2_process_mturk_results_example

March 13, 2023

1 mTurk_Results_analysis

This performs a basic analysis of mTurk results for any results file

[1]:
```python
import pandas as pd
import statistics
from collections import defaultdict
from tqdm import tqdm, trange
import matplotlib.pyplot as plt

myth_name = 'mosquitoes'
sample_size = '146'
date = '041822'

file_name = f'../data/samples/{myth_name}/
 ↪myth_{myth_name}_sample_{sample_size}_{date}-results.csv'

# Make sure to comment out the tested myth
topics = ['Answer.Prevention/Treatment - Disinfectants.on',
 'Answer.Prevention/Treatment - Home Remedy.on',
 'Answer.Prevention/Treatment - Technology.on',
 'Answer.Prevention/Treatment - Weather.on',
 'Answer.Prevention/Treatment - Masks.on',
 'Answer.Prevention/Treatment - Other.on',
 'Answer.Coronavirus Spread - Transmission.on',
 'Answer.Coronavirus Spread - Other.on',
 'Answer.Coronavirus Origins.on',
 'Answer.Coronavirus Testing.on',
 'Answer.Coronavirus Government Policy.on',
 'Answer.Coronavirus Humor/Sarcasm.on',
 'Answer.Coronavirus Coping Strategies.on',
 'Answer.Coronavirus New Cases.on']

if myth_name == 'disinfectants': topics.remove('Answer.Prevention/Treatment -␣
 ↪Disinfectants.on')
elif myth_name == 'home_remedies': topics.remove('Answer.Prevention/Treatment -␣
 ↪Home Remedy.on')
```

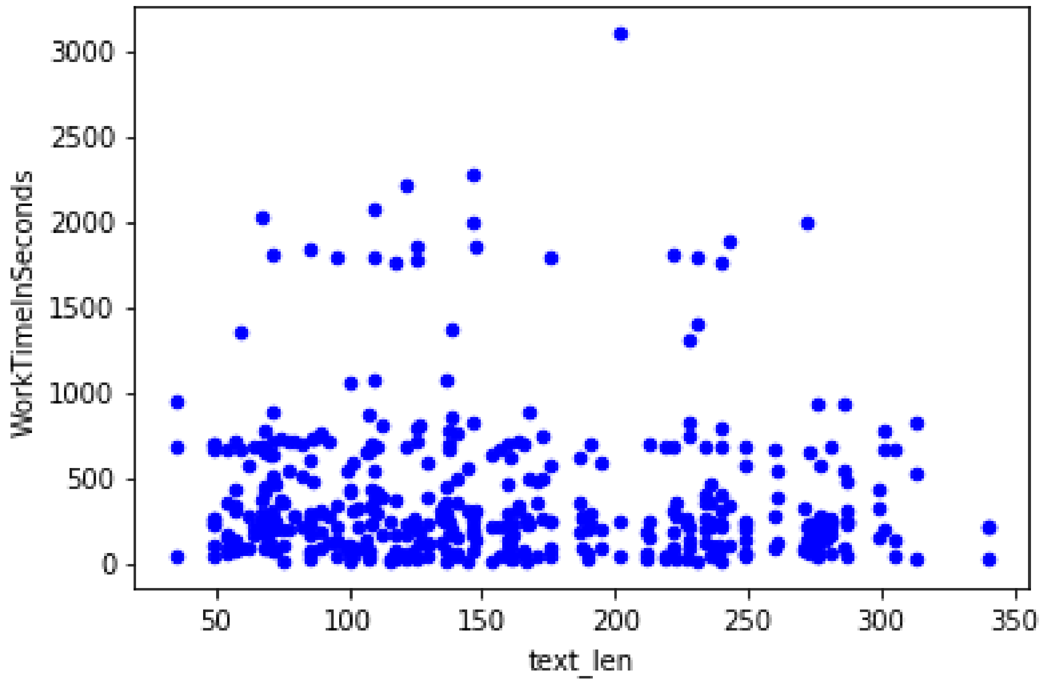
```
elif myth_name == 'weather': topics.remove('Answer.Prevention/Treatment -␣
 ↪Weather.on')
elif myth_name == 'origins': topics.remove('Answer.Coronavirus Origins.on')
elif myth_name == 'masks': topics.remove('Answer.Prevention/Treatment - Masks.
 ↪on')

df = pd.read_csv(file_name)
list(df)
```

```
[1]: ['HITId',
      'HITTypeId',
      'Title',
      'Description',
      'Keywords',
      'Reward',
      'CreationTime',
      'MaxAssignments',
      'RequesterAnnotation',
      'AssignmentDurationInSeconds',
      'AutoApprovalDelayInSeconds',
      'Expiration',
      'NumberOfSimilarHITs',
      'LifetimeInSeconds',
      'AssignmentId',
      'WorkerId',
      'AssignmentStatus',
      'AcceptTime',
      'SubmitTime',
      'AutoApprovalTime',
      'ApprovalTime',
      'RejectionTime',
      'RequesterFeedback',
      'WorkTimeInSeconds',
      'LifetimeApprovalRate',
      'Last30DaysApprovalRate',
      'Last7DaysApprovalRate',
      'Input.id',
      'Input.created_at',
      'Input.full_text_censored',
      'Input.myth',
      'Answer.Coronavirus Coping Strategies.on',
      'Answer.Coronavirus Government Policy.on',
      'Answer.Coronavirus Humor/Sarcasm.on',
      'Answer.Coronavirus New Cases.on',
      'Answer.Coronavirus Origins.on',
      'Answer.Coronavirus Spread - Other.on',
      'Answer.Coronavirus Testing.on',
      'Answer.Prevention/Treatment - Disinfectants.on',
      'Answer.Prevention/Treatment - Home Remedy.on',
      'Answer.Prevention/Treatment - Other.on',
      'Answer.Prevention/Treatment - Technology.on',
      'Answer.Prevention/Treatment - Weather.on',
      'Answer.feedback',
      'Answer.myth_no.on',
      'Answer.myth_supports_no.on',
      'Answer.myth_supports_unsure.on',
      'Answer.myth_supports_yes.on',
      'Answer.myth_unsure.on',
      'Answer.myth_yes.on',
      'Answer.topic',
      'Approve',
      'Reject']
```

