EEweather Documentation

Release 0.3.23

Phil Ngo

Apr 30, 2020

Contents

EEweather — tools for matching to and fetching data from NCDC ISD, TMY3, or CZ2010 weather stations.

EEweather comes with a database of weather station metadata, ZCTA metadata, and GIS data that makes it easier to find the right weather station to use for a particular ZIP code or lat/long coordinate.

CHAPTER 1

Installation

EEweather is a python package and can be installed with pip.

```
$ pip install eeweather
```

CHAPTER 2

Supported Sources of Weather Data

- NCDC Integrated Surface Database (ISD)
- Global Summary of the Day (GSOD)
- NREL Typical Meteorological Year 3 (TMY3)
- California Energy Commission 1998-2009 Weather Normals (CZ2010)

CHAPTER 3

Features

- Match by lat/long coordinates
- Convert ZIP code (ZCTA) to lat/long coordinates of its centroid
- Use user-supplied weather station mappings
- Match within climate zones
  - IECC Climate Zones
  - IECC Moisture Regimes
  - Building America Climate Zones
  - California Building Climate Zone Areas
- User-friendly SQLite database of metadata compiled from primary sources
  - US Census Bureau (ZCTAs, county shapefiles)
  - Building America climate zone county lists
  - NOAA NCDC Integrated Surface Database Station History
  - NREL TMY3 site
- Plot maps of outputs

CHAPTER 4

Command-line Usage

Once installed, `eeweather` can be run from the command-line. To see all available commands, run `eeweather --help`.

View ISD station metadata:

```
$ eeweather inspect_isd_station 722874
{
  "usaf_id": "722874",
  "wban_ids": "93134",
  "recent_wban_id": "93134",
  "name": "DOWNTOWN L.A./USC CAMPUS",
  "latitude": "+34.024",
  "longitude": "-118.291",
  "elevation": "+0054.6",
  "quality": "high",
  "iecc_climate_zone": "3",
  "iecc_moisture_regime": "B",
  "ba_climate_zone": "Hot-Dry",
  "ca_climate_zone": "CA_08"
}
```

Download raw ISD files:

```
$ wget `eeweather inspect_isd_filenames 722874 2017`
```

Download raw GSOD files:

```
$ wget `eeweather inspect_gsod_filenames 722874 2017`
```

Enter the SQLite command line for the metadata database:

```
$ eeweather inspect_db
SQLite version 3.19.3 2017-06-27 16:48:08
Enter ".help" for usage hints.
```

(continues on next page)

(continued from previous page)

```
sqlite> .tables
ba_climate_zone_metadata       isd_file_metadata
ca_climate_zone_metadata       isd_station_metadata
cz2010_station_metadata        tmy3_station_metadata
iecc_climate_zone_metadata     zcta_metadata
iecc_moisture_regime_metadata
sqlite> .headers on
sqlite> select * from isd_station_metadata where ca_climate_zone = 'CA_06' limit 10;
usaf_id|wban_ids|recent_wban_id|name|latitude|longitude|elevation|quality|iecc_
↪climate_zone|iecc_moisture_regime|ba_climate_zone|ca_climate_zone
722883|99999|99999|HERMOSA BEACH PIER|+33.870|-118.400|+0008.0|low|3|B|Hot-Dry|CA_06
722885|93197,99999|93197|SANTA MONICA MUNI AIRPORT|+34.016|-118.451|+0053.
↪0|high|3|B|Hot-Dry|CA_06
722913|99999|99999|MARINA DEL REY|+33.970|-118.430|+0008.0|low|3|B|Hot-Dry|CA_06
722917|99999|99999|LONG BEACH|+33.770|-118.170|+0003.0|low|3|B|Hot-Dry|CA_06
722933|99999|99999|SAN CLEMENTE|+33.420|-117.620|+0003.0|low|3|B|Hot-Dry|CA_06
722935|99999|99999|EL CAPITAN BEACH|+34.467|-120.033|+0027.0|low|3|C|Marine|CA_06
722950|23174|23174|LOS ANGELES INTERNATIONAL AIRPORT|+33.938|-118.389|+0029.
↪6|high|3|B|Hot-Dry|CA_06
722954|99999|99999|ZUMA BEACH|+34.020|-118.820|+0006.0|low|3|B|Hot-Dry|CA_06
722955|03122,03174,99999|03174|ZAMPERINI FIELD AIRPORT|+33.803|-118.340|+0029.
↪6|low|3|B|Hot-Dry|CA_06
722974|99999|99999|LONG BEACH|+33.767|-118.167|+0003.0|low|3|B|Hot-Dry|CA_06
sqlite> .quit
```

CHAPTER 5

Usage Guides

5.1 Basic Usage

This document describes how to get started with eeweather.

5.1.1 Matching to weather stations

EEweather is designed to support the process of finding sources of data that correspond to particular sites. As there are many approaches to this process of matching, the EEweather package is designed to be flexible.

EEweather provides sensible default mappings from geographical markers to weather stations so that it can be used out of the box.

EEweather uses lat/long coordinates as targets for weather matching. This method is described below.

Latitude/Longitude Coordinates

The recommended way to find the weather station(s) that correspond to a particular site is to use the lat-long coordinates of that site.

