hockey_scraper Documentation

Release 1.1

Harry Shomer

Feb 11, 2018

Contents

CHAPTER 1

Contents

hockey_scraper

The hockey_scraper module contains all of the functions used for scraping.

Scraping

There are three ways to scrape games:

1. *Scrape by Season*:

Scrape games on a season by season level (Note: A given season is referred to by the first of the two years it spans. So you would refer to the 2016-2017 season as 2016).

```
import hockey_scraper

# Scrapes the 2015 & 2016 season with shifts and stores the data in a Csv file (both␣
↪are equivalent!!!)
hockey_scraper.scrape_seasons([2015, 2016], True)
hockey_scraper.scrape_seasons([2015, 2016], True, data_format='Csv')

# Scrapes the 2008 season without shifts and returns a json string of the data
scraped_data = hockey_scraper.scrape_seasons([2008], False, data_format='Json')

# Scrapes 2014 season without shifts including preseason games
hockey_scraper.scrape_seasons([2014], False, preseason=True)
```

2. *Scrape by Game*:

Scrape a list of games provided. All game ID's can be found using this link (you need to play around with the dates in the url).

```
import hockey_scraper
```

```
# Scrapes the first game of 2014, 2015, and 2016 seasons with shifts and stores the␣
↪data in a Csv file
hockey_scraper.scrape_games([2014020001, 2015020001, 2016020001], True)

# Scrapes the first game of 2007, 2008, and 2009 seasons with shifts and returns a␣
↪Json string of the data
scraped_data = hockey_scraper.scrape_games([2007020001, 2008020001, 2009020001], True,
↪ data_format='Json')
```

3. *Scrape by Date Range*:

Scrape all games between a specified date range. All dates must be written in a "yyyy-mm-dd" format.

```
import hockey_scraper

# Scrapes all games between date range without shifts and stores the data in a Csv␣
↪file (both are equivalent!!!)
hockey_scraper.scrape_date_range('2016-10-10', '2016-10-20', False)
hockey_scraper.scrape_date_range('2016-10-10', '2016-10-20', False, preseason=False)

# Scrapes all games between 2015-1-1 and 2015-1-15 without shifts and returns a Json␣
↪string of the data
scraped_data = hockey_scraper.scrape_date_range('2015-1-1', '2015-1-15', False, data_
↪format='Json')

# Scrapes all games from 2014-09-15 to 2014-11-01 with shifts including preseason␣
↪games
hockey_scraper.scrape_date_range('2014-09-15', '2014-11-01', True, preseason=True)
```

Notes:

1. For all three functions you must specify if you want to also scrape shifts (TOI tables) with a boolean. The Play by Play is automatically scraped.

2. When scraping by date range or by season, preseason games aren't scraped unless otherwise specified.

3. For all three functions the scraped data is deposited into a Csv file unless it's specified to return it as a Json string.

4. The Json string returned is structured like so:

```
# When scraping by game or date range
"
{
   'pbp': [
      Plays
   ],
   'shifts': [
      Shifts
   ]
}
"

# When scraping by season
"
{
   'pbp': {
      'Seasons': [
         Plays
      ]
   },
```

```
    'shifts': {
        'Seasons': [
            Plays
        ]
    }
}
"

# For example, if you scraped the 2008 and 2009 seasons the Json will look like this:
"
{
    'pbp': {
        '2008': [
            Plays
        ],
        '2009': [
            Plays
        ]
    },
    'shifts': {
        '2008': [
            Shifts
        ],
        '2009': [
            Shifts
        ]
    }
}
"
```

Functions

Scrape Functions

Functions to scrape by season, games, and date range

hockey_scraper.scrape_functions.**check_data_format**(*data_format*)

Checks if data_format specified (if it is at all) is either None, 'Csv', or 'json'. It exits program with error message if input isn't good.

