ENERGY DEMAND FLEXIBILITY IN ELECTRIC VEHICLES

CSE 523

VISHNU TAMMISHETTI
113082746

INTRODUCTION

Demand Flexibility:

Demand Flexibility refers to the portion of demand in the system that can be shifted within a specific duration.

Demand Flexibility can help make electricity more affordable by helping customers use more power when prices are high. Properly regulated, this will help in making the electricity grids more reliable.

Demand Flexibility in Electric Vehicles:

The world's EV population is growing at an exponential rate. In 2020, the total count of electric vehicles worldwide was about 7 million, which is about a 500% increase since 2016 when it was only about 1.2 million. This trend has only strengthened with world moving towards adopting to electric vehicles as their daily drivers. Car manufacturers investing heavily in the EV market, and the government offering incentives has only resulted in getting more electric vehicles on road.

This increasing number of electric vehicles resulted in an increase in number of charging stations across the world. It is estimated that US alone could end up using 25% more electricity than it does today.

This will invariably increase the stress on the electricity grids. Hence might make them unreliable and comparatively costlier to maintain and to use for customers.

Hence the application of Demand Flexibility in this area will help in properly utilizing the electricity. This can make charging more affordable for EV drivers by helping them charge at a time when the price is low. This will also keep the grid more reliable even with the predicted increase in the number of electric vehicles.

With the trend observed and what is predicted, it is clear that we are moving towards Electric Vehicles and hence, more charging stations will be created and more electricity will be consumed from the grid to serve this increasing demand due to the influx in EVs. Hence Demand Flexibility Analysis in this area will prove to very beneficial in properly utilizing the grid and also in providing affordable charging sessions to the users.

Demand Flexibility in time interval [t, t + delta] is the amount of charge that can be delayed during that particular time interval.

DATA COLLECTION

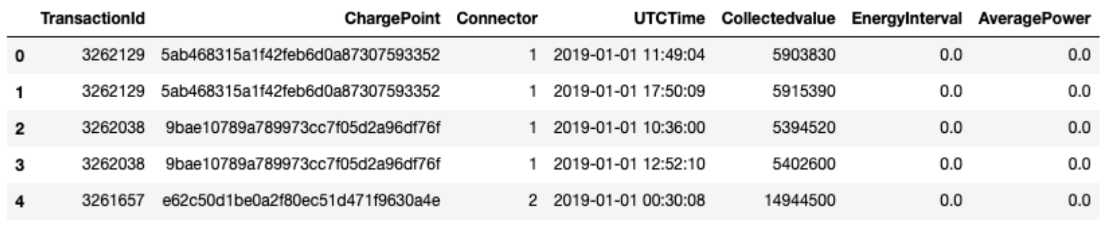
I obtained the ElaadNL data set from a Netherlands based knowledge and innovation center. Link : https://platform.elaad.io/analyses/ElaadNL_opendata.php

I obtained an Excel file with two sheets Meter_Values, and Transactions. I converted them into pandas' dictionary for data wrangling.

Meter Values Dataset:
This dataset had 417141 records, where each record had 7 attributes describing it. The attributes are as follows,

Attribute	Description
Transaction ID	A unique transaction code.
ChargePoint	A unique code of a charging station.
Connector	Many charging stations have two connections (two sockets for charge plugs) and this indicates what connector was used for the transactions.
UTCTime	The instance at which this meter value has been logged (logged in locale time zone).
CollectedValue	The reading on the meter at that instance.
EnergyInterval	Total energy (kWh) transfer between two consecutive meter readings.
AveragePower	Average power in kW between two consecutive meter readings.

First five records of the dataset are as follows,



	TransactionId	ChargePoint	Connector	UTCTime	Collectedvalue	EnergyInterval	AveragePower
0	3262129	5ab468315a1f42feb6d0a87307593352	1	2019-01-01 11:49:04	5903830	0.0	0.0
1	3262129	5ab468315a1f42feb6d0a87307593352	1	2019-01-01 17:50:09	5915390	0.0	0.0
2	3262038	9bae10789a789973cc7f05d2a96df76f	1	2019-01-01 10:36:00	5394520	0.0	0.0
3	3262038	9bae10789a789973cc7f05d2a96df76f	1	2019-01-01 12:52:10	5402600	0.0	0.0
4	3261657	e62c50d1be0a2f80ec51d471f9630a4e	2	2019-01-01 00:30:08	14944500	0.0	0.0

Since this dataset is not containing the information that we need for our analysis, we proceed with the Transactions dataset.

Transactions Dataset:

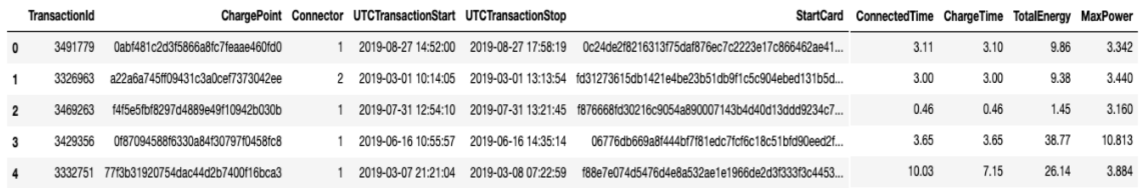
This dataset had data of 10000 unique transactions. Where each transaction has about 10 attributes describing the particular transaction.