Example usage:

```
>>> import eeweather
>>> ranked_stations = eeweather.rank_stations(35, -95)
>>> station, warnings = eeweather.select_station(ranked_stations)
>>> station
ISDStation('720627')
>>> ranked_stations.loc[station.usaf_id]
rank                                  1
distance_meters                 32692.7
latitude                         35.283
longitude                         -95.1
```

(continues on next page)

(continued from previous page)

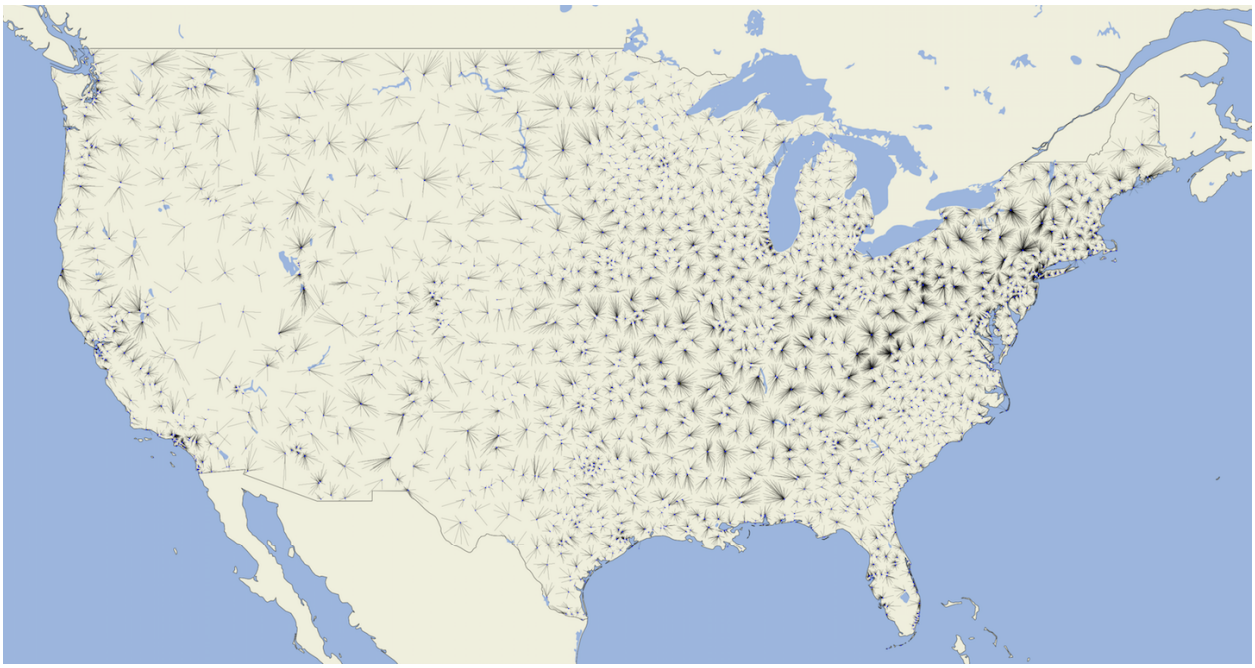
```
iecc_climate_zone                         3
iecc_moisture_regime                      A
ba_climate_zone                Mixed-Humid
ca_climate_zone                       None
rough_quality                          low
elevation                            183.2
state                                   OK
tmy3_class                            None
is_tmy3                              False
is_cz2010                            False
difference_elevation_meters           None
Name: 720627, dtype: object
>>> warnings
[]
```

That particular result has no associated warnings, but other mappings may have associated warnings, such as the mapping from this point which is in the middle of the Gulf of Mexico, 700km away from the nearest weather station and outside of the climate zone boundary:

```
>>> ranked_stations = eeweather.rank_stations(20, -95)
>>> station, warnings = eeweather.select_station(ranked_stations)
>>> warnings
['Distance from target to weather station is greater than 50km.', 'Distance from␣
↪target to weather station is greater than 200km.']
```

ZIP Code Tabulation Areas (ZCTAs)

ZIP codes are often abused as rough geographic markers. They are not particularly well set up be used as the basis of a GIS system - some ZIP codes correspond to single buildings or post-offices, some cover thousands of square miles of land. The US Census Bureau transforms census blocks into what they call ZIP Code Tabulation Areas, and use these instead. There are roughly 10k ZIP codes that are not used as ZCTAs, and ZCTAs do not correspond directly to ZIP codes, but for matching to weather stations, which are much sparser than ZIP codes, this rough mapping is usually sufficient. Often tens or hundreds of ZCTAs will be matched to the same weather station. We provide a function `eeweather.zcta_to_lat_long` which allows for a ZCTA to be converted into a latitude and longitude (the centroid of the ZCTA) which can be used to match to a weather station using the latitude/longitude method mentioned above.



Note: The default mapping concentrates on weather stations in US states (including AK, HI) and territories, including PR, GU, VI etc).

Example usage:

```
>>> lat, long = eeweather.zcta_to_lat_long('91104')
>>> lat, long
(34.1678418058534, -118.123485581459)
```

5.1.2 Obtaining temperature data

These matching results carry a reference to a weather station object. The weather station object has some associated metadata and - most importantly - has methods for obtaining weather data.

Let's look at the station object from above:

```
>>> station = result.isd_station
>>> station
ISDStation('722178')
```

This `ISDStation` object carries information about that station and methods for fetching corresponding weather data.

