# Winning Jeopardy

Jeopardy is a popular TV show in the US where participants answer questions to win money. In this project, we will work with the dataset of Jeopardy questions to figure out some patterns in the questions that could help win the game.


The dataset can be found at this link (https://www.reddit.com/r/datasets/comments/1uyd0t/200000_jeopardy_questions_in_a_json_file/). Each row in the dataset corresponds to one questions asked on a single episode. Description of a few columns:

- Show Number - the Jeopardy episode number of the show this question was in.
- Air Date - the date the episode aired.
- Round - the round of Jeopardy that the question was asked in. Jeopardy has several rounds as each episode progresses.
- Category - the category of the question.
- Value - the number of dollars answering the question correctly is worth.
- Question - the text of the question.
- Answer - the text of the answer.

In [77]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk import bigrams
from scipy.stats import chisquare,chi2_contingency
```

In [78]:
```
import pandas as pd
```

In [79]:
```
jeopardy= pd.read_csv("jeopardy.csv")
```

In [80]:
```
jeopardy.head()
```

Out[80]:

| | 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 |
| **1** | 4680 | 2004-12-31 | Jeopardy! | ESPN's TOP 10 ALL-TIME ATHLETES | $200 | No. 2: 1912 Olympian; football star at Carlisl... | Jim Thorpe |
| **2** | 4680 | 2004-12-31 | Jeopardy! | EVERYBODY TALKS ABOUT IT... | $200 | The city of Yuma in this state has a record av... | Arizona |
| **3** | 4680 | 2004-12-31 | Jeopardy! | THE COMPANY LINE | $200 | In 1963, live on "The Art Linkletter Show", th... | McDonald's |
| **4** | 4680 | 2004-12-31 | Jeopardy! | EPITAPHS & TRIBUTES | $200 | Signer of the Dec. of Indep., framer of the Co... | John Adams |

In [81]:
```
jeopardy.columns
```

Out[81]:
```
Index(['Show Number', ' Air Date', ' Round', ' Category', ' Value',
       ' Question', ' Answer'],
      dtype='object')
```

In [82]:
```
cols = jeopardy.columns
jeopardy.columns = cols.str.strip().str.lower().str.replace(" ","_")
jeopardy.head(3)
```

Out[82]:

| | 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 |
| **1** | 4680 | 2004-12-31 | Jeopardy! | ESPN's TOP 10 ALL-TIME ATHLETES | $200 | No. 2: 1912 Olympian; football star at Carlisl... | Jim Thorpe |
| **2** | 4680 | 2004-12-31 | Jeopardy! | EVERYBODY TALKS ABOUT IT... | $200 | The city of Yuma in this state has a record av... | Arizona |

In [83]:
```
jeopardy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19999 entries, 0 to 19998
Data columns (total 7 columns):
show_number    19999 non-null int64
air_date       19999 non-null object
round          19999 non-null object
category       19999 non-null object
value          19999 non-null object
question       19999 non-null object
answer         19999 non-null object
dtypes: int64(1), object(6)
memory usage: 1.1+ MB
```

# Normalize

Let us normalize the questions and ansewrs columns to remove punctuations and convert all words to lower case. Some questions also contains html tags, so we will remove them as well. This way we can easily use the words for comparision later on.

In [84]:
```
jeopardy["question"].head()
```

Out[84]:
```
0    For the last 8 years of his life, Galileo was ...
1    No. 2: 1912 Olympian; football star at Carlisl...
2    The city of Yuma in this state has a record av...
3    In 1963, live on "The Art Linkletter Show", th...
4    Signer of the Dec. of Indep., framer of the Co...
Name: question, dtype: object
```

In [85]:
```
import re
def normalize_text(text):
    text = text.lower()           #Convert str to lowercase
    text = re.sub("[^a-zA-Z0-9/s]"," ",text)
    text = re.sub("/s+"," ",text)   #dấu câu
    return text