[2]:
```
# Declare rater number
rater_num = 3
```

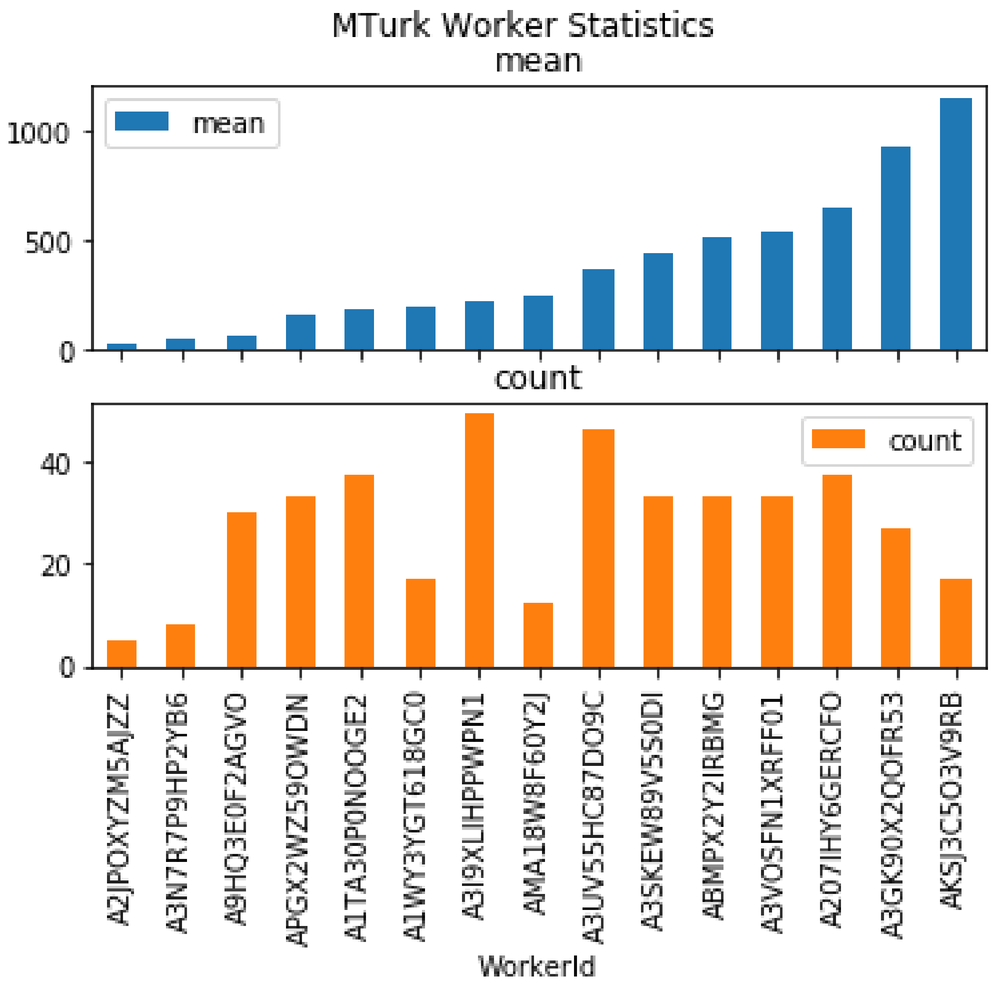
[3]:
```
for col in list(df):
    if col.startswith('Input.full_text'):
        full_text_col_name = col
        break
df['text_len'] = df[full_text_col_name].apply(len)
df.plot(kind='scatter',x='text_len',y='WorkTimeInSeconds',color='blue')
```