The `.json()` method gives a quick summary of associated metadata in a format that can easily be serialized:

```
>>> import json
>>> print(json.dumps(station.json(), indent=2)
{
  "elevation": 137.5,
  "latitude": 35.021,
  "longitude": -94.621,
  "icao_code": "KRKR",
  "name": "ROBERT S KERR AIRPORT",
```

(continues on next page)

(continued from previous page)


```
  "quality": "high",
  "wban_ids": [
    "53953",
    "99999"
  ],
  "recent_wban_id": "53953",
  "climate_zones": {
    "iecc_climate_zone": "3",
    "iecc_moisture_regime": "A",
    "ba_climate_zone": "Mixed-Humid",
    "ca_climate_zone": null
  }
}
```


Most of these are also stored as attributes on the object:

```
>>> station.usaf_id
'722178'
>>> station.latitude, station.longitude
(35.021, -94.621)
>>> station.coords
(35.021, -94.621)
>>> station.name
'ROBERT S KERR AIRPORT'
>>> station.iecc_climate_zone
'3'
>>> station.iecc_moisture_regime
'A'
```

In addition to these simple attributes there are a host of methods that can be used to fetch temperature data. The simplest are these, which return *pandas.Series* objects. The start and end date timezones must be explicilty set to UTC.

Note that this temperature data is given in degrees *Celsius*, not Fahrenheit. ($T_F = T_C \cdot 1.8 + 32$), and that the `pd.Timestamp` index is given in UTC.

ISD temperature data as an hourly time series:

```
>>> import datetime
>>> import pytz
>>> start_date = datetime.datetime(2016, 6, 1, tzinfo=pytz.UTC)
>>> end_date = datetime.datetime(2017, 9, 15, tzinfo=pytz.UTC)
>>> tempC = station.load_isd_hourly_temp_data(start_date, end_date)
>>> tempC.head()
2016-06-01 00:00:00+00:00    21.3692
2016-06-01 01:00:00+00:00    20.6325
2016-06-01 02:00:00+00:00    19.4858
2016-06-01 03:00:00+00:00    19.0883
2016-06-01 04:00:00+00:00    18.8858
Freq: H, dtype: float64
>>> tempF = tempC * 1.8 + 32
>>> tempF.head()
2016-06-01 00:00:00+00:00    70.46456
2016-06-01 01:00:00+00:00    69.13850
2016-06-01 02:00:00+00:00    67.07444
2016-06-01 03:00:00+00:00    66.35894
2016-06-01 04:00:00+00:00    65.99444
```

ISD temperature data as a daily time series:

```
>>> tempC = station.load_isd_daily_temp_data(start_date, end_date)
>>> tempC.head()
2016-06-01 00:00:00+00:00    21.329063
2016-06-02 00:00:00+00:00    21.674583
2016-06-03 00:00:00+00:00    22.434306
2016-06-04 00:00:00+00:00    22.842674
2016-06-05 00:00:00+00:00    21.850521
Freq: D, dtype: float64
>>> tempF = tempC * 1.8 + 32
>>> tempF.head()
2016-06-01 00:00:00+00:00    70.392313
2016-06-02 00:00:00+00:00    71.014250
2016-06-03 00:00:00+00:00    72.381750
2016-06-04 00:00:00+00:00    73.116813
2016-06-05 00:00:00+00:00    71.330937
Freq: D, dtype: float64
```

GSOD temperature data as a daily time series:

```
>>> tempC = station.load_gsod_daily_temp_data(start_date, end_date)
>>> tempC.head()
2016-06-01 00:00:00+00:00    21.111111
2016-06-02 00:00:00+00:00    21.833333
2016-06-03 00:00:00+00:00    22.277778
2016-06-04 00:00:00+00:00    22.777778
2016-06-05 00:00:00+00:00    21.833333
Freq: D, dtype: float64
>>> tempF = temps * 1.8 + 32
>>> tempF.head()
2016-06-01 00:00:00+00:00    70.0
2016-06-02 00:00:00+00:00    71.3
2016-06-03 00:00:00+00:00    72.1
2016-06-04 00:00:00+00:00    73.0
2016-06-05 00:00:00+00:00    71.3
Freq: D, dtype: float64
```

This station does not contain TMY3 data. To require that TMY3 data is available at the matched weather station, restrict the ranked weather stations to only those which have TMY3 data:

```
>>> ranked_stations = eeweather.rank_stations(35, -95, is_tmy3=True)
>>> station, warnings = eeweather.select_station(ranked_stations)
>>> station
ISDStation('723440')
```

TMY3 temperature data as an hourly time series:

```
>>> tempC = station.load_tmy3_hourly_temp_data(start_date, end_date)
>>> tempC.head()

2016-06-01 00:00:00+00:00    26.7
2016-06-01 01:00:00+00:00    26.3
2016-06-01 02:00:00+00:00    26.0
2016-06-01 03:00:00+00:00    25.6
2016-06-01 04:00:00+00:00    25.3
Freq: D, dtype: float64
>>> tempF = temps * 1.8 + 32
```

(continues on next page)

(continued from previous page)

```
>>> tempF.head()
2016-06-01 00:00:00+00:00    80.06
2016-06-01 01:00:00+00:00    79.34
2016-06-01 02:00:00+00:00    78.80
2016-06-01 03:00:00+00:00    78.08
2016-06-01 04:00:00+00:00    77.54
Freq: D, dtype: float64
```

A similar restriction can be made for CZ2010 stations, which are specific to California:

```
>>> ranked_stations = eeweather.rank_stations(35, -95, is_cz2010=True)
>>> station, warnings = eeweather.select_station(ranked_stations)
>>> station
ISDStation('723805')
```

CZ2010 temperature data as an hourly time series:

```
>>> tempC = station.load_cz2010_hourly_temp_data(start_date, end_date)
>>> tempC.head()
2016-06-01 00:00:00+00:00    26.7
2016-06-01 01:00:00+00:00    26.3
2016-06-01 02:00:00+00:00    26.0
2016-06-01 03:00:00+00:00    25.6
2016-06-01 04:00:00+00:00    25.3
Freq: D, dtype: float64
>>> tempF = temps * 1.8 + 32
>>> tempF.head()
2016-06-01 00:00:00+00:00    80.06
2016-06-01 01:00:00+00:00    79.34
2016-06-01 02:00:00+00:00    78.80
2016-06-01 03:00:00+00:00    78.08
2016-06-01 04:00:00+00:00    77.54
Freq: H, dtype: float64
```

The station ranking function *eeweather.rank_stations* has many more options, including distance restriction and climate zone restriction, which may come in handy.