jeopardy["clean_question"] = jeopardy["question"].apply(normalize_text)
jeopardy["clean_answer"] = jeopardy["answer"].apply(normalize_text)
```

In [86]:
```
jeopardy["clean_question"].head(5)
```

Out[86]:
```
0    for the last 8 years of his life  galileo was ...
1    no  2  1912 olympian  football star at carlisl...
2    the city of yuma in this state has a record av...
3    in 1963  live on  the art linkletter show   th...
4    signer of the dec  of indep   framer of the co...
Name: clean_question, dtype: object
```

## Normalize value

The value column must be numeric and the air_date a 'datetime' object rather than a string. So let us normalize these as well.

In [87]:
```
import re
def normalize_value(value):
    value = re.sub("[^a-zA-Z0-9/s]", "",value)
    if value != 'None':
        value = value
    else:
        value = 0
    value = int(value)
    return value
```

In [88]:
```
jeopardy["value"] = jeopardy["value"].apply(normalize_value)
jeopardy["value"].head()
```

Out[88]:
```
0    200
1    200
2    200
3    200
4    200
Name: value, dtype: int64
```

## Normalize Date

In [89]:
```
jeopardy["air_date"] = pd.to_datetime(jeopardy["air_date"])
```

In [90]:
```
jeopardy["air_date"].head()
```

Out[90]:
```
0   2004-12-31
1   2004-12-31
2   2004-12-31
3   2004-12-31
4   2004-12-31
Name: air_date, dtype: datetime64[ns]
```

## Answers in Questions

It would be helpful to figure two things when trying to analyze the game inorder to win it.

- How often the answer is deducible from the question.
- How often new questions are repeats of older questions.

For the first question, we will see how many times on average do the answers appear or are mentioned of in the questions. For every answer we will check the corresponding questions to see if the answer or any part of the answer was in it. We will remove the Stopwords from the questions and answers as Stopwords are very common and can be misleading in our case.

The basic idea is to find on average how many times do the questions contain the answers, so we will, for each answer check the corresponding questions and find the proportion of answer present in the question, we will then take its mean to get a general idea.

In [91]:
```
def count_matches(row):
    split_answer = row["clean_answer"].split()
    split_question = row["clean_question"].split()
    if "the" in split_answer:
        split_answer.remove("the")
    if len(split_answer) == 0:
        return 0
    match_count = 0
    for answer in split_answer:
        if answer in split_question:
            match_count += 1
    return match_count/len(split_answer)
```

In [124]:
```
jeopardy.head(3)
```

Out[124]:

| | show_number | air_date | round | category | value | question | answer | clean_q |
|---|---|---|---|---|---|---|---|---|
| **19325** | 10 | 1984-09-21 | Final Jeopardy! | U.S. PRESIDENTS | 0 | Adventurous 26th president, he was 1st to ride... | Theodore Roosevelt | adve 26th p he wa |
| **19301** | 10 | 1984-09-21 | Double Jeopardy! | LABOR UNIONS | 200 | Notorious labor leader missing since '75 | Jimmy Hoffa | notorio leader |
| **19302** | 10 | 1984-09-21 | Double Jeopardy! | 1789 | 200 | Washington proclaimed Nov. 26, 1789 this first... | Thanksgiving | was proclai 26 1 |

In [93]:
```
jeopardy["answer_in_question"] =  jeopardy.apply(count_matches,axis = 1)
```

In [125]:
```
jeopardy.head(3)
```

Out[125]:

| | show_number | air_date | round | category | value | question | answer | clean_q |
|---|---|---|---|---|---|---|---|---|
| **19325** | 10 | 1984-09-21 | Final Jeopardy! | U.S. PRESIDENTS | 0 | Adventurous 26th president, he was 1st to ride... | Theodore Roosevelt | adve 26th p he wa |
| **19301** | 10 | 1984-09-21 | Double Jeopardy! | LABOR UNIONS | 200 | Notorious labor leader missing since '75 | Jimmy Hoffa | notorio leader |
| **19302** | 10 | 1984-09-21 | Double Jeopardy! | 1789 | 200 | Washington proclaimed Nov. 26, 1789 this first... | Thanksgiving | was proclai 26 1 |

In [95]:
```
jeopardy.answer_in_question.mean()
```

Out[95]:
```
0.06291895444478074
```

## Only 6% the answer can be used for a question

We found the mean to be - 6% This is actually a very small proportion (only 6.29%) of questions that contain some part of the answer in them. This tells us that just by this idea, we cannot win Jeopardy.

## Investigate about repeat question

Lets now try to see how often new questions are repeat of older ones. Now the dataset(sample) we are working with is just a representative of the population, hence we can only investigate this phenomenon and try to generalize it.

In [96]:
```
jeopardy = jeopardy.sort_values('air_date',ascending = True)
question_overlap = []
terms_used = set()
for i,row in jeopardy.iterrows():
    split_question = row["clean_question"].split()
    for characters in split_question:
        if len(characters) <=6:
            split_question.remove(characters)
    match_count = 0
    for word in split_question:
        if word in terms_used:
            match_count += 1
    for word in split_question:
        terms_used.add(word)
    if len(split_question) > 0:
        match_count /= len(split_question)
    question_overlap.append(match_count)