[3]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb49304c990>
```



[4]:
```python
# Group by WorkerID, and take the average time to complete to look
# for outliers that may have completed the tasks too fast.
df.groupby('WorkerId')['WorkTimeInSeconds']\
  .agg(['mean', 'count'])\
  .sort_values(by='mean')\
  .plot(subplots=True, kind='bar', title='MTurk Worker Statistics')
```

[4]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fb491e6e250>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fb491e3a810>],
      dtype=object)
```



[5]:
```python
# The radio buttons from the MTurk form store their boolean result in a
# single column; we will combine these into their respective questions.
```

```
null_task_ids = set()

def merge_radios(row, topic):
    """
    To be used in an `apply` method to combine boolean radio buttons into a
    single column
    """
    if row['Answer.{}_yes.on'.format(topic)]:
        return 'yes'
    elif row['Answer.{}_no.on'.format(topic)]:
        return 'no'
    elif row['Answer.{}_unsure.on'.format(topic)]:
        return 'unsure'
    elif 'Answer.{}_broken_links.on'.format(topic) in list(df) and row['Answer.
 ↪{}_broken_links.on'.format(topic)]:
        return 'broken_links'
    else:
        if topic == 'myth_supports':
            # Because this question is conditonal, some tasks might not have␣
 ↪this field
            # Assume they are NO
            return 'no'

        raise ValueError("The chosen choice is not defined.") # If the worker␣
 ↪didn't choose any choices
        # print(row.AssignmentId)
        # null_task_ids.add(row.HITId)
        # return None

df['is_myth'] = df.apply (lambda row: merge_radios(row, 'myth'), axis=1)
df['is_myth_supports'] = df.apply (lambda row: merge_radios(row,␣
 ↪'myth_supports'), axis=1)

print('There are {} tasks ids with null values'.format(len(null_task_ids)))
```

```
There are 0 tasks ids with null values
```

[6]:
```
# Drop tasks with null
df = df[~df['HITId'].isin(list(null_task_ids))]
```

[7]:
```
# Drop all tasks performed by rejected workers
rejected_workers = ["AAXYYH9MI3PJM"]

df = df[~df['WorkerId'].isin(rejected_workers)]

df.shape
```

[7]: (417, 56)

[8]:
```
# is_myth Answer Statistics

def check_agree_on(col):
    # Group the results by Task ID and their answer to the gun_violence␣
↪question, then
    # count the number of records in those groups. This determines how many␣
↪answers
    # there were per choice, per task. Rename the count column.
    df_gun_violence_counts = df.groupby(['HITId', col]).size().to_frame().
↪reset_index()
    df_gun_violence_counts.rename(columns={0: 'count'}, inplace=True)

    # All three workers answered 'unsure'
    unsure = df_gun_violence_counts[col] == 'unsure'
    three = df_gun_violence_counts['count'] >= 2
    num_all_unsure = len(df_gun_violence_counts[unsure & three])

    # All three workers answered 'yes'
    yes = df_gun_violence_counts[col] == 'yes'
    num_all_yes = len(df_gun_violence_counts[yes & three])

    # All three workers answered 'no'
    no = df_gun_violence_counts[col] == 'no'
    num_all_no = len(df_gun_violence_counts[no & three])

    print('Agreed on "yes": {}\nAgreed on "No": {}\nAgreed on "Unsure": {}'.
↪format(num_all_yes, num_all_no, num_all_unsure))

    # There was no majority, answers were 'yes', 'no', and 'unsure'
    df_gun_violence_count_size = df_gun_violence_counts.groupby('HITId').size().
↪to_frame().rename(columns={0: 'count'})
    # df_gun_violence_count_size[df_gun_violence_count_size['count'] == 3]
```

[9]:
```
check_agree_on(col="is_myth")
```

```
Agreed on "yes": 88
Agreed on "No": 51
Agreed on "Unsure": 0
```

[10]:
```
check_agree_on(col="is_myth_supports")
```

```
Agreed on "yes": 15
Agreed on "No": 88
Agreed on "Unsure": 32
```

