Implementation of Deep Learning Algorithm for Solar Forecasting on Cloud/IoT Platform

Generated by Doxygen 1.9.4

Chapter 1

Implementation of Deep Learning Algorithm for Solar Forecasting on Cloud/IoT Platform

Authors

Basian Lesi

Chapter 2

Thesis Project

Implementation of Deep Learning Algorithm for Solar Forecasting on Cloud/IoT Platform.

2.1 Table of Contents

2.2 General Information

- Solar power forecasting pipeline (Data acquisition, processing, forecasting and publishing).
- Training an LSTM model for solar power forecasting, (training is not performed on Raspberry Pi).
- LSTM model is used to predict the solar power output for the next 24 hours.
- Predictions are uploaded to cloud (Google sheets).
- The project is hosted on a Raspberry Pi model 3b

2.3 Setup Windows or Linux Environment

To set up on Windows or Linux machine, in case of additionally training the model.

- Create a python virtual environment.
- install requirements `pip install -r requirements.txt`.

2.4 Setup Raspberry Pi Environment

It is recommended to install the specified versions, for compatibility reasons with RPi's cpu ARM architecture.

2.4.1 1. Install recommended OS, python and Tensorflow versions

- Raspberry pi OS.
  - **Debian version: 10 (buster)** : https://downloads.raspberrypi.org/raspios_lite_armhf/images/raspios_lite_armhf-2021-01-12/.
- python 3.7.3 (By default included in Debian 10 (buster)).
- Tensorflow - 2.4.0: https://github.com/lhelontra/tensorflow-on-arm/releases.

2.4.2 2. Install required libraries specified in requirements.txt file:

```
pip install -r requirements.txt
```

2.4.3 2.1 - If numpy installation error occurs follow this steps:

```
sudo apt-get update\
pip install cython
sudo apt-get install gcc python3.7-dev
pip install pycocotools==2.0.0
pip install --upgrade numpy==1.20.1
```

2.4.4 2.2 - Same if h5py installation error occurs:

```
pip uninstall h5py\
sudo apt-get install libhdf5-dev\
pip install h5py
pip install --upgrade h5py==2.10.0
```

2.4.5 3. Modify Tensorflow function

To overcome this issue: https://github.com/tensorflow/models/issues/9706.

by following glemarivero's suggestion:\ modify ∗∗_constant_if_small(value, shape, dtype, name)∗∗ function:
`sudo vim /home/pi/tensorflow/lib/python3.7/site-packages/tensorflow/python/ops/array_ops.py`

replace `np.prod(shape)` with `reduce_prod(shape)`, and \ import `from tensorflow.math import reduce_prod` : the modified code should look like this:

```
from tensorflow.math import reduce_prod # we import this on top of the file
def _constant_if_small(value, shape, dtype, name):\
  try:\
    if np.prod(shape) < 1000:
      return constant(value, shape=shape, dtype=dtype, name=name)\
  except TypeError:\
    # Happens when shape is a Tensor, list with Tensor elements, etc.\
    pass\
  return None
```

2.5 Obtain API keys

This project uses the below API's:

- https://openweathermap.org/ - to obtain weather data.
- https://console.cloud.google.com/ - Google Sheets where the forecasted data will be uploaded and displayed.

2.5.1 1. OpenWeather API key

1. Go to https://home.openweathermap.org/api_keys.

1. Create key -> insert the key name and press Generate.
2. Copy the generated key and add it on the existing OpenWeather_API_settings.json file.

3. The final file file should looks like below:

```
{"api_key": "xxxxxxxxxxxxxxx", "lat": "55.1449", "lon": "14.9170"}
```

2.5.2 2. Google Sheets API key

The aim is to create API keys for Google Sheets access and obtain credentials creds.json file similar to below:

```
{
    "type": "service_account",
    "project_id": "testsheets-xxxxxx",
    "private_key_id": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
    "private_key": "-----BEGIN PRIVATE KEY-----\nxxxxxxxxxxxxxxxxxxxxxxx\n-----END PRIVATE KEY-----\n",
    "client_email": "example@testsheets-xxxxx.iam.gserviceaccount.com",
    "client_id": "xxxxxxxxxxxx",
    "auth_uri": "https://accounts.google.com/o/oauth2/auth",
    "token_uri": "https://oauth2.googleapis.com/token",
    "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
    "client_x509_cert_url":
       "https://www.googleapis.com/robot/v1/metadata/x509/example%40testsheets-xxxxxx.iam.gserviceaccount.com"
}
```

1. Go to https://console.cloud.google.com/ Dashboard and CREATE PROJECT (follow the set up steps).

1. Navigation button -> APIs & Services -> ENABLE APIS AND SERVICES.

1. Select Google Sheets API -> ENABLE.

1. Go to Credentials tab -> CREATE CREDENTIALS -> Service account (follow the set up steps).

1. On Credentials tab -> click the email on Service Accounts section example@testsheets-xxxxx.↩ iam.gserviceaccount.com.

1. On Keys section click ADD KEY -> Create new key -> JSON -> creds.json file will be downloaded (rename to creds.json if necessary).

1. Create a new Google spreadsheet https://docs.google.com/spreadsheets/.

1. On the created spreadsheet page click Share -> add example@testsheets-xxxxx.iam.↩ gserviceaccount.com (with editor option) -> and press Share.