Shape: 10000 * 10

The attributes are as follows,

Attribute	Description
Transaction ID	ID The unique transaction code.
ChargePoint ID	The unique code of a charging station.
Connector ID	Many charging stations have two connections (two sockets for charge plugs) and this indicates what connector was used for the transactions.
UTCTransactionStart	The moment the transaction was started (logged in locale time zone).
UTCTransactionStop	The moment the plug was disconnected and the transaction was stopped.
StartCard	The RFID card (hashed) which has been used to start a transaction.
ConnectedTime	Time difference between the start and end of a transaction.
ChargeTime	Total time wherein energy transfer took place.
TotalEnergy	The total energy demand (kWh) per session.
MaxPower	The maximum charging rate (kW) during a session.

First five records are as follows,



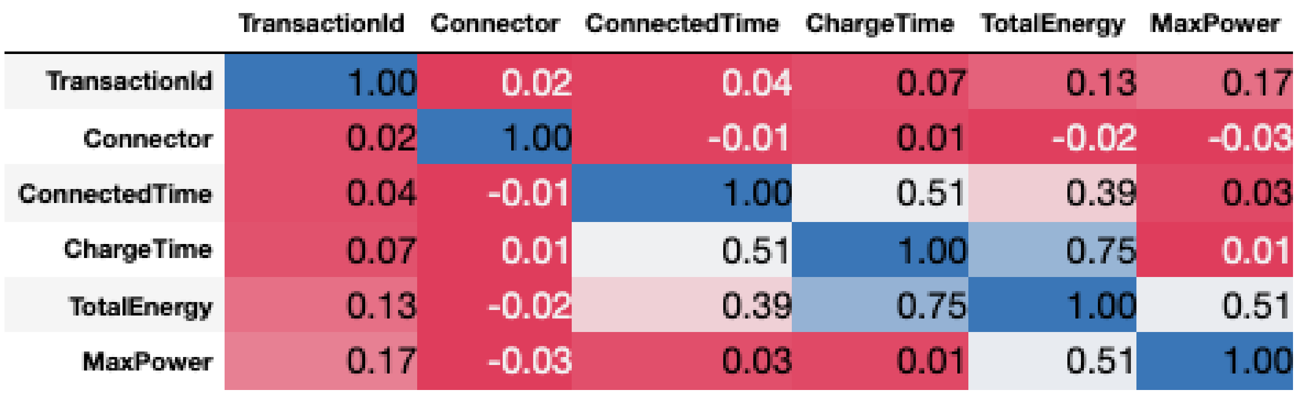
	TransactionId	ChargePoint	Connector	UTCTransactionStart	UTCTransactionStop	StartCard	ConnectedTime	ChargeTime	TotalEnergy	MaxPower
0	3491779	0abf481c2d3f5866a8fc7feaae460fd0	1	2019-08-27 14:52:00	2019-08-27 17:58:19	0c24de2f8216313f75daf876ec7c2223e17c866462ae41...	3.11	3.10	9.86	3.342
1	3326963	a22a6a745ff09431c3a0cef7373042ee	2	2019-03-01 10:14:05	2019-03-01 13:13:54	fd31273615db1421e4be23b51db9f1c5c904ebed131b5d...	3.00	3.00	9.38	3.440
2	3469263	f4f5e5fbf8297d4889e49f10942b030b	1	2019-07-31 12:54:10	2019-07-31 13:21:45	f876668fd30216c9054a890007143b4d40d13ddd9234c7...	0.46	0.46	1.45	3.160
3	3429356	0f87094588f6330a84f30797f0458fc8	1	2019-06-16 10:55:57	2019-06-16 14:35:14	06776db669a8f444bf7f81edc7fcf6c18c51bfd90eed2f...	3.65	3.65	38.77	10.813
4	3332751	77f3b31920754dac44d2b7400f16bca3	1	2019-03-07 21:21:04	2019-03-08 07:22:59	f88e7e074d5476d4e8a532ae1e1966de2d3f333f3c4453...	10.03	7.15	26.14	3.884

Since the attributes UTCTransactionStart, and UTCTransactionStop denote time but are given as strings (Objects in pandas), I converted them into Datatime objects.

DATA ANALYSIS

Correlation Analysis:

I then correlation analysis between the numeric attributes using Pearson Correlation and the results are as follows,

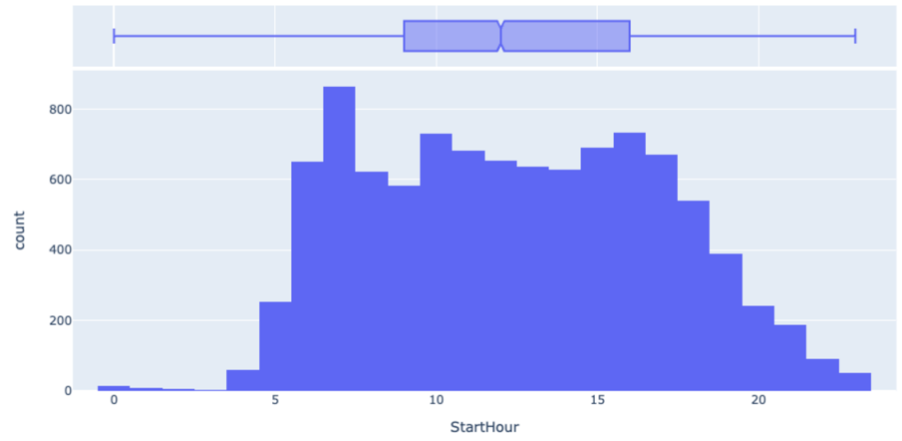


	TransactionId	Connector	ConnectedTime	ChargeTime	TotalEnergy	MaxPower
TransactionId	1.00	0.02	0.04	0.07	0.13	0.17
Connector	0.02	1.00	-0.01	0.01	-0.02	-0.03
ConnectedTime	0.04	-0.01	1.00	0.51	0.39	0.03
ChargeTime	0.07	0.01	0.51	1.00	0.75	0.01
TotalEnergy	0.13	-0.02	0.39	0.75	1.00	0.51
MaxPower	0.17	-0.03	0.03	0.01	0.51	1.00

The highest correlated attributes were TotalEnergy and ChargeTime which are naturally very interlinked. TotalEnergy is the amount of energy that has been consumed and ChargeTime is the time for which charging took place. Correlation between them was 0.75.
This was followed by ConnectedTime and ChargeTime which showed about 0.51 correlation.

