Design & Implementierung eines Echtzeit-Q&A-Systems als Erweiterung des IAmA-Subreddits

-

Python Code Documentation

Benedikt Hierl
Version 1.0
Sun Mar 20 2016

Table of Contents

Namespace Index

Packages

Here are the packages with brief descriptions (if available):

File Index

File List

Here is a list of all files with brief descriptions:

Namespace Documentation

analyze_correlation_upvote_reaction_time_pieChart Namespace Reference

Functions

- def initialize_mongo_db_parameters ()
- def check_script_arguments ()
- def calculate_time_difference (comment_time_stamp, answer_time_stamp_iama_host)
- def check_if_comment_is_answer_from_thread_author (author_of_thread, comment_acutal_id, comments_cursor)
- def check_if_comment_is_not_from_thread_author (author_of_thread, comment_author)
- def check_if_comment_is_on_tier_1 (comment_parent_id)
- def check_if_comment_is_a_question (given_string)
- def calculate_comment_upvotes_and_response_time_by_host (id_of_thread, author_of_thread)
- def generate_data_to_be_analyzed ()
- def plot_the_generated_data ()

Variables

- string argument_year = ""
- string argument_tier_in_scope = ""
- string argument_plot_time_unit = ""
- int argument_plot_x_limiter = 0
- mongo_DB_Client_Instance = None
- mongo_DB_Threads_Instance = None
- mongo_DB_Thread_Collection = None
- mongo_DB_Comments_Instance = None
- list list_To_Be_Plotted = []

Function Documentation

def analyze_correlation_upvote_reaction_time_pieChart.calculate_comment_upvotes_and_response_time_by_host (*id_of_thread*, *author_of_thread*)

```
Calculates the arithmetic mean of the answer time by the iama host in minutes

In dependence of the given tier argument (second argument) the processing of tiers will be filtered

Args:
    id_of_thread (str): The id of the thread which is actually processed. (Necessary for checking
if a question
        lies on tier 1 or any other tier)
    author_of_thread (str): The name of the thread author. (Necessary for checking if a given answer
is from the
        iama host or not)
Returns:
    Whenever there was a minimum of 1 question asked and 1 answer from the iama host:
        amount of upvotes reaction time (int) : The amount of the arithmetic mean time of
    Whenever there no questions have been asked for that thread / or no answers were given /
        or all values in the database were null:
        None:   Returns an empty object of the type None
```

def analyze_correlation_upvote_reaction_time_pieChart.calculate_time_difference (***comment_time_stamp*****,** ***answer_time_stamp_iama_host*****)**

```
Calculates the time difference in seconds between the a comment and its answer from the iama host

1. The time stamps will be converted from epoch
    into float and afterwards into str again
    (necessary for correct subtraction)
2. Then the time stamps will be subtracted from each other
3. The containing time difference will be converted into seconds (int)

Args:
    comment time stamp (str): The time stamp of the comment
    answer_time_stamp_iama_host (str): The time stamp of the iAMA hosts answer
Returns:
    time_difference_in_seconds (int) : The time difference of the comment and its answer by the
iAMA host in seconds
```

def analyze_correlation_upvote_reaction_time_pieChart.check_if_comment_is_a_question (***given_string*****)**

```
Simply checks whether a given string is a question or not

1. This method simply checks wether a question mark exists within that string or not..
    This is just that simple because messing around with natural processing kits to determine the
semantic sense
    would blow up my bachelor work...

Args:
    given string (int) : The string which will be checked for a question mark
Returns:
    True (bool): Whenever the given string is a question
```

False (bool): Whenever the given string is not a question

def analyze_correlation_upvote_reaction_time_pieChart.check_if_comment_is_answer_from_thread_author (***author_of_thread*****,** ***comment_acutal_id*****,** ***comments_cursor*****)**

```
Checks whether both strings are equal or not

1. A dictionary containing flags whether that a question is answered by the host with the appropriate
timestamp will
    be created in the beginning.
2. Then the method iterates over every comment within that thread
    1.1. Whenever an answer is from the iAMA hosts and the processed comments 'parent id' matches
the iAMA hosts
        comments (answers) id, the returned dict will contain appropriate values and will be
returned
    1.2. If this is not the case, it will be returned in its default condition

Args:
    author_of_thread (str) : The name of the thread author (iAMA-Host)
    comment_acutal_id (str) : The id of the actually processed comment
    comments cursor (Cursor) : The cursor which shows to the amount of comments which can be iterated
Returns:
    True (bool): Whenever the strings do not match
    False (bool): Whenever the strings do match
     answered that given question)
```

def analyze_correlation_upvote_reaction_time_pieChart.check_if_comment_is_not_from_thread_author (*author_of_thread*, *comment_author*)

```
Checks whether both strings are equal or not

1. This method simply checks wether both strings match each other or not.
    I have built this extra method to have a better overview in the main code..

Args:
    author of thread (str) : The name of the thread author (iAMA-Host)
    comment_author (str) : The name of the comments author
Returns:
    True (bool): Whenever the strings do not match
    False (bool): Whenever the strings do match
     answered that given question)
```

def analyze_correlation_upvote_reaction_time_pieChart.check_if_comment_is_on_tier_1 (*comment_parent_id*)

```
Checks whether a comment relies on the first tier or any other tier

Args:
    comment_parent_id (str) : The name id of the comments parent
Returns:
    True (bool): Whenever the comment lies on tier 1
    False (bool): Whenever the comment lies on any other tier
```

def analyze_correlation_upvote_reaction_time_pieChart.check_script_arguments ()

```
Checks if enough and correct arguments have been given to run this script adequate

1. It checks in the first instance if enough arguments have been given
2. Then necessary variables will be filled with appropriate values

Args:
    -
Returns:
    -
```

def analyze_correlation_upvote_reaction_time_pieChart.generate_data_to_be_analyzed ()

```
Generates the data which will be analyzed

1. This method iterates over every thread
    1.1. It filters if that iterated thread is an iAMA-request or not
        1.1.1. If yes: this thread gets skipped and the next one will be processed
        1.1.2. If no: this thread will be processed
2. If the thread gets processed it will receive the arithmetic mean of answer time
3. This value will be added to a global list and will be plotted later on
Args:
    -
Returns:
    -
```

def analyze_correlation_upvote_reaction_time_pieChart.initialize_mongo_db_parameters ()

```
Instantiates all necessary variables for the correct usage of the mongoDB-Client

Args:
    -
Returns:
    -
```

def analyze_correlation_upvote_reaction_time_pieChart.plot_the_generated_data ()

```
Plots the data which is to be generated

1. This method plots the data which has been calculated before by using 'matplotlib.pyplot-library'
2. In dependence of the chosen time unit the values will be seperated in either minutes or hours

Args:
    -
Returns:
    -
```

Variable Documentation

string analyze_correlation_upvote_reaction_time_pieChart.argument_plot_time_unit = ""

int analyze_correlation_upvote_reaction_time_pieChart.argument_plot_x_limiter = 0

string analyze_correlation_upvote_reaction_time_pieChart.argument_tier_in_scope = ""

string analyze_correlation_upvote_reaction_time_pieChart.argument_year = ""

list analyze_correlation_upvote_reaction_time_pieChart.list_To_Be_Plotted = []

analyze_correlation_upvote_reaction_time_pieChart.mongo_DB_Client_Instance = None

analyze_correlation_upvote_reaction_time_pieChart.mongo_DB_Comments_Instance = None

analyze_correlation_upvote_reaction_time_pieChart.mongo_DB_Thread_Collection = None

analyze_correlation_upvote_reaction_time_pieChart.mongo_DB_Threads_Instance = None

analyze_thread_lifeSpan_n_average_commentTime_pieChart Namespace Reference

Functions

- def initialize_mongo_db_parameters ()
- def check_script_arguments ()
- def calculate_time_difference (id_of_thread, creation_date_of_thread)
- def generate_data_to_be_analyzed ()
- def prepare_dict_by_time_separation_for_life_span ()
- def prepare_dict_by_time_separation_for_comment_time ()
- def plot_the_generated_data ()