If desired, *eeweather.ISDStation* objects can also be created directly:

```
>>> eeweather.ISDStation('722880')
ISDStation('722880')
```

If the station is not recognized, an error will be thrown:

```
>>> eeweather.ISDStation('BAD_STATION')
...
eeweather.exceptions.UnrecognizedUSAFIDError: BAD_STATION
```

5.2 Advanced Usage

Digging deeper into eeweather features.

5.2.1 Caching Weather Data

By default, a small SQLite database is setup at `~/.eeweather/cache.db` that is used to save weather data that is pulled from primary sources, such as the NOAA FTP site. This is done partially out of courtesy to the service, but also because it vastly speeds up the process of obtaining weather data. This local cache can be pointed to a different database by setting the environment variable *EEWEATHER_CACHE_URL* to any URL supported by SQLalchemy.

For example:

```
export EEWEATHER_CACHE_URL=postgres://user:password@host:port/dbname
```

5.2.2 ZCTA to latitude/longitude conversion

Convert ZCTA targets into latitude/longitudes based on their centroid:

```
>>> eeweather.zcta_to_lat_long(90210)
(34.1010279124639, -118.414760978568)
```

If the ZCTA or station is not recognized, an error will be thrown:

```
>>> eeweather.zcta_to_lat_long('BAD_ZCTA')
...
UnrecognizedZCTAError: BAD_STATION
```

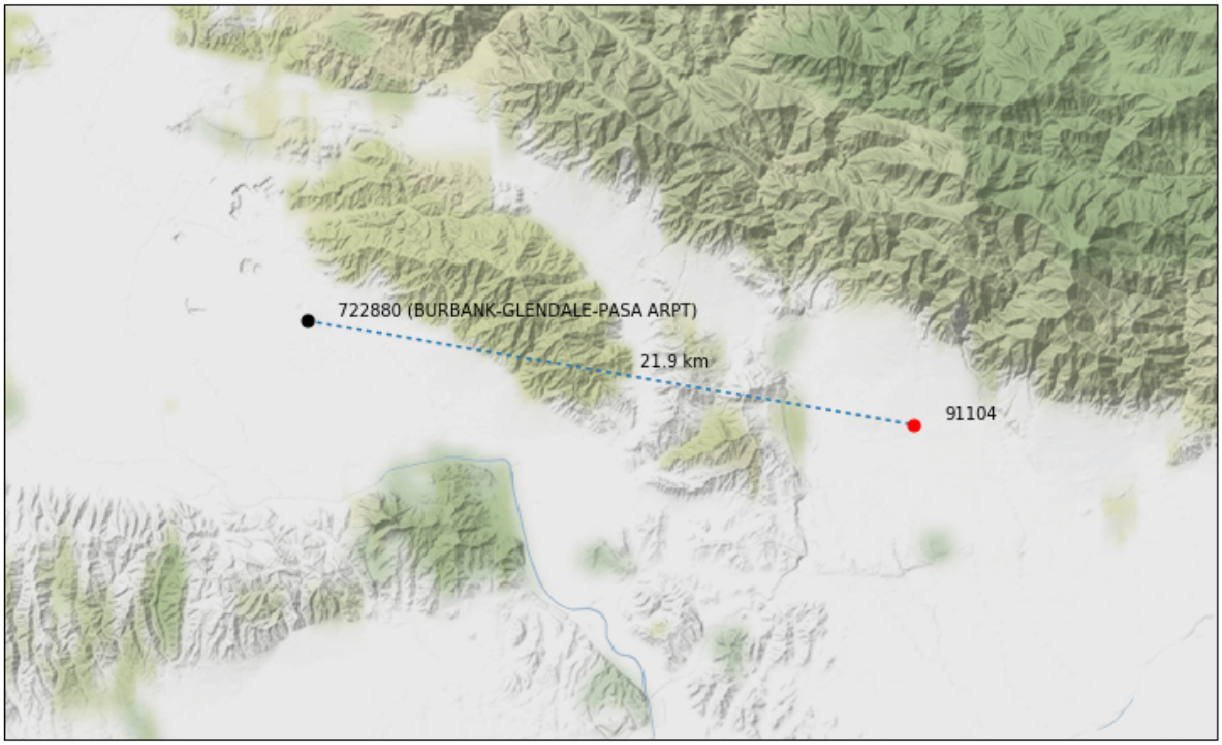
5.2.3 Charting Station mappings

Note: Requires *matplotlib* to be installed.

Within (for example) a jupyter notebook you can create plots like this:

```
>>> station = eeweather.ISDStation('722990')
>>> eeweather.plot_station_mapping(
...     lat, lng, station, distance_meters=21900, target='91104')
```

This will create a plot like the following:



5.2.4 Advanced database inspection

Using the CLI

If you prefer a GUI: SQLite Browser

The default database location is `~/.eeweather/cache.db`.