StartTime:

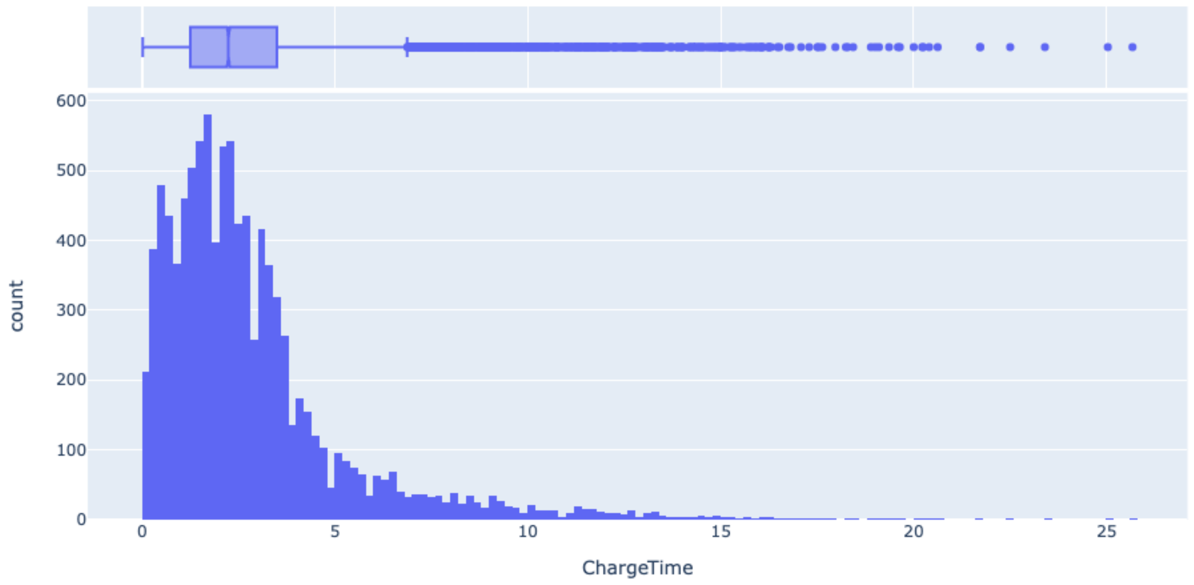
The distribution of the hours at which people started charging is as follows,



People generally started plugging in their chargers starting from 6 AM with most of them at 7 AM. Majority of the distribution lies between 9 AM and 3 PM with very less people starting to charge between 8 PM and 6 AM. This distribution almost resembles a bell curve.

ChargeTime:

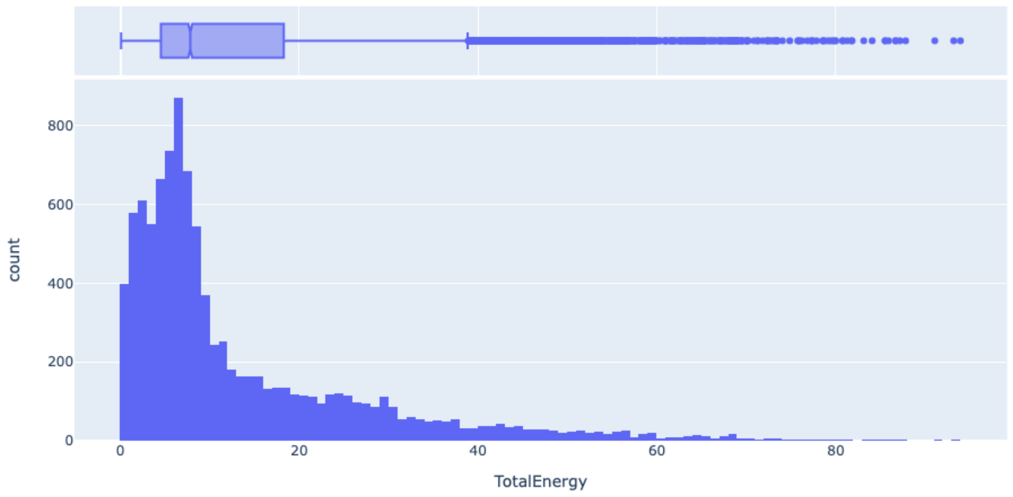
The distribution of the charging times for the 10000 transactions is as follows,



Majority of the sessions show a charging duration of times between 1 hour 15 minutes and 3 and a half hours. While the median lies at 2.24 hours.

