

Code Documentation

AlGreenBox Template
(Example data - movement of birds)



Repository.....	3
Environmental setup.....	3
Template description.....	5
Dataset.....	5
Model purpose.....	5
Documentation purpose.....	5
Real case scenario.....	5
Overview.....	6
Model overview.....	6
Global settings.....	7
parameters.py.....	7
Services.....	8
Service 0:1: Data preparation.....	8
General description.....	8
Input.....	9
Output.....	9
Visual flow: Data preparation phase.....	10
Service 0:2: Verification phase.....	11
General description.....	11
Visual flow: Verification phase.....	14
Service 1: Training phase.....	15
General description.....	15
Visual flow: Training phase.....	16
Service 2: Prediction phase.....	17
General description.....	17
Visual flow: Prediction phase.....	17
Code walkthrough (Verification Phase).....	18
Script #1 - main.py.....	18
Script #2 - main_functions.py.....	19
Function: verification_phase.....	19
Function: hyper_parameter_phase.....	20
Parameter: dataset.....	20
Parameter: params.....	21
Script #3 - model.py.....	21
Function: data_split_80_20.....	22
Function: scaling_data.....	22
Function: series_to_supervised.....	23
Function: predict_verification_data.....	24
Function: train_predict_verification.....	24
Function: plot_result_verification.....	25

Repository

All code is found in a *repo* called “*AI_template_greenBox*”:

https://github.com/liecha/AI_template_greenBox

Environmental setup

To be able to run the code you need to set up the python environment. This is described in the file:

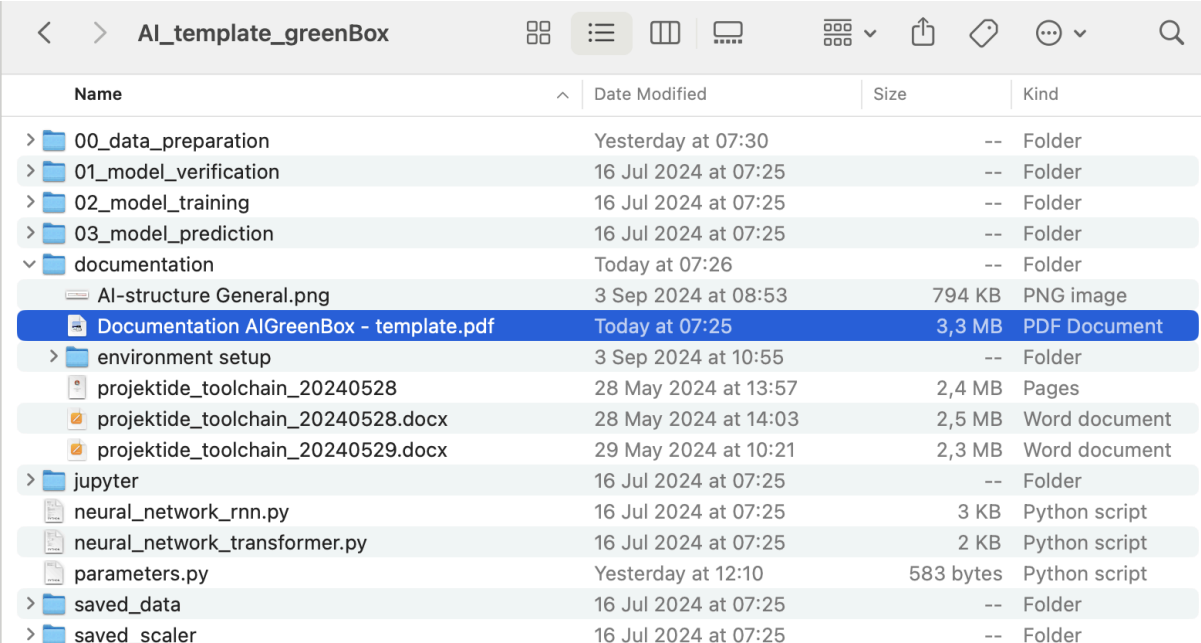
01 Environment Setup.pdf

which is found in the folder *environment setup*. An environment file called:

learningenvironment.yml

can be used to import the environment through anaconda. The process on how to do this is described in file:

01 Environment Setup.pdf



Name	Date Modified	Size	Kind
> 00_data_preparation	Yesterday at 07:30	--	Folder
> 01_model_verification	16 Jul 2024 at 07:25	--	Folder
> 02_model_training	16 Jul 2024 at 07:25	--	Folder
> 03_model_prediction	16 Jul 2024 at 07:25	--	Folder
> documentation	Today at 07:26	--	Folder
AI-structure General.png	3 Sep 2024 at 08:53	794 KB	PNG image
Documentation AIGreenBox - template.pdf	Today at 07:25	3,3 MB	PDF Document
> environment setup	3 Sep 2024 at 10:55	--	Folder
projektide_toolchain_20240528	28 May 2024 at 13:57	2,4 MB	Pages
projektide_toolchain_20240528.docx	28 May 2024 at 14:03	2,5 MB	Word document
projektide_toolchain_20240529.docx	29 May 2024 at 10:21	2,3 MB	Word document
> jupyter	16 Jul 2024 at 07:25	--	Folder
neural_network_rnn.py	16 Jul 2024 at 07:25	3 KB	Python script
neural_network_transformer.py	16 Jul 2024 at 07:25	2 KB	Python script
parameters.py	Yesterday at 12:10	583 bytes	Python script
> saved_data	16 Jul 2024 at 07:25	--	Folder
> saved_scaler	16 Jul 2024 at 07:25	--	Folder

I prefer to work with the application jupyter notebook and spyder which is installed with anaconda. If you prefer to set up your own python working environment the required libraries are listed below:

matplotlib
numpy
pandas

Last edited 2024-09-05 by Emelie Chandni Jutvik
emelie.chandni.jutvik@decerno.se

scikit-learn
keras
keras-preprocessing
tensorflow

If it will be of interest to make a connection to azure in the future - this library has to be added to the project:

azure-storage-blob

Template description

Dataset

The dataset used in this template was picked from kaggle.com. It contains the movement (longitude, latitude, speed, direction) of three birds (Nico, Eric and Sanne).

Model purpose

This model is mainly put up to demonstrate the flow of the code base that could be used for the AIGreenBox. The dataset chosen aims to demonstrate an easy example on how to use the code base. This code could be further developed and used with other datasets when the user/developer is familiar with the flow of this template/code base.

Documentation purpose

This document aims to describe the flow of this code base and also the purpose of the main functions of the code. The code base follows a structure that becomes handy while AI-projects are scaled up. This structure can be further developed or changed as Decerno prefers. The aim with this template is to give an example that could be further developed or just be used as inspiration. It could also be used as an educational basis while spreading this knowledge in the company.

