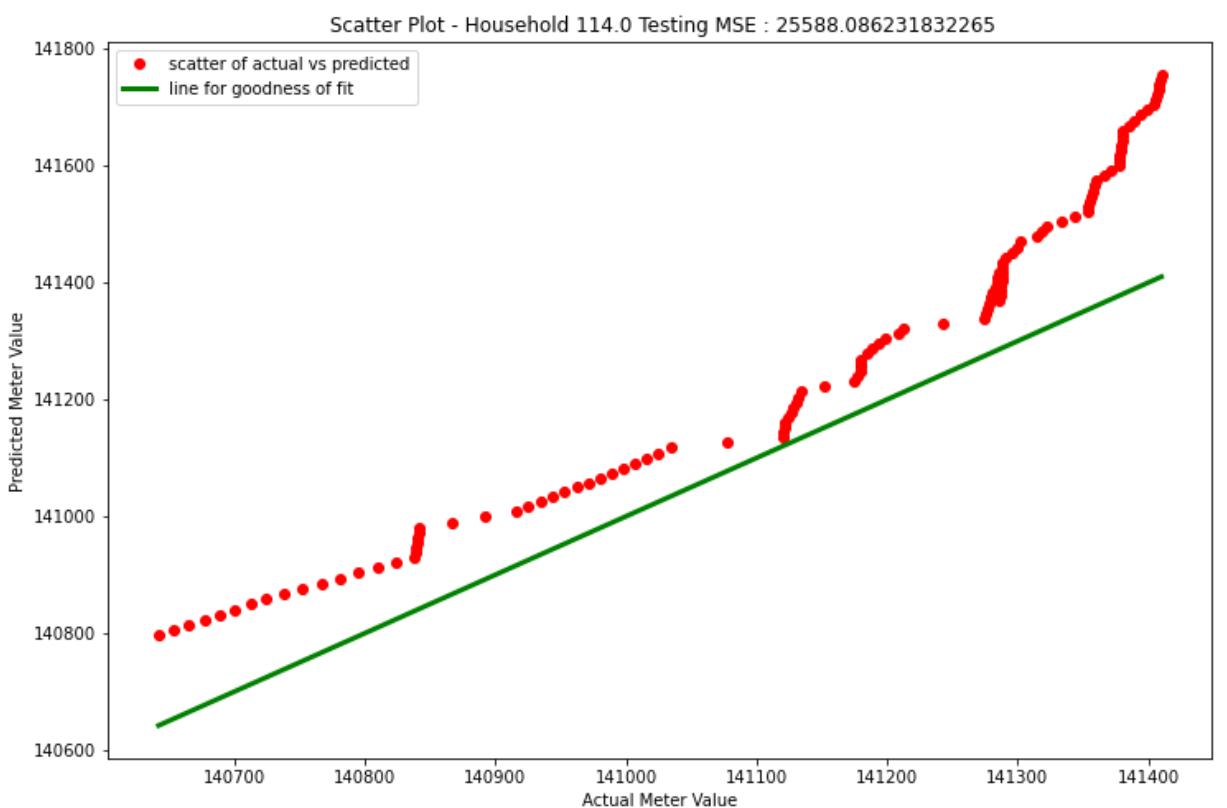
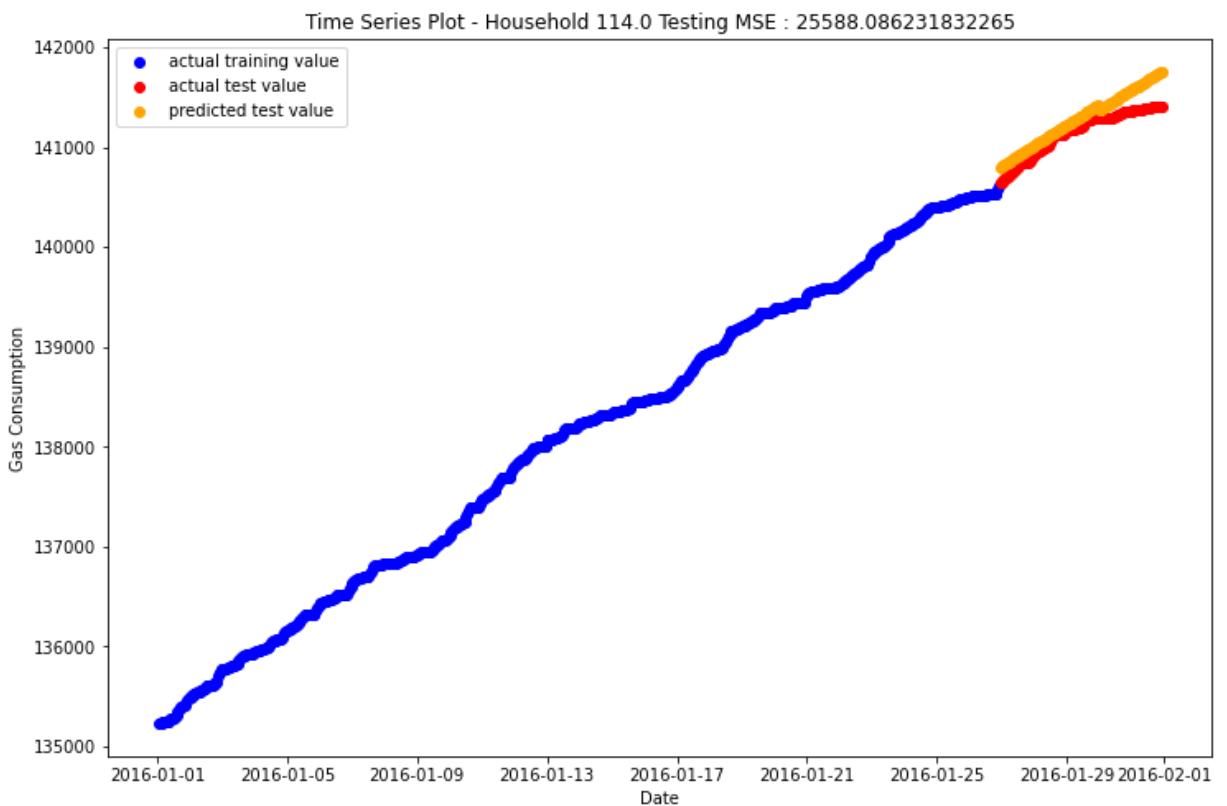


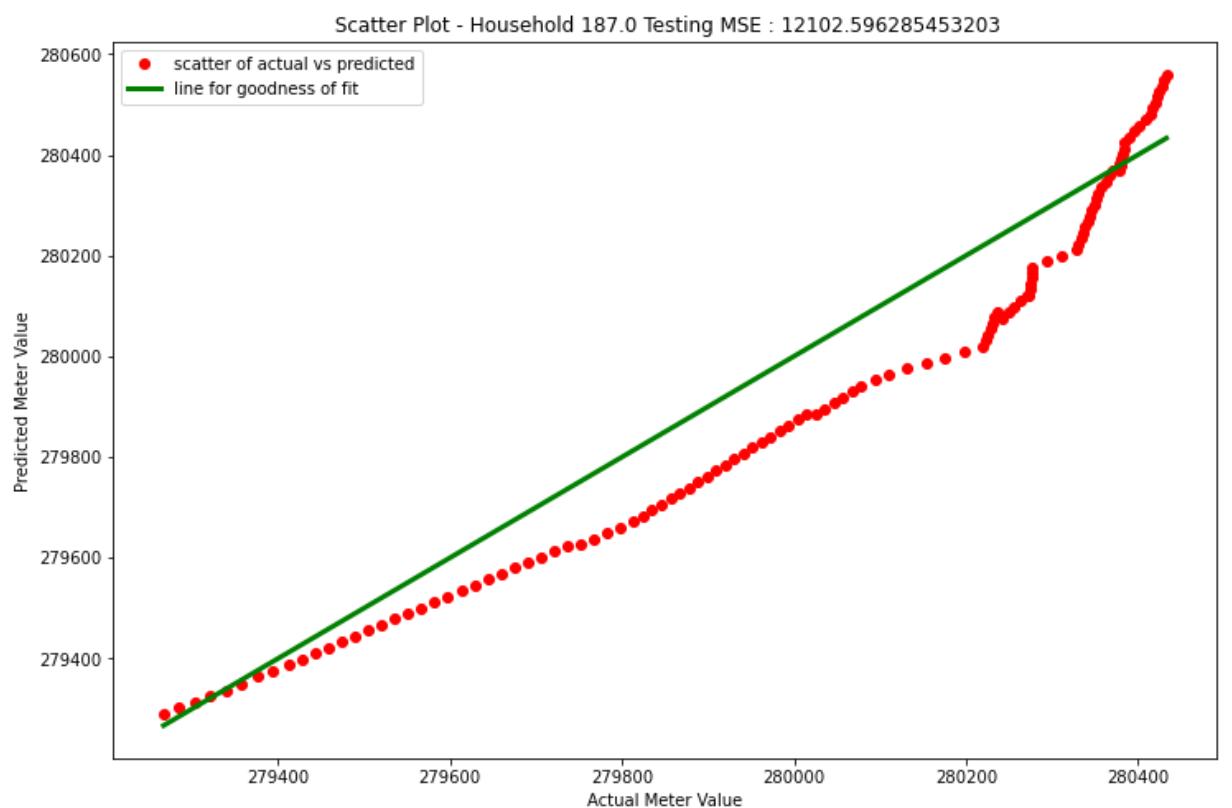
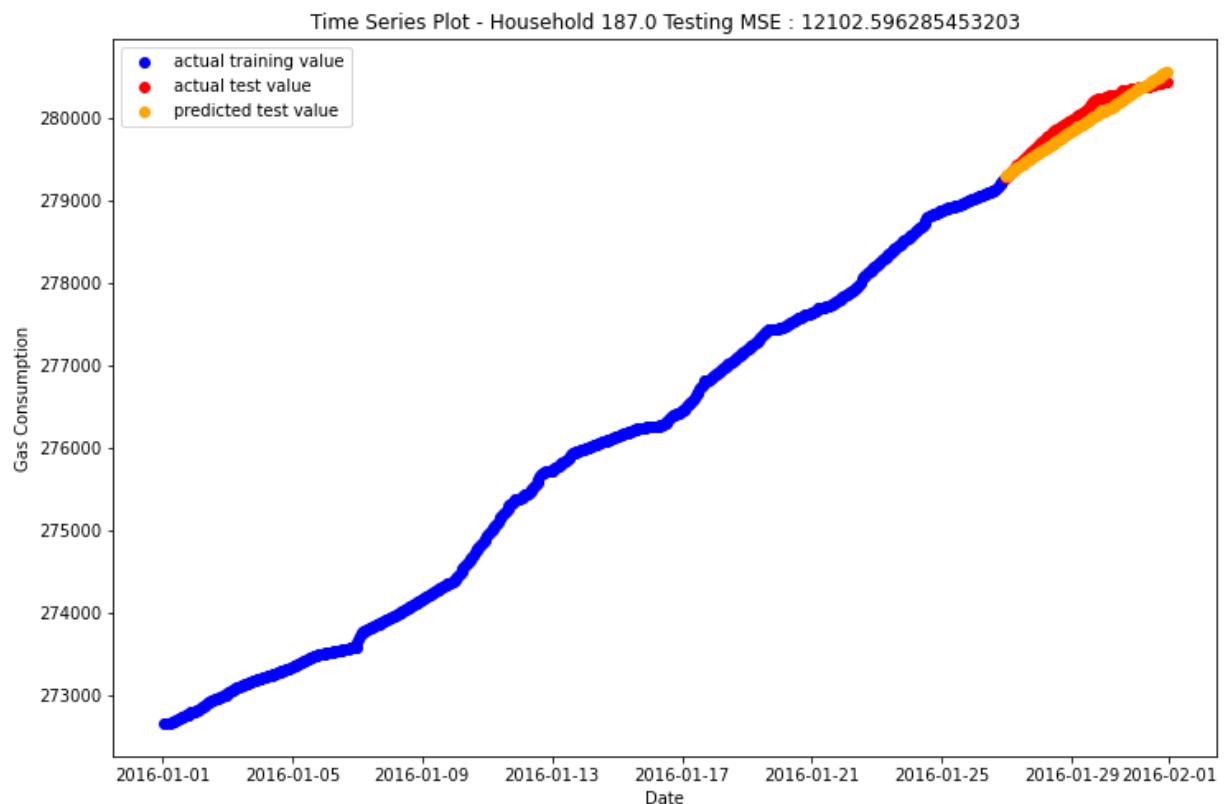
Time Series Plot - Household 94.0 Testing MSE : 22374.699875376224



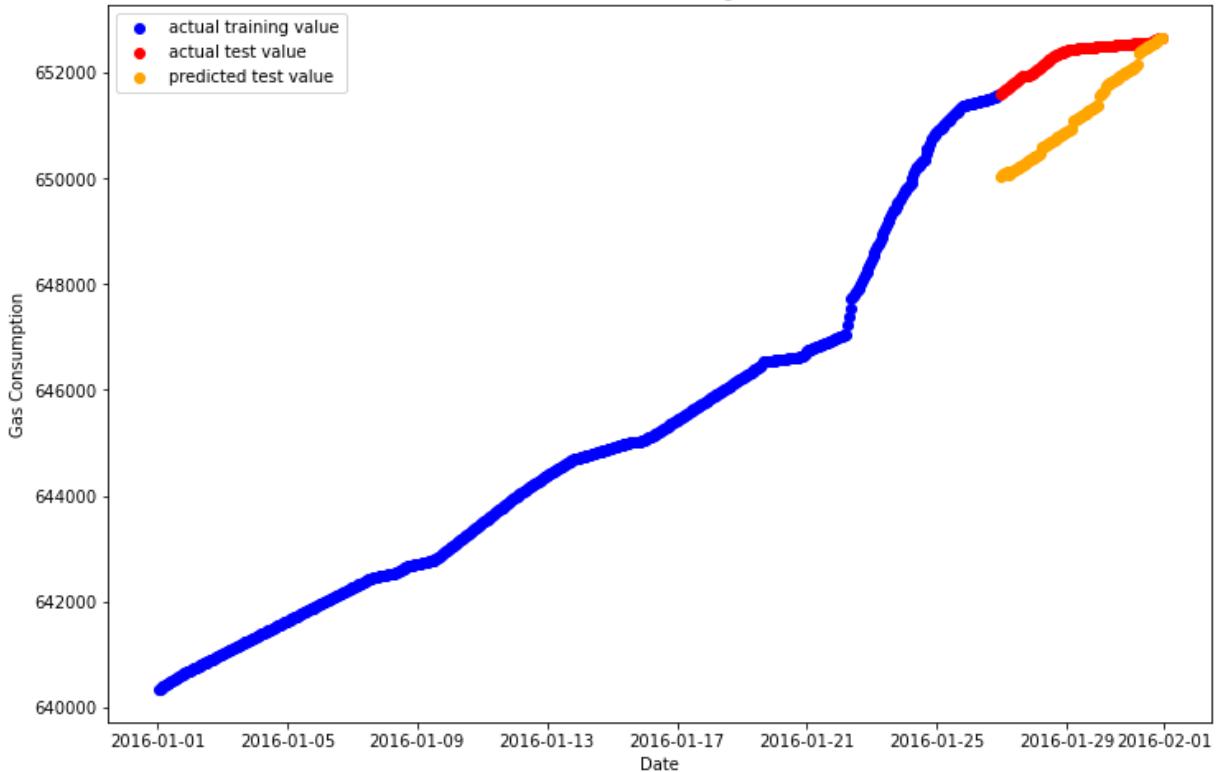
Scatter Plot - Household 94.0 Testing MSE : 22374.699875376224



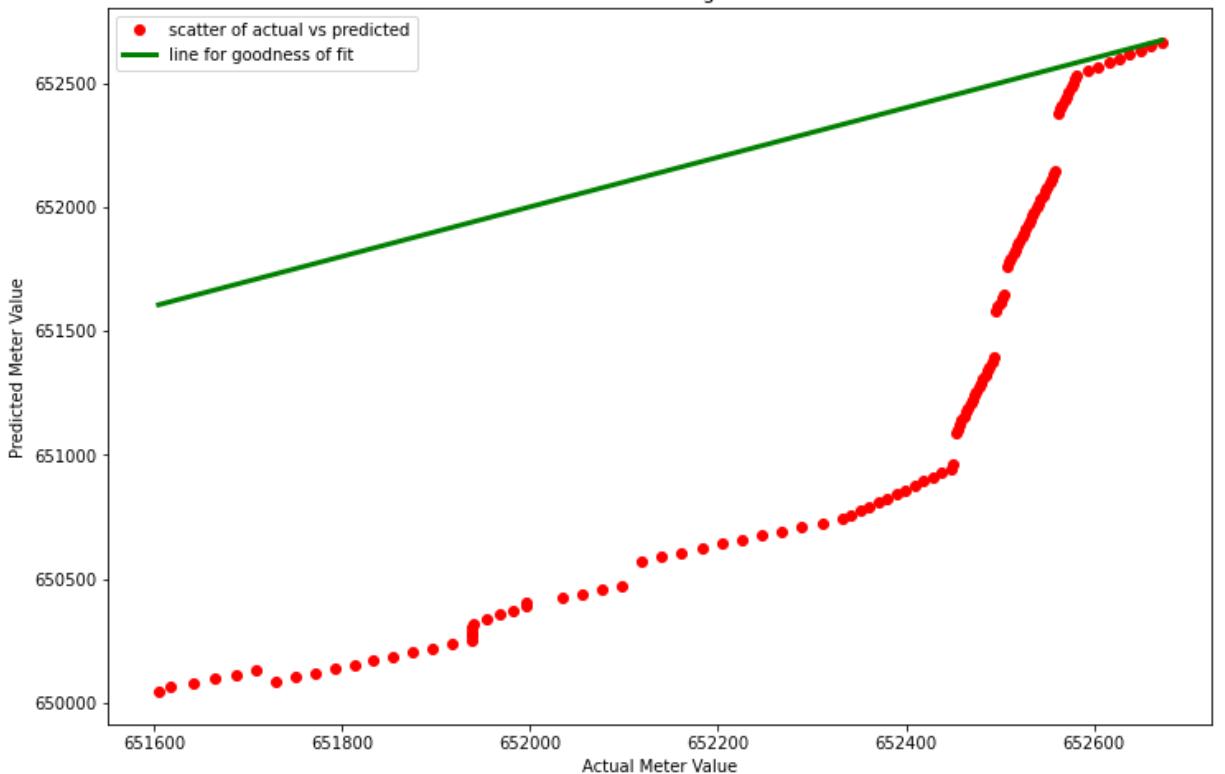


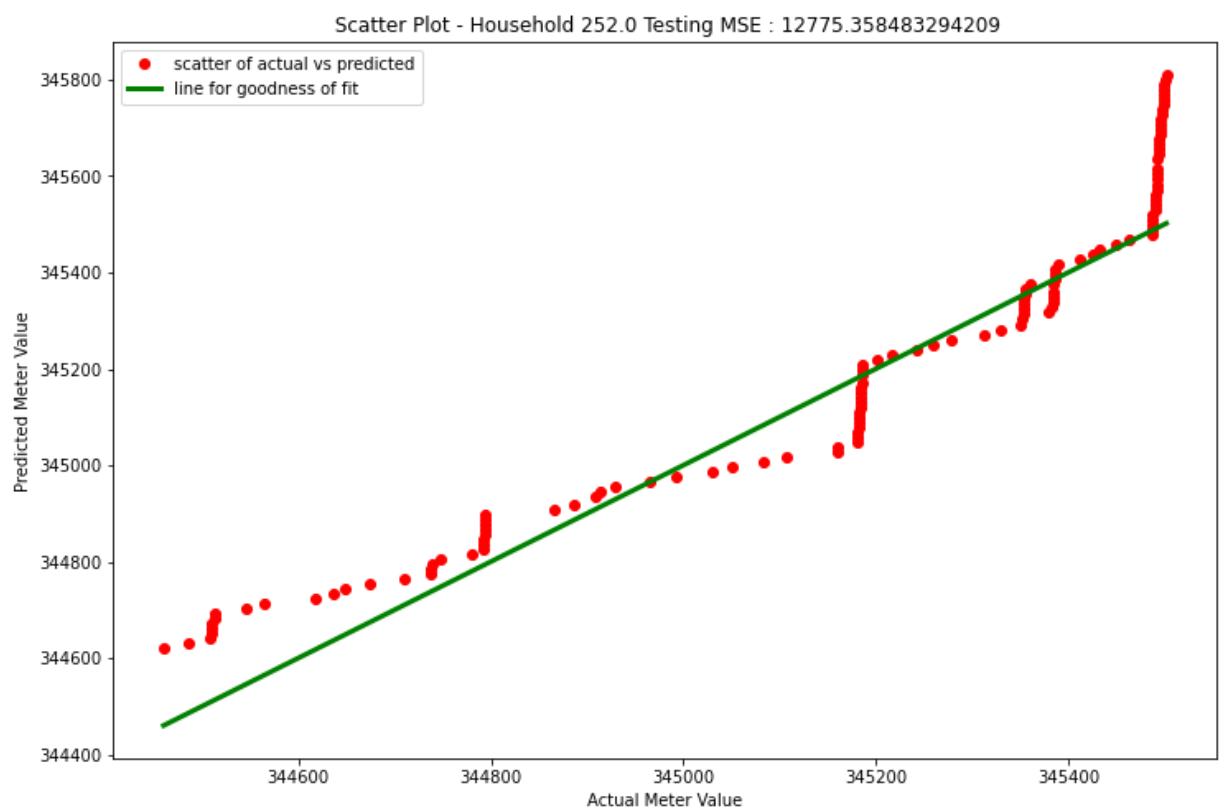
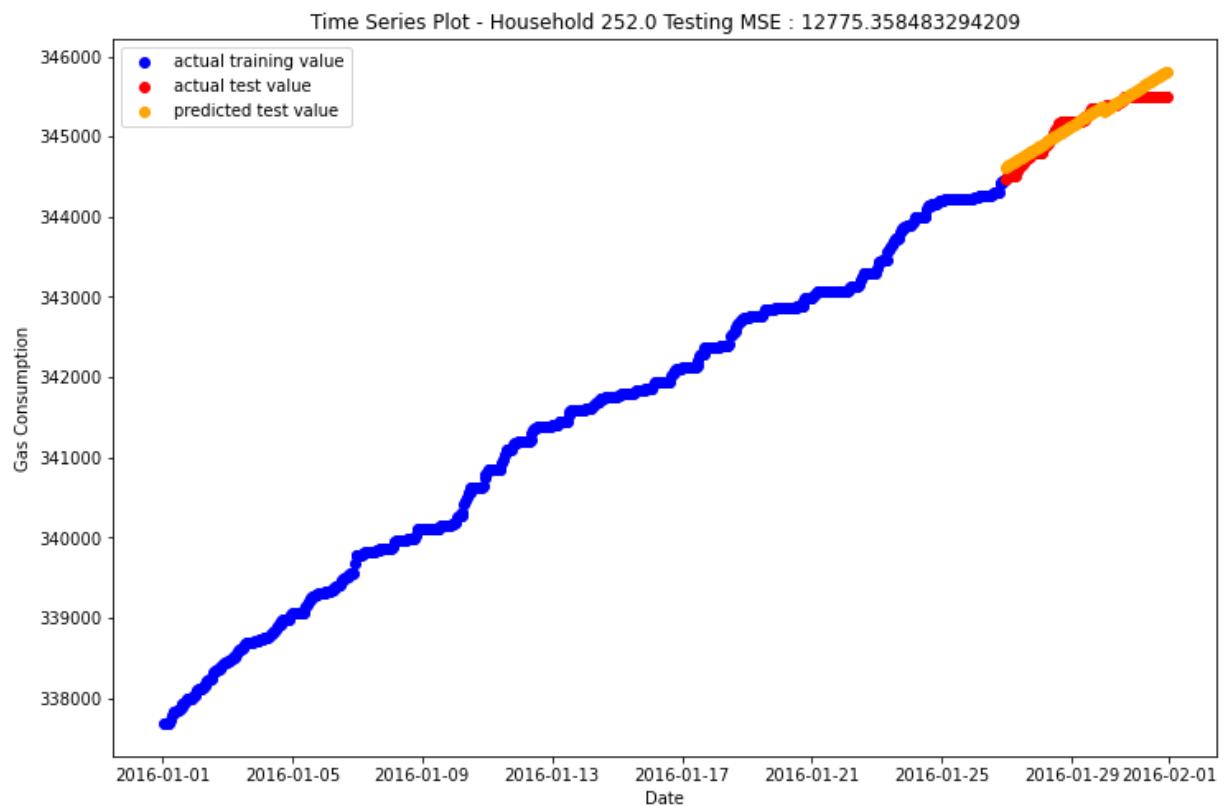


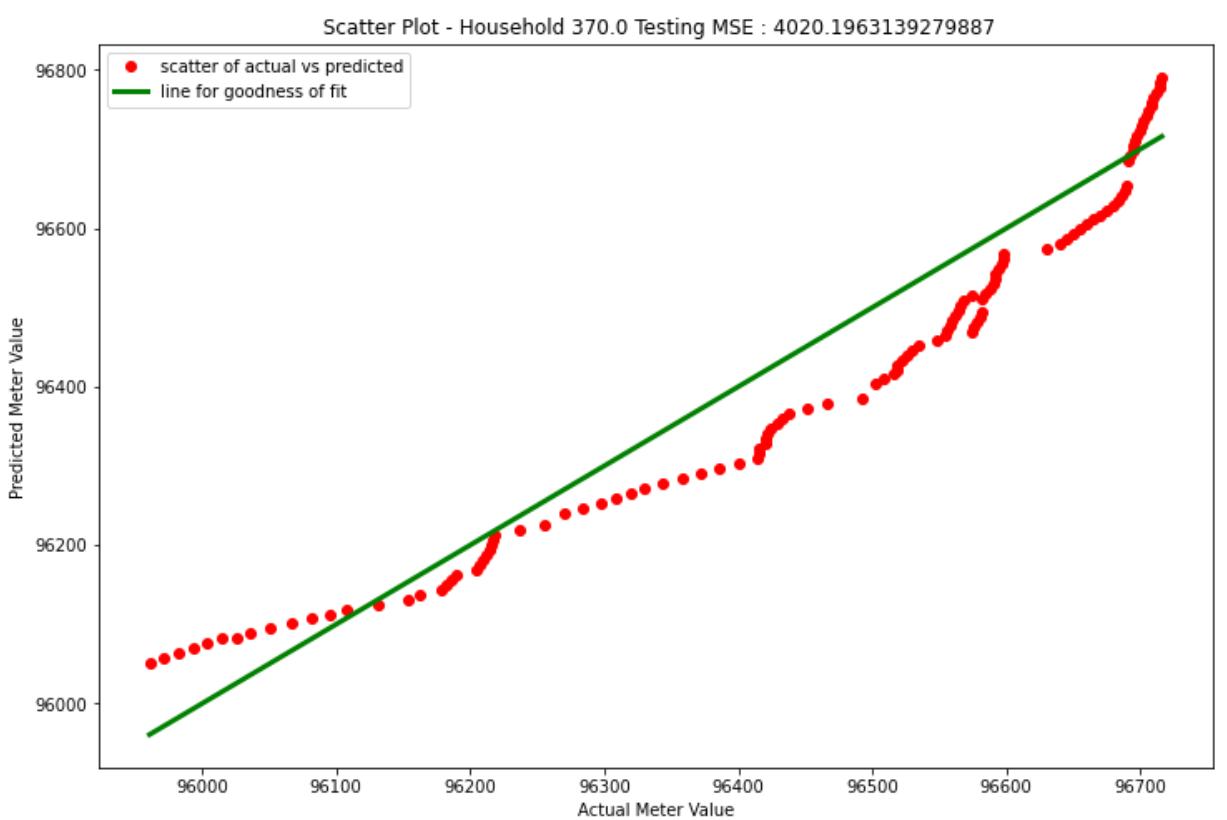
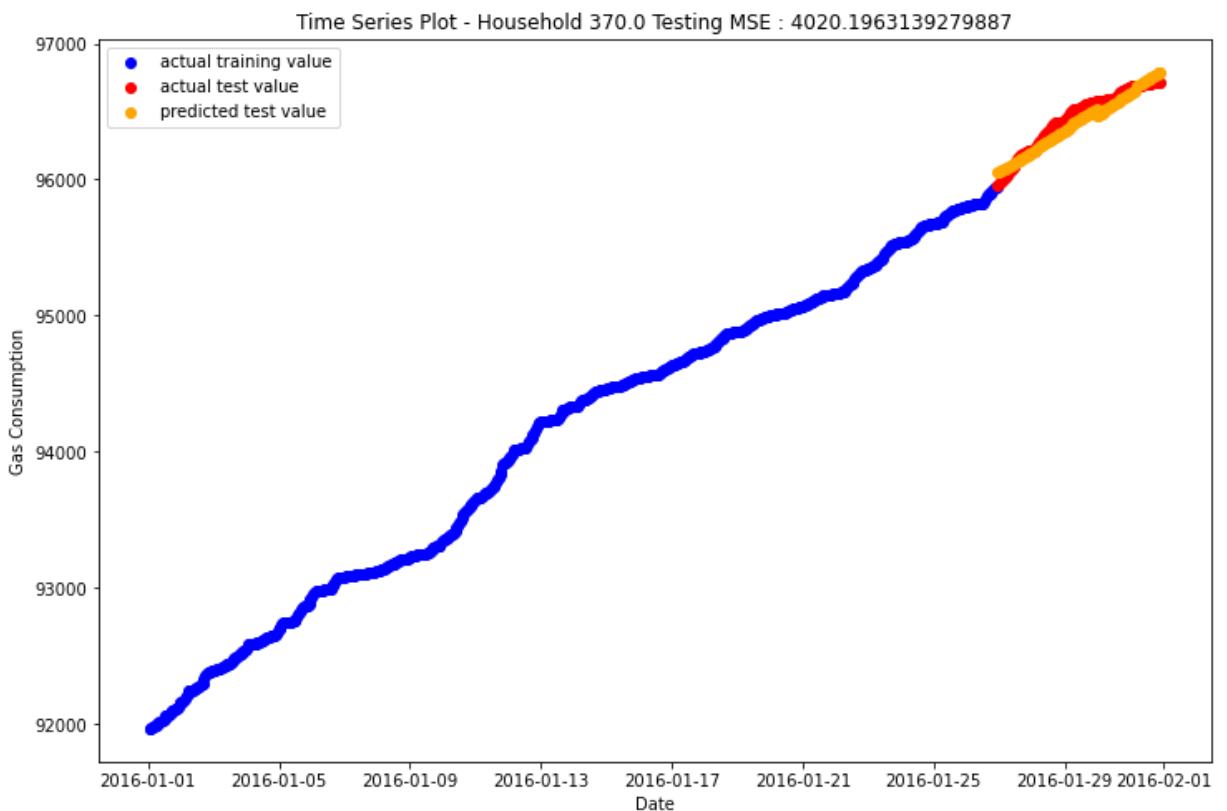
Time Series Plot - Household 222.0 Testing MSE : 1489349.406957789

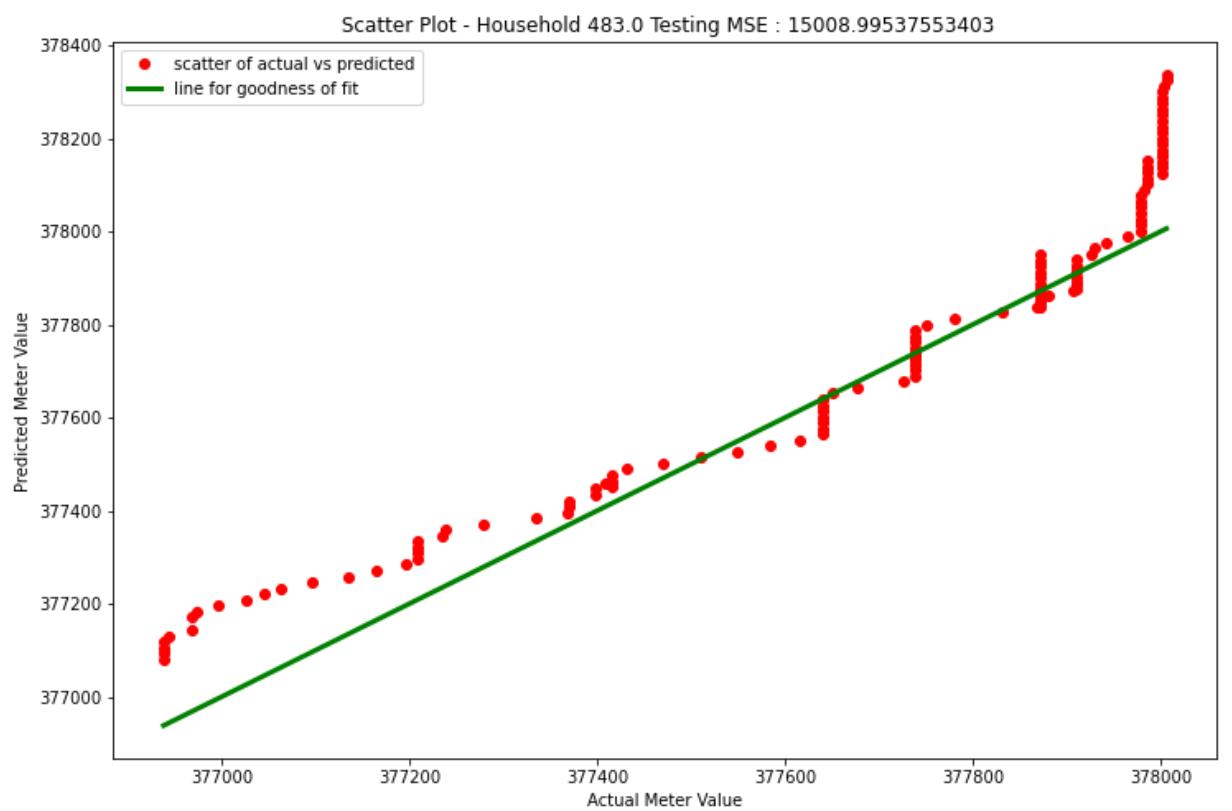
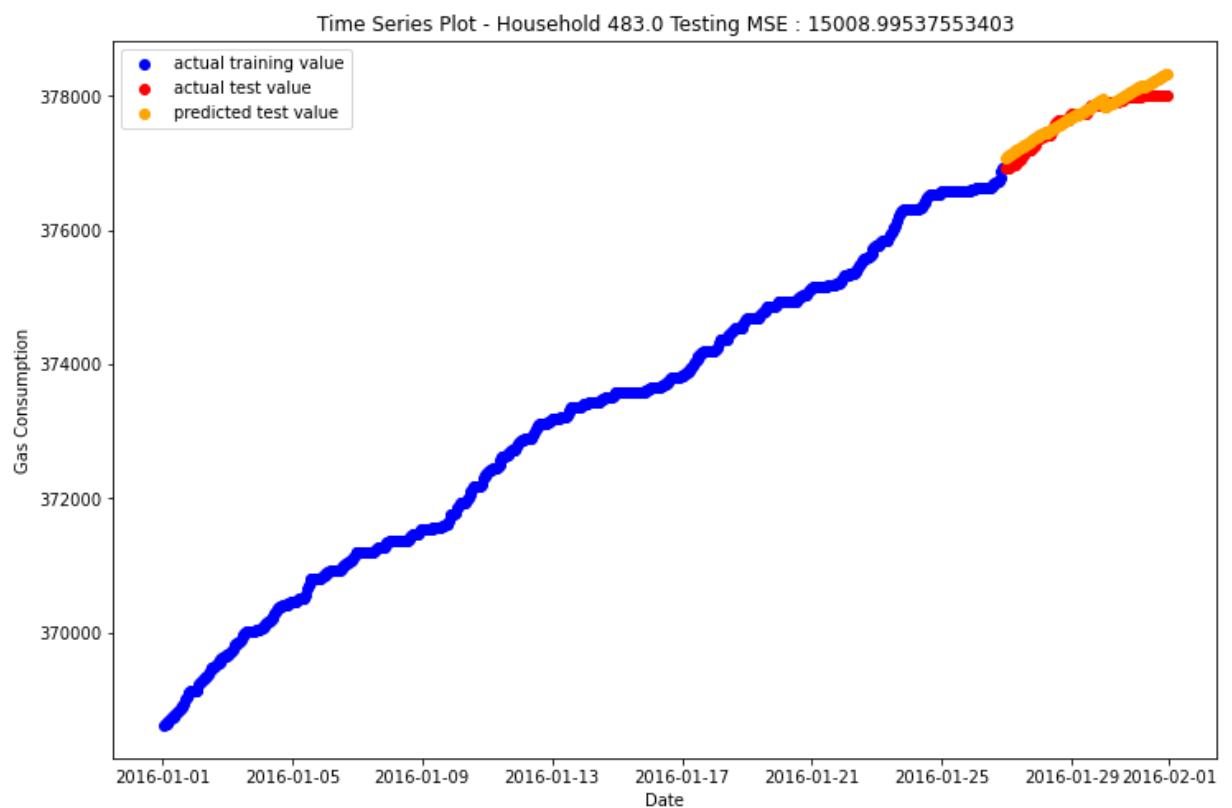


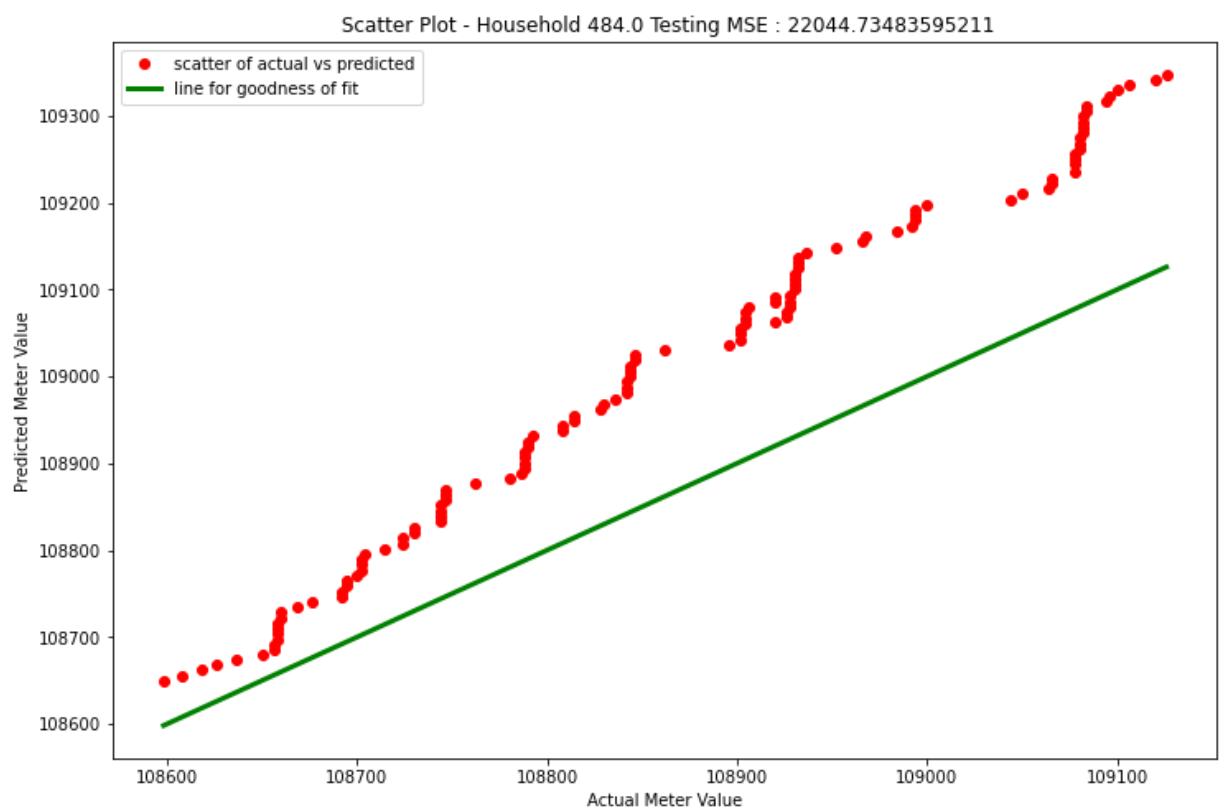
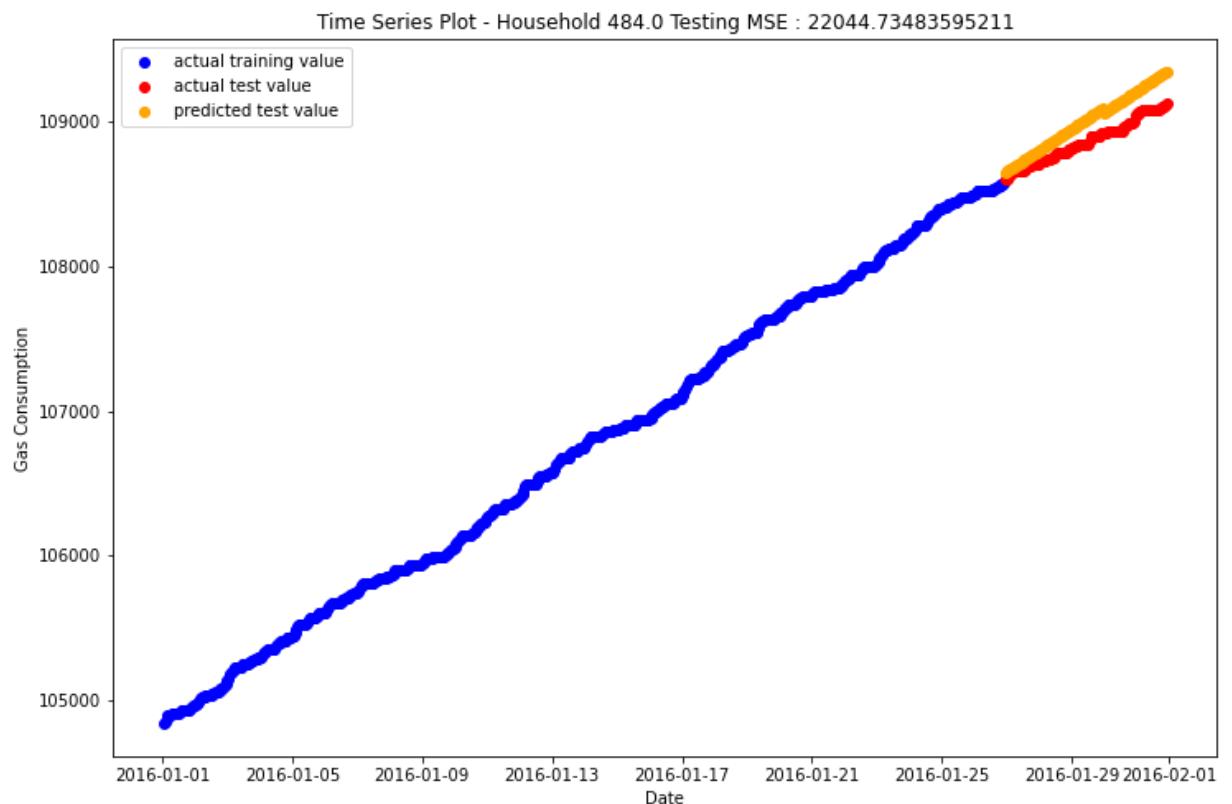
Scatter Plot - Household 222.0 Testing MSE : 1489349.406957789

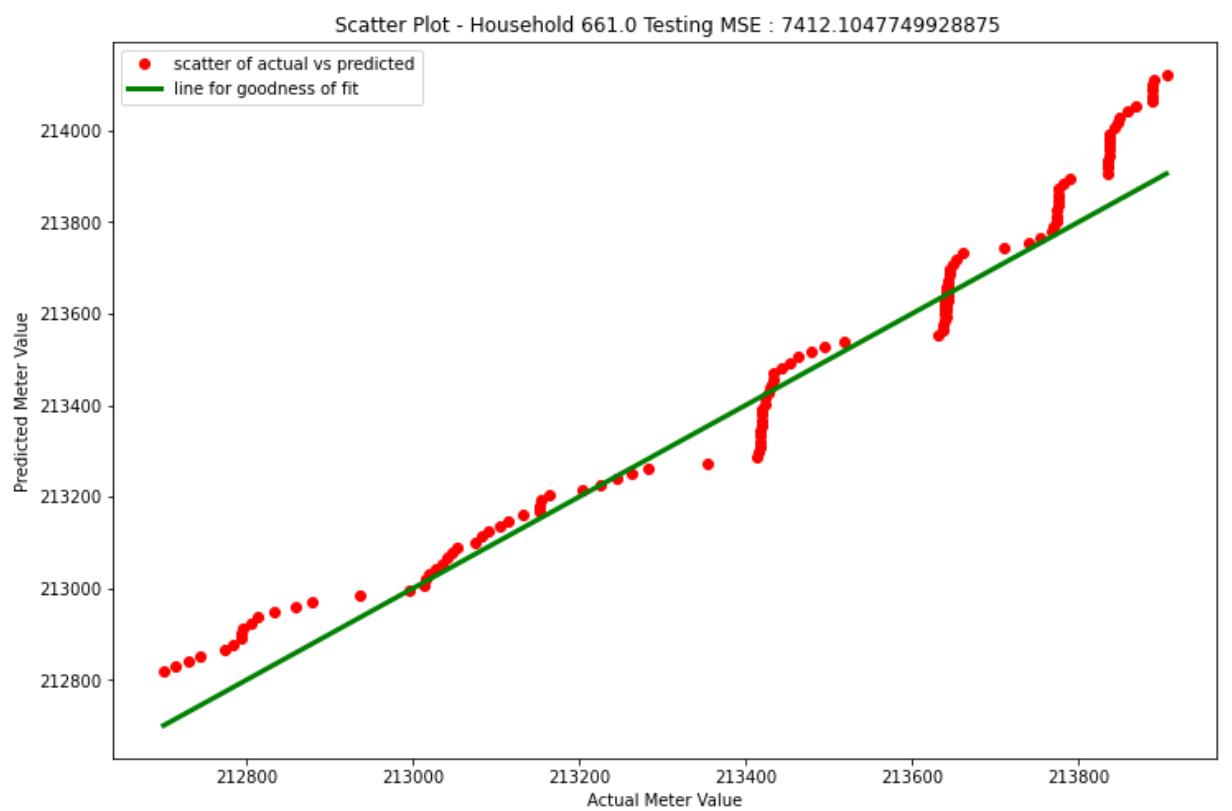
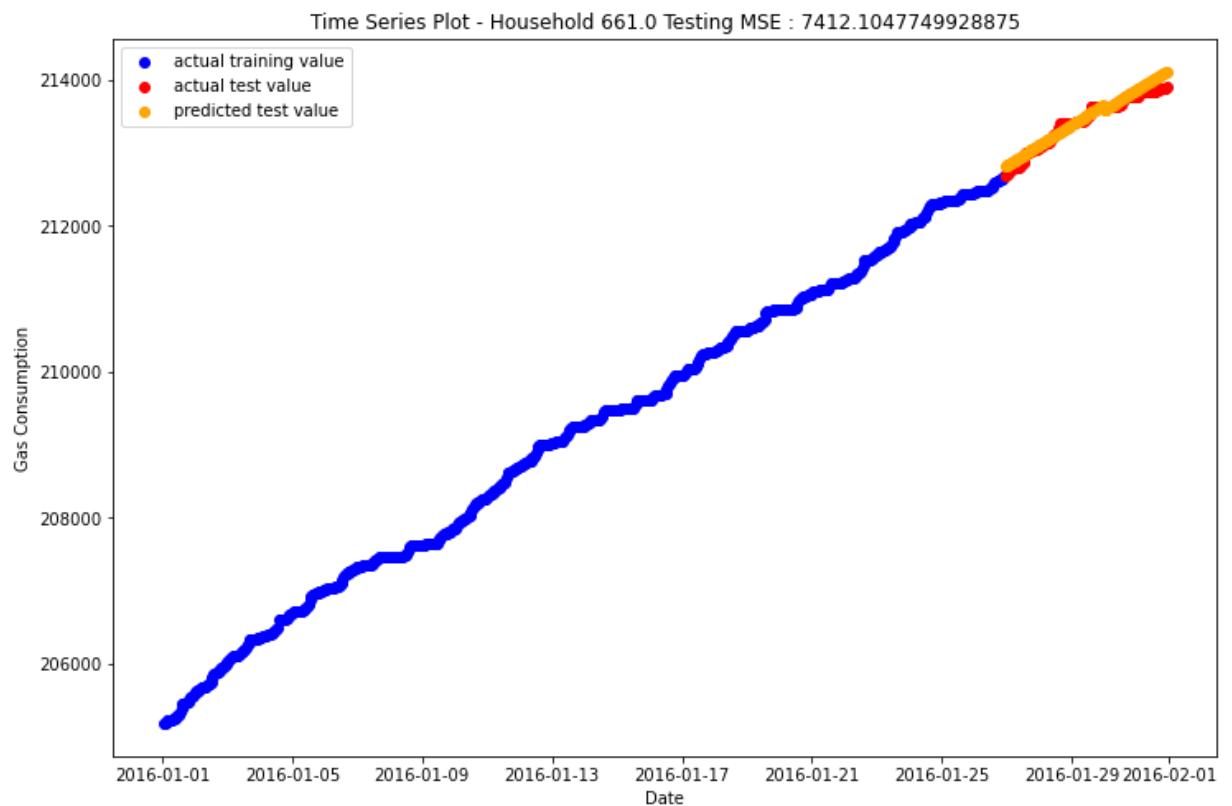


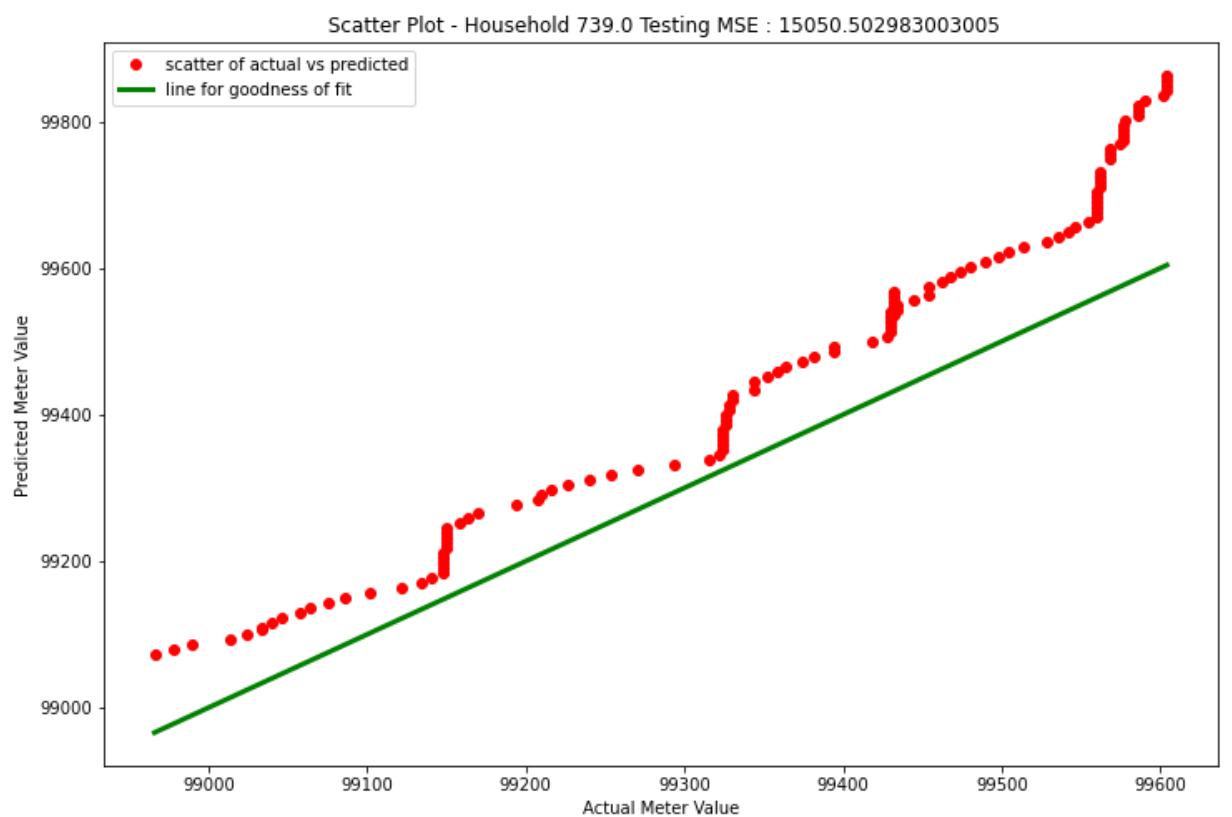
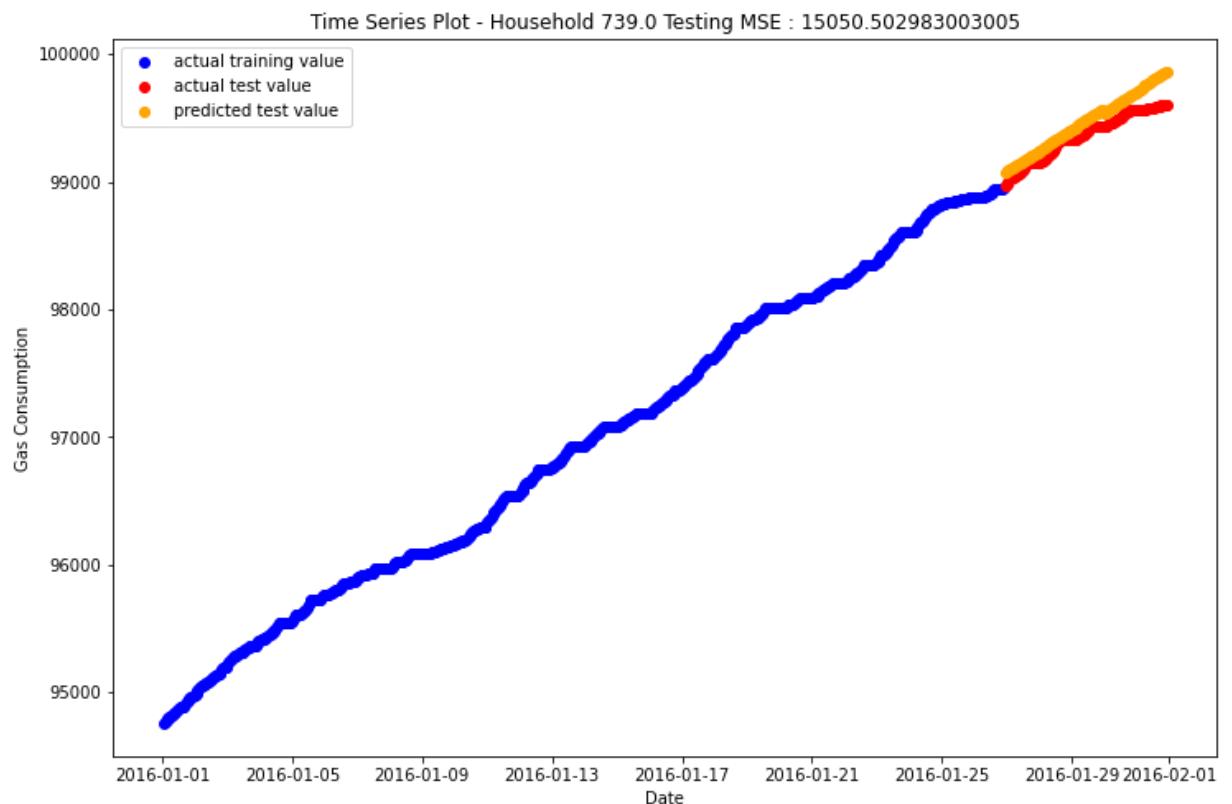




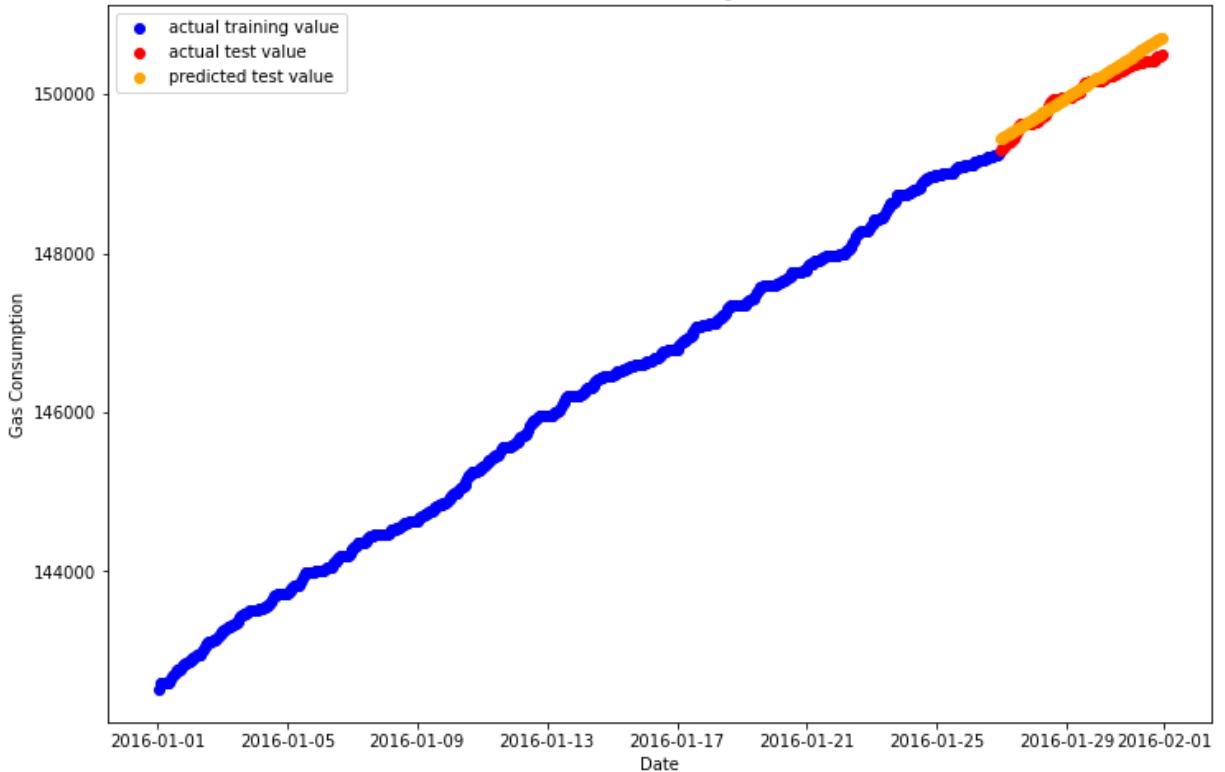




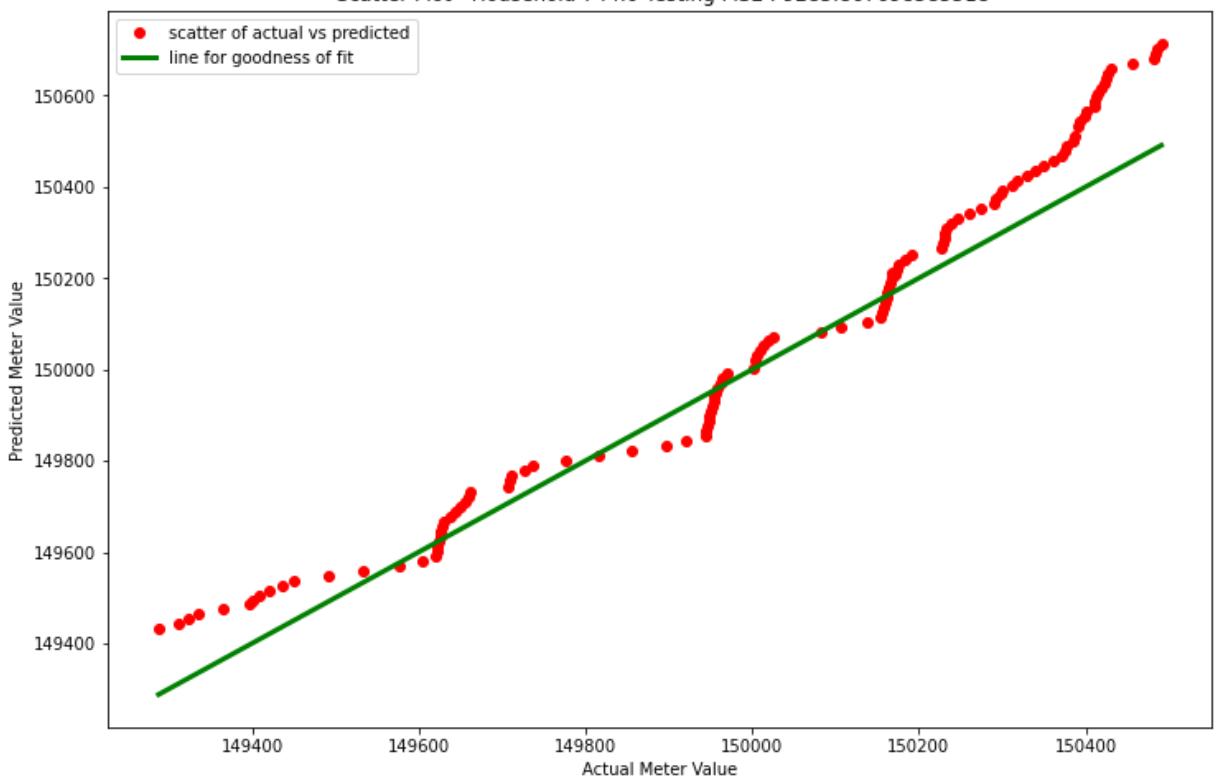




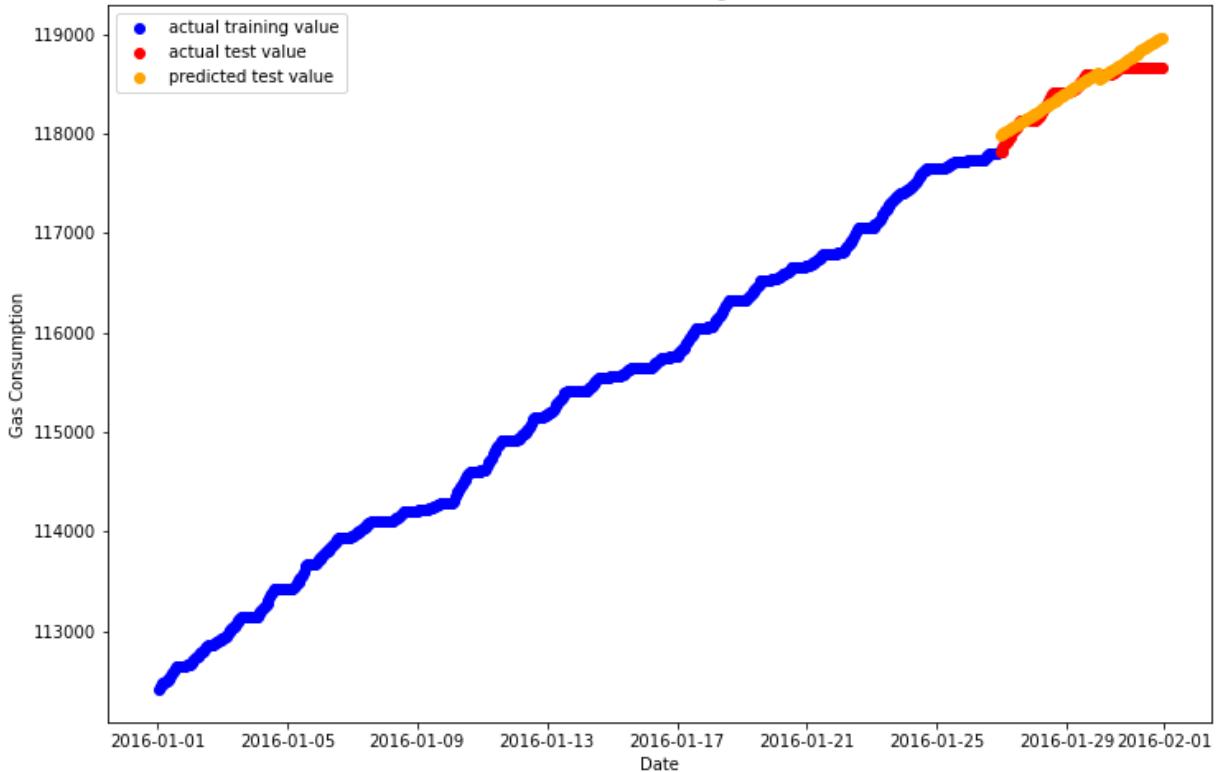
Time Series Plot - Household 744.0 Testing MSE : 9285.867698585518



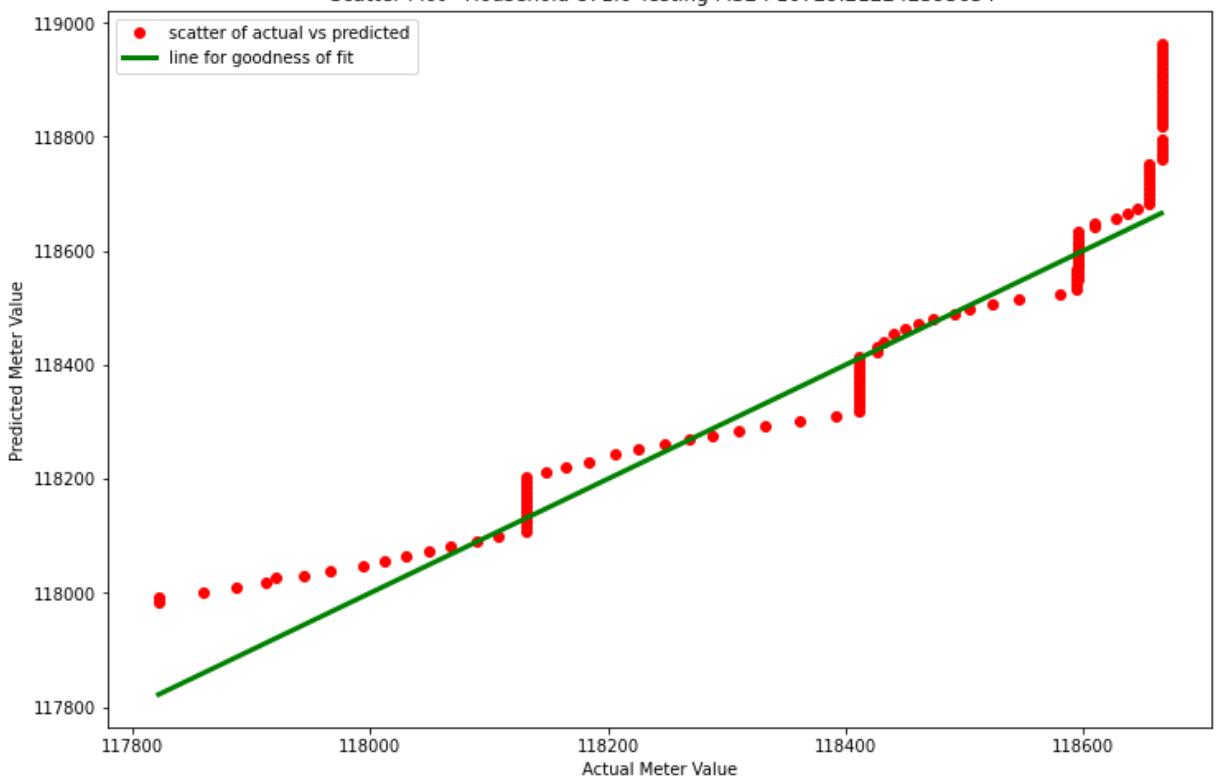
Scatter Plot - Household 744.0 Testing MSE : 9285.867698585518



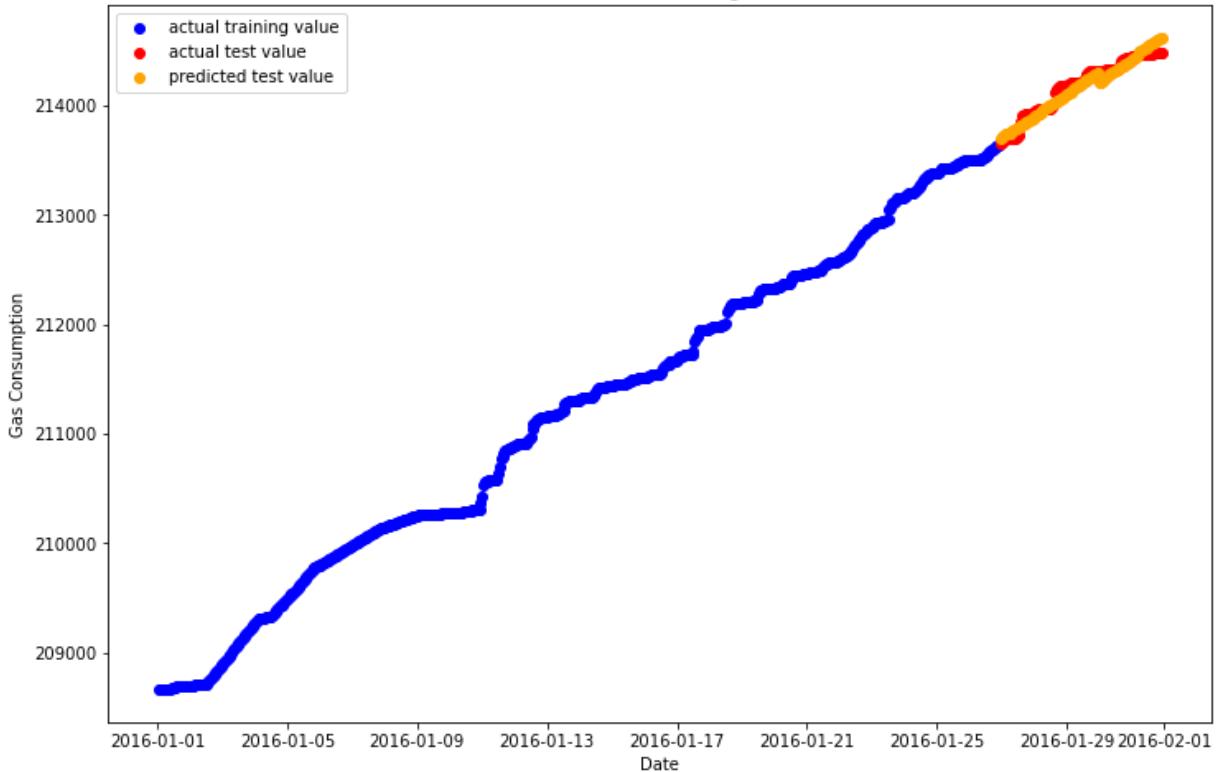
Time Series Plot - Household 871.0 Testing MSE : 10729.212242393634



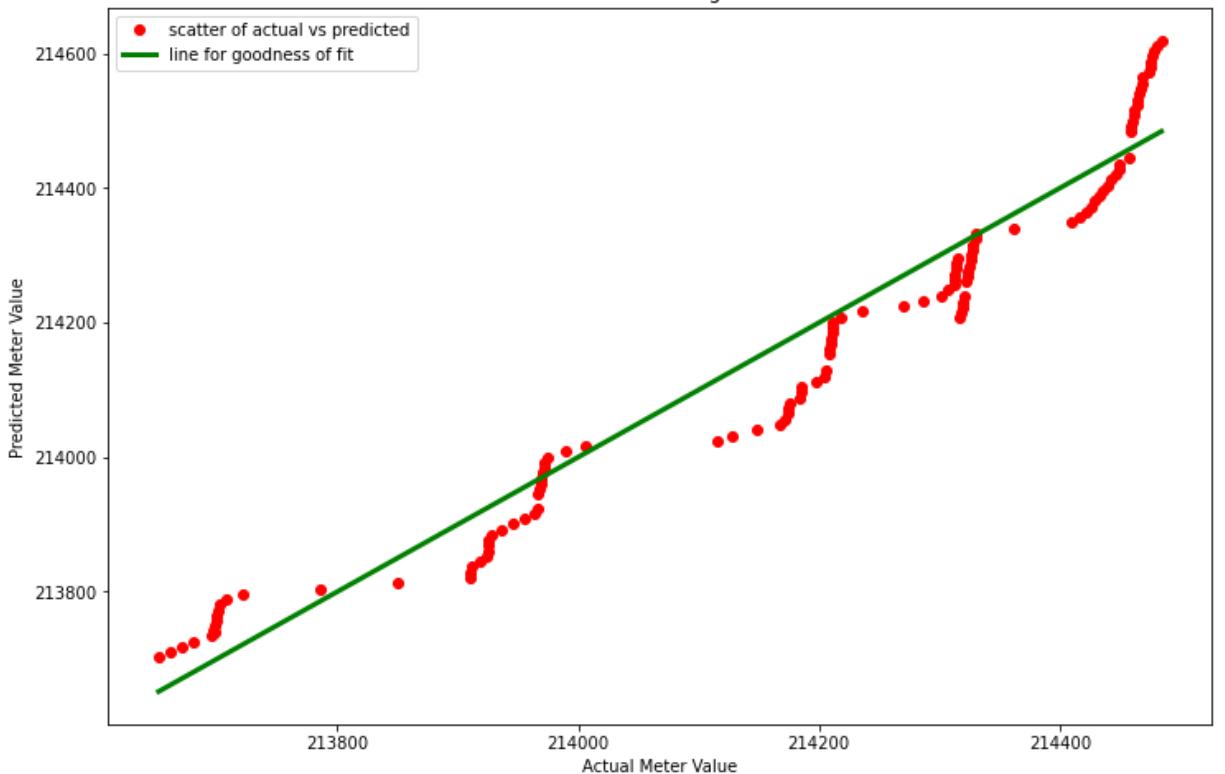
Scatter Plot - Household 871.0 Testing MSE : 10729.212242393634

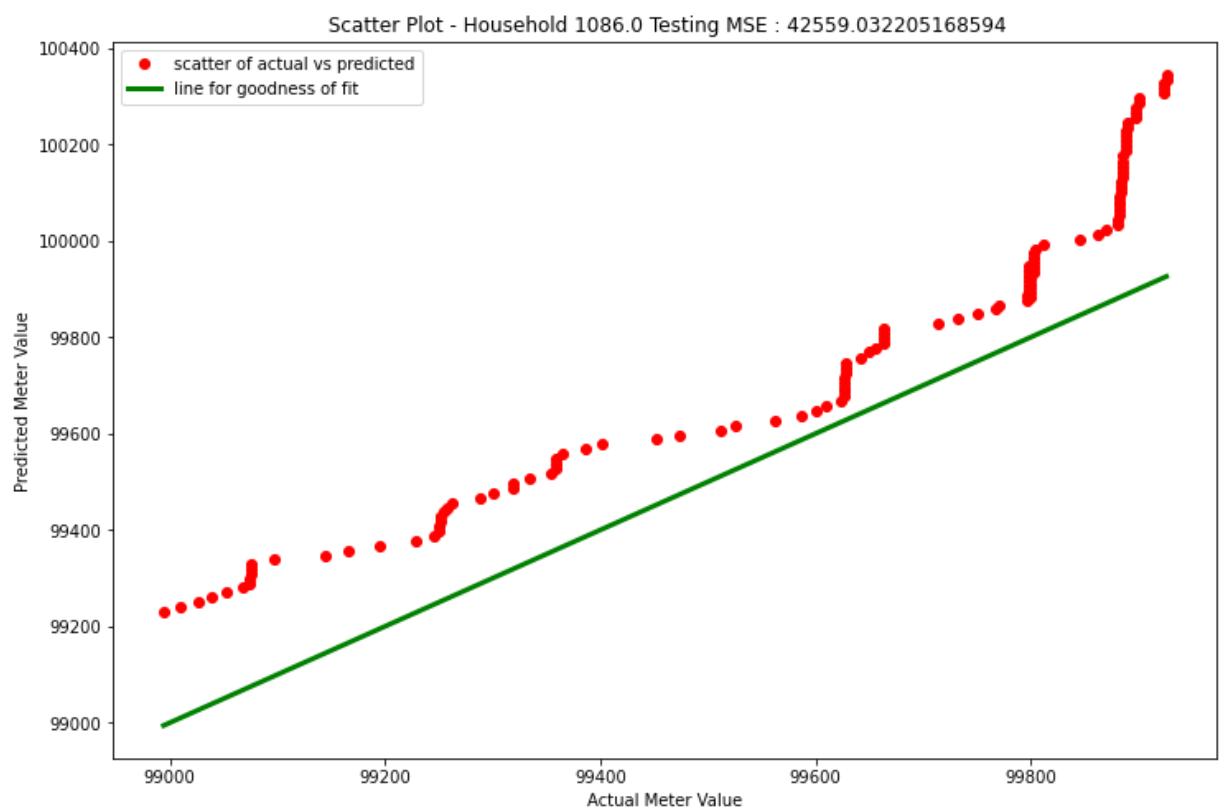
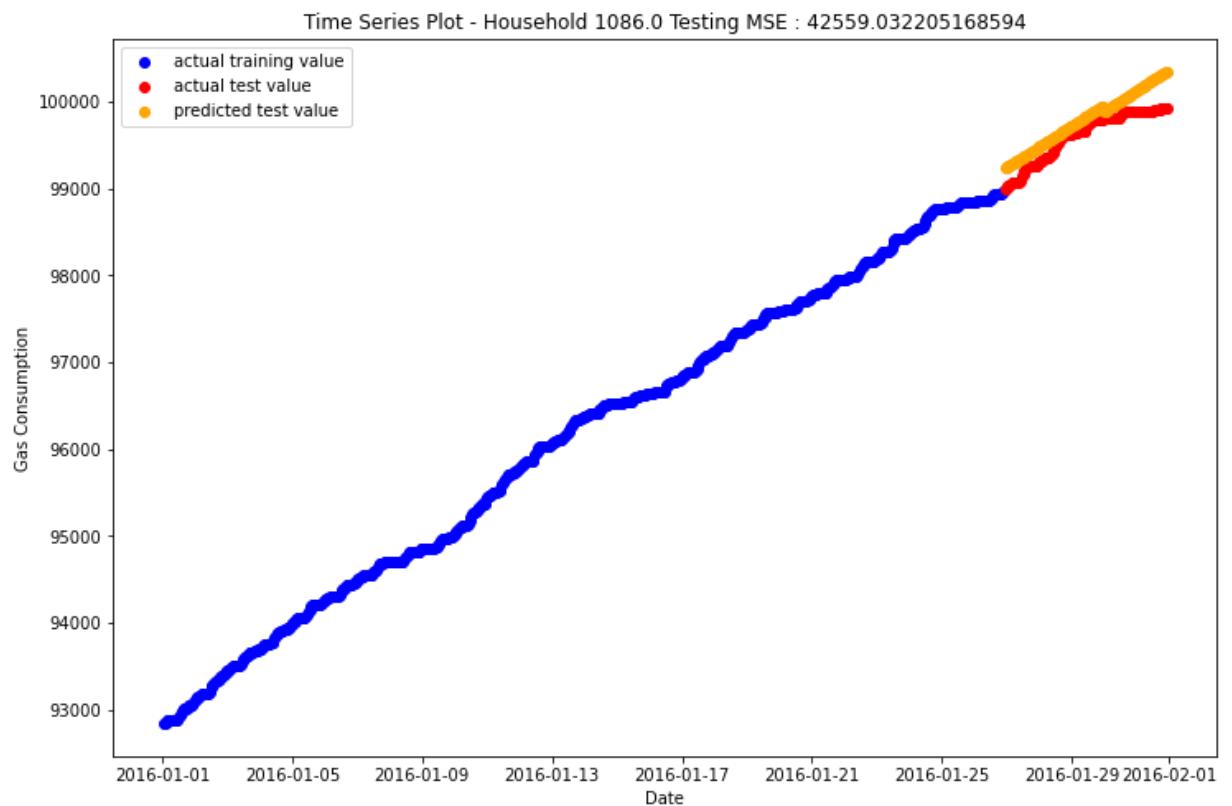


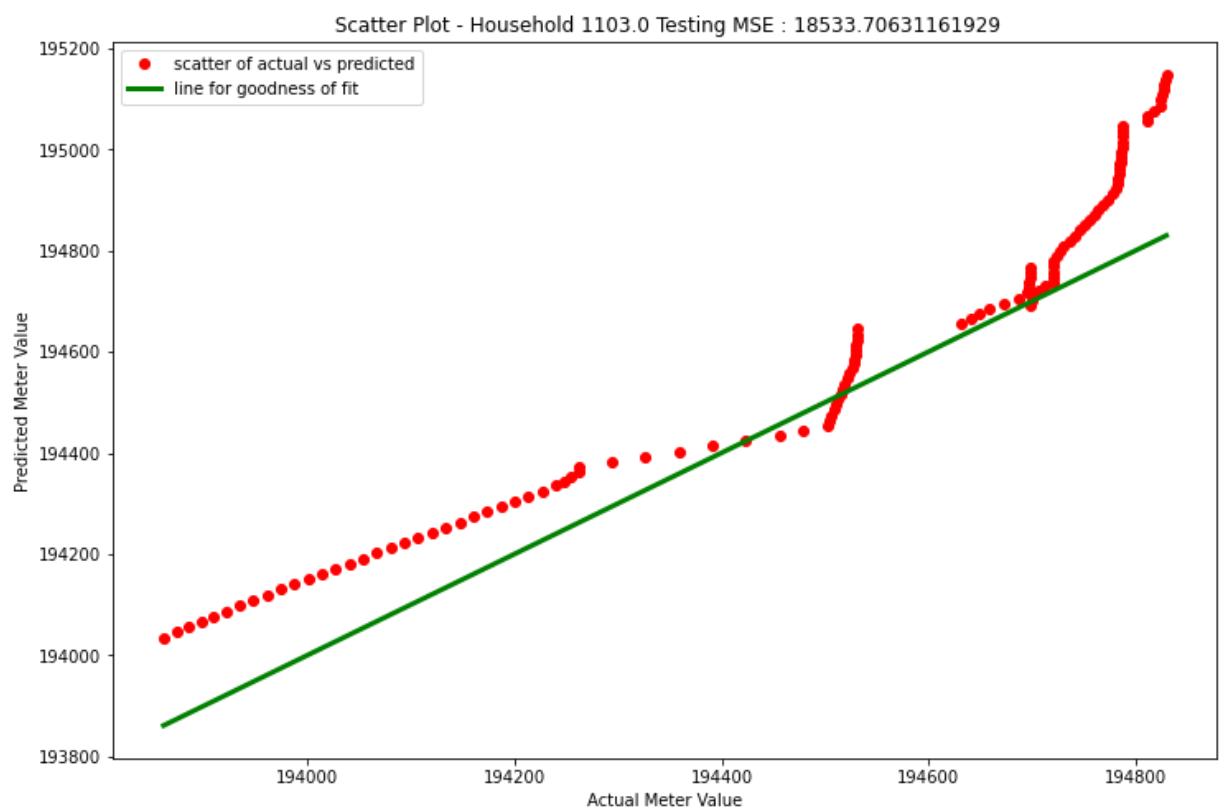
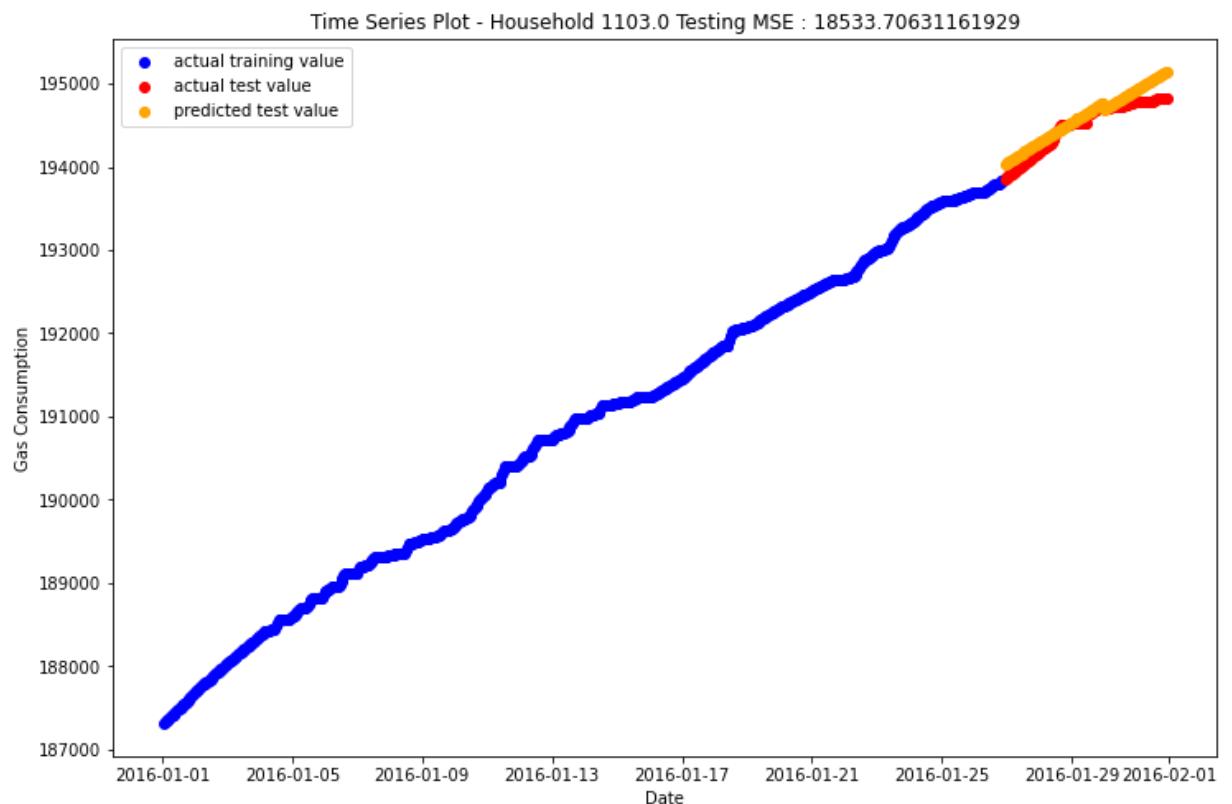
Time Series Plot - Household 1042.0 Testing MSE : 4137.700326479305

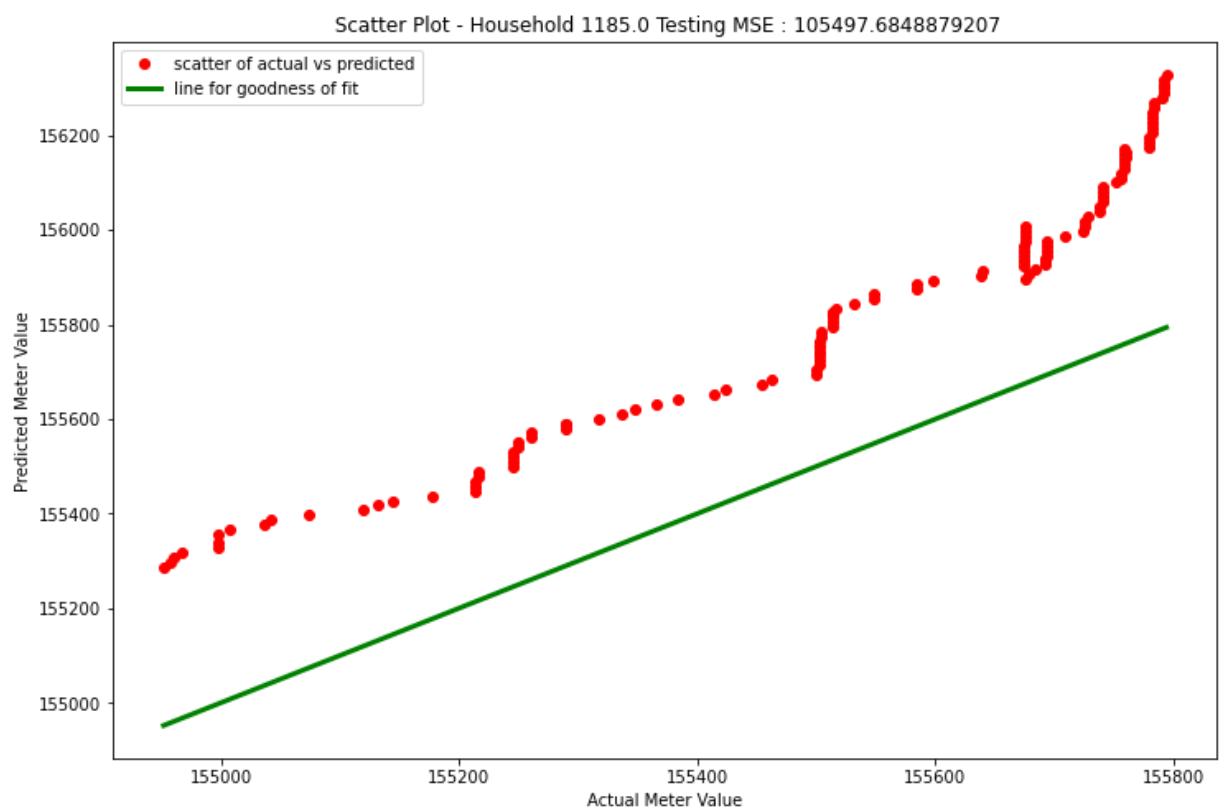
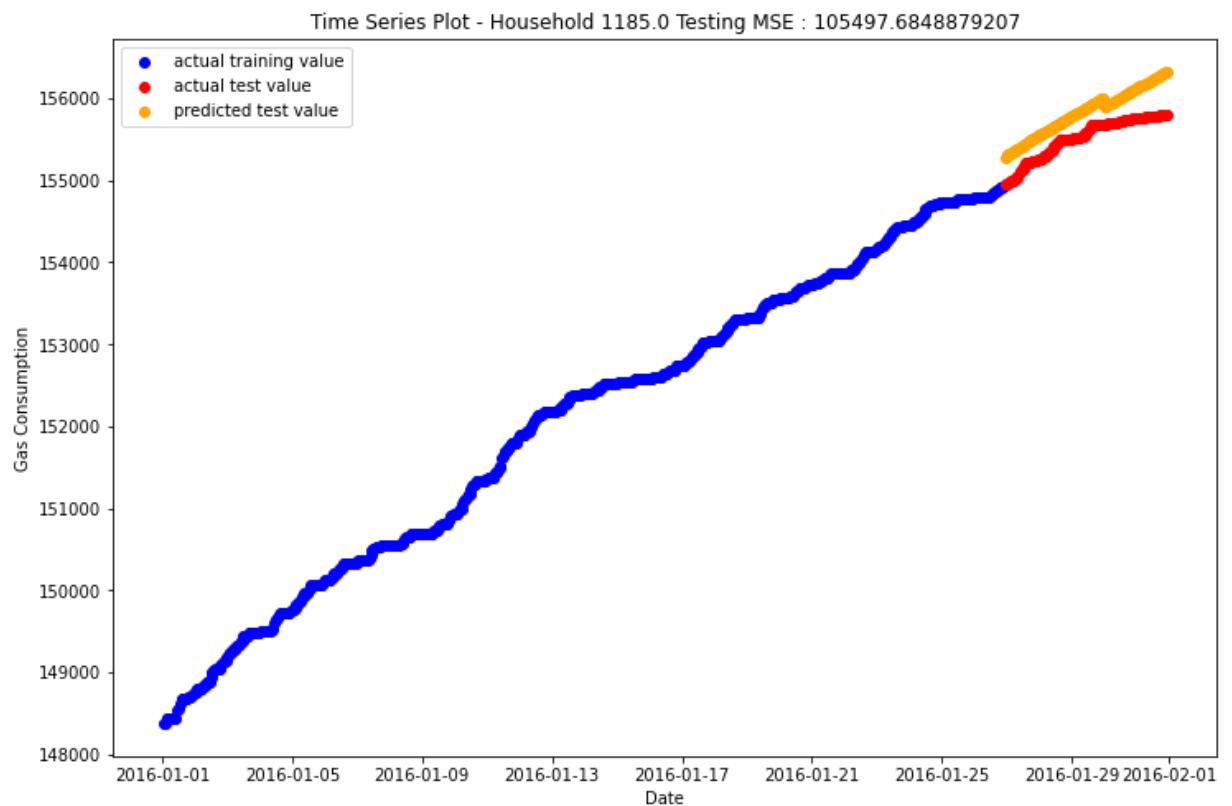


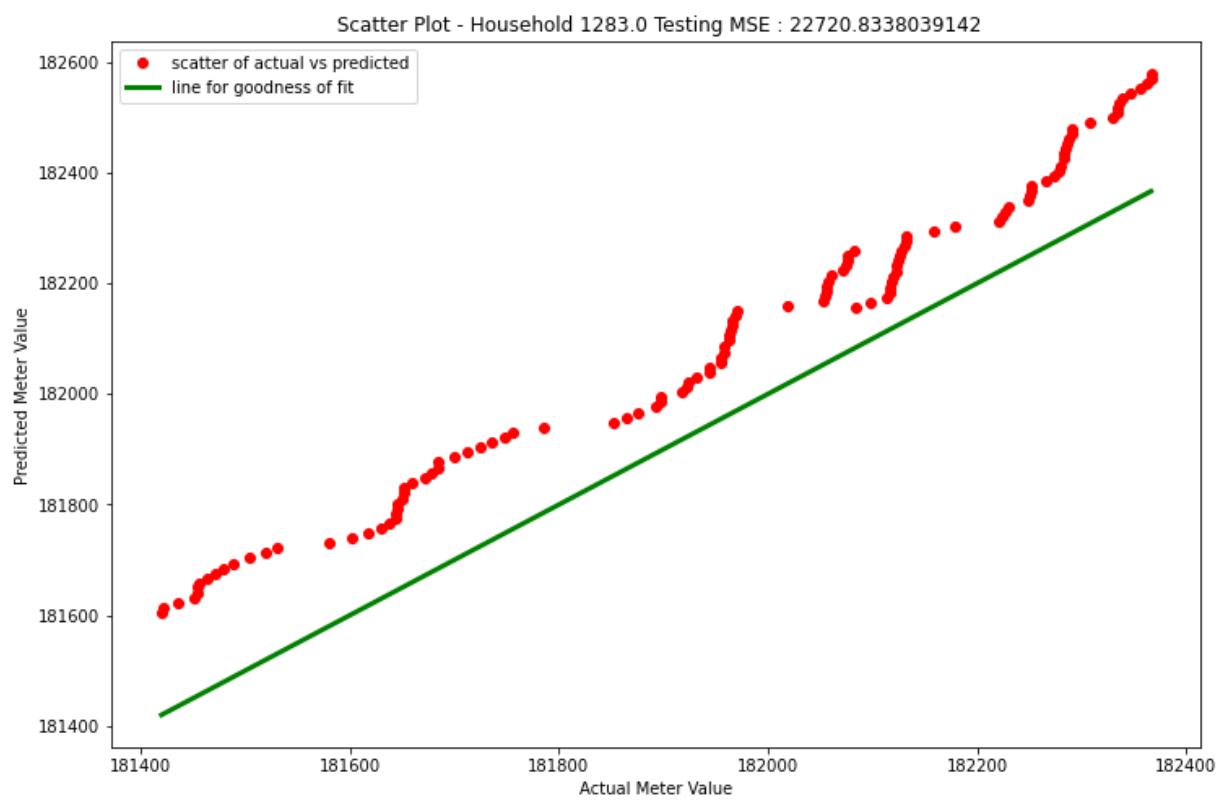
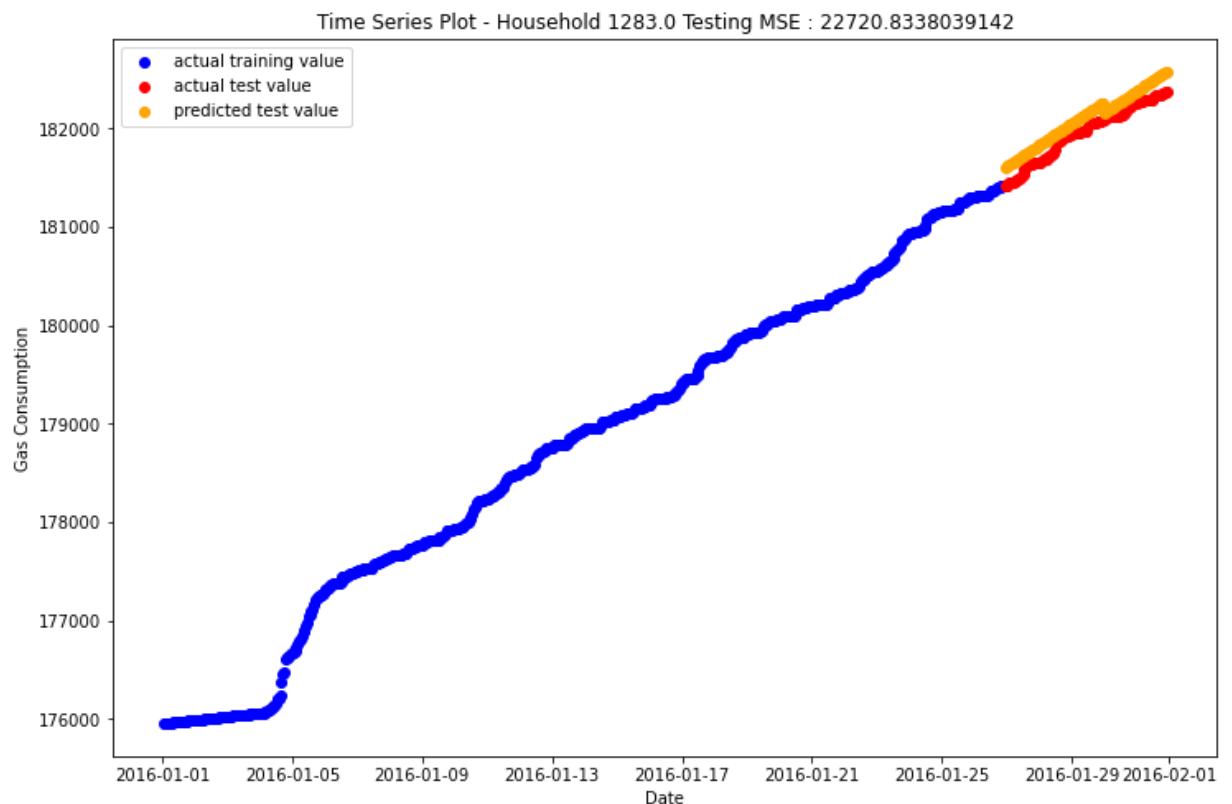
Scatter Plot - Household 1042.0 Testing MSE : 4137.700326479305

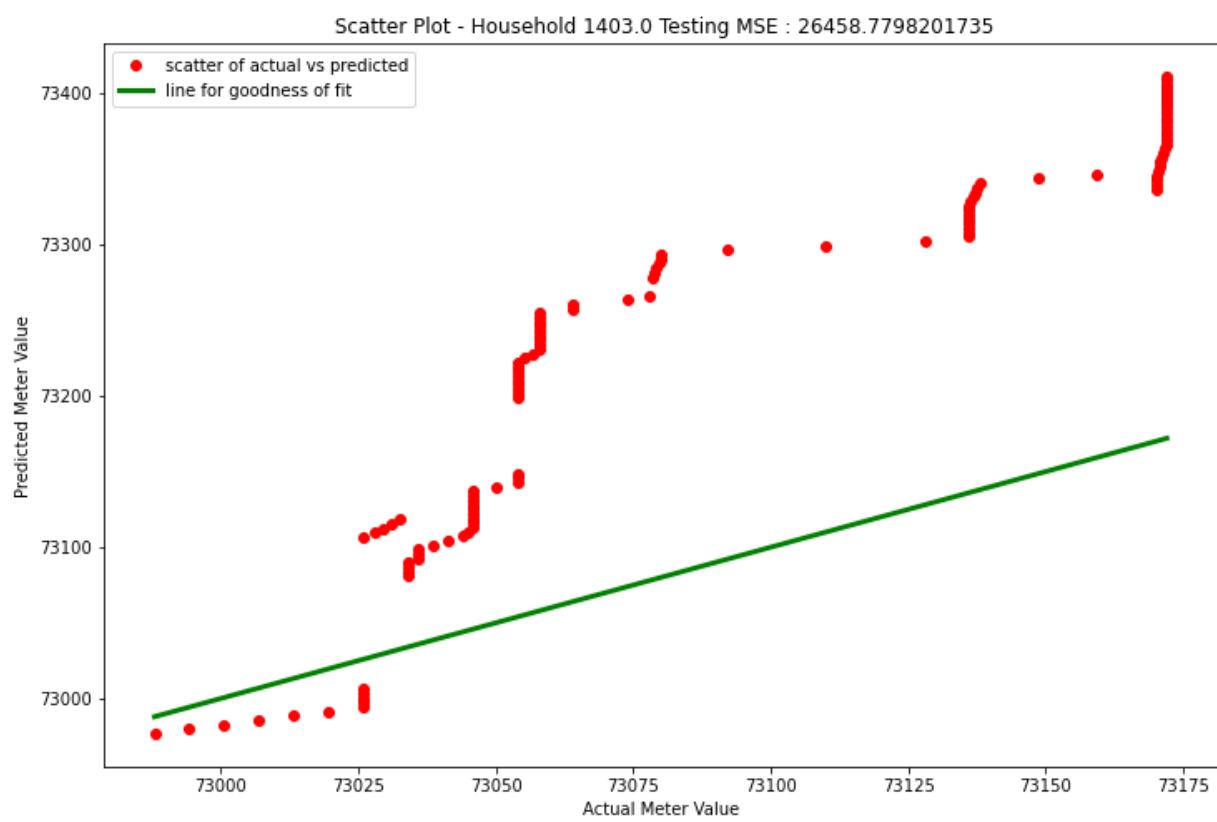
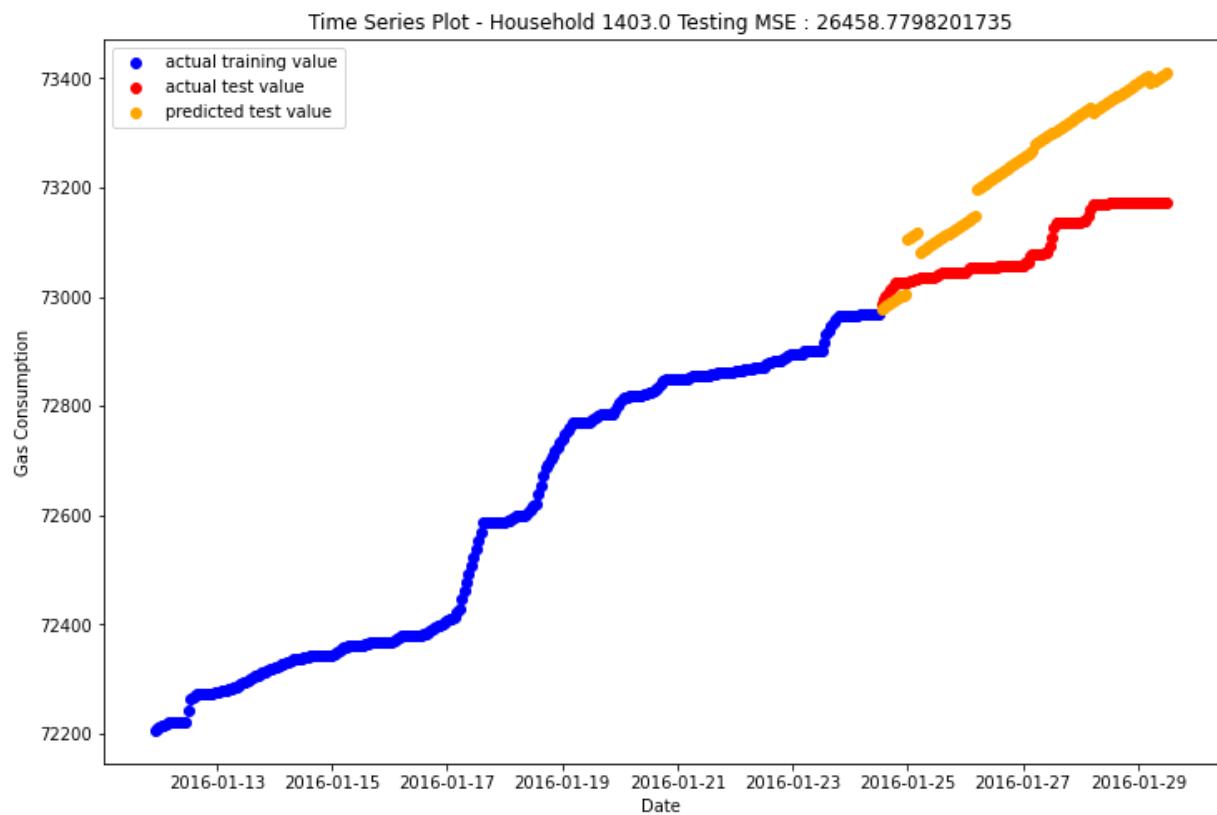


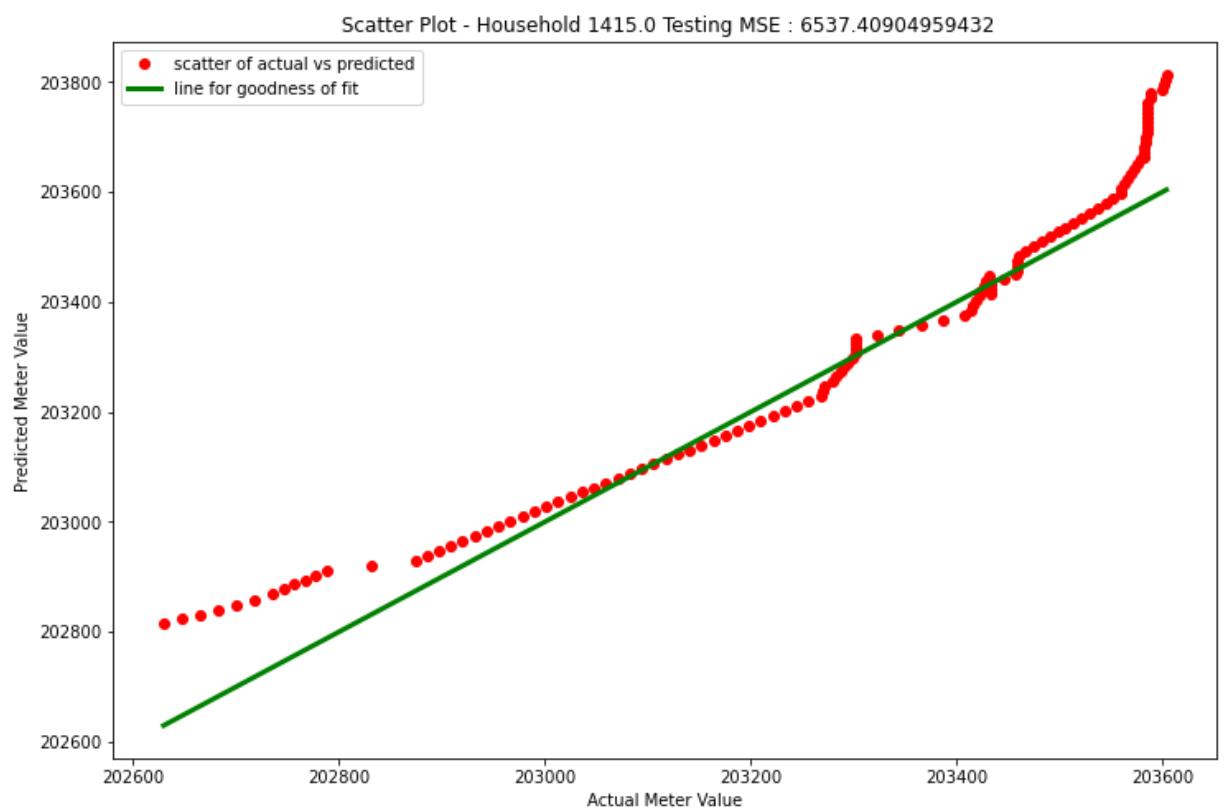
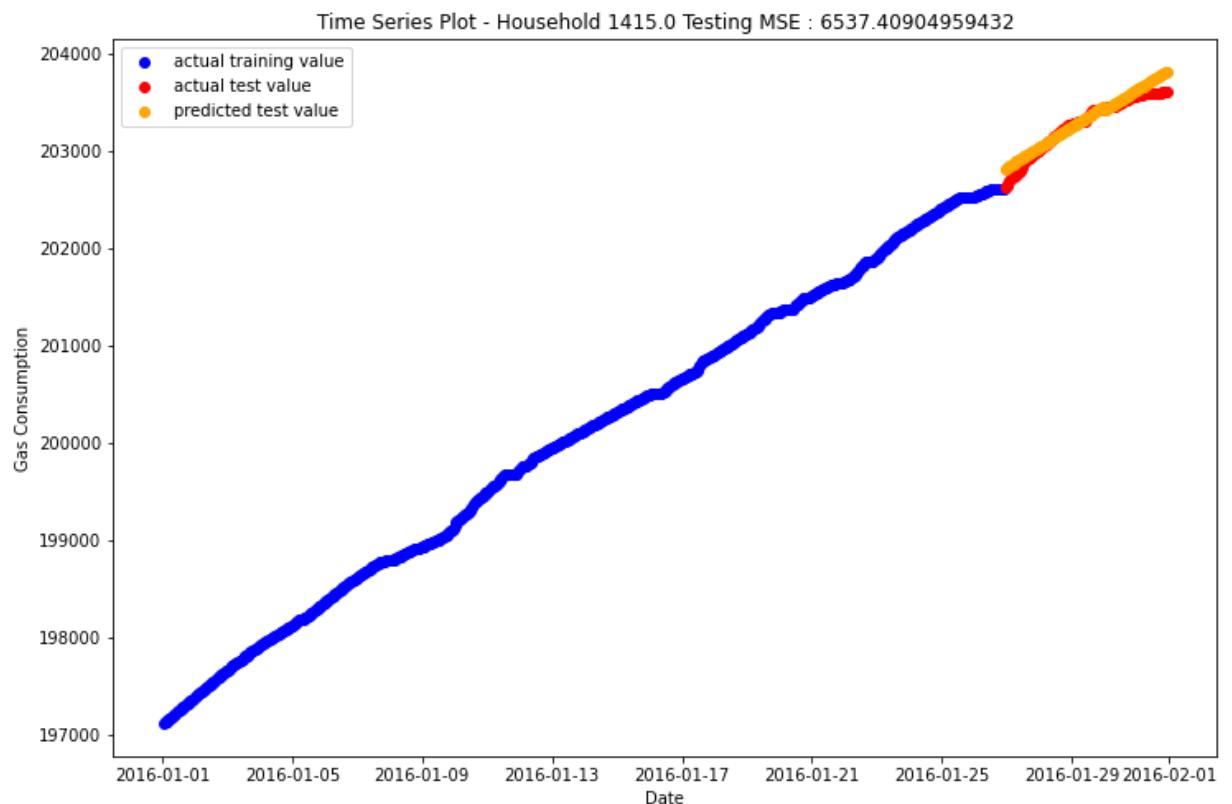


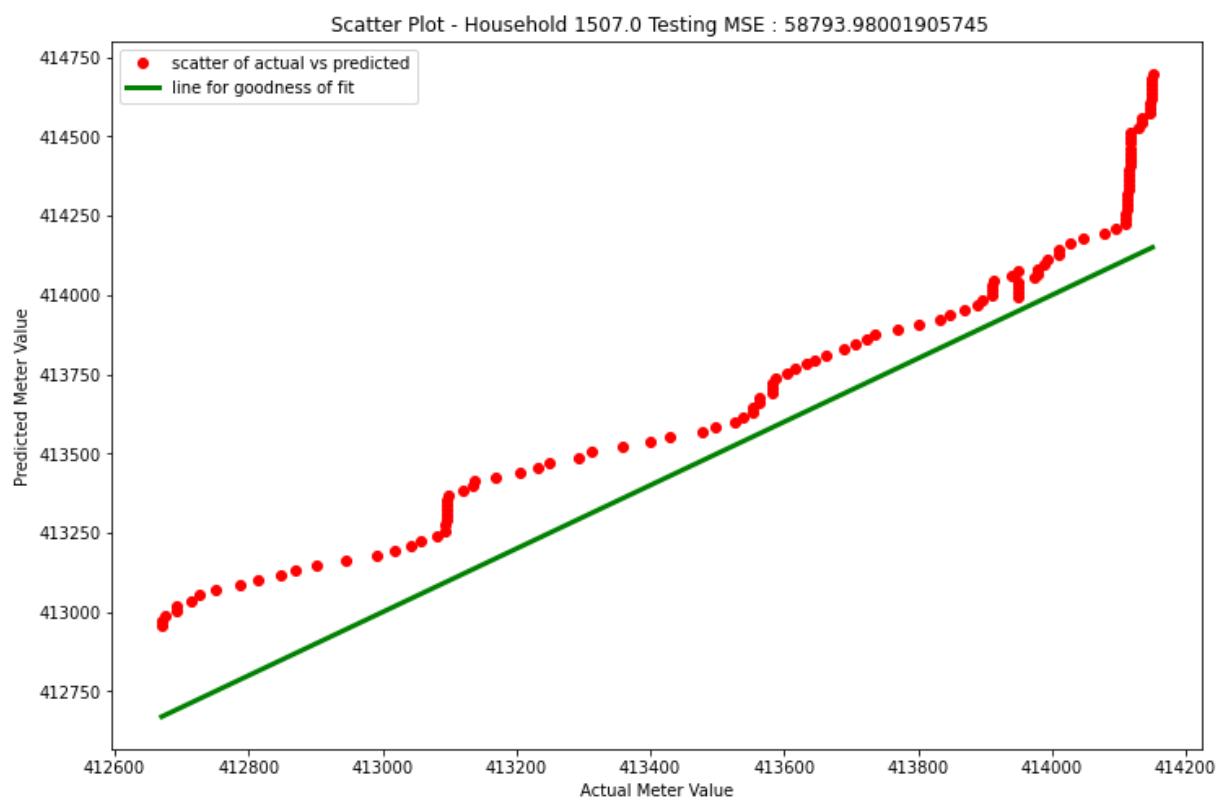
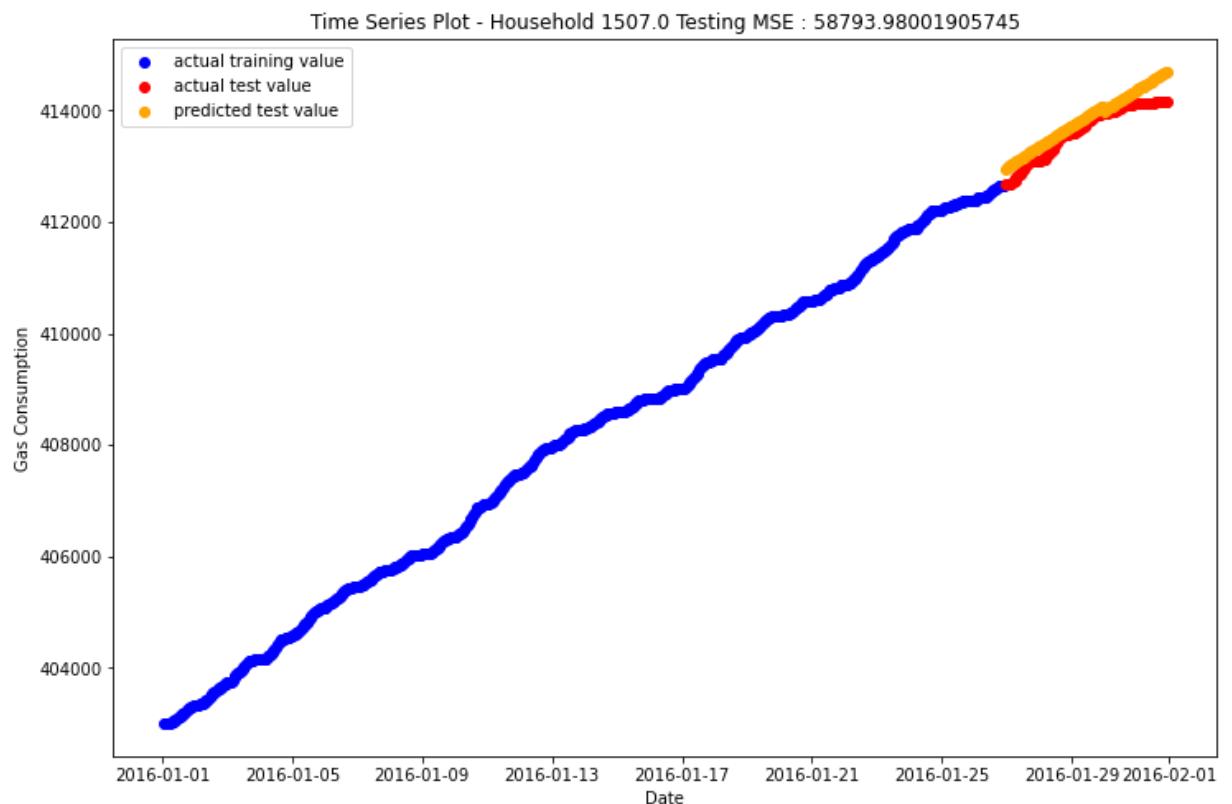




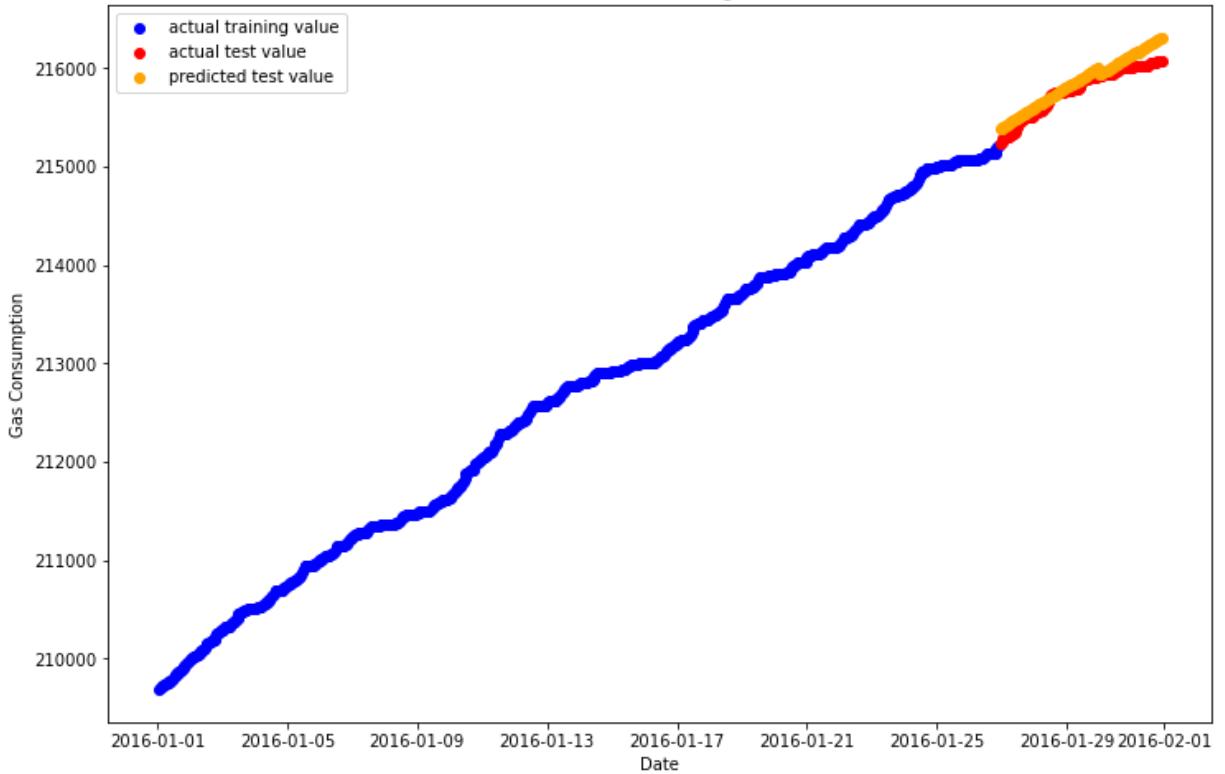




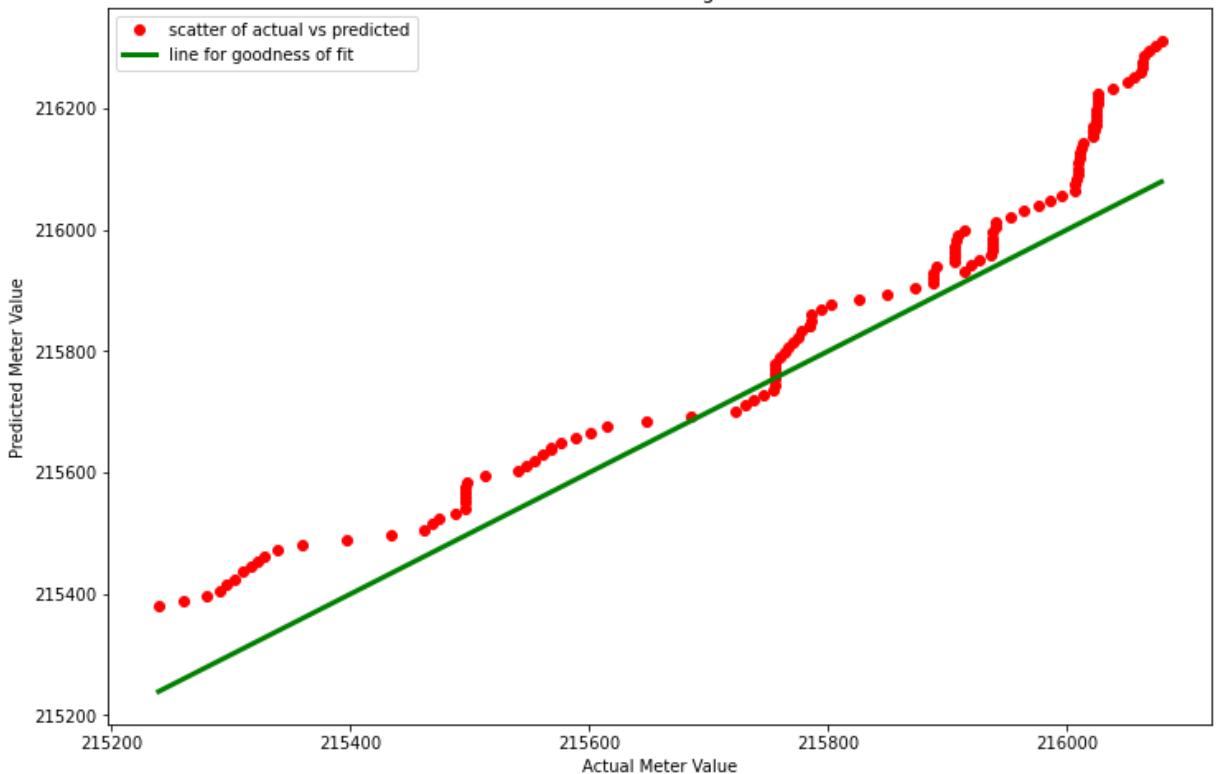


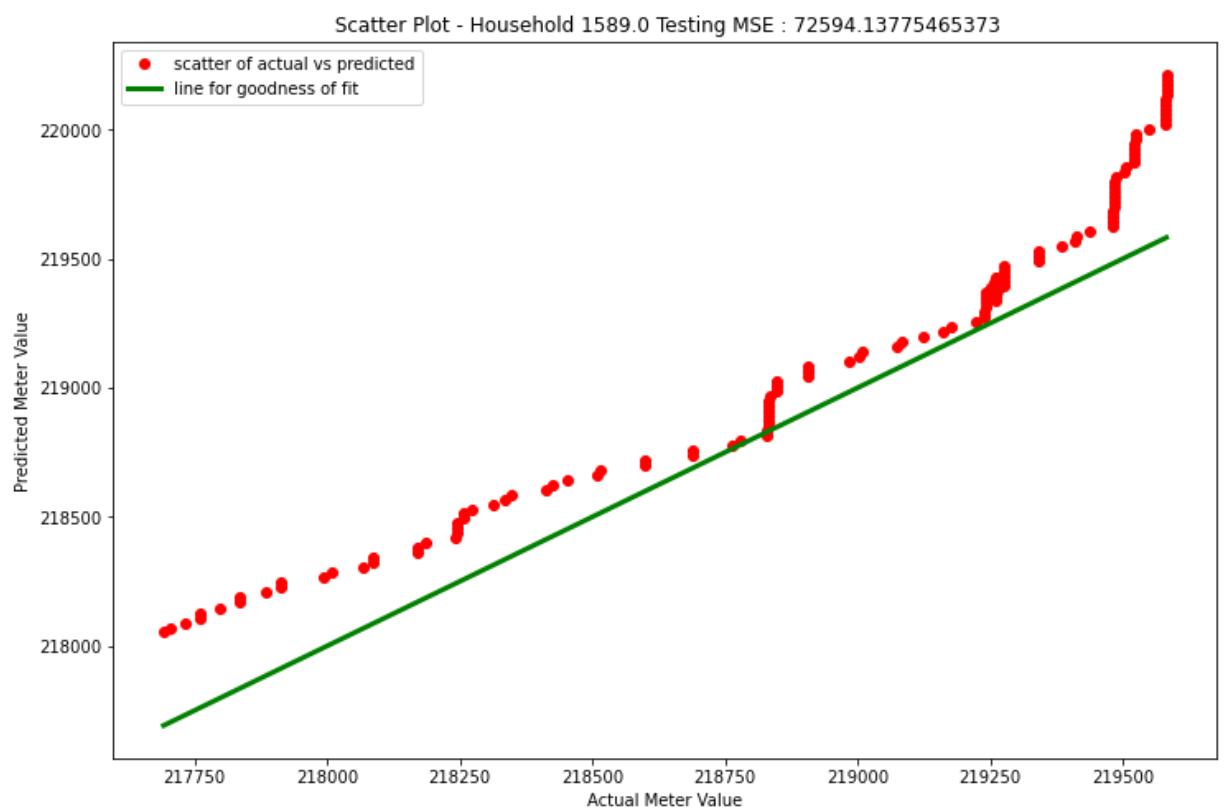
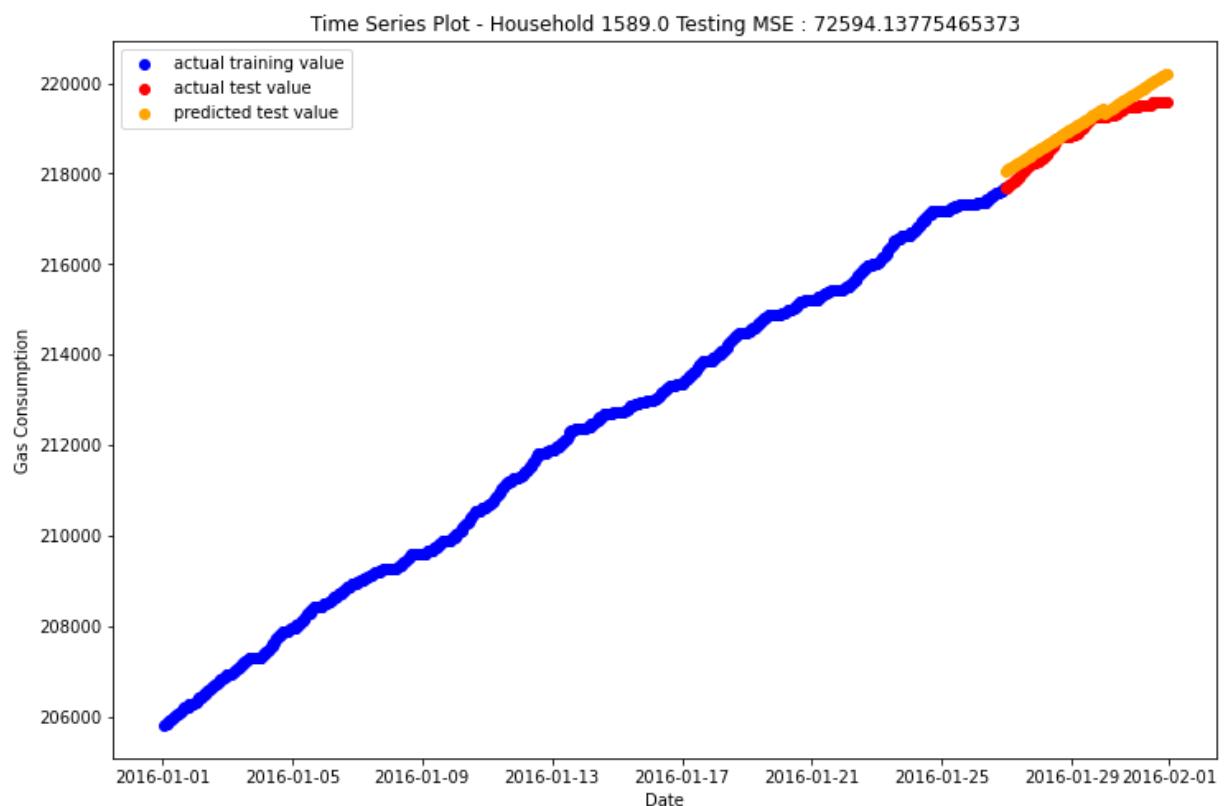


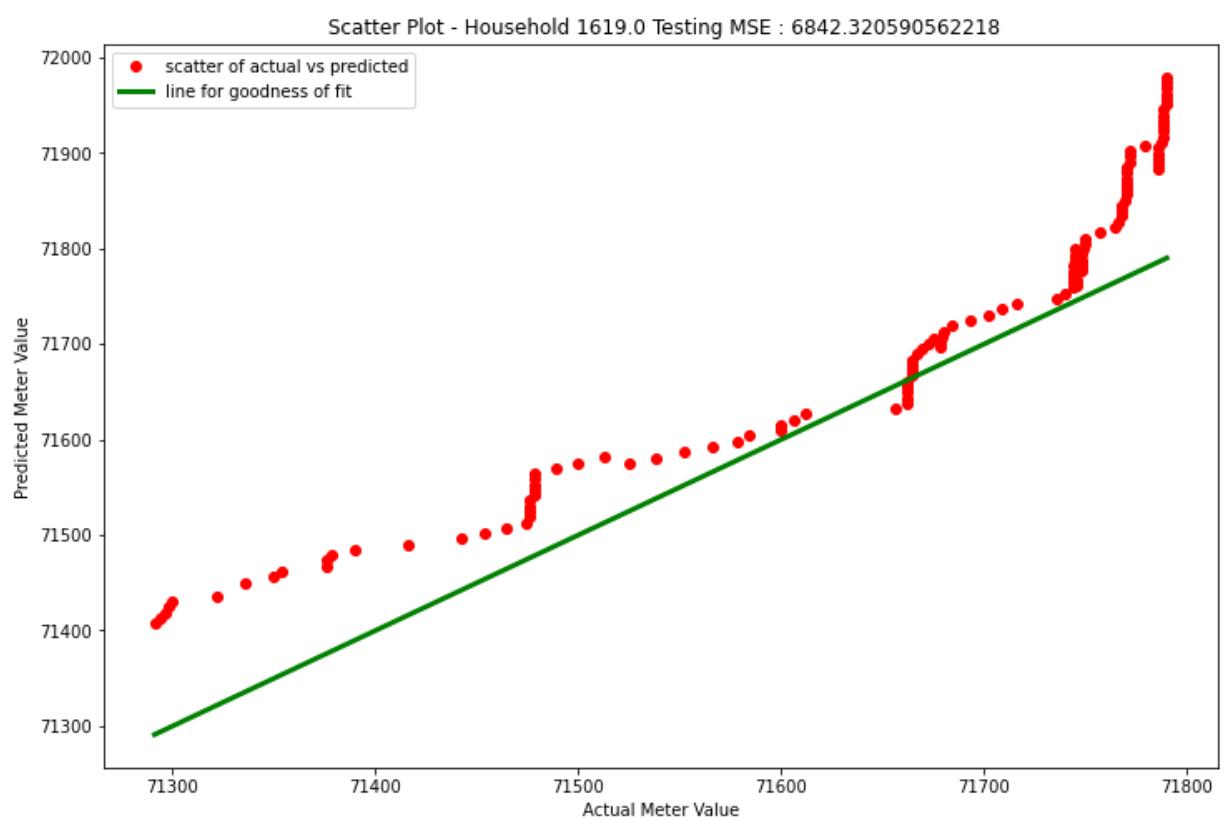
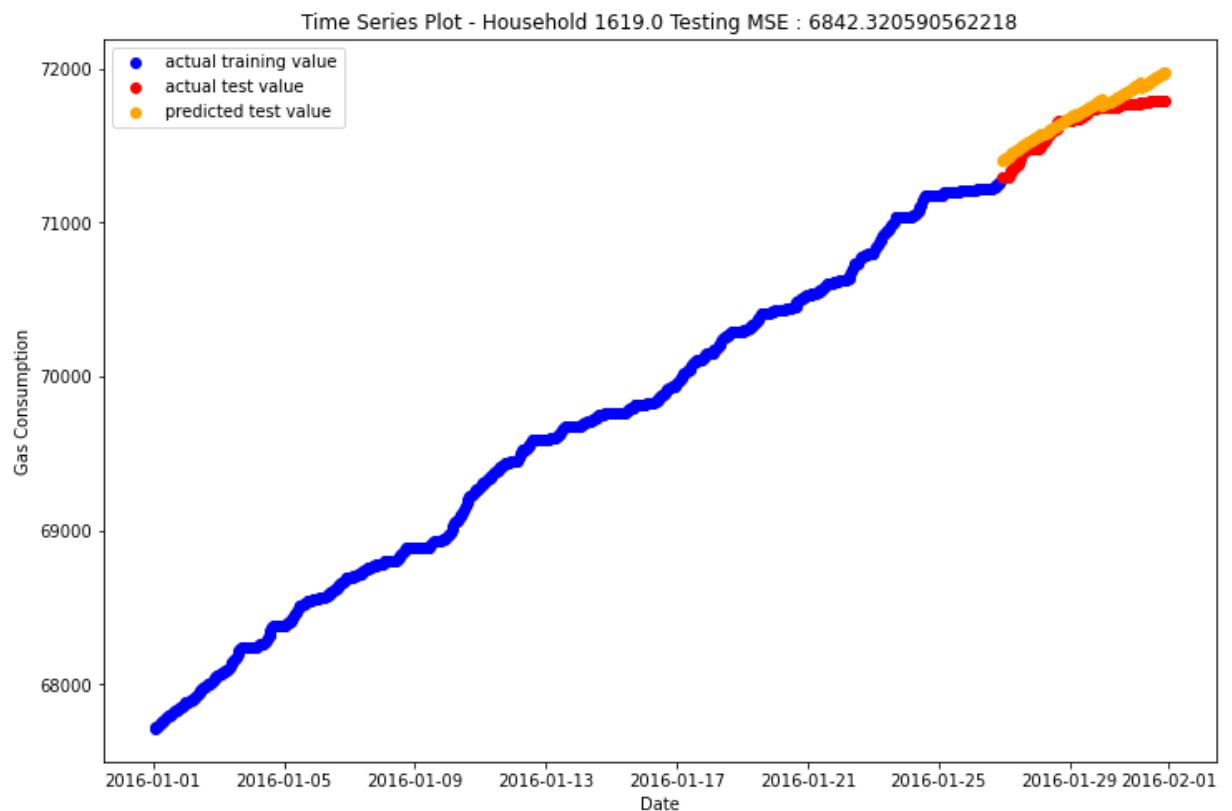
Time Series Plot - Household 1556.0 Testing MSE : 10455.783589726658

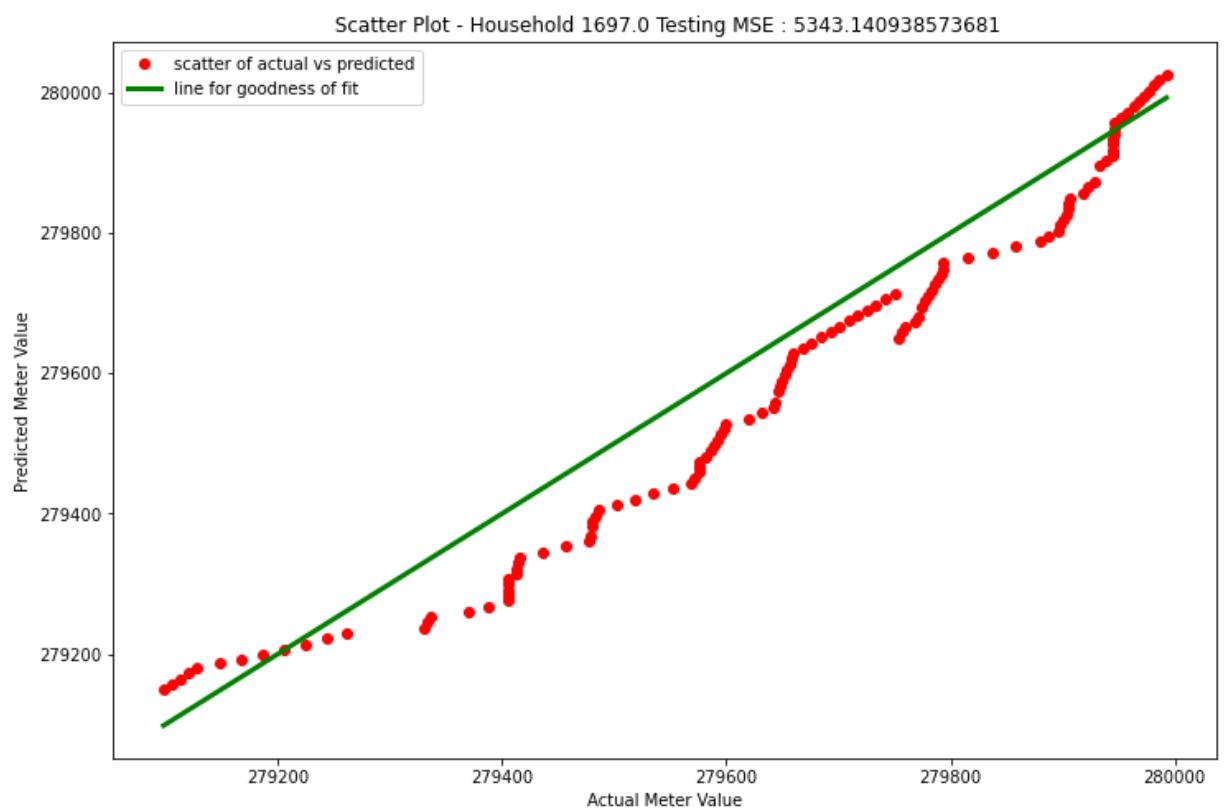
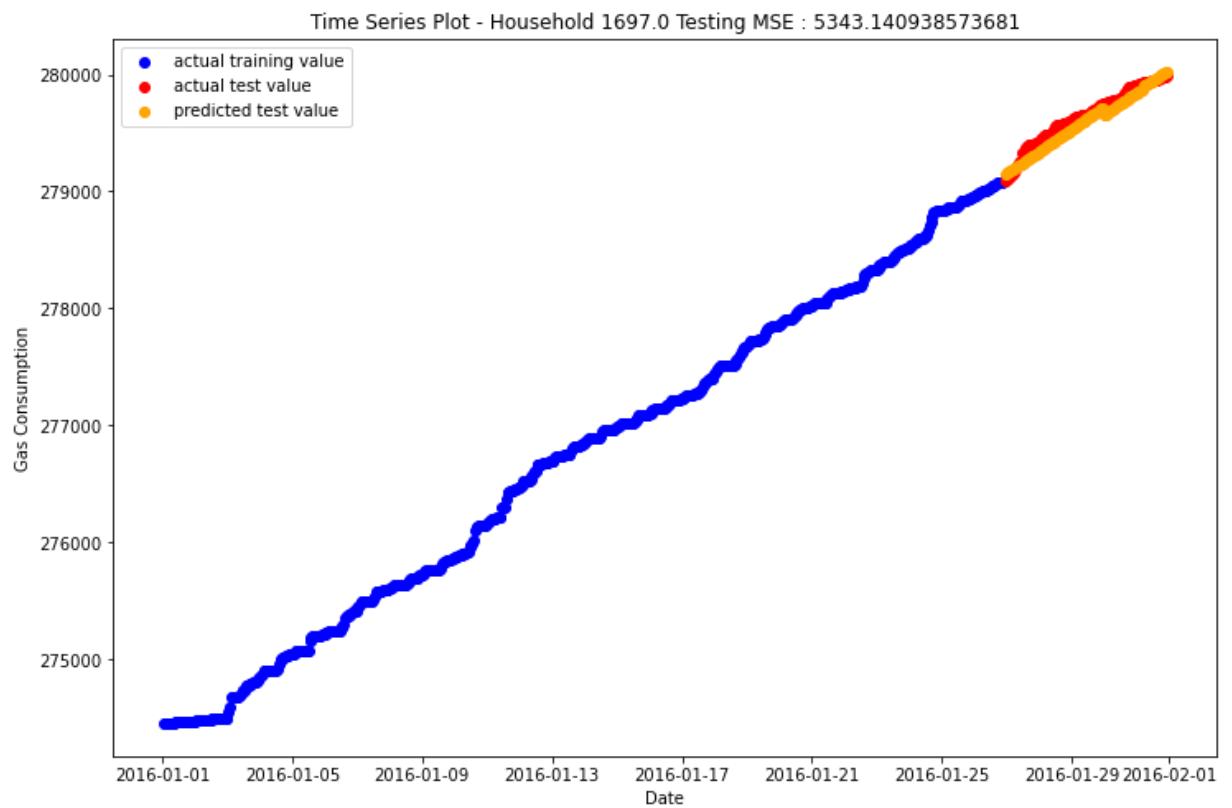


Scatter Plot - Household 1556.0 Testing MSE : 10455.783589726658

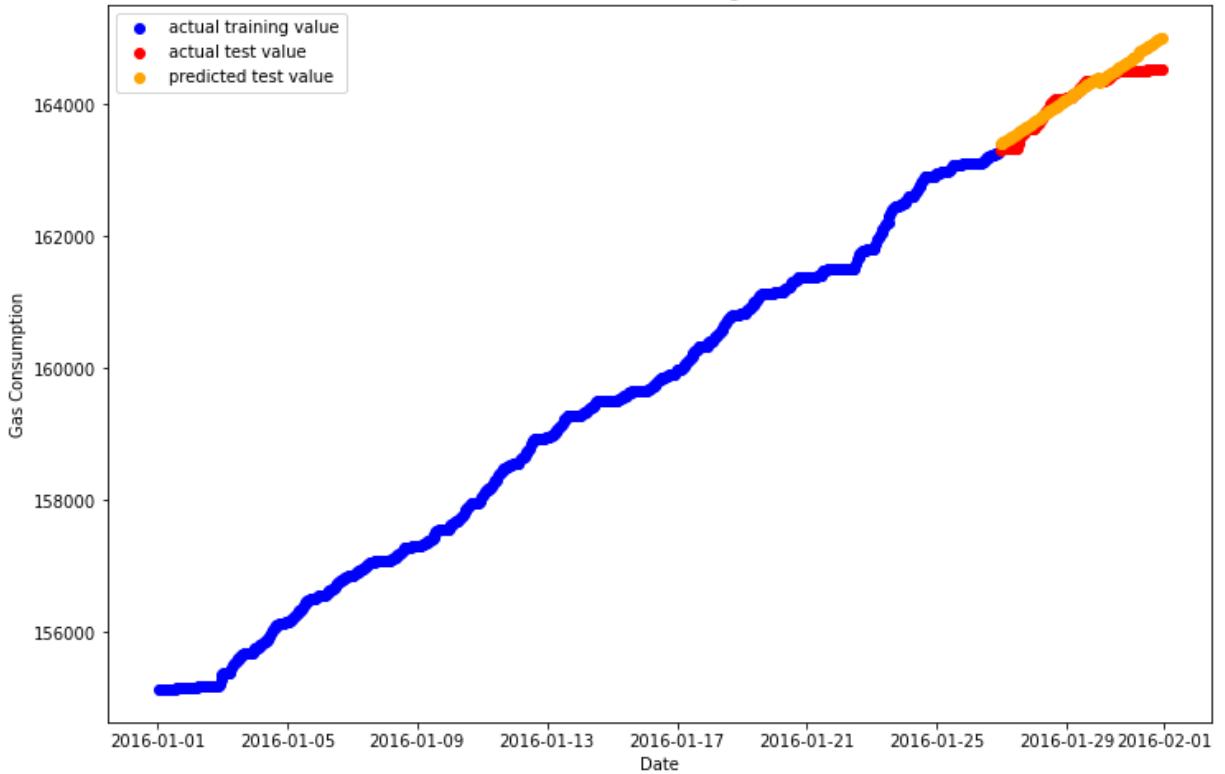




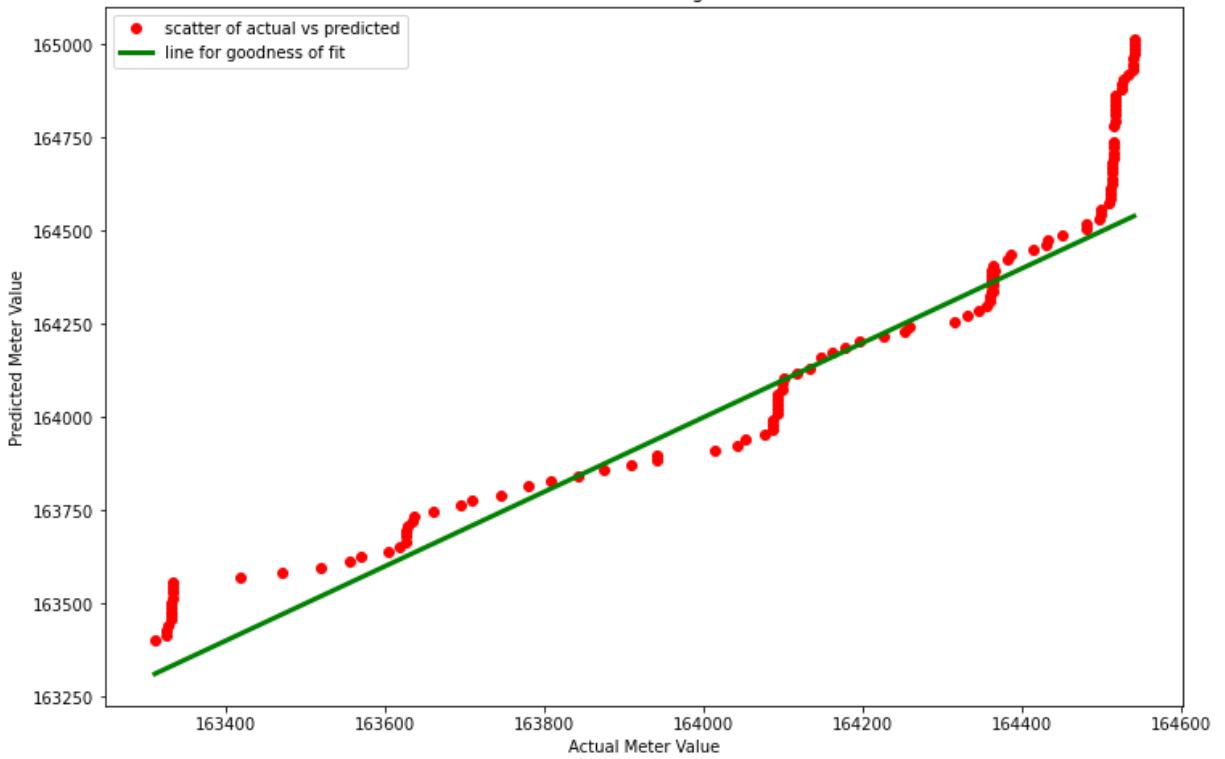


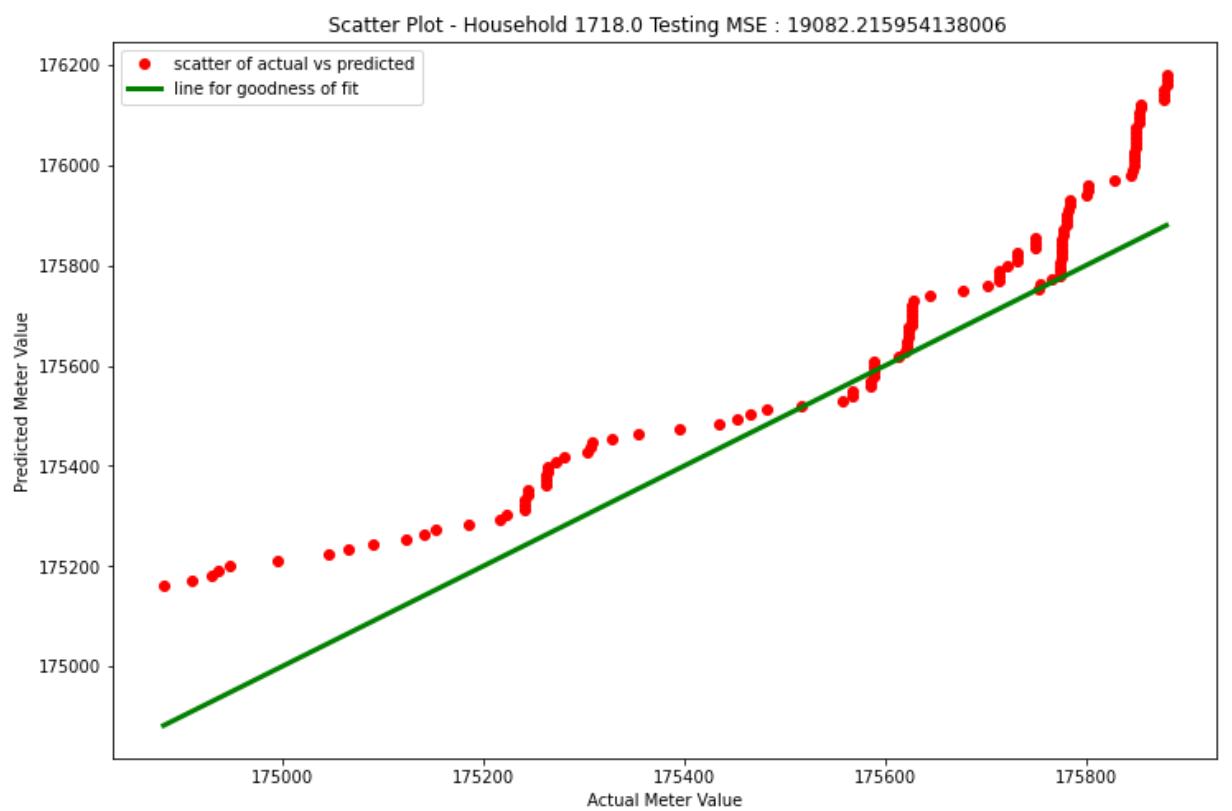
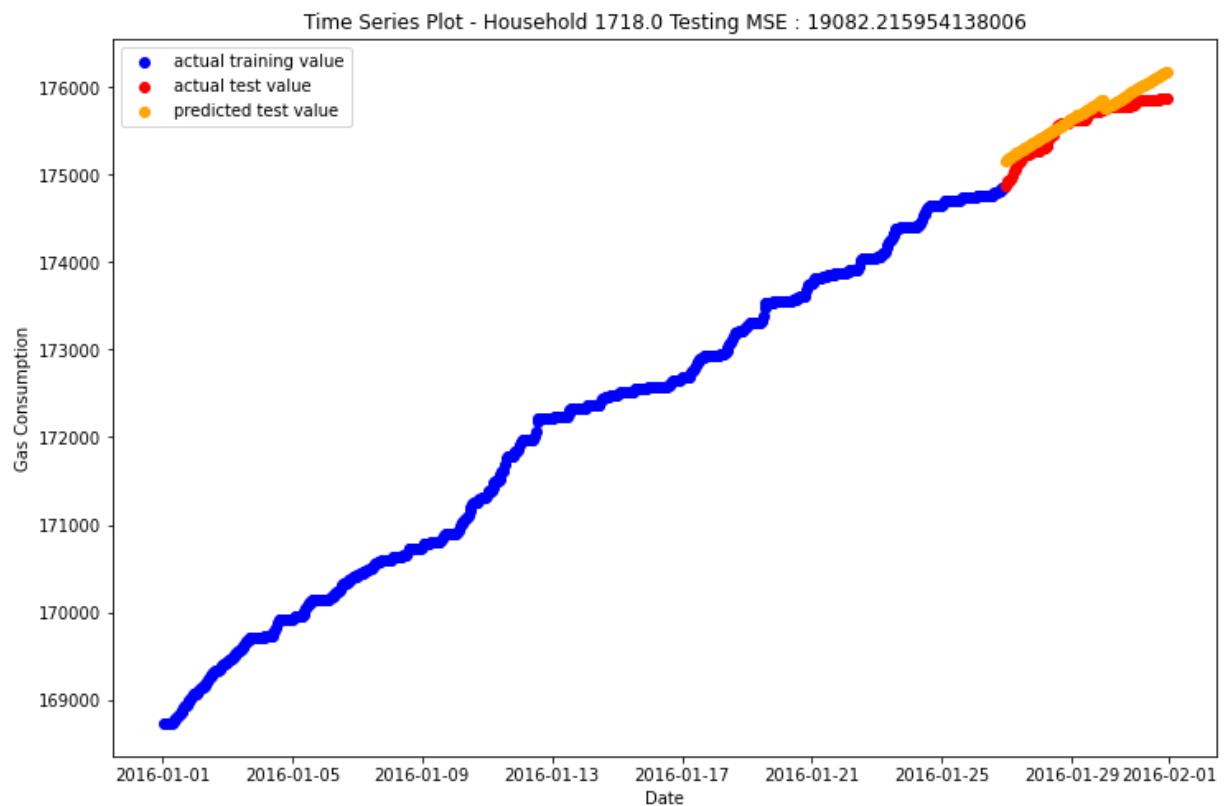


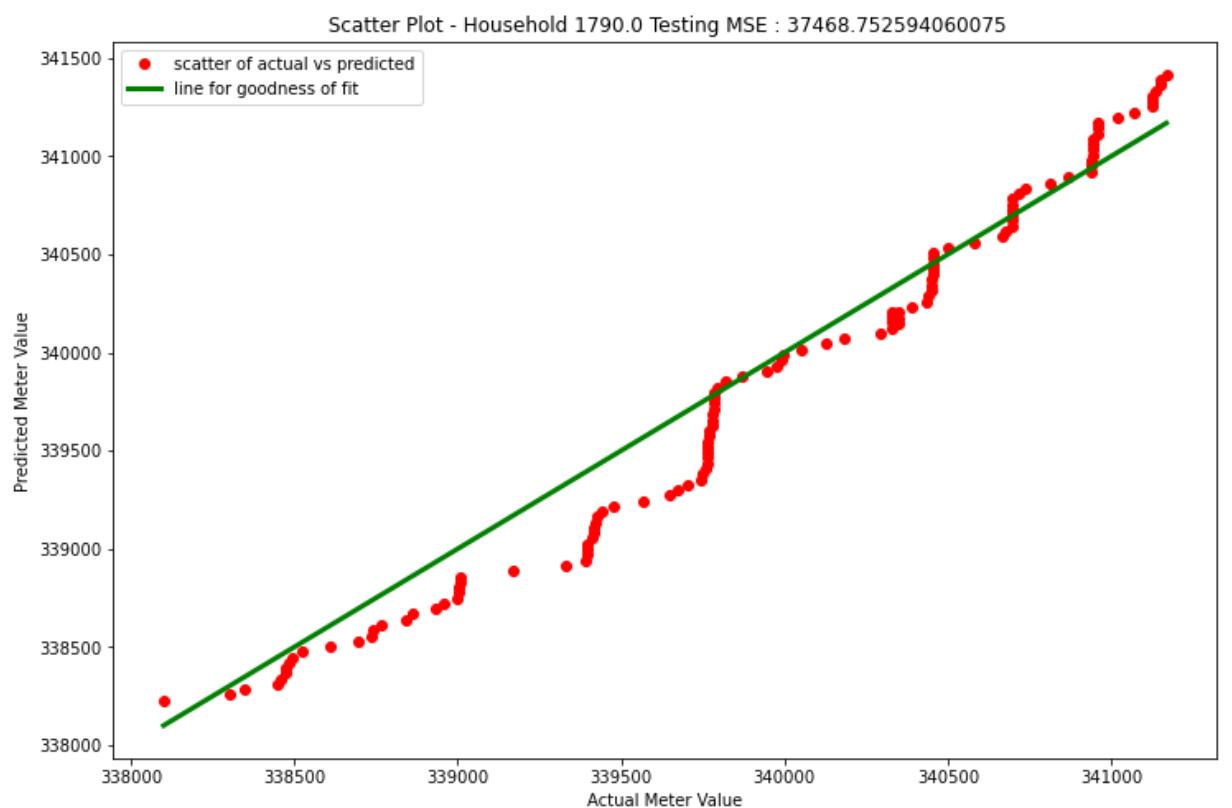
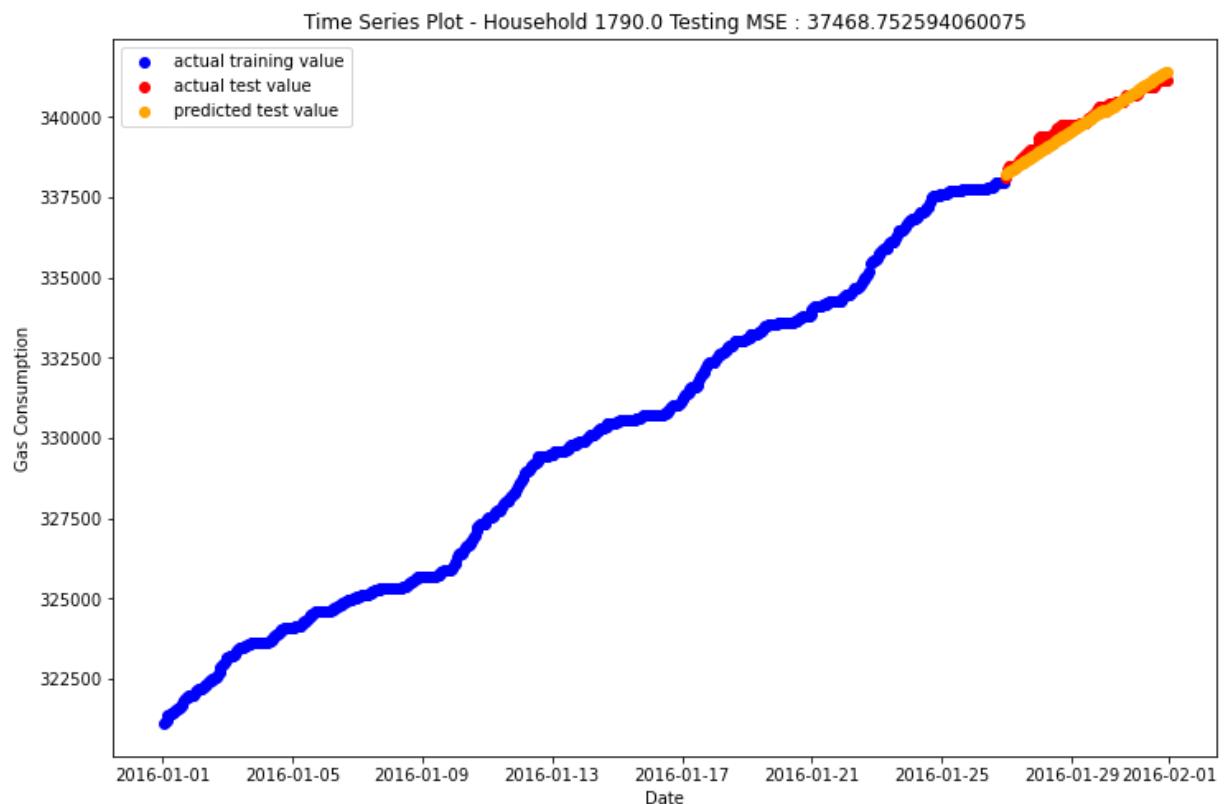
Time Series Plot - Household 1714.0 Testing MSE : 28625.33439221291



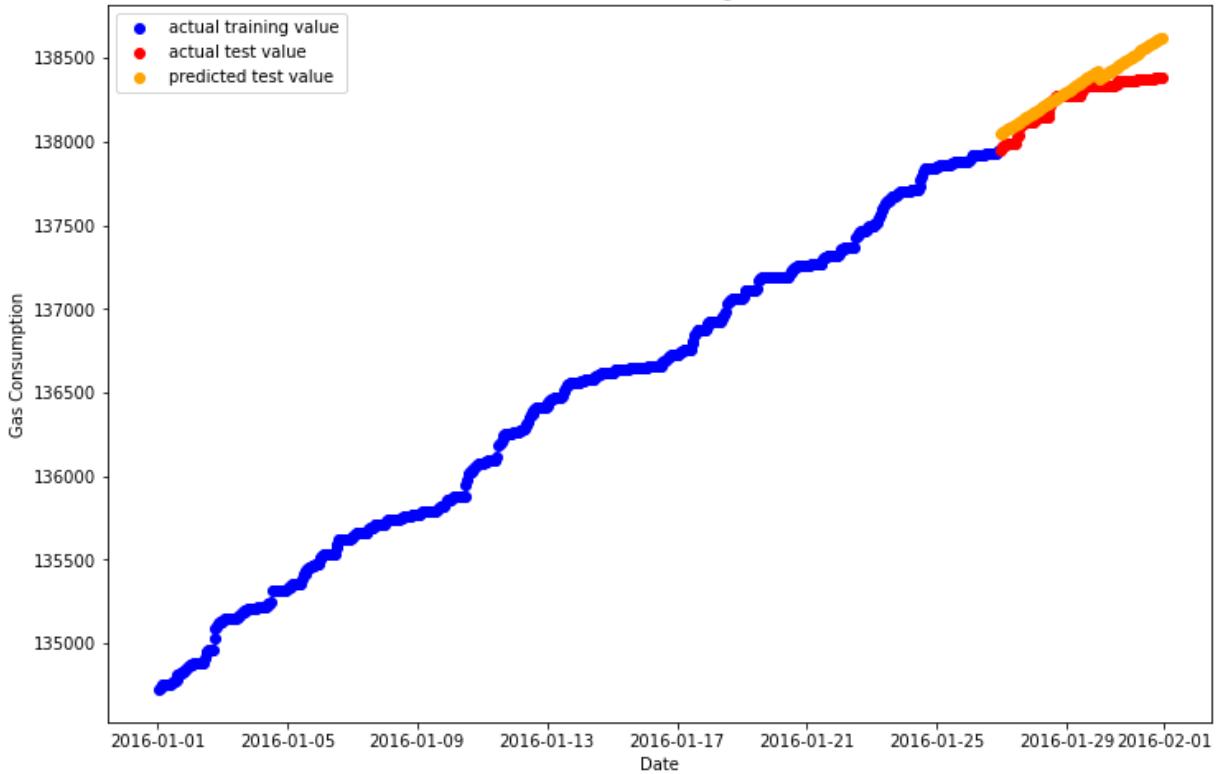
Scatter Plot - Household 1714.0 Testing MSE : 28625.33439221291



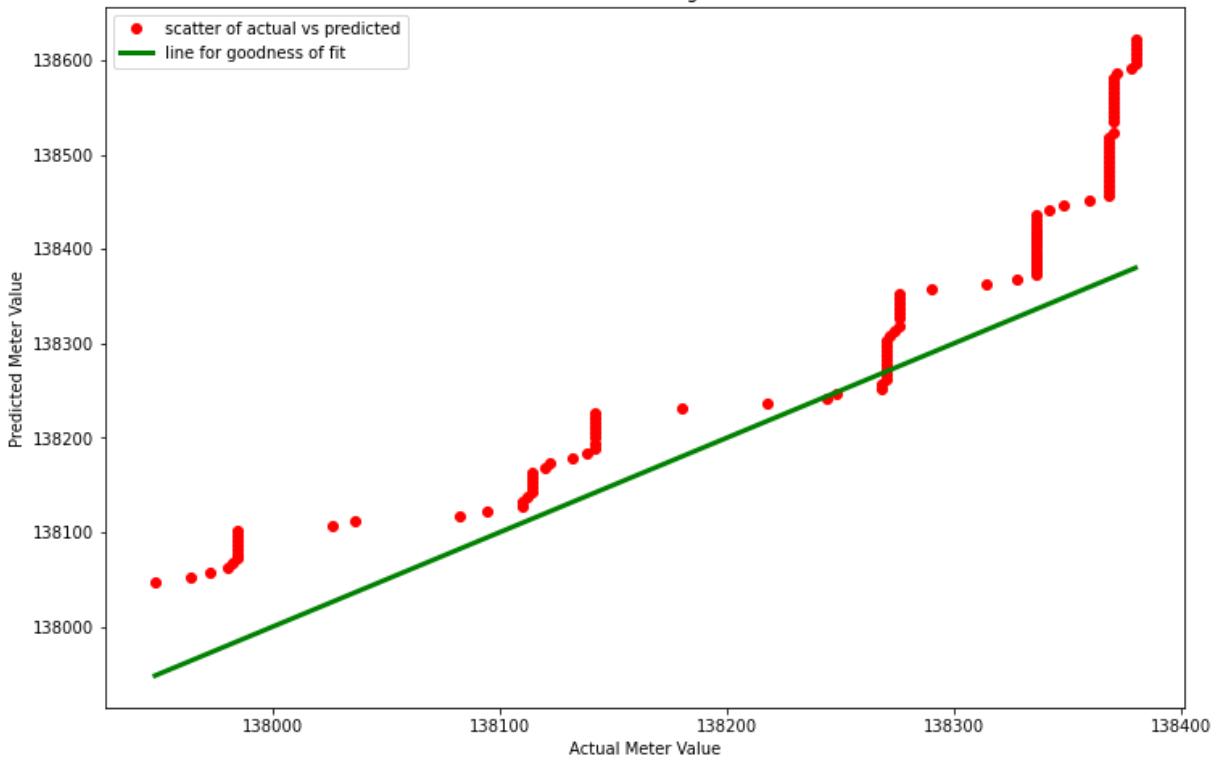




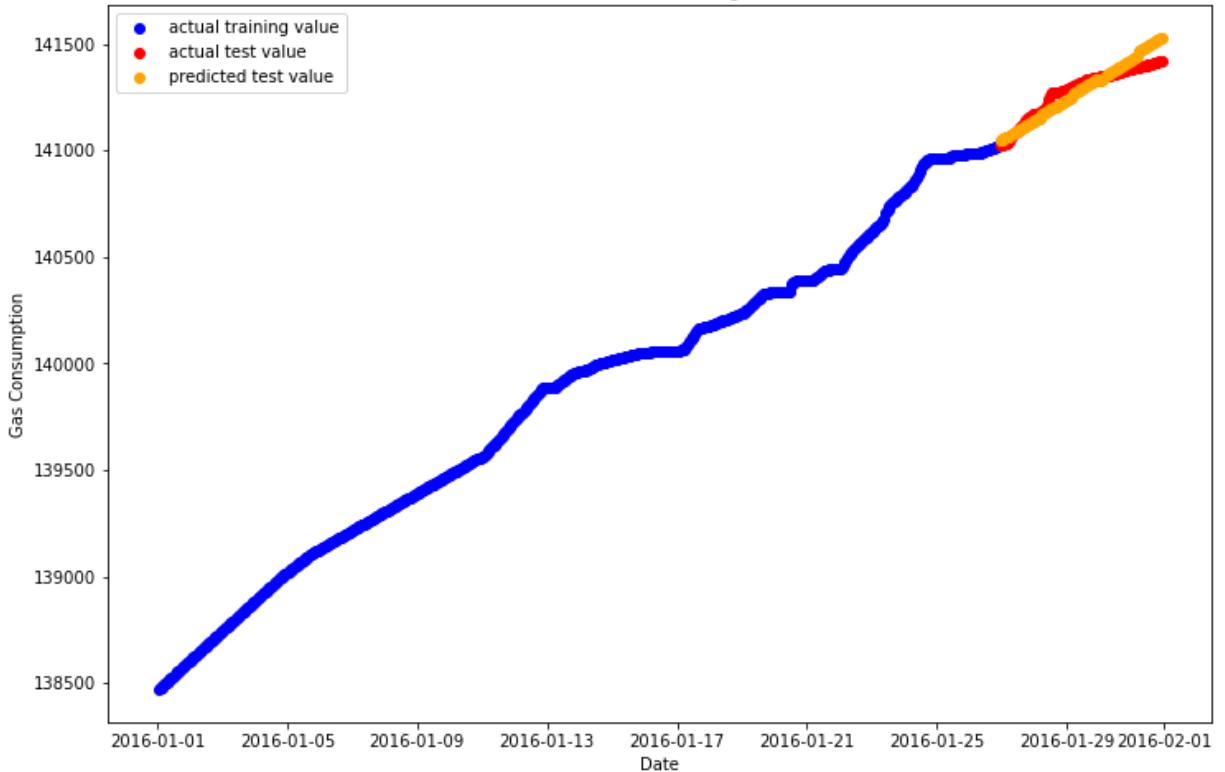
Time Series Plot - Household 1791.0 Testing MSE : 11176.973462160273



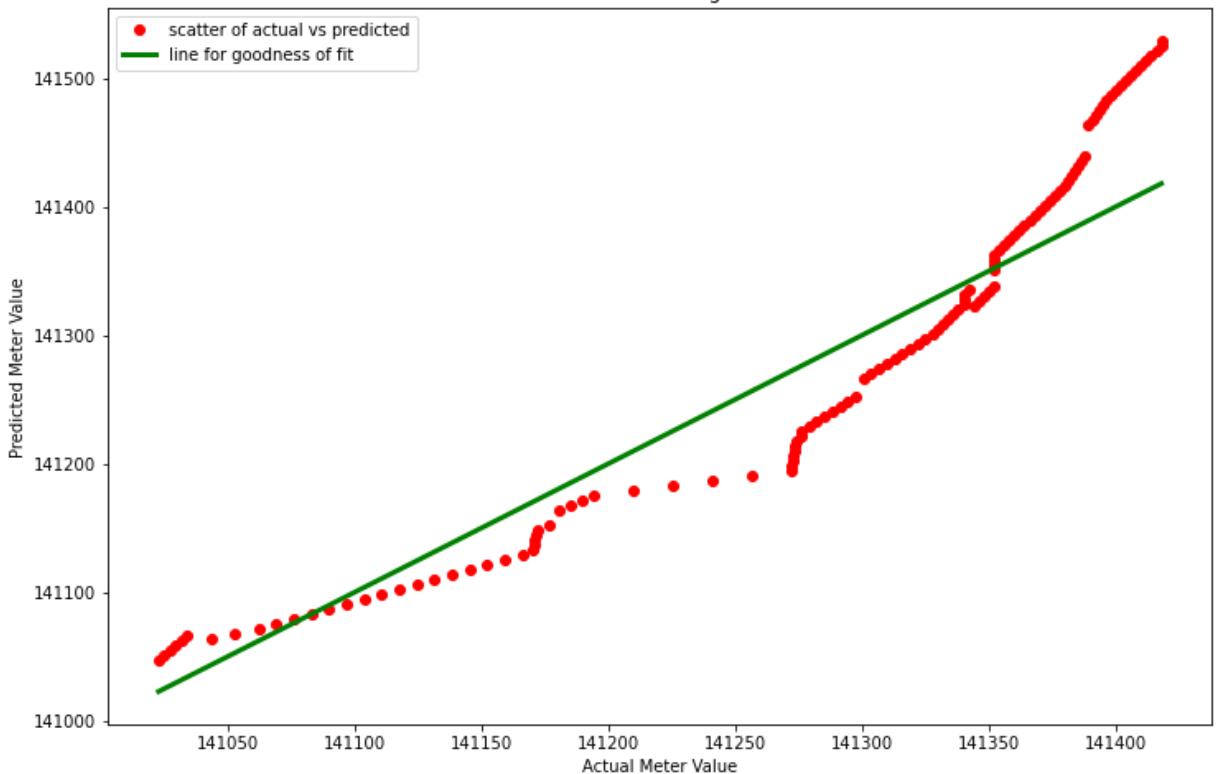
Scatter Plot - Household 1791.0 Testing MSE : 11176.973462160273

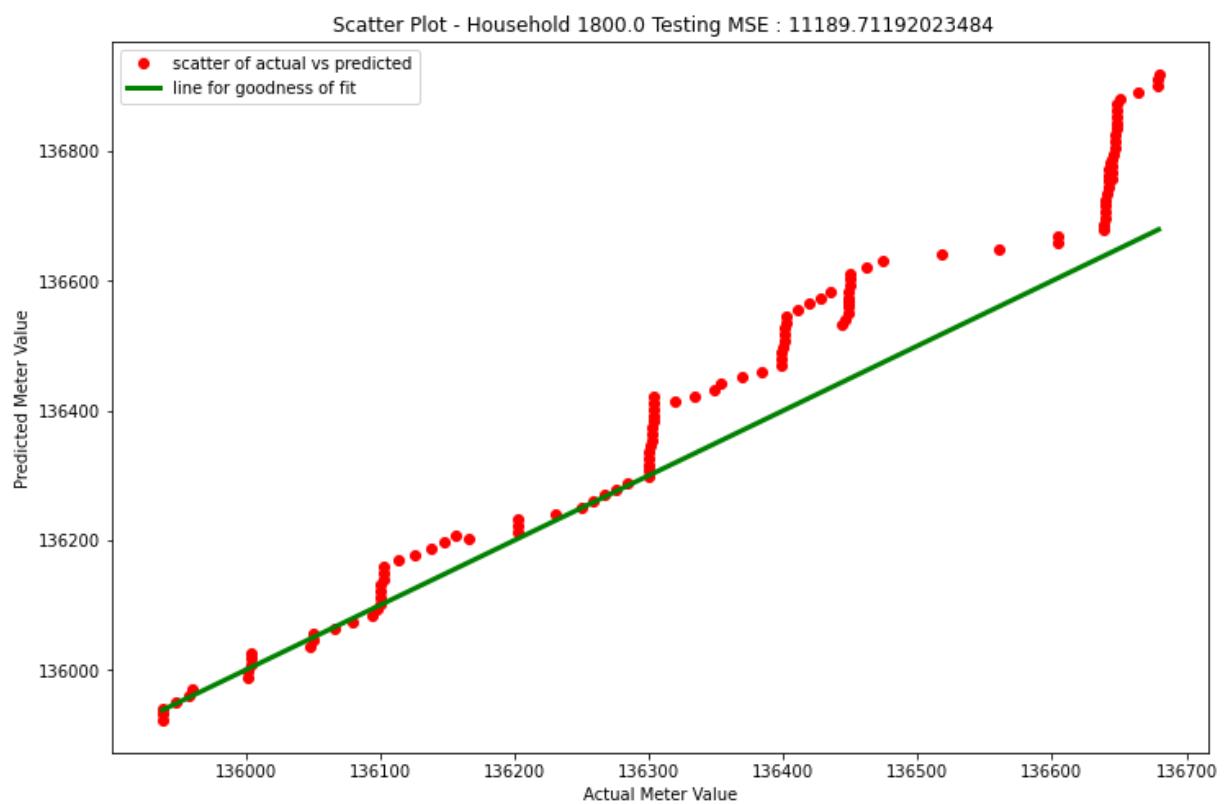
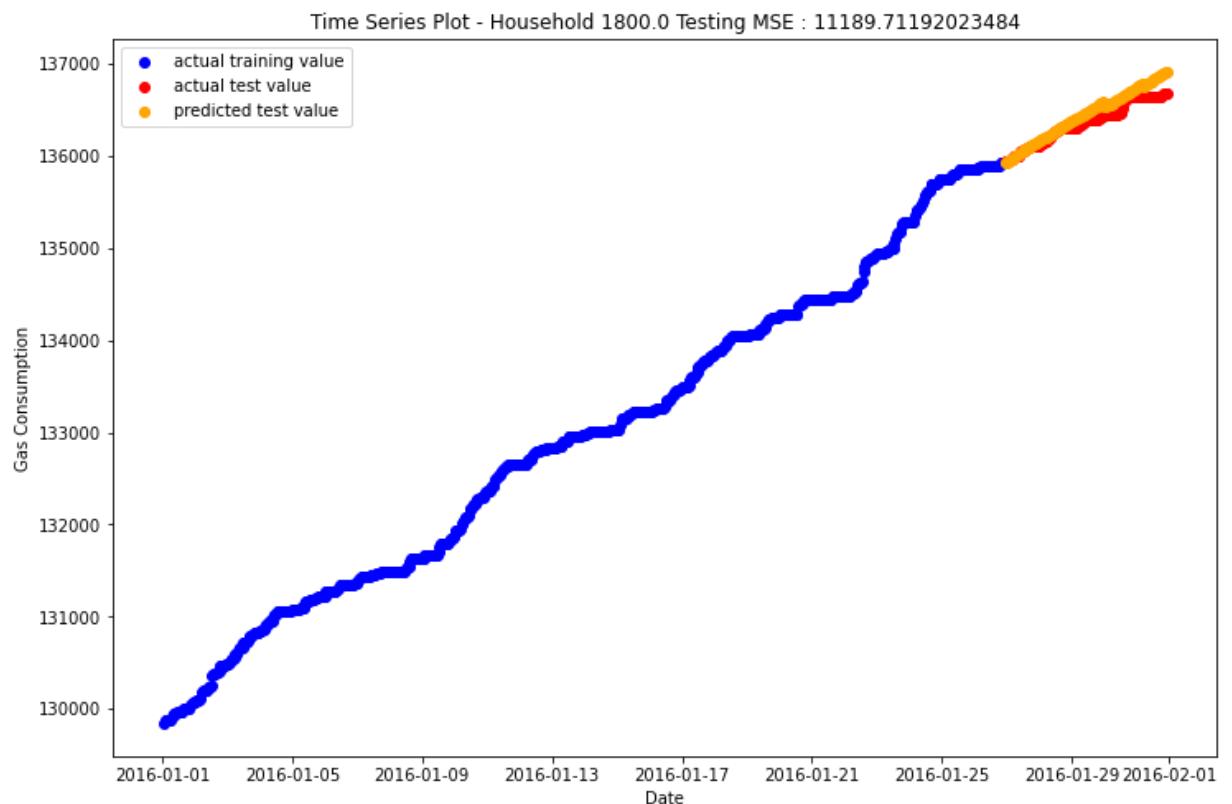


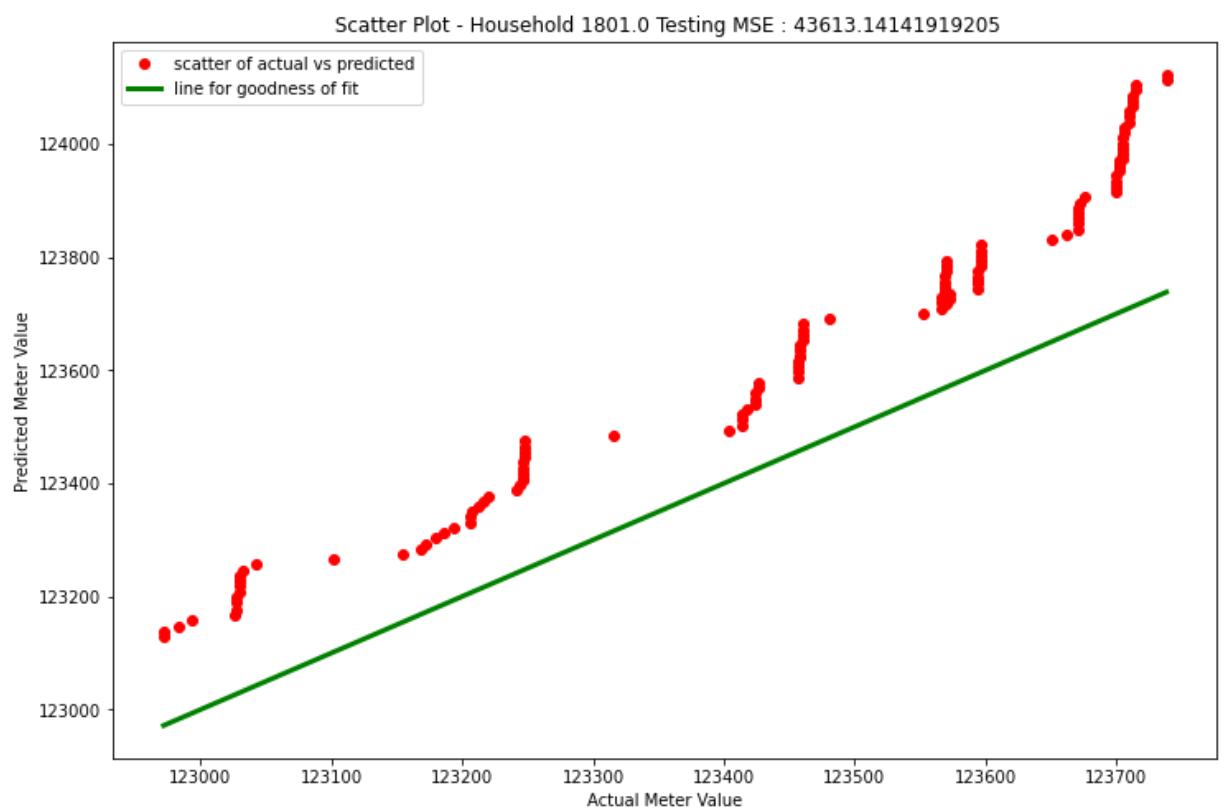
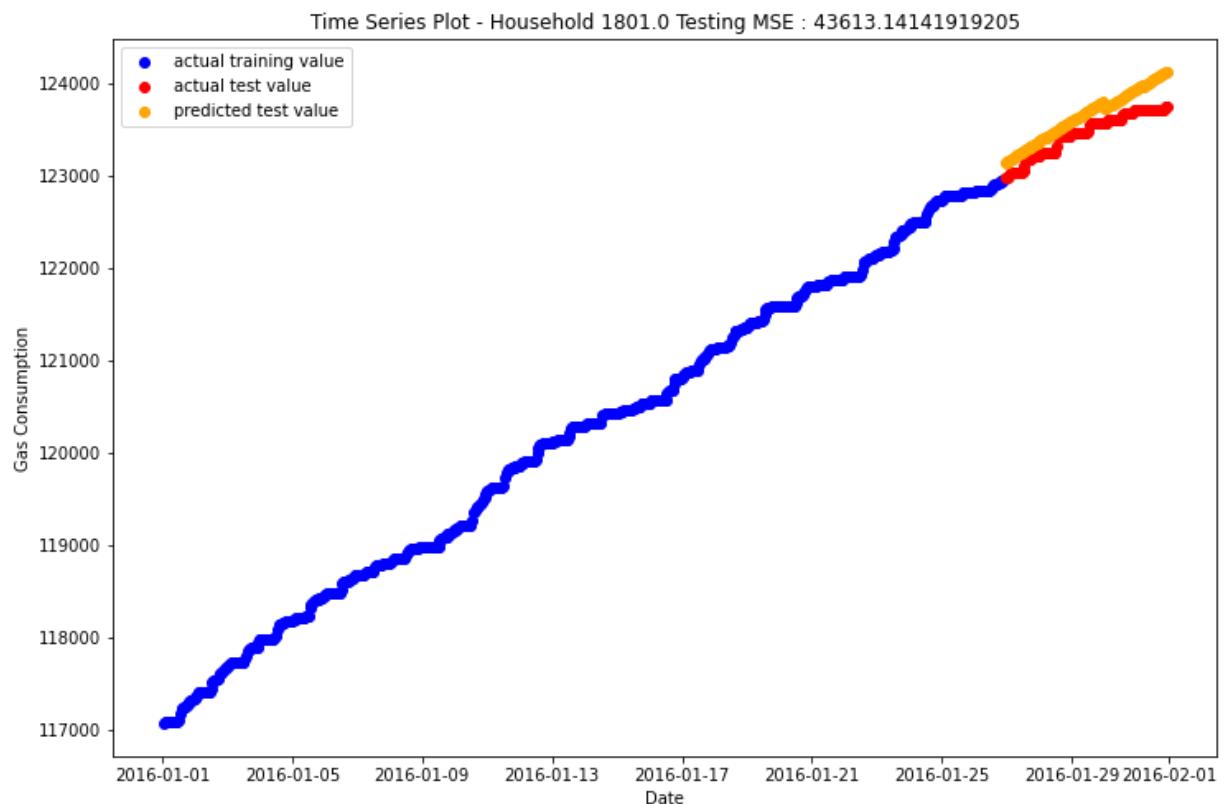
Time Series Plot - Household 1792.0 Testing MSE : 2305.066981839582

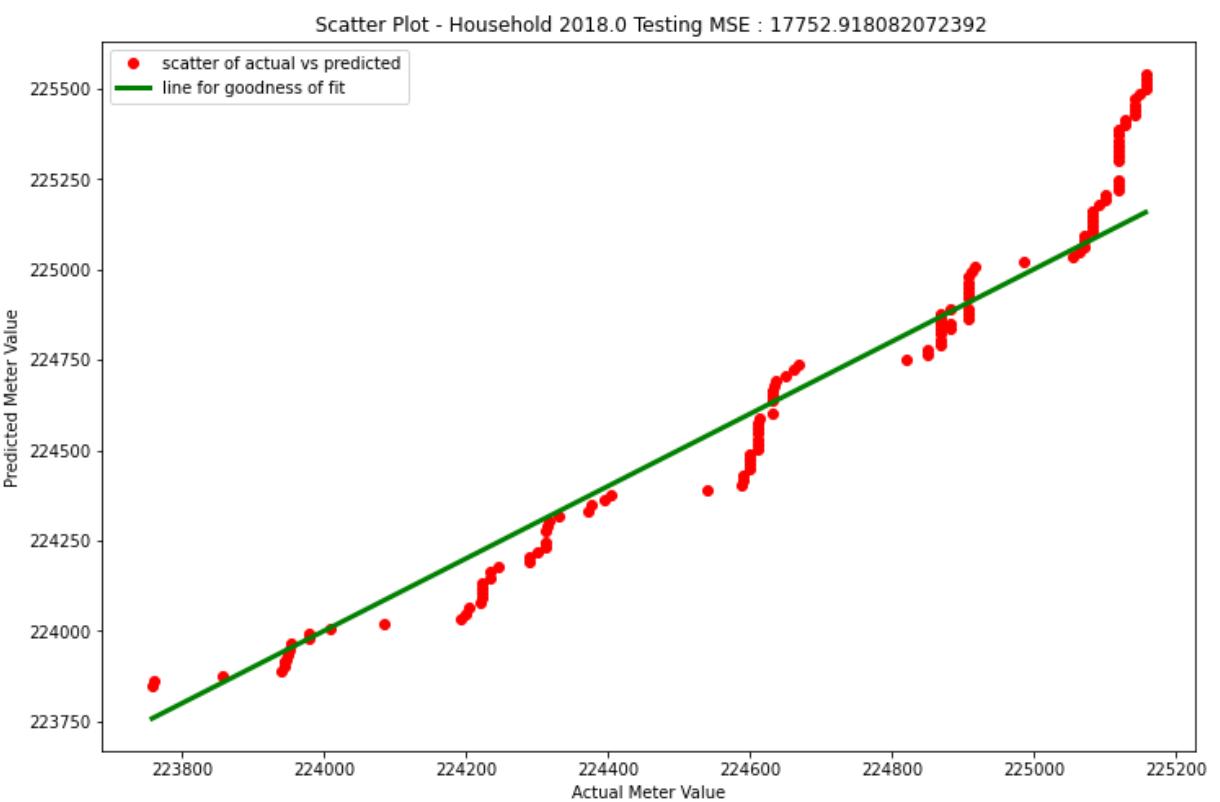
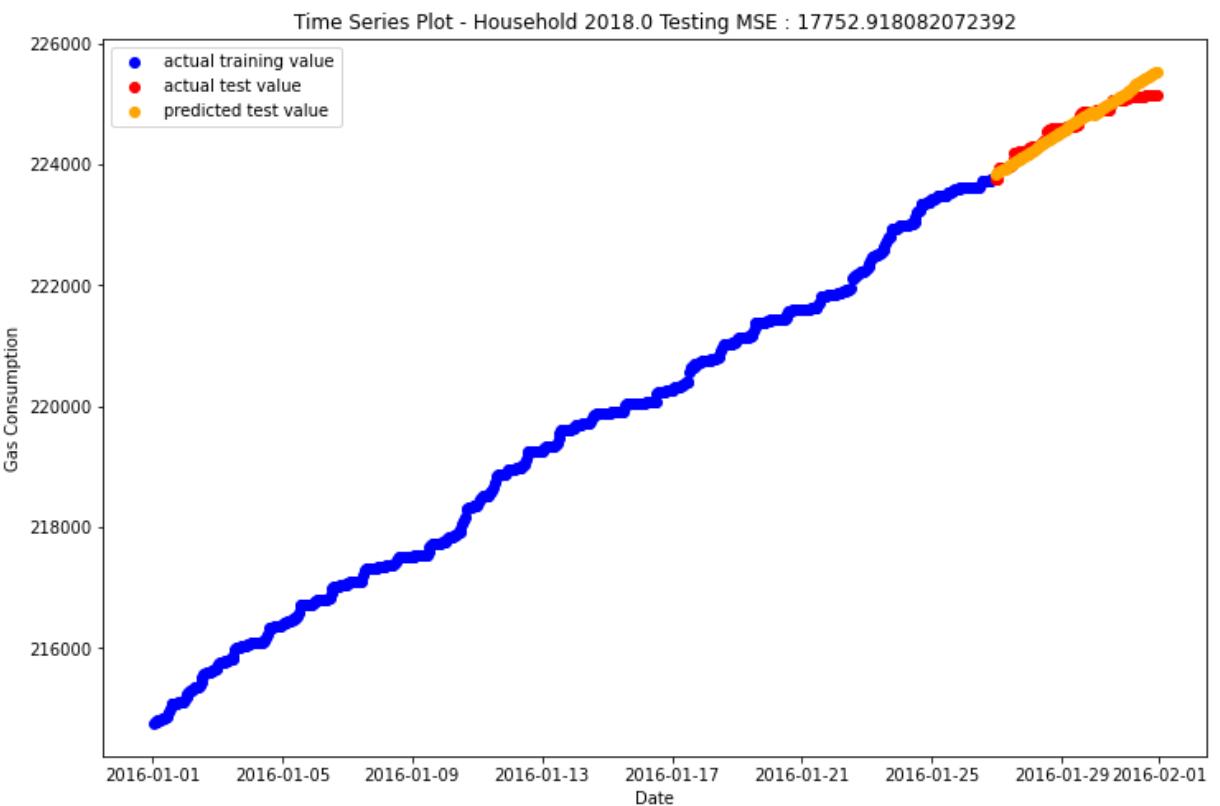


Scatter Plot - Household 1792.0 Testing MSE : 2305.066981839582

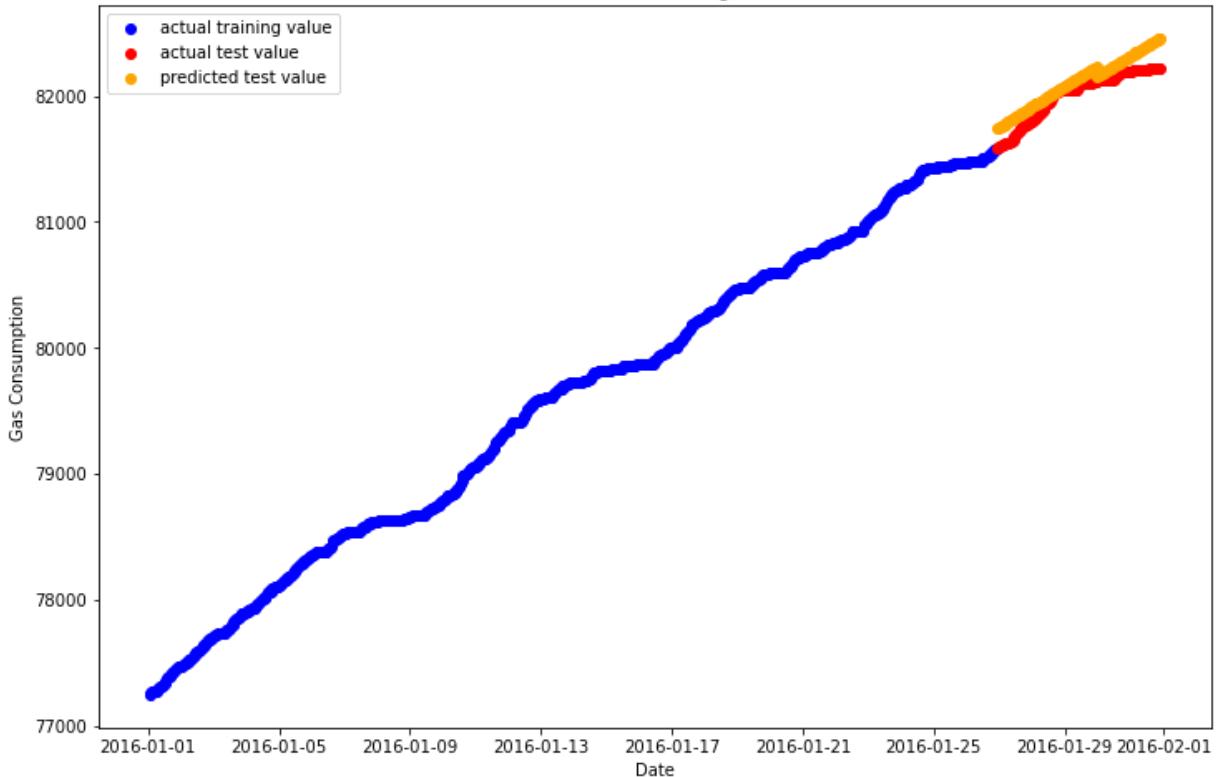




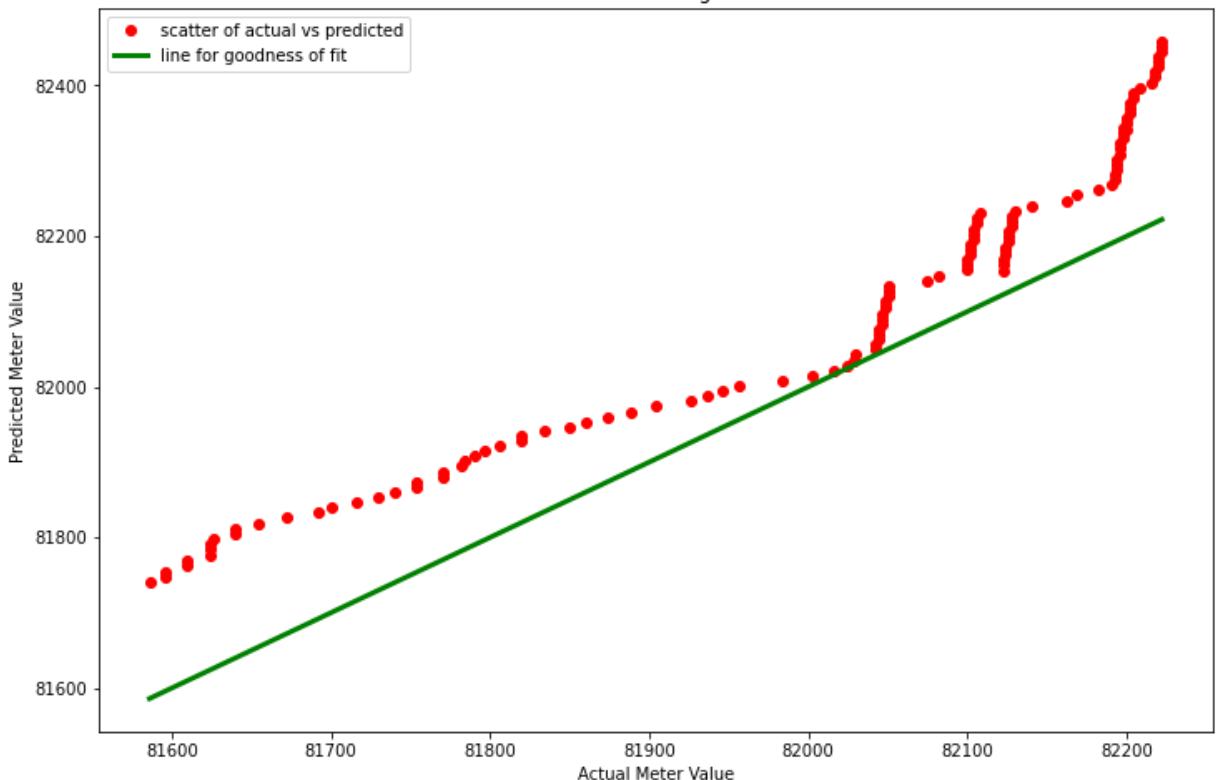


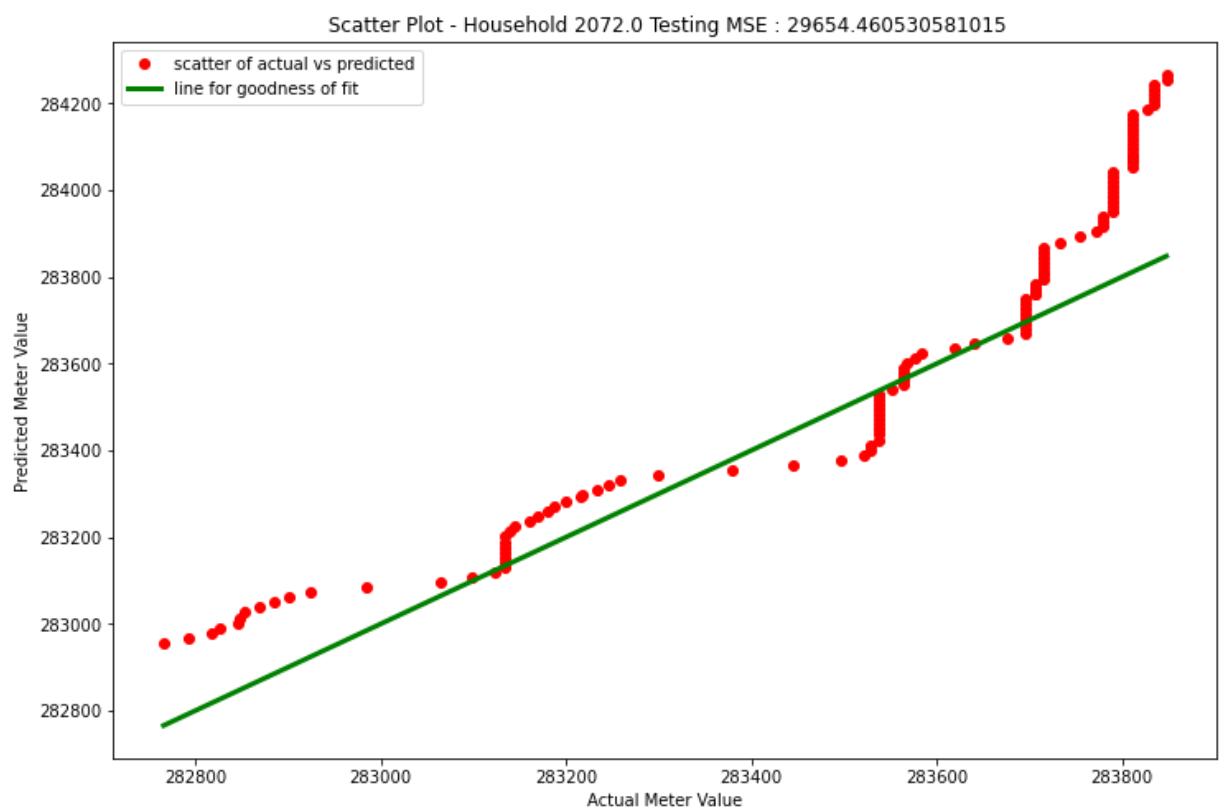
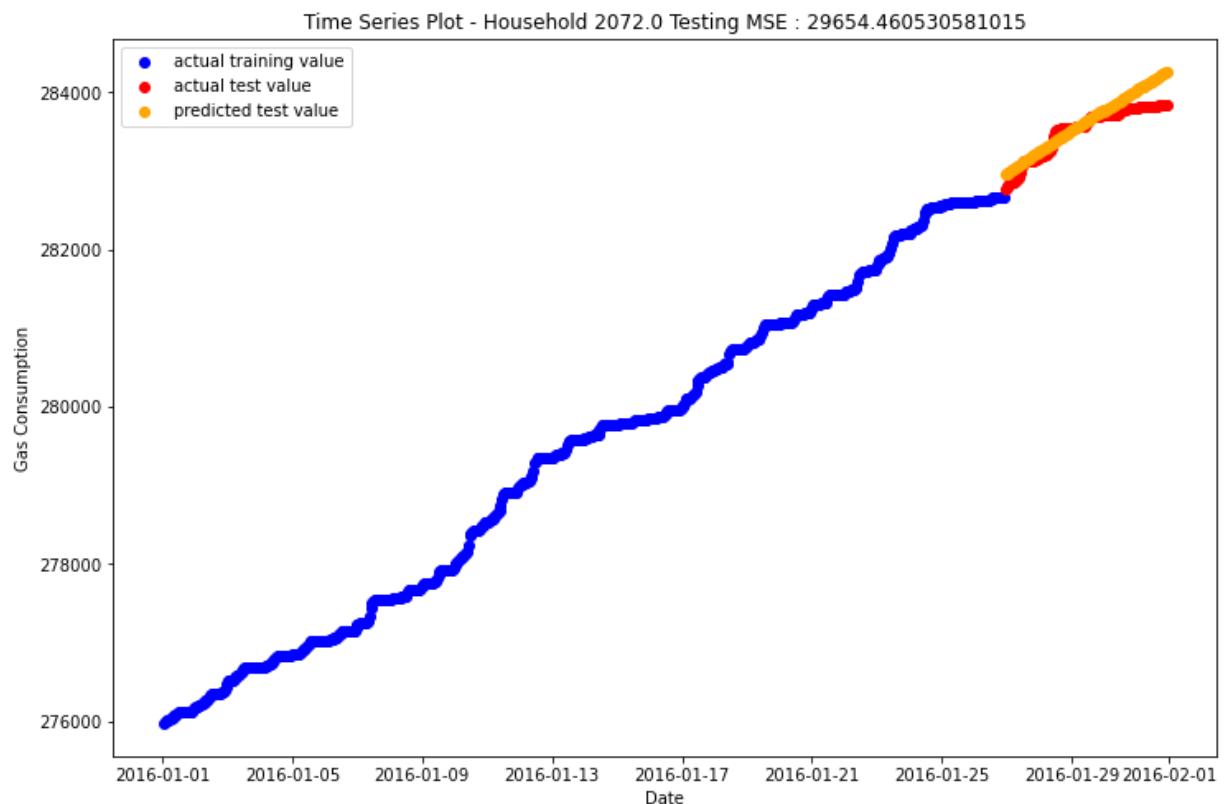


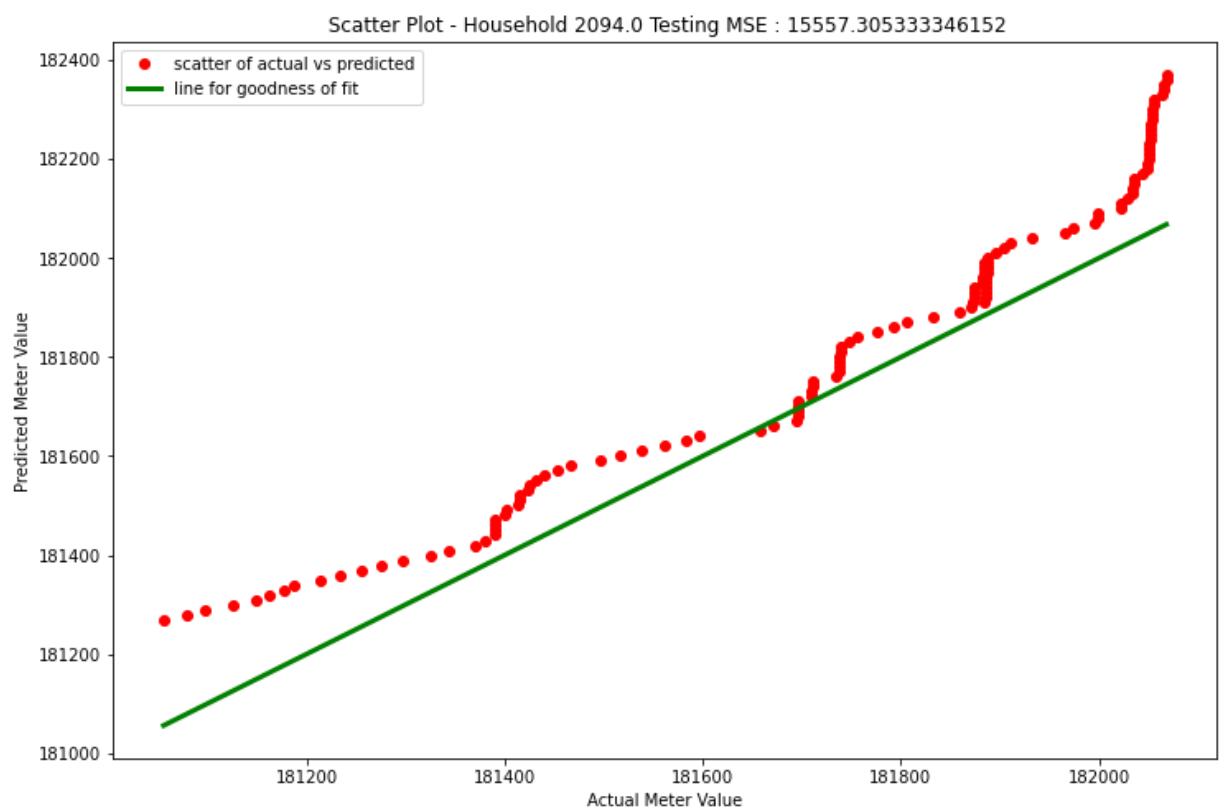
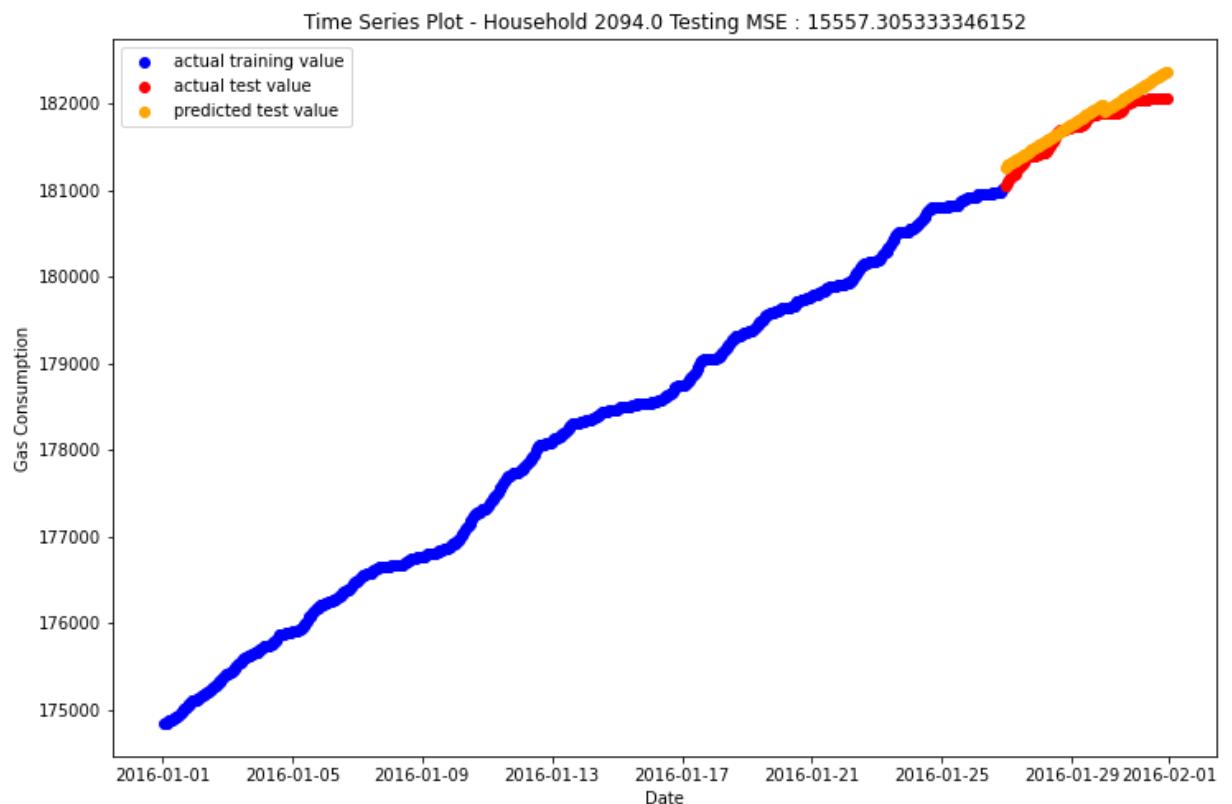
Time Series Plot - Household 2034.0 Testing MSE : 14025.652947933499



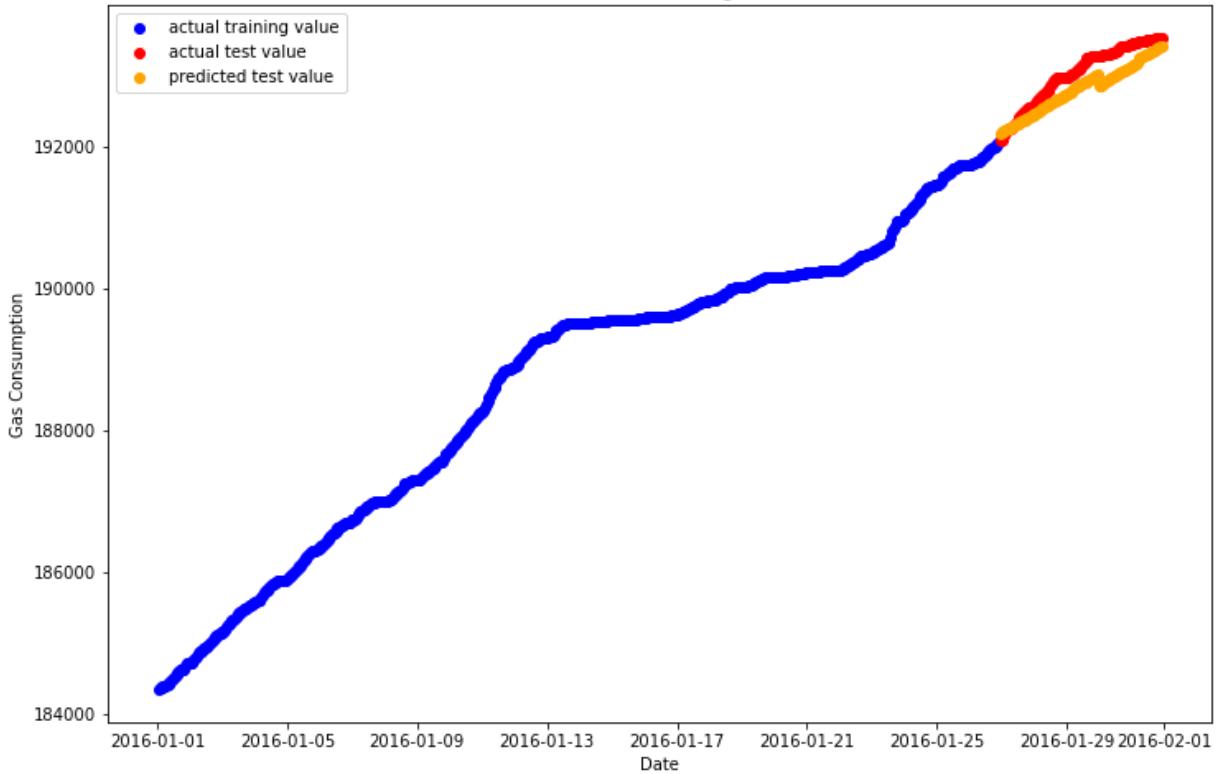
Scatter Plot - Household 2034.0 Testing MSE : 14025.652947933499



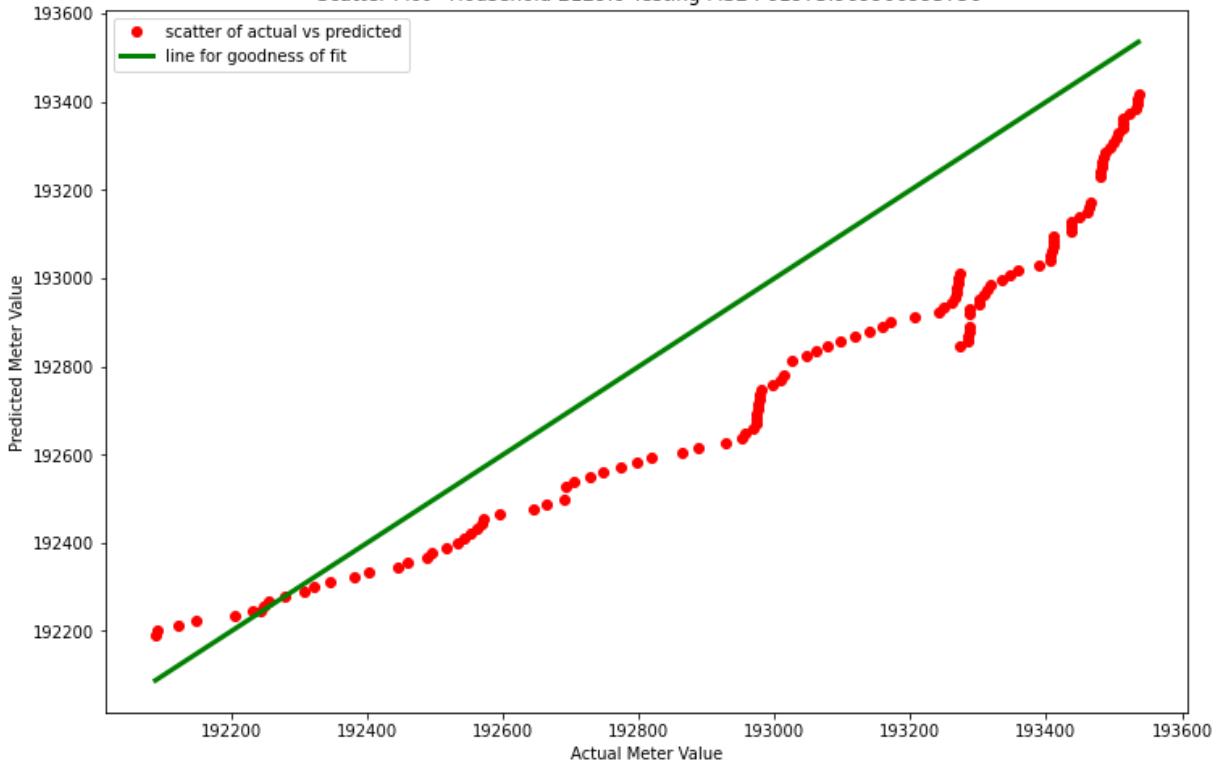




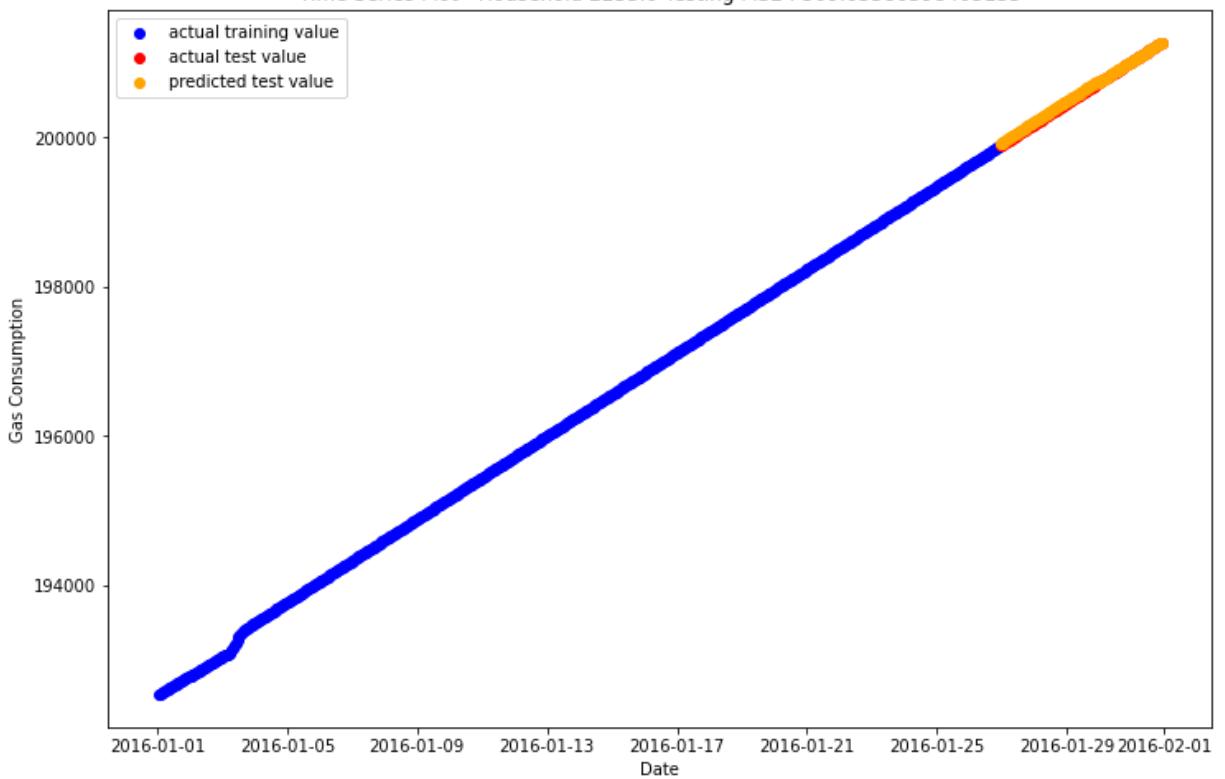
Time Series Plot - Household 2129.0 Testing MSE : 61973.969906993756



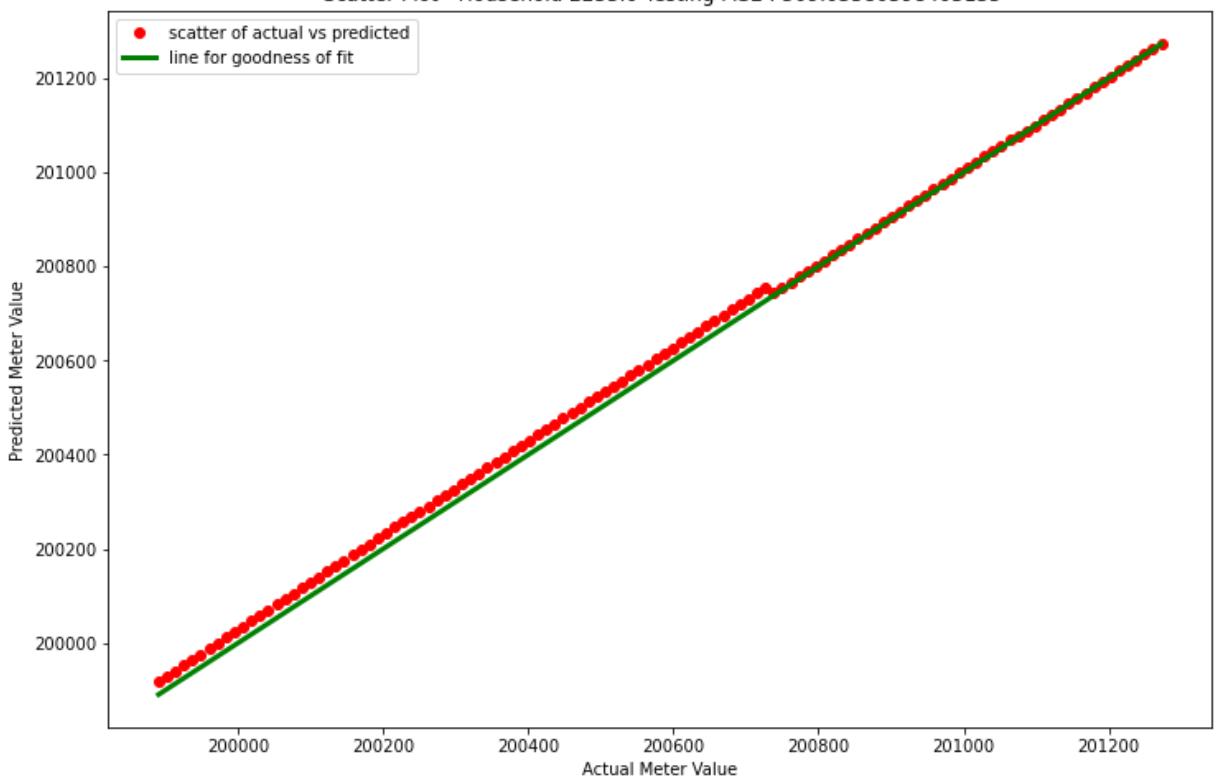
Scatter Plot - Household 2129.0 Testing MSE : 61973.969906993756



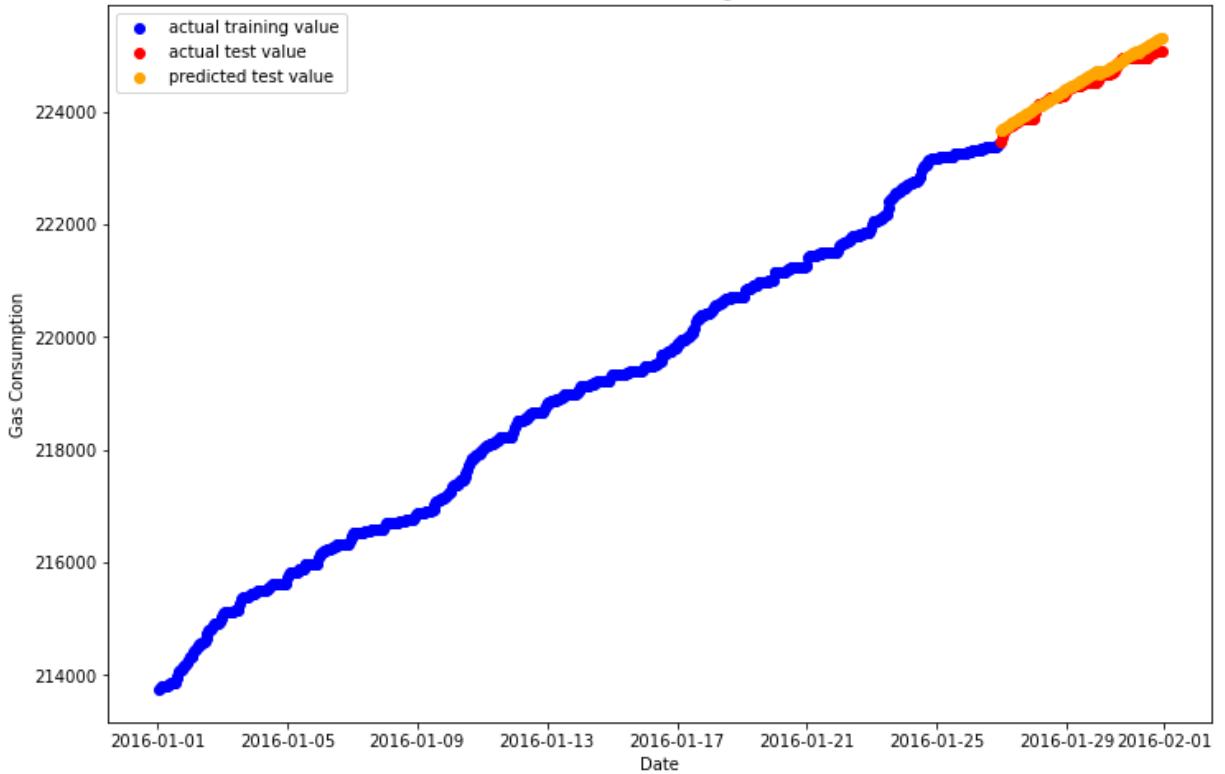
Time Series Plot - Household 2233.0 Testing MSE : 509.03380398403135



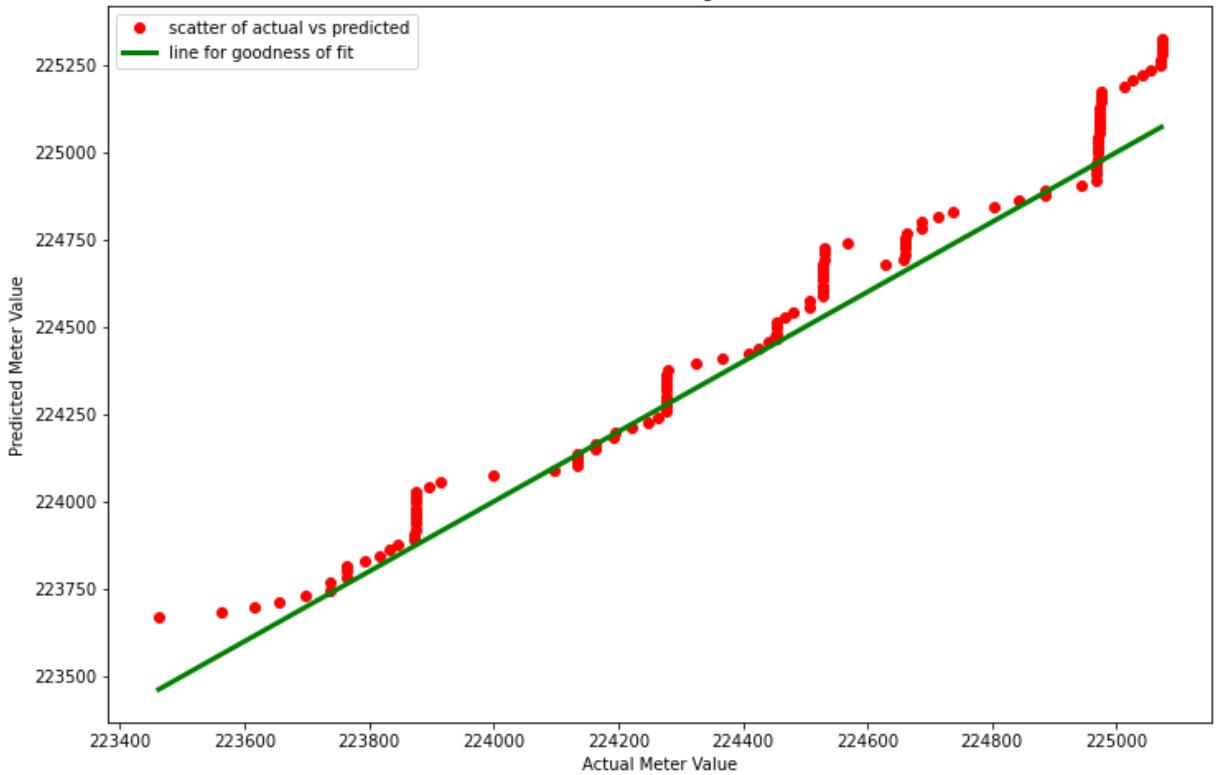
Scatter Plot - Household 2233.0 Testing MSE : 509.03380398403135

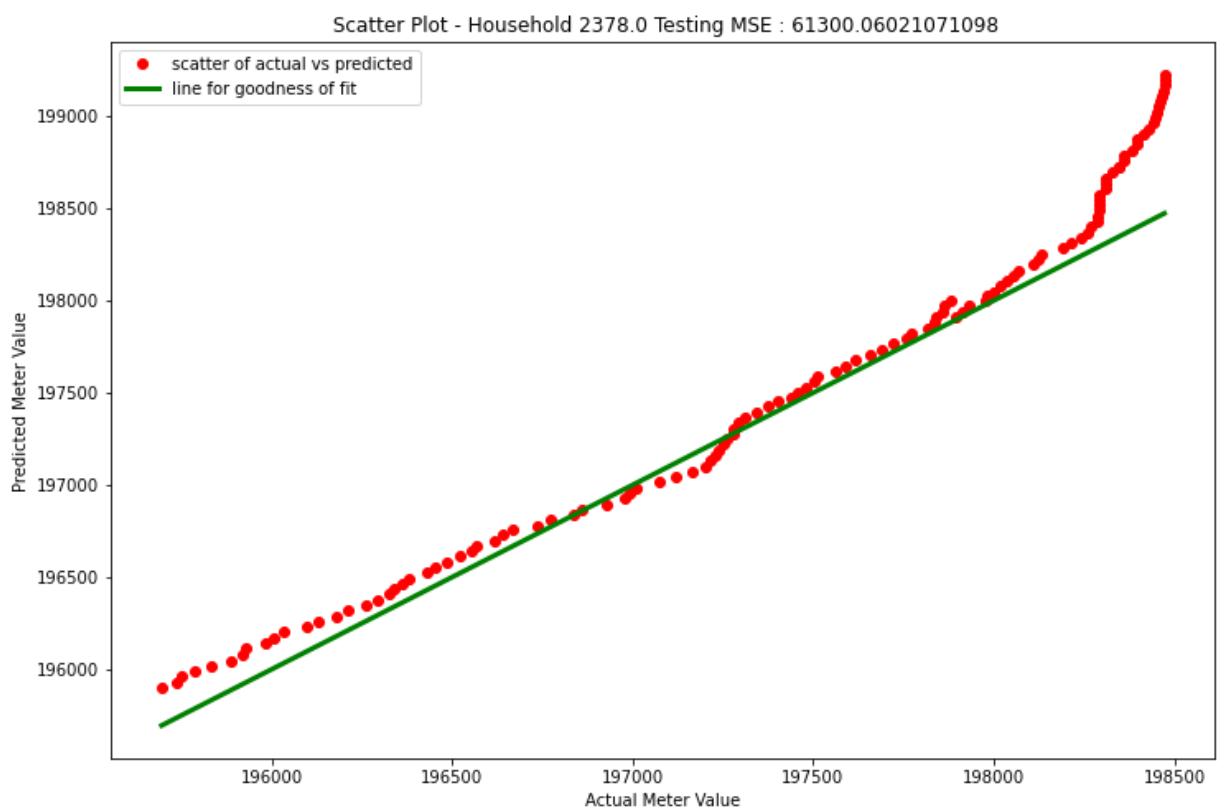
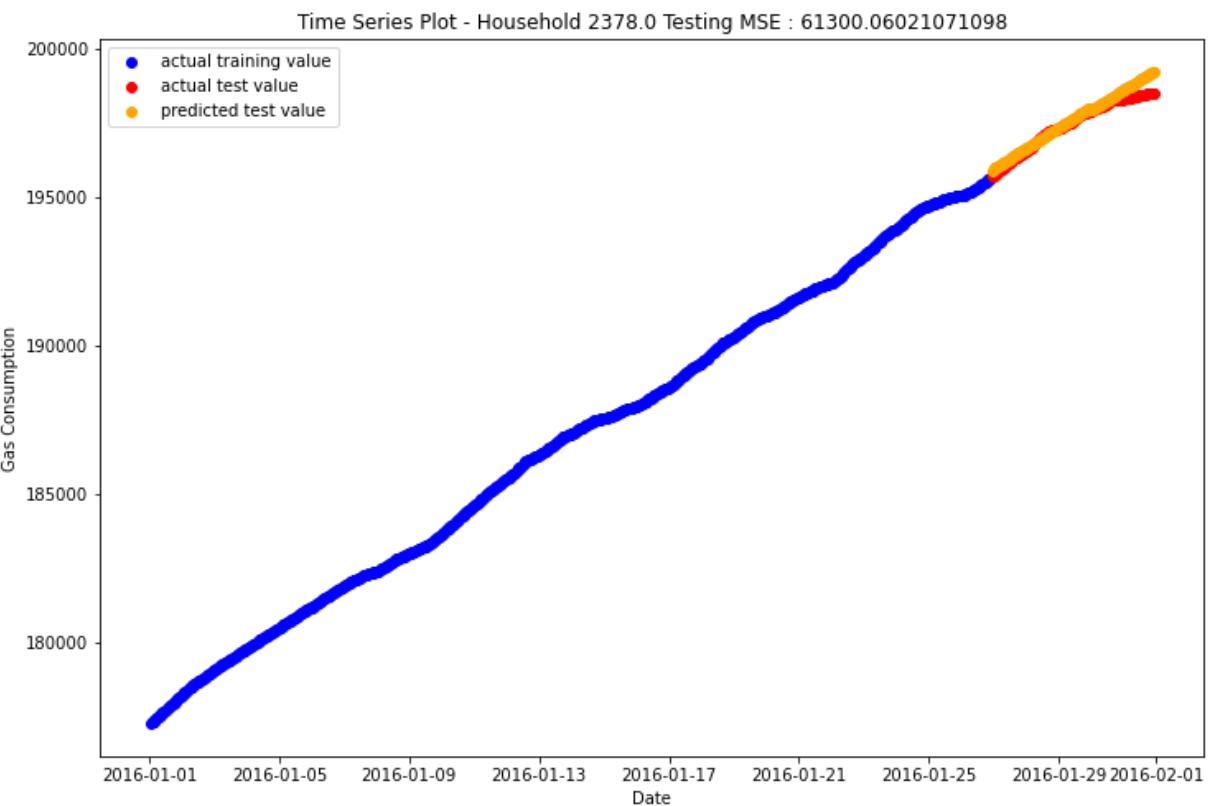


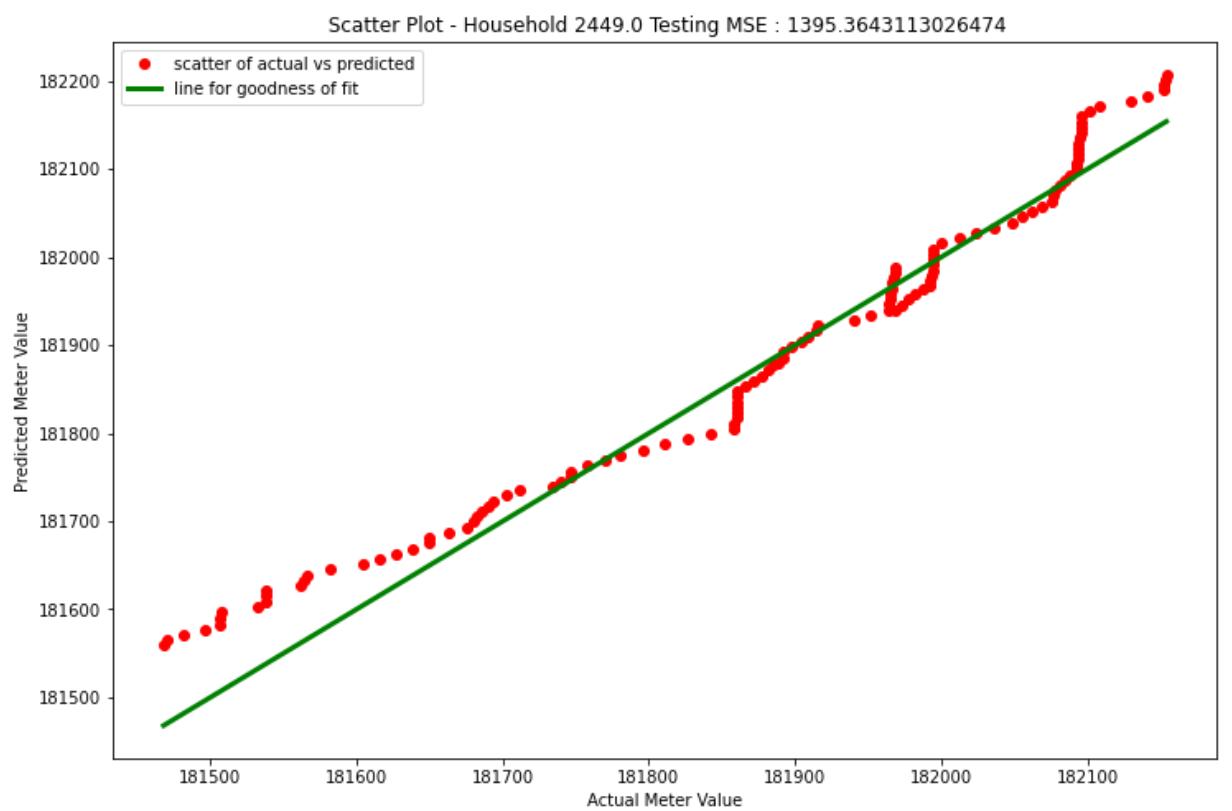
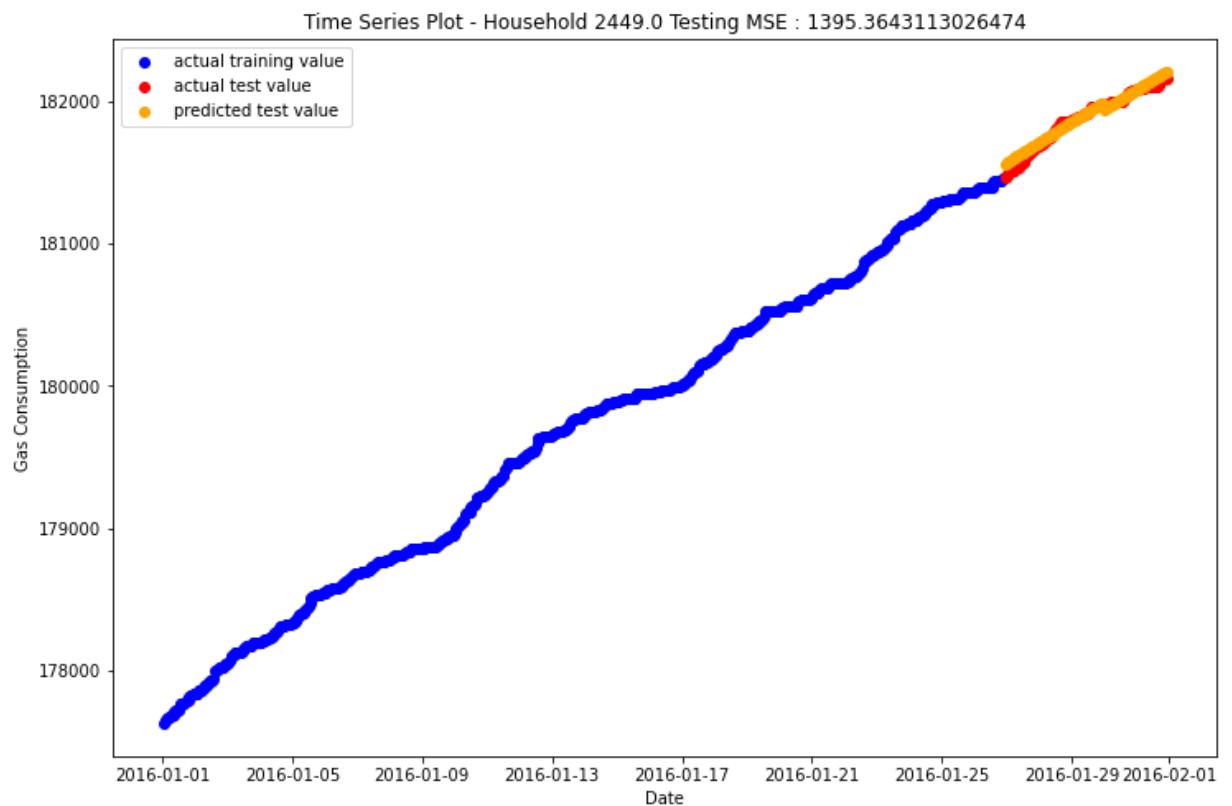
Time Series Plot - Household 2335.0 Testing MSE : 10134.800754932201



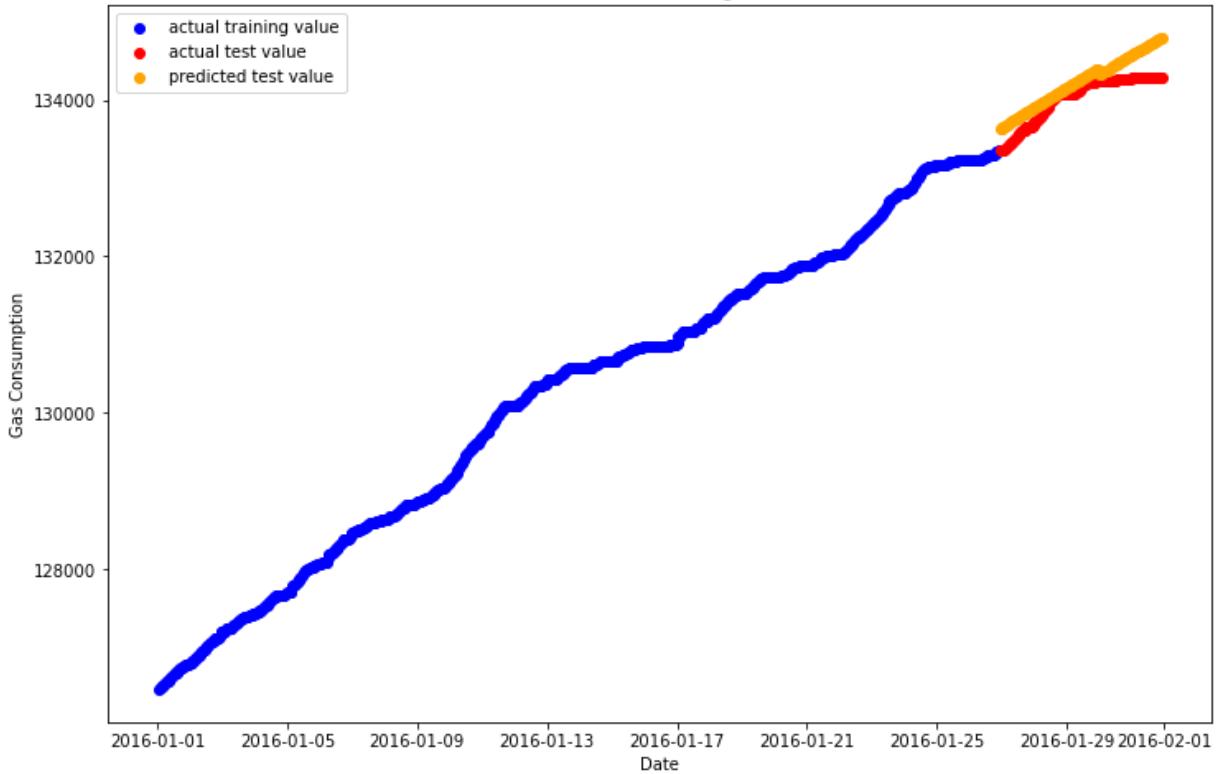
Scatter Plot - Household 2335.0 Testing MSE : 10134.800754932201



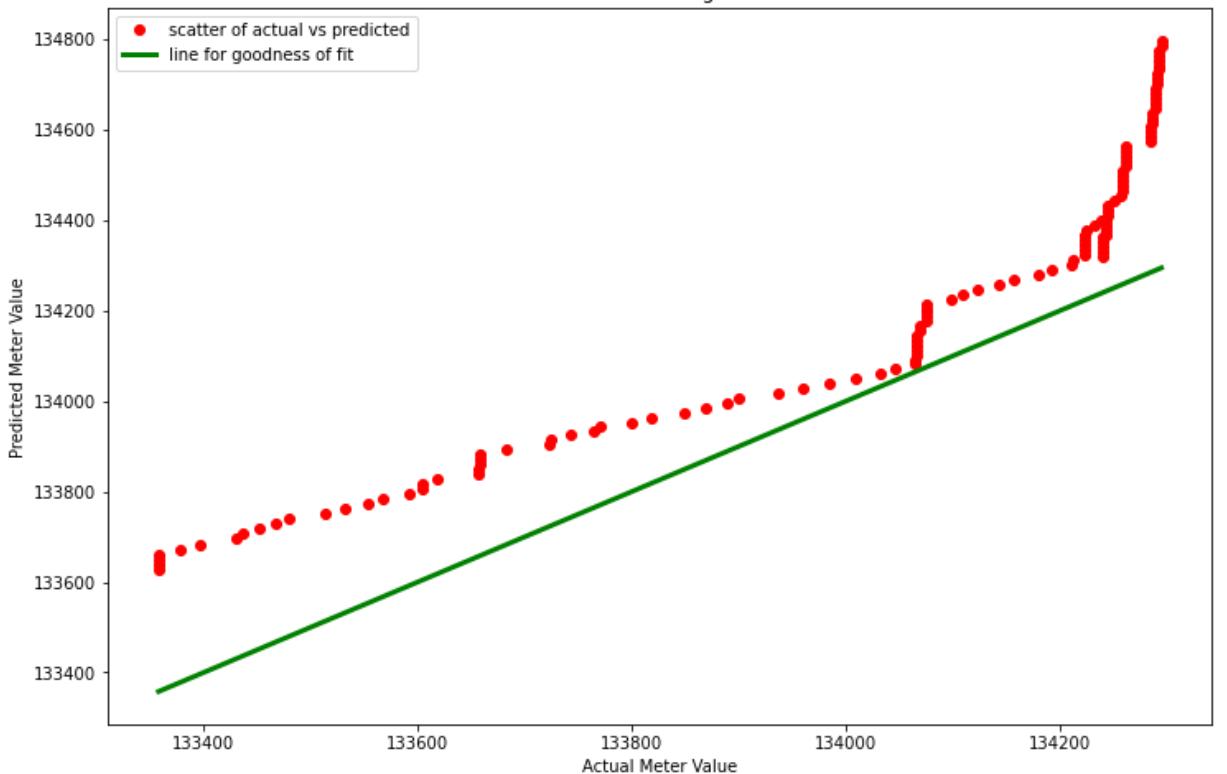


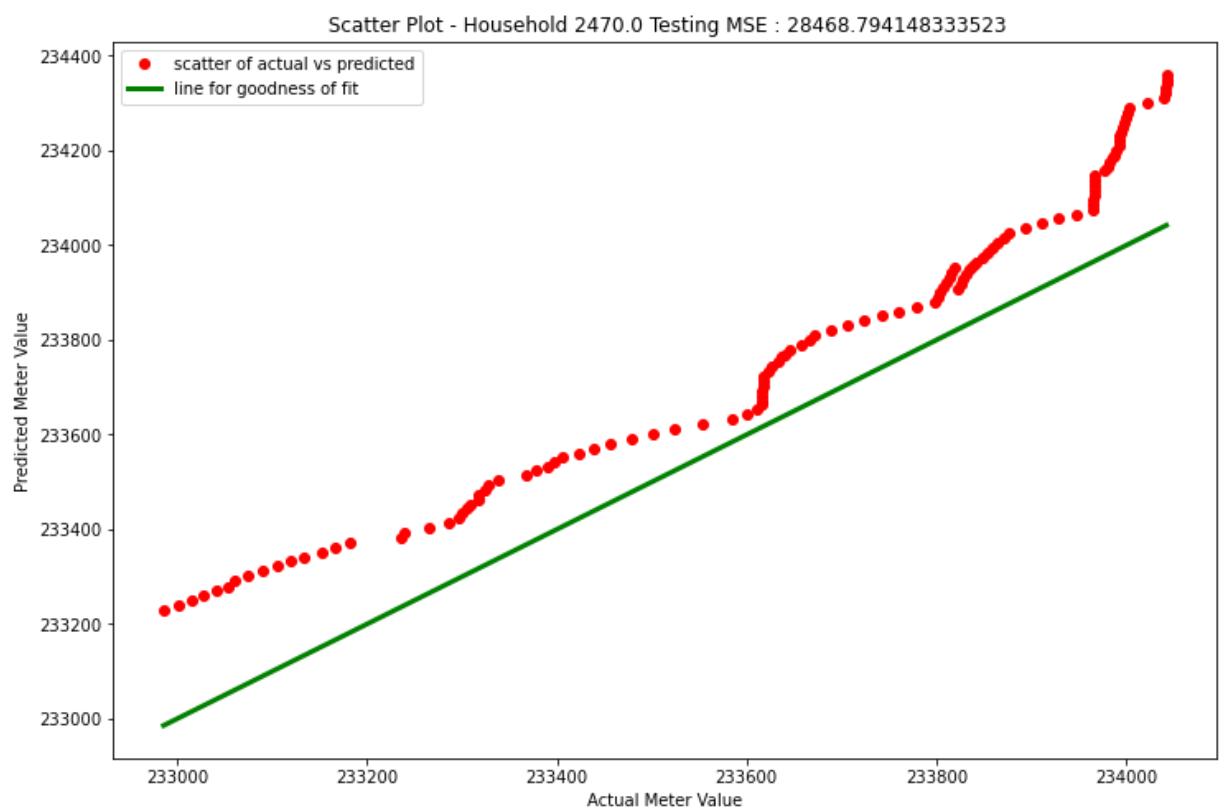
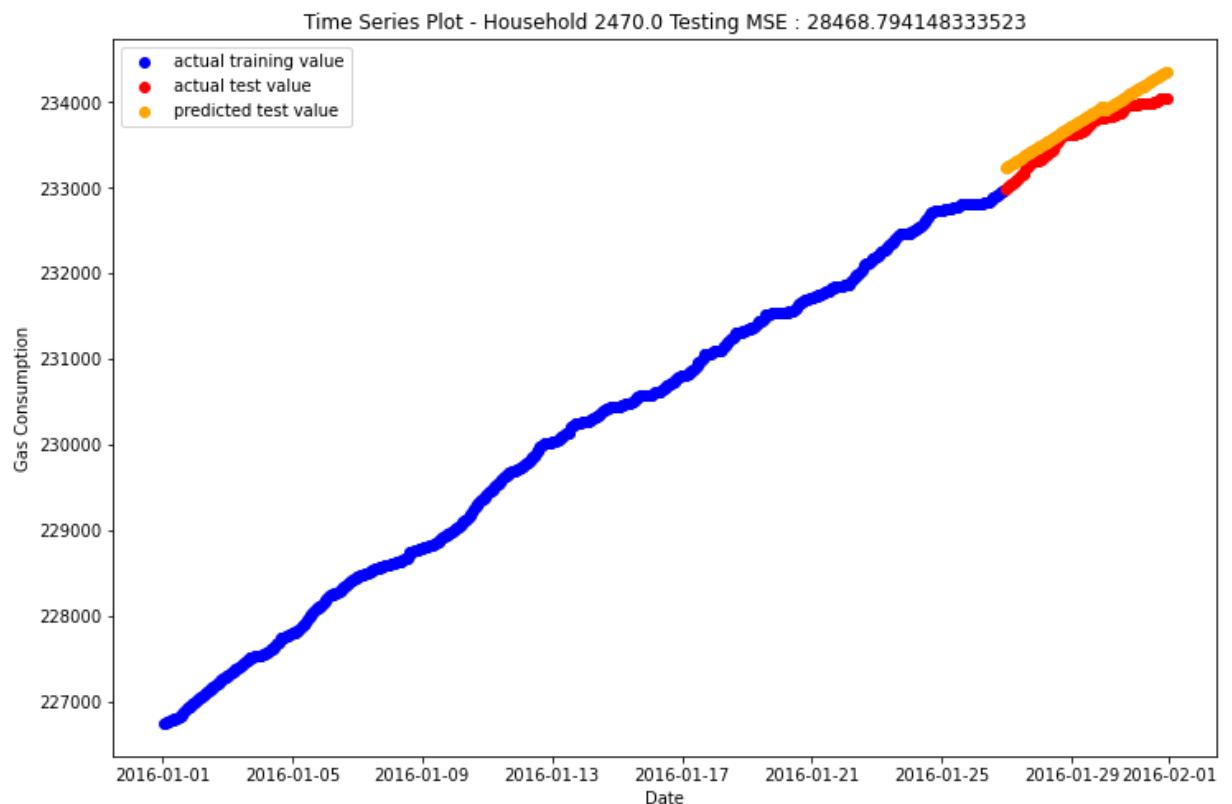


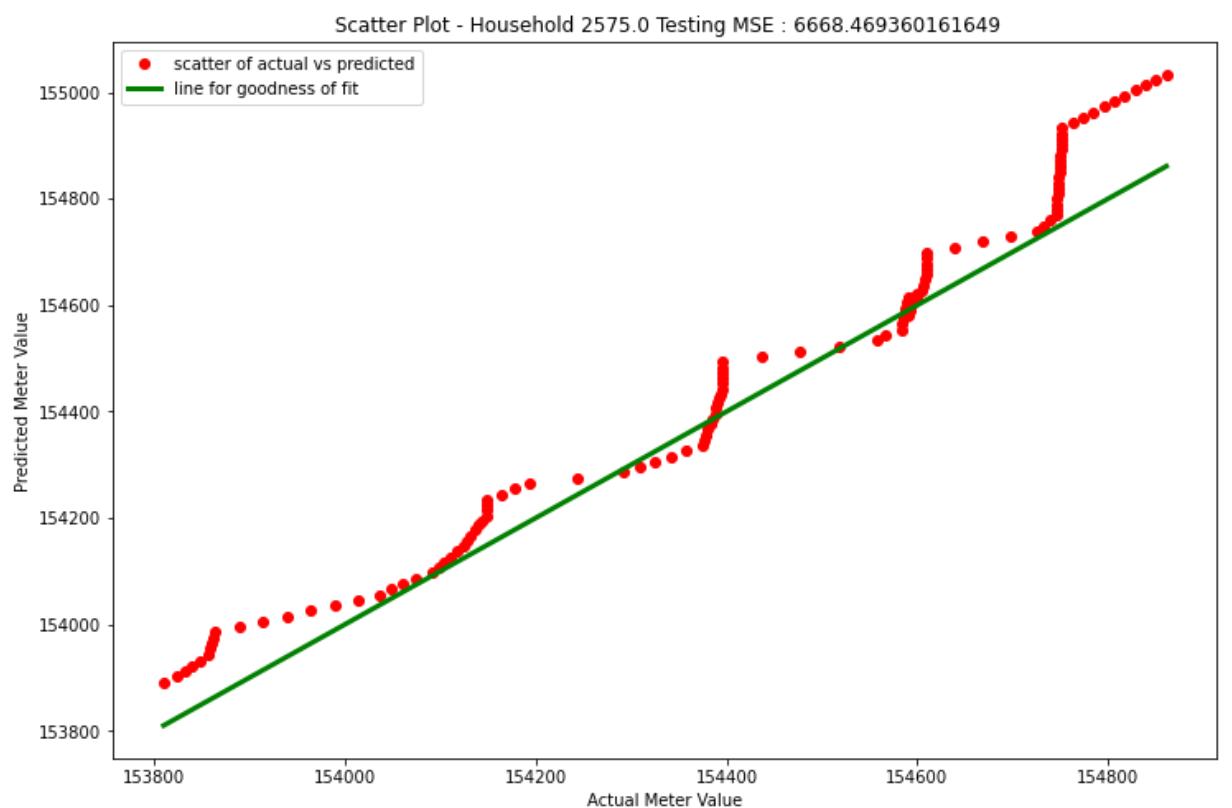
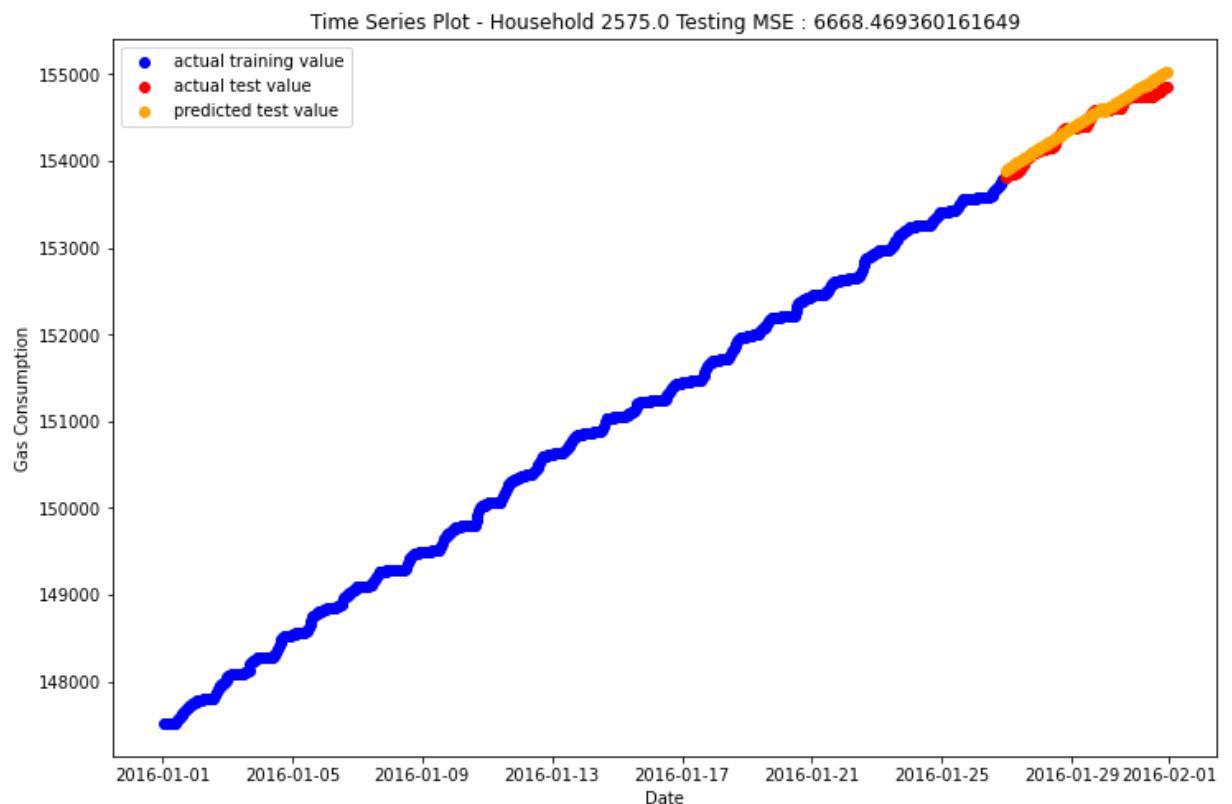
Time Series Plot - Household 2461.0 Testing MSE : 55963.043340091004

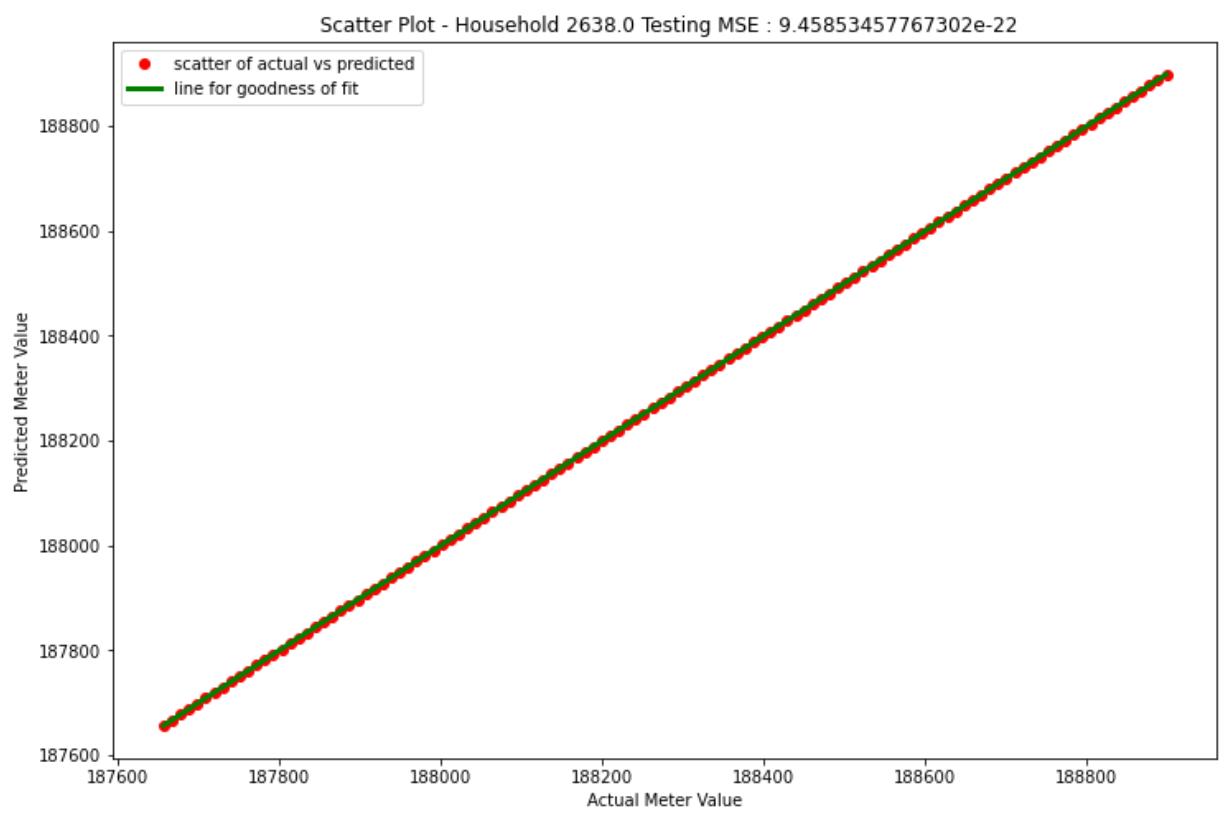
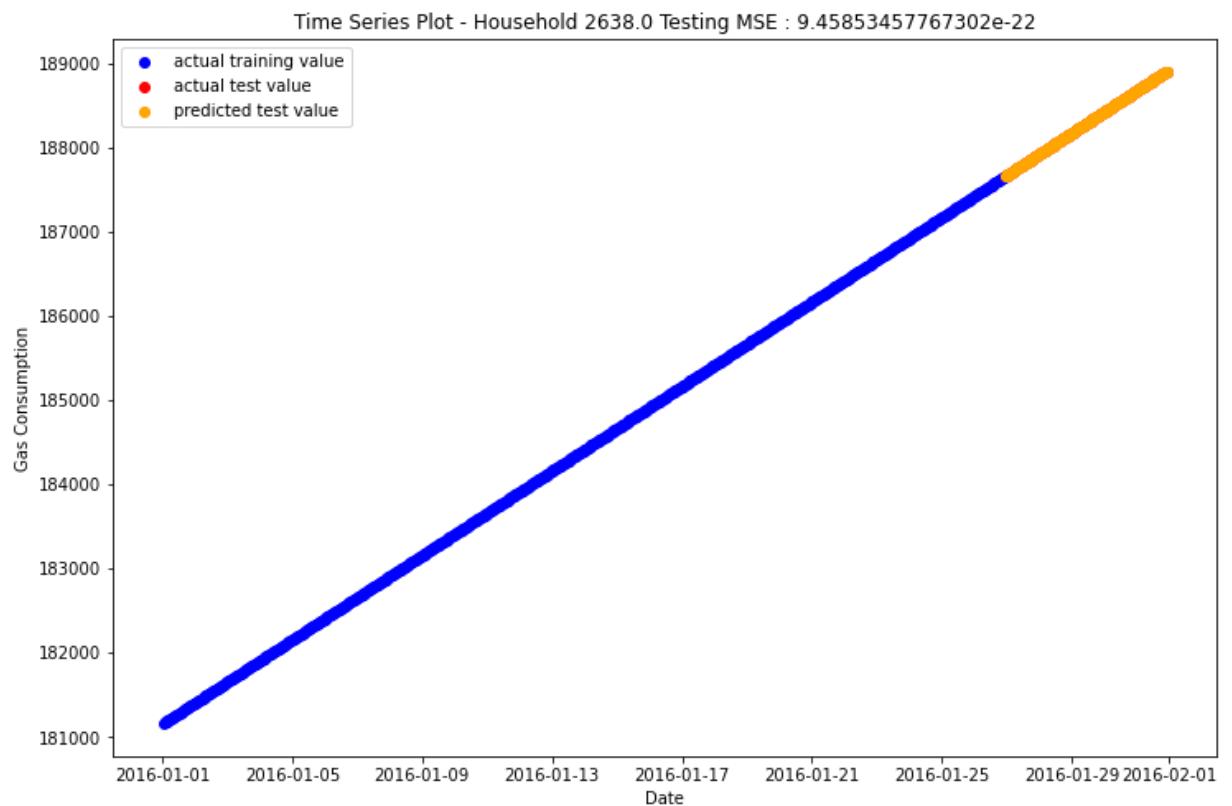


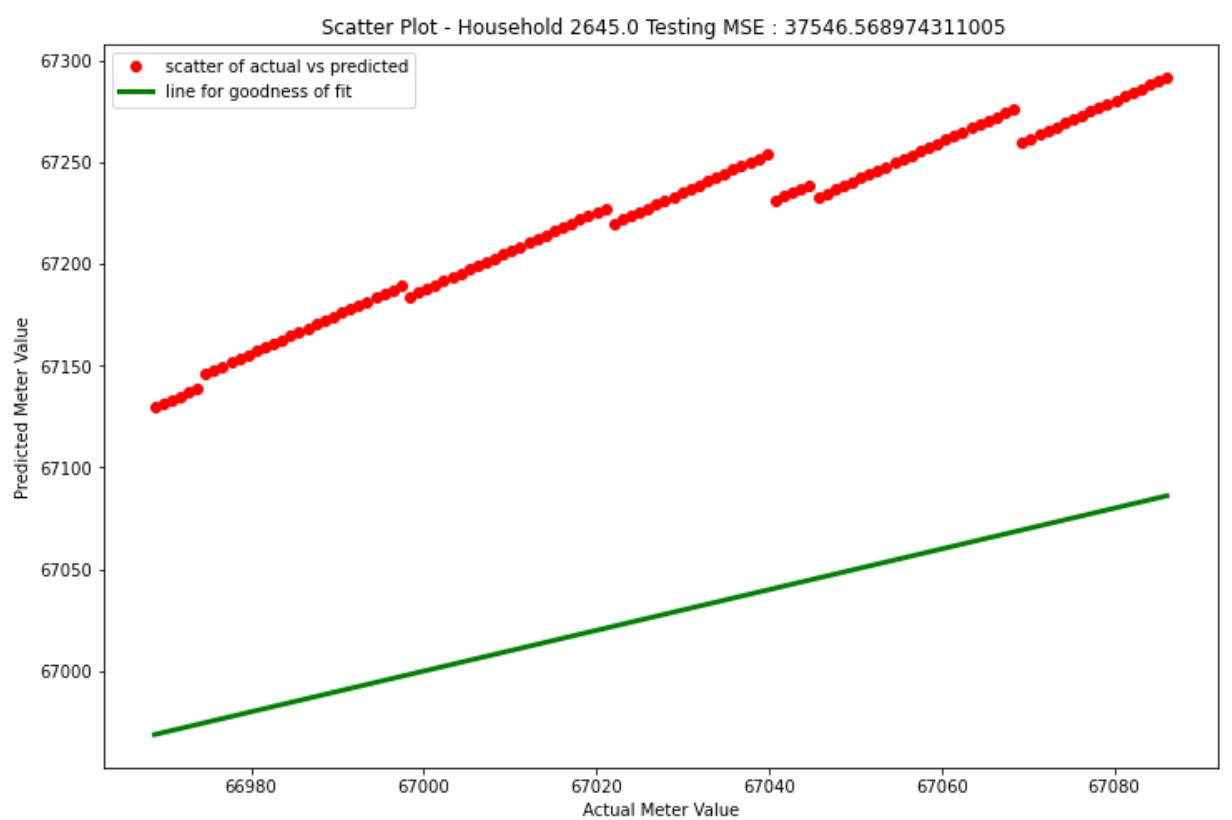
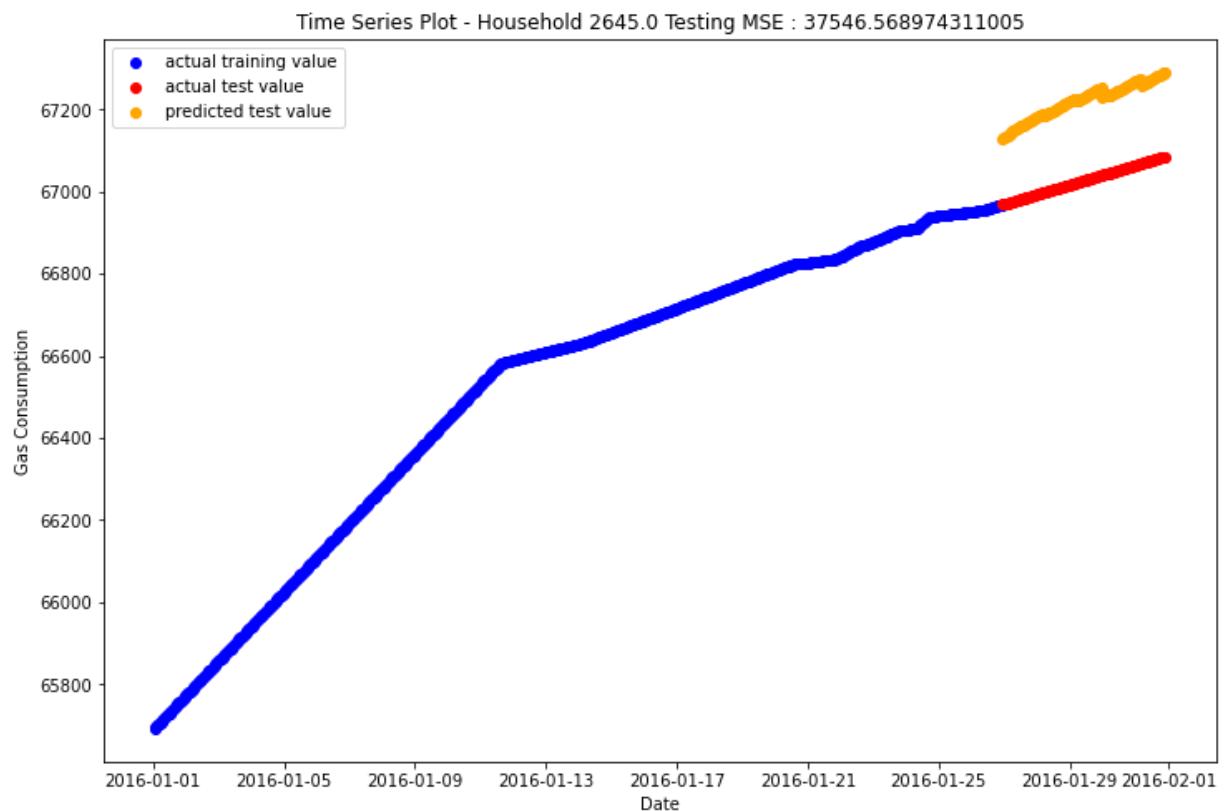
Scatter Plot - Household 2461.0 Testing MSE : 55963.043340091004



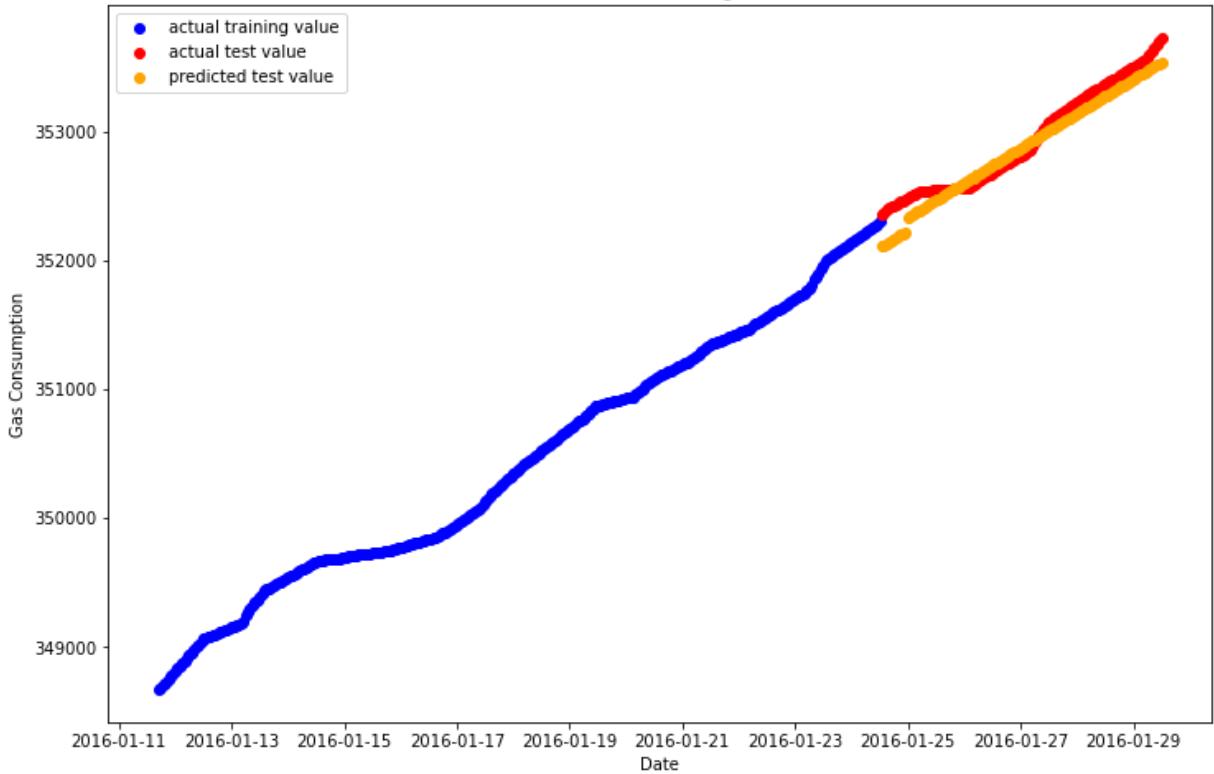




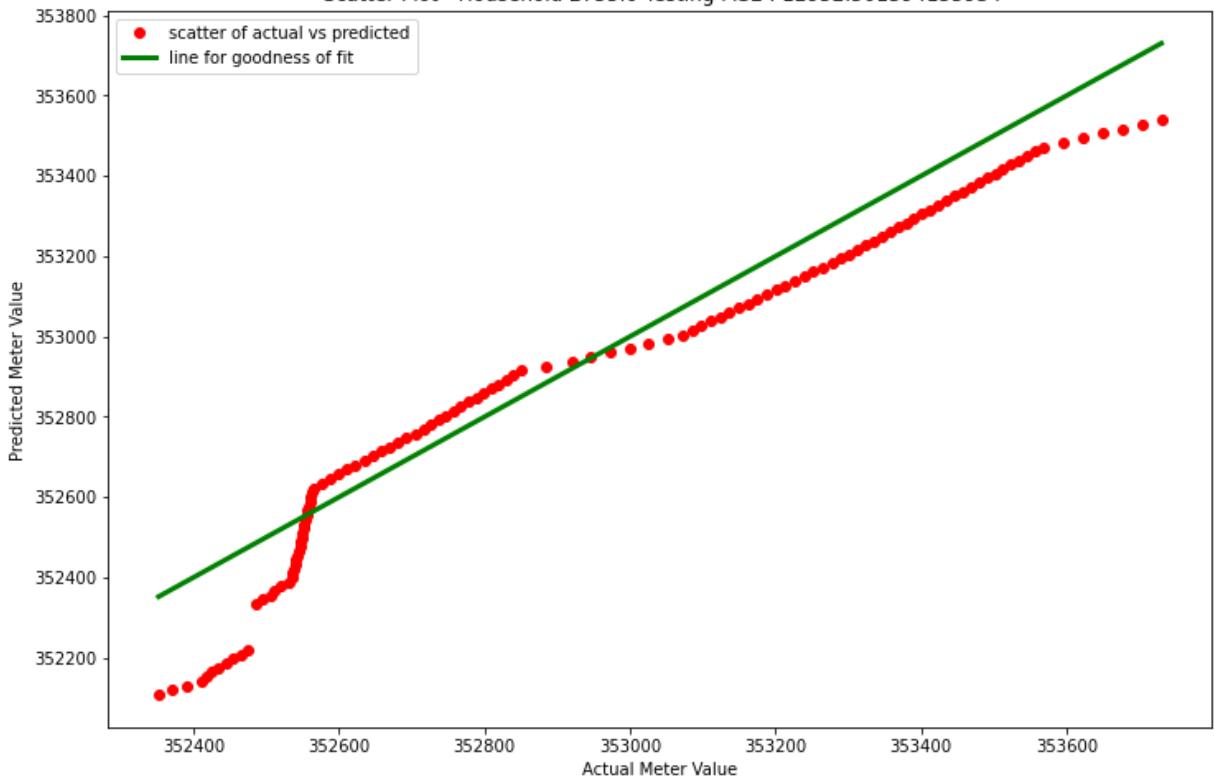


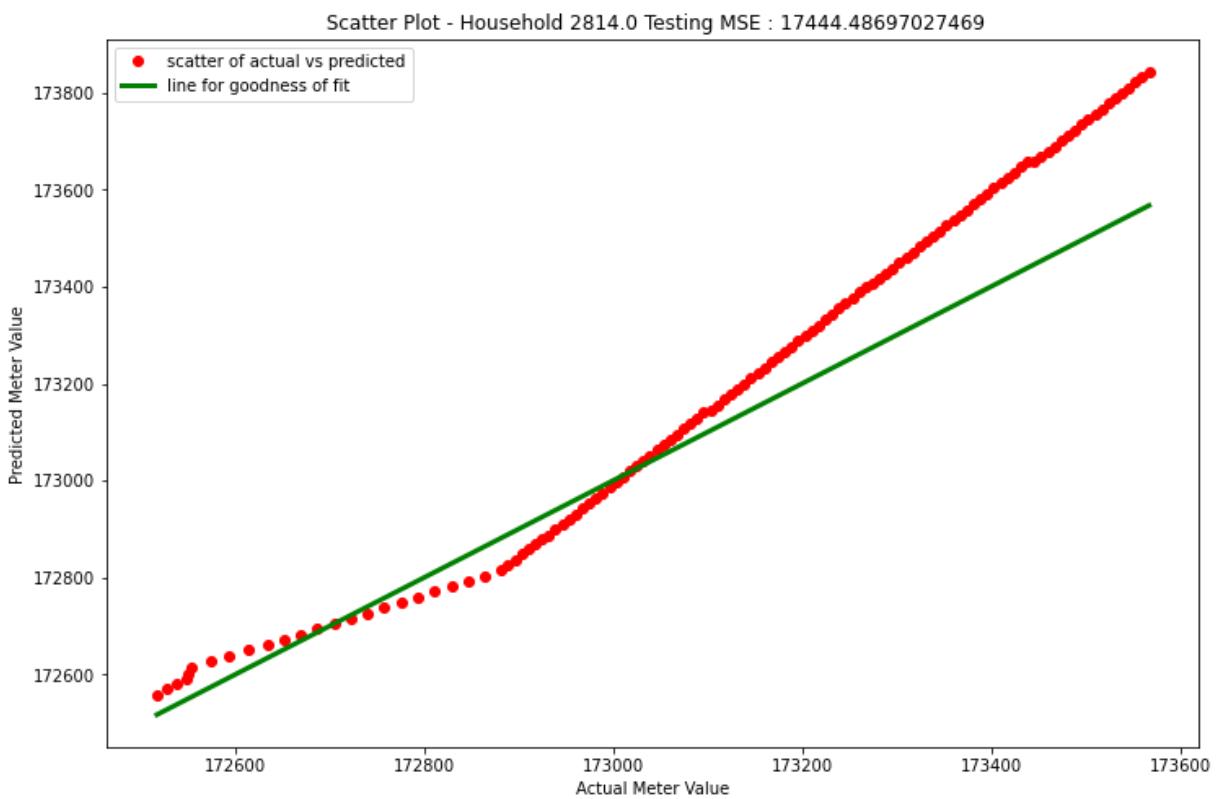
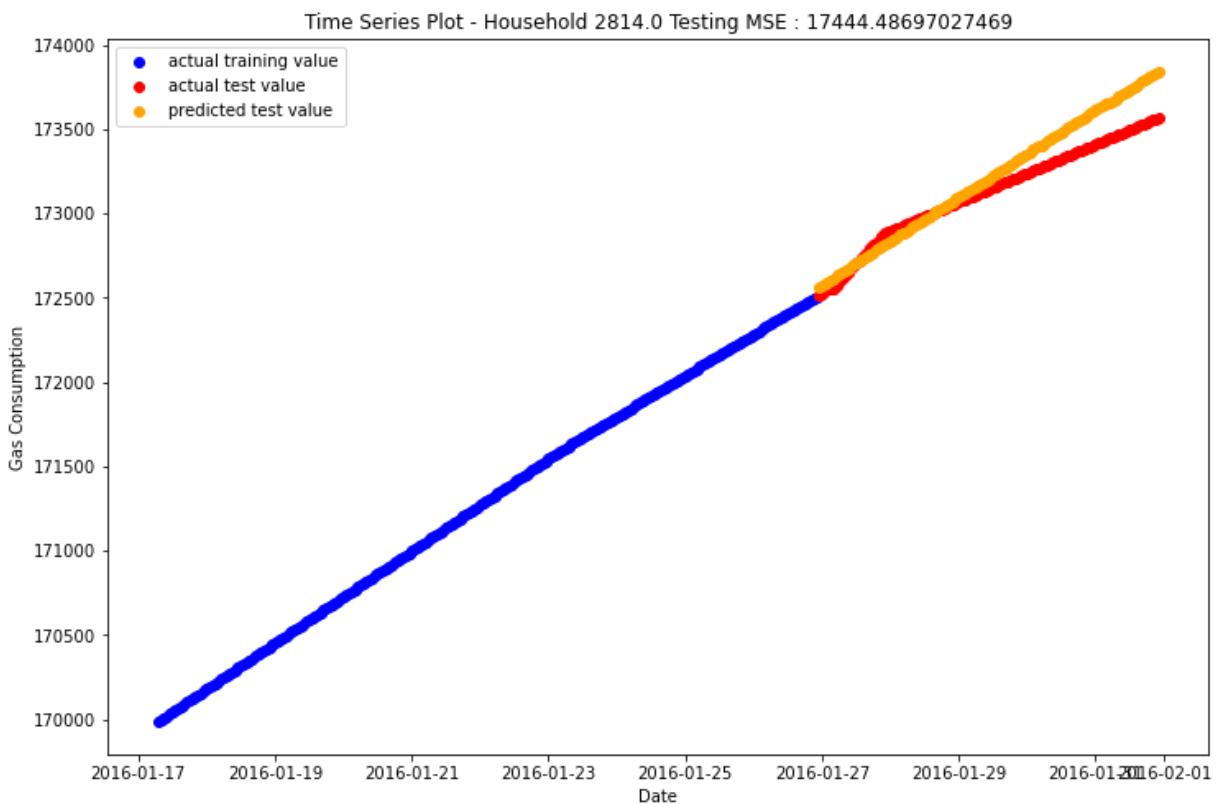


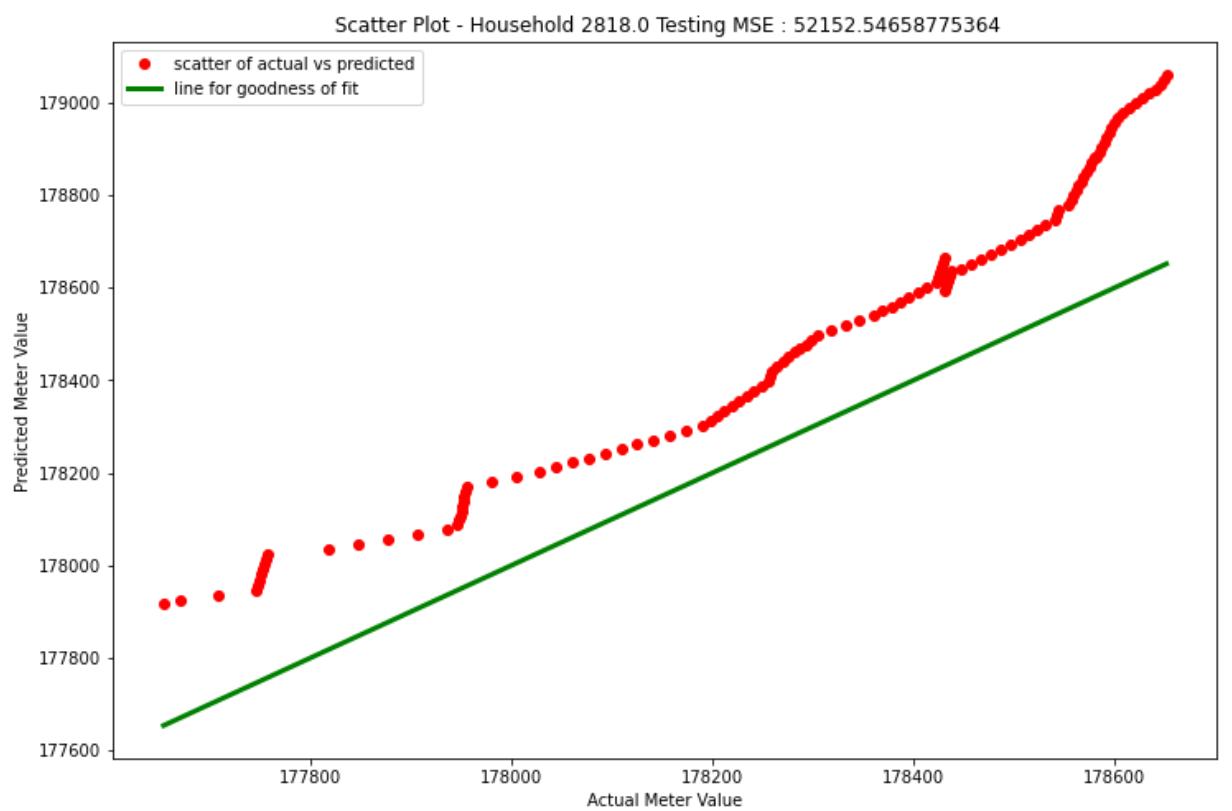
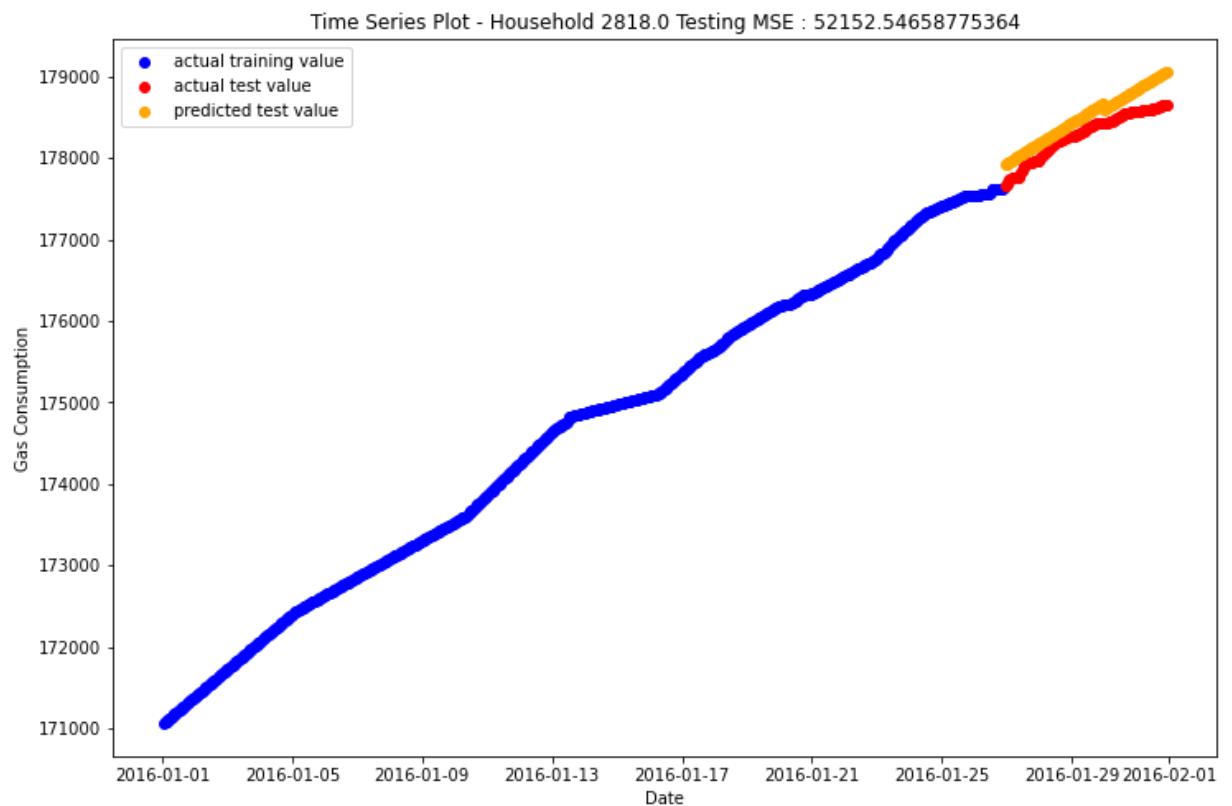
Time Series Plot - Household 2755.0 Testing MSE : 12952.501394155934

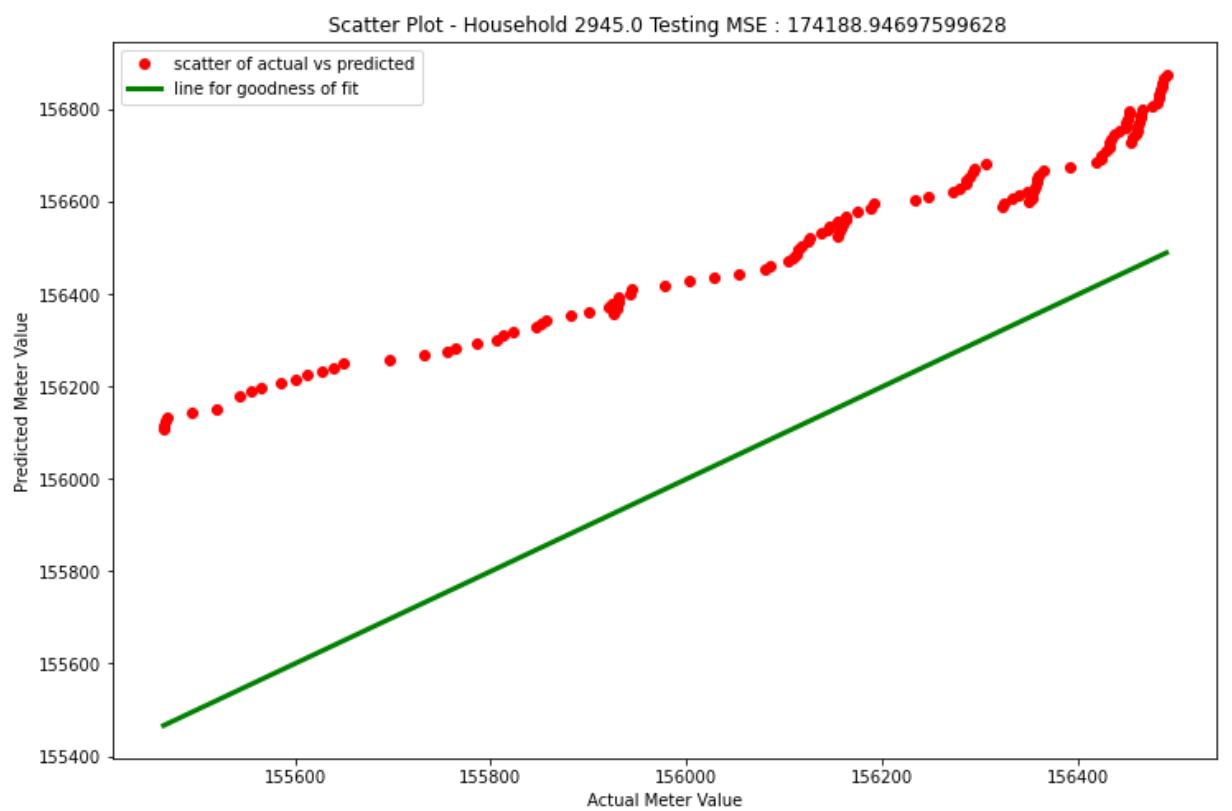
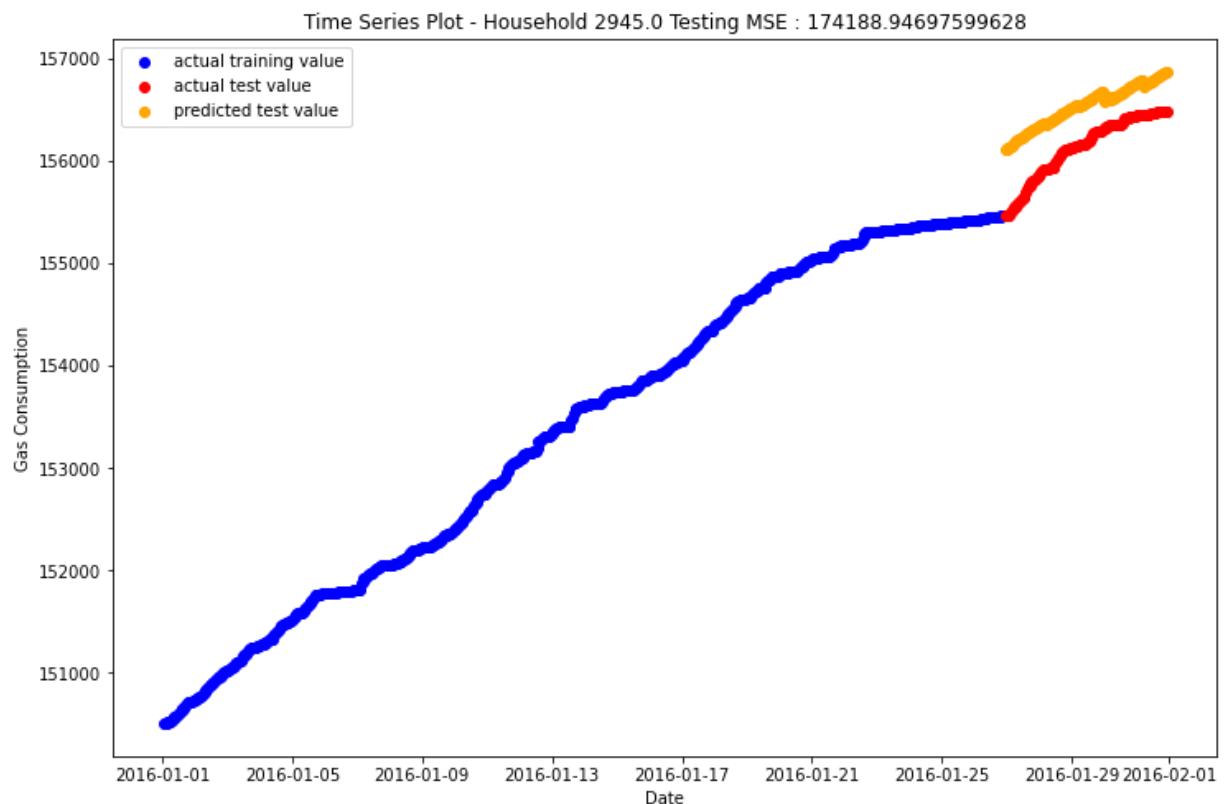