TotalEnergy:

The distribution of the total amount of energies consumed during the charging sessions are as follows,



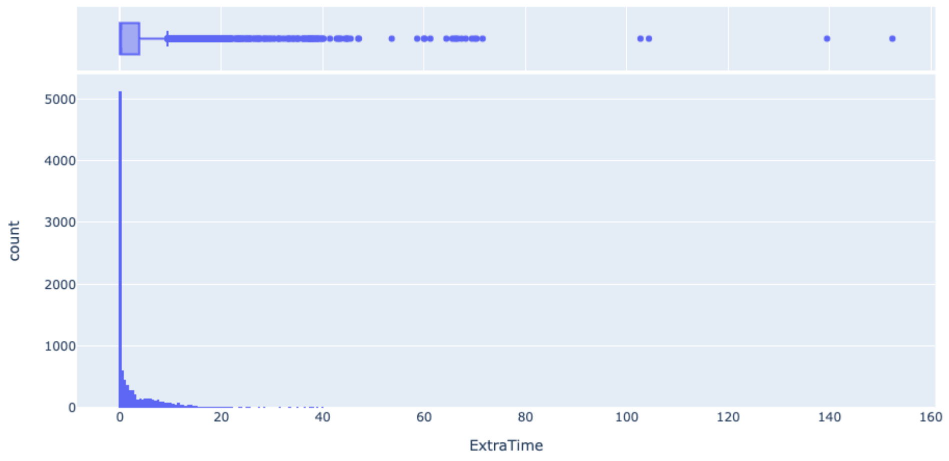
Majority of the sessions consume energies ranging from 4.5 kWh to 18.25 kWh. While the median lies at 7.83 kWh.

ExtraTime:

ExtraTime is a new attribute created from ConnectedTime and ChargeTime. This denotes the amount of time the charger remained connected after the charging is done. This can be called as Idle time as well.

ExtraTime = ConnectedTime – ChargeTime

On plotting the distribution of ExtraTimes we get,



Majority of the Idle times range between 0 to 3.75 hours. While the upper fence of the boxplot is at 9.27 hours. Other than that, many can be considered outliers.

Using the interquartile range, I dropped various outliers for different attributes.

<u>LSTM – PREDICTION</u>

<u>Flexibility calculation:</u>

Flexibility in the time interval [t, t + Δ] is the amount of energy that can be deferred during the time interval.
For each time interval τ in [t, t + Δ], we check if the charging in that slot can be deferred to the time between the time when charging is originally done and the time when the charger is removed. The sum of such flexibilities in the time slots of length τ in time interval [t, t + Δ] is the flexibility in the time interval [t, t + Δ].

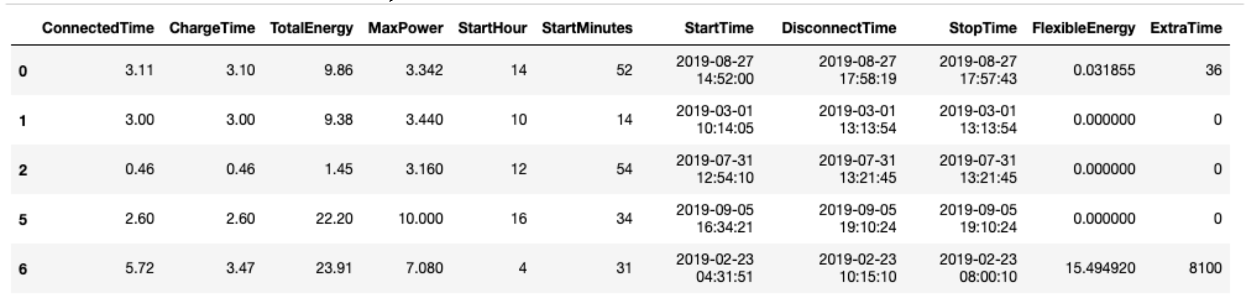
I calculated the total flexibility with parameters,
Δ : 1 hour (60 minutes)
τ : 0.25 hour (15 minutes)

After cleaning the raw dataset, I was left with a dataset which had information like "StartTime", "StopTime", "DisconnectTime", "TotalEnergy".
StartTime -> Denotes the time at which the transaction started.
StopTime -> Denotes the time at which the charging is complete.
DisconnectTime -> Denotes the time at which the user plugs out from the station.

The time between "StopTime" and "DisconnectTime" is the idle time during which no charging takes place, but the charger remains connected.

Beforehand I created a new column called "ExtraTime" for all transactions which denotes the idle time in seconds. [DisconnectTime – StopTime].

The dataset looks like this,



	ConnectedTime	ChargeTime	TotalEnergy	MaxPower	StartHour	StartMinutes	StartTime	DisconnectTime	StopTime	FlexibleEnergy	ExtraTime
0	3.11	3.10	9.86	3.342	14	52	2019-08-27 14:52:00	2019-08-27 17:58:19	2019-08-27 17:57:43	0.031855	36
1	3.00	3.00	9.38	3.440	10	14	2019-03-01 10:14:05	2019-03-01 13:13:54	2019-03-01 13:13:54	0.000000	0
2	0.46	0.46	1.45	3.160	12	54	2019-07-31 12:54:10	2019-07-31 13:21:45	2019-07-31 13:21:45	0.000000	0
5	2.60	2.60	22.20	10.000	16	34	2019-09-05 16:34:21	2019-09-05 19:10:24	2019-09-05 19:10:24	0.000000	0
6	5.72	3.47	23.91	7.080	4	31	2019-02-23 04:31:51	2019-02-23 10:15:10	2019-02-23 08:00:10	15.494920	8100