Real case scenario

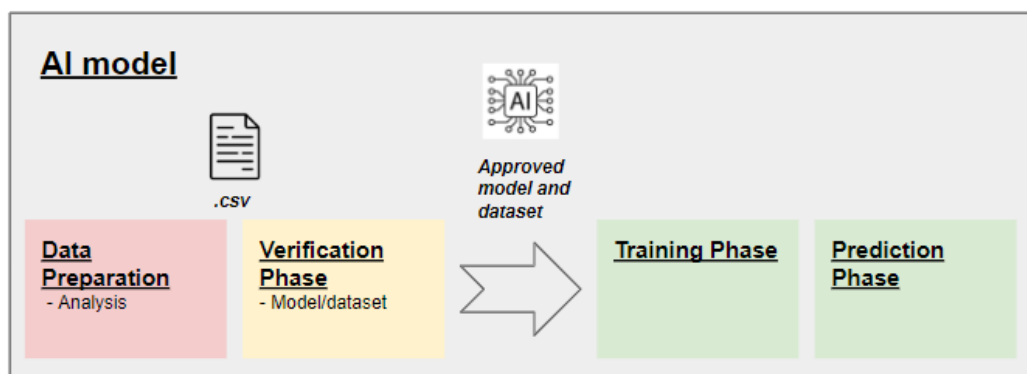
Note that this documentation focuses on describing the verification phase. This code base is built to structure the process from preparing the dataset to production mode (real case scenario - production mode). It is still rare to run these kinds of AI-models in irl-scenarios. This model has been implemented in real case scenarios and the code base is prepared to handle this. BUT it is hard to demonstrate a real case in an example like this since it is hard to find a case where we could pick a dataset and then be able to get irl-data for the prediction phase. The solution for the prediction phase will also depend on what kind of technology and data sources that have to be connected to the AI-model. Therefore the prediction phase is something that you will develop further when you enter a production phase with a customer. So this code is mainly left as an example of a possible solution but it will most likely not work on any problem.

Overview

Model overview

The model contains four phases:

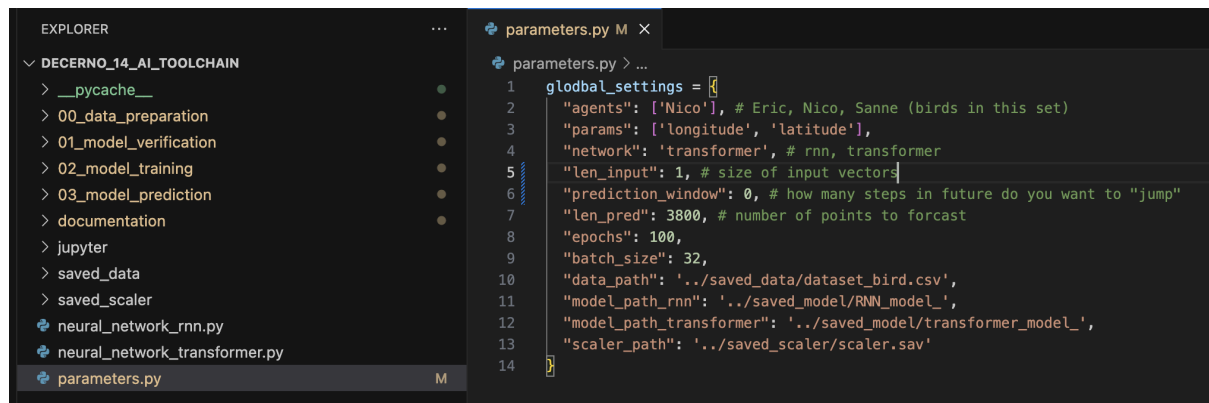
- *The data preparation phase* where we fetch data from a data source. The fetched data is prepared, cleaned and organized to be able to be read by an AI-model. The prepared dataset is saved in a csv-file.
- *The verification phase* where you use the prepared dataset and run through a verification mode. This phase makes sure that the data we got from the data source is valid. This phase is important for tuning the network.
- *The training phase* is where we train and save the network with the verified dataset and the tuned model from previous steps.
- *The prediction phase* is where we load the pre-trained network from the previous step. This network fetches the most recent data points to predict the forecasted path for the agent.



Global settings

parameters.py

All global parameters used in the code can be handled in the global setting dictionary. This is where you decide on which agent/agents and parameters the AI model should train on/predict. Important settings for the networks should be specified in this script. These settings are best tuned in the verifications phase (see *Chapter Services / Service 0:2 Verification phase*).



```
1 global_settings = {
2     "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)
3     "params": ['longitude', 'latitude'],
4     "network": 'transformer', # rnn, transformer
5     "len_input": 1, # size of input vectors
6     "prediction_window": 0, # how many steps in future do you want to "jump"
7     "len_pred": 3800, # number of points to forecast
8     "epochs": 100,
9     "batch_size": 32,
10    "data_path": '../saved_data/dataset_bird.csv',
11    "model_path_rnn": '../saved_model/RNN_model_',
12    "model_path_transformer": '../saved_model/transformer_model_',
13    "scaler_path": '../saved_scaler/scaler.sav'
14 }
```

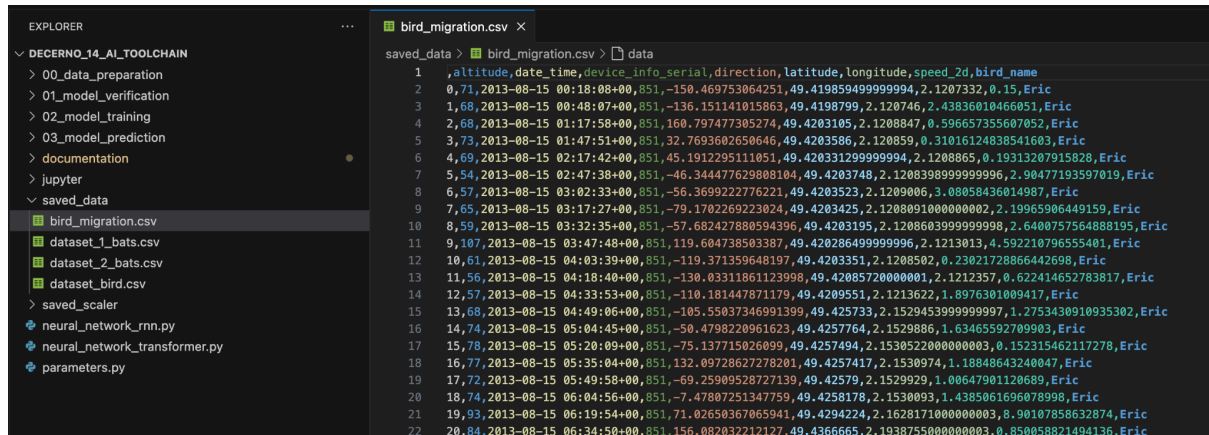
```
global_settings = {
    "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)
    "params": ['longitude', 'latitude'],
    "network": 'transformer', # rnn, transformer
    "len_input": 1, # size of input vectors
    "prediction_window": 0, # how many steps in future do you want to "jump"
    "len_pred": 3800, # number of points to forecast
    "epochs": 100,
    "batch_size": 32,
    "data_path": '../saved_data/dataset_bird.csv',
    "model_path_rnn": '../saved_model/RNN_model_',
    "model_path_transformer": '../saved_model/transformer_model_',
    "scaler_path": '../saved_scaler/scaler.sav'
}
```

