Photovoltaic Systems Modelling and Predictions

Data Mining 2020

*Metodija Novkovski*

*Risto Trajanov*

# Introduction

The general goal of this project is to use data from photovoltaic and global forecasting systems, measurements from local and global stations to create a model which we can predict with the consumption and production of electrical energy from a given photovoltaic station. Furthermore, these predictions will help the users of these photovoltaic stations to determine what is the most efficient way of using the generated electrical energy.

To achieve this goal, we used data from a local photovoltaic system which measurements are stored in a PostgreSQL database, data from GFS (Global Forecast System) retrieved from their open API and data from SMA (sunnyportal.com) for features from global photovoltaic systems.

The data from the photovoltaic systems will be used to train our models for prediction of consumption (for the local system) and production (for the global system) as targets respectively. While the data from the GFS will be used to make a prediction for a specific day.

To manage our workflow and ETL processes for this project, we used Apache Airflow which is an open-source workflow management platform made by AirBnB.

# Setup

Airflow

Airflow is a platform to programmatically author, schedule and monitor workflows or data pipelines. By workflow we mean a sequence of tasks, started on a specific schedule or triggered by an event which is frequently used to handle big data processing pipelines.

[*Official Airflow Documentation*](https://airflow.apache.org/)

*Typical workflow:*

Typical scenario

The core concept of Airflow is the Directed Acyclic Graph (DAG) which consists of multiple tasks which can be executed independently. These DAG files are written in Python and they follow a pretty organized structure.

Airflow is built to integrate with all databases, systems, cloud environments. Therefore, managing and maintaining all of the dependencies in the Airflow environment would be difficult. That’s why Airflow is usually ran in containers provided by Docker, for multi-platform, shared, lightweight environment.

[Docker](https://www.docker.com/)

We used community based Docker [image](https://github.com/puckel/docker-airflow) for the environment, which consists of a web-server used to serve the Airflow UI, scheduler and worker for the tasks (with a Celery executor), postgres database and redis cache for the queue and ‘Flower’ which is a celery monitoring tool.

Connections

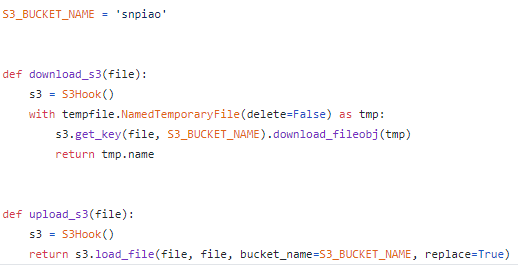
For persisting of our data we need several connections:

* PostgreSQL Connection (for loading the data for preprocessing and training our models)
* Amazon S3 Connection (for persisting the models we trained)

These connections are easily managed by Airflow with the UI.

For execution of these tasks regarding the persistence of our data to a database, Airflow uses [operators](https://airflow.apache.org/docs/stable/howto/operator/index.html) and [hooks](https://airflow.apache.org/docs/stable/howto/custom-operator.html#hooks).

*Example:*



# Extraction (Datasets)

## Cams Merra Data - Local Static Data

Static data that is extracted from real systems from location in Macedonia. The features in this data are the exact for the datetime, not a forecast like from gfs so the predictions are slightly better. The data features are picked to be the most representable for the target variable in this case energy consumption. The consumption is taken from a real photovoltaic system. The features were picked with ML methods and expert knowledge also.

## Sunny Portal - Gathering Data

Sunny Portal allows users to monitor and compare photovoltaic systems. We crawled a system in Eschwege, Germany and got the production of energy from that system for a period of time. Then we needed to get the same features that we got from the static data so we can train our new model with different features and different targets (production of energy).

We used BeautifulSoup to extract the output from the tables for production.

Problem occured: We could not get directly to the api page so we could extract our data. We went through a series of pages so we kind of simulated page interaction so we can get to our PV system. We also made a crawler with selenium but we considered that an overkill for this task.

def login(self, plantOid):

response = self.session.get(SMA.BASE\_URL + SMA.LOGIN\_URL)

self.session.get(SMA.BASE\_URL + SMA.EXAMPLE\_PLANTS)

response = self.session.get(SMA.BASE\_URL + '/RedirectToPlant/' + plantOid)

self.plantOid = plantOid

return response

## Global Forecast Systems (GFS) - Gathering Data

The GFS API allows users to get predicted weather data based on longitude and latitude so we extracted data for the location that we crawled from the sunny portal. We crawled the latitude and longitude from Eshwege, Germany and merged the data with the production based on datetime.