1.1 Check data validity

[11]:
```
# These columns must not be None
answer_cols = [ e for e in list(df) if e.startswith("Answers.") ]
for col in answer_cols:

    for v in df[col]:
        if v is None:
            raise ValueError('None exists in {}'.format(col))
```

[12]:
```
# Check that each task has three rated values
for v in df.groupby('HITId').size():
    if v != rater_num:
        raise ValueError("There is a task with raters not equal to {}, found {}.
 ↪".format(rater_num, v))
print('Every task has {} workers for each'.format(rater_num))
```

```
Every task has 3 workers for each
```

2 Agreement Computation

2.1 1. Task-based agreement

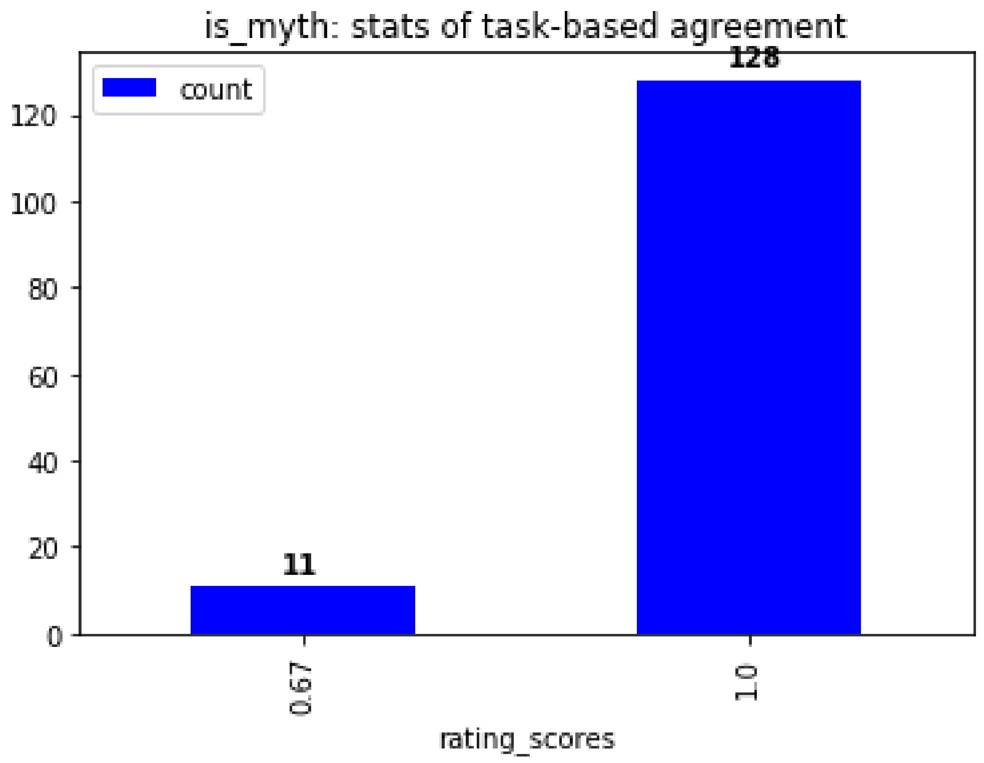
[13]:
```
def get_task_based_rating(df, answer_col, rater_num=5):
    """
    Task-based rating score computation

    Args:
        df:
            A dataFrame with columns ['WorkerId', 'HITId', 'is_parenting']␣
 ↪ordered by ['HITId', 'WorkerId']
        answer_col:
            Column name to compute rating scores
        rater_num:
            Number of raters for each task

    Return:
        A list of task-based rating scores
    """

    rating_scores = []
    for i in range(0, df.shape[0], rater_num):
        if len(set(df.iloc[i:i + rater_num]['HITId'])) != 1:
            raise ValueError('Each task must contains {} rates, wrong at {}'.
 ↪format(rater_num, i))
        answers = df.iloc[i:i + rater_num][answer_col].tolist()
        # Get number of majority
        majority_num = max(df.iloc[i:i + rater_num].groupby(answer_col).size())
        # Store rating score
        rating_scores.append(round(float(majority_num / rater_num), 2))

    return rating_scores
```