Services

Service 0:1: Data preparation

General description

To be able to fulfill the requirements of the AI-model we simply need to prepare a dataset for the model that contains all data that the AI needs. The input file in this walkthrough example looks like this:



	altitude	date_time	device_info_serial	direction	latitude	longitude	speed_2d	bird_name
1	0,71	2013-08-15 00:18:08+00	851,-150.469753064251	49.419859499999994	2.1207332	0.15	Eric	
2	1,68	2013-08-15 00:48:07+00	851,-136.151141015863	49.4198799	2.120746	2.43836010466051	Eric	
3	2,68	2013-08-15 01:17:58+00	851,160.797477305274	49.4203105	2.1208847	0.596657355607052	Eric	
4	3,73	2013-08-15 01:47:51+00	851,32.7693602650646	49.4203586	2.120859	0.31016124838541603	Eric	
5	4,69	2013-08-15 02:17:42+00	851,45.1912295111051	49.420331299999994	2.1208865	0.19313207915828	Eric	
6	5,54	2013-08-15 02:47:38+00	851,-46.344477629808104	49.4203748	2.1208398999999996	2.90477193597019	Eric	
7	6,57	2013-08-15 03:02:33+00	851,-56.3699222776221	49.4203523	2.1209006	3.08058436014987	Eric	
8	7,65	2013-08-15 03:17:27+00	851,-79.1702269223024	49.4203425	2.1208091000000002	2.19965906449159	Eric	
9	8,59	2013-08-15 03:32:35+00	851,-57.682427880594396	49.4203195	2.1208603999999998	2.6400757564888195	Eric	
10	9,107	2013-08-15 03:47:48+00	851,119.604738503387	49.420286499999996	2.1213013	4.592210796555401	Eric	
11	10,61	2013-08-15 04:03:39+00	851,-119.371359648197	49.4203351	2.1208502	0.23021728866442698	Eric	
12	11,56	2013-08-15 04:18:40+00	851,-130.03311861123998	49.42085720000001	2.1212357	0.622414652783817	Eric	
13	12,57	2013-08-15 04:33:53+00	851,-110.181447871179	49.4209551	2.1213622	1.8976301009417	Eric	
14	13,68	2013-08-15 04:49:06+00	851,-105.55037346991399	49.425733	2.1529453999999997	1.2753430910935302	Eric	
15	14,74	2013-08-15 05:04:45+00	851,-50.4798220861623	49.4257764	2.1529086	1.63465592709903	Eric	
16	15,78	2013-08-15 05:20:09+00	851,-75.137715026099	49.4257494	2.1530522000000003	0.152315462117278	Eric	
17	16,77	2013-08-15 05:35:04+00	851,132.09728627278201	49.4257417	2.1530974	1.18840643240047	Eric	
18	17,72	2013-08-15 05:49:58+00	851,-69.25909528727139	49.42579	2.1529929	1.00647901120689	Eric	
19	18,74	2013-08-15 06:04:56+00	851,-7.47807251347759	49.4258178	2.1530093	1.4385061696078998	Eric	
20	19,93	2013-08-15 06:19:54+00	851,71.02650367065941	49.4294224	2.1628171000000003	8.90107858632874	Eric	
21	20,84	2013-08-15 06:34:50+00	851,156.082032212127	49.4366665	2.1938755000000003	0.850058821494136	Eric	
22								

The model that is described in this walkthrough of the model will focus on predicting the movement of a bird. The original dataset gives more information than what is needed for the basic model demonstrated in this documentation. Parameters that have to be given to the basic model are latitude and longitude. It is also important to create a filter for each agent (bird) in the dataset. This dataset contains three birds - Eric, Nico and Sanne. It is of great importance to filter the dataset in one batch for each bird.

NOTE:

It is possible to run all or selections of agents (birds) in sequence after each other by setting the agents parameter to `agents = ['Nico', 'Eric', 'Sanne']`:

```
global_settings = {
    "agents": ['Nico', 'Eric', 'Sanne'], # Eric, Nico, Sanne (birds in this set)
    "params": ['longitude', 'latitude'],
    "network": 'transformer', # rnn, transformer
    "len_input": 1, # size of input vectors
    "prediction_window": 0, # how many steps in future do you want to "jump"
    "len_pred": 3800, # number of points to forecast
    "epochs": 100,
    "batch_size": 32,
    "data_path": './saved_data/dataset_bird.csv',
    "model_path_rnn": './saved_model/RNN_model_',
    "model_path_transformer": './saved_model/transformer_model_',
    "scaler_path": './saved_scaler/scaler.sav'
}
```


Given the example above (global_settings where *agents* = ['Nico'] and *params* = ['longitude', 'latitude']) the csv-file could look like this:

agent,	date_time,	longitude	latitude
Nico,	2013-08-15 00:18:08,	2.12,	49.41
Nico,	2013-08-15 00:48:07,	2.13	49.41
etc.			

As a user of this framework you could choose to build this csv-file on any parameters given in the original data source/file. The important goal to achieve is to create a csv-file that correlates to the given global parameters settings - if longitude and latitude are listed as *params* in the global_settings these parameters have to be found in the original data source/file.

The dataset can also contain more parameters than those specified in the *params* array. The only parameters used by the AI model are those listed in the *params* array. In this given example speed and direction are parameters given in the original data source/file that could be used to further develop the model. This walkthrough will only demonstrate a basic case where longitude and latitude will be predicted.

Input

Data could be fetched from arbitrary sources (.json, mongodb, mqtt, etc.). The important step is to shape the data from the source to fit the requirements of the AI-model.

Output

Csv-file. With this given global settings for *agents* and *params*:

```
global_settings = {  
    "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)  
    "params": ['longitude', 'latitude'],  
    "network": 'transformer', # rnn, transformer  
    "len_input": 1, # size of input vectors  
    "prediction_window": 0, # how many steps in future do you want to "jump"  
    "len_pred": 3800, # number of points to forecast  
    "epochs": 100,  
    "batch_size": 32,  
    "data_path": './saved_data/dataset_bird.csv',  
    "model_path_rnn": './saved_model/RNN_model_',  
    "model_path_transformer": './saved_model/transformer_model_',  
    "scaler_path": './saved_scaler/scaler.sav'  
}
```

We want this csv-file to look like this:

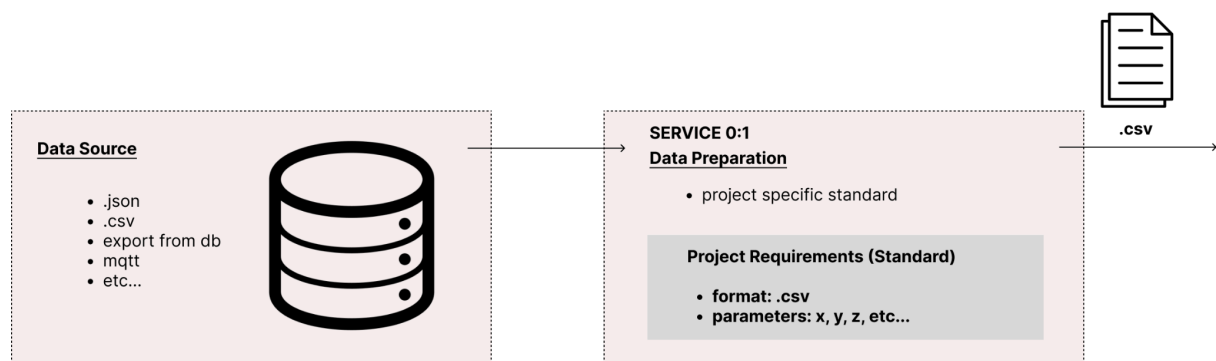
agent,	date_time,	longitude	latitude
Nico,	2013-08-15 00:18:08,	2.12,	49.41
Nico,	2013-08-15 00:48:07,	2.13	49.41
etc.			

NOTE:

The global setting parameter *data_path* points out where this csv-file should be placed. Make sure that the file name on your csv-file corresponds to the file name in this setting.

```
global_settings = {  
    "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)  
    "params": ['longitude', 'latitude'],  
    "network": 'transformer', # rnn, transformer  
    "len_input": 1, # size of input vectors  
    "prediction_window": 0, # how many steps in future do you want to "jump"  
    "len_pred": 3800, # number of points to forecast  
    "epochs": 100,  
    "batch_size": 32,  
    "data_path": './saved_data/dataset_bird.csv',  
    "model_path_rnn": './saved_model/RNN_model_',  
    "model_path_transformer": './saved_model/transformer_model_',  
    "scaler_path": './saved_scaler/scaler.sav'  
}
```

Visual flow: Data preparation phase



Service 0:2: Verification phase

General description

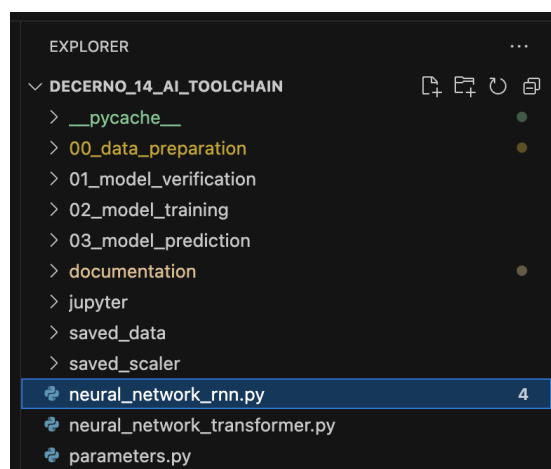
The verification step in this process is a quality assurance test. This step is required to make sure that the given csv-file in the preparation phase contains all data needed to train the network. If the csv-file fulfills requirements set by the AI-model the model will run a verification test run to test its performance.

While the csv-file is verified and approved the AI-model will divide this dataset into a training set and a test set. This will default be done by training the model on 80% of the data and using the rest 20% for a verification and test case. The result of this process will be a plot. This plot helps you to verify the performance of the model and laborate with global settings to improve the model. Parameters that affect the model performance are *network*, *len_input*, *prediction_window*, *len_pred*, *epochs* and *batch_size*.

```
glodbal_settings = {  
    "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)  
    "params": ['longitude', 'latitude'],  
    "network": 'transformer', # rnn, transformer  
    "len_input": 1, # size of input vectors  
    "prediction_window": 0, # how many steps in future do you want to "jump"  
    "len_pred": 3800, # number of points to forecast  
    "epochs": 100,  
    "batch_size": 32,  
    "data_path": './saved_data/dataset_bird.csv',  
    "model_path_rnn": './saved_model/RNN_model_',  
    "model_path_transformer": './saved_model/transformer_model_',  
    "scaler_path": './saved_scaler/scaler.sav'  
}
```

NOTE:

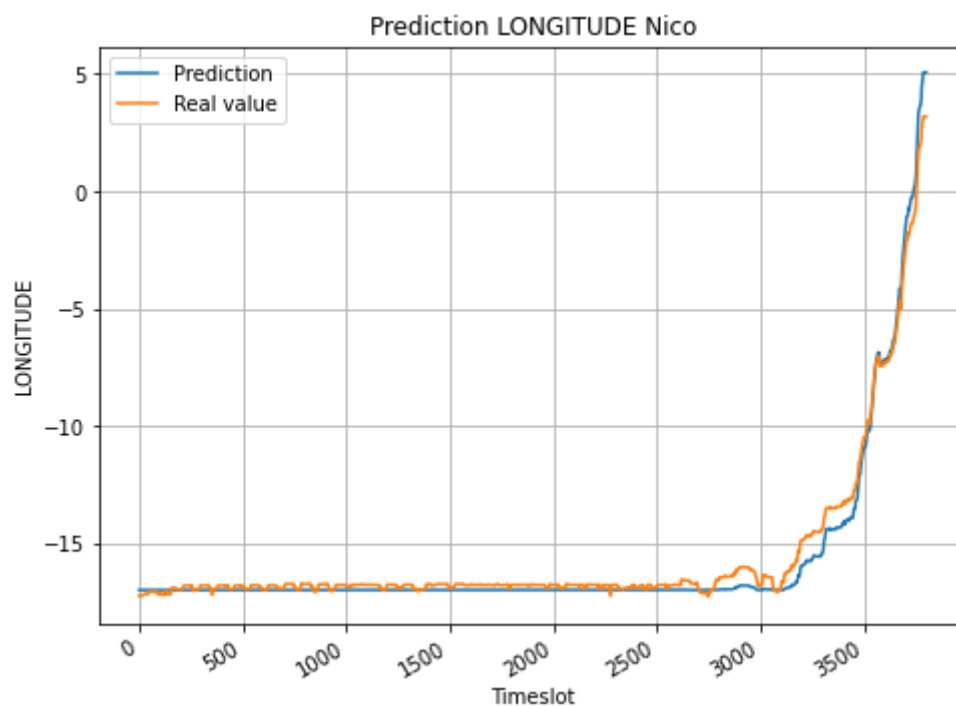
There is currently one kind of two available for this model which are a transformer (*neural_network_transformer*) and RNN (*neural_network_rnn*).

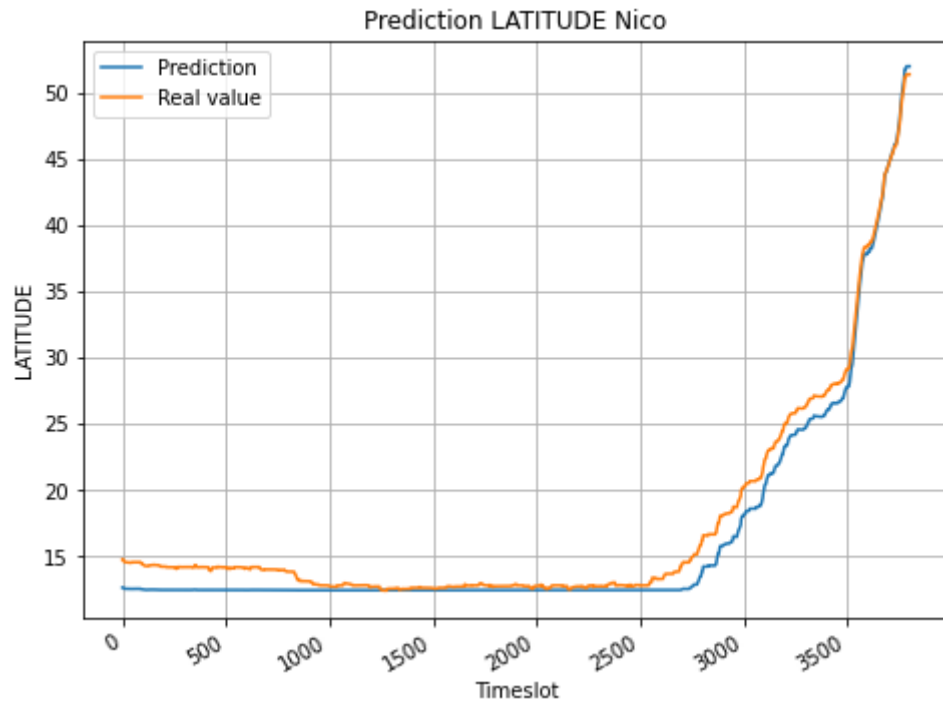


The user specifies which network to use in the global settings. It is possible to implement any neural network by adding a script with this network in the general folder of the project.