Variables

- string argument_year = ""
- string argument_calculation = ""
- string argument_plot_time_unit = ""
- mongo_DB_Client_Instance = None
- mongo_DB_Threads_Instance = None
- mongo_DB_Thread_Collection = None
- mongo_DB_Comments_Instance = None
- list list_To_Be_Plotted = []

Function Documentation

def analyze_thread_lifeSpan_n_average_commentTime_pieChart.calculate_time_difference (*id_of_thread*, *creation_date_of_thread*)

```
Calculates the difference between thread creation date and the last comment found in that thread

1. The creation date of a thread gets determined
2. Then the comments will be iterated over, creating a dictionary which is structured as follows:
  {
      ('first_Comment_After_Thread_Started', int),
      ('thread Lifespan', int),
      ('arithmetic_Mean_Response_Time', int),
      ('median Response Time', int),
      ('id')
  }
3. That returned dictionary will be appended to a global list
4. That List will be iterated later on and the appropriate graph will be plotted

Args:
    id_of_thread (str) : The string which contains the id of the actually processed thread
    creation_date_of_thread (str) : The string which contains the creation date of the thread (in
epoch formatation)
Returns:
```

dict_to_be_returned (dict) : Containing information about the time difference

def analyze_thread_lifeSpan_n_average_commentTime_pieChart.check_script_arguments ()

```
Checks if enough and correct arguments have been given to run this script adequate

1. It checks in the first instance if enough arguments have been given
2. Then necessary variables will be filled with appropriate values
```

```
Args:
    -
Returns:
    -
```

def analyze_thread_lifeSpan_n_average_commentTime_pieChart.generate_data_to_be_analyzed ()

```
Generates the data which will be analyzed

1. This method iterates over every thread
    1.1. It filters if that iterated thread is an iAMA-request or not
        1.1.1. If yes: this thread gets skipped and the next one will be processed
        1.1.2. If no: this thread will be processed
2. If the thread gets processed it will receive the life span and other information about the thread
as dictionary
3. This dictionary will be added to a global list and will be plotted later on

Args:
    -
Returns:
    -
```

def analyze_thread_lifeSpan_n_average_commentTime_pieChart.initialize_mongo_db_parameters ()

```
Instantiates all necessary variables for the correct usage of the mongoDB-Client

Args:
    -
Returns:
    -
```

def analyze_thread_lifeSpan_n_average_commentTime_pieChart.plot_the_generated_data ()

```
Plots the data which is to be generated

1. This method plots the data which has been calculated before by using 'matplotlib.pyplot-library'
2. Depending on the committed year the title will be adapted appropriate
3. Time units will be separated into days, because this gives us the best overview

Args:
    -
Returns:
    -
```

def analyze_thread_lifeSpan_n_average_commentTime_pieChart.prepare_dict_by_time_separation_for_comment_time ()

```
Restructures the dictionary which is to be plotted for the display of the average mean comment time

1. This method processes the data in dependence of the commited time

Args:
    -
Returns:
```

```
    -
```

def analyze_thread_lifeSpan_n_average_commentTime_pieChart.prepare_dict_by_time_separation_for_life_span ()

```
Restructures the dictionary which is to be plotted for the display of the life span

1. This method processes the data in dependence of the commited time

Args:
    -
Returns:
    -
```

Variable Documentation

string analyze_thread_lifeSpan_n_average_commentTime_pieChart.argument_calculation = ""

string analyze_thread_lifeSpan_n_average_commentTime_pieChart.argument_plot_time_unit = ""

string analyze_thread_lifeSpan_n_average_commentTime_pieChart.argument_year = ""

list analyze_thread_lifeSpan_n_average_commentTime_pieChart.list_To_Be_Plotted = []

analyze_thread_lifeSpan_n_average_commentTime_pieChart.mongo_DB_Client_Instance = None

analyze_thread_lifeSpan_n_average_commentTime_pieChart.mongo_DB_Comments_Instance = None

analyze_thread_lifeSpan_n_average_commentTime_pieChart.mongo_DB_Thread_Collection = None

analyze_thread_lifeSpan_n_average_commentTime_pieChart.mongo_DB_Threads_Instance = None

analyze_tier_answered_percentage_pieChart Namespace Reference

Functions

- def initialize_mongo_db_parameters ()
- def check_script_arguments ()
- def calculate_percentage_distribution (amount_of_questions, amount_of_questions_answered)
- def check_if_comment_is_not_from_thread_author (author_of_thread, comment_author)
- def check_if_comment_is_answer_from_thread_author (author_of_thread, comment_actual_id, comments_cursor)
- def check_if_comment_is_on_tier_1 (comment_parent_id)
- def check_if_comment_is_a_question (given_string)
- def amount_of_questions_answered_by_host (id_of_thread, author_of_thread)
- def generate_data_to_be_analyzed ()
- def plot_the_generated_data ()

Variables

- string argument_year = ""
- string argument_tier_in_scope = ""
- mongo_DB_Client_Instance = None
- mongo_DB_Threads_Instance = None
- mongo_DB_Thread_Collection = None
- mongo_DB_Comments_Instance = None
- list list_To_Be_Plotted = []

Function Documentation

def analyze_tier_answered_percentage_pieChart.amount_of_questions_answered_by_host (*id_of_thread*, *author_of_thread*)

```
Generates the data which will be analyzed

1. It iterates over every comment and
     1.1. checks if the iterated comment is a question
     1.2. checks if the iterated comment has been posted on tier 1 level
     1.3. checks if that comment is from the iAMA-Host himself or not
2. Now the distribution of questions answered / not answered will be calculated depending on the
commited tier level
    3. A dictionary containing the amounts in percentage will be returned

id_of_thread (str) : Contains the id of the processed thread
author_of_thread (str) : Contains the iAMA-Hosts name

Returns:
dict_to_be_returned_percentage_answered_questions (dict) : Containing the percentage amount of
questions
```

which have been answered and which have not been answered

def analyze_tier_answered_percentage_pieChart.calculate_percentage_distribution (*amount_of_questions*, *amount_of_questions_answered*)

```
Checks whether both strings are equal or not

1. This method simply checks wether both strings match each other or not.
    I have built this extra method to have a better overview in the main code..
```

```
Args:
    amount_of_questions (int) : The amount of questions which have been asked at all
    amount_of_questions_answered (int) : The amount of questions which have been answered
Returns:
    dict_to_be_returned (dict): A dictionary containing
        1. The amount of questions answered as int
        2. The amount of questions which have not been answered
     answered that given question)
```

def analyze_tier_answered_percentage_pieChart.check_if_comment_is_a_question (*given_string*)

```
Simply checks whether a given string is a question or not

1. This method simply checks whether a question mark exists within that string or not..
    This is just that simple because messing around with natural processing kits to determine the
semantic sense
    would blow up my bachelor work...

Args:
    given string (int) : The string which will be checked for a question mark
Returns:
    True (bool): Whenever the given string is a question
```