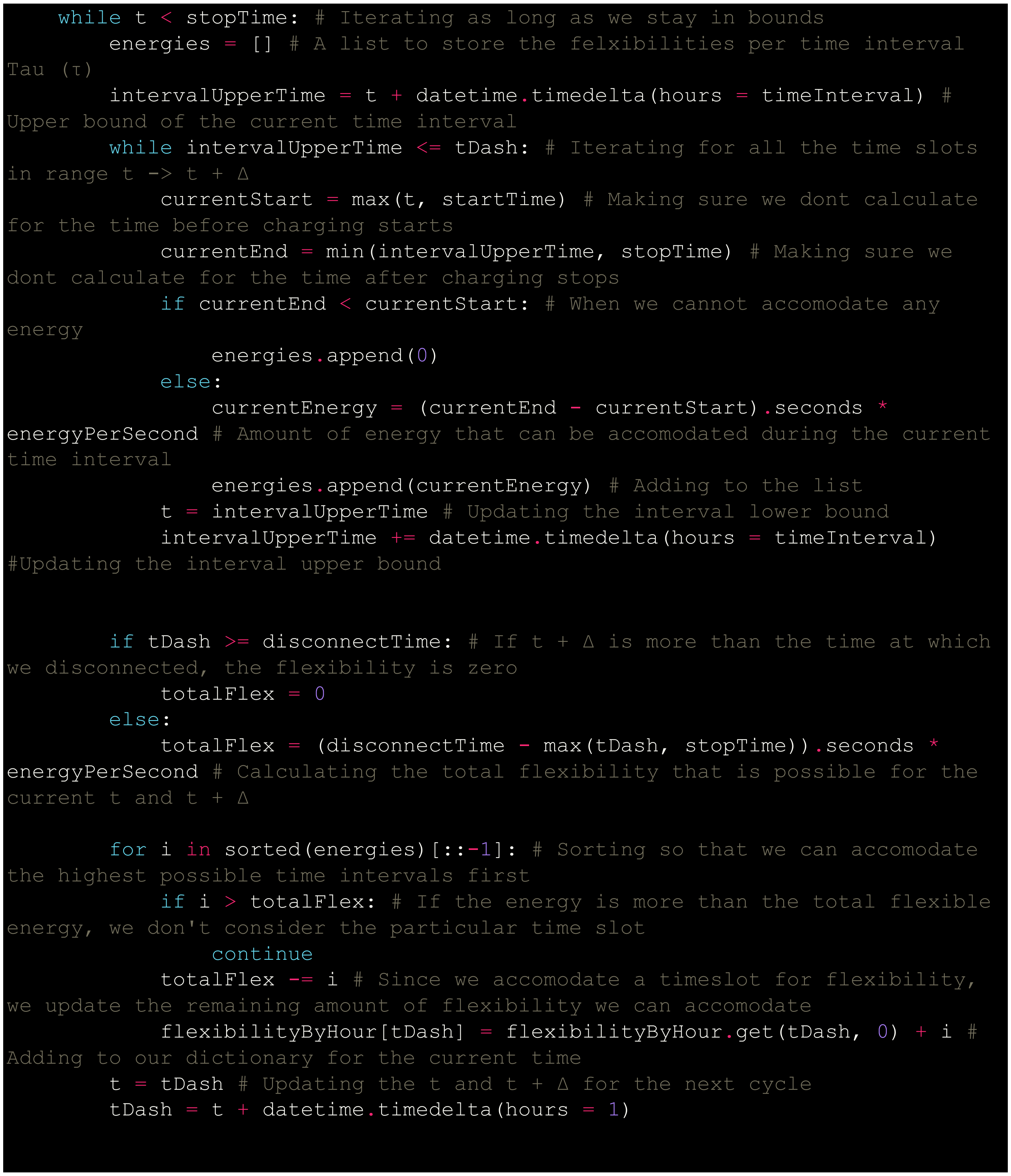
I calculated the flexibility for each hour using the following steps,
I iterated through all the transactions in the database, and for each transaction,

- I calculated the energy consumed per second using the TotalEnergy, StartTime, and StopTime. *[TotalEnergy / [StopTime – StartTime]]*.
- I further calculated the total amount of energy that can be used during this idle time, called MaxFlexibleEnergy. I used the calculated energy rate in the previous step and the idle time to get this value. *[energyPerSecond * Idle time]*.
- I initialized the time t, and using the Δ, I calculated t′ (t + Δ). And for the possible t and t + Δ,
    - For each interval of time τ I calculated the amount of energy that can be deferred.
    - As τ is fifteen minutes, for the interval, I get 4 values of flexibility for the corresponding time intervals of size τ.
    - Using these values, I allocate the flexibilities interval by interval so that no interval's flexibility is overlapped.
    - An interval's flexibility will be the minimum of both the amount of energy that can be consumed during the interval of size τ and the total amount of energy that can be consumed during the idle time.
    - If the amount of energy that can be consumed during this interval is more than the maximum amount of flexible energy that can be allocated, that means we can't delay the power during this time interval.
    - All the energies that are flexible during the 4 time intervals are summed up and they denote the total amount of energy flexible during that time interval.
    - This sum for the time interval is inserted into the keymap that I created earlier.
    - Then, t and t′ are updated and the process continues.
- When iterated for all the transactions in the database, our hash map will be filled with the flexibilities for each hour of each day of the year 2019.