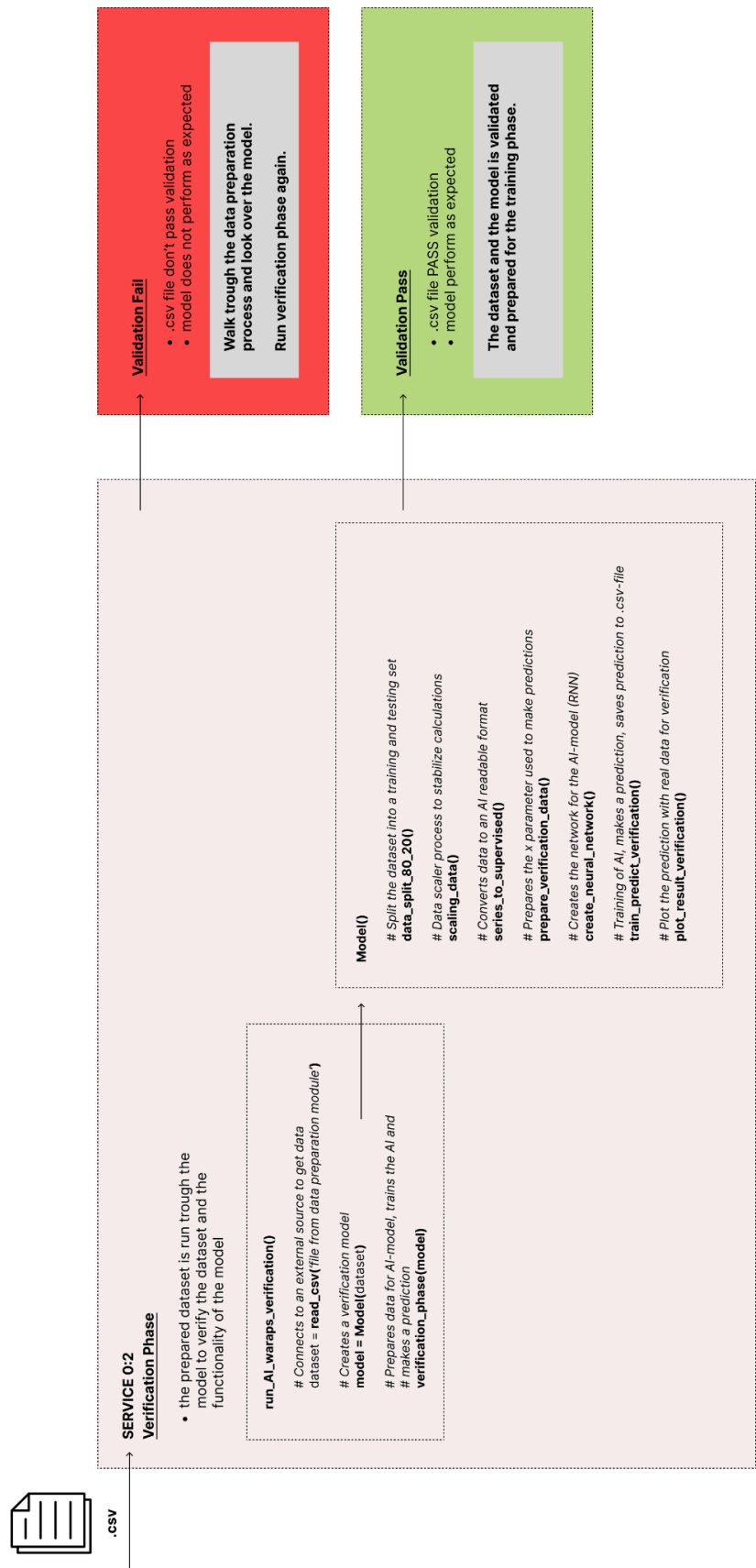
```
glodbal_settings = {  
    "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)  
    "params": ['longitude', 'latitude'],  
    "network": 'transformer', # rnn, transformer  
    "len_input": 1, # size of input vectors  
    "prediction_window": 0, # how many steps in future do you want to "jump"  
    "len_pred": 3800, # number of points to forecast  
    "epochs": 100,  
    "batch_size": 32,  
    "data_path": './saved_data/dataset_bird.csv',  
    "model_path_rnn": './saved_model/RNN_model_',  
    "model_path_transformer": './saved_model/transformer_model_',  
    "scaler_path": './saved_scaler/scaler.sav'  
}
```

The global setting is set to prediction of longitude and latitude for the bird Nico. Further len_pred is 3800 which means that 3800 points of movement should be predicted. The network trains (100 epochs in this case) with all given input data except for these 3800 points which are used for validation. This is an example of the resulting plot from the verification phase:





Visual flow: Verification phase



Service 1: Training phase

General description

The previous two steps (data preparation and model verification) are just preparation work for the next two steps. The first step is to train the model and the next step would be to use the trained model to make a prediction.

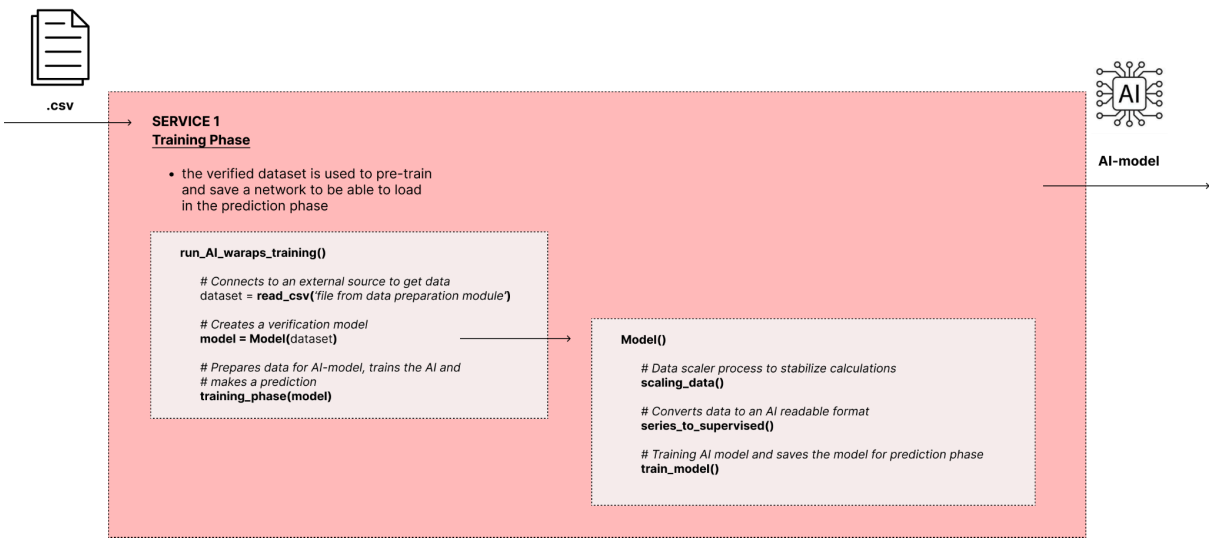
It is highly important that you are tuning your network in the verification phase to perform well. These settings are then used in the training phase. You make sure that your network performs well by evaluating the results given in the verification phase.