[14]:
```
# Gun_violence
def check_task_based_score(col):
    df_task_based = df[['WorkerId', 'HITId', col]].sort_values(by=['HITId',␣
 ↪'WorkerId'])

    # DataFrame for rating scores
    df_task_based_rates = df_task_based.drop_duplicates(subset=['HITId'])
    df_task_based_rates.drop(['WorkerId', col], axis=1, inplace=True)

    # Compute scores
    rating_scores = get_task_based_rating(df_task_based, answer_col=col,␣
 ↪rater_num=rater_num)
    df_task_based_rates = df_task_based_rates.
 ↪assign(rating_scores=rating_scores)

    # Sort and check statistics
    df_task_based_rates.sort_values(by=['rating_scores'], inplace=True)

    # Check stats
    stats_task_based_gun_violence = df_task_based_rates.
 ↪groupby('rating_scores').size().to_frame().reset_index().rename(columns={0:␣
 ↪'count'})
    ax = stats_task_based_gun_violence.plot(kind='bar', x='rating_scores',␣
 ↪y='count', color='blue', \
                    title='{}: stats of task-based agreement'.format(col))

    for i, v in enumerate(stats_task_based_gun_violence['count']):
        ax.text(i-0.05, v+5, str(v), va='center', fontsize=10,␣
 ↪fontweight='bold')

    df_gun_violence_task_based_rates = df_task_based_rates.
 ↪sort_values(by=['rating_scores'], ascending=False)
    print(df_gun_violence_task_based_rates.head(10))

    only_high_task_based_tweet_ids =␣
 ↪df_gun_violence_task_based_rates[df_gun_violence_task_based_rates['rating_scores']␣
 ↪>= 0.8]['HITId'].values
    print("There are {} tweets with high task-based scores".
 ↪format(only_high_task_based_tweet_ids.shape[0]))
```

```
    avg_score = statistics.
↪mean(df_gun_violence_task_based_rates['rating_scores'].values)
    print("Avg task-based score for {} = {}".format(col, avg_score))
```

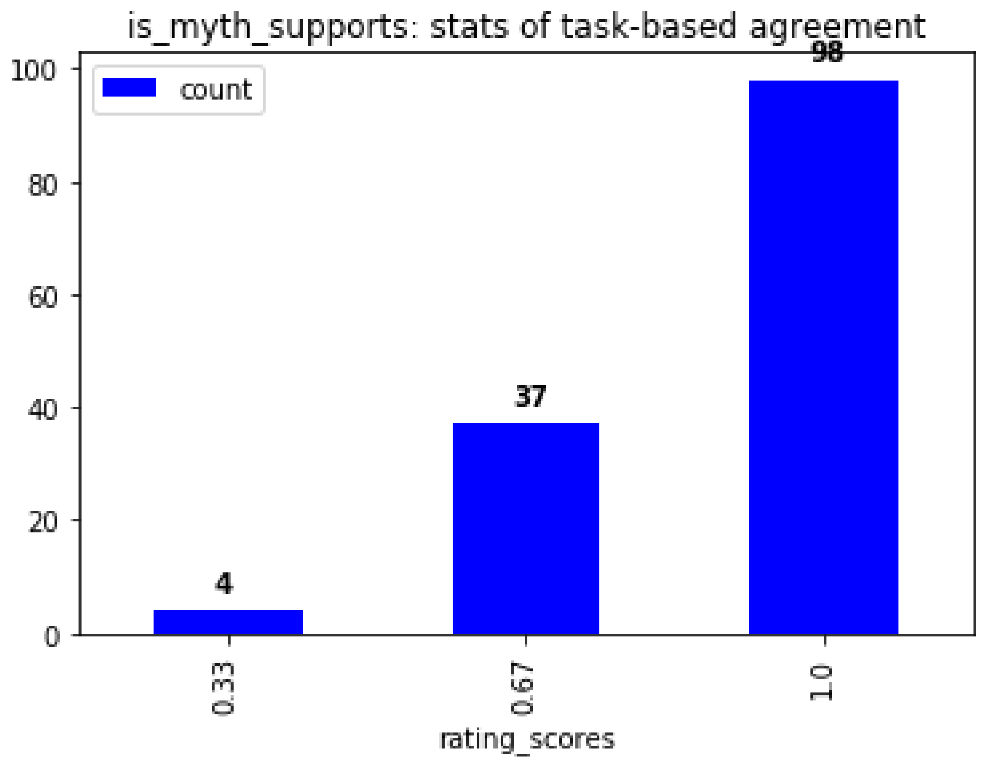
[15]:
```
check_task_based_score(col='is_myth')
```

```
/home/app59/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py:4315:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,

                              HITId  rating_scores
324  3VLL1PIENS87UPL3RGSPKVX5MZ6OZ9            1.0
96   32LAQ1JNTB96LIW1HAFGXDODJP8TUB            1.0
222  33QQ60S6AU2559ZAJ8R6D2M8MEBU00            1.0
84   33CLA8O0MKV4MLMJIWFMBYMSLHWRFX            1.0
363  338431Z1FNZYSYZCHZYCYSV6DINR0J            1.0
168  335VBRURDLK6C7Q1T4VU2V1Y3UBE9F            1.0
69   334ZEL5JX8ZATDZPWOYHO202ZK60S4            1.0
183  32W3UF2EZQ5QI30BGIT8VZQFWQKC4O            1.0
390  31SIZS5W5BZ1DACSWCQGNXQ6SJ5RQR            1.0
225  3H1C3QRA0338A5X4505L4L269FEECW            1.0
There are 128 tweets with high task-based scores
Avg task-based score for is_myth = 0.9738848920863309
```



```
[16]: check_task_based_score(col='is_myth_supports')
```

```
/home/app59/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py:4315:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,

                              HITId  rating_scores
117  3UZUVSO3P9FLJR9VIDZZ7H7I3TIMEM            1.0
21   34O39PNDK8SE94A3DUTQS8V76J4RBF            1.0
204  35U0MRQMULRQL5GTQJ3MZPK9Y000V9            1.0
351  35JDMRECC6TLRHD97A14UCNL776EG6            1.0
258  359AP8GAGI4N1QIUIGRFYBAPNSAC78            1.0
276  3511RHPADXYQMDWF47IK6BRCIDIRL2            1.0
306  38LRF35D5NG1M1Y72V9PF0KFPQVU3V            1.0
312  34XASH8KLS6PGIYLRRFA1T6MH0KMP7            1.0
210  33W1NHWFYJ5N7HQBNXE0E53SL7NTZ0            1.0
57   30Y6N4AHYRG7O18NRQKWNYJ794GRDM            1.0
There are 98 tweets with high task-based scores
Avg task-based score for is_myth_supports = 0.8928776978417267
```