Code: *(Attaching the .ipynb file with comments as well)*

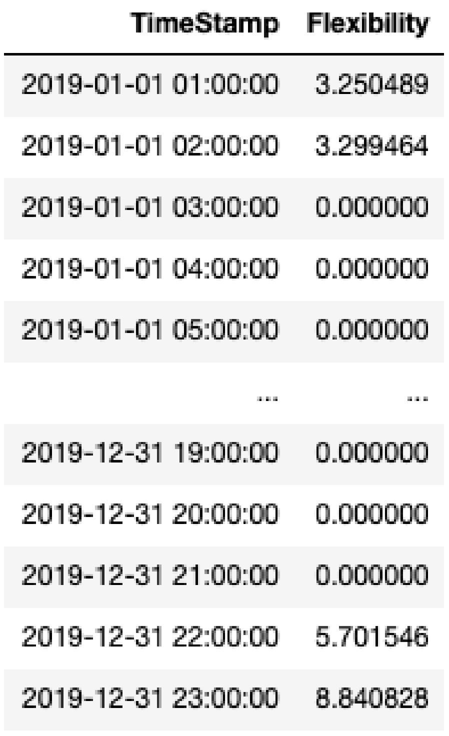
```
'''
My dataframe df has the following columns:
ConnectedTime            Total time connected. In hours.
ChargeTime               Total time charged. In hours.
StartTime                Time at which the car was plugged in. Time when
charging began.
StopTime                 Time at which the charging was completed.
DisconnectTime           Time at which the car was plugged out from the
charging port.
```

```
TotalEnergy                Total energy consumed from StartTime to StopTime.
FlexibleEnergy             The maximum amount of energy that can be consumed in
the idle time.
ExtraTime                  Time between DisconnectTime and StopTime. In
seconds.
'''
'''
I have a dictionary "hourlyConsumption", which stores the following key value
pairs:
Keys: TimeStamp (Hourly timestamps).
Values: Flexible Energy (Total amount of flexible energy in that 1 hour
interval).
'''
delta = 1
timeInterval = 0.25
flexibilityByHour = hourlyConsumption.copy() # Creating a new dictionary
flexibilityByHour to store the flexibilities by hour
for idx, row in df.iterrows(): # Iterating through all the transactions in
the dataframe df.

    totalEnergy = row["TotalEnergy"] # Total energy consumed during this
transaction
    remainingTime = row["ChargeTime"] * 60 * 60 # This is the total time the
charging has taken place. Converting it from hours to seconds.

    maxFlexibleEnergy = row["FlexibleEnergy"] # This is the maximum amount of
flexible energy that can be consumed during the idle time.
    energyPerSecond = totalEnergy / remainingTime # This is the amount of
energy that can be consumed per second

    startTime = row["StartTime"] # The time at which the charging starts
    stopTime = row["StopTime"] # The time at which the charging is done
    disconnectTime = row["DisconnectTime"] # The time at which the charger is
disconnected from the station

    idleTime = row["ExtraTime"] # This is the total amount from the instant
when charging is done to the charger is removed from the slot.
                                # idleTime = startTime -> disconnectTime
    lowerTime = datetime.datetime(year = startTime.year, month =
startTime.month,
                                    day = startTime.day, hour =
startTime.hour)
                                # This is the starting time for our interval.
-> t

    upperTime = lowerTime + datetime.timedelta(hours = delta) # This is the
right bound. t + Δ

    currentTime = startTime # Assigning to currentTime since we are going to
be iterating.
    t = lowerTime # t -> starting time of the interval
    tDash = upperTime # tDash = t + delta
```



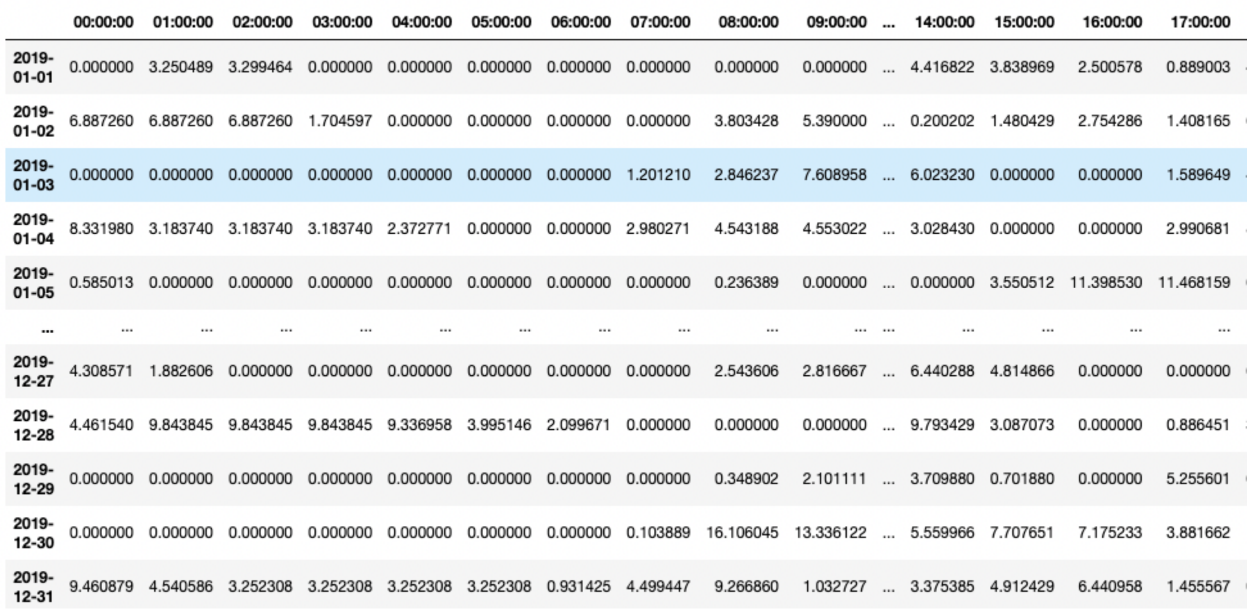
```
    while t < stopTime: # Iterating as long as we stay in bounds
        energies = [] # A list to store the felxibilities per time interval Tau (τ)
        intervalUpperTime = t + datetime.timedelta(hours = timeInterval) # Upper bound of the current time interval
        while intervalUpperTime <= tDash: # Iterating for all the time slots in range t -> t + Δ
            currentStart = max(t, startTime) # Making sure we dont calculate for the time before charging starts
            currentEnd = min(intervalUpperTime, stopTime) # Making sure we dont calculate for the time after charging stops
            if currentEnd < currentStart: # When we cannot accomodate any energy
                energies.append(0)
            else:
                currentEnergy = (currentEnd - currentStart).seconds * energyPerSecond # Amount of energy that can be accomodated during the current time interval
                energies.append(currentEnergy) # Adding to the list
            t = intervalUpperTime # Updating the interval lower bound
            intervalUpperTime += datetime.timedelta(hours = timeInterval) #Updating the interval upper bound


        if tDash >= disconnectTime: # If t + Δ is more than the time at which we disconnected, the flexibility is zero
            totalFlex = 0
        else:
            totalFlex = (disconnectTime - max(tDash, stopTime)).seconds * energyPerSecond # Calculating the total flexibility that is possible for the current t and t + Δ

        for i in sorted(energies)[::-1]: # Sorting so that we can accomodate the highest possible time intervals first
            if i > totalFlex: # If the energy is more than the total flexible energy, we don't consider the particular time slot
                continue
            totalFlex -= i # Since we accomodate a timeslot for flexibility, we update the remaining amount of flexibility we can accomodate
            flexibilityByHour[tDash] = flexibilityByHour.get(tDash, 0) + i # Adding to our dictionary for the current time
        t = tDash # Updating the t and t + Δ for the next cycle
        tDash = t + datetime.timedelta(hours = 1)
```