The csv-file created in the data preparation phase is also validated in the verification phase. This csv-file has to fulfill the AI-model requirements. If the csv-file is not fulfilling the requirements the model will not be able to run.

If both the csv-file and the model pass through the validation step then this phase is good to go. Just run the training model and the trained networks will be saved in the given global settings for *model_path_rnn* or *model_path_transformer*.

```
glodbal_settings = {  
    "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)  
    "params": ['longitude', 'latitude'],  
    "network": 'transformer', # rnn, transformer  
    "len_input": 1, # size of input vectors  
    "prediction_window": 0, # how many steps in future do you want to "jump"  
    "len_pred": 3800, # number of points to forecast  
    "epochs": 100,  
    "batch_size": 32,  
    "data_path": './saved_data/dataset_bird.csv',  
    "model_path_rnn": './saved_model/RNN_model_',  
    "model_path_transformer": './saved_model/transformer_model_',  
    "scaler_path": './saved_scaler/scaler.sav'  
}
```

Visual flow: Training phase



Service 2: Prediction phase

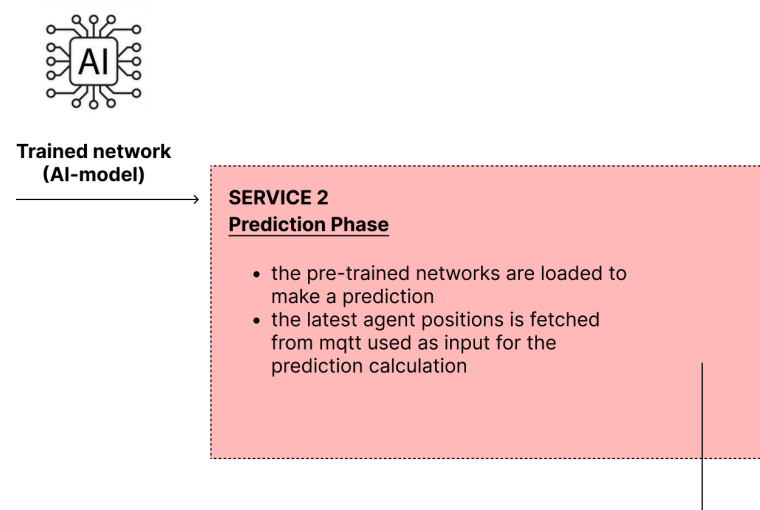
General description

In the prediction phase we are able to load the pre-trained networks from the previous step. This specific model will get the latest position data (longitude, latitude) of an agent (bird) as input for the calculation. The pre-trained network predicts future positions. The amount of future positions to predict is decided by the *len_pred* parameter in global settings.

```
global_settings = {  
    "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)  
    "params": ['longitude', 'latitude'],  
    "network": 'transformer', # rnn, transformer  
    "len_input": 1, # size of input vectors  
    "prediction_window": 0, # how many steps in future do you want to "jump"  
    "len_pred": 3800, # number of points to forecast  
    "epochs": 100,  
    "batch_size": 32,  
    "data_path": './saved_data/dataset_bird.csv',  
    "model_path_rnn": './saved_model/RNN_model_',  
    "model_path_transformer": './saved_model/transformer_model_',  
    "scaler_path": './saved_scaler/scaler.sav'  
}
```

This phase could not be demonstrated with this model since we don't have a real time measure of the movement of the birds. The model is prepared for this functionality but Decerno's developers need to connect the dots here in the future! :)

Visual flow: Prediction phase



Plot that shows real irl-movements with the predicted movements.

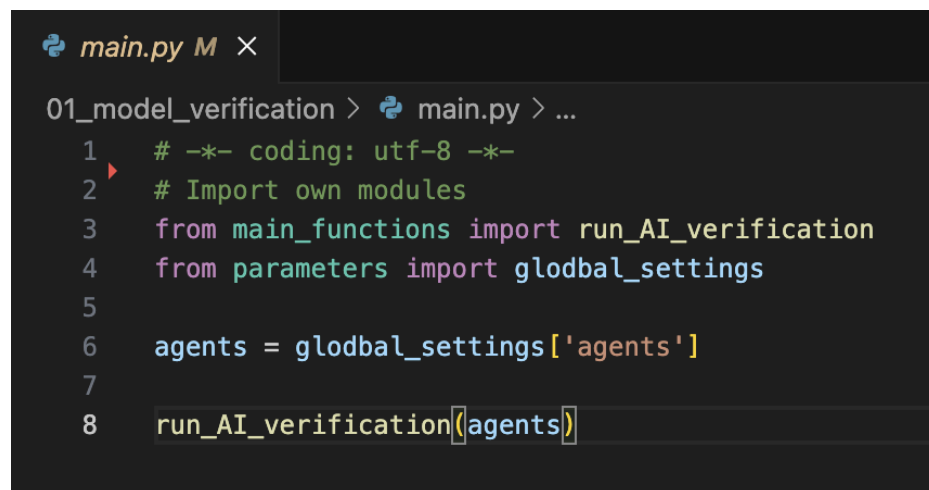
Code walkthrough (Verification Phase)

The code base contain three main files:

- *main.py*
- *main_functions.py*
- *model.py*

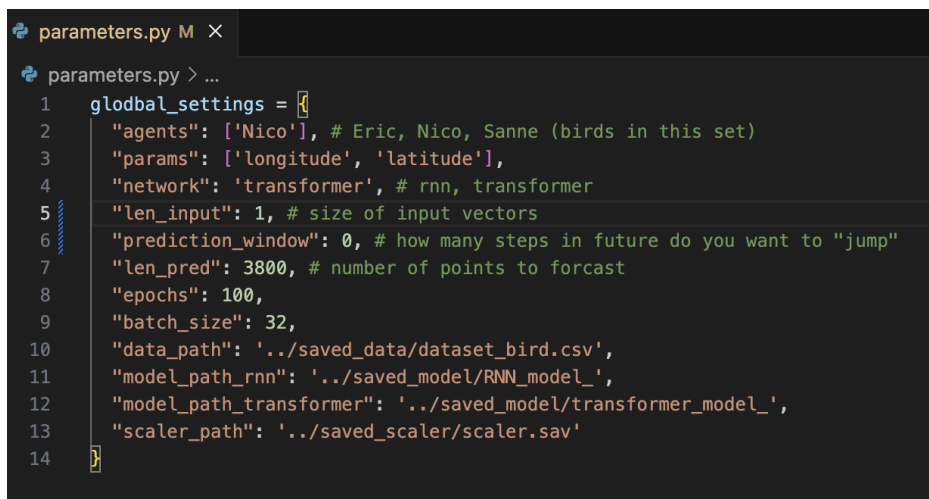
Script #1 - main.py

The file *main.py* contains one function that triggers the whole calculation.



```
main.py M X
01_model_verification > main.py > ...
1  # -*- coding: utf-8 -*-
2  # Import own modules
3  from main_functions import run_AI_verification
4  from parameters import glodbal_settings
5
6  agents = glodbal_settings['agents']
7
8  run_AI_verification(agents)
```

The only thing to be aware of is which agent/agents (in this case bird/birds) you want to evaluate. This is specified in the global setting (*paramters.py*).



```
parameters.py M X
parameters.py > ...
1  glodbal_settings = {
2      "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)
3      "params": ['longitude', 'latitude'],
4      "network": 'transformer', # rnn, transformer
5      "len_input": 1, # size of input vectors
6      "prediction_window": 0, # how many steps in future do you want to "jump"
7      "len_pred": 3800, # number of points to forecast
8      "epochs": 100,
9      "batch_size": 32,
10     "data_path": '../saved_data/dataset_bird.csv',
11     "model_path_rnn": '../saved_model/RNN_model_',
12     "model_path_transformer": '../saved_model/transformer_model_',
13     "scaler_path": '../saved_scaler/scaler.sav'
14 }
```

While running the *main.py* script the two scripts *main_functions.py* and *model.py* will be triggered.

