Social Media Analysis Twitter and Instagram

Release 1

Ali Adelkhah

Sep 22, 2023

CONTENTS:

CHAPTER

ONE

CONDAINSTAGRAMANALYS

1.1 Betweenness_pageRank_Degree module

Betweenness_pageRank_Degree.**activate_to_json**(*A*, *data*)

Betweenness_pageRank_Degree.**convert_igraph_to_networkx**(*igraph_graph*)

Convert an igraph graph to a networkx graph.

Args:
igraph_graph (igraph.Graph): The igraph graph to be converted.

Returns:
networkx.DiGraph: The converted networkx graph.

Betweenness_pageRank_Degree.**create_igraph**(*source*, *target*, *vertic*)

Create a directed graph using the igraph library.

Args:
source (list): A list of source vertices. target (list): A list of target vertices. vertic (list): A list of all vertices in the graph.

Returns:
Graph: A directed graph object created using the igraph library.

Betweenness_pageRank_Degree.**finding_influencer_betweeness**(*g*, *prop*, *numberOfinfluencer*, *mc=10*)

Find the top influencers using the betweenness centrality measure.

Args:
g (igraph.Graph): The input graph. prop (str): The property to be influenced. numberOfinfluencer (int): The number of influencers to find. mc (int, optional): The number of Monte Carlo simulations. Defaults to 10.

Returns:
tuple: A tuple containing the top influencers, their spread, and the time taken.

Betweenness_pageRank_Degree.**finding_influencer_cluster_pagerank**(*g*, *prop*, *numberOfinfluencer*, *mc=10*)

Find the top influencers using the PageRank algorithm within clusters.

Args:
g (igraph.Graph): The input graph. prop (str): The property to be influenced. numberOfinfluencer (int): The number of influencers to find. mc (int, optional): The number of Monte Carlo simulations. Defaults to 10.

Returns:
tuple: A tuple containing the top influencers, their spread, and the time taken.

Betweenness_pageRank_Degree.**finding_influencer_degree**(*g*, *prop*, *numberOfinfluencer*, *mc=10*)

Find the top influencers using the degree centrality measure.

Args:

g (igraph.Graph): The input graph. prop (str): The property to be influenced. numberOfinfluencer (int): The number of influencers to find. mc (int, optional): The number of Monte Carlo simulations. Defaults to 10.

Returns:

tuple: A tuple containing the top influencers, their spread, and the time taken.

Betweenness_pageRank_Degree.**finding_influencer_pagerank**(*g*, *prop*, *numberOfinfluencer*, *mc=10*)

Find the top influencers using the PageRank algorithm.

Args:

g (igraph.Graph): The input graph. prop (str): The property to be influenced. numberOfinfluencer (int): The number of influencers to find. mc (int, optional): The number of Monte Carlo simulations. Defaults to 10.

Returns:

tuple: A tuple containing the top influencers, their spread, and the time taken.

Betweenness_pageRank_Degree.**independent_cascade**(*g*, *S*, *p*)

Simulate the independent cascade model on a graph.

Args:

g (igraph.Graph): The input graph. S (list): The initial set of activated nodes. p (float): The probability of activation.

Returns:

list: The set of activated nodes after the propagation process.

Betweenness_pageRank_Degree.**read_json_file**(*filename*)

Betweenness_pageRank_Degree.**top_n_keys**(*d*, *n*)

Return the keys with the n highest values in dictionary d.

1.2 community_detection module

community_detection.**community_detection**(*input_txt_file*, *input_json_file*, *output_json_file*)

Detect communities in a graph using the Louvain method.

Args:

input_txt_file (str): The path to the input text file. input_json_file (str): The path to the input JSON file. output_json_file (str): The path to the output JSON file.

Returns:

None

1.3 greedy_celf module

greedy_celf.**IC**(*g*, *S*, *p=0.1*, *mc=1000*)

This function simulates the Independent Cascade (IC) model of information propagation on a given graph for a specified number of Monte-Carlo simulations.

Parameters: g (igraph.Graph): The graph object on which the propagation process is simulated. S (list): The set of seed nodes from which the propagation starts. p (float, optional): The propagation probability. Default is 0.1. mc (int, optional): The number of Monte-Carlo simulations to run. Default is 1000.