The resultant data is as follows,



TimeStamp	Flexibility
2019-01-01 01:00:00	3.250489
2019-01-01 02:00:00	3.299464
2019-01-01 03:00:00	0.000000
2019-01-01 04:00:00	0.000000
2019-01-01 05:00:00	0.000000
...	...
2019-12-31 19:00:00	0.000000
2019-12-31 20:00:00	0.000000
2019-12-31 21:00:00	0.000000
2019-12-31 22:00:00	5.701546
2019-12-31 23:00:00	8.840828

I transformed the data into the following format,



	00:00:00	01:00:00	02:00:00	03:00:00	04:00:00	05:00:00	06:00:00	07:00:00	08:00:00	09:00:00	...	14:00:00	15:00:00	16:00:00	17:00:00
2019-01-01	0.000000	3.250489	3.299464	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	4.416822	3.838969	2.500578	0.889003
2019-01-02	6.887260	6.887260	6.887260	1.704597	0.000000	0.000000	0.000000	0.000000	3.803428	5.390000	...	0.200202	1.480429	2.754286	1.408165
2019-01-03	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.201210	2.846237	7.608958	...	6.023230	0.000000	0.000000	1.589649
2019-01-04	8.331980	3.183740	3.183740	3.183740	2.372771	0.000000	0.000000	2.980271	4.543188	4.553022	...	3.028430	0.000000	0.000000	2.990681
2019-01-05	0.585013	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.236389	0.000000	...	0.000000	3.550512	11.398530	11.468159
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2019-12-27	4.308571	1.882606	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.543606	2.816667	...	6.440288	4.814866	0.000000	0.000000
2019-12-28	4.461540	9.843845	9.843845	9.843845	9.336958	3.995146	2.099671	0.000000	0.000000	0.000000	...	9.793429	3.087073	0.000000	0.886451
2019-12-29	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.348902	2.101111	...	3.709880	0.701880	0.000000	5.255601
2019-12-30	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.103889	16.106045	13.336122	...	5.559966	7.707651	7.175233	3.881662
2019-12-31	9.460879	4.540586	3.252308	3.252308	3.252308	3.252308	0.931425	4.499447	9.266860	1.032727	...	3.375385	4.912429	6.440958	1.455567

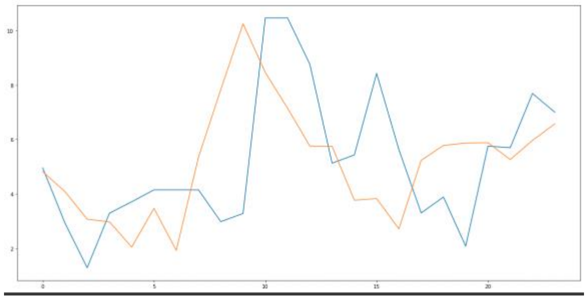
The data frame now has data for 365 days of the year 2019. Each row stores the total amount of flexible energy per each hour of the day (00 -> 23).

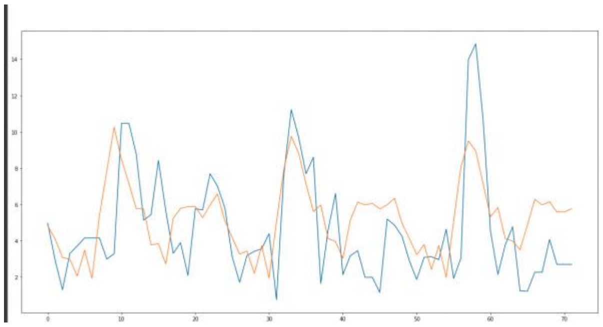
Using this newly created dataset, I tried forecasting the future 24 hours flexibility using LSTM model. (Multi Variate Time Series Forecasting).

I used MinMax sclaer to transform the input values. Scaled values will enable the Neural Network models to perform better.

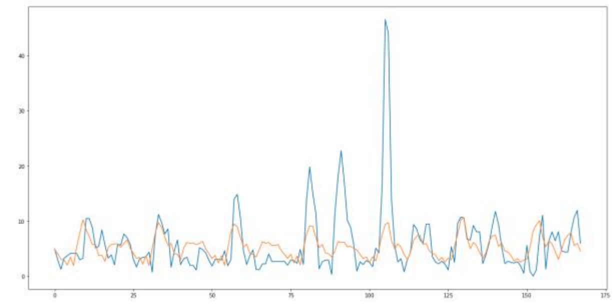
Coming to the model, after trying out many architectures, I created a model with 4 layers with 100 units in each layer. Total parameters that had to be trained were about 293,624. Using huber loss function and training for 25 epochs, the best error I could achieve was in range of **46% - 48%**, while the baseline error was 71%.