Script #2 - main_functions.py

When we put this AI-solution into production we as developers pre-code parameters in the global settings. During the verification phase we want to be able to laborate with these parameters to improve the model performance.

```
main_functions.py 1, M ×
01_model_verification > main_functions.py > ...
1  # -*- coding: utf-8 -*-
2  # Import own modules
3  import pandas as pd
4
5  # Import own modules
6  from model import Model
7  from parameters import glodbal_settings
8
9
10 def run_AI_verification(agents):
11     params = glodbal_settings['params']
12     dataset = pd.read_csv(glodbal_settings['data_path'])
13     for agent in agents:
14         for i in range(0, len(params)):
15             model = Model(dataset, agent, params[i])
16             print('Preparing dataset...')
17             print('----- START PROCESSING PARAMETER ' + params[i] + ' -----')
18             print('Entering verification phase...')
19             verification_phase(model)
20     print('VERIFICATION PROCESS COMPLETED...')
21
```

Function: verification_phase

There are two different paths to choose in the *main_function* script. In the figure above the *verification_phase* was chosen. This gives the developer more responsibility to tune the parameters for the neural network (*parameters.py*). A good start is to set *epochs* to 100 and *batch_size* to 32 or 64. The history plots given after every calculation will give an indication if these parameters needs tuning.

```
def verification_phase(model):
    print('Split dataset 80/20 training/test...')
    model.data_split_80_20()
    print('Scaling process started...')
    model.scaling_data()
    print('Reshaped data for supervised learning...')
    model.series_to_supervised()
    print('Prepare data for test and verification...')
    model.prepare_verification_data()
    print('Training and verification process in action...')
    model.train_predict_verification(glodbal_settings['network'])
    print('Printing the results...')
    model.plot_result_verification()
```

```
glodbal_settings = {
    "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)
    "params": ['longitude', 'latitude'],
    "network": 'transformer', # rnn, transformer
    "len_input": 1, # size of input vectors
    "prediction_window": 0, # how many steps in future do you want to "jump"
    "len_pred": 3800, # number of points to forecast
    "epochs": 100,
    "batch_size": 32,
    "data_path": './saved_data/dataset_bird.csv',
    "model_path_rnn": './saved_model/RNN_model_',
    "model_path_transformer": './saved_model/transformer_model_',
    "scaler_path": './saved_scaler/scaler.sav'
}
```

Function: `hyper_parameter_phase`

The second path available to choose is the `hyper_parameter_phase` function. This function will run the model over and over again to try to find the best possible hyper parameters. This is a really heavy process for a laptop and might therefore be a bad option at this point. This process can still be a little bit unstable so I suggest having this code and continue to develop it. It is a good tool but not optimal to use yet. It is often more efficient to do the tuning manually by inputting standard values to begin with.

```
def hyper_parameter_phase(model):
    print('Split dataset 80/20 training/test...')
    model.data_split_80_20()
    print('Scaling process started...')
    model.scaling_data()
    print('Reshaped data for supervised learning...')
    model.series_to_supervised()
    print('Prepare data for test and verification...')
    model.prepare_verification_data()

    if glodbal_settings['network'] == 'rnn':
        print('Building hyper parameter model Recurrent Neural Network...')
        print('Tuning hyper parameter model...')
        model.tuning_model_rnn()
        print('Printing the results...')
        model.plot_result_verification()

    if glodbal_settings['network'] == 'transformer':
        print('Building hyper parameter model Transformer...')
        print('Tuning hyper parameter model...')
        model.tuning_model_transformer()
        print('Printing the results...')
        model.plot_result_verification()
```

Parameter: dataset

During the development phase it is always nice to create the local dataset from a file. It takes some time to get data from the database and could be a pretty annoying wait if you are running

the code while you are in developing mode. This dataset could easily be prepared in the data preparation phase.

Parameter: params

The agents in this example have longitude and latitude parameters. If you want to experiment with these parameters they are found in the global settings *parameters.py*. It is possible to add parameters to work with by adding them to the array *params*.

```
glodbal_settings = {
    "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)
    "params": ['longitude', 'latitude'],
    "network": 'transformer', # rnn, transformer
    "len_input": 1, # size of input vectors
    "prediction_window": 0, # how many steps in future do you want to "jump"
    "len_pred": 3800, # number of points to forecast
    "epochs": 100,
    "batch_size": 32,
    "data_path": './saved_data/dataset_bird.csv',
    "model_path_rnn": './saved_model/RNN_model_',
    "model_path_transformer": './saved_model/transformer_model_',
    "scaler_path": './saved_scaler/scaler.sav'
}
```

Script #3 - model.py

In the previous file *main_functions.py* you were presented to a list if functions needed for the AI-model to make go into verification mode:

```
def verification_phase(model):
    print('Split dataset 80/20 training/test...')
    model.data_split_80_20()
    print('Scaling process started...')
    model.scaling_data()
    print('Reshaped data for supervised learning...')
    model.series_to_supervised()
    print('Prepare data for test and verification...')
    model.prepare_verification_data()
    print('Training and verification process in action...')
    model.train_predict_verification(glodbal_settings['network'])
    print('Printing the results...')
    model.plot_result_verification()
```

All these functions are coded in the *model.py* file. We will now look into important functions, settings and parameters that you as a developer need to be aware of while working with this file.

Function: data_split_80_20

This function splits the dataset into one training set with 80% of the given input data and one test set (verification data) with 20% of the given input data. The test set data will later be used to compare predicted data to real data (test set data).

```
def data_split_80_20(self):
    dataset = self.dataset
    limit = int(len(dataset)*0.8)
    df_training_set = dataset.iloc[0:limit]
    df_test_set = dataset.iloc[limit:-1]
    self.df_split_train_test.append(df_training_set)
    self.df_split_train_test.append(df_test_set)

    print('Dataset split into training and testdata...')
    print(df_training_set)
    print(df_test_set)
```

Function: scaling_data

This function is pretty straight forward. To get more knowledge about the scaling phase in AI-development a recommend to read this:

<https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35>

```
def scaling_data(self):
    # Feature Scaling
    scaler = MinMaxScaler(feature_range=(0, 1))
    self.scaler = []
    self.scaler.append(scaler)

    # Save the scaler to a file
    pickle.dump(scaler, open(globbal_settings['scaler_path'], 'wb'))

    print('Scaling data for verification phase...')
    data_train = self.df_split_train_test[0][self.param].values.reshape(-1, 1)
    data_test = self.df_split_train_test[1][self.param].values.reshape(-1, 1)

    scaled_data_train = scaler.fit_transform(data_train)
    self.scaled_data.append(scaled_data_train)

    scaled_data_test = scaler.fit_transform(data_test)
    self.scaled_data.append(scaled_data_test)
    print('Data for parameter ' + self.param + ' is scaled...')
```