jeopardy["question_overlap"] = question_overlap
```

In [97]:
```
jeopardy["question_overlap"].head(10)
```

Out[97]:
```
19325    0.000000
19301    0.000000
19302    0.000000
19303    0.200000
19304    0.142857
19305    0.000000
19306    0.000000
19307    0.200000
19308    0.166667
19309    0.000000
Name: question_overlap, dtype: float64
```

In [98]:
```
terms_used.d # xóa .d để hiện kết quả
```

```
AttributeErrorTraceback (most recent call last)
<ipython-input-98-4b5c5592b5fe> in <module>()
----> 1 terms_used.d # xóa .d để hiện kết quả

AttributeError: 'set' object has no attribute 'd'
```

In [99]:
```
jeopardy["question_overlap"].mean()
```

Out[99]:
```
0.8258987099594451
```

The percentage is around - 82.58%. This is a considerable amount but we are only considering unigrams. This high percentage can be because certain words repeat multiple times but not neccessarily in the same context.

## Low Value vs High Value Question

The game is all about answering questions and earning money for every correct answer. So let us try to seggregate our analysis into high value questions and low value questions.

Let us consider a threshold for high and low separation.

## Determine Value

In [100]:
```
def determine_value(row):
    value = 0
    if row["value"] > 750:
        value = 1
    return value
```

## Determine which questions are high and low value.

In [101]:
```
jeopardy["high_value"] =  jeopardy.apply(determine_value,axis = 1)
```

In [102]:
```
jeopardy["high_value"].head()
```

Out[102]:
```
19325    0
19301    0
19302    0
19303    0
19304    0
Name: high_value, dtype: int64
```

In [103]:
```
## Muốn lặp DF thì dùng iterrows để lặp các dòng
## Tạo ra 1 function chỉ ra "từ cần tìm" trong câu hỏi xuất hiện ở high_value
bao nhiêu lần, ở lơ_value bap nhiêu lần
def count_usage(term):
    low_count = 0
    high_count = 0
    for i, row in jeopardy.iterrows():
        if term in row["clean_question"].split(" "):
            if row["high_value"] == 1:
                high_count += 1
            else:
                low_count += 1
    return high_count, low_count
```

In [104]:
```
count_usage("term")
```

Out[104]:
```
(138, 140)
```

In [105]:
```
jeopardy.head()
```

Out[105]:

| | show_number | air_date | round | category | value | question | answer | clean_q |
|---|---|---|---|---|---|---|---|---|
| **19325** | 10 | 1984-09-21 | Final Jeopardy! | U.S. PRESIDENTS | 0 | Adventurous 26th president, he was 1st to ride... | Theodore Roosevelt | adve 26th p he wa |
| **19301** | 10 | 1984-09-21 | Double Jeopardy! | LABOR UNIONS | 200 | Notorious labor leader missing since '75 | Jimmy Hoffa | notorio leader |
| **19302** | 10 | 1984-09-21 | Double Jeopardy! | 1789 | 200 | Washington proclaimed Nov. 26, 1789 this first... | Thanksgiving | was proclai 26 1 |
| **19303** | 10 | 1984-09-21 | Double Jeopardy! | TOURIST TRAPS | 200 | Both Ferde Grofe' & the Colorado River dug thi... | the Grand Canyon | both fer the river |
| **19304** | 10 | 1984-09-21 | Double Jeopardy! | LITERATURE | 200 | Depending on the book, he could be a "Jones", ... | Tom | deper the co |

## RANDOM

Now we have this, let us use the set words_used that we created earlier and observe the frequency of that word for high and low value questions.

In [106]:
```
import random

