Energy Forecast Project

Deep Learning Course - MSIT 2020

Author(s): Pranav Surampudi

Table of Contents

Abstract:

The Problem at hand is forecasting the electricity consumption of the top household(s) with the highest number of samples on an hourly basis based on the previous usage pattern. As on date the prediction of household energy consumption (domestic usage) is usually done at production level based on mathematical models and there isn't model to estimate the usage and if possible, predict it beforehand at a single customer level. So technically it's uncharted territory and thus carries some novelty.

The problem statement can be classified as a Time Series Forecasting problem. The energy consumption varies with time and an hourly analysis has to be performed on the data in such a way that will lead to the prediction of the usage. More specifically it is a Multi-Series Problem. The scope of the data provided is just big enough for the forecast to be possible. Although the other data elements provided were found out to be less related to the power usage, the dataset provided has the scope wide enough to perform exploratory analysis.

General assumptions include

- Time spent by people in the house is constant.
- No unlikely events like Blackouts are considered.

Technical assumptions include

- No faulty data from the Smart Meters.
- Weather data throughout the data is constant.

A solution for this problem is to perform Time Series Forecasting on the data using an Artificial Neural Network. Using Recurrent Neural Networks (RNNs), data which varies over time can be predicted.

If done properly we can estimate the power usage for a consumer and give insights into things like

1. How does peak power usage vary with time
2. Develop strategies to minimize costs etc.

The biggest risks would be the model being over or underfit and the model not being able to generalize patterns of power usage.

As far as costs are concerned, there are none as the data is available at free of cost and I have a GPU in my machine to build and test neural networks.

There are fixed metrics defined to quantitatively measure the correctness of the model that has been developed.

Problem Statement:

The Problem at hand is forecasting the electricity consumption of the top household(s) with the highest number of samples on an hourly basis based on the previous usage pattern. As on date, the prediction of household energy consumption (domestic usage) is usually done at production level based on mathematical models and there isn't a model to estimate the usage and if possible, predict it beforehand at a single customer level. So technically it's uncharted territory and thus carries some novelty.

Motivation:

It is a common sight in summers in India that the electricity bill shoots up and in the winter it dips low. So if we can predict the power usage we can translate that monetary benefit for the average consumer. Because the cost per unit(KiloWatt Hour) is fixed for a domestic customer so that becomes very easy to save money.

Related Work:

Most of the existing solutions to this problem statement tend to use ARIMA (Auto-Regressive Integrated Moving Average) and the one's using RNN are likely to end up using LSTM with very basic configuration leading to a comparatively high error rate(RMSE, MAE), my approach would be to use a well-configured stateful(preferred when data is one long time series), GPU-Optimized Recurrent Neural Network with LSTM hidden layers which would then be trained with a mini-batch gradient descent which seems optimal for the data.

The solution proposed in my abstract is limited by the limitations of data which is the reason I've decided not to include an autoencoder block in the architecture. The model proposed in the paper[1] with state explainable autoencoder(Jin et al.) is defined to match the complex feature space(Reactive Power, Active Power, Voltage, etc) of the dataset considered which consists of 2 million data points with a 2-minute gap over 4 years and the wide range of demand predictions taken into consideration (15, 30, 45, 60 minutes). The data associated with the problem statement I chose was from Kaggle and has a single stream of energy consumption for each house with a data point for every half hour making the total data points to be around 75k for each of the top 3 households. It required an hourly prediction and moreover a bare-bones

implementation of the proposed model yielded a lower MSE than that of the state explainable auto-encoder method specified in the abstract of the paper[1].