Parameters **data_format** – data_format provided

Returns None

hockey_scraper.scrape_functions.**check_valid_dates**(*from_date*, *to_date*)

Check if it's a valid date range

Parameters

- **from_date** – date should scrape from
- **to_date** – date should scrape to

Returns None, will exit if not valid

hockey_scraper.scrape_functions.**scrape_date_range**(*from_date*, *to_date*, *if_scrape_shifts*, *data_format='csv'*, *preseason=False*)

Scrape games in given date range

Parameters

- **from_date** – date you want to scrape from
- **to_date** – date you want to scrape to
- **if_scrape_shifts** – Boolean indicating whether to also scrape shifts
- **data_format** – format you want data in - csv or json (csv is default)
- **preseason** – Boolean indicating whether include preseason games (default if False)

Returns Json string or None

hockey_scraper.scrape_functions.**scrape_games**(*games*, *if_scrape_shifts*, *data_format='csv'*)

Scrape a list of games

Parameters

- **games** – list of game_ids
- **if_scrape_shifts** – Boolean indicating whether to also scrape shifts
- **data_format** – format you want data in - csv or json (csv is default)
- **preseason** – Boolean indicating whether include preseason games (default if False)

Returns Json string or None

hockey_scraper.scrape_functions.**scrape_list_of_games**(*games*, *if_scrape_shifts*)

Given a list of game_id's (and a date for each game) it scrapes them

Parameters

- **games** – list of [game_id, date]
- **if_scrape_shifts** – Boolean indicating whether to also scrape shifts

Returns DataFrame of pbp info, also shifts if specified

hockey_scraper.scrape_functions.**scrape_seasons**(*seasons*, *if_scrape_shifts*, *data_format='csv'*, *preseason=False*)

Given list of seasons it scrapes all the seasons

Parameters

- **seasons** – list of seasons
- **if_scrape_shifts** – Boolean indicating whether to also scrape shifts
- **data_format** – format you want data in - csv or json (csv is default)
- **preseason** – Boolean indicating whether include preseason games (default if False)

Returns Json string or None

hockey_scraper.scrape_functions.**to_csv**(*file_name*, *pbp_df*, *shifts_df*)

Write DataFrame(s) to csv file(s)

Parameters

- **file_name** – name of file
- **pbp_df** – pbp DataFrame
- **shifts_df** – shifts DataFrame

Returns None

Game Scraper

This module contains code to scrape data for a single game

hockey_scraper.game_scraper.**check_goalie**(*row*)
Checks for bad goalie names (you can tell by them having no player id)

Parameters **row** – df row

Returns None

hockey_scraper.game_scraper.**combine_espn_html_pbp**(*html_df*, *espn_df*, *game_id*, *date*, *away_team*, *home_team*)
Merge the coordinate from the espn feed into the html DataFrame

Parameters

- **html_df** – DataFrame with info from html pbp
- **espn_df** – DataFrame with info from espn pbp
- **game_id** – json game id
- **date** – ex: 2016-10-24
- **away_team** – away team
- **home_team** – home team

Returns merged DataFrame

hockey_scraper.game_scraper.**combine_html_json_pbp**(*json_df*, *html_df*, *game_id*, *date*)
Join both data sources

Parameters

- **json_df** – json pbp DataFrame
- **html_df** – html pbp DataFrame
- **game_id** – id of game
- **date** – date of game

Returns finished pbp

hockey_scraper.game_scraper.**combine_players_lists**(*json_players*, *roster_players*, *game_id*)
Combine the json list of players (which contains id's) with the list in the roster html

Parameters

- **json_players** – dict of all players with id's
- **roster_players** – dict with home and and away keys for players
- **game_id** – id of game

Returns dict containing home and away keys -> which contains list of info on each player

hockey_scraper.game_scraper.**get_players_json**(*json*)
Return dict of players for that game

Parameters **json** – gameData section of json

Returns dict of players->keys are the name (in uppercase)

hockey_scraper.game_scraper.**get_teams_and_players**(*game_json*, *roster*, *game_id*)
Get list of players and teams for game

Parameters

- **game_json** – json pbp for game
- **roster** – players from roster html
- **game_id** – id for game

Returns dict for both - players and teams

hockey_scraper.game_scraper.**print_errors**()
Print errors with scraping

Returns None

hockey_scraper.game_scraper.**scrape_game**(*game_id*, *date*, *if_scrape_shifts*)
This scrapes the info for the game. The pbp is automatically scraped, and the whether or not to scrape the shifts is left up to the user.

Parameters

- **game_id** – game to scrap
- **date** – ex: 2016-10-24
- **if_scrape_shifts** – Boolean indicating whether to also scrape shifts

Returns DataFrame of pbp info (optional) DataFrame with shift info otherwise just None

hockey_scraper.game_scraper.**scrape_pbp**(*game_id*, *date*, *roster*, *game_json*, *players*, *teams*)
Automatically scrapes the json and html, if the json is empty the html picks up some of the slack and the espn xml is also scraped for coordinates.