1. Add creds.json file to the project ./config/ directory.

2.6 Usage

After successfully completing the above steps, to start the pipeline run:

```
python ./src/main.py
```

Now the pipeline should be up and running. It should now successfully make solar power generation forecasting for the next 24 hours and upload to cloud. The forecasting is updated hourly.

2.6.0.1 Additional settings

In `src/config.py` file we can modify the settings below:

```
DEBUG = False # Set True to print debug messages
VISUAL = False # Set True to plot
TUNING = False # Set True to for hyperparameter tuning using bayesian optimization
SAVE_FIGURES = False # Set True to save figures
```

2.7 Project Status

in progress

2.8 Room for Improvement

- Develop a robust reliable source for fetching Solar and Wind power generated data for bornholm area past input in the model:
- Improve Wind power model by adding additional features weather features (wind direction)

2.9 Contact

Created by `@BasianLesi` - feel free to contact me!

Chapter 3

Namespace Index

3.1 Package List

Here are the packages with brief descriptions (if available):

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

Namespace Documentation

5.1 config Namespace Reference

- Project configurations (root project and useful directories), project settings modes (debug, visual, tuning, save_figures)

Functions

- None load_config_files ()

  Read OpenWeather_API_settings.json file and set global variables.
- None log (s)

  Modified python print function to check for DEBUG flag before printing to console.

Variables

- bool **DEBUG** = False
- bool **VISUAL** = False
- bool **TUNING** = False
- bool **SAVE_FIGURES** = False
- **ROOT_DIR** = str(Path(__file__).parent.parent)
- string **raw_data_dir** = ROOT_DIR+"/data/raw/"
- string **processed_data_dir** = ROOT_DIR+"/data/processed/"
- string **weather_dir** = ROOT_DIR+"/data/weather/"
- string **prediction_dir** = ROOT_DIR+"/data/predictions/"
- string **model_dir** = ROOT_DIR+"/models/"
- string **figures_dir** = ROOT_DIR+"/reports/figures/"
- string **metrics_dir** = ROOT_DIR+"/reports/metrics/"
- string **log_dir** = ROOT_DIR+"/models/tensorboard_logs/"
- string **config_dir** = ROOT_DIR+"/config/"
- int **day** = 60∗60∗24
- **today** = date.today()
- **seconds** = int(datetime.today().timestamp())
- int **tomorrow** = seconds + day;
- int **yesterday** = seconds - day;
- string **api_key** = ""
- string **lat** = ""
- string **lon** = ""
- string **yesterday_url** = ""
- string **today_url** = ""
- string **two_day_forecast_url** = ""

5.1.1 Detailed Description

- Project configurations (root project and useful directories), project settings modes (debug, visual, tuning, save_figures)

```
Project configuration module:

    1. Settings
        i.   DEBUG - Set to print debug messages
        ii.  VISUAL - Set True to plot
        iii. TUNING - Set True to for hyperparameter tuning using bayesian optimization
        iv.  SAVE_FIGURES - Set True to save figures

    2. Directories:
        i. raw_data_dir - Directory or raw data
        ii. processed_data_dir - Directory of processed data
        iii. weather_data_dir - Directory of weather data
        iv.  model_dir - Directory of saved models
        v. figures_dir - Directory of saved figures
        vi. metrics - Directory of models metric
        vii. log_dir - Directory of tensorboard logs

    3. log(s) - modified print() function
```

5.1.2 Function Documentation

5.1.2.1 load_config_files()

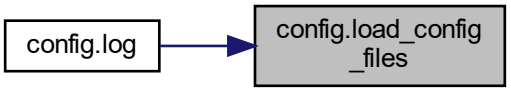
```
None config.load_config_files ( )
```

Read OpenWeather_API_settings.json file and set global variables.

Returns

None

Here is the caller graph for this function:



5.1.2.2 log()

```
None config.log (
              s )
```

Modified python print function to check for DEBUG flag before printing to console.

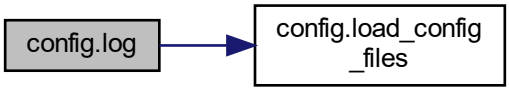
Parameters

s:string	- String to be printed out to console

Returns

None

Here is the call graph for this function:



5.2 data_fetching Namespace Reference

- Get weather forecasting data required as input to the model for the forecasting process.

Functions

- None update_global_variables ()

  Update global time and time dependent variables.
- pd.DataFrame weather_api_call (str url)

  API call to OpenWeather.com.
- str **generate_url** (int _seconds)
- None get_past_weather ()

  Get past weather data and save to past.csv.
- None get_future_weather ()

  Get future weather forecasting data and save to future.csv.
- None get_present_weather ()

  Get present weather data and save to future.csv.
- None update_data ()

  Update past, present and future data.
- pd.DataFrame normalize_column (pd.DataFrame df_forecast, int col=1, int a=0, int b=1)

  Normalize dataframe column on range[a,b].
- None make_predictions_data ()

  Process and normalize fetched data to be ready for forecasting model input.

