HistoricScraper

View Source

class HistoricScraper: View Source

This class scrapes weather.gov historical weather data pages and pulls max & min temps, snow, and rainfall for each day starting from June 2017 to the present (April 2021, as I write this). One thing to note is that the only method that needs to be manually called (via a if __name__=='__main__' block) is the scraper() method. All other methods are called in the scraping process by scraper().

All data tables from the weather.gov histroical site are set up the exact same way, which makes extracting the data as simple as being able to deal with python strings and lists and methods pertaining to these data types. A sample web page to be scraped can be found by visiting the below url:

https://forecast.weather.gov/product.php?site=NWS&issuedby=PIT&product=CF6&format=CI&version=14&glossary=0

HistoricScraper() View Source

def visit_url(self, url): View Source

Uses urllib.request.urlopen to open specified url (provided that the status code received from this request is 200) and returns html document in raw bytes form. Can throw a ValueError if the url is invalid (not returning a 200 code) and can also throw an AttributeError if url is not entered as a string.

Args: url- string representation of url of desired webpage

Returns: raw data (in bytes) that was obtained from opening the requested url if status code is 200. Otherwise, return None Raises: ValueError if url argument is invalid but still in passed as a string AttributeError if url argument is not a string

def get_month(self, header): View Source

Pass header information from data table as a list so that month/year pair can be extracted and returned. Note that list of header information (obtained in main body of weather() method) should always have length 37(except Jan. Feb. of 2020, they have an extra element in their header. Can throw a TypeError if header is not passed as a list

Args: header: header information from scraped web page(Contains date info) and should be passed as a list

Returns: a tuple where the first element is the month of weather observations and second element is the corresponding year

Raises: TypeError if header is not passed as a list KeyError if date information passed has invalid month key

def table_splitter(self, table, delim): View Source

Splits data table from scraped web page using Python's built-in split() function. The argument to split() is the delimeter (string of equals signs used in the table on the web page beings scraped. Note: for the weather.gov historical data, this should ALWAYS return a list with 6 elements (all hist- orical weather observation tables have the same format). Can give a TypeError if table is not passed as a string.

Args: table - string representation of historical weather observations table scraped from weather.gov web page delim - delimeter used on the web page to orient table

Returns: list of elements from the table after splitting on the given delimeter.

Raises: TypeError if table or delim args are not passed as strings

def scrape_ify(self, data, when): View Source

Builds a dictionary with relevant weather data information for a single month

Args: data - a single row of weather observations from the data table, entered as a string! when - a 2-tuple where the first element is the month and second element is the year of corresponding data table (get_month function returns a tuple of this form)

Returns: dictionary containing relevant scraped weather information

Raises: TypeError if date infomation is wrong or weather information was input incorrectly in the table. If the latter is the case, the observation will just be skipped (though I don't think this is a problem I have encountered thus far

def weather_extracter(self, url): View Source

Scrapes raw daily historical weather data from weather.gov and then uses the scrape_ify function to build a dictionary of weather observations for each day in a given month

Args: url: weather.gov url that contains weather data to be extracted

Returns: List of dictionaries of weather data for a single month

Raises: urllib.error.URLError if url is invalid AttributeError is raised when a url for a non-weather.gov site is entered

def scraper(self): View Source

This function is the driving force that scrapes all 50 weather.gov pages of historical data by calling the weather_extracter function, and is the only function that needs to be called in order successfully scrape weather data. Note that the urls for the weather.gov historical data pages differ only in one spot, and that is in the version='' attribute within the url. Each version is a number between 1 and 50 and corresponds to months of weather observations (i.e. April 2021 is version 1, March 2021 is version 2, ... July 2017 is version 49, and June 2017 is version 50).

Args: None

Returns: Nested list of dictionaries for each day that contain weather observation data across all available months. Each day's weather observations will be in a dictionary, a single month of these observation dictionaries will be a list, and each of the 47 months worth of data is then stored in another list

Raises: Any exceptions raised will be thrown by the weather_extracer function

def pretty_print(self, data, length): View Source

Prints data values from a nested list so that one element is output per line.

Args: data: Nested list of data whose elements are to be printed.
length: integer number of elements of the data argument to be printed. If length argument is longer than the length the data argument, then the method breaks and a warning message is dispayed

Returns: None, but does give a good output of data that is (semi) easy to read

Raises: TypeError is raised when arguments are not of the right type

def weather_yester_data(self): View Source

Fetches weather observations from previous day

Args: None

Returns: Dictionary of previous day weather observations

Raises: As long as the datetime module from Python does not change, this method should not raise any errors