False (bool): Whenever the given string is not a question

def analyze_tier_answered_percentage_pieChart.check_if_comment_is_answer_from_thread_author (*author_of_thread*, *comment_actual_id*, *comments_cursor*)

```
Checks whether both strings are equal or not

1. A dictionary containing flags whether that a question is answered by the host with the appropriate
timestamp will
    be created in the beginning.
2. Then the method iterates over every comment within that thread
    1.1. Whenever an answer is from the iAMA hosts and the processed comments 'parent id' matches
the iAMA hosts
        comments (answers) id, the returned dict will contain appropriate values and will be
returned
    1.2. If this is not the case, it will be returned in its default condition

Args:
    author_of_thread (str) : The name of the thread author (iAMA-Host)
    comment_actual_id:  (str) : The id of the actually processed comment
    comments_cursor (Cursor) : The cursor which shows to the amount of comments which can be iterated
Returns:
    True (bool): Whenever the strings do not match
    False (bool): Whenever the strings do match
     answered that given question)
     :param
```

def analyze_tier_answered_percentage_pieChart.check_if_comment_is_not_from_thread_author (*author_of_thread*, *comment_author*)

```
Checks whether both strings are equal or not

1. This method simply checks wether both strings match each other or not.
    I have built this extra method to have a better overview in the main code..

Args:
    author_of_thread (str) : The name of the thread author (iAMA-Host)
```

```
    comment_author (str) : The name of the comments author
Returns:
    True (bool): Whenever the strings do not match
    False (bool): Whenever the strings do match
     answered that given question)
```

def analyze_tier_answered_percentage_pieChart.check_if_comment_is_on_tier_1 (*comment_parent_id*)

```
Simply checks whether a given string is a question posted on tier 1 or not

1. This method simply checks whether a question has been posted on tier 1 by looking whether the
given
    string contains the substring "t3 " or not
Args:
    comment parent id (str): The string which will be checked for "t3 " appearance in it
Returns:
    -
```

def analyze_tier_answered_percentage_pieChart.check_script_arguments ()

```
Checks if enough and correct arguments have been given to run this script adequate

1. It checks in the first instance if enough arguments have been given
2. Then necessary variables will be filled with appropriate values

Args:
    -
Returns:
    -
```

def analyze_tier_answered_percentage_pieChart.generate_data_to_be_analyzed ()

```
Generates the data which will be analyzed

1. This method iterates over every thread
    1.1. It filters if that iterated thread is an iAMA-request or not
        1.1.1. If yes: this thread gets skipped and the next one will be processed
        1.1.2. If no: this thread will be processed
2. If the thread gets processed it will receive the amount of questions answered
3. This value will be added to a global list and will be plotted later on
Args:
    -
Returns:
    -
```

def analyze_tier_answered_percentage_pieChart.initialize_mongo_db_parameters ()

```
Instantiates all necessary variables for the correct usage of the mongoDB-Client

Args:
    -
Returns:
    -
```

def analyze_tier_answered_percentage_pieChart.plot_the_generated_data ()

```
Plots the data which is to be generated

1. This method plots the data which has been calculated before by using 'matplotlib.pyplot-library'
2. Depending on the committed year and tier scope the title will be adapted appropriate

Args:
    -
Returns:
    -
```

Variable Documentation

string analyze_tier_answered_percentage_pieChart.argument_tier_in_scope = ""

string analyze_tier_answered_percentage_pieChart.argument_year = ""

list analyze_tier_answered_percentage_pieChart.list_To_Be_Plotted = []

analyze_tier_answered_percentage_pieChart.mongo_DB_Client_Instance = None

analyze_tier_answered_percentage_pieChart.mongo_DB_Comments_Instance = None

analyze_tier_answered_percentage_pieChart.mongo_DB_Thread_Collection = None

analyze_tier_answered_percentage_pieChart.mongo_DB_Threads_Instance = None

analyze_tier_answered_time_pieChart Namespace Reference

Functions

- def initialize_mongo_db_parameters ()
- def check_script_arguments ()
- def calculate_time_difference (comment_time_stamp, answer_time_stamp_iama_host)
- def check_if_comment_is_answer_from_thread_author (author_of_thread, comment_actual_id, comments_cursor)
- def check_if_comment_is_not_from_thread_author (author_of_thread, comment_author)
- def check_if_comment_is_on_tier_1 (comment_parent_id)
- def check_if_comment_is_a_question (given_string)
- def calculate_ar_mean_answer_time_for_questions (id_of_thread, author_of_thread)
- def generate_data_to_be_analyzed ()
- def plot_the_generated_data ()

Variables

- string argument_year = ""
- string argument_tier_in_scope = ""
- string argument_plot_time_unit = ""
- mongo_DB_Client_Instance = None
- mongo_DB_Threads_Instance = None
- mongo_DB_Thread_Collection = None
- mongo_DB_Comments_Instance = None
- list list_To_Be_Plotted = []

Function Documentation

def analyze_tier_answered_time_pieChart.calculate_ar_mean_answer_time_for_questions (*id_of_thread*, *author_of_thread*)

```
Calculates the arithmetic mean of the answer time by the iama host in minutes

In dependence of the given tier argument (second argument) the processing of tiers will be filtered

Args:
    id_of_thread (str): The id of the thread which is actually processed. (Necessary for checking
if a question
        lies on tier 1 or any other tier)
    author_of_thread (str): The name of the thread author. (Necessary for checking if a given answer
is from the
        iama host or not)
Returns:
    Whenever there was a minimum of 1 question asked and 1 answer from the iama host:
        amount_of_answer_times (int) : The amount of the arithmetic mean time of
    Whenever there no questions have been asked for that thread / or no answers were given /
        or all values in the database were null:
        None:   Returns an empty object of the type None
```

def analyze_tier_answered_time_pieChart.calculate_time_difference (*comment_time_stamp*, *answer_time_stamp_iama_host*)

```
Calculates the time difference in seconds between the a comment and its answer from the iama host
```

```
1. The time stamps will be converted from epoch
    into float and afterwards into str again
    (necessary for correct subtraction)
2. Then the time stamps will be subtracted from each other
3. The containing time difference will be converted into seconds (int)

Args:
    comment_time_stamp (str): The time stamp of the comment
    answer time stamp iama host (str): The time stamp of the iAMA hosts answer
Returns:
    time difference in seconds (int) : The time difference of the comment and its answer by the
iAMA host in seconds
```

def analyze_tier_answered_time_pieChart.check_if_comment_is_a_question (*given_string*)

```
Simply checks whether a given string is a question or not

1. This method simply checks wether a question mark exists within that string or not..
    This is just that simple because messing around with natural processing kits to determine the
semantic sense
    would blow up my bachelor work...

Args:
    given string (int) : The string which will be checked for a question mark
Returns:
    True (bool): Whenever the given string is a question
```