5.2.1 Detailed Description

- Get weather forecasting data required as input to the model for the forecasting process.

```
Data fetching module:

    1. Fetch Weather Data
        i.   past - used as past datapoints in the forecasting model
        ii.  present - data at present time
        iii. future - weather forecasting for the next 48 hours

    2. Preprocessing:
        i.   Merge past, present and future data into one dataframe
        ii.  Extract features and normalize
        iii. Save data for predictions
```

5.2.2 Function Documentation

5.2.2.1 get_future_weather()

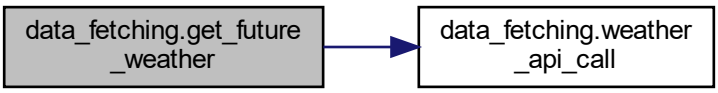
```
None data_fetching.get_future_weather ( )
```

Get future weather forecasting data and save to future.csv.

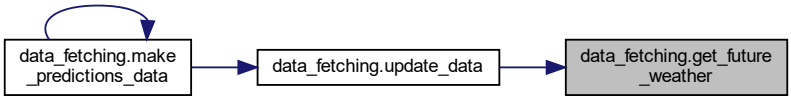
Returns

None

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.2 get_past_weather()

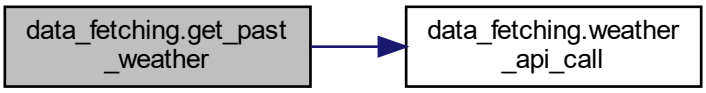
```
None data_fetching.get_past_weather ( )
```

Get past weather data and save to past.csv.

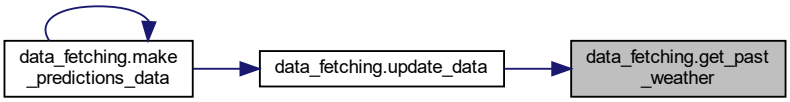
Returns

None

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.3 get_present_weather()

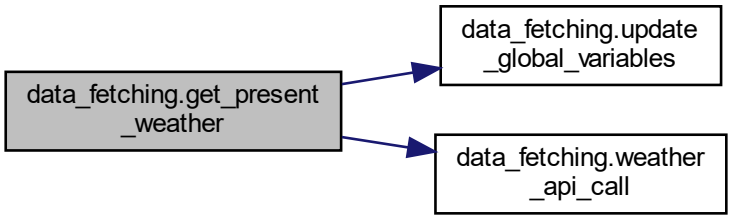
```
None data_fetching.get_present_weather ( )
```

Get present weather data and save to future.csv.

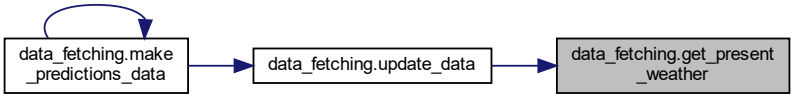
Returns

None

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.4 make_predictions_data()

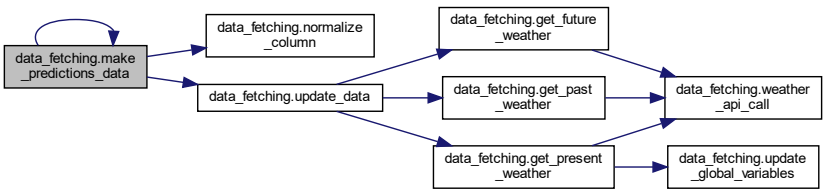
```
None data_fetching.make_predictions_data ( )
```

Process and normalize fetched data to be ready for forecasting model input.

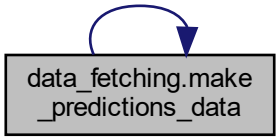
Returns

None

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.5 normalize_column()

```
pd.DataFrame data_fetching.normalize_column (
             pd.DataFrame df_forecast,
             int  col = 1,
             int a = 0,
             int b = 1 )
```

Normalize dataframe column on range[a,b].

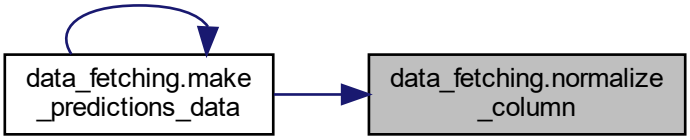
Parameters

df:pd.DataFrame	- Dataframe to normalize
col:int	- Index to column
a:int	- Min value (default 0)
b:int	- Max value (default 1)

Returns

pd.DataFrame - Normalized dataframe

Here is the caller graph for this function:



5.2.2.6 update_data()

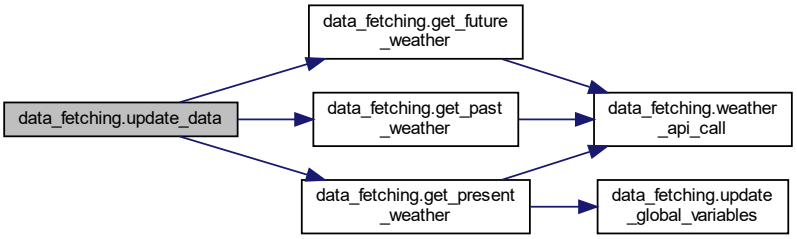
```
None data_fetching.update_data ( )
```

Update past, present and future data.

