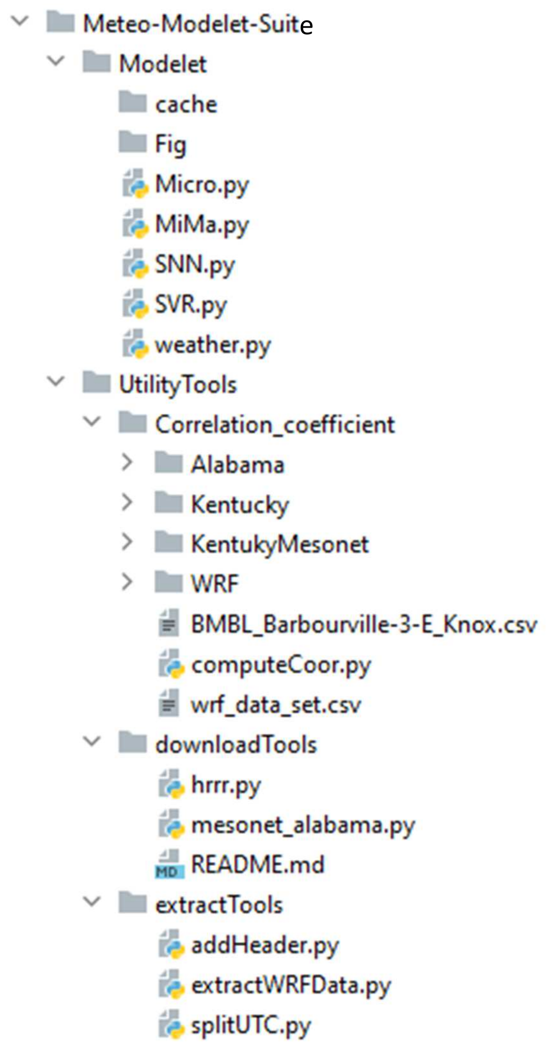
This file contains detailed information, including data collection, data preprocessing, and Meteo Modelet training and testing with the desired parameter sets. **PLEASE** read the file carefully before you start working on weather parameter predictions at locations in a targeted region. Although the University of South Alabama data set is not used in the current version of the article, its data collection and preprocessing components are included in this suite. You are assumed to have a working python environment. If not, please follow the section of **"Setting up Python Environment for Windows Platform"** at the end of this document, to get an environment ready.

Note: A set of preprocessed data (data/WRF_Data and data/Kentucky_Data) is also included in this suite. If you want to use these data sets, start reading Section 4.0 on the model of weather parameter predictions.

Resource directory structure:



```
Meteo-Modelet-Suite
  Modelet
    cache
    Fig
    Micro.py
    MiMa.py
    SNN.py
    SVR.py
    weather.py
  UtilityTools
    Correlation_coefficient
      Alabama
      Kentucky
      KentukyMesonet
      WRF
      BMBL_Barbourville-3-E_Knox.csv
      computeCoor.py
      wrf_data_set.csv
    downloadTools
      hrrr.py
      mesonet_alabama.py
      README.md
    extractTools
      addHeader.py
      extractWRFData.py
      splitUTC.py
```

1.0 Data Collection: *[check @: Meteo-Modelet-Suit/utilityTools/downloadTools]*

1.1 University of South Alabama (USA) Mesonet:

The following command downloads the dataset of all USA mesonet stations from 2019-01-1 to 2019-01-2 and stores at "./alabama" directory, where "." represents the current working directory.

python mesonet_alabama.py --begin_date 20190101 --end_date 20190103

1.2 Kentucky Mesonet:

Kentucky Mesonet data is not publicly available. Please request, download, and store them in the year-wise directory. For example, the dataset for the year 2019 should be stored in "./Kentucky/2019" directory.

1.3 Weather Research and Forecasting model (WRF):

The following command downloads the dataset in GRIB2 file format from 2019-01-1 to 2019-01-2 and stores them at the "./data" directory. One day data is organized on the hourly basis, including 24 ".grib2" files.