False (bool): Whenever the given string is not a question

def analyze_tier_answered_time_pieChart.check_if_comment_is_answer_from_thread_author (*author_of_thread*, *comment_actual_id*, *comments_cursor*)

```
Checks whether both strings are equal or not

1. A dictionary containing flags whether that a question is answered by the host with the appropriate
timestamp will
    be created in the beginning.
2. Then the method iterates over every comment within that thread
    1.1. Whenever an answer is from the iAMA hosts and the processed comments 'parent_id' matches
the iAMA hosts
        comments (answers) id, the returned dict will contain appropriate values and will be
returned
    1.2. If this is not the case, it will be returned in its default condition

Args:
    author_of_thread (str) : The name of the thread author (iAMA-Host)
    comment actual id (str) : The id of the actually processed comment
    comments_cursor (Cursor) : The cursor which shows to the amount of comments which can be iterated
Returns:
    True (bool): Whenever the strings do not match
    False (bool): Whenever the strings do match
     answered that given question)
```

def analyze_tier_answered_time_pieChart.check_if_comment_is_not_from_thread_author (*author_of_thread*, *comment_author*)

```
Checks whether both strings are equal or not

1. This method simply checks wether both strings match each other or not.
    I have built this extra method to have a better overview in the main code..

Args:
    author_of_thread (str) : The name of the thread author (iAMA-Host)
```

```
    comment_author (str) : The name of the comments author
Returns:
    True (bool): Whenever the strings do not match
    False (bool): Whenever the strings do match
     answered that given question)
```

def analyze_tier_answered_time_pieChart.check_if_comment_is_on_tier_1 (*comment_parent_id*)

```
Checks whether a comment relies on the first tier or any other tier

Args:
    comment_parent_id (str) : The name id of the comments parent
Returns:
    True (bool): Whenever the comment lies on tier 1
    False (bool): Whenever the comment lies on any other tier
```

def analyze_tier_answered_time_pieChart.check_script_arguments ()

```
Checks if enough and correct arguments have been given to run this script adequate

1. It checks in the first instance if enough arguments have been given
2. Then necessary variables will be filled with appropriate values

Args:
    -
Returns:
    -
```

def analyze_tier_answered_time_pieChart.generate_data_to_be_analyzed ()

```
Generates the data which will be analyzed

1. This method iterates over every thread
    1.1. It filters if that iterated thread is an iAMA-request or not
        1.1.1. If yes: this thread gets skipped and the next one will be processed
        1.1.2. If no: this thread will be processed
2. If the thread gets processed it will receive the arithmetic mean of answer time
3. This value will be added to a global list and will be plotted later on
Args:
    -
Returns:
    -
```

def analyze_tier_answered_time_pieChart.initialize_mongo_db_parameters ()

```
Instantiates all necessary variables for the correct usage of the mongoDB-Client

Args:
    -
Returns:
    -
```

def analyze_tier_answered_time_pieChart.plot_the_generated_data ()

```
Plots the data which is to be generated

1. This method plots the data which has been calculated before by using 'matplotlib.pyplot-library'
2. In dependence of the chosen time unit the values will be seperated in either minutes or hours

Args:
    -
Returns:
    -
```

Variable Documentation

string analyze_tier_answered_time_pieChart.argument_plot_time_unit = ""

string analyze_tier_answered_time_pieChart.argument_tier_in_scope = ""

string analyze_tier_answered_time_pieChart.argument_year = ""

list analyze_tier_answered_time_pieChart.list_To_Be_Plotted = []

analyze_tier_answered_time_pieChart.mongo_DB_Client_Instance = None

analyze_tier_answered_time_pieChart.mongo_DB_Comments_Instance = None

analyze_tier_answered_time_pieChart.mongo_DB_Thread_Collection = None

analyze_tier_answered_time_pieChart.mongo_DB_Threads_Instance = None

analyze_tier_question_distribution_pieChart Namespace Reference

Functions

- def initialize_mongo_db_parameters ()
- def check_script_arguments ()
- def check_if_comment_is_not_from_thread_author (author_of_thread, comment_author)
- def calculate_percentage_distribution (amount_of_tier_1_questions, amount_of_tier_x_questions)
- def check_if_comment_is_on_tier_1 (comment_parent_id)
- def check_if_comment_is_a_question (given_string)
- def amount_of_tier_1_questions_percentage (id_of_thread, author_of_thread)
- def generate_data_to_be_analyzed ()
- def plot_the_generated_data ()

Variables

- string argument_year = ""
- mongo_DB_Client_Instance = None
- mongo_DB_Threads_Instance = None
- mongo_DB_Thread_Collection = None
- mongo_DB_Comments_Instance = None
- list list_To_Be_Plotted = []

Function Documentation

def analyze_tier_question_distribution_pieChart.amount_of_tier_1_questions_percentage (*id_of_thread*, *author_of_thread*)

```
Generates the data which will be analyzed

1. It iterates over every comment and
     1.1. checks if the iterated comment is a question
     1.2. checks if the iterated comment has been posted on tier 1 level
     1.3. checks if that comment is from the iAMA-Host himself or not

2. Now the distribution of on the tier levels will be calculated
3. Then a dict will be returned containing the distribution amount of questions on the tier levels
in percentage

id_of_thread (str) : Contains the id of the processed thread
author of thread (str) : Contains the iAMA-Hosts name

Returns:
dict_to_be_returned_percentage_answered_questions (dict) : Containing the percentage amount of
questions
```

on tier 1 and the other tiers

def analyze_tier_question_distribution_pieChart.calculate_percentage_distribution (*amount_of_tier_1_questions*, *amount_of_tier_x_questions*)

```
Checks whether both strings are equal or not

1. This method simply checks wether both strings match each other or not.
    I have built this extra method to have a better overview in the main code..

Args:
    amount_of_tier_1_questions (int) : The amount of questions which have been asked at all
```

```
    amount_of_tier_x_questions (int) : The amount of questions which have been answered
Returns:
    dict_to_be_returned (dict): A dictionary containing
        1. The amount of questions answered as int
        2. The amount of questions which have not been answered
     answered that given question)
```

def analyze_tier_question_distribution_pieChart.check_if_comment_is_a_question (*given_string*)

```
Simply checks whether a given string is a question or not

1. This method simply checks whether a question mark exists within that string or not..
    This is just that simple because messing around with natural processing kits to determine the
semantic sense
    would blow up my bachelor work...

Args:
    given_string (int) : The string which will be checked for a question mark
Returns:
    True (bool): Whenever the given string is a question
```

False (bool): Whenever the given string is not a question

def analyze_tier_question_distribution_pieChart.check_if_comment_is_not_from_thread_author (*author_of_thread*, *comment_author*)

```
Checks whether both strings are equal or not

1. This method simply checks wether both strings match each other or not.
    I have built this extra method to have a better overview in the main code..

Args:
    author_of_thread (str) : The name of the thread author (iAMA-Host)
    comment author (str) : The name of the comments author
Returns:
    True (bool): Whenever the strings do not match
    False (bool): Whenever the strings do match
     answered that given question)
```

def analyze_tier_question_distribution_pieChart.check_if_comment_is_on_tier_1 (*comment_parent_id*)

```
Simply checks whether a given string is a question posted on tier 1 or not

1. This method simply checks whether a question has been posted on tier 1 by looking whether the
given
    string contains the substring "t3_" or not
Args:
    comment_parent_id (str): The string which will be checked for "t3_" appearance in it
Returns:
    -
```

def analyze_tier_question_distribution_pieChart.check_script_arguments ()

```
Checks if enough and correct arguments have been given to run this script adequate

1. It checks in the first instance if enough arguments have been given
2. Then necessary variables will be filled with appropriate values
```

```
Args:
    -
Returns:
    -
```

def analyze_tier_question_distribution_pieChart.generate_data_to_be_analyzed ()

```
Generates the data which will be analyzed

1. This method iterates over every thread
    1.1. It filters if that iterated thread is an iAMA-request or not
        1.1.1. If yes: this thread gets skipped and the next one will be processed
        1.1.2. If no: this thread will be processed
2. If the thread gets processed it will receive the distribution of questions on the tiers
3. This value will be added to a global list and will be plotted later on
Args:
    -
Returns:
    -
```

def analyze_tier_question_distribution_pieChart.initialize_mongo_db_parameters ()

```
Instantiates all necessary variables for the correct usage of the mongoDB-Client

Args:
    -
Returns:
    -
```

def analyze_tier_question_distribution_pieChart.plot_the_generated_data ()

```
Plots the data which is to be generated

1. This method plots the data which has been calculated before by using 'matplotlib.pyplot-library'
2. Depending on the committed year the title will be adapted appropriate

Args:
    -
Returns:
    -
```