[1] Electric Energy Consumption Prediction by Deep Learning with State Explainable Autoencoder, February 2019, DOI: 10.3390/en12040739 Jin-Young Kim and Sung-Bae Cho

https://www.researchgate.net/publication/331291419_Electric_Energy_Consumption_Prediction_by_Deep_Learning_with_State_Explainable_Autoencoder/fulltext/5c70a214a6fdcc47159424b9/331291419_Electric_Energy_Consumption_Prediction_by_Deep_Learning_with_State_Explainable_Autoencoder.pdf?origin=publication_detail

Data:

The Data is tabular and thus is structured by nature. The source of data is the LONDON Data Store that is maintained by the Government of UK and is free for all to use for all intents and purposes. The Data is numeric is nature is ordered by time and hence the name time series data. The Actual Dataset proposed in the paper[1] is a much larger one and is complex, thus for the scope of this project I have picked up a chunk of similar data that is easier to handle and yet allows me to effectively demonstrate the use of algorithms to achieve realistic results.

Following are the fields present in the dataset

	LCLid	stdorToU	DateTime	KWh	Acorn	Acorn_grouped
0	MAC000002	Std	2012-10-12 00:30:00.0000000	0.0	ACORN-A	Affluent
1	MAC000002	Std	2012-10-12 01:00:00.0000000	0.0	ACORN-A	Affluent
2	MAC000002	Std	2012-10-12 01:30:00.0000000	0.0	ACORN-A	Affluent
3	MAC000002	Std	2012-10-12 02:00:00.0000000	0.0	ACORN-A	Affluent
4	MAC000002	Std	2012-10-12 02:30:00.0000000	0.0	ACORN-A	Affluent

Fig 1. Structure of Data

The total number of tuples in the dataset is 999971. The variable that is up for prediction is KWh. There arent any null or missing values in this dataset and thus no preprocessing steps are required as such. In order to speed up the training process for the RNN-LSTM, I have normalized the data, another advantage of doing that is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values

Methods:

Three methods have been used in the context of this project to find a feasible solution to the problem statement, these include

1. Linear Regression
2. ARIMA - Autoregressive Integrated Moving Average
3. LSTM - Long Short Term Memory

Since this the data is continuous, the problem is clearly a regression task the elementary algorithm to do just that is linear regression. But in the end, the LSTM method netted better results and it doesn't require the arduous task of feature selection.

ARIMA, on the other hand, provides another approach to time series forecasting. Exponential smoothing and ARIMA models are the two most widely used approaches to time series forecasting and provide complementary approaches to the problem. While exponential smoothing models are based on a description of the trend and seasonality in the data, ARIMA models aim to describe the autocorrelations in the data.

The AR part of ARIMA indicates that the evolving variable of interest is regressed on its own lagged (i.e., prior) values. The MA part indicates that the regression error is actually a linear combination of error terms whose values occurred contemporaneously and at various times in the past.

The I (for "integrated") indicates that the data values have been replaced with the difference between their values and the previous values (and this differencing process may have been performed more than once). The purpose of each of these features is to make the model fit the data as well as possible.

Plausible Solutions - Justification for using DL:

I have used simple linear regression to solve this problem but it's just that there are specific tools that only work for time series data that sometimes do a better job.
The main argument against using linear regression for time series data is that we're usually interested in predicting the future, which would be extrapolation (prediction outside the range of the data) for linear regression. Extrapolating linear regression is seldom reliable. However, many popular time series models do not extrapolate reliably either.

Autocorrelation is nothing special for time series. With any linear regression, we can do a “residual versus fitted” chart to see if the residual shows any pattern with respect to your independent variable. We have a special word for those patterns if the independent variable is time, but the issue is the same for any type of data.

A lot of plausible solutions with a statistical modelling approach such as ARIMA or other methods based on weighted averages were proposed. Typically DL methods such as LSTM tend to outperform classic statistical models given enough data. In addition to that statistical models tend to focus on samples of data that are representative of the population rather than the population itself, DL, on the other hand, takes all of the data into consideration before picking up patterns.
This project proposes a deep learning approach(LSTM) which is able to pick up long-range patterns given enough data. An illustration of the proposed model can be found below.