Returns: float: The average number of nodes influenced by the seed nodes over all simulations.

greedy_celf.**celf**(*g*, *k*, *p=0.1*, *mc=1000*)

This function implements the Cost-Effective Lazy Forward (CELF) algorithm for influence maximization. It finds the optimal seed set and measures the resulting spread and time for each iteration.

Parameters: g (igraph.Graph): The graph object on which the propagation process is simulated. k (int): The number of seed nodes to be found. p (float, optional): The propagation probability. Default is 0.1. mc (int, optional): The number of Monte-Carlo simulations to run. Default is 1000.

Returns: tuple: A tuple containing the following elements:

S (list): The optimal seed set. SPREAD (list): The resulting spread for each iteration. timelapse (list): The time elapsed for each iteration. LOOKUPS (list): The number of times the spread is computed for each iteration.

greedy_celf.**create_igraph**(*source*, *target*, *vertic*)

greedy_celf.**greedy**(*g*, *k*, *p=0.1*, *mc=1000*)

This function implements the greedy algorithm for influence maximization. It finds the optimal seed set and measures the resulting spread and time for each iteration.

Parameters: g (igraph.Graph): The graph object on which the propagation process is simulated. k (int): The number of seed nodes to be found. p (float, optional): The propagation probability. Default is 0.1. mc (int, optional): The number of Monte-Carlo simulations to run. Default is 1000.

Returns: tuple: A tuple containing the following elements:

S (list): The optimal seed set. spread (list): The resulting spread for each iteration. timelapse (list): The time elapsed for each iteration.

1.4 helper_functions module

helper_functions.**add_cluster_to_json**(*input_dict*, *cluster_dict*)

This function adds a cluster group to each node in the input dictionary based on a provided cluster dictionary.

Parameters: input_dict (dict): A dictionary containing 'nodes' and 'links'. Each node is a dictionary with at least a 'name' key. cluster_dict (dict): A dictionary where the keys are node names and the values are the corresponding cluster groups.

Returns: dict: The input dictionary updated with the 'group' key for each node, representing its cluster group.

helper_functions.**centrality_to_str_arr**(*centrality*)

This function converts a list of tuples representing node centrality into a list of strings.

Parameters: centrality (list): A list of tuples where each tuple contains a node name and its centrality value.

Returns: list: A list of strings where each string represents a node and its centrality value in the format 'node | centrality'.

helper_functions.**create_undirected_graph_from_txt**(*input_txt_file*)

This function creates an undirected graph from a text file. Each line in the text file represents an edge between two nodes.

Parameters: input_txt_file (str): The path to the input text file. Each line in the file should contain

two node names separated by a space, representing an edge between them.

Returns: nx.Graph: An undirected graph created from the edges specified in the input text file.

helper_functions.**sort_and_small_dict**(*d*, *n*)

This function sorts a dictionary in descending order based on its values and returns the first 'n' items.

Parameters: d (dict): The dictionary to be sorted. n (int): The number of items to return after sorting.

Returns: list: A list of tuples representing the first 'n' items from the sorted dictionary.

1.5 instagramscraper module

instagramscraper.**clean_text**(*text_list*)

This function cleans a list of text strings by removing all non-alphanumeric characters, converting to lowercase, and removing extra whitespace.

Parameters: text_list (list): A list of text strings to be cleaned.

Returns: list: A list of cleaned text strings.

instagramscraper.**count_jpg_files**(*name*)

This function counts the number of '.jpg' files in a specific directory.

Parameters: name (str): The name of the profile. The function will look in a directory named after the profile under 'instagramuser/images'.

Returns: int: The count of '.jpg' files in the specified directory.

instagramscraper.**download_images**(*driver*, *name*, *imagenumber*)

This function downloads a specified number of images for a specific Instagram user.

Parameters: driver: The webdriver instance to interact with the web page. name (str): The name of the Instagram user. The function will look for images in a directory named after the profile under 'instagramuser/images'. imagenumber (int): The number of images to download.

Note: The function will create a directory 'instagramuser/{name}/images' if it does not exist and download the images into this directory. Each image is saved as 'image-{number}.jpg', where {number} is the order of the image.

instagramscraper.**find_profile_detail**(*driver*, *name*)

This function finds the profile details of a specific Instagram user and saves them to a text file. It also downloads the profile image.

