TITLE Evaluate Jeopardy question/answer for common words and questions for word repeats

Jeopardy 1984 - 2010 over 200,000 questions

DESCRIPTION Over 200,000 questions from Jeopardy shows aired 1984 - 2010 were evaluated for 1) word matches between question and answer, 2) word repetition in questions over time. The evaluation was performed as follows:

- clean string data using regex, convert to lower case
- convert any float questions or answers to string
- convert Value from string to numeric
- create function to check if question and answer words match
- evaluate frequency word repetition in questions
- create new column for high value questions with 0 or 1 value for each row
- count frequency of words found in high value or low value questions
- chi square analysis to see if observed frequency count statistically different from expected.

In [1]:
```
import pandas as pd
```

In [3]:
```
jeopardy = pd.read_csv(r"C:\Users\drrdm\Data Quest Guided Projects\12th Guided Project - Jeopardy\JEOPARDY_CSV.csv")
jeopardy.head(1)
```

Out[3]:

	Show Number	Air Date	Round	Category	Value	Question	Answer
0	4680	2004-12-31	Jeopardy!	HISTORY	$200	For the last 8 years of his life, Galileo was ...	Copernicus

In [5]:
```
column_dict = {'Show Number': 'Show_Number', ' Air Date': 'Air_Date',' Round': 'Round', ' Category': 'Category',
               ' Value':'Value',' Question': 'Question', ' Answer': 'Answer' }

jeopardy.rename(axis = 'columns', mapper = column_dict, inplace = True)
```

In [7]:
```
import re

def float_text(string):  #change float values to string if needed
    if type(string) == float:
        my_string = str(string)
    else:
        my_string = string.lower()
        my_string = re.sub(r'[^\w\s]+', '',my_string) #regex sub any char except digit,letter,whitespace w/whitespace
    return my_string
```

In [8]:
```
jeopardy['clean_question'] = jeopardy['Question'].apply(float_text)
jeopardy['clean_answer'] = jeopardy['Answer'].apply(float_text)
jeopardy.head(2)
```

Out[8]:

	Show_Number	Air_Date	Round	Category	Value	Question	Answer	clean_question
0	4680	2004-12-31	Jeopardy!	HISTORY	$200	For the last 8 years of his life, Galileo was ...	Copernicus	for the last 8 years of his life galileo was u...
1	4680	2004-12-31	Jeopardy!	ESPN's TOP 10 ALL-TIME ATHLETES	$200	No. 2: 1912 Olympian; football star at Carlisl...	Jim Thorpe	no 2 1912 olympian football star at carlisle i...

In [9]:
```
#convert a $ value string to a number (Got this code from Dataquest)
def string_numeric(string):
    string = re.sub(r'[^\w]+', '', string) #regex sub any char except digit,letter whitespace
    try:
        string = int(string) #try converting string to numeric value
    except:
        string = 0
    return string
```

In [10]:
```
jeopardy['clean_value'] = jeopardy['Value'].apply(string_numeric)
jeopardy.head(2)
```

Out[10]:

	Show_Number	Air_Date	Round	Category	Value	Question	Answer	clean_question
0	4680	2004-12-31	Jeopardy!	HISTORY	$200	For the last 8 years of his life, Galileo was ...	Copernicus	for the last 8 years of his life galileo was u...
1	4680	2004-12-31	Jeopardy!	ESPN's TOP 10 ALL-TIME ATHLETES	$200	No. 2: 1912 Olympian; football star at Carlisl...	Jim Thorpe	no 2 1912 olympian football star at carlisle i...

In [11]:
```
#remove old Value, Question and Answer columns
jeopardy.drop(columns = ['Value', 'Question', 'Answer'], inplace = True)
```

In [12]:
```
#convert Air_Date column to dt
jeopardy['Air_Date'] = pd.to_datetime(jeopardy['Air_Date'])
```

In [14]:
```
#check df datatypes
jeopardy.dtypes
```

Out[14]:
```
Show_Number                int64
Air_Date          datetime64[ns]
Round                     object
Category                  object
clean_question            object
clean_answer              object
clean_value                int64
dtype: object
```

In [17]:
```
#function to see if answer words match question words
def split_row(row):
    split_answer = row['clean_answer'].split(' ')
    split_question = row['clean_question'].split(' ')
    match_count = 0
    if 'the' in split_answer:
        split_answer.remove('the') #remove 'the' from the split answer words,
 not a meaningful word match w/split_question
    if len(split_answer) == 0:
        return 0
    for i in split_answer:
        if i in split_question:
            match_count +=1
    return match_count/len(split_answer) #the proportion of question words fou
nd in the answer
```

In [18]:
```
jeopardy['answer_in_question'] = jeopardy.apply(split_row, axis = 1) #new colu
mn q words in answer/total words in answer
jeopardy['answer_in_question'].mean() #average q words in answer/total words i
n answer
```

Out[18]:
```
0.05934282490603389
```

Word(s) in the question match word(s) in the answer for only 6% of the words in the answers. So relying on the words of the question to determine the answer is not likely to be very helpful.