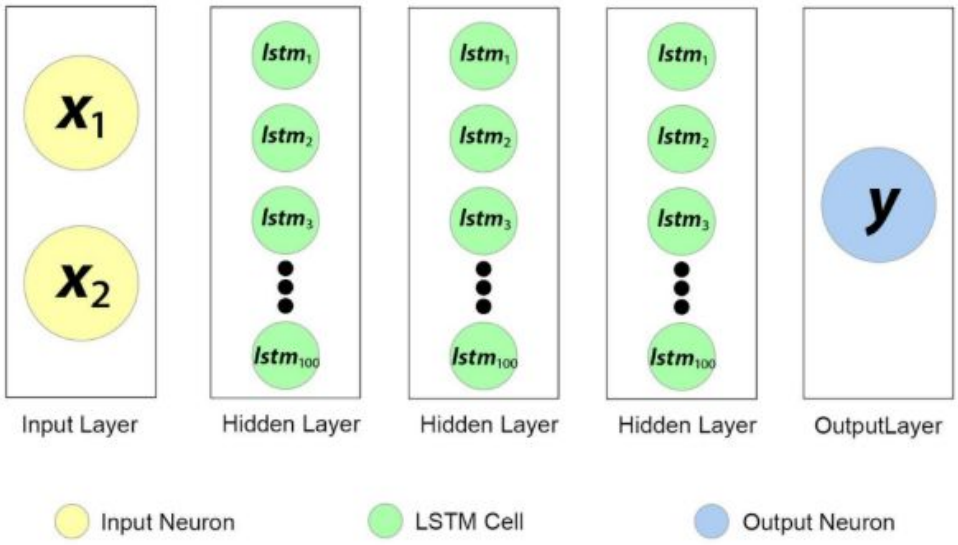


Fig 2. LSTM Representation

Experiments:

As mentioned above three different models are constructed using three different approaches and the results are listed in the latter portion of this report. The first two approaches are the stock versions of the specified algorithms and didn’t require much tweaking, the LSTM required careful considerations in terms of Neural Architecture and the hyperparameters used to build, train and test the model. The models are as follows

1. **Linear Regression**

Using the raw output () of a linear model as the actual prediction in a regression model. The goal of a regression problem is to make a real-valued prediction. For example, if the raw output () of a linear model is 8.37, then the prediction is 8.37.

Contrast linear regression with logistic regression. Also, contrast regression with classification.

The Scikit Learn implementation of linear regression was used to fit the dataset as mentioned in the data section of this report. Results for the same can be found in the results section.

```
reg = LinearRegression()
reg.fit(X_train, y_train.reshape(-1,1))
```

Fig 3. Linear Regression Model

2. ARIMA

The ARIMA predictor of Statsmodels library is used to build the following model. The metrics and results of this model can be found below.

```
model = ARIMA(top_usage['KWh'].values[:X_train.shape[0]], order=(5,1,0))
model_fit = model.fit(disp=0)
print(model_fit.summary())
```

Fig 4. ARIMA Model

3. LSTM

A type of cell in a recurrent neural network used to process sequences of data in applications such as handwriting recognition, machine translation, and image captioning. LSTMs address the vanishing gradient problem that occurs when training RNNs due to long data sequences by maintaining history in an internal memory state based on new input and context from previous cells in the RNN.

```
# Building the model
regressor_mae = tf.keras.Sequential()
regressor_mae.add(tf.keras.layers.CuDNNLSTM(100, stateful=True, return_sequences=True, input_shape=(2, series), batch_size = batc
regressor_mae.add(tf.keras.layers.CuDNNLSTM(100, stateful=True, return_sequences=True))
regressor_mae.add(tf.keras.layers.CuDNNLSTM(100, stateful=True, return_sequences=True))
regressor_mae.add(tf.keras.layers.Dense(1))
regressor_mae.compile(optimizer='adam', loss = 'mae')
regressor_mae.summary()
```

```
WARNING:tensorflow:From c:\program files\python36\lib\site-packages\tensorflow_core\python\ops\resource_variable_ops.py:1630: c
alling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will
be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
cu_dnnlstm (CuDNNLSTM)       (32, 2, 100)              41200
_________________________________________________________________
cu_dnnlstm_1 (CuDNNLSTM)     (32, 2, 100)              80800
_________________________________________________________________
cu_dnnlstm_2 (CuDNNLSTM)     (32, 2, 100)              80800
_________________________________________________________________
dense (Dense)                (32, 2, 1)                101
=================================================================
Total params: 202,901
Trainable params: 202,901
Non-trainable params: 0
_________________________________________________________________
```