Parameters: driver: The webdriver instance to interact with the web page. name (str): The name of the Instagram user. The function will look for profile details in a directory named after the profile under 'instagramuser'.

Returns: list: A list of profile details read from the web page.

instagramscraper.**read_alluser_fromFile**()

This function reads a list of all Instagram users from a text file.

Returns: list: A list of Instagram user names read from the file. Each user name is a line from the file with the trailing linebreak removed.

instagramscraper.**read_comments**(*name*)

This function reads the comments of a given Instagram user's posts from a file.

Parameters: name (str): The username of the Instagram account.

Returns: list: A list of comments from the user's posts.

instagramscraper.**read_follower_fromFile**(*name*)

This function reads a list of followers from a text file for a specific Instagram user.

Parameters: name (str): The name of the Instagram user. The function will look for a file named 'follower.txt' in a directory named after the profile under 'instagramuser'.

Returns: list: A list of follower names read from the file. Each follower name is a line from the file with the trailing linebreak removed.

instagramscraper.**read_following_fromFile**(*name*)

This function reads a list of followings from a text file for a specific Instagram user.

Parameters: name (str): The name of the Instagram user. The function will look for a file named 'following.txt' in a directory named after the profile under 'instagramuser'.

Returns: list: A list of following names read from the file. Each following name is a line from the file with the trailing linebreak removed.

instagramscraper.**read_profile**(*name*)

This function reads a profile from a text file and returns a list of names.

Parameters: name (str): The name of the profile. The function will look for a file named 'profile.txt' in a directory named after the profile under 'instagramuser'.

Returns: list: A list of names read from the file. Each name is a line from the file with the trailing linebreak removed.

instagramscraper.**read_tags**(*name*)

This function reads the tags of a given Instagram user from a file.

Parameters: name (str): The username of the Instagram account.

Returns: list: A list of tags from the user's posts.

instagramscraper.**remove_html_tags_word**(*text*)

This function removes all HTML tags from a text string.

Parameters: text (str): The text string from which to remove HTML tags.

Returns: str: The text string without any HTML tags.

instagramscraper.**scrape_followers**(*driver*, *username*, *user_input*)

This function scrapes the followers of a given Instagram user.

Parameters: driver (webdriver): The webdriver instance to interact with the web page. username (str): The username of the Instagram account to scrape followers from. user_input (int): The number of followers to scrape.

Returns: list: A list of usernames of the followers.

instagramscraper.**scrape_followings**(*driver*, *username*, *user_input*)

This function scrapes the followings of a given Instagram user.

Parameters: driver (webdriver): The webdriver instance to interact with the web page. username (str): The username of the Instagram account to scrape followings from. user_input (int): The number of followings to scrape.

Returns: list: A list of usernames of the followings.

instagramscraper.**scrape_instagram**(*driver*, *username*, *imageinput*, *ferinput*, *finginput*)

This function scrapes an Instagram profile and downloads images from the profile. It also scrapes the followers and followings of the profile.

Parameters: driver (webdriver object): The webdriver object to interact with the Instagram page. username (str): The username of the Instagram profile to scrape. imageinput (int): The number of images to download from the profile. ferinput (int): The number of followers to scrape from the profile. finginput (int): The number of followings to scrape from the profile.

Returns: None: This function does not return anything but writes the scraped data to files.

instagramscraper.**scrape_tags**(*driver*, *username*)

This function scrapes the tags and comments of a given Instagram user's posts.

Parameters: driver (webdriver): The webdriver instance to interact with the web page. username (str): The username of the Instagram account to scrape tags and comments from.

Returns: tuple: A tuple containing two lists -

1. A list of tags from the user's posts.
2. A list of comments from the user's posts.

instagramscraper.**write_alluser_toFile**(*lst*)

This function writes a list of Instagram users to a text file.

Parameters: lst (list): A list of Instagram user names.

Note: The function will create a directory 'instagramuser' if it does not exist and write the user names to a file named 'ALLinstagramUSERS.txt' in this directory. Each user name is written on a new line.

instagramscraper.**write_comments**(*lst*, *name*)

This function writes the comments of a given Instagram user's posts to a file.

Parameters: lst (list): A list of comments. name (str): The username of the Instagram account.