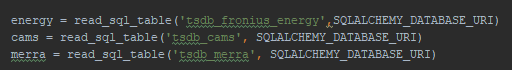
We used .sh scripts to download the data from GFS.

# Transformation and Preprocessing

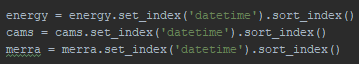
For our needs of creating models for prediction, we need a homogenous dataset which we can train our models on. For that purpose we need to transform and preprocess our raw data.

Regarding the data from our local photovoltaic system we need to do several steps:

* Read the data from the various sensors



* Set and sort the index for the time series format



* Merge the data



* Dealing with missing values

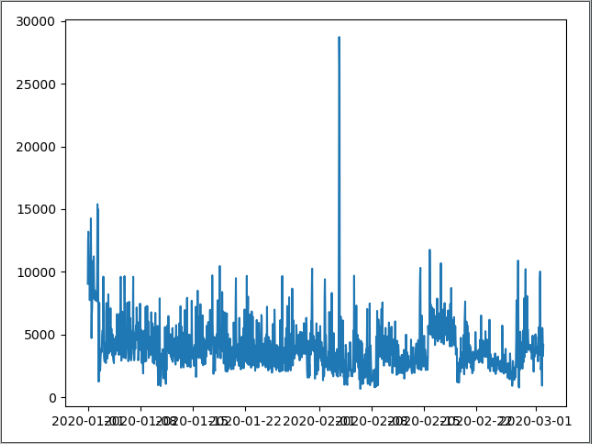


*Using linear interpolation method because our weather features are linearly correlated with time.*

* Tidying up our features and target



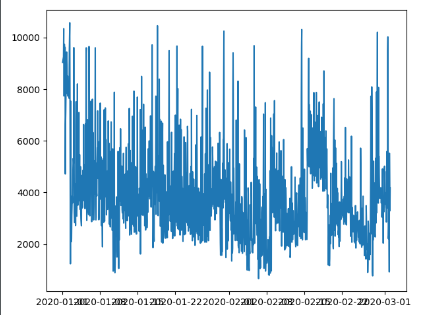
* Dealing with outliers



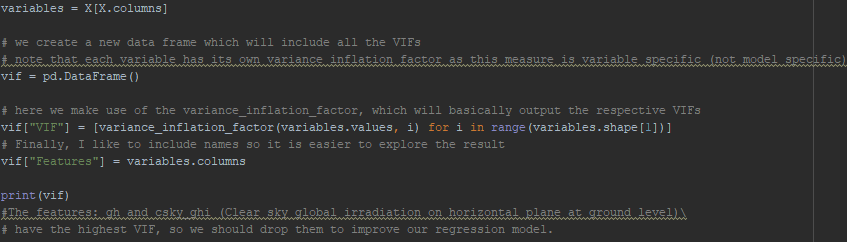
*Visible outlier in our target ‘consumption’*

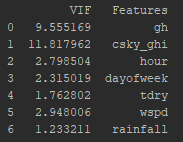


*Getting rid of it by removing above the 99 percentile*



* Testing the features for multicollinearity (Variance Inflation Factor)





Usually a high variance inflation factor indicates that the variable has negative or no impact at all for the standard regression models.

In our dataset such variables are ‘gh’ and ‘csky\_ghi’ which represent the global illumination at the station, so we should not consider them in building standard linear regression models.

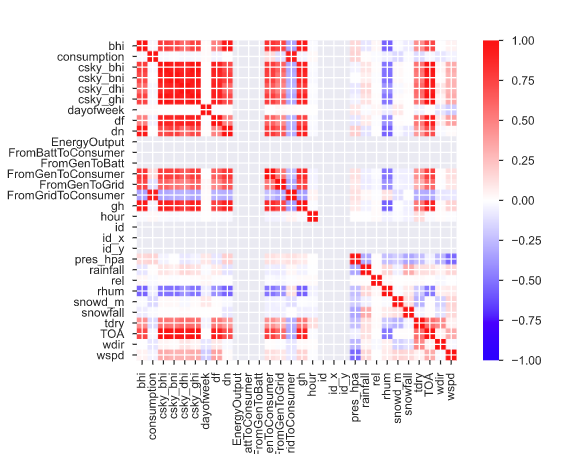
After all of these transformations and preprocessing, the final dataset is persisted back on to the Postgres database.