Parameters

- **game_id** – json game id
- **date** – date of game
- **roster** – list of players in pre game roster
- **game_json** – json pbp for game
- **players** – dict of players
- **teams** – dict of teams

Returns DataFrame with info or None if it fails

hockey_scraper.game_scraper.**scrape_shifts**(*game_id*, *players*, *date*)
Scrape the Shift charts (or TOI tables)

Parameters

- **game_id** – json game id
- **players** – dict of players with numbers and id's
- **date** – date of game

Returns DataFrame with info or None if it fails

Html PBP

This module contains functions to scrape the Html Play by Play for any given game

hockey_scraper.html_pbp.**add_event_players**(*event_dict*, *event*, *players*, *home_team*)
Add players involved in the event to event_dict

Parameters

- **event_dict** – dict of parsed event stuff
- **event** – fixed up html
- **players** – dict of players and id's
- **home_team** – home team

Returns None

hockey_scraper.html_pbp.**add_event_team**(*event_dict*, *event*)
Add event team for event

Parameters

- **event_dict** – dict of event info
- **event** – list with parsed event info

Returns None

hockey_scraper.html_pbp.**add_home_zone**(*event_dict*, *home_team*)
Determines the zone relative to the home team and add it to event

Parameters

- **event_dict** – dict of event info
- **home_team** – home team

Returns None

hockey_scraper.html_pbp.**add_period**(*event_dict*, *event*)
Add period for event

Parameters

- **event_dict** – dict of event info
- **event** – list with parsed event info

Returns None

hockey_scraper.html_pbp.**add_score**(*event_dict*, *event*, *current_score*, *home_team*)
Change if someone scored...also change current score

Parameters

- **event_dict** – dict of parsed event stuff
- **event** – event info from pbp
- **current_score** – current score in game
- **home_team** – home team for game

Returns None

hockey_scraper.html_pbp.**add_strength**(*event_dict*, *home_players*, *away_players*)
Get strength for event -> It's home then away

Parameters

- **event_dict** – dict of event info
- **home_players** – list of players for home team
- **away_players** – list of players for away team

Returns None

hockey_scraper.html_pbp.**add_time**(*event_dict*, *event*)
Fill in time and seconds elapsed

Parameters

- **event_dict** – dict of parsed event stuff
- **event** – event info from pbp

Returns None

hockey_scraper.html_pbp.**add_type**(*event_dict*, *event*)
Add "type" for event -> either a penalty or a shot type

Parameters

- **event_dict** – dict of event info
- **event** – list with parsed event info

Returns None

hockey_scraper.html_pbp.**add_zone**(*event_dict*, *play_description*)
Determine which zone the play occurred in (unless one isn't listed) and add it to dict

Parameters

- **event_dict** – dict of event info
- **play_description** – the zone would be included here

Returns Off, Def, Neu, or NA

hockey_scraper.html_pbp.**clean_html_pbp**(*html*)
Get rid of html and format the data

Parameters **html** – the requested url

Returns a list with all the info

hockey_scraper.html_pbp.**get_pbp**(*game_id*)
Given a game_id it returns the raw html Ex: http://www.nhl.com/scores/htmlreports/20162017/PL020475.HTM

Parameters **game_id** – the game

Returns raw html of game

hockey_scraper.html_pbp.**get_penalty**(*play_description*)
Get the penalty info

Parameters **play_description** – description of play field

Returns penalty info

hockey_scraper.html_pbp.**get_player_name**(*number*, *players*, *team*, *home_team*)
This function is used for the description field in the html. Given a last name and a number it return the player's full name and id.

Parameters

- **number** – player's number
- **players** – all players with info
- **team** – team of player
- **home_team** – home team

Returns dict with full and and id

hockey_scraper.html_pbp.**get_soup**(*game_html*)

Uses Beautiful soup to parses the html document. Some parsers work for some pages but don't work for others....I'm not sure why so I just try them all here in order

Parameters **game_html** – html doc

Returns "soupified" html and player_shifts portion of html (it's a bunch of td tags)

hockey_scraper.html_pbp.**if_valid_event**(*event*)

Checks if it's a valid event ('#' is meaningless and I don't like the other ones) to parse

Parameters **event** – list of stuff in pbp

Returns boolean

hockey_scraper.html_pbp.**parse_event**(*event*, *players*, *home_team*, *if_plays_in_json*, *current_score*)

Receives an event and parses it

Parameters

- **event** – event type
- **players** – players in game
- **home_team** – home team
- **if_plays_in_json** – If the pbp json contains the plays
- **current_score** – current score for both teams

Returns dict with info

hockey_scraper.html_pbp.**parse_html**(*html*, *players*, *teams*, *if_plays_in_json*)

Parse html game pbp

Parameters

- **html** – raw html
- **players** – players in the game (from json pbp)
- **teams** – dict with home and away teams
- **if_plays_in_json** – If the pbp json contains the plays

Returns DataFrame with info

hockey_scraper.html_pbp.**populate_players**(*event_dict*, *players*, *away_players*, *home_players*)

Populate away and home player info (and num skaters on each side) NOTE: Could probably do this in a much neater way...

Parameters

- **event_dict** – dict with event info
- **players** – all players in game and info

- **away_players** – players for away team
- **home_players** – players for home team

Returns None

hockey_scraper.html_pbp.**return_name_html**(*info*)
In the PBP html the name is in a format like: 'Center - MIKE RICHARDS' Some also have a hyphen in their last name so can't just split by '-'

Parameters **info** – position and name

Returns name

hockey_scraper.html_pbp.**scrape_game**(*game_id*, *players*, *teams*, *if_plays_in_json*)
Scrape the data for the game

Parameters

- **game_id** – game to scrape
- **players** – dict with player info
- **teams** – dict with home and away teams
- **if_plays_in_json** – boolean, if the plays are in the json

Returns DataFrame of game info or None if it fails

hockey_scraper.html_pbp.**shot_type**(*play_description*)
Determine which zone the play occurred in (unless one isn't listed)

Parameters **play_description** – the type would be in here

Returns the type if it's there (otherwise just NA)

hockey_scraper.html_pbp.**strip_html_pbp**(*td*)
Strip html tags and such

Parameters **td** – pbp

Returns list of plays (which contain a list of info) stripped of html

Json PBP

This module contains functions to scrape the Json Play by Play for any given game

hockey_scraper.json_pbp.**change_event_name**(*event*)
Change event names from json style to html ex: BLOCKED_SHOT to BLOCK

Parameters **event** – event type

Returns fixed event type

hockey_scraper.json_pbp.**get_pbp**(*game_id*)
Given a game_id it returns the raw json Ex: http://statsapi.web.nhl.com/api/v1/game/2016020475/feed/live

Parameters **game_id** – the game

Returns raw json of game or None if couldn't get game

hockey_scraper.json_pbp.**get_teams**(*pbp_json*)
Get teams

Parameters **json** – pbp json

Returns dict with home and away

hockey_scraper.json_pbp.**parse_event**(*event*)
Parses a single event when the info is in a json format

Parameters **event** – json of event

Returns dictionary with the info

hockey_scraper.json_pbp.**parse_json**(*game_json*, *game_id*)
Scrape the json for a game

Parameters

- **game_json** – raw json
- **game_id** – game id for game

Returns Either a DataFrame with info for the game

hockey_scraper.json_pbp.**scrape_game**(*game_id*)
Used for debugging. HTML depends on json so can't follow this structure

Parameters **game_id** – game to scrape

Returns DataFrame of game info

Espn PBP

This module contains code to scrape coordinates for games off of espn for any given game

hockey_scraper.espn_pbp.**event_type**(*play_description*)
Returns the event type (ex: a SHOT or a GOAL...etc) given the event description

Parameters **play_description** – description of play

Returns event

hockey_scraper.espn_pbp.**get_espn**(*date*, *home_team*, *away_team*)
Gets the ESPN pbp feed Ex: http://www.espn.com/nhl/gamecast/data/masterFeed?lang=en&isAll=true&gameId=400885300

Parameters

- **date** – date of the game
- **home_team** – home team
- **away_team** – away team

Returns raw xml

hockey_scraper.espn_pbp.**get_espn_game_id**(*date*, *home_team*, *away_team*)
Scrapes the day's schedule and gets the id for the given game Ex: http://www.espn.com/nhl/scoreboard?date=20161024

Parameters

- **date** – format-> YearMonthDay-> 20161024
- **home_team** – home team
- **away_team** – away team

Returns 9 digit game id

hockey_scraper.espn_pbp.**get_game_ids**(*response*)
Get game_ids for date from doc

Parameters **response** – doc

Returns list of game_ids

hockey_scraper.espn_pbp.**get_teams**(*response*)
Extract Teams for date from doc

Parameters **response** – doc

Returns list of teams

hockey_scraper.espn_pbp.**parse_espn**(*espn_xml*)
Parse feed

Parameters **espn_xml** – raw xml of feed

Returns DataFrame with info

hockey_scraper.espn_pbp.**parse_event**(*event*)
Parse each event. In the string each field is separated by a '~'. Relevant for here: The first two are the x and y coordinates. And the 4th and 5th are the time elapsed and period.