Returns: None

instagramscraper.**write_follower_toFile**(*lst*, *name*)

This function writes a list of followers to a text file for a specific Instagram user.

Parameters: lst (list): A list of follower names. name (str): The name of the Instagram user.

Note: The function will create a directory 'instagramuser/{name}' if it does not exist and write the follower names to a file named 'follower.txt' in this directory. Each follower name is written on a new line.

instagramscraper.**write_following_toFile**(*lst*, *name*)

This function writes a list of followings to a text file for a specific Instagram user.

Parameters: lst (list): A list of following names. name (str): The name of the Instagram user.

Note: The function will create a directory 'instagramuser/{name}' if it does not exist and write the following names to a file named 'following.txt' in this directory. Each following name is written on a new line.

instagramscraper.**write_tag**(*lst*, *name*)

This function writes the tags of a given Instagram user's posts to a file.

Parameters: lst (list): A list of tags. name (str): The username of the Instagram account.

Returns: None

1.6 mainInstagram module

async mainInstagram.**form_submit**(*request: Request*, *name: str = Form(PydanticUndefined)*, *gender: str = Form(PydanticUndefined)*, *age: int = Form(PydanticUndefined)*)

Currently not used recive 'name' , 'gender' , 'age' from the HTML and return JSON

async mainInstagram.**homepage**(*request: Request*)

Currently not used HTML form with fields 'name' , 'gender' , 'age' and with pressing the button POST them

async mainInstagram.**read_item**(*request: Request*)

will show network graph

mainInstagram.**read_json_file**(*filename*)

1.7 mainTwitter module

async mainTwitter.**form_submit**(*request: Request*, *name: str = Form(PydanticUndefined)*, *gender: str = Form(PydanticUndefined)*, *age: int = Form(PydanticUndefined)*)

Currently not used recive 'name' , 'gender' , 'age' from the HTML and return JSON

async mainTwitter.**homepage**(*request: Request*)

Currently not used HTML form with fields 'name' , 'gender' , 'age' and with pressing the button POST them

async mainTwitter.**read_item**(*request: Request*)

will show trending tags and thier sentiment score

mainTwitter.**read_json_file**(*filename*)

1.8 relation_to_json module

relation_to_json.**allrealation_toText_Instagram**(*file_name*)

This function reads the followers and followings of all instagram users from files, creates a list of all relations, and writes them to a text file.

Parameters: file_name (str): The name of the output text file. Each line in the file will contain

two usernames separated by a space, representing a relation between them.

Returns: None: This function does not return anything but writes the relations to a text file.

relation_to_json.**allrealation_toText_Twitter**(*file_name*)

This function reads the followers and followings of all twitter users from files, creates a list of all relations, and writes them to a text file.

Parameters: file_name (str): The name of the output text file. Each line in the file will contain

two usernames separated by a space, representing a relation between them.

Returns: None: This function does not return anything but writes the relations to a text file.

relation_to_json.**relations_to_json**(*input_txt_file*, *output_json_file*)

Converts a text file describing relations between entities into a JSON file.

The input text file should contain lines where each line describes a relation between two entities. Each line should contain two strings separated by a space, representing the two entities.

The output JSON file contains a list of nodes and a list of links. Each node is represented as a dictionary with "id", "name", and "group" as keys. Each link is represented as a dictionary with "id", "source", "target", "value", and "bi_directional" as keys. The 'source' and 'target' in each link are ids of the two entities.

If a connection exists in both directions (entity A to entity B and entity B to entity A), the "bi_directional" key in the corresponding link will be True. Otherwise, it will be False.

1.8.1 Parameters

input_txt_file
[str] The path to the input text file.

output_json_file
[str] The path to the output JSON file.

1.8.2 Returns

tuple
A tuple containing a dictionary mapping each entity's name to its id and a set of all edges as tuples.

1.8.3 Raises

IOError
If the input file could not be read or the output file could not be written.

1.9 similarity module

similarity.**common_instaneighbor_two_user**(*u1*, *u2*)

This function finds the common Instagram followers between two users.

Parameters: u1 (str): The username of the first user. u2 (str): The username of the second user.

Returns: int: The number of common followers between the two users.

similarity.**common_twitterneighbor_two_user**(*u1*, *u2*)

This function finds the common Twitter followers between two users.