Variable Documentation

string analyze_tier_question_distribution_pieChart.argument_year = ""

list analyze_tier_question_distribution_pieChart.list_To_Be_Plotted = []

analyze_tier_question_distribution_pieChart.mongo_DB_Client_Instance = None

analyze_tier_question_distribution_pieChart.mongo_DB_Comments_Instance = None

analyze_tier_question_distribution_pieChart.mongo_DB_Thread_Collection = None

analyze_tier_question_distribution_pieChart.mongo_DB_Threads_Instance = None

analyze_top100_pieChart Namespace Reference

Functions

- def initialize_mongo_db_parameters ()
- def check_script_arguments ()
- def calculate_time_difference (comment_time_stamp, answer_time_stamp_iama_host)
- def check_if_comment_is_answer_from_thread_author (author_of_thread, comment_acutal_id, comments_cursor)
- def check_if_comment_is_not_from_thread_author (author_of_thread, comment_author)
- def check_if_comment_is_a_question (given_string)
- def calculate_answered_question_upvote_correlation (id_of_thread, author_of_thread, thread_creation_date)
- def generate_data_to_be_analyzed ()
- def prepare_and_print_data_to_be_plotted ()
- def plot_generated_data (amount_of_questions_not_answered)

Variables

- string argument_year = ""
- argument_sorting = bool
- mongo_DB_Client_Instance = None
- mongo_DB_Threads_Instance = None
- mongo_DB_Thread_Collection = None
- mongo_DB_Comments_Instance = None
- list list_To_Be_Plotted = []

Function Documentation

def analyze_top100_pieChart.calculate_answered_question_upvote_correlation (*id_of_thread*, *author_of_thread*, *thread_creation_date*)

```
Checks whether an iterated question has been answered by the iama host or not

1. This method checks at first whether an iterated comment contains values (e.g. is not none)
    1.1. If not: That comment will be skipped / if no comment is remaining None will be returned
    1.2. If yes: That comment will be processed
2. Now it will be checked whether that iterated comment is a question or not
3. Afterwards it will be checked wether that comment is a comment from the iAMA Host or not
    3.1. If this is not the case the next comment will be processed
4. Whenever that processed comment is a question and not (!!) from the thread author:
    amount of tier any questions (int) will be increased by one
5. Now it will be checked whether that comment has a comment ( answer ) below it which is from the
iAMA-host
    5.1. If yes: amount_of_tier_any_questions_answered (int) will be increased by one and the
dictionary, which
        is to be returned will be filled with values
    5.2. If no: the dictionary, which is to be returned will be filled with values

Args:
    id_of_thread (str) : Contains the id of the thread which is to be iterated
    author of thread (str) : Contains the name of the thread author
    thread creation date (str): Contains the time
Returns:
    amount_of_questions_not_answered (int) : The amount of questions which have not been answered
```

def analyze_top100_pieChart.calculate_time_difference (*comment_time_stamp*, *answer_time_stamp_iama_host*)

```
Calculates the time difference in seconds between the a comment and its answer from the iama host

1. The time stamps will be converted from epoch
    into float and afterwards into str again
    (necessary for correct subtraction)
2. Then the time stamps will be subtracted from each other
3. The containing time difference will be converted into seconds (int)

Args:
    comment time stamp (str): The time stamp of the comment
    answer_time_stamp_iama_host (str): The time stamp of the iAMA hosts answer
Returns:
    time_difference_in_seconds (int) : The time difference of the comment and its answer by the
iAMA host in seconds
```

def analyze_top100_pieChart.check_if_comment_is_a_question (*given_string*)

```
Simply checks whether a given string is a question or not

1. This method simply checks wether a question mark exists within that string or not..
    This is just that simple because messing around with natural processing kits to determine the
semantic sense
    would blow up my bachelor work...

Args:
    given_string (int) : The string which will be checked for a question mark
Returns:
    True (bool): Whenever the given string is a question
```

False (bool): Whenever the given string is not a question

def analyze_top100_pieChart.check_if_comment_is_answer_from_thread_author (*author_of_thread*, *comment_acutal_id*, *comments_cursor*)

```
Checks whether both strings are equal or not

1. A dictionary containing flags whether that a question is answered by the host with the appropriate
timestamp will
    be created in the beginning.
2. Then the method iterates over every comment within that thread
    1.1. Whenever an answer is from the iAMA hosts and the processed comments 'parent_id' matches
the iAMA hosts
        comments (answers) id, the returned dict will contain appropriate values and will be
returned
    1.2. If this is not the case, it will be returned in its default condition

Args:
    author of thread (str) : The name of the thread author (iAMA-Host)
    comment_acutal_id (str) : The id of the actually processed comment
    comments_cursor (Cursor) : The cursor which shows to the amount of comments which can be iterated
Returns:
    True (bool): Whenever the strings do not match
    False (bool): Whenever the strings do match
     answered that given question)
```

def analyze_top100_pieChart.check_if_comment_is_not_from_thread_author (*author_of_thread*, *comment_author*)

```
Checks whether both strings are equal or not

1. This method simply checks wether both strings match each other or not.
    I have built this extra method to have a better overview in the main code..

Args:
    author_of_thread (str) : The name of the thread author (iAMA-Host)
    comment_author (str) : The name of the comments author
Returns:
    True (bool): Whenever the strings do not match
    False (bool): Whenever the strings do match
     answered that given question)
```

def analyze_top100_pieChart.check_script_arguments ()

```
Checks if enough and correct arguments have been given to run this script adequate

1. It checks in the first instance if enough arguments have been given
2. Afterwards it fills 'argument year' with the first argument (str) and 'argument sorting' with
a boolean value,
    by previously parsing and checking that value.

Args:
    -
Returns:
    -
```

def analyze_top100_pieChart.generate_data_to_be_analyzed ()

```
Generates the data which will be analyzed

1. This method iterates over every thread
    1.1. It filters if that iterated thread is an iAMA-request or not
        1.1.1. If yes: this thread gets skipped and the next one will be processed
        1.1.2. If no: this thread will be processed
2. If the thread gets processed it will receive an ordered dictionary containing information about
every question
    whether it has been answered or not
3. This ordered dictionary will be applied to a global list, which will be processed after wards
for the generation
    of plots

Args:
    -
Returns:
    -
```

def analyze_top100_pieChart.initialize_mongo_db_parameters ()

```
Instantiates all necessary variables for the correct usage of the mongoDB-Client

Args:
    -
Returns:
    -
```

def analyze_top100_pieChart.plot_generated_data (*amount_of_questions_not_answered*)

```
Plots the data which is to be generated

1. This method plots the data which has been calculated before by using 'matplotlib.pyplot-library'

Args:
    amount_of_questions_not_answered (int): The amount of questions which have not been answered.
Returns:
    -
```

def analyze_top100_pieChart.prepare_and_print_data_to_be_plotted ()

```
Prepares data and prints data into the command line

1. This method prepares the data, in kind of sorting and counting amount of questions not being
answered
2. Afterwards it prints, in dependency of the second argument given of this script, whether the
    TOP or WORST 100 questions have been answered or not

Args:
    -
Returns:
    amount_of_questions_not_answered (int) : The amount of questions which have not been answered
```