Fig 5. LSTM Model

Hyper Parameters of LSTM

Note: A lot of hyperparameters were hand picked (trial and error) to increase accuracy instead of a tuning engine due to lack of resources

I. Epochs

A full training pass over the entire dataset such that each example has been seen once. Thus, an epoch represents N/batch size training iterations, where N is the total number of examples.

The current value of this hyperparameter in the current model is **12,** this value is hand-picked (early stopping).

II. Batch Size

The number of examples in a batch. For example, the batch size of SGD is 1, while the batch size of a mini-batch is usually between 10 and 1000. Batch size is usually fixed during training and inference; however, TensorFlow does permit dynamic batch sizes.

The value of this parameter in the current context is **32.** Batch size is a mandatory hyperparameter for stateful LSTMS, the chosen value is the largest possible batch of training data that can fit in the V-RAM of the GPU.

III. Number of Layers

This parameter is the number of hidden layers in the Neural Network. The Number of layers in the current setup is **3.** This value is hand-picked after trial and error.

IV. Number of Cells Per Layer

The number of LSTM cells per each hidden layer. This is an arbitrary value as it has less impact on the overall accuracy of the model. The value chosen in this case is **100.**

V. Optimizer - ADAM (Adaptive Moment Estimation)

The following are the default values provided by tf.keras module. These values were suggested by the very research paper[2] that has proposed adam optimizer.

[2] ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION, Diederik P. Kingma* et al
https://arxiv.org/pdf/1412.6980.pdf

i. Learning Rate (α)

A scalar used to train a model via gradient descent. During each iteration, the gradient descent algorithm multiplies the learning rate by the gradient. The resulting product is called the gradient step. Learning rate is a key hyperparameter.

Value: `0.001`

ii. Exponential Weighted Moving Average Variables (β1, β2)

In statistics, a moving average (rolling average or running average) is a calculation to analyze data points by creating a series of averages of different subsets of the full data set. It is also called a moving mean (MM)[1] or rolling mean and is a type of finite impulse response filter. Variations include: simple, and cumulative, or weighted forms

Values:
β1 = `0.9`
β2 = `0.999`

iii. Epsilon

It is an infinitesimal threshold value used especially to avoid division by zero.

Value:

ε = `1e-07`

Findings / Results:

Deep learning is the right fit for this problem and the proposed model could generalize well over the usage habits of the household. Training on a GPU yielded better results over that of CPU. However, it is observed that data regarding additional parameters which might affect the power consumption such as weather (which was assumed constant) might provide even better performance.

Here's a representation of the Training Loss during the course of the training process

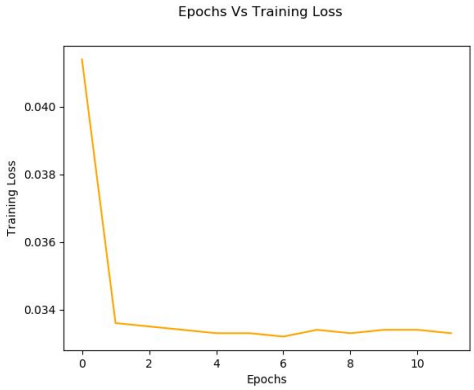


Fig 6. Epochs Vs Training Loss

We can clearly observe that the loss has steadily decreased and has settled at around 0.034. Also here is a comparison of the Actual Vs Predicted values when the model is evaluated on the Testing Set.