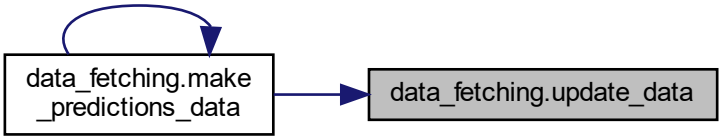
Returns

None

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.2.7 update_global_variables()

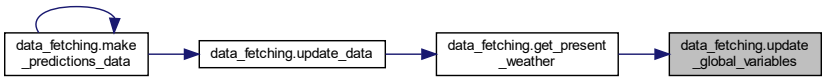
```
None data_fetching.update_global_variables ( )
```

Update global time and time dependent variables.

Returns

None

Here is the caller graph for this function:



5.2.2.8 weather_api_call()

```
pd.DataFrame data_fetching.weather_api_call (
             str url )
```

API call to OpenWeather.com.

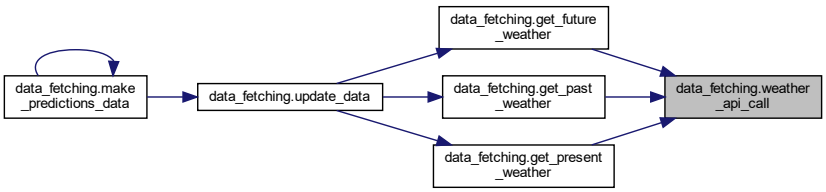
Parameters

url:string	- the url for the api request

Returns

pd.DataFrame - processed Dataframe with api response features of interest

Here is the caller graph for this function:



5.3 main Namespace Reference

- main pipeline function to run on loop

Variables

- int **sec** = 60
- int **min** = 60
- **hour** = float(sec∗min)
- **starttime** = time.time()

5.3.1 Detailed Description

- main pipeline function to run on loop

5.4 make_dataset Namespace Reference

- Module for data preprocessing and feature extraction

Functions

- None import_and_merge_data (str input_filepath, str output_filepath)

  Imports data from raw data directory preprocesses data, extracts features and generates data for model training.
- bool check_nan_values (pd.DataFrame df, int col=1)

  Checks if specified column of dataframe contains any NaN values.
- pd.DataFrame handle_nan_values (pd.DataFrame df, int col=1)

  Handles NaN values with a method depending on the set flag.
- pd.DataFrame normalize_column (pd.DataFrame df, int col=1, int a=0, int b=1)

  Normalizes specified index columb.

Variables

- bool **FILL_NAN_WITH_ZERO** = False
- bool **FILL_NAN_WITH_MEDIAN** = False
- bool **FILL_NAN_WITH_MEAN** = False
- bool **INTERPOLATE_NAN_VALUES** = False
- bool **DELETE_NAN_ROWS** = False
- bool **REPLACE_WITH_PREVIOUS_DAY** = False
- bool **LINEAR_REGRESSION** = True

5.4.1 Detailed Description

- Module for data preprocessing and feature extraction

5.4.2 Function Documentation

5.4.2.1 check_nan_values()

```
bool make_dataset.check_nan_values (
            pd.DataFrame df,
            int col = 1 )
```

Checks if specified column of dataframe contains any NaN values.

Parameters

df:pd.Dataframe	- dataframe to be iterated
col:int	column - index to check for nan values

Returns

bool - True if any nan values are found, False otherwise

5.4.2.2 handle_nan_values()

```
pd.DataFrame make_dataset.handle_nan_values (
            pd.DataFrame df,
            int  col = 1 )
```

Handles NaN values with a method depending on the set flag.

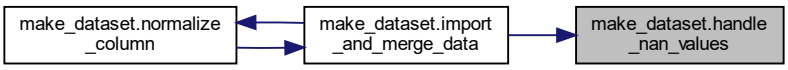
Parameters

df:pd.Dataframe	- dataframe to be processed
col:int	- index of column to be processed

Returns

pd.Dataframe - processed dataframe

Here is the caller graph for this function:



5.4.2.3 import_and_merge_data()

```
None make_dataset.import_and_merge_data (
            str input_filepath,
            str output_filepath )
```

Imports data from raw data directory preprocesses data, extracts features and generates data for model training.

Parameters

input_filepath:str	string path to raw data directory
output_filepath:str	string path to output directory

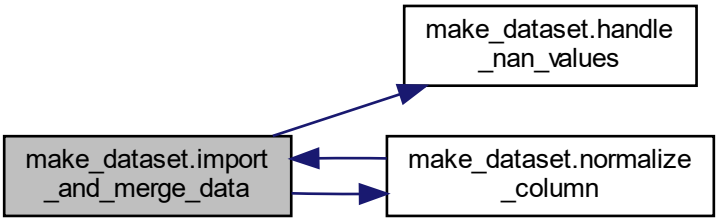
Generated files:

```
:merged.csv, merged_norm.csv: - Full dataset merged , normalized
:pv.csv, pv_norm: - dataset with features required for PV model training
:wp.csv, wp_norm: - dataset with features required for Wind model training / normalized
```

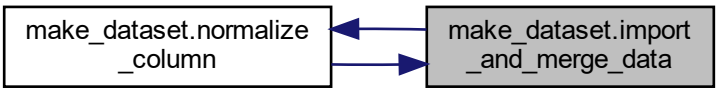
Returns

None

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.2.4 normalize_column()

```
pd.DataFrame make_dataset.normalize_column (
            pd.DataFrame df,
            int col = 1,
            int a = 0,
            int b = 1 )
```

Normalizes specified index columb.

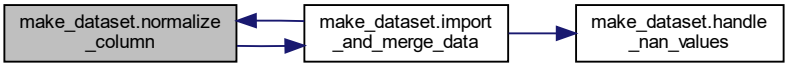
Parameters

df:pd.Dataframe	- dataframe to be iterated
col:int	- index to check for nan values
a:int	-

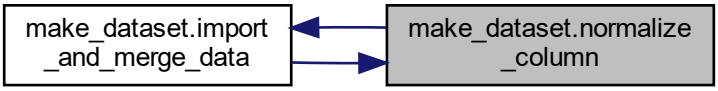
Returns

pd.Dataframe - dataframe with normalized column

Here is the call graph for this function:



Here is the caller graph for this function:



5.5 predict_model Namespace Reference

- Use best model to predict PV power for the next 48 hours

Functions

- def load_model_from_json (str model_name="model")

  Load specified model by name from models_dir.
- def normalize_column (pd.DataFrame df, int col=1, int a=0, int b=1)

  Normalize dataframe column on range[a,b].
- pd.DataFrame reverse_normalize (pd.DataFrame df, str col_name, int a=0, int b=1)

  Reverse normalize dataframe column from past range[a,b].
- pd.DataFrame add_day_sin_cos_and_normalize (pd.DataFrame df)

  Add sine and cosine to seconds of day and normalize dataframe.
- pd.DataFrame keep_next_24_hours_data (pd.DataFrame df, int seconds)

  Add sine and cosine to seconds of day and normalize dataframe.

- np.array predict_next_hour (pd.DataFrame df, model, int look_back=24, str target="PV power")

  Predict next hour of the target feature.
- None predict_pv_power (pd.DataFrame df, model, int look_back=24, str target="PV power")

  Function to make PV power predictions for a period of the available forecast weather data predictions are saved in the predictions folder as predicted.csv and pv_predicted_{Date}.csv.
- None forecast_PV_power ()

  Load model and Data and start the PV power prediction.

5.5.1 Detailed Description

- Use best model to predict PV power for the next 48 hours

5.5.2 Function Documentation

5.5.2.1 add_day_sin_cos_and_normalize()

```
pd.DataFrame predict_model.add_day_sin_cos_and_normalize (
             pd.DataFrame df )
```

Add sine and cosine to seconds of day and normalize dataframe.

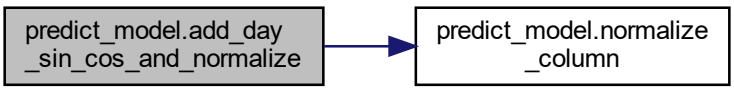
Parameters

df:pd.DataFrame	- Dataframe to normalize

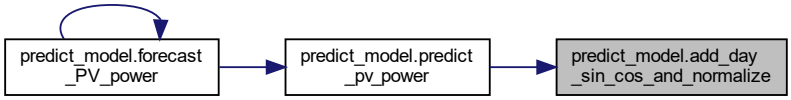
Returns

df:pd.DataFrame - processed dataframe

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.2.2 forecast_PV_power()

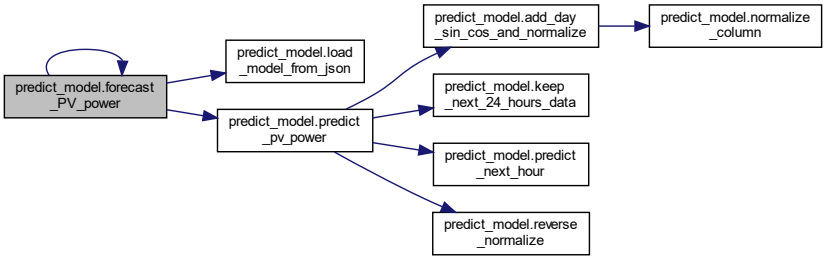
```
None predict_model.forecast_PV_power ( )
```

Load model and Data and start the PV power prediction.

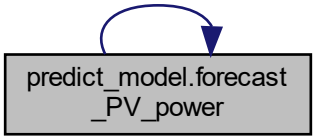
Returns

None

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.2.3 keep_next_24_hours_data()

```
pd.DataFrame predict_model.keep_next_24_hours_data (
            pd.DataFrame df,
            int seconds )
```

Add sine and cosine to seconds of day and normalize dataframe.

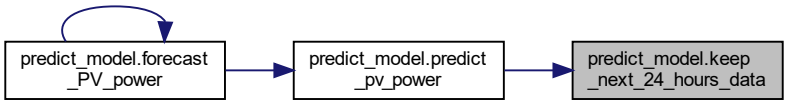
Parameters

df:pd.DataFrame	- Dataframe to normalize
seconds:int	- keep datapoints after this timestamp

Returns

df:pd.DataFrame - processed dataframe

Here is the caller graph for this function:



5.5.2.4 load_model_from_json()

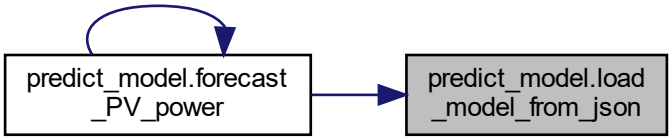
```
def predict_model.load_model_from_json (
            str model_name = "model" )
```

Load specified model by name from models_dir.