Note: The date range of January 1st, 2019 through March 1st, 2019 is currently invalid for hrrr.py.

(script from http://home.chpc.utah.edu/~u0553130/Brian_Blaylock/)

```
python hrrr.py --begin_date 20190101 --end_date 20190103
```

The output will look like:

```
data/2019/20190101/hrrr.20190101.00.00.grib2
data/2019/20190101/hrrr.20190101.01.00.grib2
data/2019/20190101/hrrr.20190101.02.00.grib2
```

2.0 Data extraction: *[check @: Meteo-Modelet-Suit/utilityTools/extractTools]*

2.1 USA Mesonet:

The downloaded data is organized on a daily basis. Since each file is already in the comma-separated CSV file, no further operations are needed to extract the data.

2.2 Kentucky Mesonet:

The downloaded data is organized on a yearly basis (say, "Kentucky/2019" for the year of 2019), and each file is in comma-separated CSV format. Since our Meteo Modelets need to select a range of training dates and testing dates, please run the following command to include the date and time in the dataset. This command will take a while to complete.

```
python splitUTC.py
```

2.3 Weather Research and Forecasting model (WRF):

2.3.1 The WRF dataset is in a compressed format, and it requires the 'pygrib' package for data decompression. If you have not installed 'pygrib', please install it and modify the following lines in "extractWRFData.py" to reflect respective parameters.

```
self.data_path = "data/"   # data source (grib file
location)
self.write_data_path = "WRF_Data/" # path to store
extracted data
self.start_date = date(2019,1,1)   # start date
self.end_date = date(2019, 1,3)    # end date
```

And execute the following command, which will extract 132 parameters for 39 Kentucky Mesonet stations and 26 USA Mesonet stations. The extracted data will be organized in a daily structure, and each file must contain 24 lines (hourly data).

```
python extractWRFData.py
```

2.3.2 The current version of "extractWRFData.py" cannot insert the header in the dataset automatically. Instead, please execute the following command for header insertion to each extracted file.

```
python addHeader.py
```

3.0 Feature Selection: *[check @: Meteo-Modelet-Suit/utilityTools/Correlation_coefficient]*

The dataset contains many features, and we are to predict a particular feature at a location by one MiMa modelet individually. Including all features in the input set to predict a given parameter is unsuitable for various reasons, such as time/space complexity rises and prediction accuracy declines due to additional noise introduced by irrelevant features. To select the relevant features of adequate significance, we gathered the extracted data of a given station (say, BMBL, one Kentucky Mesonet station) for a reasonable time duration (say, one-year dataset). Then, we compute the cross-correlation (or mutual information) among the features. Finally, the features are ranked according to the absolute value of the correlation coefficient (or mutual information) to pick top β relevant features, where β is a design parameter.

Would you please check the "Correlation_coefficient" directory to access the precomputed features rank? If you want to compute coefficients yourself, please compile a one-year dataset for a chosen station (say, Agricola in USA Mesonet) to produce a file (say, agricola_2019-01-01_2020-01-01.csv) and run 'computeCoor.py'.

```
python computeCoor.py
```

This action will create a file for each of the features. The file lists the relevant features in a descending order of the correlation degree.

Similarly, for other datasets (say, Kentucky Mesonet stations or WRF), you can follow similar steps to determine the most relevant features. For details, please look into the "computeCoor.py" file.