Parameters **event** – string with info

Returns return dict with relevant info

hockey_scraper.espn_pbp.**scrape_game**(*date*, *home_team*, *away_team*)
Scrape the game

Parameters

- **date** – ex: 2016-20-24
- **home_team** – tricode
- **away_team** – tricode

Returns DataFrame with info

Json Shifts

This module contains functions to scrape the Json toi/shifts for any given game

hockey_scraper.json_shifts.**fix_team_tricode**(*tricode*)
Some of the tricodes are different than how I want them

Parameters **tricode** – 3 letter team name - ex: NYR

Returns fixed tricode

hockey_scraper.json_shifts.**get_shifts**(*game_id*)
Given a game_id it returns the raw json Ex: http://www.nhl.com/stats/rest/shiftcharts?cayenneExp=gameId=2010020001

Parameters **game_id** – the game

Returns json or None

hockey_scraper.json_shifts.**parse_json**(*shift_json*, *game_id*)
Parse the json

Parameters

- **shift_json** – raw json
- **game_id** – if of game

Returns DataFrame with info

hockey_scraper.json_shifts.**parse_shift**(*shift*)
Parse shift for json

Parameters **shift** – json for shift

Returns dict with shift info

hockey_scraper.json_shifts.**scrape_game**(*game_id*)
Scrape the game.

Parameters **game_id** – game

Returns DataFrame with info for the game

Html Shifts

This module contains functions to scrape the Html Toi Tables (or shifts) for any given game

hockey_scraper.html_shifts.**analyze_shifts**(*shift*, *name*, *team*, *home_team*, *player_ids*)
Analyze shifts for each player when using. Prior to this each player (in a dictionary) has a list with each entry being a shift.

Parameters

- **shift** – info on shift
- **name** – player name
- **team** – given team
- **home_team** – home team for given game
- **player_ids** – dict with info on players

Returns dict with info for shift

hockey_scraper.html_shifts.**get_shifts**(*game_id*)
Given a game_id it returns a DataFrame with the shifts for both teams Ex: http://www.nhl.com/scores/htmlreports/20162017/TV020971.HTM

Parameters **game_id** – the game

Returns DataFrame with all shifts or None

hockey_scraper.html_shifts.**get_soup**(*shifts_html*)
Uses Beautiful soup to parses the html document. Some parsers work for some pages but don't work for others....I'm not sure why so I just try them all here in order

Parameters **shifts_html** – html doc

Returns "soupified" html and player_shifts portion of html (it's a bunch of td tags)

hockey_scraper.html_shifts.**get_teams**(*soup*)
Return the team for the TOI tables and the home team

Parameters **soup** – souped up html

Returns list with team and home team

hockey_scraper.html_shifts.**parse_html**(*html*, *player_ids*, *game_id*)
Parse the html

Parameters

- **html** – cleaned up html
- **player_ids** – dict of home and away players
- **game_id** – id for game

Returns DataFrame with info

hockey_scraper.html_shifts.**scrape_game**(*game_id*, *players*)
Scrape the game.

Parameters

- **game_id** – id for game
- **players** – list of players

Returns DataFrame with info for the game

Schedule

This module contains functions to scrape the json schedule for any games or date range

hockey_scraper.json_schedule.**get_current_season**()
Get Season based on today's date

Returns season -> ex: 2016 for 2016-2017 season

hockey_scraper.json_schedule.**get_dates**(*games*)
Given a list game_ids it returns the dates for each game

Parameters **games** – list with game_id's ex: 2016020001

Returns list with game_id and corresponding date for all games

hockey_scraper.json_schedule.**get_schedule**(*date_from*, *date_to*)
Scrapes games in date range Ex: https://statsapi.web.nhl.com/api/v1/schedule?startDate=2010-10-03&endDate=2011-06-20

Parameters

- **date_from** – scrape from this date
- **date_to** – scrape until this date

Returns raw json of schedule of date range

hockey_scraper.json_schedule.**scrape_schedule**(*date_from*, *date_to*, *preseason=False*)
Calls getSchedule and scrapes the raw schedule JSON