How to log into the database:

```
$ eeweather inspect_db
SQLite version 3.19.3 2017-06-27 16:48:08
Enter ".help" for usage hints.
sqlite>
```

List all tables:

```
sqlite> .tables
```

Turn on headers for results:

```
sqlite> .headers on
```

Example queries

Get more information about a specific ISD station.

```
select
  *
from
  isd_station_metadata
where
  usaf_id = '722860'
```

5.2.5 Rebuilding the Database

The metadata database can be rebuilt from primary sources using the CLI.

Exercise some caution when running this command, as it will overwrite the existing db:

```
$ eeweather rebuild_db
```

To see all options, run:

```
$ eeweather rebuild_db --help
Usage: eeweather rebuild_db [OPTIONS]

Options:
  --zcta-geometry / --no-zcta-geometry
  --iecc-climate-zone-geometry / --no-iecc-climate-zone-geometry
  --iecc-moisture-regime-geometry / --no-iecc-moisture-regime-geometry
  --ba-climate-zone-geometry / --no-ba-climate-zone-geometry
  --ca-climate-zone-geometry / --no-ca-climate-zone-geometry
  --n-closest-stations INTEGER
  --help                          Show this message and exit.
```

5.3 API Docs

5.3.1 Ranking

eeweather.**rank_stations**(*site_latitude*, *site_longitude*, *site_state=None*, *site_elevation=None*, *match_iecc_climate_zone=False*, *match_iecc_moisture_regime=False*, *match_ba_climate_zone=False*, *match_ca_climate_zone=False*, *match_state=False*, *minimum_quality=None*, *minimum_tmy3_class=None*, *max_distance_meters=None*, *max_difference_elevation_meters=None*, *is_tmy3=None*, *is_cz2010=None*)

Get a ranked, filtered set of candidate weather stations and metadata for a particular site.

Parameters

- **site_latitude** (*float*) – Latitude of target site for which to find candidate weather stations.
- **site_longitude** (*float*) – Longitude of target site for which to find candidate weather stations.
- **site_state** (*str, 2 letter abbreviation*) – US state of target site, used optionally to filter potential candidate weather stations. Ignored unless `match_state=True`.

- **site_elevation** (*float*) – Elevation of target site in meters, used optionally to filter potential candidate weather stations. Ignored unless `max_difference_elevation_meters` is set.
- **match_iecc_climate_zone** (*bool*) – If `True`, filter candidate weather stations to those matching the IECC climate zone of the target site.
- **match_iecc_moisture_regime** (*bool*) – If `True`, filter candidate weather stations to those matching the IECC moisture regime of the target site.
- **match_ca_climate_zone** (*bool*) – If `True`, filter candidate weather stations to those matching the CA climate zone of the target site.
- **match_ba_climate_zone** (*bool*) – If `True`, filter candidate weather stations to those matching the Building America climate zone of the target site.
- **match_state** (*bool*) – If `True`, filter candidate weather stations to those matching the US state of the target site, as specified by `site_state=True`.
- **minimum_quality** (str, `'high'`, `'medium'`, `'low'`) – If given, filter candidate weather stations to those meeting or exceeding the given quality, as summarized by the frequency and availability of observations in the NOAA Integrated Surface Database.
- **minimum_tmy3_class** (str, `'I'`, `'II'`, `'III'`) – If given, filter candidate weather stations to those meeting or exceeding the given class, as reported in the NREL TMY3 metadata.
- **max_distance_meters** (*float*) – If given, filter candidate weather stations to those within the `max_distance_meters` of the target site location.
- **max_difference_elevation_meters** (*float*) – If given, filter candidate weather stations to those with elevations within `max_difference_elevation_meters` of the target site elevation.
- **is_tmy3** (*bool*) – If given, filter candidate weather stations to those for which TMY3 normal year temperature data is available.
- **is_cz2010** (*bool*) – If given, filter candidate weather stations to those for which CZ2010 normal year temperature data is available.

Returns

ranked_filtered_candidates – Index is `usaf_id`. Each row contains a potential weather station match and metadata. Contains the following columns:

- `rank`: Rank of weather station match for the target site.
- `distance_meters`: Distance from target site to weather station site.
- `latitude`: Latitude of weather station site.
- `longitude`: Longitude of weather station site.
- `iecc_climate_zone`: IECC Climate Zone ID (1-8)
- `iecc_moisture_regime`: IECC Moisture Regime ID (A-C)
- `ba_climate_zone`: Building America climate zone name
- `ca_climate_zone`: Califoria climate zone number
- `rough_quality`: Approximate measure of frequency of ISD observations data at weather station.
- `elevation`: Elevation of weather station site, if available.

- `state`: US state of weather station site, if applicable.
- `tmy3_class`: Weather station class as reported by NREL TMY3, if available
- `is_tmy3`: Weather station has associated TMY3 data.
- `is_cz2010`: Weather station has associated CZ2010 data.
- `difference_elevation_meters`: Absolute difference in meters between target site elevation and weather station elevation, if available.

Return type pandas.DataFrame

eeweather.**combine_ranked_stations**(*rankings*)

Combine pandas.DataFrame s of candidate weather stations to form a hybrid ranking dataframe.

Parameters **rankings** (list of pandas.DataFrame) – Dataframes of ranked weather station candidates and metadata. All ranking dataframes should have the same columns and must be sorted by rank.

Returns **ranked_filtered_candidates** – Dataframe has a rank column and the same columns given in the source dataframes.

Return type pandas.DataFrame

eeweather.**select_station**(*candidates*, *coverage_range=None*, *min_fraction_coverage=0.9*, *distance_warnings=(50000, 200000)*, *rank=1*, *fetch_from_web=True*)

Select a station from a list of candidates that meets given data quality criteria.