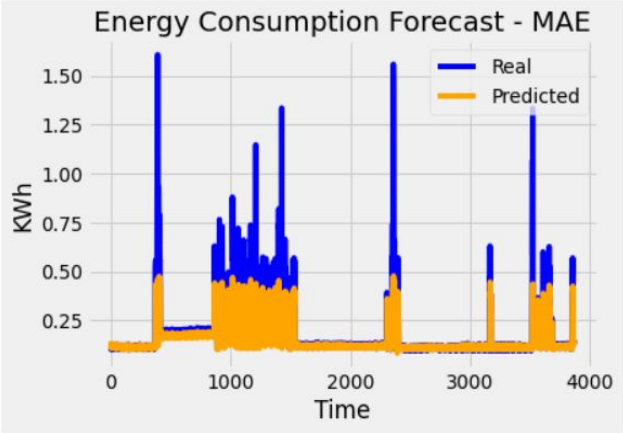


Fig 7. LSTM Model Performance

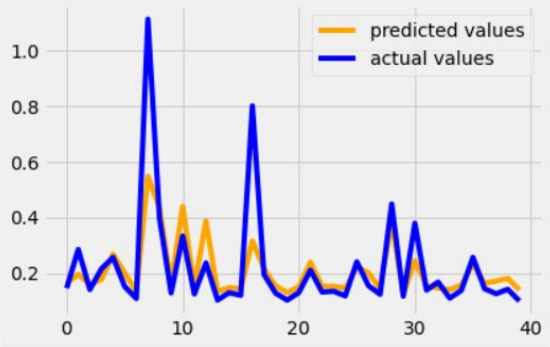


Fig 8. Linear Regression Performance

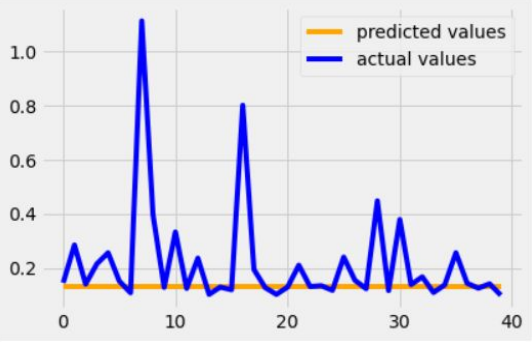


Fig 9. ARIMA Performance

The above figures represent the performance of the first two models on the test set, one astute observation is that in the case of ARIMA the line for prediction is rather flat (y = k) this is because the ARIMA model has under fitted the data provided to it. Linear regression does a decent job but from the metrics data available in the latter portion of this report we can state that LSTM does nearly 50 - 60% better in all of the metrics.

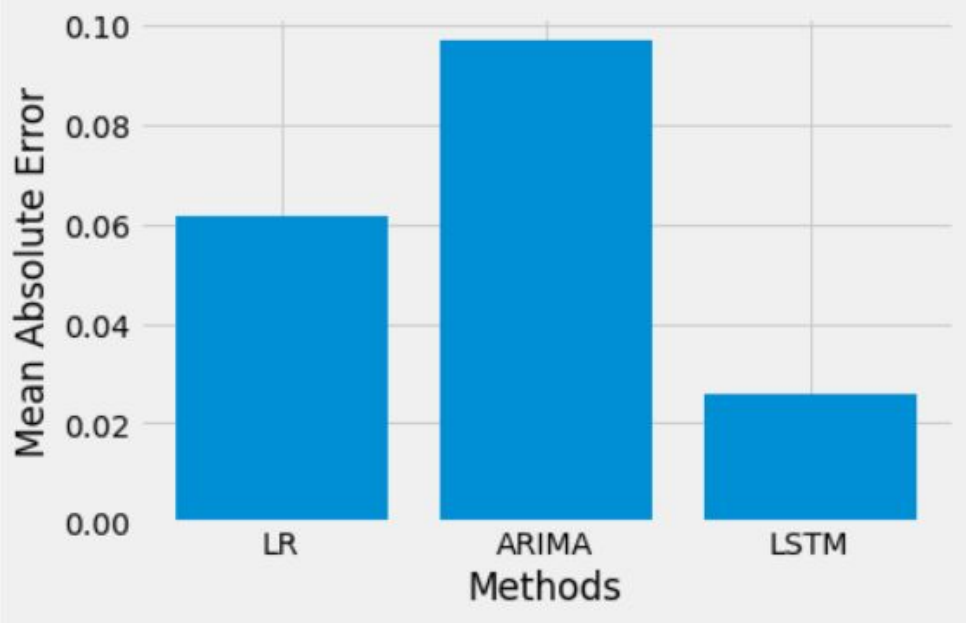
Various Metrics used to evaluate the models are as follows, there are three metrics that I have used to evaluate the regressors built using each approach, viz