# Visualization

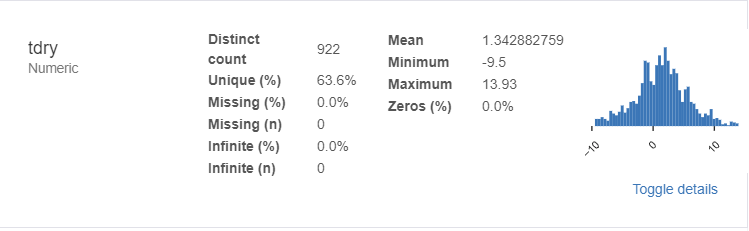
We used matplotlib and pandas-profiling for general visualizations and profiling of our data.

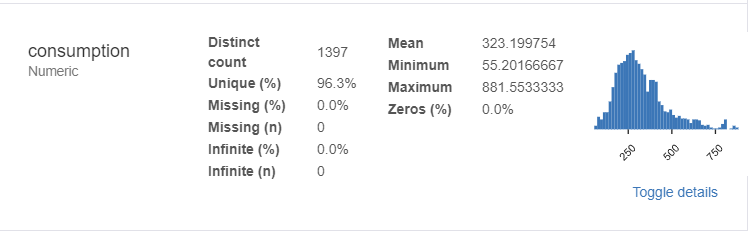
It is the easiest way to gain intuitive knowledge about the dataset you are working on. Also, it helps with catching outliers or any other irregularities. Correlation matrices furthermore enhance our knowledge for the features in our dataset in order to make a robust model with selecting the right features.

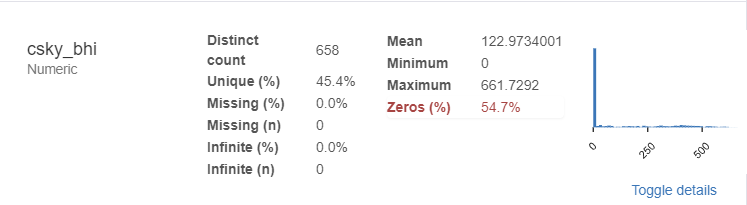
The data profiling for our most important feature tells us that we are dealing with variables with a normal distribution, and we can also see that some features are mostly with values 0 (ex. Global illumination at night) which we should be aware of.

******

We can see some obvious observations for the correlation between some features like ‘bhi’ and ‘csky\_bhi’ which both are attributes for the global illumination with clouds and under a clear sky respectively.



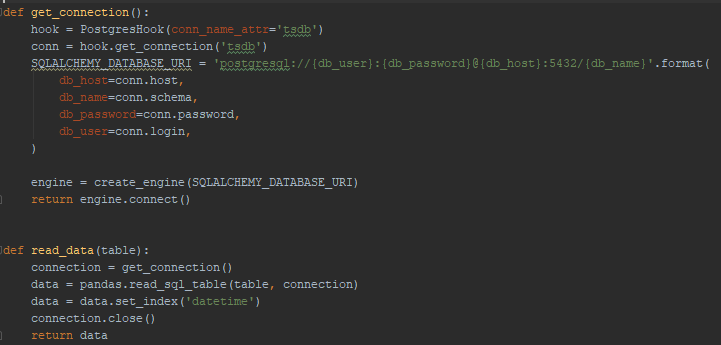




# Load - Database persistence

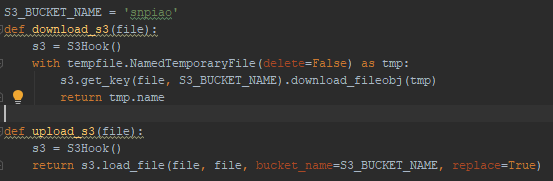
Loading the data on the airflow workers from the Postgres database can be done in several ways. One of them is to use Postgres Operator already-made by the community which integrates Postgres Hooks for connecting to the database and generic functions for manipulation with the queries. The other way is to make a custom function which deals with loading and persisting data to the database. We opted for the later one.

We used Postgres Hooks for establishing the connection with the database. It is easily done with editing the default Airflow connection for Postgres or creating a new connection with a new ID. After the connection is made we used SQLAlchemy to create a SQL Engine for managing transactions and queries to the database.