Parameters

model_name:str	- Name of the model return model:keras.models

Here is the caller graph for this function:



5.5.2.5 normalize_column()

```
def predict_model.normalize_column (
            pd.DataFrame df,
            int  col = 1,
            int a = 0,
            int b = 1 )
```

Normalize dataframe column on range[a,b].

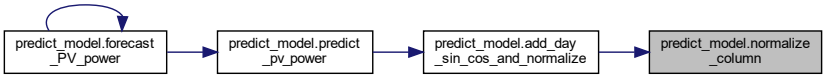
Parameters

df:pd.DataFrame	- Dataframe to normalize
col:int	- Index to column
a:int	- Min value (default 0)
b:int	- Max value (default 1)

Returns

df:pd.DataFrame - Normalized dataframe

Here is the caller graph for this function:



5.5.2.6 predict_next_hour()

```
np.array predict_model.predict_next_hour (
             pd.DataFrame df,
              model,
             int look_back = 24,
             str target = "PV power" )
```

Predict next hour of the target feature.

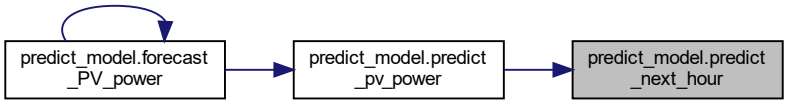
Parameters

df:pd.DataFrame	- Dataframe to normalize
model:keras.models	- model to be used for the the prediction
look_back:int	- timesteps to look back
target:string	- feature to predict

Returns

df:pd.DataFrame - processed dataframe

Here is the caller graph for this function:



5.5.2.7 predict_pv_power()

```
None predict_model.predict_pv_power (
             pd.DataFrame df,
              model,
             int look_back = 24,
             str target = "PV power" )
```

Function to make PV power predictions for a period of the available forecast weather data
predictions are saved in the predictions folder as predicted.csv and pv_predicted_{Date}.csv.

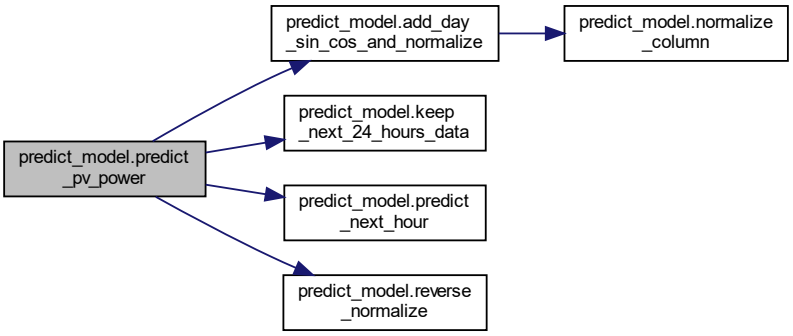
Parameters

df:pd.DataFrame	- Dataframe to normalize
model:keras.models	- model to be used for the the prediction
look_back:int	- timesteps to look back
target:string	- feature to predict

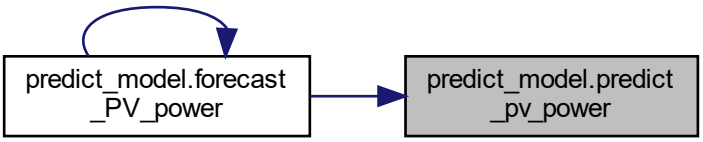
Returns

df:pd.DataFrame - processed dataframe

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.2.8 reverse_normalize()

```
pd.DataFrame predict_model.reverse_normalize (
            pd.DataFrame df,
            str col_name,
            int a = 0,
            int b = 1 )
```

Reverse normalize dataframe column from past range[a,b].

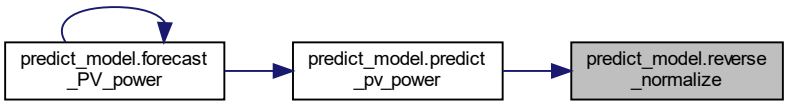
Parameters

df:pd.DataFrame	- Dataframe to normalize
col:int	- Index to column
a:int	- Min value (default 0)
b:int	- Max value (default 1)

Returns

df:pd.DataFrame - Normalized dataframe

Here is the caller graph for this function:



5.6 train_model Namespace Reference

- Module for training and tuning of models

Classes

- class Dataset
- class Model

Functions

- None create_directory_if_missing (str path)

  Helper function - Creates directory if it does not exist.
- pd.DataFrame reverse_normalize (pd.DataFrame df, str target)

  Normalize target value - used by benchmark for the predicted and actual values.
- Sequential bayesian_optimization (X_train, y_train, num_of_epochs)

  Bayesian optimization - used to find the best parameters for the model (20 best models are saved)
- Sequential build_model (hp)

  Bayesian optimization helper function hypermodel(hp) constructor.