Variable Documentation

analyze_top100_pieChart.argument_sorting = bool

string analyze_top100_pieChart.argument_year = ""

list analyze_top100_pieChart.list_To_Be_Plotted = []

analyze_top100_pieChart.mongo_DB_Client_Instance = None

analyze_top100_pieChart.mongo_DB_Comments_Instance = None

analyze_top100_pieChart.mongo_DB_Thread_Collection = None

analyze_top100_pieChart.mongo_DB_Threads_Instance = None

crawl_differences Namespace Reference

Functions

- def check_script_arguments ()
- def initialize_mongo_db_parameters ()
- def crawl_missing_collection_into_comments_database (name_of_missing_collection)
- def check_if_collection_is_missing_in_comments_database ()
- def crawl_missing_collection_into_threads_database (name_of_missing_collection)
- def check_if_collection_is_missing_in_threads_database ()
- def start_crawling_for_diffs ()

Variables

- mongo_DB_Client_Instance = None
- mongo_DB_Threads_Instance = None
- mongo_DB_Thread_Collection = None
- mongo_DB_Comments_Instance = None
- mongo_DB_Comments_Collection = None
- string argument_year_beginning = ""
- string argument_year_end = ""
- string argument_inverse_crawling = ""

Function Documentation

def crawl_differences.check_if_collection_is_missing_in_comments_database ()

```
Checks if a specific collection (thread) is missing in the appropriate comments database

    The method starts the diff checking for all collections within the threads database.
    Whenever a thread exists in the comment database but not in the threads database it will be
crawled from the
    reddit servers and written into the database.

Args:
    -
Returns:
    -
```

def crawl_differences.check_if_collection_is_missing_in_threads_database ()

```
Checks if a specific collection (thread) is missing in the appropriate threads database

    The method starts the diff checking for all collections within the threads database.
    Whenever a thread exists in the comment database but not in the threads database it will be
crawled from the
    reddit servers and written into the database.

Args:
    -
Returns:
    -
```

def crawl_differences.check_script_arguments ()

```
Checks if enough and correct arguments have been given to run this script adequate

1. It checks in the first instance if enough arguments have been given
2. Then necessary variables will be filled with appropriate values

Args:
    -
Returns:
    -
```

def crawl_differences.crawl_missing_collection_into_comments_database (*name_of_missing_collection*)

```
Crawls a specific thread, which is missing in the comments database and writes the appropriate entry
in the db

    The method works as follows:
    1. It checks whether that thread / collection is really missing (even when that has been done
before, we check
        it again here, just to make sure that collection has not been created in the meanwhile by
another crawling
        process.
    2. Now the comments will be crawled from the reddit servers with flattened hierarchy
    3. Yet the comments will be written into the appropriate comments database. The correct database
will be
        deviated from the threads creation timestamp.


Args:
    name of missing collection (str) : The id of the collection which is actually missing in the
comments database
Returns:
    -
```

def crawl_differences.crawl_missing_collection_into_threads_database (*name_of_missing_collection*)

```
Crawls a specific thread, which is missing in the thread database and writes the appropriate entry
in the db

    The method works as follows:
    1. It checks whether that thread / collection is really missing (even when that has been done
before, we check
        it again here, just to make sure that collection has not been created in the meanwhile by
another crawling
        process.
    2. Now the the thread will be crawled from the reddit servers
    3. Yet the thread will be written into the appropriate threads database. The correct database
will be
        deviated from the threads creation timestamp.

Args:
    name_of_missing_collection (str) : The id of the collection which is actually missing in the
comments database
Returns:
    -
```

def crawl_differences.initialize_mongo_db_parameters ()

```
Instantiates all necessary variables for the correct usage of the mongoDB-Client

Args:
    -
Returns:
    -
```

def crawl_differences.start_crawling_for_diffs ()

```
This method starts the crawling, with the method you have defined in your arguments

Args:
    -
Returns:
    -
```

Variable Documentation

string crawl_differences.argument_inverse_crawling = ""

string crawl_differences.argument_year_beginning = ""

string crawl_differences.argument_year_end = ""

crawl_differences.mongo_DB_Client_Instance = None

crawl_differences.mongo_DB_Comments_Collection = None

crawl_differences.mongo_DB_Comments_Instance = None

crawl_differences.mongo_DB_Thread_Collection = None

crawl_differences.mongo_DB_Threads_Instance = None

crawl_threads_n_comments Namespace Reference

Functions

- def initialize_mongo_db_parameters ()
- def check_script_arguments ()
- def convert_argument_year_to_epoch (year)
- def crawl_data ()
- def crawl_threads ()
- def crawl_comments ()
- def check_if_coll_in_db_already_exists_up2date (submission)

Variables

- mongo_DB_Client_Instance = None
- reddit_Instance = None
- argument_crawl_type = None
- argument_year_beginning = None
- argument_year_end = None
- argument_hours_to_shift = None
- time_shift_difference

Function Documentation

def crawl_threads_n_comments.check_if_coll_in_db_already_exists_up2date (*submission*)

```
Checks if a collection already exists in the database or not

This is necessary, otherwise thread information would be written into the database twice.
It works the following way:

1. Define a tolerance factor (necessary because reddit skews information about the amount of
    "upvotes"). Without defining that tolerance factor every thread would be created anew.
    After messing around a few days I found this one to be the best value to work with

2. Create values for temporary values for checking

3. Check and recreate collection if necessary

4. Return appropriate boolean value if collection already existed within the database or not


Args:
    submission (Submission) : The thread which will be processed / iterated over at the moment
Returns:
    True / False (bool) : Whenever the collection already exists within the database (True) or not
(False)
```

def crawl_threads_n_comments.check_script_arguments ()

```
Checks if enough and correct arguments have been given to run this script adequate

1. It checks in the first instance if enough arguments have been given
2. Then necessary variables will be filled with appropriate values

Args:
    -
```

```
Returns:
    -
```

def crawl_threads_n_comments.convert_argument_year_to_epoch (*year*)

```
"Converts" a given string into the appropriate epoch string format (int)

Args:
    year (str) : The year which will be "converted" into epoch format (necessary for correct PRAW
API behaviour)
Returns:
    year (int) : The year "converted" into epoch format as integer
```

def crawl_threads_n_comments.crawl_comments ()

```
Crawls thread information and writes them into the mongoDB storage
It works as follwoing:

1. At first an attempt to the amazon cloud search will be made, with necessary parameters which
returns an object,
    of the class "Generator" which contains all comments for the given / crawled time windows

2. After that the "Generator"s elements will be iterated over

    2.1. It will be checked if that iterated collection already exists within the database or not

        2.2.1. If it already exists, it will be checked whether if it is up to date or not
            2.2.1.1. If up2date: do nothing
            2.2.1.2. If not up2date: drop that collection within the database and crawl the
collection anew

        2.2.2. If it does not yet exist: create that collection in the database with the necessary
information

3. Whenever there are no elements left to iterate over the time crawling window will be shifted
into the future by
    using the given amount in hours (fourth argument), whenever the ending year (third argument)
is not reached yet

Args:
    -
Returns:
    -
```

def crawl_threads_n_comments.crawl_data ()

```
Crawls data from reddit, depending on the first argument (threads / comments) you give the script

Args:
    -
Returns:
    -
```