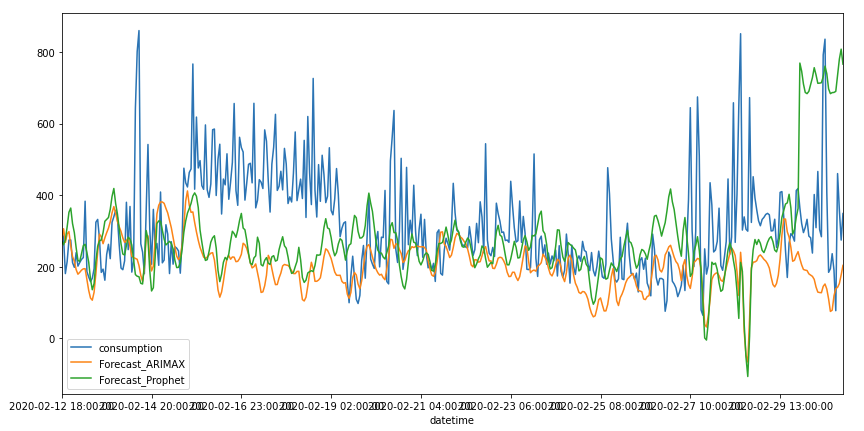
Model persistence

For the persistence of the models we used pickle and the keras tools for saving a model. After that the file is uploaded to Amazon S3 for further use for predictions.



# Models

Arima & Prophet



## LSTM

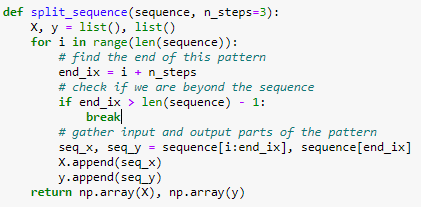
Time series prediction problems are a difficult type of predictive modeling problem. Unlike regression predictive modeling, time series also adds the complexity of a sequence dependence in the input variables.

A powerful type of neural networks from the black-box deep learning domain, which can handle sequence dependence are called recurrent neural networks (RNN). The Long Short-Term Memory neural network or LSTM is a type of a RNN and can be used to train on time-series data with pretty generalizing results.

For this project we covered two flavours of LSTM and those are univariate and multivariate LSTMs.

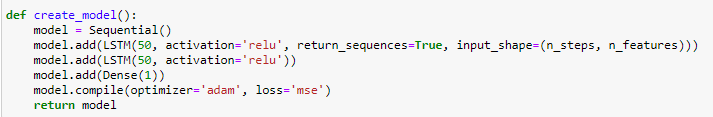
Univariate LSTM

Univariate LSTM is self-explanatory with its name. It uses only one variable (usually the target) to train a regressive model on the sequence dependence of that variable with respect to time.

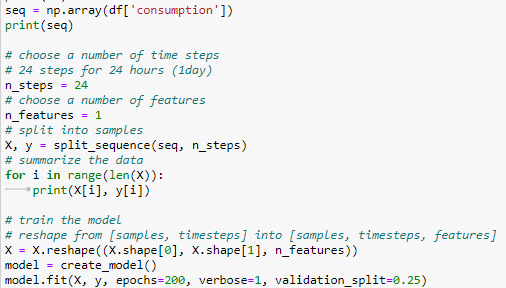


For this purpose we need to take our target variable, make it a sequence and split it such that for every n-steps we make the next one supervised value as a target for that sequence.

For the architecture of the model Keras with TensorFlow is used. The architecture is pretty simple with low number and parameters, but it is enough to provide consistent and generalized enough results. It uses relu for an activation function, adam for a fast optimizer and mean squared error for the loss function.

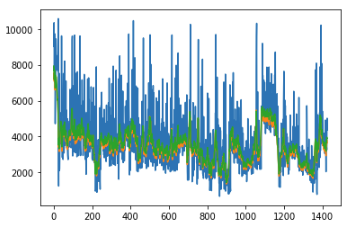


This model has two input parameters. One of them is number of features which is obviously one when we want to make a Univariate LSTM model, and the more interesting one is the number of time steps which the network takes to train from the sequence.



For choosing the number of time steps which the model will take to train the sequence dependence, we will use some expert knowledge from the users of the photovoltaic system. We know that the consumption of the electrical energy from the system is made by home appliances. We don’t know exactly which appliances, but we can assume that the lights are connected in the home. That would suggest that the lights are off during the day and on during the night so we should see some discrepancy there in our target variable. Also, there might be an AC which is working during the day and off during the night, opposite off the lights. Therefore, because all of these scenarios suggest a daily pattern in the usage of the energy, we set the parameter of the model to one day or 24 time-steps for 24 hours of the time series.

This model can make one day or several day ahead predictions.