1. Mean Absolute Error
2. Mean Squared Error
3. Root Mean Squared Error

Mean Absolute Error(values)

Approach	Value
Linear Regression	0.06172975189574167
ARIMA	0.0969249825873675
LSTM	0.02588401634485302

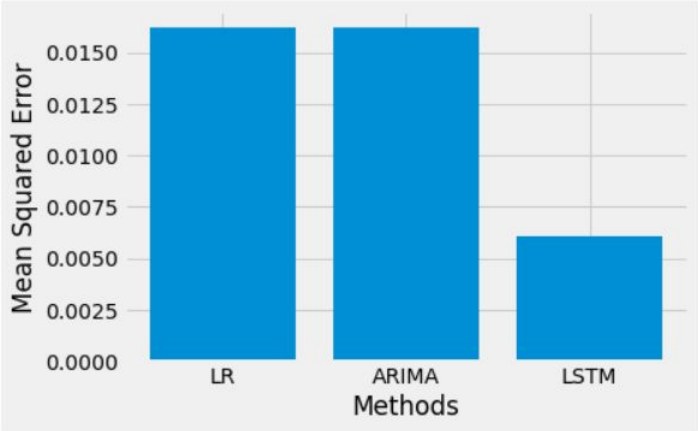
Visualization for MAE



Mean Squared Error (Data)

Approach	Value
Linear Regression	0.01619956318834035
ARIMA	0.01619956318834035
LSTM	0.006083750659449235

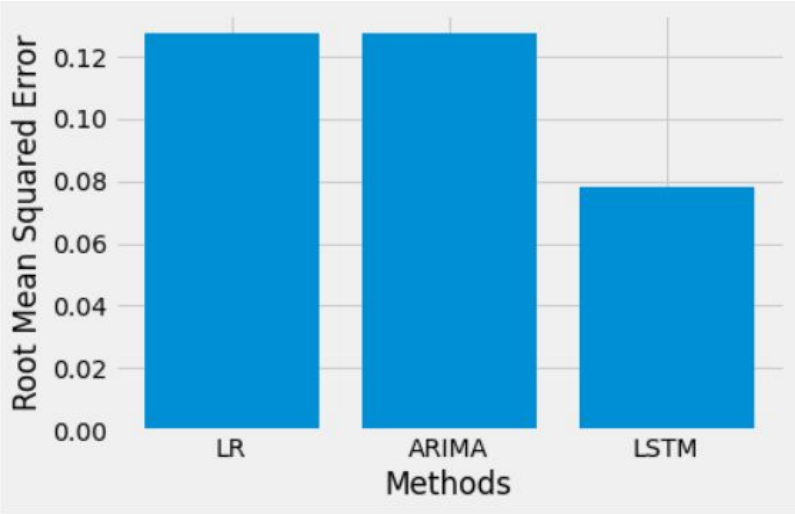
Visualization for MSE



Root Mean Squared Error (Data)

Approach	Value
Linear Regression	0.12727750464375215
ARIMA	0.12727750464375215
LSTM	0.07799840164675963

Visualization for RMSE



We can infer from the above data and visualizations that LSTM has indeed outperformed the other two approaches across the board

Conclusion:

The requirements of the problem statement were fulfilled with a deep learning approach with a mean squared error of the order 10^{-2}. LSTM is a powerful tool when working with time series data and offers a fluid construct that can be adapted and scaled across multiple types of data and problems as such.