Variables

- **dataset1** = Dataset(name = "pv_norm", look_back = 24, target = "PV power")
- **dataset2** = Dataset(name = "wp_norm", look_back = 24, target = "Wind power")
- **model1** = Model("pv_model", dataset1, 40, 64)
- **model2** = Model("wp_model", dataset2, 40, 64)

5.6.1 Detailed Description

- Module for training and tuning of models

```
train_model module - model training and tuning.

    1. Train PV and Wind power forecasting models
    2. Options for bayesian optimizatino tuning
    3. Benchmarking of models
    4. Save models training logs and display using tensorboard
    5. Save models for future use
```

5.6.2 Function Documentation

5.6.2.1 bayesian_optimization()

```
Sequential train_model.bayesian_optimization (
              X_train,
              y_train,
              num_of_epochs )
```

Bayesian optimization - used to find the best parameters for the model (20 best models are saved)

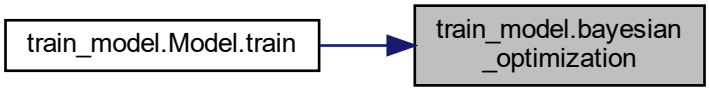
Parameters

X_train:numpy.ndarray	- training data
y_train:numpy.ndarray	- training target
num_of_epochs:int	- number of epochs

Returns

Sequential - trained model

Here is the caller graph for this function:



5.6.2.2 build_model()

```
Sequential train_model.build_model (
              hp )
```

Bayesian optimization helper function hypermodel(hp) constructor.

Parameters

hp:Hypermodel	- Hypermodel object

Returns

Sequential - trained model

5.6.2.3 create_directory_if_missing()

```
None train_model.create_directory_if_missing (
            str path )
```

Helper function - Creates directory if it does not exist.

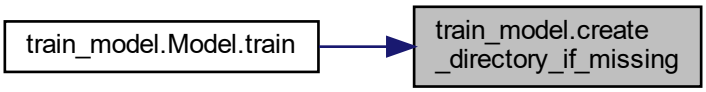
Parameters

path:string	- path to directory

Returns

:None

Here is the caller graph for this function:



5.6.2.4 reverse_normalize()

```
pd.DataFrame train_model.reverse_normalize (
            pd.DataFrame df,
            str target )
```

Normalize target value - used by benchmark for the predicted and actual values.

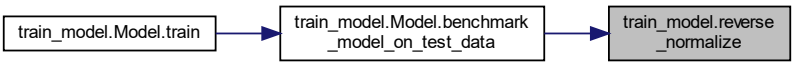
Parameters

df:DataFrame	- dataframe with target value
target:string	- target column name

Returns

DataFrame - normalized dataframe

Here is the caller graph for this function:



5.7 upload_to_cloud Namespace Reference

- Upload PV power forecasting to google sheets

Functions

- None upload_to_google_sheets (int hours=24)

  Uplaod predicted.csv data to the cloud.

5.7.1 Detailed Description

- Upload PV power forecasting to google sheets

5.7.2 Function Documentation

5.7.2.1 upload_to_google_sheets()

```
None upload_to_cloud.upload_to_google_sheets (
            int hours = 24 )
```

Uplaod predicted.csv data to the cloud.

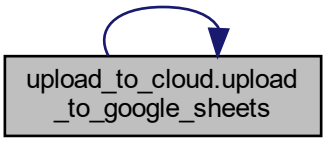
Parameters

hours:int	- hours of future predictions to be uploaded

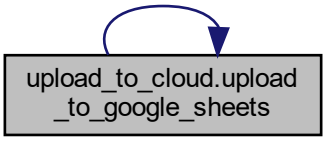
Returns

None

Here is the call graph for this function:



Here is the caller graph for this function:



5.8 visualization Namespace Reference

- Visualization functions

Functions

- def plot_predictions (pd.DataFrame df_pred, model, int start=0, int end=400)

  Plots model predictions vs actual values.
- def plot_model_history (model)

  Plots model history - Training vs Validation loss and RMSE.

5.8.1 Detailed Description

- Visualization functions

5.8.2 Function Documentation

5.8.2.1 plot_model_history()

```
def visualization.plot_model_history (
              model )
```

Plots model history - Training vs Validation loss and RMSE.

Parameters

df_pred:pd.Dataframe	- dataframe with the actual and predicted values
model:Model	- Model object

Note

- Generated plots are saved in figures_dir specified in config.py module

Returns

None

5.8.2.2 plot_predictions()

```
def visualization.plot_predictions (
            pd.DataFrame df_pred,
             model,
            int start = 0,
            int end = 400 )
```

Plots model predictions vs actual values.

Parameters

df_pred:pd.Dataframe	- dataframe with the actual and predicted values
model:Model	- Model object

Note

- Generated plots are saved in figures_dir specified in config.py module

Returns

None

Chapter 6

Class Documentation

6.1 train_model.Dataset Class Reference

Public Member Functions

- def __init__ (self, str name, int look_back=24, str target="PV power")

  Object initialization.
- None split_data (self, float t_size=0.8, float v_size=0.1)

  split data into train, validation and test sets
- Union[np.array, np.array] df_to_input_X_y (self)

  Generate input and target numpy arrays from dataframe.

Public Attributes

- **y_train**
- **y_val**
- **y_test**

6.1.1 Constructor & Destructor Documentation

6.1.1.1 __init__()

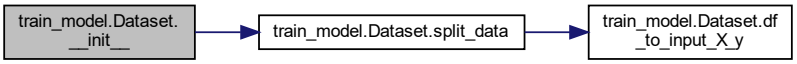
```
def train_model.Dataset.__init__ (
              self,
             str name,
             int look_back = 24,
             str target = "PV power" )
```

Object initialization.