In [20]:
```
#how often are new questions repeats of old ones? (Needed help from Dataquest
 with the list comprehension )

question_overlap = []
terms_used = set()
jeopardy = jeopardy.sort_values(by = "Air_Date") #the loop needs to start at t
he beginning air time

for index,row in jeopardy.iterrows():  #yields tuple of index, row Series
    split_question = row['clean_question'].split(" ") #split question into wor
ds

    #iterate through words, replace split_question with words >5 characters (f
ilter out common words)
    split_question = [word for word in split_question if len(word) > 5]
    match_count = 0
    for word in split_question:
        if word in terms_used: #if split question word used by previous questi
on, add 1 to terms used
            match_count +=1
    for word in split_question: #add word to terms_used if not already in the
 set
        terms_used.add(word)
    if len(split_question) > 0: #eliminate divide by zero
        match_count = match_count / len(split_question) #proportion of questio
n words used in previous questions
    question_overlap.append(match_count)
```

In [23]:
```
jeopardy["question_overlap"] = question_overlap
print('Avg % of >5 ltr questions words used in previous questions: {0:5.0%}'.f
ormat(jeopardy["question_overlap"].mean()))
```

```
Avg % of >5 ltr questions words used in previous questions:   87%
```

CONCLUSION: Studying previous questions may help with winning at Jeopardy, because on average 87% of > 5 ltr words in questions were used in previous questions.

In [31]:
```
#function to determine if a question was a high $ value
def high_value(row):
    if row['clean_value'] > 800:
        value = 1
    else:
        value = 0
    return value
```

In [32]:
```
jeopardy['high_value'] = jeopardy.apply(high_value, axis = 1 )
```

In [57]:
```
# next code blocks are to loop through terms_used and find out the count for u
sing the term
# in a high value question vs. use in a low value question for the first 5 ter
ms

def assign(word):
    low_count = 0
    high_count = 0
    for index,row in jeopardy.iterrows():
        split_question = row['clean_question'].split(' ')  my version
        if word in split_question:
            if row['high_value'] == 1:
                high_count += 1
            else:
                low_count += 1

    return high_count, low_count
```

In [65]:
```
#Note: this takes a while to run
observed_expected = []
terms_used = list(terms_used) #convert terms_used set to a list
comparison_terms = terms_used[:5] #take the first 5 terms used
for terms in comparison_terms:
    result = assign(term)
    observed_expected.append(result)
```

In [66]:
```
observed_expected
```

Out[66]:
```
[(1, 0), (1, 0), (1, 0), (1, 0), (1, 0)]
```

In [72]:
```
low_value_count = jeopardy['high_value'].value_counts().loc[0]
high_value_count = jeopardy['high_value'].value_counts().loc[1]
```

In [77]:
```
from  scipy.stats import chisquare
chi_squared = []

for elements in observed_expected:
    total =  elements[0] + elements[1]
    total_prop = total/jeopardy.shape[0]
    exp_high_value = total_prop * high_value_count
    exp_low_value = total_prop *low_value_count

    obs_high_value = observed_expected[0]
    obs_low_value = observed_expected[1]

    result = scipy.stats.chisquare([obs_high_value, obs_low_value], [exp_high_
value, exp_low_value])
    chi_squared.append(result)


chi_squared
```

Out[77]:
```
[Power_divergenceResult(statistic=array([3.6298769 , 1.43371595]), pvalue=arr
ay([0.05675101, 0.23115895])),
 Power_divergenceResult(statistic=array([3.6298769 , 1.43371595]), pvalue=arr
ay([0.05675101, 0.23115895])),
 Power_divergenceResult(statistic=array([3.6298769 , 1.43371595]), pvalue=arr
ay([0.05675101, 0.23115895])),
 Power_divergenceResult(statistic=array([3.6298769 , 1.43371595]), pvalue=arr
ay([0.05675101, 0.23115895])),
 Power_divergenceResult(statistic=array([3.6298769 , 1.43371595]), pvalue=arr
ay([0.05675101, 0.23115895]))]
```

Dataquest had this chisquared analysis for the Guided Project. This is really of no value: the expected and observed values are too small to be meaningful. Would to use terms from `terms_used` that had larger values for expected and observed.

In [ ]:
```

```