def crawl_threads_n_comments.crawl_threads ()

```
Crawls thread information and writes them into the mongoDB storage
It works as follwoing:
```

```
1. At first an attempt to the amazon cloud search will be made, with necessary parameters which
returns an object,
    of the class "Generator" which contains all threads for the given / crawled time windows

2. After that the "Generator"s elements will be iterated over

    2.1. It will be checked if that iterated collection already exists within the database or not

        2.2.1. If it already exists, it will be checked whether if it is up to date or not
            2.2.1.1. If up2date: do nothing
            2.2.1.2. If not up2date: drop that collection within the database and crawl the
collection anew

        2.2.2. If it does not yet exist: create that collection in the database with the necessary
information

3. Whenever there are no elements left to iterate over the time crawling window will be shifted
into the future by
    using the given amount in hours (third argument), whenever the ending year (second argument)
is not reached yet

Args:
    -
Returns:
    -
```

def crawl_threads_n_comments.initialize_mongo_db_parameters ()

```
Instantiates all necessary variables for the correct usage of the mongoDB-Client

Args:
    -
Returns:
    -
```

Variable Documentation

crawl_threads_n_comments.argument_crawl_type = None

crawl_threads_n_comments.argument_hours_to_shift = None

crawl_threads_n_comments.argument_year_beginning = None

crawl_threads_n_comments.argument_year_end = None

crawl_threads_n_comments.mongo_DB_Client_Instance = None

crawl_threads_n_comments.reddit_Instance = None

crawl_threads_n_comments.time_shift_difference

```
Initial value:    1 = int(
    2     round(time.mktime(
    3         (datetime.fromtimestamp(argument_year_beginning) +
    4          timedelta(
    5              hours=argument_hours_to_shift)
    6          ).timetuple()
    7     )
    8     )
```

```
    9 )
```

File Documentation

analyze_correlation_upvote_reaction_time_pieChart.py File Reference

Namespaces

- analyze_correlation_upvote_reaction_time_pieChart

Functions

- def analyze_correlation_upvote_reaction_time_pieChart.initialize_mongo_db_parameters ()
- def analyze_correlation_upvote_reaction_time_pieChart.check_script_arguments ()
- def analyze_correlation_upvote_reaction_time_pieChart.calculate_time_difference (comment_time_stamp, answer_time_stamp_iama_host)
- def analyze_correlation_upvote_reaction_time_pieChart.check_if_comment_is_answer_from_thread_author (author_of_thread, comment_acutal_id, comments_cursor)
- def analyze_correlation_upvote_reaction_time_pieChart.check_if_comment_is_not_from_thread_author (author_of_thread, comment_author)
- def analyze_correlation_upvote_reaction_time_pieChart.check_if_comment_is_on_tier_1 (comment_parent_id)
- def analyze_correlation_upvote_reaction_time_pieChart.check_if_comment_is_a_question (given_string)
- def analyze_correlation_upvote_reaction_time_pieChart.calculate_comment_upvotes_and_response_time_by_host (id_of_thread, author_of_thread)
- def analyze_correlation_upvote_reaction_time_pieChart.generate_data_to_be_analyzed ()
- def analyze_correlation_upvote_reaction_time_pieChart.plot_the_generated_data ()

Variables

- string analyze_correlation_upvote_reaction_time_pieChart.argument_year = ""
- string analyze_correlation_upvote_reaction_time_pieChart.argument_tier_in_scope = ""
- string analyze_correlation_upvote_reaction_time_pieChart.argument_plot_time_unit = ""
- int analyze_correlation_upvote_reaction_time_pieChart.argument_plot_x_limiter = 0
- analyze_correlation_upvote_reaction_time_pieChart.mongo_DB_Client_Instance = None
- analyze_correlation_upvote_reaction_time_pieChart.mongo_DB_Threads_Instance = None
- analyze_correlation_upvote_reaction_time_pieChart.mongo_DB_Thread_Collection = None
- analyze_correlation_upvote_reaction_time_pieChart.mongo_DB_Comments_Instance = None
- list analyze_correlation_upvote_reaction_time_pieChart.list_To_Be_Plotted = []

analyze_thread_lifeSpan_n_average_commentTime_pieChart.py File Reference

Namespaces

- analyze_thread_lifeSpan_n_average_commentTime_pieChart

Functions

- def analyze_thread_lifeSpan_n_average_commentTime_pieChart.initialize_mongo_db_parameters ()
- def analyze_thread_lifeSpan_n_average_commentTime_pieChart.check_script_arguments ()
- def analyze_thread_lifeSpan_n_average_commentTime_pieChart.calculate_time_difference (id_of_thread, creation_date_of_thread)
- def analyze_thread_lifeSpan_n_average_commentTime_pieChart.generate_data_to_be_analyzed ()
- def analyze_thread_lifeSpan_n_average_commentTime_pieChart.prepare_dict_by_time_separation_for_life_span ()
- def analyze_thread_lifeSpan_n_average_commentTime_pieChart.prepare_dict_by_time_separation_for_comment_time ()
- def analyze_thread_lifeSpan_n_average_commentTime_pieChart.plot_the_generated_data ()

Variables

- string analyze_thread_lifeSpan_n_average_commentTime_pieChart.argument_year = ""
- string analyze_thread_lifeSpan_n_average_commentTime_pieChart.argument_calculation = ""
- string analyze_thread_lifeSpan_n_average_commentTime_pieChart.argument_plot_time_unit = ""
- analyze_thread_lifeSpan_n_average_commentTime_pieChart.mongo_DB_Client_Instance = None
- analyze_thread_lifeSpan_n_average_commentTime_pieChart.mongo_DB_Threads_Instance = None
- analyze_thread_lifeSpan_n_average_commentTime_pieChart.mongo_DB_Thread_Collection = None
- analyze_thread_lifeSpan_n_average_commentTime_pieChart.mongo_DB_Comments_Instance = None
- list analyze_thread_lifeSpan_n_average_commentTime_pieChart.list_To_Be_Plotted = []

analyze_tier_answered_percentage_pieChart.py File Reference

Namespaces

- analyze_tier_answered_percentage_pieChart

Functions

- def analyze_tier_answered_percentage_pieChart.initialize_mongo_db_parameters ()
- def analyze_tier_answered_percentage_pieChart.check_script_arguments ()
- def analyze_tier_answered_percentage_pieChart.calculate_percentage_distribution (amount_of_questions, amount_of_questions_answered)
- def analyze_tier_answered_percentage_pieChart.check_if_comment_is_not_from_thread_author (author_of_thread, comment_author)
- def analyze_tier_answered_percentage_pieChart.check_if_comment_is_answer_from_thread_author (author_of_thread, comment_actual_id, comments_cursor)
- def analyze_tier_answered_percentage_pieChart.check_if_comment_is_on_tier_1 (comment_parent_id)
- def analyze_tier_answered_percentage_pieChart.check_if_comment_is_a_question (given_string)
- def analyze_tier_answered_percentage_pieChart.amount_of_questions_answered_by_host (id_of_thread, author_of_thread)
- def analyze_tier_answered_percentage_pieChart.generate_data_to_be_analyzed ()
- def analyze_tier_answered_percentage_pieChart.plot_the_generated_data ()