Predictions:

- Next 24 hours: (X – Axis: Time vs Y- Axis: Flexibility in KWH)
- 

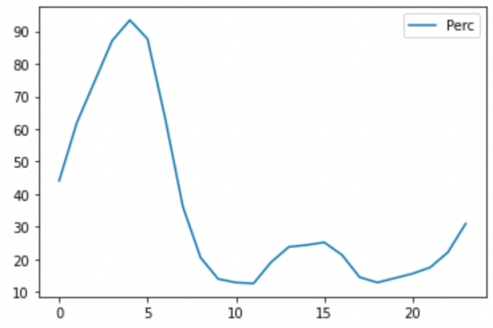
- Next 72 hours: (X – Axis: Time vs Y- Axis: Flexibility in KWH)
- 

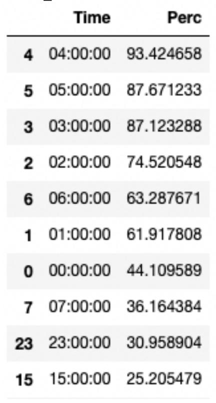
- Next week: (X – Axis: Time vs Y- Axis: Flexibility in KWH)

- 

Performance issues:

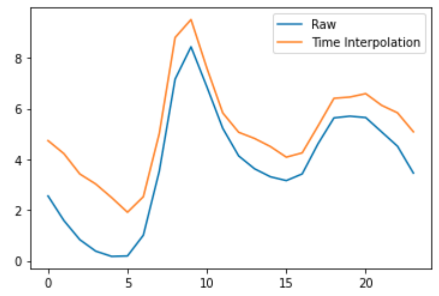
Initially after getting poor results on trying on various base models, I found several issues with the data that I believe can be the cause for this poor performance.

- About 35% of the data had only zeros, which I believe will be very misleading and considerable amount of days were missing.
- In that, several columns had majority of their values as zero.
- 

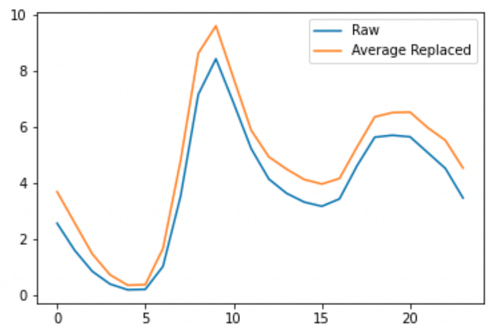
- This is a plot showing the percentage of values that are zero at each hour.
- It is observed that about 93% values for the hour 04:00 are zeros.
- Top ten columns with majority zeros,
- 

	Time	Perc
4	04:00:00	93.424658
5	05:00:00	87.671233
3	03:00:00	87.123288
2	02:00:00	74.520548
6	06:00:00	63.287671
1	01:00:00	61.917808
0	00:00:00	44.109589
7	07:00:00	36.164384
23	23:00:00	30.958904
15	15:00:00	25.205479

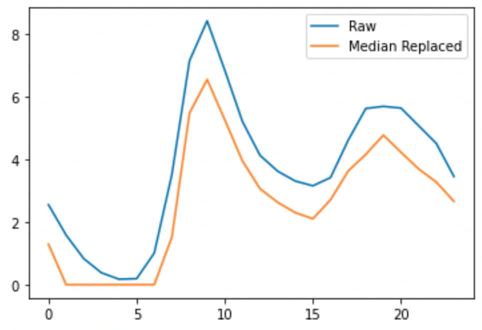
- This resulted in about 36% values being zeros.
- By training the model considering zeros as is, I obtained an error of about 59%

- Then I tried out various interpolation methods and ran the models again.
  Interpolation:
- After Interpolation,
  

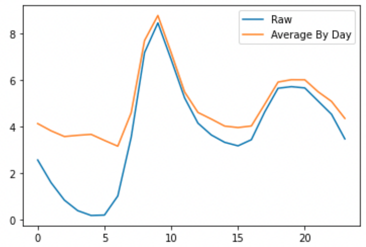
-
- The averages per hour remained almost the same for all the hours except for the early hours.
- Using this, I obtained an error of 55%.

- **Replacing with Average Values:**
  

-
- The averages per hour remained same for all the times in this case.
- Using this data, I obtained an error of 49%.

- **Replacing with Median Values:**
  

-
- Although not as close, the averages per hour mostly still remains close to the raw data.
- Using this data, I obtained an error of 51%

- **Replace with Average of the day:**
- In this approach, I replaced the zeros with the average flexibility that is observed during that day.
- 

- By doing so, the new averages per hour were very close for hours greater than 5:00 but increased for the earlier hours.

- **Using this I obtained an error of about 46%.** Hence, I used this model for the final predictions.

I also acknowledge that replacing the zeros might not be the ideal approach.

- It can be possible that based on the data we have, the flexibility during the early hours of the day is very low indeed and hence zeros.
- However, this brings me to another problem that I believe caused this poor performance, that is very less data.
- This dataset has data for only 365 days, that is year 2019 and considerable amount of values were missing.
- I believe this amount data is very less to train an RNN and for Multi Variate Time Series Forecasting.
- Again, splitting the data into train and test datasets further decreases the train size.
- Hence, I believe the models are failing to pick up on a pattern if there is any in the dataset. Provided more data, I believe we can obtain better results.

Conclusion:

I was able to calculate flexibility for each hour for each day in the year 2019. I was able to train a LSTM model for day-ahead-predictions of flexibility and achieved an error of 46% while the baseline stands at 71% error.

Though the model beats the baseline standards, there is still room for improvement provided we have more data.