Parameters: u1 (str): The username of the first user. u2 (str): The username of the second user.

Returns: int: The number of common followers between the two users.

similarity.**cos_similarity**(*a*, *b*)

This function calculates the cosine similarity between two vectors.

Parameters: a (np.array): The first vector. b (np.array): The second vector.

Returns: float: The cosine similarity between the two vectors.

similarity.**count_jpg_files**(*name*)

This function counts the number of .jpg files in a specific directory.

Parameters: name (str): The name of the directory to search in.

Returns: int: The number of .jpg files in the directory.

similarity.**english_text_similarity**(*text1*, *text2*)

This function calculates the text similarity between two English texts.

Parameters: text1 (str): The first text. text2 (str): The second text.

Returns: float: The text similarity between the two texts.

similarity.**find_most_similar_instagram_users**(*numberOfuser*, *minimumNeighbor*)

This function finds the most similar Instagram users based on their followers.

Parameters: numberOfuser (int): The number of users to find. minimumNeighbor (int): The minimum number of common followers between two users to consider them similar.

Returns: None: The function writes the results to a JSON file and does not return anything.

similarity.**find_most_similar_twitter_users**(*numberOfuser*, *minimumNeighbor*)

This function finds the most similar Twitter users based on their followers.

Parameters: numberOfuser (int): The number of users to find. minimumNeighbor (int): The minimum number of common followers between two users to consider them similar.

Returns: None: The function writes the results to a JSON file and does not return anything.

similarity.**image_similarity_two_user**(*u1*, *u2*)

This function calculates the average image similarity between two users' images.

Parameters: u1 (str): The username of the first user. u2 (str): The username of the second user.

Returns: float: The average image similarity between the two users' images.

similarity.**post_similarity**(*u1*, *u2*)

This function calculates the average text similarity between the posts of two users.

Parameters: u1 (str): The username of the first user. u2 (str): The username of the second user.

Returns: float: The average text similarity between the posts of the two users. If the languages of the posts do not match or are not supported, it ignores those pairs of posts in the calculation.

similarity.**read_comments**(*name*)

similarity.**text_similarity**(*text1*, *text2*)

This function calculates the text similarity between two texts. It supports both English and Farsi texts.

Parameters: text1 (str): The first text. text2 (str): The second text.

Returns: float: The text similarity between the two texts. If the languages of the texts do not match or are not supported, it returns -10.

similarity.**tweet_similarity**(*u1*, *u2*)

This function calculates the average text similarity between the tweets of two users.

Parameters: u1 (str): The username of the first user. u2 (str): The username of the second user.

Returns: float: The average text similarity between the tweets of the two users. If the languages of the tweets do not match or are not supported, it ignores those pairs of tweets in the calculation.

1.10 tagAnalysis module

tagAnalysis.**find_trend**(*tagnumber=10*, *wordnumber=5*)

This function finds the trending hashtags and their associated words and phrases.

Parameters: tagnumber (int): The number of top trending hashtags to find. Default is 10. wordnumber (int): The number of top words and phrases to find for each hashtag. Default is 5.

Returns: tuple: A tuple containing three dictionaries -

1. A dictionary where the keys are hashtags and the values are their counts.
2. A dictionary where the keys are hashtags and the values are lists of top words associated with each hashtag.
3. A dictionary where the keys are hashtags and the values are lists of top two-word phrases associated with each hashtag.

tagAnalysis.**most_frequent_two_word_phrases**(*texts*)

This function finds the most frequent two-word phrases in a list of texts.

Parameters: texts (list): A list of text strings to analyze.

Returns: dict: A dictionary where the keys are hashtags and the values are lists of tuples. Each tuple contains a two-word phrase and its frequency.

tagAnalysis.**most_frequent_words**(*texts*)

This function finds the most frequent words in a list of texts.

Parameters: texts (list): A list of text strings to analyze.

Returns: dict: A dictionary where the keys are hashtags and the values are lists of tuples. Each tuple contains a word and its frequency.

tagAnalysis.**read_tags_insta**(*name*)

tagAnalysis.**remove_short_words**(*s*, *min_length=4*)

This function removes words from a string that are shorter than a specified length.