Note: the parameter is needed for this function since the scaling will be done in steps - in this specific case we first scale the longitude data and in the second step the latitude data. The only reason for this order of events are that this is the order of these parameters given in the param array:

```
global_settings = {
    "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)
    "params": ['longitude', 'latitude'],
    "network": 'transformer', # rnn, transformer
    "len_input": 1, # size of input vectors
    "prediction_window": 0, # how many steps in future do you want to "jump"
    "len_pred": 3800, # number of points to forecast
    "epochs": 100,
    "batch_size": 32,
    "data_path": './saved_data/dataset_bird.csv',
    "model_path_rnn": './saved_model/RNN_model_',
    "model_path_transformer": './saved_model/transformer_model_',
    "scaler_path": './saved_scaler/scaler.sav'
}
```

Function: series_to_supervised

An AI-model has really specific requirements on the shape of the data that it should read. This function makes sure that the dataset given to the model is reshaped to be able to be read by the AI-model.

```
def series_to_supervised(self):
    dataset_scaled = self.scaled_data[0]
    X_train = []
    y_train = []

    len_trainset = len(dataset_scaled)
    for i in range(self.len_input, len_trainset - self.prediction_win + 1):
        X_train.append(dataset_scaled[i - self.len_input:i,0])
        y_train.append(dataset_scaled[i + self.prediction_win - 1:i + self.prediction_win,0])

    X_train, y_train = np.array(X_train), np.array(y_train)
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

    self.X_train = X_train
    self.y_train = y_train
    print(' - shape supervised serie == ', self.X_train.shape, self.y_train.shape)
    print(' - total dataset length == ', len(dataset_scaled))
    print(' - intervall range == ' + str(self.len_input) + ' --> ' + str(len_trainset - self.prediction_win + 1))
    print(' - no of X batches == ', (len_trainset - self.prediction_win + 1) - self.len_input)
    print(' - data in each batch == ', self.len_input)
```

It is important to know that the *len_pred* parameter given in the global settings decides the length of the prediction (in the prediction phase, in the verification phase the dataset is split into 80/20 splits described above). If you aim for a longer/shorter prediction interval you need to change this value. In this specific example the AI-model predicts 3800 timesteps into the future.

```
global_settings = {
    "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)
    "params": ['longitude', 'latitude'],
    "network": 'transformer', # rnn, transformer
    "len_input": 1, # size of input vectors
    "prediction_window": 0, # how many steps in future do you want to "jump"
    "len_pred": 3800, # number of points to forecast
    "epochs": 100,
    "batch_size": 32,
    "data_path": './saved_data/dataset_bird.csv',
    "model_path_rnn": './saved_model/RNN_model_',
```

```
        "model_path_transformer": './saved_model/transformer_model_',  
        "scaler_path": './saved_scaler/scaler.sav'  
    }
```

Another important parameter in this function is the *len_input*. This parameter decides the size of the lstm - how many steps before the current steps that should be used while making the prediction. Feel free to experiment with this parameter to see how it affects the result (prediction) made by the AI-model.

prediction_window is a parameter that decides how many steps the forecast should be displaced into the future.

Function: *predict_verification_data*

To be able to make a prediction the AI-model wants an input of movements of an agent in the most recent moment. This data has to be reshaped to be able to be understood by the AI-model. This function makes sure that the present data (*X_test* parameter) is reshaped and valid as an input to the AI-model. This follows the same procedure as the input (training) dataset (function - *supervised_to_series*).

```
def prepare_verification_data(self):  
    dataset_scaled_test = self.scaled_data[1]  
  
    X_test = []  
    y_test = []  
  
    for i in range(self.len_input, self.len_input + self.len_pred + 1):  
        X_test.append(dataset_scaled_test[i-self.len_input:i, 0])  
        y_test.append(dataset_scaled_test[i + self.prediction_win - 1:i + self.prediction_win,0])  
  
    X_test, y_test = np.array(X_test), np.array(y_test)  
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))  
    self.X_test = X_test  
    self.y_test = y_test  
    print(' - shape verification data == ', self.X_test.shape, self.y_test.shape)  
    print(' - intervall range      == ' + str(self.len_input) + ' --> ' + str(self.len_input + self.len_pred + 1))  
    print(' - predicted no of value == ', (self.len_input + self.len_pred + 1) - self.len_input)  
    print(' - data in each batch    == ', self.len_input)
```

Function: *train_predict_verification*

In this function we pick the network that was chosen in the *global_settings*. In the given example below the transformer network was chosen.

```
global_settings = {  
    "agents": ['Nico'], # Eric, Nico, Sanne (birds in this set)  
    "params": ['longitude', 'latitude'],  
    "network": 'transformer', # rnn, transformer  
    "len_input": 1, # size of input vectors  
    "prediction_window": 0, # how many steps in future do you want to "jump"  
    "len_pred": 3800, # number of points to forecast  
    "epochs": 100,  
    "batch_size": 32,  
    "data_path": './saved_data/dataset_bird.csv',  
    "model_path_rnn": './saved_model/RNN_model_',  
    "model_path_transformer": './saved_model/transformer_model_',  
}
```



```
        "scaler_path": './saved_scaler/scaler.sav'  
    }
```

Parameters that are important to know for the training process are *epochs* and *batch_size*. Both parameters are set in the global_settings (parameters.py). Feel free to experiment with these parameters to look for good results. It could differ a lot - a neural network often reaches a good training result after 50-100 epochs. You can experiment with different values of batch size and see which value gives a better result (try for example with 16, 32, 64, 128 etc.)

```
def train_predict_verification(self, network):  
    if network == 'rnn':  
        print('Predparing neural network (RECURRENT Neural Network...')  
        neural_net = RNN(self.X_train)  
    if network == 'transformer':  
        print('Predparing neural network (TRANSFORMER Neural Network...')  
        neural_net = Transformer(self.X_train)  
    history = neural_net.fit(self.X_train, self.y_train, epochs=self.epochs, batch_size=self.batch_size,  
                             #self.plot_training_history(history, 0)  
    scaler = self.scaler[0]  
    self.y_pred = neural_net.predict(self.X_test)  
    self.y_pred = self.y_pred.reshape(-1,1)  
    self.y_pred_inv = scaler.inverse_transform(self.y_pred)  
    self.y_test_inv = scaler.inverse_transform(self.y_test)
```

Function: plot_result_verification

This is straight forward - the predicted result is plotted with the real values (test values) to validate the result given by the AI-model.

```
def plot_result_verification(self):  
    fig = plt.figure()  
    ax = fig.add_axes([0,0,1,1])  
  
    title = ""  
    label = ""  
  
    if self.param == 'longitude':  
        title = 'Prediction LONGITUDE ' + self.agent  
        label = 'LONGITUDE'  
    if self.param == 'latitude':  
        title = 'Prediction LATITUDE ' + self.agent  
        label = 'LATITUDE'  
  
    ax.plot(self.y_pred_inv)  
    ax.plot(self.y_test_inv)  
  
    ax.grid(True)  
    fig.autofmt_xdate()  
    plt.title(title)  
    plt.xlabel('Timeslot')  
    plt.ylabel(label)  
    plt.legend(['Prediction', 'Real value'])
```