Parameters **candidates** (pandas.DataFrame) – A dataframe of the form given by *eeweather.rank_stations* or *eeweather.combine_ranked_stations*, specifically having at least an index with `usaf_id` values and the column `distance_meters`.

Returns **isd_station, warnings** – A qualified weather station. `None` if no station meets criteria.

Return type tuple of (*eeweather.ISDStation*, list of str)

5.3.2 `ISDStation` objects

class eeweather.**ISDStation**(*usaf_id*, *load_metadata=True*)

A representation of an Integrated Surface Database weather station.

Contains data about a particular ISD station, as well as methods to pull data for this station.

Parameters

- **usaf_id** (*str*) – ISD station USAF ID
- **load_metatdata** (*bool, optional*) – Whether or not to auto-load metadata for this station

usaf_id

ISD station USAF ID

Type str

iecc_climate_zone

IECC Climate Zone

Type str

iecc_moisture_regime

IECC Moisture Regime

Type str

ba_climate_zone
Building America Climate Zone

Type str

ca_climate_zone
California Building Climate Zone

Type str

elevation
elevation of station

Type float

latitude
latitude of station

Type float

longitude
longitude of station

Type float

coords
lat/long coordinates of station

Type tuple of (float, float)

name
name of the station

Type str

quality
"high", "medium", "low"

Type str

wban_ids
list of WBAN IDs, or "99999" which have been used to identify the station.

Type list of str

recent_wban_id = None
WBAN ID most recently used to identify the station.

climate_zones = {}
dict of all climate zones.

cached_gsod_daily_temp_data_is_expired(*year*)
Return True if cache of resampled daily GSOD temperature data has expired or does not exist for the given year.

cached_isd_daily_temp_data_is_expired(*year*)
Return True if cache of resampled daily ISD temperature data has expired or does not exist for the given year.

cached_isd_hourly_temp_data_is_expired(*year*)
Return True if cache of resampled hourly ISD temperature data has expired or does not exist for the given year.

deserialize_cz2010_hourly_temp_data(*data*)
Deserialize JSON representation of hourly CZ2010 into pandas time series.

deserialize_gsod_daily_temp_data(*data*)
Deserialize JSON representation of resampled daily GSOD into pandas time series.

deserialize_isd_daily_temp_data(*data*)
Deserialize JSON representation of resampled daily ISD into pandas time series.

deserialize_isd_hourly_temp_data(*data*)
Deserialize JSON representation of resampled hourly ISD into pandas time series.

deserialize_tmy3_hourly_temp_data(*data*)
Deserialize JSON representation of hourly TMY3 into pandas time series.

destroy_cached_cz2010_hourly_temp_data()
Remove cached hourly CZ2010 temperature data to cache.

destroy_cached_gsod_daily_temp_data(*year*)
Remove cached resampled daily GSOD temperature data to cache for given year.

destroy_cached_isd_daily_temp_data(*year*)
Remove cached resampled daily ISD temperature data to cache for given year.

destroy_cached_isd_hourly_temp_data(*year*)
Remove cached resampled hourly ISD temperature data to cache for given year.

destroy_cached_tmy3_hourly_temp_data()
Remove cached hourly TMY3 temperature data to cache.

fetch_cz2010_hourly_temp_data()
Pull hourly CZ2010 temperature hourly time series from URL.

fetch_gsod_daily_temp_data(*year*)
Pull raw GSOD temperature data for the given year directly from FTP and resample to daily time series.

fetch_gsod_raw_temp_data(*year*)
Pull raw GSOD data for the given year directly from FTP.

fetch_isd_daily_temp_data(*year*)
Pull raw ISD temperature data for the given year directly from FTP and resample to daily time series.

fetch_isd_hourly_temp_data(*year*)
Pull raw ISD temperature data for the given year directly from FTP and resample to hourly time series.

fetch_isd_raw_temp_data(*year*)
Pull raw ISD data for the given year directly from FTP.

fetch_tmy3_hourly_temp_data()
Pull hourly TMY3 temperature hourly time series directly from NREL.

get_cz2010_hourly_temp_data_cache_key()
Get key used to cache CZ2010 weather-normalized temperature data.

get_gsod_daily_temp_data_cache_key(*year*)
Get key used to cache resampled daily GSOD temperature data for the given year.

get_gsod_filenames(*year=None*, *with_host=False*)
Get filenames of raw GSOD station data.

get_isd_daily_temp_data_cache_key(*year*)
Get key used to cache resampled daily ISD temperature data for the given year.

get_isd_file_metadata()
Get raw file metadata for the station.

get_isd_filenames(*year=None*, *with_host=False*)
Get filenames of raw ISD station data.

get_isd_hourly_temp_data_cache_key(*year*)
Get key used to cache resampled hourly ISD temperature data for the given year.

get_tmy3_hourly_temp_data_cache_key()
Get key used to cache TMY3 weather-normalized temperature data.

json()
Return a JSON-serializeable object containing station metadata.

load_cached_cz2010_hourly_temp_data()
Load all cached hourly TMY3 temperature data (the year is set to 1900)

load_cached_gsod_daily_temp_data()
Load all cached resampled daily GSOD temperature data.

load_cached_isd_daily_temp_data()
Load all cached resampled daily ISD temperature data.

load_cached_isd_hourly_temp_data()
Load all cached resampled hourly ISD temperature data.

load_cached_tmy3_hourly_temp_data()
Load all cached hourly TMY3 temperature data (the year is set to 1900)

load_cz2010_hourly_temp_data(*start*, *end*, *read_from_cache=True*, *write_to_cache=True*, *fetch_from_web=True*)
Load hourly CZ2010 temperature data from start date to end date (inclusive).