Parameters: s (str): The string from which to remove short words. min_length (int): The minimum length of words to keep. Default is 4.

Returns: str: The string with short words removed.

tagAnalysis.**sentiment_analys**(*text*)

This function performs sentiment analysis on a given text. It detects the language of the text and calls the appropriate sentiment analysis function based on the language.

Parameters: text (str): The text to perform sentiment analysis on.

Returns: str: The sentiment of the text. It can be ‘joy’, ‘sad’, ‘hate’, ‘love’, ‘angry’, ‘fear’, or ‘none’ if the language is not supported.

tagAnalysis.**sentiment_analys_english**(*text*)

This function performs sentiment analysis on a given English text.

Parameters: text (str): The English text to perform sentiment analysis on.

Returns: str: The sentiment of the text. It can be 'joy', 'sad', 'hate', 'love', 'angry', or 'fear'.

tagAnalysis.**sentiment_analys_farsi**(*text*)

This function performs sentiment analysis on a given Farsi text.

Parameters: text (str): The Farsi text to perform sentiment analysis on.

Returns: str: The sentiment of the text. It can be 'joy', 'sad', 'hate', 'love', 'angry', or 'fear'.

tagAnalysis.**top_n_keys**(*d*, *n*)

This function finds the top N keys in a dictionary based on their values.

Parameters: d (dict): The dictionary to analyze. n (int): The number of keys to return.

Returns: list: A list of the top N keys in the dictionary.

tagAnalysis.**trend_tag_sentiment**(*tagnumber=10*)

This function finds the sentiment of the tweets associated with the trending hashtags.

Parameters: tagnumber (int): The number of top trending hashtags to analyze. Default is 10.

Returns: dict: A dictionary where the keys are hashtags and the values are dictionaries. Each inner dictionary contains the sentiment counts for the associated hashtag.

1.11 taggraph module

taggraph.**build_tag_graph**(*file_name*)

This function builds a graph of tags from all users' comments and writes it to a file.

Parameters: file_name (str): The name of the file to write the tag graph to.

Returns: None: The function writes the tag graph to a file and does not return anything.

taggraph.**count_each_tag**()

This function counts the occurrence of each tag in all users' comments.

Returns: dict: A dictionary where the keys are tags and the values are their counts.

taggraph.**read_json_file**(*filename*)

taggraph.**read_tags_insta**(*name*)

This function reads the tags of a given Instagram user from a file.

Parameters: name (str): The username of the Instagram account.

Returns: list: A list of tags from the user's posts.

taggraph.**tags_relation_tojason**(*counttag*, *tag_edges*, *tag_graph_json*)

This function builds a JSON object representing the relations between tags and writes it to a file.

Parameters: counttag (dict): A dictionary where the keys are tags and the values are their counts. tag_edges (str): The name of the file containing the tag edges. tag_graph_json (str): The name of the file to write the JSON object to.

Returns: None: The function writes the JSON object to a file and does not return anything.

1.12 twitterscrape module

twitterscrape.**clean_text**(*text_list*)

This function cleans a list of text strings by removing all non-alphanumeric characters, converting to lowercase, and removing extra whitespace.

Parameters: text_list (list): A list of text strings to be cleaned.

Returns: list: A list of cleaned text strings.

twitterscrape.**find_ID**(*text*)

This function finds all Twitter usernames in a text string.

Parameters: text (str): The text string to search for Twitter usernames.

Returns: list: A list of Twitter usernames found in the text string.

twitterscrape.**find_ID_list**(*l*)

This function finds all Twitter usernames in a list of text strings.

Parameters: l (list): A list of text strings to search for Twitter usernames.

Returns: list: A list of Twitter usernames found in the text strings.

twitterscrape.**find_hashtags**(*text*)

This function finds all hashtags in a text string.

Parameters: text (str): The text string to search for hashtags.

Returns: list: A list of hashtags found in the text string.

twitterscrape.**find_hashtags_list**(*l*)

This function finds all hashtags in a list of text strings.

Parameters: l (list): A list of text strings to search for hashtags.

Returns: list: A list of hashtags found in the text strings.

twitterscrape.**read_alluser_fromFile**()

This function reads a list of all Twitter users from a file.