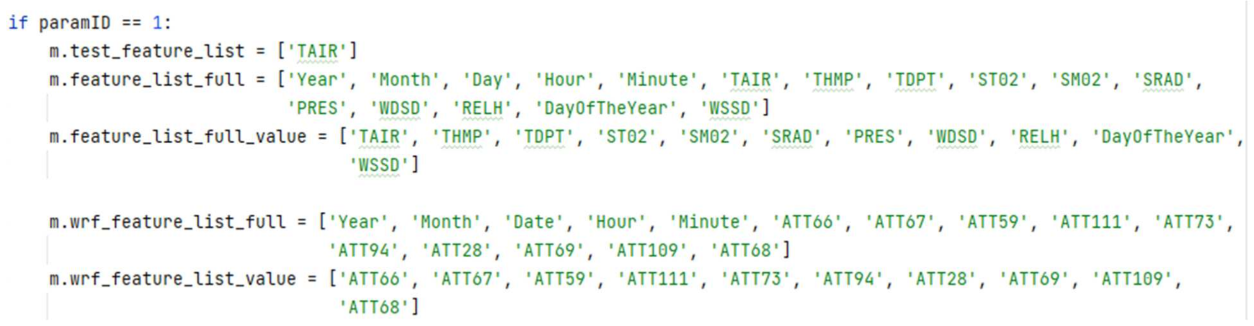
4.0 Weather parameter predictions model: *[check @: Meteo-Modelet-Suit/Modelet]*

We have two variants of Meteo modelets: (1) Micro and (4) MiMa. Each modelet needs to set a few environment variables/parameters before training it for later prediction use. Would you please specify the desired parameters in the modelet (say, Micro.py)? For your reference, the following aims to predict a parameter on August 20, 2020, at a Kentucky Mesonet station (**must be listed in stationList**) by using two years (2018 and 2019) seasonal datasets to train its modelet.

```
stationList = ['BMBL', 'QKSD'] # include your station list here

##################################################
# Training date range
```

```
during = 90  # How many days of data in each year
year_during = 2  # How many years of data
start_date = "0715"  # Start date of each year
year = "2018"  # Start year
seq = 5 # Prediction interval

# Prediction date range
T_year_during = 1 # This should be 1
T_during = 1      # How many days do you want to predict
T_start_date = "0820" # Start date to predict the parameter
T_year = "2020" #Year

model_epochs = 60 # Number of epochs for training
##################################################
```

Beneath these variables are the commands that set the weather feature to be tested as well as the features that are associated with it.



```
if paramID == 1:
    m.test_feature_list = ['TAIR']
    m.feature_list_full = ['Year', 'Month', 'Day', 'Hour', 'Minute', 'TAIR', 'THMP', 'TDPT', 'ST02', 'SM02', 'SRAD',
                           'PRES', 'WDSD', 'RELH', 'DayOfTheYear', 'WSSD']
    m.feature_list_full_value = ['TAIR', 'THMP', 'TDPT', 'ST02', 'SM02', 'SRAD', 'PRES', 'WDSD', 'RELH', 'DayOfTheYear',
                                 'WSSD']

    m.wrf_feature_list_full = ['Year', 'Month', 'Date', 'Hour', 'Minute', 'ATT66', 'ATT67', 'ATT59', 'ATT111', 'ATT73',
                               'ATT94', 'ATT28', 'ATT69', 'ATT109', 'ATT68']
    m.wrf_feature_list_value = ['ATT66', 'ATT67', 'ATT59', 'ATT111', 'ATT73', 'ATT94', 'ATT28', 'ATT69', 'ATT109',
                                'ATT68']
```

As you'll notice, for paramID == 1 (which belongs to the Temperature, see 4.2, below) a set of features is assigned. Similarly, for each paramID value, a collection of features is given.

4.1 Where is your training dataset?
Please supply the following three weather class data path attributes in the 'weather.py.'

```
class Weather():
  def __init__(self):
    self.data_path_0 = "/path/to/alabama_dataset/"
    self.wrf_data_path_0 = "/path/to/WRF_dataset/"
    self. kentucky_data_path_0 = "/path/to/Kentucky_dataset/"
```

NOTE: "/path/to/dataset/" must contain year directories, which store the daily (or yearly) data set for each observational station for alabama_dataset and WRF_dataset (or Kentucky_dataset) in the CSV format.

4.2 **Run a modelet**:
Command format:

```
python <modelet_name.py> <Station-Name> <Parameter-ID>
```

Example 1:

python Micro.py BMBL 1

In the above example command, (1) python is the interpreter needed to execute the program code, (2) Micro.py is a micro model based on the Kentucky Mesonet data set, (3) BMBL is the station name, and (4) 1 (integer number 1) is the paramID for Temperature. The paramID can go up to 4 (1: Temperature, 2: Relative humidity, 3: Wind speed, and 4: Pressure).

Example 2: Predict the Wind speed at the Kentucky BMBL station by a Micro modelet.
python Micro_base.py BMBL 3

Example 3: Predict the Pressure at the Kentucky BMTN station by a MiMa modelet.
python MiMa.py BMTN 4

Example 4: Predict the Pressure at the Kentucky BMTN station under the SNN (simple neural network) model.
python SNN.py BMTN 4

This action covers how to train a modelet, generate an actual vs. predicted graph (if the figure plot section in weather.py is enabled), and print the RMSE prediction accuracy metric before completing the program execution.

(a) In addition to printing the RMSE result, it saves the various error matrices and best prediction result in the cache directory under the file name specified by "global_path." Would you please look into the "Micro.py" for detailed information?

(b) File terminated with:

  I. *_MSE_error.txt* => logs RMSE, MAP, and MAPE error matrices on each run for a given paramID for the given Station.

  II. _Best_pred.txt => logs the best predicted result set for a given paramID for the given Station.

  III. .tru => records the true value

(c) Except for ML (SNN and SVR) model, _MSE_error.txt file contains 12 floating values (for the interval of 5, 10, ….., 55, and 60 minutes prediction range) followed by integer 1 (dummy value) for each run.

(d) You can access the data from .true file using load method of pickle package as,
data = pickle.load(open("/location/of/filename.tru", "rb")). And it loads the value into the 'data' array.

4.3 Additional models: SNN (Simple Neural Network, with three layers) and SVR (Support Vector Regression)

Unlike modelets, these models only predict a given seq (say, for a five-minute sequence, seq = 5) prediction interval. Thus, for predicting a series of 10, 15, … minutes, you must train the model for separate corresponding seq values. Please change the seq = DESIRED-VALUE in the SNN.py and SRV.py model if you like to run the models.

python SNN.py <Station-Name> <ParamID>

```
python  SRV.py <Station-Name> <ParamID>
```

5.0 GPU/CPU selection:

This section assumes that you are working with a Linux machine, installed with GPU card(s), and the related environment (say, cuda library) is ready. If "LD_LIBRARY_PATH" for cuda is not set appropriately, the modelet will run in CPU only. If this happens, please **locate "cuda/lib64" in your system and set the LD_LIBRARY_PATH at the end of your ".bashrc" file**. In my case, this location is at /usr/lib/cuda/lib64.

#Set LD_LIBRARY_PATH for cuda in .bashrc file as follows

```
export LD_LIBRARY_PATH=/usr/lib/cuda/lib64:$LD_LIBRARY_PATH

export LD_LIBRARY_PATH=/usr/lib/cuda/include:$LD_LIBRARY_PATH
```

Before importing TensorFlow and Keras, set the following environment variable to train the modelets (say, alabama_micro_base.py) in a specific GPU or CPU.

```
import os
os.environ ["CUDA_VISIBLE_DEVICES"]="0" # to execute in first GPU
os.environ ["CUDA_VISIBLE_DEVICES"]="1" # to execute in second GPU
os.environ ["CUDA_VISIBLE_DEVICES"]="-1" # to execute in CPU only
```

6.0 Running multiple models concurrently in GPU/CPU selection (tested in Linux platform):

Each modelet is programmed to randomly select a GPU (from the two available GPUs on each testbed in our Lab). While executing multiple modelets on both GPUs concurrently, our evaluation shows reasonable load distribution over the two GPUs. A simple script, shown below, can be used to lunch multiple modelets to predict various parameters at different stations. **We tested up to 56 modelets concurrently on a testbed without any issue, and all were completed within 30 minutes.**

Example: The following set of commands, listed in the lunch.sh file, runs concurrently to predict Temperature at BMTN station and relative humidity at BMBL station under different modelets.

1. copy the following commands (replace */path/to/modelet/* with your path) in the lunch.sh file

```
#!/bin/bash
# Contents of the lunch.sh file
python /path/to/modelet/ SNN.py BMTN 1 &
python /path/to/modelet/ SVR.py BMTN 1 &
python /path/to/modelet/Micro.py BMTN 1 &
python /path/to/modelet/Micro_base.py BMTN 1 &
python /path/to/modelet/MiMa.py BMTN 1 &
python /path/to/modelet/MiMa.py BMTN 1 &
python /path/to/modelet/ SNN.py BMBL 2 &
python /path/to/modelet/ SVR.py BMBL 2 &
python /path/to/modelet/Micro.py BMBL 2 &
python /path/to/modelet/Micro.py BMBL 2 &
python /path/to/modelet/MiMa.py BMBL 2 &
python /path/to/modelet/MiMa.py BMBL 2 &
```

2. *Set execution permission*

   ```
   chmod 755 lunch.sh (you might need sudo)
   ```

3. *Execute the bash file*

   ```
   ./lunch.sh
   ```

4. *Check the GPU status*

   ```
   nvidia-smi
   ```

5. *Collect results from cache directory*

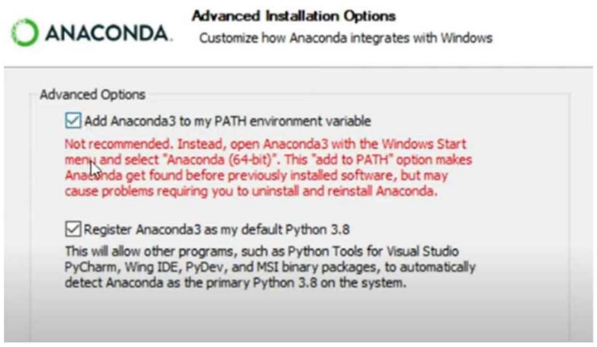
   _MSE_error.txt => contains error matrices

   _Best_pred.txt => contains prediction result (if run multiple times, the best one will be recorded).

Setting up Python Environment for Windows Platform

Steps are as follows for running the original MeteoModelet code on a Windows OS using the Anaconda navigator:

1. **Download Anaconda Navigator**
   a. Go to anaconda.com/products/individual
   b. Download the version of Anaconda corresponding to your operating system
   c. Once the setup file finishes downloading, double click on it to open
   d. In the setup window, accept the license and choose your desired destination folder
   e. If you wish, check the box that says "Add Anaconda3 to my PATH environment variable"



ANACONDA. **Advanced Installation Options**
Customize how Anaconda integrates with Windows

Advanced Options

☑ Add Anaconda3 to my PATH environment variable

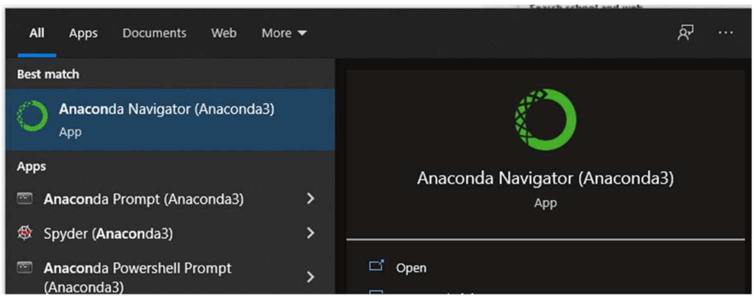
Not recommended. Instead, open Anaconda3 with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

☑ Register Anaconda3 as my default Python 3.8

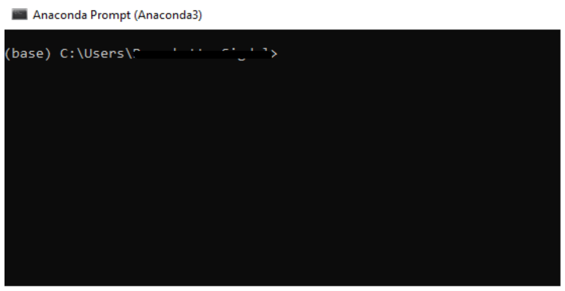
This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.8 on the system.

   f. Click finish

2. **Create a new environment**
   a. In your windows search bar, please search for the Anaconda Prompt and open it


All
Apps
Documents
Web
More
Best match
Anaconda Navigator (Anaconda3)
App
Apps
Anaconda Prompt (Anaconda3)
Spyder (Anaconda3)
Anaconda Powershell Prompt (Anaconda3)
Anaconda Navigator (Anaconda3)
App
Open

   b. Should open up to something like this:



Anaconda Prompt (Anaconda3)

```
(base) C:\Users\[redacted]>
```

The "(base)" means our standard environment. Create a different environment to each separate package and keep all packages organized. Some programs will work only if they have specific package versions. We create diverse environments to have their particular packages installed to not interfere with other running programs.

c. First, upgrade the pip

    pip install --upgrade pip

d. To create an environment, type the command

    conda create --n newEnv python=3.8 anaconda

    **Note: you can replace “newEnv” with the desired name of your choice.

e. When conda asks you to proceed, type “y”

```
Proceed ([y]/n)? y
```

f. Activate this environment

    conda activate newEnv

3. Download necessary packages for the environment

a. Install the following:

    pip install tensorflow==2.3.1
    pip install keras==2.4.3
    conda install -c conda-forge matplotlib
    conda install -c anaconda scikit-learn
    conda install -c anaconda pandas

b. Downgrade the numpy

    pip install -U numpy==1.18.5

c. Install pygrib

    conda install -c conda-forge pygrib

Setting up Python Environment for Linux Platform (Ubuntu 20.04)

1. Download the Anaconda installation script with wget (execute the following command):

wget -P /tmp https://repo.anaconda.com/archive/Anaconda3-2020.02-Linux-x86_64.sh

This action will downolad the “Anaconda3-2020.02-Linux-x86_64.sh” files at “/temp.”

2. Run the script to start the installation process

bash /tmp/Anaconda3-2020.02-Linux-x86_64.sh

You should see an output like the following:

```
Welcome to Anaconda3 2020.02

In order to continue the installation process, please review the license agreement.
Please, press ENTER to continue
>>>
```

Press ENTER to continue, you'll be asked to approve the license terms.

```
Do you approve the license terms? [yes|no]
```

Type yes to accept the license, and you'll be prompted to choose the installation location. For default location, please press ENTER key.

The installation may take some time, and once completed, the script will ask you whether you want to run conda init. Type yes.

```
Installation finished.

Do you wish the installer to initialize Anaconda3 by running conda init? [yes|no]
```

This will add the command-line tool conda to your system's PATH.

To activate the Anaconda installation, you can either close and re-open your shell or load the new PATH environment variable into the current shell session by typing:

```
source ~/.bashrc
```

To verify the installation type conda in your terminal.

3. Create a new environment

a. Execute the following command

```
conda create -n newEnv python=3.8
```

**Note: you can replace "newEnv" with the desired name of your choice.

b. Activate the environment

```
conda activate newEnv
```

4. Download necessary packages for the environment

a. First, upgrade the pip

```
pip install --upgrade pip
```

b. Install the following:

```
pip install tensorflow==2.3.1
pip install keras==2.4.3
conda install -c conda-forge matplotlib
conda install -c anaconda scikit-learn
conda install -c anaconda pandas
```

c. Downgrade packages

```
pip install -U numpy==1.18.5
pip install matplotlib==3.5.2
```

d. Install pygrib

```
conda install -c conda-forge pygrib
```