2.2 2. Worker-based agreement

Compute the score by comparing each task of each worker to the other. If an answer of a worker is equal to its majority of the corresponding task, then the number of correct answers of the work increases by one. The avg_score of each worker is computed by dividing the number of correct answers by total number of tasks that the worker have done.

[17]:
```
def get_worker_based_rating(df, answer_col):
    """
    Worker-based rating score computation

    Args:
        df:
            A dataFrame with columns ['WorkerId', 'HITId', 'is_parenting']␣
↪ordered by ['WorkerId', 'HITId']
        answer_col:
            Column name to compute rating scores

    Return:
        A list of worker-based rating scores
    """
    df = df.reset_index(drop=True)
    # Variables for rating score computation
    current_woker_id = ""
    total_task_num = 0
    correct_answer_num = 0

    rating_scores = []
    total_task_nums = []
    for idx, row in df.iterrows():
        worker_id = row['WorkerId']
        task_id = row['HITId']
        answer = row[answer_col]

        # Get next worker ID
        if worker_id != current_woker_id:
            current_woker_id = worker_id

            if idx > 0:
                # Compute rating scores
                rating_scores.append(correct_answer_num / total_task_num)
                total_task_nums.append(total_task_num)

                # Reset variables
                total_task_num = 0
                correct_answer_num = 0

        # Check whether the answer is the same as majority answer
        df_answer_counts = df[df['HITId'] == task_id].groupby(answer_col).
↪size().to_frame().rename(columns={0: 'count'}).sort_values(by=['count'],␣
↪ascending=False).reset_index()
        majority_count = df_answer_counts.iloc[0]['count']

        try:
            answer_count = df_answer_counts[df_answer_counts[answer_col] ==␣
↪answer].iloc[0]['count']
        except Exception as e:
            print(e)
            print(df[df['HITId'] == task_id])

        if answer_count == majority_count and majority_count > 1:
            is_majority = True
        else:
            is_majority = False

        # Counting if the answer is the same as the majority vote
        if is_majority:
            correct_answer_num += 1
        total_task_num += 1
        # Last task
        if idx + 1 > max(df.index):
            # Compute rating scores
            rating_scores.append(correct_answer_num / total_task_num)
            total_task_nums.append(total_task_num)

    # print(len(set(df['WorkerId'].values)))
    # print("Rating score: " + str(len(rating_scores)))
    # print("Total task number: " + str(len(total_task_nums)))

    return rating_scores, total_task_nums
```

[18]:

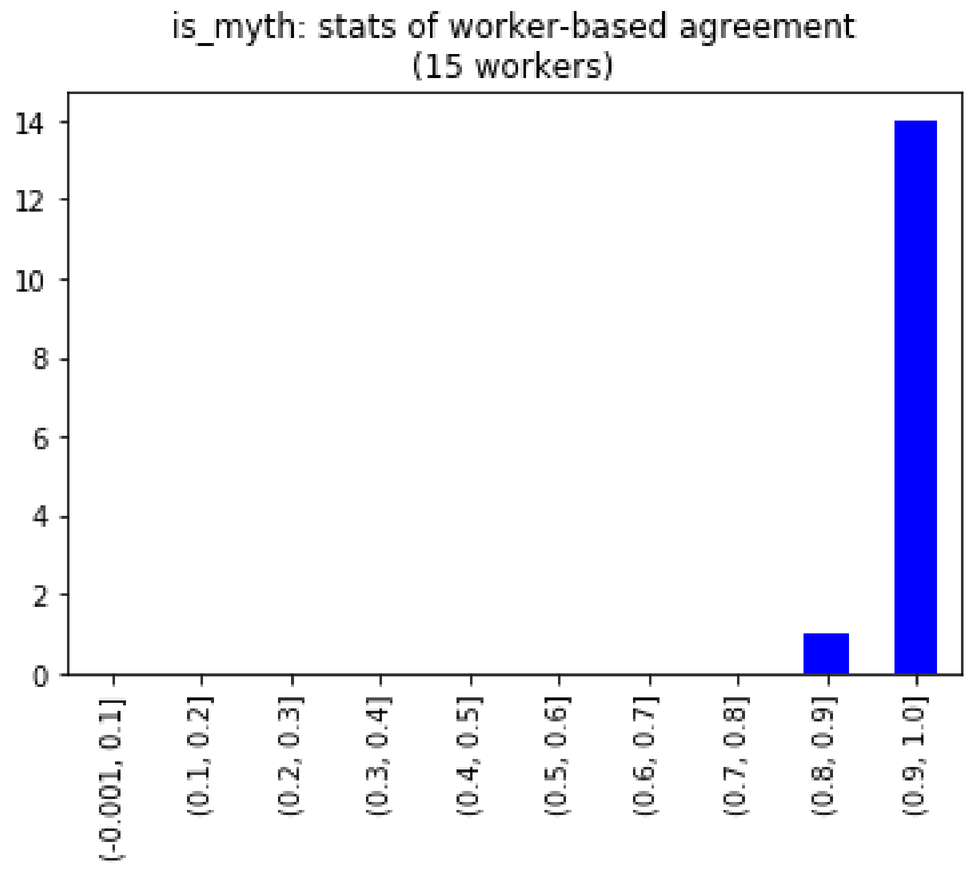
```
# Gun violence
def check_worker_based_score(col):
    df_worker_based = df[['WorkerId', 'HITId', col]].
 ↪sort_values(by=['WorkerId', 'HITId'])

    df_worker_based_rates = df_worker_based.drop_duplicates(subset=['WorkerId'])
    df_worker_based_rates.drop(['HITId', col], axis=1, inplace=True)

    # # Compute scores
    rating_scores, total_task_nums = get_worker_based_rating(df_worker_based,␣
 ↪answer_col=col)
    df_worker_based_rates = df_worker_based_rates.
 ↪assign(rating_scores=rating_scores)
    df_worker_based_rates = df_worker_based_rates.
 ↪assign(total_task_nums=total_task_nums)

    # Filter workers that have done only one task
    df_worker_based_rates =␣
 ↪df_worker_based_rates[df_worker_based_rates['total_task_nums'] > 1]

    # Check stats
    stats_worker_based_gun_violence = pd.
 ↪cut(df_worker_based_rates['rating_scores'].values, \
                        bins=[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,␣
 ↪1.0], \
                        include_lowest=True).value_counts()
    stats_worker_based_gun_violence.plot(kind='bar', x='rating_scores',␣
 ↪y='count', color='blue', \
                    title='{}: stats of worker-based agreement\n({} workers)'.
 ↪format(col, df_worker_based_rates.shape[0]))

    df_gun_violence_worker_based_rates = df_worker_based_rates.
 ↪sort_values(by=['rating_scores'], ascending=False)
```

```
    avg_score = statistics.
  ↪mean(df_gun_violence_worker_based_rates['rating_scores'].values)

    print("Avg worker-based score for {} = {}".format(col, avg_score))

    done_enough_gun_violence_worker_ids =␣
  ↪df_gun_violence_worker_based_rates[(df_gun_violence_worker_based_rates['rating_scores']␣
  ↪>= 0.5)]\
['WorkerId'].values
    print("There are {} good workers".
  ↪format(len(done_enough_gun_violence_worker_ids)))

    bad_gun_violence_worker_ids =␣
  ↪df_gun_violence_worker_based_rates[(df_gun_violence_worker_based_rates['rating_scores']␣
  ↪< 0.5)]\
['WorkerId'].values
    print("There are {} bad workers".format(len(bad_gun_violence_worker_ids)))
    print(bad_gun_violence_worker_ids)
```