Parameters

- **date_from** – scrape from this date
- **date_to** – scrape until this date
- **preseason** – Boolean indicating whether include preseason games (default if False)

Returns list with all the game id's

Playing Roster

This module contains functions to scrape the Html game roster for any given game

hockey_scraper.playing_roster.**fix_name**(*player*)
Get rid of (A) or (C) when a player has it attached to their name

Parameters **player** – list of player info -> [number, position, name]

Returns fixed list

hockey_scraper.playing_roster.**get_coaches**(*soup*)
scrape head coaches

Parameters **soup** – html

Returns dict of coaches for game

hockey_scraper.playing_roster.**get_content**(*roster*)
Uses Beautiful soup to parses the html document. Some parsers work for some pages but don't work for others....I'm not sure why so I just try them all here in order

Parameters **roster** – doc

Returns players and coaches

hockey_scraper.playing_roster.**get_players**(*soup*)
scrape roster for players

Parameters **soup** – html

Returns dict for home and away players

hockey_scraper.playing_roster.**get_roster**(*game_id*)
Given a game_id it returns the raw html Ex: http://www.nhl.com/scores/htmlreports/20162017/RO020475.HTM

Parameters **game_id** – the game

Returns raw html of game

hockey_scraper.playing_roster.**scrape_roster**(*game_id*)
For a given game scrapes the roster

Parameters **game_id** – id for game

Returns dict of players (home and away) an dict for both head coaches

Shared Functions

This file is a bunch of the shared functions or just general stuff used by the different scrapers in the package.

hockey_scraper.shared.**convert_to_seconds**(*minutes*)
Return minutes remaining in time format to seconds elapsed

Parameters **minutes** – time remaining

Returns time elapsed in seconds

hockey_scraper.shared.**fix_name**(*name*)
Check if a name falls under those that need fixing. If it does...fix it.

Parameters **name** – name in pbp

Returns Either the given parameter or the fixed name

hockey_scraper.shared.**get_url**(*url*)
Get the url

Parameters **url** – given url

Returns page

License

CHAPTER 2

Purpose

This package is designed to allow people to scrape the Play by Play and Shift data off of the National Hockey League (NHL) API and website for all preseason, regular season, and playoff games since the 2007-2008 season.

CHAPTER 3

Prerequisites

You are going to need to have python installed for this. This should work for both python 2.7 and 3 (I recommend having from at least version 3.6.0 but earlier versions should be fine).

If you don't have python installed on your machine, I'd recommend installing it through the anaconda distribution. Anaconda comes with a bunch of libraries pre-installed so it's easier to start off.

CHAPTER 4

Installation

To install all you need to do is open up your terminal and type in:

```
pip install hockey_scraper
```

CHAPTER 5

Usage

Scrape data on a season by season level:

```
import hockey_scraper

# Scrapes the 2015 & 2016 season with shifts and stores the data in a Csv file
hockey_scraper.scrape_seasons([2015, 2016], True)

# Scrapes the 2008 season without shifts and returns a json string of the data
scraped_data = hockey_scraper.scrape_seasons([2008], False, data_format='Json')
```

Scrape a list of games:

```
import hockey_scraper

# Scrapes the first game of 2014, 2015, and 2016 seasons with shifts and stores the␣
↪data in a Csv file
hockey_scraper.scrape_games([2014020001, 2015020001, 2016020001], True)

# Scrapes the first game of 2007, 2008, and 2009 seasons with shifts and returns a␣
↪Json string of the data
scraped_data = hockey_scraper.scrape_games([2007020001, 2008020001, 2009020001], True,
↪ data_format='Json')
```

Scrape all games in a given date range:

```
import hockey_scraper

# Scrapes all games between 2016-10-10 and 2016-10-20 without shifts and stores the␣
↪data in a Csv file
hockey_scraper.scrape_date_range('2016-10-10', '2016-10-20', False)

# Scrapes all games between 2015-1-1 and 2015-1-15 without shifts and returns a Json␣
↪string of the data
scraped_data = hockey_scraper.scrape_date_range('2015-1-1', '2015-1-15', False, data_
↪format='Json')
```

The full documentation can be found here.

CHAPTER 6

Contact

Please contact me for any issues or suggestions. For any bugs or anything related to the code please open an issue. Otherwise you can email me at Harryshomer@gmail.com.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

H

I

P

R

S

T