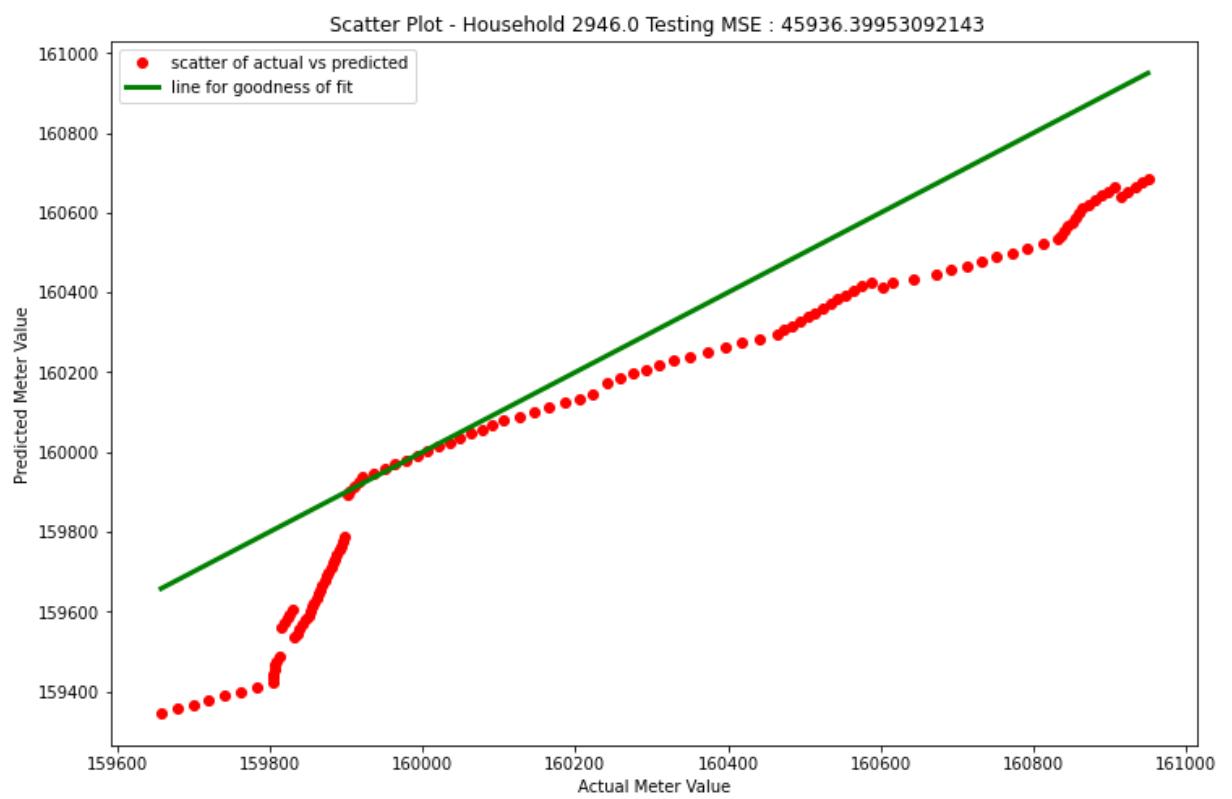
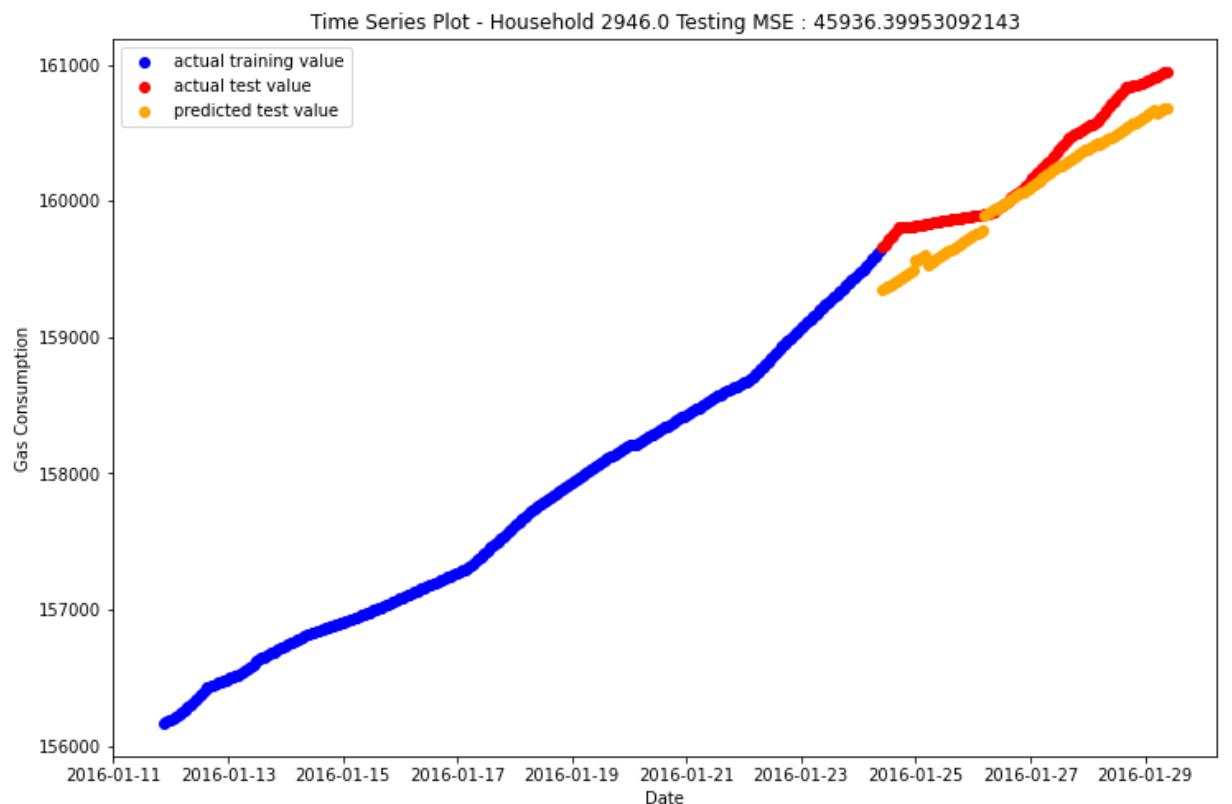
Scatter Plot - Household 2755.0 Testing MSE : 12952.501394155934

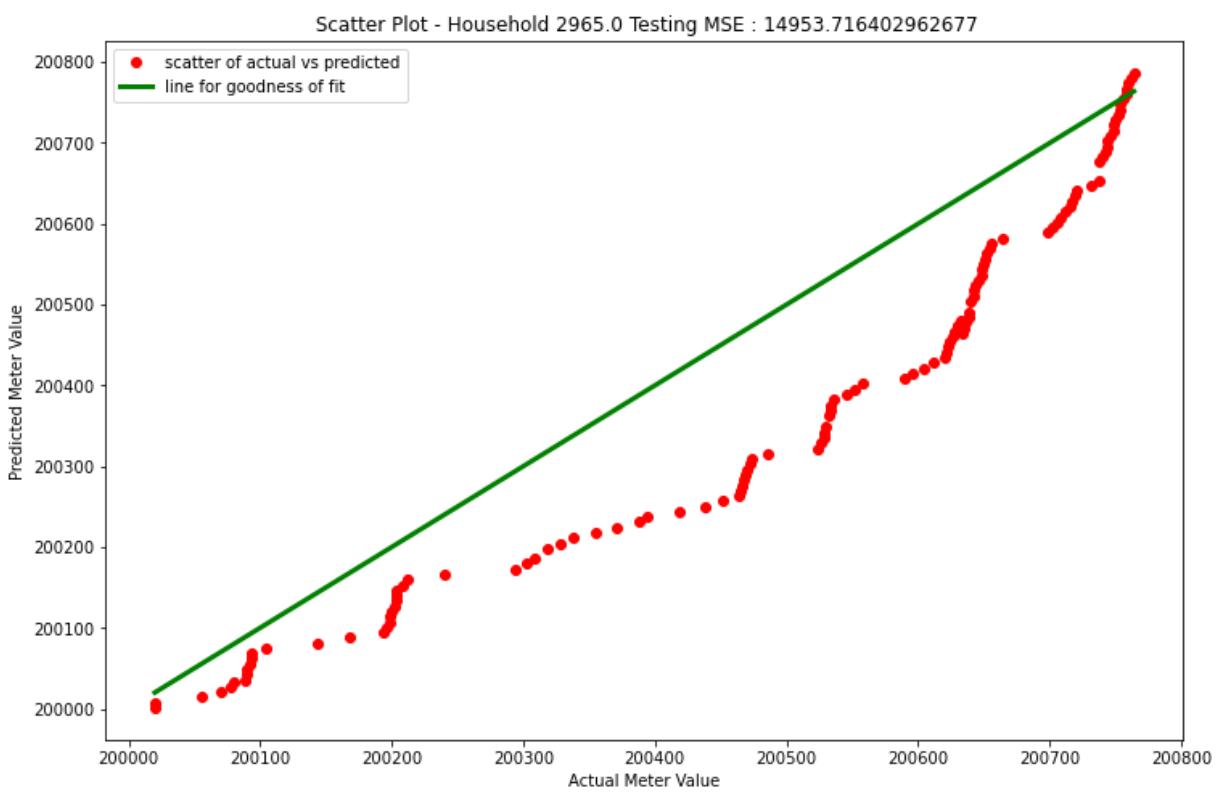
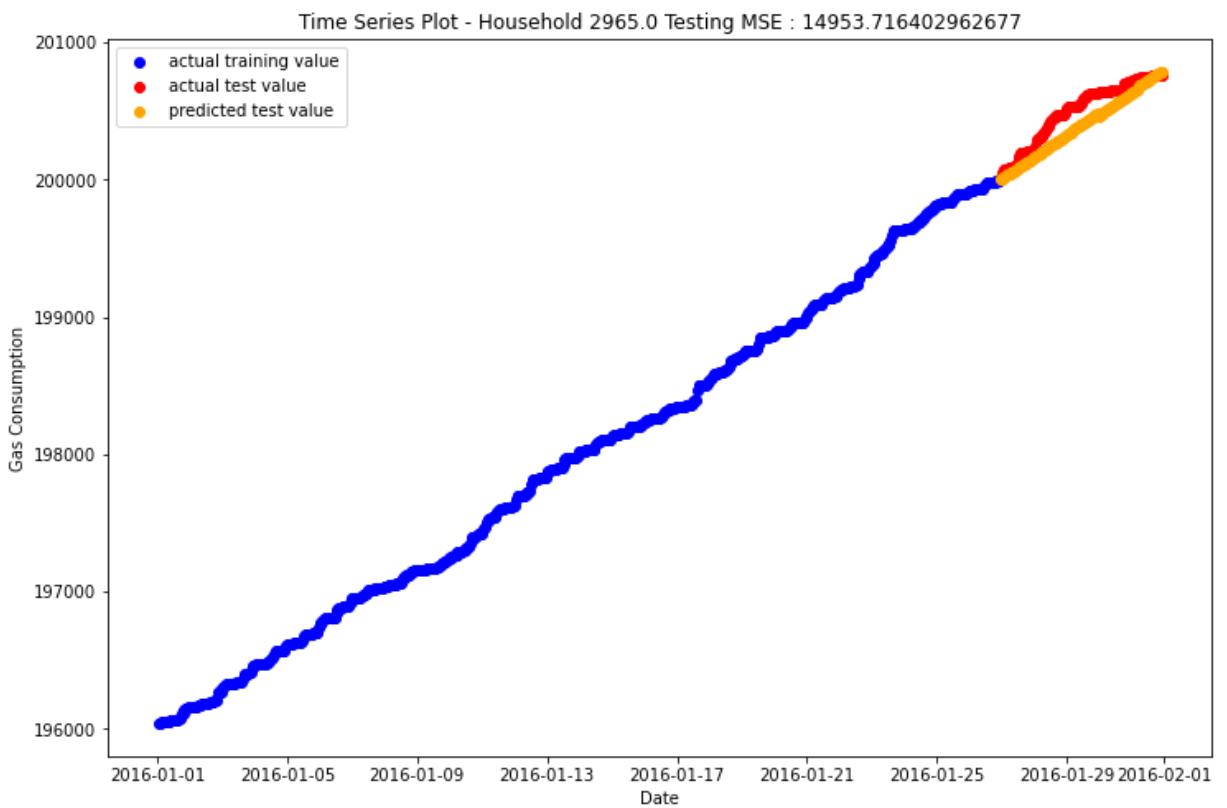




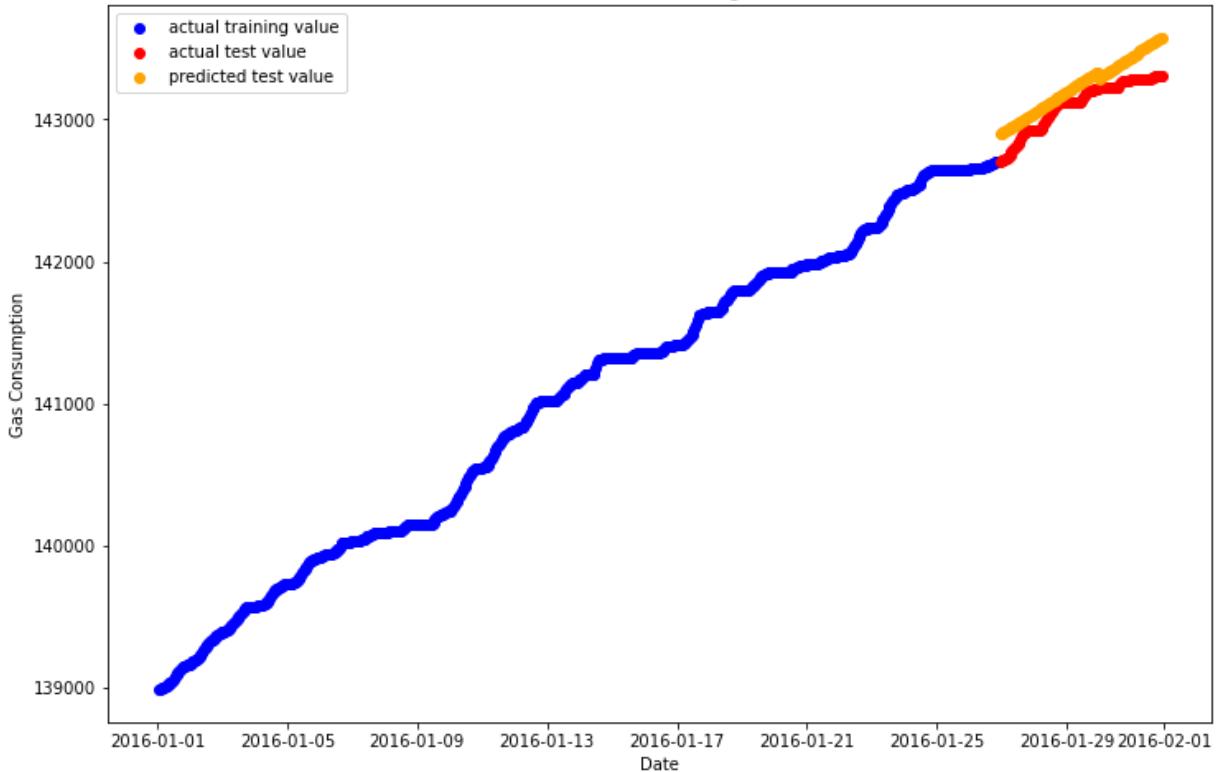




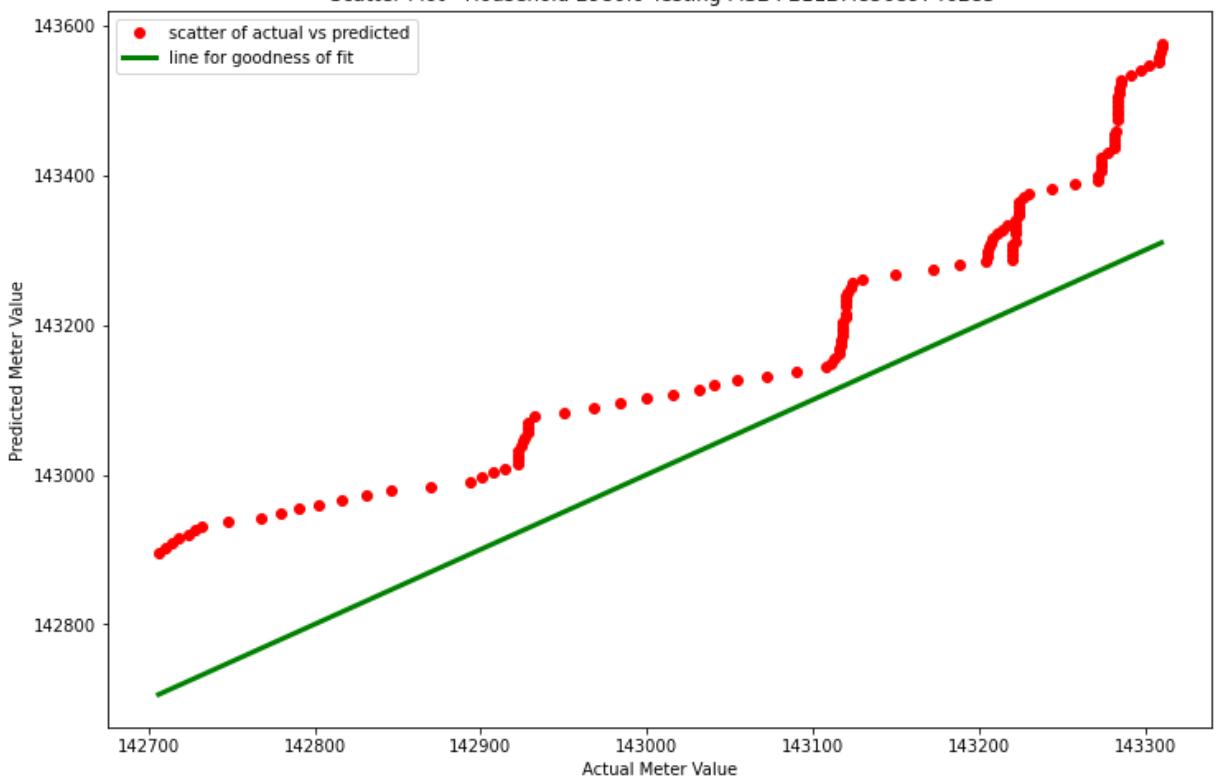


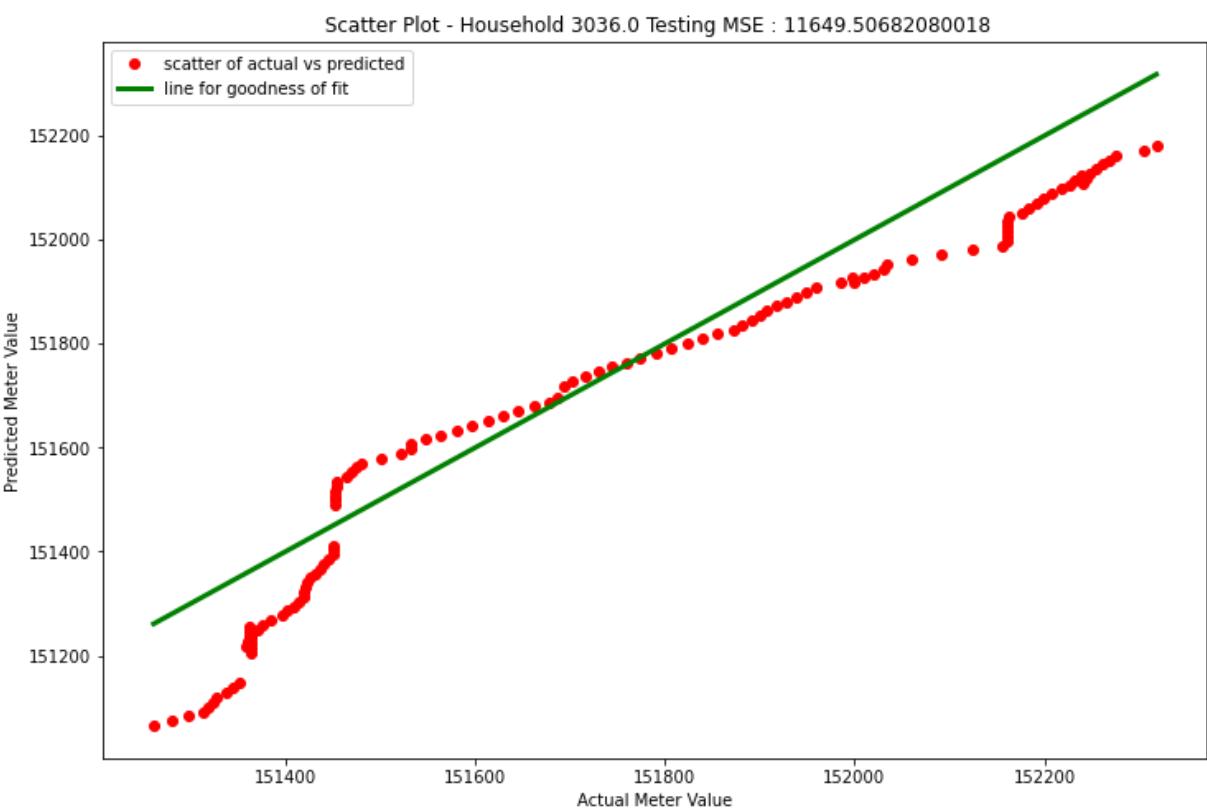
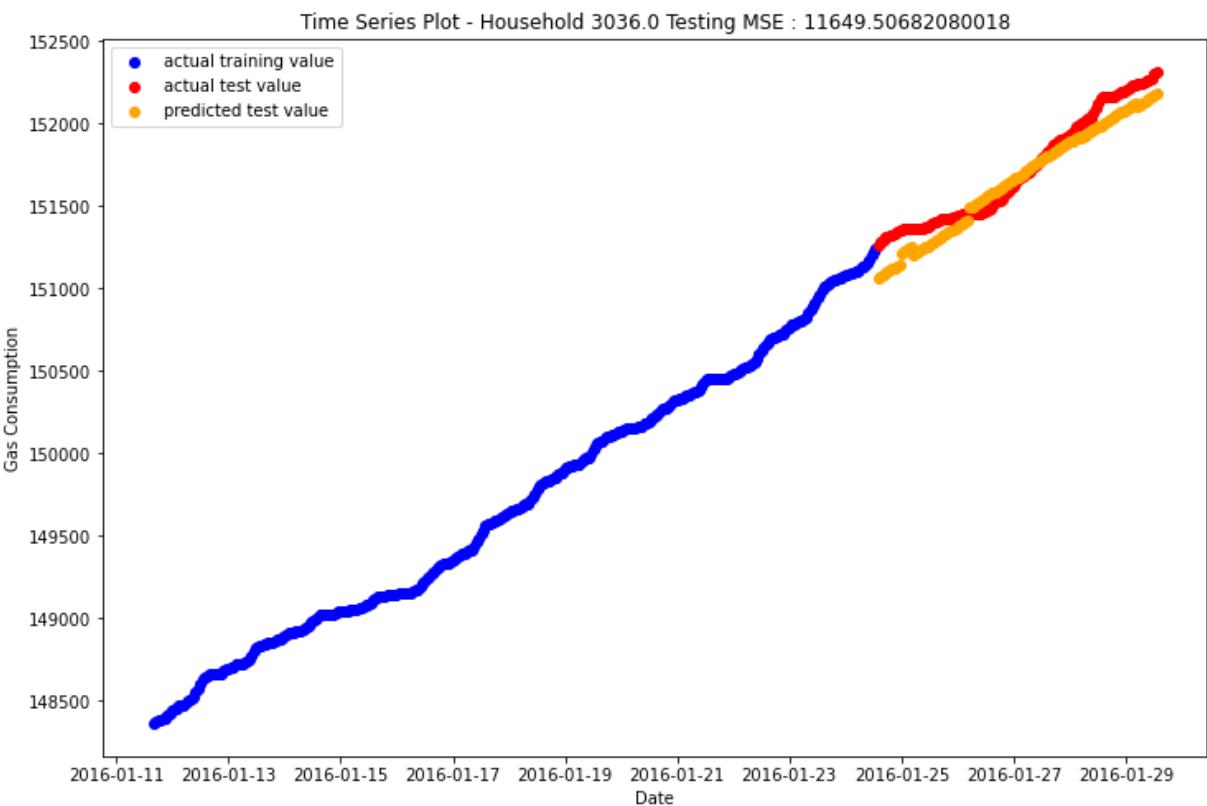


Time Series Plot - Household 2980.0 Testing MSE : 21127.85689746285

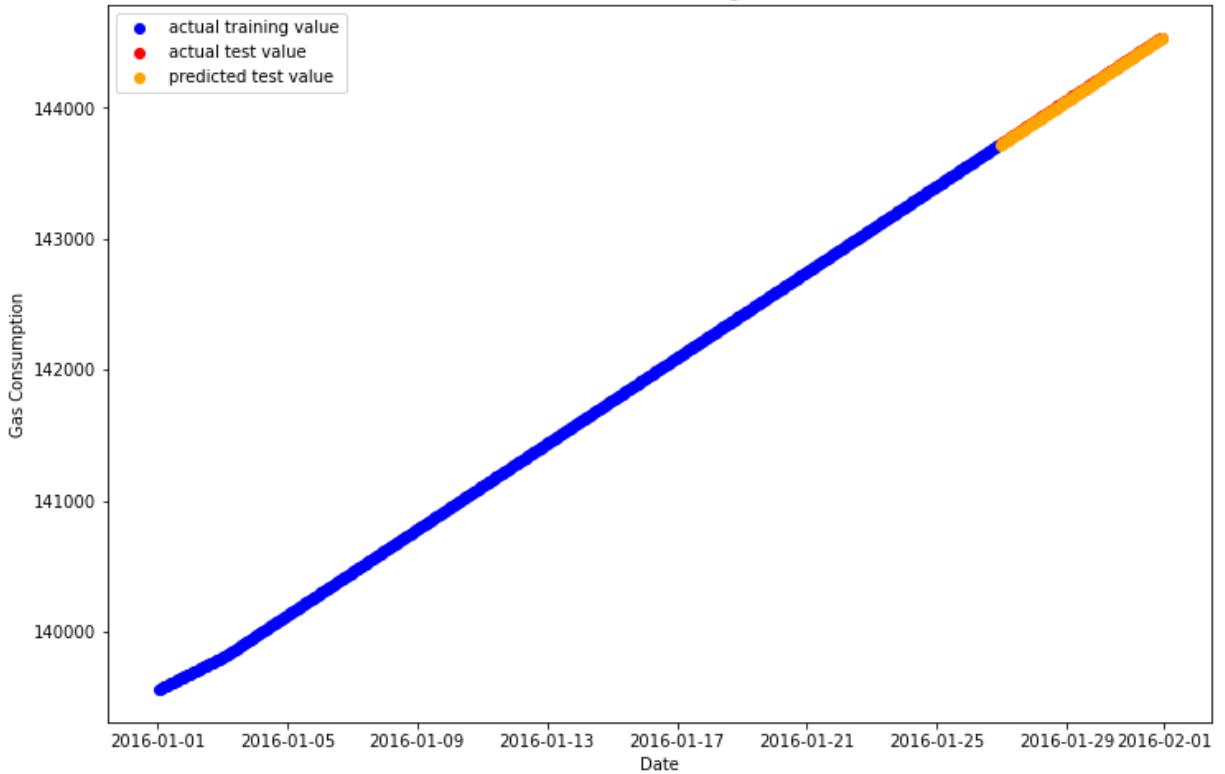


Scatter Plot - Household 2980.0 Testing MSE : 21127.85689746285

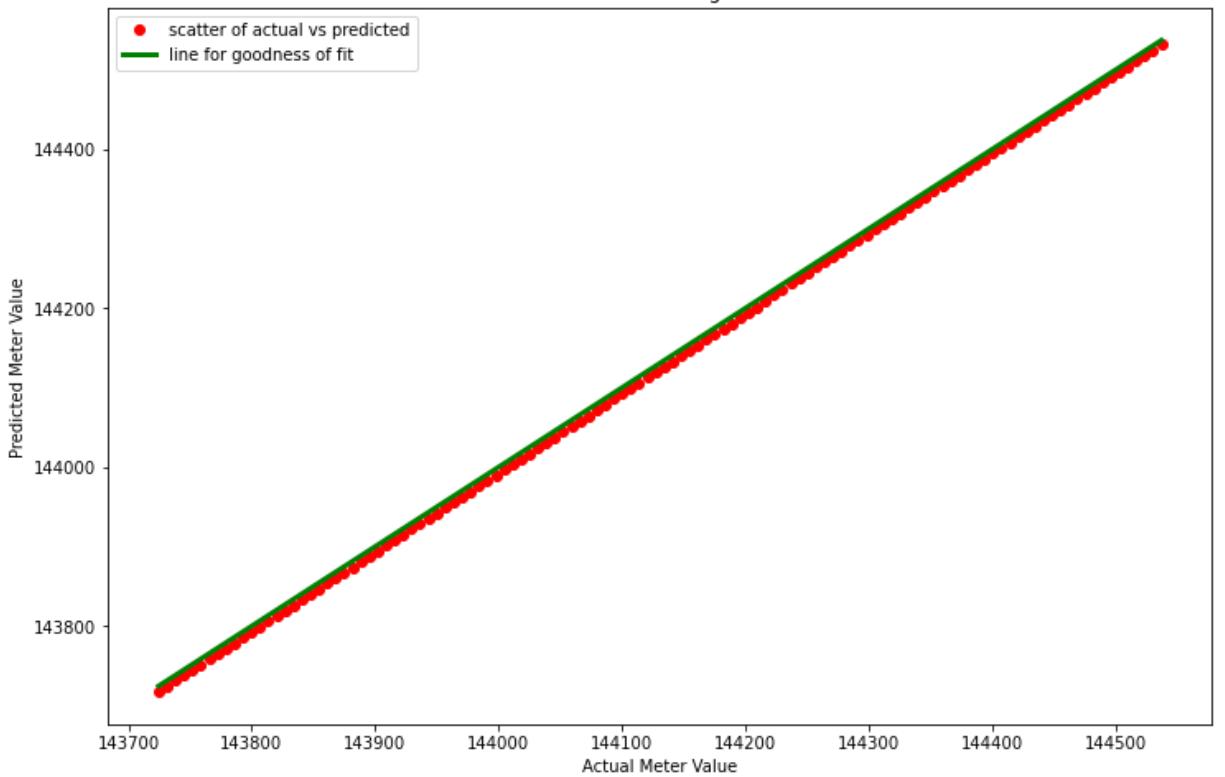




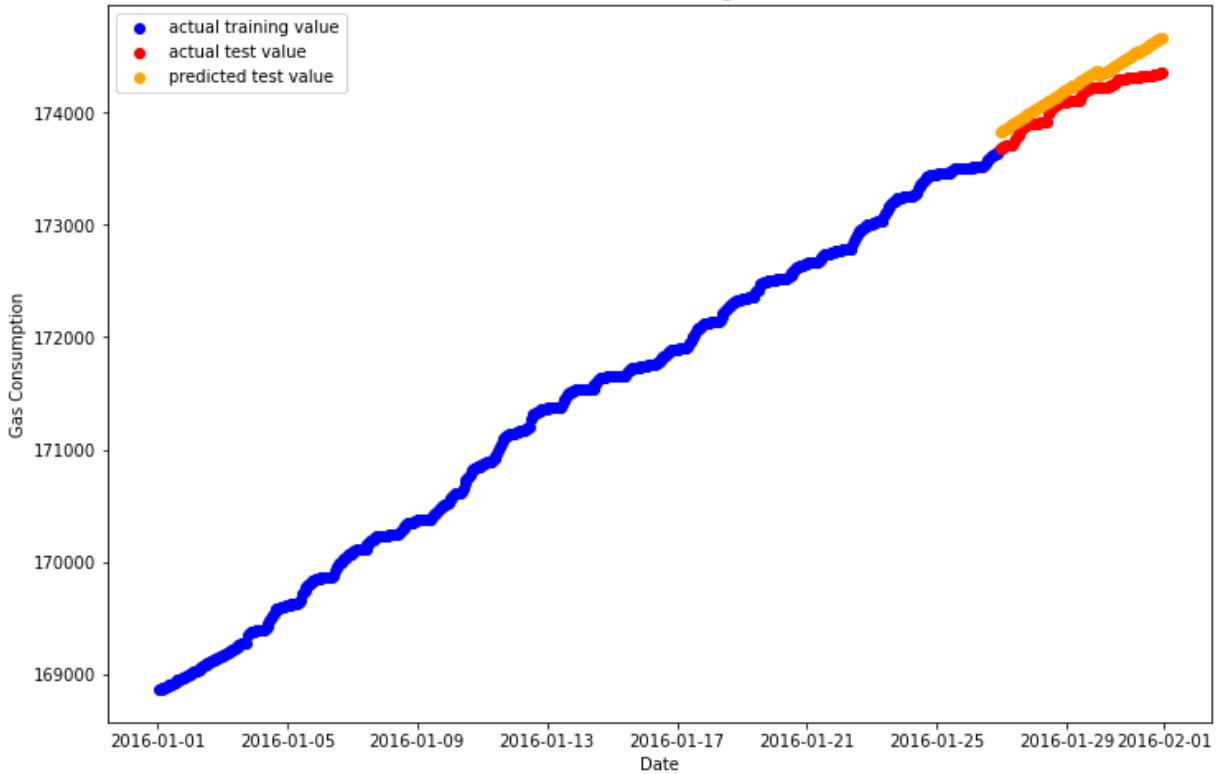
Time Series Plot - Household 3039.0 Testing MSE : 47.31391489654776



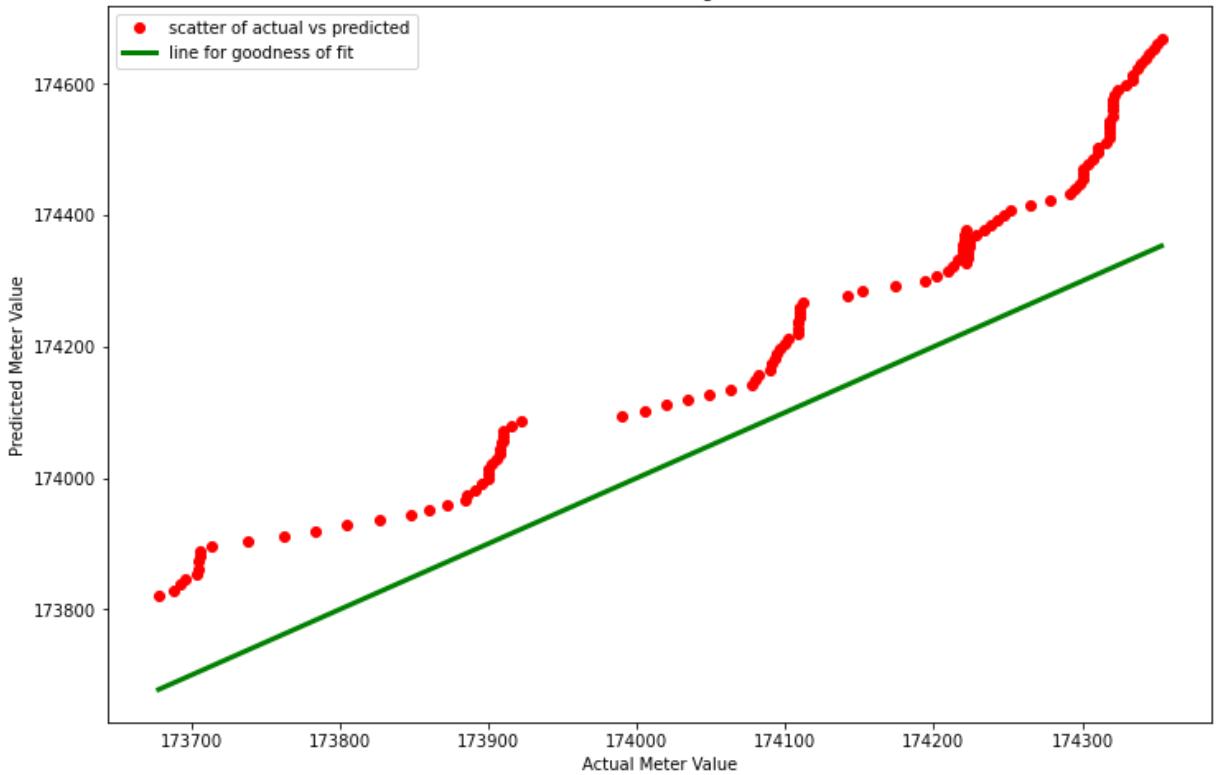
Scatter Plot - Household 3039.0 Testing MSE : 47.31391489654776

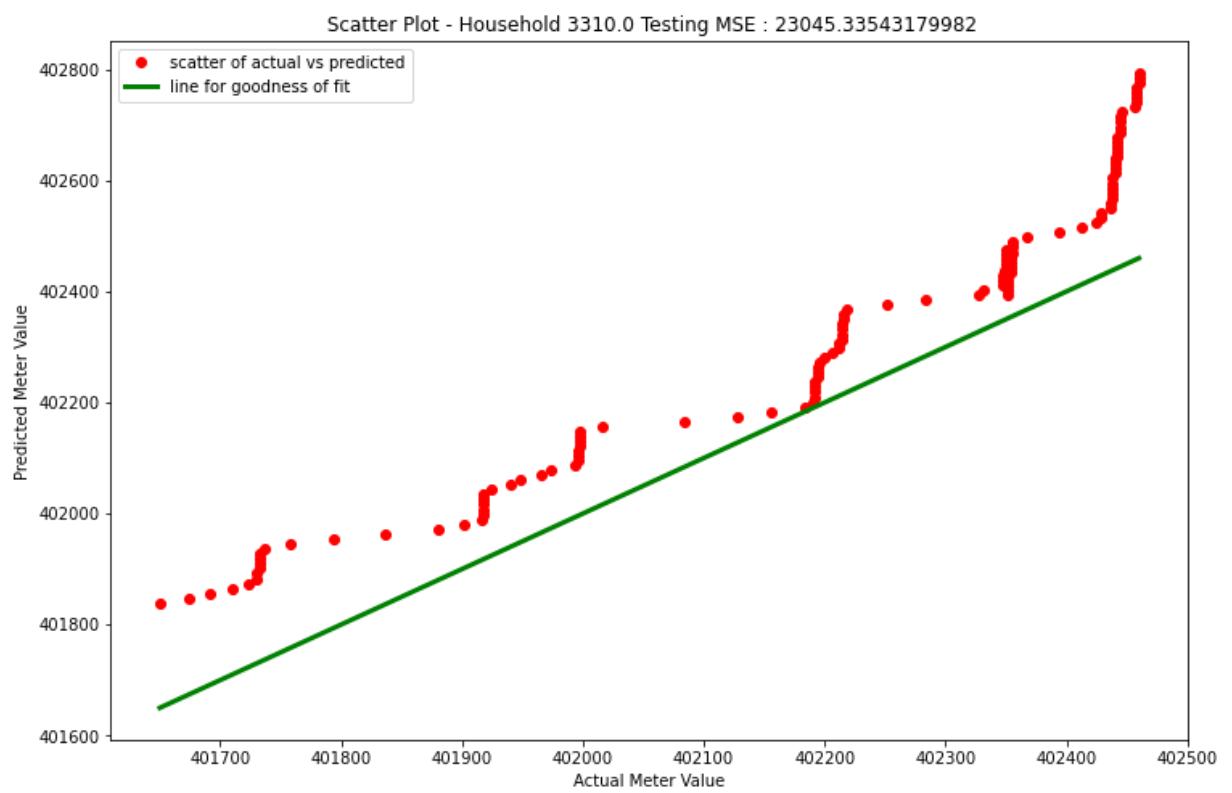
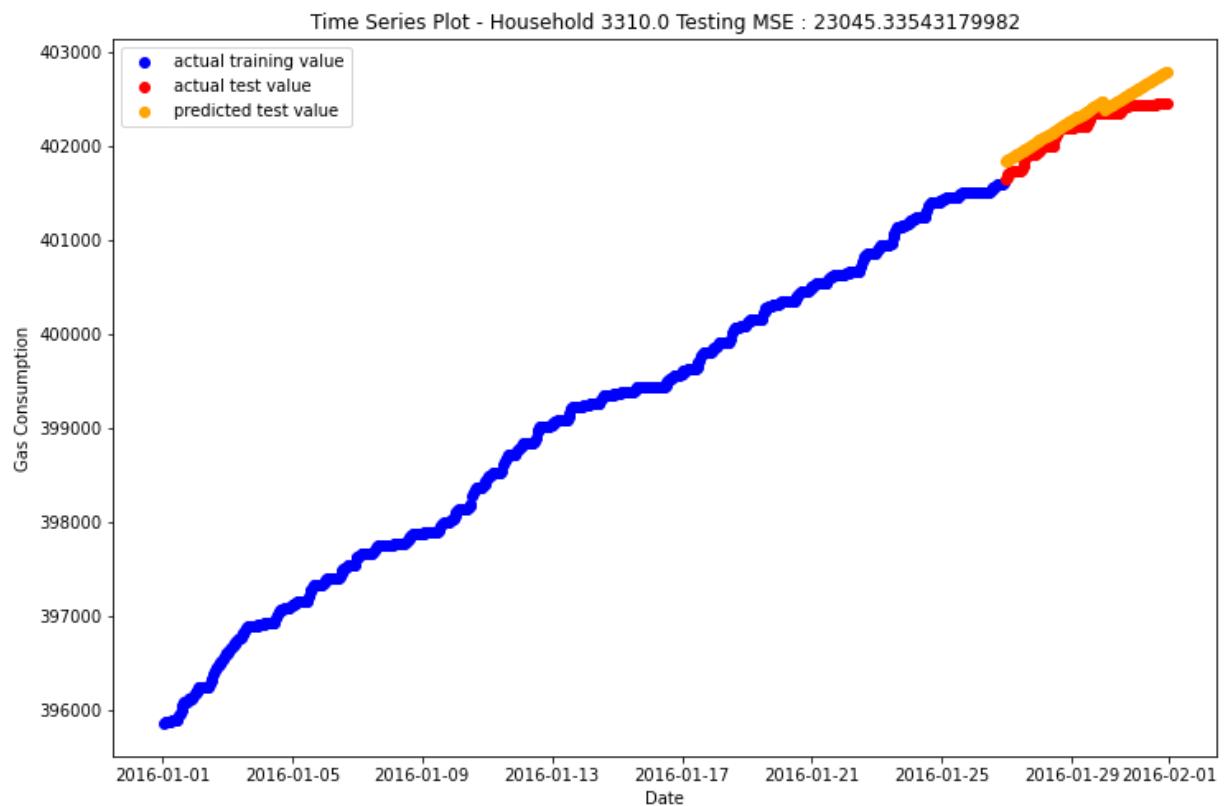


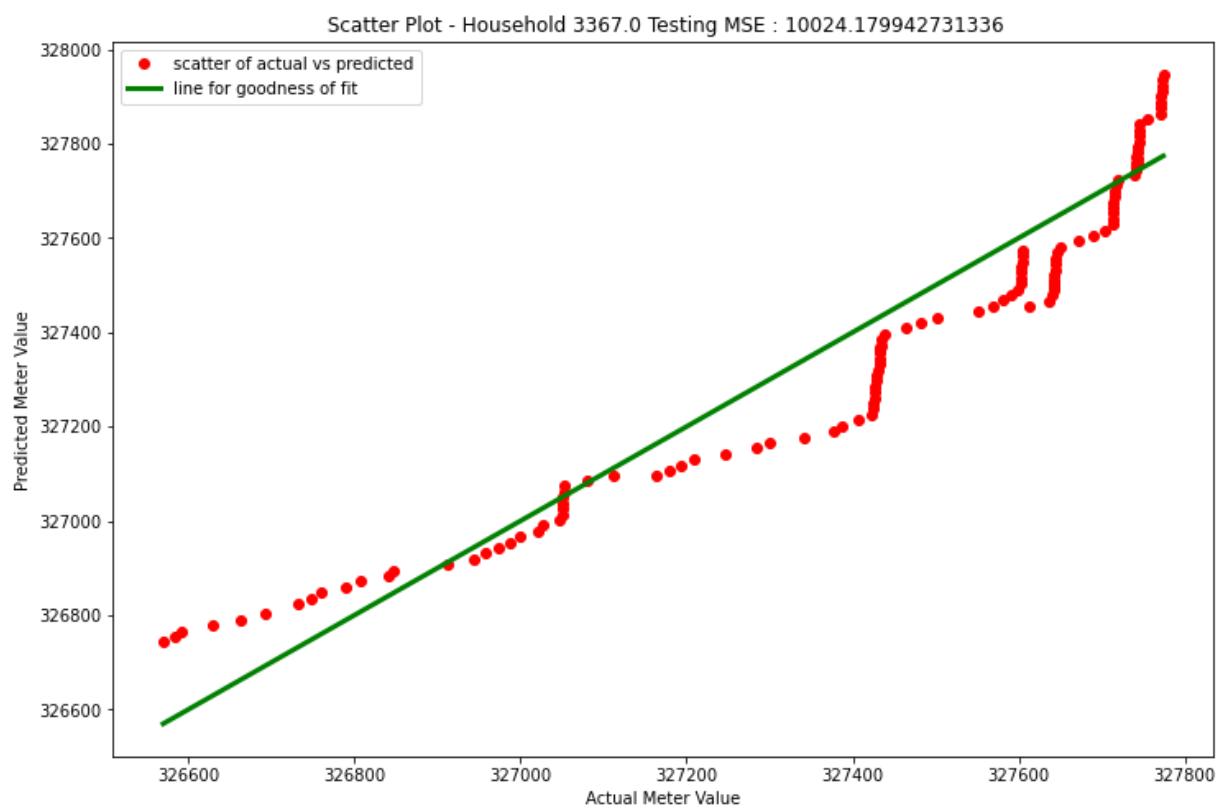
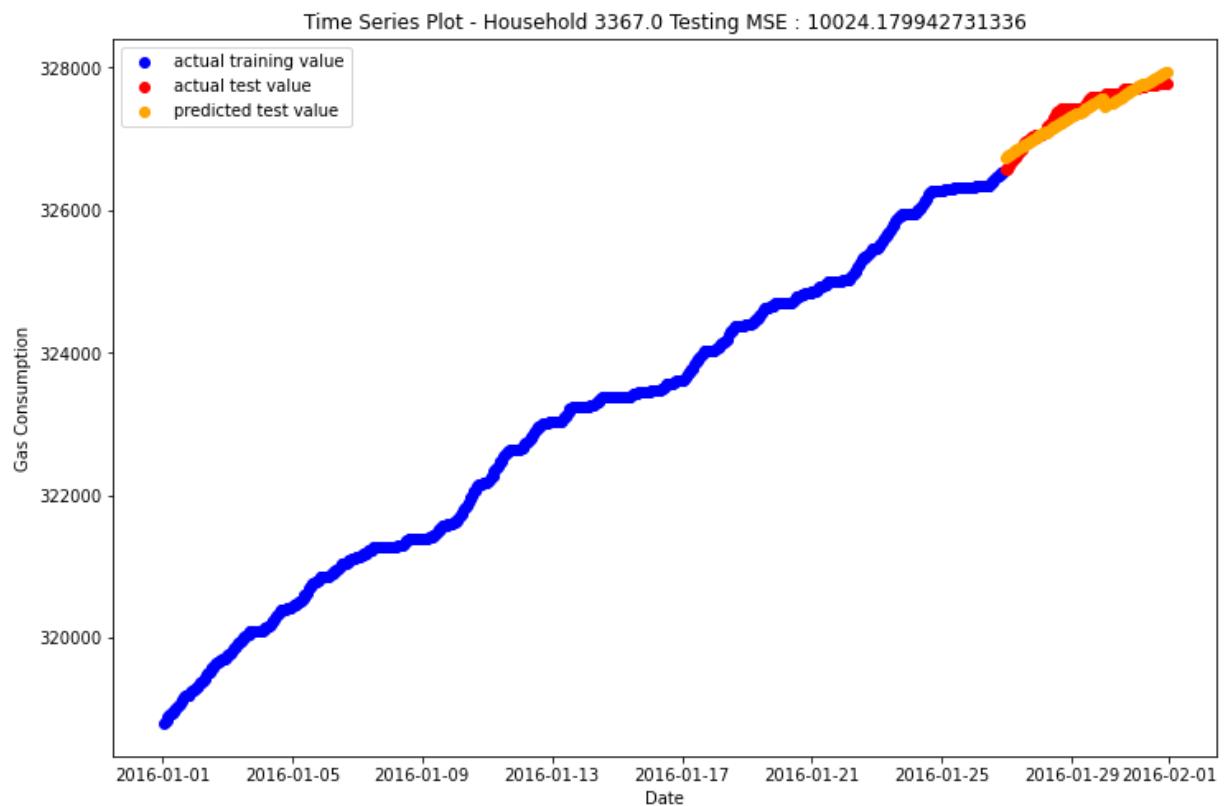
Time Series Plot - Household 3134.0 Testing MSE : 27042.011703239237

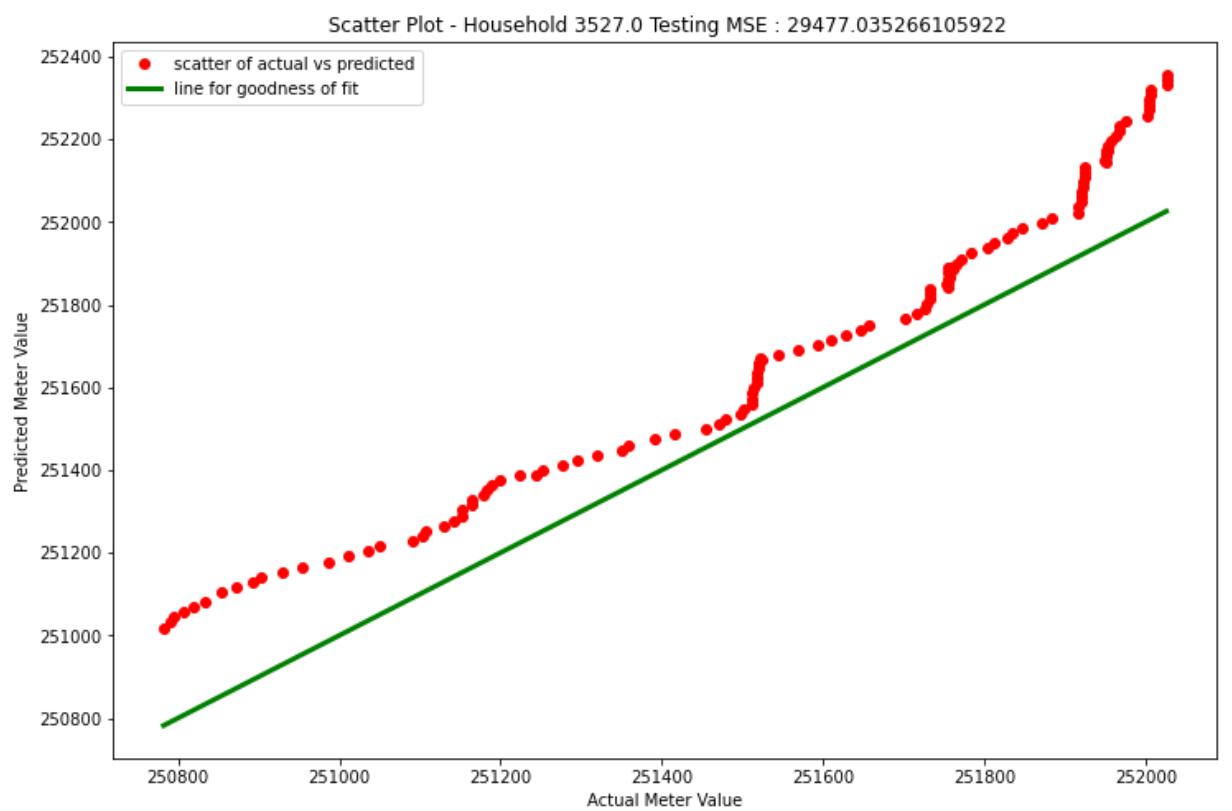
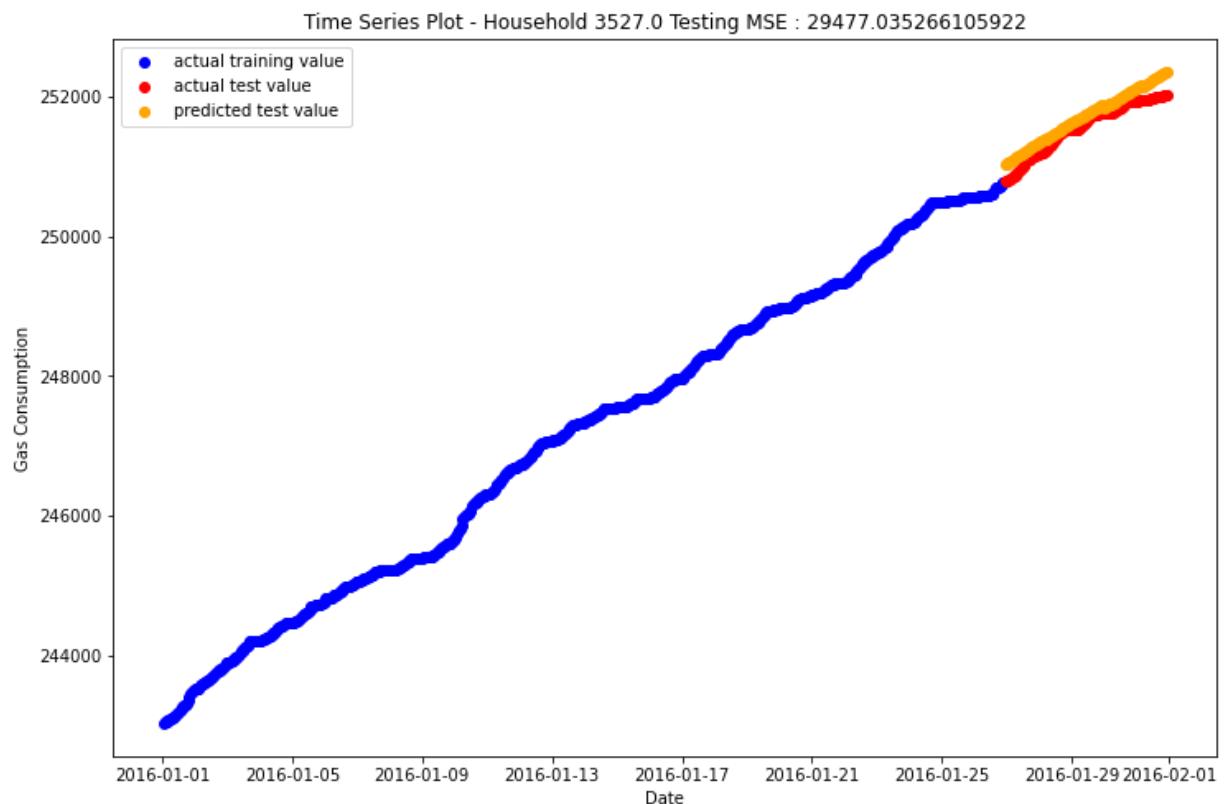


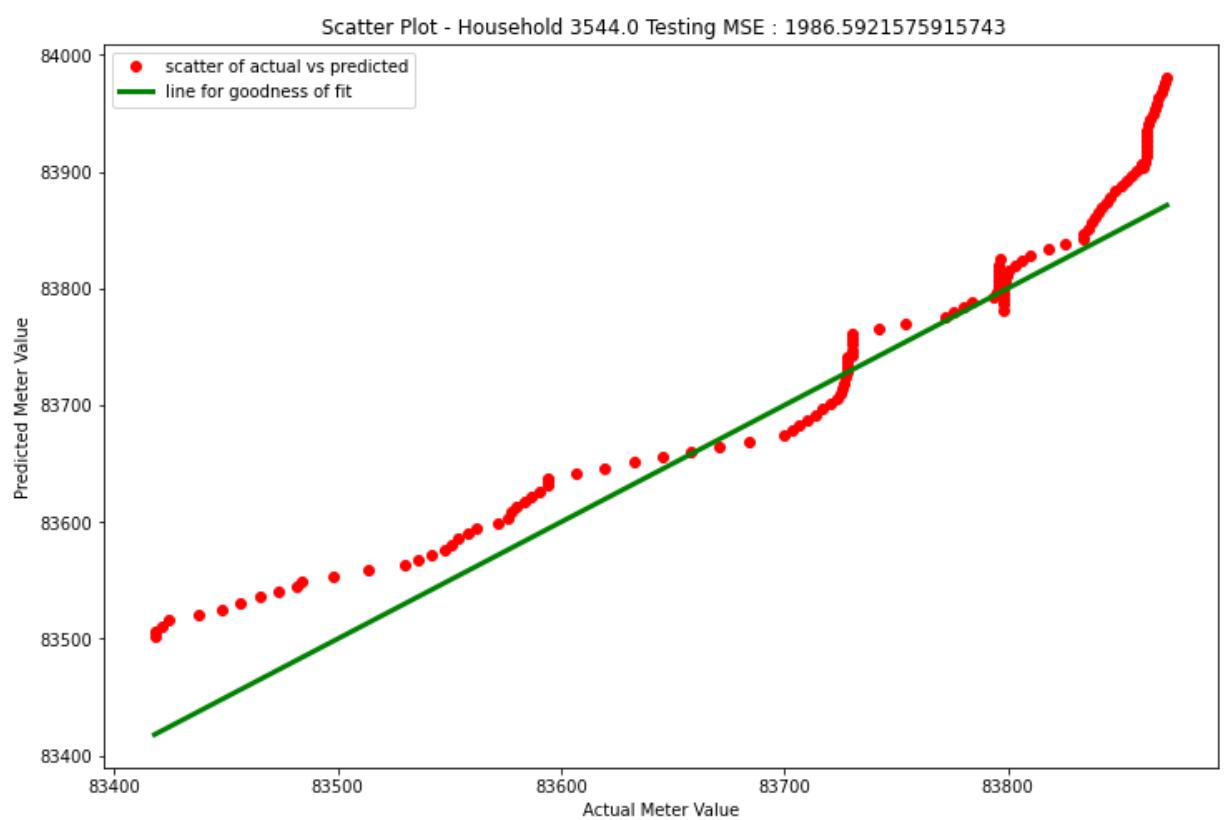
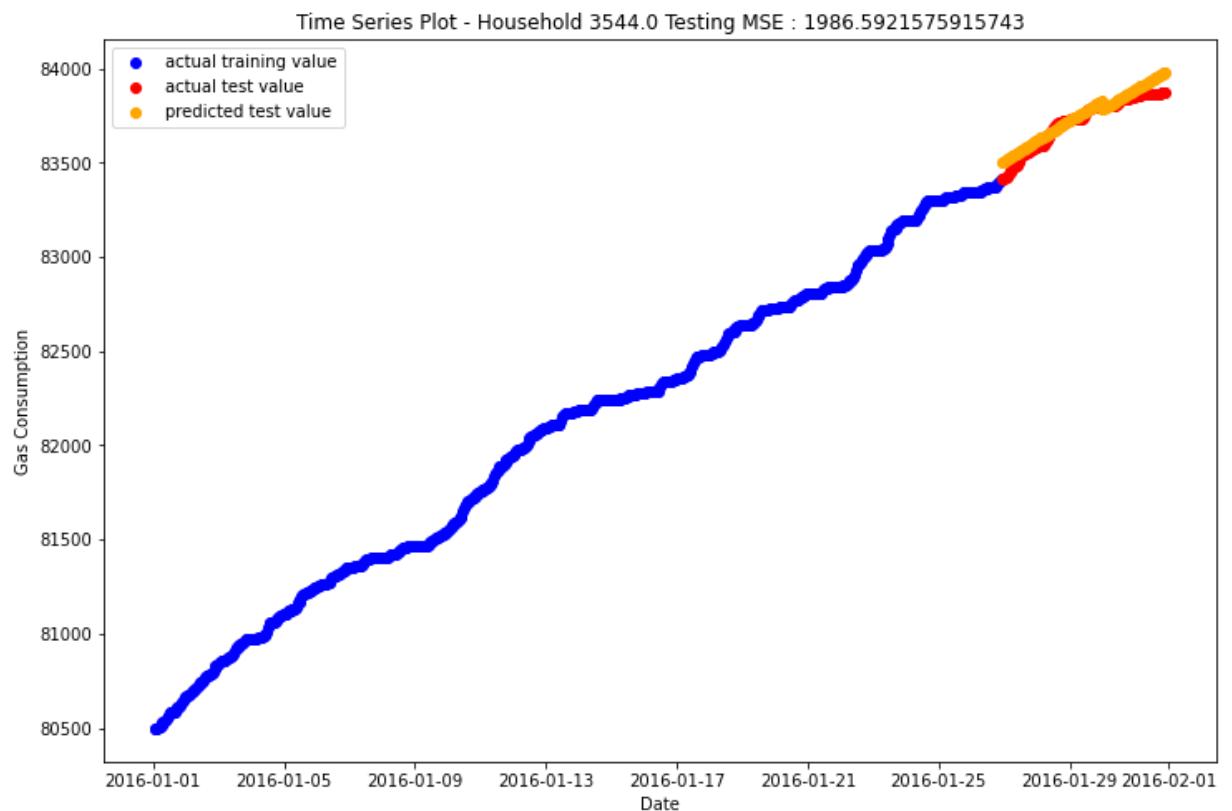
Scatter Plot - Household 3134.0 Testing MSE : 27042.011703239237



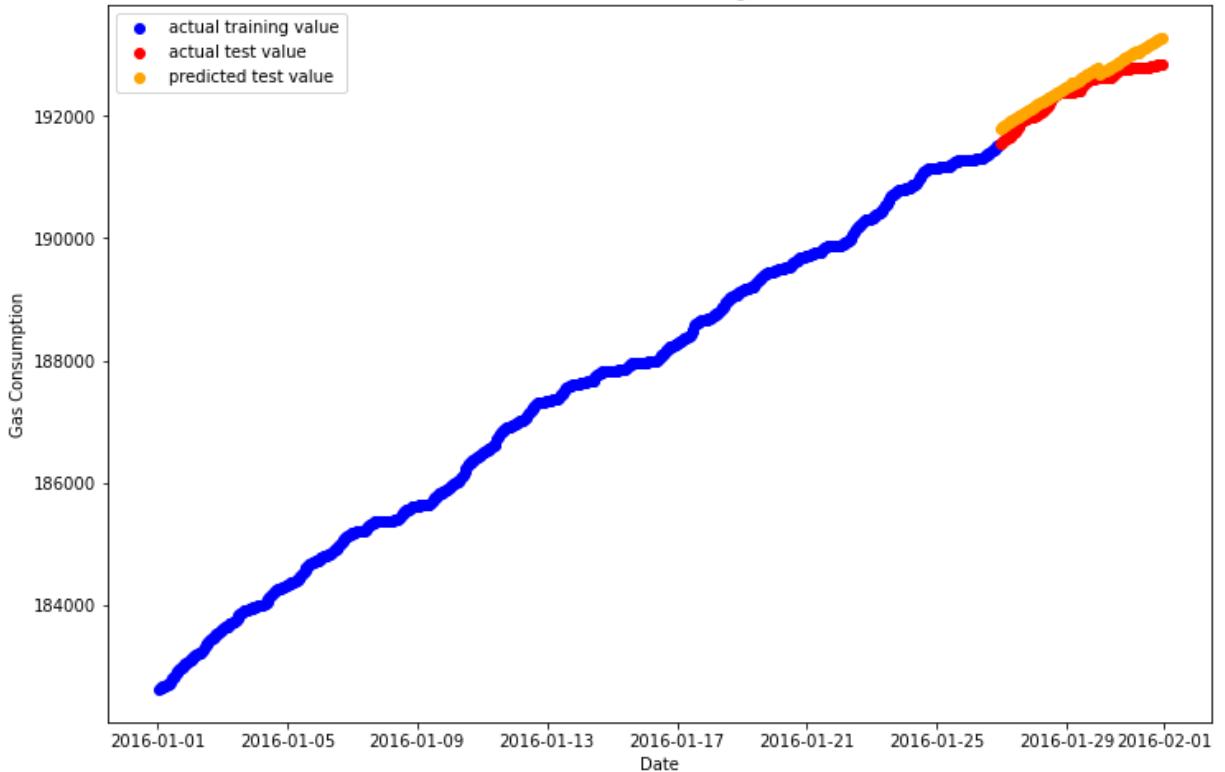




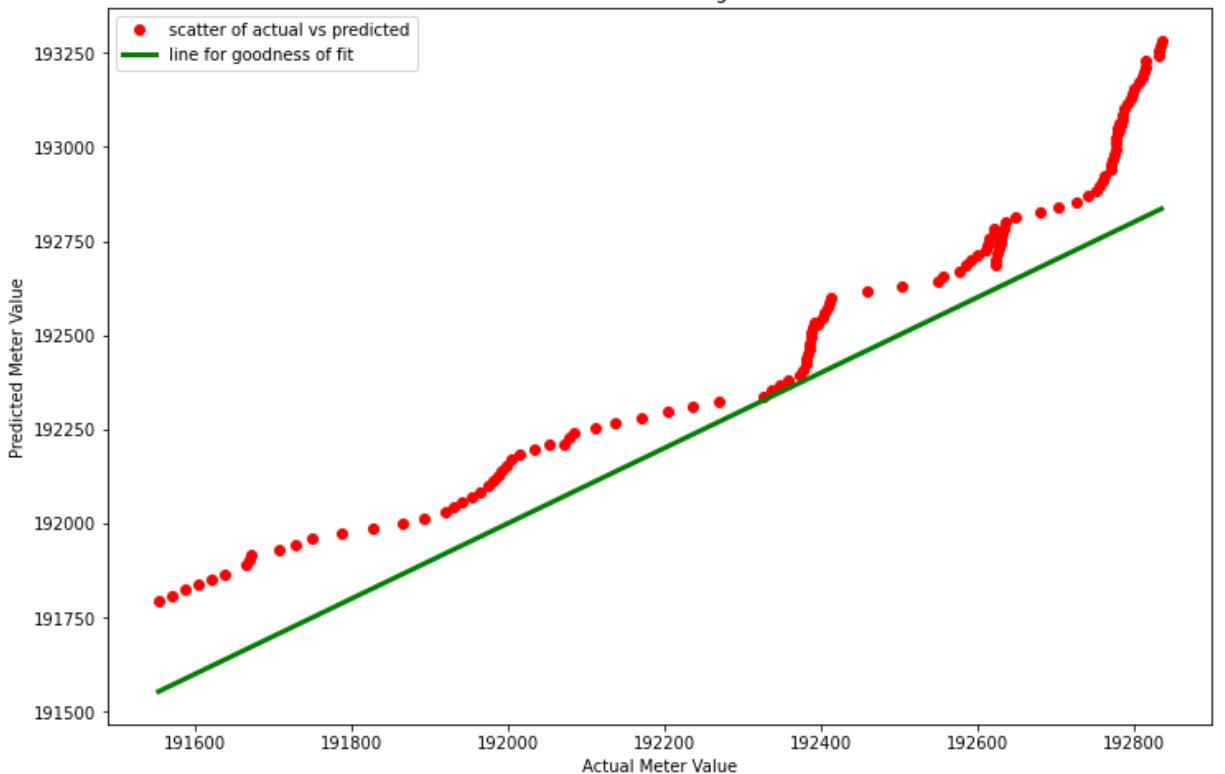


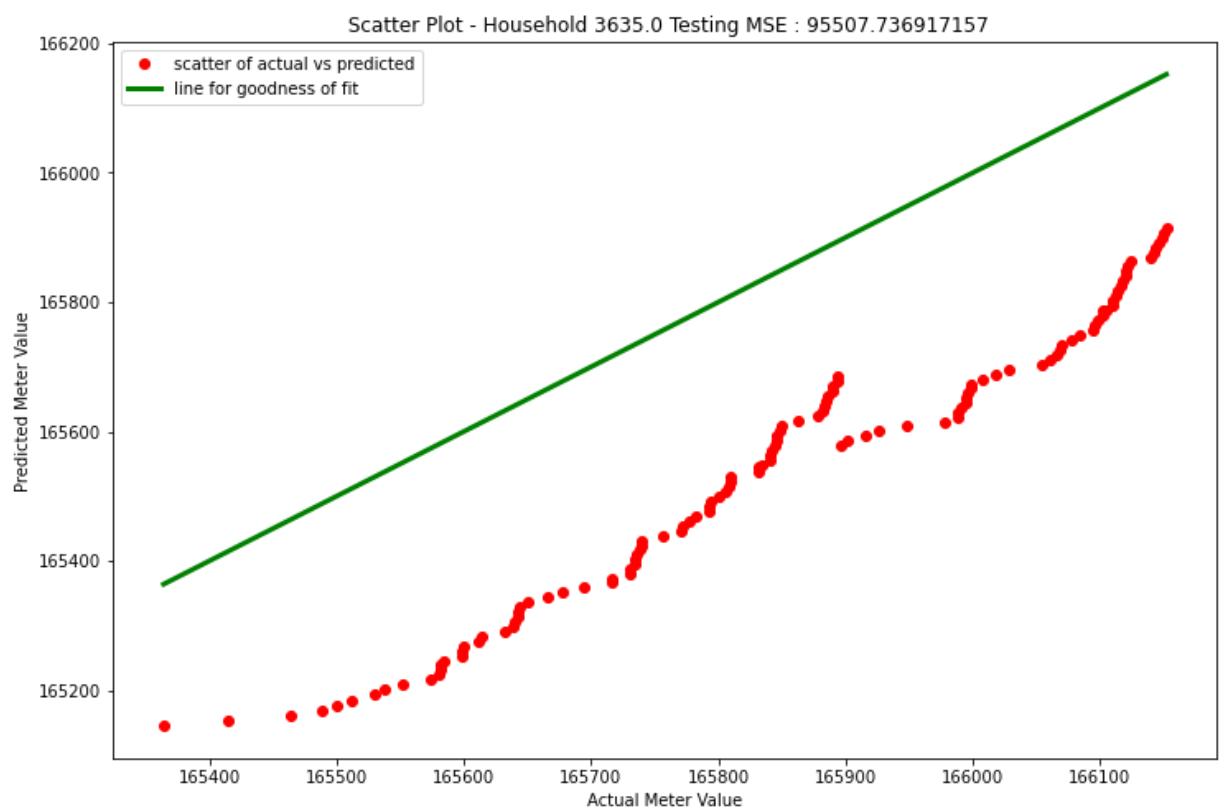
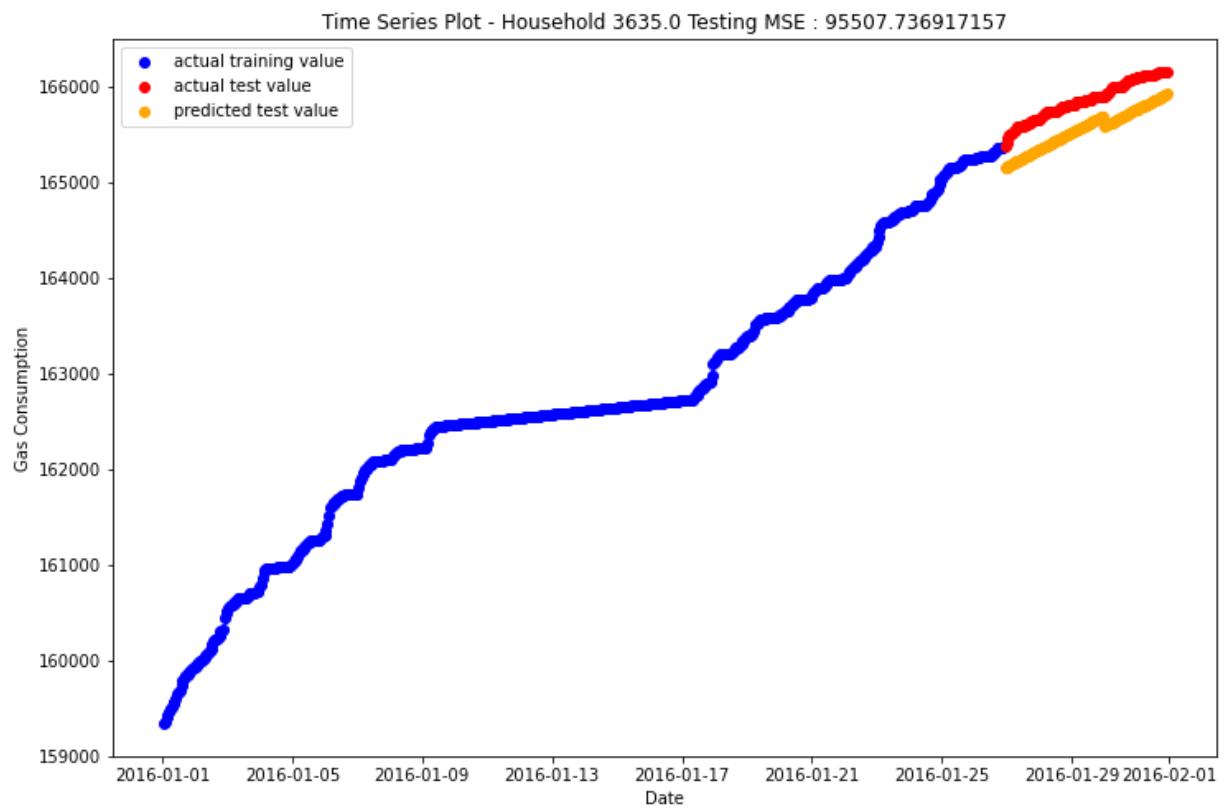


Time Series Plot - Household 3577.0 Testing MSE : 39821.4134416316

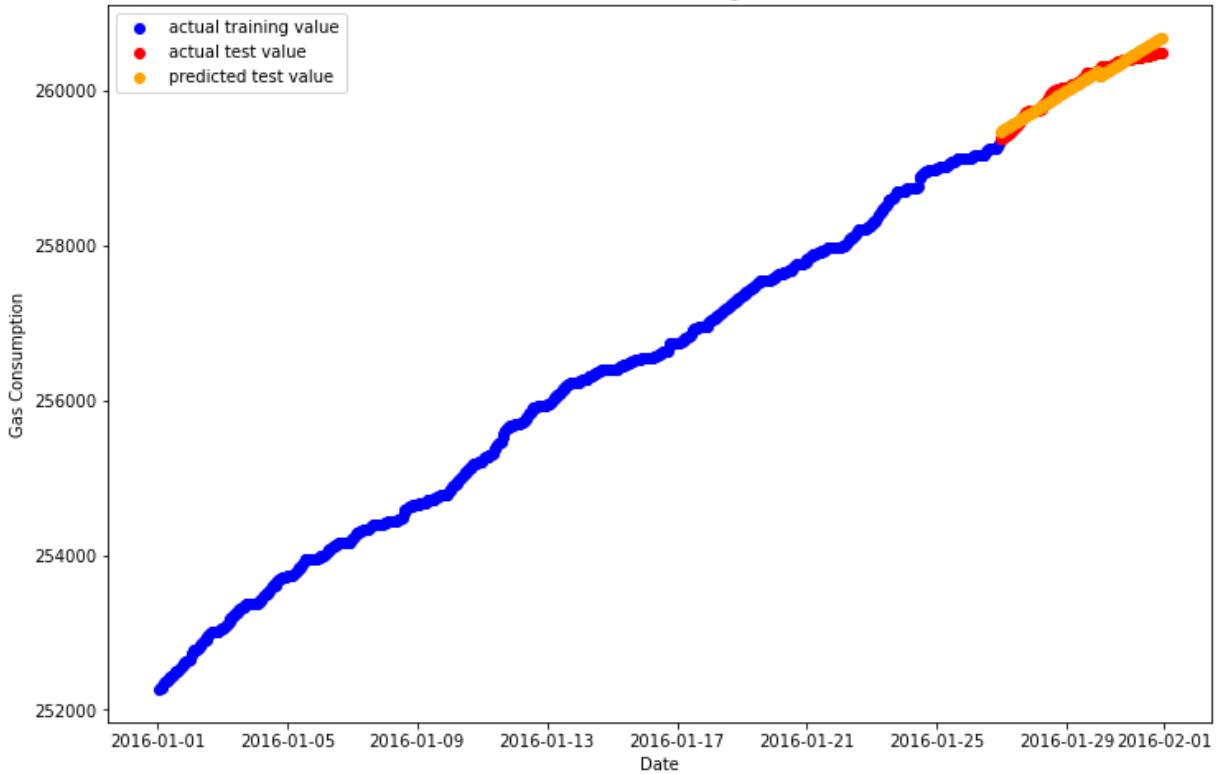


Scatter Plot - Household 3577.0 Testing MSE : 39821.4134416316

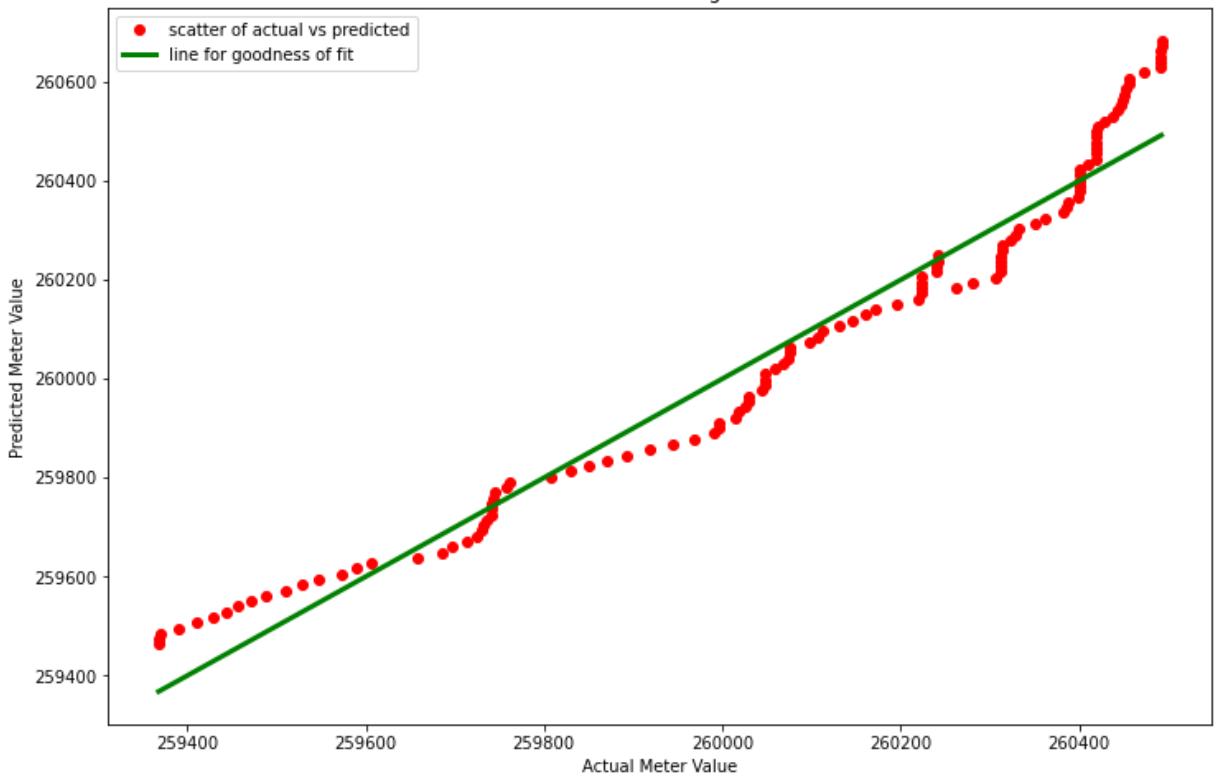


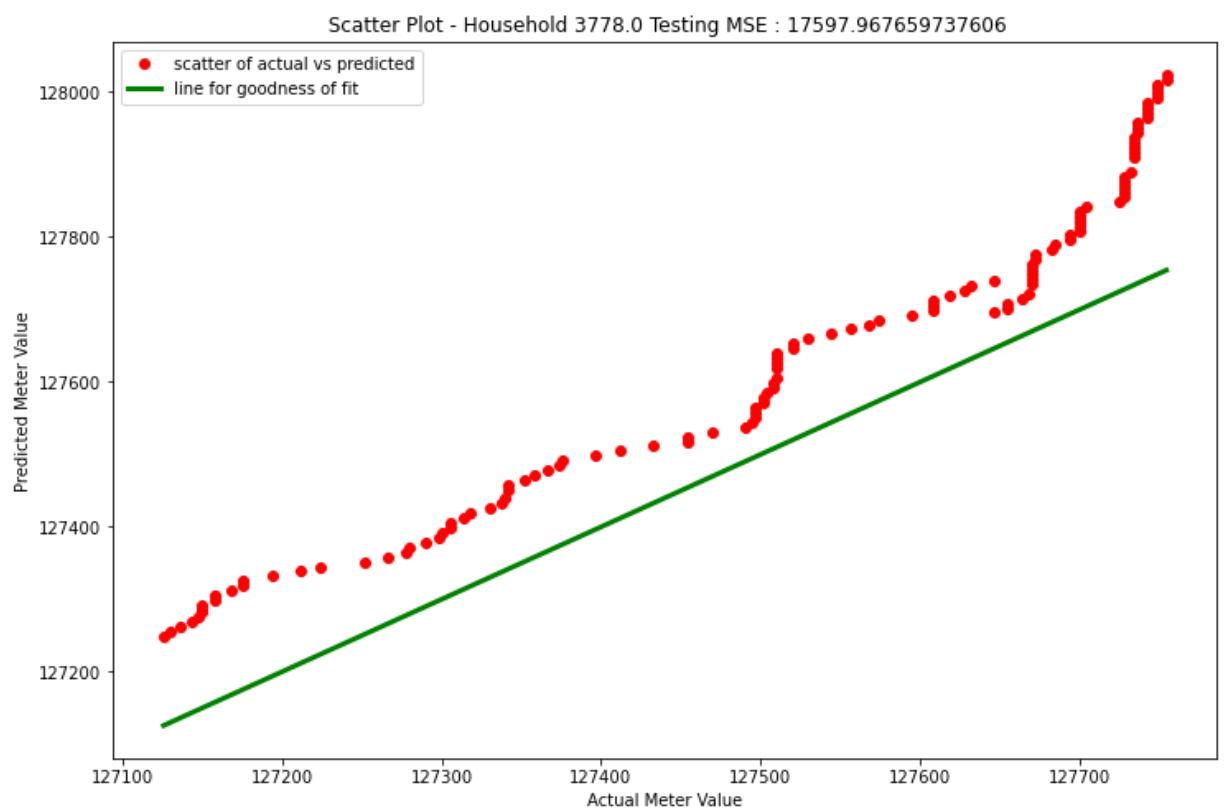
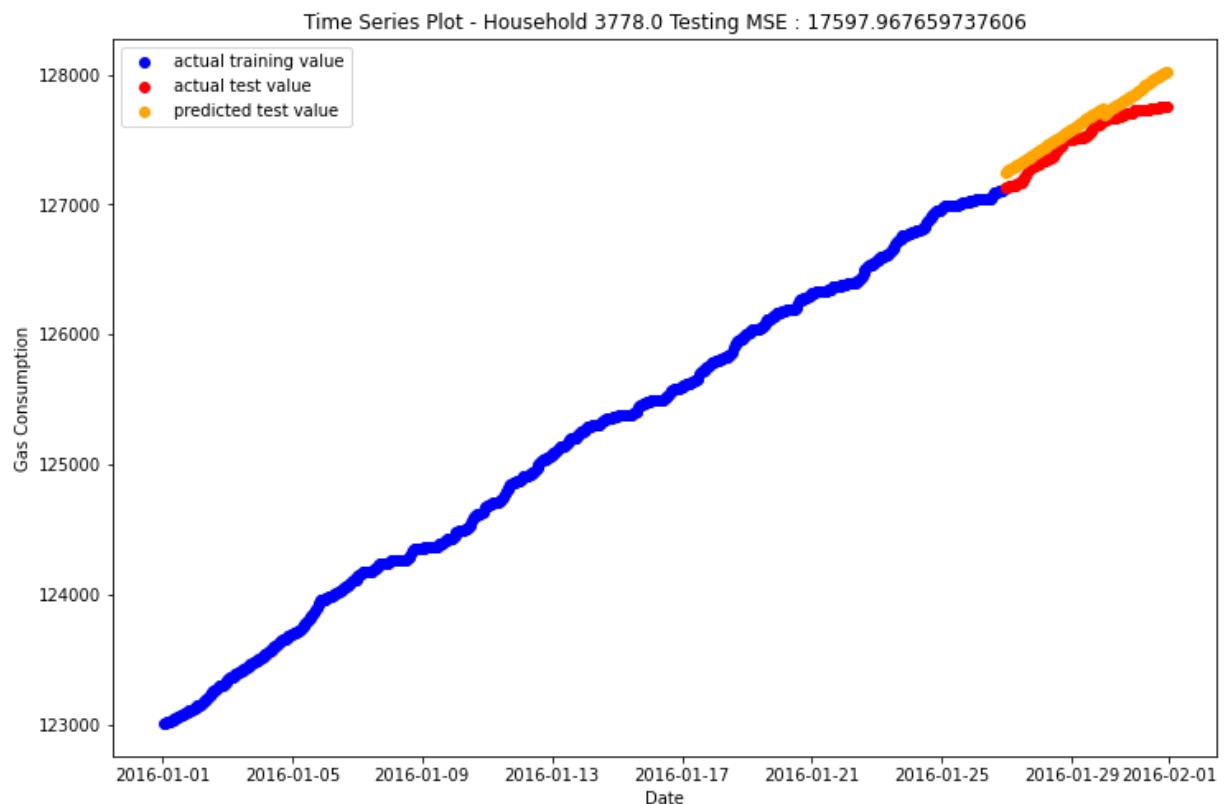


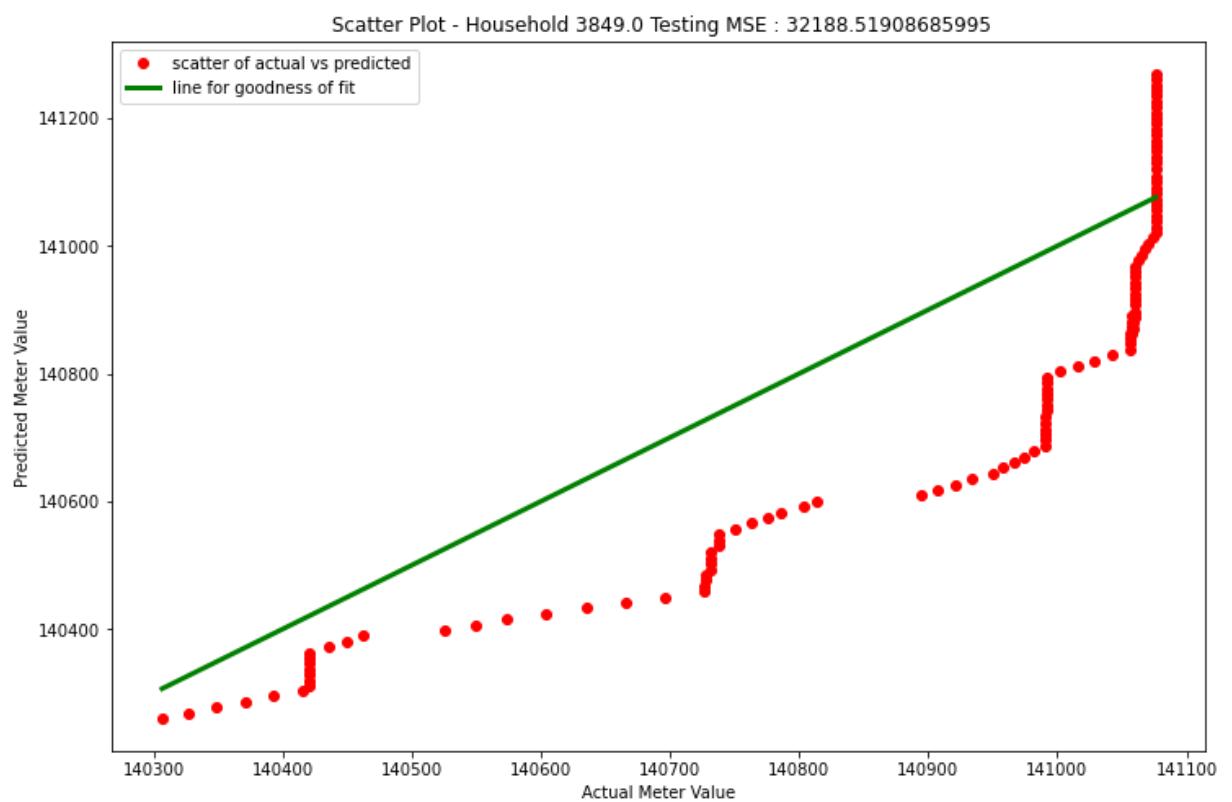
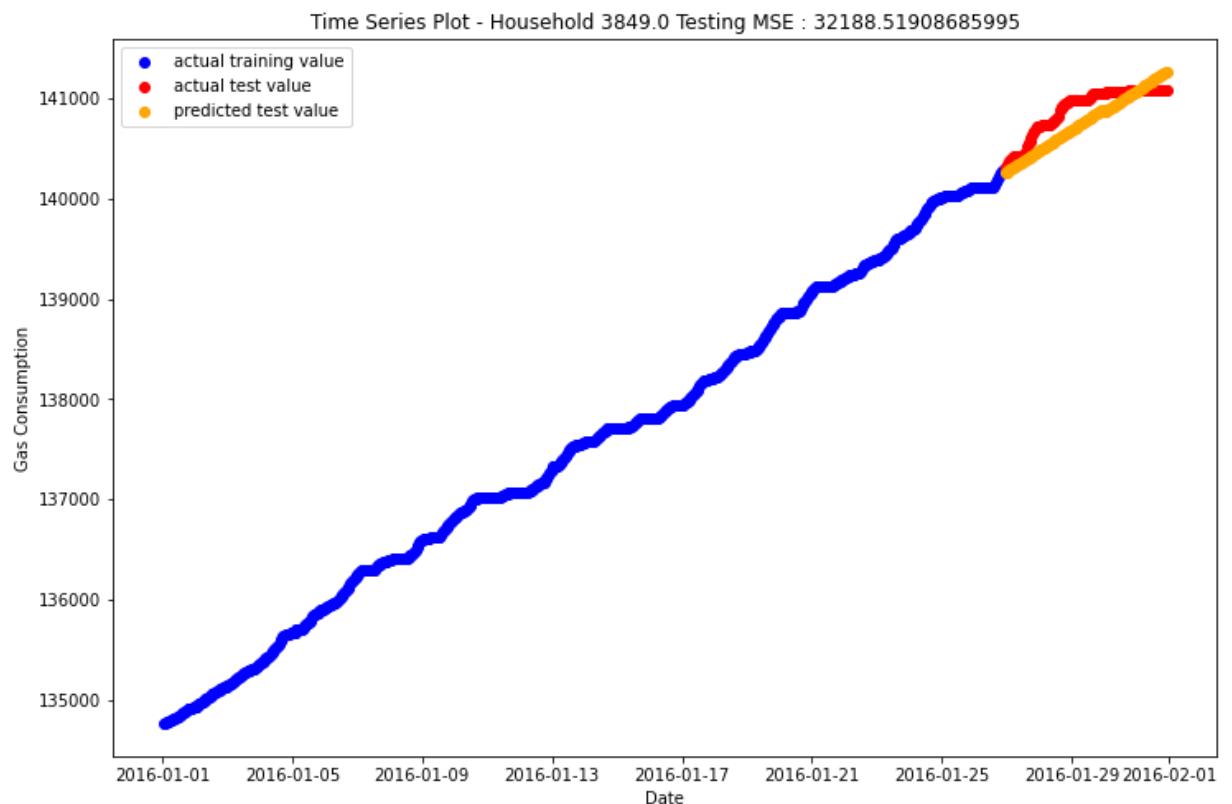
Time Series Plot - Household 3723.0 Testing MSE : 5309.990352939731

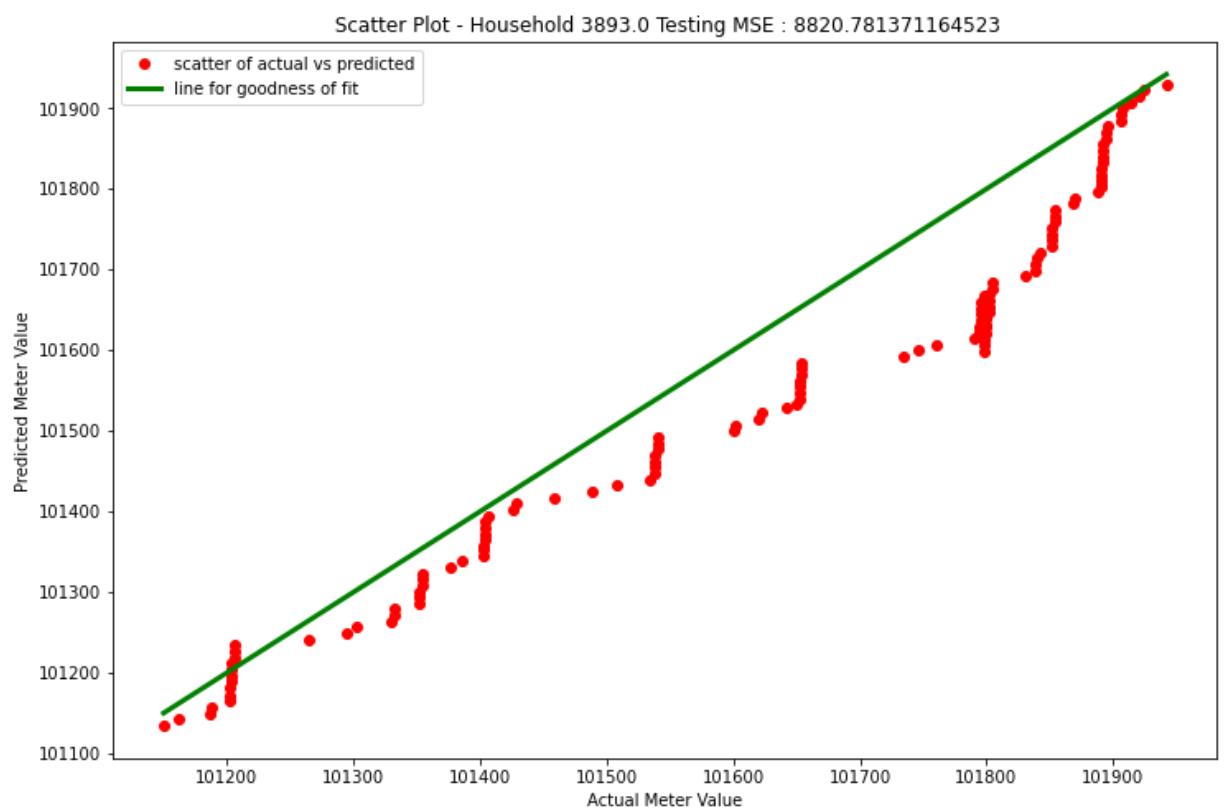
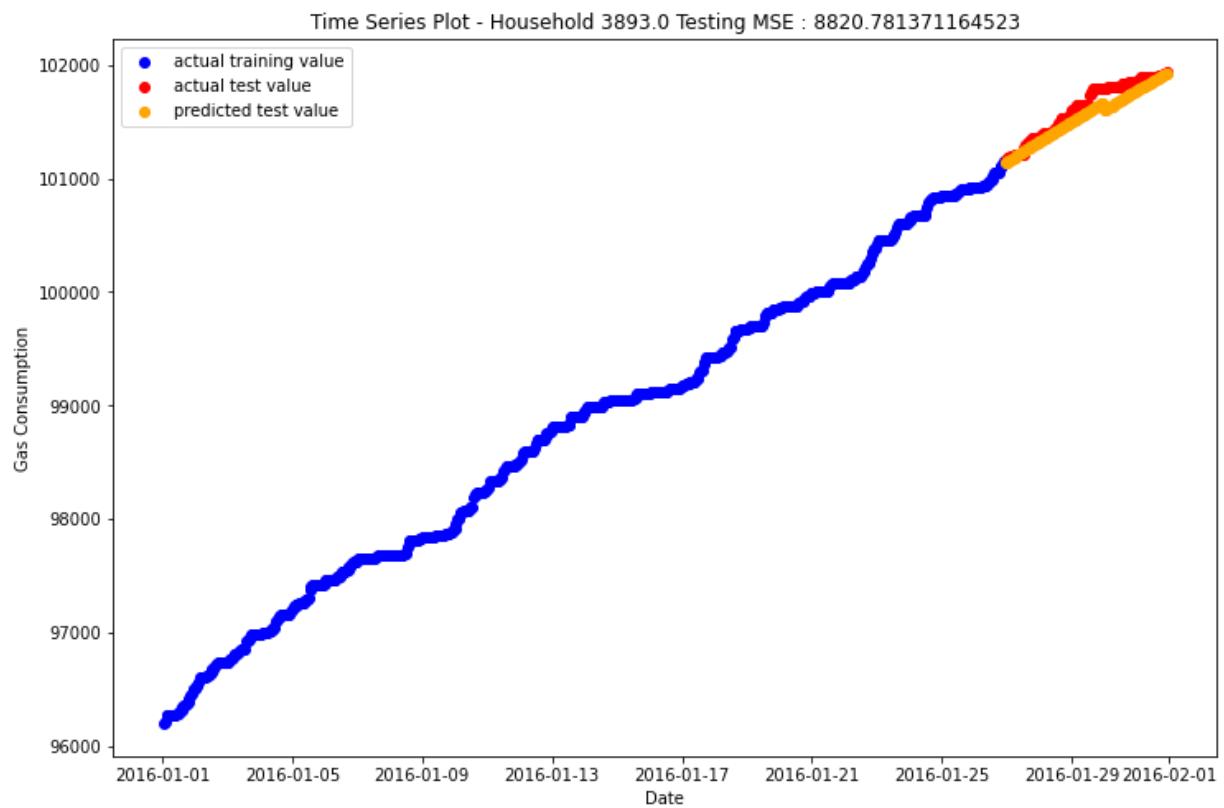


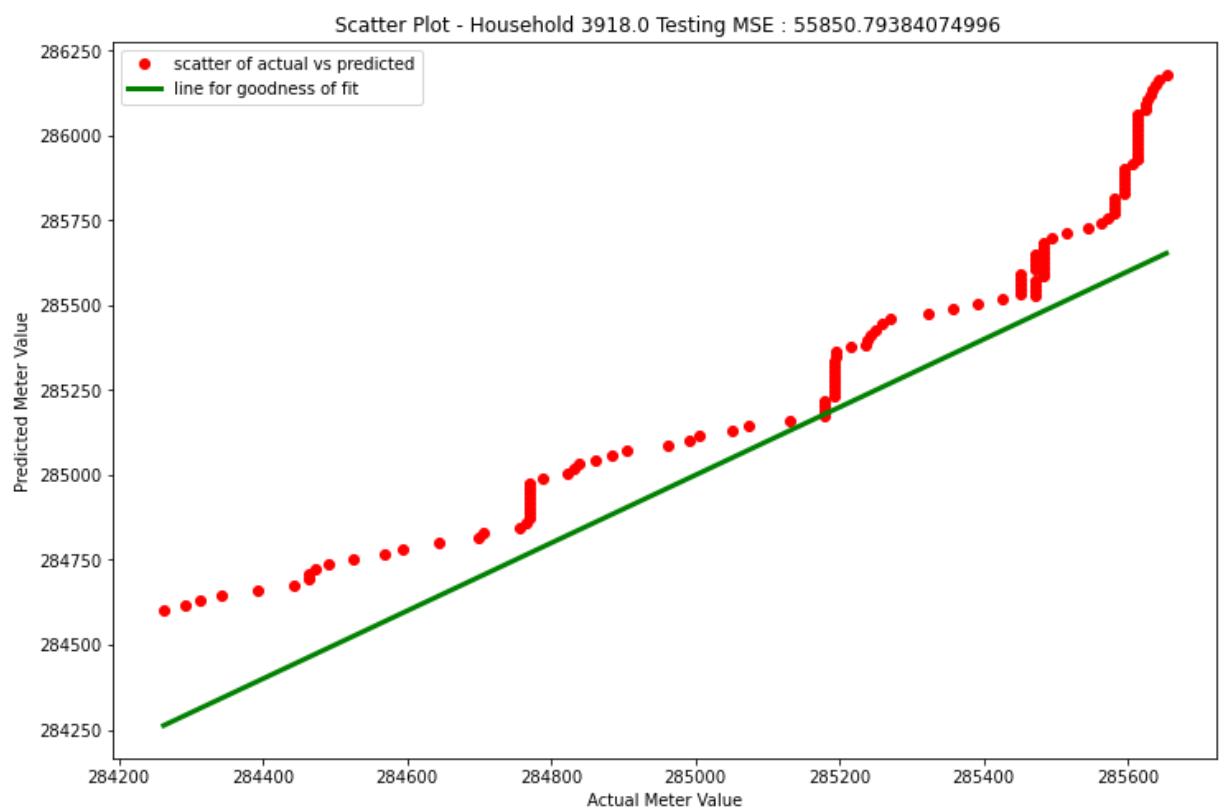
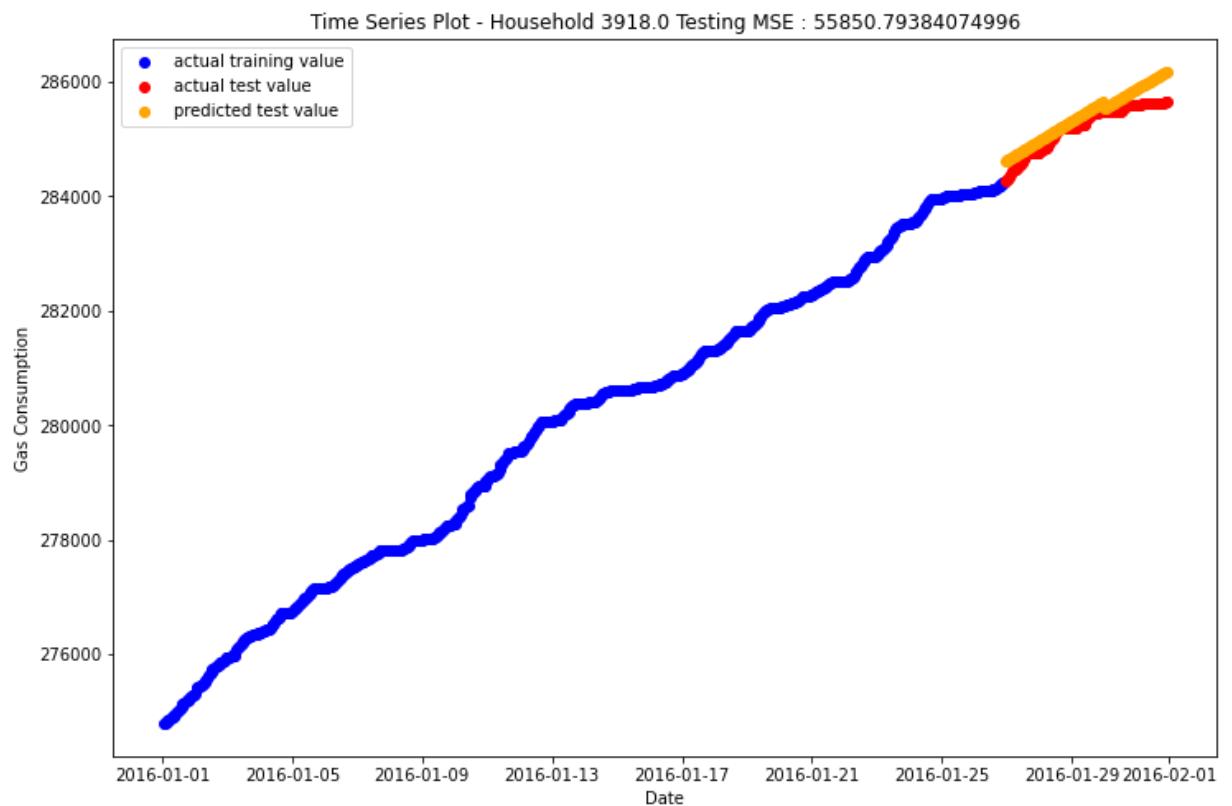
Scatter Plot - Household 3723.0 Testing MSE : 5309.990352939731

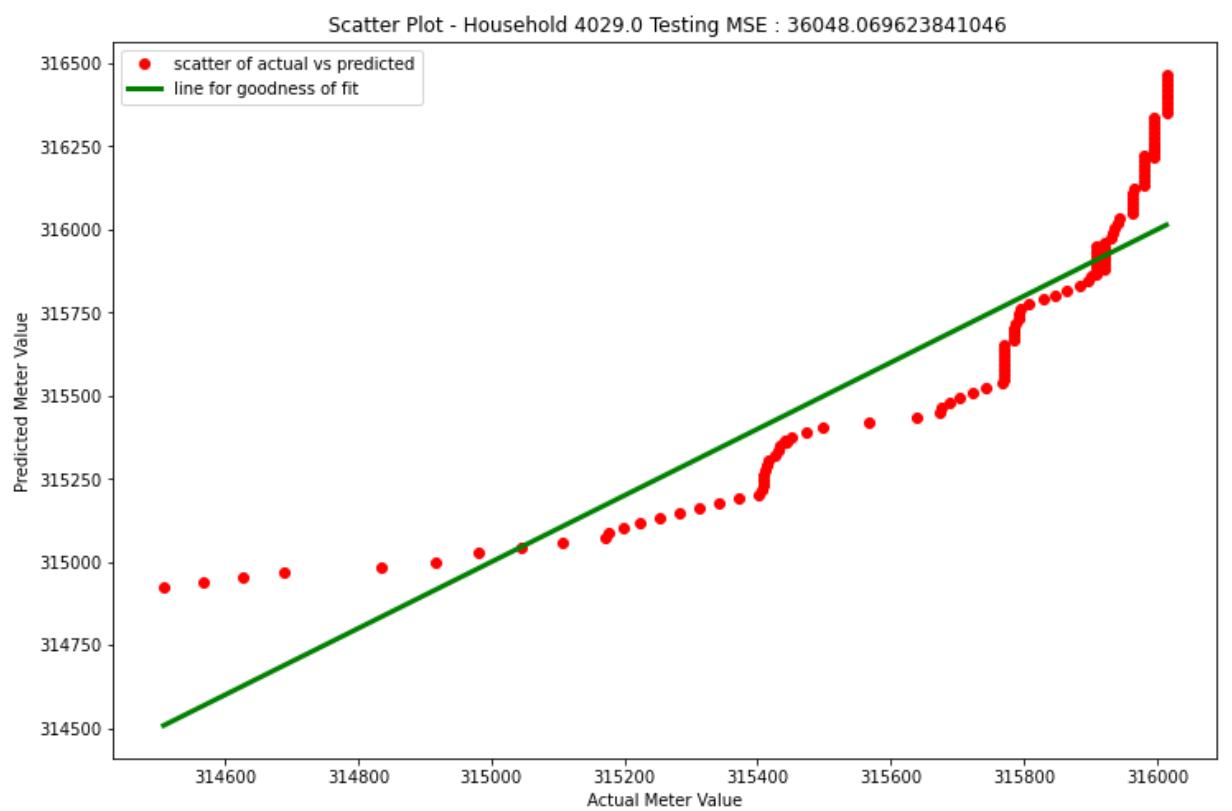
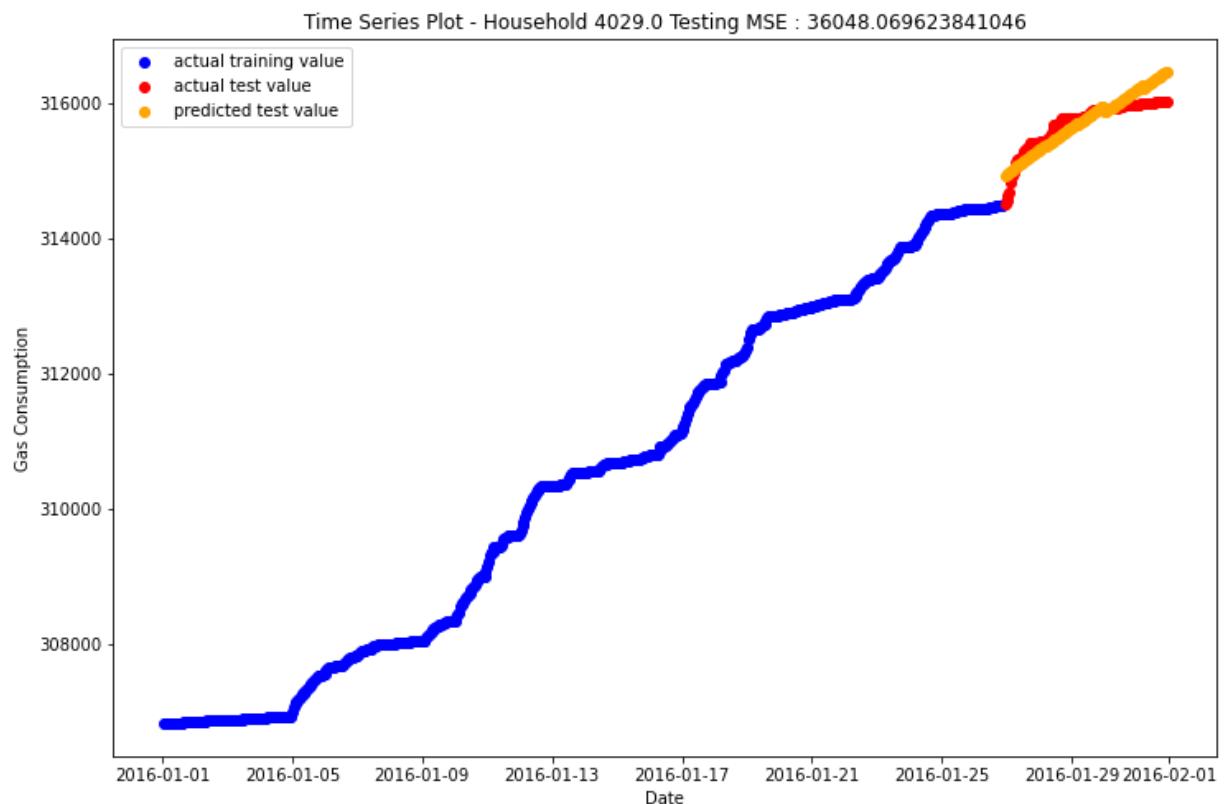


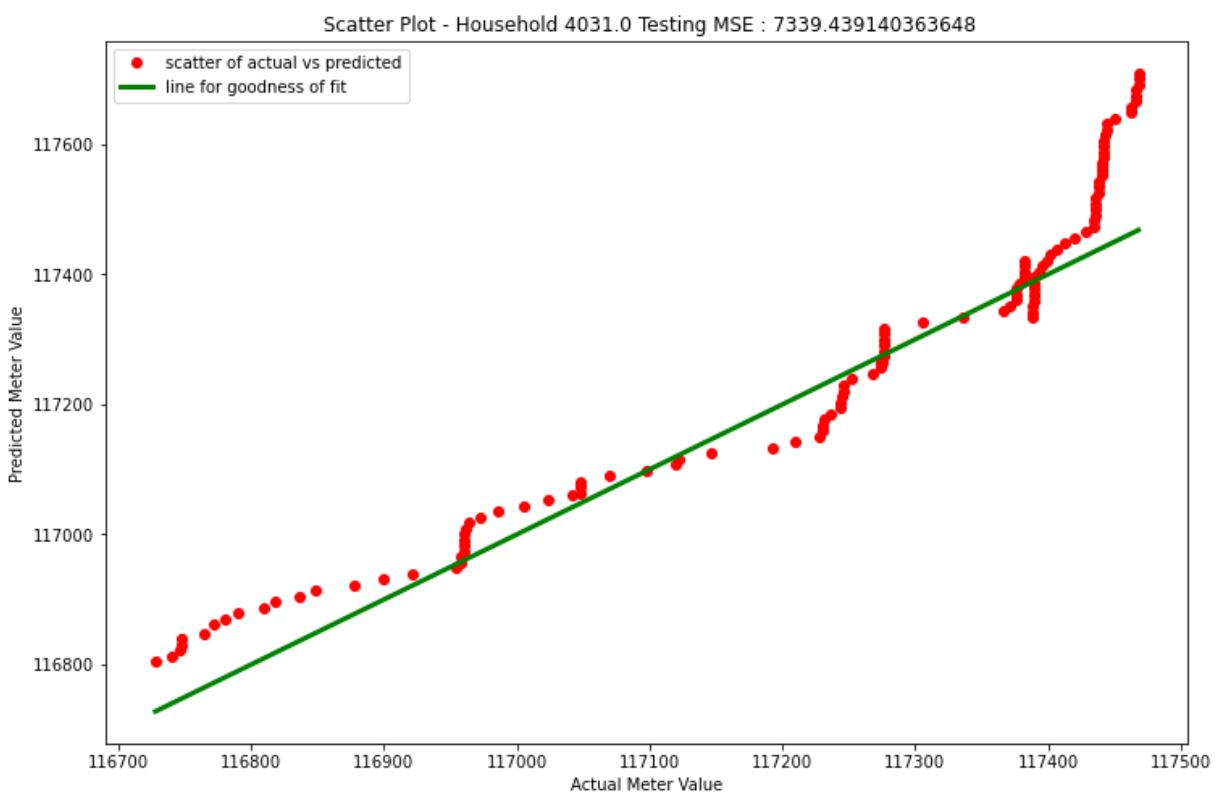
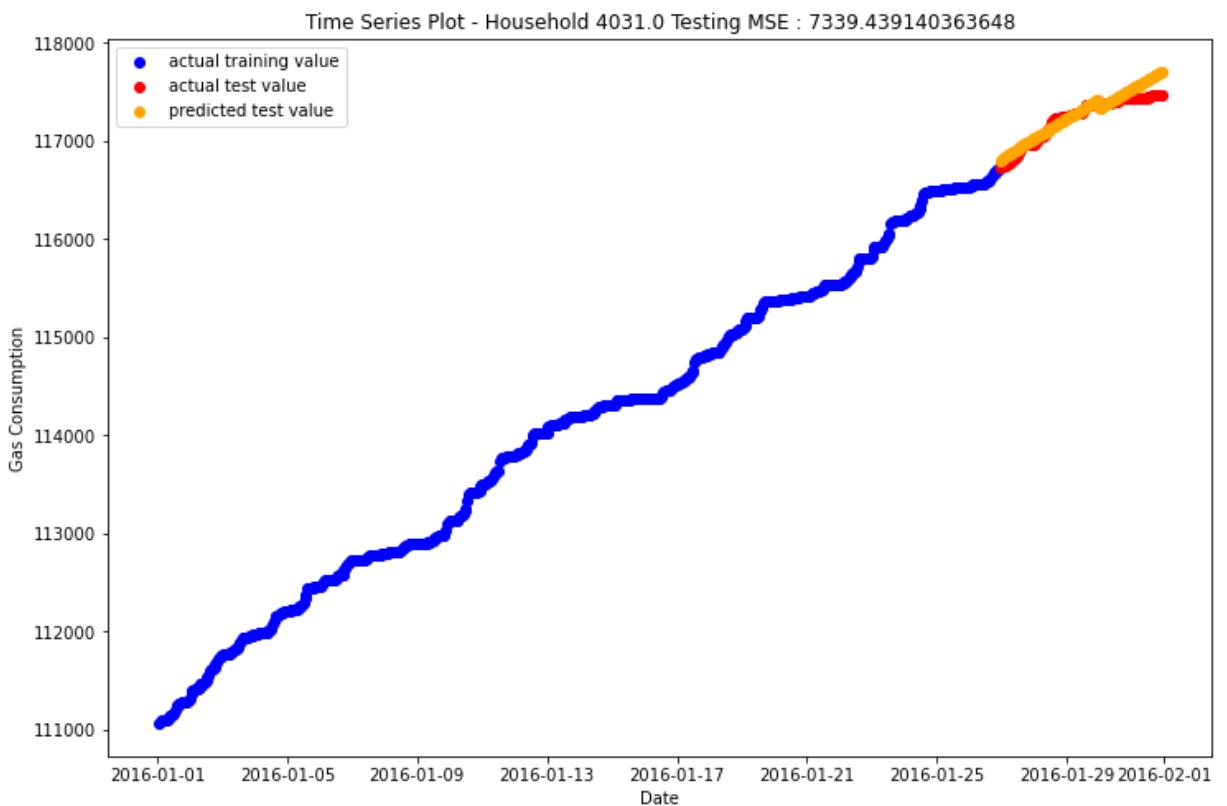


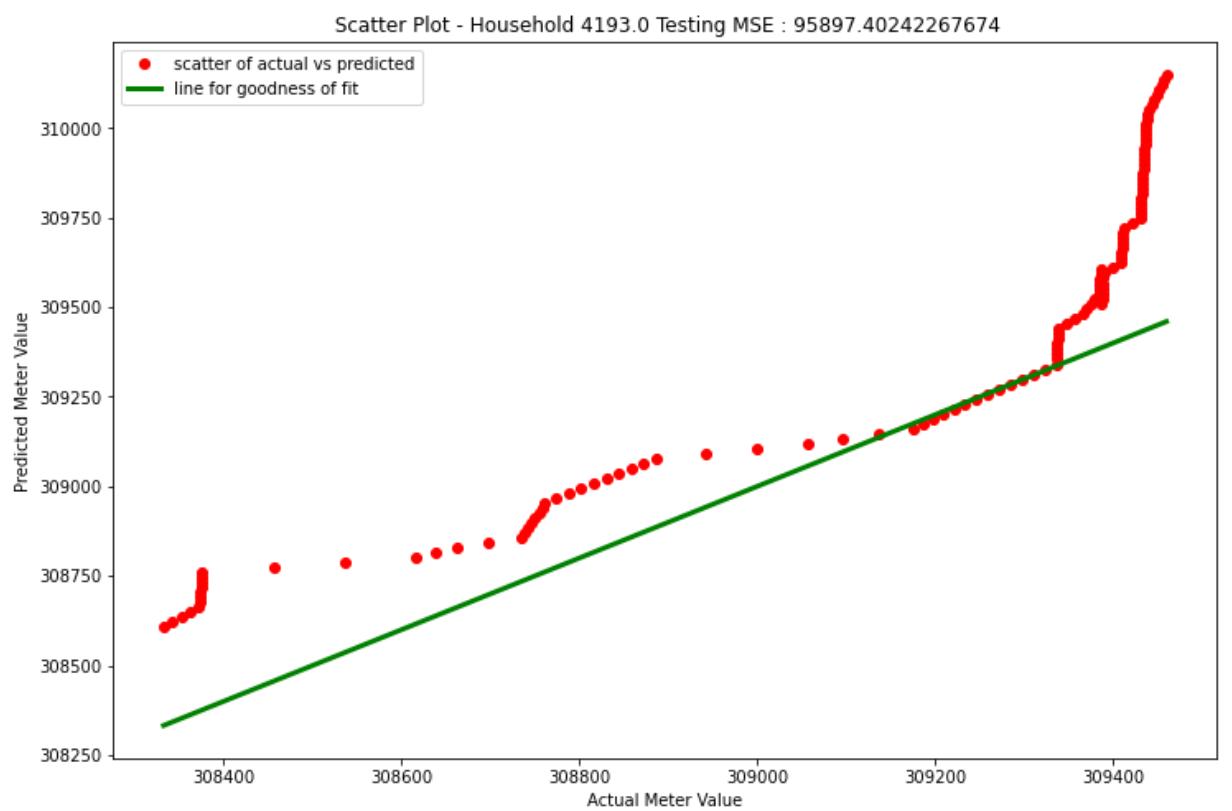
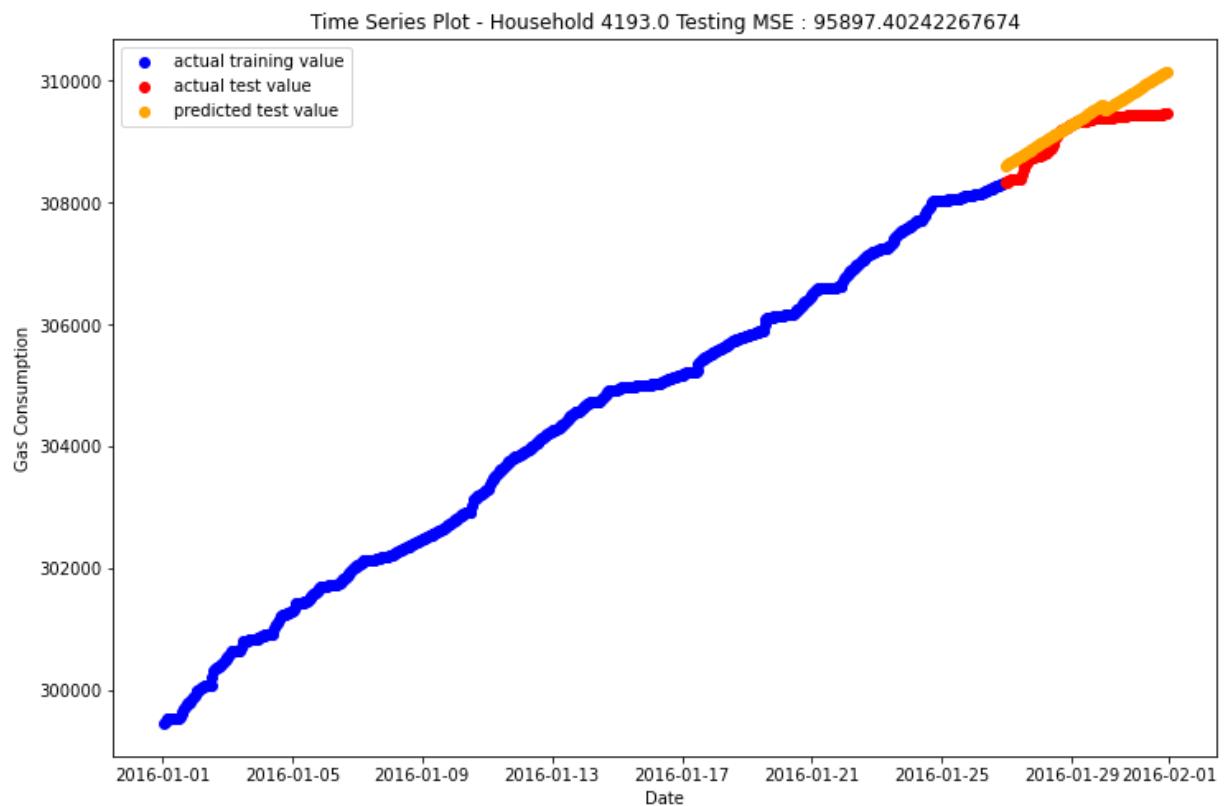


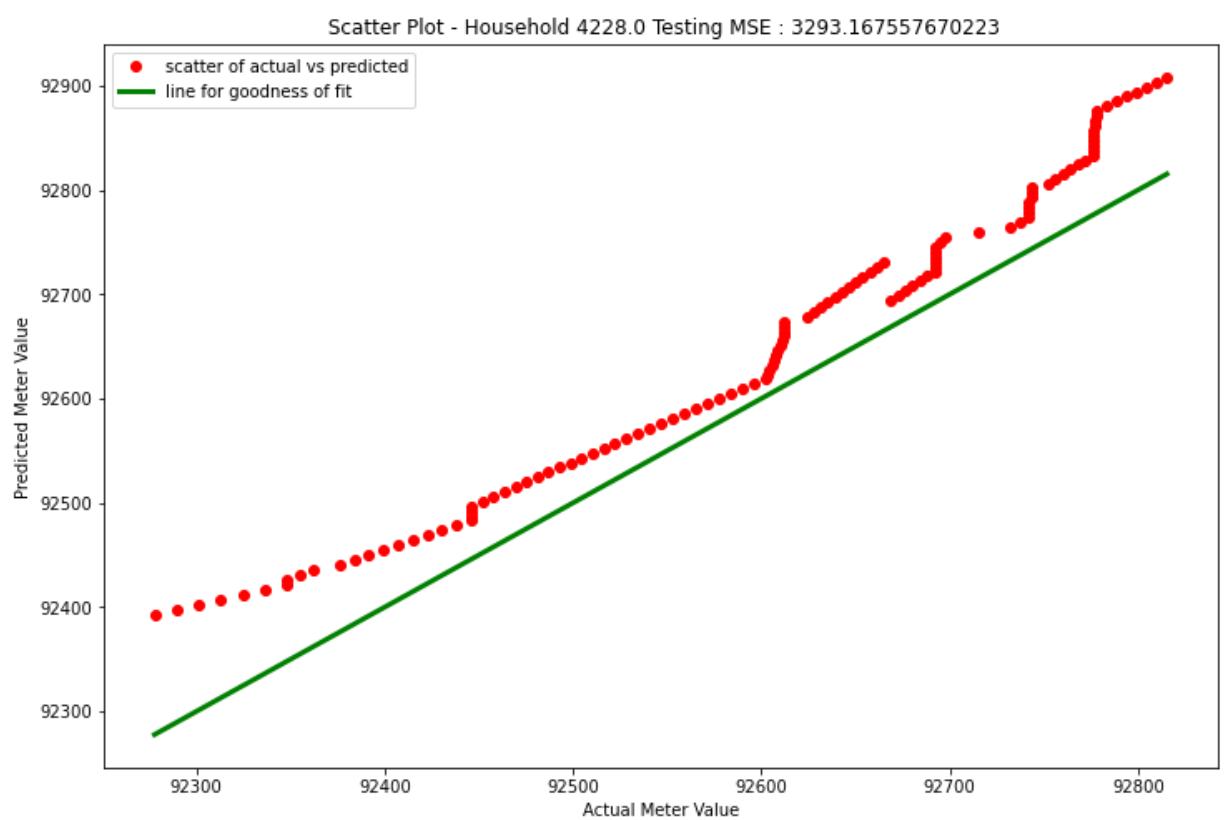
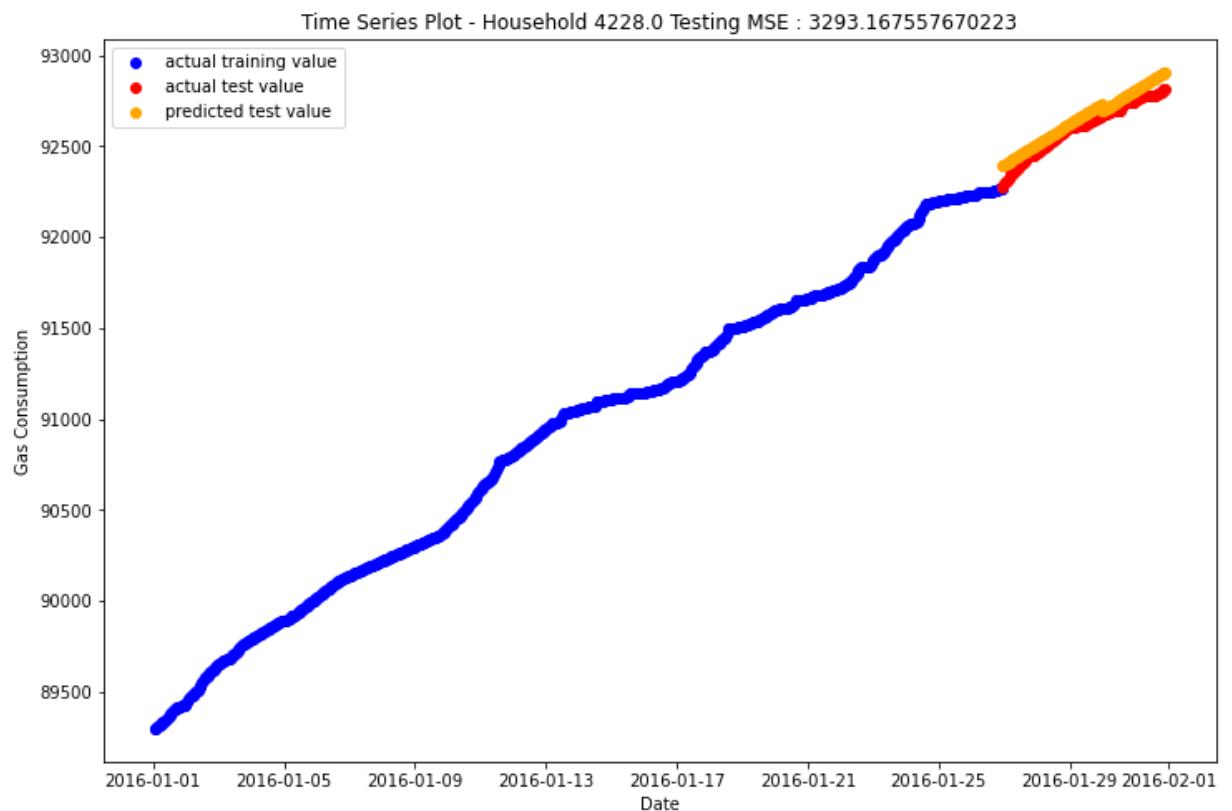


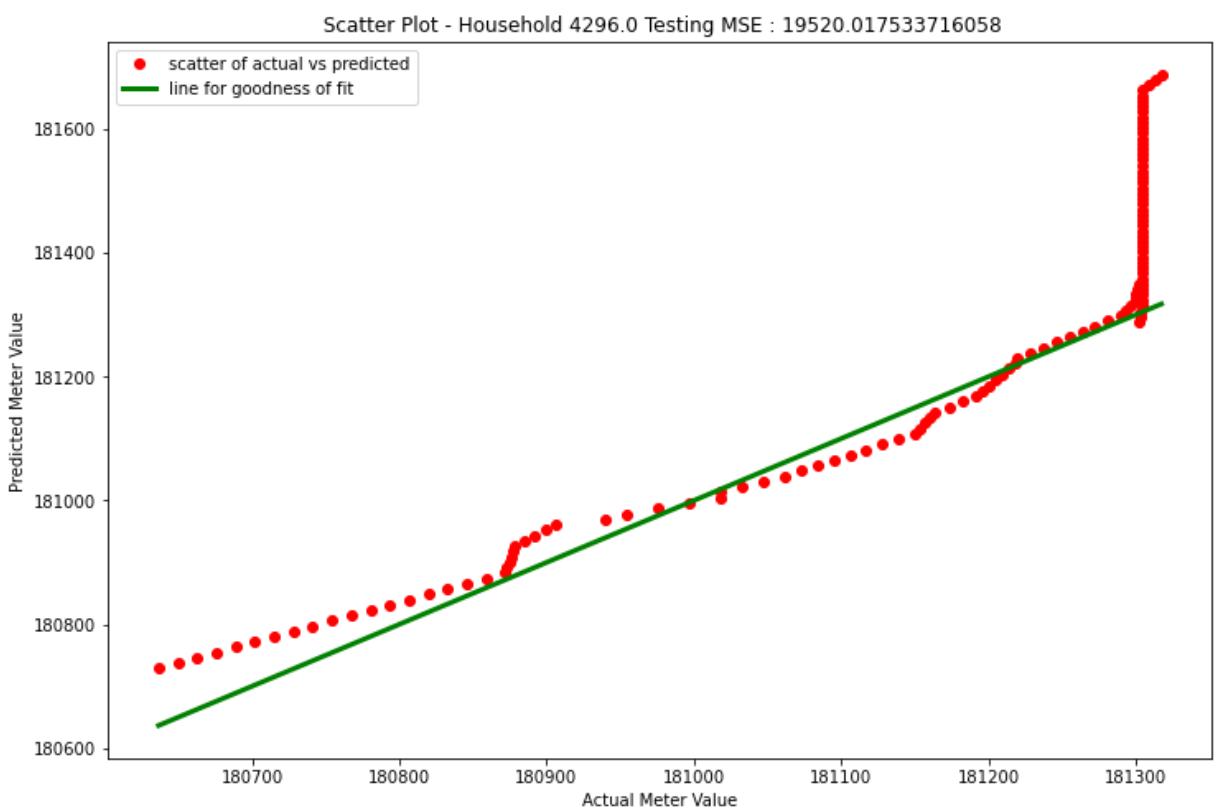
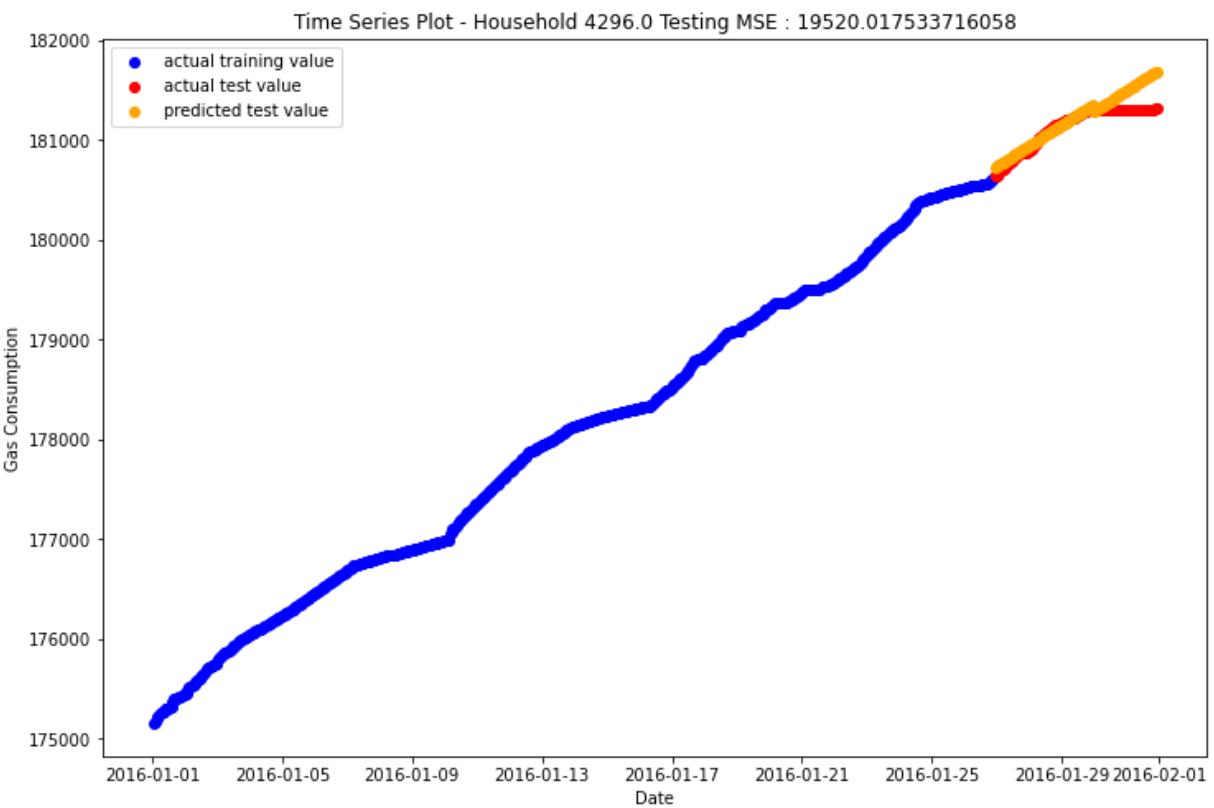


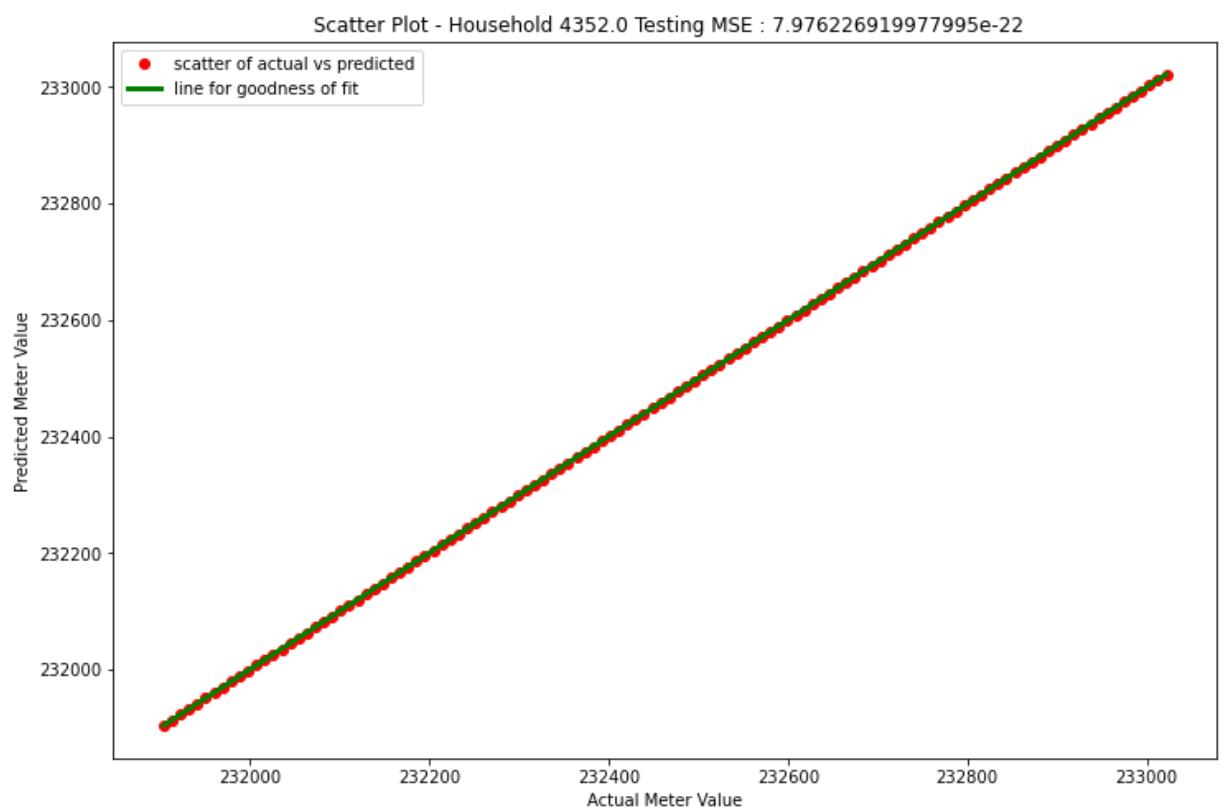
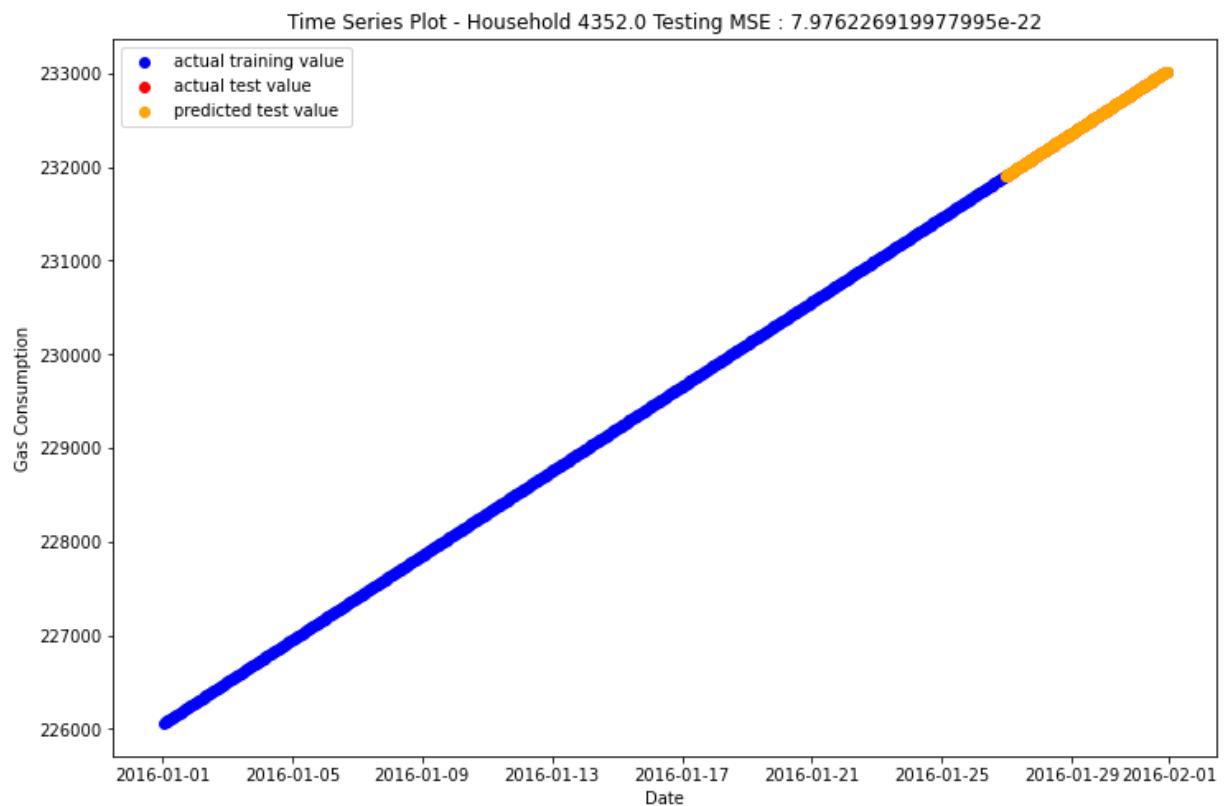


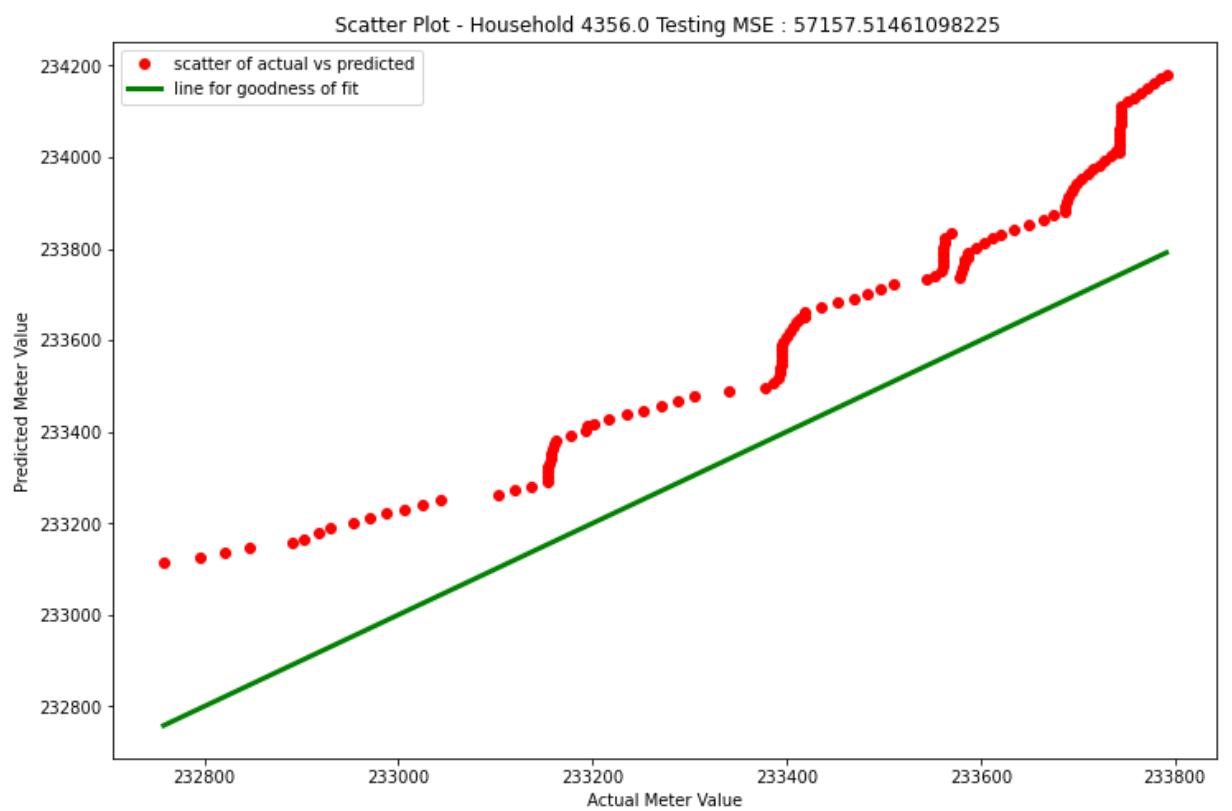
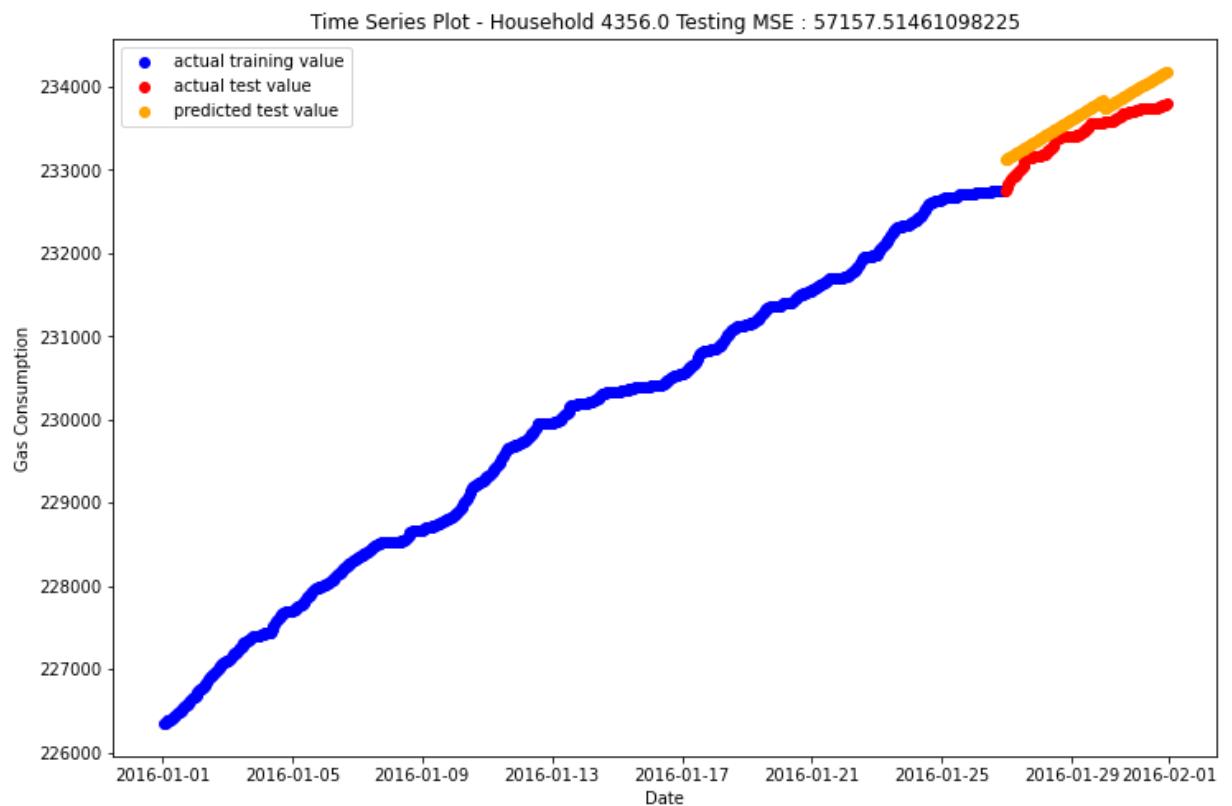


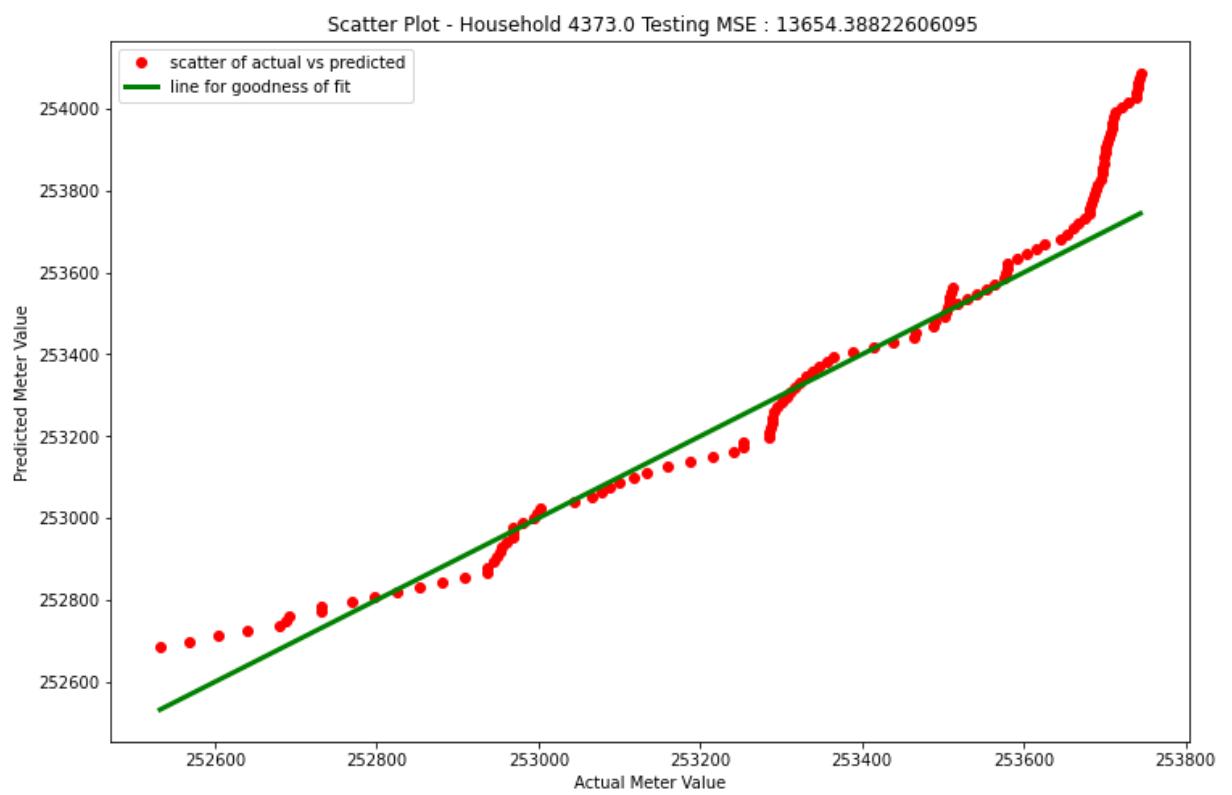
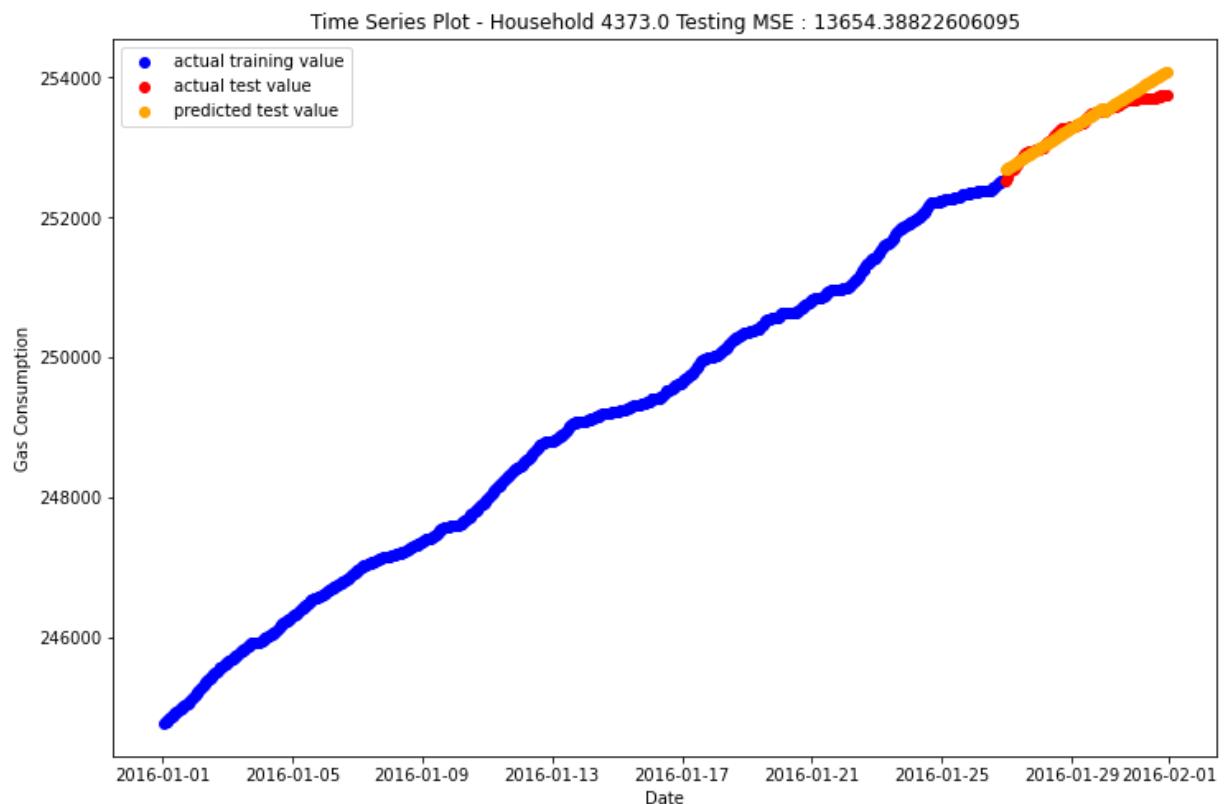


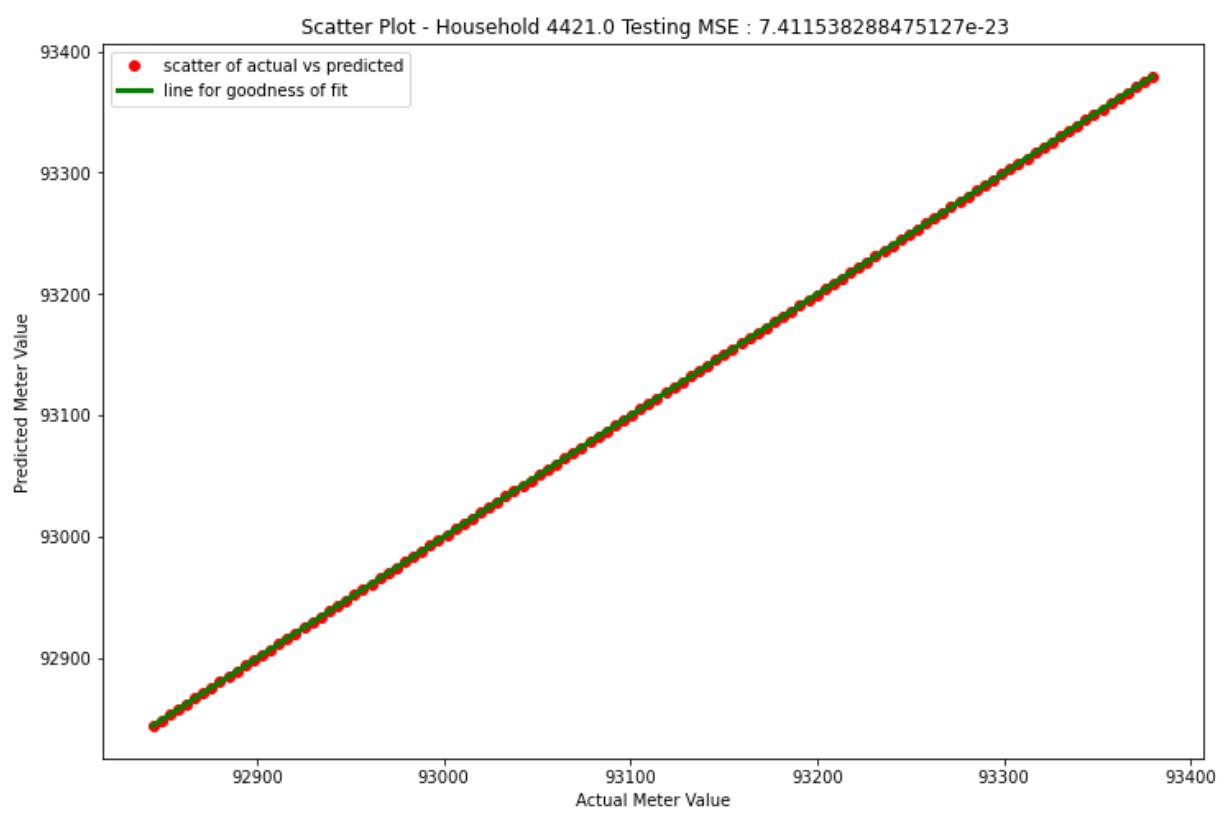
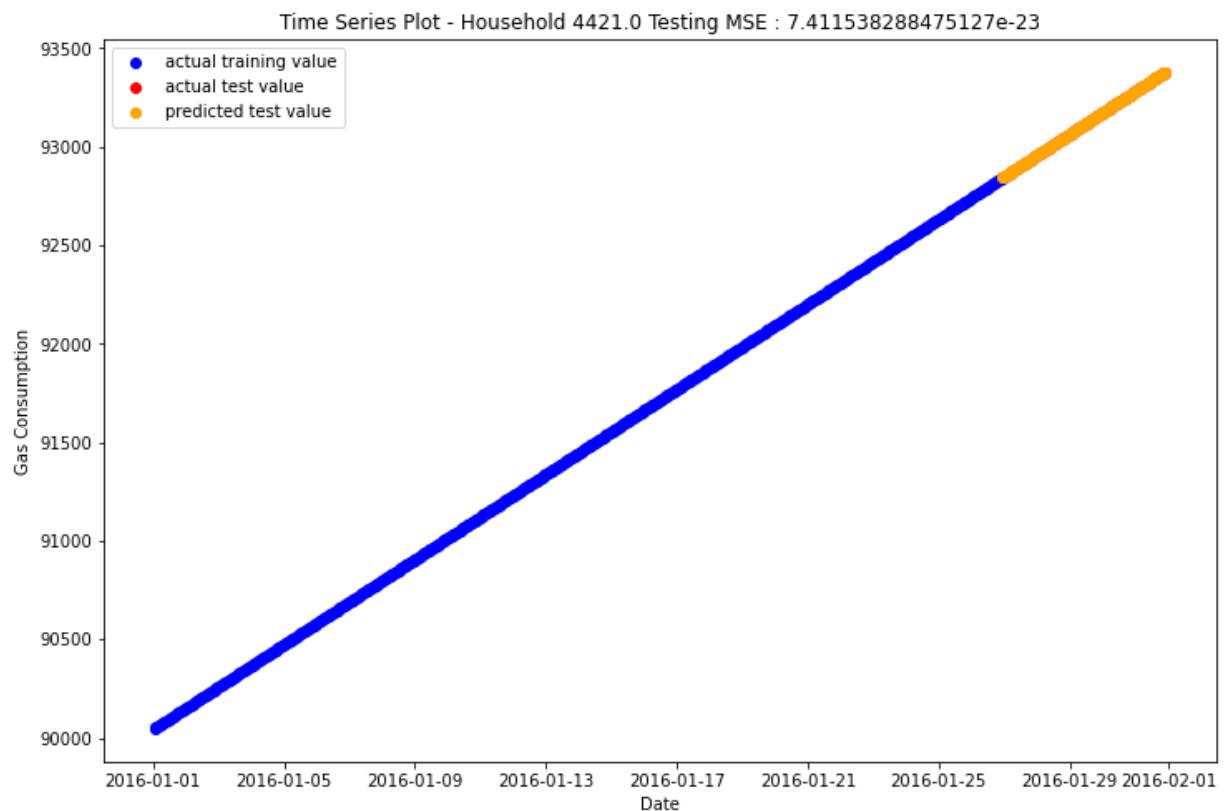


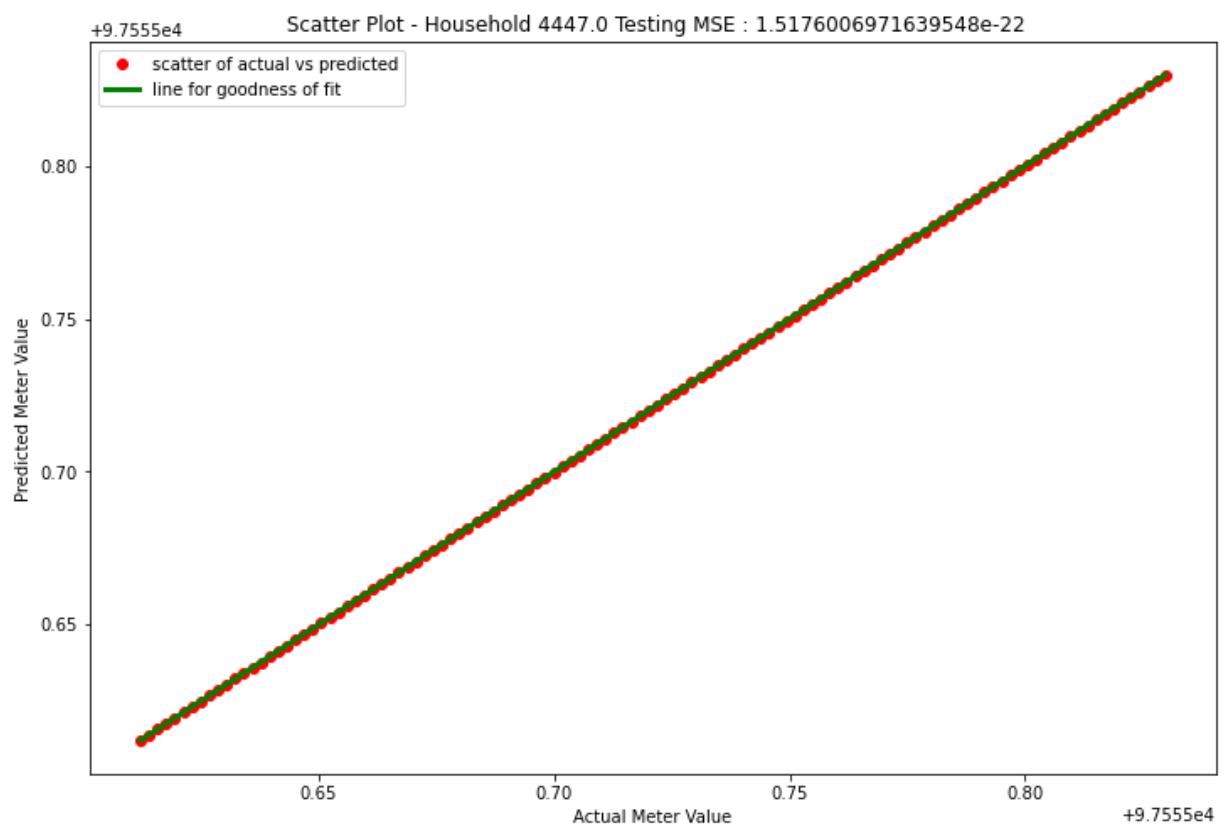
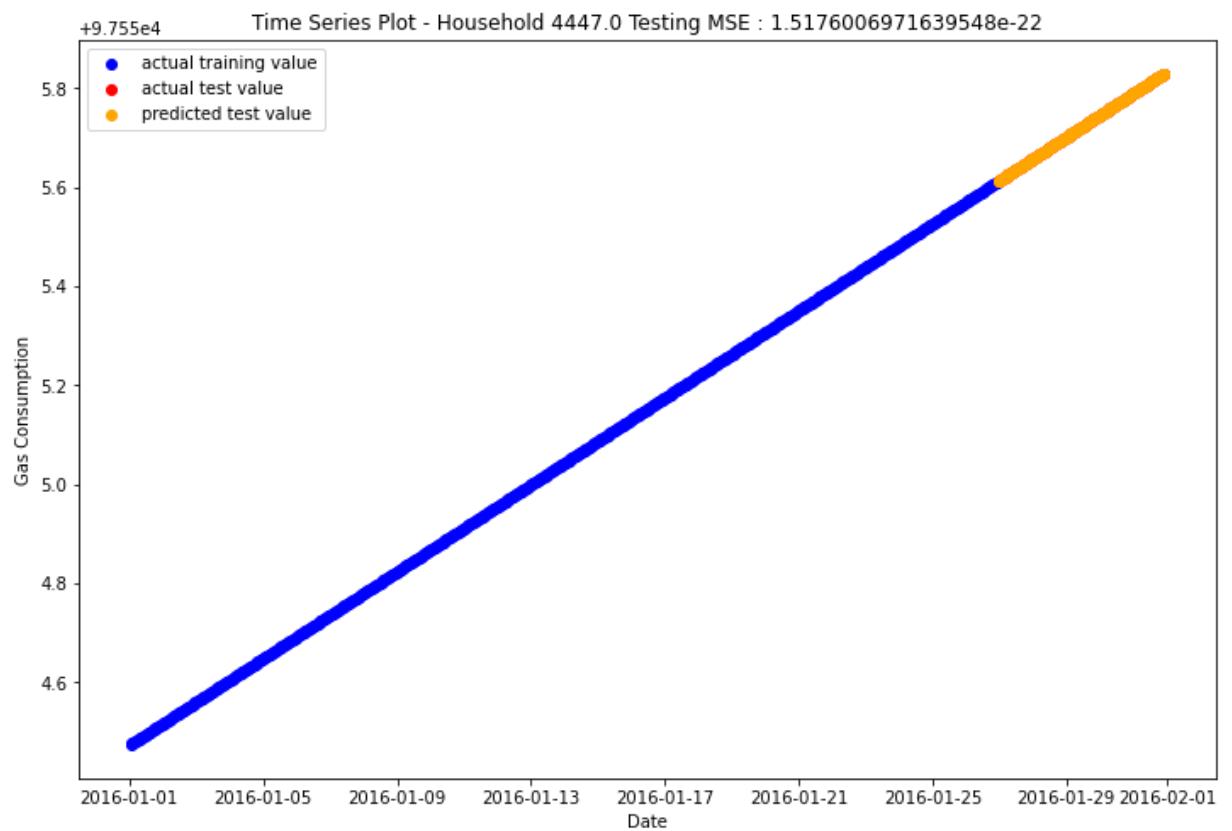


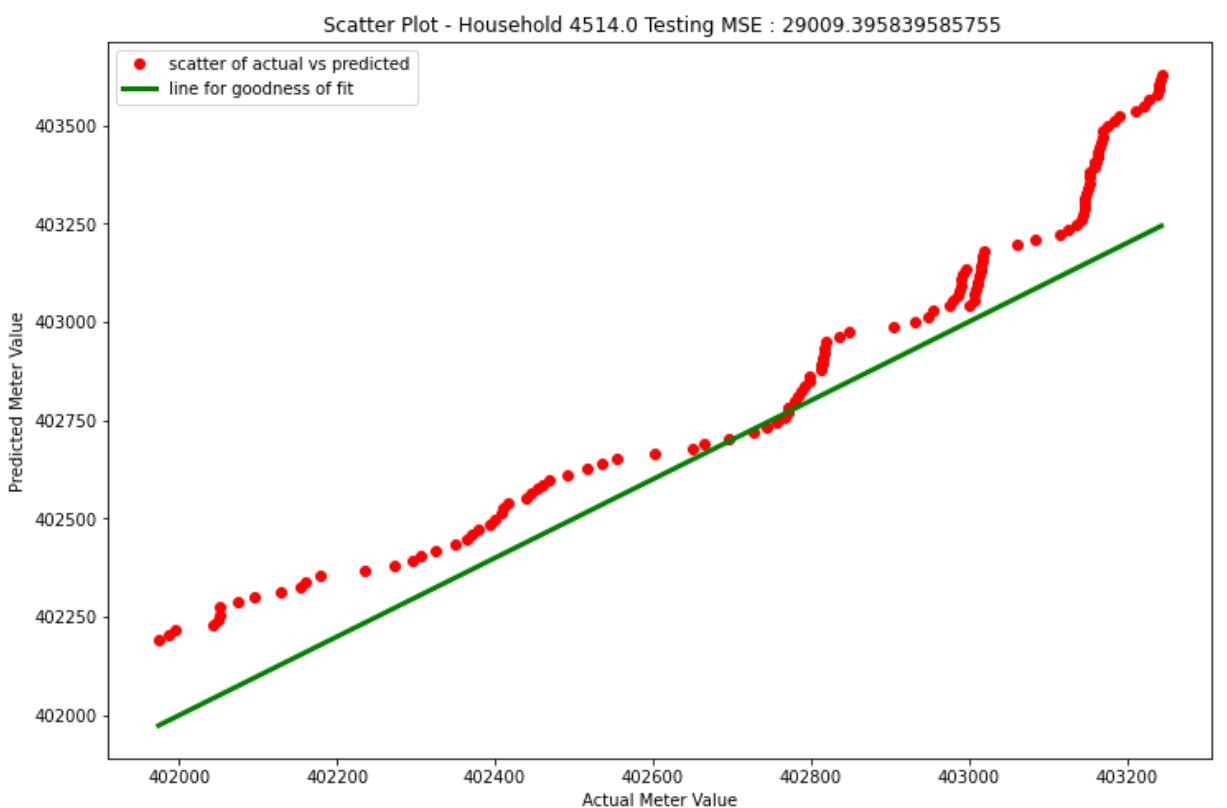
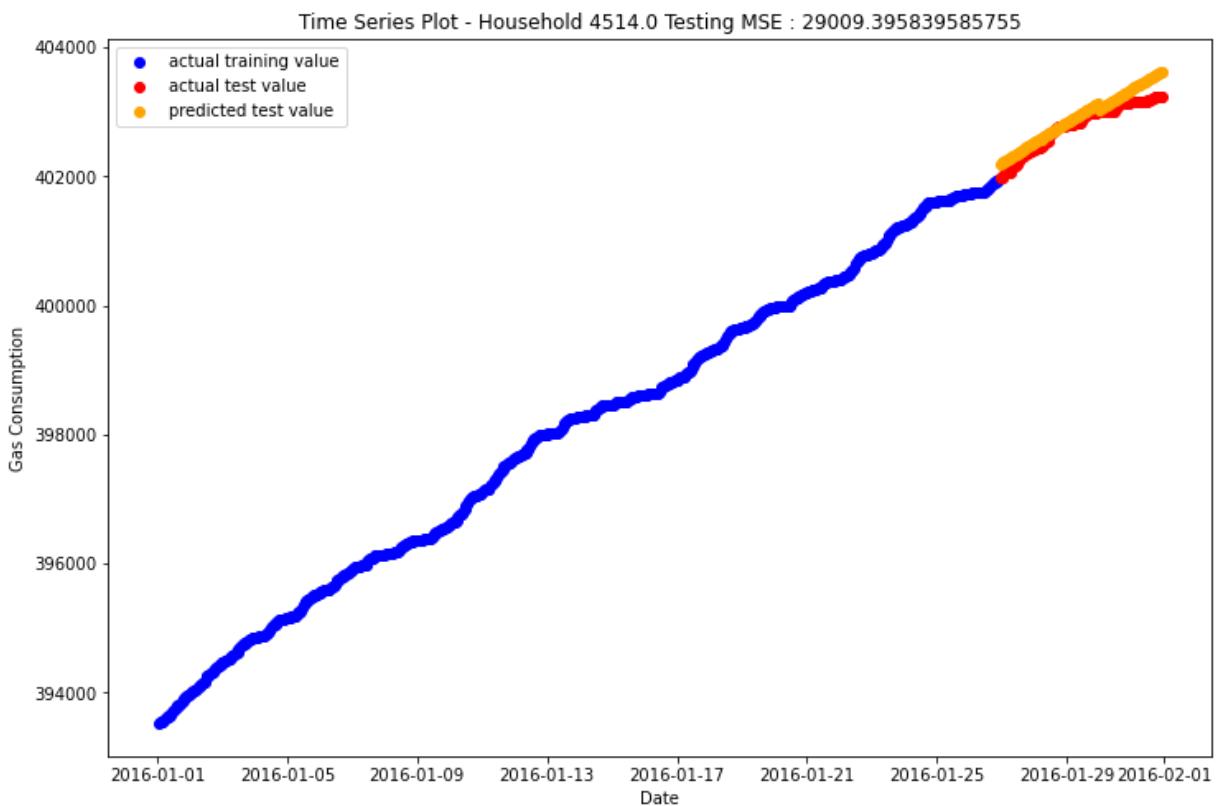


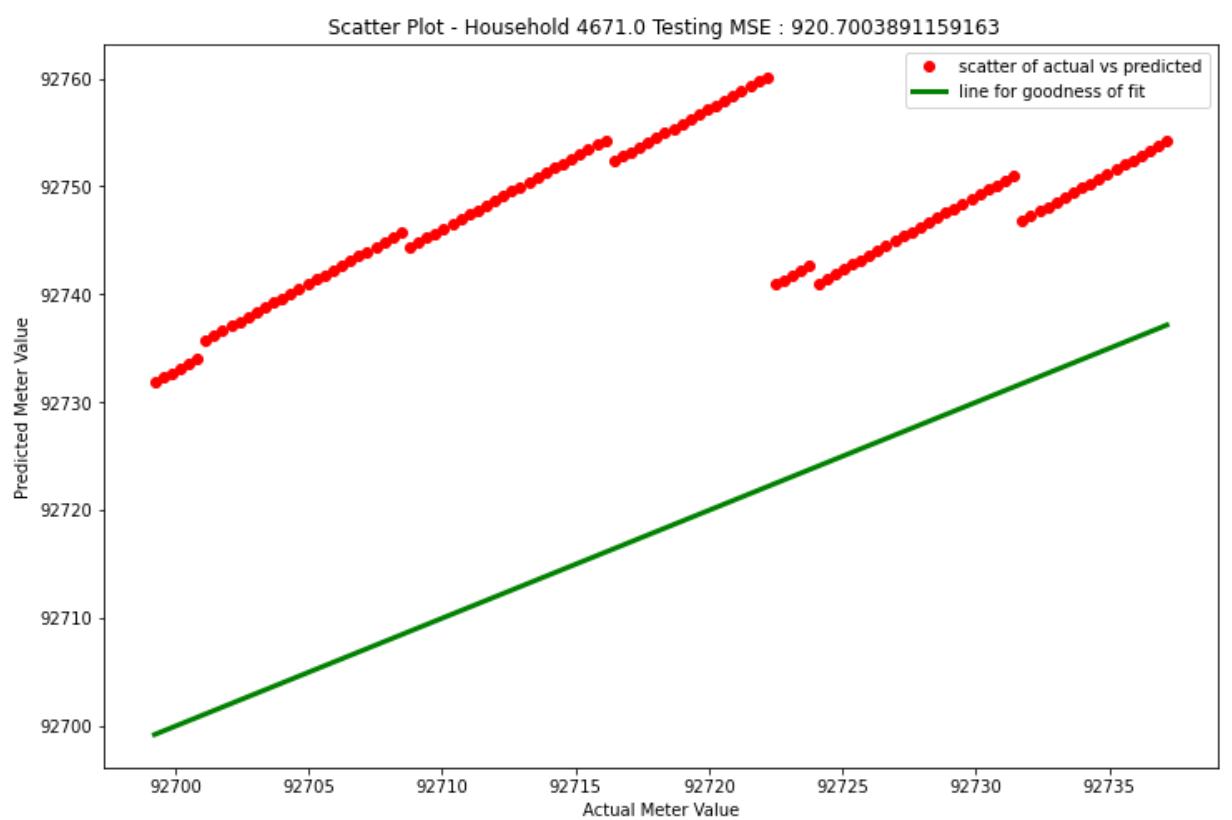
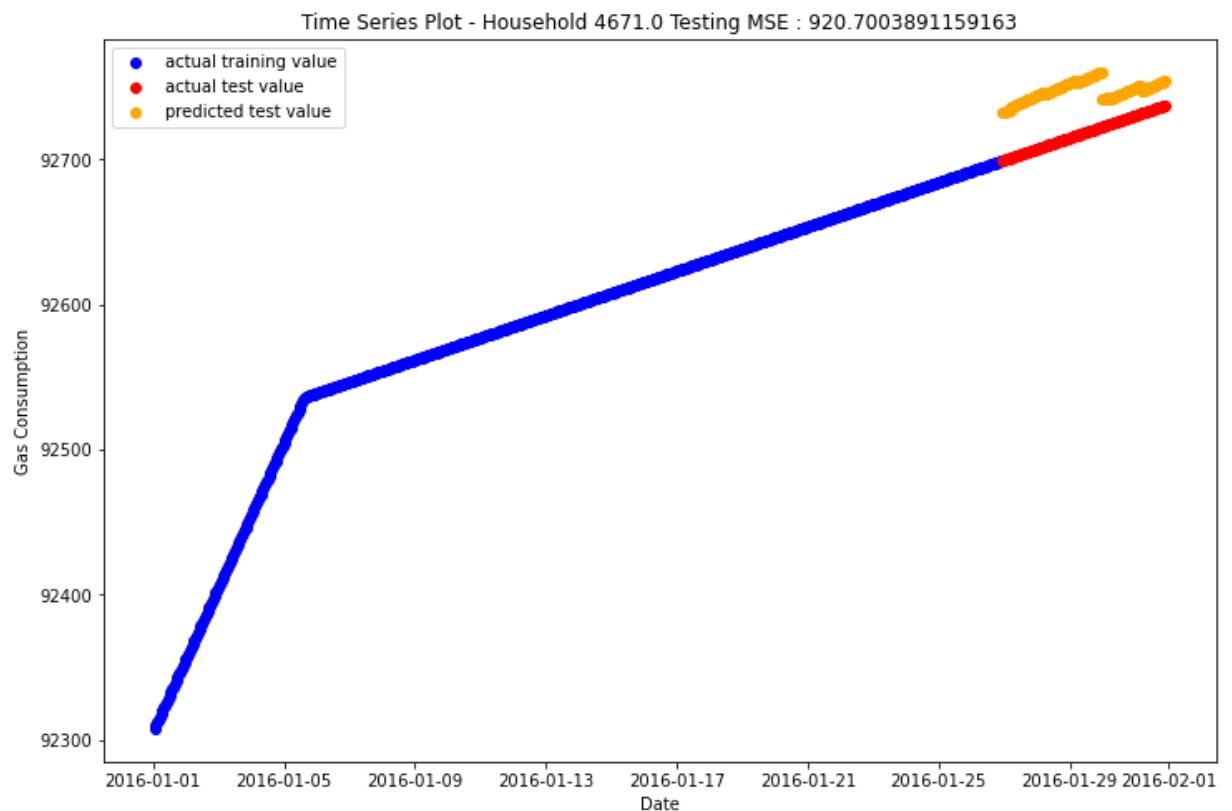


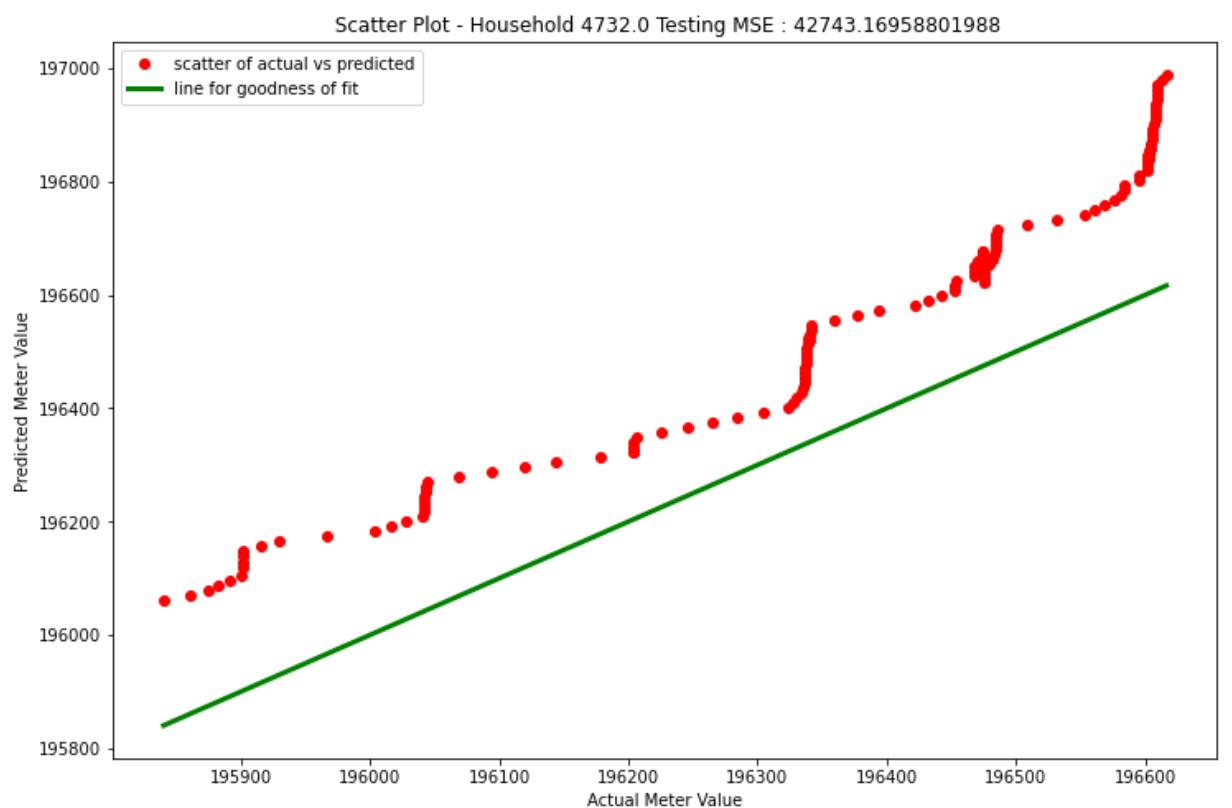
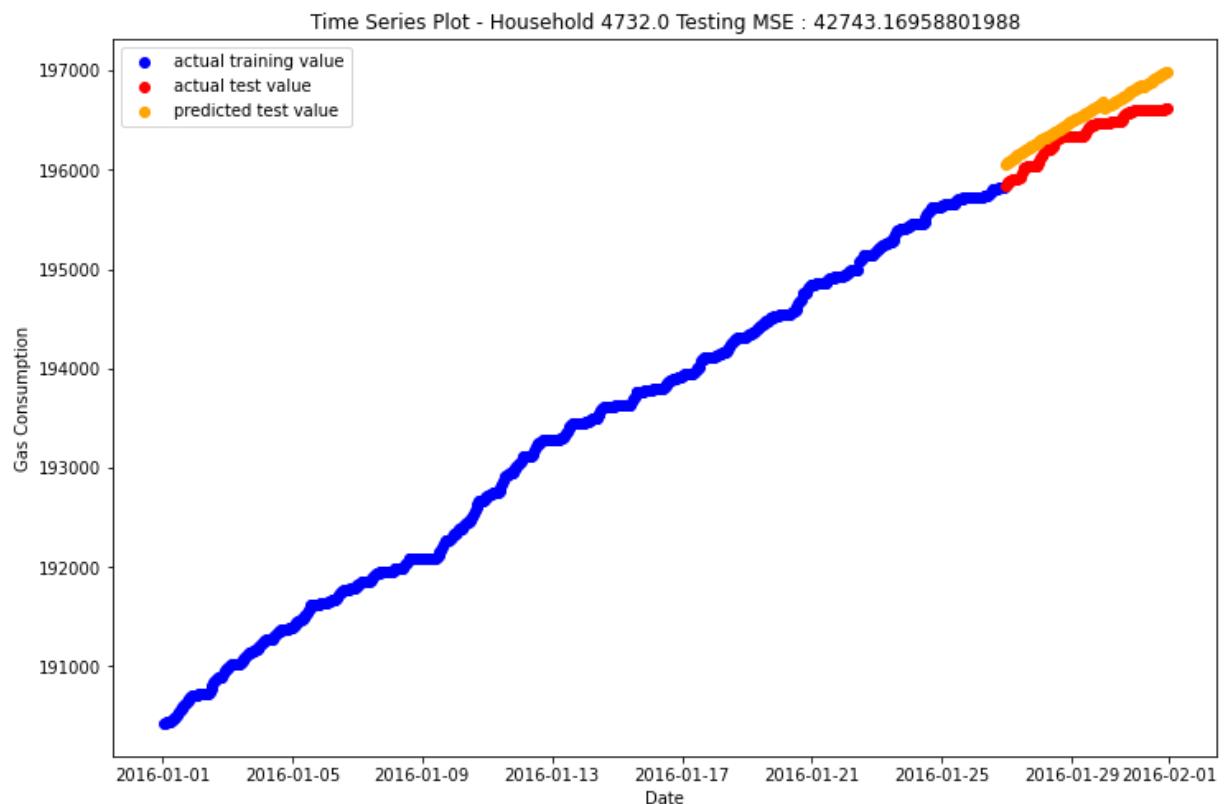


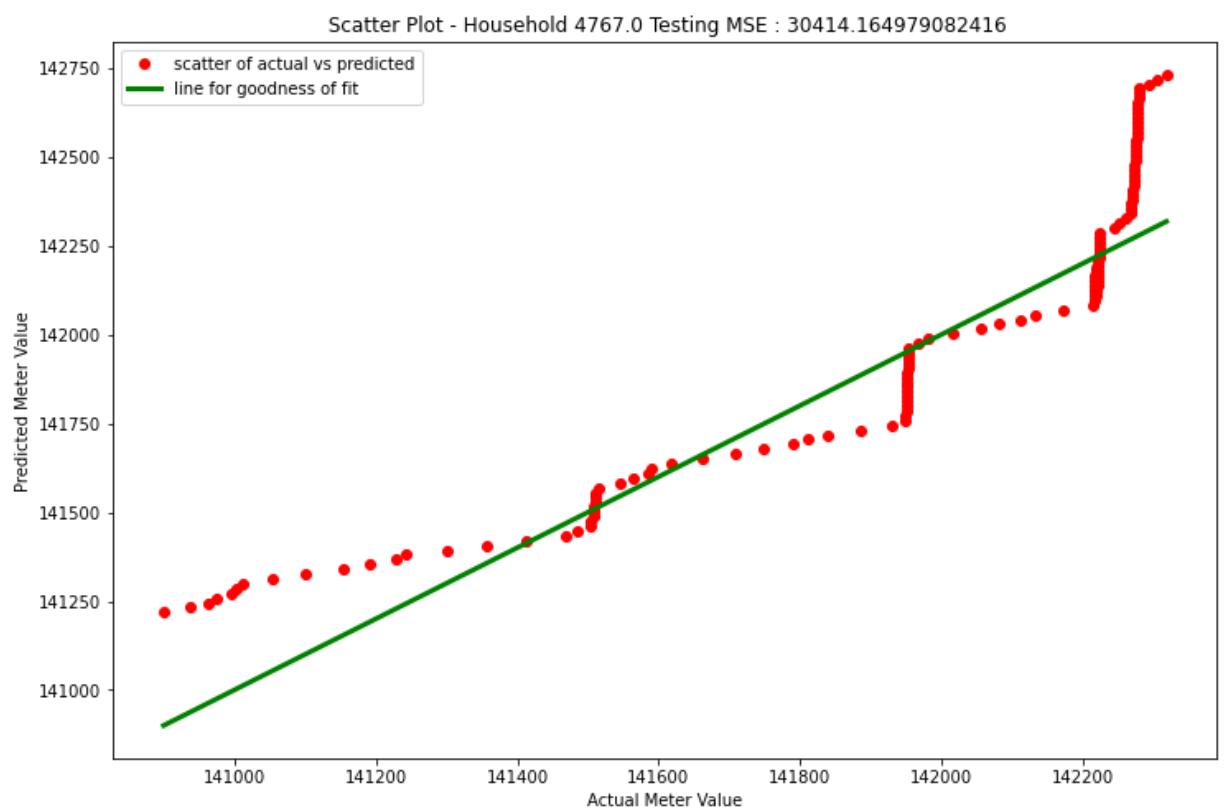
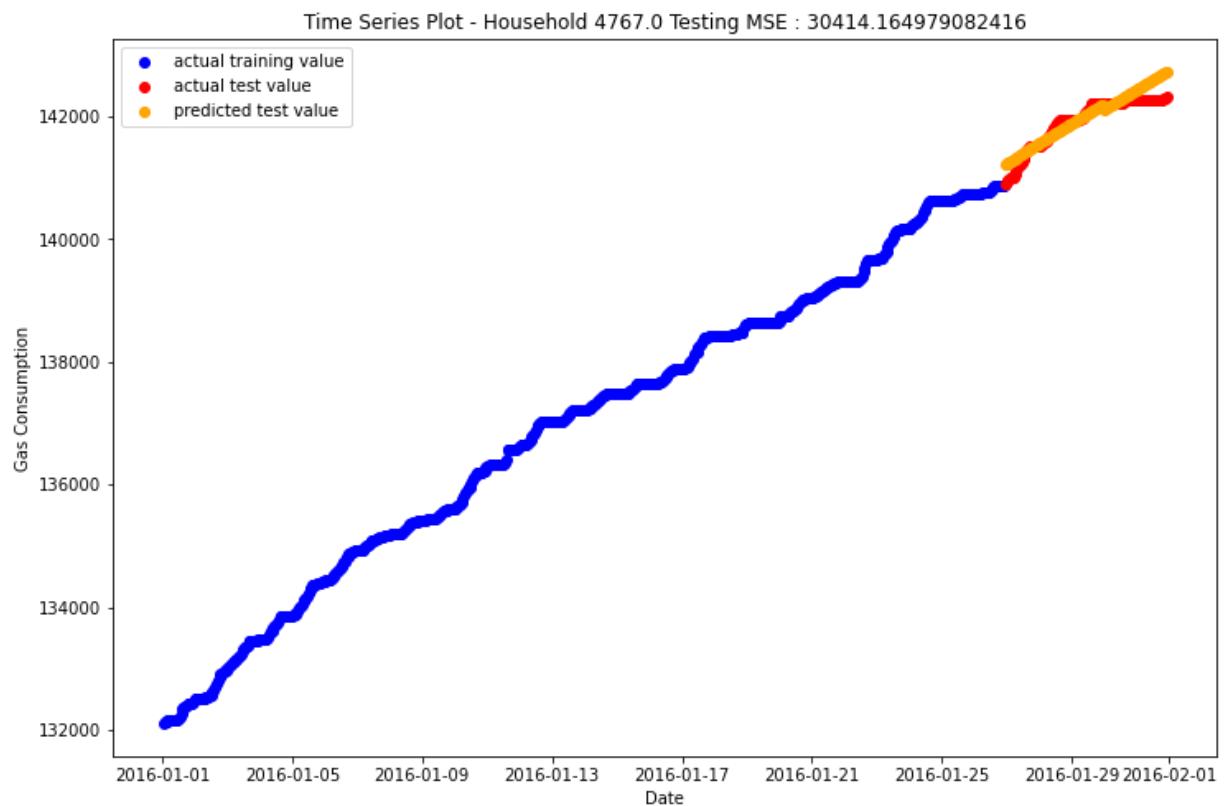


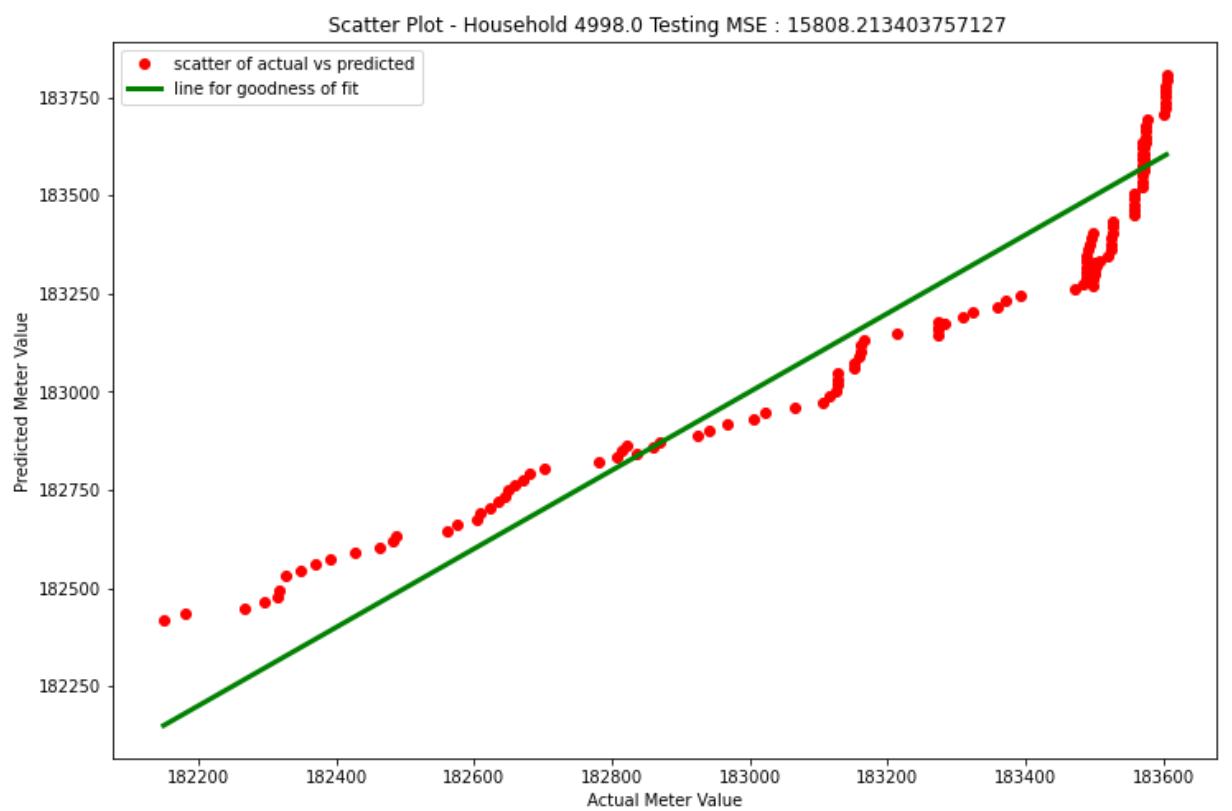
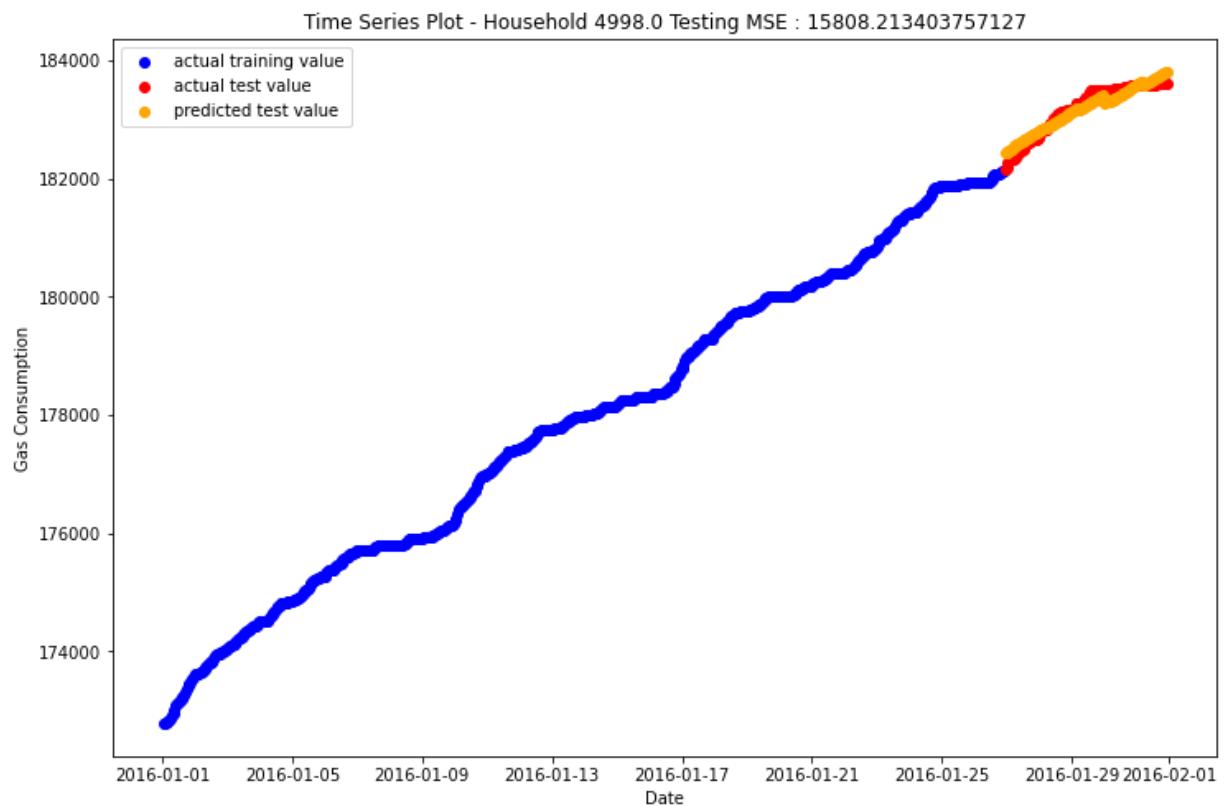


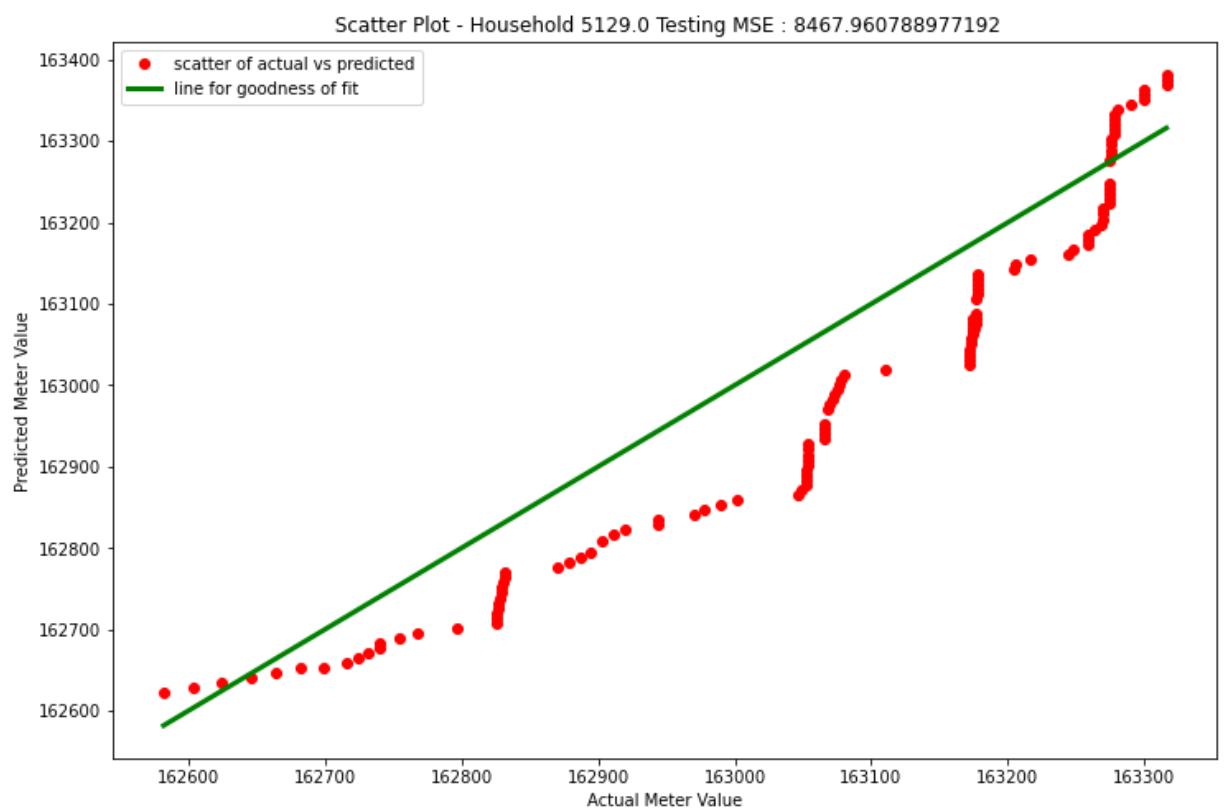
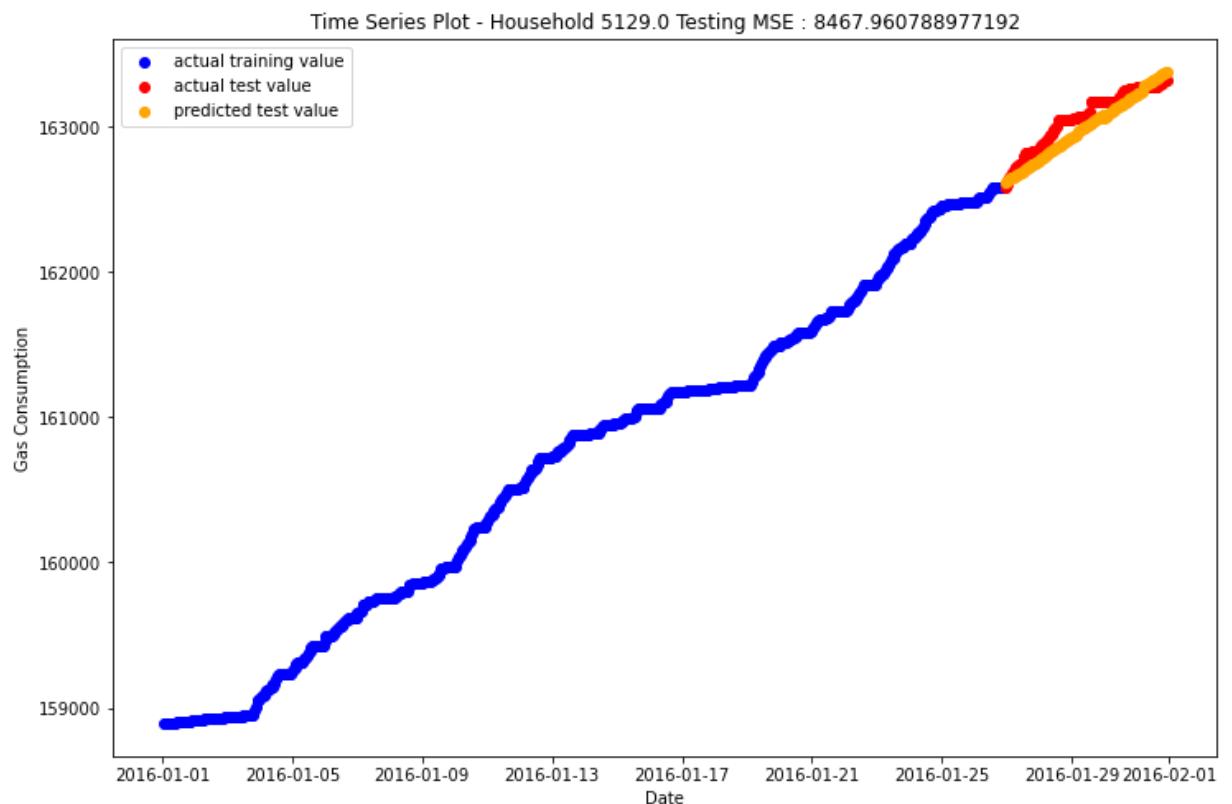


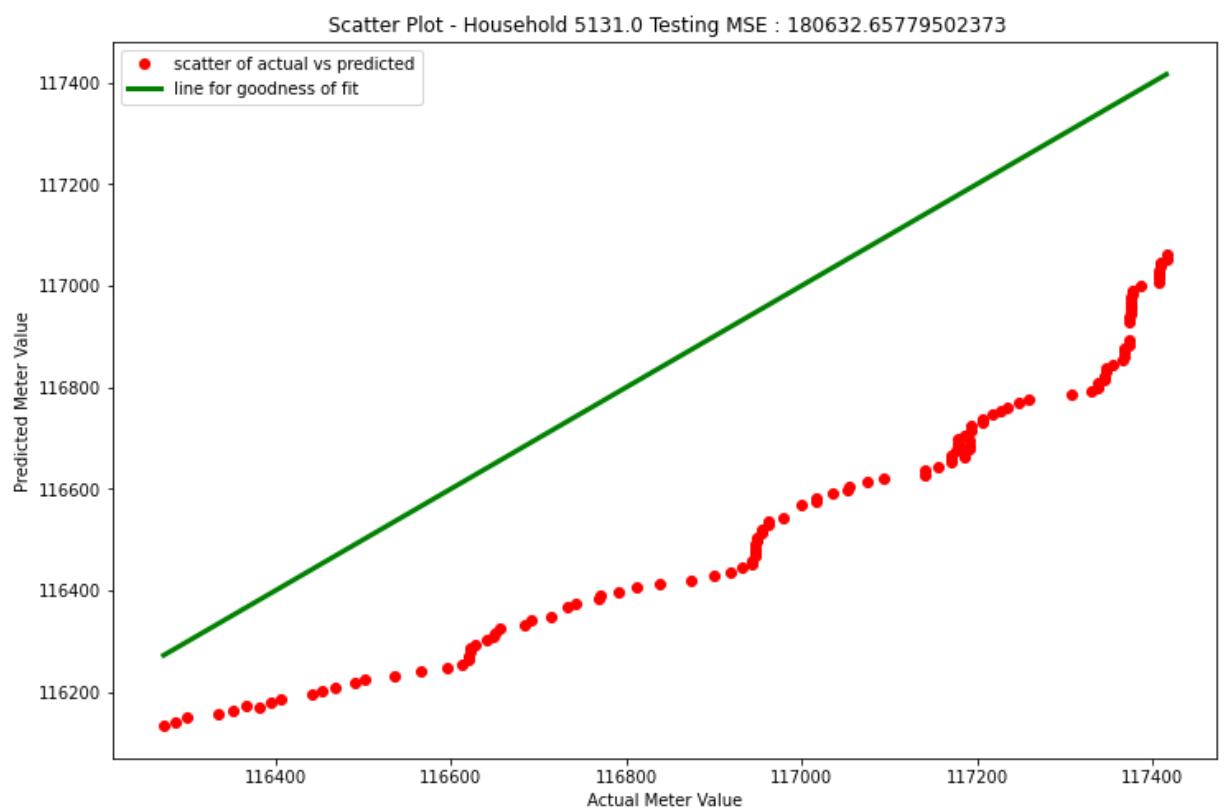
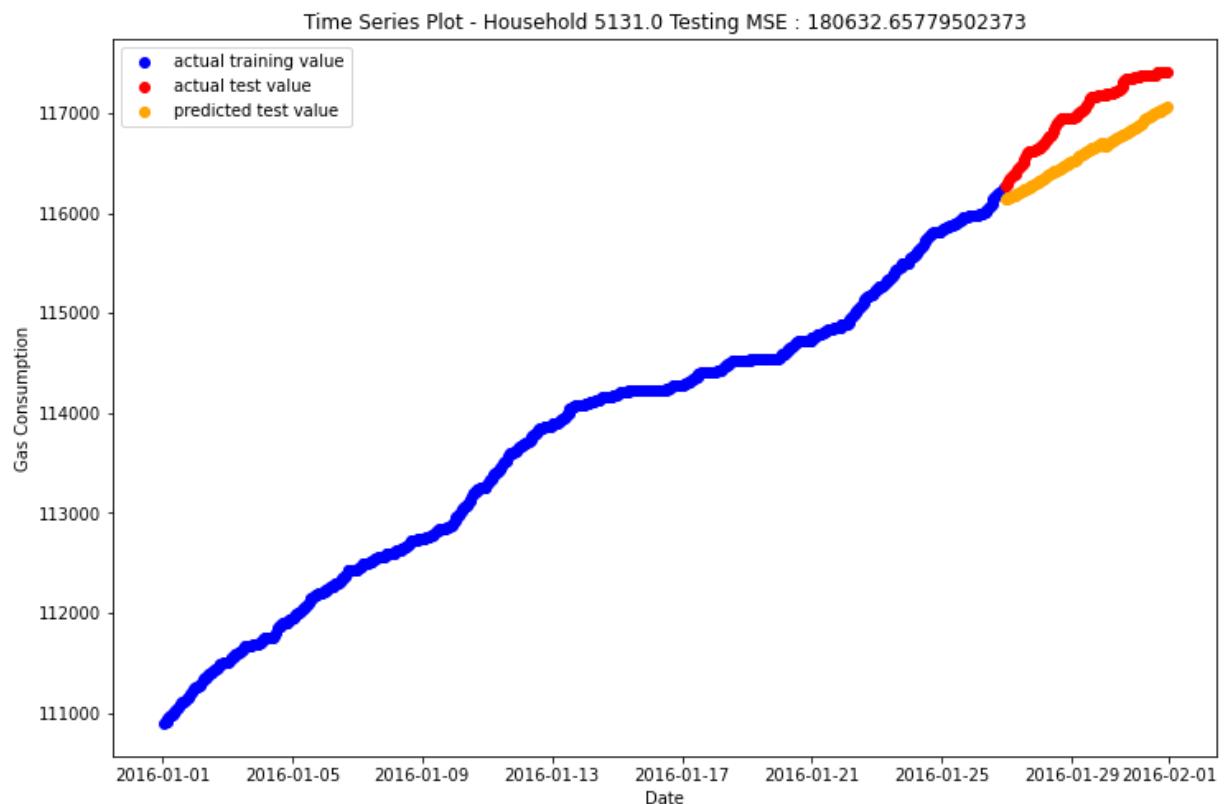


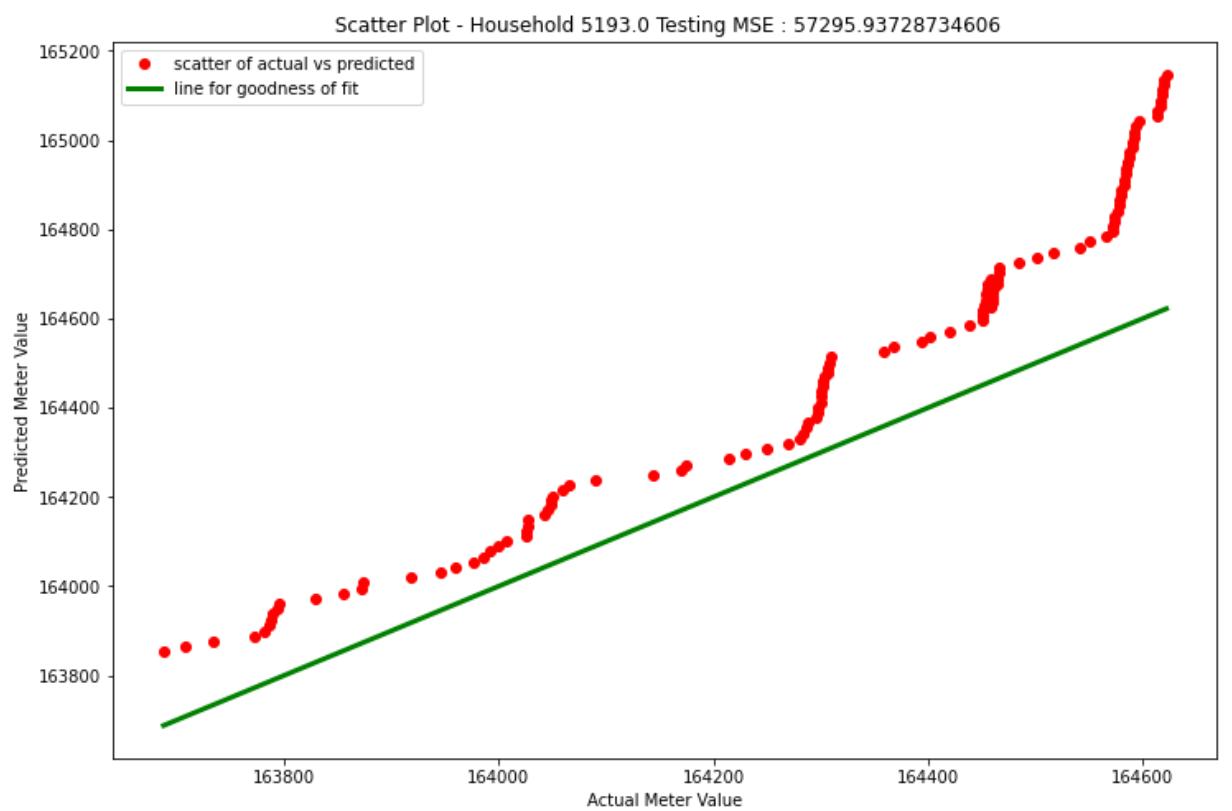
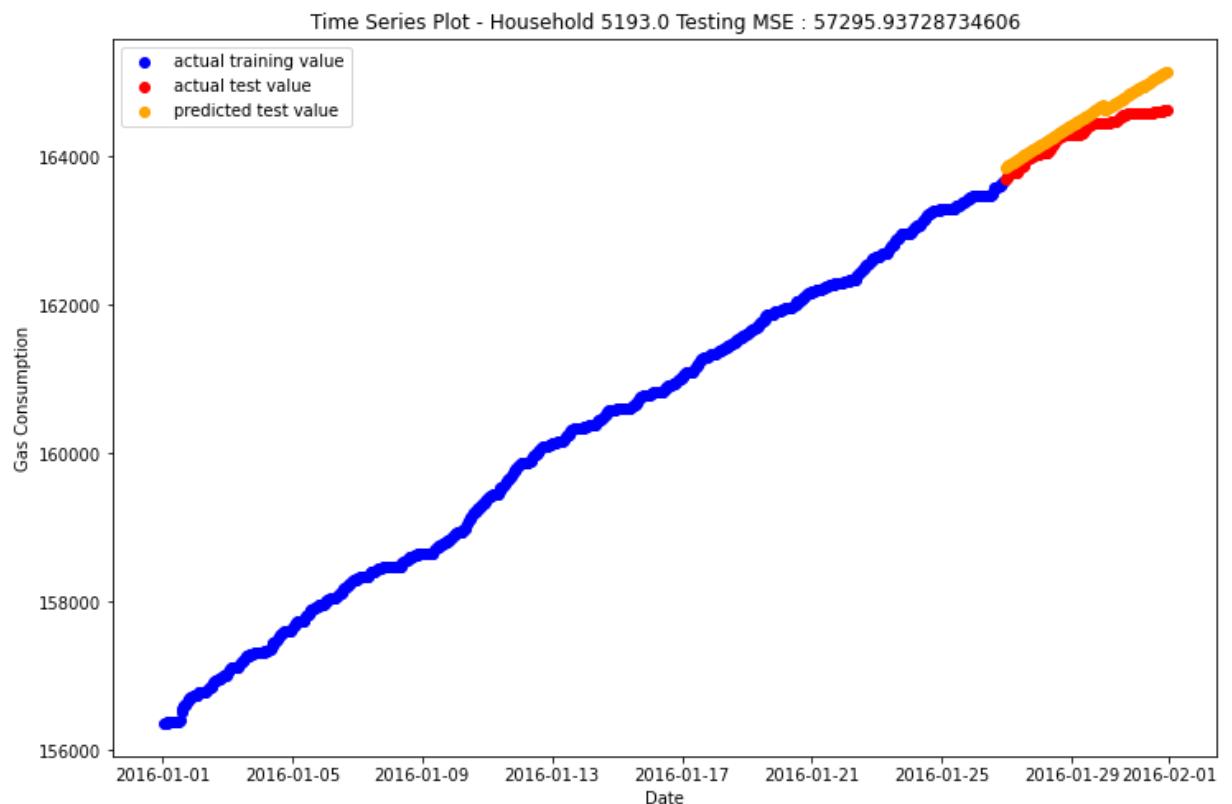


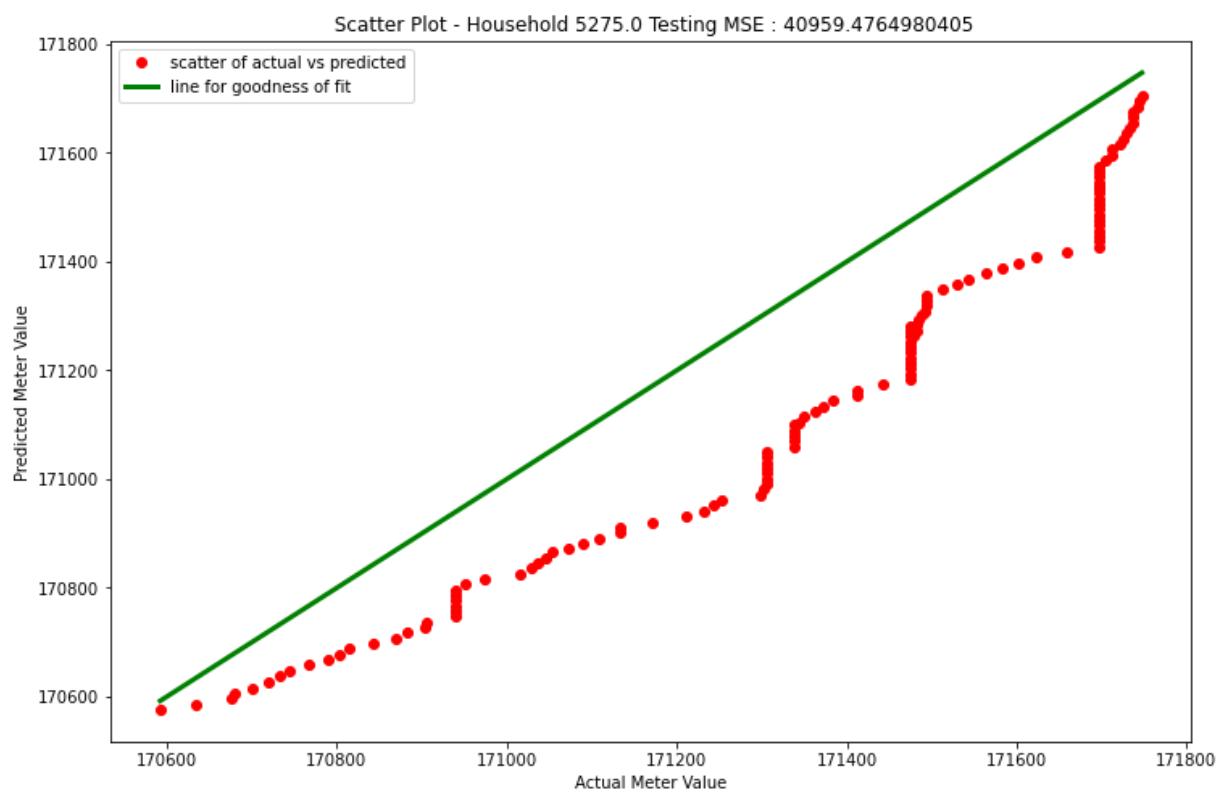
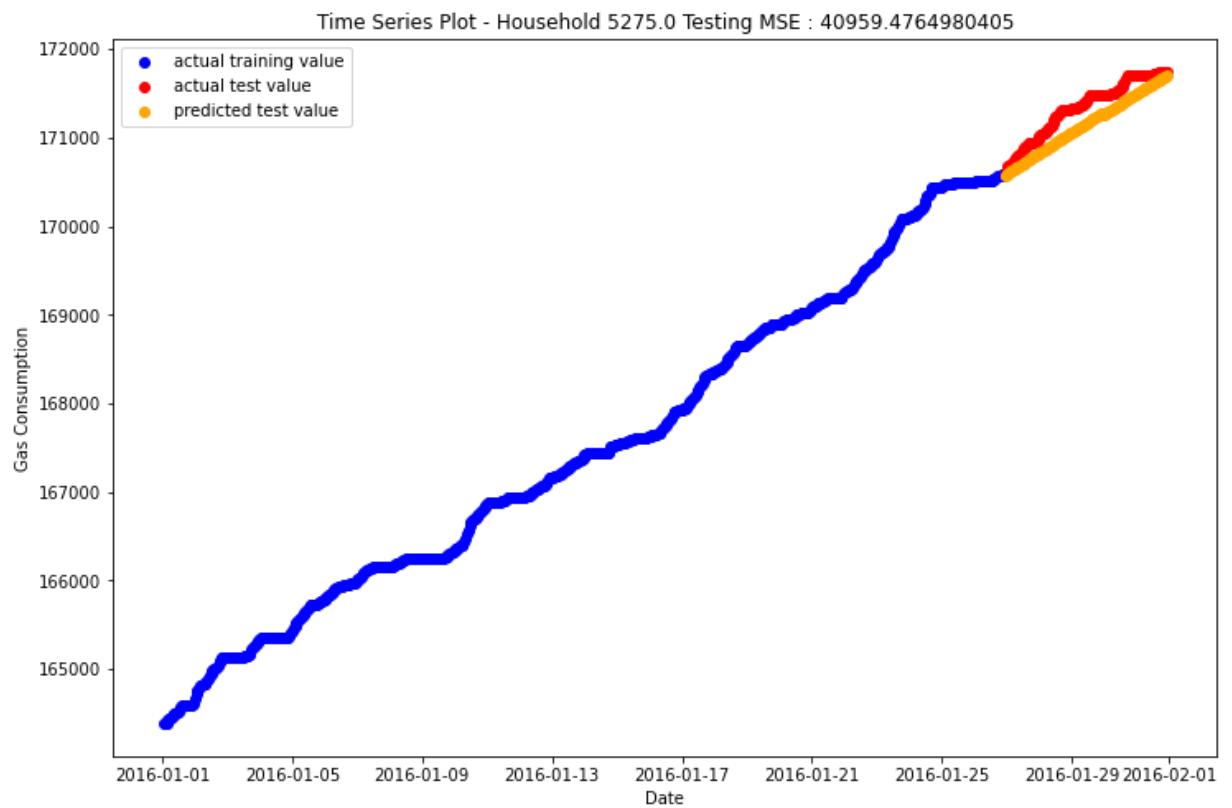


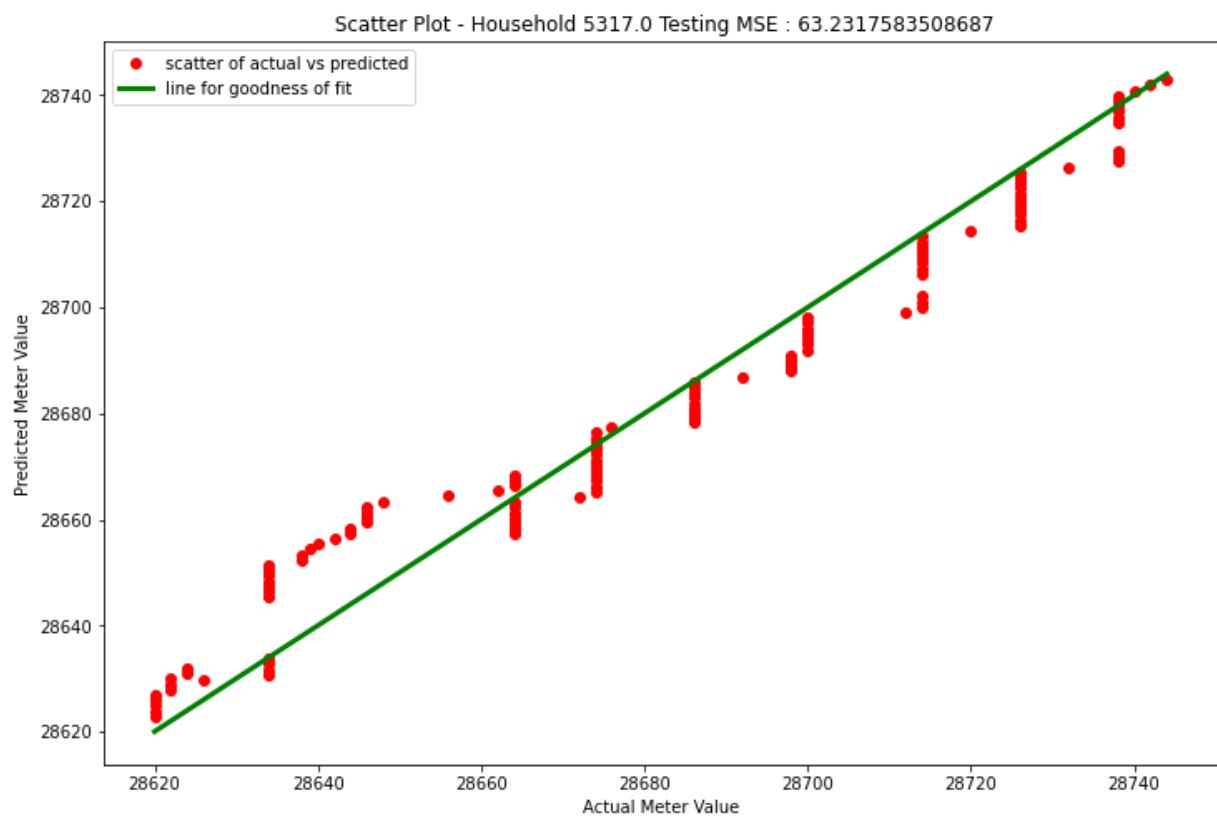
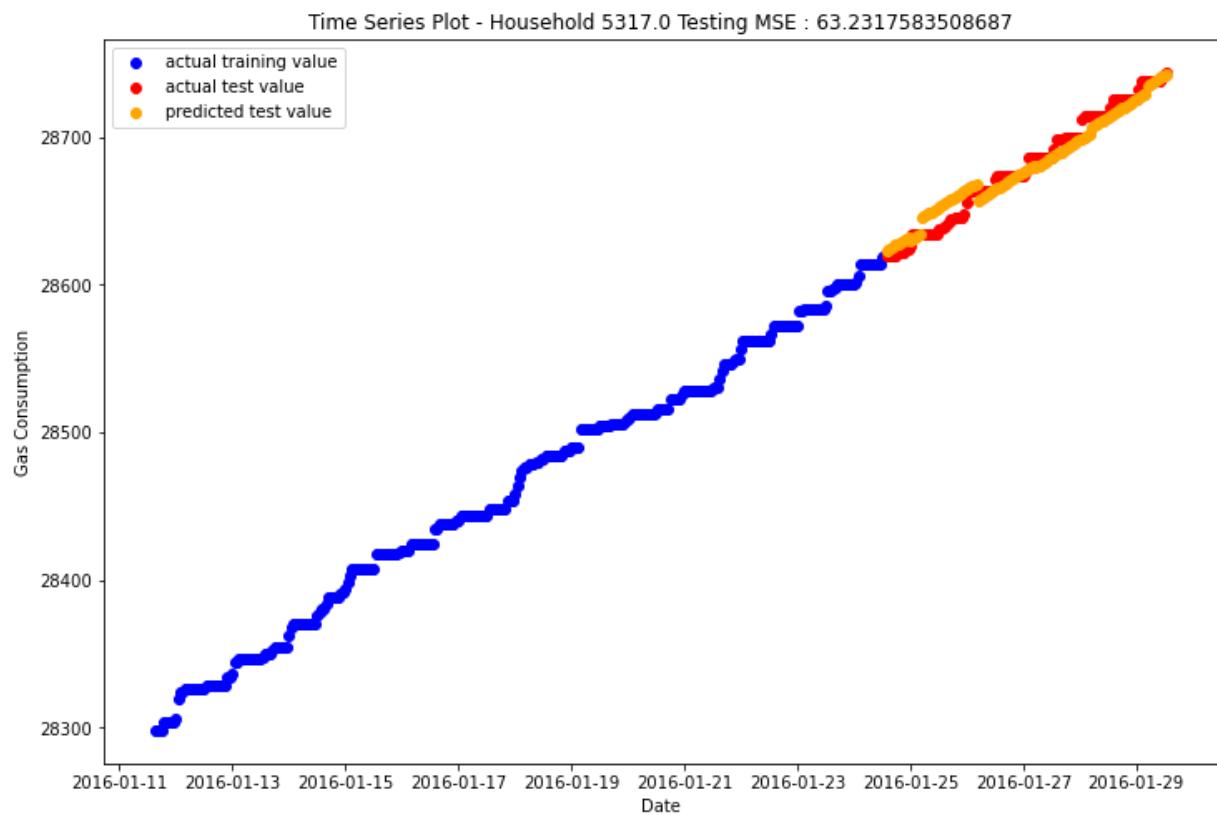


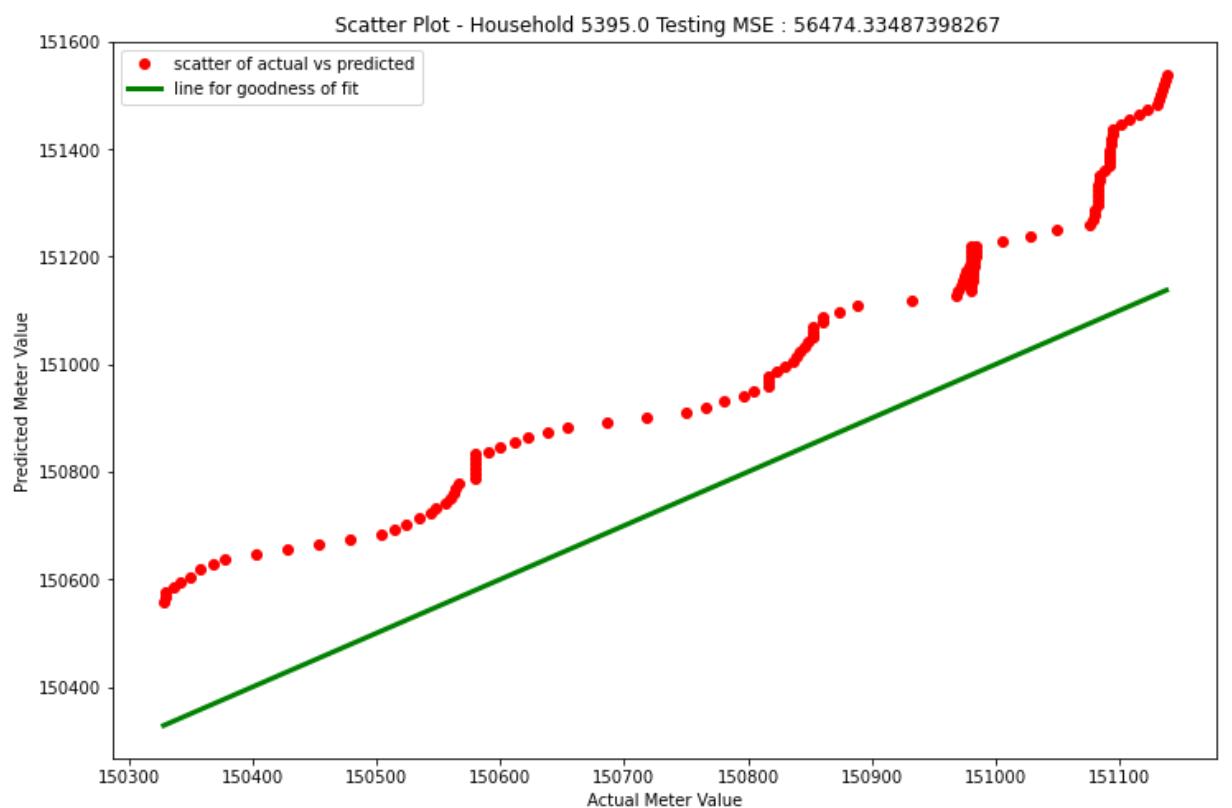
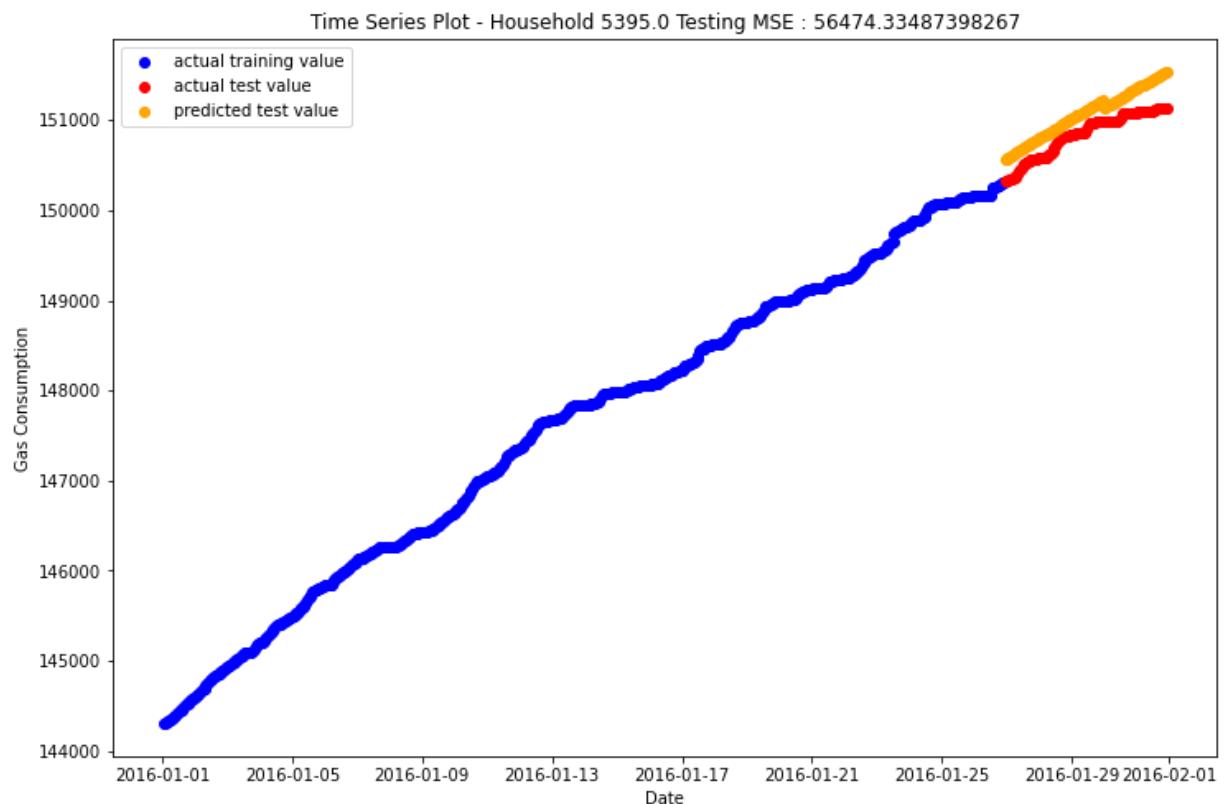


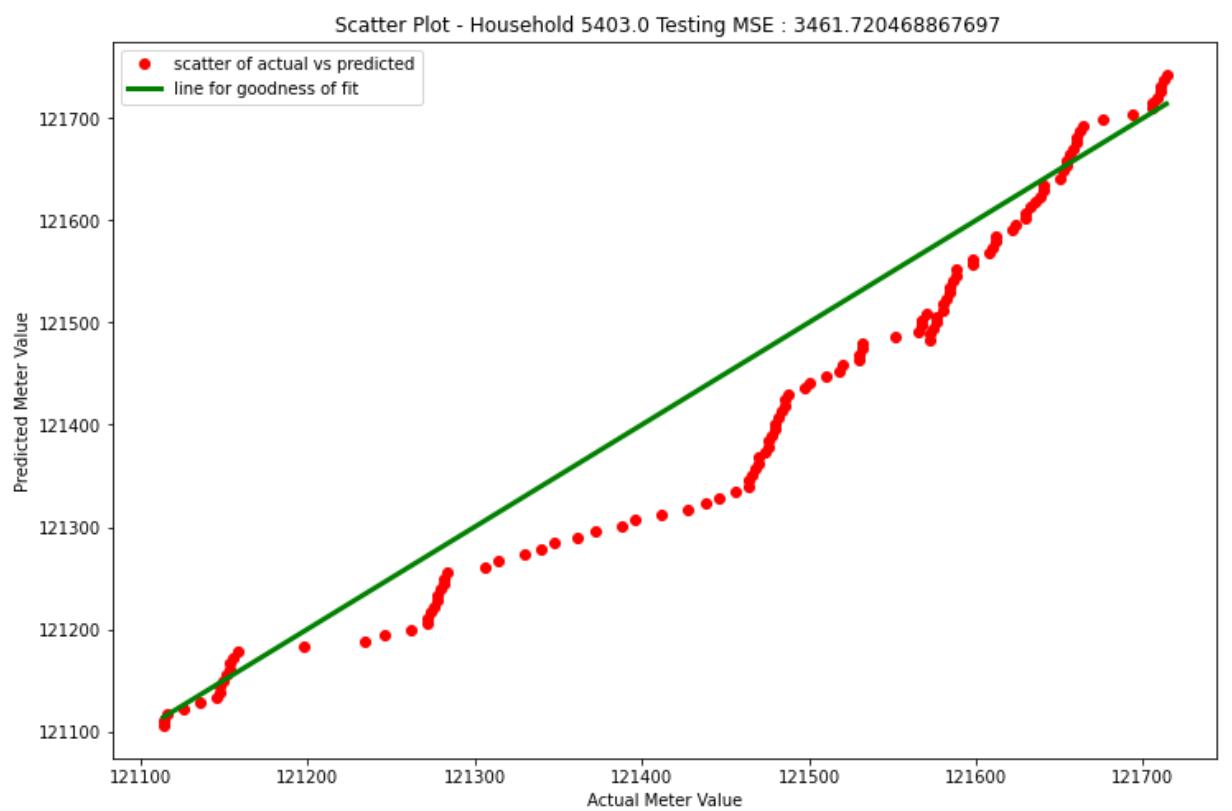
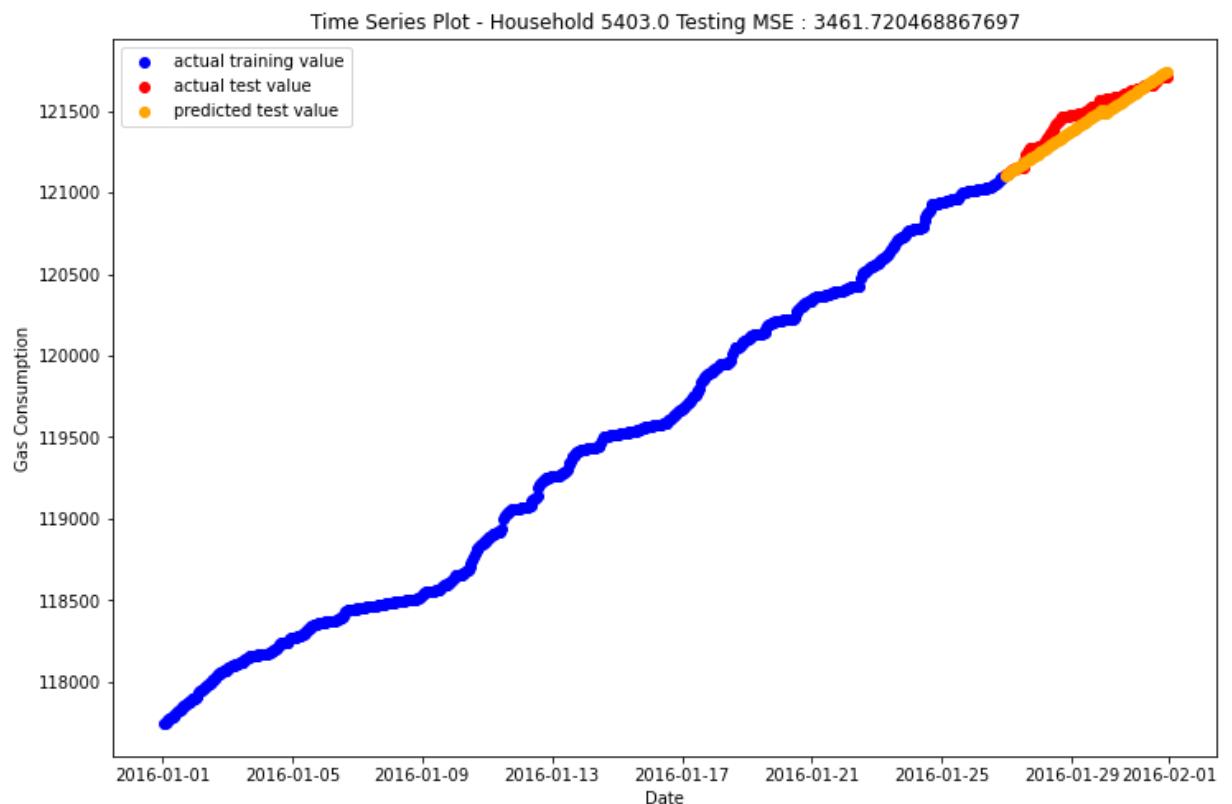




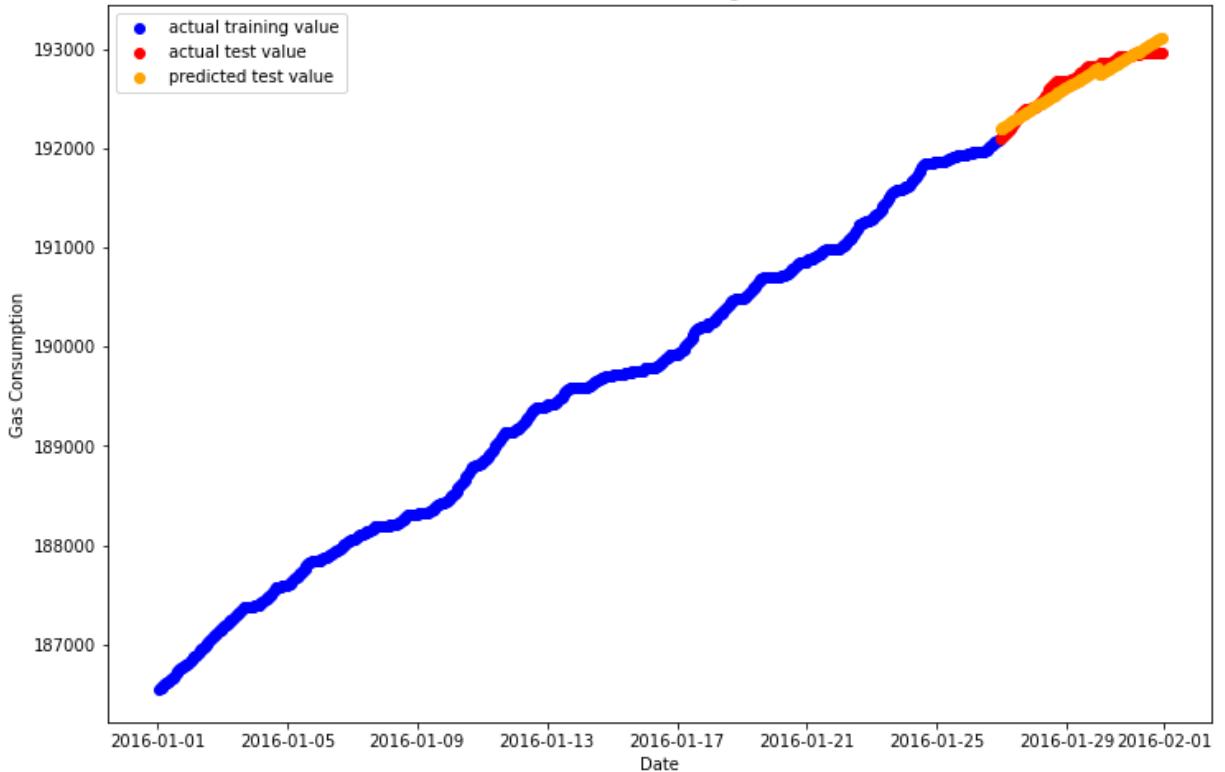




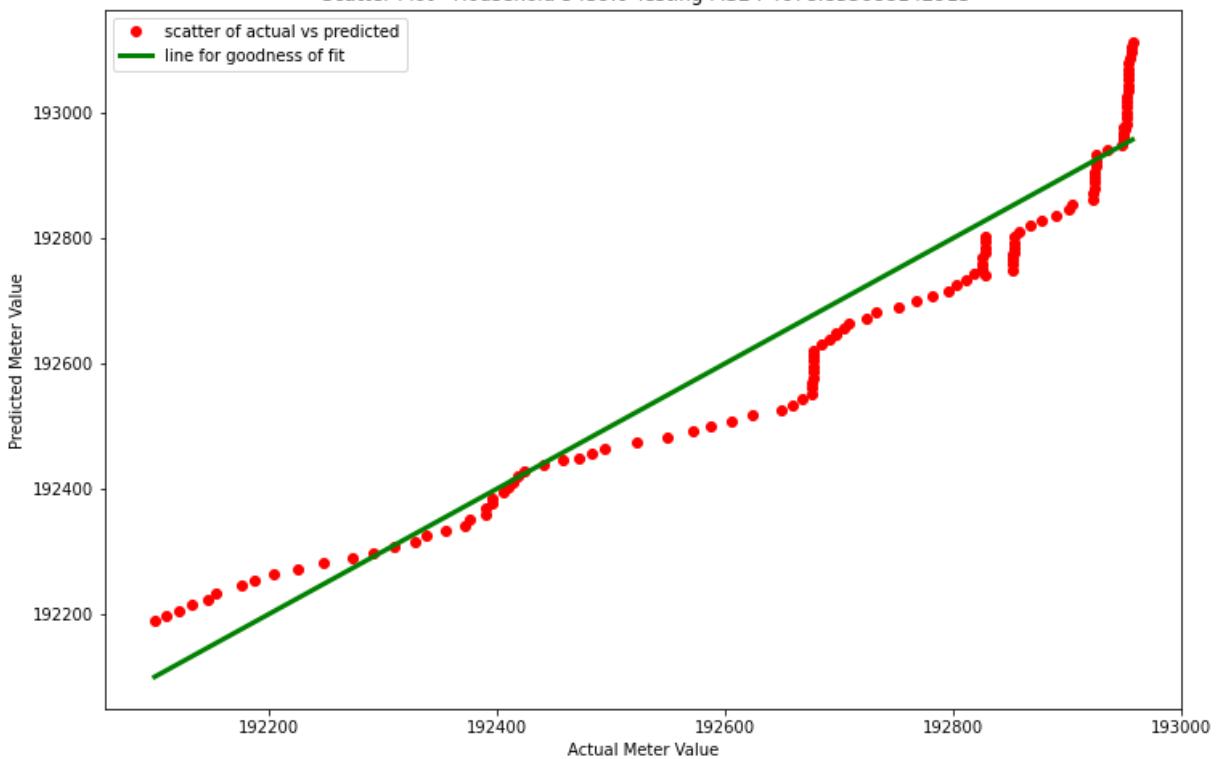




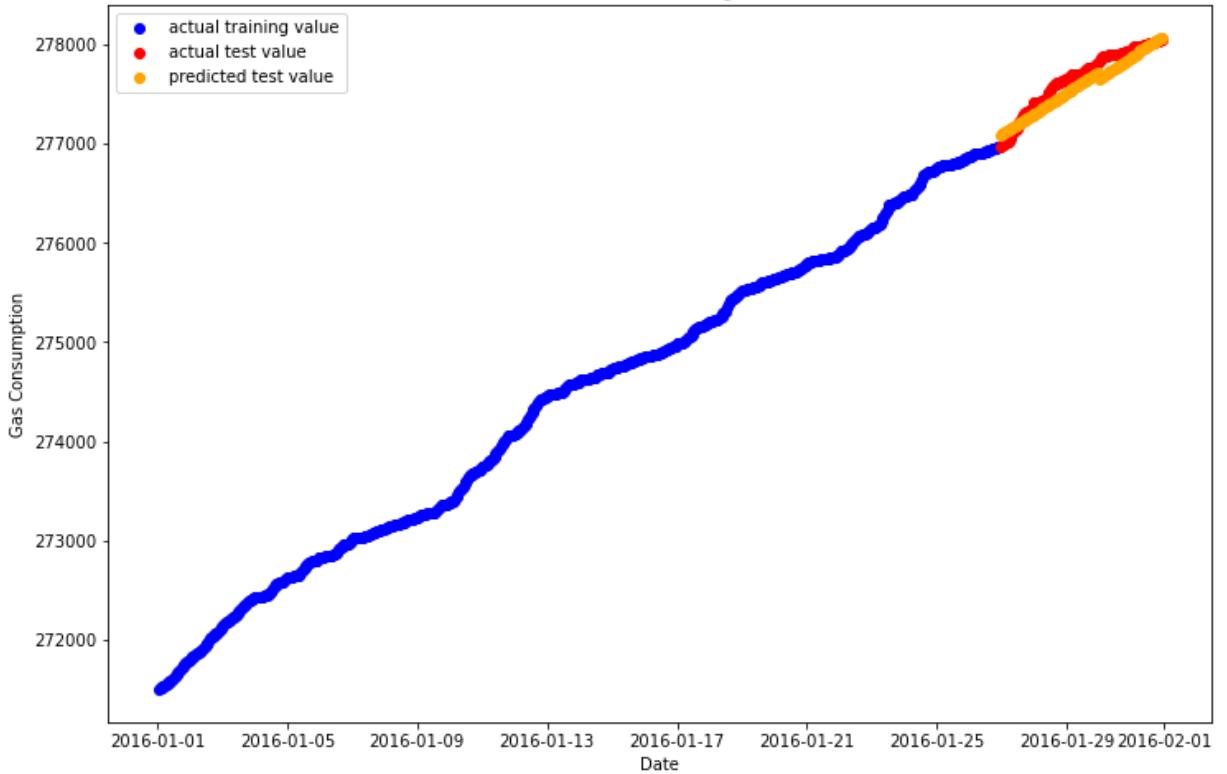
Time Series Plot - Household 5439.0 Testing MSE : 4678.833688142913



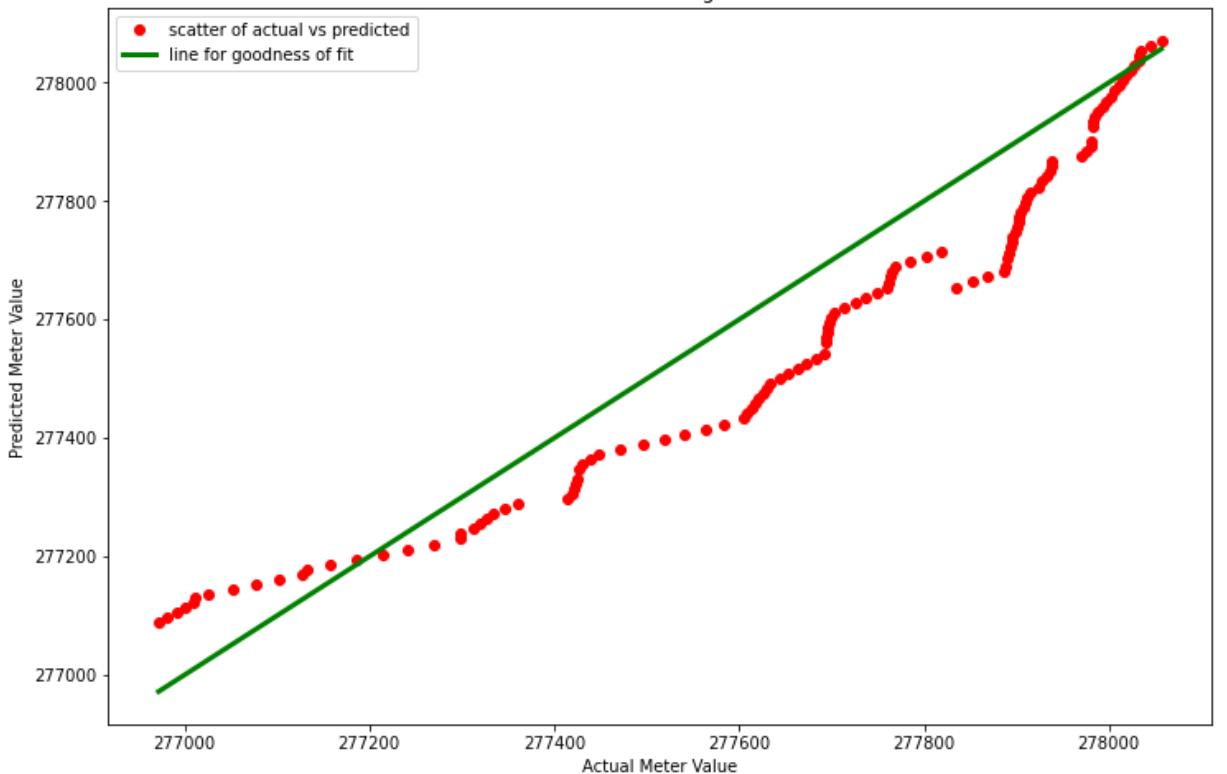
Scatter Plot - Household 5439.0 Testing MSE : 4678.833688142913



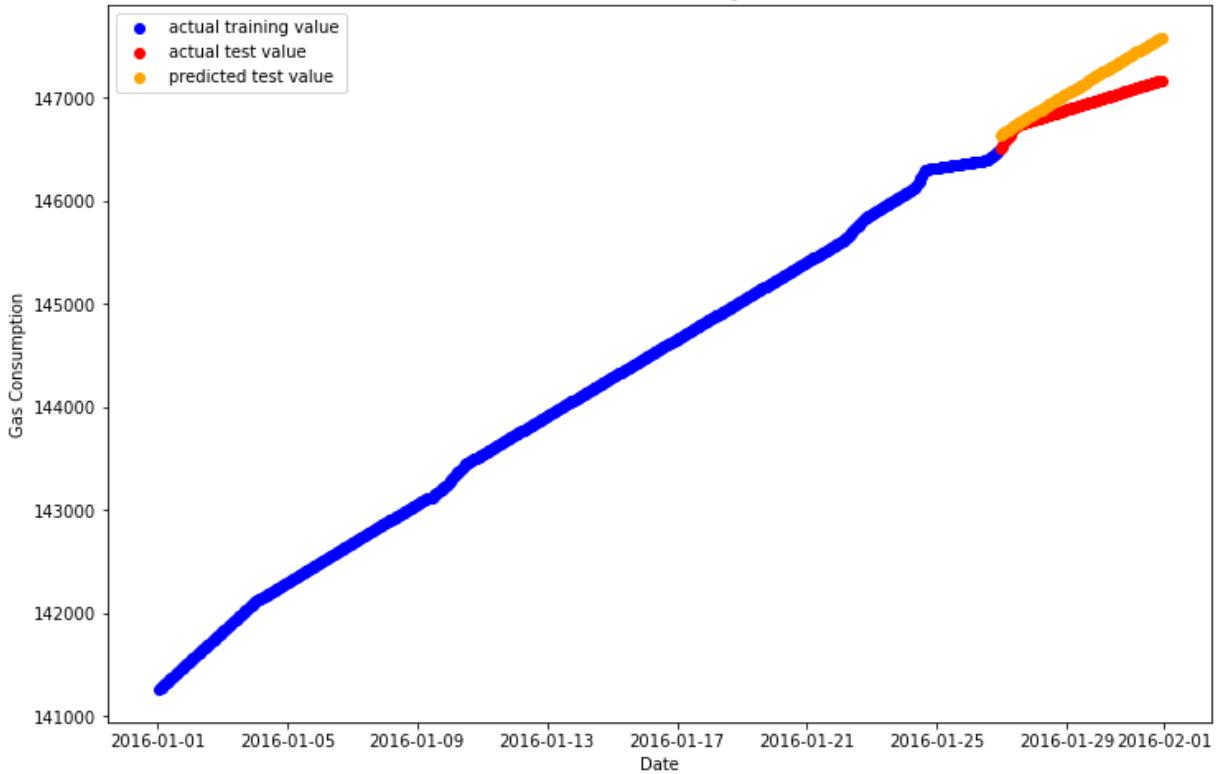
Time Series Plot - Household 5484.0 Testing MSE : 11852.143040546802



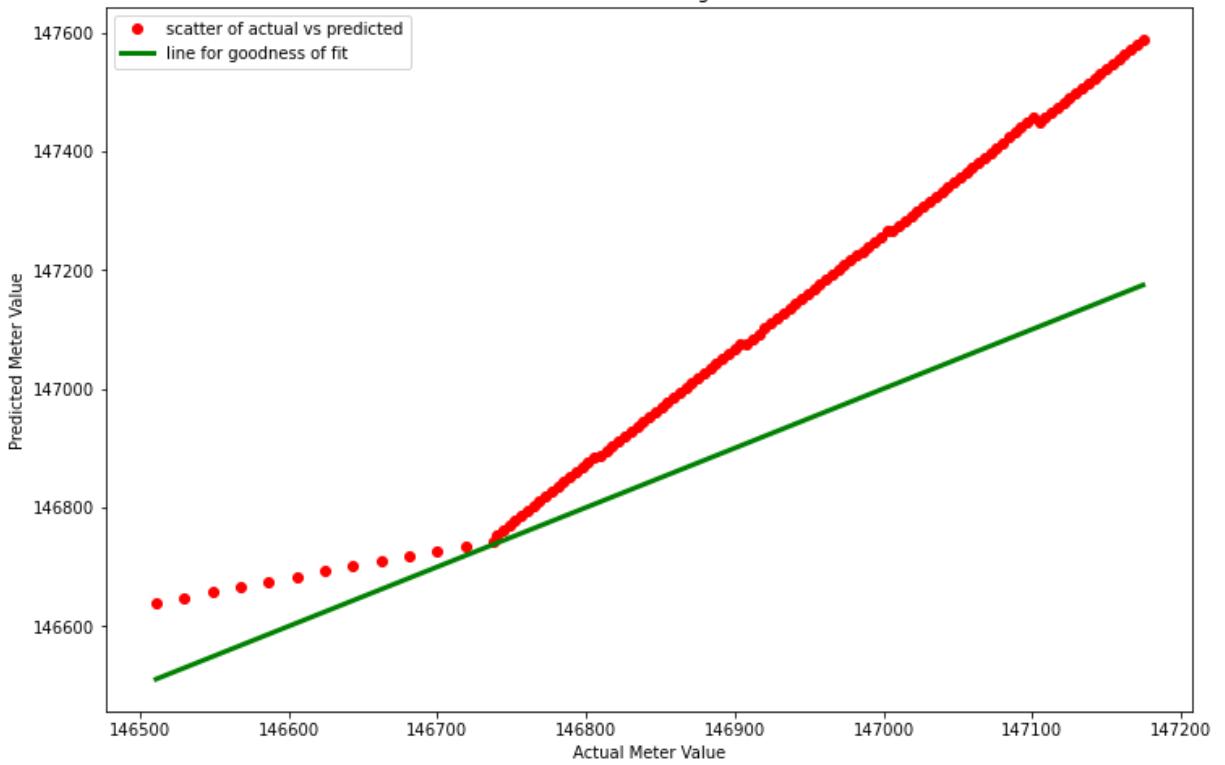
Scatter Plot - Household 5484.0 Testing MSE : 11852.143040546802

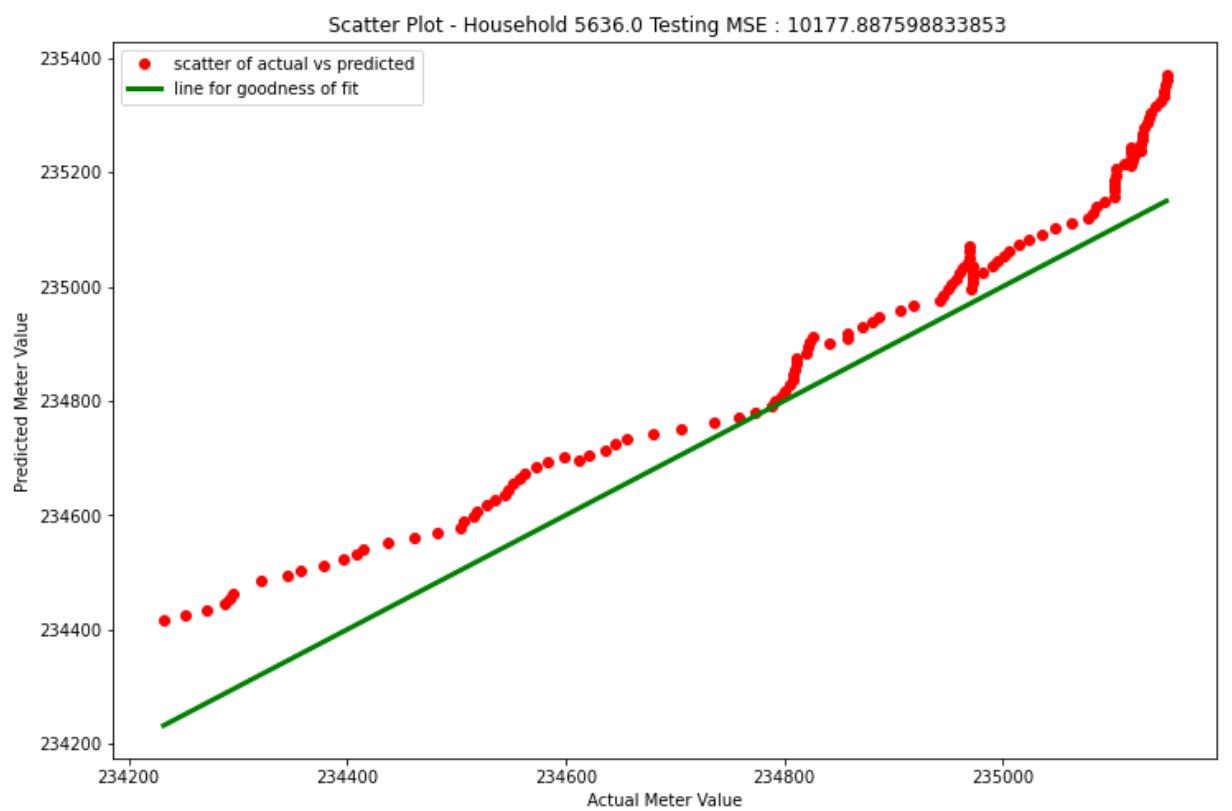
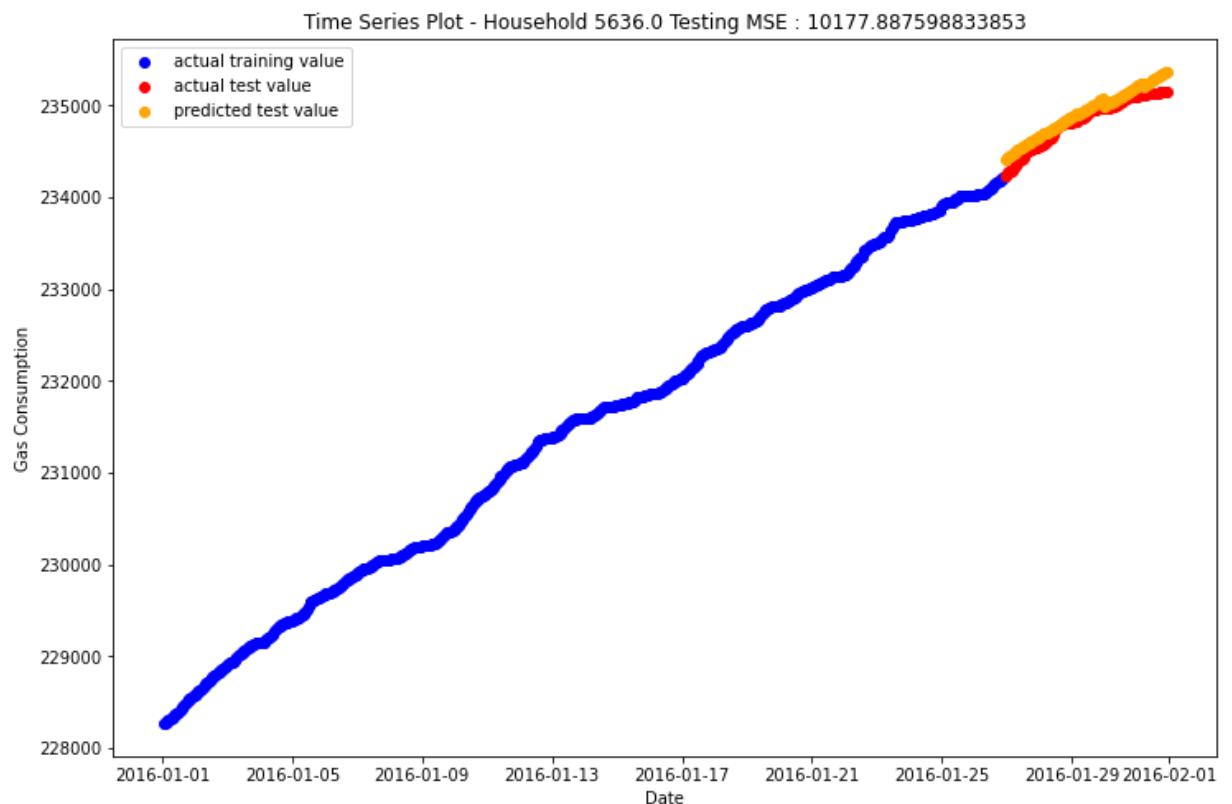


Time Series Plot - Household 5545.0 Testing MSE : 54422.88577809072

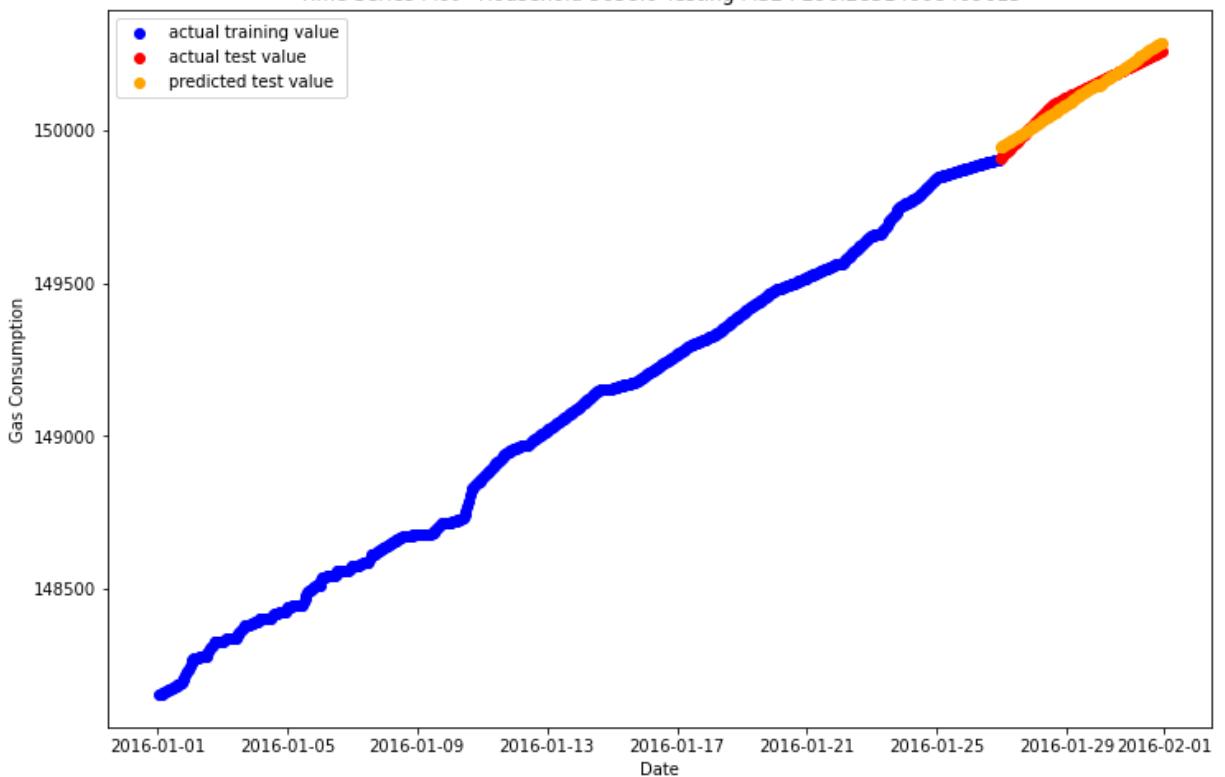


Scatter Plot - Household 5545.0 Testing MSE : 54422.88577809072

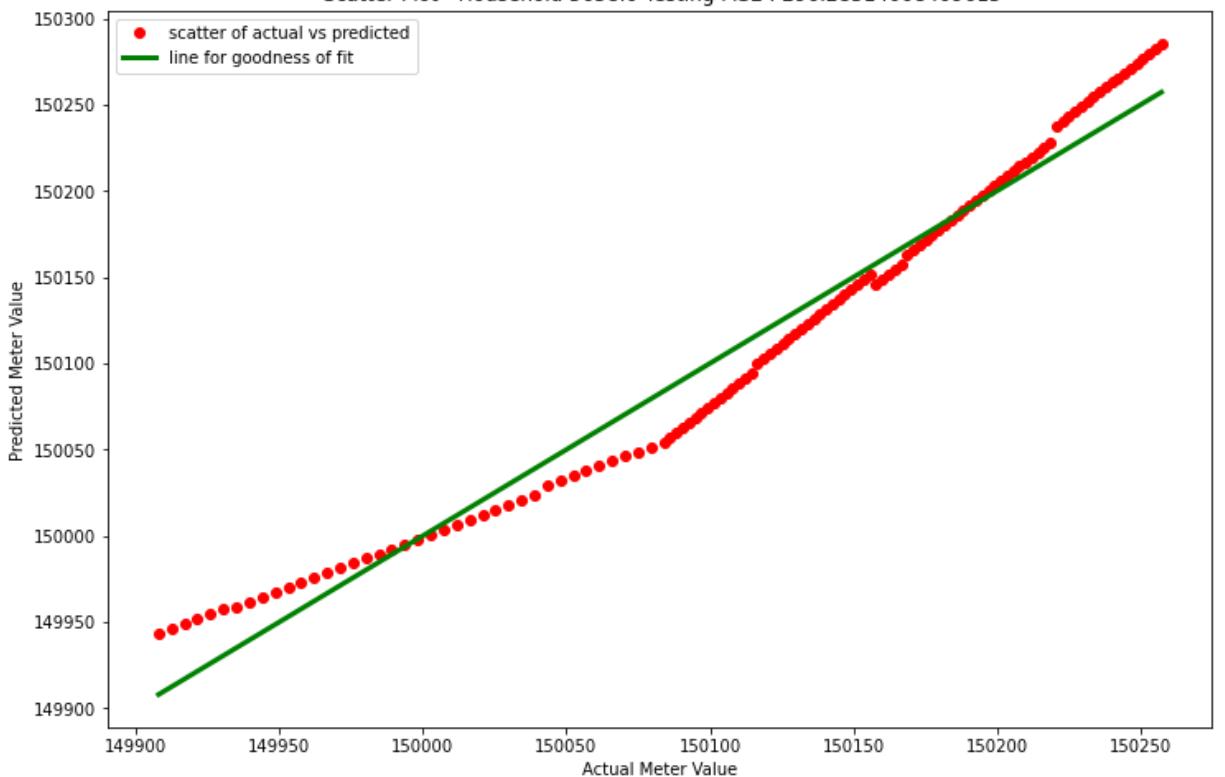


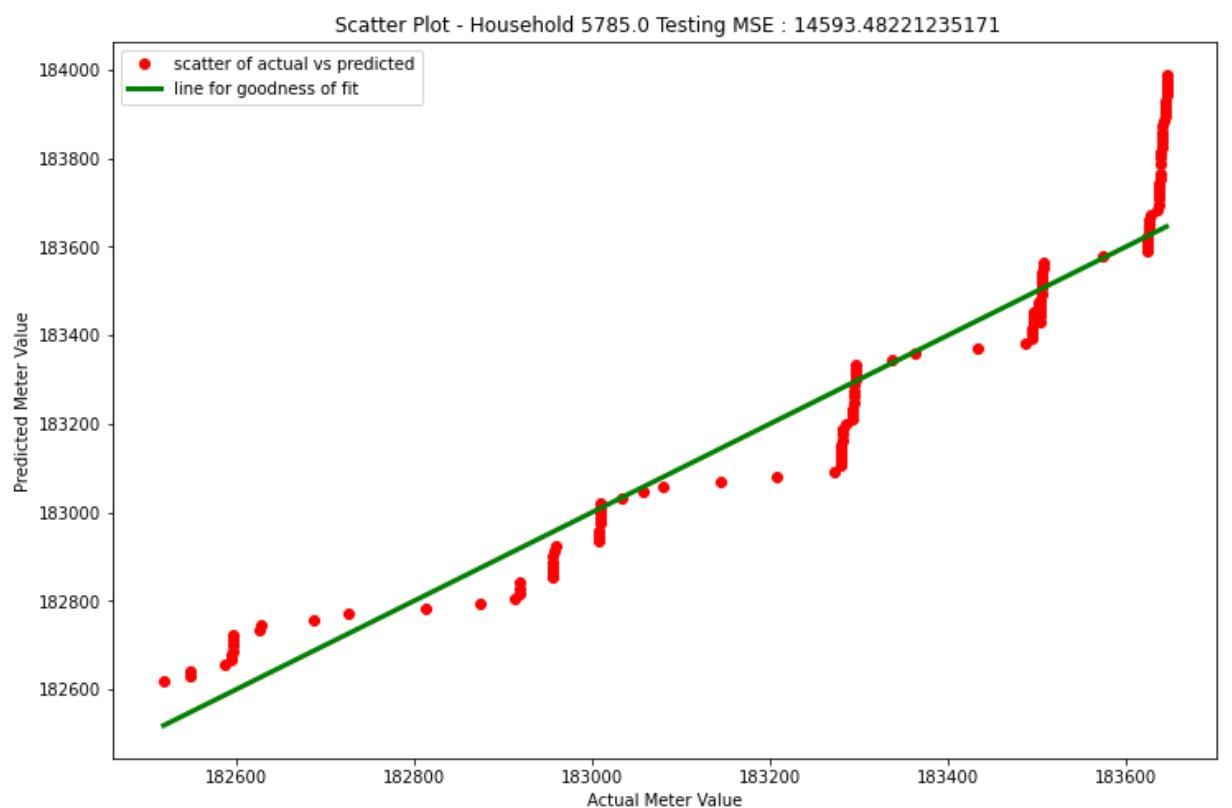
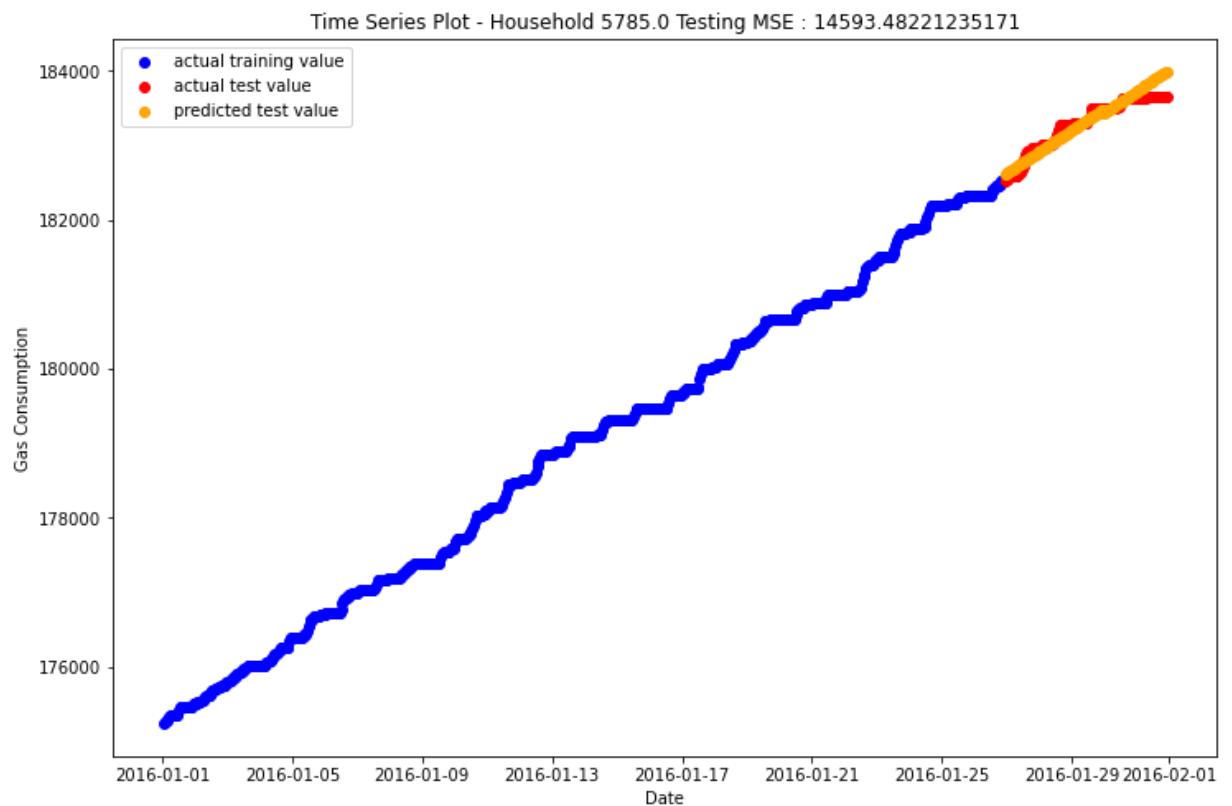


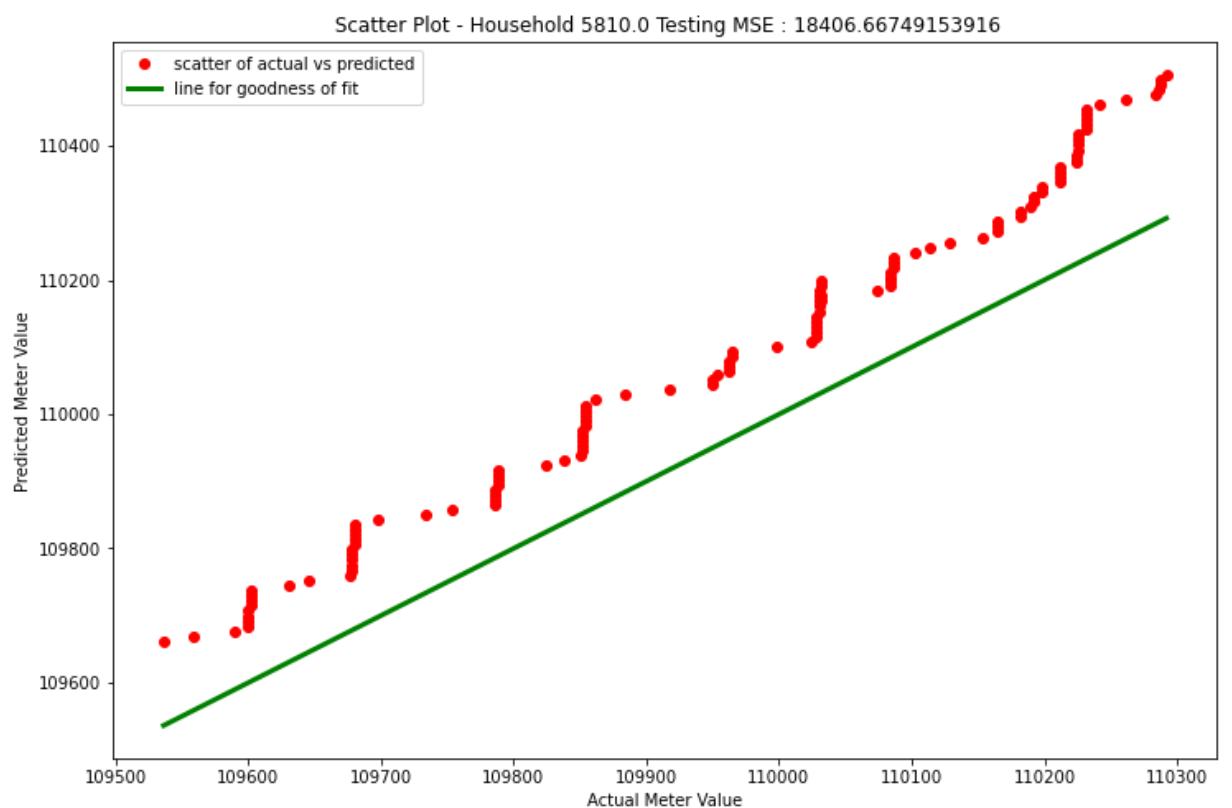
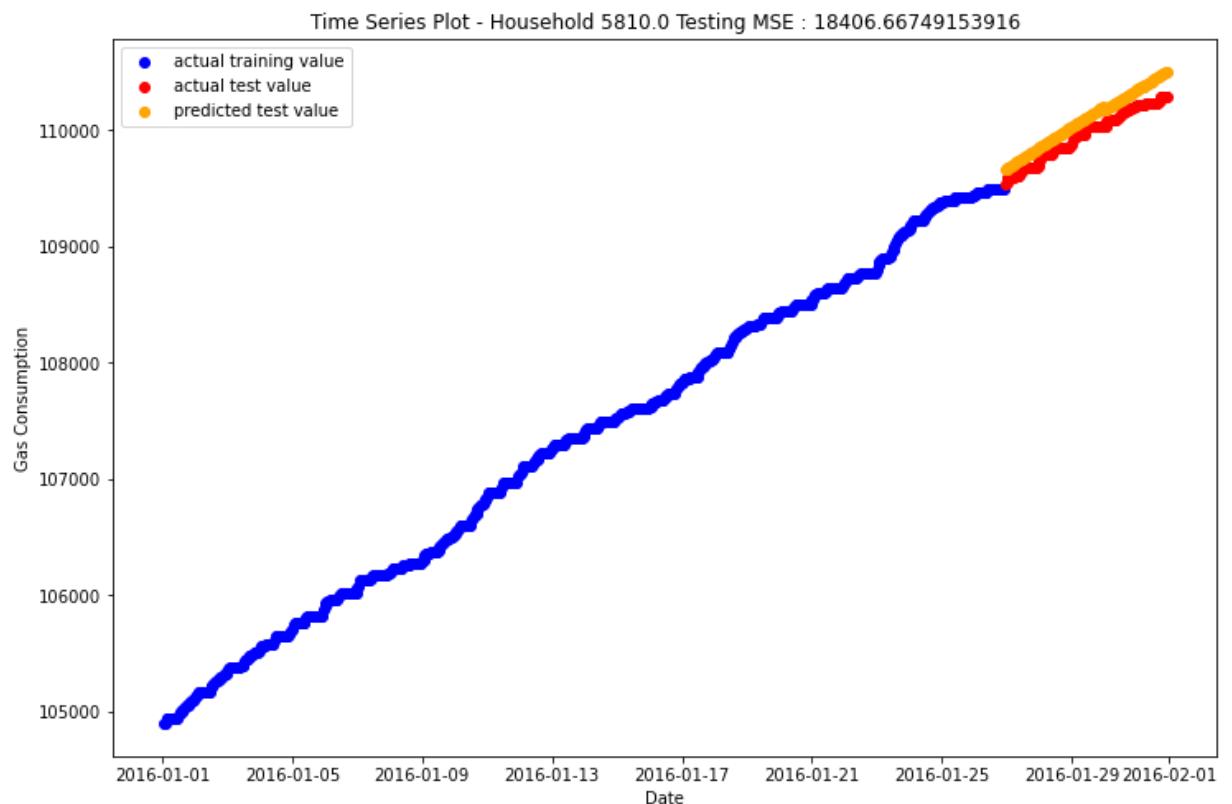
Time Series Plot - Household 5658.0 Testing MSE : 296.28314008409615



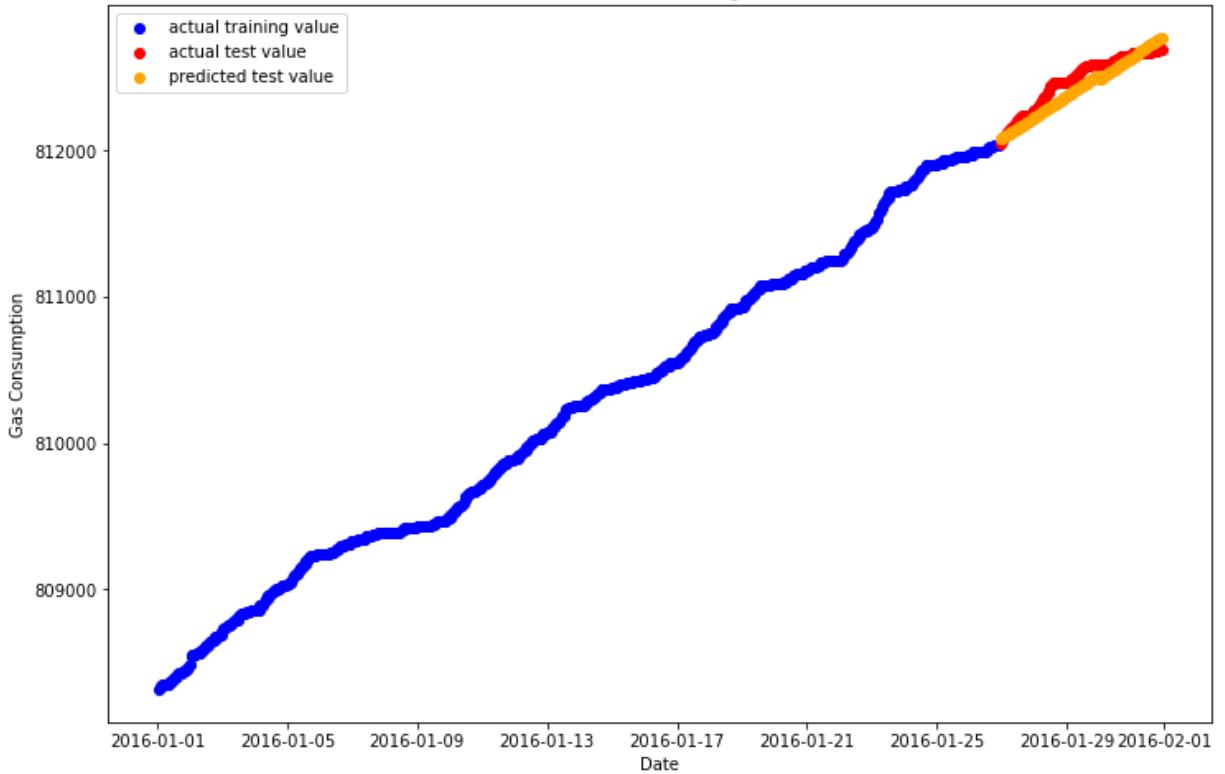
Scatter Plot - Household 5658.0 Testing MSE : 296.28314008409615



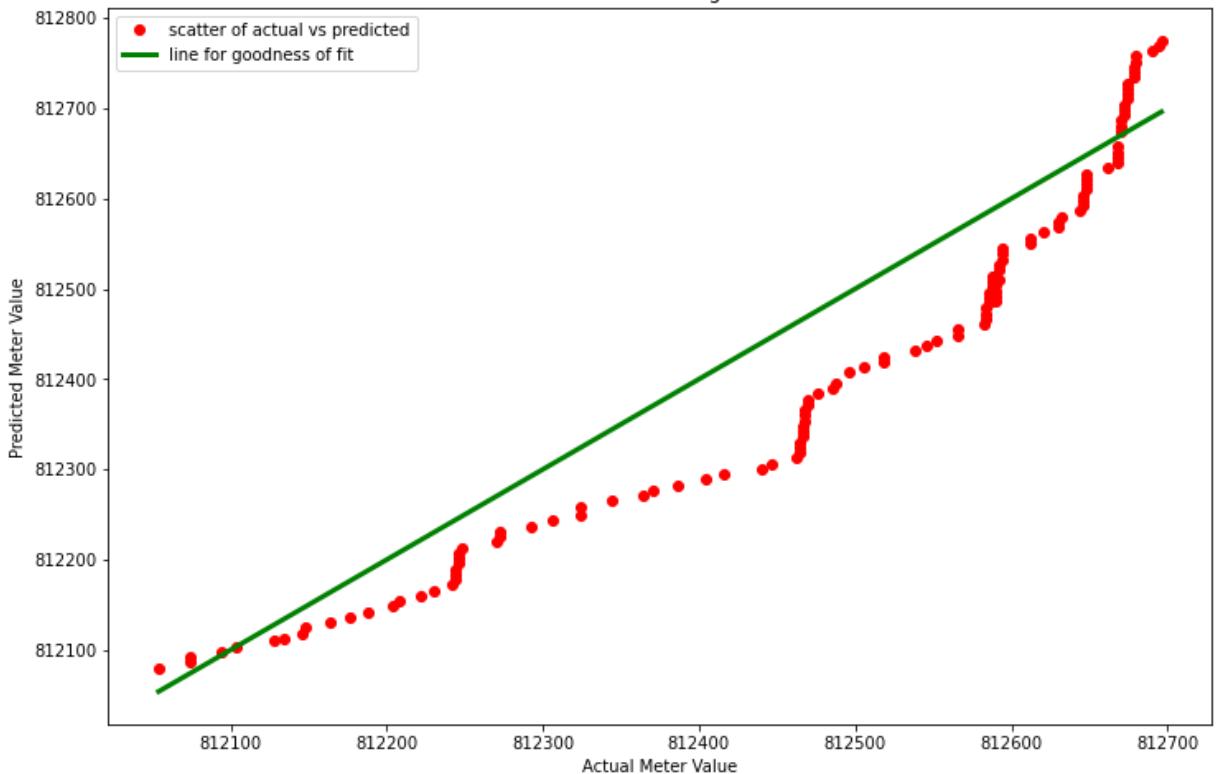


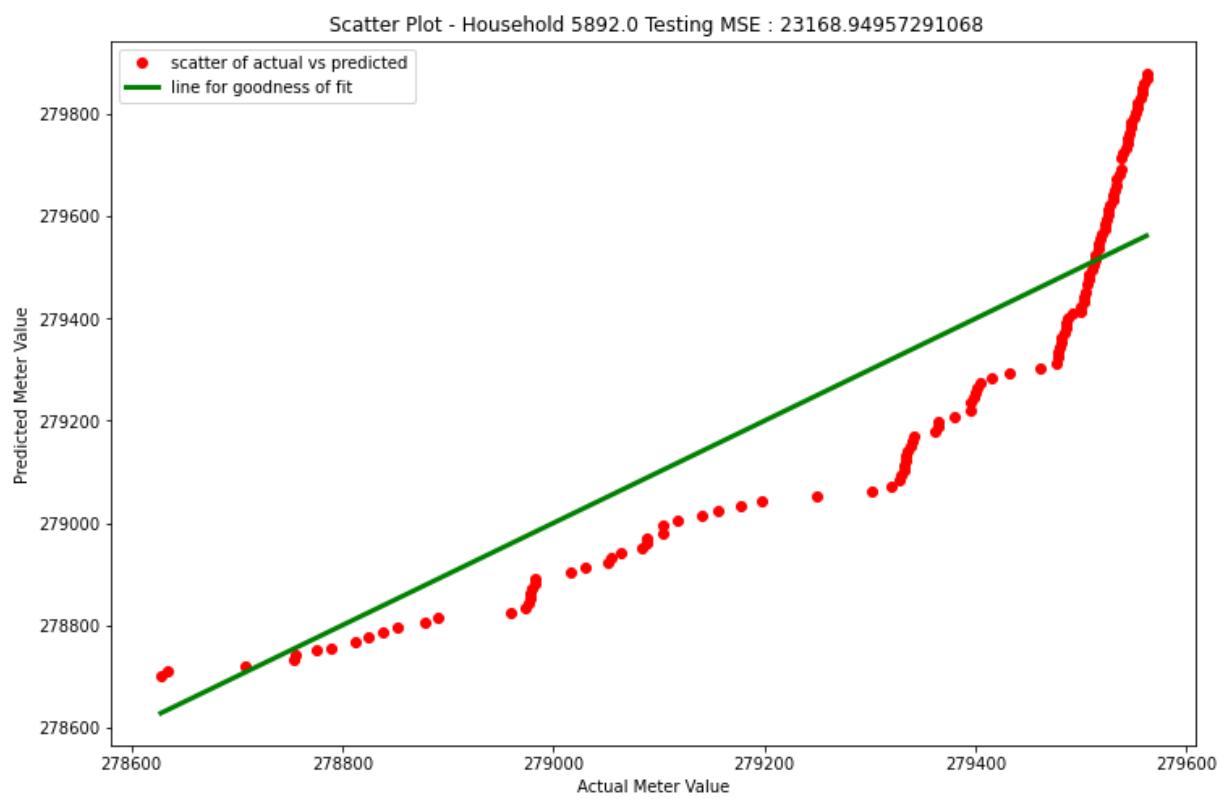
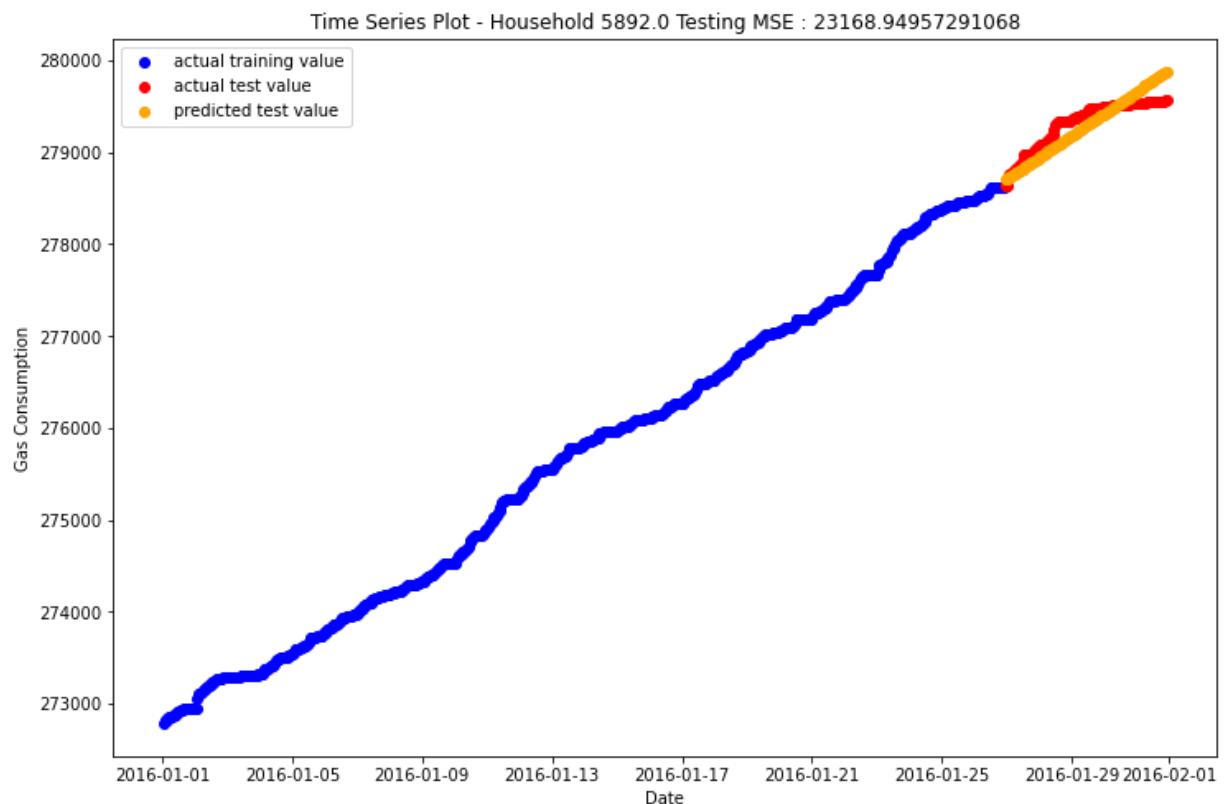


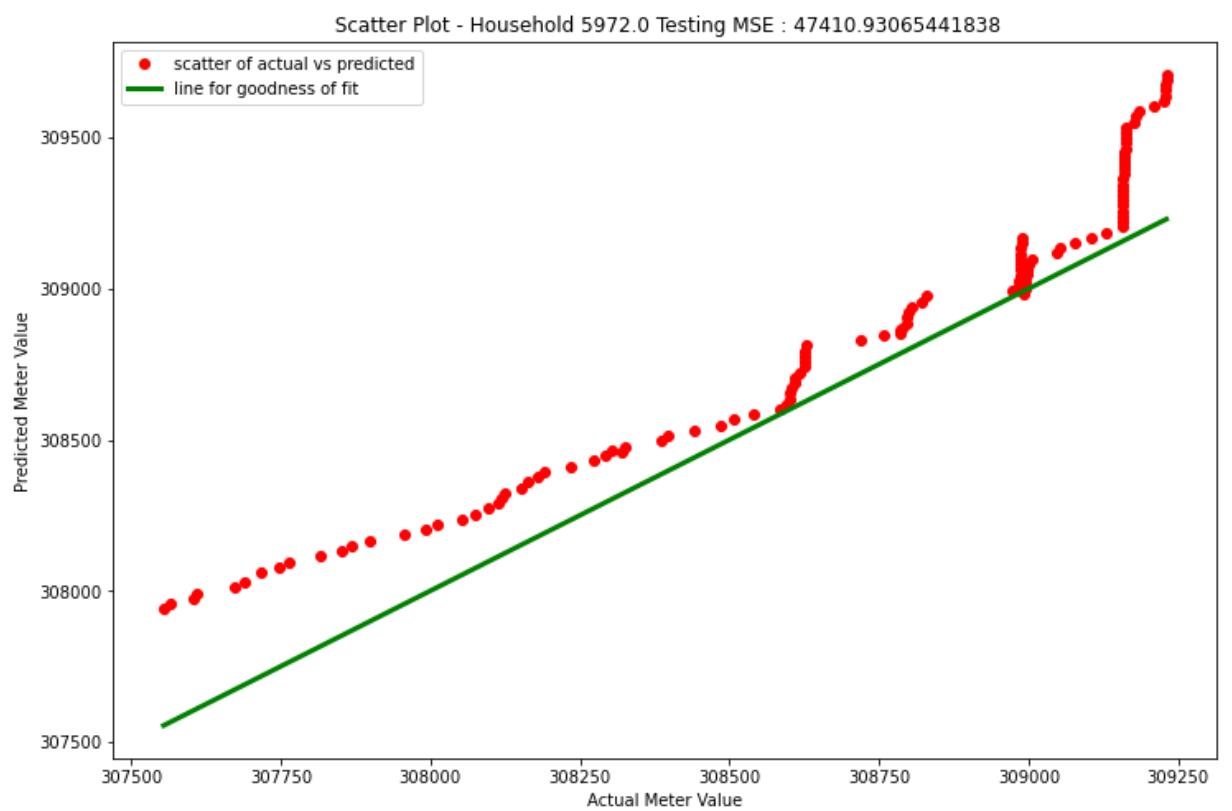
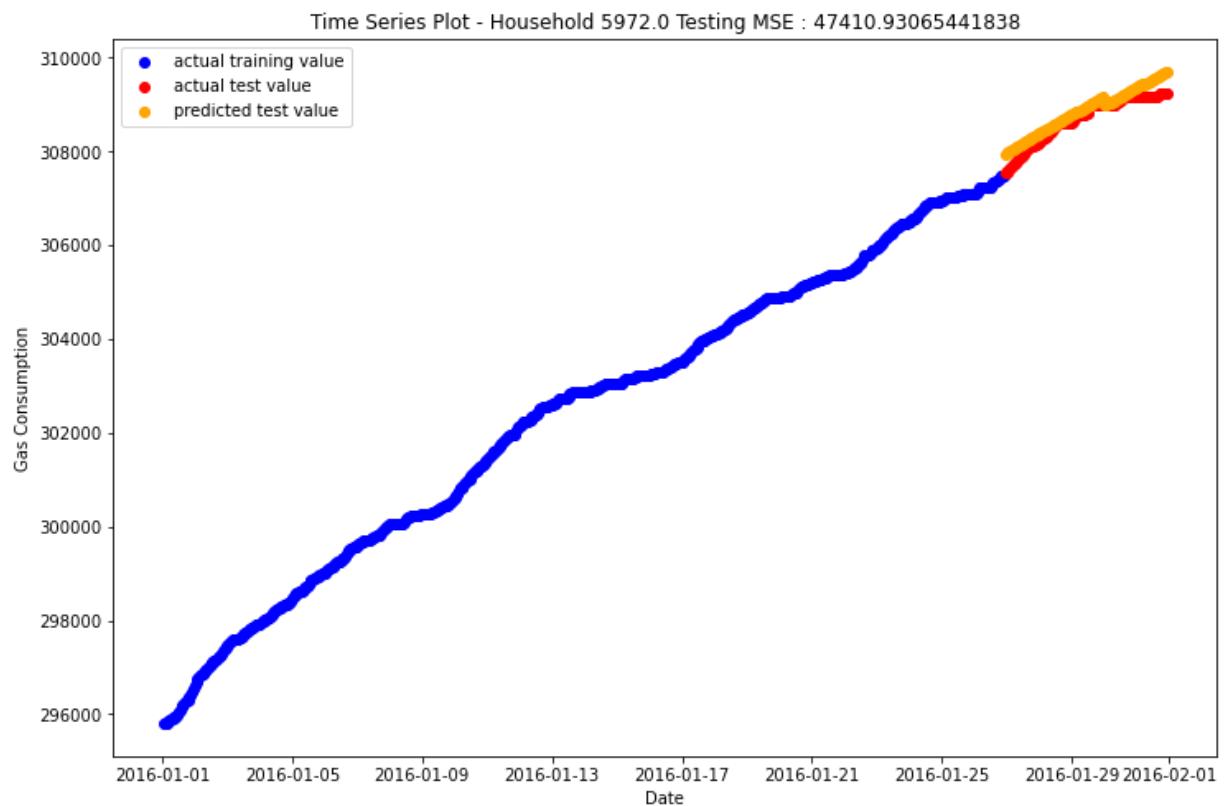
Time Series Plot - Household 5814.0 Testing MSE : 5998.639150035636

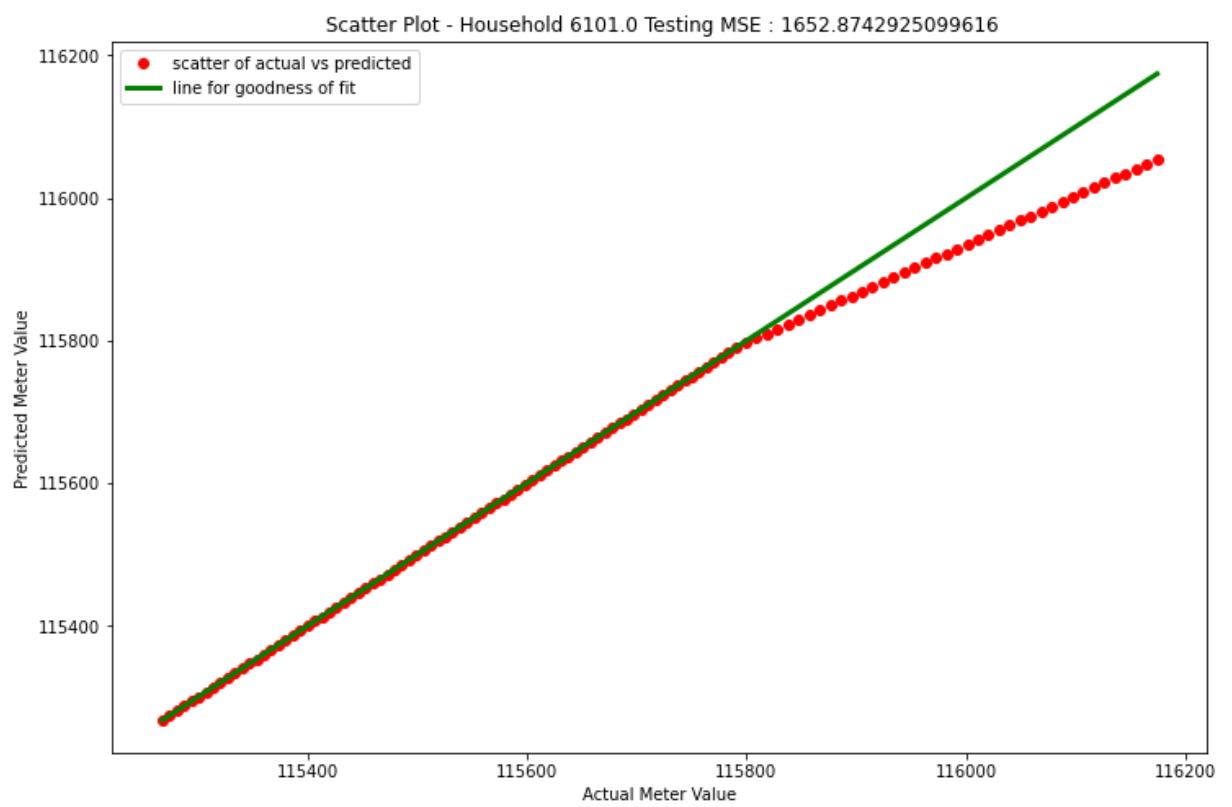
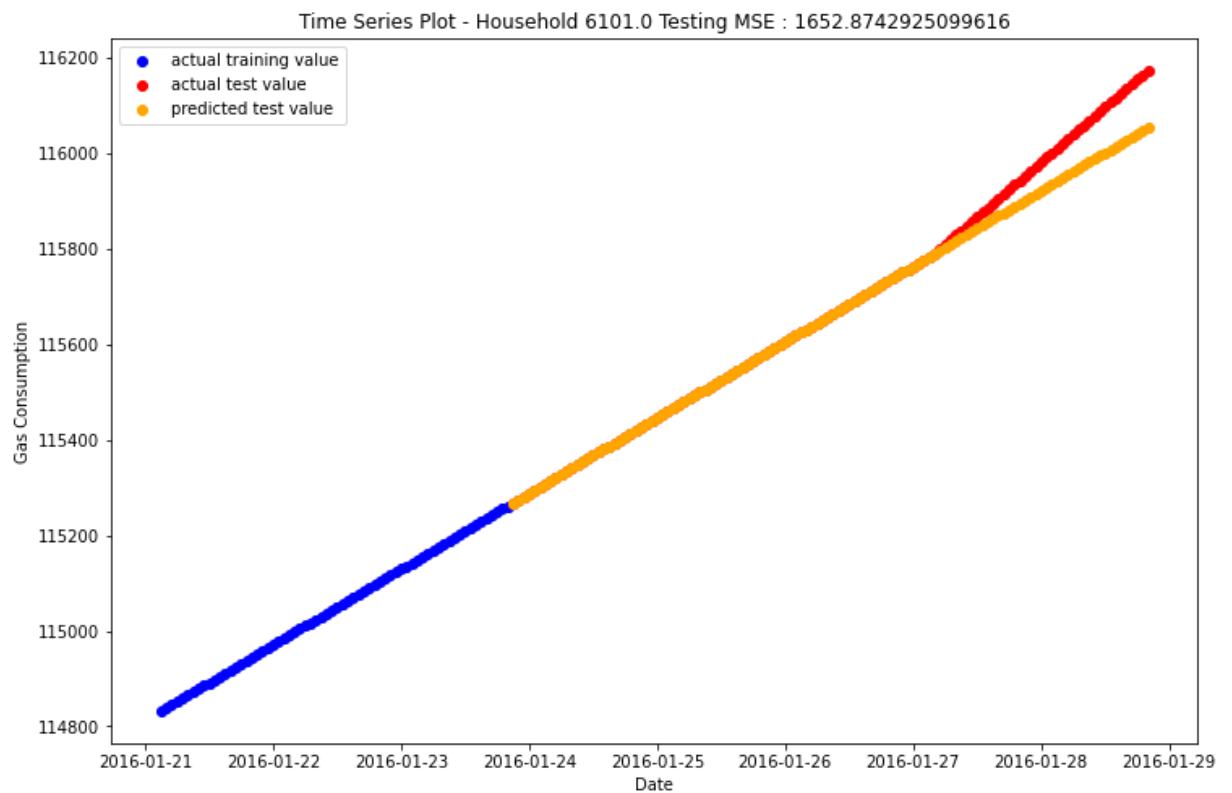


Scatter Plot - Household 5814.0 Testing MSE : 5998.639150035636

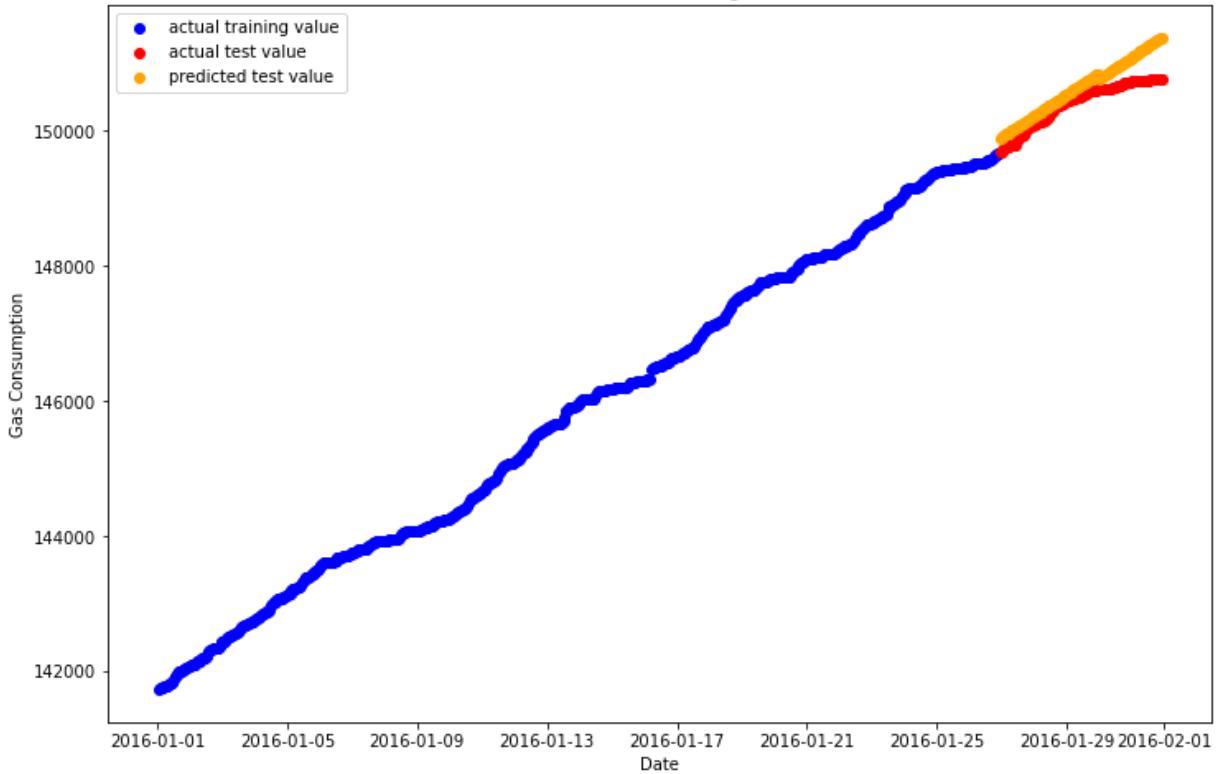




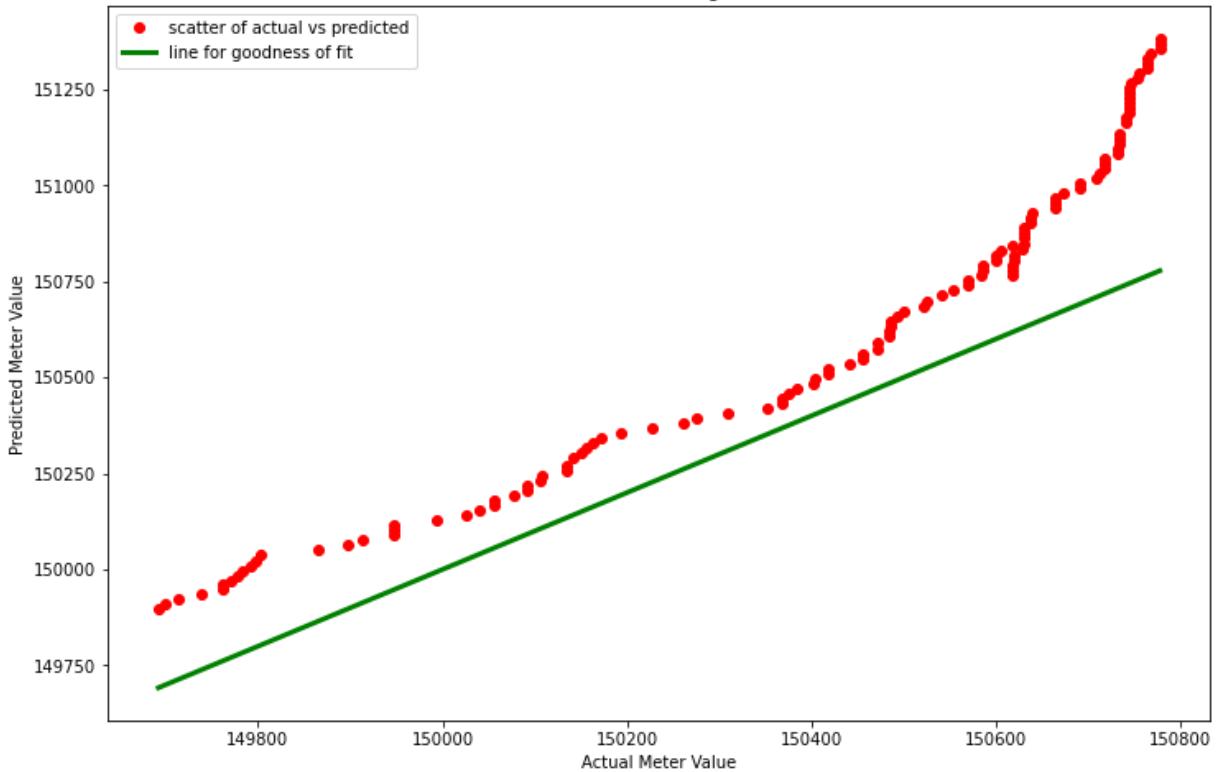


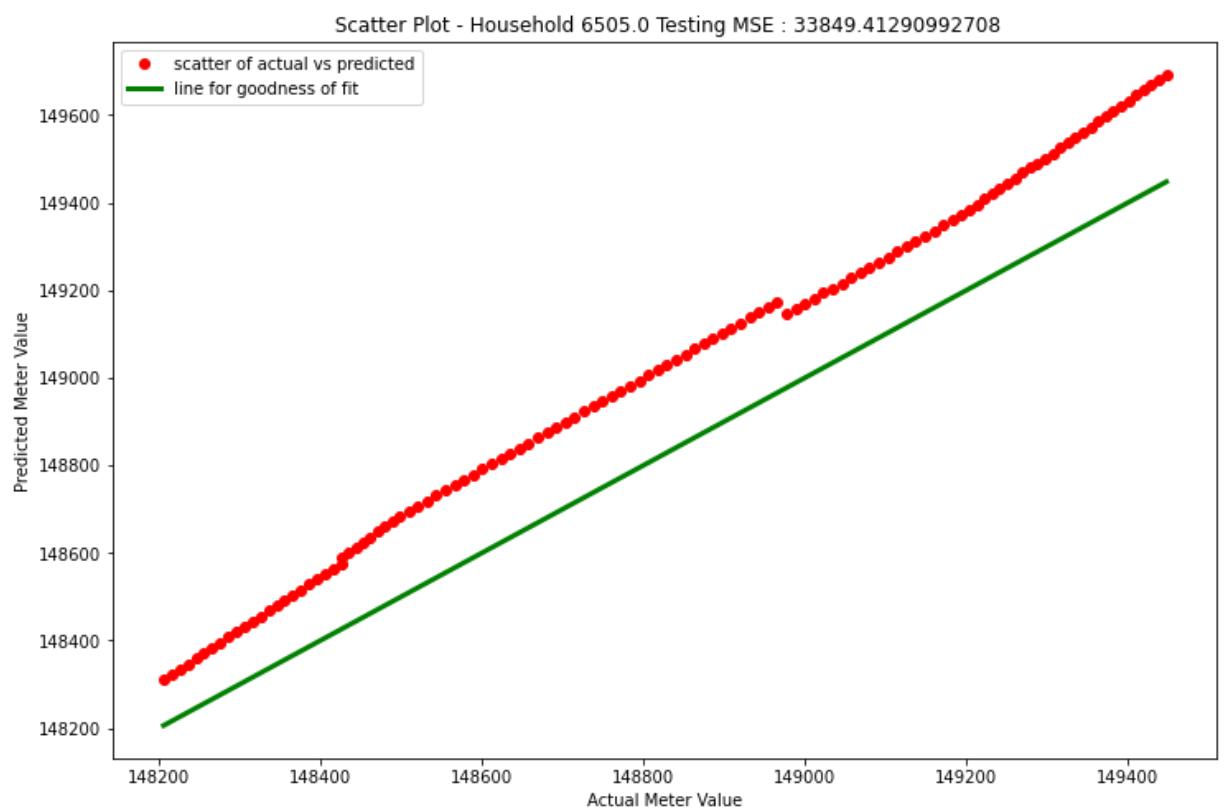
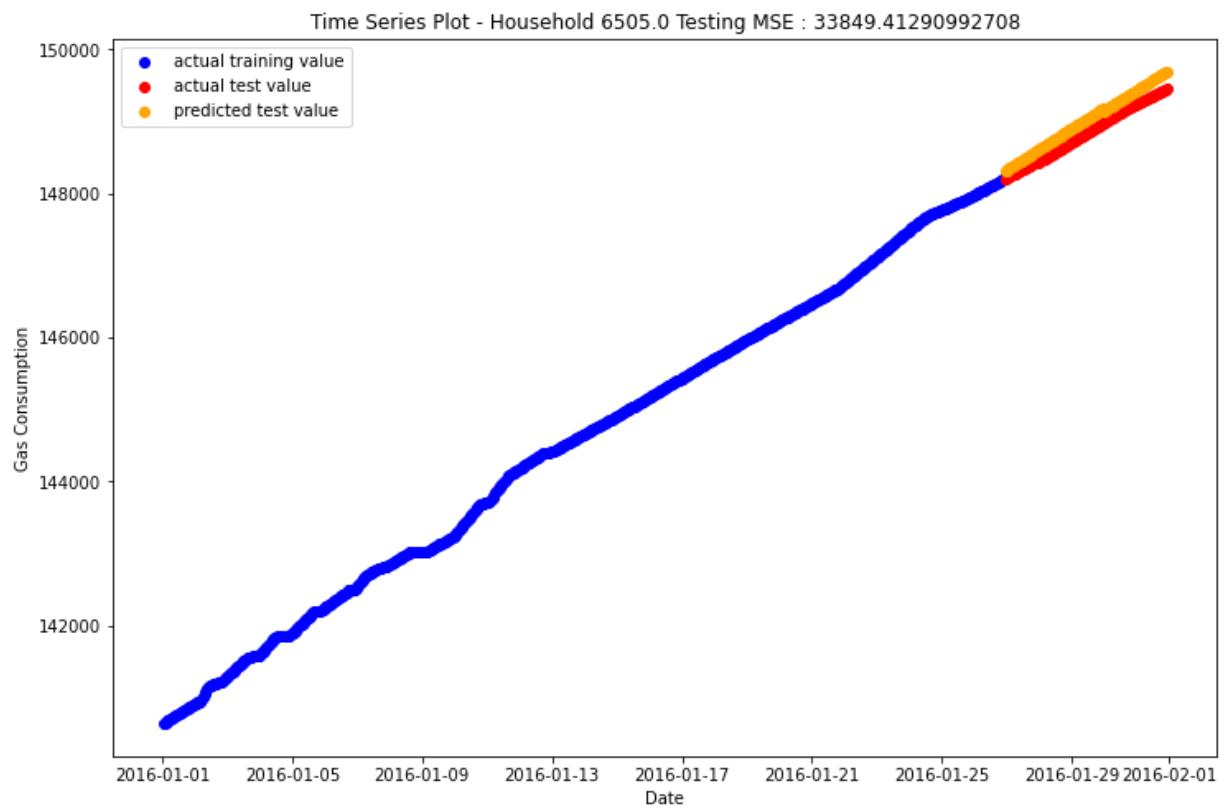


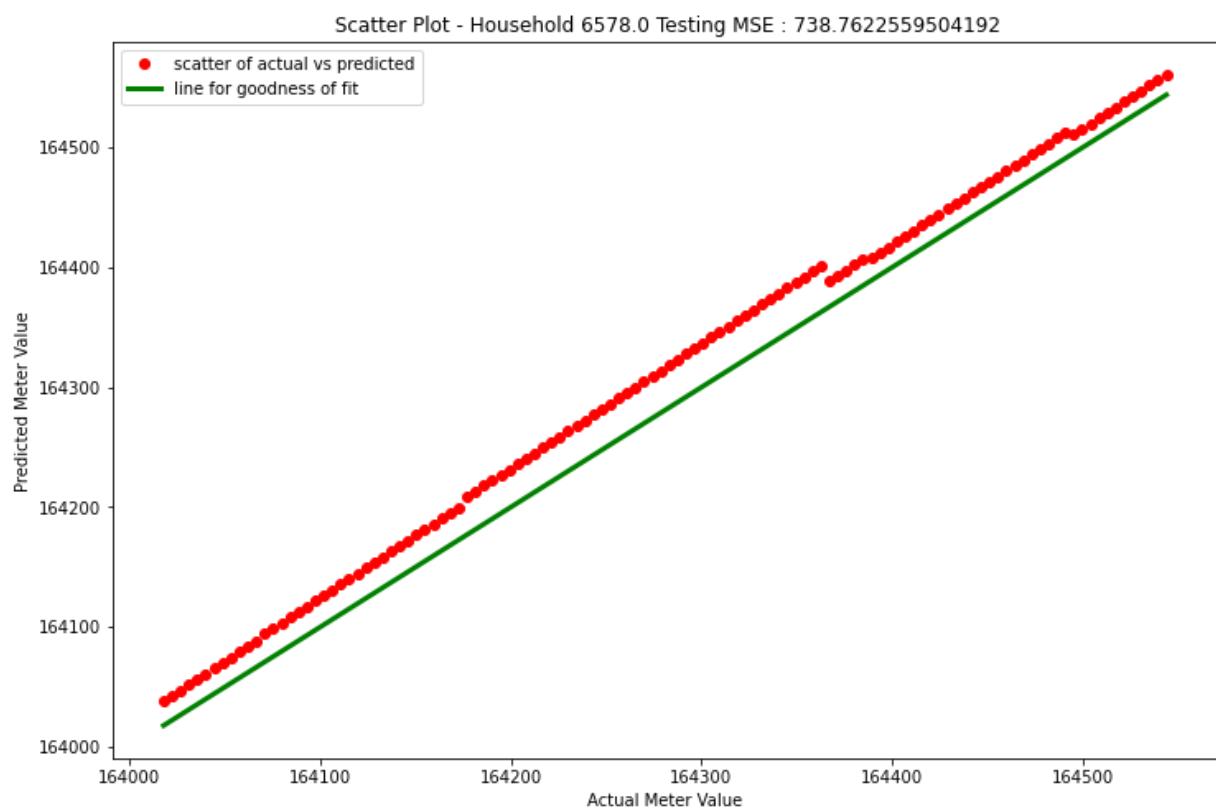
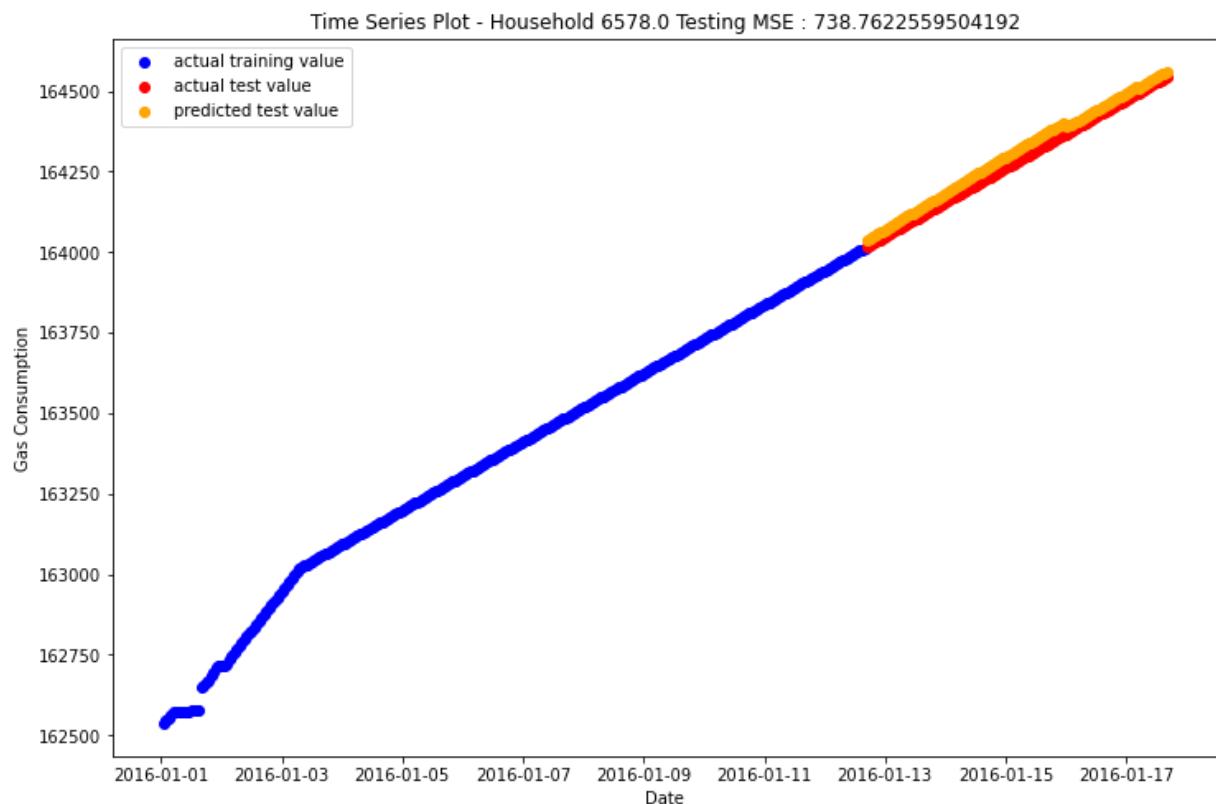
Time Series Plot - Household 6412.0 Testing MSE : 76126.47387305662

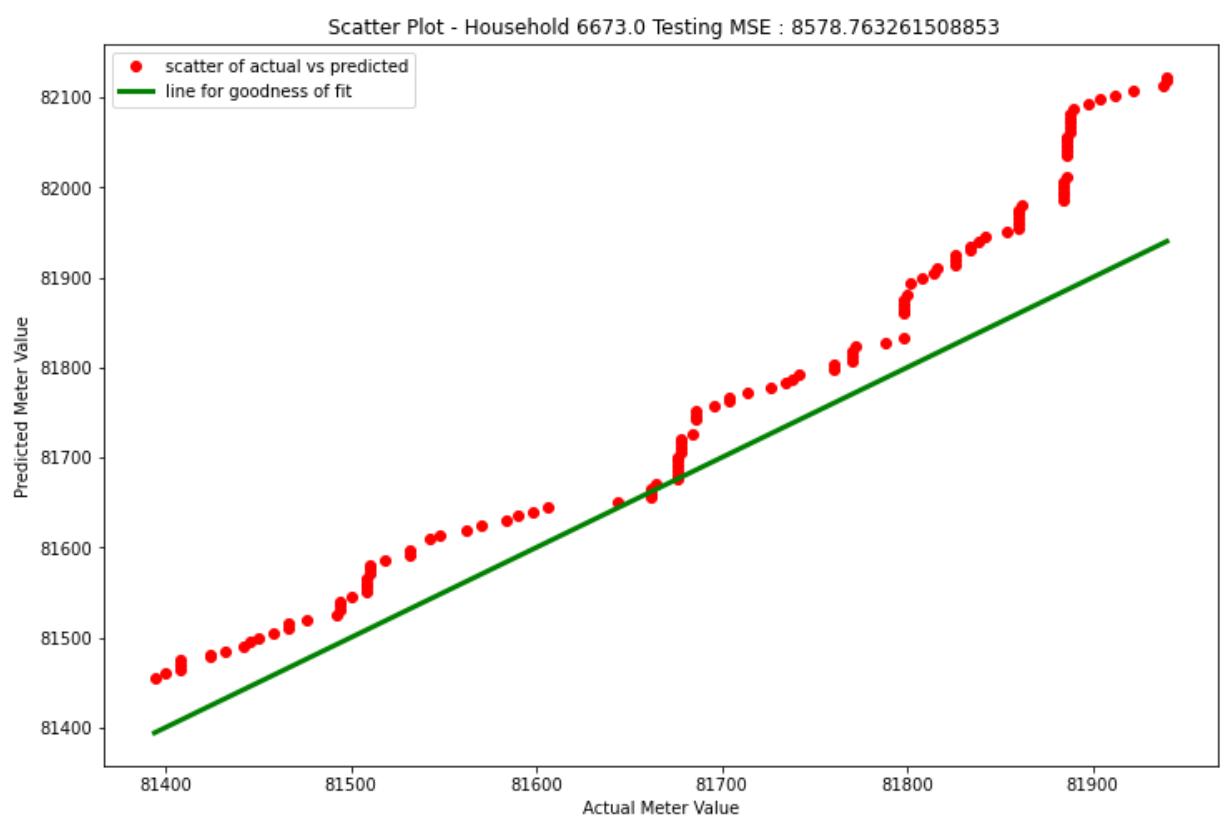
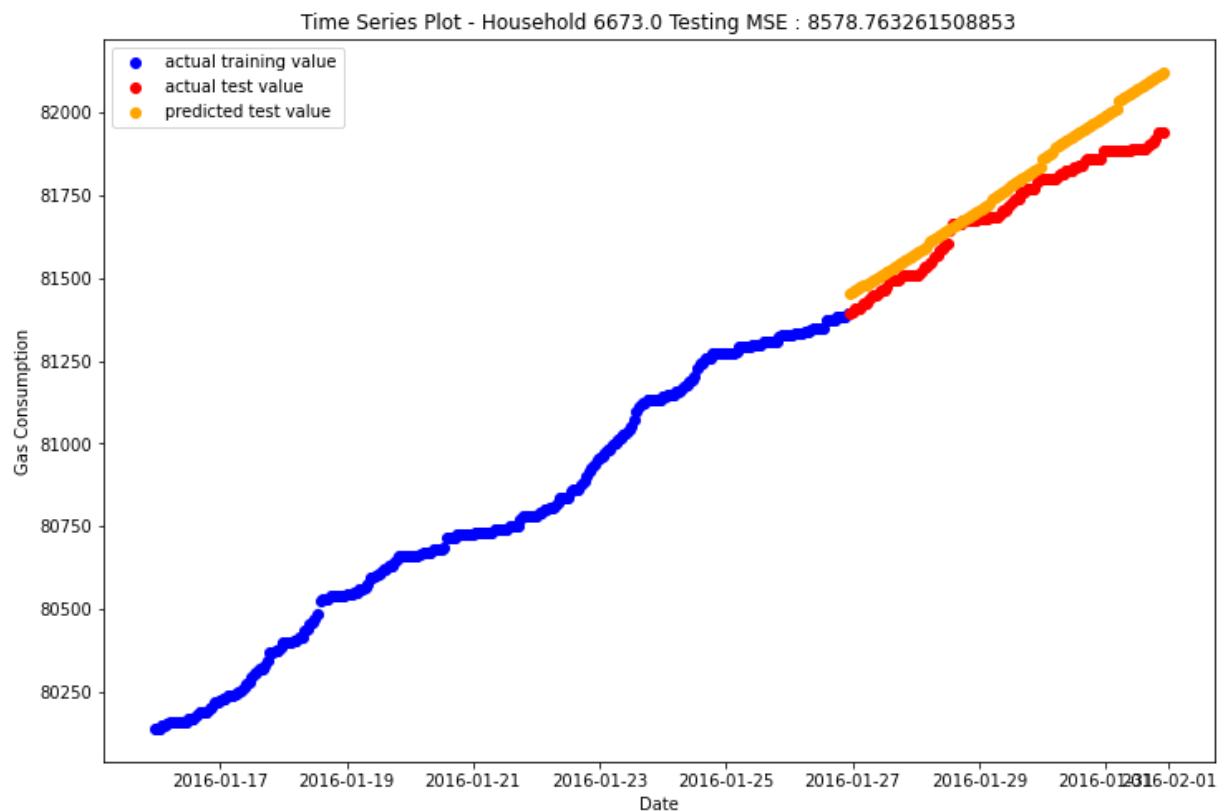


Scatter Plot - Household 6412.0 Testing MSE : 76126.47387305662

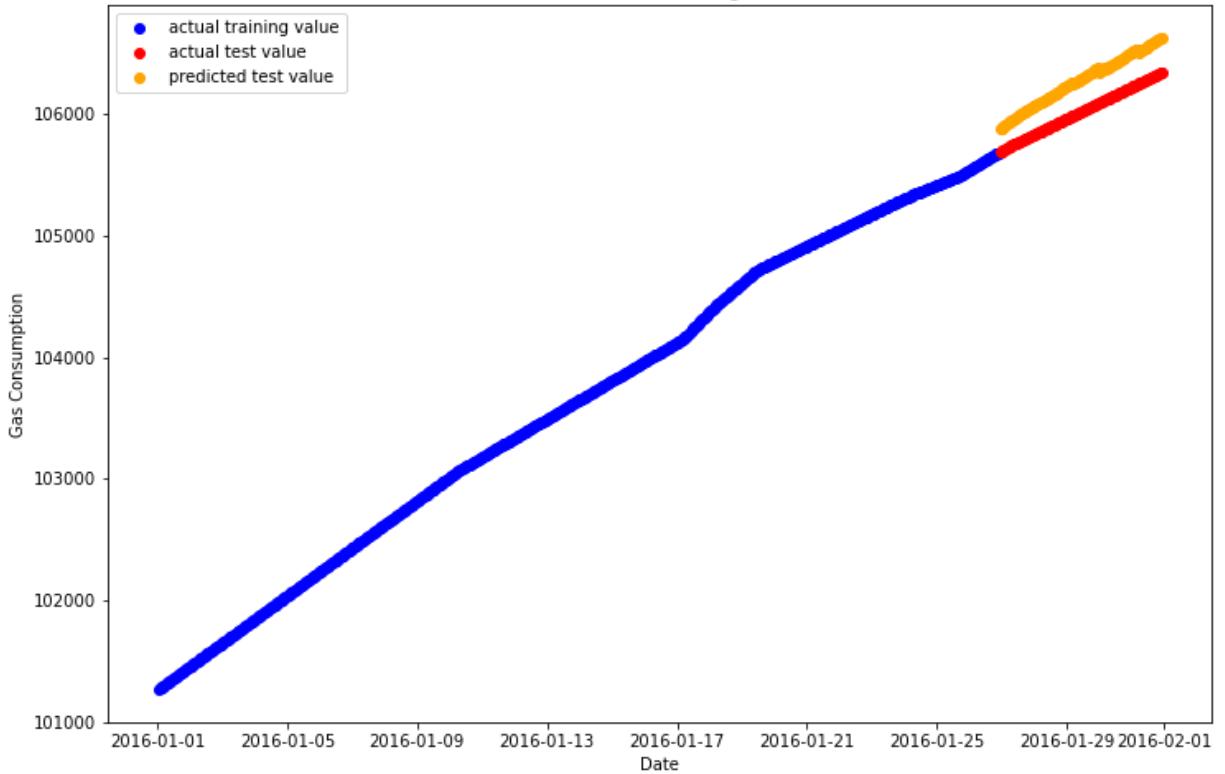




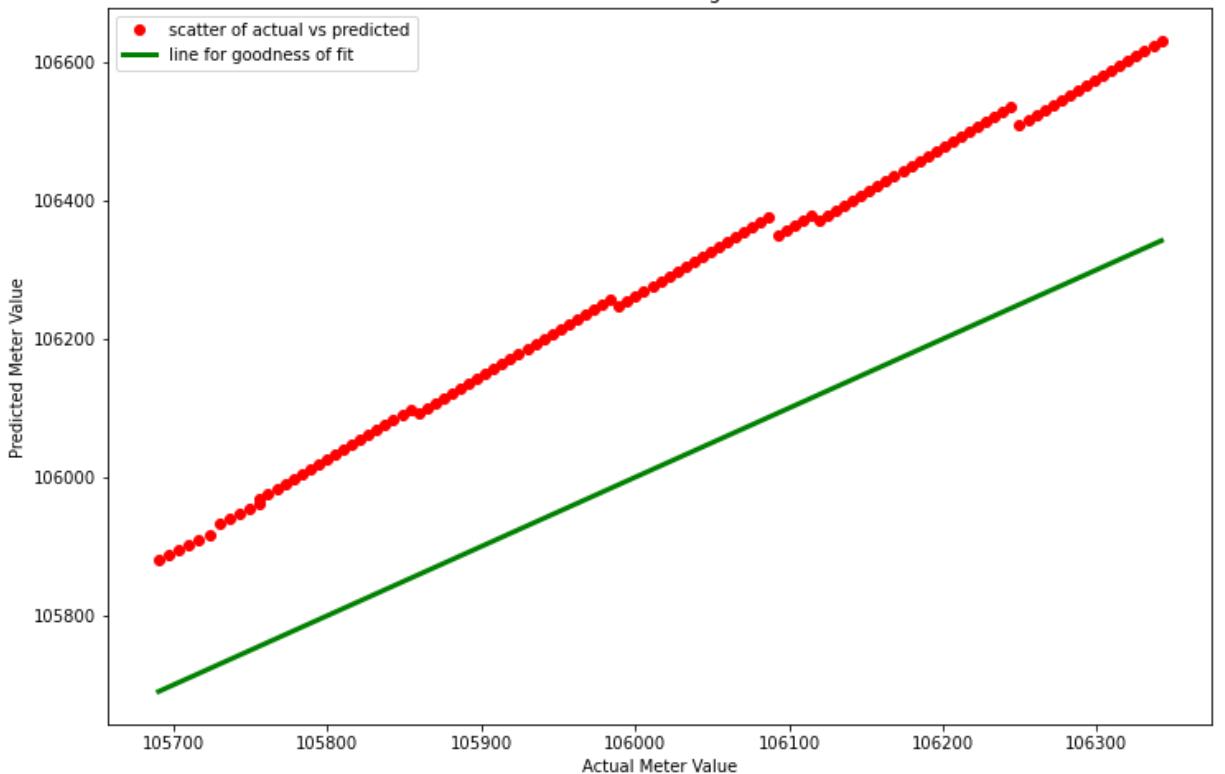


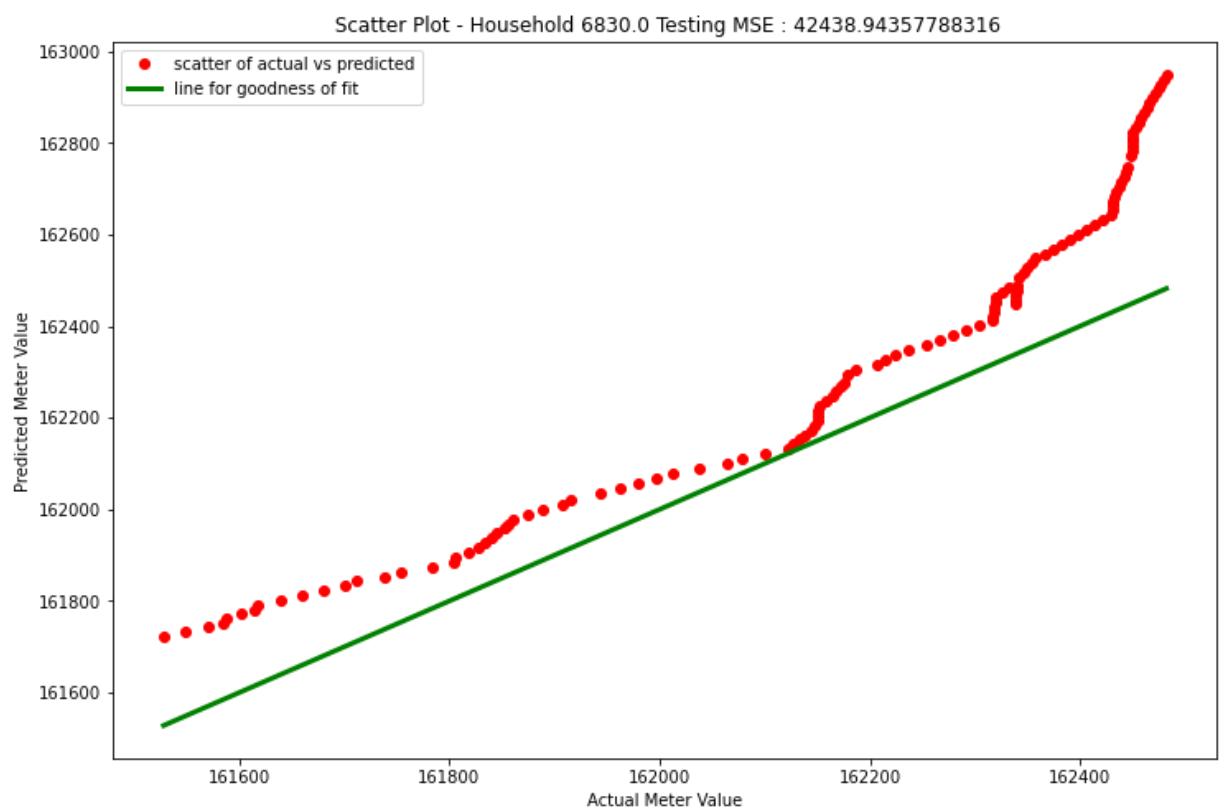
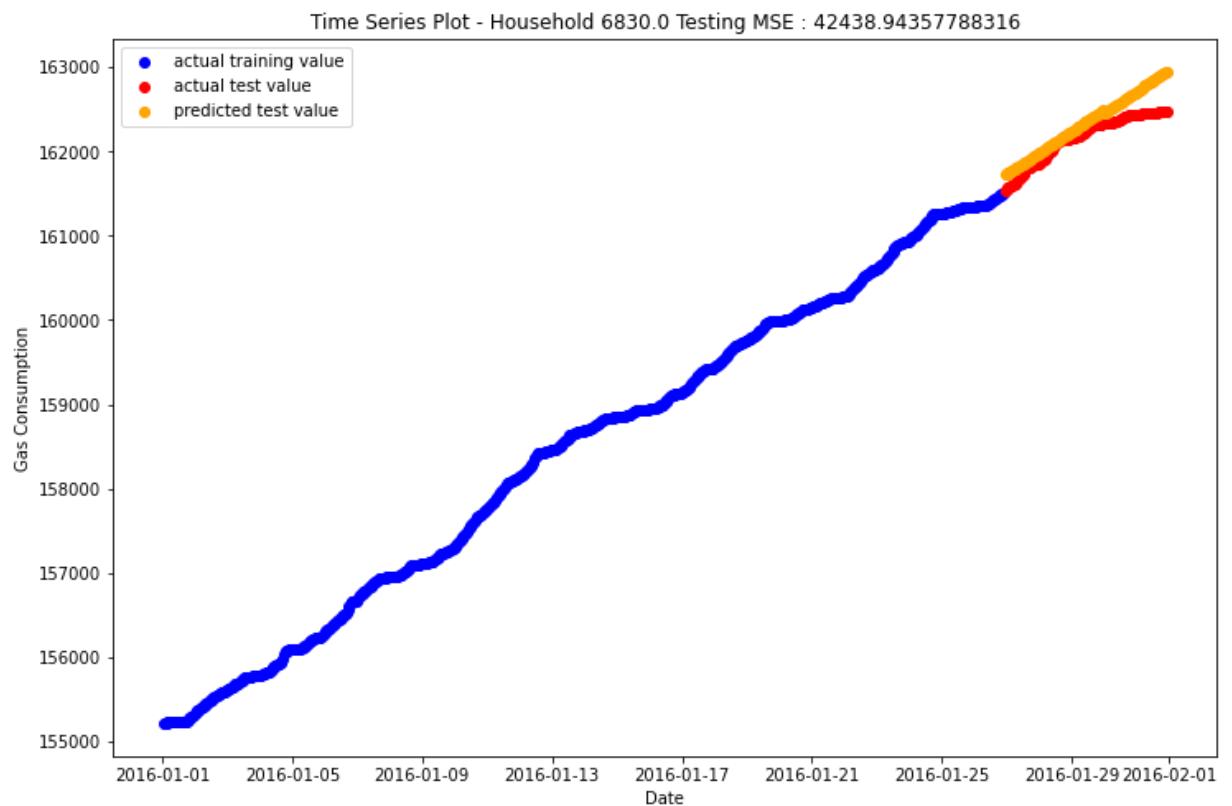


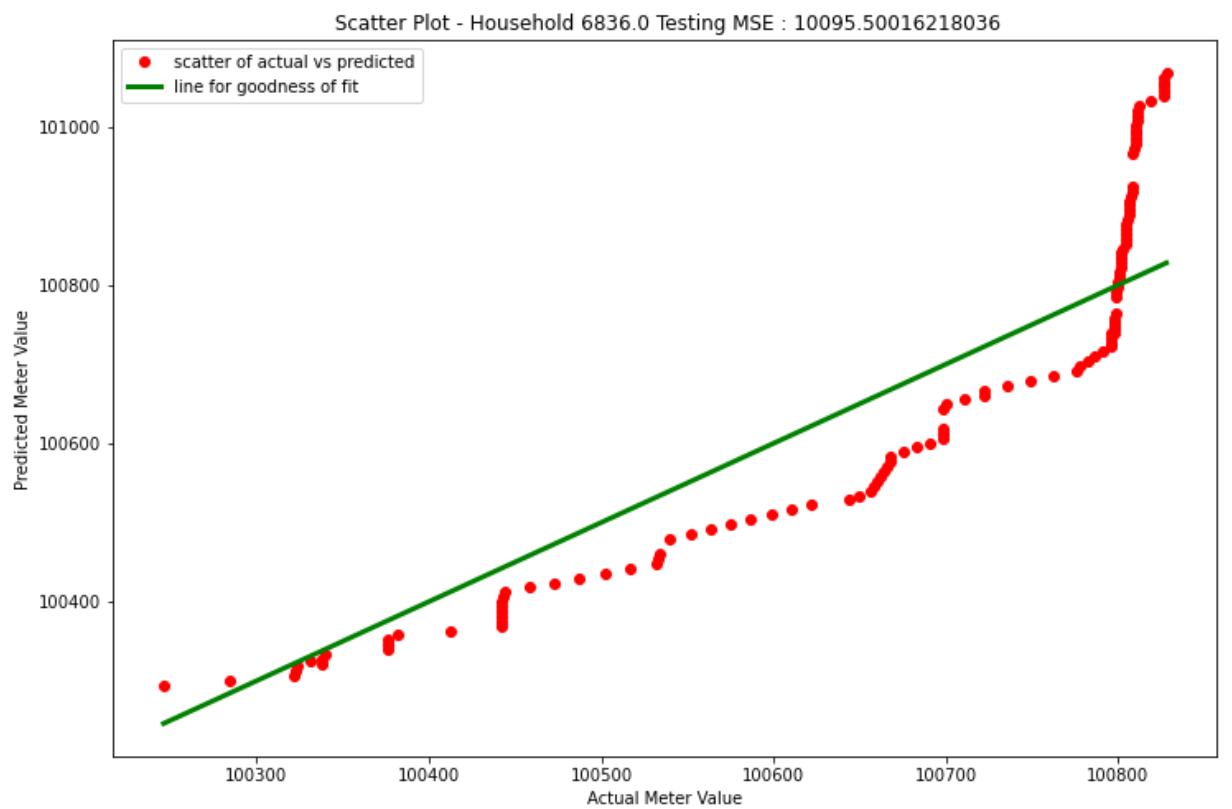
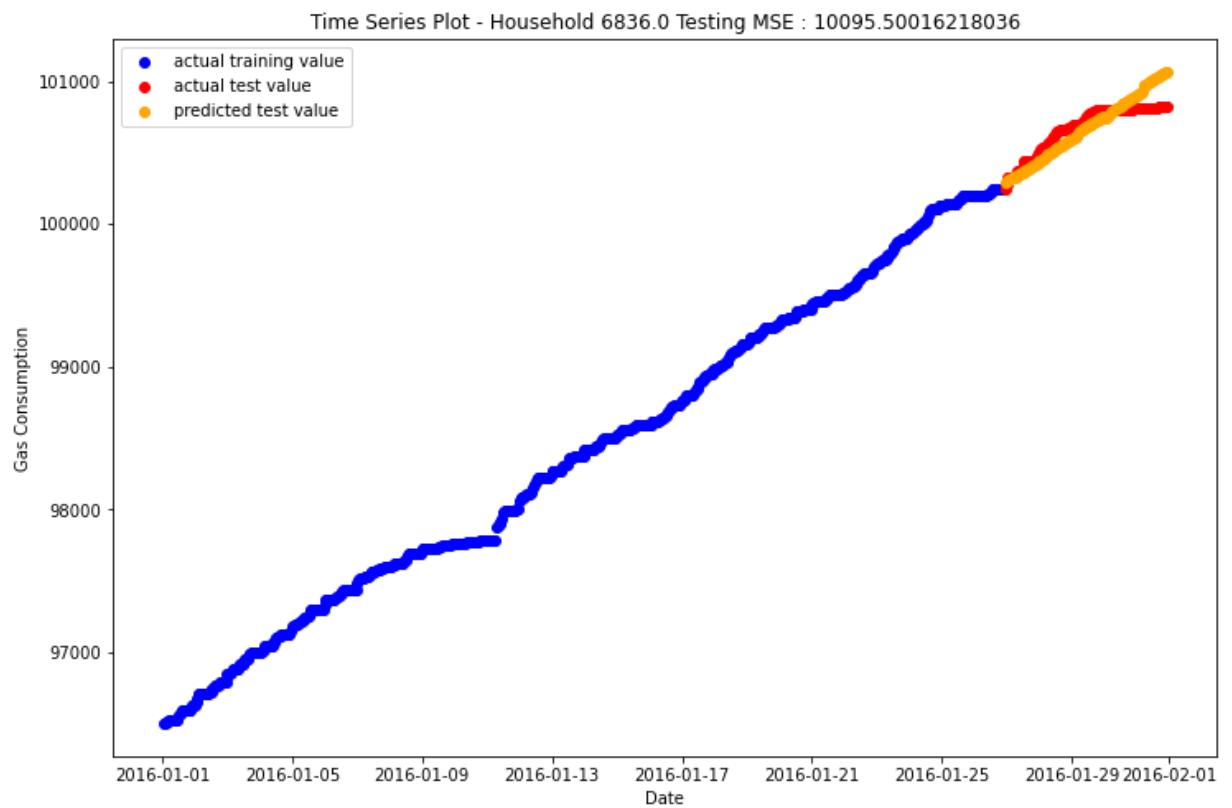
Time Series Plot - Household 6685.0 Testing MSE : 65542.90306459372

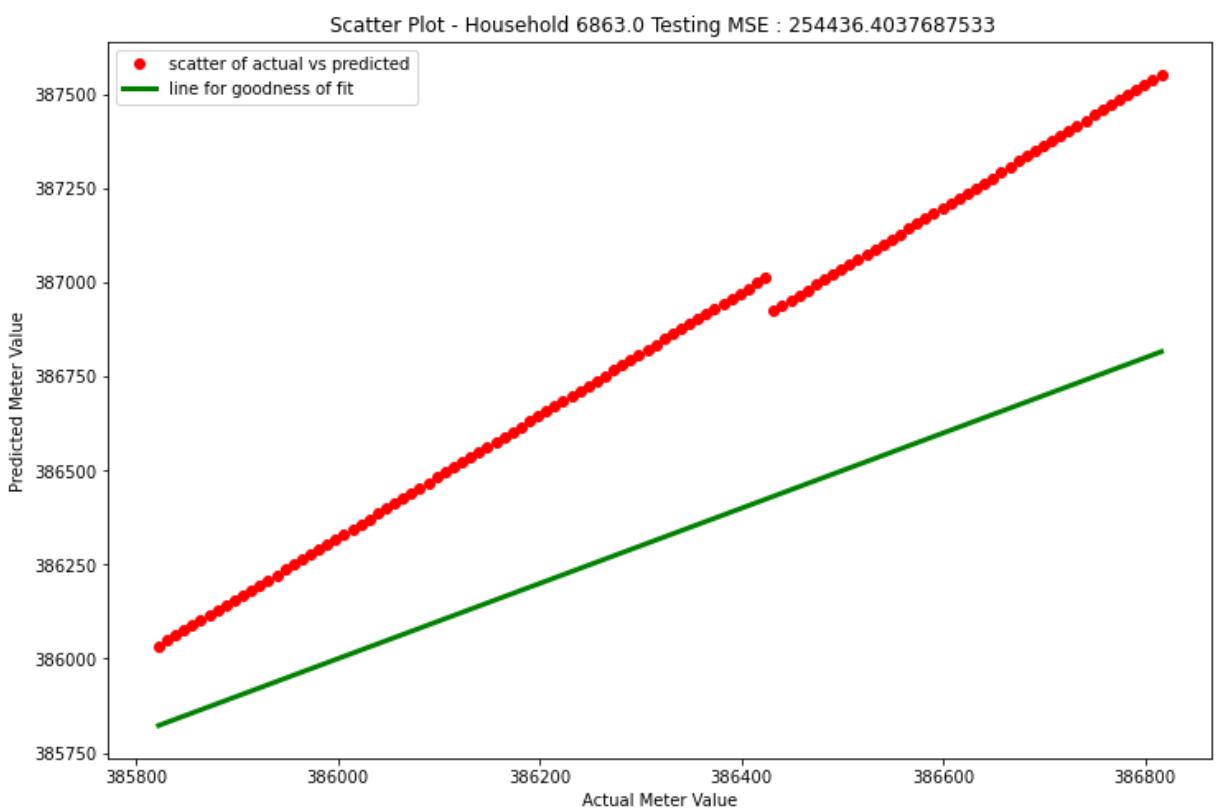
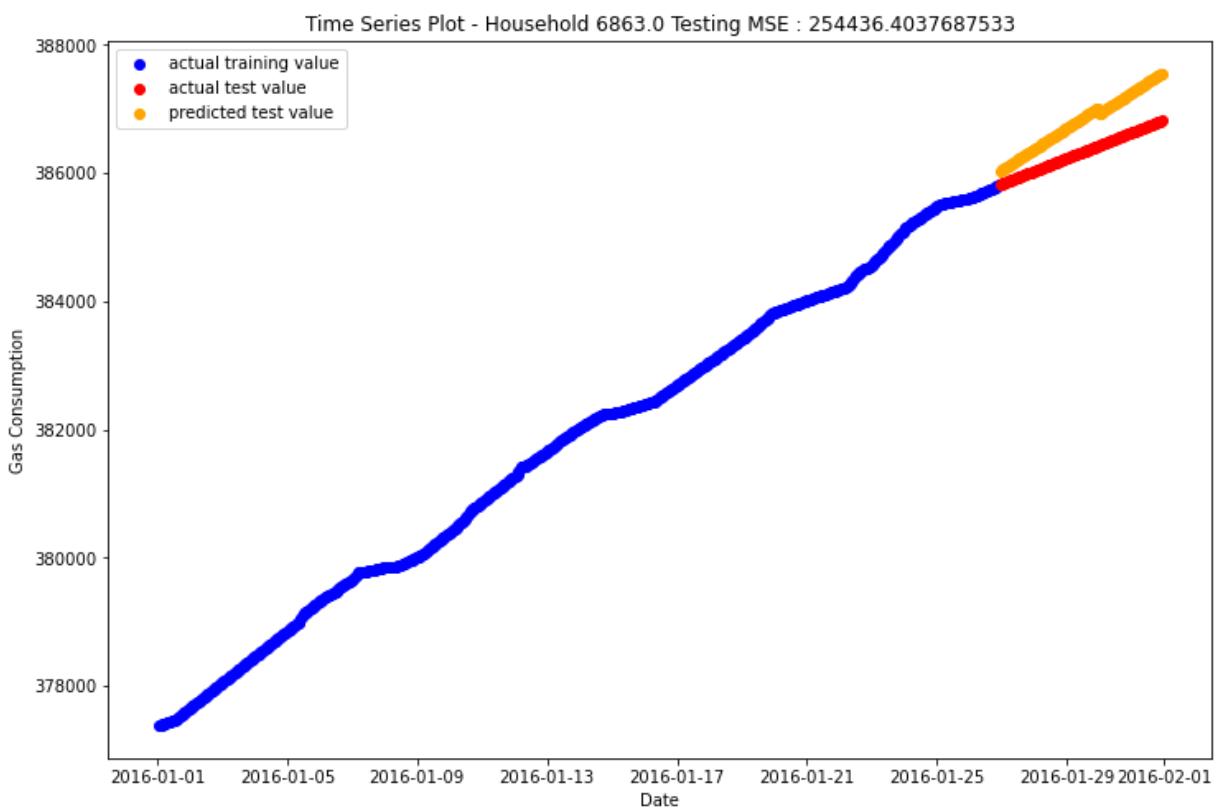


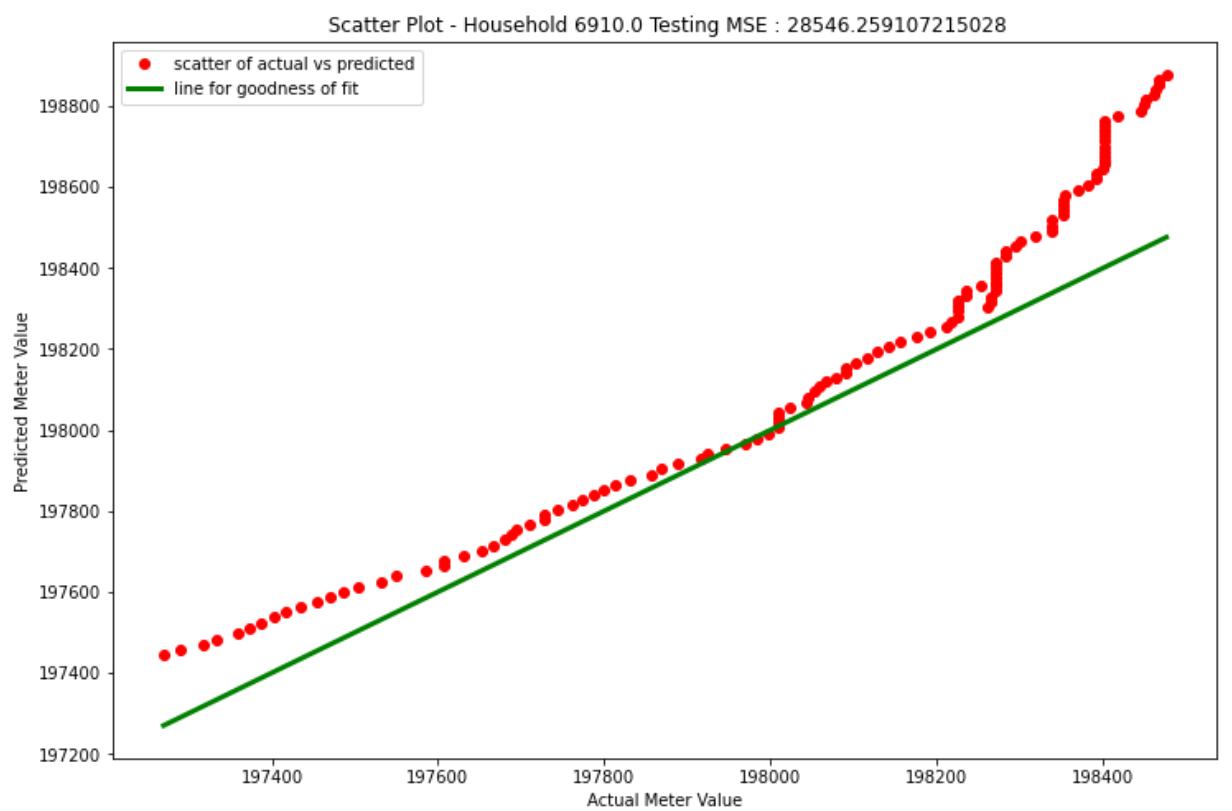
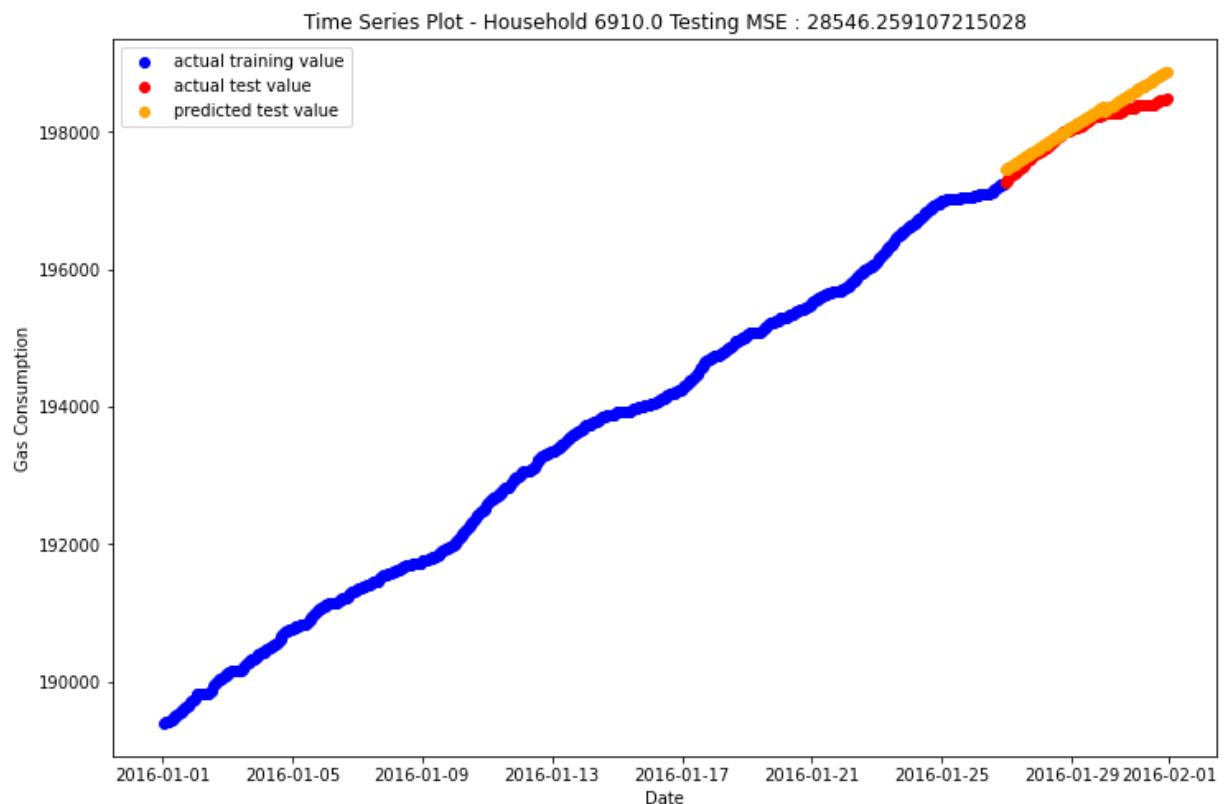
Scatter Plot - Household 6685.0 Testing MSE : 65542.90306459372

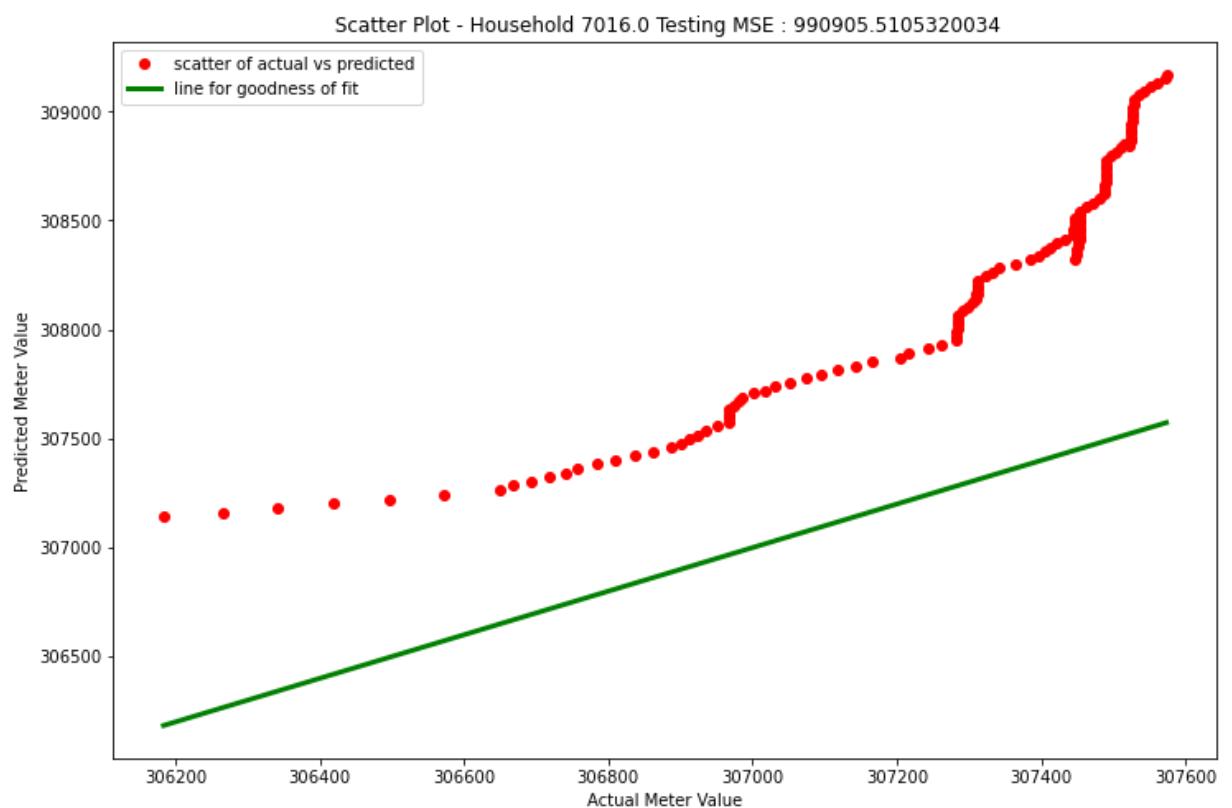
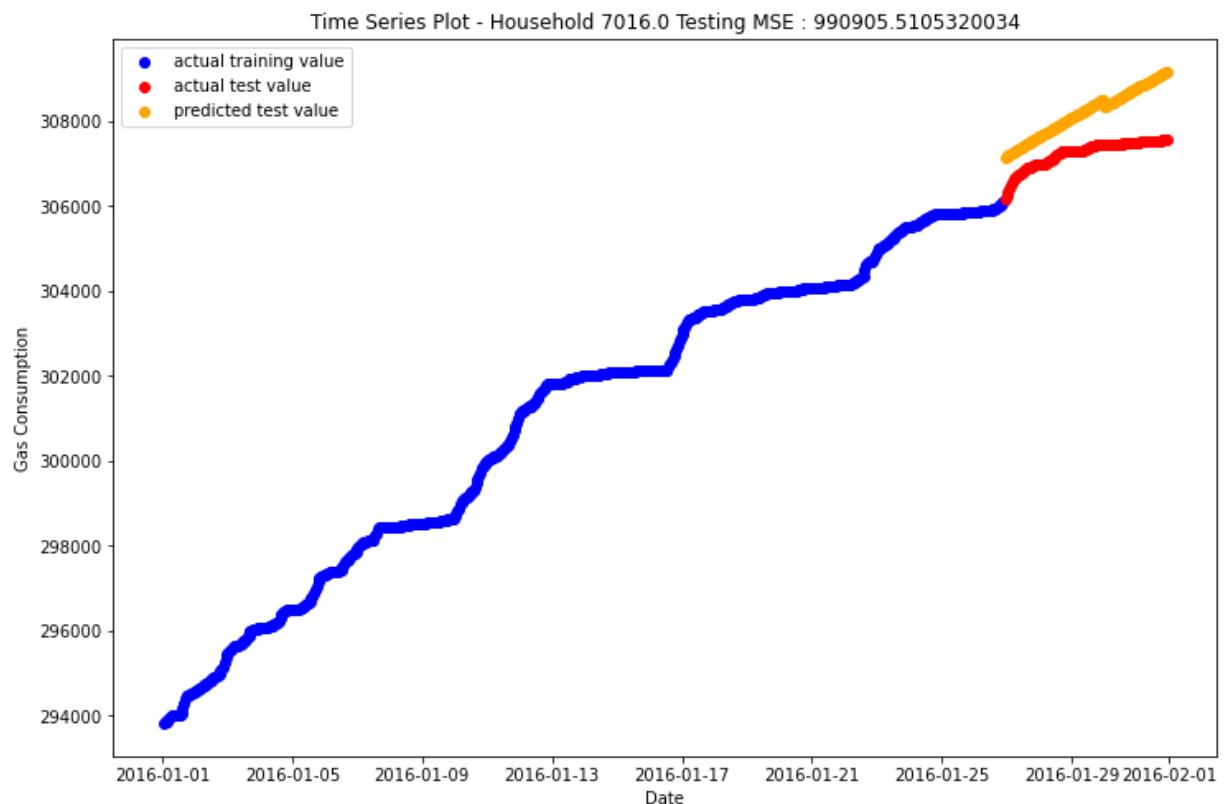


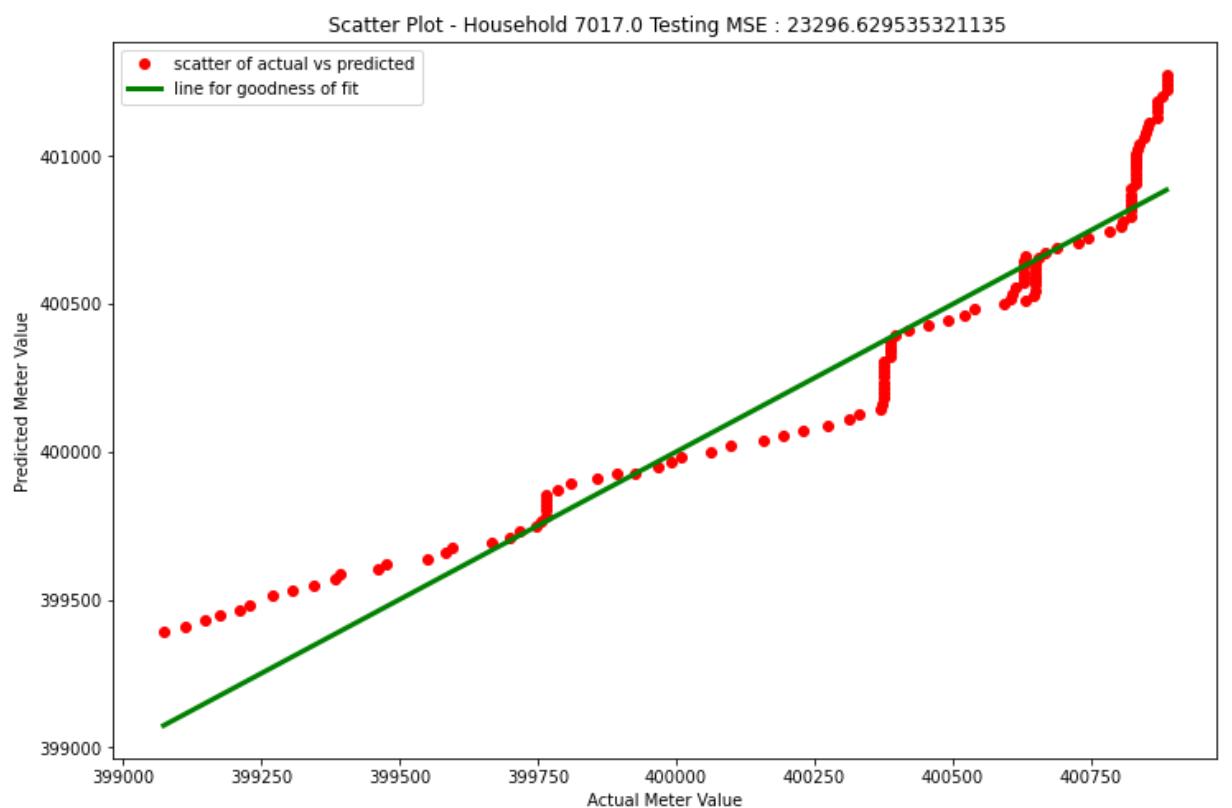
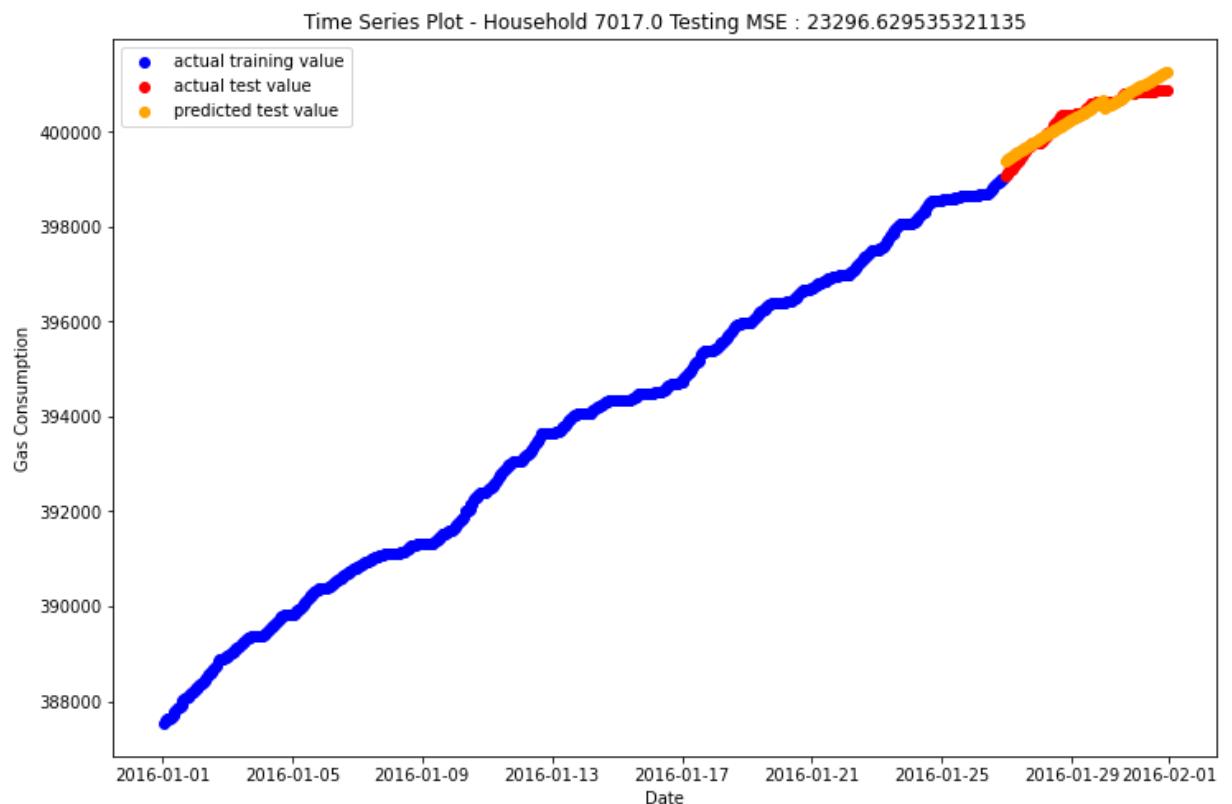


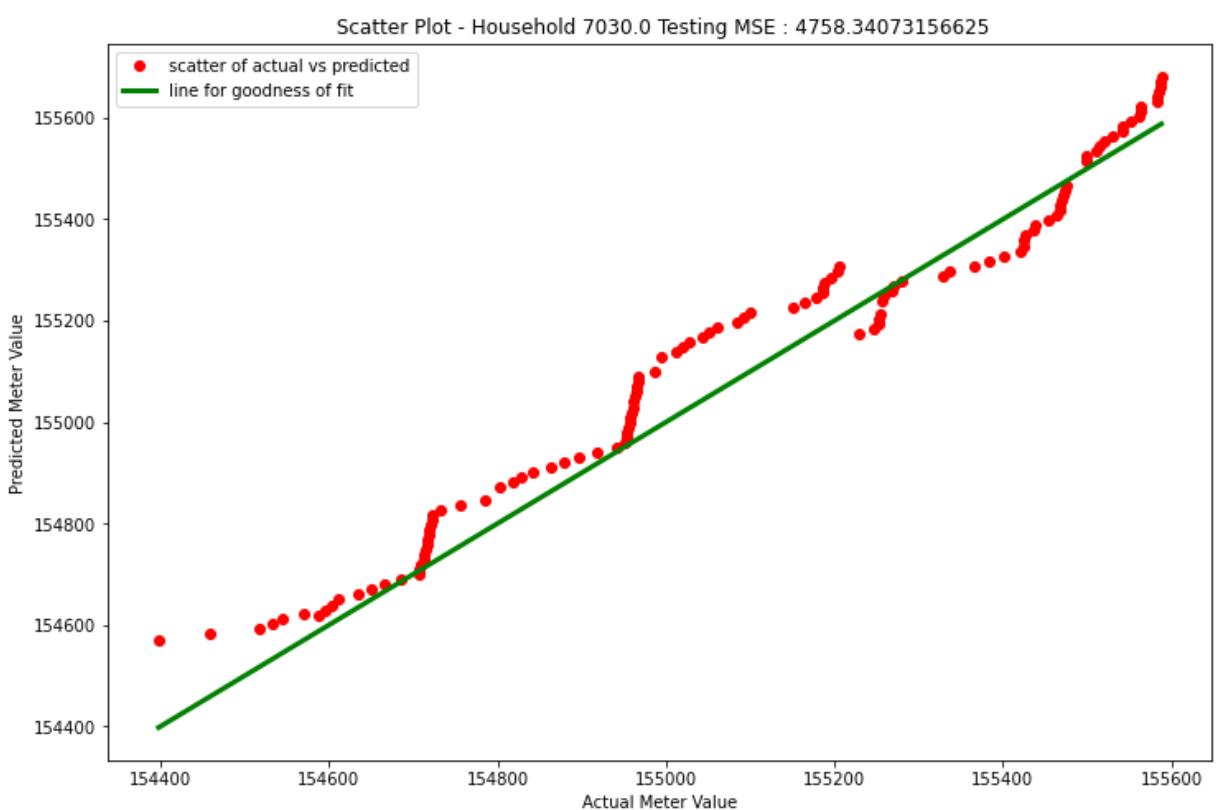
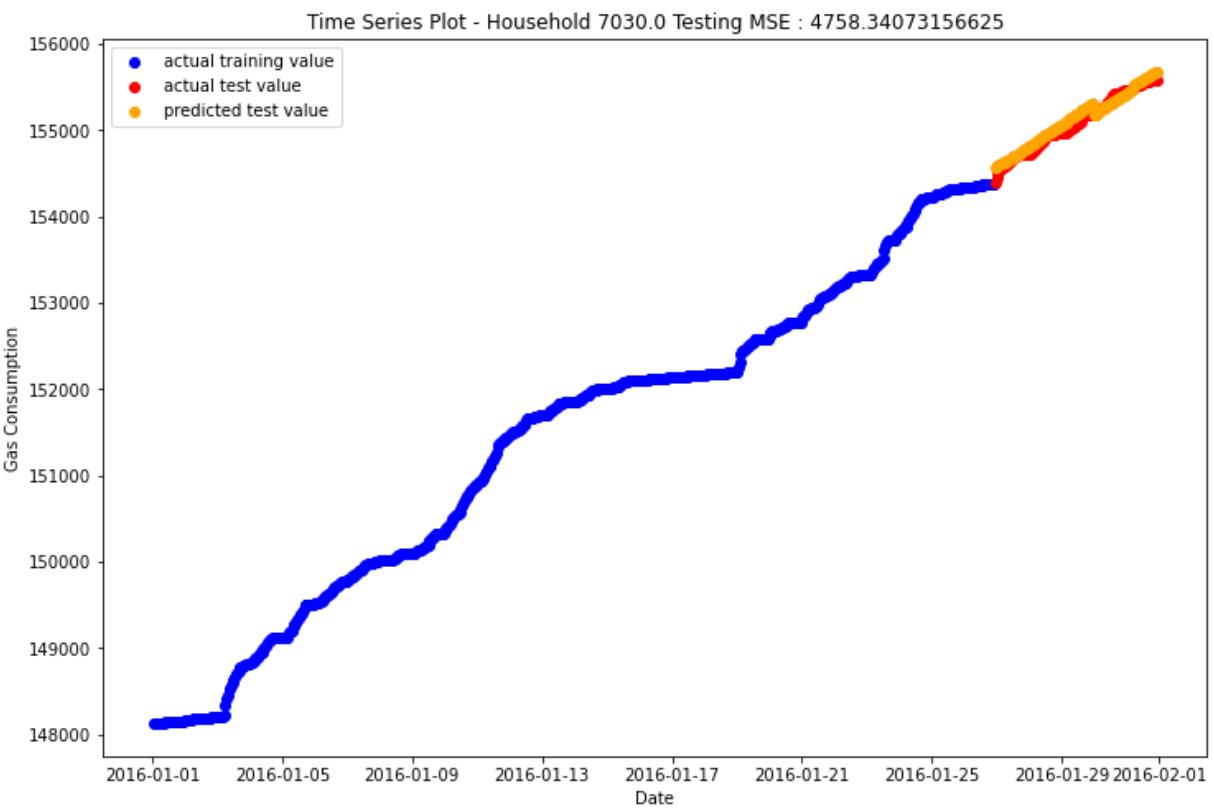


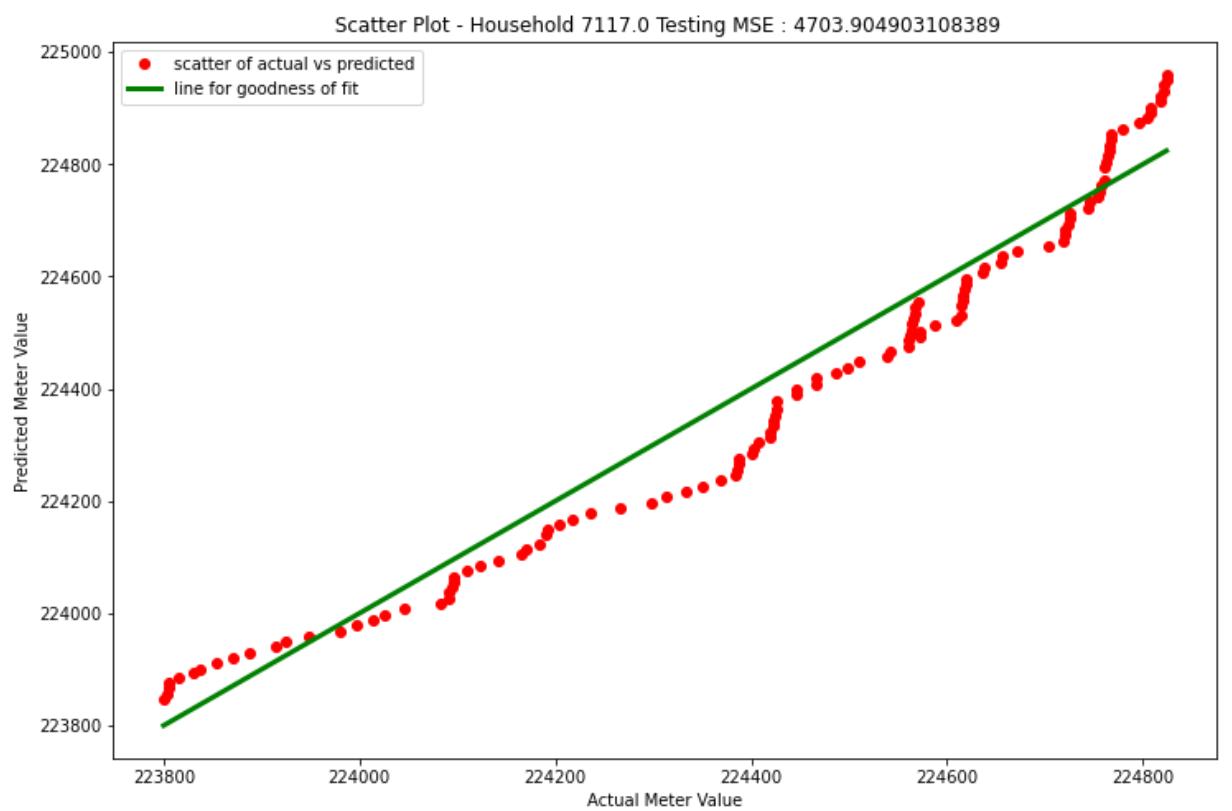
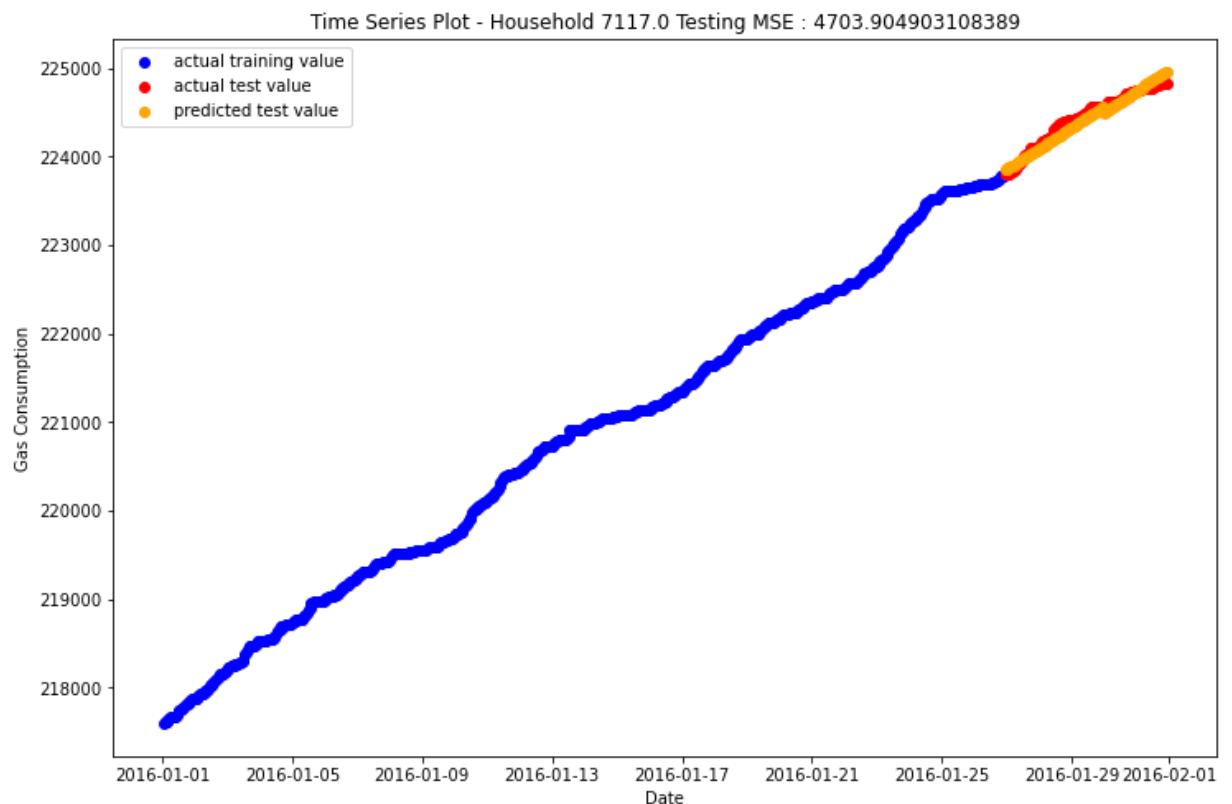




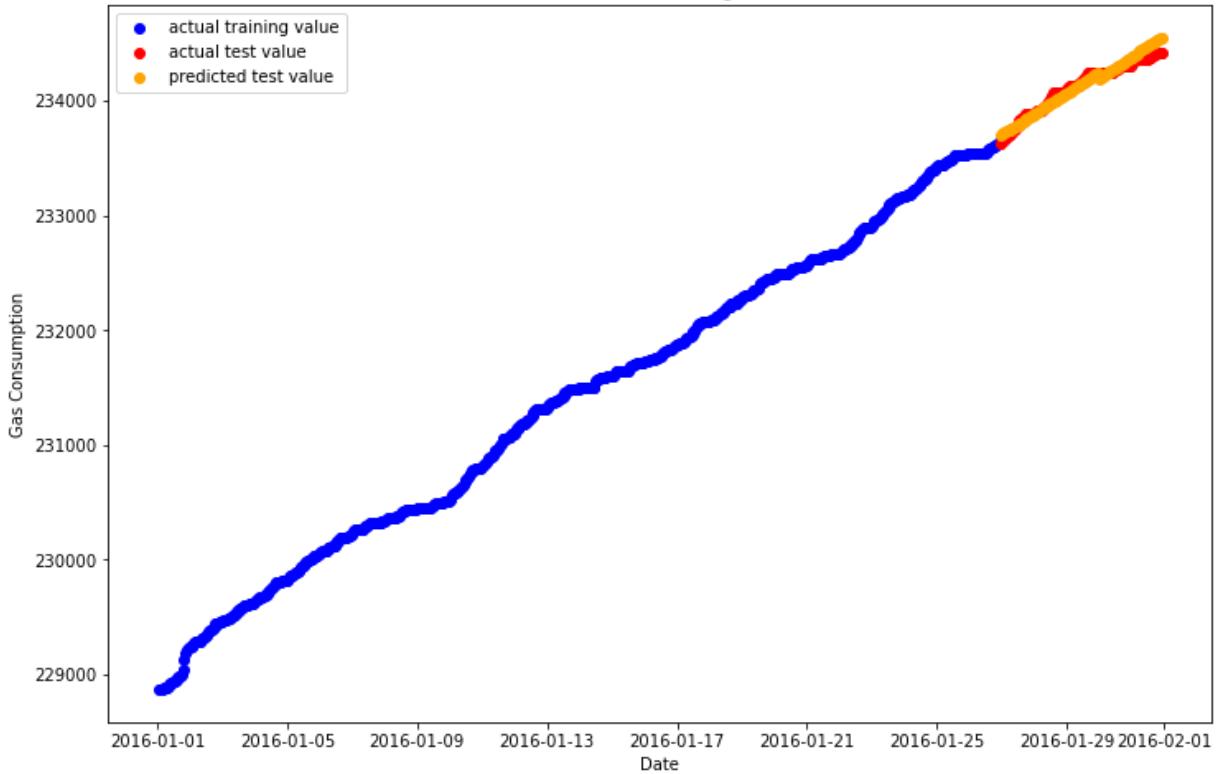




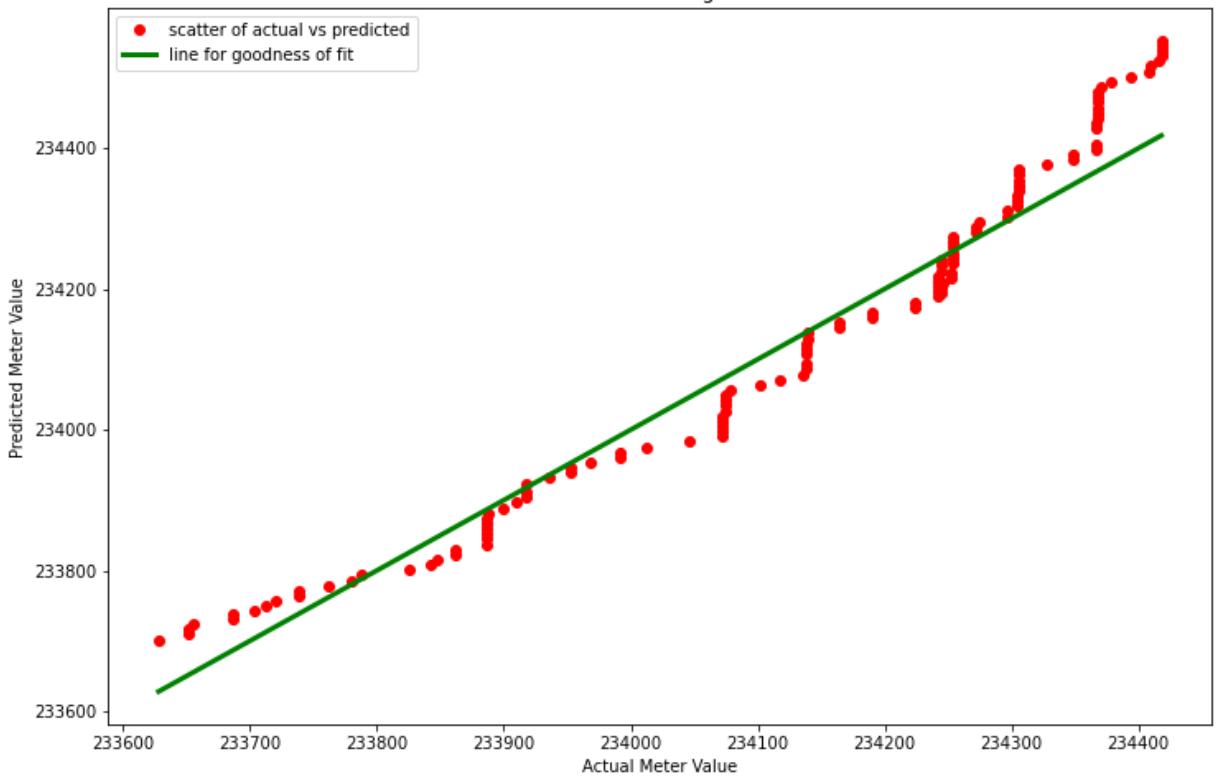


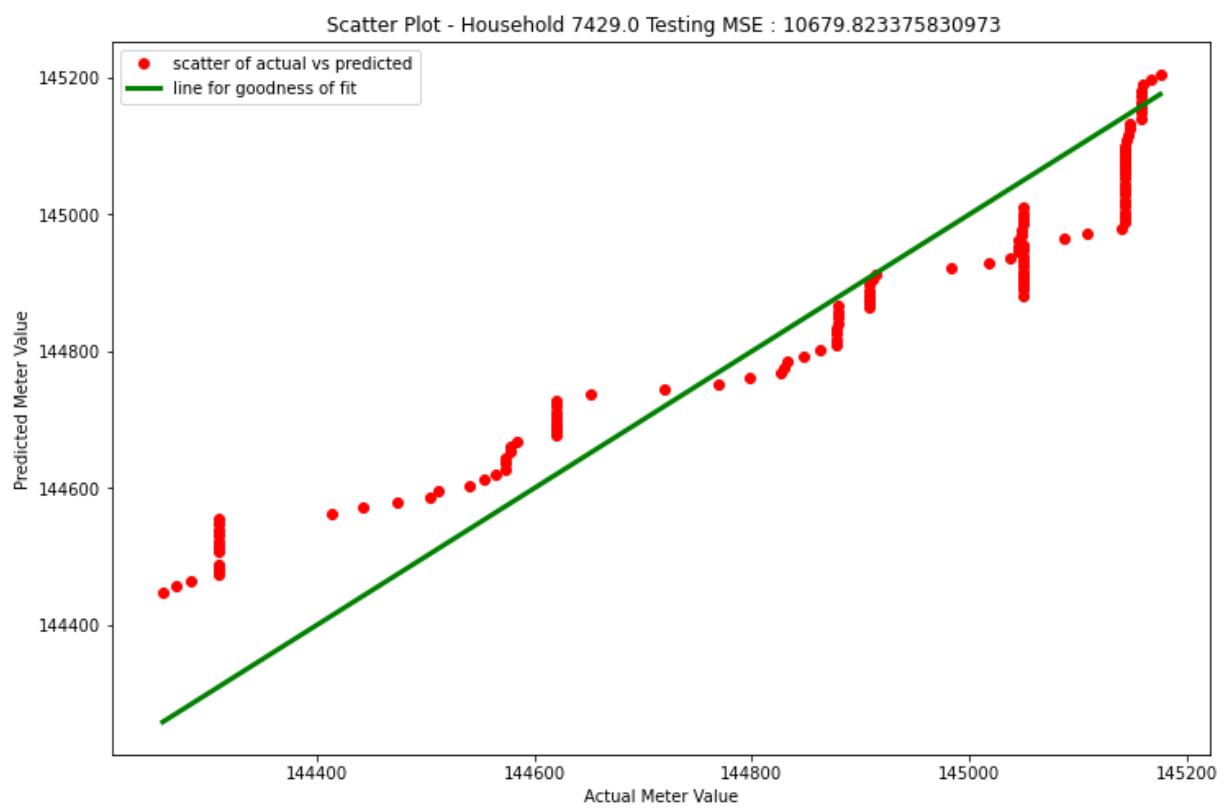
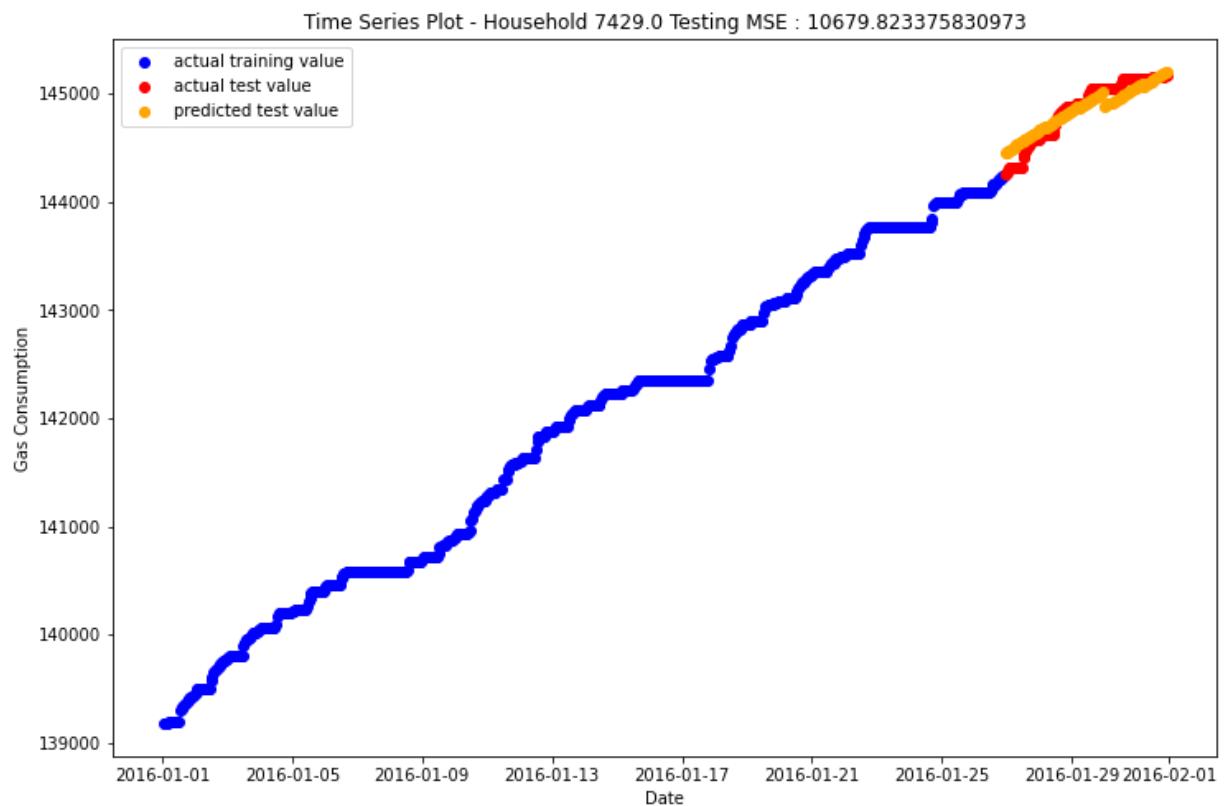


Time Series Plot - Household 7287.0 Testing MSE : 2779.1273787409364

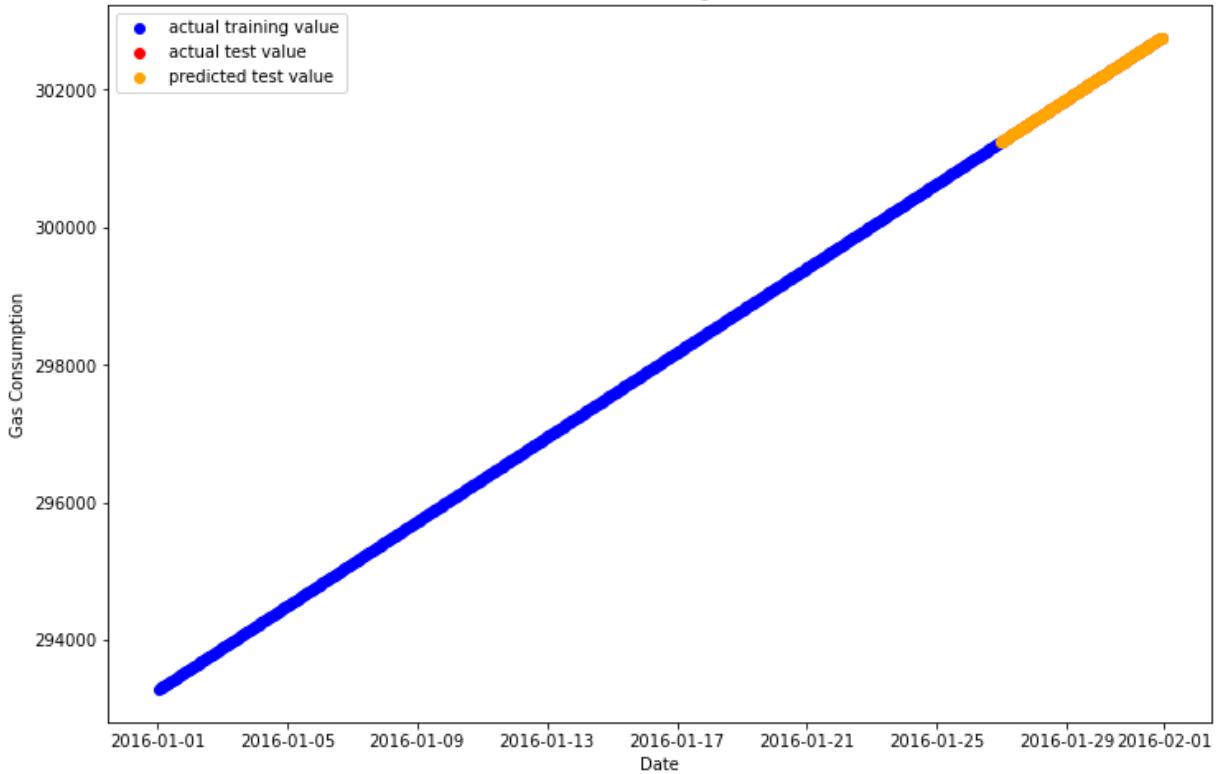


Scatter Plot - Household 7287.0 Testing MSE : 2779.1273787409364

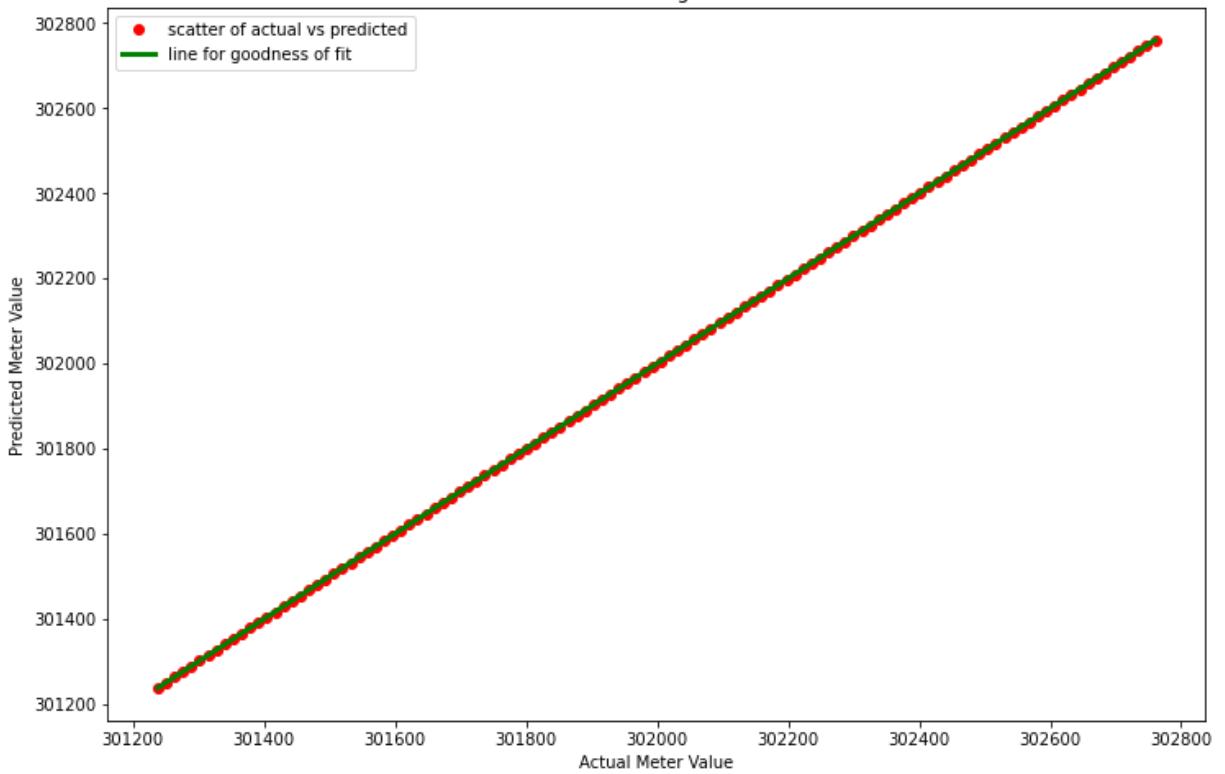


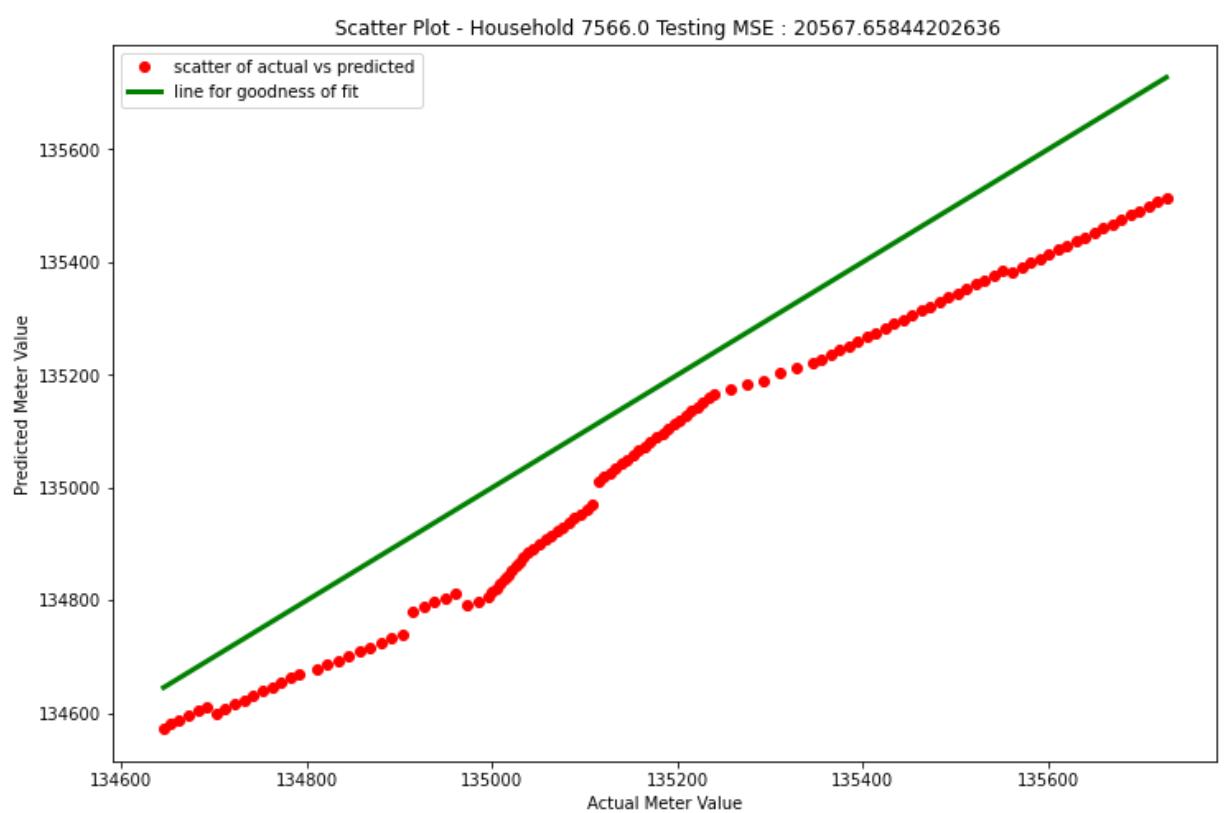
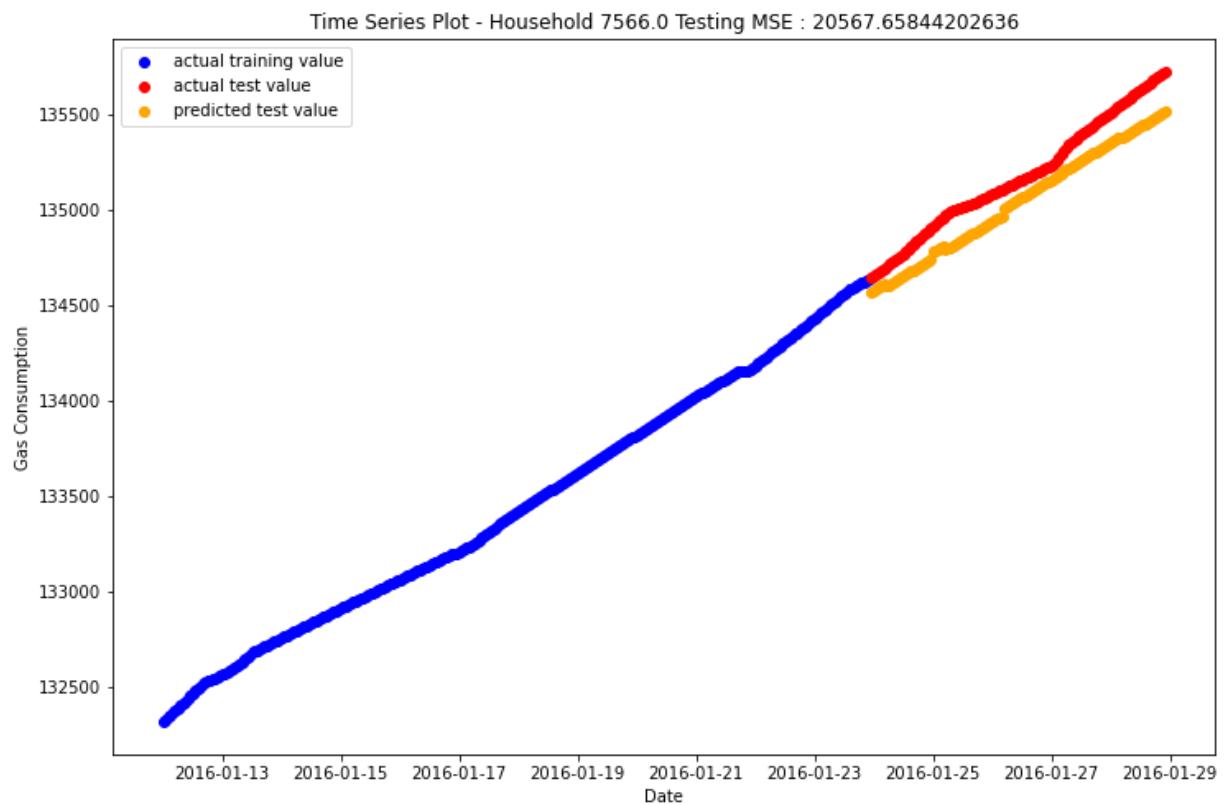


Time Series Plot - Household 7460.0 Testing MSE : 1.0164395367051604e-21

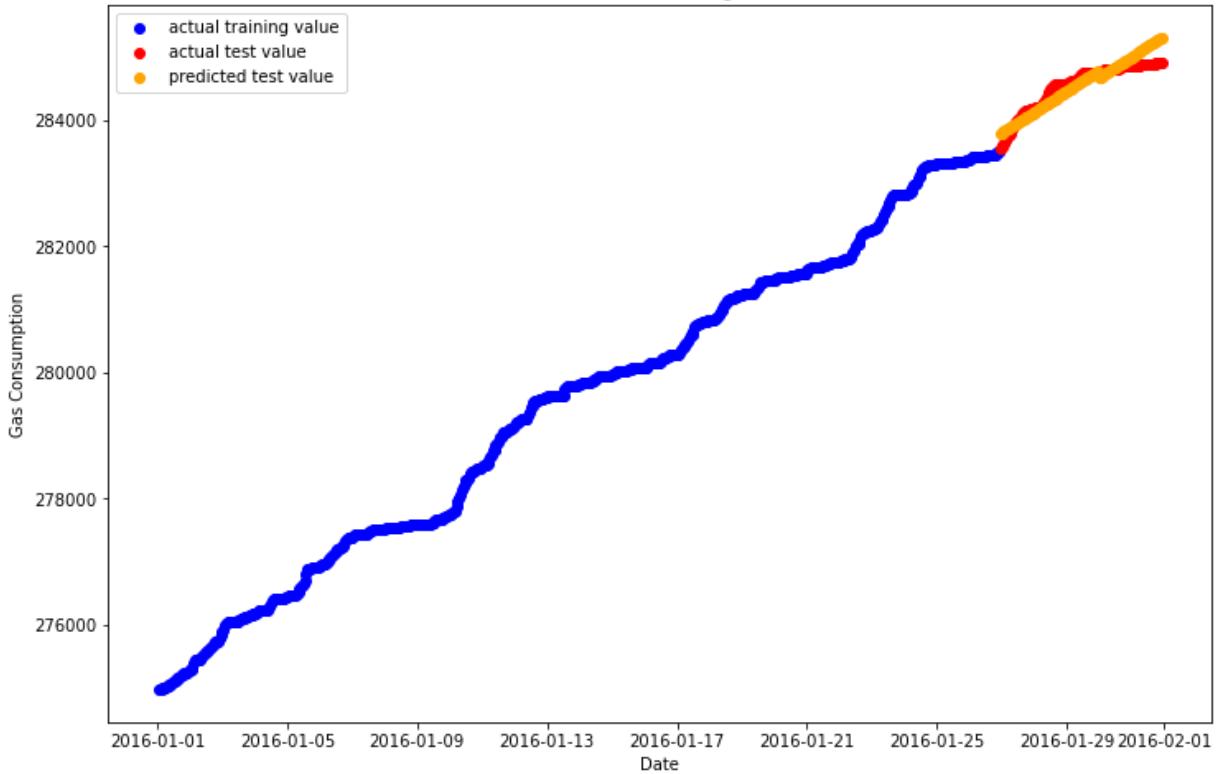


Scatter Plot - Household 7460.0 Testing MSE : 1.0164395367051604e-21

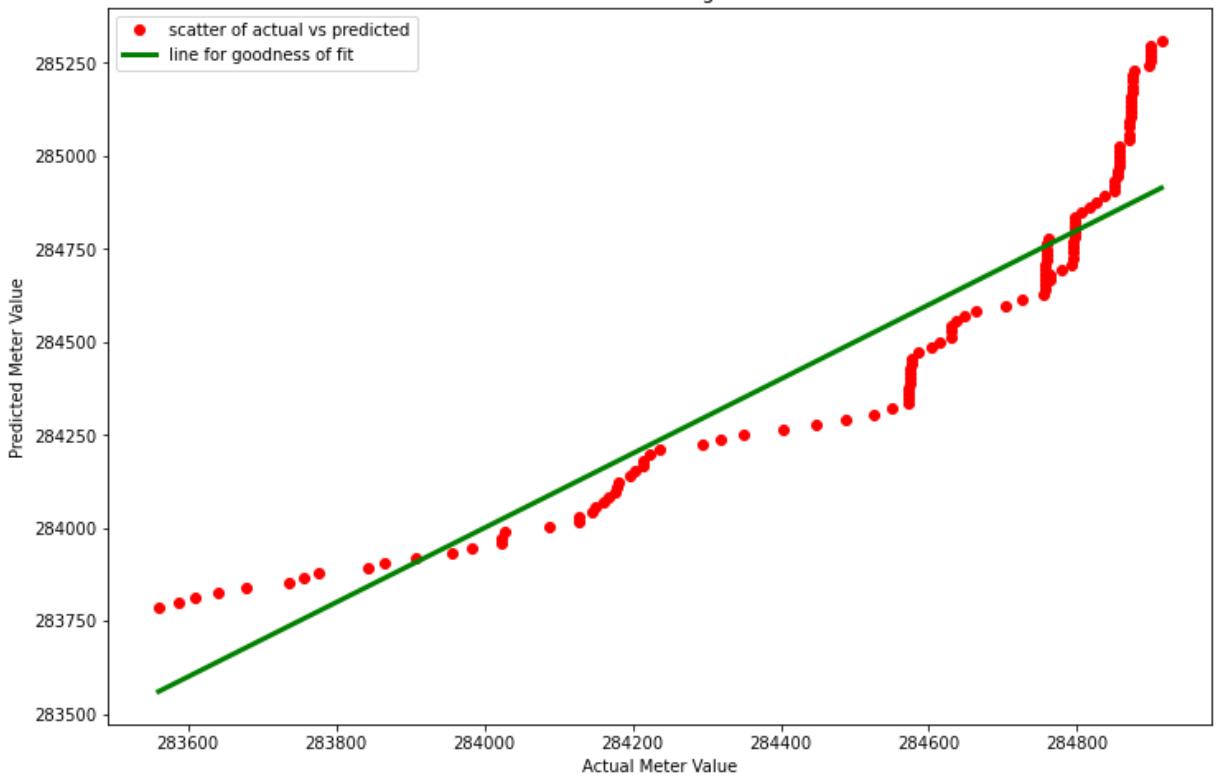


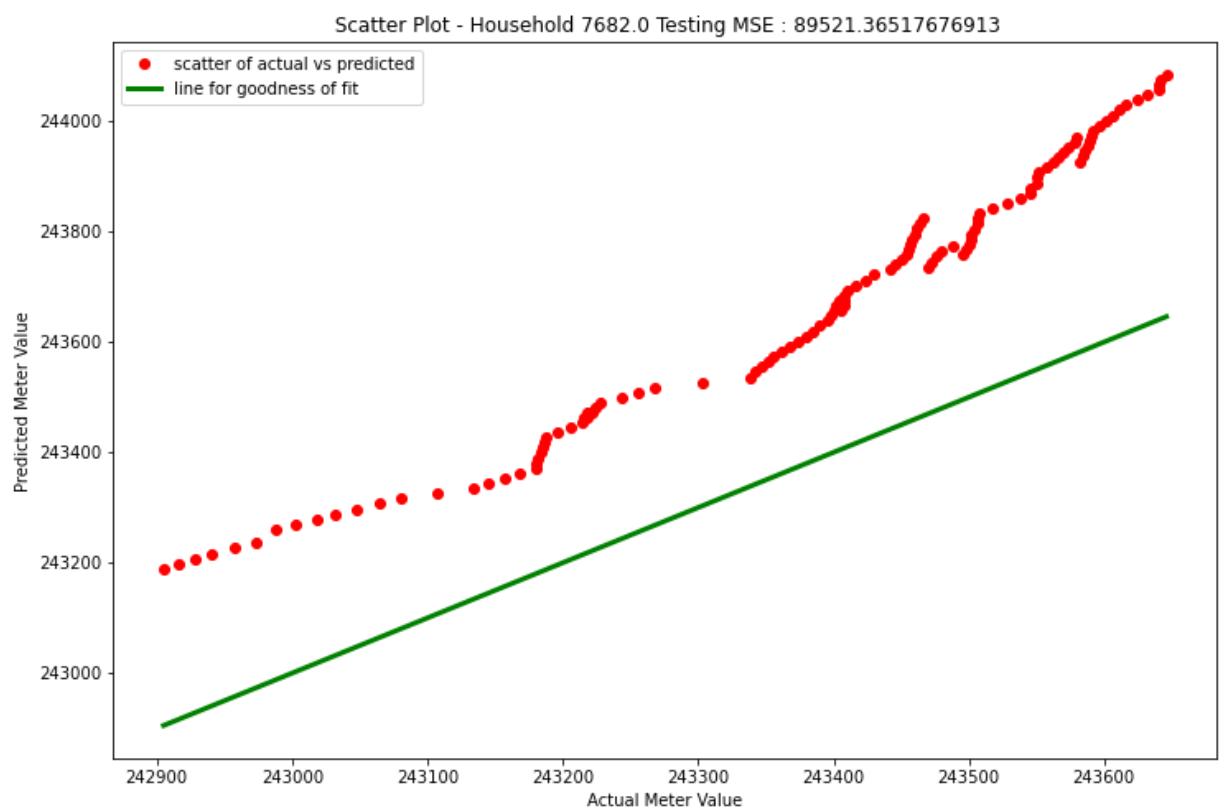
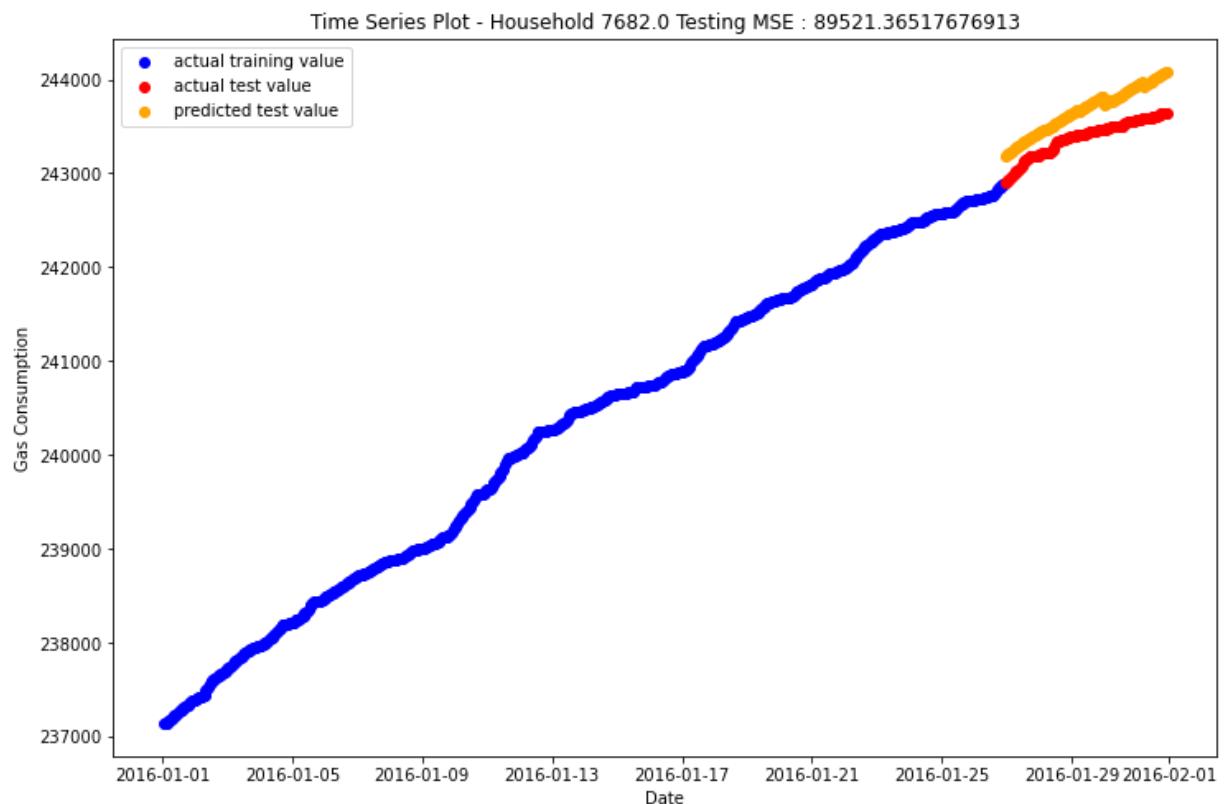


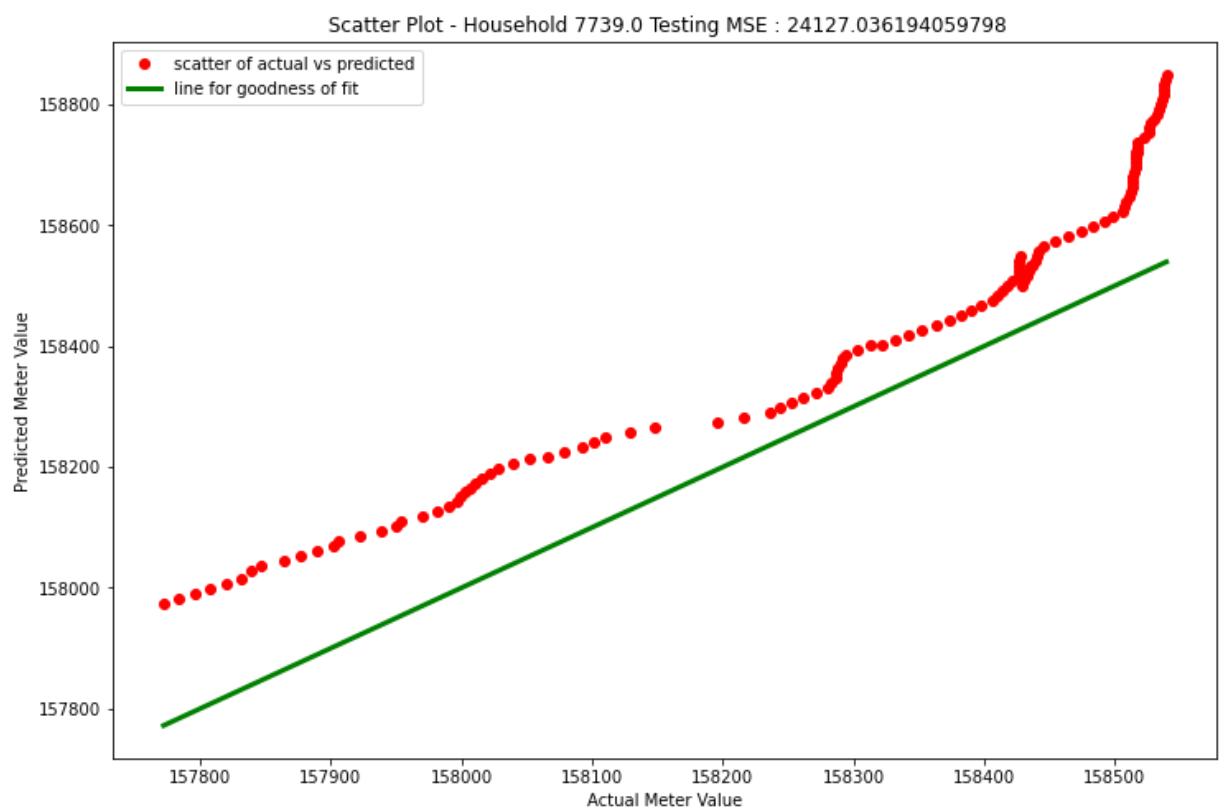
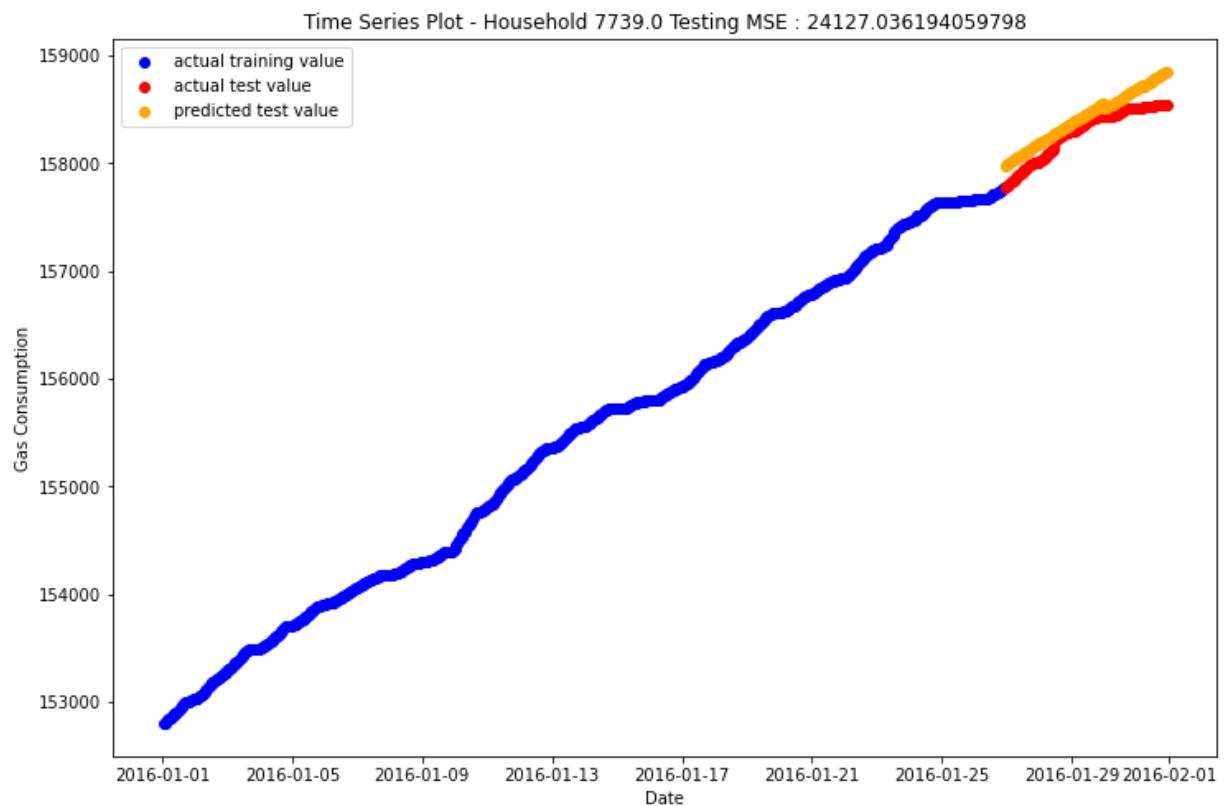
Time Series Plot - Household 7674.0 Testing MSE : 26072.205941722284



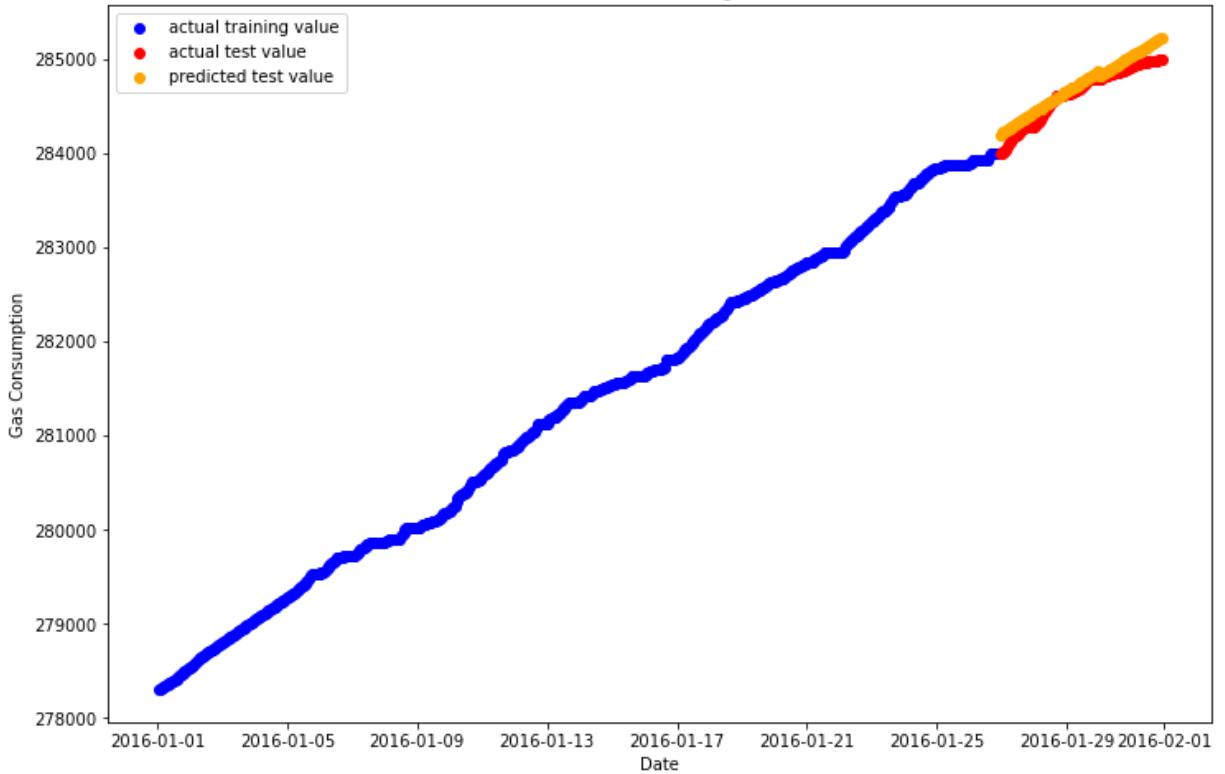
Scatter Plot - Household 7674.0 Testing MSE : 26072.205941722284



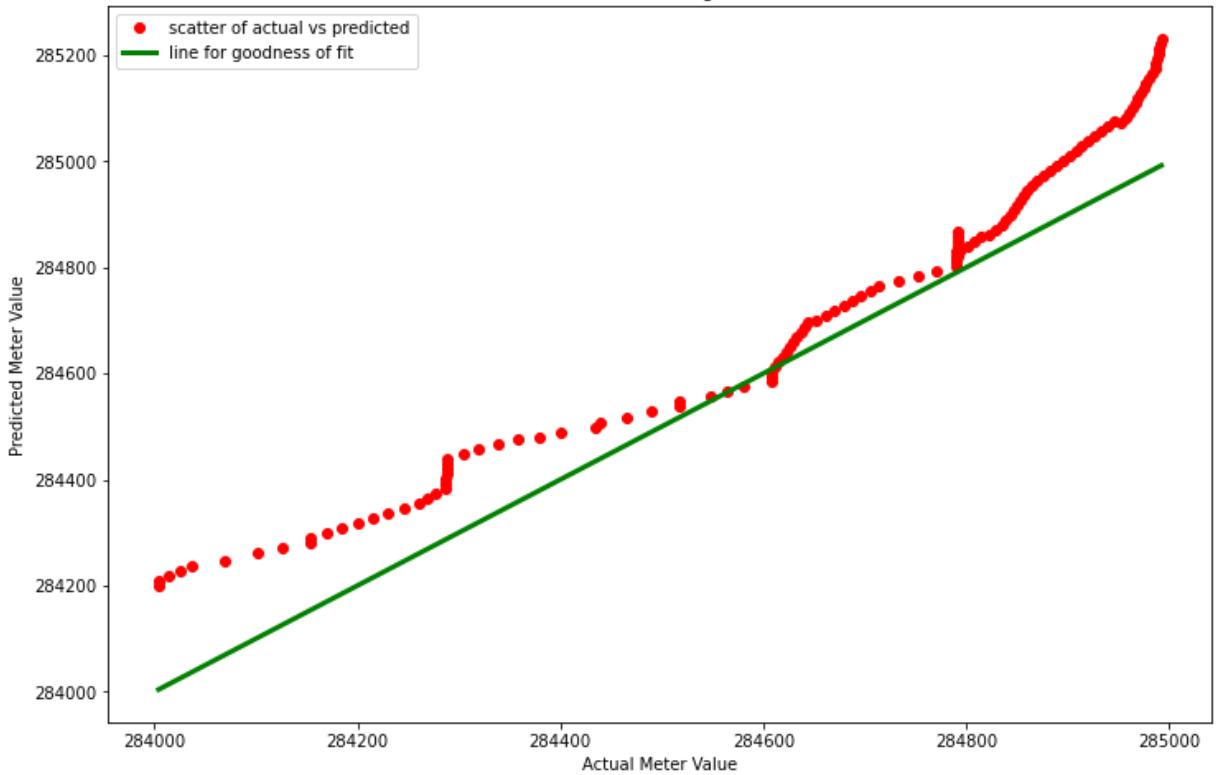


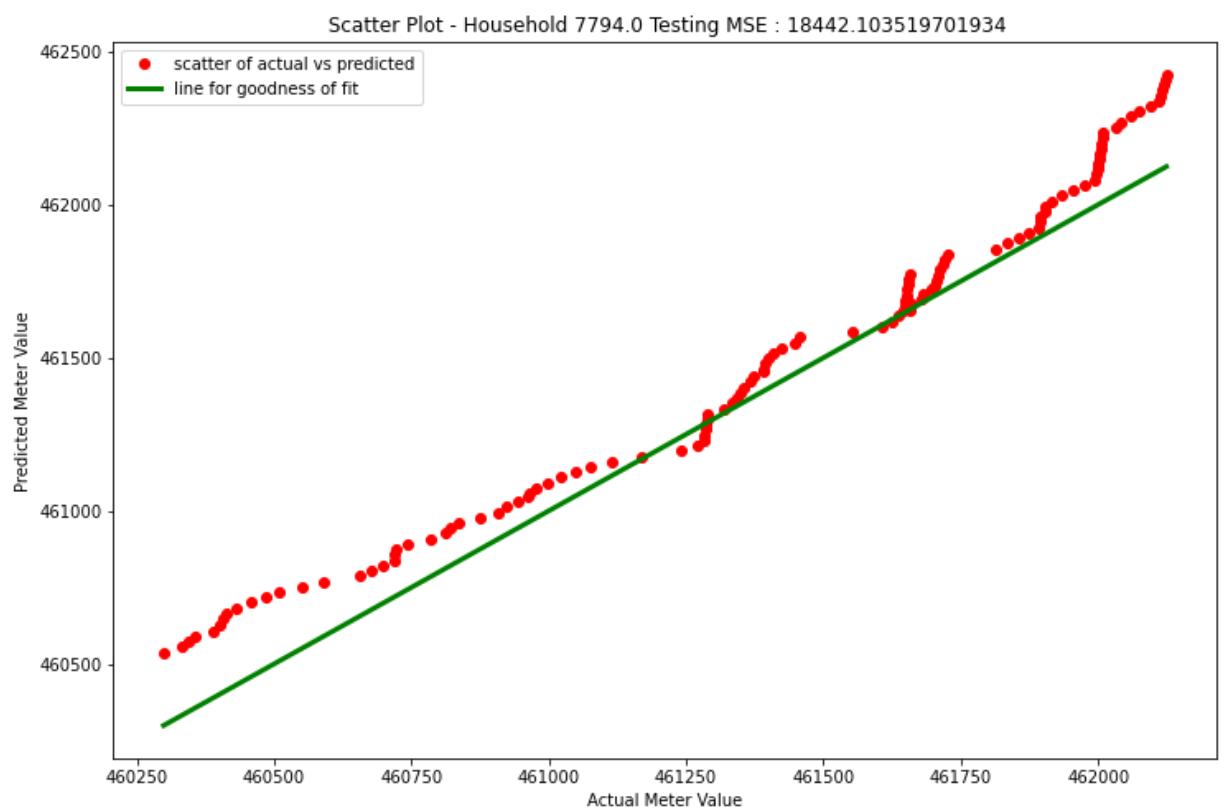
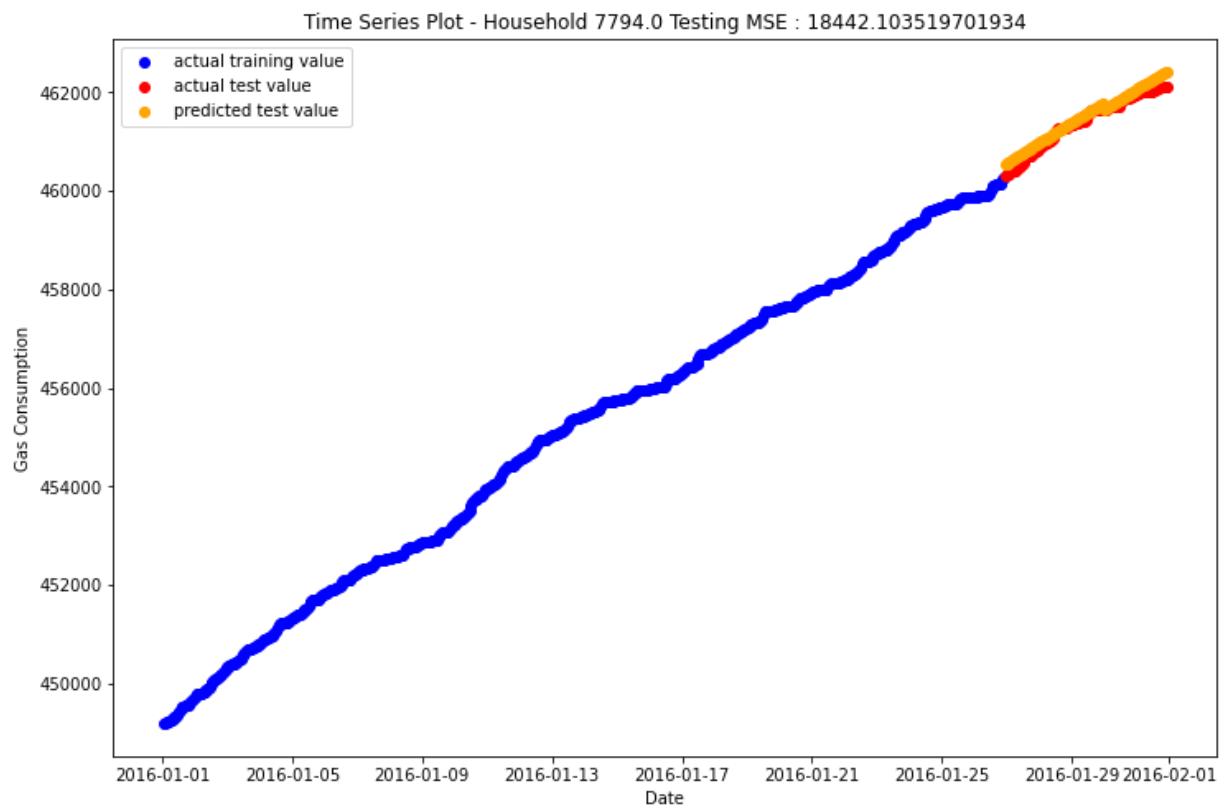


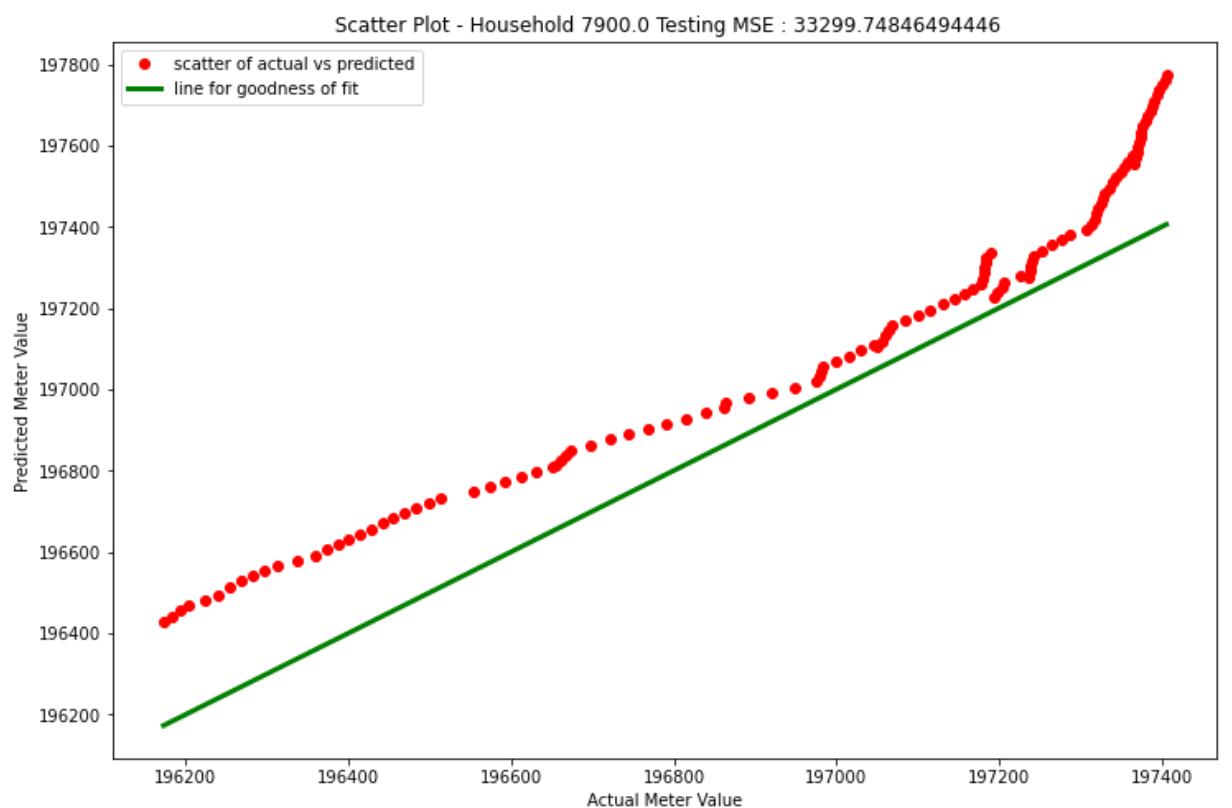
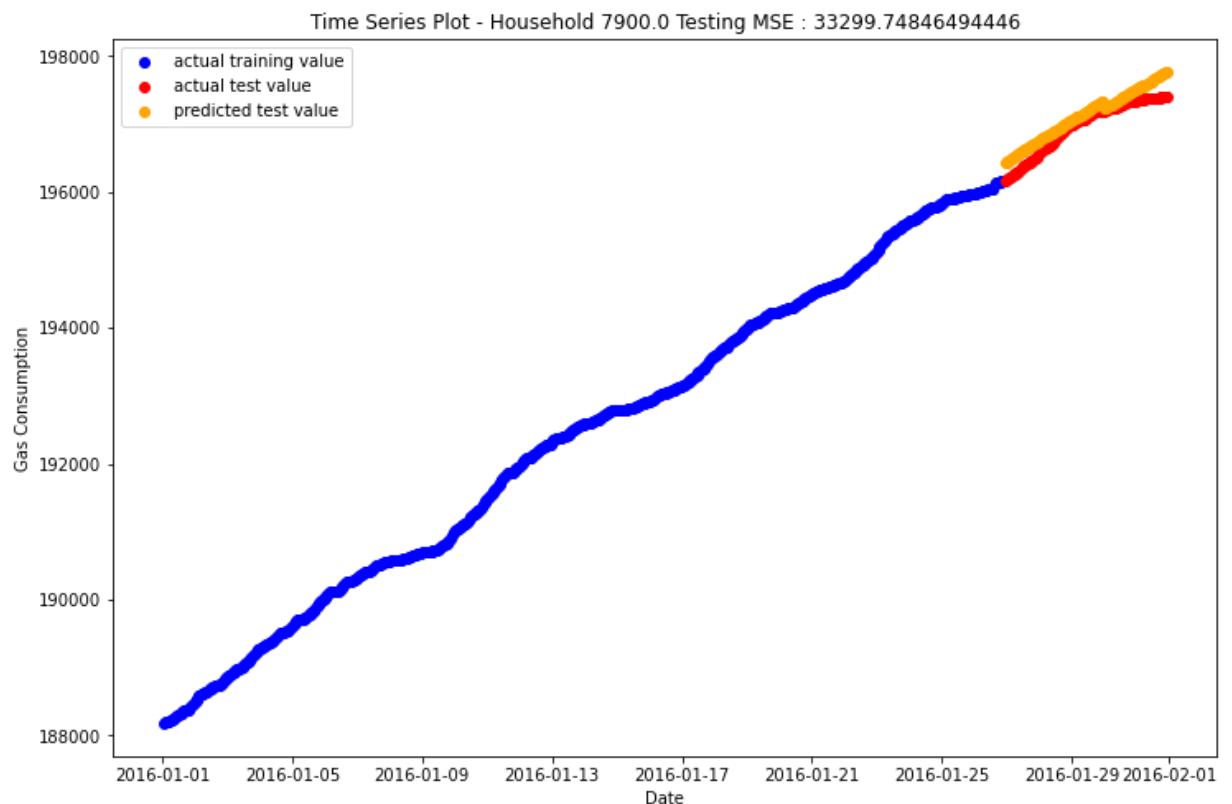
Time Series Plot - Household 7741.0 Testing MSE : 12375.445808251405



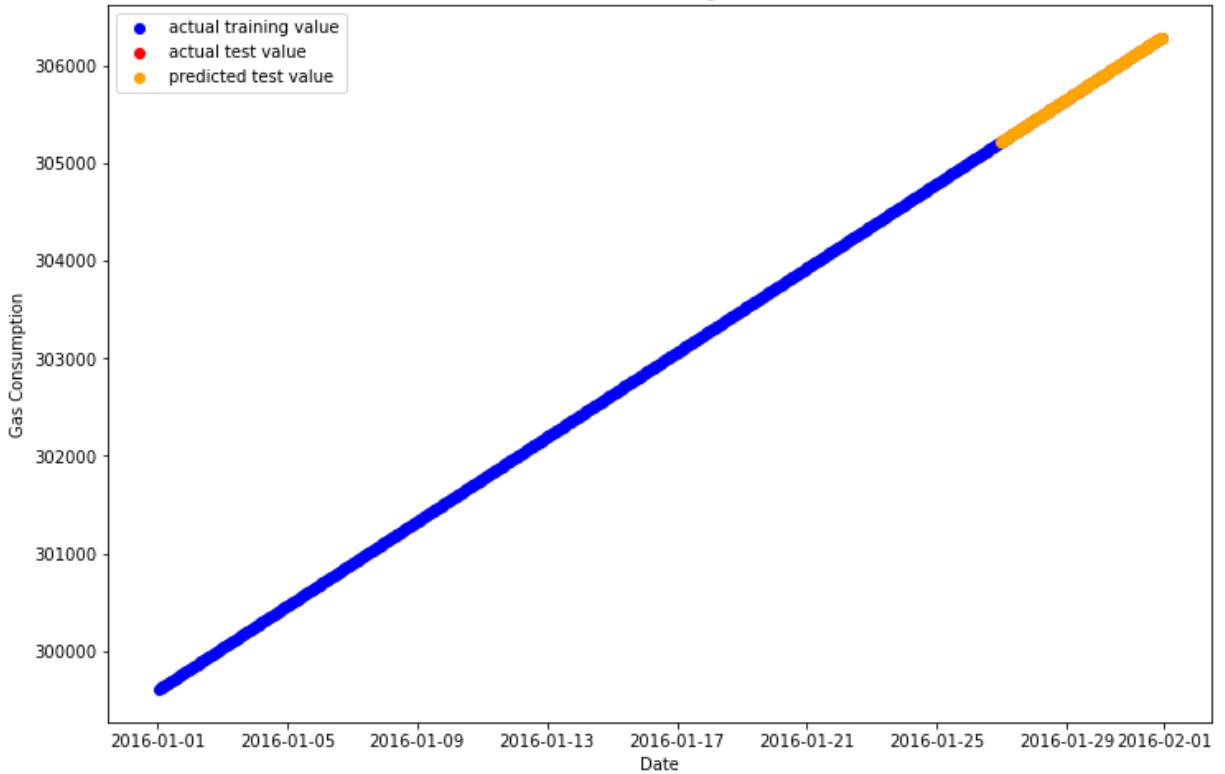
Scatter Plot - Household 7741.0 Testing MSE : 12375.445808251405



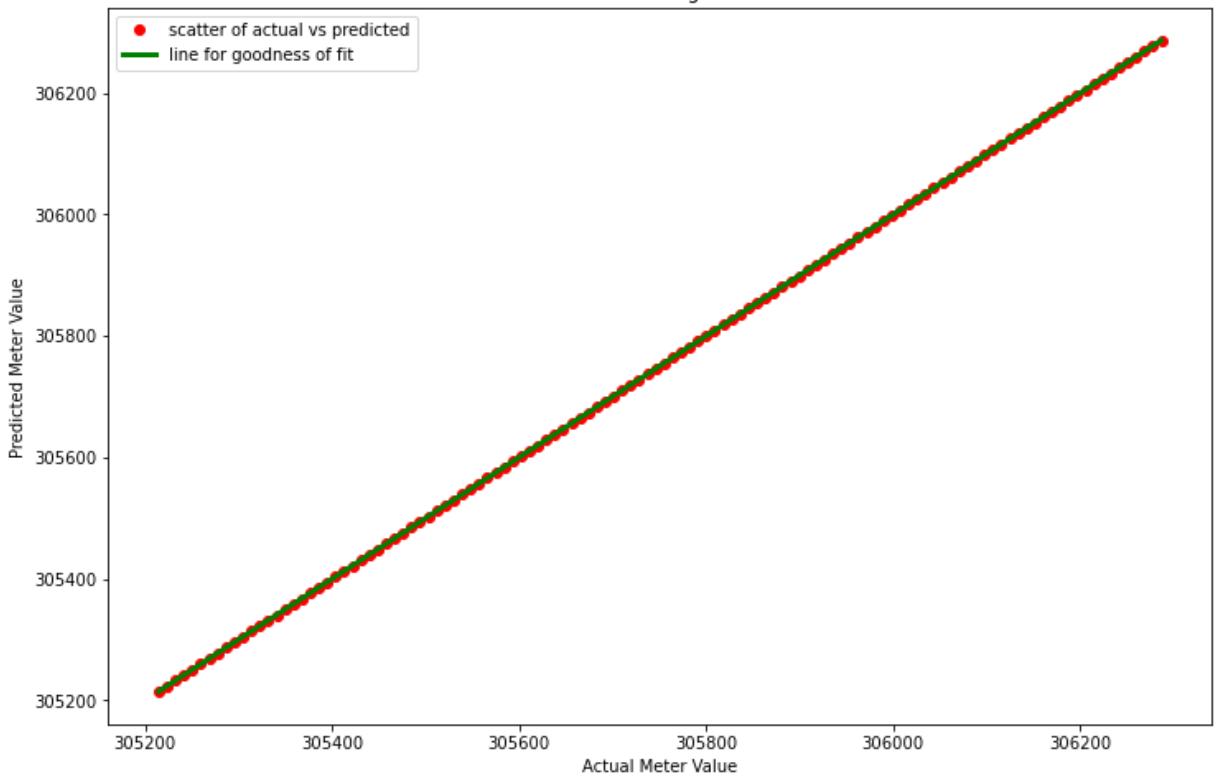


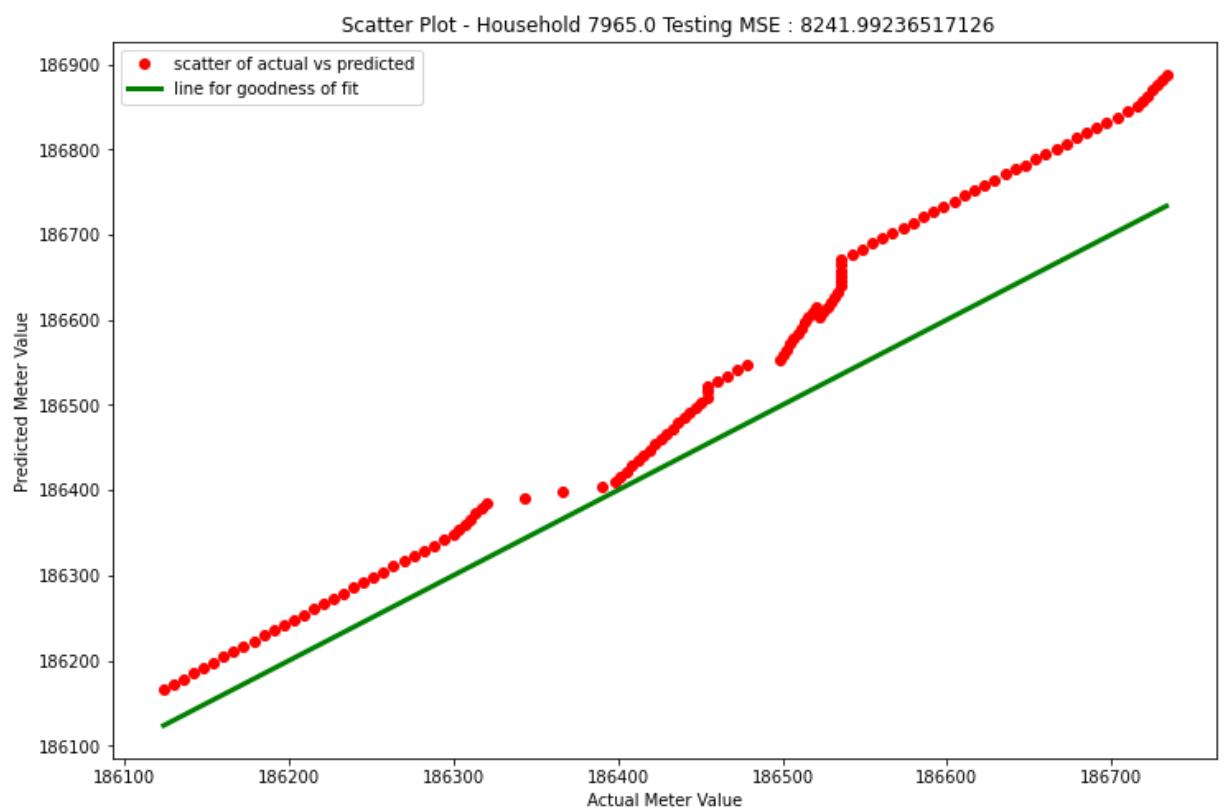
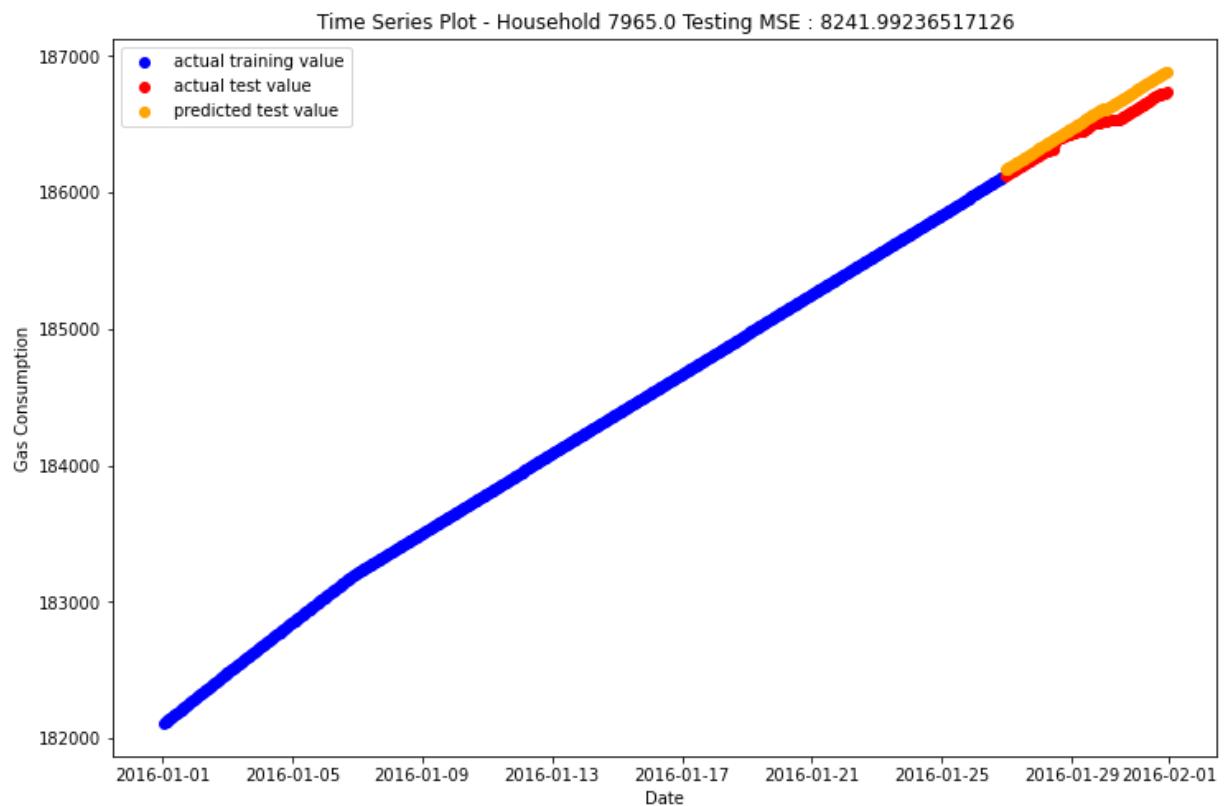


Time Series Plot - Household 7919.0 Testing MSE : 7.623296525288703e-22

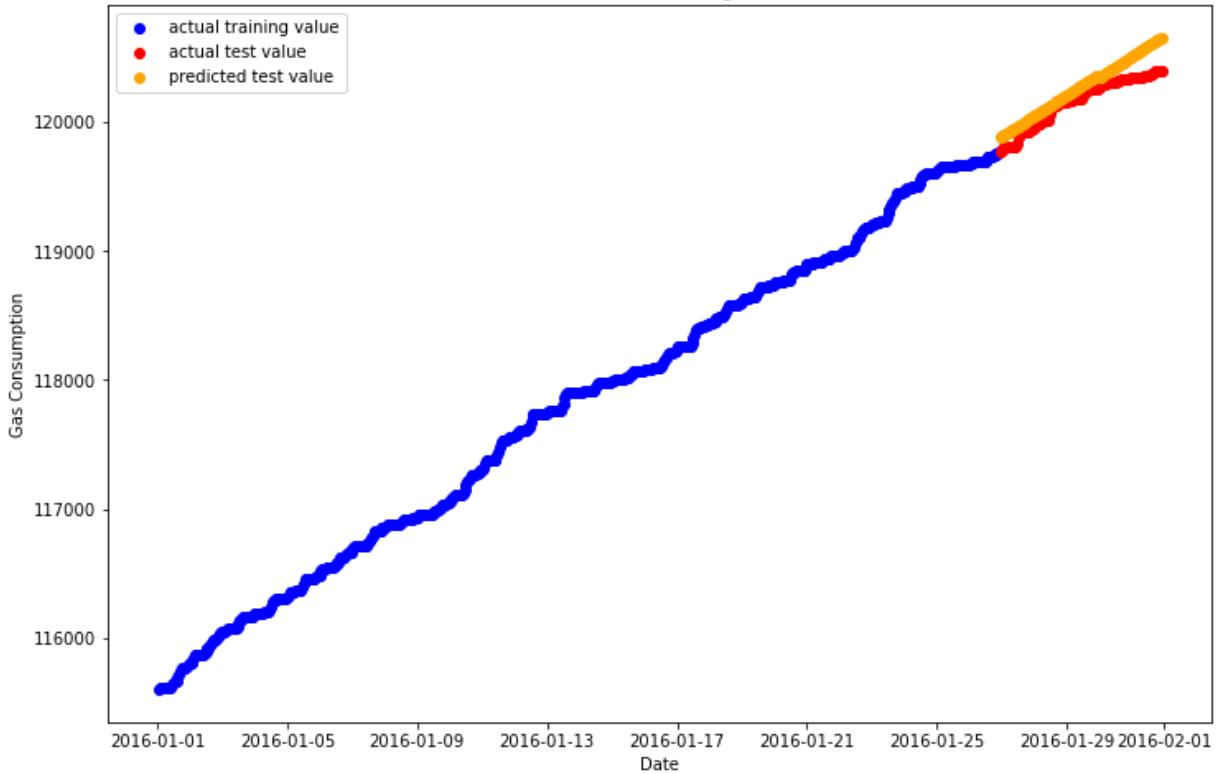


Scatter Plot - Household 7919.0 Testing MSE : 7.623296525288703e-22

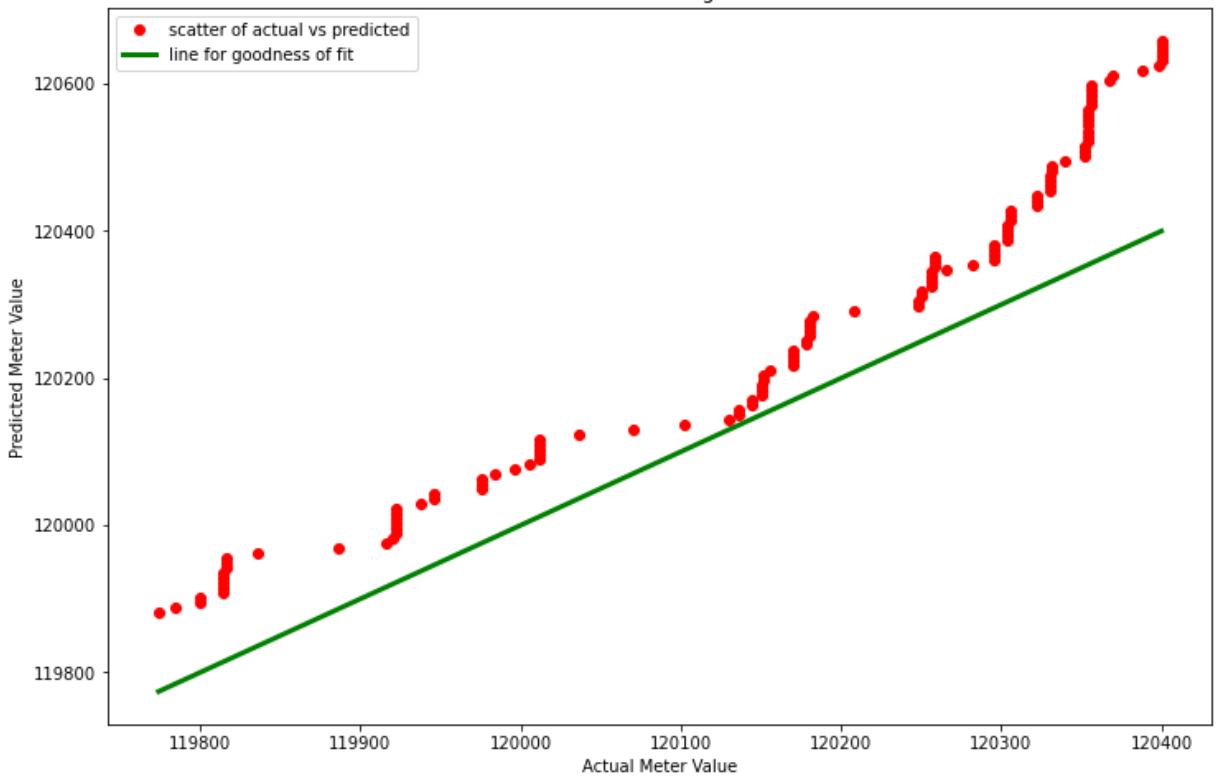


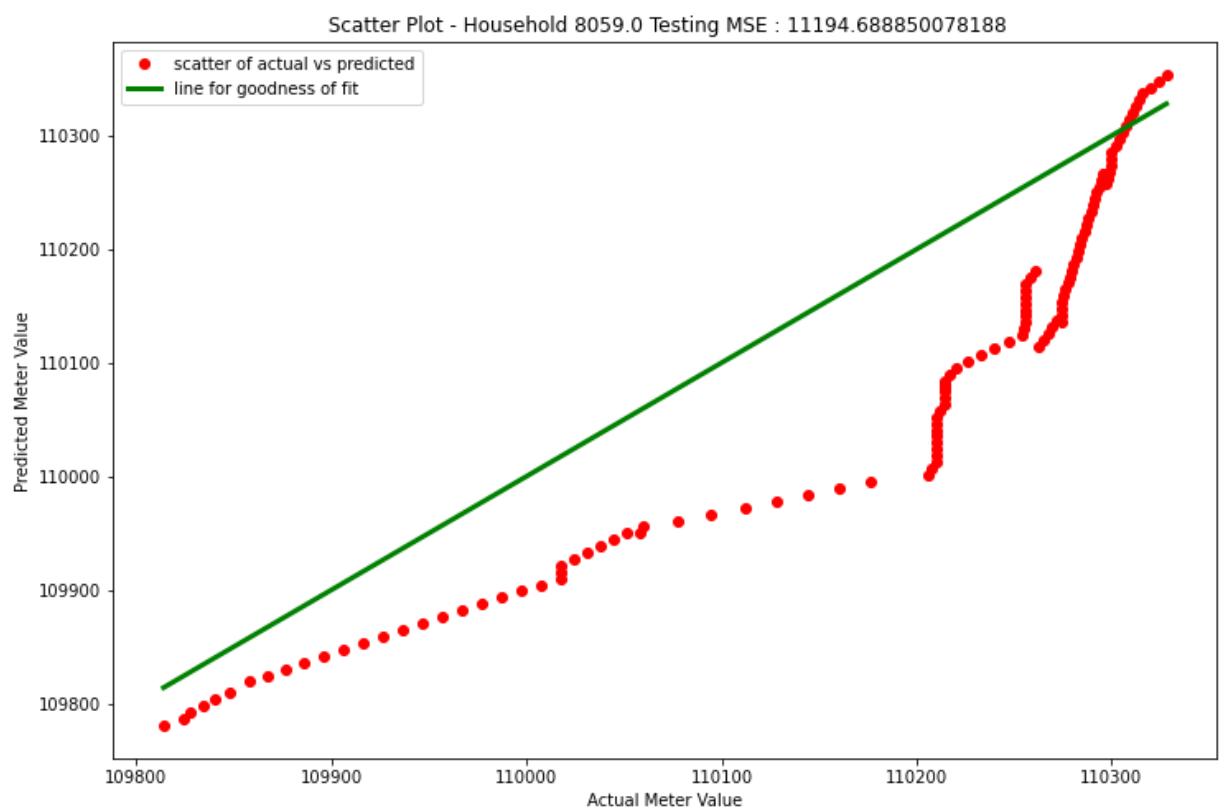
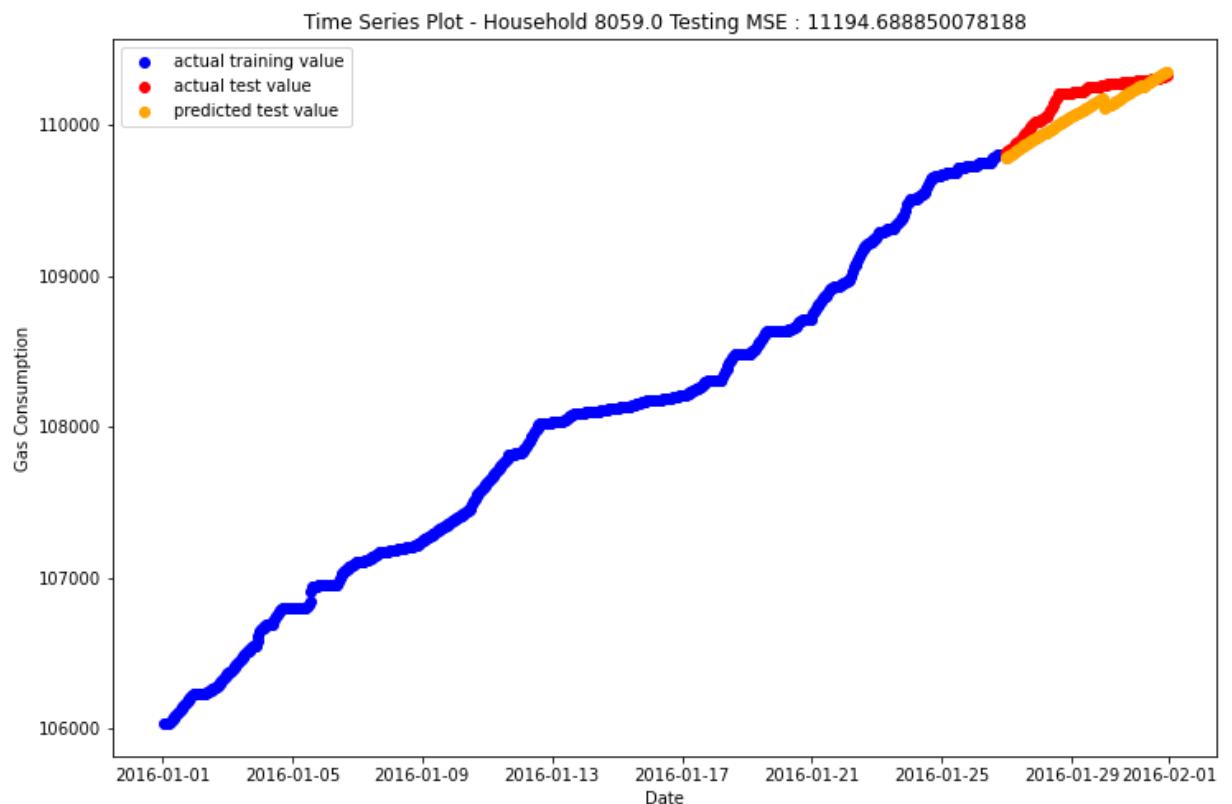


Time Series Plot - Household 7989.0 Testing MSE : 15985.140057497158

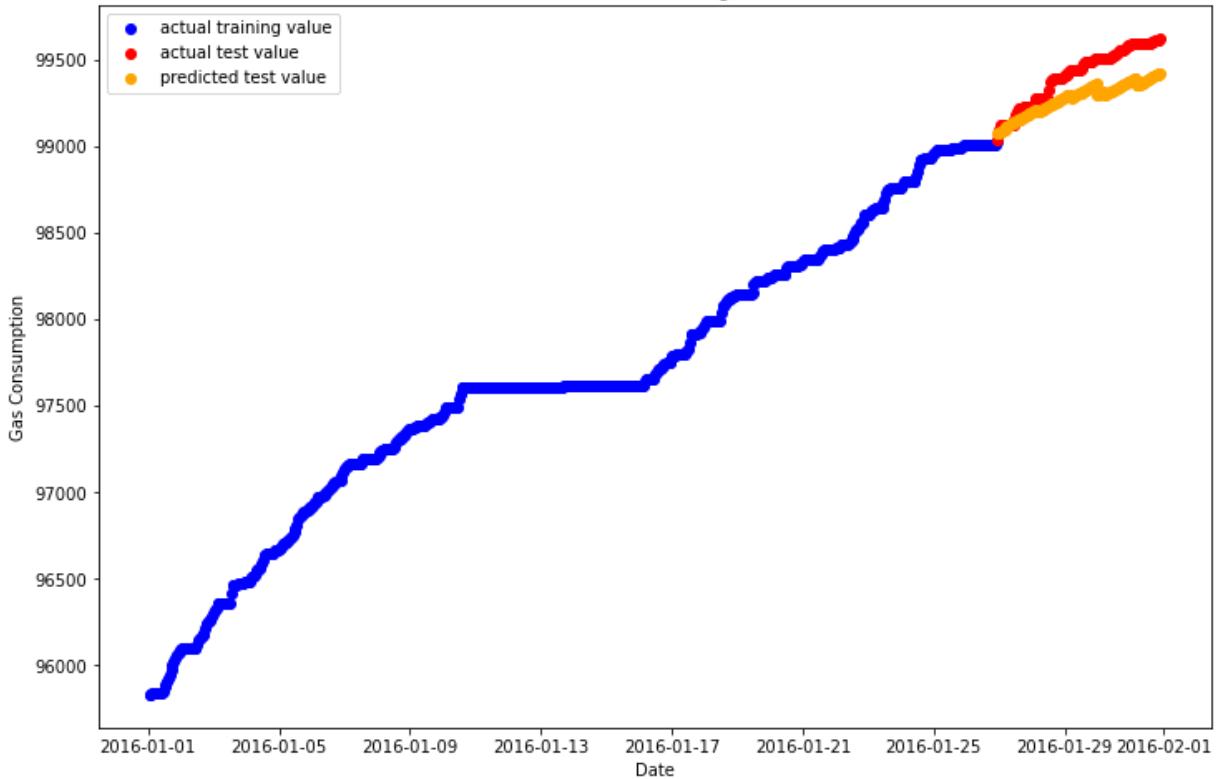


Scatter Plot - Household 7989.0 Testing MSE : 15985.140057497158

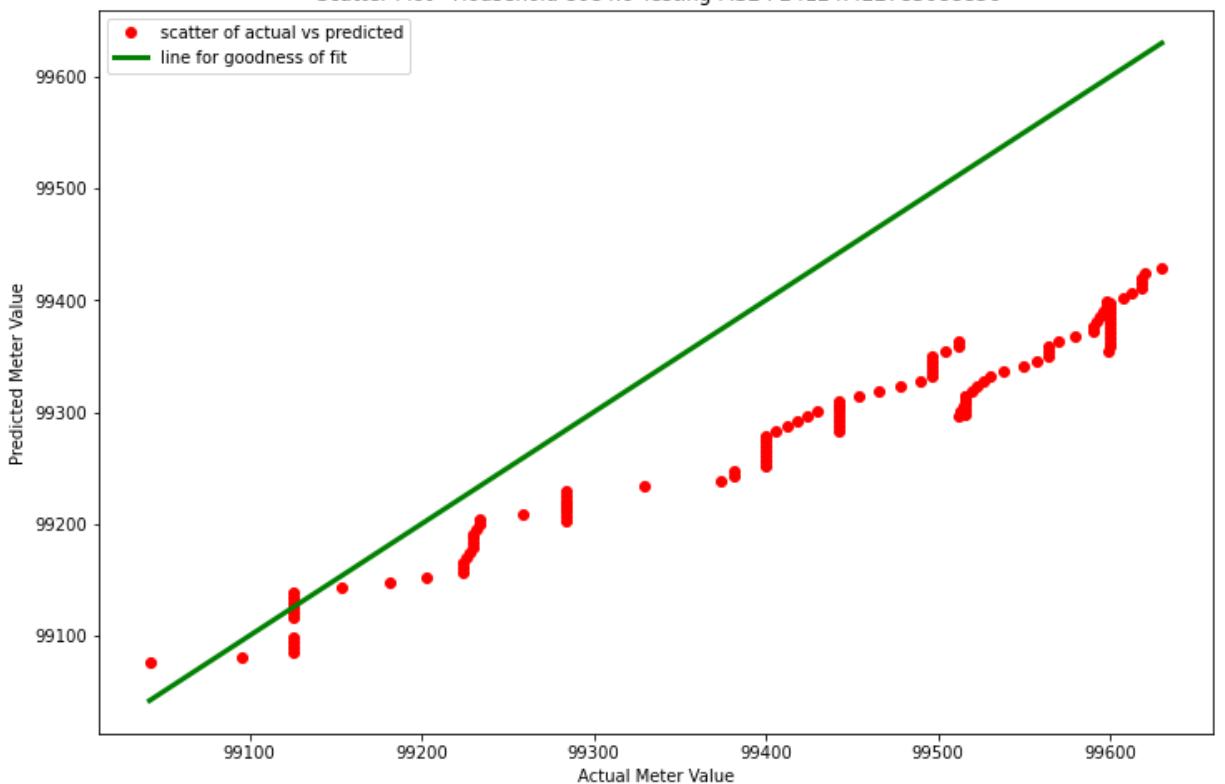


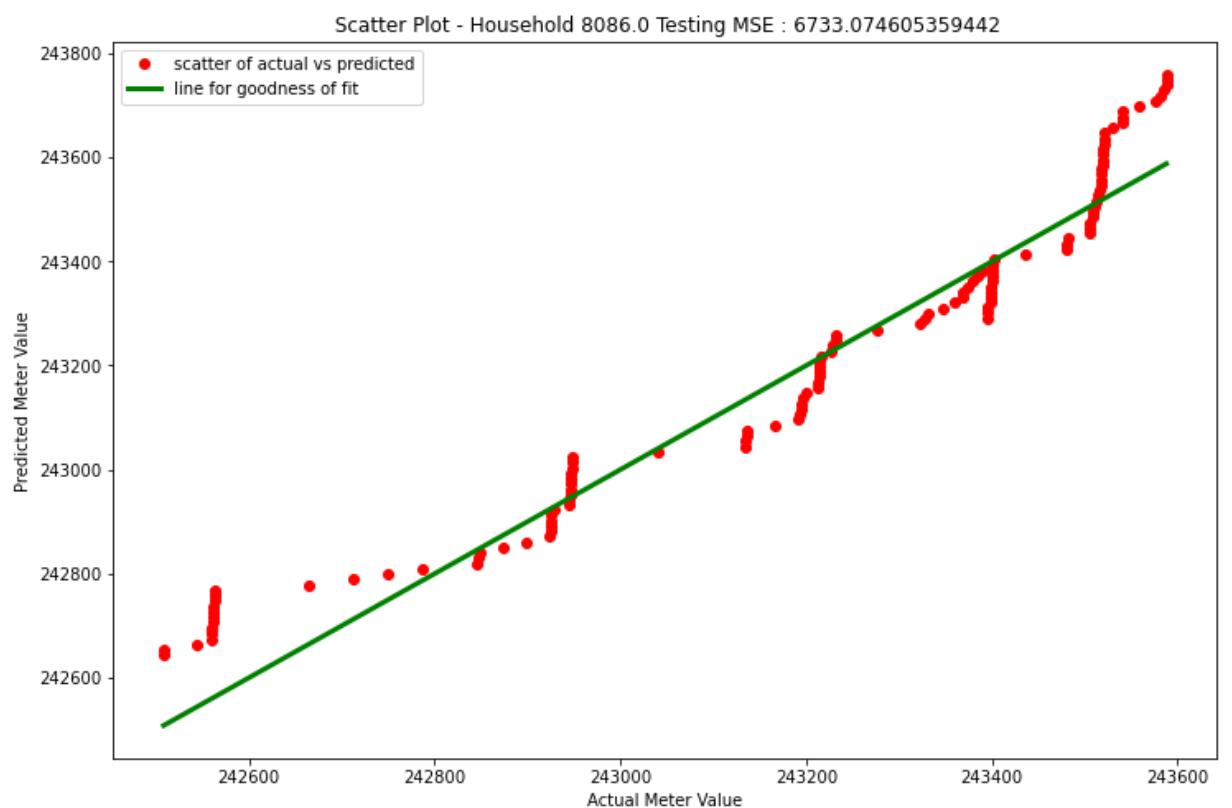
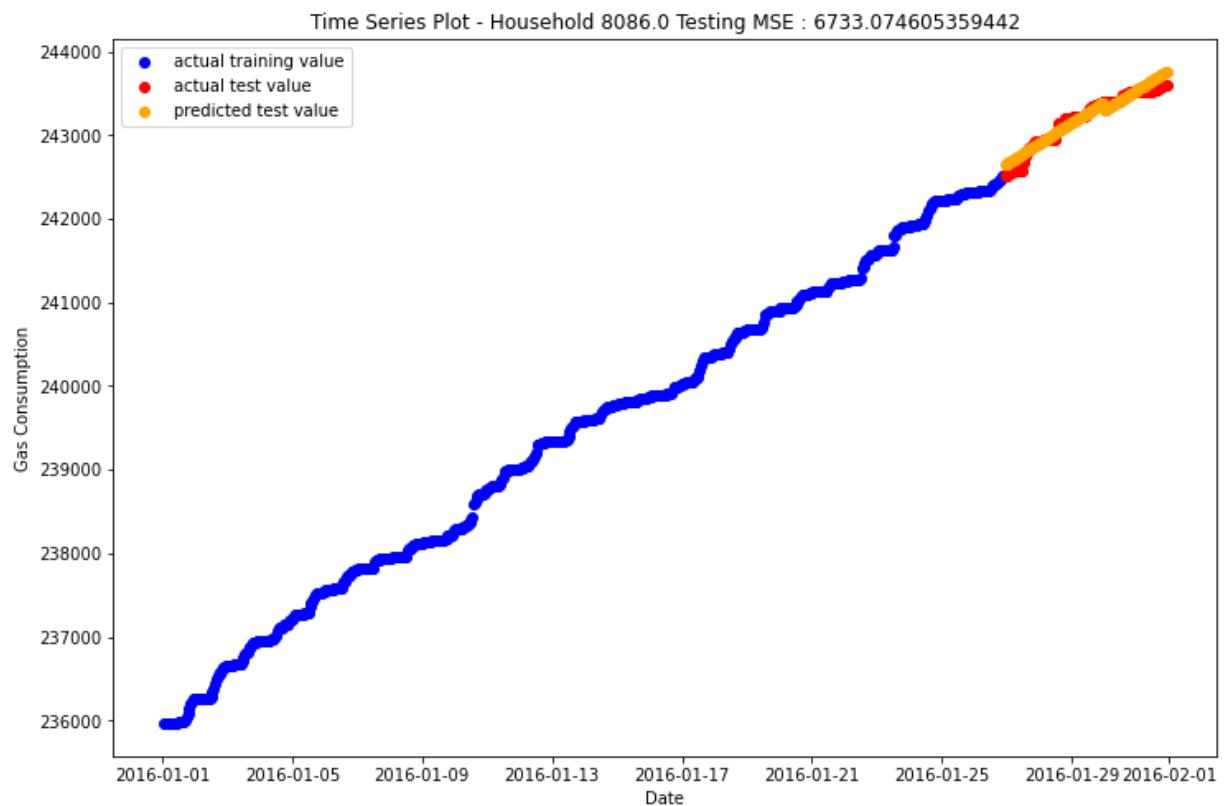


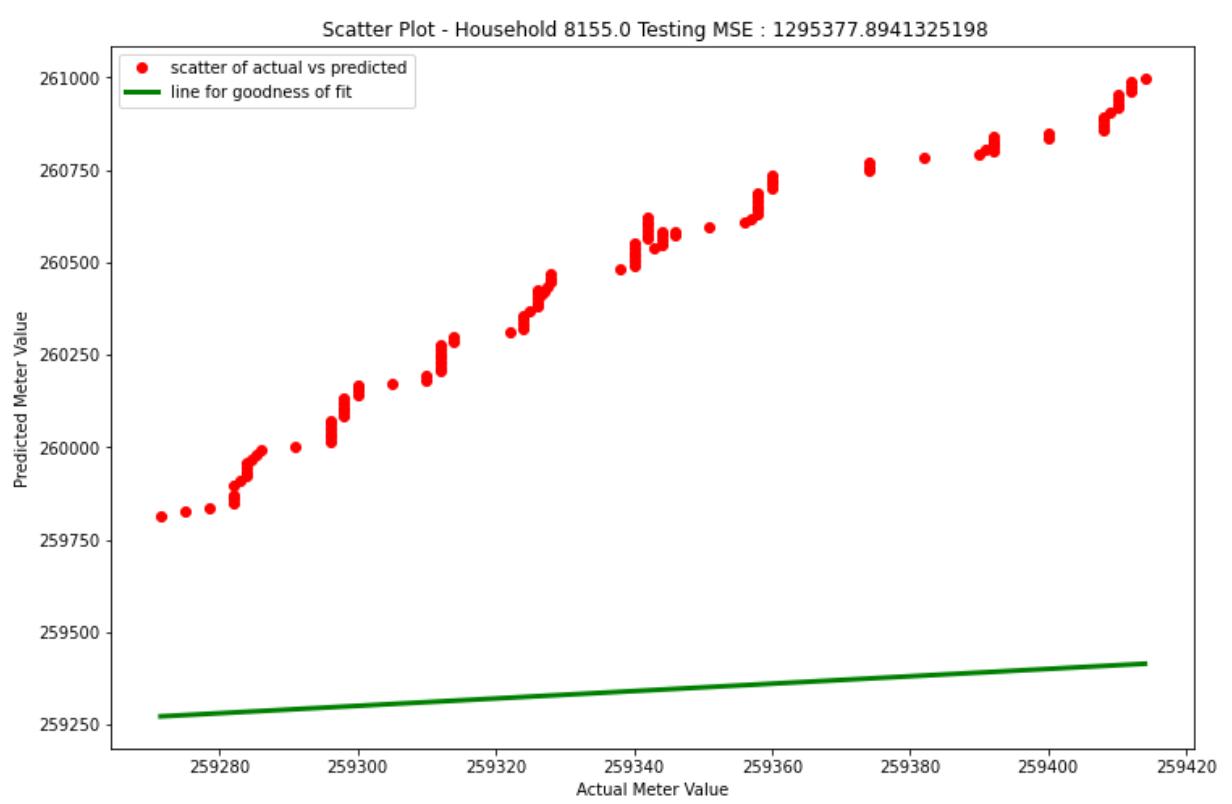
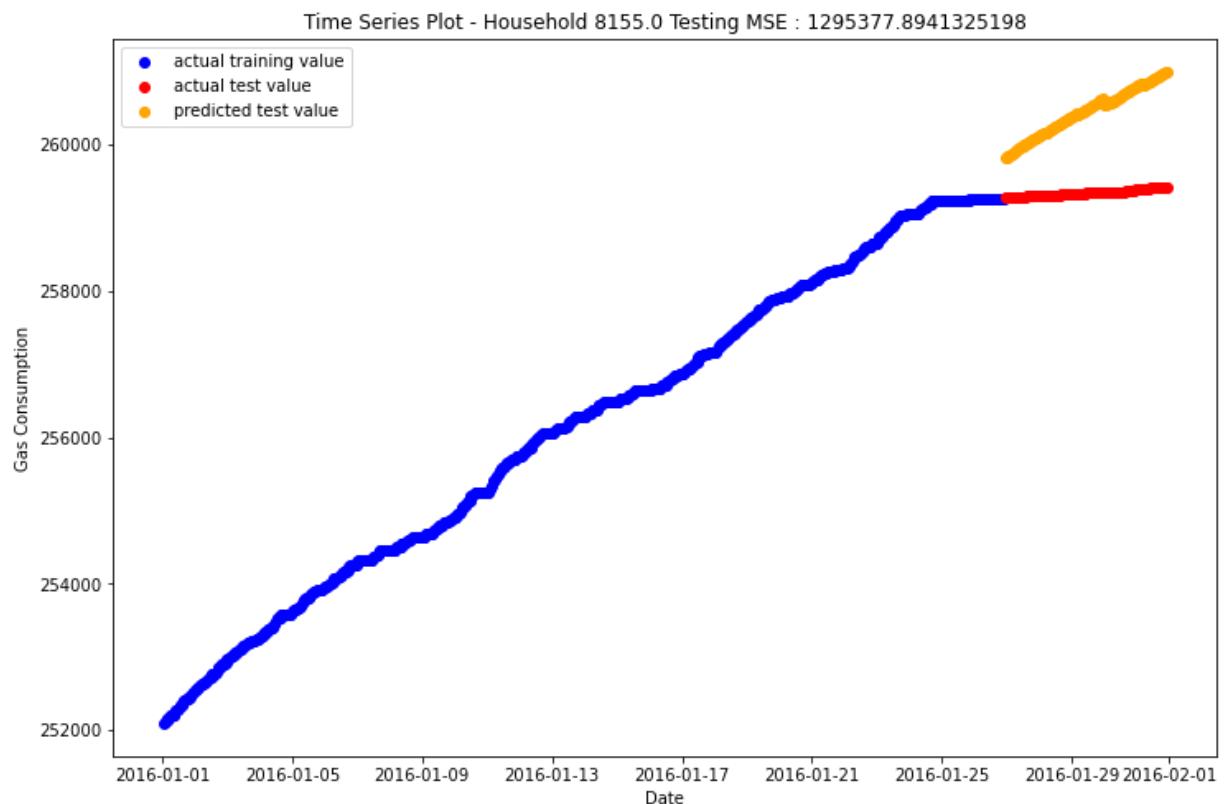
Time Series Plot - Household 8084.0 Testing MSE : 24124.422783088856

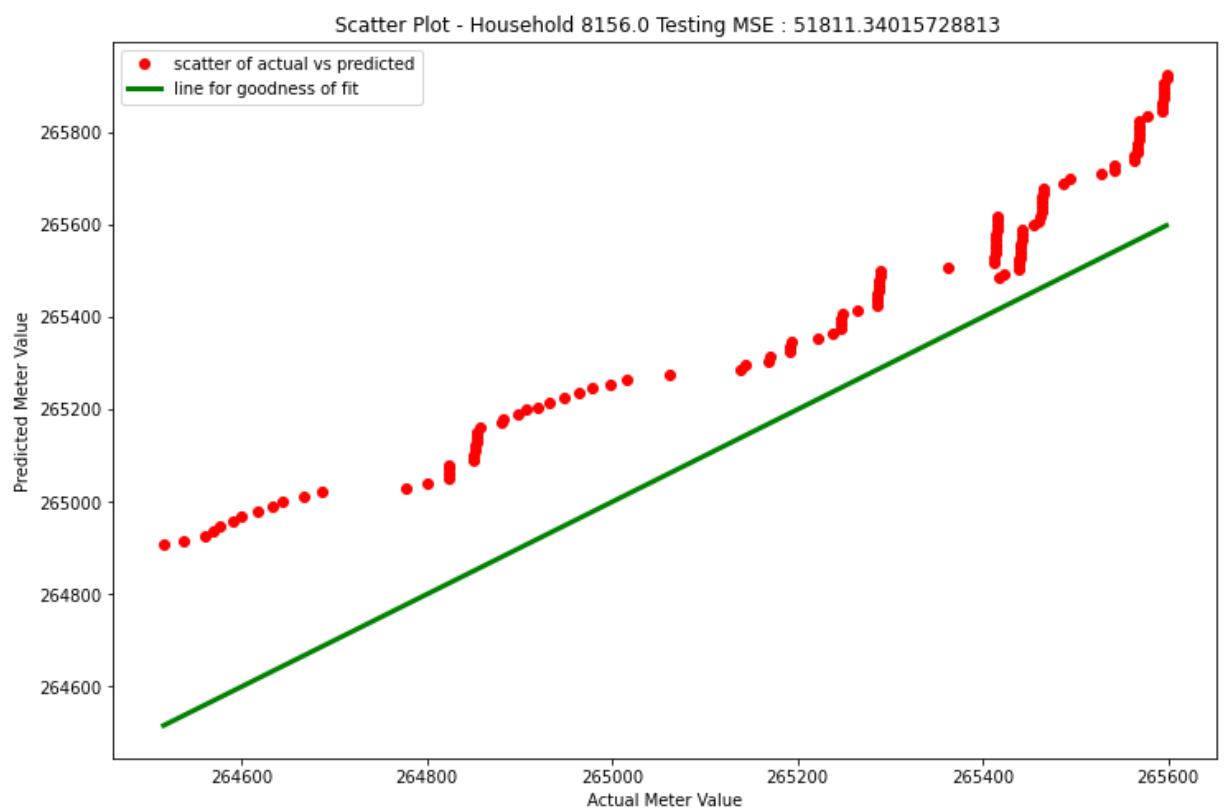
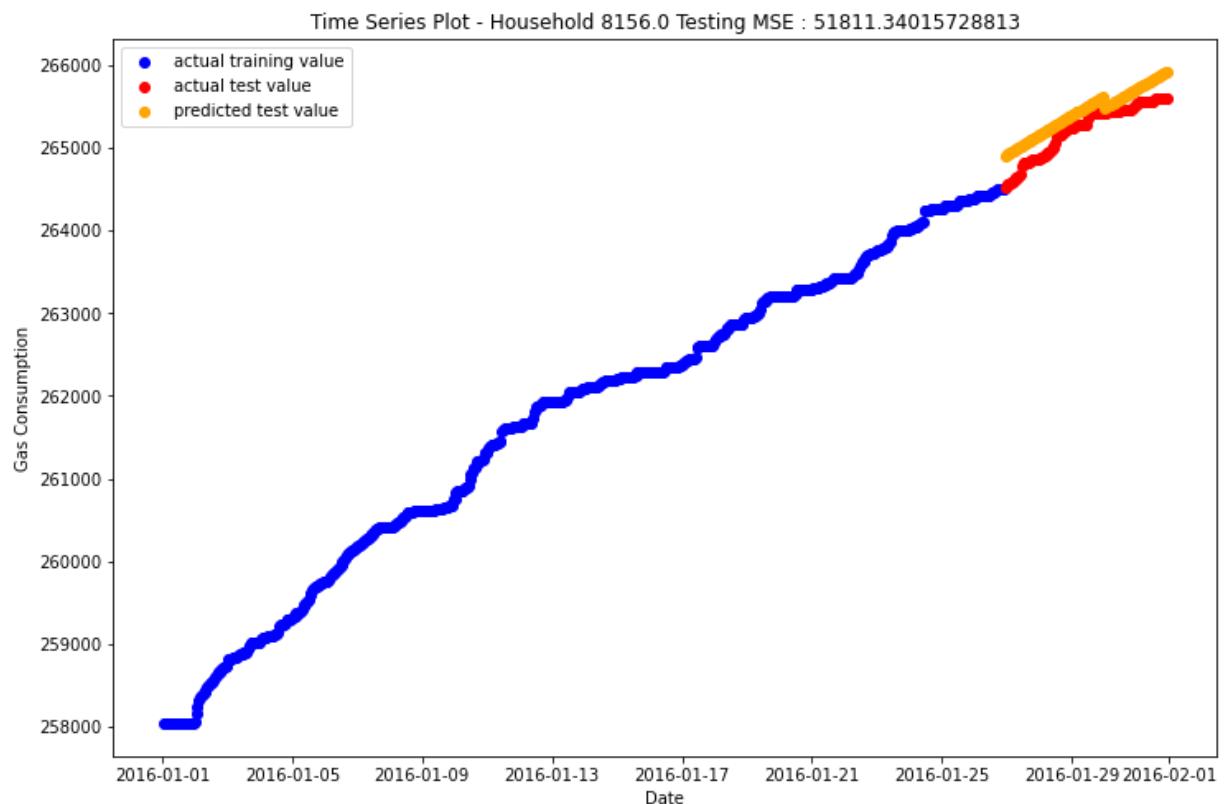


Scatter Plot - Household 8084.0 Testing MSE : 24124.422783088856

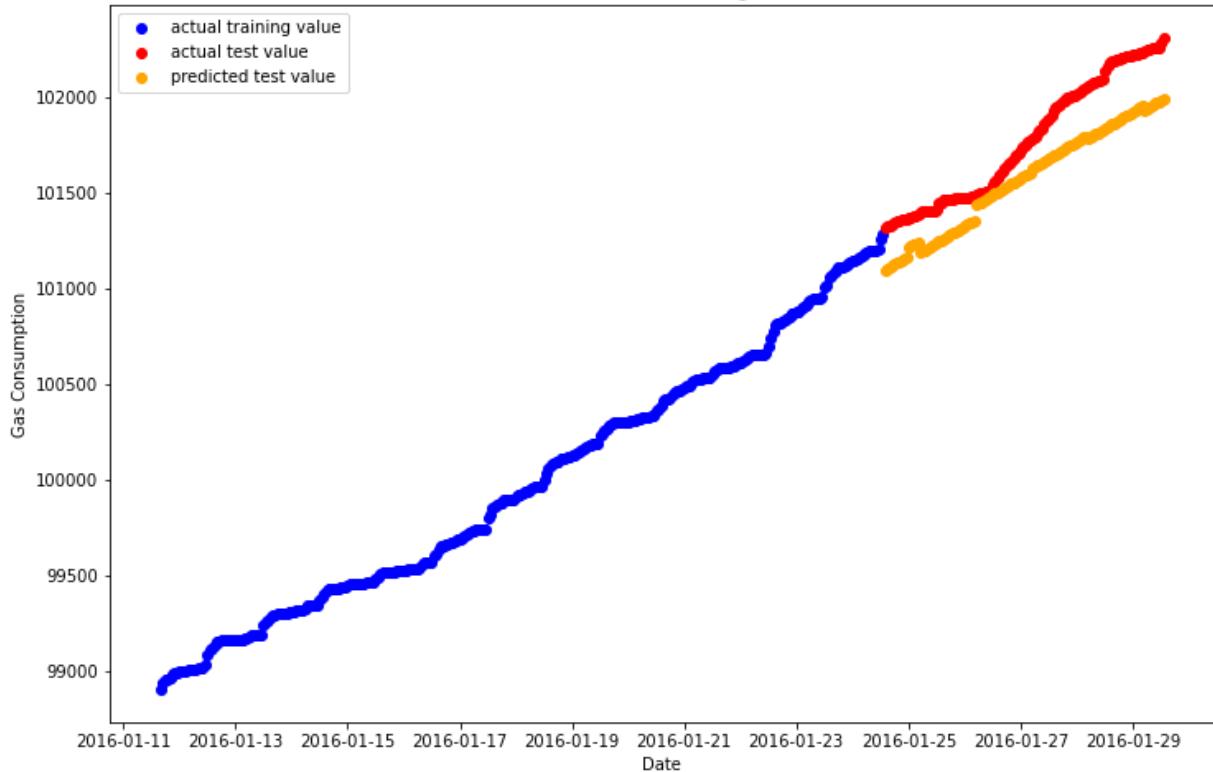




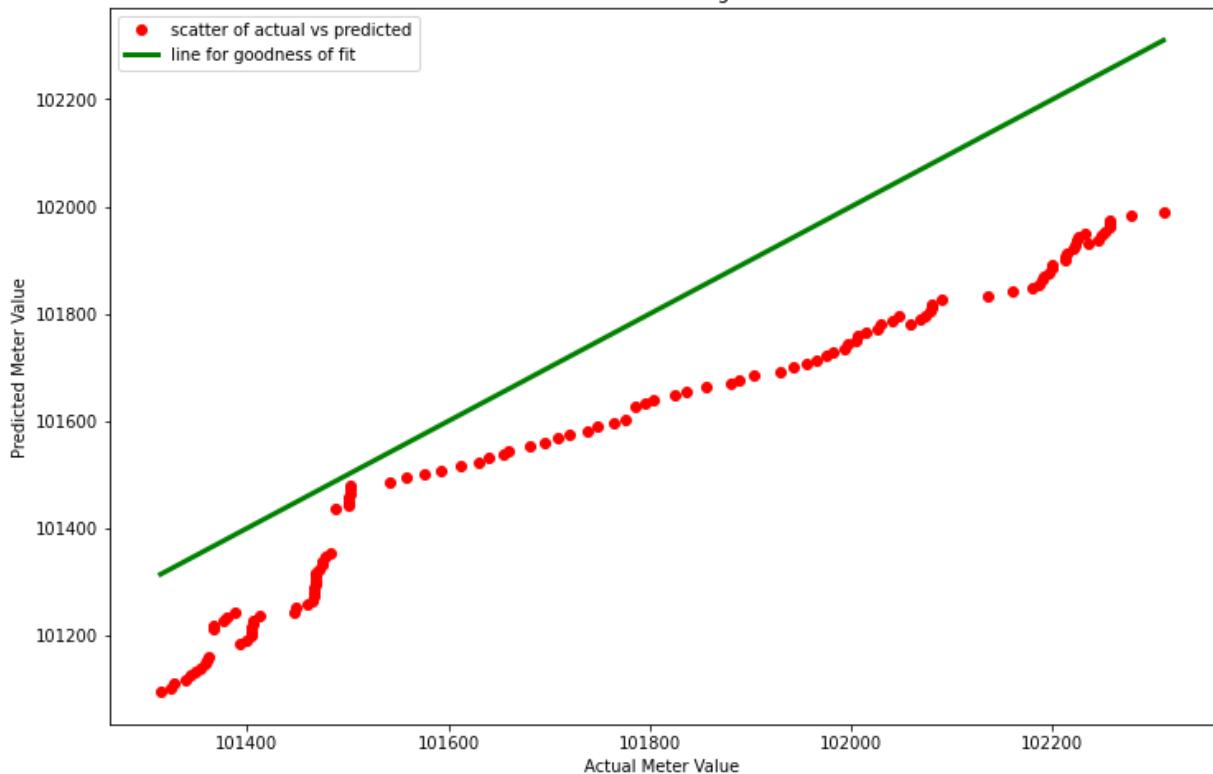




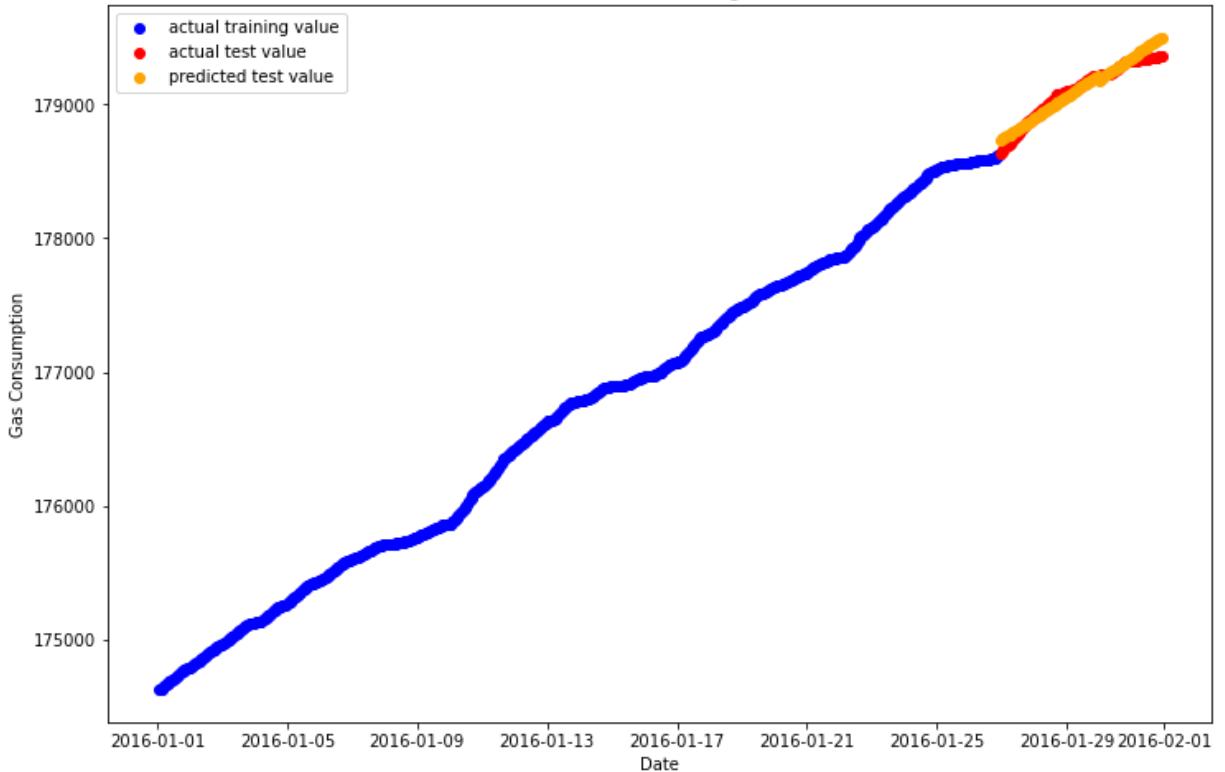
Time Series Plot - Household 8244.0 Testing MSE : 48446.0231813814



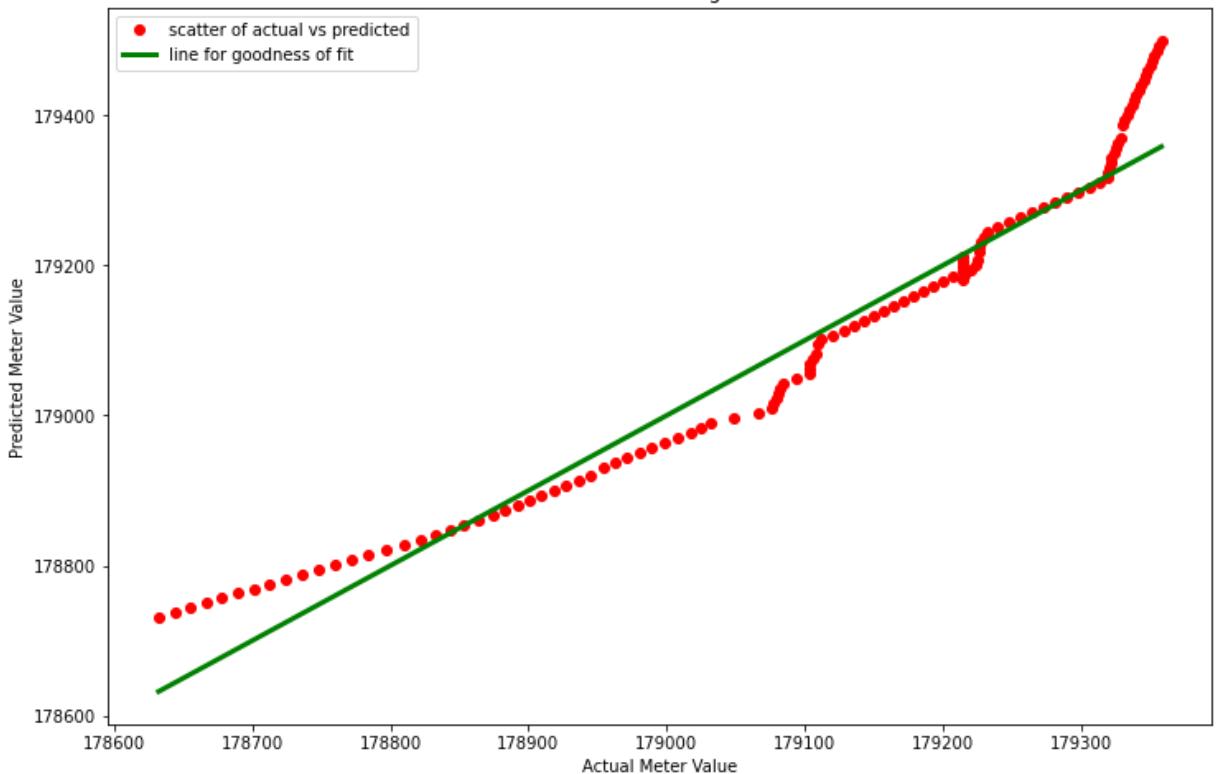
Scatter Plot - Household 8244.0 Testing MSE : 48446.0231813814



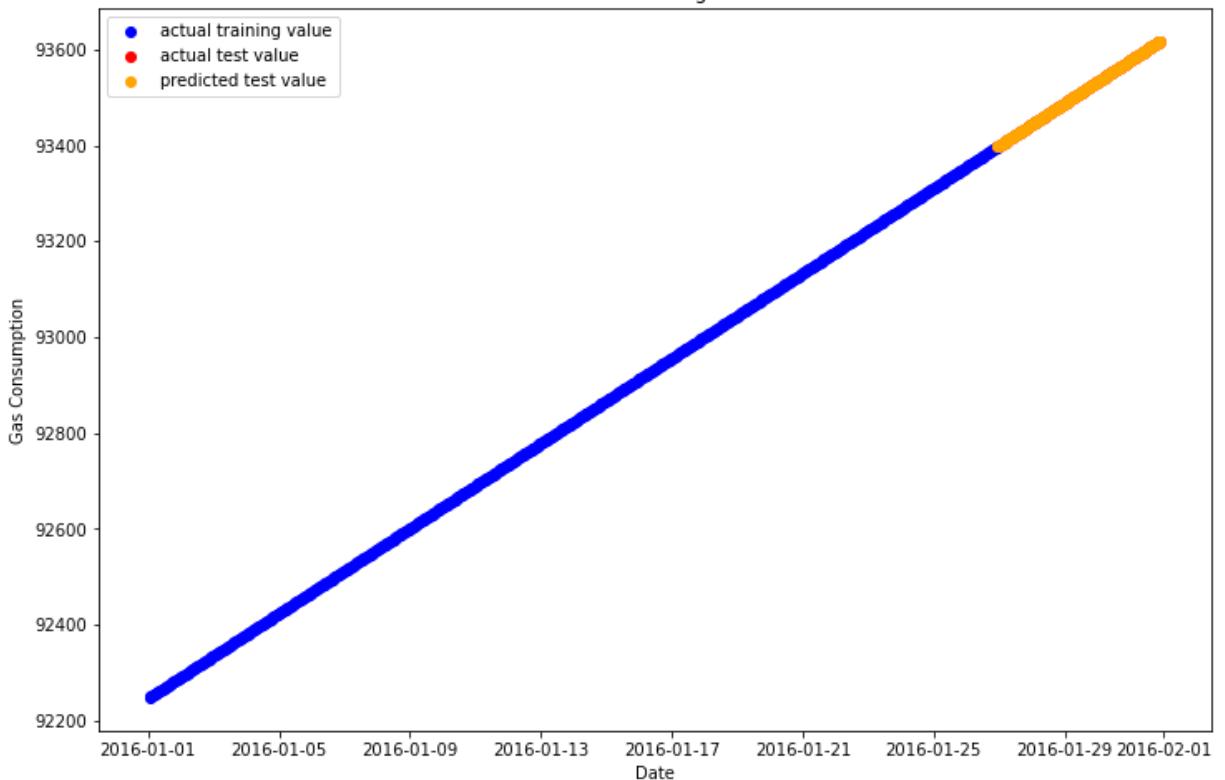
Time Series Plot - Household 8386.0 Testing MSE : 2685.731828008371



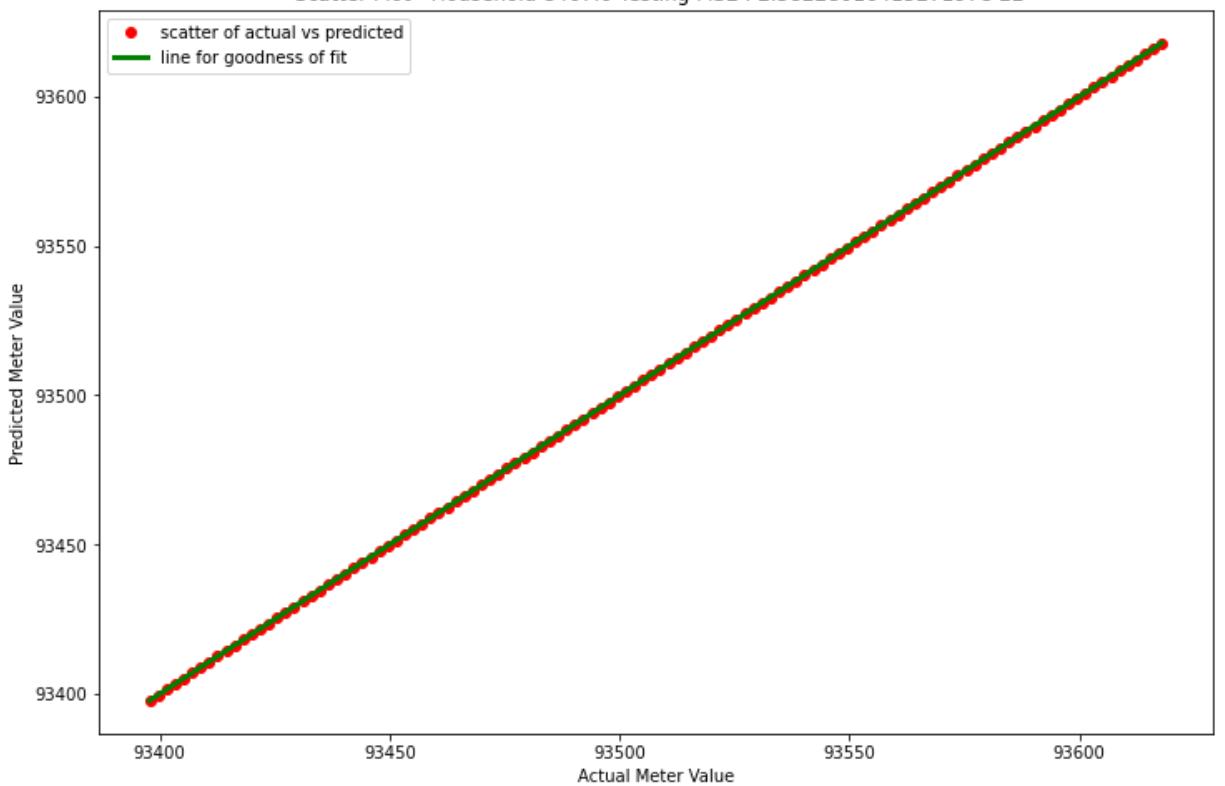
Scatter Plot - Household 8386.0 Testing MSE : 2685.731828008371



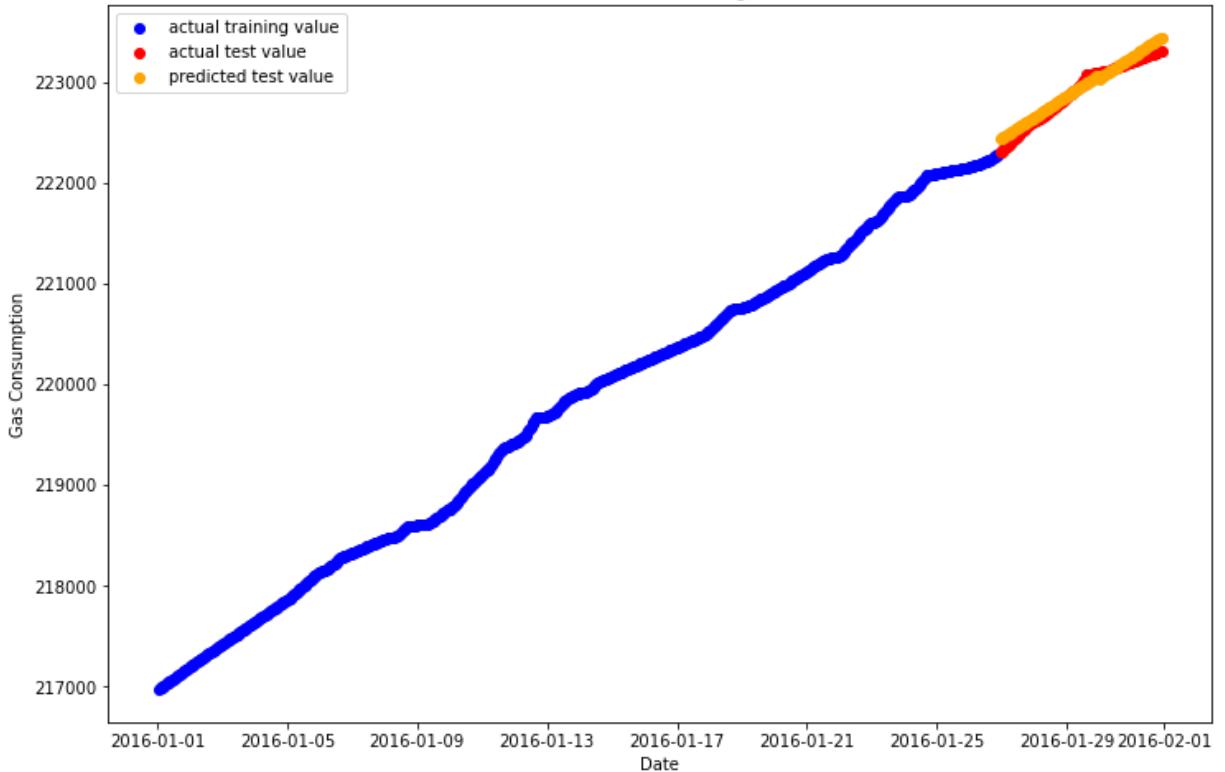
Time Series Plot - Household 8467.0 Testing MSE : 2.3822801641527197e-22



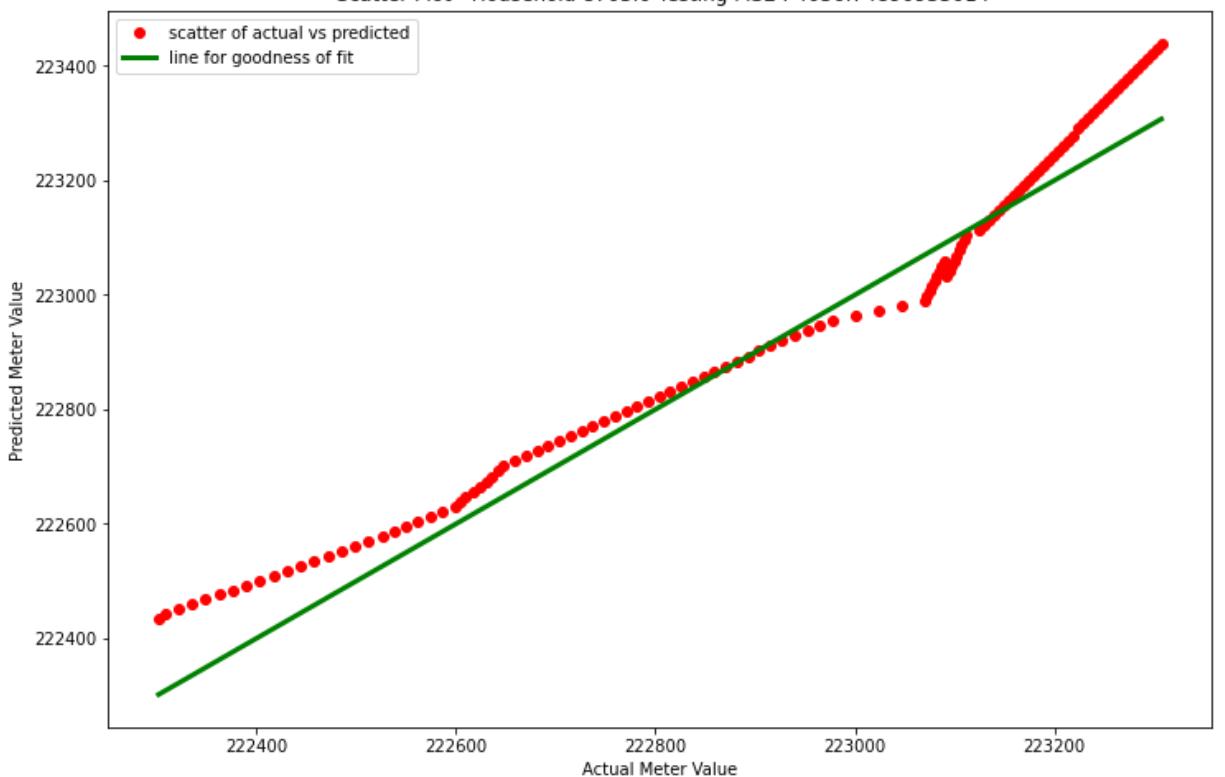
Scatter Plot - Household 8467.0 Testing MSE : 2.3822801641527197e-22

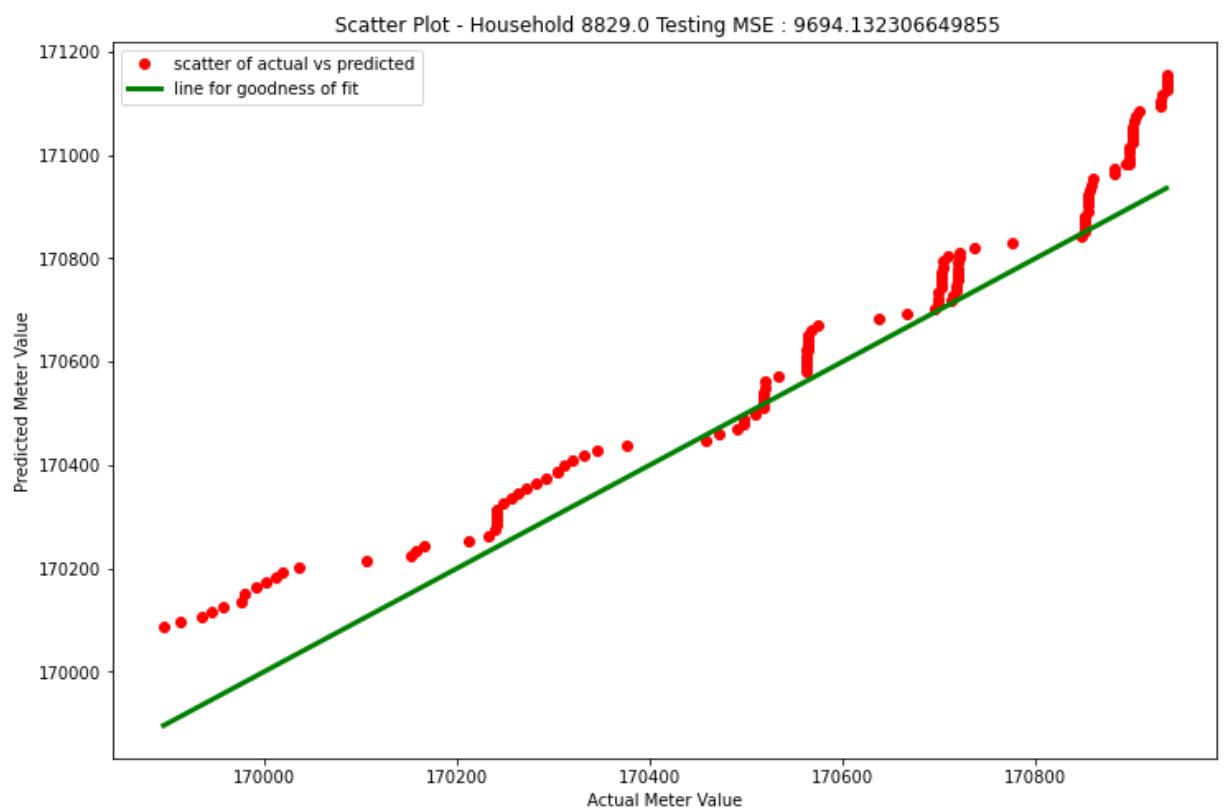
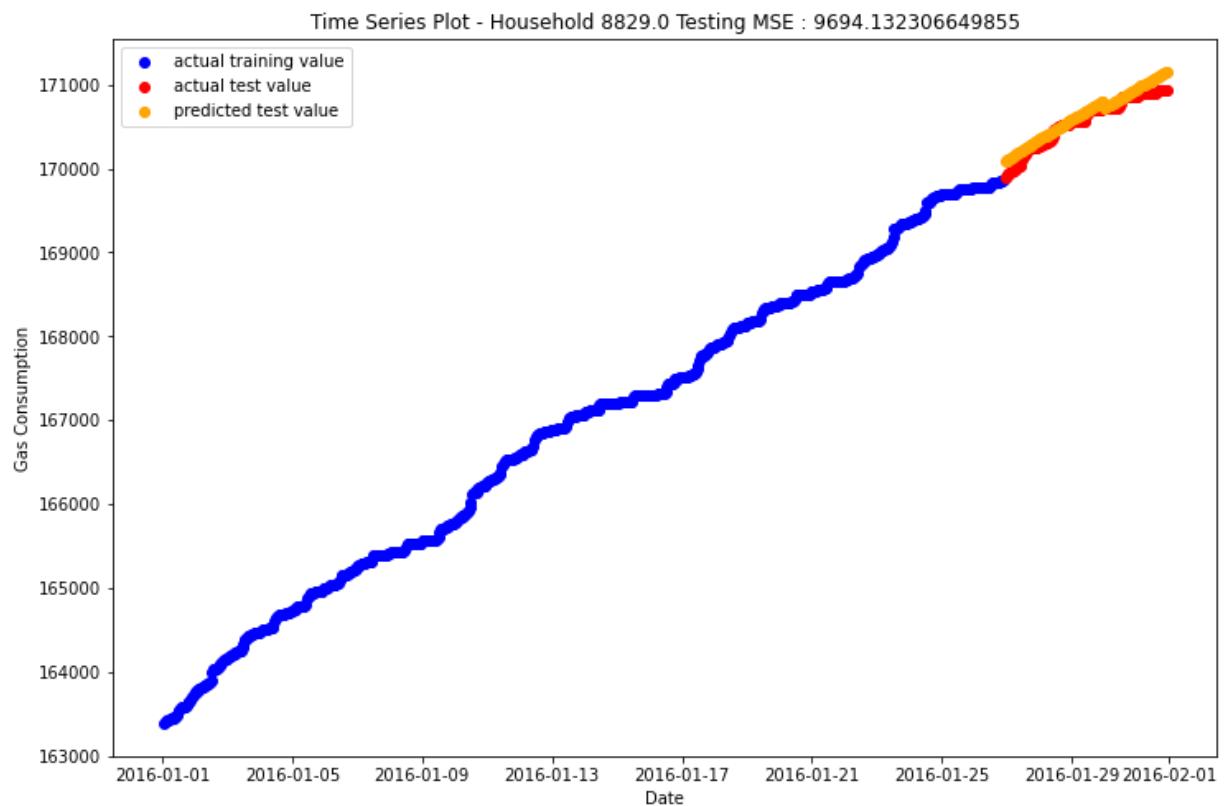


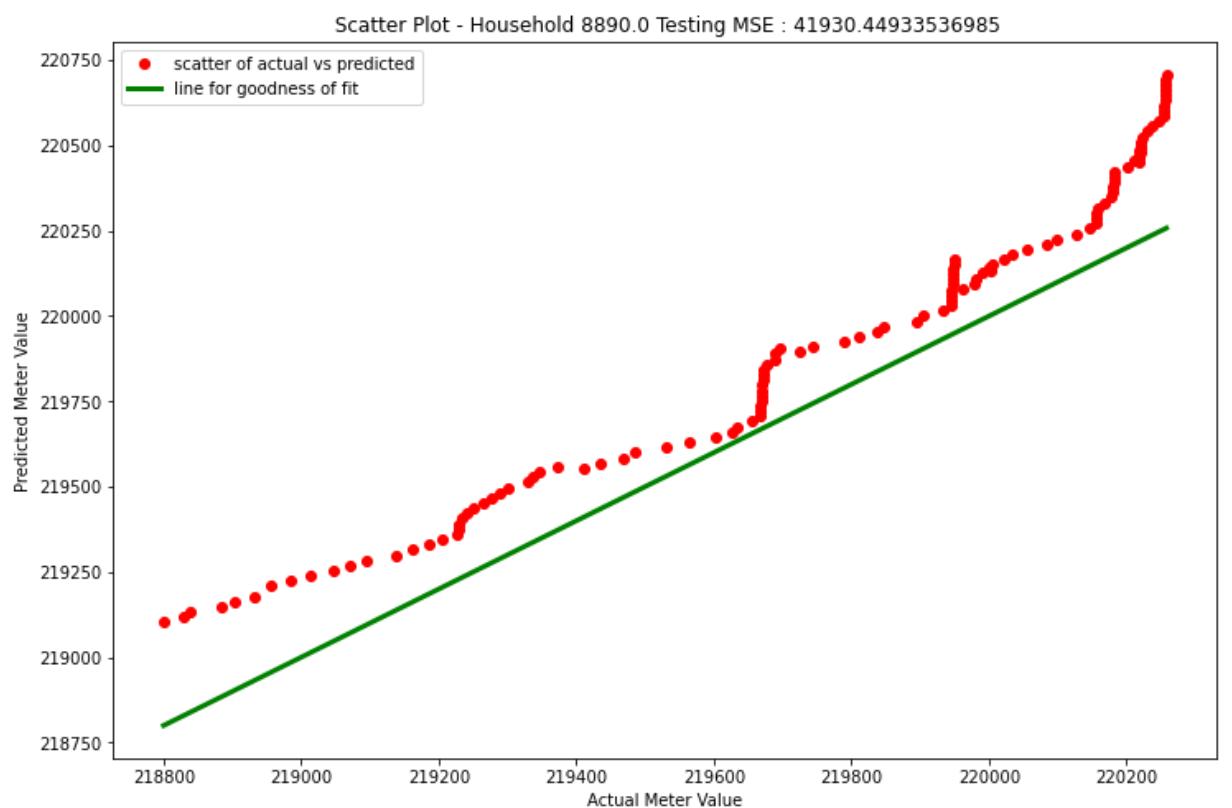
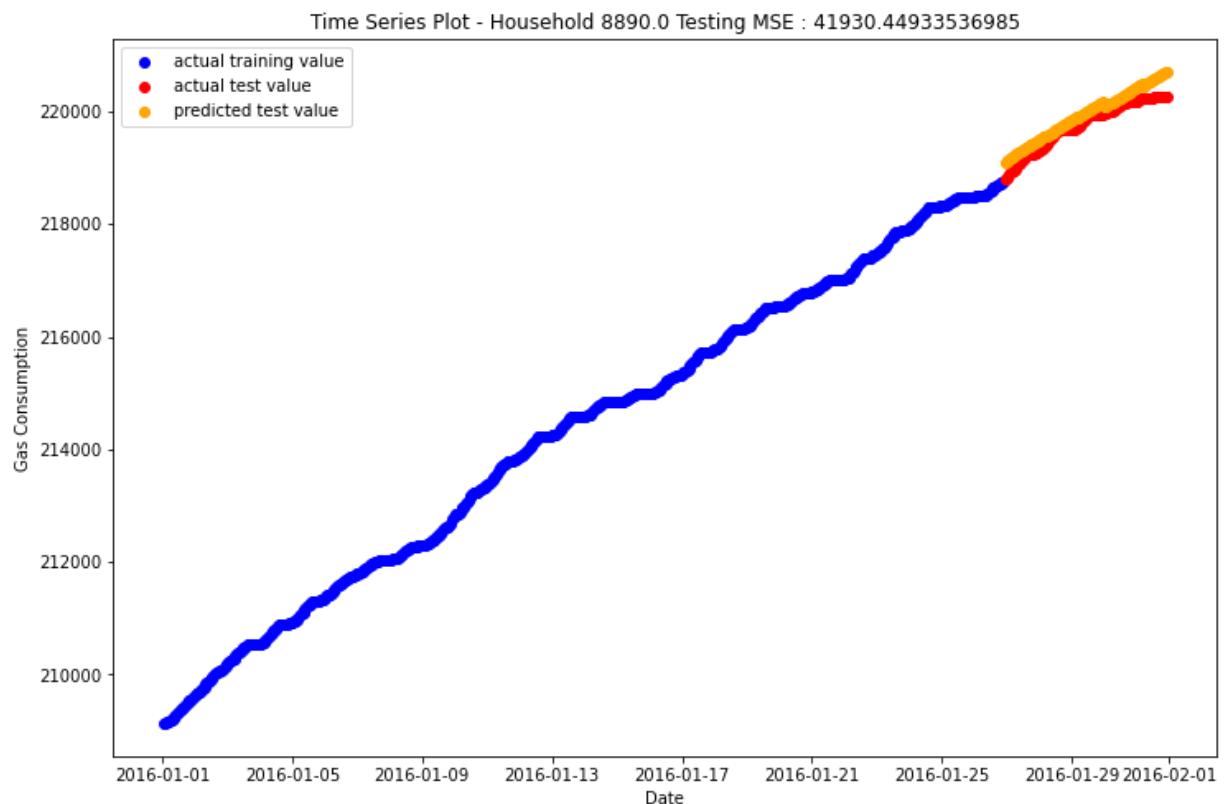
Time Series Plot - Household 8703.0 Testing MSE : 4036.74890933014

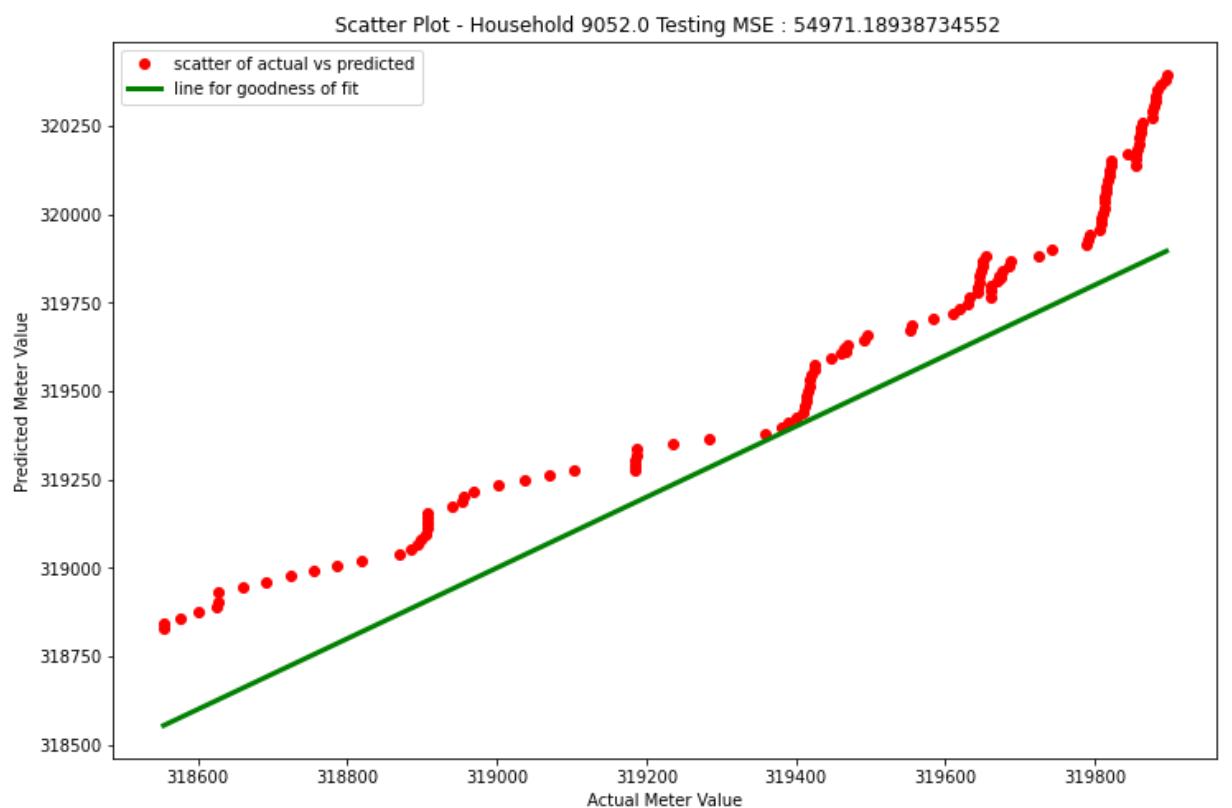
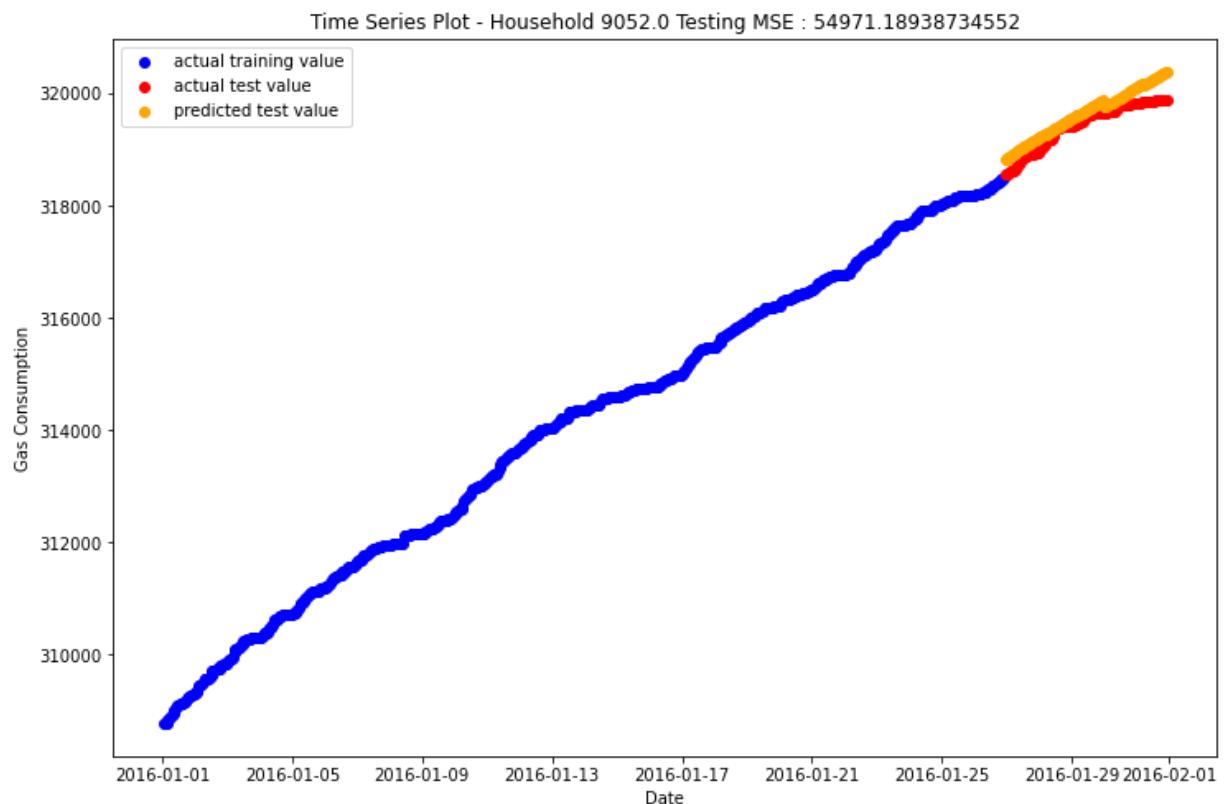


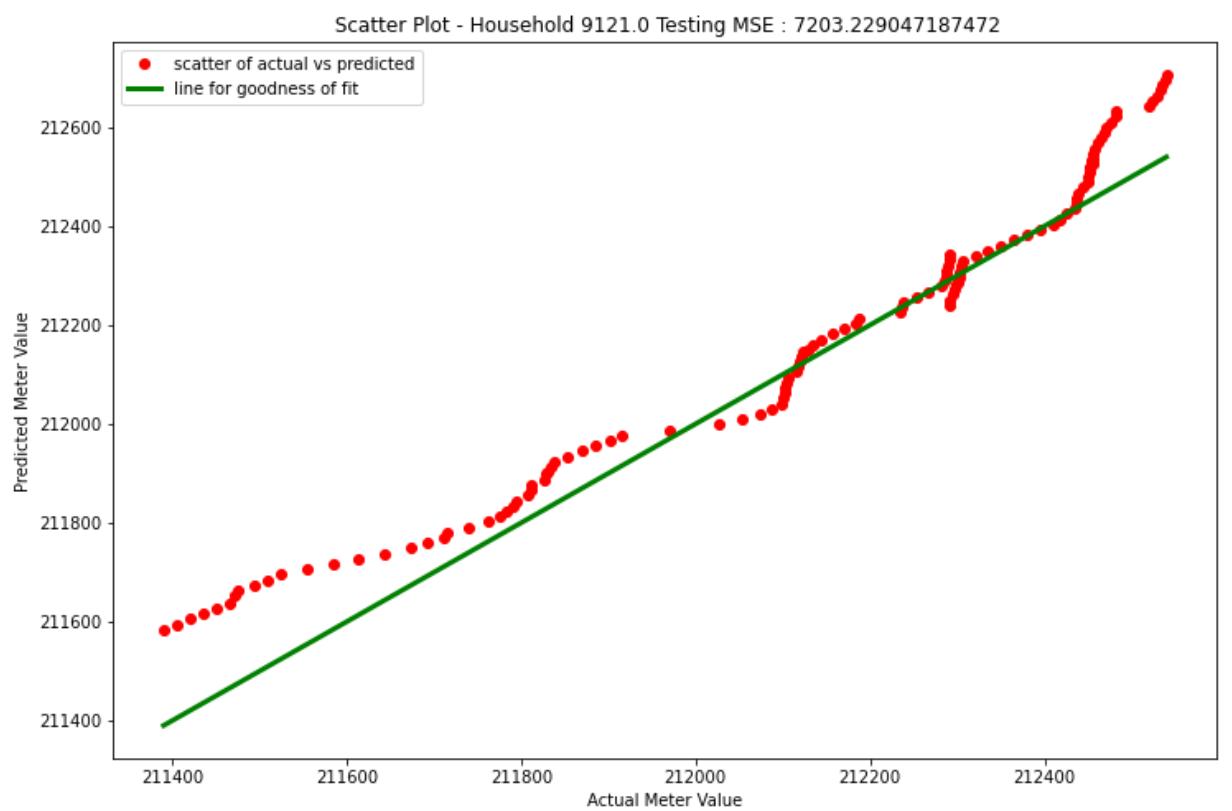
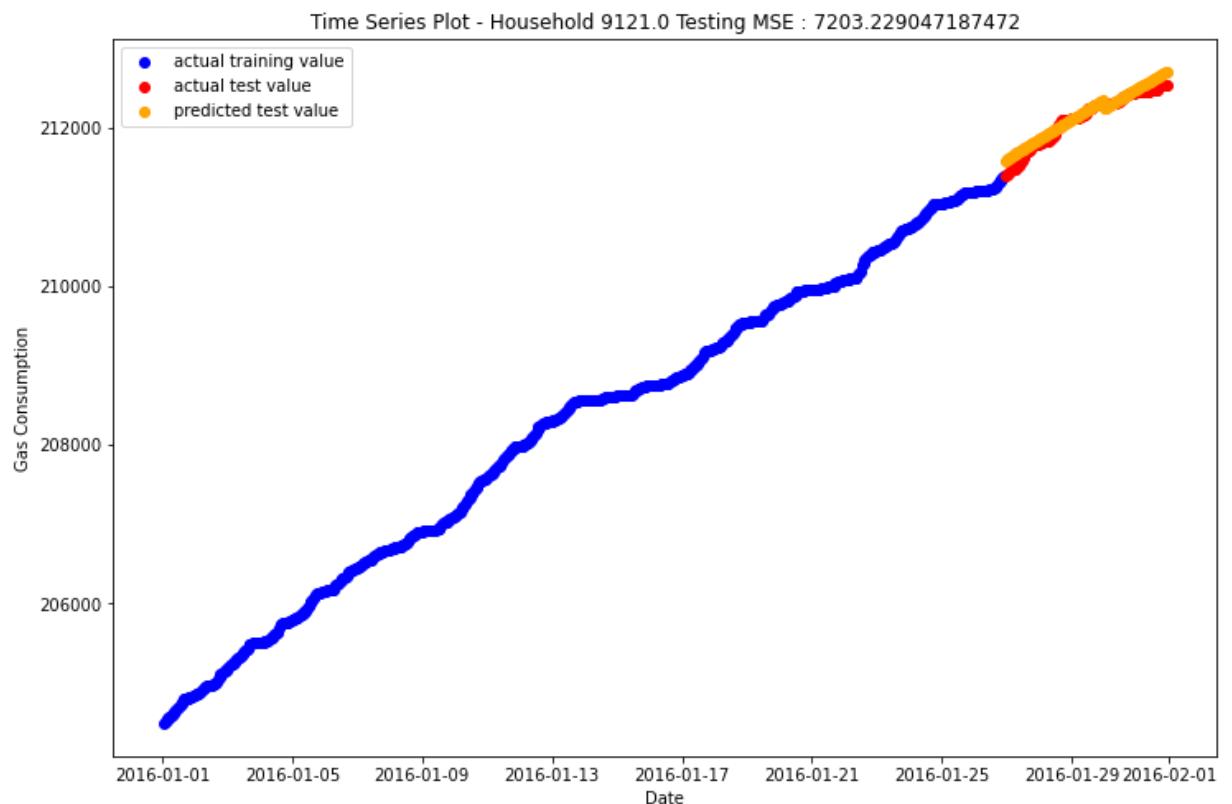
Scatter Plot - Household 8703.0 Testing MSE : 4036.74890933014

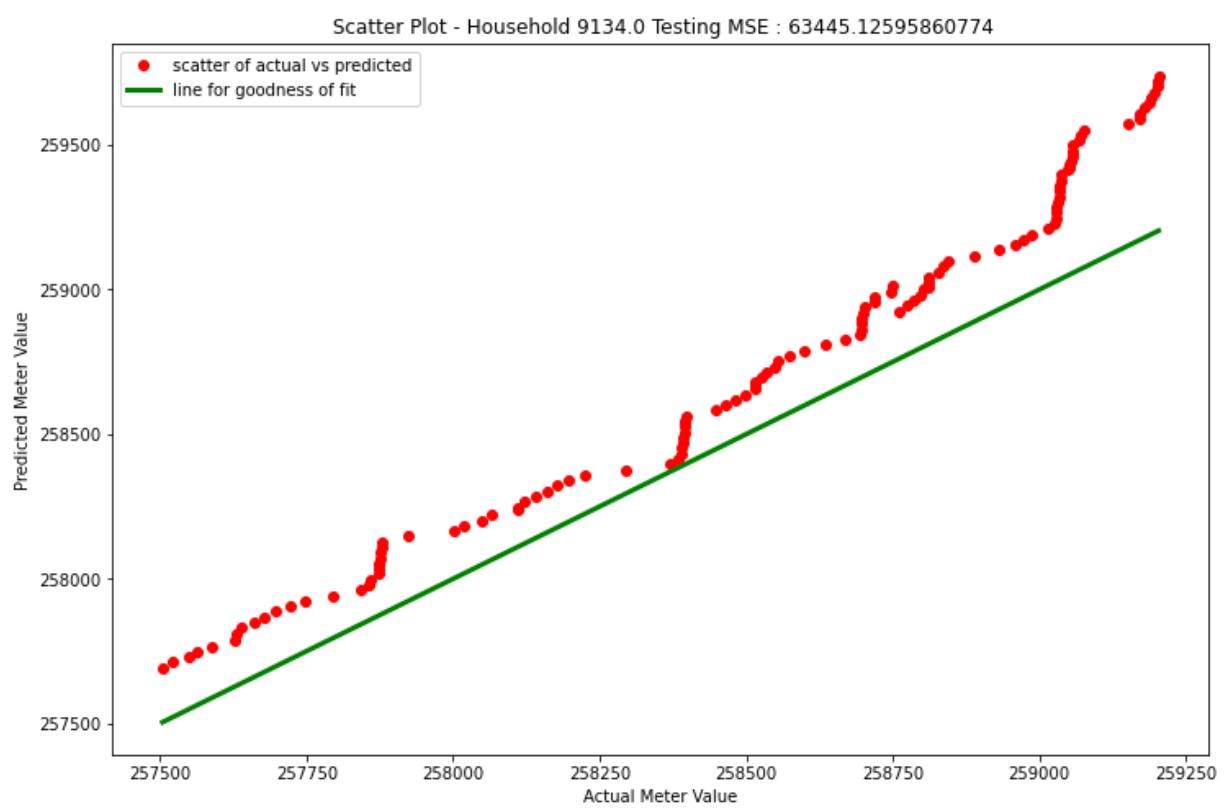
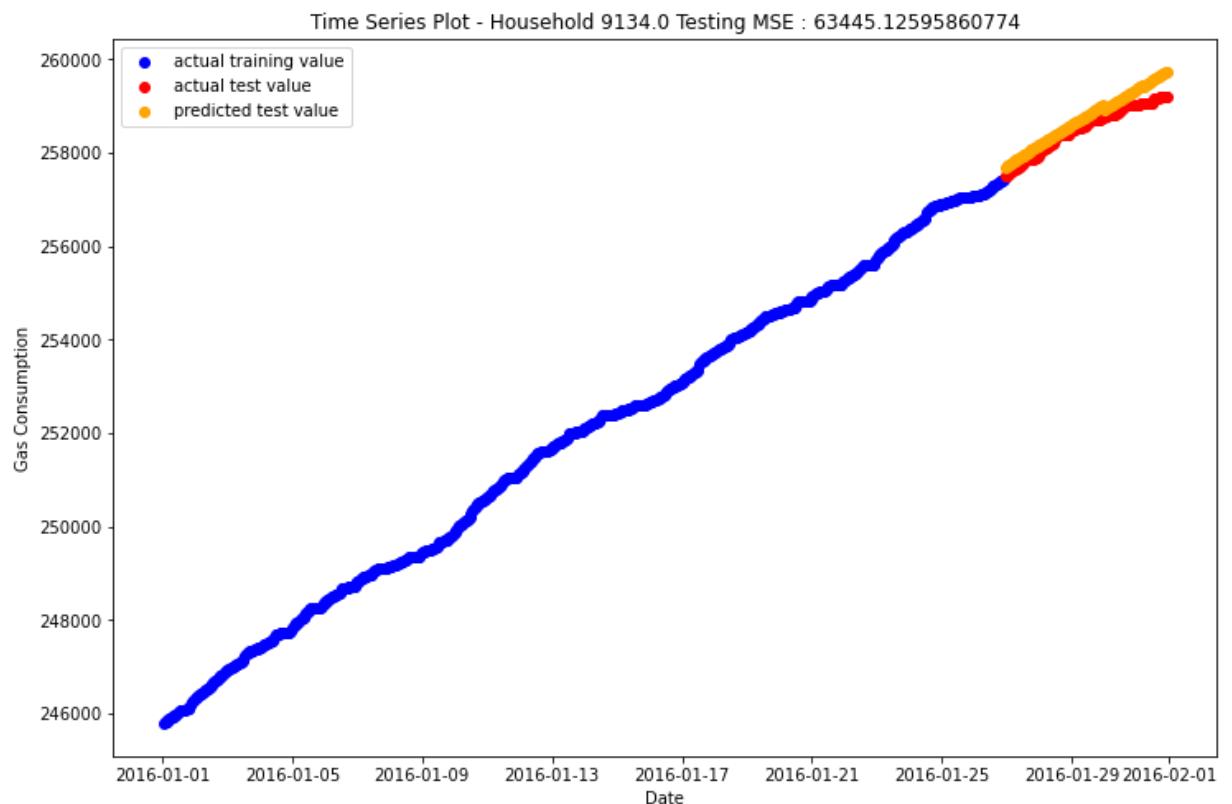




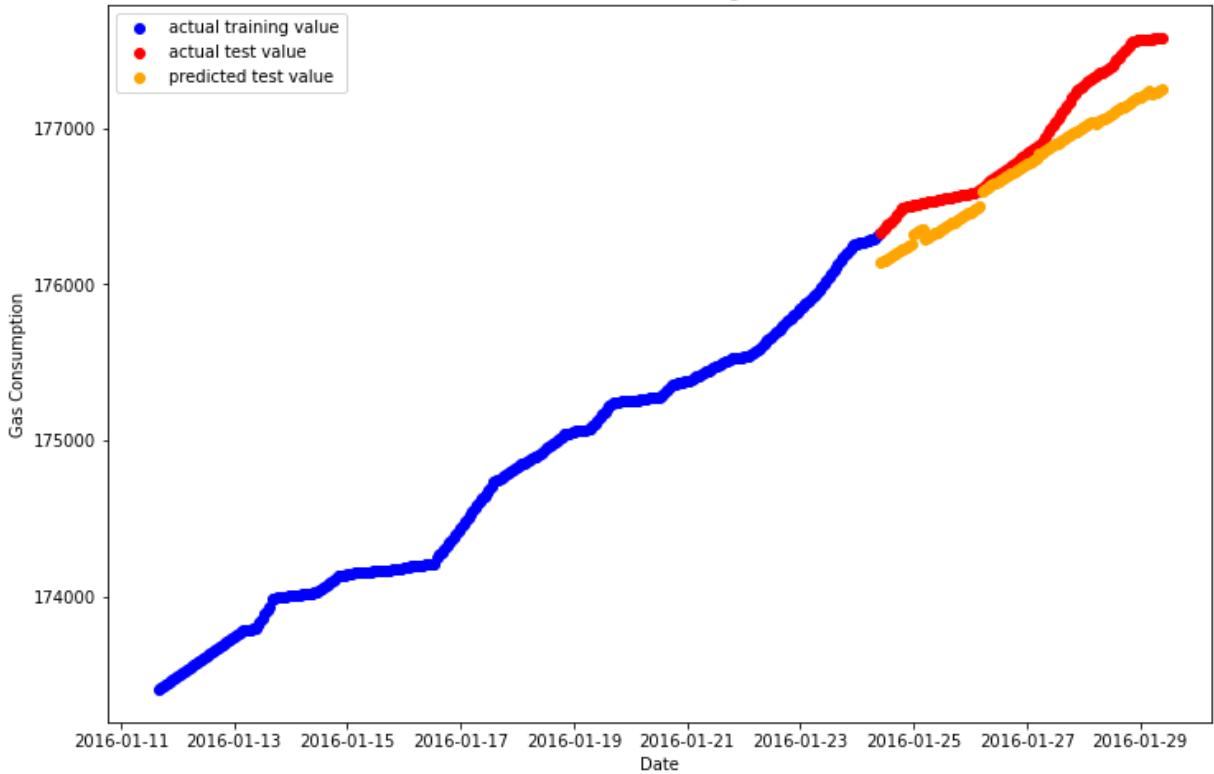




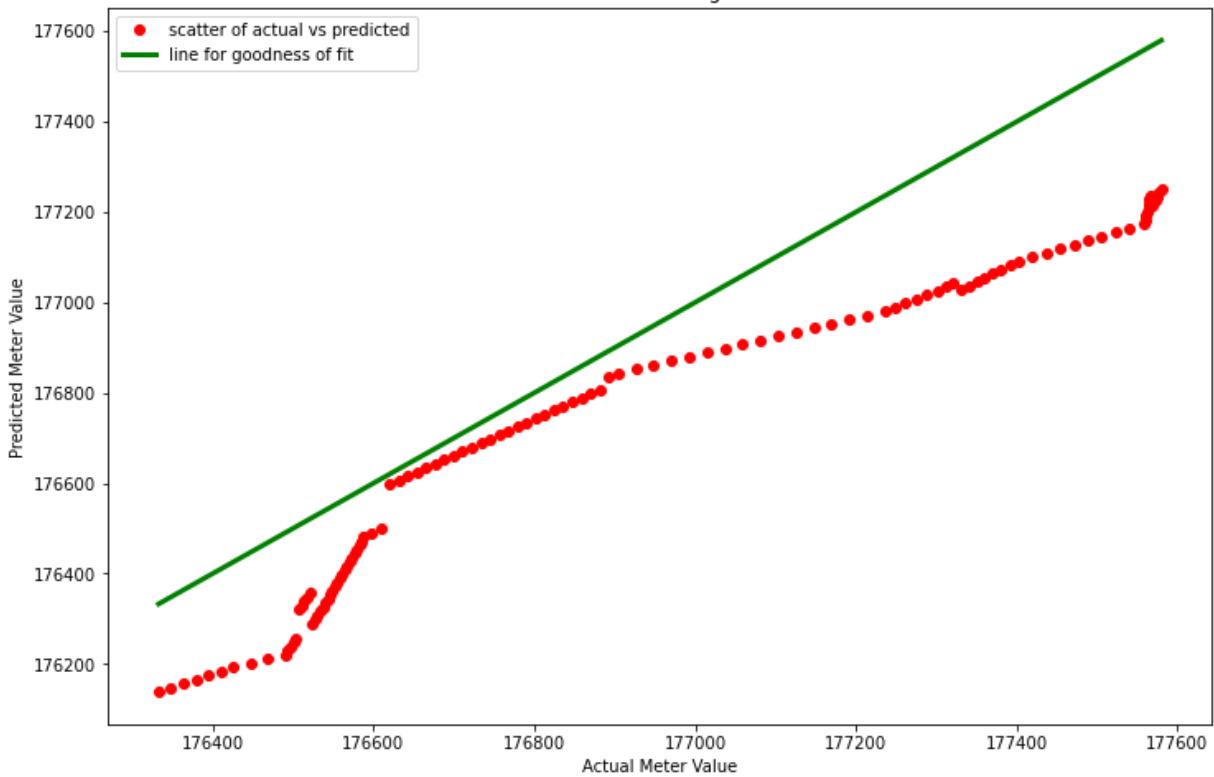


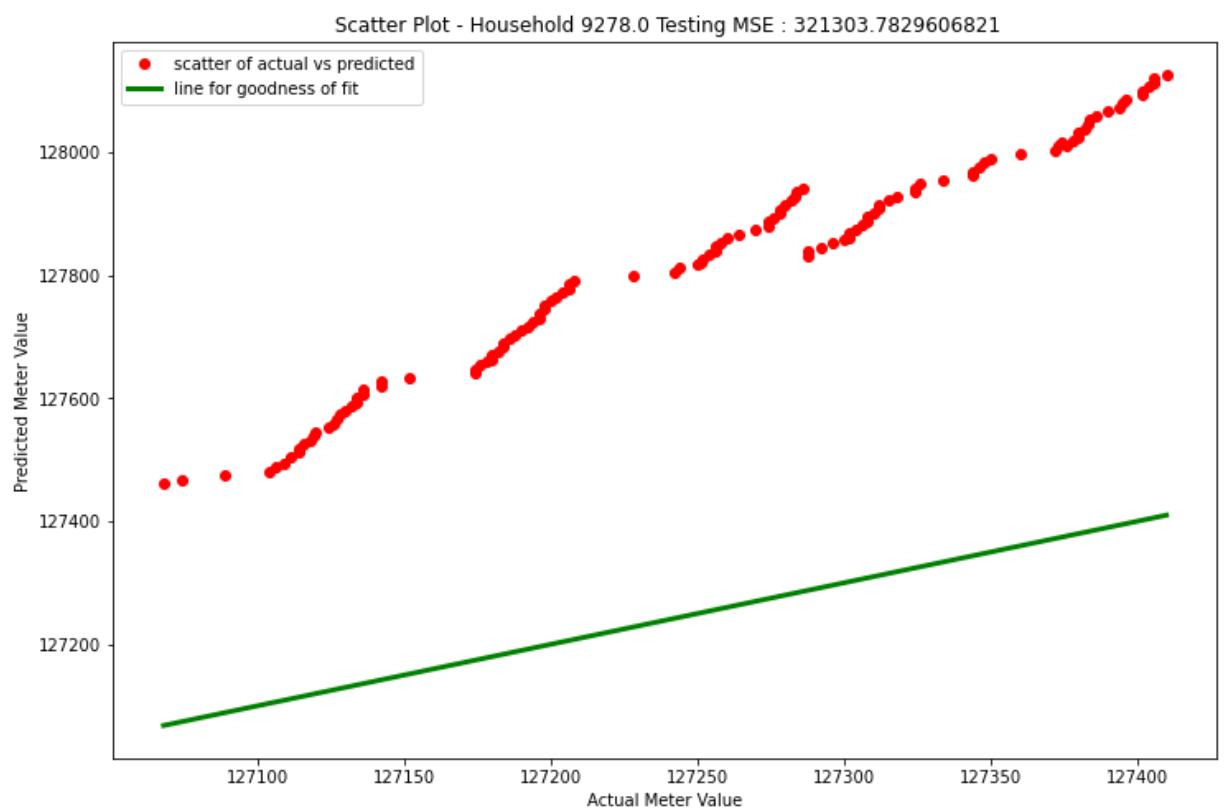
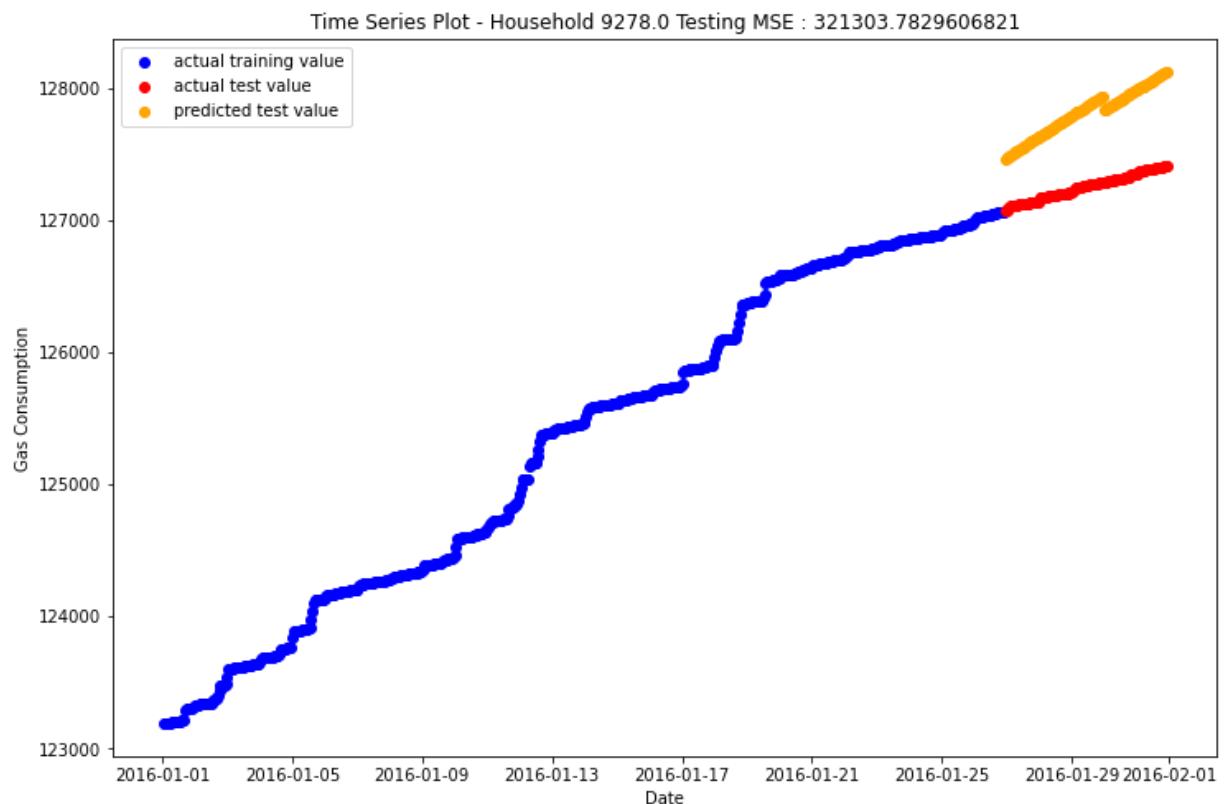


Time Series Plot - Household 9160.0 Testing MSE : 49844.29366581544

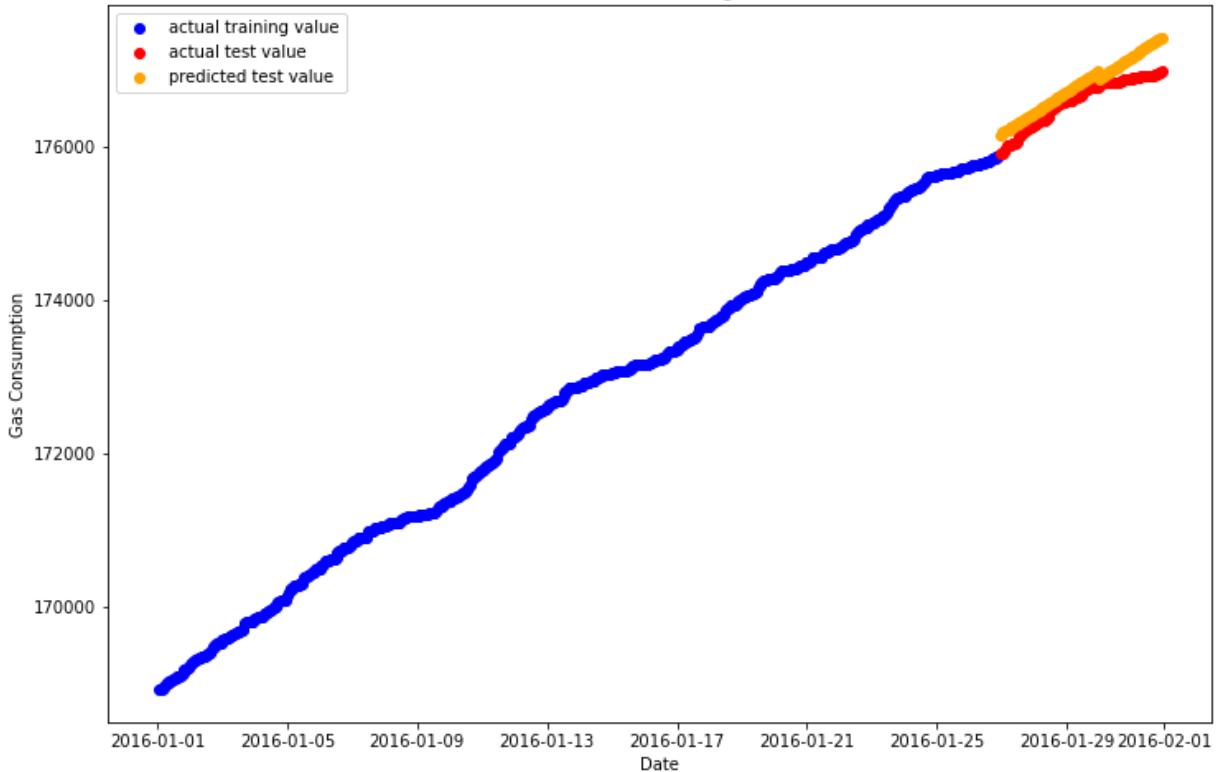


Scatter Plot - Household 9160.0 Testing MSE : 49844.29366581544

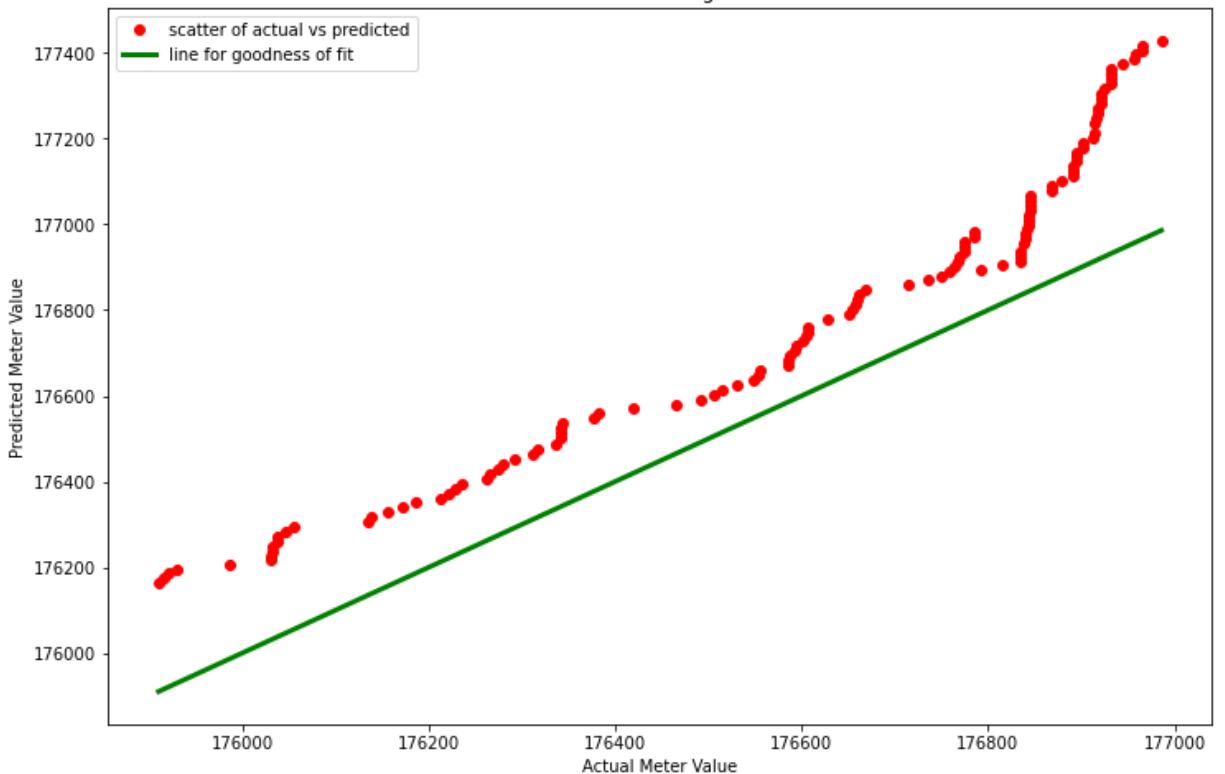


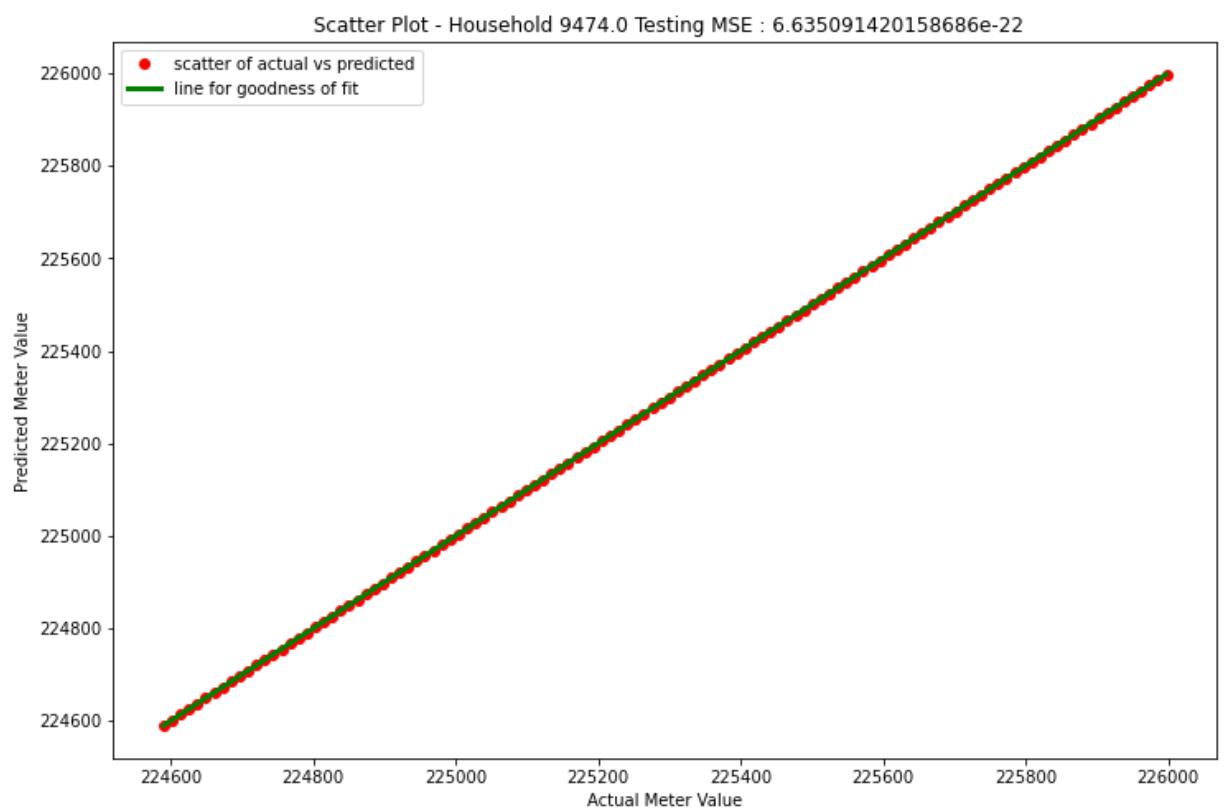
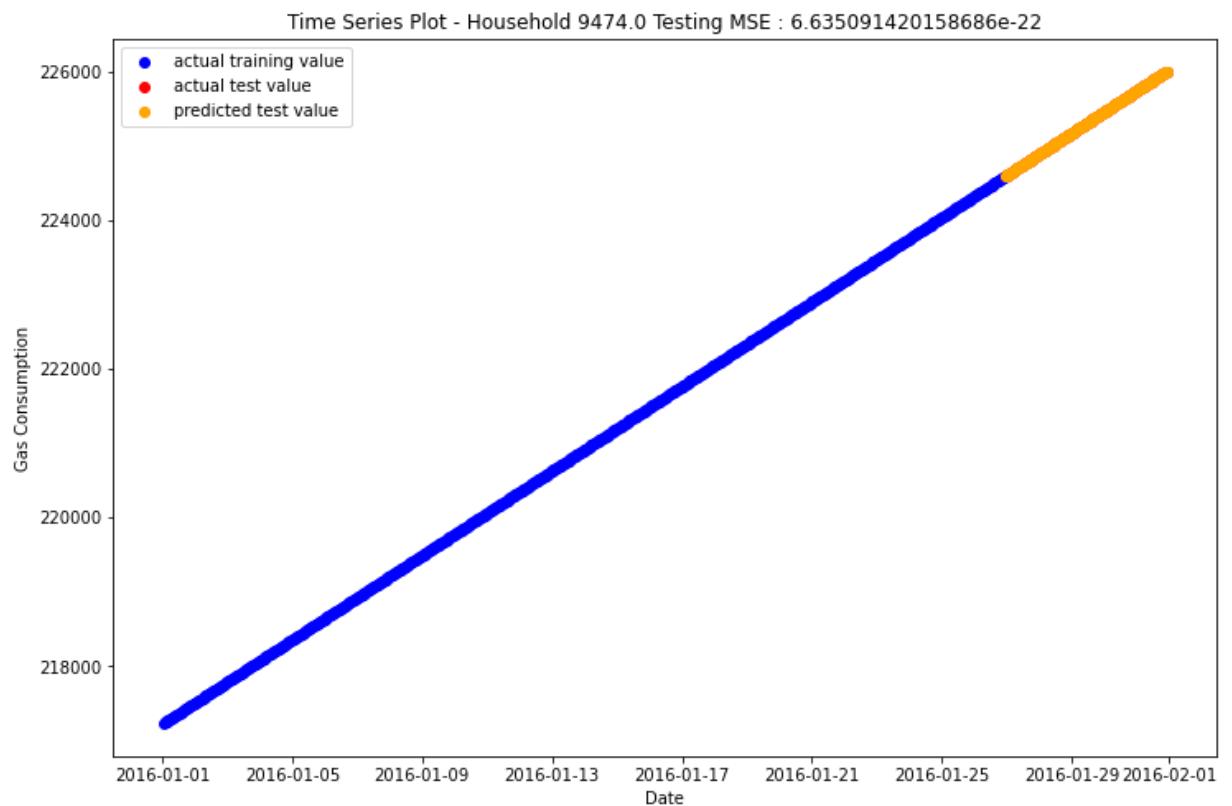


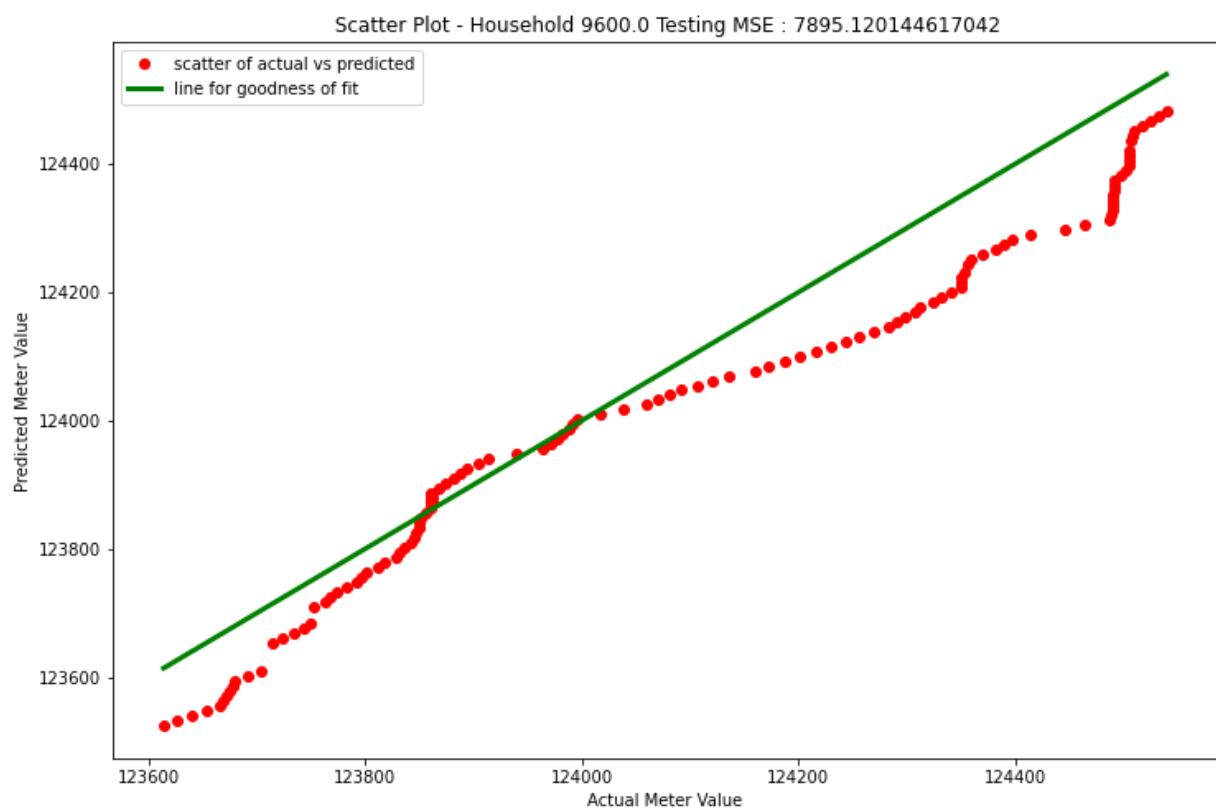
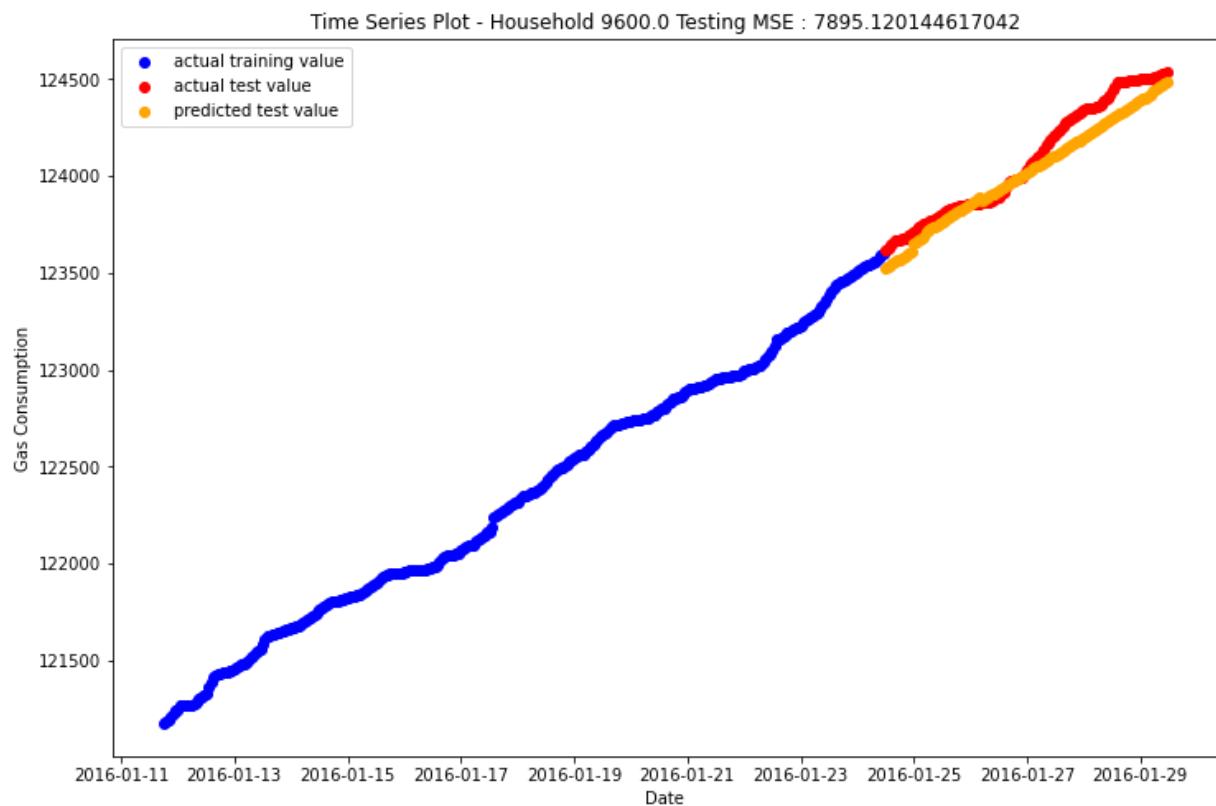
Time Series Plot - Household 9295.0 Testing MSE : 50920.476494250885

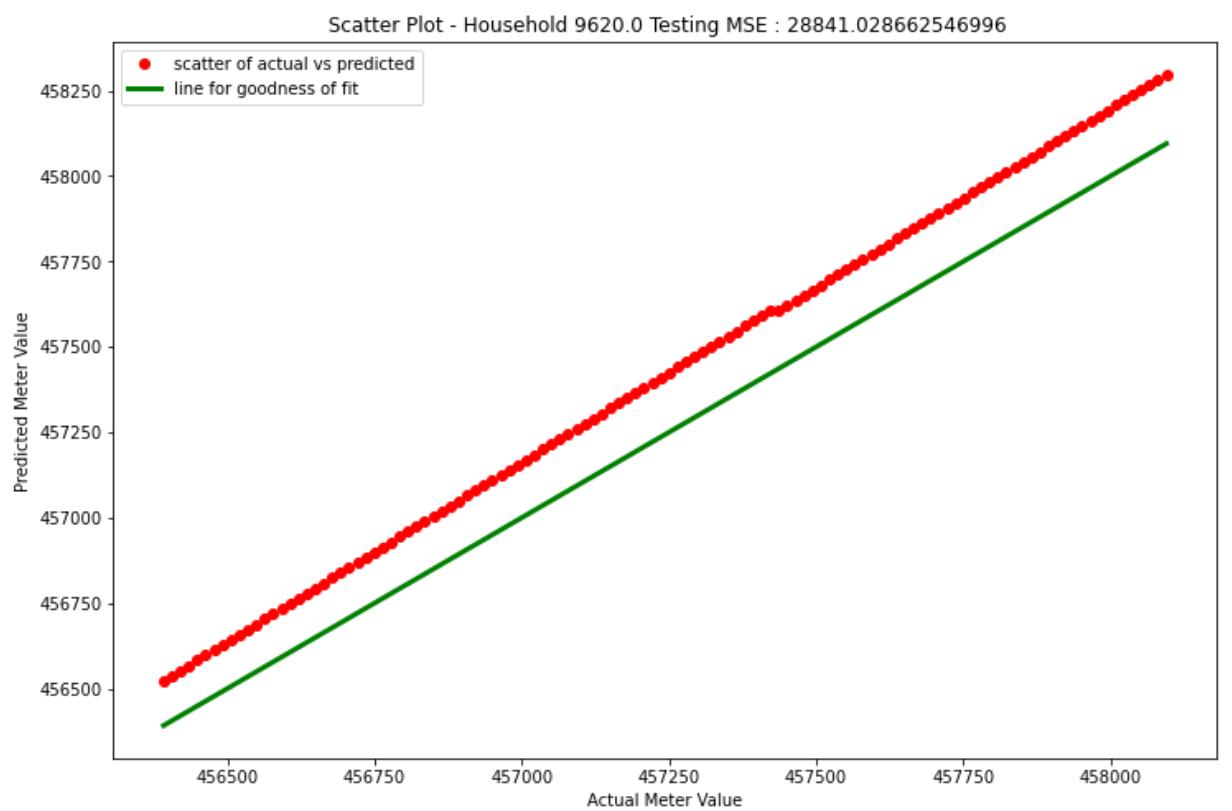
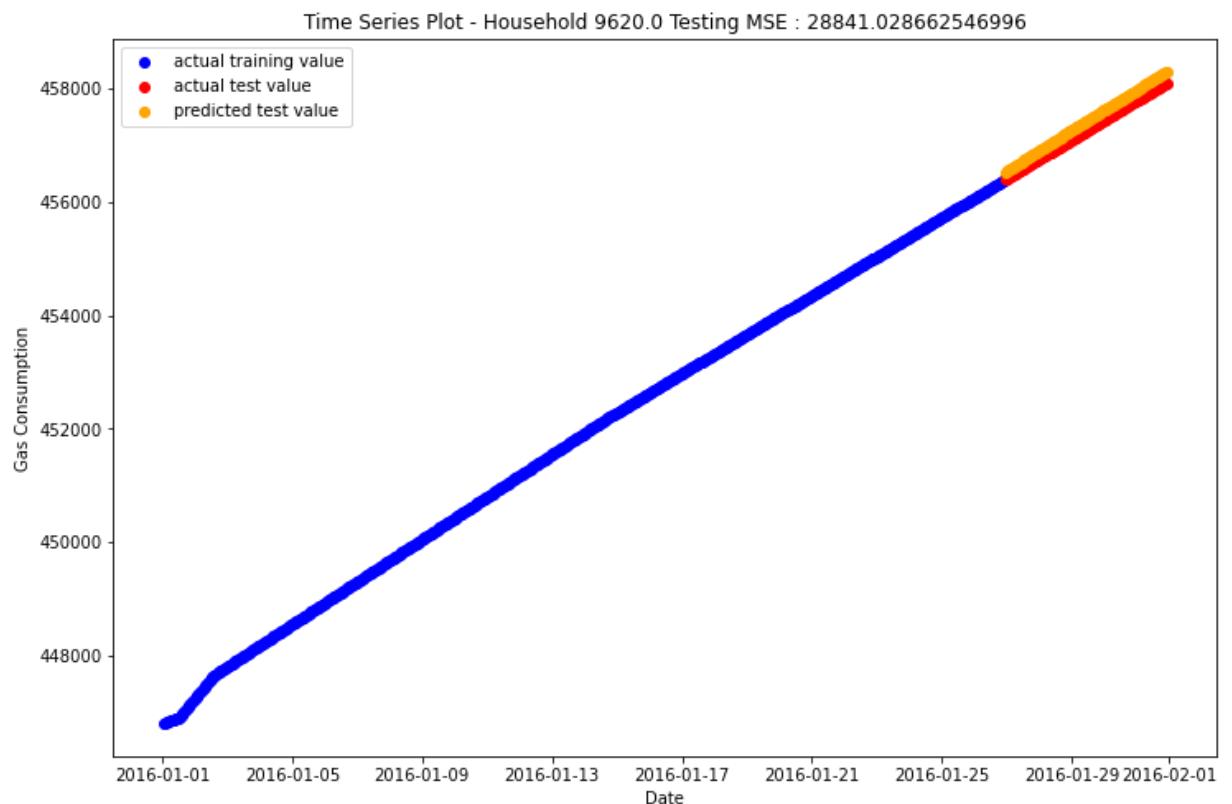


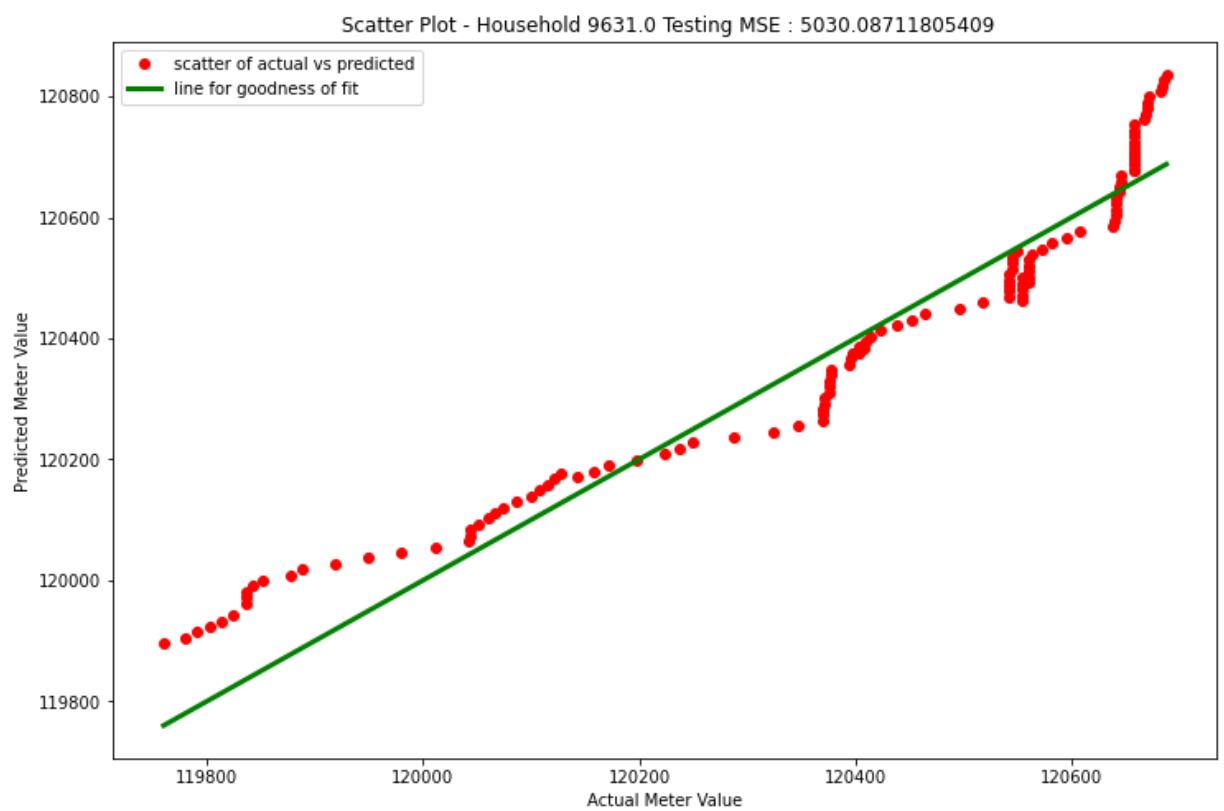
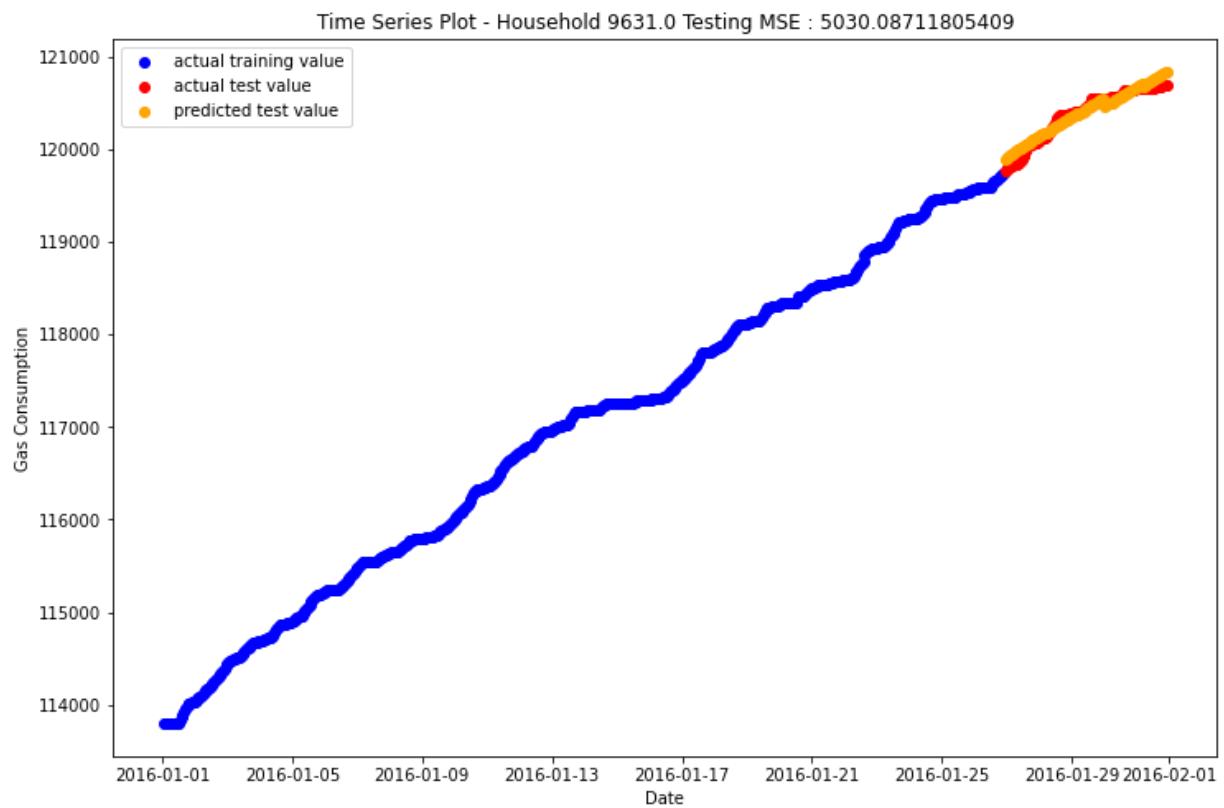
Scatter Plot - Household 9295.0 Testing MSE : 50920.476494250885

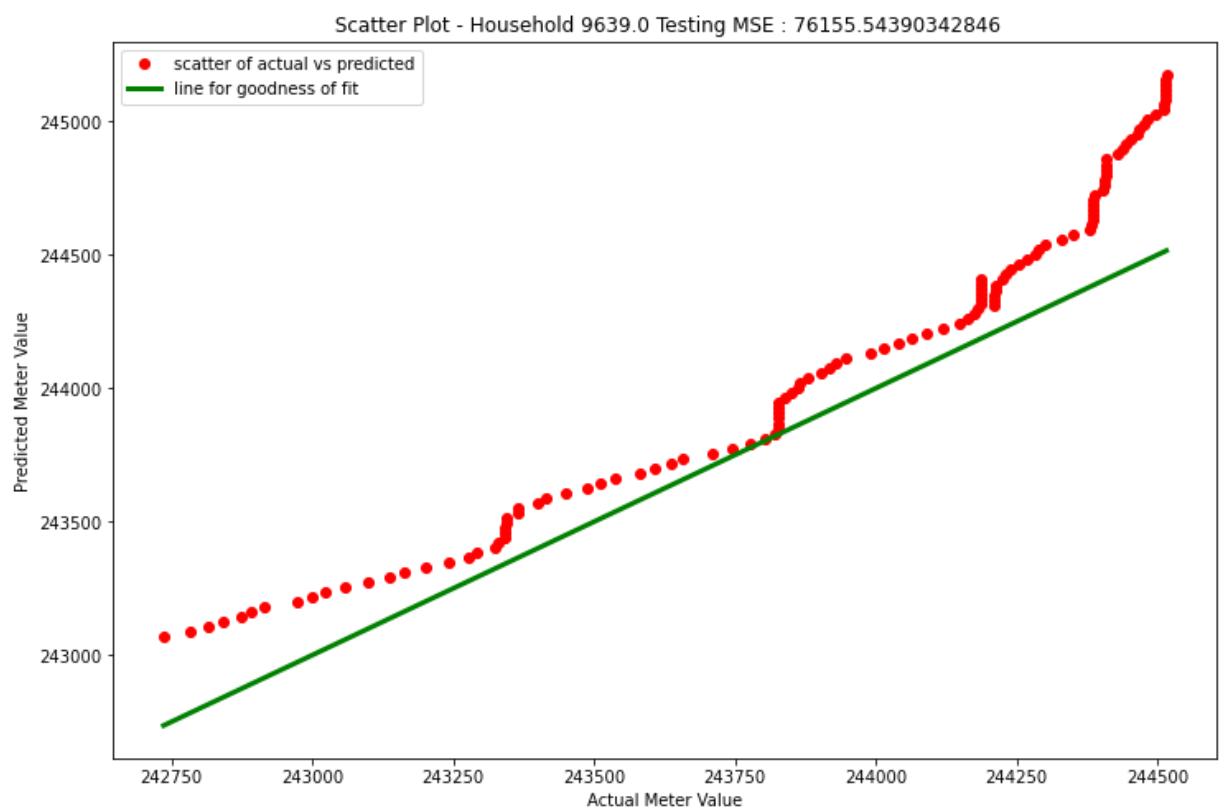
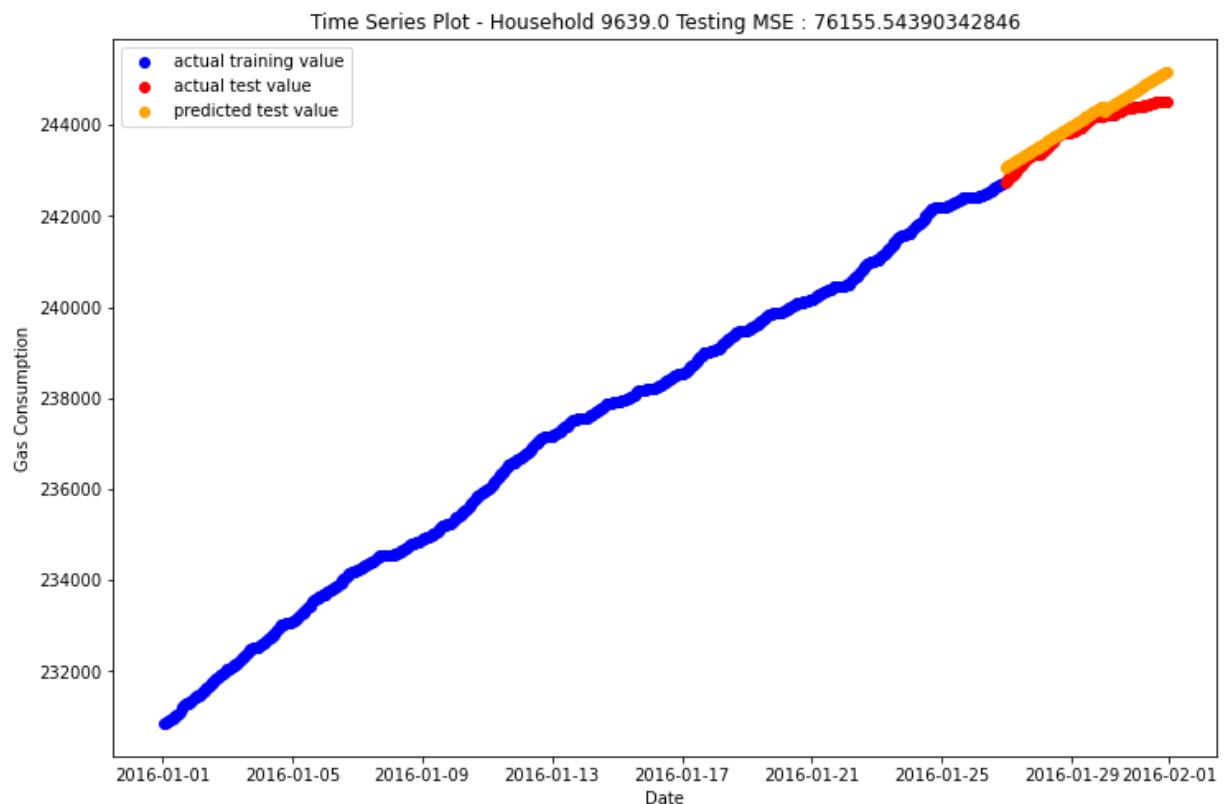


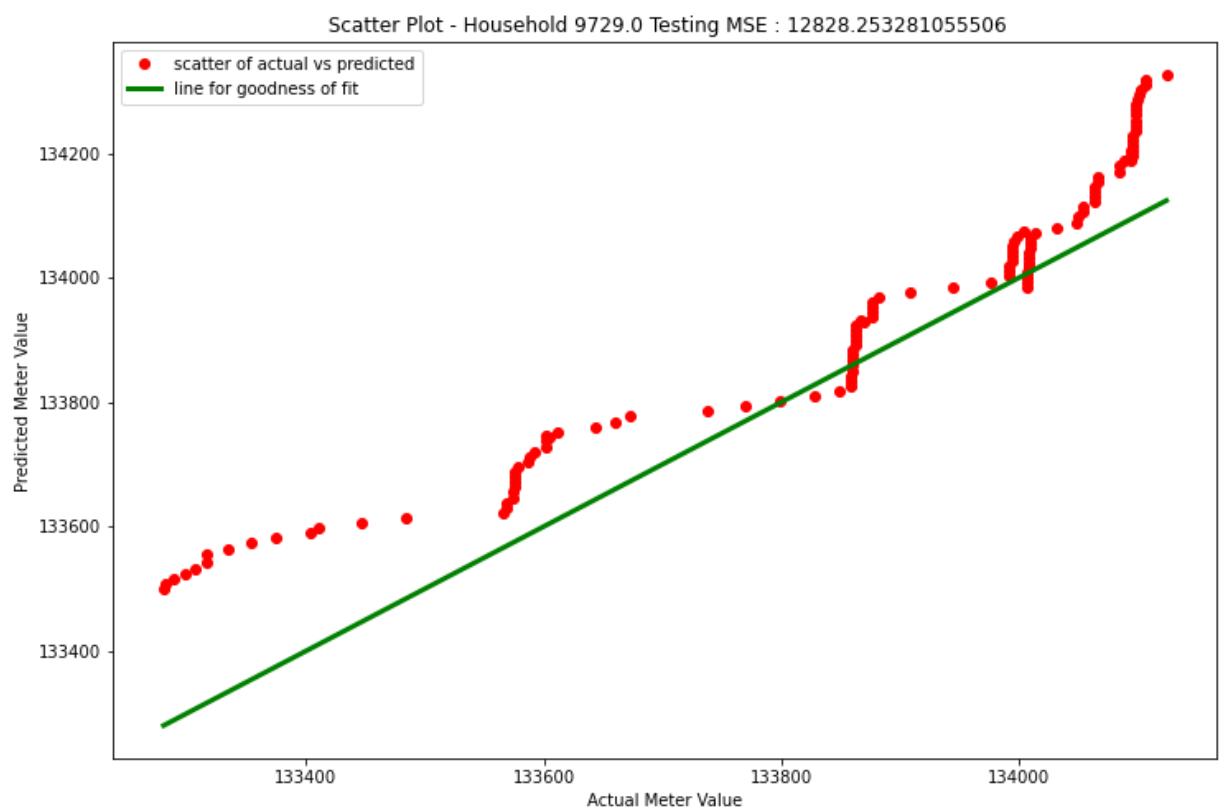
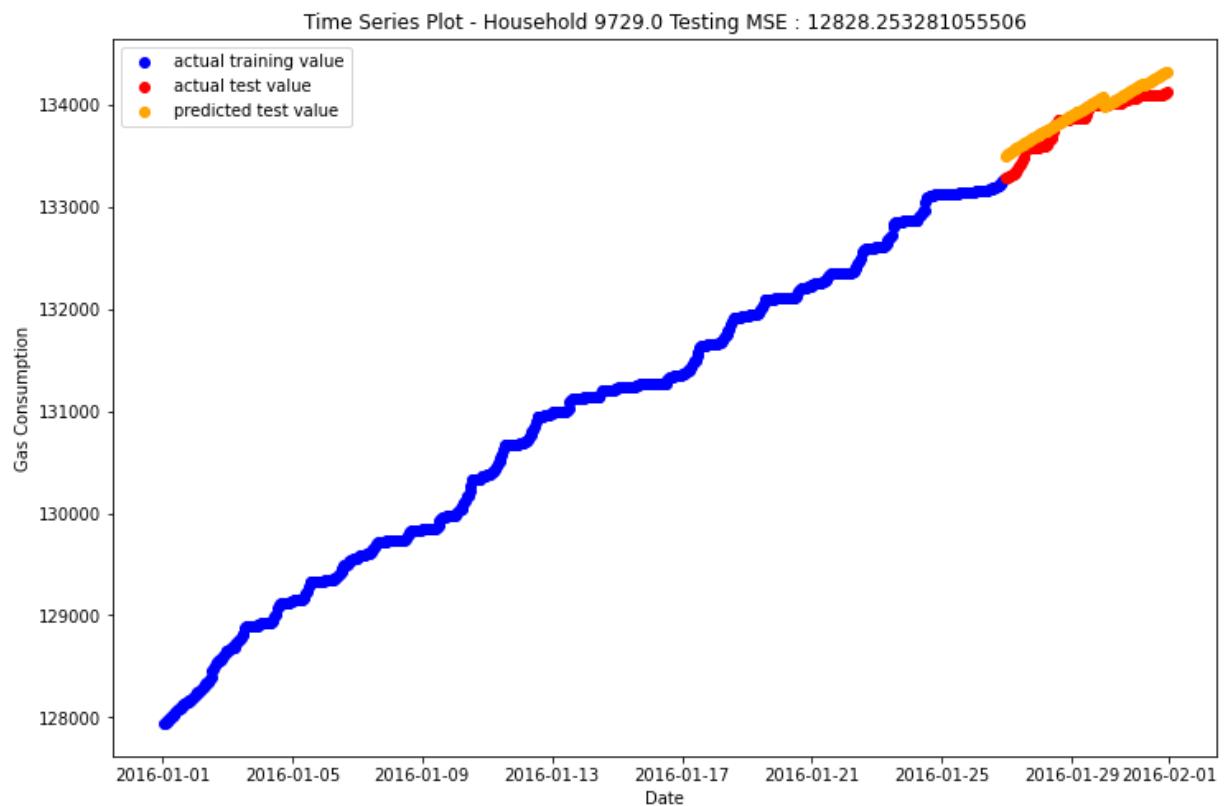


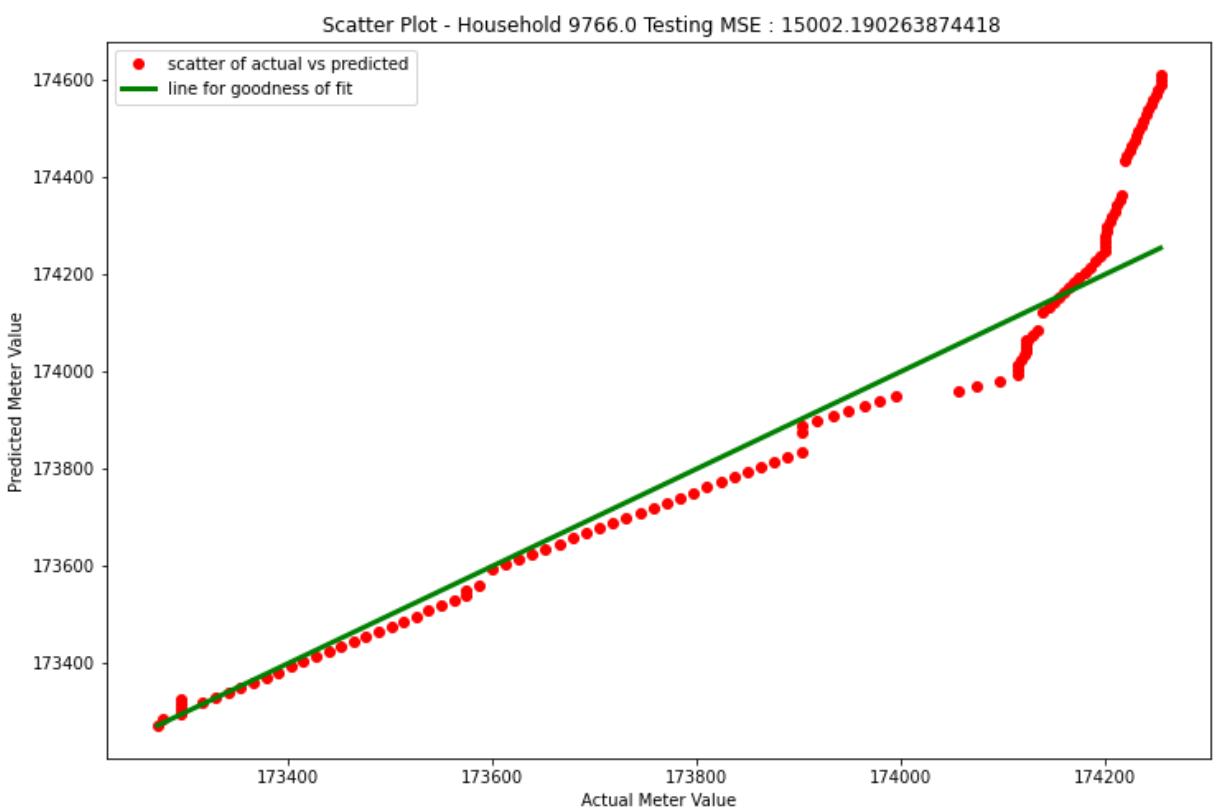
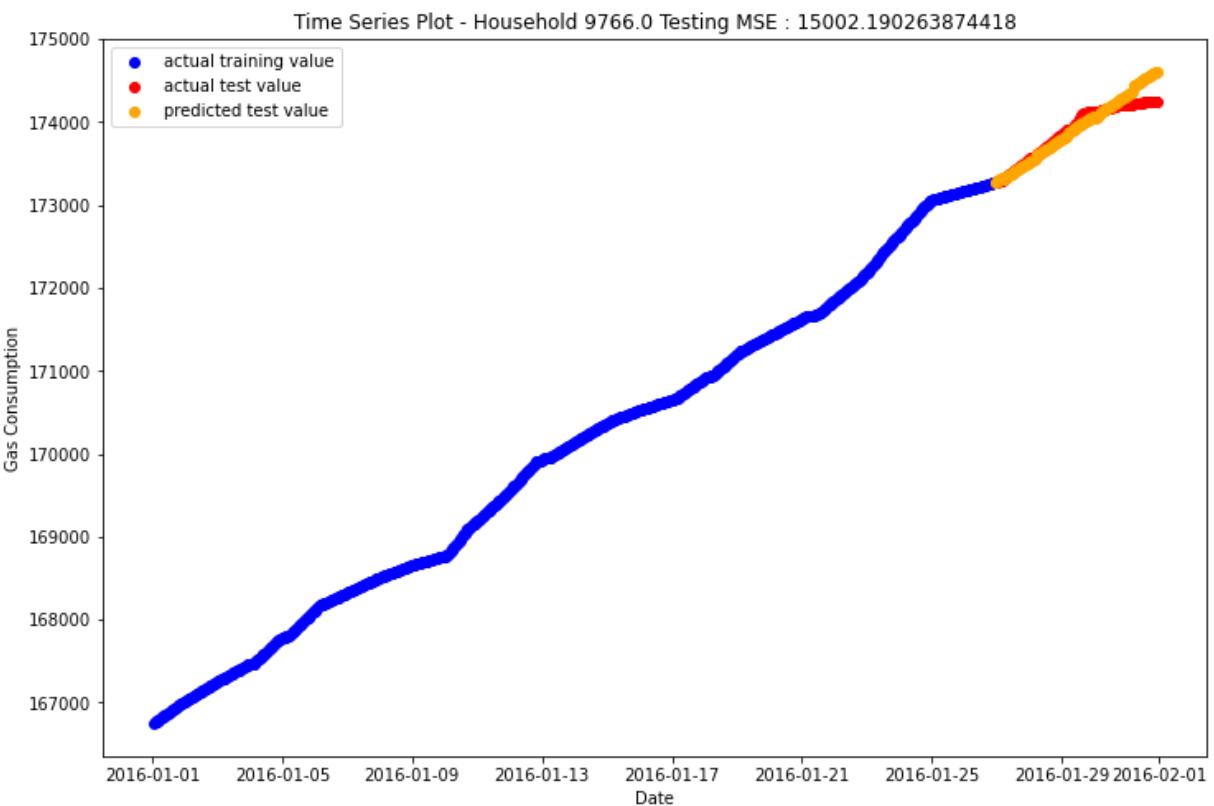


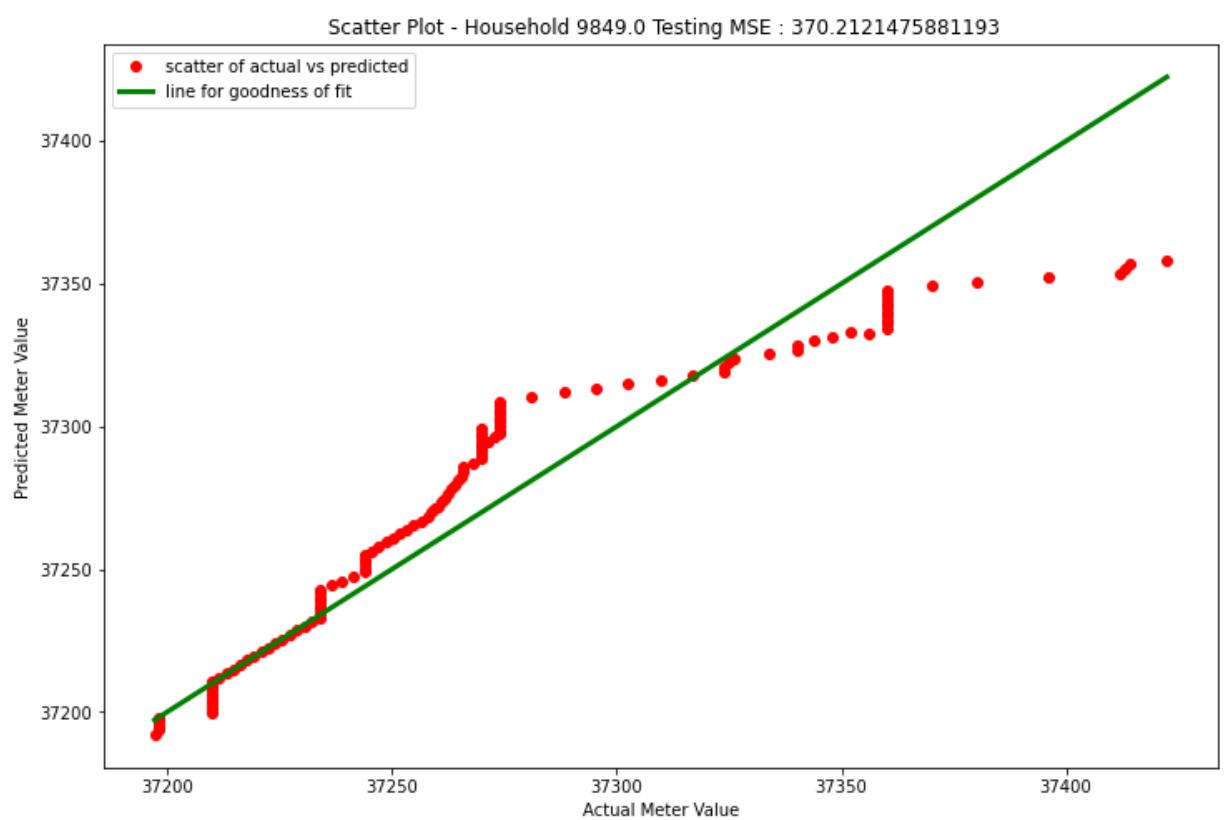
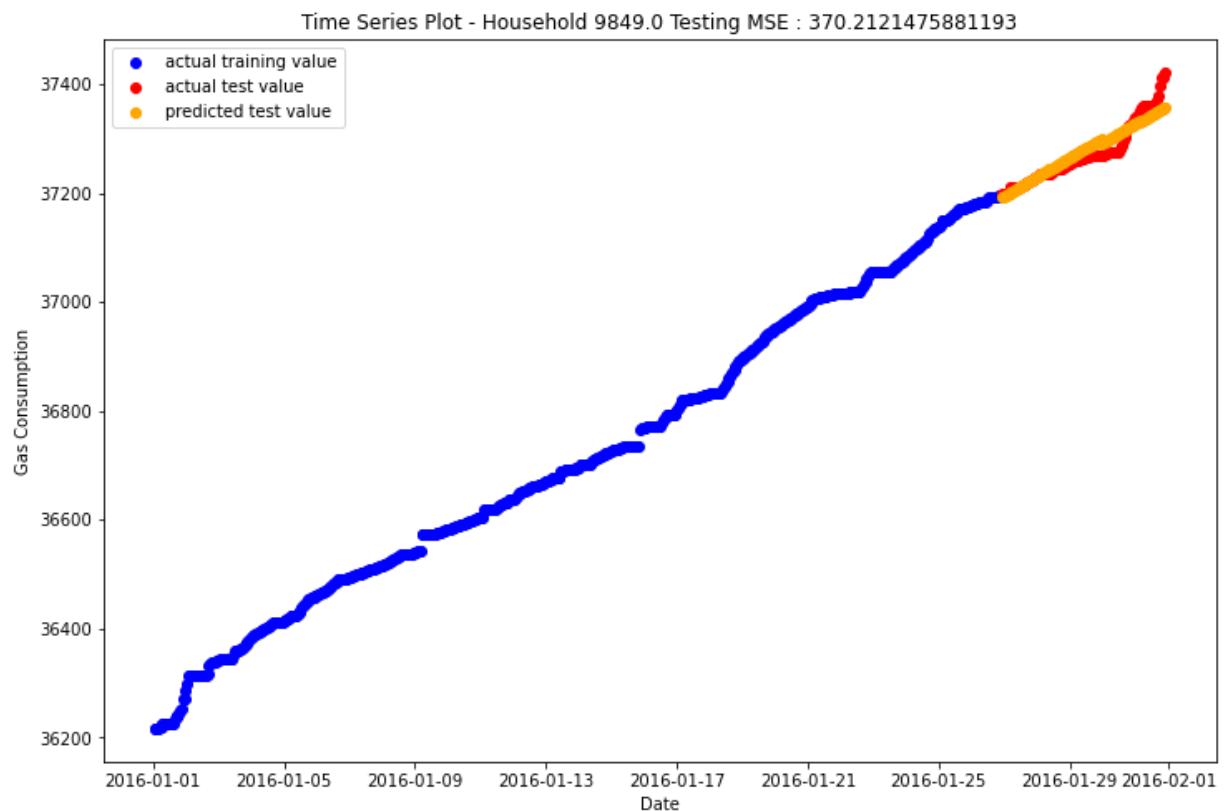


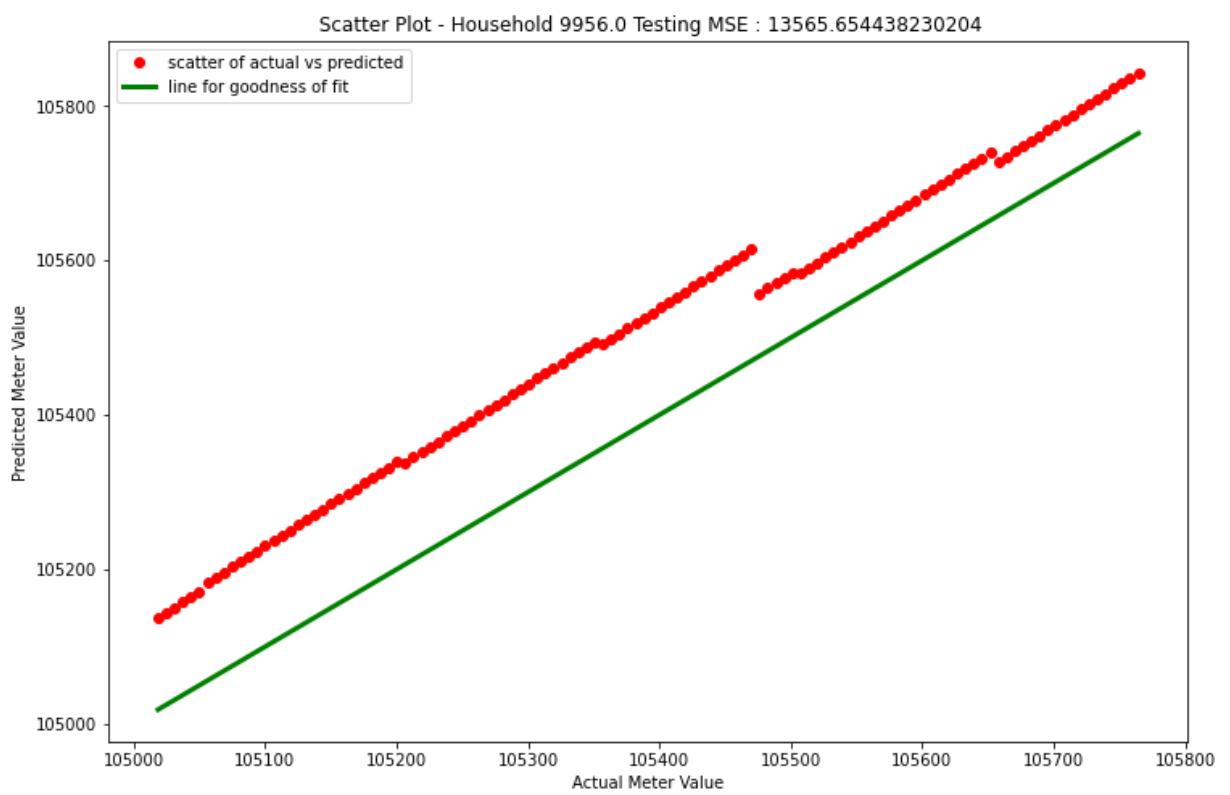
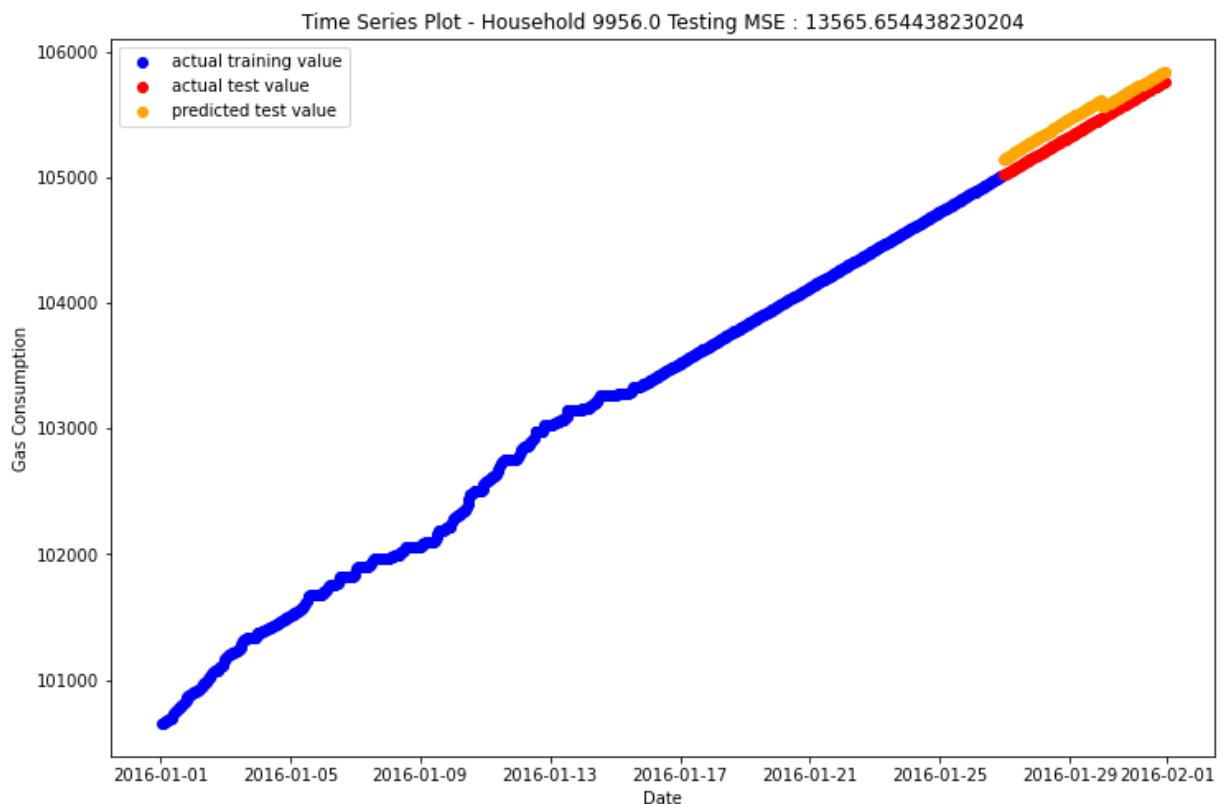


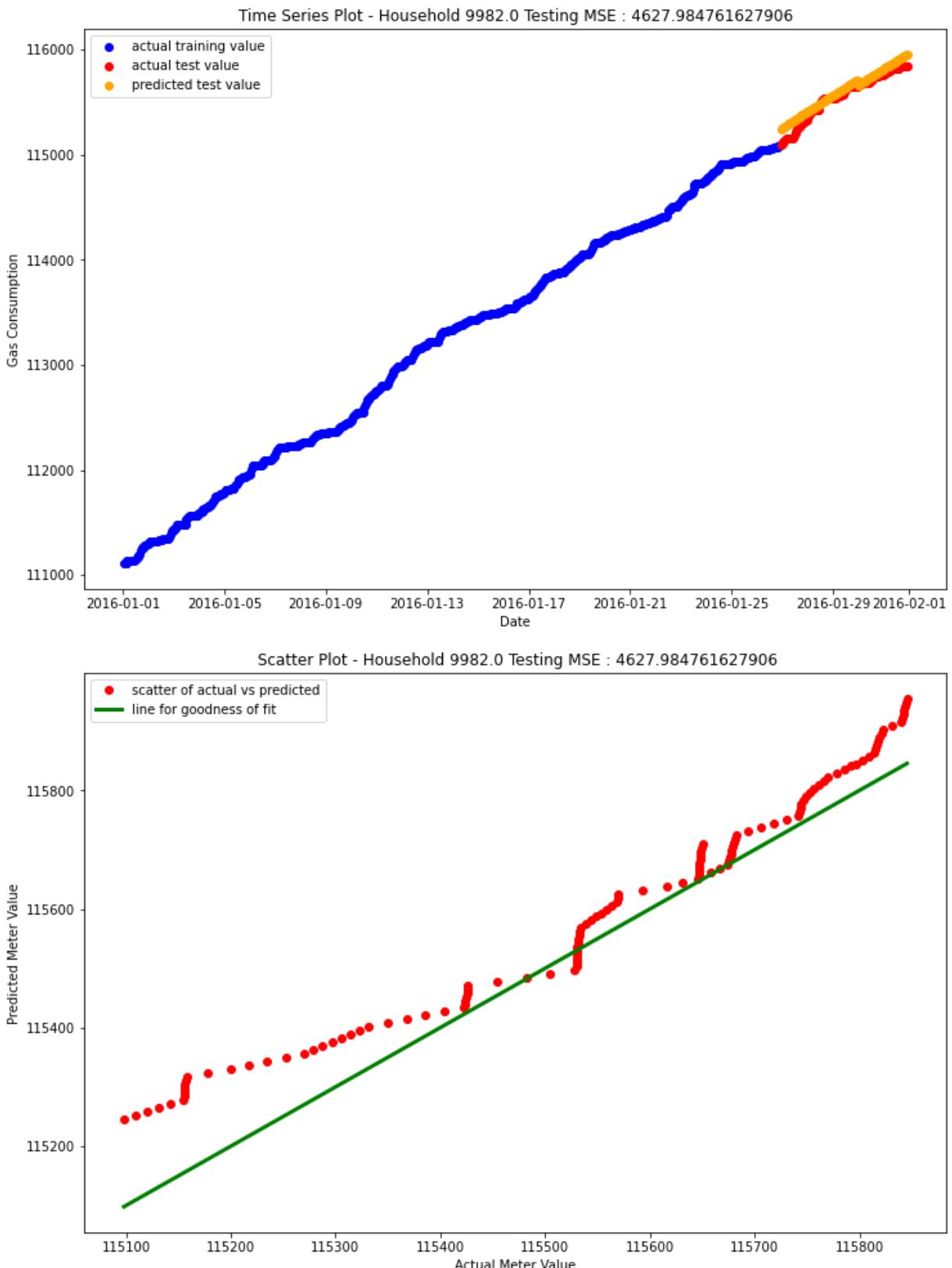












Above, we can see the timeseries and scatter plots for each of the households. The respective testing MSE value (derived from the testing dataset for each respective household) has been calculated and reflected at the title of each plot. Upon close inspection and observation, the obtained MSE values are not significantly lower than what we obtained for question 2 without the weather attributes.

With that, let's now build a neural network with the same attributes (including the weather attributes) to see whether we are able to achieve better performance.

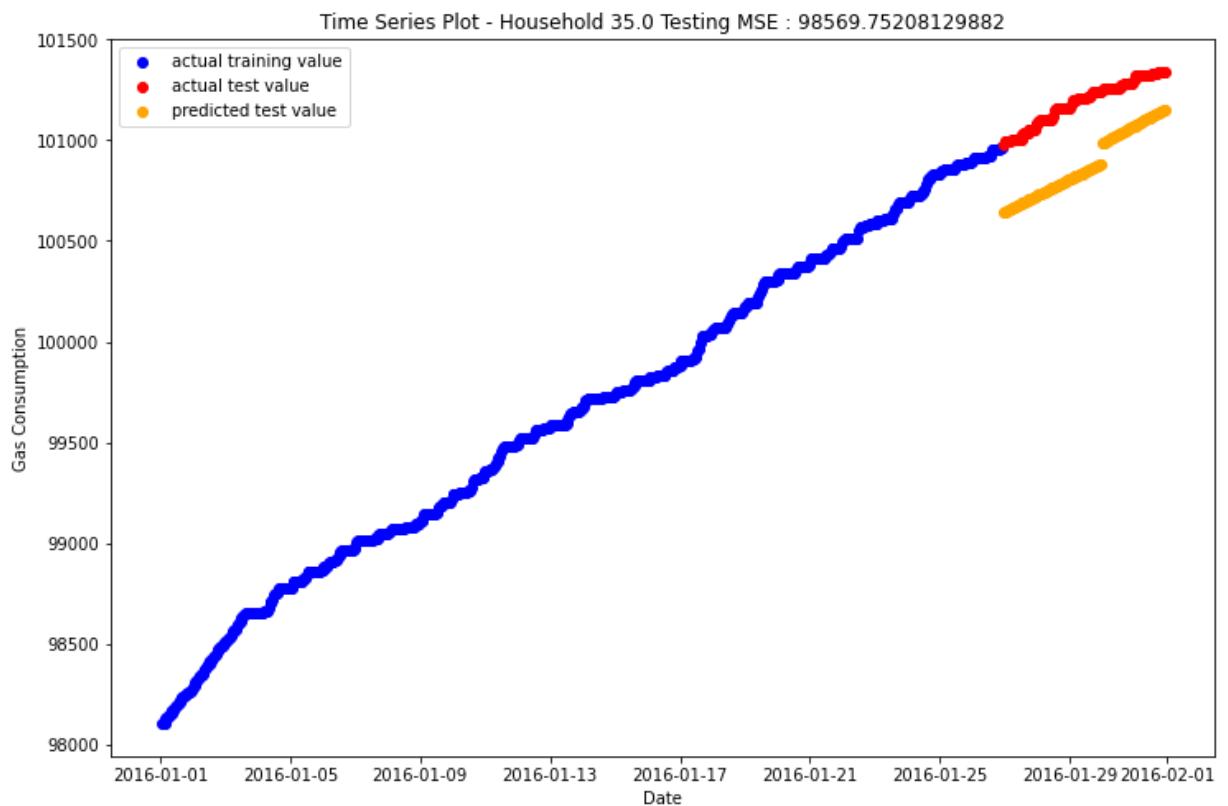
#### 4. Build a neural network and evaluate its accuracy.

For simplicity, we will be using a two-layer neural network with Rectify Linear Unit (ReLU) as the activation function and "Adam" as the optimizer. We chose ReLU as it is more computationally efficient to compute and has better convergence performance than sigmoid. We decided to use "Adam" optimizer as it is regarded as one of the best optimizers because it provides an optimization algorithm that can handle sparse gradients on noisy problems and hence gives much higher performance.

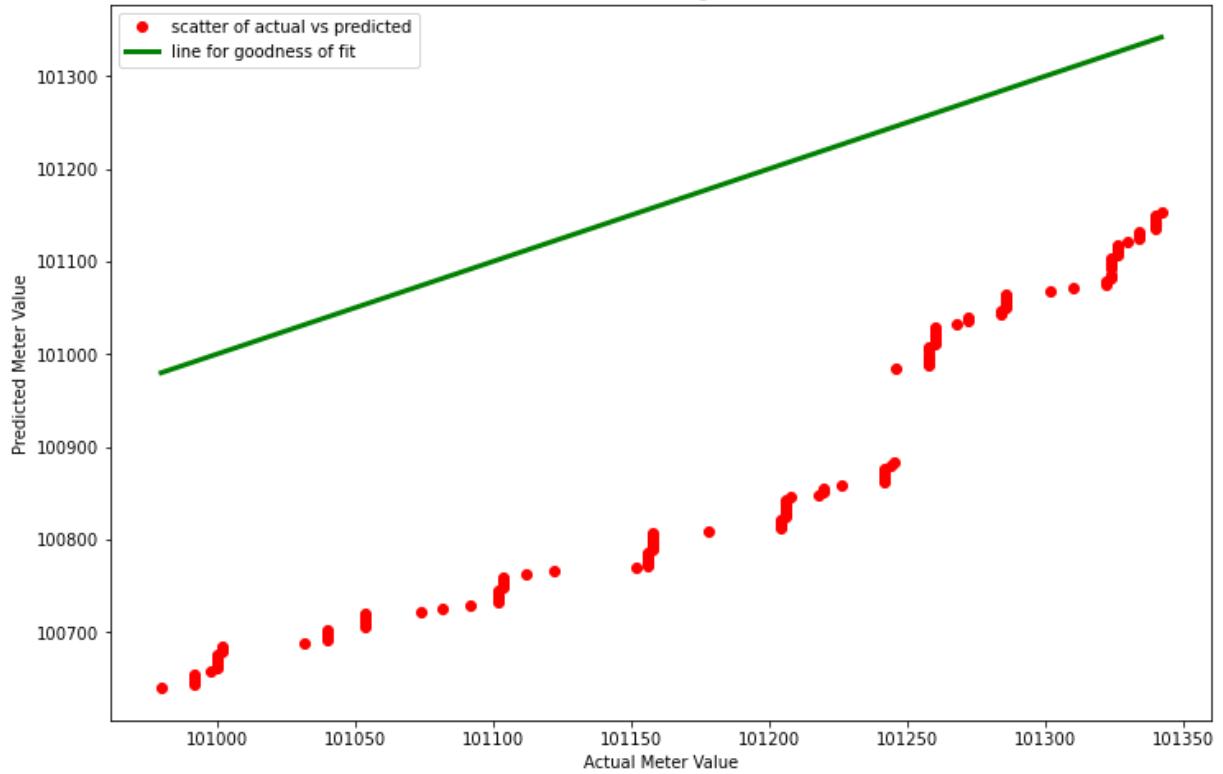
Below we will run the algorithm to create neural networks for each household and then predict meter values for the testing dataset. Then, we will plot the timeseries and scatter plots to compare the predicted values against the actual meter readings for easier ease of observation.

In [113]:

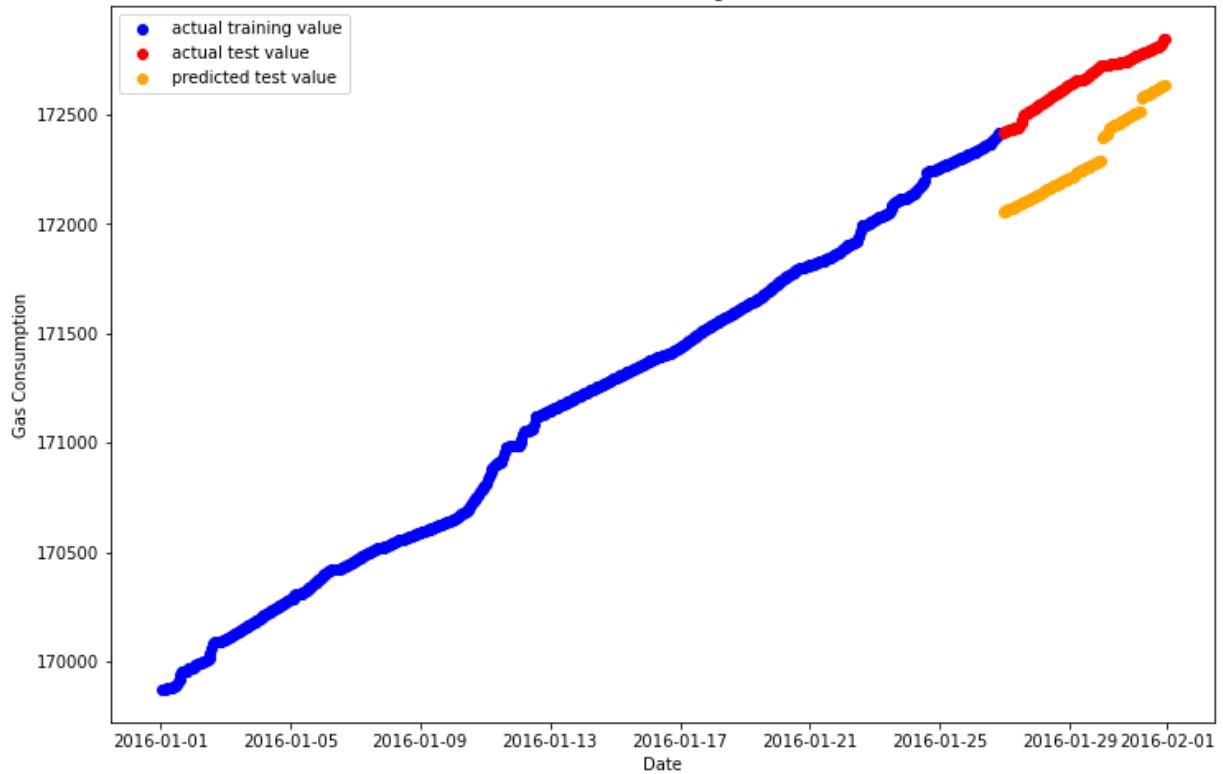
```
train_each_house_nn(hours_to_predict)
```

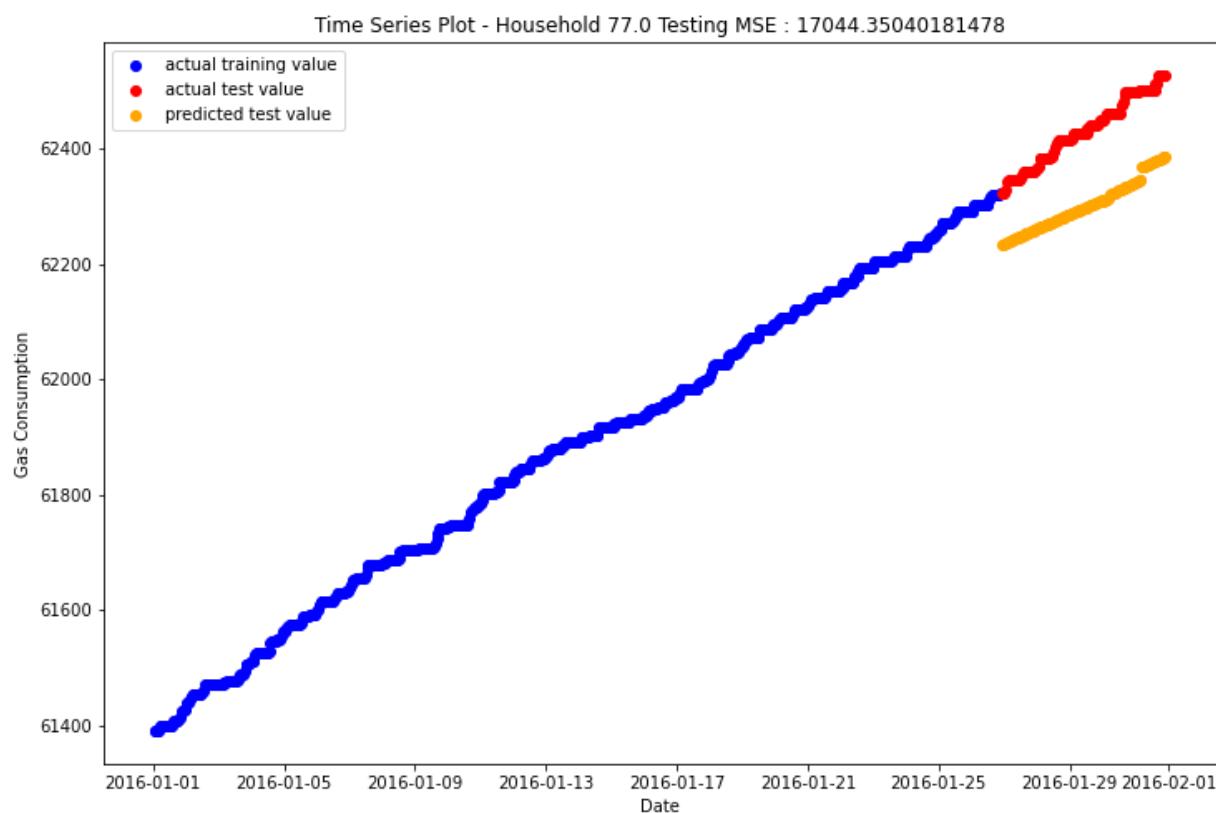
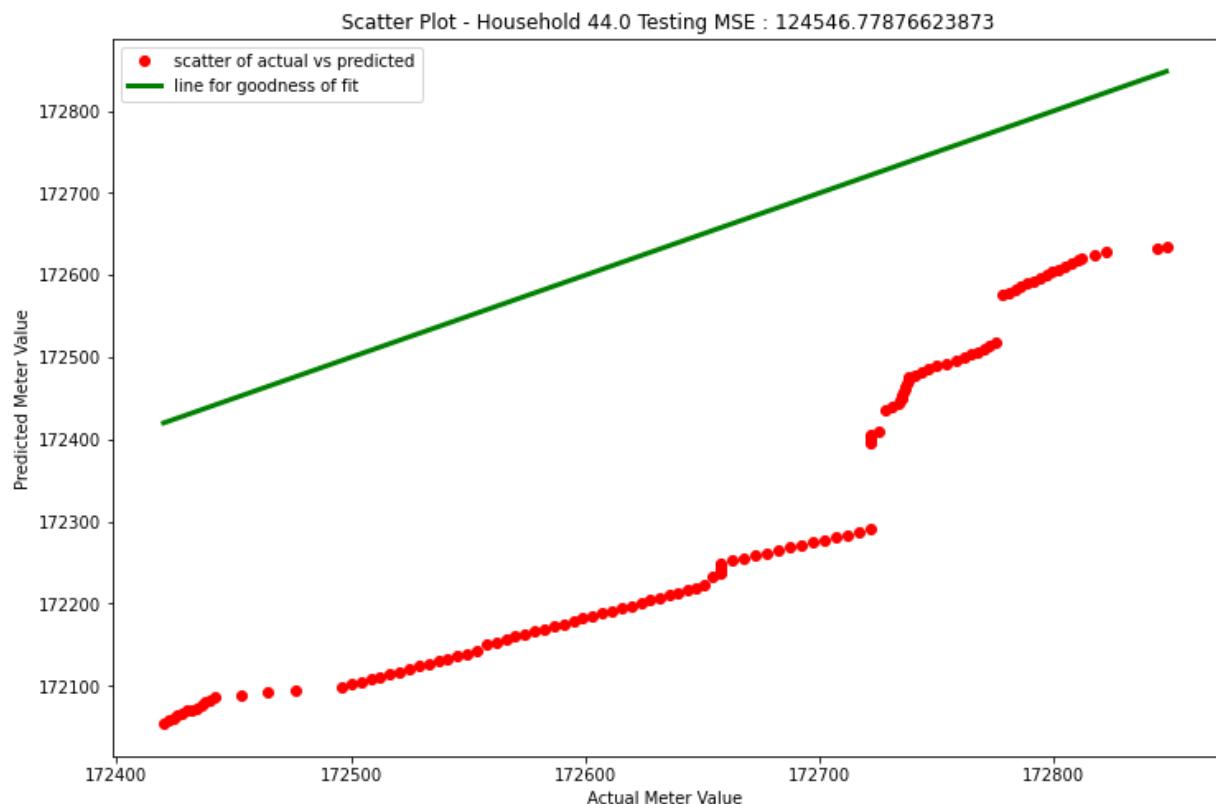


Scatter Plot - Household 35.0 Testing MSE : 98569.75208129882

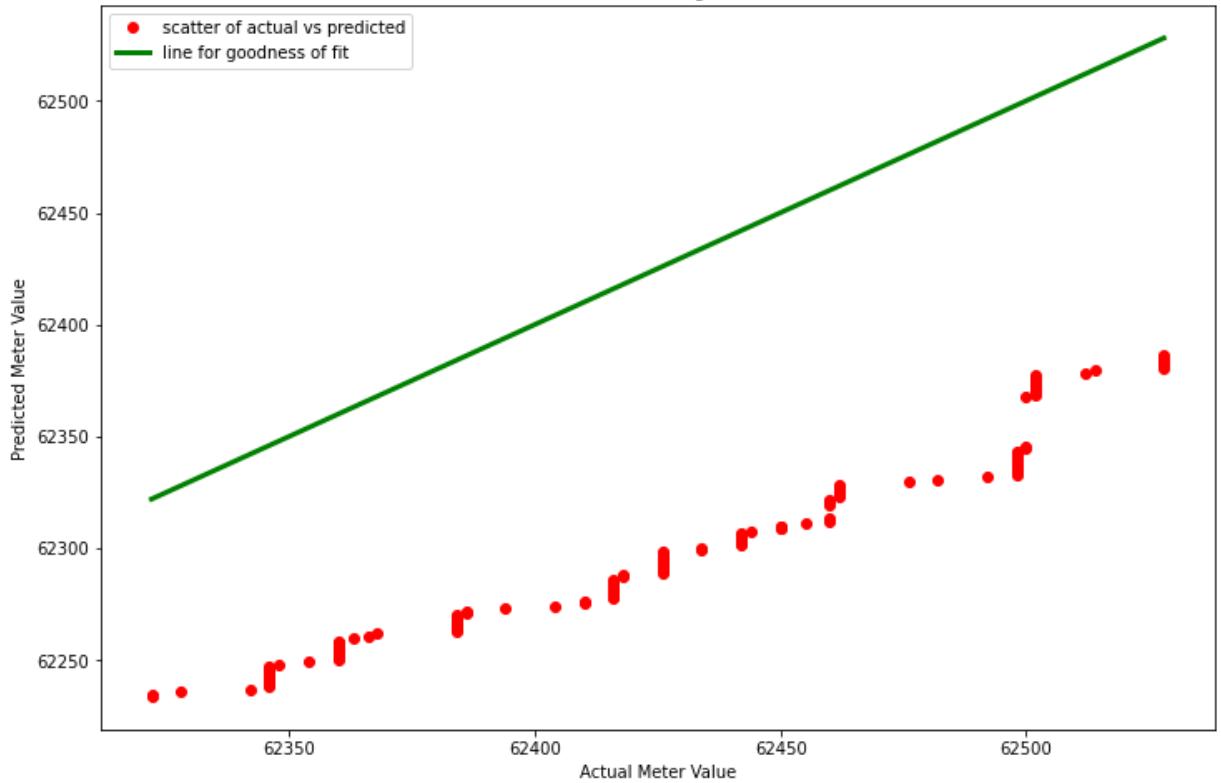


Time Series Plot - Household 44.0 Testing MSE : 124546.77876623873

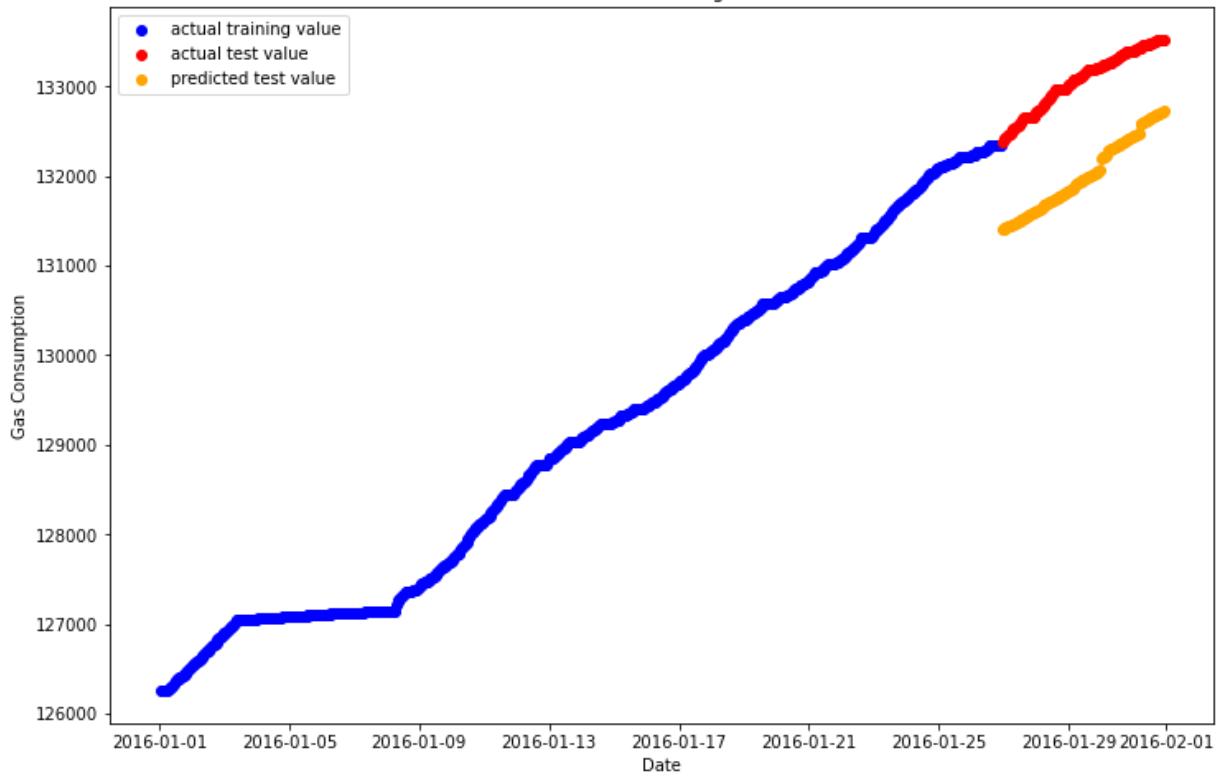


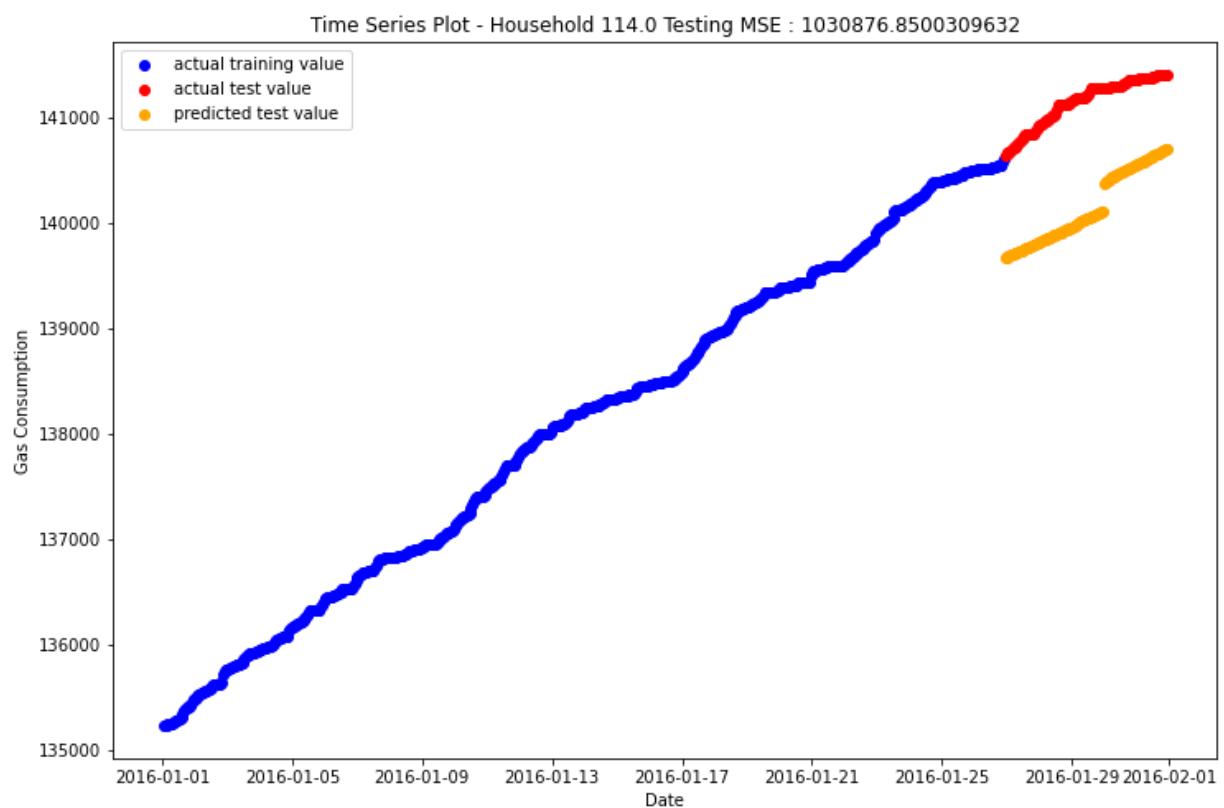
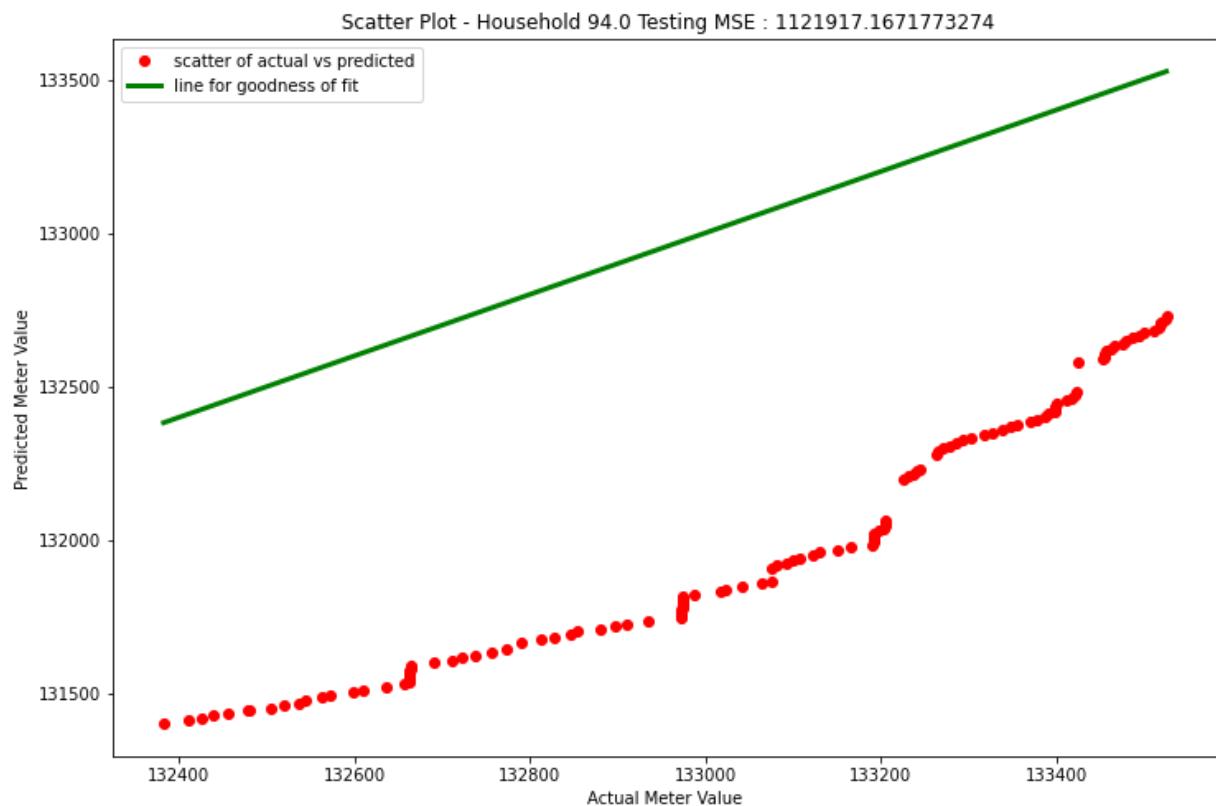


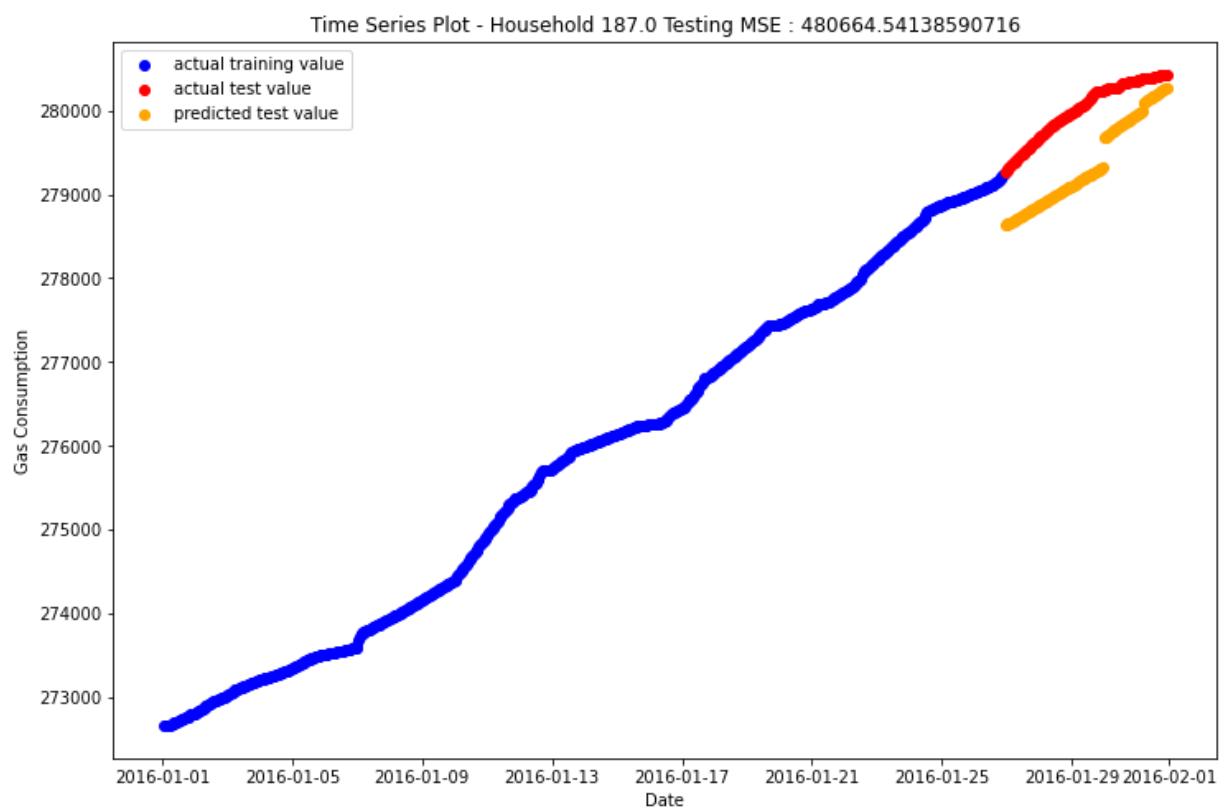
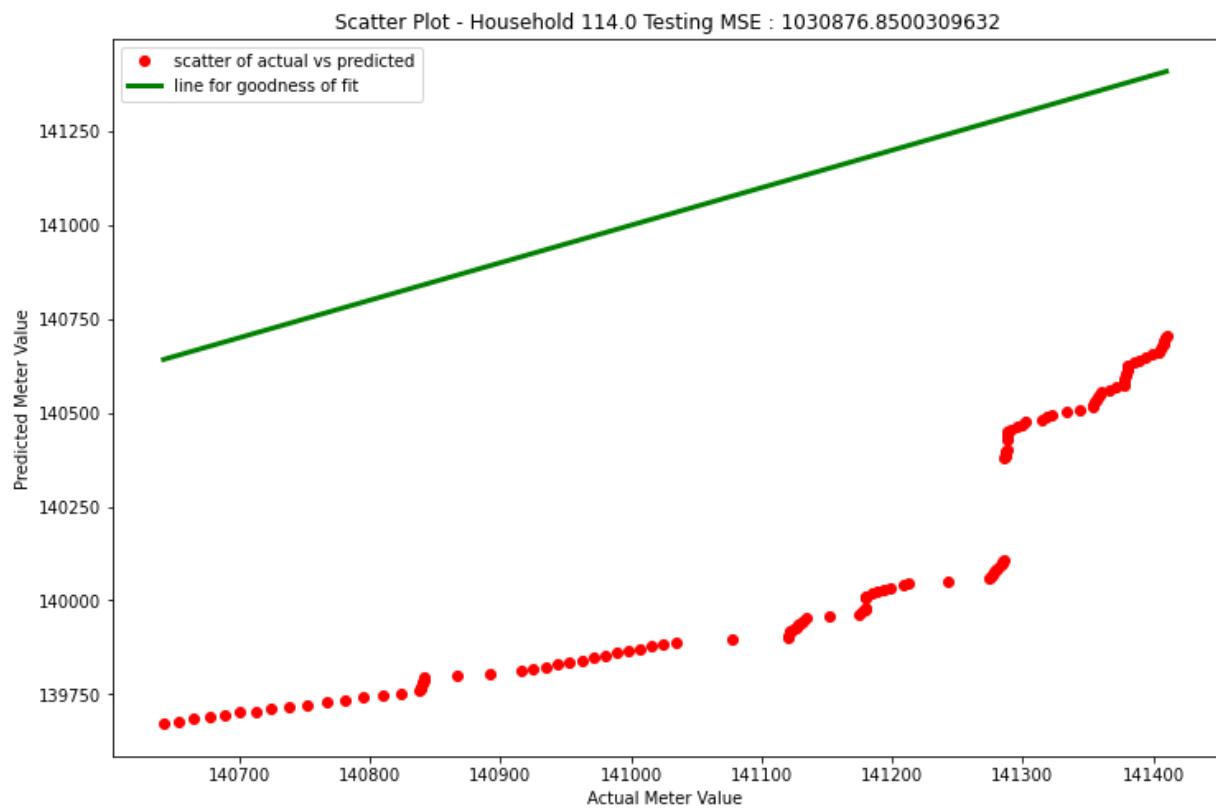
Scatter Plot - Household 77.0 Testing MSE : 17044.35040181478

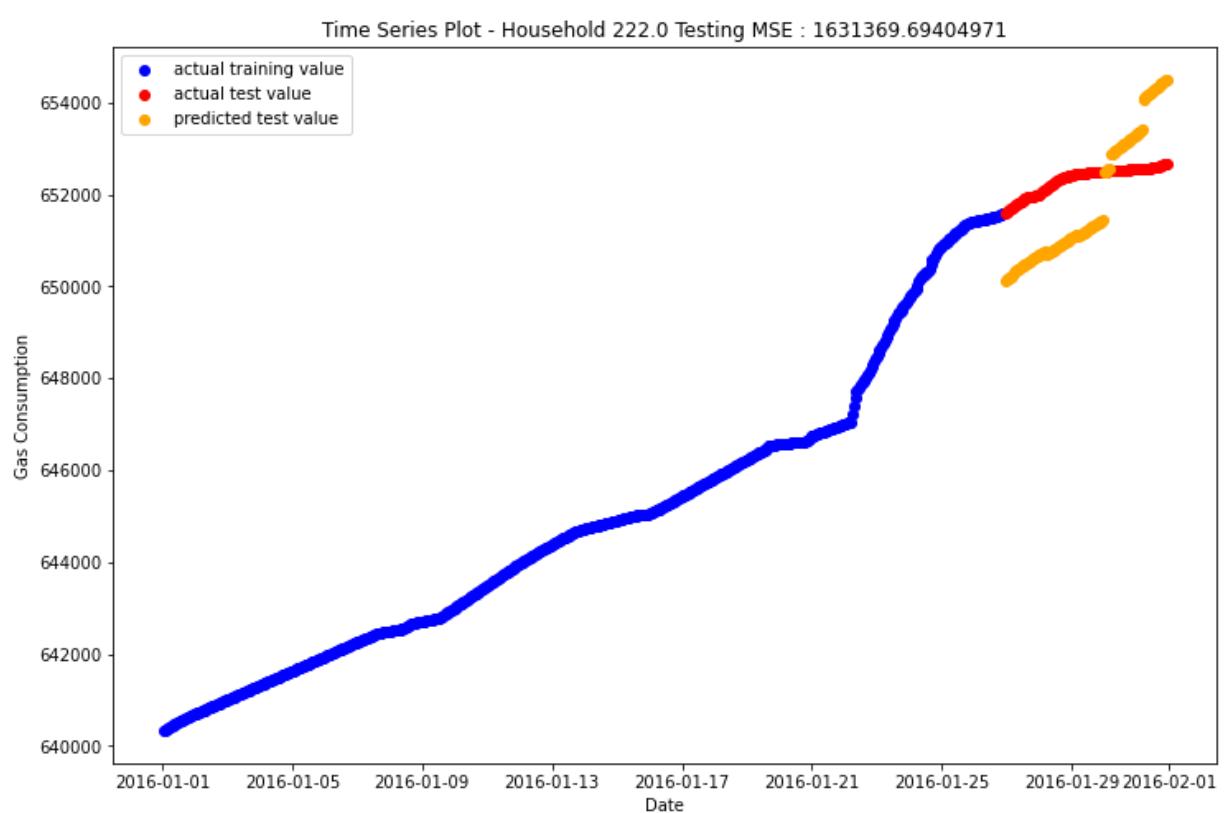
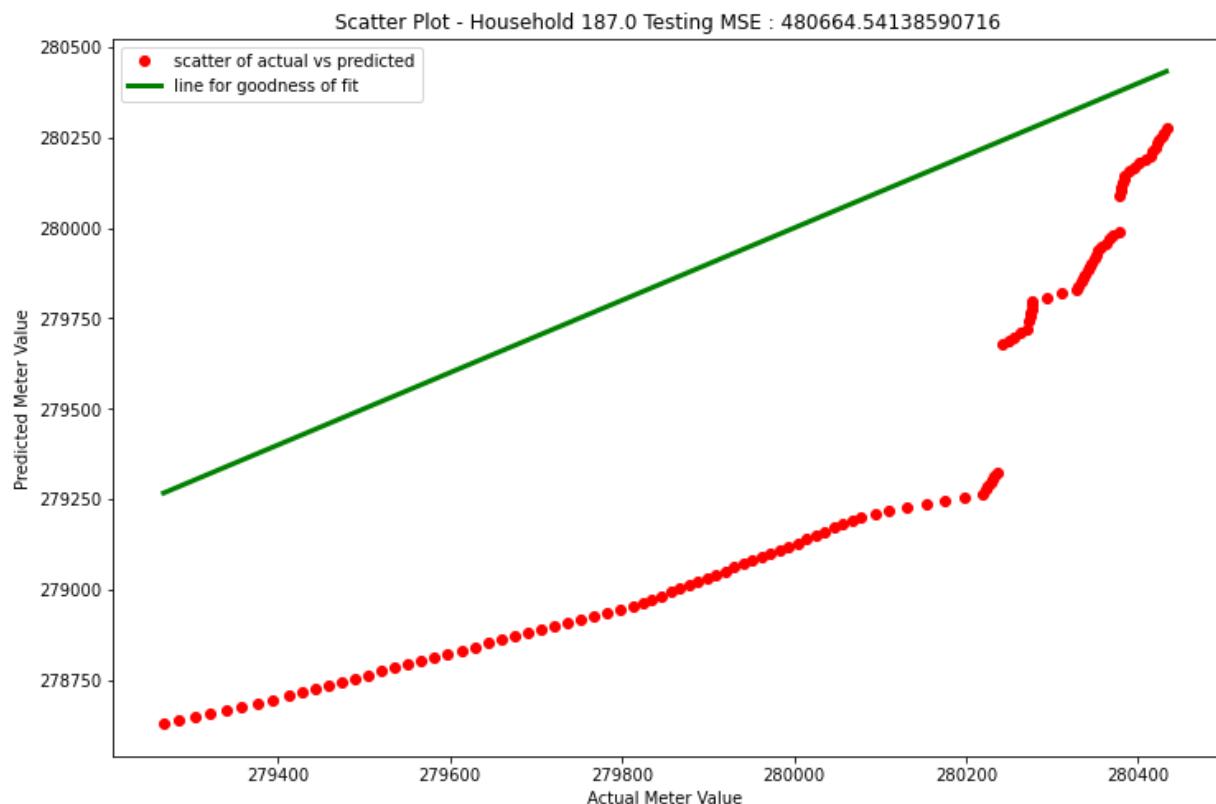


Time Series Plot - Household 94.0 Testing MSE : 1121917.1671773274

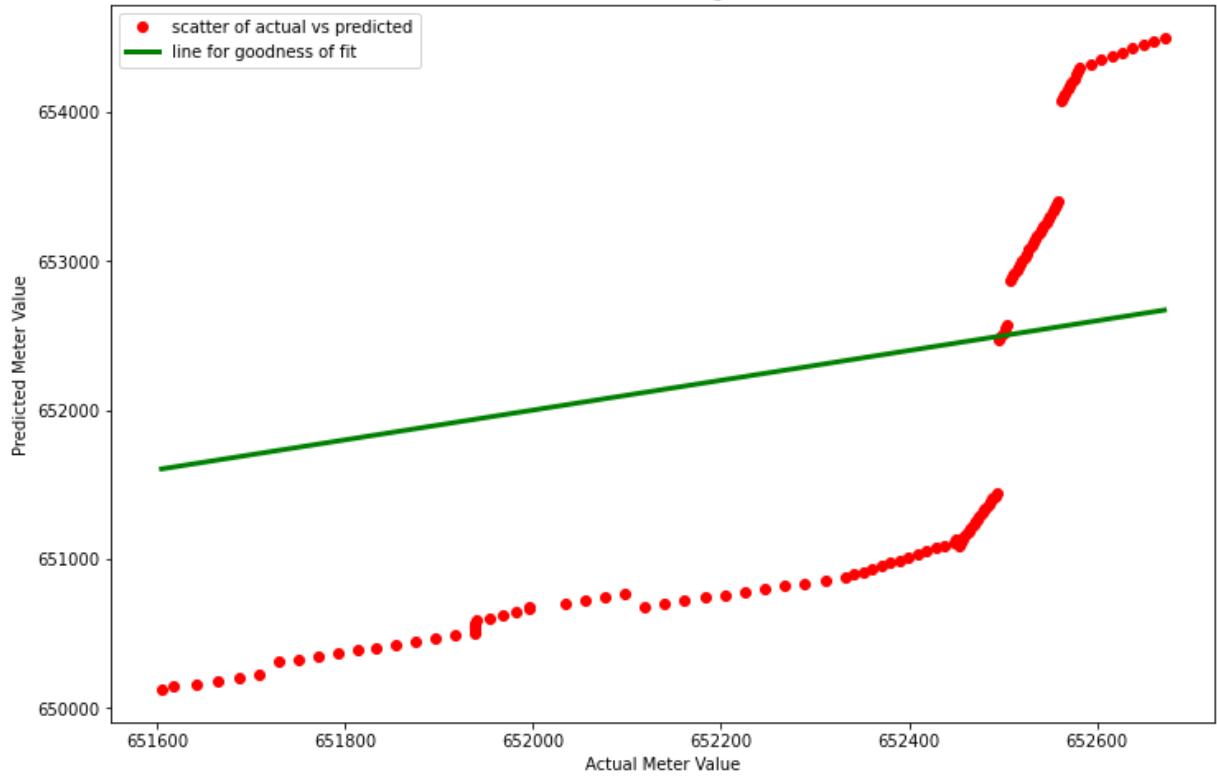




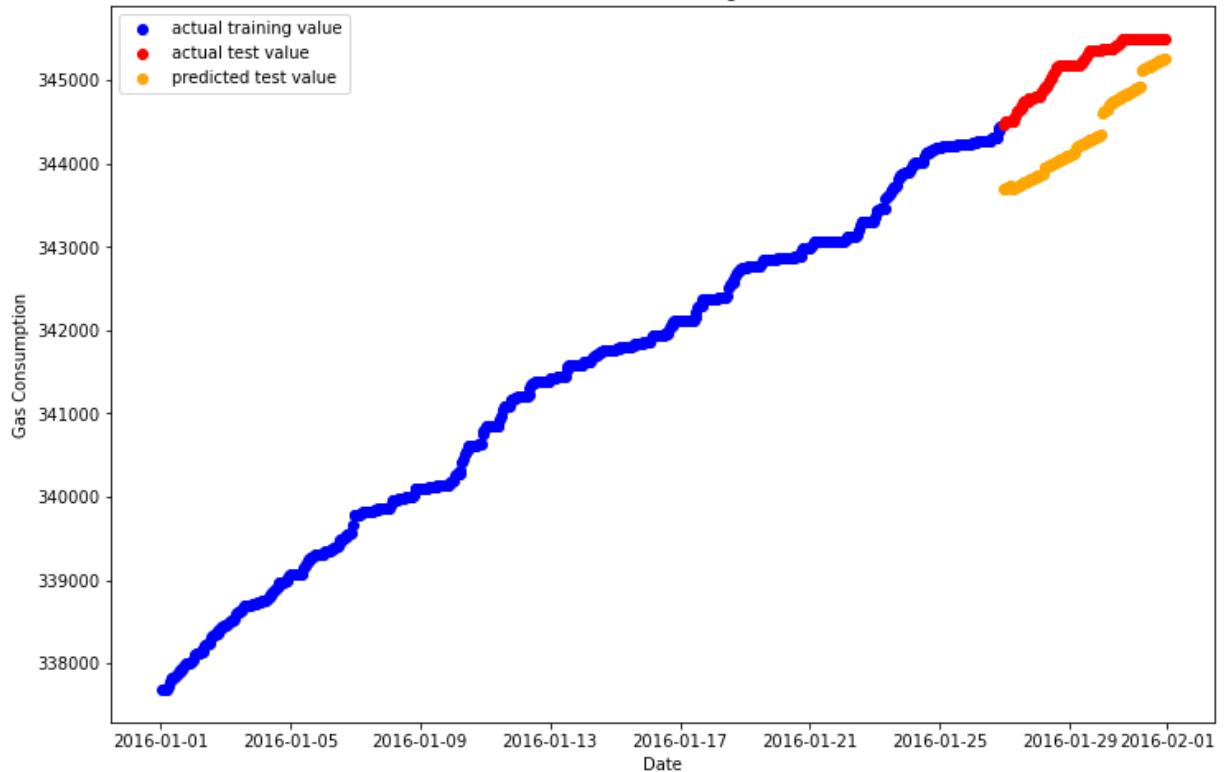


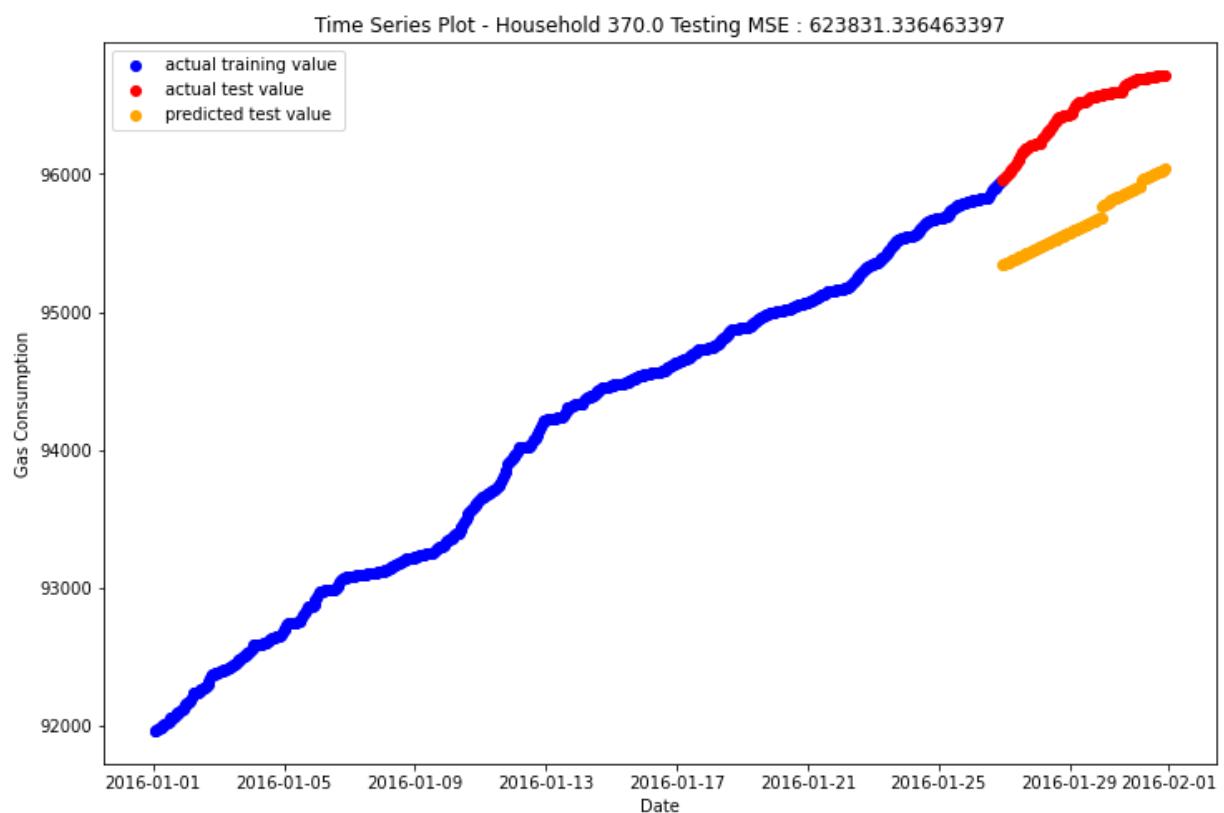
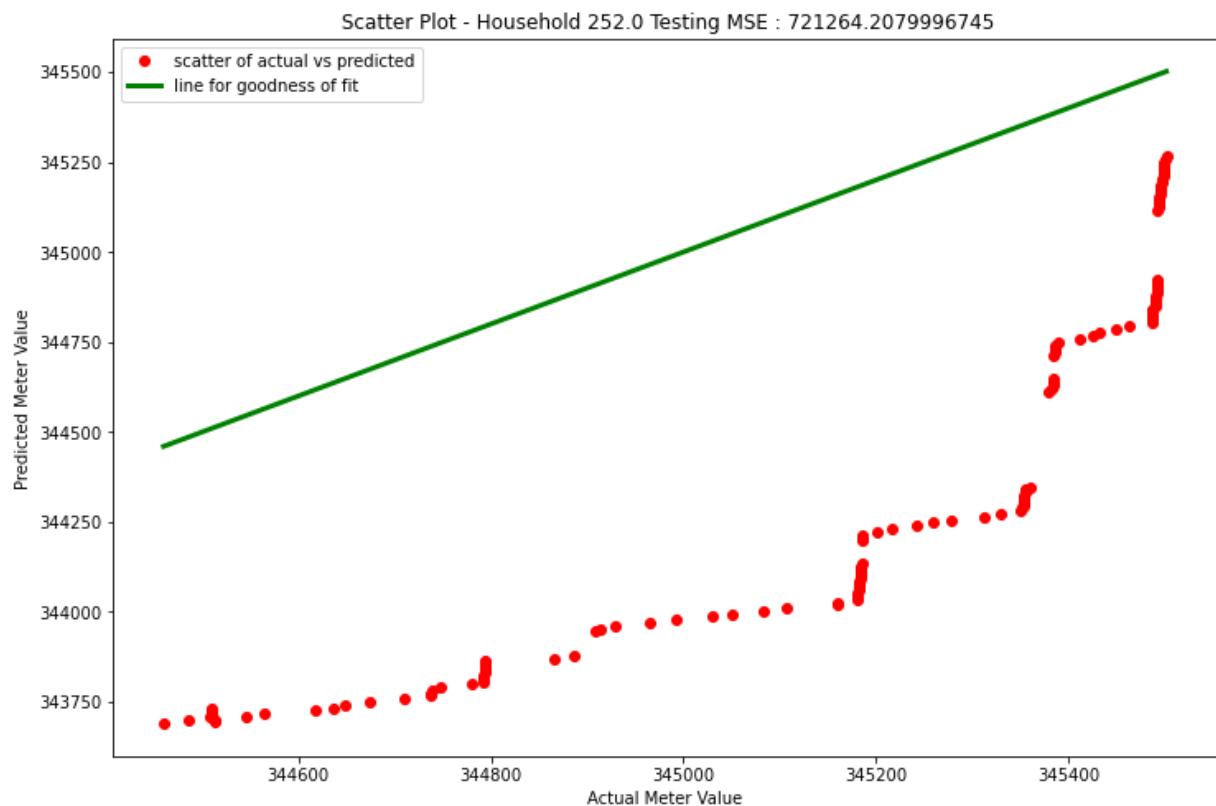


Scatter Plot - Household 222.0 Testing MSE : 1631369.69404971

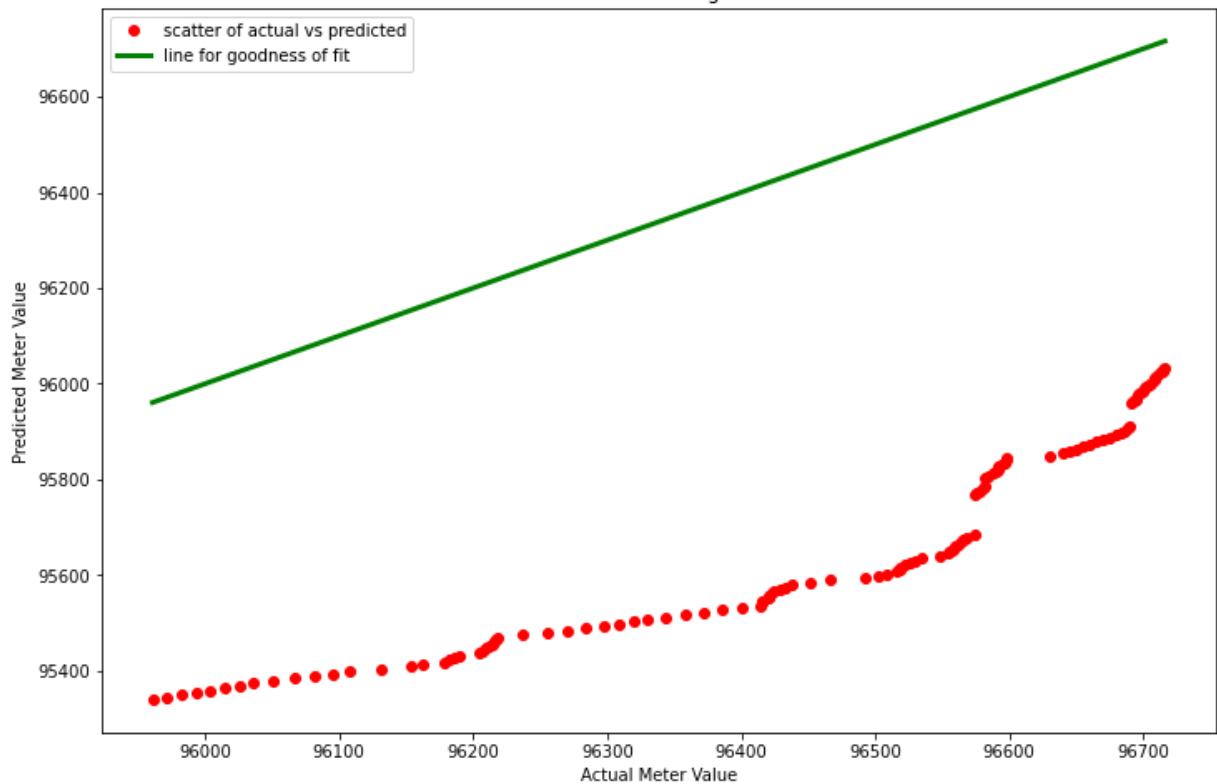


Time Series Plot - Household 252.0 Testing MSE : 721264.2079996745

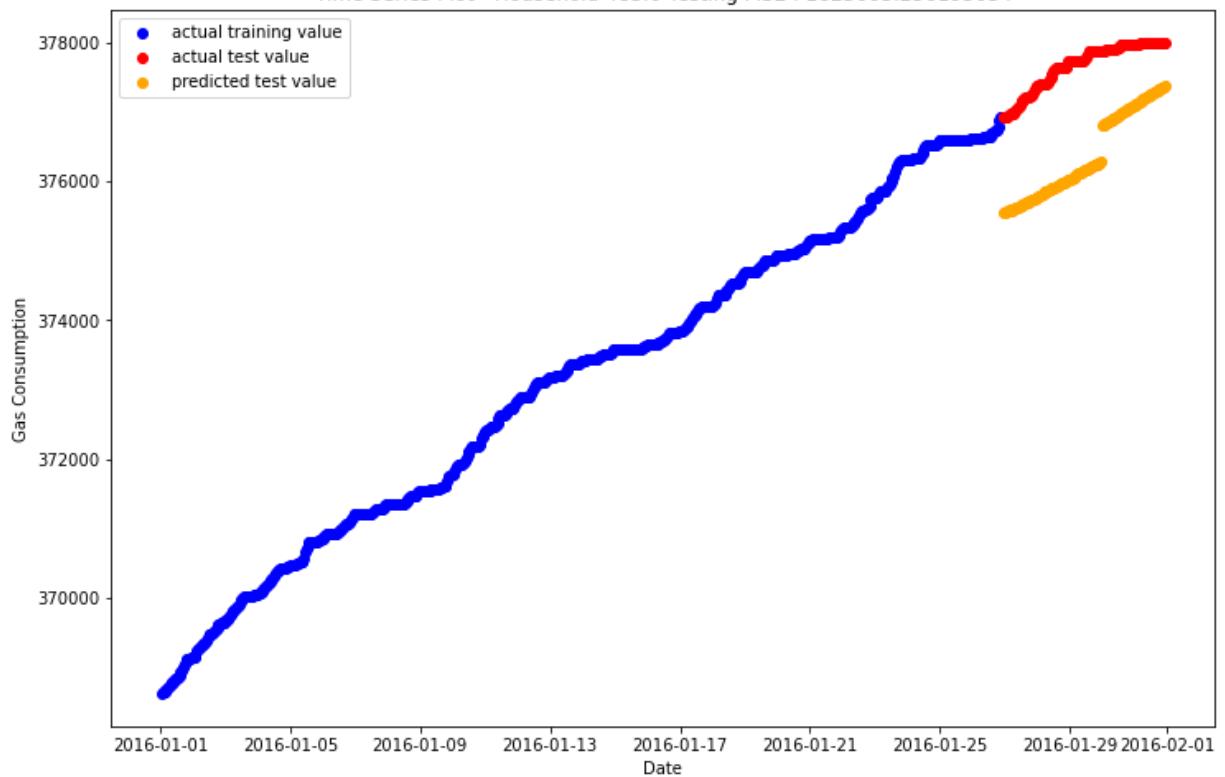


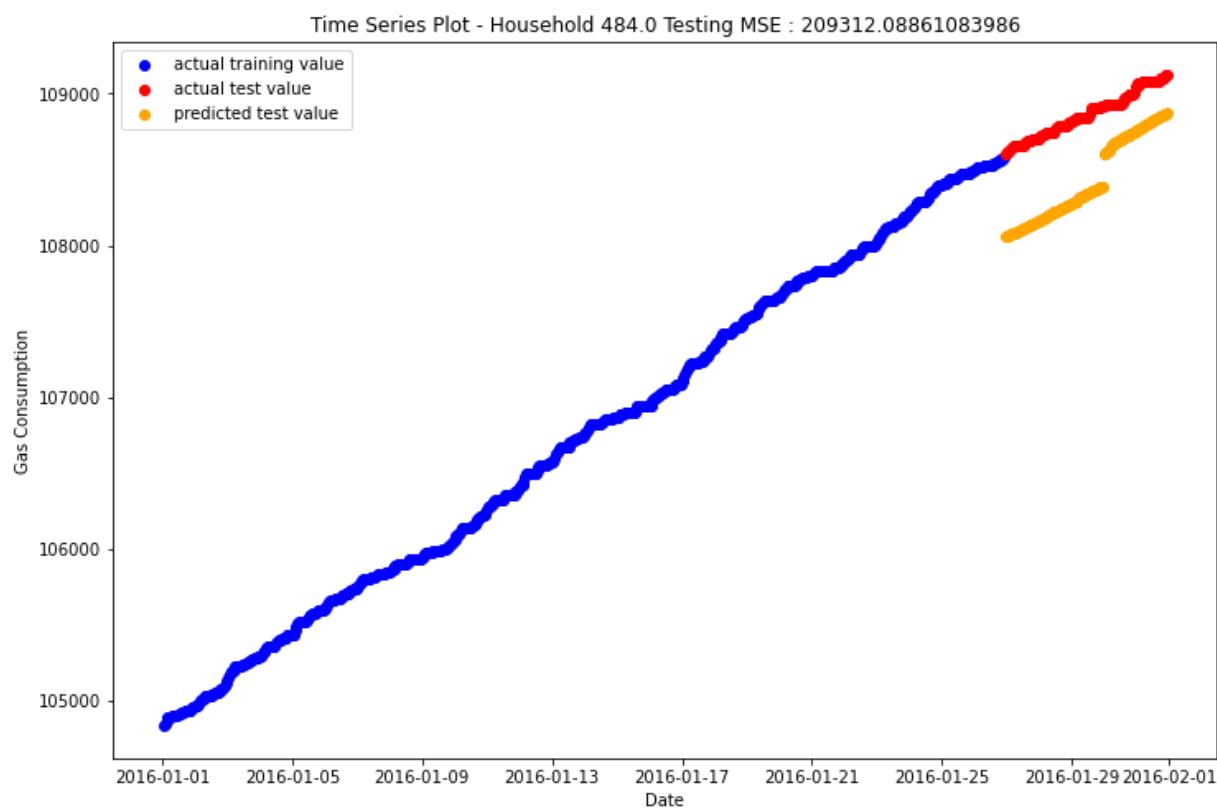
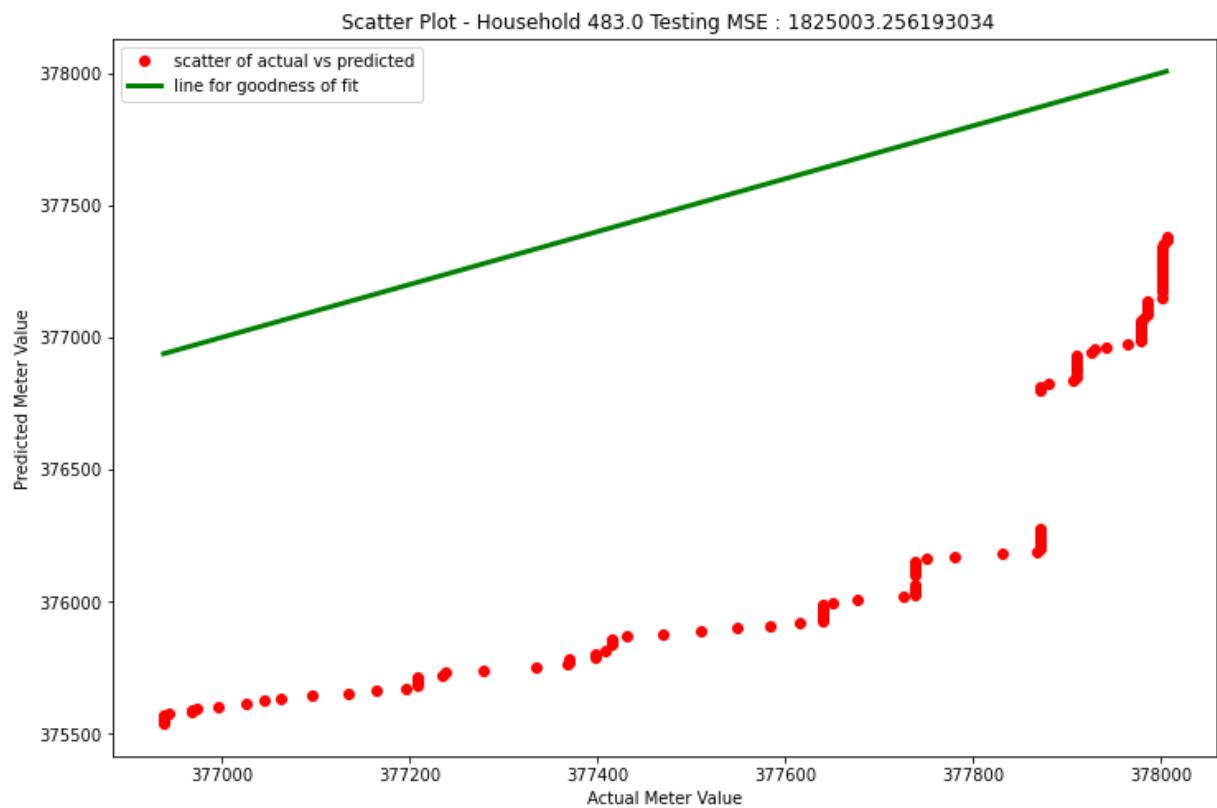


Scatter Plot - Household 370.0 Testing MSE : 623831.336463397

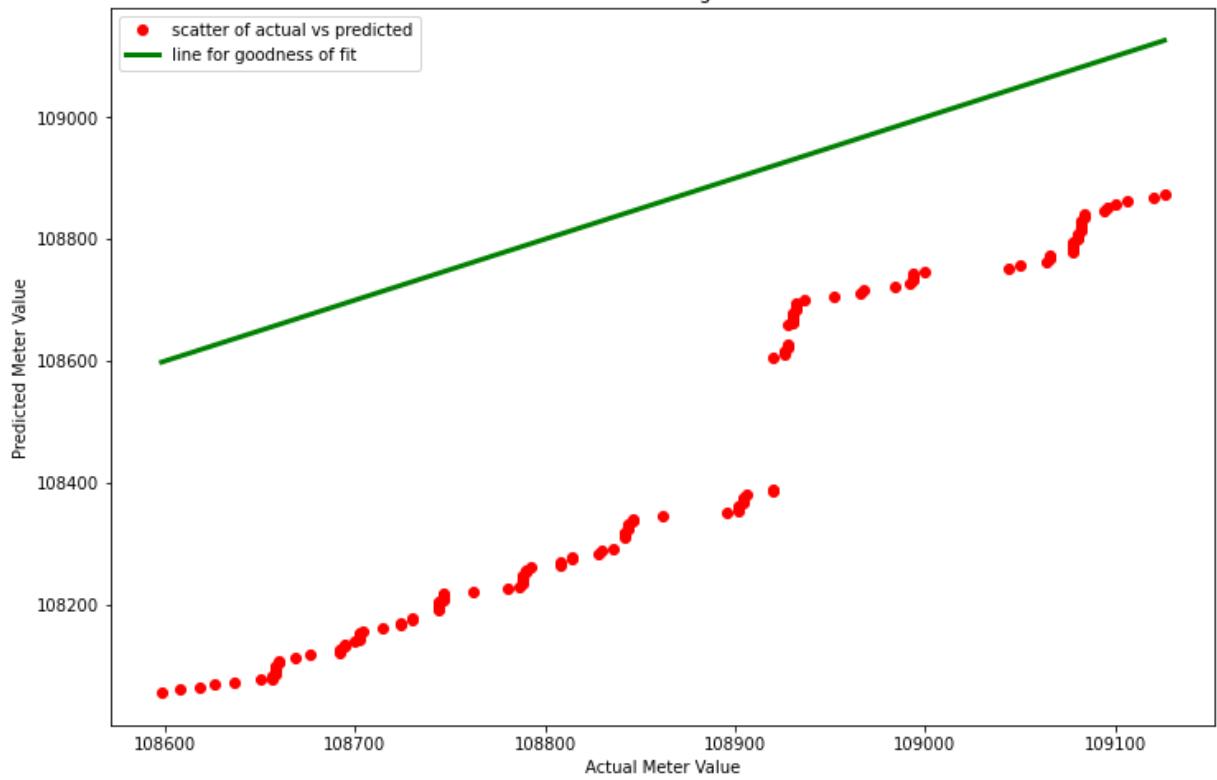


Time Series Plot - Household 483.0 Testing MSE : 1825003.256193034

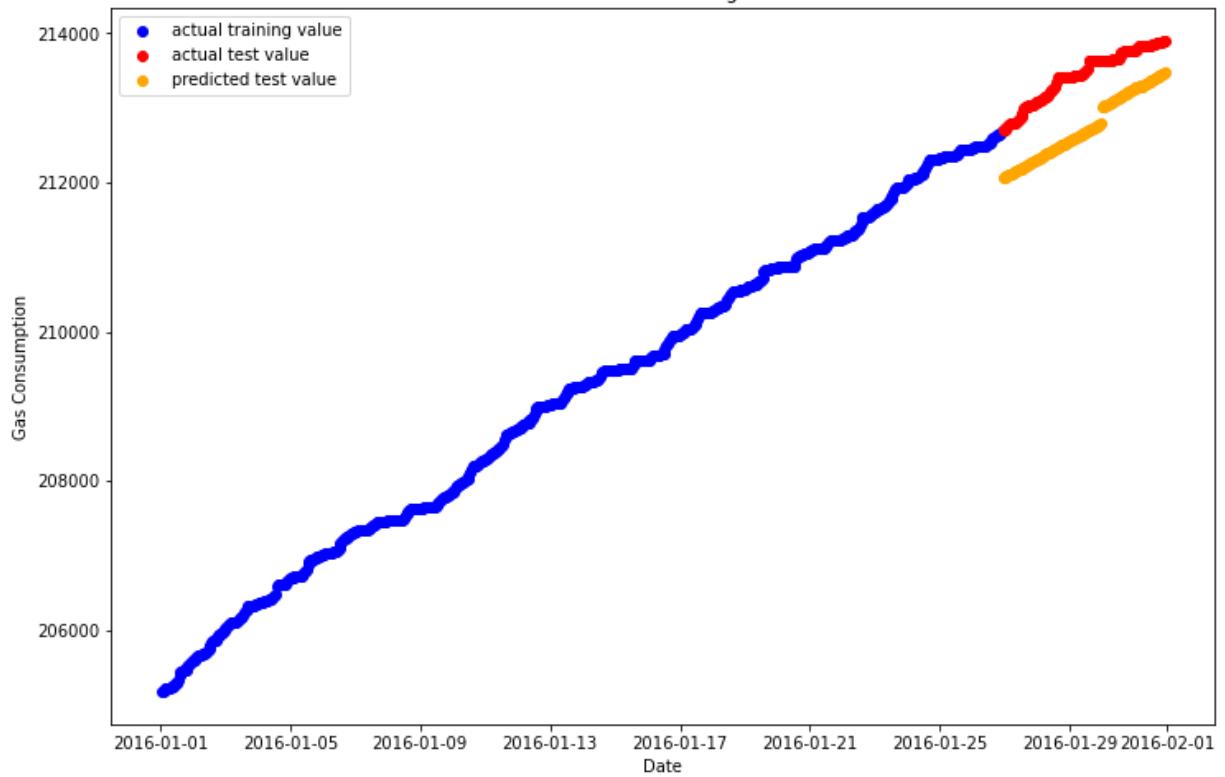




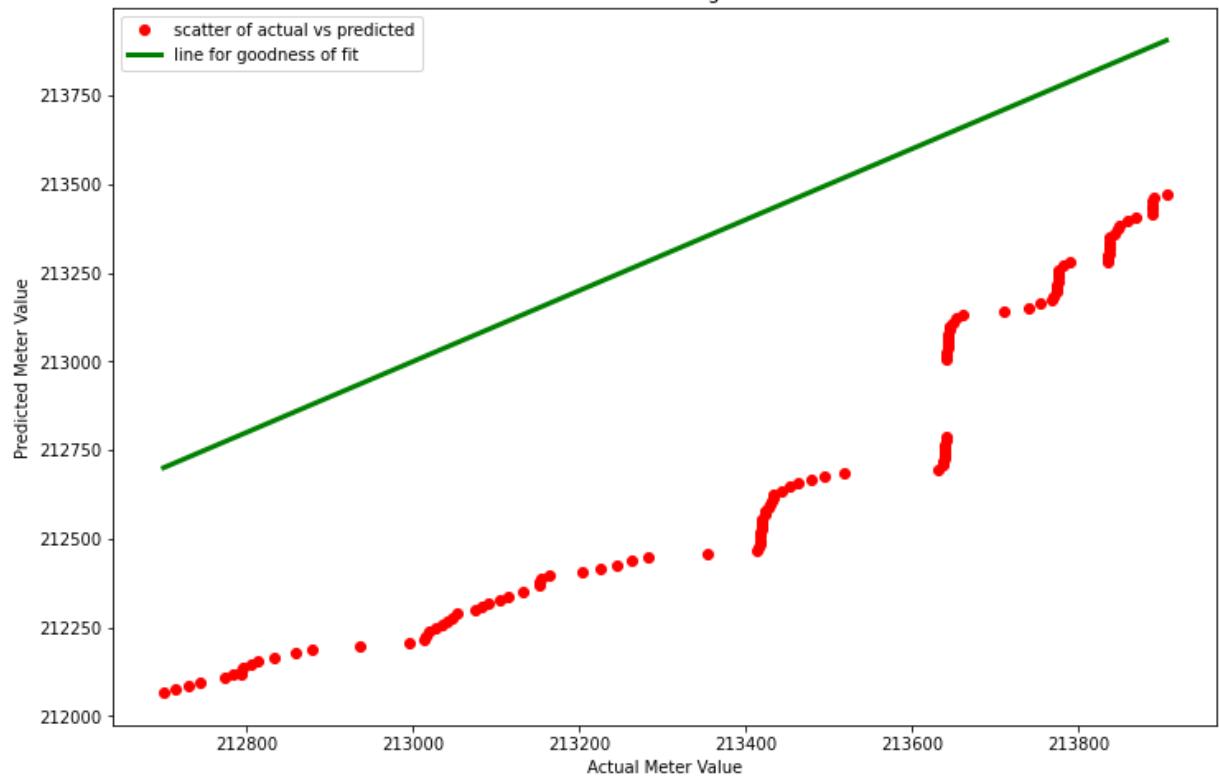
Scatter Plot - Household 484.0 Testing MSE : 209312.08861083986



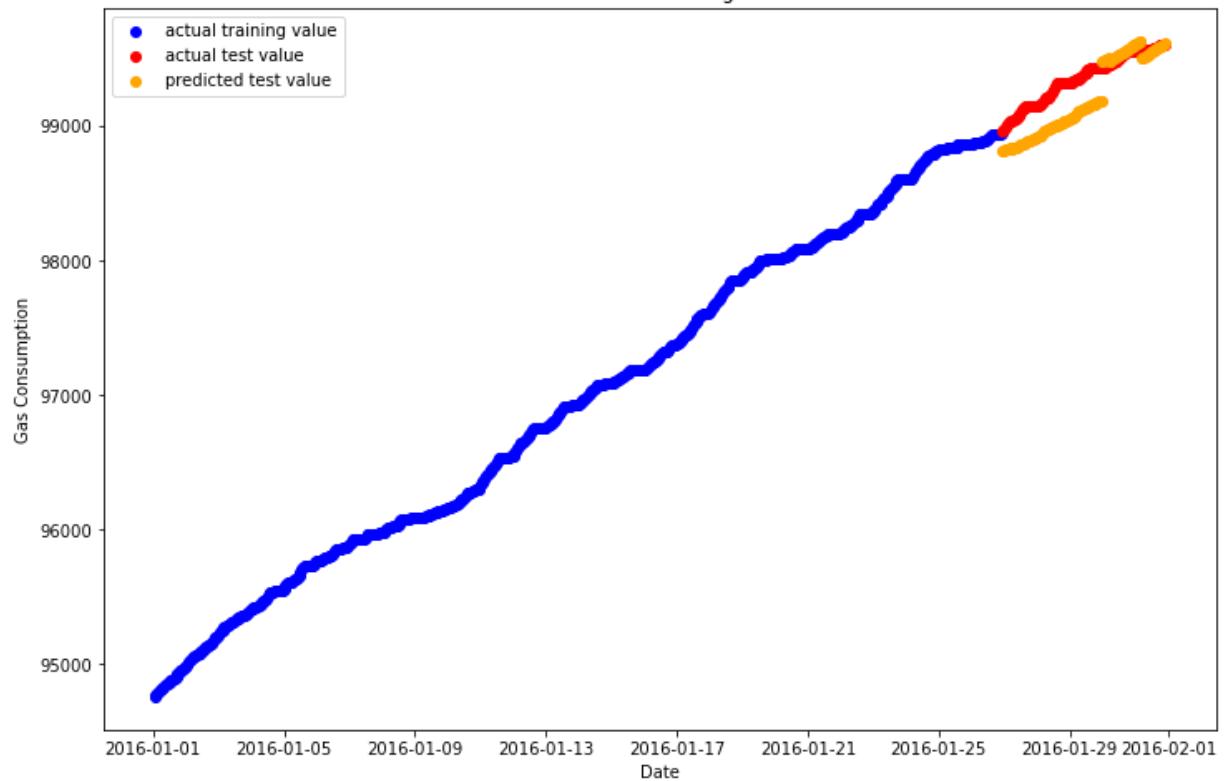
Time Series Plot - Household 661.0 Testing MSE : 507377.3477376302

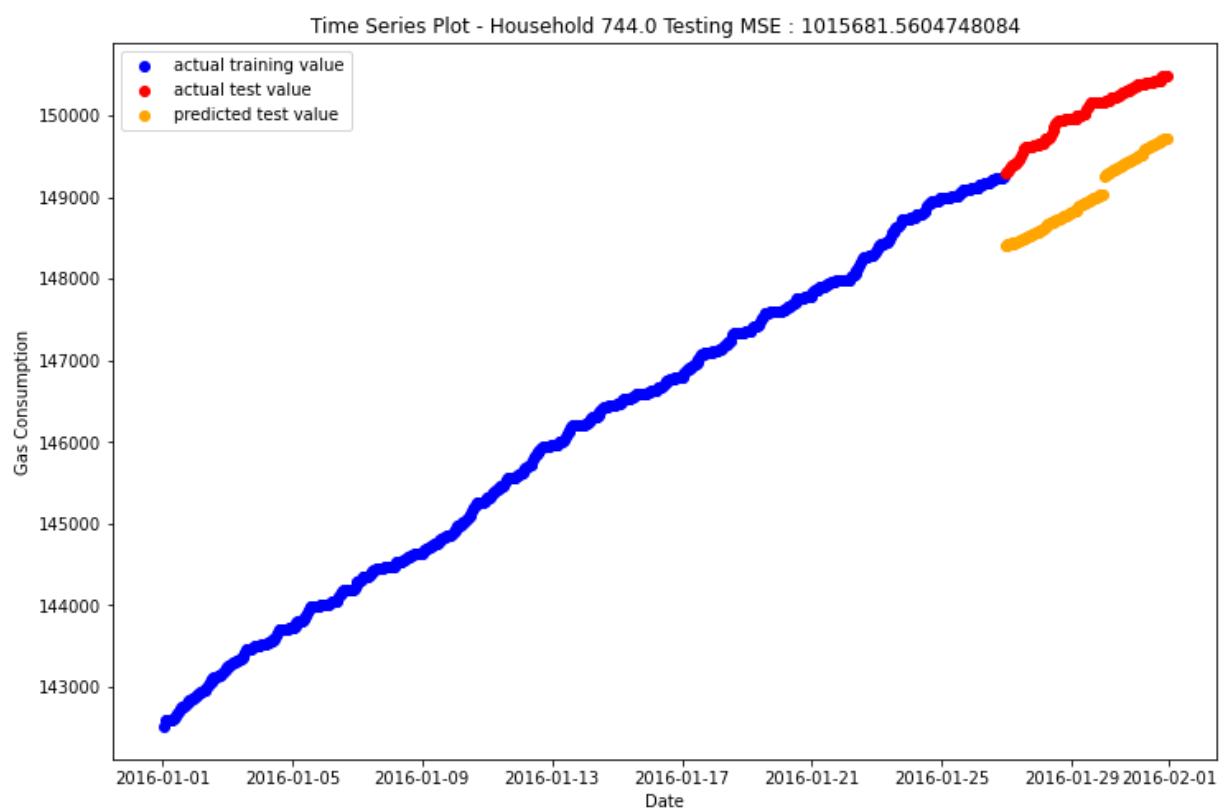
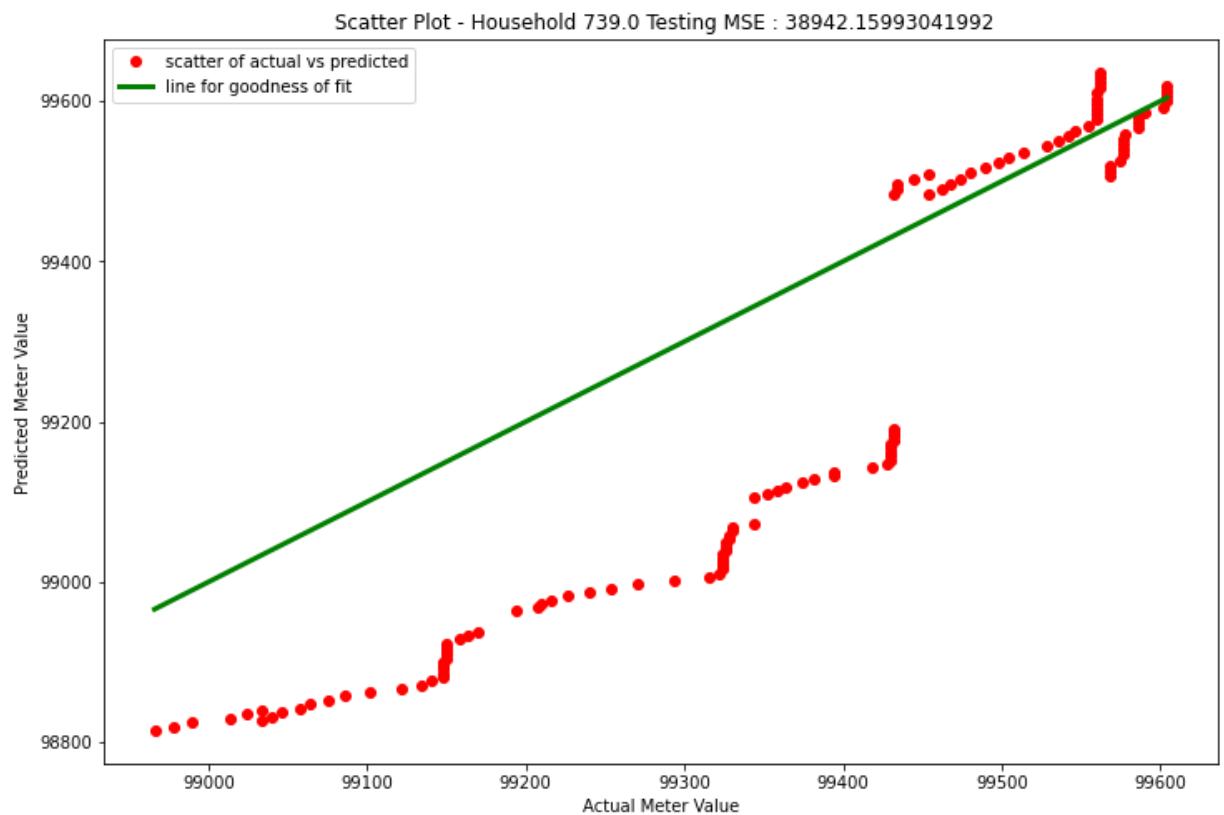


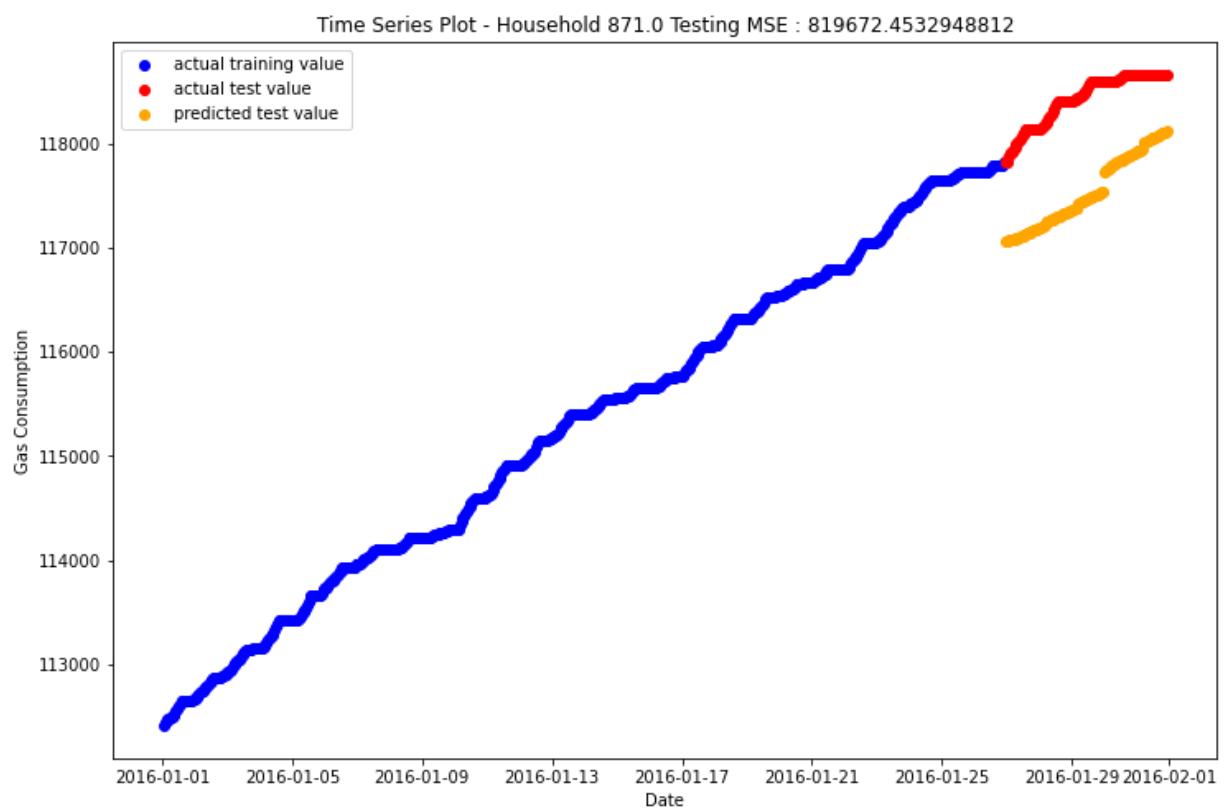
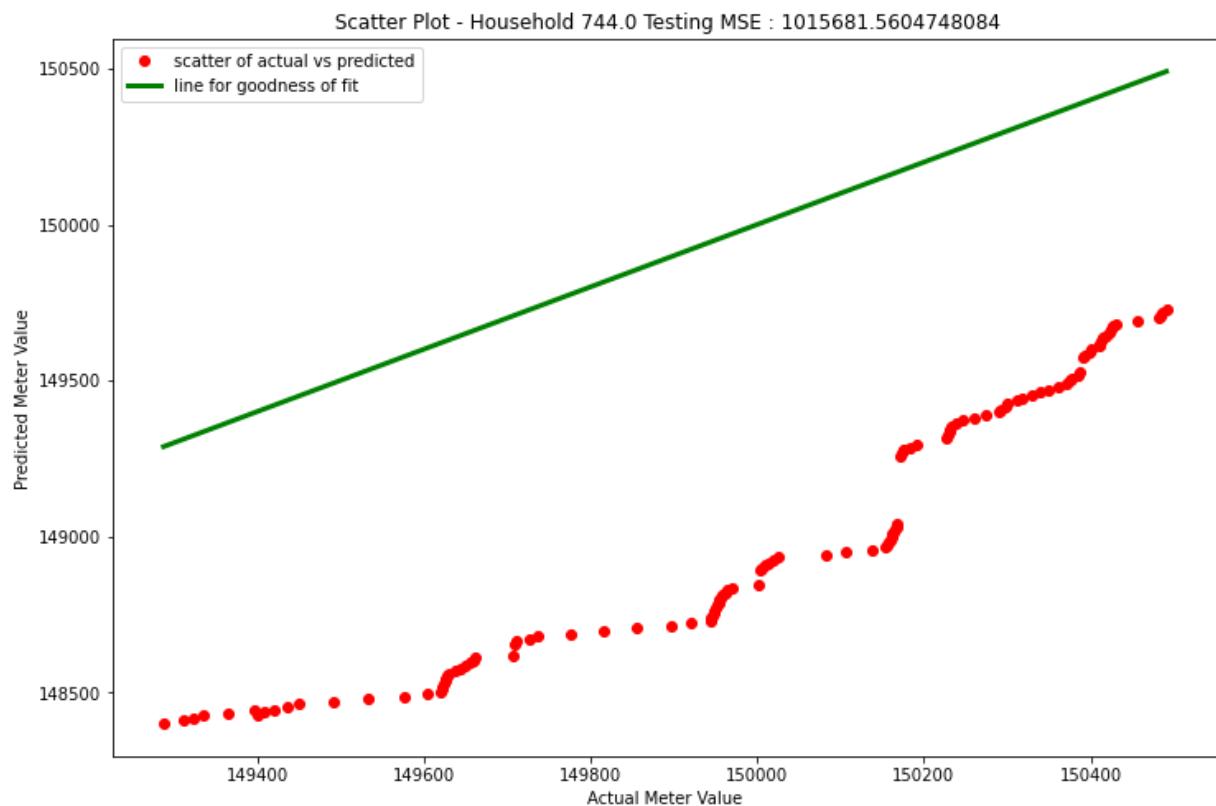
Scatter Plot - Household 661.0 Testing MSE : 507377.3477376302

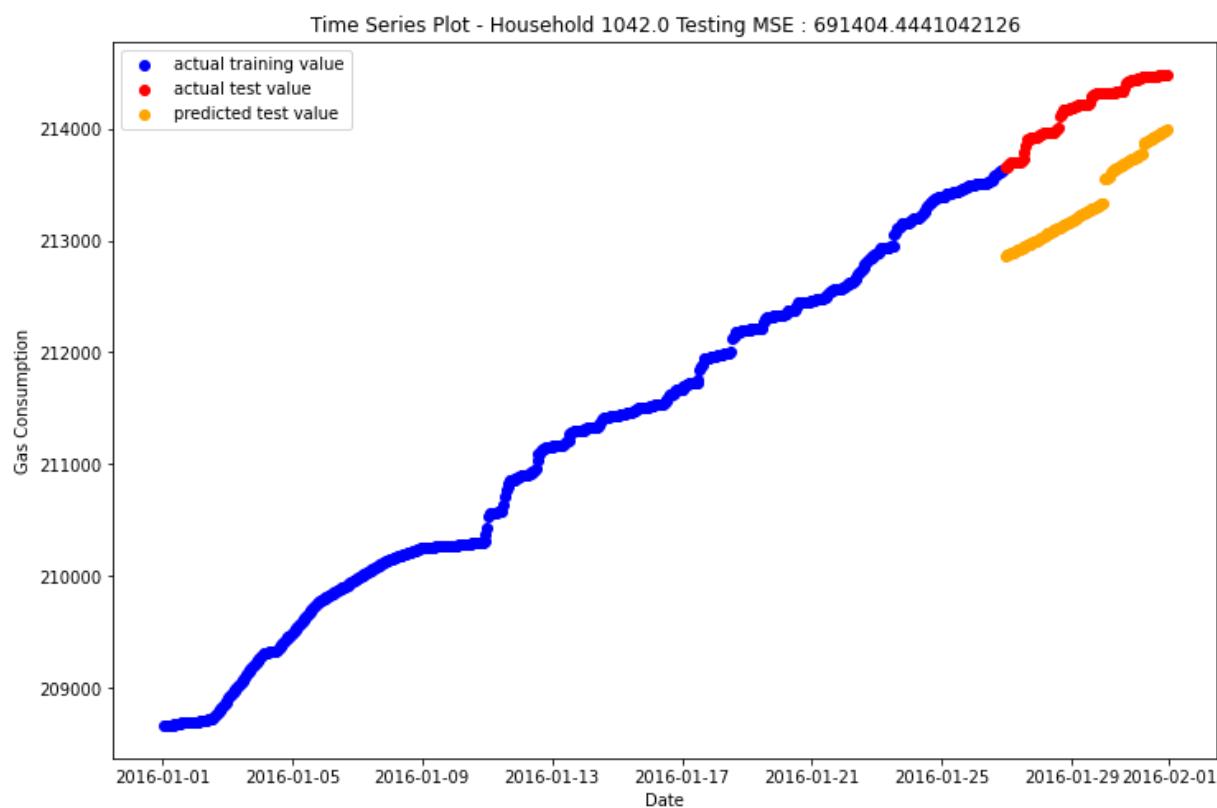
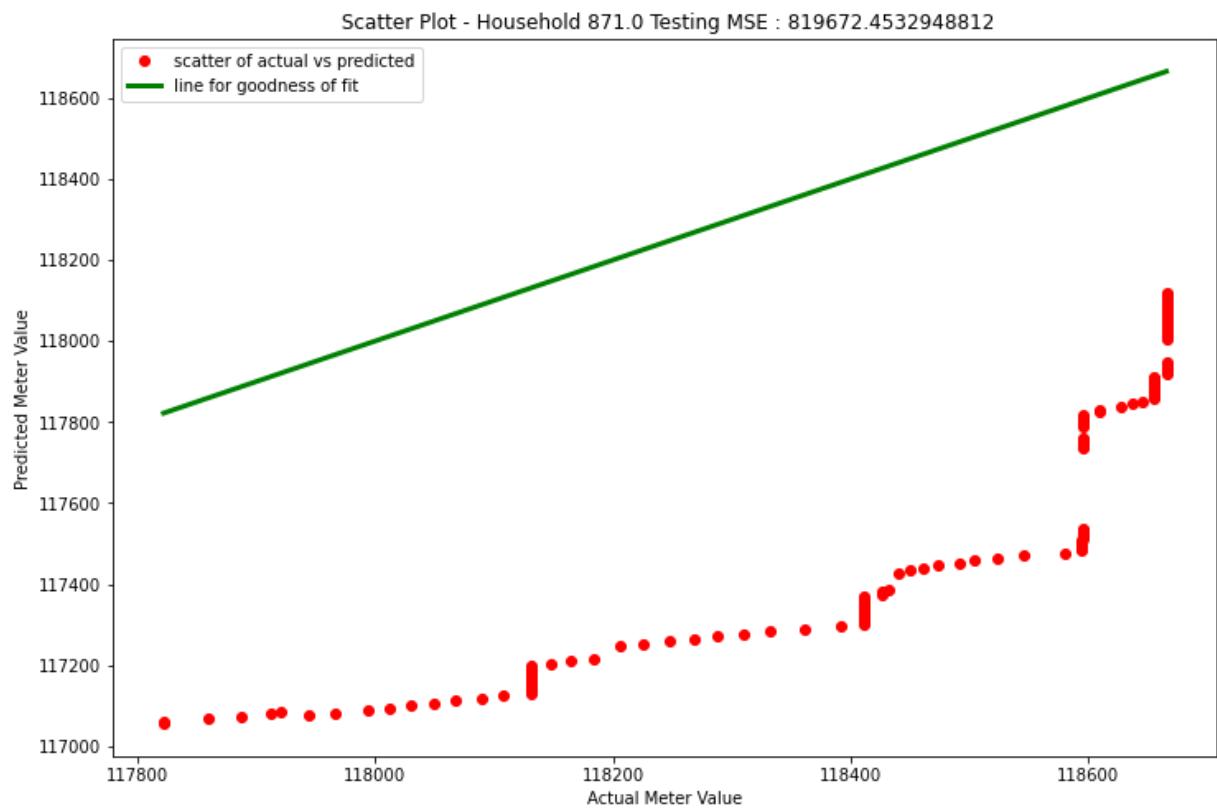


Time Series Plot - Household 739.0 Testing MSE : 38942.15993041992

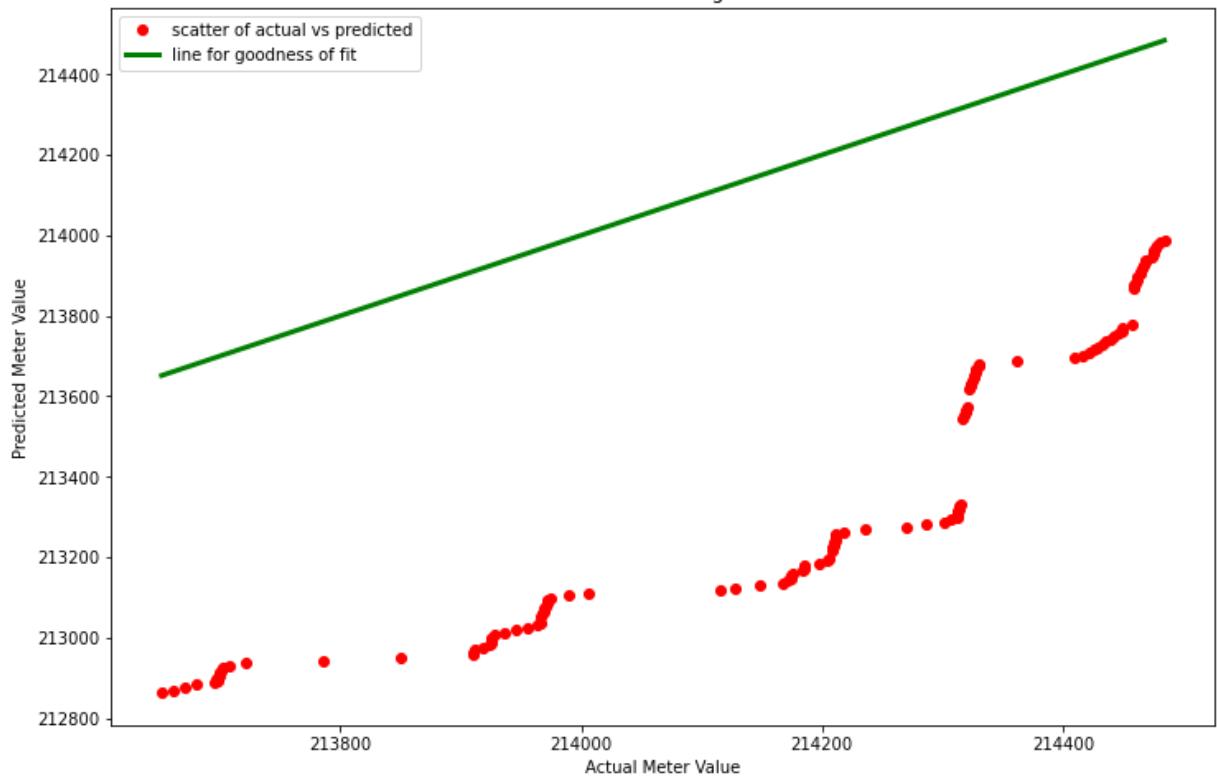




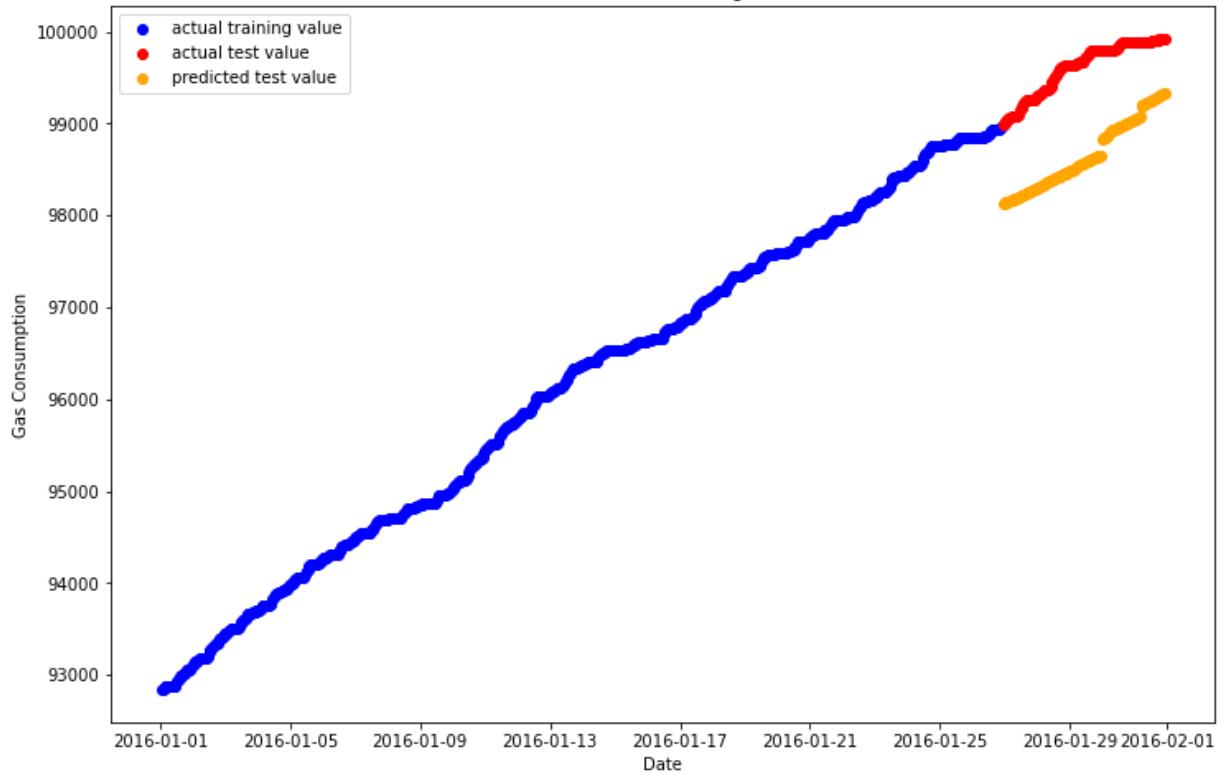


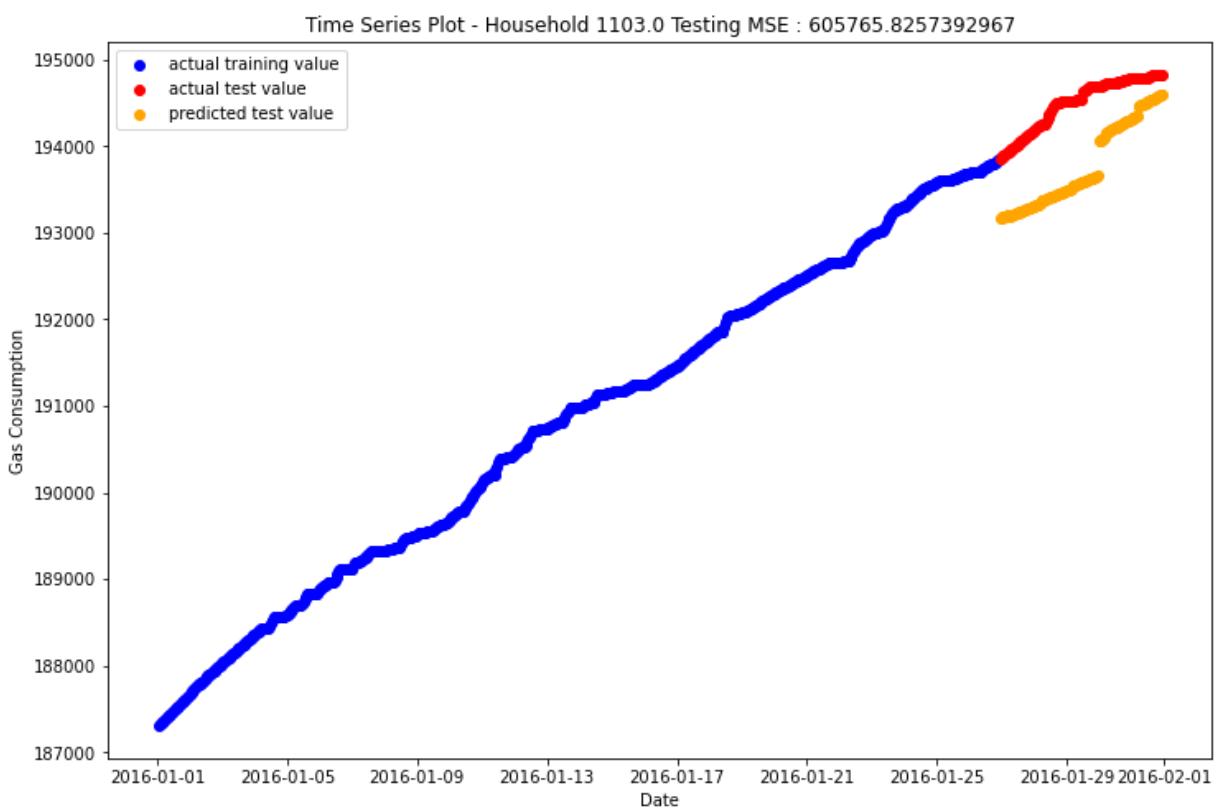
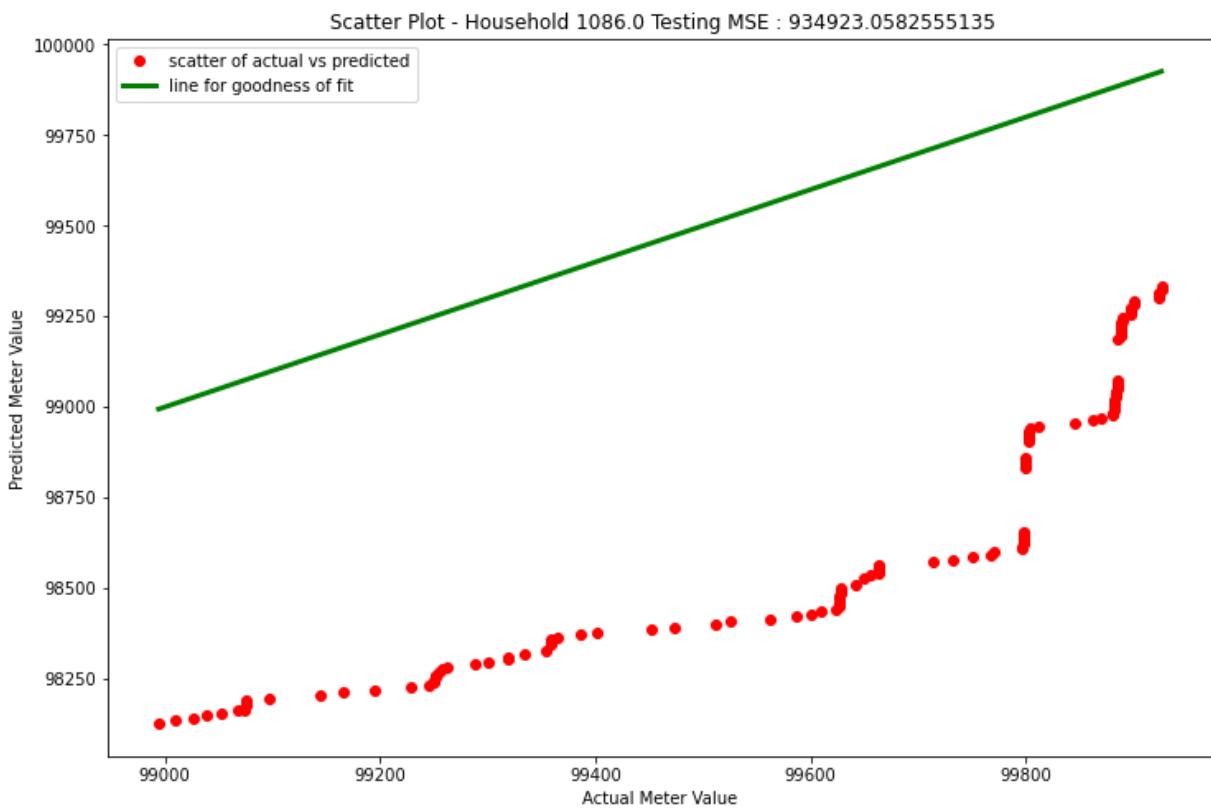


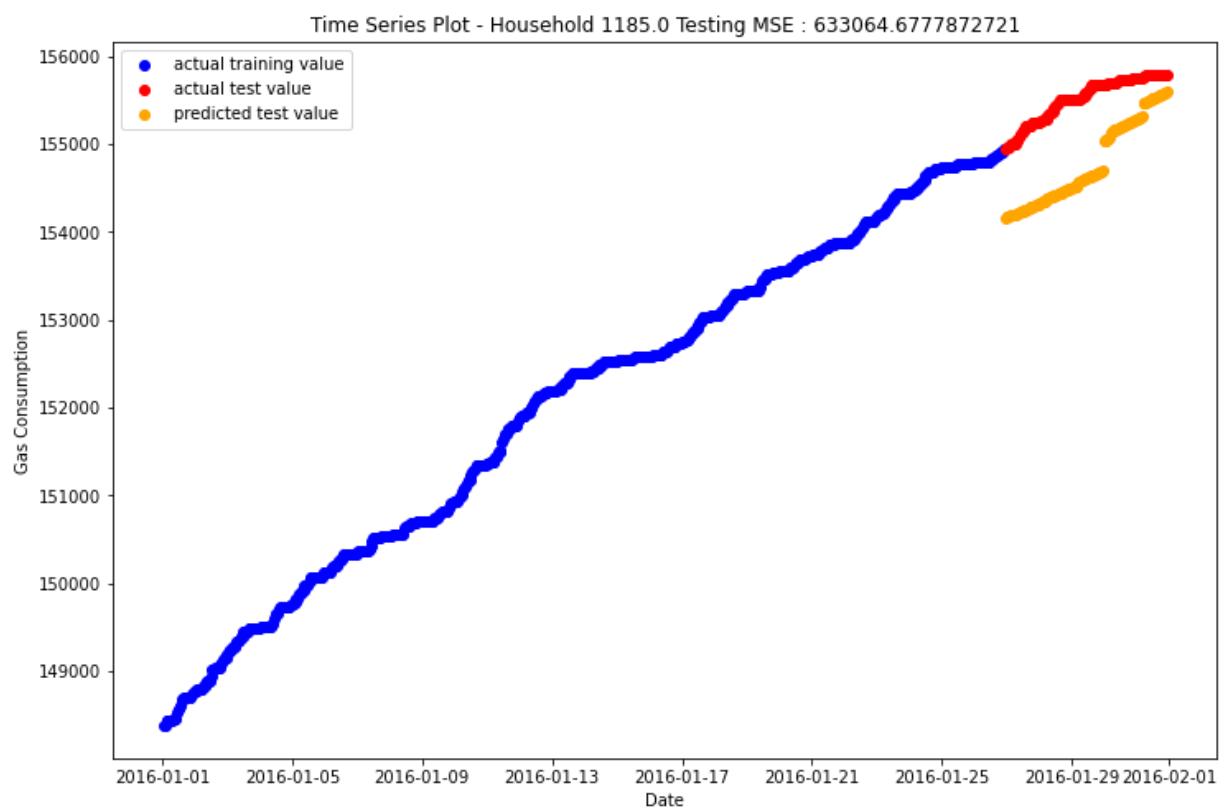
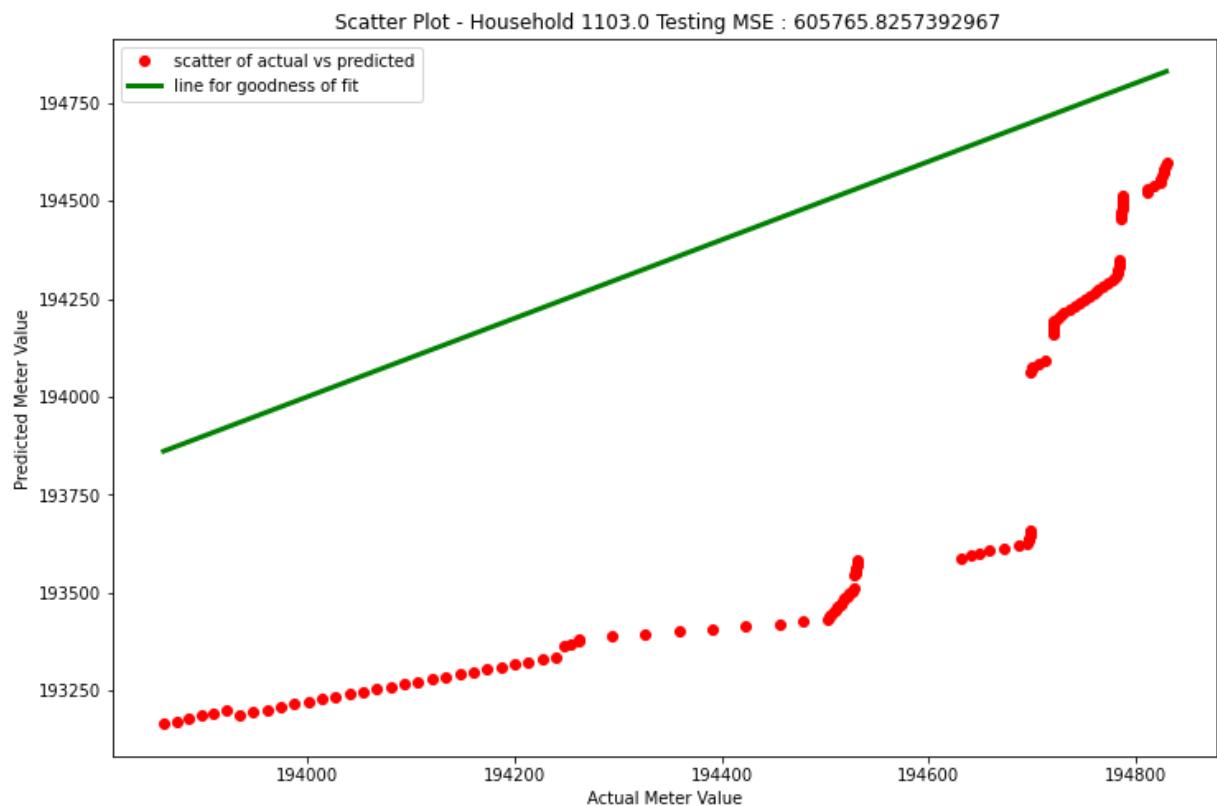
Scatter Plot - Household 1042.0 Testing MSE : 691404.4441042126

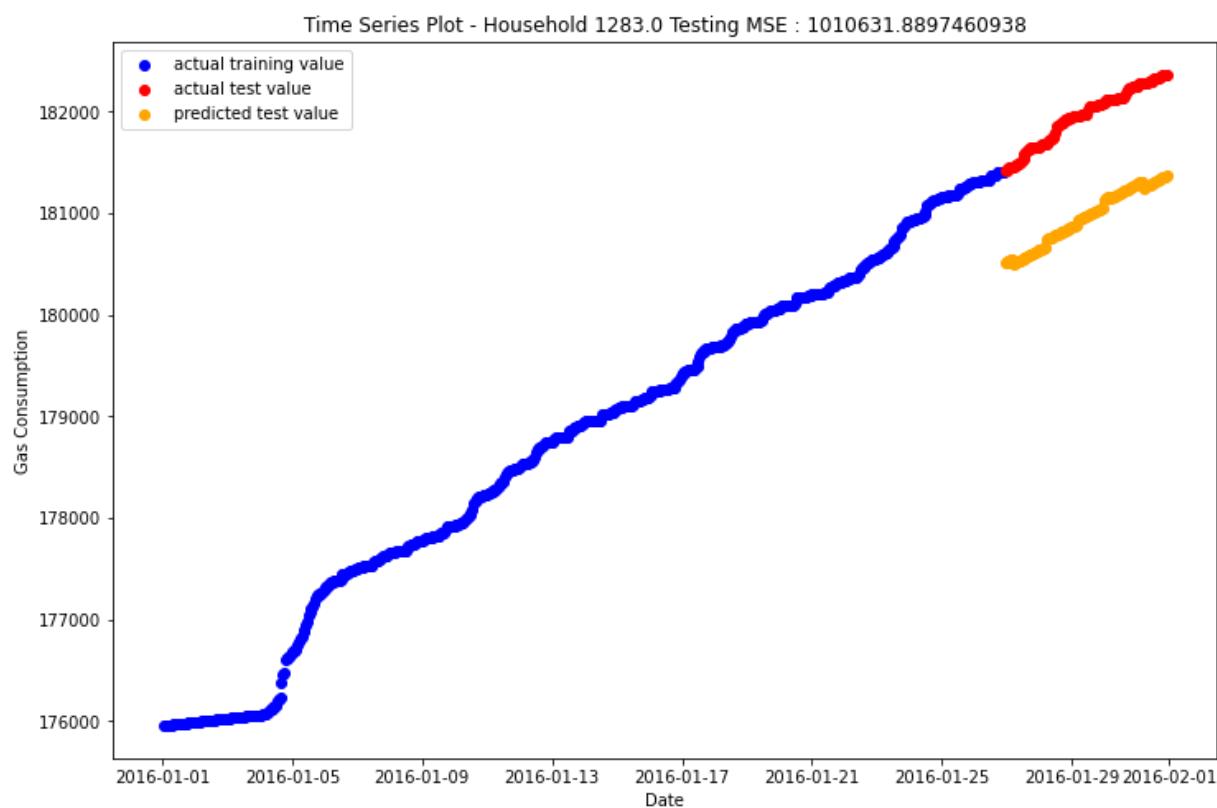
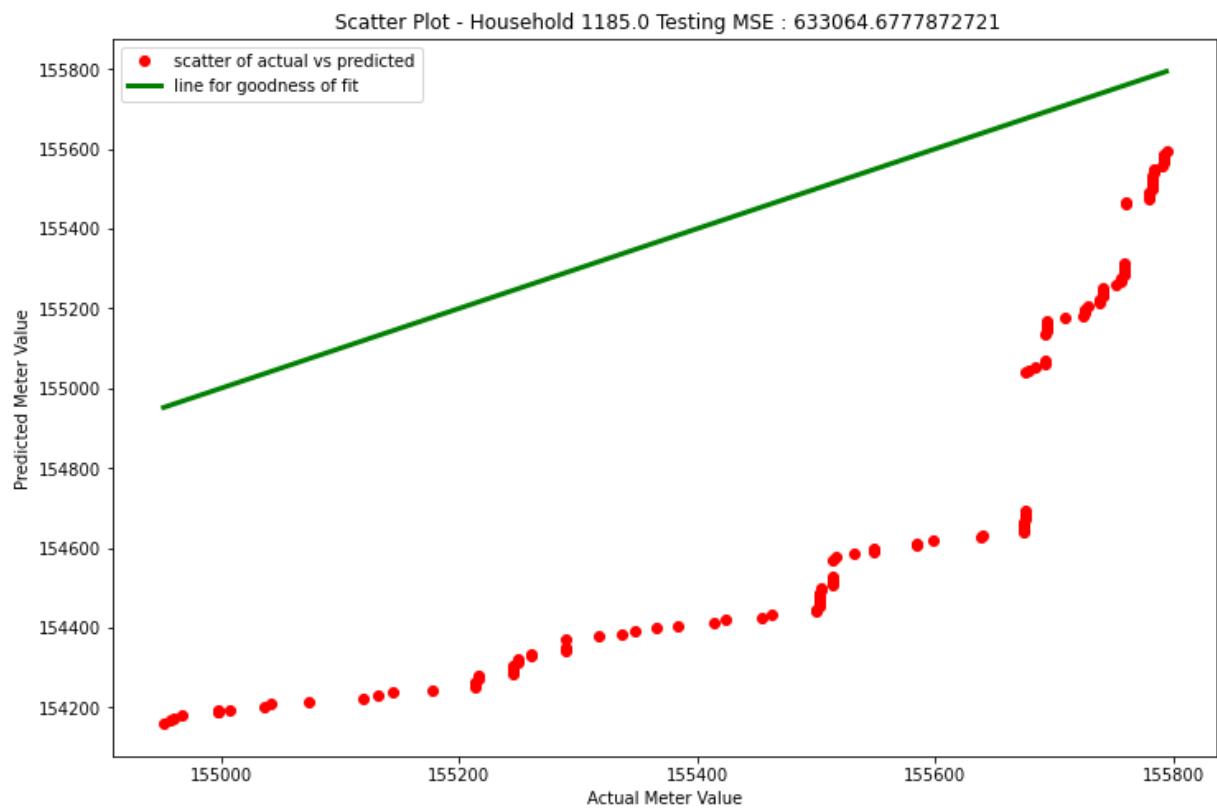


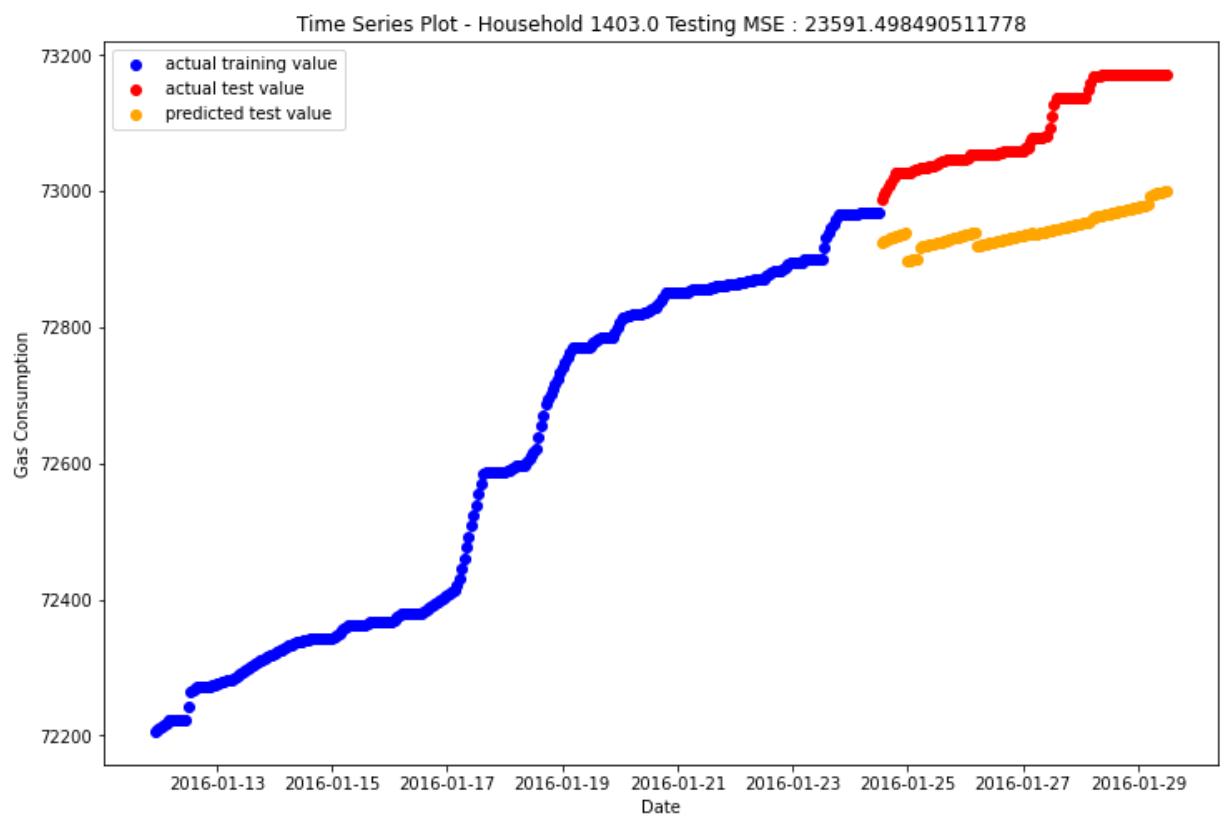
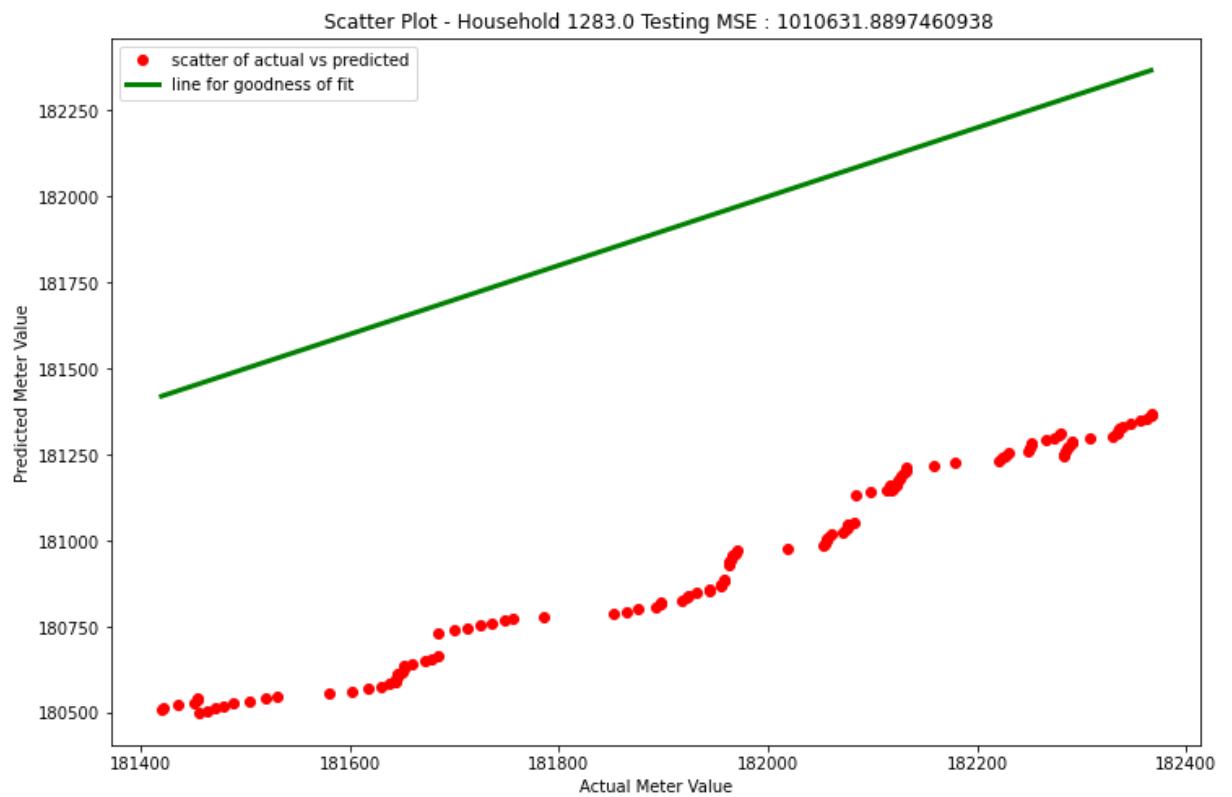
Time Series Plot - Household 1086.0 Testing MSE : 934923.0582555135



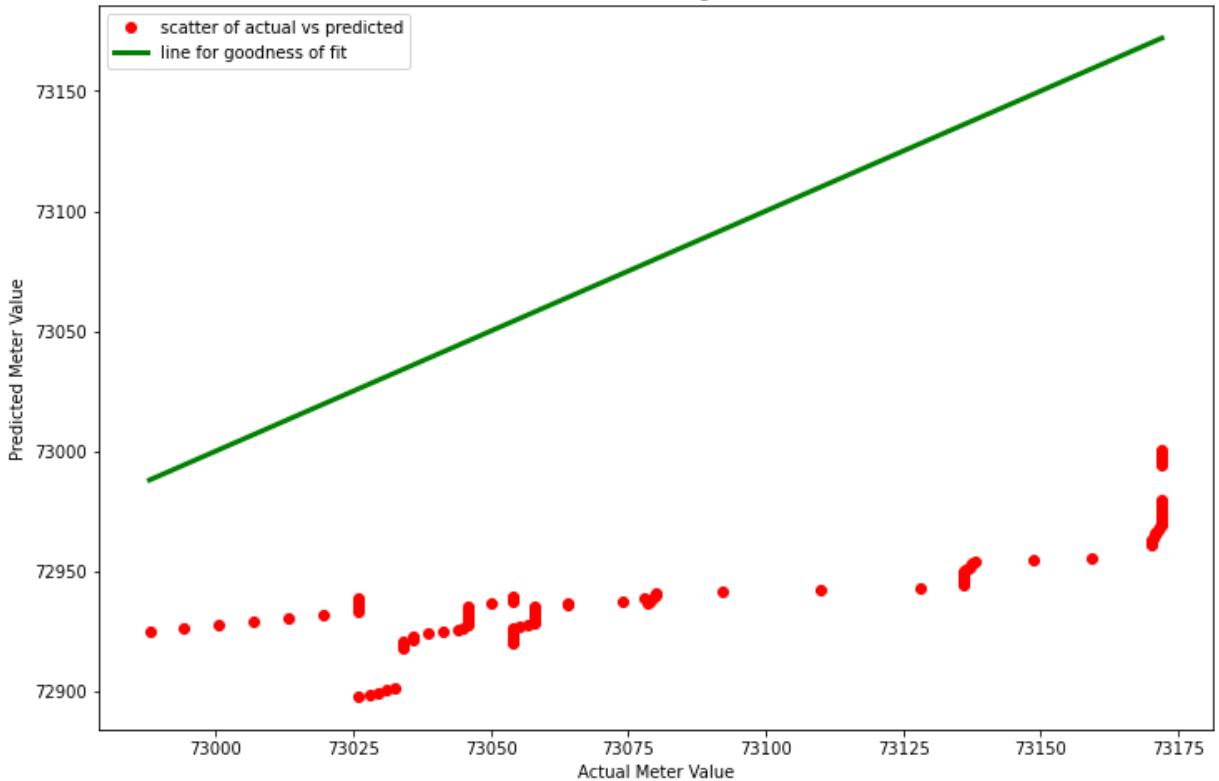




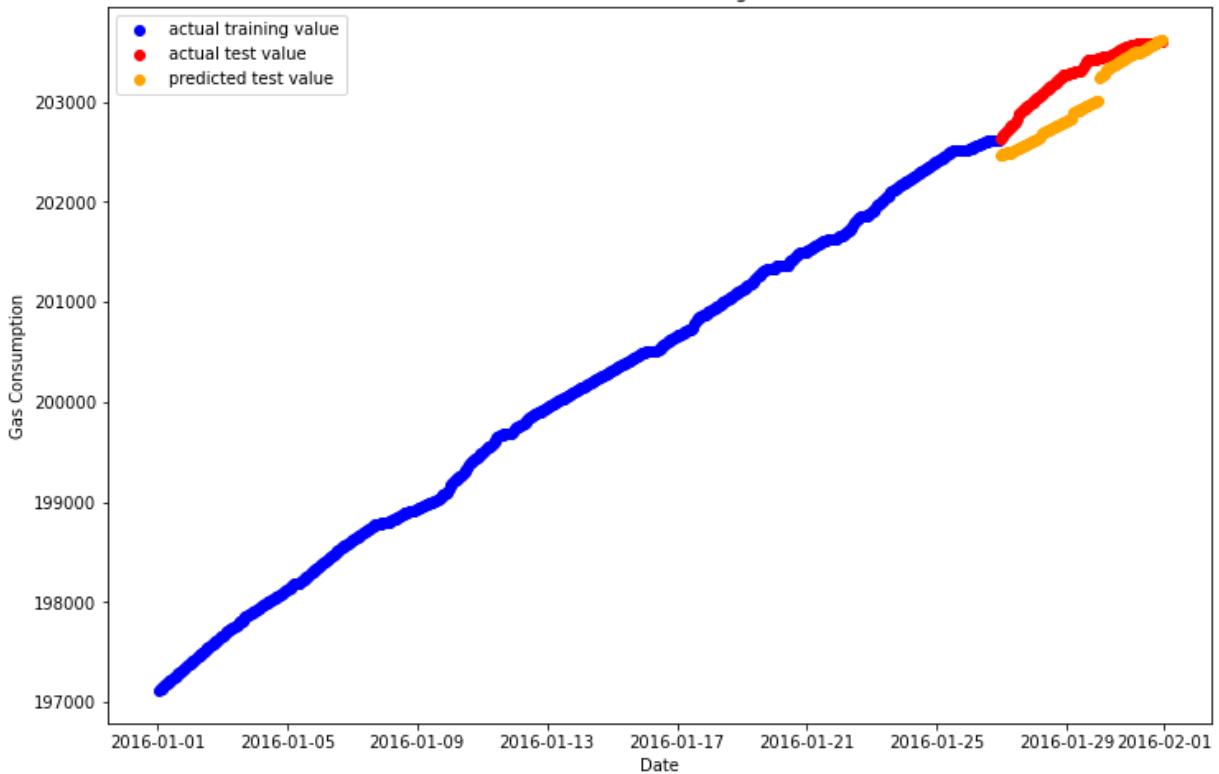


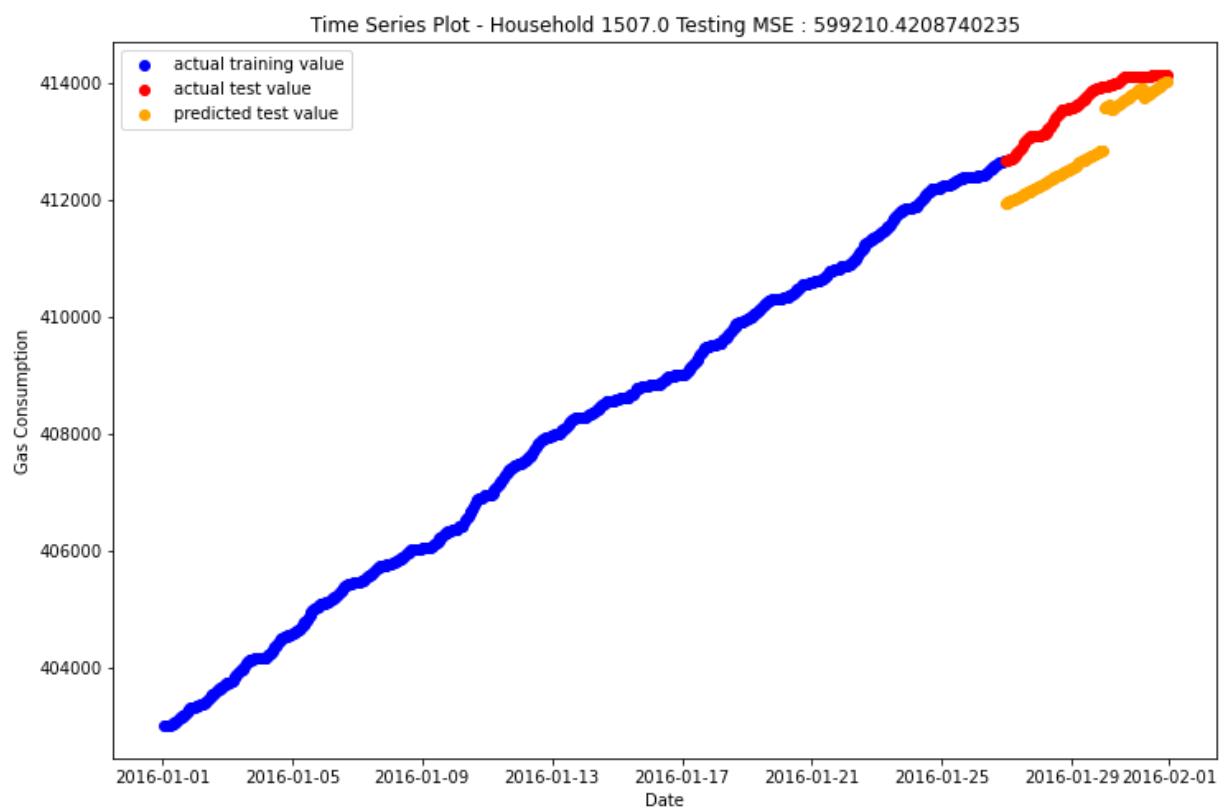
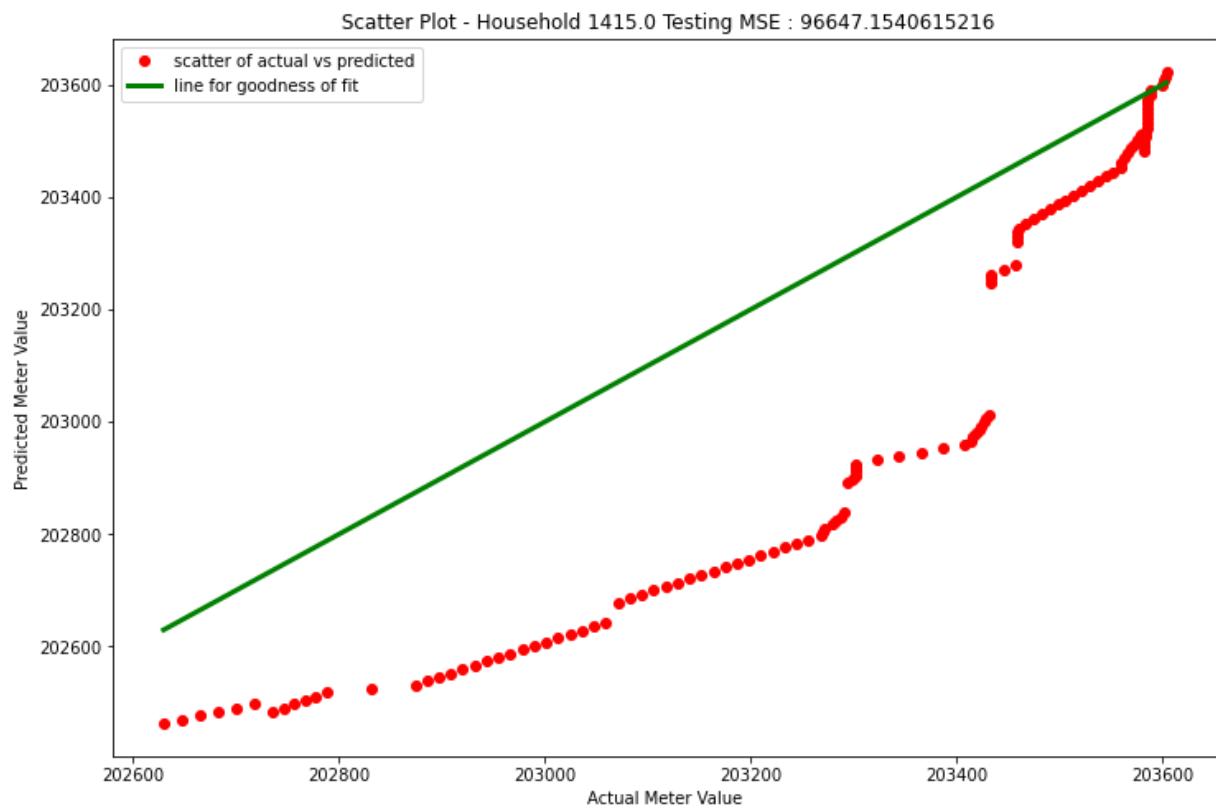


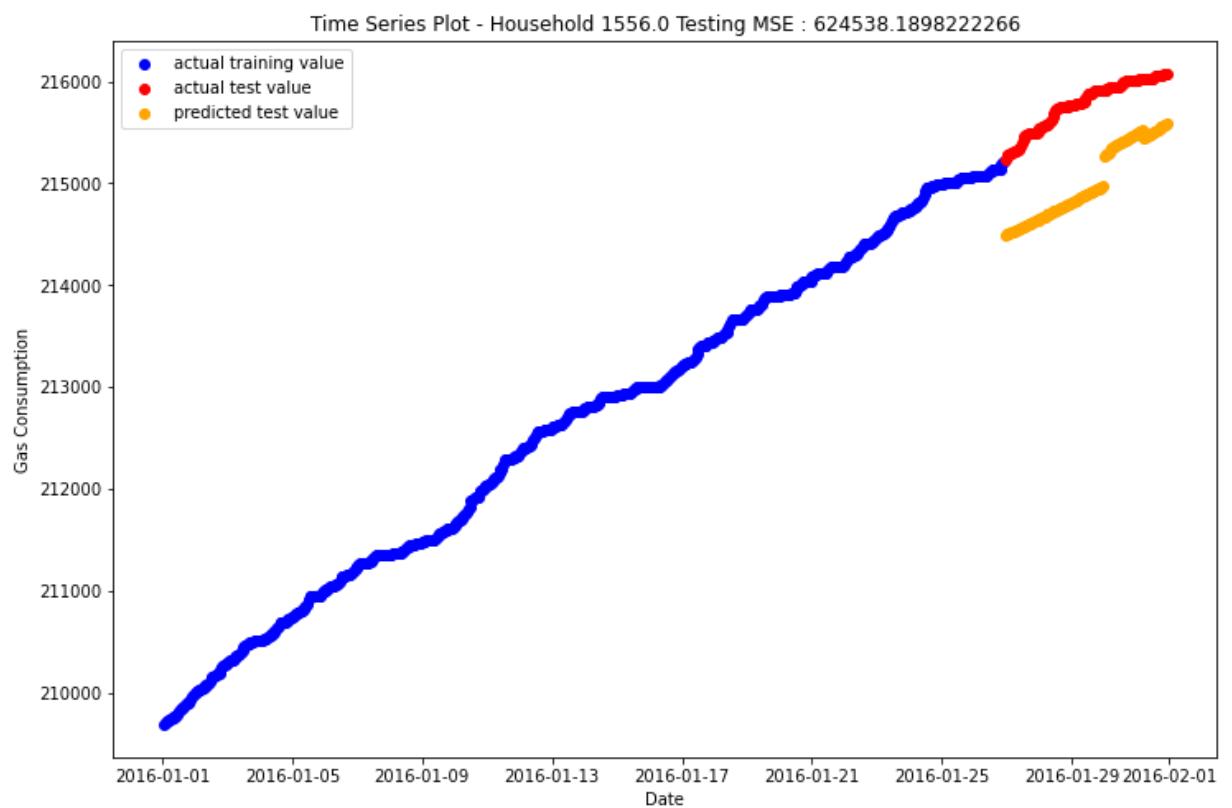
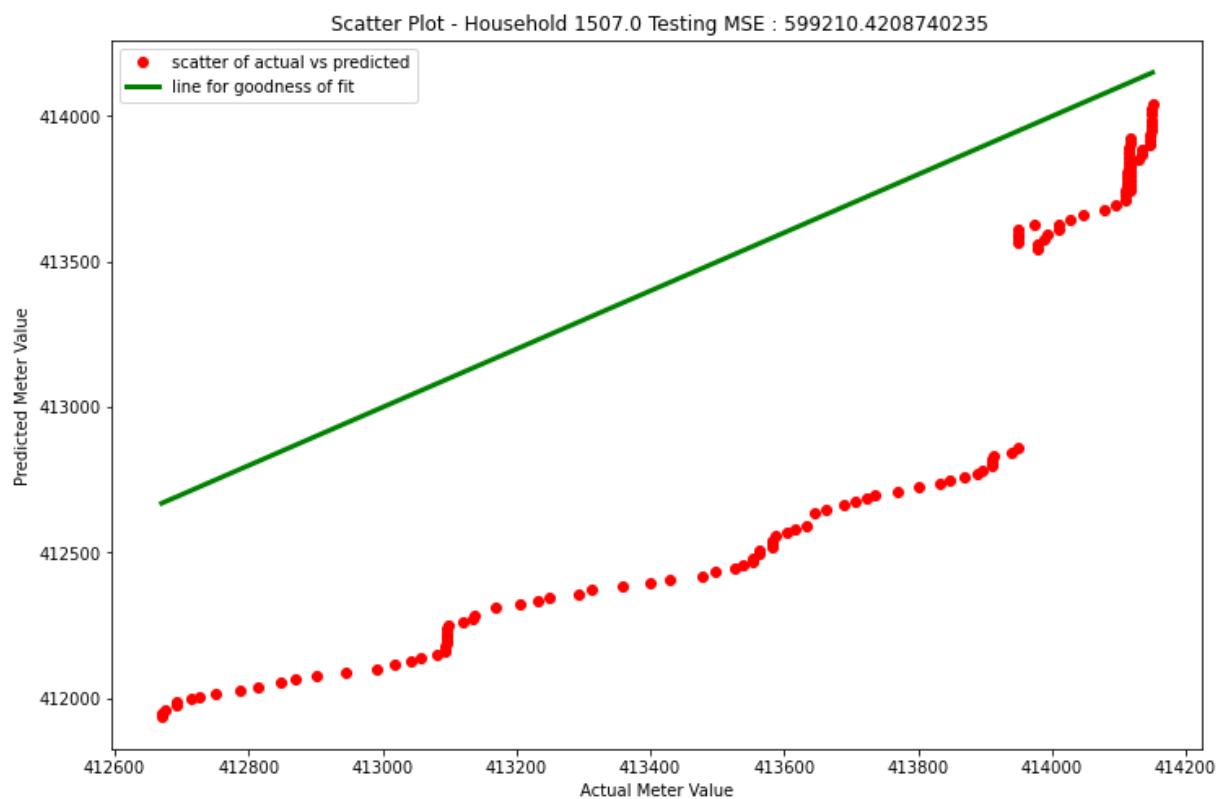
Scatter Plot - Household 1403.0 Testing MSE : 23591.498490511778

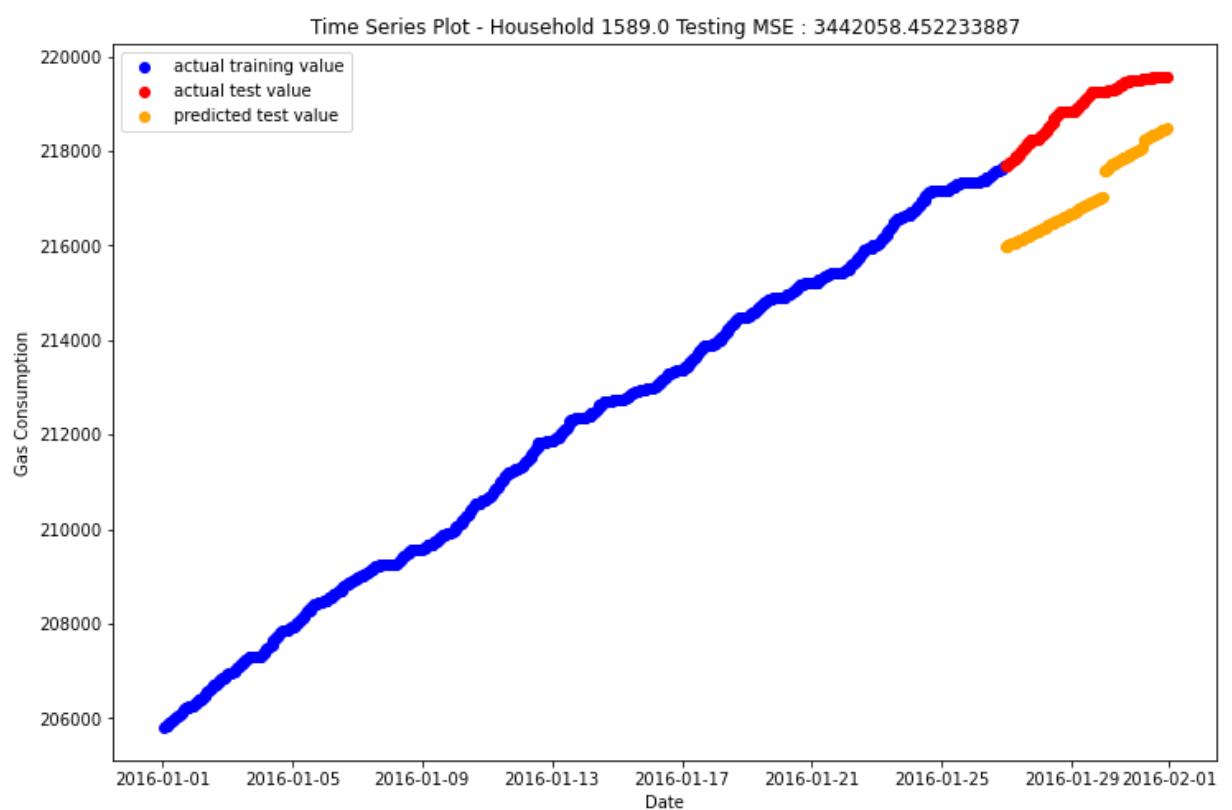
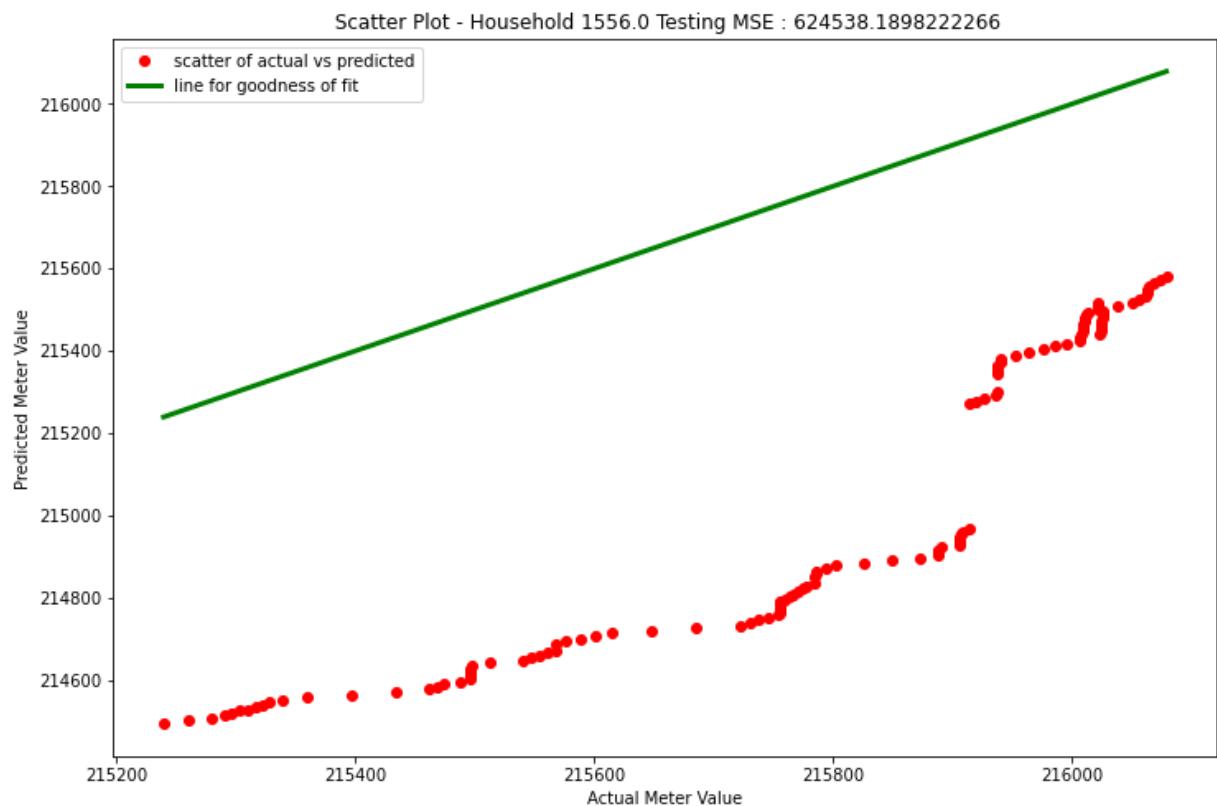


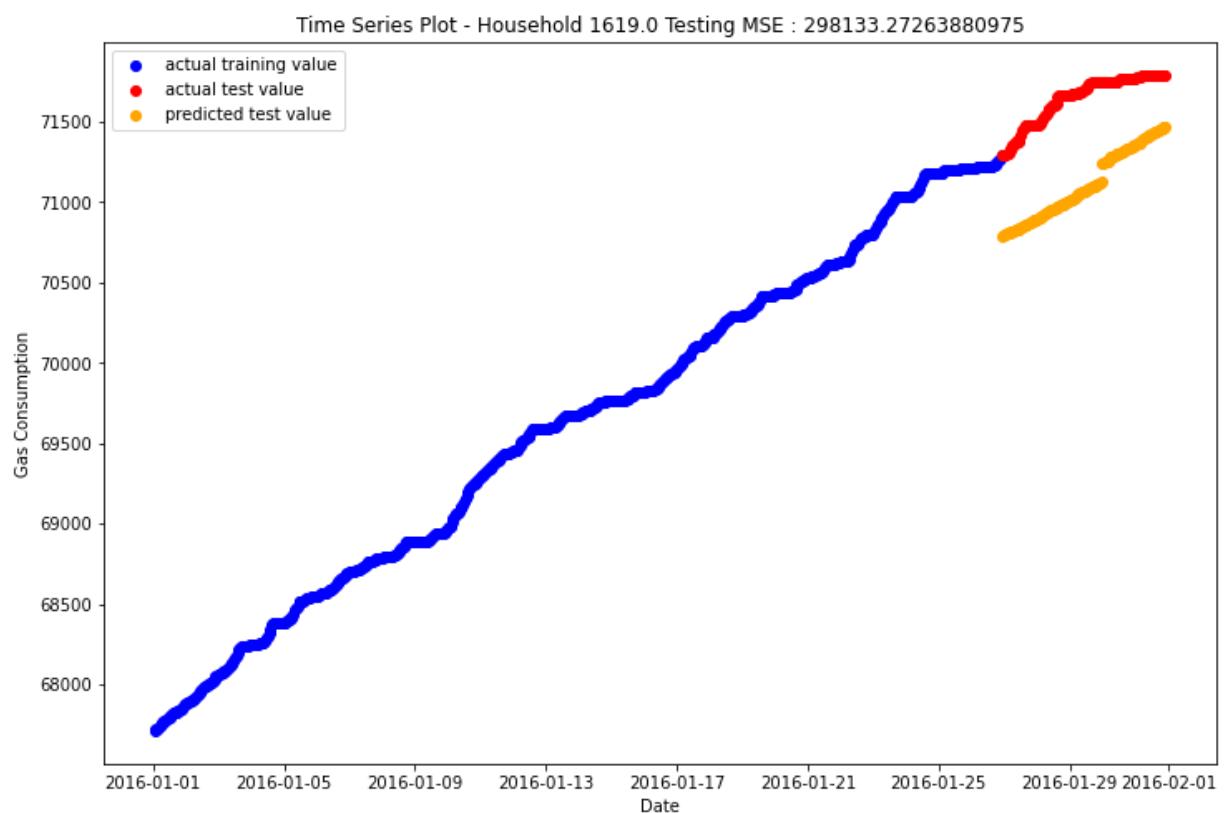
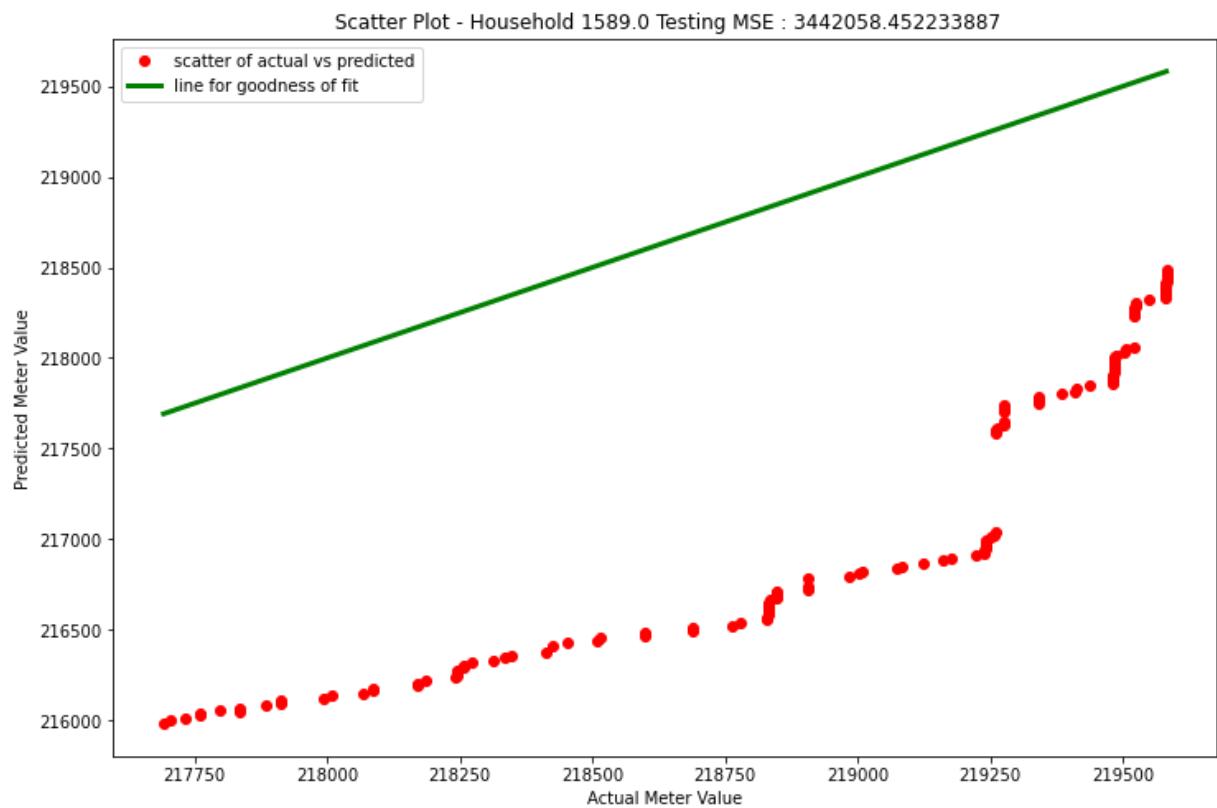
Time Series Plot - Household 1415.0 Testing MSE : 96647.1540615216

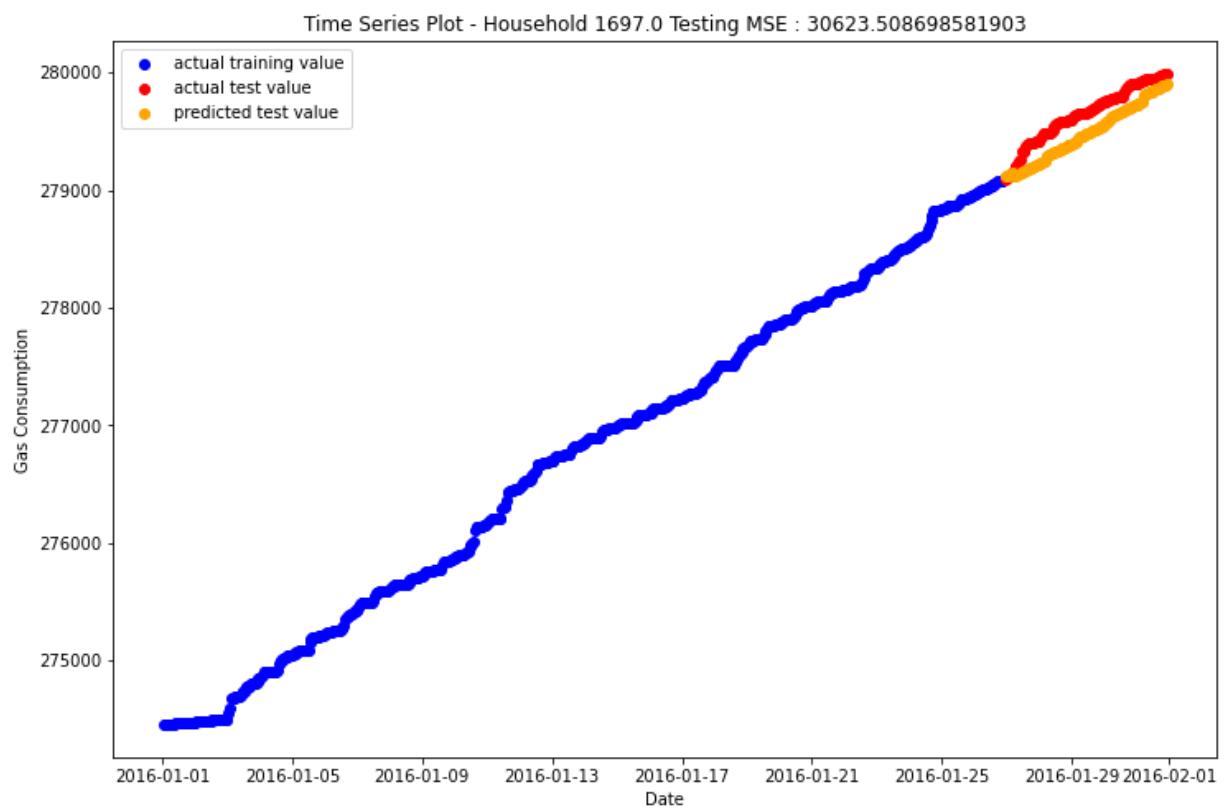
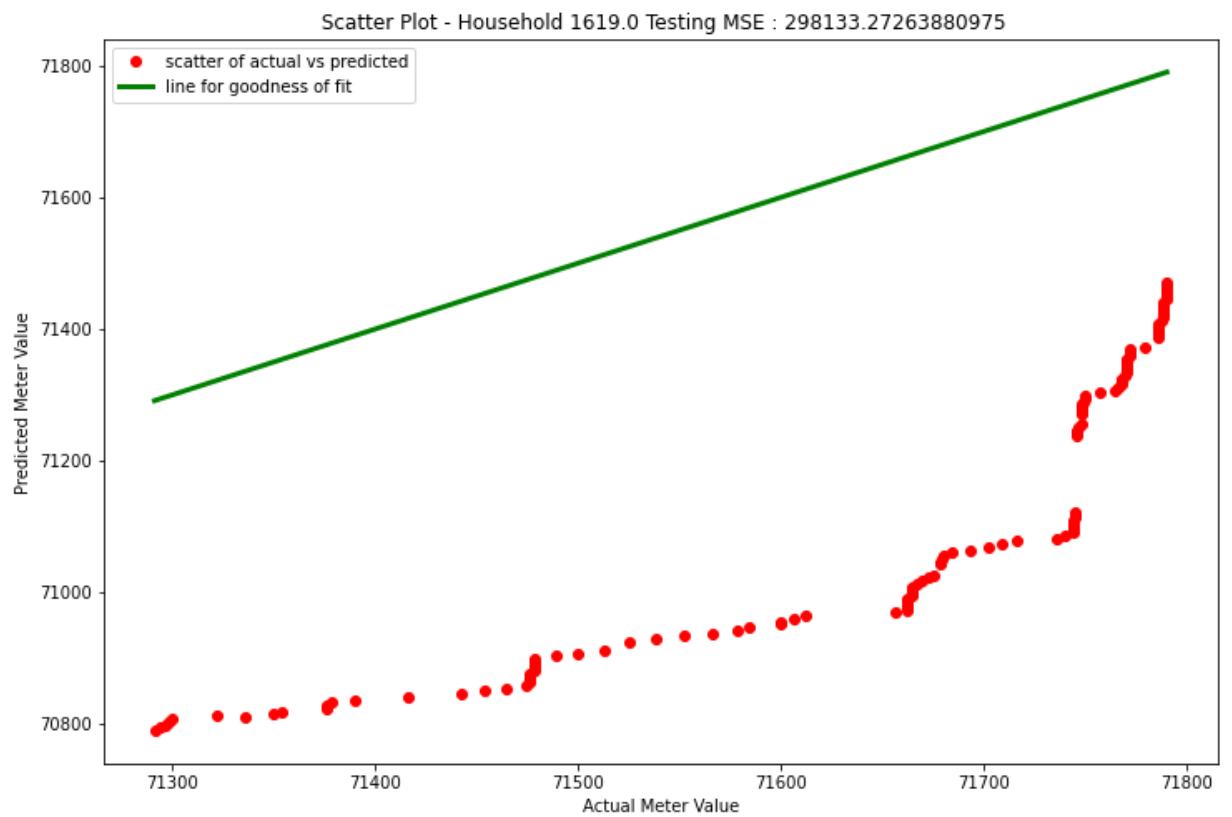


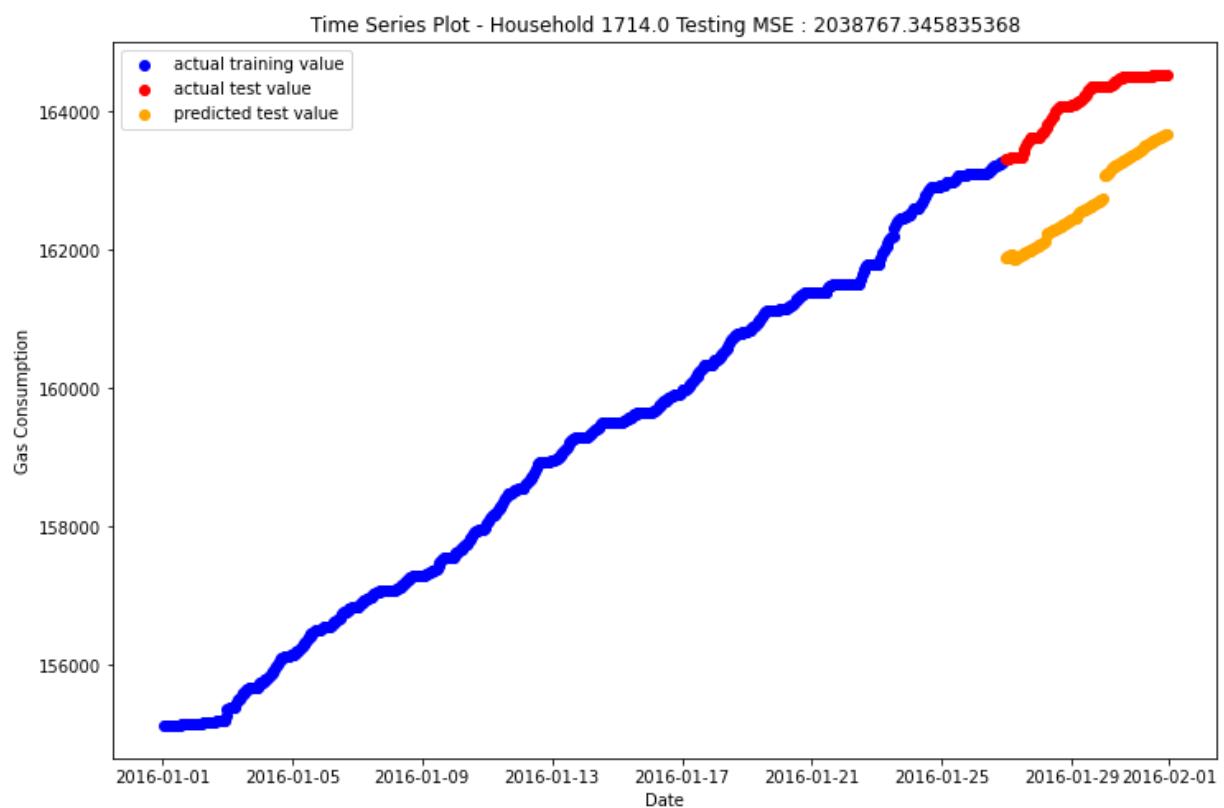
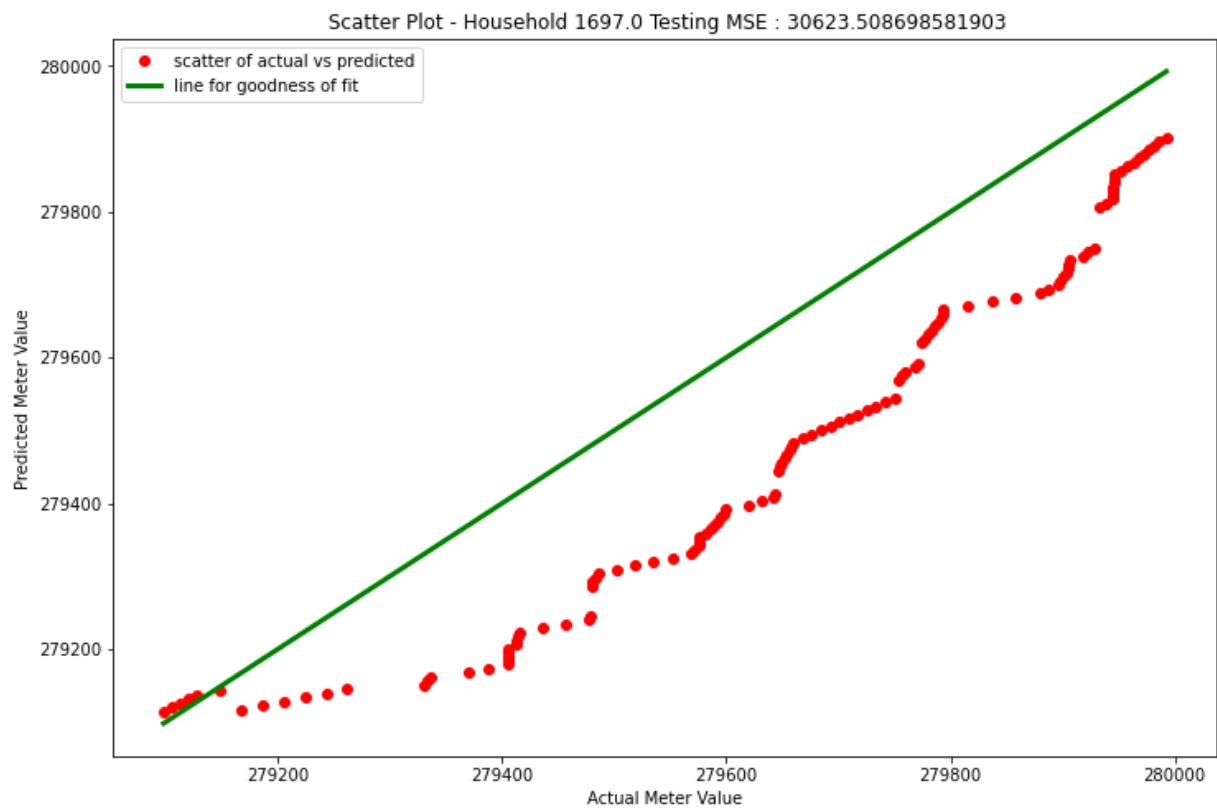


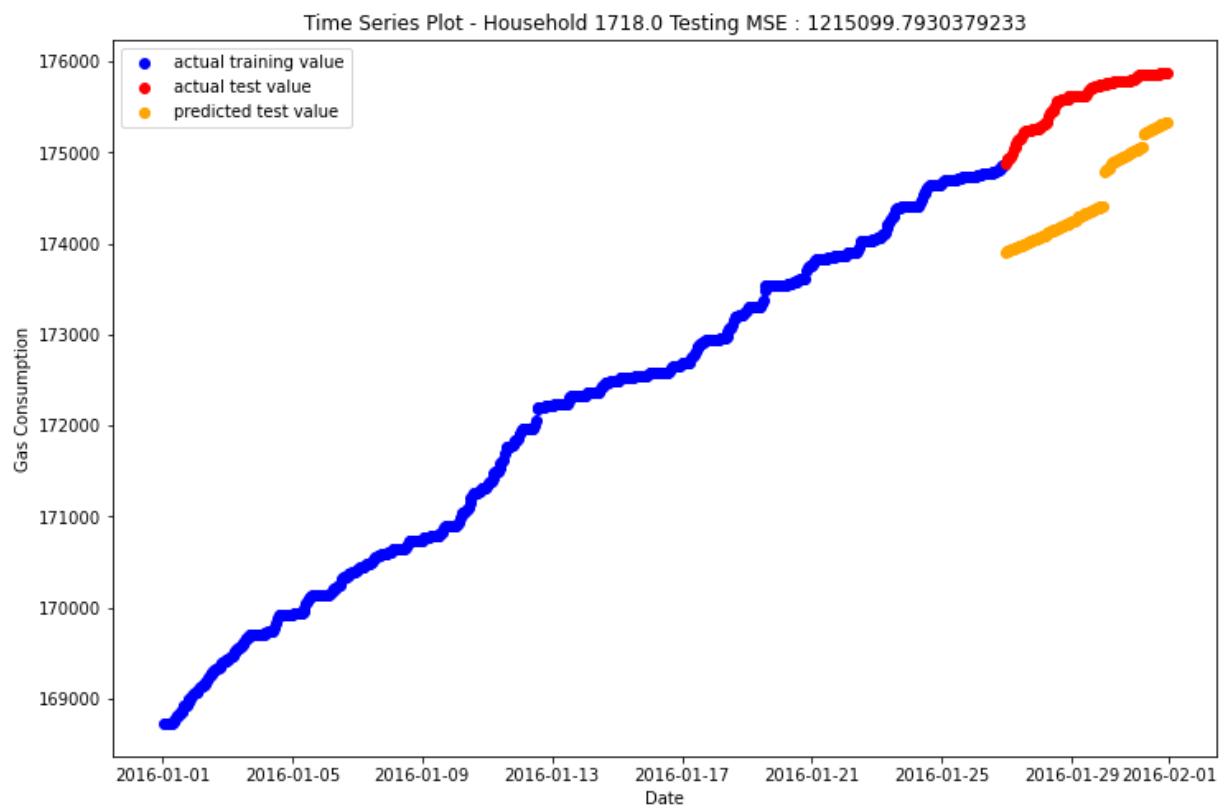
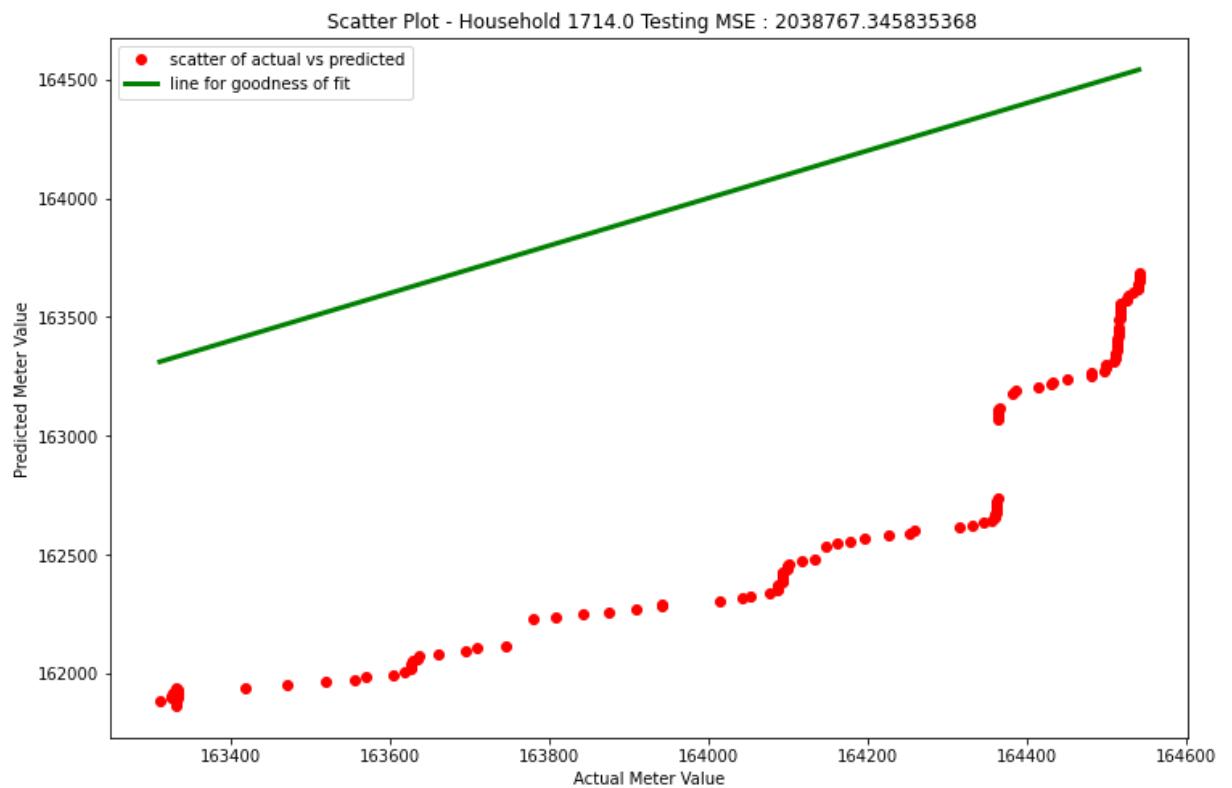


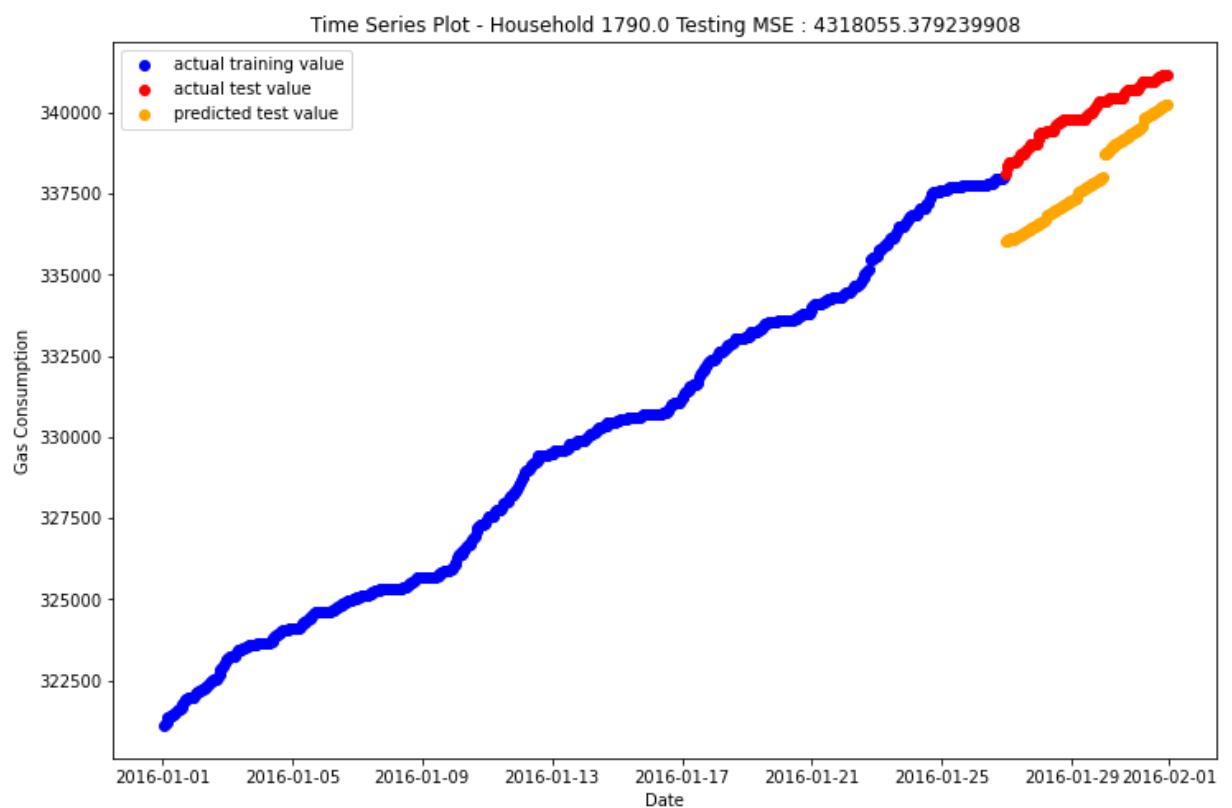
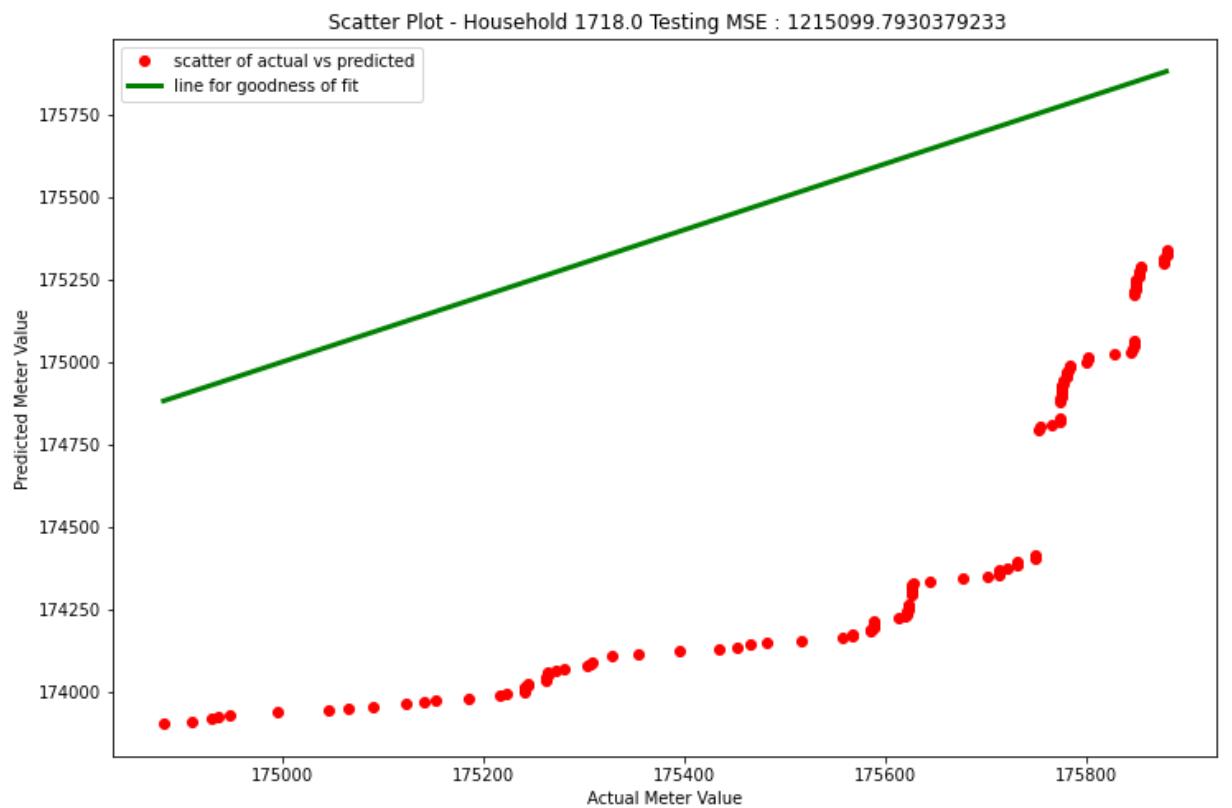


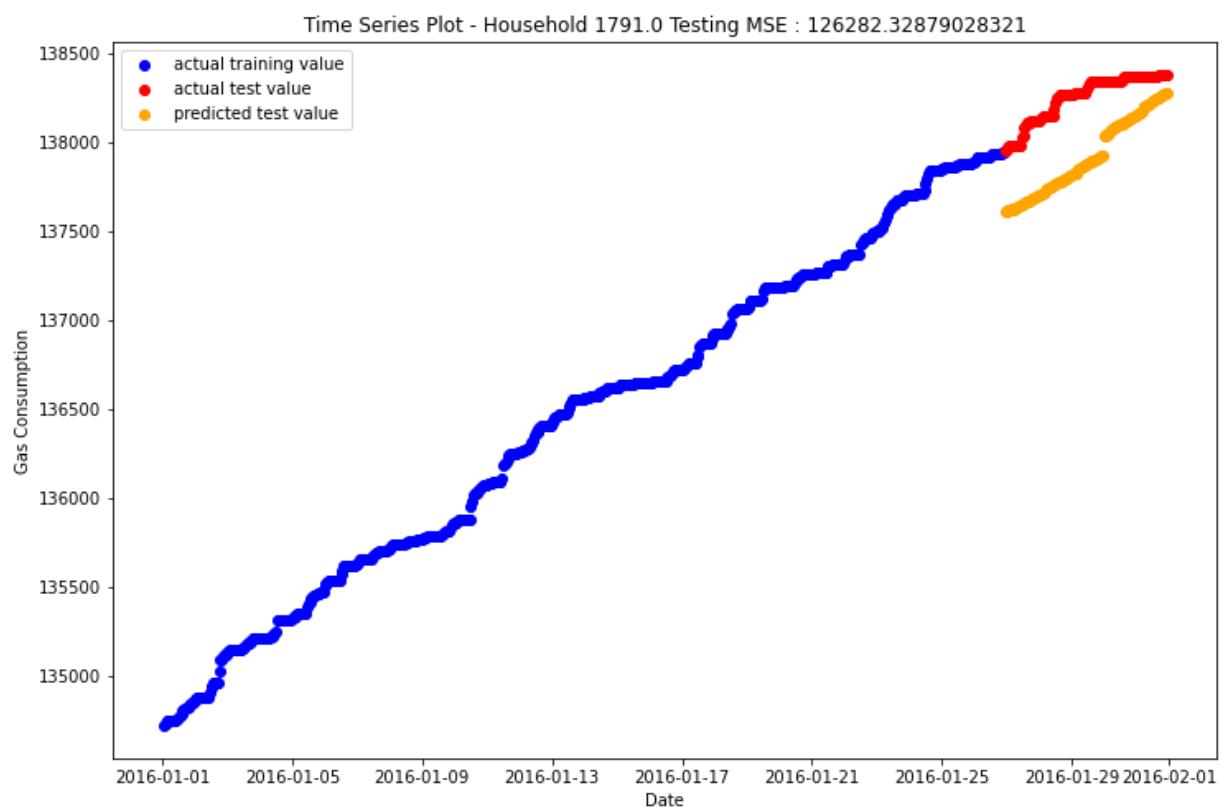
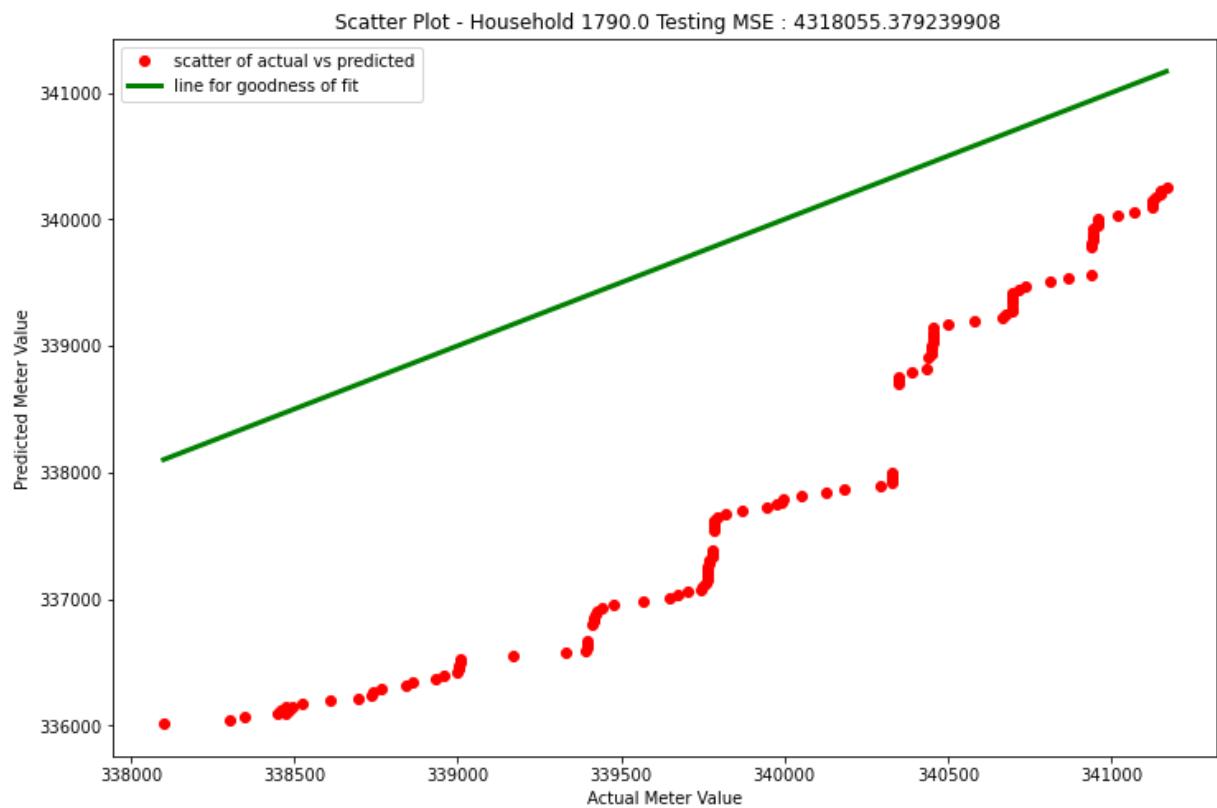


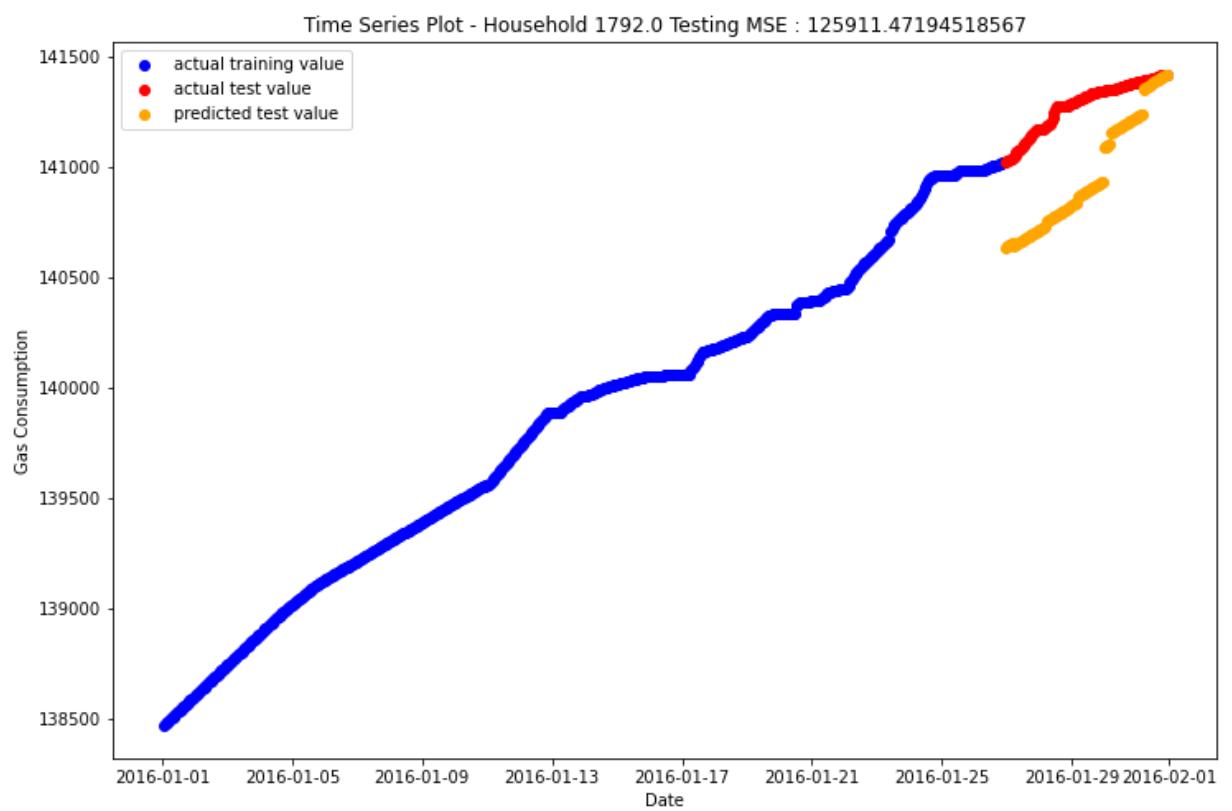
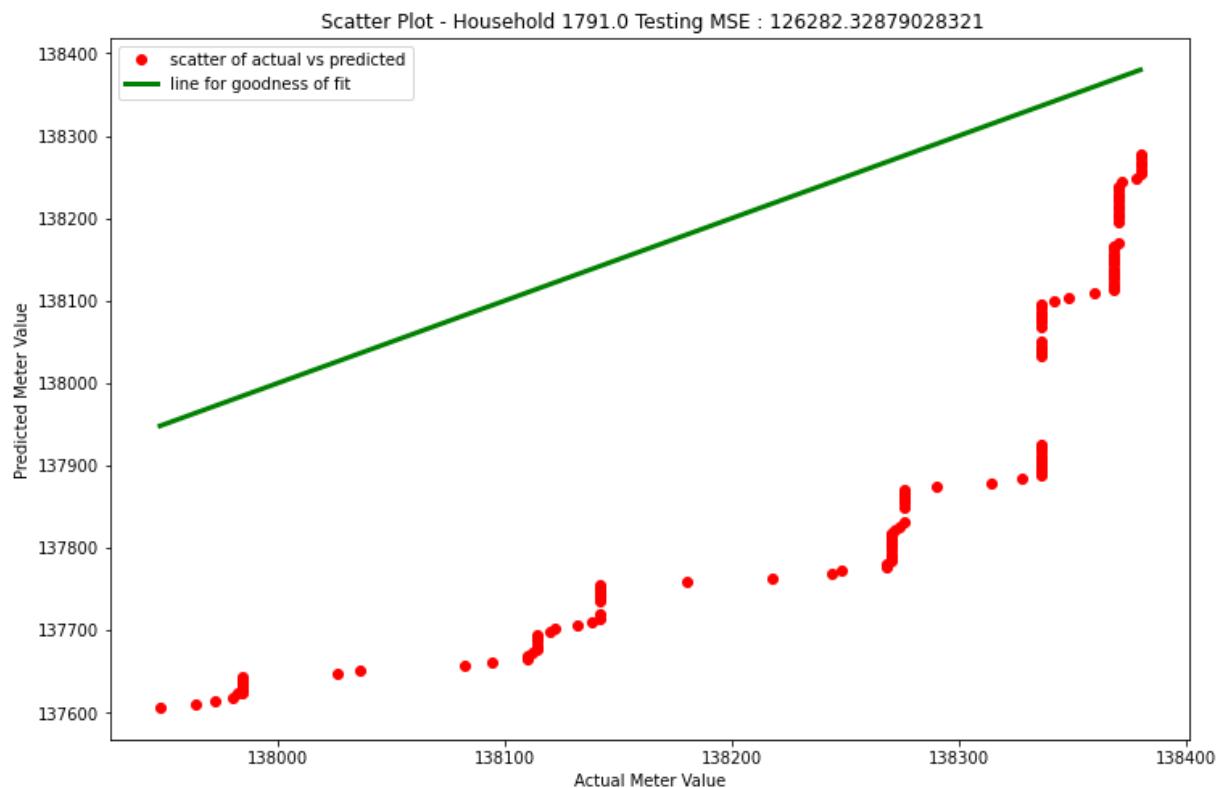


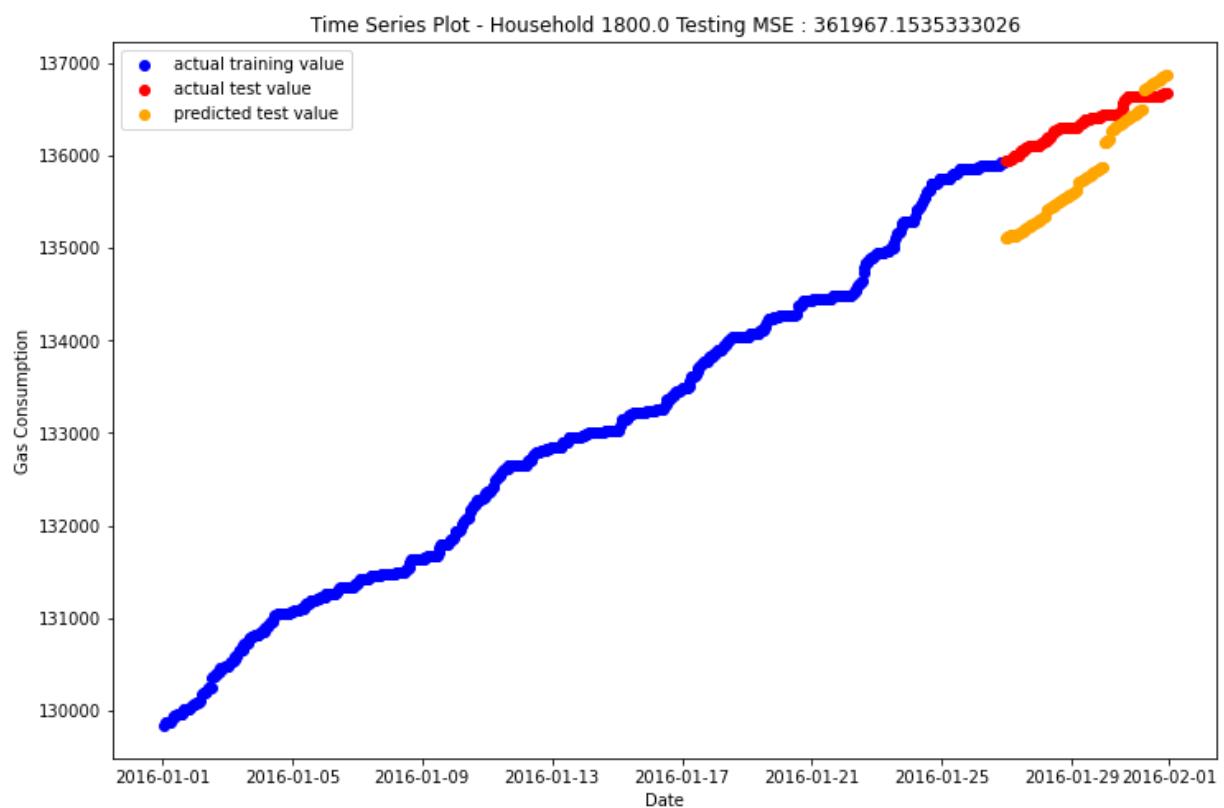
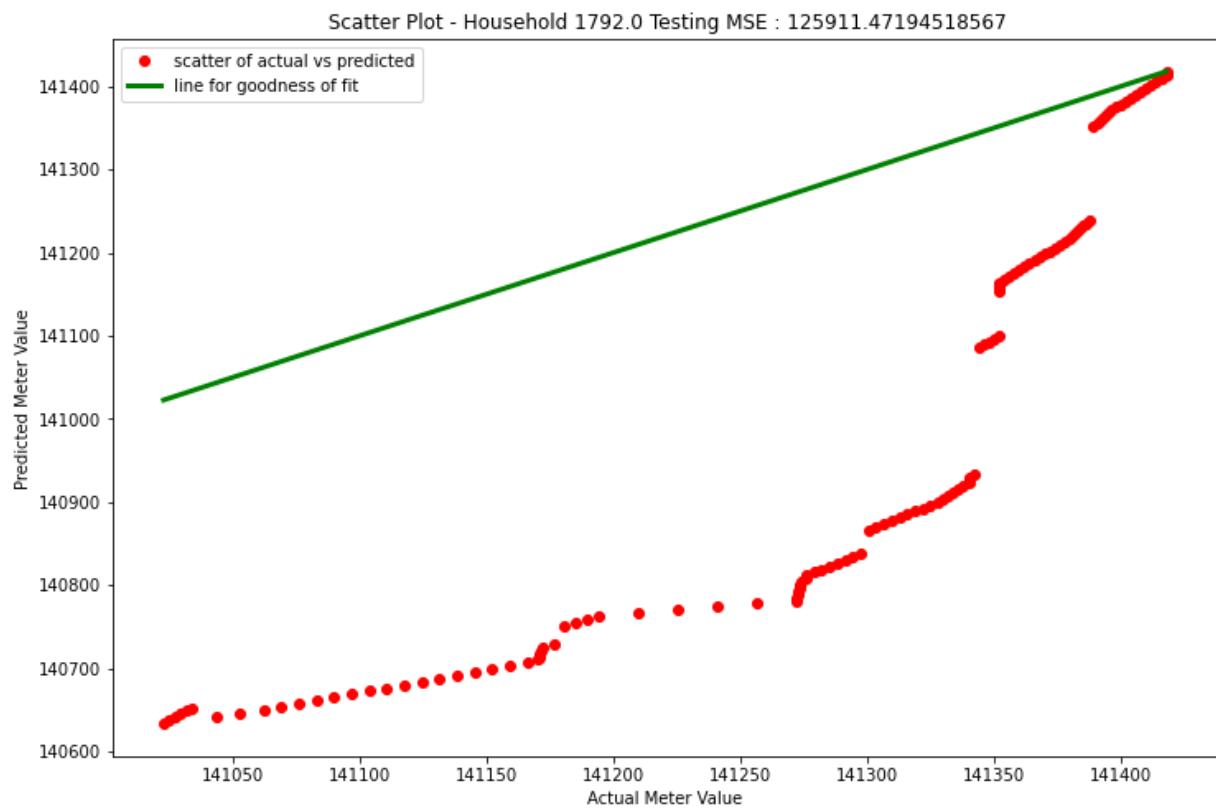


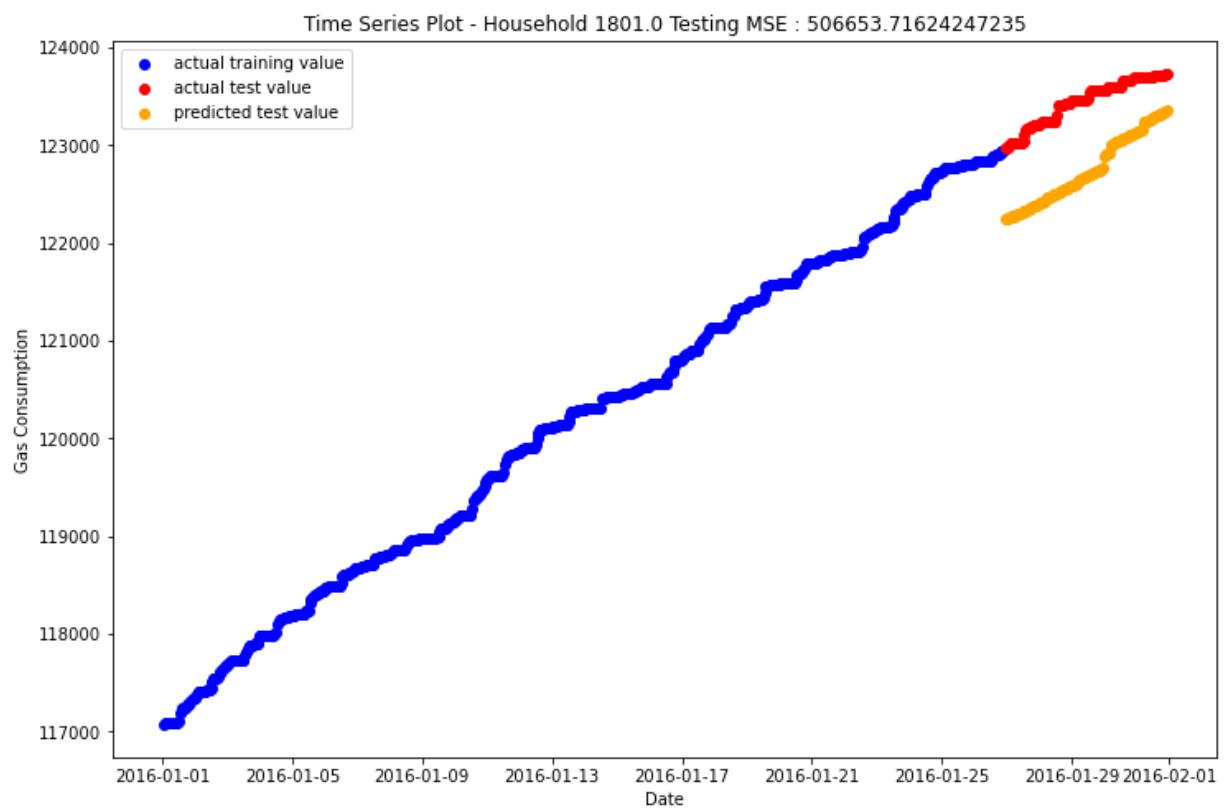
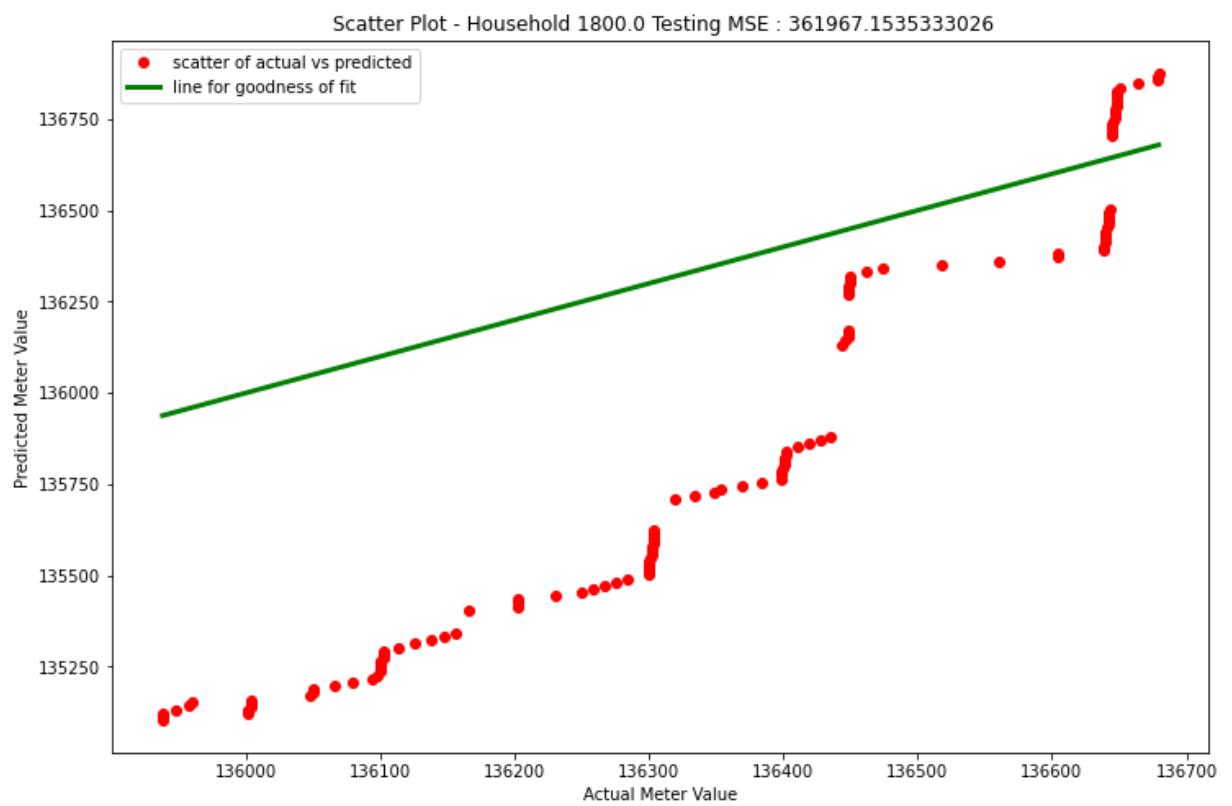


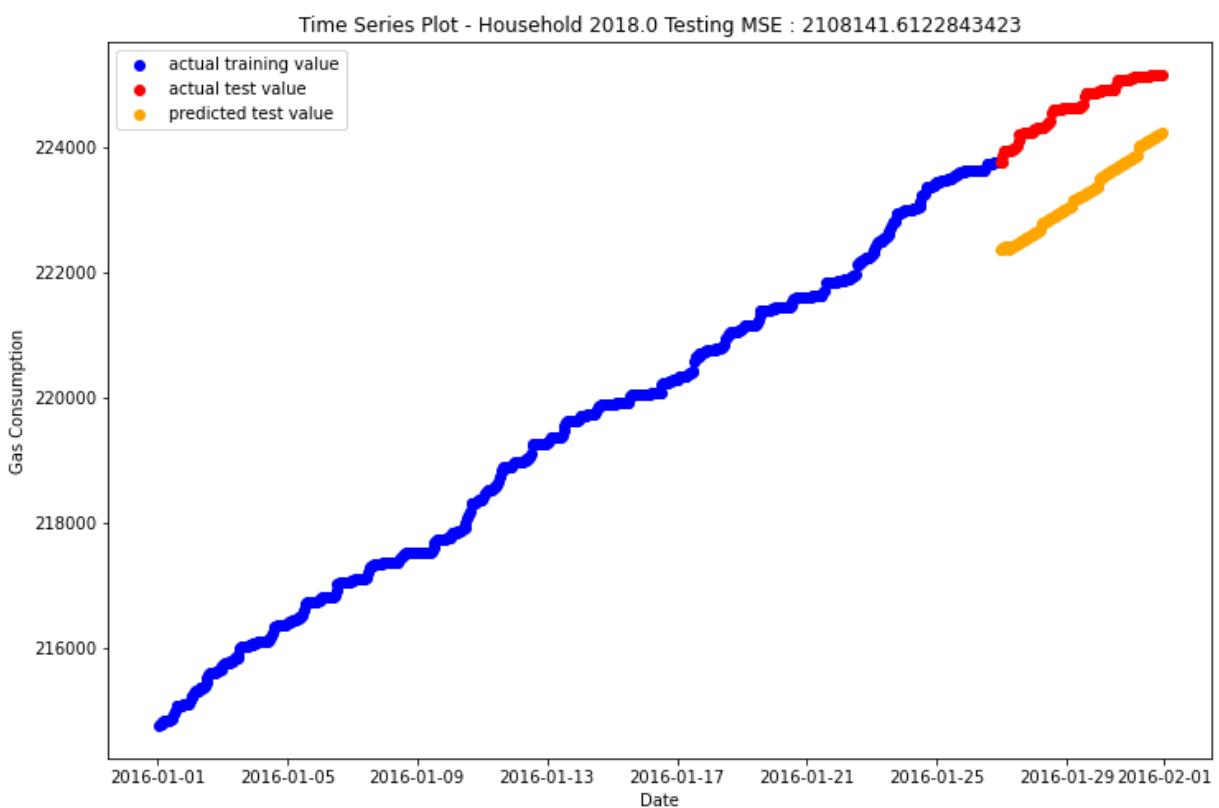
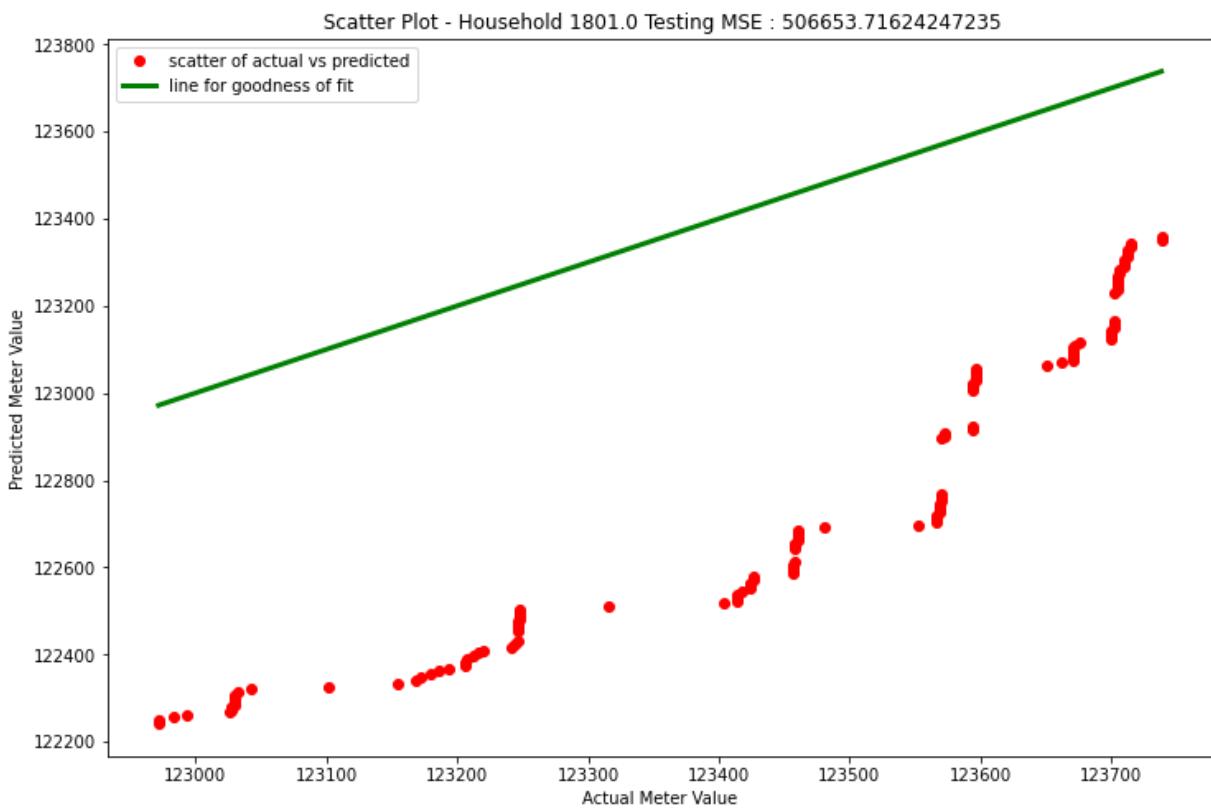


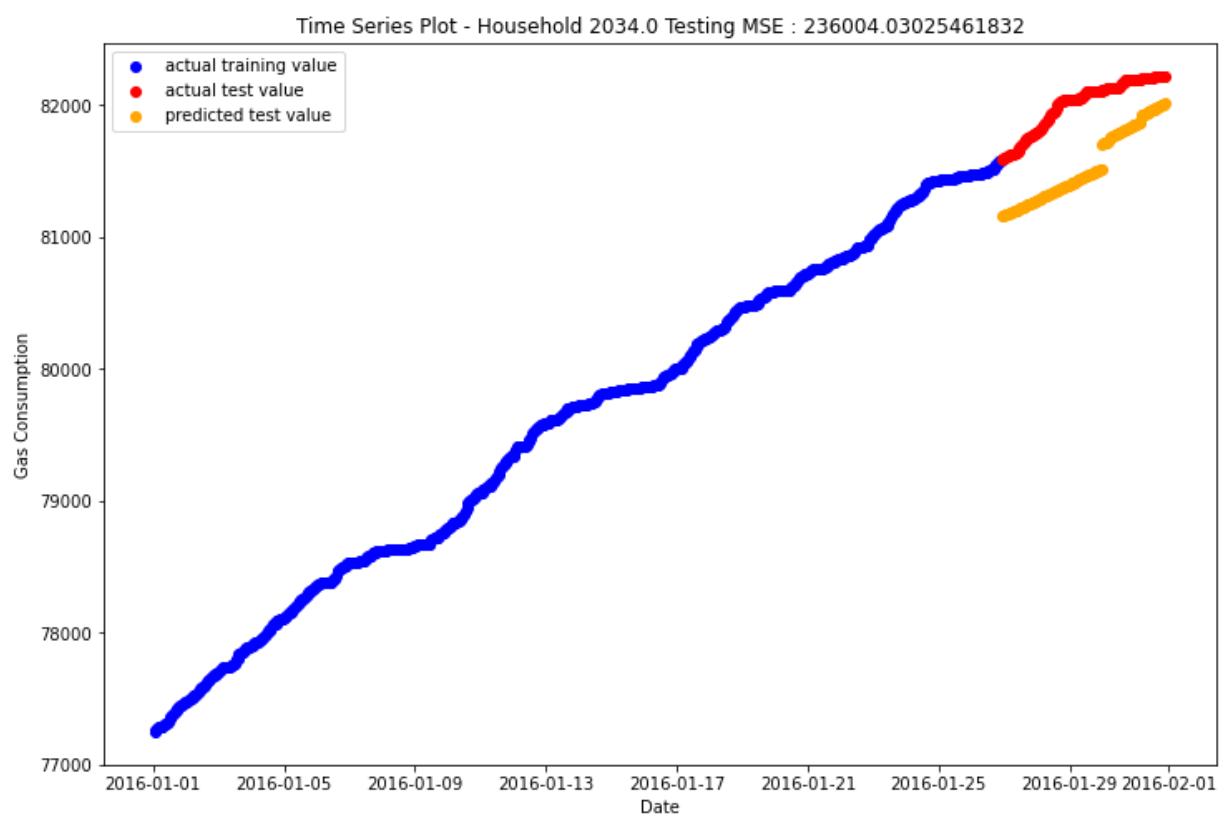
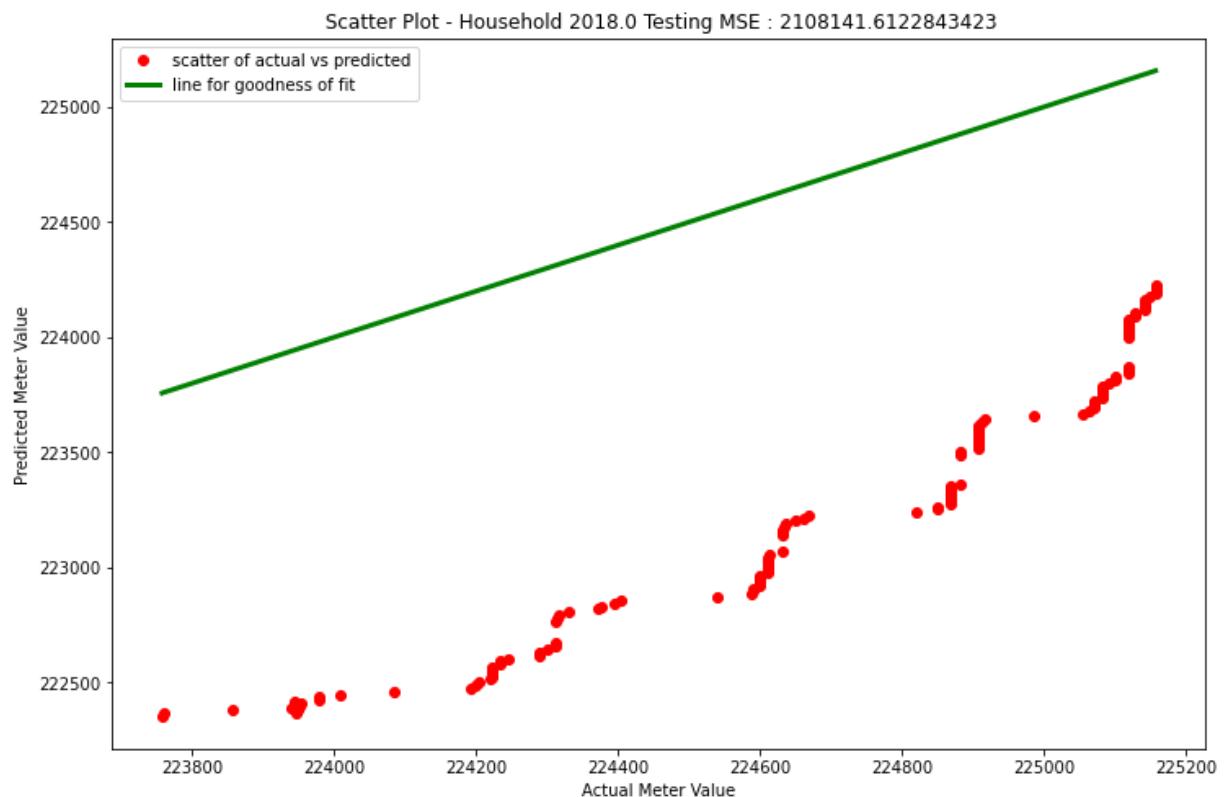


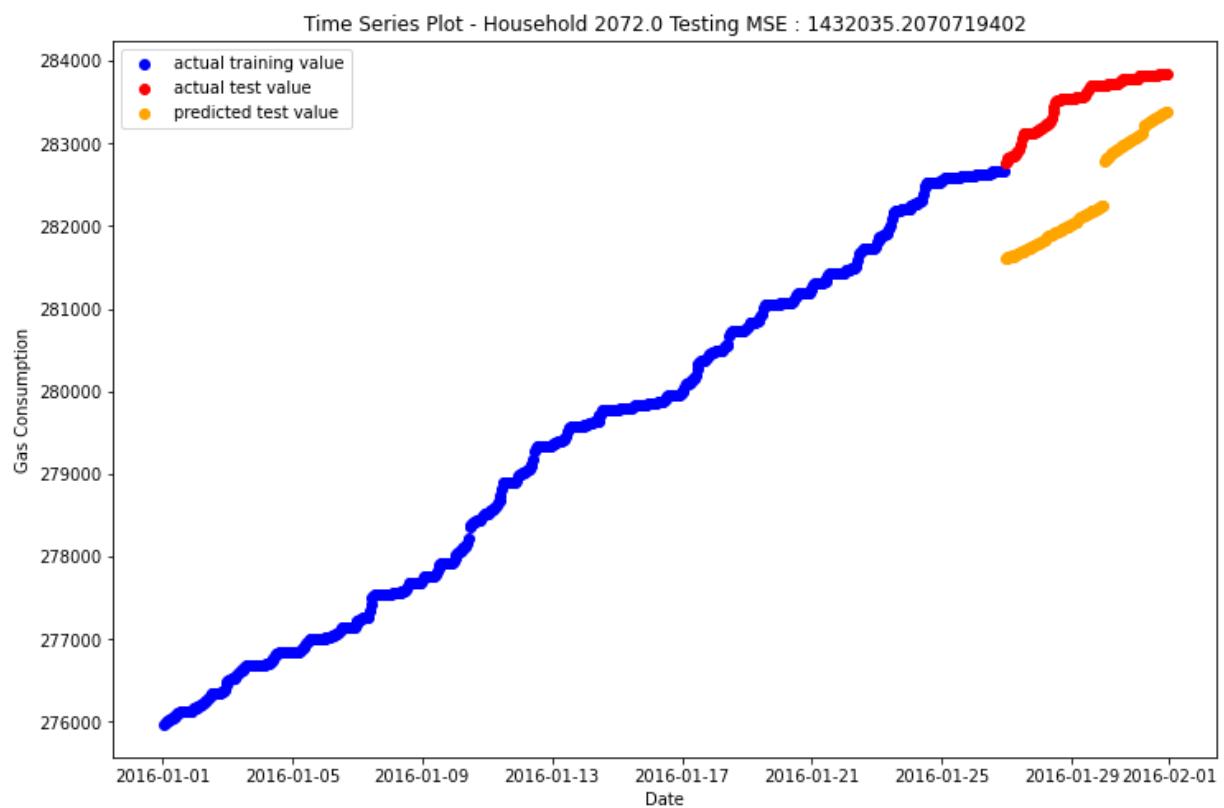
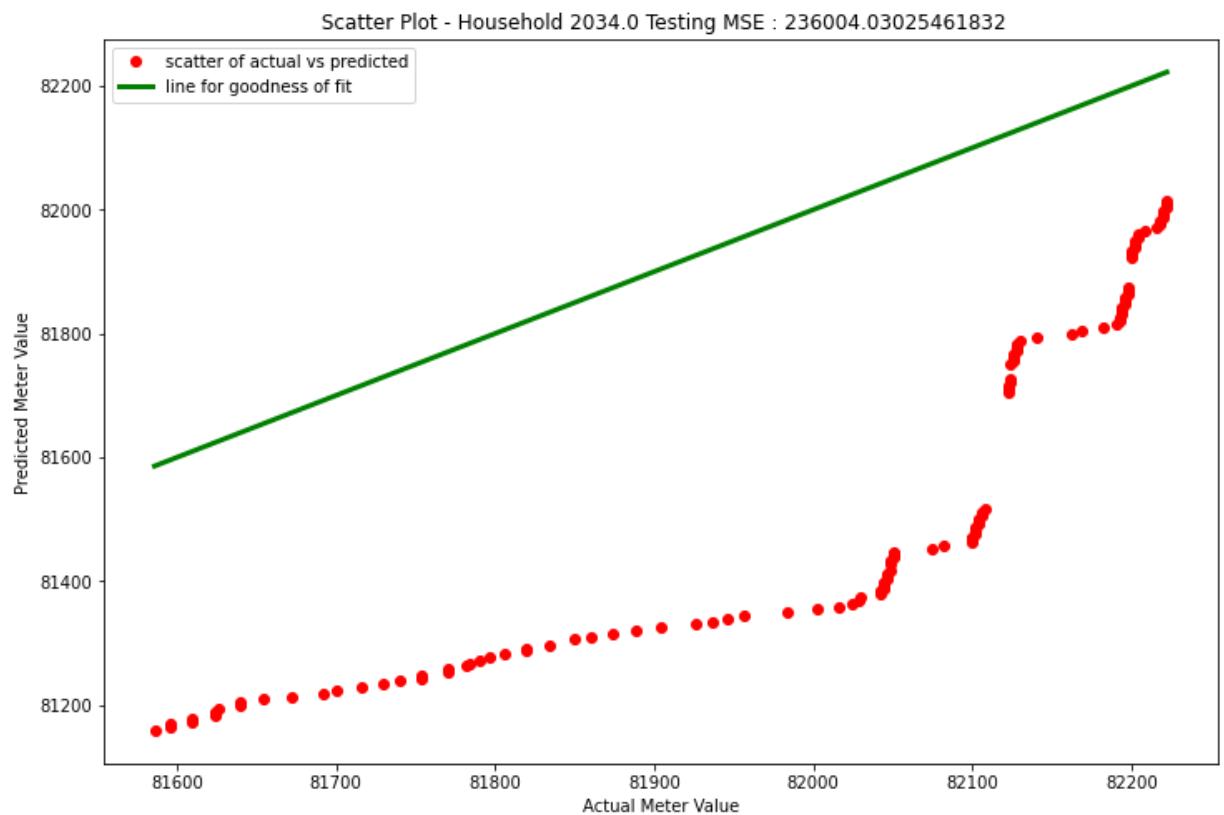


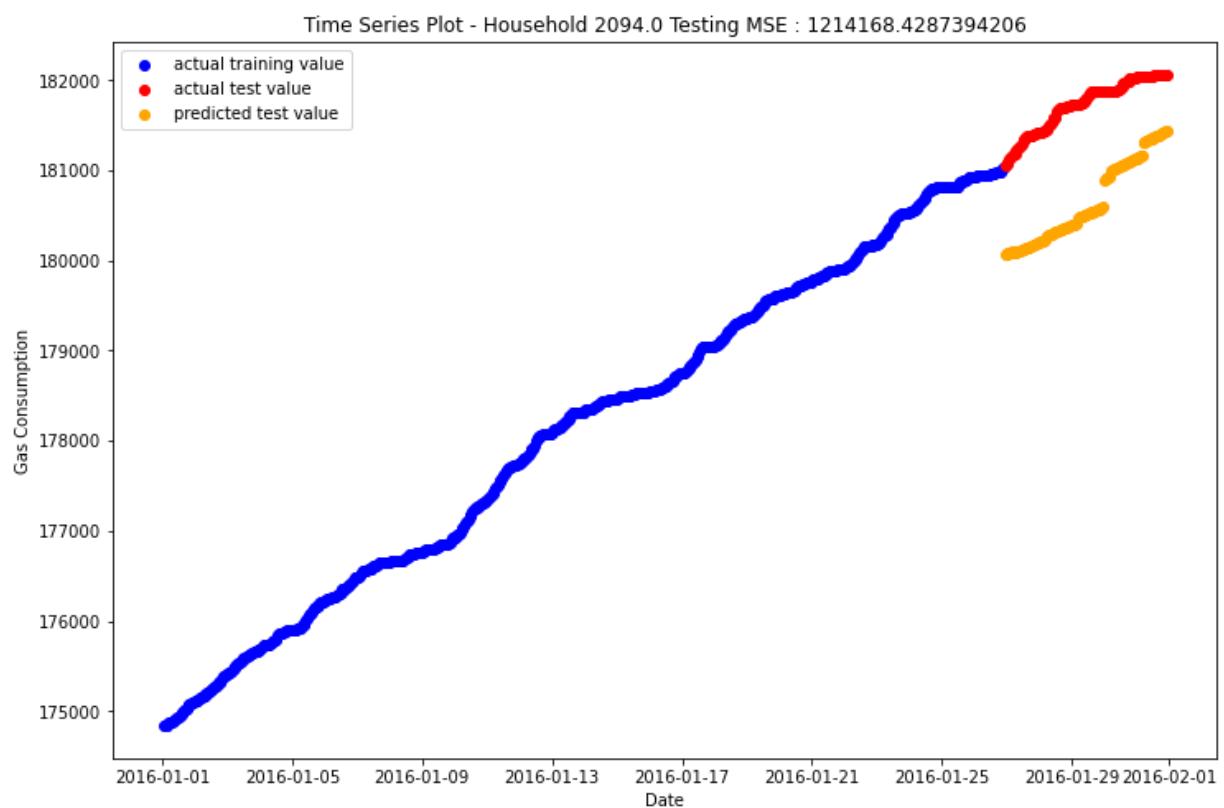
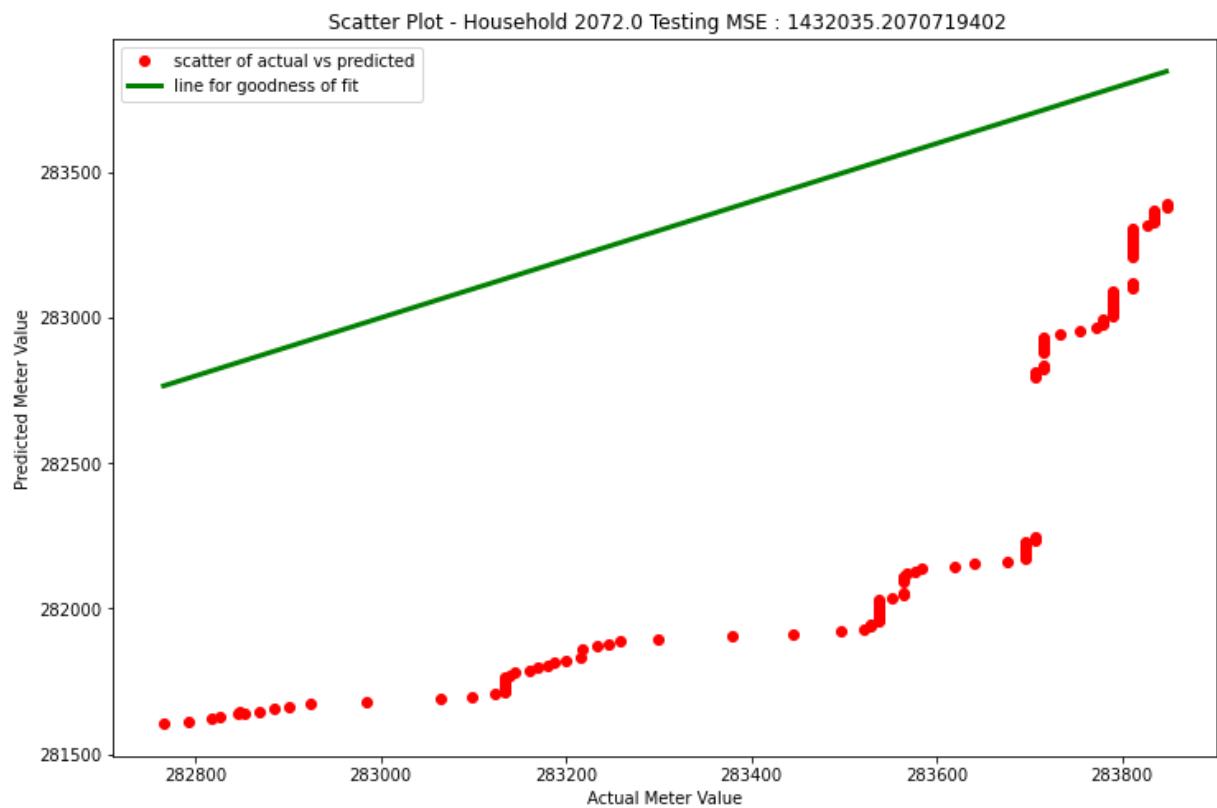


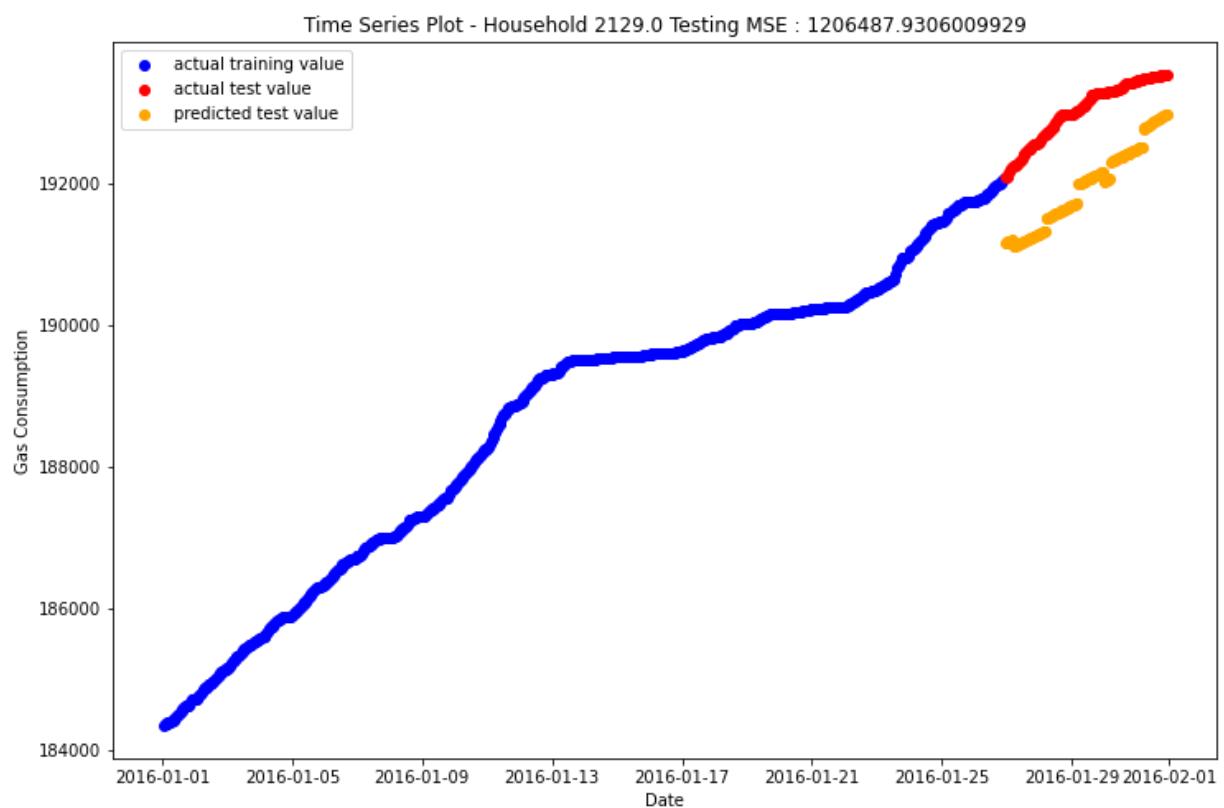
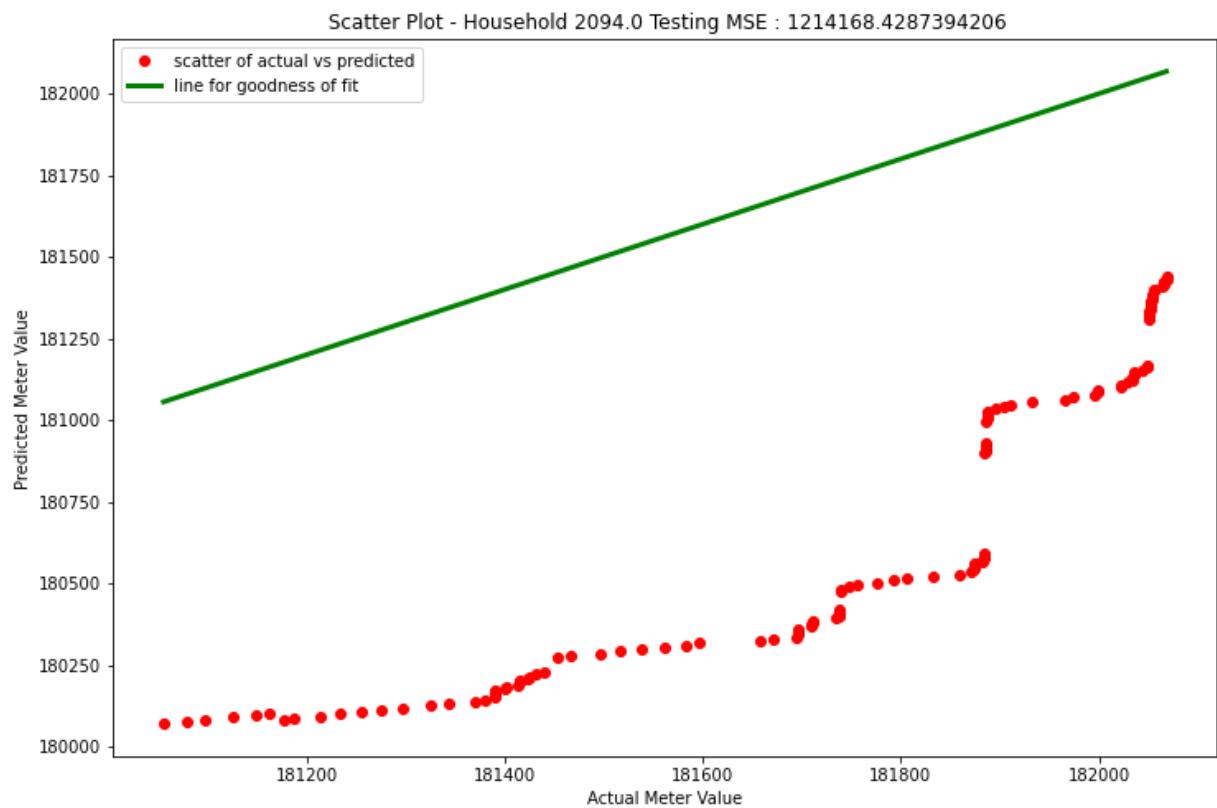


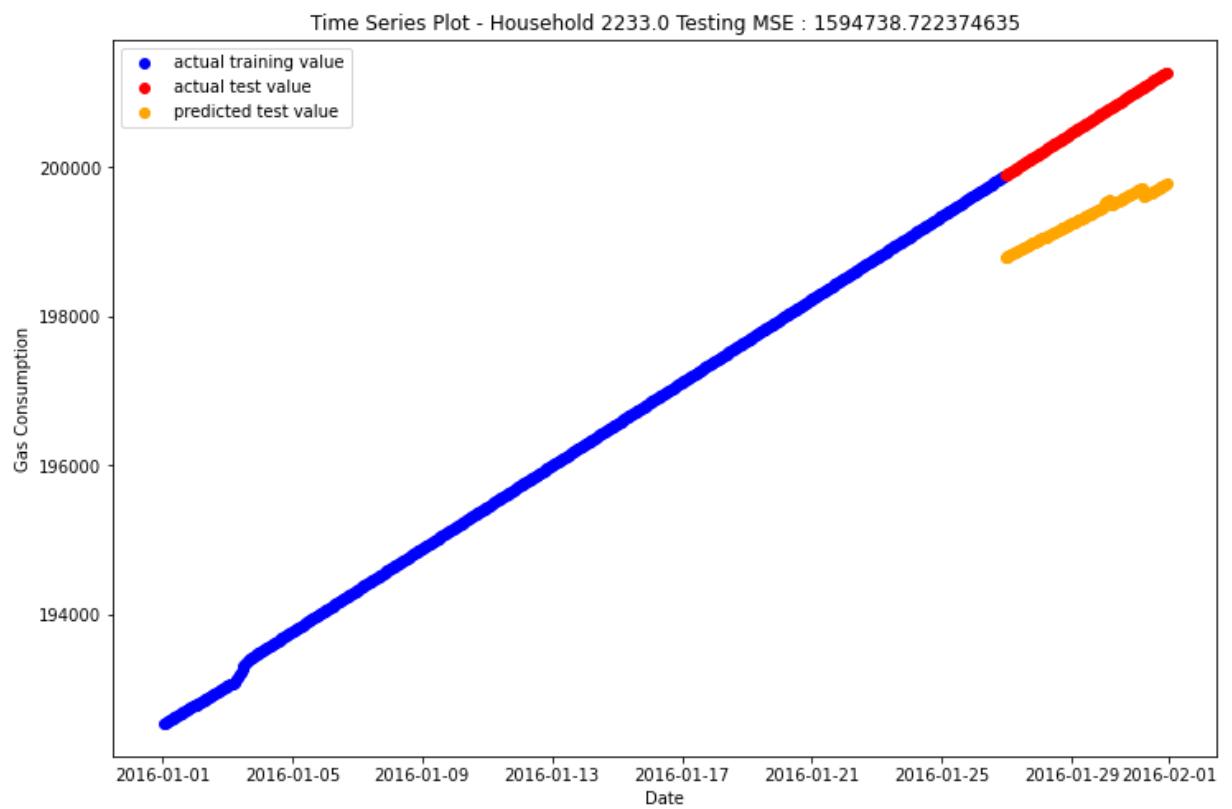
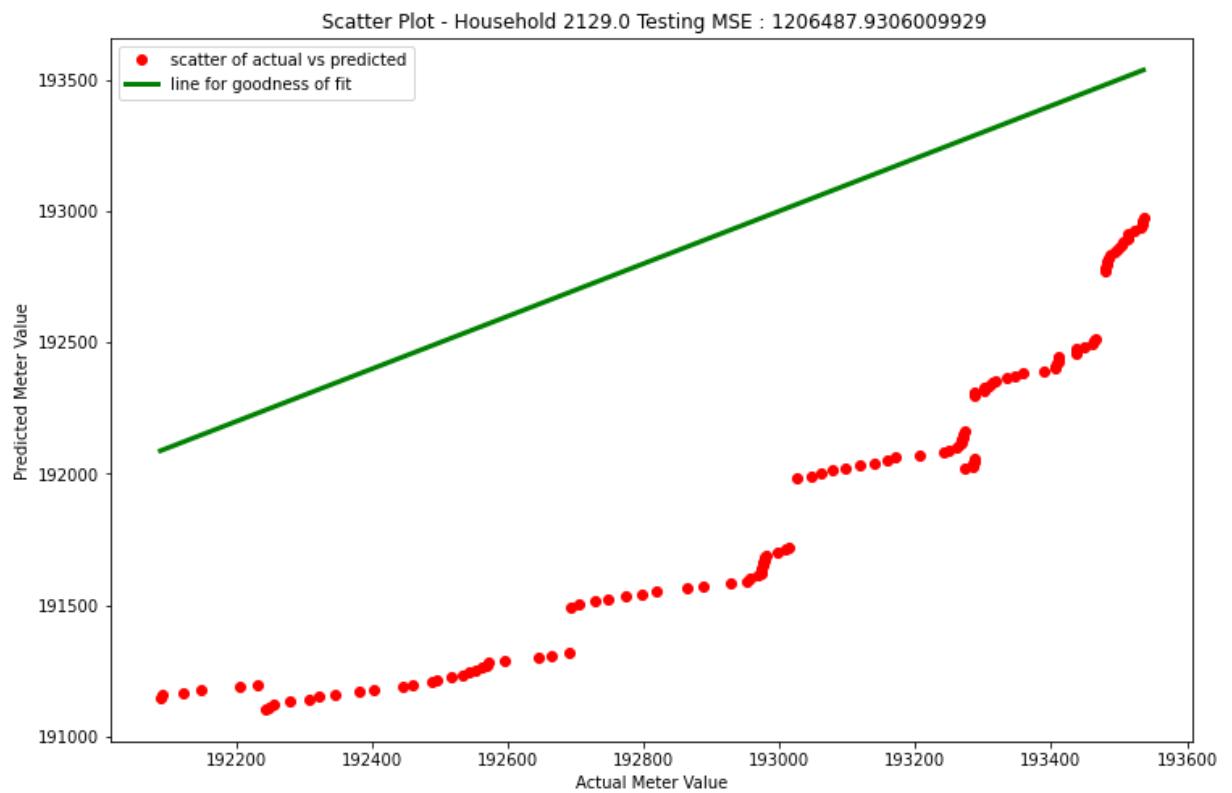


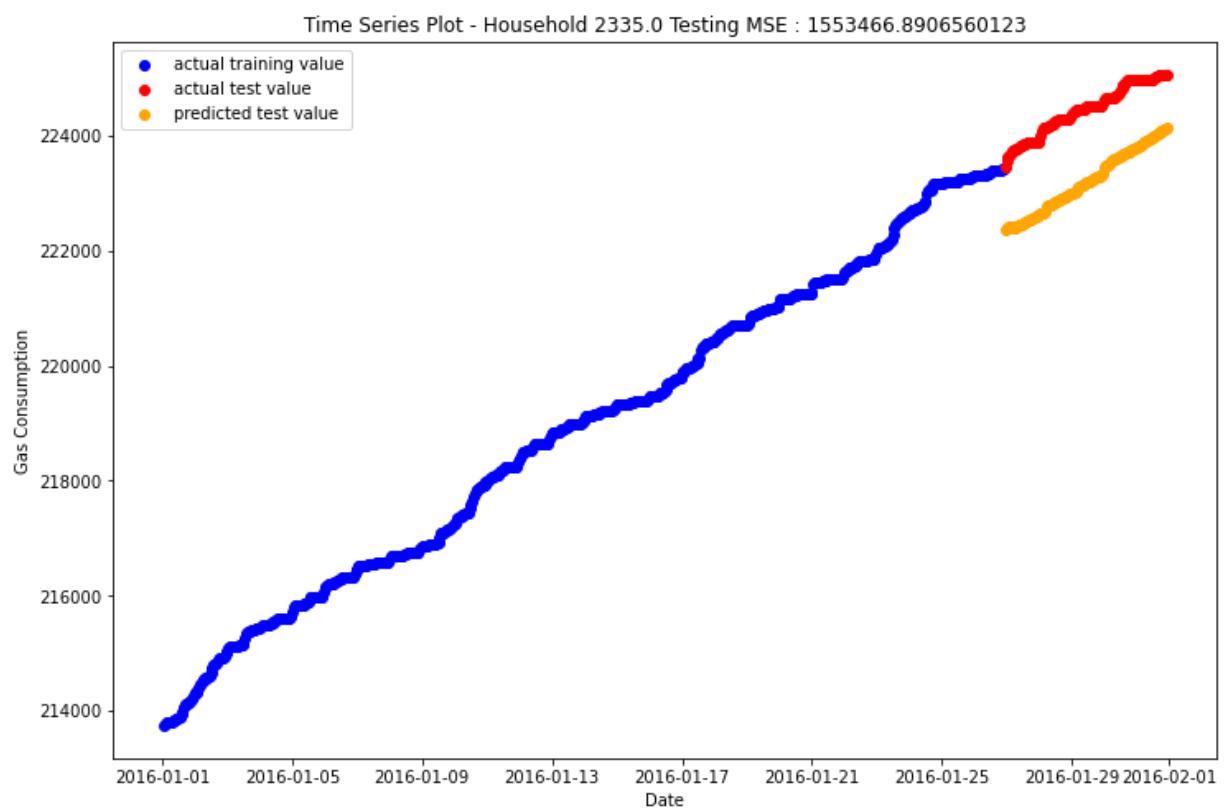


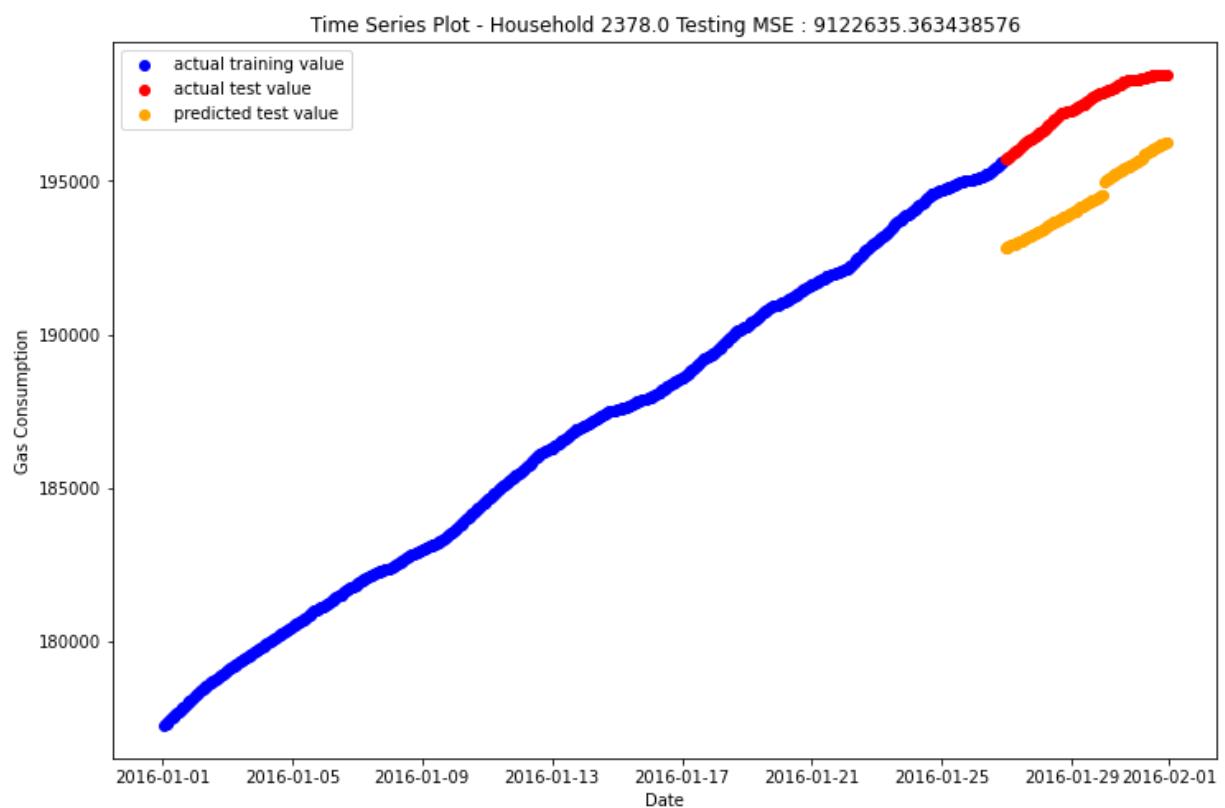
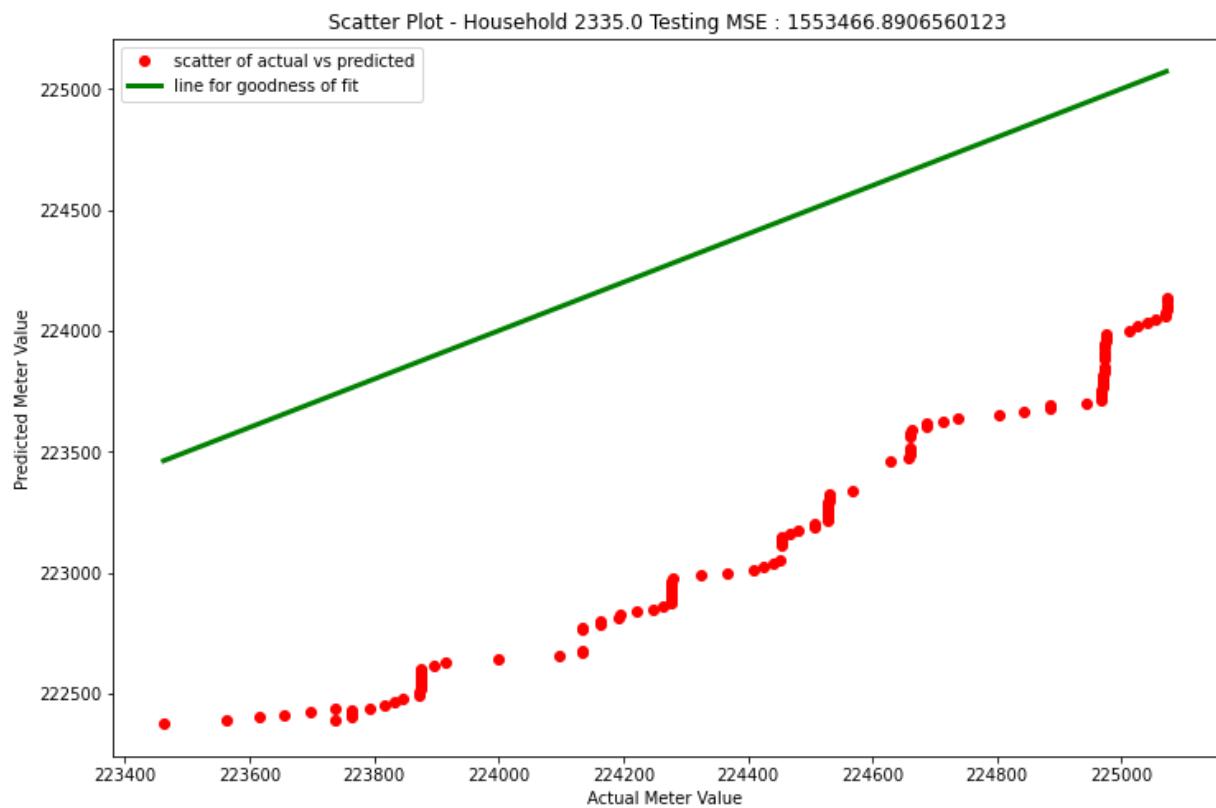


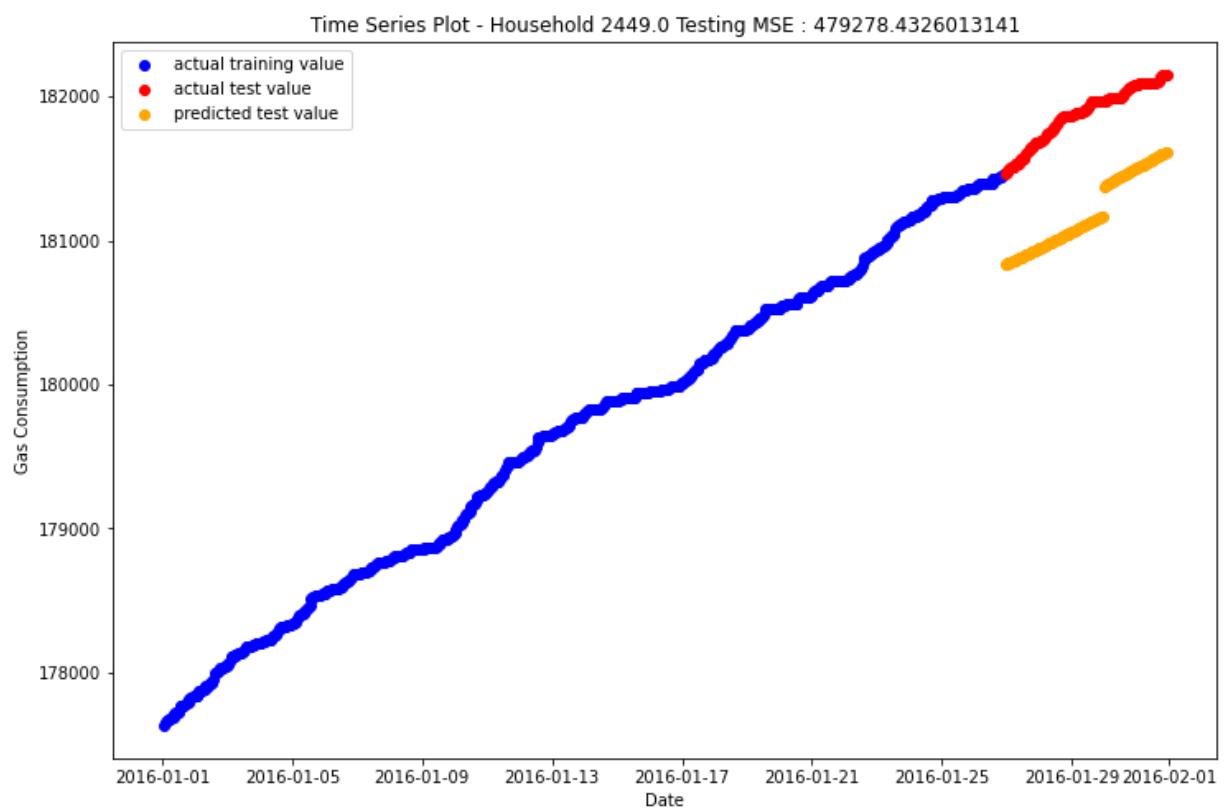
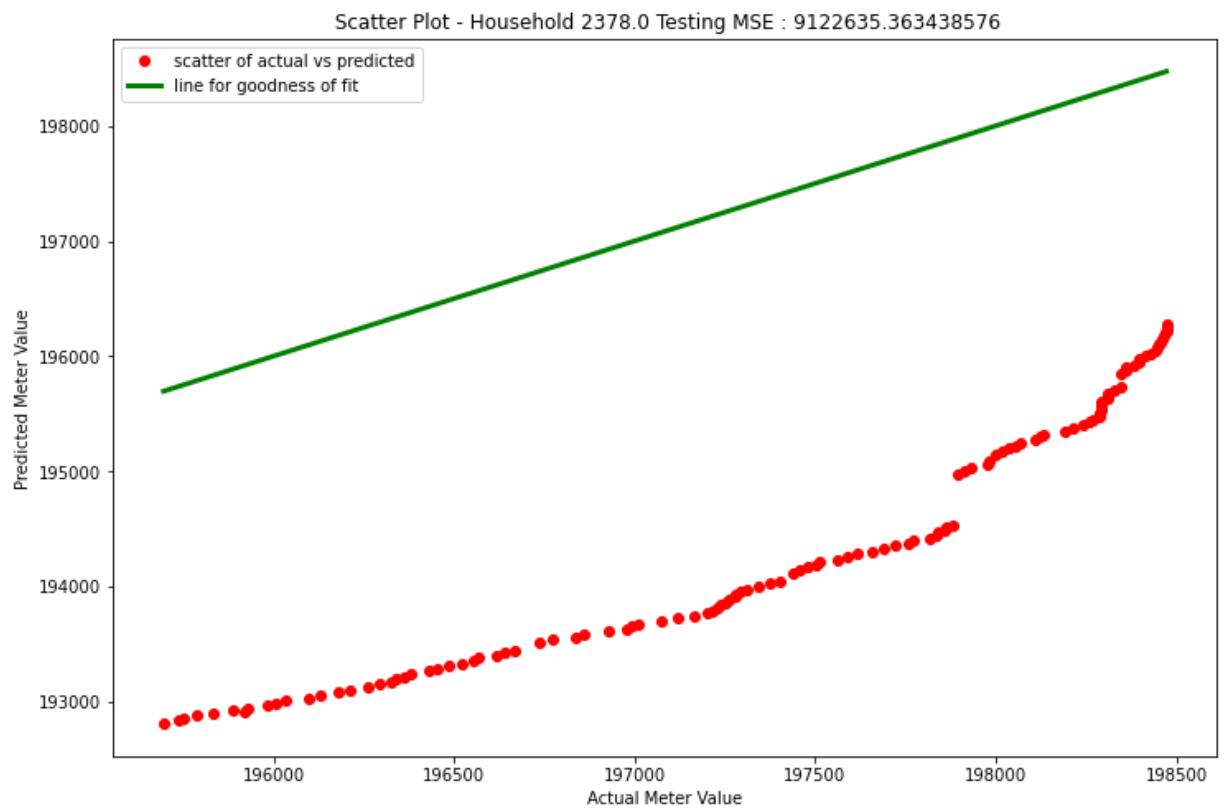


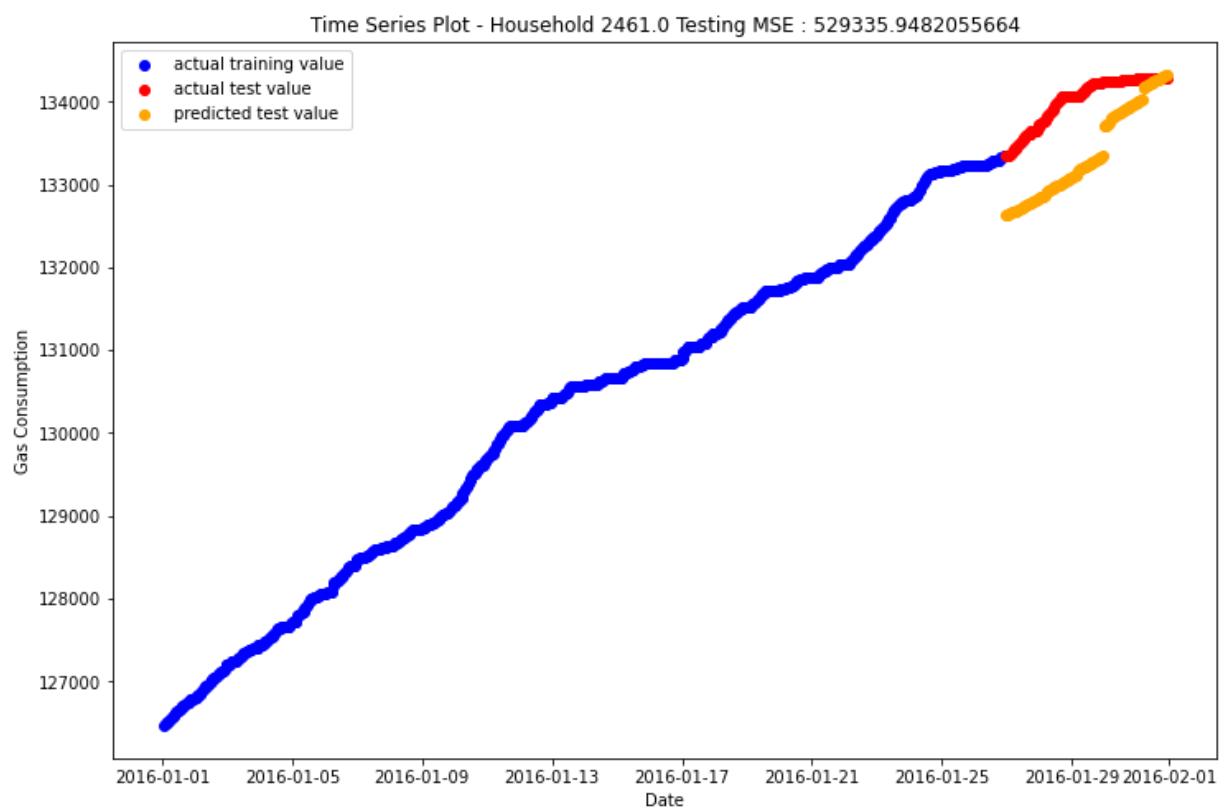
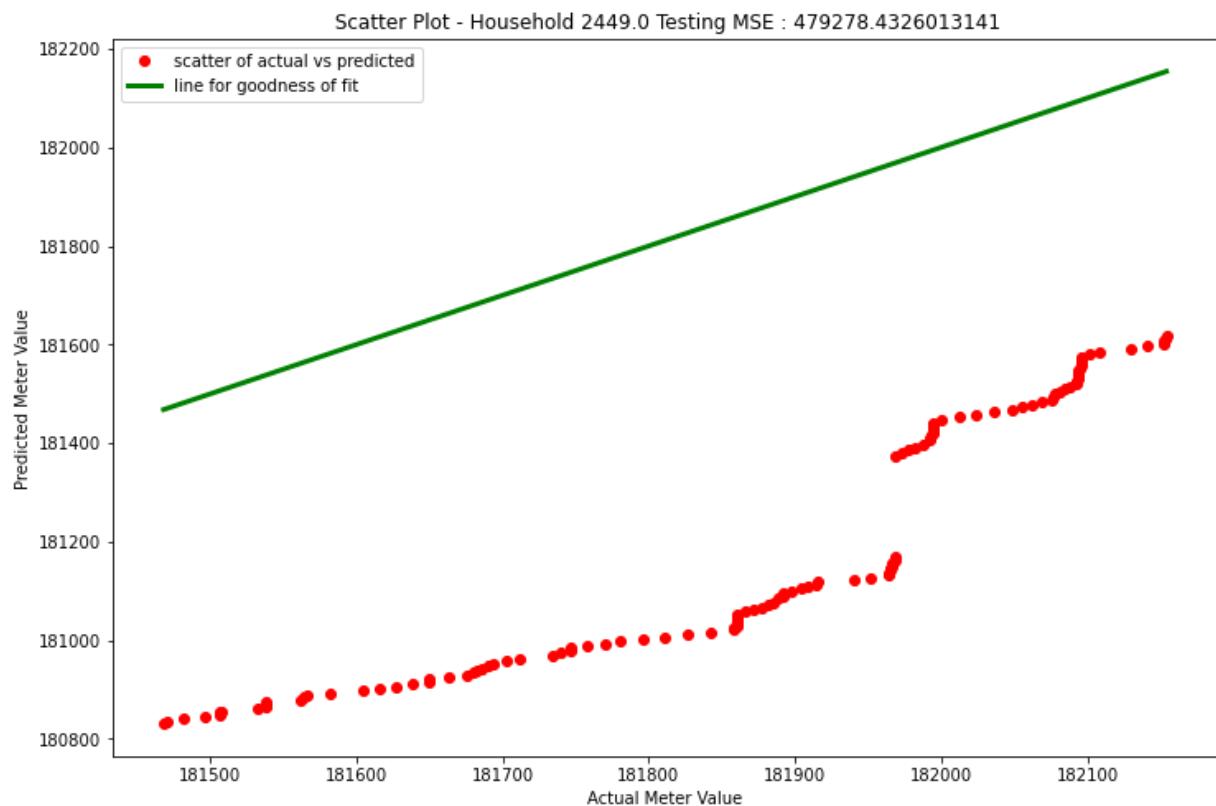


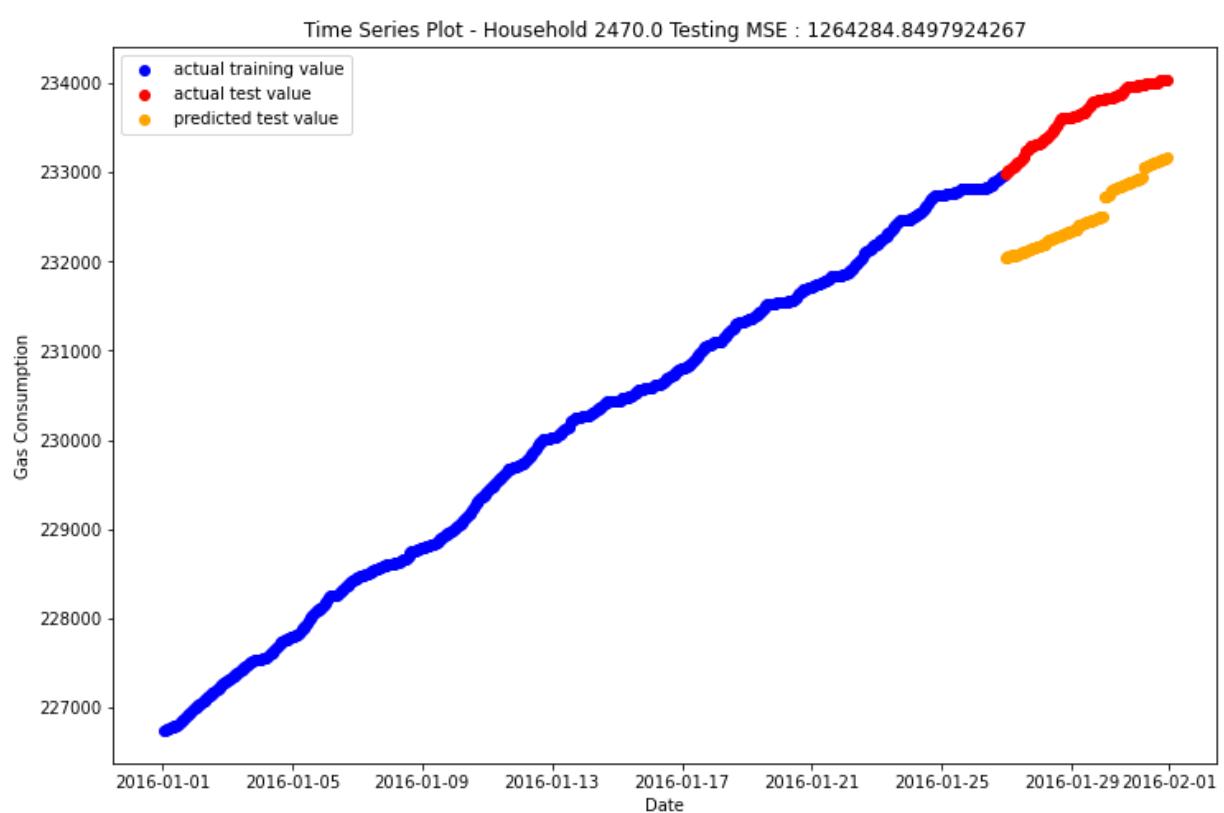
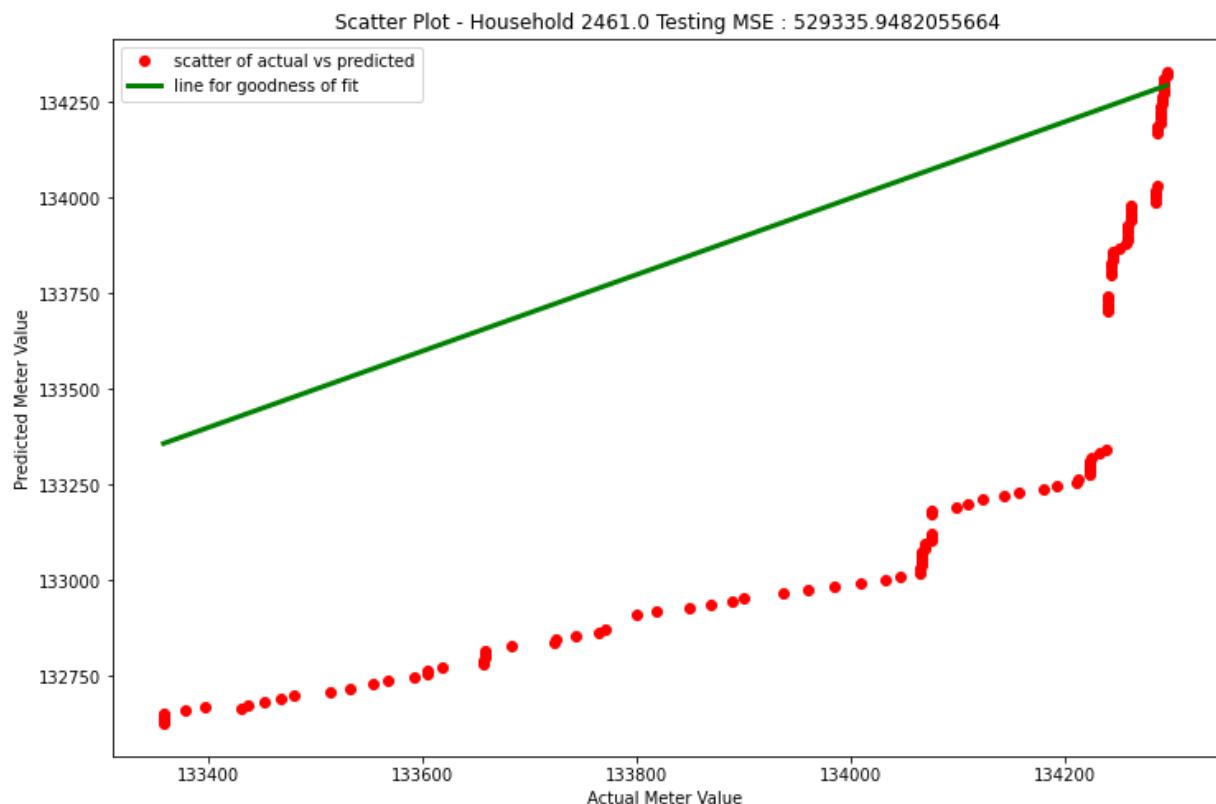


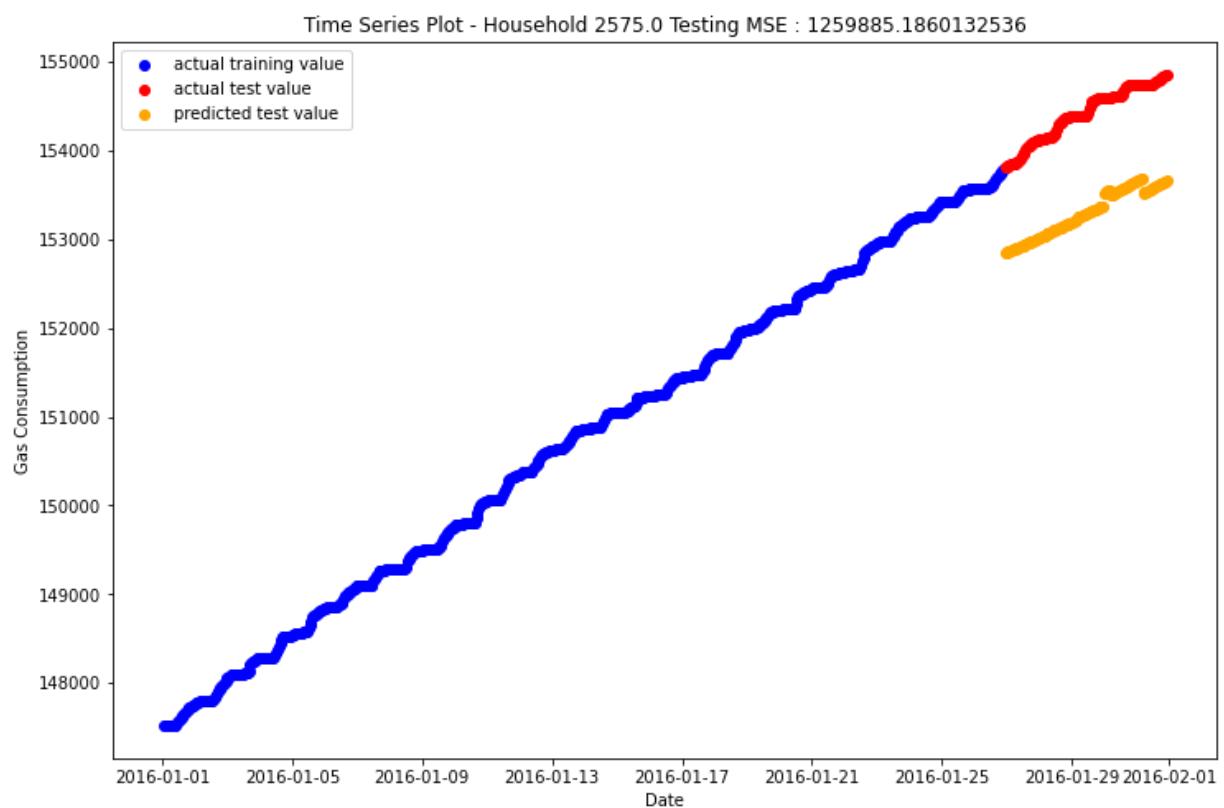
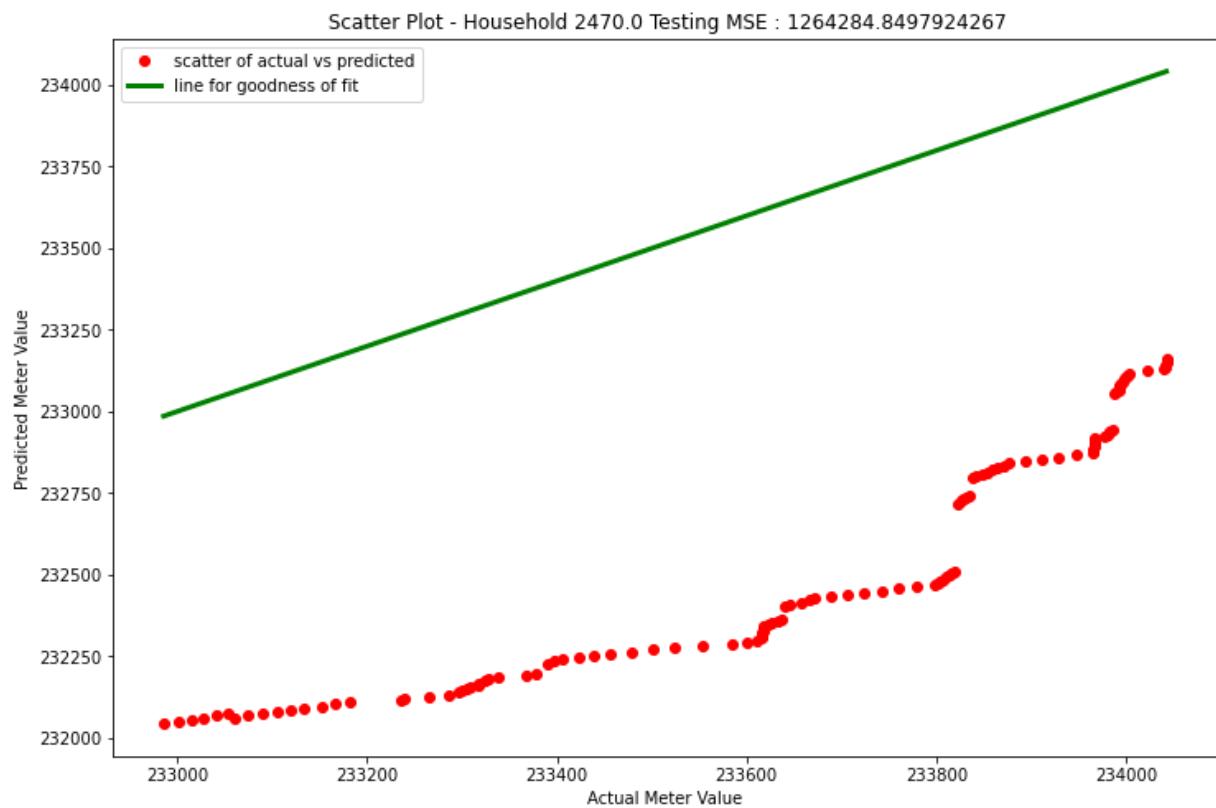


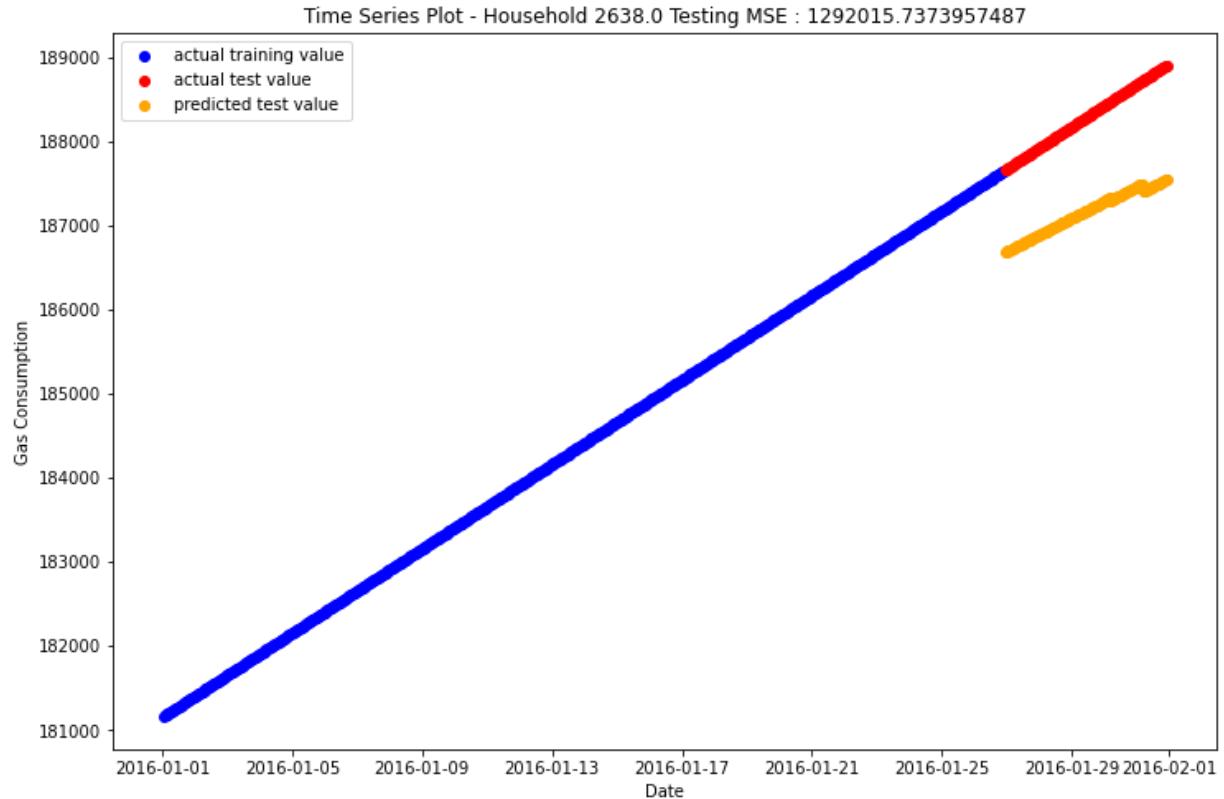
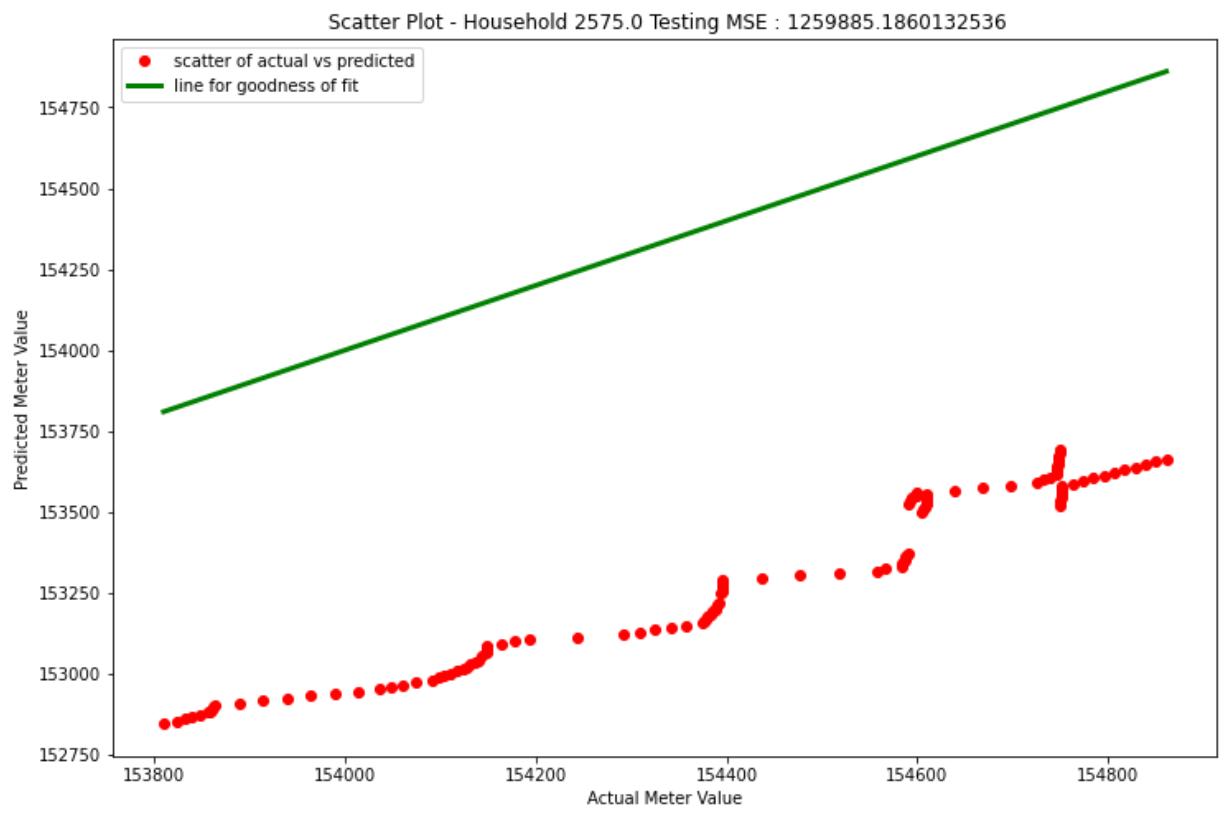


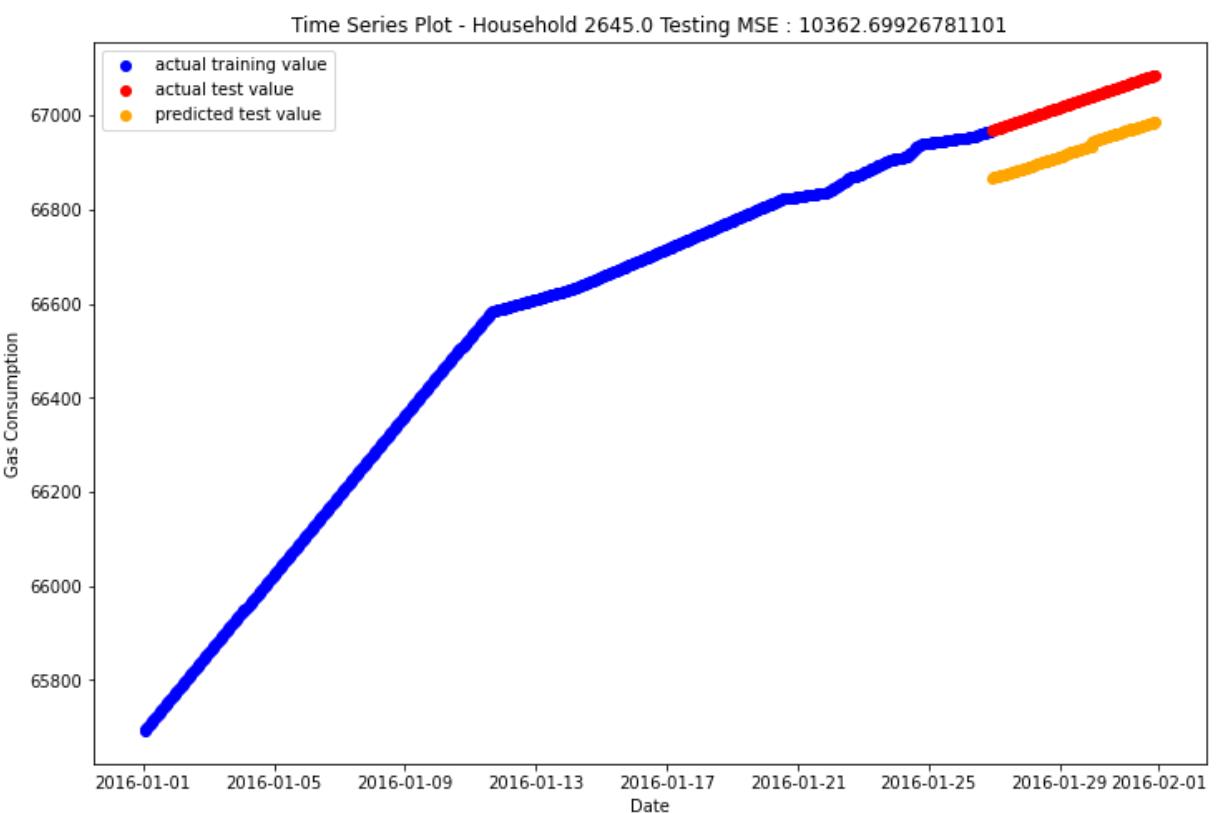
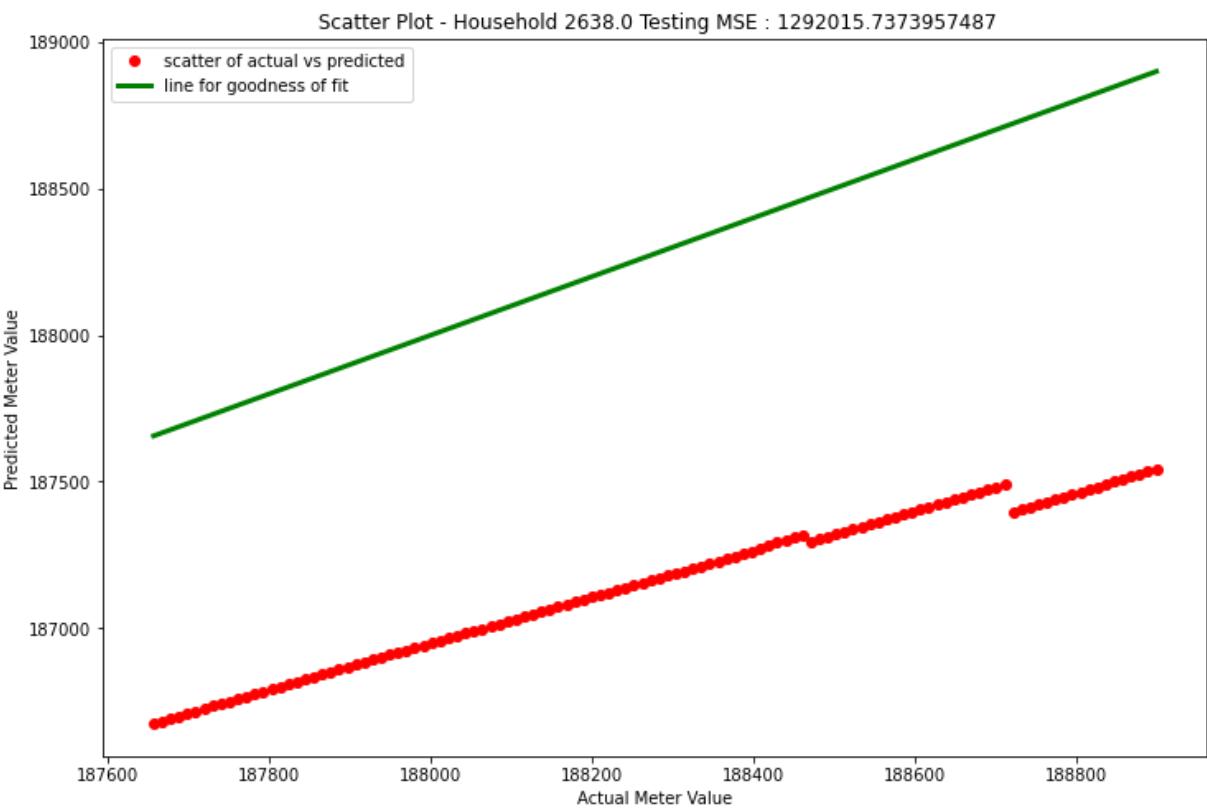




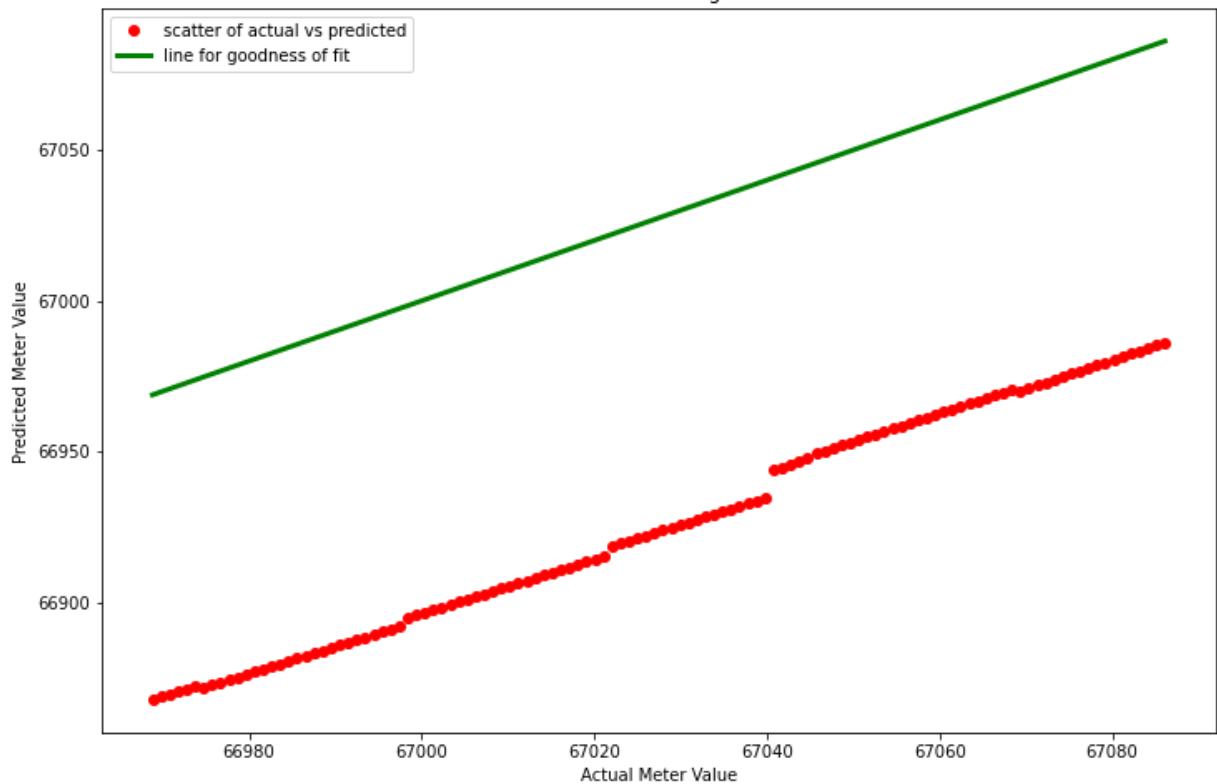




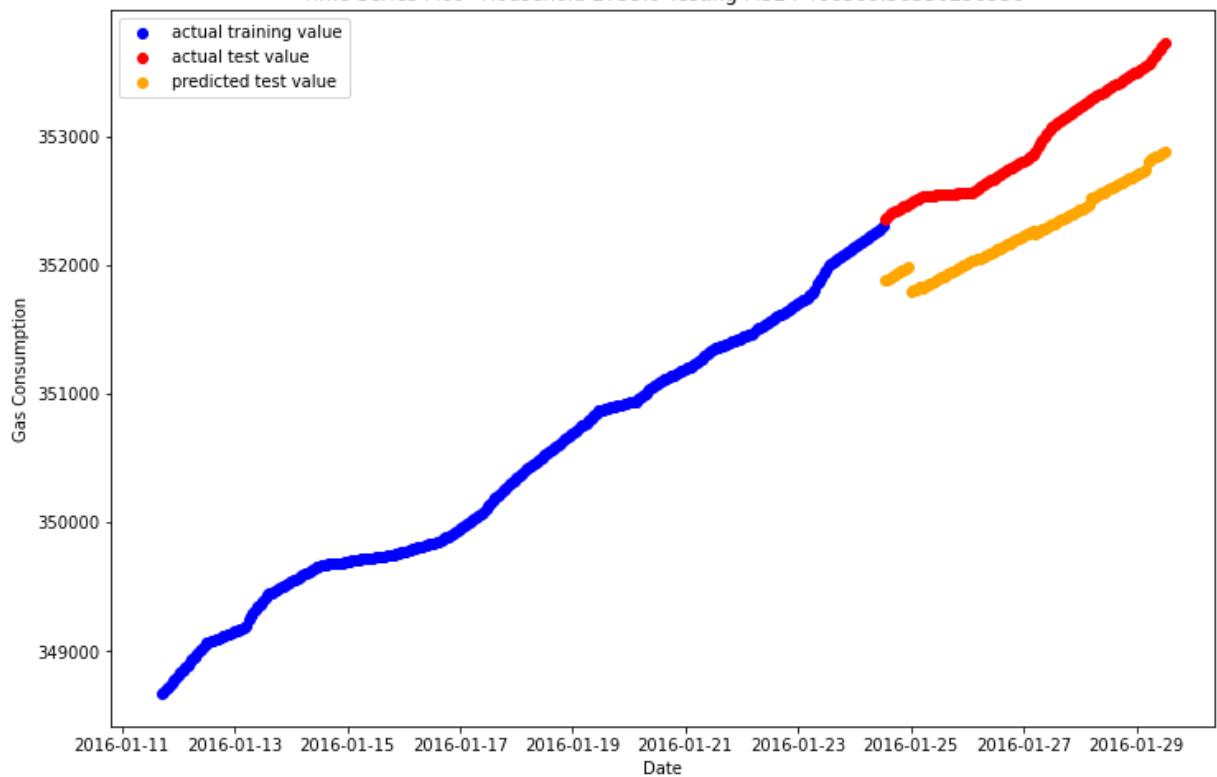


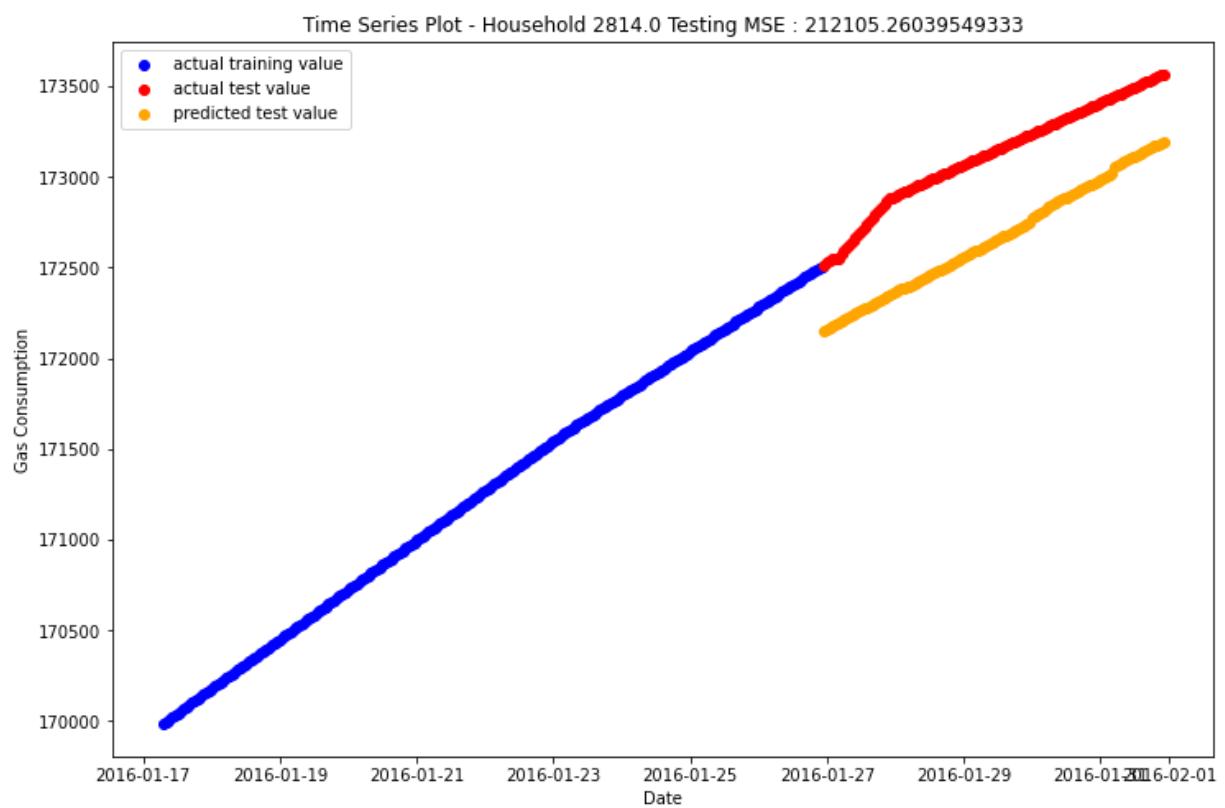
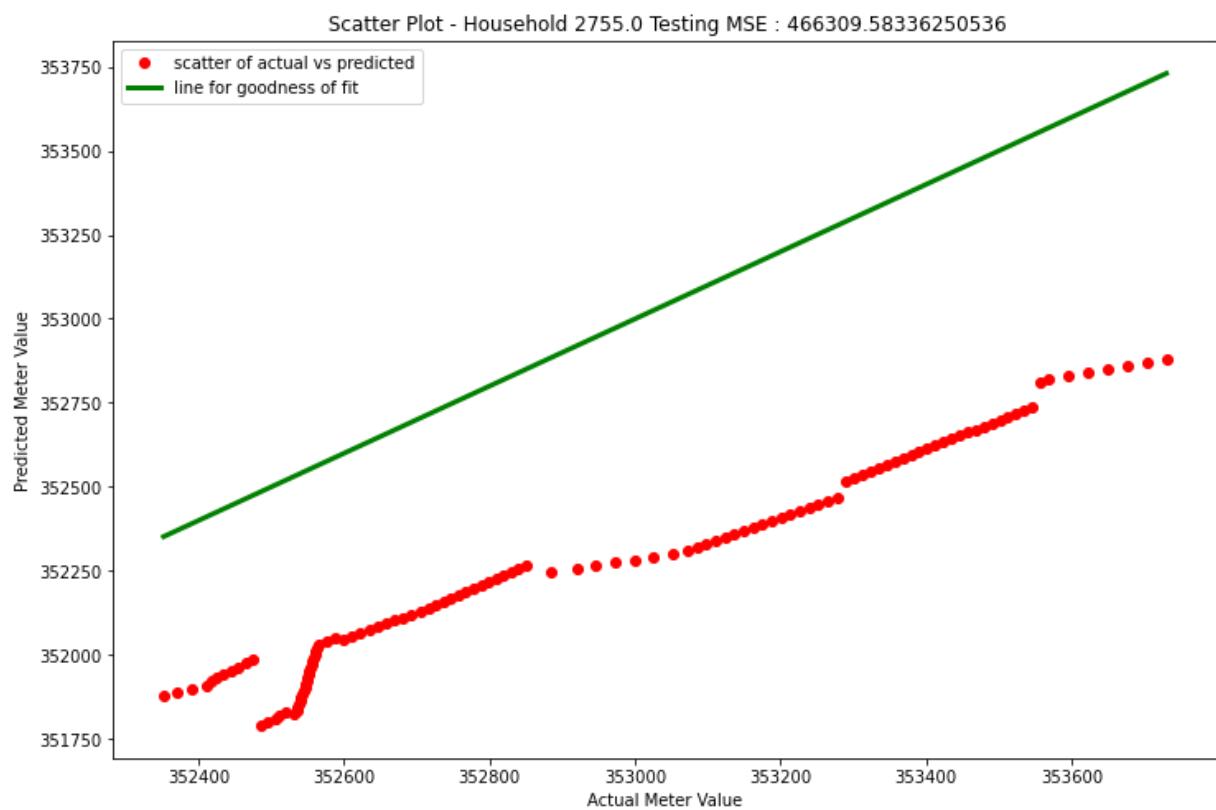


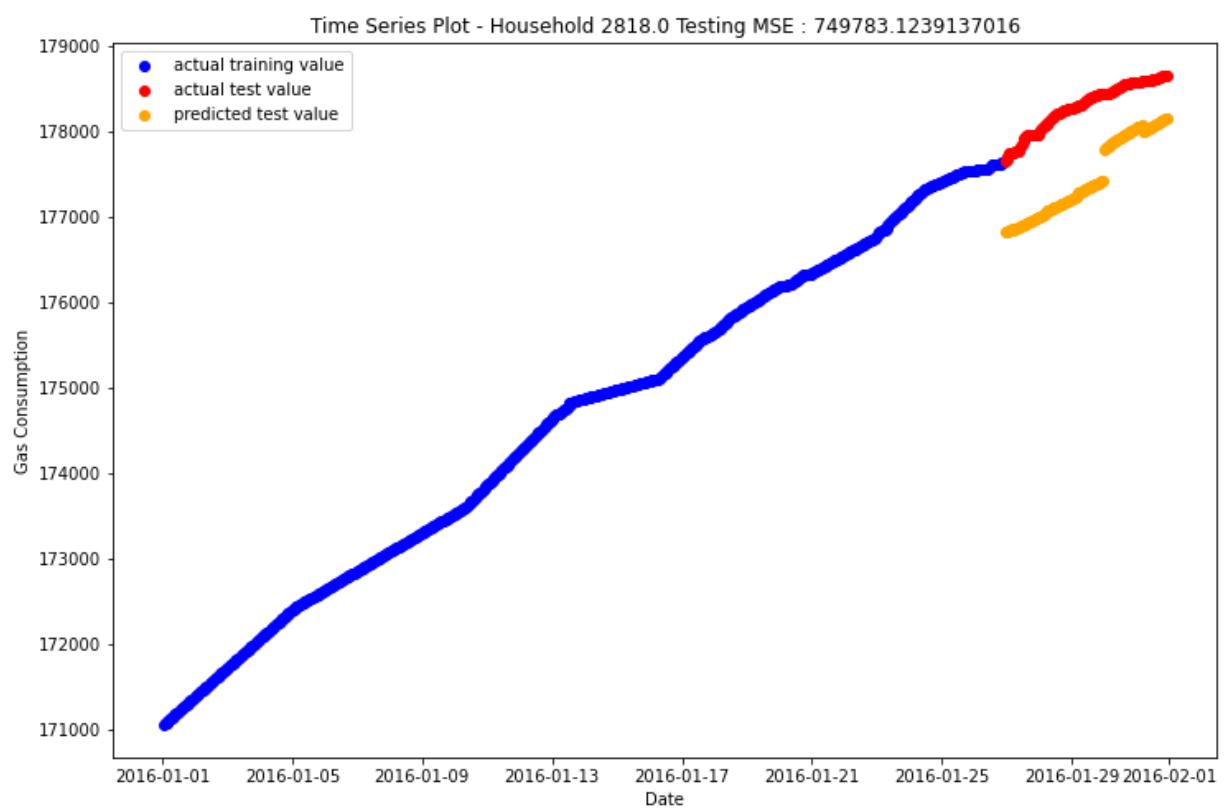
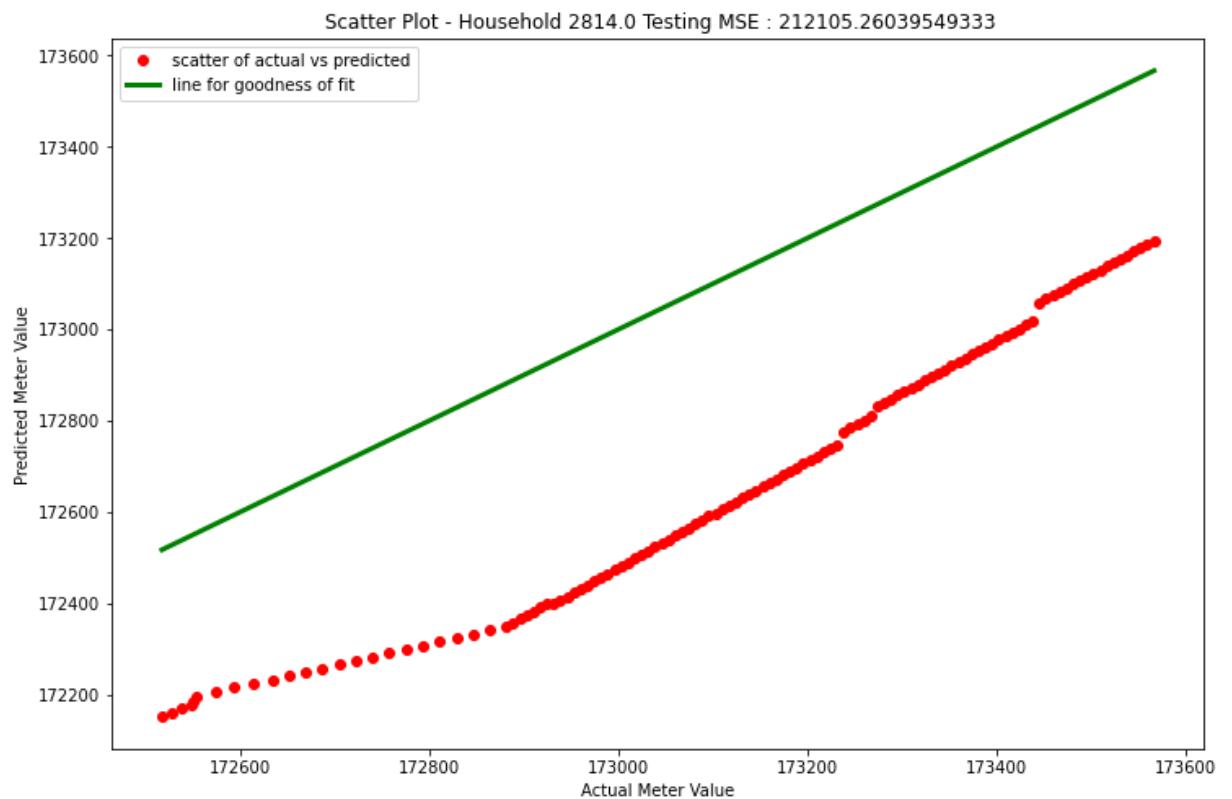
Scatter Plot - Household 2645.0 Testing MSE : 10362.69926781101



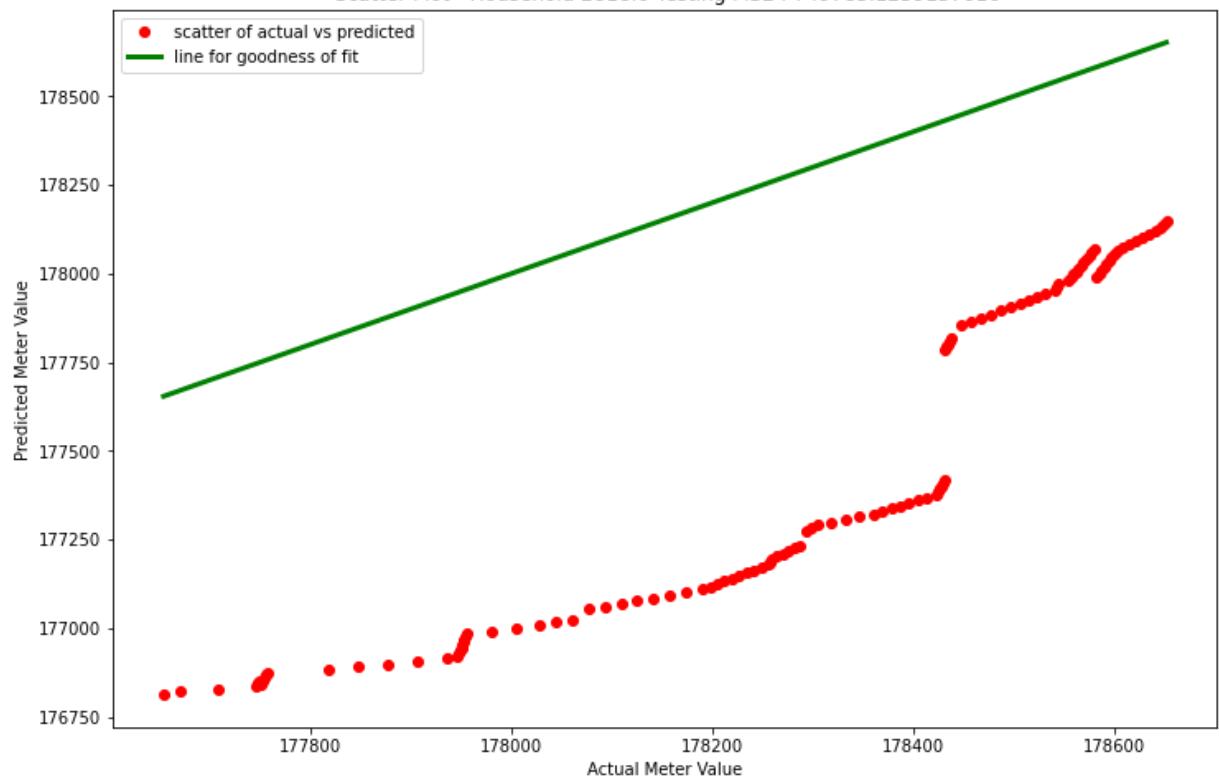
Time Series Plot - Household 2755.0 Testing MSE : 466309.58336250536



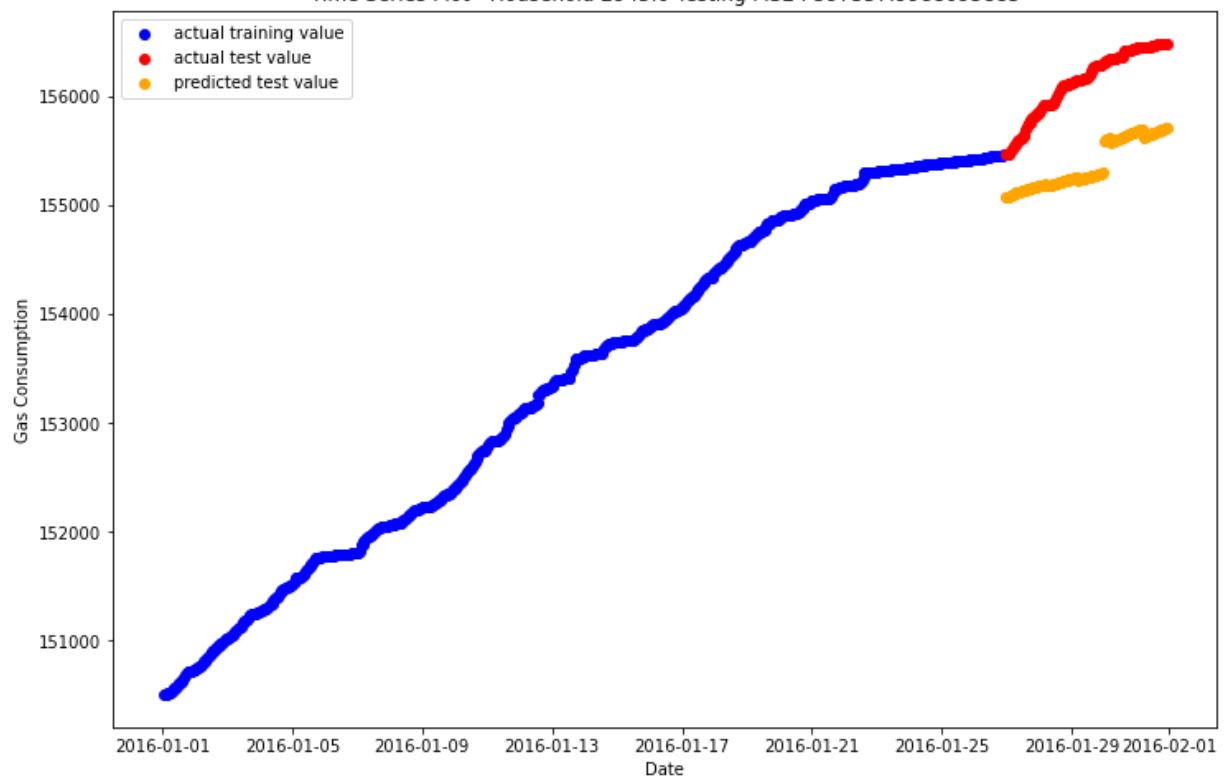


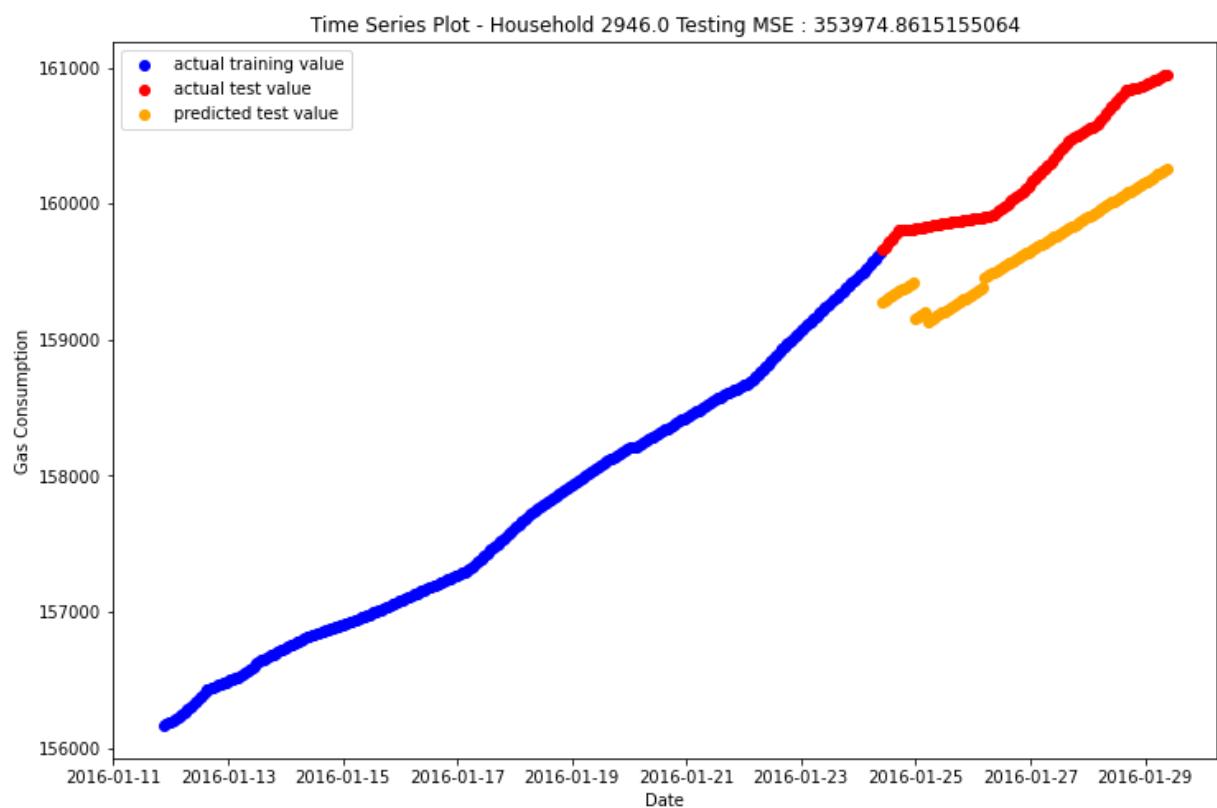
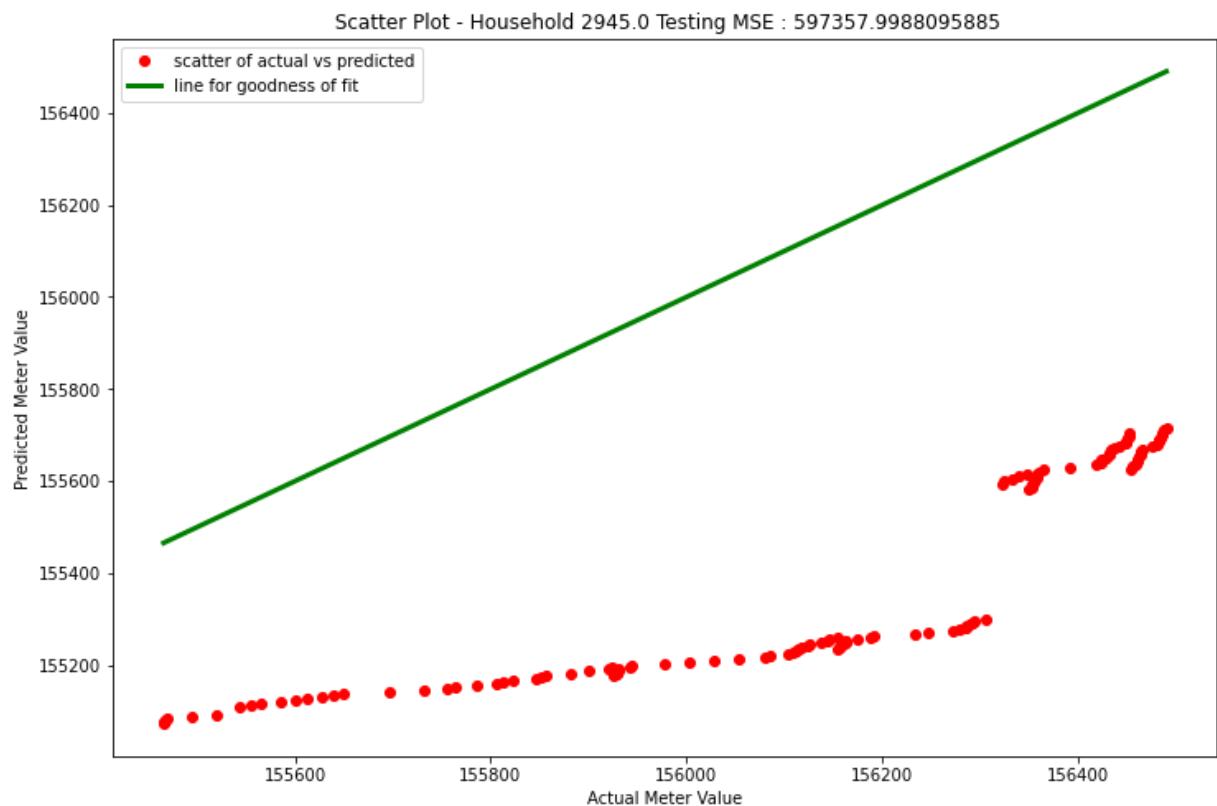


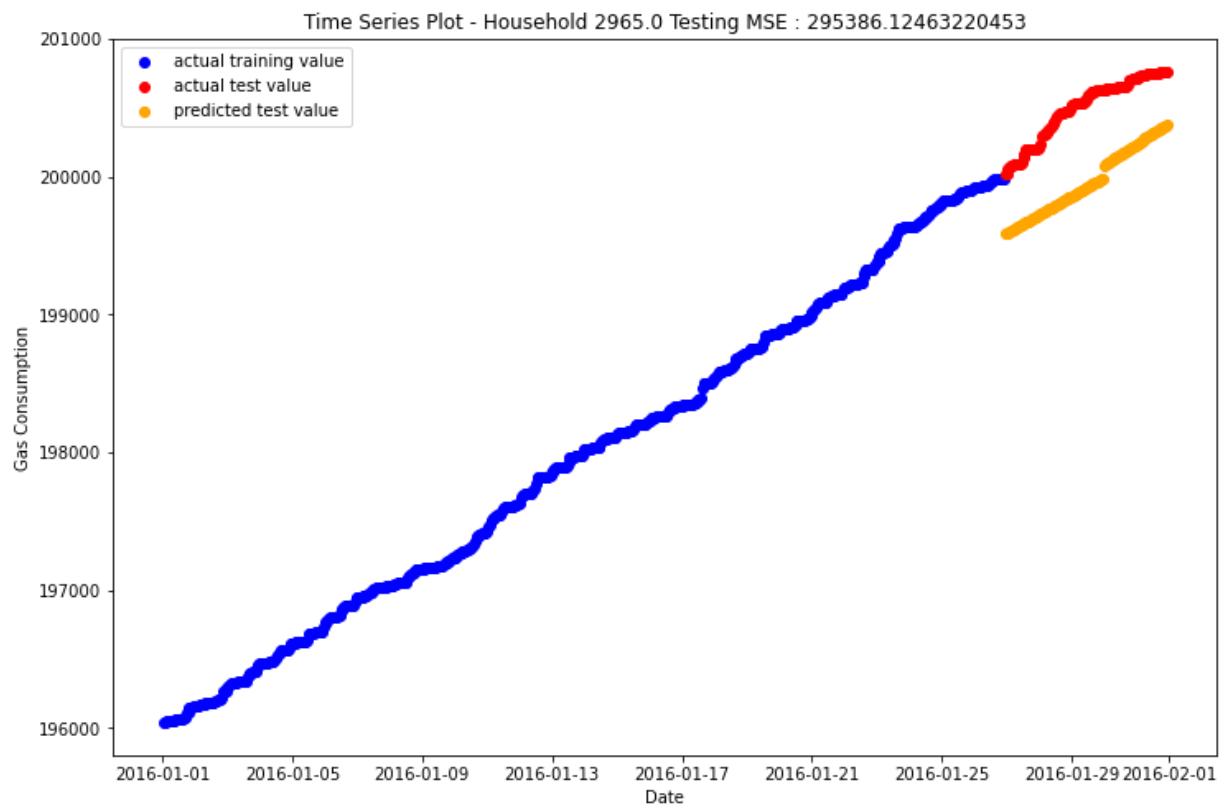
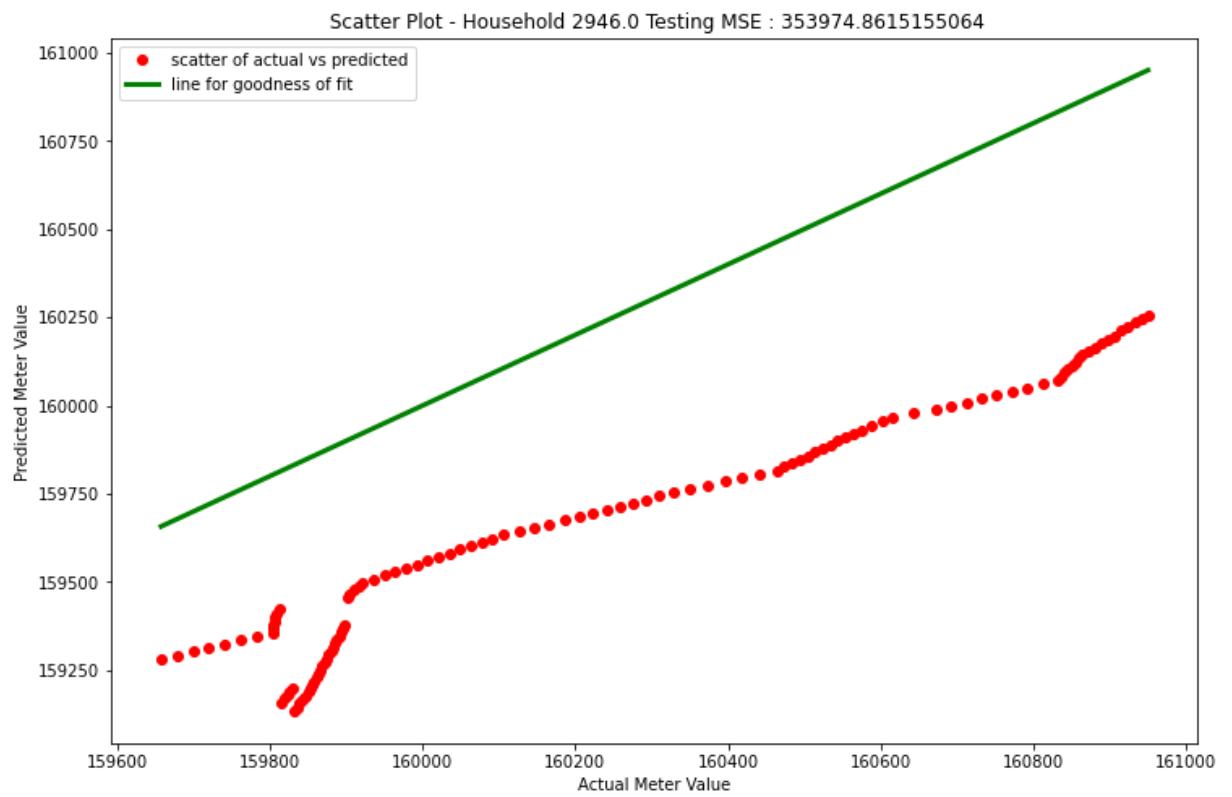
Scatter Plot - Household 2818.0 Testing MSE : 749783.1239137016

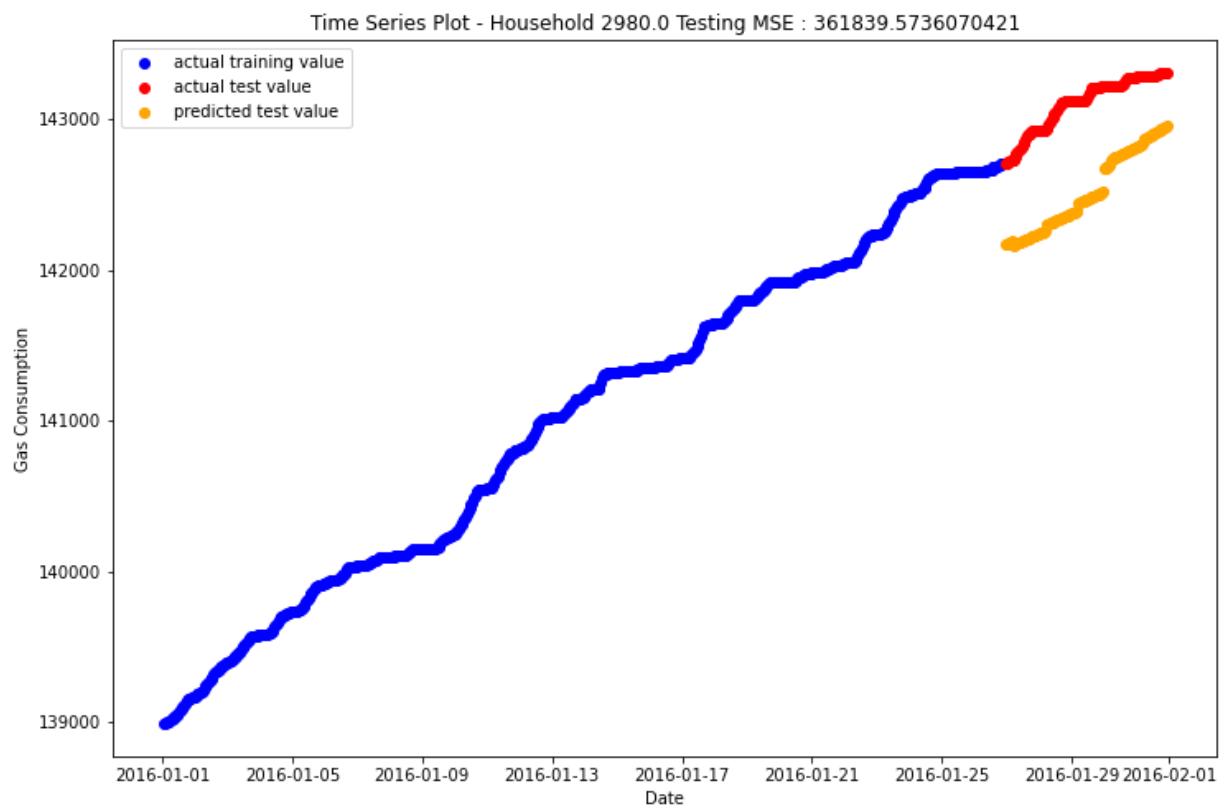
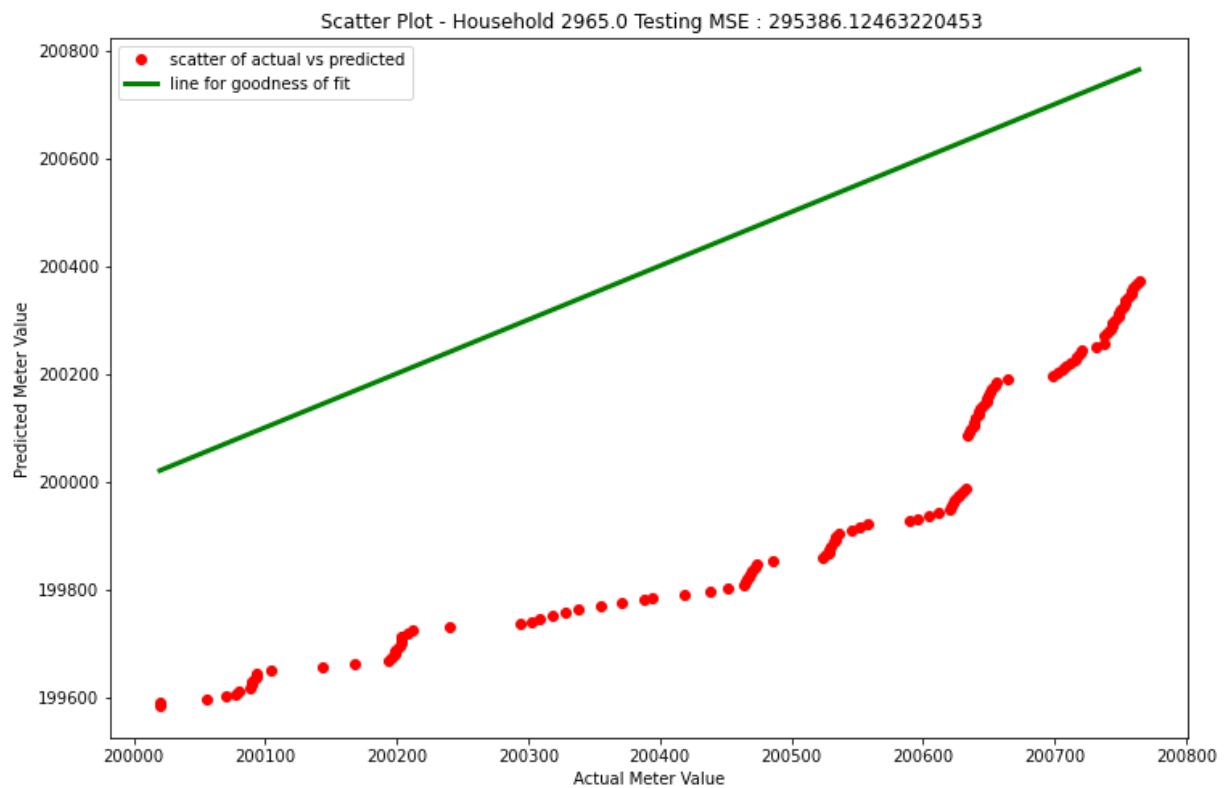


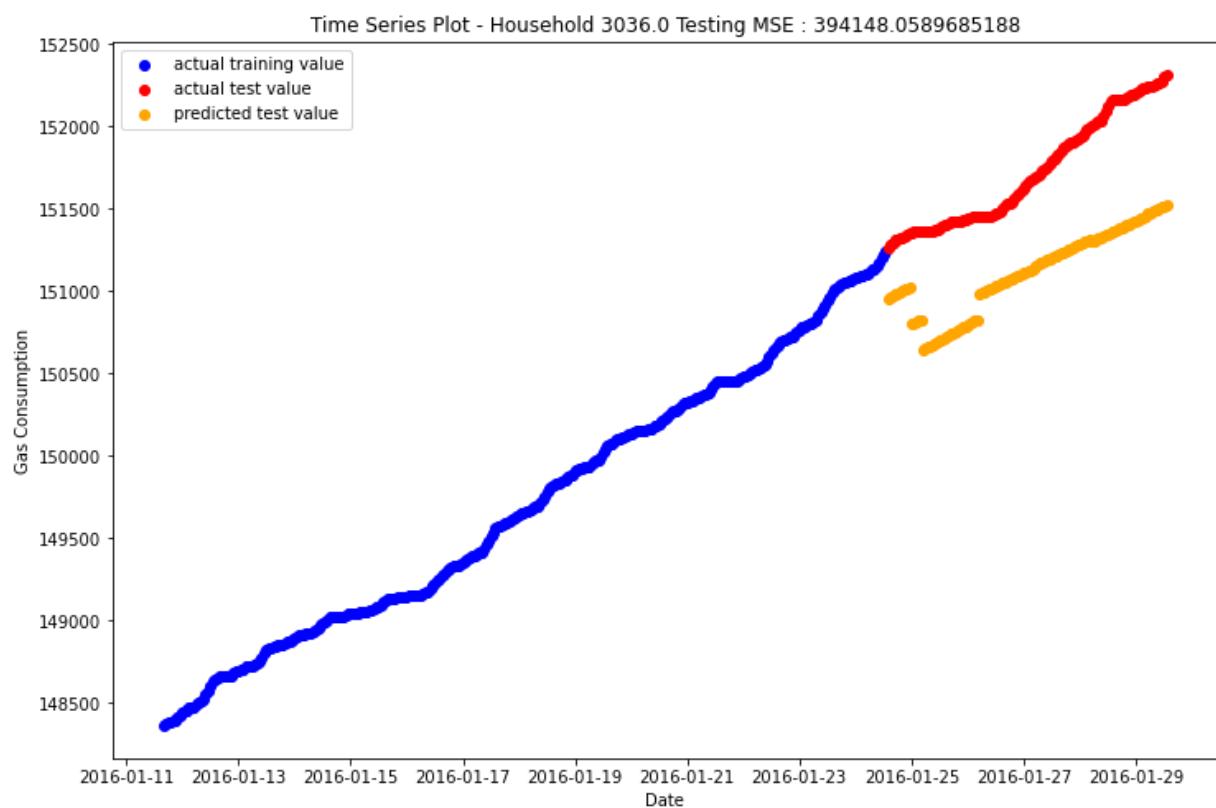
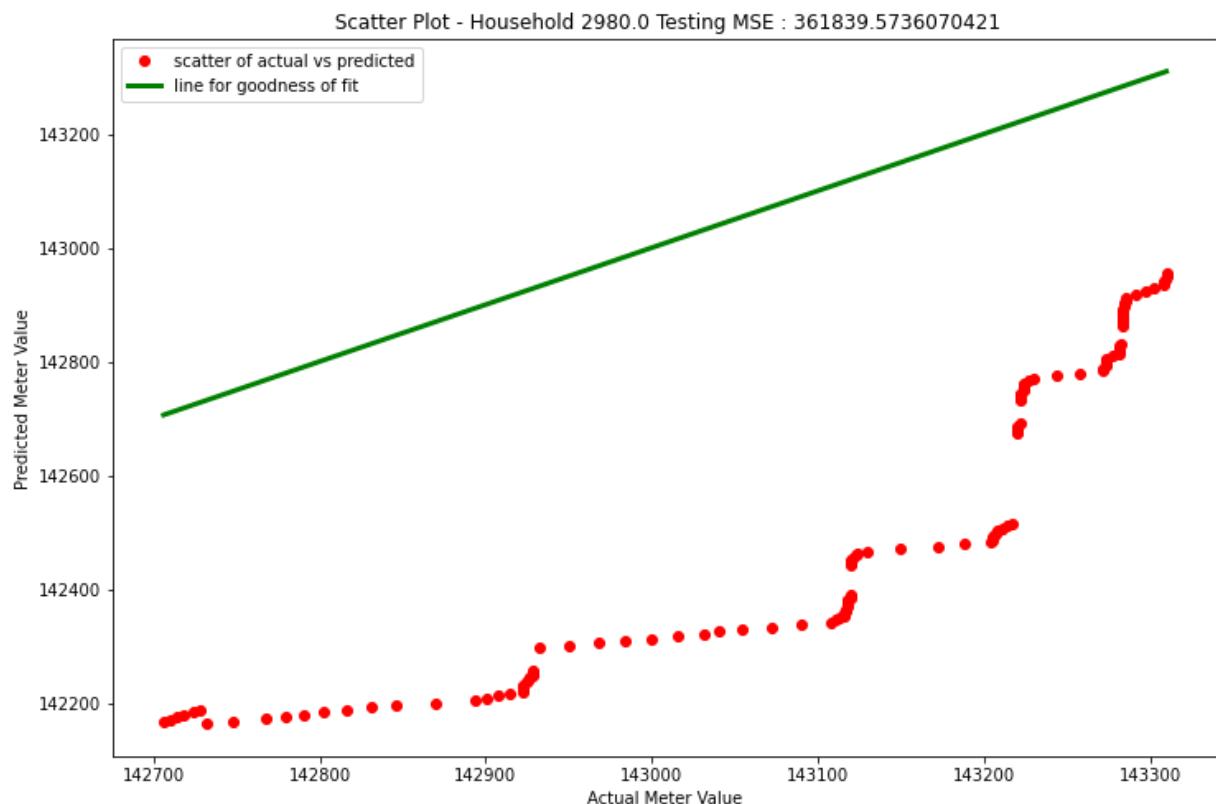
Time Series Plot - Household 2945.0 Testing MSE : 597357.9988095885

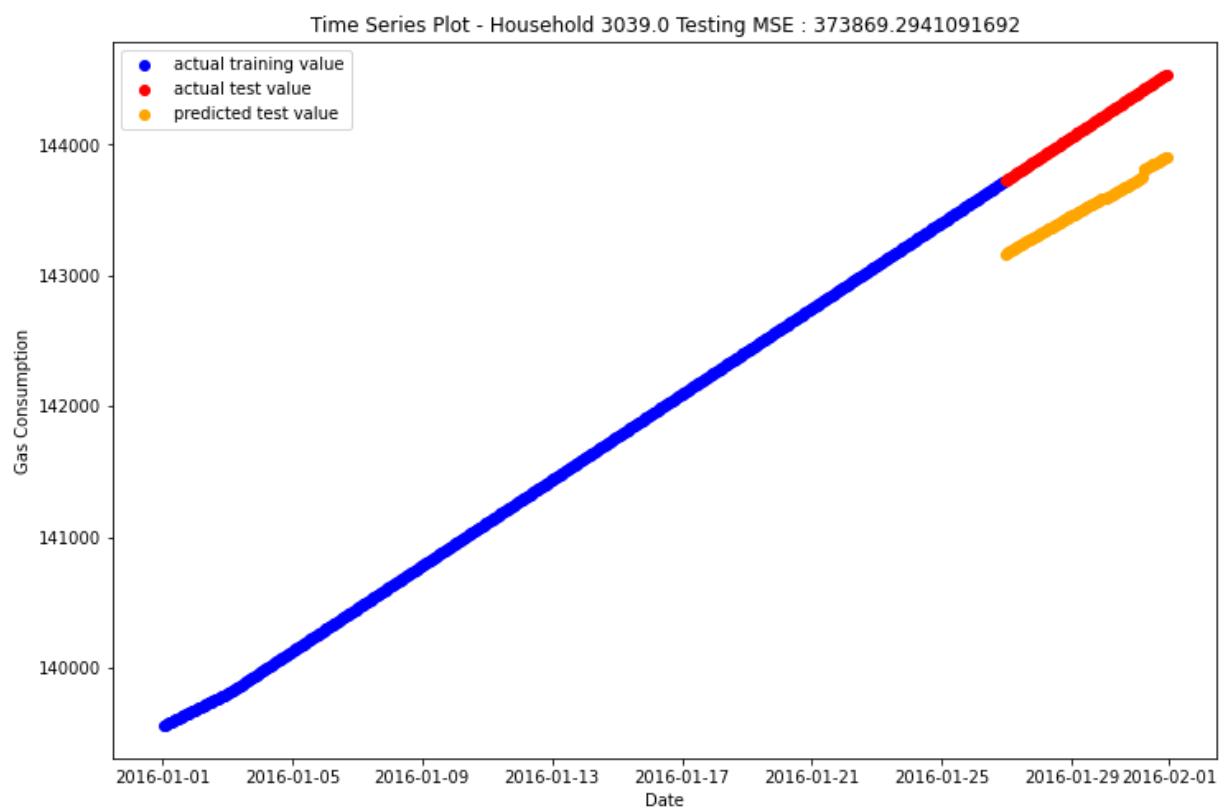
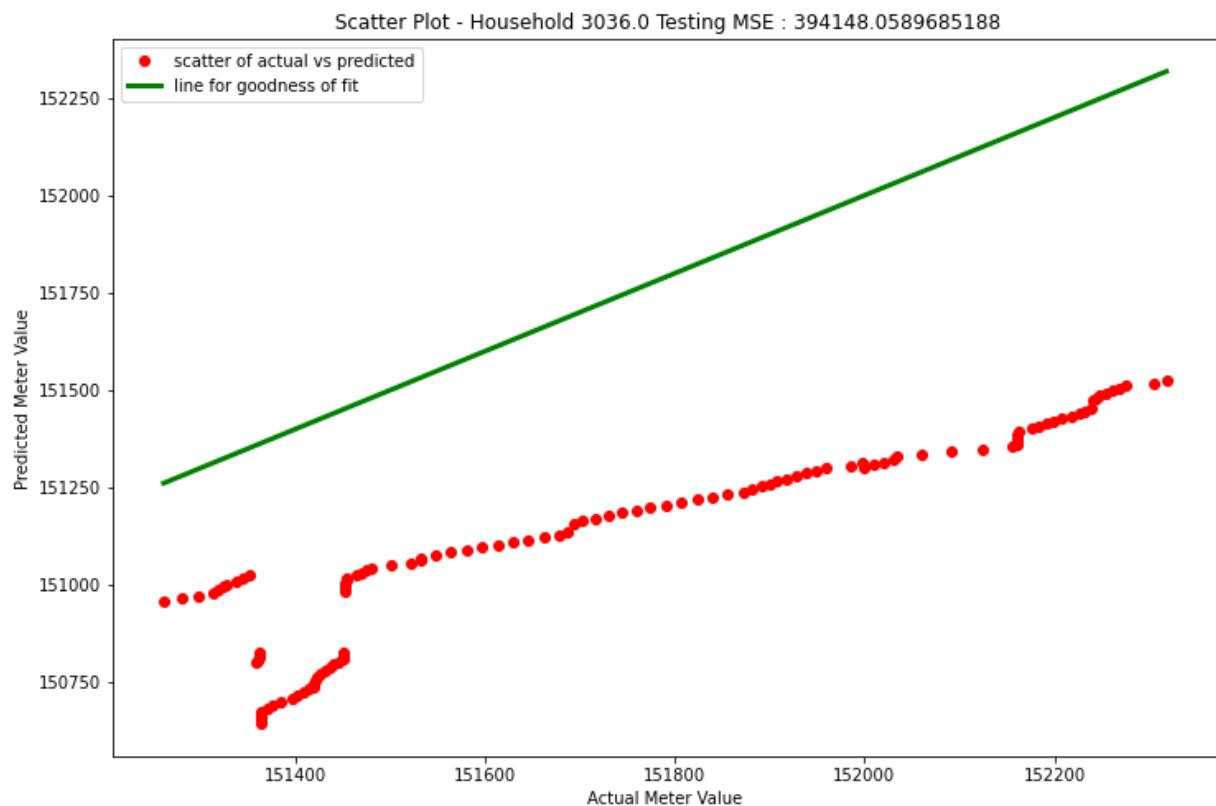


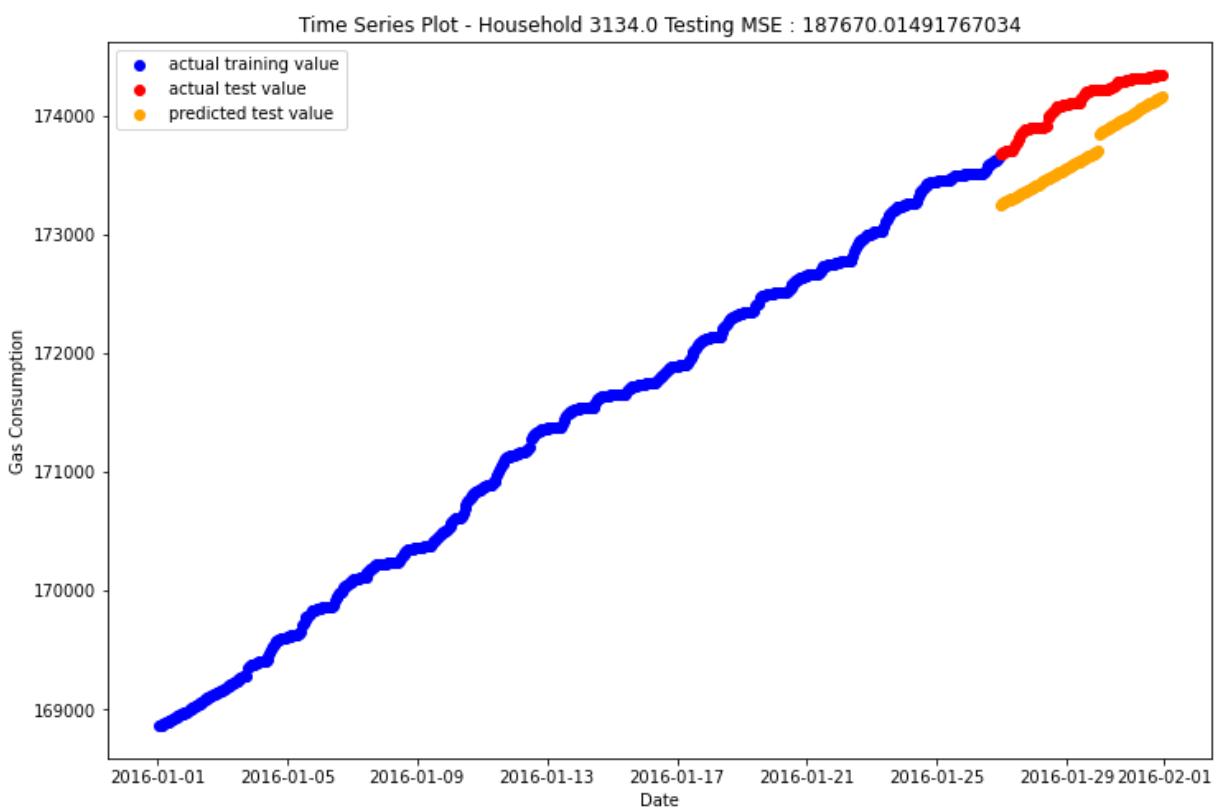
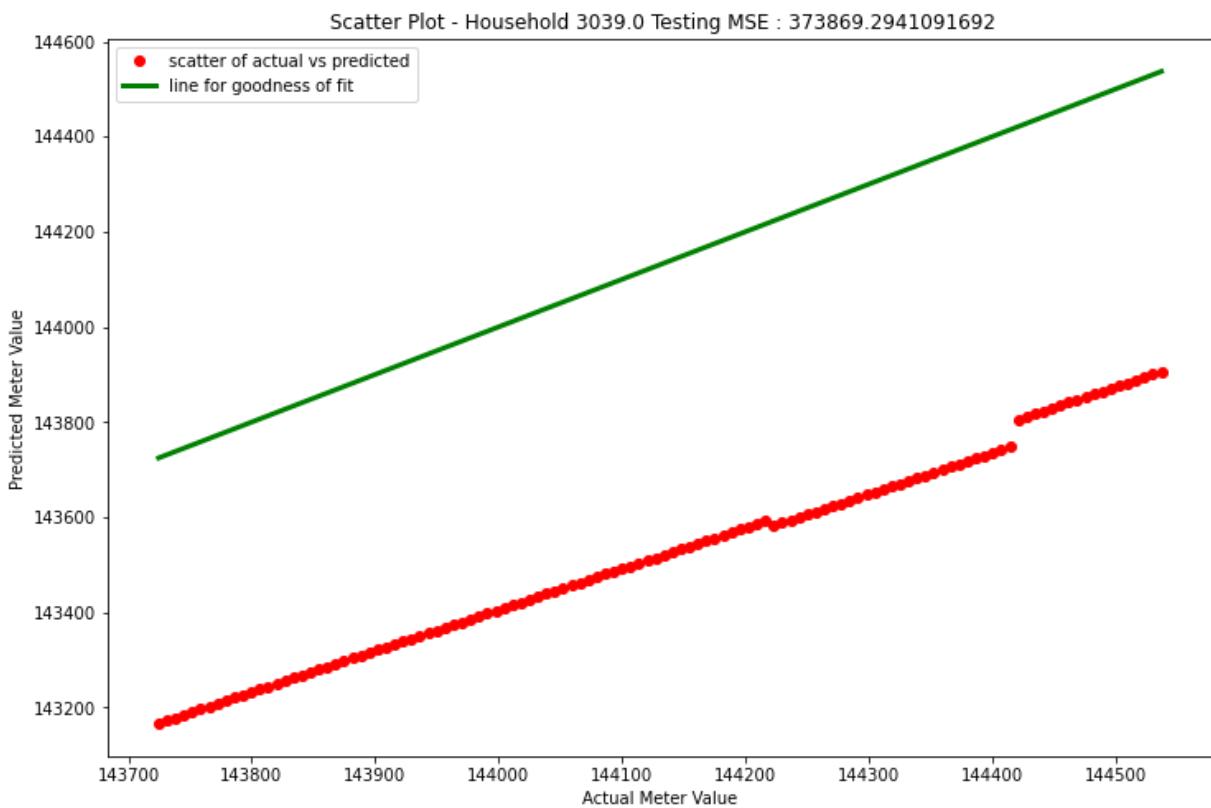


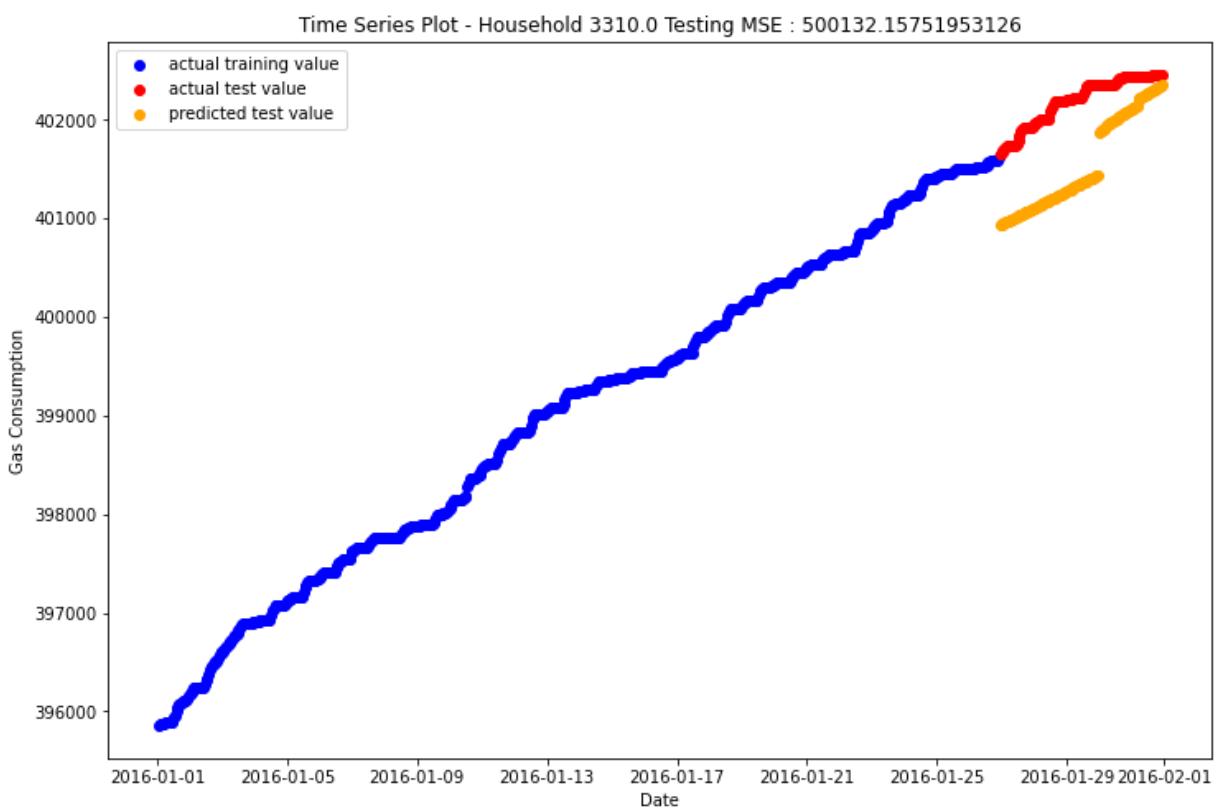
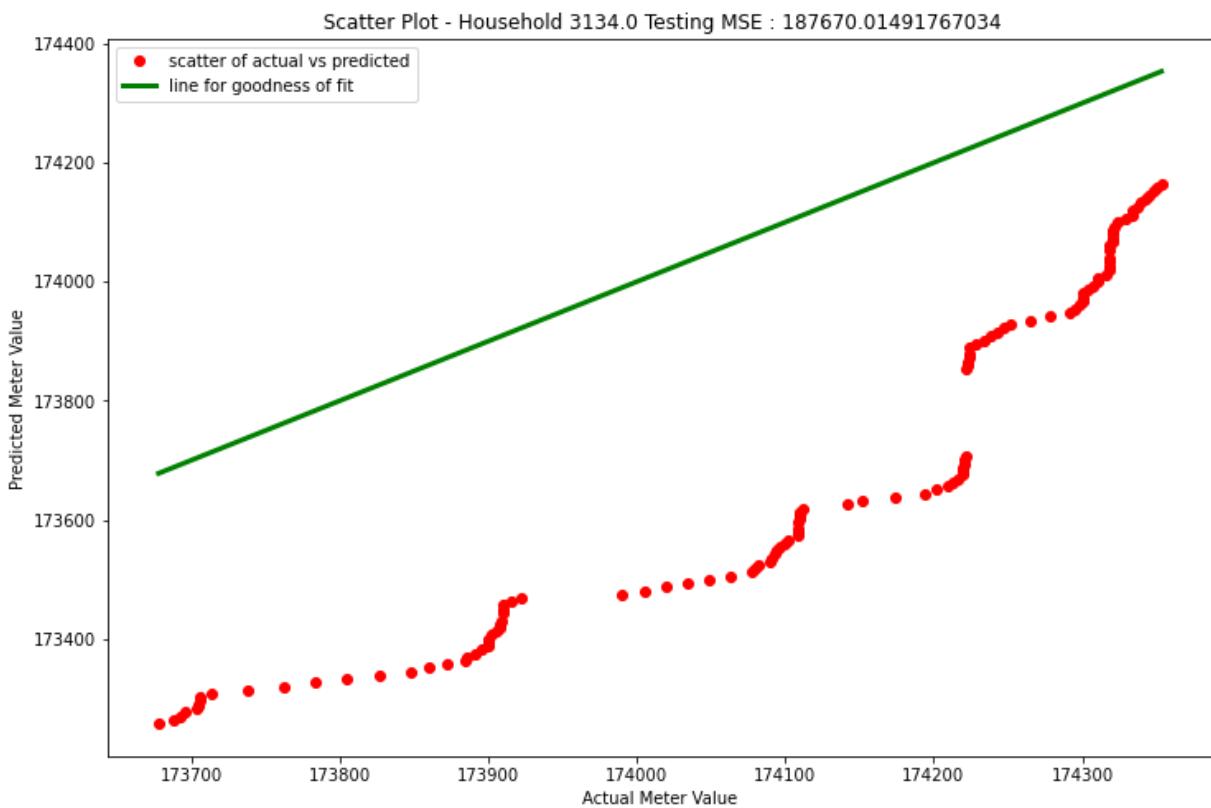


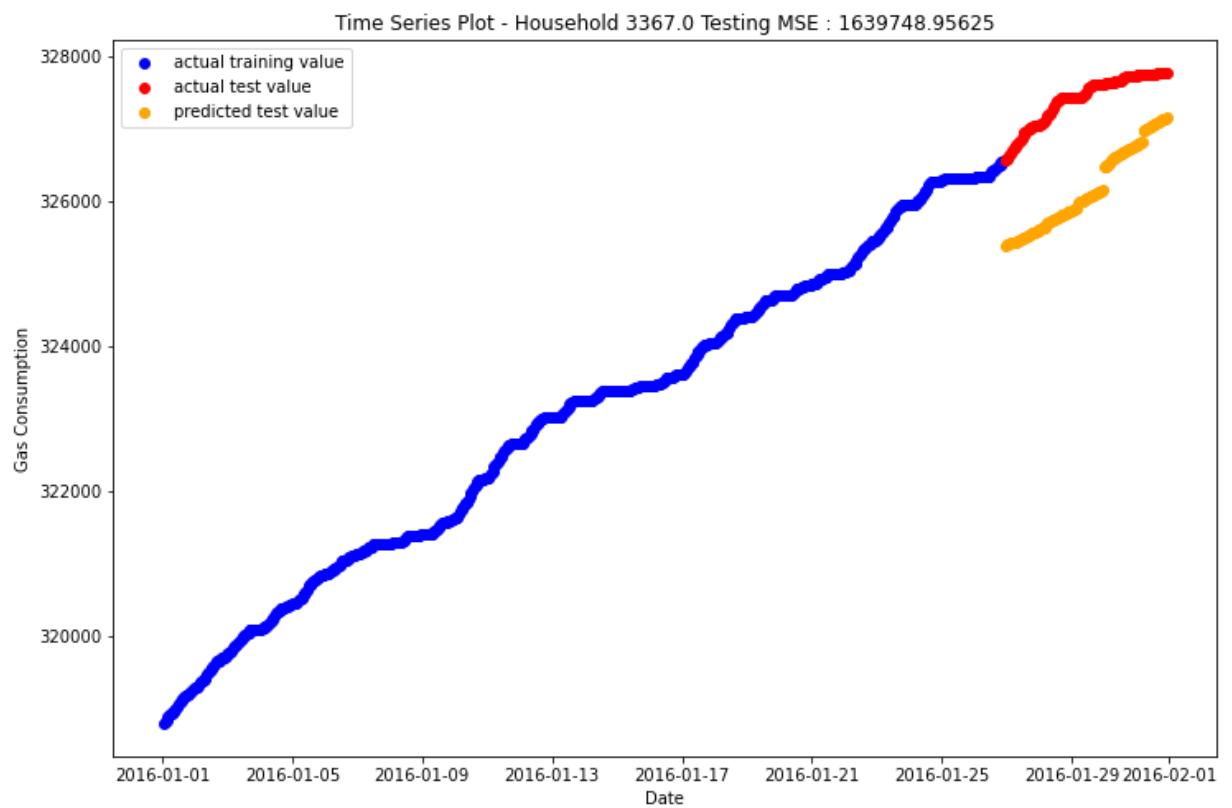
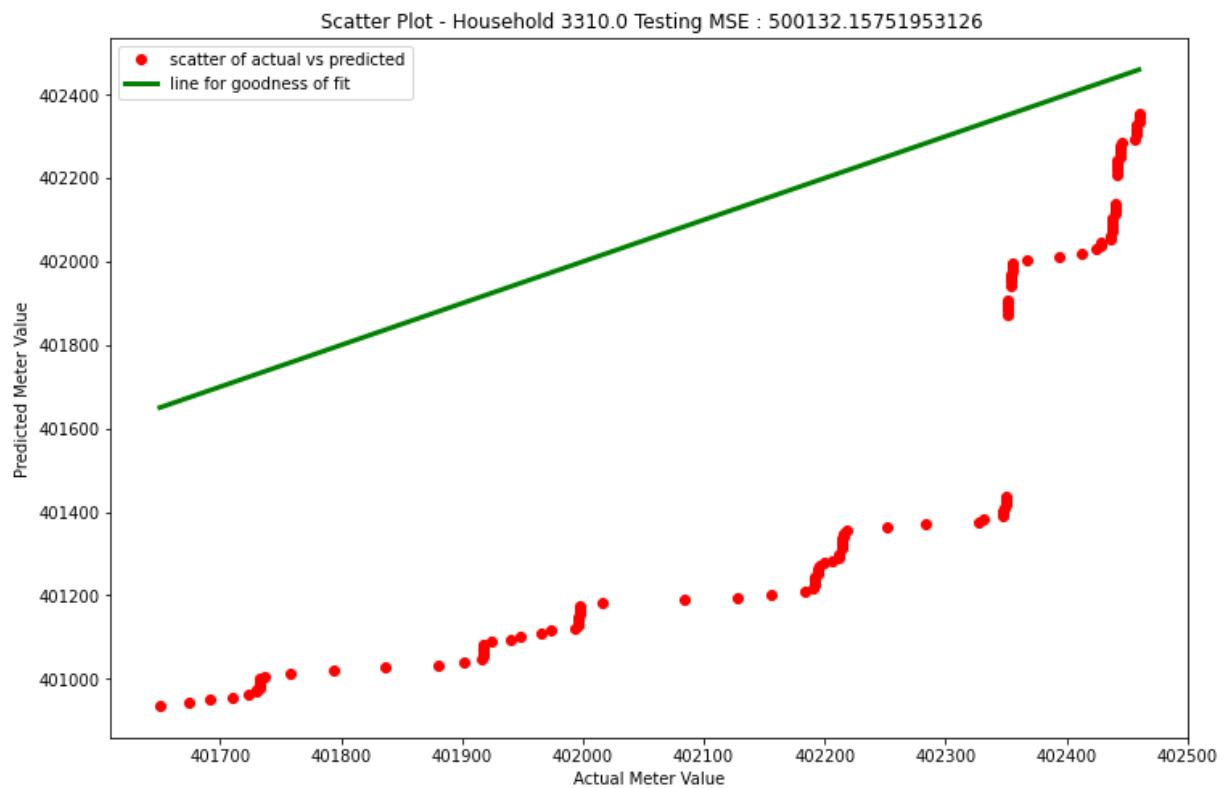


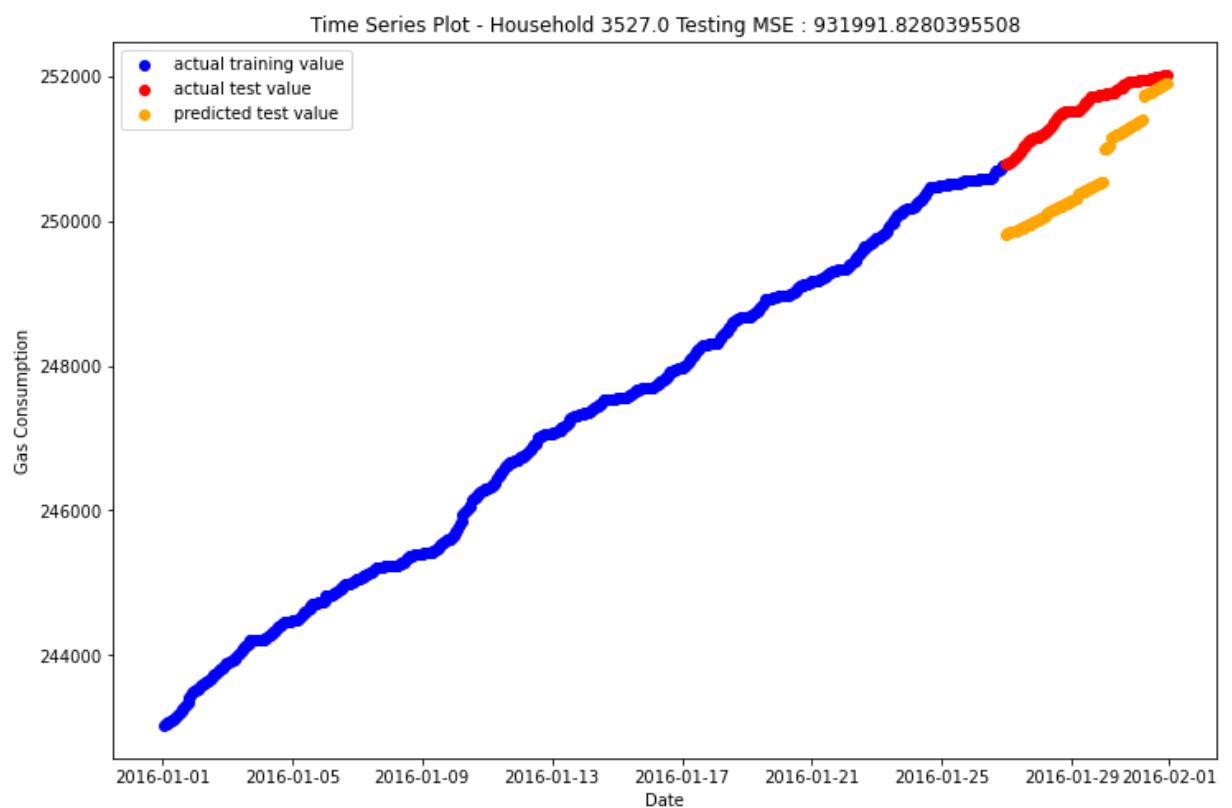
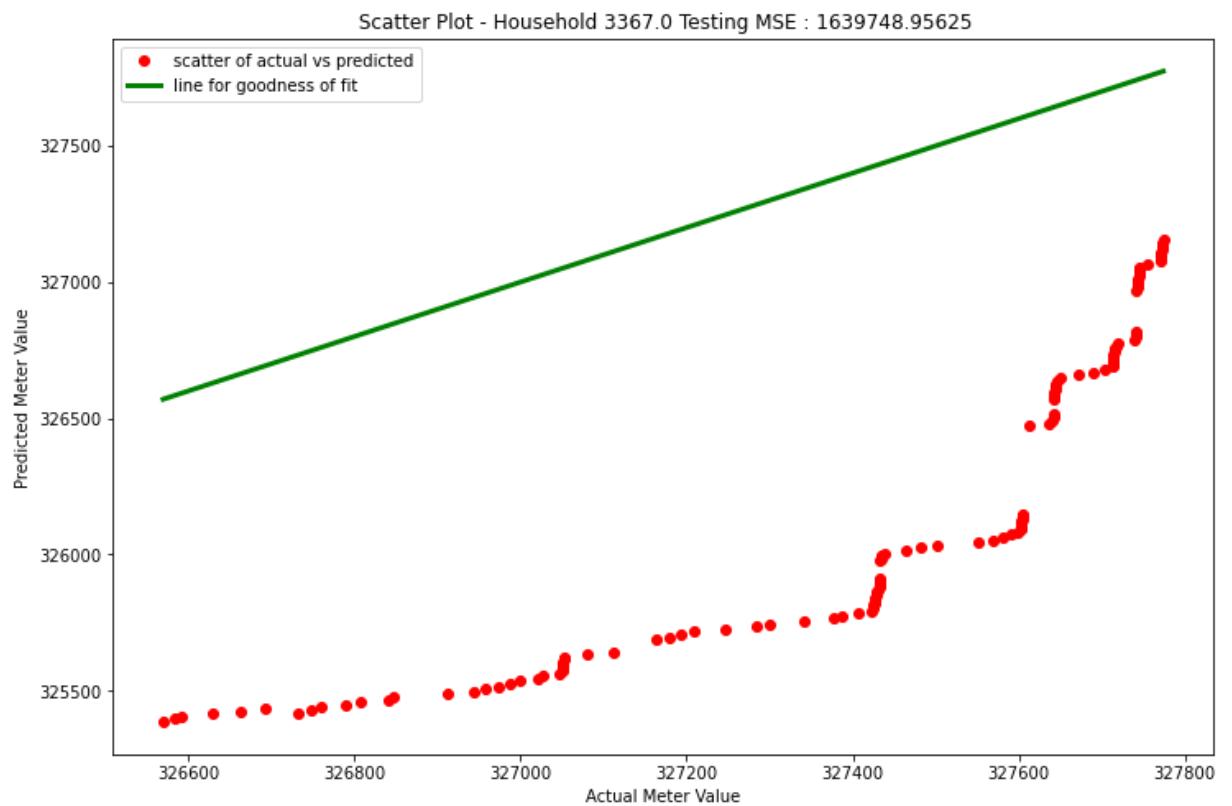


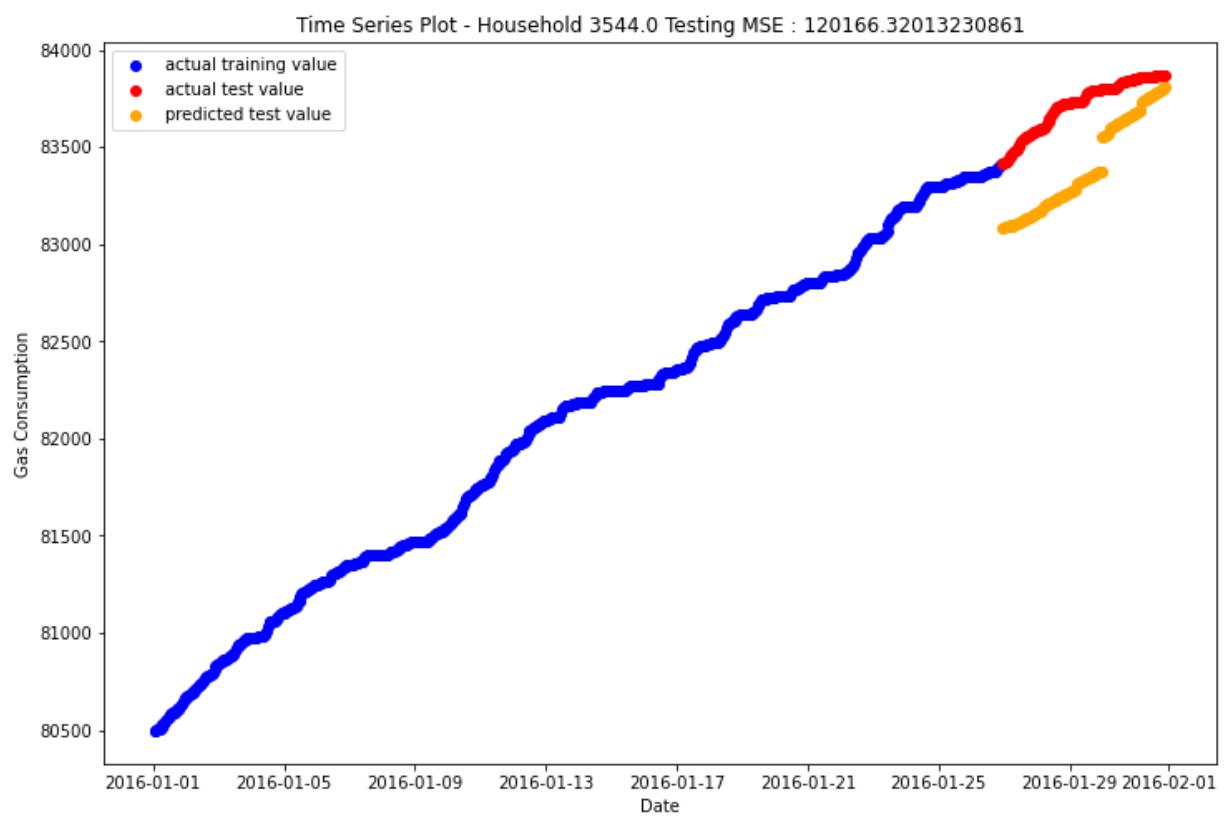
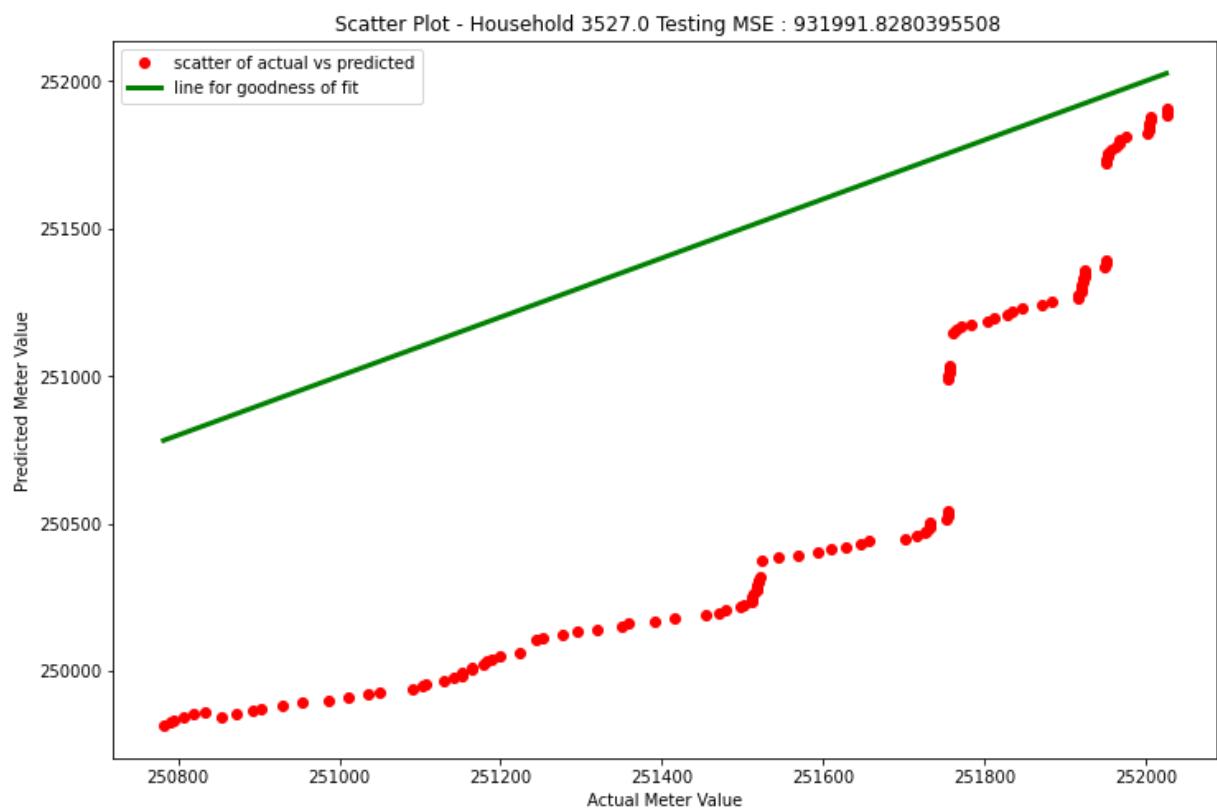


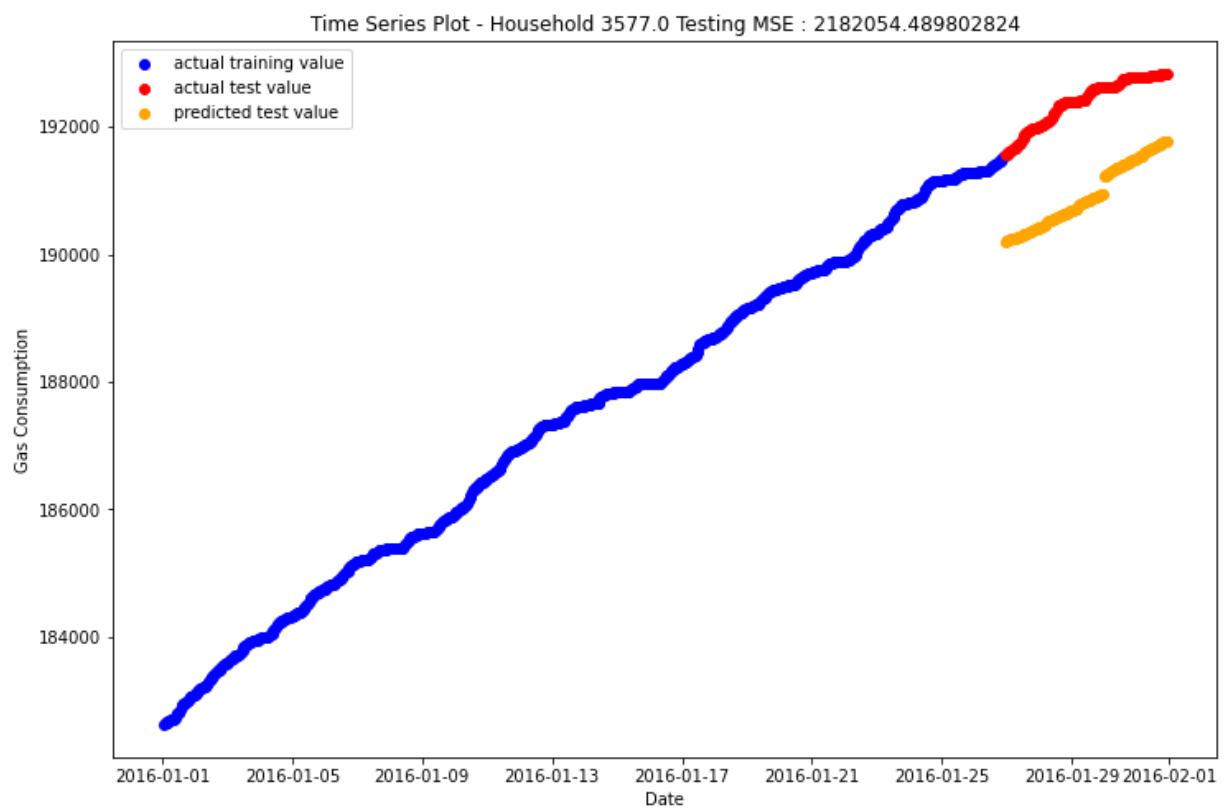
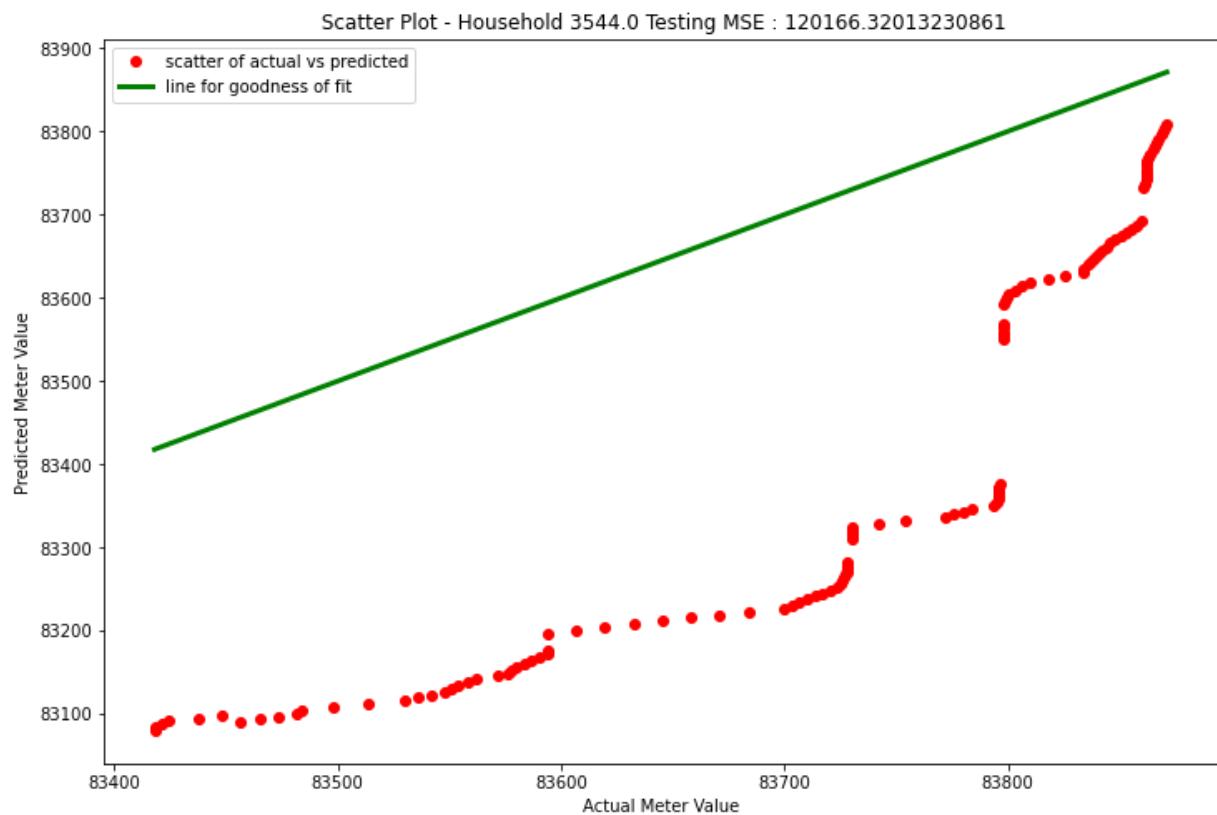


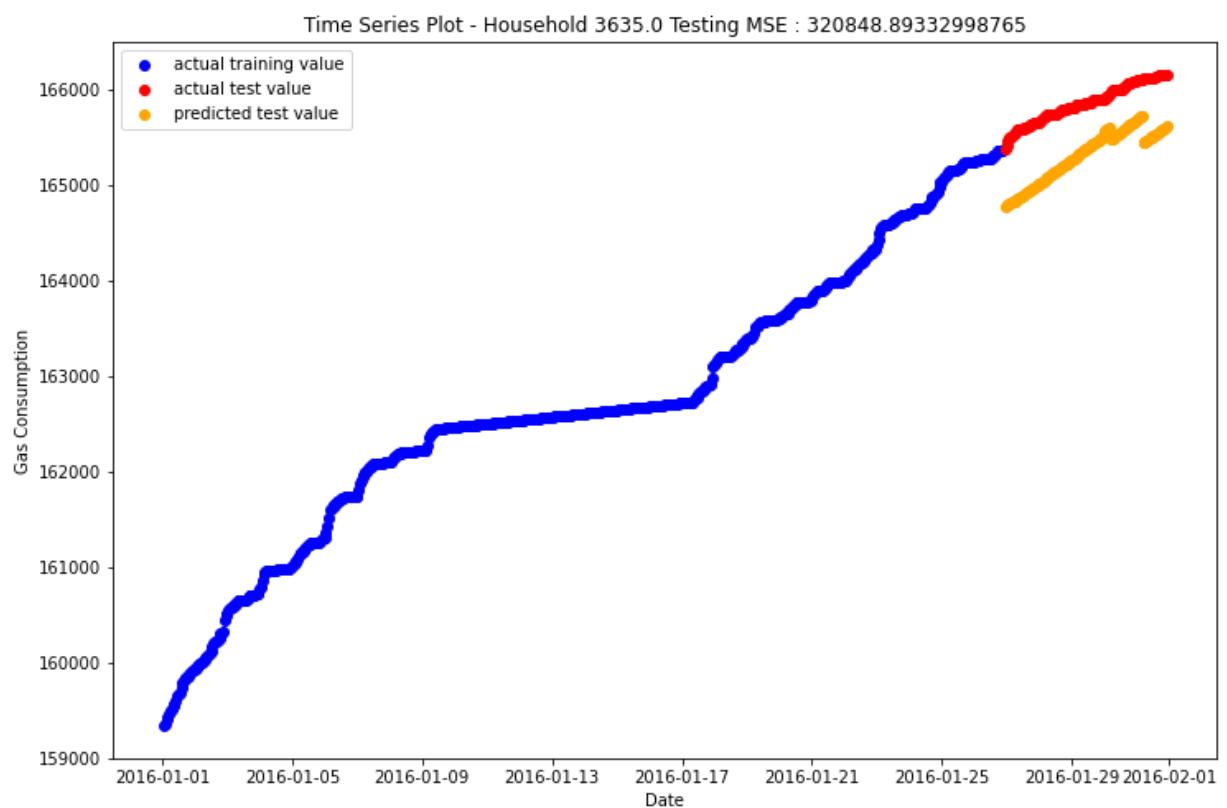
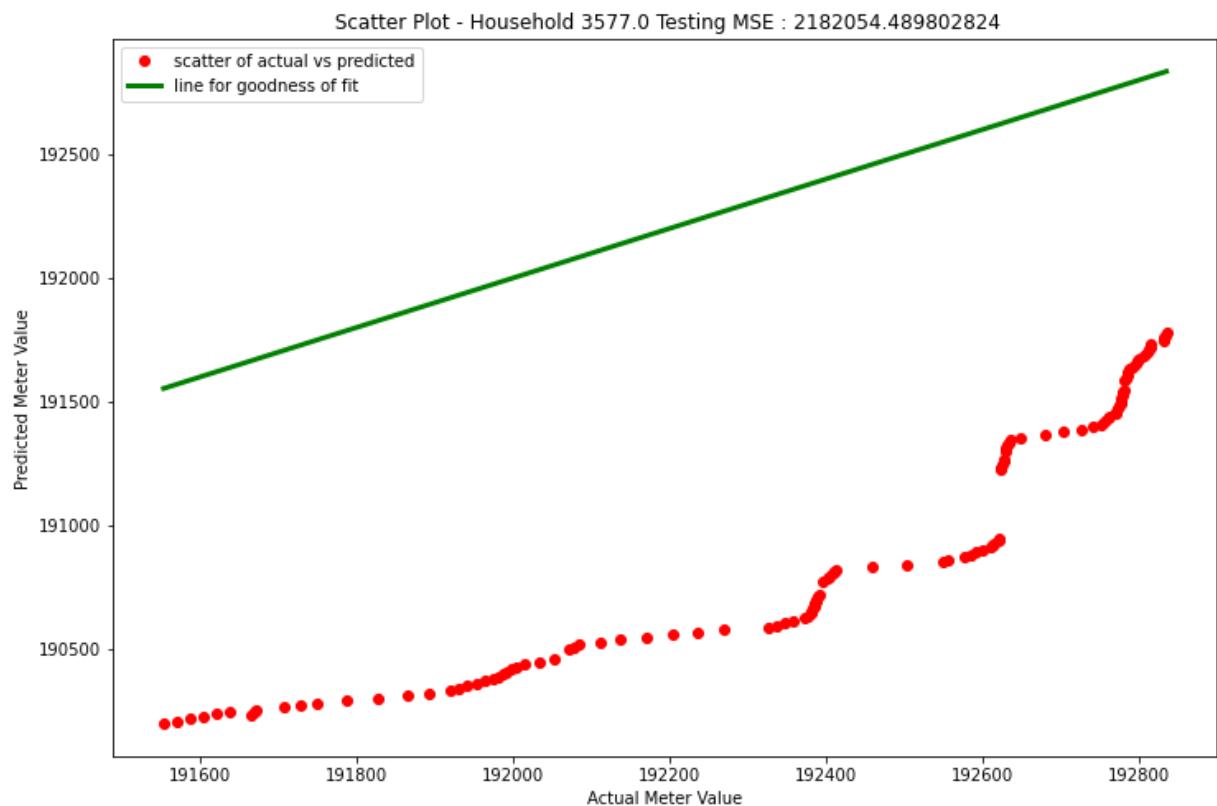


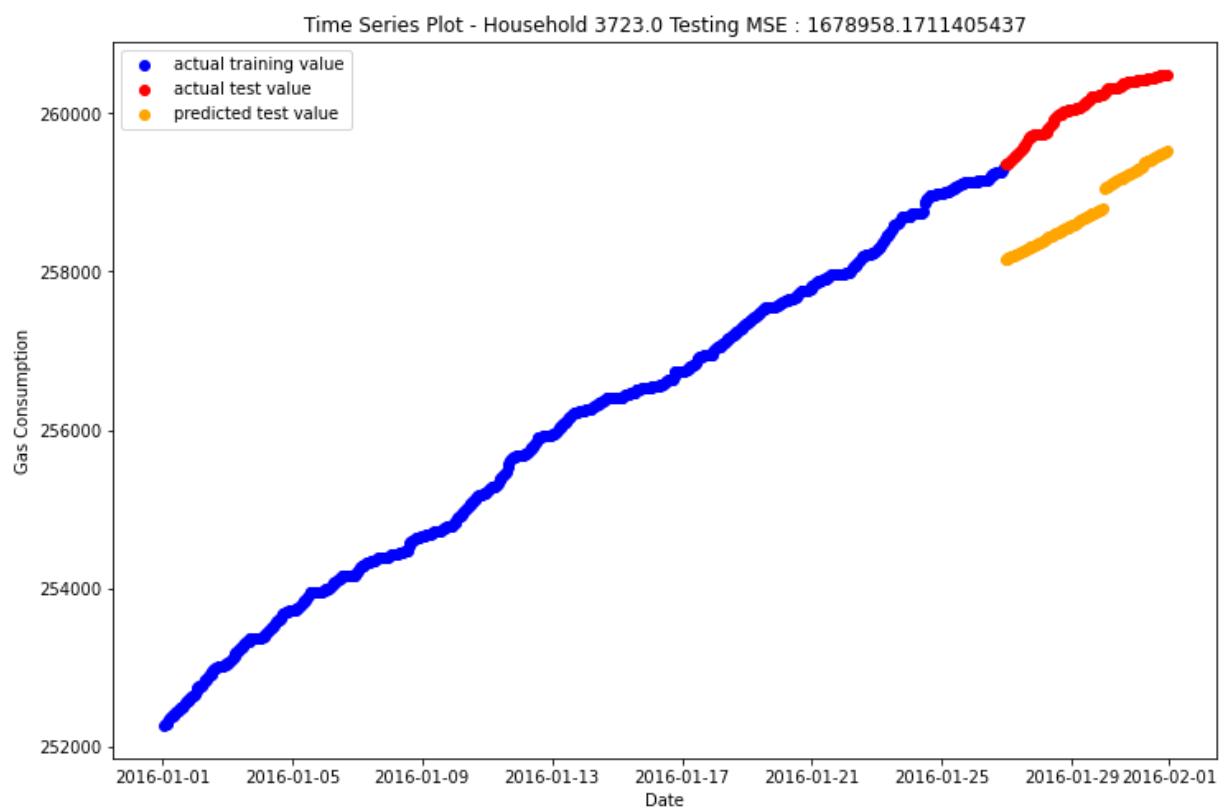
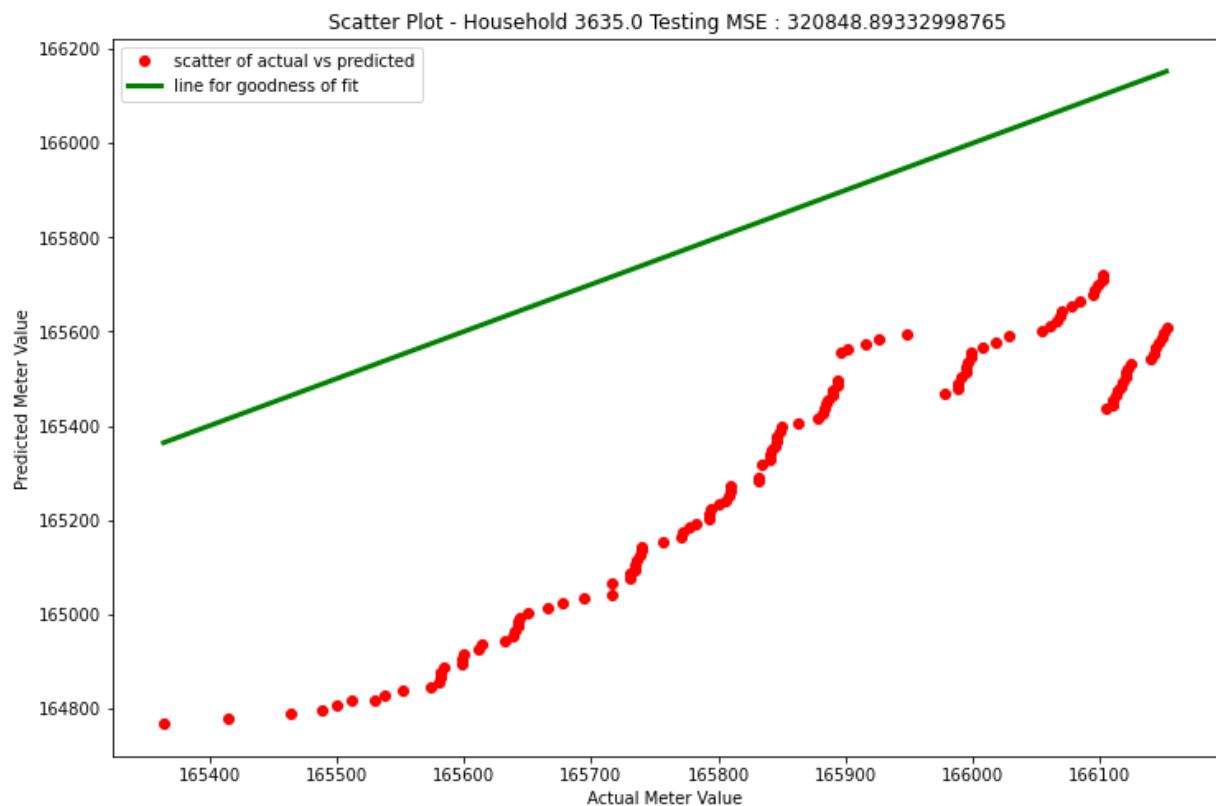


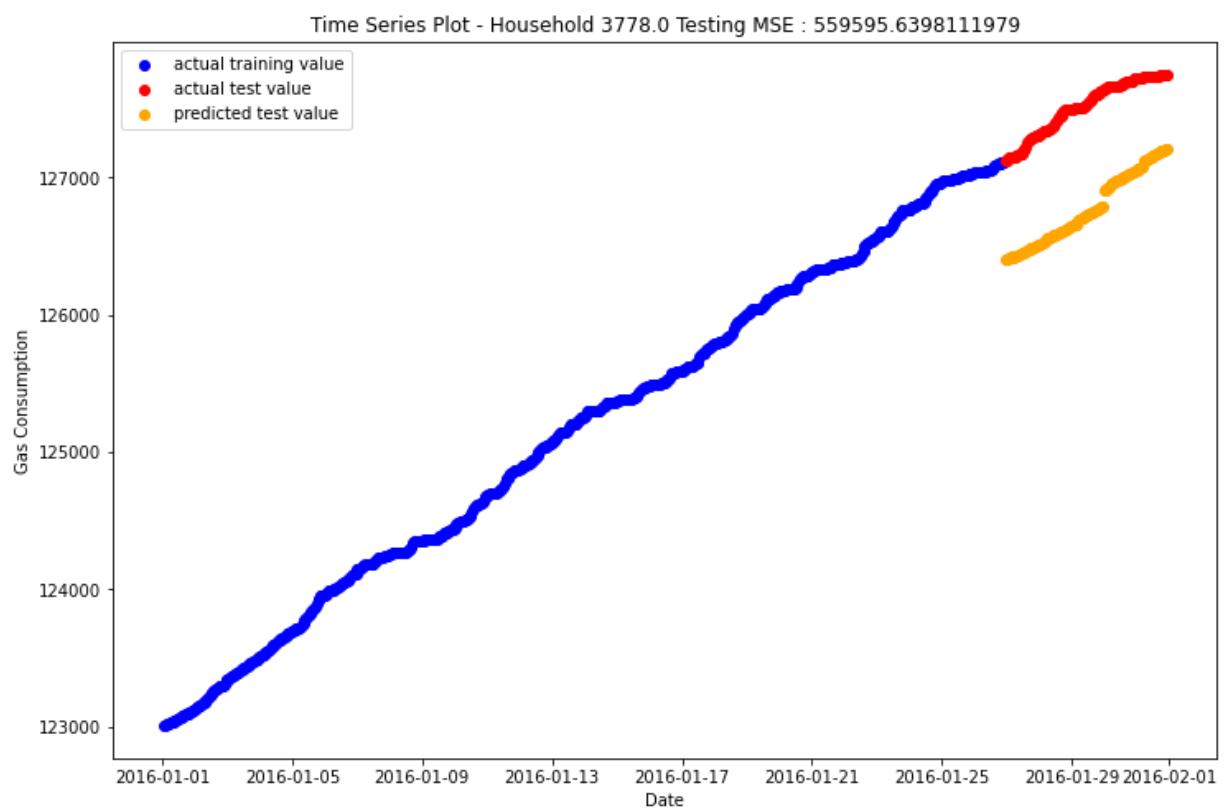
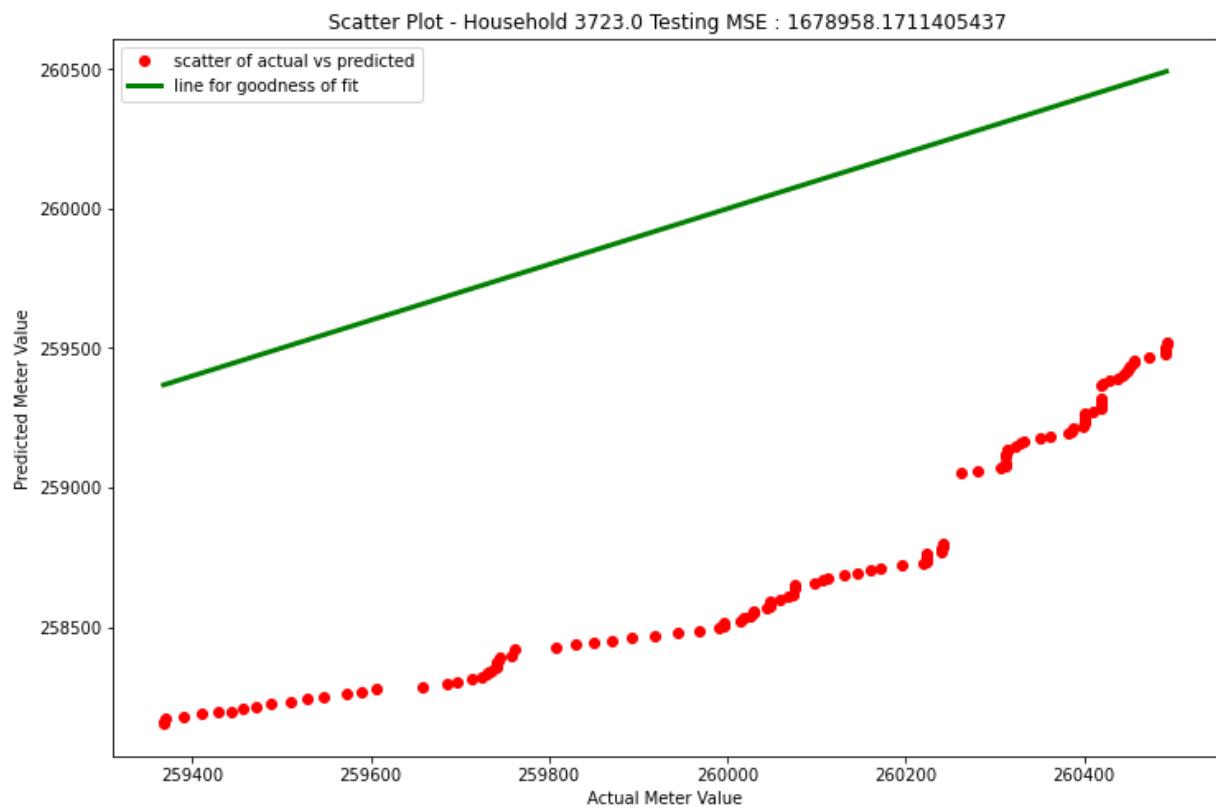


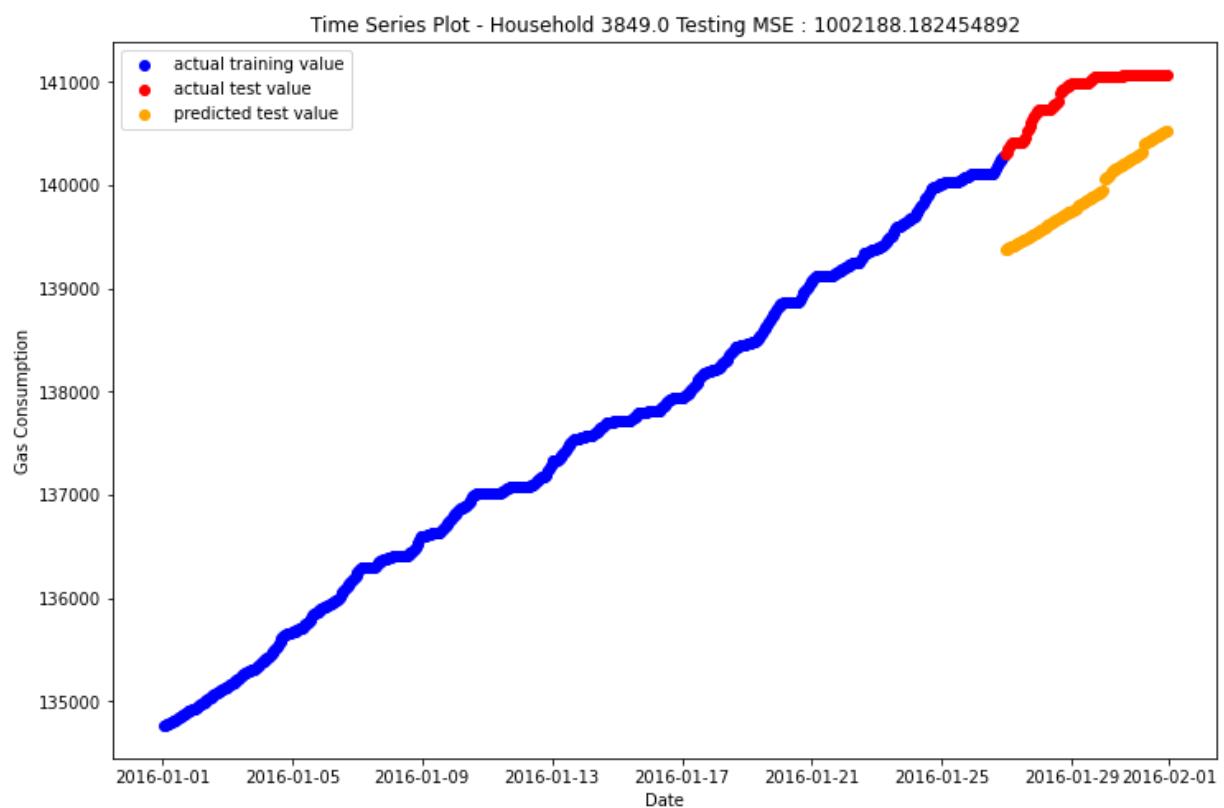
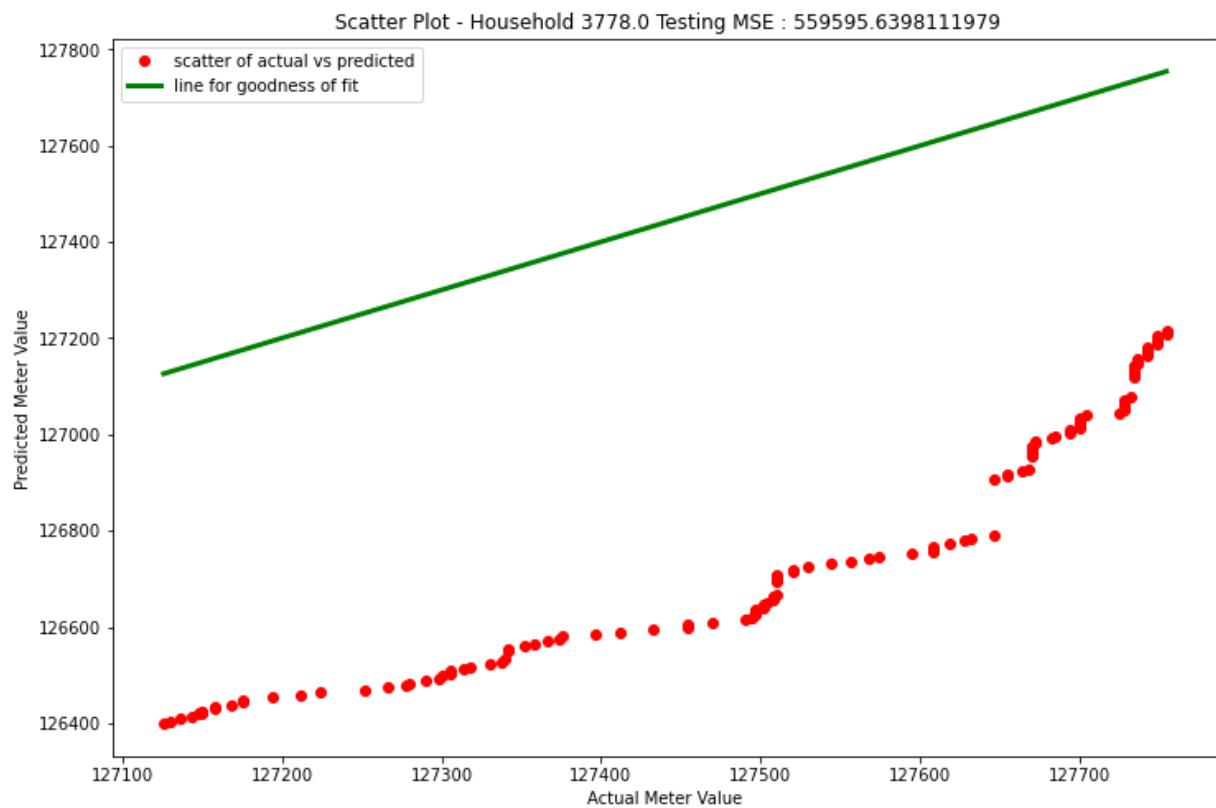


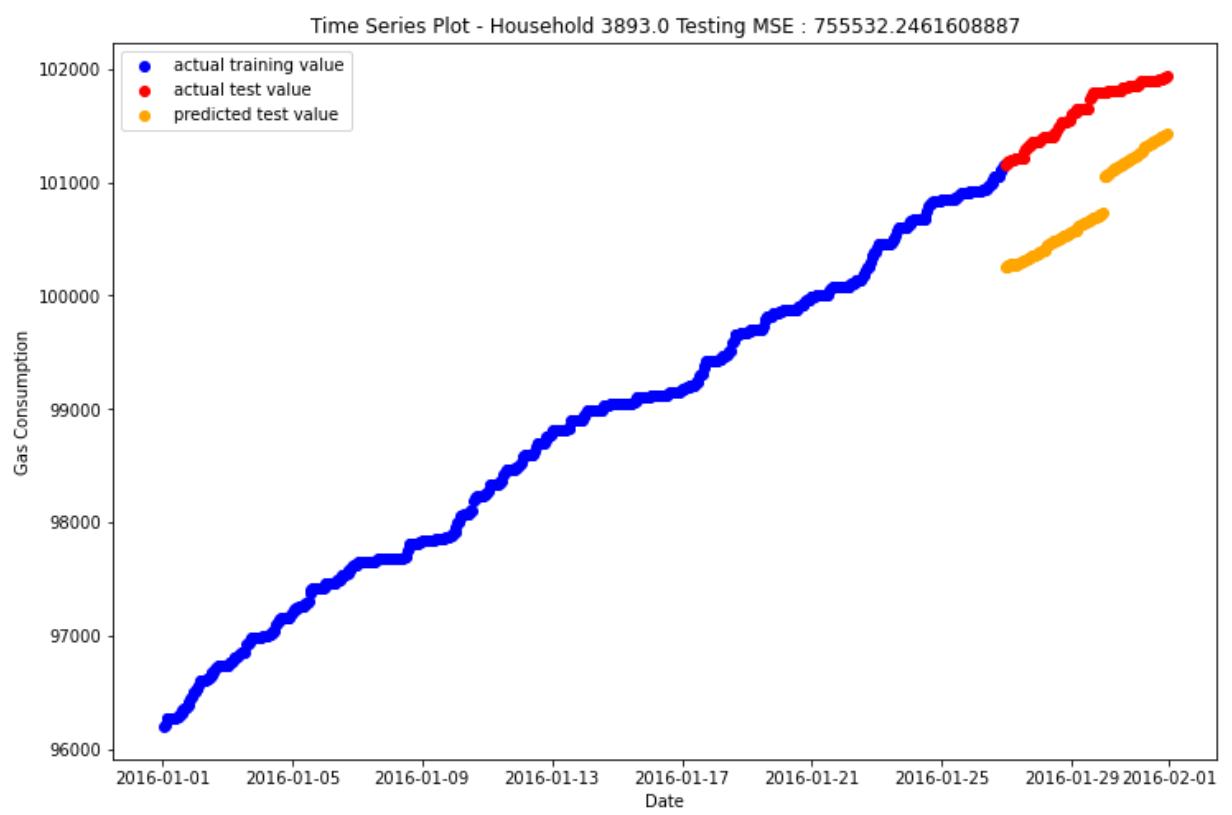
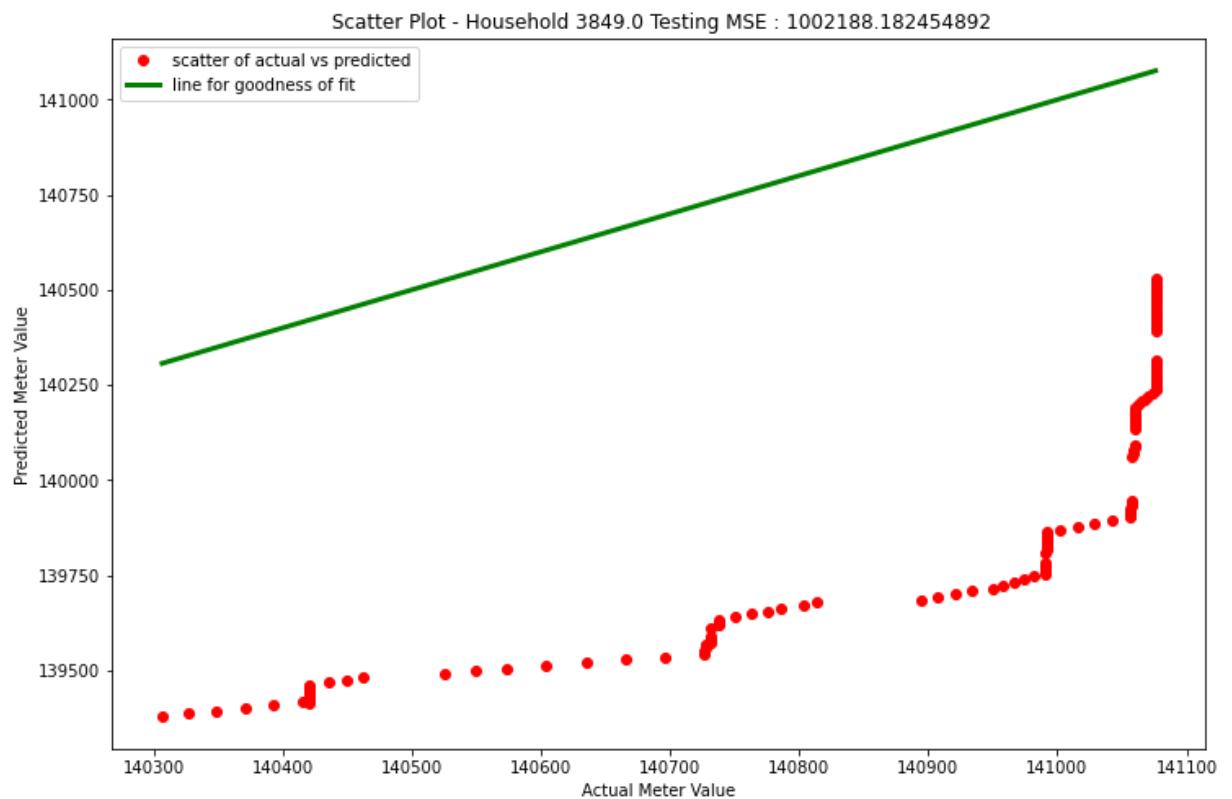


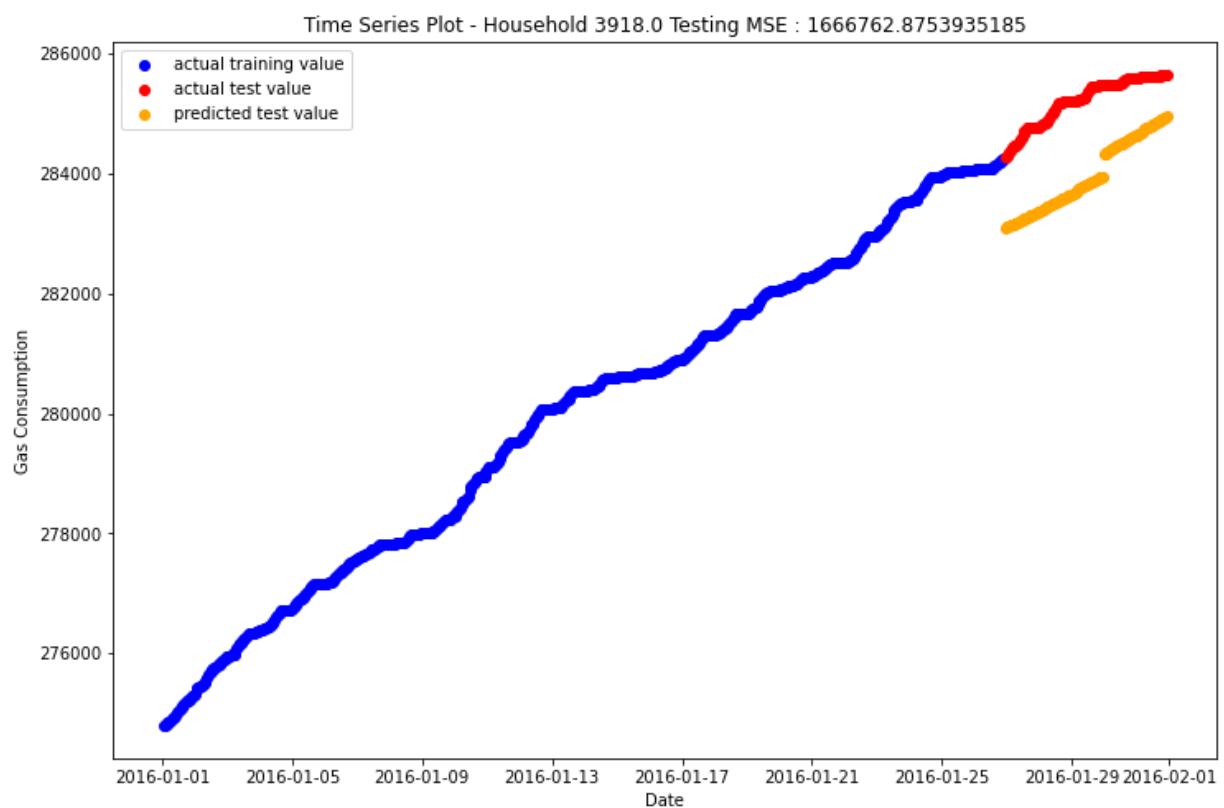
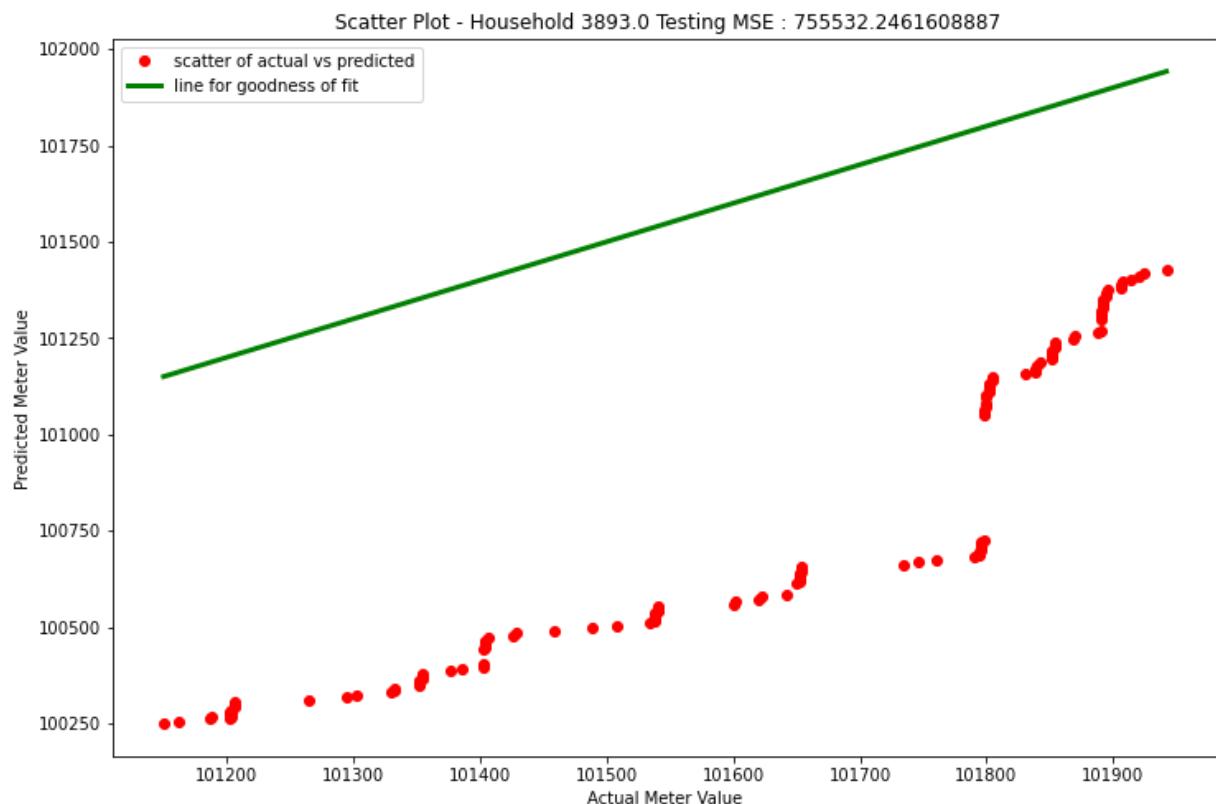


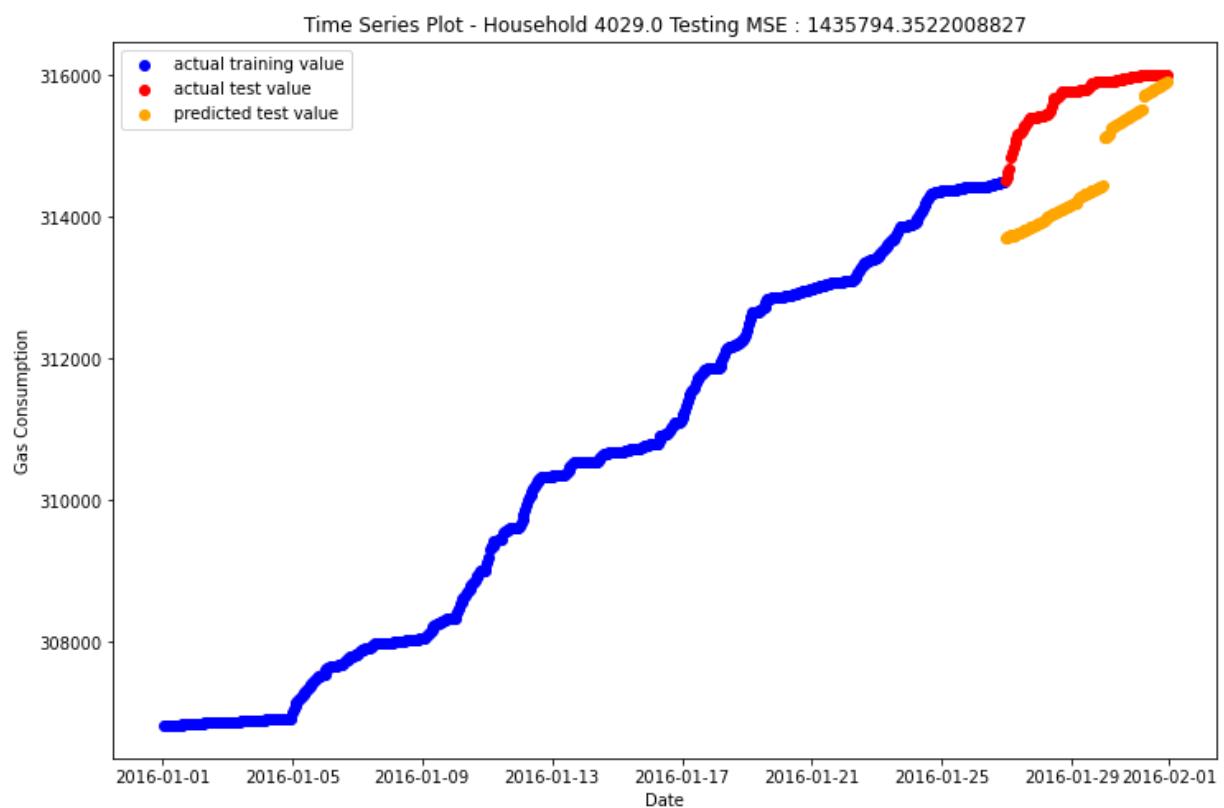
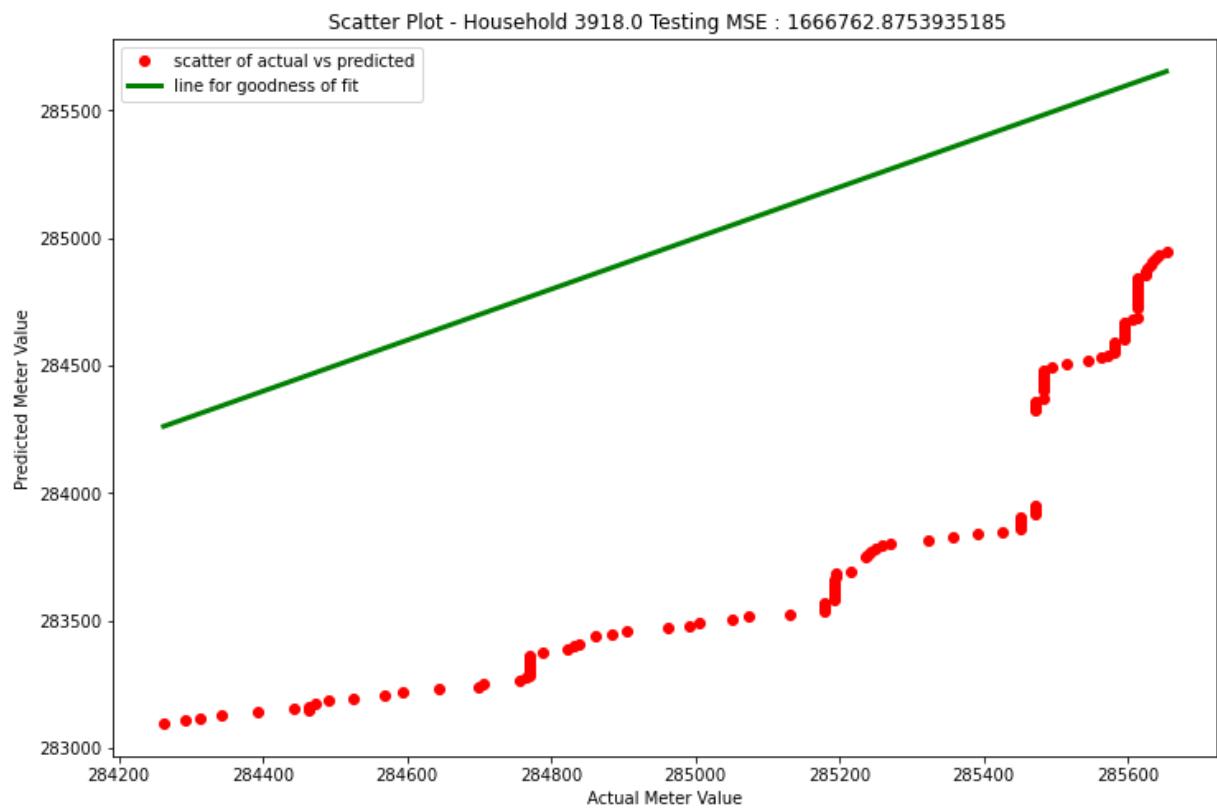


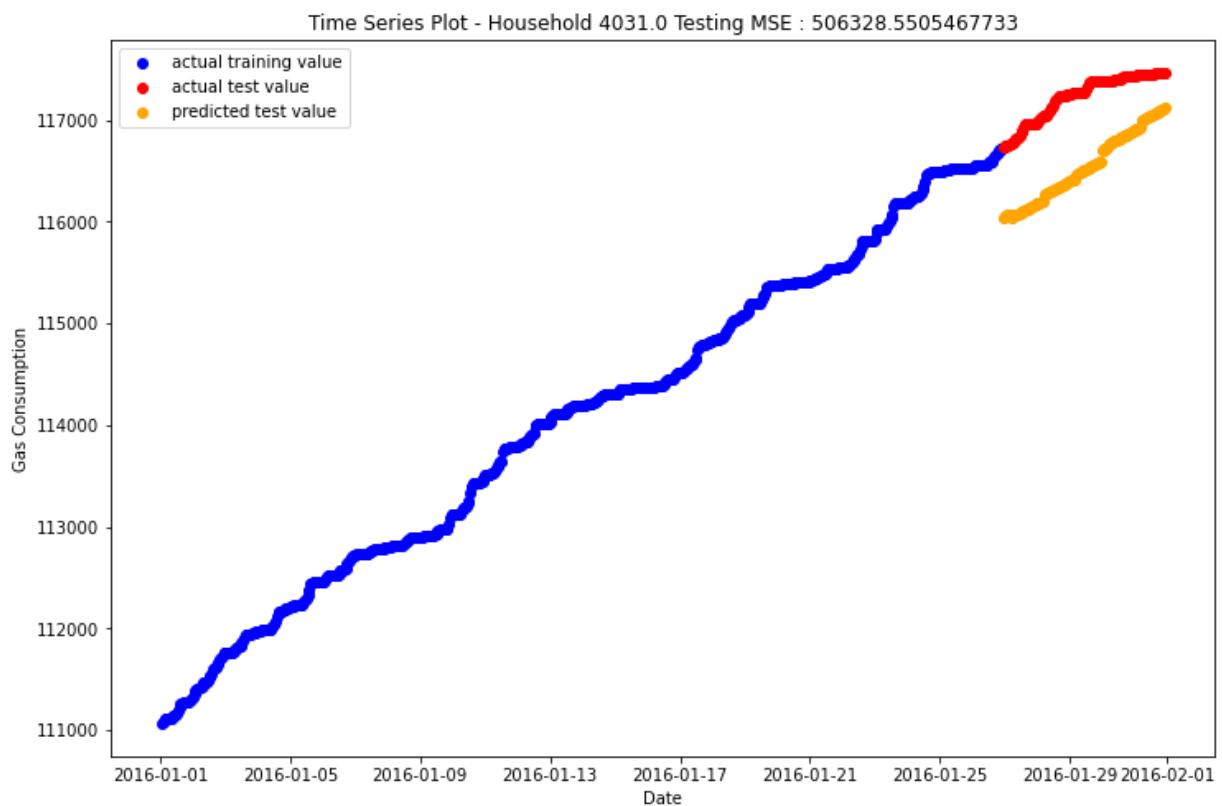
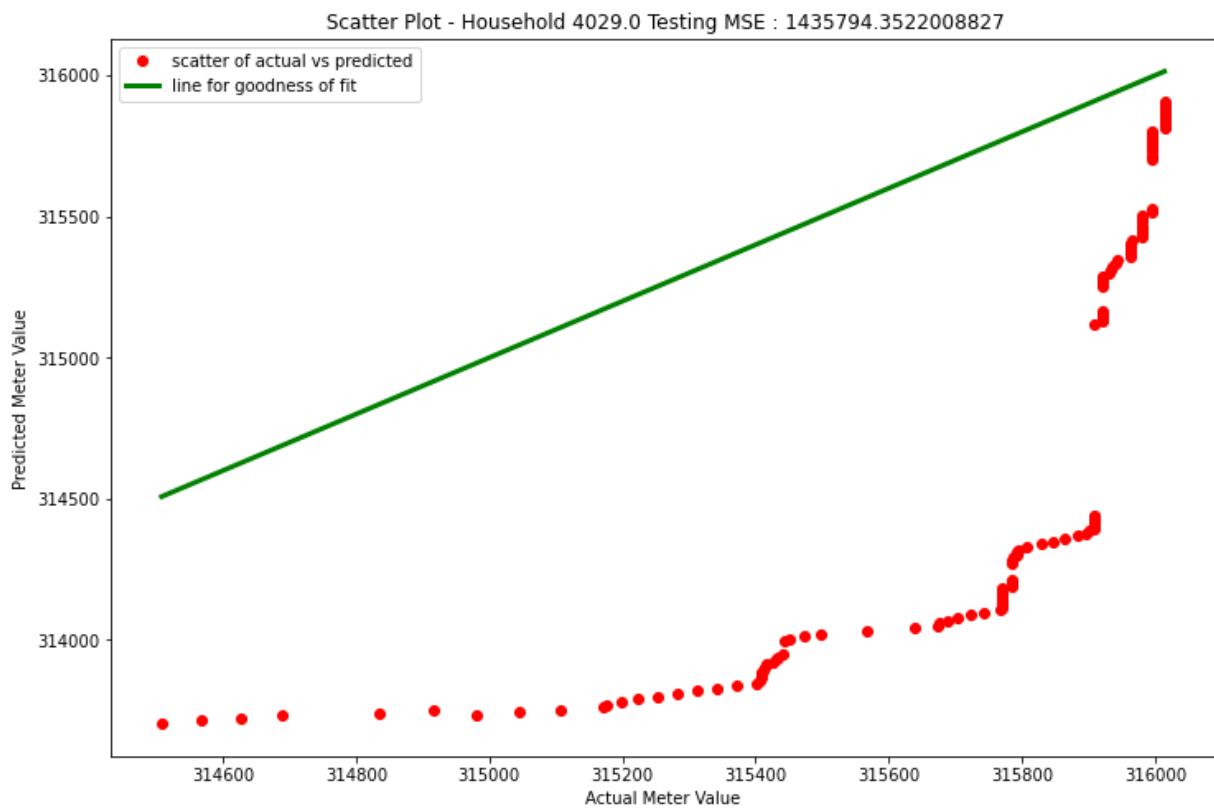


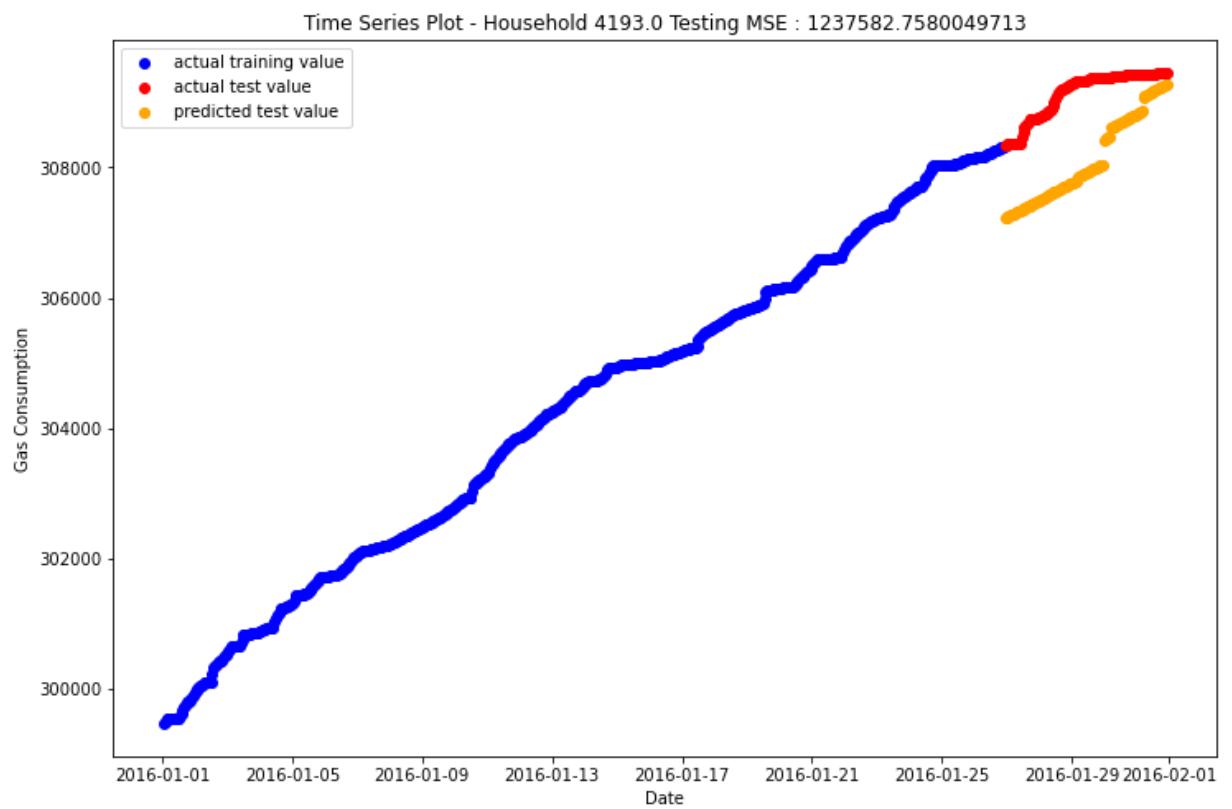
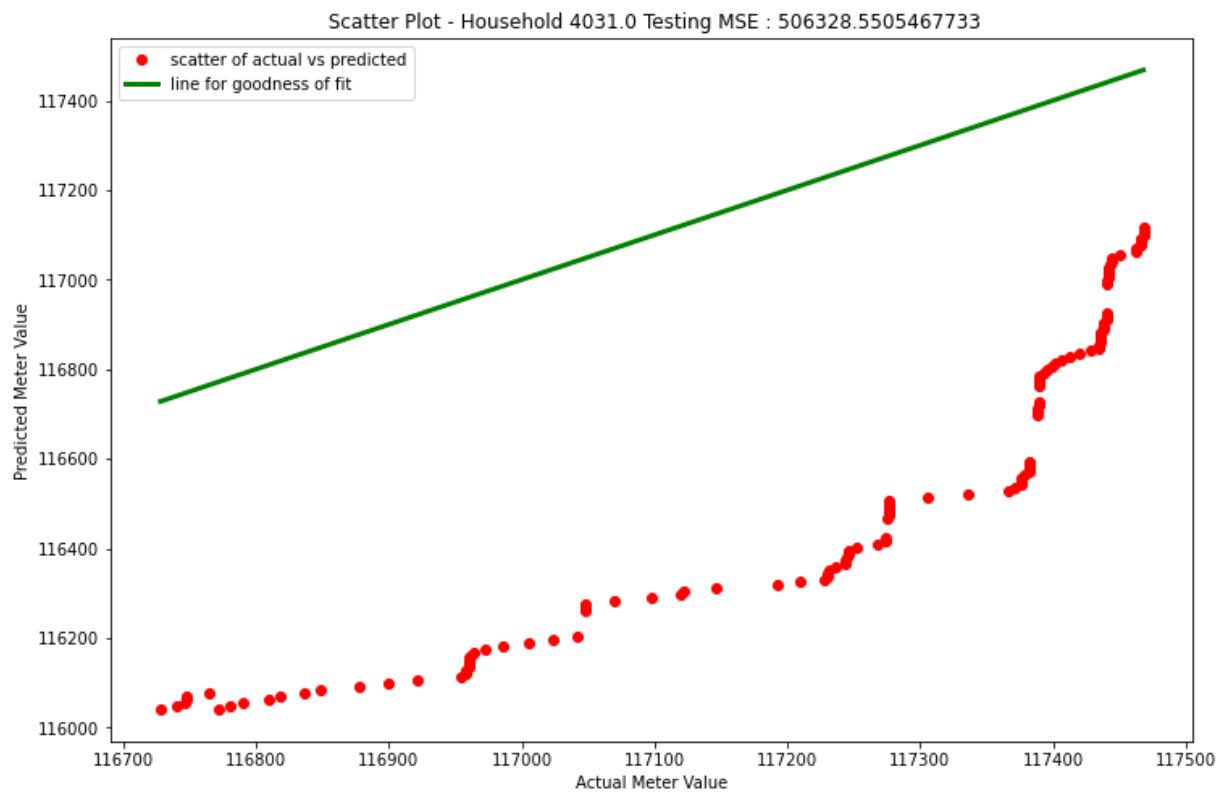


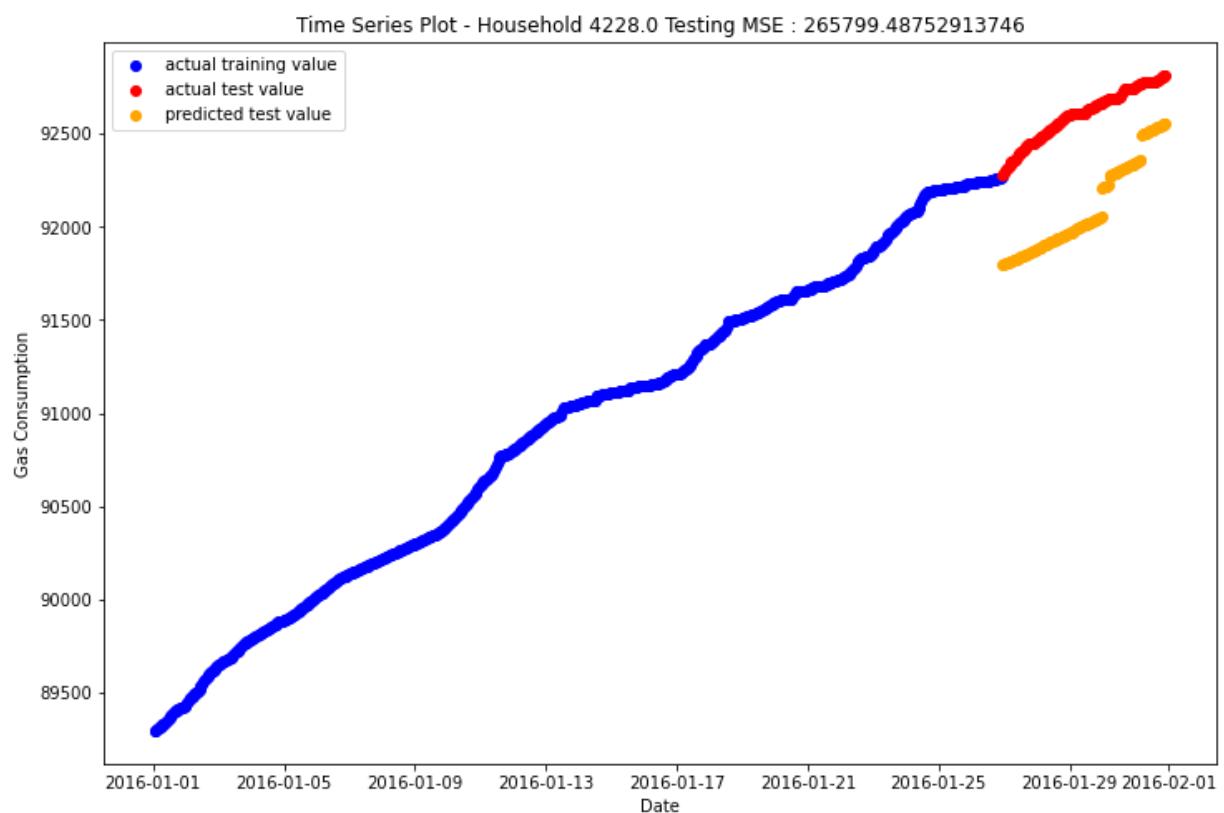
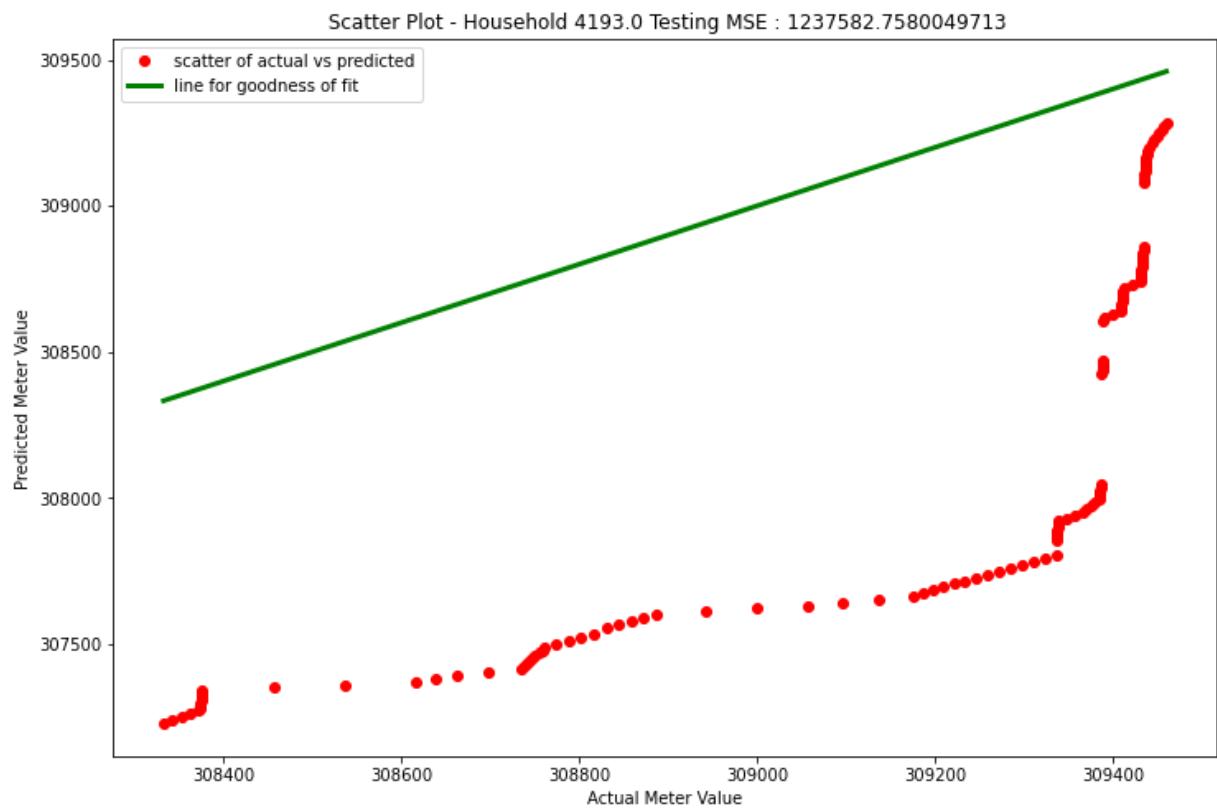


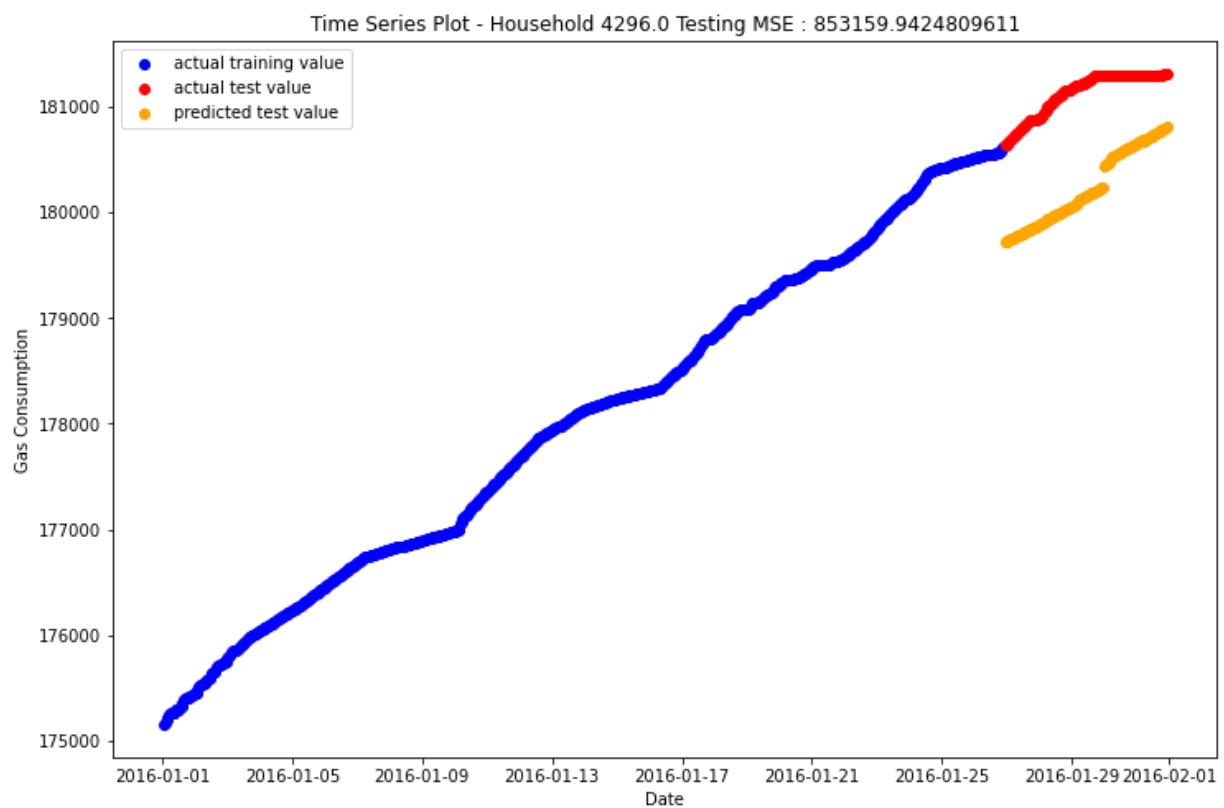
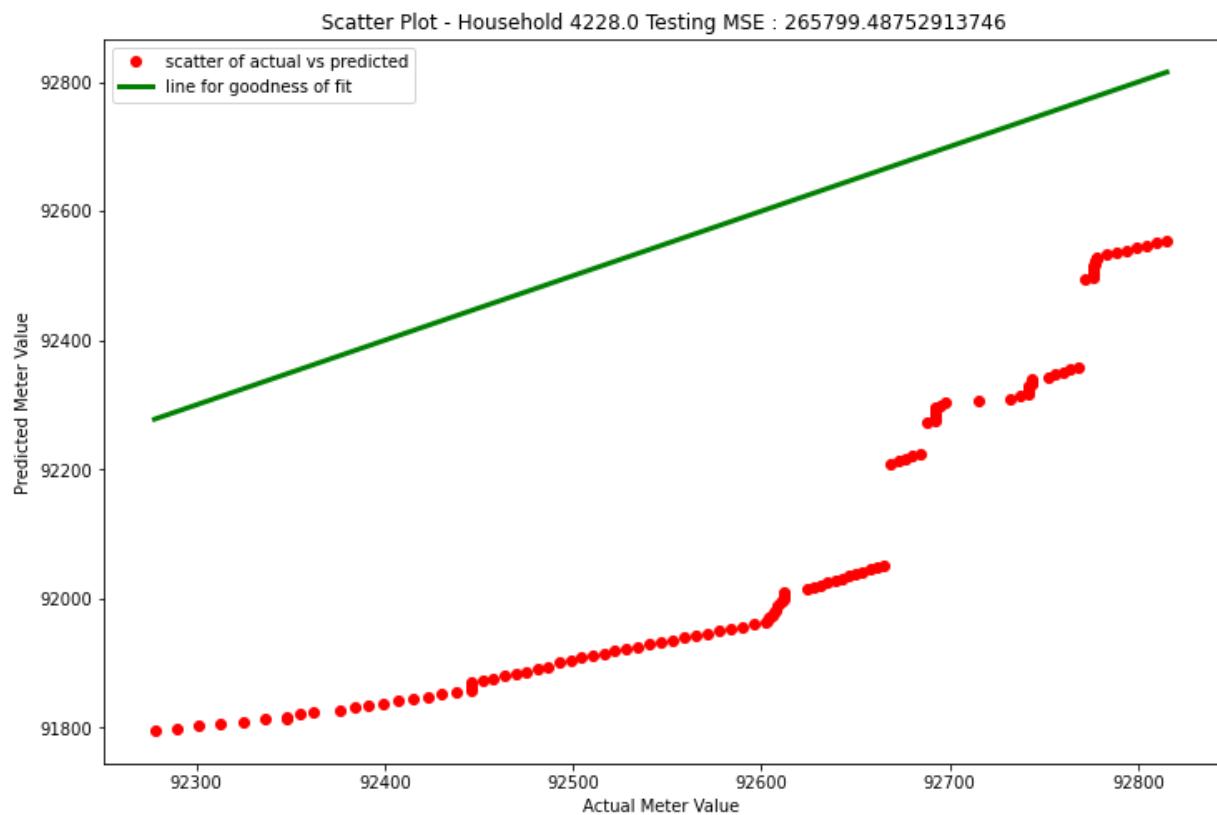


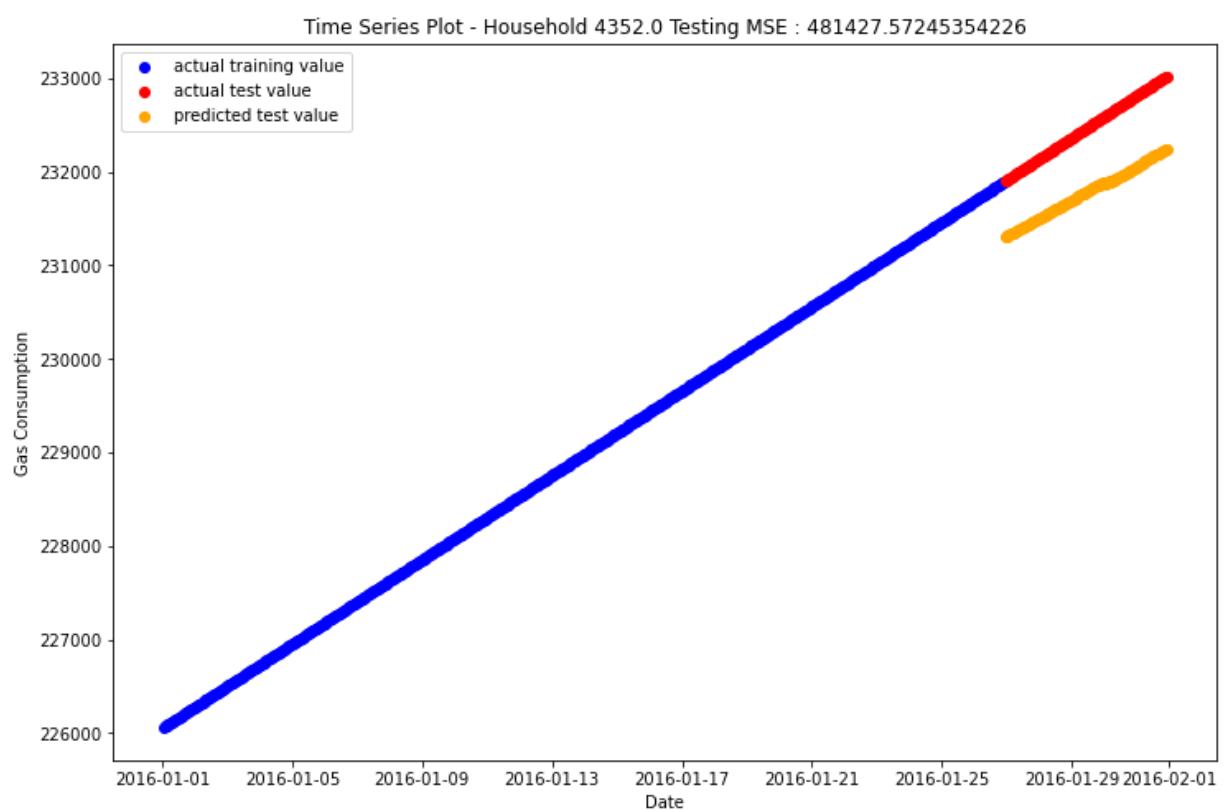
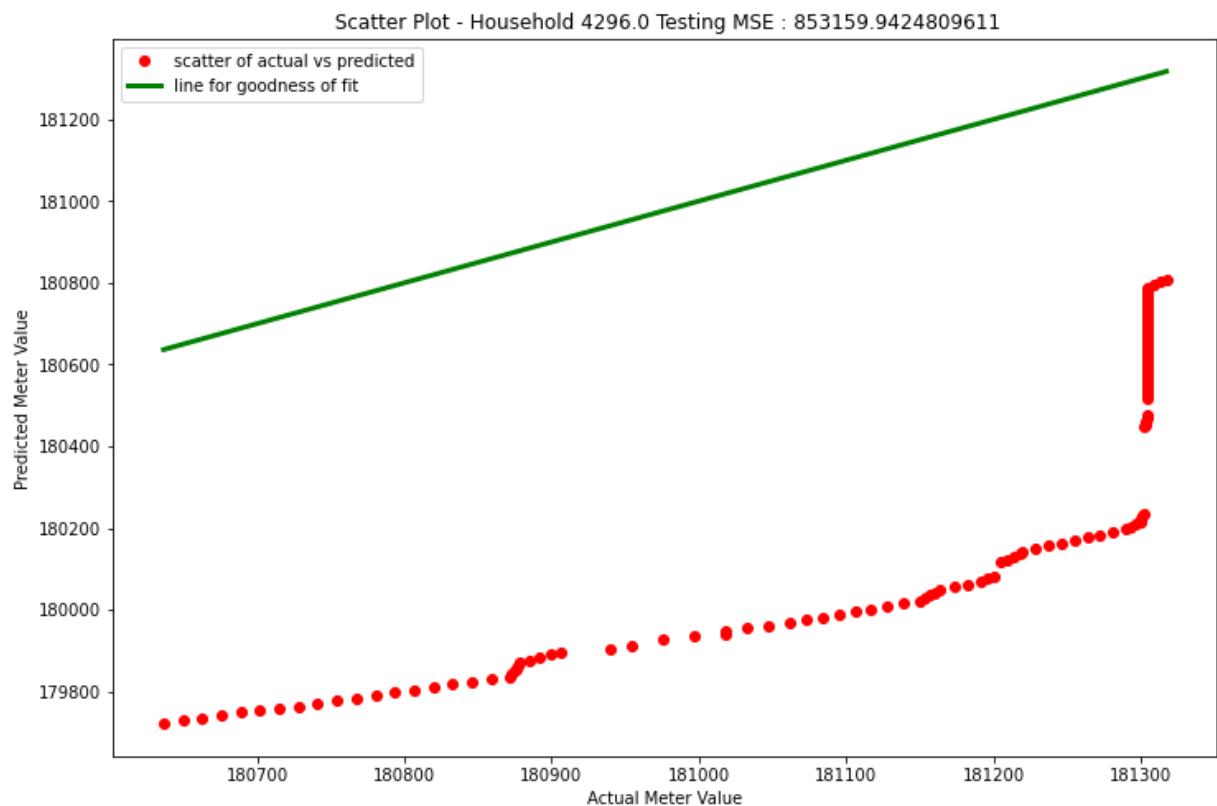


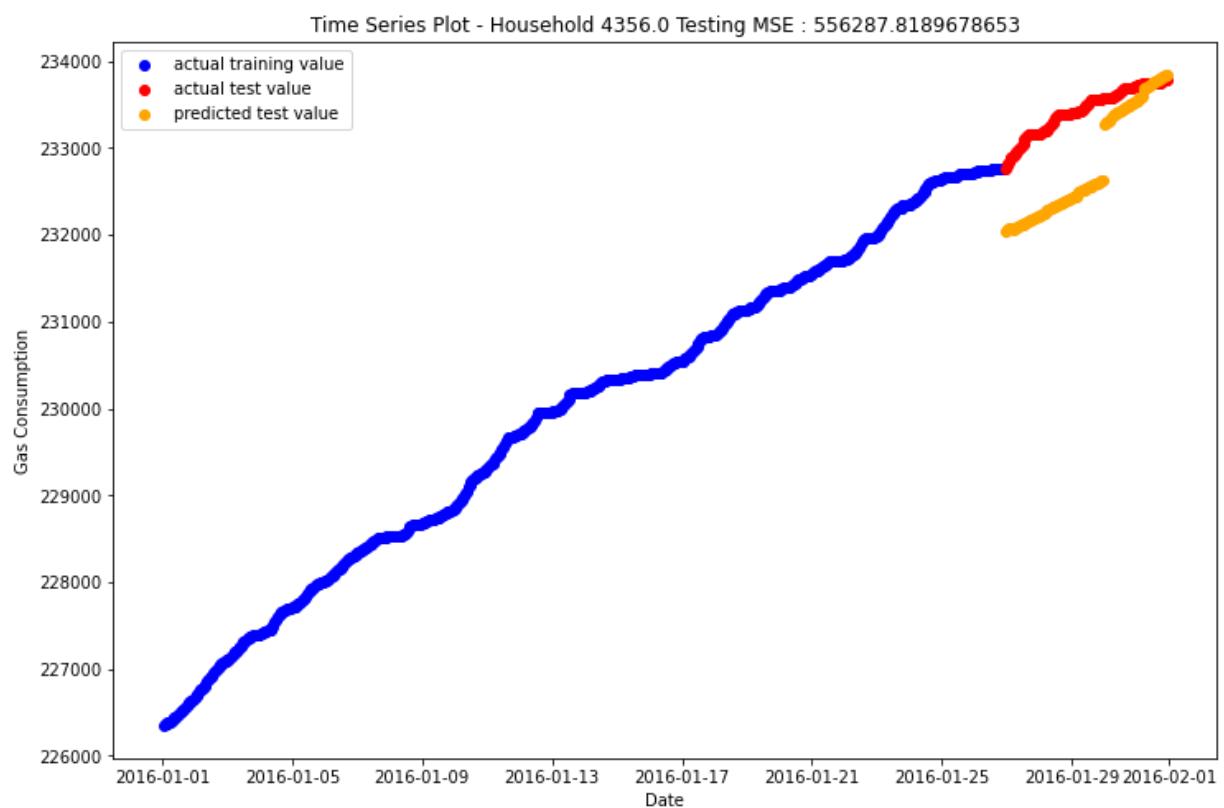
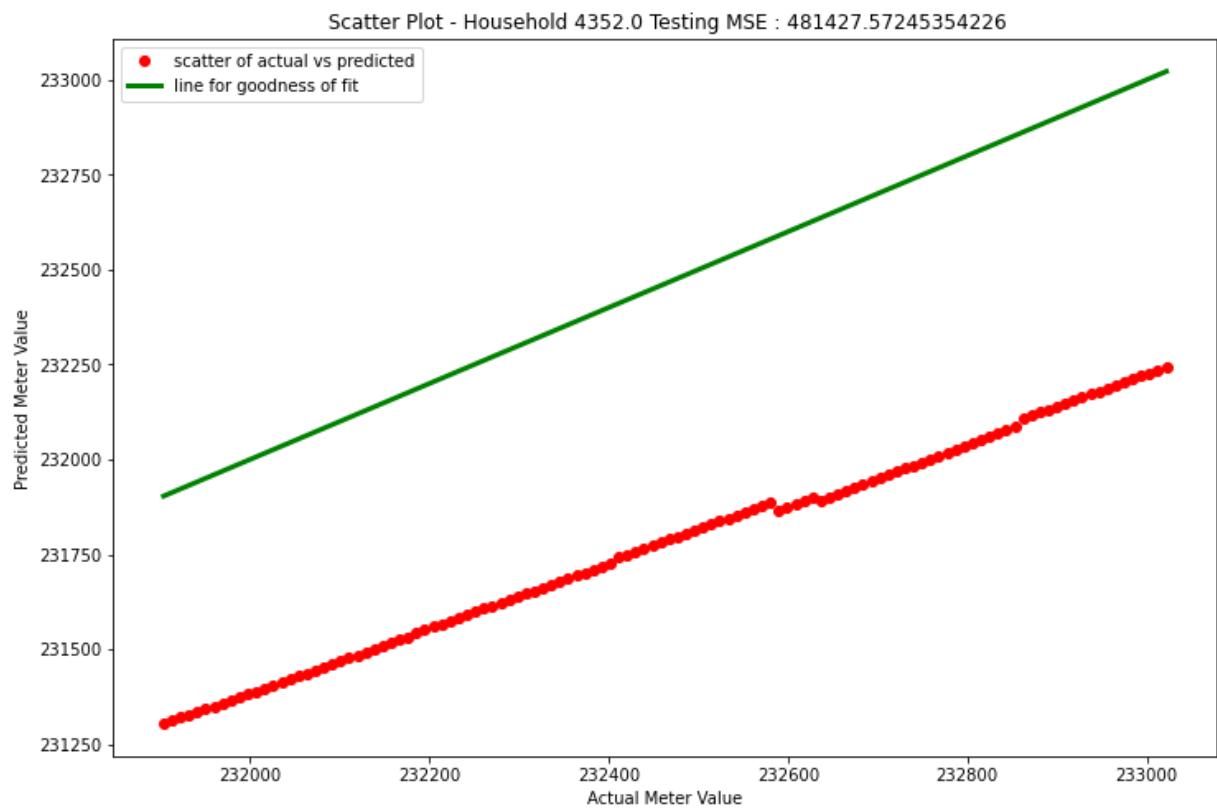


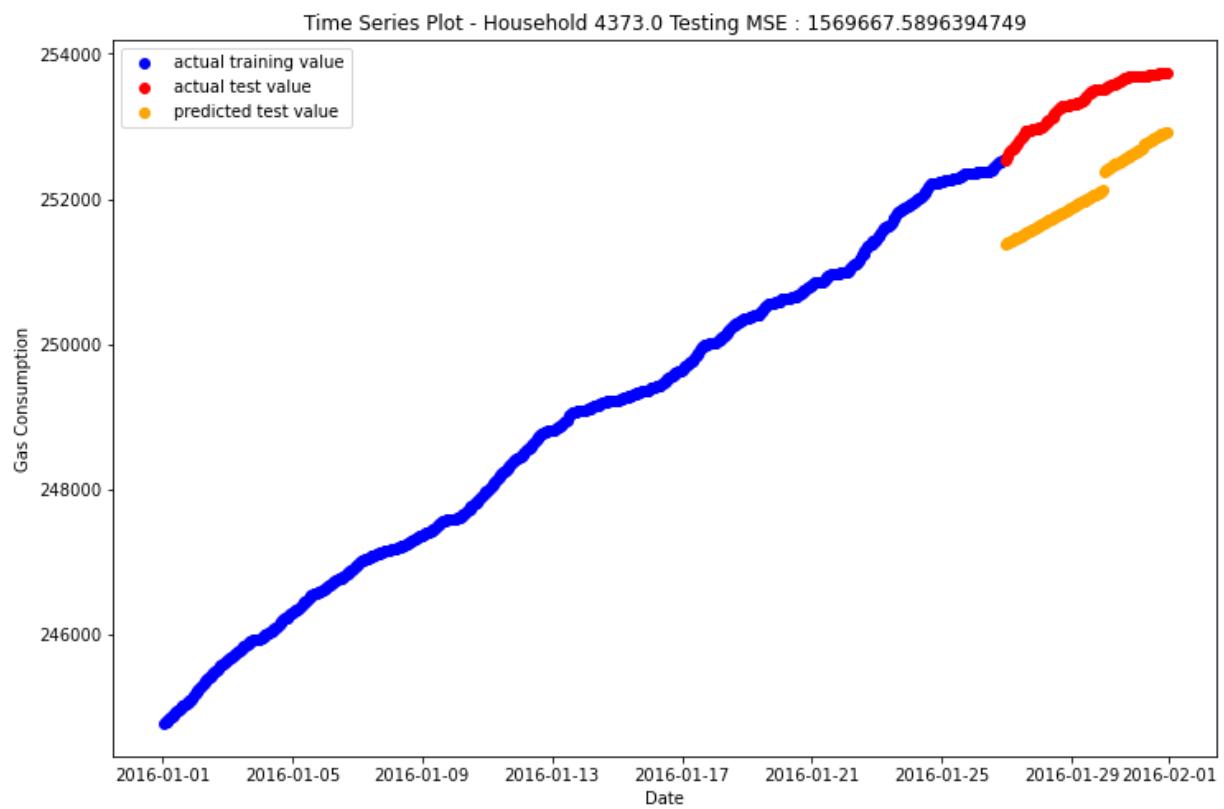
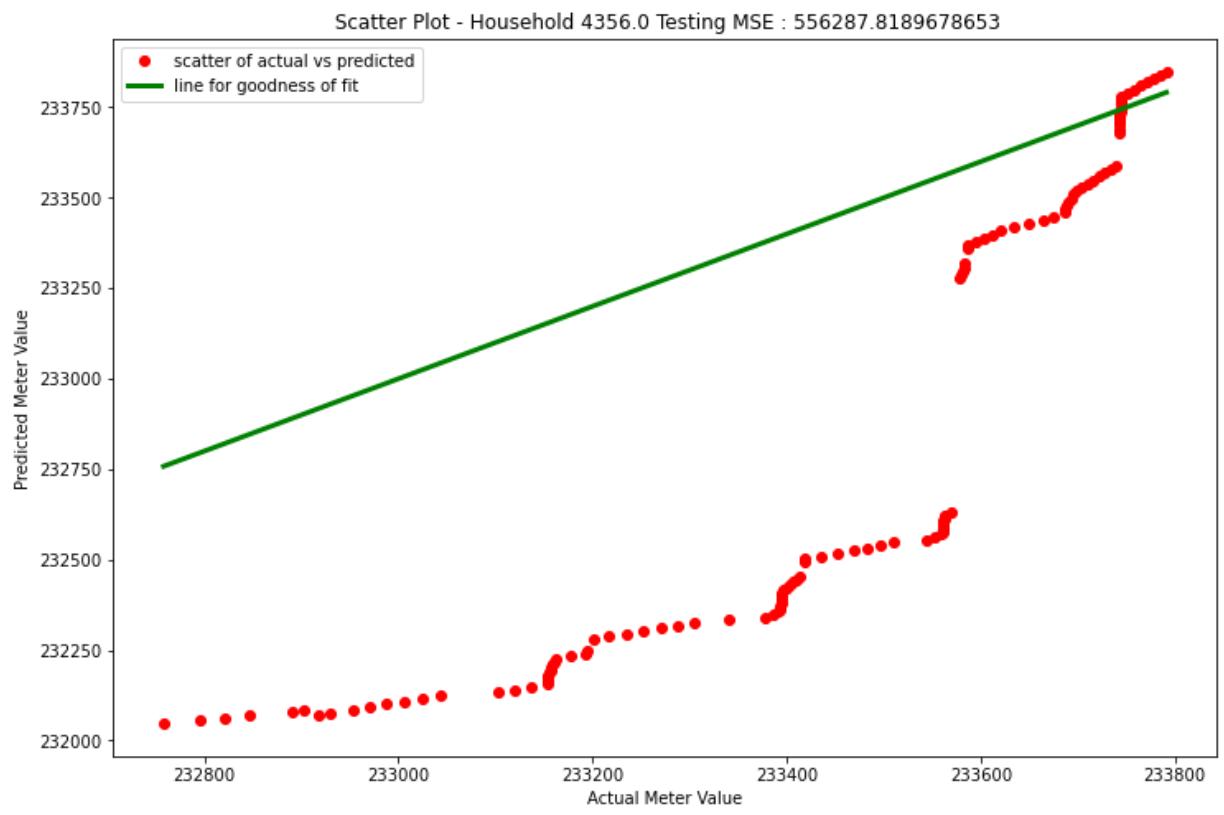




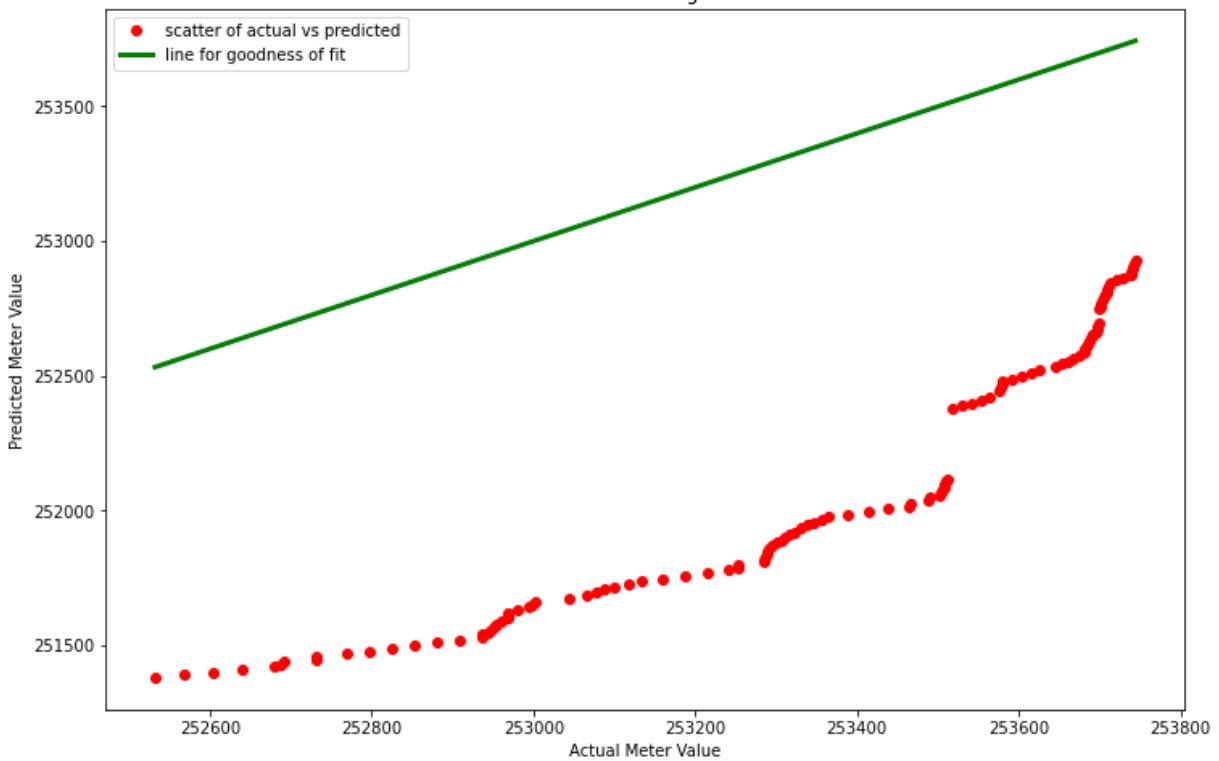




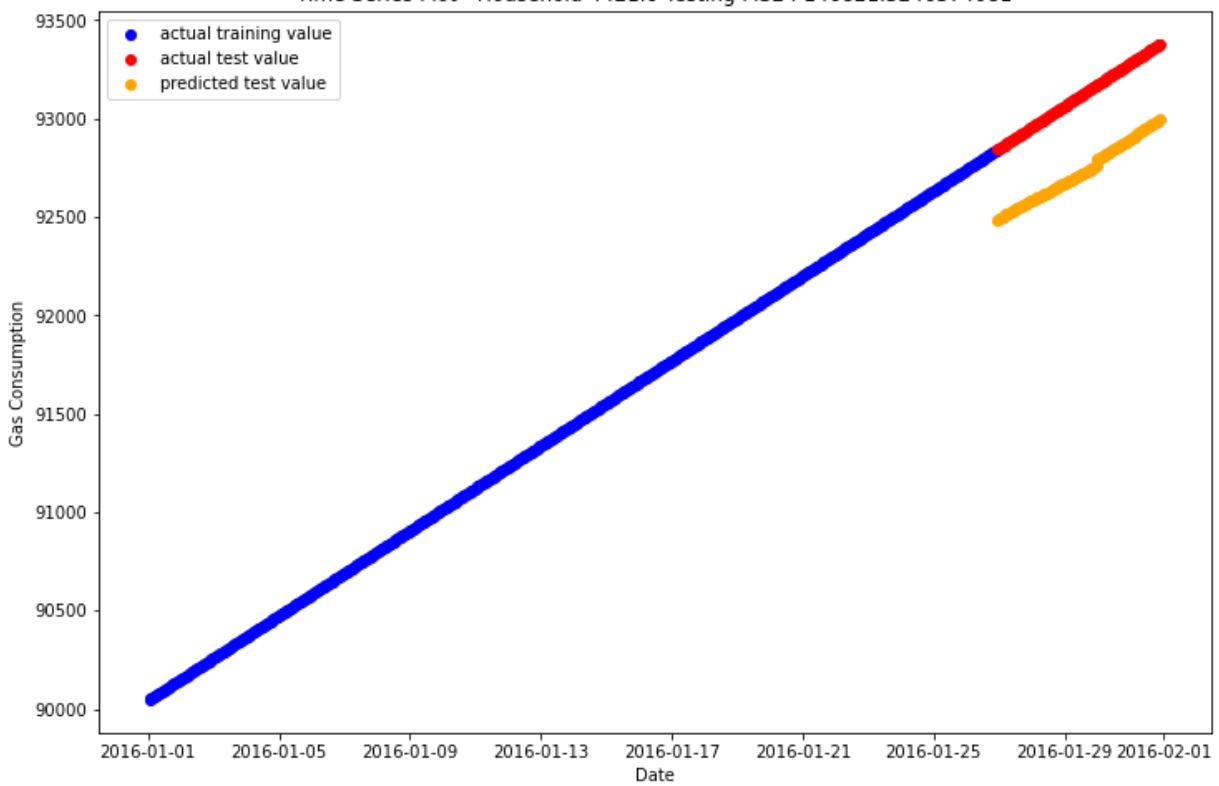


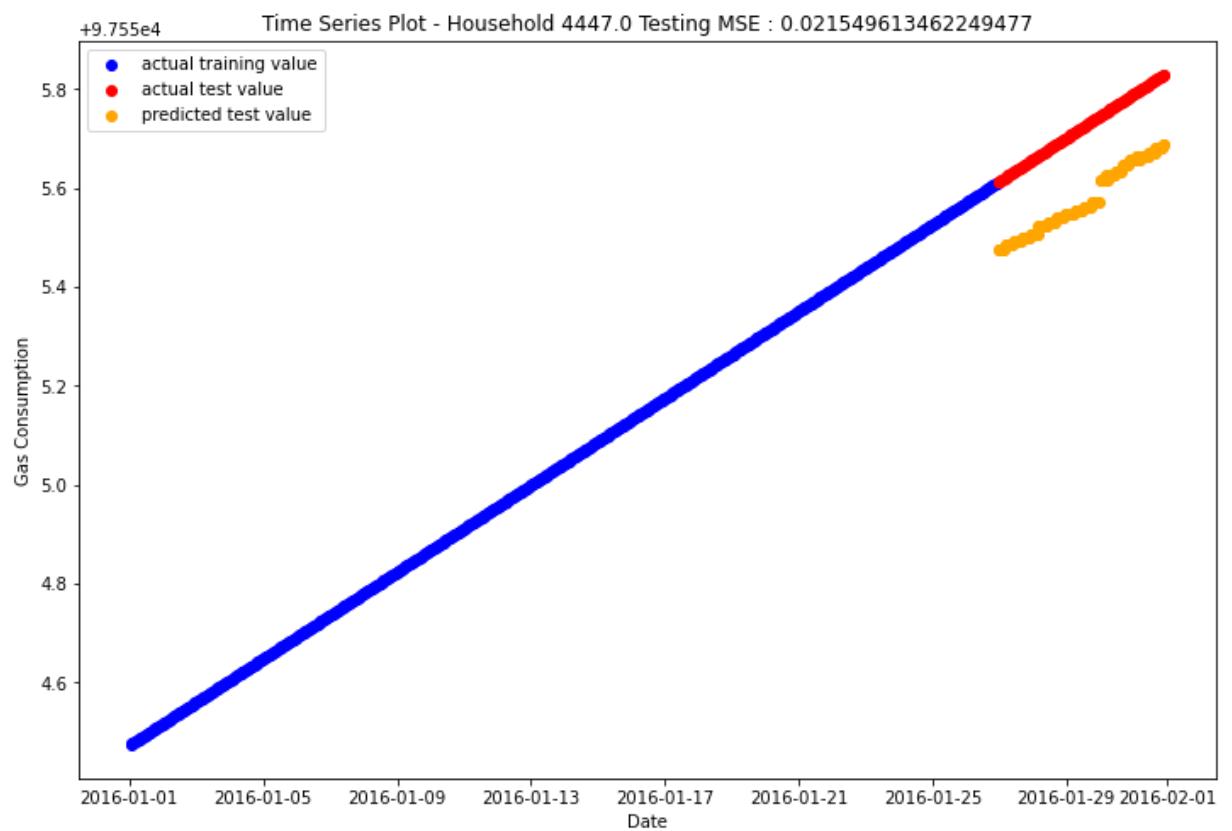
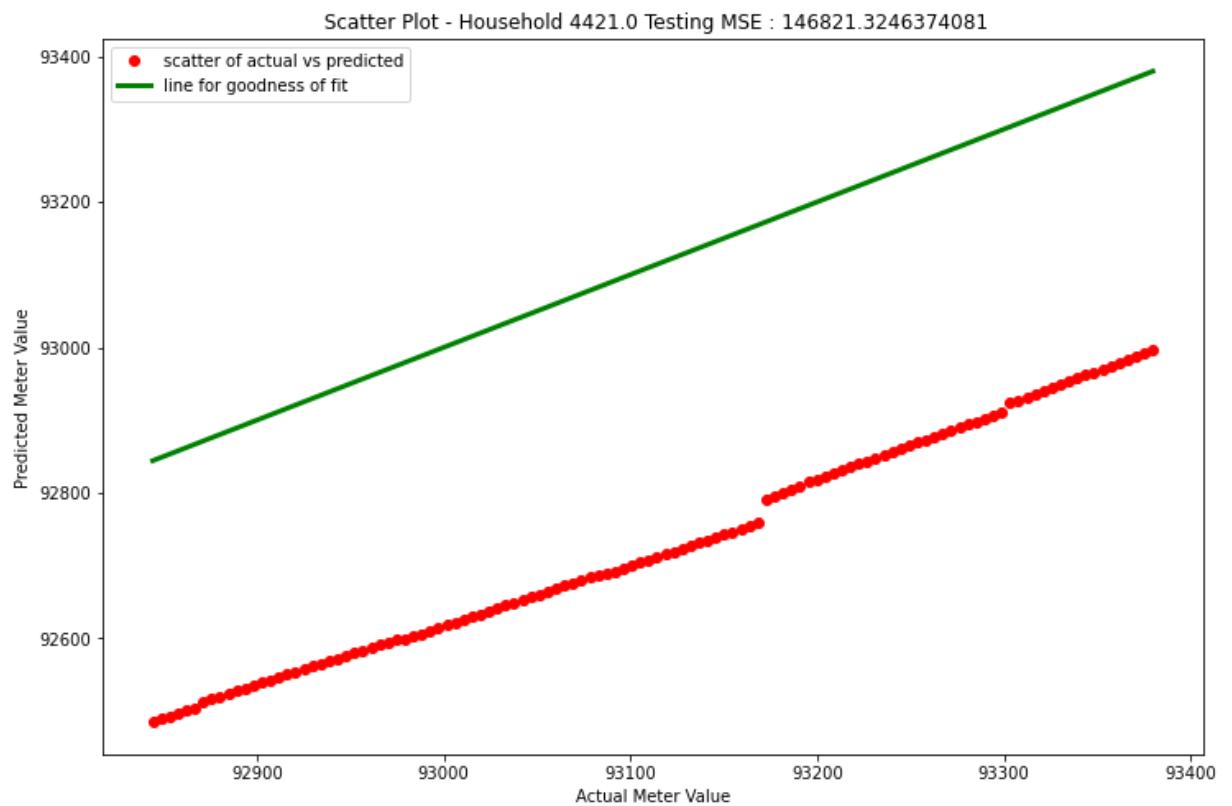


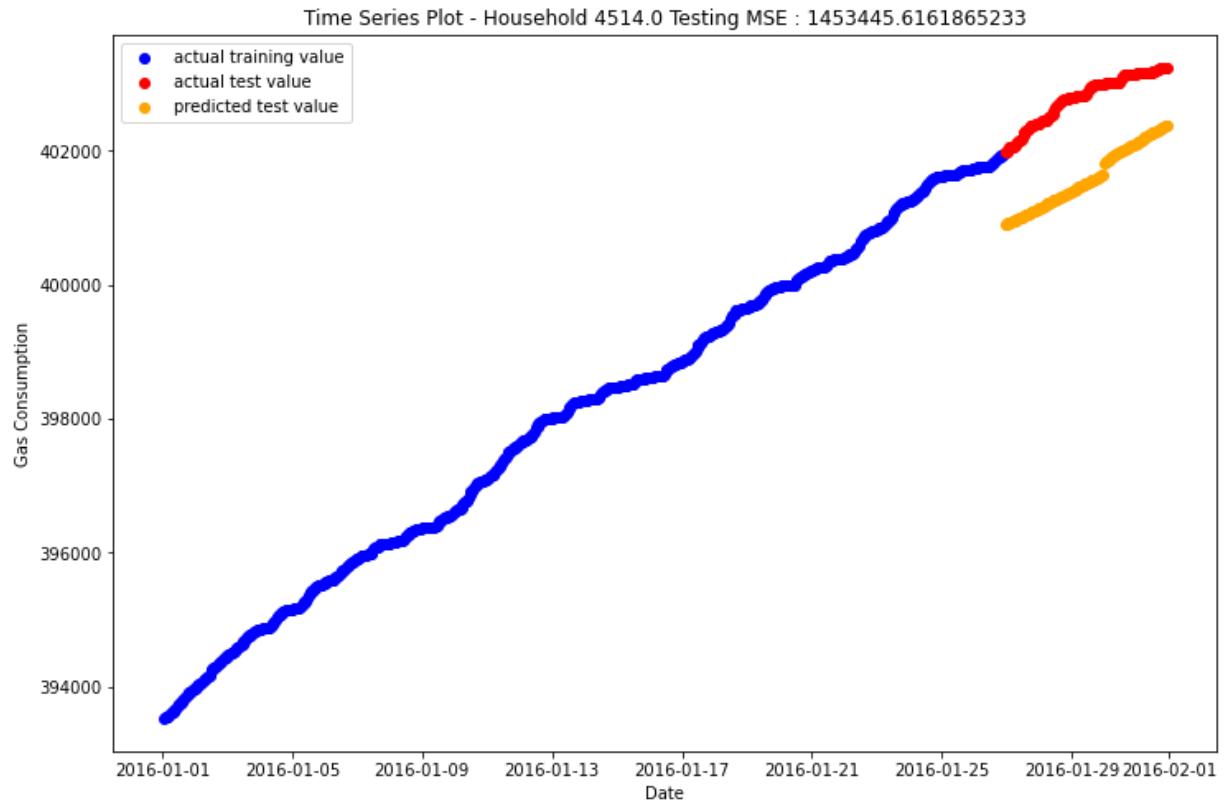
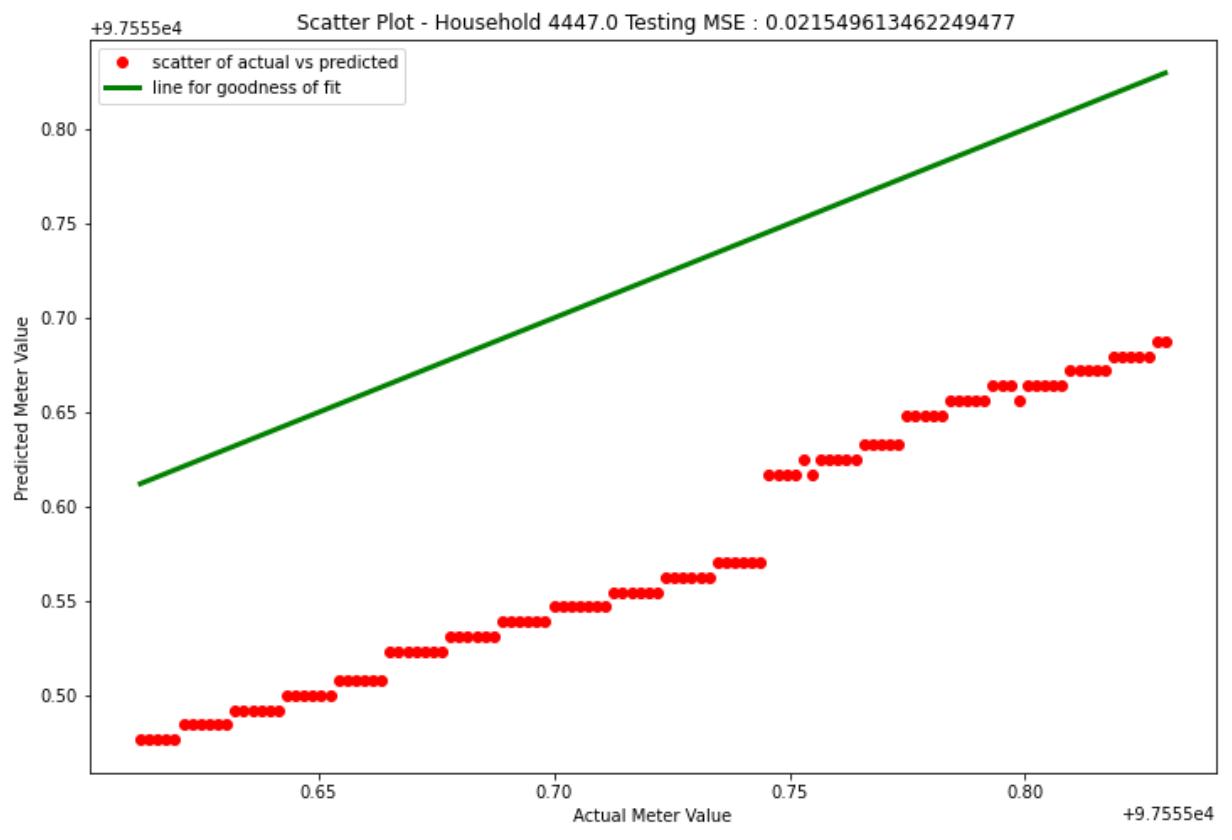
Scatter Plot - Household 4373.0 Testing MSE : 1569667.5896394749

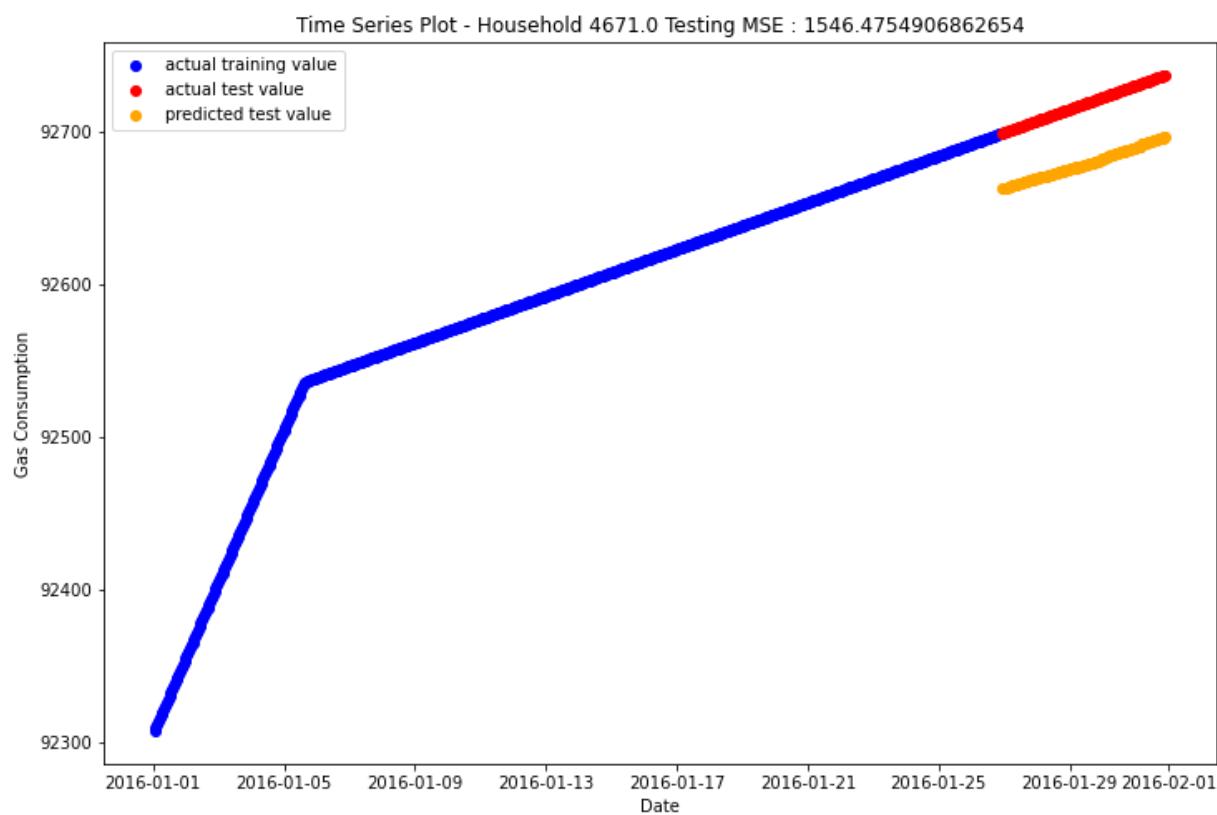
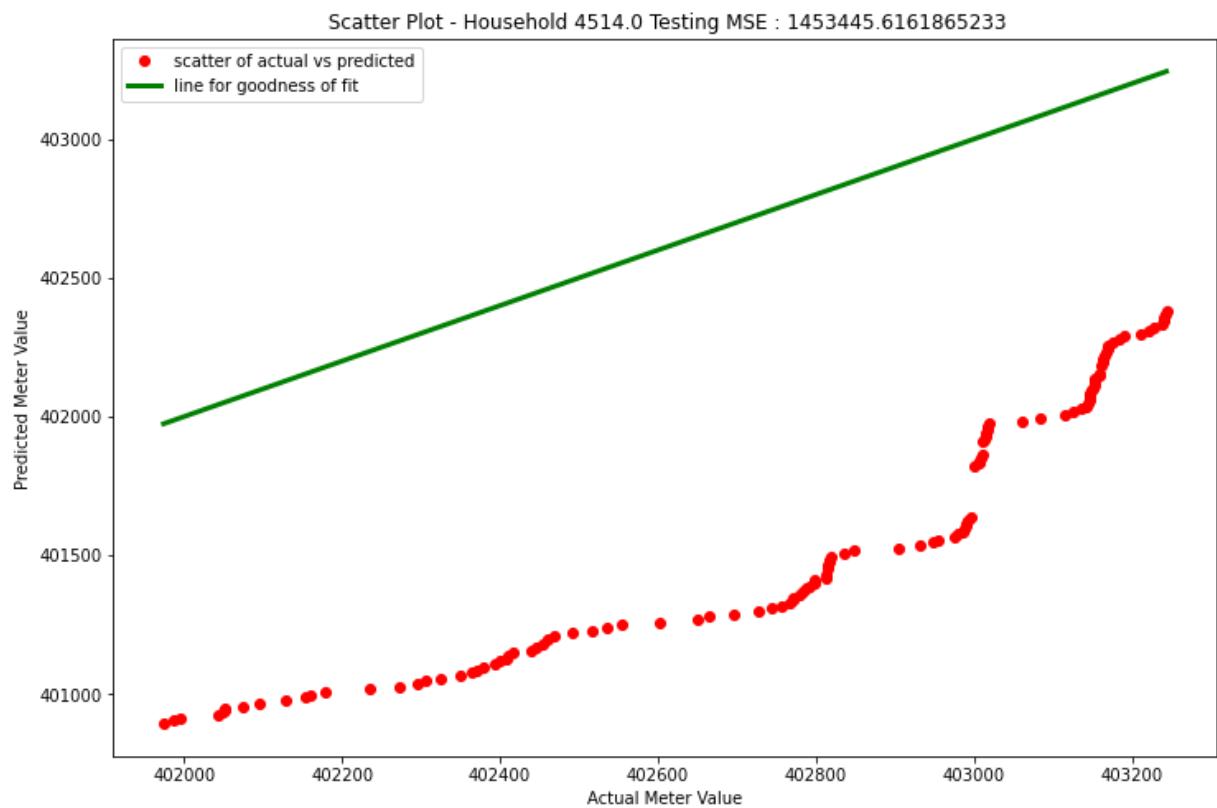


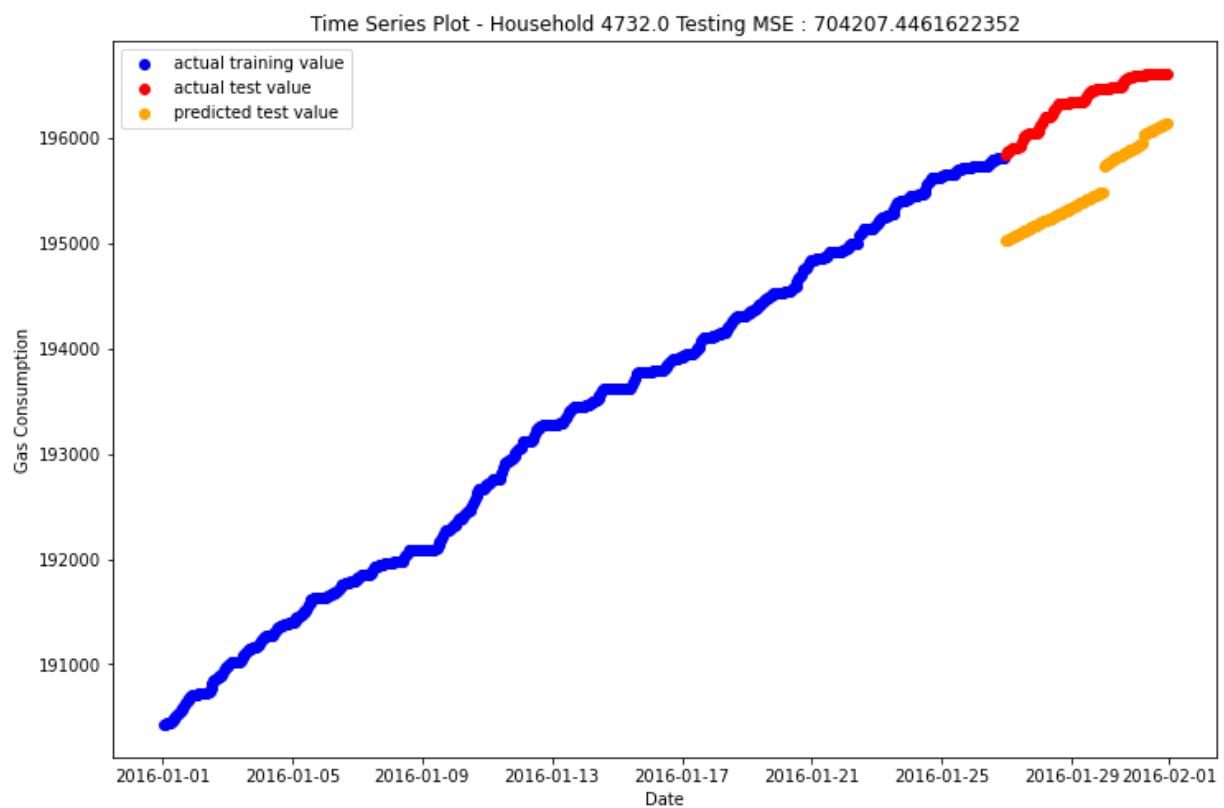
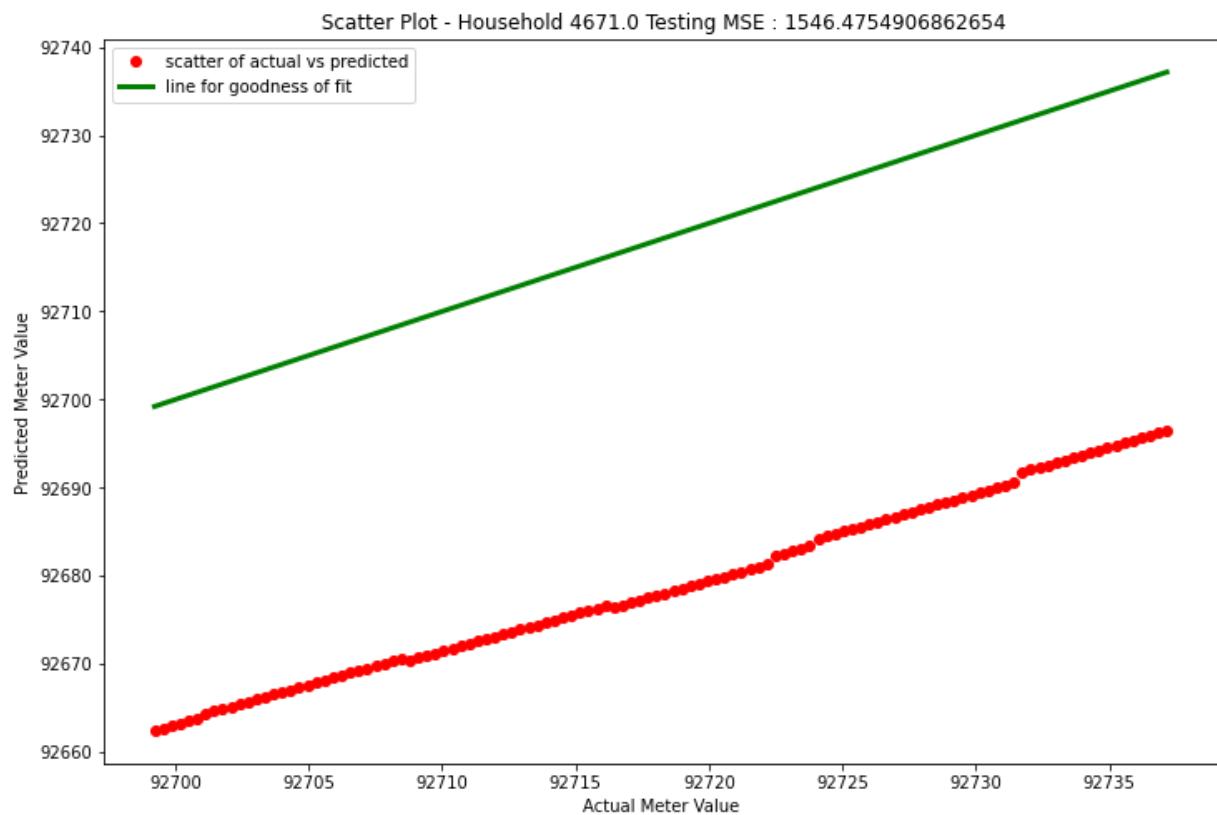
Time Series Plot - Household 4421.0 Testing MSE : 146821.3246374081

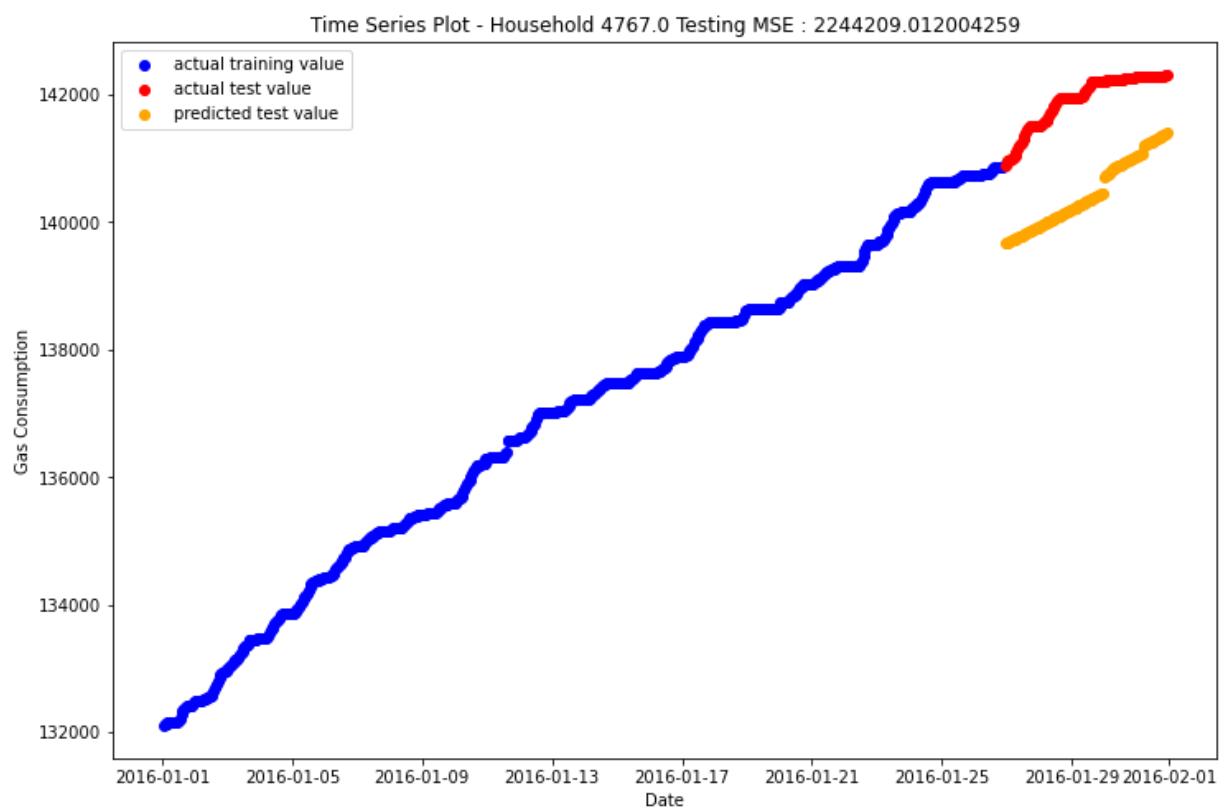
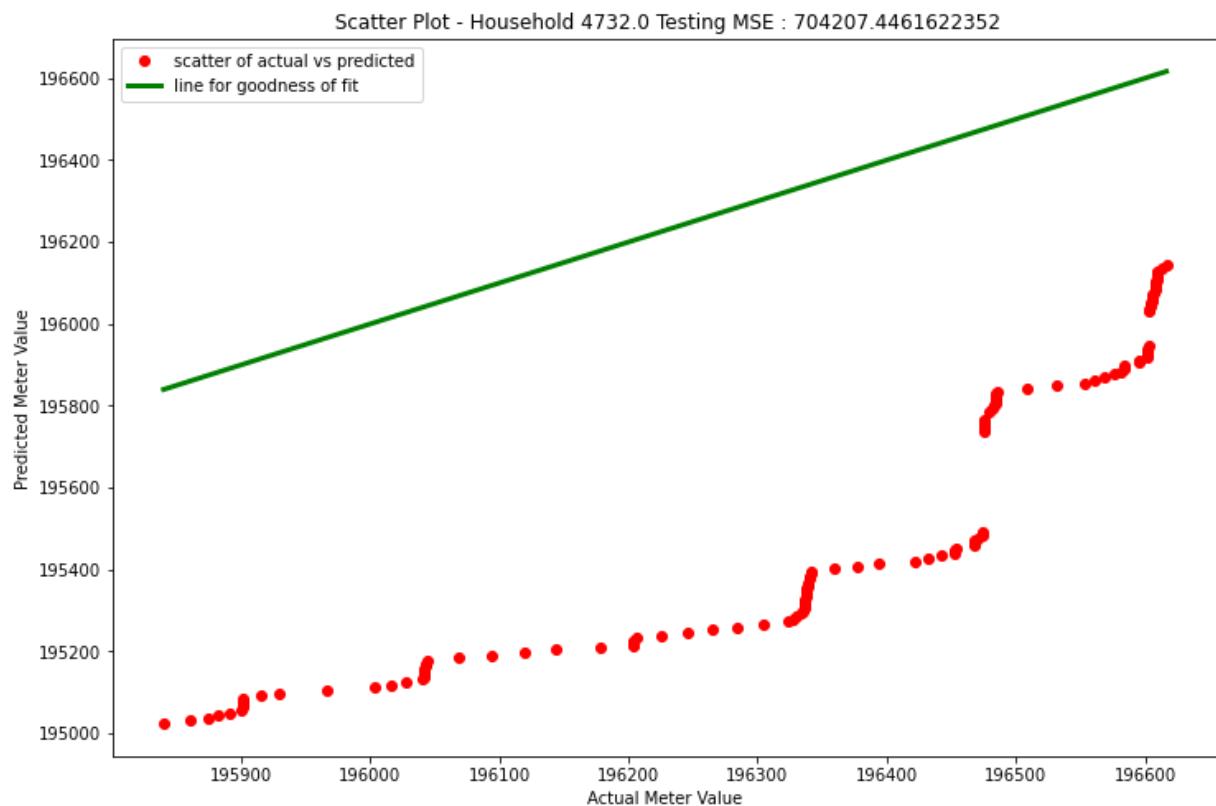




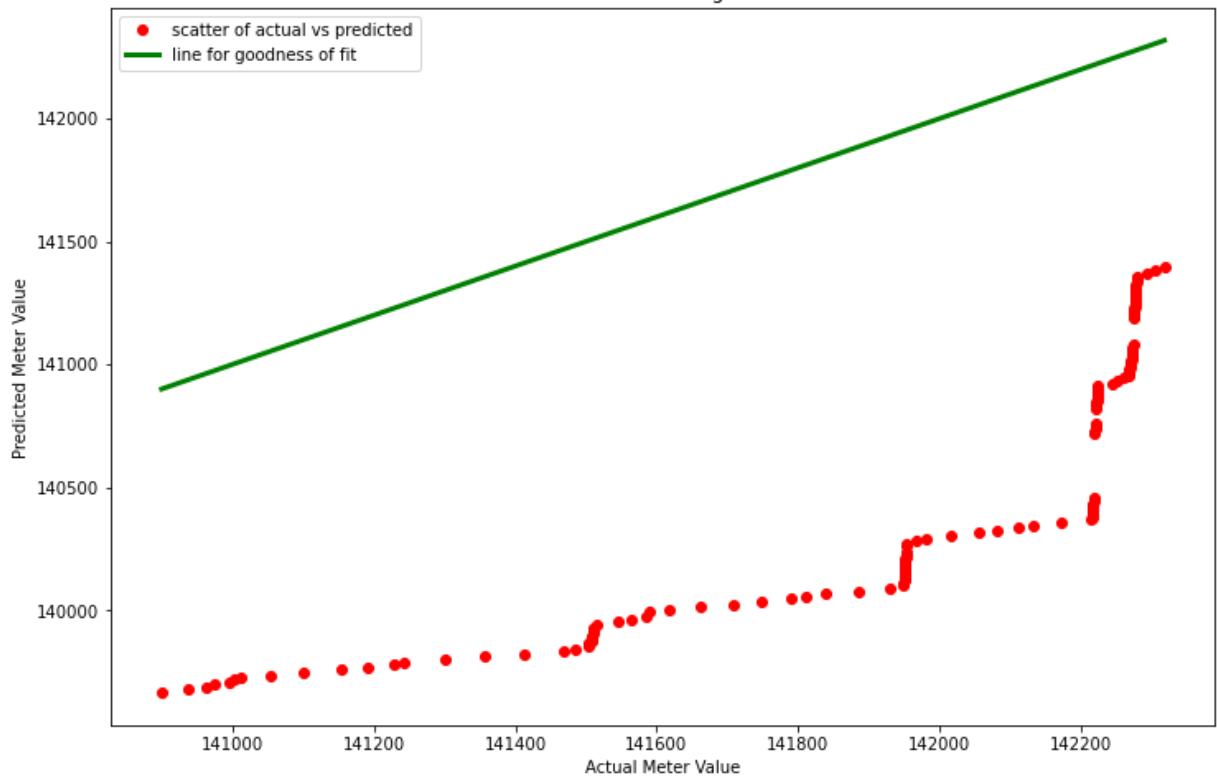




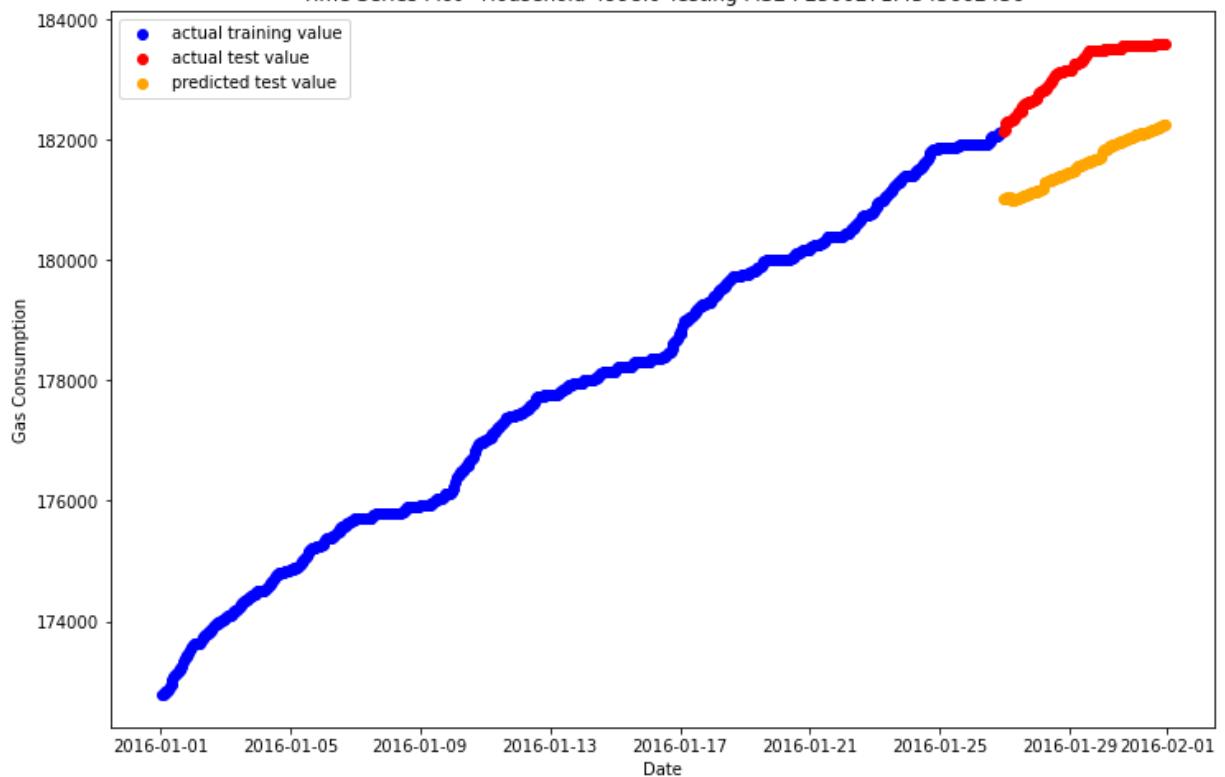


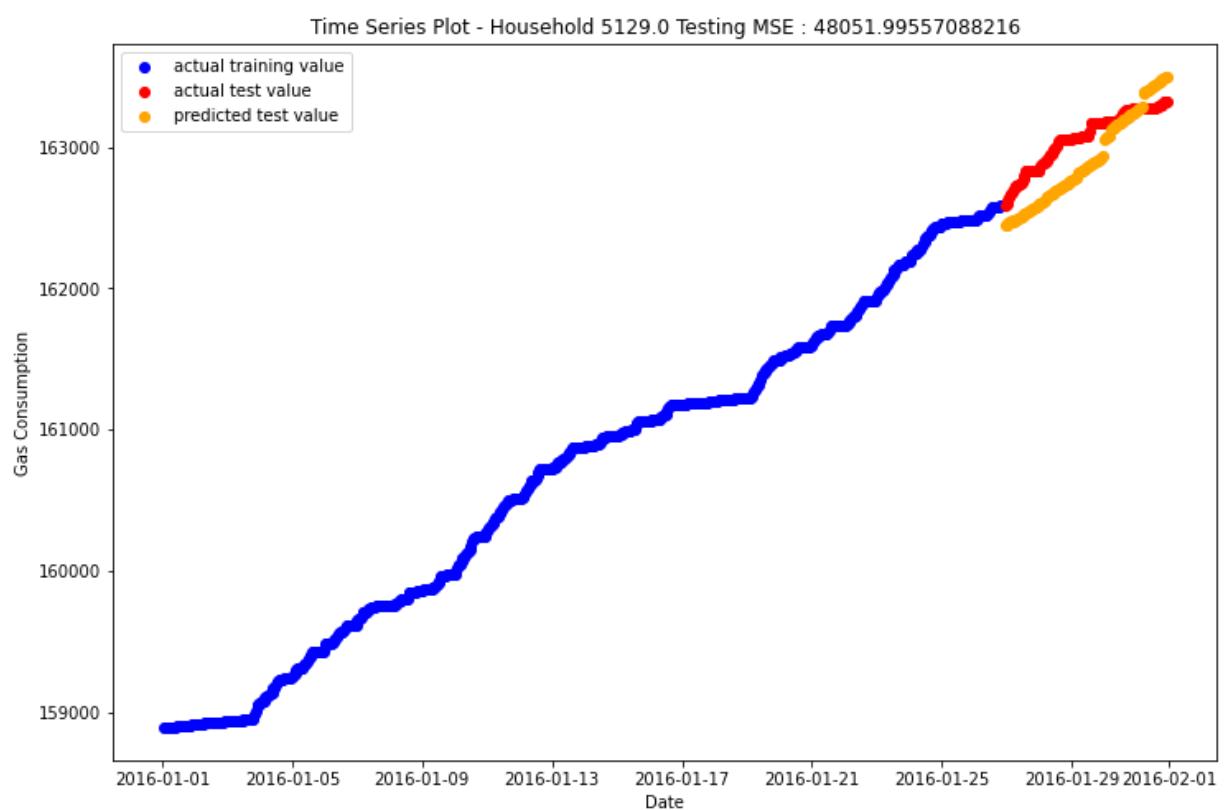
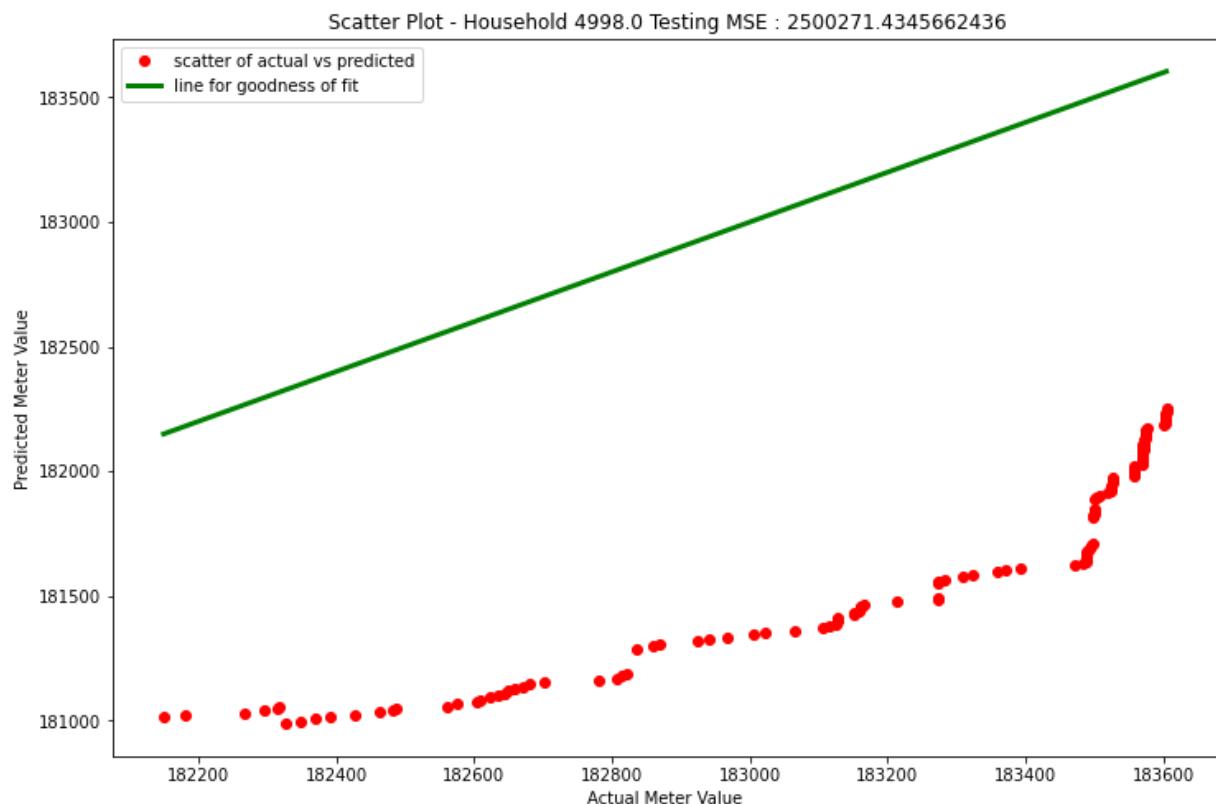


Scatter Plot - Household 4767.0 Testing MSE : 2244209.012004259

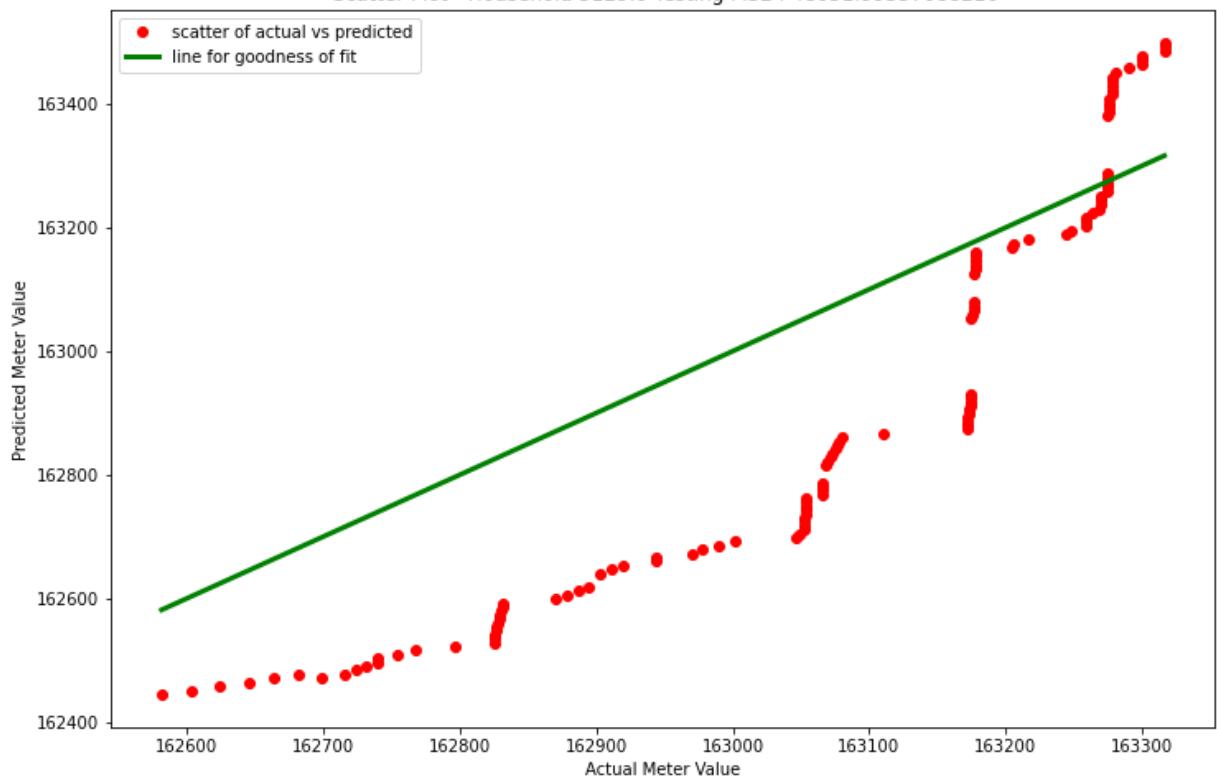


Time Series Plot - Household 4998.0 Testing MSE : 2500271.4345662436

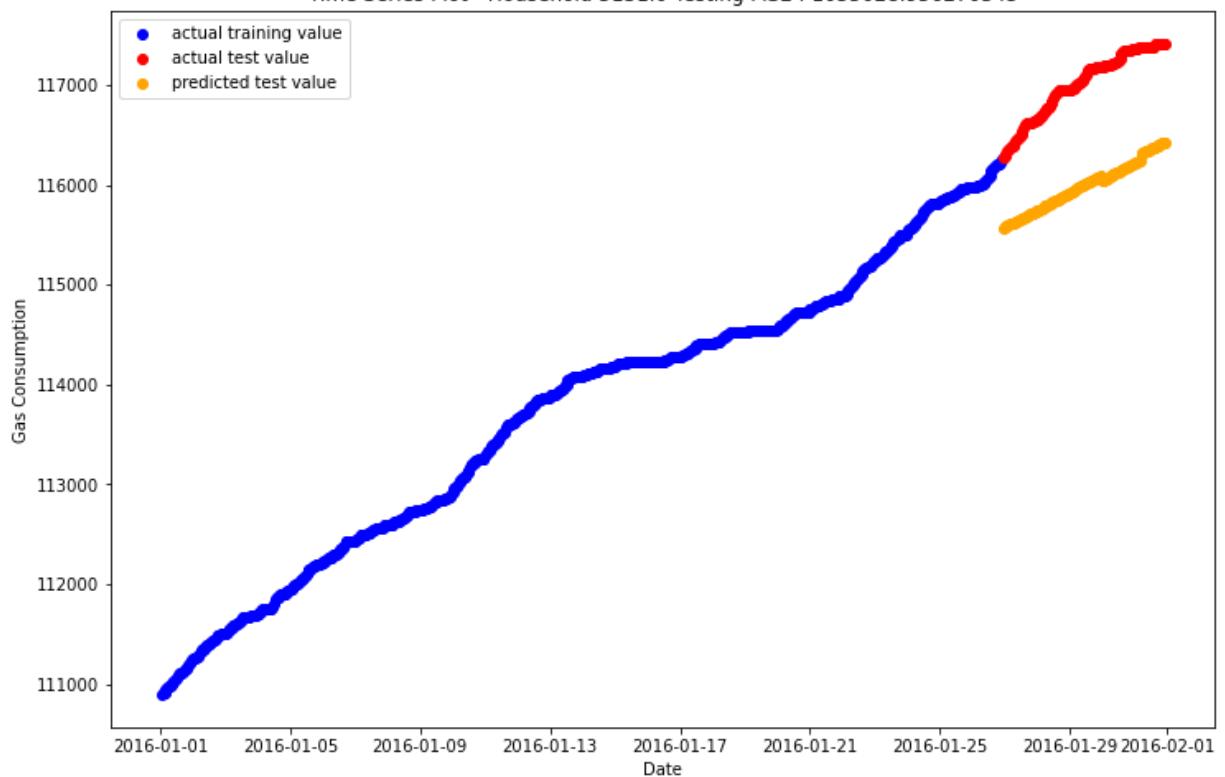


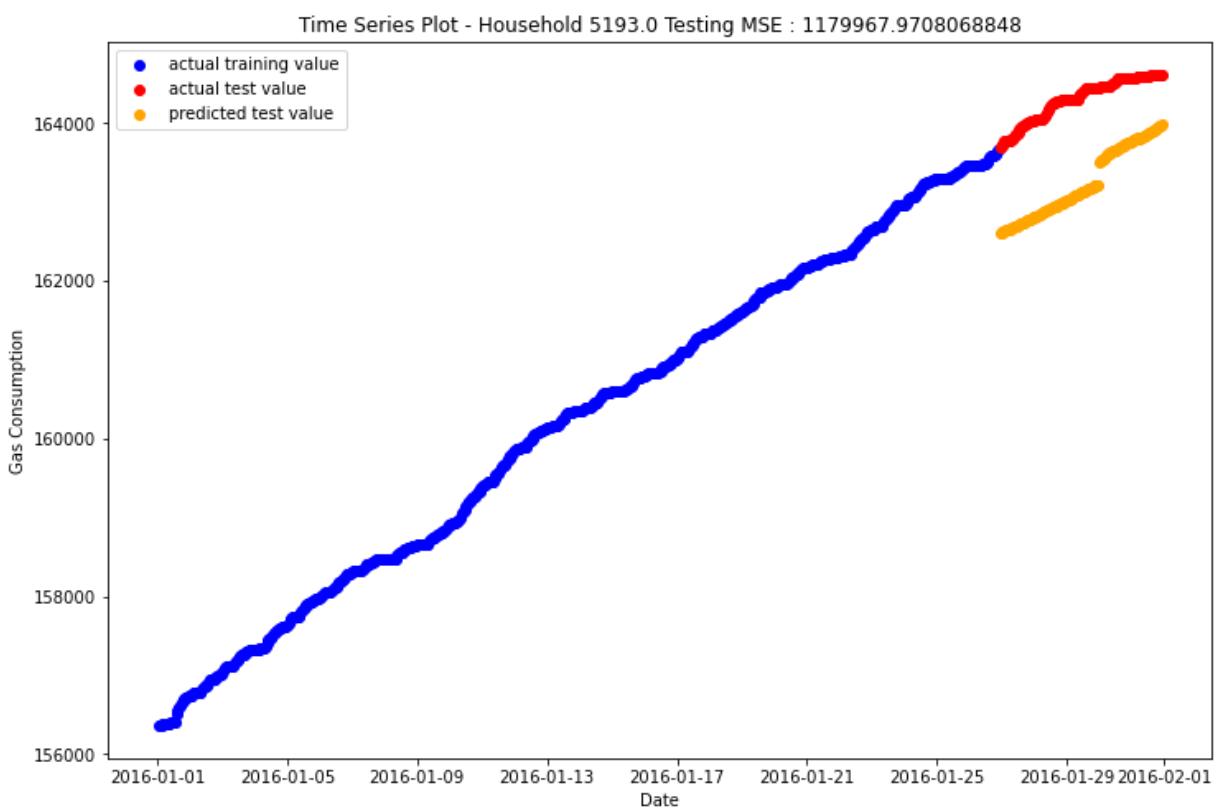
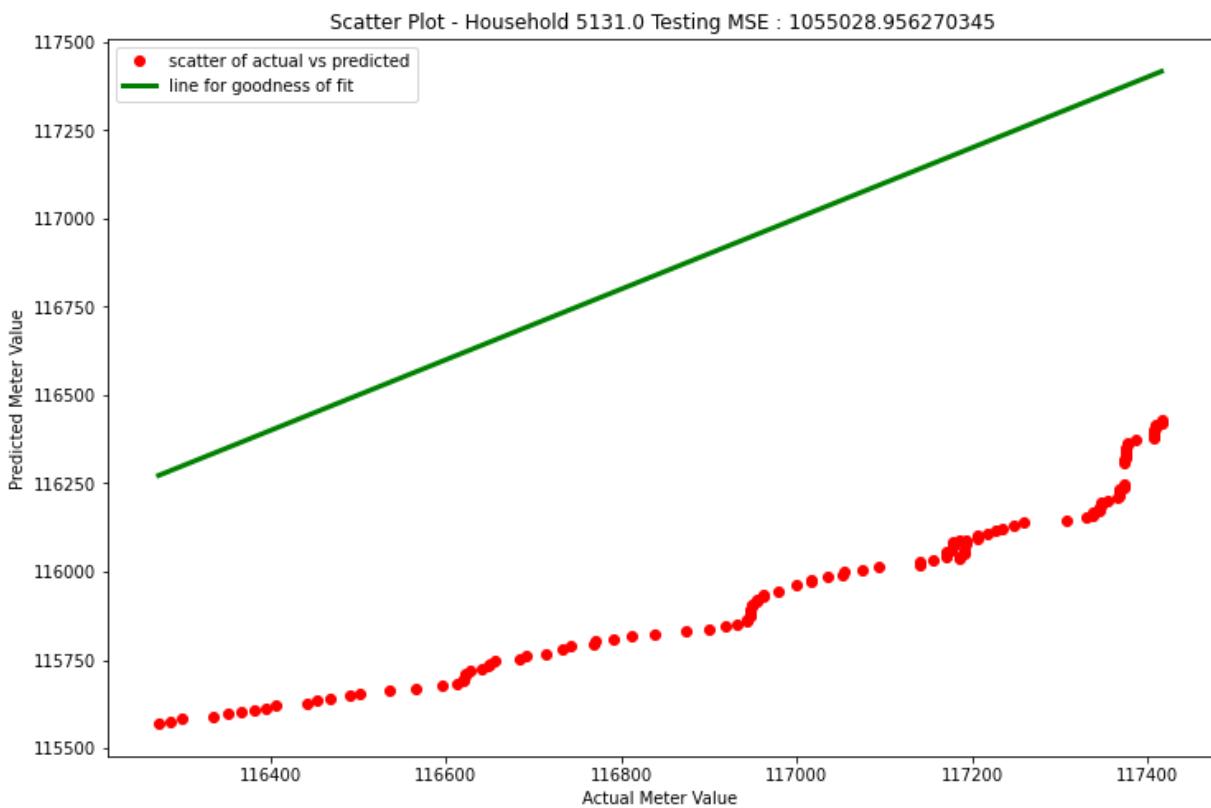


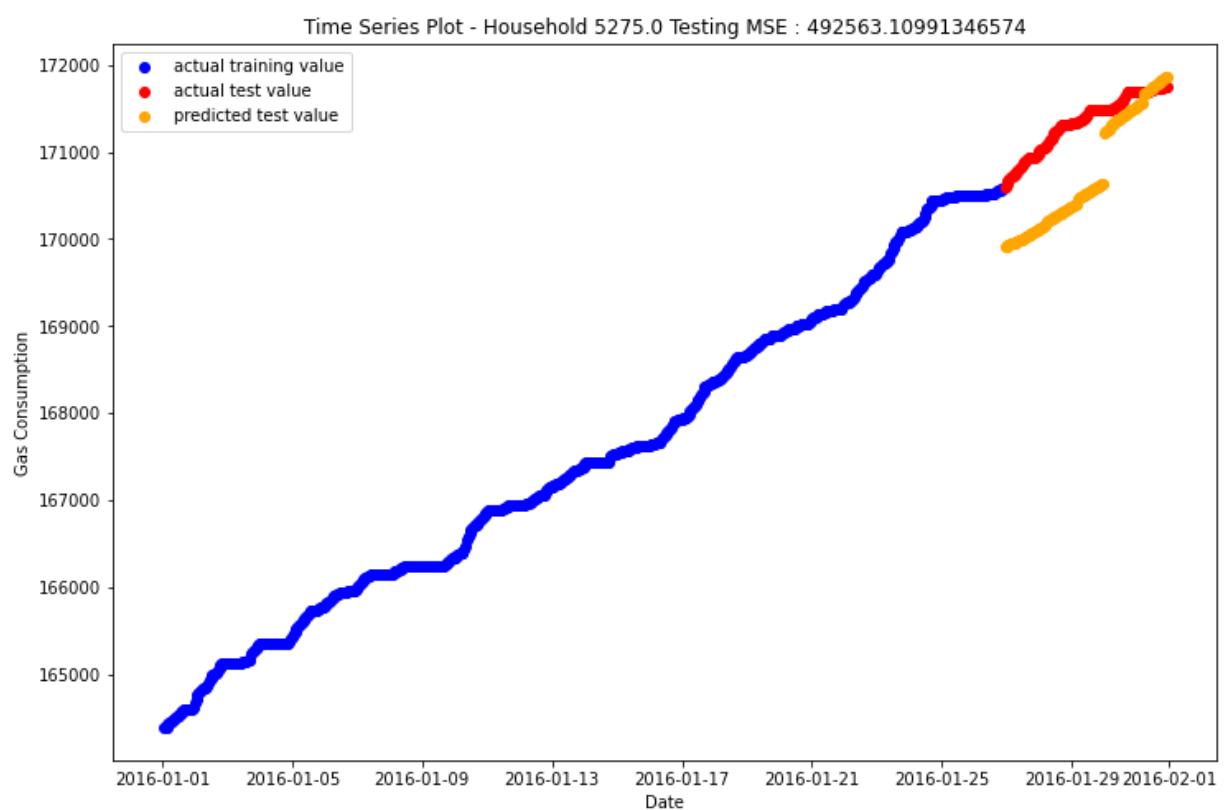
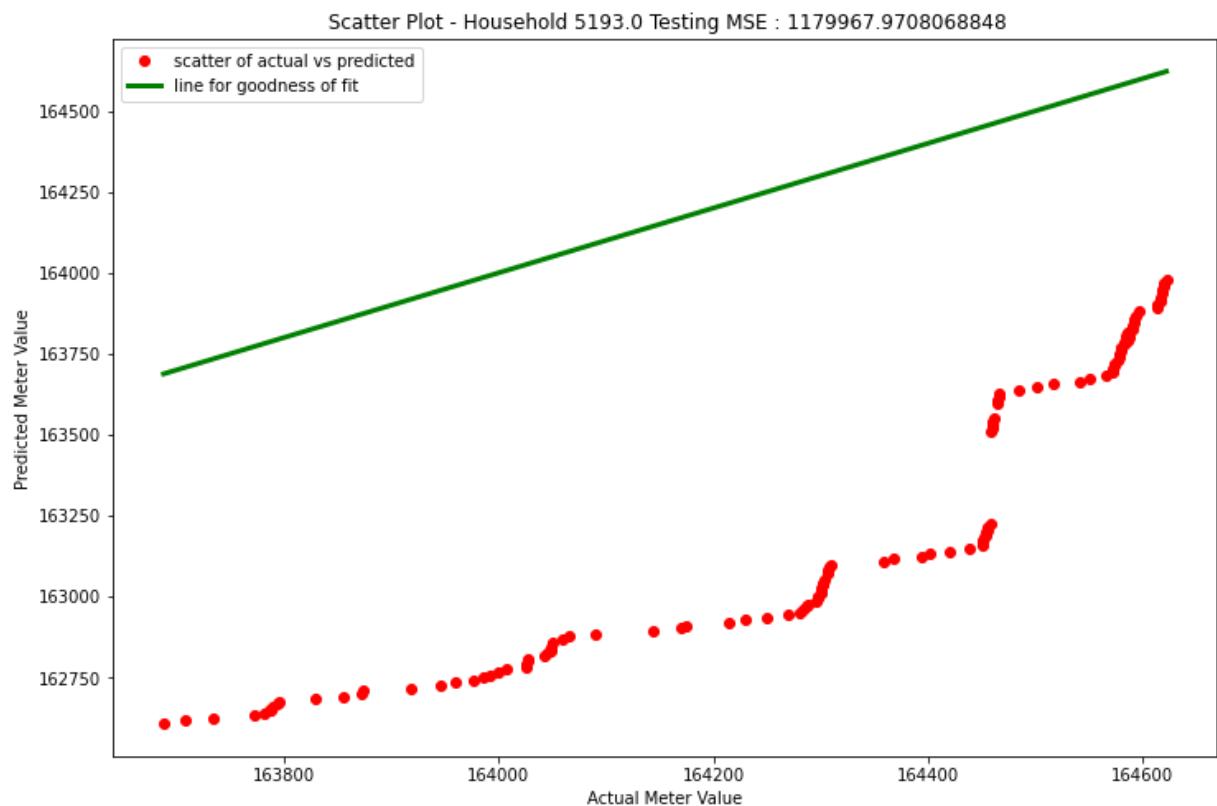
Scatter Plot - Household 5129.0 Testing MSE : 48051.99557088216



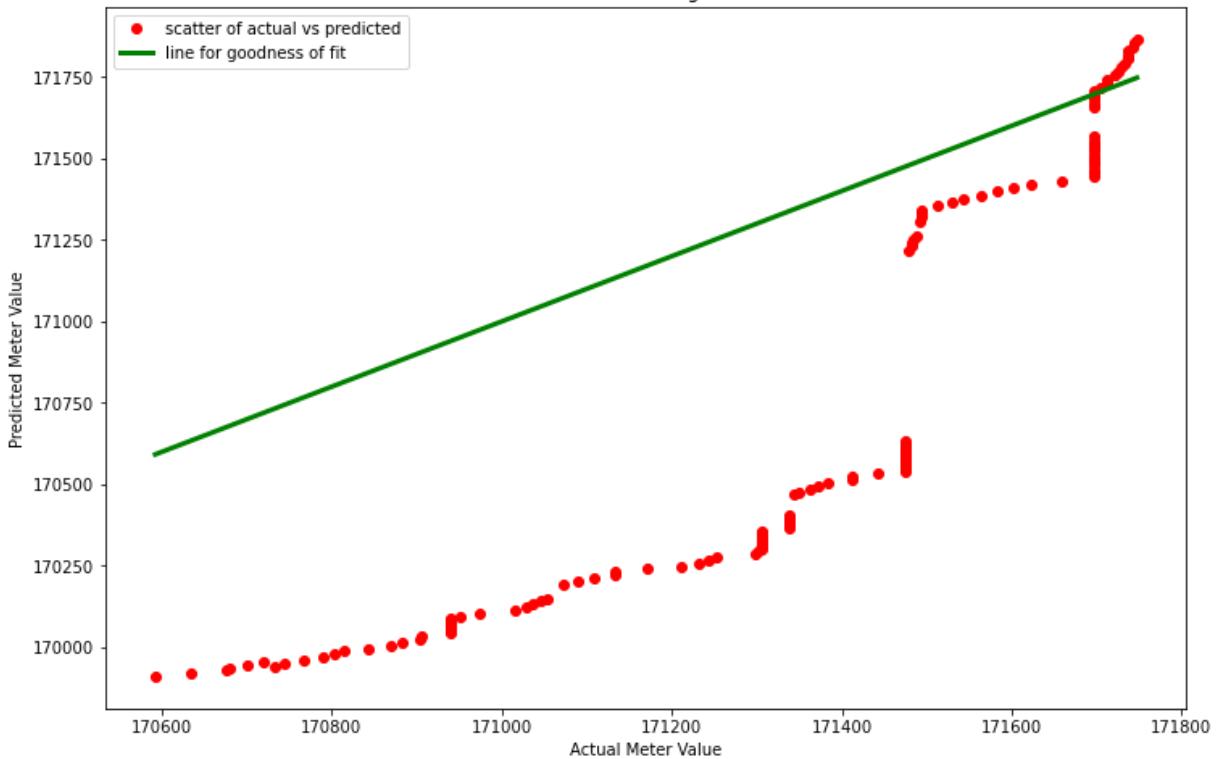
Time Series Plot - Household 5131.0 Testing MSE : 1055028.956270345



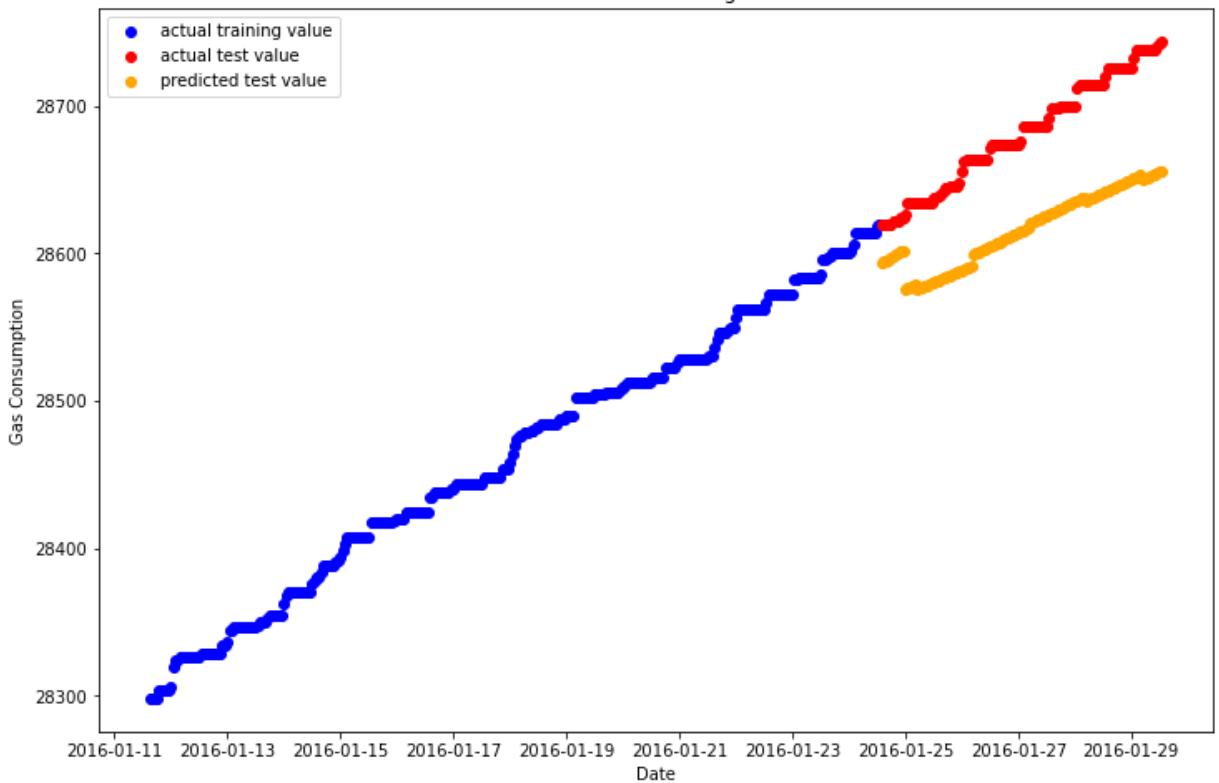


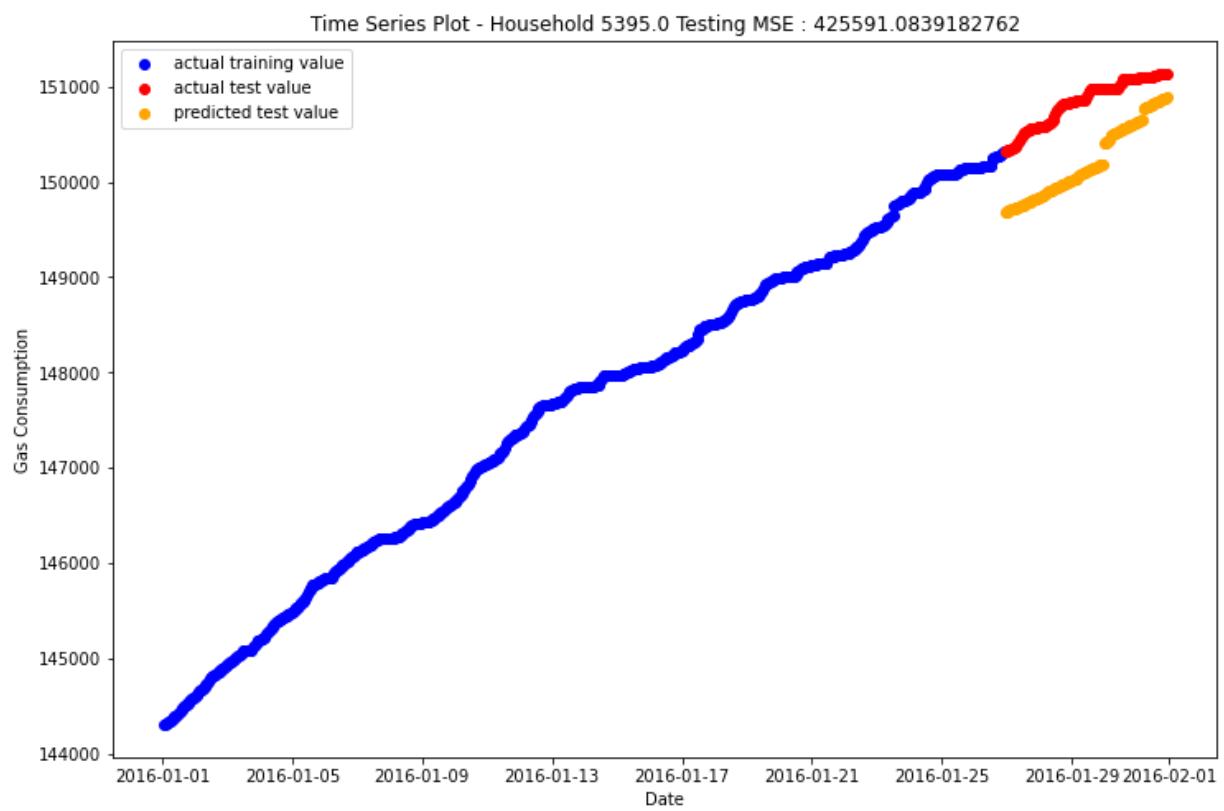
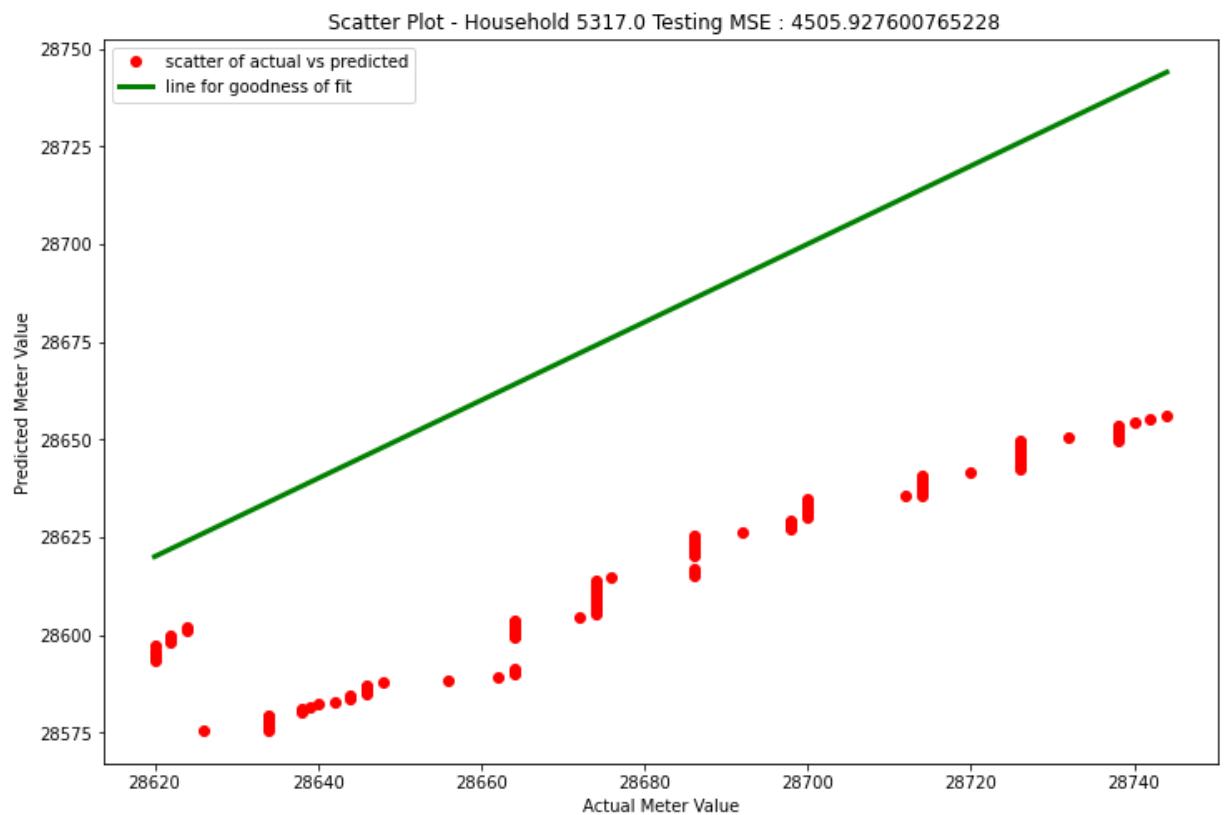


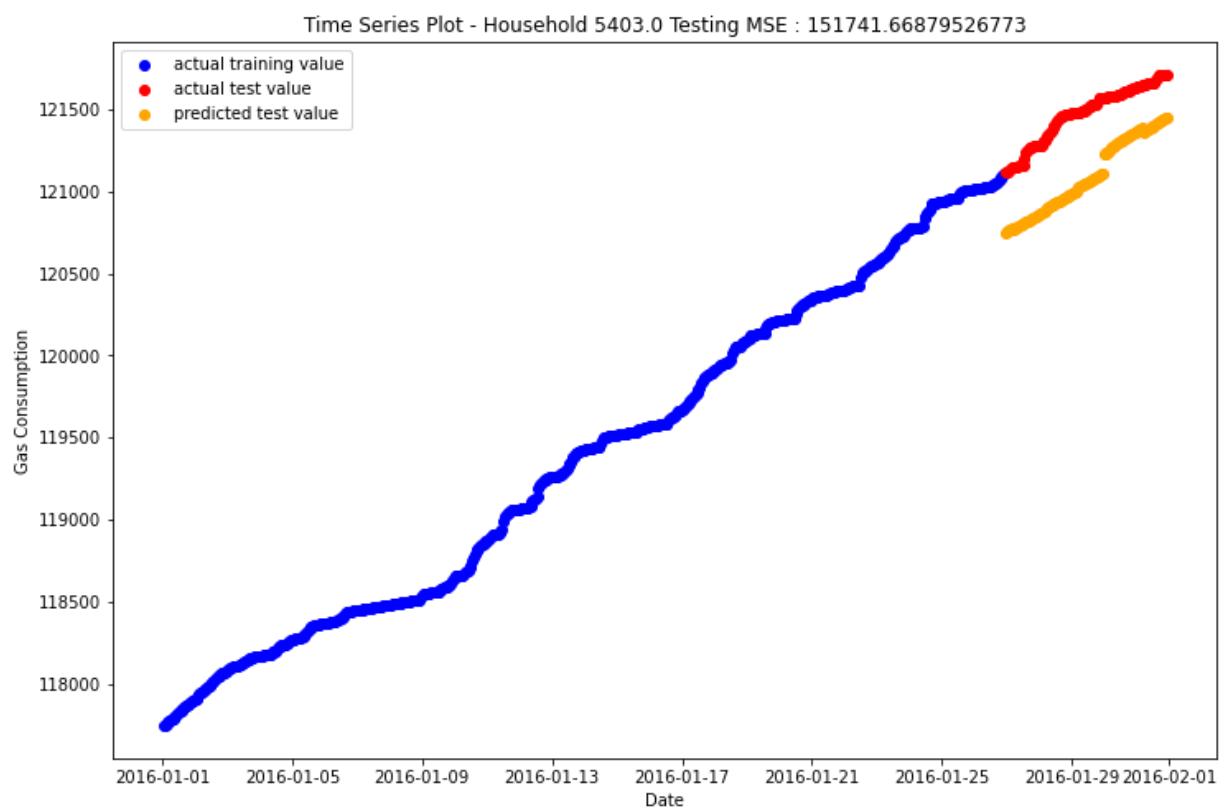
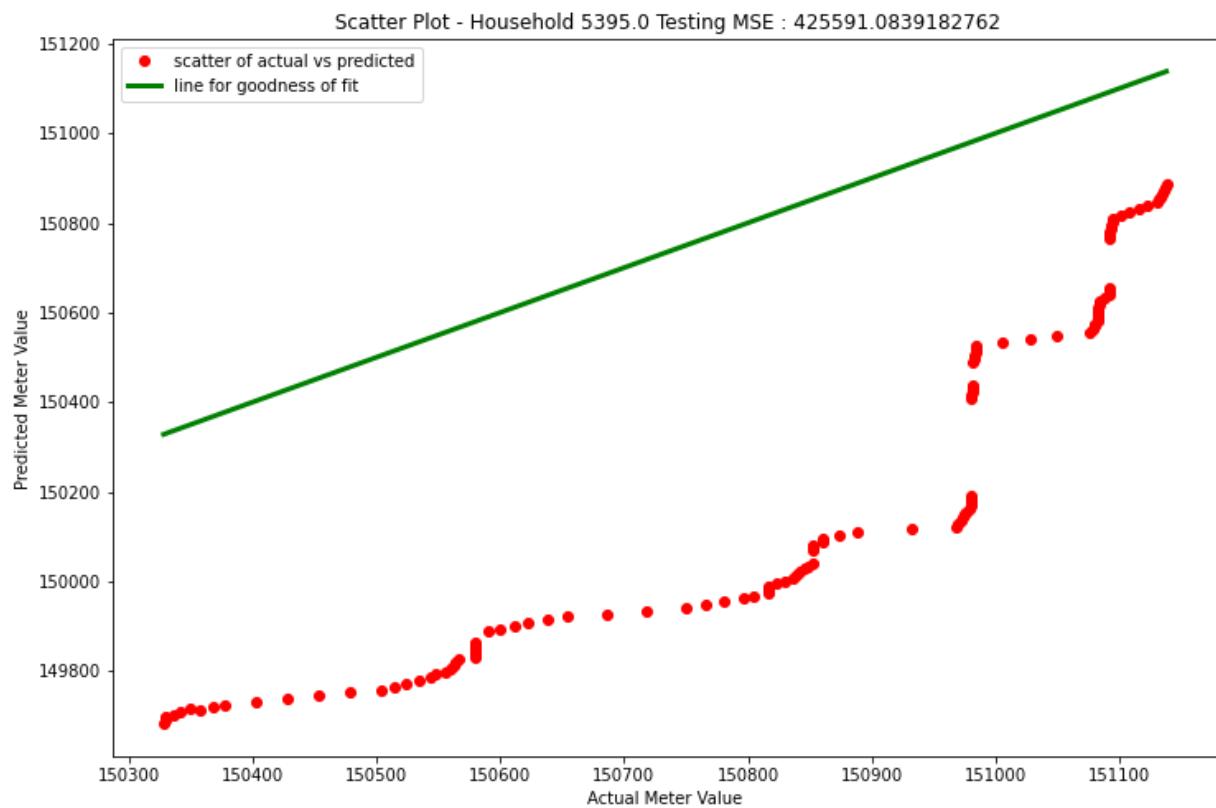
Scatter Plot - Household 5275.0 Testing MSE : 492563.10991346574

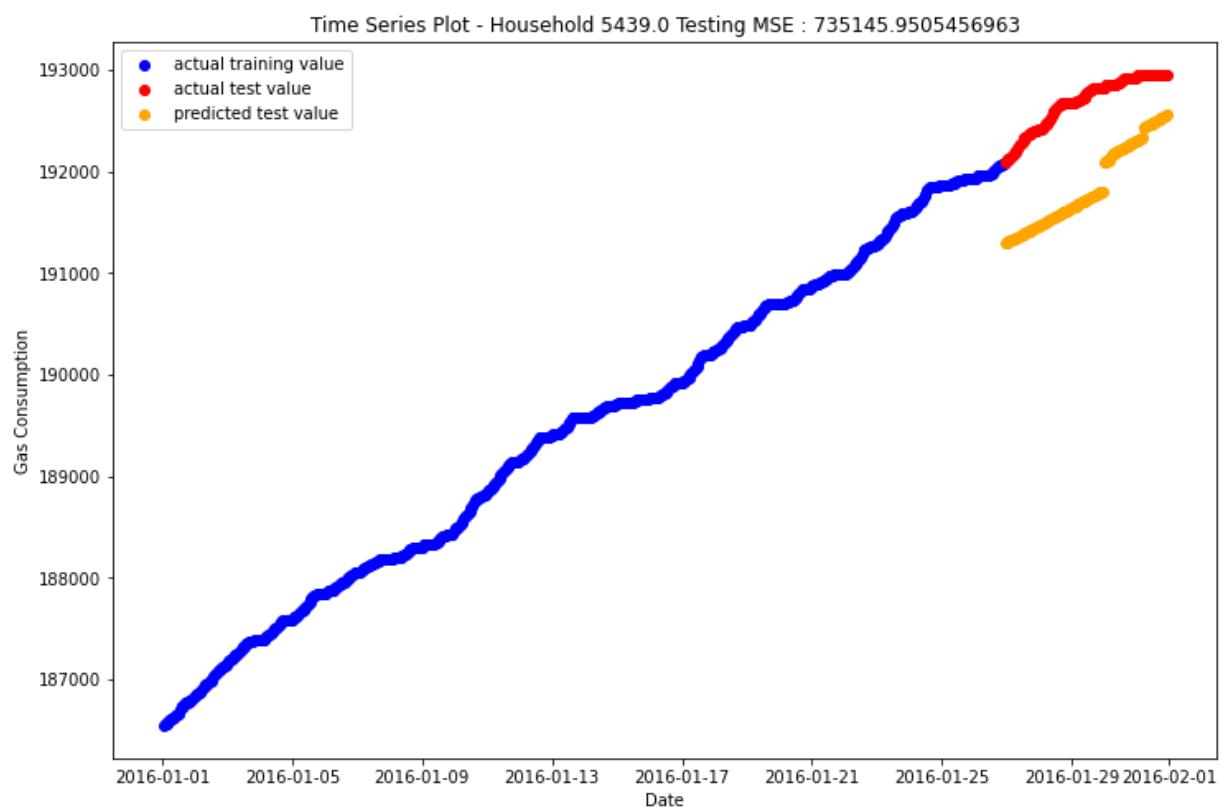
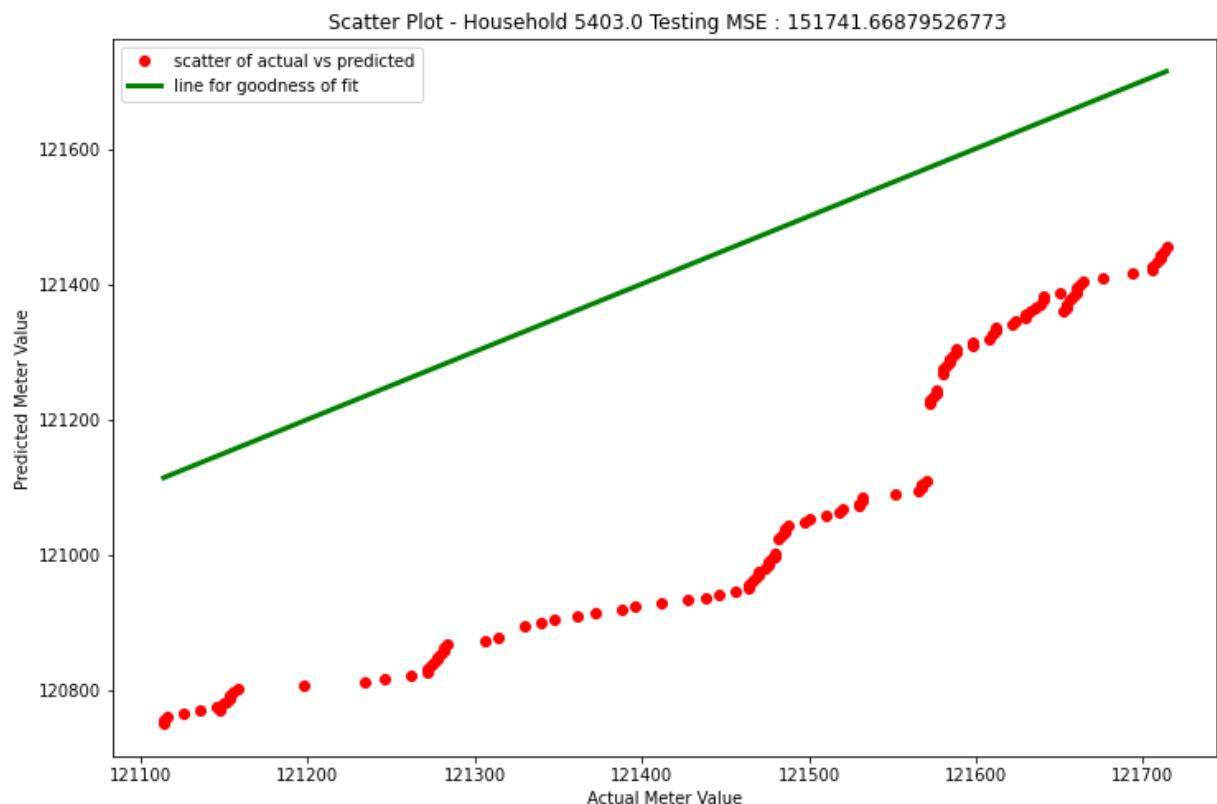


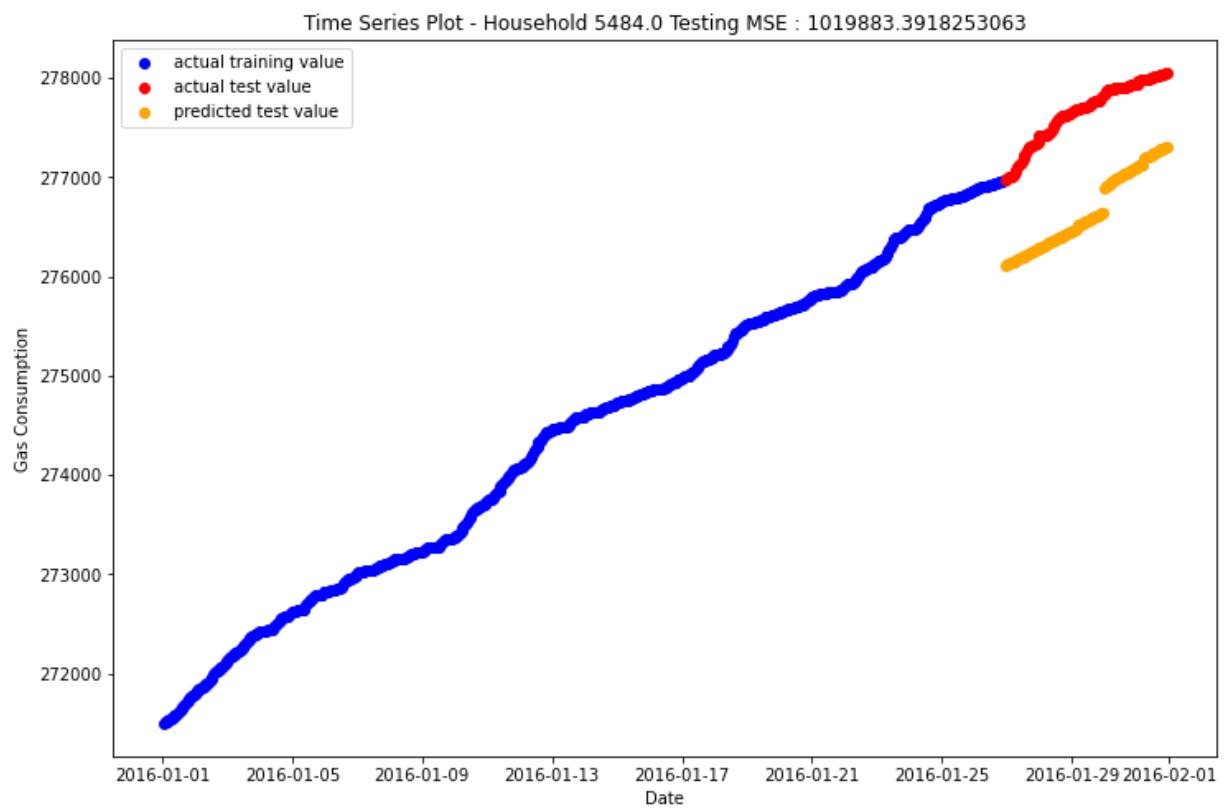
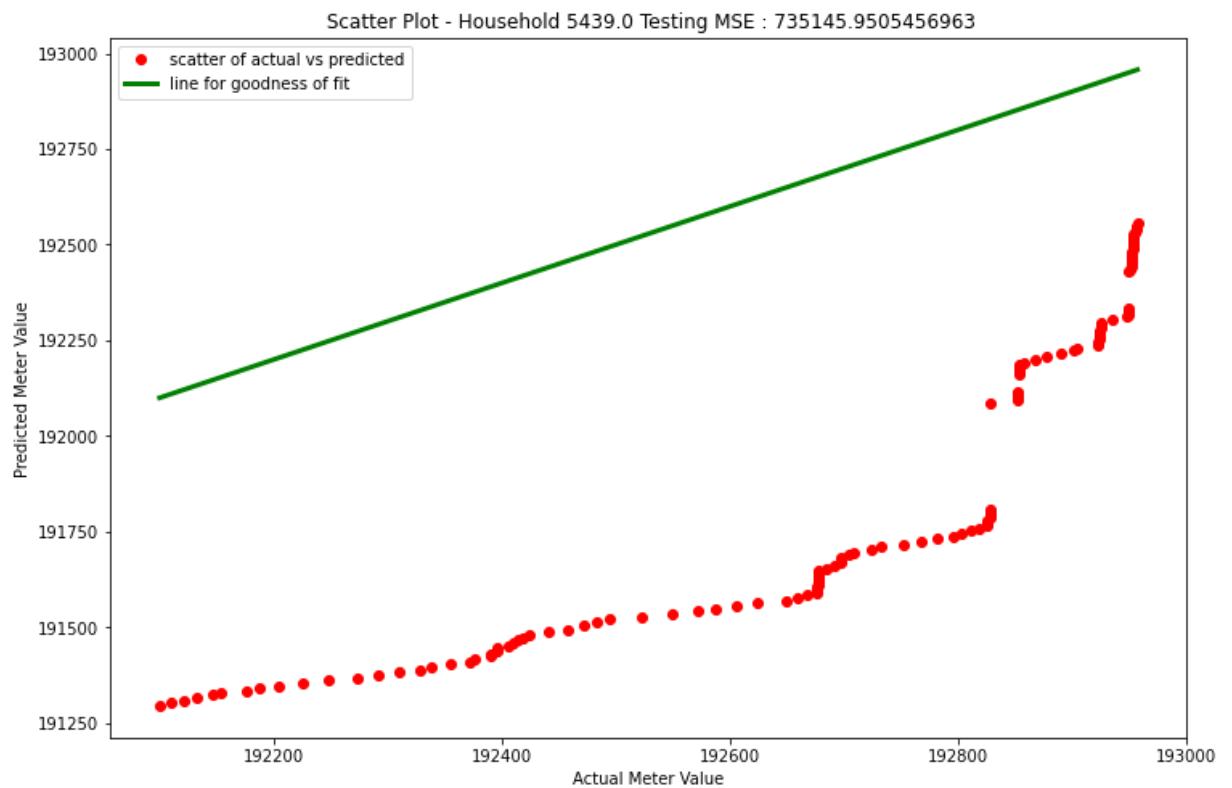
Time Series Plot - Household 5317.0 Testing MSE : 4505.927600765228

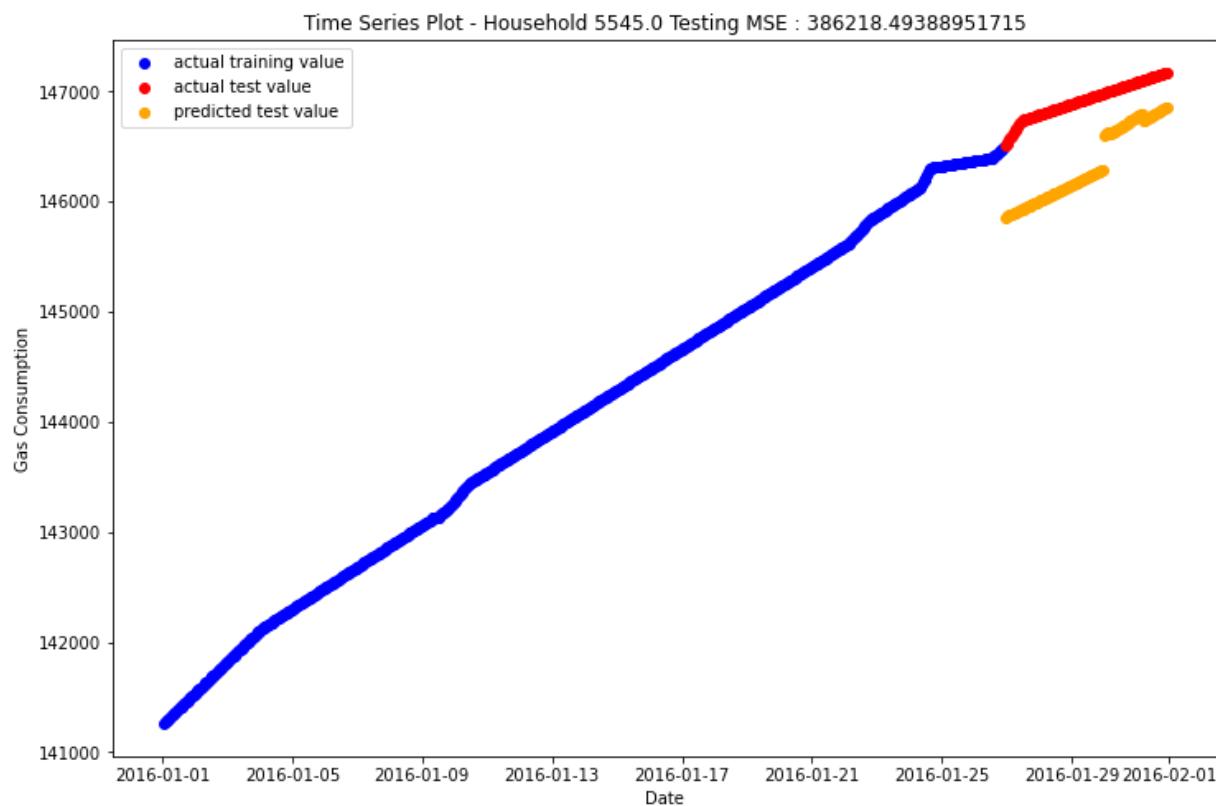
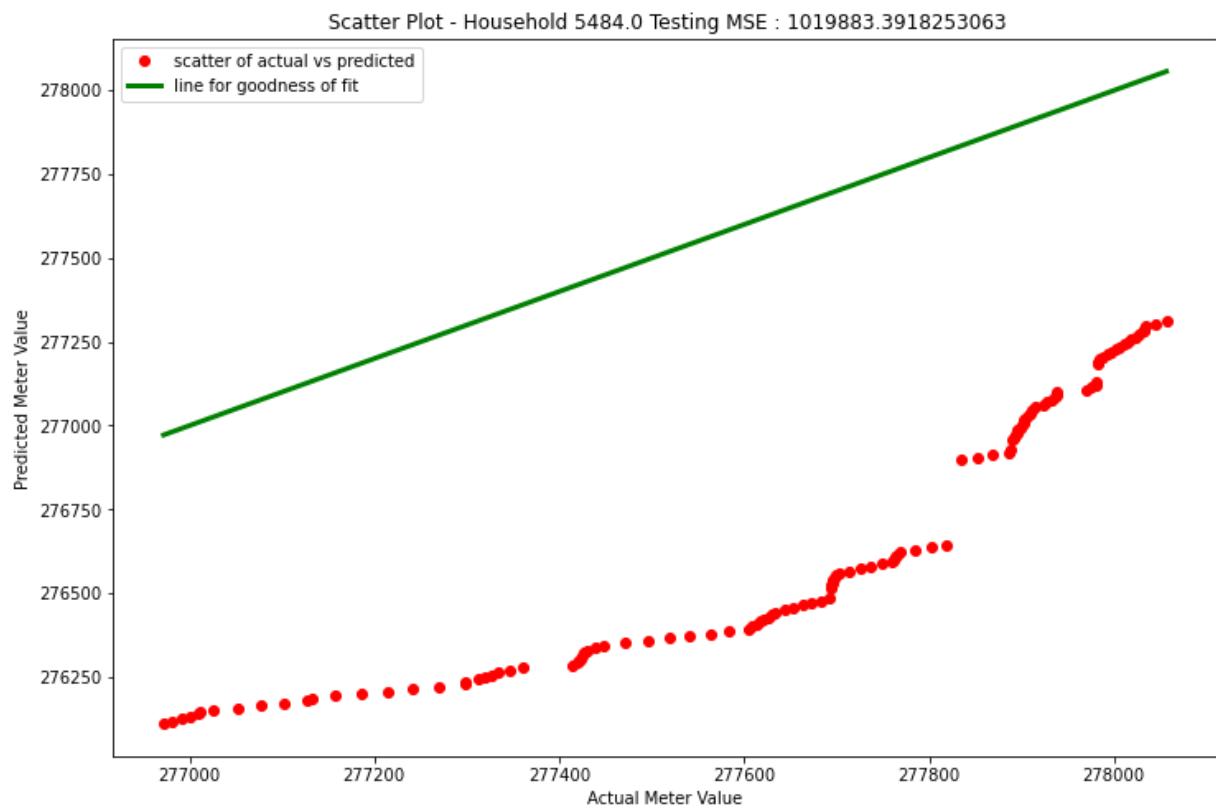


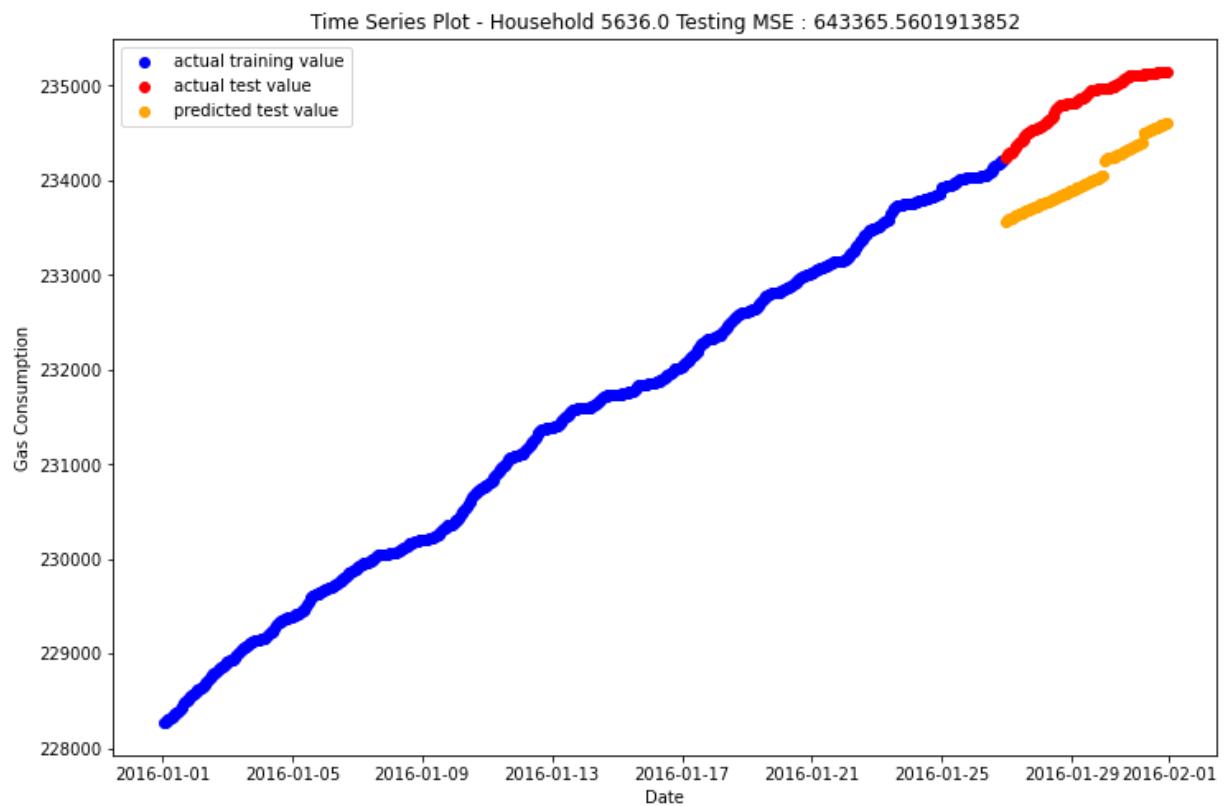
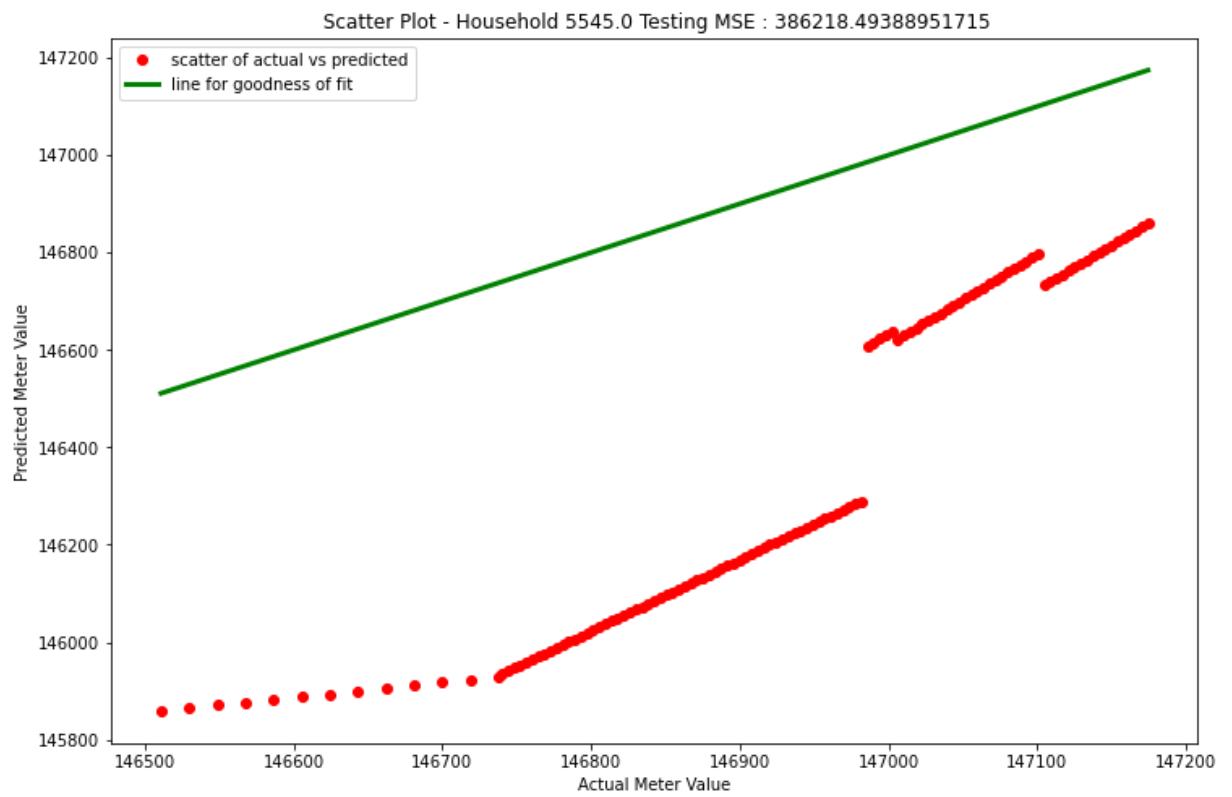


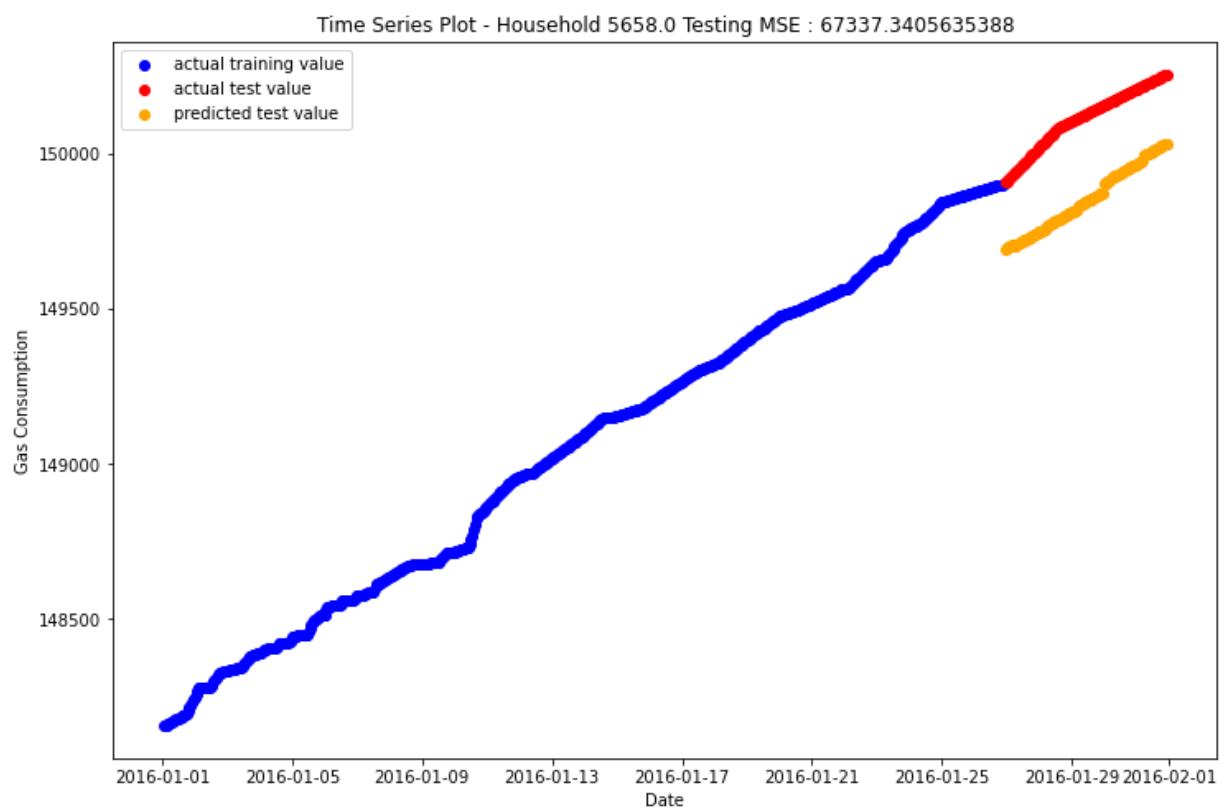
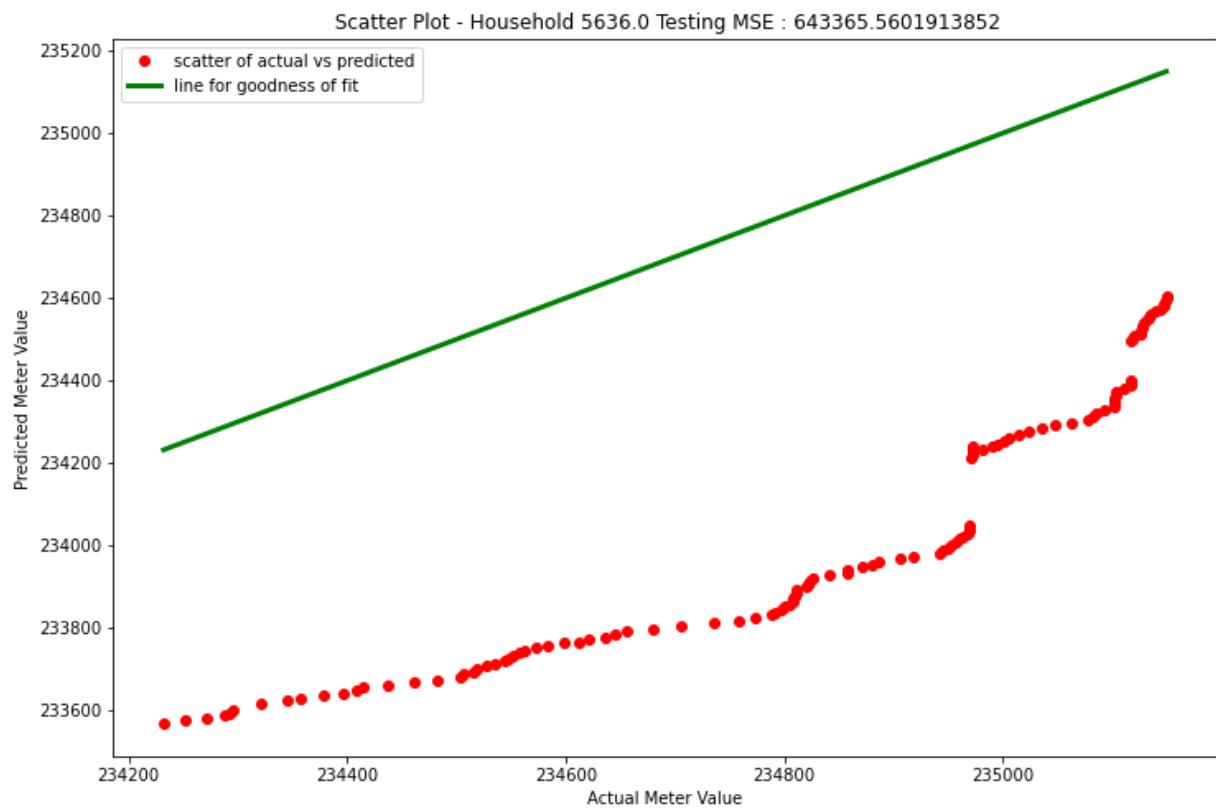




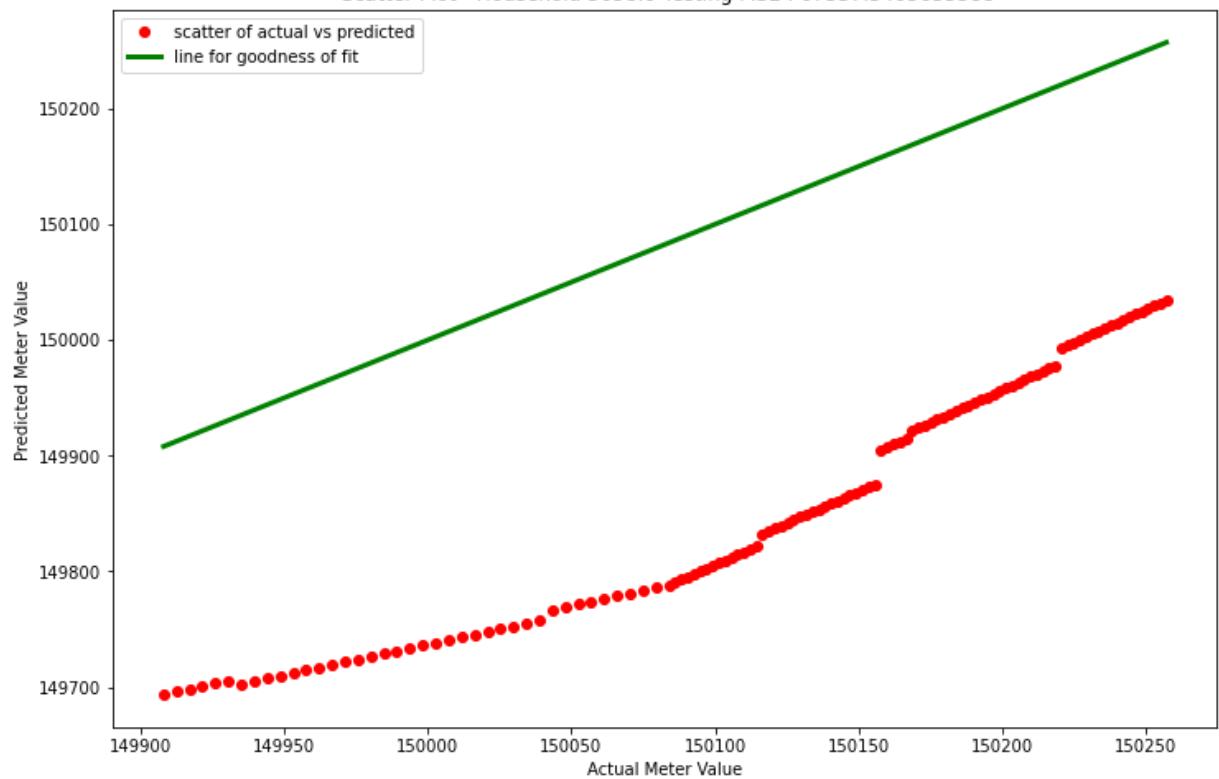




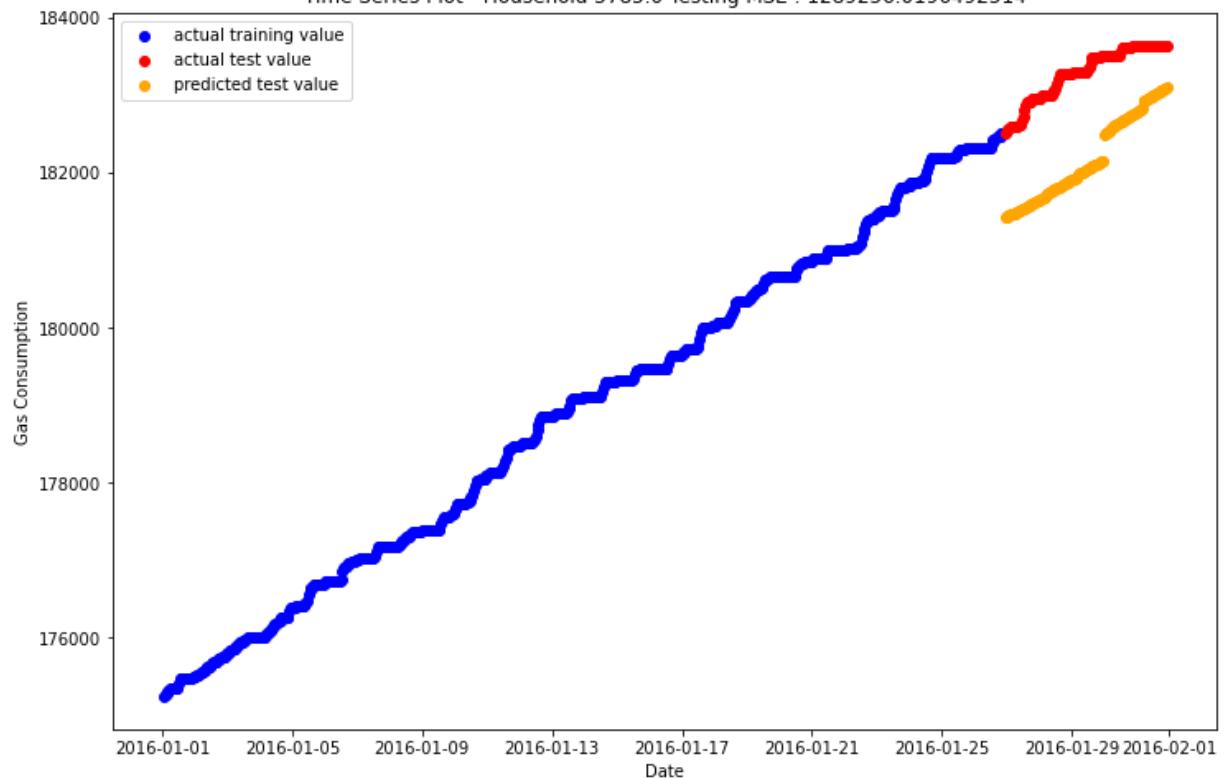


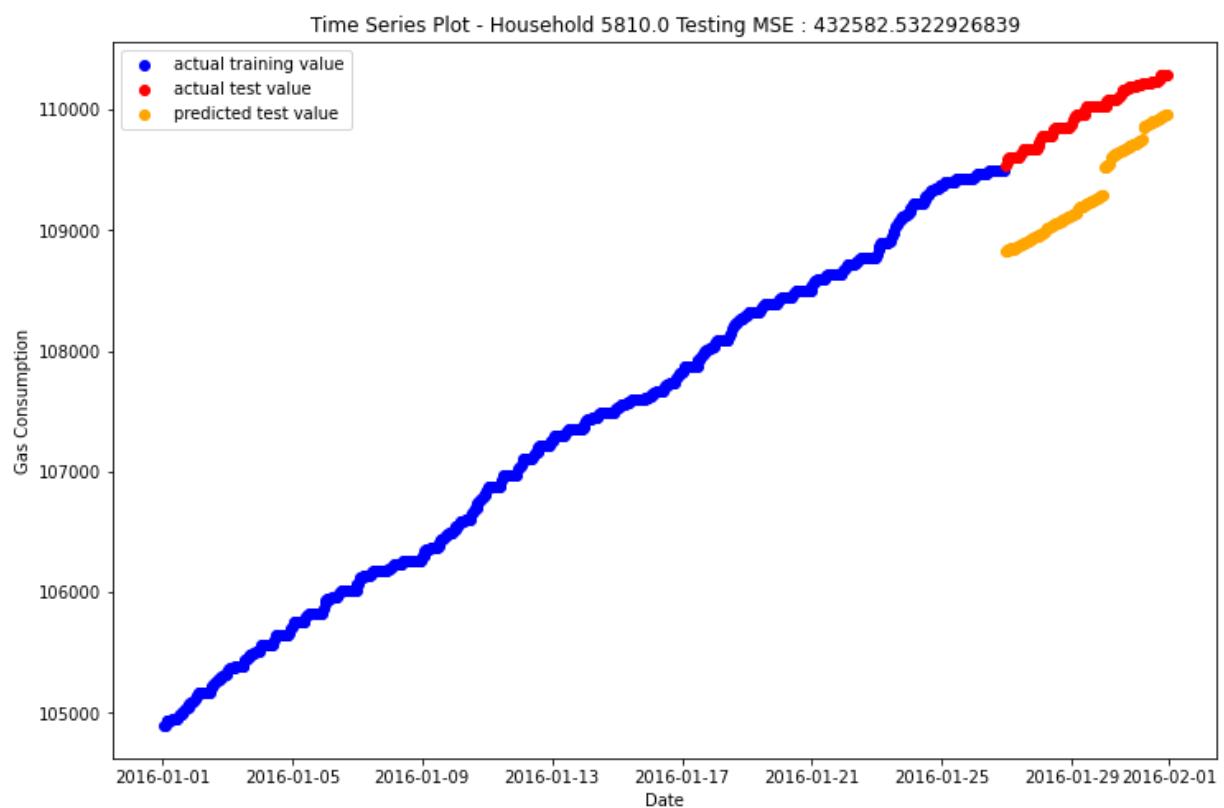
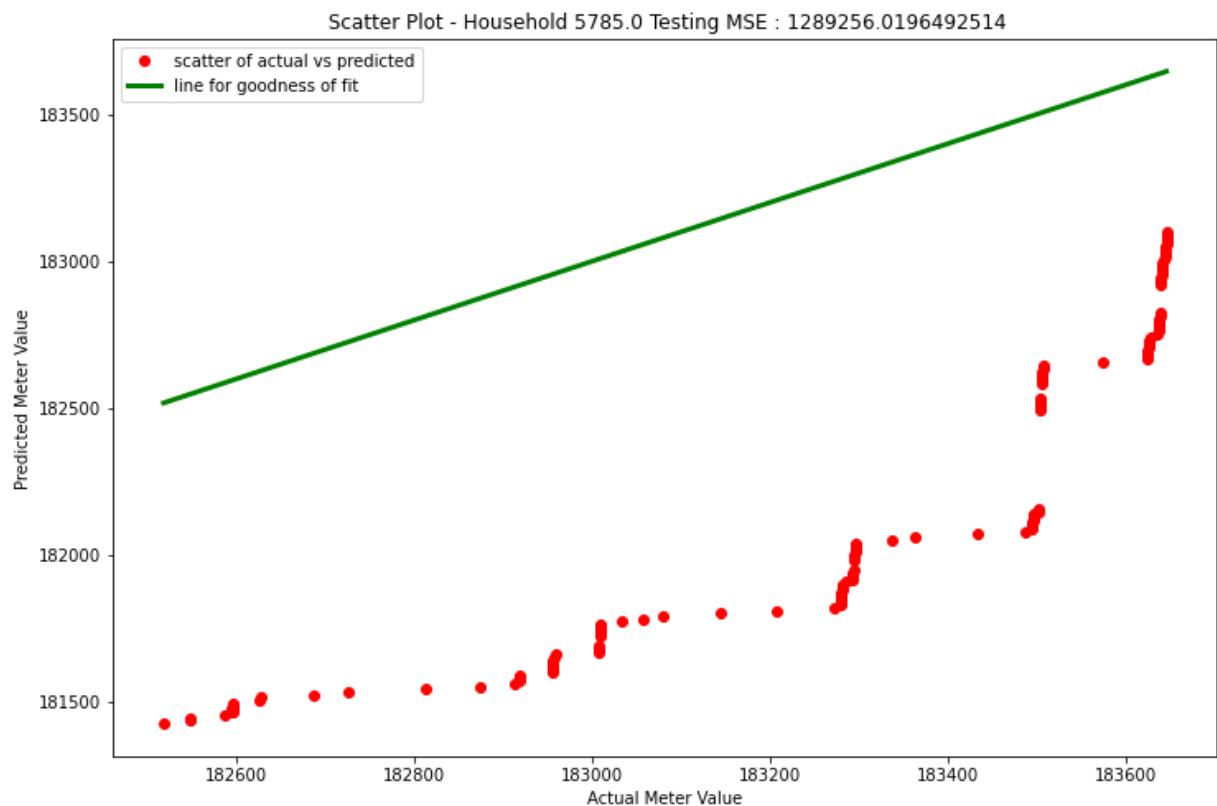


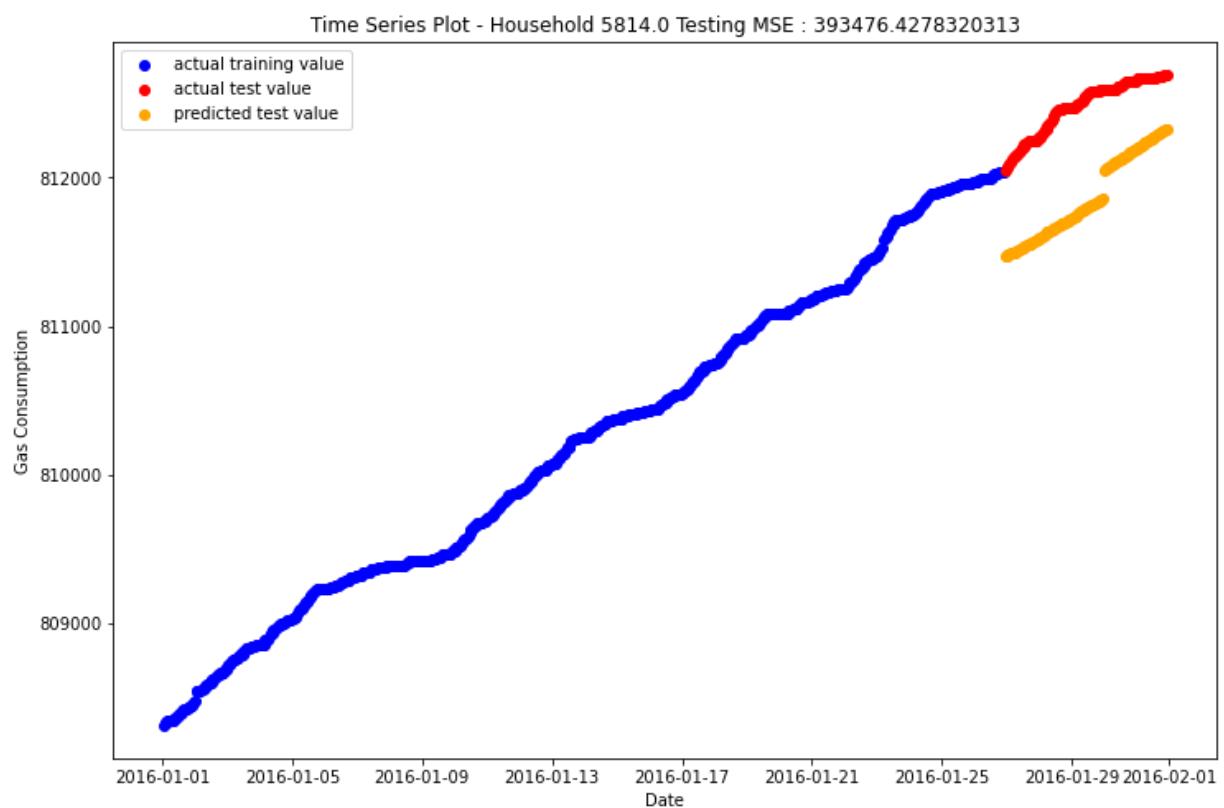
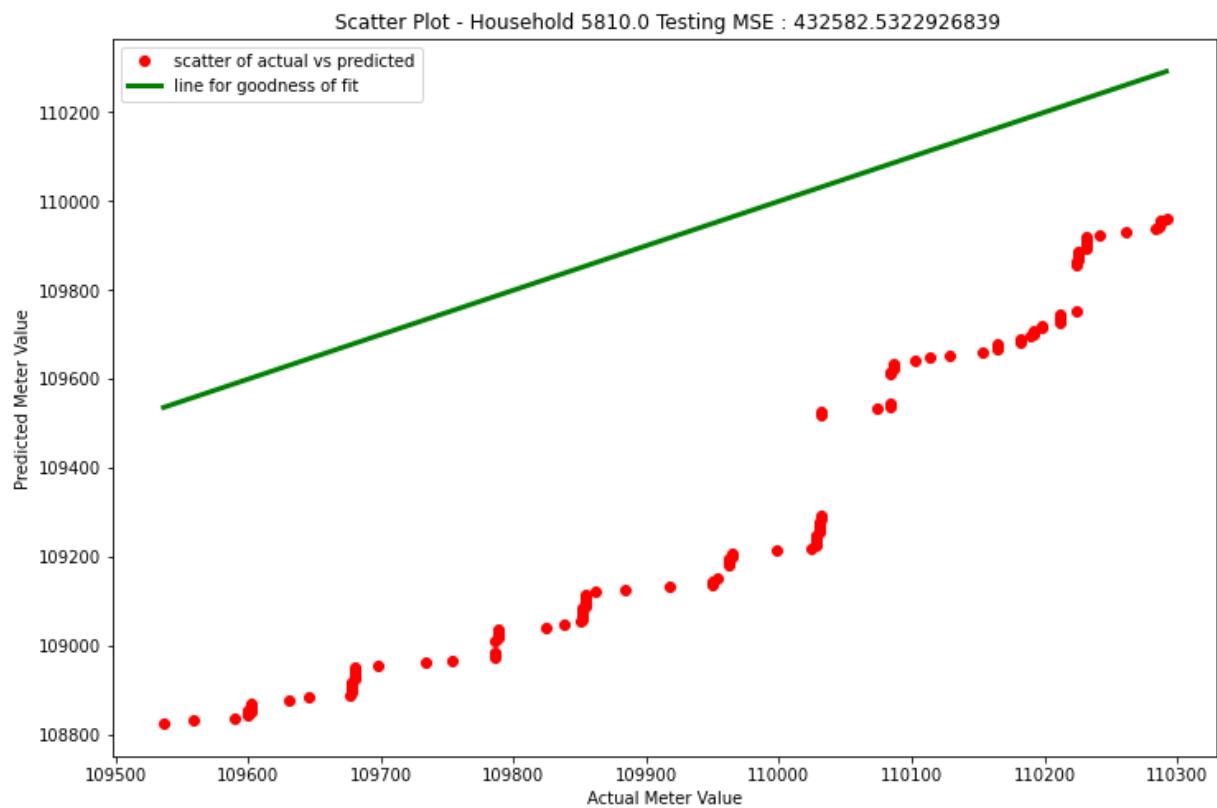
Scatter Plot - Household 5658.0 Testing MSE : 67337.3405635388



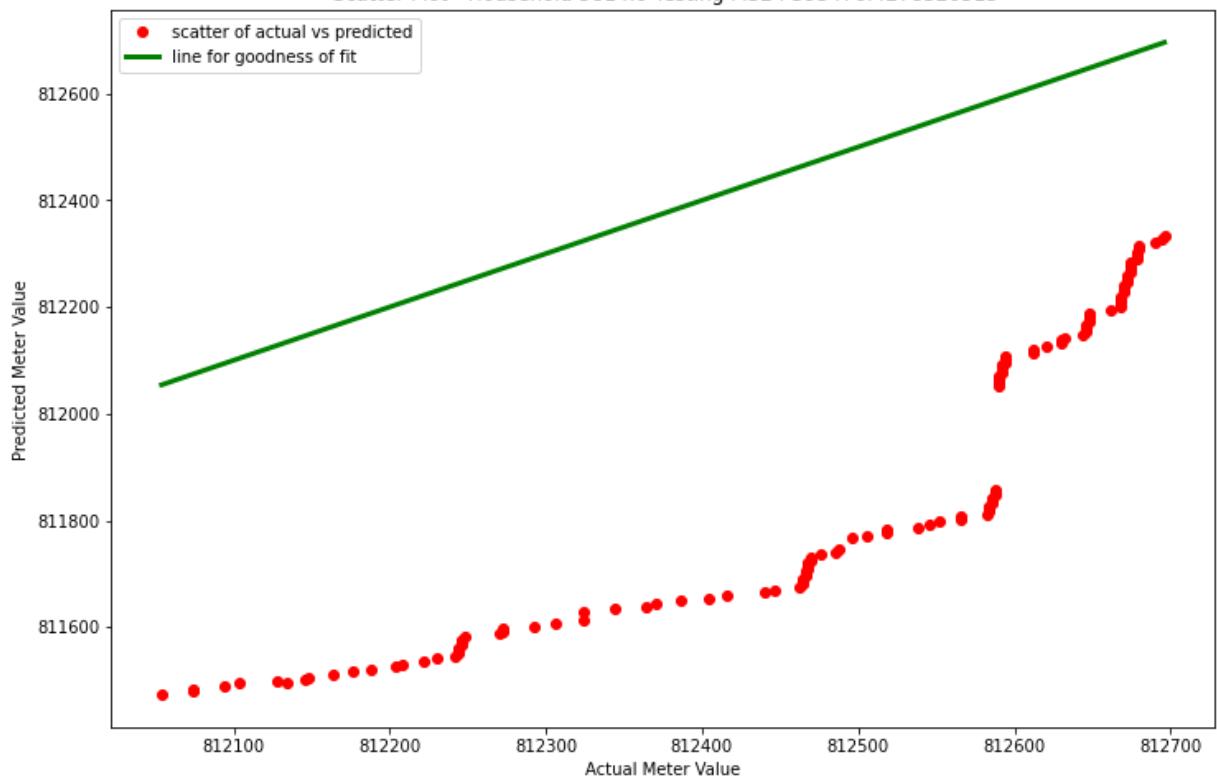
Time Series Plot - Household 5785.0 Testing MSE : 1289256.0196492514



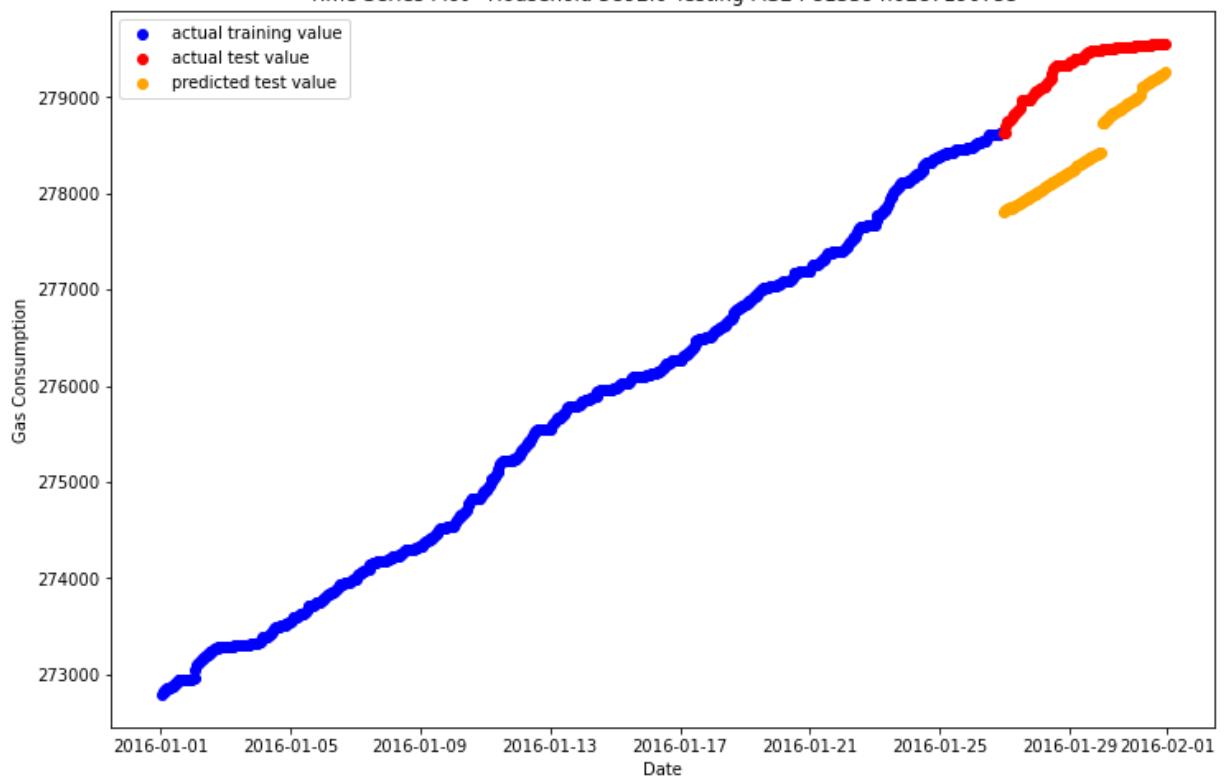


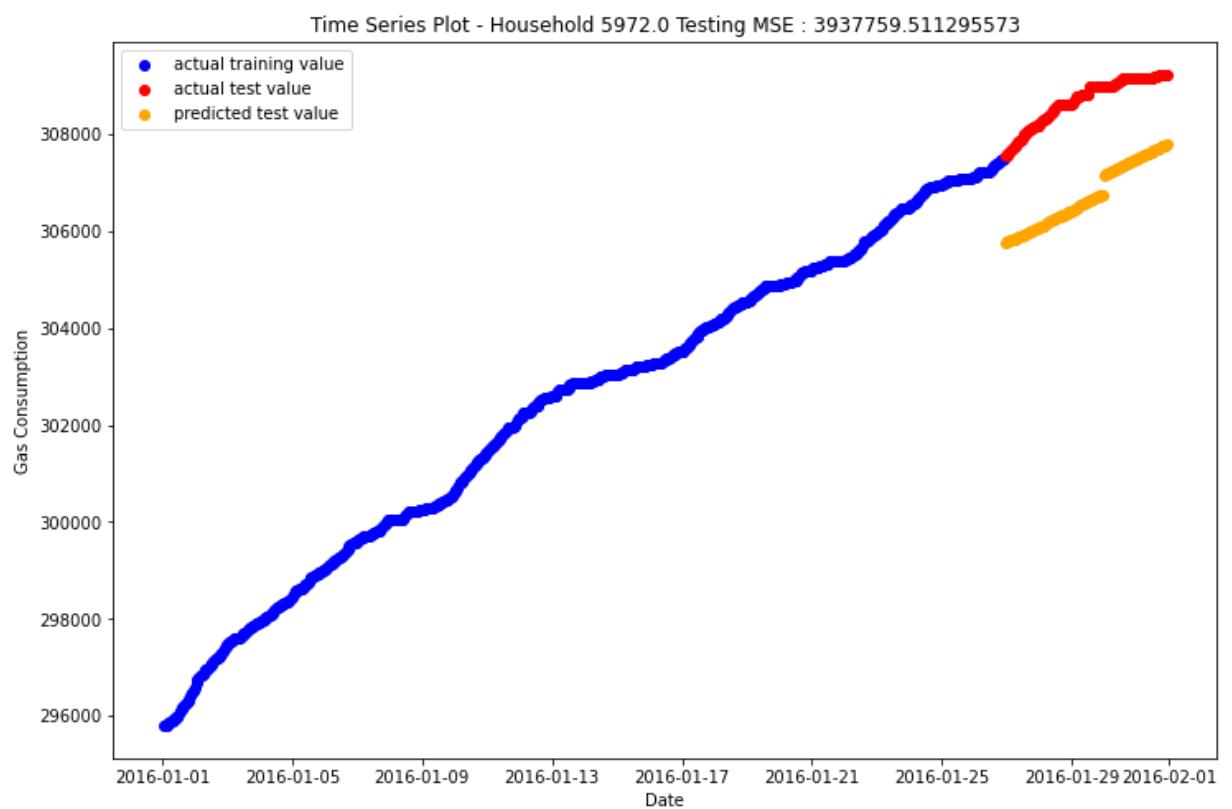
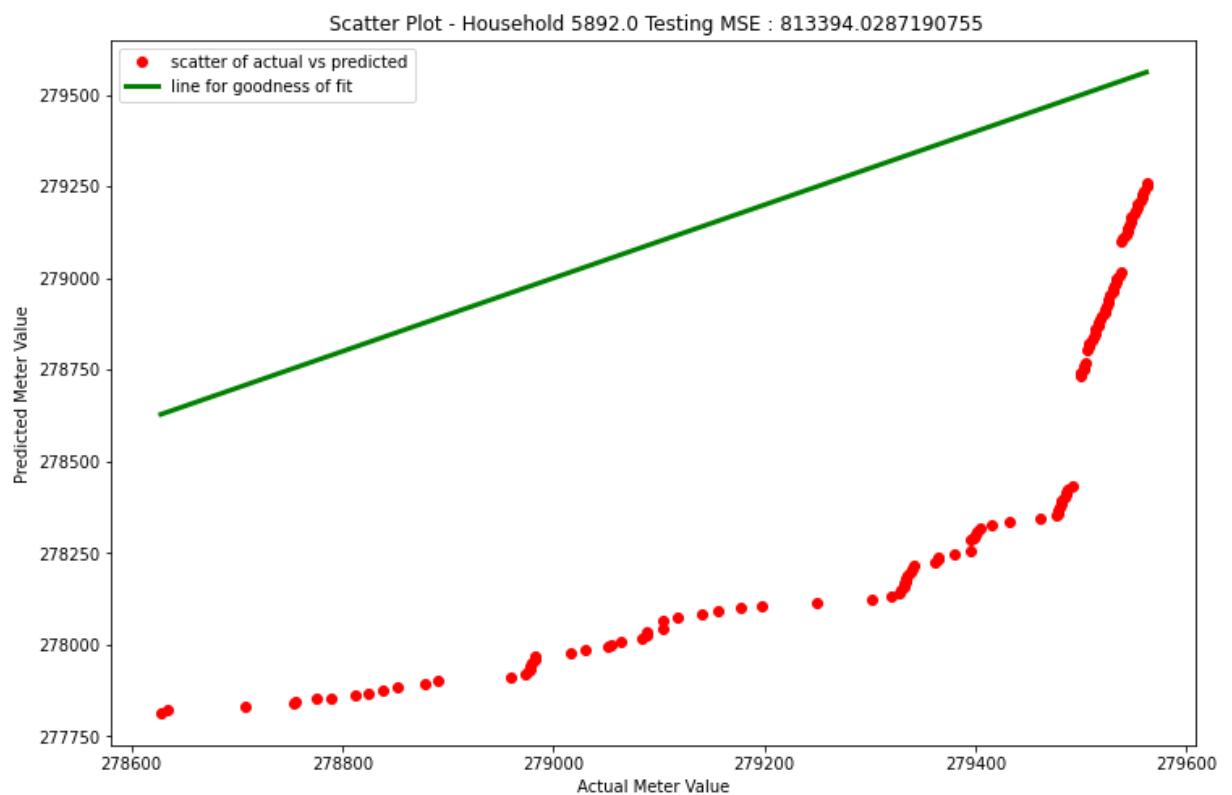


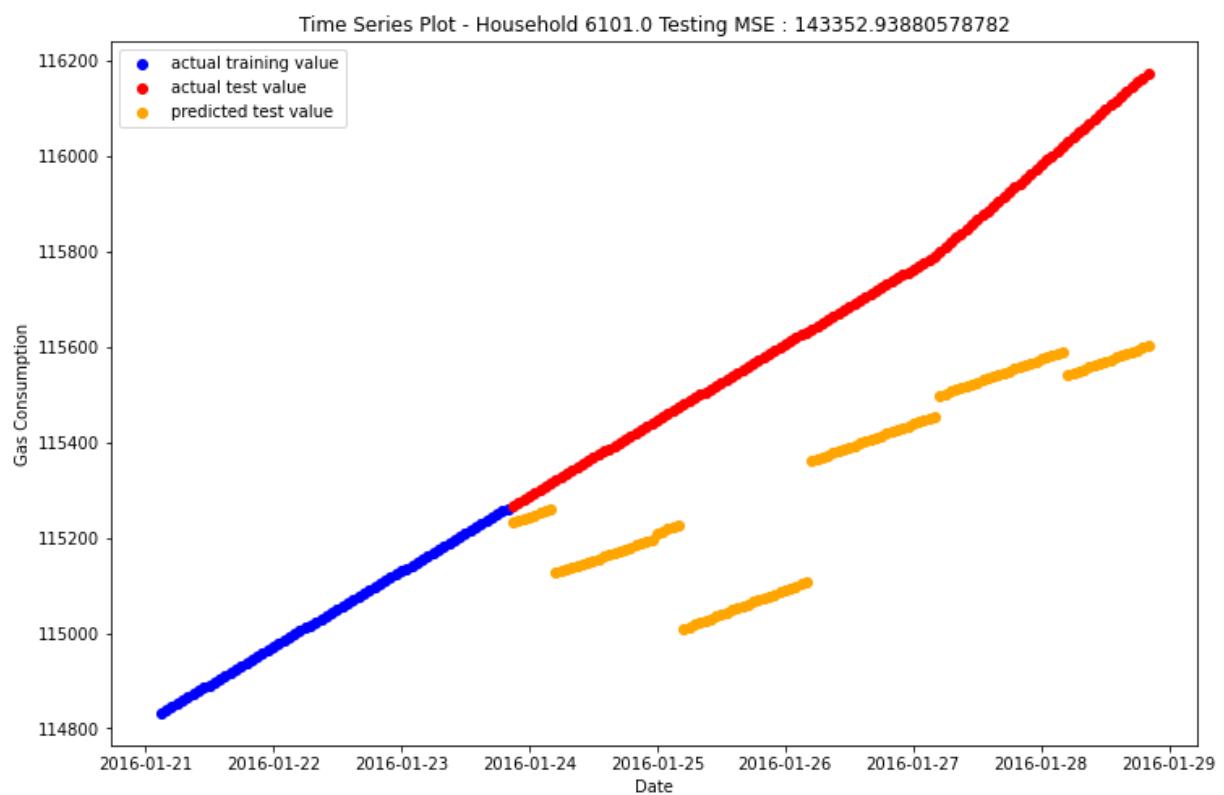
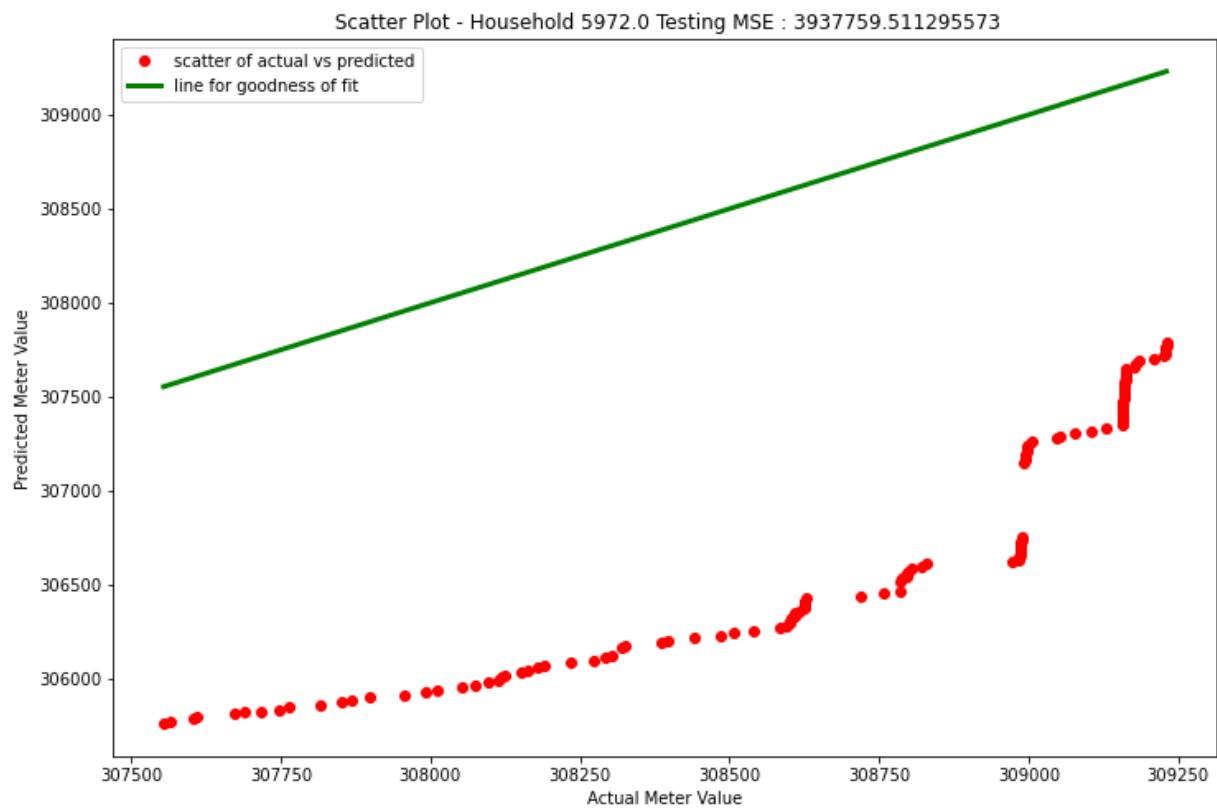
Scatter Plot - Household 5814.0 Testing MSE : 393476.4278320313

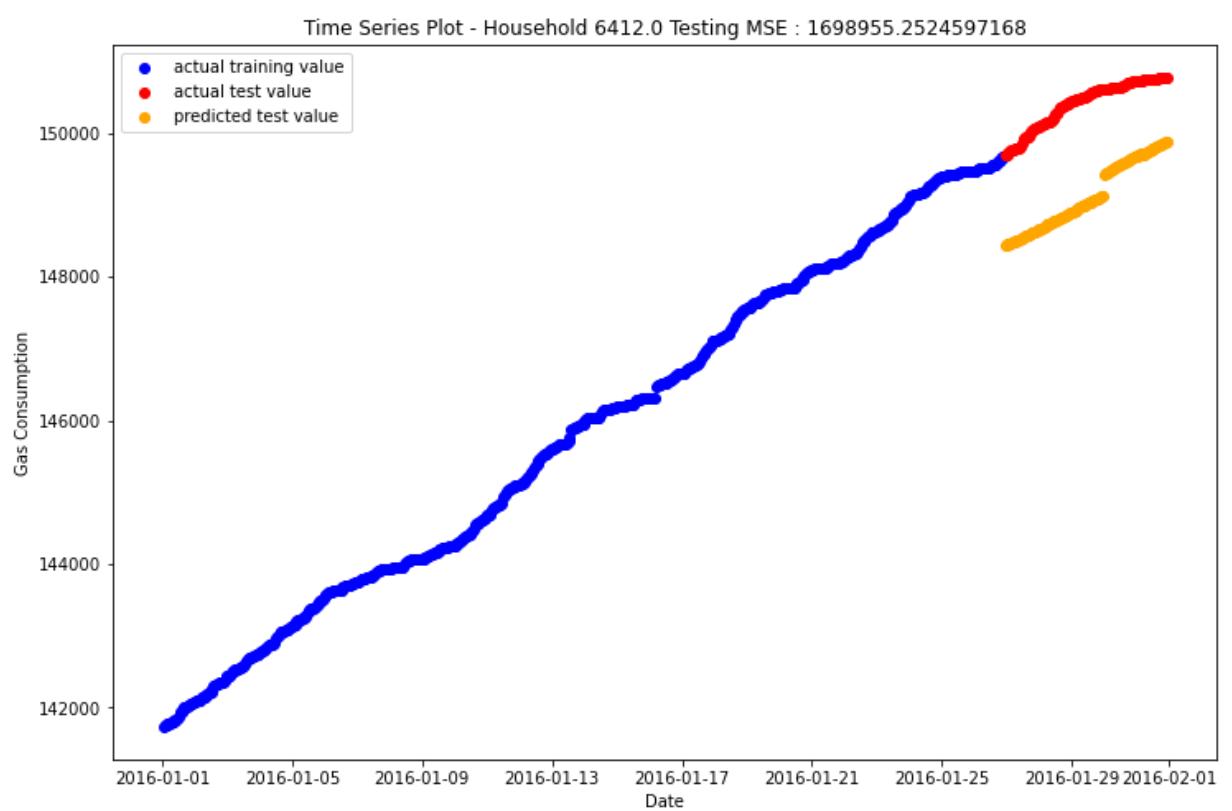
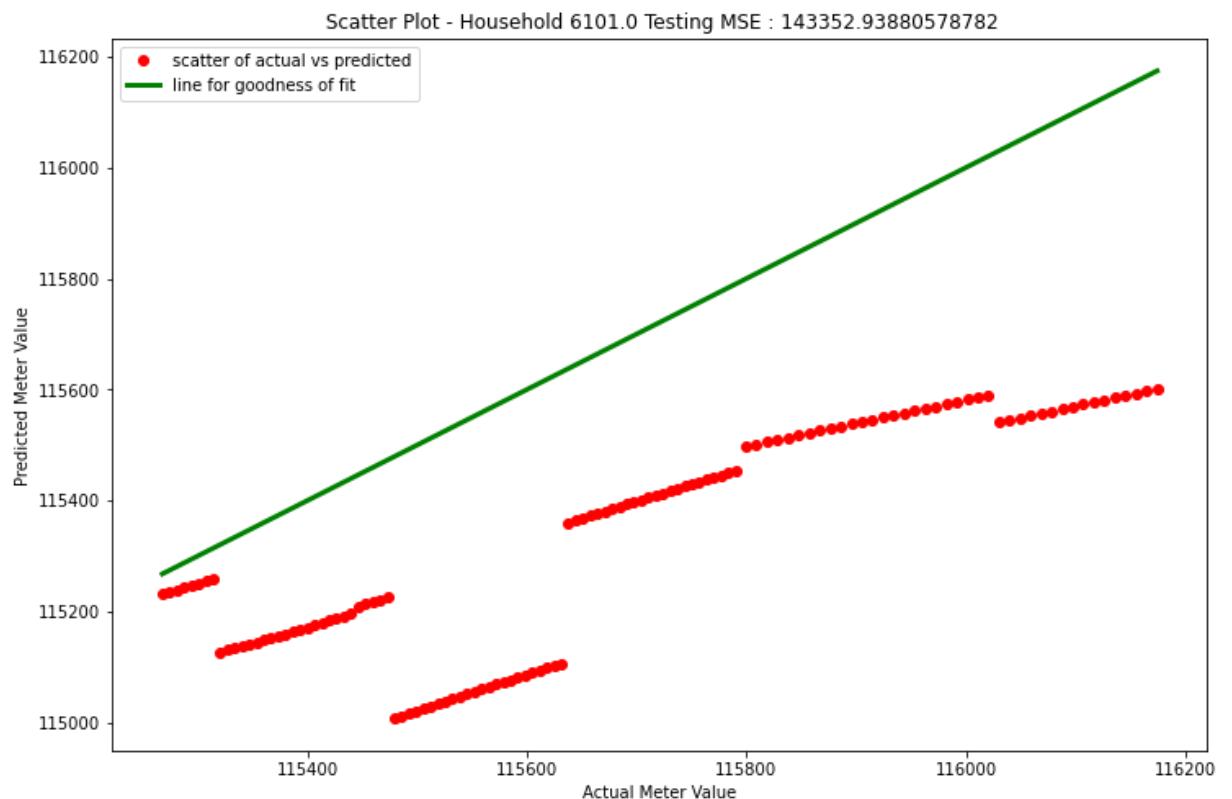


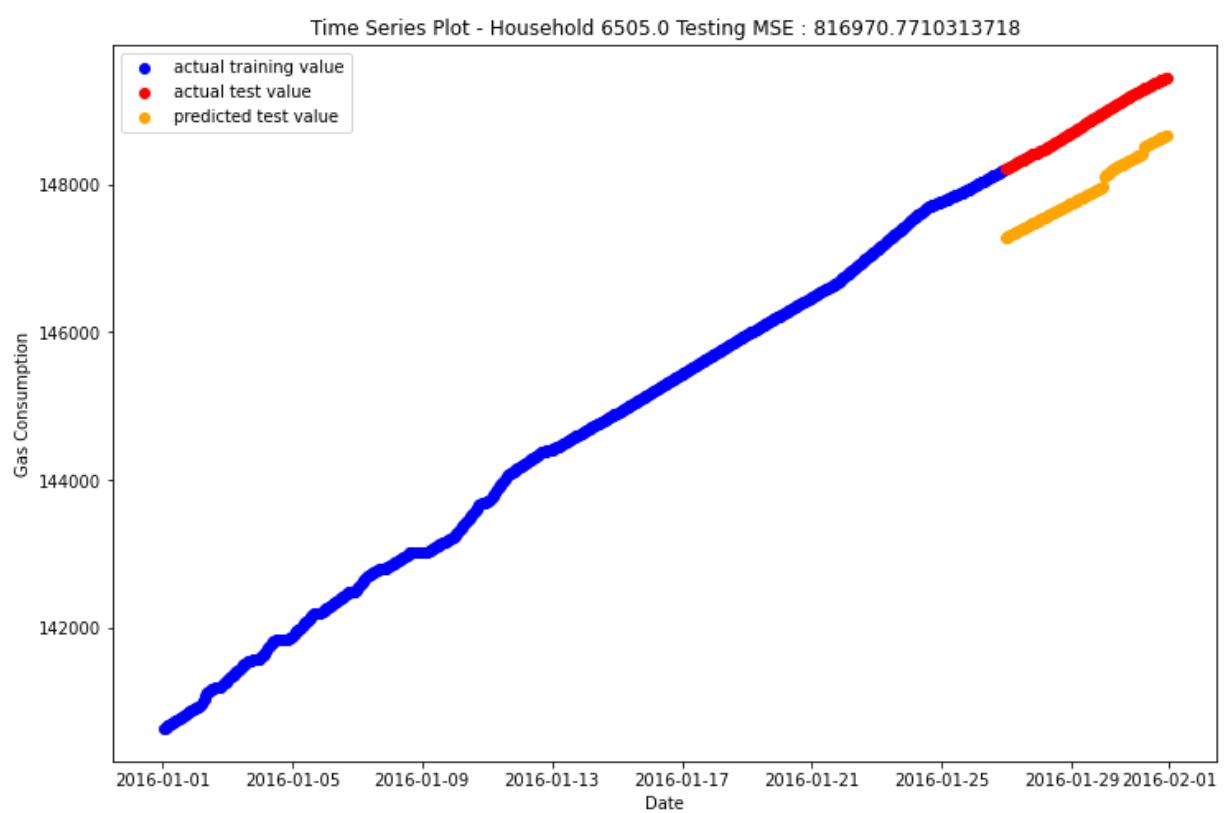
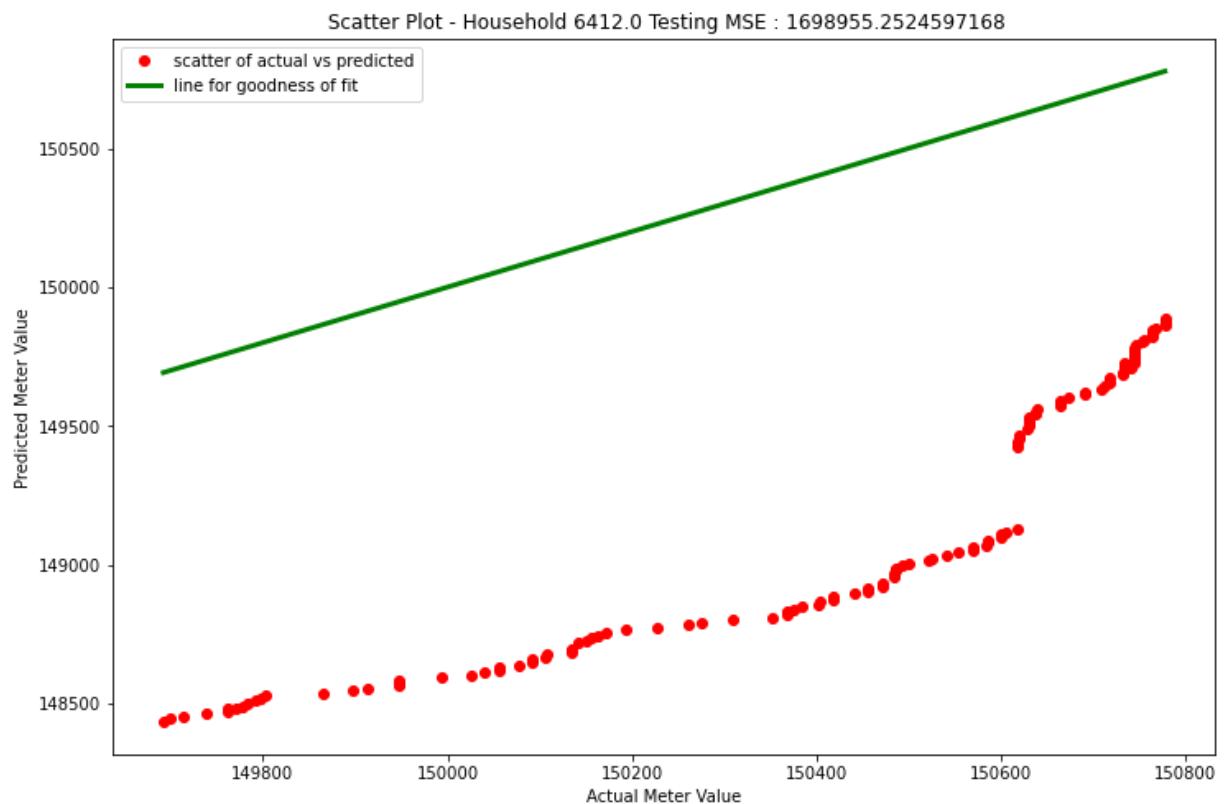
Time Series Plot - Household 5892.0 Testing MSE : 813394.0287190755

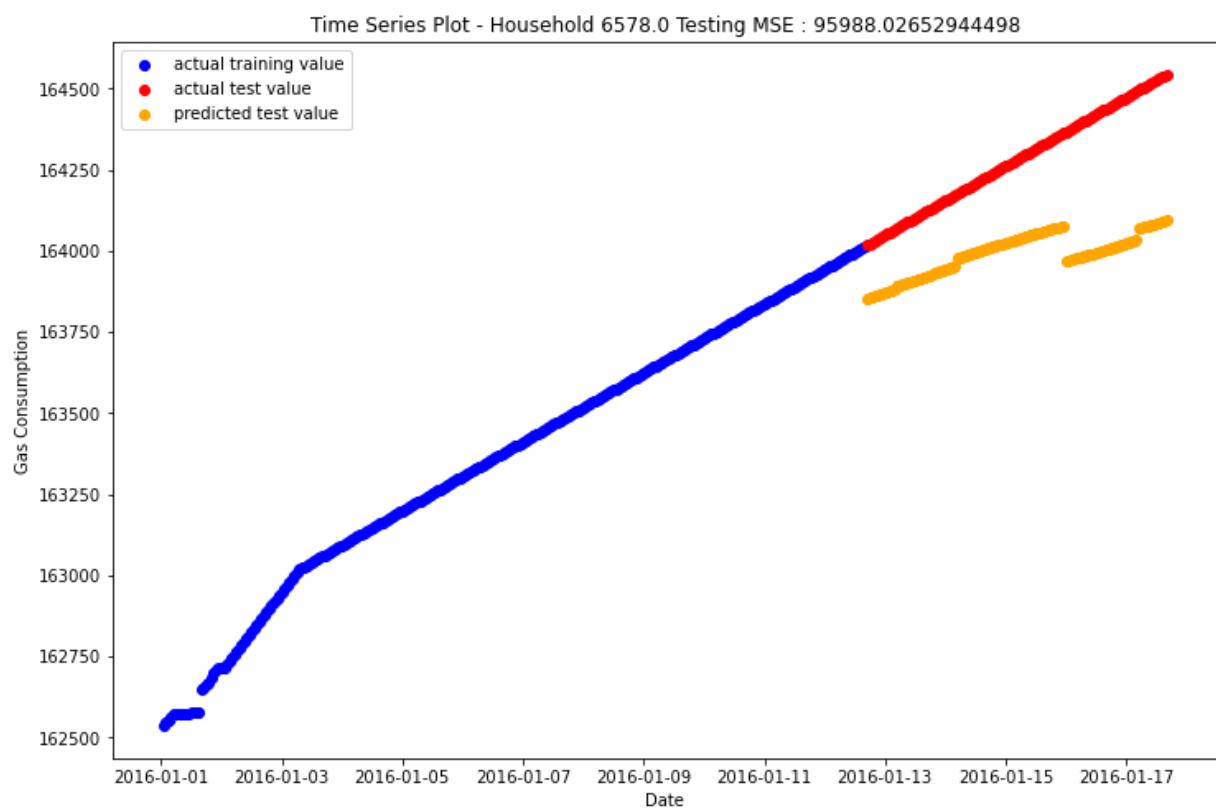
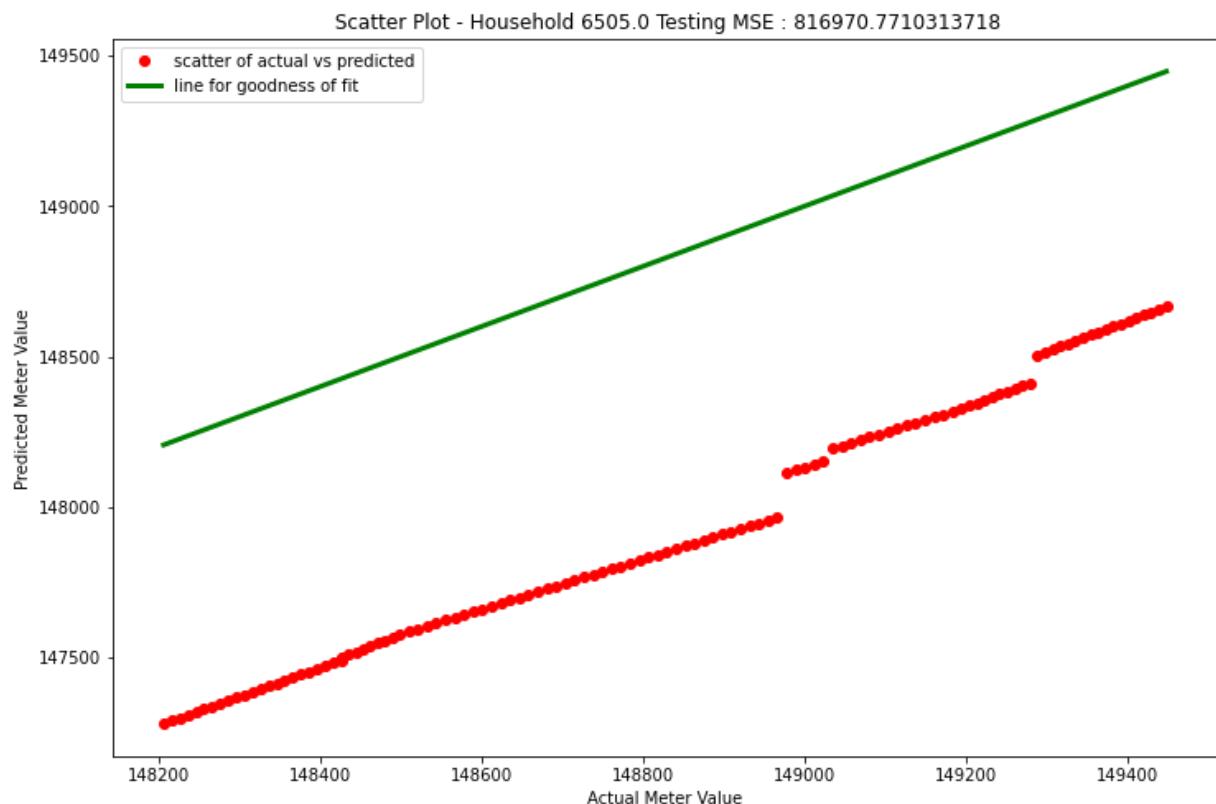


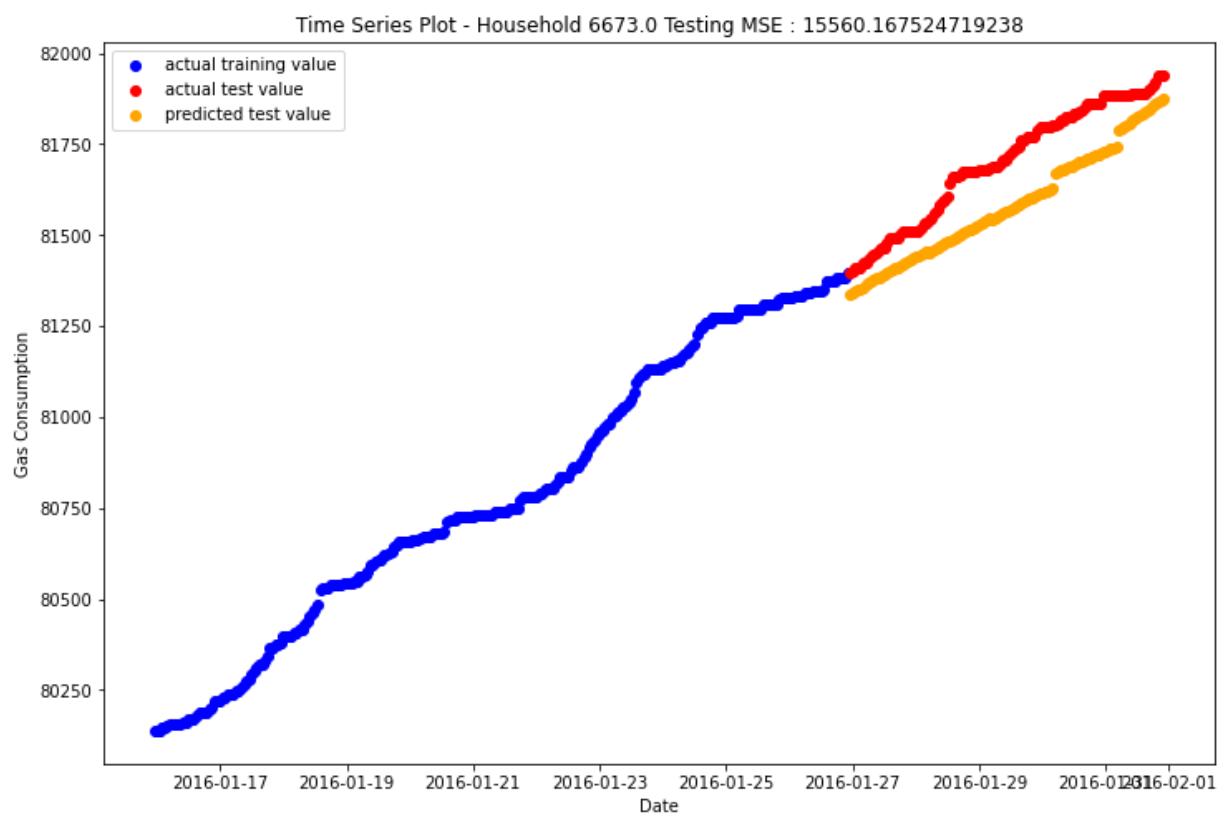
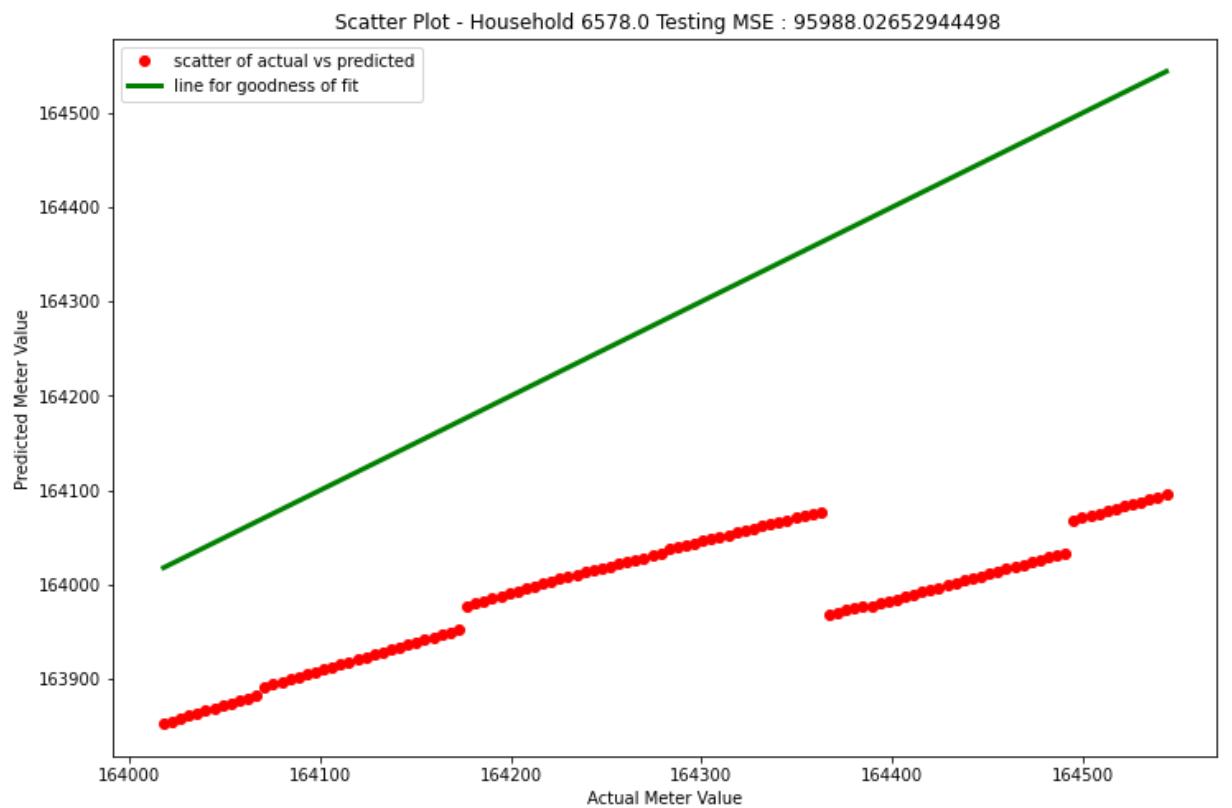




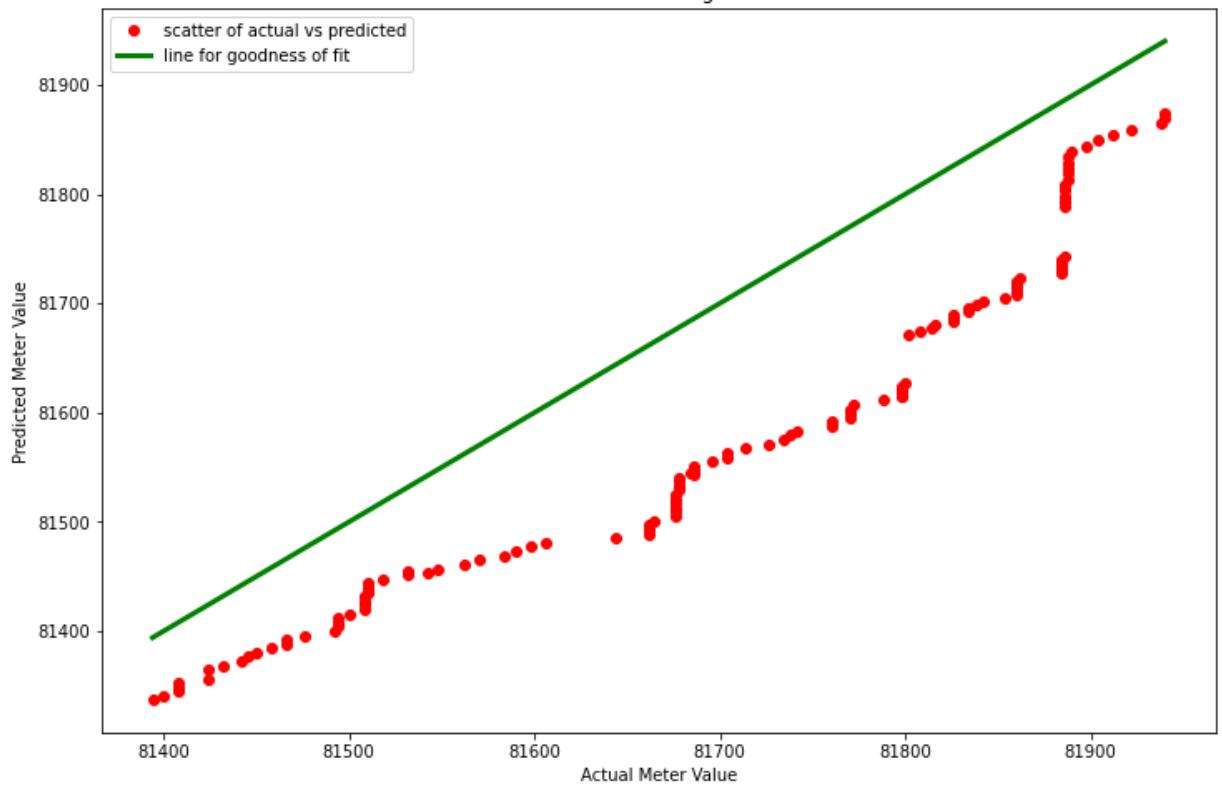




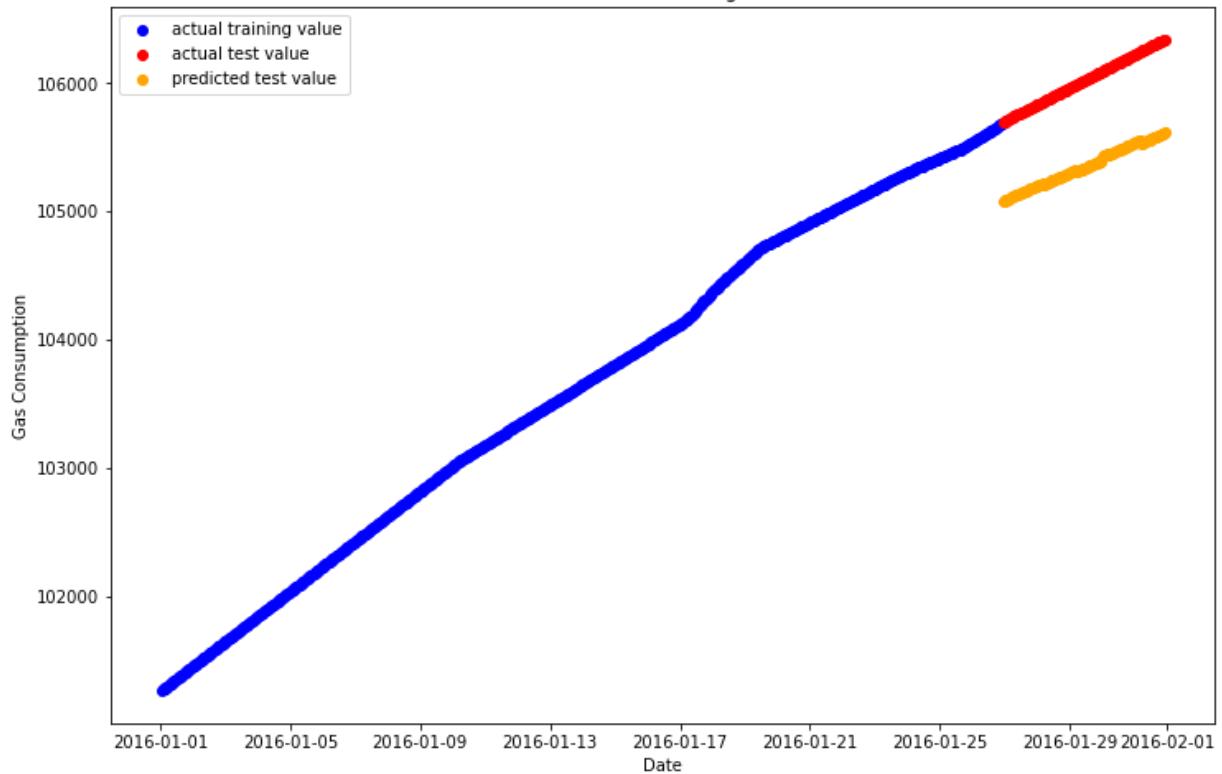


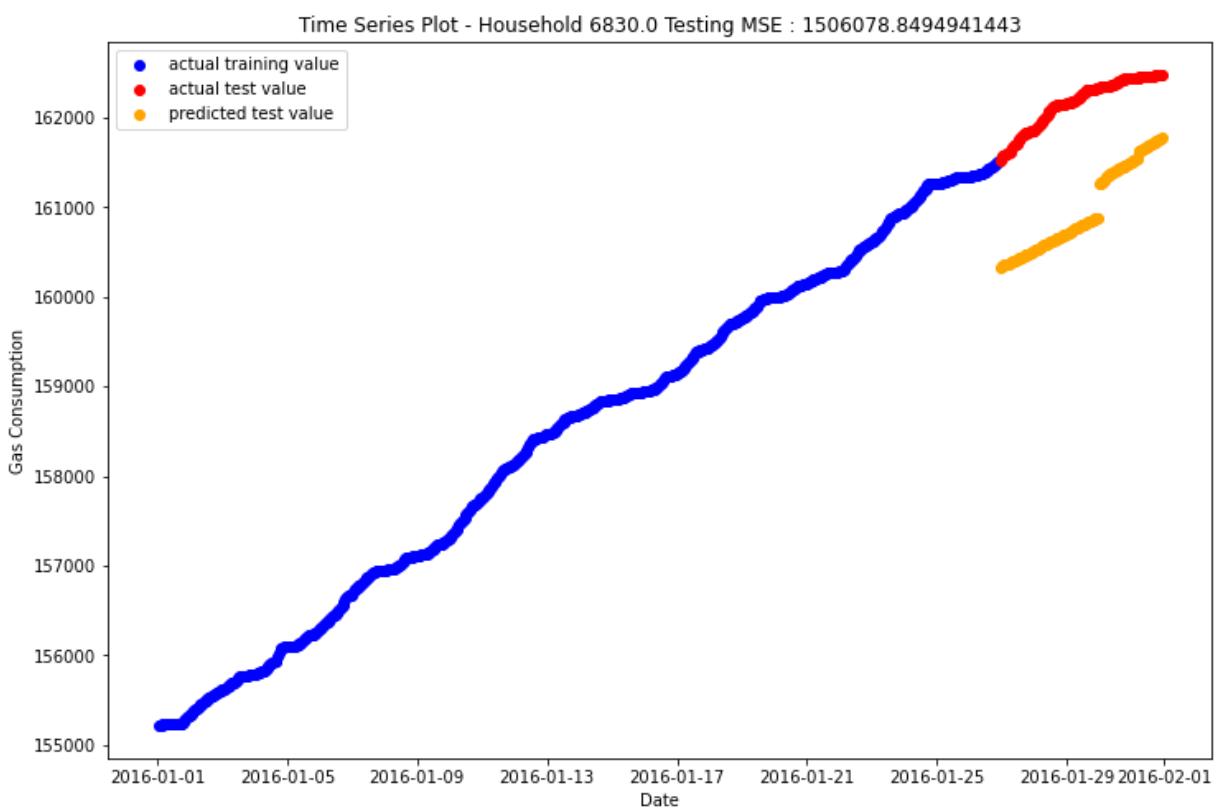
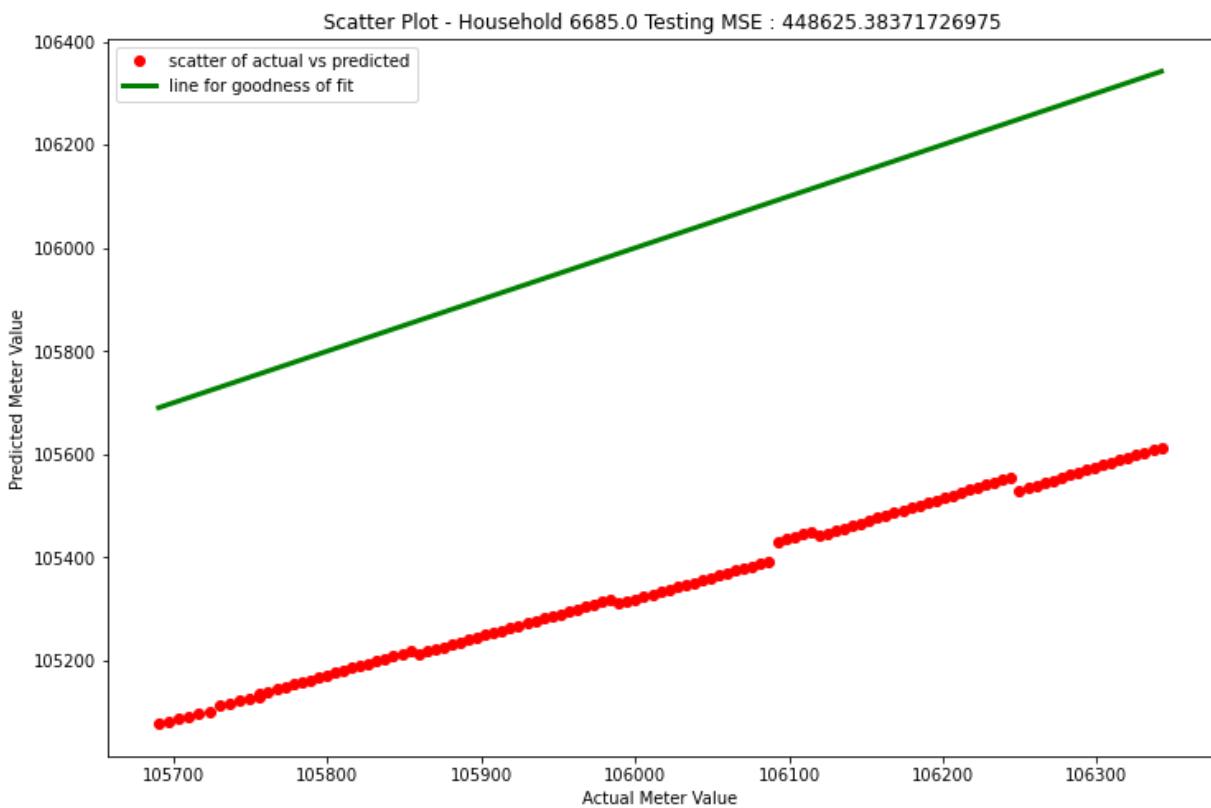


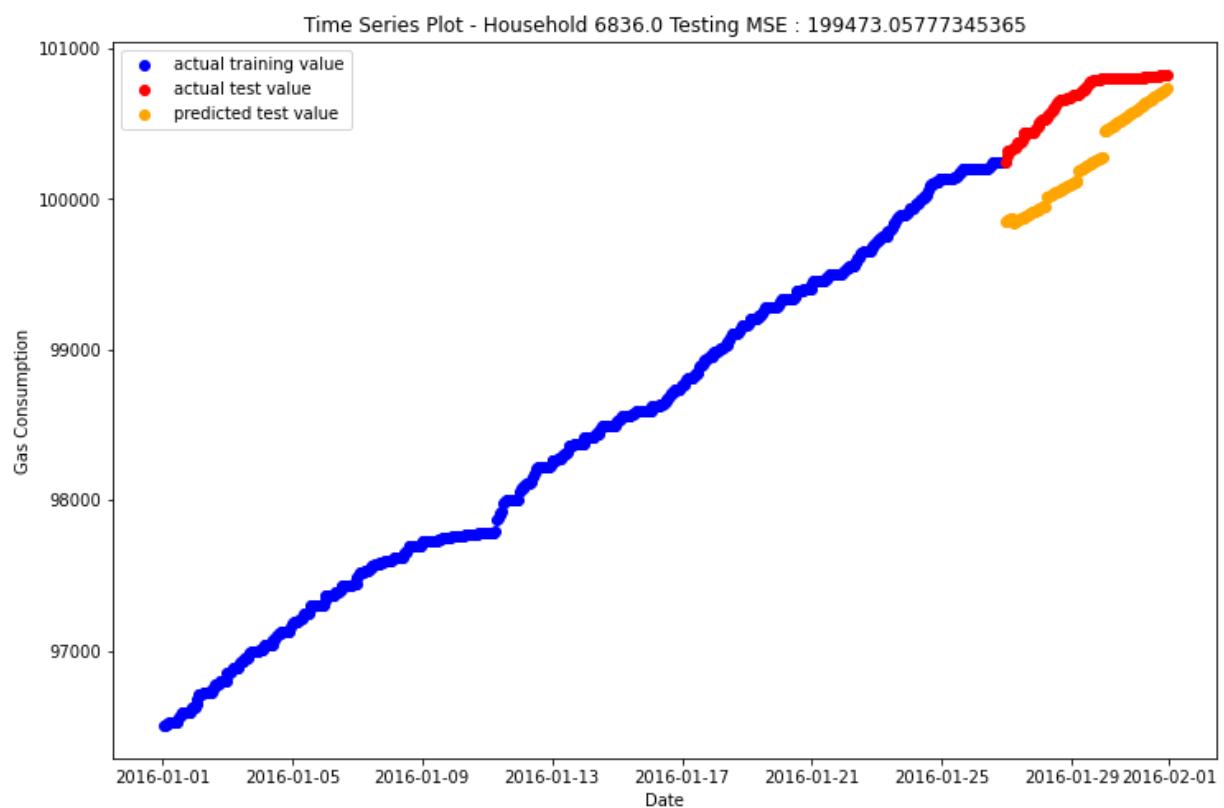
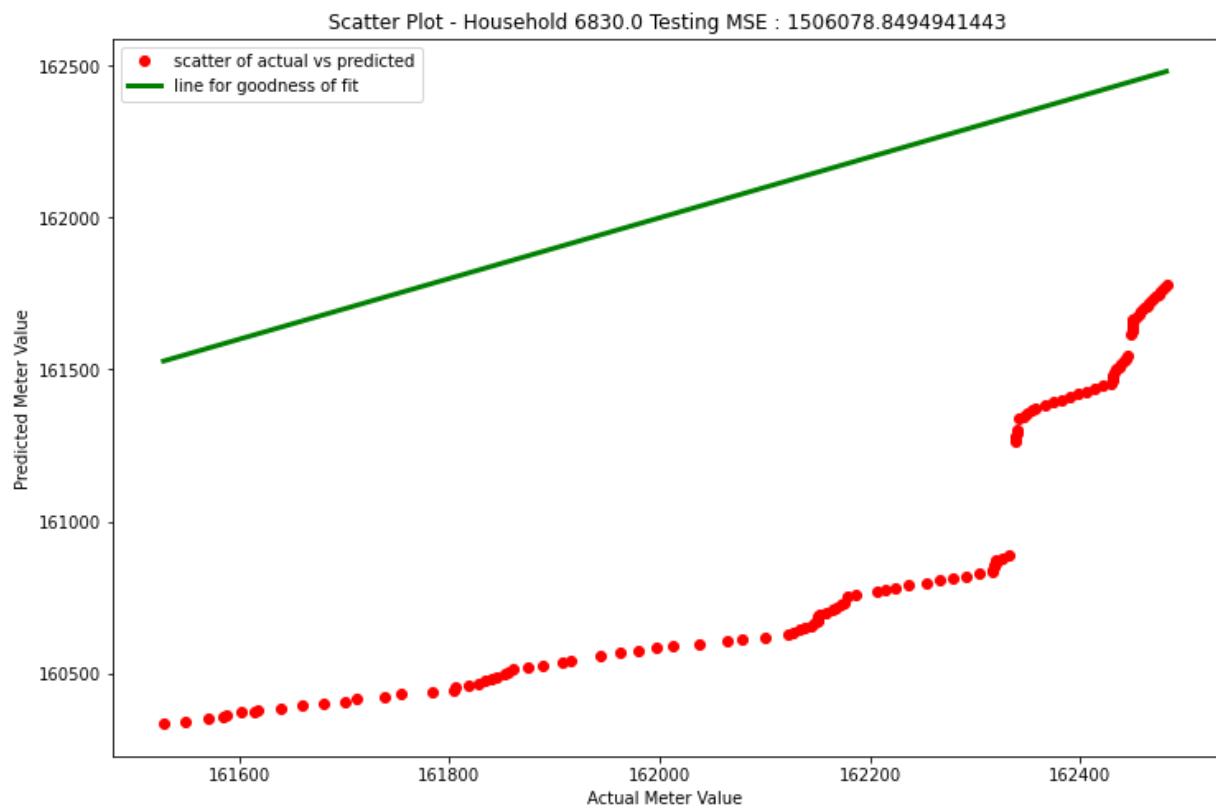
Scatter Plot - Household 6673.0 Testing MSE : 15560.167524719238

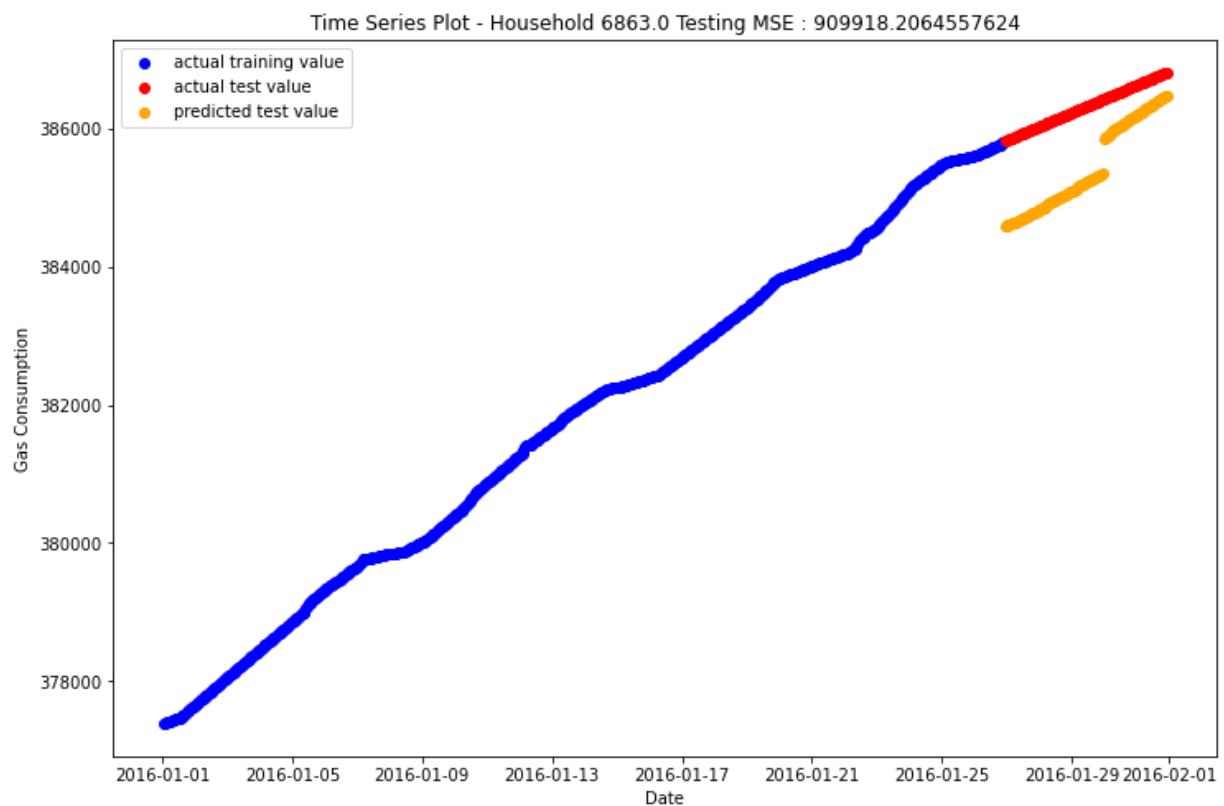
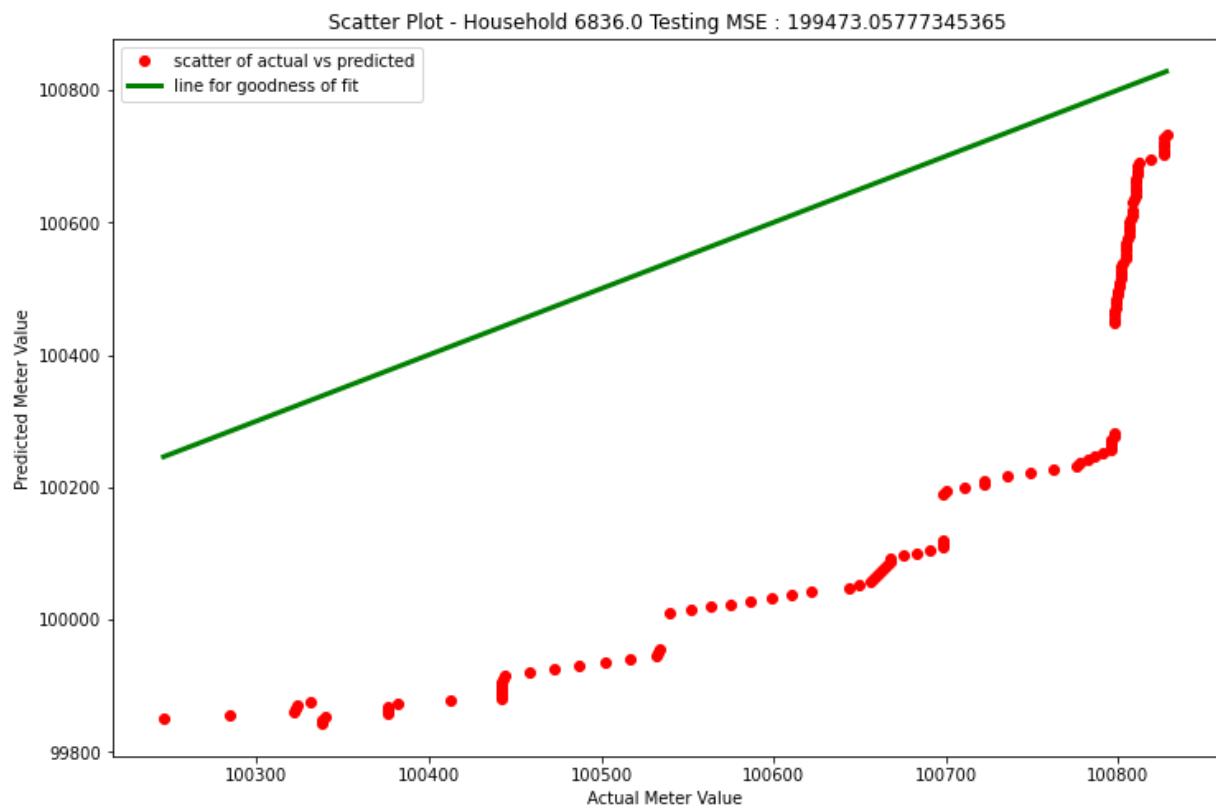


Time Series Plot - Household 6685.0 Testing MSE : 448625.38371726975

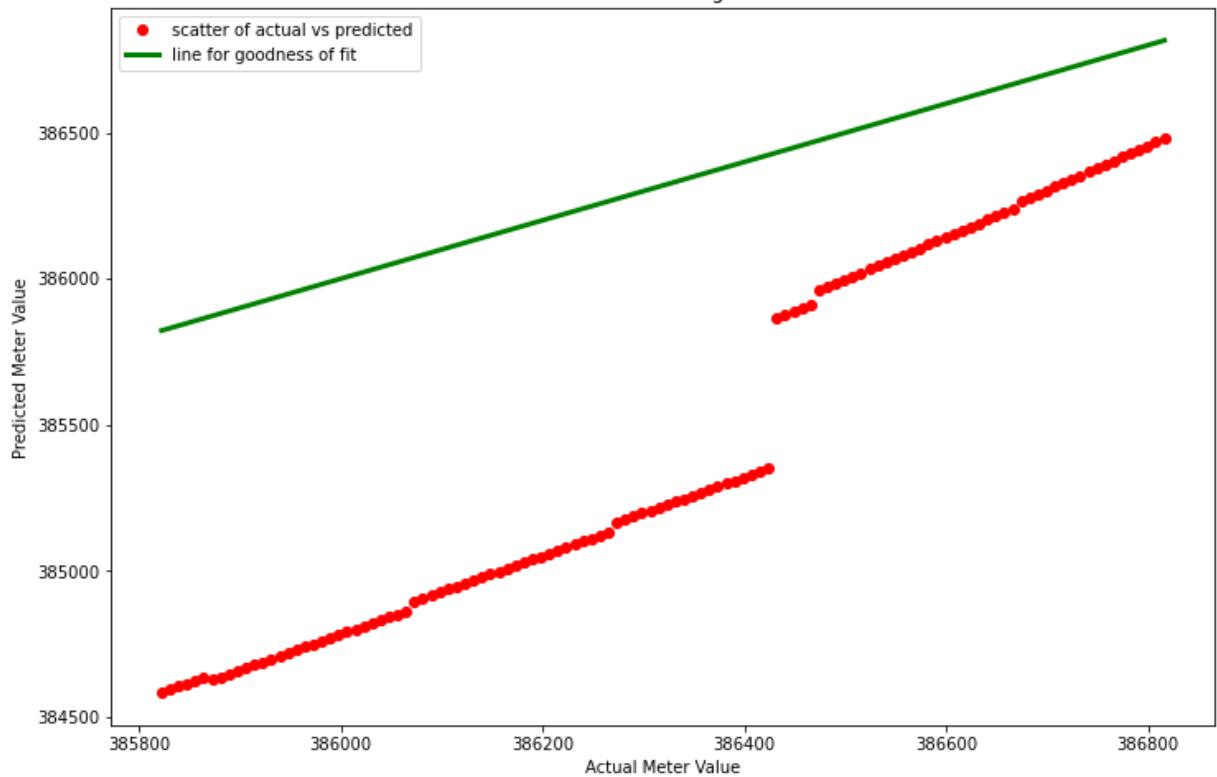




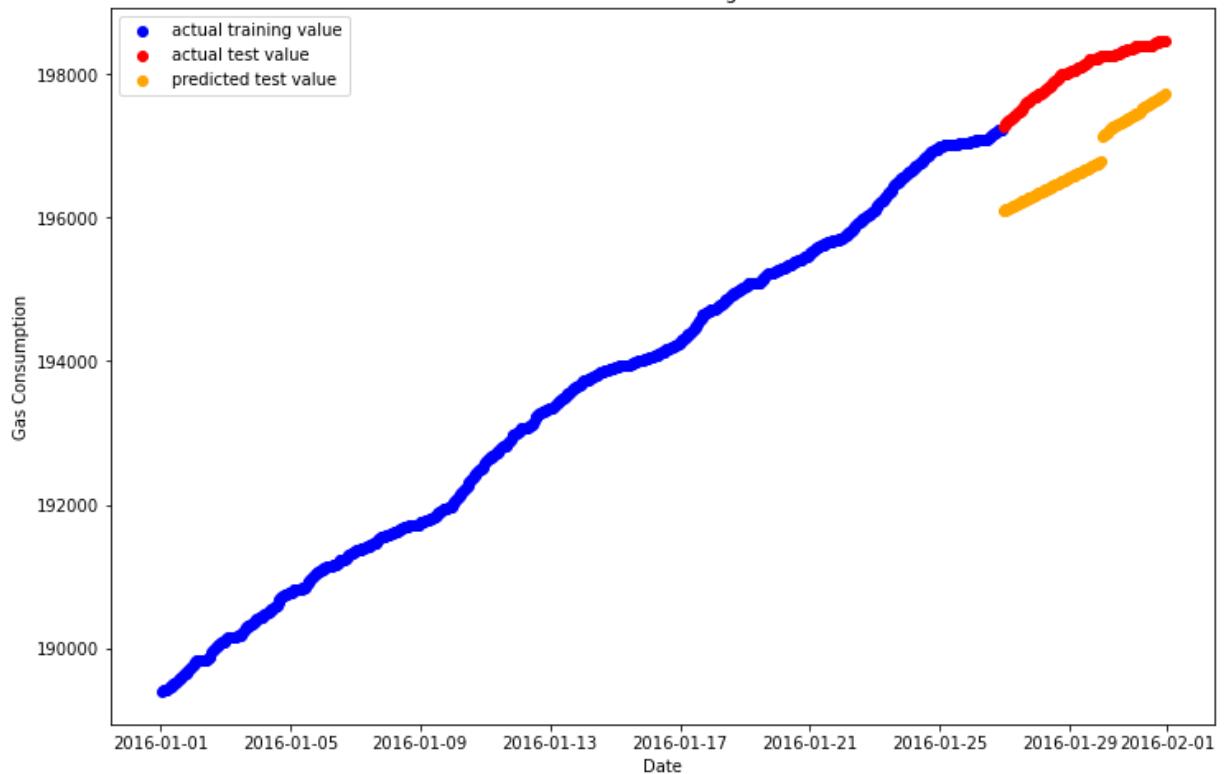


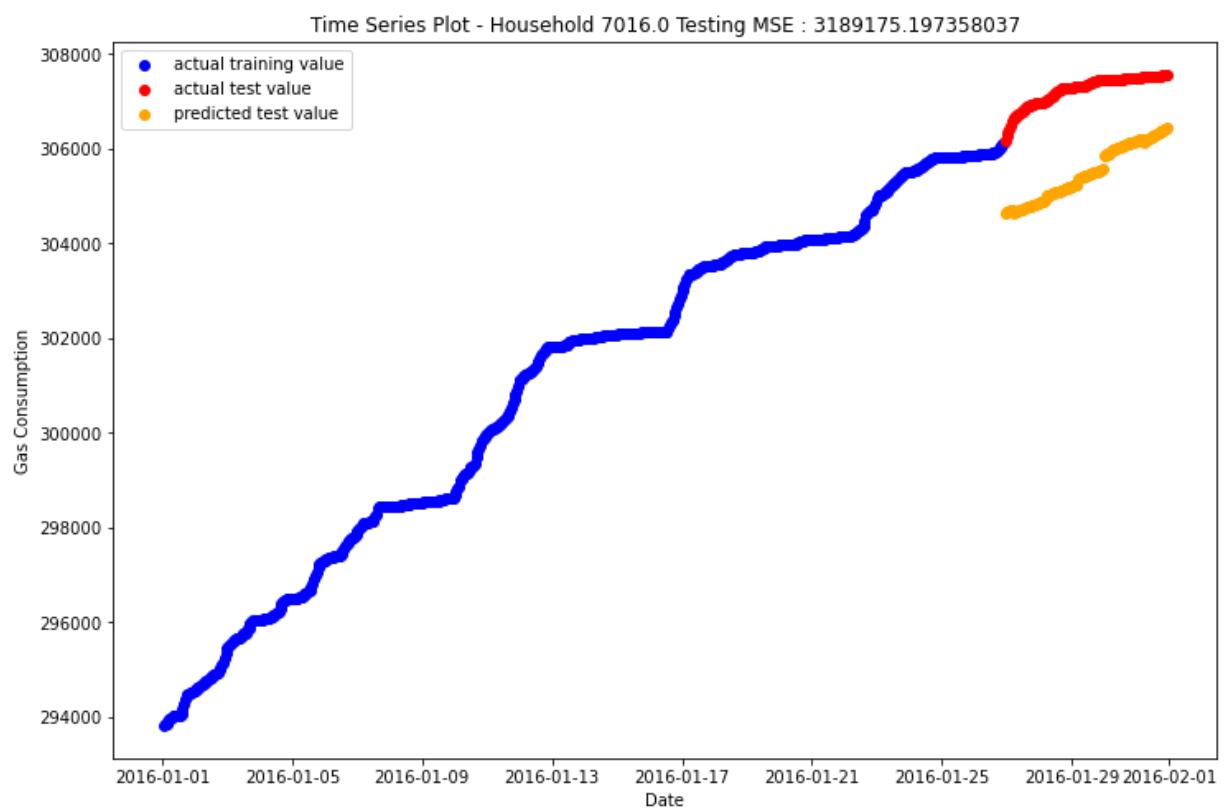
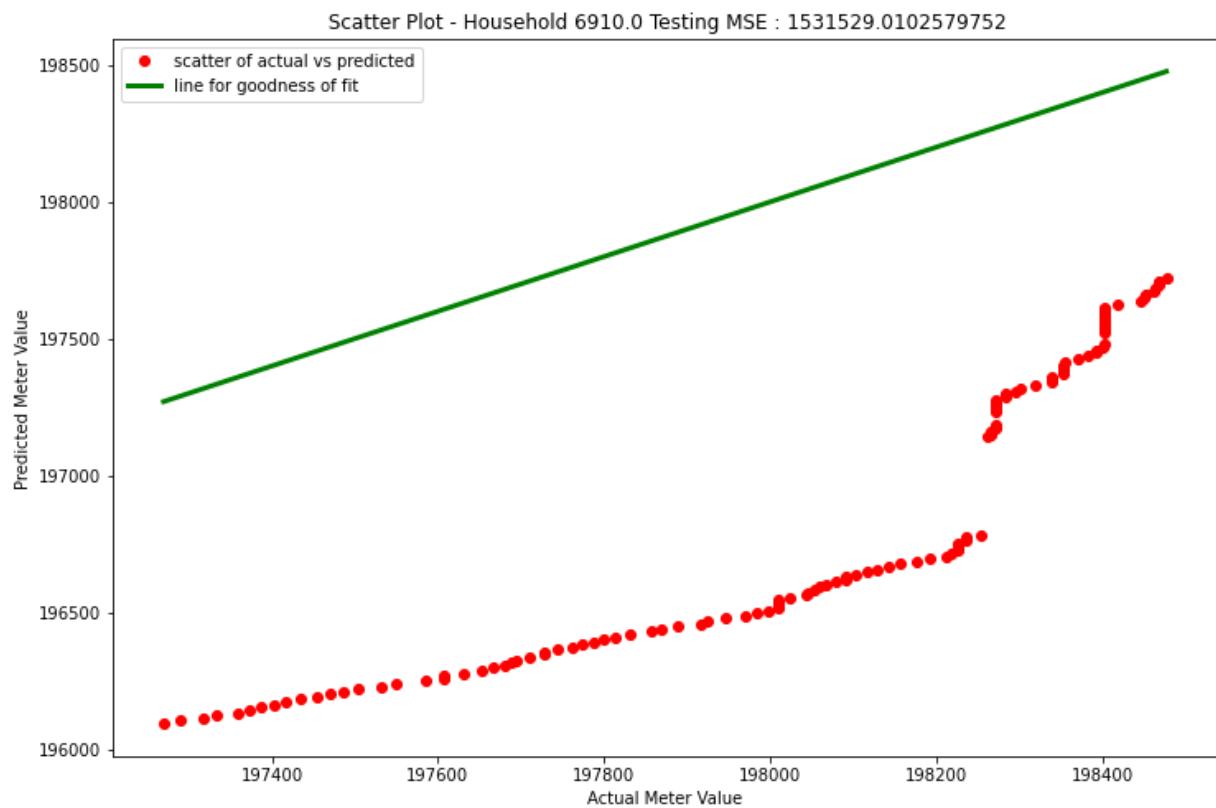


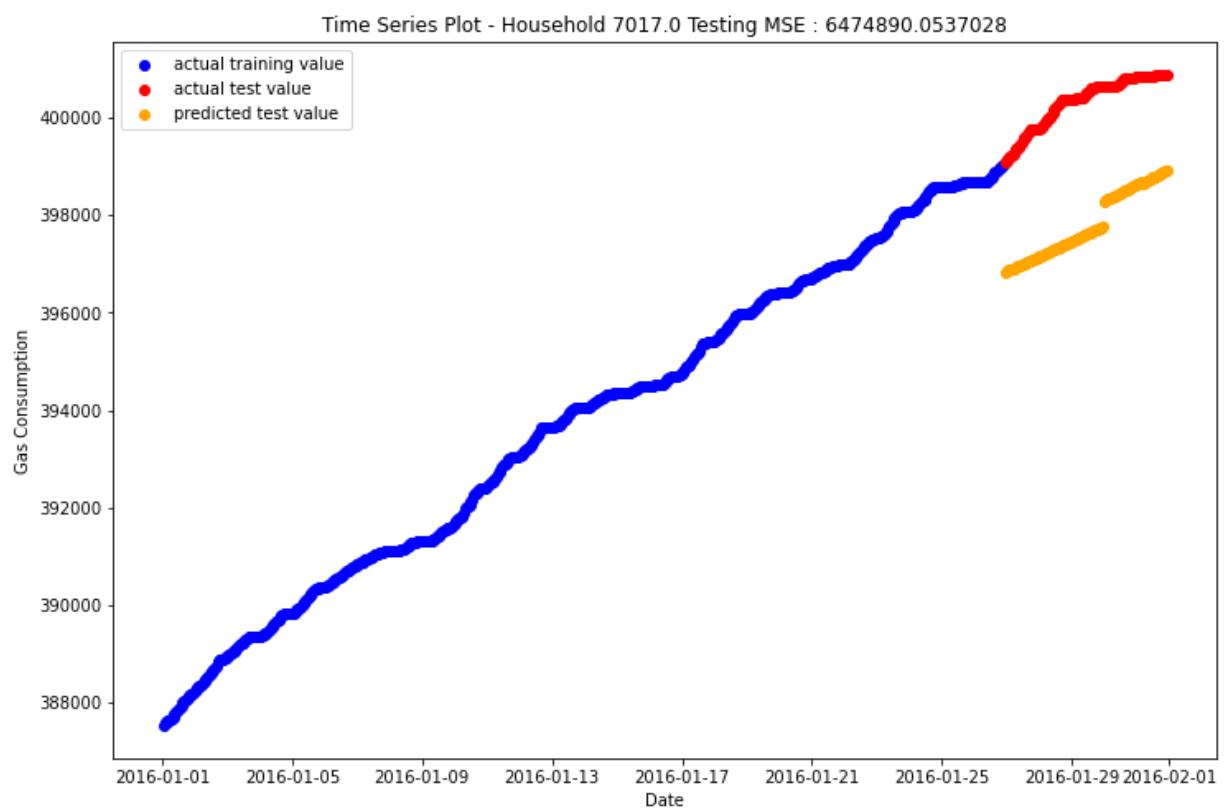
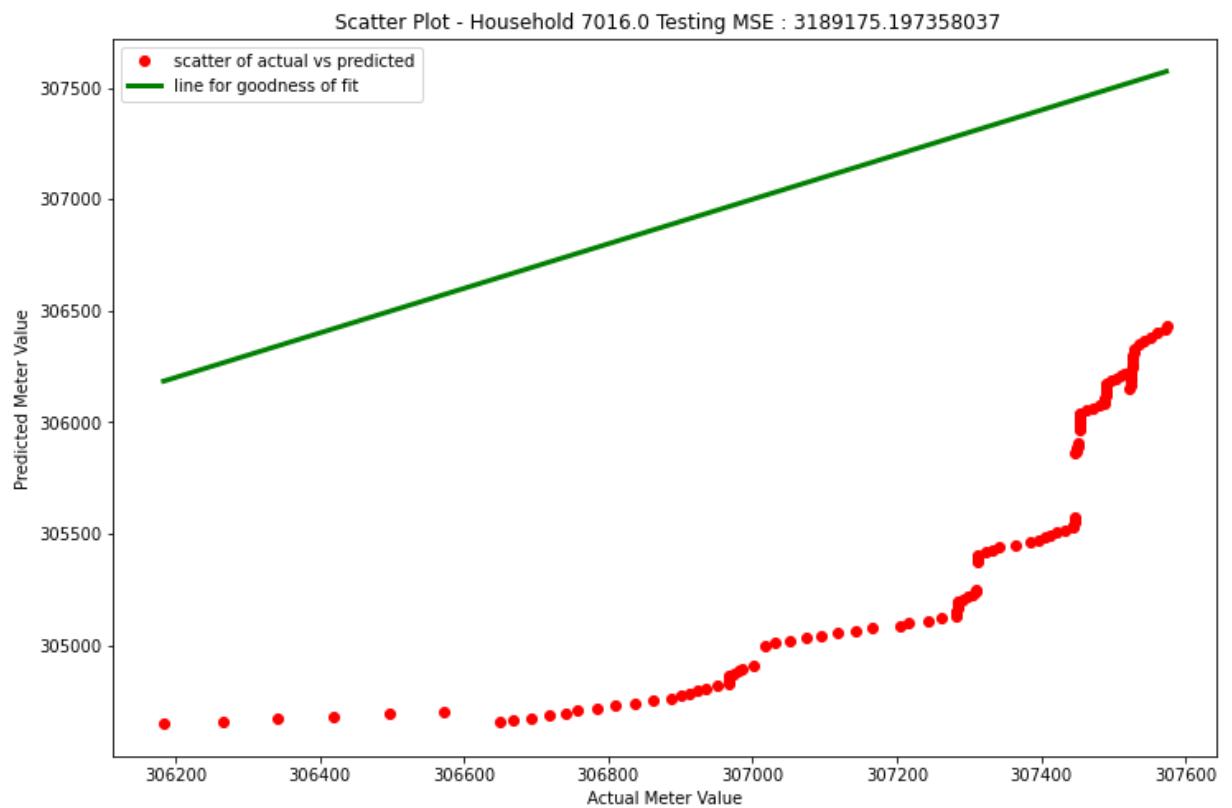
Scatter Plot - Household 6863.0 Testing MSE : 909918.2064557624

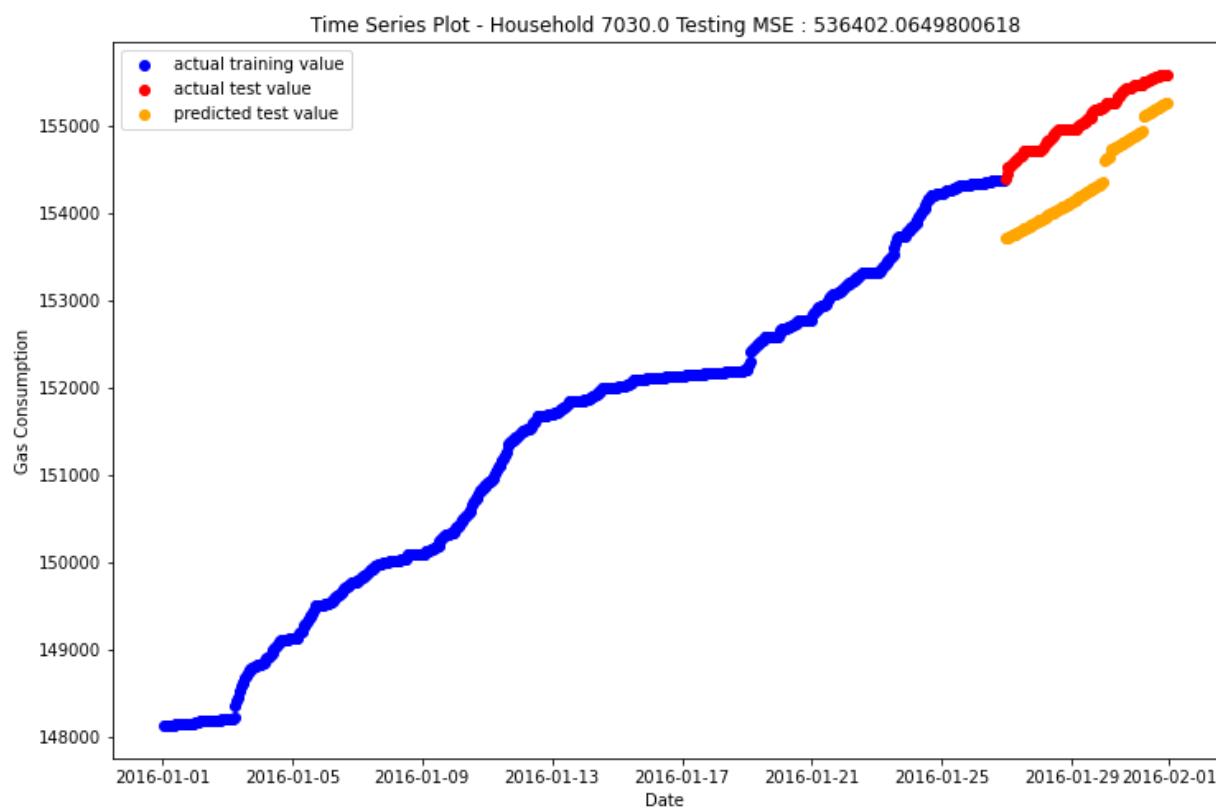
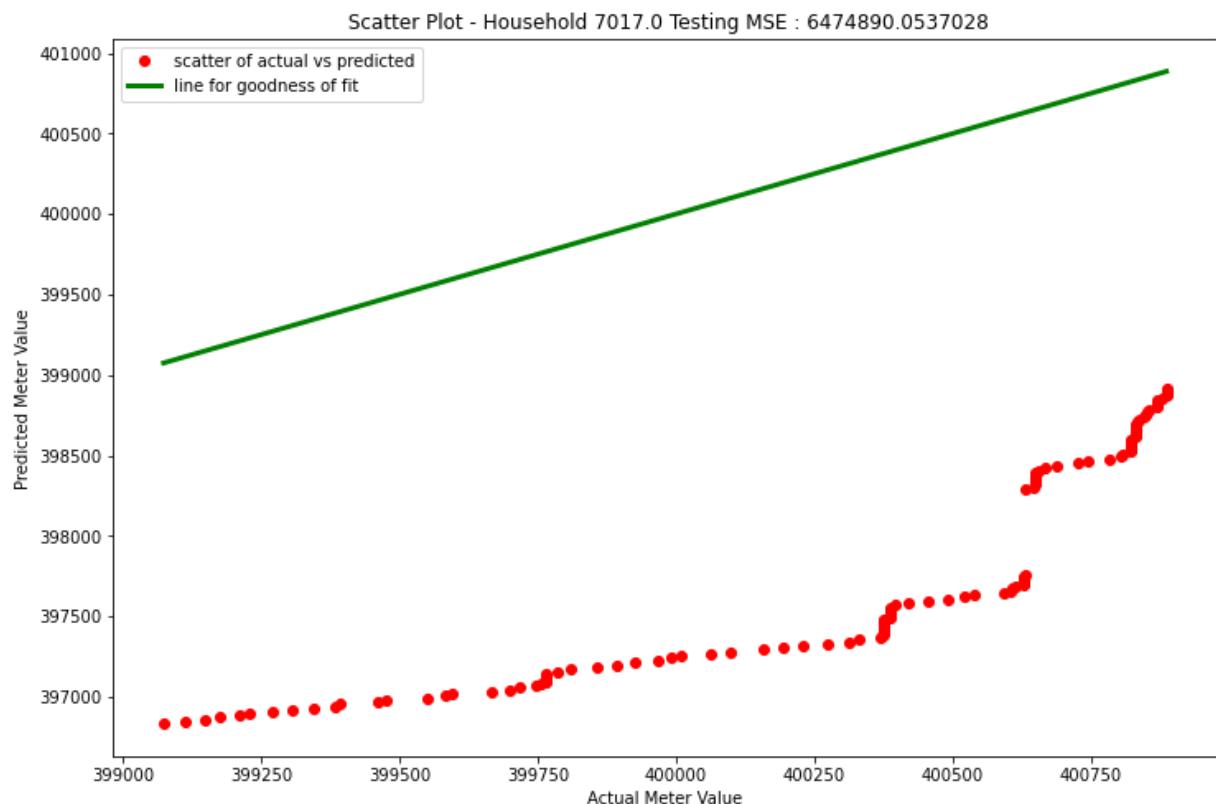


Time Series Plot - Household 6910.0 Testing MSE : 1531529.0102579752

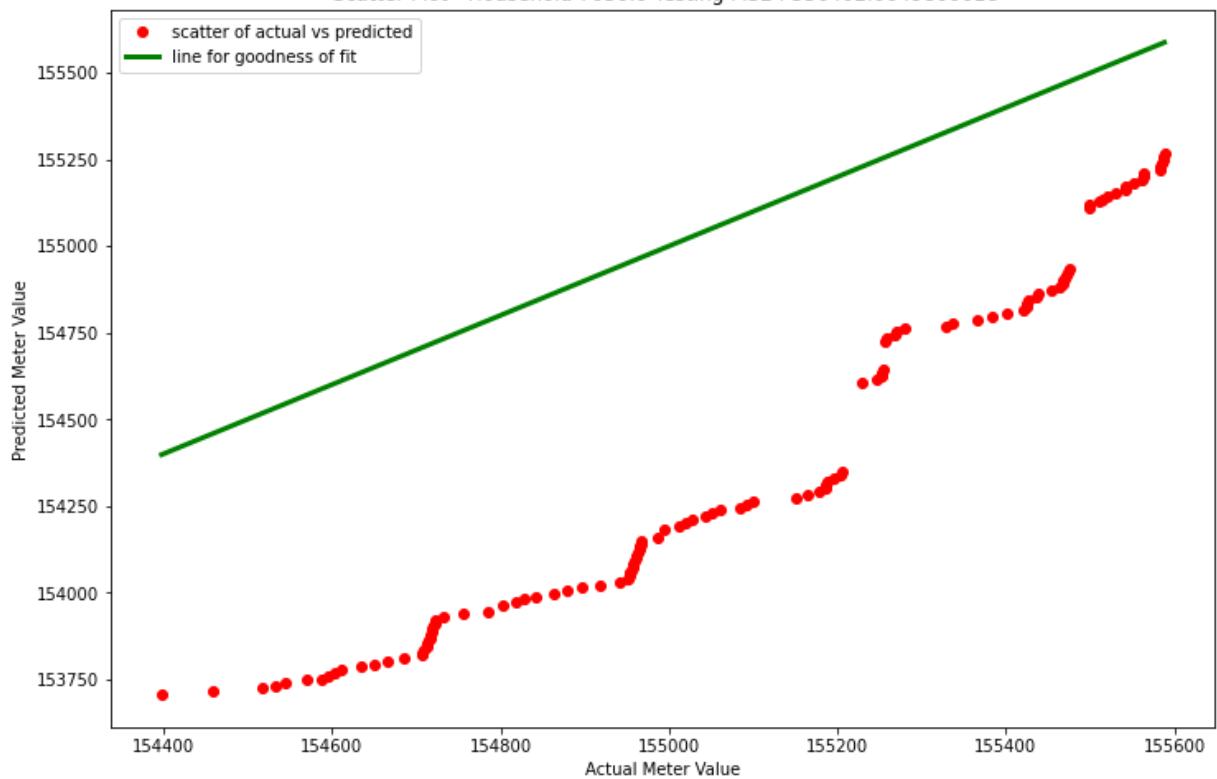




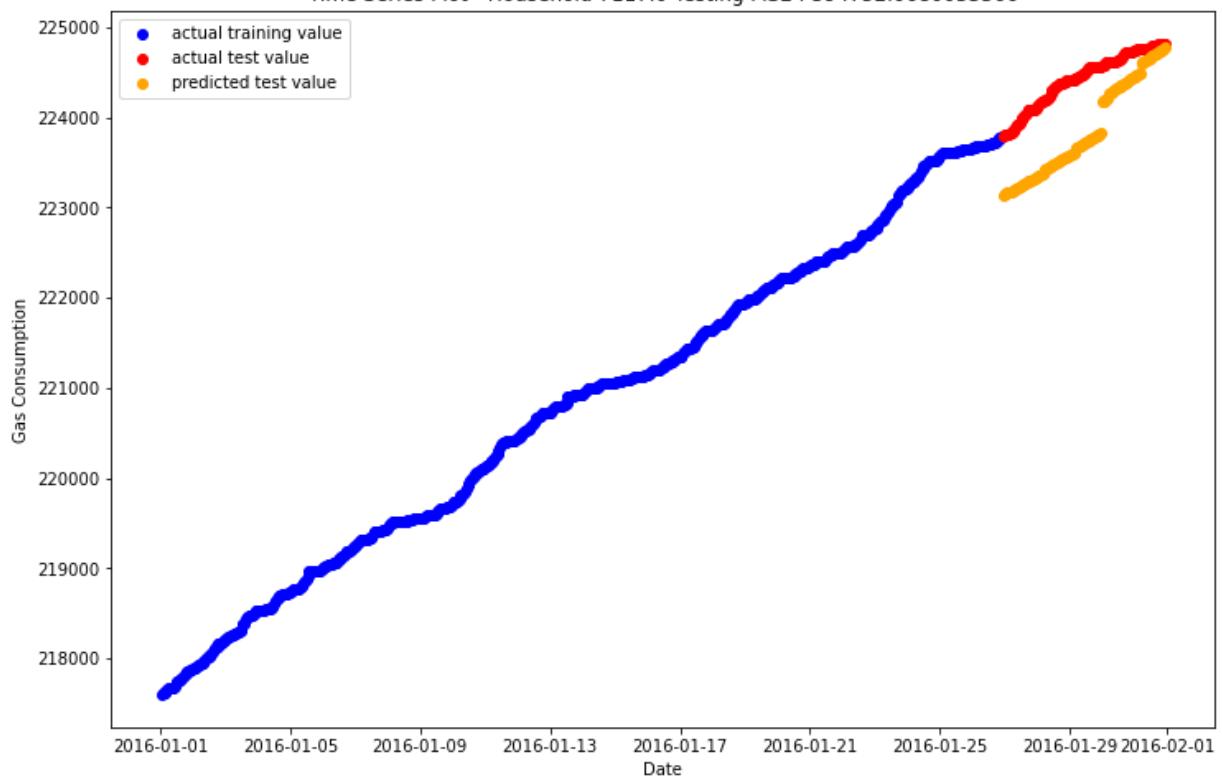


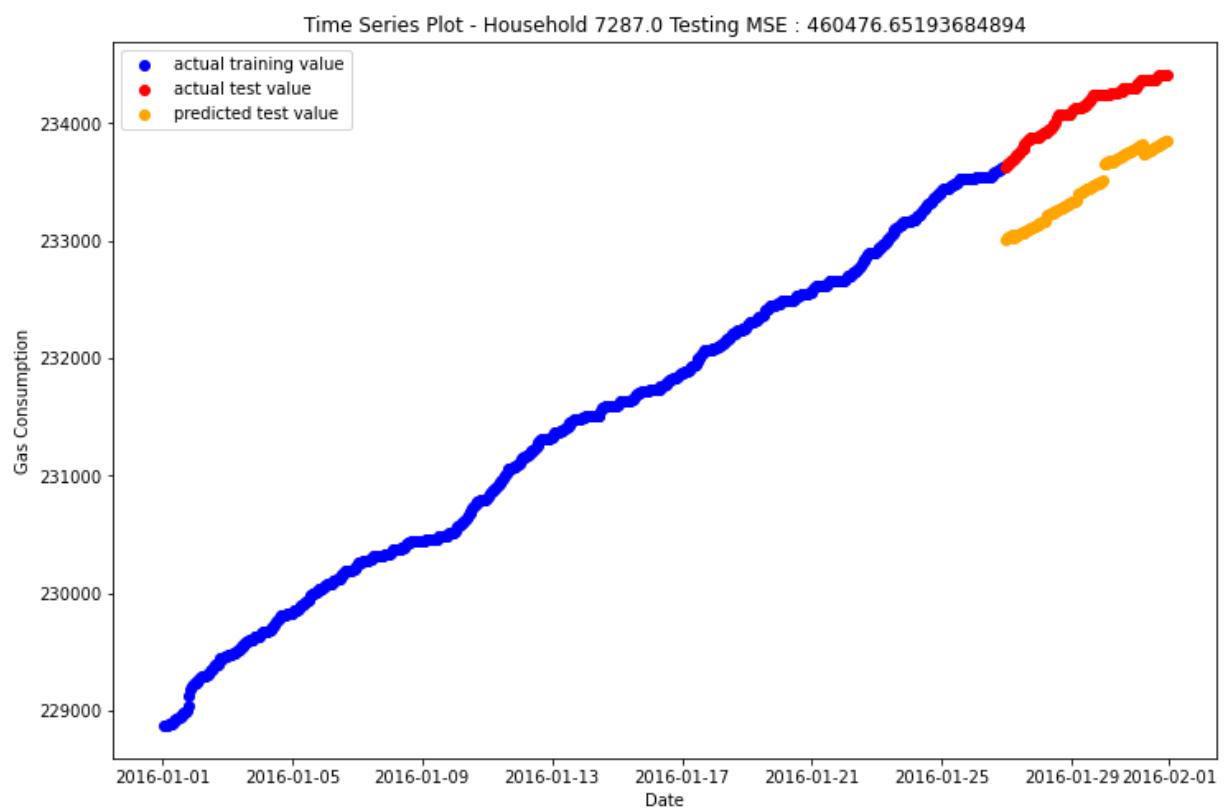
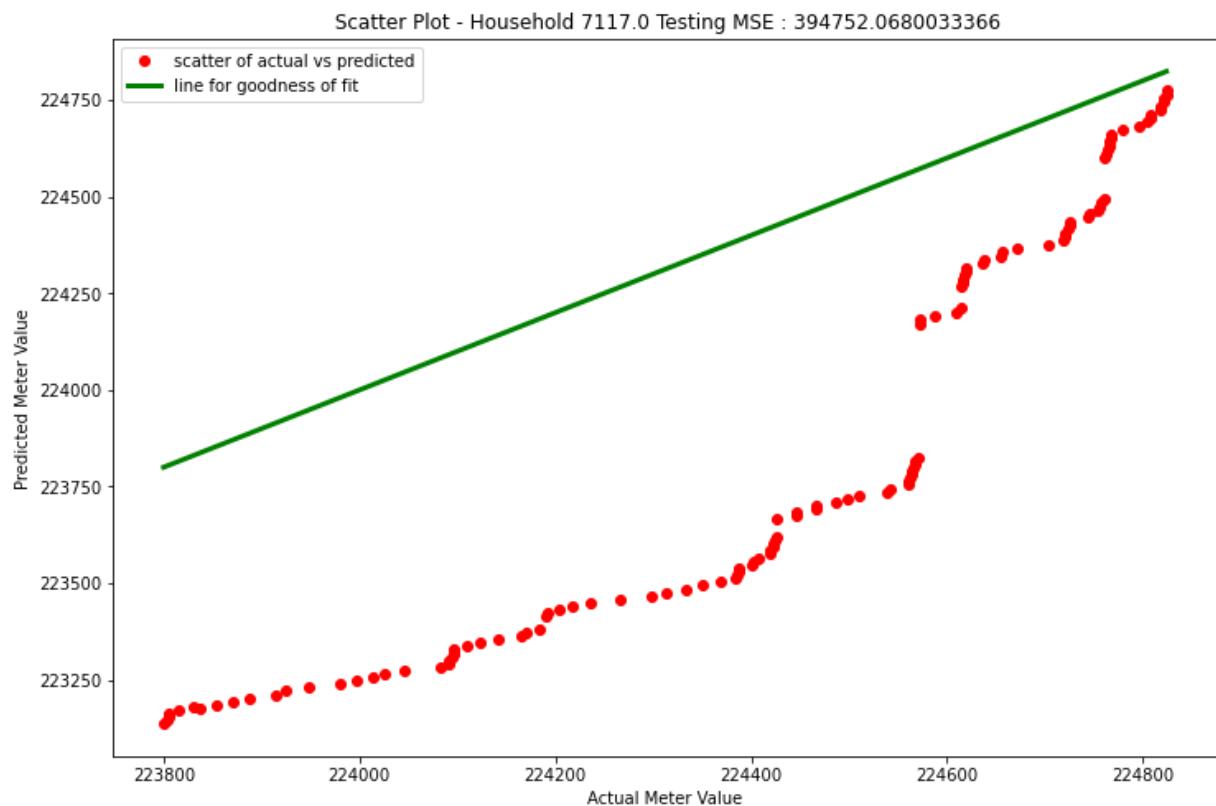


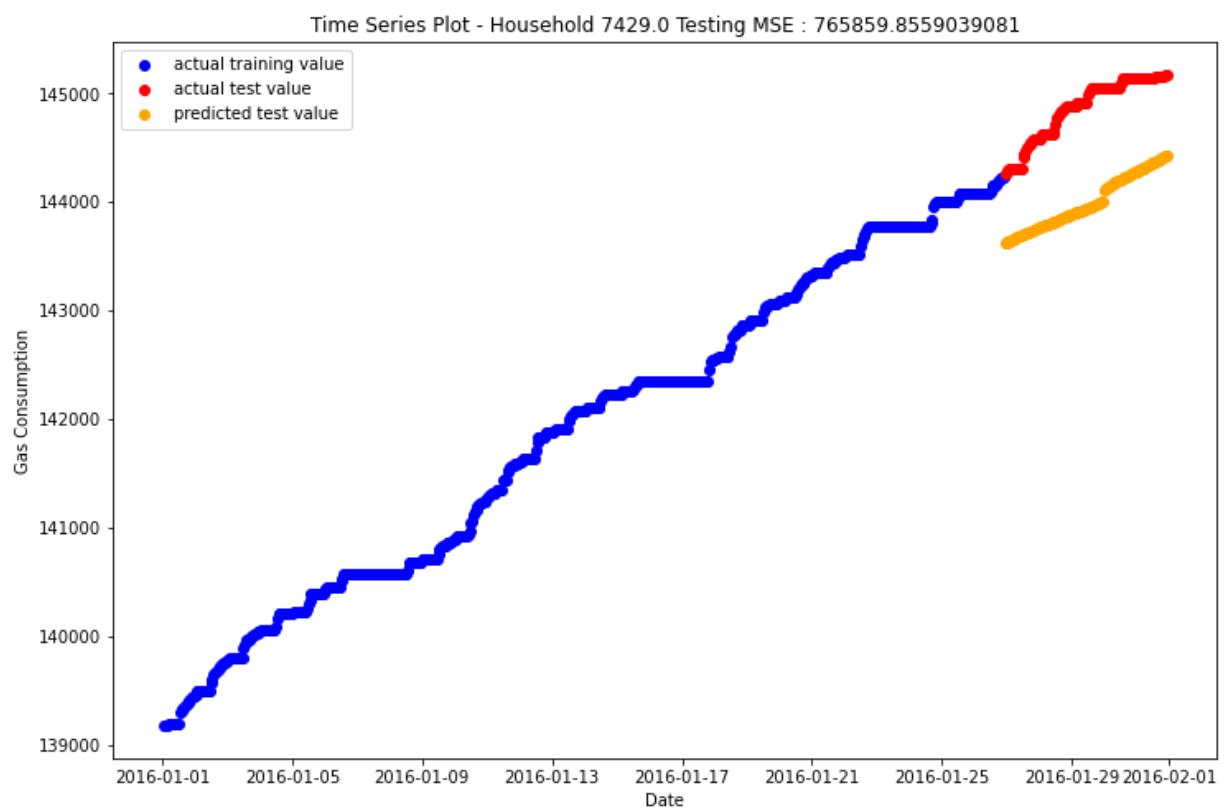
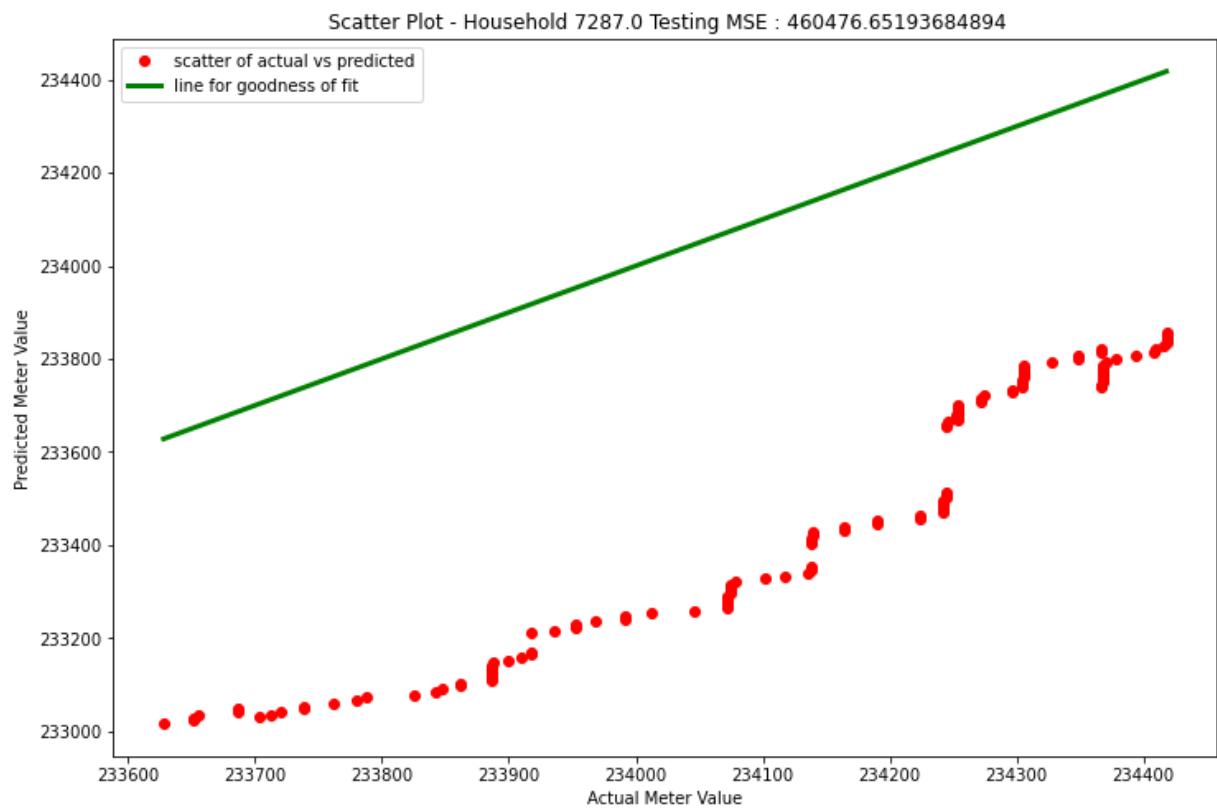
Scatter Plot - Household 7030.0 Testing MSE : 536402.0649800618

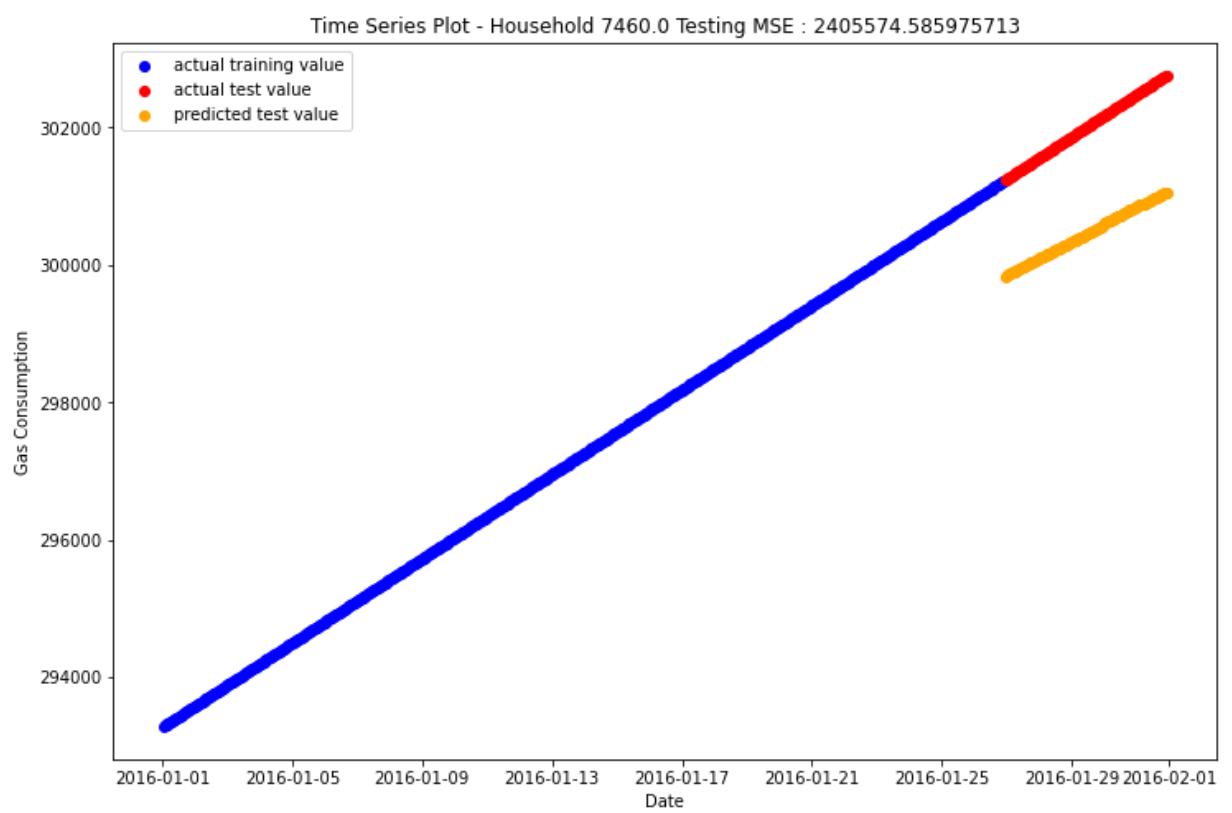
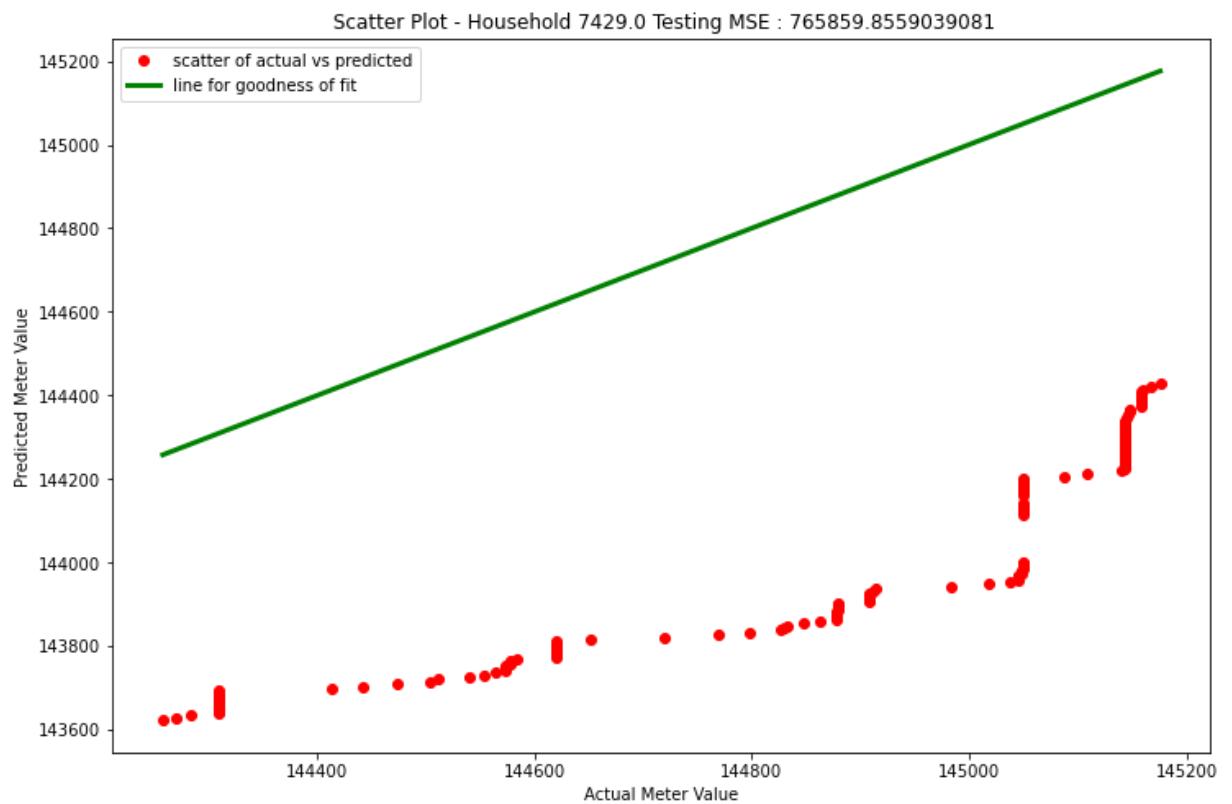


Time Series Plot - Household 7117.0 Testing MSE : 394752.0680033366

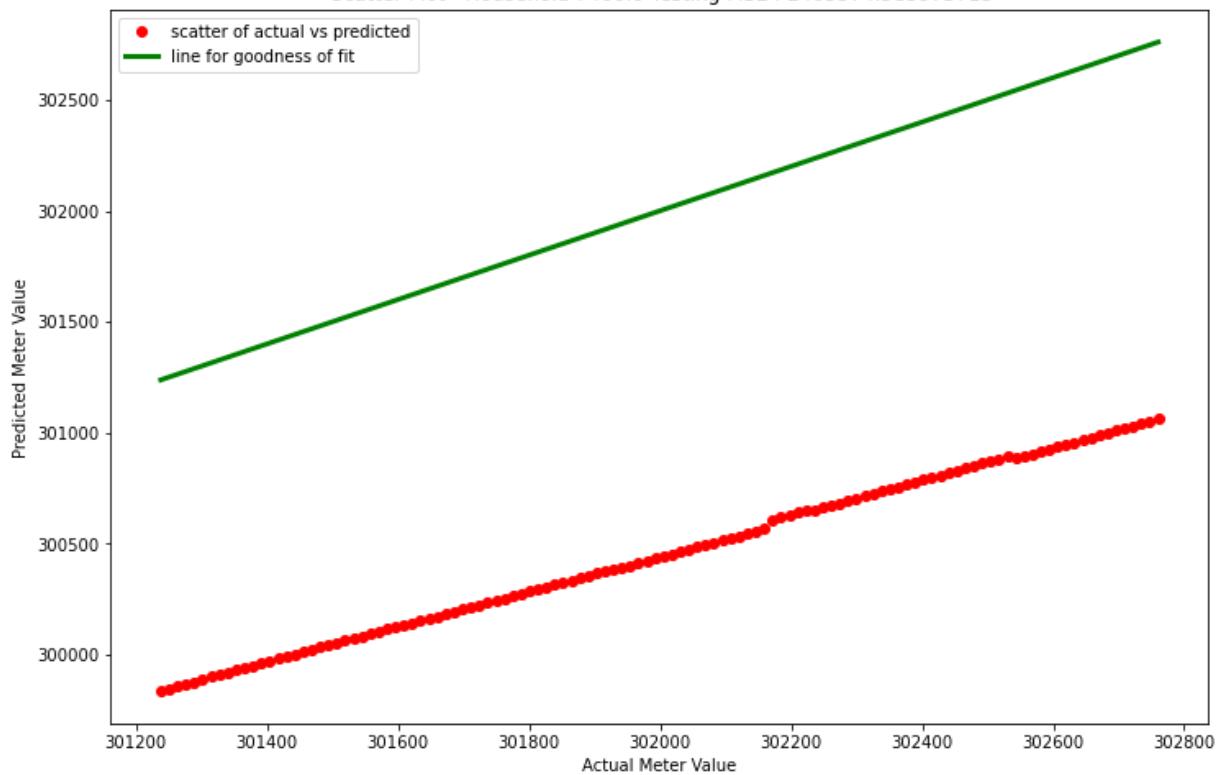




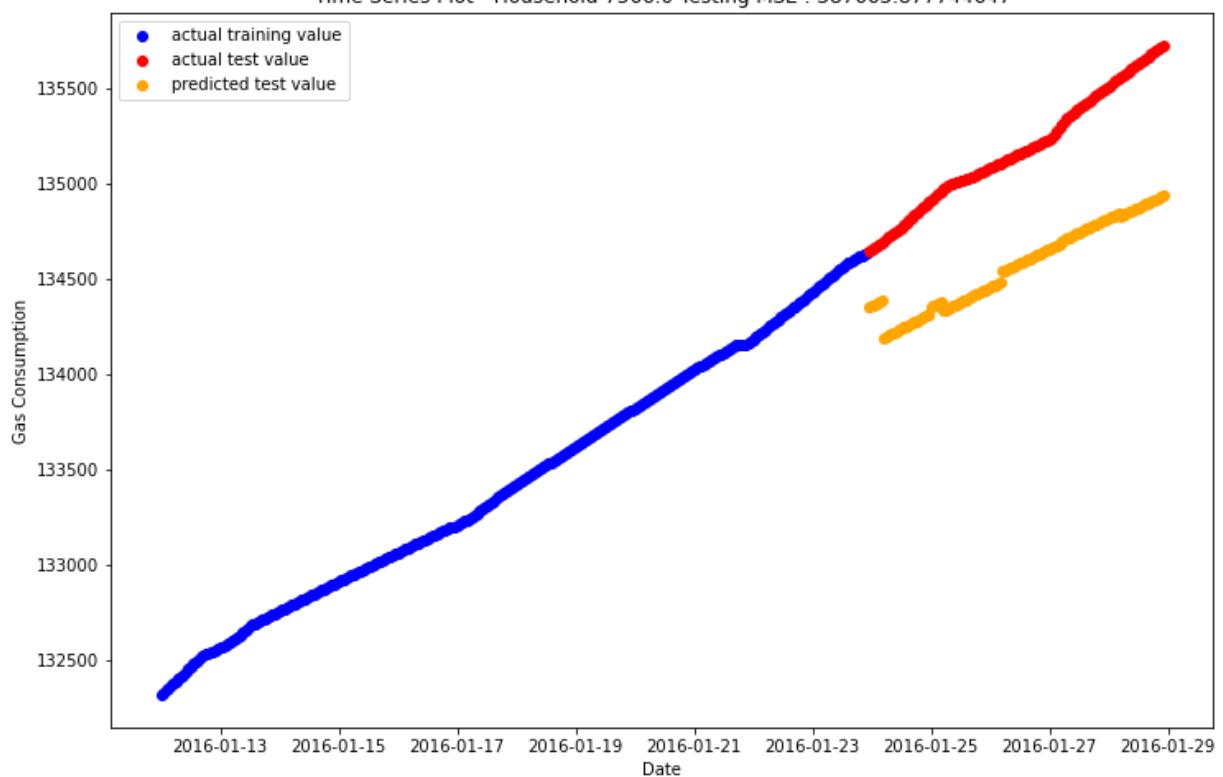


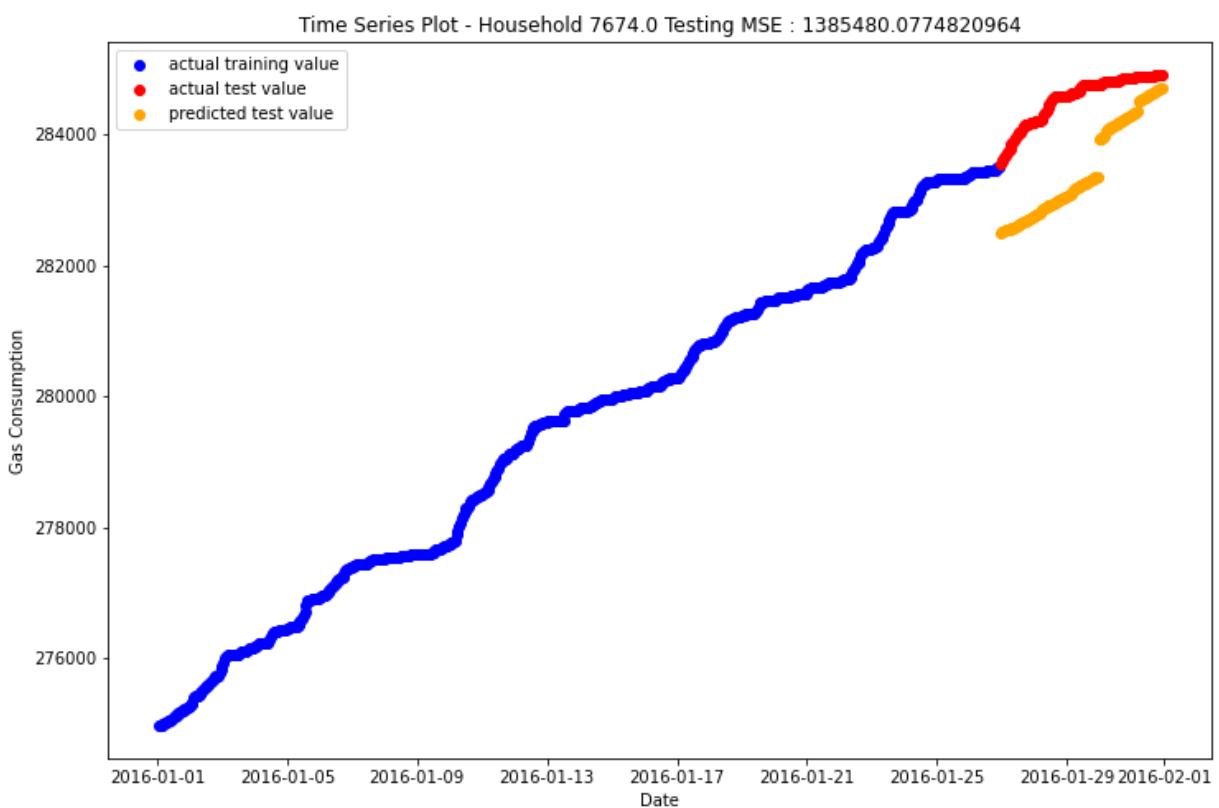
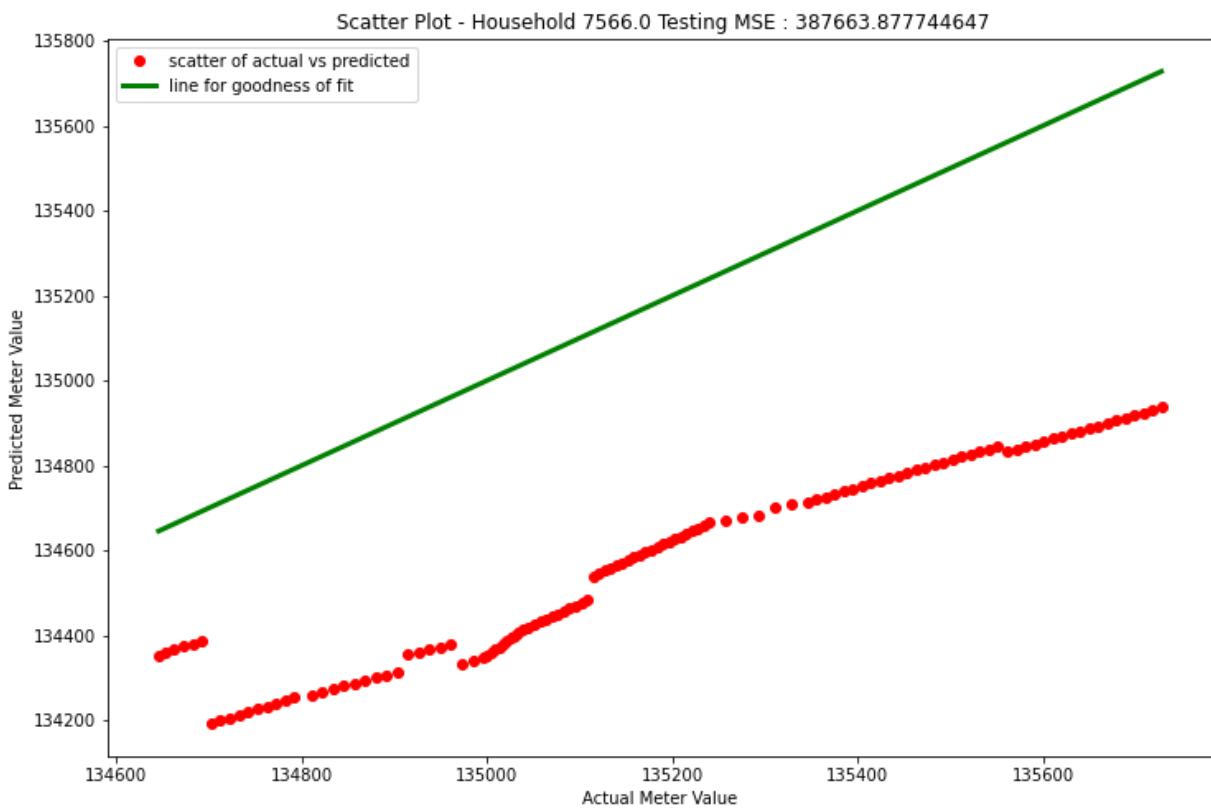


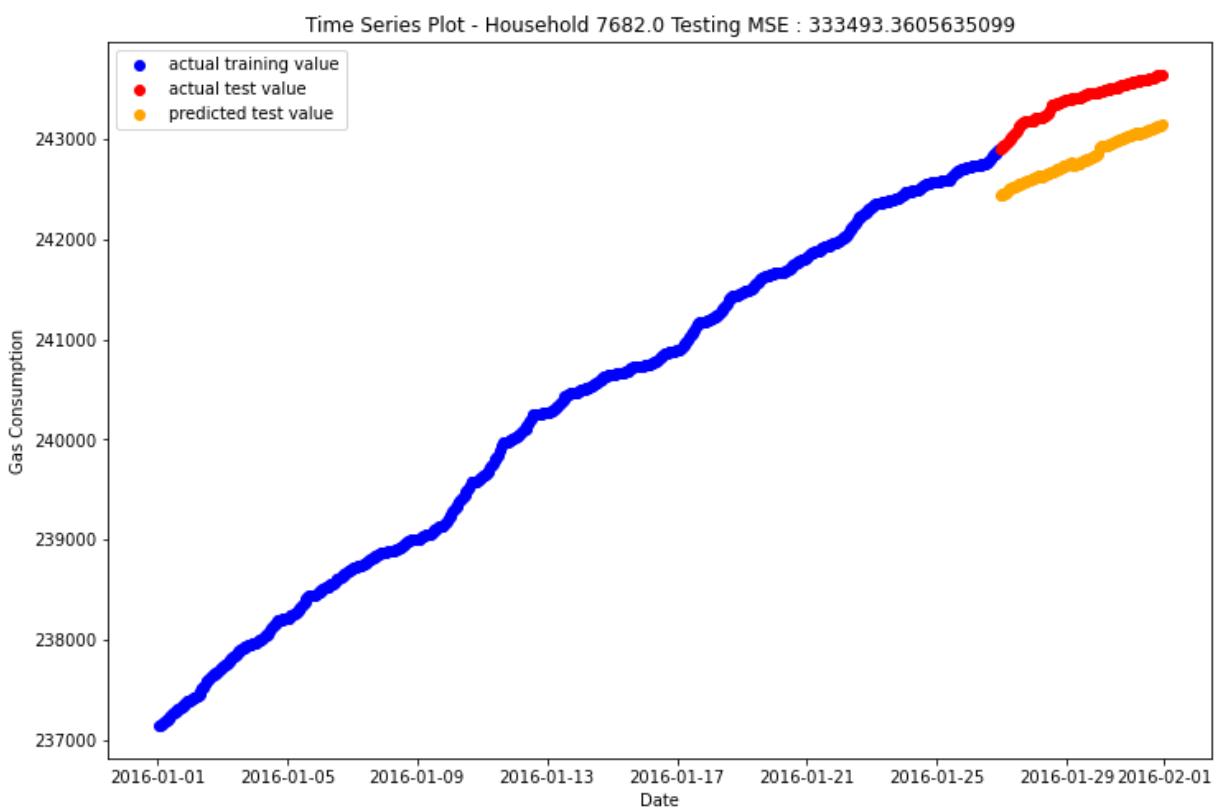
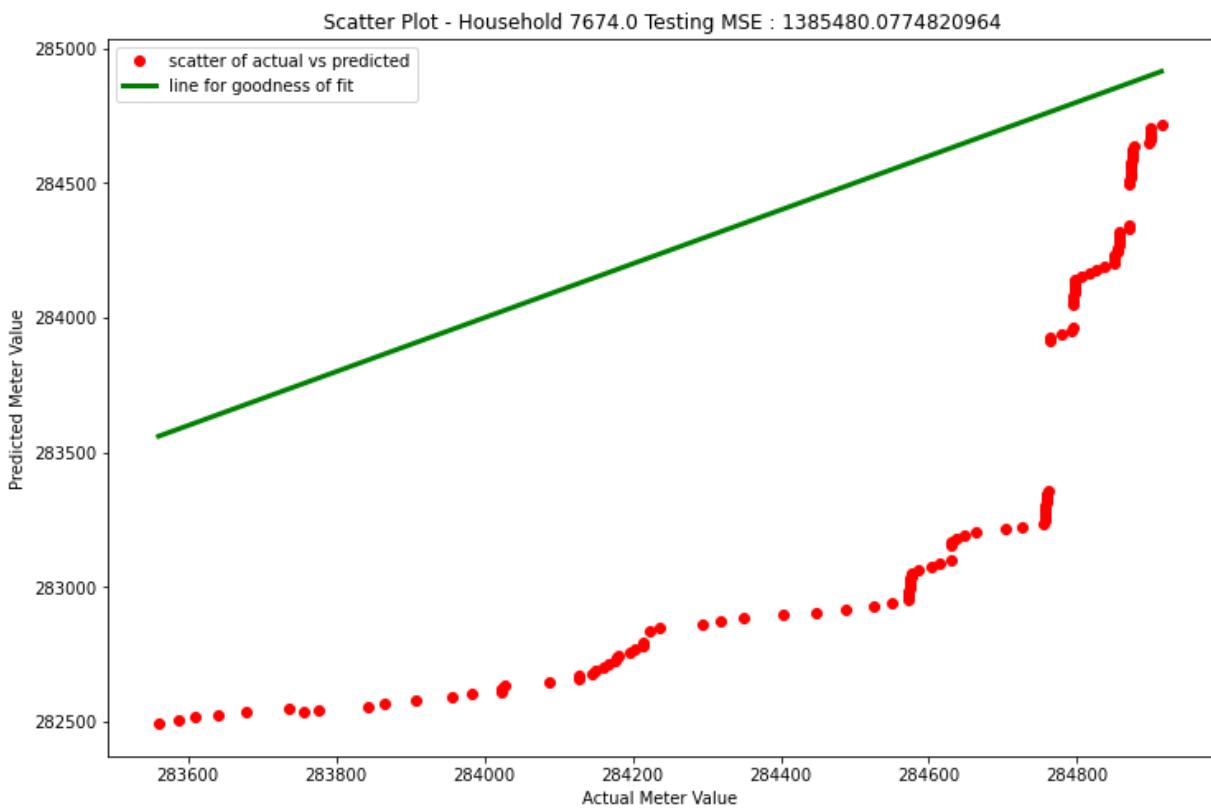
Scatter Plot - Household 7460.0 Testing MSE : 2405574.585975713

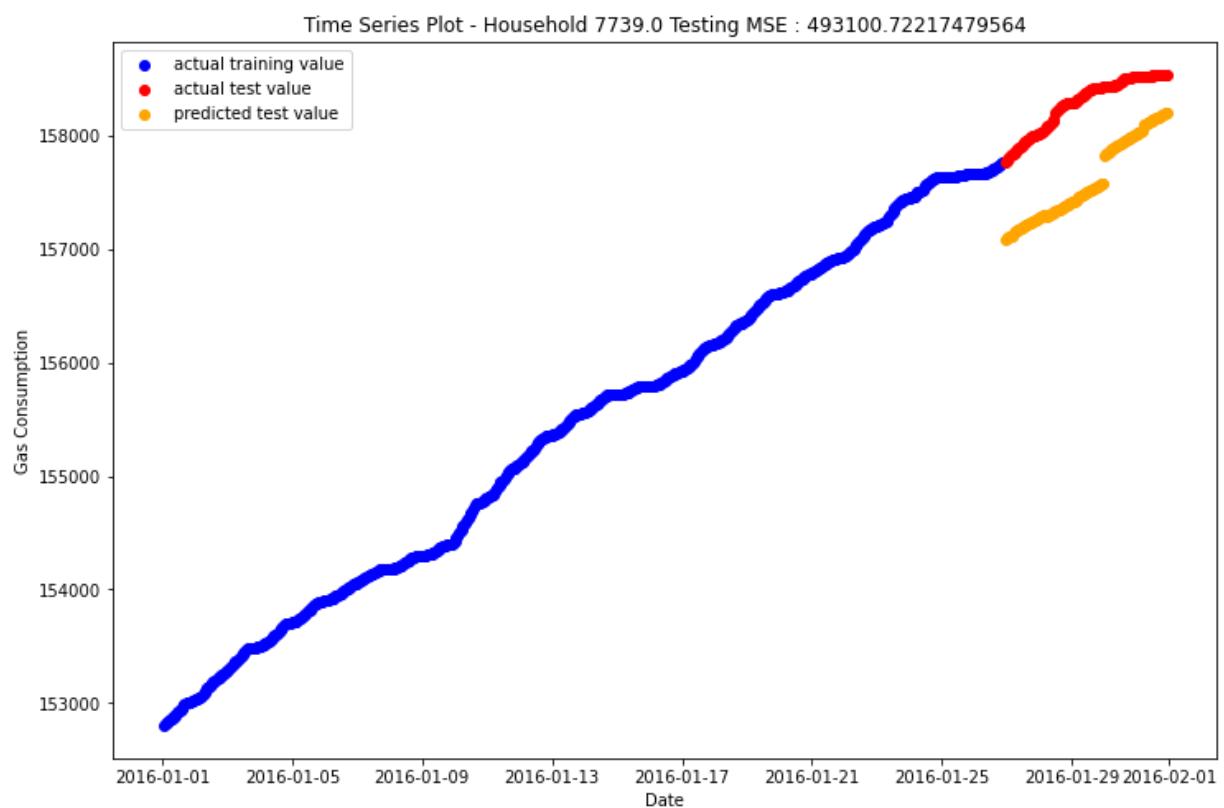
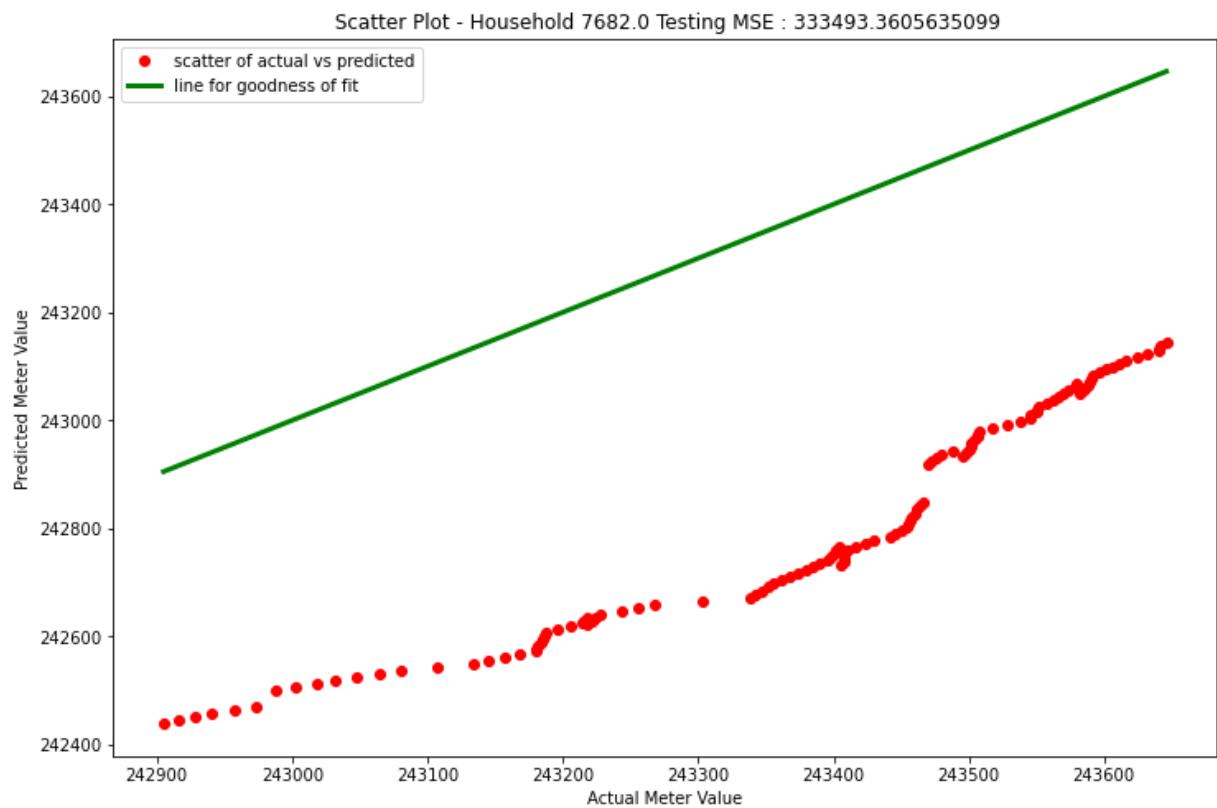


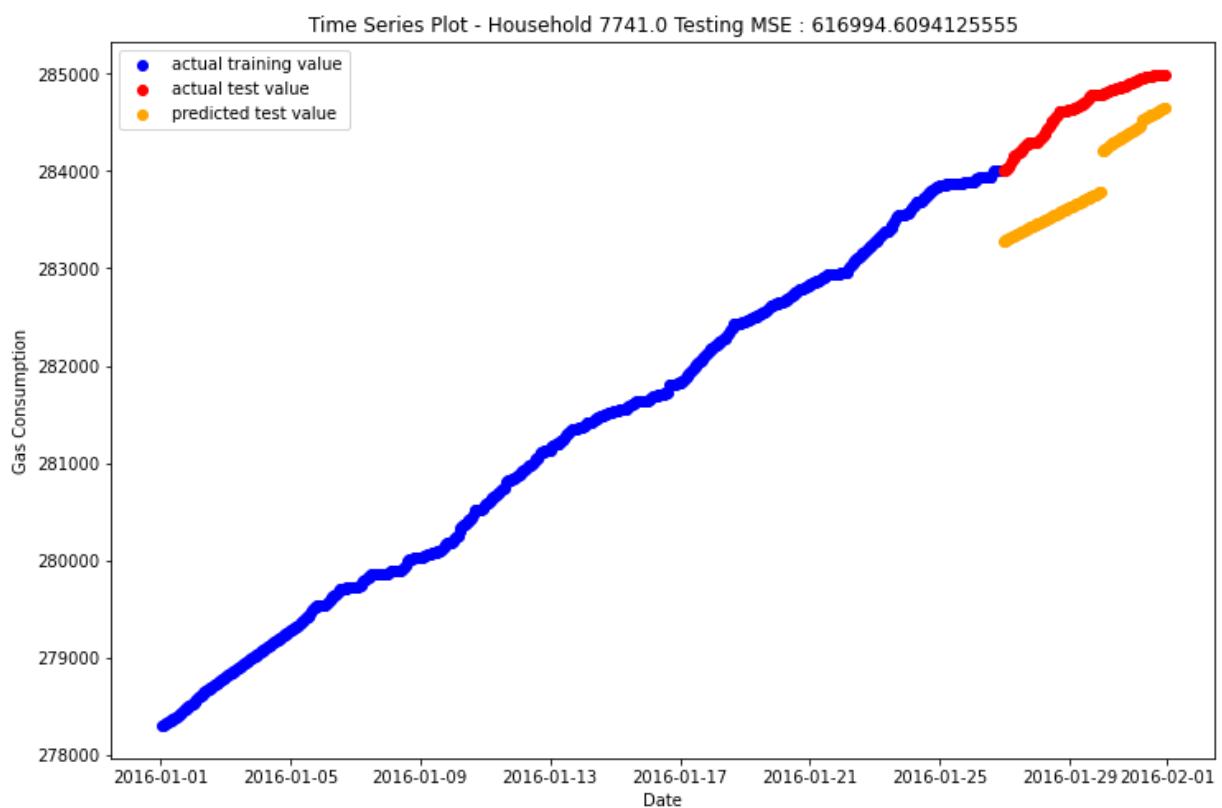
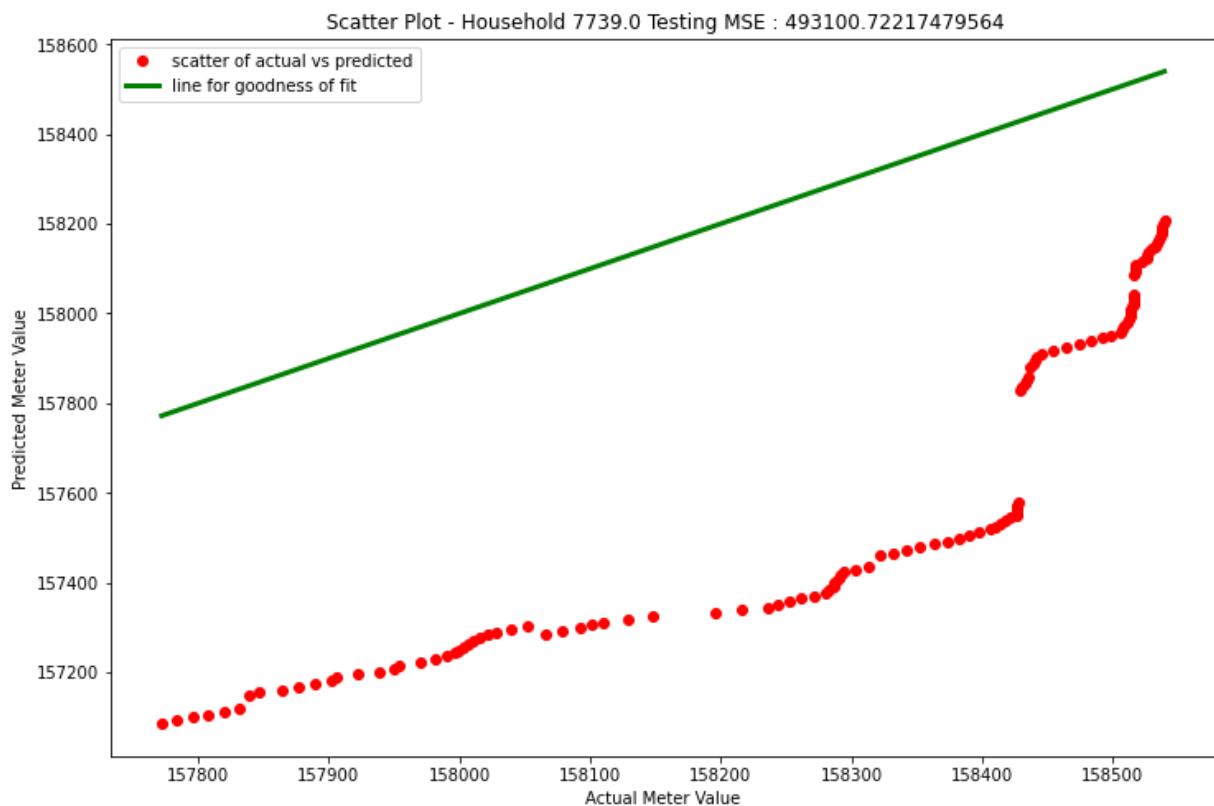
Time Series Plot - Household 7566.0 Testing MSE : 387663.877744647

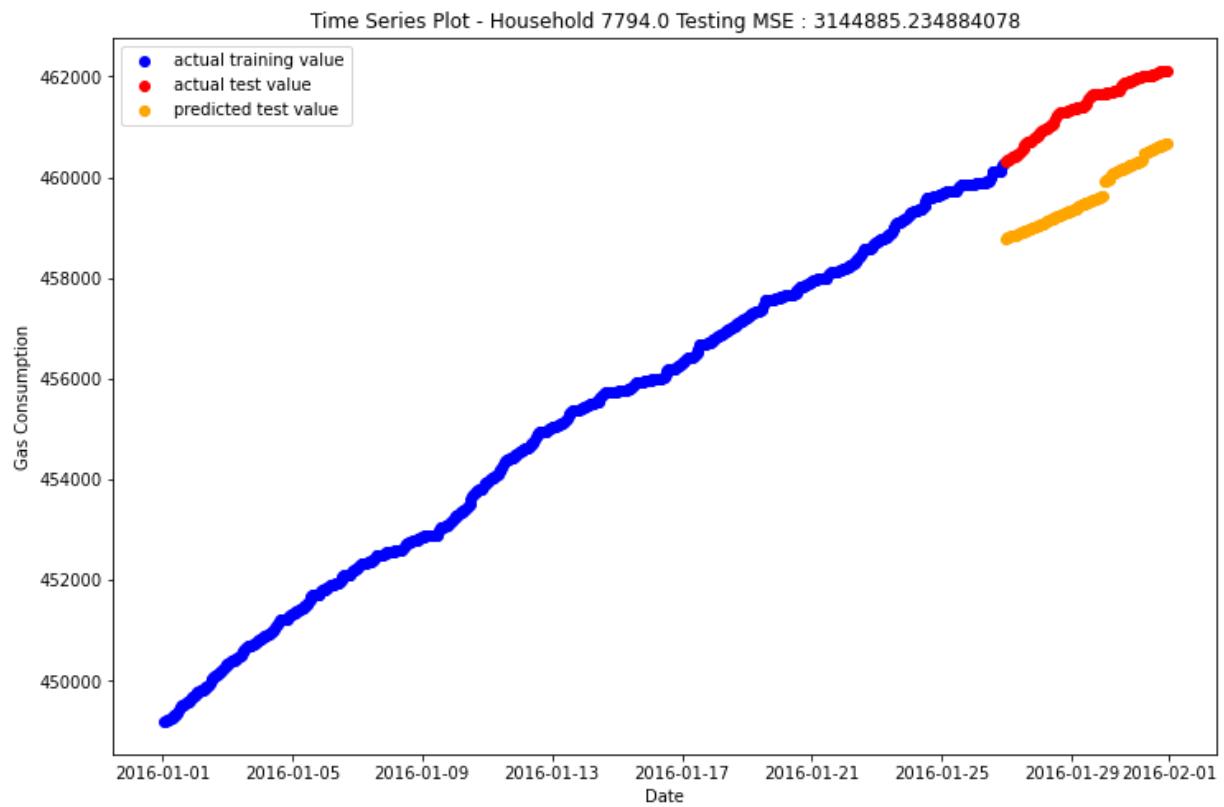
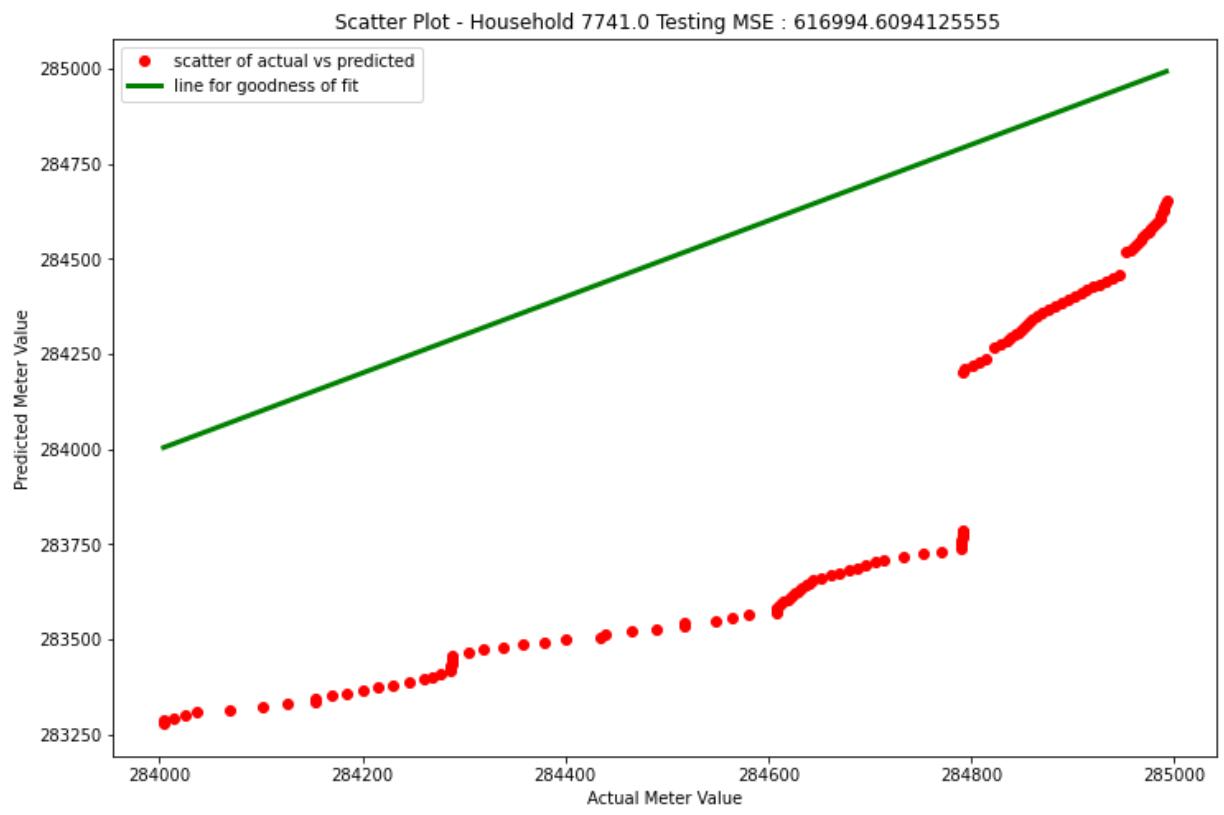


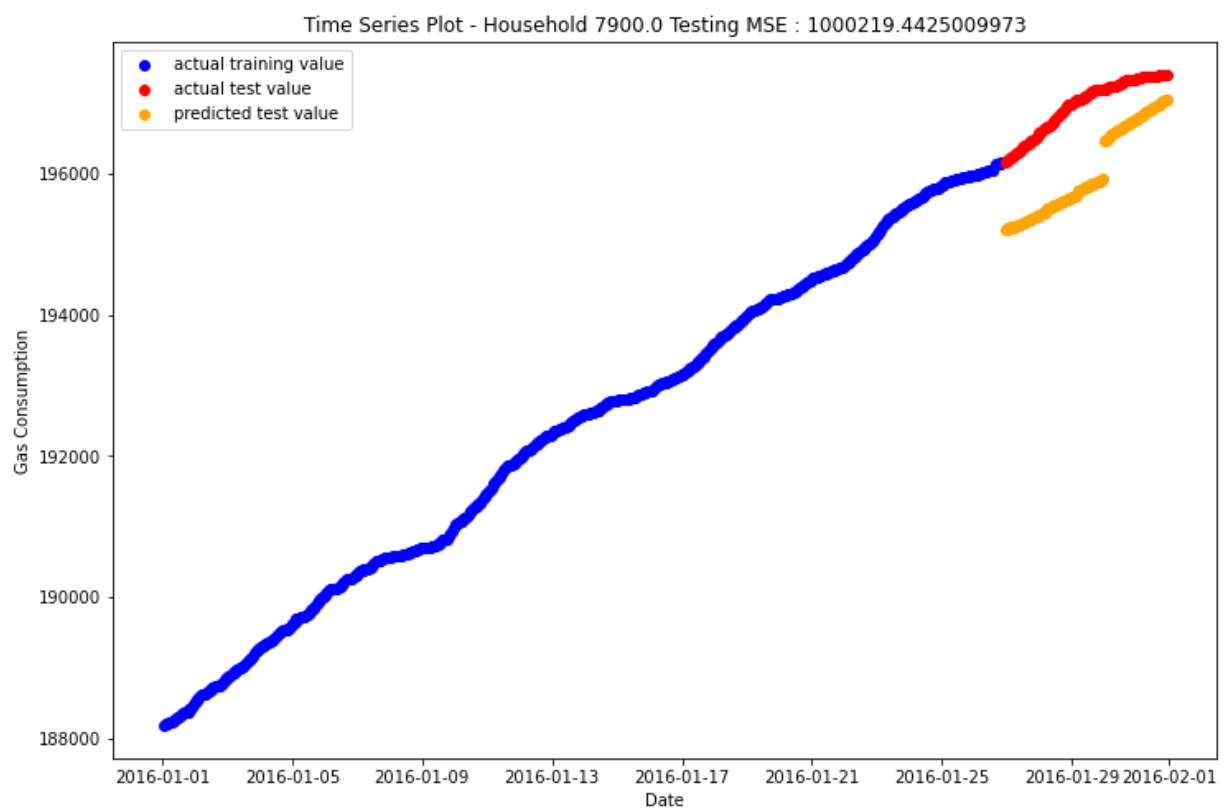
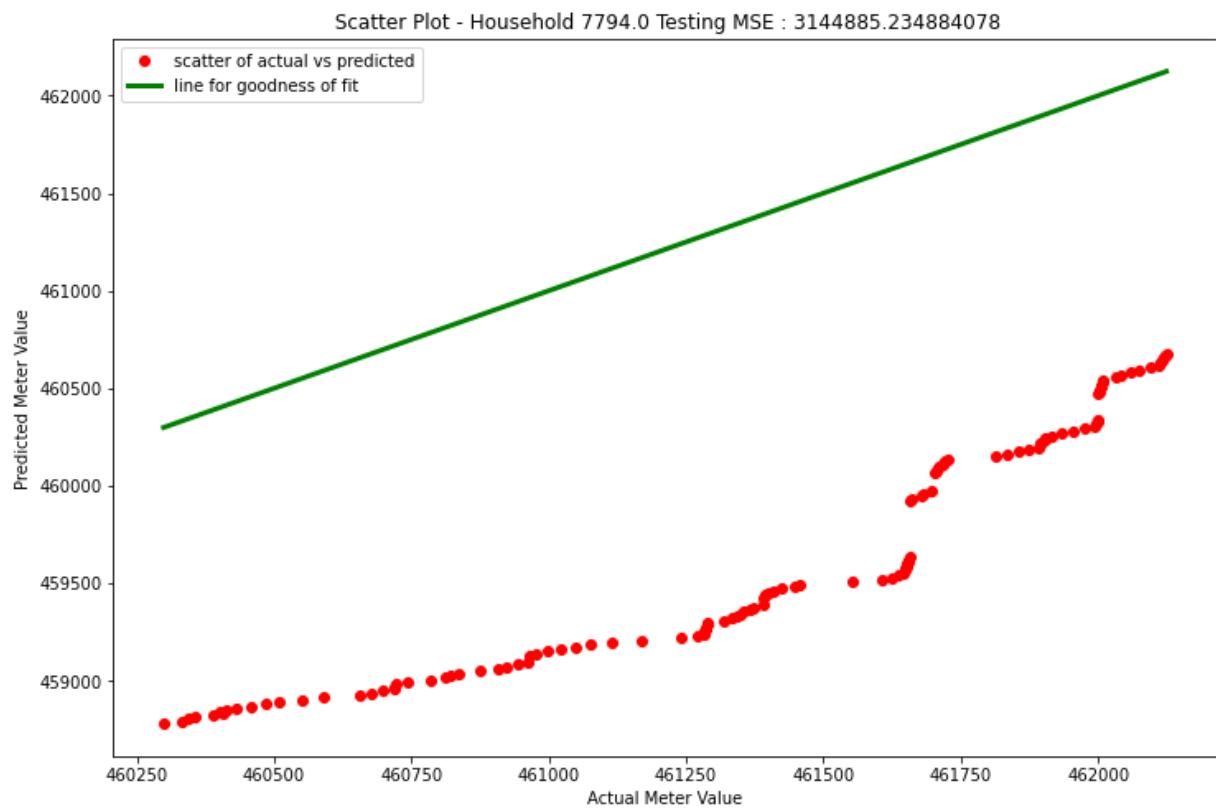


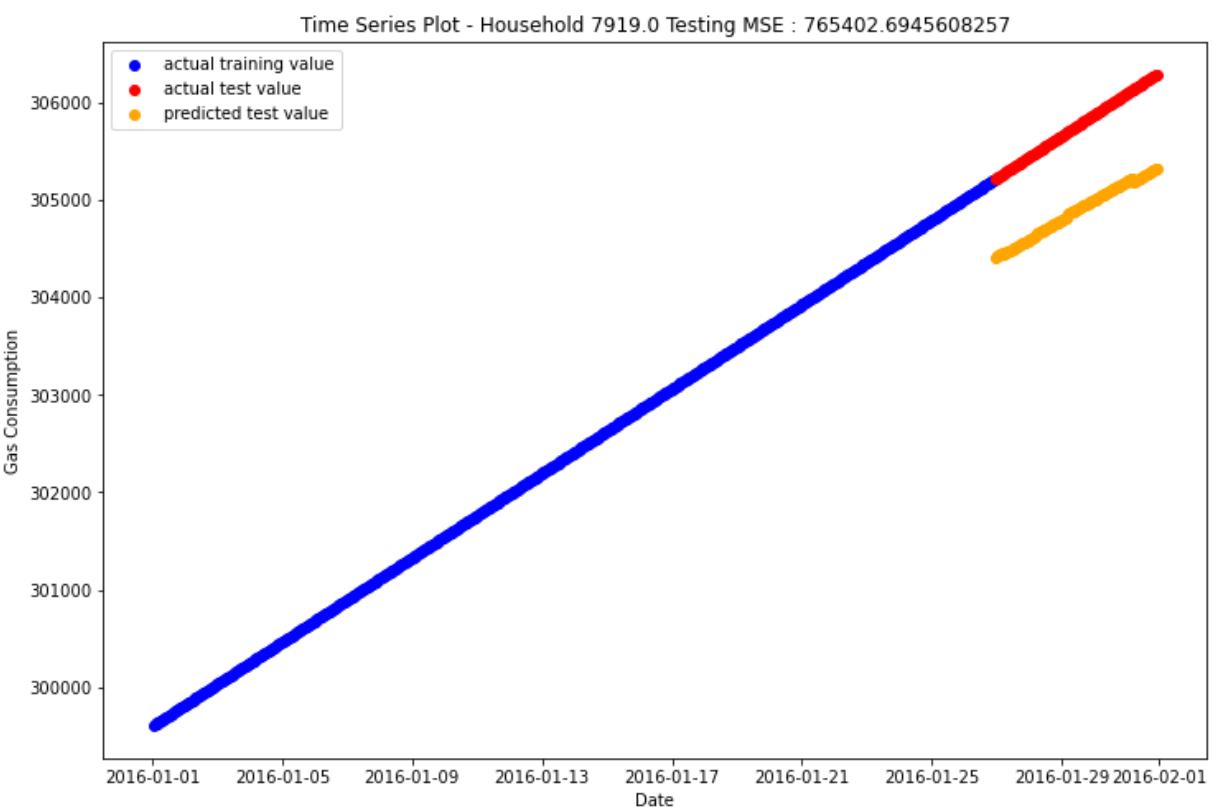
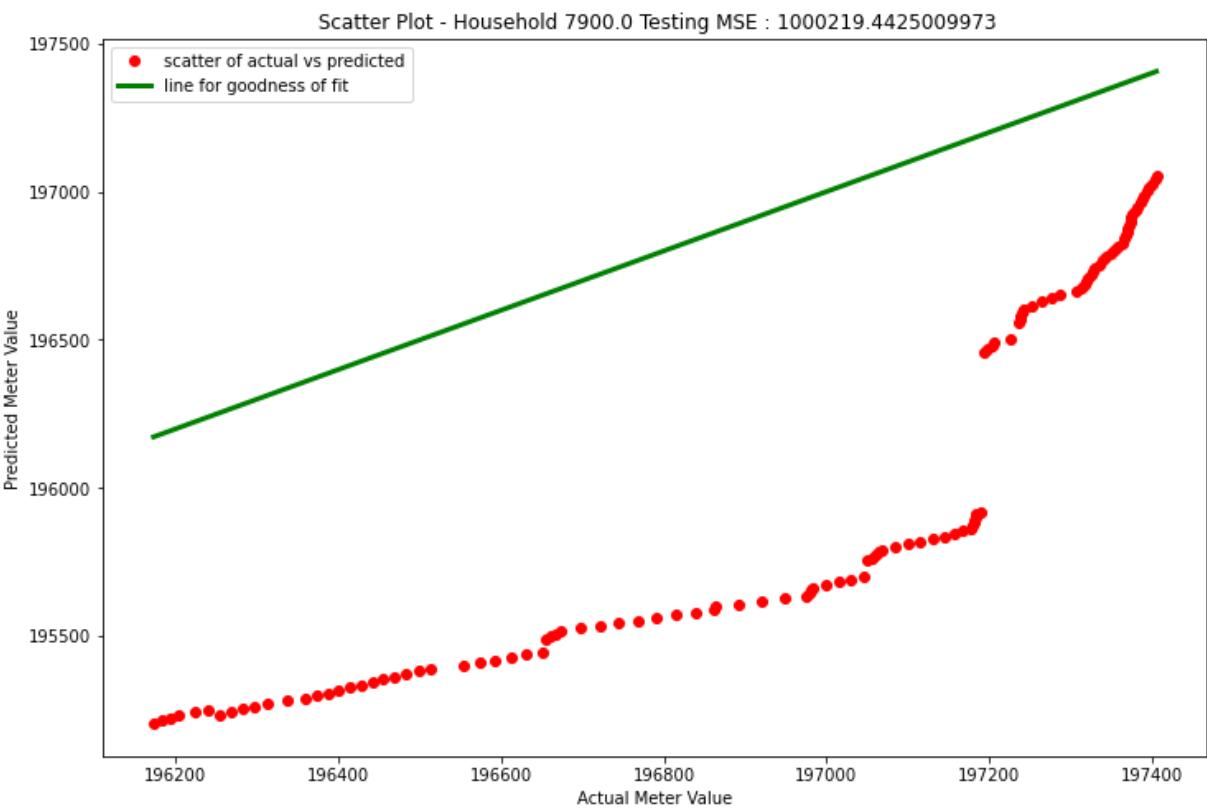


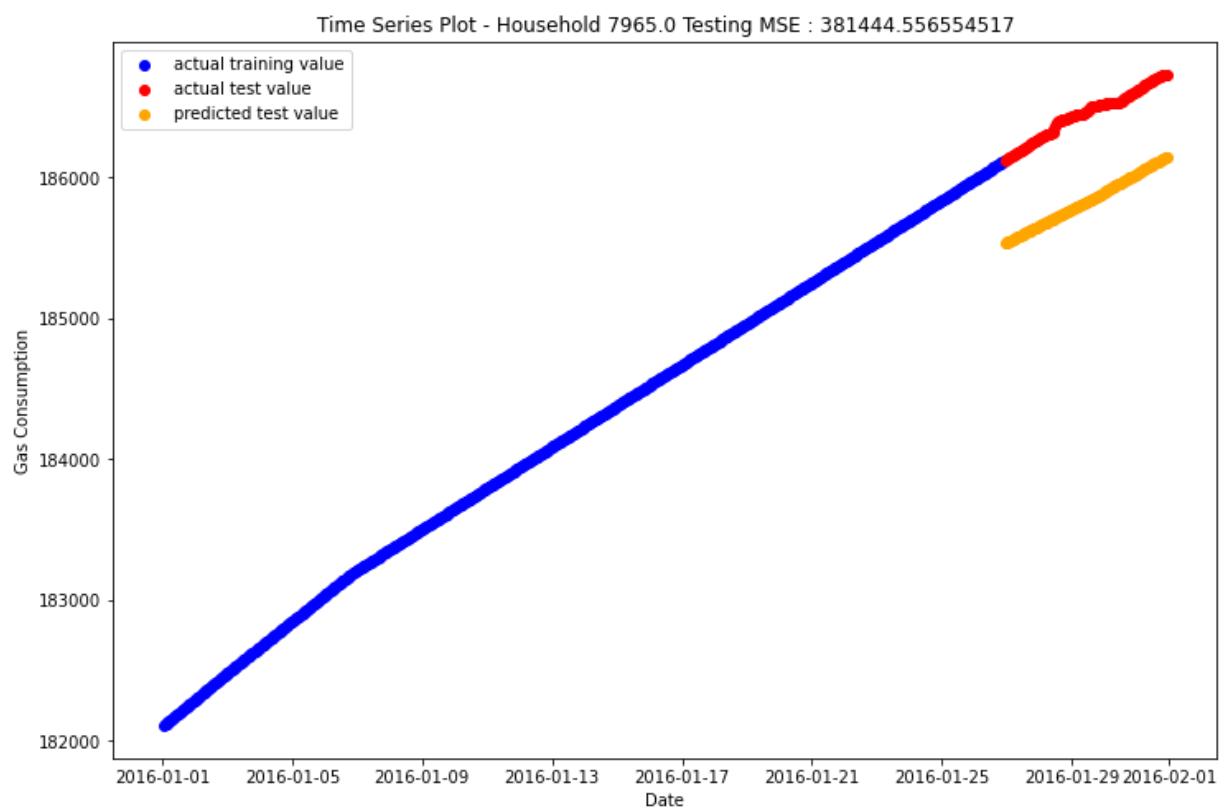
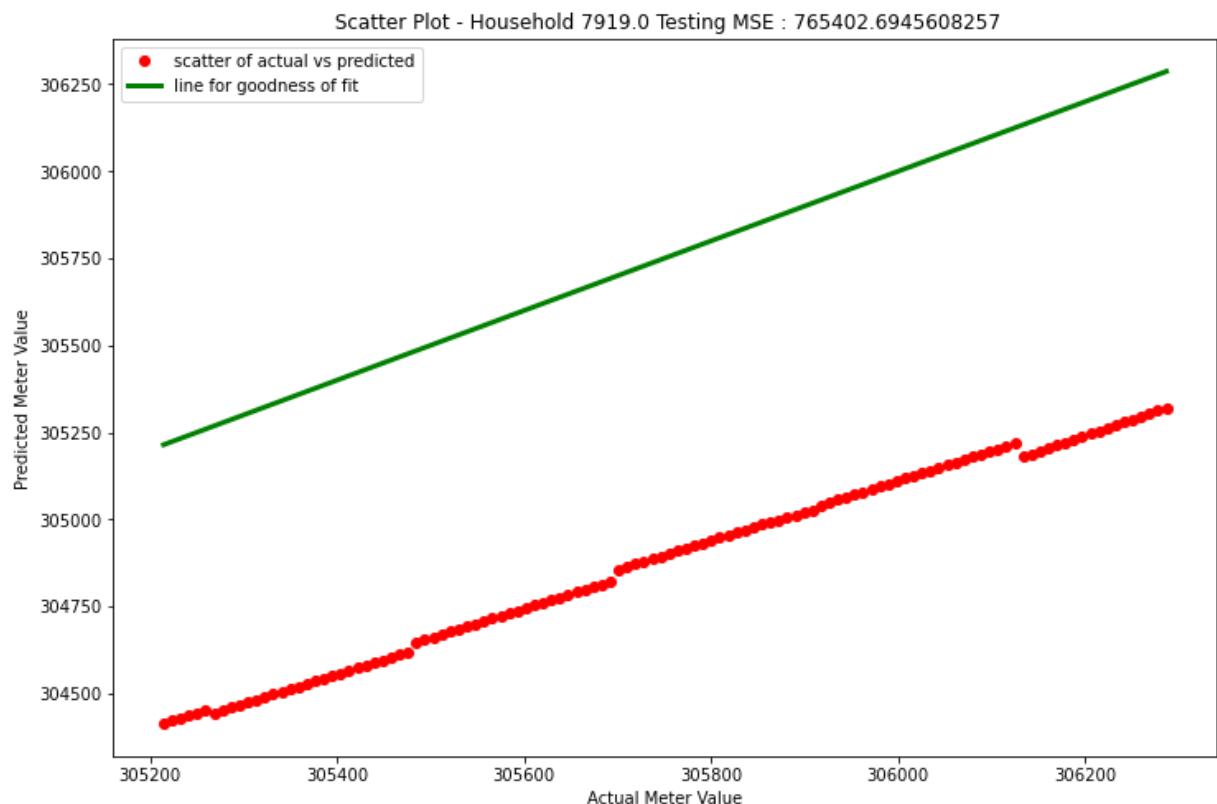




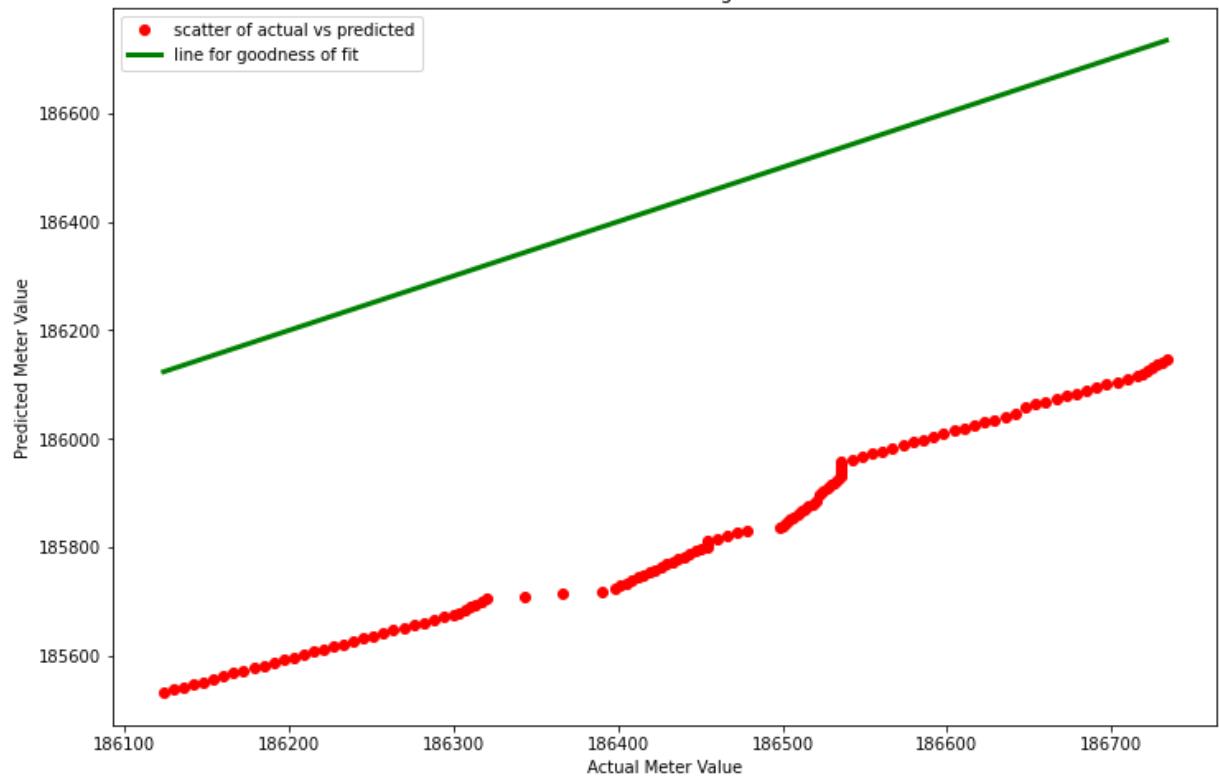




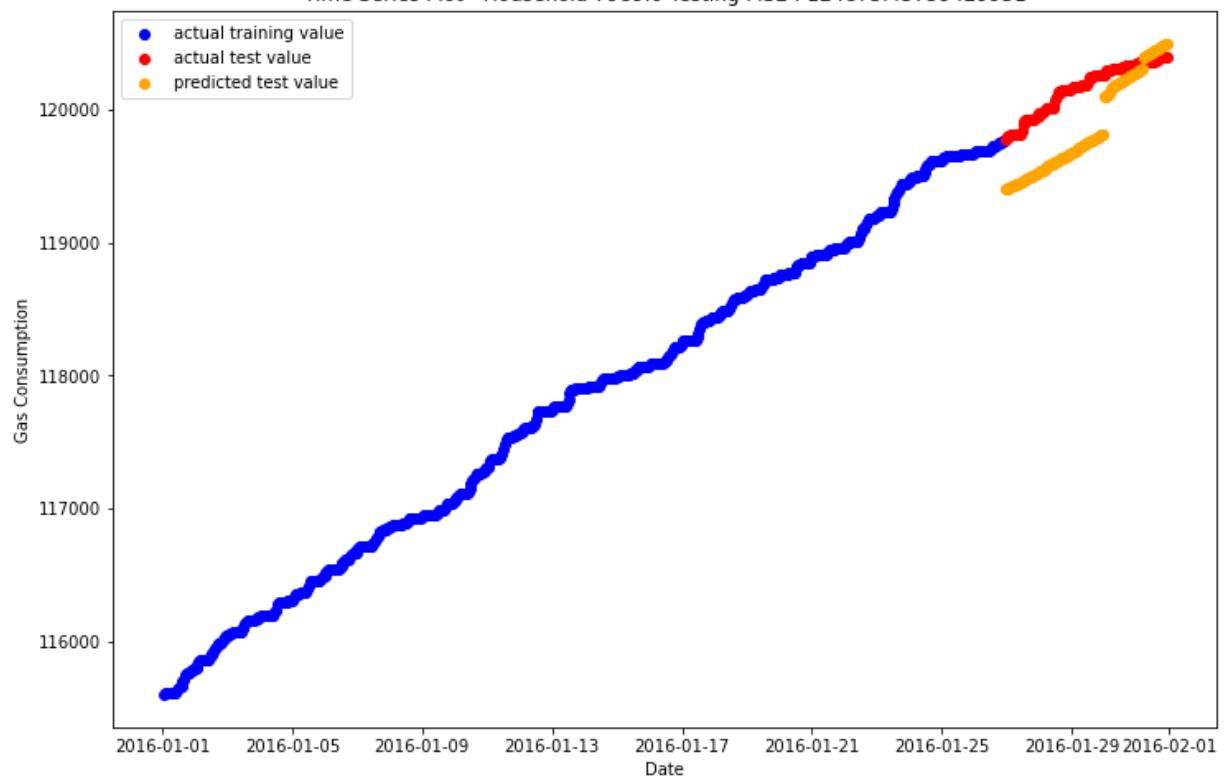


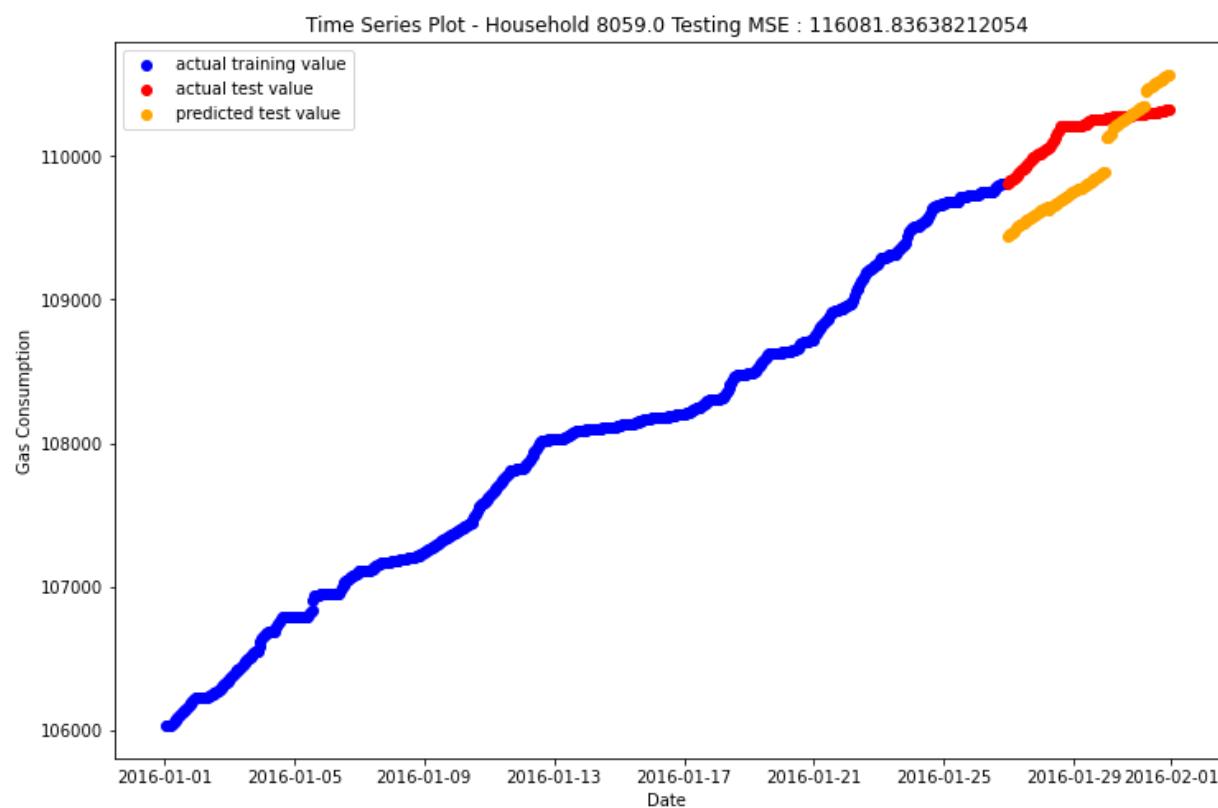
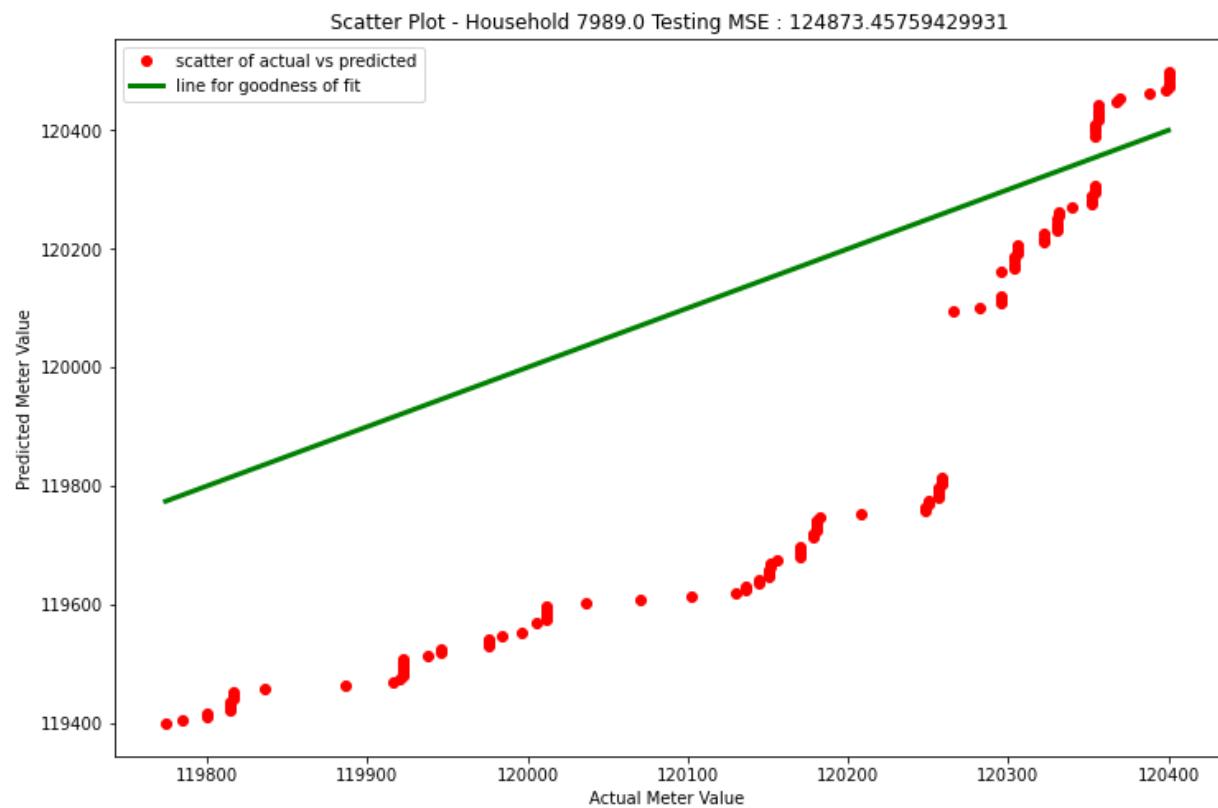


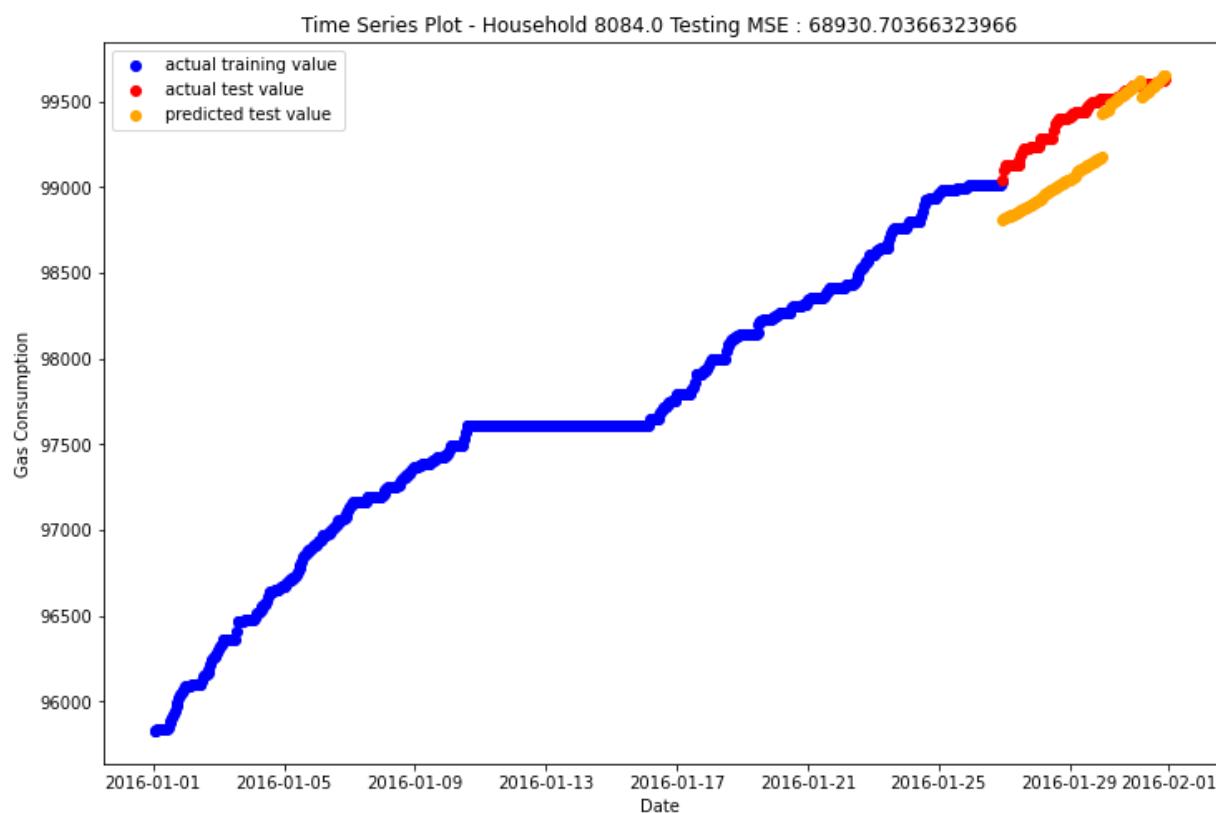
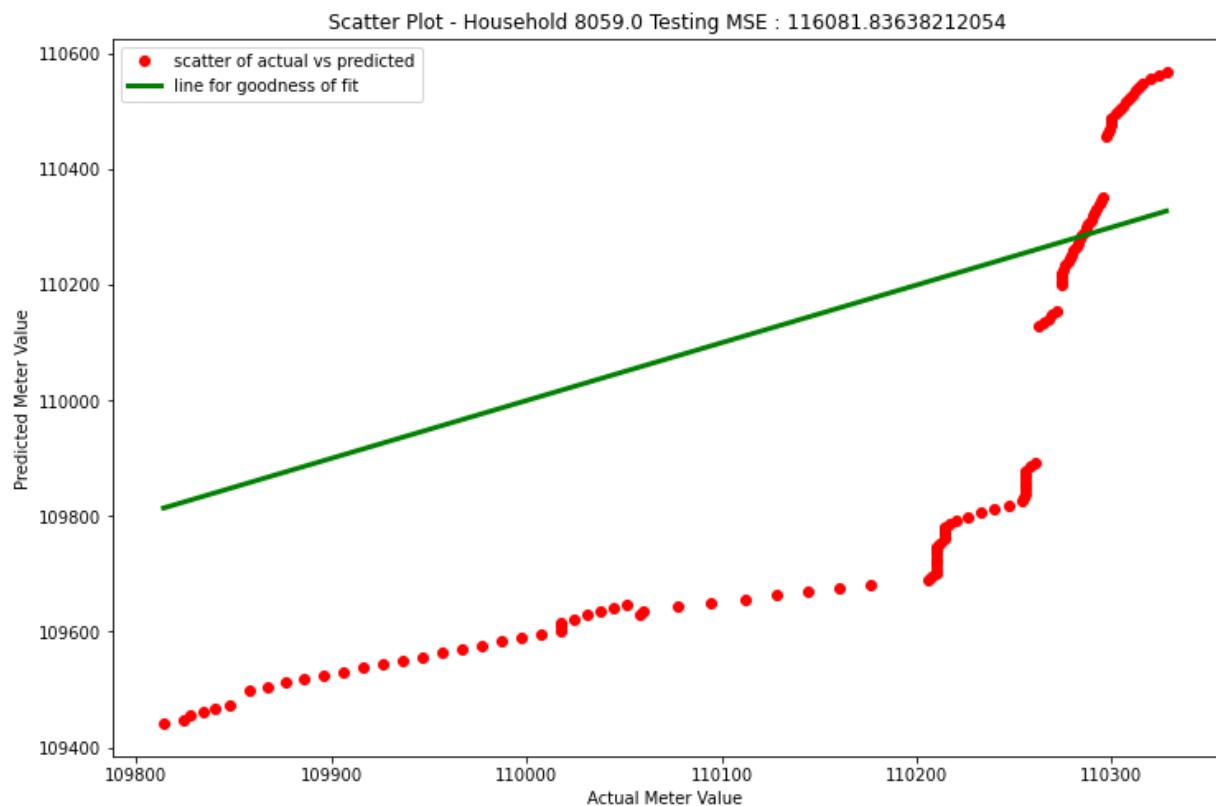
Scatter Plot - Household 7965.0 Testing MSE : 381444.556554517

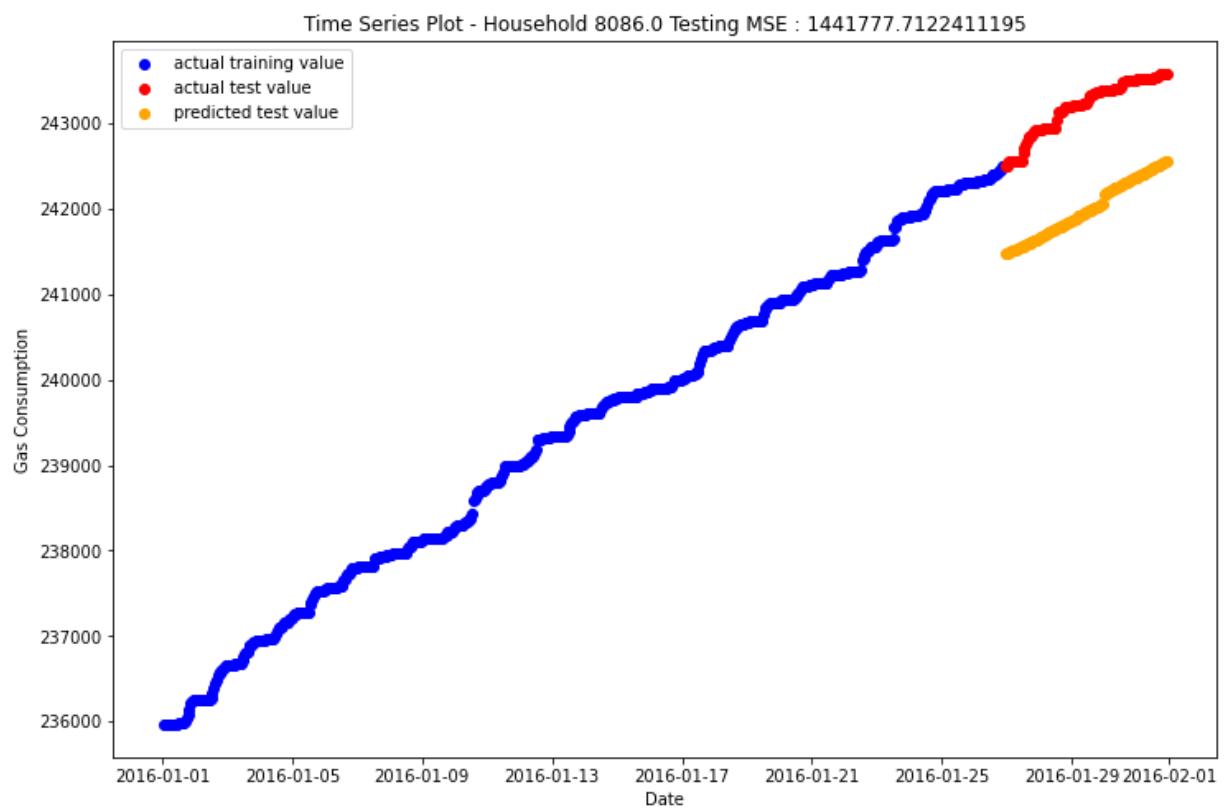
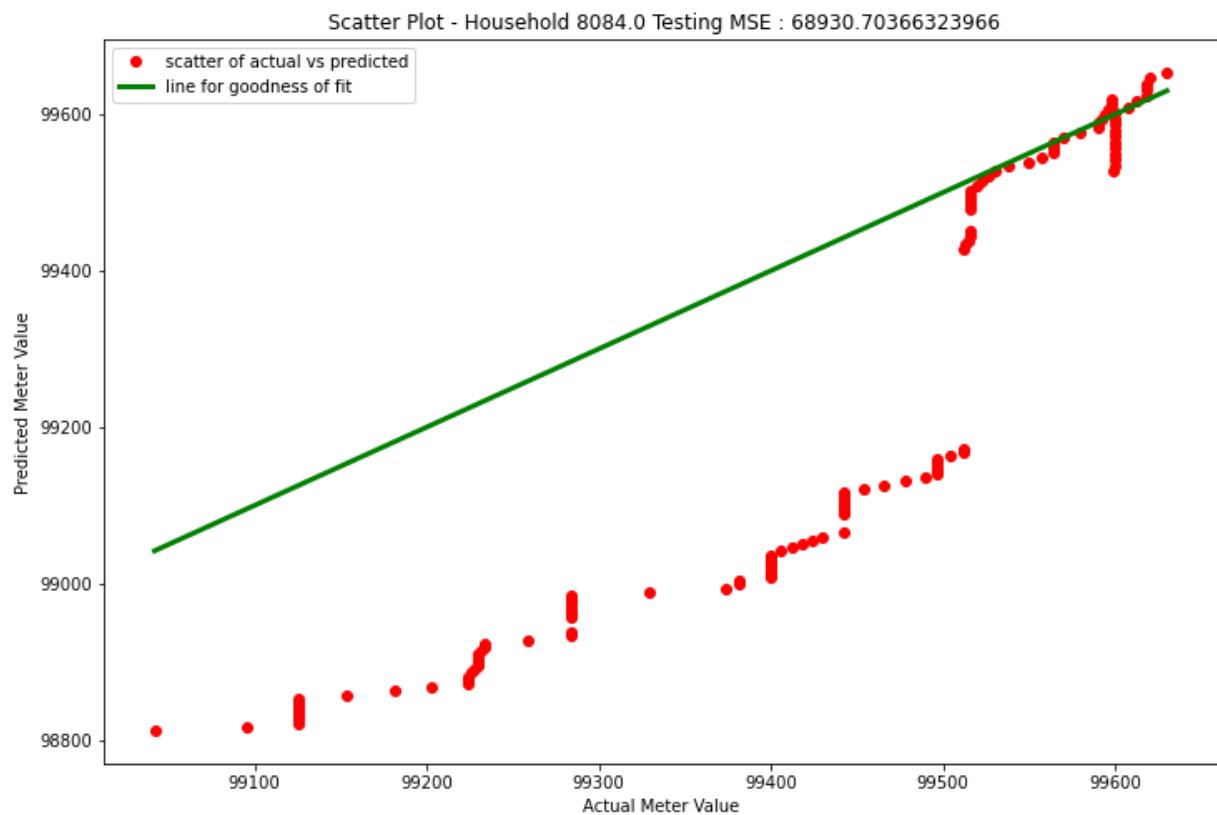


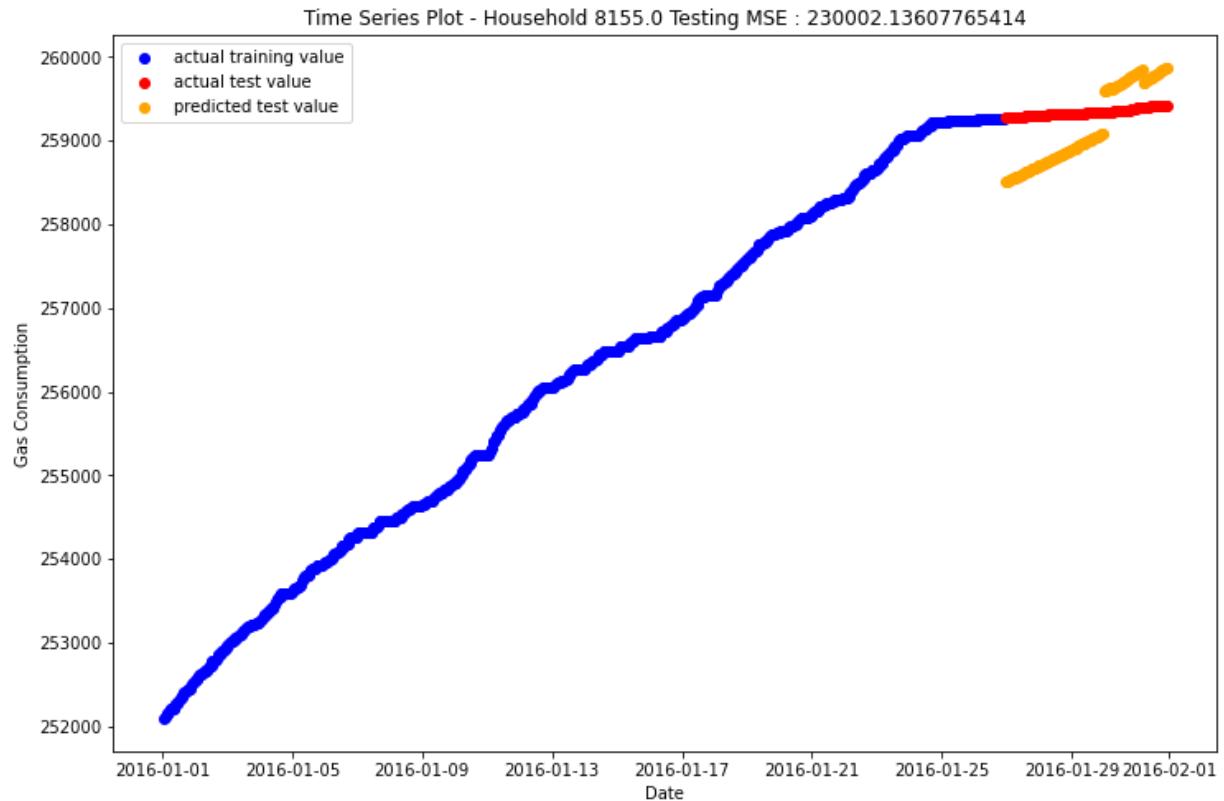
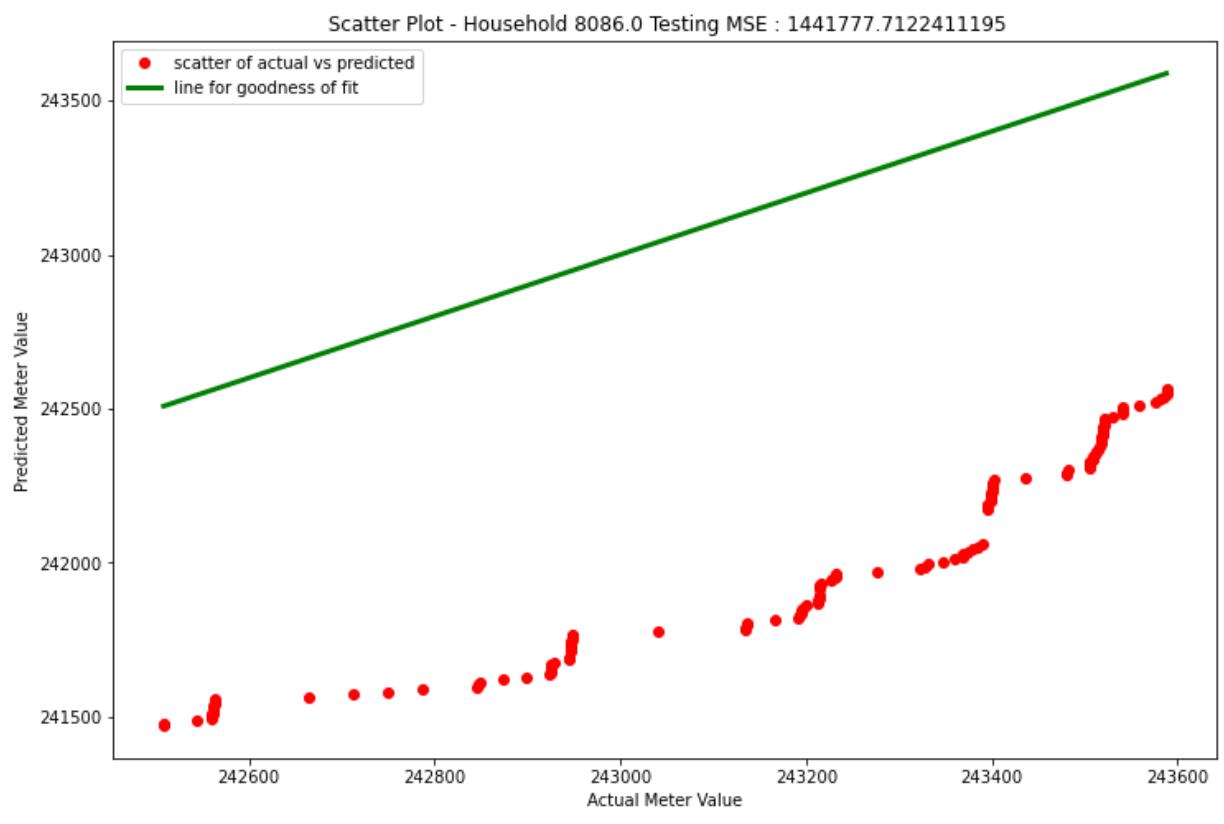
Time Series Plot - Household 7989.0 Testing MSE : 124873.45759429931



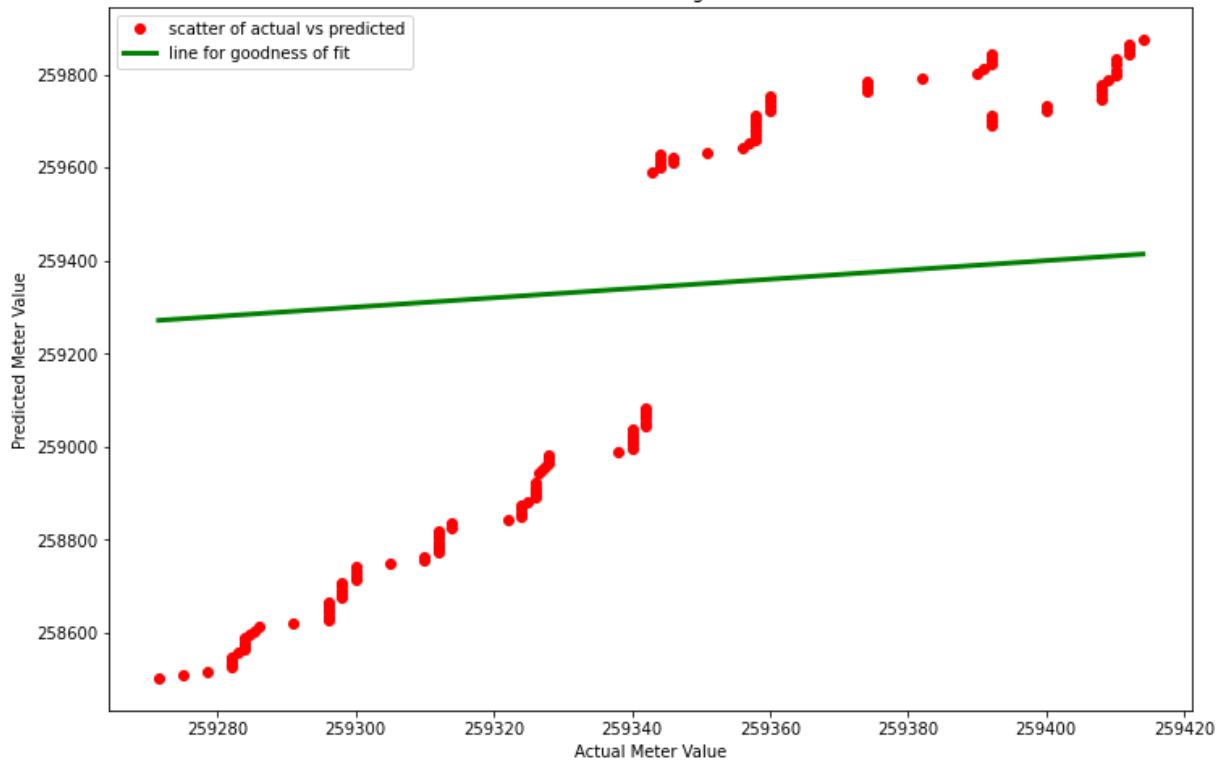




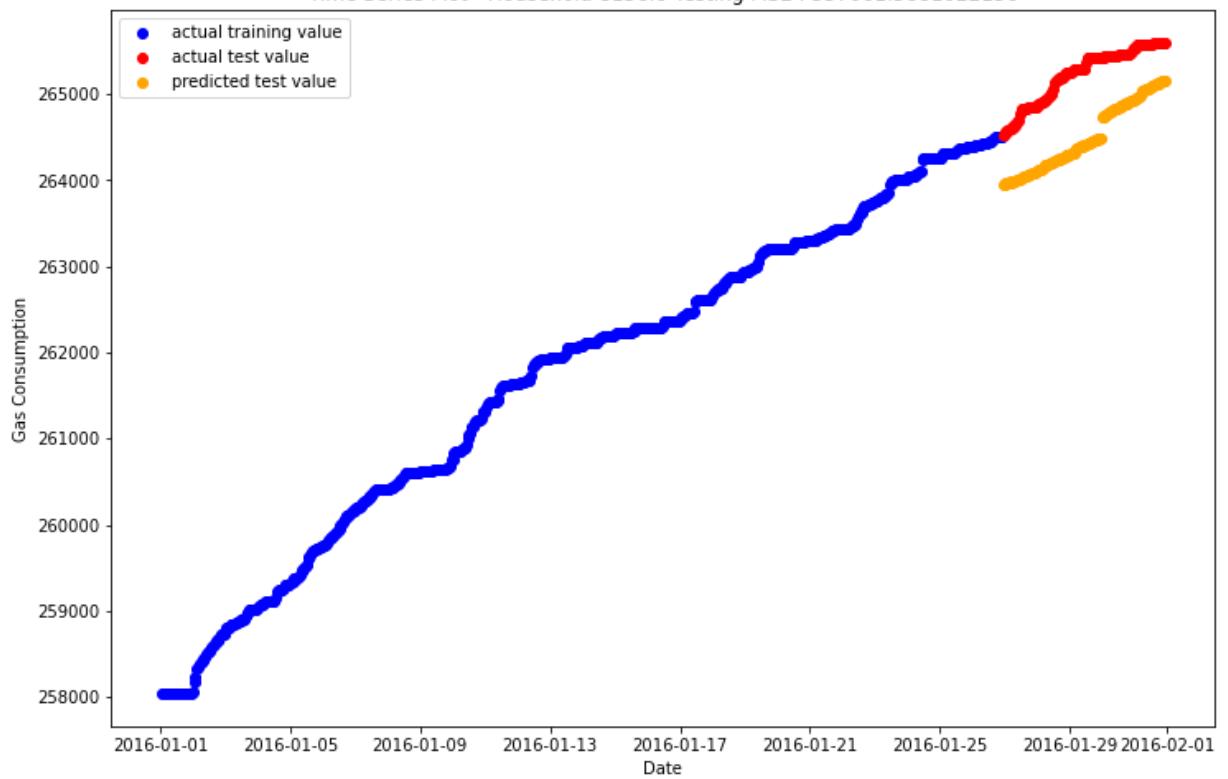


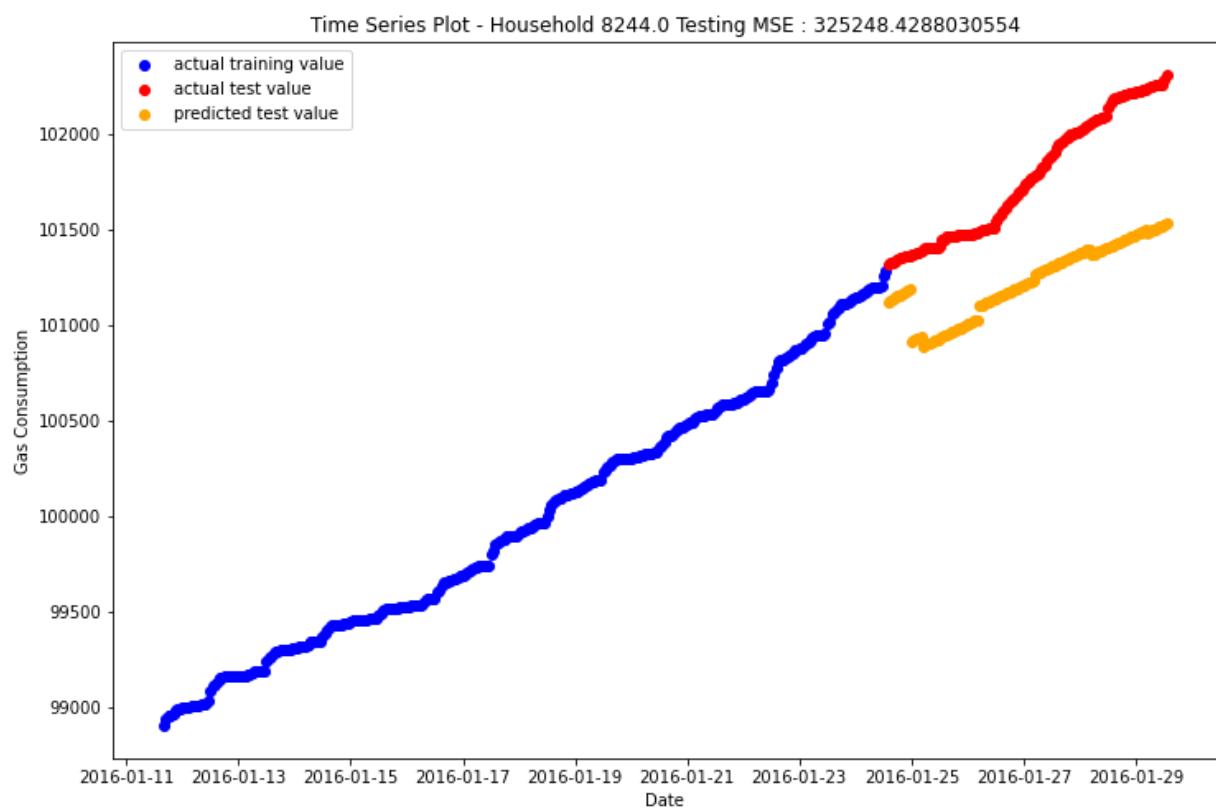
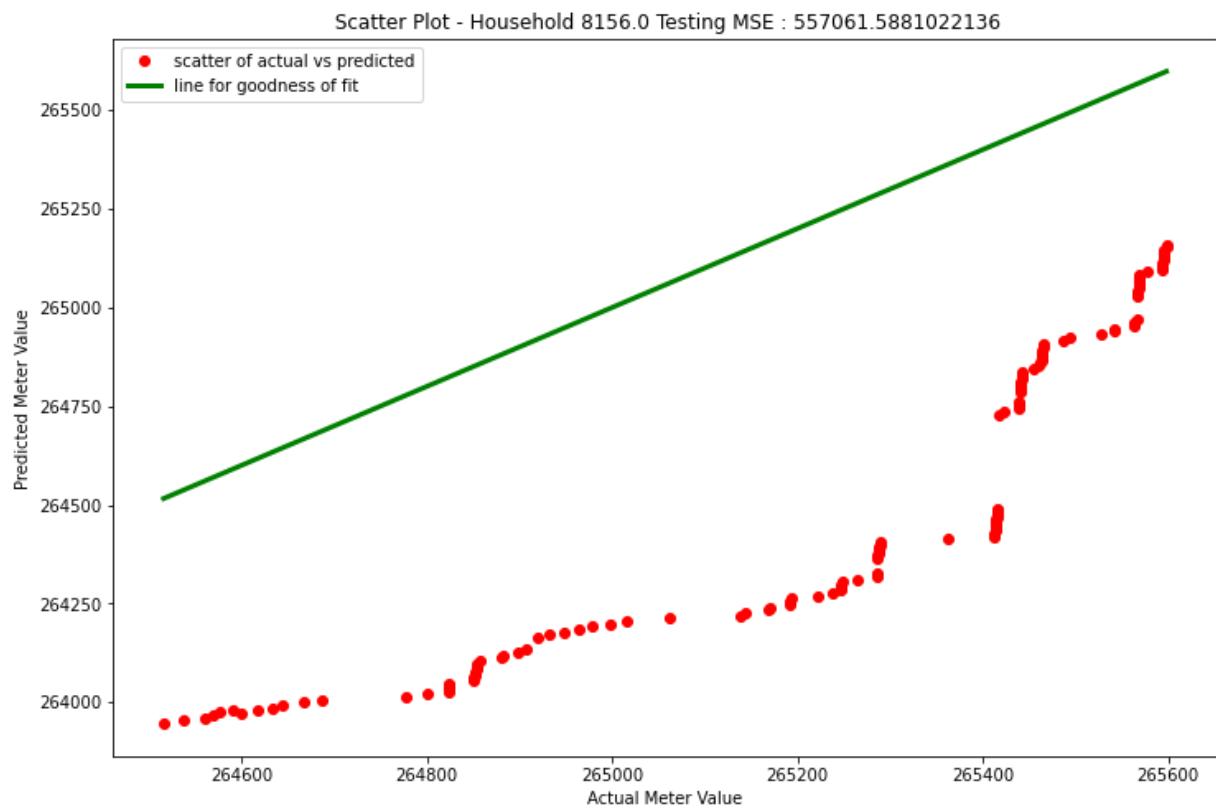


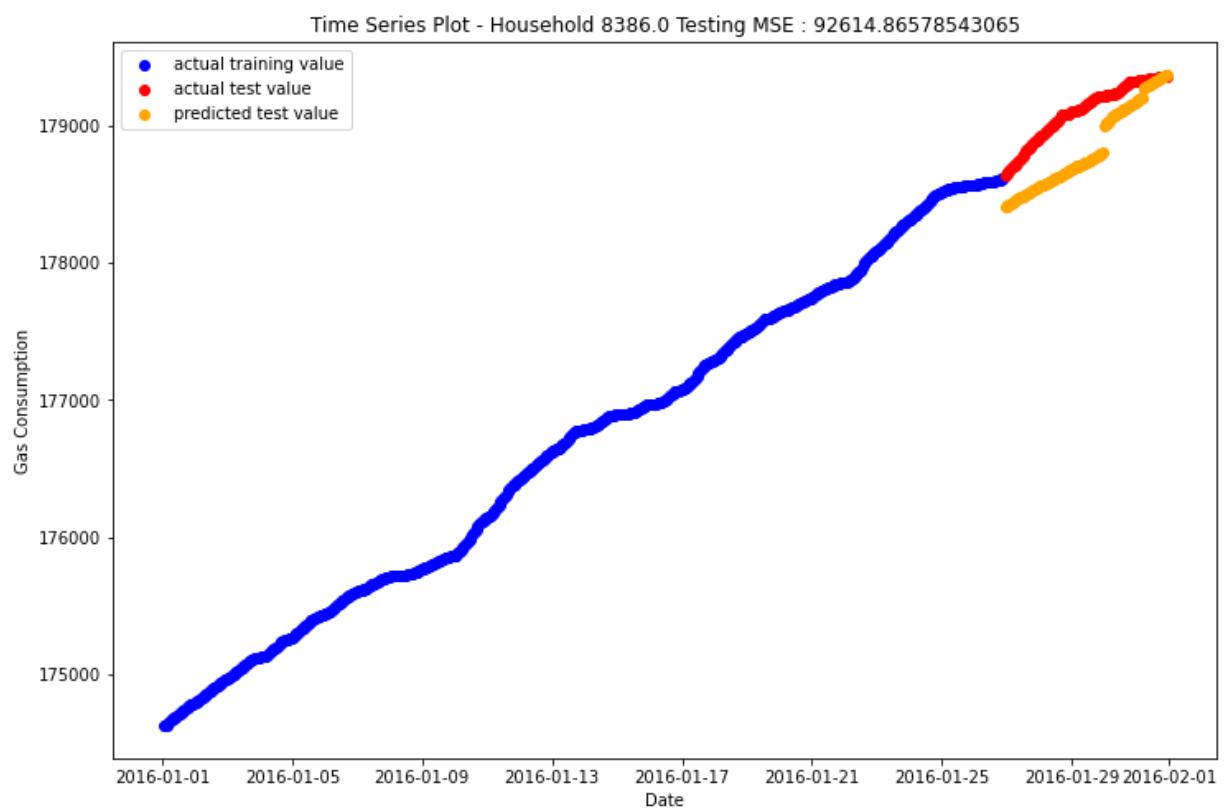
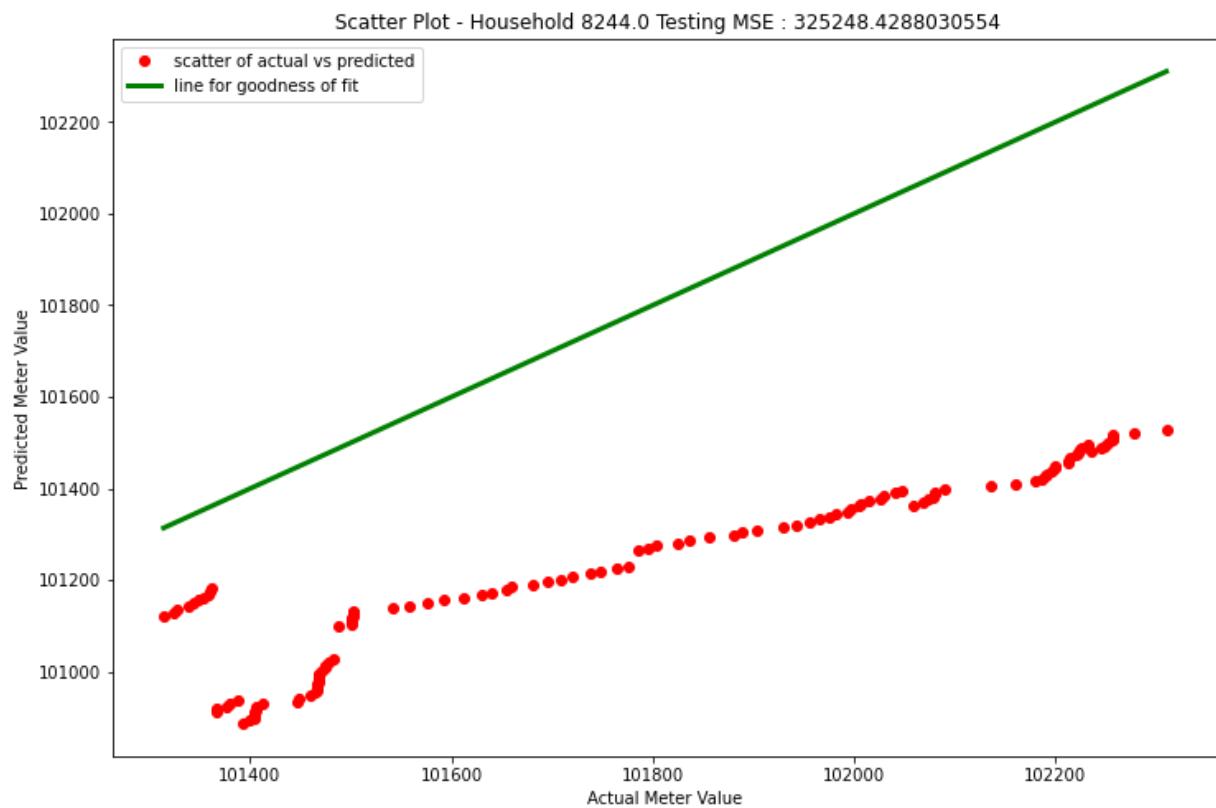
Scatter Plot - Household 8155.0 Testing MSE : 230002.13607765414

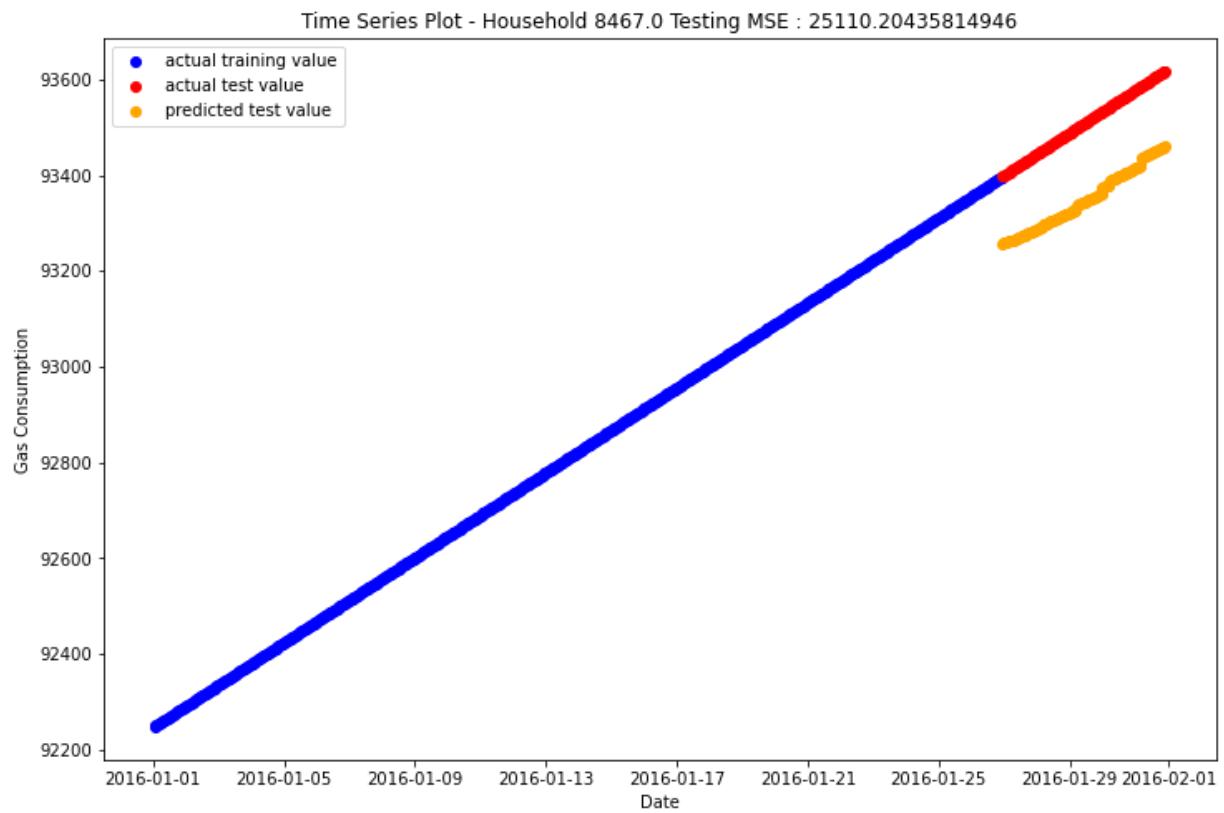
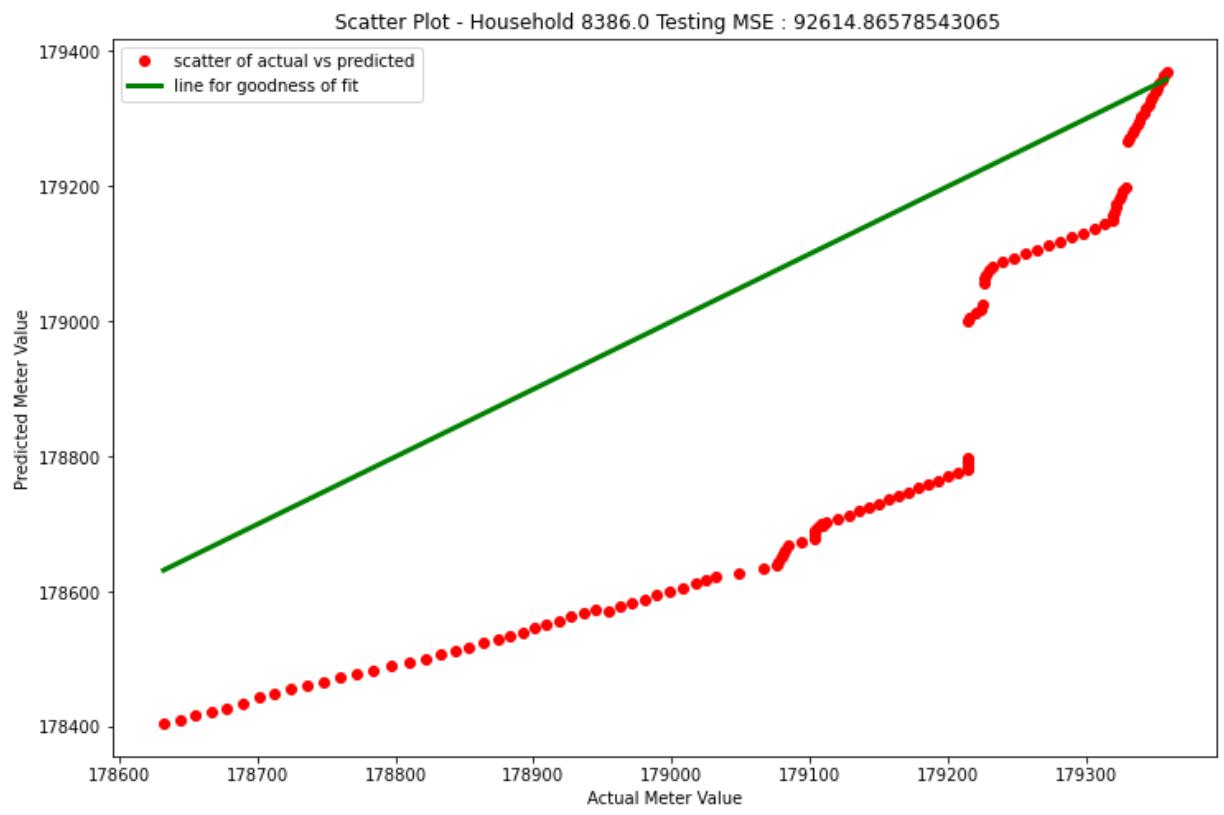


Time Series Plot - Household 8156.0 Testing MSE : 557061.5881022136

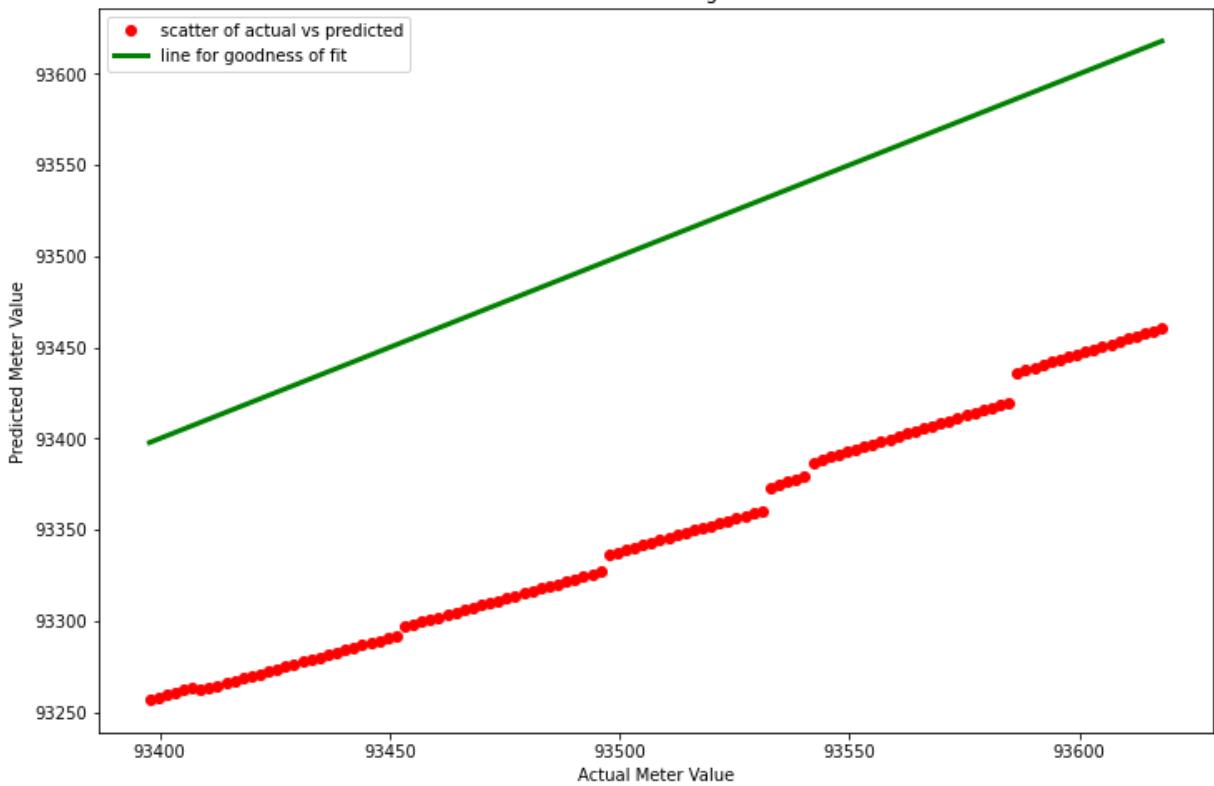




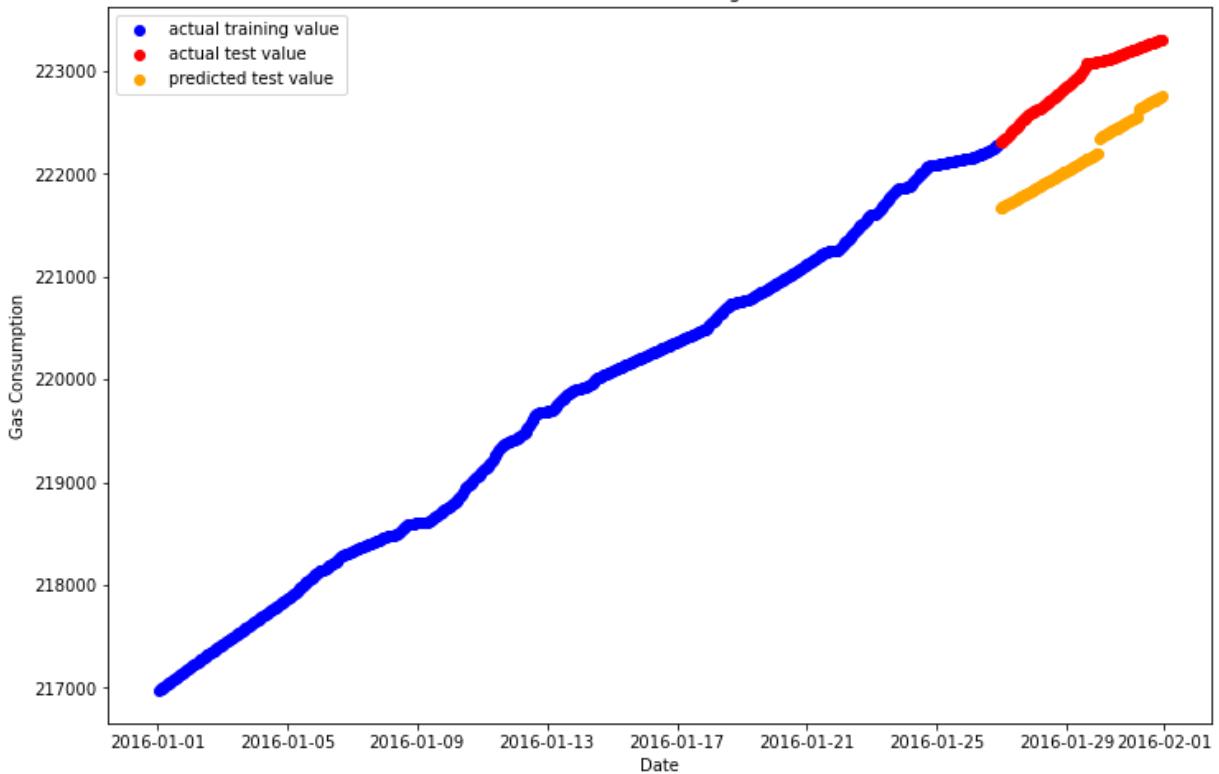


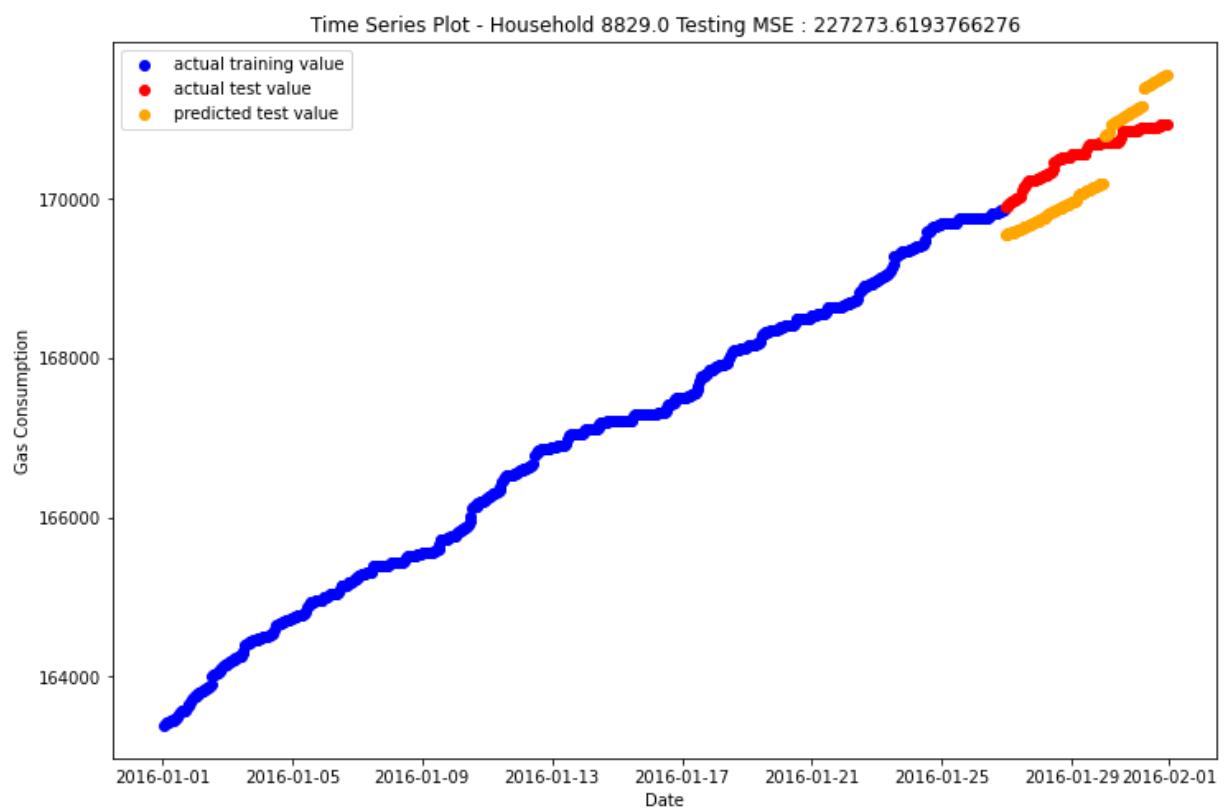
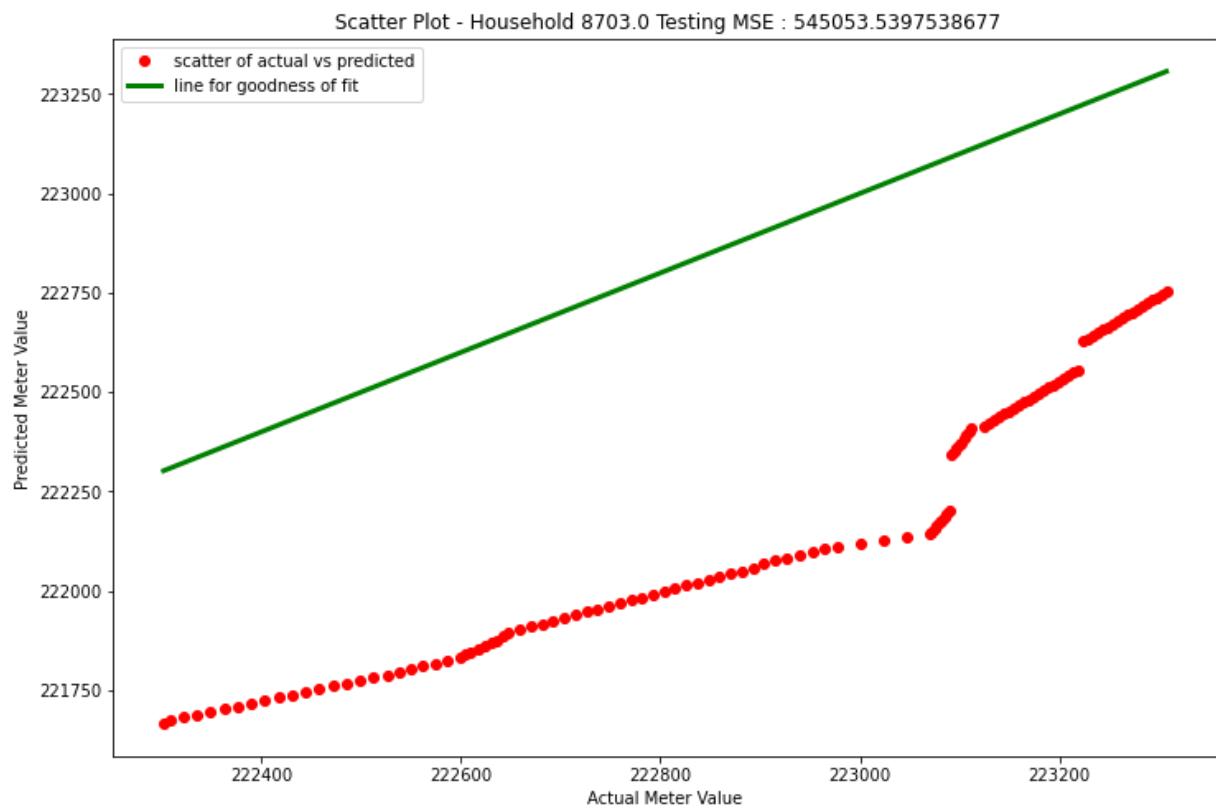


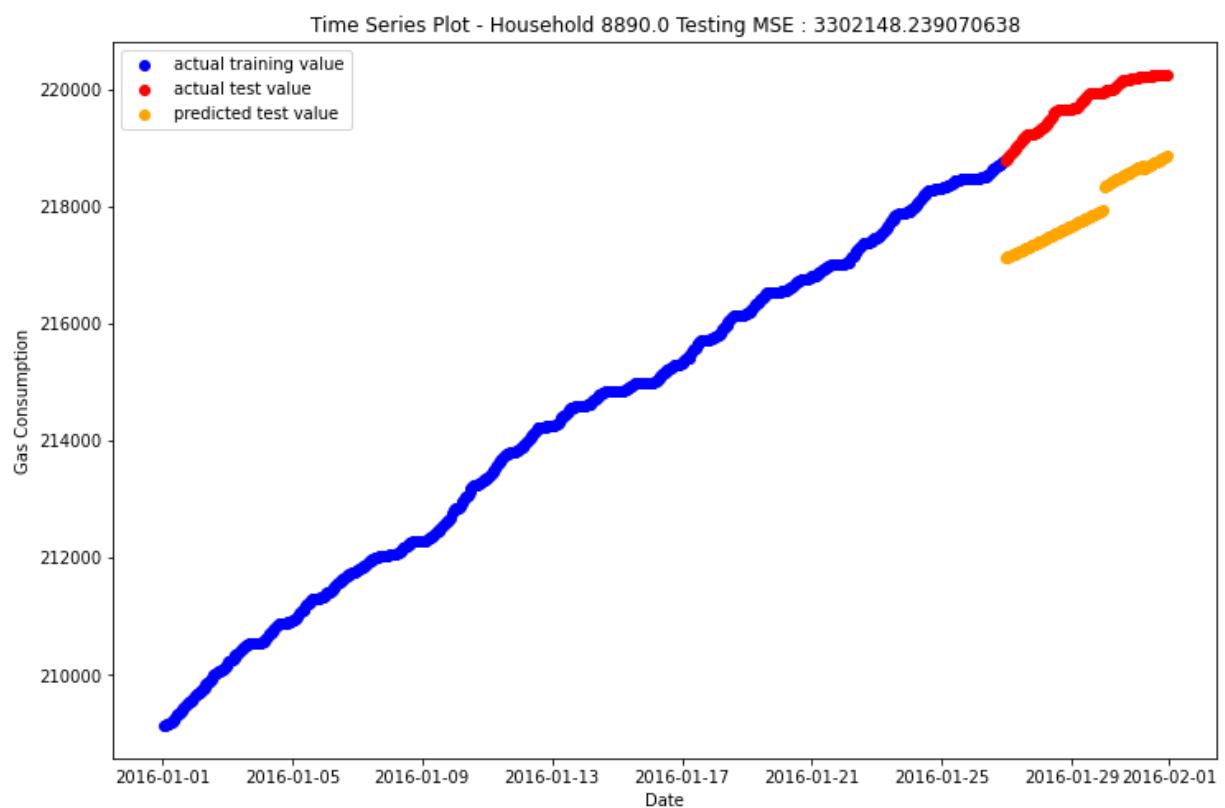
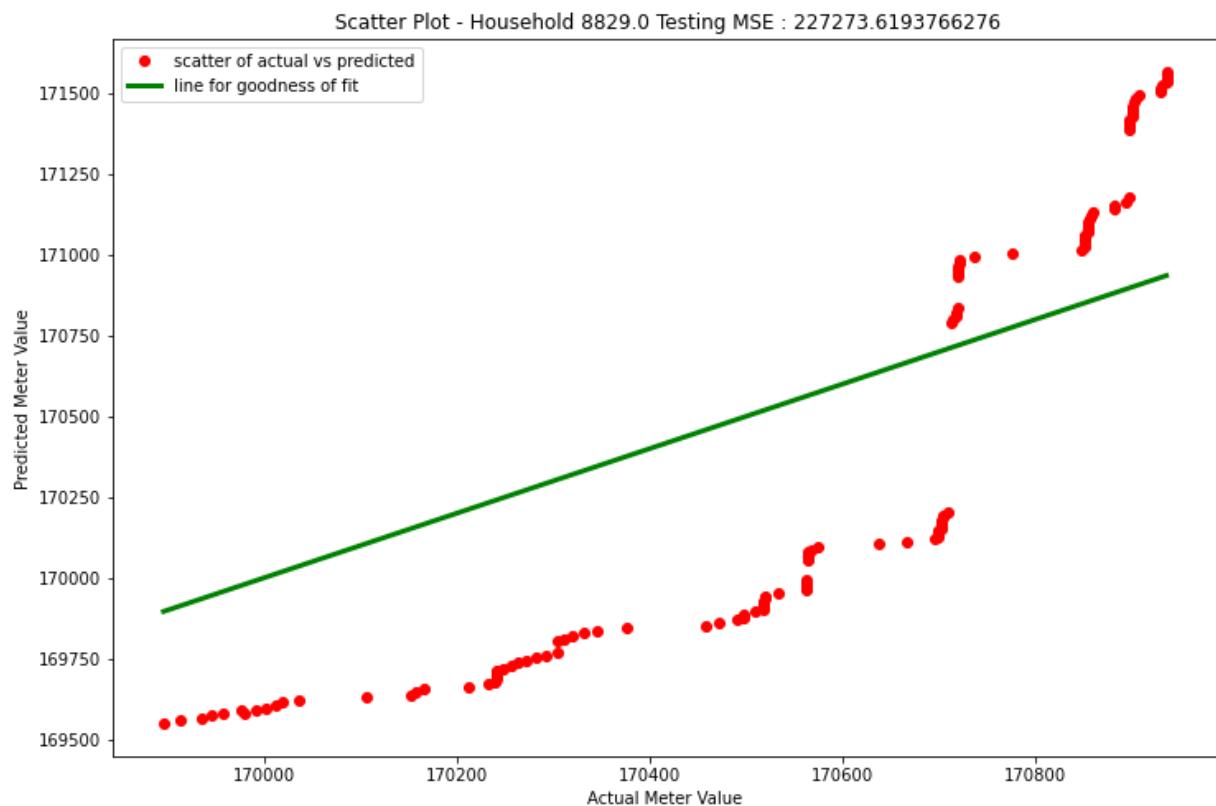
Scatter Plot - Household 8467.0 Testing MSE : 25110.20435814946



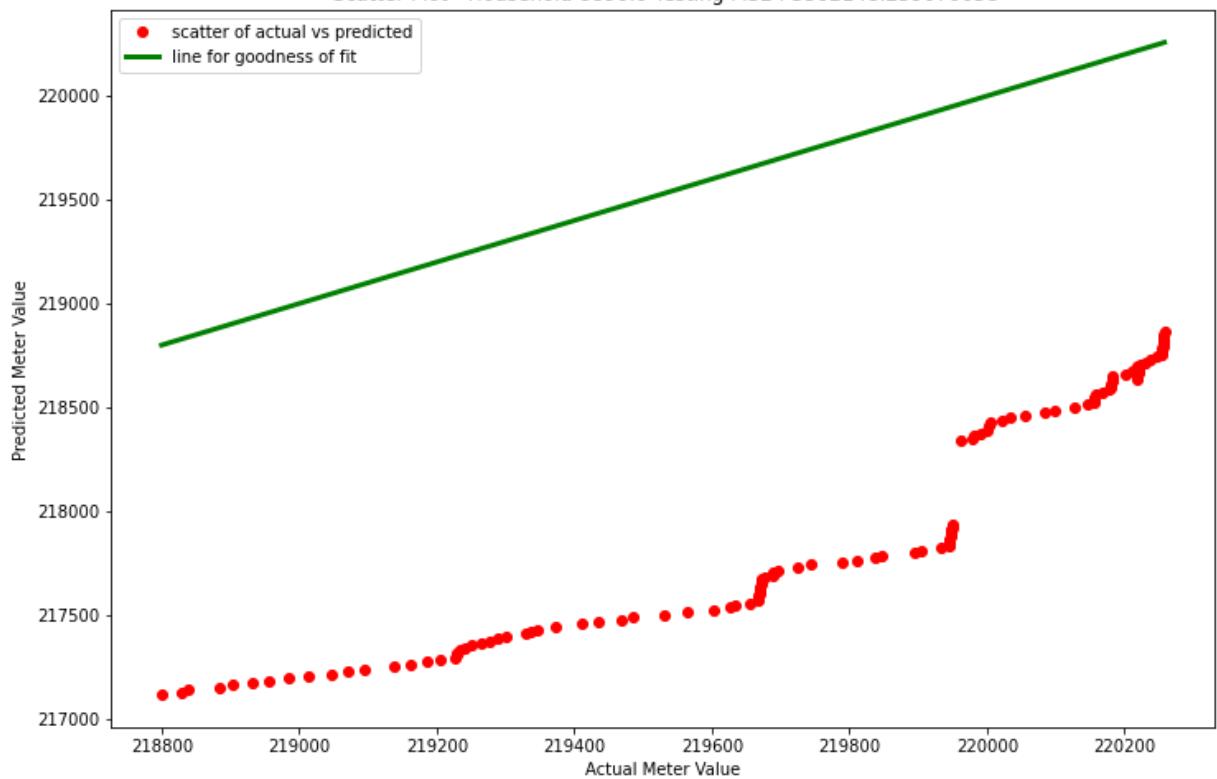
Time Series Plot - Household 8703.0 Testing MSE : 545053.5397538677



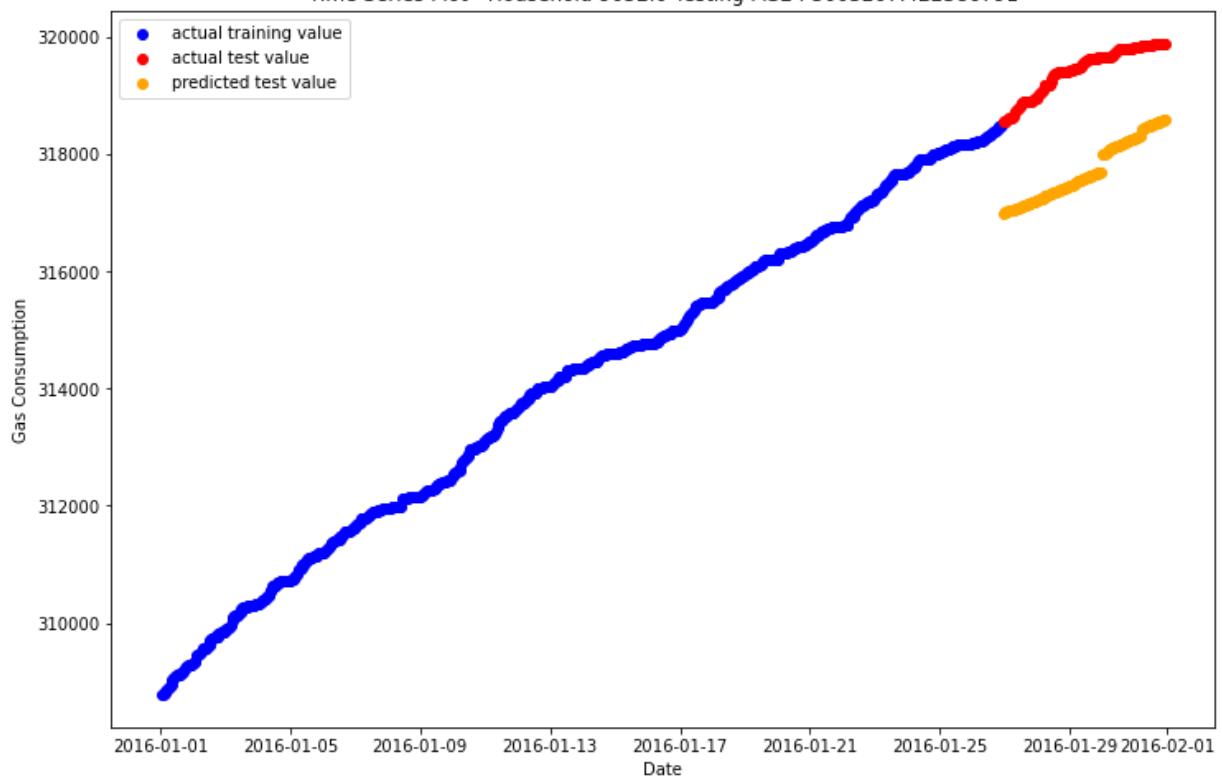


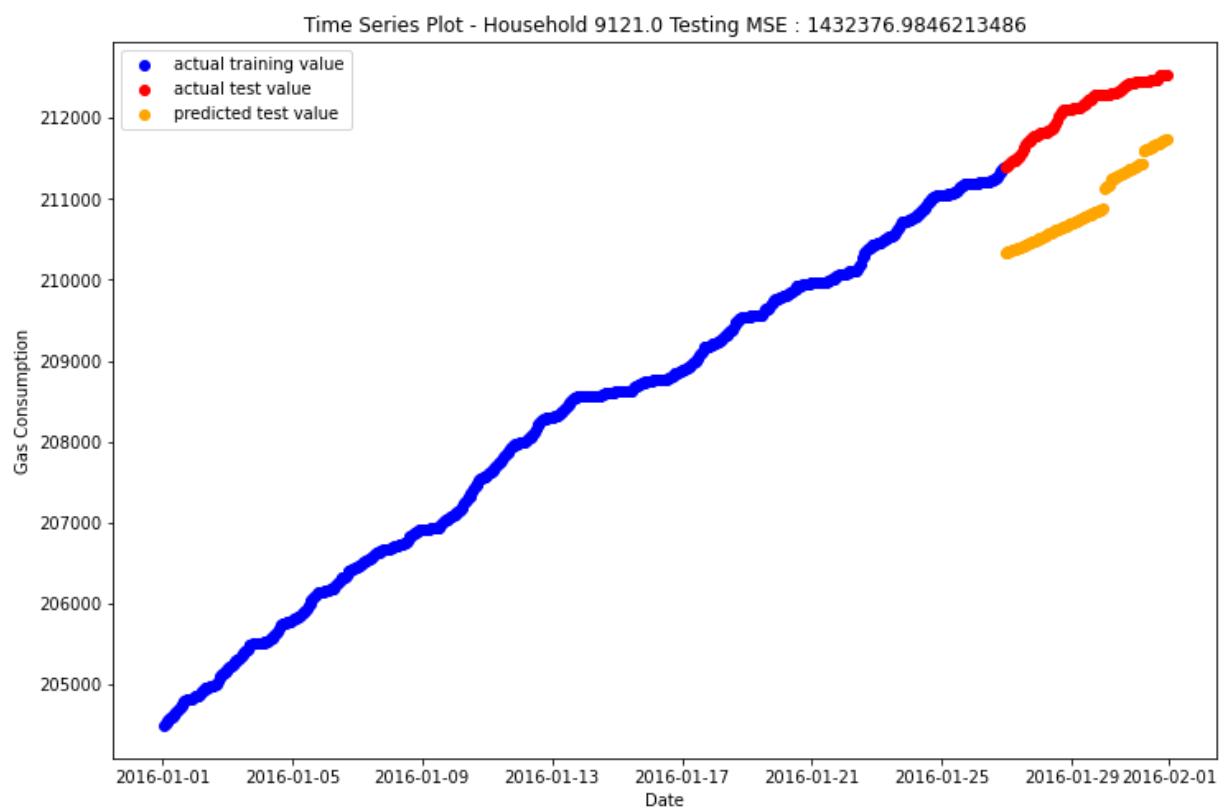
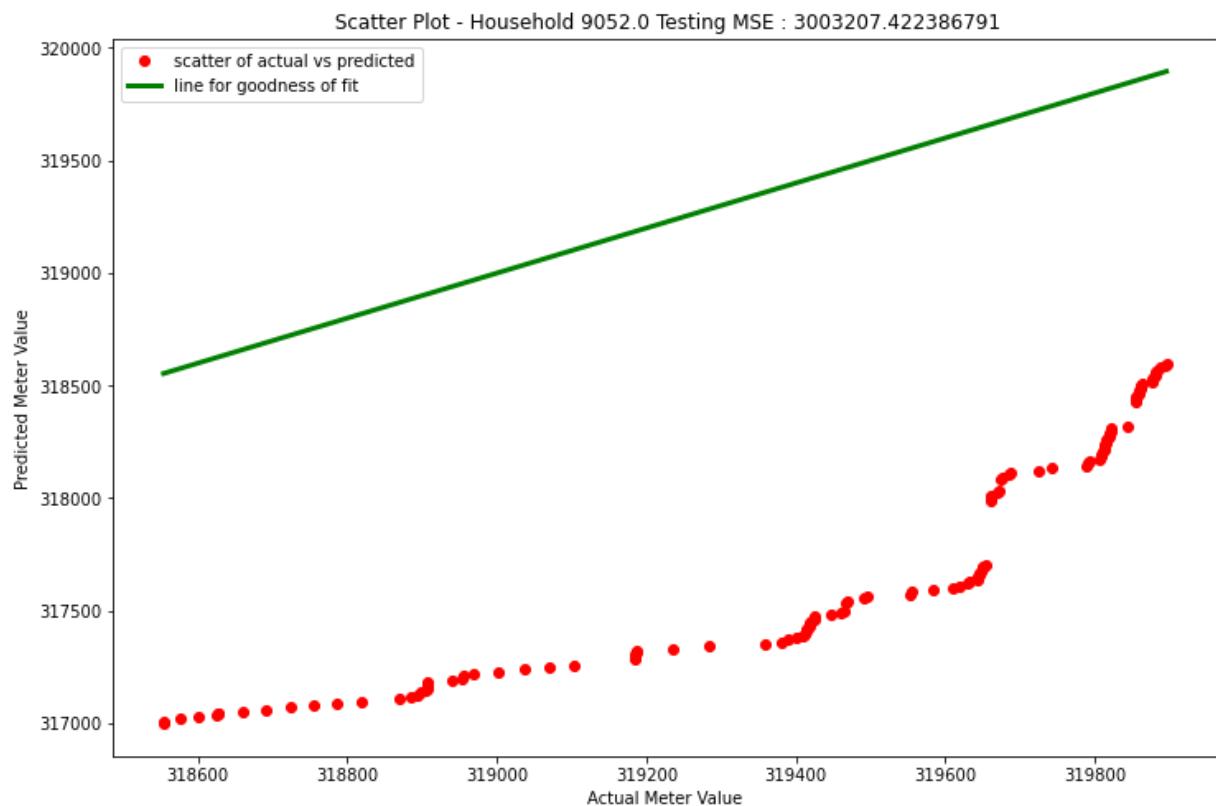


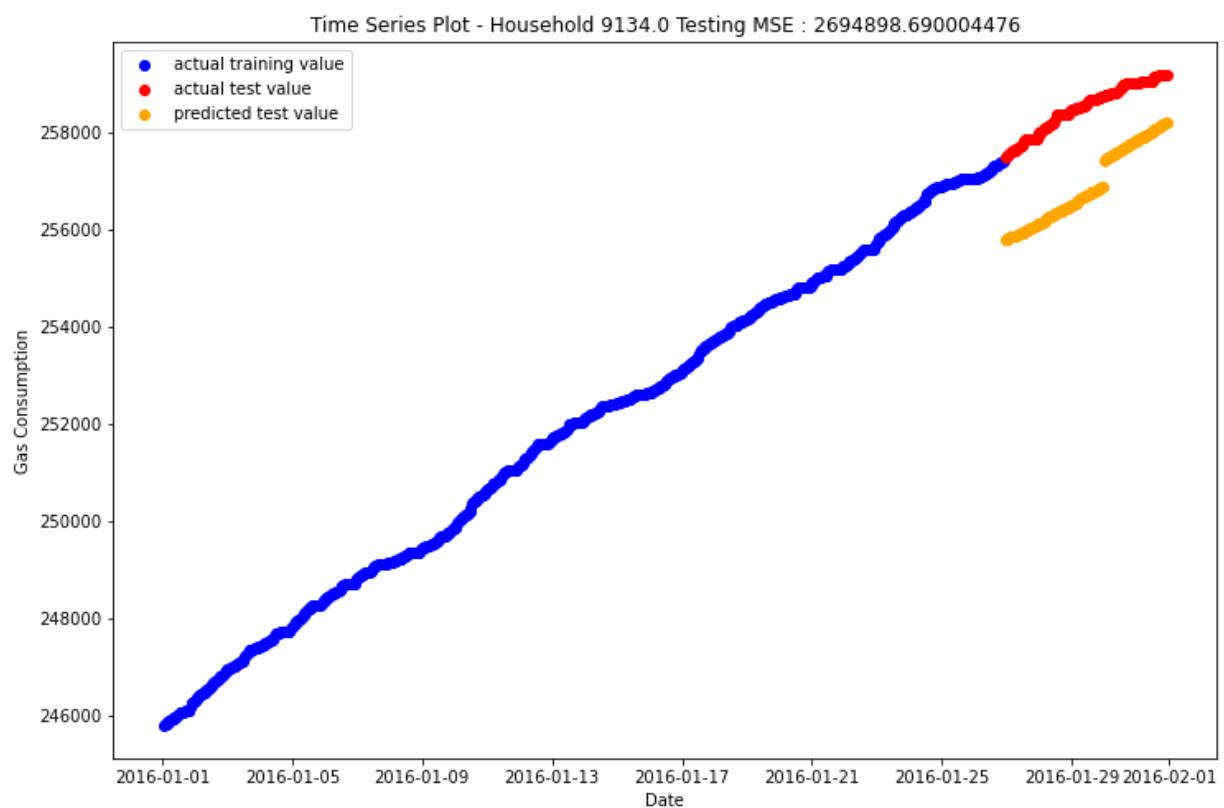
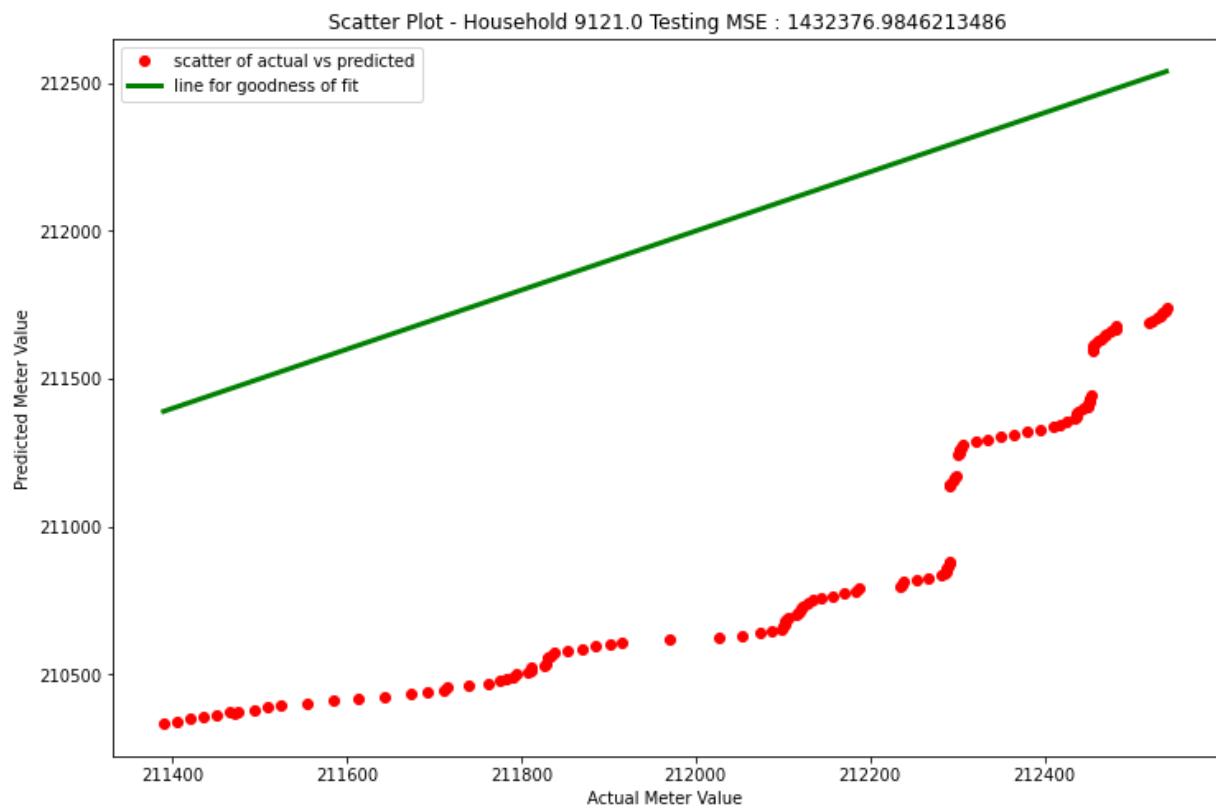
Scatter Plot - Household 8890.0 Testing MSE : 3302148.239070638

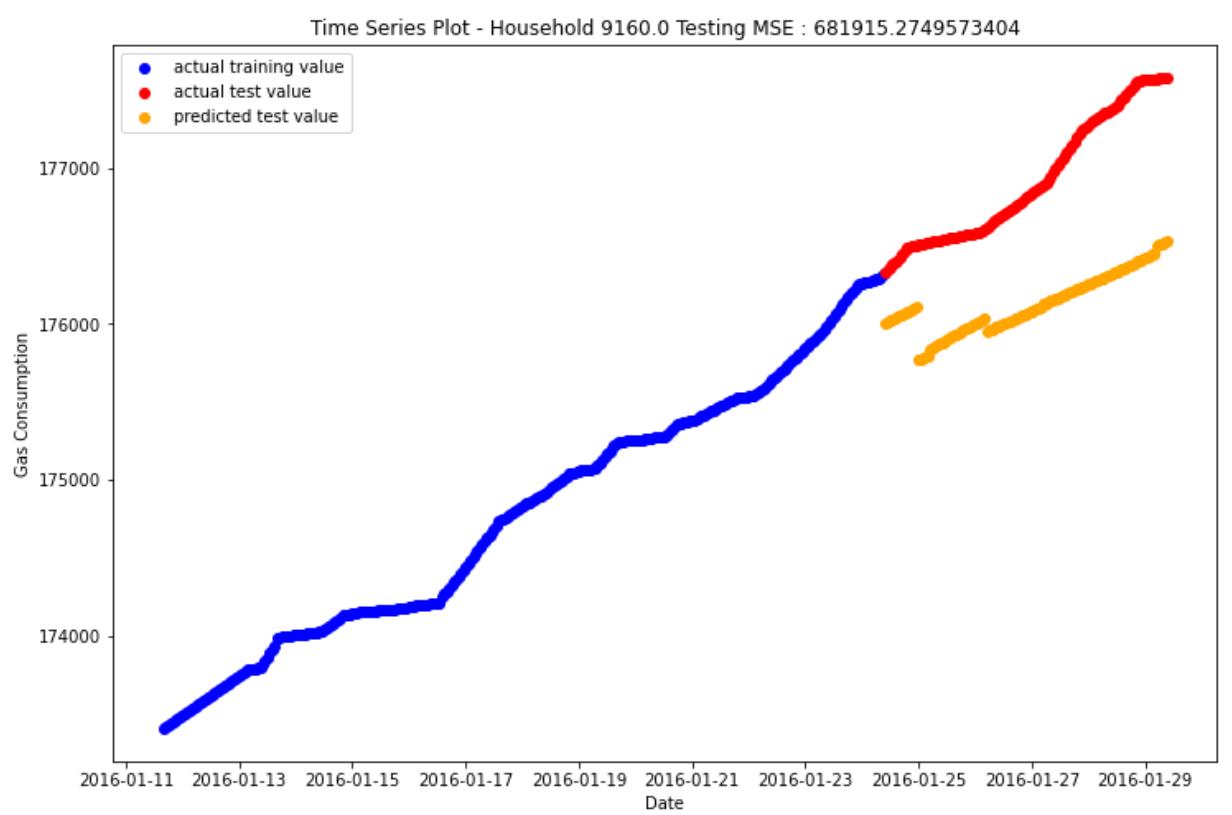
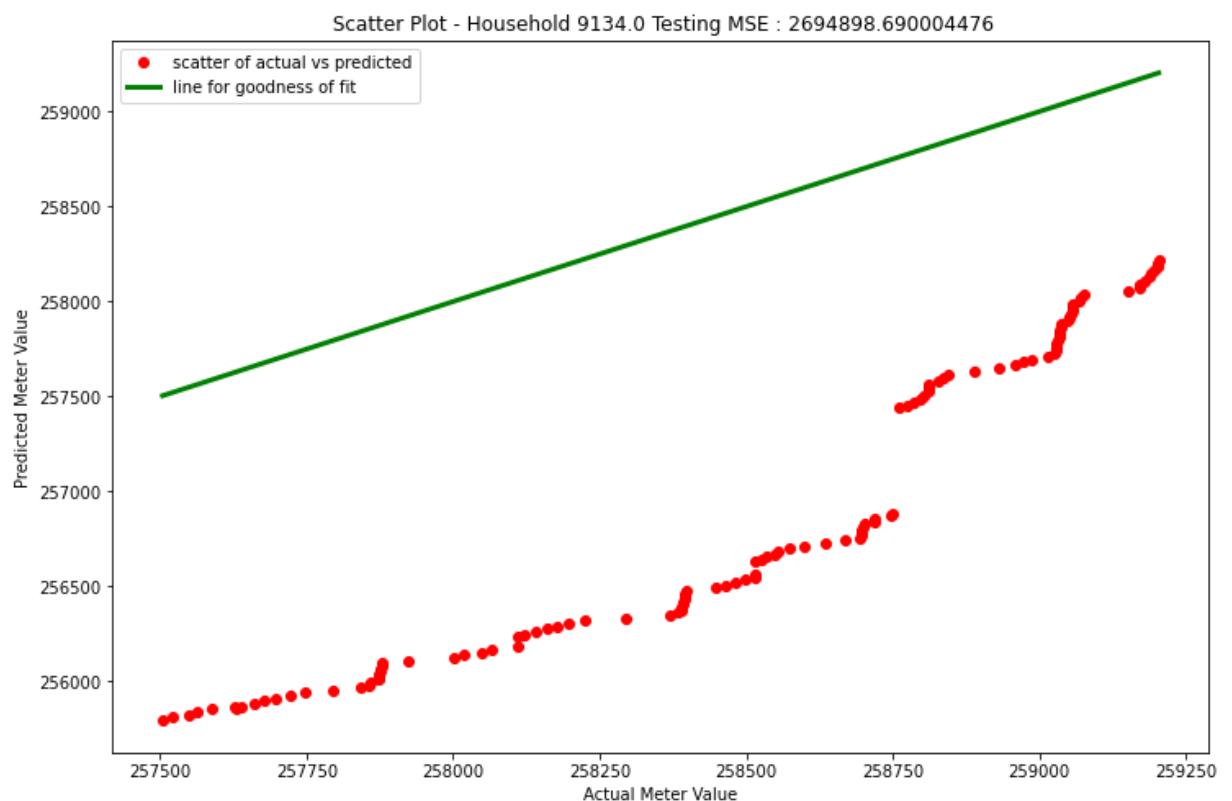


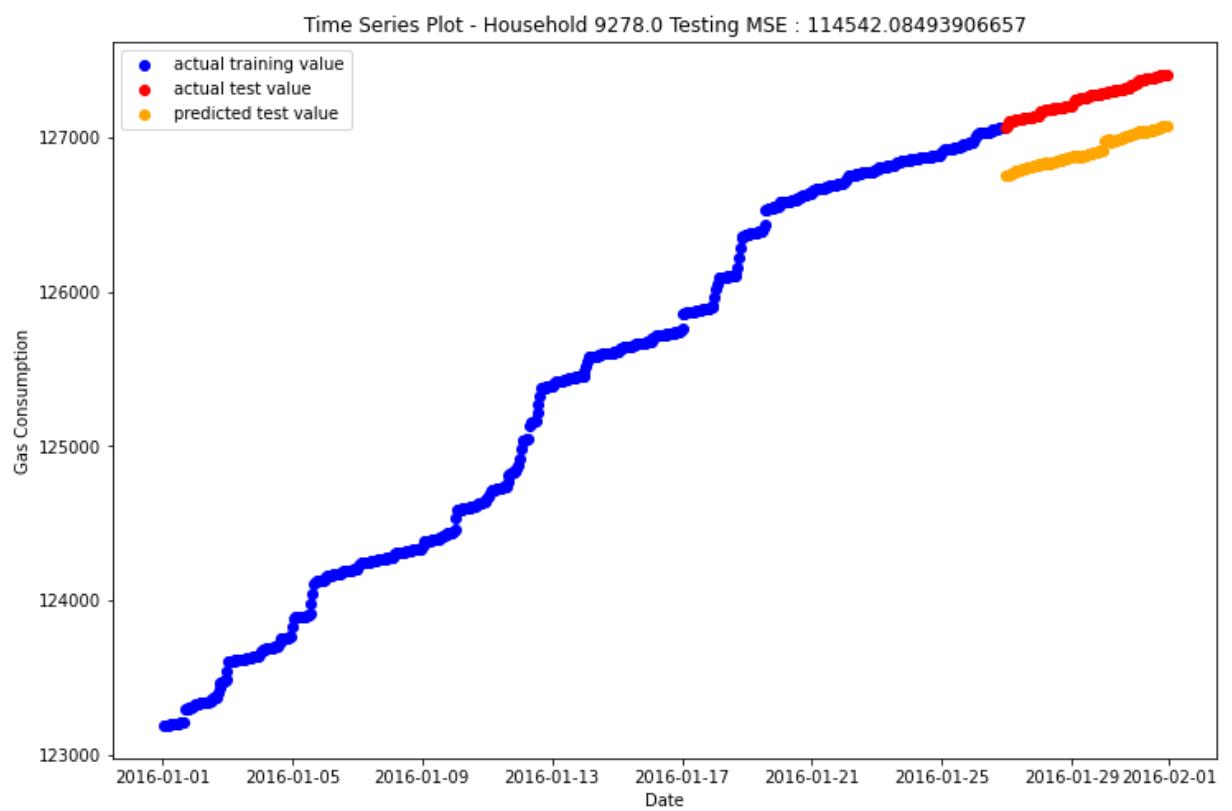
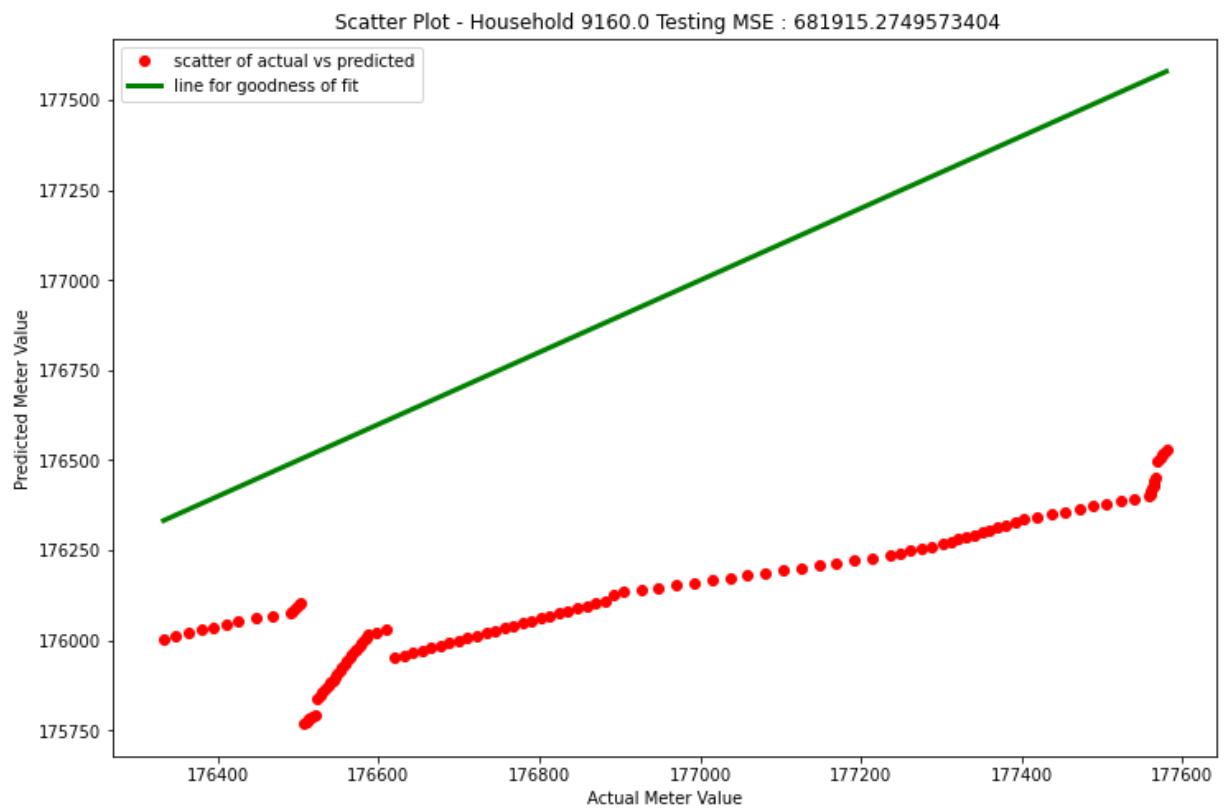
Time Series Plot - Household 9052.0 Testing MSE : 3003207.422386791

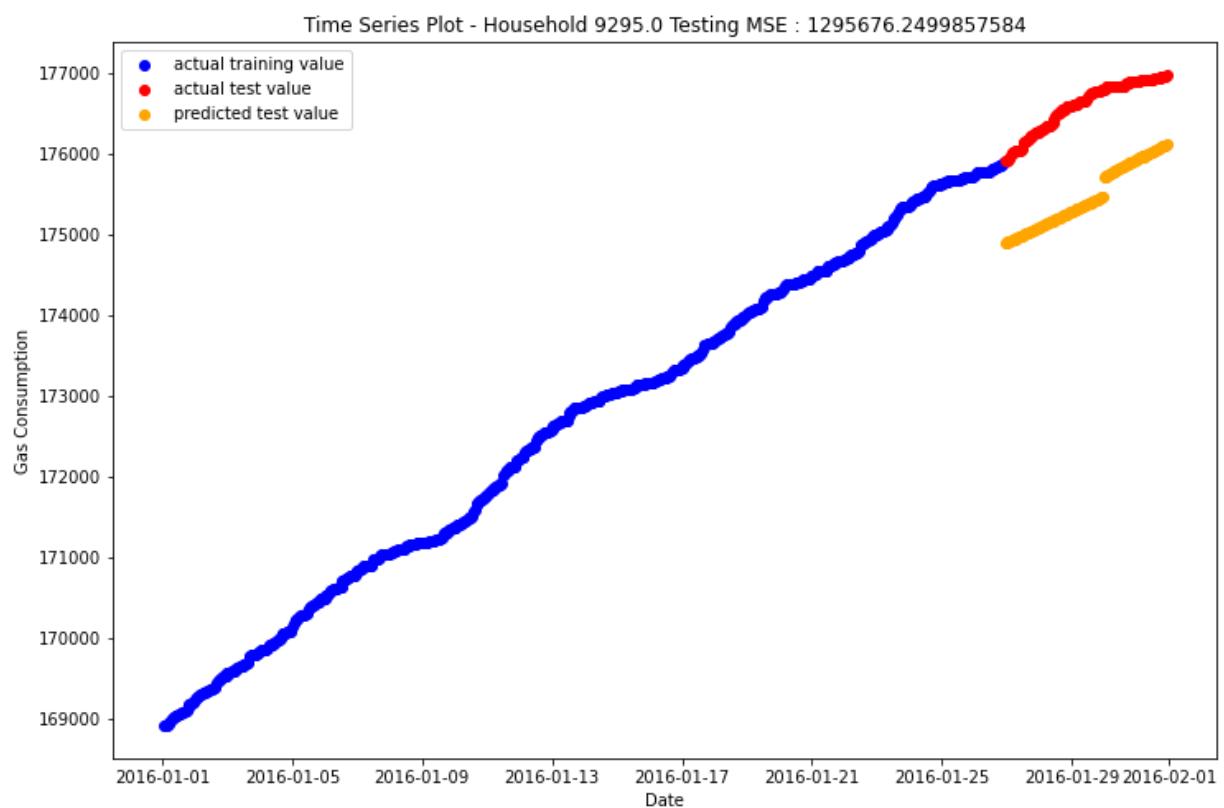
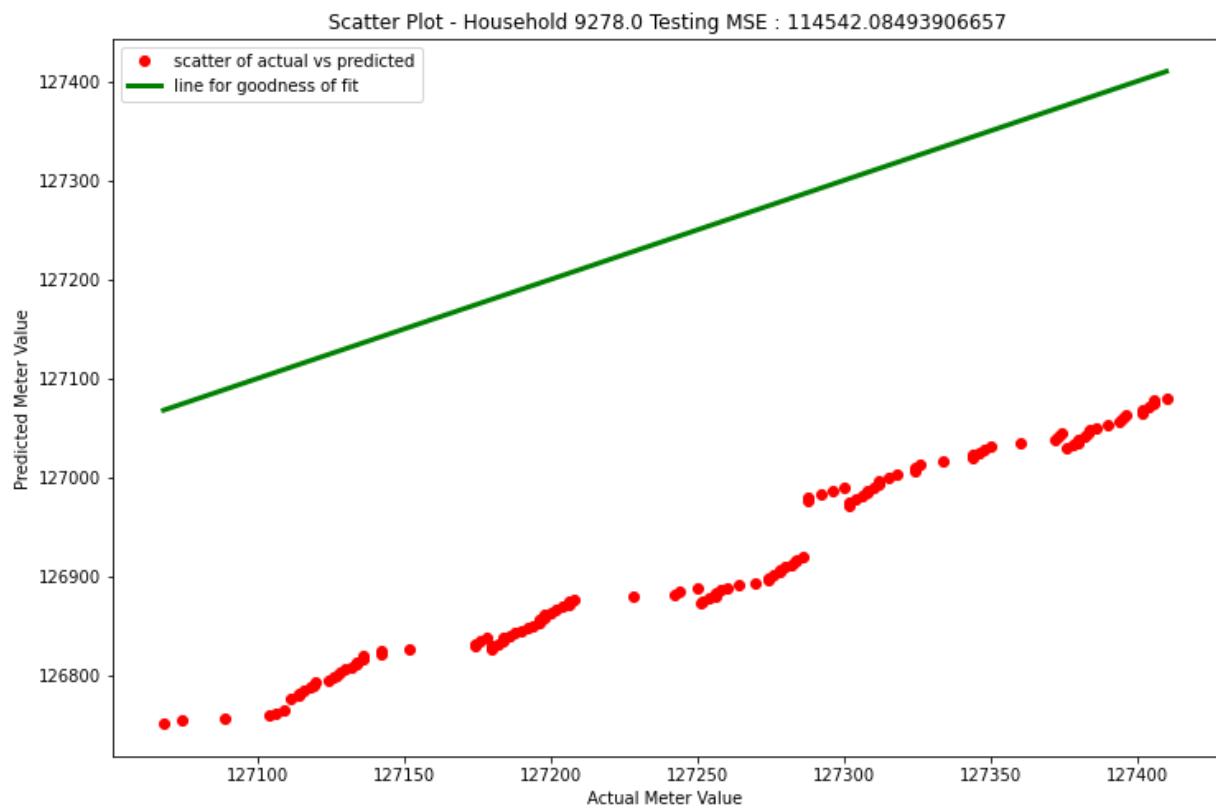


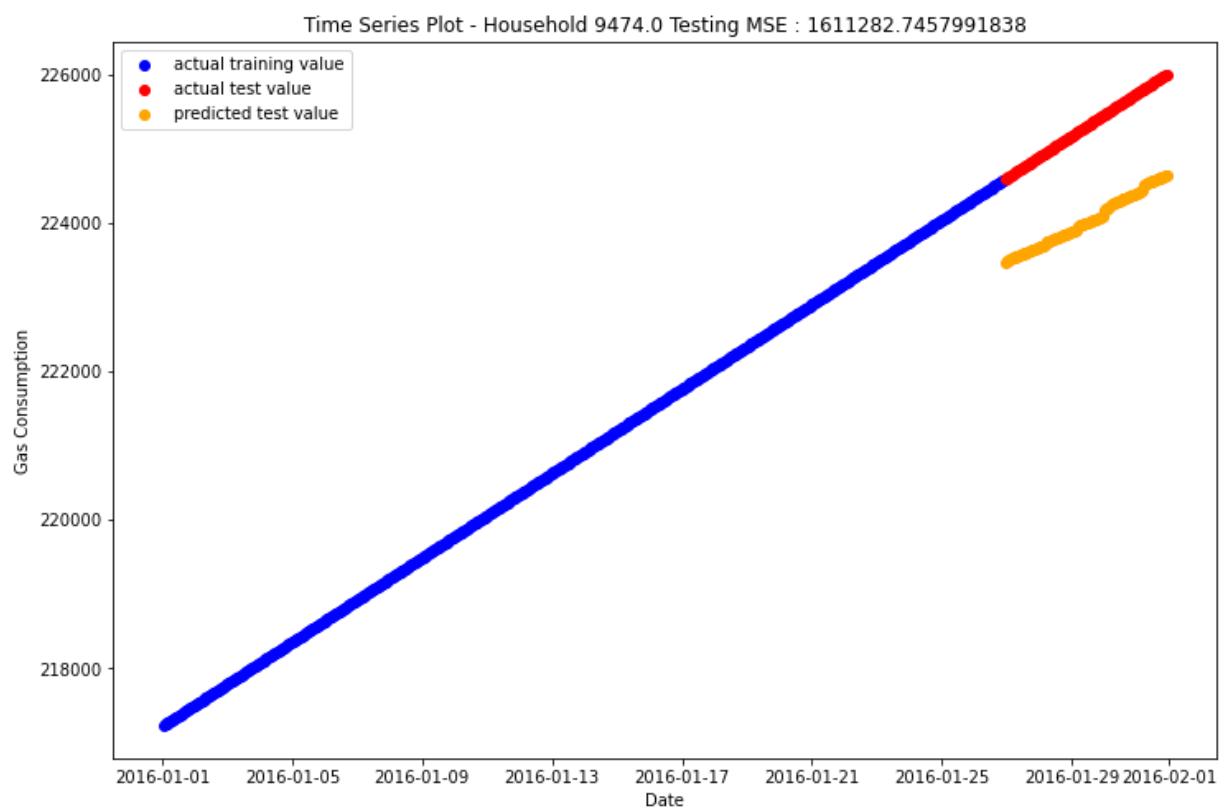
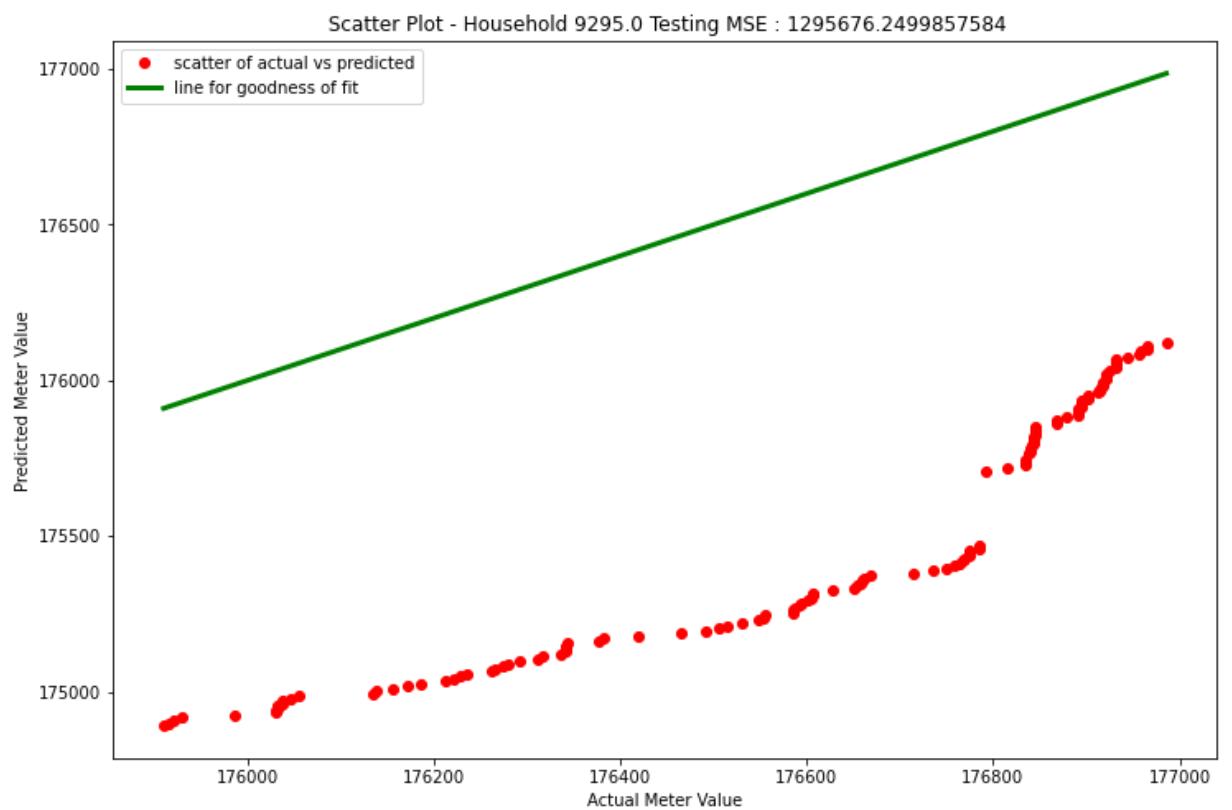


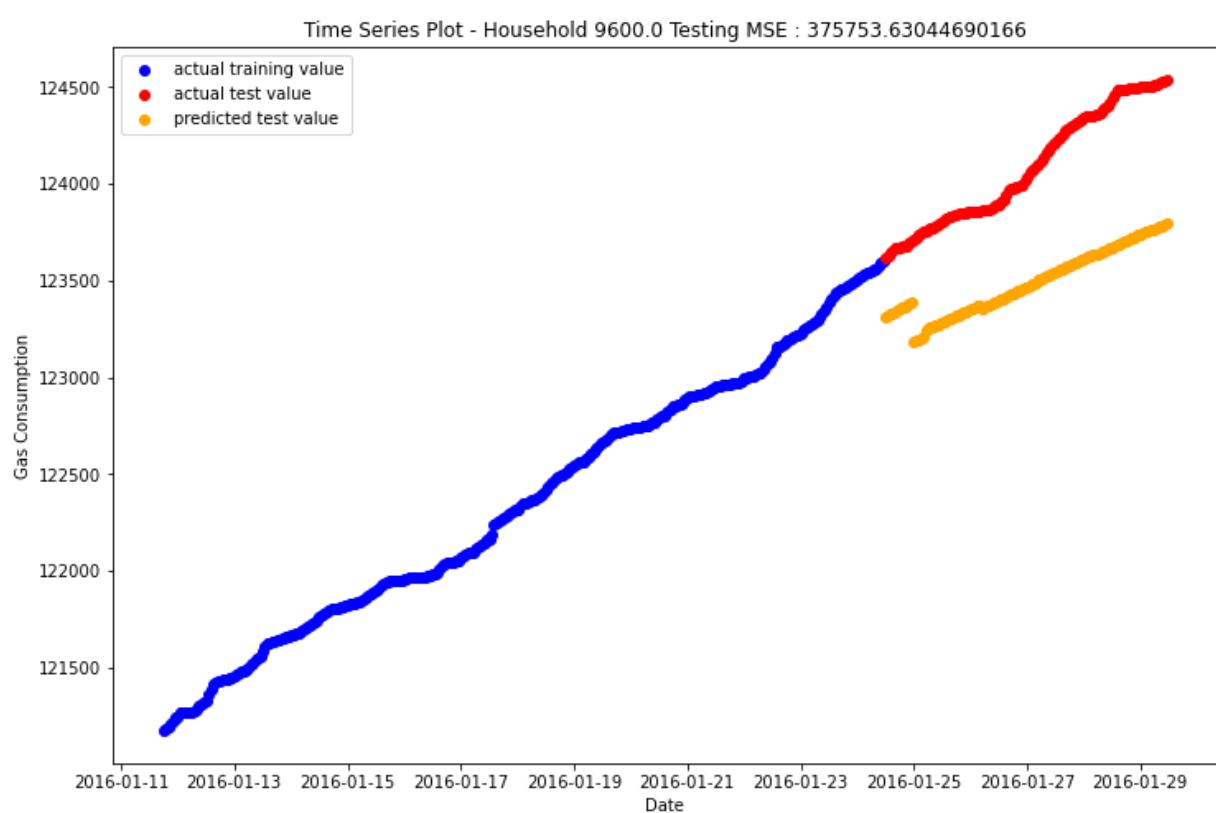
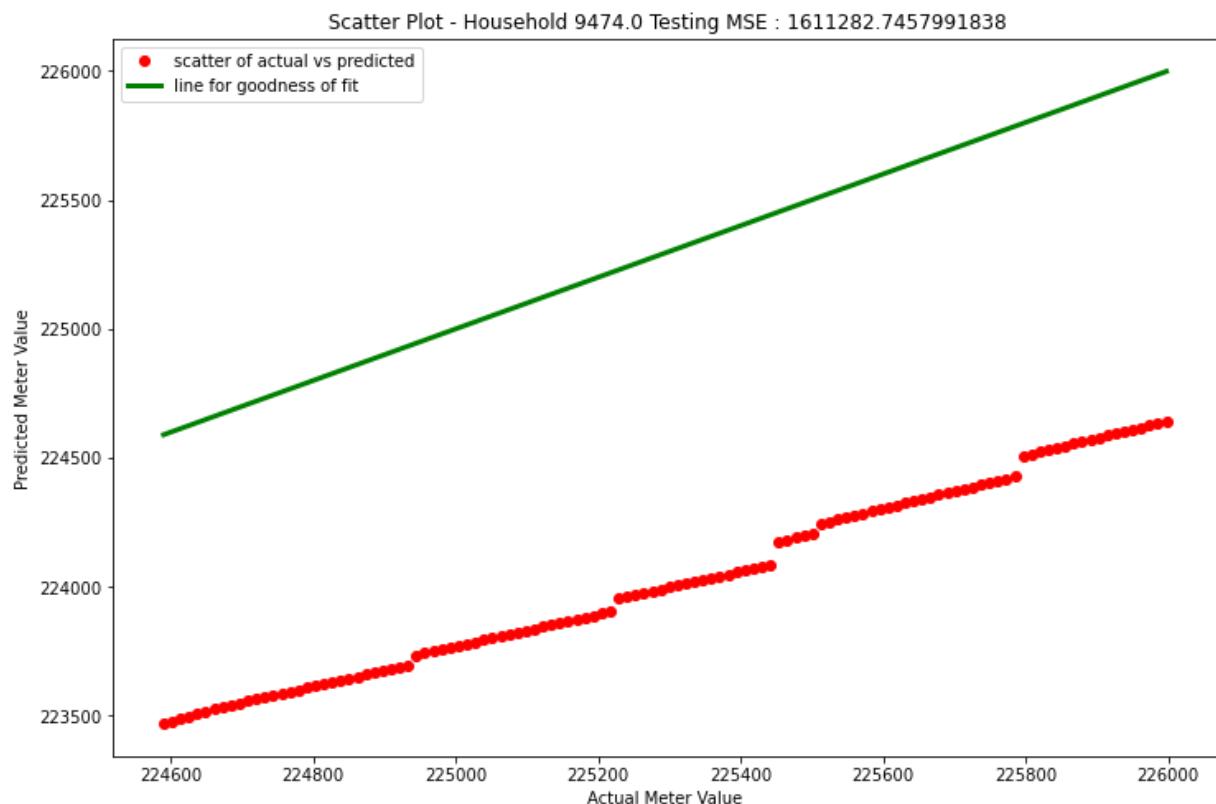


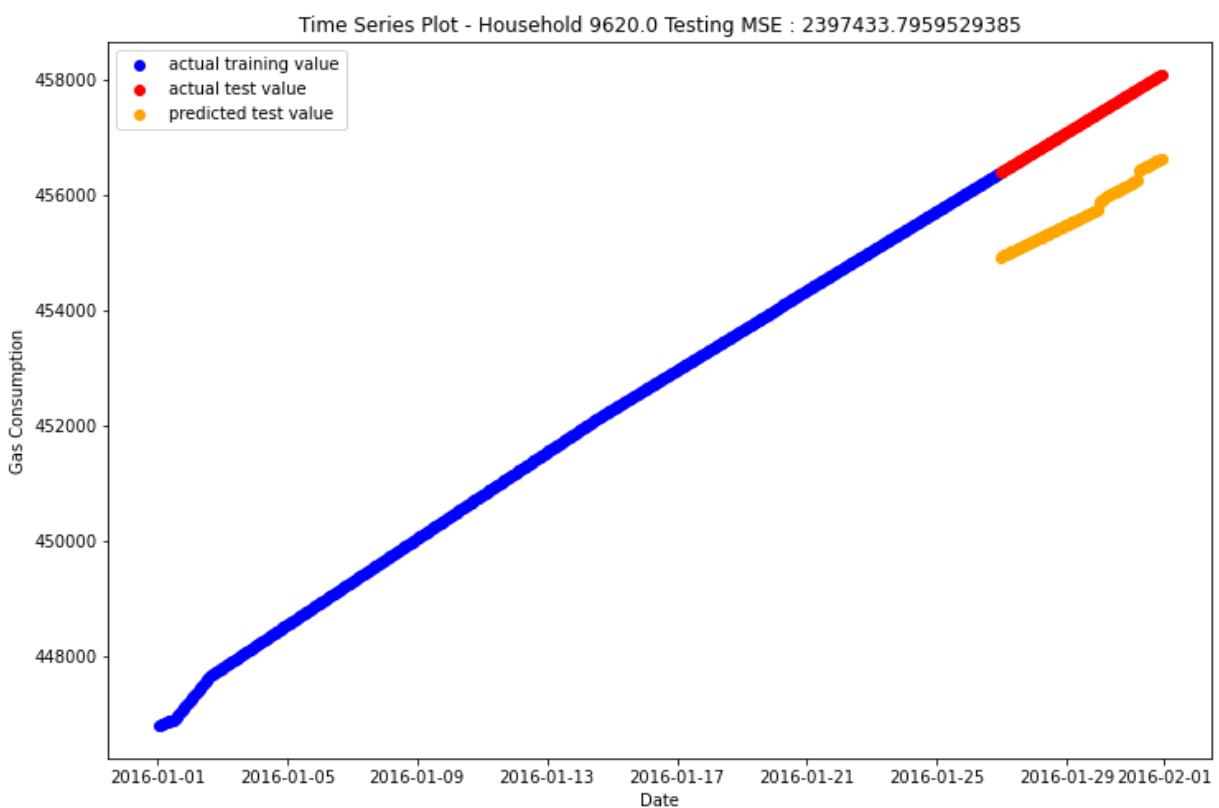
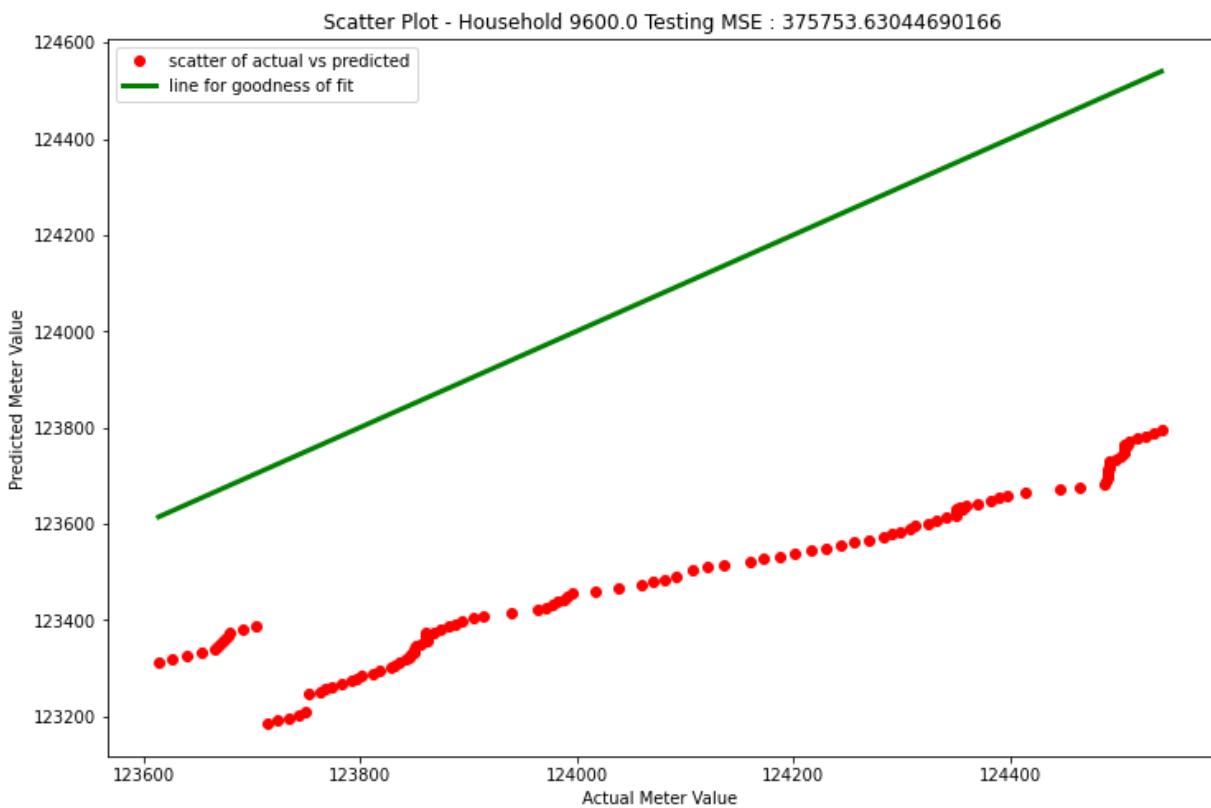


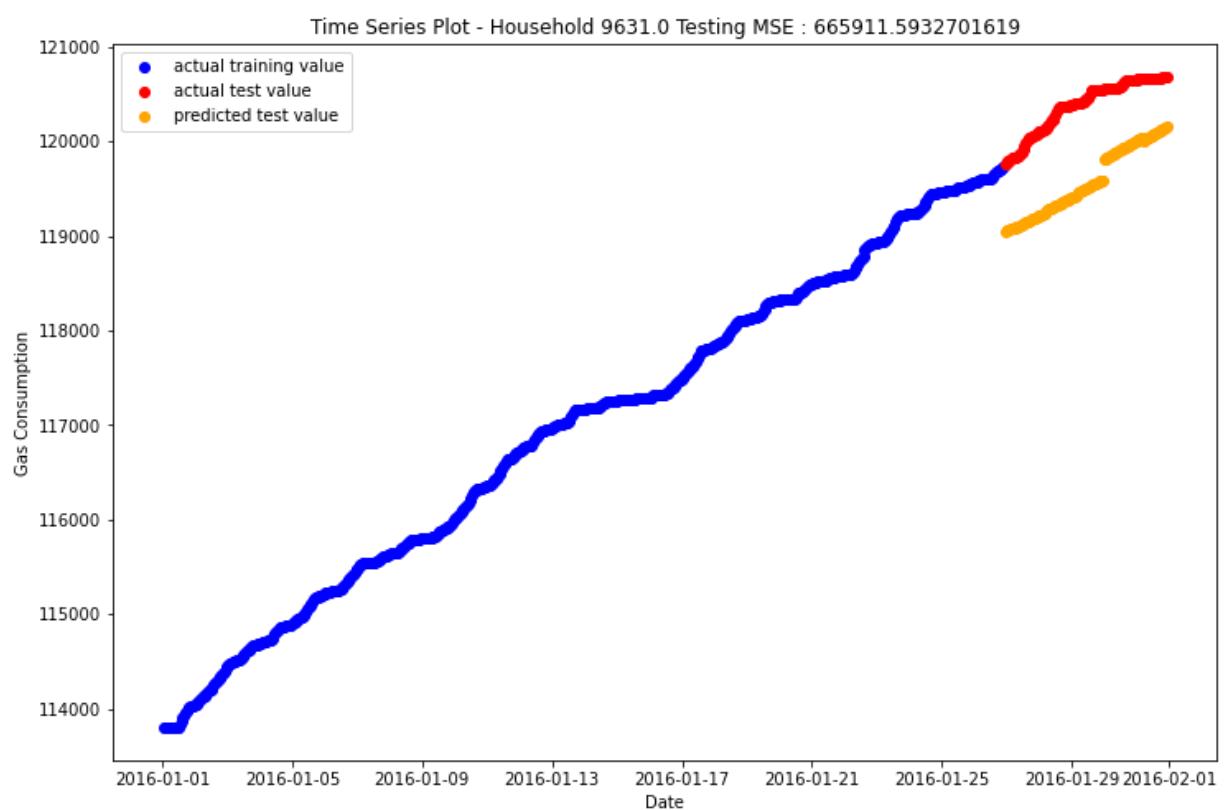
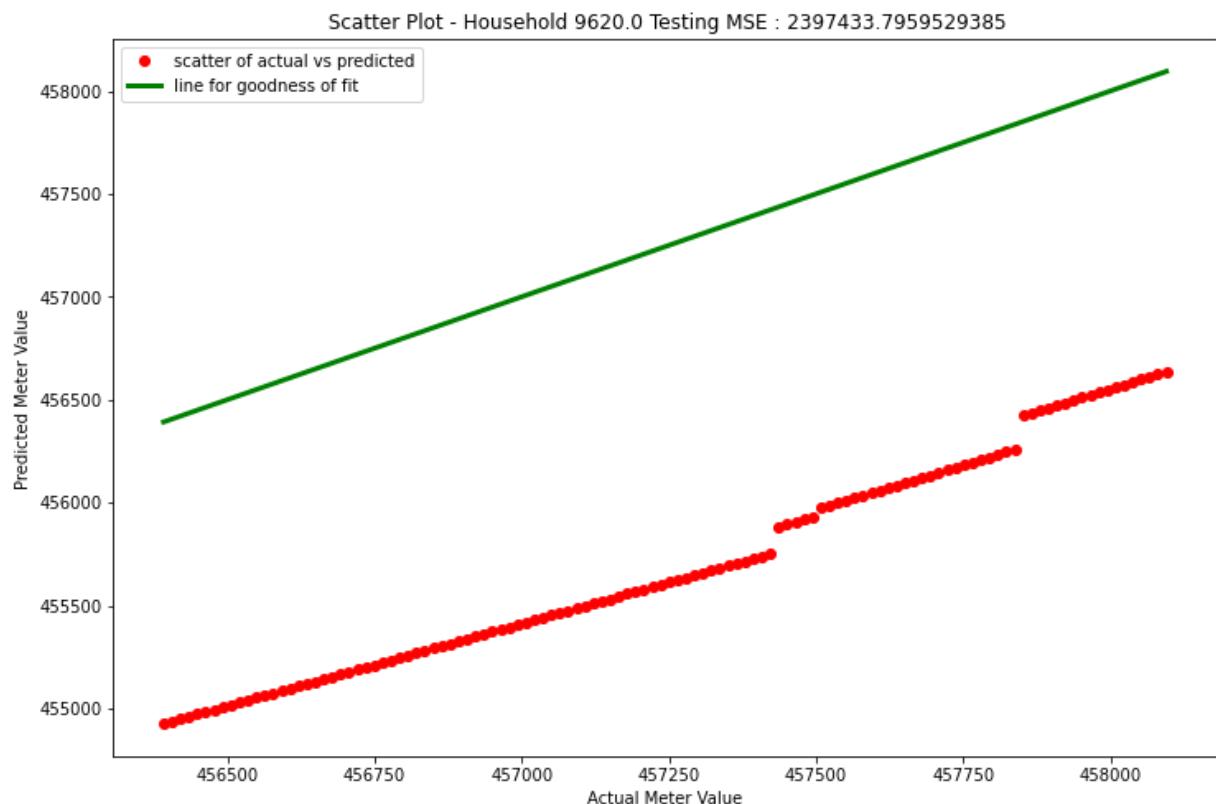


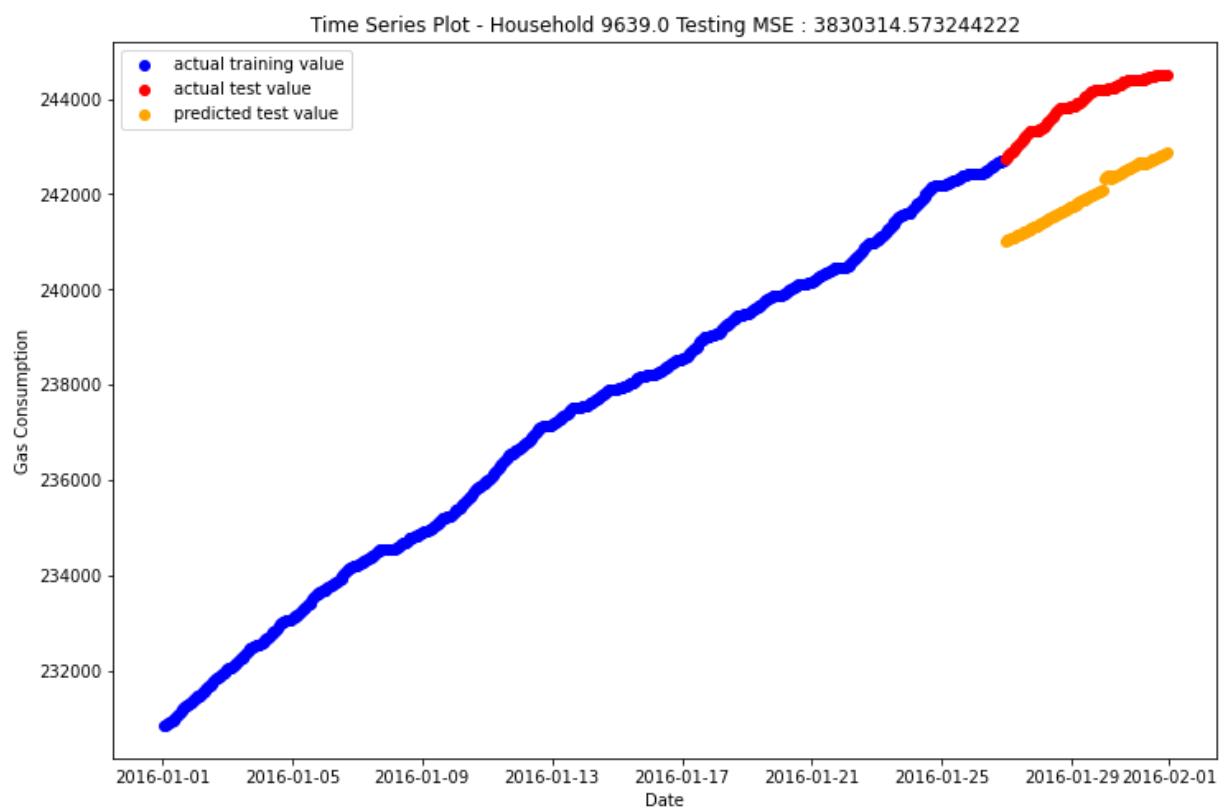
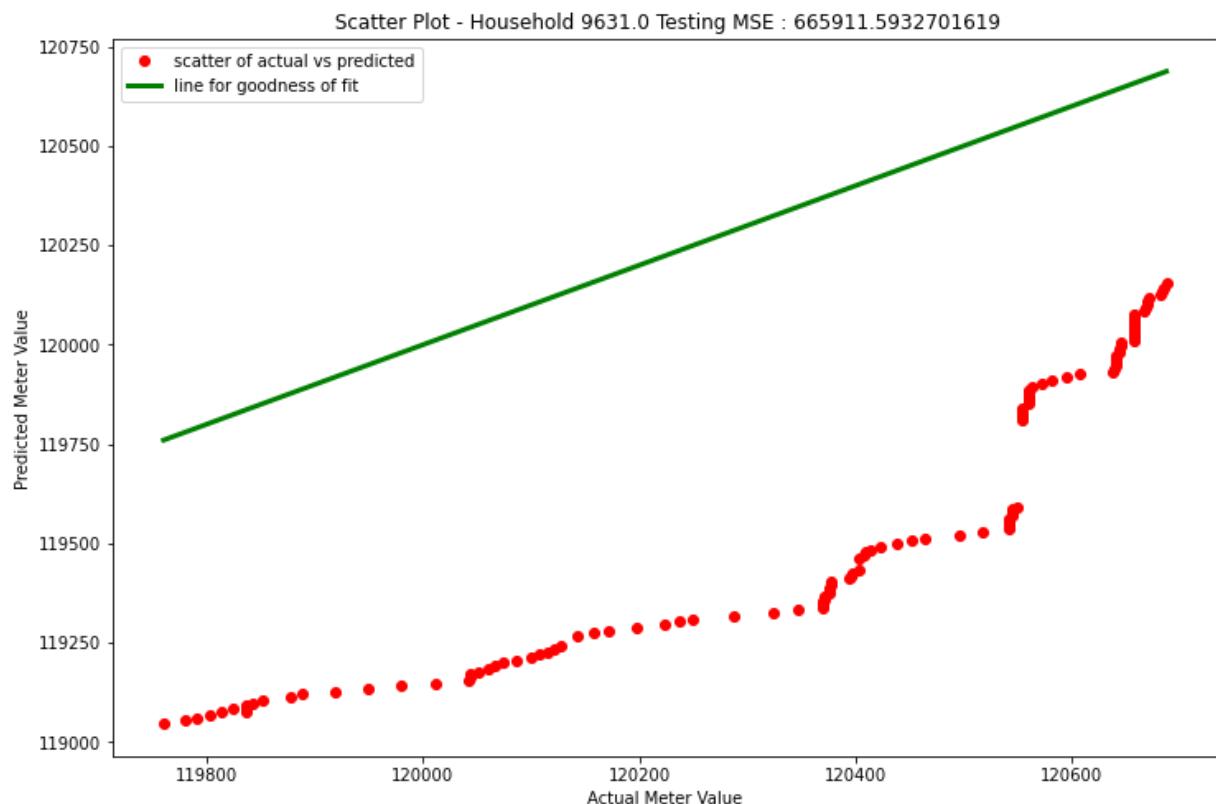


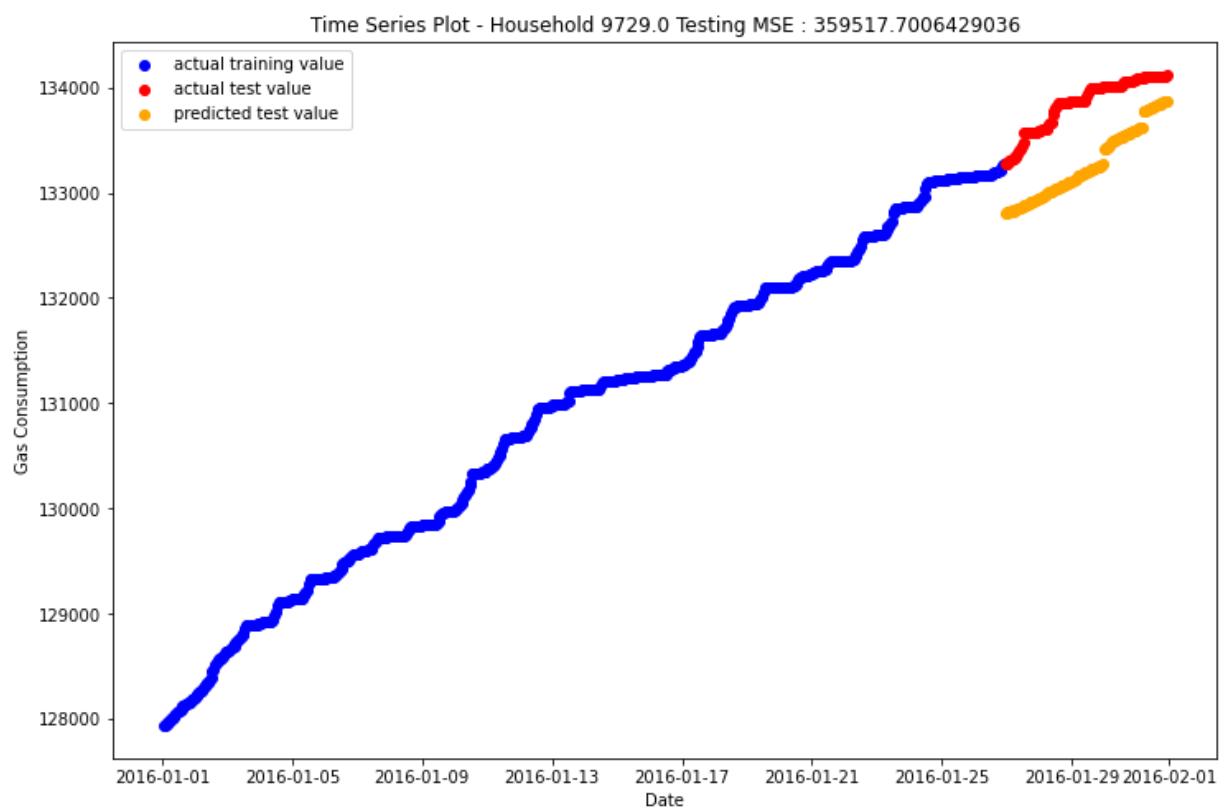
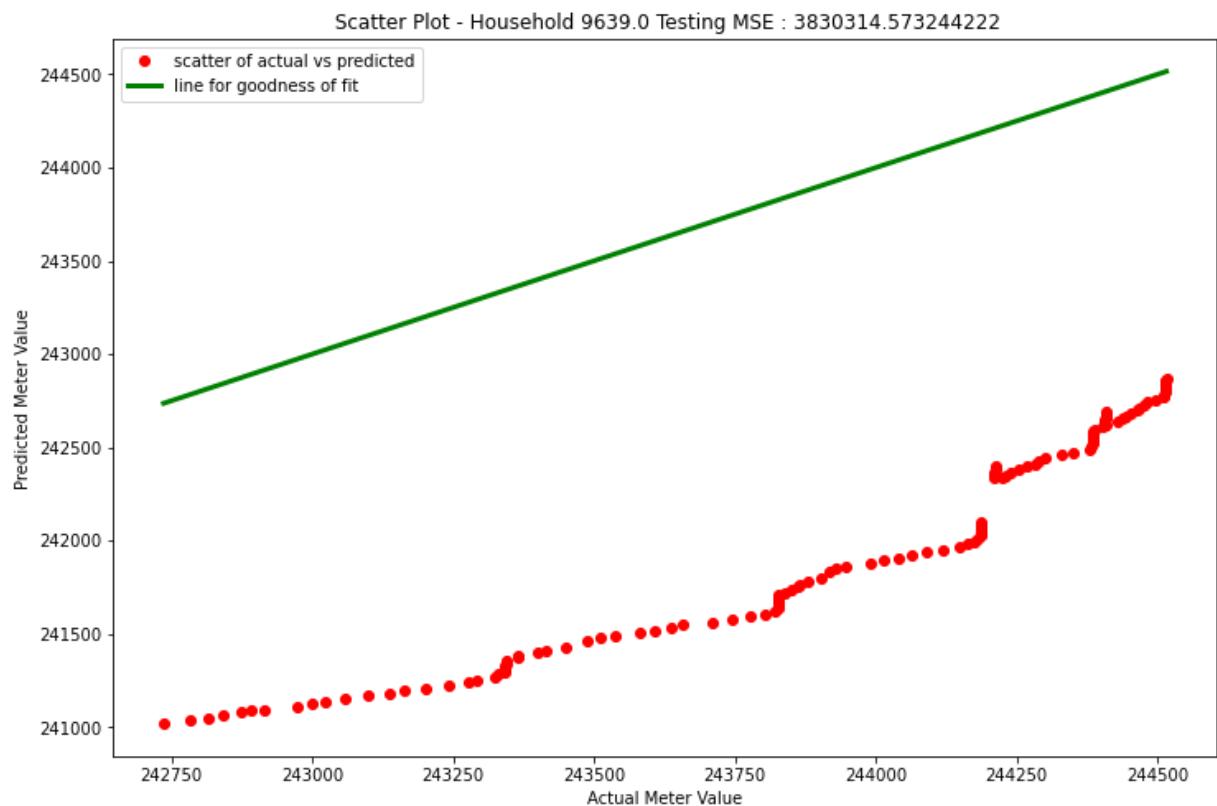




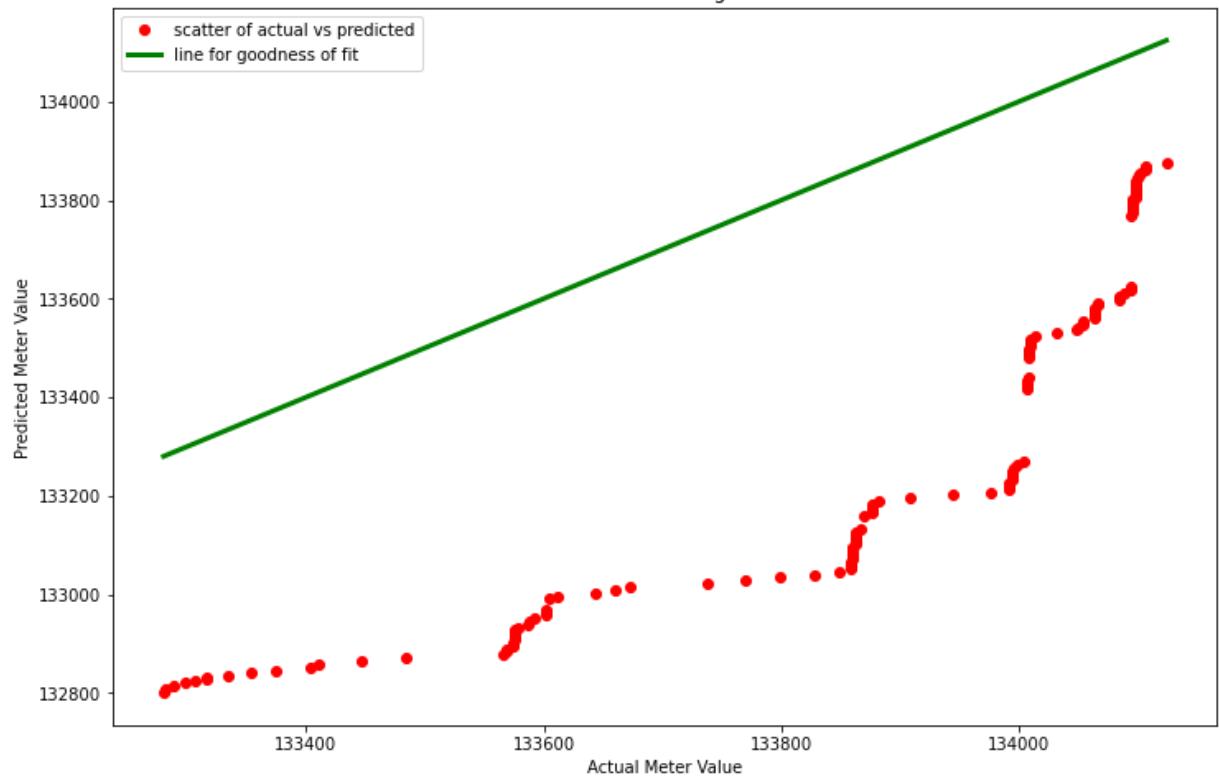




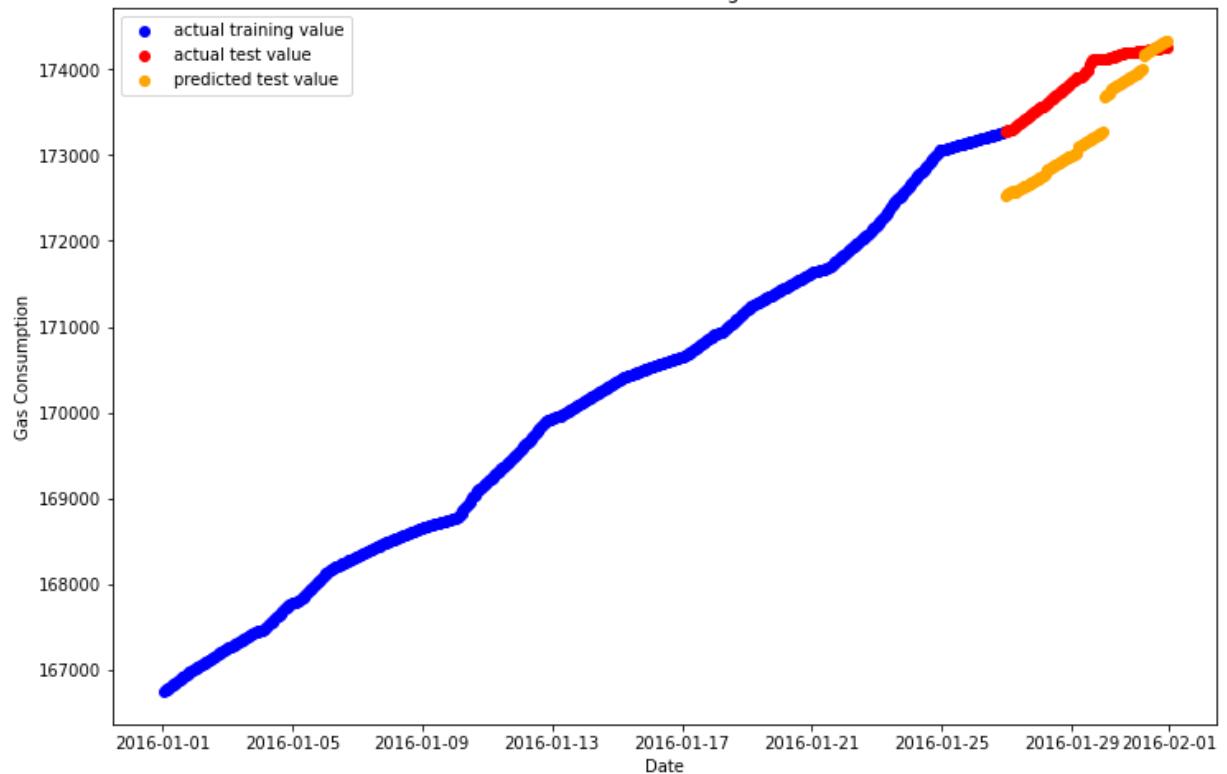


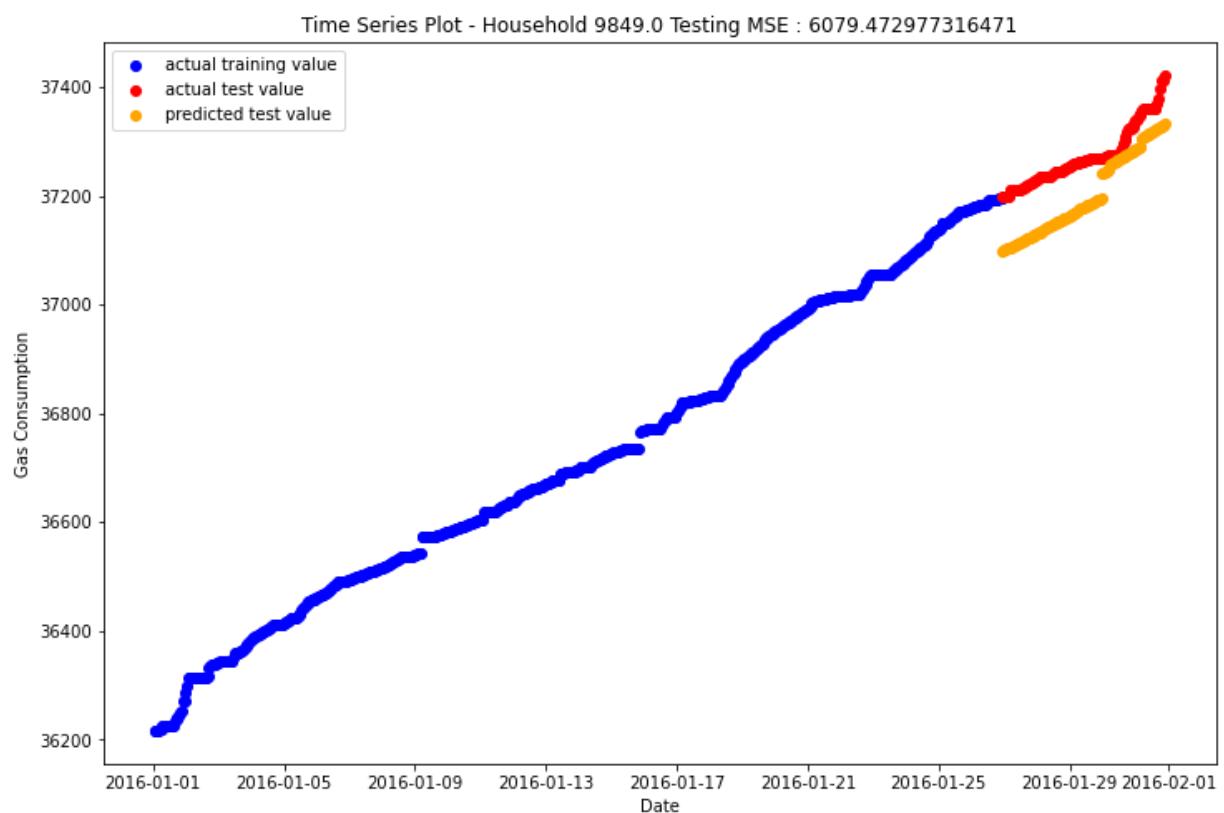
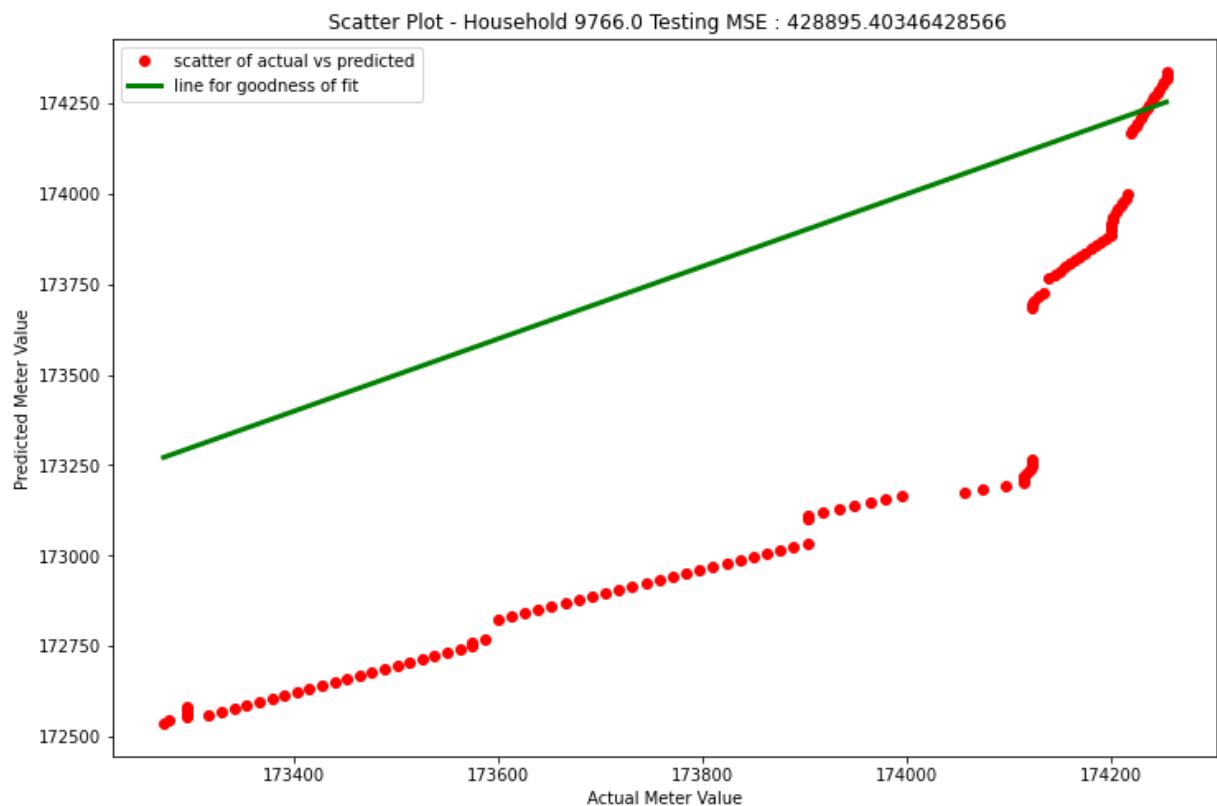


Scatter Plot - Household 9729.0 Testing MSE : 359517.7006429036

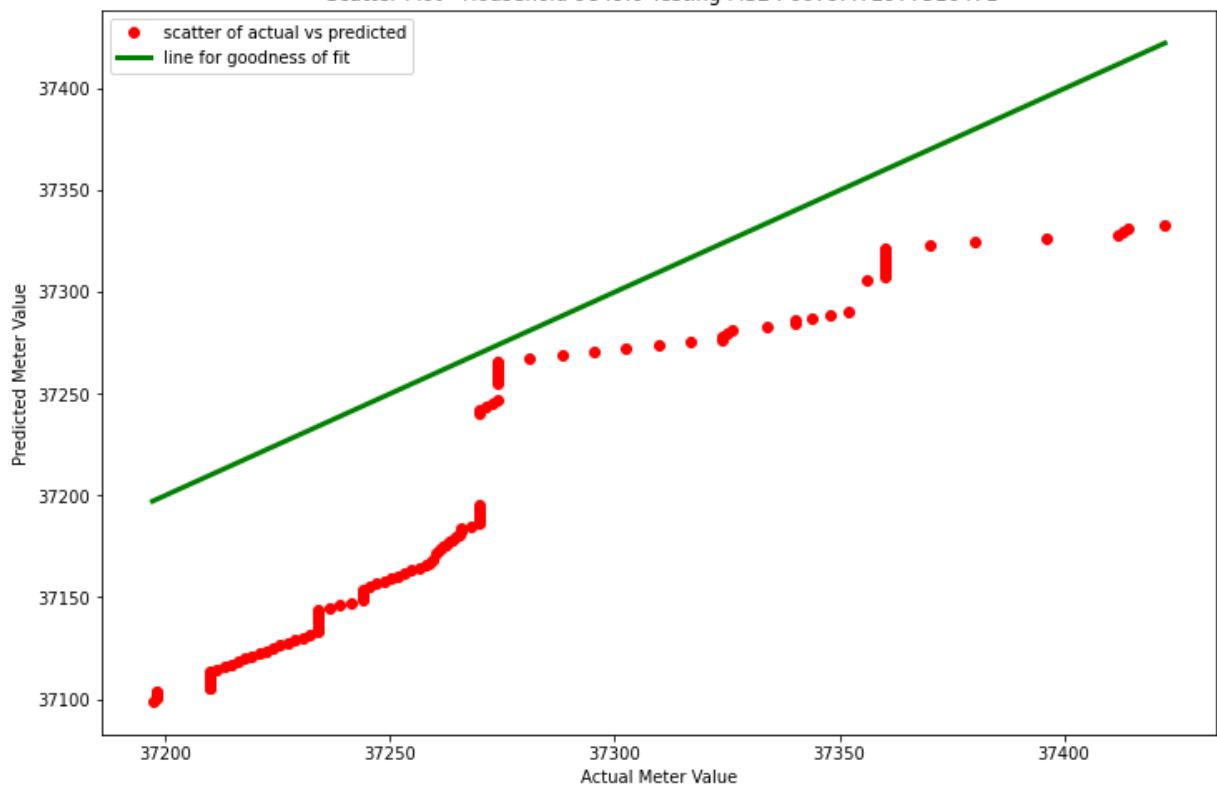


Time Series Plot - Household 9766.0 Testing MSE : 428895.40346428566

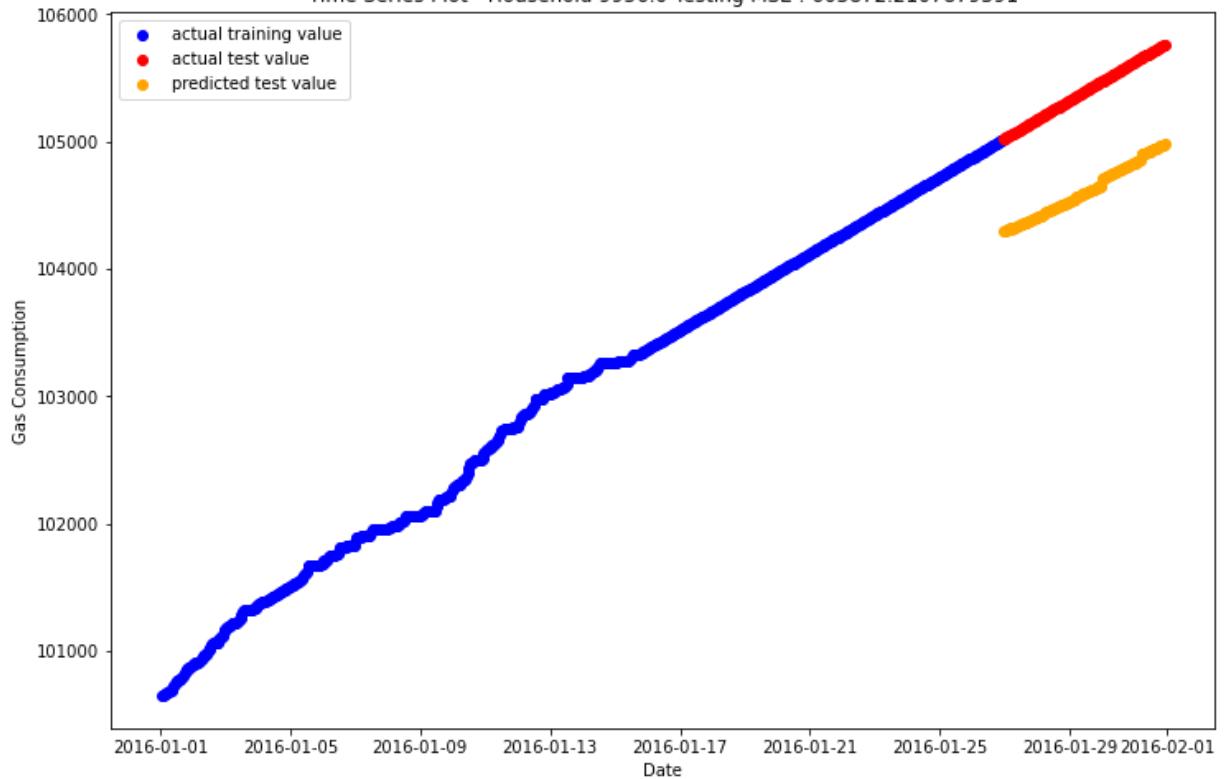


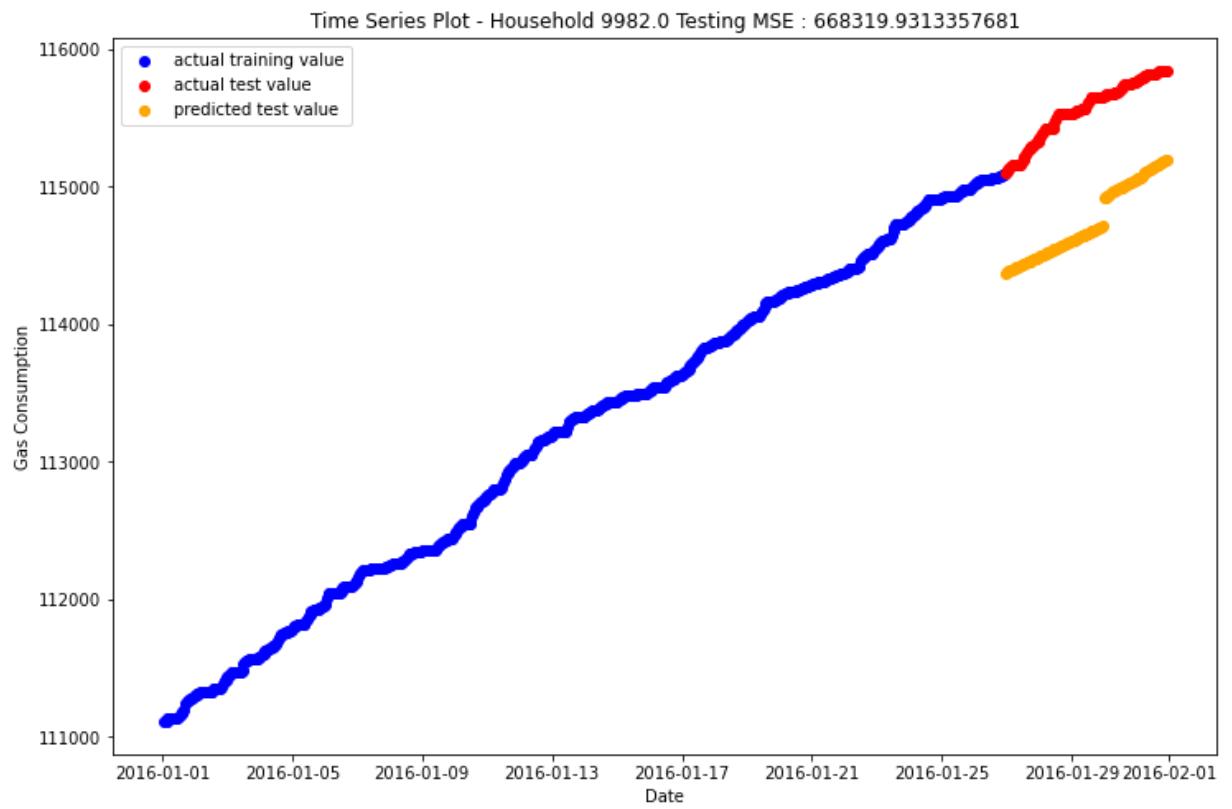
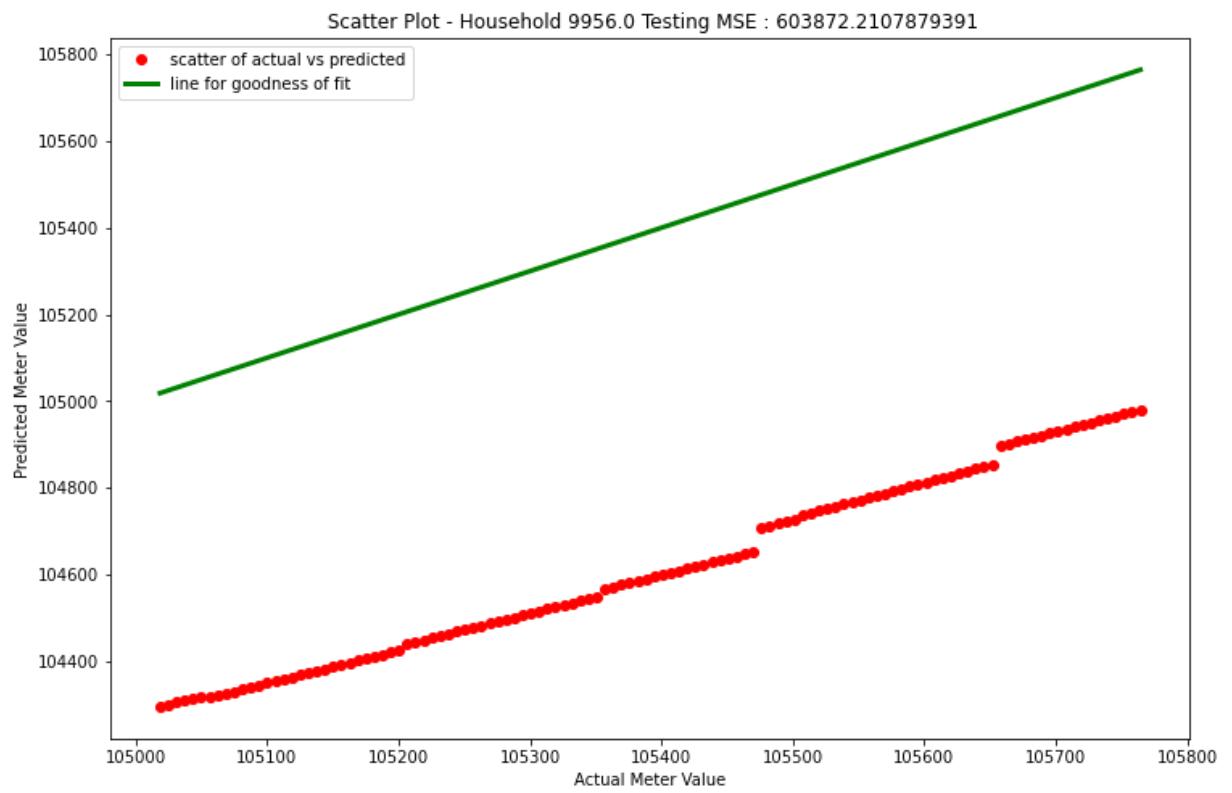


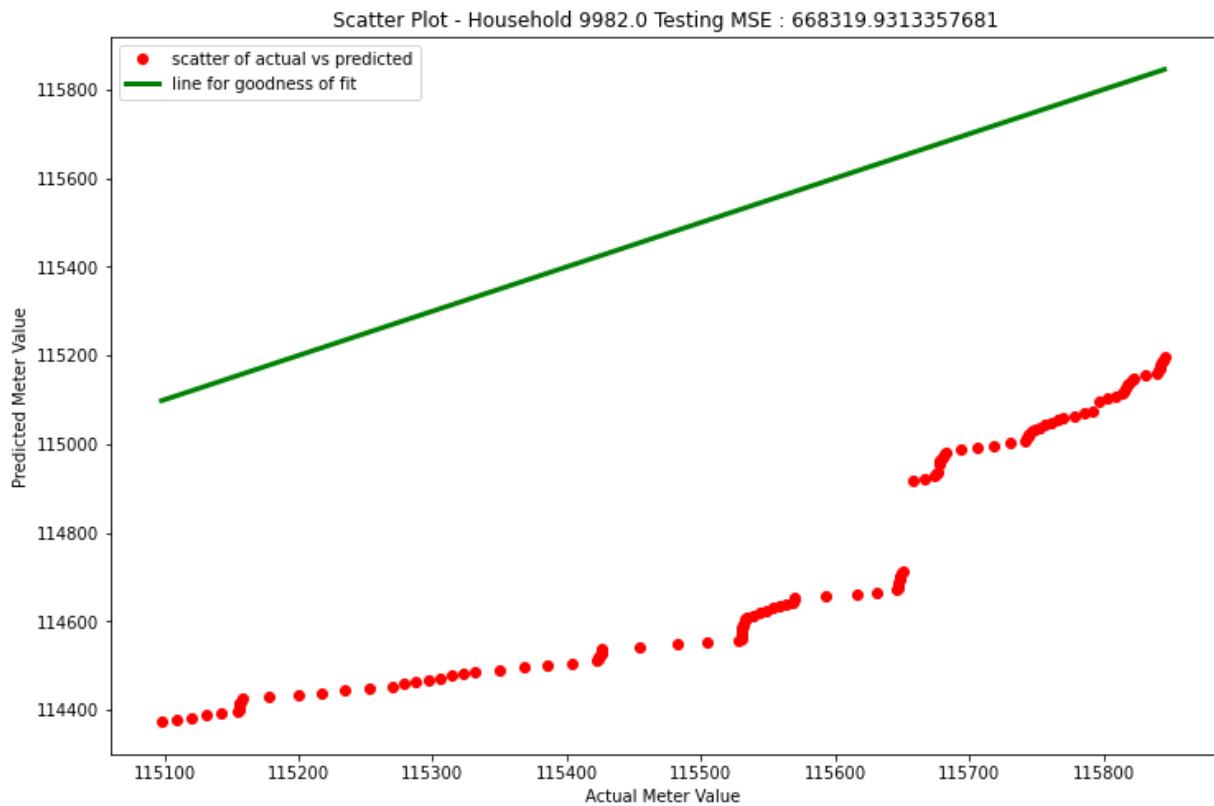
Scatter Plot - Household 9849.0 Testing MSE : 6079.472977316471



Time Series Plot - Household 9956.0 Testing MSE : 603872.2107879391







Above, the timeseries and scatter plots for the respective households can be seen. The respective testing MSE value (derived from the testing dataset for each respective household) has been calculated and reflected at the title of each plot. Upon close inspection and observation, again the obtained MSE values are not significantly lower than what we obtained for question 2 or the linear regression model above. With the weather attributes, we expected the accuracy to go up and MSE values to come down. That clearly don't seem to be the case. So what happened?

## Section 2.5: Analysis

Above, even with the inclusion of the weather attributes, there was not significant increase in the accuracy.

There are a few reasons that we could come up that we thought could explain why we were not able to significantly improve the performance of the model.

### Reason 1: Using daily weather attributes as hourly weather attributes.

In the weather dataset, the attributes were provided in daily basis. However, we were using hourly meter readings and wanted to convert the daily weather attributes to hourly weather attributes so that we could easily add them to our hourly meter readings dataset. So, we ended up taking the daily weather values and using it as the hourly weather values for all the 24 hours in that particular day. This should have been avoided because clearly, we cannot assume that the weather will be the same for every hour throughout the whole day. For instance, it is common for the temperature to be hot during the day and colder during the night. And as such, we would probably expect the gas consumption to rise in the night due to colder temperature. But by using the average daily temperature for both the day and night (instead of the actual temperature on that hour), the models were probably unable to figure out the correlation between temperature and gas consumption as the temperature value fitted to the model for both the day and night hours was the same. Thus, we should have used the exact hourly weather

data (comprising of hourly weather readings) instead. However, such data was not readily available which made us use the daily weather data in the first place. Nonetheless, we have now learnt an important lesson that not only the attributes selected should be related to the dependent variable (that we want to predict) but the values for the selected attribute need to be accurate as well. If not, we might be able to obtain decent results as in our case but not the best and precise results.

#### **Reason 2: Need to consider other important attributes**

Weather may have an impact on gas consumption but not necessarily be the most important factor that decides gas consumption. For instance, family and house demographics might have a bigger role to play in influencing gas consumption than weather. Unfortunately, such data was never readily available in the first place despite our best efforts.

#### **Reason 3: Neural network may not have converged**

The neural network may not have been converged within the 150 epochs. As we didn't have much time, we were unable to experiment with different parameters and fine-tuning hyperparameters.

#### **Reason 4: Not much data**

As discussed at the top, including datapoints from older consumptions led to more noise during training. However, if there were data from more households from the same neighbourhood, that could have increased the performance of the models. In addition, if data during the summer and spring seasons were collected and used in the model, that could have enhanced the accuracy as well.

#### **Reason 5: Not random train-test split**

The train-test split was made based on the time sequence and was not a random split. Using a random split could have made the model more robust but unfortunately, we didn't have a large dataset to have several contiguous and sufficiently large train-test splits.

#### **Other remarks:**

For our neural network, we used MSE as our evaluation metric. However, in reality, we may need to treat overestimation and underestimation differently depending on how the gas provider wants to treat it.

## **Section 2.6: What's next?**

So, our attempt to build a highly accurate model by including new useful attributes did not work as much as expected. Still, we have learnt some useful lessons and now have better appreciation towards some of the actual real-life challenges with building forecast models. With that, in the next section, we will attempt to explore our next idea to create a highly accurate model, which is to use a LSTM model.

---

## **Section 3: Building a better forecast model - Using LSTM models**

In this section, we will explore our second idea to create a better forecast model, using Long Short-Term Memory (LSTM) network model. We will discuss the motivation behind creating a LSTM model, our development process and then evaluate its performance in the end.

In summary, we built a LSTM model that takes in 1 week of meter values to predict the hourly meter values for the next week. In other words, our LSTM model will take in 168 meter values (24 hourly readings x 7 past days) and output 168 future meter value predictions (24 hourly readings x 7 future days). To train this model, we made use of 1 month of data.

## Section 3.1: Motivation behind using LSTM models

LSTM is a type of recurrent neural network that is capable of learning long term dependencies in data. As the name itself suggests, LSTM consist of both a short-term and a long-term memory component. And as such, LSTM models are widely used to model time series forecasting problems. These are problems where you have either a single or multiple series of observations and a model is required to learn from the series of past observations to predict the next value(s) in the sequence.

It is very clear that our problem falls under the category of time series forecasting problems. As LSTM models are well-suited to predict time series data, why not try it to forecast gas consumption in our case?

**As compared to linear regression, we are able to get a forecast where it takes into account the time of day, as compared to linear regression which assumes constant usage throughout the day.**

This is a valuable piece of information for both the gas companies and the consumers since

- gas companies would be able to know peak usage times, and thus be ready to meet the demands during these period, ensuring availability (can keep up with the throughput of gas required at a certain point in time)
- consumers can get a more personalized forecast of their usage since it is based on their usage patterns, and can directly see this pattern in their forecast.

## Section 3.2: Key details about our LSTM model

1. As written above, we will aim to predict hourly readings of a particular week with the past 7 days of hourly readings.
2. Each household will have its own LSTM model.
3. Training set will consist of 1 month of training data, which would be January of 2016. We use a span of a single month as it is sufficient data to infer a particular household's usage habits.
4. Our model uses usage data rather than raw meter\_values, as through our experimentation, LSTM performs better when forecasting a periodic signal as compared to extrapolating from an increasing set of values.
5. Scaling and normalizing of hourly usage values is performed before fitting to the model for better accuracy.
6. `split_sequence` function is used to generate samples which will consist of a particular number of past time steps as input and a particular number of step predictions as output.
7. MSE is used as our evaluation metric.

8. Since we are using hourly usage values (instead of actual meter readings), our model will output usage values. Therefore, we will convert these back into predicted future meter readings similar to that of what is given to us in the dataset.

### Section 3.3: Algorithm approach

Below is a quick summary of the steps involved.

1. **Data import:** We will first load the dataset and then remove houses which have incomplete data as it can affect the performance of the model.
2. **Data filter:** We will be creating the LSTM model on data after 1st January 2016. Hence, we will first filter the data to contain data instances that are within 7 weeks from 1st January 2016.
3. **Compute diff value:** Instead of using the actual meter values for training and testing, for a particular household, we will be first computing the difference in the meter values from the meter value on 1st January 2016 for that household. This will be used for the LSTM model for that particular household. We will refer to this as the diff data or diff values.
4. **Normalization:** We will scale and normalize the diff values accordingly since we do not know the distribution of the data with the belief that this will improve the performance of the model.
5. **Data preparation:** Next, we will transform this diff data to obtain samples which will consist of a particular number of past time steps as input and a particular number of step predictions as output. Our model requires 1 week diff values to predict the next 1 week diff values. Hence, each of our samples will consist of 168 past meter diff values (24 hourly readings x 7 past days) as input and 168 future meter diff values (24 hourly readings x 7 future days) as output. Therefore, we generate a set of these sequences of inputs and outputs using our split\_sequence function on our dataset to generate our train and test sets.
6. **Data Reshape:** The LSTM library expects data to have a three-dimensional structure of [samples, timesteps, features]. In our case, we only have one feature. So, we will reshape the diff data samples accordingly.
7. **Create the LSTM model:** Our LSTM model will use the "Adam" optimizer.
8. **Fit the training data:** We will be using the first four weeks of diff data samples for training.
9. **Predict for test data:** Using the obtained model, we will use the remaining three weeks of diff data samples for testing.
10. **Scale back predictions to actual value:** The values that we obtain from the model would be the scaled diff values and not predicted meter readings in the original form. So, we will invert the transformation to return the values back into the original scale so that we can compare these predicted values (no longer diff values anymore) with the actual meter readings and calculate error scores.
11. **Plot predicted vs actual:** Plot the predicted meter readings vs actual meter readings.
12. **Deployment** After training the model and evaluating its performance, we deploy the model into a telegram bot service as would be mentioned in section 4, where the trained model will be using the data in the previous week to make predictions. To see it in action, take a look at section 4 of this notebook, our video presentation and the code (specifically bot.py and tfserver.py under the "Python Telegram Bot" zip file) for the implementation details.

### Section 3.4: Code

## Step 1. Data import and manipulation

In this stage, we will first load the dataset (that we used for question 2 and the models above) and then first remove households with data ID 83 and 140 as they have incomplete data.

In [145...]

```
# Load the dataset
df = pd.read_csv('Question_2_dataset.csv', index_col='localminute', parse_dates=True)

# remove households 83 and 140 as they have incomplete data
unique_households = df['dataid'].unique()
unique_households.sort()
unique_households = np.delete(unique_households, [83,140])
```

## Steps 2 - 11

The rest of the steps are done below.

In [147...]

```
mse_lstm = []
ADDITIONAL_WEEKS_TO_PREDICT = 3

# this function is used for step 5: data preparation
def split_sequence(data_seq, steps_in_no, steps_out_no):
    ...

    Given the data sequence, split the given data sequence into multiple samples.
    Each sample will have a specified number of time steps as input
    and specified number of time steps as output.
    For instance, if data is [10, 20, 30] and we want the sample to have 2 past read
    and 1 future reading as output, the output returned from the function will be [1

    Parameters
    -----
    data_seq: data sequence to be split
    steps_in_no: indicate how many time steps as input
    steps_out_no: indicate how many time steps as output

    Returns
    -----
    X: input time steps
    y: output time steps
    ...
    X, y = list(), list()
    for i in range(len(data_seq)):
        # find end of pattern
        end_ix = i + steps_in_no
        out_end_ix = end_ix + steps_out_no
        # check within data sequence
        if out_end_ix > len(data_seq):
            break
        # gather and append time steps respectively
        seq_x, seq_y = data_seq[i:end_ix], data_seq[end_ix:out_end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return np.asarray(X), np.asarray(y)

# step 2: data filter
index = pd.date_range("2016-01-01", periods=24 * 9 * 7, freq="H")
test_index = index[24 * 28: 24* 35]
pred_index = index[24 * 35: 24 * 35 + 24 * 7 * (ADDITIONAL_WEEKS_TO_PREDICT + 1)]

for house in unique_households:
```

```

try:

    checkpoint_path = "./checkpoints/" + str(house) + "/cp.ckpt"
    checkpoint_dir = os.path.dirname(checkpoint_path)
    cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                    save_weights_only=True,
                                                    verbose=0)

    # step 3: compute diff values
    house_df = df.loc[df['dataid'] == house]
    house_df['diff'] = house_df['meter_value'].diff()
    house_df = house_df.loc[datetime.fromisoformat("2016-01-01 00:00:00+00:00"):

    # step 4: normalization
    scaler = StandardScaler()
    scaler = scaler.fit(house_df['diff'].values.reshape(len(house_df['diff']), 1))
    scaled = scaler.transform(house_df['diff'].values.reshape(len(house_df['diff']), 1))

    # step 5: data preparation
    n_steps_in, n_steps_out = 24 * 7, 24 * 7
    X, y = split_sequence(scaled[:24 * 28], n_steps_in, n_steps_out)

    # step 6: data Reshape
    # reshape from [samples, timesteps] into [samples, timesteps, features]
    n_features = 1
    X = X.reshape((X.shape[0], X.shape[1], n_features))

    # step 7: create the LSTM model
    # define model
    model = tf.keras.Sequential()
    model.add(layers.LSTM(168, return_sequences = True, input_shape=(n_steps_in, 1)))
    model.add(layers.LSTM(168))
    model.add(layers.Dense(n_steps_out))
    model.compile(optimizer='adam', loss='mse')

    # step 8: fit the training data
    # fit model
    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='loss',
                                                       patience=2,
                                                       mode='min')

    if os.path.exists(checkpoint_dir):
        model.load_weights(checkpoint_path).expect_partial()
        print("model loaded" + checkpoint_path)
    else:
        model.fit(X, y, epochs=20, verbose=0, callbacks=[early_stopping, cp_callback])

    # step 9: predict for test data
    # demonstrate prediction
    x_input = np.asarray(scaled[24 * 28: 24 * 35])
    x_input = x_input.reshape((1, n_steps_in, n_features))
    predictions = []
    yhat = model.predict(x_input, verbose=1)
    res = yhat

    ADDITIONAL_WEEKS_TO_PREDICT = 3
    for i in range(ADDITIONAL_WEEKS_TO_PREDICT):
        x_input = np.asarray(yhat)
        x_input = x_input.reshape((1, n_steps_in, n_features))
        yhat = model.predict(x_input, verbose=1)
        res = np.concatenate((yhat, res), axis=1)

    # step 10: scale back predictions to actual value

```

```

predicted = pd.Series(scaler.inverse_transform(res)[0], pred_index )
predicted = predicted.cumsum() + house_df['meter_value'][24 * 35 - 1]

# step 11: plot predicted vs actual
# compute mse
mean_squared_error = mse(predicted, house_df['meter_value'][24 * 35: 24 * 35])
mse_lstm.append(mean_squared_error)

# plot graphs
plt.figure(figsize=(12, 8))
# plt.plot(house_df.index[:24 * 21], scaled[:24* 21], c='blue' , linewidth=3
plt.plot(test_index, house_df['meter_value'][24 * 28: 24* 35] , c='orange'
plt.plot(pred_index[: len(house_df['meter_value'])[24 * 35: 24 * 35 + 24 * 7
plt.plot(pred_index, predicted , c='green' , label = 'predicted', linewidth

# plt.plot(test_df['meter_value'], test_df['mean_predicted_meter_value'], 'o
plt.xlabel('Actual Meter Value')
plt.ylabel('Predicted Meter Value')
plt.title(f'Household {house} \n Testing MSE: {mean_squared_error} \n')
# plt.title(f'Household {house} \nTesting SVR MSE : {mean_svr_squared_error}
plt.legend()
plt.show
except Exception as e:
    print(e, 'cannot predict for house:', house)

```

```

model loaded./checkpoints/35.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 994us/step
1/1 [=====] - 0s 994us/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/44.0/cp.ckpt
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/77.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 705us/step
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/94.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 975us/step
1/1 [=====] - 0s 999us/step
model loaded./checkpoints/114.0/cp.ckpt
1/1 [=====] - 0s 995us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/187.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 968us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/222.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 967us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/252.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 961us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/370.0/cp.ckpt

```

```
1/1 [=====] - 0s 995us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/483.0/cp.ckpt
1/1 [=====] - 0s 968us/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 969us/step
model loaded./checkpoints/484.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/661.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 968us/step
model loaded./checkpoints/739.0/cp.ckpt
1/1 [=====] - 0s 971us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 968us/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/744.0/cp.ckpt
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/871.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/1042.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 992us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/1086.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/1103.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 1ms/step
Found input variables with inconsistent numbers of samples: [672, 457] cannot predict for house: 1103.0
model loaded./checkpoints/1185.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/1283.0/cp.ckpt
1/1 [=====] - 0s 994us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/1403.0/cp.ckpt
cannot reshape array of size 0 into shape (1,168,1) cannot predict for house: 1403.0
model loaded./checkpoints/1415.0/cp.ckpt
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/1507.0/cp.ckpt
```

```
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/1556.0/cp.ckpt
1/1 [=====] - 0s 965us/step
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 965us/step
1/1 [=====] - 0s 999us/step
model loaded./checkpoints/1589.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 972us/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/1619.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/1697.0/cp.ckpt
1/1 [=====] - 0s 995us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/1714.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/1718.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/1790.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 960us/step
model loaded./checkpoints/1791.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 858us/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/1792.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/1800.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 971us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/1801.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 995us/step
model loaded./checkpoints/2018.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 964us/step
1/1 [=====] - 0s 994us/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/2034.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 1ms/step
```

```
model loaded./checkpoints/2072.0/cp.ckpt
1/1 [=====] - 0s 959us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/2094.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/2129.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/2233.0/cp.ckpt
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 973us/step
model loaded./checkpoints/2335.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/2378.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 995us/step
1/1 [=====] - 0s 951us/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/2449.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/2461.0/cp.ckpt
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 999us/step
model loaded./checkpoints/2470.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 994us/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/2575.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/2638.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/2645.0/cp.ckpt
1/1 [=====] - 0s 962us/step
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 993us/step
1/1 [=====] - 0s 2ms/step
Found input variables with inconsistent numbers of samples: [672, 564] cannot predict for house: 2645.0
model loaded./checkpoints/2755.0/cp.ckpt
cannot reshape array of size 0 into shape (1,168,1) cannot predict for house: 2755.0
model loaded./checkpoints/2814.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
```

```
Found input variables with inconsistent numbers of samples: [672, 156] cannot predict for house: 2814.0
model loaded./checkpoints/2818.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 993us/step
1/1 [=====] - 0s 996us/step
model loaded./checkpoints/2945.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 992us/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/2946.0/cp.ckpt
cannot reshape array of size 0 into shape (1,168,1) cannot predict for house: 2946.0
model loaded./checkpoints/2965.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/2980.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 964us/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/3036.0/cp.ckpt
cannot reshape array of size 0 into shape (1,168,1) cannot predict for house: 3036.0
model loaded./checkpoints/3039.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 544us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/3134.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/3310.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 995us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/3367.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/3527.0/cp.ckpt
1/1 [=====] - 0s 956us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/3544.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 994us/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/3577.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 963us/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/3635.0/cp.ckpt
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 1000us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/3723.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 0s/step
```

```
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 996us/step
model loaded./checkpoints/3778.0/cp.ckpt
1/1 [=====] - 0s 924us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/3849.0/cp.ckpt
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/3893.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 996us/step
model loaded./checkpoints/3918.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/4029.0/cp.ckpt
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/4031.0/cp.ckpt
1/1 [=====] - 0s 959us/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/4193.0/cp.ckpt
1/1 [=====] - 0s 992us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/4228.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/4296.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/4352.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/4356.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/4373.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 968us/step
model loaded./checkpoints/4421.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/4447.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
```

```
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 996us/step
model loaded./checkpoints/4514.0/cp.ckpt
1/1 [=====] - 0s 956us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/4671.0/cp.ckpt
1/1 [=====] - 0s 983us/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 998us/step
Found input variables with inconsistent numbers of samples: [672, 533] cannot predict for house: 4671.0
model loaded./checkpoints/4732.0/cp.ckpt
1/1 [=====] - 0s 968us/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 1000us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/4767.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/4998.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/5129.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/5131.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 968us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 999us/step
model loaded./checkpoints/5193.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 995us/step
1/1 [=====] - 0s 962us/step
model loaded./checkpoints/5275.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/5317.0/cp.ckpt
cannot reshape array of size 0 into shape (1,168,1) cannot predict for house: 5317.0
model loaded./checkpoints/5395.0/cp.ckpt
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/5403.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/5439.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/5484.0/cp.ckpt
1/1 [=====] - 0s 995us/step
```

```
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 518us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/5545.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 2ms/step
Found input variables with inconsistent numbers of samples: [672, 668] cannot predict for house: 5545.0
model loaded./checkpoints/5636.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 994us/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/5658.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/5785.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/5810.0/cp.ckpt
1/1 [=====] - 0s 995us/step
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 995us/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/5814.0/cp.ckpt
1/1 [=====] - 0s 1000us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/5892.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/5972.0/cp.ckpt
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 2ms/step
tuple index out of range cannot predict for house: 6101.0
model loaded./checkpoints/6412.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 995us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 990us/step
model loaded./checkpoints/6505.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/6578.0/cp.ckpt
cannot reshape array of size 0 into shape (1,168,1) cannot predict for house: 6578.0
model loaded./checkpoints/6673.0/cp.ckpt
1/1 [=====] - 0s 966us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 1ms/step
Input contains NaN, infinity or a value too large for dtype('float32'). cannot predict for house: 6673.0
model loaded./checkpoints/6685.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 1ms/step
```

```
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/6830.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 1000us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 993us/step
model loaded./checkpoints/6836.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/6863.0/cp.ckpt
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 955us/step
1/1 [=====] - 0s 3ms/step
model loaded./checkpoints/6910.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/7016.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 878us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/7017.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/7030.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/7117.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/7287.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/7429.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 996us/step
model loaded./checkpoints/7460.0/cp.ckpt
1/1 [=====] - 0s 995us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 999us/step
model loaded./checkpoints/7566.0/cp.ckpt
cannot reshape array of size 0 into shape (1,168,1) cannot predict for house: 7566.0
model loaded./checkpoints/7674.0/cp.ckpt
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 311us/step
model loaded./checkpoints/7682.0/cp.ckpt
1/1 [=====] - 0s 994us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 0s/step
model loaded./checkpoints/7739.0/cp.ckpt
```

```
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 967us/step
model loaded./checkpoints/7741.0/cp.ckpt
1/1 [=====] - 0s 995us/step
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/7794.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 966us/step
model loaded./checkpoints/7900.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/7919.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 0s/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/7965.0/cp.ckpt
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/7989.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/8059.0/cp.ckpt
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 995us/step
model loaded./checkpoints/8084.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 5ms/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/8086.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 964us/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/8155.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 991us/step
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/8156.0/cp.ckpt
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 966us/step
model loaded./checkpoints/8244.0/cp.ckpt
cannot reshape array of size 0 into shape (1,168,1) cannot predict for house: 8244.0
model loaded./checkpoints/8386.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 995us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 3ms/step
Found input variables with inconsistent numbers of samples: [672, 449] cannot predict for house: 8386.0
model loaded./checkpoints/8467.0/cp.ckpt
```

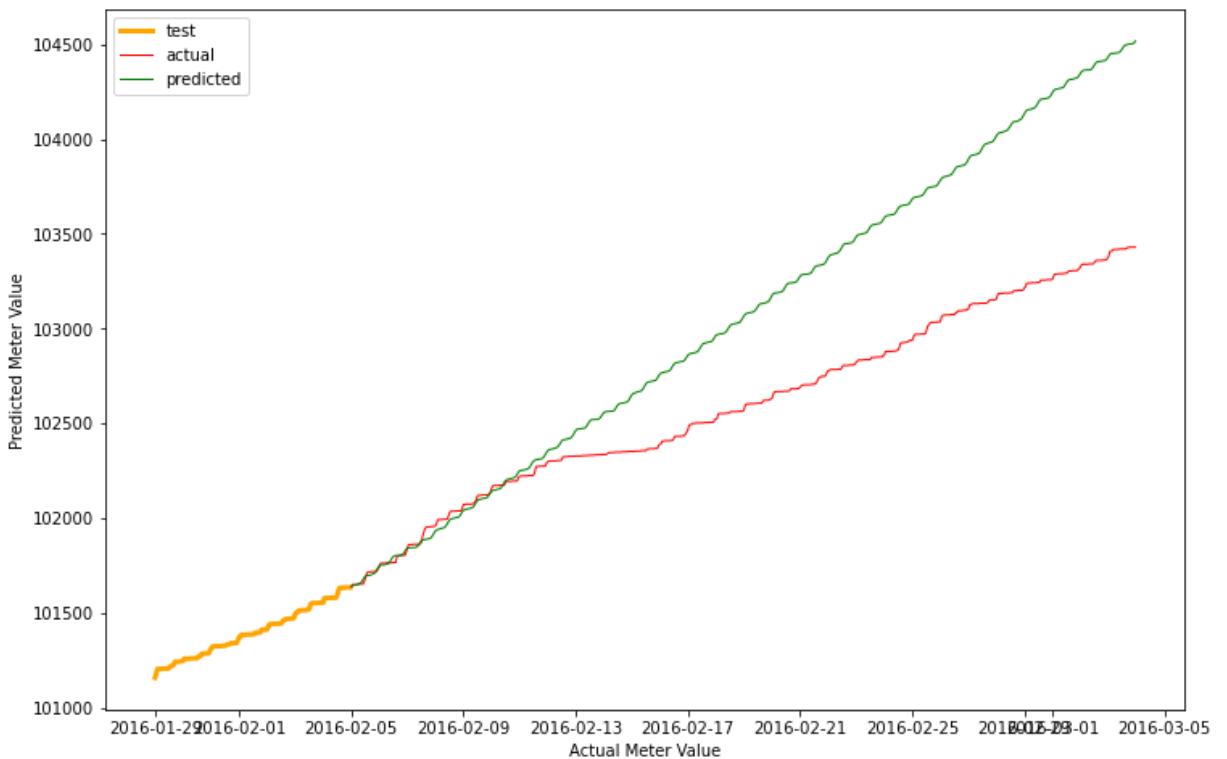
```
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/8703.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
model loaded./checkpoints/8829.0/cp.ckpt
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/8890.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/9052.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/9121.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 995us/step
model loaded./checkpoints/9134.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 3ms/step
model loaded./checkpoints/9160.0/cp.ckpt
cannot reshape array of size 0 into shape (1,168,1) cannot predict for house: 9160.0
model loaded./checkpoints/9278.0/cp.ckpt
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/9295.0/cp.ckpt
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/9474.0/cp.ckpt
1/1 [=====] - 0s 984us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 3ms/step
model loaded./checkpoints/9600.0/cp.ckpt
cannot reshape array of size 0 into shape (1,168,1) cannot predict for house: 9600.0
model loaded./checkpoints/9620.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 1000us/step
model loaded./checkpoints/9631.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 964us/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/9639.0/cp.ckpt
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 957us/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/9729.0/cp.ckpt
```

```

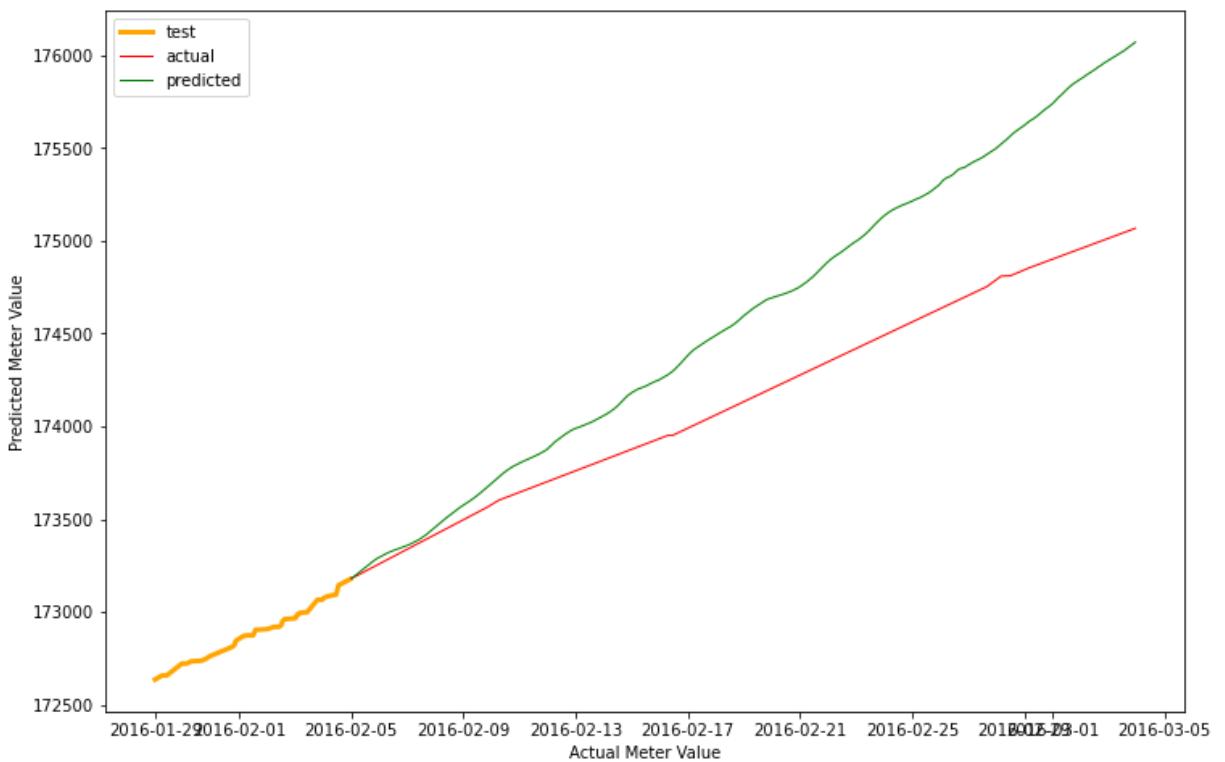
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 998us/step
model loaded./checkpoints/9766.0/cp.ckpt
1/1 [=====] - 0s 1ms/step
1/1 [=====] - 0s 992us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 1ms/step
model loaded./checkpoints/9849.0/cp.ckpt
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 2ms/step
model loaded./checkpoints/9956.0/cp.ckpt
1/1 [=====] - 0s 3ms/step
1/1 [=====] - 0s 998us/step
1/1 [=====] - 0s 999us/step
1/1 [=====] - 0s 1ms/step
Found input variables with inconsistent numbers of samples: [672, 95] cannot predict
for house: 9956.0
model loaded./checkpoints/9982.0/cp.ckpt
1/1 [=====] - 0s 997us/step
1/1 [=====] - 0s 996us/step
1/1 [=====] - 0s 2ms/step
1/1 [=====] - 0s 2ms/step

```

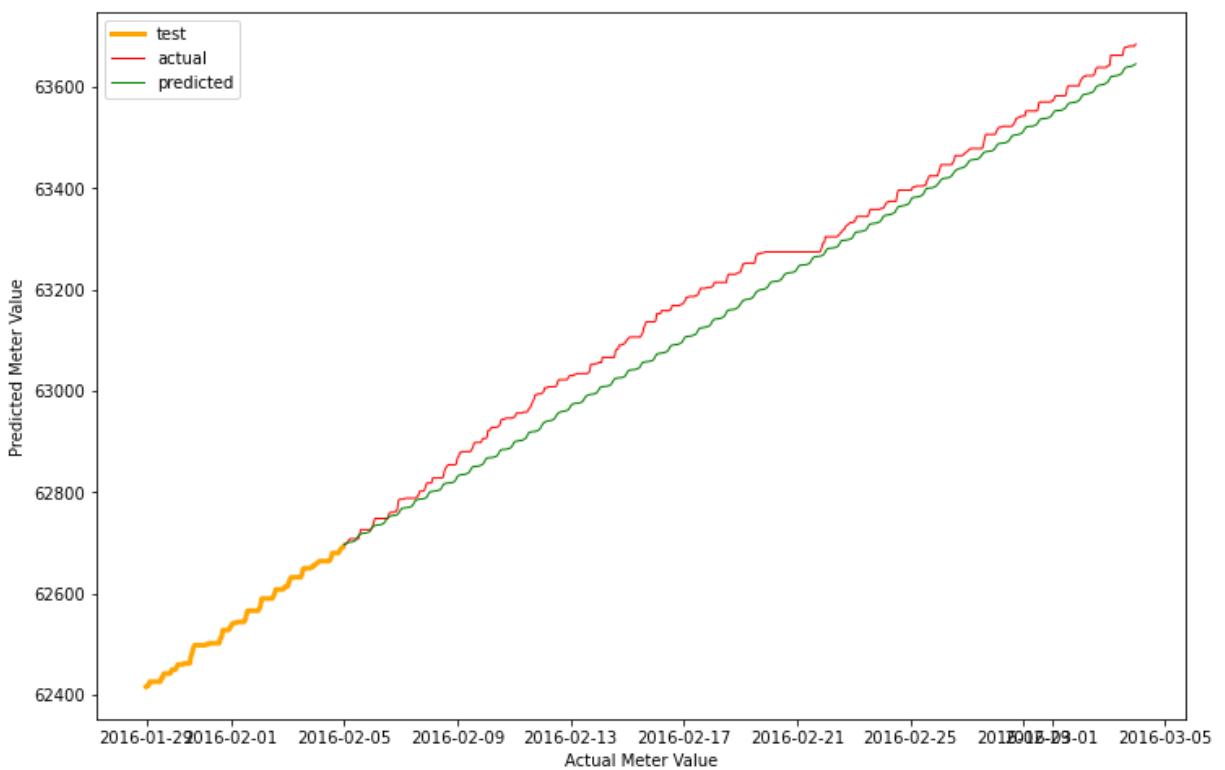
Household 35.0  
Testing MSE: 343684.3636423747



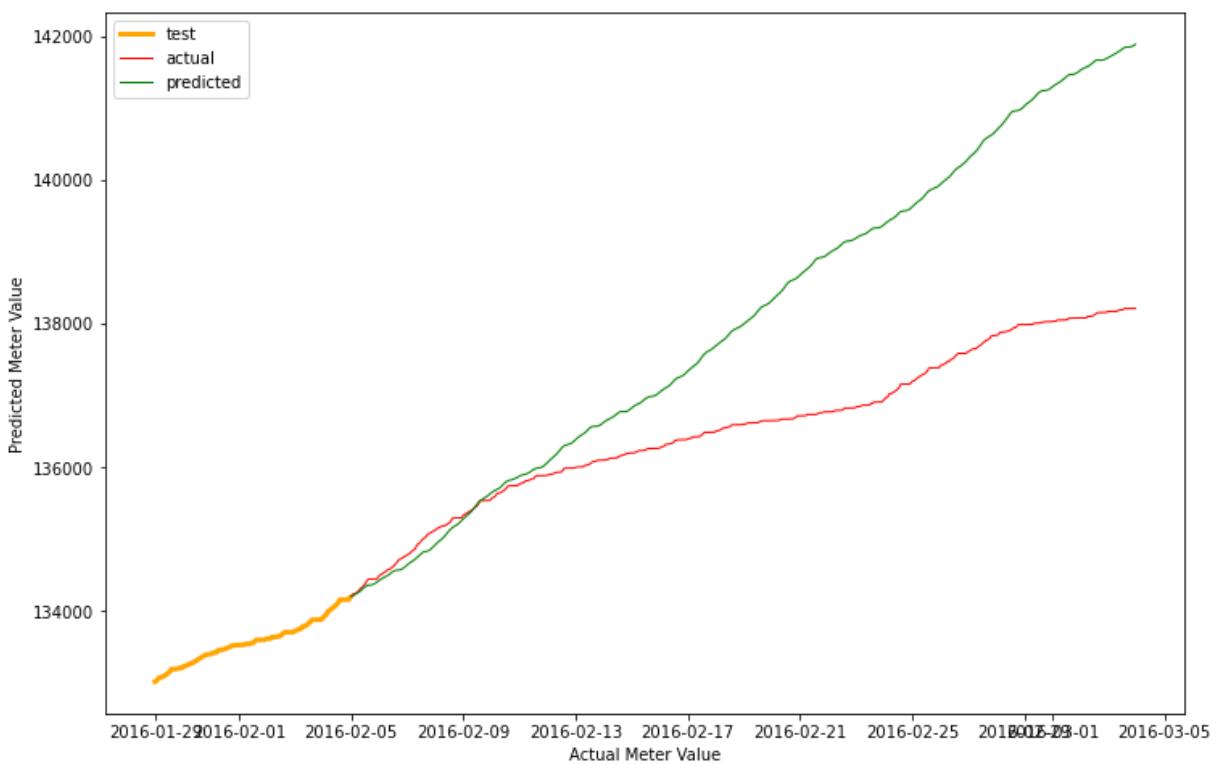
Household 44.0  
Testing MSE: 280844.9015897075



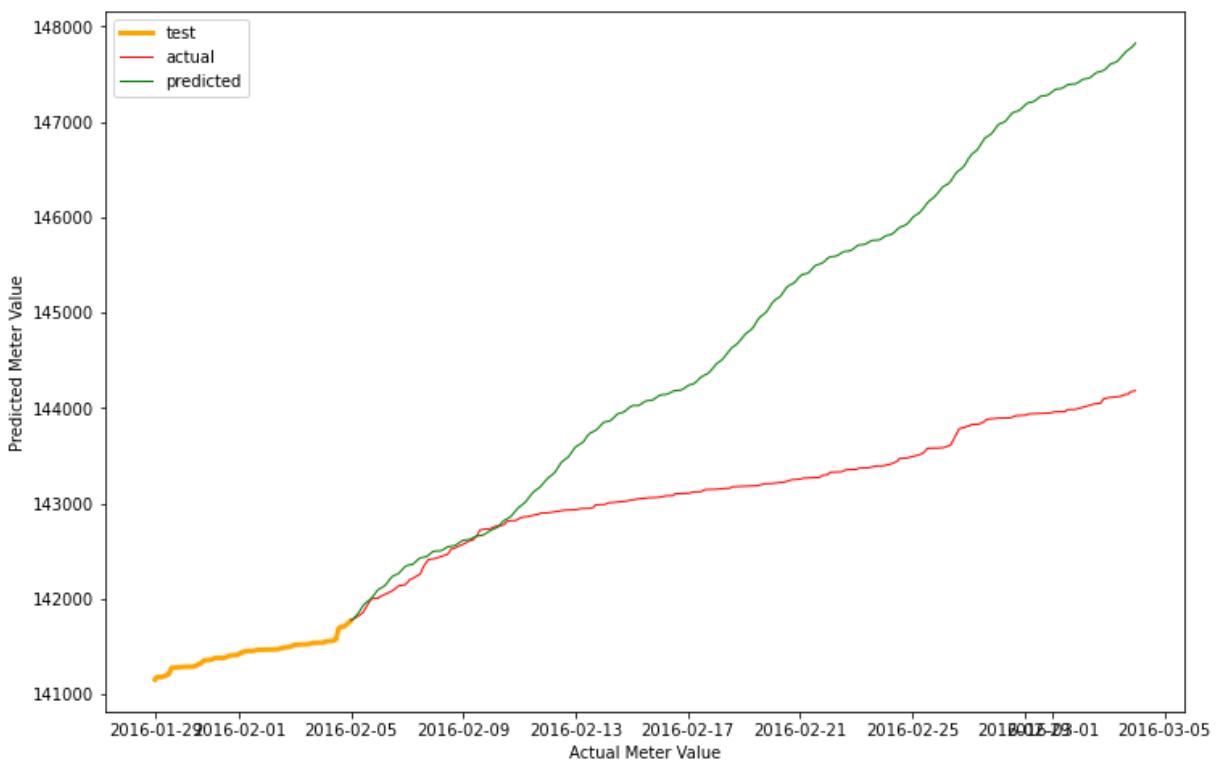
Household 77.0  
Testing MSE: 2084.9461636038677



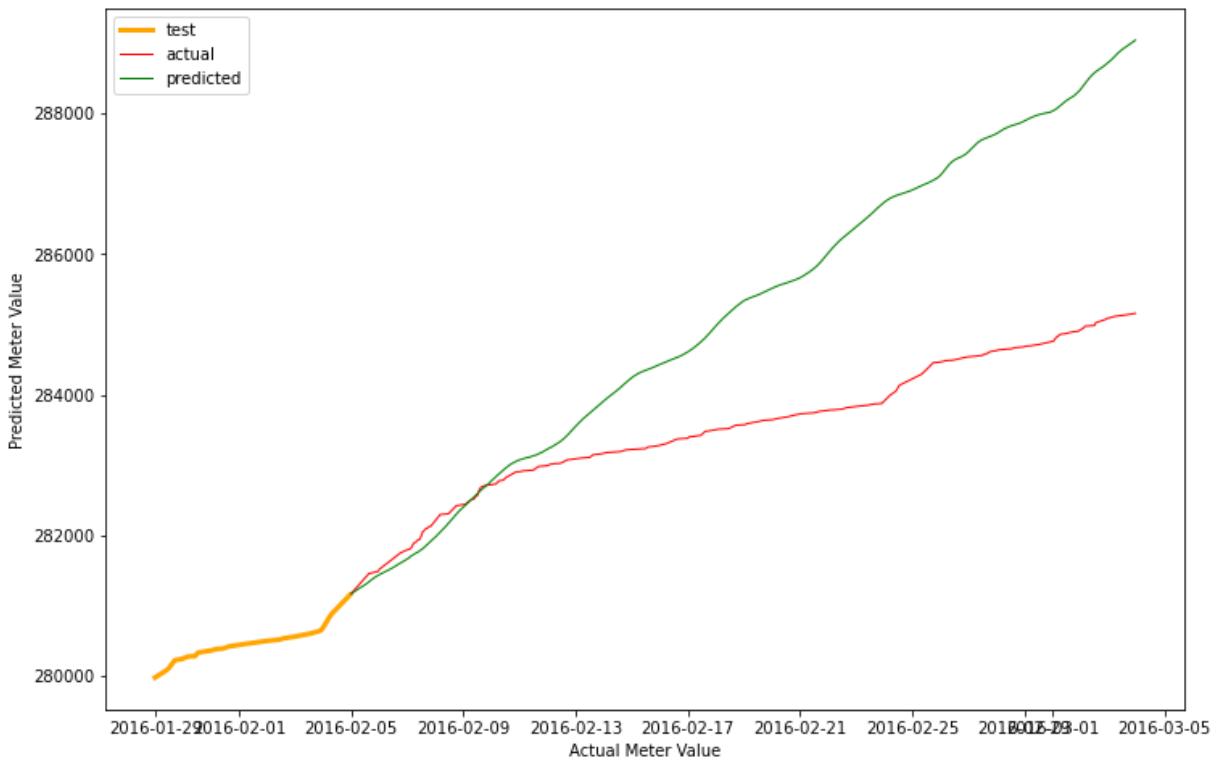
Household 94.0  
Testing MSE: 3851415.9569153558



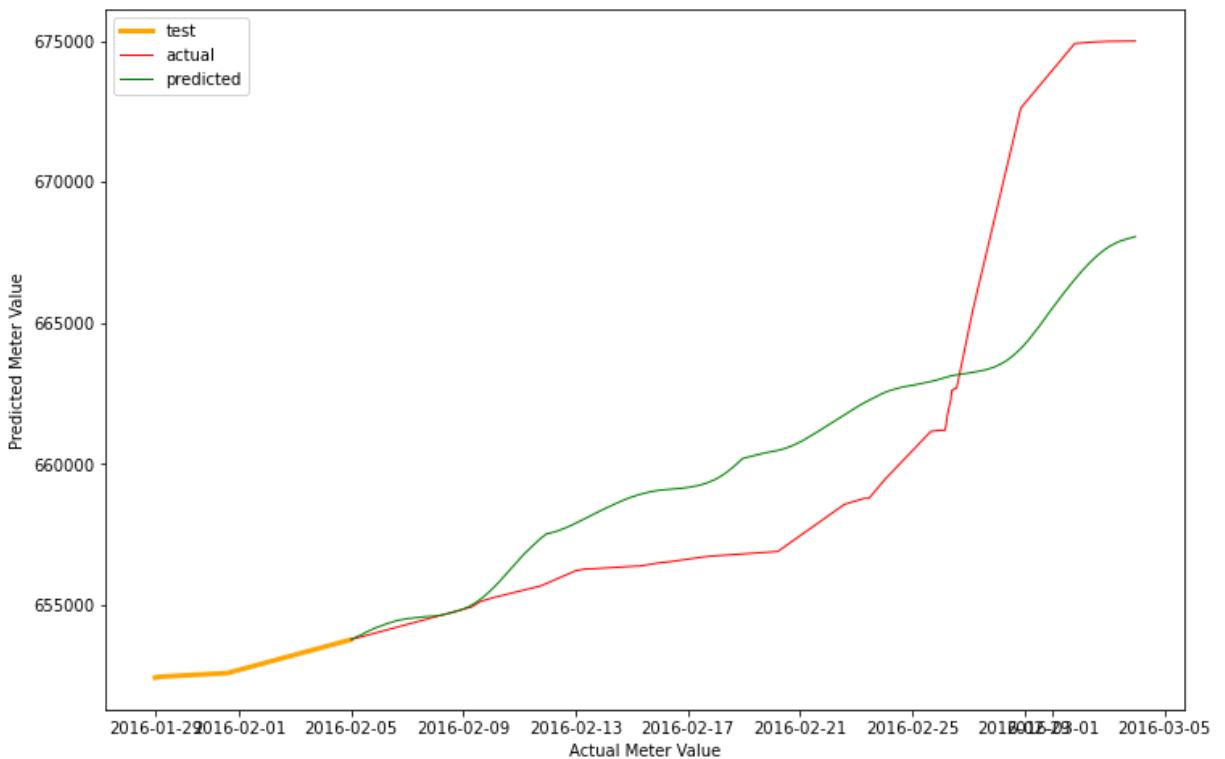
Household 114.0  
Testing MSE: 4125055.6063366523



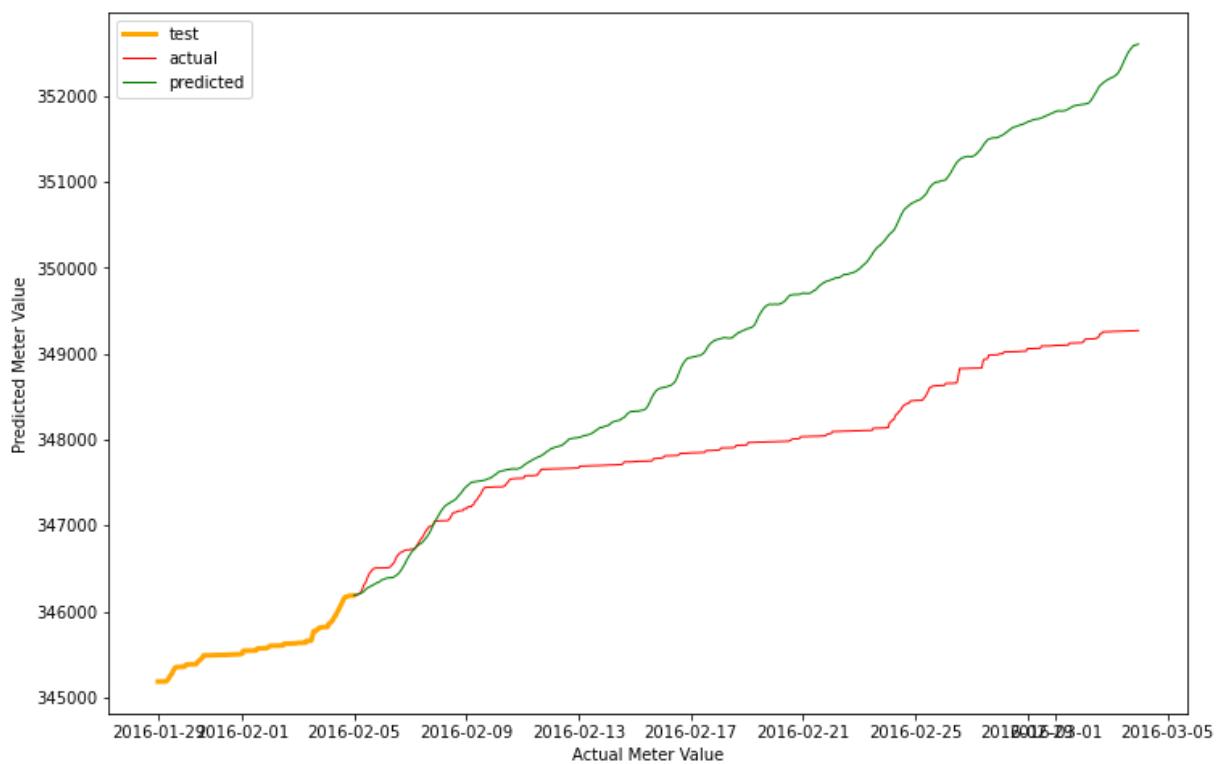
Household 187.0  
Testing MSE: 4382640.512212513



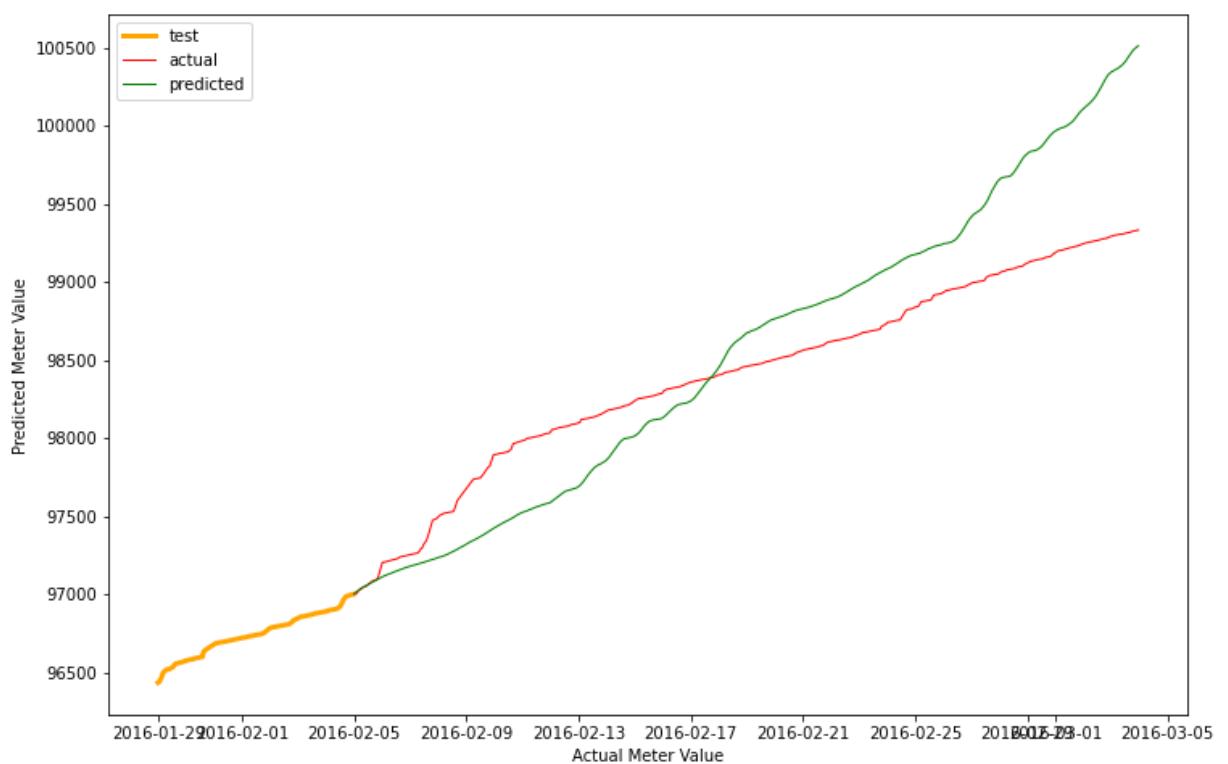
Household 222.0  
Testing MSE: 15372787.14556137



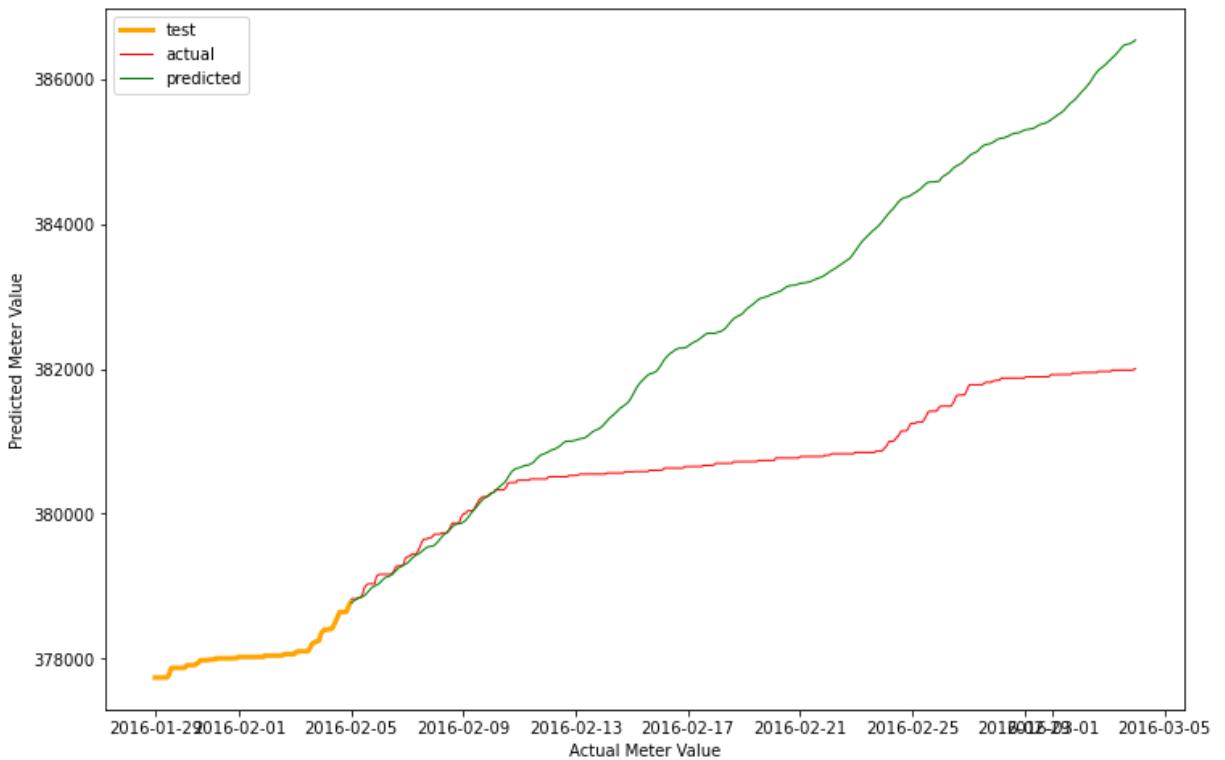
Household 252.0  
Testing MSE: 2958151.18147625



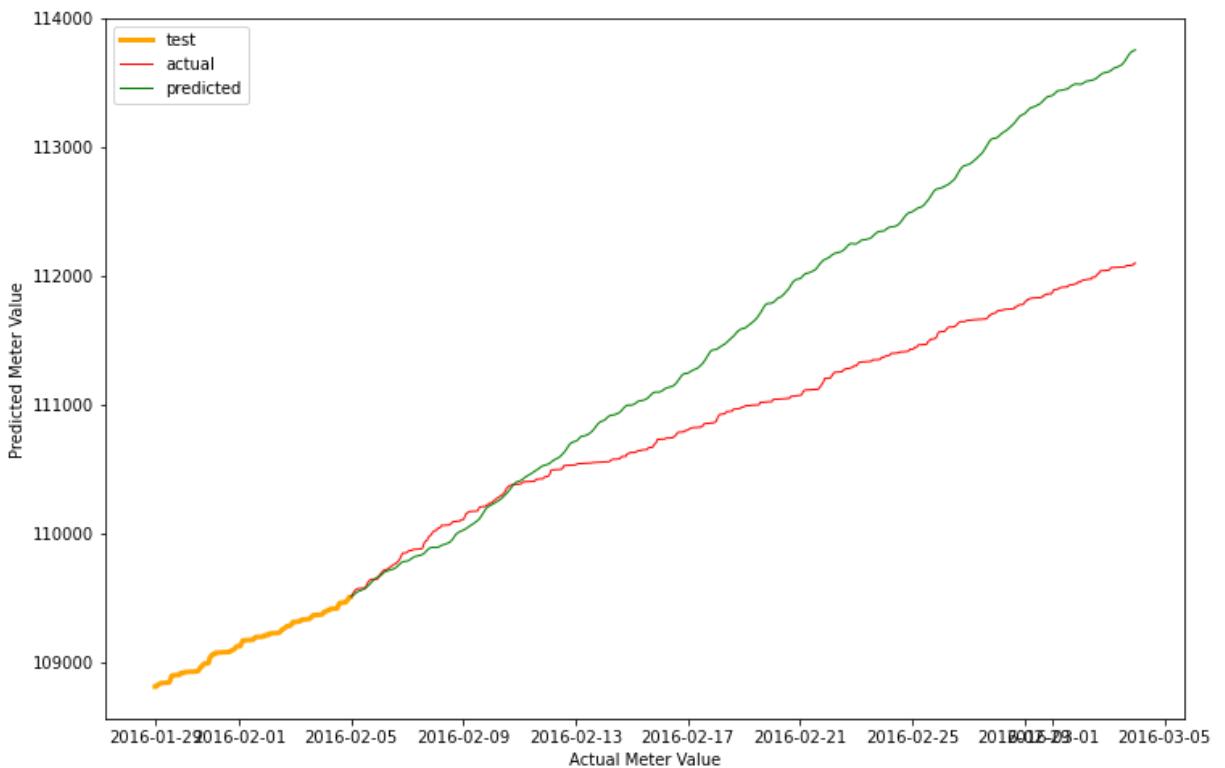
Household 370.0  
Testing MSE: 208497.5675130174



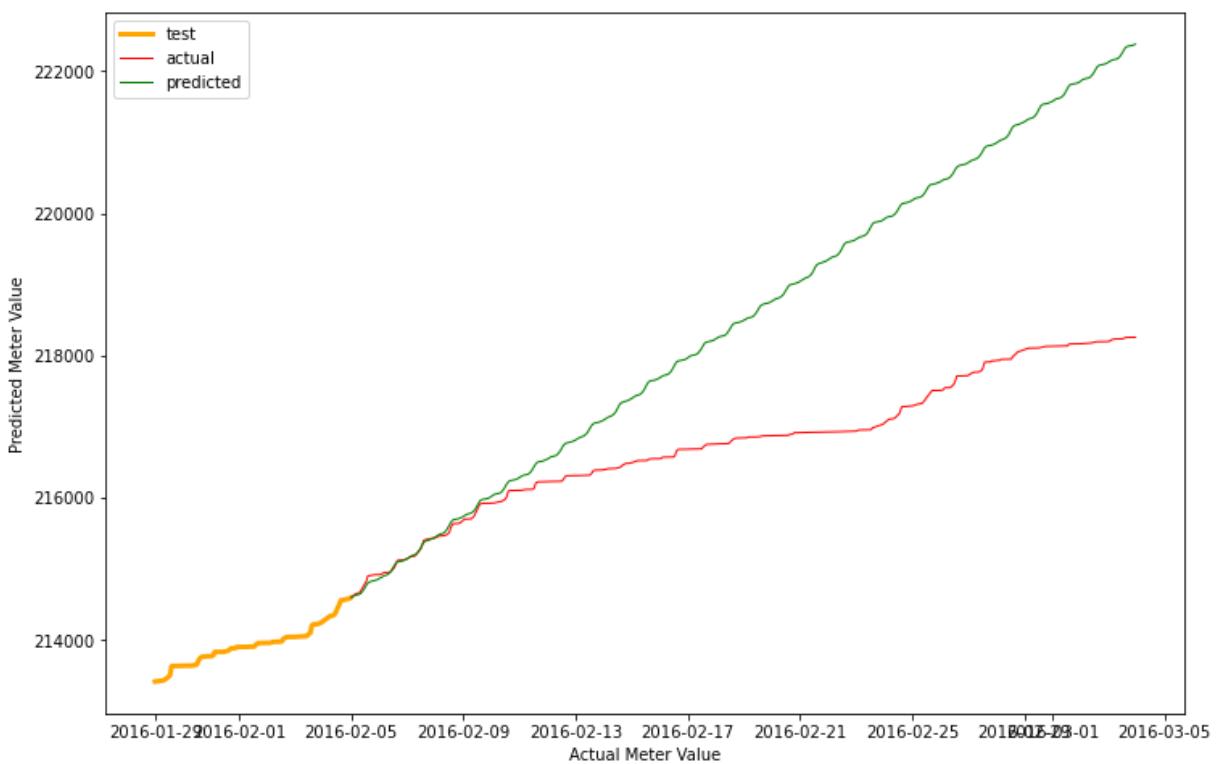
Household 483.0  
Testing MSE: 5667468.8848237535



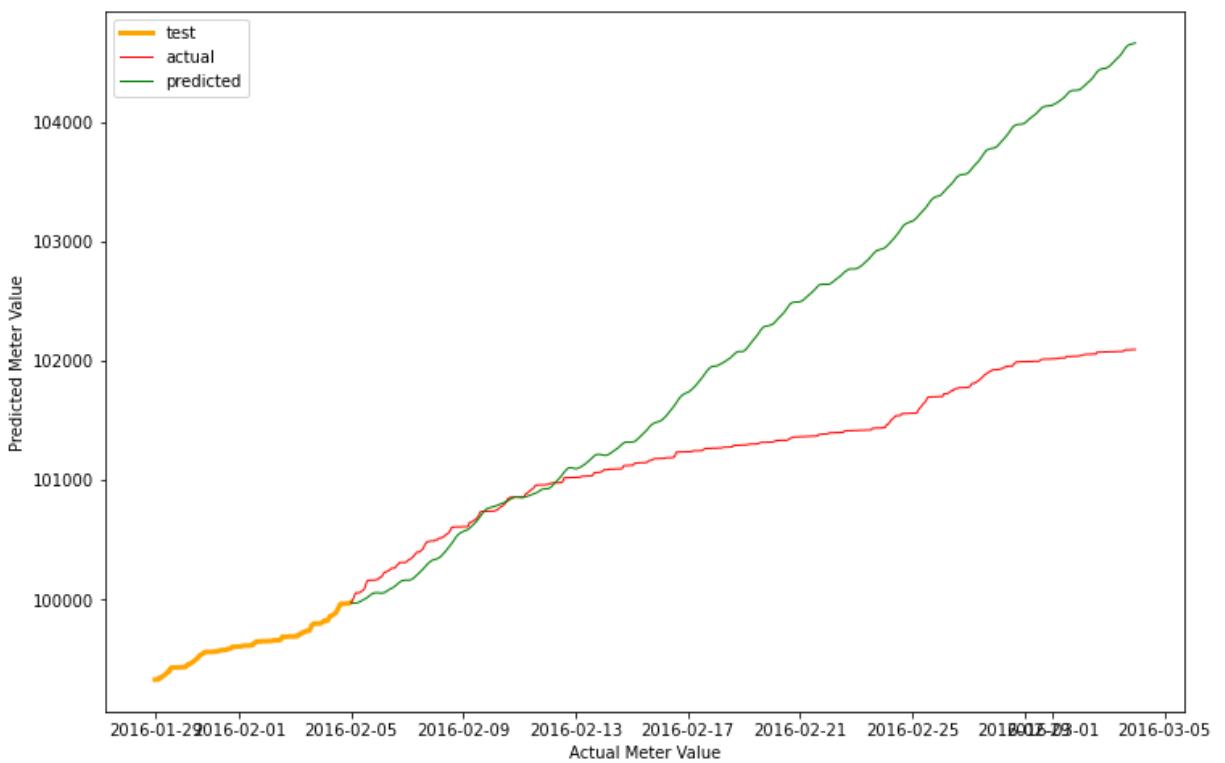
Household 484.0  
Testing MSE: 773345.2384964171



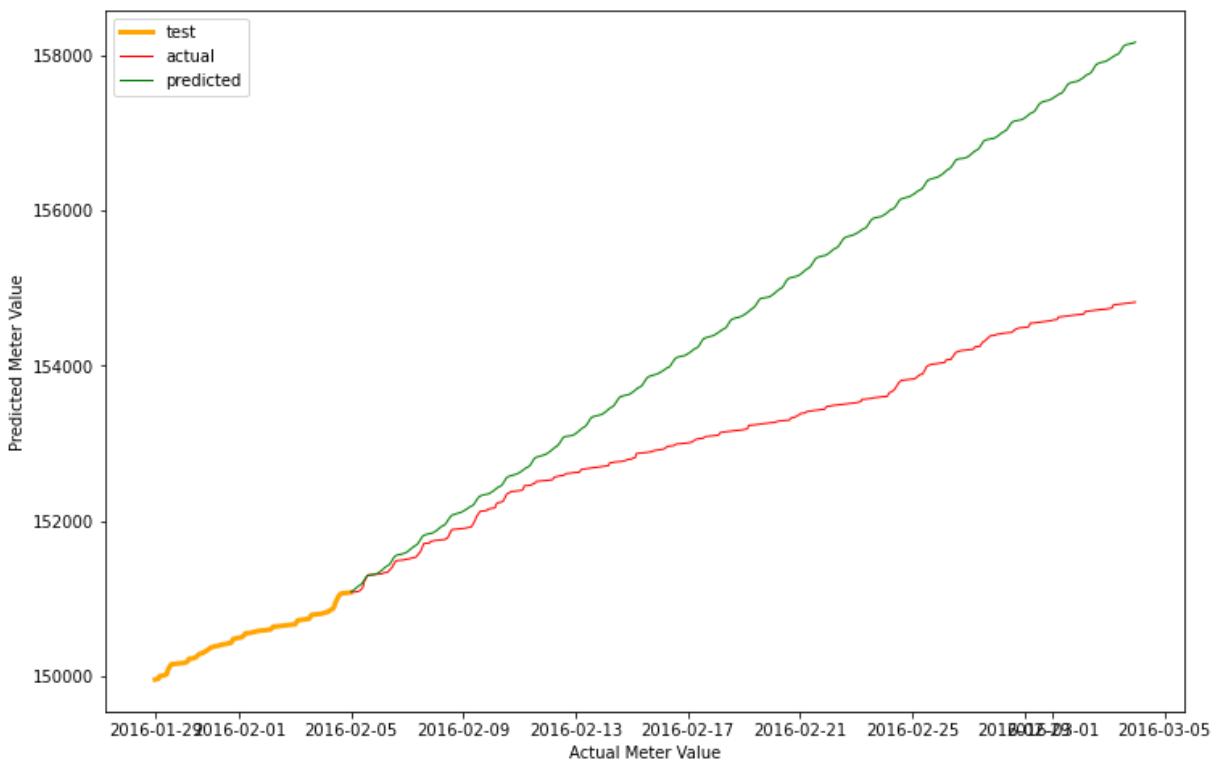
Household 661.0  
Testing MSE: 4785231.01421047



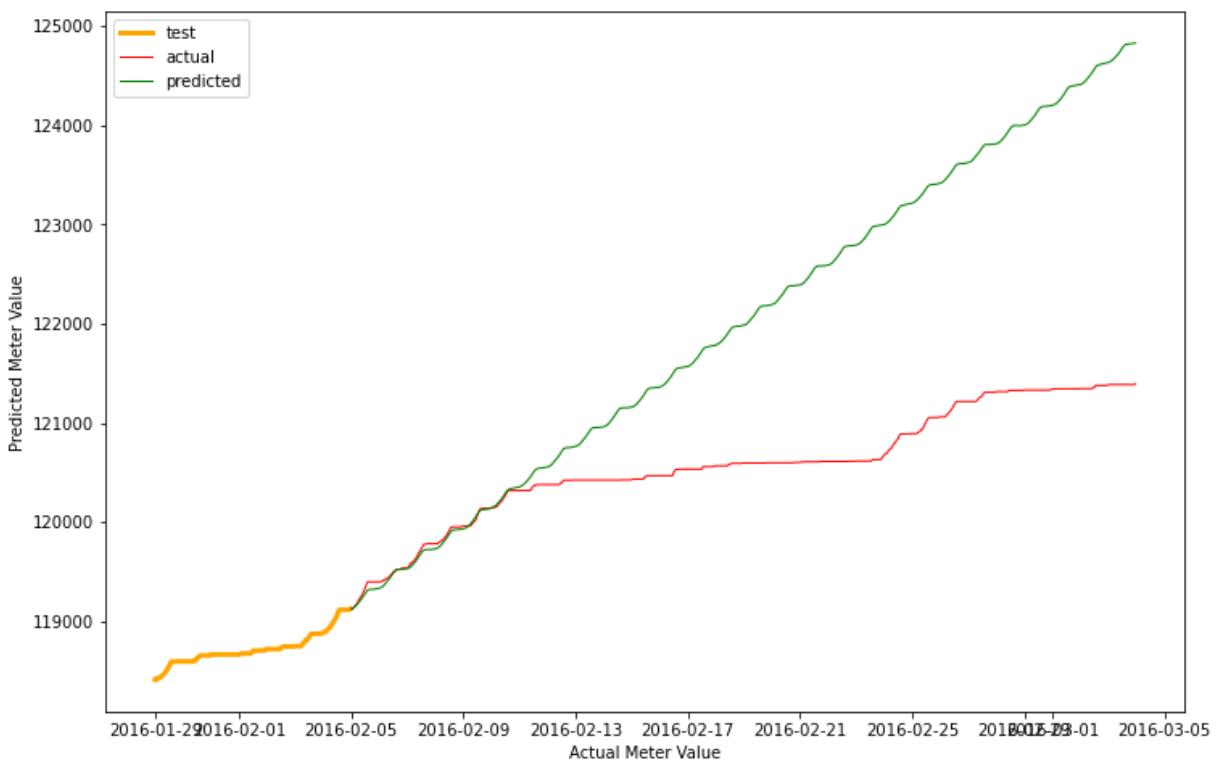
Household 739.0  
Testing MSE: 1589182.3459304627



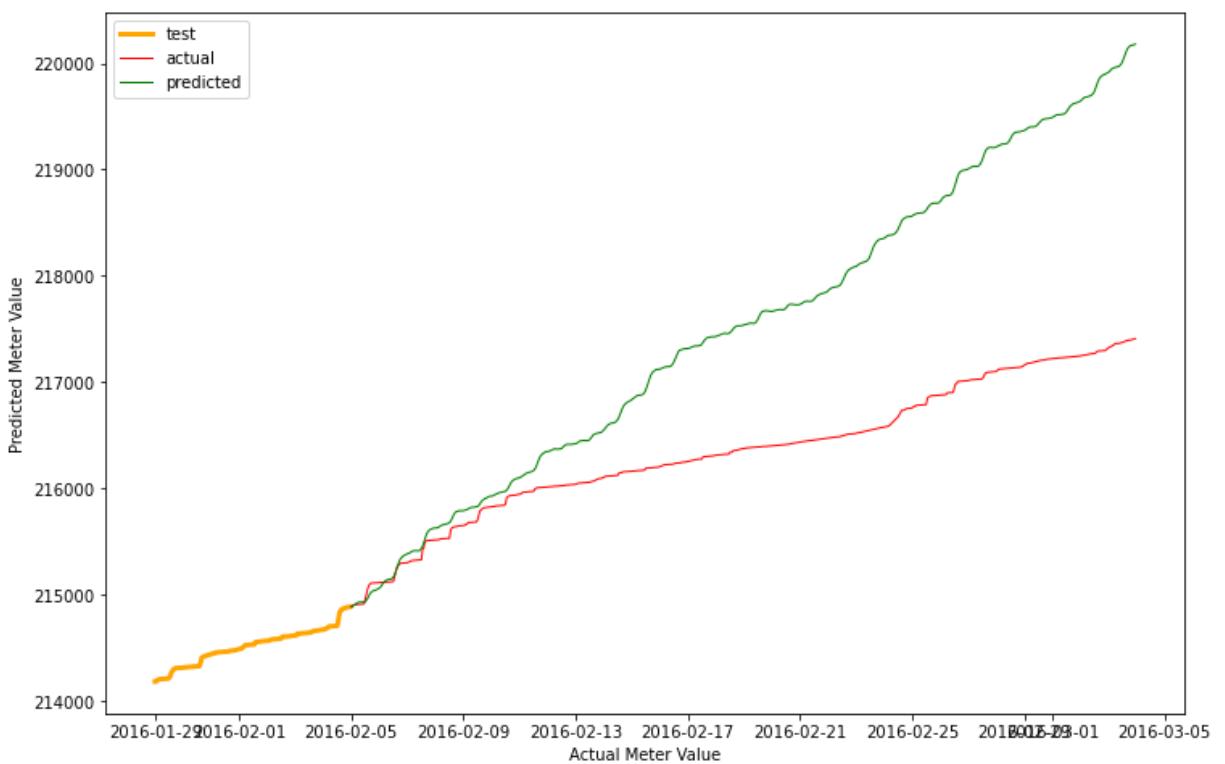
Household 744.0  
Testing MSE: 3322923.582923075



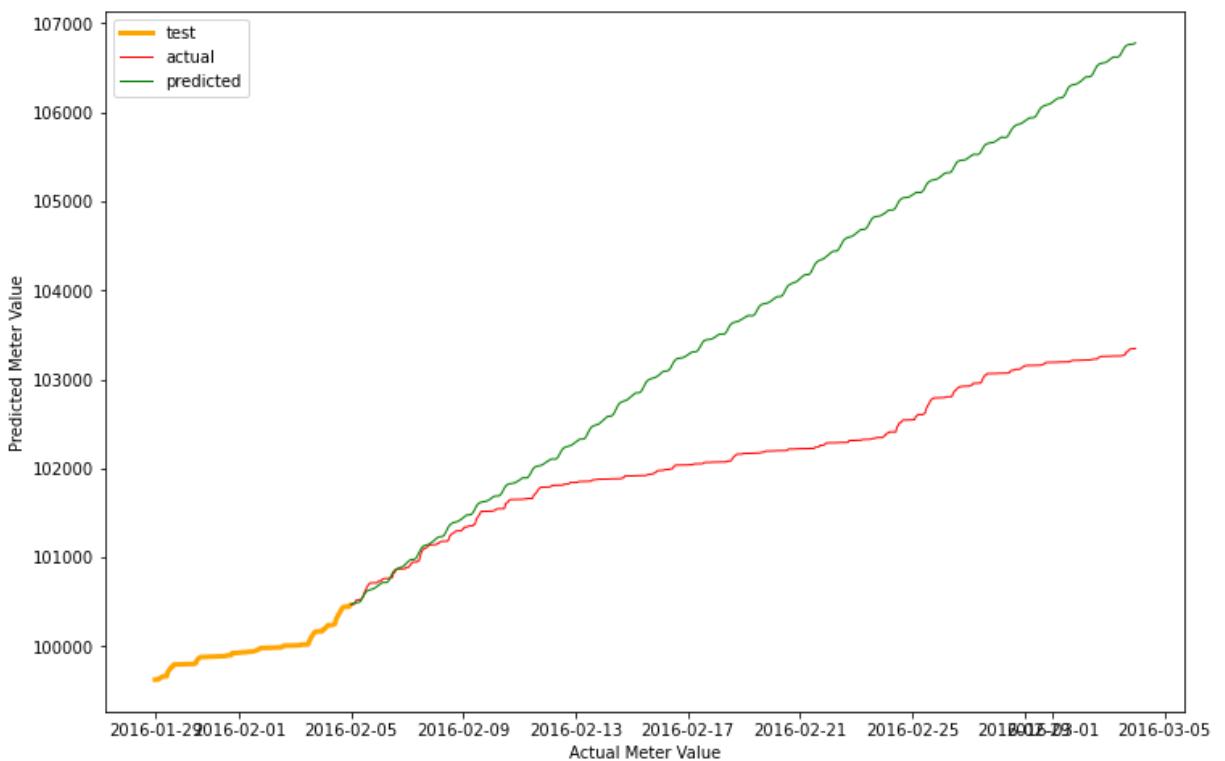
Household 871.0  
Testing MSE: 3273817.131345204



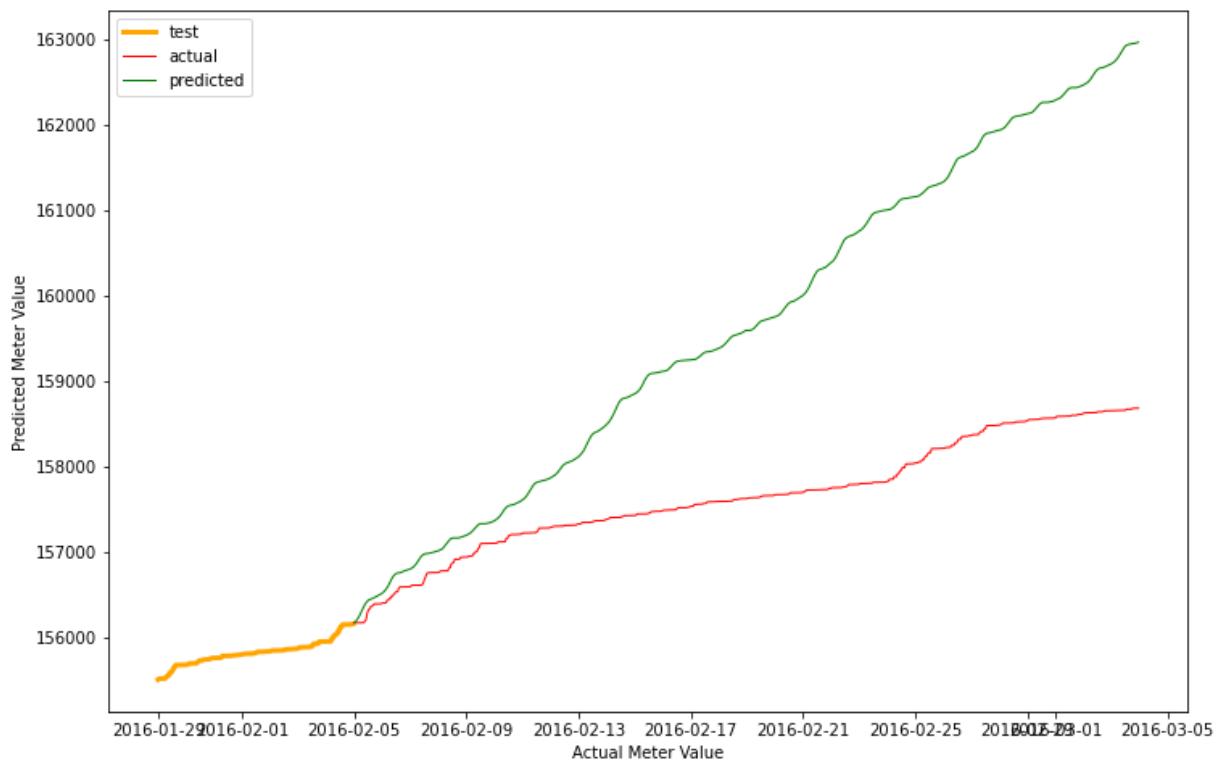
Household 1042.0  
Testing MSE: 2048088.8642474643



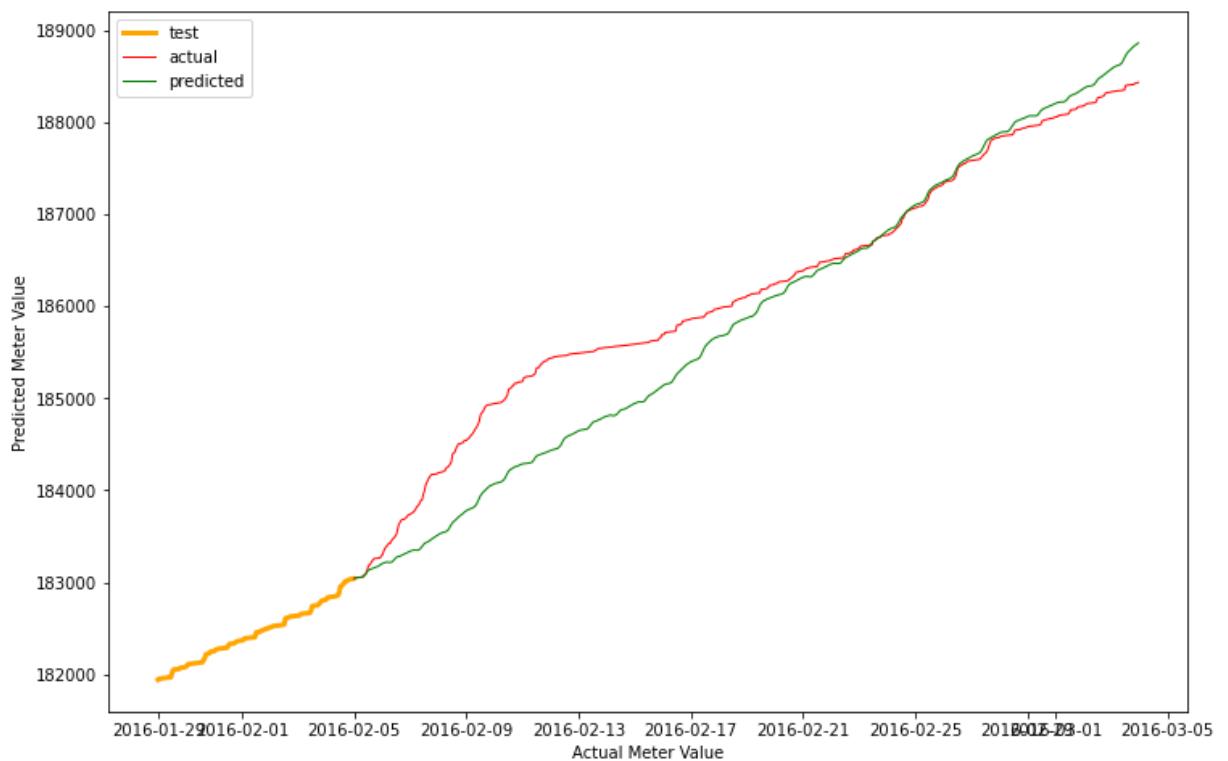
Household 1086.0  
Testing MSE: 3579169.0382626853



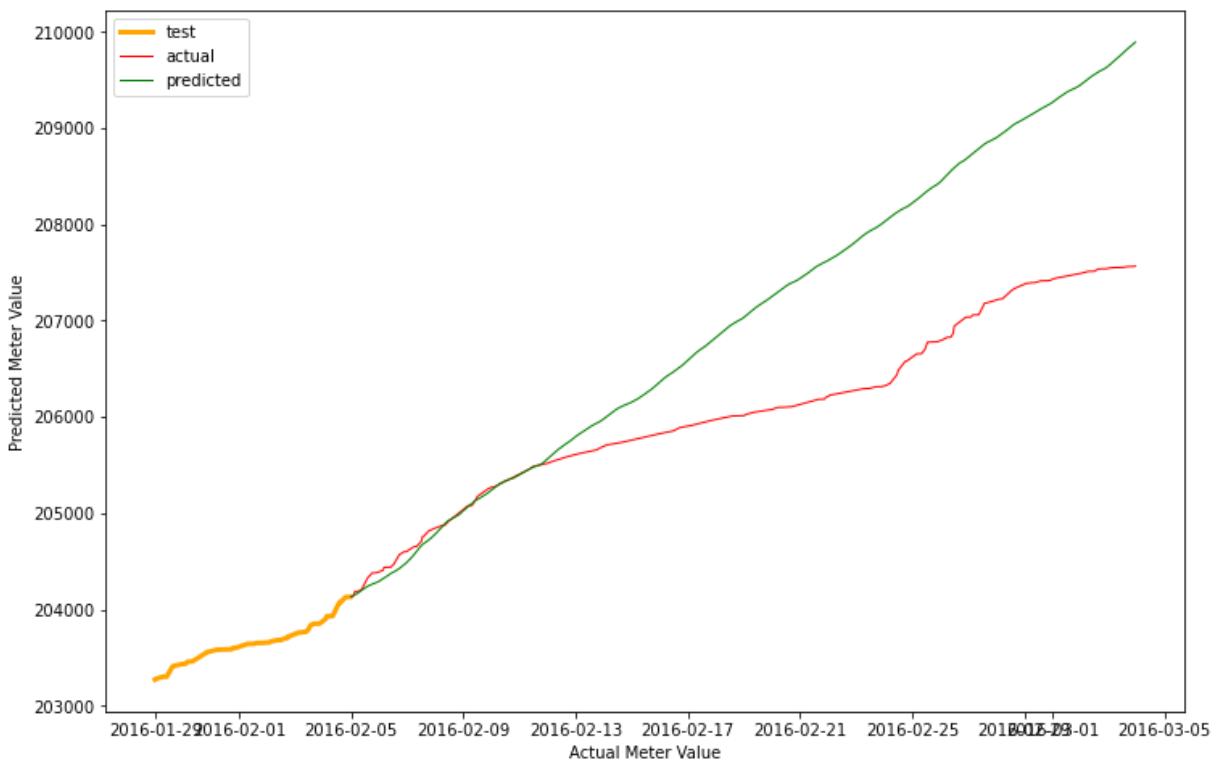
Household 1185.0  
Testing MSE: 5873575.816976274



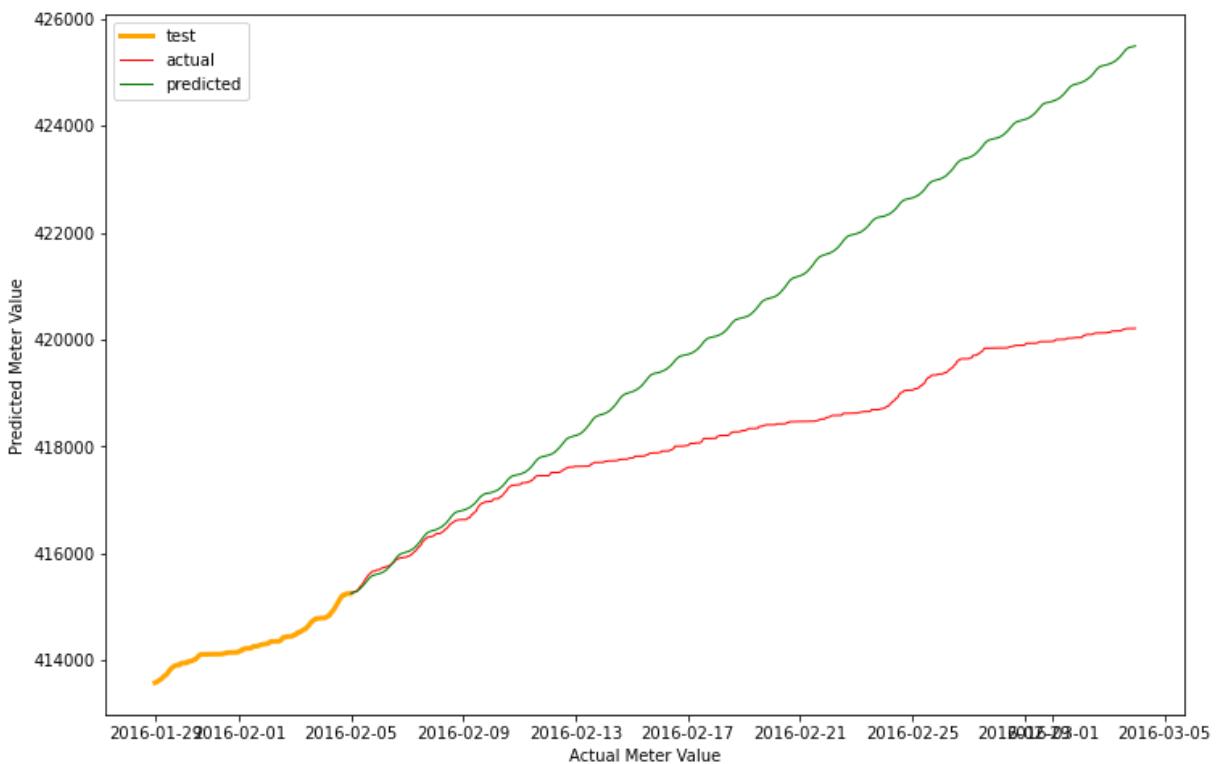
Household 1283.0  
Testing MSE: 230831.33363822653



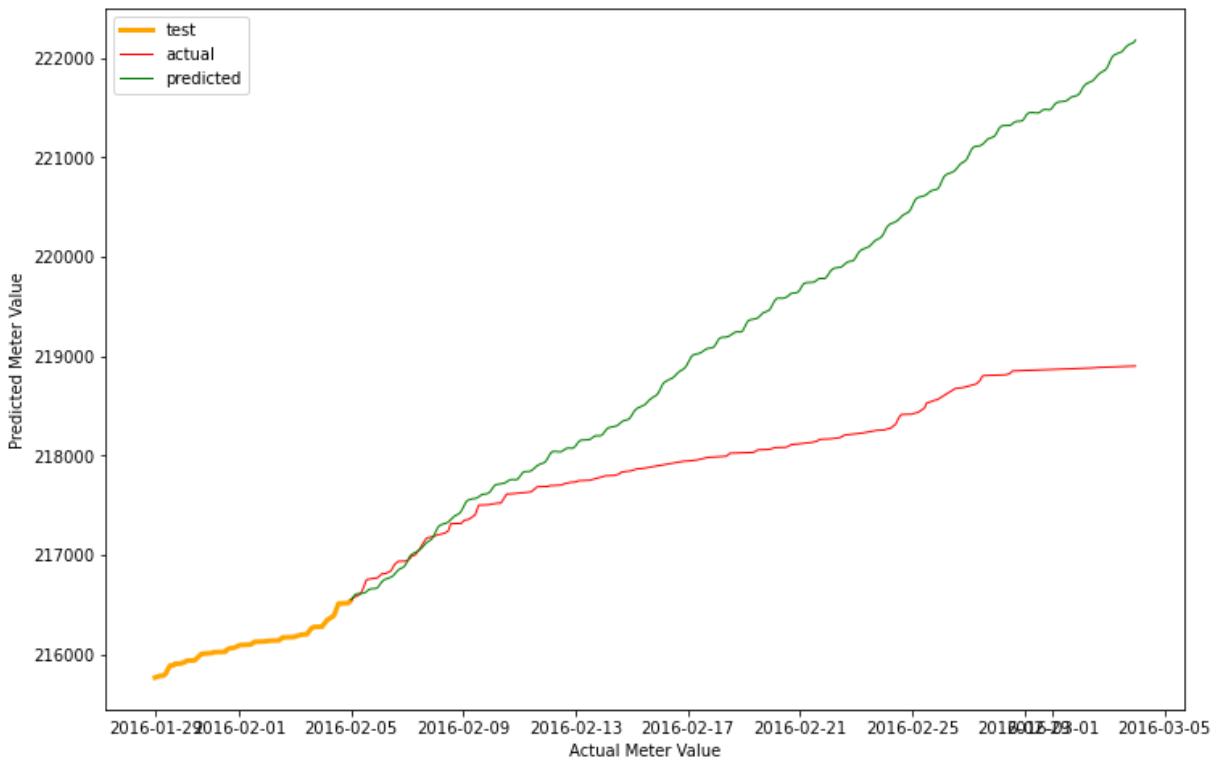
Household 1415.0  
Testing MSE: 1478312.2619468293



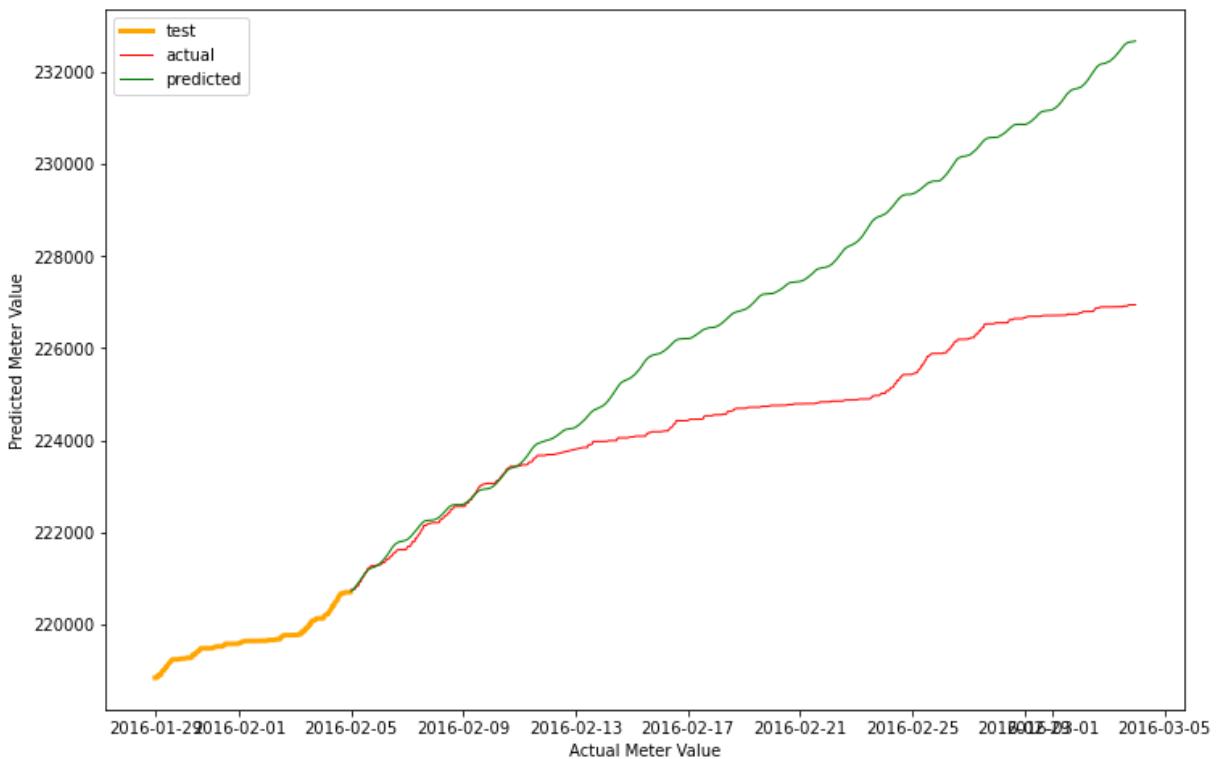
Household 1507.0  
Testing MSE: 7781991.382316953



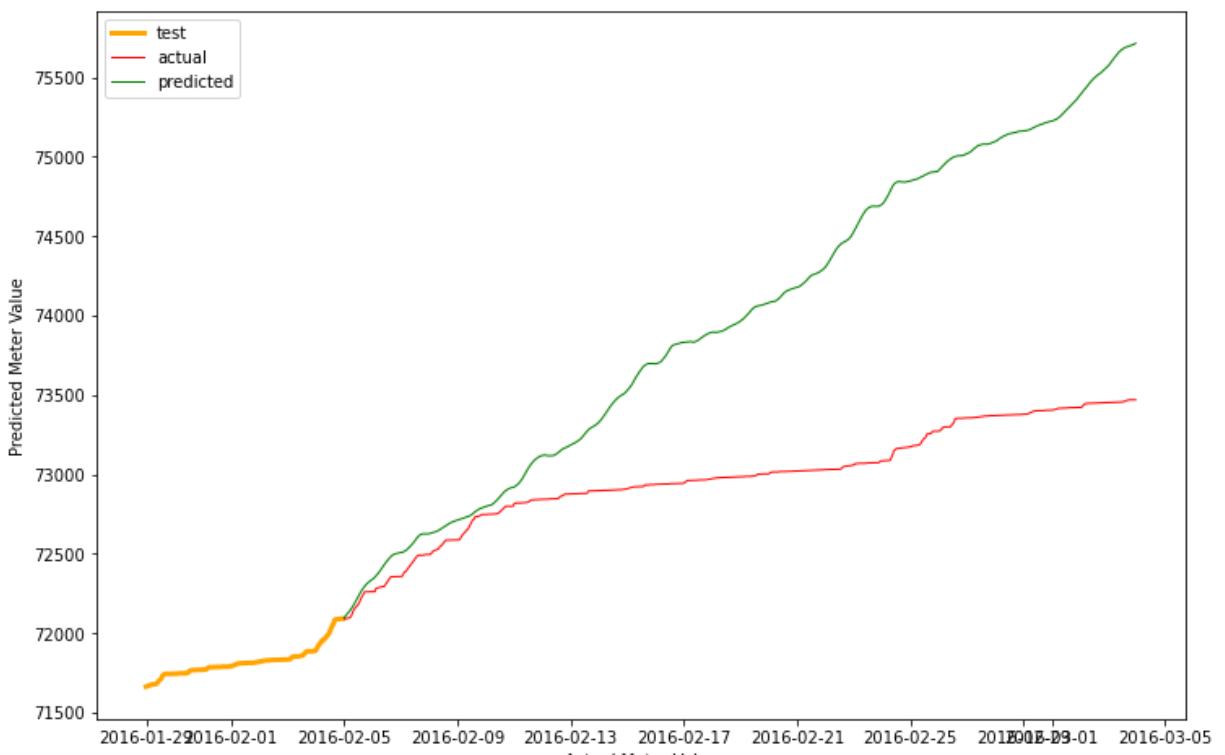
Household 1556.0  
Testing MSE: 2701485.3484587753



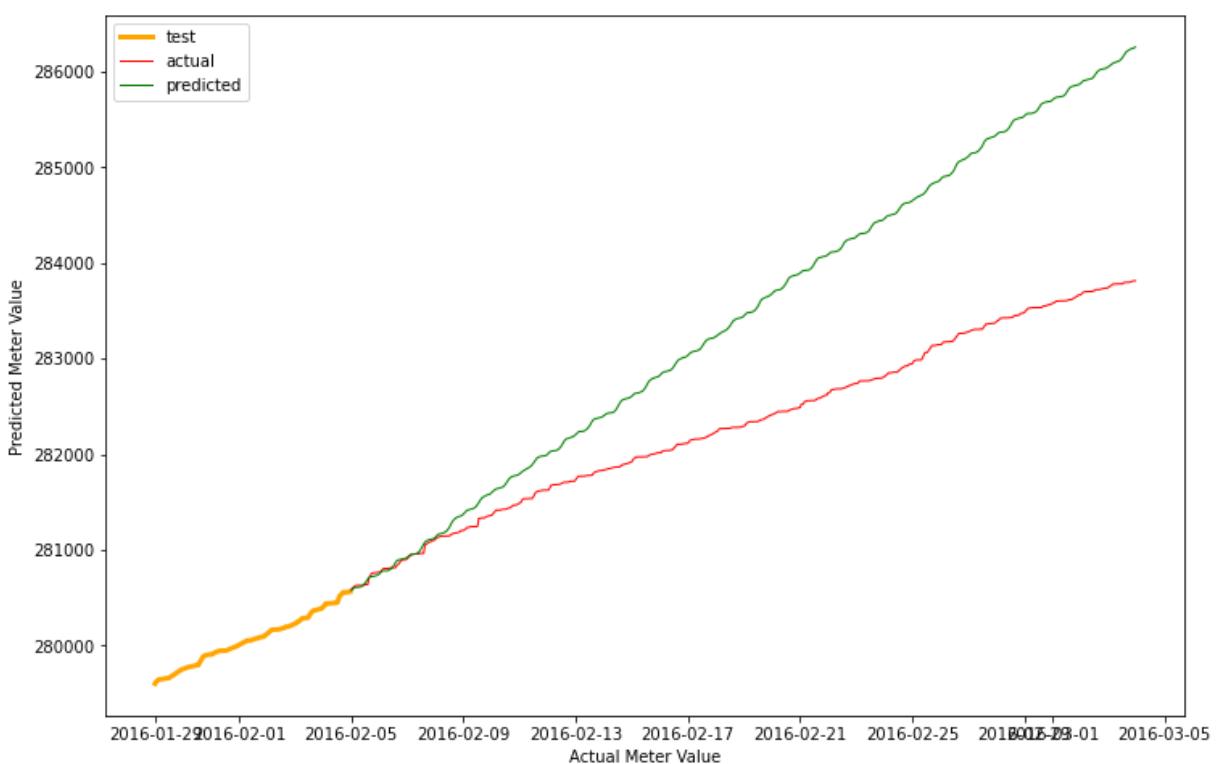
Household 1589.0  
Testing MSE: 8438557.609950766



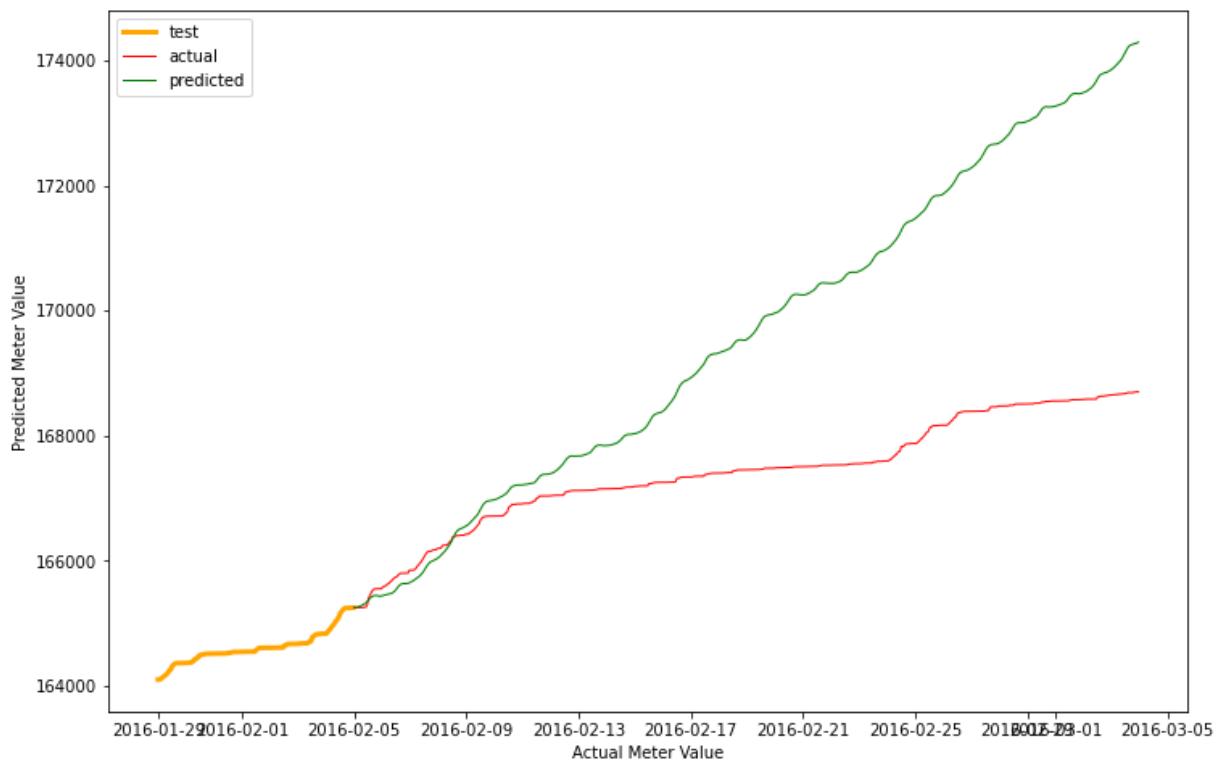
Household 1619.0  
Testing MSE: 1490522.721288326



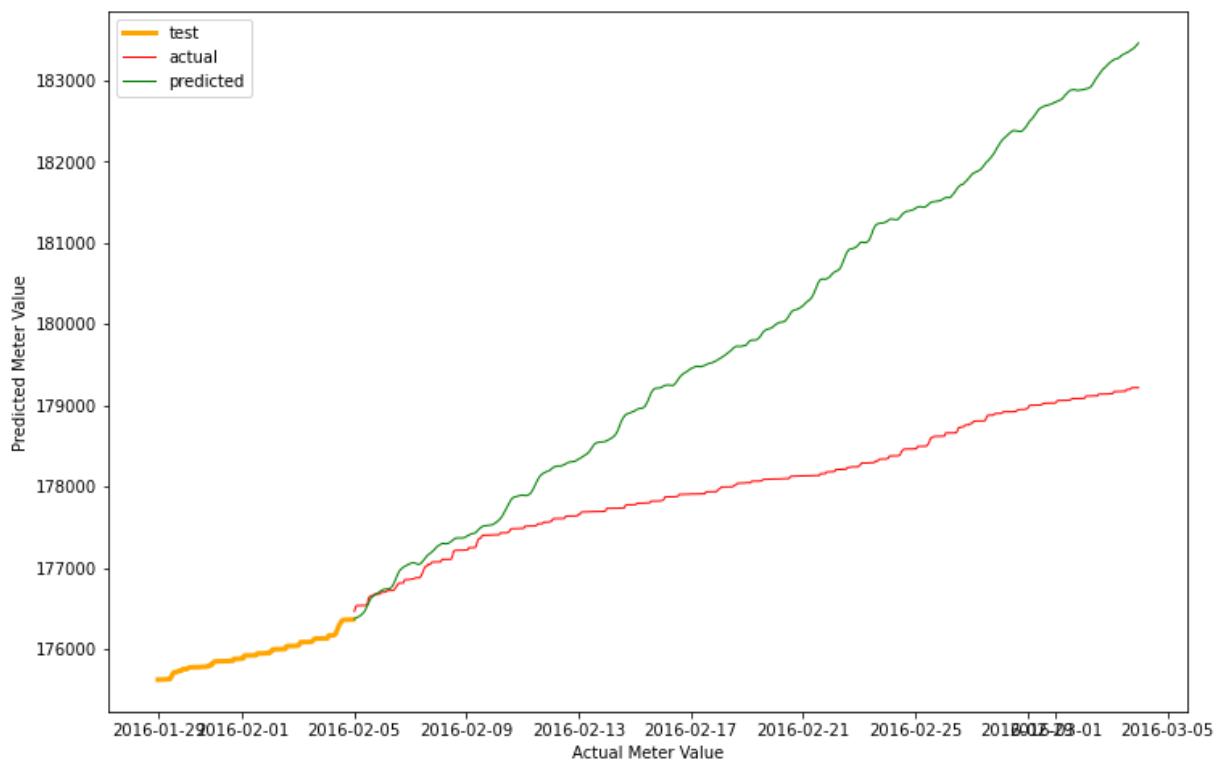
Household 1697.0  
Testing MSE: 1808339.4130042777



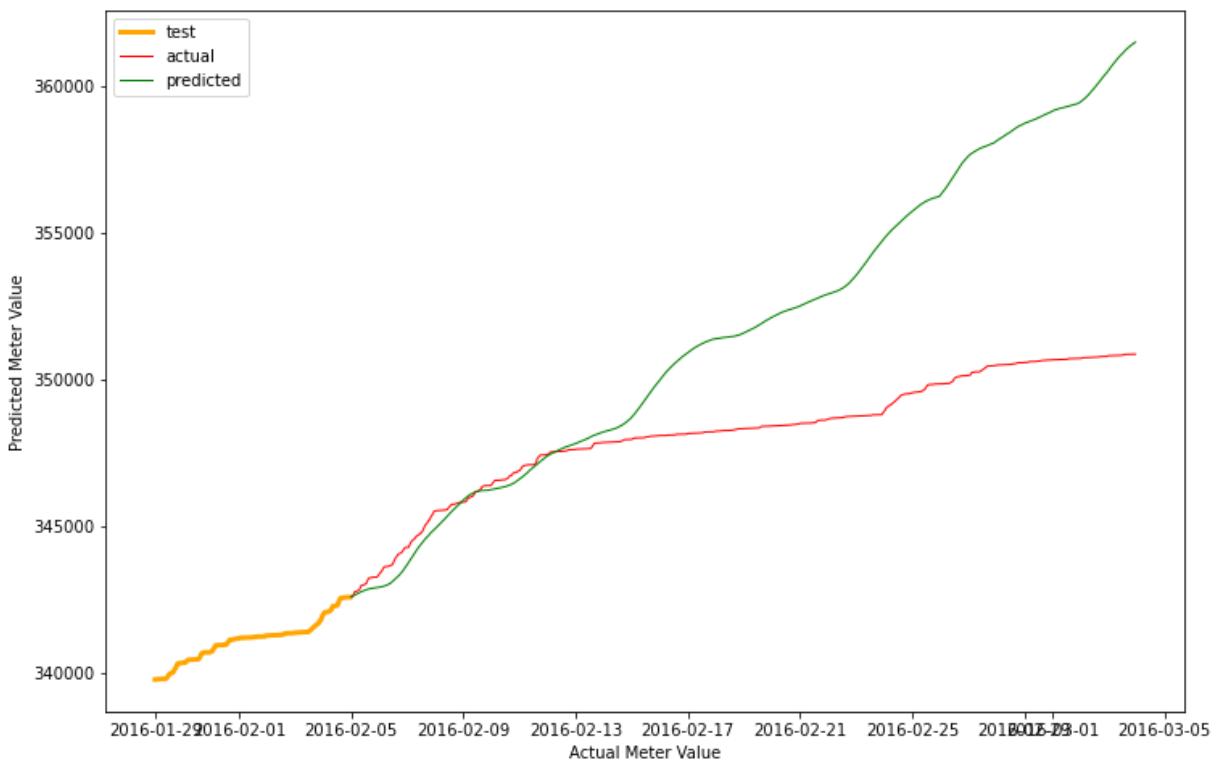
Household 1714.0  
Testing MSE: 8120902.223580496



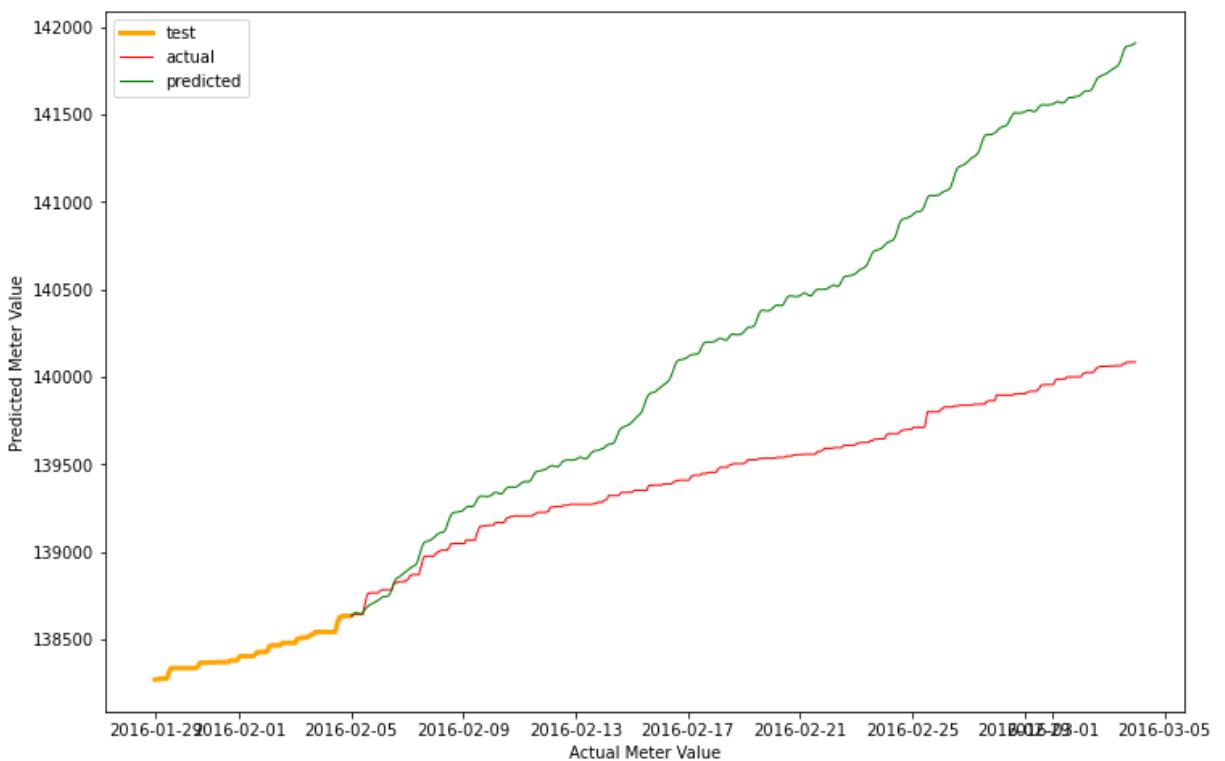
Household 1718.0  
Testing MSE: 5202489.462025234



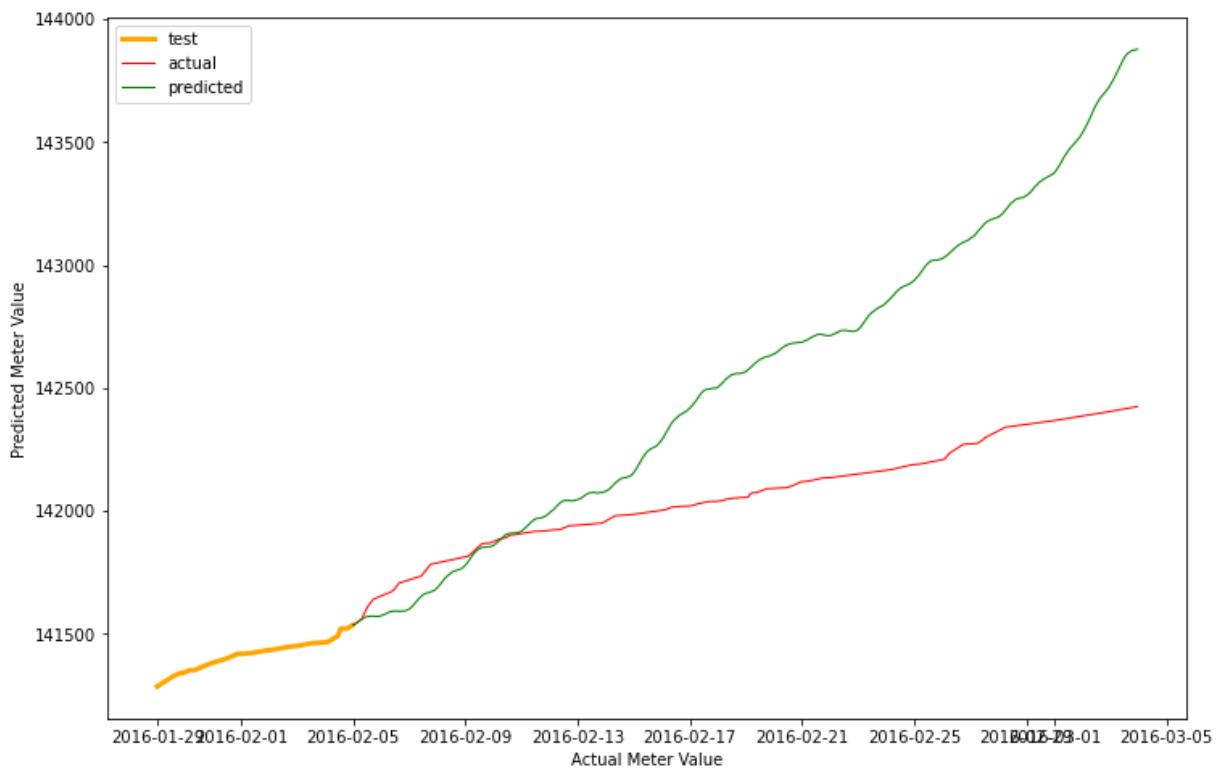
Household 1790.0  
Testing MSE: 24901097.963914014



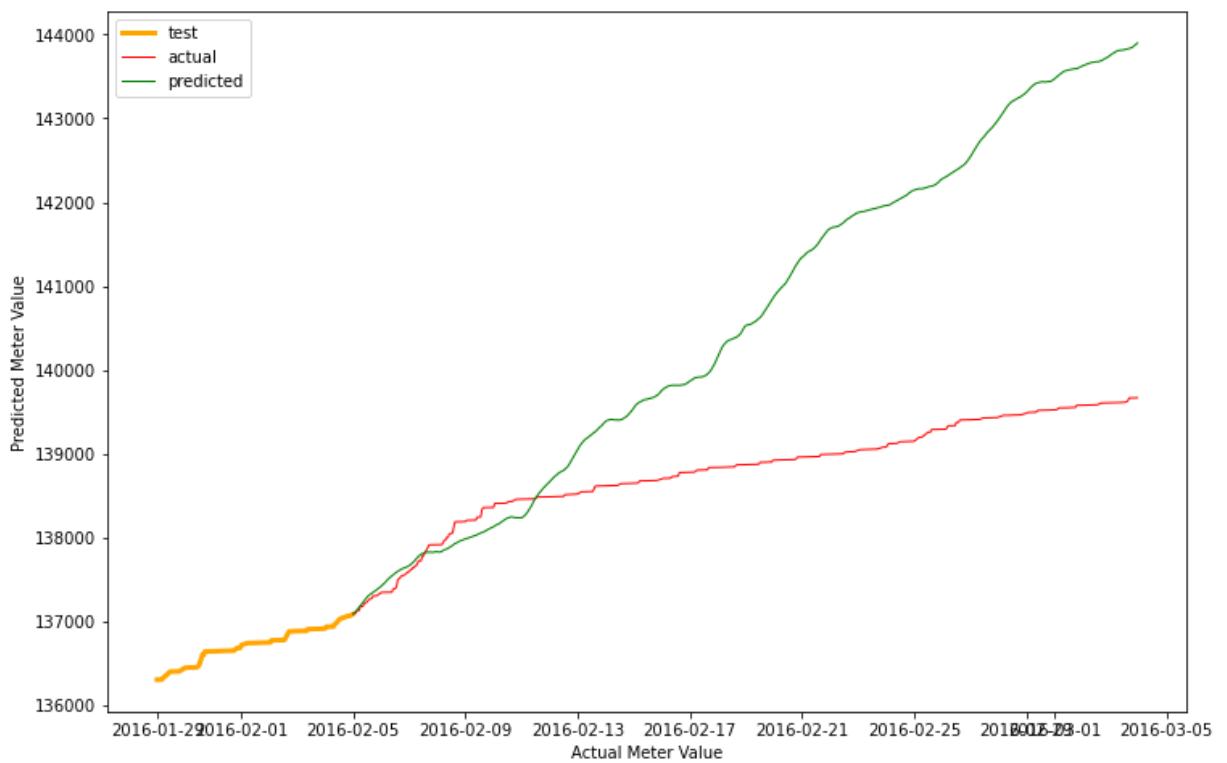
Household 1791.0  
Testing MSE: 944391.2801003432



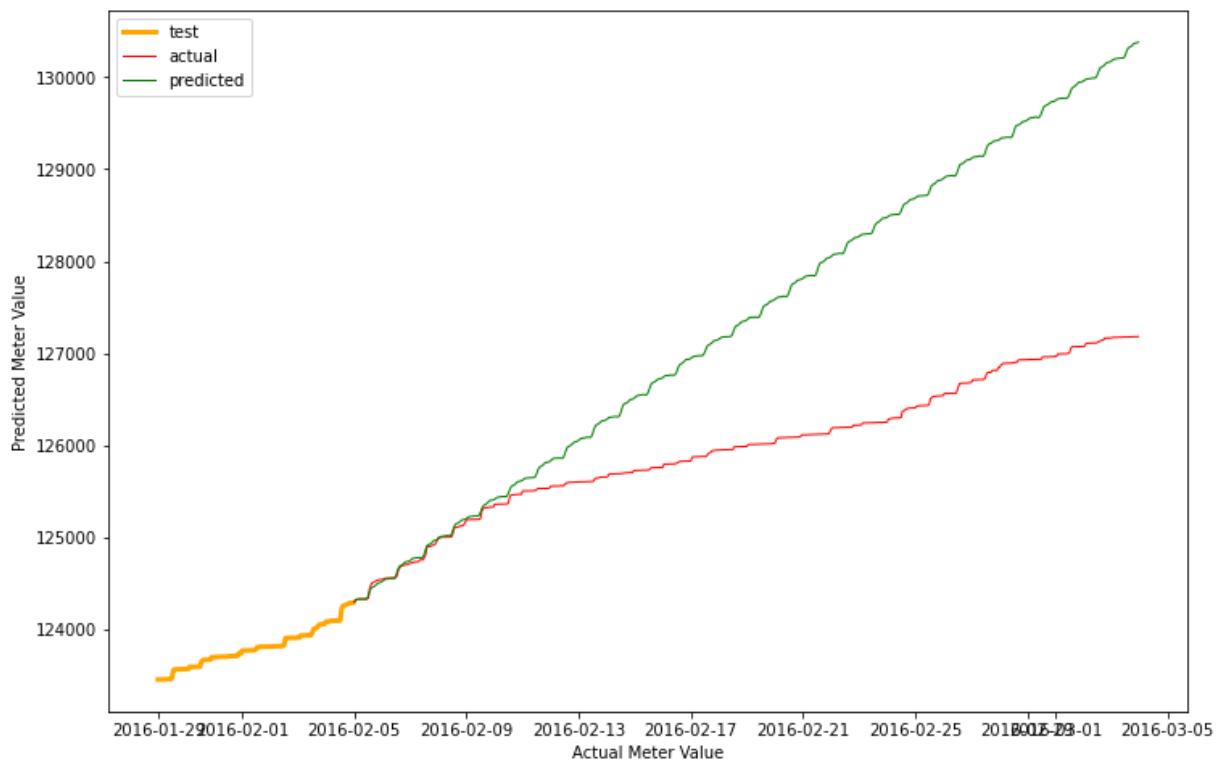
Household 1792.0  
Testing MSE: 408915.3139036455



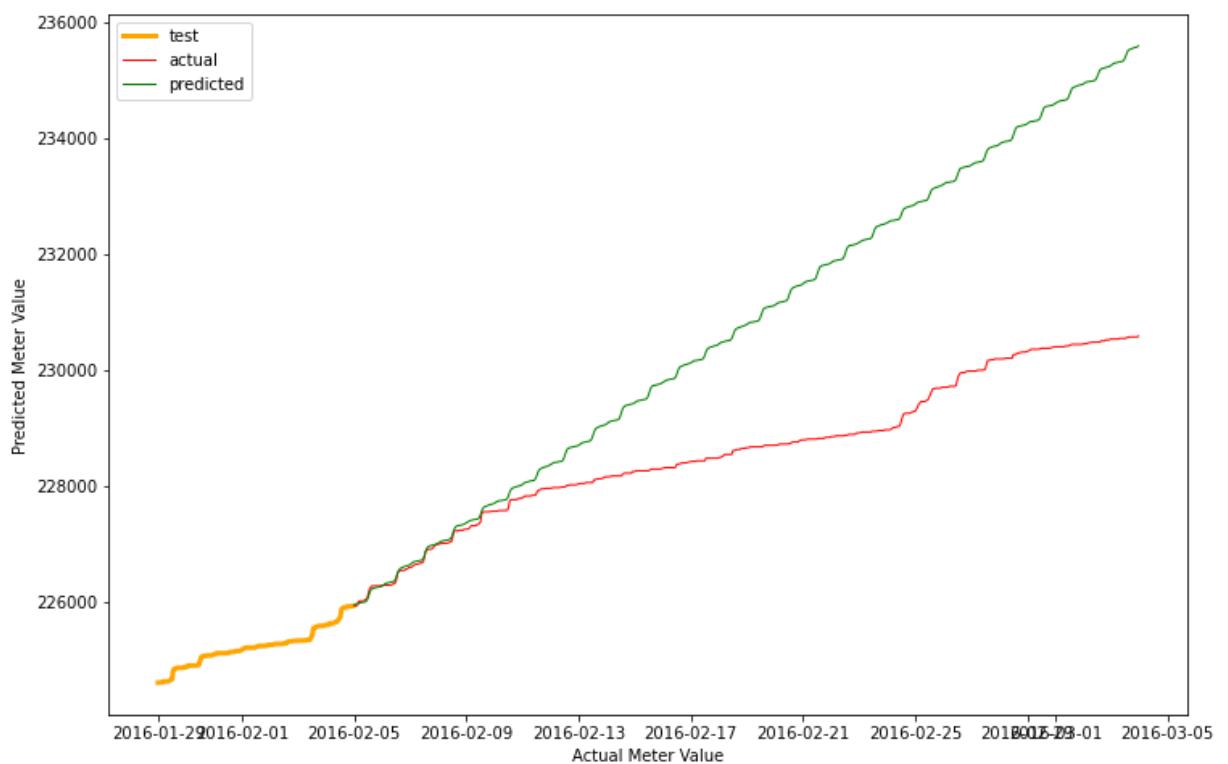
Household 1800.0  
Testing MSE: 5499167.881954899



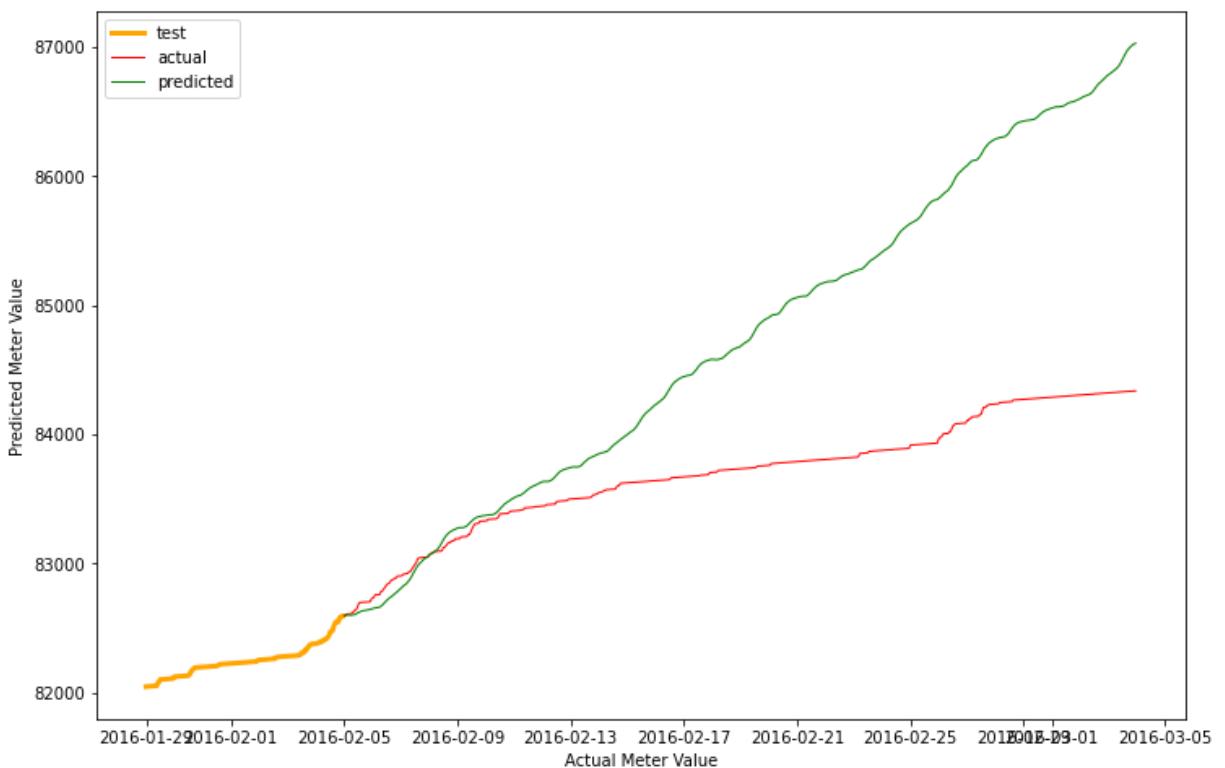
Household 1801.0  
Testing MSE: 2976625.194179081



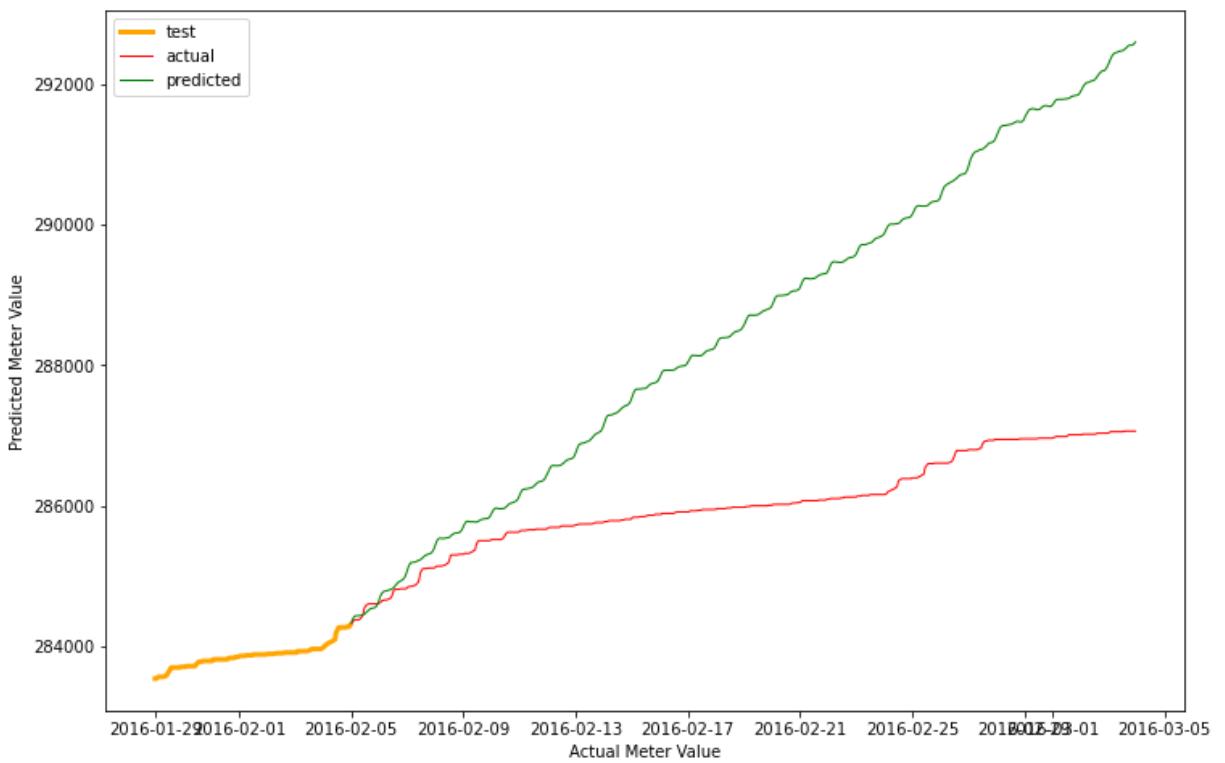
Household 2018.0  
Testing MSE: 7151796.662317268



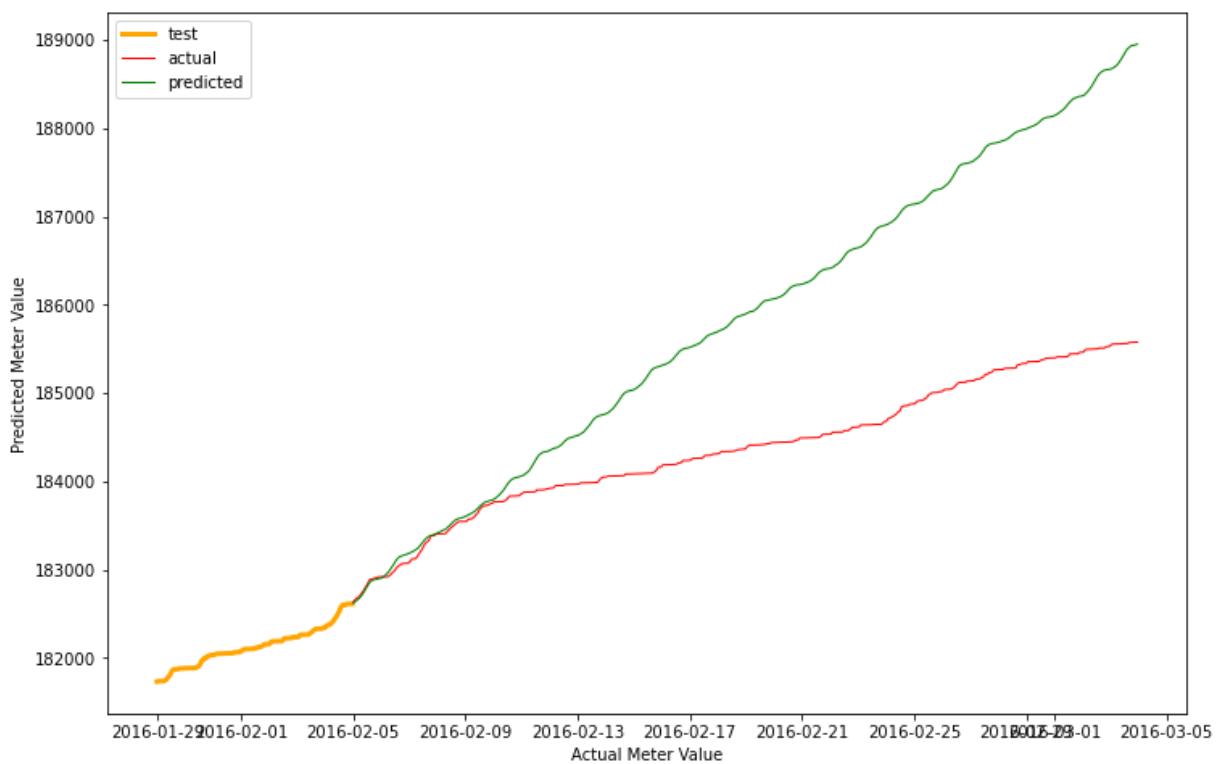
Household 2034.0  
Testing MSE: 1822592.2088178906



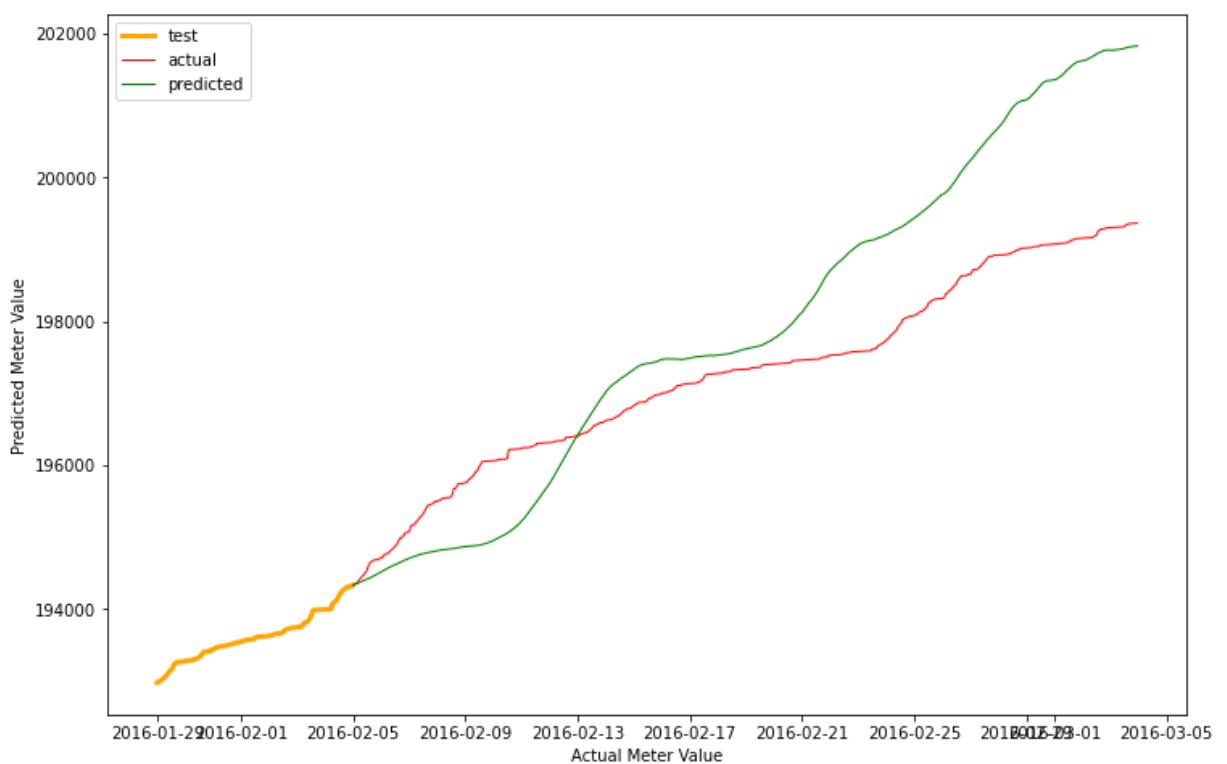
Household 2072.0  
Testing MSE: 9147926.337569464



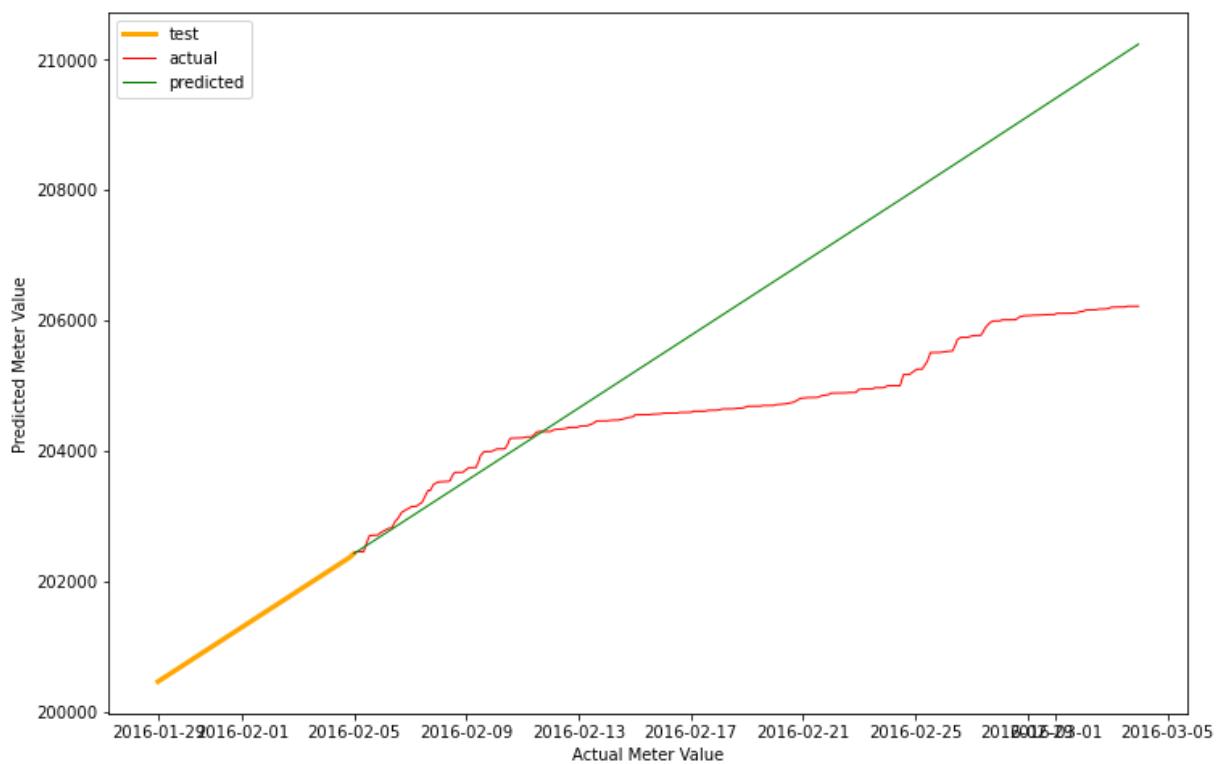
Household 2094.0  
Testing MSE: 3201305.8315091813



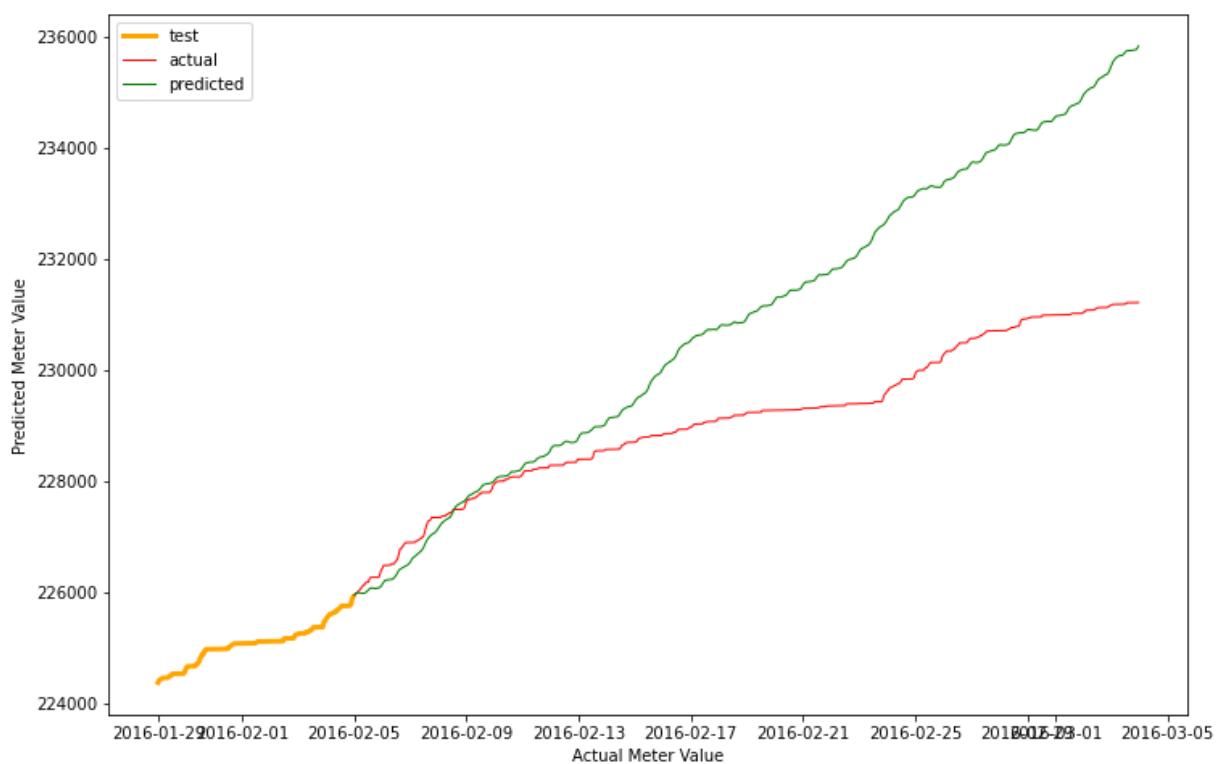
Household 2129.0  
Testing MSE: 1624107.693620591



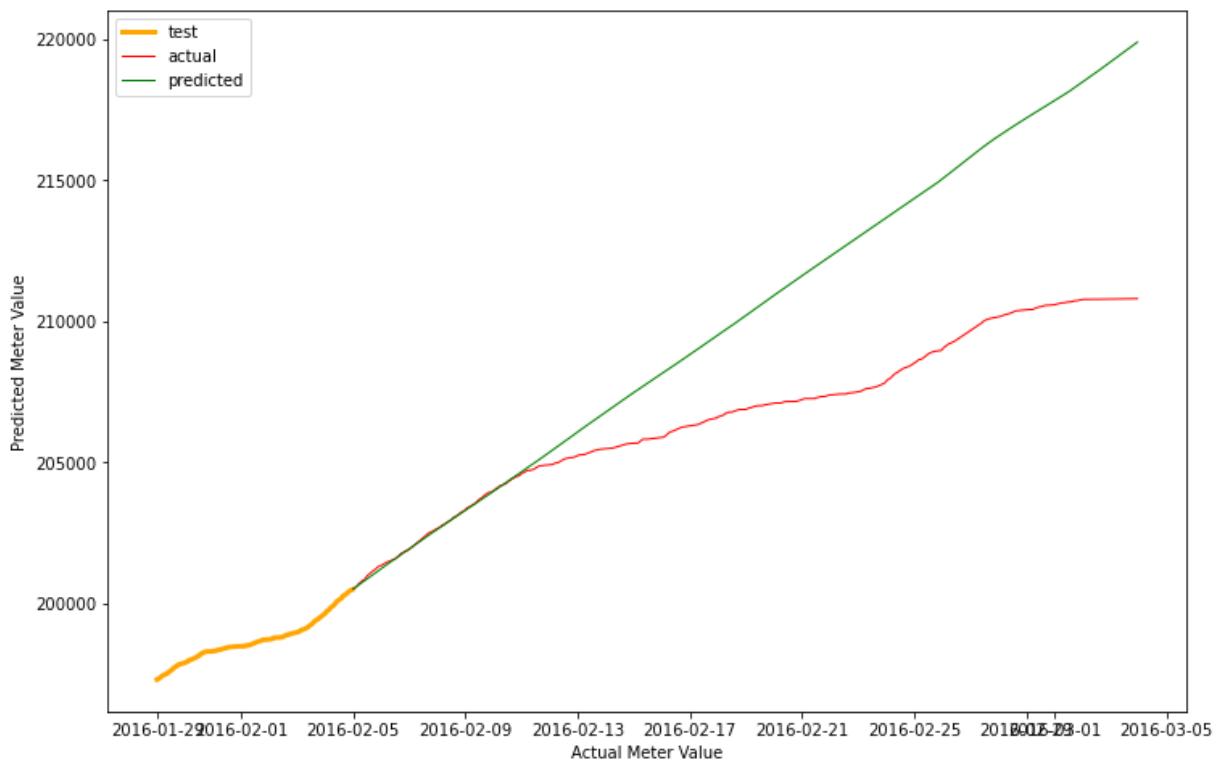
Household 2233.0  
Testing MSE: 4321402.848033329



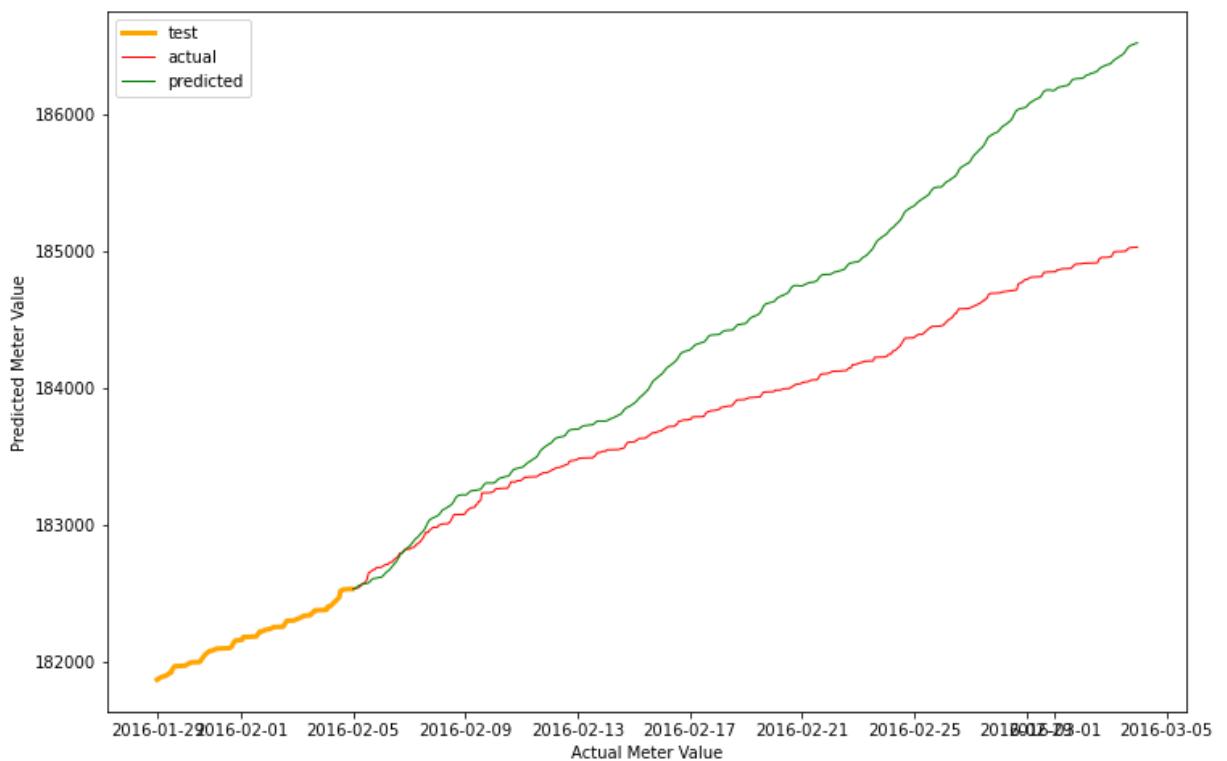
Household 2335.0  
Testing MSE: 5425720.374178098



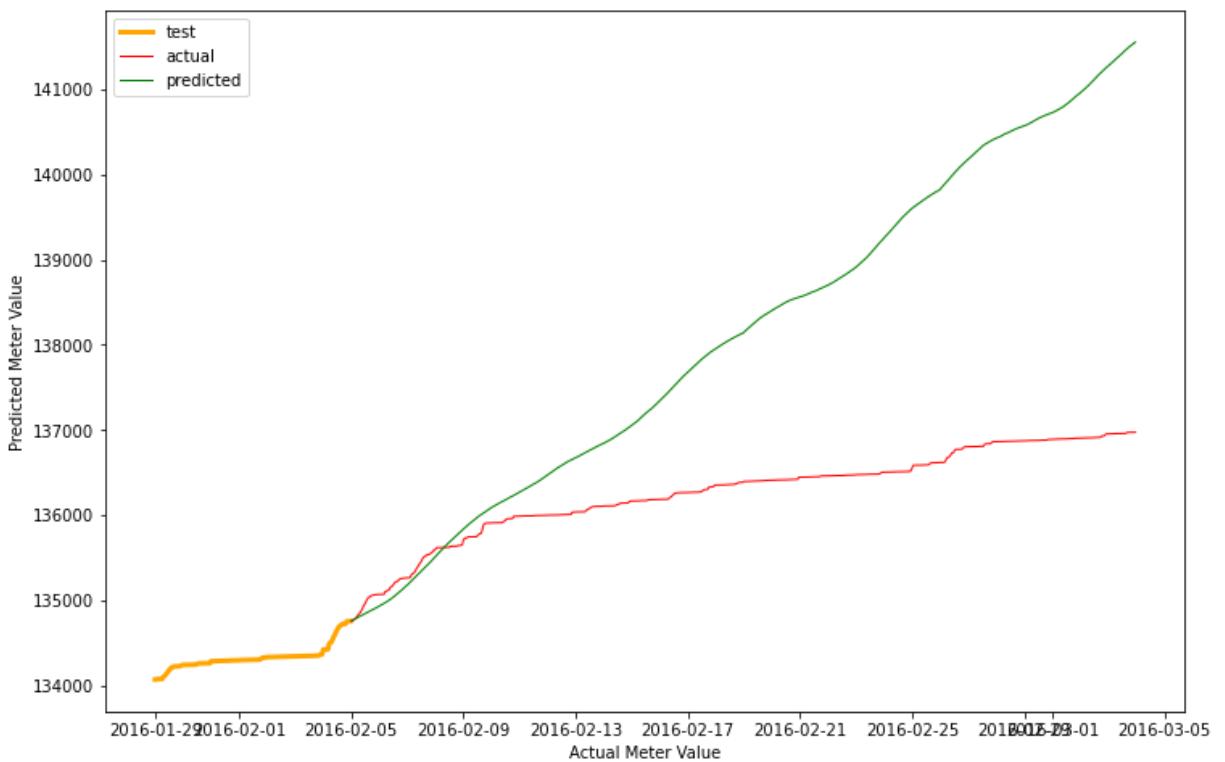
Household 2378.0  
Testing MSE: 20508046.80049327



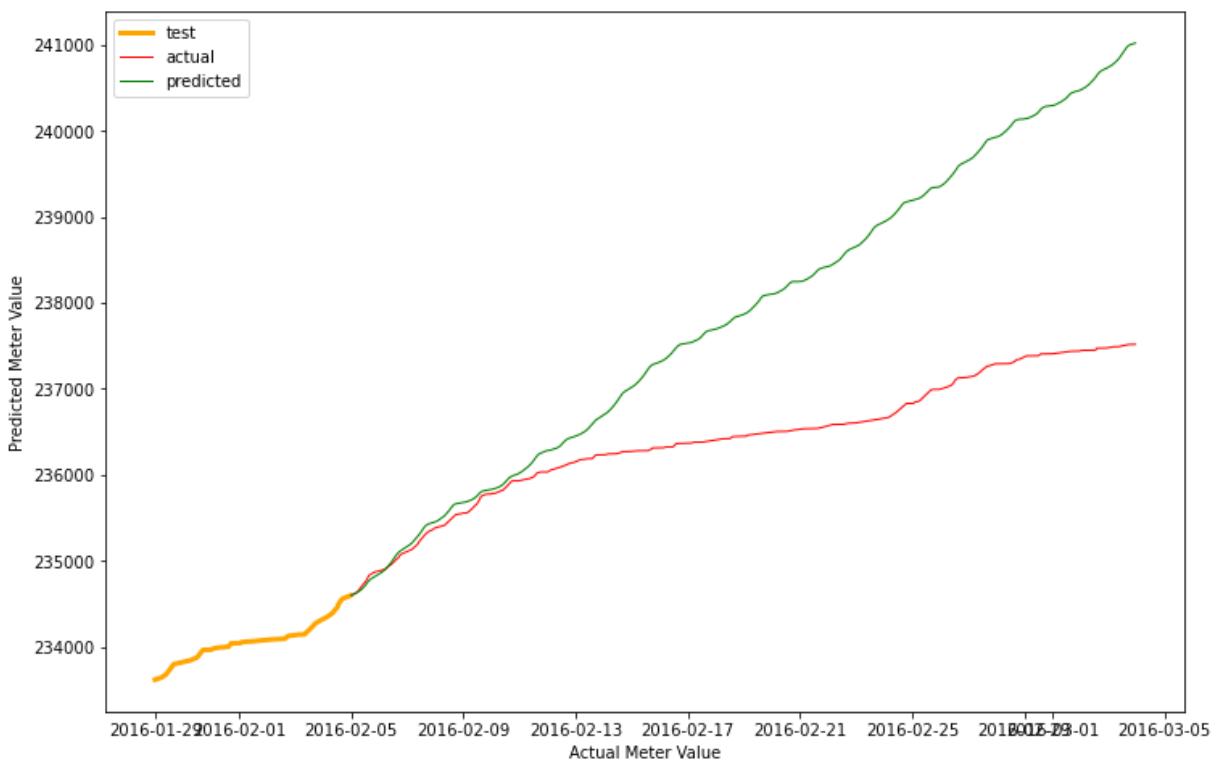
Household 2449.0  
Testing MSE: 608086.3818753771



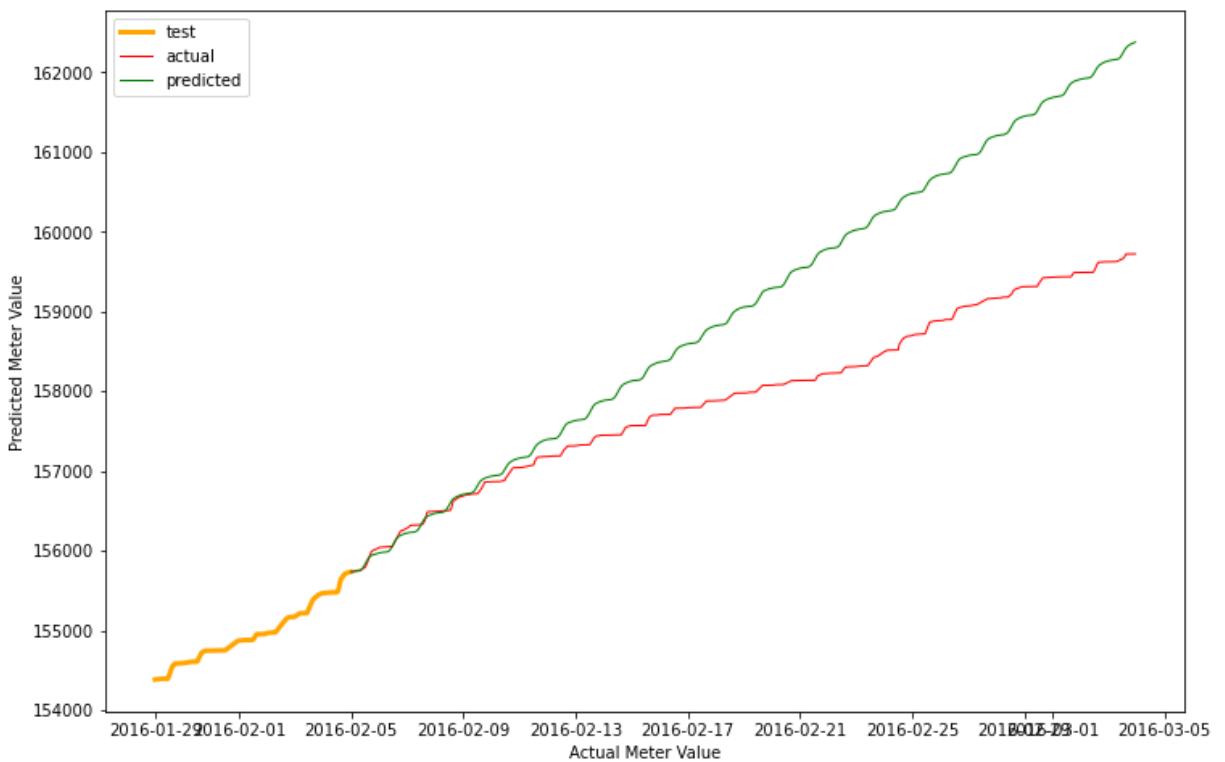
Household 2461.0  
Testing MSE: 5576559.028985886



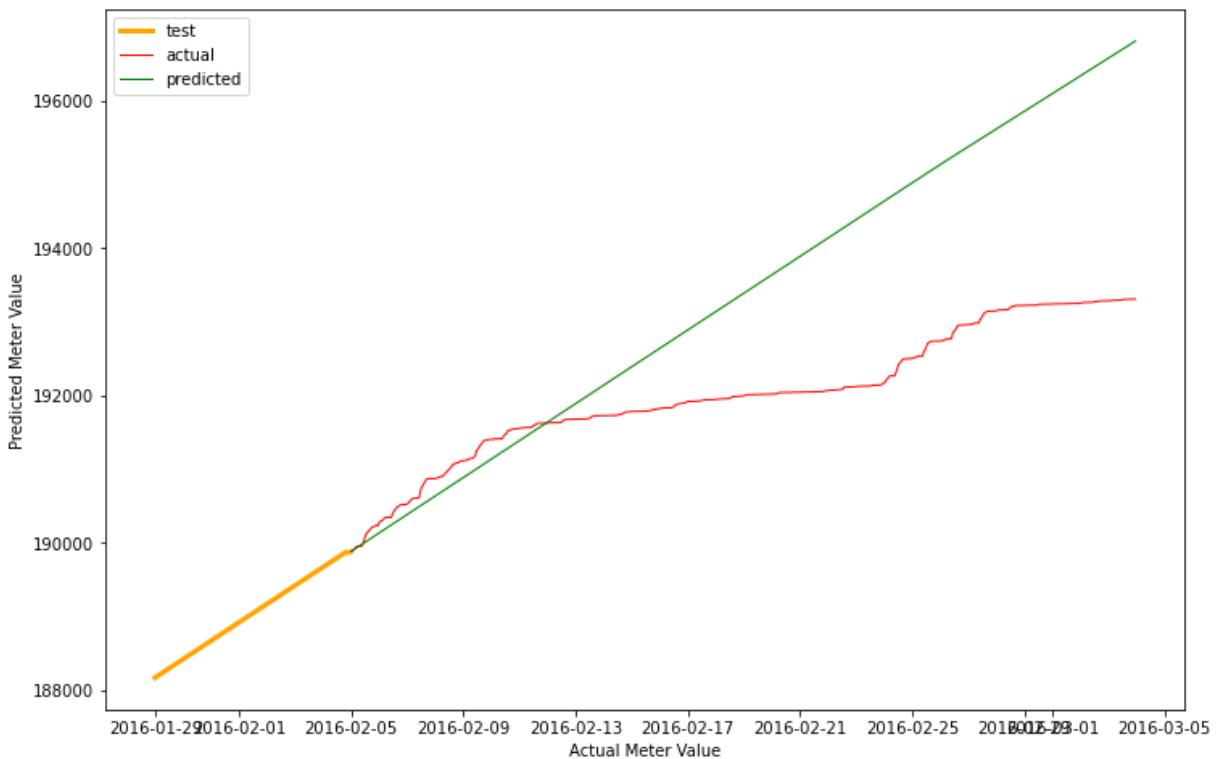
Household 2470.0  
Testing MSE: 3285804.337272809



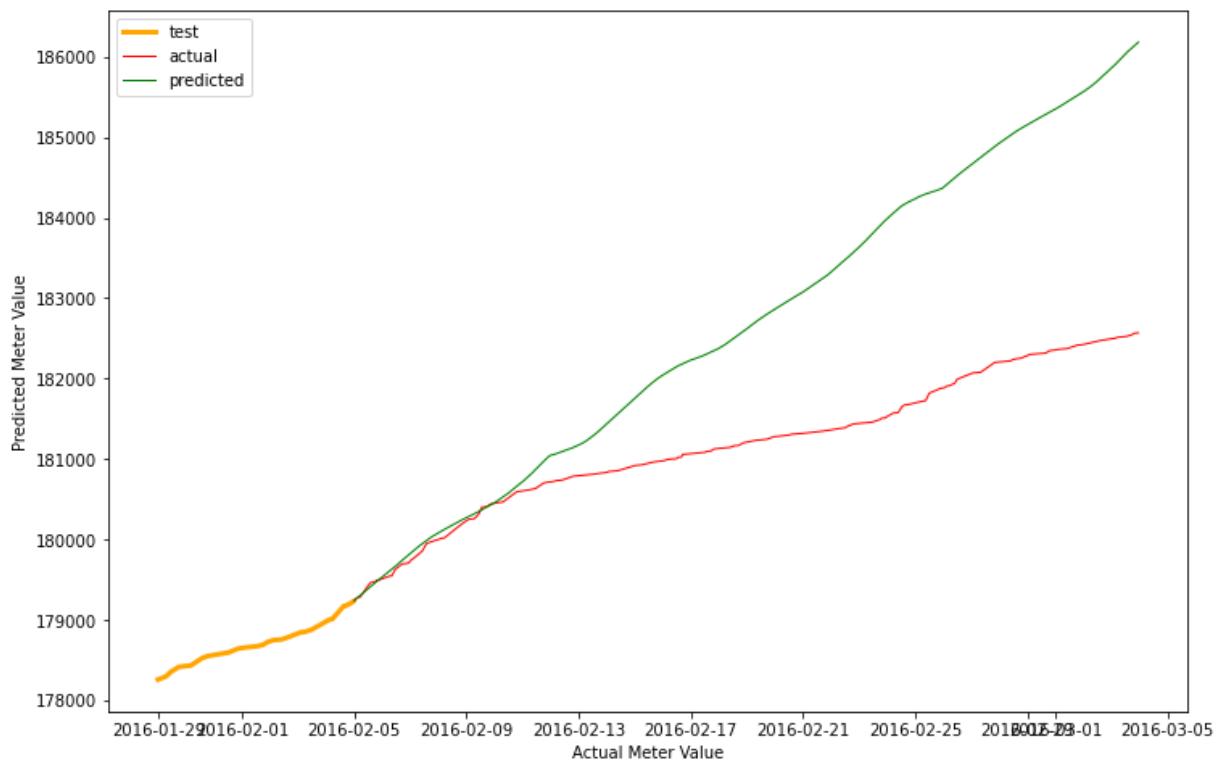
Household 2575.0  
Testing MSE: 1959162.556035524



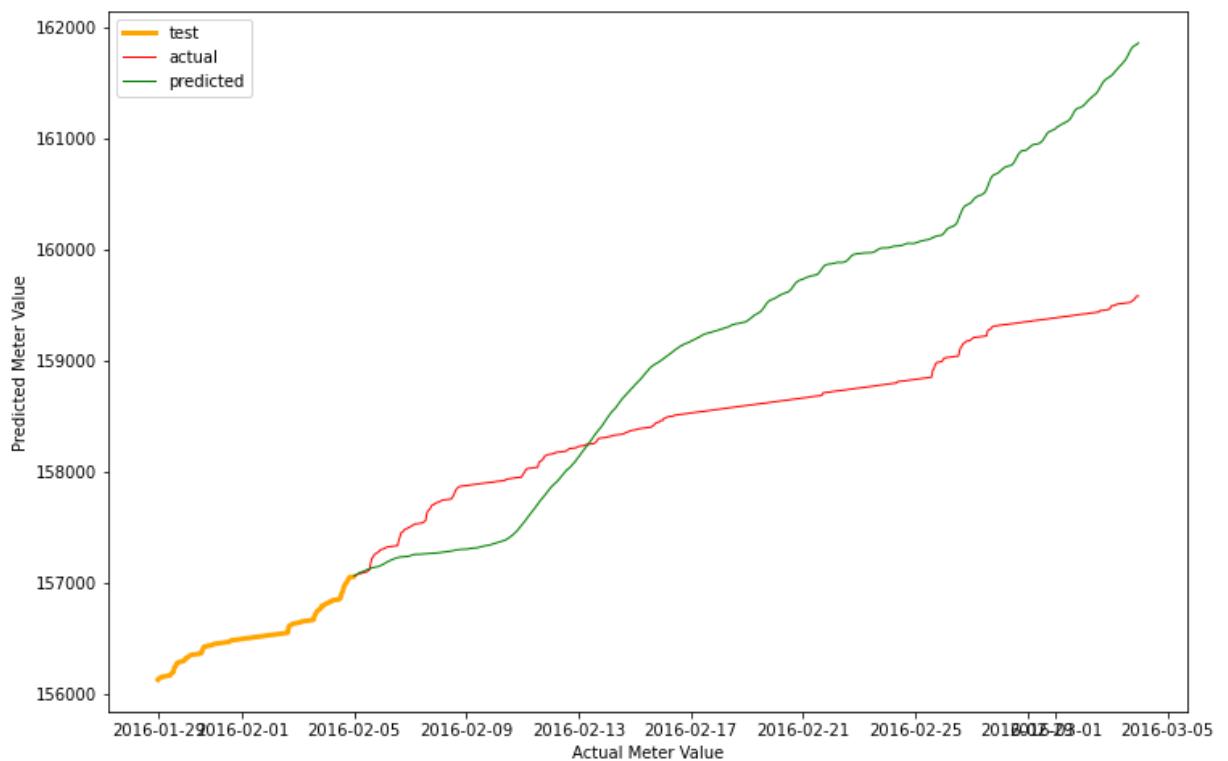
Household 2638.0  
Testing MSE: 3277753.7321163565



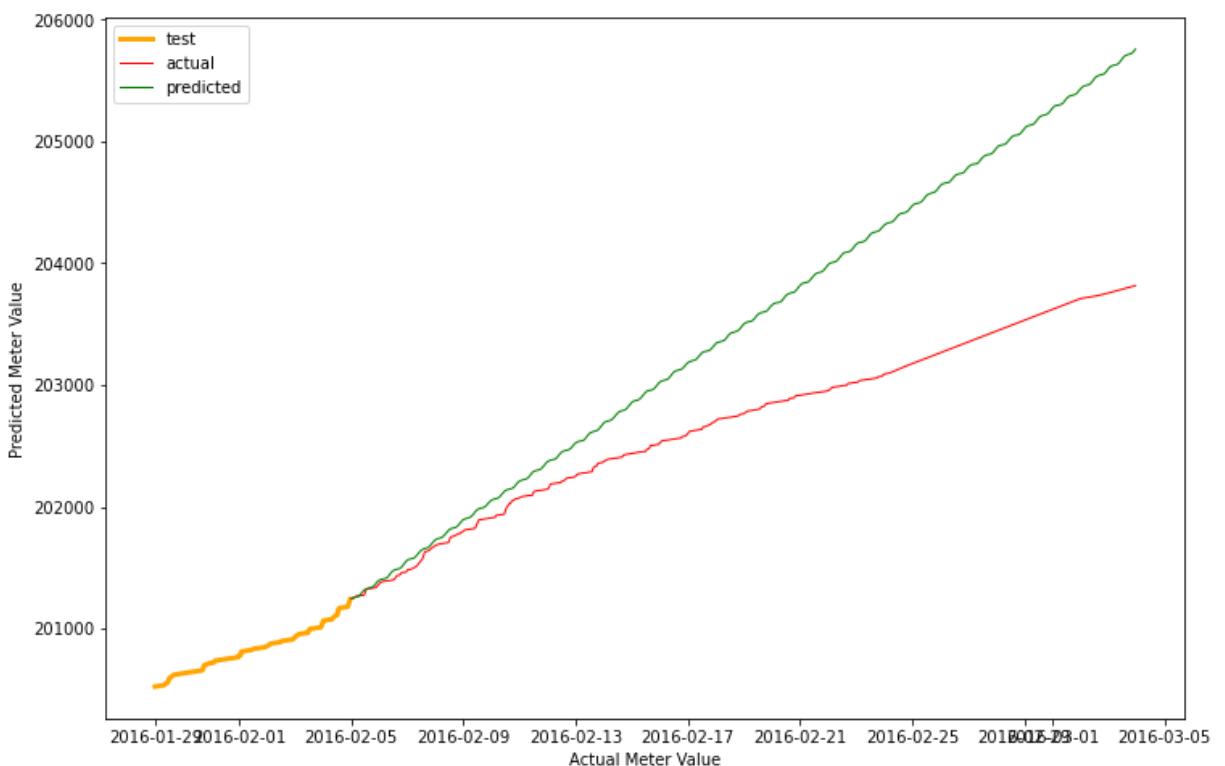
Household 2818.0  
Testing MSE: 3542455.186728921



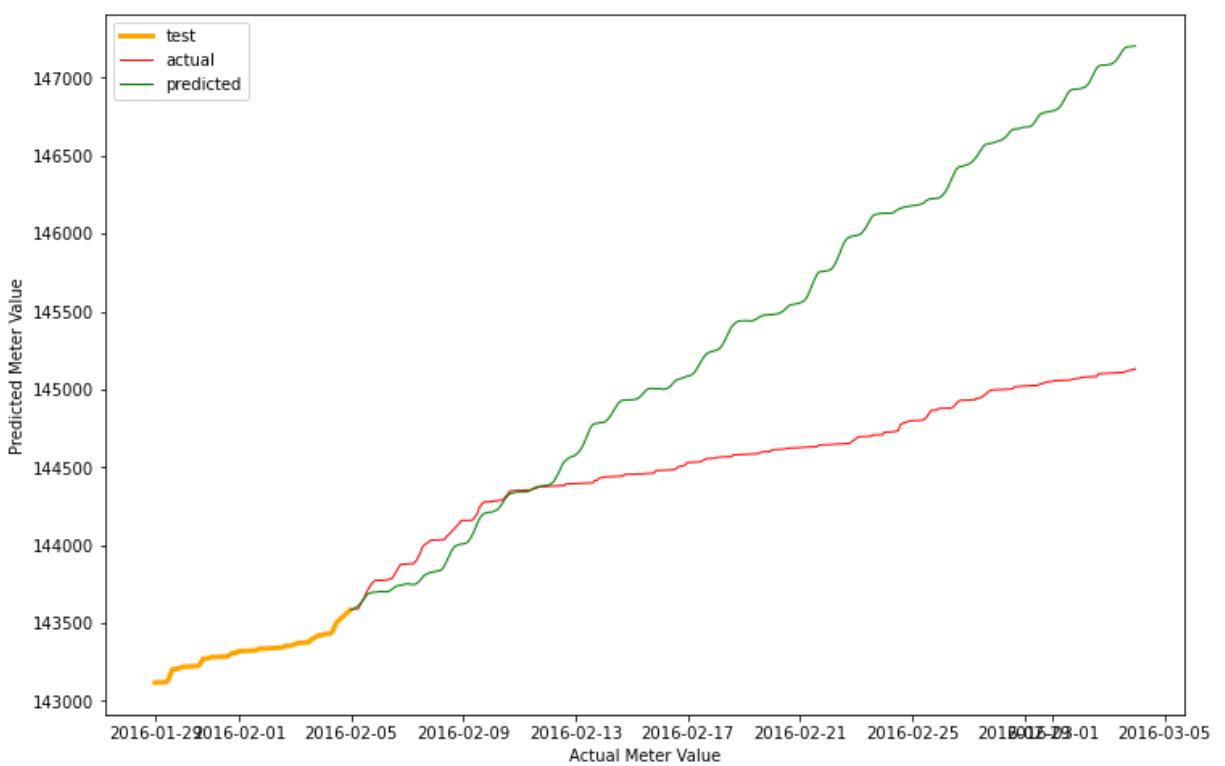
Household 2945.0  
Testing MSE: 1123978.7728980333



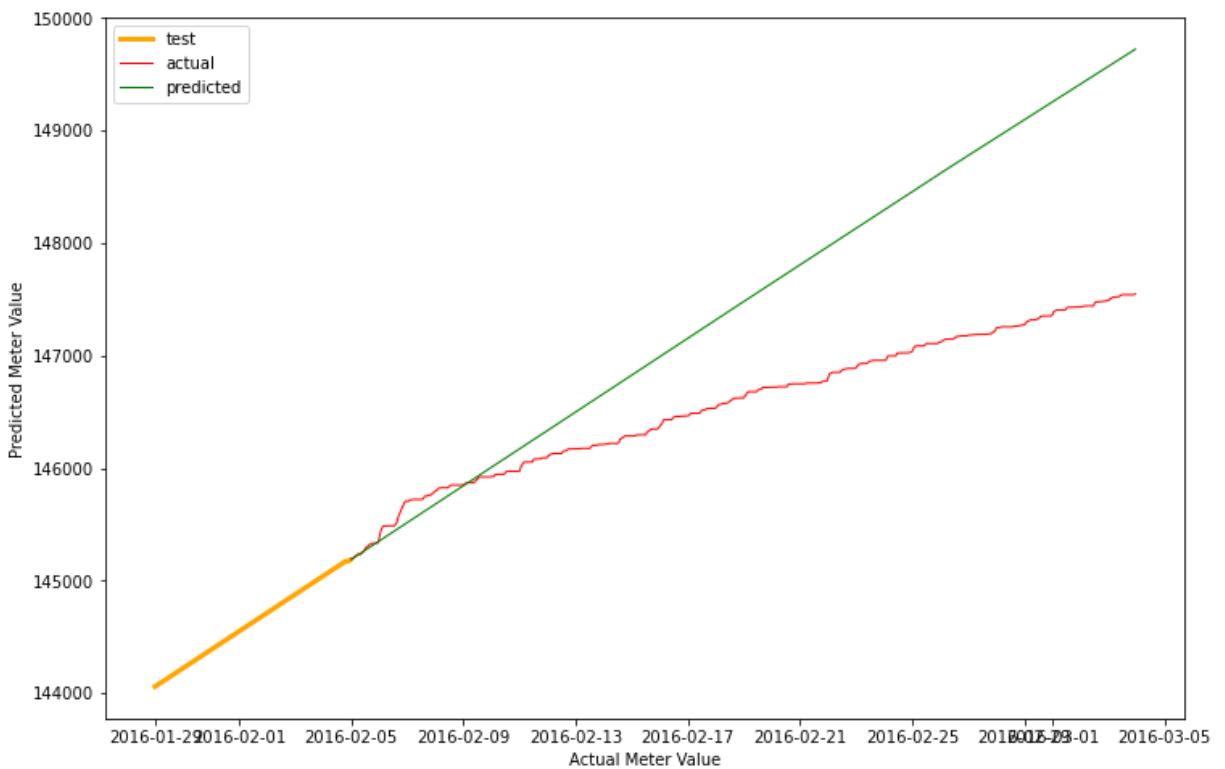
Household 2965.0  
Testing MSE: 1008215.5786349631



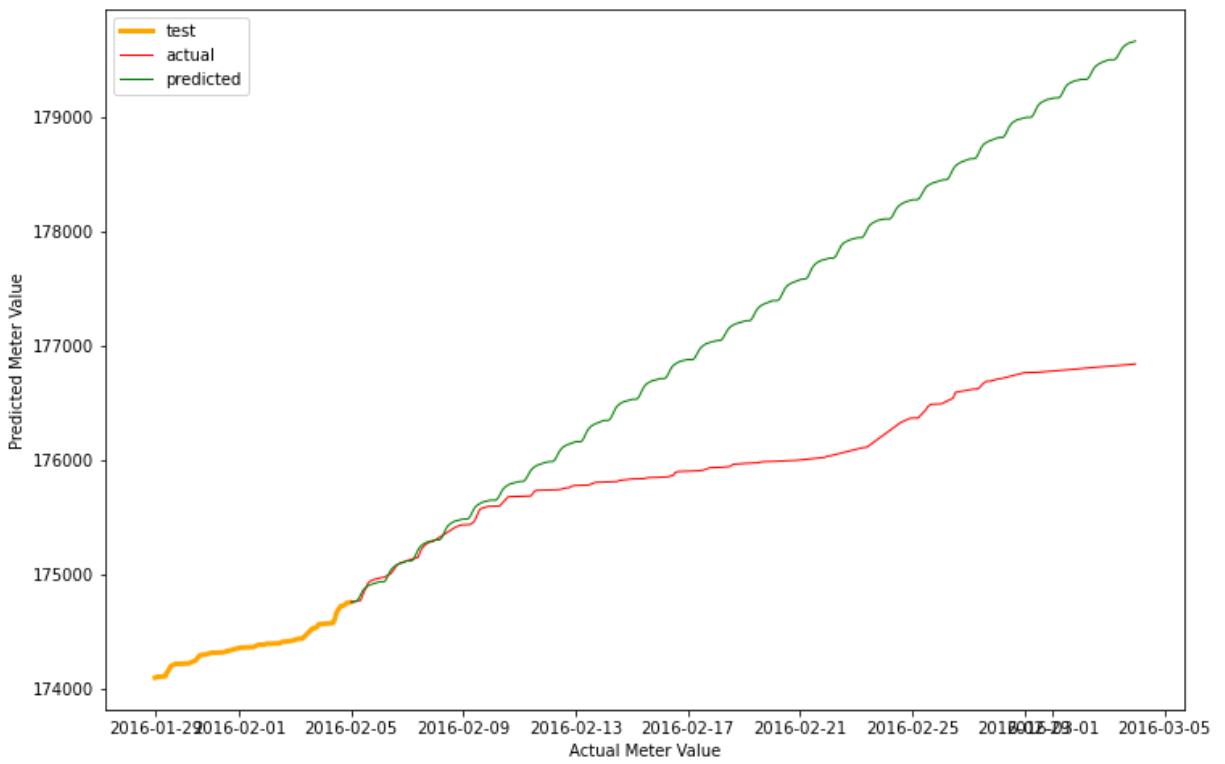
Household 2980.0  
Testing MSE: 1184992.2793116784



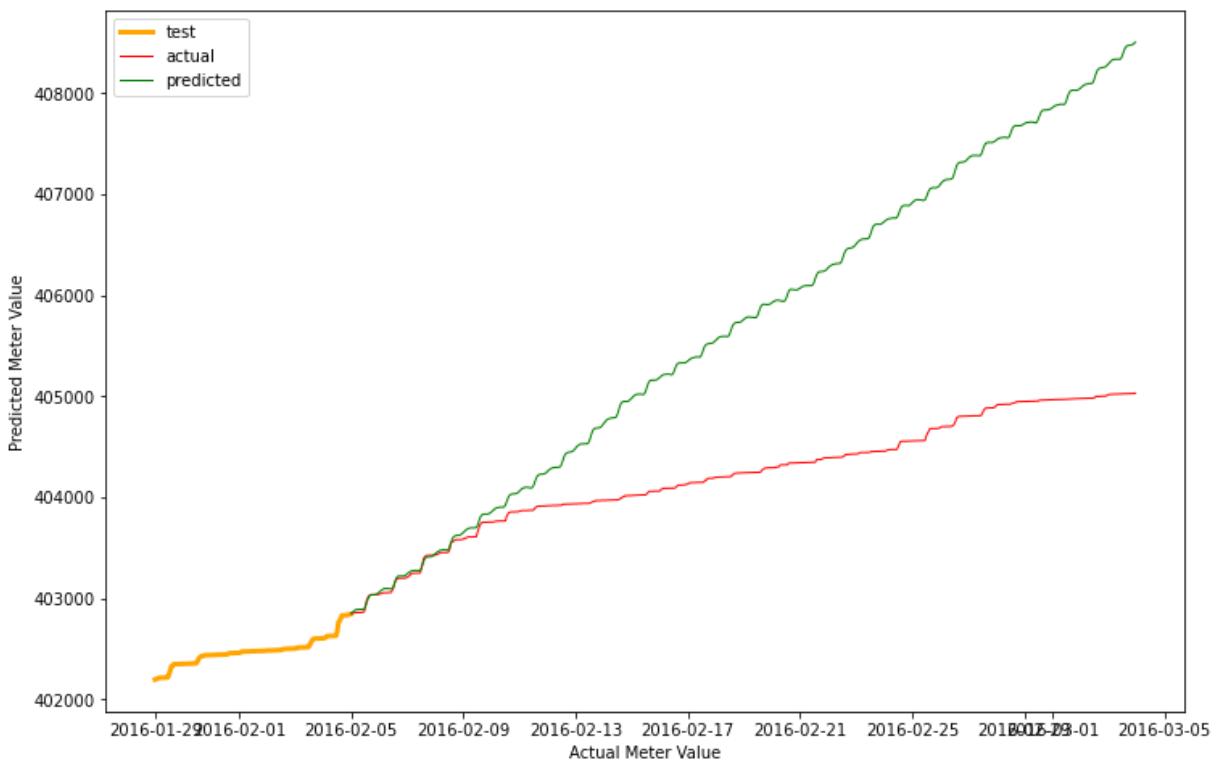
Household 3039.0  
Testing MSE: 1298754.9163097804



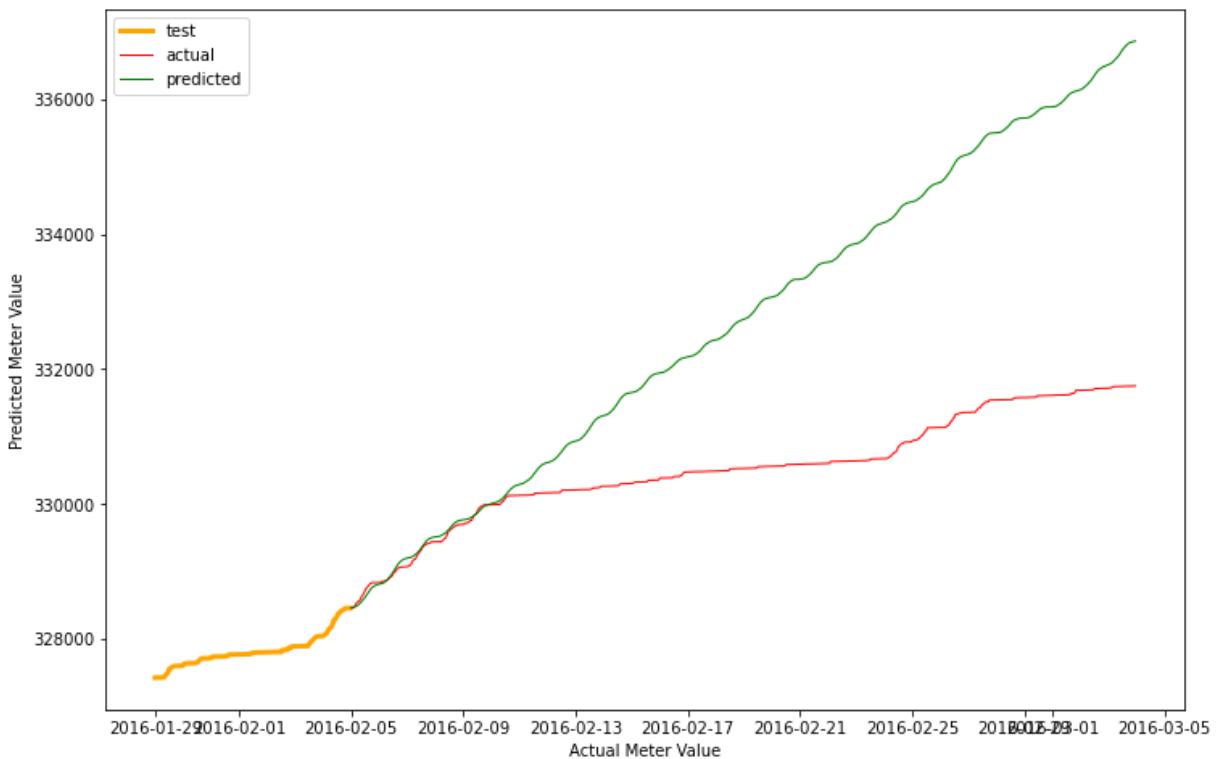
Household 3134.0  
Testing MSE: 2291743.539972858



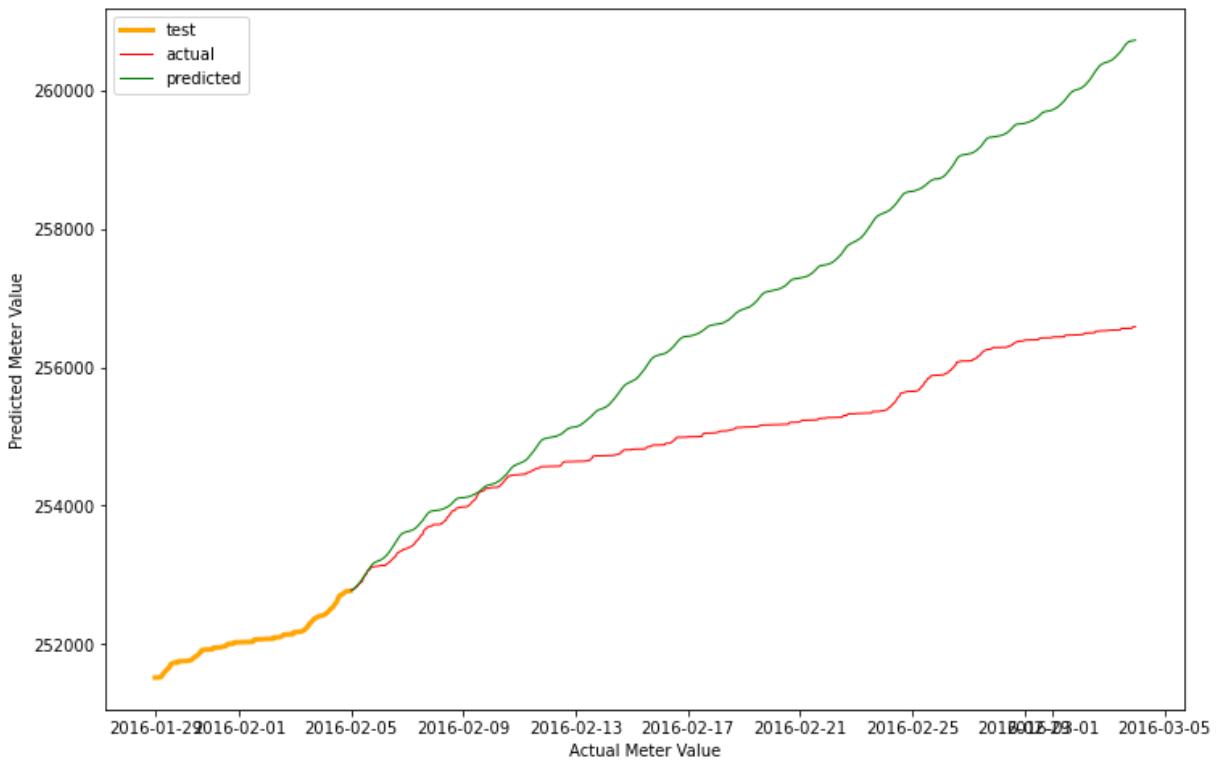
Household 3310.0  
Testing MSE: 3357726.7645221693



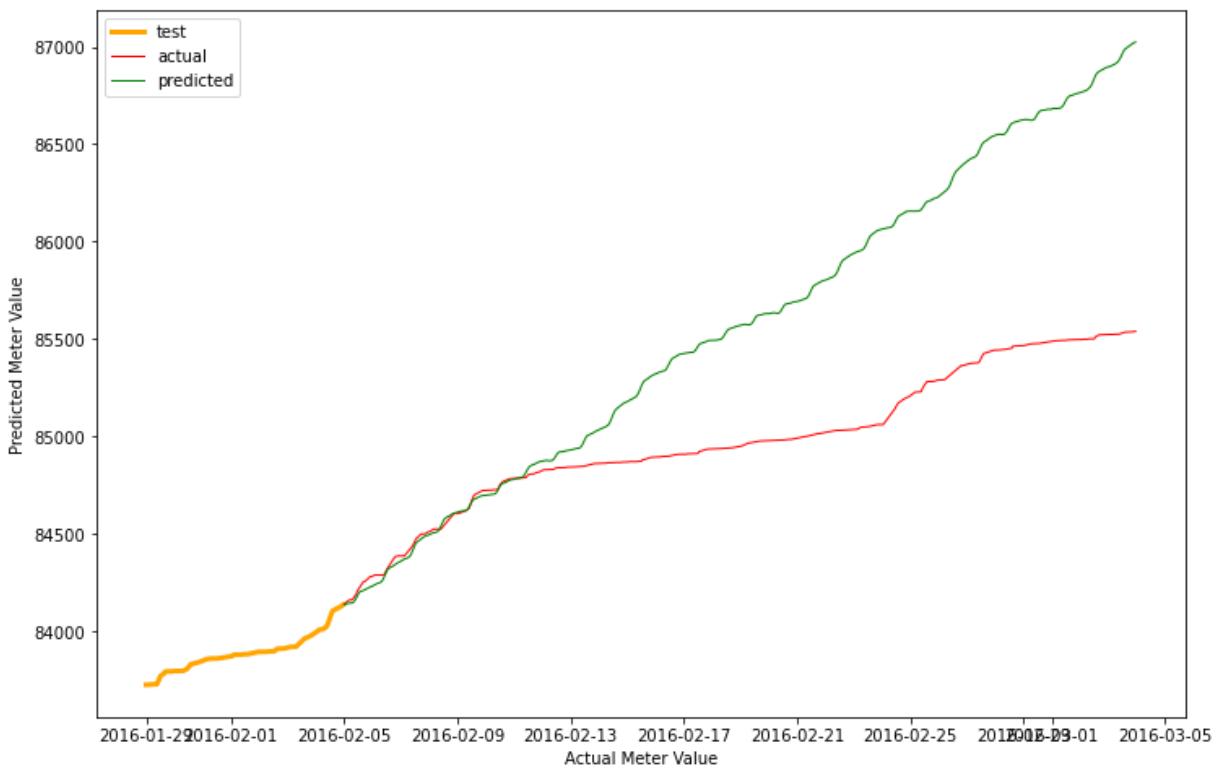
Household 3367.0  
Testing MSE: 7520897.773588635



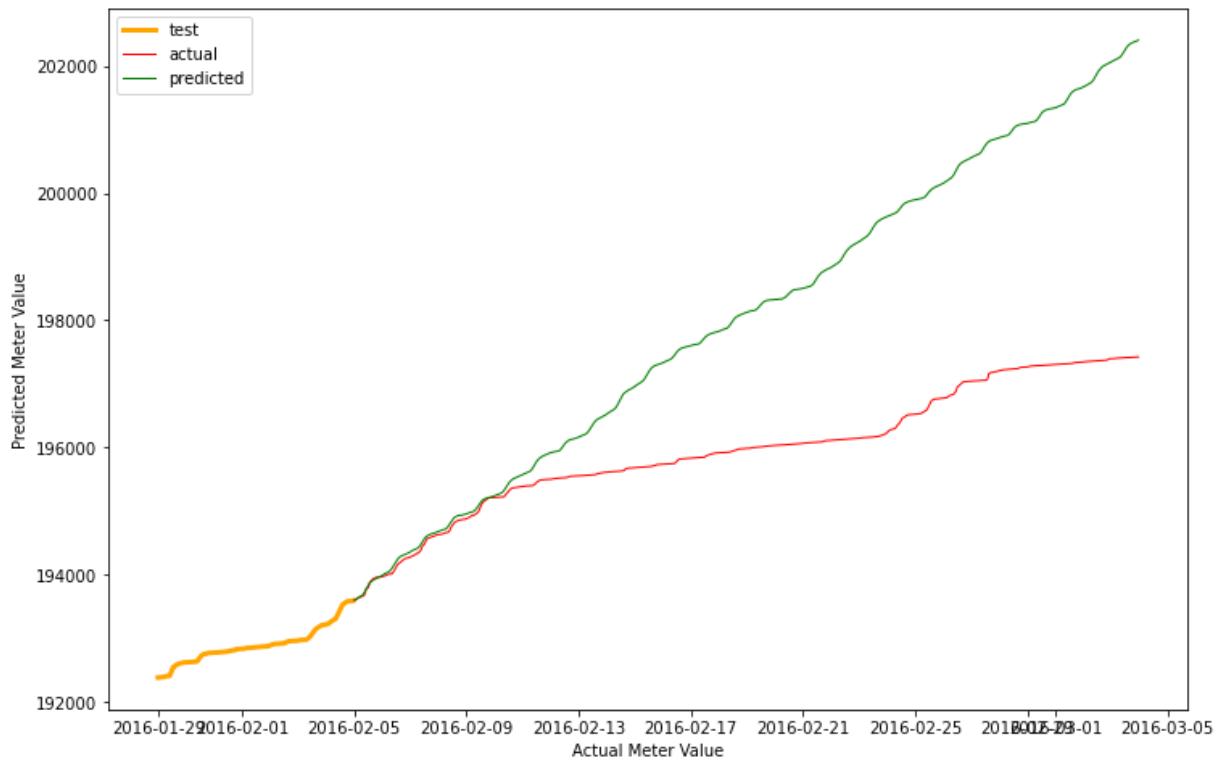
Household 3527.0  
Testing MSE: 4645520.042836751



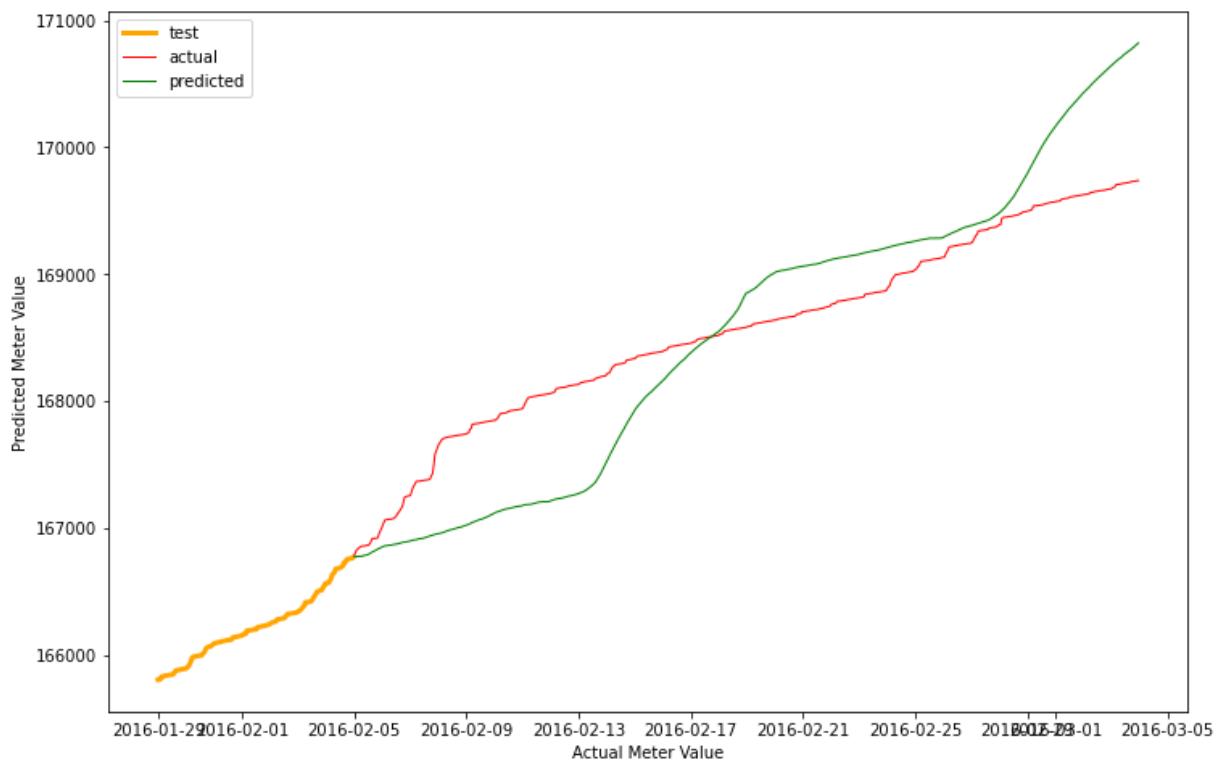
Household 3544.0  
Testing MSE: 570416.5714502314



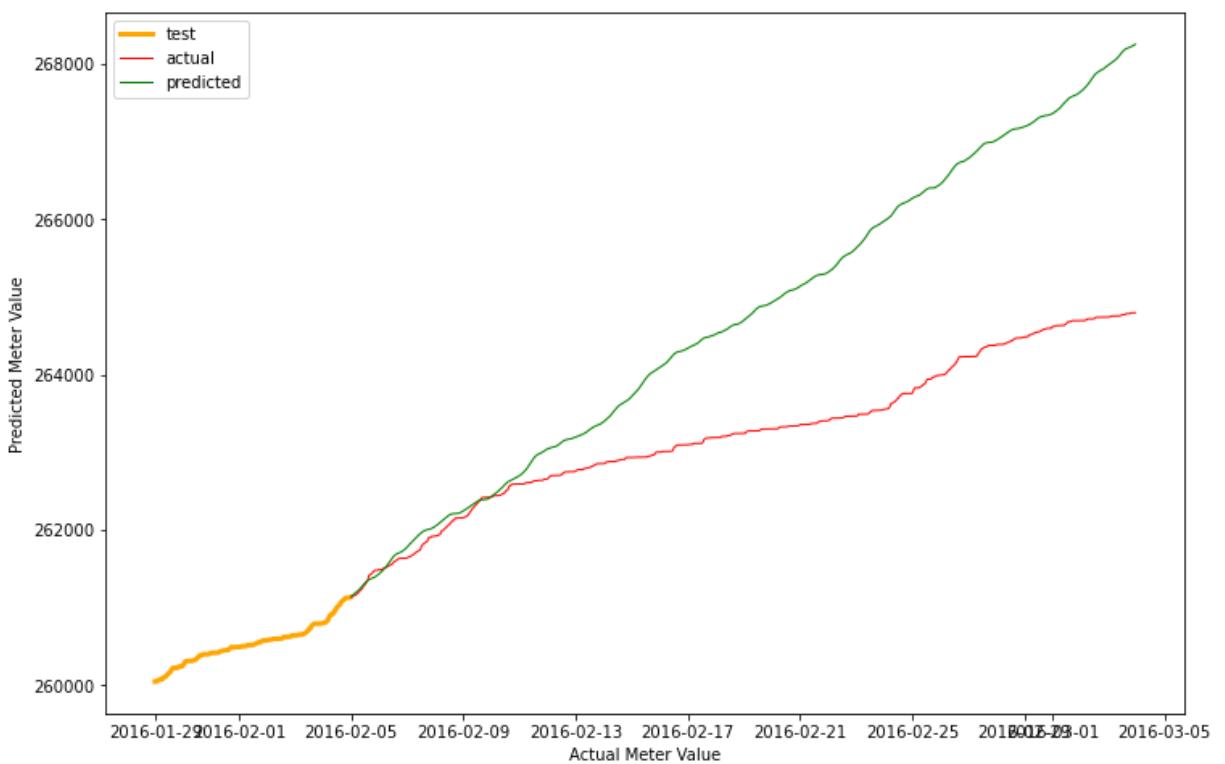
Household 3577.0  
Testing MSE: 6756006.151253728



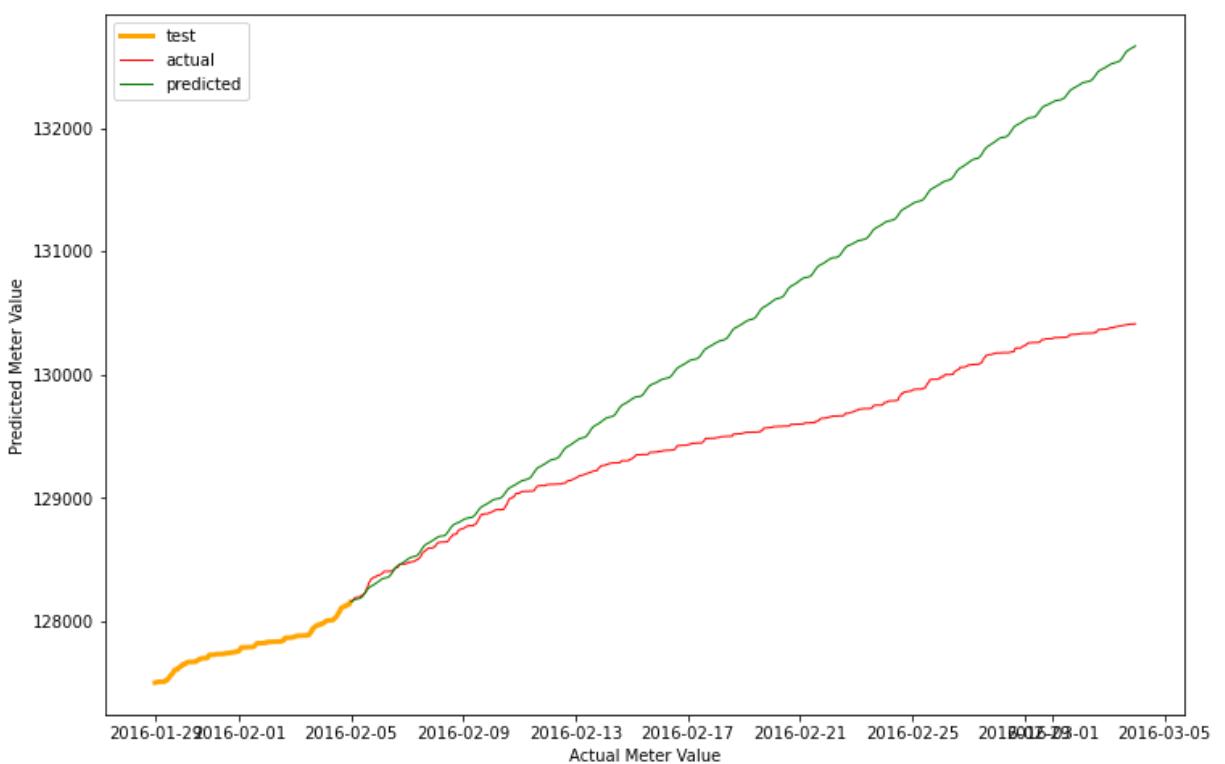
Household 3635.0  
Testing MSE: 278862.31404198543



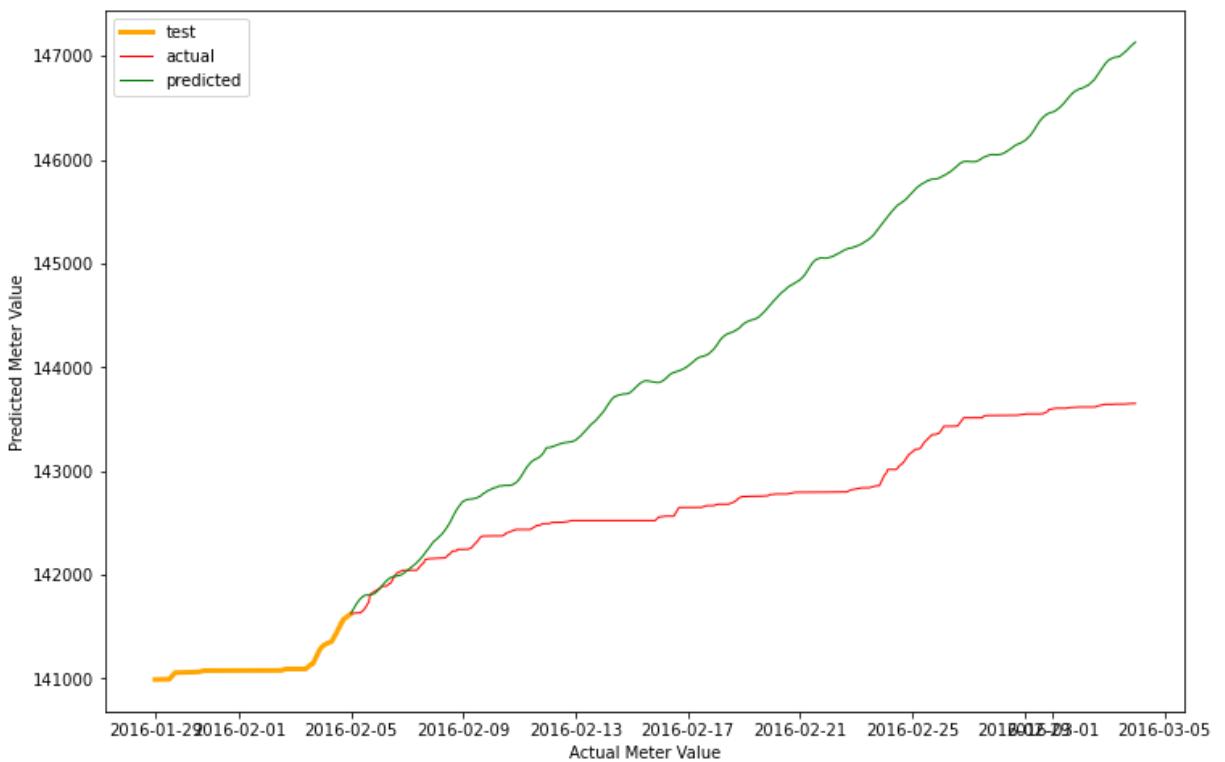
Household 3723.0  
Testing MSE: 3362224.3296120455



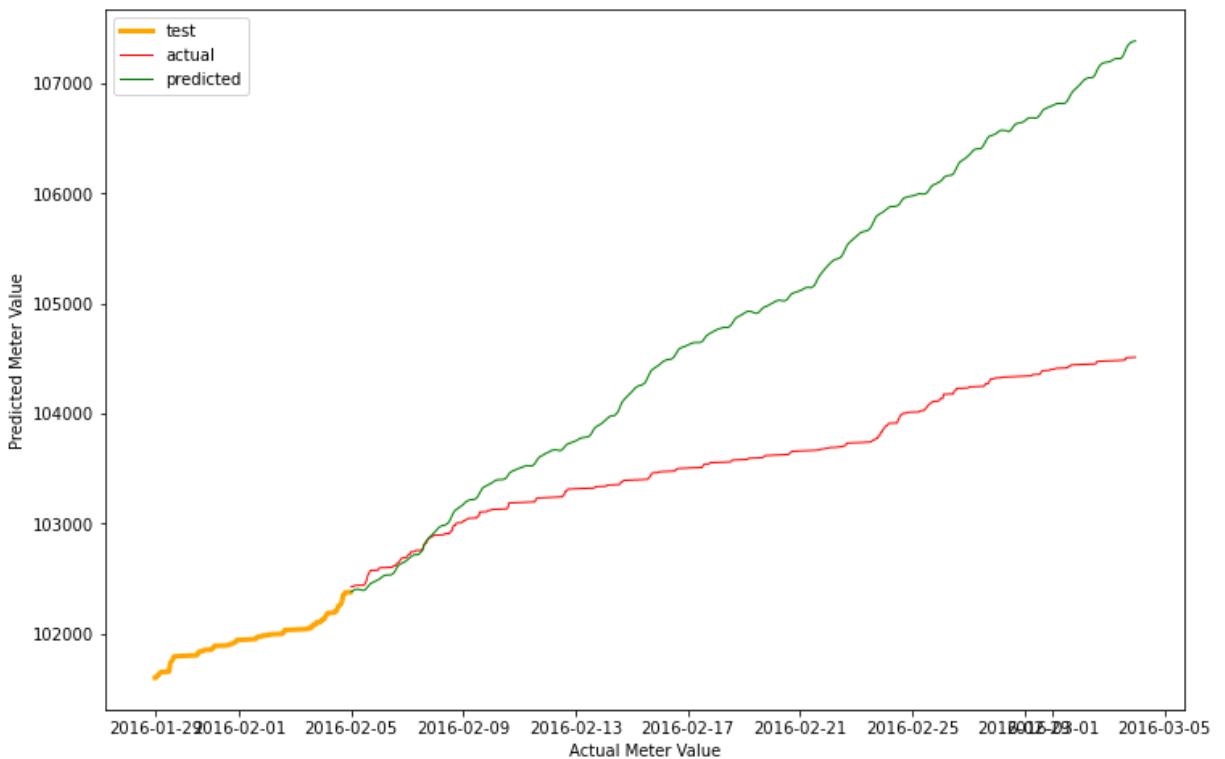
Household 3778.0  
Testing MSE: 1393576.7537089582



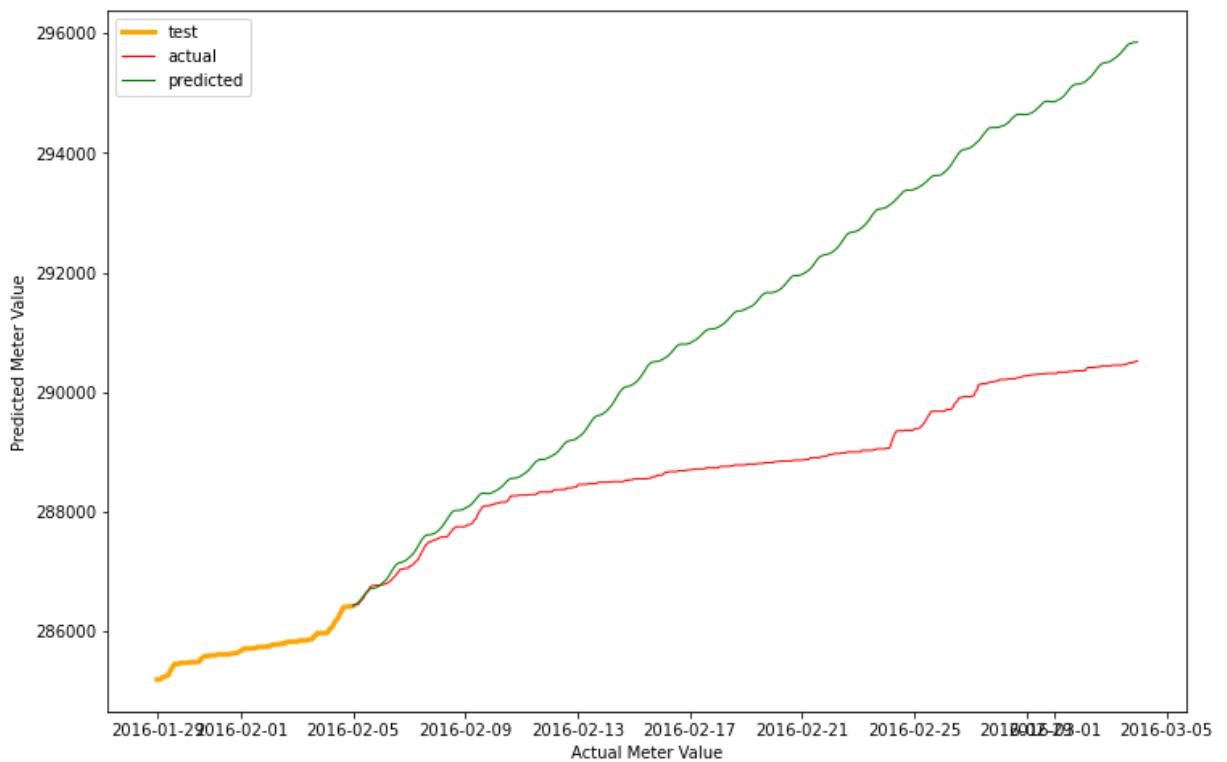
Household 3849.0  
Testing MSE: 3712123.0698783356



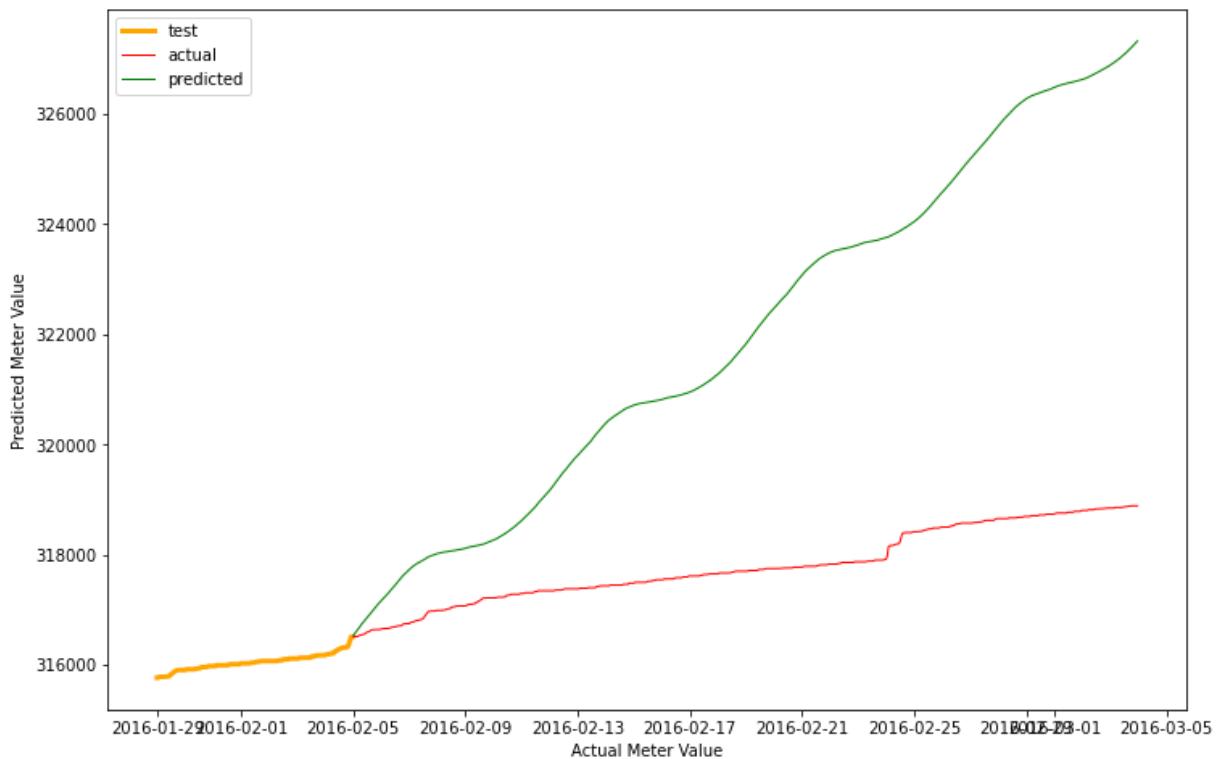
Household 3893.0  
Testing MSE: 2388232.6730327667



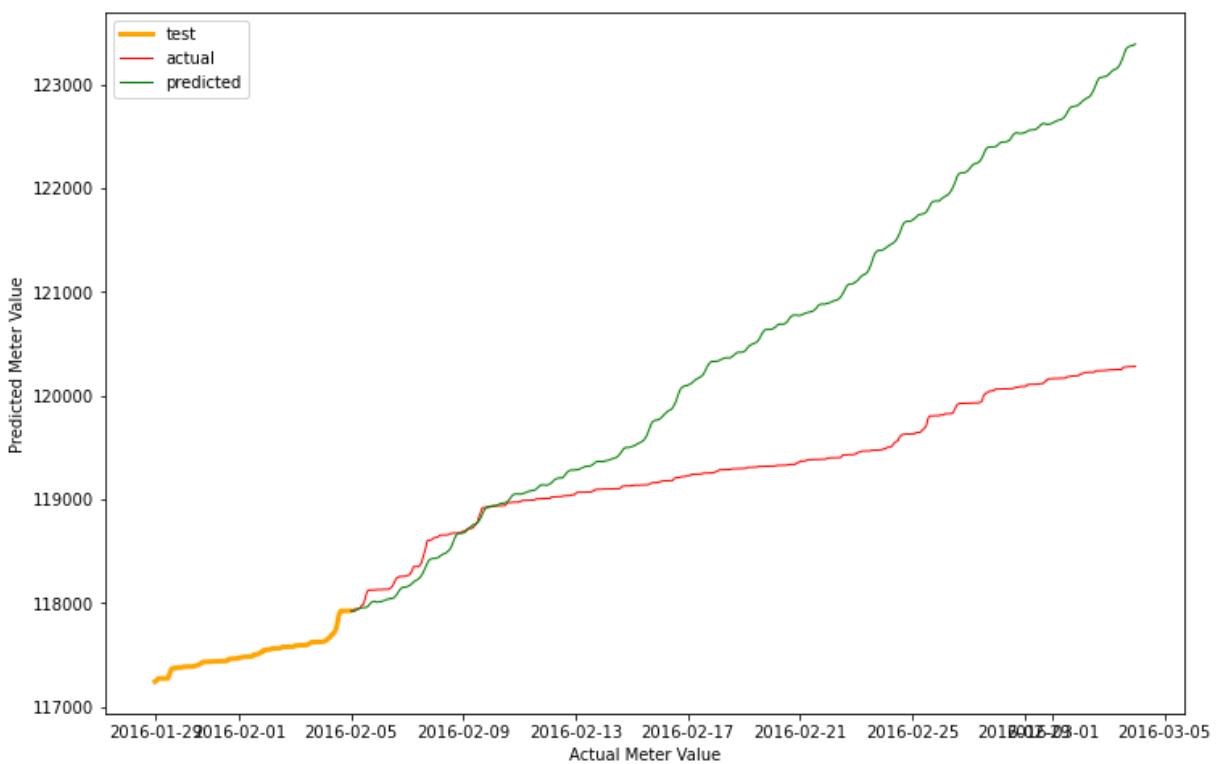
Household 3918.0  
Testing MSE: 9158986.738932937



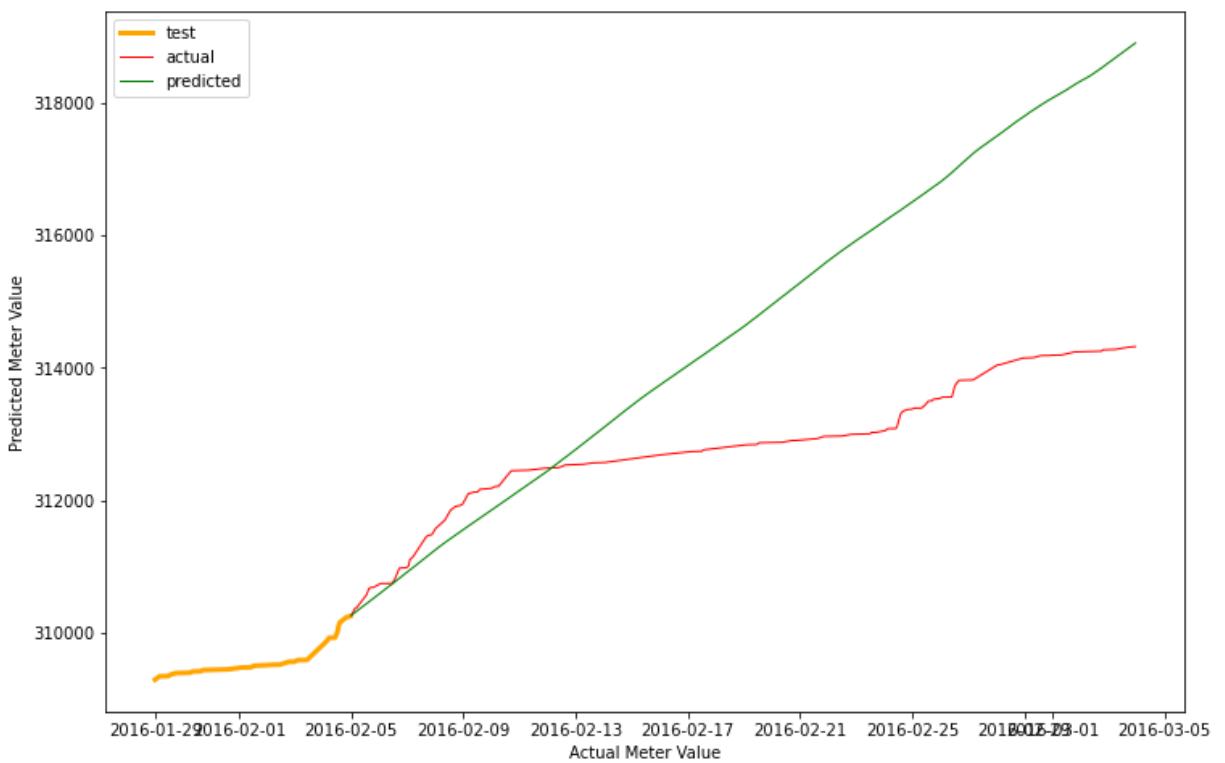
Household 4029.0  
Testing MSE: 24112221.108280532



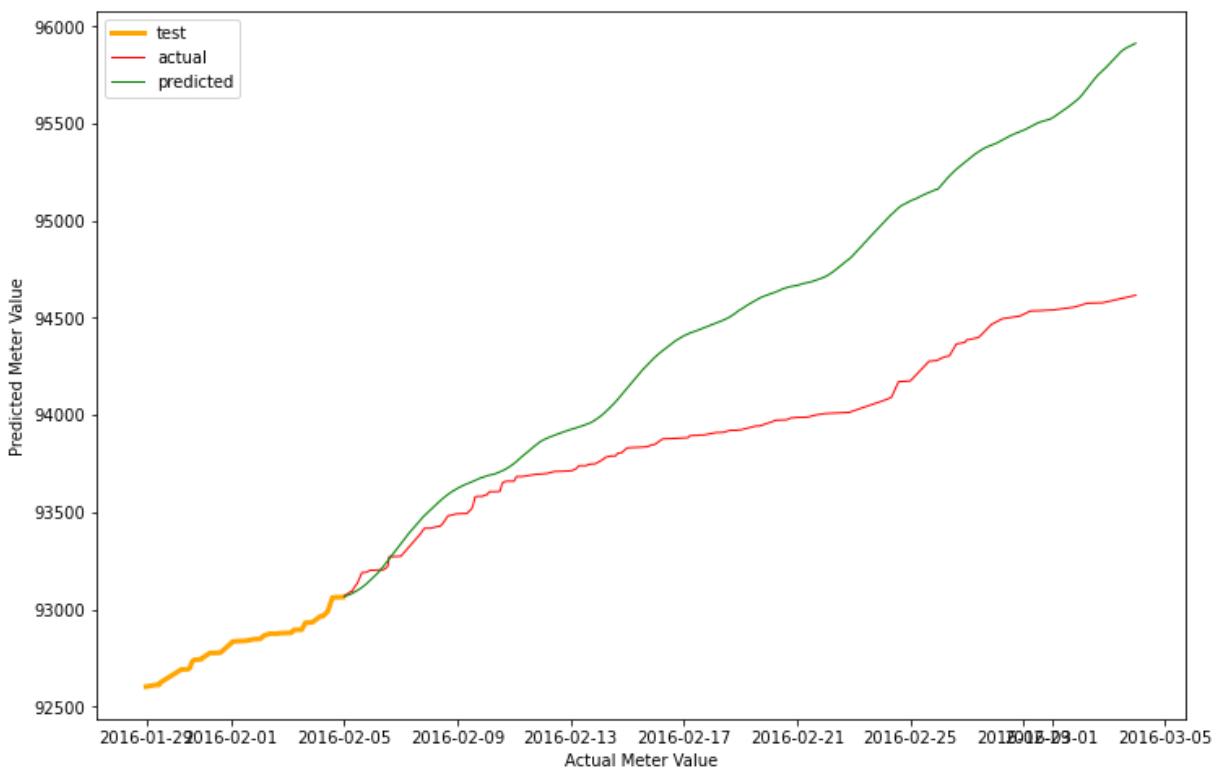
Household 4031.0  
Testing MSE: 2404544.5056948583



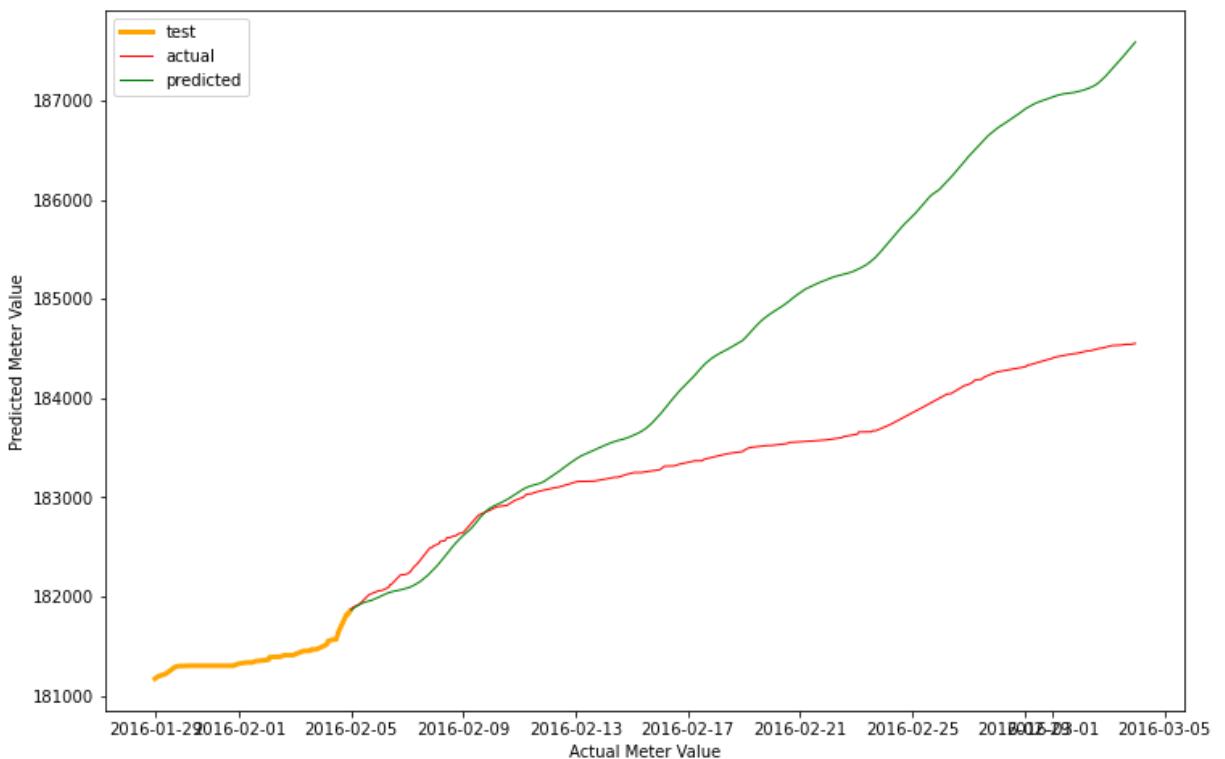
Household 4193.0  
Testing MSE: 5842570.131708123



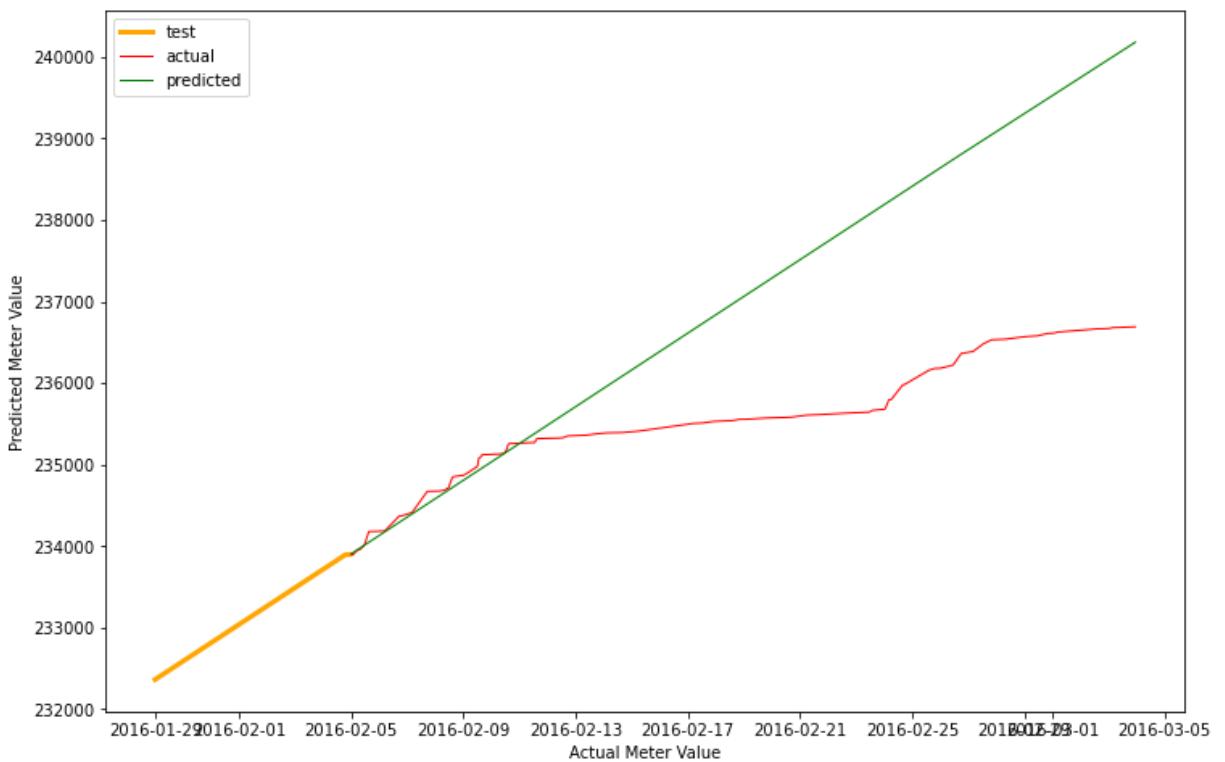
Household 4228.0  
Testing MSE: 463367.59227020433



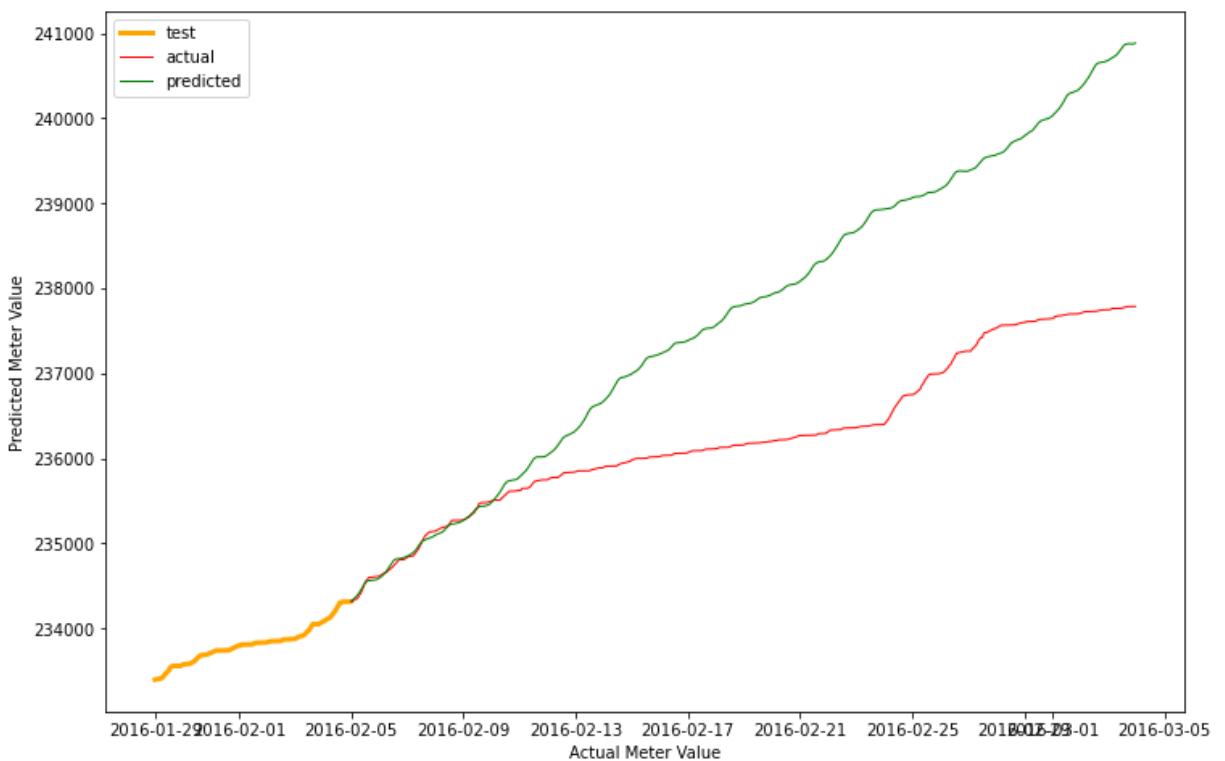
Household 4296.0  
Testing MSE: 2444874.508020242



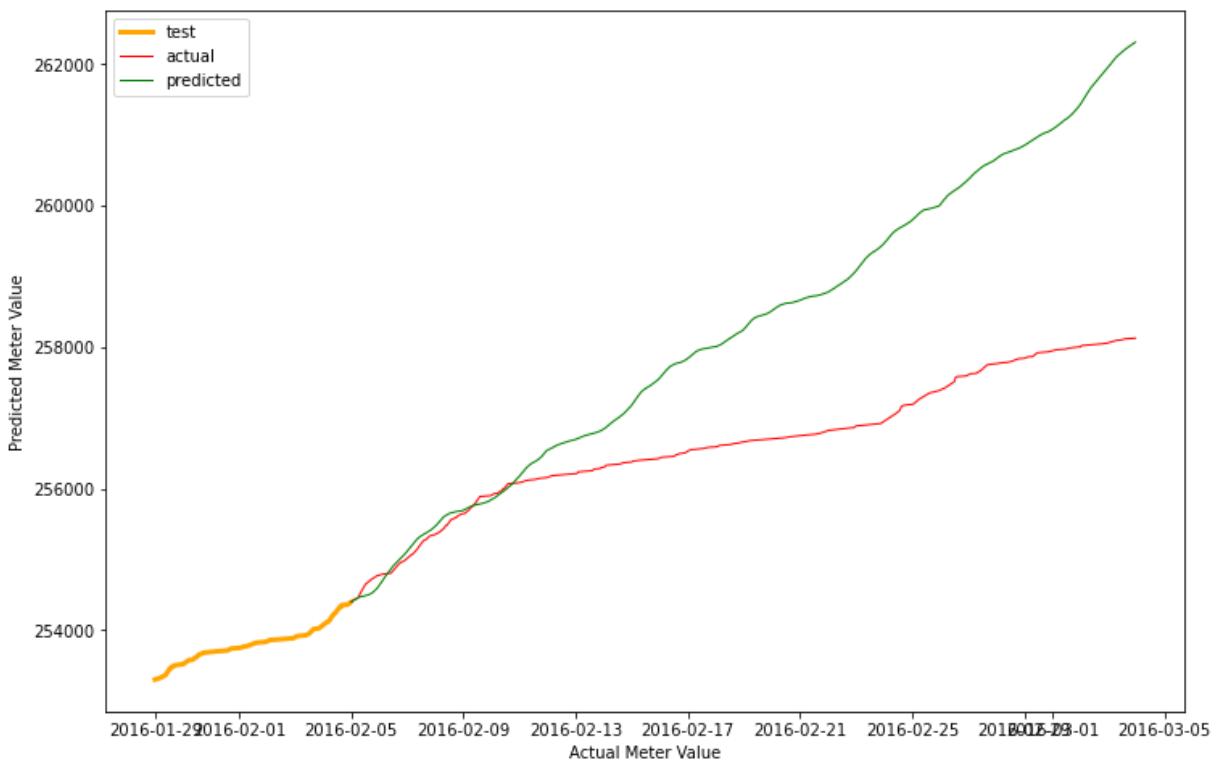
Household 4352.0  
Testing MSE: 3444280.899278597



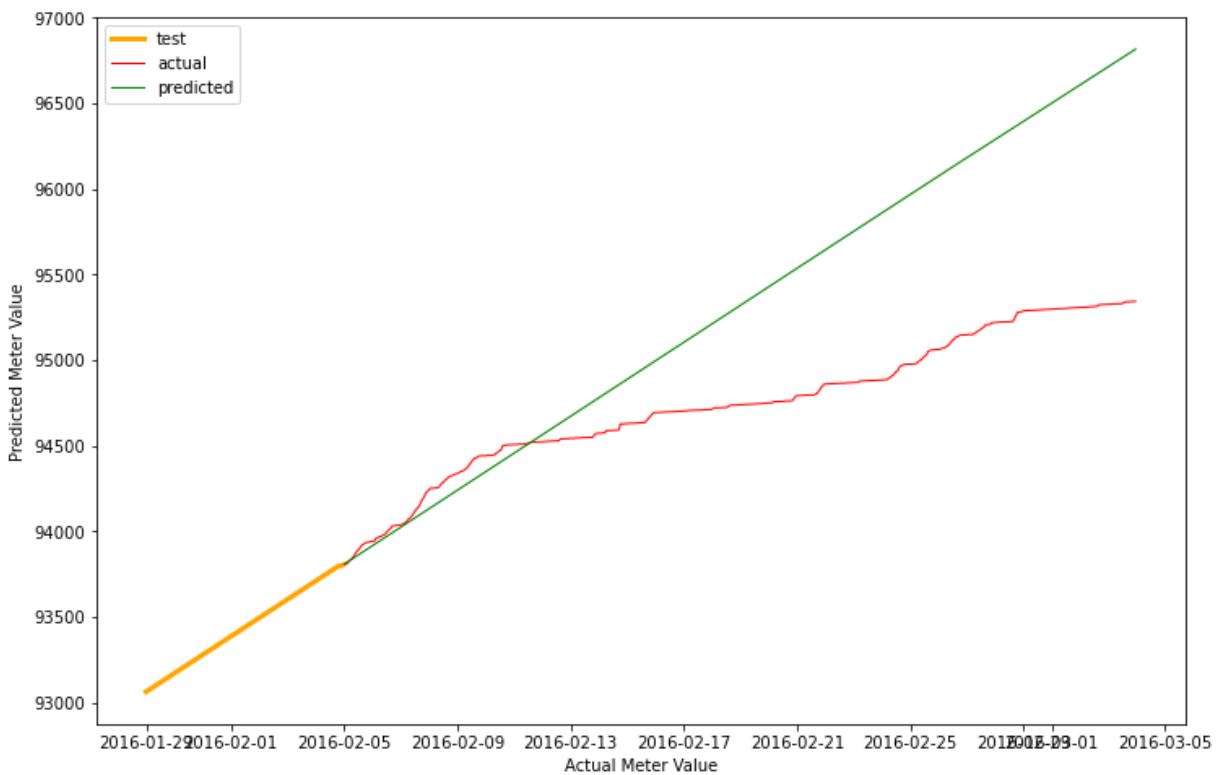
Household 4356.0  
Testing MSE: 2936544.353635128



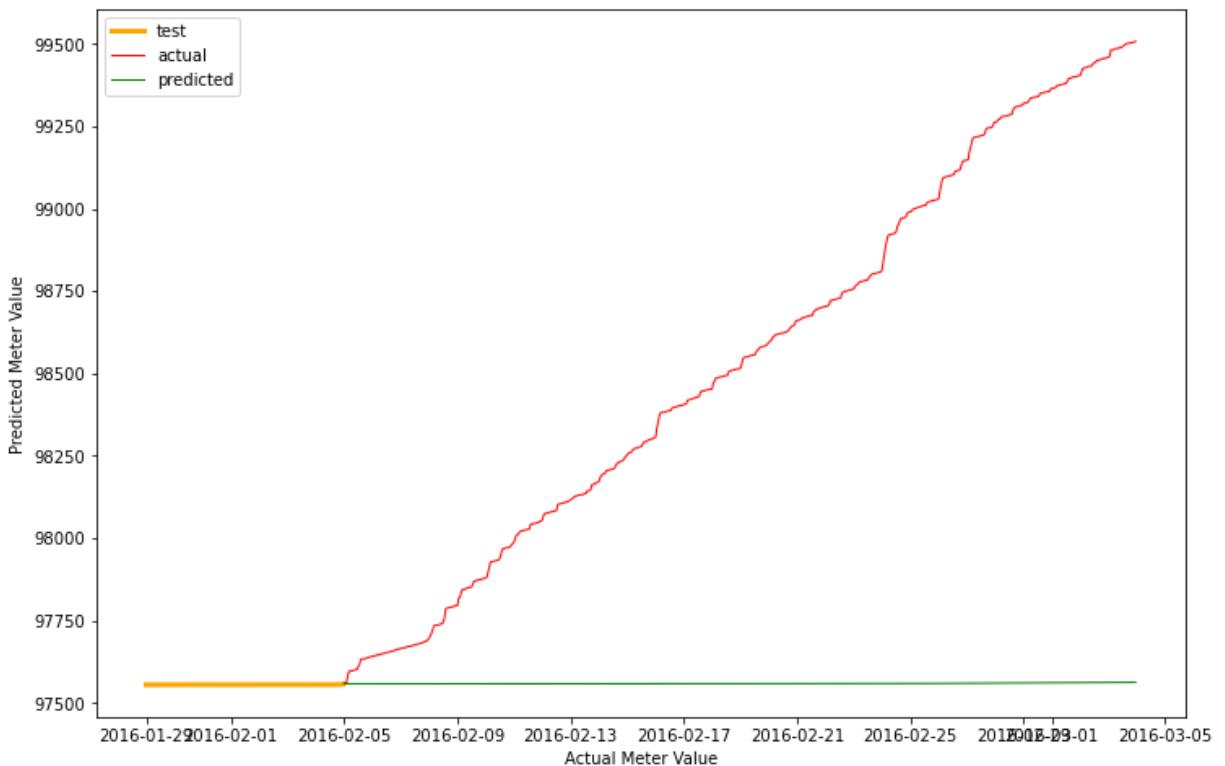
Household 4373.0  
Testing MSE: 4120443.1656985255



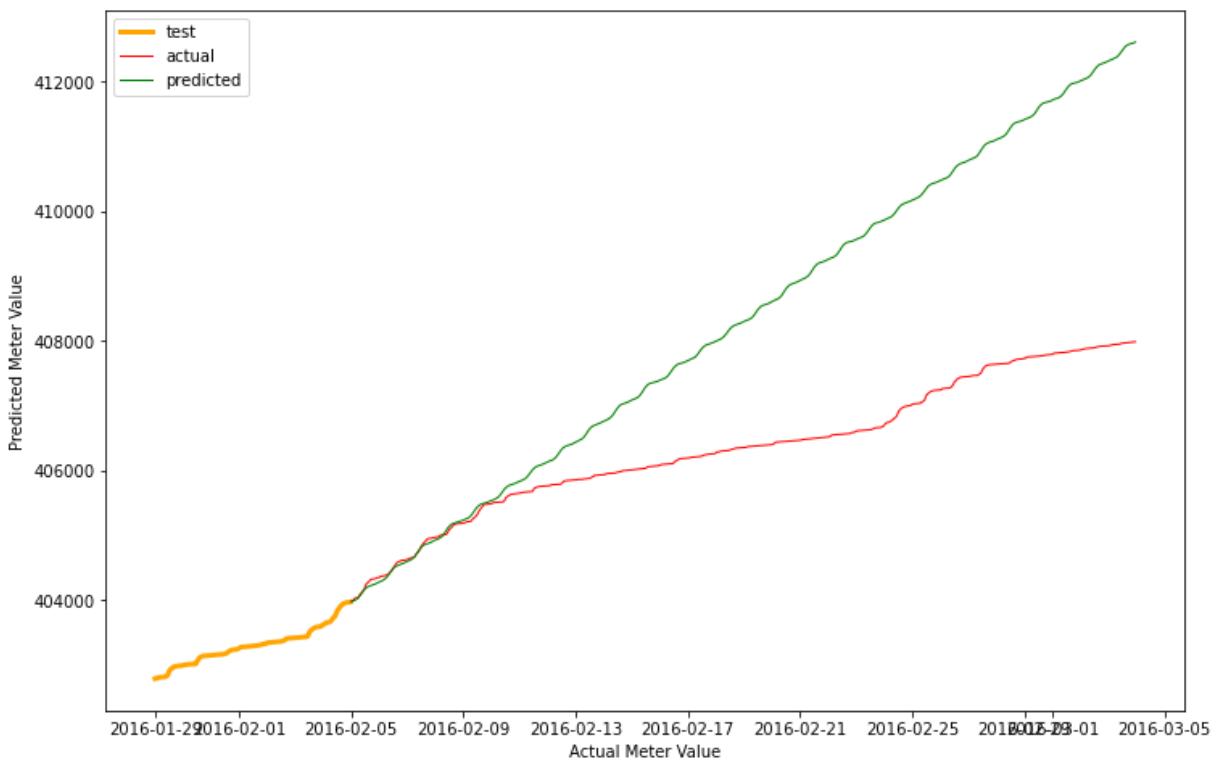
Household 4421.0  
Testing MSE: 569975.4846230461



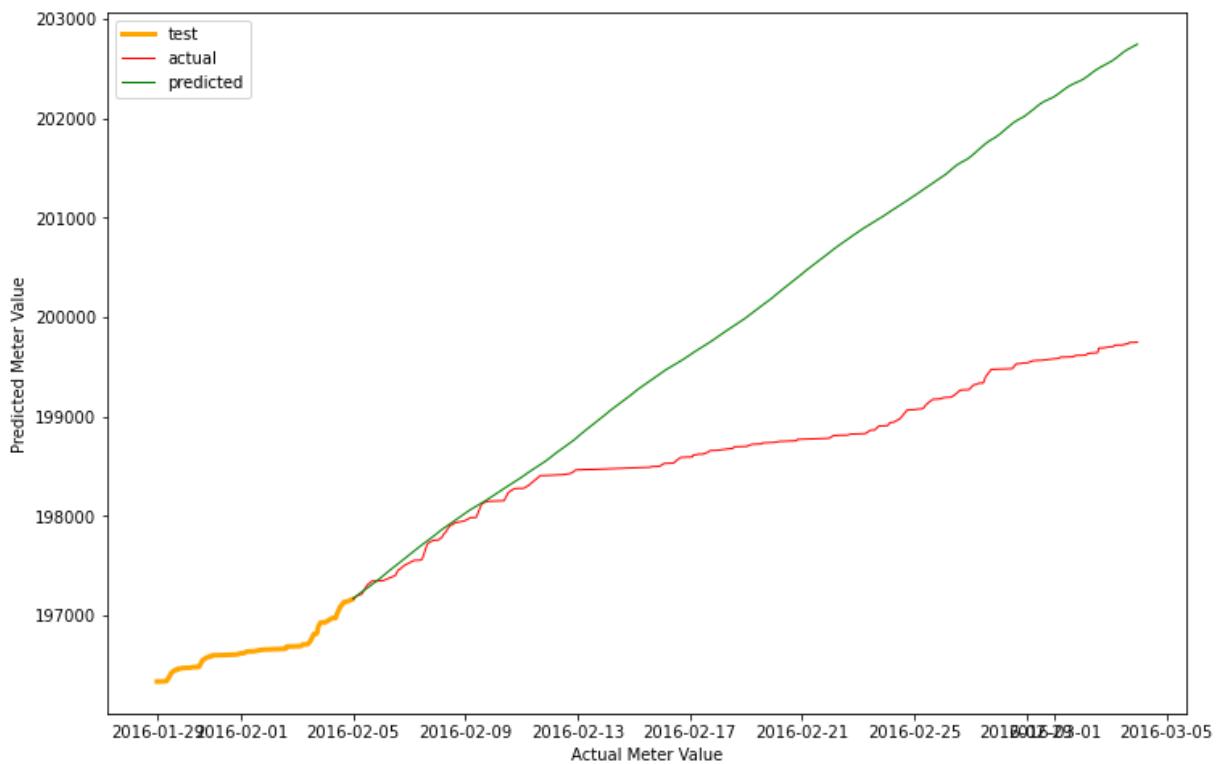
Household 4447.0  
Testing MSE: 1306112.350388431



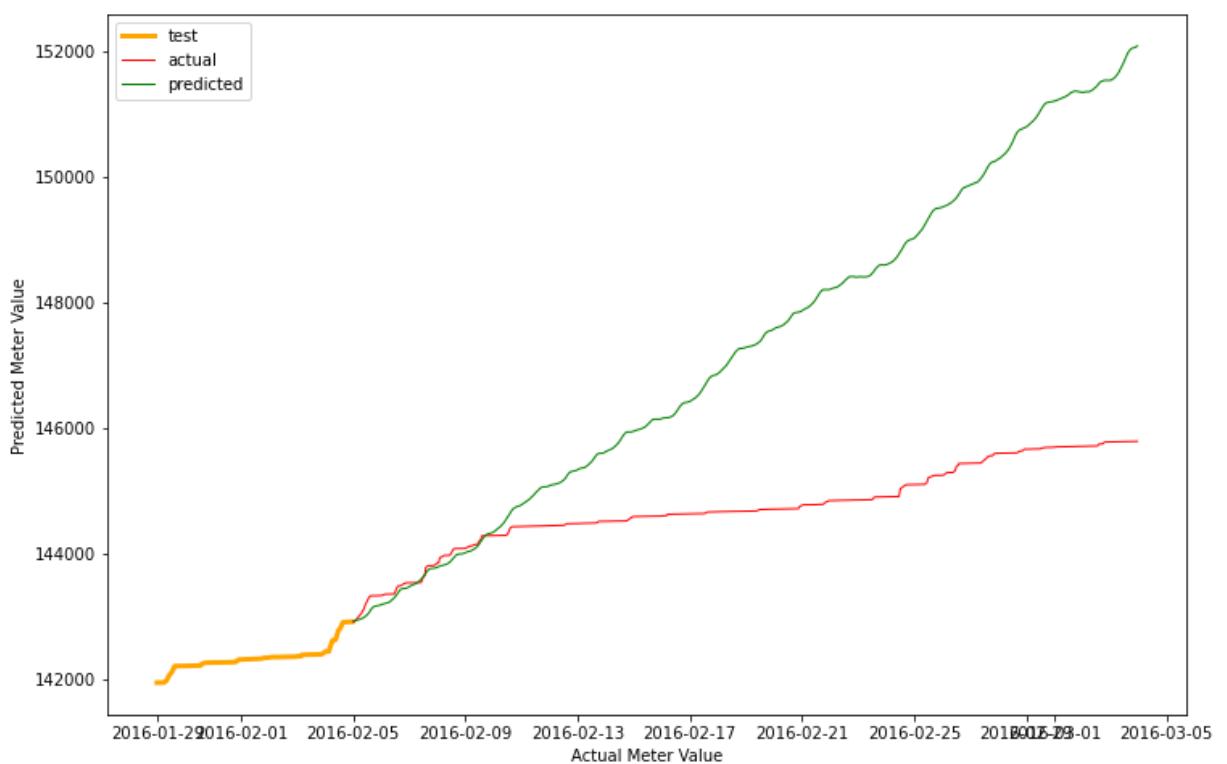
Household 4514.0  
Testing MSE: 6078152.259947278



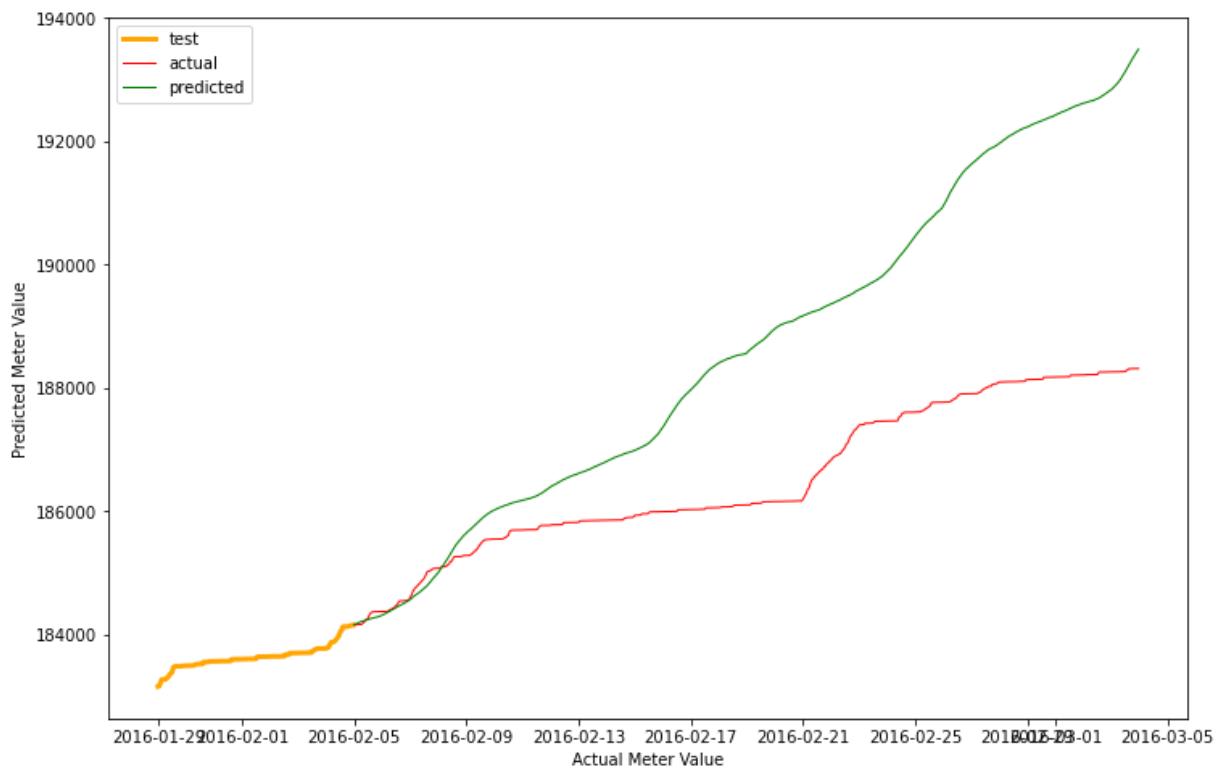
Household 4732.0  
Testing MSE: 2748255.784014062



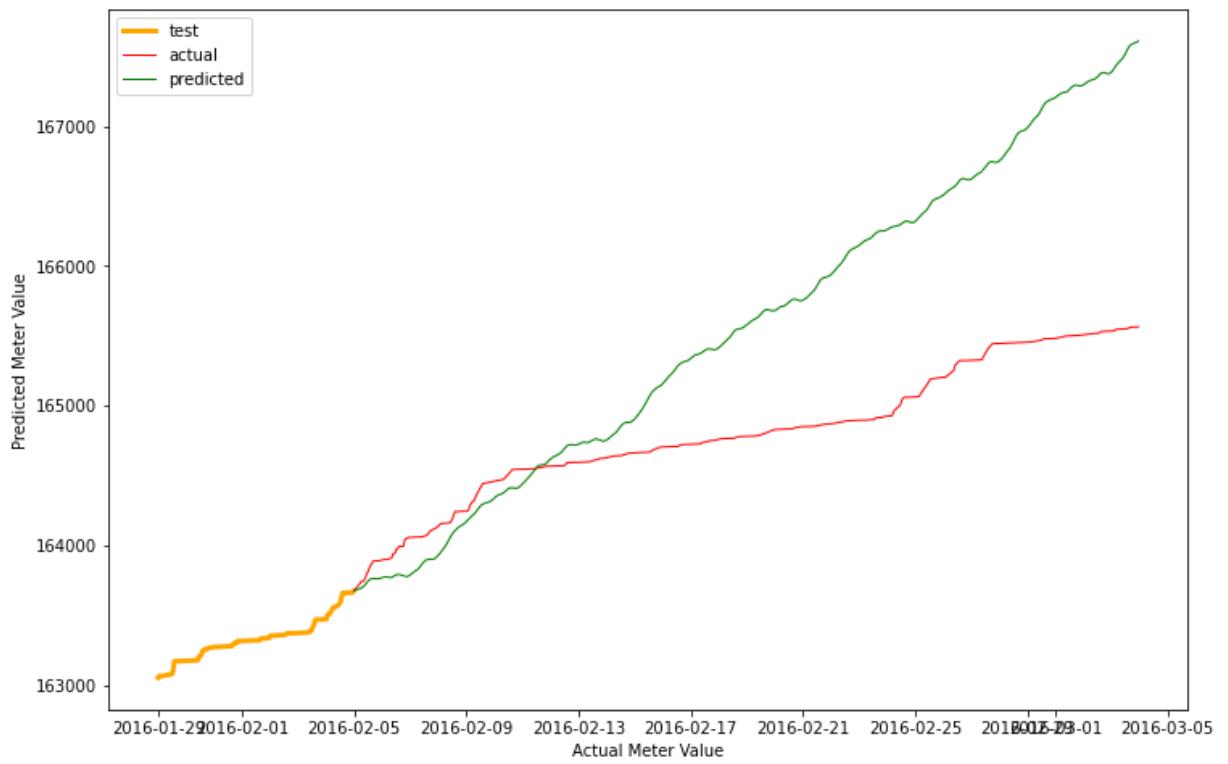
Household 4767.0  
Testing MSE: 10502097.480513493



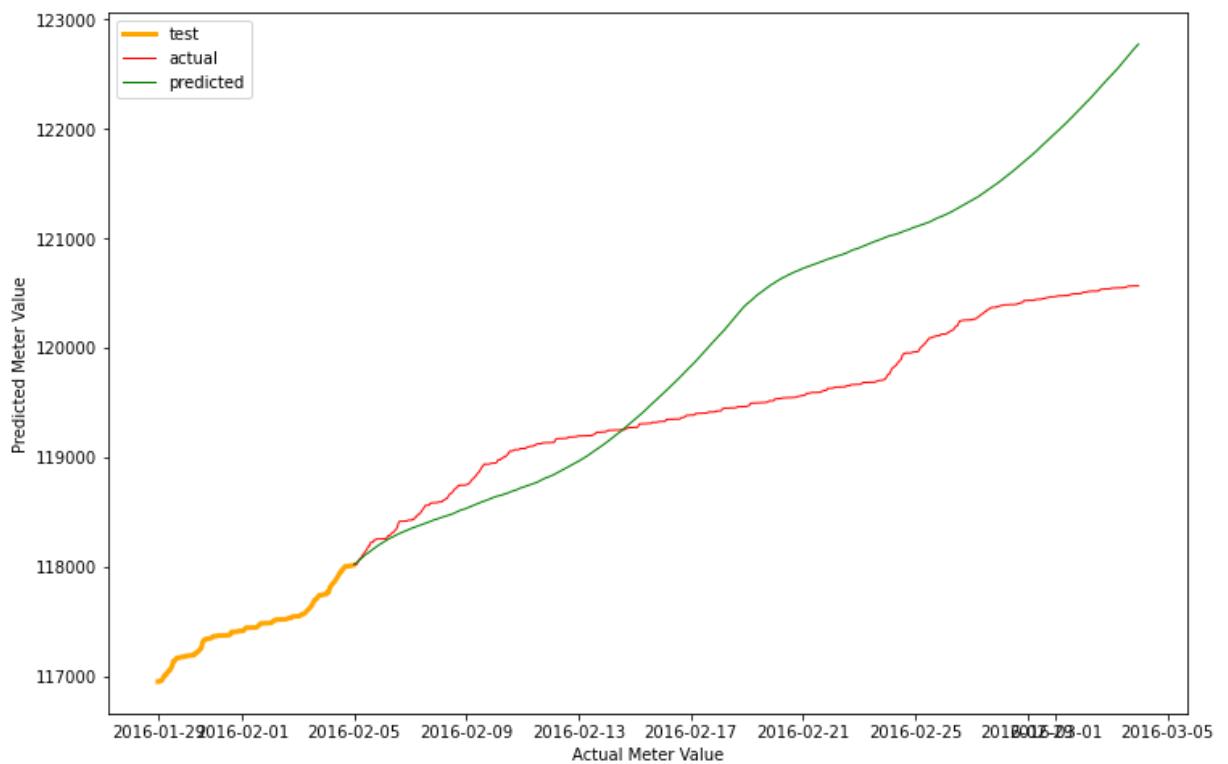
Household 4998.0  
Testing MSE: 6774184.223382093



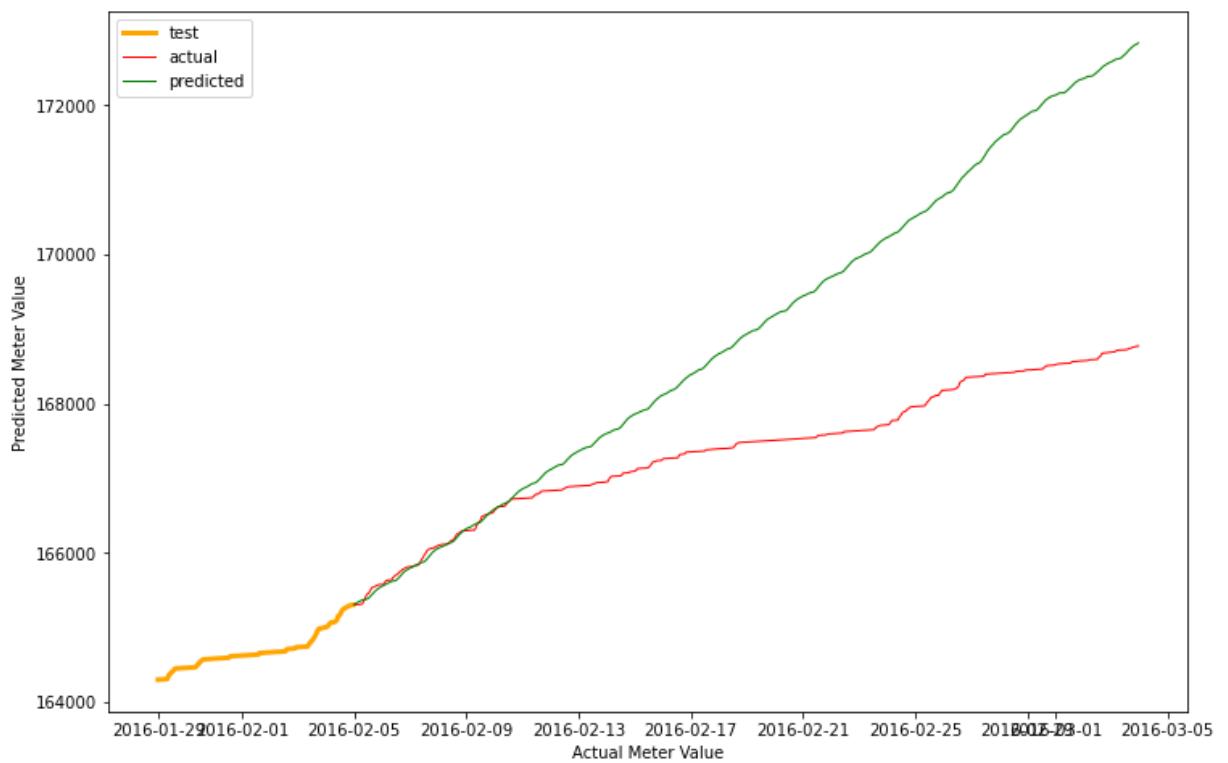
Household 5129.0  
Testing MSE: 1033397.9276699612



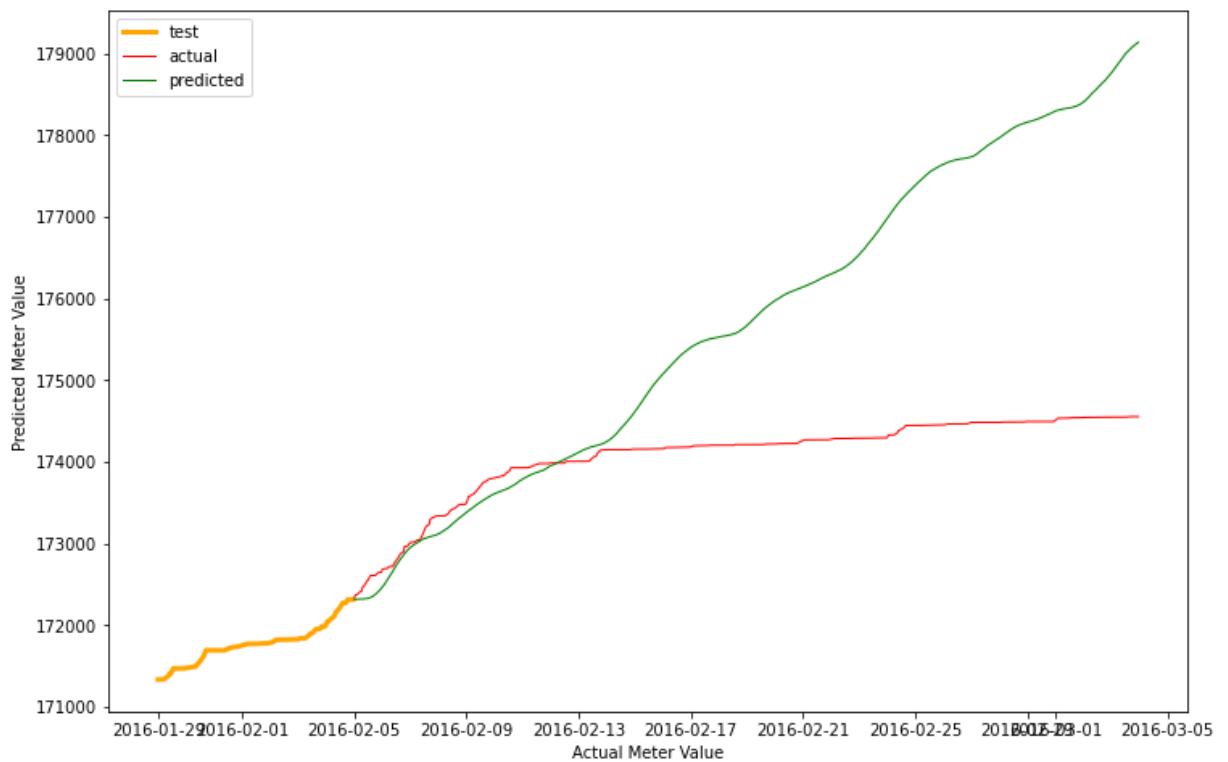
Household 5131.0  
Testing MSE: 959776.3819703602



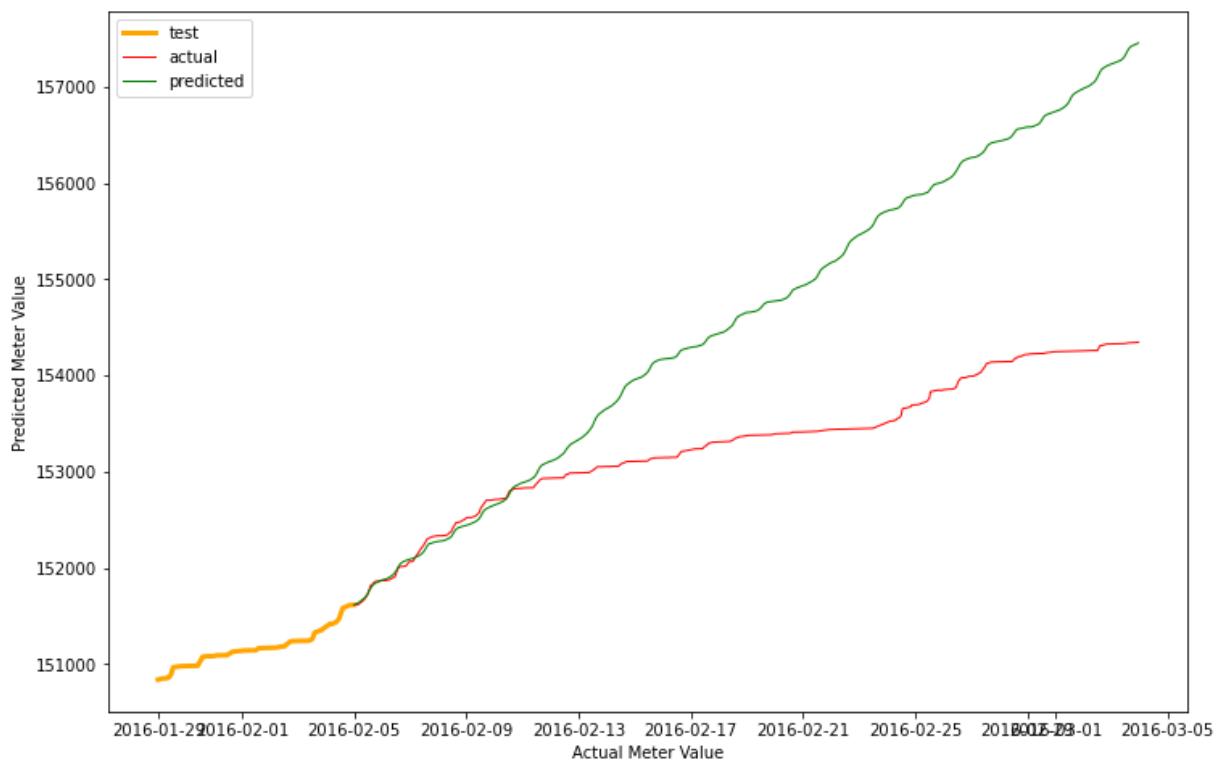
Household 5193.0  
Testing MSE: 4359440.579172407



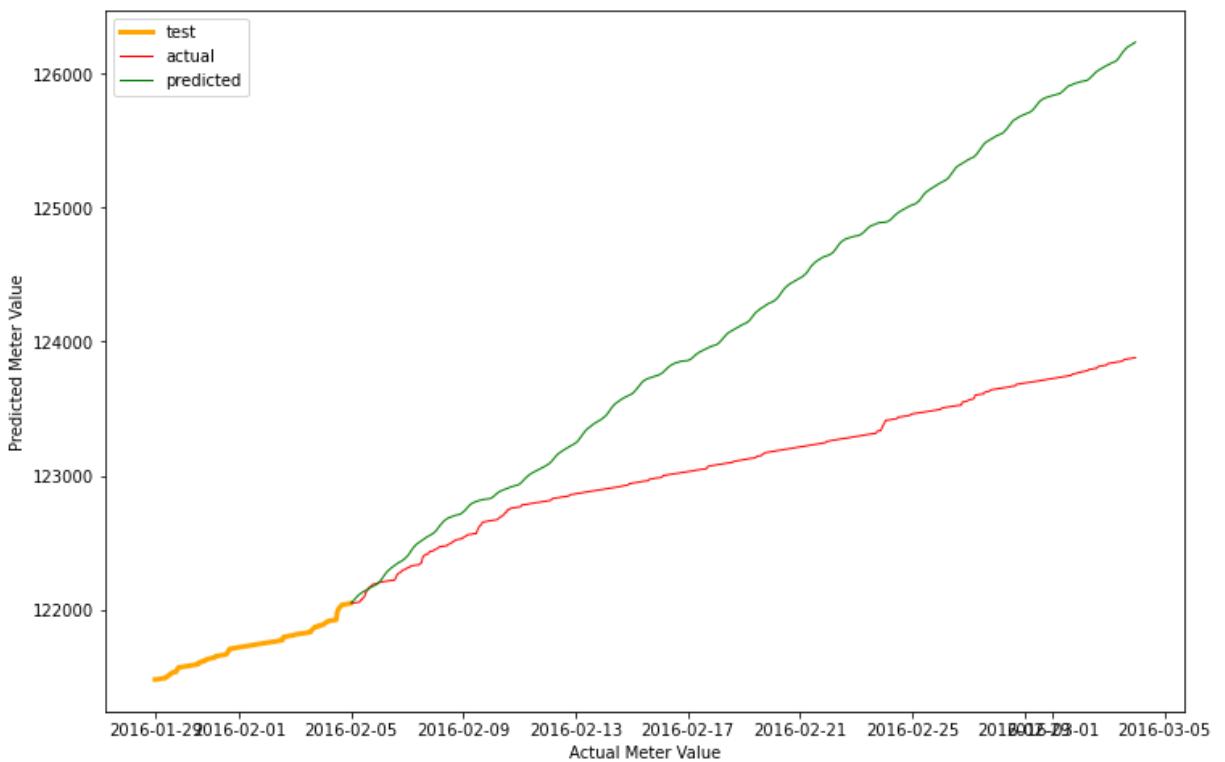
Household 5275.0  
Testing MSE: 5056891.399225096



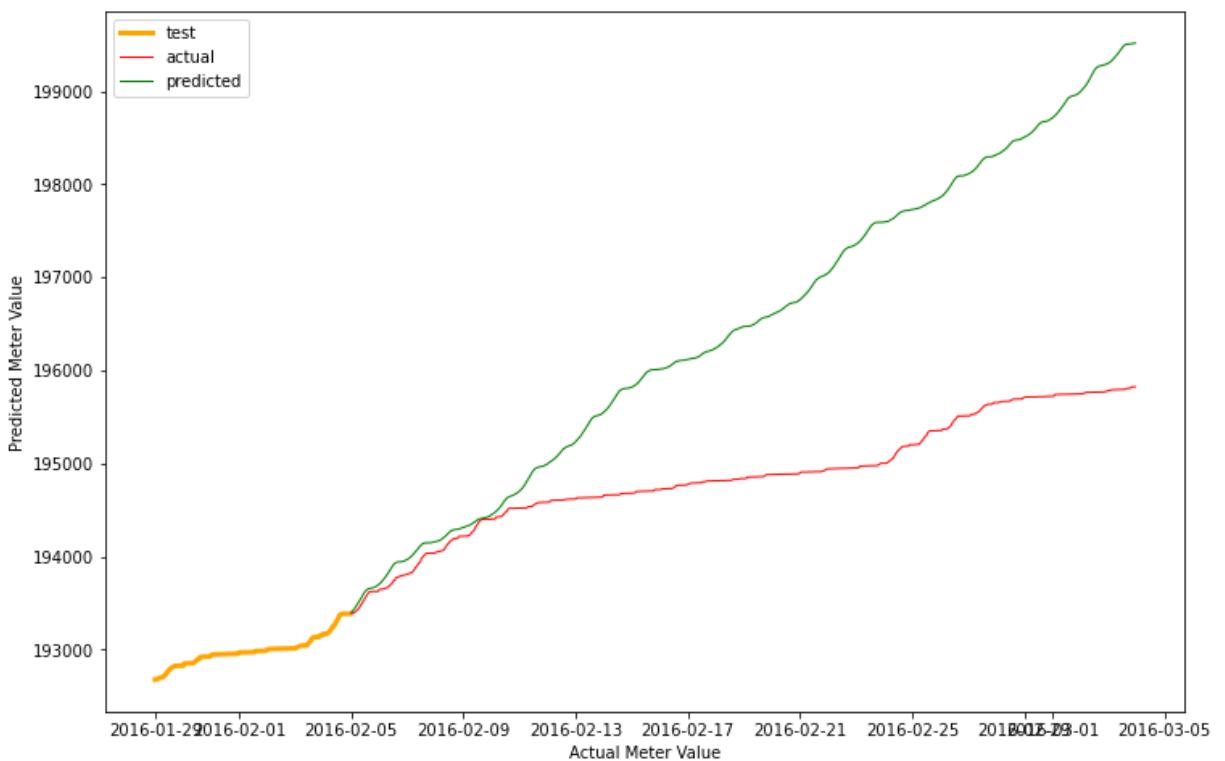
Household 5395.0  
Testing MSE: 2659467.4414885896



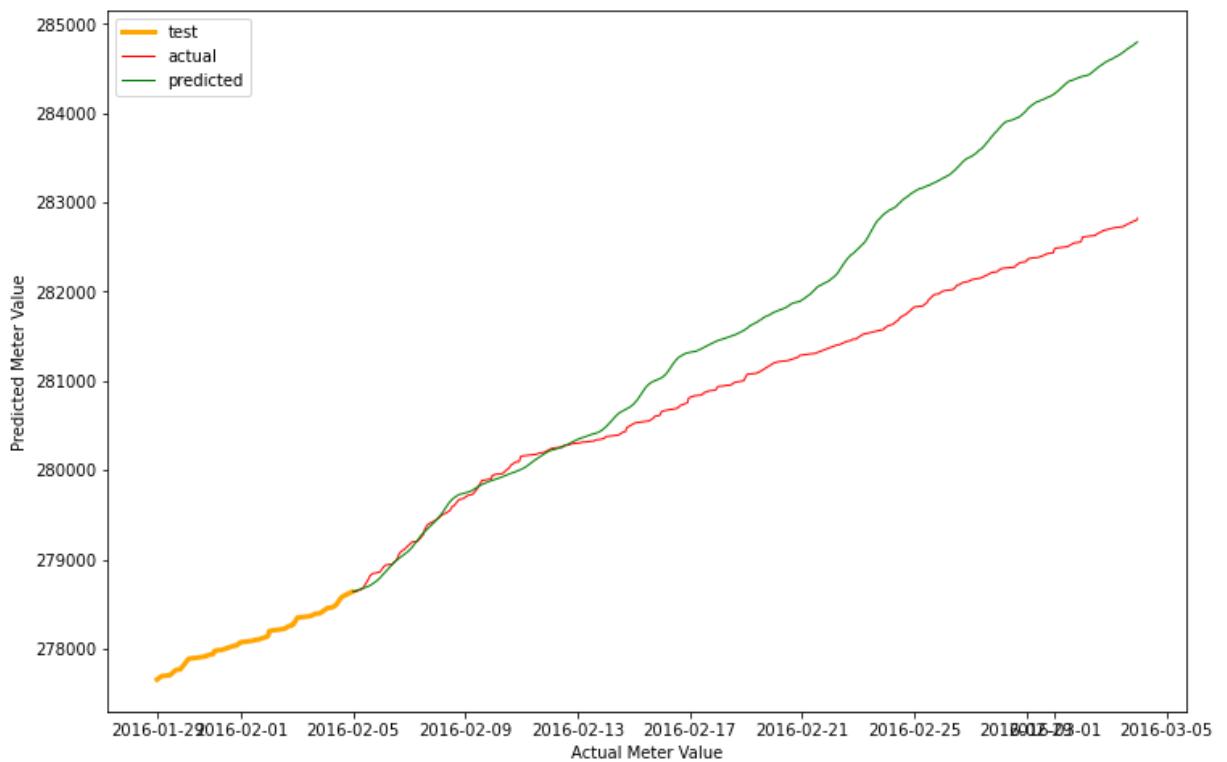
Household 5403.0  
Testing MSE: 1672918.627989996



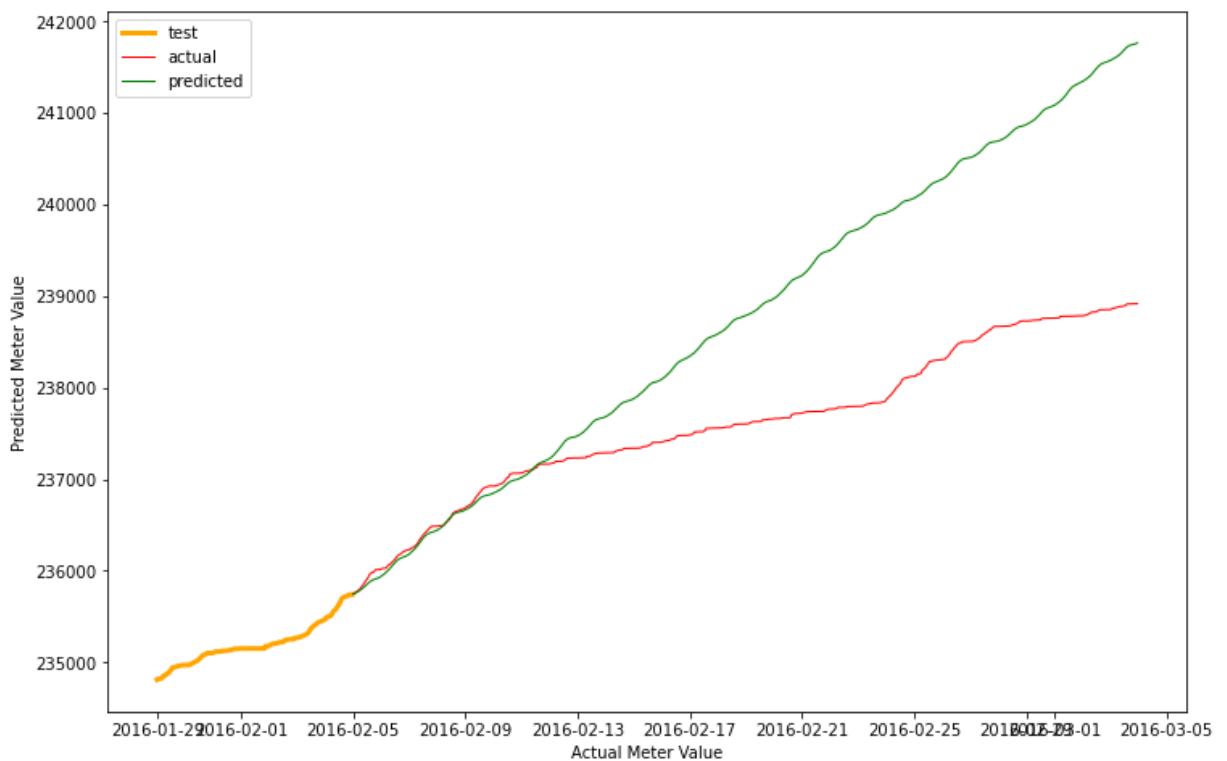
Household 5439.0  
Testing MSE: 3864596.178168855



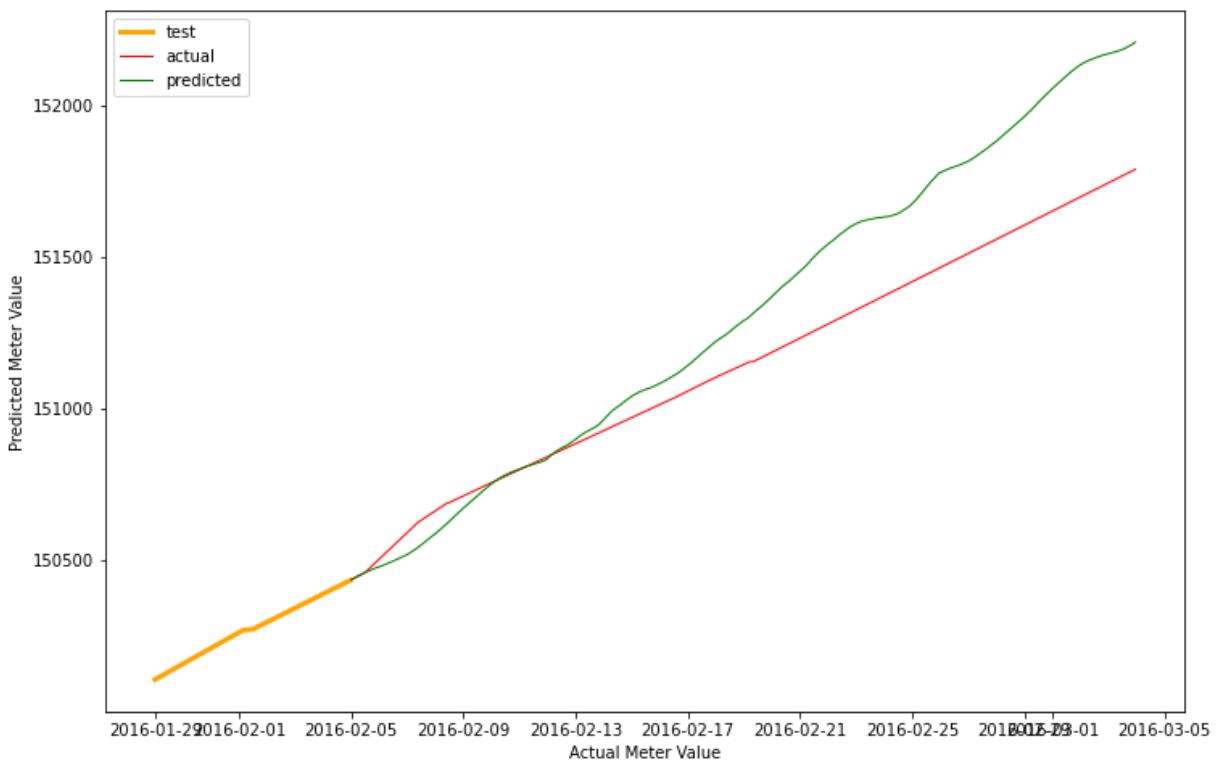
Household 5484.0  
Testing MSE: 989497.8527130575



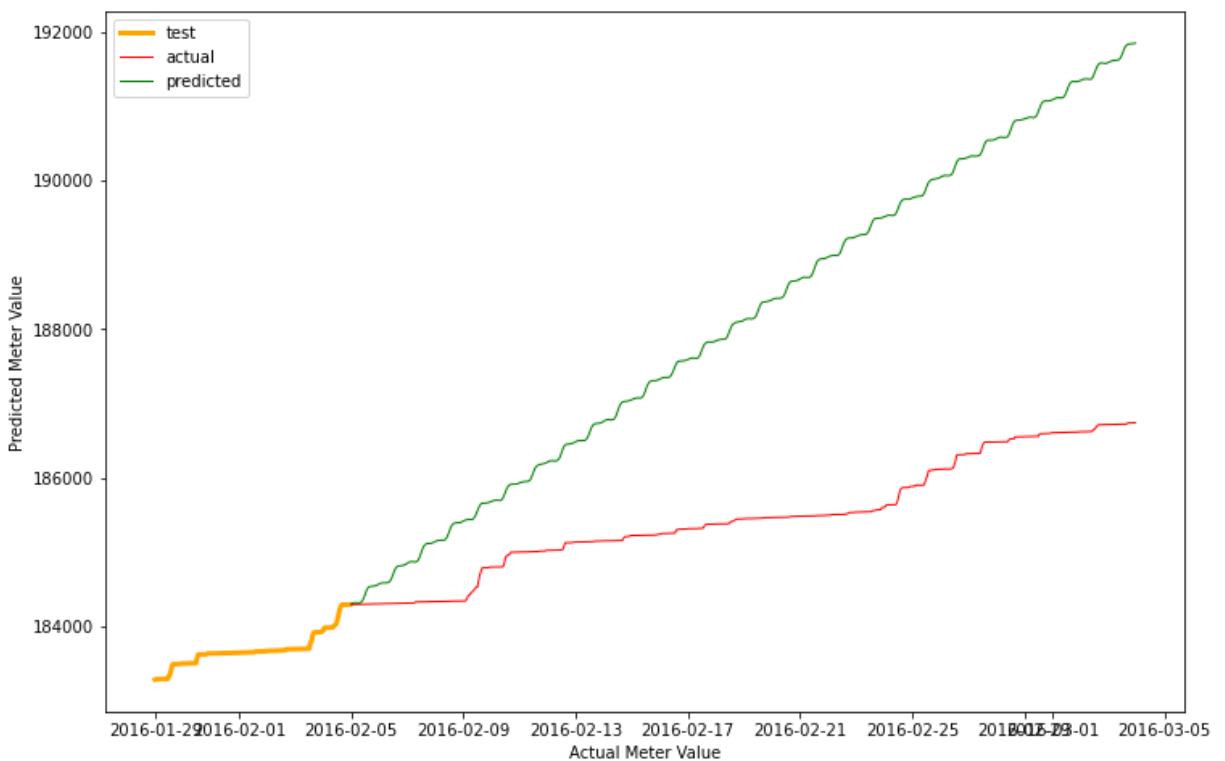
Household 5636.0  
Testing MSE: 2251091.58460547



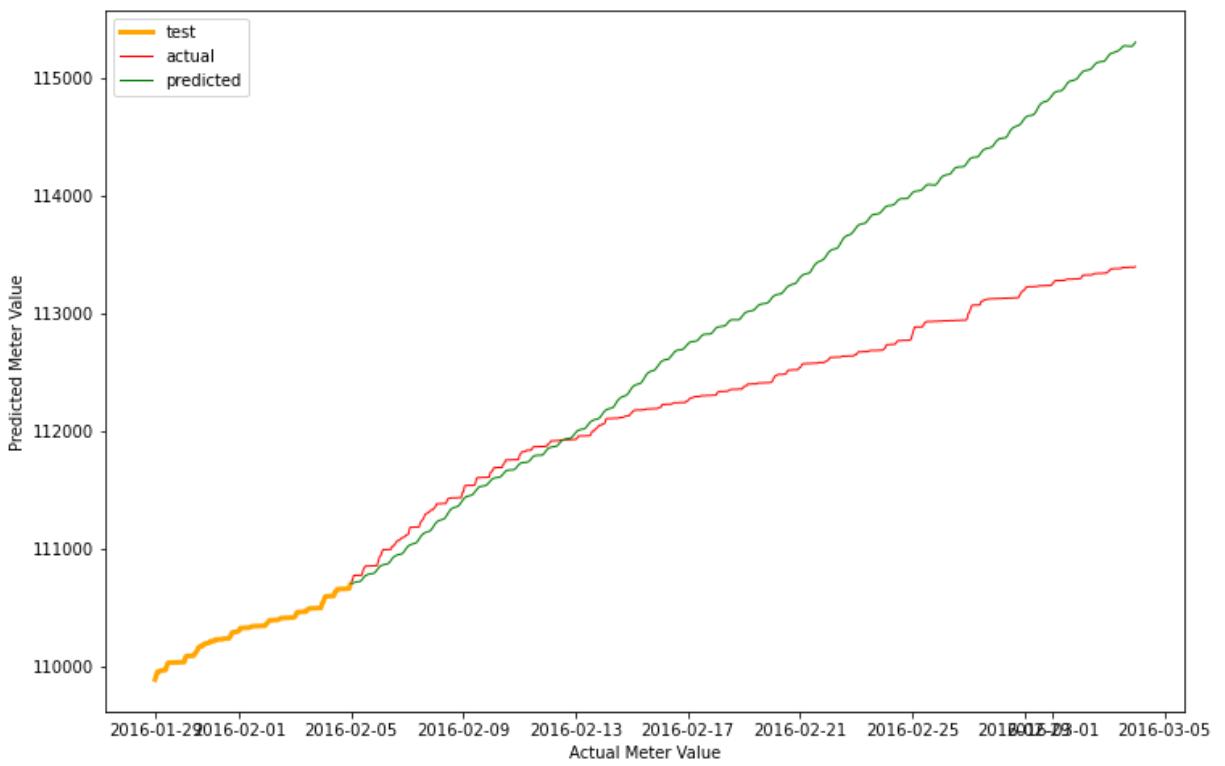
Household 5658.0  
Testing MSE: 52671.19024579045



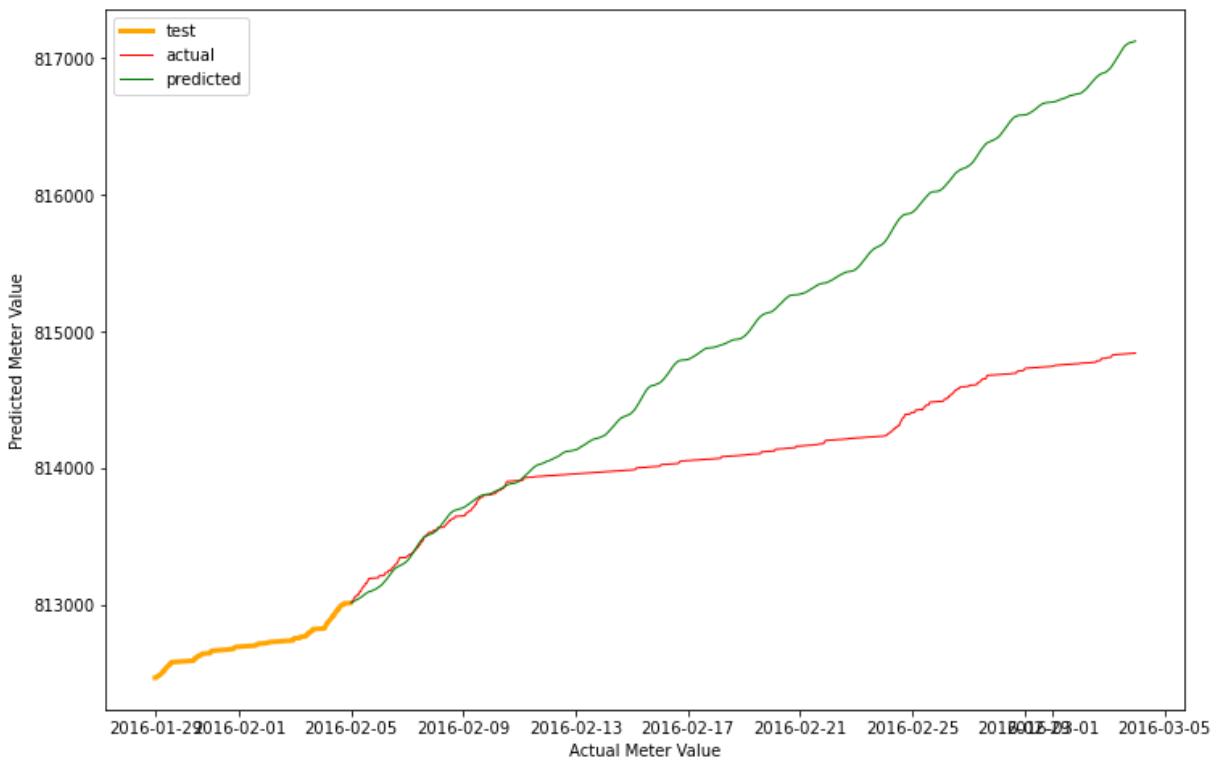
Household 5785.0  
Testing MSE: 9113270.96671929



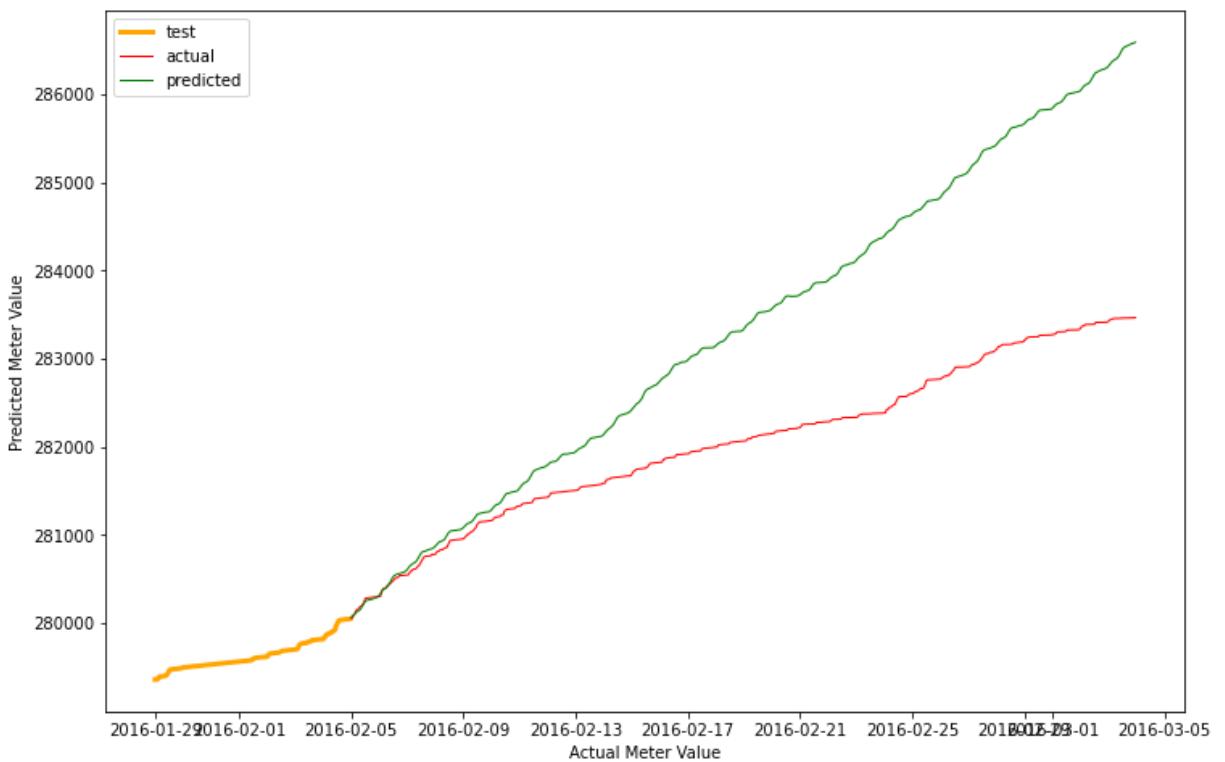
Household 5810.0  
Testing MSE: 872863.2632553463



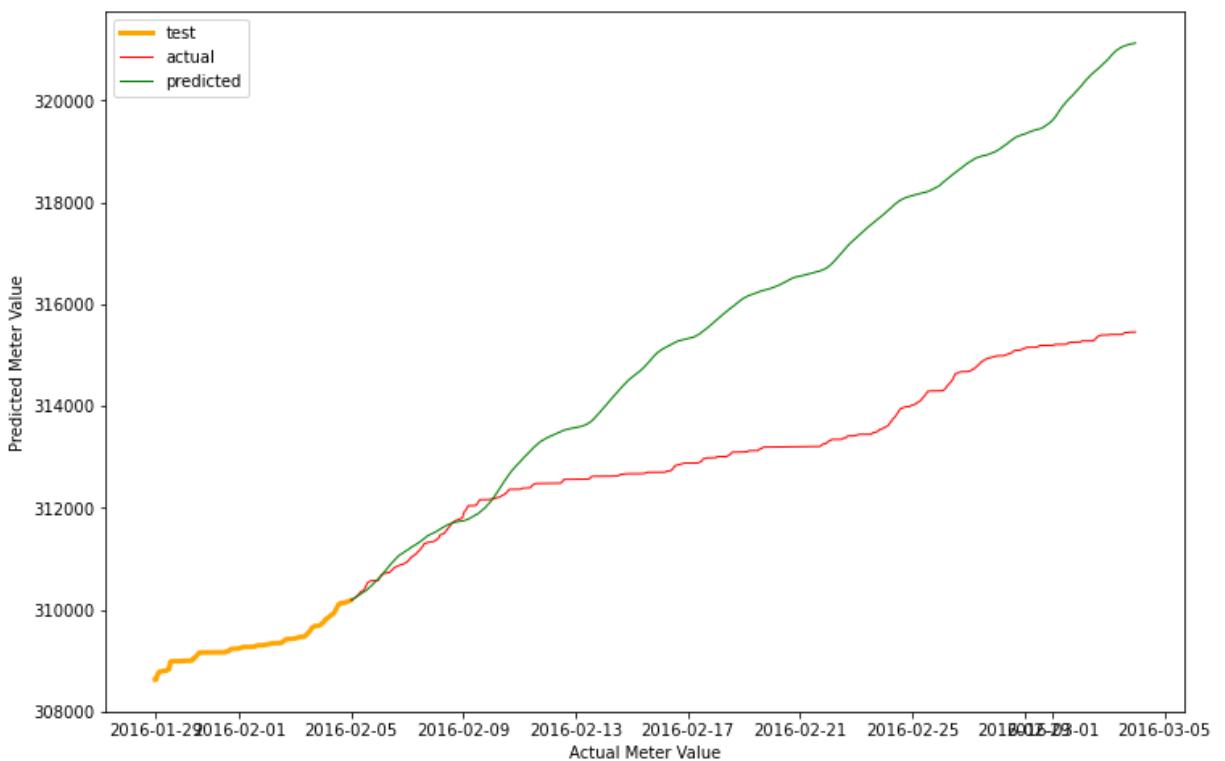
Household 5814.0  
Testing MSE: 1376269.0971156529



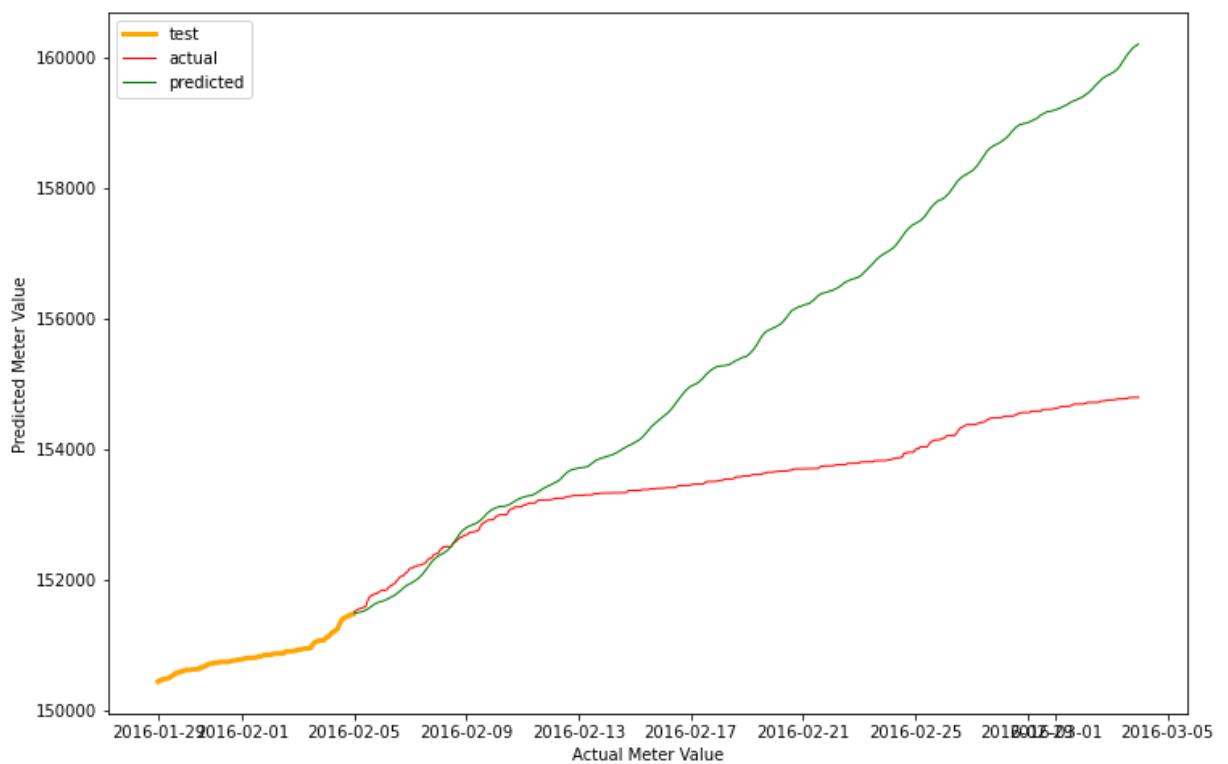
Household 5892.0  
Testing MSE: 2587643.5489085363



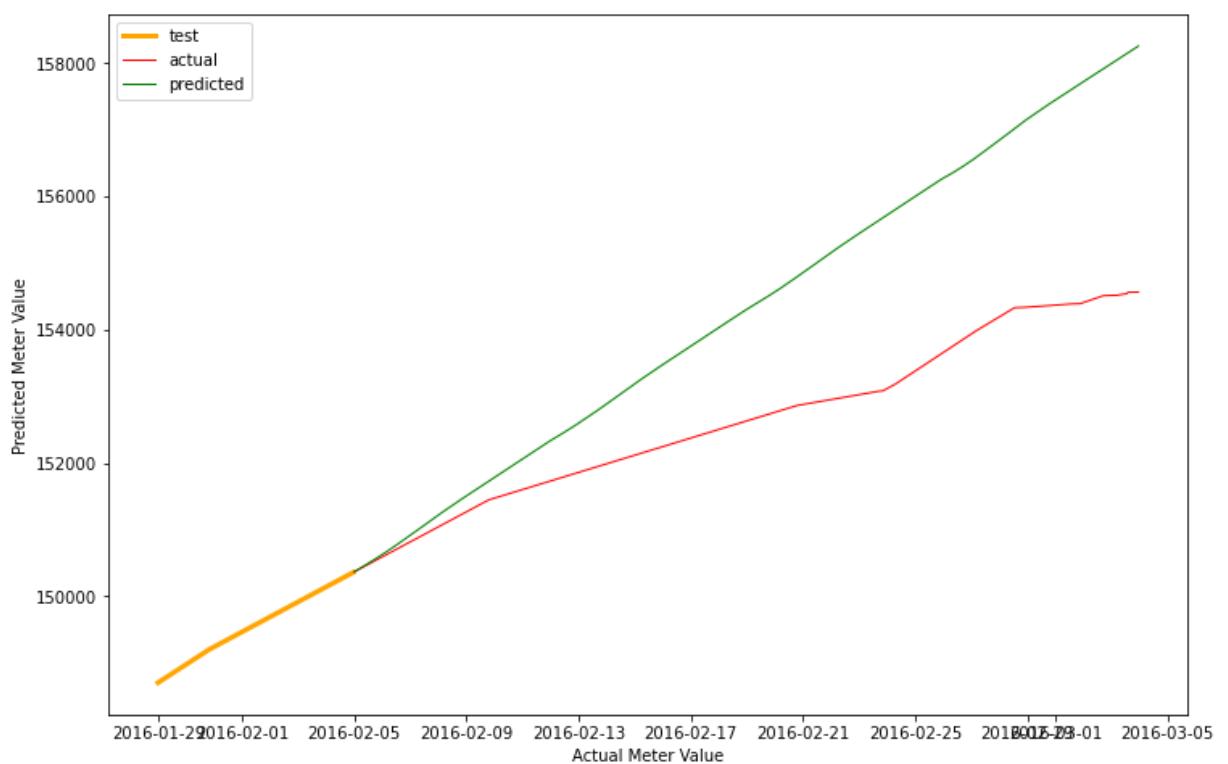
Household 5972.0  
Testing MSE: 9799572.5254798



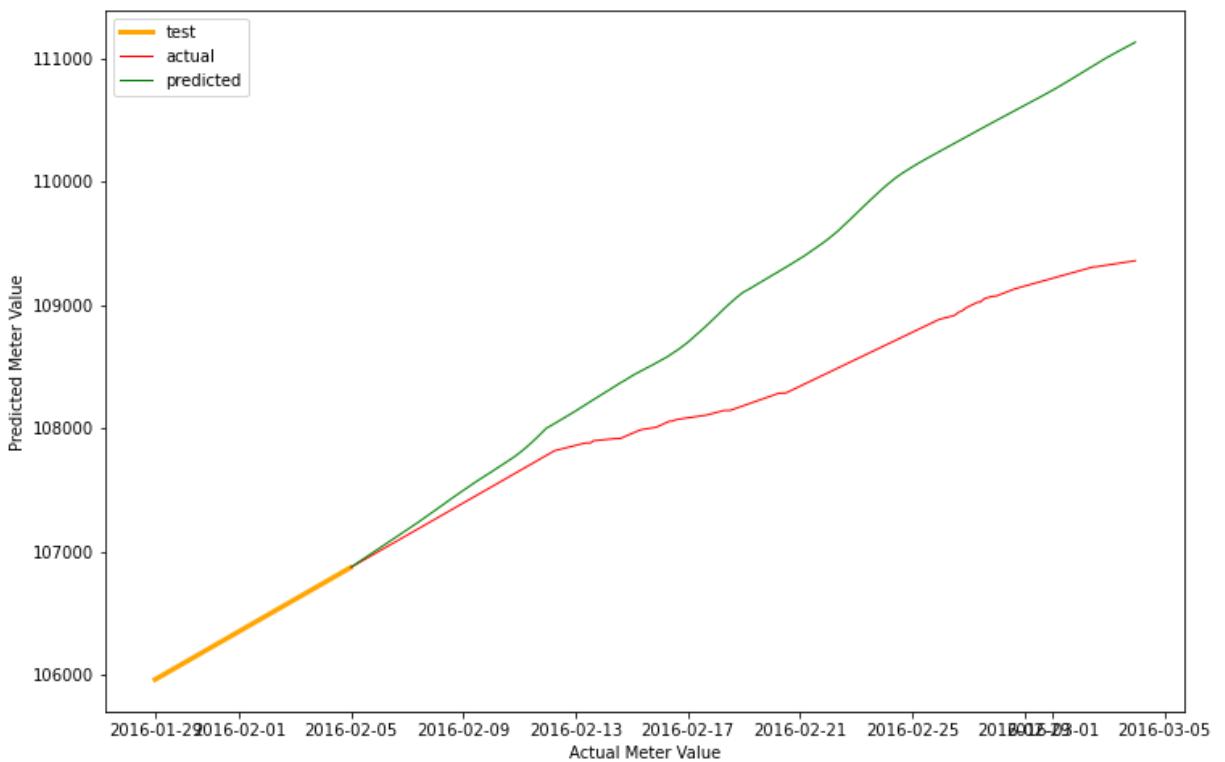
Household 6412.0  
Testing MSE: 7449953.312407139



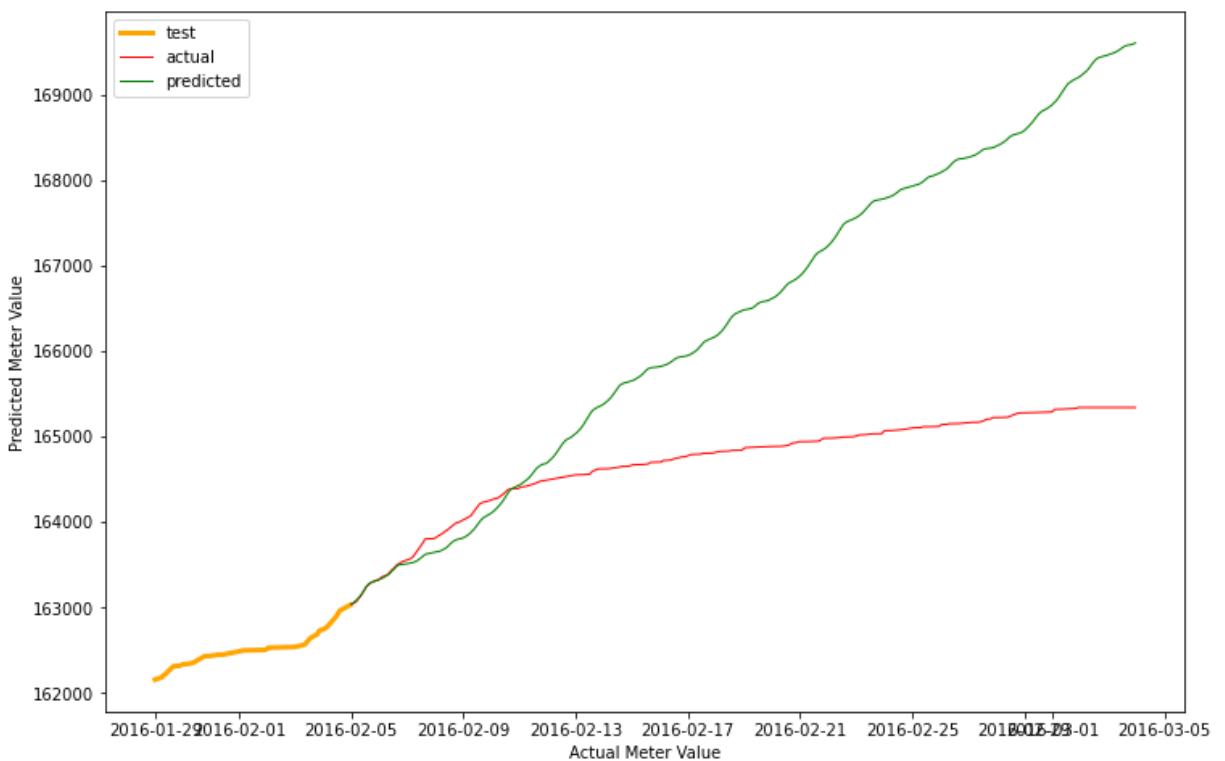
Household 6505.0  
Testing MSE: 3956323.120714846



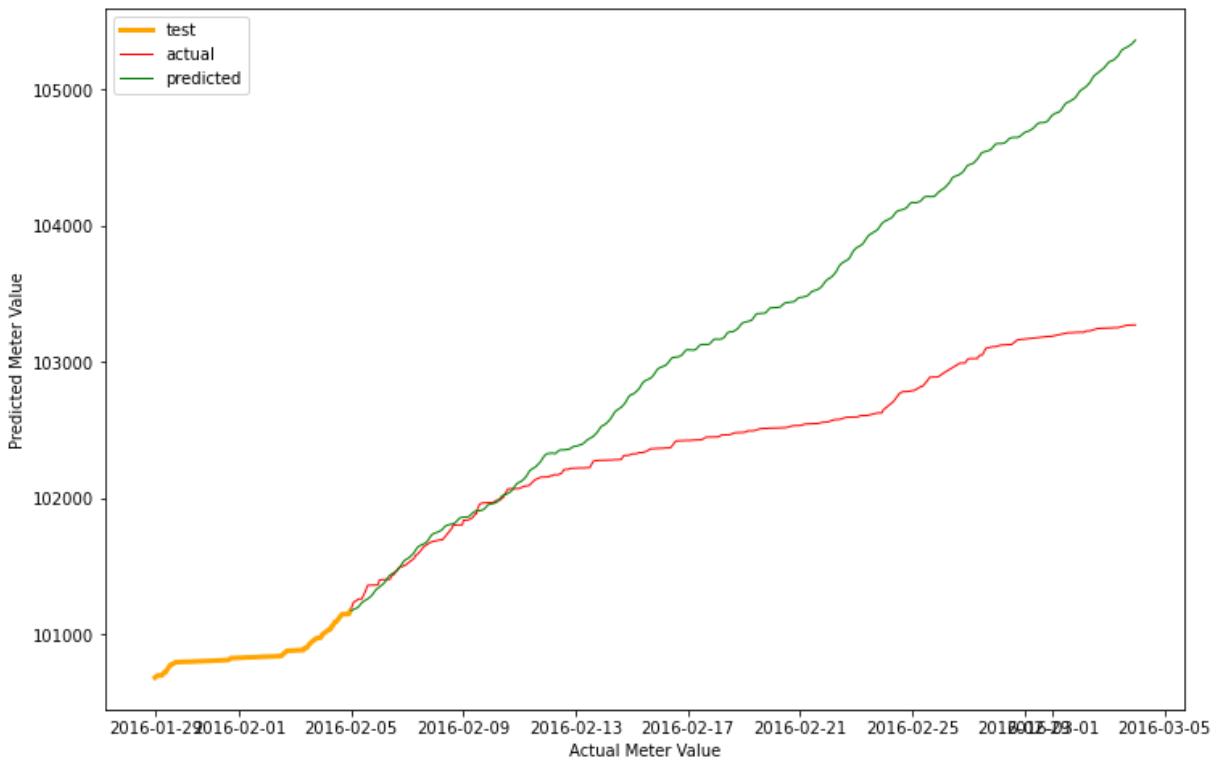
Household 6685.0  
Testing MSE: 994759.4568592506



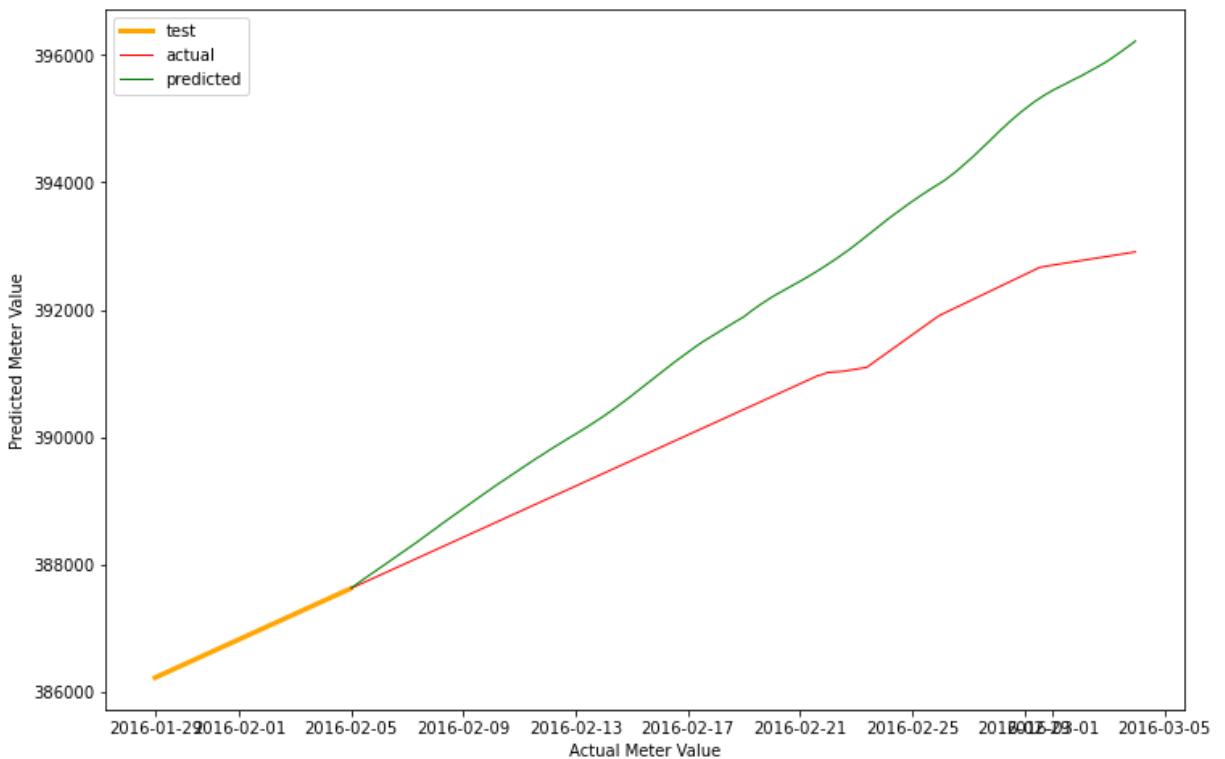
Household 6830.0  
Testing MSE: 4892929.923875945



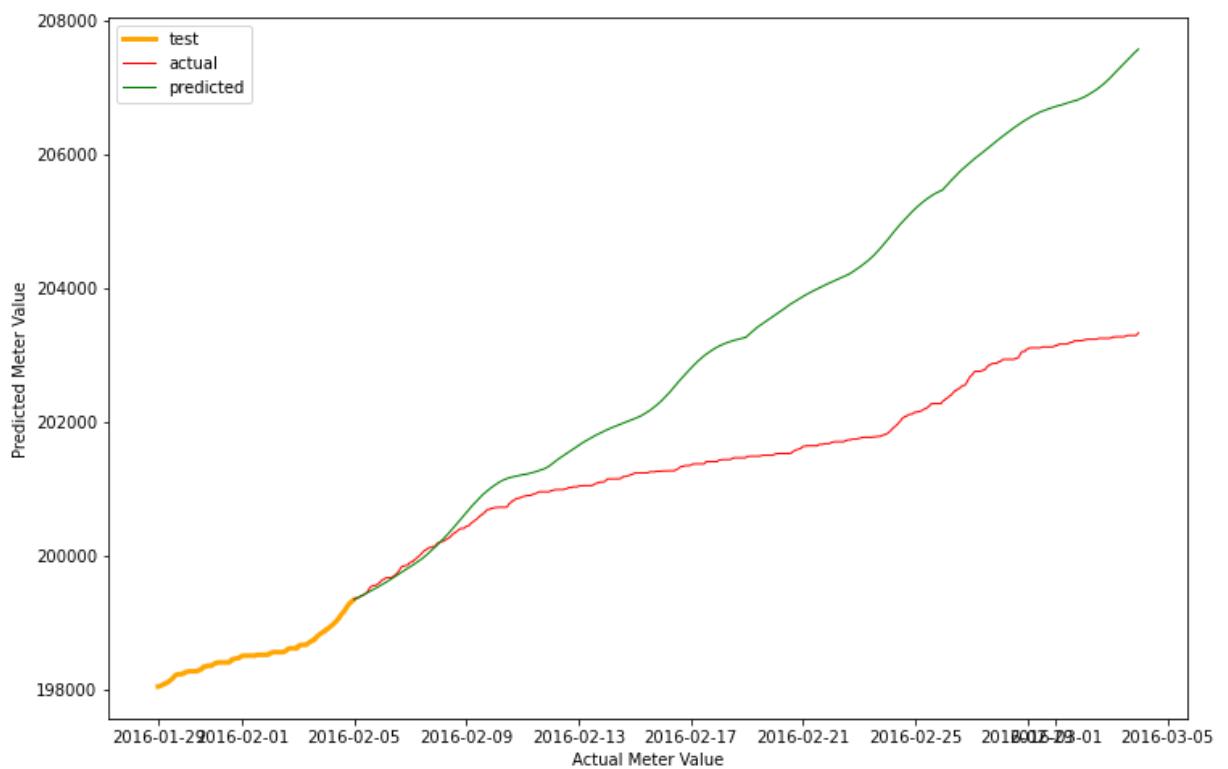
Household 6836.0  
Testing MSE: 1074064.6671210688



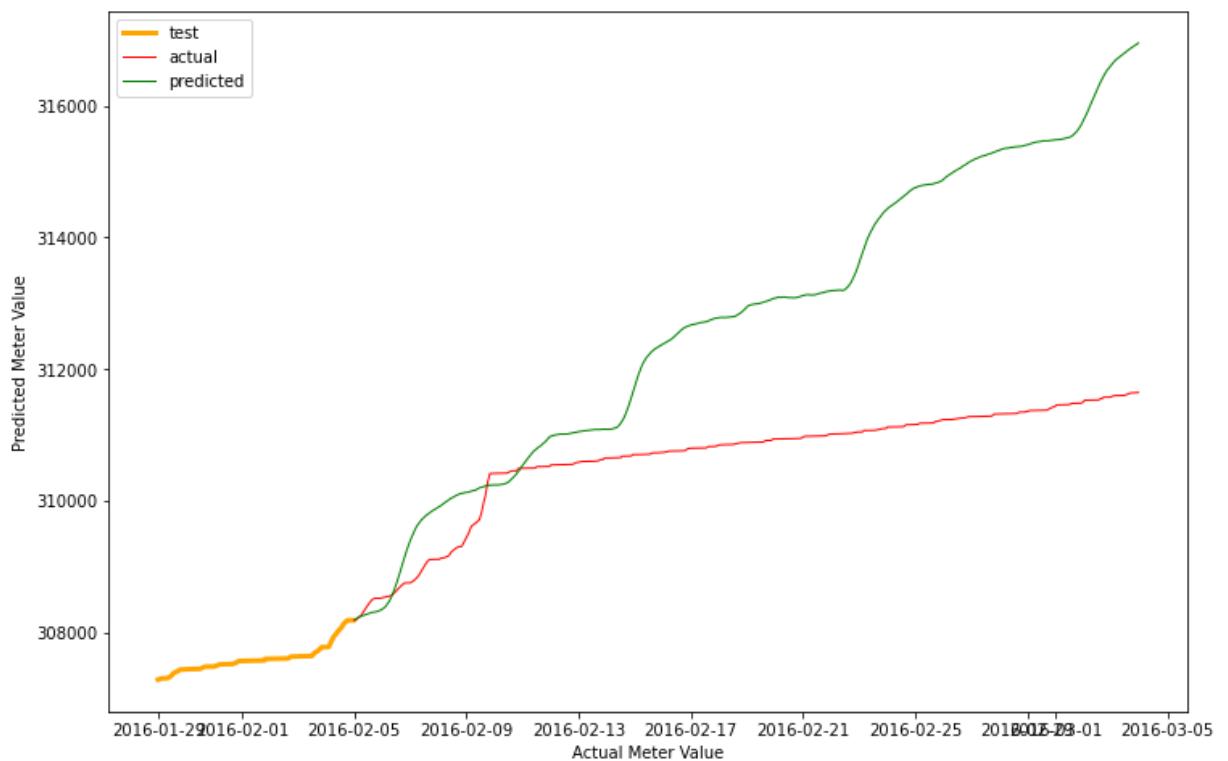
Household 6863.0  
Testing MSE: 2999198.592820969



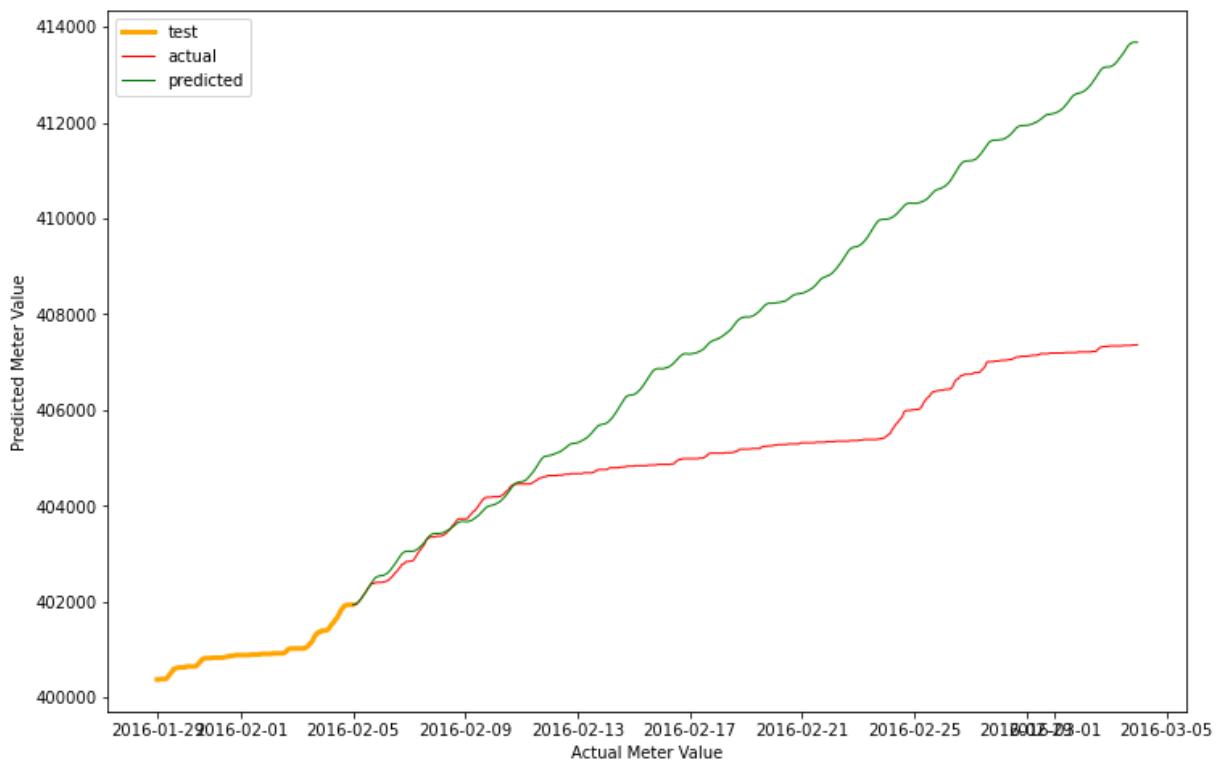
Household 6910.0  
Testing MSE: 5144336.703882126



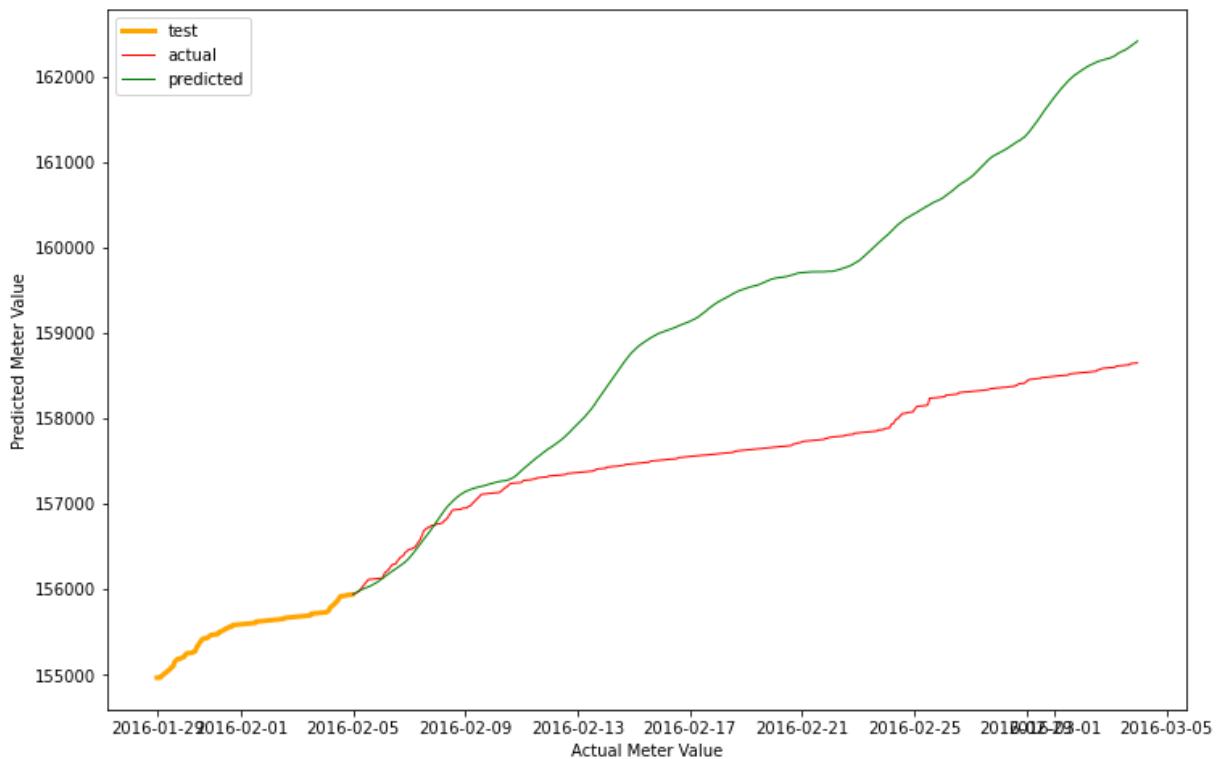
Household 7016.0  
Testing MSE: 7003434.752356711



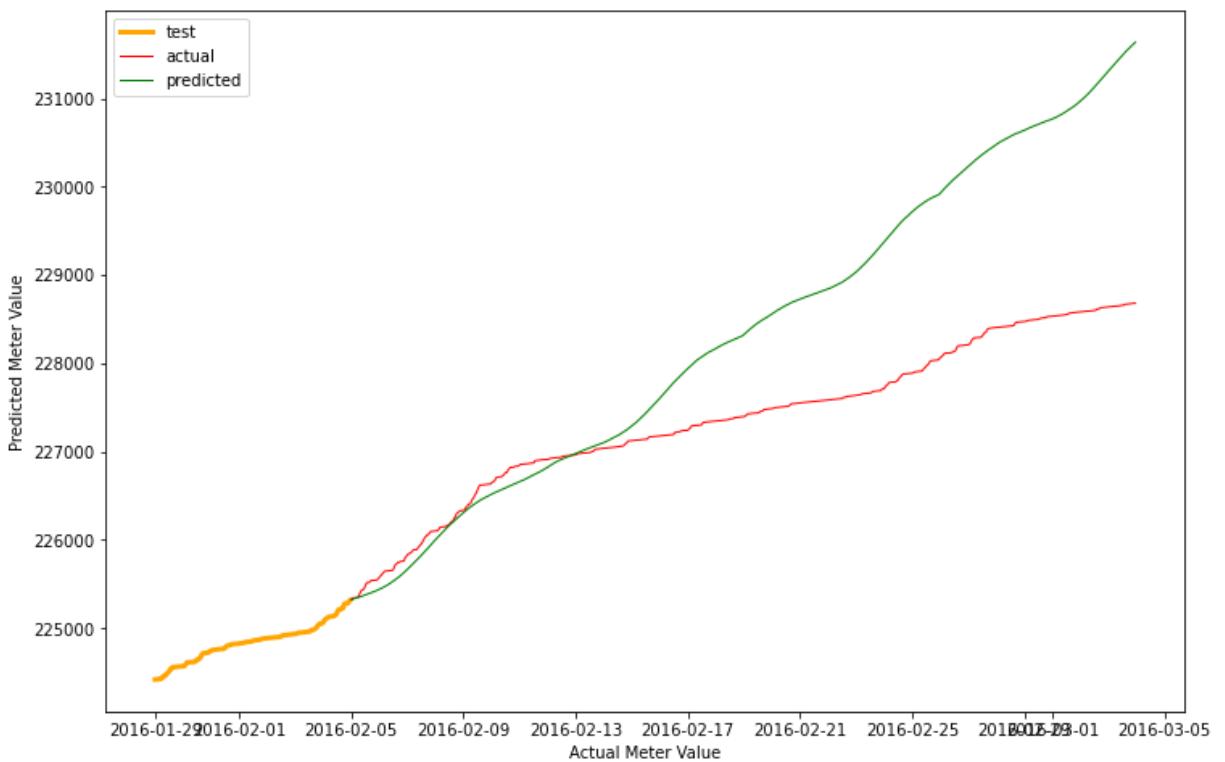
Household 7017.0  
Testing MSE: 10842558.961212158



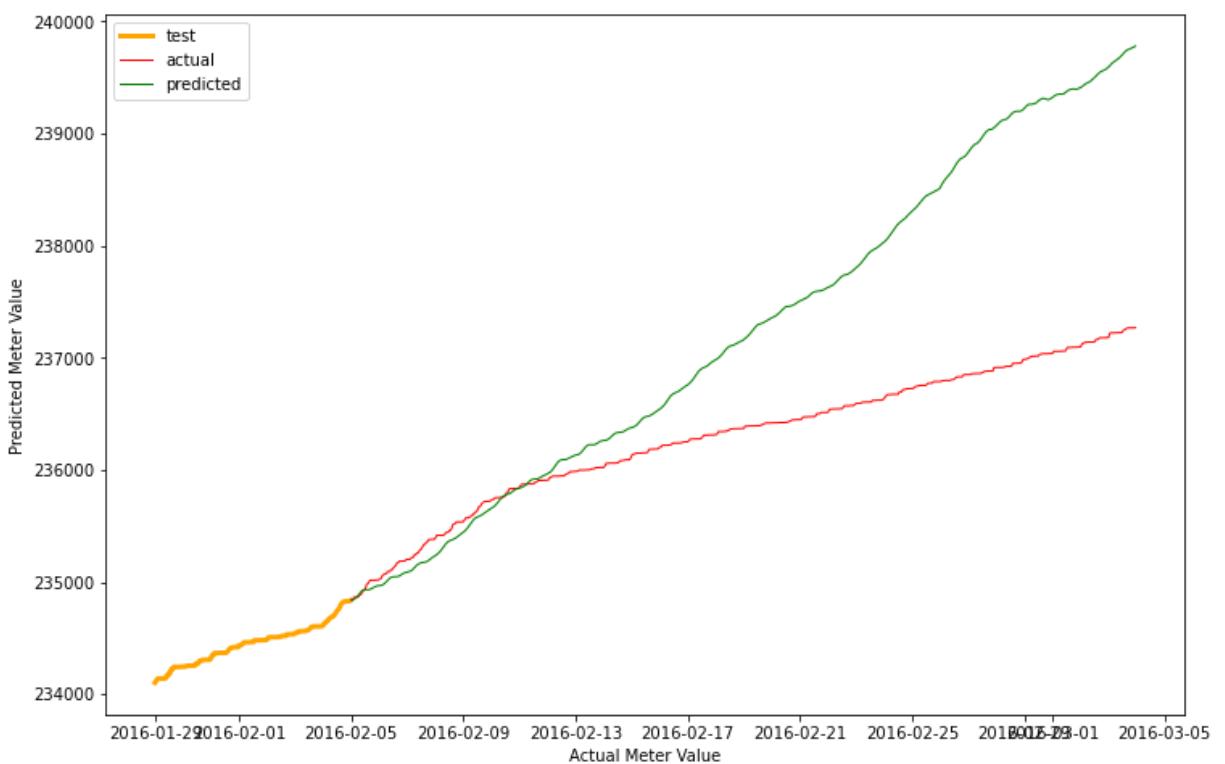
Household 7030.0  
Testing MSE: 3980049.64671471



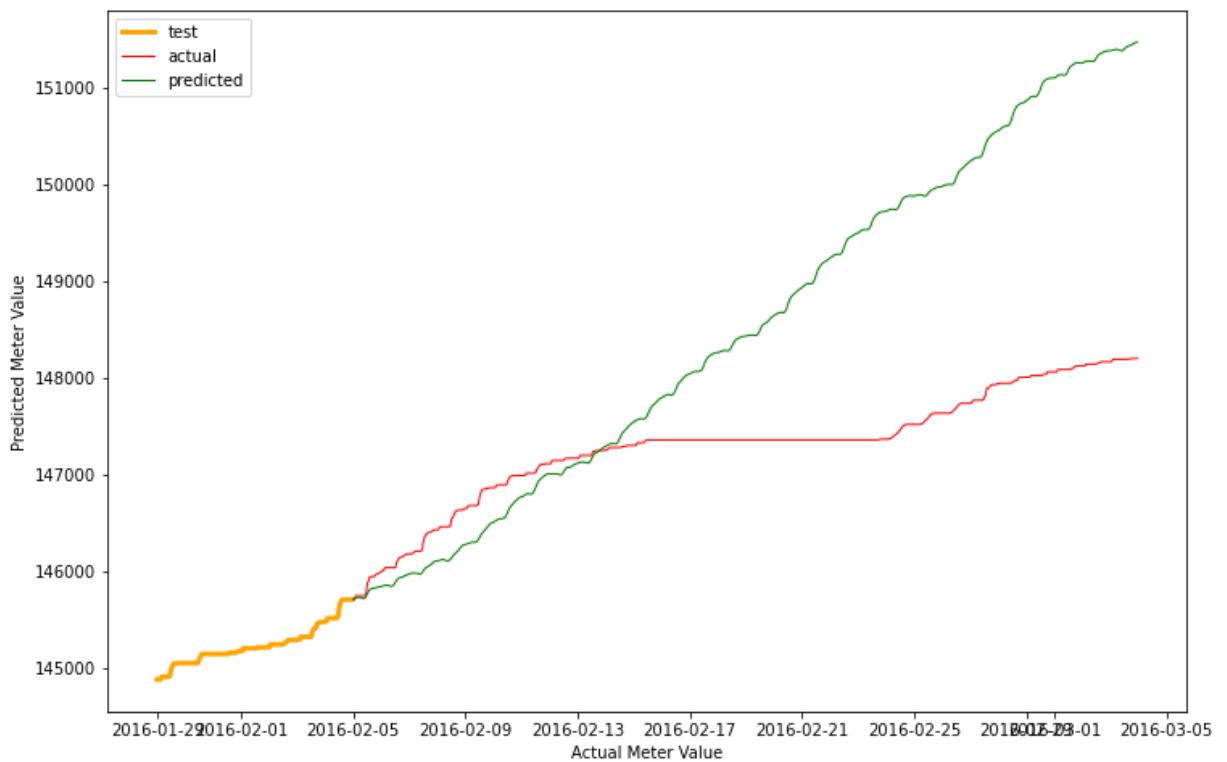
Household 7117.0  
Testing MSE: 1893169.8358699253



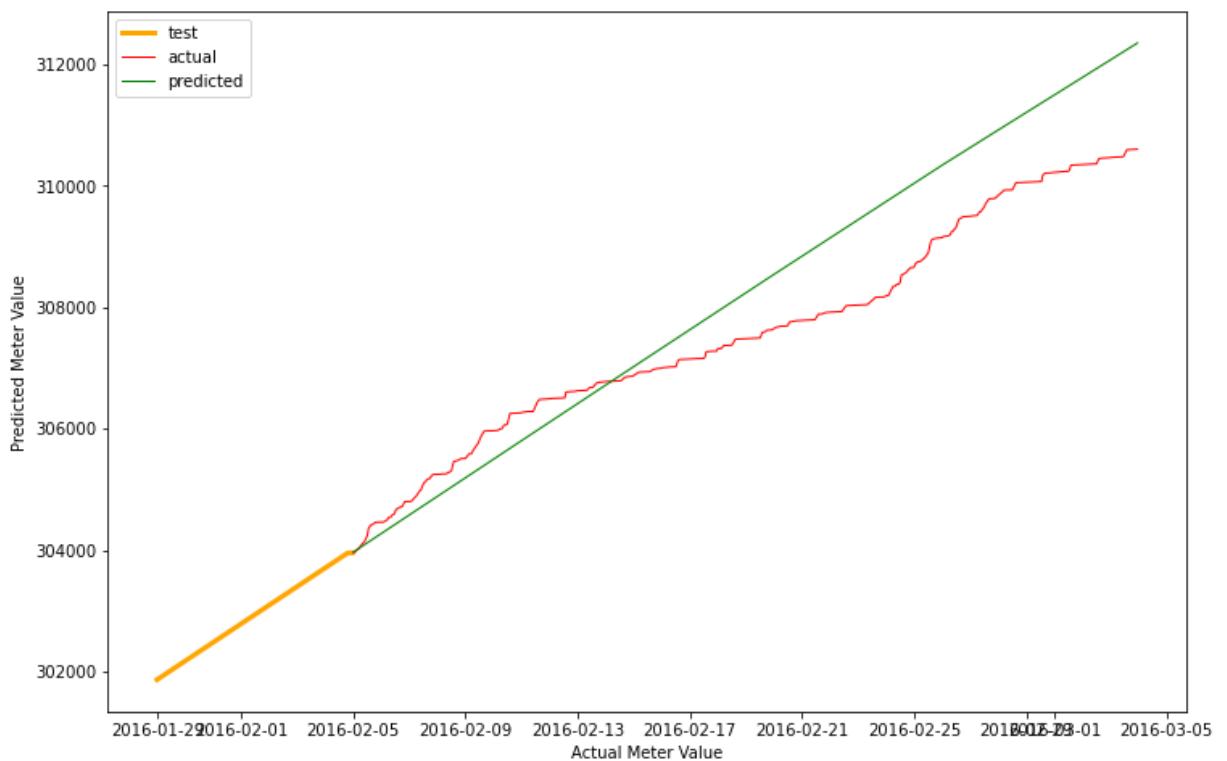
Household 7287.0  
Testing MSE: 1680684.5117256527



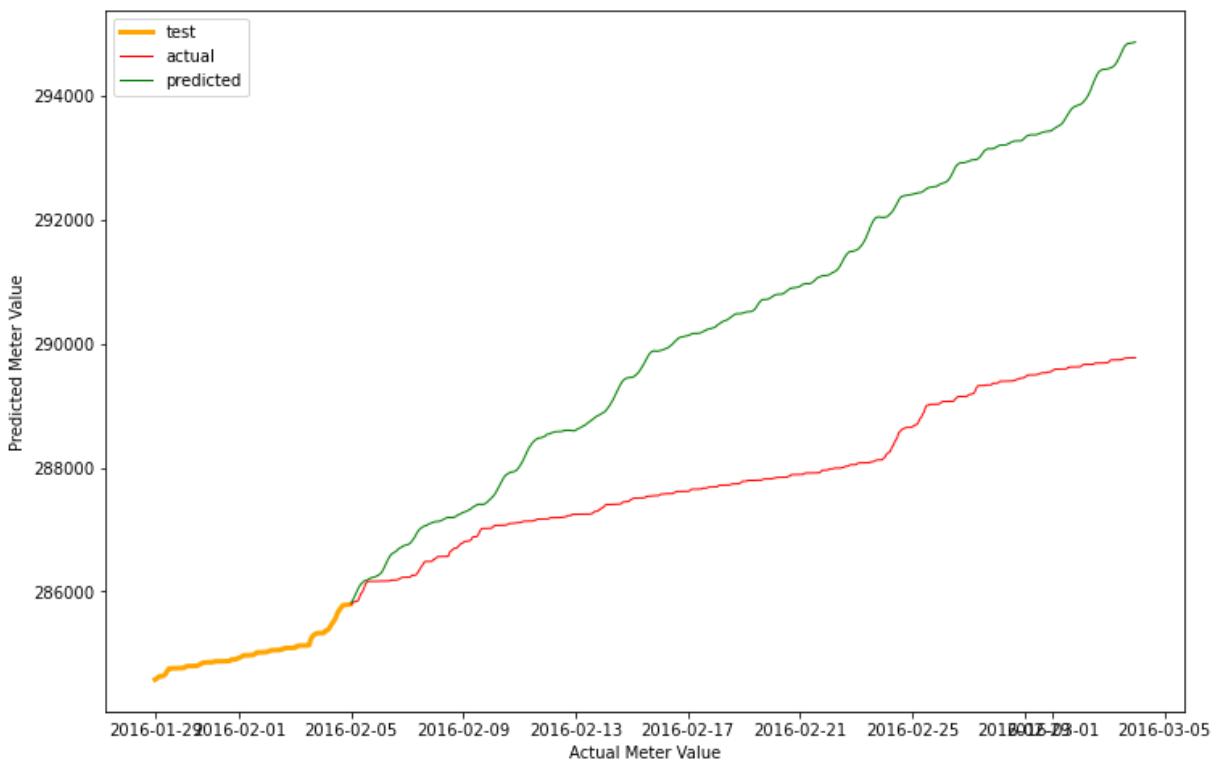
Household 7429.0  
Testing MSE: 3122322.011177426



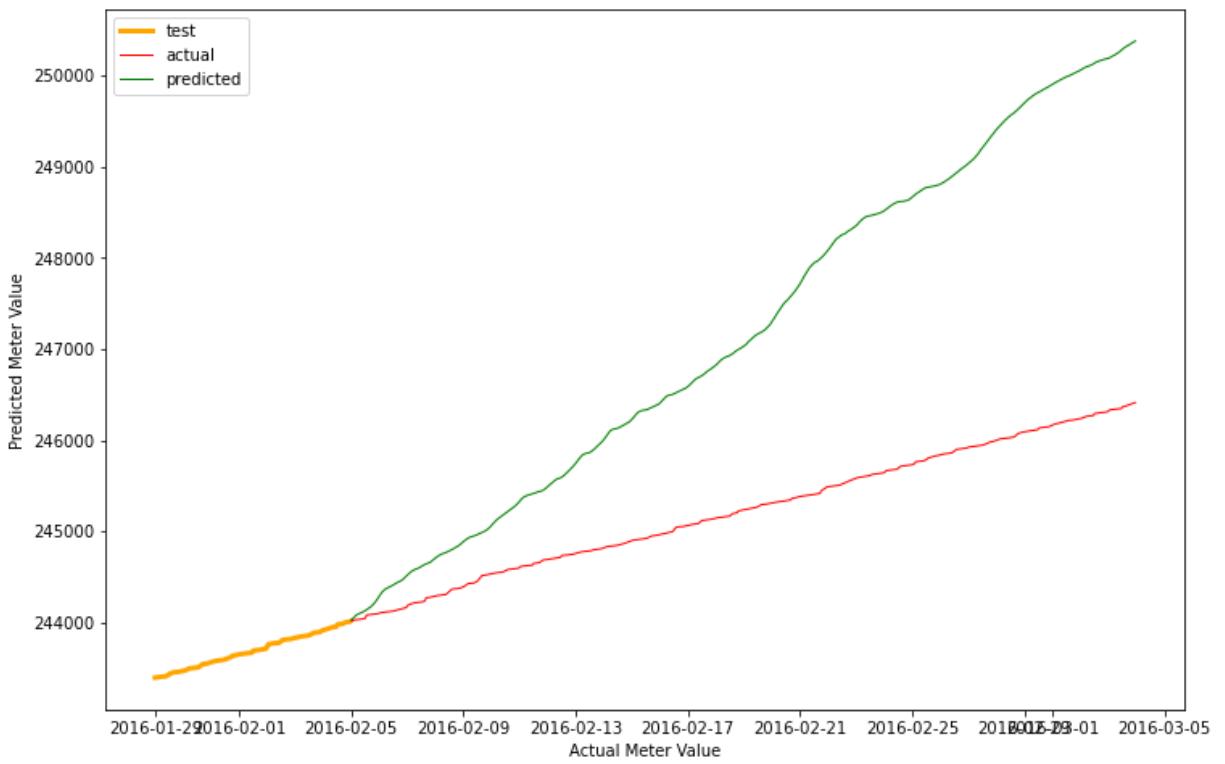
Household 7460.0  
Testing MSE: 886580.0922996239



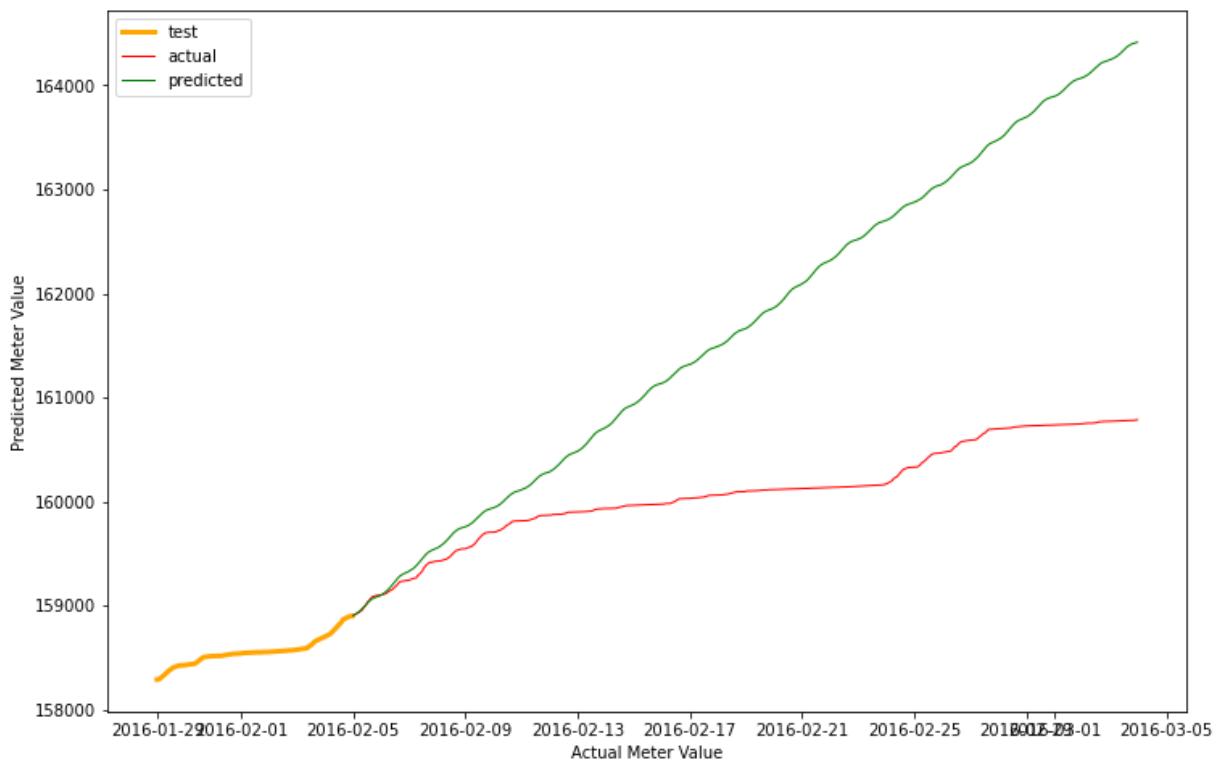
Household 7674.0  
Testing MSE: 8290717.265190488



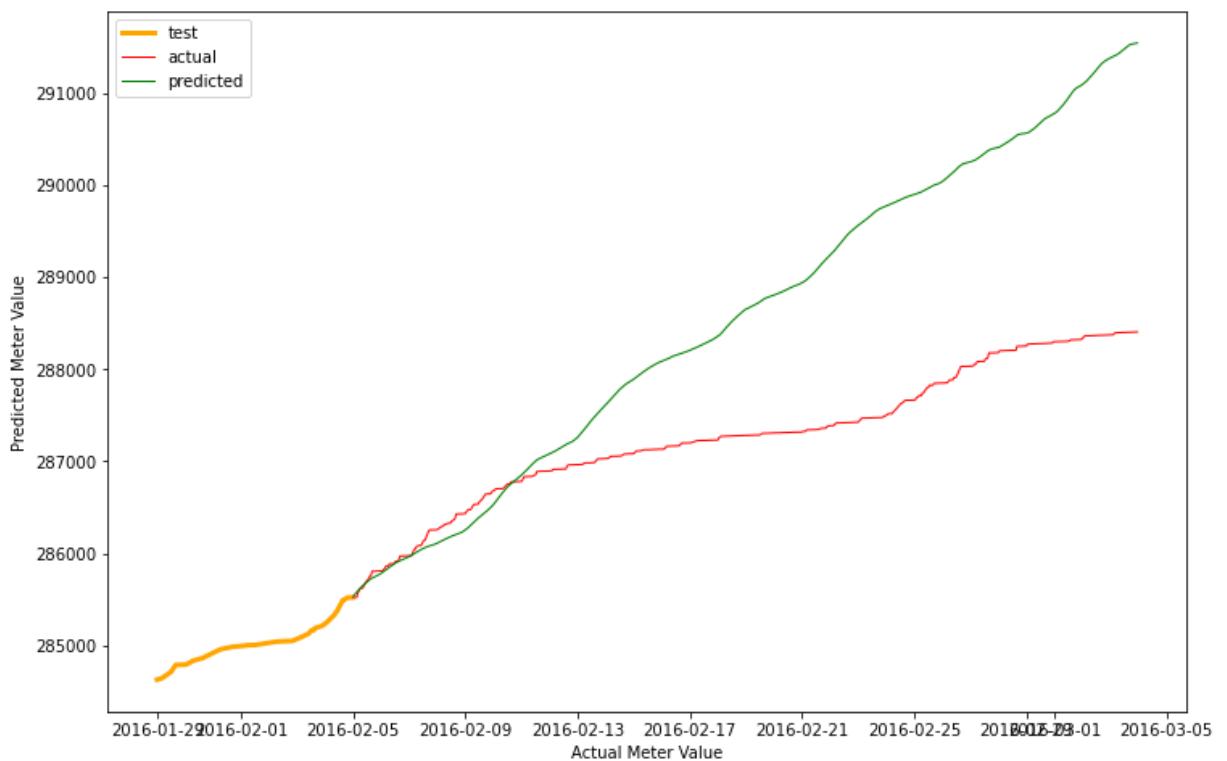
Household 7682.0  
Testing MSE: 5438310.9077138575



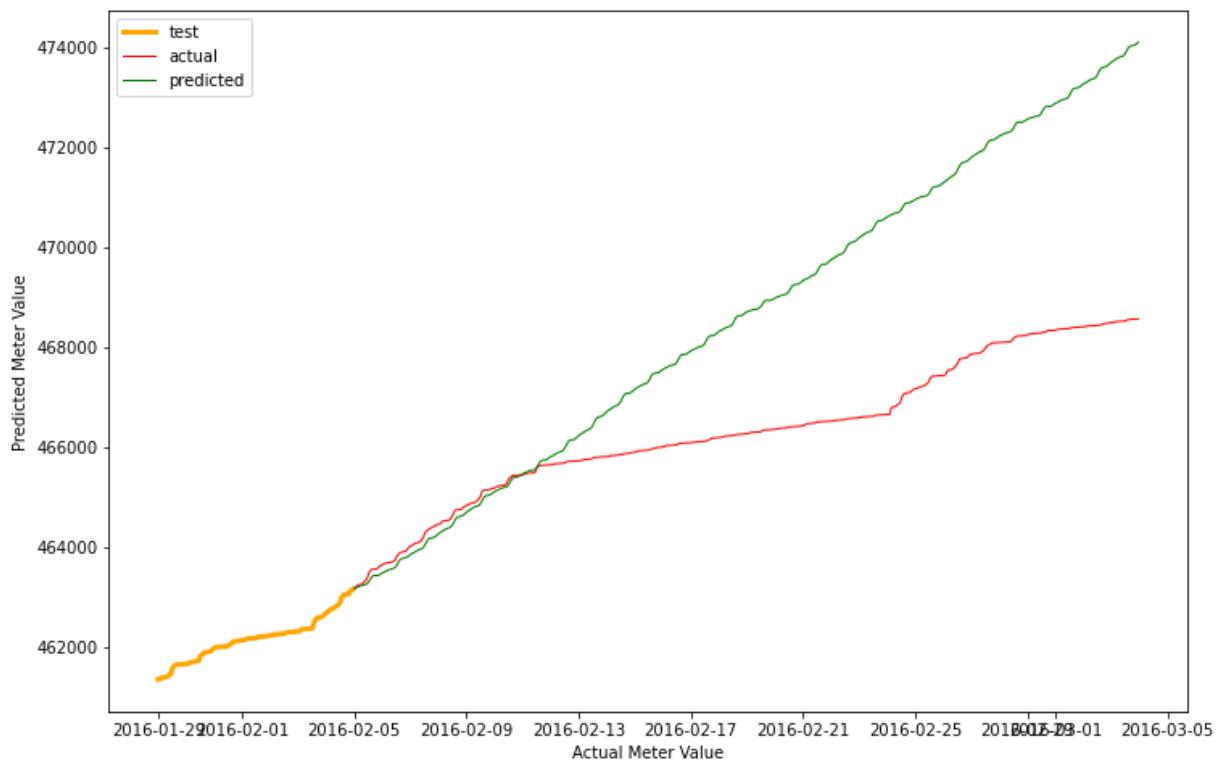
Household 7739.0  
Testing MSE: 3940142.2287210906



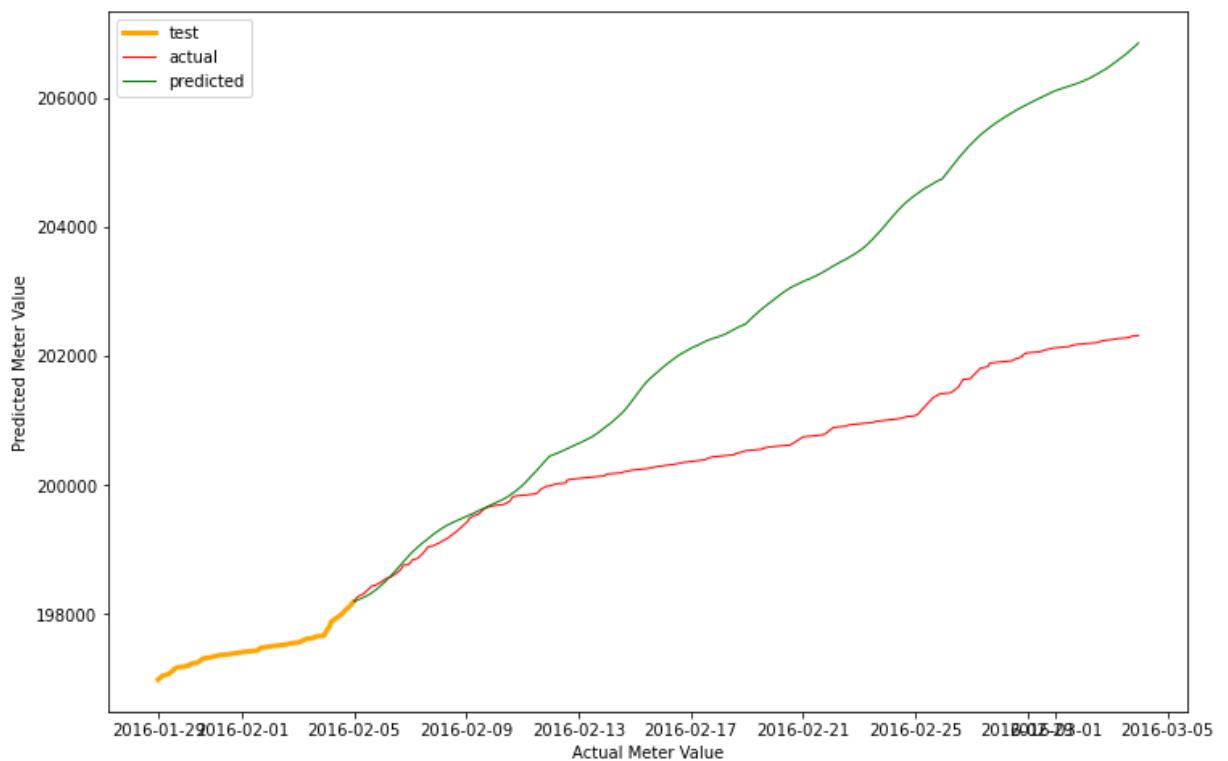
Household 7741.0  
Testing MSE: 2738302.768418085



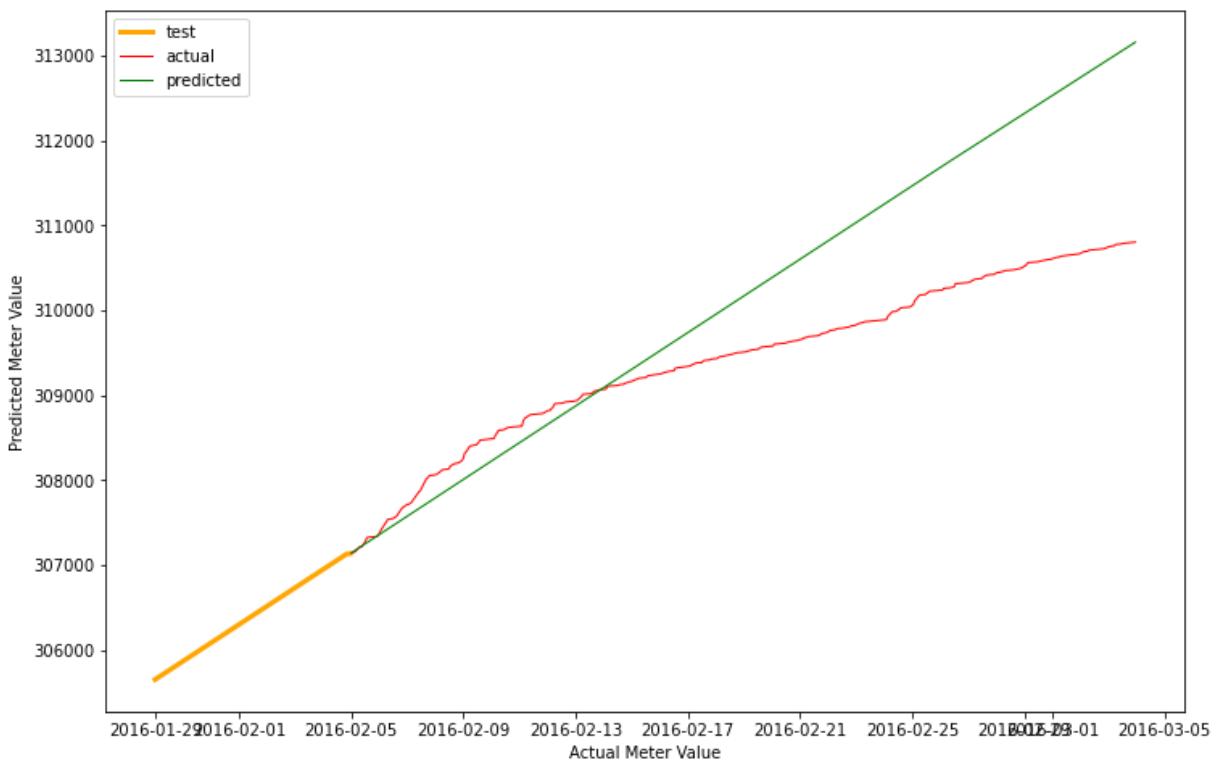
Household 7794.0  
Testing MSE: 8496022.352969054



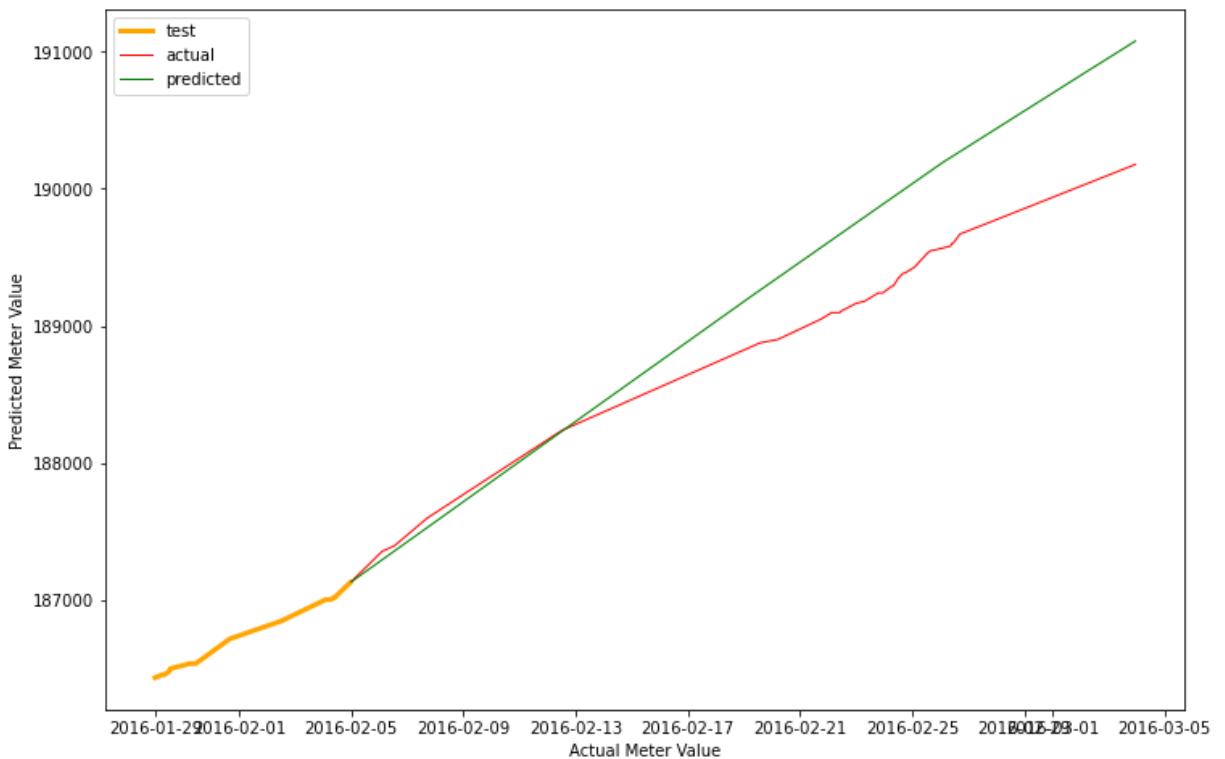
Household 7900.0  
Testing MSE: 6245964.691392444



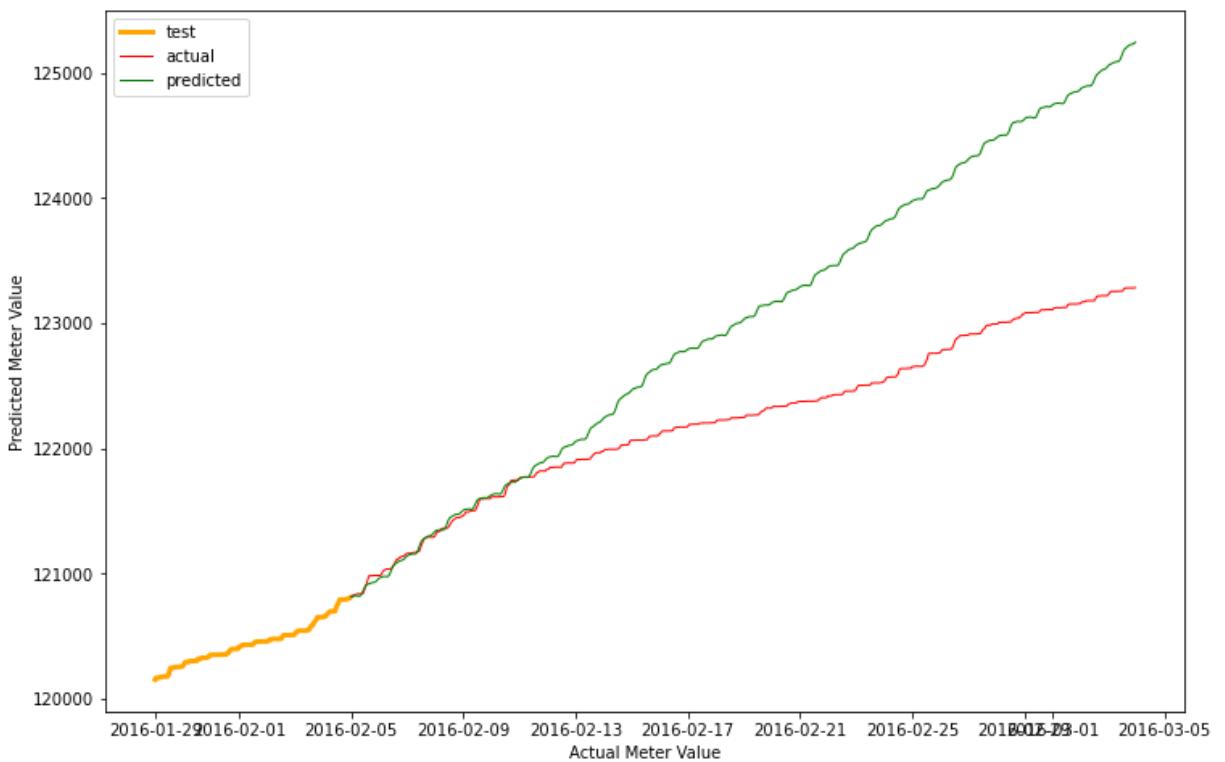
Household 7919.0  
Testing MSE: 1250460.434251864



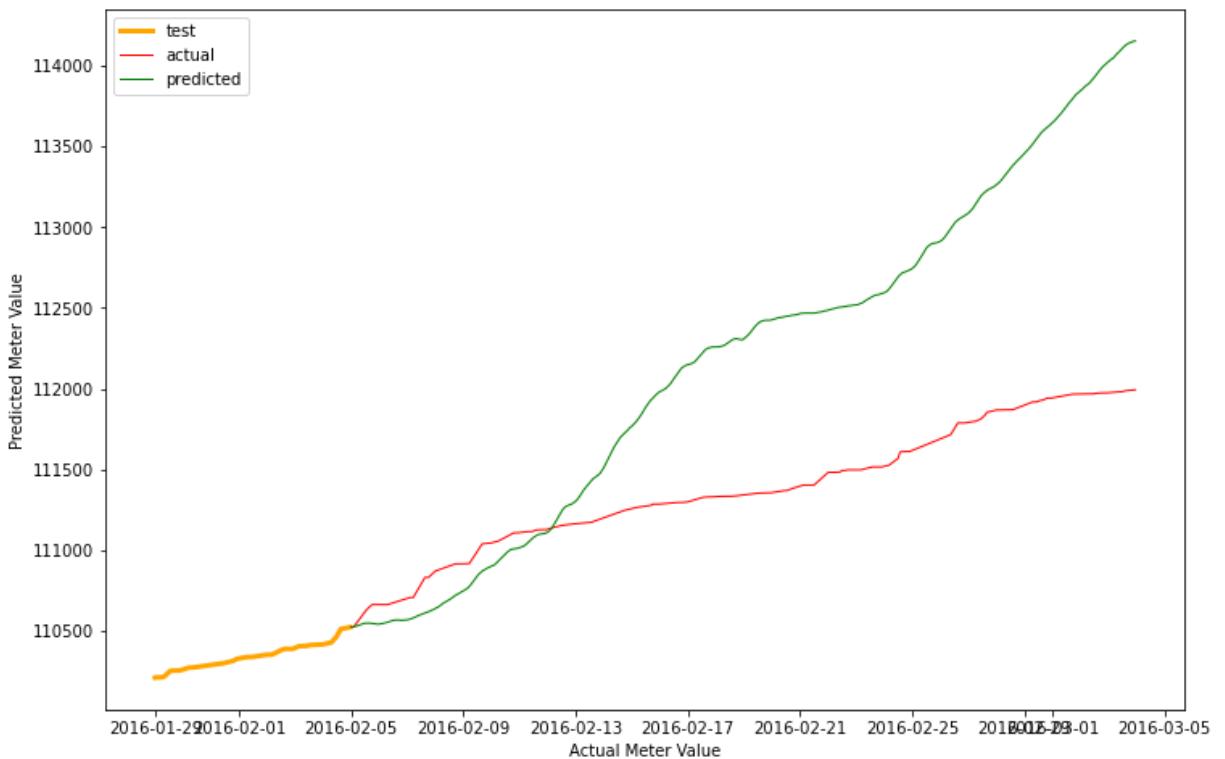
Household 7965.0  
Testing MSE: 222726.71286646728



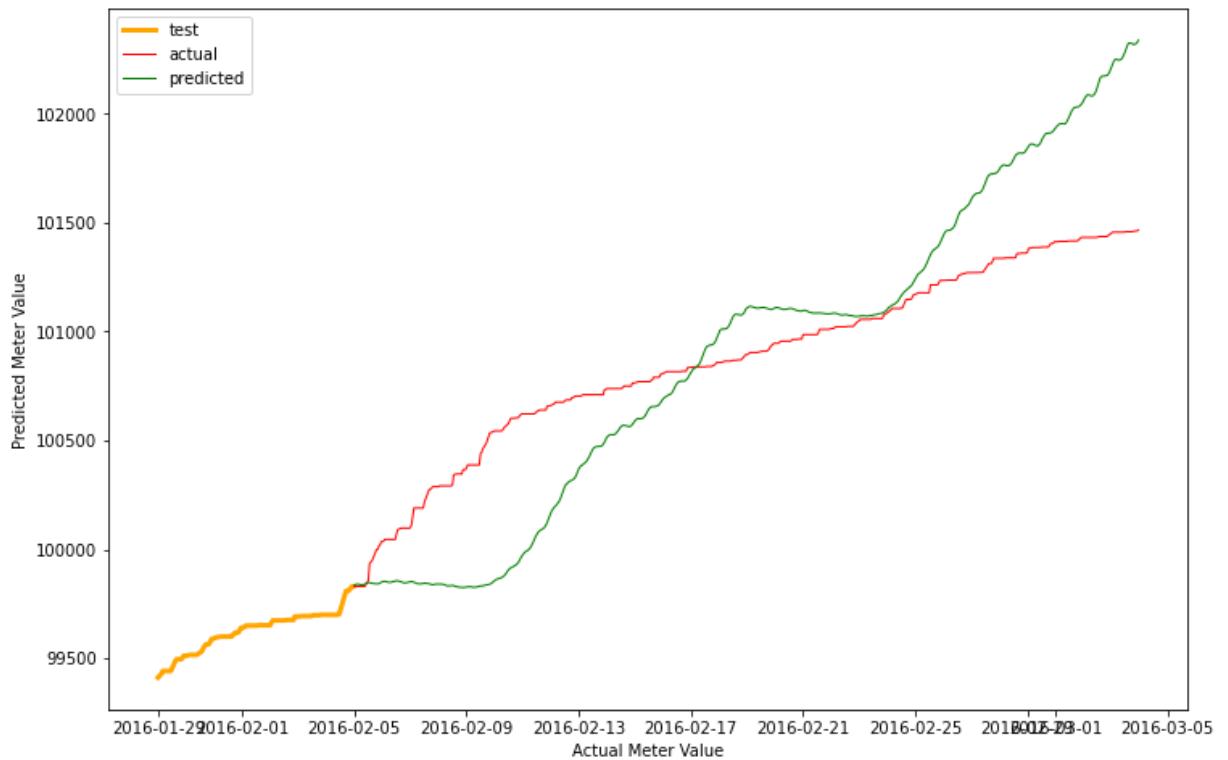
Household 7989.0  
Testing MSE: 1009190.7195862362



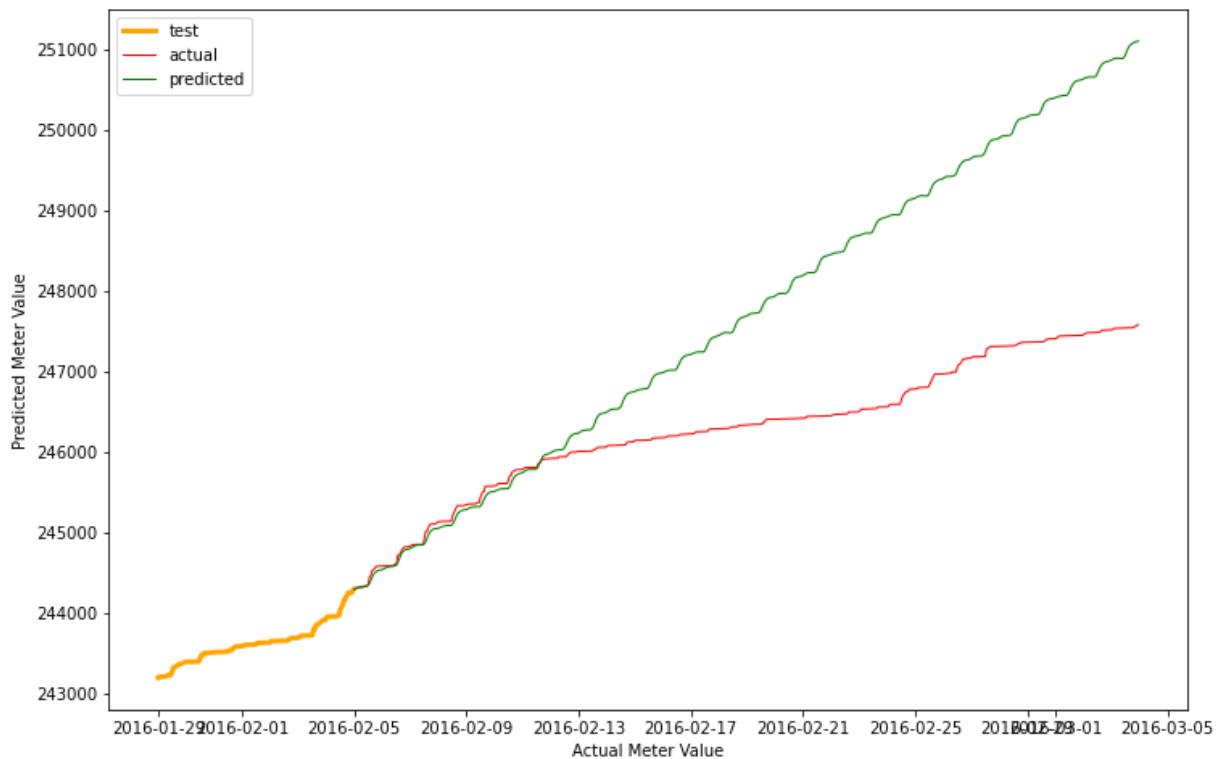
Household 8059.0  
Testing MSE: 1103881.5673352836



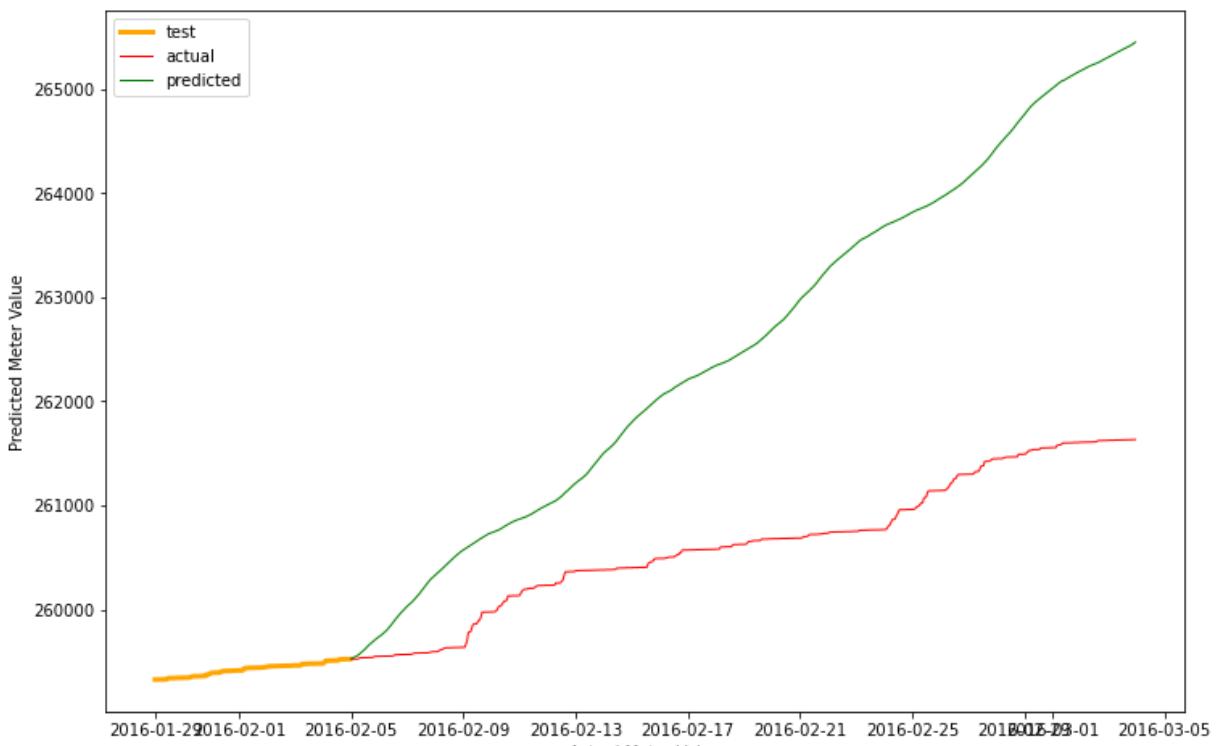
Household 8084.0  
Testing MSE: 150135.7348276108



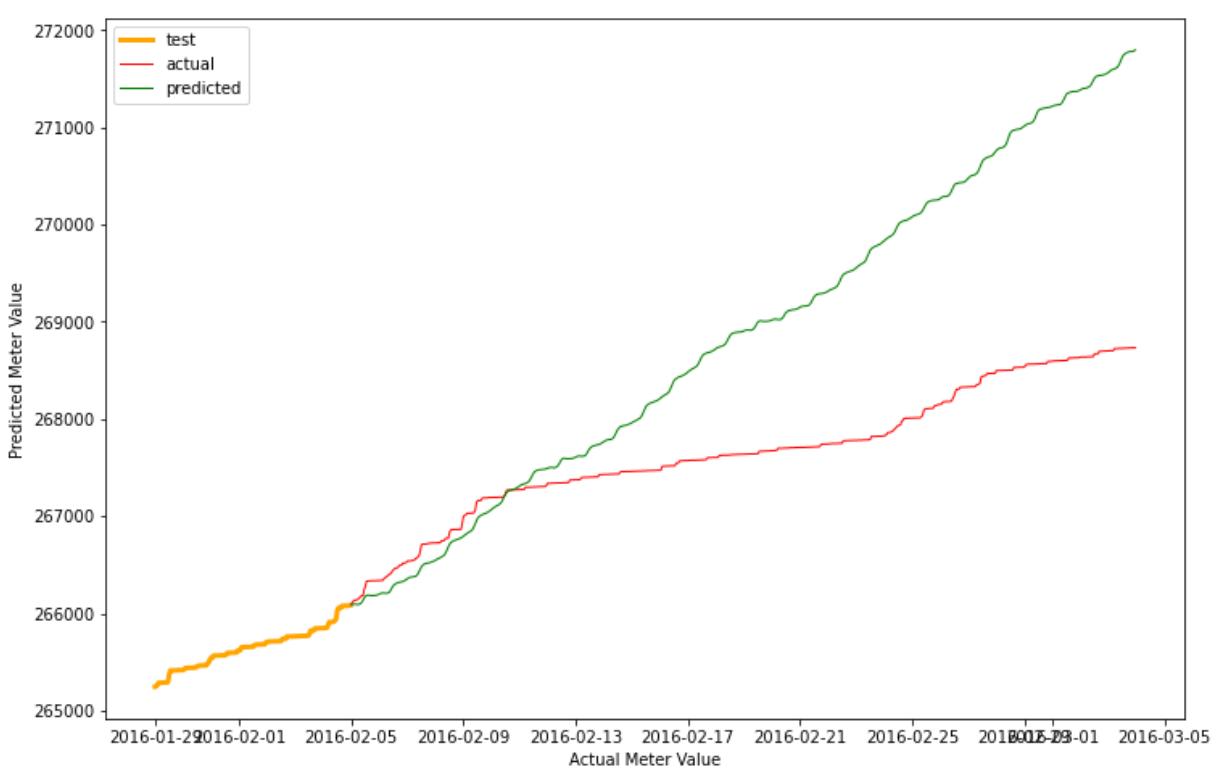
Household 8086.0  
Testing MSE: 3309980.540578098



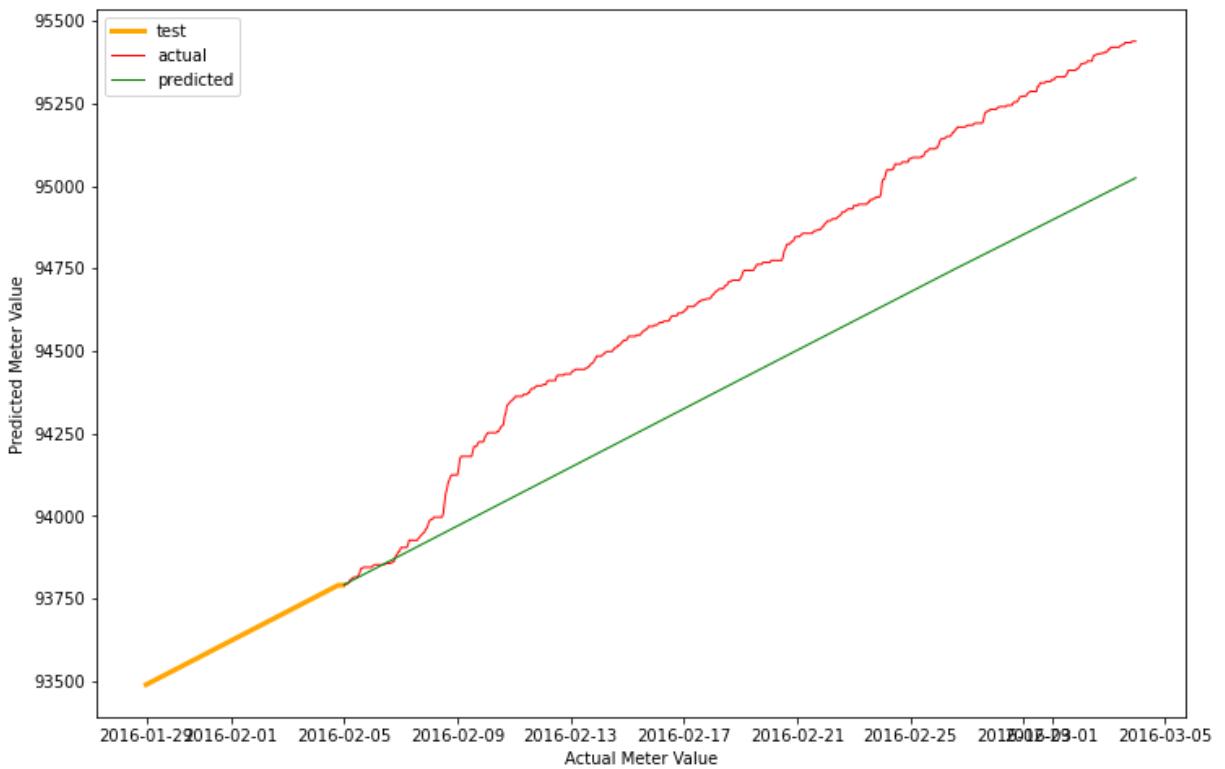
Household 8155.0  
Testing MSE: 4955170.76868411



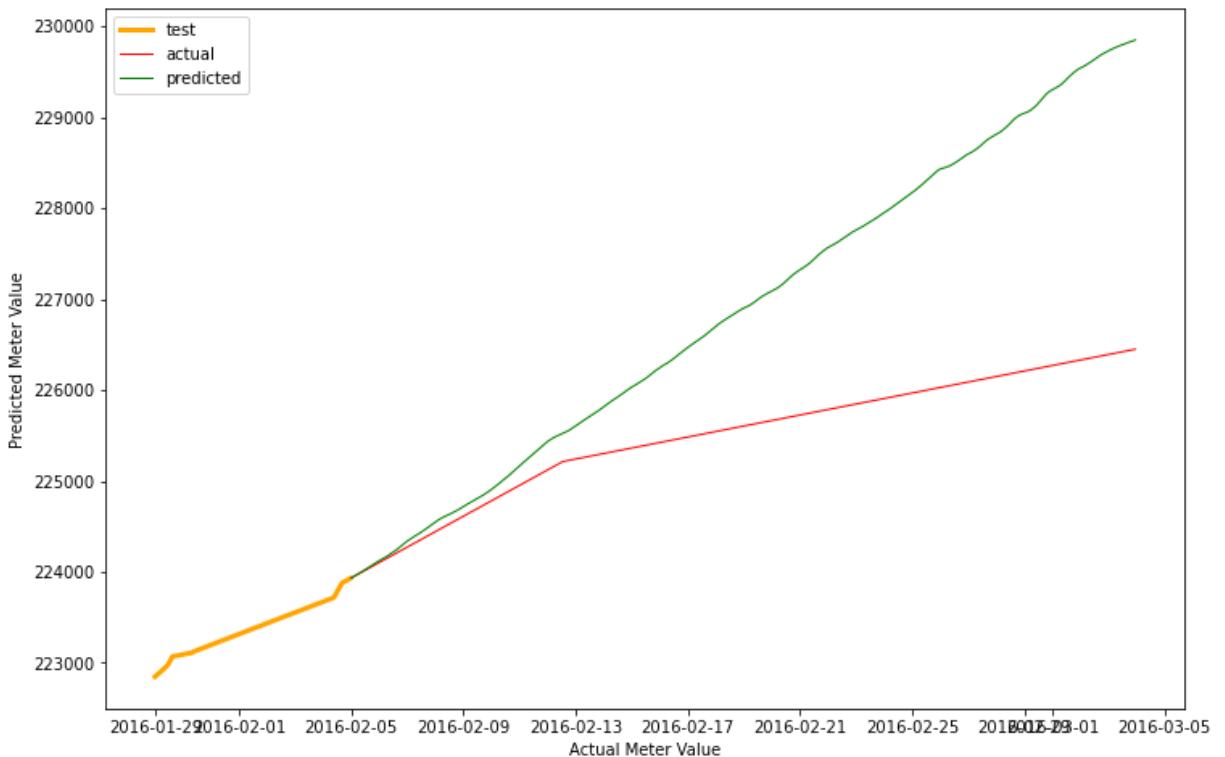
Household 8156.0  
Testing MSE: 2520137.2862287248



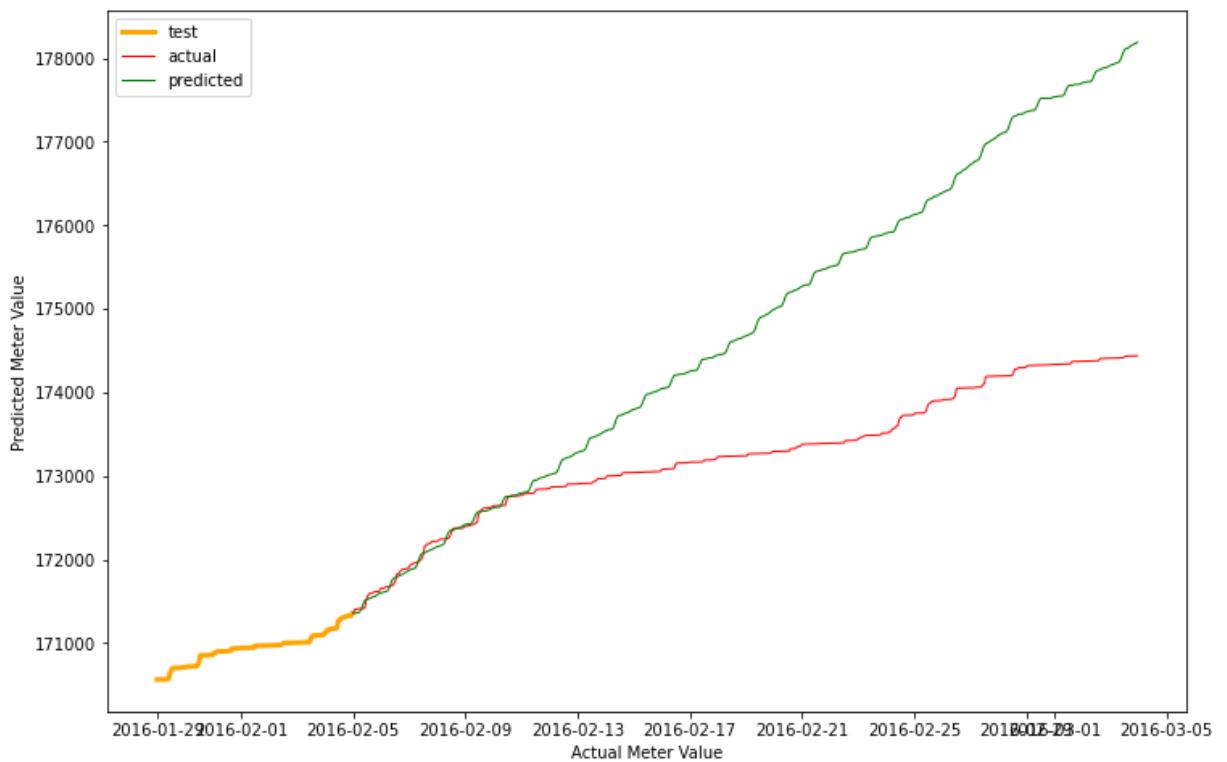
Household 8467.0  
Testing MSE: 104385.7012272083



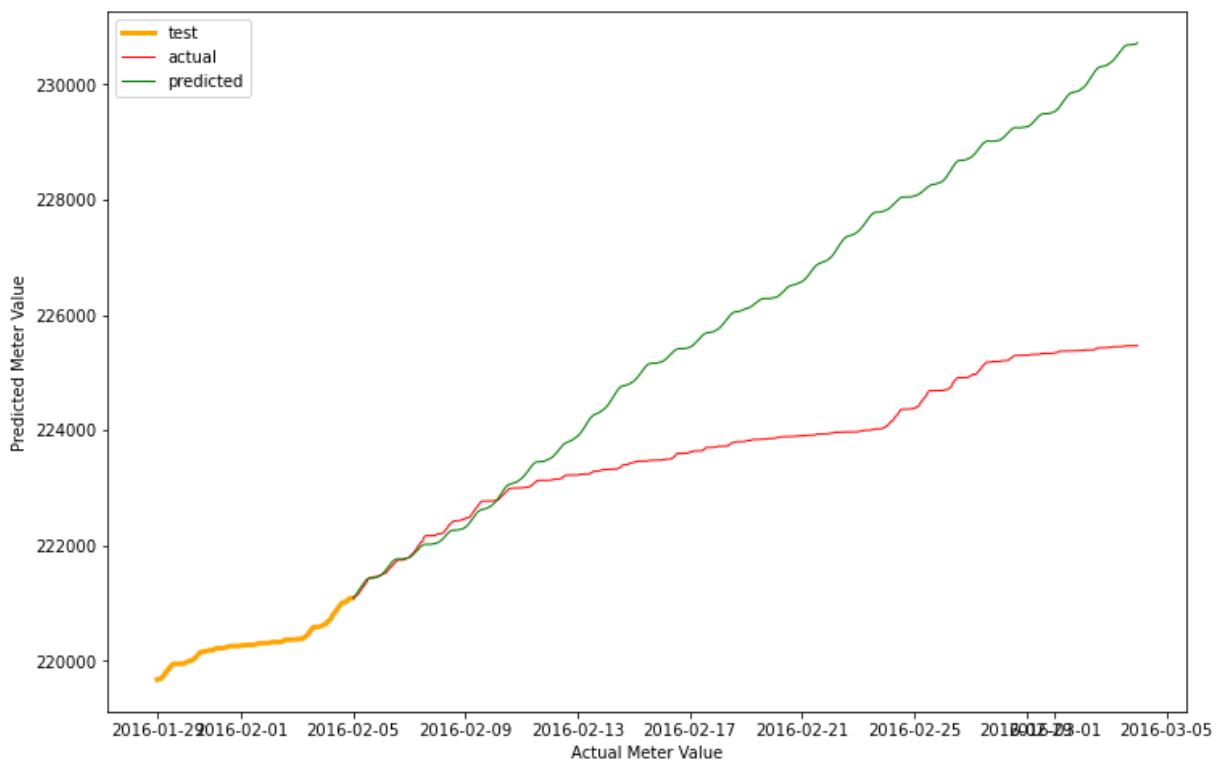
Household 8703.0  
Testing MSE: 3190991.1323186345



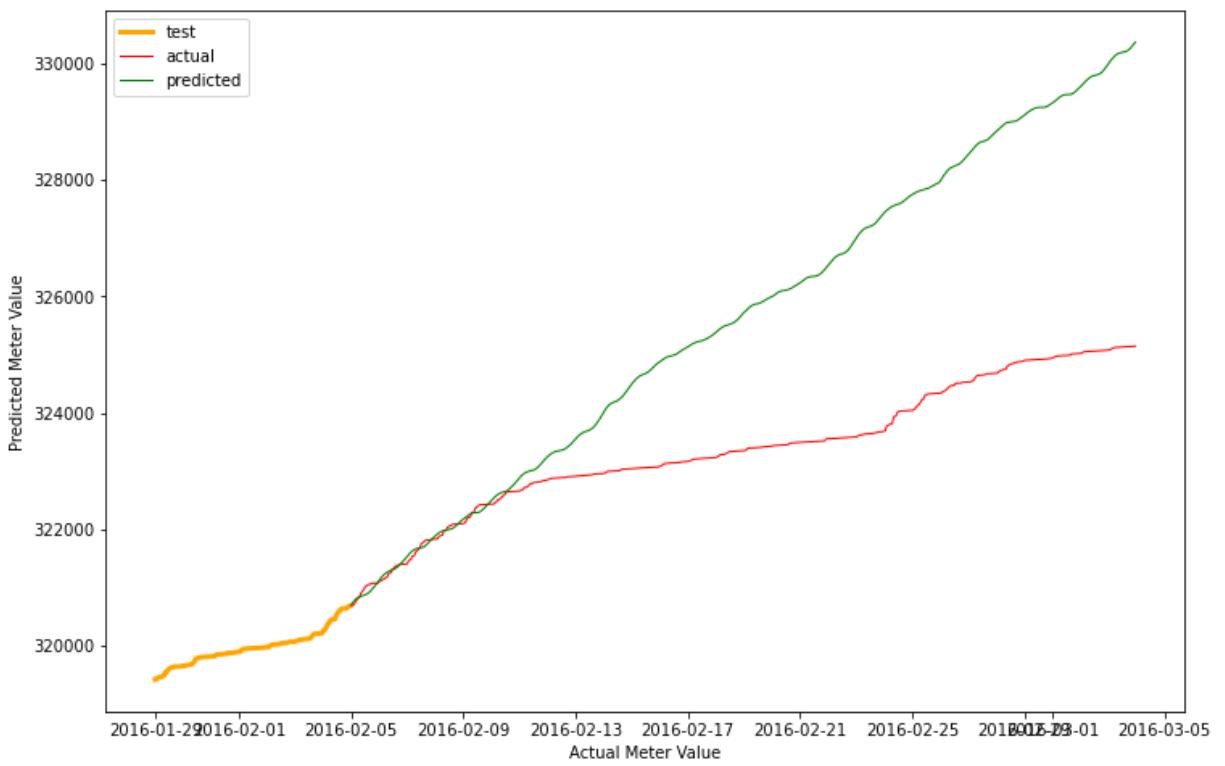
Household 8829.0  
Testing MSE: 3793319.793337585



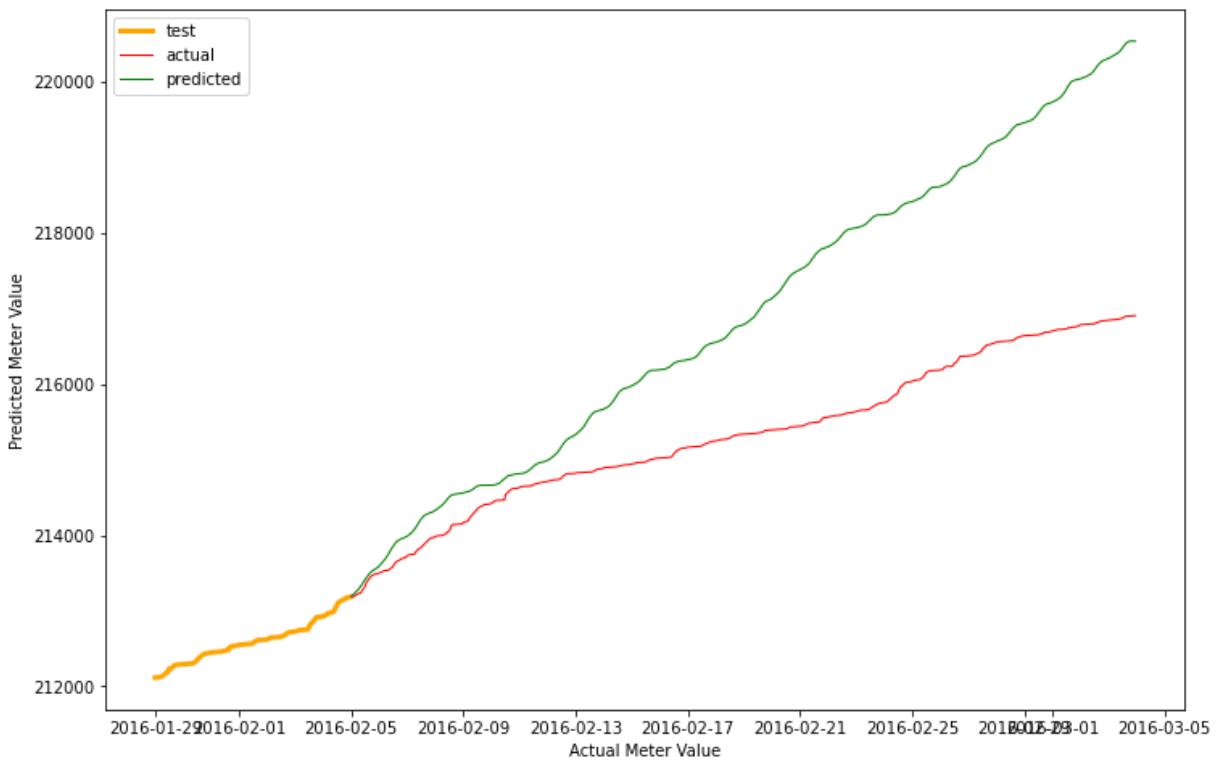
Household 8890.0  
Testing MSE: 7812693.315435137



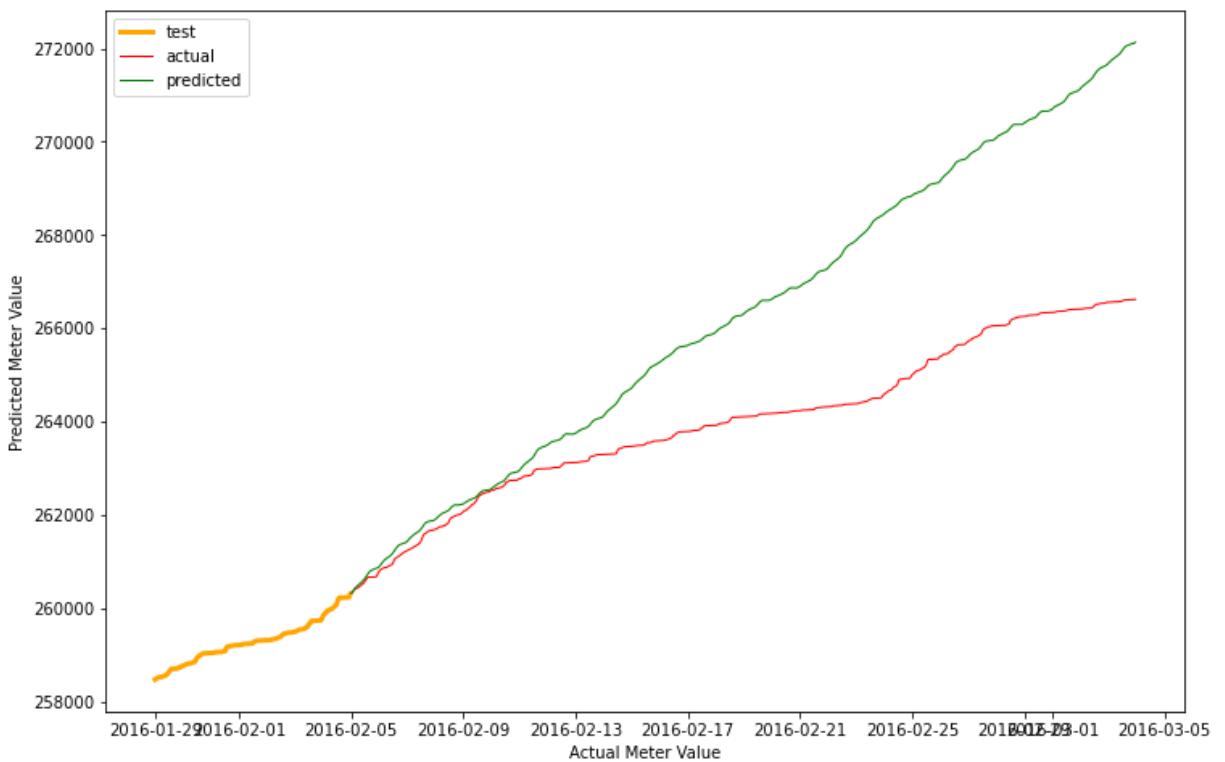
Household 9052.0  
Testing MSE: 7918257.377587052



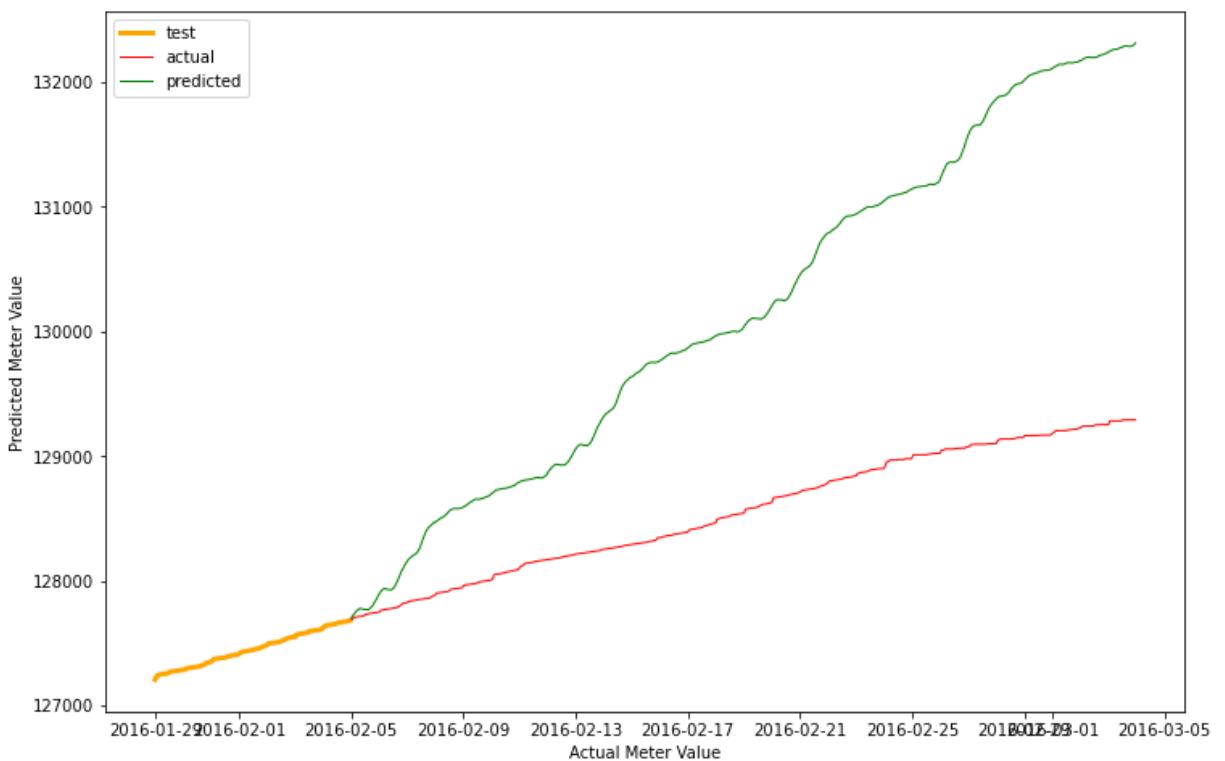
Household 9121.0  
Testing MSE: 3751353.5116885058



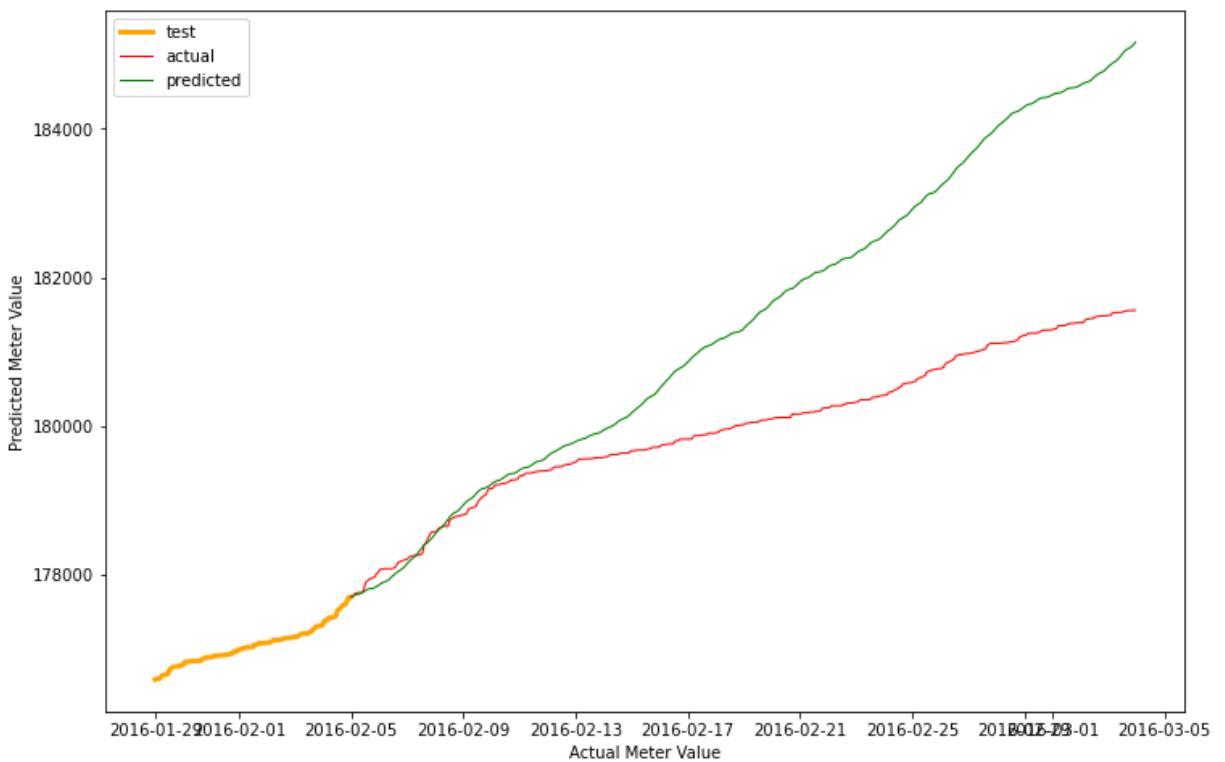
Household 9134.0  
Testing MSE: 8204312.923376982



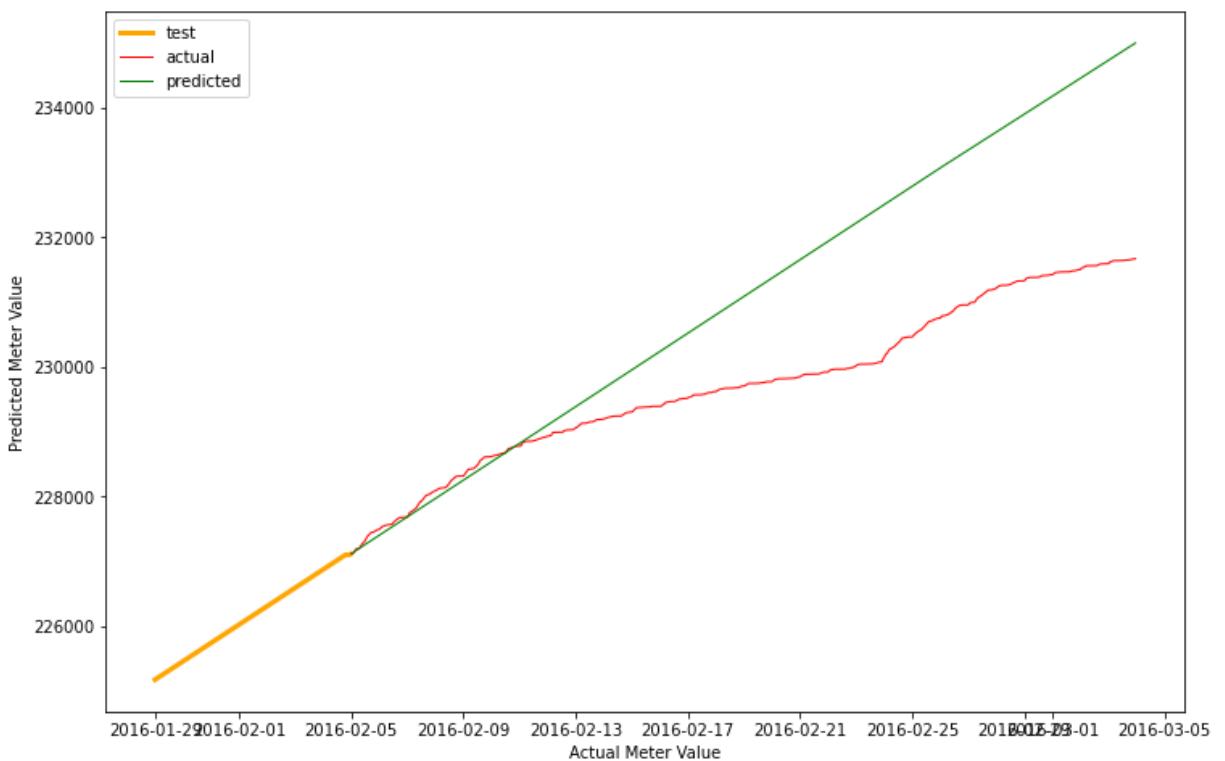
Household 9278.0  
Testing MSE: 3370007.784355345



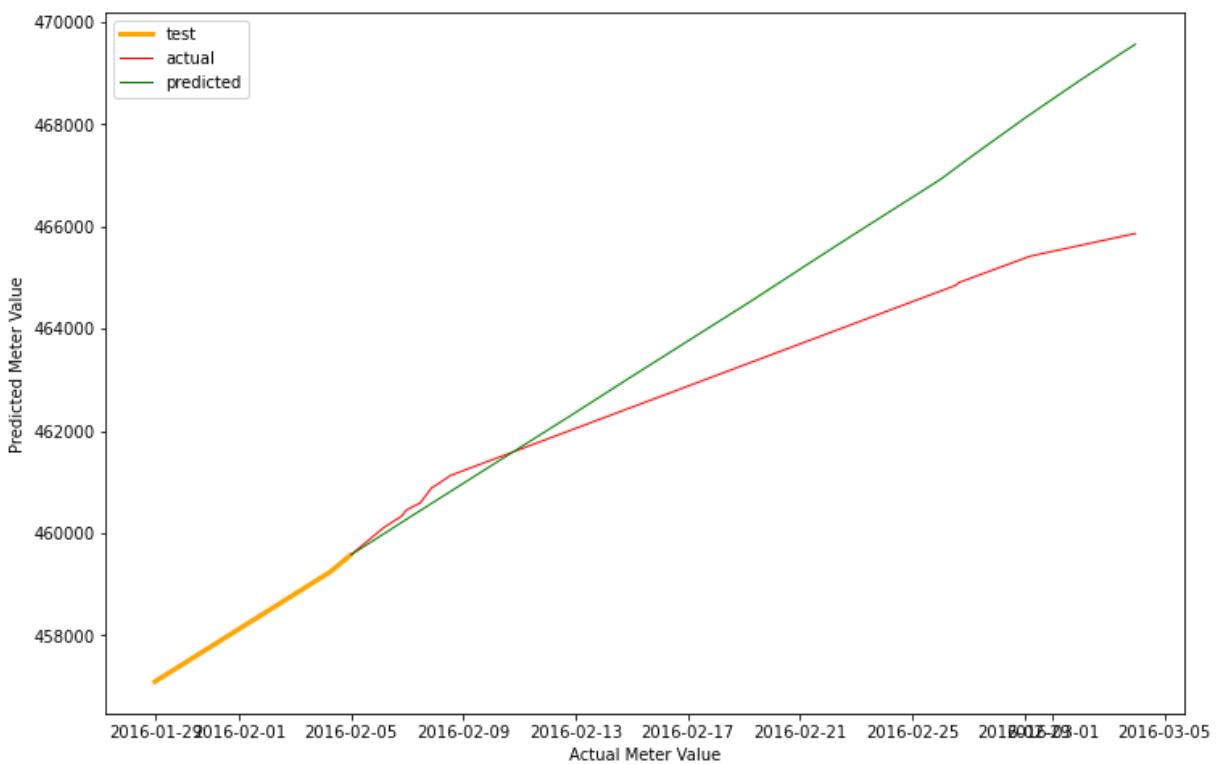
Household 9295.0  
Testing MSE: 3455515.3638332714



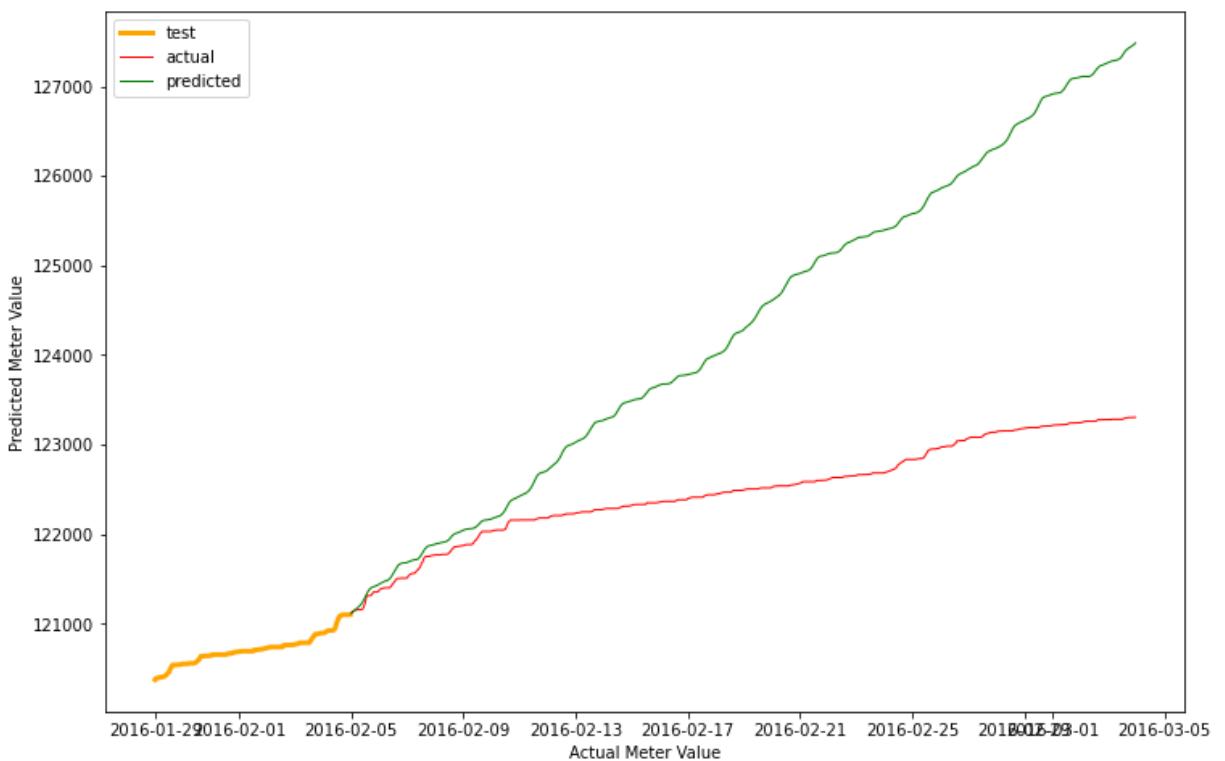
Household 9474.0  
Testing MSE: 3060210.679082382



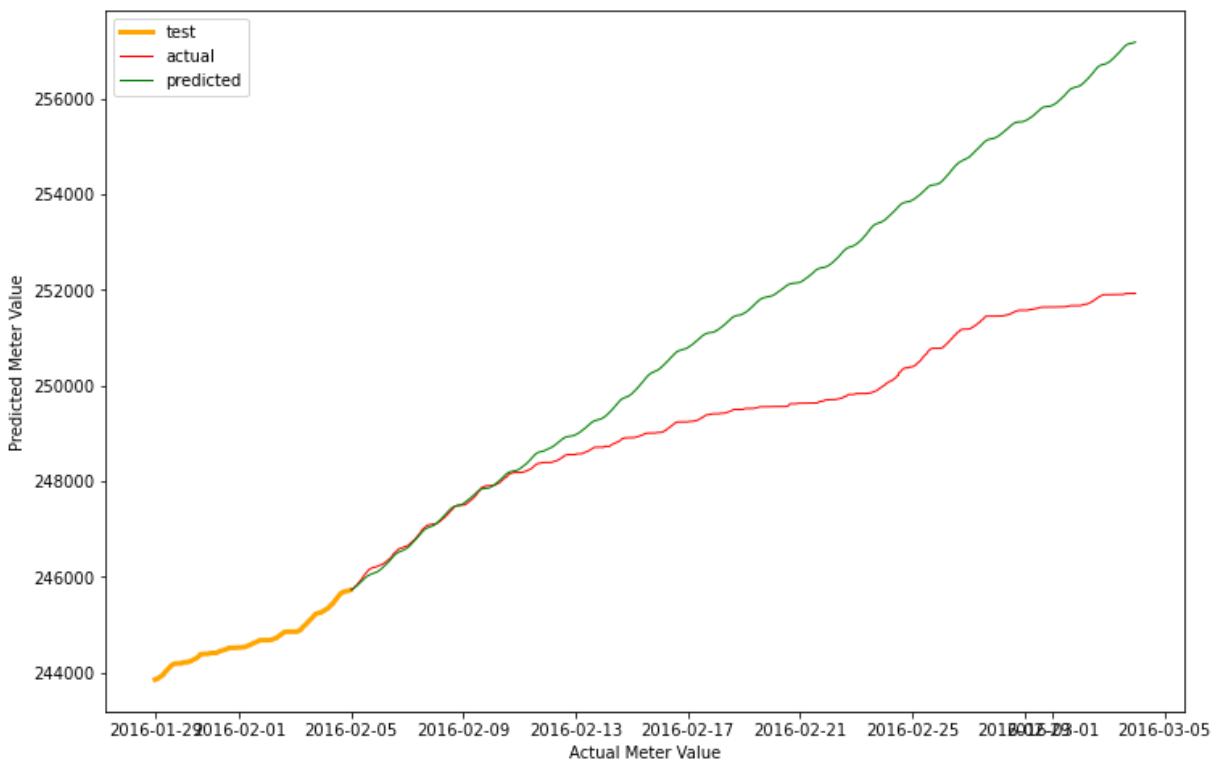
Household 9620.0  
Testing MSE: 3008856.0906004836



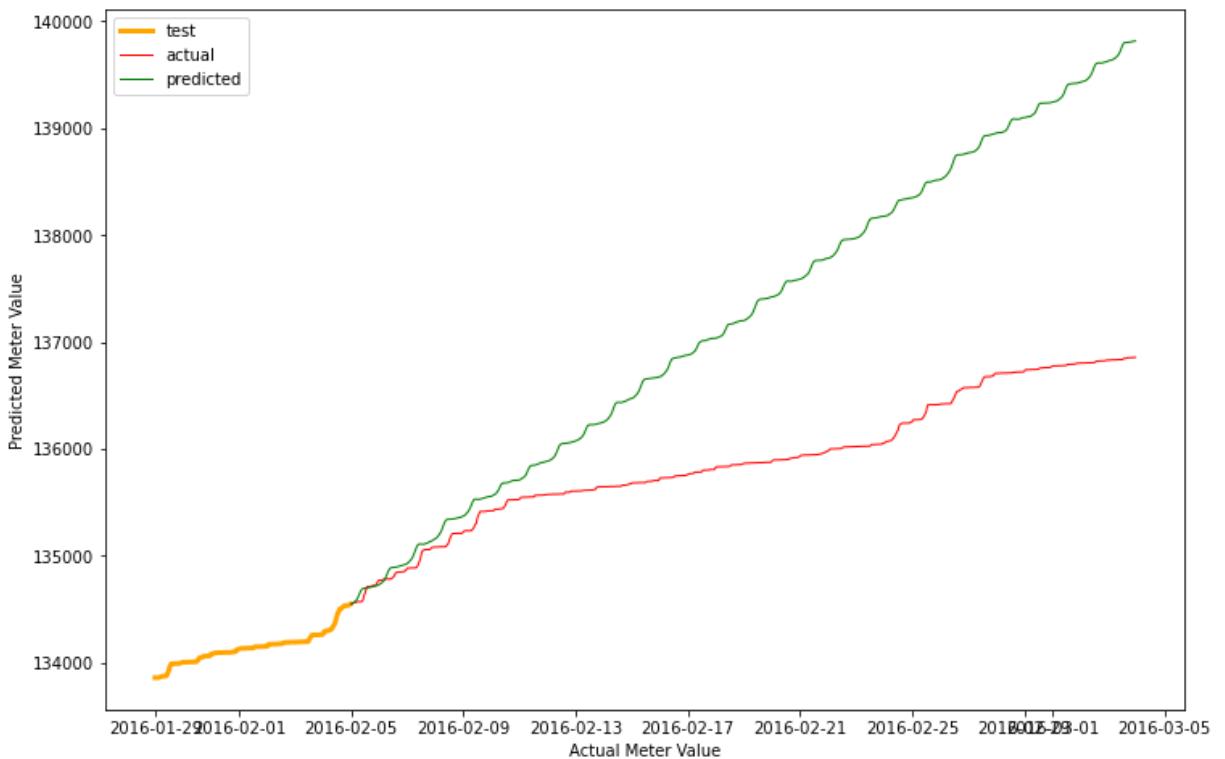
Household 9631.0  
Testing MSE: 5087809.926228156



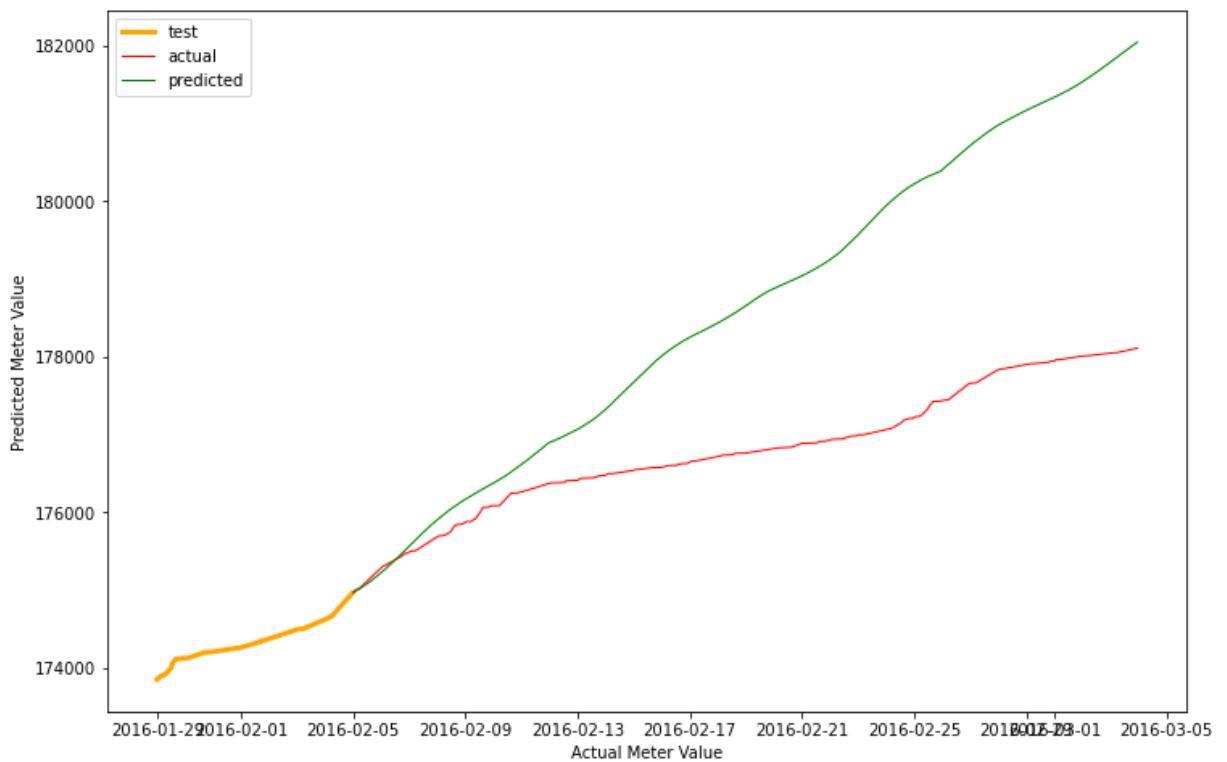
Household 9639.0  
Testing MSE: 7112967.8011797555



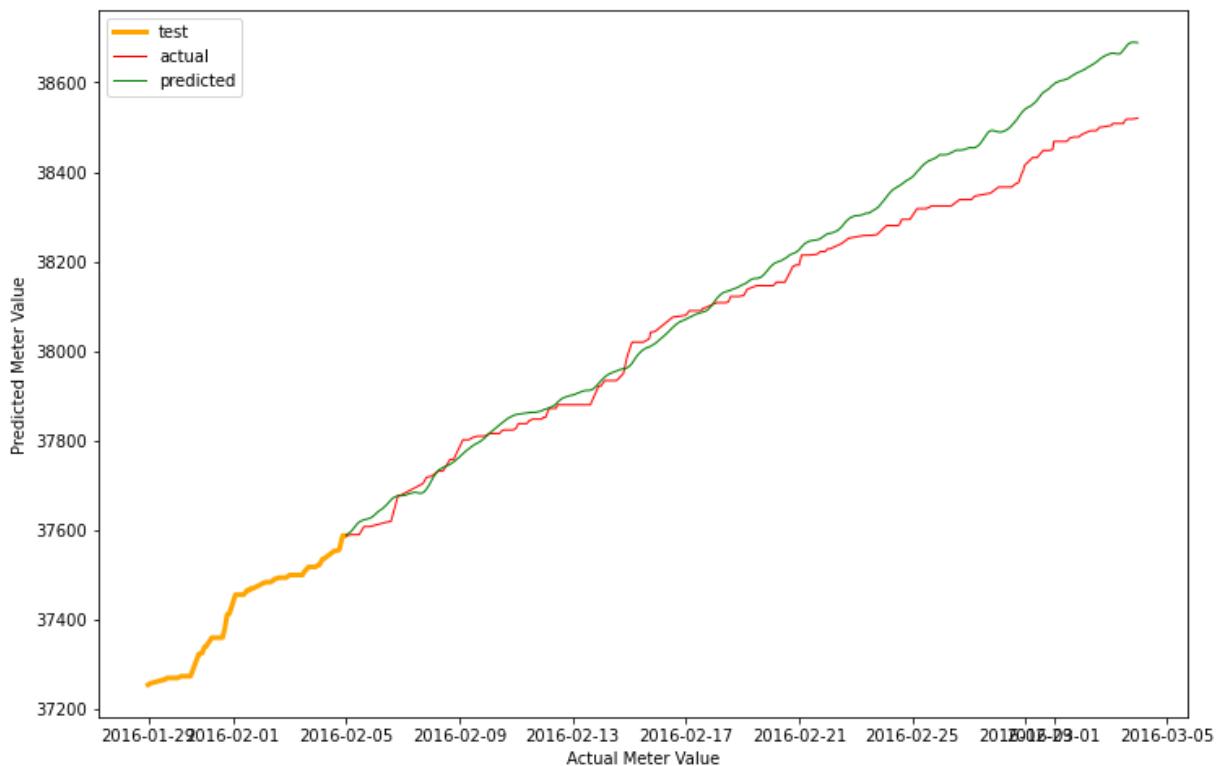
Household 9729.0  
Testing MSE: 2649583.136858622

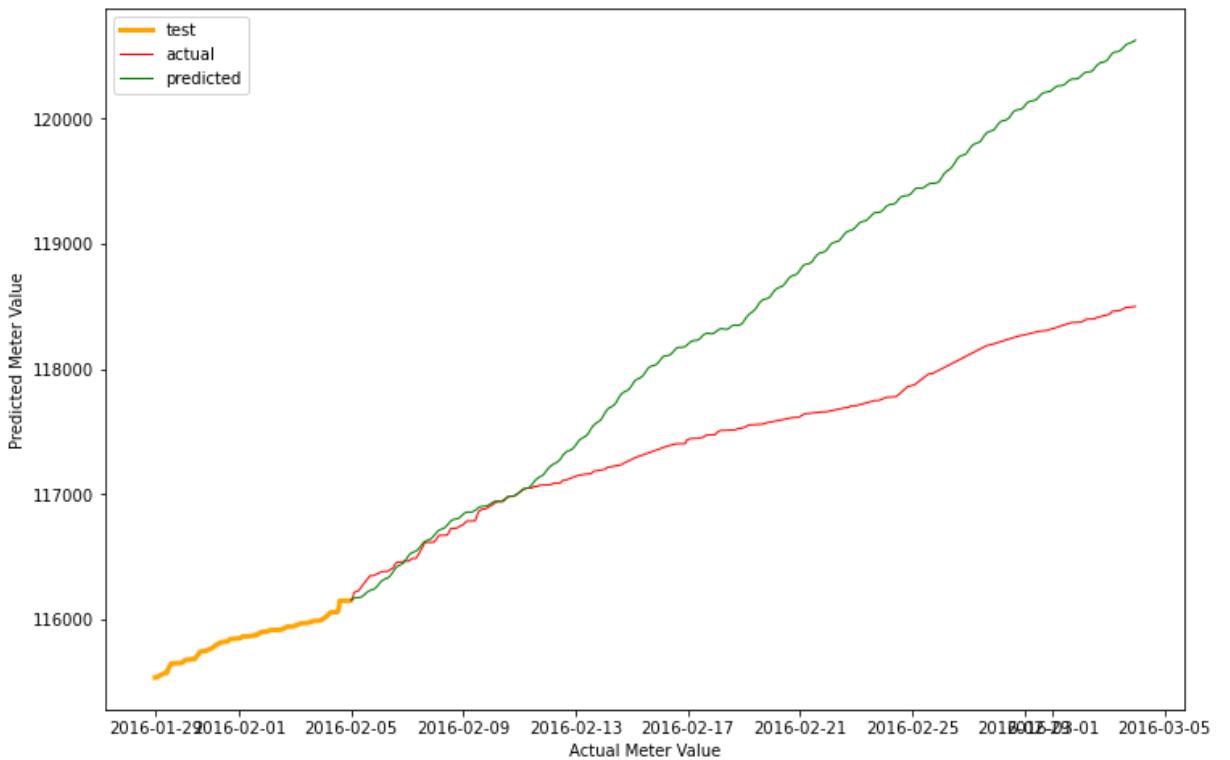


Household 9766.0  
Testing MSE: 4884760.594407199



Household 9849.0  
Testing MSE: 5719.280819659118





Above, we have plots showing the actual meter values versus the predicted meter values for each of the respective household. In every plot, the household ID is reflected on the top in the title of the plot. Similarly, the MSE error value for the testing dataset for that household is also shown on the top in the title of the plot.

Upon close and through inspection, we actually feel that the LSTM model displays the best accuracy that we have seen among all the models that we have created so far. This is despite the fact that the data that we used to build the model is not large.

### Section 3.4: How the LSTM model can be improved?

Above, with the use of LSTM model, we were able to achieve some improvement in accuracy compared to all the models that we have tried so far. Below, we will list some other ways in which we feel the model can be improved further.

#### 1: Tune hyperparameters

We can change some of the hyperparameters to see the effect on accuracy and performance. These hyperparameters include number of nodes and hidden layers, dropout, number of epochs and etc. Due to time constraint, we were not able to experiment with many of these hyperparameters. But given sufficient time, one can definitely tweak the hyperparameters to observe higher accuracy. On a similar note, we felt that the MSE values were oscillating probably due to random initialization of the weights. That could have been changed as well.

#### 2: Adding more data

LSTM being a neural network tend to perform well with more data. And as discussed under section 2, if there were data from more households from the same neighbourhood, that could have increased the performance of the models. In addition, if data during the summer and

spring seasons were collected and used in the model, that could have enhanced the accuracy as well.

### **3: Use random train-test split**

The train-test split was made based on the time sequence and was not a random split. Using a random split could have made the model more robust but unfortunately, we didn't have a large dataset to have several contiguous and sufficiently large train-test splits.

### **4: Use of transfer learning**

As written in point 2 above, neural networks require a lot of data which may not be always available. That's where transfer learning can come in. Firstly, transfer learning does not require much data as the model is already pre-trained. Secondly, transfer learning saves training time and more importantly gives better performance (in most cases), despite not needing a lot of data.

#### **Other remarks:**

For our neural network, we used MSE as our evaluation metric. However, in reality, we may need to treat overestimation and underestimation differently depending on how the gas provider wants to treat it.

## **Section 3.5: What's next?**

So far, we have seen different models with different accuracies. It is now time to decide which model to choose so that we can then build a tool to provide forecast to households based on the chosen model. After much discussion, we have decided to choose the LSTM model as out of all the models, the LSTM model had marginally better performance. More importantly, with more data, we believe that the LSTM model will perform better compared to other models. As such though LSTM models take time to build and require more computation, we will still choose our LSTM model to provide forecast to households.

---

## **Section 4: Tool to provide the forecast to the respective households**

In the previous sections, we spent much time exploring various models and evaluating their accuracy. Now, it is time to finally build a tool that would be able to provide gas consumption forecast to households using our chosen model. For this, we have decided to build a telegram bot.

### **Section 4.1: About the telegram bot**

Here is quick breakdown about the telegram bot that we have created from scratch solely just for this project.

#### **1. Name of the telegram bot**

The name of the bot is "Pecan street smart meter bot".

#### **2. How can I view the bot?**

Simply click on this link <https://t.me/PecanStreetGasMeterBot>. Then follow the user guide in

section 4.2 to use the various functionalities of the telegram bot.

### **3. Purpose of the bot**

The purpose of the telegram bot is for the gas provider to communicate with residents and provide various services for them to use, such as viewing past consumptions as well as even predicting future consumptions. This also serves to demonstrate one of the applications of building forecast model. We did not want to simply stop at building a forecast model as we wanted to demonstrate what a gas provider can do with the gas forecast model that can benefit both the gas provider and the households. As such, we built a functioning MVP that a gas provider can create with useful features such as providing future gas forecast to households, viewing past consumption as well as reporting problems.

### **4. Features of the bot**

The functionalities of the bot are shown below:

- `/login` allows the user to login with their household ID and passwords before they can access other services provided by the bot.
- `/check_consumption` displays a chart showing past gas consumption meter readings for residents to view or verify.
- `/get_daily` displays a chart showing predicted gas consumption up to 2359hrs of that day for a particular household together with the respective predicted bill that the resident might need to pay
- `/get_next_day` displays a chart showing predicted gas consumption for next 24 hours for a particular household together with the respective predicted bill that the resident might need to pay
- `/get_weekly` displays a chart showing predicted gas consumption till coming Monday for a particular household together with the respective predicted bill that the resident might need to pay
- `/get_next_week` displays a chart showing predicted gas consumption for next 7 days for a particular household together with the respective predicted bill that the resident might need to pay
- `/get_monthly` displays a chart showing predicted gas consumption till the end of the month for a particular household together with the respective predicted bill that the resident might need to pay
- `/get_next_month` displays a chart showing predicted gas consumption for the next 30 days for a particular household together with the respective predicted bill that the resident might need to pay
- `/report_problem` provides option for residents to feedback or complain about anything to the gas provider

To look at how to use the various functionalites, please look at the user guide in section 4.2

### **5. Why a telegram bot?**

The main reason was because Telegram is a popular messaging application and provides an API for us to use to create Telegram bots. Coupled with a cloud service such as Heroku, we are able to effectively use the Bot as a medium of communication between the gas provider and the residents. Since the Telegram API and our machine learning models both used Python as a programming language, it was easy to integrate the components together to form a working product.

## **6. How will the bot benefit stakeholders**

For the residents, the benefits are numerous. Firstly, they now have an easier way to know their past and predicted future gas consumption, throughout the whole day, 24/7. With that, they would be able to monitor and control their gas consumption accordingly. In addition, there is also an option for them to feedback or complain any matters to the company through the bot. This will provide an alternative and much convenient way for residents to inform the gas provider about any issues that they encounter.

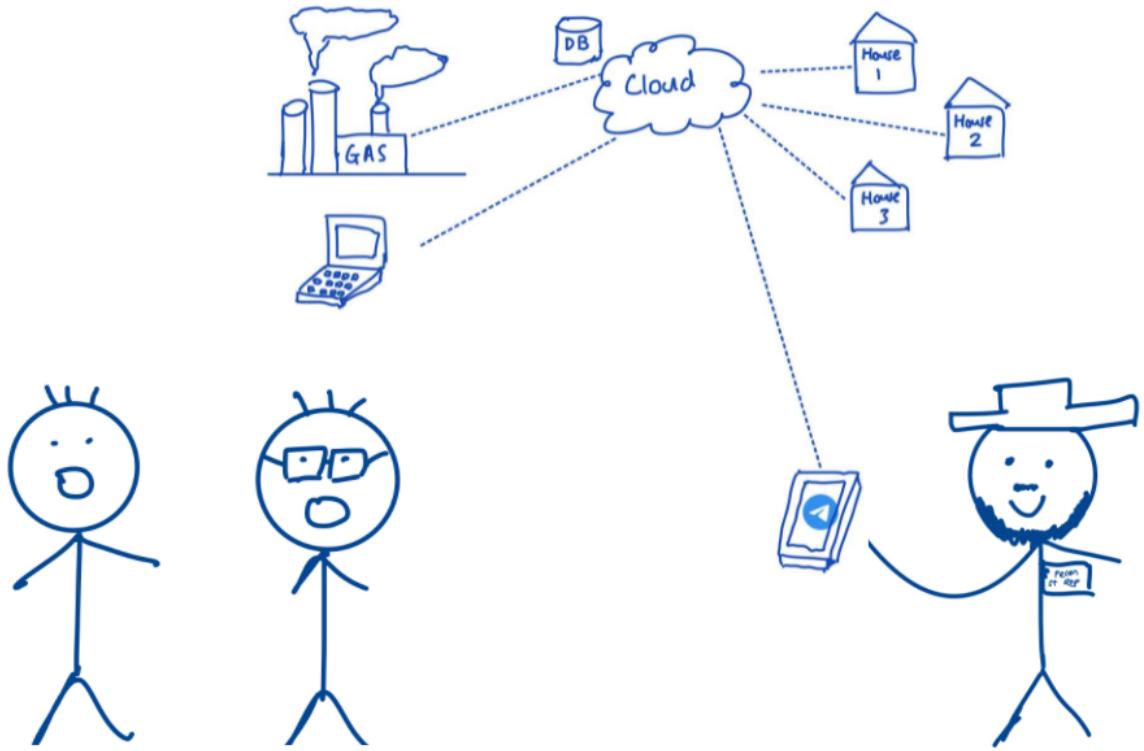
For the gas providers, this bot provides a good opportunity to communicate and connect with residents. It allows them to hear and better understand any issues/concerns that residents might have. With the bot, the gas provider can easily come up with various demand response measures to change consumer patterns to better balance the demand for gas throughout the day. For instance, if the forecast projects that more people are consuming gas during the peak hours, the gas provider might come up with campaigns such as discounts and flexible pricing to encourage more people to consume gas during off-peak hours. The bot would be the perfect place for the gas provider to inform these campaigns to the people and for the people to sign up for the campaigns and/or ask any questions that they might have. On the long run, this will allow the gas provider to better balance demand with supply which will allow it to reduce wastage, costs and energy. More importantly, in business perspective, such a bot would be able to project the gas provider to more people and appeal to them. This will allow the gas provider to build a good rapport with the people which will in turn bring in new consumers and grow their business and increase their revenue streams.

## **7. What model are we using to provide gas forecast?**

As could have seen already, in order to provide the forecast to households, we need to use a forecast model. And that is why over the last two sections, we were looking at various models to see which one would give the highest accuracy.

After looking at all the models, we have decided to use the LSTM model for our bot. This is mainly due to good accuracy compared to other models even though LSTM models take a lot of time and computation to train.

## **8. How does the bot works?**

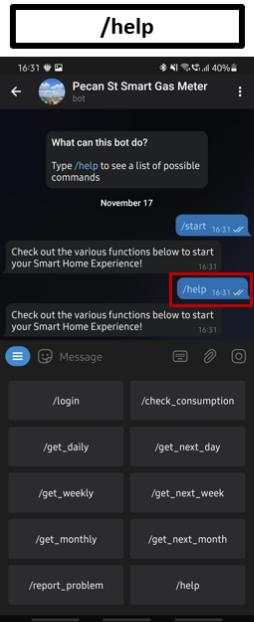


Probably the above image explains all. The bot is connected to the cloud where the chosen LSTM model for the respective households are saved too. When a resident uses the bot, he/she will first have to login. From the login information, the bot will get the user's household ID information from a PostgreSQL database and for subsequent queries, will respond to their commands accordingly with the aid of the resident's respective household model.

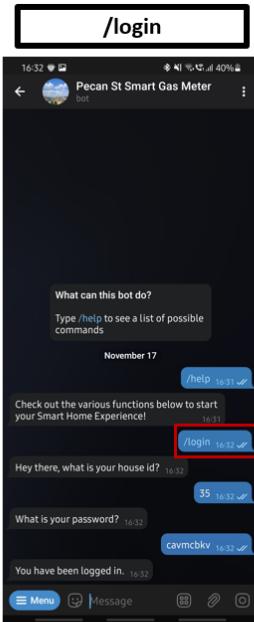
## Section 4.2: User guide

The below images show the functionalities available. At the top of the image, you will see the respective command and at the bottom, you will see a short description of what it does.

**Note** Before you can use the functionalities, the bot will require you to login with a household ID and the respective password. The list of all the household IDs and their respective passwords can be seen in the "password\_list.csv" file under the "Python Telegram Bot" zip file.



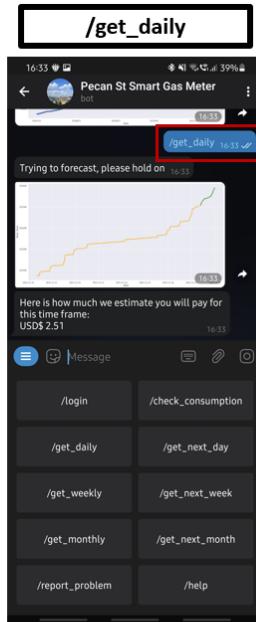
For residents to see the available options



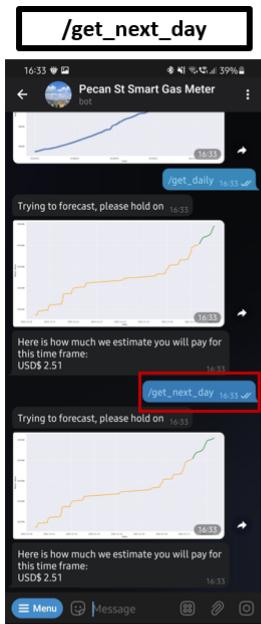
For residents to login



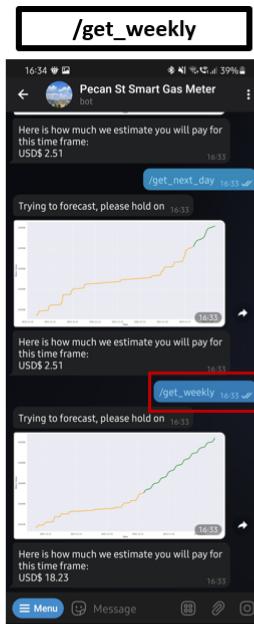
To check previous gas consumption



To see prediction till 2359hrs of that day



To see prediction for next 24 hours



To see prediction till coming Monday

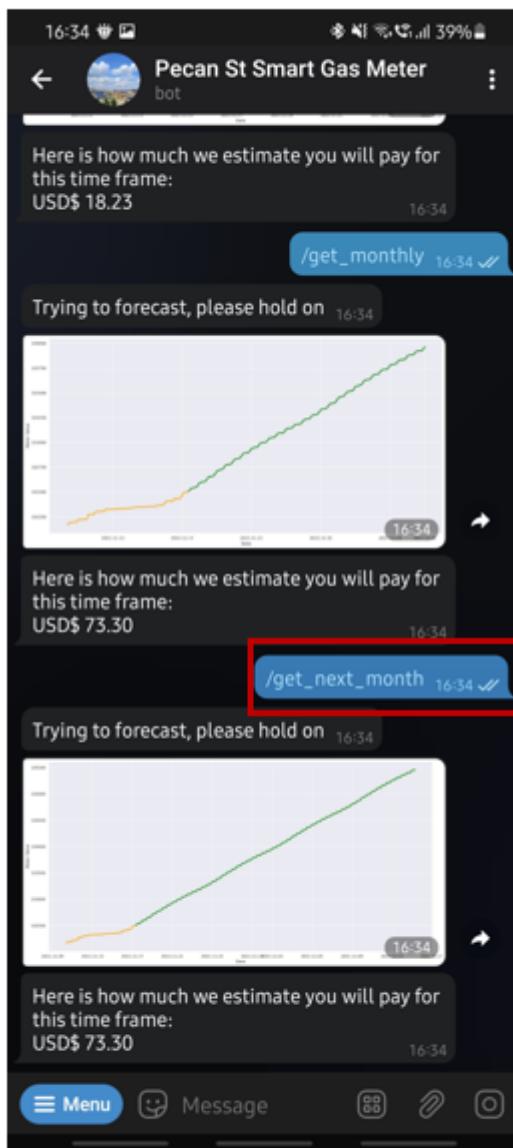


To see prediction for next 7 days



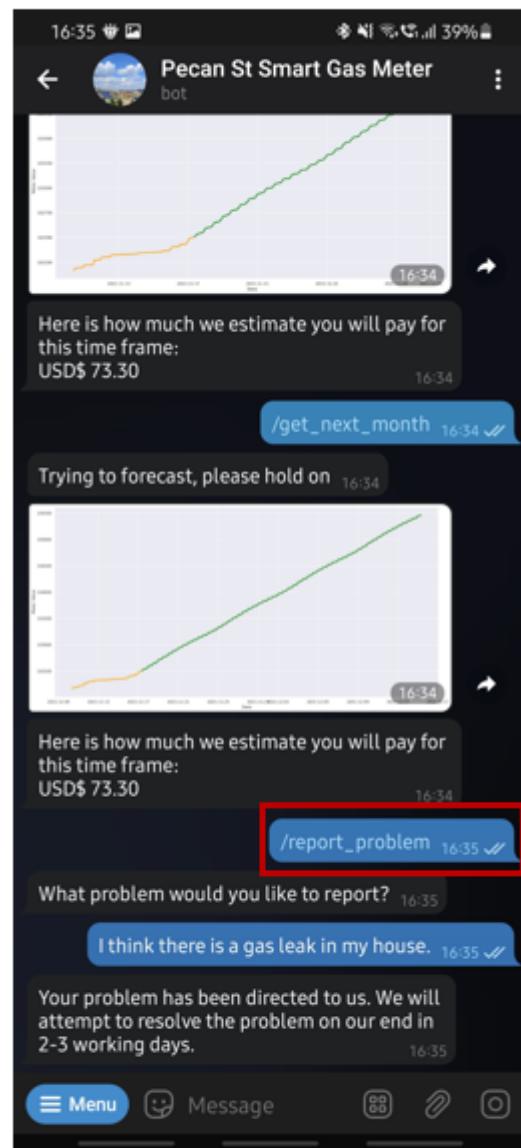
To see prediction till end of month

## /get\_next\_month



To see prediction for  
next 30 days

## /report\_problem



To report problem to  
the gas company

### Section 4.3: Some ethical considerations

Not all consumers may like the idea of the gas provider collecting their gas readings and building a model based on that. Some may consider it to be private and confidential data. Hence, it is important that in real-life, a gas provider seeks the permission of the consumers before collecting data and building a model.

There is also the risk of unauthorised access to the bot. A third party or adversary may obtain the households data and models illegally and then proceed to either leak or use it for unintended purposes. Such an issue may make people dislike the bot and lose their trust on the gas provider and can lead to serious consequences such as filing police report or deciding to switch to another gas provider. Hence, in real life, it is important that the gas provider or vendor running the bot enforce strict measures to ensure no unauthorized access to third parties.

## Section 4.4: Limitations, possible problems & how can it improved?

Our bot does have some limitations and may pose problems. After all, our bot is meant to be a prototype for demonstration purposes to showcase one of the application of forecast models. It was not intended for immediate public release. Here, we will discuss some of the issues with our bot and possible ways they can be resolved.

### **1. Can we guarantee that the predictions are or close to 100% right?**

Unfortunately no. As discussed in the previous sections, we were not able to achieve full accuracy but decent performance. This was due to certain things, some that were beyond our capabilities such as lack of data. That being said, we assume that a gas provider with more resources and capabilities would be able to overcome the issues that we faced, to come up with a better forecast model than us. Even then, forecasts may not be perfect all the time. As such, before launching fully to everyone, the gas provider may decide to conduct trials with selected groups of people to test the accuracy of the models, tweak or change the models accordingly and then release the model when the gas provider is more confident about the accuracy of the model.

### **2. Old data was used to build the forecast model**

The data that we were provided is between October 2015 and March 2016. Hence, to create the model for the bot and to predict for current year, we treated the old data as though it was collected recently (i.e between 2020 and 2021). This is clearly wrong. However, as we have discussed, we were unable to obtain recent data from Pecan Street and we only wanted to build a simple prototype for demonstration purposes. This will probably not be an issue in real world as we assume that a gas provider would not even consider building the forecast model and the bot if it does not have enough current data at the first place.

### **3. Anyone with the household credentials can login and view information illegally**

This would be a serious issue in real life. Hence, if such a bot were to be deployed in real life, a better authentication would be needed. Perhaps something like 2FA might work. Or the gas provider may decide to host the service on their website or a mobile application instead of a telegram bot. These are just some ideas and depending on the requirements and cost, different gas providers or vendors may decide to implement different things.

In addition, there are other issues such as need for fast response, scalability and etc which all could impose additional requirements or constraints and require different solutions/implementations.

---

## Section 5 Quick summary, closing remarks and reflections

### Section 5.1: Quick summary

In this report, we want to predict a household's future gas consumption based on that household's past consumption.

In section 1, we talked about the problem, motivation behind it and lastly, we broke down the problem into two subtasks.

The first subtask was finding a good model that could accurately forecast household gas consumption. We already had a forecast model that we created in question 2. However, we were not totally delighted with its accuracy performance. So we came up with two ideas. The first idea was adding more useful attributes to the dataset with the belief that this would improve the accuracy of the model. More specifically, we added some weather attributes to the dataset, created a new linear regression and neural network model. There was not much improvement to the accuracy. Section 2 discussed this in great detail.

Our second idea was to create a Long Short-Term Memory (LSTM) network model as it is suitable for timeseries data. We prepared the data with respect to the requirement of the LSTM model and trained accordingly. There was a noticeable increase in accuracy. Section 3 discussed this in great detail.

With that, we moved on the next subtask which was to create a tool that can retrieve the gas forecast from our chosen model and provide this forecast to the respective households. We didn't want to simply stop at building a good forecast model and wanted to take a step further to demonstrate what a gas provider can then do with the gas forecast model that can benefit both the gas provider and the households. So, we have created a telegram bot from scratch, just for this project, that can provide various services to residents such as sending households their respective gas consumption forecast, from our chosen model. Our chosen forecast model was the LSTM model that we created in section 3. We provided details about the bot along with the user guide and ethical considerations and limitations. Details about the bot are written in section 4 and also presented in the video presentation.

## Section 5.2: Closing remarks (limitations and applicability)

### Limitations

Throughout the report, there were quite a few places where we mentioned certain limitations that we encountered during the project. Here, we will aim to consolidate some of the important ones for ease of convenience.

#### **1. No large dataset**

As discussed before, we felt that the dataset provided was not large. For instance, if there were data from more households from the same neighbourhood, that could have increased the performance of the models. In addition, if data during the summer and spring seasons were collected and used in the model, that could have increased the accuracy as well.

#### **2. Difficult to obtain suitable attributes**

As discussed in section 2, it was difficult to find suitable attributes that could add value to the model. These attributes were either not readily available or were from unreliable sources.

### Applicability

The models that we have created may not be perfect and for certain households, may give wrong predictions. Nonetheless, we feel that, overall, the models have decent accuracy. And with better funetuning of hyperparameters and more data, the accuracy of the model can be further improved.

As for the bot, it may not meet certain real life constraints and problems. Then again, the bot was meant to be simple prototype to showcase our idea. As such, certain issues were beyond the scope of the project as well. Nonetheless, we are satisfied with how the bot has come out.

## Section 5.3: Reflections (challenges, mistakes and lesson learnt)

### Challenges

- ideation of the project, where it took some time before we found a solution that improves on the previous part of the project since we are continuing to build on the idea of forecasting which was already done in question 2
- creating a good forecast model with good performance. We had to spent a lot of time researching and discussing before finalizing our ideas.
- Our project stood atop the shoulders of giants, where we had to follow quite a few different online resources to understand and create a usable LSTM model, since LSTM and deep learning was not something that was explicitly covered in class.
- For the bot, creating a skeleton bot first to check for plausibility was challenging and took a lot more time and effort than expected.
- We were initially unable to deploy our tensorflow model due to slug size limits on Heroku. However, we managed to overcome this using Google Cloud.

### Mistakes

- For the LSTM model, initially we tried to train and evaluate it using raw meter\_values, but soon found that the model was unable to extrapolate values. However, after transforming the data into usage domain, we managed to get quite a good model. From this, we learnt that processing data before hand and looking at the data from a slightly different perspective can make a huge difference in the world of data science.

### Lessons learnt

- Throughout the project, there were many limitations and constraints that we had to overcome. Reflecting about them, we now realize that the issues that we faced are not very different from what data scientists or corporate companies that build forecast models, face everyday. Problems such as insufficient dataset or lack of data for suitable attributes are some of the challenges that almost every professional in the industry face. As such, after going through similar problems, we now have a better understanding and appreciation about some of the challenges in data science. And through the project, we believe that we have acquired more knowledge and experience about how we can tackle them. This is really useful especially for those of us who intend to pursue a career in data science.
- On the similar note, through the project, we have become more proficient in programming, analytics, modeling, data visualizations and machine learning methods. This is once again very useful as these are some of the must-have skills that every data scientist or someone applying for a data science job should possess.
- Through the project, we have also learnt the importance of soft skills such as communication, teamwork and resilience. These soft skills that we have developed will be very helpful in our future workplace regardless of the industry that we are working.

## Section 5.4: Ideas for the future

- **Comprehensive study on factors that affect household gas consumption:** One idea is to conduct a study to look at different possible attributes that can have an impact on gas consumption and publish a report or literature study about it. If it is possible to collect data during the study, that would be better as well, as those data can be used to build a forecast model with better accuracy.
- **Post study on effectiveness of the bot in reducing unwanted gas consumption:** One of the aim of the bot was to reduce unwanted usage of household gas consumption as we believe that through the bot, people would constantly monitor their gas consumption and aim to reduce their consumption and cost in the long run. It would be good to conduct a study/survey on how effective the bot has managed to achieve this. If the gas provider had implemented demand response program in addition to the bot, the post study/survey can also include people's reaction and behaviour change after those measures. With the collected data, we can then even model gas consumption with respect to the measures implemented to find the optimal/best measures that would have the maximum desirable impact.