This is the primary convenience method for loading hourly CZ2010 temperature data.

Parameters

- **start** (*datetime.datetime*) – The earliest date from which to load data.
- **end** (*datetime.datetime*) – The latest date until which to load data.
- **read_from_cache** (*bool*) – Whether or not to load data from cache.
- **write_to_cache** (*bool*) – Whether or not to write newly loaded data to cache.
- **fetch_from_web** (*bool*) – Whether or not to fetch data from ftp.

load_cz2010_hourly_temp_data_cached_proxy(*fetch_from_web=True*)
Load hourly CZ2010 temperature data from cache, or if it is expired or hadn't been cached, fetch from URL.

load_gsod_daily_temp_data(*start*, *end*, *read_from_cache=True*, *write_to_cache=True*, *fetch_from_web=True*)
Load resampled daily GSOD temperature data from start date to end date (inclusive).

This is the primary convenience method for loading resampled daily GSOD temperature data.

Parameters

- **start** (*datetime.datetime*) – The earliest date from which to load data.
- **end** (*datetime.datetime*) – The latest date until which to load data.
- **read_from_cache** (*bool*) – Whether or not to load data from cache.
- **write_to_cache** (*bool*) – Whether or not to write newly loaded data to cache.
- **fetch_from_web** (*bool*) – Whether or not to fetch data from ftp.

load_gsod_daily_temp_data_cached_proxy(*year*, *fetch_from_web=True*)
Load resampled daily GSOD temperature data from cache, or if it is expired or hadn't been cached, fetch from FTP for given year.

load_isd_daily_temp_data(*start*, *end*, *read_from_cache=True*, *write_to_cache=True*, *fetch_from_web=True*)
Load resampled daily ISD temperature data from start date to end date (inclusive).

This is the primary convenience method for loading resampled daily ISD temperature data.

Parameters

- **start** (*datetime.datetime*) – The earliest date from which to load data.
- **end** (*datetime.datetime*) – The latest date until which to load data.
- **read_from_cache** (*bool*) – Whether or not to load data from cache.
- **fetch_from_web** (*bool*) – Whether or not to fetch data from ftp.
- **write_to_cache** (*bool*) – Whether or not to write newly loaded data to cache.

load_isd_daily_temp_data_cached_proxy(*year*, *fetch_from_web=True*)
Load resampled daily ISD temperature data from cache, or if it is expired or hadn't been cached, fetch from FTP for given year.

load_isd_hourly_temp_data(*start*, *end*, *read_from_cache=True*, *write_to_cache=True*, *fetch_from_web=True*, *error_on_missing_years=True*)
Load resampled hourly ISD temperature data from start date to end date (inclusive).

This is the primary convenience method for loading resampled hourly ISD temperature data.

Parameters

- **start** (*datetime.datetime*) – The earliest date from which to load data.
- **end** (*datetime.datetime*) – The latest date until which to load data.
- **read_from_cache** (*bool*) – Whether or not to load data from cache.
- **fetch_from_web** (*bool*) – Whether or not to fetch data from ftp.
- **write_to_cache** (*bool*) – Whether or not to write newly loaded data to cache.

load_isd_hourly_temp_data_cached_proxy(*year*, *fetch_from_web=True*)
Load resampled hourly ISD temperature data from cache, or if it is expired or hadn't been cached, fetch from FTP for given year.

load_tmy3_hourly_temp_data(*start*, *end*, *read_from_cache=True*, *write_to_cache=True*, *fetch_from_web=True*)
Load hourly TMY3 temperature data from start date to end date (inclusive).

This is the primary convenience method for loading hourly TMY3 temperature data.

Parameters

- **start** (*datetime.datetime*) – The earliest date from which to load data.
- **end** (*datetime.datetime*) – The latest date until which to load data.
- **read_from_cache** (*bool*) – Whether or not to load data from cache.
- **write_to_cache** (*bool*) – Whether or not to write newly loaded data to cache.
- **fetch_from_web** (*bool*) – Whether or not to fetch data from ftp.

load_tmy3_hourly_temp_data_cached_proxy(*fetch_from_web=True*)
Load hourly TMY3 temperature data from cache, or if it is expired or hadn't been cached, fetch from NREL.

read_cz2010_hourly_temp_data_from_cache()
Get cached version of hourly TMY3 temperature data.

read_gsod_daily_temp_data_from_cache(*year*)
Get cached version of resampled daily GSOD temperature data for given year.

read_isd_daily_temp_data_from_cache(*year*)
Get cached version of resampled daily ISD temperature data for given year.

read_isd_hourly_temp_data_from_cache(*year*)
Get cached version of resampled hourly ISD temperature data for given year.

read_tmy3_hourly_temp_data_from_cache()
Get cached version of hourly TMY3 temperature data.

serialize_cz2010_hourly_temp_data(*ts*)
Serialize hourly CZ2010 pandas time series as JSON for caching.

serialize_gsod_daily_temp_data(*ts*)
Serialize resampled daily GSOD pandas time series as JSON for caching.

serialize_isd_daily_temp_data(*ts*)
Serialize resampled daily ISD pandas time series as JSON for caching.

serialize_isd_hourly_temp_data(*ts*)
Serialize resampled hourly ISD pandas time series as JSON for caching.

serialize_tmy3_hourly_temp_data(*ts*)
Serialize hourly TMY3 pandas time series as JSON for caching.

validate_cz2010_hourly_temp_data_cache()
Check if CZ2010 data exists in cache.