Parameters

self:Dataset	- object instance
name:string	- name of dataset
look_back:int	- look back period for model training input shape
target:string	- target column name for model training

Here is the call graph for this function:



6.1.2 Member Function Documentation

6.1.2.1 df_to_input_X_y()

```
Union[np.array, np.array] train_model.Dataset.df_to_input_X_y (
              self )
```

Generate input and target numpy arrays from dataframe.

Parameters

self:Dataset	- object instance

Returns

np.array, np.array - input and target numpy arrays

Here is the caller graph for this function:

6.1.2.2 split_data()

```
None train_model.Dataset.split_data (
              self,
             float t_size = 0.8,
             float v_size = 0.1 )
```

split data into train, validation and test sets

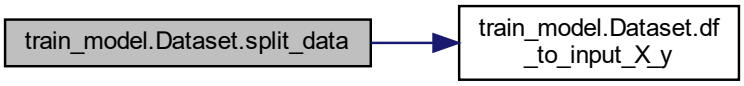
Parameters

self:Dataset	- object instance
t_size:float	- train data size (default 0.8)
v_size:float	- validation data size (default 0.2)

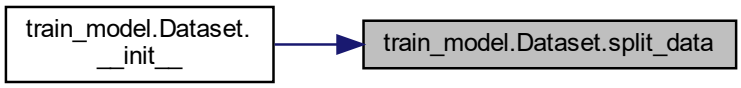
Returns

None

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following file:

- C:/Users/roni1/Thesis/solar-power-forecasting-thesis/src/train_model.py

6.2 train_model.Model Class Reference

Public Member Functions

- def __init__ (self, str name, Dataset dataset, int num_of_epochs=40, int num_of_units=64)

  Object initialization.
- None train (self)

  Model training.
- None save_model (self)

  Save trained model in h5 format.
- None benchmark_model_on_test_data (self)

  Trained model benchmark on test data.

Public Attributes

- **name**
- **model_dir**
- **dataset**
- **df**
- **look_back**
- **num_of_epochs**
- **num_of_units**
- **target**
- **timestamp**
- **trained_model**
- **model_history**

6.2.1 Constructor & Destructor Documentation

6.2.1.1 __init__()

```
def train_model.Model.__init__ (
              self,
             str name,
             Dataset dataset,
             int num_of_epochs = 40,
             int num_of_units = 64 )
```

Object initialization.

Parameters

self:Model	- object instance
name:string	- model name
dataset:Dataset	- dataset object instance containing training data
num_epochs:int	- number of epochs for model training (default 40)
num_of_units:int	- number of units in hidden layers (default 64)

6.2.2 Member Function Documentation

6.2.2.1 benchmark_model_on_test_data()

```
None train_model.Model.benchmark_model_on_test_data (
              self )
```

Trained model benchmark on test data.

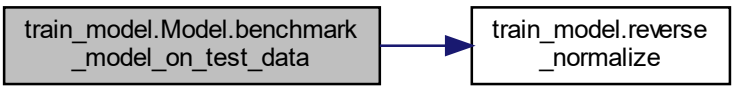
Parameters

self:Model	- object instance results and figures generated are saved in reports directory

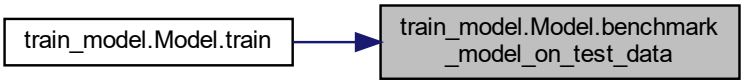
Returns

None

Here is the call graph for this function:



Here is the caller graph for this function:



6.2.2.2 save_model()

```
None train_model.Model.save_model (
              self )
```

Save trained model in h5 format.

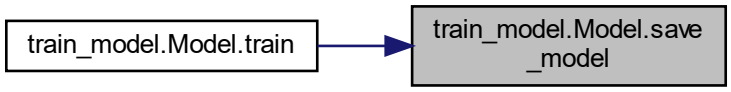
Parameters

self:Model	- object instance

Returns

None

Here is the caller graph for this function:



6.2.2.3 train()

```
None train_model.Model.train (
              self )
```

Model training.

Parameters

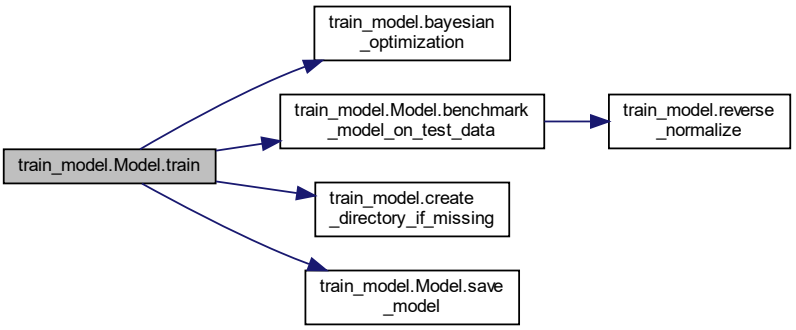
self:Model	- object instance

training model using Keras Sequential model
two LSTM layers are used for model training
one dropout Layer and two dense layers with relu activation
trained model is saved along with benchmark metrics and tensorboard logs

Returns

None

Here is the call graph for this function:



The documentation for this class was generated from the following file:

- C:/Users/roni1/Thesis/solar-power-forecasting-thesis/src/train_model.py

Index