[19]:
```
check_worker_based_score("is_myth")
```

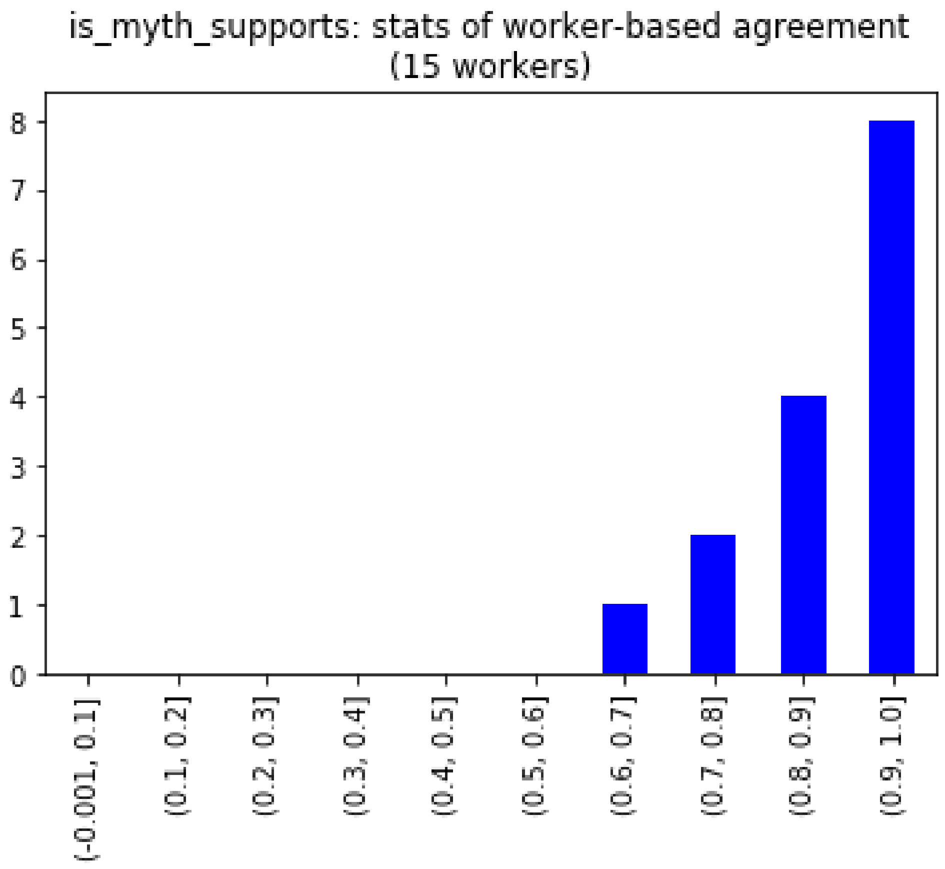
```
/home/app59/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py:4315:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

```
Avg worker-based score for is_myth = 0.9731528685255393
There are 15 good workers
There are 0 bad workers
[]
```



```
[20]: check_worker_based_score("is_myth_supports")
```

```
/home/app59/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py:4315:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,

Avg worker-based score for is_myth_supports = 0.8822651441930458
There are 15 good workers
There are 0 bad workers
[]
```



2.3 3 Alpha agreement score

[21]:
```
from nltk.metrics import agreement

data = df[['WorkerId', 'HITId', 'is_myth']].values

data = [ e for e in data if e[2] != 'unsure' ]

rating = agreement.AnnotationTask(data=data)

#print("kappa " + str(rating.kappa()))
#print("fleiss " + str(rating.multi_kappa()))
print("alpha " + str(rating.alpha()))
#print("scotts " + str(rating.pi()))
```

```
alpha 0.9275012569130217
```

[22]:
```
data = df[['WorkerId', 'HITId', 'is_myth_supports']].values
data = [ e for e in data if e[2] != 'unsure' ]

rating = agreement.AnnotationTask(data=data)

#print("kappa " + str(rating.kappa()))
#print("fleiss " + str(rating.multi_kappa()))
print("alpha " + str(rating.alpha()))
#print("scotts " + str(rating.pi()))
```

```
alpha 0.7145038167938931
```

[23]:
```
worker_ids = list(set(df['WorkerId']))
print('There are {} raters'.format(len(worker_ids)))

task_ids = list(set(df['HITId']))
print('There are {} tweets'.format(len(task_ids)))
```

```
There are 15 raters
There are 139 tweets
```

[24]:
```
df[['is_myth']].groupby('is_myth').size()
```

[24]:
```
is_myth
no         153
unsure       4
yes        260
dtype: int64
```

[25]:
```
df[['is_myth_supports']].groupby('is_myth_supports').size()
```

[25]:
```
is_myth_supports
no         263
unsure      90
yes         64
dtype: int64
```