Returns: list: A list of all Twitter usernames.

twitterscrape.**read_comments**(*name*)

This function reads the comments of a given Twitter user's posts from a file.

Parameters: name (str): The username of the Twitter account.

Returns: list: A list of comments from the user's posts.

twitterscrape.**read_follower_fromFile**(*name*)

This function reads a list of a Twitter user's followers from a file.

Parameters: name (str): The username of the Twitter account.

Returns: list: A list of the user's followers' usernames.

twitterscrape.**read_following_fromFile**(*name*)

This function reads a list of a Twitter user's followings from a file.

Parameters: name (str): The username of the Twitter account.

Returns: list: A list of the user's followings' usernames.

twitterscrape.**read_tags**(*name*)

This function reads the tags of a given Twitter user from a file.

Parameters: name (str): The username of the Twitter account.

Returns: list: A list of tags from the user's posts.

twitterscrape.**remove_hashtag**(*text*)

This function removes the hashtag symbol from a text string.

Parameters: text (str): The text string from which to remove the hashtag symbol.

Returns: str: The text string without the hashtag symbol.

twitterscrape.**remove_html_tags_word**(*text*)

This function removes all HTML tags from a text string.

Parameters: text (str): The text string from which to remove HTML tags.

Returns: str: The text string without any HTML tags.

twitterscrape.**remove_newlines**(*text*)

This function removes all newline characters from a text string.

Parameters: text (str): The text string from which to remove newline characters.

Returns: str: The text string without any newline characters.

twitterscrape.**scrape_followers**(*driver*, *username*, *user_input*)

This function scrapes the followers of a given Twitter user.

Parameters: driver (webdriver): The webdriver instance to interact with the web page. username (str): The username of the Twitter account to scrape followers from. user_input (int): The number of followers to scrape.

Returns: list: A list of usernames of the followers.

twitterscrape.**scrape_followings**(*driver*, *username*, *user_input*)

This function scrapes the followings of a given Twitter user.

Parameters: driver (webdriver): The webdriver instance to interact with the web page. username (str): The username of the Twitter account to scrape followings from. user_input (int): The number of followings to scrape.

Returns: list: A list of usernames of the followings.

twitterscrape.**scrape_tags**(*driver*, *username*)

This function scrapes the tags and comments of a given Twitter user's posts.

Parameters: driver (webdriver): The webdriver instance to interact with the web page. username (str): The username of the Twitter account to scrape tags and comments from.

Returns: None: The function writes the tags and comments to files and does not return anything.

twitterscrape.**scrape_twitter**(*driver*, *username*, *ferinput*, *finginput*)

This function scrapes an twitter profile with their tweets and tags. It also scrapes the followers and followings of the profile.

Parameters: driver (webdriver object): The webdriver object to interact with the Instagram page. username (str): The username of the Instagram profile to scrape. imageinput (int): The number of images to download from the profile. ferinput (int): The number of followers to scrape from the profile. finginput (int): The number of followings to scrape from the profile.

Returns: None: This function does not return anything but writes the scraped data to files.

twitterscrape.**write_alluser_toFile**(*lst*)

This function writes a list of all Twitter users to a file.

Parameters: lst (list): A list of Twitter usernames.

Returns: None: The function writes the list to a file and does not return anything.

twitterscrape.**write_comments**(*lst*, *name*)

This function writes the comments of a given Twitter user's posts to a file.

Parameters: lst (list): A list of comments. name (str): The username of the Twitter account.

Returns: None: The function writes the list to a file and does not return anything.

twitterscrape.**write_follower_toFile**(*lst*, *name*)

This function writes a list of a Twitter user's followers to a file.

Parameters: lst (list): A list of the user's followers' usernames. name (str): The username of the Twitter account.

Returns: None: The function writes the list to a file and does not return anything.

twitterscrape.**write_following_toFile**(*lst*, *name*)

This function writes a list of a Twitter user's followings to a file.

Parameters: lst (list): A list of the user's followings' usernames. name (str): The username of the Twitter account.

Returns: None: The function writes the list to a file and does not return anything.

twitterscrape.**write_tag**(*lst*, *name*)

This function writes the tags of a given Twitter user's posts to a file.

Parameters: lst (list): A list of tags. name (str): The username of the Twitter account.

Returns: None: The function writes the list to a file and does not return anything.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX