EE4211 Project

Team 1: Quadratic Done by Choi Jae Hyung, Lee Min Young, Nicholas Lui Ming Yang, Tran Duy Anh

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import datetime
from scipy import stats
from functools import reduce

import sklearn.linear_model as lm
import sklearn
from sklearn.linear_model import LinearRegression as LR
from sklearn.svm import SVR
```

In [2]:

```
df = pd.read_csv("project_data.csv")
# df.shape - (1584823, 3)
# df column values:
# localminute - Timestamp
# dataid - MeterID
# meter_value - meter reading
```

Q1.1

How many houses are included in the measurement study?

In [3]:

```
# Since dataid is unique for each meter, we can count the number of unique dataid numbers
df.dataid.value_counts().size
```

Out[3]:

157

This gives us 157 for the number of houses included in the study

Are there any malfunctioning meters? If so, identify them and the time periods where they were malfunctioning.

There are various ways to define a malfunctioning meter. Let us explore some of them

Let us first check, if there are *meters that report a lower value at a later timestamp*. Since meter_value is cumulative consumption, meter_value should not be lower at a later timestamp for the same meter

In [4]:

```
grouped = df.groupby(['dataid'], sort=['localminute'])
def has_decreasing_values(df):
    current_value = 0
    for index, val in df.iteritems():
        if val < current_value:
            return True
        current_value = val

meters_with_decreasing = (grouped['meter_value']
                          .agg(has_decreasing_values)
                          .where(lambda x: x)
                          .dropna()
                          .keys())
```

In [5]:

```
print(len(meters_with_decreasing))
meters_with_decreasing
```

```
43
```

Out[5]:

```
Int64Index([  35,   77,   94,  483,  484, 1042, 1086, 1185, 1507, 1556, 1718,
            1790, 1801, 2129, 2335, 2449, 3134, 3527, 3544, 3893, 4031, 4193,
            4514, 4998, 5129, 5131, 5193, 5403, 5810, 5814, 5892, 6836, 7017,
            7030, 7117, 7739, 7794, 7989, 8156, 8890, 9134, 9639, 9982],
           dtype='int64', name='dataid')
```

Wow, we have 43 meters that have a decreasing value! Let's zoom in and get the offending sections for each meter. (In other words, find the exact data points which show this descreasing value)

In [6]:

```
## Need to re-sort after filter since filter takes all rows and breaks original sorting
dec_meters = (grouped.filter(lambda x: int(x.name) in meters_with_decreasing)
              .groupby(['dataid'], sort=['localminute']))

## Iterate over values to find offending rows for each meter
## WARNING: RUNS VERY SLOWLY. TODO: OPTIMIZE
offending_values = {}
for group_id, rows in dec_meters.groups.items():
    offending_values[group_id] = []
    current_value = 0
    group_rows = dec_meters.get_group(group_id)
    group_rows_count = group_rows.shape[0]
    for i in range(group_rows_count):
        if group_rows.iloc[i]['meter_value'] < current_value:
            offending_values[group_id].append([group_rows.iloc[i-1], group_rows.iloc[i]])
        current_value = group_rows.iloc[i]['meter_value']
```

Number of 'broken data' instances for each meter

In [7]:

```
print("Meter ID |", "Number of broken readings")
for k, v in offending_values.items():
    print(str(k).ljust(20), len(v))
```

```
Meter ID | Number of broken readings
35                   1
77                   1
94                   6
483                  1
484                  9
1042                 1
1086                 1
1185                 135
```

```
1507                2
1556                12
1718                4
1790                1
1801                1
2129                3
2335                5
2449                93
3134                18
3527                1
3544                18
3893                2
4031                16
4193                1
4514                141
4998                1
5129                76
5131                1
5193                4
5403                156
5810                10
5814                1
5892                1
6836                51
7017                1
7030                90
7117                123
7739                1
7794                1
7989                2
8156                151
8890                44
9134                115
9639                2
9982                2
```

Just knowing the number of broken readings may not be useful.

Let's say that we want to know which meters should be fixed, since faulty meters result in us inaccurately measuring consumption, and possibly losing money.

There should be some measure of tolerance in deciding if a meter is broken. In this case, let's check the average decreasing amount, and the ratio of "broken" readings

In [8]:

```
print("Meter ID |", "Number of broken readings |", "Average decrease across broken readings |", "Percentage of broken readings for meter")
for k, v in offending_values.items():
    print(str(k).ljust(20),
          str(len(v)).ljust(20),
          str(reduce(lambda x, y: x + abs(y[1]['meter_value'] - y[0]['meter_value']) / len(v), [0] + v)).ljust(40),
          (len(v) / dec_meters.get_group(k).shape[0]) * 100)
```

```
Meter ID | Number of broken readings | Average decrease across broken readings | Percentage of broken readings for meter
35                  1                   2.0                                     0.008423180592991915
77                  1                   2.0                                     0.009360666479453337
94                  6                   9.0                                     0.016513003990642632
483                 1                   14.0                                    0.003619516432604603
8
484                 9                   2.4444444444444446                      0.020438751873552256
1042                1                   2.0                                     0.02610966057441253
1086                1                   2.0                                     0.003330114222917846
1185                135                 16267.674074074097                      0.7314694408322496
1507                2                   2.0                                     0.006134404809373371
1556                12                  13574.0                                 0.3252032520325203
1718                4                   5.0                                     0.016346546791990192
1790                1                   2.0                                     0.007494004796163069
1801                1                   2.0                                     0.006292474200855777
2129                3                   2.0                                     0.021759628635671283
2335                5                   22942.0                                 0.05611672278338946
```

```
2449                93                  15472.860215053784                  1.7067351807671134
3134                18                  12877.222222222224                  0.4480955937266617
3527                1                   2.0                                 0.009214042200313279
3544                18                  8488.222222222223                   0.8104457451598379
3893                2                   2.0                                 0.007450454477723141
4031                16                  2.875                               0.1276527844263603
4193                1                   2.0                                 0.09813542688910697
4514                141                 23971.8156028369                    0.7392261717521234
4998                1                   14.0                                0.007156147130385
5129                76                  11116.815789473698                  1.6941596076683014
5131                1                   2.0                                 0.00658457891617831
5193                4                   5.0                                 0.02055076037813399
5403                156                 10897.37179487177                   0.6103525177041355
5810                10                  4.6000000000000005                  0.023677605720509542
5814                1                   2.0                                 0.002357156326607580
4
5892                1                   2.0                                 0.007072635971426551
6836                51                  11397.17647058824                   1.1283185840707965
7017                1                   4.0                                 0.003955852684046046
5
7030                90                  17217.15555555556                   0.5023723137036004
7117                123                 16809.414634146335                  0.6002049480310351
7739                1                   2.0                                 0.022558087074216106
7794                1                   6.0                                 0.011724703951225232
7989                2                   26.0                                0.005034866450167409
8156                151                 16260.622516556286                  0.596932321315623
8890                44                  2.6363636363636354                  0.26547604682032094
9134                115                 32439.008695652185                  0.8176905574516496
9639                2                   29376.0                             0.014496955639315743
9982                2                   14036.0                             0.12987012987012986
```

With the data laid out like this, it is pretty clear that most meters are not really broken if we allow a 2%error rate

However, in terms of error volume, some are pretty suspect. Let's use an average volume error of 100 Cubic foot (that's a lot!) as our threshold, and filter our results

In [9]:

```
print("Meter ID |", "Number of broken readings |", "Average decrease across broken readings |", "Percentage of broken readings for meter")
for k, v in offending_values.items():
    measure = str(reduce(lambda x, y: x + abs(y[1]['meter_value'] - y[0]['meter_value']) / len(v), [0] + v )).ljust(40)
    if float(measure) > 10:
        print(str(k).ljust(20),
              str(len(v)).ljust(20),
              measure,
              (len(v) / dec_meters.get_group(k).shape[0]) * 100)
```

```
Meter ID | Number of broken readings | Average decrease across broken readings | Percentage of broken readings for meter
483                 1                   14.0                                0.0036195164326046038
1185                135                 16267.674074074097                  0.7314694408322496
1556                12                  13574.0                             0.3252032520325203
2335                5                   22942.0                             0.05611672278338946
2449                93                  15472.860215053784                  1.7067351807671134
3134                18                  12877.222222222224                  0.4480955937266617
3544                18                  8488.222222222223                   0.8104457451598379
4514                141                 23971.8156028369                    0.7392261717521234
4998                1                   14.0                                0.007156147130385
5129                76                  11116.815789473698                  1.6941596076683014
5403                156                 10897.37179487177                   0.6103525177041355
6836                51                  11397.17647058824                   1.1283185840707965
7030                90                  17217.15555555556                   0.5023723137036004
7117                123                 16809.414634146335                  0.6002049480310351
7989                2                   26.0                                0.005034866450167409
8156                151                 16260.622516556286                  0.596932321315623
9134                115                 32439.008695652185                  0.8176905574516496
9639                2                   29376.0                             0.014496955639315743
9982                2                   14036.0                             0.12987012987012986
```

In [10]:

```
"""Use this function to validate/check each bad meter given our heuristic"""
def print_bad_meter_readings(meterID):
    meter_readings = offending_values[meterID]
    print("time apart |".ljust(20), "meter value initial |", "meter value after |", "difference")
    for readings_pair in meter_readings:

        print(str(pd.to_datetime(readings_pair[1]['localminute']) - pd.to_datetime(readings_pair[0]['lo
calminute'])).ljust(25),
              str(readings_pair[0]['meter_value']).ljust(20),
              str(readings_pair[1]['meter_value']).ljust(20),
              readings_pair[1]['meter_value'] - readings_pair[0]['meter_value'])
print_bad_meter_readings(1556)
```

```
time apart |          meter value initial | meter value after | difference
0 days 00:00:13           203154               203152               -2
0 days 00:06:59           203580               203578               -2
0 days 01:52:10.446150    223266               206892               -16374
0 days 00:15:09.023424    223266               206892               -16374
0 days 00:01:10.125932    223268               206910               -16358
0 days 02:14:09.533295    223282               206986               -16296
0 days 00:21:08.597335    223304               207046               -16258
0 days 01:39:08.641617    223306               207048               -16258
0 days 01:41:09.602248    223318               207066               -16252
0 days 00:46:07.782113    223320               207080               -16240
0 days 01:13:08.190968    223332               207094               -16238
0 days 00:14:06.456241    223340               207104               -16236
```

Another requirement of the meters is that they push their readings to be saved if the meter values differ by at least 2 cubic foot

Let us verify that every pair of readings for a meter differ by at least 2 cubic foot. To not double count, we will only check for readings where the differing value is 0 >= x > 2 so that we don't get the same error readings where the readings decrease

Since the value is smaller, we will not focus on the difference in values, but on the percentage of total readings for the meter only

In [11]:

```
grouped2 = df.groupby(['dataid'], sort=['localminute'])
def has_stagnant_values(df):
    current_value = 0
    for index, val in df.iteritems():
        if index == 0:
            current_value = val
            continue

        if val < current_value + 2 and val >= current_value:
            return True
        current_value = val

meters_with_stagnant = (grouped['meter_value']
                        .agg(has_stagnant_values)
                        .where(lambda x: x)
                        .dropna()
                        .keys())

print("Number of meters with stagnant values: ", len(meters_with_stagnant))
```

```
Number of meters with stagnant values:  155
```

The following segment should have calculated the percentage of stagnant values for each meter, but it takes too long (tried for 5 minutes and gave up). At least we know number of meters that reported stagnant values?

In [12]:

```
# ## Need to re-sort after filter since filter takes all rows and breaks original sorting
```

```
# stagnant_meters = (grouped2.filter(lambda x: int(x.name) in meters_with_stagnant)
#               .groupby(['dataid'], sort=['localminute']))

# ## Iterate over values to count offending occurences. Not counting value
# ## WARNING: RUNS VERY SLOWLY. TODO: OPTIMIZE
# offending_values2 = {}
# for group_id, rows in stagnant_meters.groups.items():
#     offending_values2[group_id] = 0
#     group_rows = stagnant_meters.get_group(group_id)
#     group_rows_count = group_rows.shape[0]
#     # Set current_value so we do not trigger first reading
#     current_value = group_rows.iloc[0]['meter_value'] - 5
#     for i in range(group_rows_count):
#         if group_rows.iloc[i]['meter_value'] < current_value + 2 and group_rows.iloc[i]['meter_value'
] >= current_value:
#             offending_values2[group_id] += 1
#         current_value = group_rows.iloc[i]['meter_value']


# print("Meter ID |", "Number of broken readings |", "Percentage of broken readings for meter")
# for k, v in offending_values2.items():
#     print(str(k).ljust(20),
#           str(v).ljust(20),
#           (v / stagnant_meters.get_group(k).shape[0]) * 100)
```

In [13]:

```
stagnant_meters = (grouped2.filter(lambda x: int(x.name) in meters_with_stagnant)
              .groupby(['dataid'], sort=['localminute']))

print("Total number of readings to iterate, which is why it is slow: ",
      reduce(lambda x, y: x + len(y), [0] + list(stagnant_meters.groups.values())))
```

```
Total number of readings to iterate, which is why it is slow:  1584818
```

We now have two different criterion for deciding what constitues a broken meter, and the readings that helped us determine that. Since the question asks for "time periods where they were malfunctioning.", let's demonstrate that we can consolidate broken readings into time periods

In [14]:

```
# let's use the meter with the most broken readings (based off decreasing values)
most_broken_values_meter = grouped.get_group(5403)
```

In [15]:

```
# We are using a stricter requirement now. Let's consider that, if the subsequent meter reading is less

# than an increment of 2 (meters are only supposed to report after at least an increment of 2)

# Broken_criteria is a helper function (comparator function) that takes in two (in-order) readings and
outputs a
# boolean of whether the later reading is "broken"

broken_criteria = lambda x, y: y['meter_value'] < x['meter_value'] + 2
```

In [16]:

```
broken_readings = 0
num_readings = most_broken_values_meter.shape[0]
for i in range(1, num_readings):
    if broken_criteria(most_broken_values_meter.iloc[i-1], most_broken_values_meter.iloc[i]):
        broken_readings += 1

print("Number of broken readings for meter 5403, based on stricter requirements: ", broken_readings)
```

```
Number of broken readings for meter 5403, based on stricter requirements:  21371
```

In [17]:

```
# Let's now create a function that can aggregate a meter's readings into "broken" time periods
# Return value of the function will be as follow:
"""
2D array.
First dimension is the time periods.
Second dimension is the consecutive broken readings that make up the broken time period
To find the actual time data, take the ['localminute'] attribute from the first and last reading per pe
riod
    time_periods = [
        [reading_1, reading_2, reading_3],
        [reading_1, reading_2]
    ]
"""
def get_broken_time_periods(broken_criteria, meter_readings):
    num_readings = meter_readings.shape[0]
    time_periods = []
    temp_period = []
    for i in range(1, num_readings):
        if broken_criteria(meter_readings.iloc[i-1], meter_readings.iloc[i]):
            temp_period.append(meter_readings.iloc[i])
        else:
            if temp_period:
                time_periods.append(temp_period)
                temp_period = []
    if temp_period:
        time_periods.append(temp_period)
    return time_periods
```

In [18]:

```
broken_time_periods = get_broken_time_periods(broken_criteria, most_broken_values_meter)
print("Number of broken time periods: ", len(broken_time_periods))
```

```
Number of broken time periods:  3545
```

Q1.2

Hourly Data Collection and Plotting the Data

Objective:

Obtain hourly data from a list of given data, and plot the obtained data

Parameters:

1. Gas meter ID

Gas meter with ID 739 will be focused for the study

2. Starting time

6th Oct 2015, 12 a.m.

3. Ending time

5th Nov 2015, 11 p.m.

Assumption (How we aggregate the data into hourly readings)

- The data is always cumulative: increasing in value as time goes.

The lowest value in an hour should be the closest value to an exact hour (hh:00:00) and a good estimate of that hour's data.

Therefore, the first data point in an hour is used to represent the hourly data. (The assumption is that we are interested in

approximating the consumption amount at the start of the hour)

How to handle missing data?

As the data is cumulative gas consumption by household, when there is no data, it can be assumed as there is insignificant gas consumption between that period.

Hence, the missing data will be given same value as most recent hourly data obtained.

Bad data readings

This plot of hourly readings does not account for "bad" data readings (consecutive reading with a lower value, which should not happen for cumulative consumption data)

The likelihood of seeing an hourly data point that is bad is Average P(bad readings an hour) *(4*60), since we have max 4 readings per minute

This reduces the code complexity to get a plot, and the usefulness of this approach depends on the usage of this plot

If we are simply looking for a general consumption level visualisation, then this plot should suffice. However, if we are looking for plots with a decrease in value to identify faulty meters, then this plot is insufficient. For that, we cannot hold our original assumption (that readings are cumulative and strictly increasing), and should take the MIN(hour_values) to represent that hour

Code

The algorithm of hourly data collection consists of the functions below.

1. In every new hour, collect the lowest value as the hourly data.
1. Look out for missing hour. When missing hour is found, previuos hour data is given for that hour.

Hourly Data Collection

Create DateHour column, that represents "Date + Hour" of the data.

In [6]:

```
#change to datetime format
df["localminute"] = pd.to_datetime(df["localminute"])
#get a dataframe of data by hour:
df_byhour = pd.DataFrame(df.groupby([df["localminute"].dt.date.rename("Date"),
                                     df["localminute"].dt.hour.rename("Hour"),
                                     df["dataid"]]).min().reset_index())

# To preview df_byhour sorted by meter and time
# df_byhour = df_byhour.sort_values=(by=['dataid', 'Date', 'Hour'])

#add a column called "DateHour" in the dataframe:
df_byhour['DateHour'] = pd.to_datetime(df_byhour['Date'].apply(str)+' '+df_byhour['Hour'].apply(str)+':
00:00')
```

Choose the meter_ID, starting time and ending time of the hourly data collection.

In [7]:

```
#Format: ID No.
ID = 739

#ranges from 06/10/2015 00:00:00 to 05/11/2015 23:00:00
start_date = datetime.date(2015, 10, 5)
end_date = datetime.date(2015, 11, 6)
filteredDF = df_byhour[(df_byhour["Date"] > start_date) & (df_byhour["Date"] < end_date)]
```

Collect hourly data.

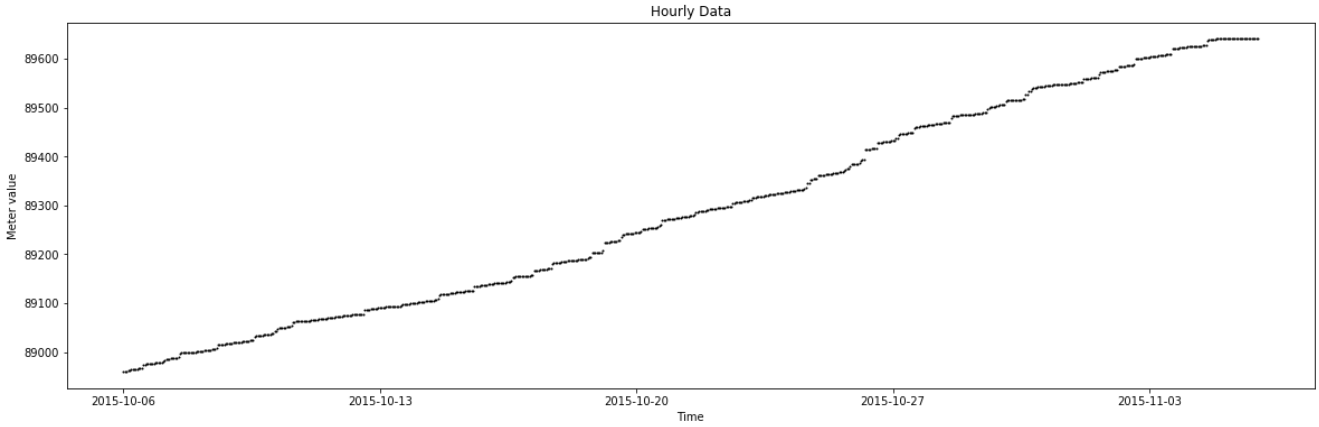
In [8]:

```
#from 06/10/2015 -> 06/11/2015 there are 31 days -> 744 data points
prevHr = pd.Timestamp('2015-10-06 00:00:00')
endHr = pd.Timestamp('2015-11-05 23:00:00')
#generate a temporary dataframe through get_group function with each id
dfTemp = filteredDF.groupby(['dataid']).get_group(ID)
#get the list of hours and meter values
hourList = dfTemp['DateHour'].tolist()
valList = dfTemp['meter_value'].tolist()
#re-initialize the prev val as the first value of the val list
prevVal = valList[0]
#re-initialize the list of values as a list with 1 item
newList = [valList[0]]
#loop through each hour in the list
for index, hr in enumerate(hourList):
    #if hour is more than an hour bigger than previous
    while (hr - prevHr).seconds >= 3600:
        newList.append(prevVal)
        #increment every hour
        prevHr += pd.Timedelta(seconds=3600)

    #update the prev value everytime one index is passed
    prevVal = valList[index]

#some data does not reach until the end date, thats where this comes in
while prevHr < endHr:
    newList.append(prevVal)
    #increment every hour
    prevHr += pd.Timedelta(seconds=3600)
```

Plot the hourly reading.

In [12]:

```
x = pd.date_range(datetime.date(2015, 10, 6), datetime.date(2015, 11, 6), freq = 'H').tolist()
#get rid of the last hour which is 00:00:00 06/11/2016
x.pop()
y = newList

fig, ax = plt.subplots(1, 1, figsize=(20, 6))
ax.plot(x, y, 'ok', ms=1)
plt.xlabel ("Time")
plt.ylabel ("Meter value")
ax.set_title('Hourly Data')
plt.show()
```



Q1.3 Find for each home, 5 houses with highest correlation

Generate dictionary of hourly data (with missing hourly data filled for every house over the 6 months).

In [13]:

```
#get a list of meter id
idList = df['dataid'].unique()

#there are 183 days => 4392 hours => 4392 readings in each list
#generate a dictionary
hourlyDataDict = {}

#define start hour and end hour
staHr = pd.Timestamp('2015-10-01 05:00:00')
endHr = pd.Timestamp('2016-04-01 04:00:00')

#loop through each id
for id1 in idList:

    #re-innitialize the prev Hour as start hour
    prevHr = staHr
    #generate a temporary dataframe through get_group function with each id
    dfTemp = df_byhour.groupby(['dataid']).get_group(id1)
    #get the list of hours and meter values
    hourList = dfTemp['DateHour'].tolist()
    valList = dfTemp['meter_value'].tolist()
    #re-initialize the prev pal as the first value of the val list
    prevVal = valList[0]
    #re-initialize the list of values as a list with 1 item
    newList = [valList[0]]
    #loop through each hour in the list
    for index, hr in enumerate(hourList):
        #if hour is more than an hour bigger than previous
        while (hr - prevHr).seconds >= 3600:
            newList.append(prevVal)
            #increment every hour
            prevHr += pd.Timedelta(seconds=3600)

        #update the prev value everytime one index is passed
        prevVal = valList[index]

    #some data does not reach until the end date, thats where this comes in
    while prevHr < endHr:
        newList.append(prevVal)
        #increment every hour
        prevHr += pd.Timedelta(seconds=3600)

    #after all that create a new entry in the Dictionary with the key as the id and value is the list o
f meter readings
    hourlyDataDict[id1] = newList
```

Creating a Dictionary of Correlation Coefficient.

In [14]:

```
corDict = {}
#loop through id List to get the first id
for id1 in idList:
    #create the empty value for the first key
    corDict[id1] = {}
    #loop through the id list to get the second id
    for id2 in idList:
        # generate the coefficient through numpy.corrcoef (can be changed later)
        coef = np.corrcoef(hourlyDataDict[id1], hourlyDataDict[id2])[0, 1]
        # assign the value to the dict with the appropriate key
        corDict[id1][id2] = coef
        #print("coefficient between " + str(id1) + " and " + str(id2) + " is: " + str(coef))
```

Now, let's find the top 5 houses with highest correlation value for each house

In [15]:

```
from collections import Counter

idList.sort()
top5dict = {}

for id in idList:

    #### This is if you just want the houseID for each (first column being the ID)

#     print(sorted(corDict[id], key =corDict[id].get, reverse=True)[:6])

    #### This is if you want to get the houseID + the value of correlation coefficient
    c = Counter(corDict[id])
    mc = c.most_common(6)
    del mc[0]

    top5dict.update({id : mc})

top5dict
```

Out[15]:

```
{35: [(5810, 0.9995398065544199),
  (1697, 0.999470909229 81),
  (7287, 0.9992099210183376),
  (5484, 0.9991133136434337),
  (7682, 0.9989979036772751)],
 44: [(6578, 0.9928492095318779),
  (1103, 0.9919935650289411),
  (8386, 0.9859651669698513),
  (2818, 0.9832848564504032),
  (7965, 0.9745610836315821)],
 77: [(9849, 0.9985861469650851),
  (1283, 0.9965229176976992),
  (1697, 0.9963737653857886),
  (35, 0.9953255843038066),
  (484, 0.9945619840199484)],
 94: [(2335, 0.9996164386486648),
  (1507, 0.999510767798402),
  (6910, 0.9995059133899636),
  (4732, 0.9992178223073451),
  (4514, 0.9992136482595508)],
 114: [(5814, 0.9996132009008888),
  (2094, 0.9995866454618761),
  (739, 0.9995103862045776),
  (5972, 0.9994932092663096),
  (5636, 0.9994824244256911)],
 187: [(4228, 0.9939913920973358),
  (8386, 0.9915229891078268),
  (9766, 0.985990598911649),
  (222, 0.9846558520509291),
  (2470, 0.9838320924661318)],
 222: [(2470, 0.9947291700518065),
  (4373, 0.9944662802109768),
  (9766, 0.9942769209363104),
  (7016, 0.9924671875344893),
  (3310, 0.9922473972063605)],
 252: [(4732, 0.999820477286879),
  (4514, 0.9998149826712783),
  (3893, 0.9997954632563258),
  (3577, 0.9997427809469283),
  (5814, 0.9997293991892912)],
 370: [(744, 0.9996874579504336),
  (7900, 0.9994311208990785),
  (3527, 0.999302645691812),
  (1589, 0.9992099525426085),
  (7674, 0.9990250403306783)],
 483: [(3367, 0.9999052779428209),
  (8829, 0.9996866736759017),
  (4998, 0.9996831589235713),
  (1714, 0.9996600891049743),
  (4767, 0.9996519884334283)],
 484: [(5484, 0.99971789908391),
  (1697, 0.9993369672470815),
  (5810, 0.9990065598221305),
```

```
 (35, 0.9987287021909909),
 (94, 0.998515326835341)],
661: [(3778, 0.999722518725858),
 (4356, 0.9996210671527851),
 (1086, 0.9995451429523443),
 (5972, 0.9994979992120004),
 (2129, 0.9993851380857133)],
739: [(5814, 0.9998201992992561),
 (9295, 0.999638740437239),
 (2094, 0.9996344656124968),
 (5972, 0.9995833327343763),
 (9121, 0.9995713825700071)],
744: [(370, 0.9996874579504337),
 (7674, 0.9993775683579453),
 (3527, 0.999361537370371),
 (2335, 0.9993122420499554),
 (2018, 0.9992102784007358)],
871: [(8890, 0.9996550575269669),
 (3577, 0.999639208131035),
 (483, 0.9994753951082331),
 (5131, 0.9994367568235588),
 (4031, 0.9994053076083367)],
1042: [(3849, 0.9988257945440744),
 (7989, 0.9979130307466568),
 (2965, 0.9966615250714466),
 (2378, 0.9959176352040268),
 (2945, 0.9955543444945455)],
1086: [(4356, 0.9997962949214336),
 (5972, 0.9997522832975294),
 (661, 0.9995451429523443),
 (9639, 0.9995201316109045),
 (9121, 0.9994811732524121)],
1103: [(44, 0.991993565028941),
 (7965, 0.9904408409713493),
 (2818, 0.9855387835322502),
 (6578, 0.9853566591149605),
 (9956, 0.9810659926085474)],
1185: [(7794, 0.9969214263284724),
 (5972, 0.9969117048769346),
 (114, 0.9967906427994196),
 (4356, 0.9967411627684567),
 (661, 0.9967380828700477)],
1283: [(1697, 0.9985681988572893),
 (484, 0.9982148711217651),
 (5484, 0.9974890331390898),
 (9849, 0.9972656502841636),
 (35, 0.997063863087497)],
1403: [(2755, 0.9935083281027876),
 (6685, 0.9905459232478045),
 (9160, 0.9899674234959066),
 (9600, 0.9893401091422287),
 (3036, 0.9884932213812098)],
1415: [(3310, 0.9984958878780769),
 (7016, 0.9975654409401622),
 (8467, 0.9959934701971438),
 (2470, 0.9959274673748065),
 (9766, 0.9953874137400498)],
1507: [(6910, 0.9998738439627559),
 (1801, 0.9998486788553647),
 (4732, 0.9998323779871467),
 (2094, 0.9997905300289346),
 (4514, 0.9997883182907749)],
1556: [(5814, 0.9947676362898391),
 (4514, 0.9947337914896599),
 (4732, 0.9947226260606261),
 (3577, 0.9947026469465342),
 (252, 0.99469932167733)],
1589: [(3527, 0.9998479847805064),
 (1714, 0.9995050976691875),
 (7017, 0.999485278286617),
 (3367, 0.9993435608842048),
 (4998, 0.9992188598084749)],
1619: [(7794, 0.9995508541534226),
 (9052, 0.9994413229686985),
 (3723, 0.9992379564882036),
 (2072, 0.9991161094454741),
 (114, 0.9989118408059005)],
```

```
 (114, 0.9991161040088968)],
1697: [(35, 0.99947090922981),
 (484, 0.9993369672470815),
 (5484, 0.9992647589428167),
 (5810, 0.9991853661090624),
 (9849, 0.9986833905451356)],
1714: [(3367, 0.9996926338540278),
 (483, 0.9996600891049744),
 (3527, 0.9996145512071303),
 (4998, 0.9996004472241617),
 (1589, 0.9995050976691875)],
1718: [(1801, 0.9998688550396241),
 (1791, 0.9998220225727514),
 (4514, 0.9997773163976941),
 (2094, 0.9997491878923508),
 (1507, 0.9997145077369125)],
1790: [(3723, 0.999397392734743),
 (739, 0.9993179554937944),
 (5814, 0.9992655721994708),
 (7794, 0.9991883457501407),
 (9639, 0.9991625774217837)],
1791: [(1718, 0.9998220225727513),
 (2094, 0.9997493919925184),
 (1801, 0.9996625845084679),
 (4514, 0.9996501951059205),
 (3893, 0.9995652013614557)],
1792: [(7741, 0.9960788860571494),
 (2378, 0.9947561847195541),
 (3849, 0.9938593066512577),
 (5275, 0.9929959164431413),
 (8155, 0.9925982850412143)],
1800: [(5193, 0.9982824268380607),
 (2980, 0.9982100947315251),
 (3527, 0.998203736843842),
 (2018, 0.9981798729083484),
 (7674, 0.9979448724049851)],
1801: [(1718, 0.999868855039624),
 (1507, 0.9998486788553647),
 (7674, 0.9997795377227343),
 (6910, 0.9997789070940649),
 (2018, 0.999775617338845)],
2018: [(1801, 0.9997756173388449),
 (7674, 0.9997669506855461),
 (6910, 0.9997340839397812),
 (2335, 0.9996428247929605),
 (1507, 0.9996199473759118)],
2034: [(8829, 0.9998517886920095),
 (4031, 0.9998420845213813),
 (9729, 0.999789125474335),
 (7429, 0.999756939274056),
 (4732, 0.9997337885095665)],
2072: [(661, 0.9991359952146371),
 (8086, 0.9991319649112245),
 (1619, 0.9991161094454742),
 (114, 0.99899682916776),
 (7794, 0.9988713731798731)],
2094: [(4514, 0.9999110312147726),
 (3893, 0.9998707735022405),
 (1507, 0.9997905300289346),
 (1791, 0.9997493919925184),
 (1718, 0.9997491878923507)],
2129: [(4356, 0.9995492588966887),
 (3778, 0.9994302479915529),
 (1086, 0.9993856995581781),
 (661, 0.9993851380857133),
 (9121, 0.999301939568832)],
2233: [(4352, 0.9990678525253845),
 (4421, 0.9963850062188582),
 (2638, 0.9960350648501745),
 (7919, 0.9953665059852453),
 (9474, 0.9946900882782871)],
2335: [(6910, 0.9997666241780453),
 (1507, 0.9996980453761146),
 (4732, 0.9996642646068835),
 (2018, 0.9996428247929606),
 (94, 0.9996164386486649)],
2378: [(5275, 0.9982047292331241),
 (3849, 0.9976929379916873),
```

```
 (3849, 0.9970929579910073),
 (8155, 0.9975698300177918),
 (2965, 0.9969097671898652),
 (7741, 0.9967744471662259)],
2449: [(35, 0.9783547362934697),
 (1697, 0.9781424242665138),
 (5810, 0.977828649772407),
 (3134, 0.977728613941448),
 (7287, 0.9774158443447433)],
2461: [(483, 0.9995352396620236),
 (5439, 0.9994609851225934),
 (4767, 0.9994564469344419),
 (1714, 0.9994296410676029),
 (3367, 0.999314388795092)],
2470: [(4373, 0.9988655164322929),
 (3918, 0.9981928751625834),
 (9766, 0.9980766544520752),
 (7016, 0.9976078875980278),
 (3310, 0.997270804725373)],
2575: [(9121, 0.9993598083098291),
 (5810, 0.9993516505135732),
 (5892, 0.9993017055084832),
 (7287, 0.9991779304095035),
 (9295, 0.9991121139878962)],
2638: [(4421, 0.9997959831187581),
 (9474, 0.9993046859007589),
 (7919, 0.9988645058580159),
 (7460, 0.9985462614276491),
 (4447, 0.9964029310329435)],
2645: [(9160, 0.9821181503440944),
 (2755, 0.9754247425332929),
 (3036, 0.9739181420409665),
 (9600, 0.9734734820881643),
 (8244, 0.9733473326452653)],
2755: [(9600, 0.9979104417874659),
 (3036, 0.997013547157039),
 (5317, 0.996933558544043),
 (8244, 0.9959049313285422),
 (1403, 0.9935083281027874)],
2814: [(5545, 0.9813190510321034),
 (7566, 0.9713014828601382),
 (2946, 0.9660450186890007),
 (6685, 0.9592287652676196),
 (9160, 0.9571544426353147)],
2818: [(8386, 0.9883110816390335),
 (1103, 0.9855387835322502),
 (44, 0.9832848564504031),
 (6578, 0.9818363600826),
 (187, 0.9764816455747428)],
2945: [(7989, 0.9988390259262655),
 (3918, 0.9981899830699945),
 (5395, 0.9980806517007996),
 (7016, 0.9970912620929885),
 (3310, 0.9965432299567419)],
2946: [(7566, 0.9934207895033633),
 (5545, 0.981683521311576),
 (2814, 0.9660450186890006),
 (1403, 0.9574938454460232),
 (6685, 0.953412631923677)],
2965: [(2378, 0.9969097671898652),
 (1042, 0.9966615250714466),
 (3849, 0.996544270889211),
 (7989, 0.9953837451854144),
 (8086, 0.9949348993357204)],
2980: [(9631, 0.9995897158599661),
 (3527, 0.9992912564034339),
 (1714, 0.9991185206301563),
 (5193, 0.9989953273131152),
 (1589, 0.9989472052806)],
3036: [(9600, 0.9996494762511329),
 (8244, 0.9995960845844737),
 (5317, 0.9989603529000504),
 (2755, 0.9970135471570389),
 (1403, 0.9884932213812098)],
3039: [(7919, 0.9975753401999666),
 (7460, 0.9975537740157381),
 (9474, 0.9971669167050534),
 (4421, 0.9964753805394063),
```

```
 (4421, 0.9964753605594065),
 (4447, 0.9963469294495091)],
3134: [(9639, 0.9954238198482723),
 (9295, 0.9950270003325591),
 (1790, 0.9949302738987648),
 (9121, 0.9949104685823904),
 (739, 0.9948383233620074)],
3310: [(7016, 0.9988200181279984),
 (1415, 0.9984958878780767),
 (2470, 0.9972708047253729),
 (4373, 0.9972214433753411),
 (9766, 0.996949306085375)],
3367: [(483, 0.9999052779428209),
 (7017, 0.9998495445821641),
 (8829, 0.9997616873636385),
 (1714, 0.9996926338540277),
 (4998, 0.9996609010530736)],
3527: [(1589, 0.9998479847805064),
 (7017, 0.9997506374581657),
 (1714, 0.99961455120713),
 (3367, 0.9995842110262699),
 (7674, 0.9994537948564227)],
3544: [(5785, 0.9651336471490208),
 (1790, 0.9648861916755058),
 (1086, 0.9647108783197827),
 (5972, 0.964647285503315),
 (5814, 0.9643792923129475)],
3577: [(8890, 0.9999213845104723),
 (4031, 0.9998582424154846),
 (4732, 0.9998262678942407),
 (252, 0.9997427809469283),
 (2034, 0.9997156592921829)],
3635: [(4356, 0.9991673332712104),
 (5814, 0.9991535892407155),
 (3778, 0.9991492666255136),
 (8890, 0.9991234501820381),
 (5972, 0.9991231901505142)],
3723: [(7794, 0.9998803898008465),
 (5636, 0.9995140480986897),
 (114, 0.9994033752447673),
 (1790, 0.999397392734743),
 (9052, 0.9993875576010164)],
3778: [(661, 0.999722518725858),
 (4356, 0.9996192752825108),
 (2129, 0.9994302479915528),
 (5972, 0.9993914358972171),
 (1086, 0.9993415352455847)],
3849: [(1042, 0.9988257945440744),
 (2378, 0.9976929379916873),
 (2965, 0.9965442708892112),
 (8155, 0.9962774858632256),
 (7989, 0.9962288252408174)],
3893: [(4514, 0.9999384859193168),
 (2094, 0.9998707735022405),
 (252, 0.999795463256326),
 (1507, 0.9997316408947636),
 (5814, 0.9997255159732666)],
3918: [(5395, 0.9993445799763959),
 (4373, 0.9983611511571743),
 (2470, 0.9981928751625835),
 (2945, 0.9981899830699945),
 (7016, 0.9972639313985648)],
4029: [(8156, 0.9988122352130498),
 (5193, 0.9984645472500384),
 (2034, 0.9983660788475004),
 (4031, 0.9983615270720674),
 (1801, 0.9983290127172147)],
4031: [(3577, 0.9998582424154847),
 (2034, 0.9998420845213813),
 (8829, 0.999826683320846),
 (9729, 0.9998059191214369),
 (4732, 0.9997954130443323)],
4193: [(5275, 0.9975051023437139),
 (2378, 0.995907883570389),
 (3849, 0.9955978837217574),
 (3310, 0.9952809090453395),
 (1415, 0.9949279971822553)],
4228: [(187, 0.9939913920973358),
```

```
4228: [(187, 0.9939913920973358),
 (9766, 0.9929961385083774),
 (3918, 0.9927640144301555),
 (4373, 0.992665158459879),
 (2470, 0.9926116974255944)],
4296: [(114, 0.9982930602126162),
 (7794, 0.9982240471805633),
 (3723, 0.9982045855725568),
 (5636, 0.9981668262940828),
 (4356, 0.9979918749396941)],
4352: [(2233, 0.9990678525253847),
 (4421, 0.996010342844439),
 (2638, 0.9960033854278041),
 (9474, 0.9949470917316648),
 (7919, 0.9948064953363083)],
4356: [(5972, 0.9998364697564001),
 (1086, 0.9997962949214335),
 (661, 0.9996210671527851),
 (3778, 0.9996192752825109),
 (9121, 0.9995789420114221)],
4373: [(2470, 0.9988655164322928),
 (3918, 0.9983611511571743),
 (5395, 0.9982152904329621),
 (9766, 0.9979944067158238),
 (3310, 0.9972214433753411)],
4421: [(2638, 0.999795983118758),
 (7919, 0.9994768145672481),
 (9474, 0.9994079041671569),
 (7460, 0.9990404713067481),
 (4447, 0.9975011652789916)],
4447: [(7460, 0.9990298824050952),
 (7919, 0.9984471984112977),
 (9474, 0.9977802127310184),
 (4421, 0.9975011652789915),
 (2638, 0.9964029310329435)],
4514: [(3893, 0.9999384859193167),
 (2094, 0.9999110312147726),
 (252, 0.9998149826712783),
 (1507, 0.9997883182907749),
 (4732, 0.999787525900665)],
4671: [(9160, 0.9487422074795259),
 (2645, 0.9479463389143676),
 (1403, 0.9416113144177581),
 (6685, 0.9412570394068254),
 (2755, 0.9352721966001165)],
4732: [(1507, 0.9998323779871466),
 (3577, 0.9998262678942408),
 (252, 0.9998204772868788),
 (4031, 0.9997954130443323),
 (4514, 0.999787525900665)],
4767: [(483, 0.9996519884334284),
 (3367, 0.9995180444121469),
 (4031, 0.9995053287663618),
 (6412, 0.9994770250060035),
 (2034, 0.9994741827689858)],
4874: [(6101, 0.9081360085302395),
 (9620, 0.796610631960247),
 (2814, 0.6550999792361529),
 (7566, 0.6180523007312853),
 (5545, 0.5779013122733112)],
4998: [(8829, 0.9997433571519947),
 (483, 0.9996831589235712),
 (3367, 0.9996609010530735),
 (1714, 0.9996004472241617),
 (6412, 0.9995915016004973)],
5129: [(1790, 0.9756176542811406),
 (9639, 0.975592873539426),
 (5814, 0.9753427497541857),
 (252, 0.9751794279601734),
 (9295, 0.975126686790169)],
5131: [(8829, 0.9996704567382493),
 (8890, 0.999649934266783),
 (4732, 0.9995990750660798),
 (3577, 0.9995967762727644),
 (7429, 0.9995587273528849)],
5193: [(3527, 0.9991322032709365),
 (1714, 0.9990715347519281),
 (2080, 0.9989952272121151),
```

```
 (2980, 0.9989953273131151),
 (3367, 0.9989837070046986),
 (7017, 0.998943020080547)],
5275: [(2378, 0.998204729233124),
 (4193, 0.9975051023437139),
 (8155, 0.9973624180851547),
 (7741, 0.9955634853671348),
 (3849, 0.9955253434072195)],
5317: [(9600, 0.9992813592667408),
 (3036, 0.9989603529000504),
 (8244, 0.9983718060559149),
 (2755, 0.9969335585440429),
 (9160, 0.9889737675020481)],
5395: [(3918, 0.9993445799763958),
 (4373, 0.9982152904329621),
 (2945, 0.9980806517007995),
 (2470, 0.9969807868692491),
 (9766, 0.9967309644195788)],
5403: [(5810, 0.9985693935459531),
 (5484, 0.9980941978037048),
 (1718, 0.9980556207951278),
 (35, 0.9980402200233943),
 (1697, 0.9980358705791674)],
5439: [(483, 0.9996322143485051),
 (6412, 0.9995724945592533),
 (3367, 0.9994993989404942),
 (2461, 0.9994609851225935),
 (4998, 0.9994328097949656)],
5484: [(484, 0.99971789908391),
 (5810, 0.9993616267584141),
 (1697, 0.9992647589428167),
 (35, 0.9991133136434336),
 (2575, 0.9990224540417103)],
5545: [(2946, 0.981683521311576),
 (2814, 0.9813190510321033),
 (7566, 0.9808197051521071),
 (1403, 0.9645809546105708),
 (9160, 0.9574563160202276)],
5636: [(7794, 0.9996143470426252),
 (9052, 0.999579658156437),
 (3723, 0.9995140480986898),
 (4031, 0.9995078110225205),
 (114, 0.999482424425691)],
5658: [(9956, 0.9915266512615706),
 (7965, 0.9872300564485511),
 (6673, 0.9796972154389366),
 (8059, 0.9769653455132536),
 (1103, 0.9711546668090155)],
5785: [(1086, 0.999086759846652),
 (1790, 0.9988034439949693),
 (9639, 0.9987991202533018),
 (661, 0.9987692914002947),
 (5972, 0.9987287438368446)],
5810: [(7287, 0.9996437846449472),
 (35, 0.9995398065544198),
 (5484, 0.999361626758414),
 (2575, 0.9993516505135731),
 (9121, 0.9993465798828168)],
5814: [(739, 0.999820199299256),
 (252, 0.9997293991892912),
 (3893, 0.9997255159732668),
 (4514, 0.9997241275254156),
 (2094, 0.9997202107650878)],
5892: [(9121, 0.9997696465977658),
 (9295, 0.9996368966277581),
 (2094, 0.9996003814637967),
 (739, 0.9995227935266908),
 (4514, 0.999469808604751)],
5972: [(4356, 0.9998364697564),
 (1086, 0.9997522832975294),
 (5814, 0.9996601270614613),
 (739, 0.9995833327343763),
 (9121, 0.9995822716968046)],
6101: [(4874, 0.9081360085302395),
 (9620, 0.8524908426378867),
 (2814, 0.6430684624351916),
 (7566, 0.5868160445383147),
 (5545, 0.5467995190143287)]
```

```
 (5545, 0.5467905199143871)],
6412: [(4031, 0.9997931344911986),
 (8829, 0.9997568062097308),
 (2034, 0.9997081416679298),
 (3577, 0.9996623348146201),
 (483, 0.9996064630096795)],
6505: [(4421, 0.993291393872761),
 (7919, 0.9931827401649254),
 (2233, 0.9924214262828203),
 (3039, 0.9923438371651361),
 (2638, 0.9921709281783669)],
6578: [(44, 0.9928492095318779),
 (8386, 0.9862626136584406),
 (1103, 0.9853566591149603),
 (2818, 0.9818363600826),
 (187, 0.9691628222166575)],
6673: [(8059, 0.9942996646120098),
 (5658, 0.9796972154389367),
 (7965, 0.9791476326396373),
 (1103, 0.9691174679586494),
 (9956, 0.9637723154984187)],
6685: [(1403, 0.9905459232478045),
 (2755, 0.987259765484761),
 (9160, 0.9867384622218613),
 (9600, 0.982950882241556),
 (5317, 0.9828459370789667)],
6830: [(2461, 0.999150519800095),
 (9631, 0.9991340620853628),
 (1714, 0.9990291595151534),
 (2980, 0.9987930855367271),
 (5439, 0.9986368027504037)],
6836: [(2018, 0.9560795839055515),
 (6910, 0.955586170579213),
 (8156, 0.9554864877594126),
 (1801, 0.9553208817283806),
 (7017, 0.9553040773342791)],
6863: [(8703, 0.9857419163983987),
 (5658, 0.9620943598492351),
 (8059, 0.9544835271529275),
 (6673, 0.9456873264951631),
 (9956, 0.934664019165783)],
6910: [(1507, 0.9998738439627559),
 (1801, 0.9997789070940649),
 (4732, 0.9997736825484306),
 (2335, 0.9997666241780453),
 (2018, 0.9997340839397812)],
7016: [(3310, 0.9988200181279984),
 (2470, 0.9976078875980277),
 (1415, 0.9975654409401623),
 (9766, 0.9974171264389377),
 (3918, 0.9972639313985648)],
7017: [(3367, 0.9998495445821641),
 (3527, 0.9997506374581657),
 (483, 0.9996457423331337),
 (8829, 0.9996408212066529),
 (2018, 0.9995478930048519)],
7030: [(8156, 0.9975756520860104),
 (2018, 0.9972153746512002),
 (1718, 0.9970323206505373),
 (1801, 0.99697310795677),
 (6910, 0.9969175080743012)],
7117: [(1589, 0.9980972888191499),
 (3527, 0.9978997978921168),
 (370, 0.9974336231934242),
 (744, 0.9973945149425663),
 (7017, 0.9973305799810995)],
7287: [(9295, 0.9996641697777989),
 (5810, 0.9996437846449472),
 (9121, 0.9996380282109898),
 (7682, 0.9995123409957433),
 (5892, 0.9994194646111346)],
7429: [(8829, 0.9998470149339266),
 (2034, 0.999756939274056),
 (4732, 0.9996738045840574),
 (4031, 0.9996607327990629),
 (9729, 0.9996454146093662)],
7460: [(9474, 0.9995954692801334),
```

```
 (7919, 0.9994409796033555),
 (4421, 0.9990404713067481),
 (4447, 0.9990298824050952),
 (2638, 0.9985462614276491)],
7566: [(2946, 0.9934207895033634),
 (5545, 0.980819705152107),
 (2814, 0.9713014828601381),
 (1403, 0.9417655231678487),
 (6685, 0.9392017716918766)],
7674: [(1801, 0.9997795377227342),
 (2018, 0.999766950685546),
 (1718, 0.9997038005786582),
 (9729, 0.9995933753559256),
 (8829, 0.9995141391228429)],
7682: [(7287, 0.9995123409957432),
 (9121, 0.9991413973924684),
 (5810, 0.999092022598209),
 (35, 0.9989979036772753),
 (9295, 0.99896677219805)],
7739: [(9631, 0.9972893046551947),
 (2980, 0.9962224290998304),
 (6830, 0.9958562310995267),
 (1589, 0.9955675229678951),
 (1714, 0.9952549126194583)],
7741: [(8086, 0.9976540242972207),
 (2072, 0.997470575294492),
 (661, 0.9974576267086213),
 (3778, 0.9970180095462596),
 (2378, 0.9967744471662258)],
7794: [(3723, 0.9998803898008465),
 (9052, 0.9996593774896806),
 (5636, 0.9996143470426252),
 (1619, 0.9995508541534226),
 (114, 0.9994674220815236)],
7900: [(8829, 0.9996631232313484),
 (9729, 0.9995559690491651),
 (4998, 0.9995377537965454),
 (5131, 0.9994789563284574),
 (7429, 0.9994613070108067)],
7919: [(4421, 0.9994768145672481),
 (7460, 0.9994409796033555),
 (9474, 0.9993893369794952),
 (2638, 0.998864505858016),
 (4447, 0.9984471984112978)],
7965: [(1103, 0.9904408409713493),
 (5658, 0.987230056448551),
 (9956, 0.9862869956960529),
 (6673, 0.9791476326396374),
 (44, 0.9745610836315821)],
7989: [(2945, 0.9988390259262654),
 (1042, 0.9979130307466569),
 (3918, 0.9963396891736758),
 (3849, 0.9962288252408172),
 (5395, 0.9960566157903086)],
8059: [(6673, 0.9942996646120098),
 (5658, 0.9769653455132536),
 (7965, 0.9705223397122131),
 (8703, 0.9666303057262045),
 (9956, 0.9550869059519329)],
8084: [(1801, 0.9992791648472383),
 (1718, 0.9992523415948974),
 (7674, 0.9992437767816449),
 (1791, 0.9992226457176149),
 (9729, 0.9990987199170149)],
8086: [(2072, 0.9991319649112246),
 (661, 0.9988365087939052),
 (114, 0.9987883536325121),
 (1086, 0.9987544603394238),
 (3778, 0.9987494605818741)],
8155: [(5785, 0.9975965944509673),
 (2378, 0.9975698300177918),
 (9278, 0.9975121540282167),
 (5275, 0.9973624180851548),
 (1619, 0.9967139681812895)],
8156: [(1801, 0.9995969840155055),
 (6910, 0.9995743021156099),
 (1718, 0.9995710006096283),
```

```
 (2018, 0.9995635343276223),
 (1507, 0.9995570763528097)],
8244: [(3036, 0.9995960845844736),
 (9600, 0.9992748696102196),
 (5317, 0.9983718060559149),
 (2755, 0.9959049313285421),
 (9160, 0.986285470306179)],
8386: [(187, 0.9915229891078267),
 (2818, 0.9883110816390335),
 (6578, 0.9862626136584406),
 (44, 0.9859651669698513),
 (4228, 0.9797815367694066)],
8467: [(9766, 0.9970465209623862),
 (3310, 0.9964028185637984),
 (1415, 0.9959934701971438),
 (7016, 0.9953657105442822),
 (4373, 0.9950597969514487)],
8703: [(6863, 0.9857419163983986),
 (8059, 0.9666303057262045),
 (5658, 0.9624832116187774),
 (6673, 0.9540073681710297),
 (7965, 0.9402994116622838)],
8829: [(2034, 0.9998517886920096),
 (7429, 0.9998470149339267),
 (4031, 0.999826683320846),
 (9729, 0.9998181022649485),
 (3367, 0.9997616873636386)],
8890: [(3577, 0.9999213845104723),
 (4732, 0.9997558630554749),
 (4031, 0.9997388840181279),
 (252, 0.9996790512022913),
 (5814, 0.9996563879466827)],
8967: [(77, 0.9288916128273637),
 (1283, 0.9225359033382659),
 (9849, 0.9141373356081968),
 (1697, 0.90853965392523),
 (484, 0.9066039994983202)],
9052: [(7794, 0.9996593774896805),
 (5636, 0.999579658156437),
 (1619, 0.9994413229686984),
 (3723, 0.9993875576010162),
 (8890, 0.9993446903382494)],
9121: [(9295, 0.9998418049649345),
 (5892, 0.9997696465977658),
 (4514, 0.9997156596488337),
 (2094, 0.9996804312407231),
 (3893, 0.9996598809358784)],
9134: [(252, 0.9937785641566648),
 (6910, 0.9937645651182937),
 (4732, 0.9937234512478815),
 (2335, 0.9936661140009556),
 (1507, 0.9936547374169856)],
9160: [(2755, 0.9927061812028467),
 (1403, 0.9899674234959066),
 (9600, 0.9898749279056445),
 (5317, 0.988973767502048),
 (3036, 0.9883874733262722)],
9278: [(5785, 0.9978200878874097),
 (8155, 0.9975121540282168),
 (661, 0.997295511553789),
 (1086, 0.9967971499731972),
 (7741, 0.996755067210582)],
9295: [(9121, 0.9998418049649345),
 (7287, 0.9996641697777799),
 (739, 0.9996387404372391),
 (5892, 0.9996368966277579),
 (2094, 0.999531594199345)],
9474: [(7460, 0.9995954692801333),
 (4421, 0.999407904167157),
 (7919, 0.9993893369794952),
 (2638, 0.999304685900759),
 (4447, 0.9977802127310184)],
9600: [(3036, 0.9996494762511329),
 (5317, 0.9992813592667408),
 (8244, 0.9992748696102195),
 (2755, 0.9979104417874659),
 (9160, 0.9898749279056446)],
```

```
 9620: [(6101, 0.8524908426378868),
  (4874, 0.796610631960247),
  (2814, 0.7957371233695427),
  (7566, 0.7848478804160264),
  (2946, 0.7588176342479984)],
 9631: [(2980, 0.9995897158599661),
  (1714, 0.9993865690693795),
  (6830, 0.9991340620853628),
  (3527, 0.9990829030785681),
  (1589, 0.9990090181740074)],
 9639: [(1086, 0.9995201316109045),
  (5972, 0.9995028879167716),
  (4356, 0.999474257742344),
  (9295, 0.9993770884739483),
  (739, 0.9993504698685001)],
 9729: [(8829, 0.9998181022649485),
  (4031, 0.9998059191214368),
  (2034, 0.999789125474335),
  (4732, 0.9997545574629064),
  (3577, 0.9997132988646733)],
 9766: [(2470, 0.9980766544520752),
  (4373, 0.9979944067158238),
  (7016, 0.9974171264389378),
  (3918, 0.9970664951591942),
  (8467, 0.9970465209623862)],
 9849: [(1697, 0.9986833905451354),
  (77, 0.9985861469650852),
  (35, 0.9985502073910255),
  (1283, 0.9972656502841635),
  (5484, 0.9972425589029247)],
 9956: [(5658, 0.9915266512615705),
  (7965, 0.9862869956960529),
  (1103, 0.9810659926085473),
  (6673, 0.9637723154984186),
  (44, 0.957112928041076)],
 9982: [(35, 0.9936215743451244),
  (5810, 0.9934047932228133),
  (5484, 0.993342491972477),
  (1697, 0.9931450651413527),
  (2575, 0.9930388562922453)]}
```

Q1.4 Student Proposal

Focus: Find out which meter needs maintenance

- Draw histogram to see the number of faulty meters with various magnitude of decreasing value. (Prof. Mehul's recommendation to visualize)

On top of magnitude, no. of times the value decrease can be second histogram to observe.

- Set our own standard from the visualization and calculate number of meters in need of maintenance

Follow-up idea:

1) Print a summary to inform the company & consumer about the meter's status. "We recommend maintenance because __."

2) Expected amount of payment for the maintenance. "Expected price: $__. Please dial 1800-900-XXXX"

Q2.1 In this part, you will asked to build a model to forecast the hourly readings in the future (next hour). Can you explain why you may want to forecast the gas consumption in the future?

Predicting gas consumption in the future is necessary because of many reasons:

1. Gas is a natural resource and it is limited, predicting gas consumption will shed some light onto how much more we need to produce and when it is necessary to cut down on the consumption for fear of exhausting this natural resource.
2. It allows the gas companies to purchase in bulk in the anticipation of the consumption. This could potentially reduce purchase price, cost of transferring gas, and inventory fee.
3. Since forecast is based on the current consumption pattern, if the real data deviates from forecast by certain percentage,

then it would be easier to investigate what has caused the deviation from forecast. i.e. Environment campaign, unexpected cold wind drift causing people to use more heater, etc.

Who would find this information valuable?

Just to list down a few: customers, government, environmental activists, gas company, maintenance company, gas production countries.

What can you do if you have a good forecasting model?

With a good forecasting model, there could be various actions that can be done: Sell the data / models to

1. Gas companies
2. Maintenance companies
3. Government or environmental activitists

This might be also able to predict

1. When should next servicing of the system be done before it becomes faulty.
2. Show trends for amount of gas consumed in a region or consumption in coming months or even years. Then, gas companies can prepare in advance for bulk orders for cheaper cost of production. Or could reduce inventory issues by preparing only the suitable amount of gas.
3. Possible effect of consumption of natural gases on global warming in the future.

Q2.2 Build linear regression to forecast the hourly readings in the future.

(Regarding Q2.2 and Q2.3 future reading estimation beyond the 6 months data, they are shown at the end of Q2.3)

In [16]:

```
hourArray = features_train = np.asarray(range(4392)).reshape(-1,1)

clfDictLR = {}
for key in hourlyDataDict:
    #each key is a linear regression model,
    clfDictLR[key] =  LR()

    #features_train = the number of hour from 5:00:00 01/10/2015,
    features_train = hourArray

    #print(features_train),
    #labels_train = values,
    labels_train = np.asarray(hourlyDataDict[key])

    #fit the model accordingly,
    clfDictLR[key].fit(features_train, labels_train)

    hourArray = features_train = np.asarray(range(4392)).reshape(-1,1)
```

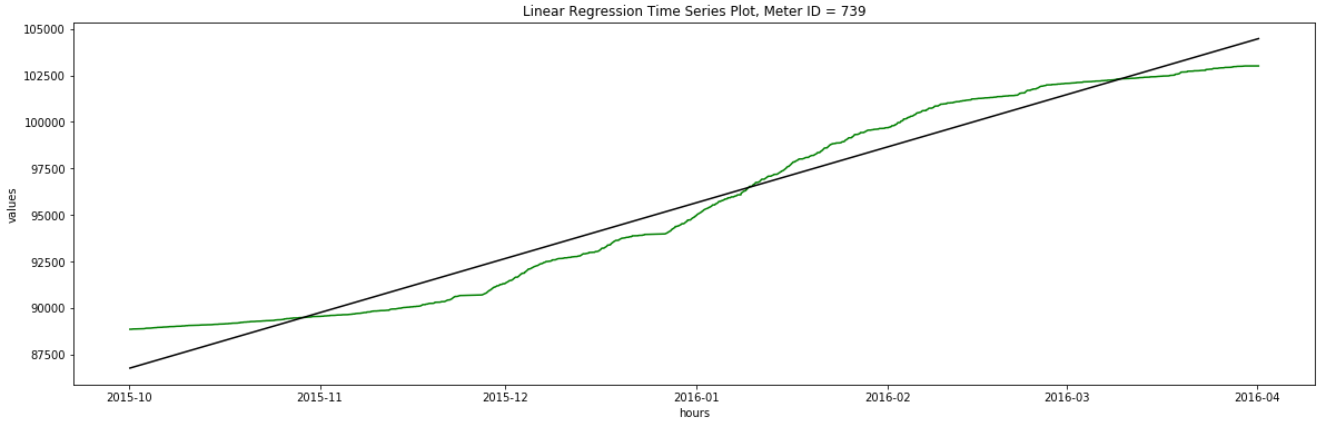
In [17]:

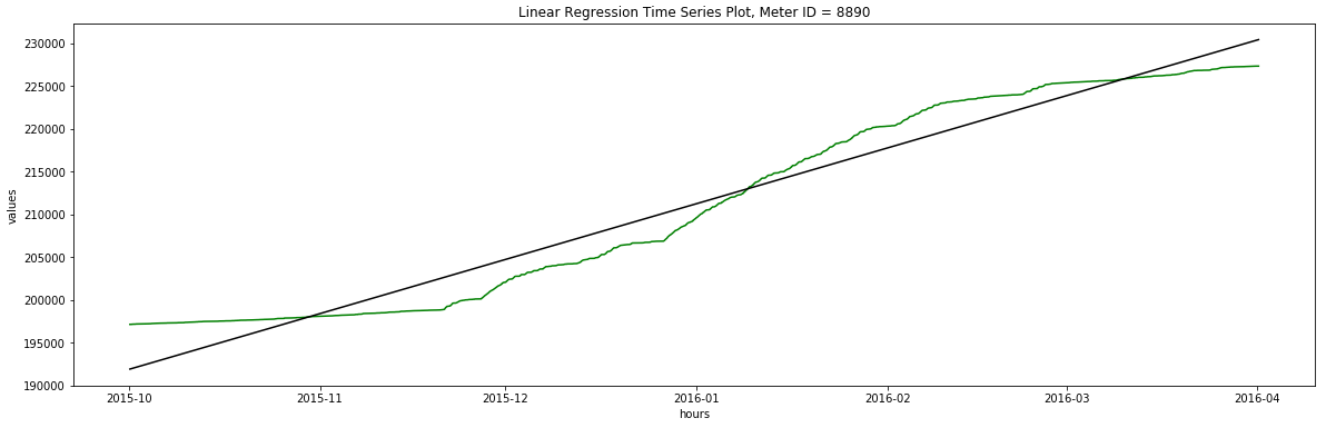
```
staHr = pd.Timestamp('2015-10-01 05:00:00')

dtStar = datetime.datetime(2015, 10, 1) + datetime.timedelta(hours=5)
dtEnd = datetime.datetime(2016, 4, 1) + datetime.timedelta(hours=4)


#get a list of dateTime from the start to the end\n

dateTimeList = pd.date_range(dtStar, dtEnd, freq = 'H').tolist()
hourList = [[((x - staHr).seconds + (x - staHr).days*86400)/ 3600] for x in dateTimeList]

def newPredLR(meter_id):
    return clfDictLR[meter_id].predict(hourList)
```

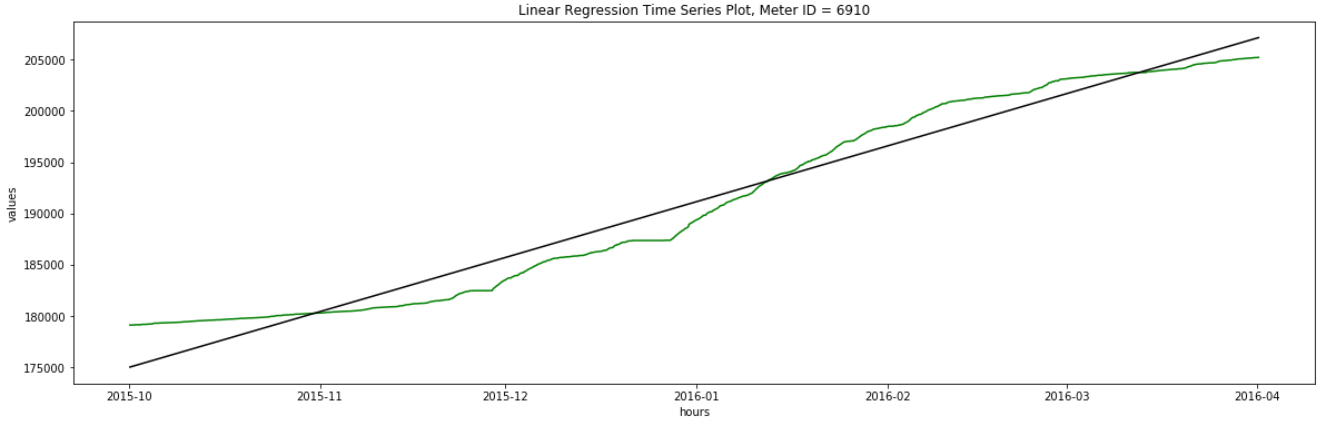
i) Time Series Plot

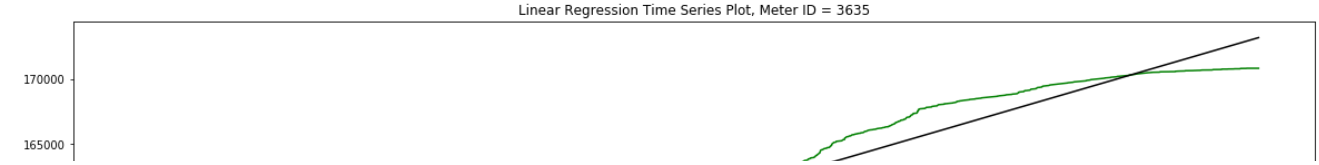
In [18]:

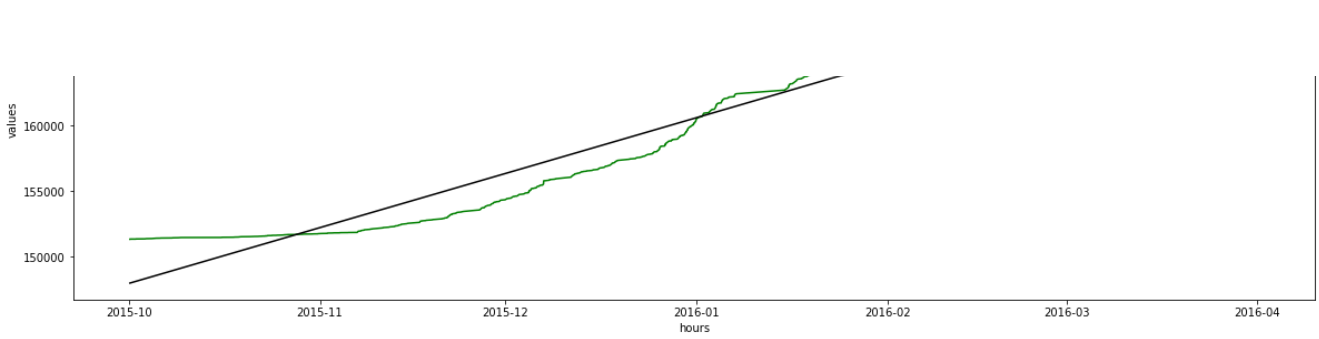
```
for key in hourlyDataDict:
    fig, ax = plt.subplots(1, 1, figsize=(20, 6))

    plt.xlabel('hours')
    plt.ylabel('values')
    x = dateTimeList
    y = np.asarray(hourlyDataDict[key])

    ax.plot(x, y,color='g')
    ax.plot(x,  newPredLR(key) ,color='k')
    plt.title("Linear Regression Time Series Plot, Meter ID = " + str(key))
    plt.show()
```

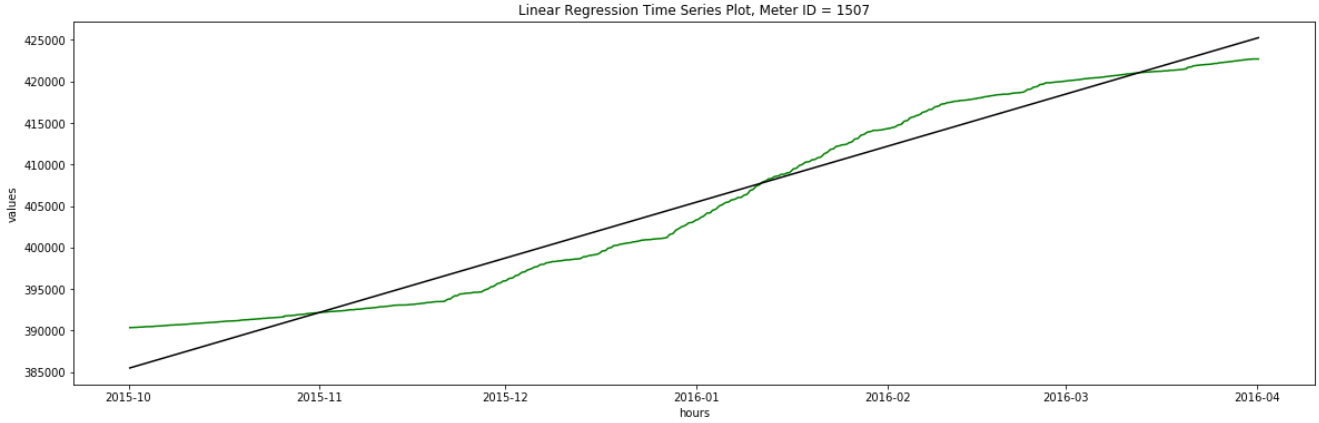


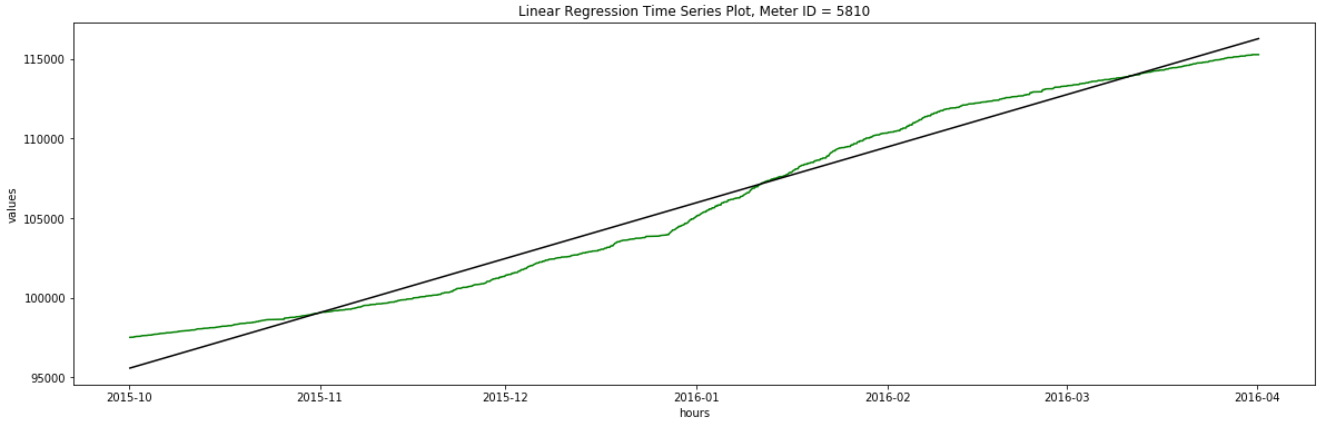


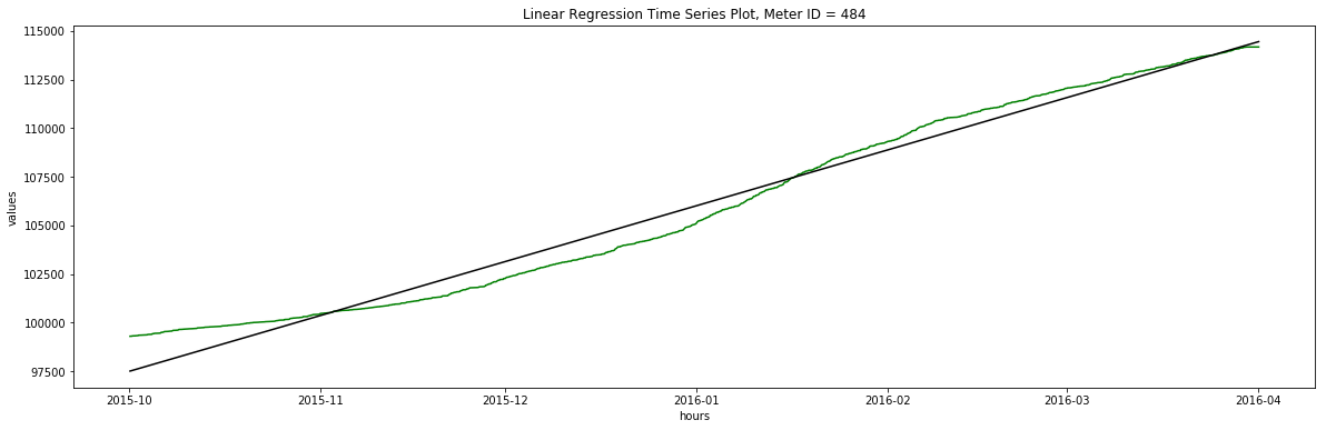


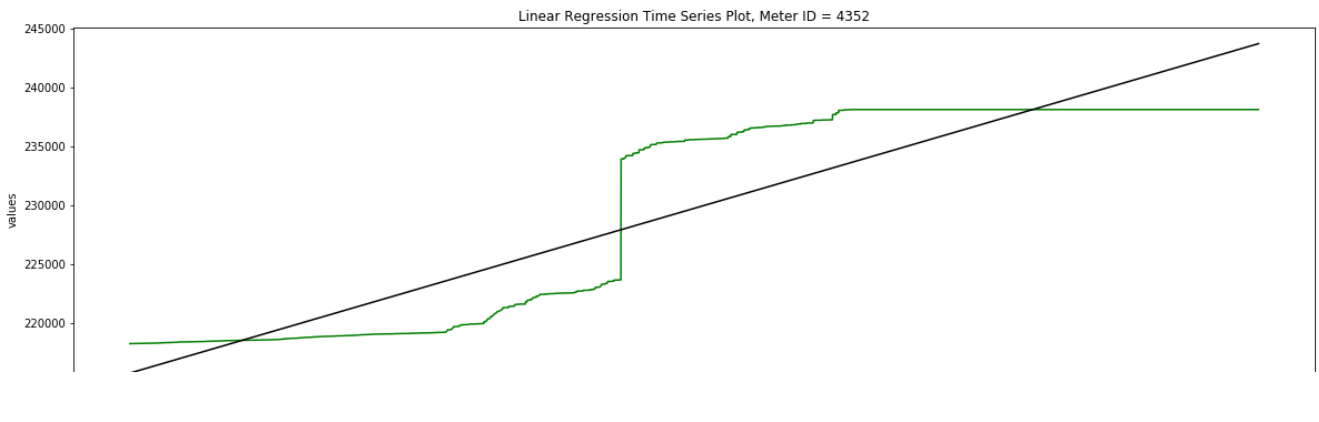


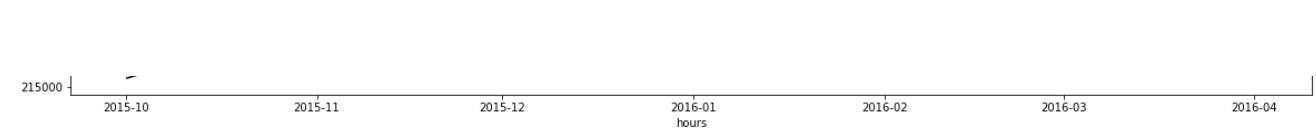

values
160000
155000
150000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

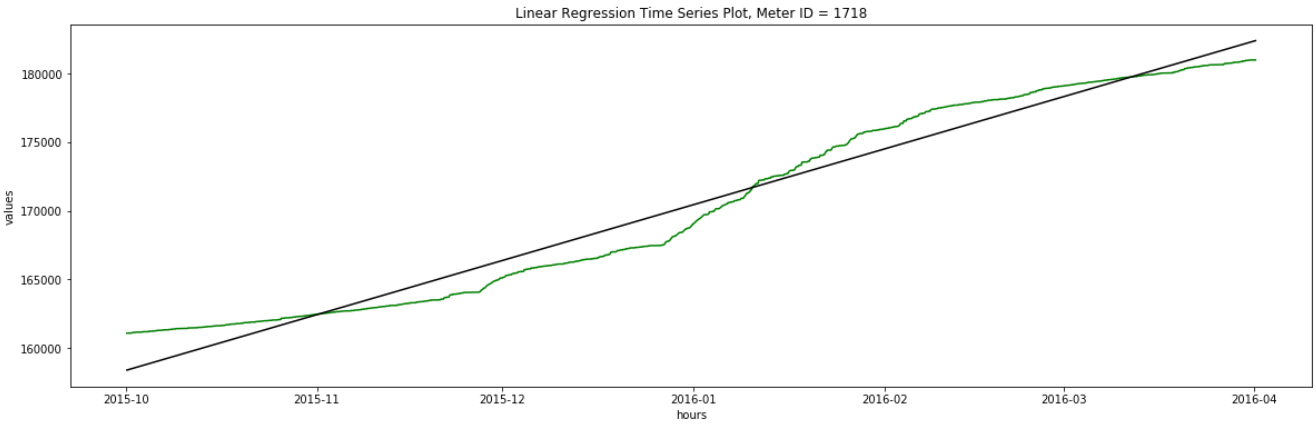

Linear Regression Time Series Plot, Meter ID = 1507
values
425000
420000
415000
410000
405000
400000
395000
390000
385000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

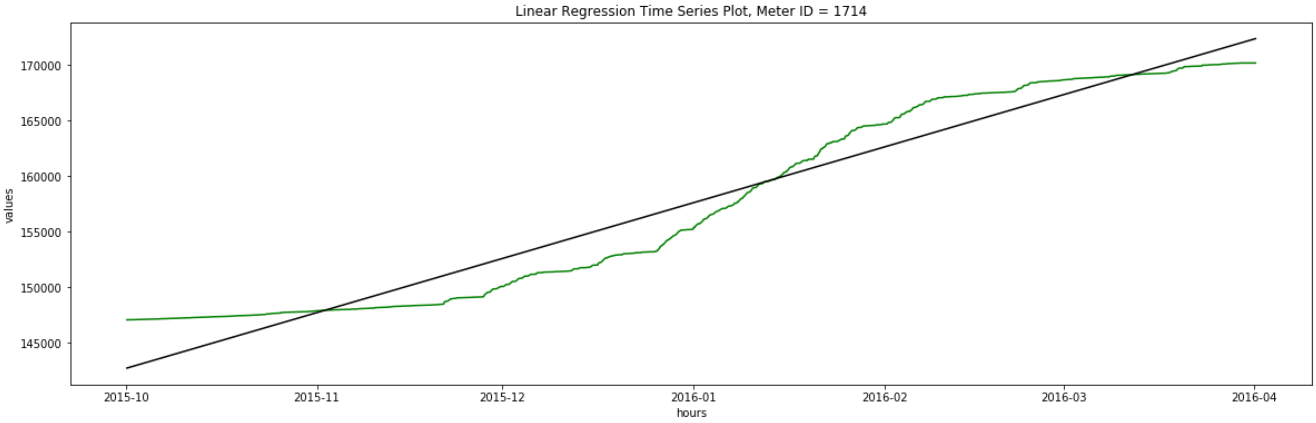

Linear Regression Time Series Plot, Meter ID = 5810
values
115000
110000
105000
100000
95000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

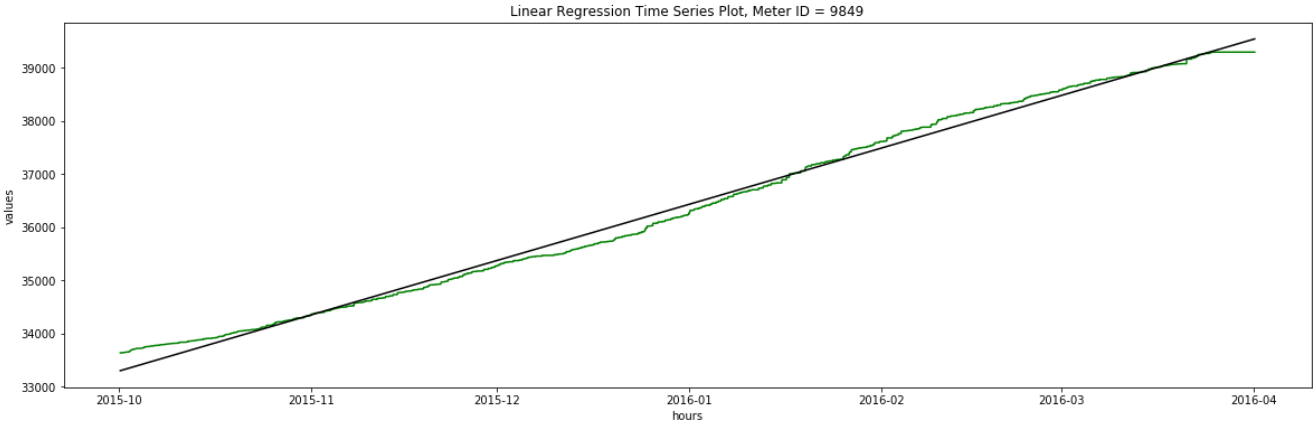

Linear Regression Time Series Plot, Meter ID = 484
values
115000
112500
110000
107500
105000
102500
100000
97500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

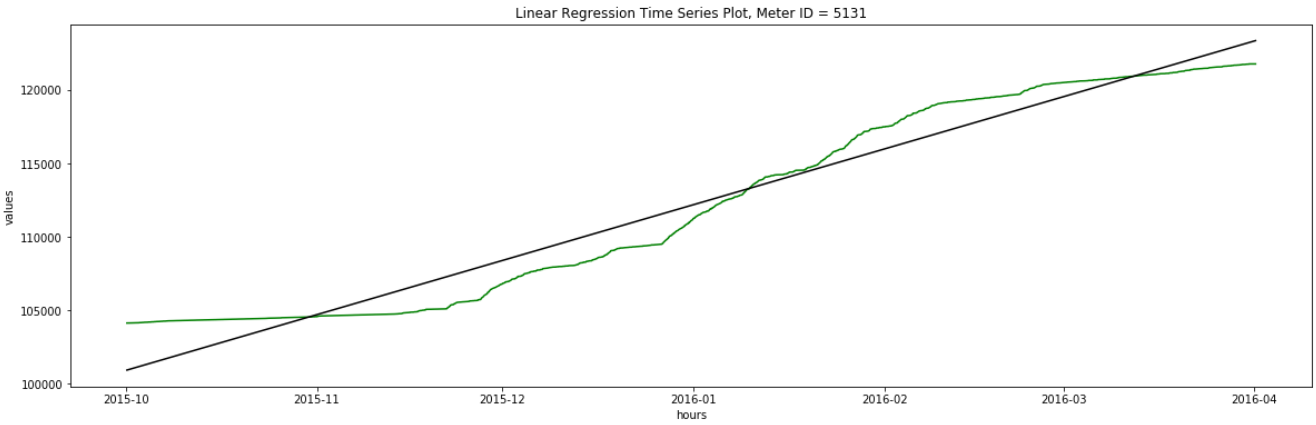

Linear Regression Time Series Plot, Meter ID = 4352
values
245000
240000
235000
230000
225000
220000

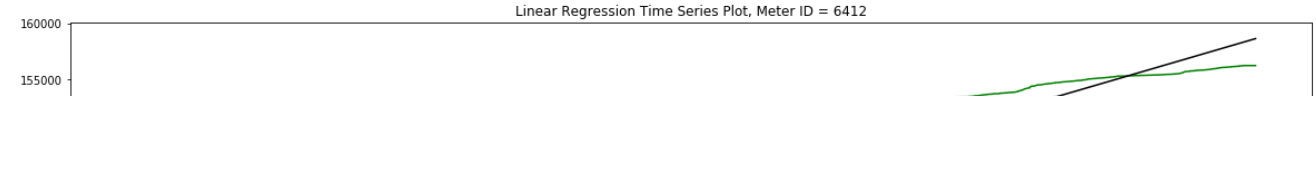

215000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

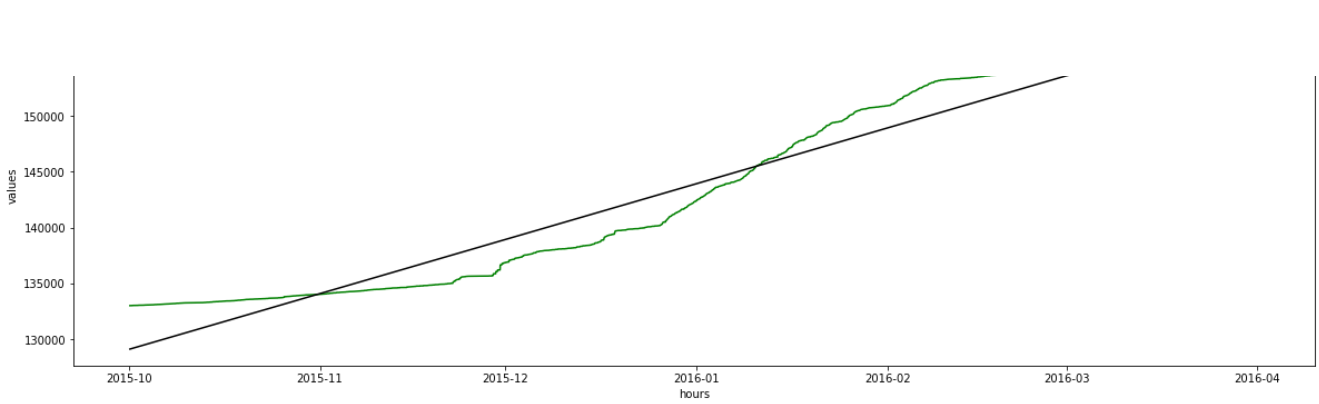

Linear Regression Time Series Plot, Meter ID = 1718
values
180000
175000
170000
165000
160000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

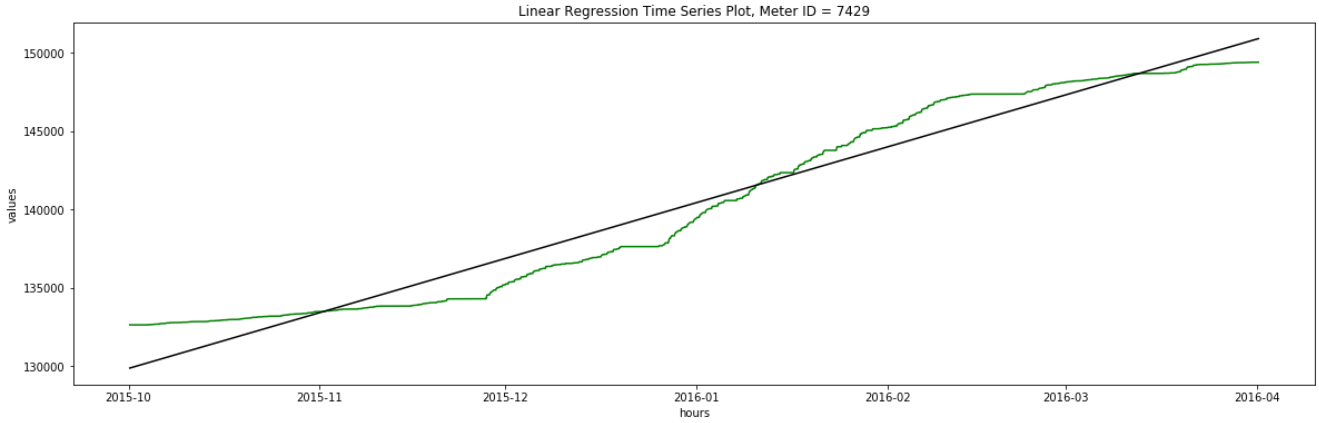

Linear Regression Time Series Plot, Meter ID = 1714
values
170000
165000
160000
155000
150000
145000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

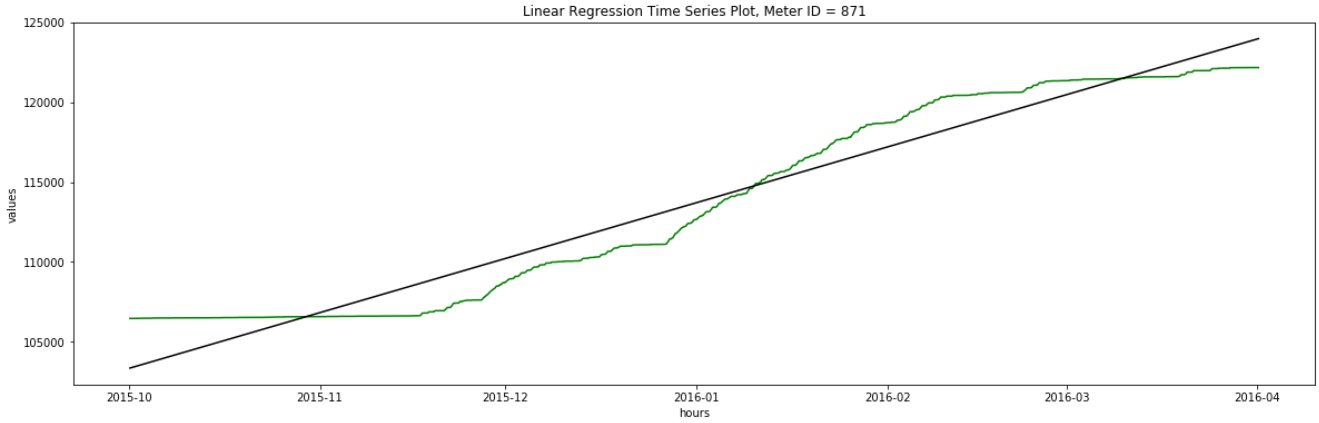

Linear Regression Time Series Plot, Meter ID = 9849
values
39000
38000
37000
36000
35000
34000
33000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

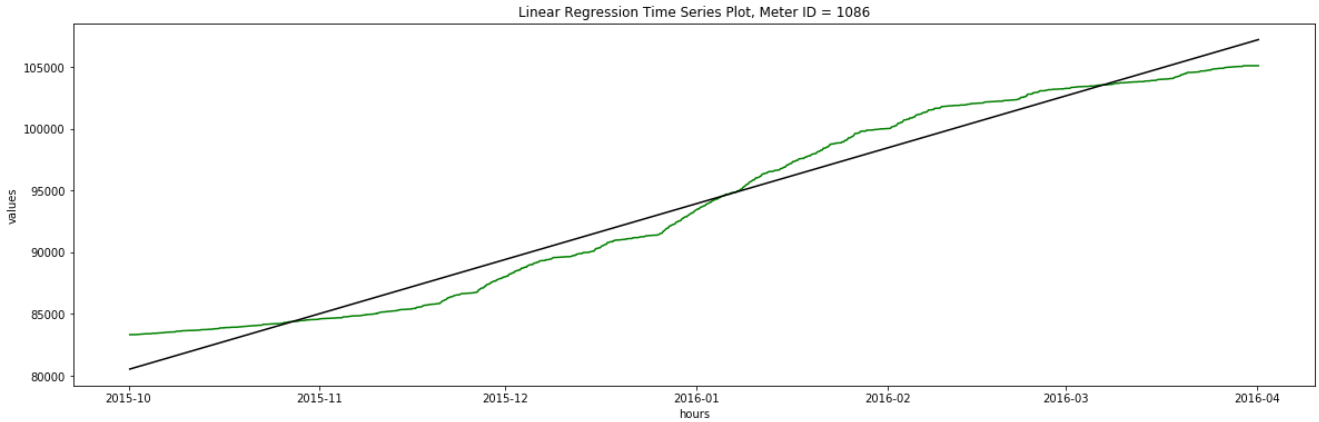

Linear Regression Time Series Plot, Meter ID = 5131
values
120000
115000
110000
105000
100000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

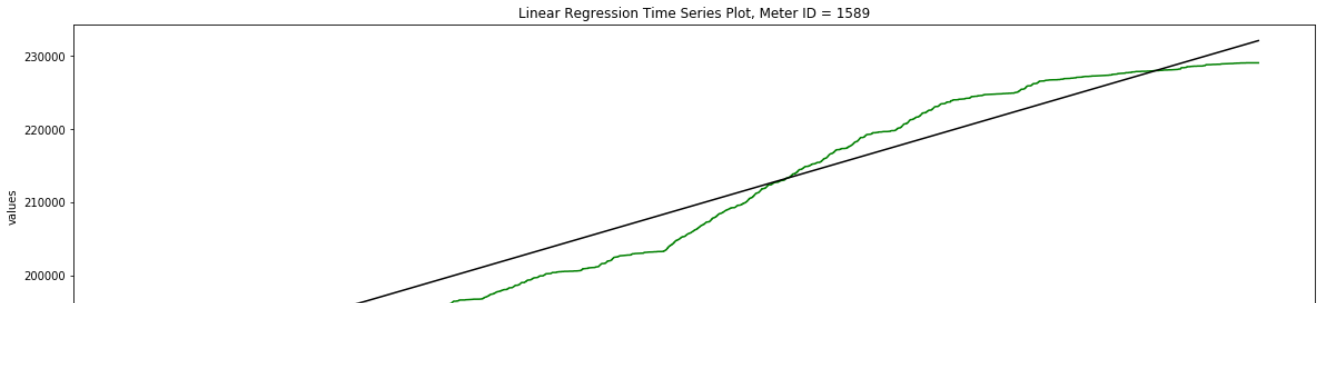

Linear Regression Time Series Plot, Meter ID = 6412
160000
155000


150000
145000
140000
135000
130000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

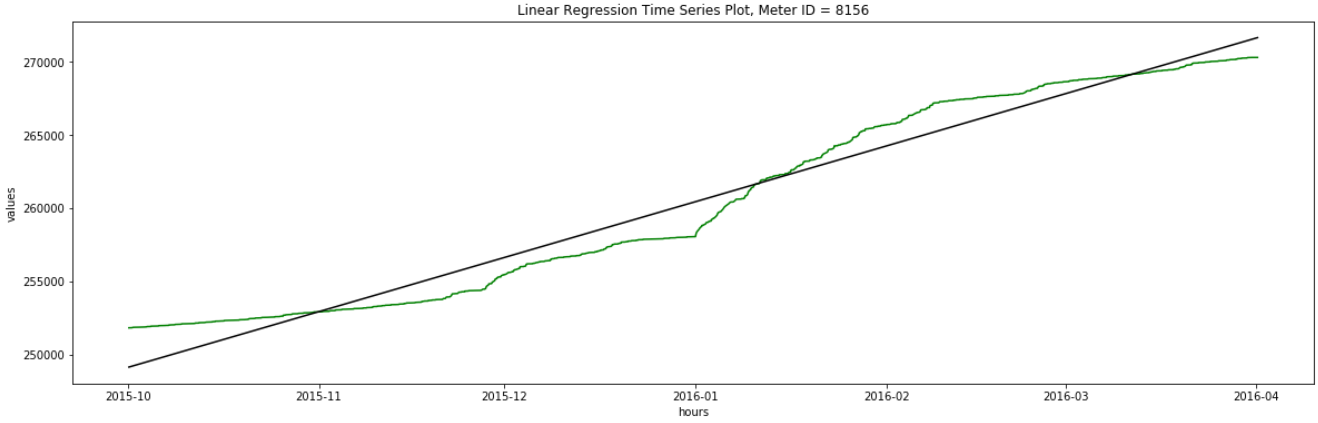

Linear Regression Time Series Plot, Meter ID = 7429
150000
145000
140000
135000
130000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

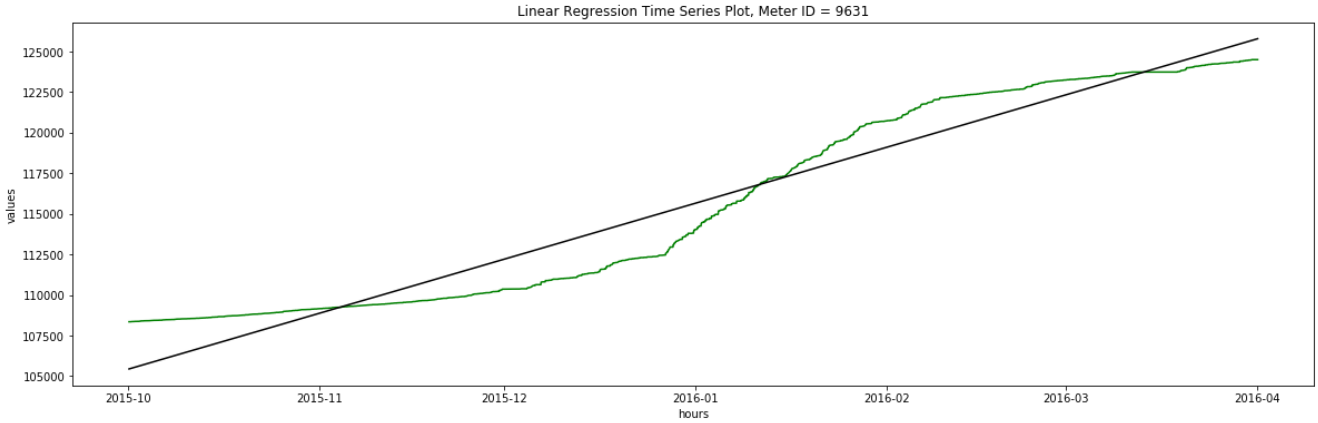

Linear Regression Time Series Plot, Meter ID = 871
125000
120000
115000
110000
105000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

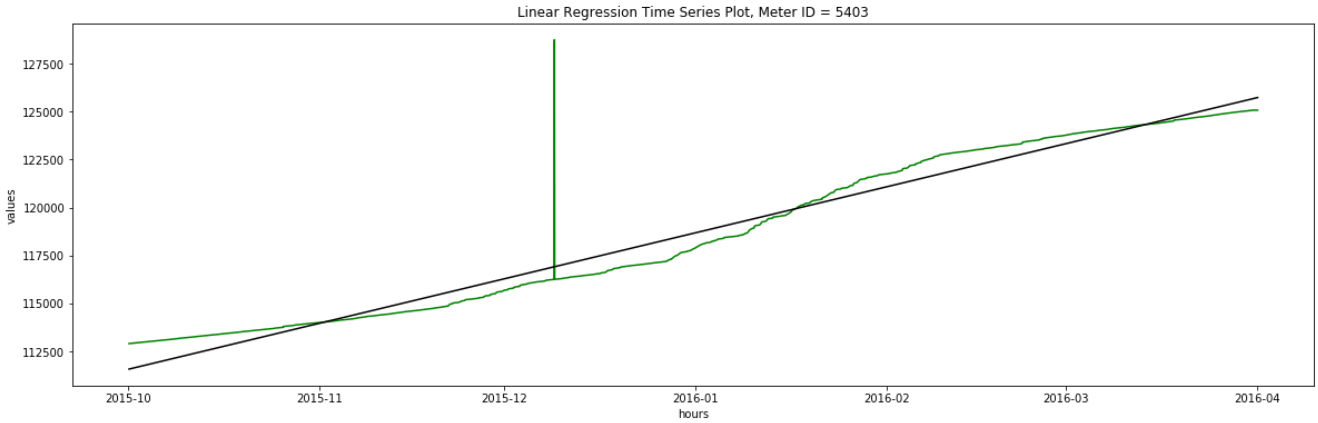

Linear Regression Time Series Plot, Meter ID = 1086
105000
100000
95000
90000
85000
80000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

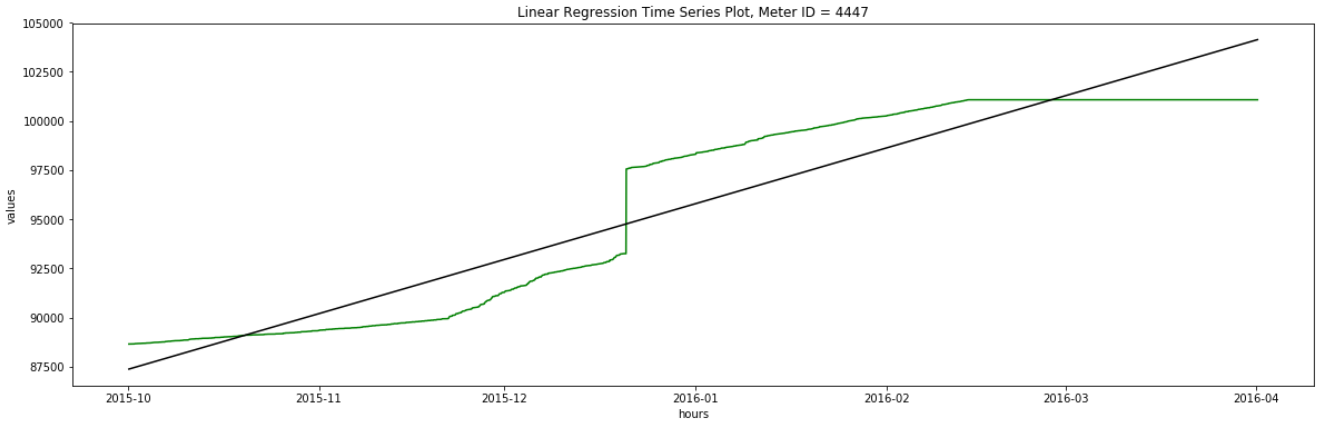

Linear Regression Time Series Plot, Meter ID = 1589
230000
220000
210000
200000
values


190000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

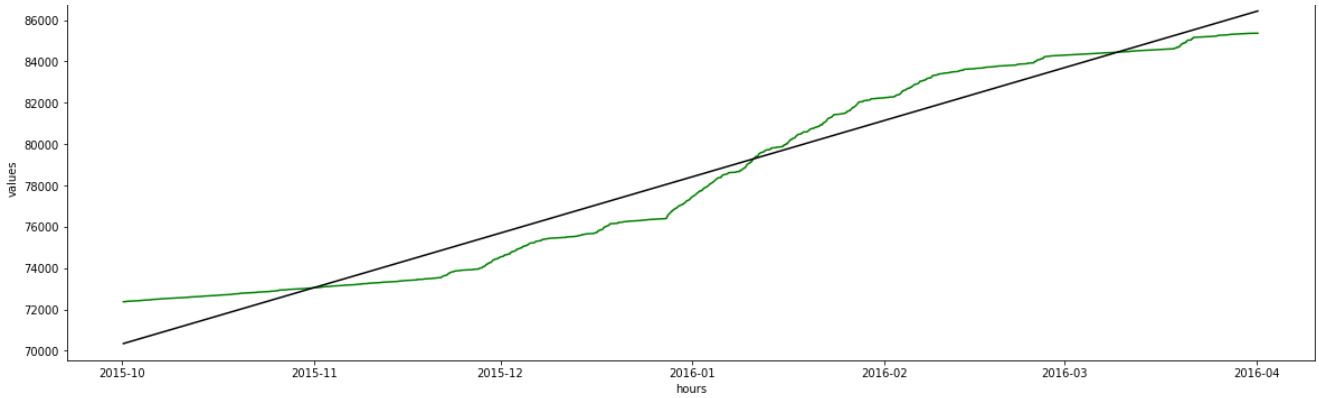

Linear Regression Time Series Plot, Meter ID = 8156
values
270000
265000
260000
255000
250000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

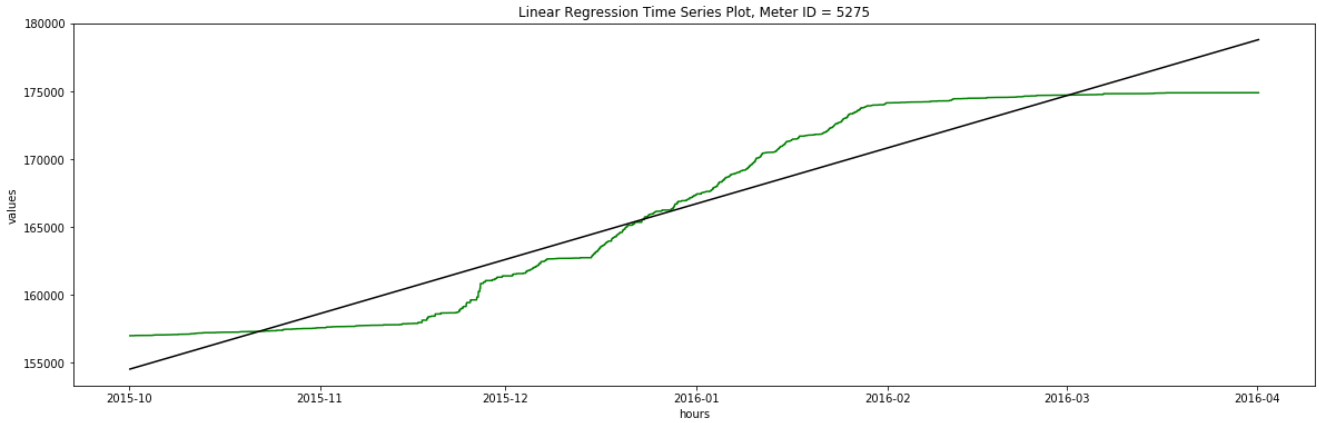

Linear Regression Time Series Plot, Meter ID = 9631
values
125000
122500
120000
117500
115000
112500
110000
107500
105000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

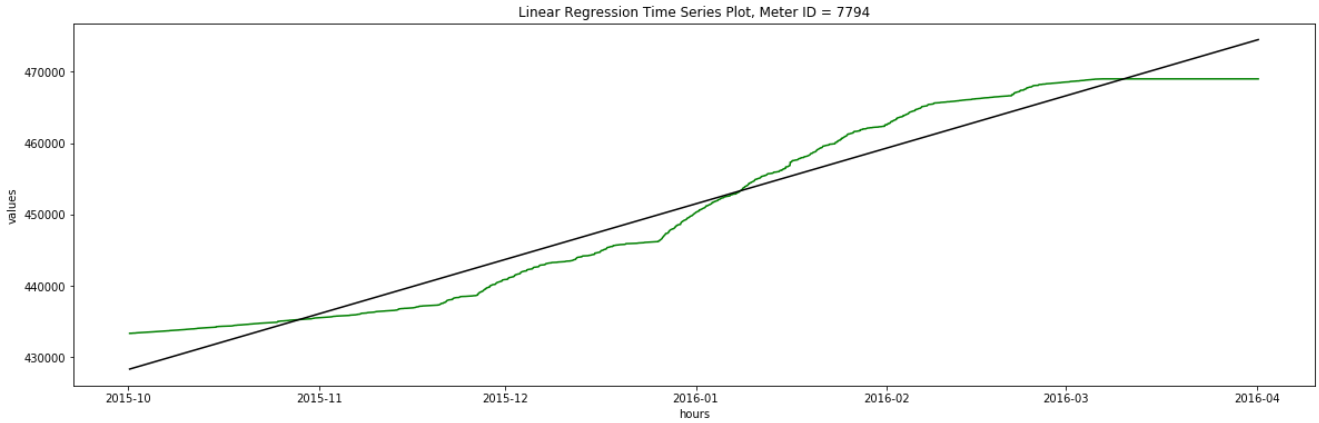

Linear Regression Time Series Plot, Meter ID = 5403
values
127500
125000
122500
120000
117500
115000
112500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

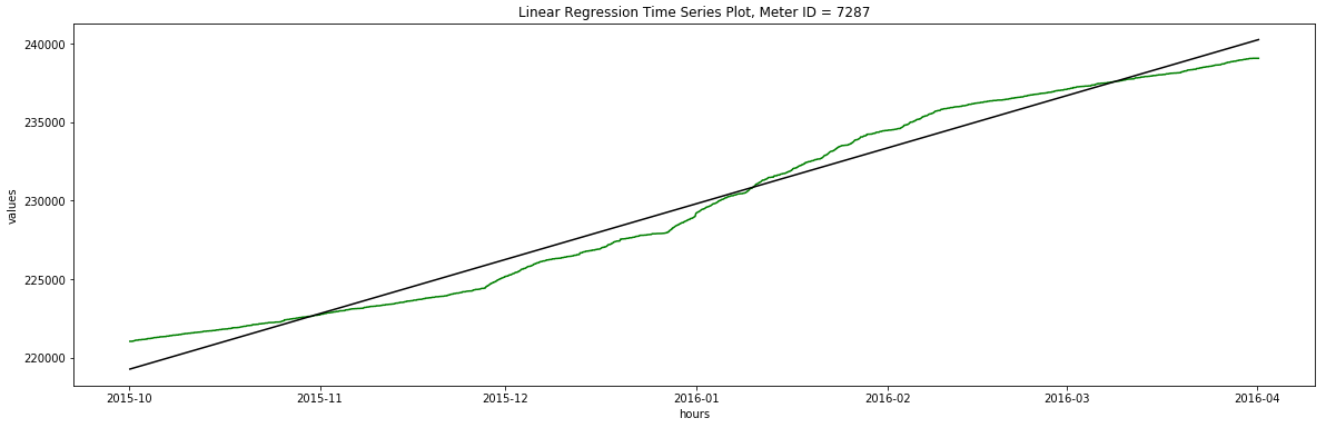

Linear Regression Time Series Plot, Meter ID = 4447
values
105000
102500
100000
97500
95000
92500
90000
87500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours


Linear Regression Time Series Plot, Meter ID = 2034

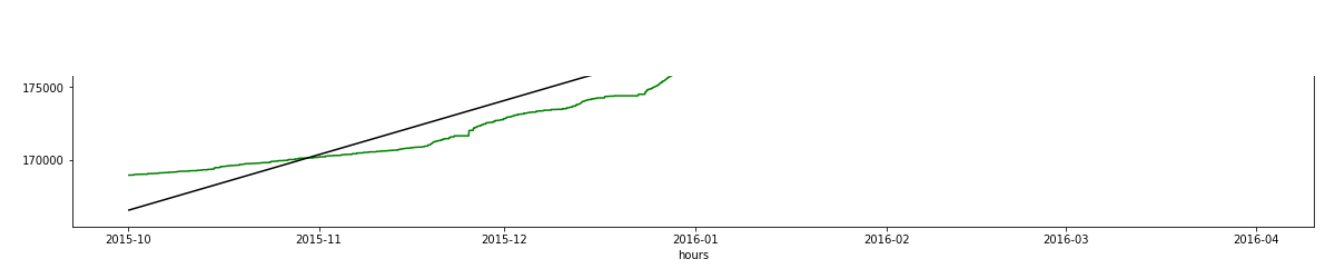

86000
84000
82000
80000
78000
76000
74000
72000
70000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

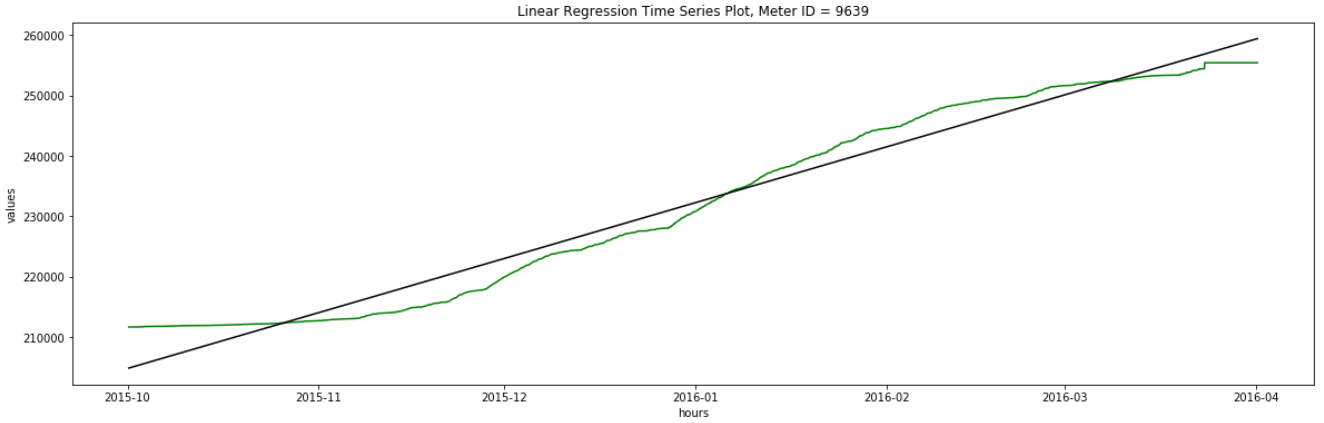

Linear Regression Time Series Plot, Meter ID = 5275
180000
175000
170000
165000
160000
155000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

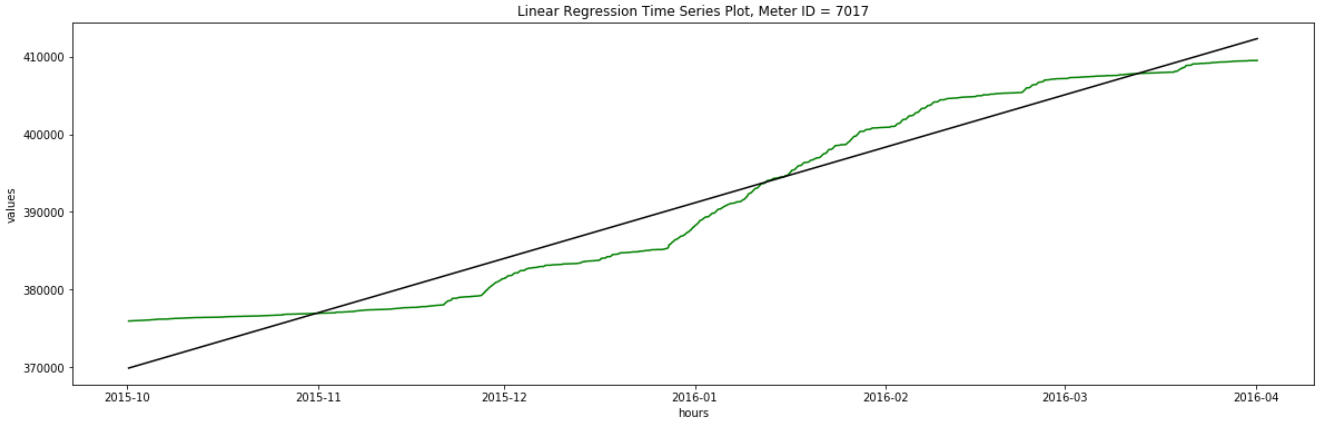

Linear Regression Time Series Plot, Meter ID = 7794
470000
460000
450000
440000
430000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

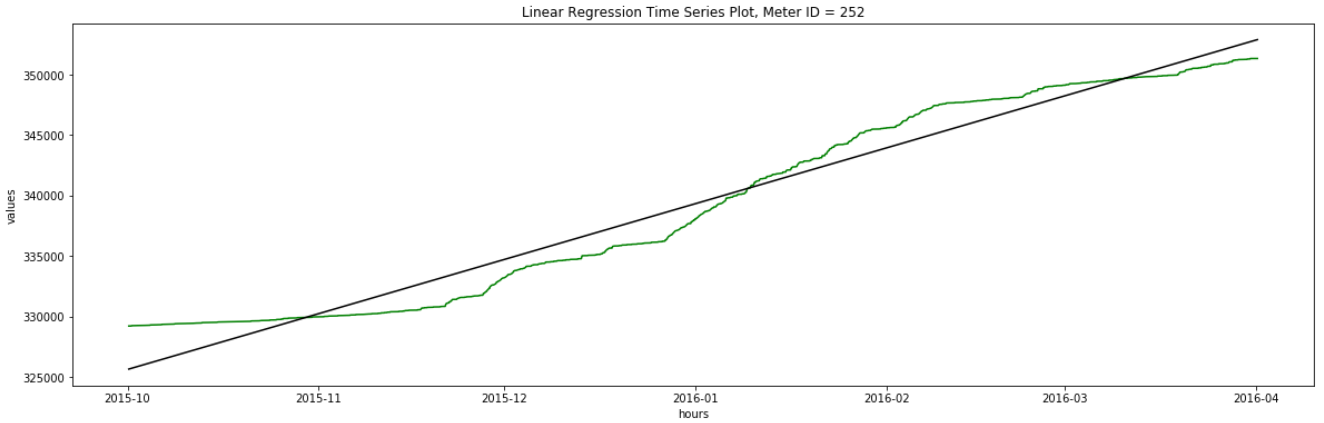

Linear Regression Time Series Plot, Meter ID = 7287
240000
235000
230000
225000
220000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

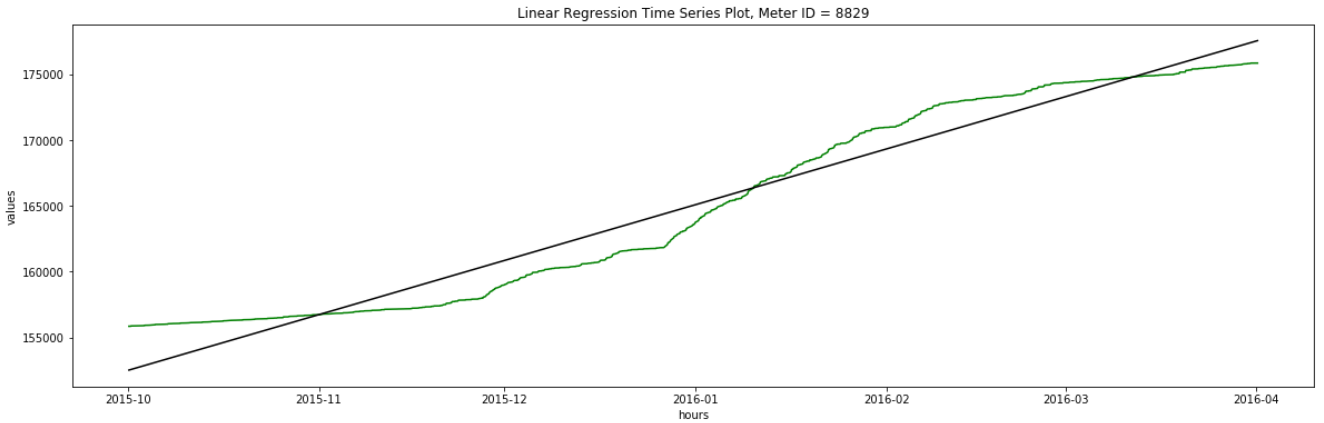

Linear Regression Time Series Plot, Meter ID = 4296
190000
185000
180000
values

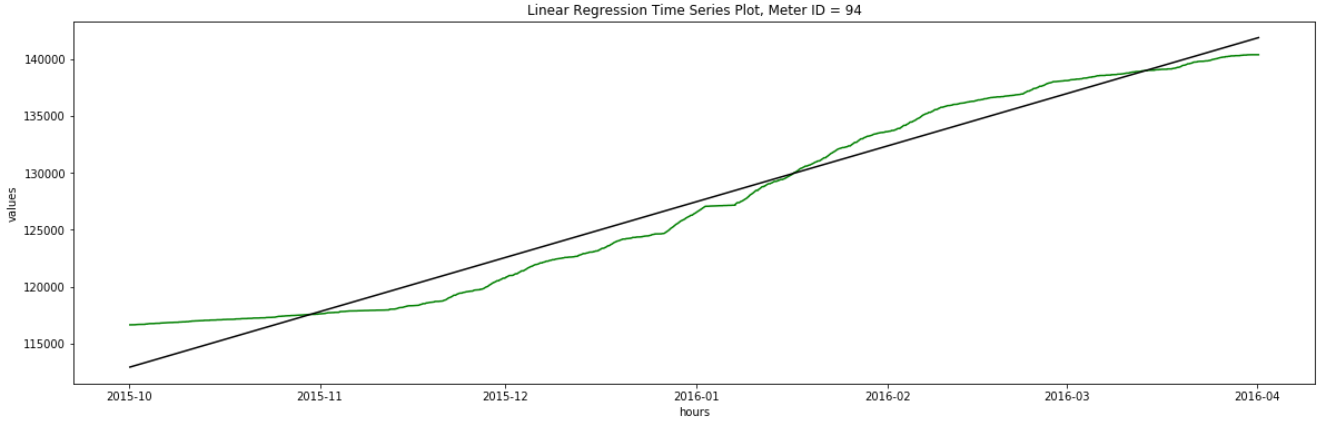

175000
170000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

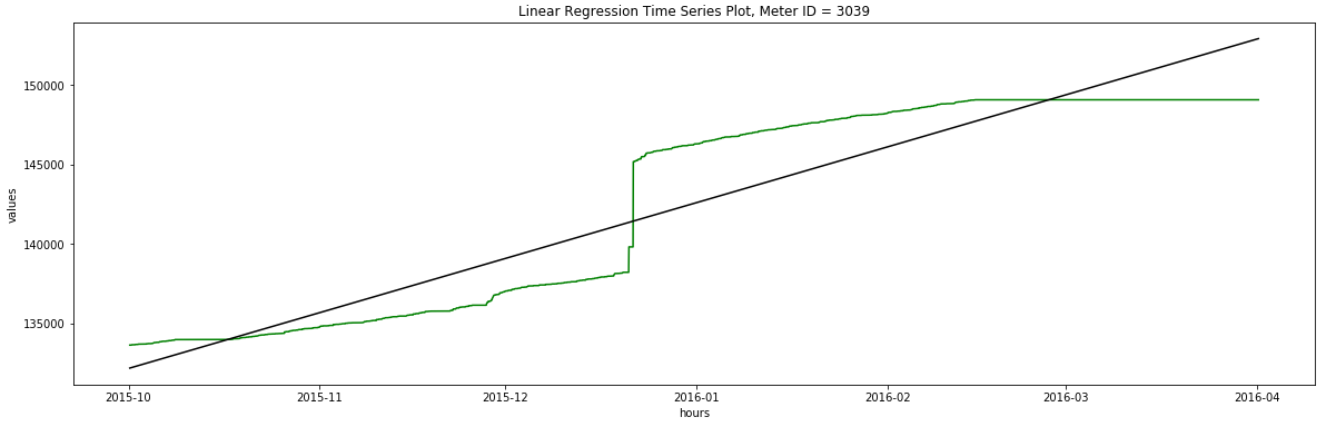

Linear Regression Time Series Plot, Meter ID = 9639
260000
250000
240000
230000
220000
210000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

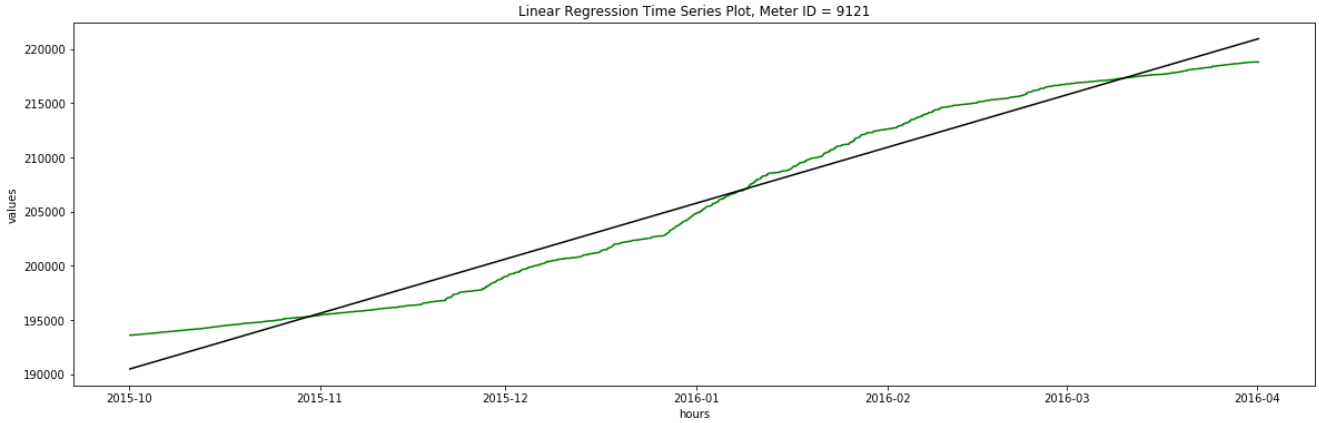

Linear Regression Time Series Plot, Meter ID = 7017
410000
400000
390000
380000
370000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

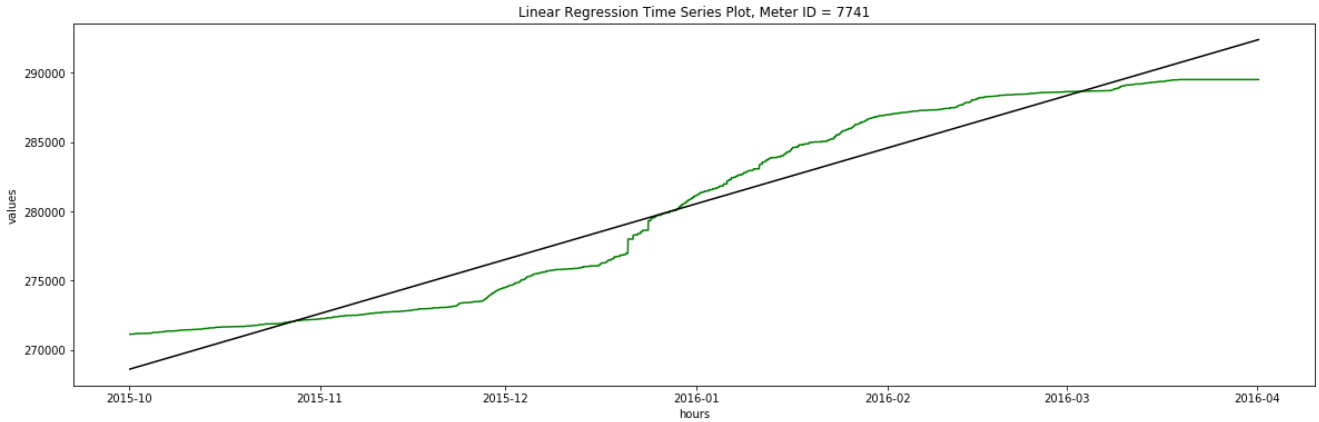

Linear Regression Time Series Plot, Meter ID = 252
350000
345000
340000
335000
330000
325000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

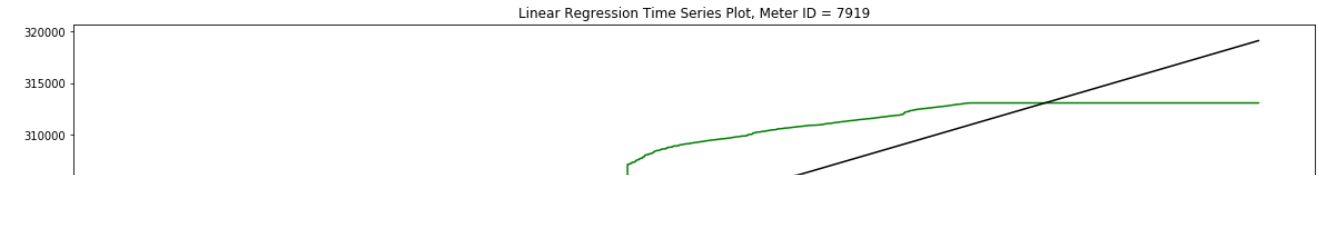

Linear Regression Time Series Plot, Meter ID = 8829
175000
170000
165000
160000
155000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

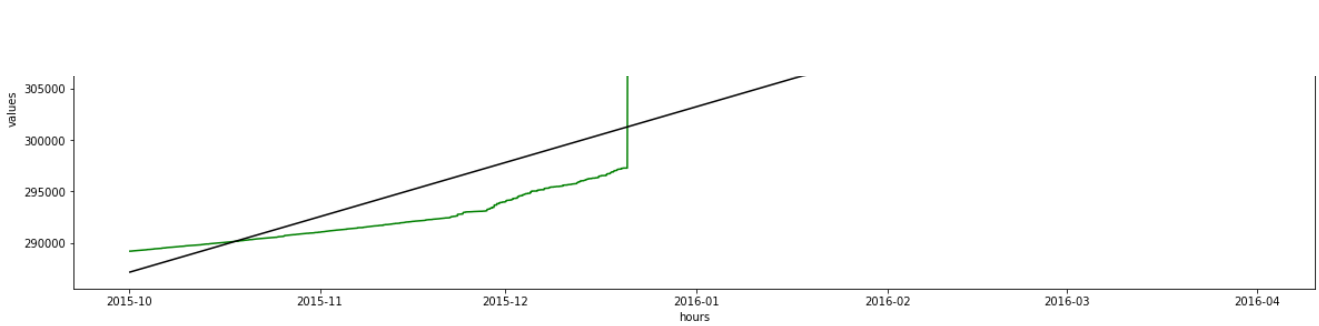

Linear Regression Time Series Plot, Meter ID = 94
values
hours
140000
135000
130000
125000
120000
115000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

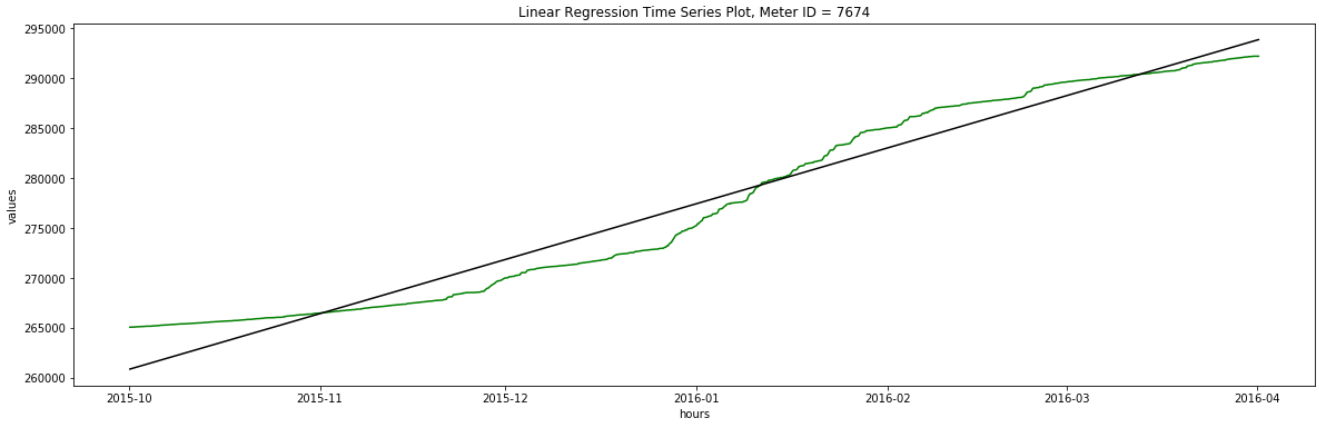

Linear Regression Time Series Plot, Meter ID = 3039
values
hours
150000
145000
140000
135000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

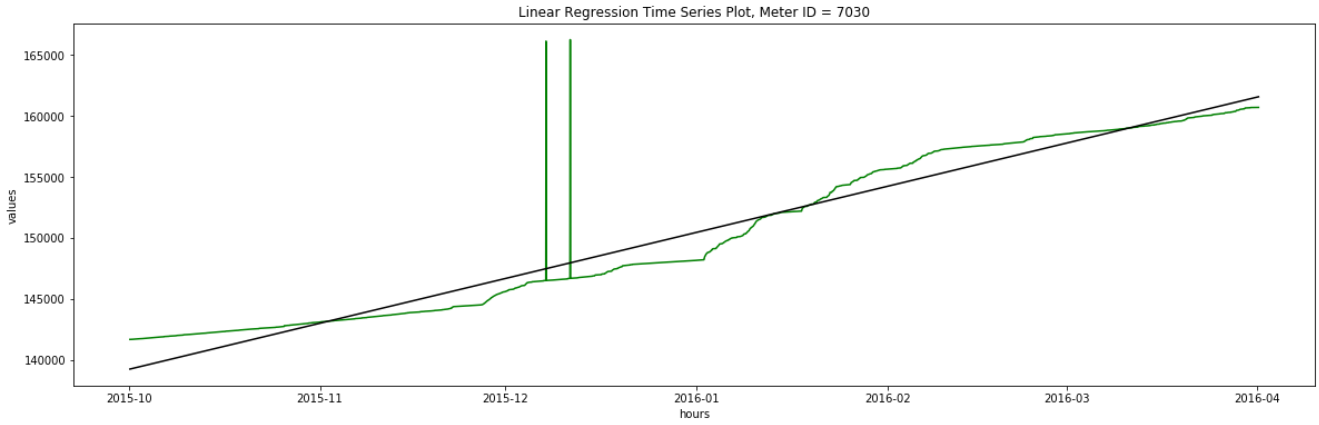

Linear Regression Time Series Plot, Meter ID = 9121
values
hours
220000
215000
210000
205000
200000
195000
190000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

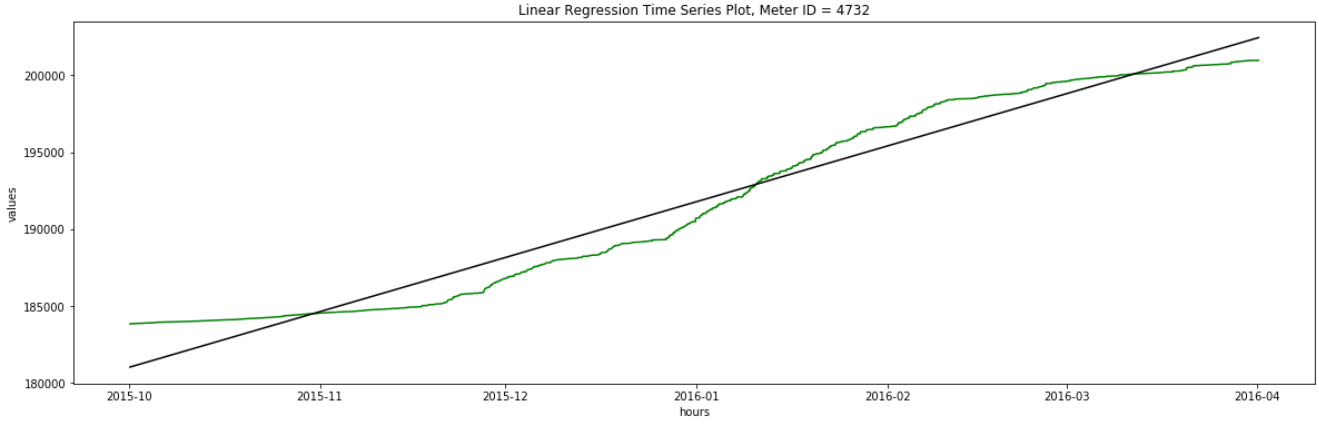

Linear Regression Time Series Plot, Meter ID = 7741
values
hours
290000
285000
280000
275000
270000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

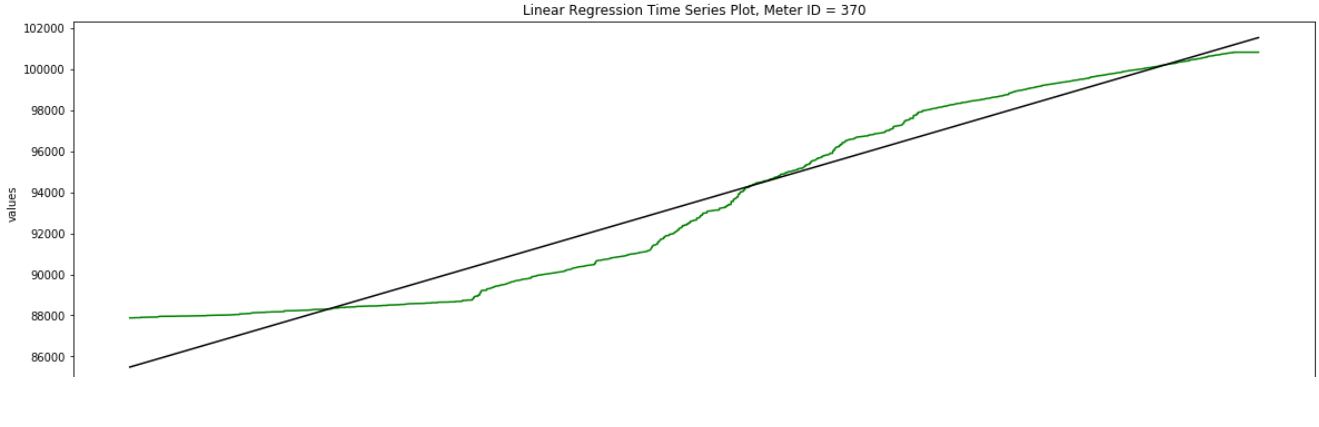

Linear Regression Time Series Plot, Meter ID = 7919
320000
315000
310000


values
305000
300000
295000
290000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

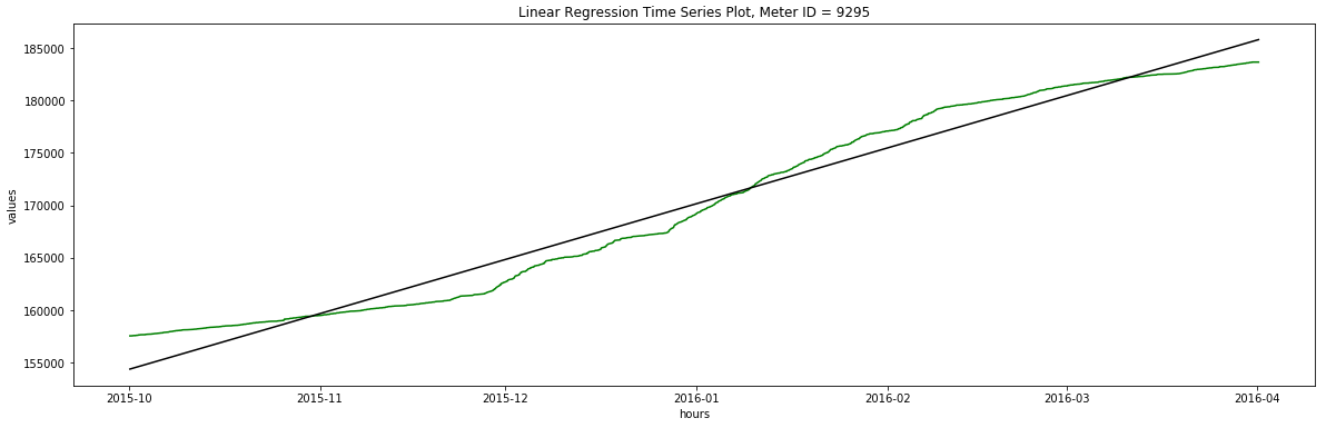

Linear Regression Time Series Plot, Meter ID = 7674
values
295000
290000
285000
280000
275000
270000
265000
260000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

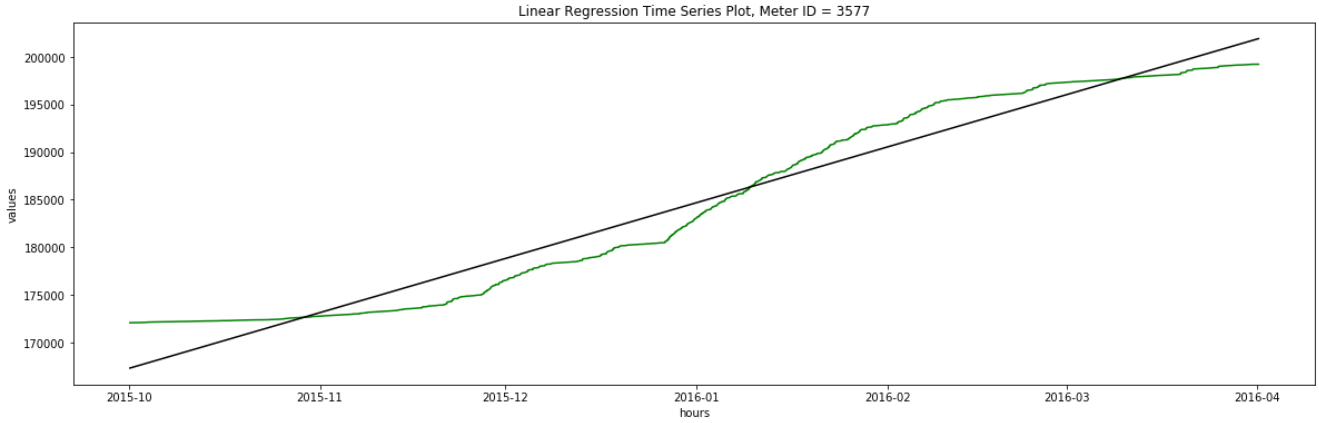

Linear Regression Time Series Plot, Meter ID = 7030
values
165000
160000
155000
150000
145000
140000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

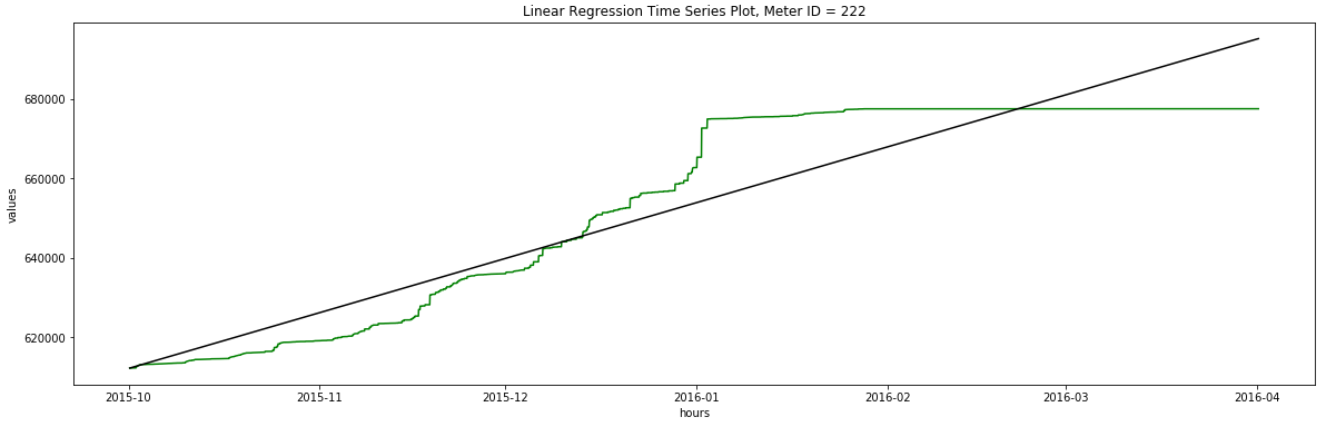

Linear Regression Time Series Plot, Meter ID = 4732
values
200000
195000
190000
185000
180000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

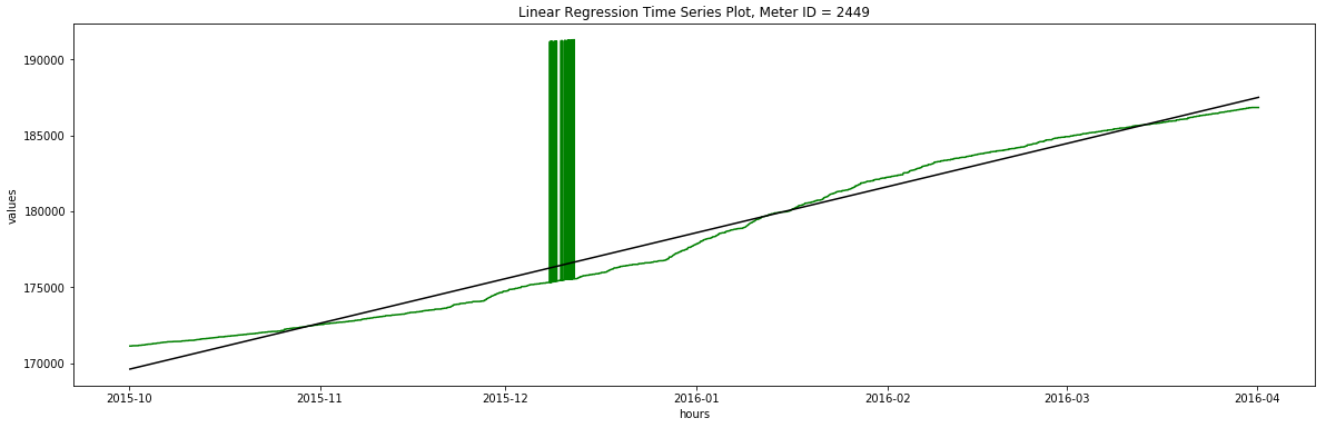

Linear Regression Time Series Plot, Meter ID = 370
values
102000
100000
98000
96000
94000
92000
90000
88000
86000

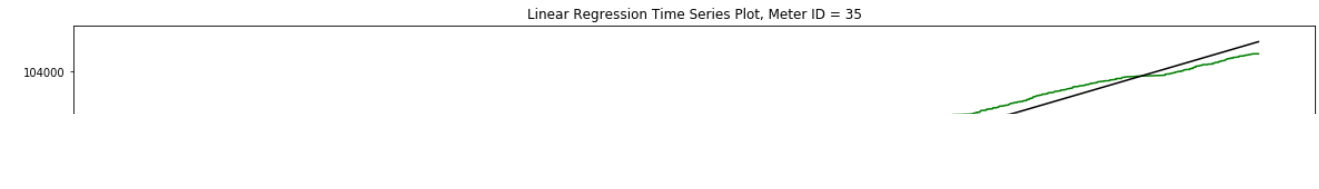

2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

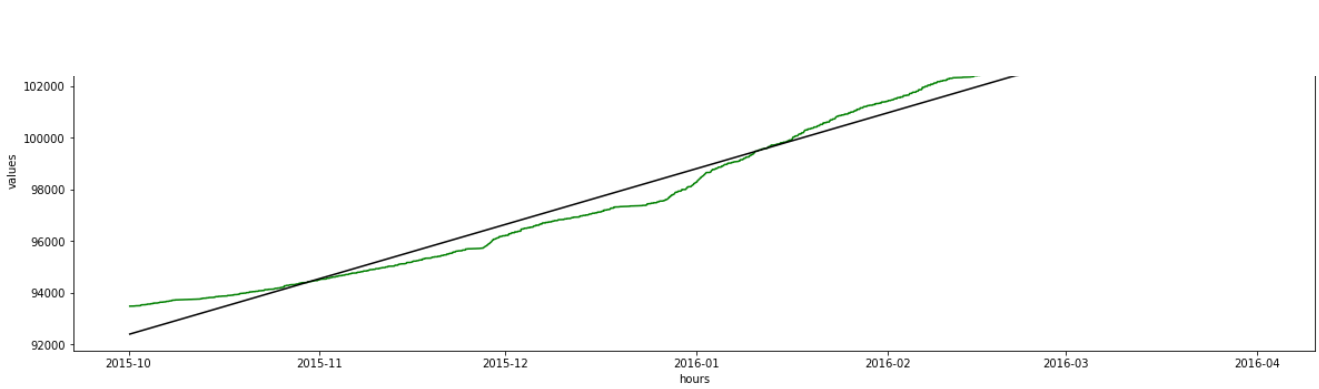

Linear Regression Time Series Plot, Meter ID = 9295
values
185000
180000
175000
170000
165000
160000
155000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

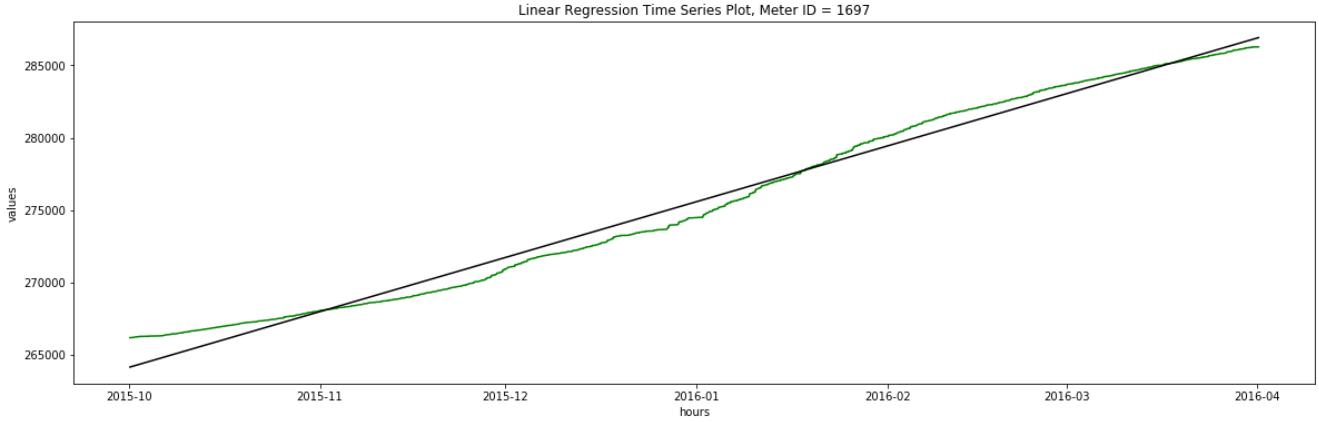

Linear Regression Time Series Plot, Meter ID = 3577
values
200000
195000
190000
185000
180000
175000
170000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

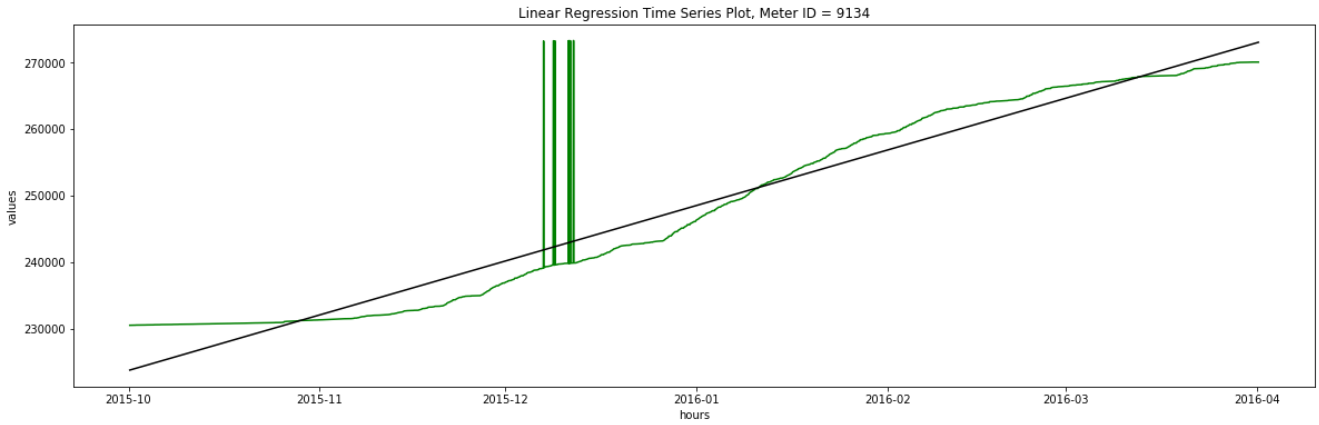

Linear Regression Time Series Plot, Meter ID = 222
values
680000
660000
640000
620000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

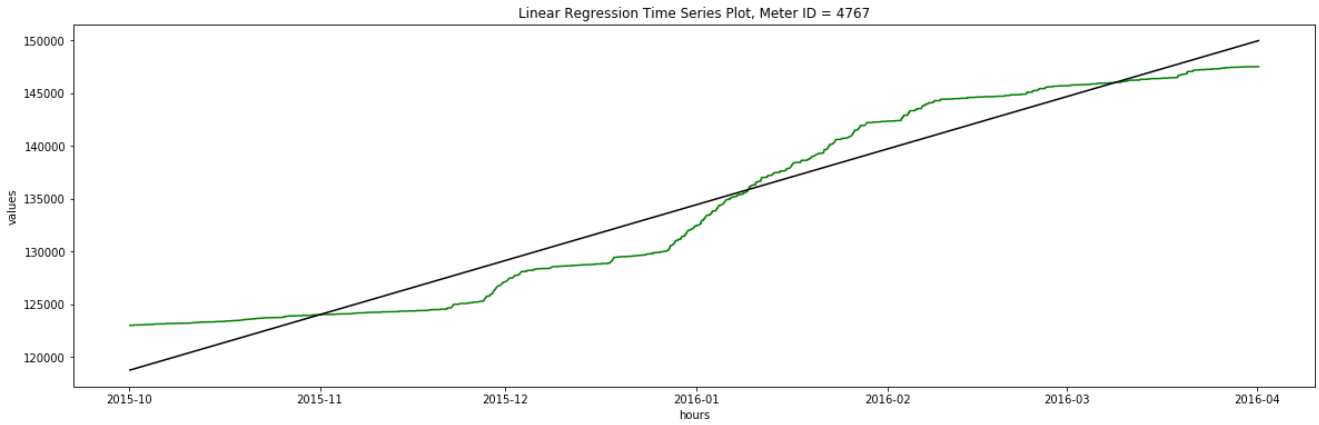

Linear Regression Time Series Plot, Meter ID = 2449
values
190000
185000
180000
175000
170000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

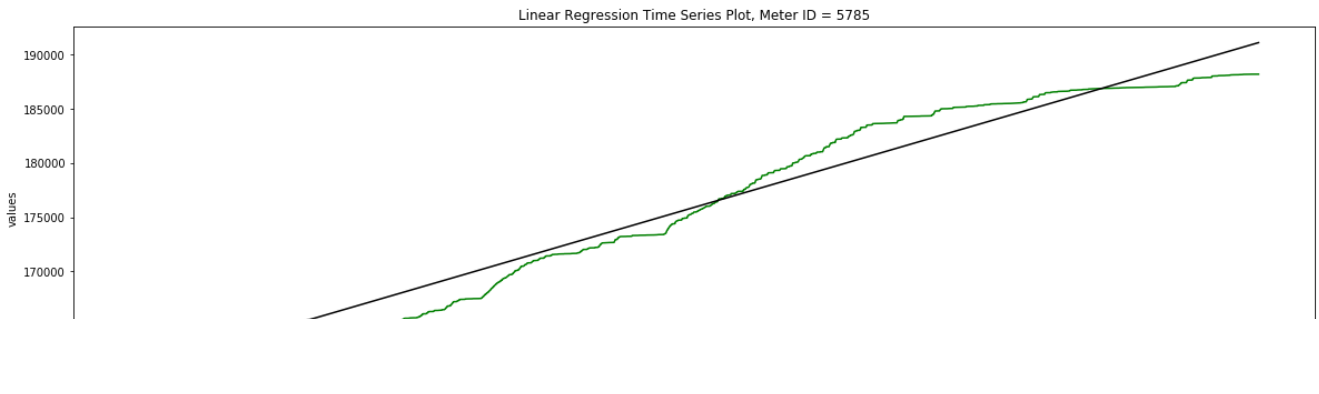

Linear Regression Time Series Plot, Meter ID = 35
104000


values
hours
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

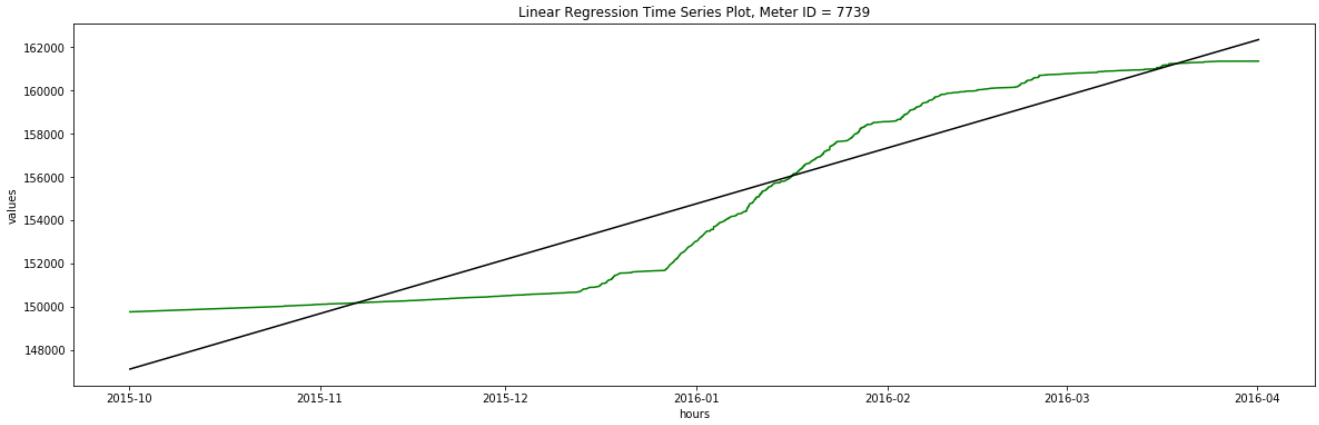

Linear Regression Time Series Plot, Meter ID = 1697
values
hours

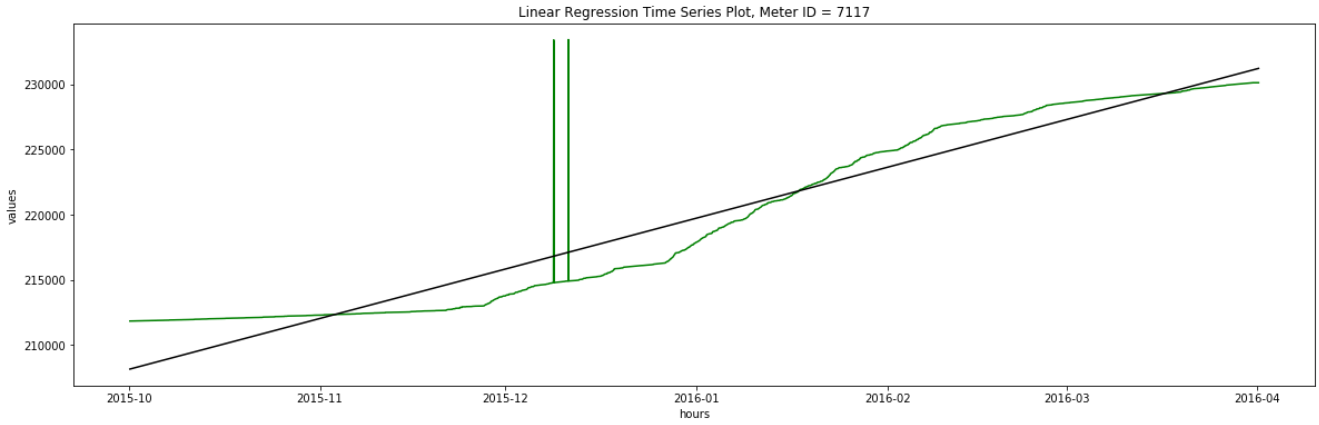

Linear Regression Time Series Plot, Meter ID = 9134
values
hours

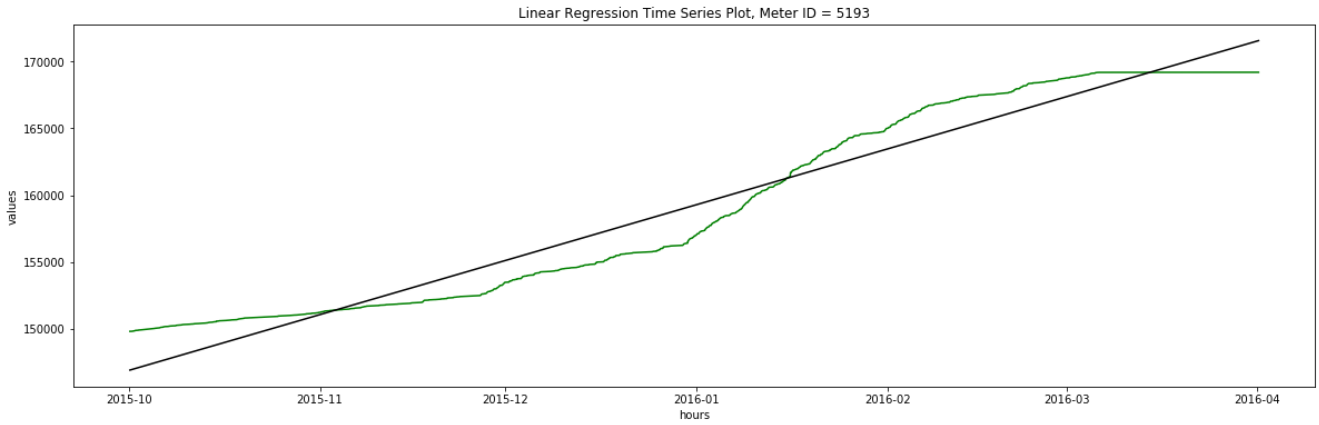

Linear Regression Time Series Plot, Meter ID = 4767
values
hours

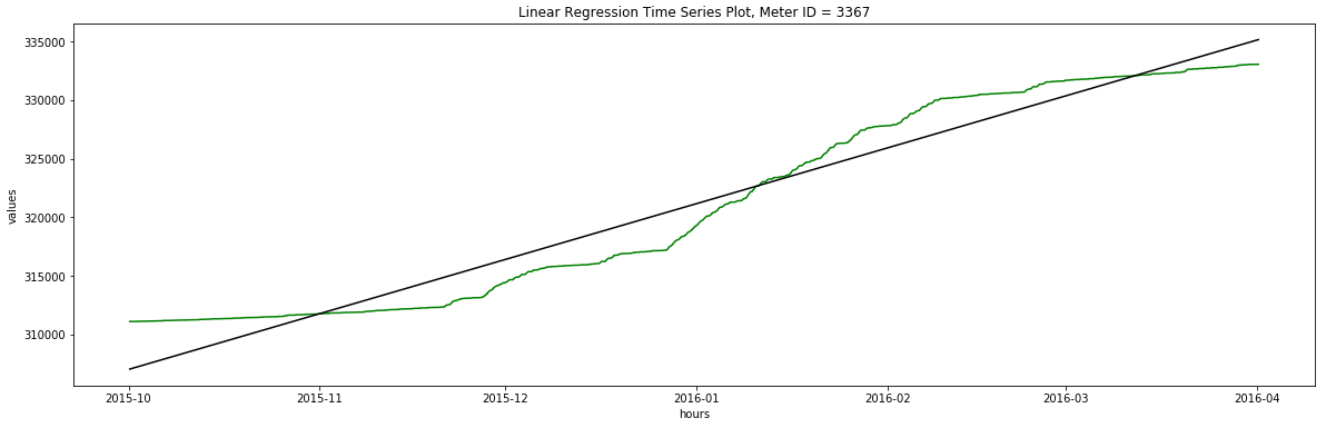

Linear Regression Time Series Plot, Meter ID = 5785
values


165000
160000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

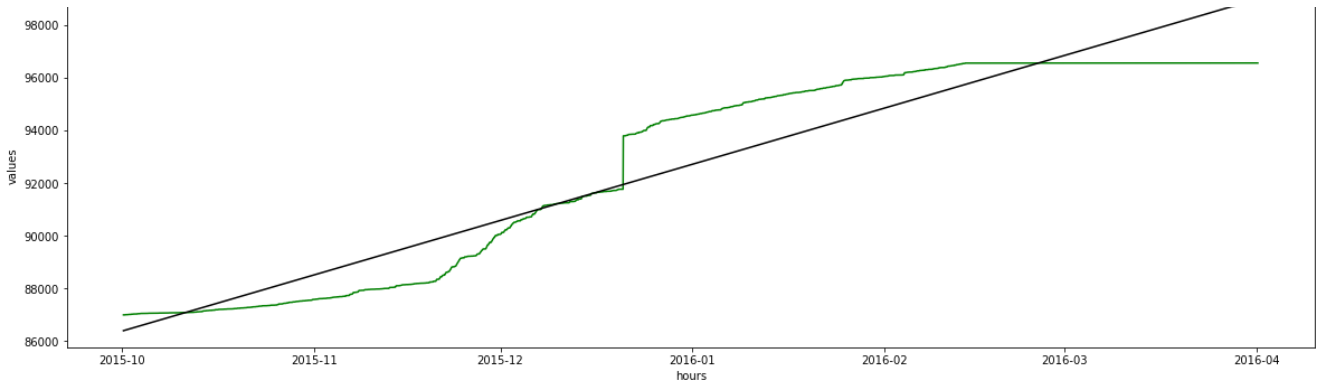

Linear Regression Time Series Plot, Meter ID = 7739
values
162000
160000
158000
156000
154000
152000
150000
148000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

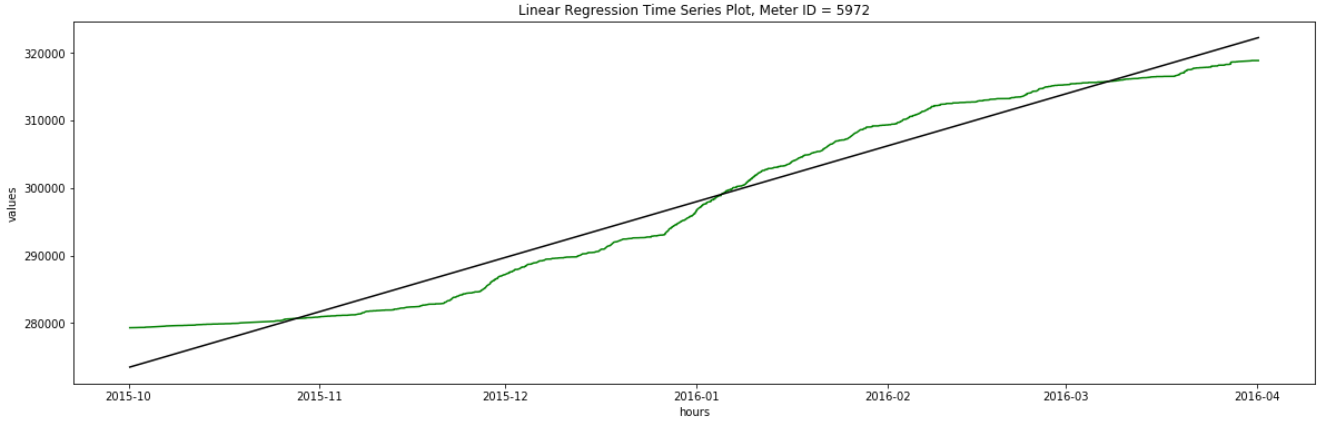

Linear Regression Time Series Plot, Meter ID = 7117
values
230000
225000
220000
215000
210000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

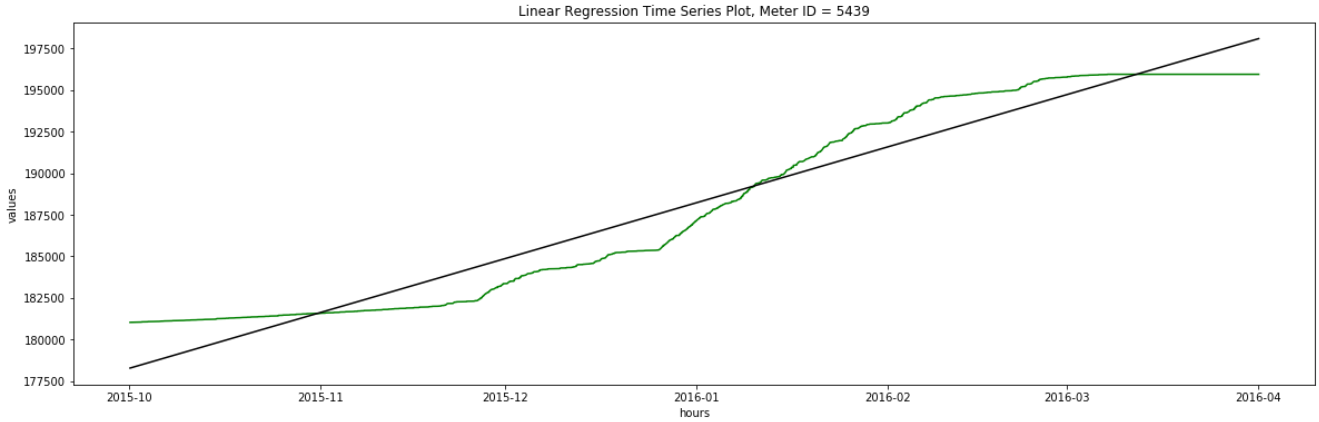

Linear Regression Time Series Plot, Meter ID = 5193
values
170000
165000
160000
155000
150000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

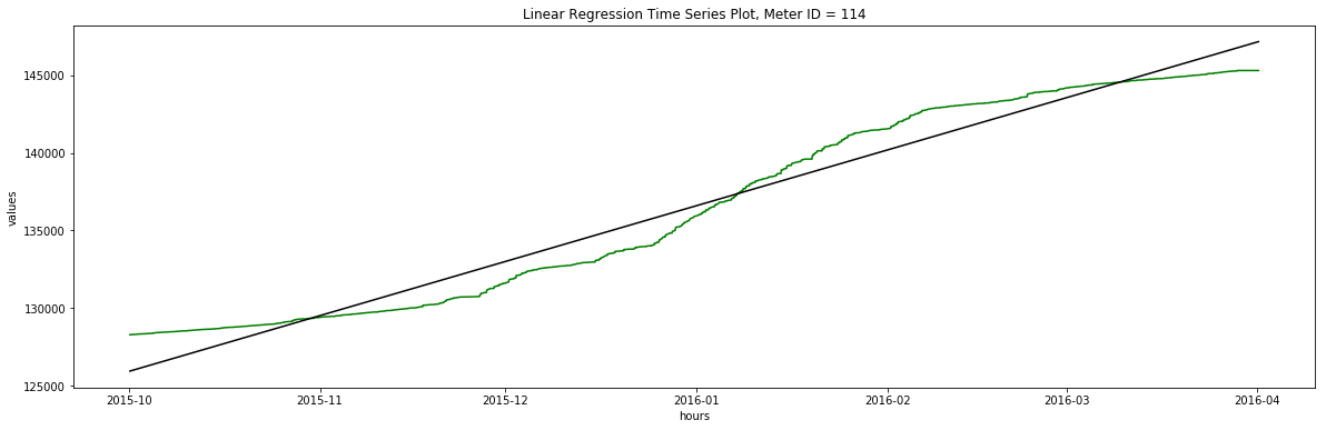

Linear Regression Time Series Plot, Meter ID = 3367
values
335000
330000
325000
320000
315000
310000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

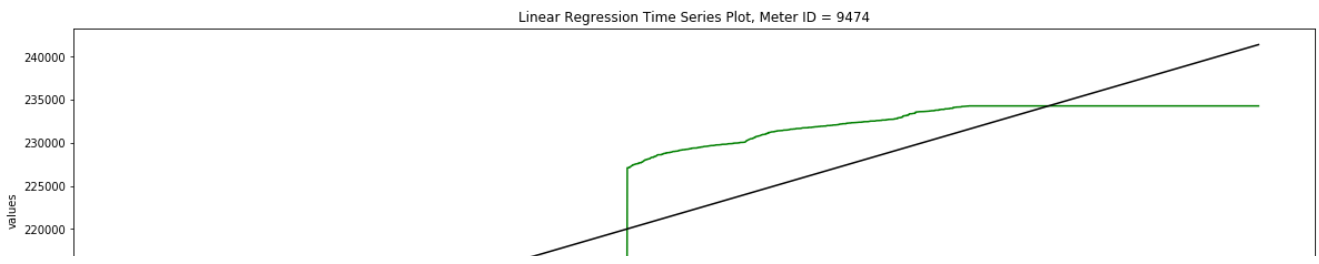

Linear Regression Time Series Plot, Meter ID = 8467

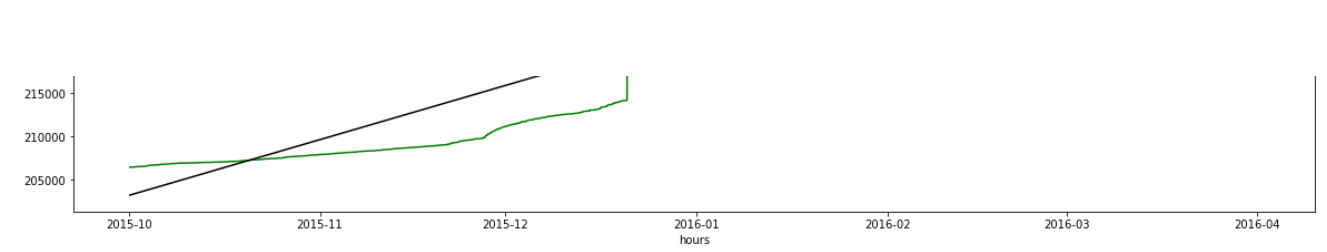

values
hours
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

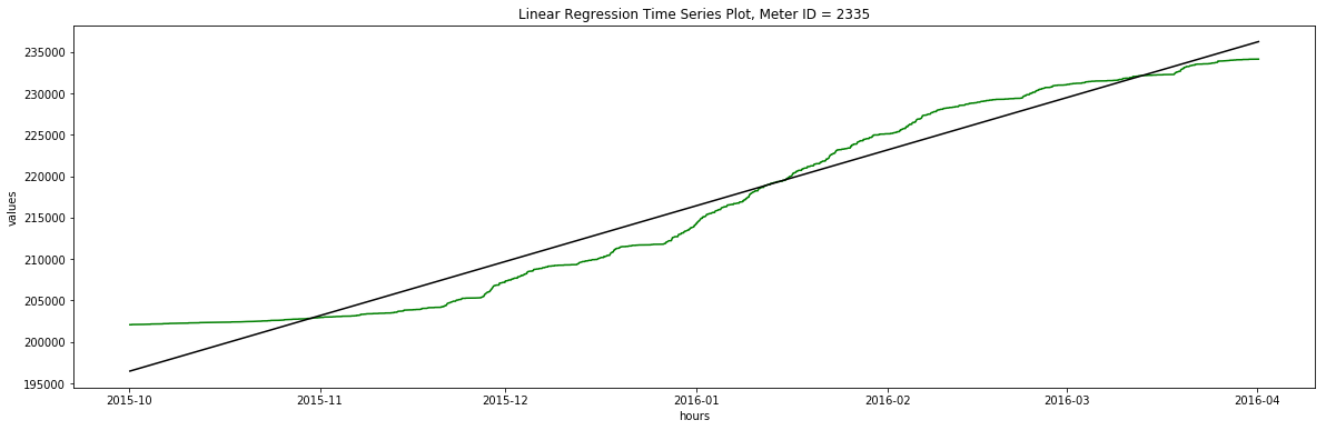

Linear Regression Time Series Plot, Meter ID = 5972
values
hours

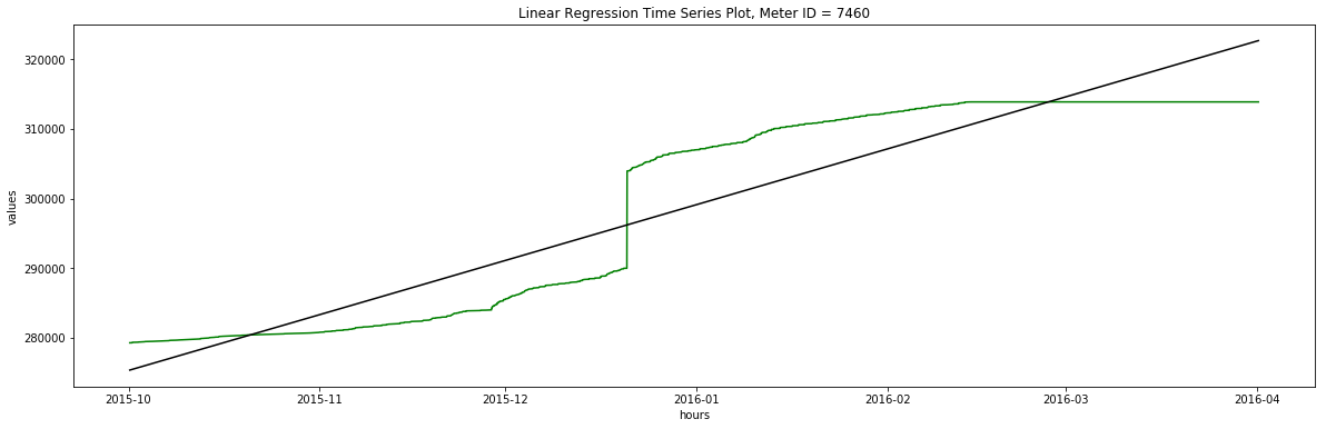

Linear Regression Time Series Plot, Meter ID = 5439
values
hours

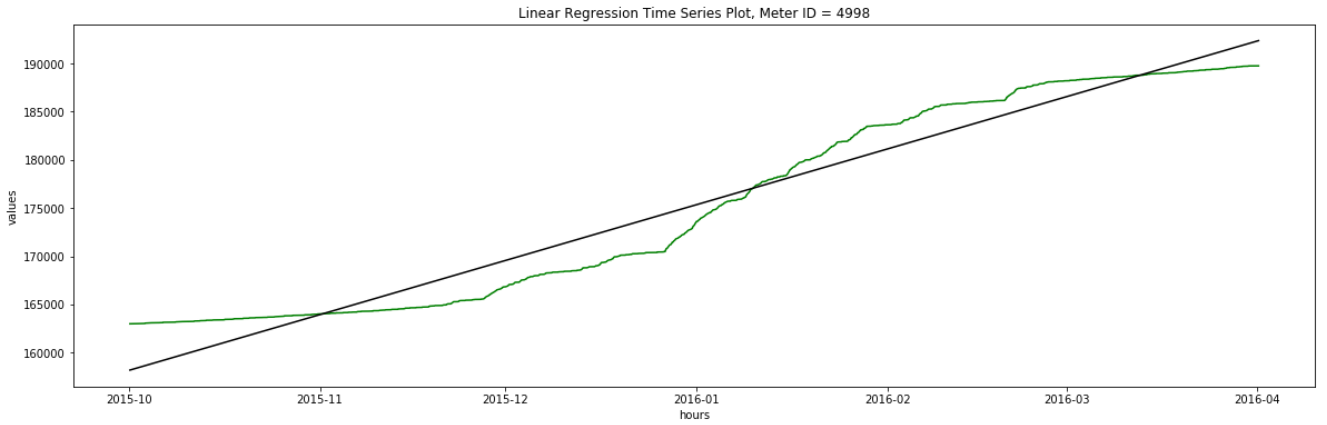

Linear Regression Time Series Plot, Meter ID = 114
values
hours

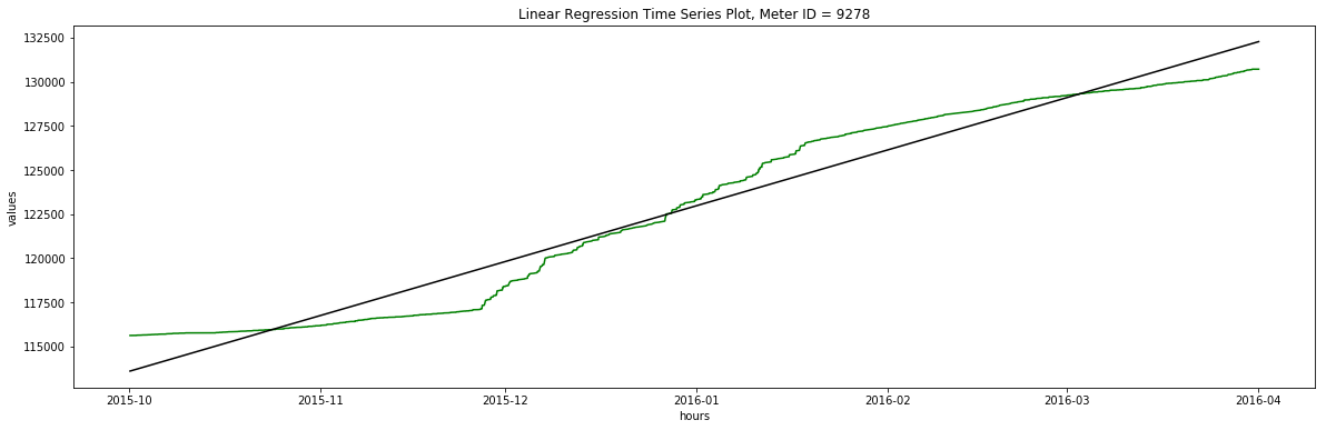

Linear Regression Time Series Plot, Meter ID = 9474
values

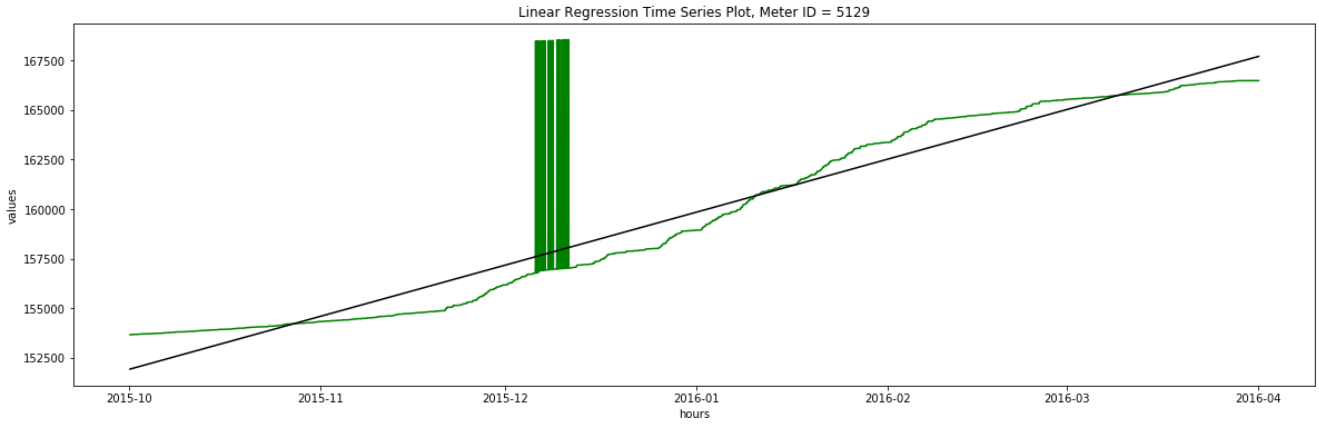

215000
210000
205000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

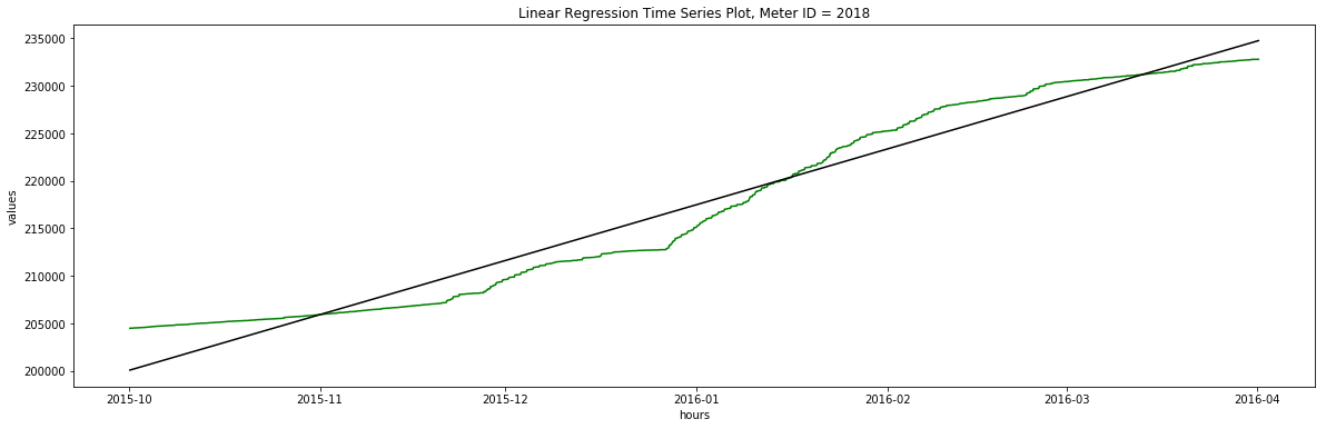

Linear Regression Time Series Plot, Meter ID = 2335
values
235000
230000
225000
220000
215000
210000
205000
200000
195000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

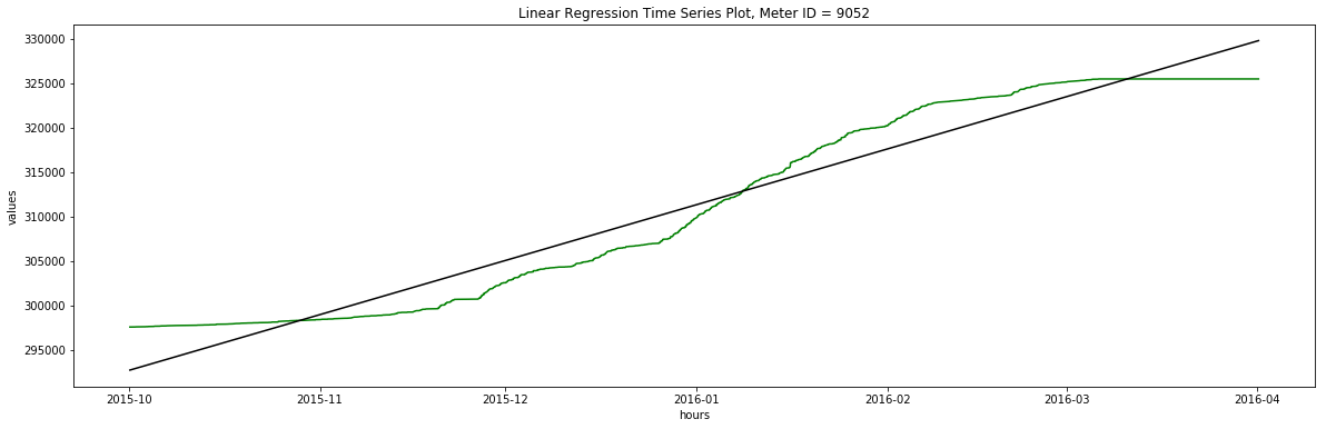

Linear Regression Time Series Plot, Meter ID = 7460
values
320000
310000
300000
290000
280000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

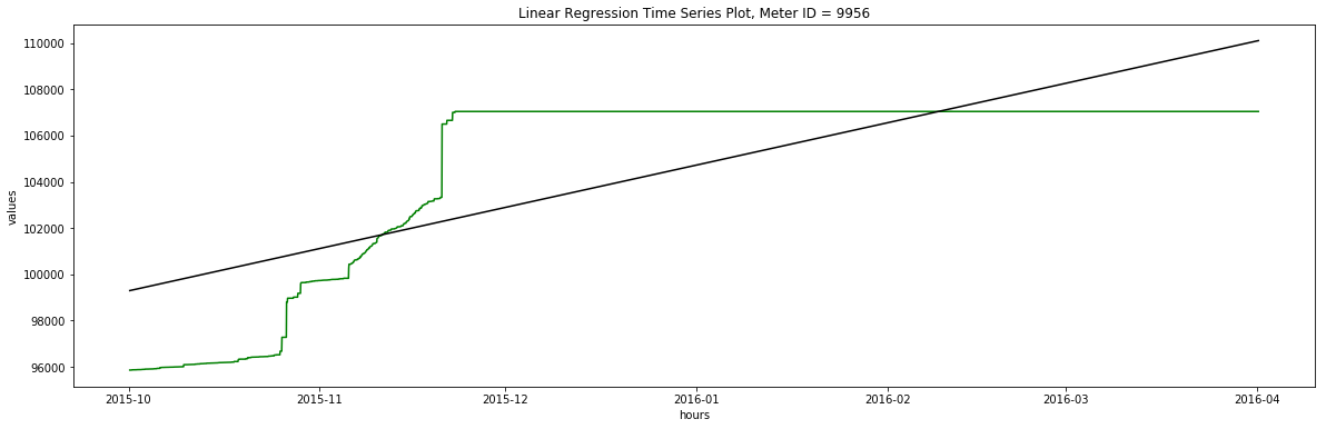

Linear Regression Time Series Plot, Meter ID = 4998
values
190000
185000
180000
175000
170000
165000
160000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

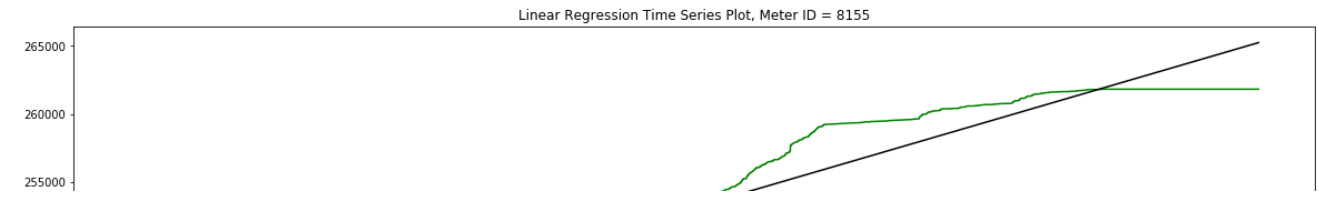

Linear Regression Time Series Plot, Meter ID = 9278
values
132500
130000
127500
125000
122500
120000
117500
115000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

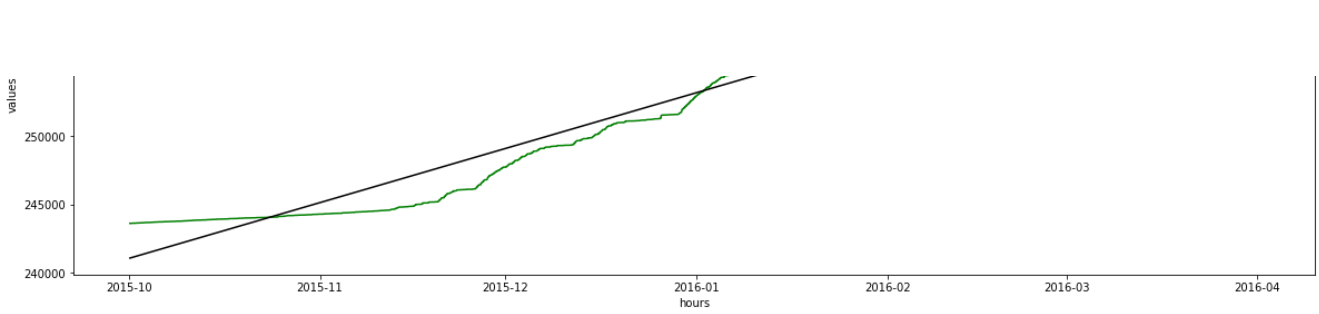

Linear Regression Time Series Plot, Meter ID = 5129
values
hours
167500
165000
162500
160000
157500
155000
152500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

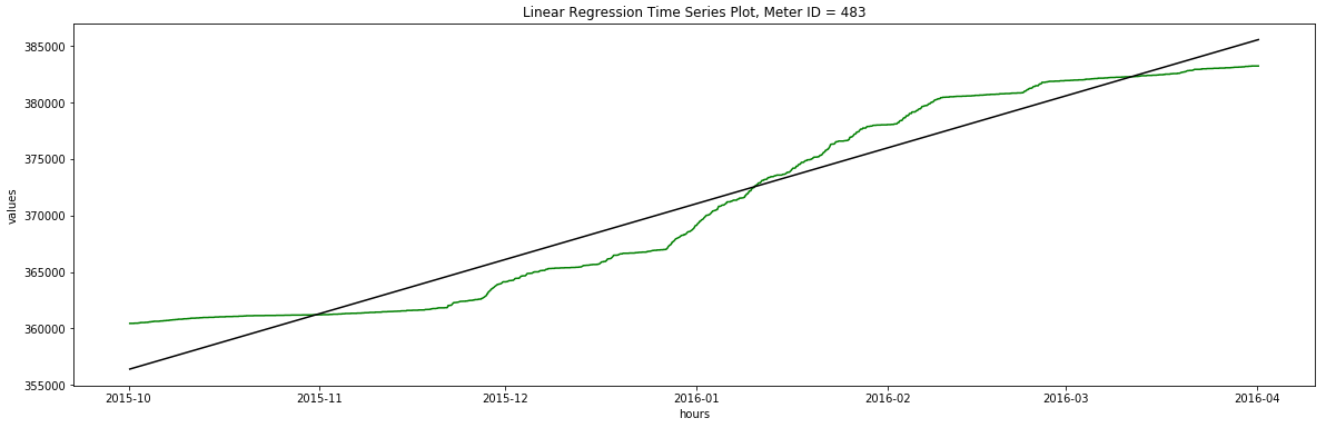

Linear Regression Time Series Plot, Meter ID = 2018
values
hours
235000
230000
225000
220000
215000
210000
205000
200000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

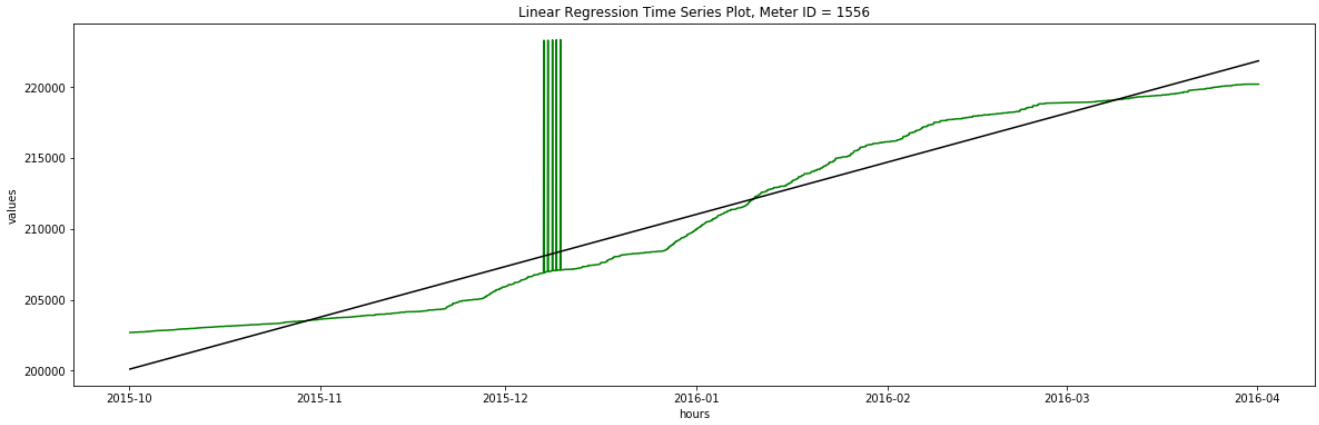

Linear Regression Time Series Plot, Meter ID = 9052
values
hours
330000
325000
320000
315000
310000
305000
300000
295000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

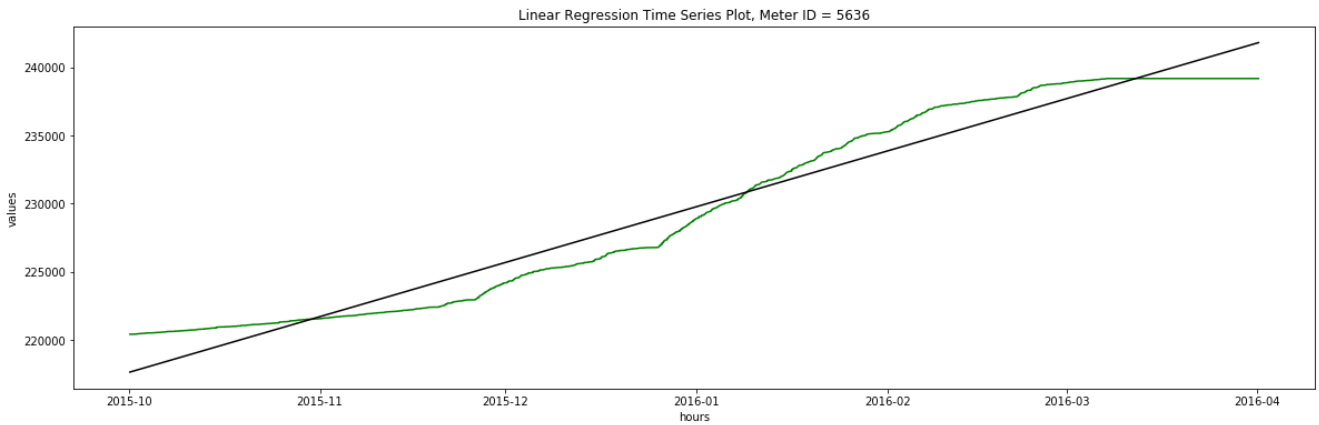

Linear Regression Time Series Plot, Meter ID = 9956
values
hours
110000
108000
106000
104000
102000
100000
98000
96000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

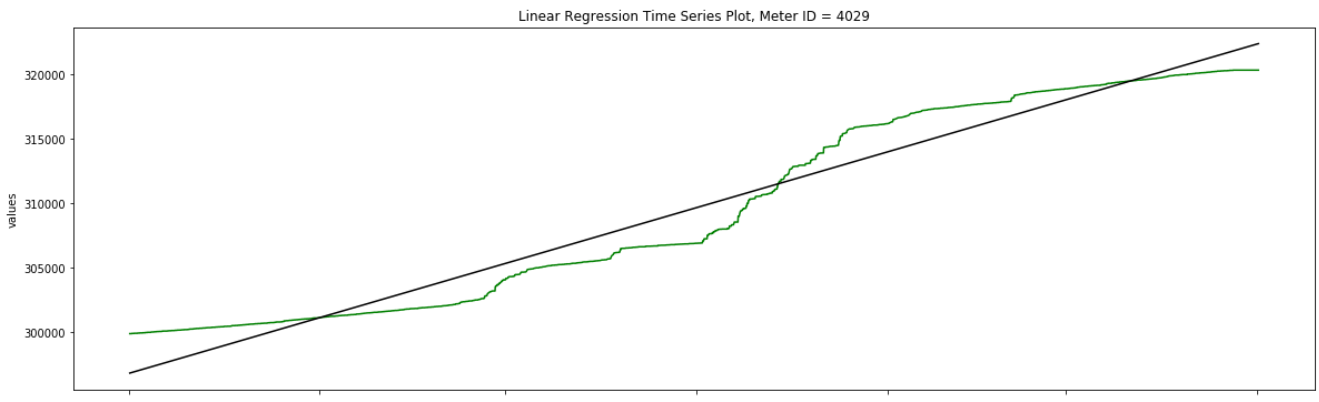

Linear Regression Time Series Plot, Meter ID = 8155
265000
260000
255000


values
250000
245000
240000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

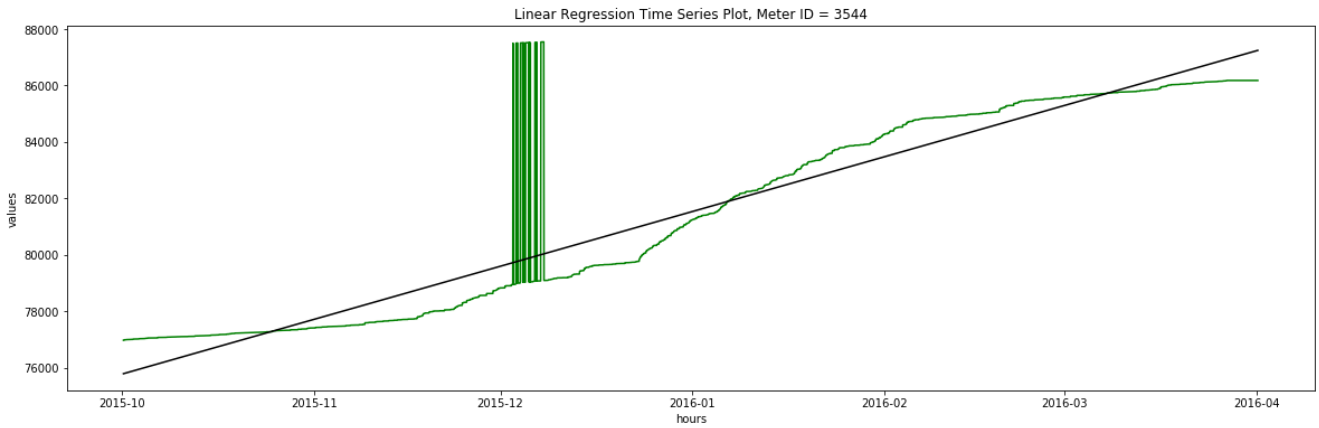

Linear Regression Time Series Plot, Meter ID = 483
values
385000
380000
375000
370000
365000
360000
355000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

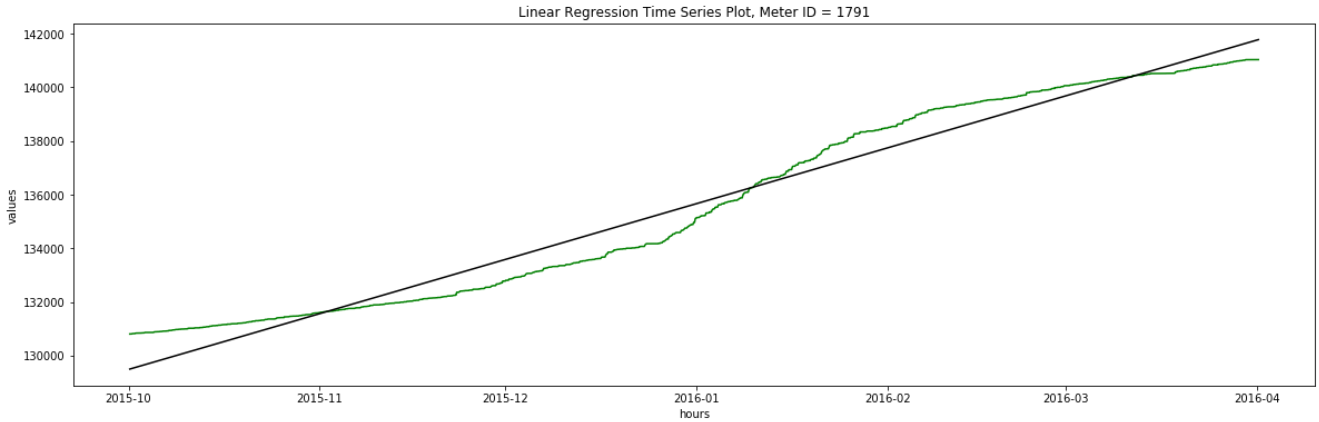

Linear Regression Time Series Plot, Meter ID = 1556
values
220000
215000
210000
205000
200000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

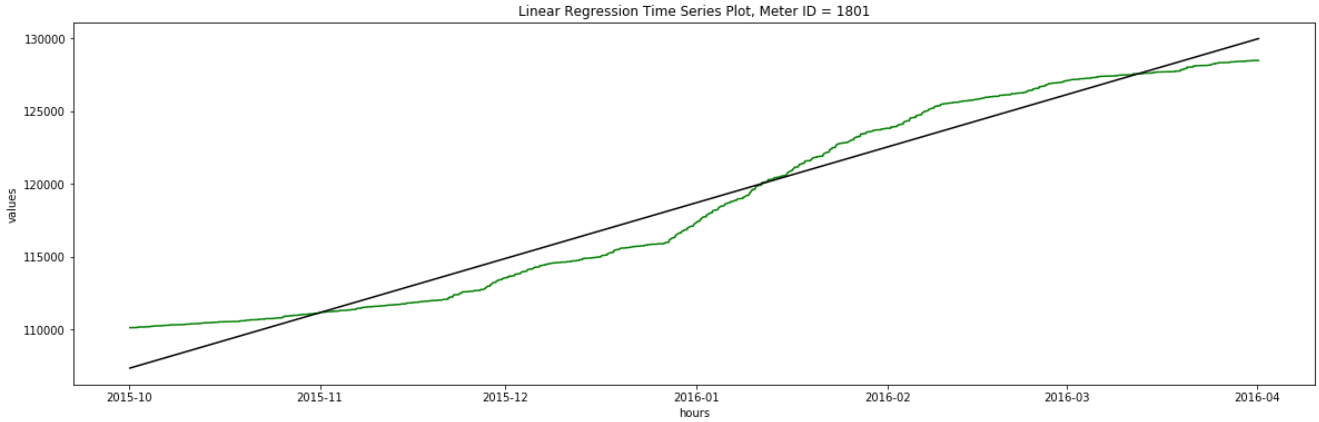

Linear Regression Time Series Plot, Meter ID = 5636
values
240000
235000
230000
225000
220000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

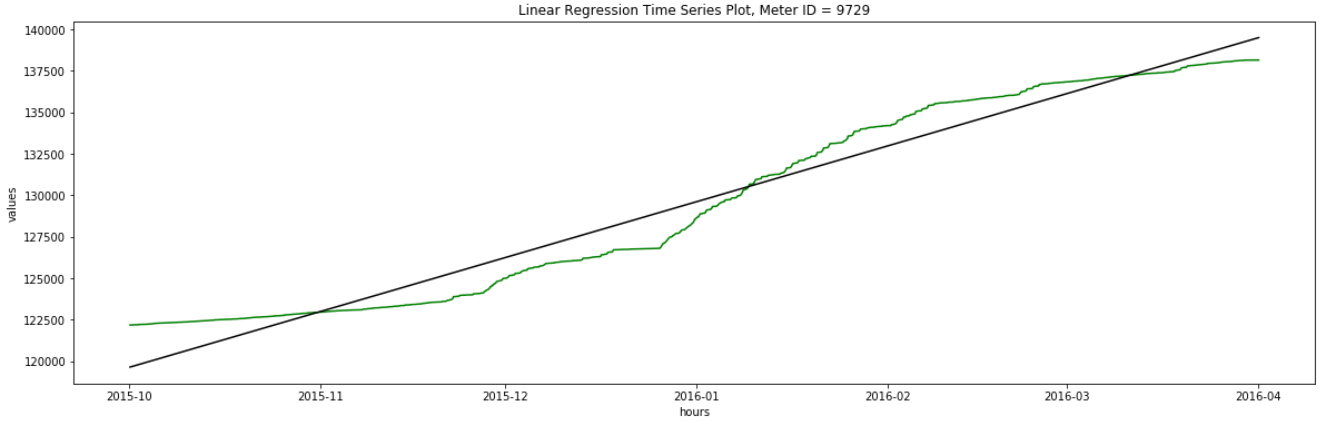

Linear Regression Time Series Plot, Meter ID = 4029
values
320000
315000
310000
305000
300000

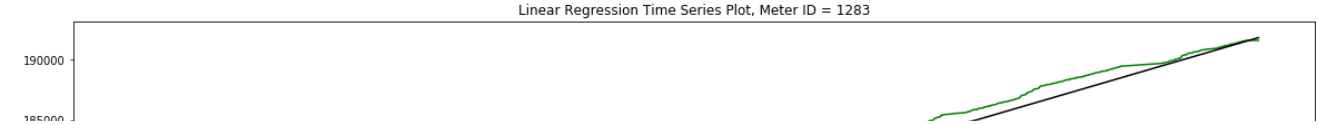

2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

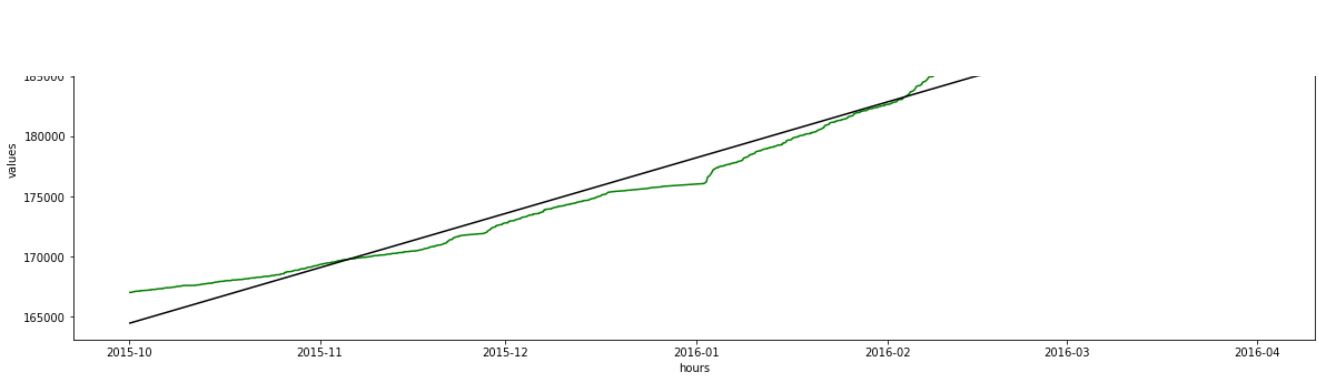

Linear Regression Time Series Plot, Meter ID = 3544
values
hours

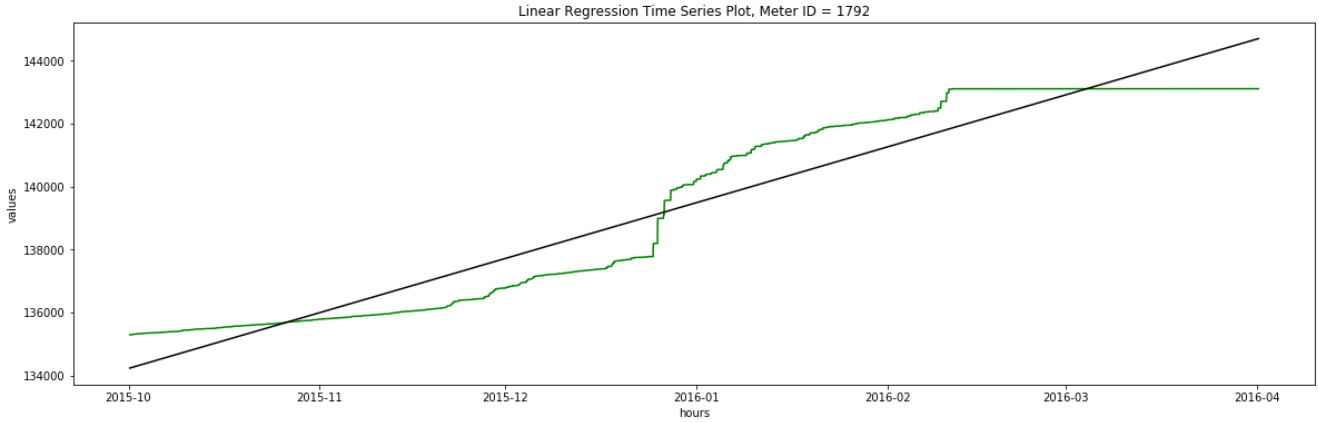

Linear Regression Time Series Plot, Meter ID = 1791
values
hours

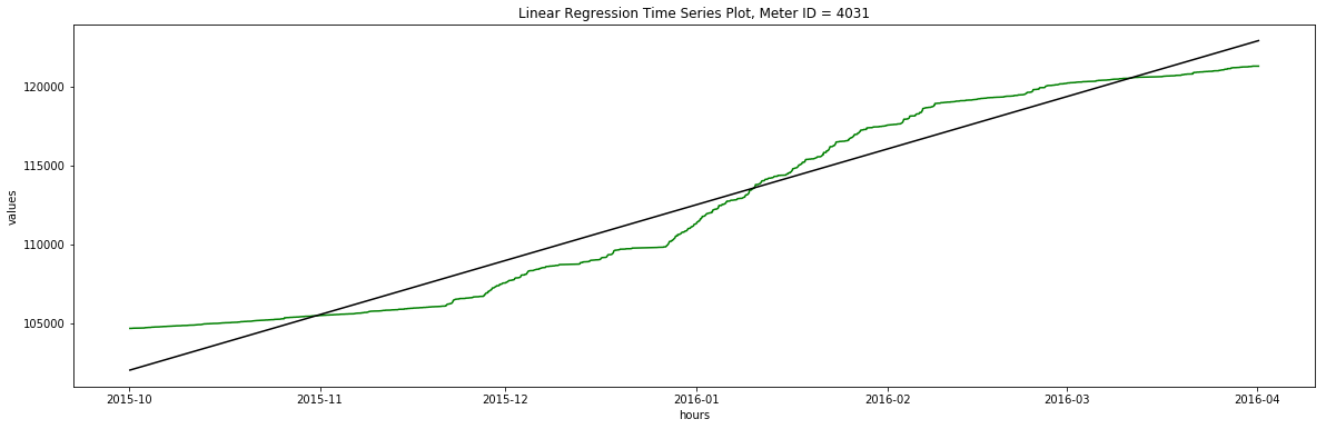

Linear Regression Time Series Plot, Meter ID = 1801
values
hours

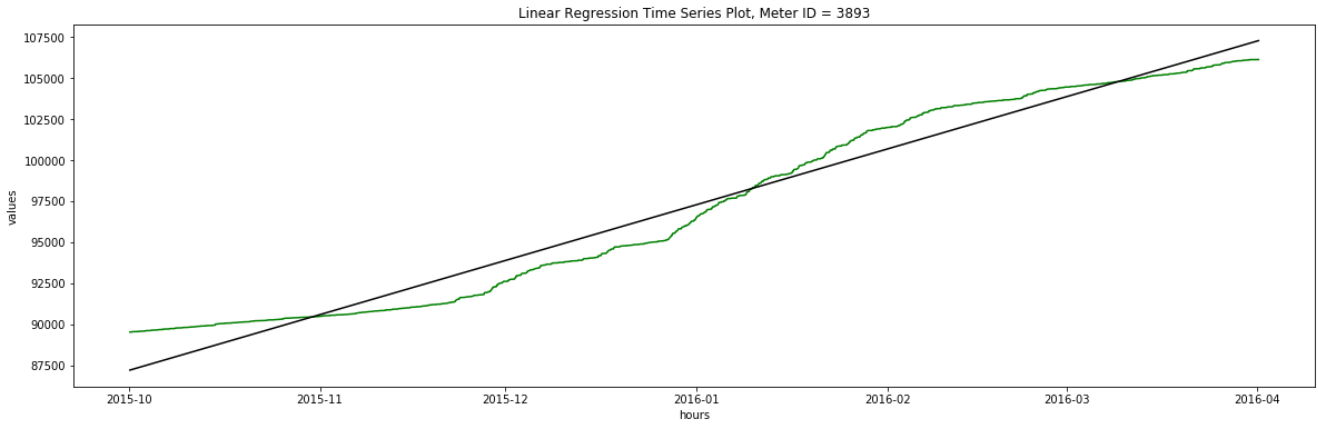

Linear Regression Time Series Plot, Meter ID = 9729
values
hours

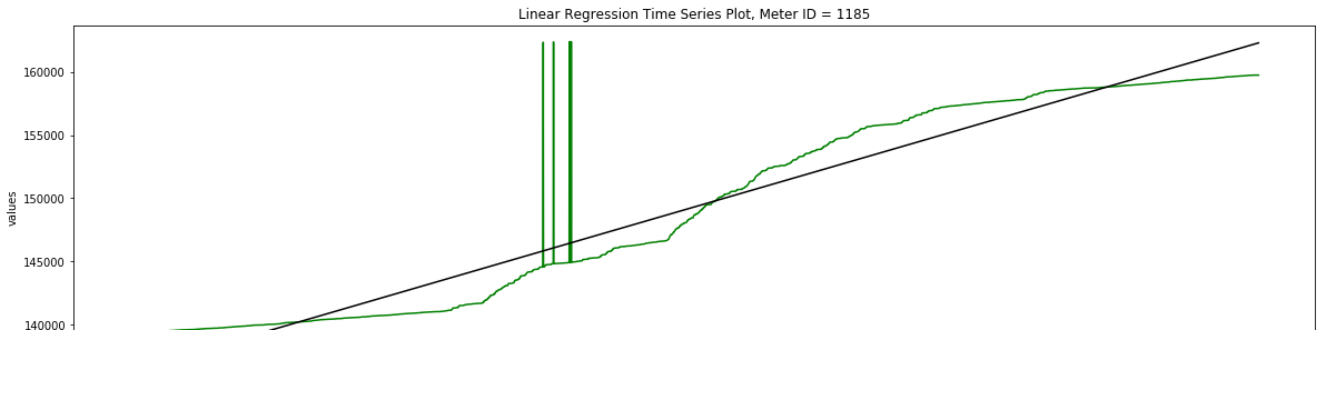

Linear Regression Time Series Plot, Meter ID = 1283

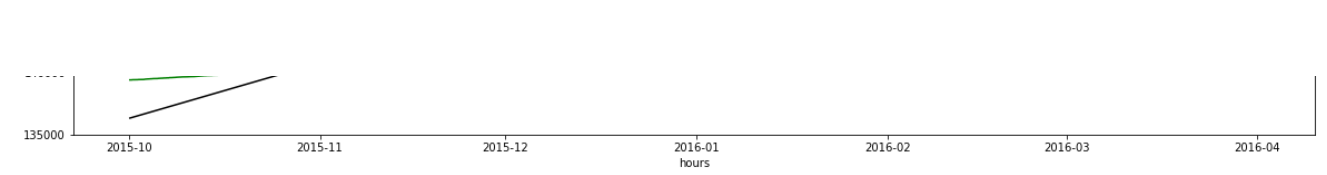

values
180000
175000
170000
165000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

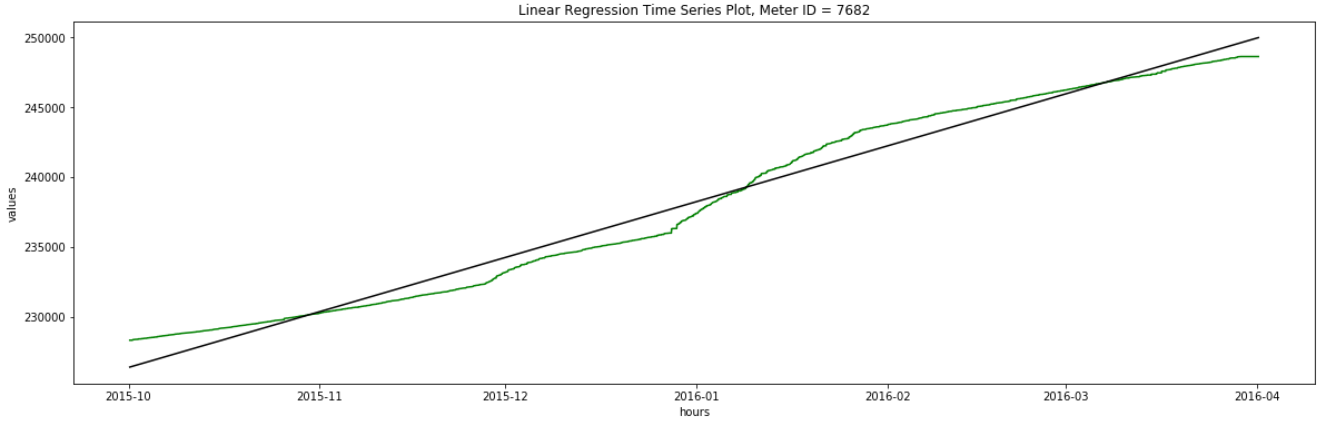

Linear Regression Time Series Plot, Meter ID = 1792
values
144000
142000
140000
138000
136000
134000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

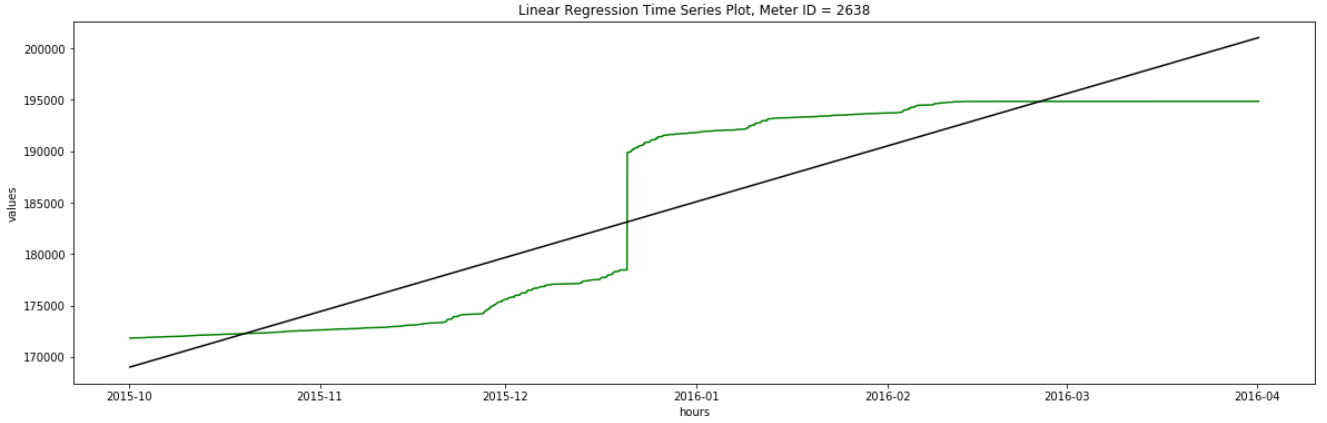

Linear Regression Time Series Plot, Meter ID = 4031
values
120000
115000
110000
105000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

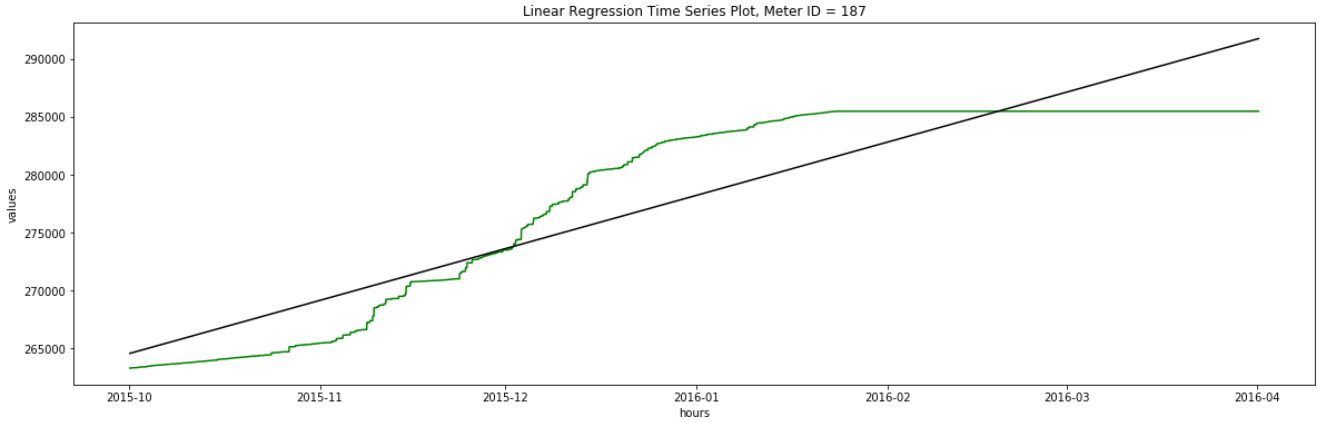

Linear Regression Time Series Plot, Meter ID = 3893
values
107500
105000
102500
100000
97500
95000
92500
90000
87500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

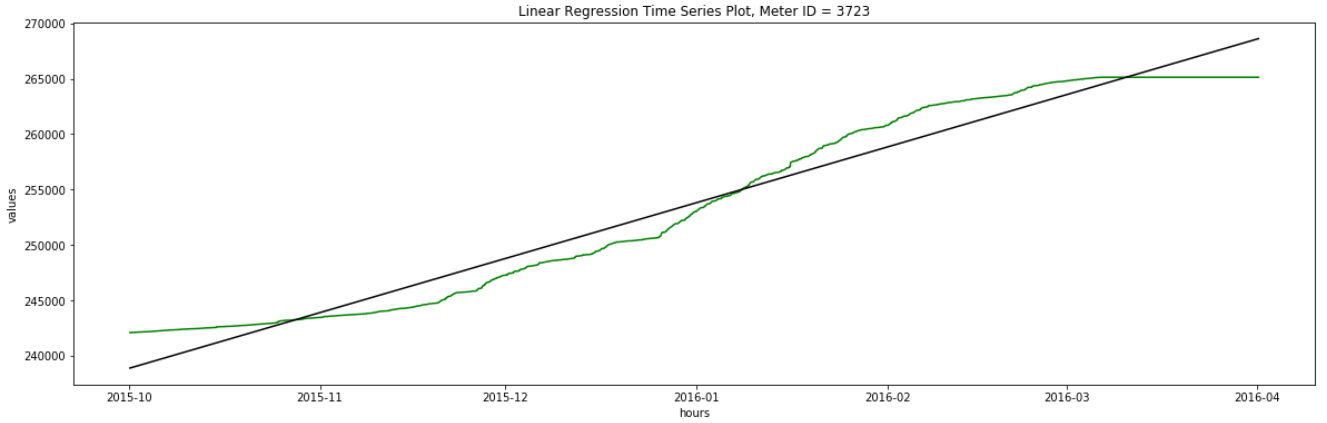

Linear Regression Time Series Plot, Meter ID = 1185
values
160000
155000
150000
145000
140000

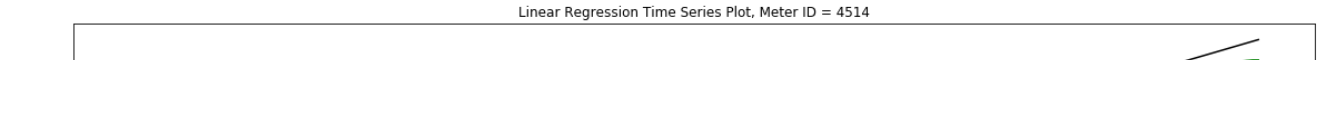

135000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

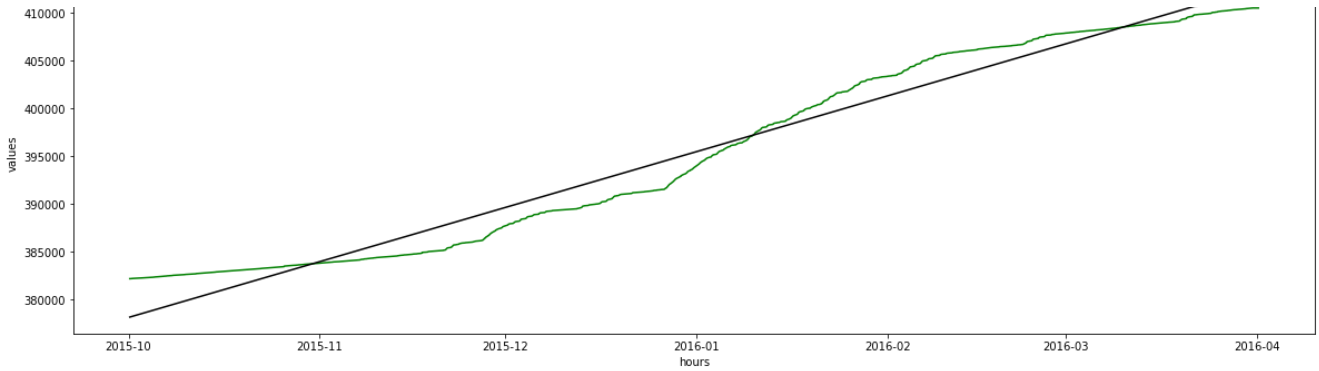

Linear Regression Time Series Plot, Meter ID = 7682
250000
245000
240000
235000
230000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

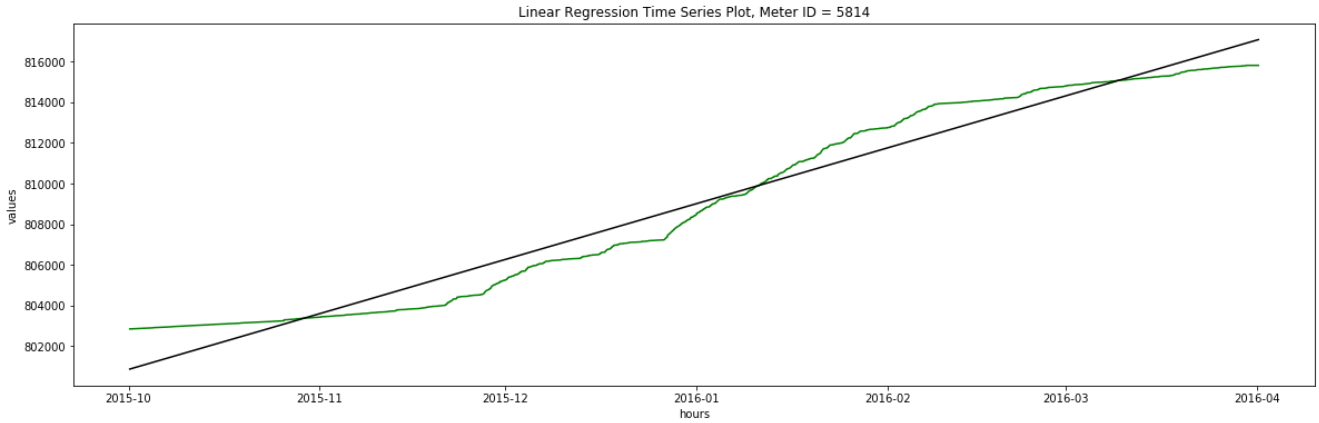

Linear Regression Time Series Plot, Meter ID = 2638
200000
195000
190000
185000
180000
175000
170000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

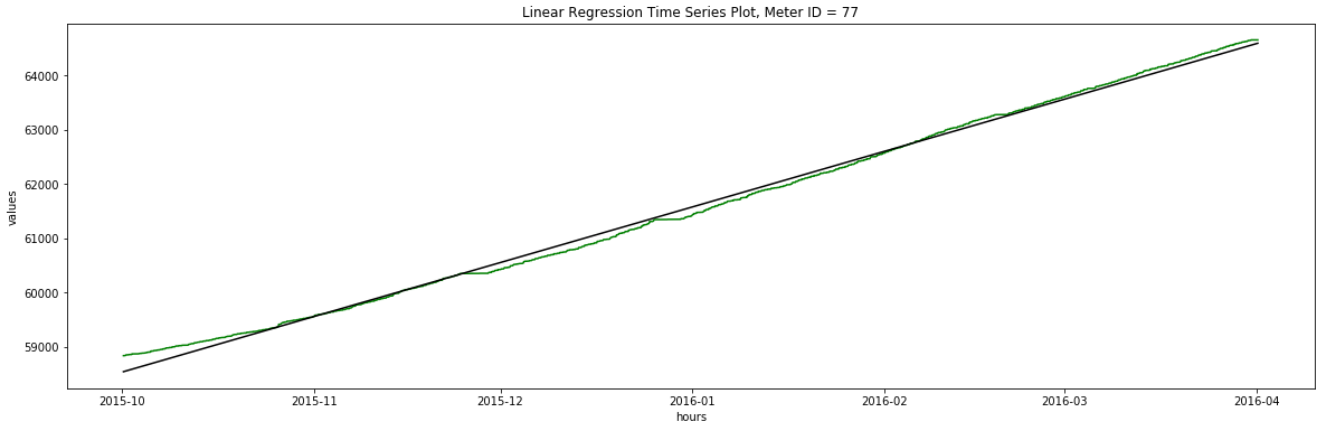

Linear Regression Time Series Plot, Meter ID = 187
290000
285000
280000
275000
270000
265000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

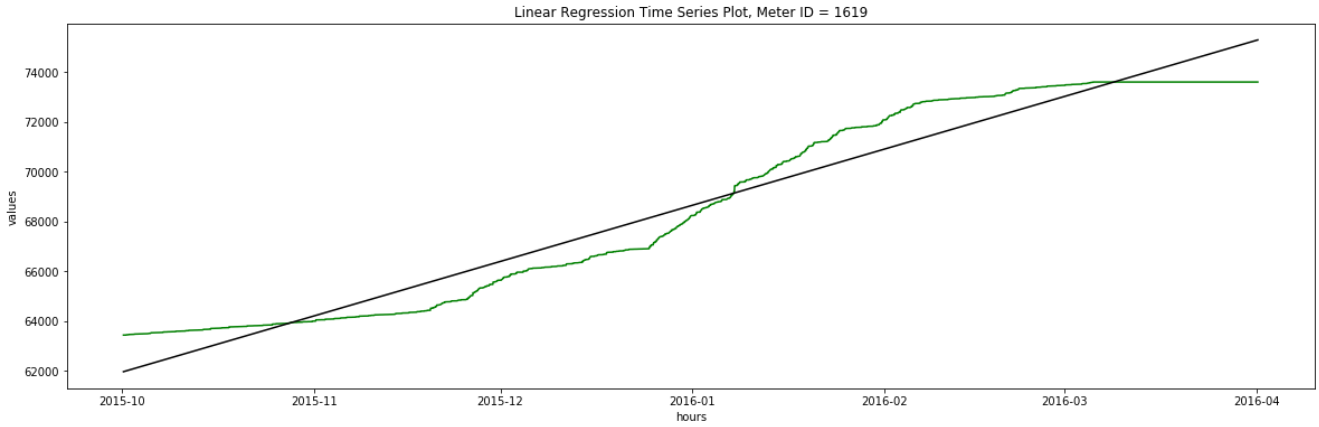

Linear Regression Time Series Plot, Meter ID = 3723
270000
265000
260000
255000
250000
245000
240000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours


Linear Regression Time Series Plot, Meter ID = 4514

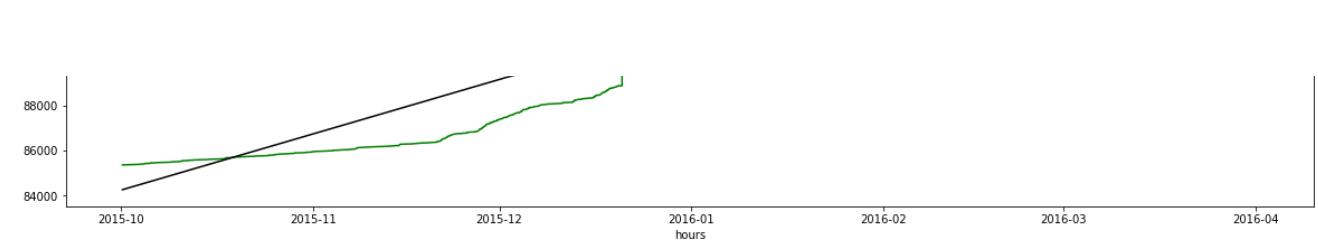

410000
405000
400000
395000
390000
385000
380000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

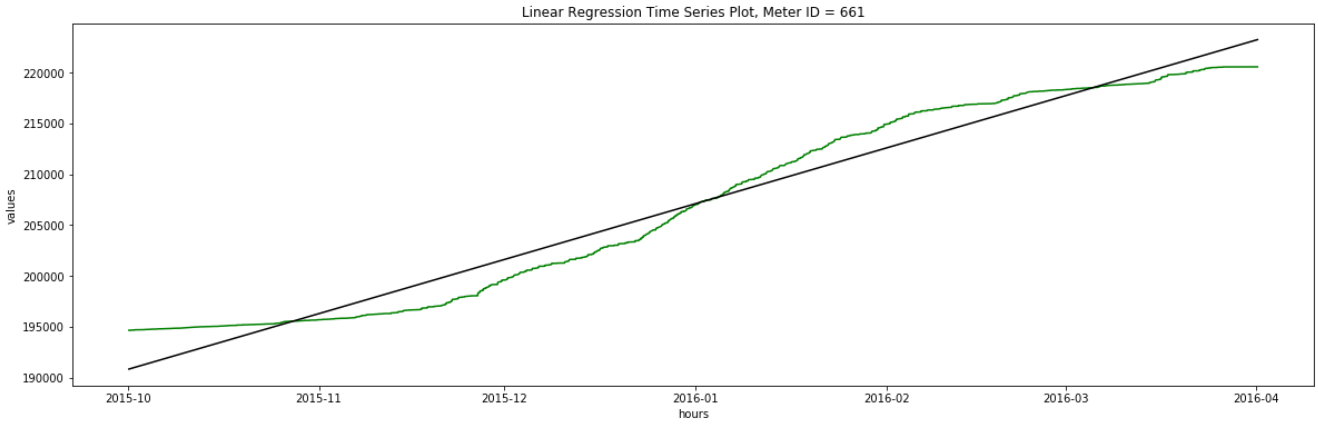

Linear Regression Time Series Plot, Meter ID = 5814
816000
814000
812000
810000
808000
806000
804000
802000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

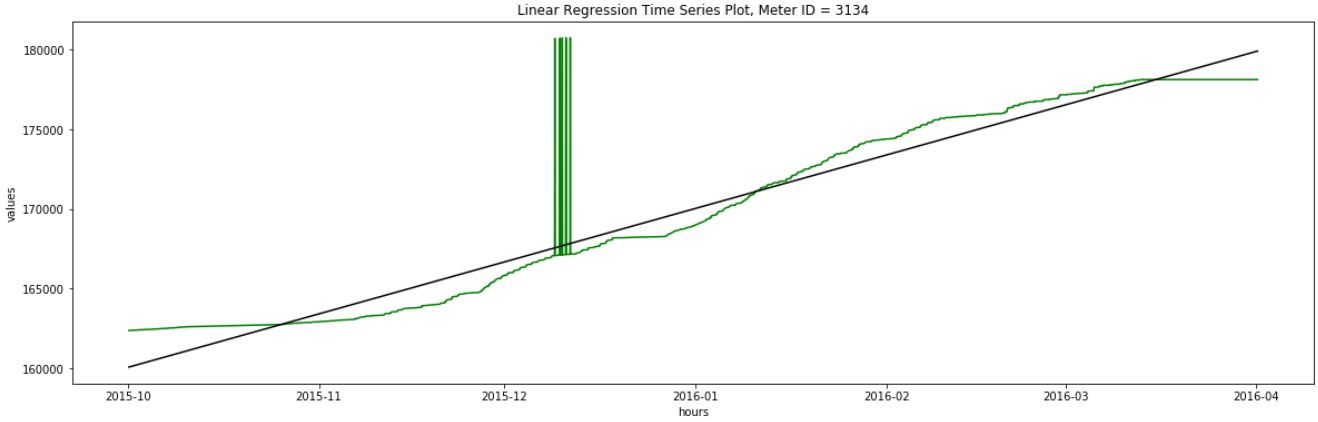

Linear Regression Time Series Plot, Meter ID = 77
64000
63000
62000
61000
60000
59000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

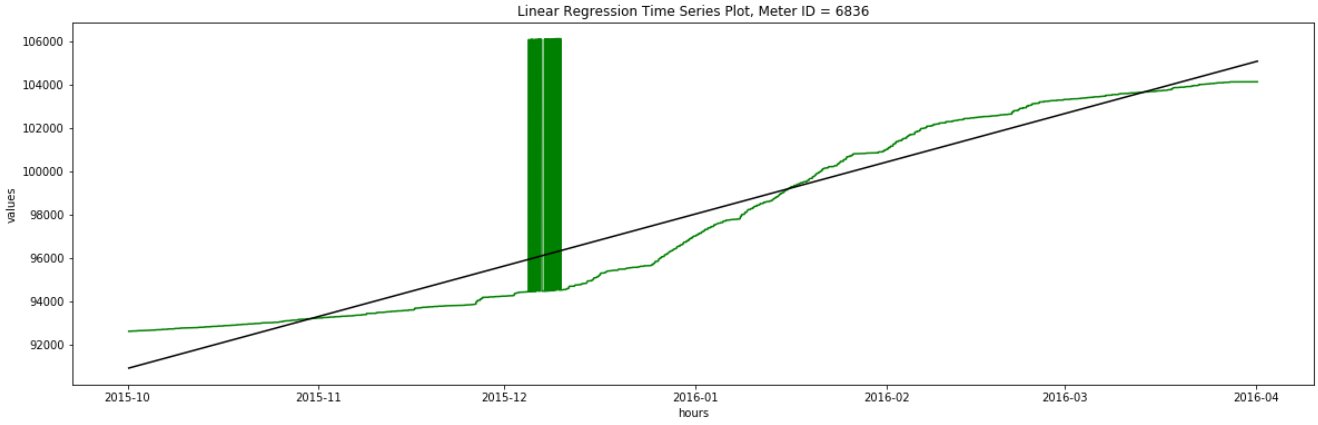

Linear Regression Time Series Plot, Meter ID = 1619
74000
72000
70000
68000
66000
64000
62000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

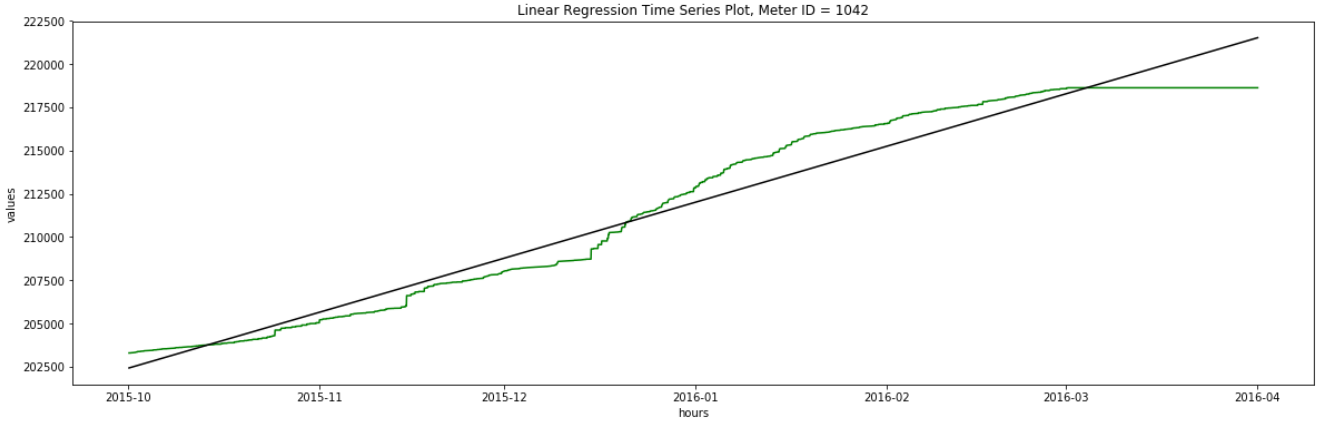

Linear Regression Time Series Plot, Meter ID = 4421
98000
96000
94000
92000
90000
values

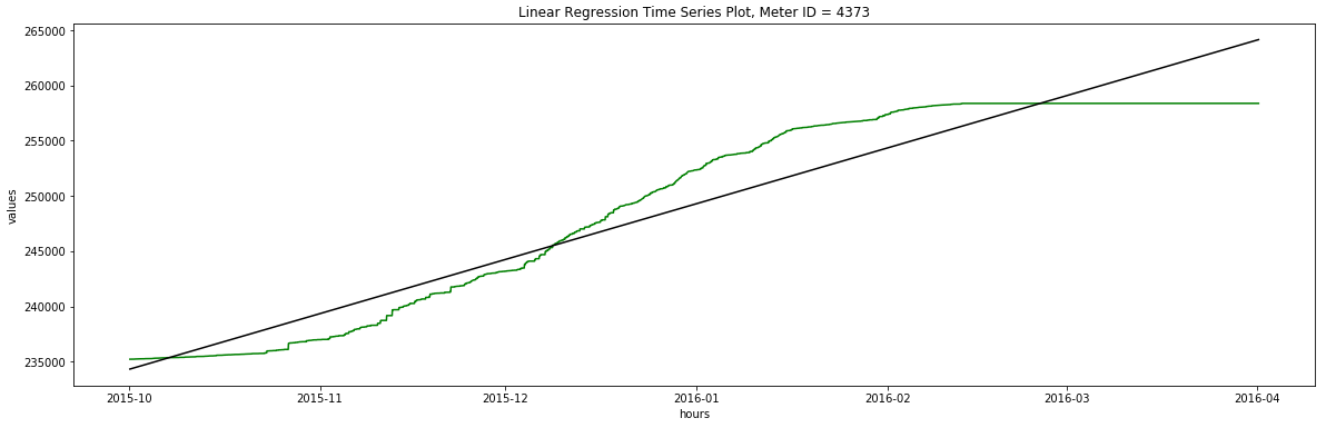

88000
86000
84000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

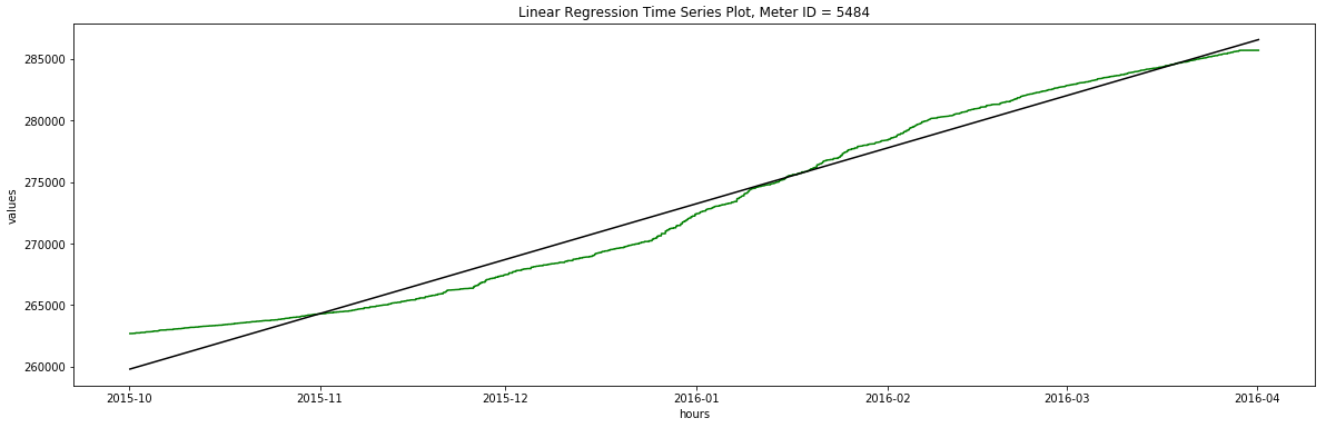

Linear Regression Time Series Plot, Meter ID = 661
values
220000
215000
210000
205000
200000
195000
190000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

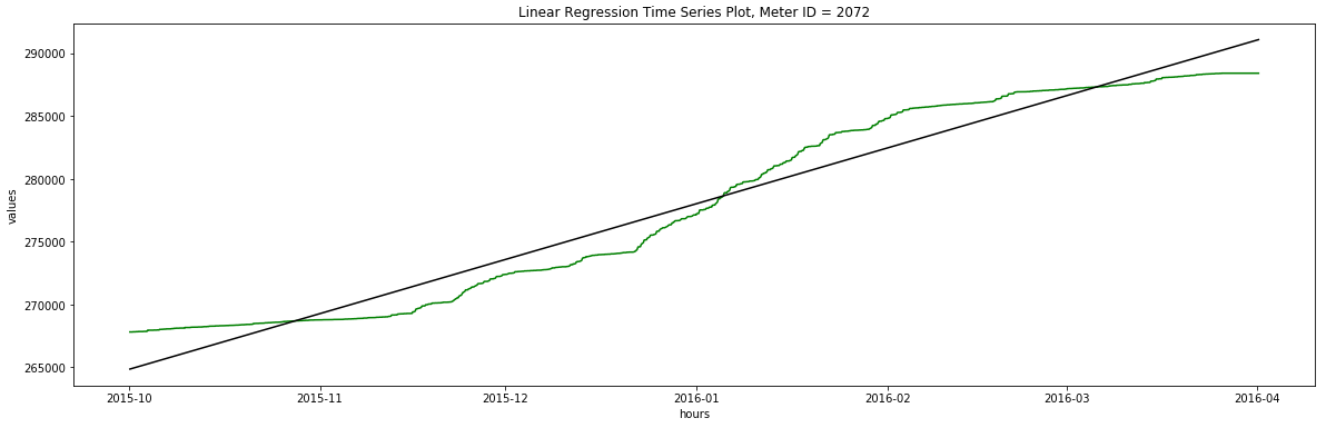

Linear Regression Time Series Plot, Meter ID = 3134
values
180000
175000
170000
165000
160000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

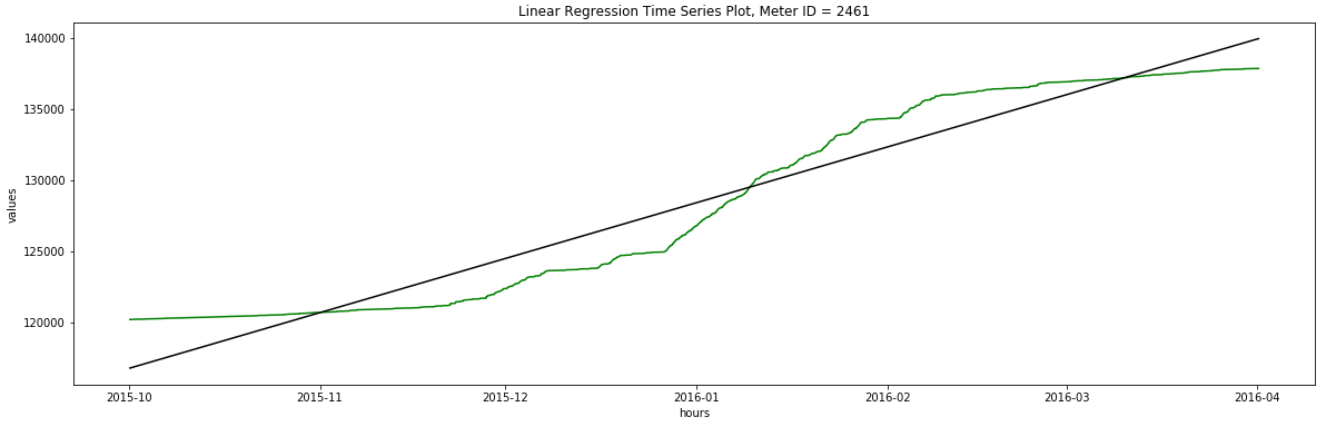

Linear Regression Time Series Plot, Meter ID = 6836
values
106000
104000
102000
100000
98000
96000
94000
92000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

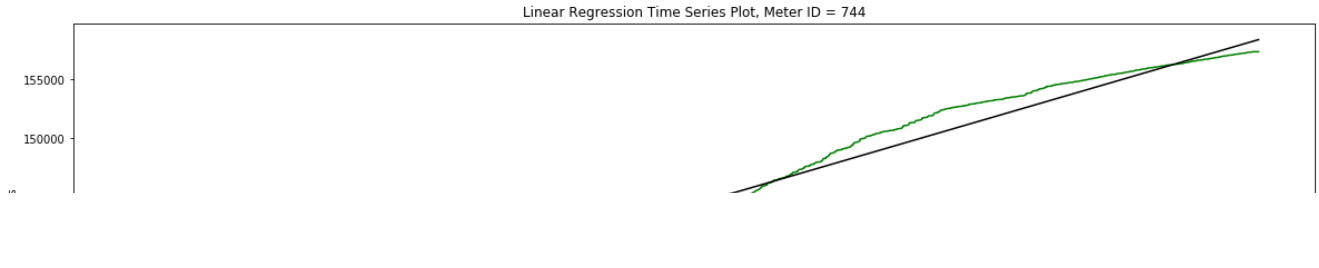

Linear Regression Time Series Plot, Meter ID = 1042
values
222500
220000
217500
215000
212500
210000
207500
205000
202500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

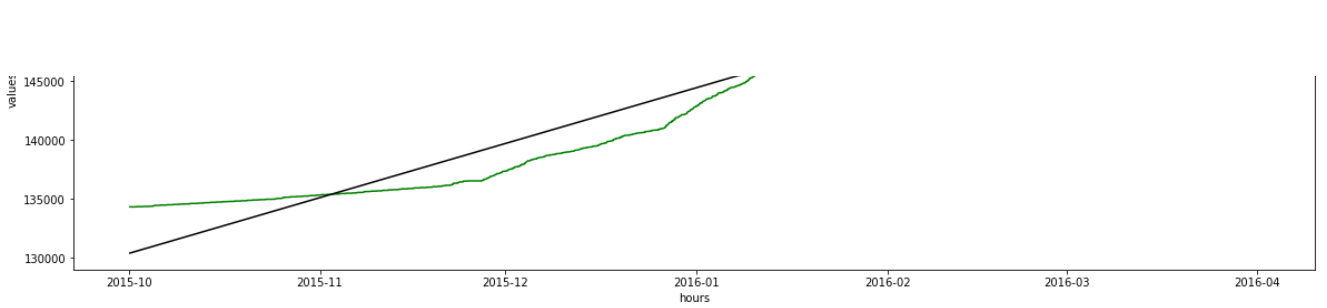

Linear Regression Time Series Plot, Meter ID = 4373
values
hours
265000
260000
255000
250000
245000
240000
235000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

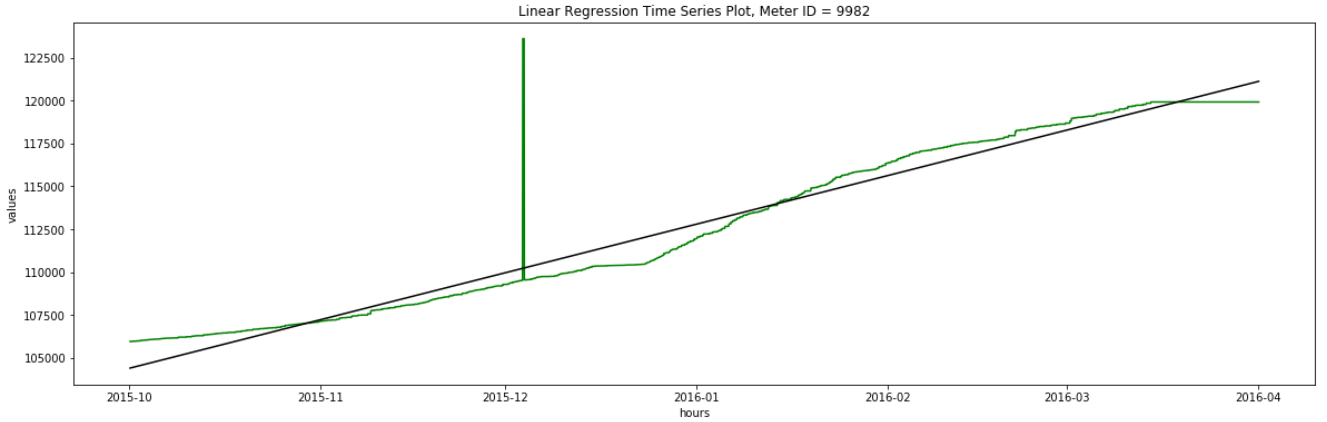

Linear Regression Time Series Plot, Meter ID = 5484
values
hours
285000
280000
275000
270000
265000
260000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

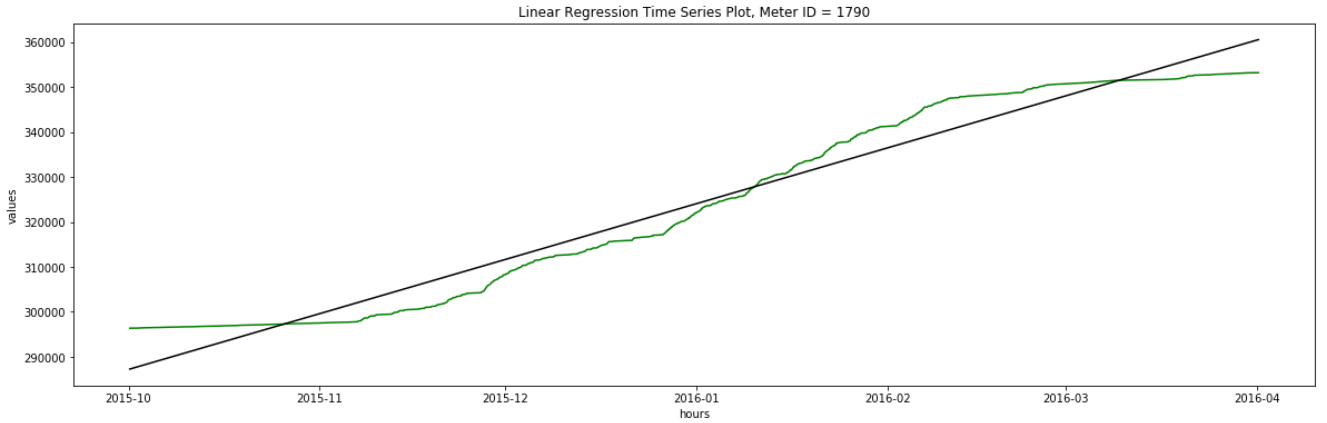

Linear Regression Time Series Plot, Meter ID = 2072
values
hours
290000
285000
280000
275000
270000
265000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

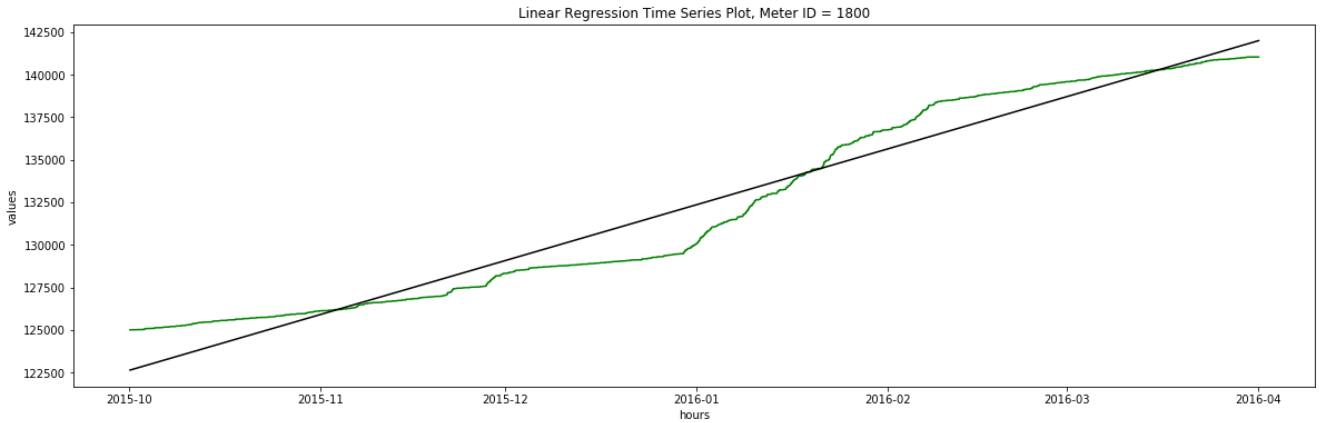

Linear Regression Time Series Plot, Meter ID = 2461
values
hours
140000
135000
130000
125000
120000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

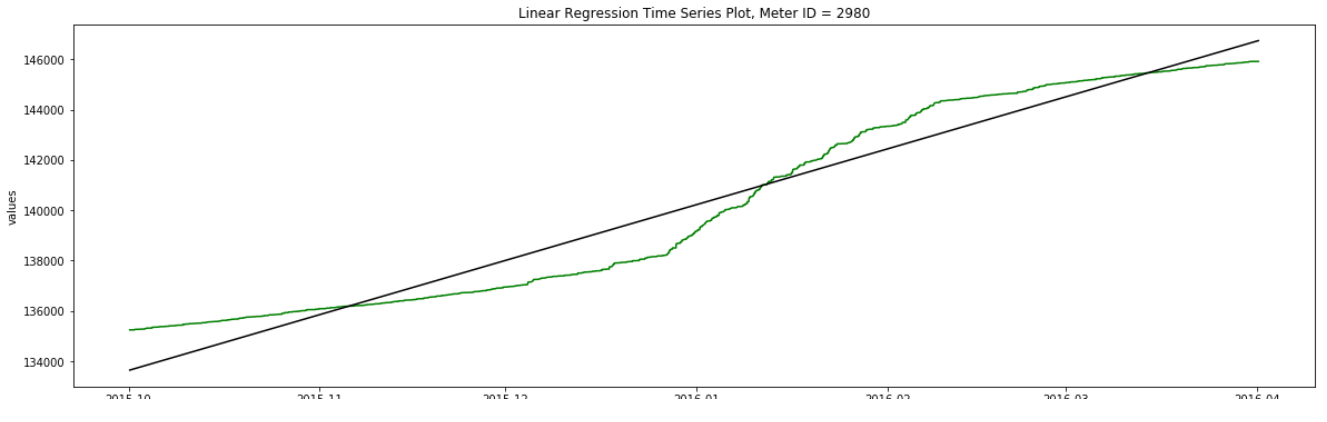

Linear Regression Time Series Plot, Meter ID = 744
155000
150000


values
145000
140000
135000
130000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

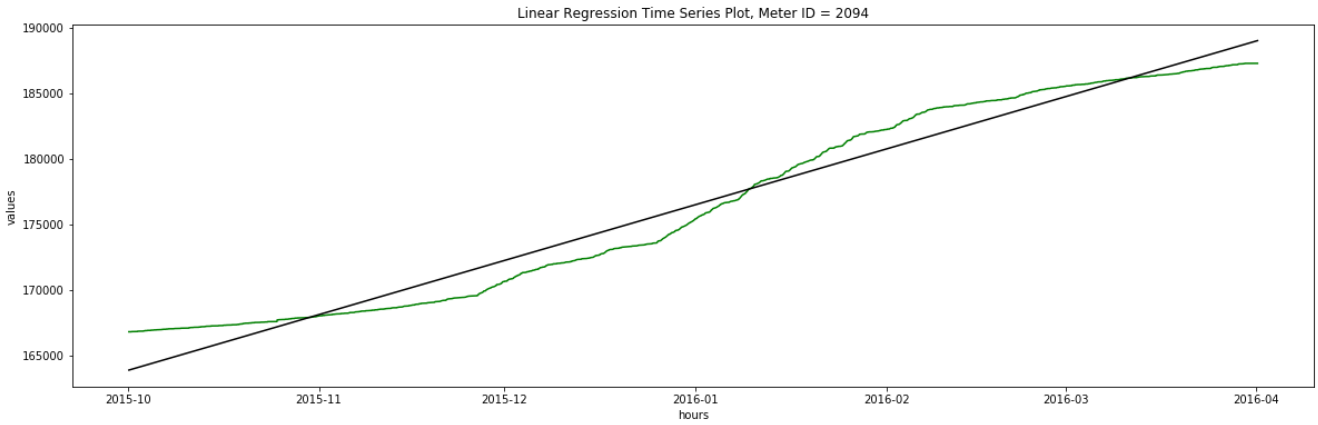

Linear Regression Time Series Plot, Meter ID = 9982
values
122500
120000
117500
115000
112500
110000
107500
105000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

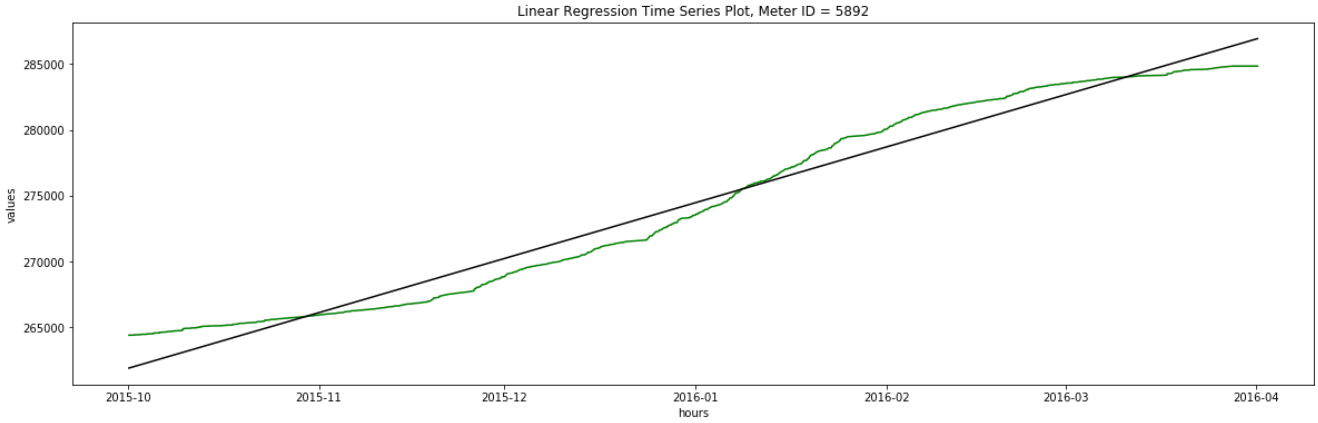

Linear Regression Time Series Plot, Meter ID = 1790
values
360000
350000
340000
330000
320000
310000
300000
290000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

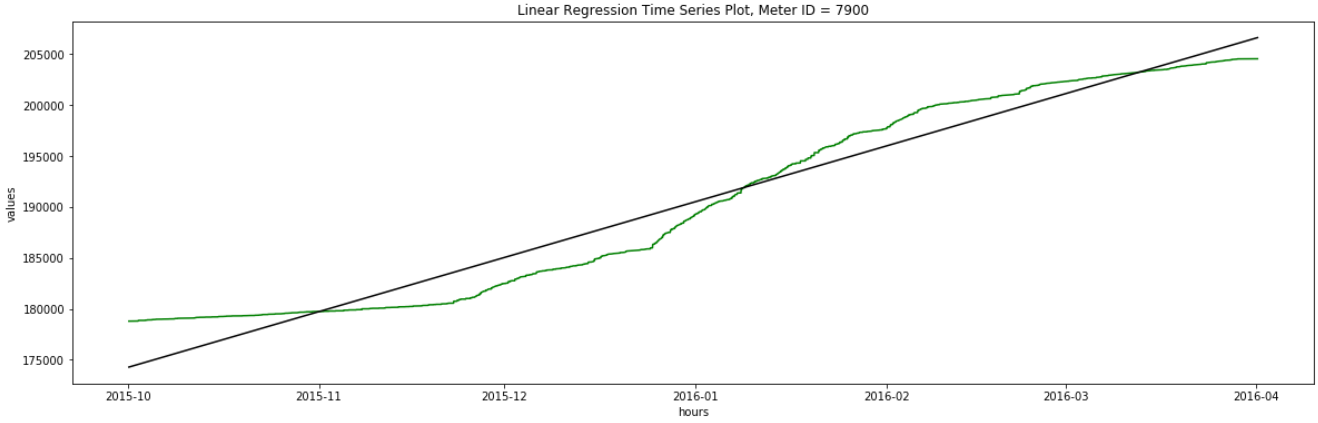

Linear Regression Time Series Plot, Meter ID = 1800
values
142500
140000
137500
135000
132500
130000
127500
125000
122500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

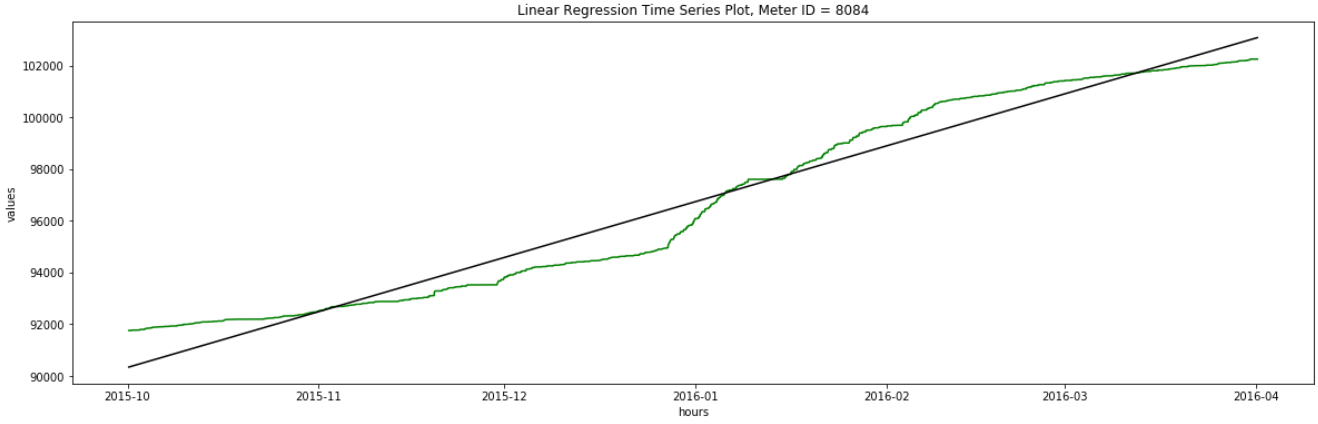

Linear Regression Time Series Plot, Meter ID = 2980
values
146000
144000
142000
140000
138000
136000
134000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

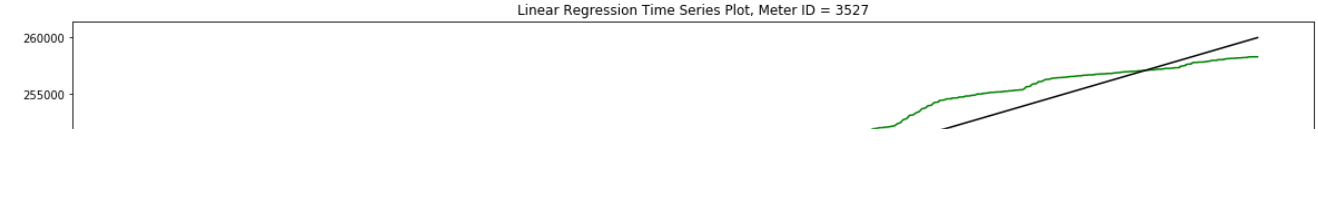

2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

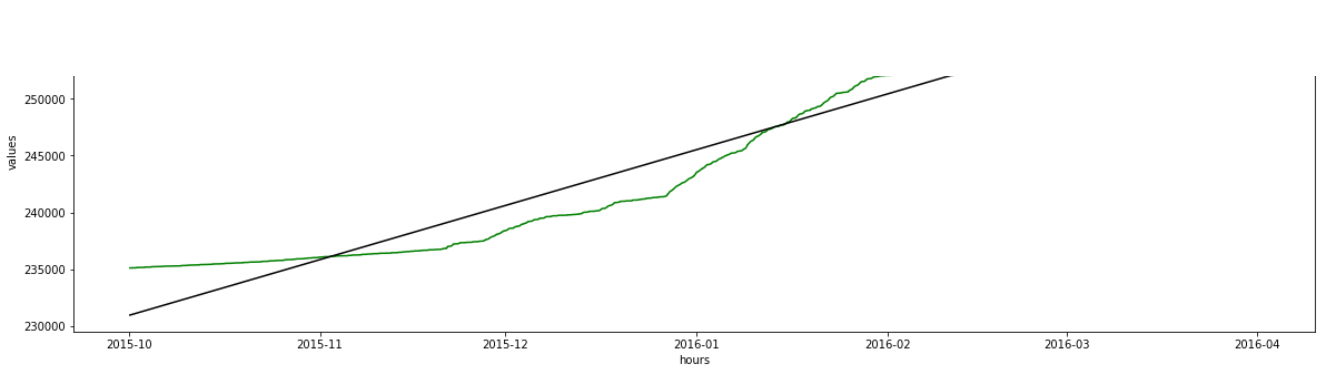

Linear Regression Time Series Plot, Meter ID = 2094
190000
185000
180000
175000
170000
165000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

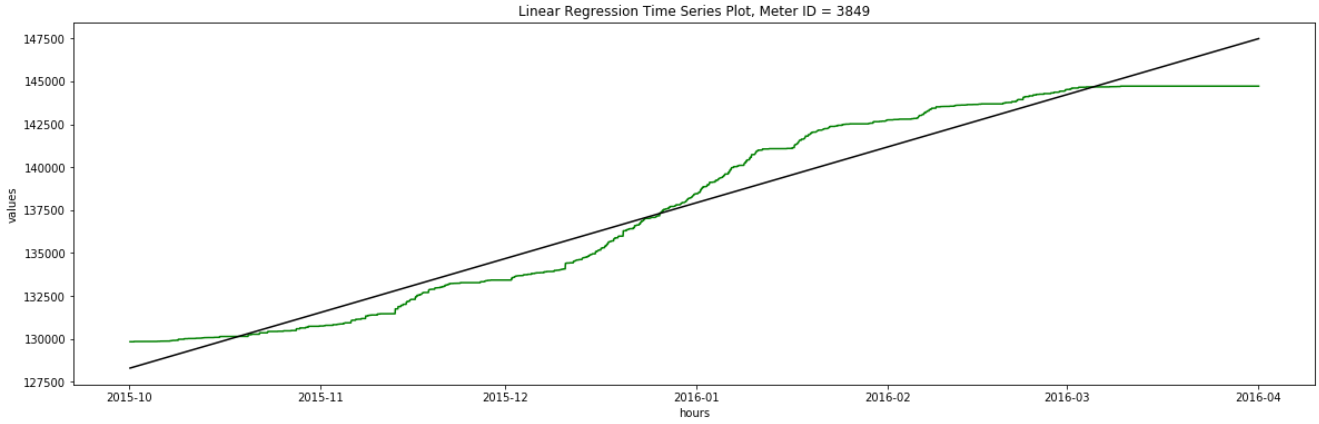

Linear Regression Time Series Plot, Meter ID = 5892
285000
280000
275000
270000
265000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

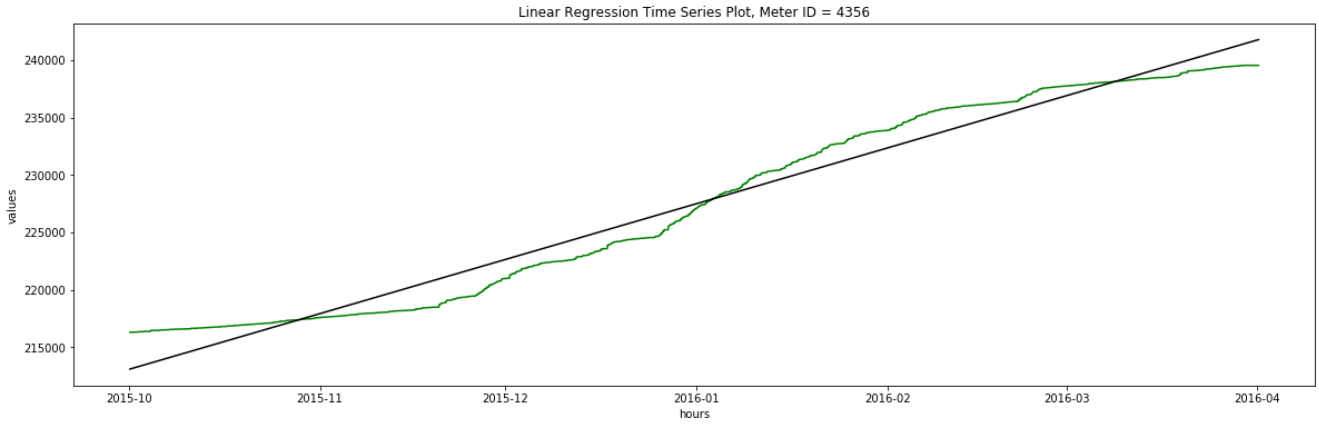

Linear Regression Time Series Plot, Meter ID = 7900
205000
200000
195000
190000
185000
180000
175000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

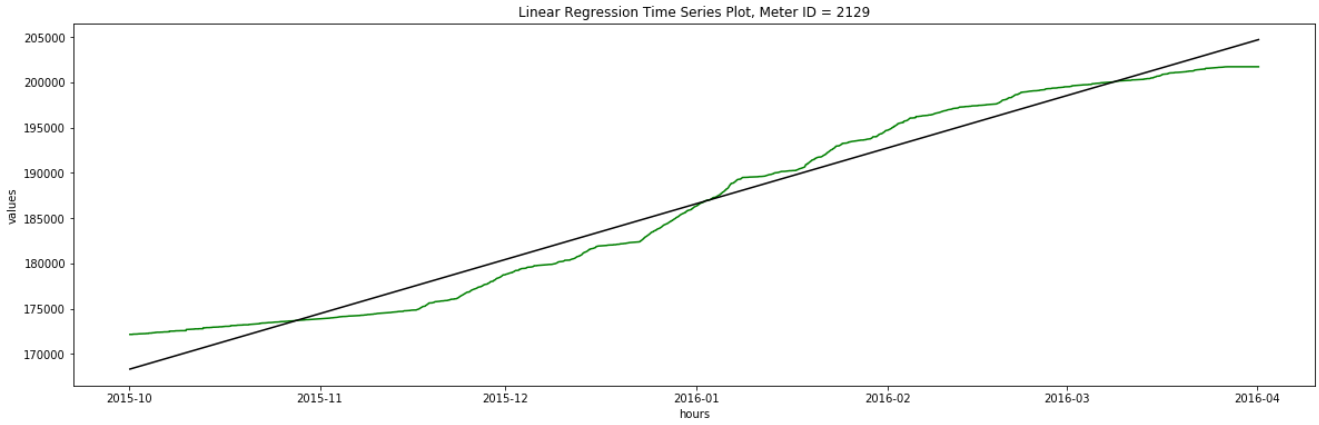

Linear Regression Time Series Plot, Meter ID = 8084
102000
100000
98000
96000
94000
92000
90000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

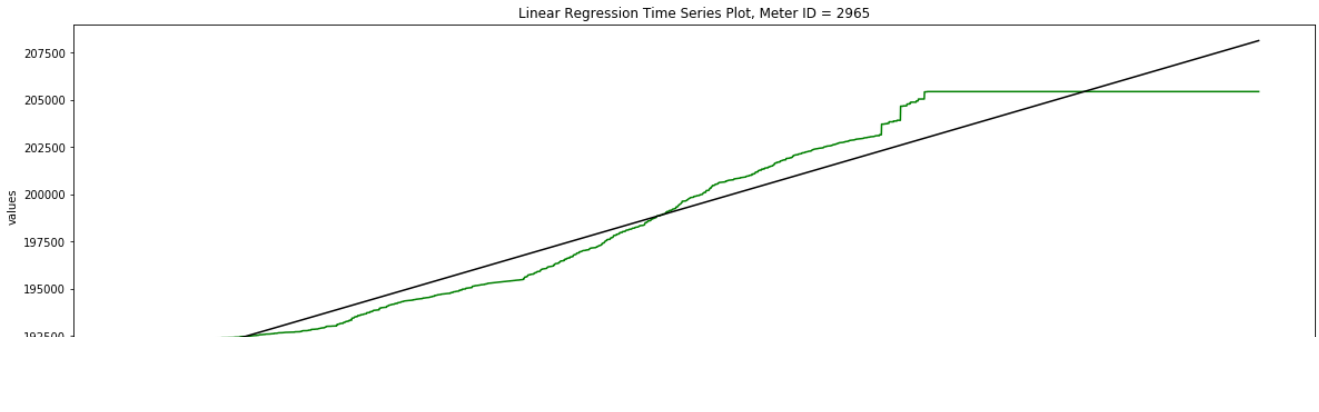

Linear Regression Time Series Plot, Meter ID = 3527
260000
255000


250000
245000
240000
235000
230000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

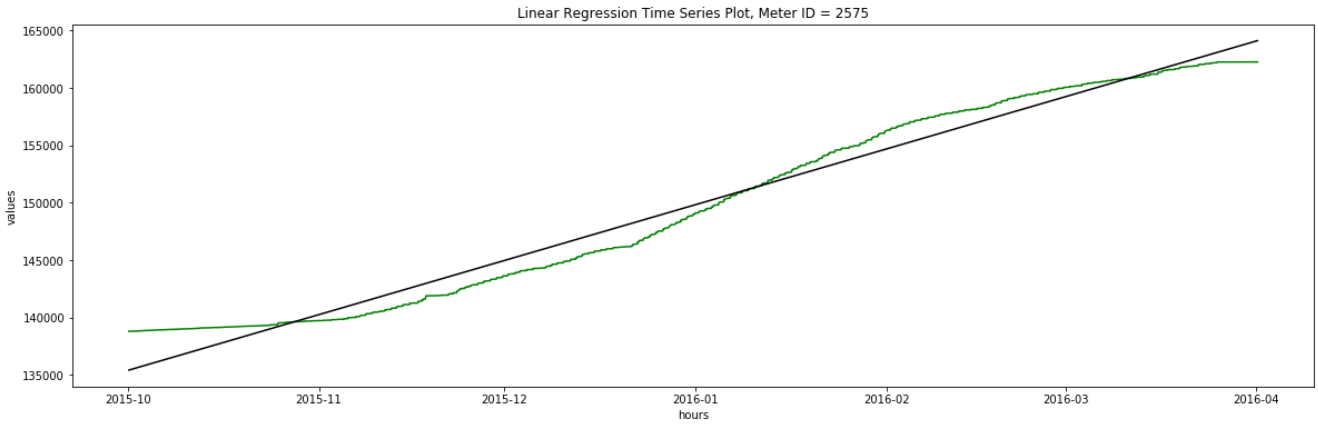

Linear Regression Time Series Plot, Meter ID = 3849
147500
145000
142500
140000
137500
135000
132500
130000
127500
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

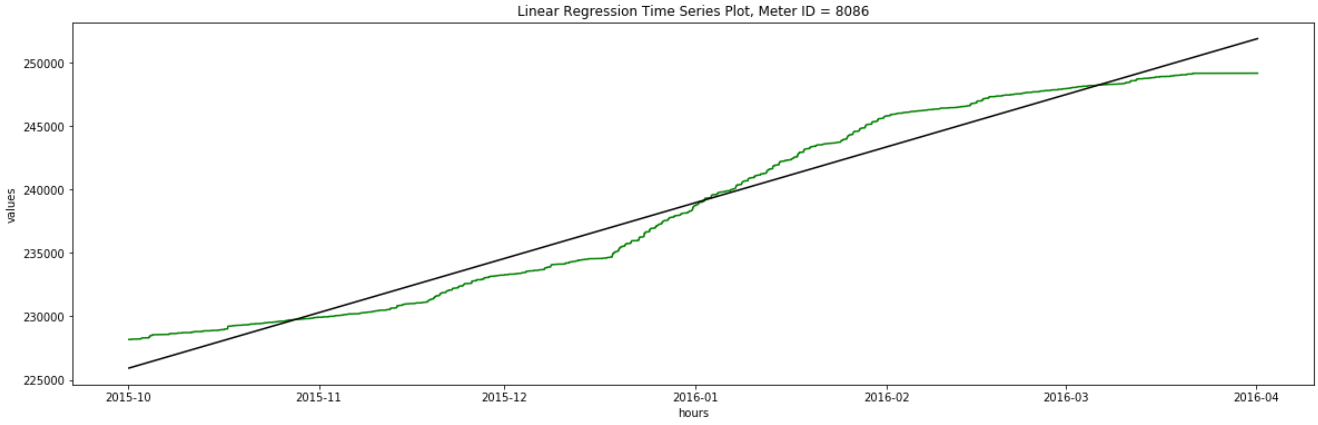

Linear Regression Time Series Plot, Meter ID = 4356
240000
235000
230000
225000
220000
215000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

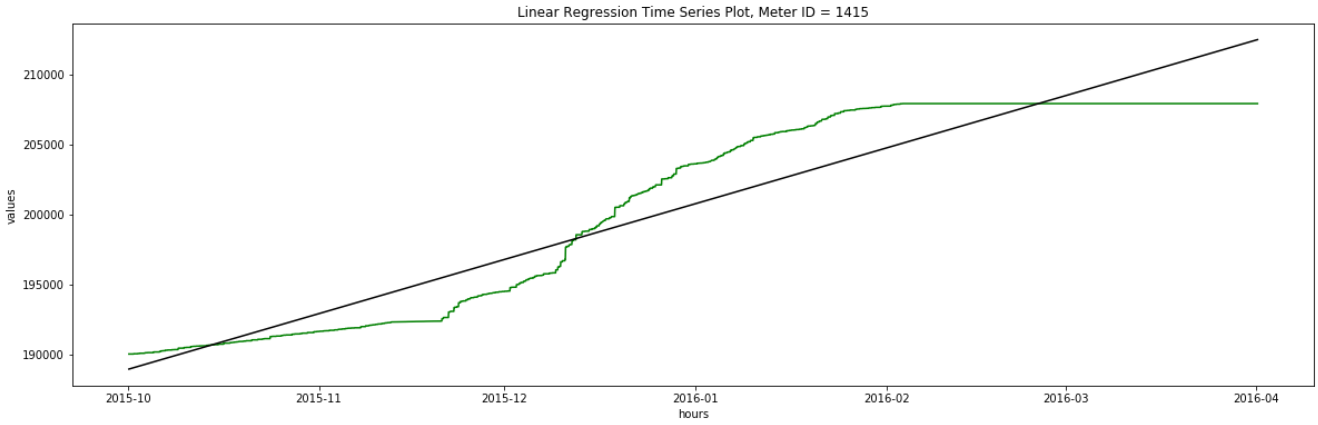

Linear Regression Time Series Plot, Meter ID = 2129
205000
200000
195000
190000
185000
180000
175000
170000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

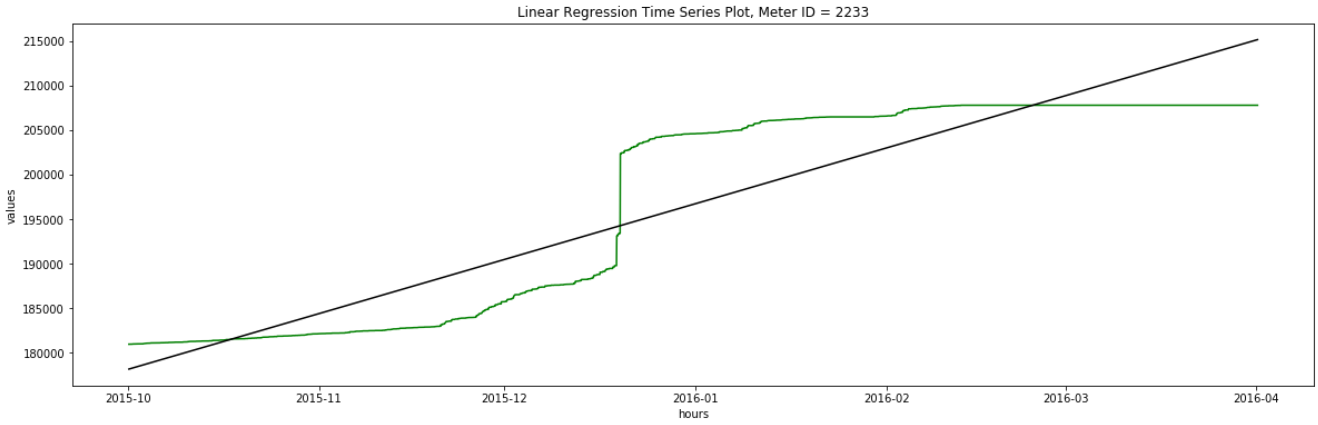

Linear Regression Time Series Plot, Meter ID = 2965
207500
205000
202500
200000
197500
195000
values

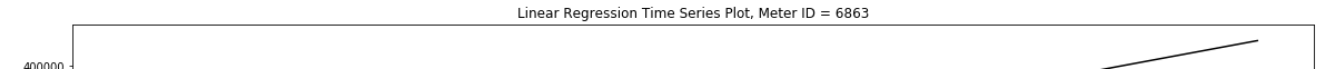

190000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

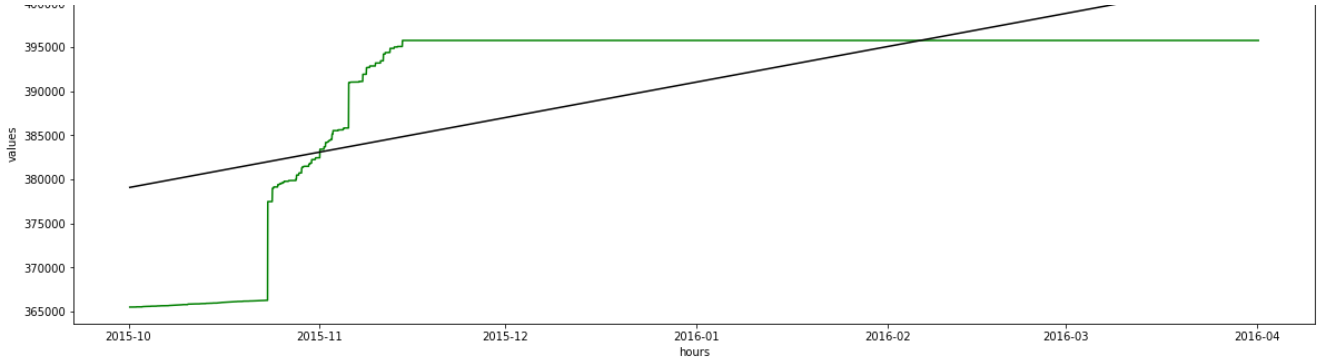

Linear Regression Time Series Plot, Meter ID = 2575
values
165000
160000
155000
150000
145000
140000
135000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

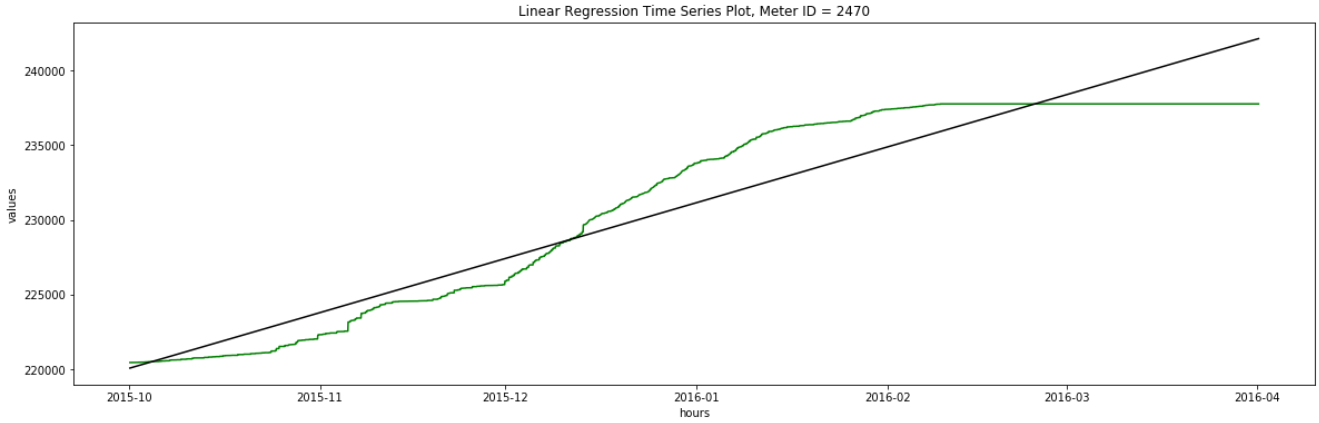

Linear Regression Time Series Plot, Meter ID = 8086
values
250000
245000
240000
235000
230000
225000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

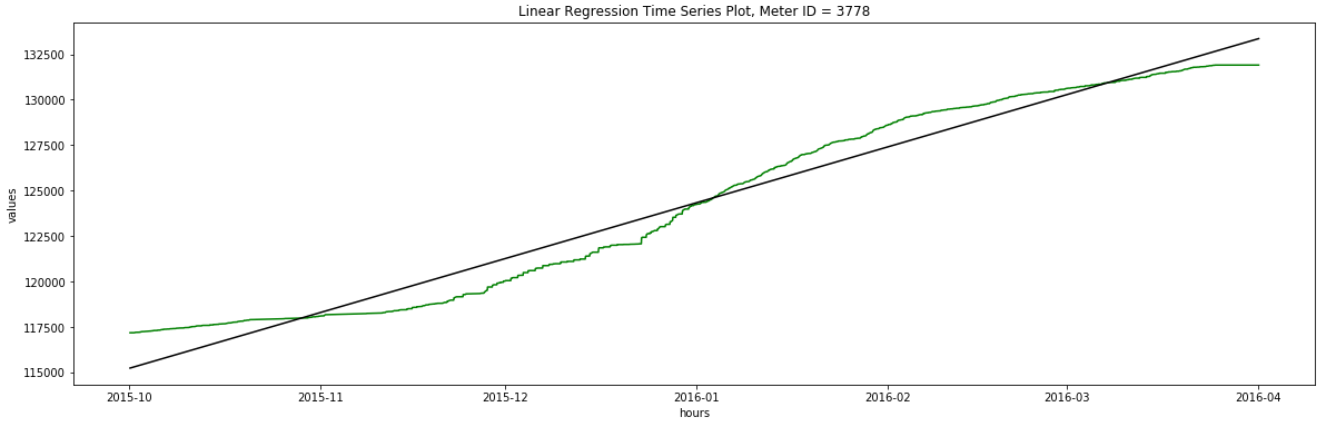

Linear Regression Time Series Plot, Meter ID = 1415
values
210000
205000
200000
195000
190000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

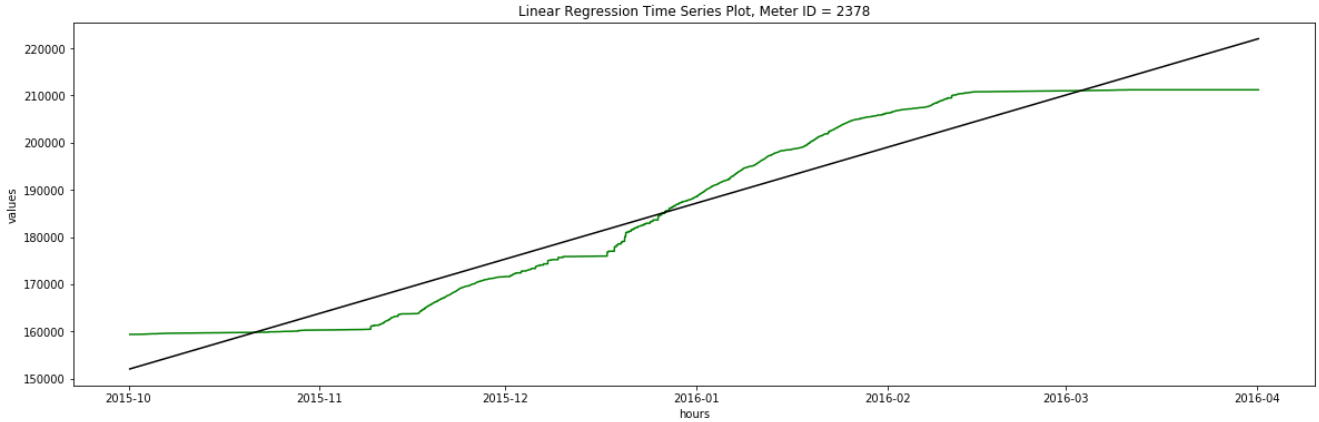

Linear Regression Time Series Plot, Meter ID = 2233
values
215000
210000
205000
200000
195000
190000
185000
180000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

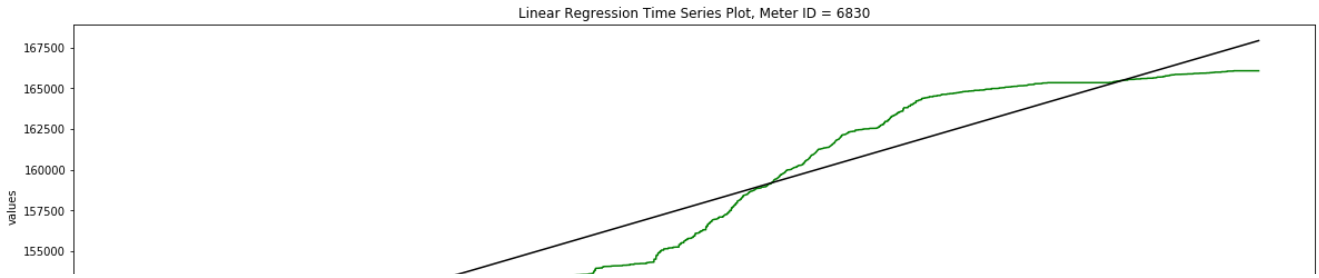

Linear Regression Time Series Plot, Meter ID = 6863

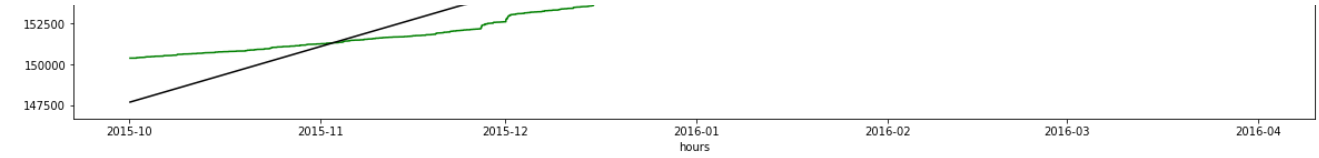

values
395000
390000
385000
380000
375000
370000
365000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

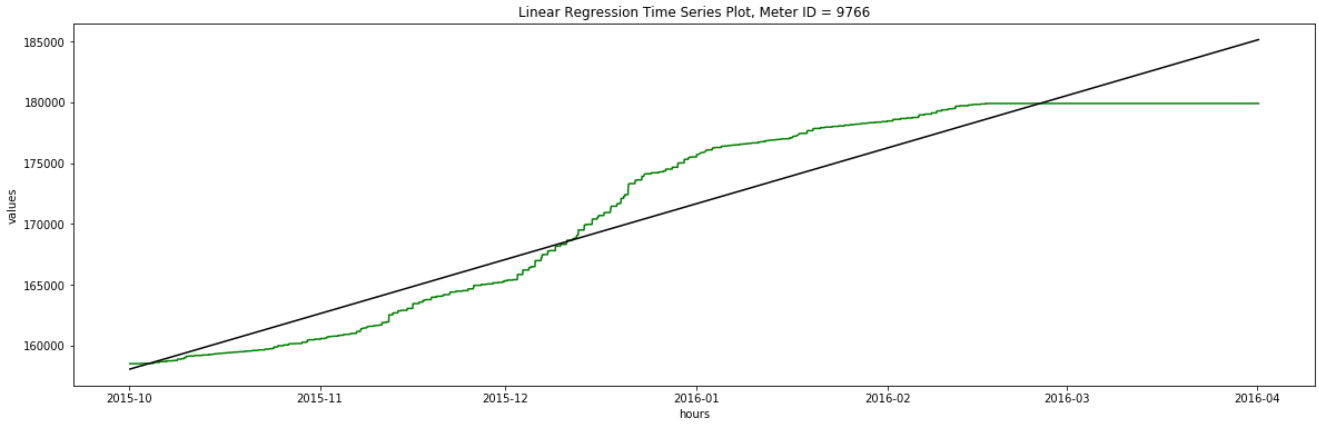

Linear Regression Time Series Plot, Meter ID = 2470
values
240000
235000
230000
225000
220000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

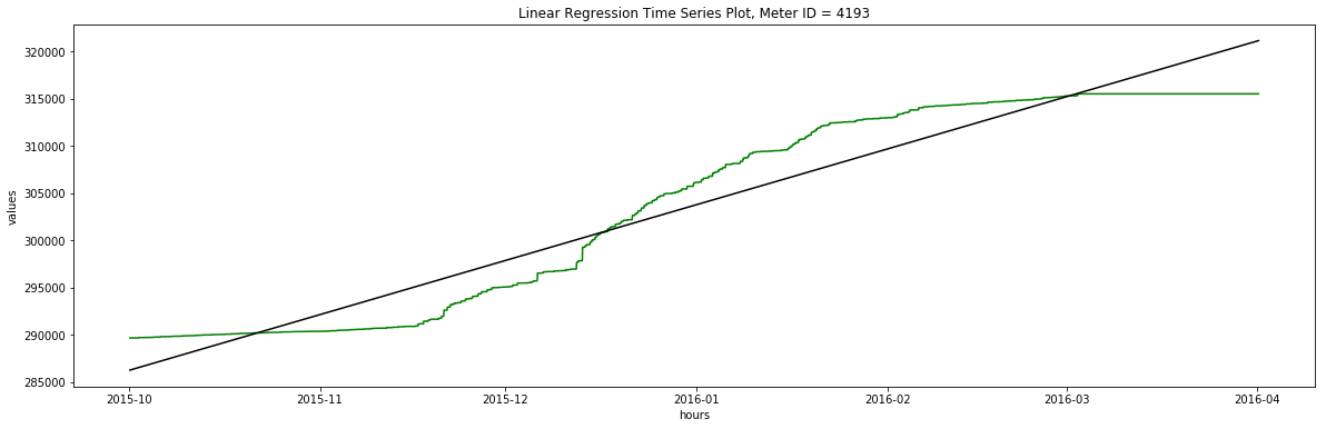

Linear Regression Time Series Plot, Meter ID = 3778
values
132500
130000
127500
125000
122500
120000
117500
115000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

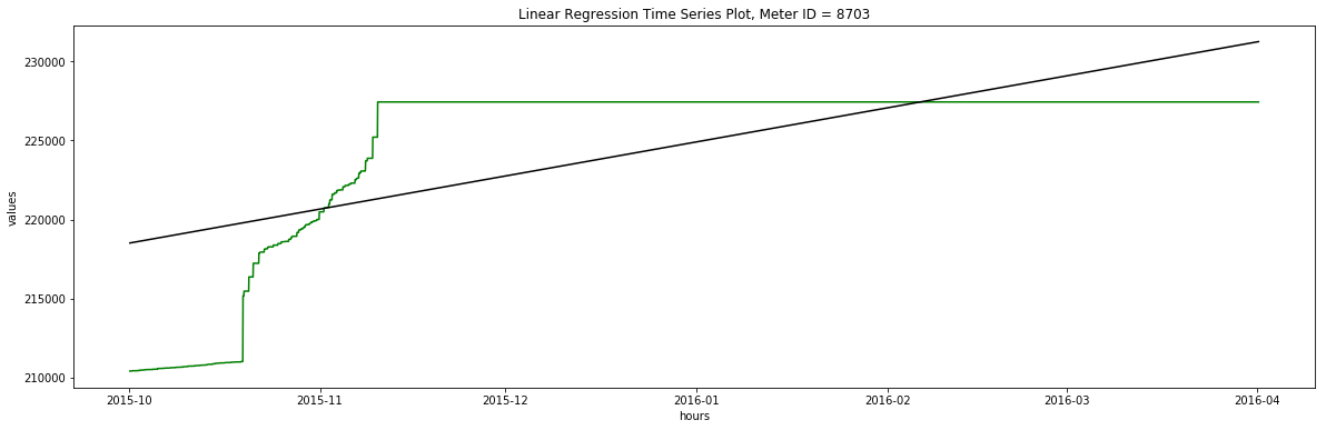

Linear Regression Time Series Plot, Meter ID = 2378
values
220000
210000
200000
190000
180000
170000
160000
150000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

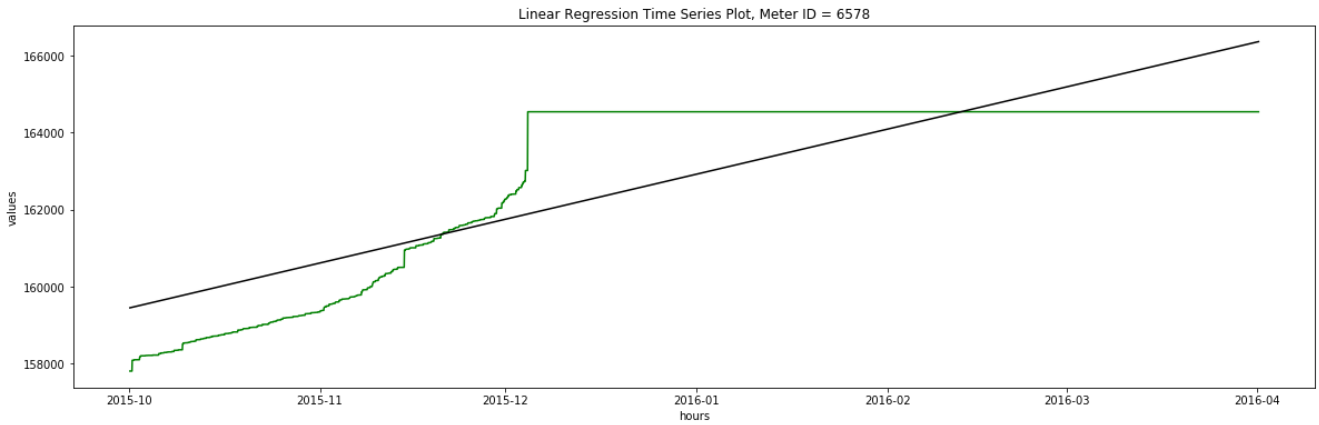

Linear Regression Time Series Plot, Meter ID = 6830
values
167500
165000
162500
160000
157500
155000

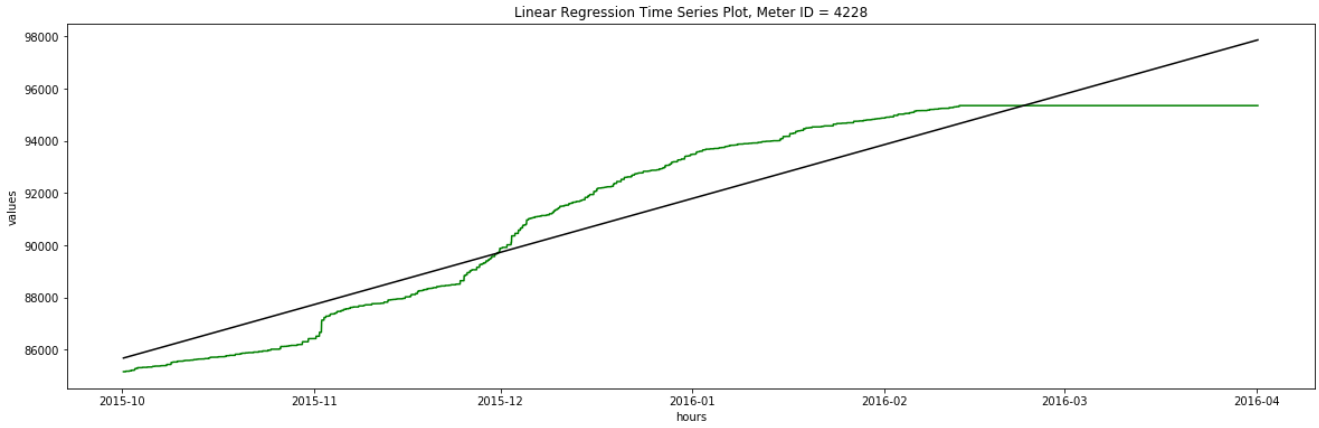

152500
150000
147500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

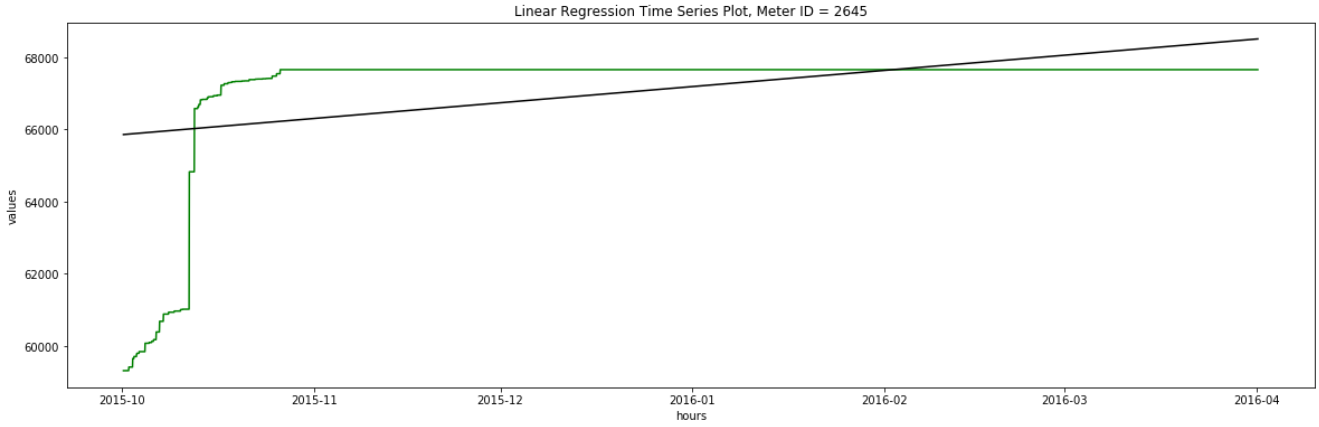

Linear Regression Time Series Plot, Meter ID = 9766
185000
180000
175000
170000
165000
160000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

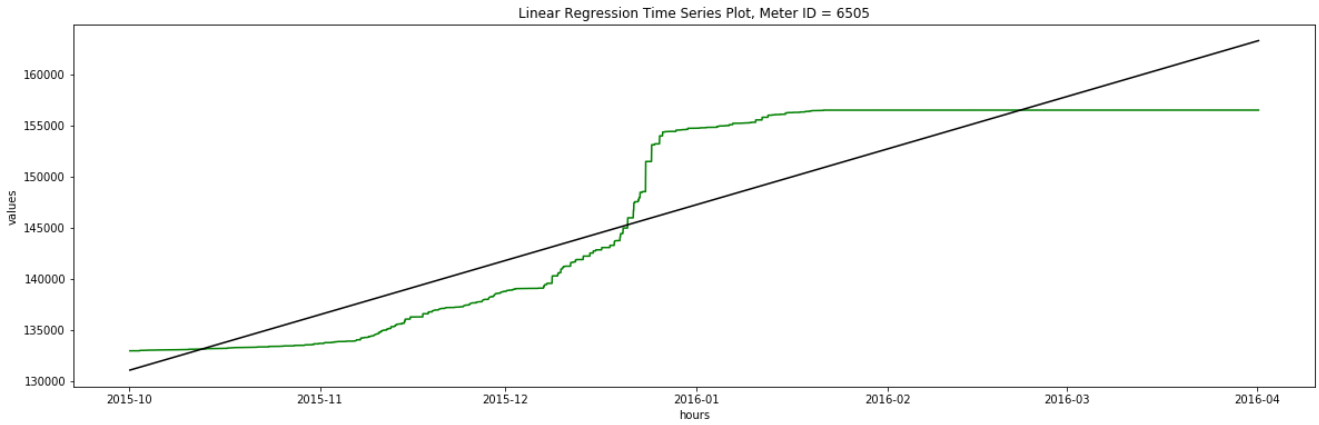

Linear Regression Time Series Plot, Meter ID = 4193
320000
315000
310000
305000
300000
295000
290000
285000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

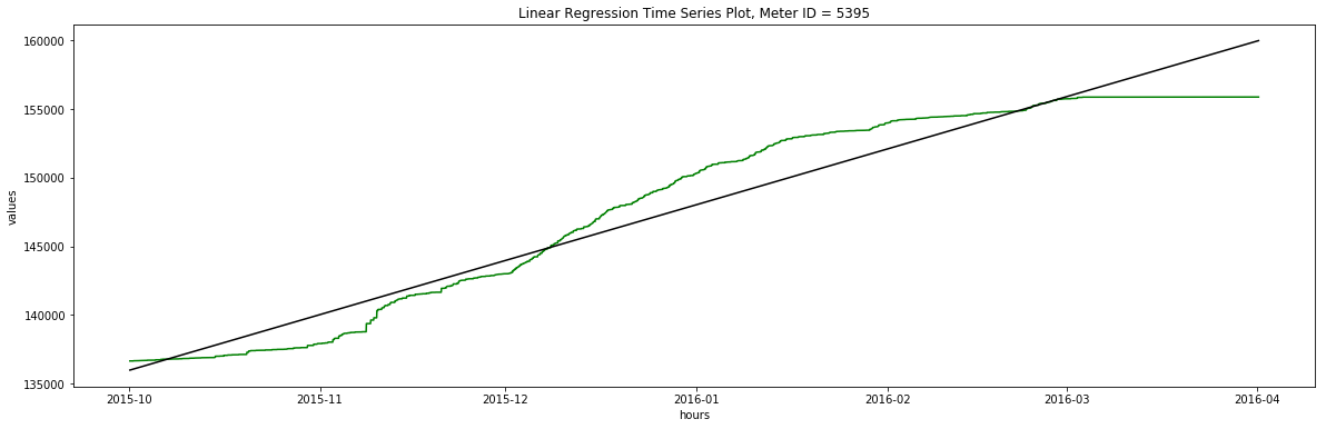

Linear Regression Time Series Plot, Meter ID = 8703
230000
225000
220000
215000
210000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

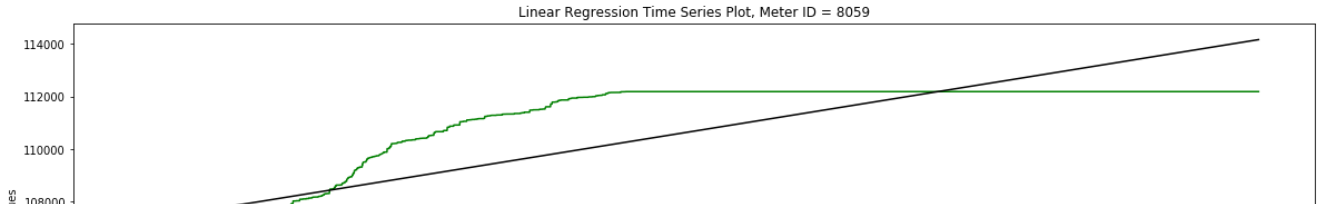

Linear Regression Time Series Plot, Meter ID = 6578
166000
164000
162000
160000
158000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

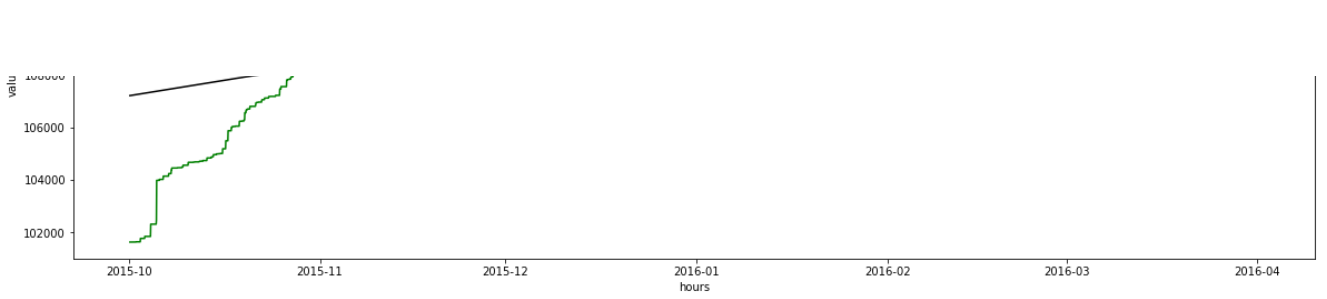

Linear Regression Time Series Plot, Meter ID = 4228
values
hours
98000
96000
94000
92000
90000
88000
86000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

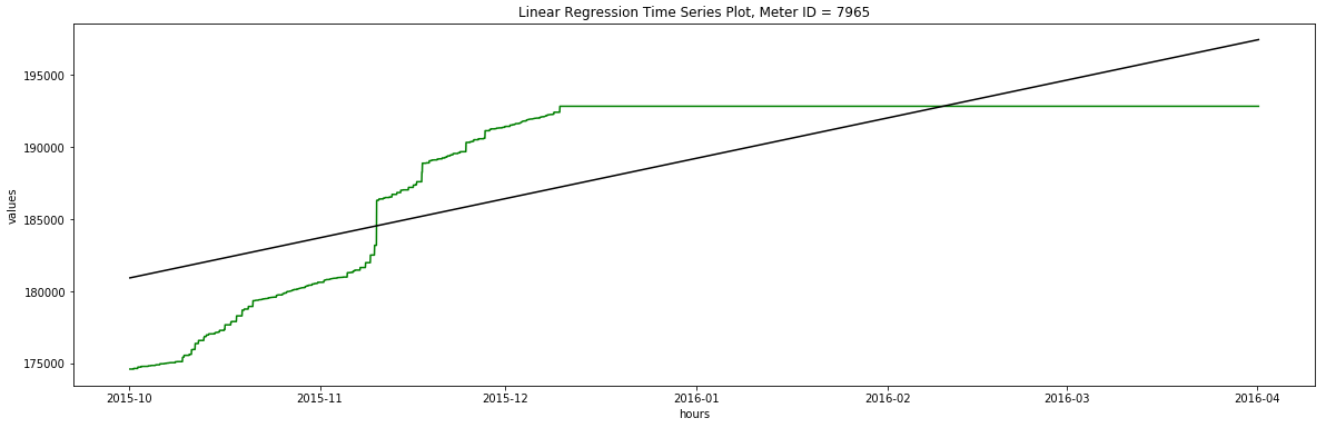

Linear Regression Time Series Plot, Meter ID = 2645
values
hours
68000
66000
64000
62000
60000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

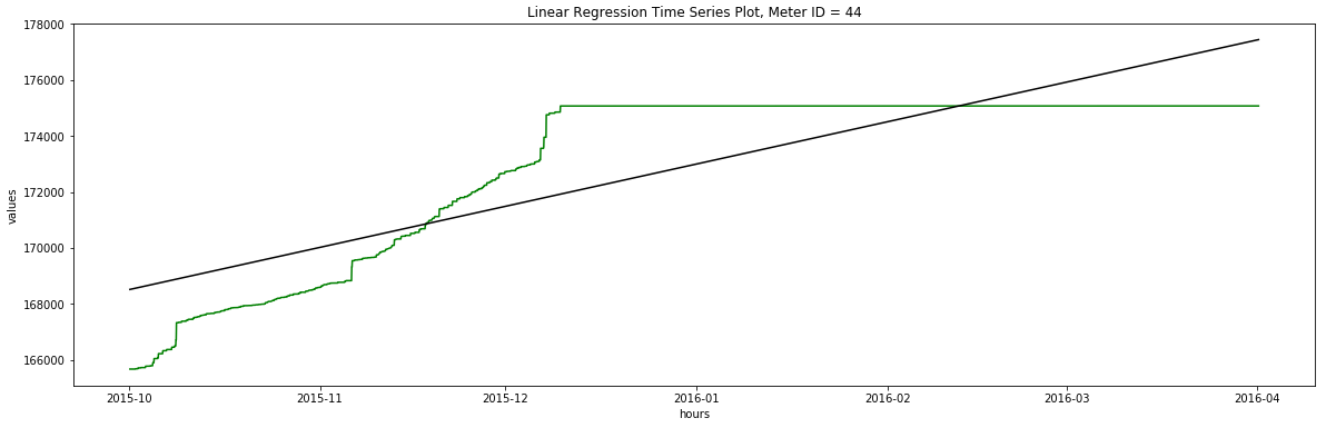

Linear Regression Time Series Plot, Meter ID = 6505
values
hours
160000
155000
150000
145000
140000
135000
130000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

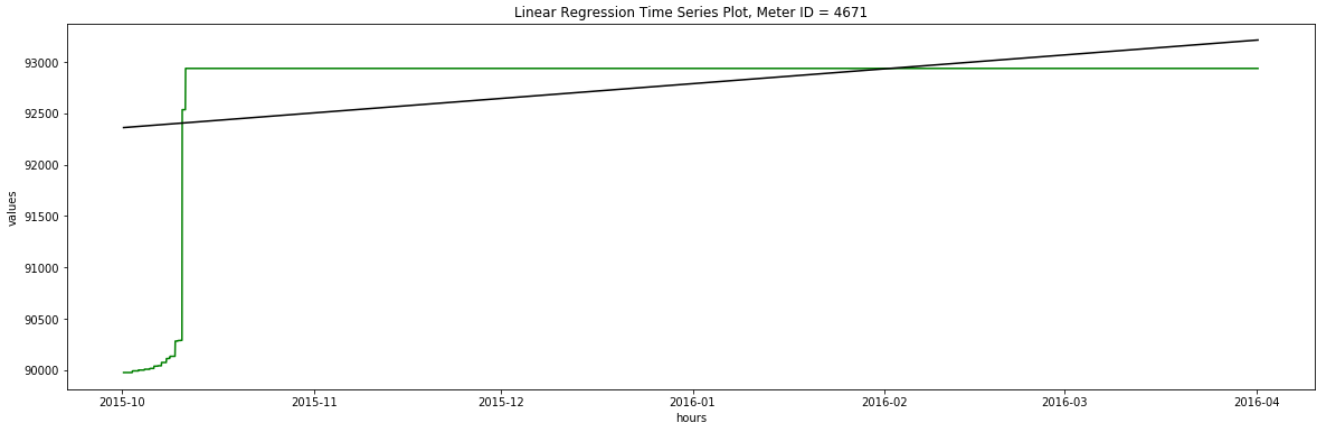

Linear Regression Time Series Plot, Meter ID = 5395
values
hours
160000
155000
150000
145000
140000
135000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

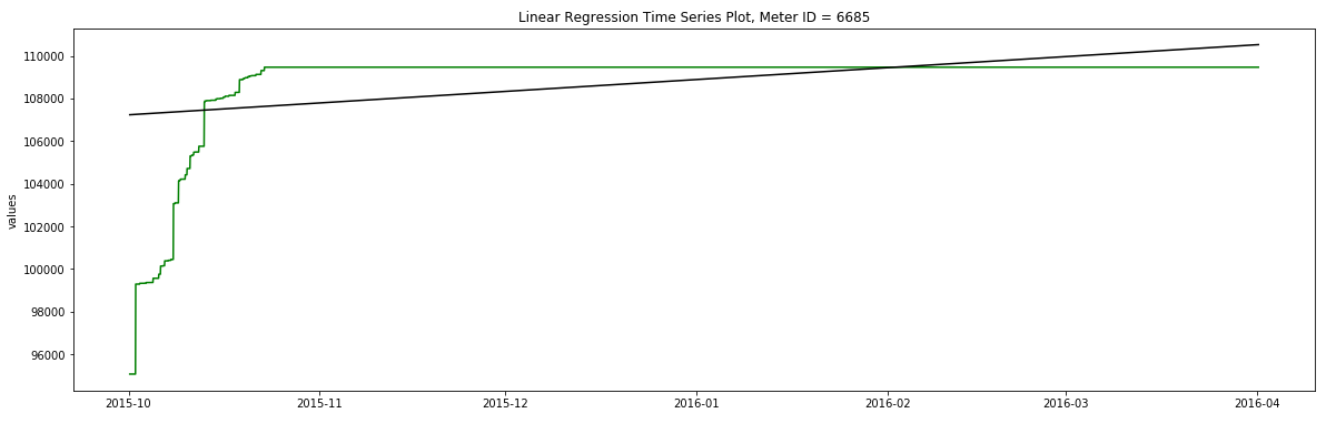

Linear Regression Time Series Plot, Meter ID = 8059
114000
112000
110000


106000
104000
102000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

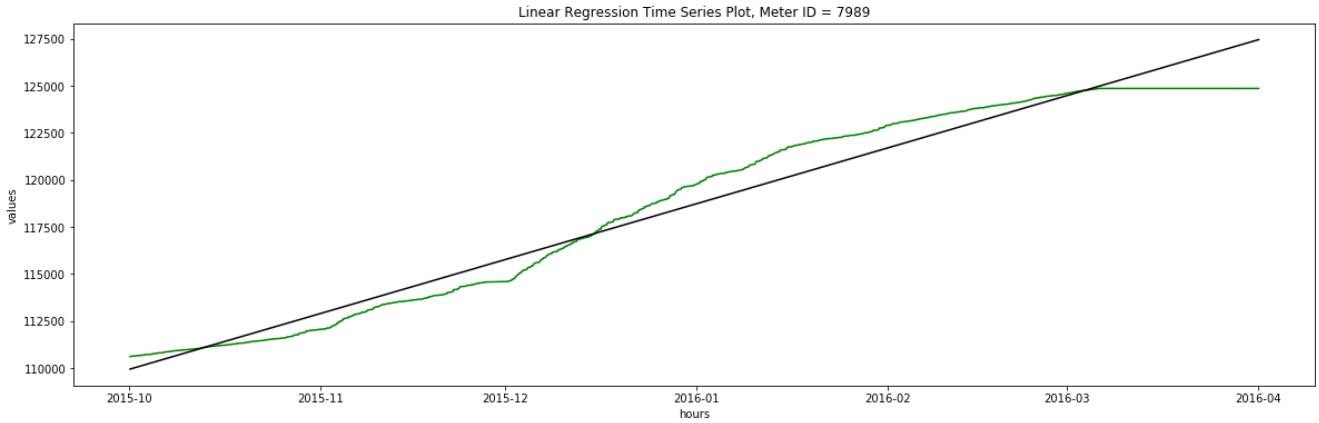

Linear Regression Time Series Plot, Meter ID = 7965
values
195000
190000
185000
180000
175000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

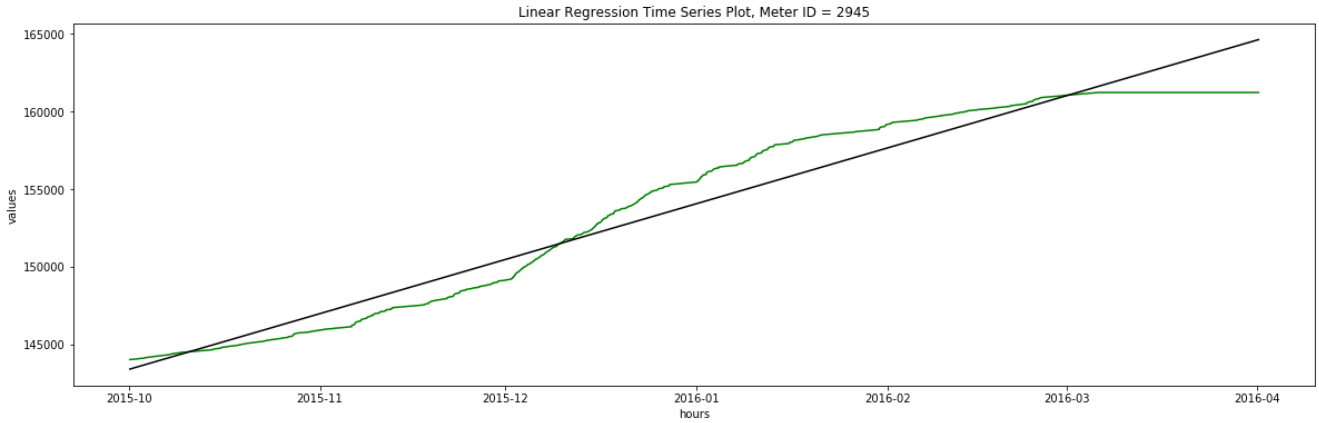

Linear Regression Time Series Plot, Meter ID = 44
values
178000
176000
174000
172000
170000
168000
166000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

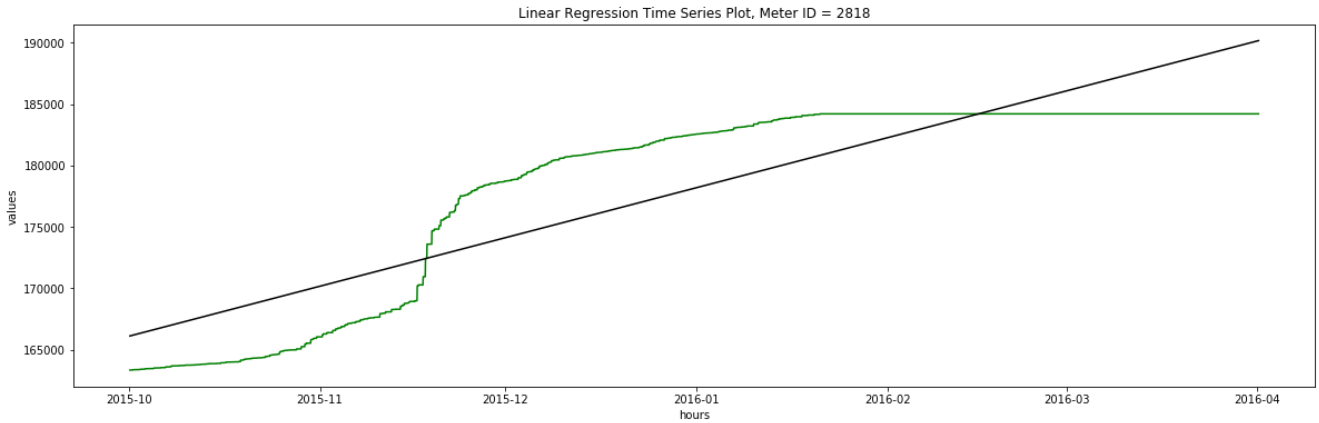

Linear Regression Time Series Plot, Meter ID = 4671
values
93000
92500
92000
91500
91000
90500
90000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

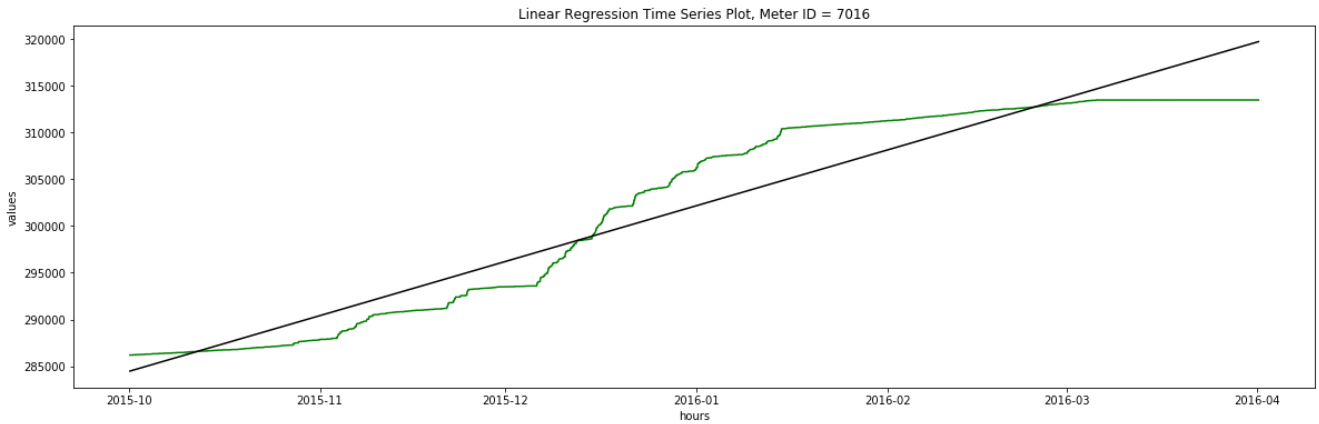

Linear Regression Time Series Plot, Meter ID = 6685
values
110000
108000
106000
104000
102000
100000
98000
96000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

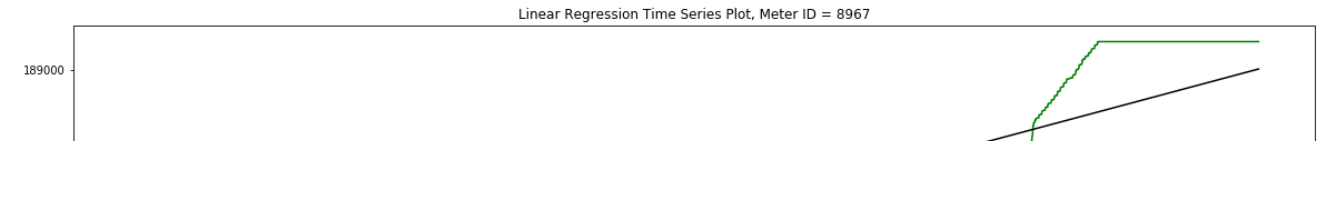
hours

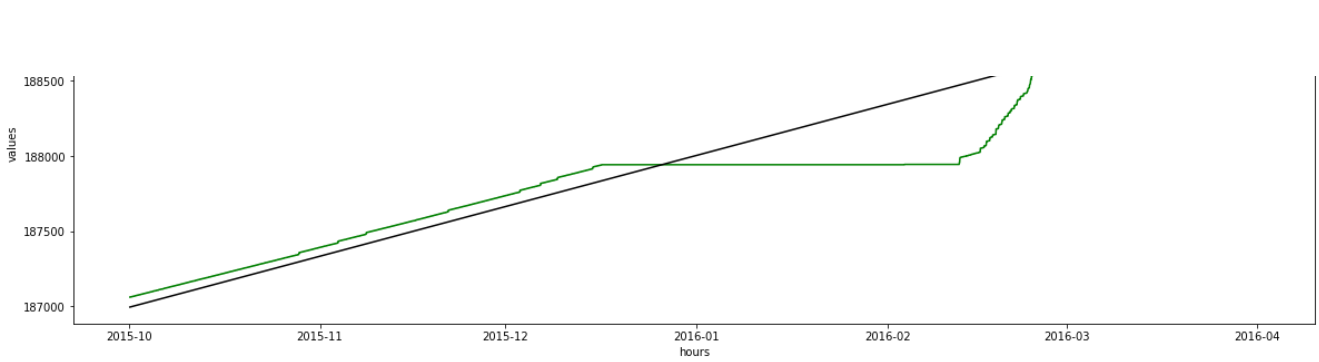

Linear Regression Time Series Plot, Meter ID = 7989
values
hours

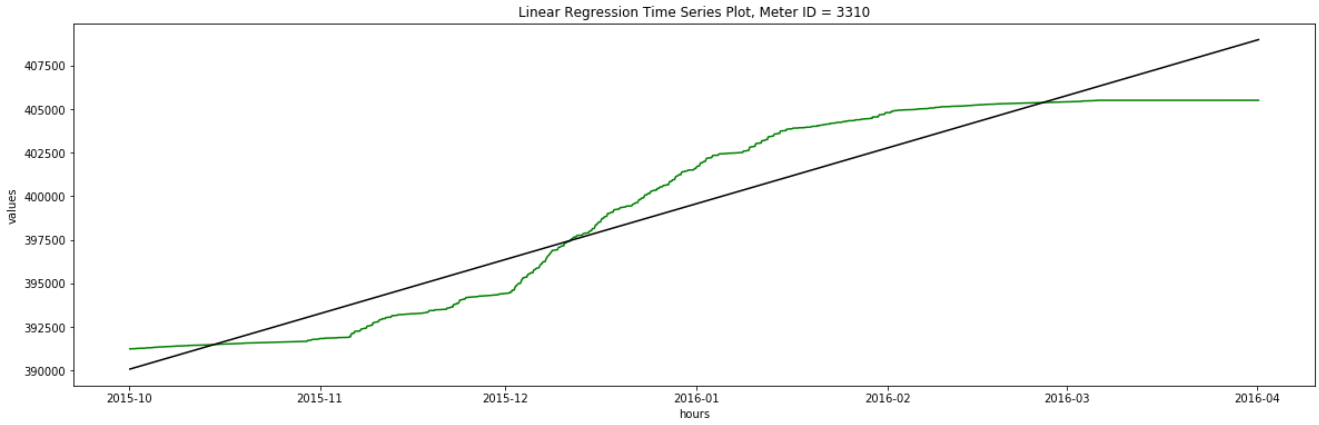

Linear Regression Time Series Plot, Meter ID = 2945
values
hours

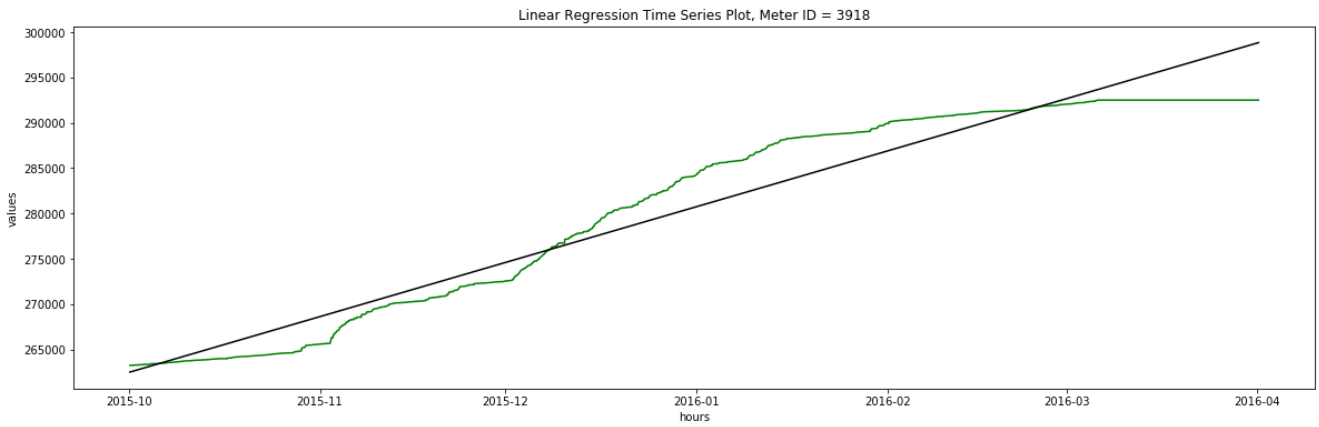

Linear Regression Time Series Plot, Meter ID = 2818
values
hours

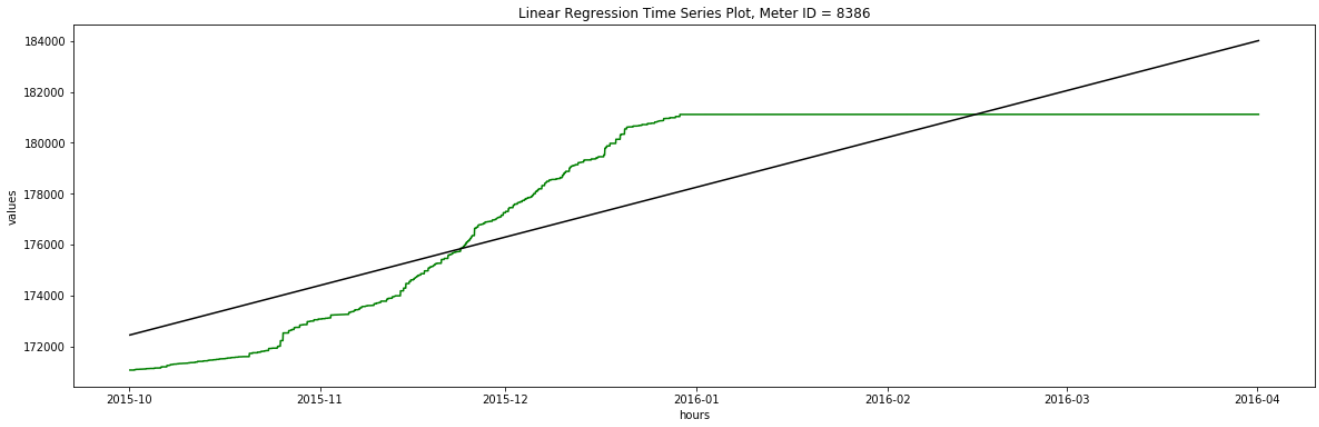

Linear Regression Time Series Plot, Meter ID = 7016
values
hours

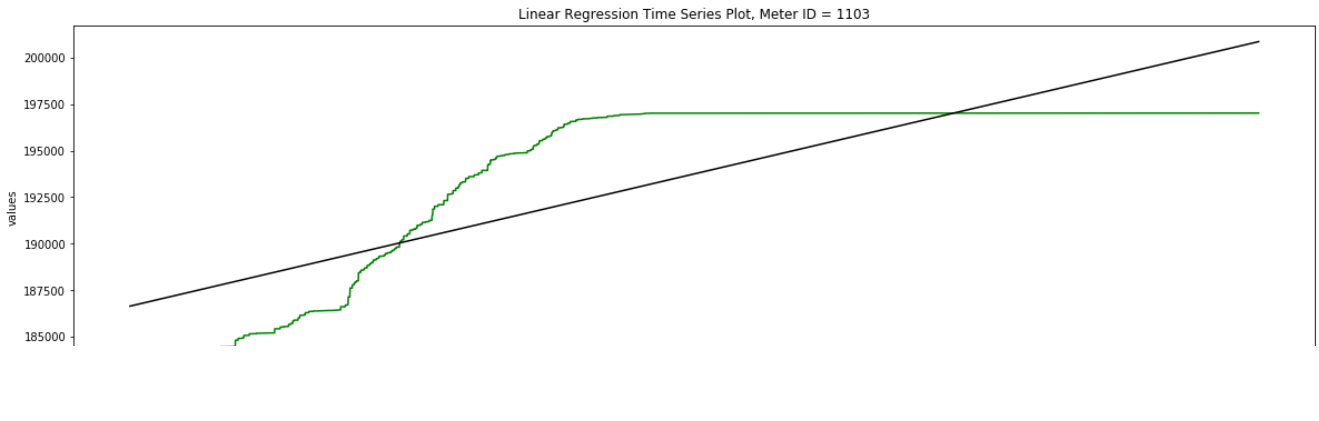

Linear Regression Time Series Plot, Meter ID = 8967

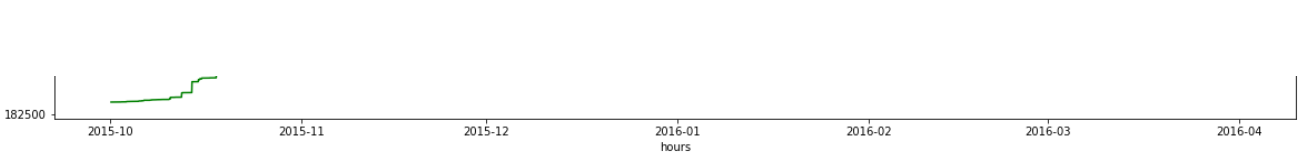

188500
188000
187500
187000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

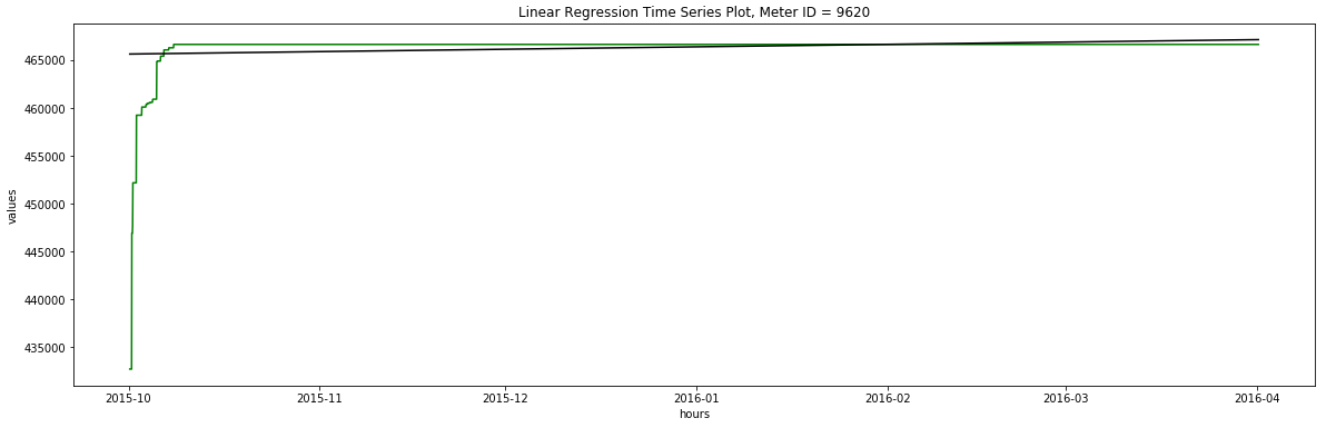

Linear Regression Time Series Plot, Meter ID = 3310
407500
405000
402500
400000
397500
395000
392500
390000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

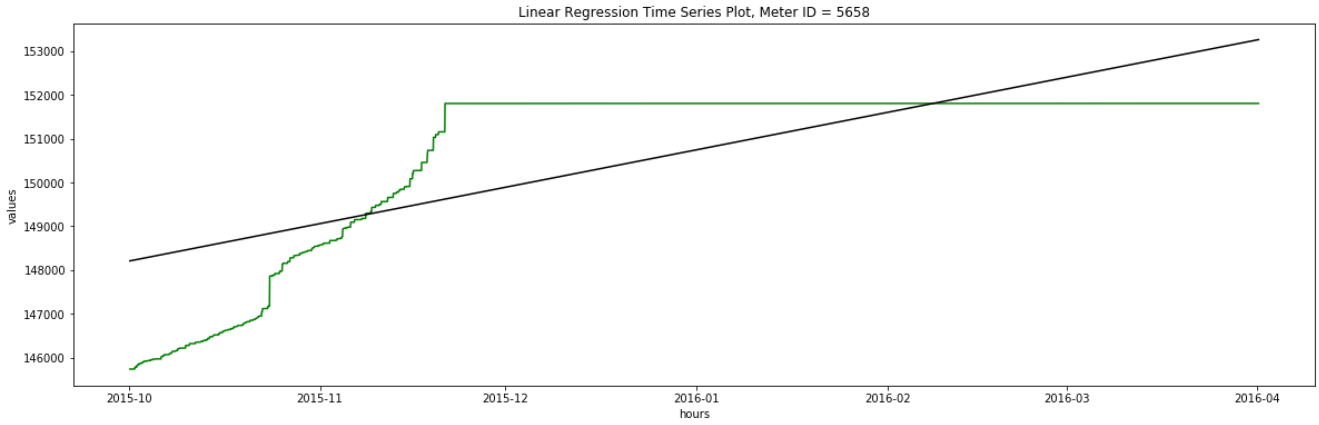

Linear Regression Time Series Plot, Meter ID = 3918
300000
295000
290000
285000
280000
275000
270000
265000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

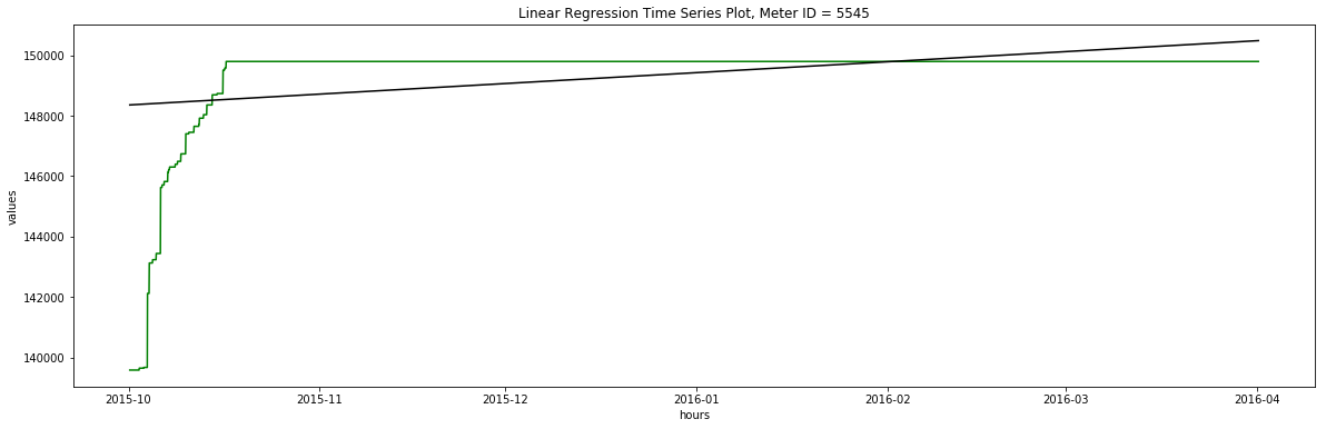

Linear Regression Time Series Plot, Meter ID = 8386
184000
182000
180000
178000
176000
174000
172000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

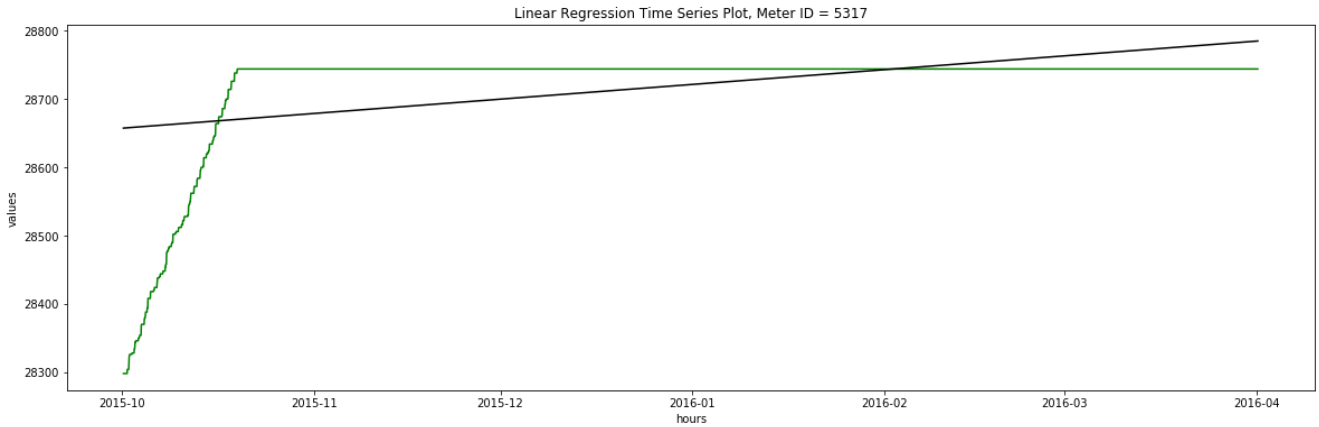

Linear Regression Time Series Plot, Meter ID = 1103
200000
197500
195000
192500
190000
187500
185000
values

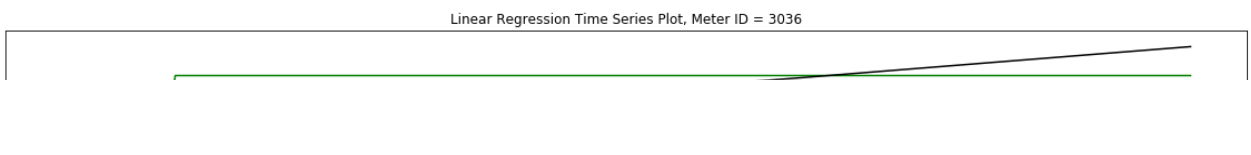

182500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

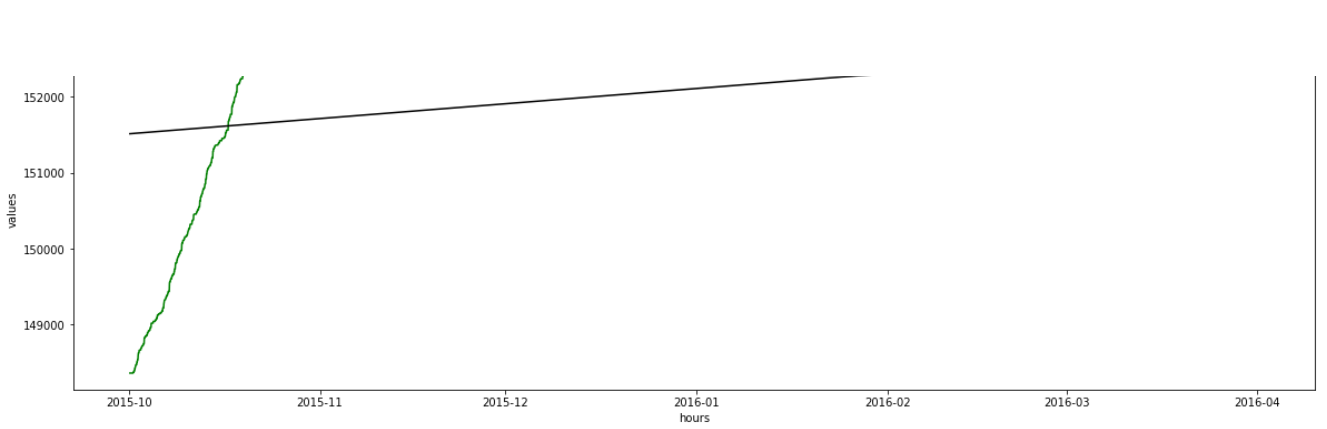

Linear Regression Time Series Plot, Meter ID = 9620
values
465000
460000
455000
450000
445000
440000
435000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

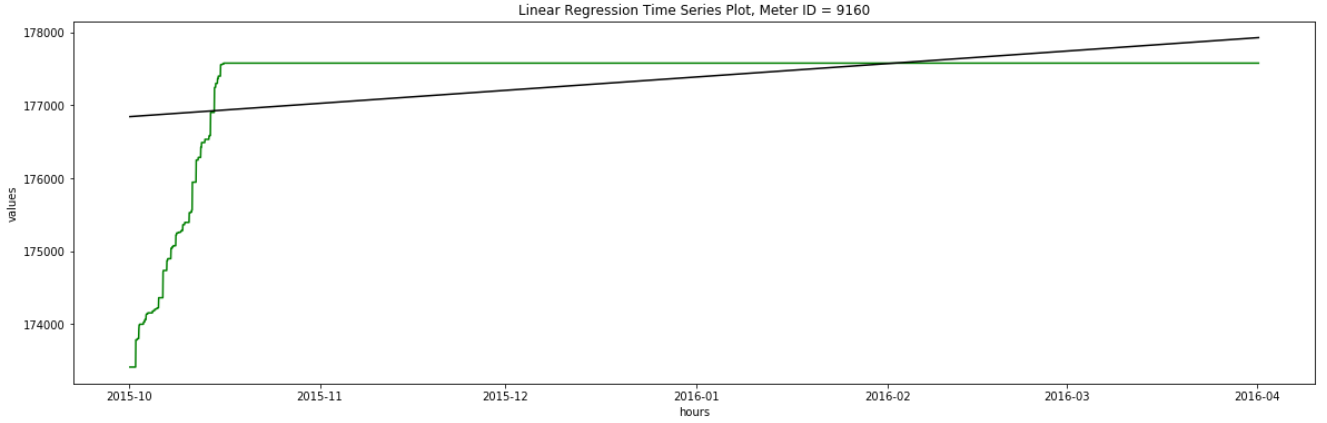

Linear Regression Time Series Plot, Meter ID = 5658
values
153000
152000
151000
150000
149000
148000
147000
146000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

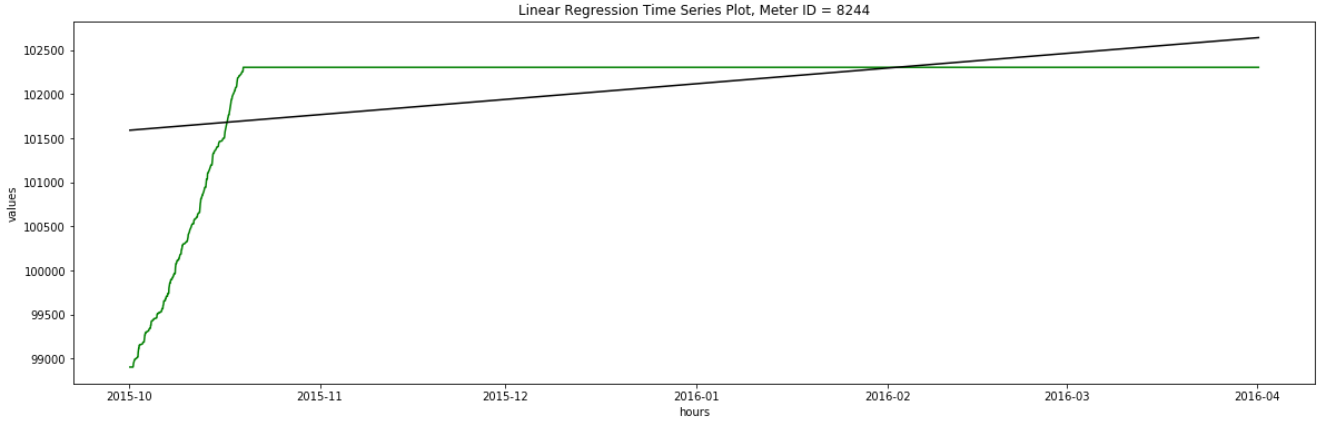

Linear Regression Time Series Plot, Meter ID = 5545
values
150000
148000
146000
144000
142000
140000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

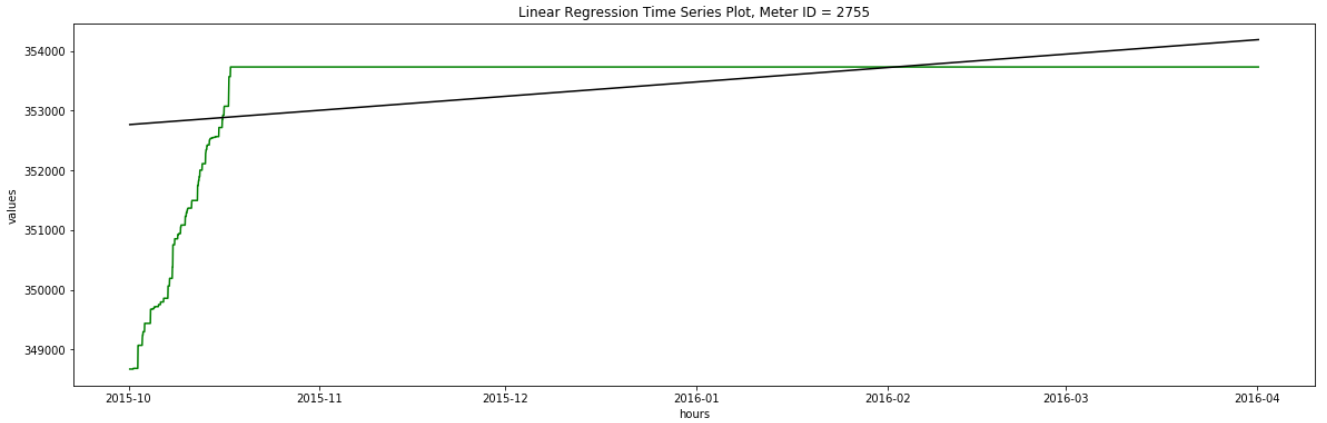

Linear Regression Time Series Plot, Meter ID = 5317
values
28800
28700
28600
28500
28400
28300
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

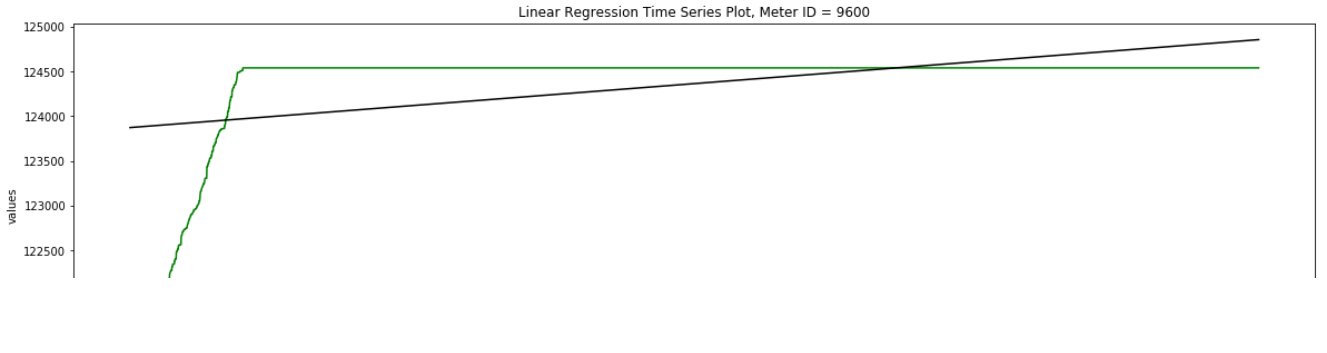

Linear Regression Time Series Plot, Meter ID = 3036

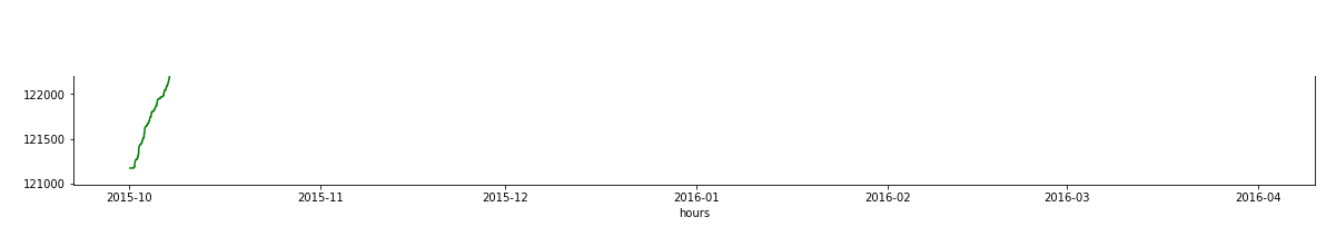

152000
151000
150000
149000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

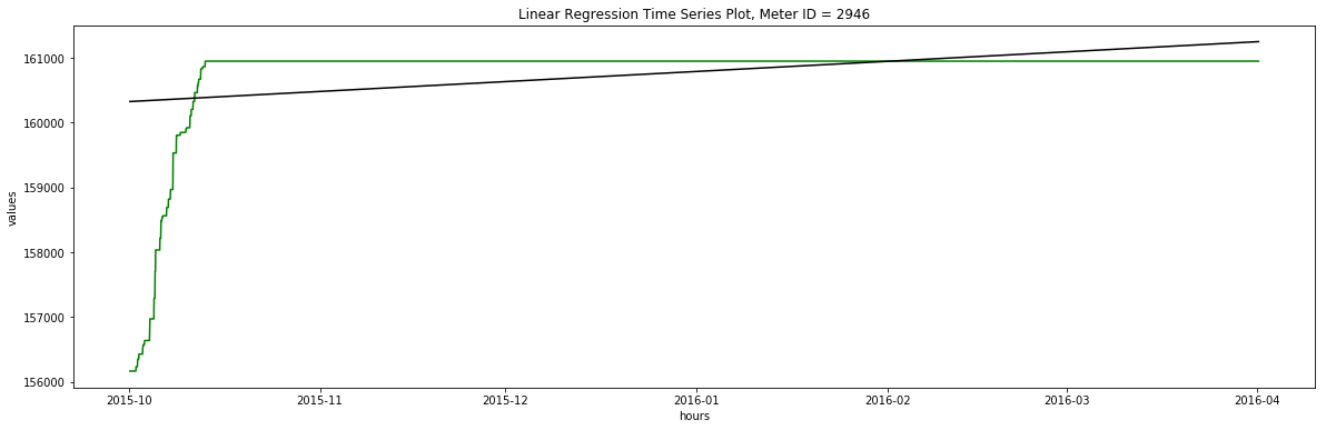

Linear Regression Time Series Plot, Meter ID = 9160
178000
177000
176000
175000
174000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

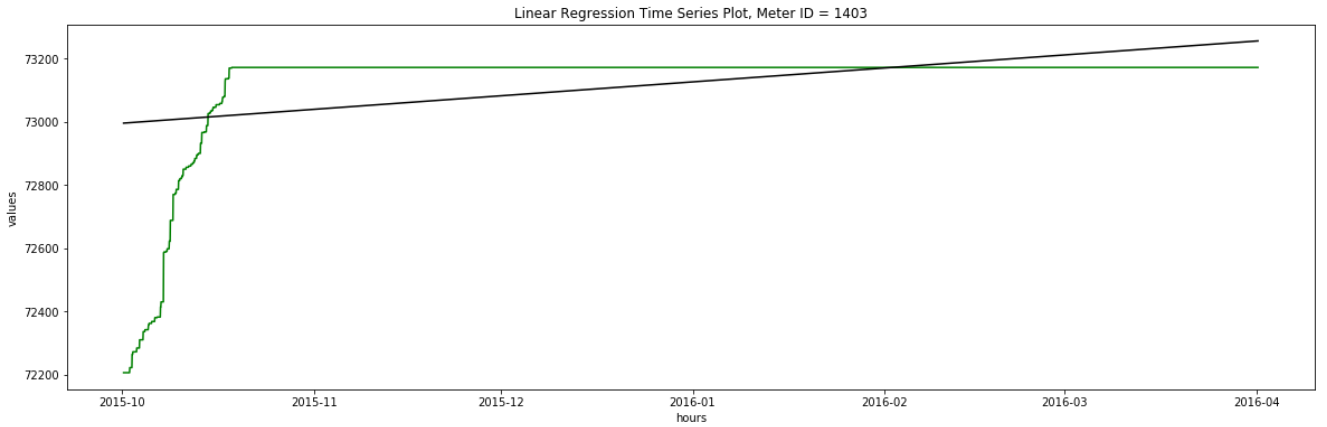

Linear Regression Time Series Plot, Meter ID = 8244
102500
102000
101500
101000
100500
100000
99500
99000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

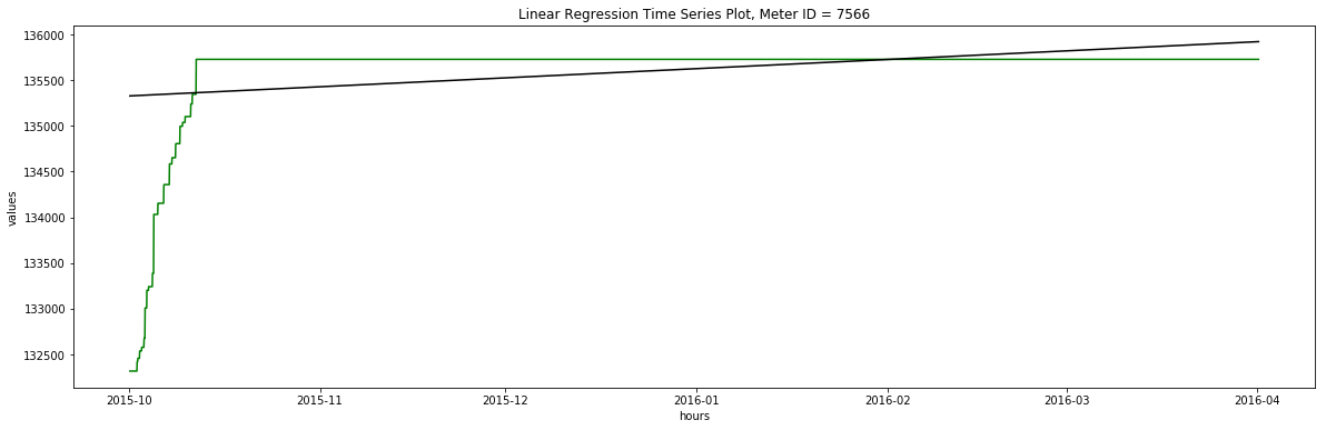

Linear Regression Time Series Plot, Meter ID = 2755
354000
353000
352000
351000
350000
349000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

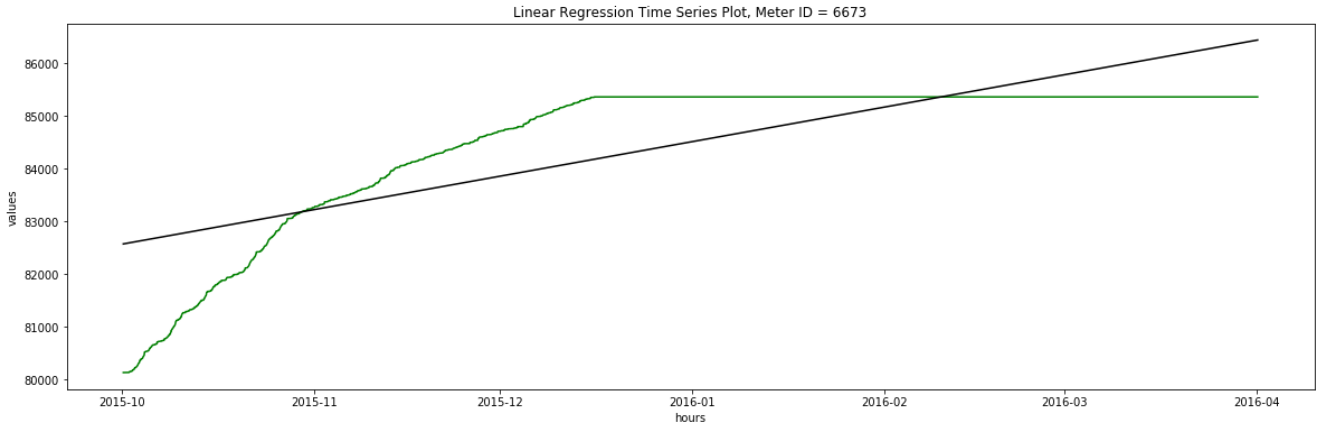

Linear Regression Time Series Plot, Meter ID = 9600
125000
124500
124000
123500
123000
122500
values

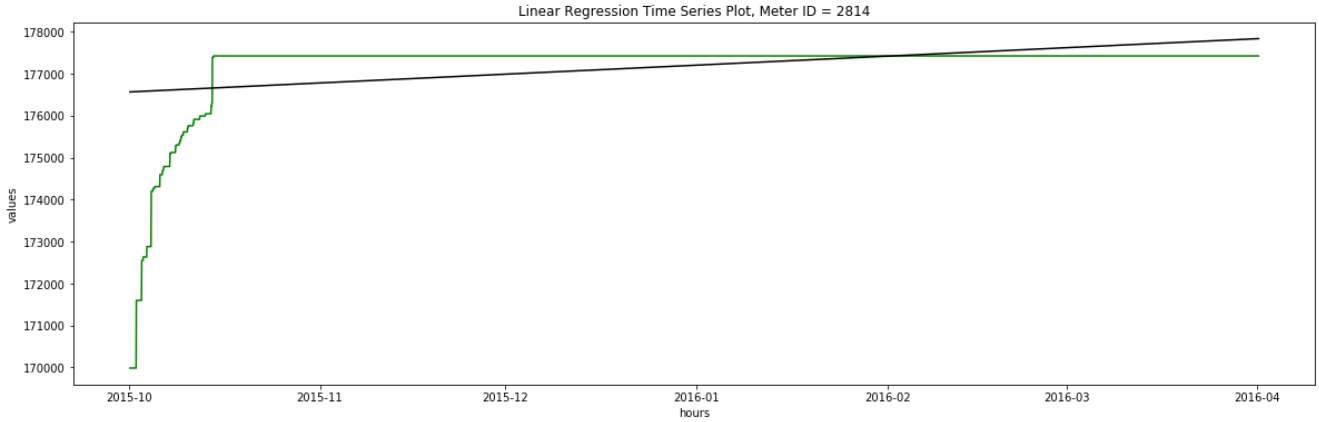

122000
121500
121000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

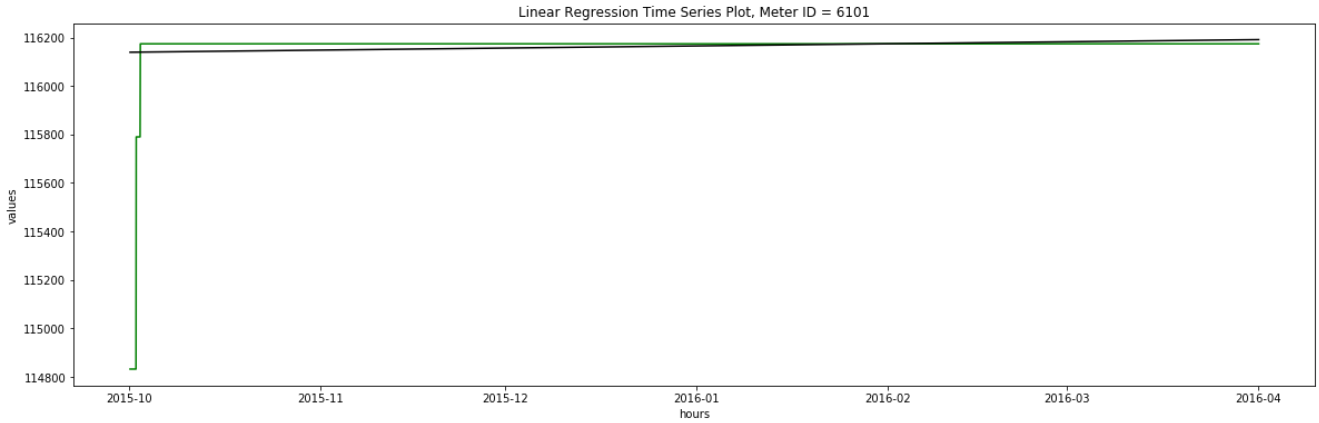

Linear Regression Time Series Plot, Meter ID = 2946
161000
160000
159000
158000
157000
156000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

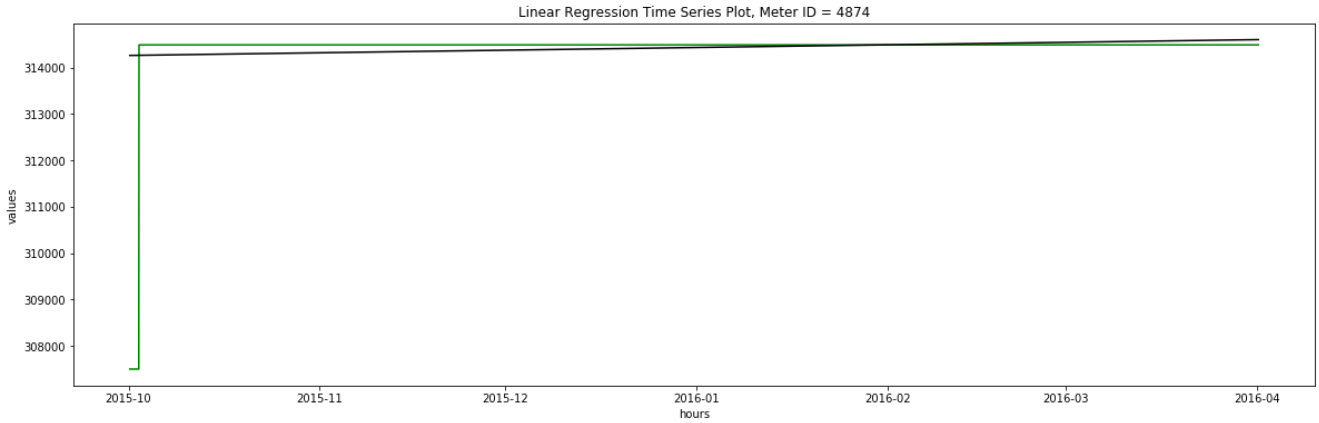

Linear Regression Time Series Plot, Meter ID = 1403
73200
73000
72800
72600
72400
72200
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours


Linear Regression Time Series Plot, Meter ID = 7566
136000
135500
135000
134500
134000
133500
133000
132500
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours


Linear Regression Time Series Plot, Meter ID = 6673
86000
85000
84000
83000
82000
81000
80000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours



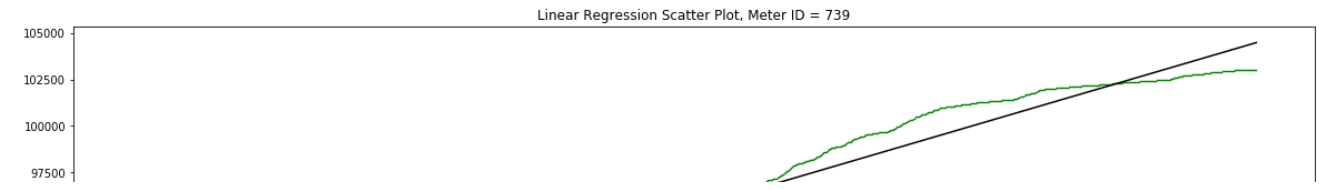


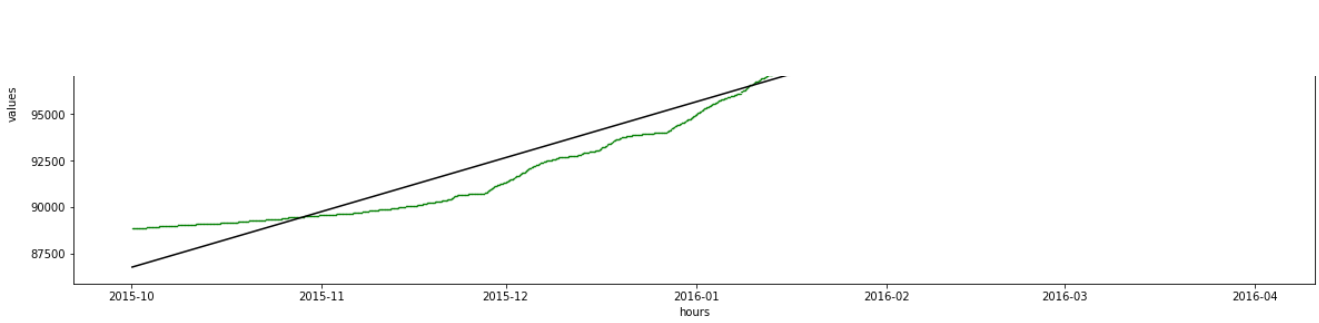


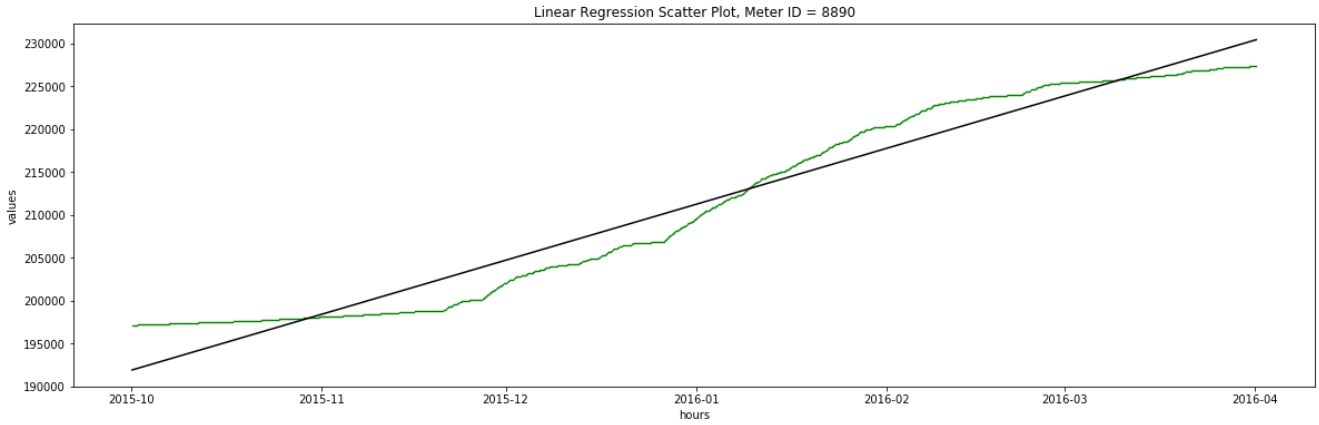
ii) Scatter Plot

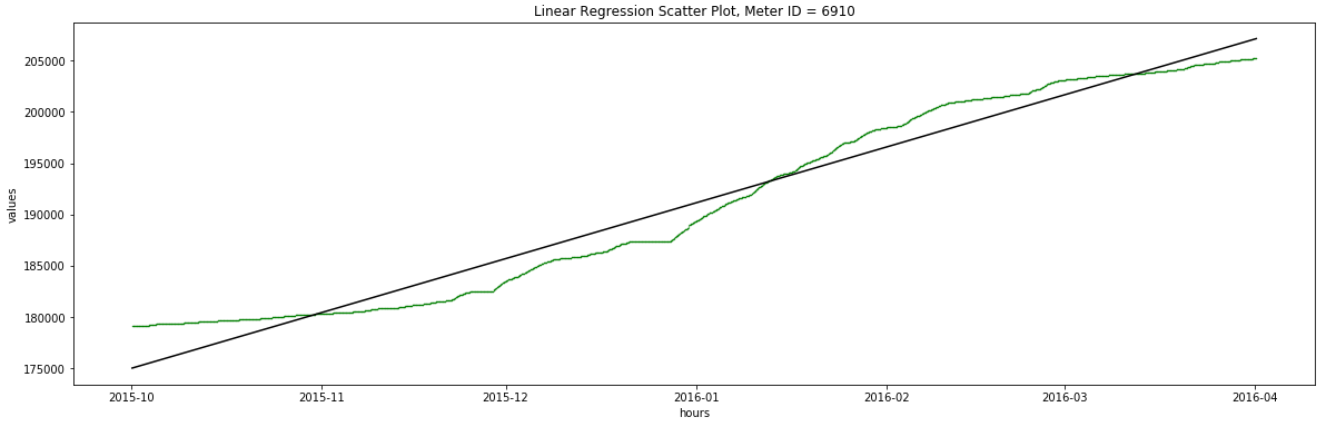
In [19]:

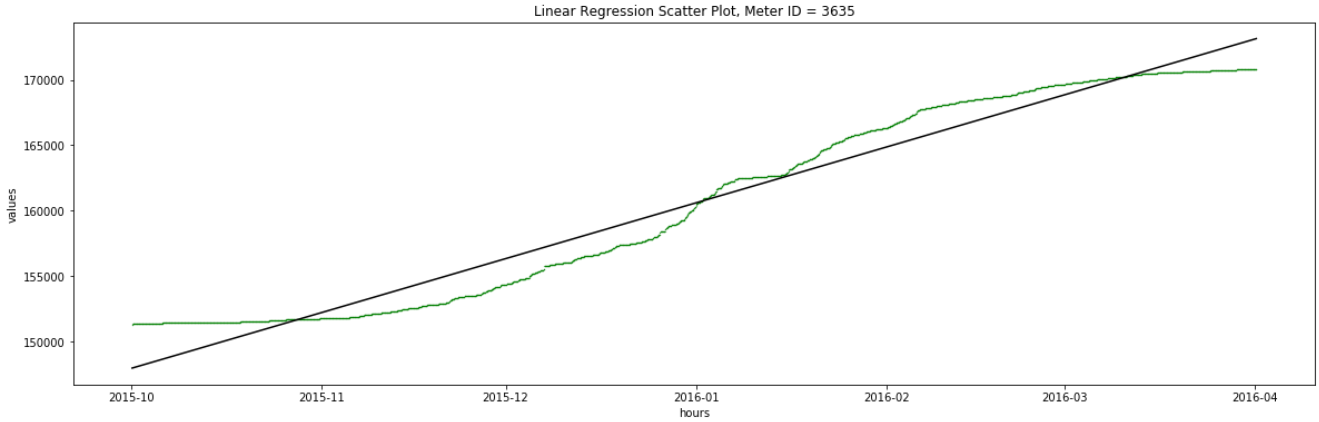
```
for key in hourlyDataDict:
    fig, ax = plt.subplots(1, 1, figsize=(20, 6))

    plt.xlabel('hours')
    plt.ylabel('values')
    x = dateTimeList
    y = np.asarray(hourlyDataDict[key])
    ax.scatter(x, y,color='g',s=0.05)
    ax.plot(x,  newPredLR(key) ,color='k')
    plt.title("Linear Regression Scatter Plot, Meter ID = " + str(key))
    plt.show()
```

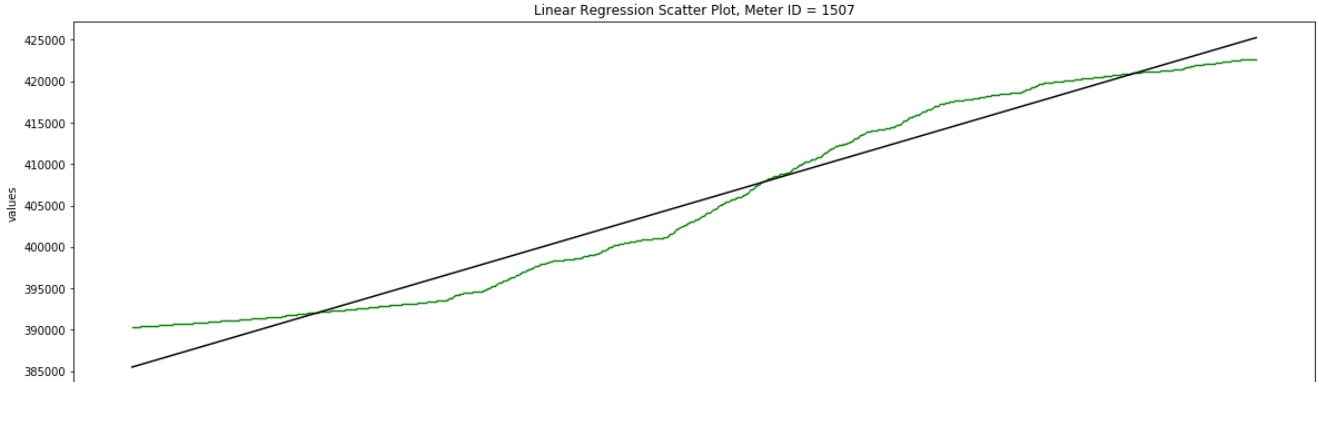



values
95000
92500
90000
87500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

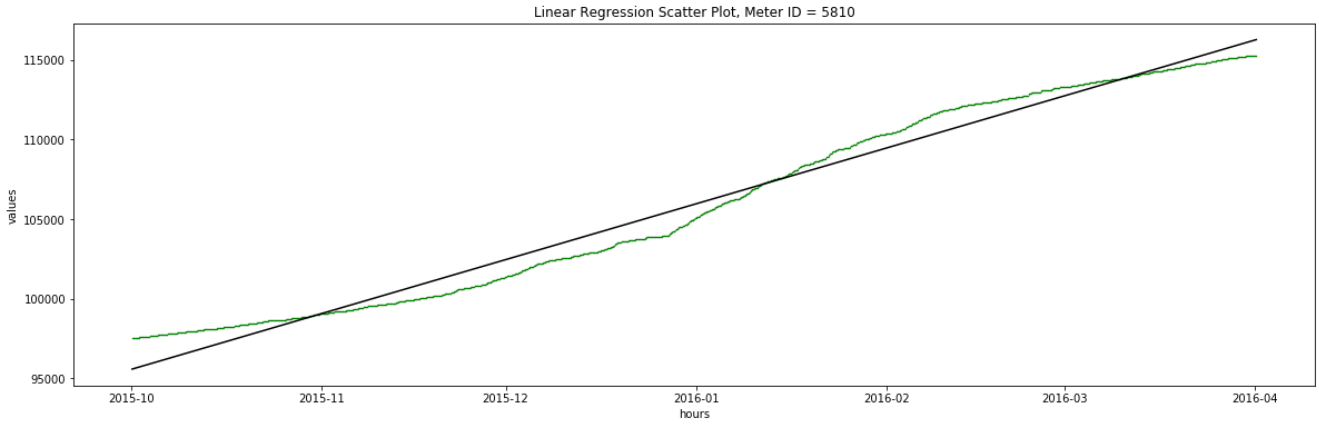

Linear Regression Scatter Plot, Meter ID = 8890
values
230000
225000
220000
215000
210000
205000
200000
195000
190000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

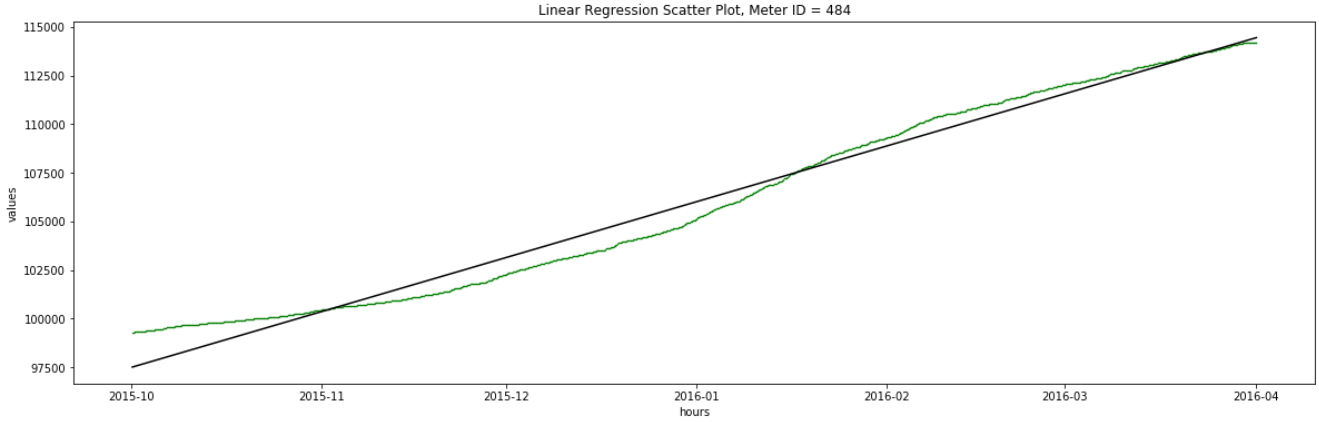

Linear Regression Scatter Plot, Meter ID = 6910
values
205000
200000
195000
190000
185000
180000
175000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

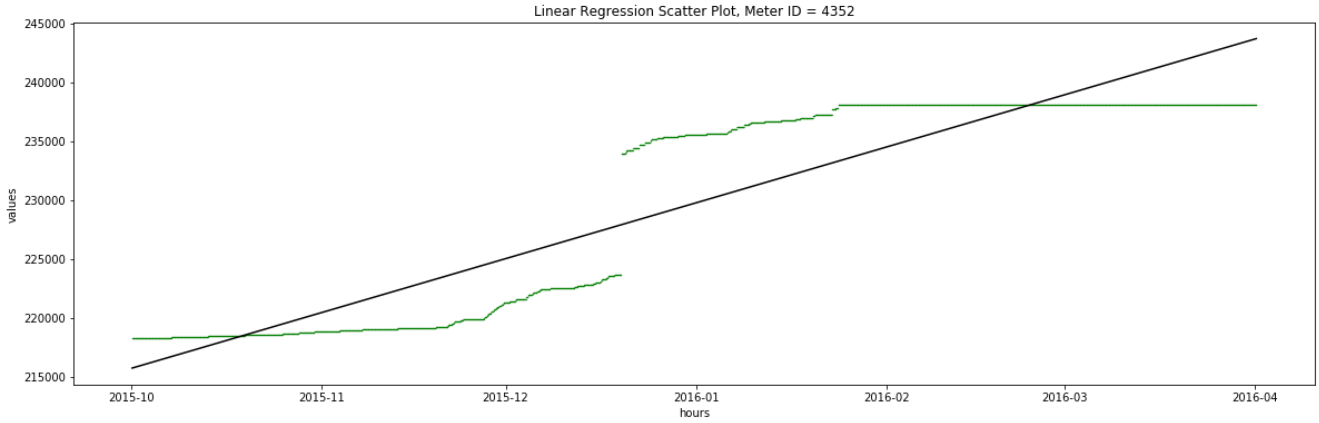

Linear Regression Scatter Plot, Meter ID = 3635
values
170000
165000
160000
155000
150000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

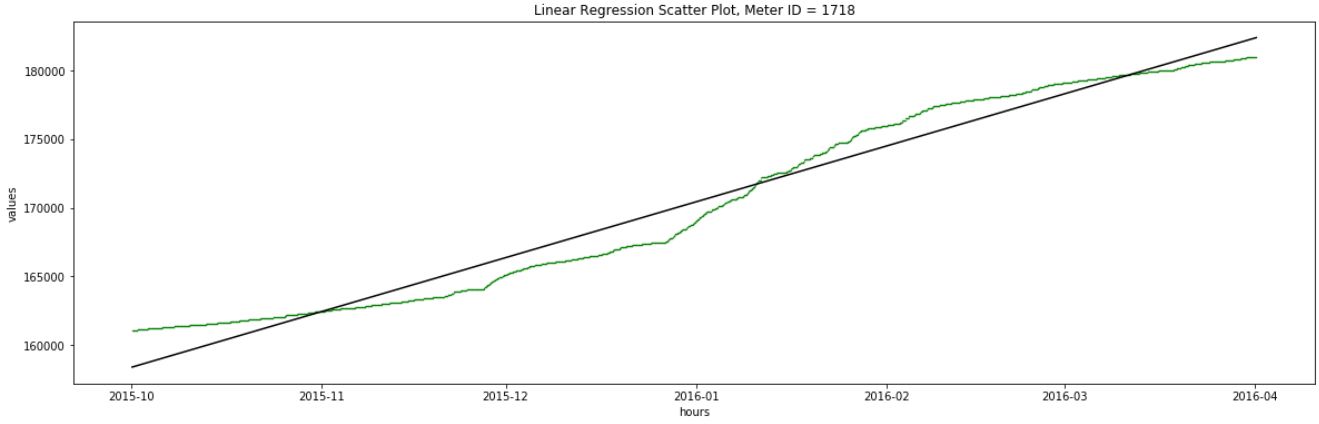

Linear Regression Scatter Plot, Meter ID = 1507
values
425000
420000
415000
410000
405000
400000
395000
390000
385000

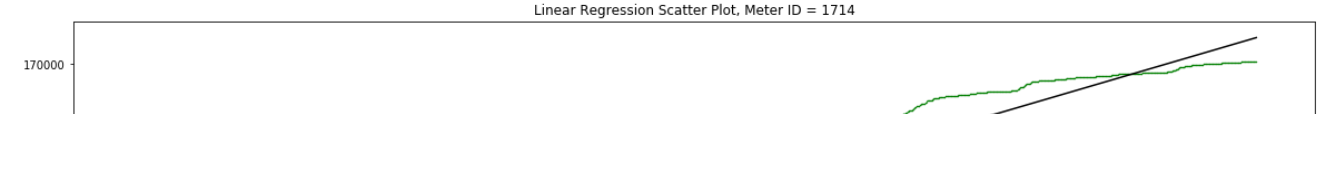

2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

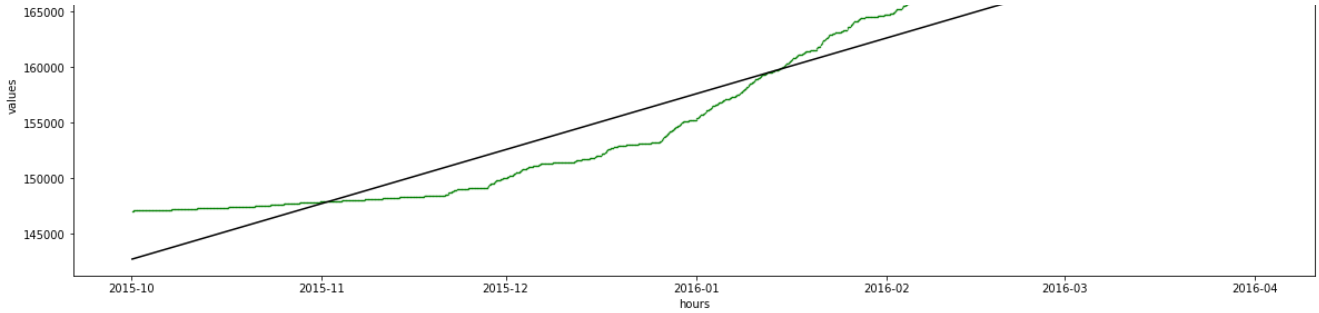

Linear Regression Scatter Plot, Meter ID = 5810
values
115000
110000
105000
100000
95000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

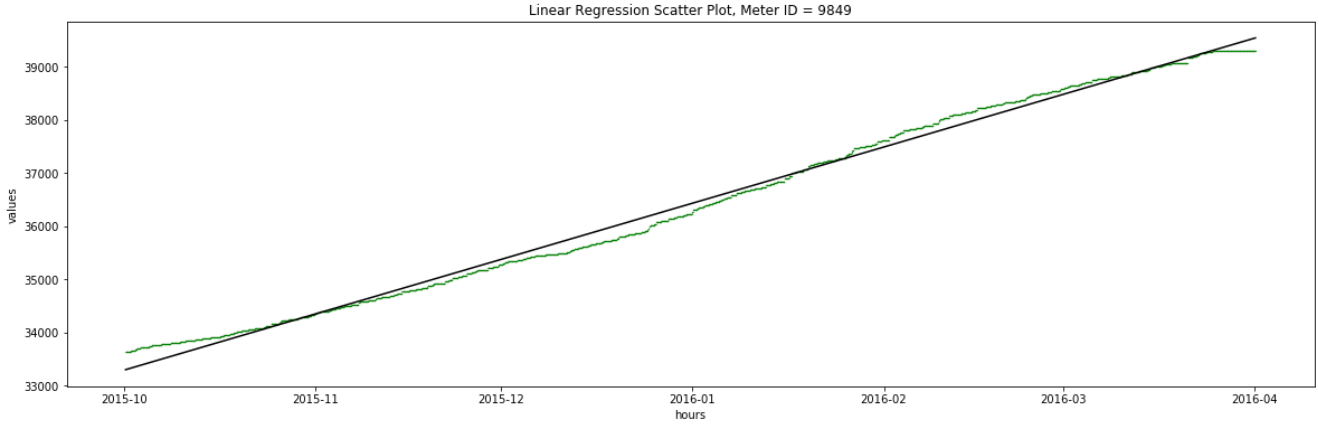

Linear Regression Scatter Plot, Meter ID = 484
values
115000
112500
110000
107500
105000
102500
100000
97500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

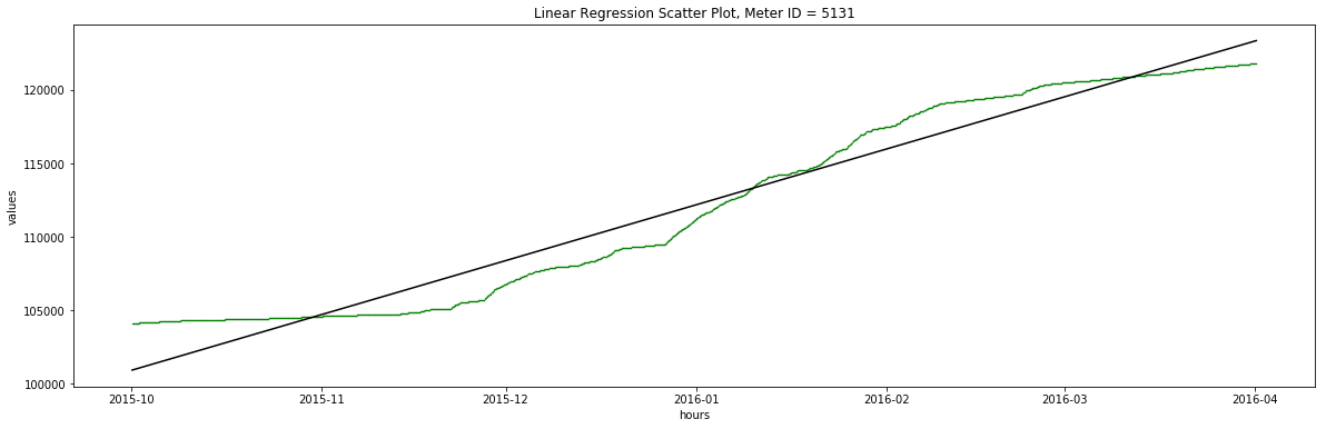

Linear Regression Scatter Plot, Meter ID = 4352
values
245000
240000
235000
230000
225000
220000
215000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

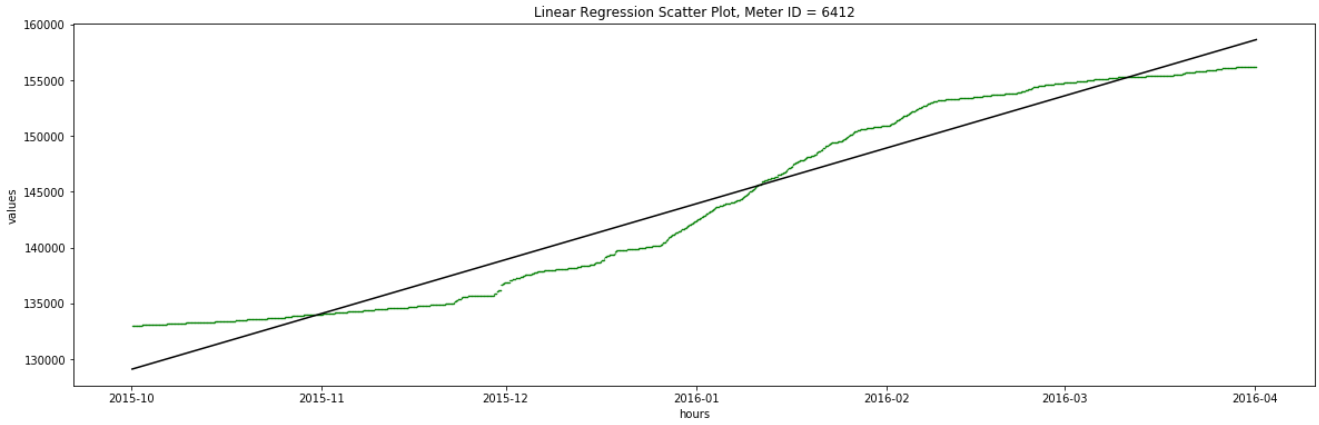

Linear Regression Scatter Plot, Meter ID = 1718
values
180000
175000
170000
165000
160000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

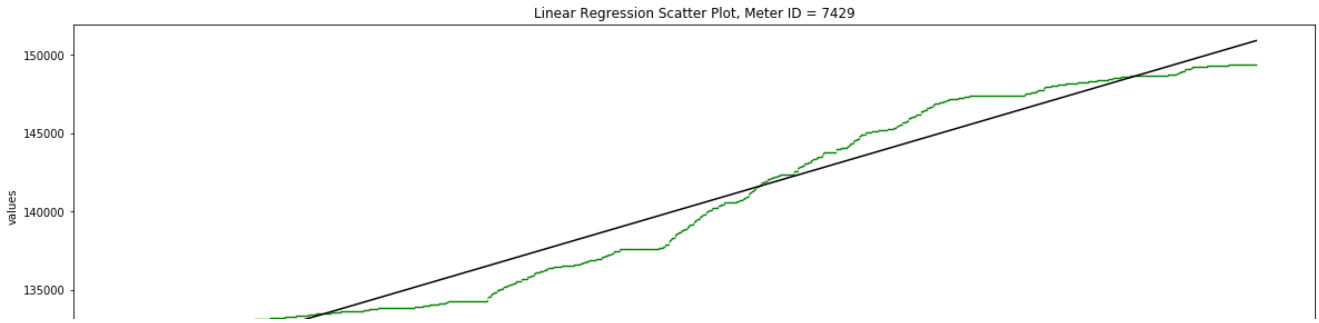

Linear Regression Scatter Plot, Meter ID = 1714
170000

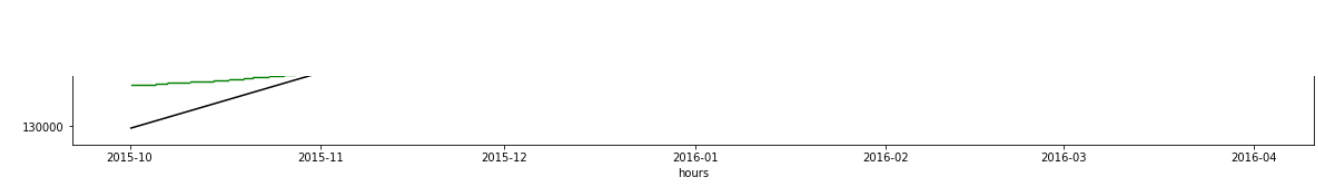

values
hours
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

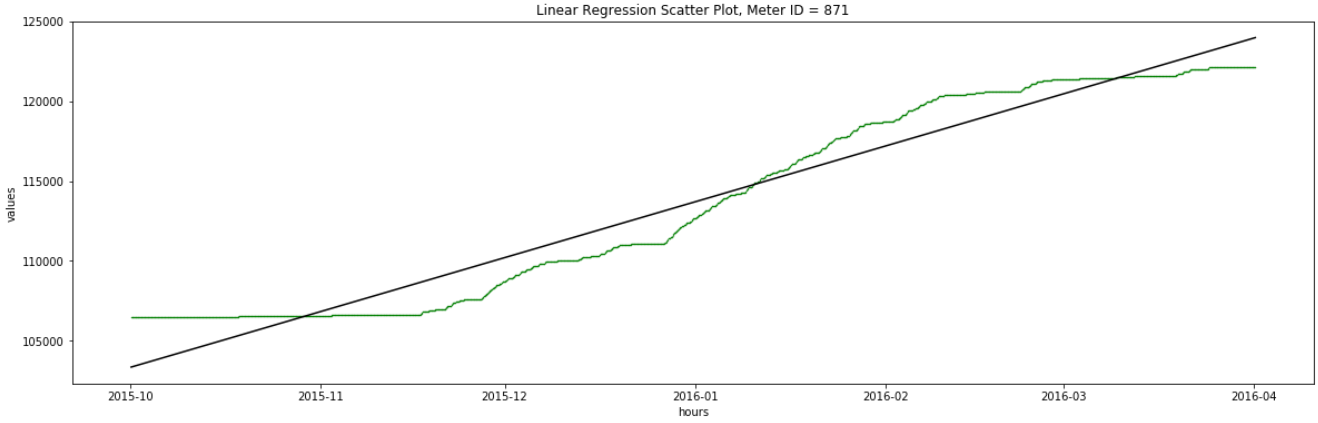

Linear Regression Scatter Plot, Meter ID = 9849
values
hours

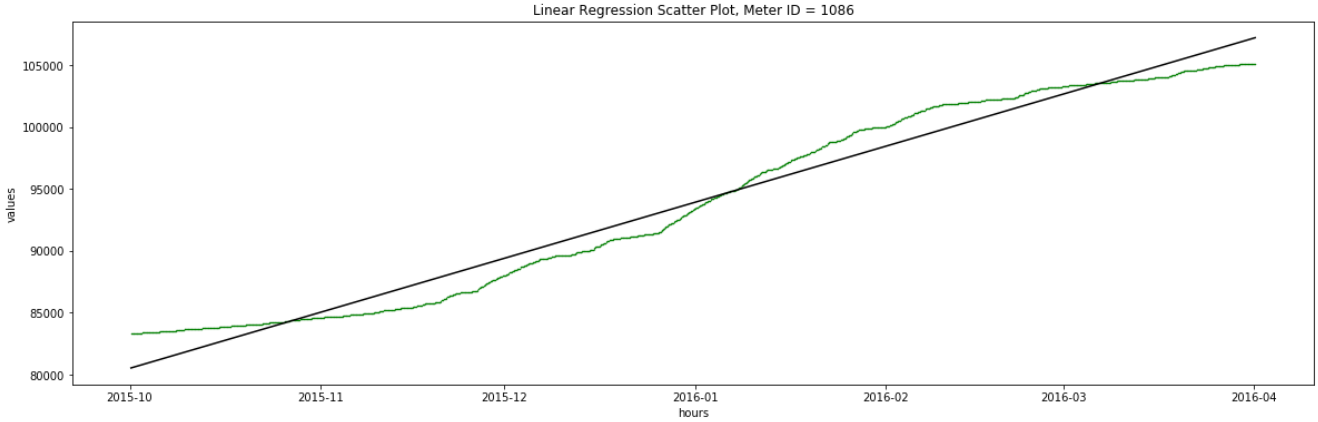

Linear Regression Scatter Plot, Meter ID = 5131
values
hours

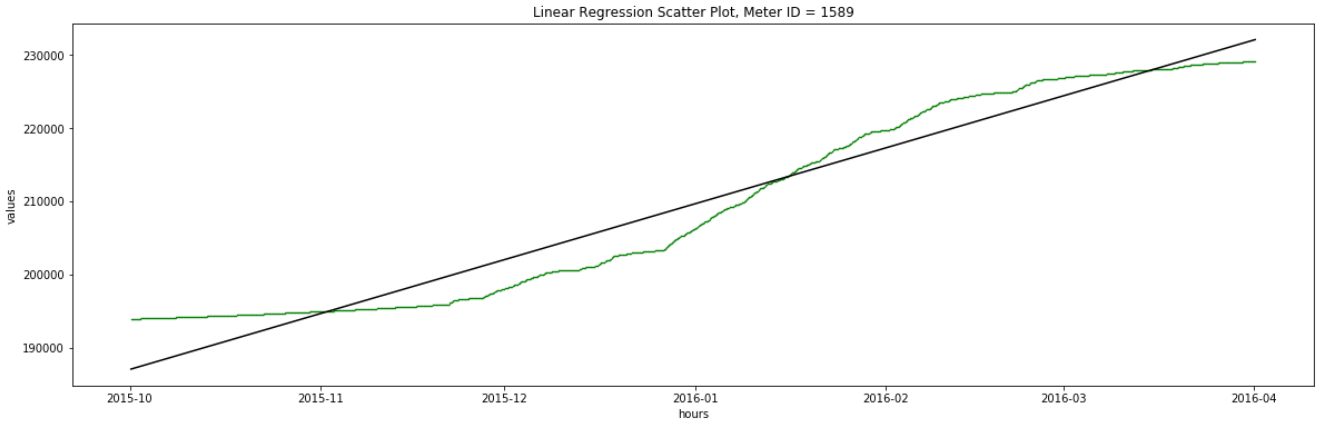

Linear Regression Scatter Plot, Meter ID = 6412
values
hours

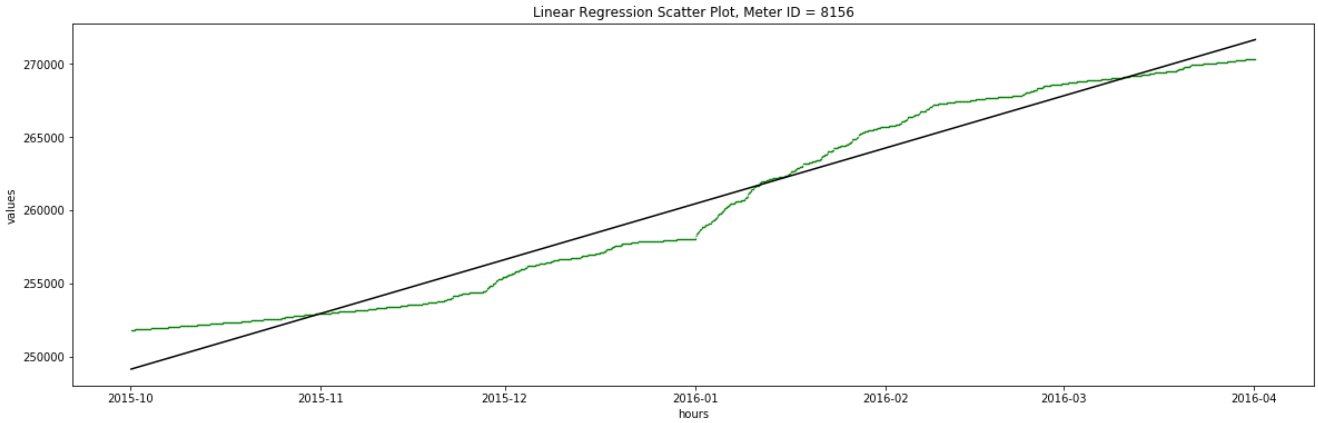

Linear Regression Scatter Plot, Meter ID = 7429
values

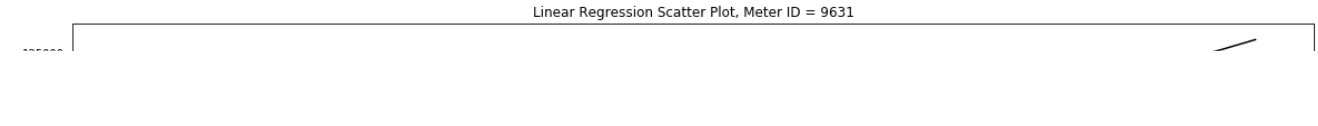

130000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

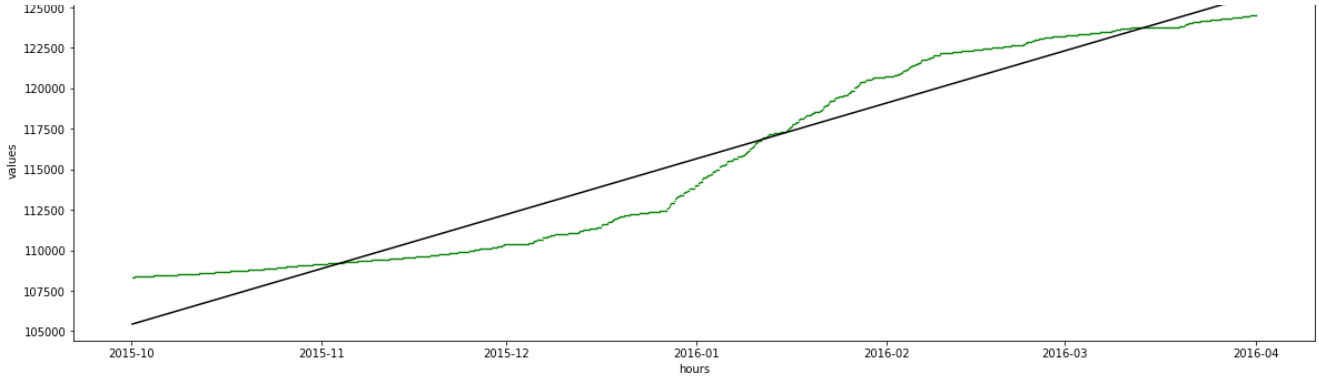

Linear Regression Scatter Plot, Meter ID = 871
125000
120000
115000
110000
105000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

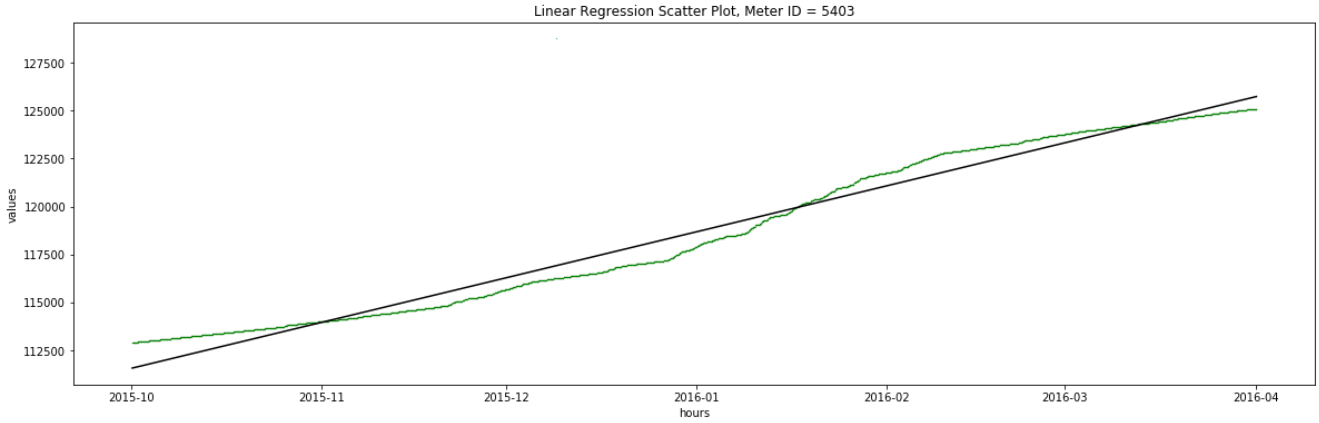

Linear Regression Scatter Plot, Meter ID = 1086
105000
100000
95000
90000
85000
80000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

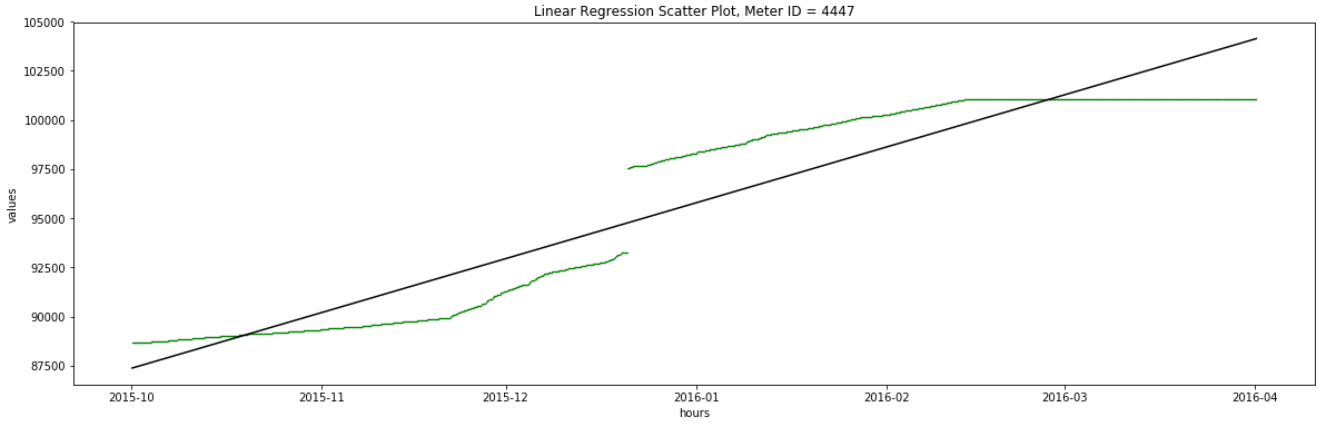

Linear Regression Scatter Plot, Meter ID = 1589
230000
220000
210000
200000
190000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

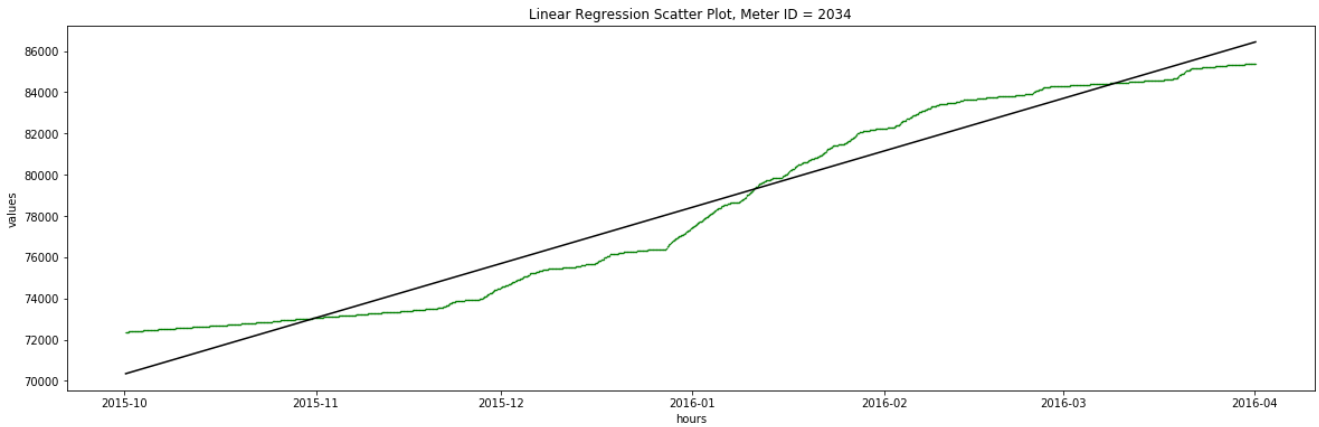

Linear Regression Scatter Plot, Meter ID = 8156
270000
265000
260000
255000
250000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

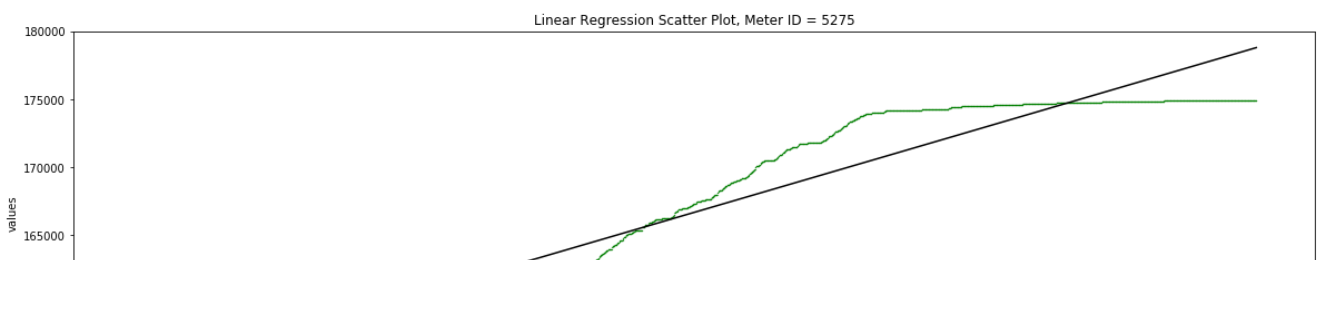

Linear Regression Scatter Plot, Meter ID = 9631

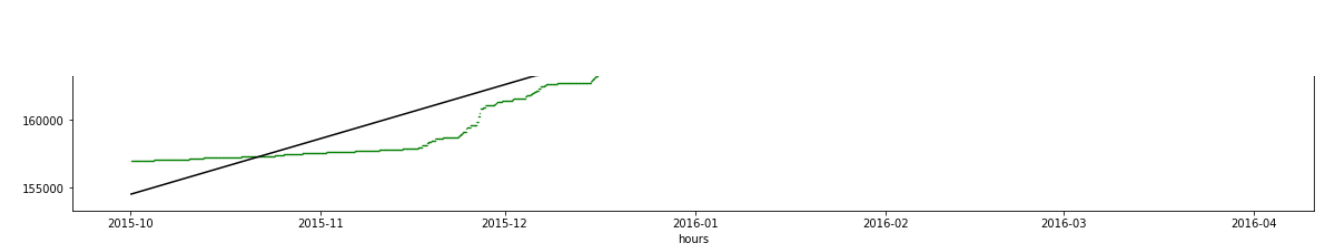

values
hours
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

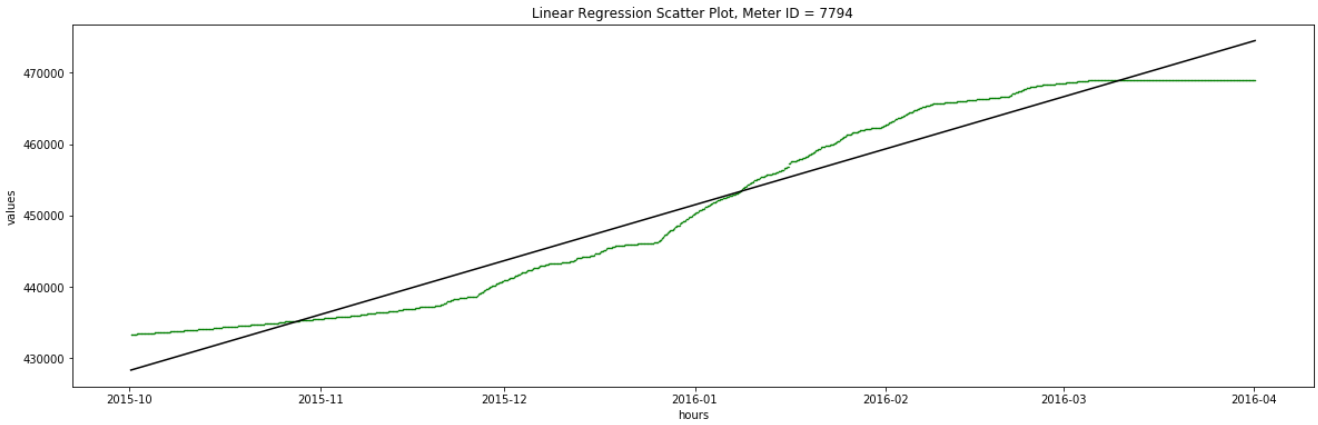

Linear Regression Scatter Plot, Meter ID = 5403
values
hours
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

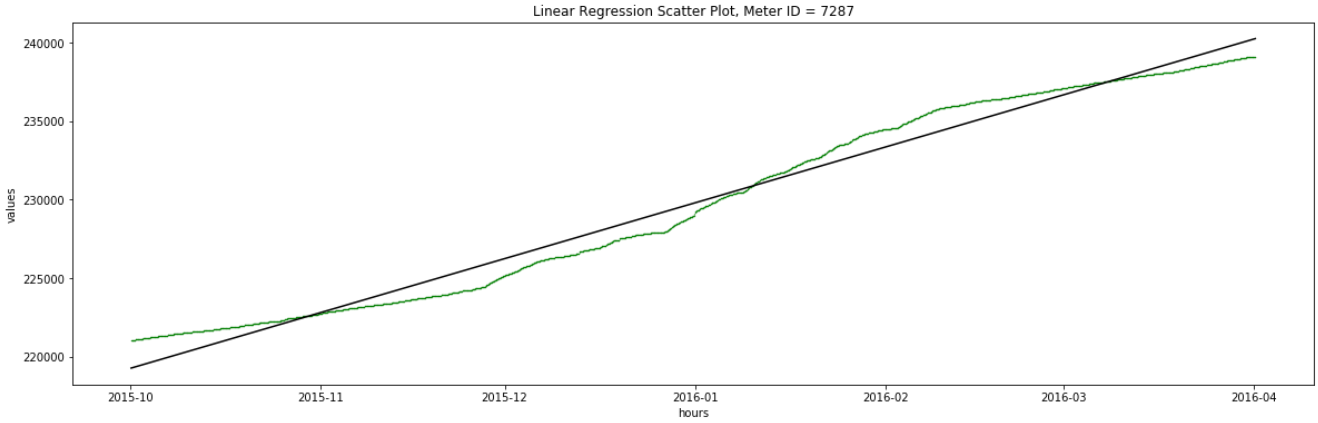

Linear Regression Scatter Plot, Meter ID = 4447
values
hours
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

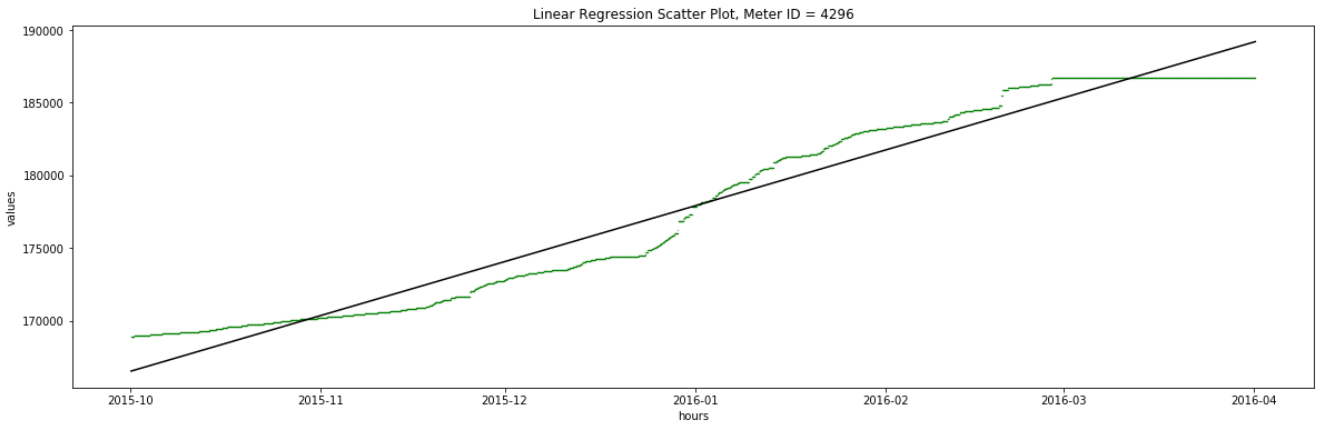

Linear Regression Scatter Plot, Meter ID = 2034
values
hours
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

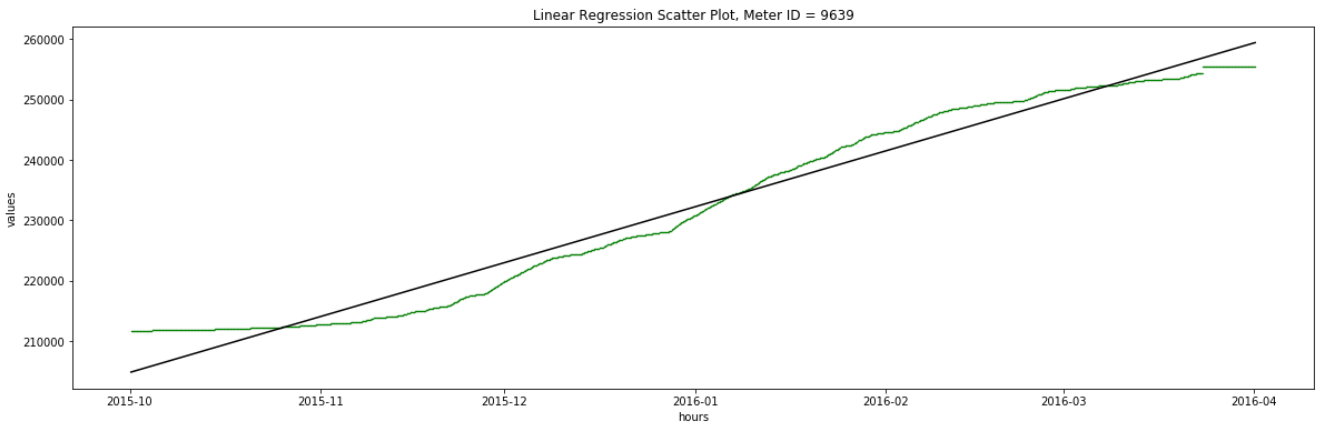

Linear Regression Scatter Plot, Meter ID = 5275
values

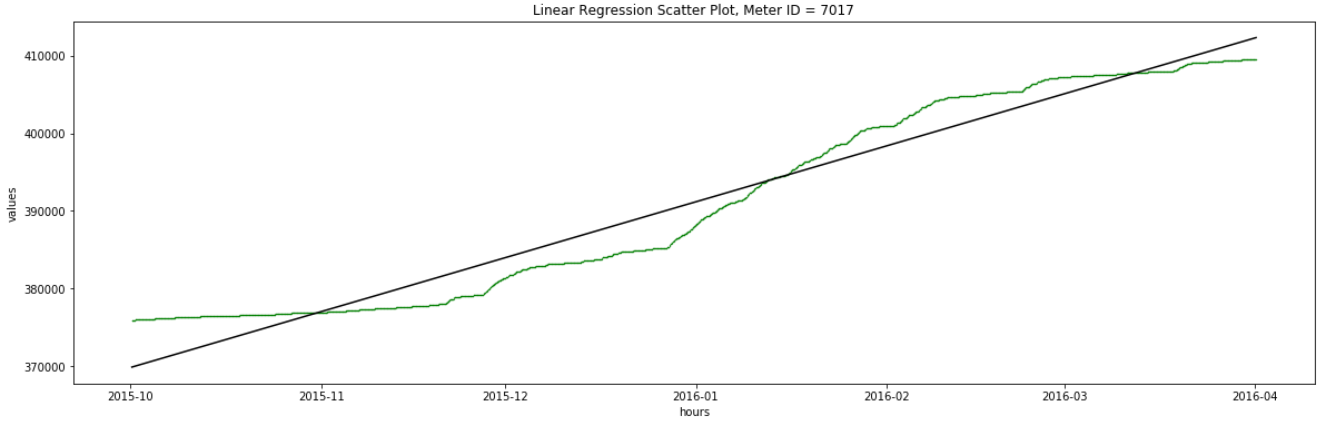

160000
155000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

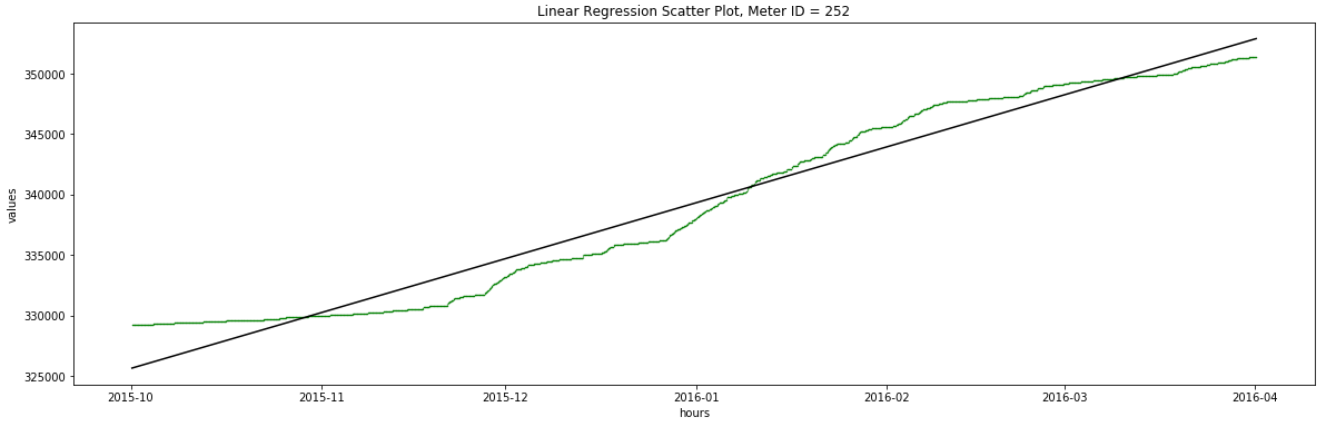

Linear Regression Scatter Plot, Meter ID = 7794
values
470000
460000
450000
440000
430000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

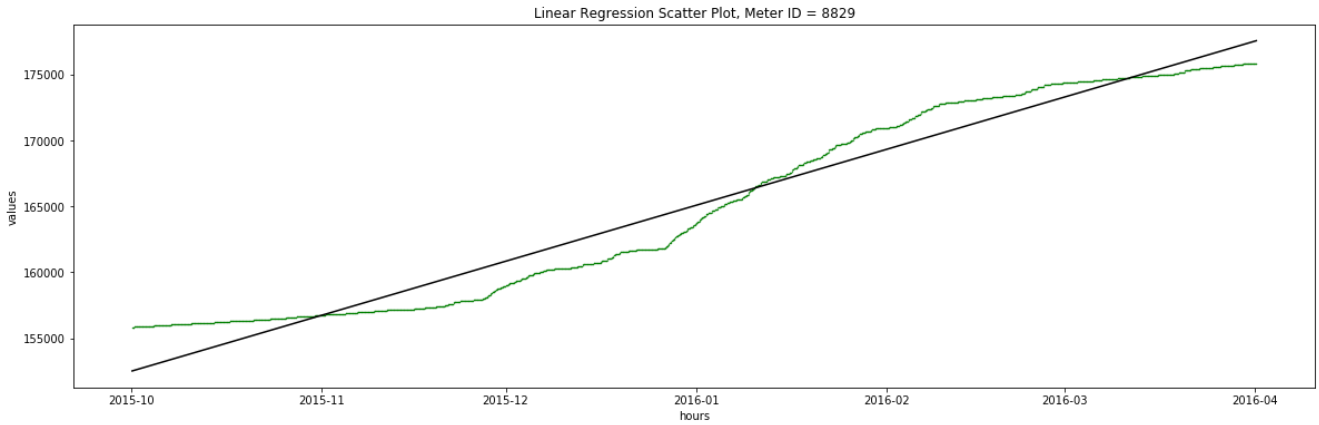

Linear Regression Scatter Plot, Meter ID = 7287
values
240000
235000
230000
225000
220000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

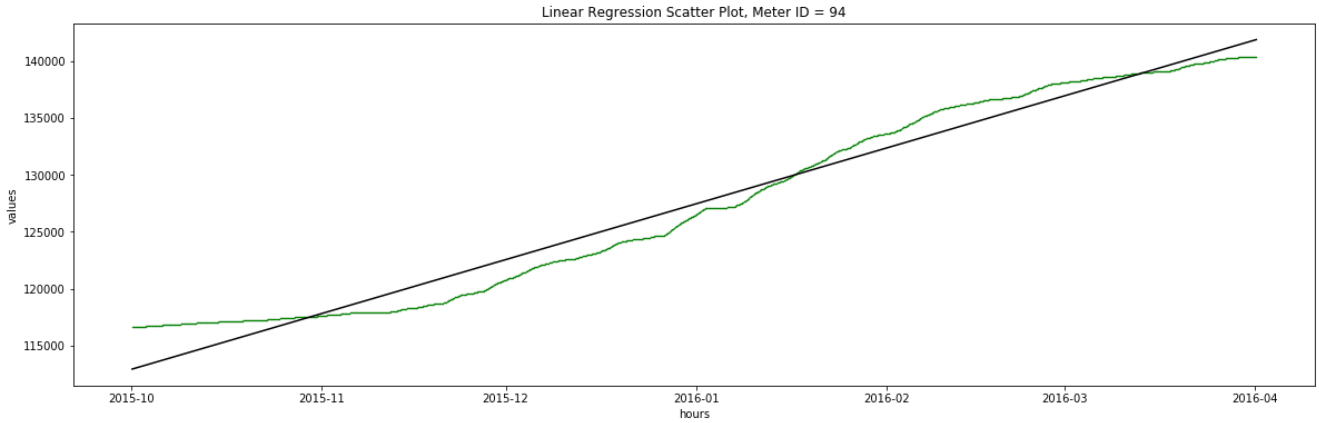

Linear Regression Scatter Plot, Meter ID = 4296
values
190000
185000
180000
175000
170000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

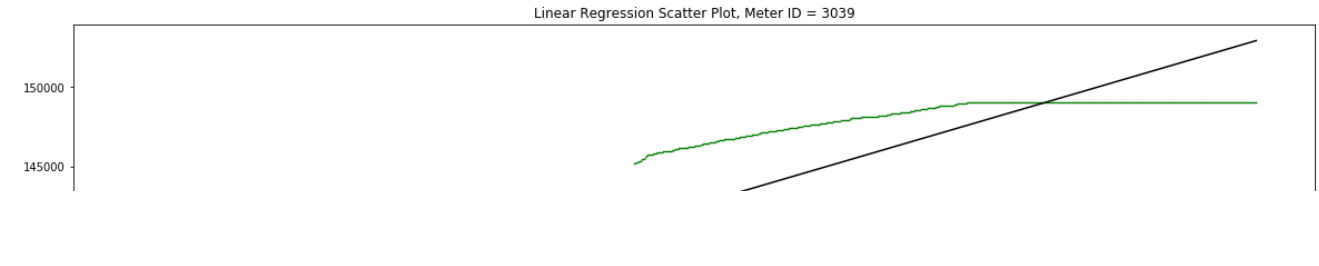

Linear Regression Scatter Plot, Meter ID = 9639
values
260000
250000
240000
230000
220000
210000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

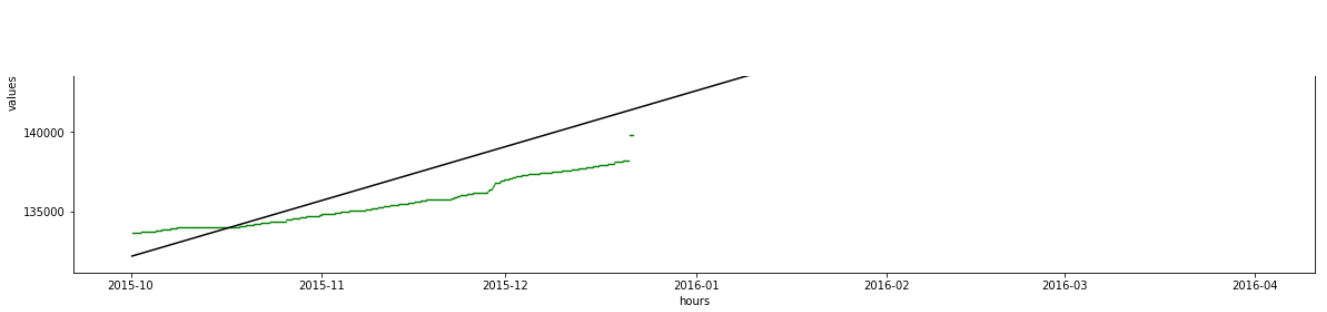

Linear Regression Scatter Plot, Meter ID = 7017
values
hours
410000
400000
390000
380000
370000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

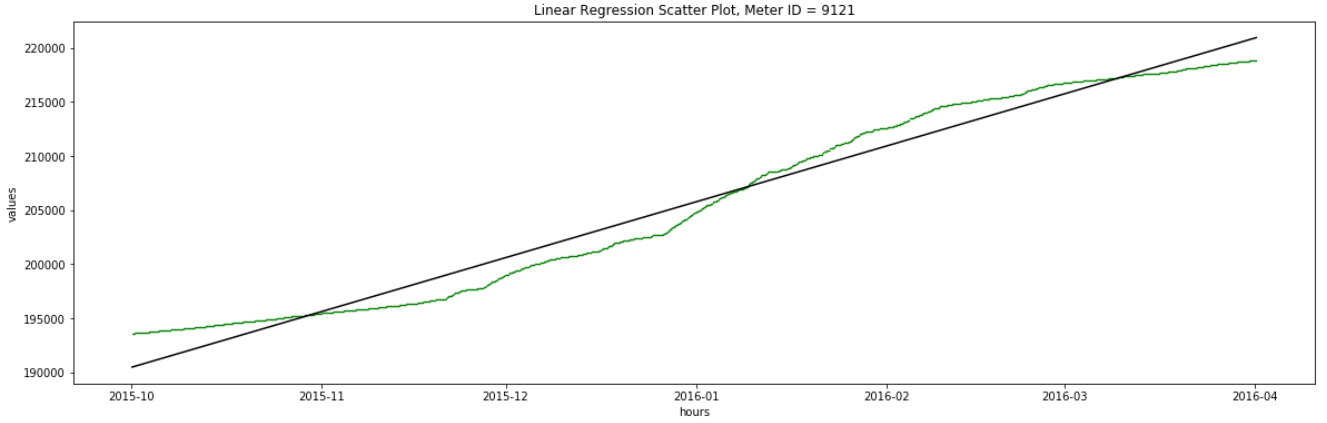

Linear Regression Scatter Plot, Meter ID = 252
values
hours
350000
345000
340000
335000
330000
325000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

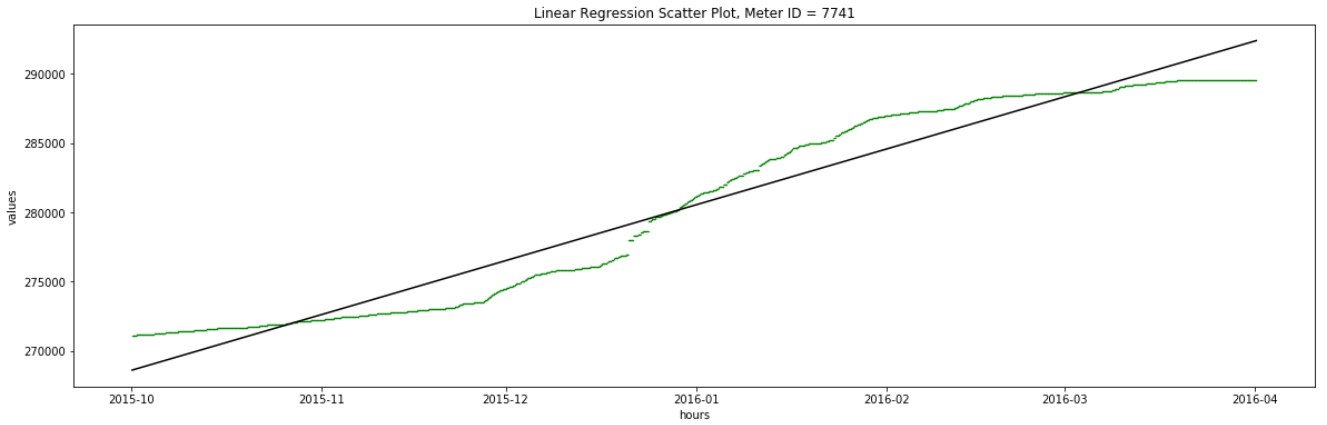

Linear Regression Scatter Plot, Meter ID = 8829
values
hours
175000
170000
165000
160000
155000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

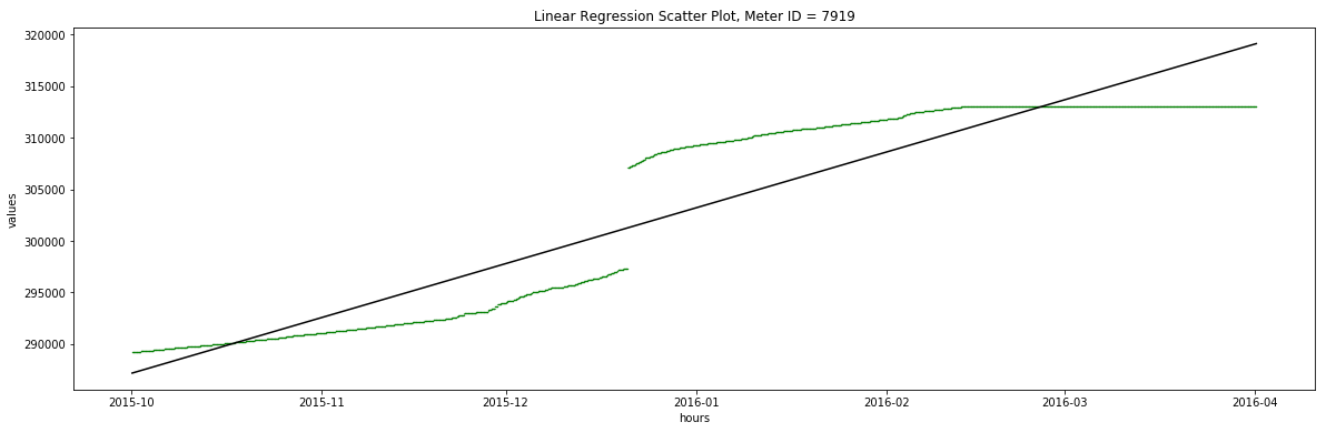

Linear Regression Scatter Plot, Meter ID = 94
values
hours
140000
135000
130000
125000
120000
115000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

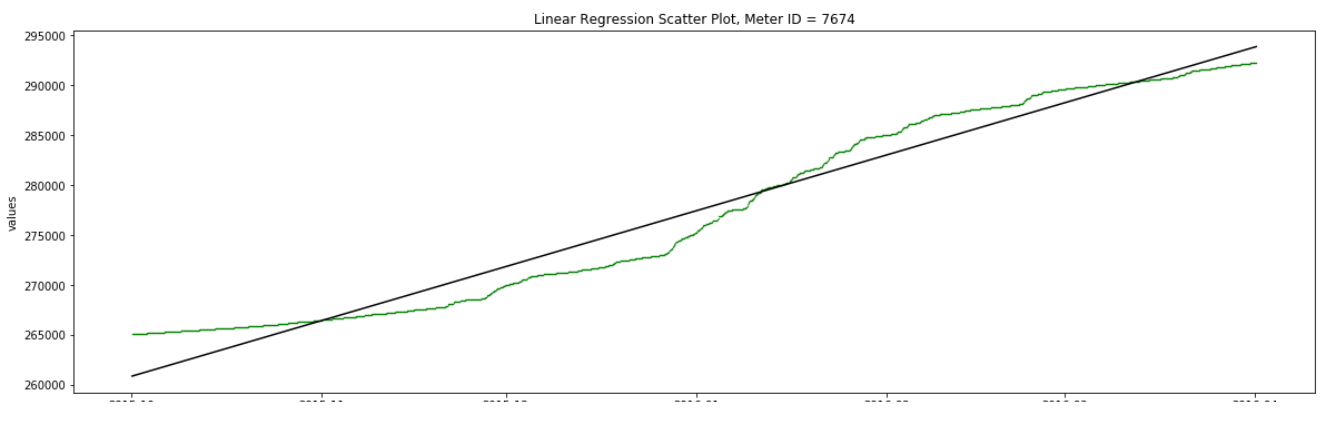

Linear Regression Scatter Plot, Meter ID = 3039
150000
145000


values
140000
135000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

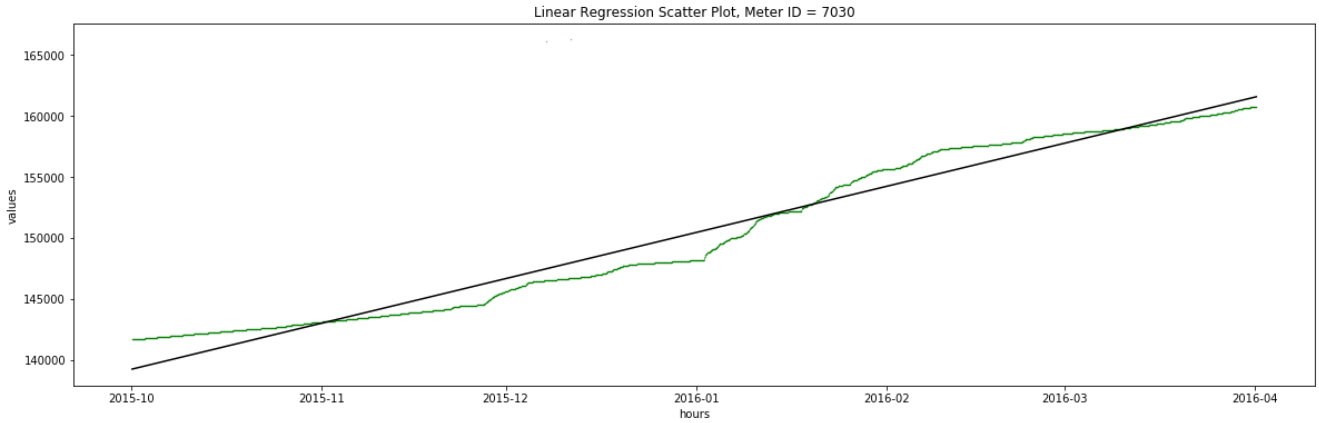

Linear Regression Scatter Plot, Meter ID = 9121
values
220000
215000
210000
205000
200000
195000
190000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

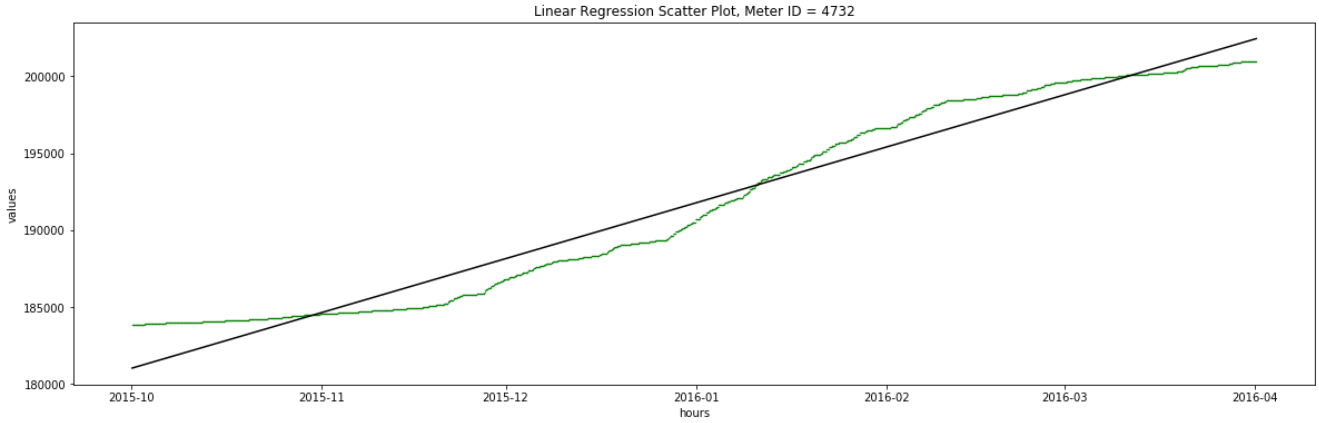

Linear Regression Scatter Plot, Meter ID = 7741
values
290000
285000
280000
275000
270000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

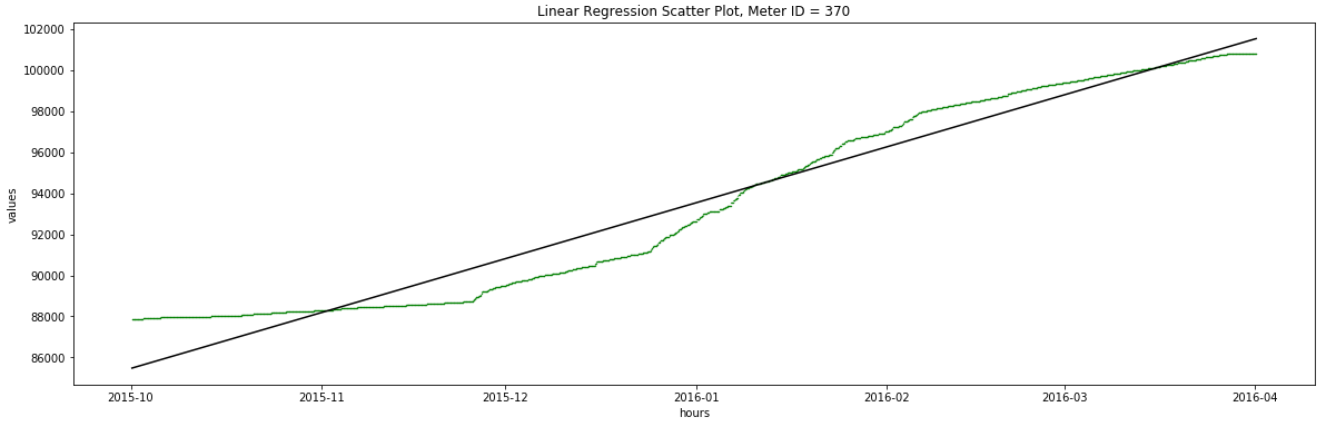

Linear Regression Scatter Plot, Meter ID = 7919
values
320000
315000
310000
305000
300000
295000
290000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

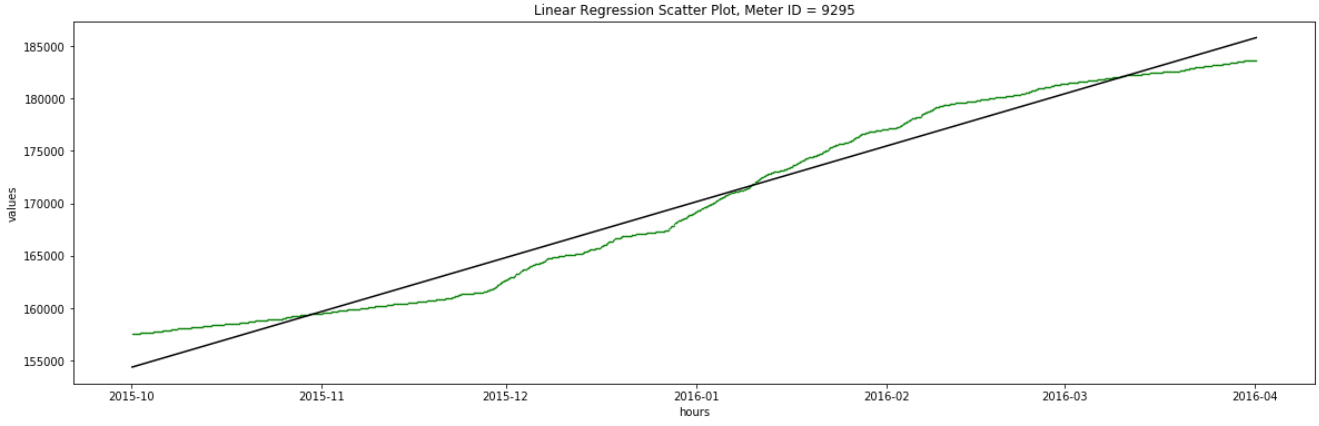

Linear Regression Scatter Plot, Meter ID = 7674
values
295000
290000
285000
280000
275000
270000
265000
260000

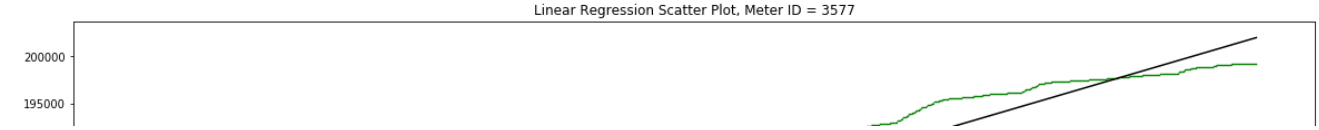

2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

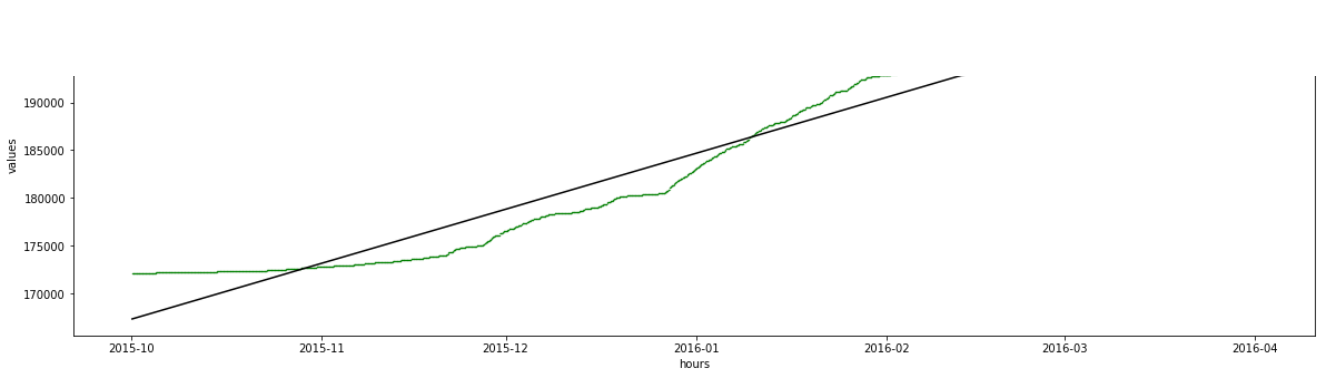

Linear Regression Scatter Plot, Meter ID = 7030
values
hours

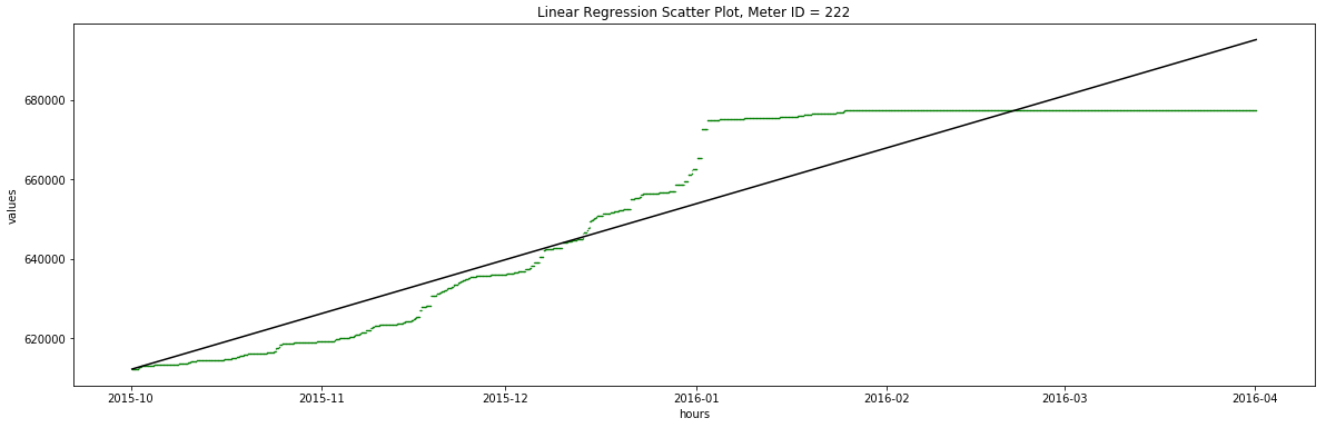

Linear Regression Scatter Plot, Meter ID = 4732
values
hours

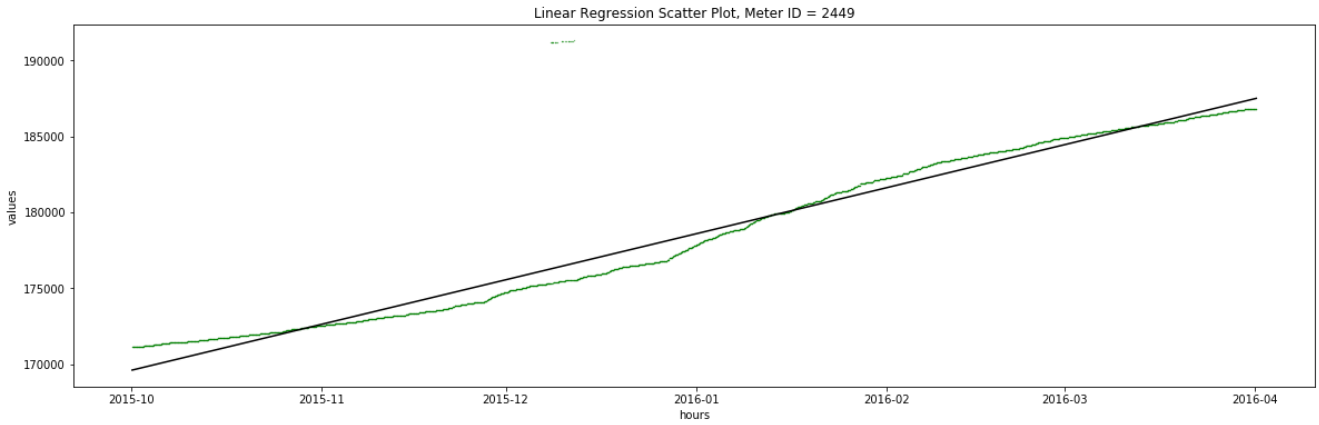

Linear Regression Scatter Plot, Meter ID = 370
values
hours

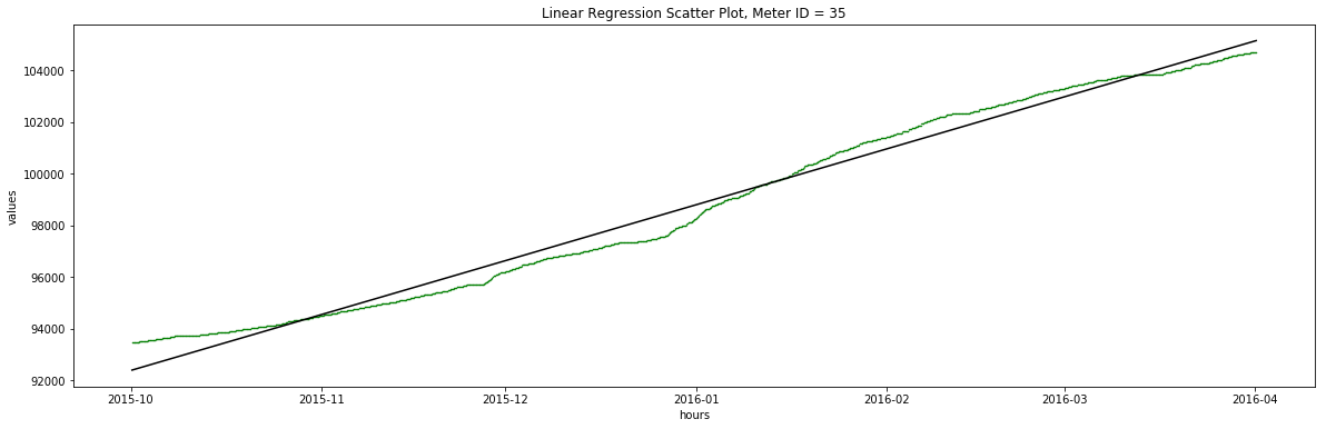

Linear Regression Scatter Plot, Meter ID = 9295
values
hours

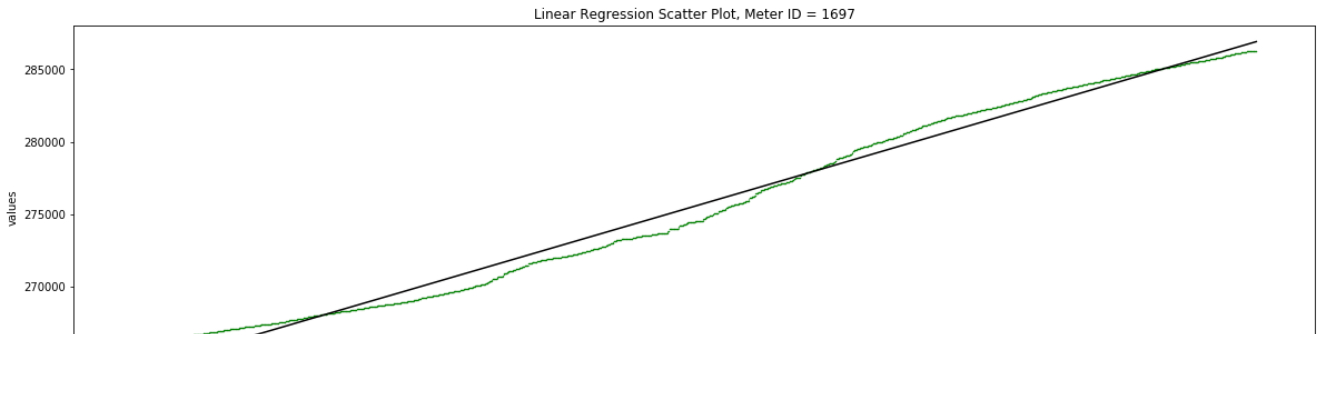

Linear Regression Scatter Plot, Meter ID = 3577

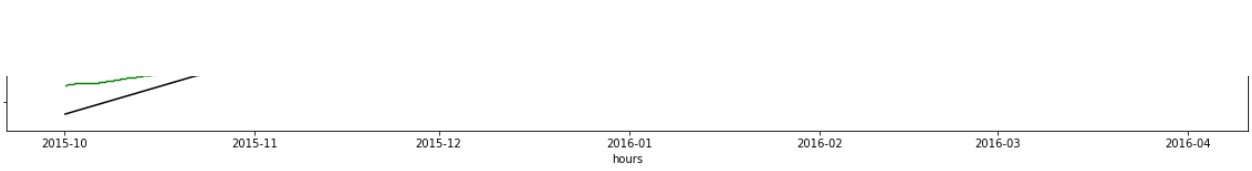

values
190000
185000
180000
175000
170000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

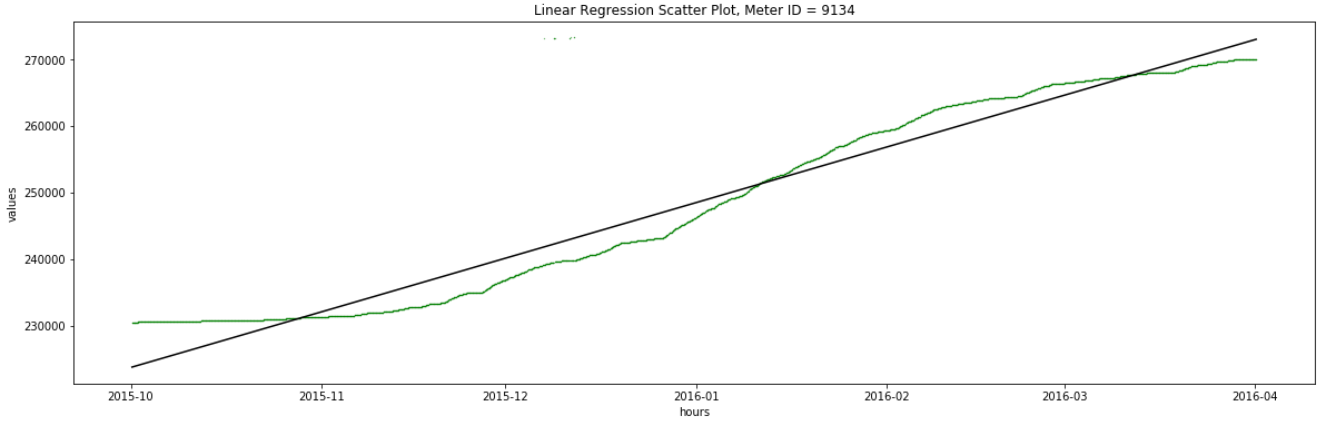

Linear Regression Scatter Plot, Meter ID = 222
values
680000
660000
640000
620000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

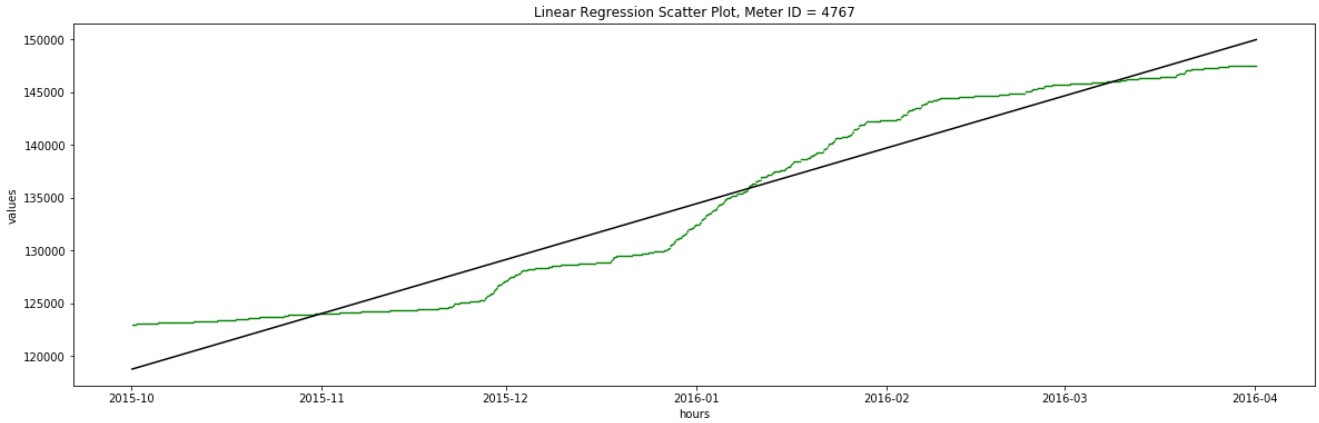

Linear Regression Scatter Plot, Meter ID = 2449
values
190000
185000
180000
175000
170000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

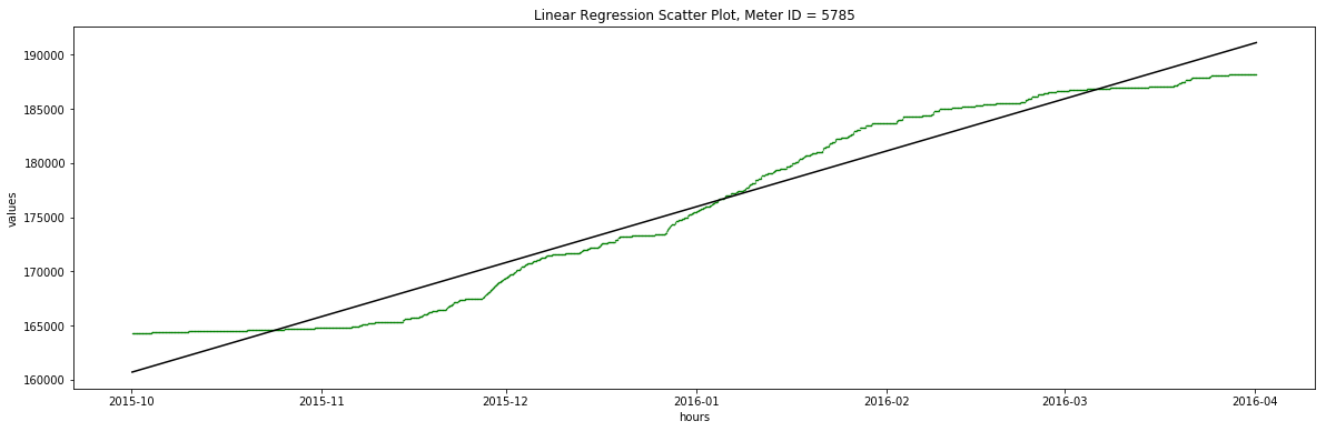

Linear Regression Scatter Plot, Meter ID = 35
values
104000
102000
100000
98000
96000
94000
92000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

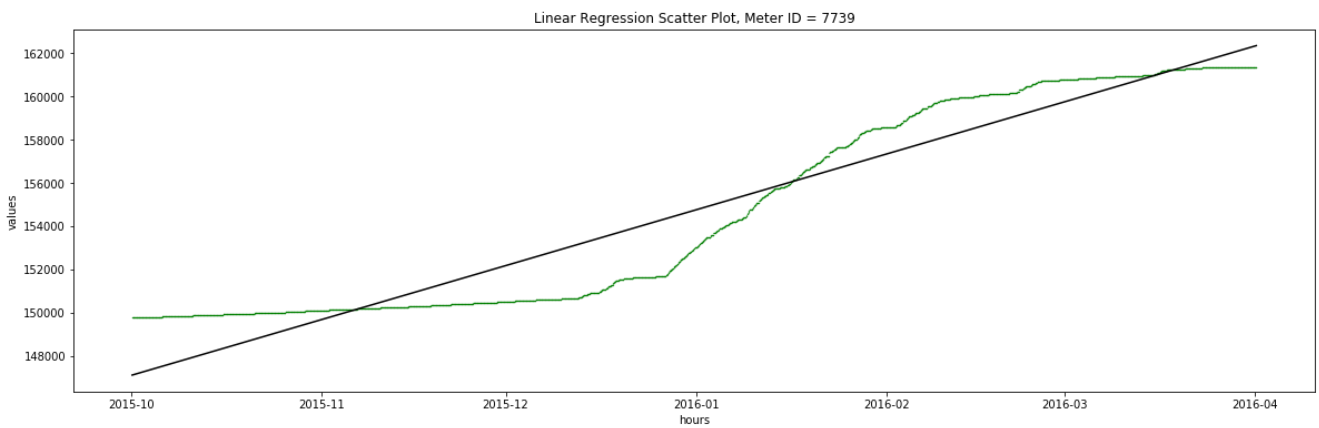

Linear Regression Scatter Plot, Meter ID = 1697
values
285000
280000
275000
270000

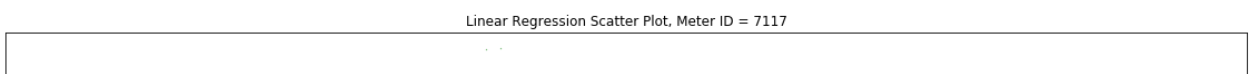

265000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

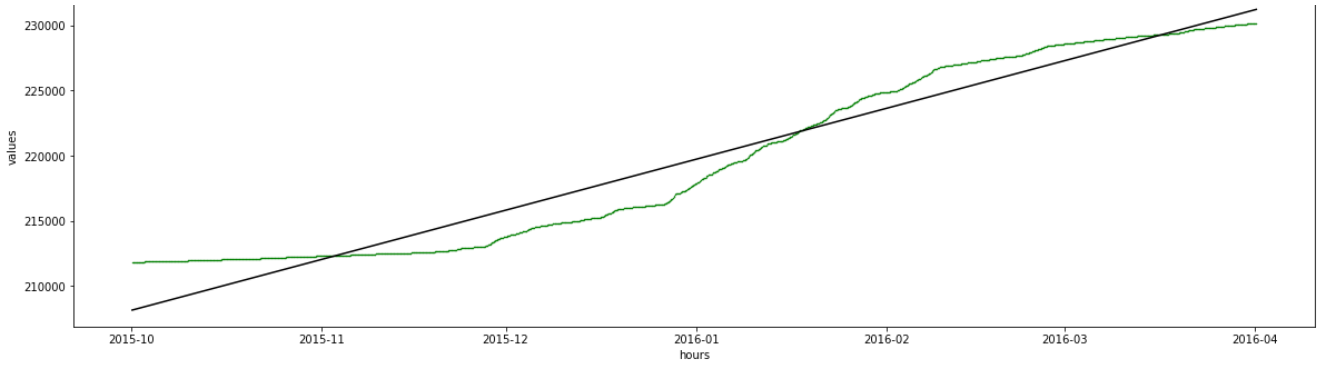

Linear Regression Scatter Plot, Meter ID = 9134
values
270000
260000
250000
240000
230000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

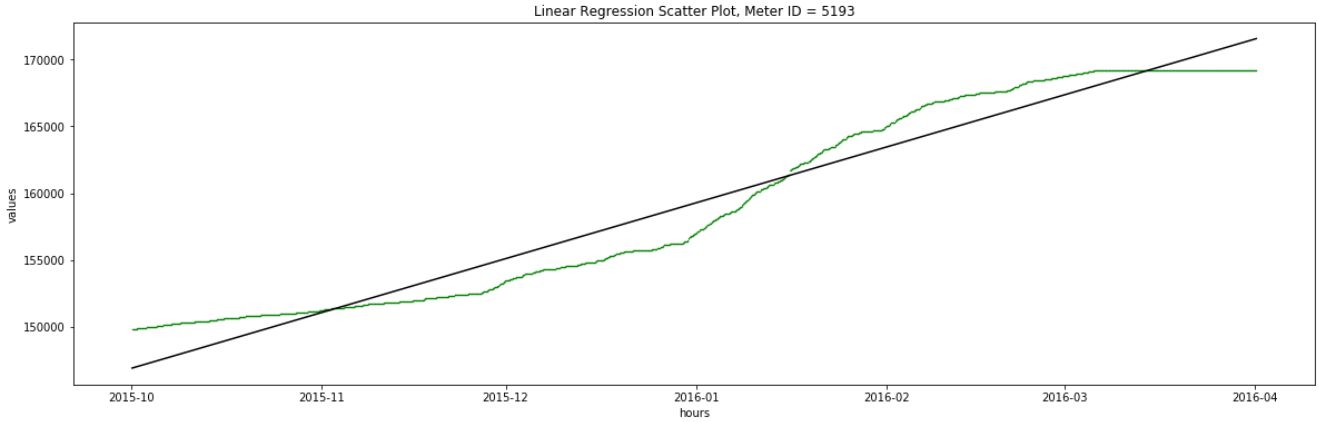

Linear Regression Scatter Plot, Meter ID = 4767
values
150000
145000
140000
135000
130000
125000
120000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

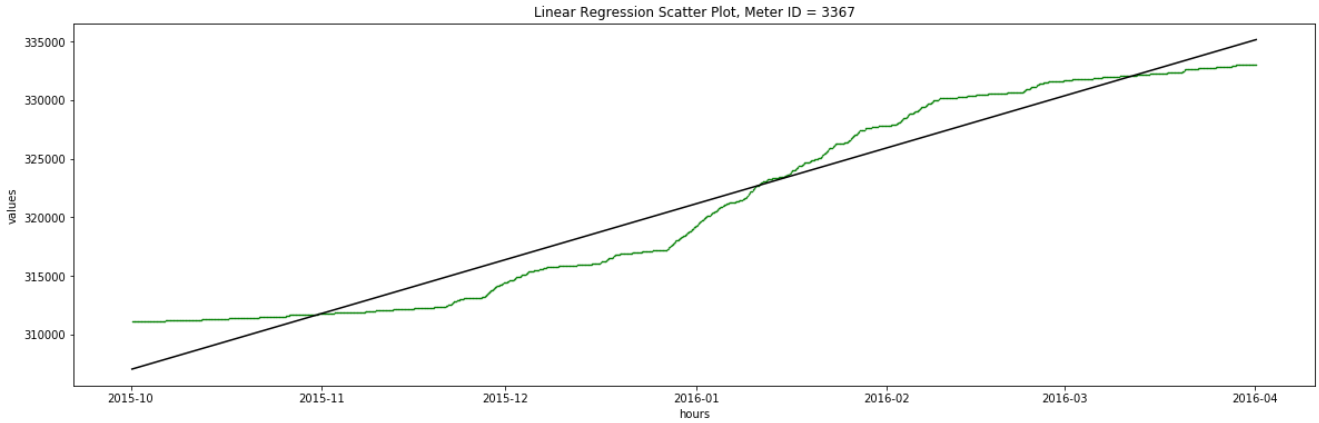

Linear Regression Scatter Plot, Meter ID = 5785
values
190000
185000
180000
175000
170000
165000
160000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

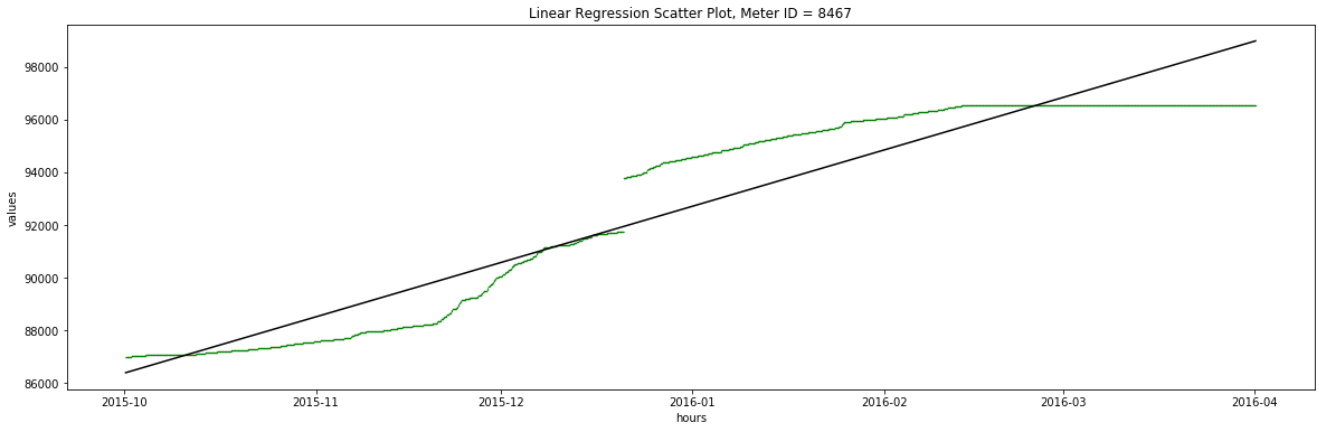

Linear Regression Scatter Plot, Meter ID = 7739
values
162000
160000
158000
156000
154000
152000
150000
148000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

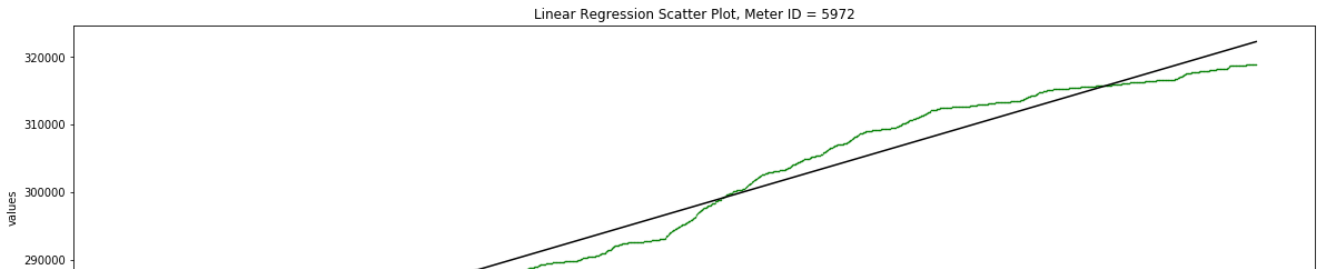

Linear Regression Scatter Plot, Meter ID = 7117


values
hours
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

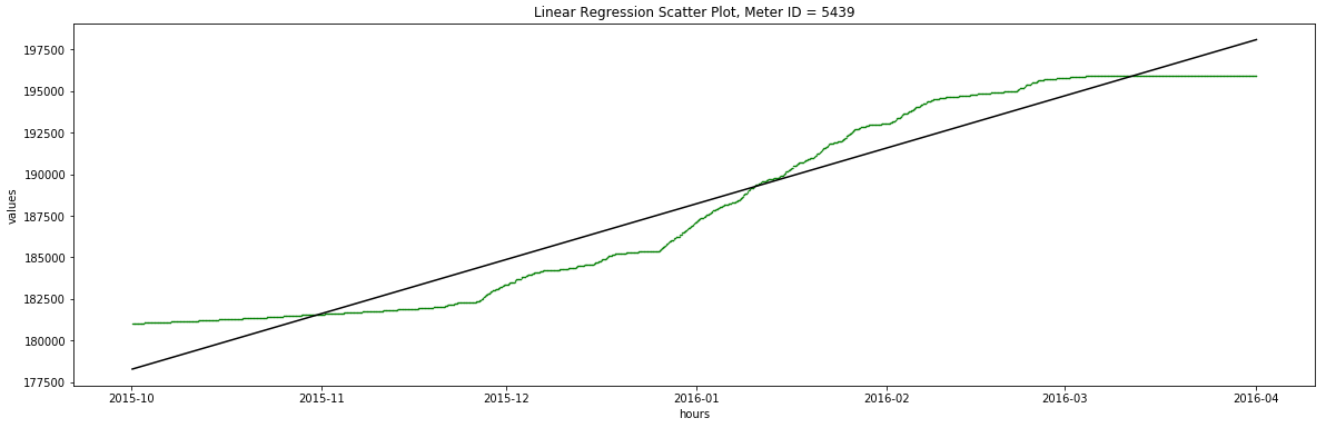

Linear Regression Scatter Plot, Meter ID = 5193
values
hours

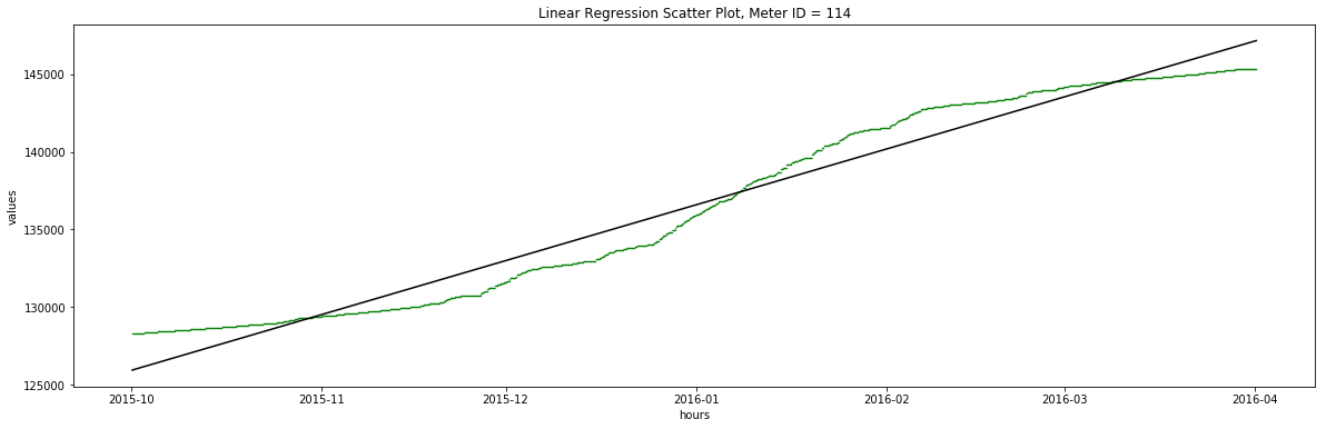

Linear Regression Scatter Plot, Meter ID = 3367
values