validate_gsod_daily_temp_data_cache(*year*)
Delete cached resampled daily GSOD temperature data if it has expired for the given year.

validate_isd_daily_temp_data_cache(*year*)
Delete cached resampled daily ISD temperature data if it has expired for the given year.

validate_isd_hourly_temp_data_cache(*year*)
Delete cached resampled hourly ISD temperature data if it has expired for the given year.

validate_tmy3_hourly_temp_data_cache()
Check if TMY3 data exists in cache.

write_cz2010_hourly_temp_data_to_cache(*ts*)
Write hourly CZ2010 temperature data to cache for given year.

write_gsod_daily_temp_data_to_cache(*year*, *ts*)
Write resampled daily GSOD temperature data to cache for given year.

write_isd_daily_temp_data_to_cache(*year*, *ts*)
Write resampled daily ISD temperature data to cache for given year.

write_isd_hourly_temp_data_to_cache(*year*, *ts*)
Write resampled hourly ISD temperature data to cache for given year.

write_tmy3_hourly_temp_data_to_cache(*ts*)
Write hourly TMY3 temperature data to cache for given year.

5.3.3 Summaries

eeweather.summaries.**get_zcta_ids**(*state=None*)
Get ids of all supported ZCTAs, optionally by state.

Parameters **state** (*str, optional*) – Select zipcodes only from this state or territory, given as 2-letter abbreviation (e.g., 'CA', 'PR').

Returns **results** – List of all supported selected ZCTA IDs.

Return type list of str

eeweather.summaries.**get_isd_station_usaf_ids**(*state=None*)
Get USAF IDs of all supported ISD stations, optionally by state.

Parameters **state** (*str, optional*) – Select ISD station USAF IDs only from this state or territory, given as 2-letter abbreviation (e.g., 'CA', 'PR').

Returns **results** – List of all supported selected ISD station USAF IDs.

Return type list of str

5.3.4 Geography

eeweather.geo.**get_lat_long_climate_zones**(*latitude*, *longitude*)
Get climate zones that contain lat/long coordinates.

Parameters

- **latitude** (*float*) – Latitude of point.
- **longitude** (*float*) – Longitude of point.

Returns **climate_zones** – Region ids for each climate zone type.

Return type dict of str

eeweather.geo.**get_zcta_metadata**(*zcta*)
Get metadata about a ZIP Code Tabulation Area (ZCTA).

Parameters **zcta** (*str*) – ID of ZIP Code Tabulation Area

Returns **metadata** – Dict of data about the ZCTA, including lat/long coordinates.

Return type dict

eeweather.geo.**zcta_to_lat_long**(*zcta*)
Get location of ZCTA centroid

Retrieves latitude and longitude of centroid of ZCTA to use for matching with weather station.

Parameters **zcta** (*str*) – ID of the target ZCTA.

Returns

- **latitude** (*float*) – Latitude of centroid of ZCTA.
- **longitude** (*float*) – Target Longitude of centroid of ZCTA.

5.3.5 Database

eeweather.database.**build_metadata_db**(*zcta_geometry=False*, *iecc_climate_zone_geometry=True*, *iecc_moisture_regime_geometry=True*, *ba_climate_zone_geometry=True*, *ca_climate_zone_geometry=True*)

Build database of metadata from primary sources.

Downloads primary sources, clears existing DB, and rebuilds from scratch.

Parameters

- **zcta_geometry** (*bool, optional*) – Whether or not to include ZCTA geometry in database.
- **iecc_climate_zone_geometry** (*bool, optional*) – Whether or not to include IECC Climate Zone geometry in database.
- **iecc_moisture_regime_geometry** (*bool, optional*) – Whether or not to include IECC Moisture Regime geometry in database.
- **ba_climate_zone_geometry** (*bool, optional*) – Whether or not to include Building America Climate Zone geometry in database.
- **ca_climate_zone_geometry** (*bool, optional*) – Whether or not to include California Building Climate Zone Area geometry in database.

5.3.6 Exceptions

exception eeweather.**EEWeatherError**

Base class for exceptions in the eeweather package.

exception eeweather.**ISDDataNotAvailableError**(*usaf_id*, *year*)

Raised when ISD data is not available for a particular station and year.

usaf_id

the USAF ID for which ISD data does not exist.

Type str

year

the year for which ISD data does not exist.

Type int

message

a message describing the error

Type str

exception eeweather.**UnrecognizedZCTAError**(*value*)

Raised when an unrecognized ZCTA is encountered.

value

the value which is not a valid ZCTA

Type str

message

a message describing the error

Type str

exception eeweather.**UnrecognizedUSAFIDError**(*value*)
Raised when an unrecognized USAF station id is encountered.

value
the value which is not a valid USAF ID

Type str

message
a message describing the error

Type str

5.3.7 Validators

eeweather.validation.**valid_zcta_or_raise**(*zcta*)
Check if ZCTA is valid and raise eeweather.UnrecognizedZCTAError if not.

eeweather.validation.**valid_usaf_id_or_raise**(*usaf_id*)
Check if USAF ID is valid and raise eeweather.UnrecognizedUSAFIDError if not.

5.3.8 Visualization

eeweather.**plot_station_mapping**(*target_latitude*, *target_longitude*, *isd_station*, *distance_meters*, *target_label='target'*)
Plots this mapping on a map.

eeweather.**plot_station_mappings**(*mapping_results*)
Plot a list of mapping results on a map.

Requires matplotlib and cartopy.

Parameters **mapping_results** (*list of MappingResult objects*) – Mapping results to plot

Index

I

J

L

M

N

P

Q

R

S

U

V

W

Y

Z