Future Work:

This project can be further fine-tuned to fit a bigger dataset or any other similar problem statement as the method and the approach is not domain bounded. Other time frames of demand prediction than that of the one specified in the problem statement (hourly) can be easily achieved. In other words, the solution is scalable can be applied over a variety of problems.

References

1. https://keras.io/examples/lstm_stateful/
2. https://www.tensorflow.org/versions/r1.15/api_docs/python/tf
3. https://www.tensorflow.org/api_docs/python/tf/compat/v1/keras/layers/CuDNNLSTM
4. https://www.tensorflow.org/install/gpu
5. https://developer.nvidia.com/cudnn
6. https://ipython.org/notebook.html
7. https://www.kaggle.com/amirrezaeian/time-series-data-analysis-using-lstm-tutorial
8. https://www.kaggle.com/zhangxiangan/time-series-forecasting-with-lstm
9. https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.bar.html
10. https://otexts.com/fpp2/arima.html
11. https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.savefig.html
12. https://data.london.gov.uk/publisher/timeline/uk-power-networks
13. https://www.researchgate.net/publication/331291419_Electric_Energy_Consumption_Prediction_by_Deep_Learning_with_State_Explainable_Autoencoder
14. https://dl.acm.org/doi/pdf/10.1145/3357777.3357792
15. https://stackoverflow.com/questions/41908379/keras-plot-training-validation-and-test-set-accuracy

16. https://datascience.stackexchange.com/questions/30701/input-0-is-incompatible-with-layer-lstm-1-expected-ndim-3-found-ndim-2
17. https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.flatten.html
18. https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima_model.ARIMA.html
19. https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/
20. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html#
21. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html
22. https://github.com/lkilcher/dolfyn/issues/5
23. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
24. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
25. https://python-forum.io/Thread-TypeError-Series-object-cannot-be-interpreted-as-an-integer
26. https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.bar.html
27. https://www.programiz.com/python-programming/datetime/strftime
28. https://stackoverflow.com/questions/43206554/typeerror-float-argument-must-be-a-string-or-a-number-not-period
29. https://stackoverflow.com/questions/1574088/plotting-time-in-python-with-matplotlib/1574146
30. https://stackoverflow.com/questions/4090383/plotting-unix-timestamps-in-matplotlib
31. https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html
32. https://docs.python.org/3/library/datetime.html
33. https://stackoverflow.com/questions/49719627/boolean-mask-from-pandas-datetime-index-using-loc-accessor
34. https://stackoverflow.com/questions/26700598/matplotlib-showing-x-tick-labels-overlapping-despite-best-efforts
35. https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.plot.html
36. https://stackoverflow.com/questions/29310116/removing-time-from-datetime-variable-in-pandas
37. https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html
38. https://matplotlib.org/3.1.0/gallery/style_sheets/fivethirtyeight.html
39. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html
40. https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.legend.html

41. https://stackoverflow.com/questions/15943769/how-do-i-get-the-row-count-of-a-pandas-dataframe
42. https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.xlabel.html
43. https://matplotlib.org/api/_as_gen/matplotlib.artist.Artist.set_axes.html#matplotlib.artist.Artist.set_axes
44. https://matplotlib.org/api/pyplot_api.html
45. https://matplotlib.org/3.2.1/gallery/subplots_axes_and_figures/figure_title.html
46. https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.title.html
47. https://matplotlib.org/3.2.1/tutorials/introductory/sample_plots.html
48. https://scikit-learn.org/stable/modules/model_evaluation.html
49. https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/
50. https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/keras