Variables

- string analyze_tier_answered_percentage_pieChart.argument_year = ""
- string analyze_tier_answered_percentage_pieChart.argument_tier_in_scope = ""
- analyze_tier_answered_percentage_pieChart.mongo_DB_Client_Instance = None
- analyze_tier_answered_percentage_pieChart.mongo_DB_Threads_Instance = None
- analyze_tier_answered_percentage_pieChart.mongo_DB_Thread_Collection = None
- analyze_tier_answered_percentage_pieChart.mongo_DB_Comments_Instance = None
- list analyze_tier_answered_percentage_pieChart.list_To_Be_Plotted = []

analyze_tier_answered_time_pieChart.py File Reference

Namespaces

- analyze_tier_answered_time_pieChart

Functions

- def analyze_tier_answered_time_pieChart.initialize_mongo_db_parameters ()
- def analyze_tier_answered_time_pieChart.check_script_arguments ()
- def analyze_tier_answered_time_pieChart.calculate_time_difference (comment_time_stamp, answer_time_stamp_iama_host)
- def analyze_tier_answered_time_pieChart.check_if_comment_is_answer_from_thread_author (author_of_thread, comment_actual_id, comments_cursor)
- def analyze_tier_answered_time_pieChart.check_if_comment_is_not_from_thread_author (author_of_thread, comment_author)
- def analyze_tier_answered_time_pieChart.check_if_comment_is_on_tier_1 (comment_parent_id)
- def analyze_tier_answered_time_pieChart.check_if_comment_is_a_question (given_string)
- def analyze_tier_answered_time_pieChart.calculate_ar_mean_answer_time_for_questions (id_of_thread, author_of_thread)
- def analyze_tier_answered_time_pieChart.generate_data_to_be_analyzed ()
- def analyze_tier_answered_time_pieChart.plot_the_generated_data ()

Variables

- string analyze_tier_answered_time_pieChart.argument_year = ""
- string analyze_tier_answered_time_pieChart.argument_tier_in_scope = ""
- string analyze_tier_answered_time_pieChart.argument_plot_time_unit = ""
- analyze_tier_answered_time_pieChart.mongo_DB_Client_Instance = None
- analyze_tier_answered_time_pieChart.mongo_DB_Threads_Instance = None
- analyze_tier_answered_time_pieChart.mongo_DB_Thread_Collection = None
- analyze_tier_answered_time_pieChart.mongo_DB_Comments_Instance = None
- list analyze_tier_answered_time_pieChart.list_To_Be_Plotted = []

analyze_tier_question_distribution_pieChart.py File Reference

Namespaces

- analyze_tier_question_distribution_pieChart

Functions

- def analyze_tier_question_distribution_pieChart.initialize_mongo_db_parameters ()
- def analyze_tier_question_distribution_pieChart.check_script_arguments ()
- def analyze_tier_question_distribution_pieChart.check_if_comment_is_not_from_thread_author (author_of_thread, comment_author)
- def analyze_tier_question_distribution_pieChart.calculate_percentage_distribution (amount_of_tier_1_questions, amount_of_tier_x_questions)
- def analyze_tier_question_distribution_pieChart.check_if_comment_is_on_tier_1 (comment_parent_id)
- def analyze_tier_question_distribution_pieChart.check_if_comment_is_a_question (given_string)
- def analyze_tier_question_distribution_pieChart.amount_of_tier_1_questions_percentage (id_of_thread, author_of_thread)
- def analyze_tier_question_distribution_pieChart.generate_data_to_be_analyzed ()
- def analyze_tier_question_distribution_pieChart.plot_the_generated_data ()

Variables

- string analyze_tier_question_distribution_pieChart.argument_year = ""
- analyze_tier_question_distribution_pieChart.mongo_DB_Client_Instance = None
- analyze_tier_question_distribution_pieChart.mongo_DB_Threads_Instance = None
- analyze_tier_question_distribution_pieChart.mongo_DB_Thread_Collection = None
- analyze_tier_question_distribution_pieChart.mongo_DB_Comments_Instance = None
- list analyze_tier_question_distribution_pieChart.list_To_Be_Plotted = []

analyze_top100_pieChart.py File Reference

Namespaces

- analyze_top100_pieChart

Functions

- def analyze_top100_pieChart.initialize_mongo_db_parameters ()
- def analyze_top100_pieChart.check_script_arguments ()
- def analyze_top100_pieChart.calculate_time_difference (comment_time_stamp, answer_time_stamp_iama_host)
- def analyze_top100_pieChart.check_if_comment_is_answer_from_thread_author (author_of_thread, comment_acutal_id, comments_cursor)
- def analyze_top100_pieChart.check_if_comment_is_not_from_thread_author (author_of_thread, comment_author)
- def analyze_top100_pieChart.check_if_comment_is_a_question (given_string)
- def analyze_top100_pieChart.calculate_answered_question_upvote_correlation (id_of_thread, author_of_thread, thread_creation_date)
- def analyze_top100_pieChart.generate_data_to_be_analyzed ()
- def analyze_top100_pieChart.prepare_and_print_data_to_be_plotted ()
- def analyze_top100_pieChart.plot_generated_data (amount_of_questions_not_answered)

Variables

- string analyze_top100_pieChart.argument_year = ""
- analyze_top100_pieChart.argument_sorting = bool
- analyze_top100_pieChart.mongo_DB_Client_Instance = None
- analyze_top100_pieChart.mongo_DB_Threads_Instance = None
- analyze_top100_pieChart.mongo_DB_Thread_Collection = None
- analyze_top100_pieChart.mongo_DB_Comments_Instance = None
- list analyze_top100_pieChart.list_To_Be_Plotted = []

crawl_differences.py File Reference

Namespaces

- crawl_differences

Functions

- def crawl_differences.check_script_arguments ()
- def crawl_differences.initialize_mongo_db_parameters ()
- def crawl_differences.crawl_missing_collection_into_comments_database (name_of_missing_collection)
- def crawl_differences.check_if_collection_is_missing_in_comments_database ()
- def crawl_differences.crawl_missing_collection_into_threads_database (name_of_missing_collection)
- def crawl_differences.check_if_collection_is_missing_in_threads_database ()
- def crawl_differences.start_crawling_for_diffs ()

Variables

- crawl_differences.mongo_DB_Client_Instance = None
- crawl_differences.mongo_DB_Threads_Instance = None
- crawl_differences.mongo_DB_Thread_Collection = None
- crawl_differences.mongo_DB_Comments_Instance = None
- crawl_differences.mongo_DB_Comments_Collection = None
- string crawl_differences.argument_year_beginning = ""
- string crawl_differences.argument_year_end = ""
- string crawl_differences.argument_inverse_crawling = ""

crawl_threads_n_comments.py File Reference

Namespaces

- crawl_threads_n_comments

Functions

- def crawl_threads_n_comments.initialize_mongo_db_parameters ()
- def crawl_threads_n_comments.check_script_arguments ()
- def crawl_threads_n_comments.convert_argument_year_to_epoch (year)
- def crawl_threads_n_comments.crawl_data ()
- def crawl_threads_n_comments.crawl_threads ()
- def crawl_threads_n_comments.crawl_comments ()
- def crawl_threads_n_comments.check_if_coll_in_db_already_exists_up2date (submission)

Variables

- crawl_threads_n_comments.mongo_DB_Client_Instance = None
- crawl_threads_n_comments.reddit_Instance = None
- crawl_threads_n_comments.argument_crawl_type = None
- crawl_threads_n_comments.argument_year_beginning = None
- crawl_threads_n_comments.argument_year_end = None
- crawl_threads_n_comments.argument_hours_to_shift = None
- crawl_threads_n_comments.time_shift_difference

Index