hours

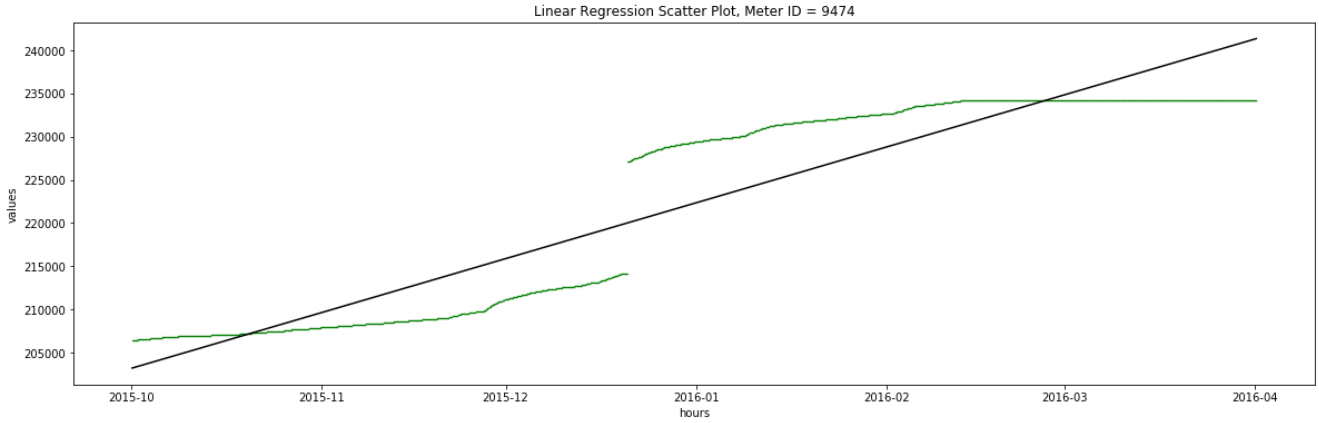

Linear Regression Scatter Plot, Meter ID = 8467
values
hours

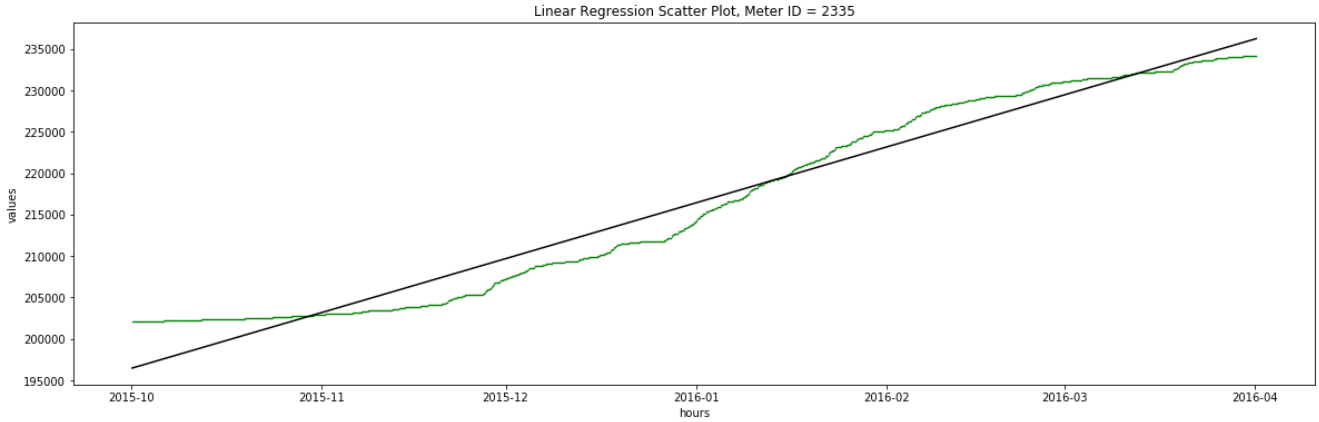

Linear Regression Scatter Plot, Meter ID = 5972
values

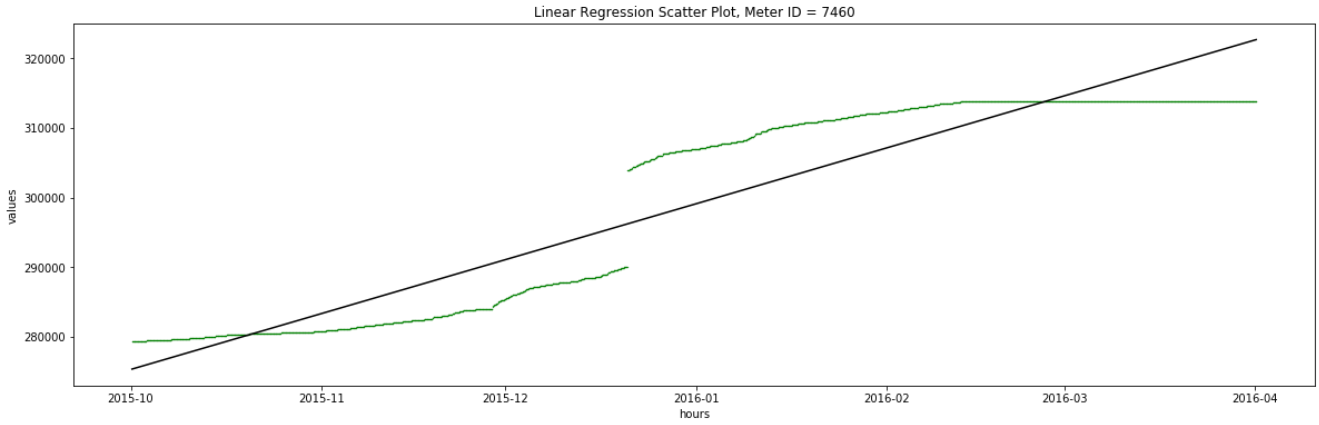

280000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

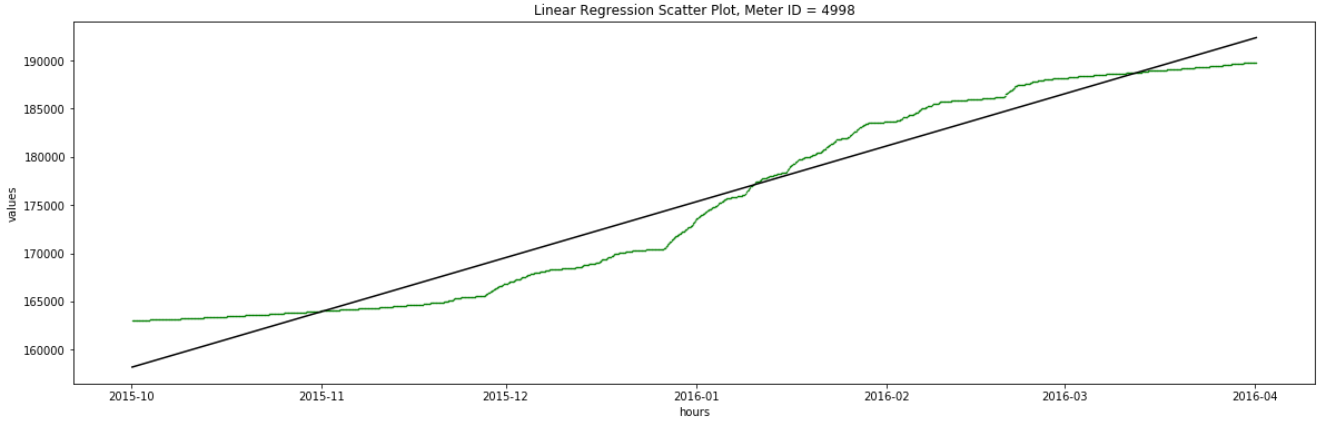

Linear Regression Scatter Plot, Meter ID = 5439
values
197500
195000
192500
190000
187500
185000
182500
180000
177500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

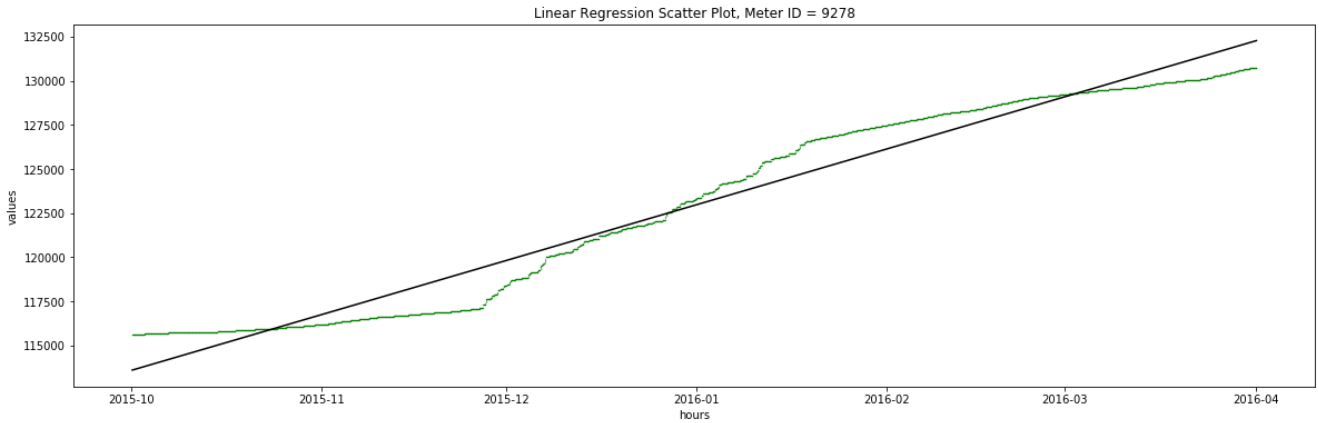

Linear Regression Scatter Plot, Meter ID = 114
values
145000
140000
135000
130000
125000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

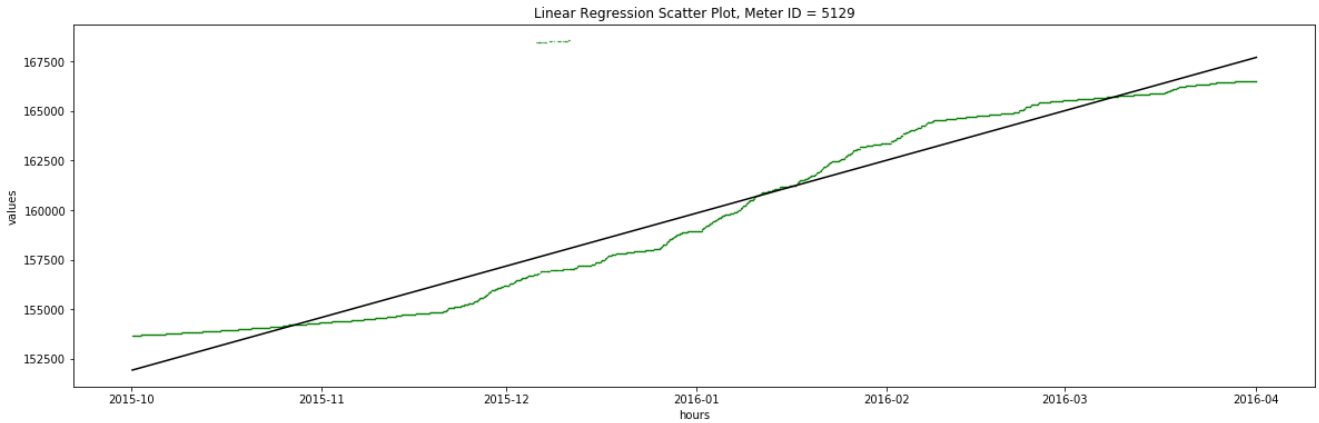

Linear Regression Scatter Plot, Meter ID = 9474
values
240000
235000
230000
225000
220000
215000
210000
205000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

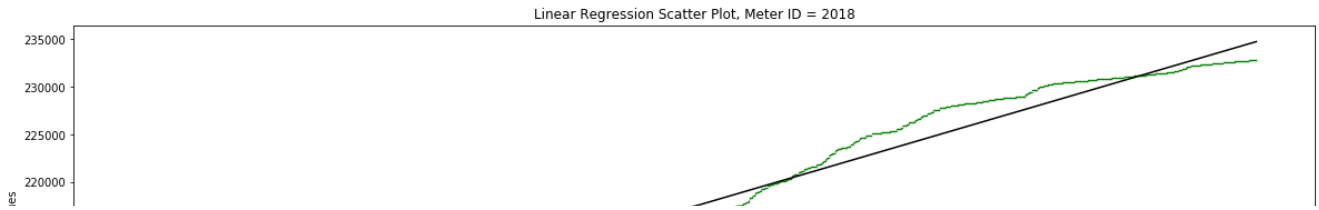

Linear Regression Scatter Plot, Meter ID = 2335
values
235000
230000
225000
220000
215000
210000
205000
200000
195000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

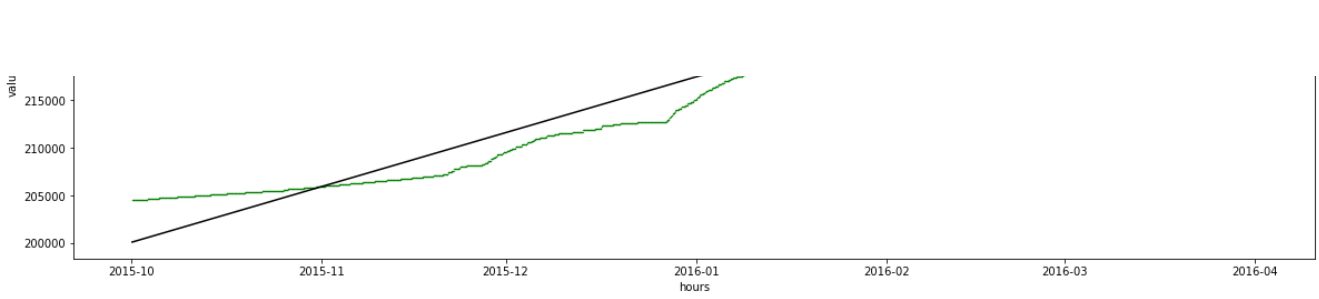

Linear Regression Scatter Plot, Meter ID = 7460
values
hours
320000
310000
300000
290000
280000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

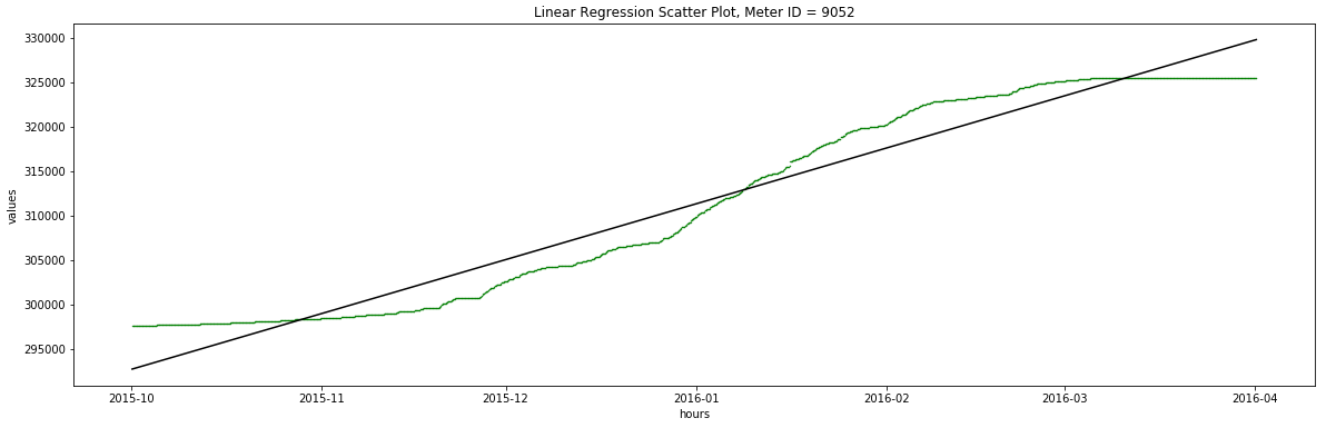

Linear Regression Scatter Plot, Meter ID = 4998
values
hours
190000
185000
180000
175000
170000
165000
160000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

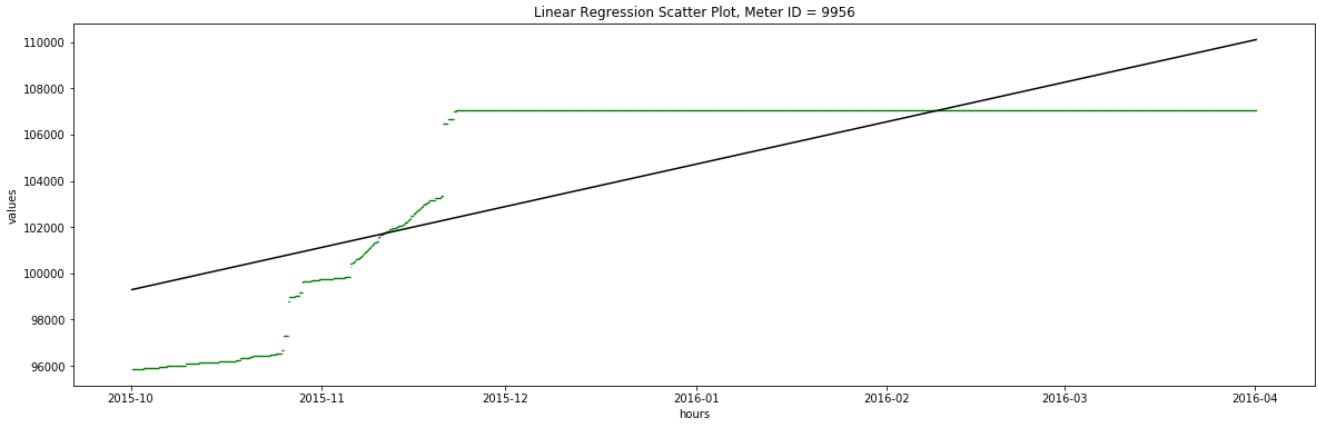

Linear Regression Scatter Plot, Meter ID = 9278
values
hours
132500
130000
127500
125000
122500
120000
117500
115000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

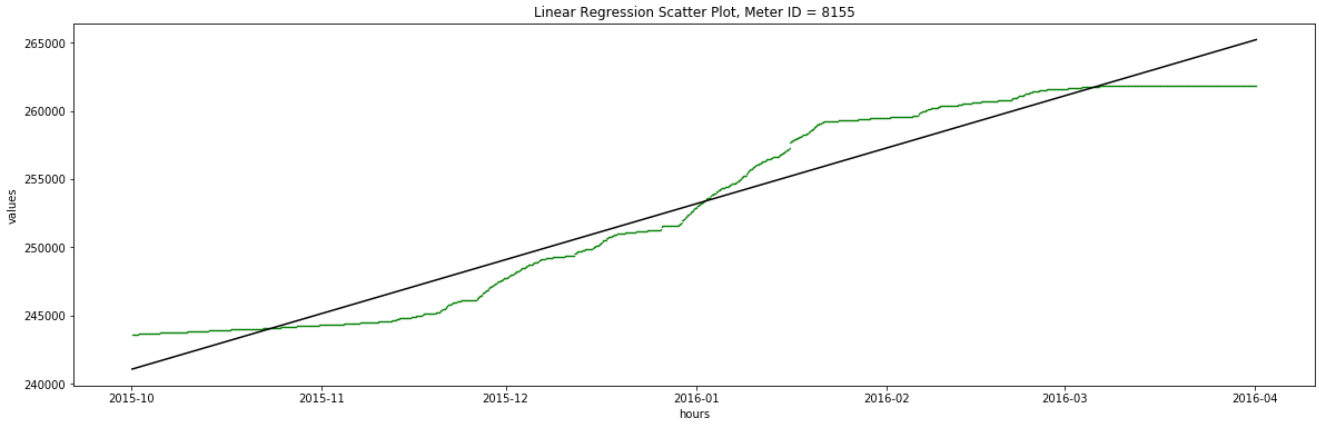

Linear Regression Scatter Plot, Meter ID = 5129
values
hours
167500
165000
162500
160000
157500
155000
152500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

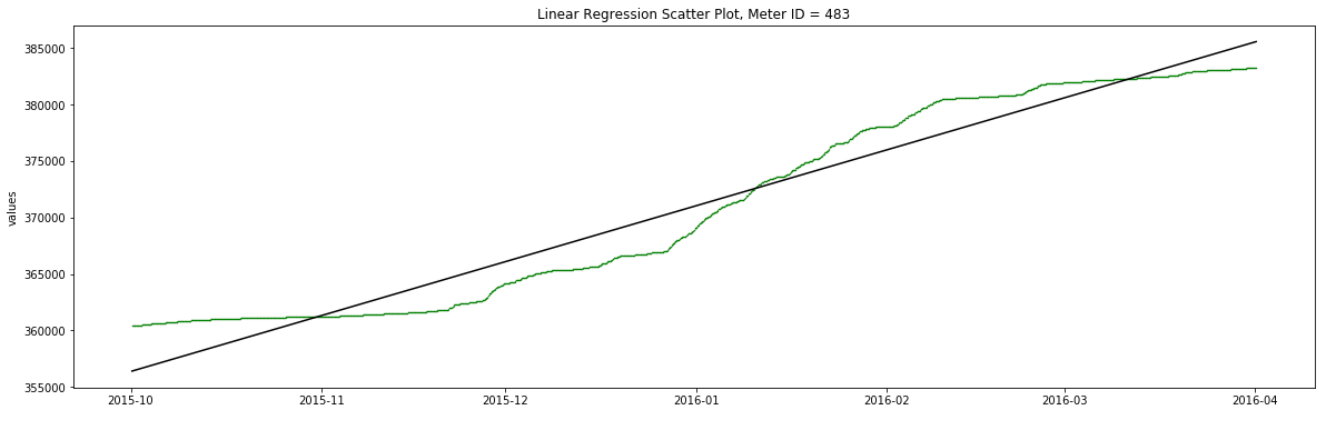

Linear Regression Scatter Plot, Meter ID = 2018
235000
230000
225000
220000


valu
215000
210000
205000
200000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

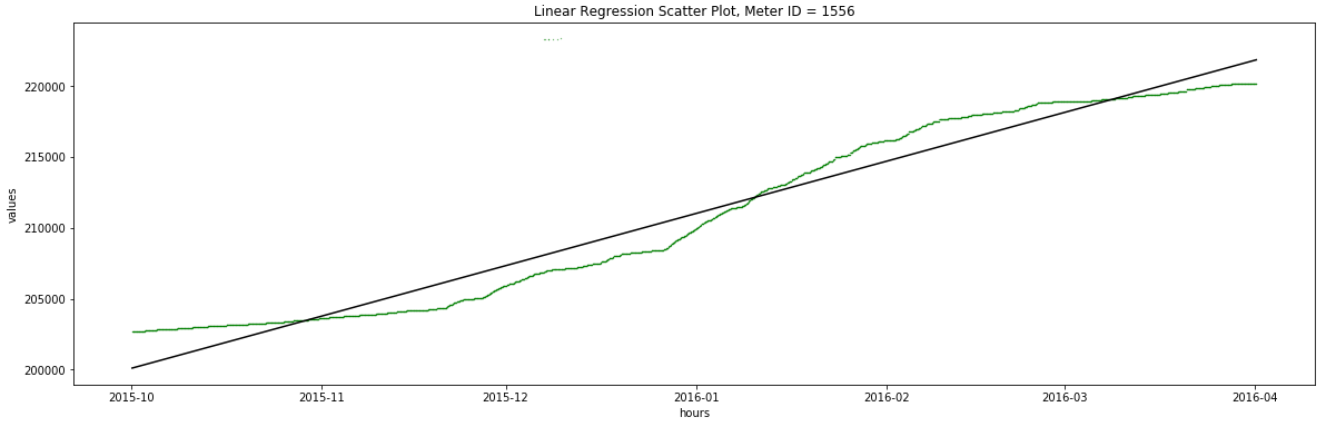

Linear Regression Scatter Plot, Meter ID = 9052
values
330000
325000
320000
315000
310000
305000
300000
295000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

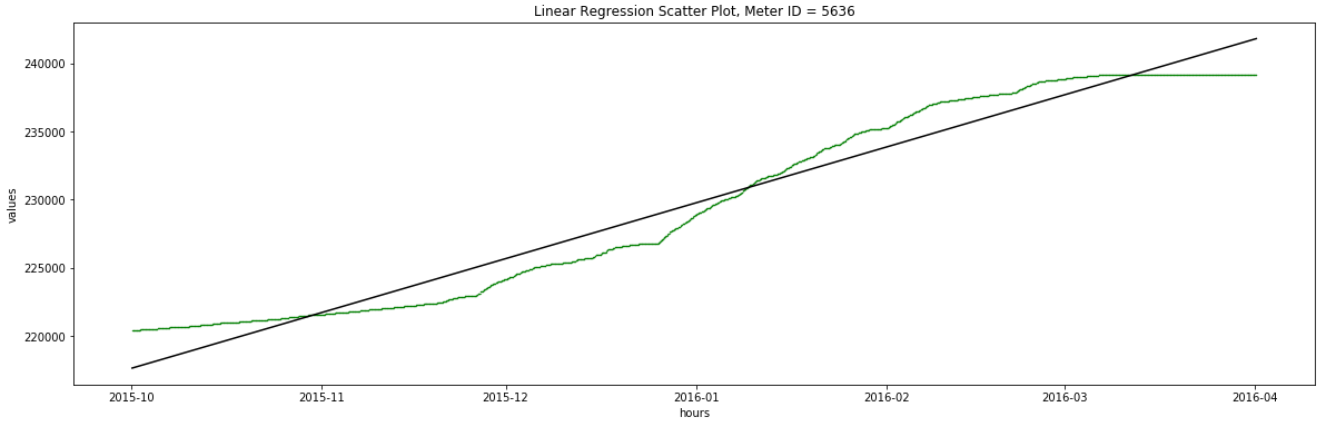

Linear Regression Scatter Plot, Meter ID = 9956
values
110000
108000
106000
104000
102000
100000
98000
96000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

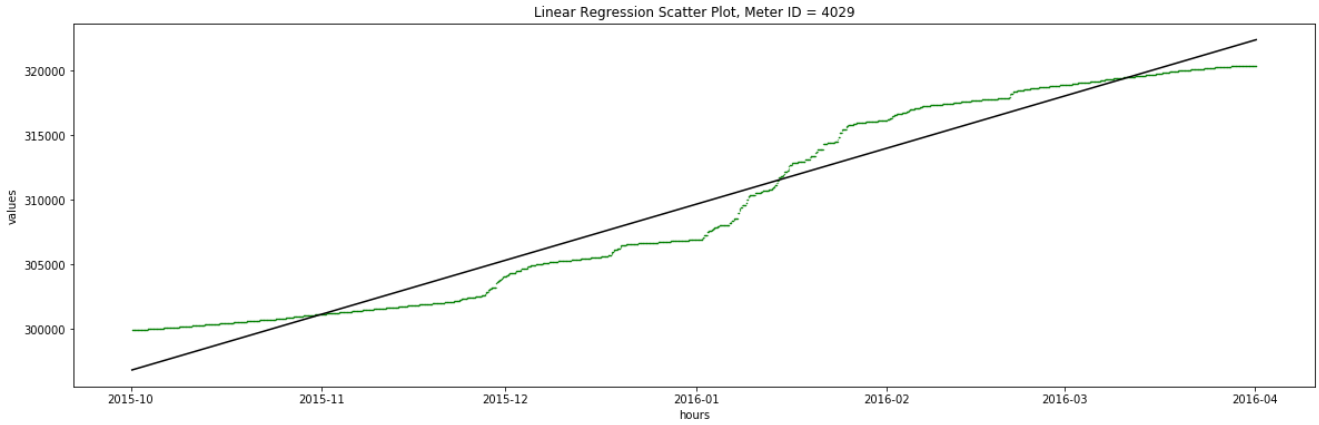

Linear Regression Scatter Plot, Meter ID = 8155
values
265000
260000
255000
250000
245000
240000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

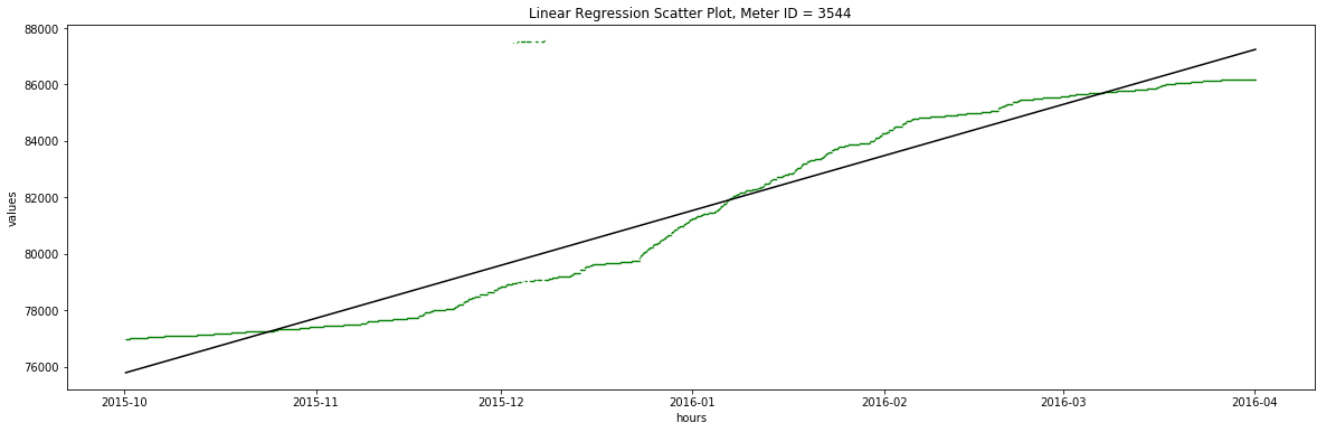

Linear Regression Scatter Plot, Meter ID = 483
values
385000
380000
375000
370000
365000
360000
355000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

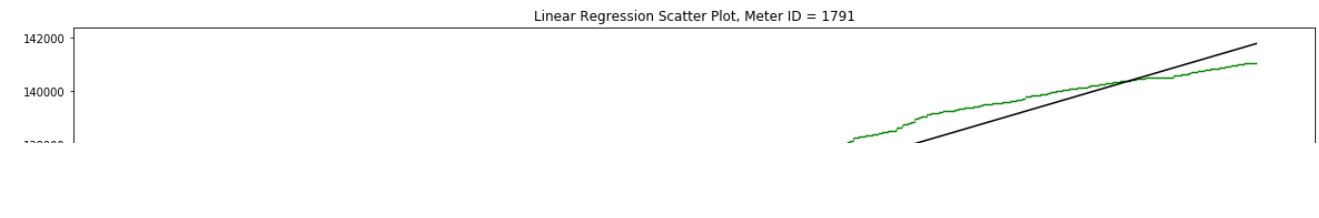
hours

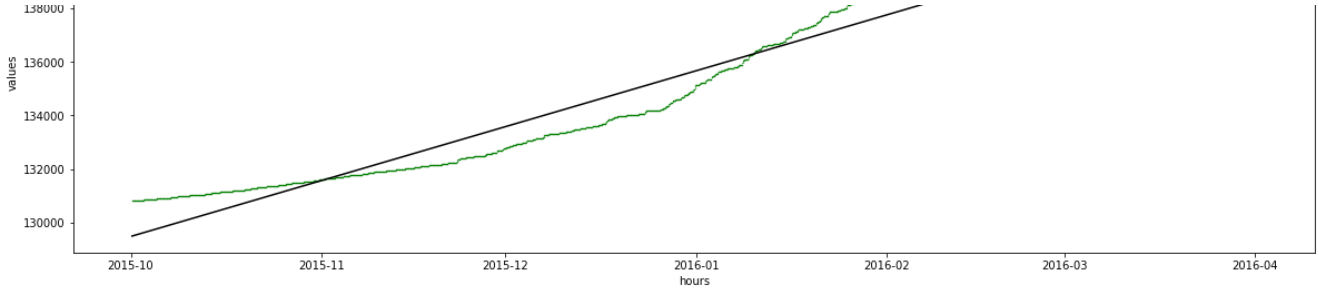

Linear Regression Scatter Plot, Meter ID = 1556
values
220000
215000
210000
205000
200000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

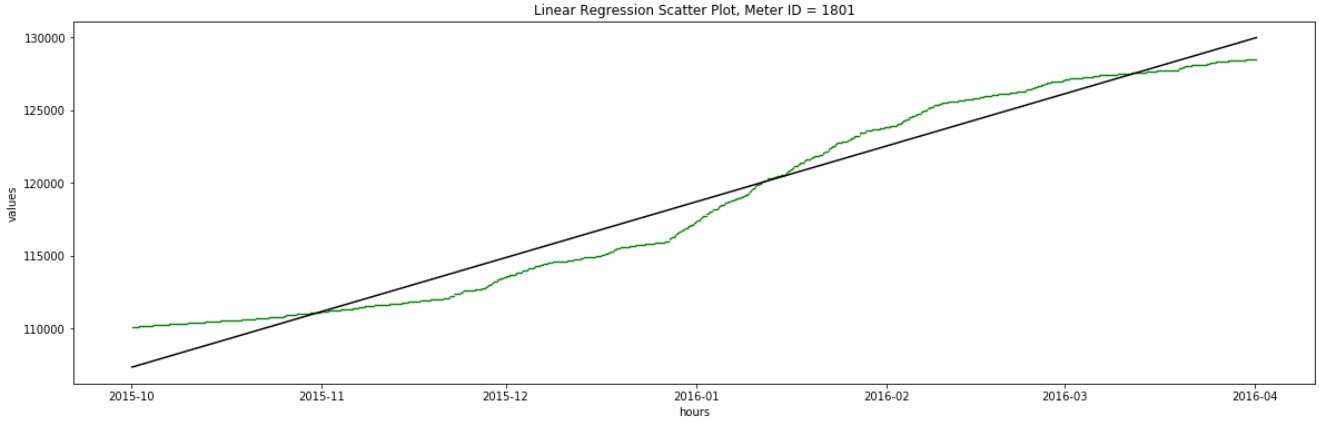

Linear Regression Scatter Plot, Meter ID = 5636
values
240000
235000
230000
225000
220000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

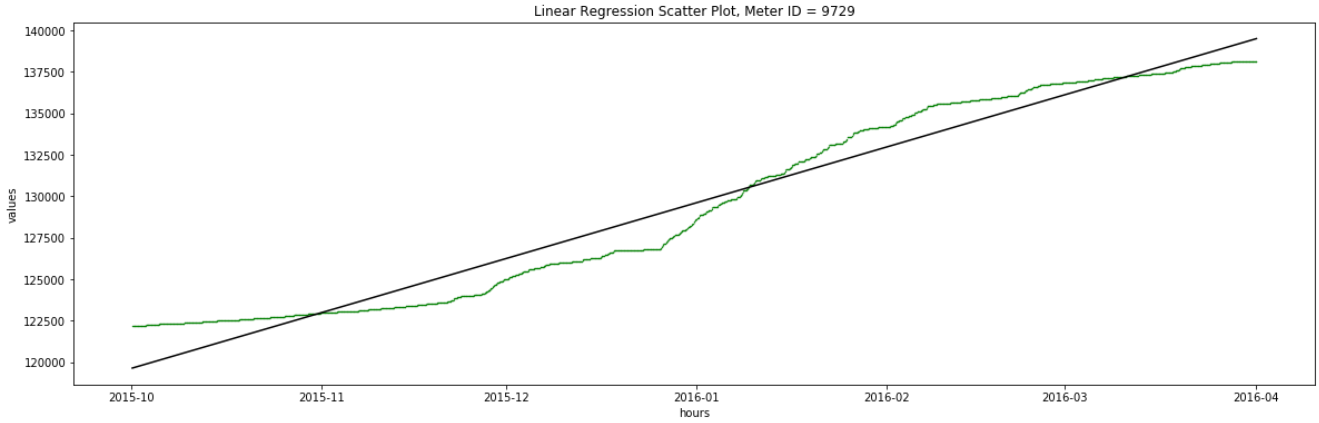

Linear Regression Scatter Plot, Meter ID = 4029
values
320000
315000
310000
305000
300000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

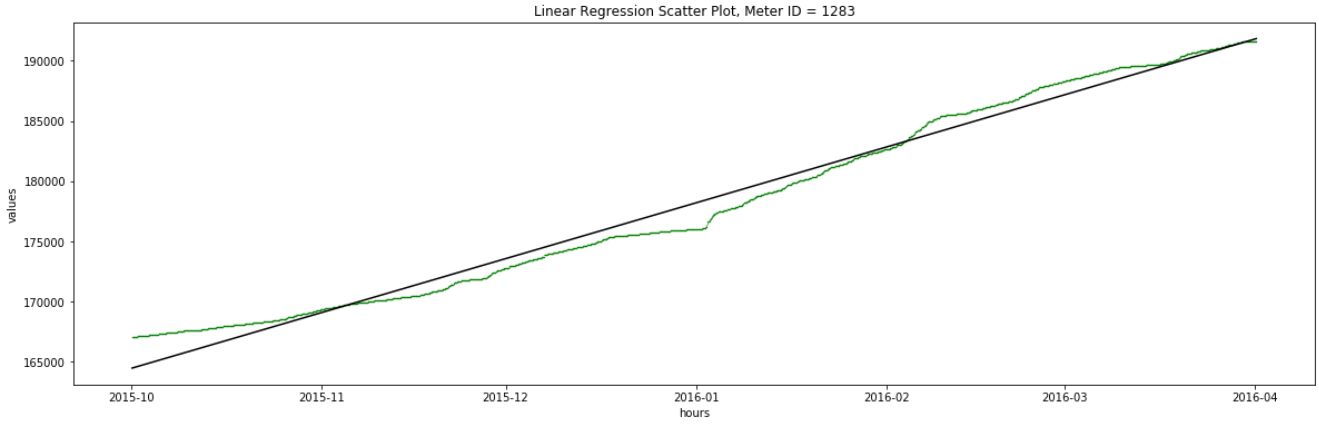

Linear Regression Scatter Plot, Meter ID = 3544
values
88000
86000
84000
82000
80000
78000
76000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

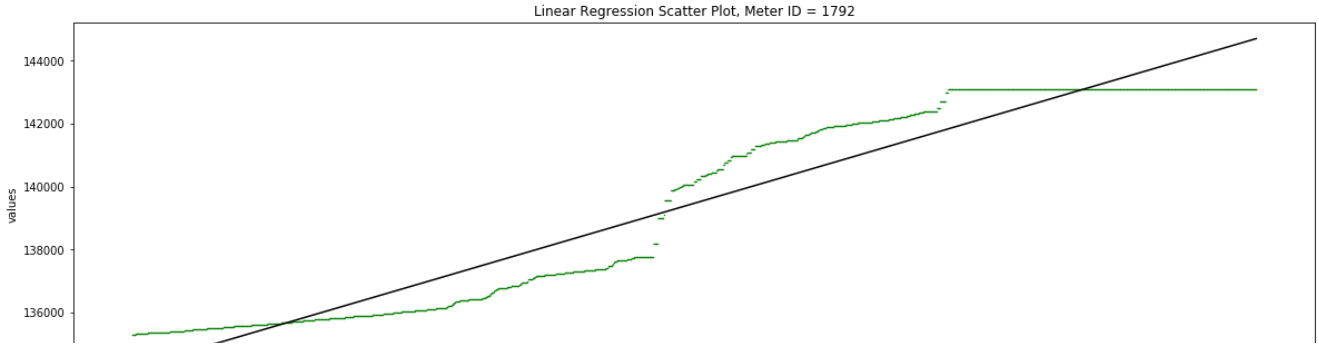

Linear Regression Scatter Plot, Meter ID = 1791
142000
140000

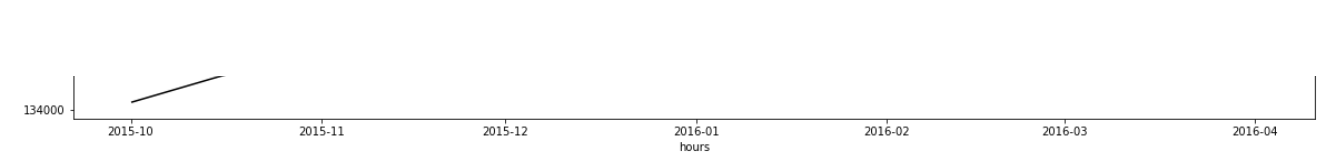

values
hours
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

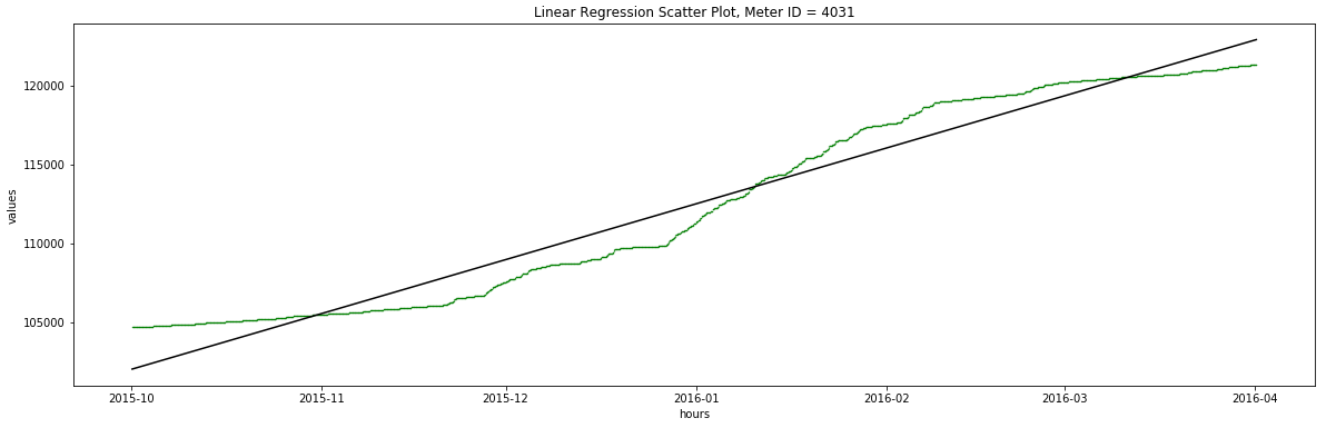

Linear Regression Scatter Plot, Meter ID = 1801
values
hours

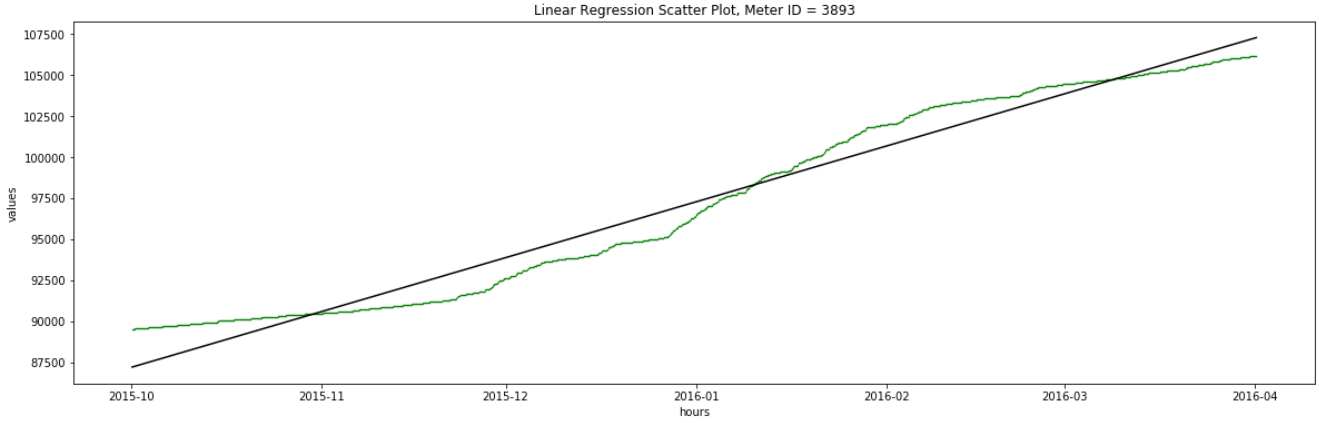

Linear Regression Scatter Plot, Meter ID = 9729
values
hours

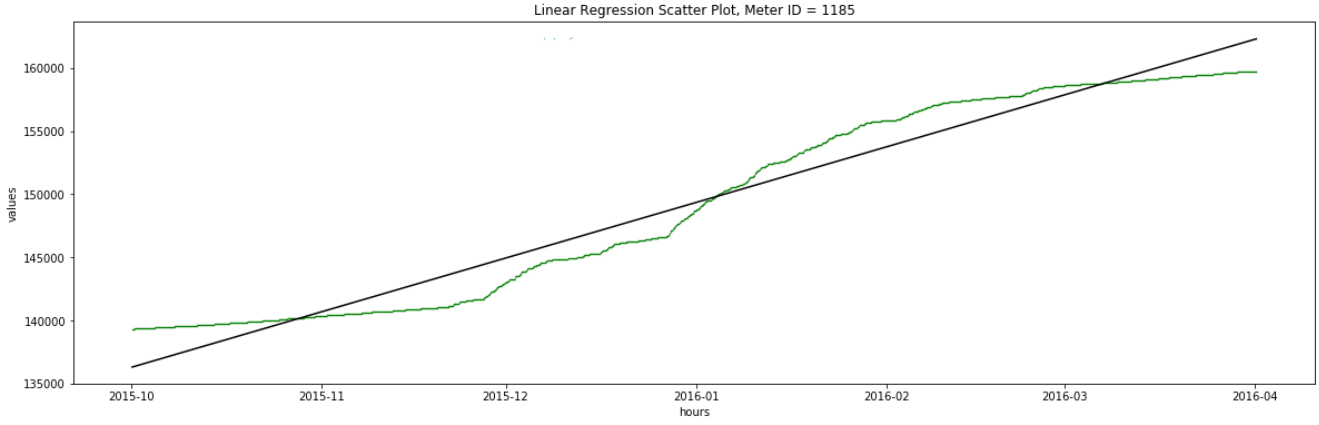

Linear Regression Scatter Plot, Meter ID = 1283
values
hours

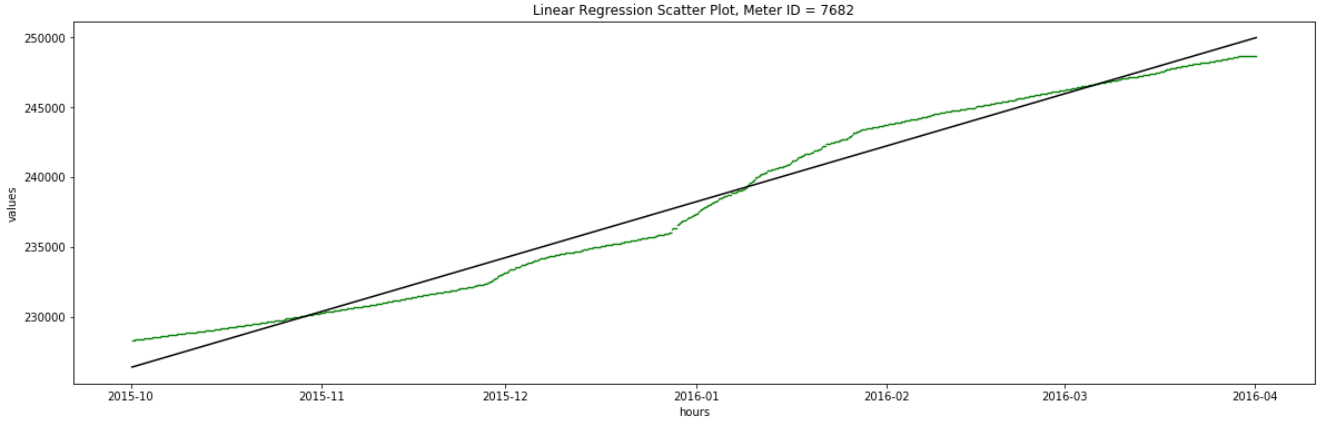

Linear Regression Scatter Plot, Meter ID = 1792
values

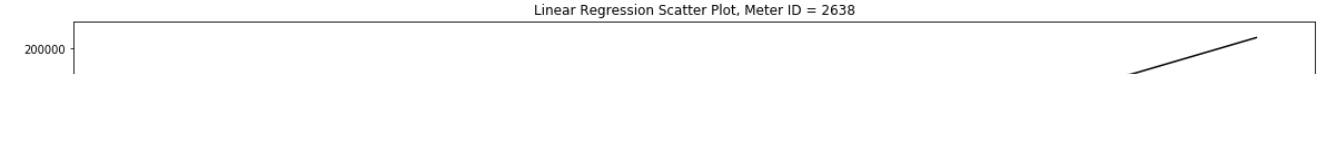

134000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

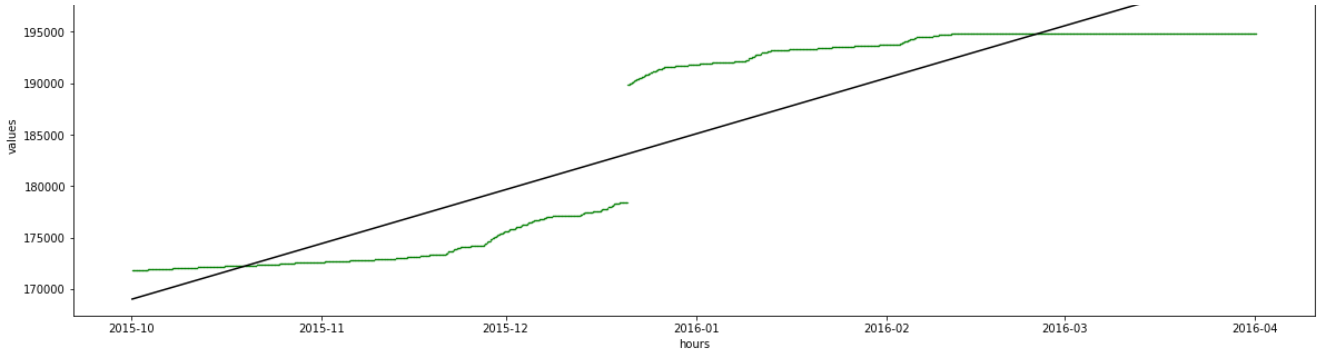

Linear Regression Scatter Plot, Meter ID = 4031
values
120000
115000
110000
105000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

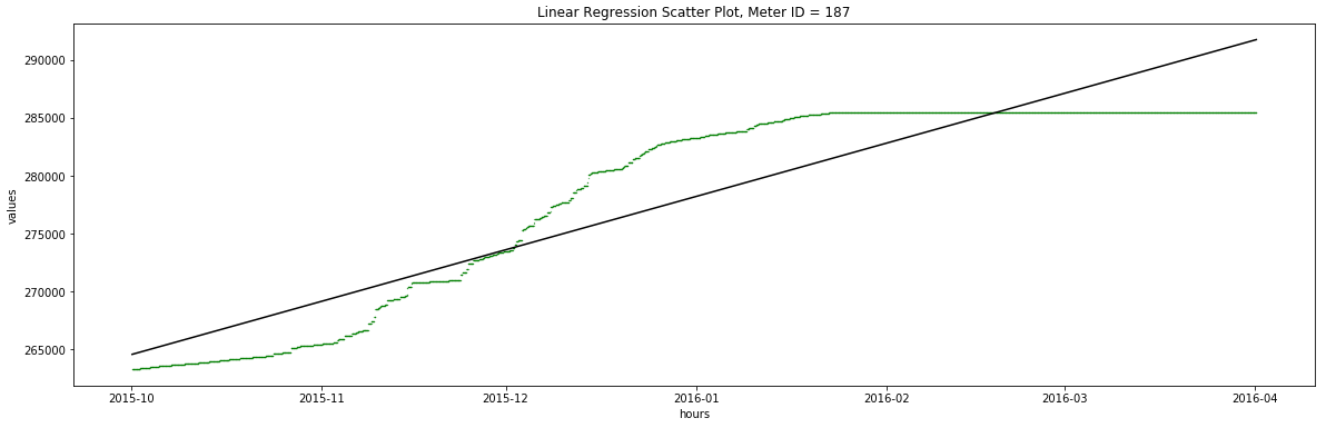

Linear Regression Scatter Plot, Meter ID = 3893
values
107500
105000
102500
100000
97500
95000
92500
90000
87500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

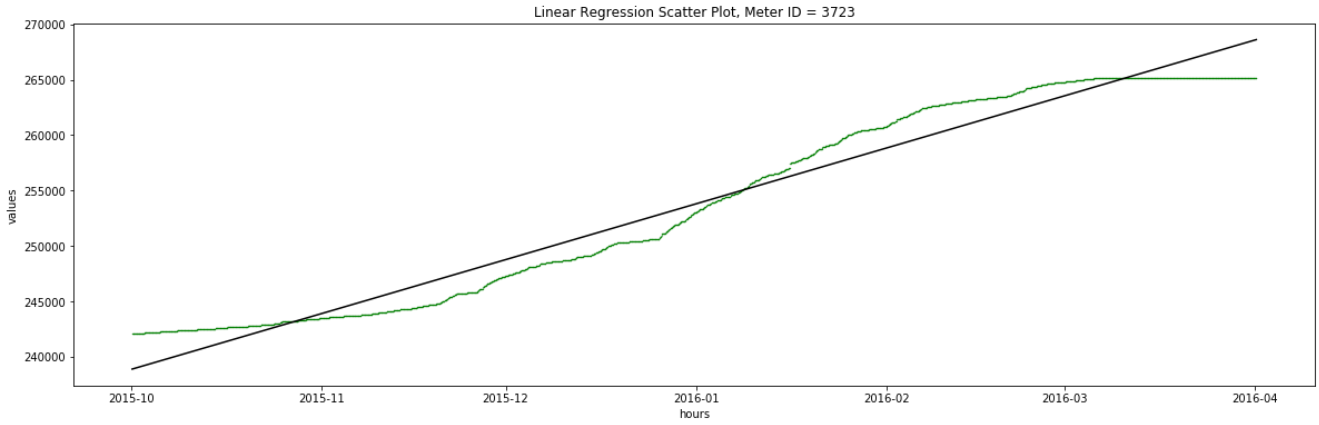

Linear Regression Scatter Plot, Meter ID = 1185
values
160000
155000
150000
145000
140000
135000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

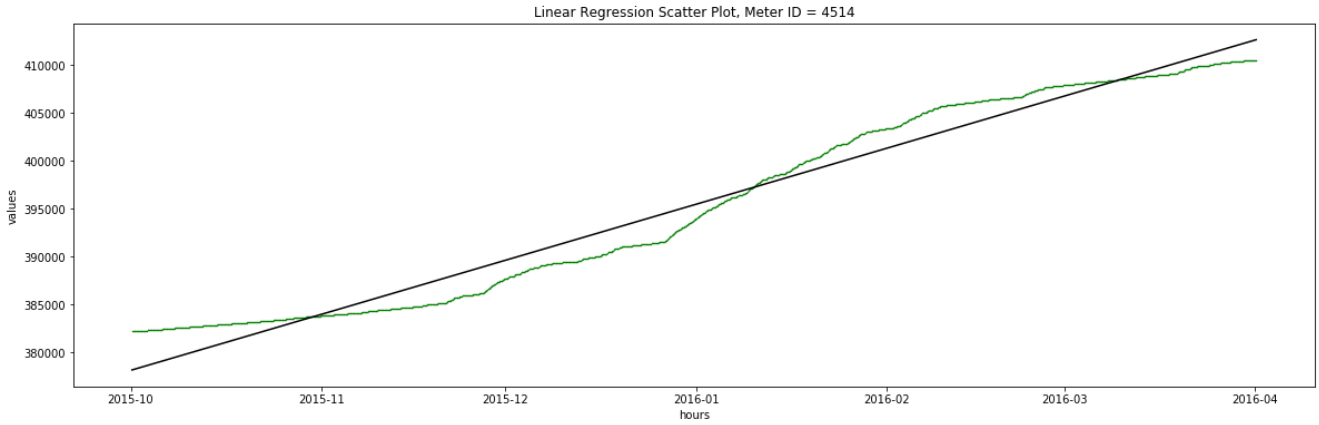

Linear Regression Scatter Plot, Meter ID = 7682
values
250000
245000
240000
235000
230000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

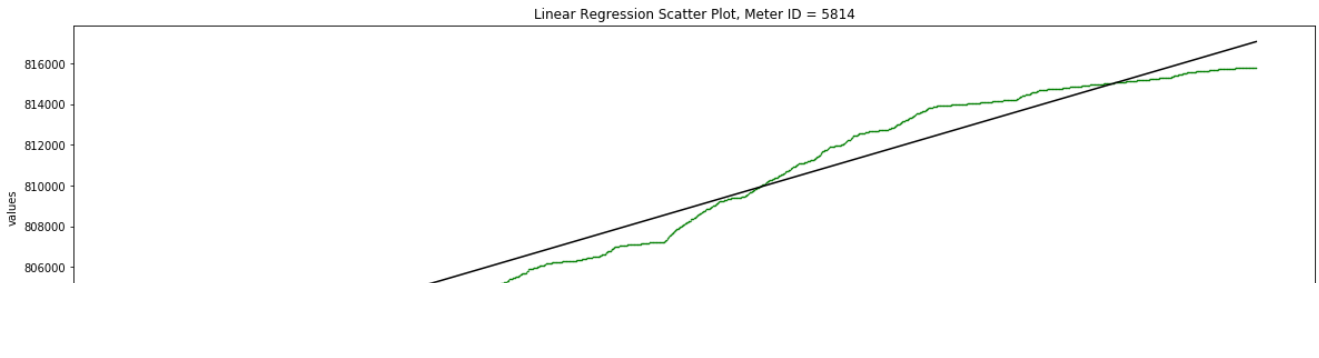

Linear Regression Scatter Plot, Meter ID = 2638
200000

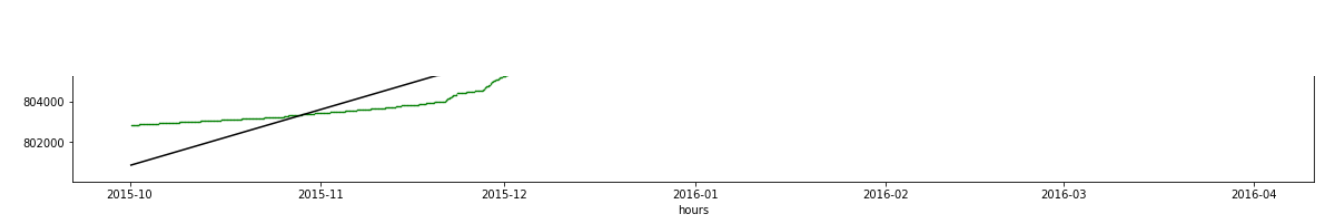

values
hours
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

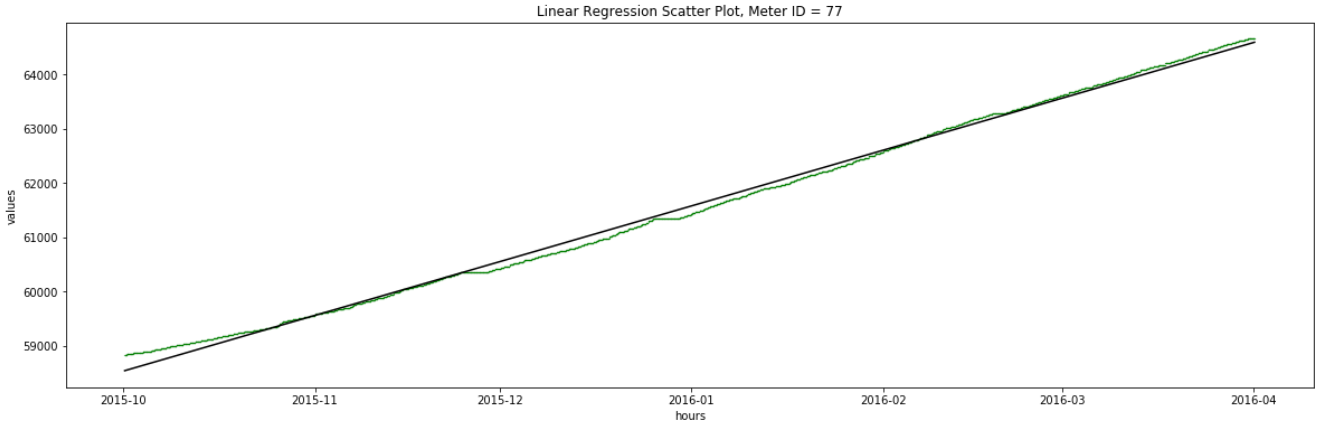

Linear Regression Scatter Plot, Meter ID = 187
values
hours

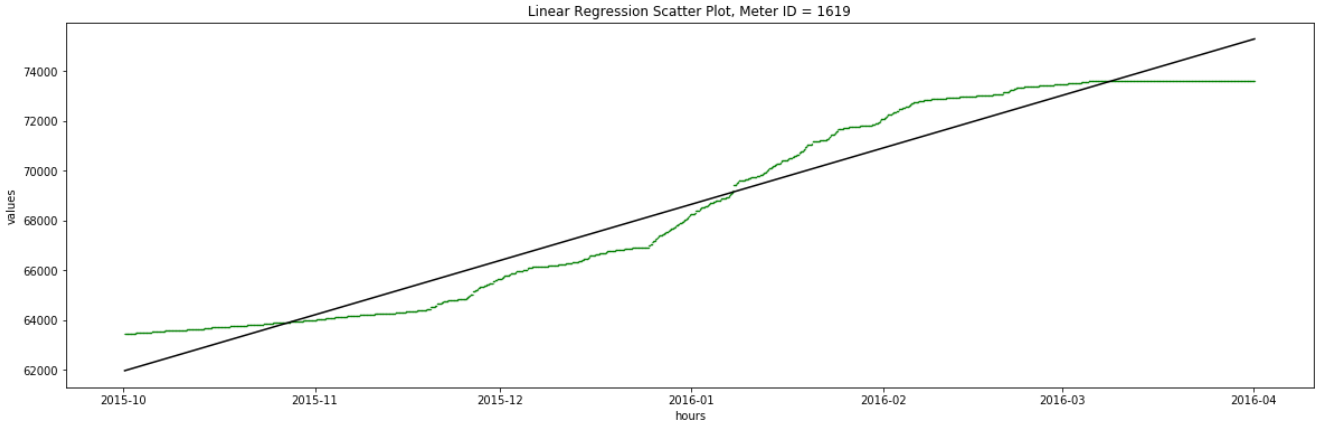

Linear Regression Scatter Plot, Meter ID = 3723
values
hours

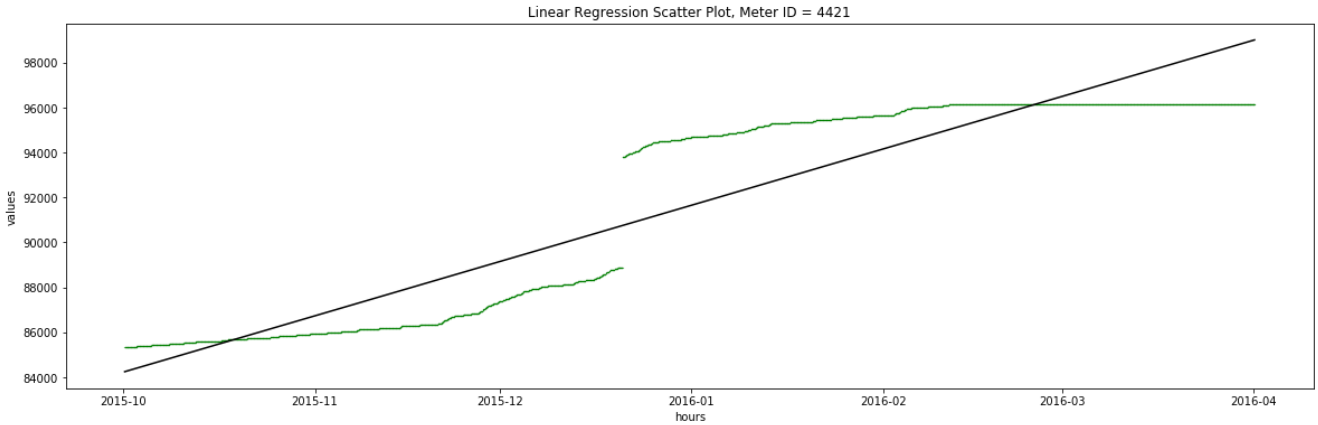

Linear Regression Scatter Plot, Meter ID = 4514
values
hours

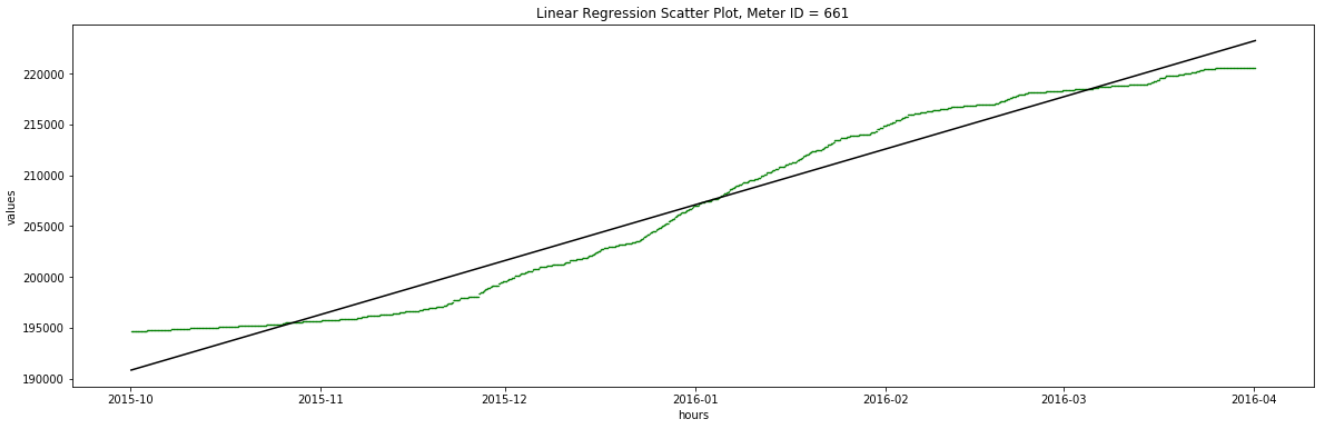

Linear Regression Scatter Plot, Meter ID = 5814
values

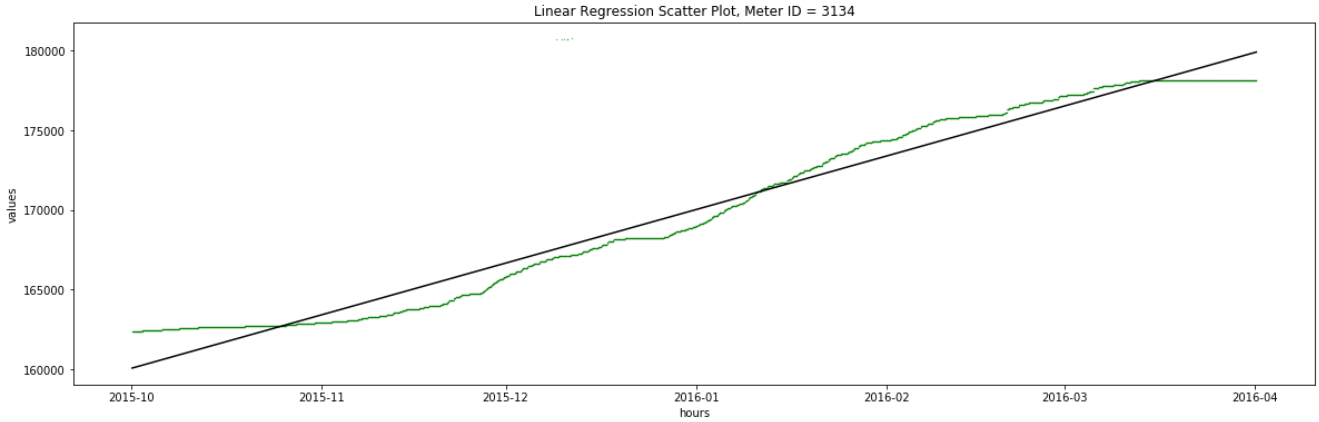

804000
802000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

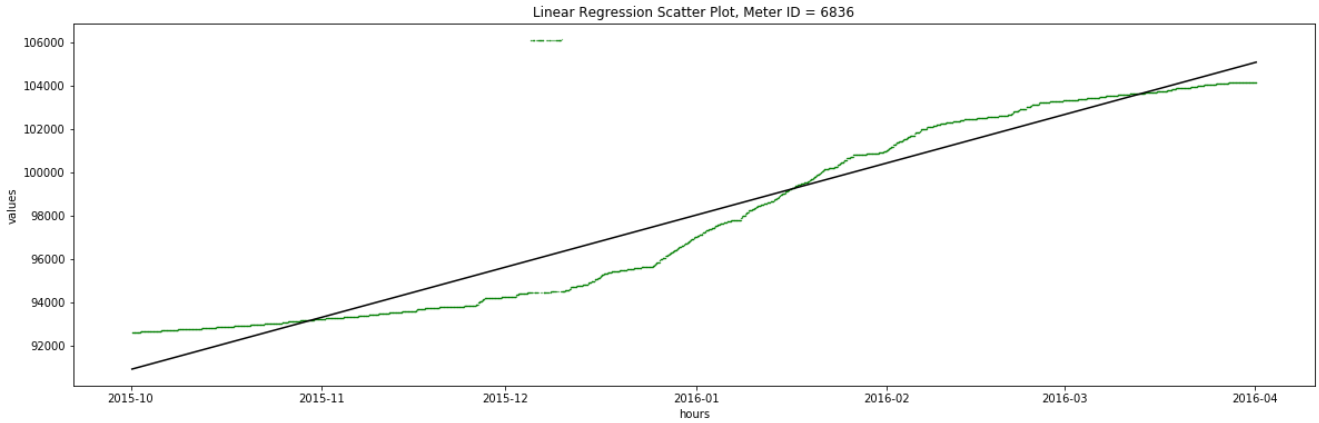

Linear Regression Scatter Plot, Meter ID = 77
64000
63000
62000
61000
60000
59000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

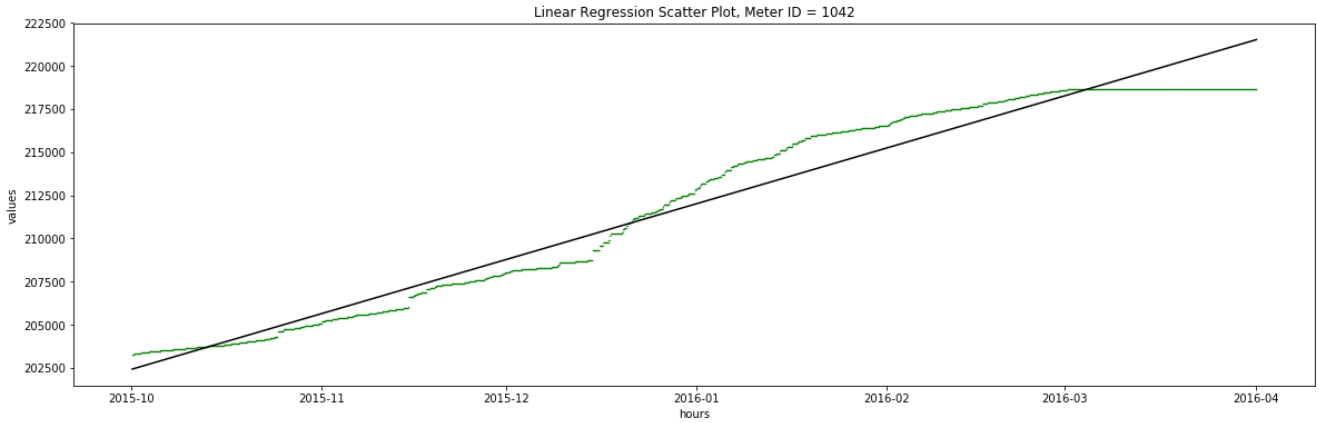

Linear Regression Scatter Plot, Meter ID = 1619
74000
72000
70000
68000
66000
64000
62000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

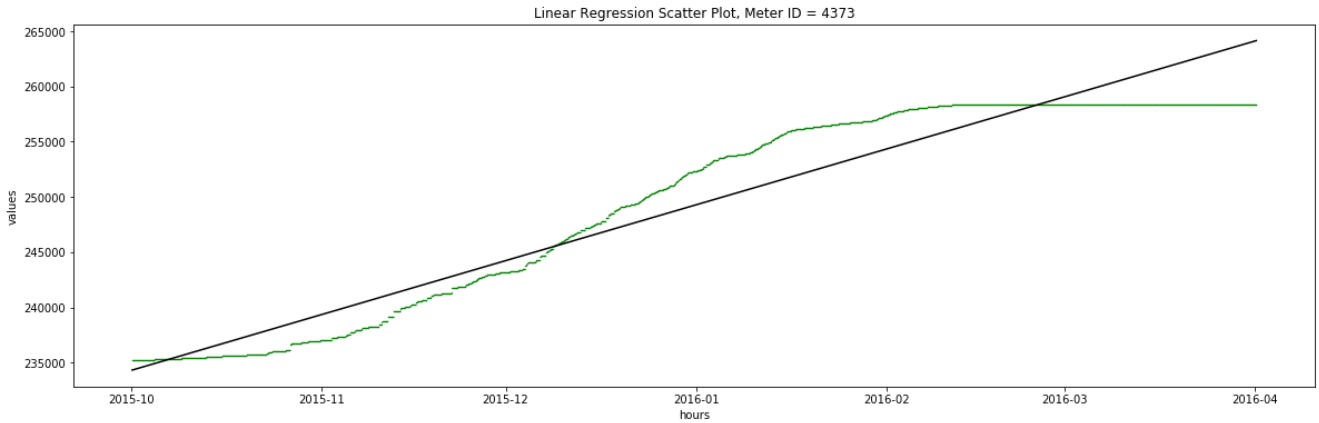

Linear Regression Scatter Plot, Meter ID = 4421
98000
96000
94000
92000
90000
88000
86000
84000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

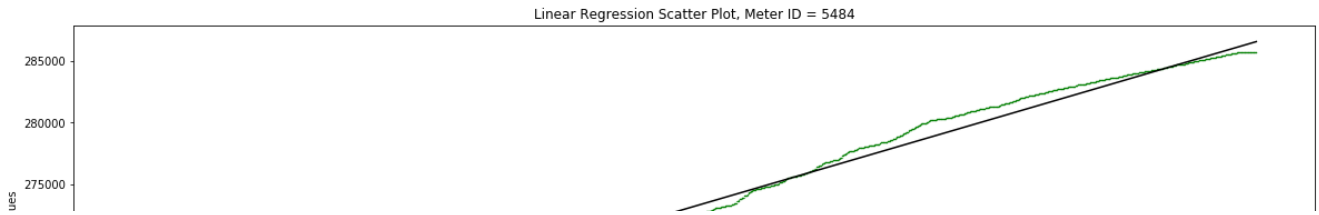

Linear Regression Scatter Plot, Meter ID = 661
220000
215000
210000
205000
200000
195000
190000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

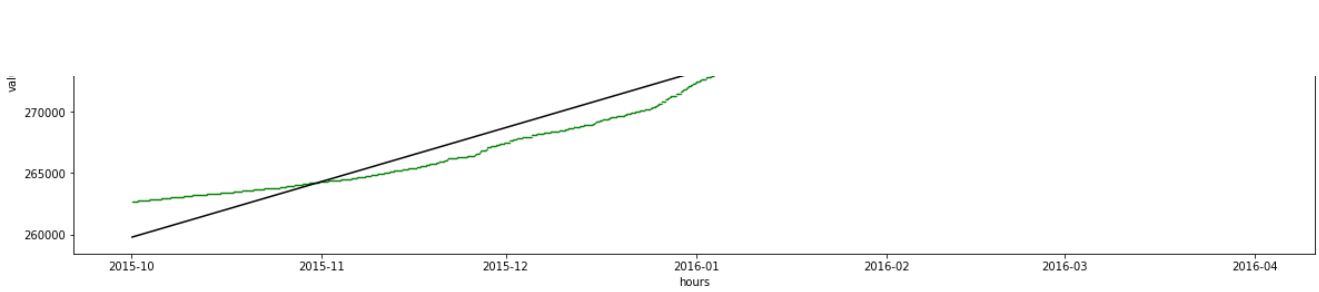

Linear Regression Scatter Plot, Meter ID = 3134
values
hours
180000
175000
170000
165000
160000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

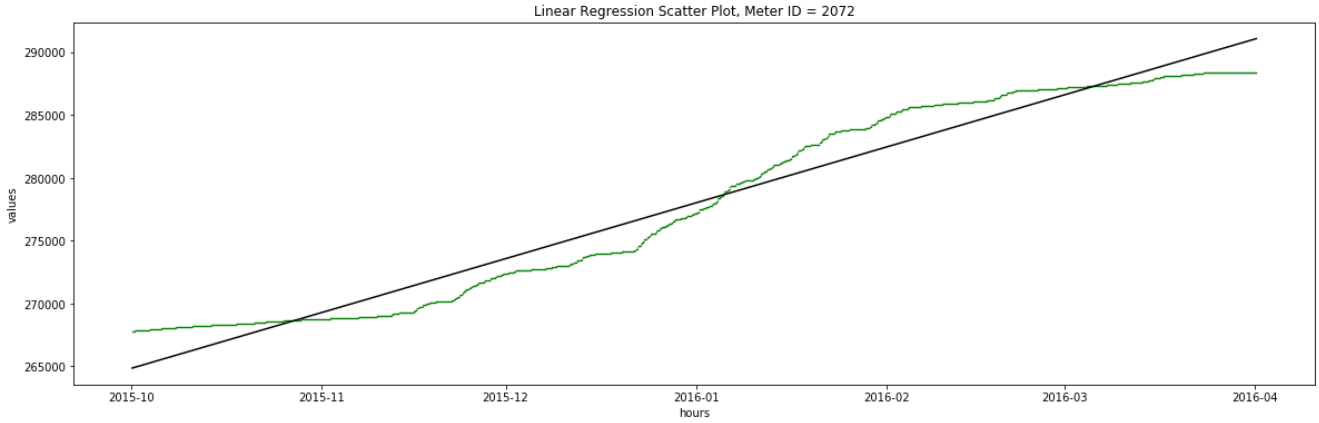

Linear Regression Scatter Plot, Meter ID = 6836
values
hours
106000
104000
102000
100000
98000
96000
94000
92000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

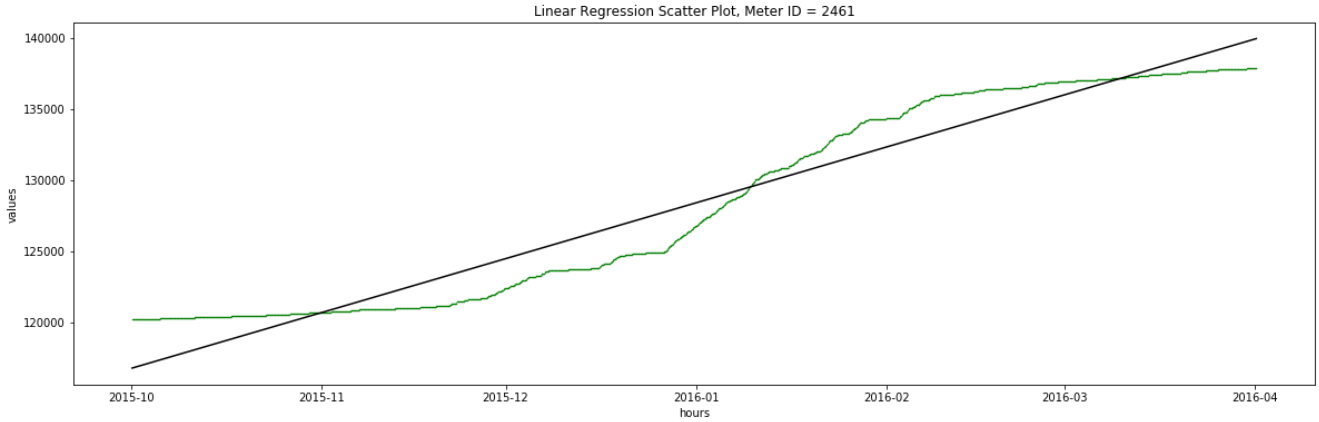

Linear Regression Scatter Plot, Meter ID = 1042
values
hours
222500
220000
217500
215000
212500
210000
207500
205000
202500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

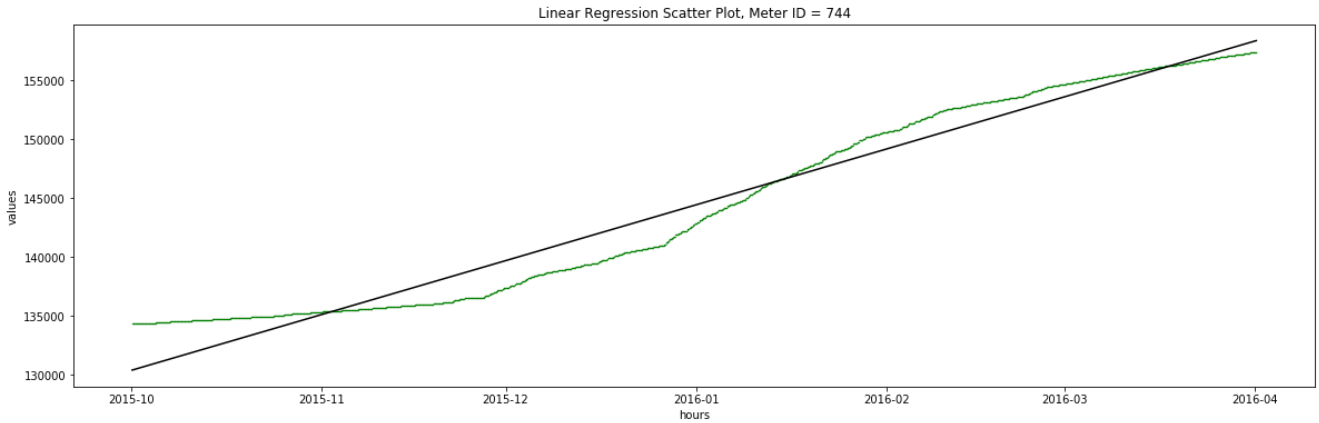

Linear Regression Scatter Plot, Meter ID = 4373
values
hours
265000
260000
255000
250000
245000
240000
235000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

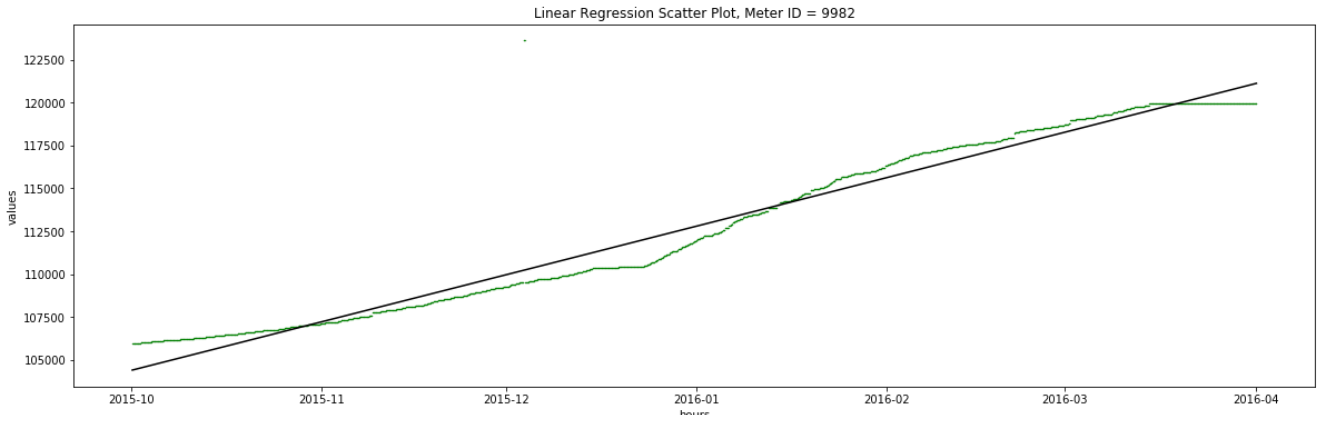

Linear Regression Scatter Plot, Meter ID = 5484
285000
280000
275000

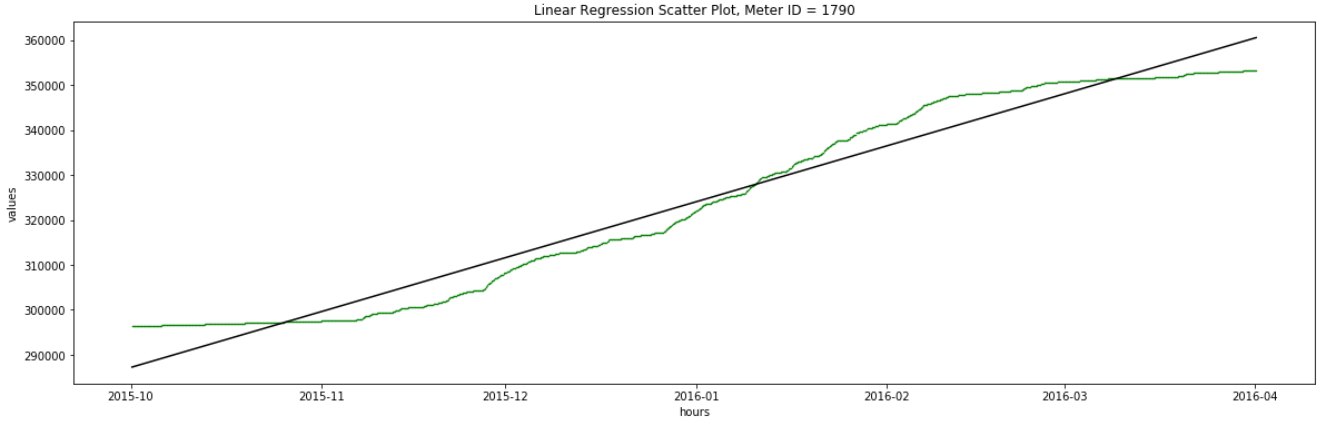

val
270000
265000
260000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

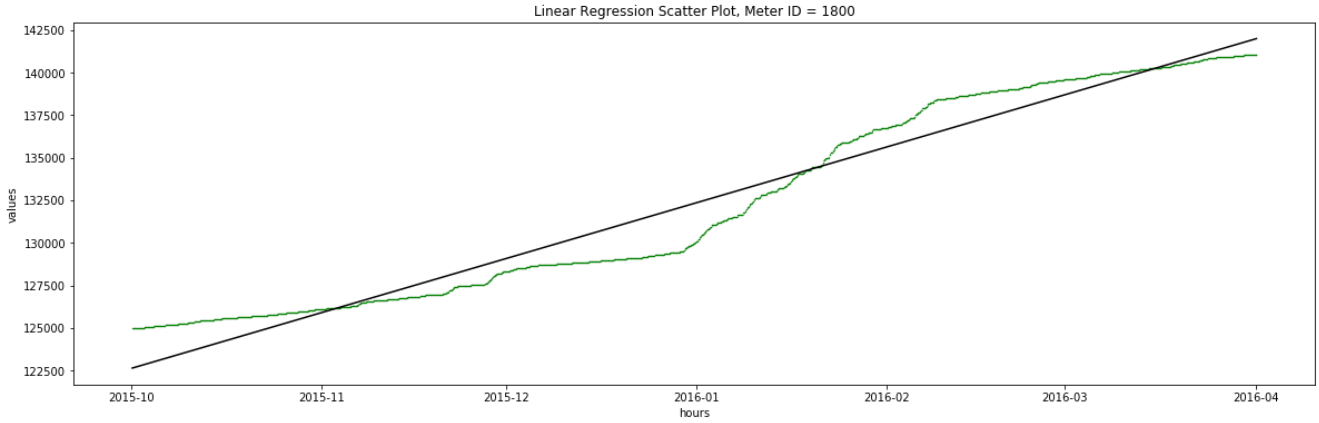

Linear Regression Scatter Plot, Meter ID = 2072
values
290000
285000
280000
275000
270000
265000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

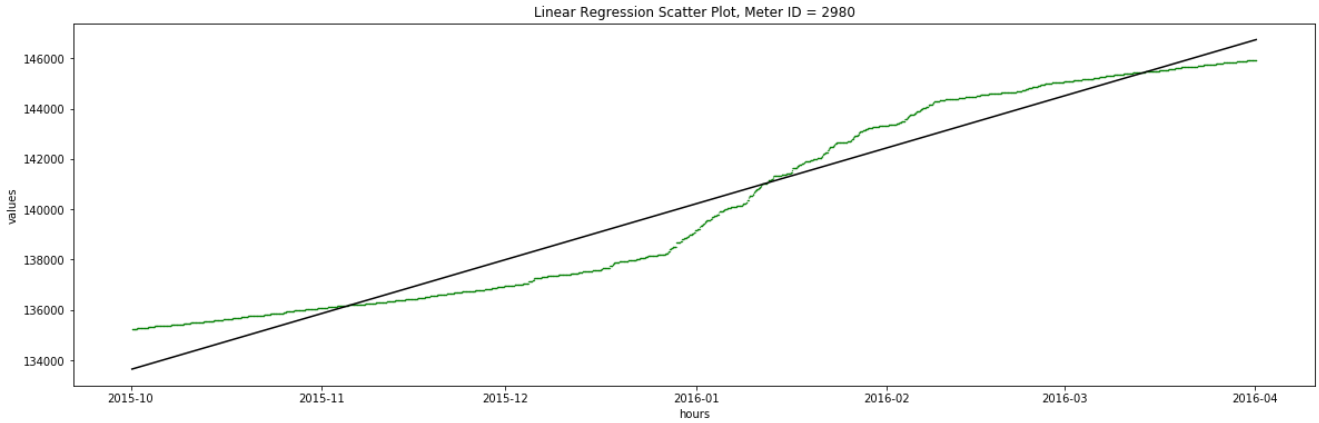

Linear Regression Scatter Plot, Meter ID = 2461
values
140000
135000
130000
125000
120000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

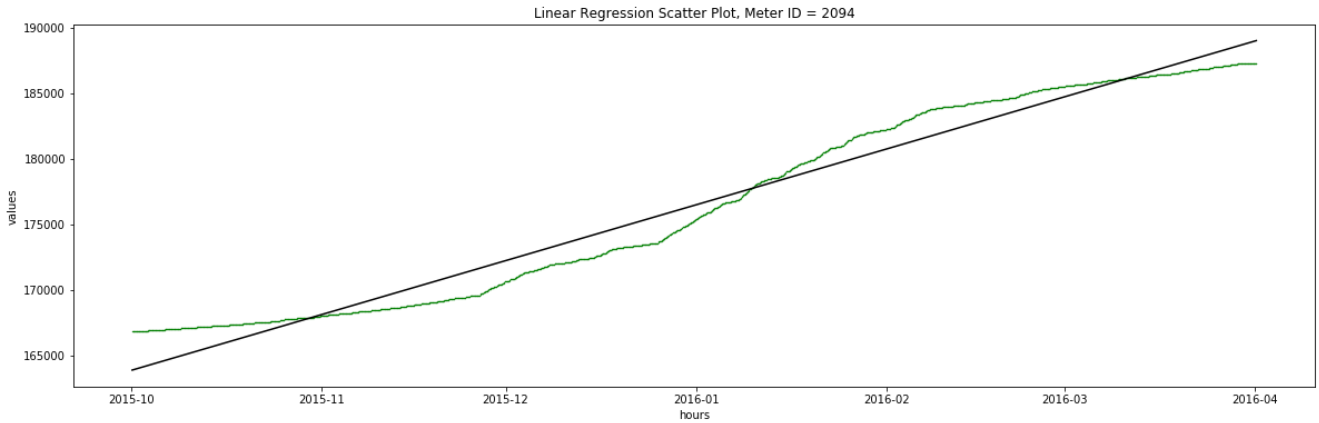

Linear Regression Scatter Plot, Meter ID = 744
values
155000
150000
145000
140000
135000
130000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

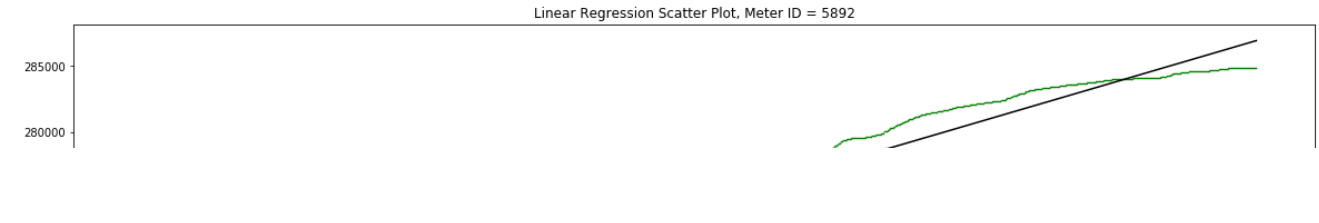

Linear Regression Scatter Plot, Meter ID = 9982
values
122500
120000
117500
115000
112500
110000
107500
105000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

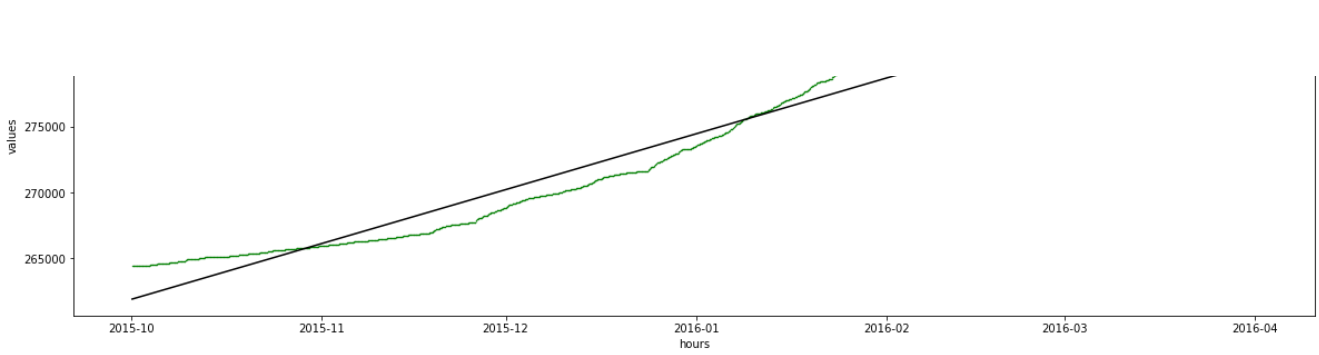

Linear Regression Scatter Plot, Meter ID = 1790
values
hours
360000
350000
340000
330000
320000
310000
300000
290000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

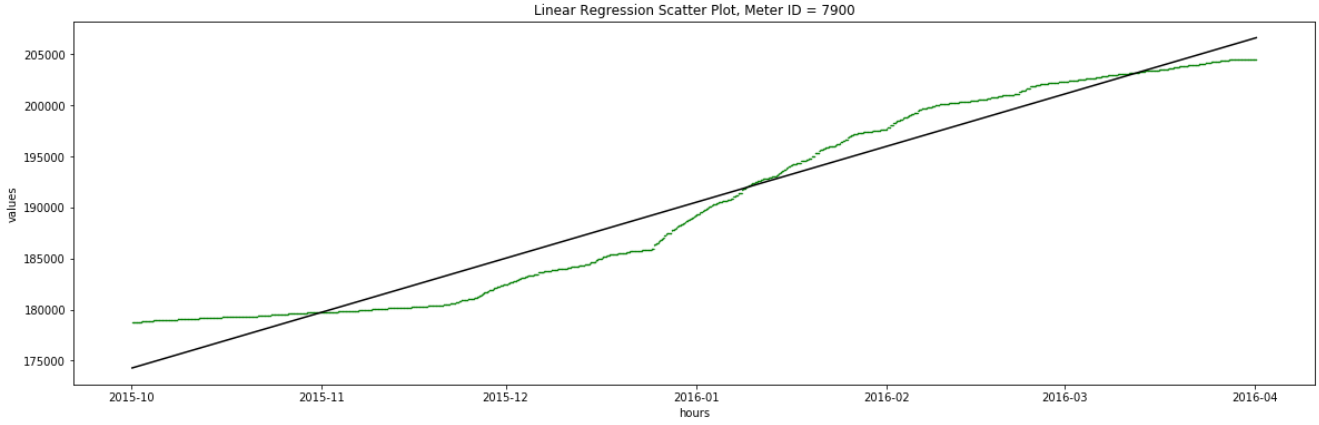

Linear Regression Scatter Plot, Meter ID = 1800
values
hours
142500
140000
137500
135000
132500
130000
127500
125000
122500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

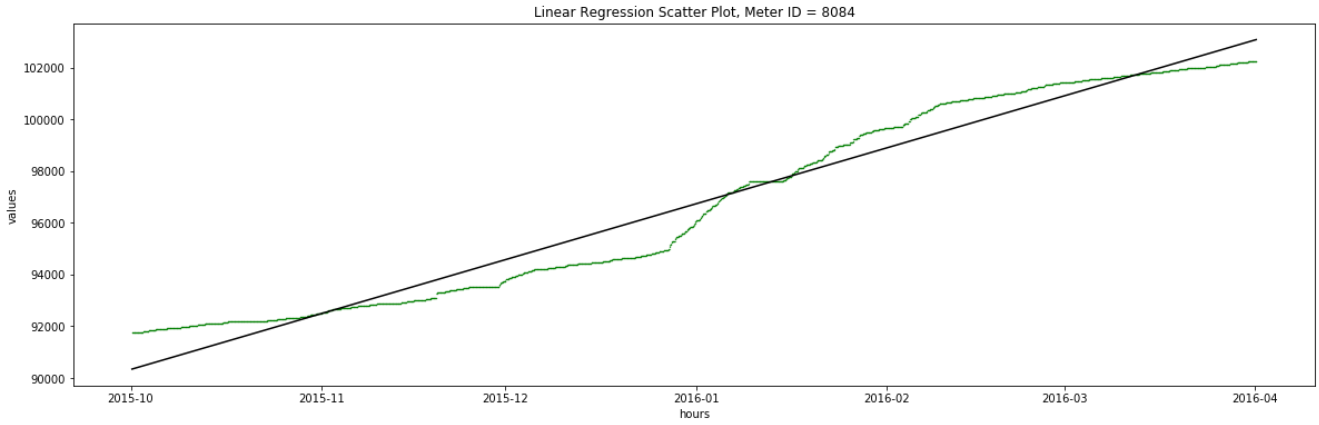

Linear Regression Scatter Plot, Meter ID = 2980
values
hours
146000
144000
142000
140000
138000
136000
134000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

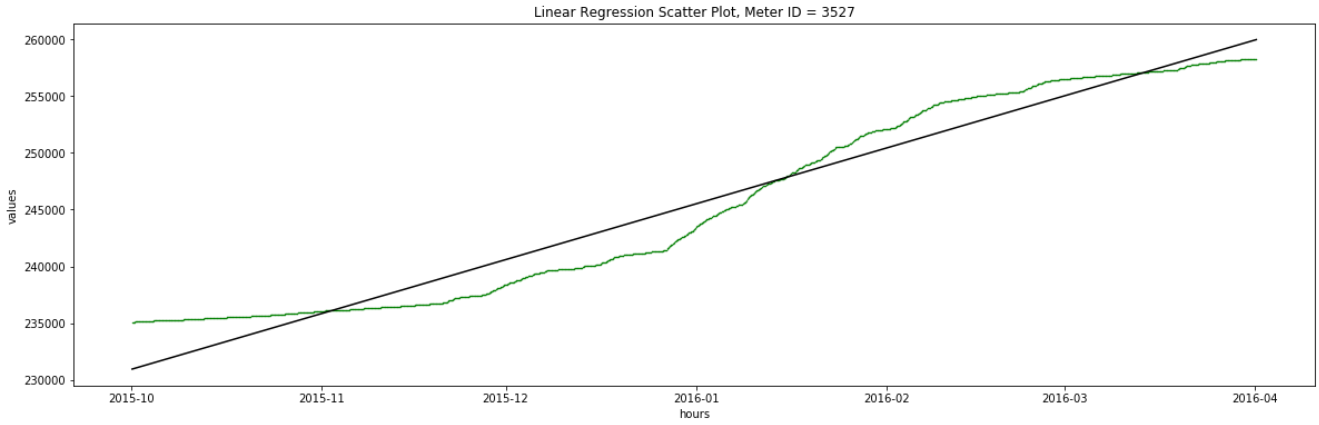

Linear Regression Scatter Plot, Meter ID = 2094
values
hours
190000
185000
180000
175000
170000
165000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

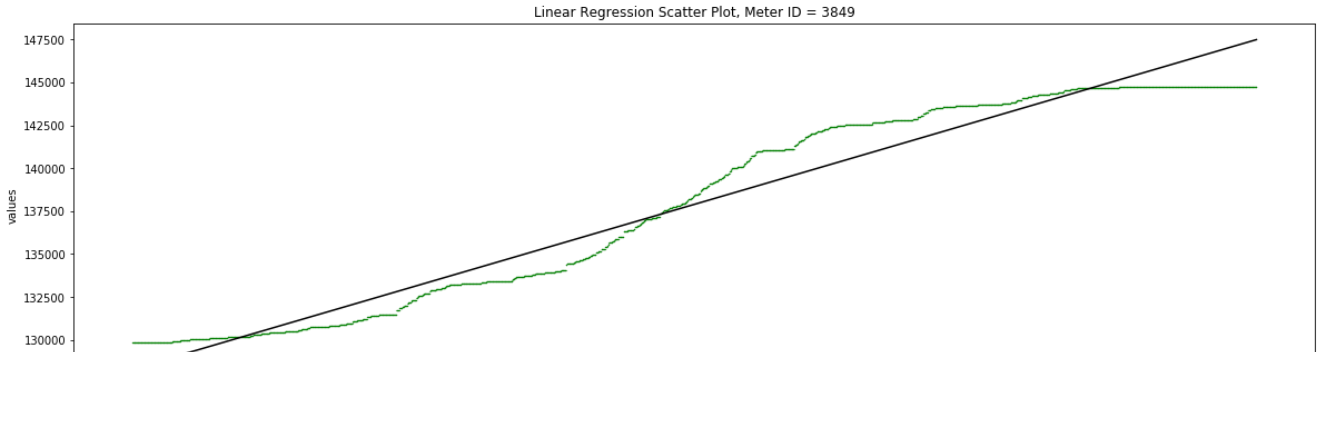

Linear Regression Scatter Plot, Meter ID = 5892
285000
280000

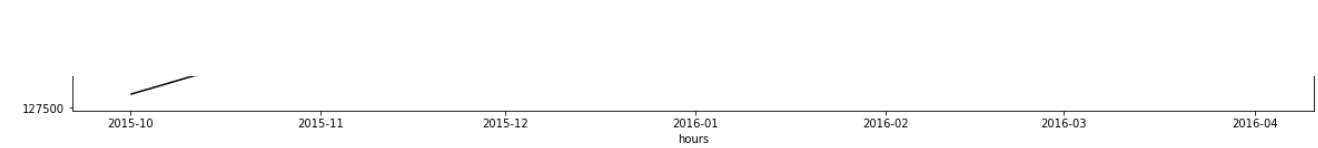

values
275000
270000
265000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

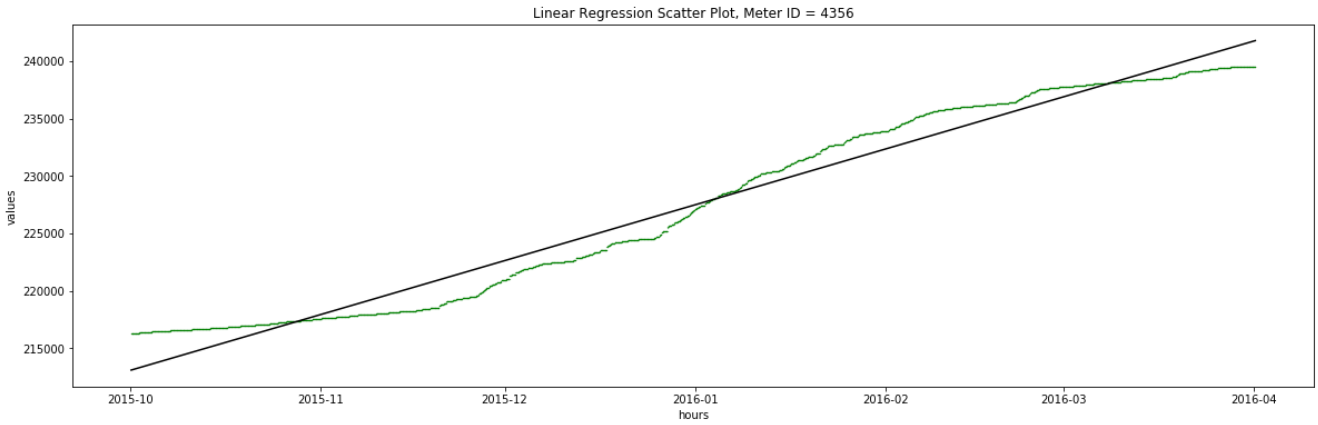

Linear Regression Scatter Plot, Meter ID = 7900
values
205000
200000
195000
190000
185000
180000
175000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

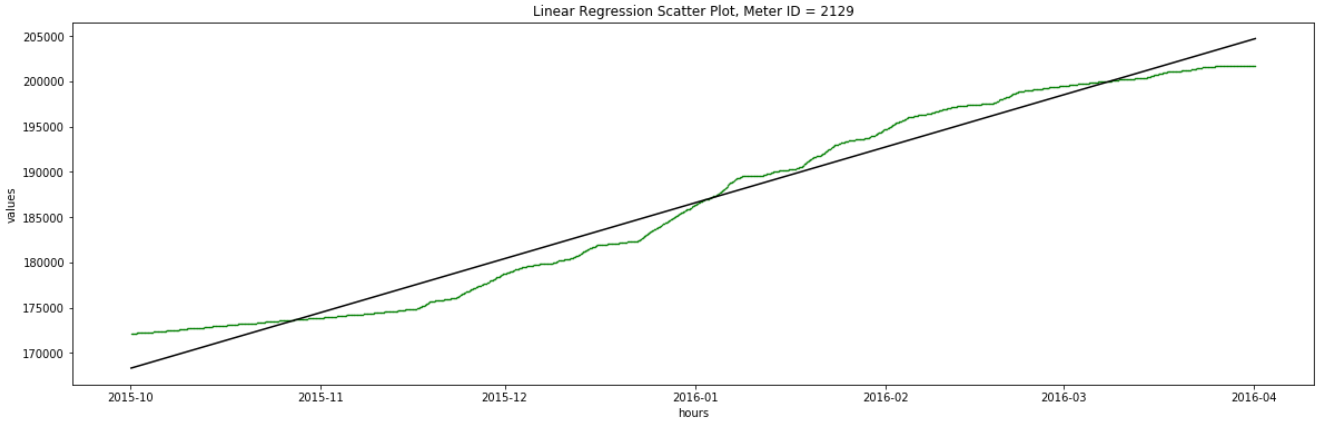

Linear Regression Scatter Plot, Meter ID = 8084
values
102000
100000
98000
96000
94000
92000
90000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

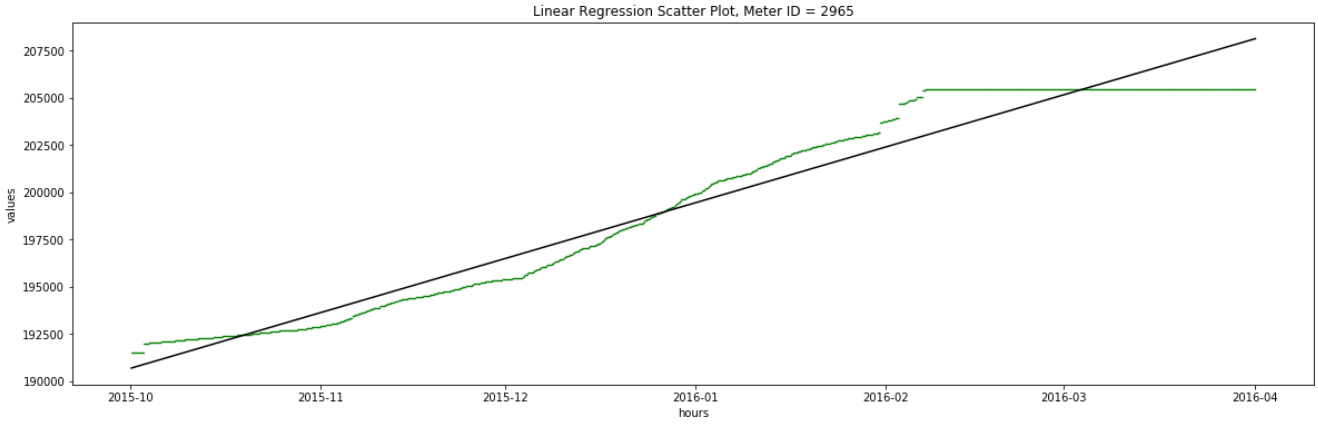

Linear Regression Scatter Plot, Meter ID = 3527
values
260000
255000
250000
245000
240000
235000
230000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

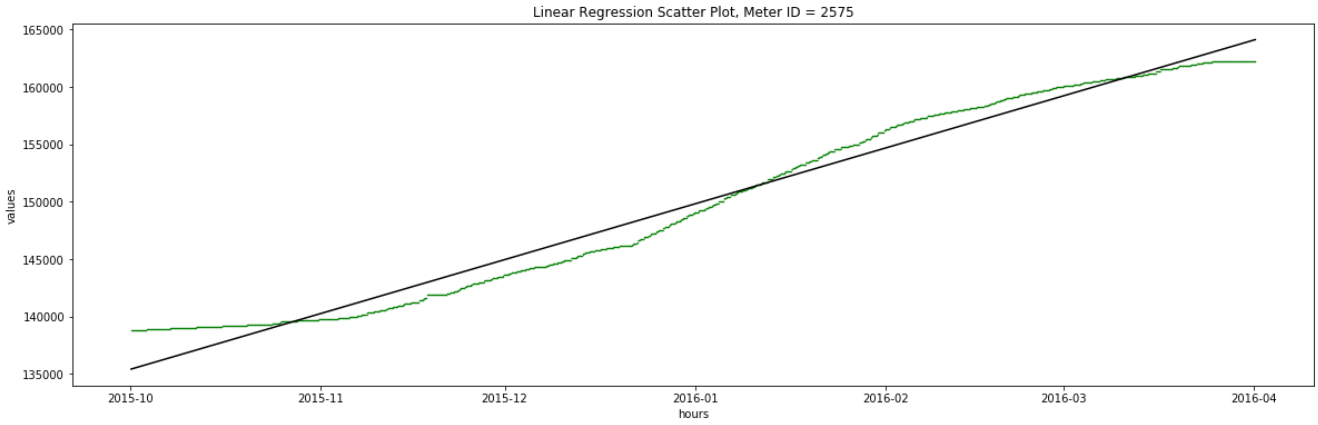

Linear Regression Scatter Plot, Meter ID = 3849
values
147500
145000
142500
140000
137500
135000
132500
130000

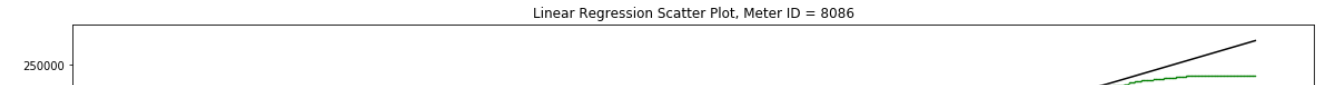

127500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

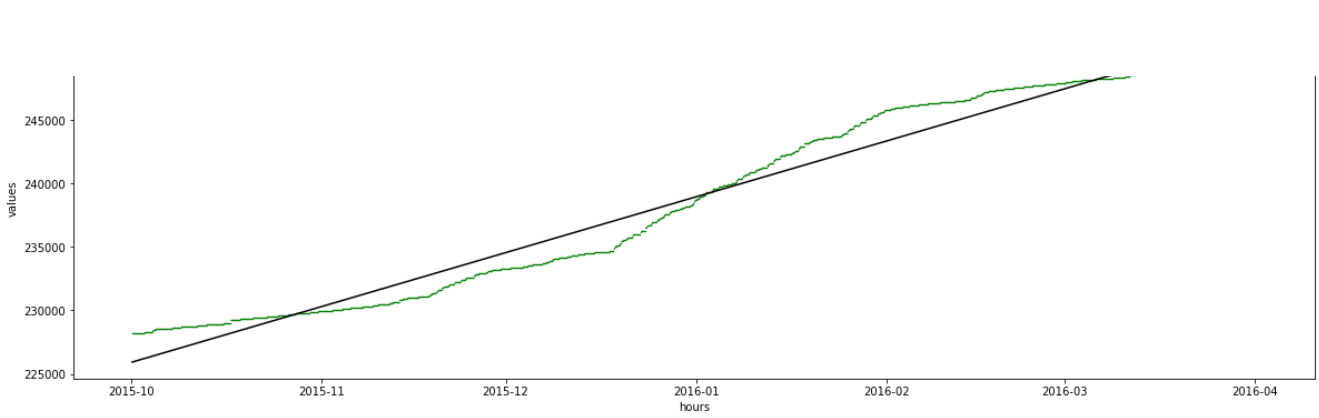

Linear Regression Scatter Plot, Meter ID = 4356
240000
235000
230000
225000
220000
215000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

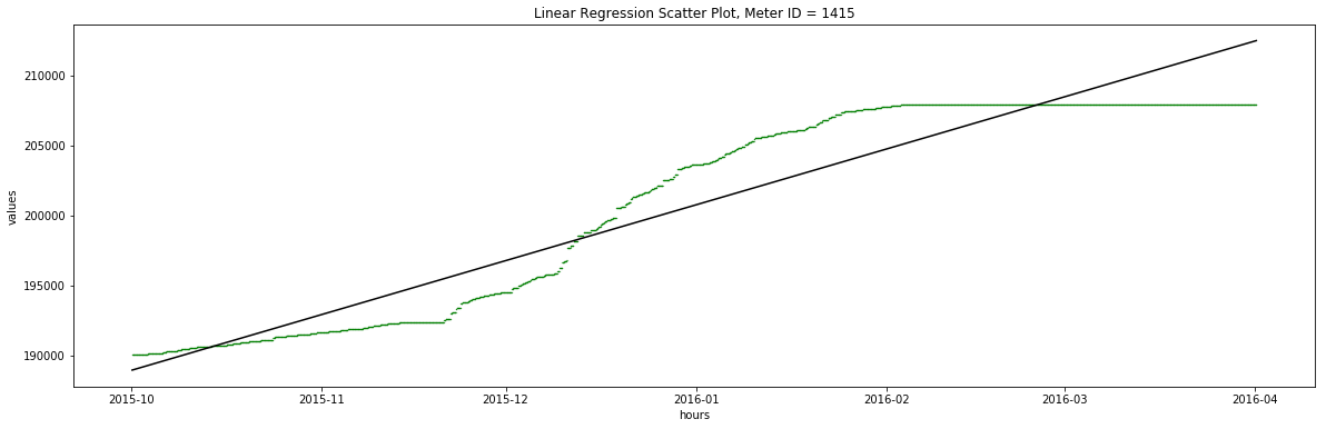

Linear Regression Scatter Plot, Meter ID = 2129
205000
200000
195000
190000
185000
180000
175000
170000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

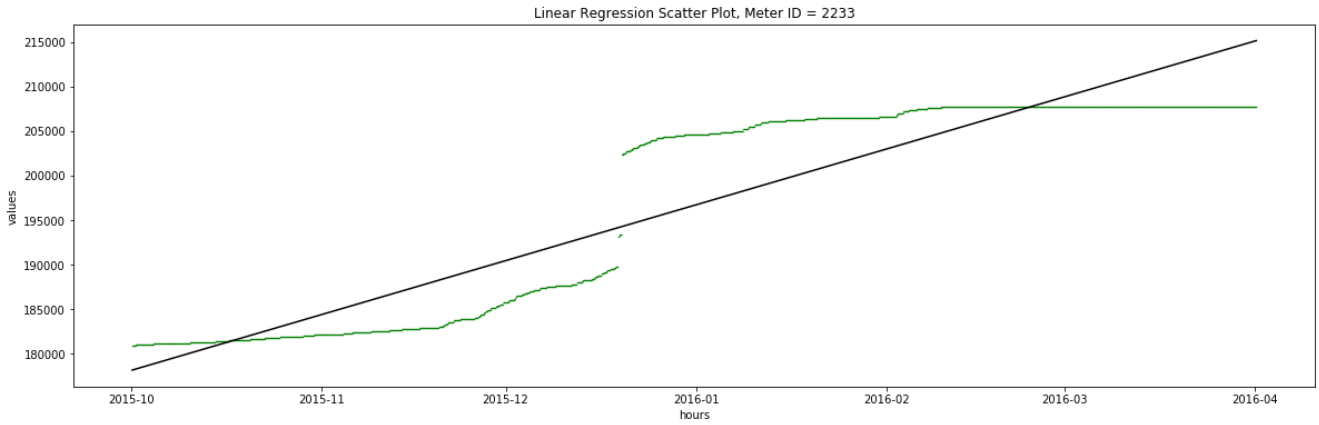

Linear Regression Scatter Plot, Meter ID = 2965
207500
205000
202500
200000
197500
195000
192500
190000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

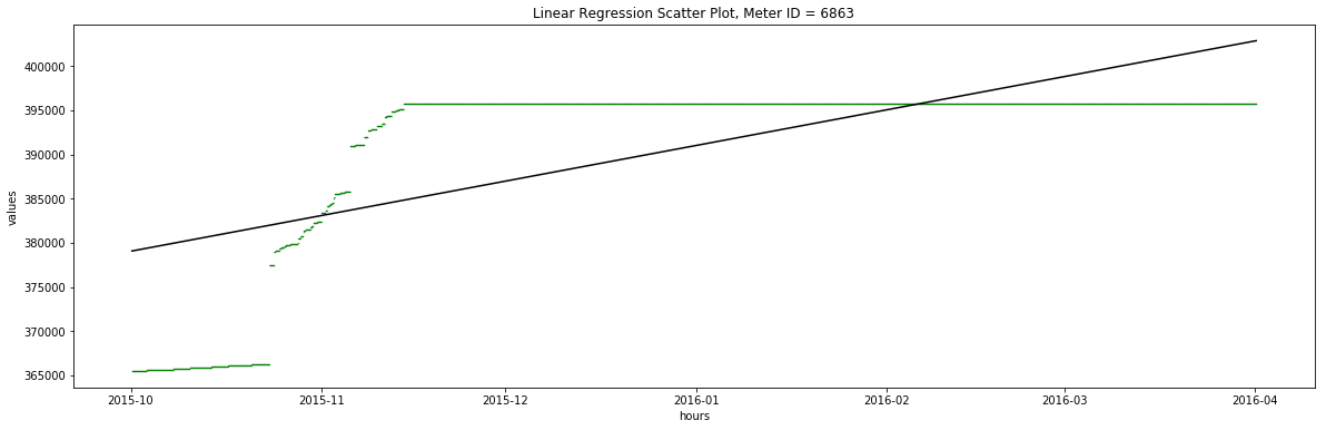

Linear Regression Scatter Plot, Meter ID = 2575
165000
160000
155000
150000
145000
140000
135000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

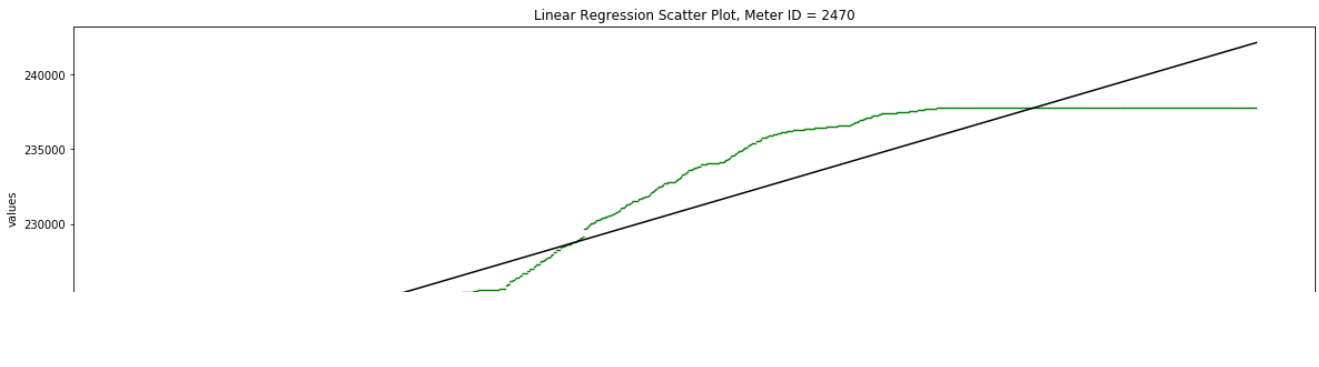

Linear Regression Scatter Plot, Meter ID = 8086
250000

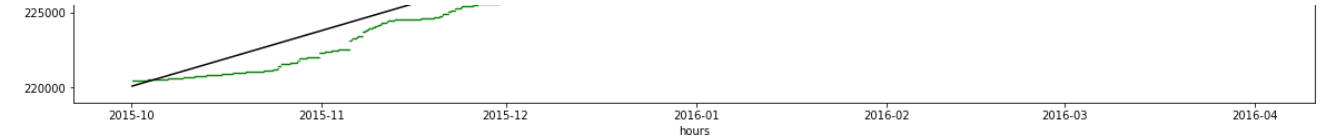

values
hours
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

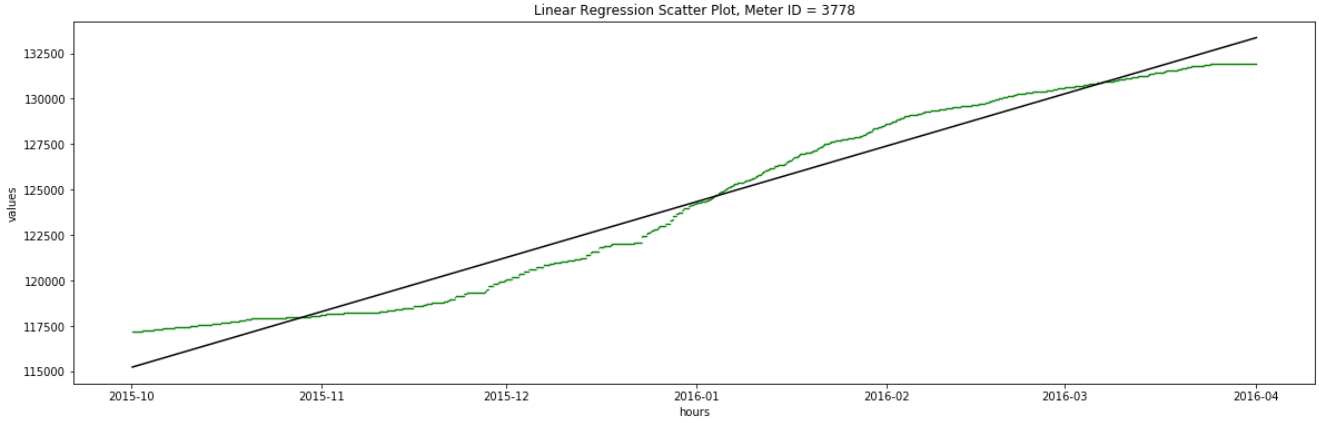

Linear Regression Scatter Plot, Meter ID = 1415
values
hours

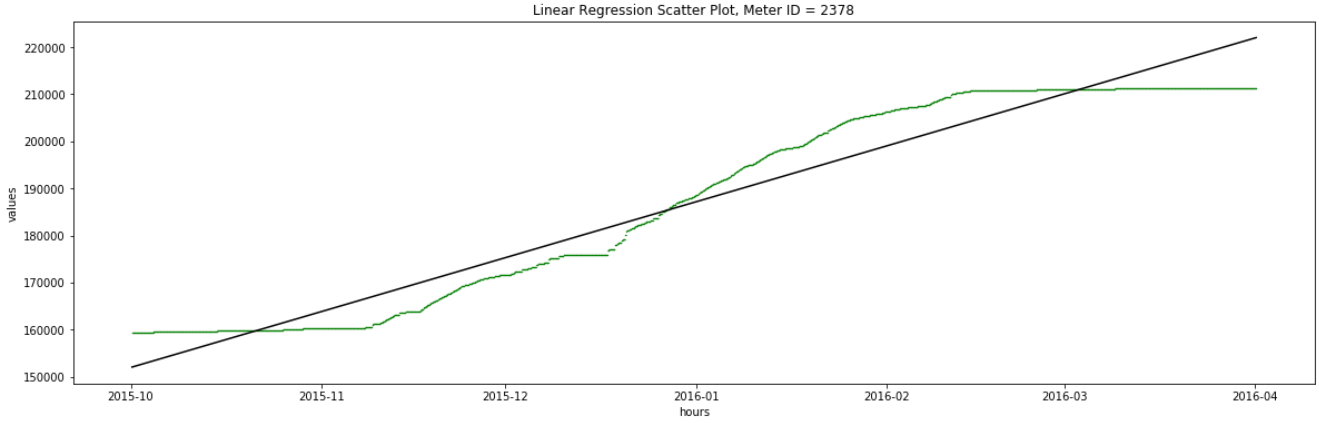

Linear Regression Scatter Plot, Meter ID = 2233
values
hours

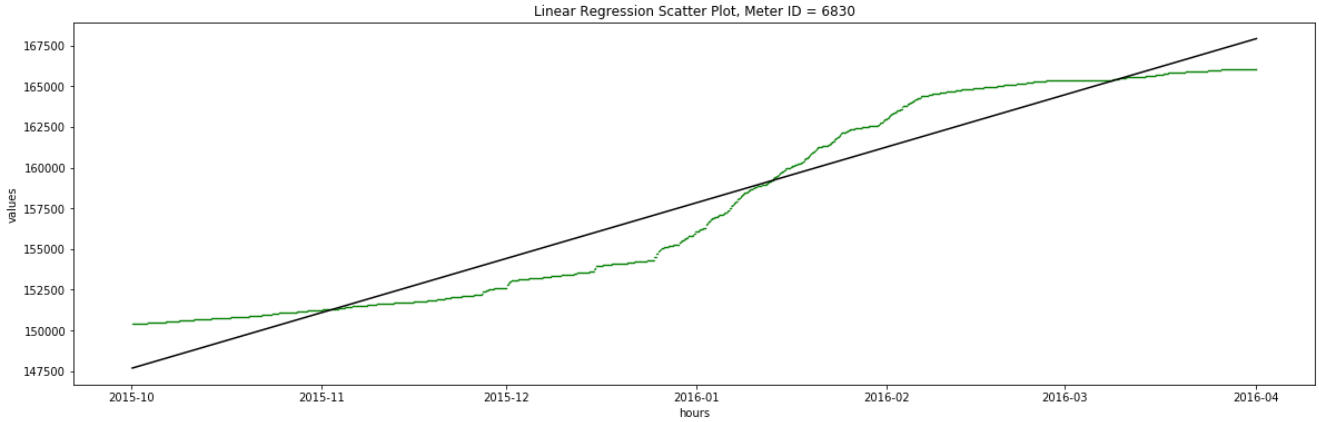

Linear Regression Scatter Plot, Meter ID = 6863
values
hours

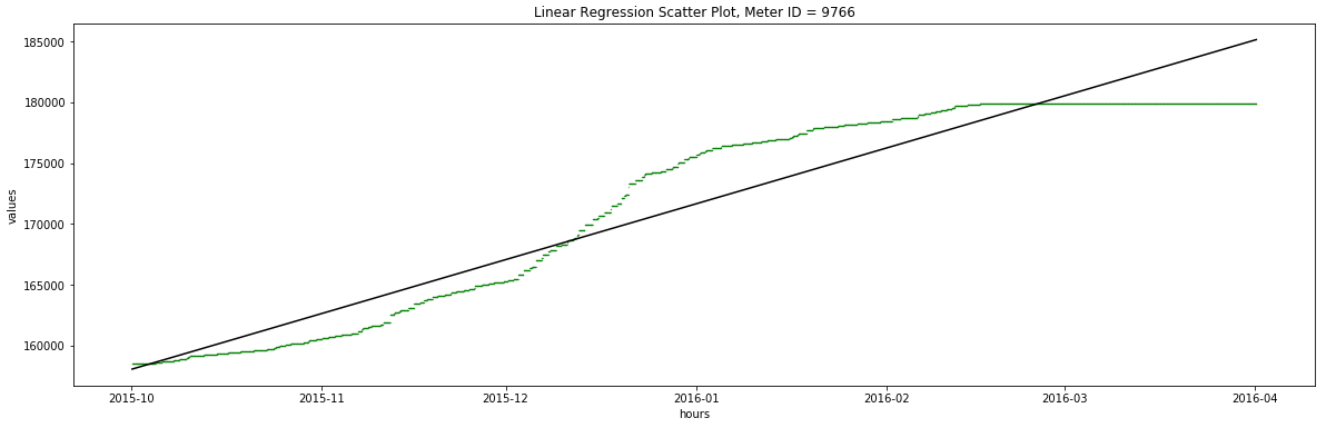

Linear Regression Scatter Plot, Meter ID = 2470
values



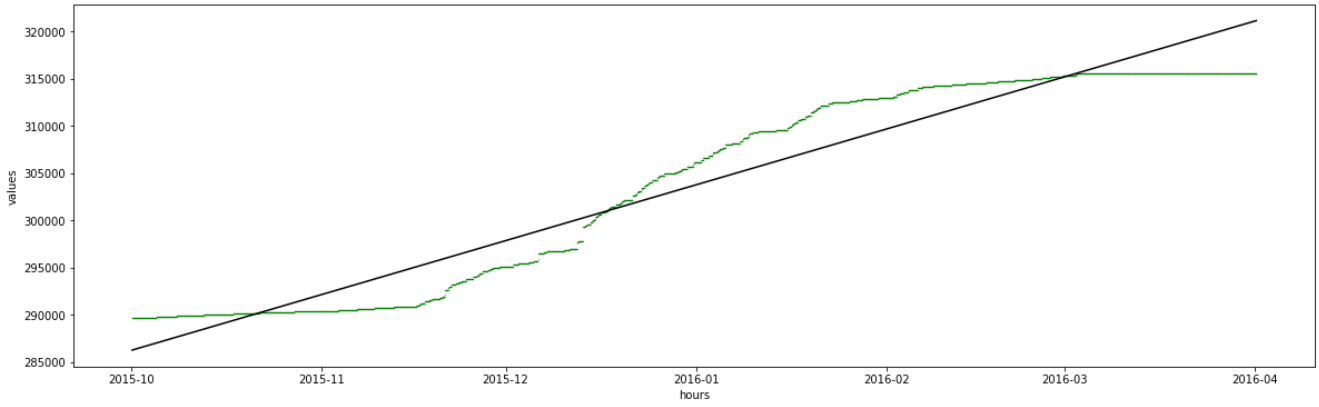


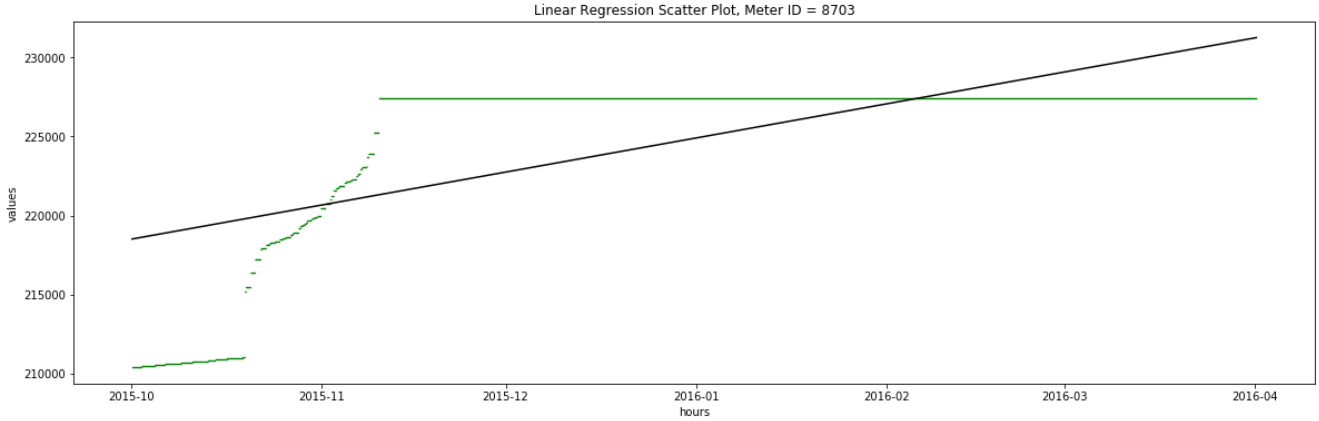


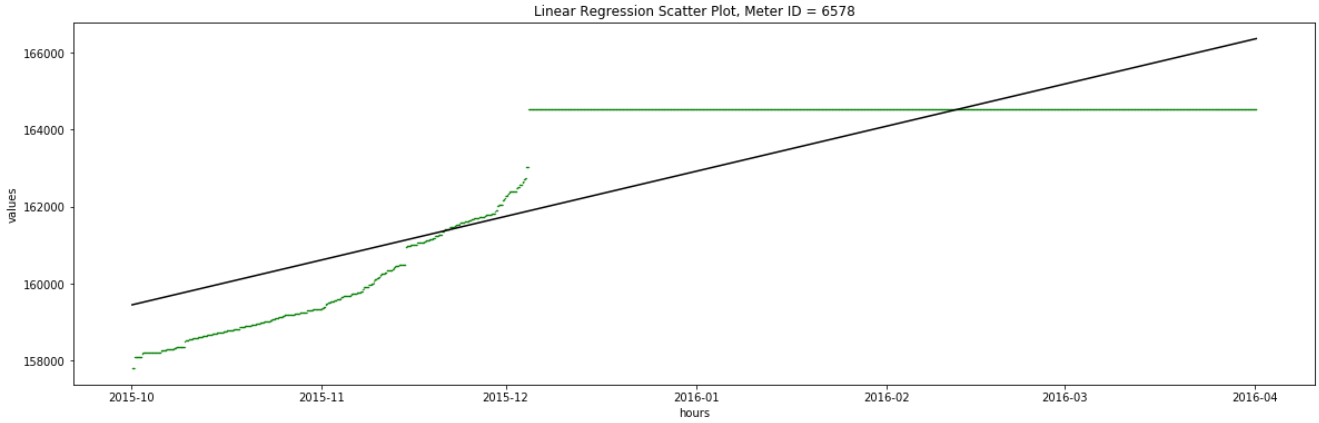


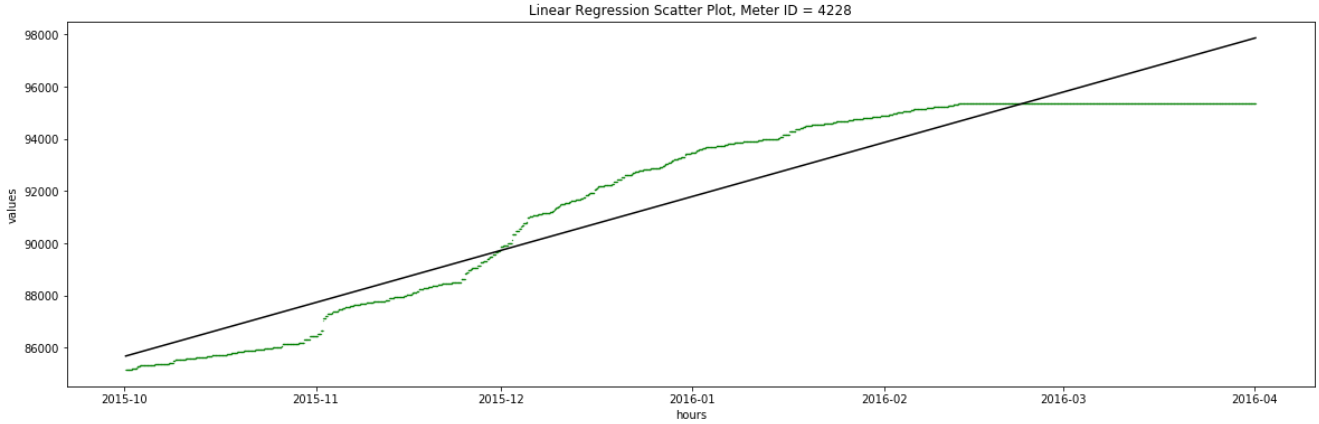


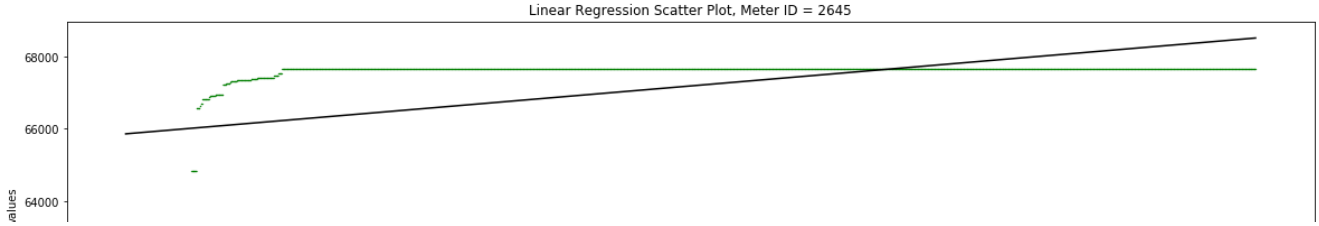
Linear Regression Scatter Plot, Meter ID = 4193

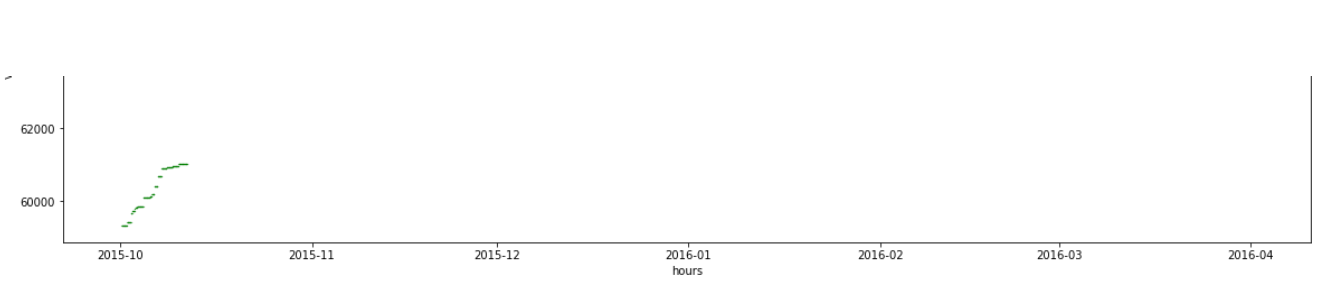

values
hours
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

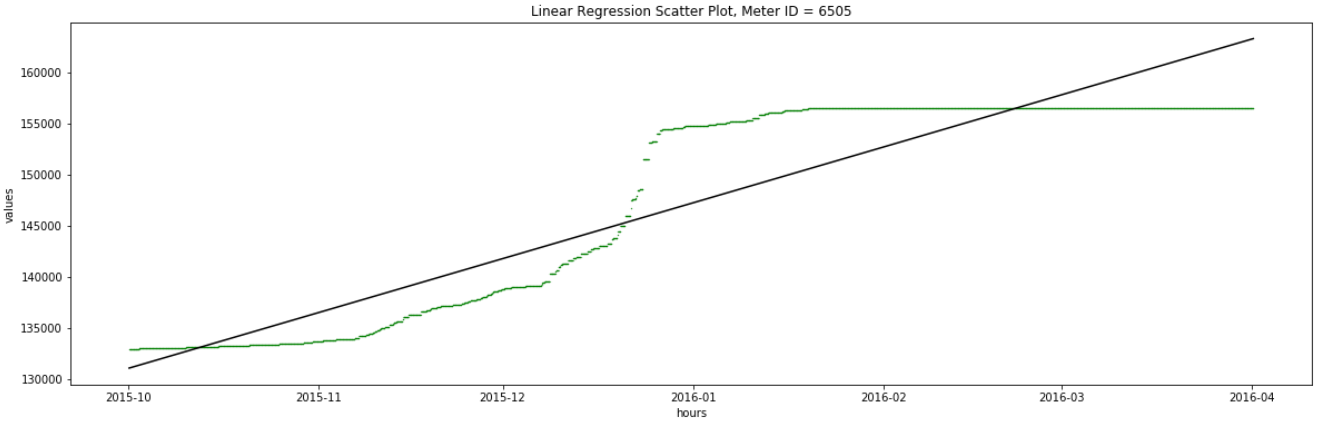

Linear Regression Scatter Plot, Meter ID = 8703
values
hours

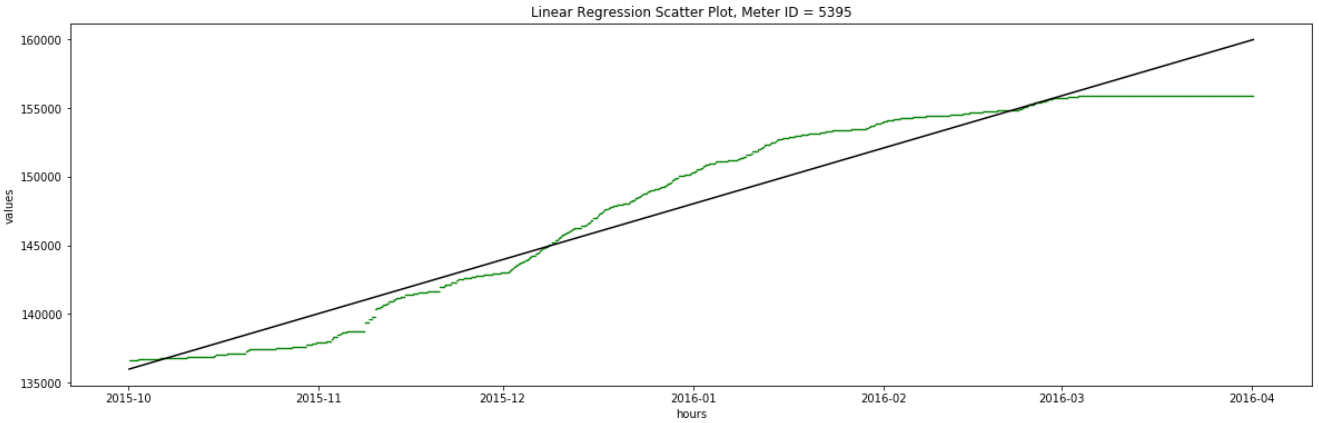

Linear Regression Scatter Plot, Meter ID = 6578
values
hours

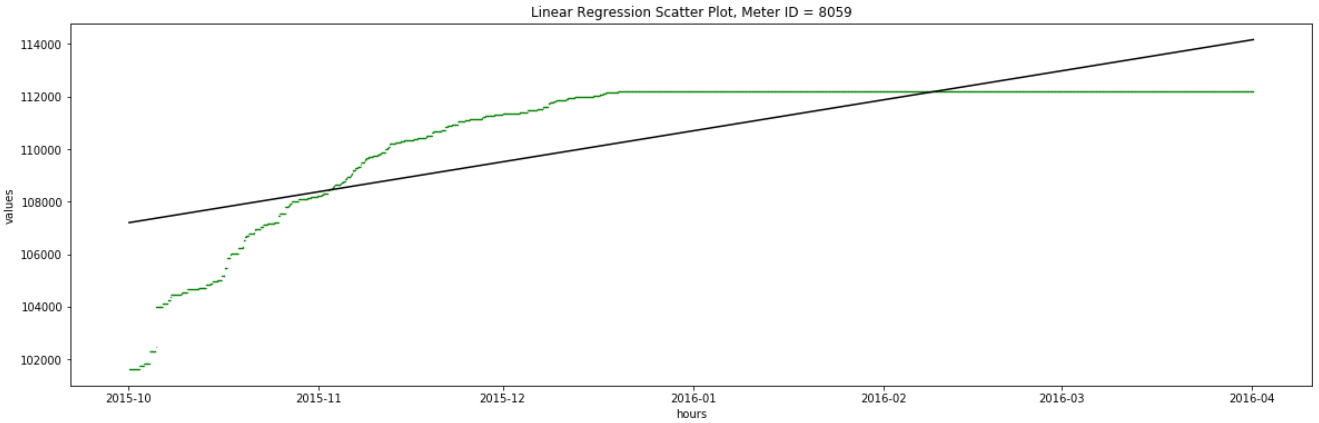

Linear Regression Scatter Plot, Meter ID = 4228
values
hours

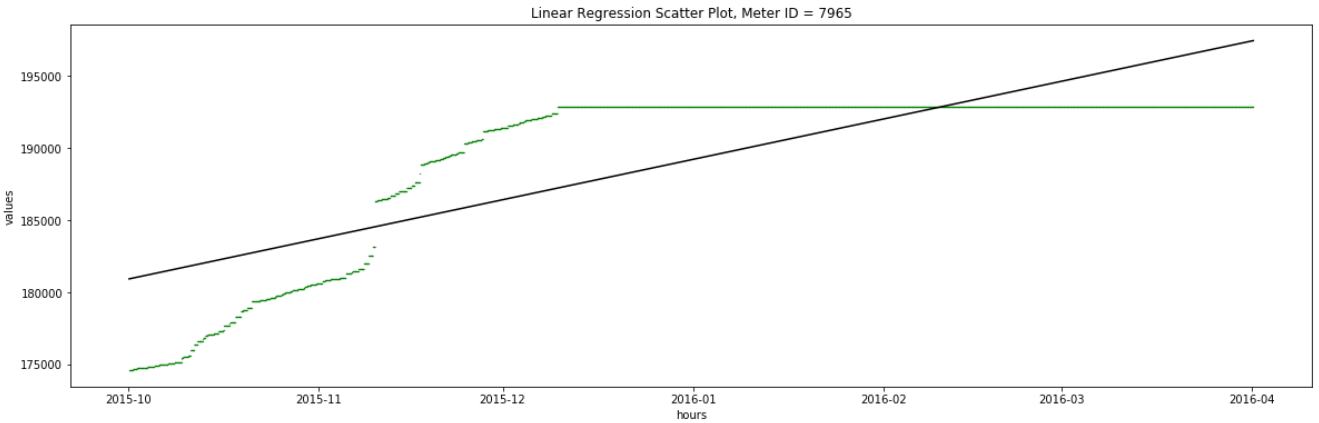

Linear Regression Scatter Plot, Meter ID = 2645
values

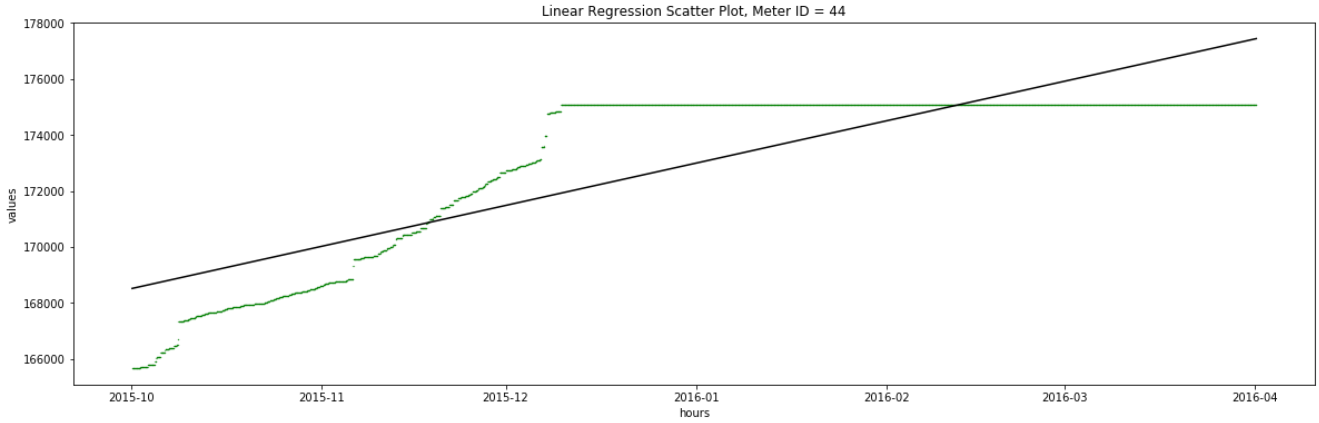

62000
60000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

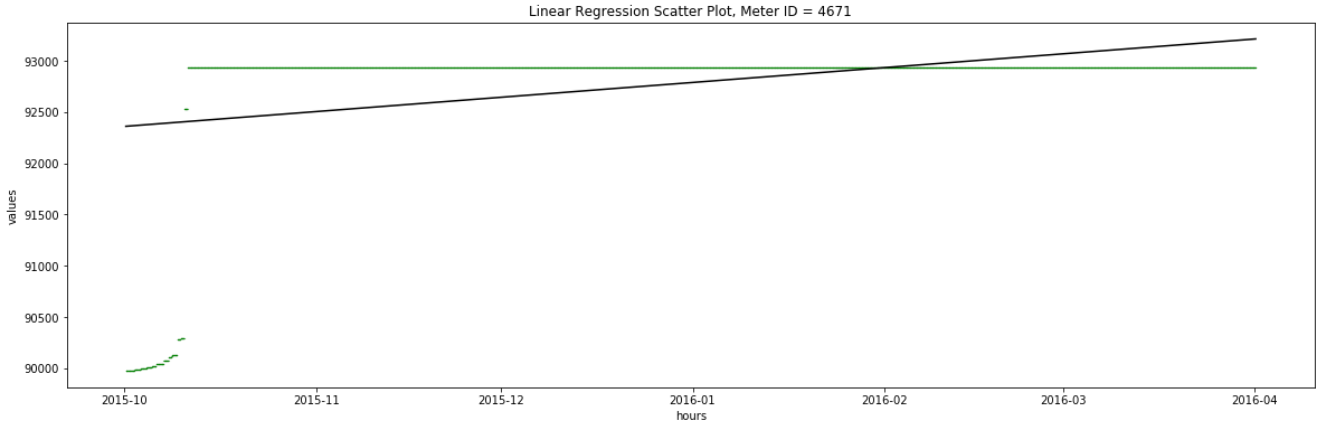

Linear Regression Scatter Plot, Meter ID = 6505
values
160000
155000
150000
145000
140000
135000
130000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

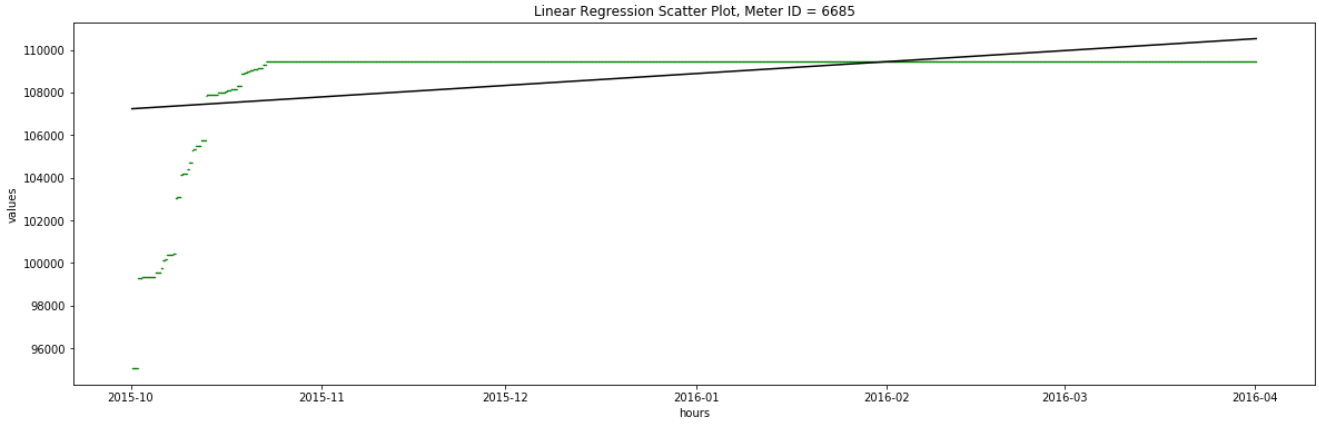

Linear Regression Scatter Plot, Meter ID = 5395
values
160000
155000
150000
145000
140000
135000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

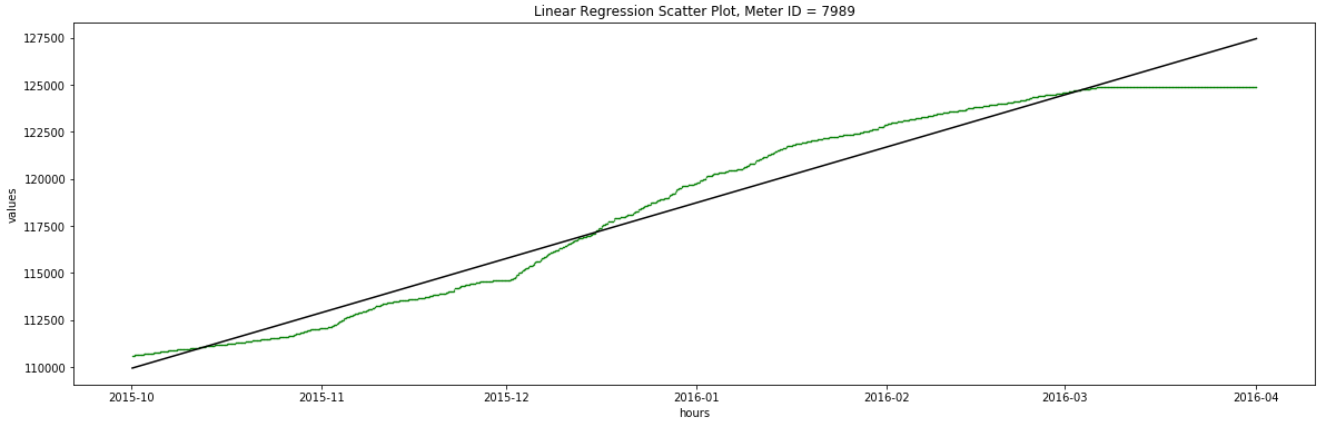

Linear Regression Scatter Plot, Meter ID = 8059
values
114000
112000
110000
108000
106000
104000
102000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

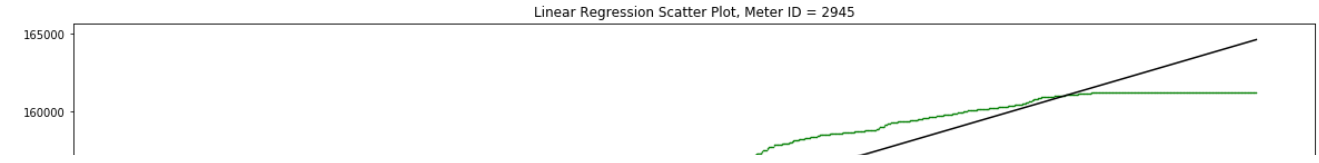

Linear Regression Scatter Plot, Meter ID = 7965
values
195000
190000
185000
180000
175000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

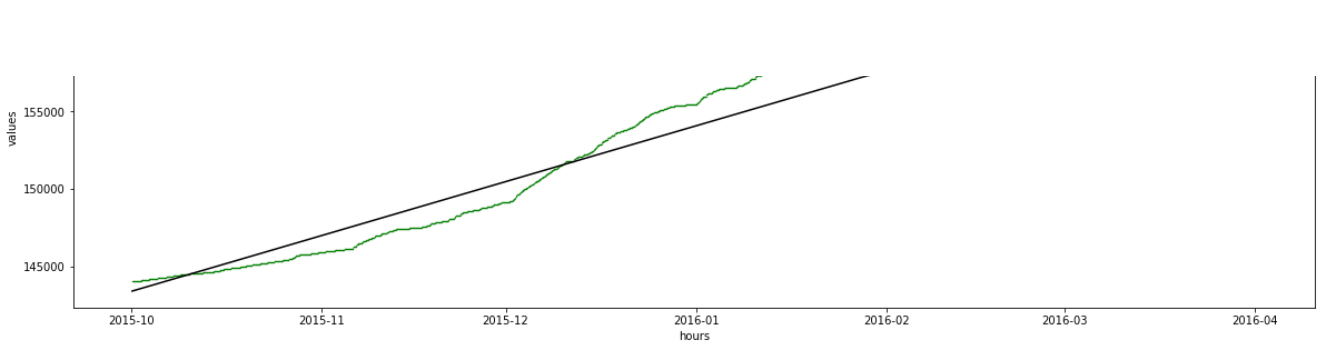

Linear Regression Scatter Plot, Meter ID = 44
values
hours
178000
176000
174000
172000
170000
168000
166000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

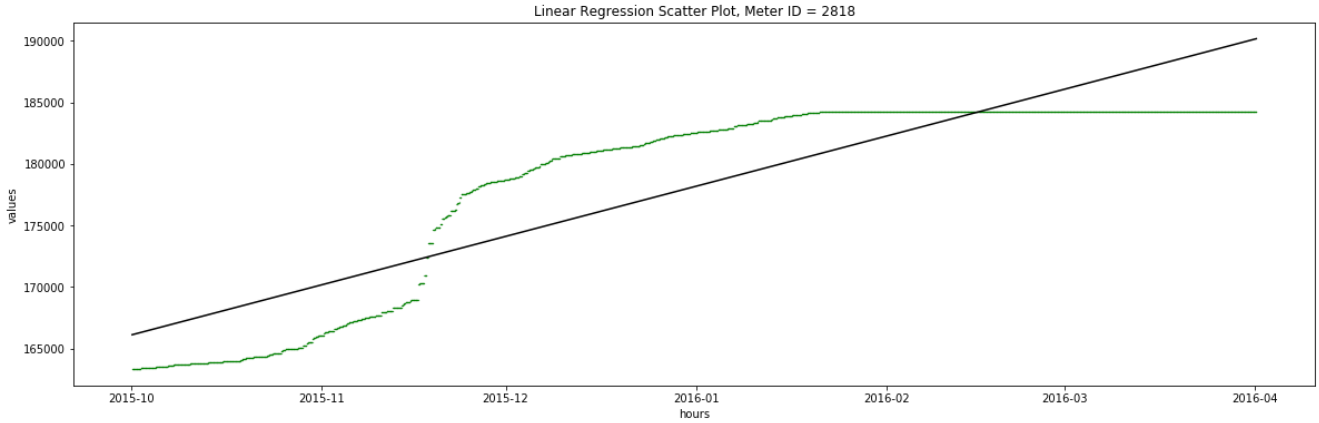

Linear Regression Scatter Plot, Meter ID = 4671
values
hours
93000
92500
92000
91500
91000
90500
90000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

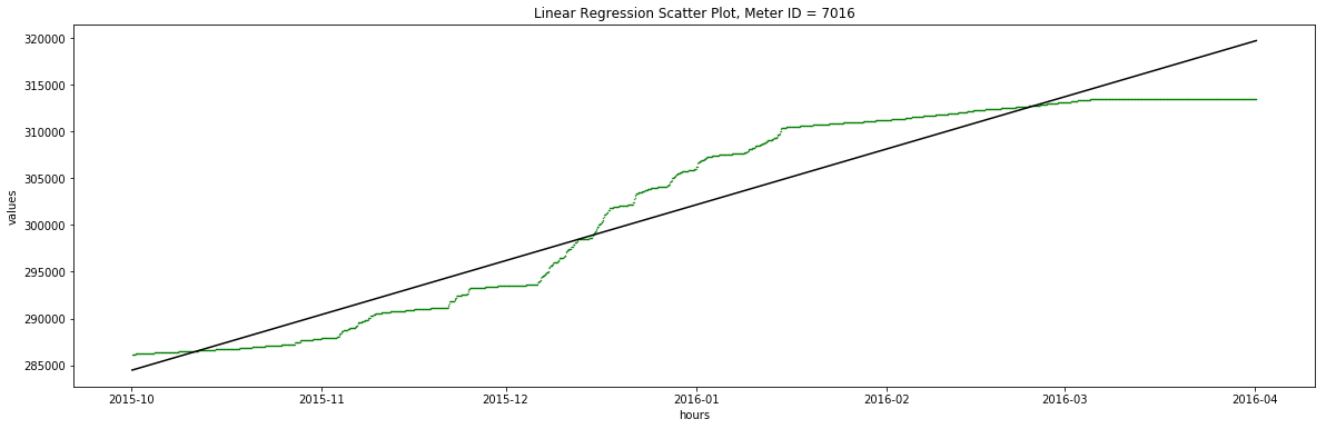

Linear Regression Scatter Plot, Meter ID = 6685
values
hours
110000
108000
106000
104000
102000
100000
98000
96000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

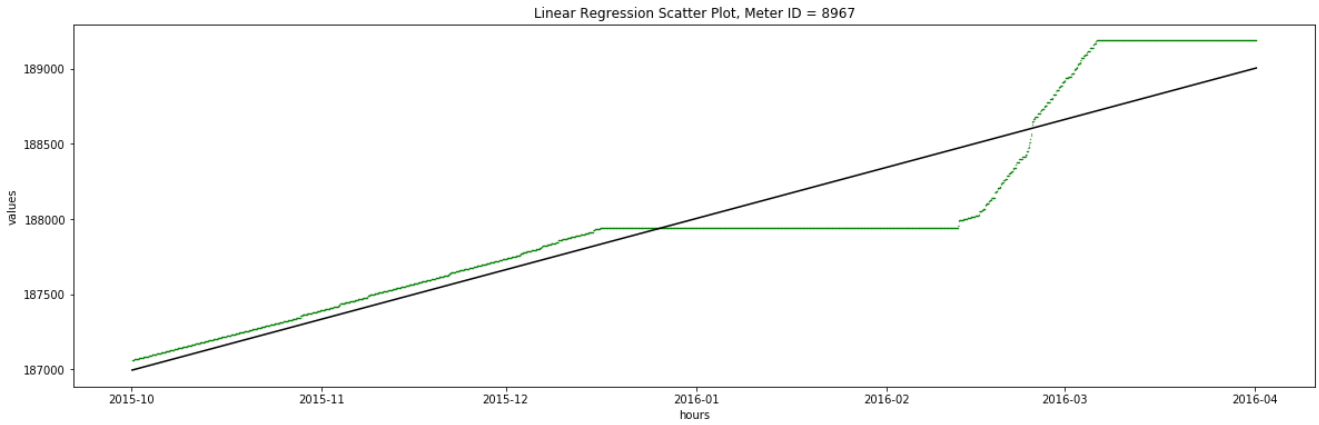

Linear Regression Scatter Plot, Meter ID = 7989
values
hours
127500
125000
122500
120000
117500
115000
112500
110000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

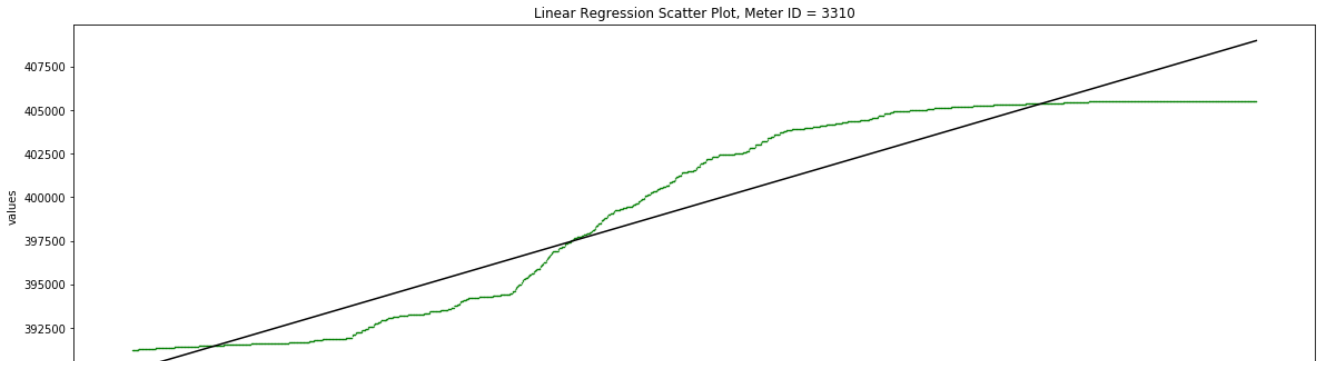

Linear Regression Scatter Plot, Meter ID = 2945
165000
160000

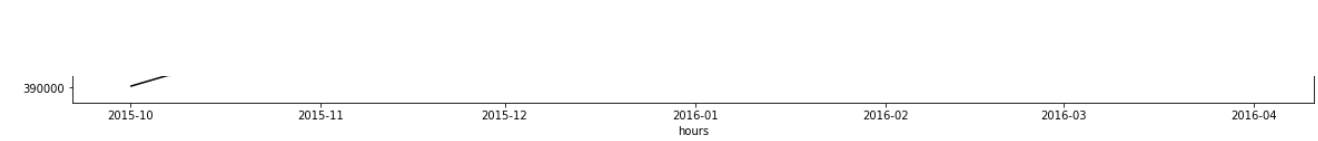

values
155000
150000
145000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

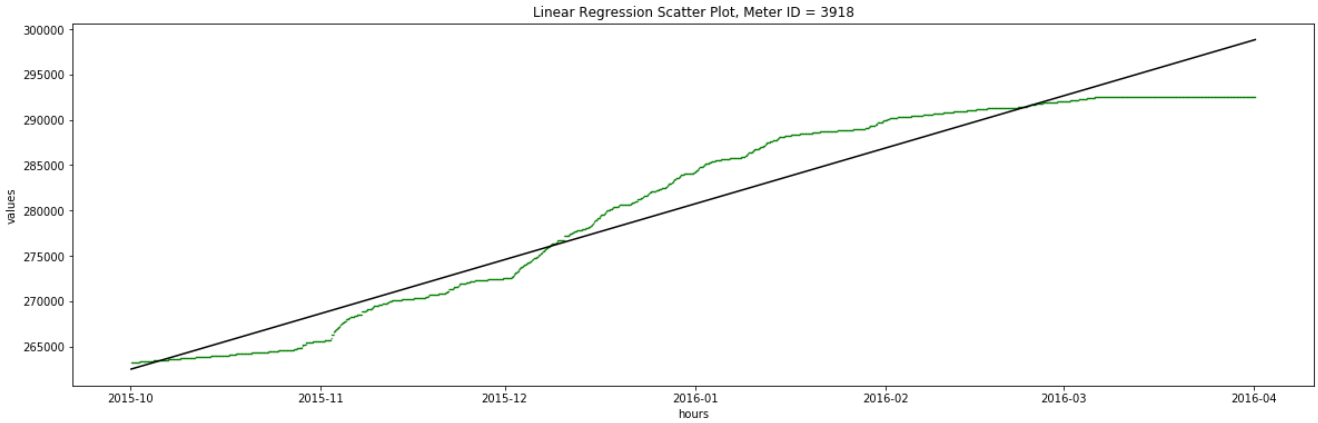

Linear Regression Scatter Plot, Meter ID = 2818
values
190000
185000
180000
175000
170000
165000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

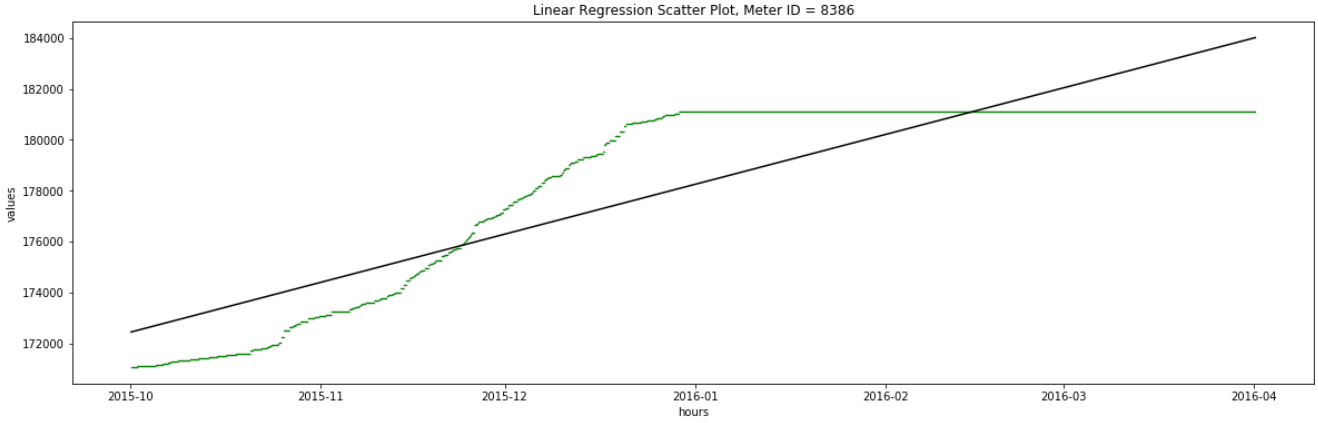

Linear Regression Scatter Plot, Meter ID = 7016
values
320000
315000
310000
305000
300000
295000
290000
285000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

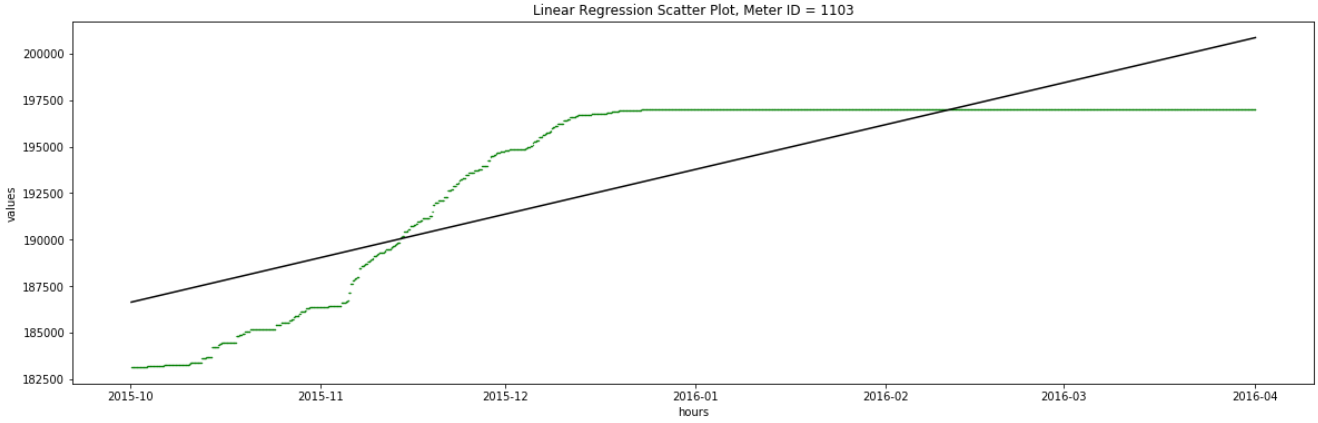

Linear Regression Scatter Plot, Meter ID = 8967
values
189000
188500
188000
187500
187000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

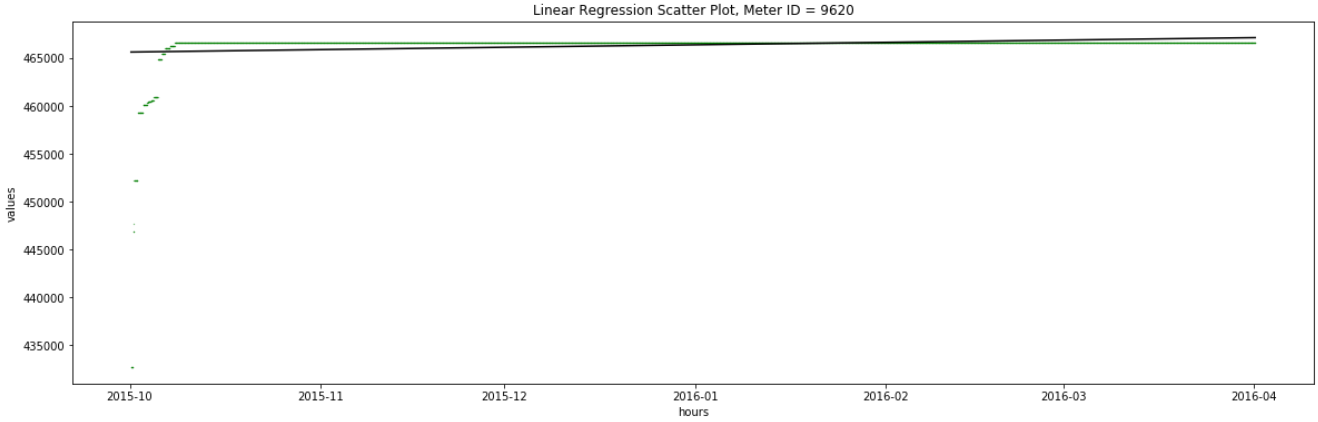

Linear Regression Scatter Plot, Meter ID = 3310
values
407500
405000
402500
400000
397500
395000
392500

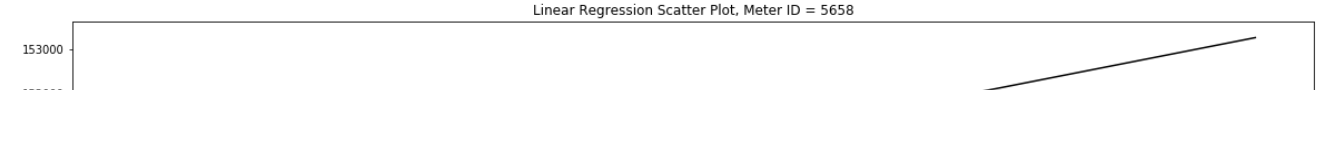

390000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

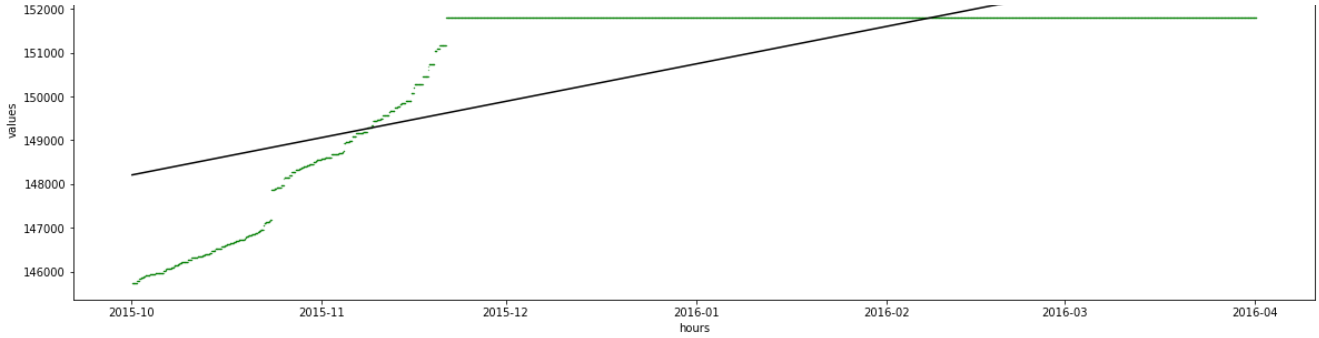

Linear Regression Scatter Plot, Meter ID = 3918
300000
295000
290000
285000
280000
275000
270000
265000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

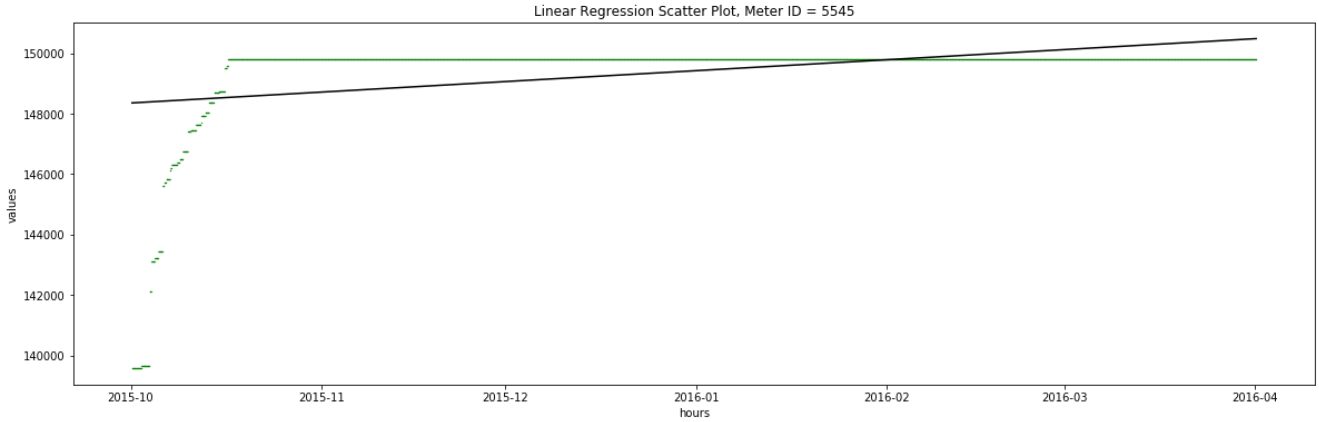

Linear Regression Scatter Plot, Meter ID = 8386
184000
182000
180000
178000
176000
174000
172000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

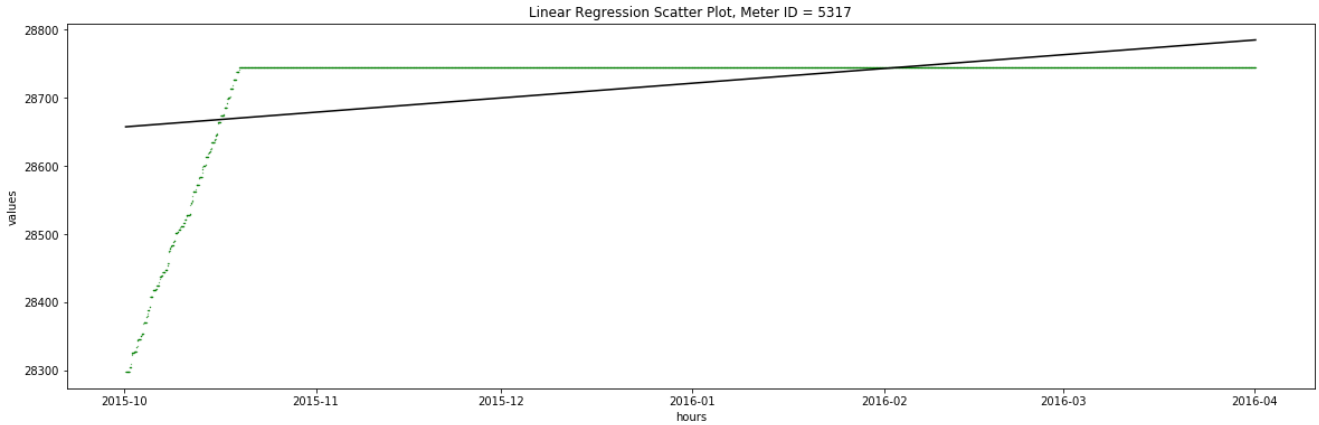

Linear Regression Scatter Plot, Meter ID = 1103
200000
197500
195000
192500
190000
187500
185000
182500
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

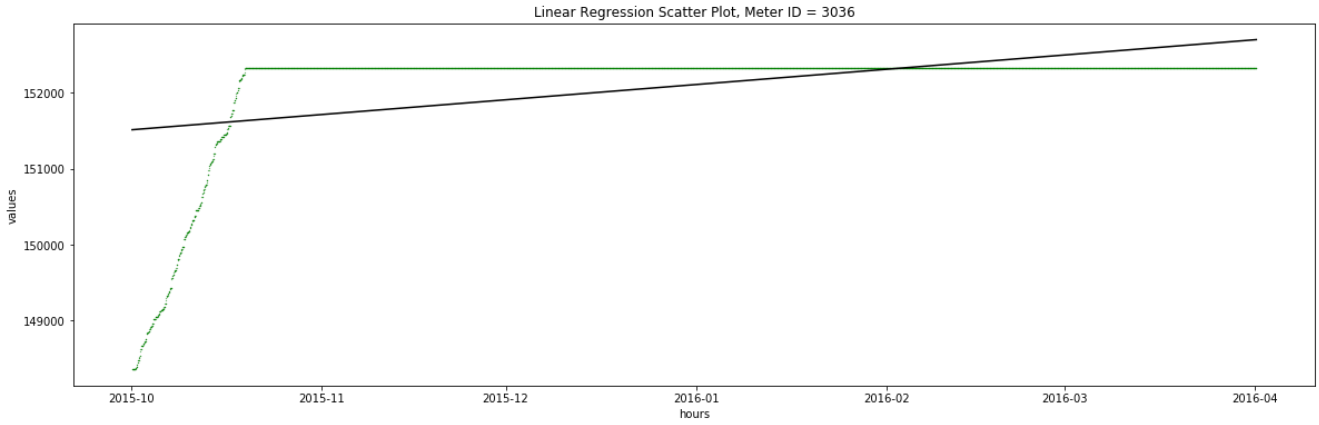

Linear Regression Scatter Plot, Meter ID = 9620
465000
460000
455000
450000
445000
440000
435000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

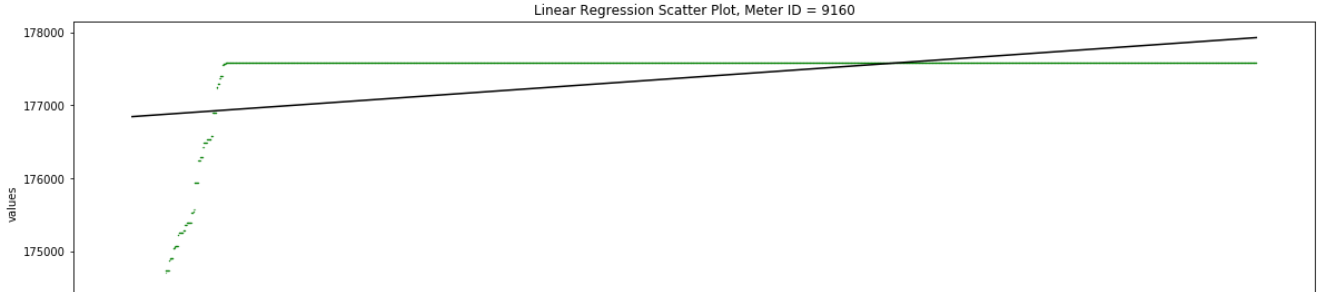

Linear Regression Scatter Plot, Meter ID = 5658
153000

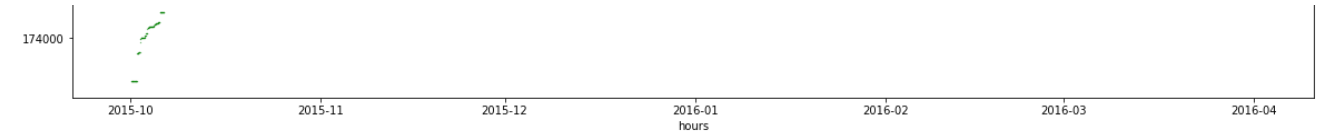

152000
151000
150000
149000
148000
147000
146000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

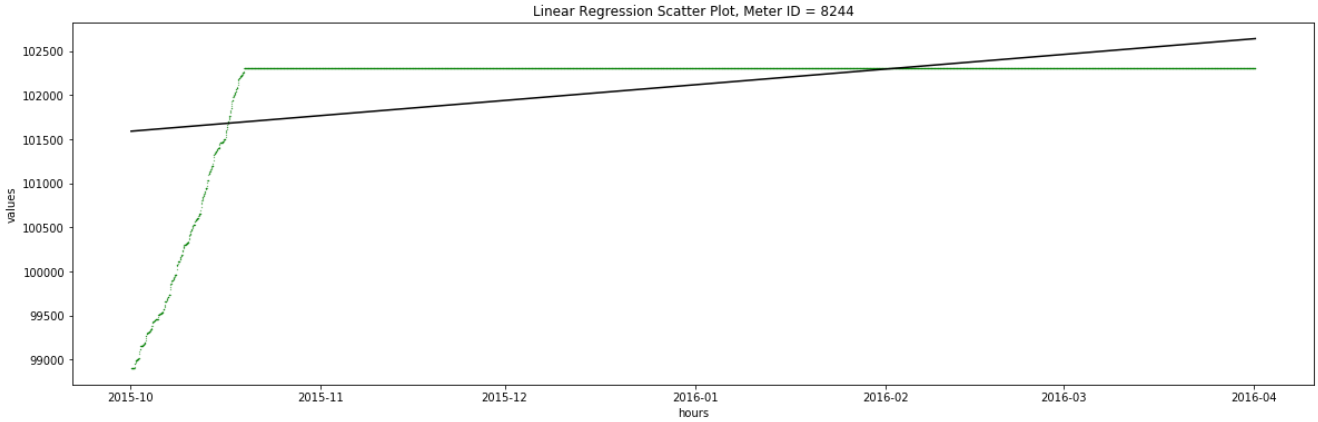

Linear Regression Scatter Plot, Meter ID = 5545
150000
148000
146000
144000
142000
140000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

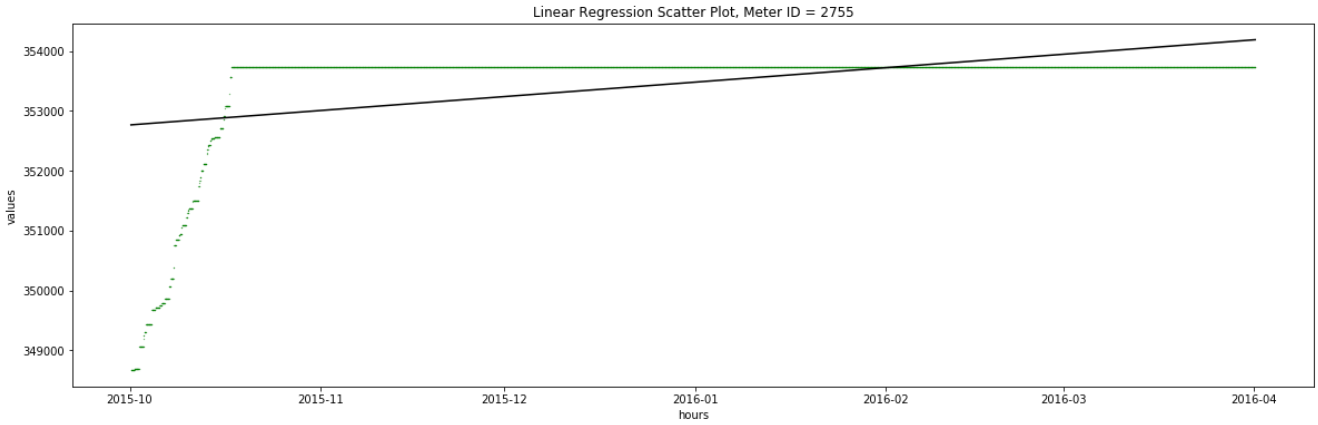

Linear Regression Scatter Plot, Meter ID = 5317
28800
28700
28600
28500
28400
28300
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

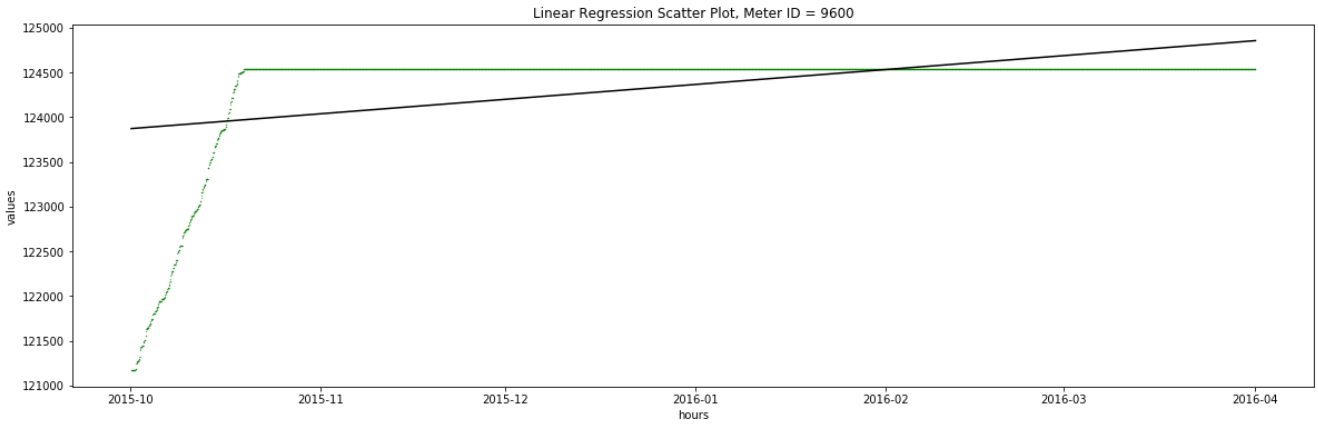

Linear Regression Scatter Plot, Meter ID = 3036
152000
151000
150000
149000
values
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

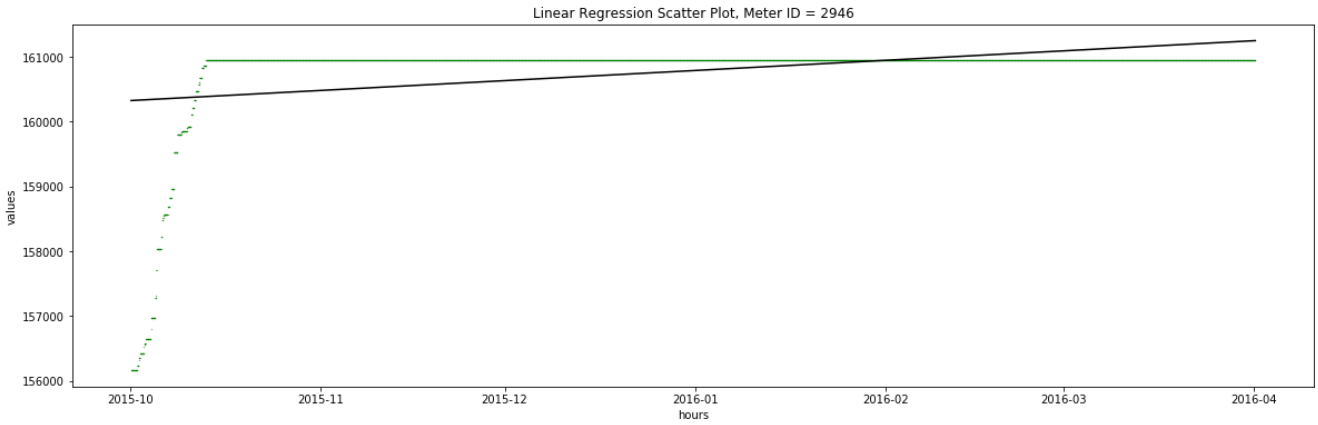

Linear Regression Scatter Plot, Meter ID = 9160
178000
177000
176000
175000
values


174000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
hours

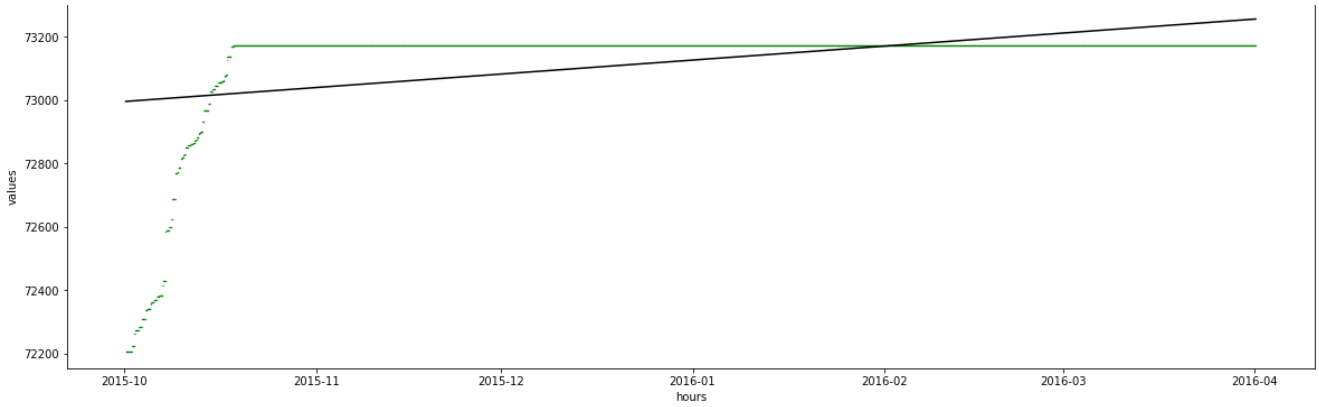

Linear Regression Scatter Plot, Meter ID = 8244
values
hours

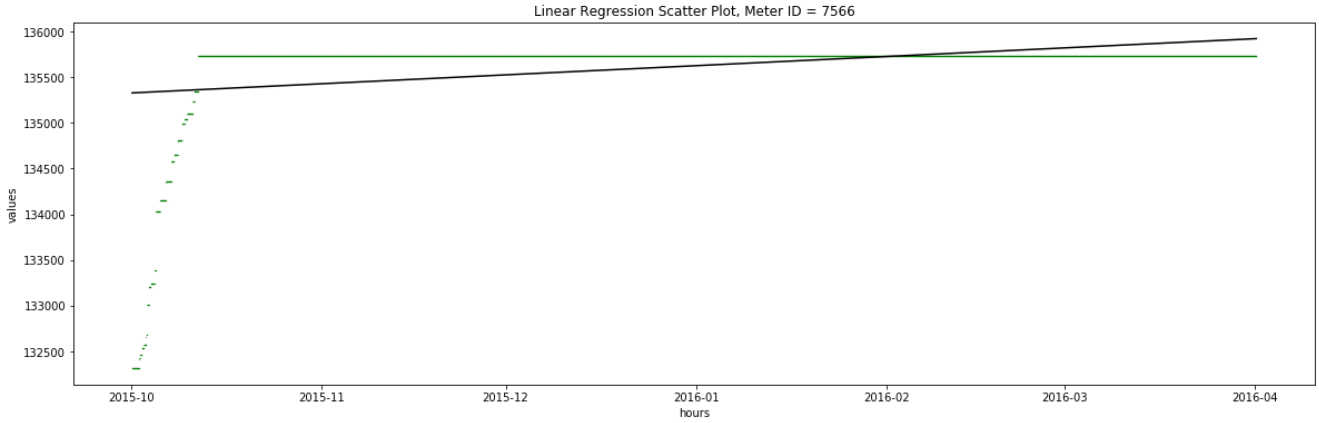

Linear Regression Scatter Plot, Meter ID = 2755
values
hours

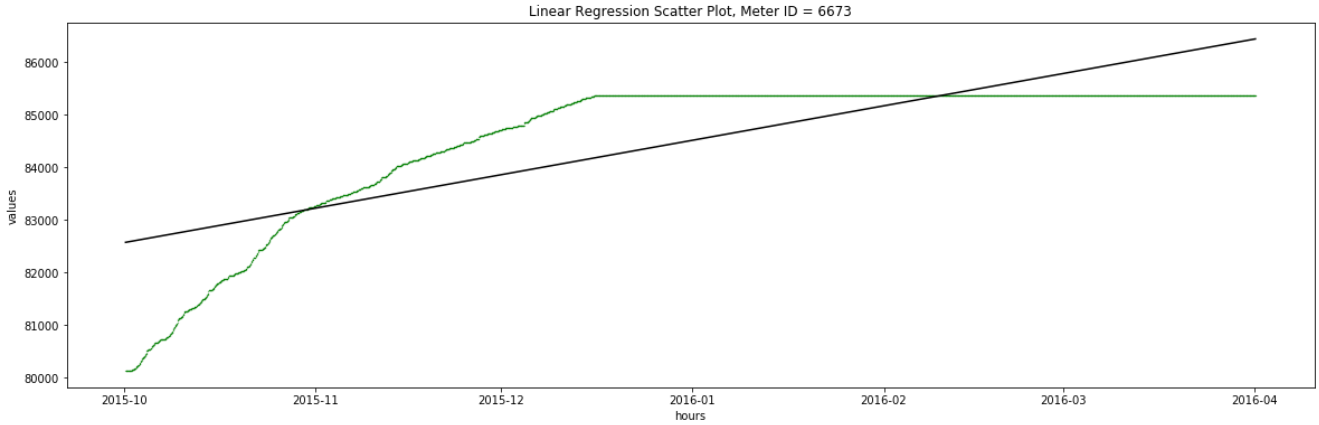

Linear Regression Scatter Plot, Meter ID = 9600
values
hours

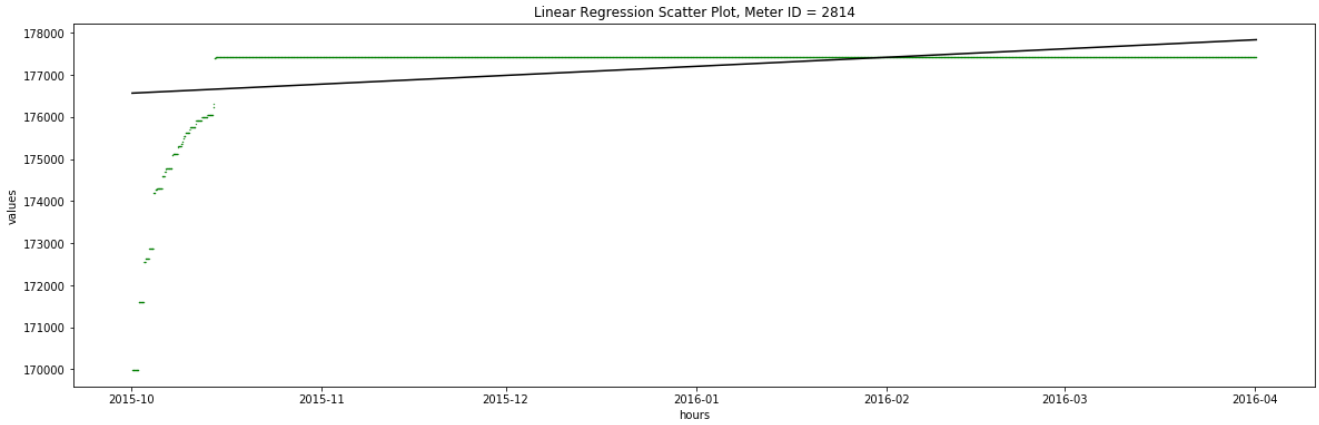

Linear Regression Scatter Plot, Meter ID = 2946
values
hours

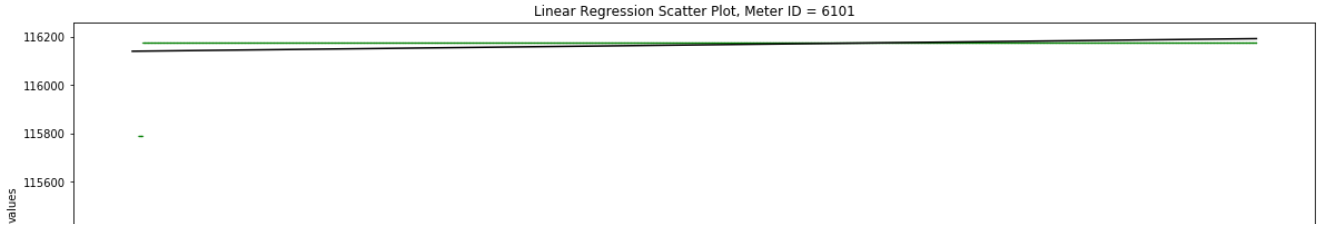

Linear Regression Scatter Plot, Meter ID = 1403

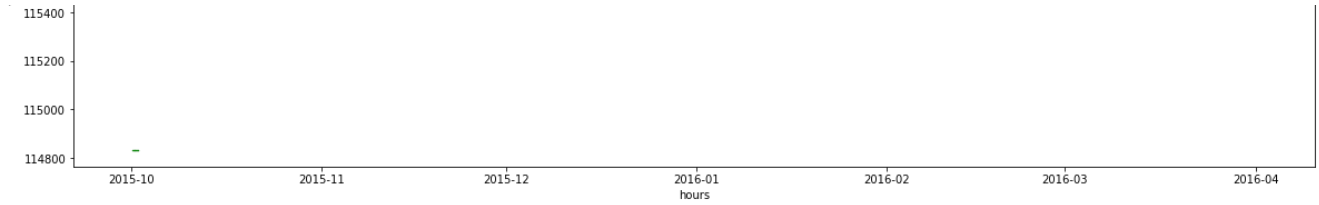

values
hours
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04

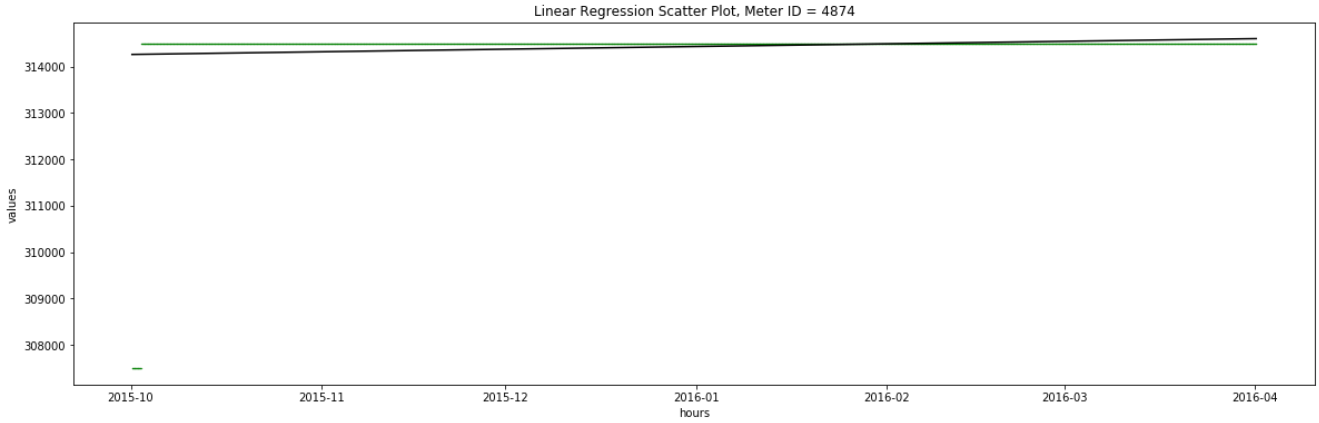

Linear Regression Scatter Plot, Meter ID = 7566
values
hours


Linear Regression Scatter Plot, Meter ID = 6673
values
hours


Linear Regression Scatter Plot, Meter ID = 2814
values
hours


Linear Regression Scatter Plot, Meter ID = 6101
values





Q2.3: Build Support Vector regression to forecast the hourly readings in the future.

Achieving more accurate prediction model through averaging the hourly values between all 157 meters.

As shown in Q2.2, there are multiple meters that are having malfunctioning. Amongst the malfunctioning meters, some jump in meter readings exceptionally. This spike increase may cause the prediction to be inaccurate.

Assumption:

There is a similar pattern in household gas consumptions. Hence taking the average value of all the gas consumption of households every hour gives general hourly consumption thereby reducing the estimation error made by huge jump in values of some meters.

Therefore, by obtaining average hourly consumption value, this general consumption is used to predict the future gas consumption.

1) Modify the hourlyDataDict to hourlyConsumptionDataDict

In [20]:

```
hourlyConsumptionDataDict = {}
for key in hourlyDataDict:
    # all values in hourly data dict is subtraced by the first value
    hourlyConsumptionDataDict[key] = [x - hourlyDataDict[key][0] for x in hourlyDataDict[key]]

# calculate the average across all meter for each hour
avgDataList = []

for i in range(4392):
    total = 0
    for key in hourlyConsumptionDataDict:
        total += hourlyConsumptionDataDict[key][i]
    #print('total is: ' + str(total))
    avg = total/157
    avgDataList.append(avg)
```

2) Build Linear Regression and Support Vector Regression classifier and fit the data

In [21]:

```
hourArray = features_train = np.asarray(range(4392)).reshape(-1,1)

features_train = hourArray
labels_train = np.asarray(avgDataList)
clfLR = LR()
clfLR.fit(features_train, labels_train)
clfSVR = SVR(kernel='linear')
clfSVR.fit(features_train, labels_train)
```

Out[21]:

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='auto',
  kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

3) Generate a list of datetime values for predictions

In [22]:

```
staHr = pd.Timestamp('2015-10-01 05:00:00')

dtStar = datetime.datetime(2015, 10, 1) + datetime.timedelta(hours=5)
dtEnd = datetime.datetime(2016, 4, 1) + datetime.timedelta(hours=4)


#get a list of dateTime from the start to the end\n

dateTimeList = pd.date_range(dtStar, dtEnd, freq = 'H').tolist()
hourList = [[((x - staHr).seconds + (x - staHr).days*86400)/ 3600] for x in dateTimeList]
```

In [23]:

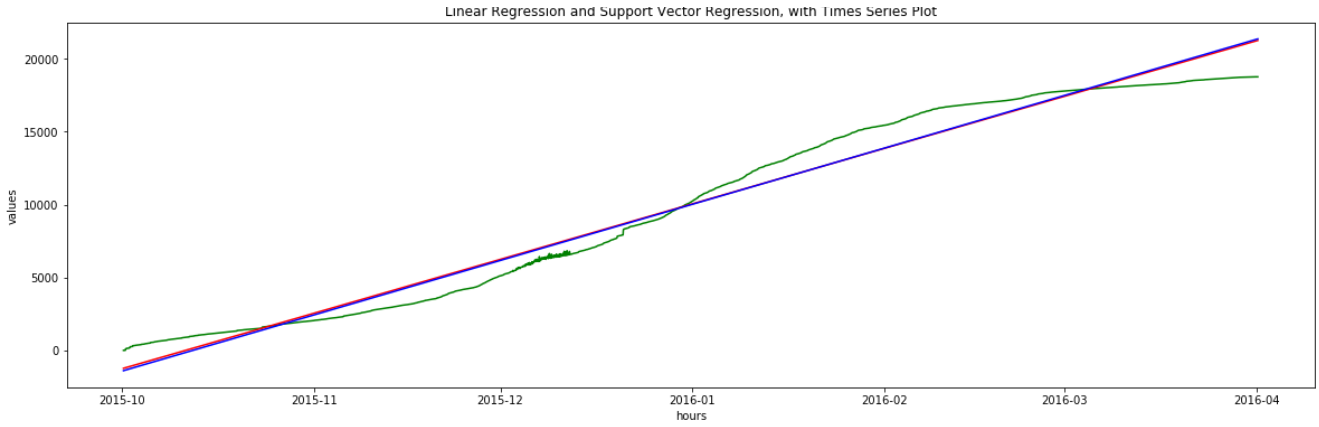
```
staHr = pd.Timestamp('2015-10-01 05:00:00')

dtStar_future = datetime.datetime(2015, 10, 1) + datetime.timedelta(hours=5)
dtEnd_future = datetime.datetime(2016, 10, 1) + datetime.timedelta(hours=4)


#get a list of dateTime from the start to the end\n

dateTimeList_future = pd.date_range(dtStar_future, dtEnd_future, freq = 'H').tolist()
hourList_future = [[((x - staHr).seconds + (x - staHr).days*86400)/ 3600] for x in dateTimeList_future]
```

4) Plot the new Linear Regression and Support Vector Regression on the same plot for comparison

i) Time Series Plot

In [24]:

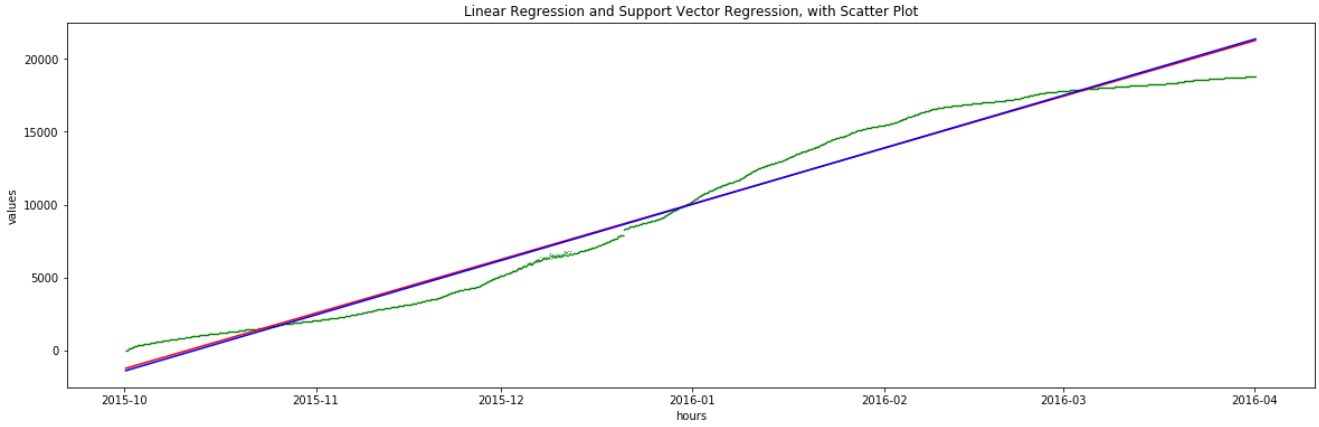
```
fig, ax = plt.subplots(1, 1, figsize=(20, 6))

plt.xlabel('hours')
plt.ylabel('values')
x = dateTimeList
y = np.asarray(avgDataList)

ax.plot(x, y,color='g')
ax.plot(x,  clfLR.predict(hourList) ,color='r')
ax.plot(x,  clfSVR.predict(hourList) ,color='b')
plt.title("Linear Regression and Support Vector Regression, with Times Series Plot")
plt.show()
```



ii) Scatter Plot

In [25]:

```
fig, ax = plt.subplots(1, 1, figsize=(20, 6))

plt.xlabel('hours')
plt.ylabel('values')
x = dateTimeList
y = np.asarray(avgDataList)

ax.scatter(x, y,color='g',s=0.05)
ax.plot(x,  clfLR.predict(hourList) ,color='r')
ax.plot(x,  clfSVR.predict(hourList) ,color='b')
plt.title("Linear Regression and Support Vector Regression, with Scatter Plot")
plt.show()
```



Effect of averaging the data

By averaging the data, the consumption generally increases, looking more like a proper cumulative gas consumption data.

Is Linear Regression & Support Vector Regression the same?

Linear Regression & Support Vector Regression looks almost the same!

Let's find out by predicting the future value, beyond the 6 months data given.

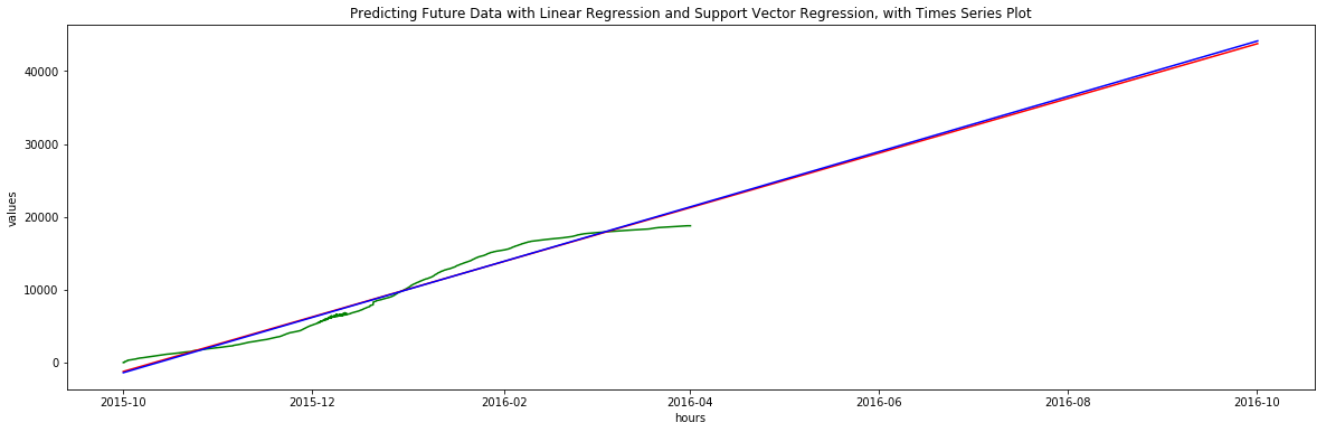
Future Data Prediction

Time series plot is shown for future data prediction.

In [26]:

```
fig, ax = plt.subplots(1, 1, figsize=(20, 6))

plt.xlabel('hours')
```

```
plt.xlabel('hours')
plt.ylabel('values')
x = dateTimeList
y = np.asarray(avgDataList)

ax.plot(x, y,color='g')
ax.plot(dateTimeList_future,  clfLR.predict(hourList_future) ,color='r')
ax.plot(dateTimeList_future,  clfSVR.predict(hourList_future) ,color='b')
plt.title("Predicting Future Data with Linear Regression and Support Vector Regression, with Times Series Plot")
plt.show()
```



From the above future data prediction plot, we can see that the linear regression line and support vector regression line is actually not the same!

Q3.1 Student Proposal

As per section 1.4:

Focus: Find out which meter needs maintenance

Draw histogram to see the number of faulty meters with various magnitude of decreasing value. (Prof. Mehul's recommendation to visualize)

On top of magnitude, no. of times the value decrease can be second histogram to observe.

Set our own standard from the visualization and calculate number of meters in need of maintenance

Follow-up idea:

1) Print a summary to inform the company & consumer about the meter's status. "We recommend maintenance because __."

2) Expected amount of payment for the maintenance. "Expected price: $__. Please dial 1800-900-XXXX"

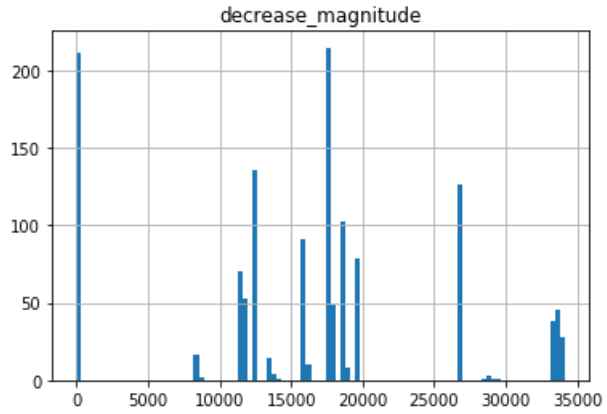
In [180]:

```
# offending_values is the variable storing all "broken meter" values
# That is consecutive readings with decreasing consumption

# Let's format this data for plotting a histogram of
# magnitude of decreasing values, to determine if there is a pattern to the broken meters
# (is there a systematic decrease)?

decreasing_histogram_values = []
for readings in offending_values.values():
    for reading in readings:
        decreasing_histogram_values.append({
            "meter_id": reading[0].dataid,
            "decrease_magnitude": reading[0].meter_value - reading[1].meter_value
        })

decreasing_values_df = pd.DataFrame(decreasing_histogram_values)
```

Histogram of decreasing values for all meters

In [182]:

```
decreasing_values_hist = decreasing_values_df.hist(bins=100, column=['decrease_magnitude'])
```

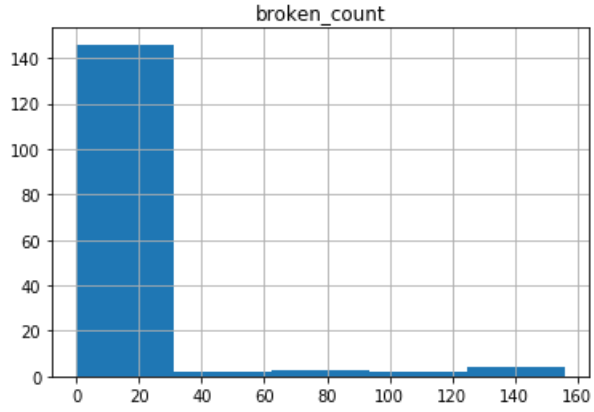


Histogram of number of broken readings for all meters

Meters with no broken readings are added so that we can compare the frequency of broken meters vs non-broken

In [184]:

```
broken_reading_count_df = decreasing_values_df.groupby(['meter_id']).agg(lambda x: x.shape[0]).rename({
'decrease_magnitude': 'broken_count'}, axis='columns')

# Add in meters with no broken readings
appending_values = []
offending_keys = offending_values.keys()
for key in grouped.groups.keys():
    if key not in offending_keys:
        appending_values.append({
            "meter_id": key,
            "broken_count": 0
        })
broken_reading_count_all_df = broken_reading_count_df.append(pd.DataFrame(appending_values).set_index('
meter_id'))

broken_reading_count_all_hist = broken_reading_count_all_df.hist(bins=5)
```



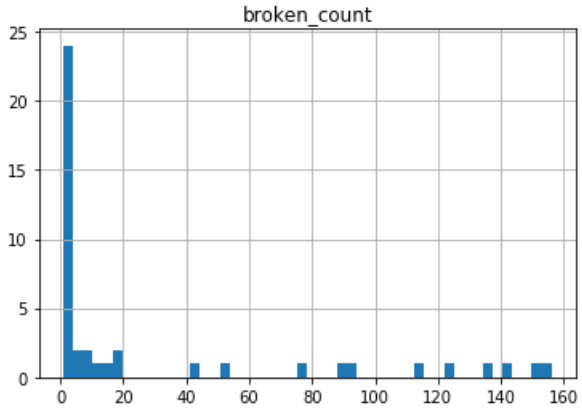
This does not tell us much, since the number of meters with no broken readings heavily skews the plot

Let's try again with just broken meters

Histogram of number of broken readings for meters with broken readings

In [185]:

```
broken_reading_count_hist = broken_reading_count_df.hist(bins=50)
```



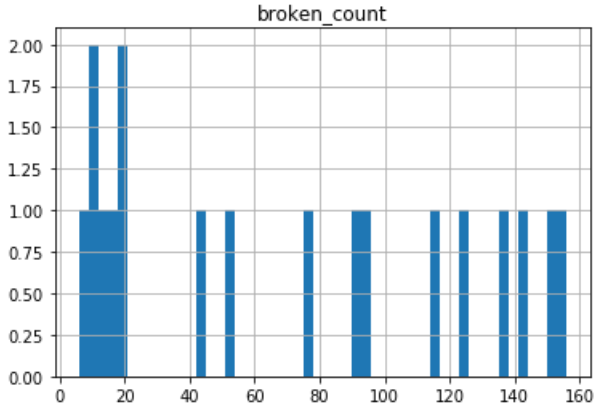
Again, this is not very useful since a majority of the "broken" meters have only a few broken readings (<5)

Let's try again with those with a few readings filtered out

Histogram of number of broken readings for meters with at least 5 broken readings

In [186]:

```
broken_reading_count_filtered_hist = (broken_reading_count_df
                                      .where(broken_reading_count_df['broken_count'] > 5)
                                      .dropna()
                                      .hist(bins=50))
```



With this, it is clear that most meters have only a few (<5) broken readings, or do not actually have any broken readings.

However, this may not be very useful, as some meters may not have reported as many readings

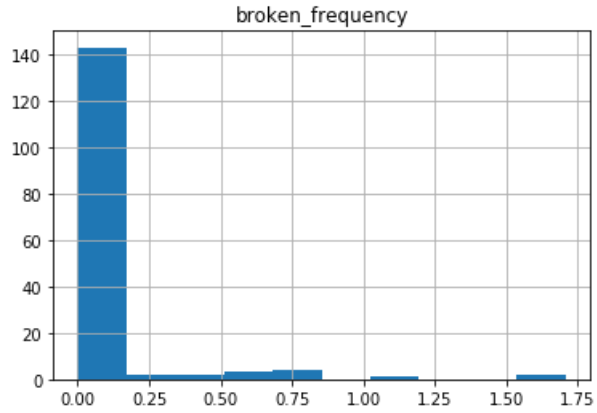
(less values reported results in less "chances" to report a broken reading)

Let's fix this by drawing a histogram of frequency of broken readings for all meters

In [187]:

```
broken_freq_values = []
for key in grouped.groups.keys():
    broken_values = 0
    broken_freq = 0
    total_values = grouped.get_group(key).shape[0]
```

```
    total_values = grouped.get_group(key).shape[0]
    if key in offending_values:
        broken_values = len(offending_values[key])
        broken_freq = broken_values / total_values * 100
    broken_freq_values.append({
        "meter_id": key,
        "broken_readings": broken_values,
        "total_readings": total_values,
        "broken_frequency": broken_freq
    })

broken_freq_df = pd.DataFrame(broken_freq_values).set_index('meter_id')

broken_freq_hist = broken_freq_df.hist(bins=10, column="broken_frequency")
```

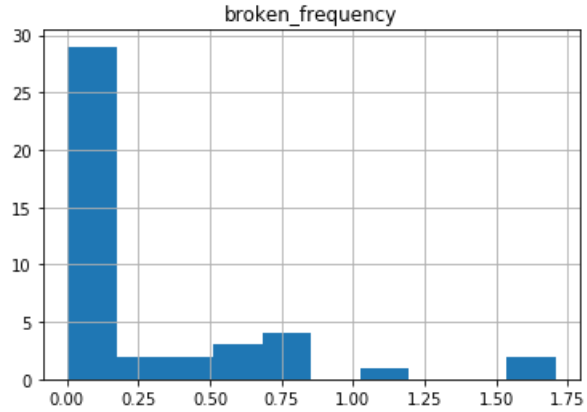


Same as before, this is heavily skewed by meters that are not broken. Let's only look at values > 0

Histogram of frequency of broken readings for meters, with frequency != 0

In [188]:

```
broken_freq_filtered_hist = (broken_freq_df
                             .where(broken_freq_df['broken_frequency'] > 0)
                             .hist(bins = 10, column="broken_frequency"))
```



From our results so far, it is obvious that most of the broken readings come from a few meters

Let's draw a cumulative sum plot to verify this

Cumulative distribution plot of broken readings

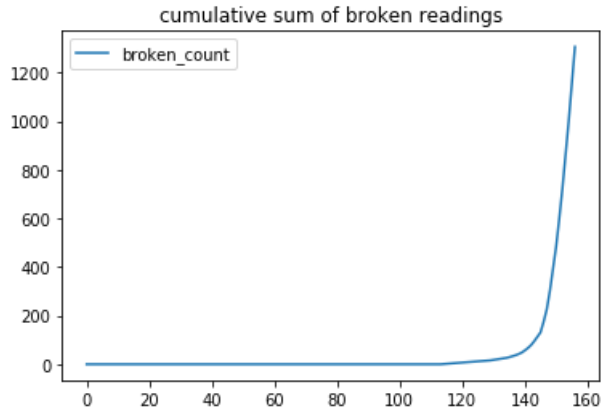
In [189]:

```
broken_readings_cumsum = broken_reading_count_all_df.sort_values(['broken_count']).cumsum()
# change index because meter id is no longer in-order, and then plot cumulative sum
```

```
broken_readings_cumsum.reset_index().plot(y="broken_count", title="cumulative sum of broken readings")
```

Out[189]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3babd3c8>
```



In [190]:

```
### Additional stats to support hypothesis

print ("70% of broken readings: ", 0.7 * broken_readings_cumsum.max().broken_count)
print ("Number of meters required to fill 70% of readings: ",
       broken_readings_cumsum
       .where(broken_readings_cumsum.broken_count
              > 0.7 * broken_readings_cumsum.max().broken_count).dropna().shape[0])
```

```
70% of broken readings:  914.1999999999999
Number of meters required to fill 70% of readings:  3
```

It seems just 3 meters alone can account for > 70% of the broken readings in this dataset

This suggests that fixing just a few meters can drastically reduce the number of broken readings

How does this apply to real life considerations?

There are several factors that affect the ability and desire of a company to repair a meter

Here are some assumptions that we will make, to simplify our modelling process

- There are resources dedicated towards repairing meters
- If unused, they are simply wasted
- As such, the problem becomes one of resource allocation. Where can we direct our resources to maximise their effectiveness?

We have earlier explored this as 3 separate measures

1) Absolute number of broken readings

2) Frequency of broken readings

3) Magnitude (or severity) of broken readings

A scoring system can be implemented, to give each meter a "broken" score across a certain time window. This will then be used to prioritize resources

prioritize resources

What if the company does not have dedicated resources for repairing meters?

- The assumption now is that there is a cost incurred for fixing meters, since there is no dedicated resource to do so
- Thus, the problem is now one of calculating the loss incurred by a meter, from the meter readings
- Same as above, we are relying on the same 3 measures

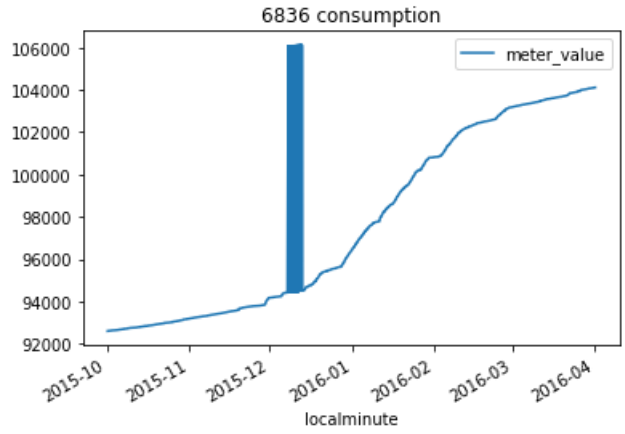
These factors are related because they can be aggregated to calculate the losses incurred from each meter owing to erroneous readings.

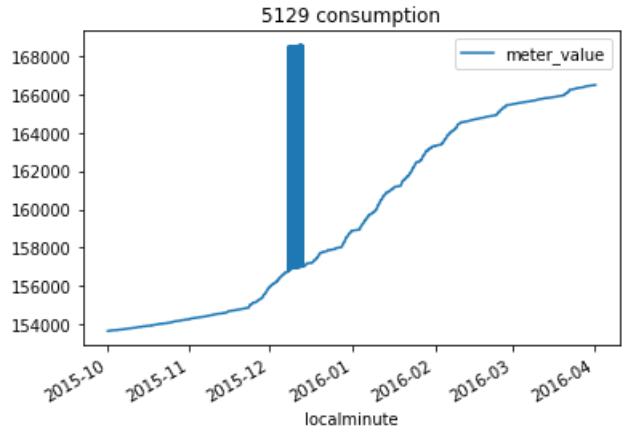
What is missing from this data set is the *actual* gas consumption. With that, we can be sure of the actual extent of misreporting incurred be erroneous readings

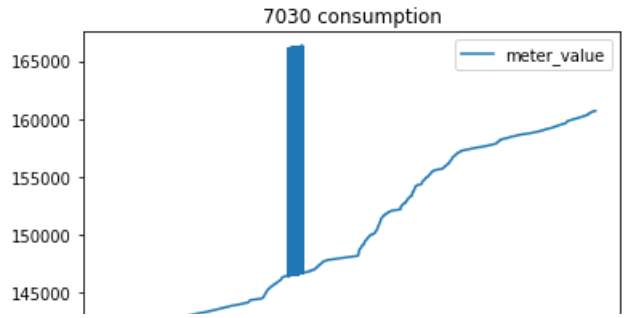
Is there anything else we can realise from looking at our data? Let's look at the ten most broken meters

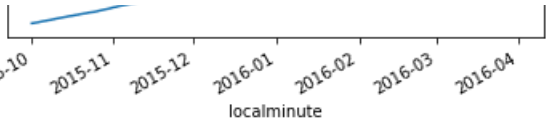
In [191]:

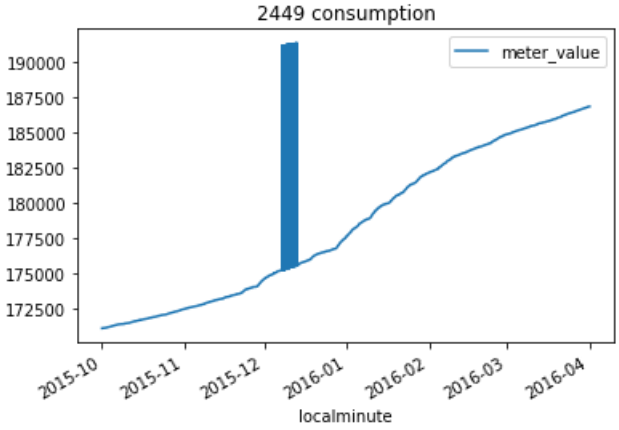
```
# 8156, 5403

ten_most_broken_meter_ids = broken_readings_cumsum.tail(10).index.values
for meter_id in ten_most_broken_meter_ids:
    grouped.get_group(meter_id).plot(x="localminute", y="meter_value", title=str(meter_id)+" consumptio
n")
```

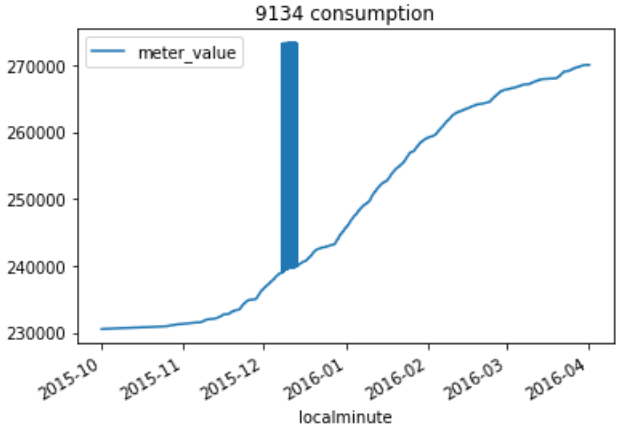


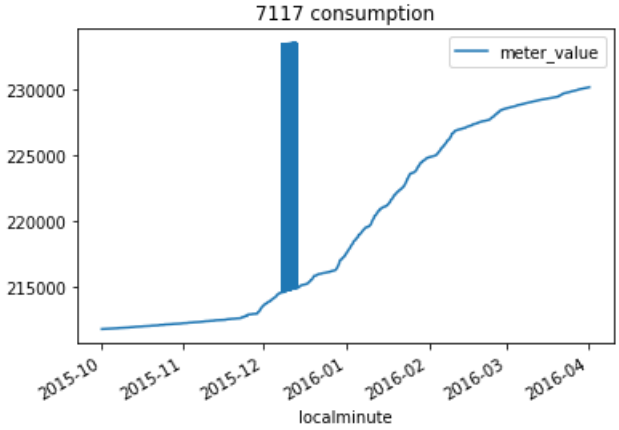


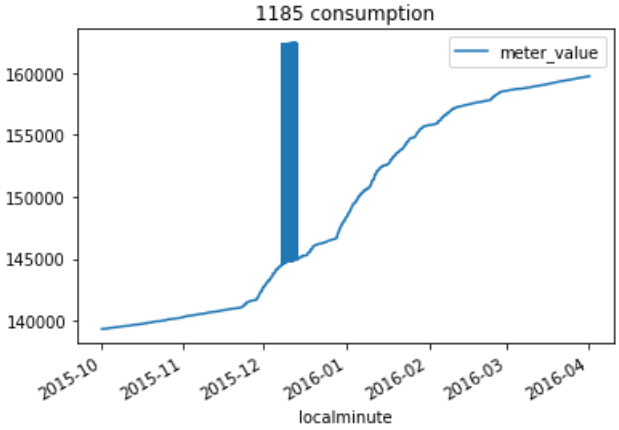


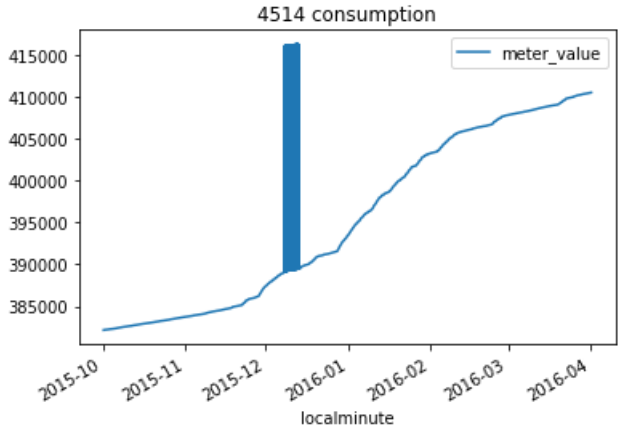

2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
localminute

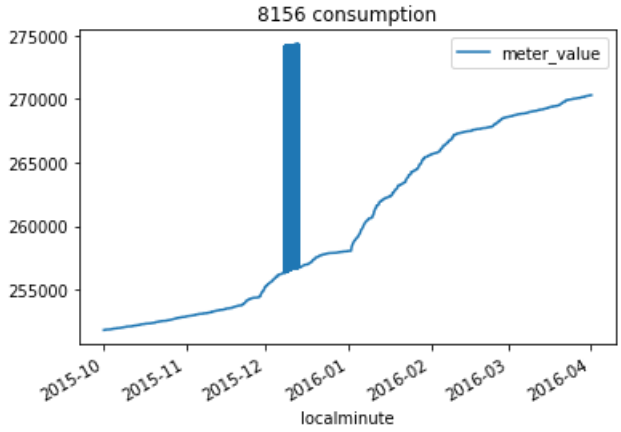

2449 consumption
meter_value
190000
187500
185000
182500
180000
177500
175000
172500
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
localminute

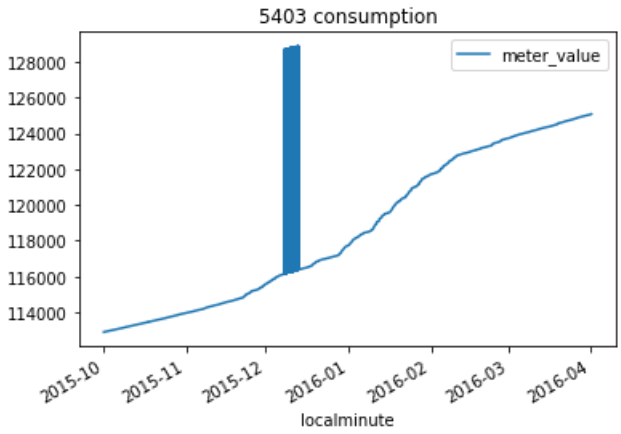

9134 consumption
meter_value
270000
260000
250000
240000
230000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
localminute


7117 consumption
meter_value
230000
225000
220000
215000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
localminute


1185 consumption
meter_value
160000
155000
150000
145000
140000
2015-10
2015-11
2015-12
2016-01
2016-02
2016-03
2016-04
localminute







Breakthrough!

It is clear that most of these broken readings occured during the same timeframe for these meters.

Rather than marking these meters as broken and wasting time fixing them, we can instead seek to understand what caused the broken readings.

- The actual company will need to investigate likely causes during the error timeframe to determine this

Instead of simply marking decreasing readings to determine whether meters are broken, we can additionally check if the decreasing readings are instead a result of spikes, and whether they normalize after some time