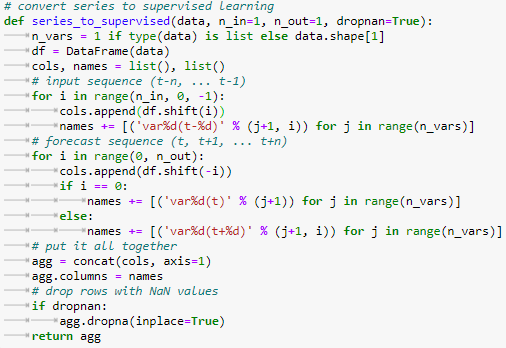
*The blue indicates the actual data, green and orange represent the forecasts for one and several days ahead respectively.*

We can see that this relatively simple model, even though it misses the high peaks, gives us very consistent and conservative results which can make pretty generalized predictions.

Multivariate LSTM

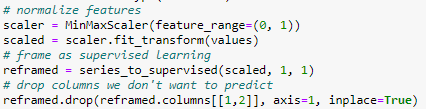
LSTM networks are pretty good at modeling problems with multiple input variables. This is a great benefit in time series forecasting, where classical linear methods can be difficult to learn to adapt to multiple input features.

As for the Univariate network, firstly we need to convert the time series to supervised learning sequence.

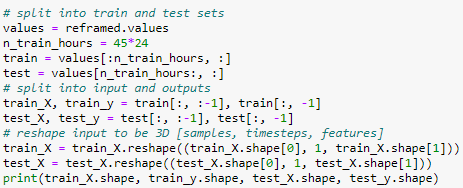


Even though neural networks aren’t bothered by redundant variables, the first two variables which had a high variance inflation factors are dropped from the dataset.

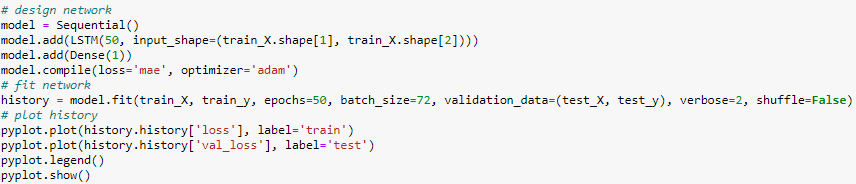
The data is normalized.

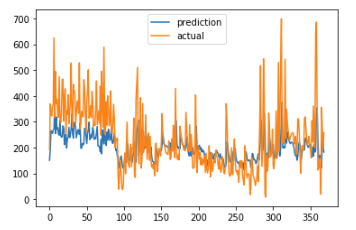
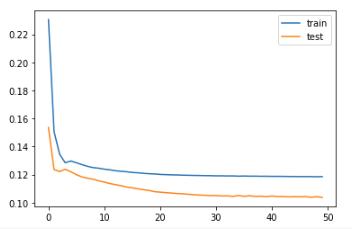


The dataset is split in to train and test sets.

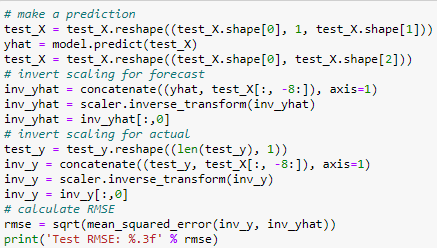


The architecture of the network is even more simple than the Univariate LSTM, with only two layers.





Even with the low amount of epochs the model is able to be again conservative and generalized.



For making a prediction we need to make invert scaling for the real values.

# Results - Conclusions

Regarding the complexity of the project with multiple raw data sources, we had a bigger challenge overcoming the ETL and workflow problems. However, Airflow manages to give a pretty abstracted framework for dealing with these processes. Otherwise, a lot of the scheduling and automation would have to involve writing CRON jobs and messing with shell scripts on a lower level.

On the other hand, even though the container should have been lightweight, by the sheer amount of libraries and dependencies we needed for the models and persisting tools, we managed to make the docker environment a little too big for the average personal computer.

However, this is easily solvable with docker swarm or kubernetes which can make a cluster of these docker containers which can run in parallel and independent of each other, optimizing efficiency and performance.

As for the models we created, they all give decently generalized results regarding our problem at hand, which can furthermore be used by the users of the photovoltaic systems for more efficient management of their yielded electrical power. However, we can notice that the auto-regressive models give more aggressive predictions of the trend, sometimes managing to catch the higher peaks, where as the neural network gave consistently more conservative results.

[GitHub Repo](https://github.com/risto-trajanov/DataMiningProject)