terms_used_list = list(terms_used)
comparison_terms = random.sample(terms_used_list,10)

observed_expected = []

for term in comparison_terms:
    observed_expected.append(count_usage(term))
```

In [107]:
```
observed_expected
```

Out[107]:
```
[(0, 1),
 (3, 0),
 (0, 1),
 (0, 1),
 (0, 2),
 (18, 32),
 (1, 0),
 (1, 0),
 (3, 1),
 (7, 6)]
```

## Chi-squared Test

In [108]:
```
high_value_count = jeopardy[jeopardy["high_value"] == 1].shape[0]
high_value_count
```

Out[108]:
```
8714
```

In [109]:
```
low_value_count = jeopardy[jeopardy["high_value"] == 0].shape[0]
low_value_count
```

Out[109]:
```
11285
```

In [110]:
```
from scipy.stats import chisquare
import numpy as np
chi_squared = []
for obs in observed_expected:
    total = jeopardy.value[0] + jeopardy.value[1]
    total_prop = total/(jeopardy.shape[0])
    high_value_expect = total_prop * high_value_count
    low_value_expecet = total_prop * low_value_count

    observed = np.array([obs[0], obs[1]])
    expect = np.array([high_value_expect,low_value_expecet])
    chi_squared.append(chisquare(observed,expect))

chi_squared
```

Out[110]:
```
[Power_divergenceResult(statistic=398.0044304386353, pvalue=1.497434181928296
6e-88),
 Power_divergenceResult(statistic=394.0516384553592, pvalue=1.086030920045637
1e-87),
 Power_divergenceResult(statistic=398.0044304386353, pvalue=1.497434181928296
6e-88),
 Power_divergenceResult(statistic=398.0044304386353, pvalue=1.497434181928296
6e-88),
 Power_divergenceResult(statistic=396.0177217545414, pvalue=4.053566369574130
5e-88),
 Power_divergenceResult(statistic=306.3957535555361, pvalue=1.331680744680097
4e-68),
 Power_divergenceResult(statistic=398.005737606151, pvalue=1.4964533578241097
e-88),
 Power_divergenceResult(statistic=398.005737606151, pvalue=1.4964533578241097
e-88),
 Power_divergenceResult(statistic=392.0560688939945, pvalue=2.953055117869824
4e-87),
 Power_divergenceResult(statistic=374.4406384922729, pvalue=2.019969533703919
6e-83)]
```

For every word, the p-value is much higher than the threshold - 0.05. Hence we fail to reject the null hypothesis. This means that by examining these 5 words, we found no statistical significance suggesting that these words can help us identify the type of question (high-value or low-value) we are dealing with.

The above result is only for 5 terms, and maybe inconclusive of the bigger picture. Thus let us try it again with more words.

In [111]:
```
def chi_test(observed_values,vocab):
    high_value_count = np.count_nonzero(jeopardy .high_value == 1)
    low_value_count = np.count_nonzero(jeopardy .high_value == 0)

    total_rows = len(jeopardy)

    for word,l in zip(vocab,observed_values):
        total = sum(l)
        total_prop = total / total_rows

        expected_high = total_prop * high_value_count
        expected_low = total_prop * low_value_count

        observed = np.array([l[0],l[1]])
        expected = np.array([expected_high,expected_low])

        chi_squared[word] = chisquare(observed,expected)
```

In [112]:
```
chi_squared = {}
chi_test(observed_expected ,comparison_terms )
chi_squared
```

Out[112]:
```
{'1772': Power_divergenceResult(statistic=0.7721754541426672, pvalue=0.379544
8984353682),
 '1824': Power_divergenceResult(statistic=1.6068893994548774, pvalue=0.204929
6084582517),
 'agent': Power_divergenceResult(statistic=0.5581074545503374, pvalue=0.45502
37844510827),
 'bernice': Power_divergenceResult(statistic=0.7721754541426672, pvalue=0.379
5448984353682),
 'citron': Power_divergenceResult(statistic=0.7721754541426672, pvalue=0.3795
448984353682),
 'goes': Power_divergenceResult(statistic=1.1660284442891515, pvalue=0.280218
7935609849),
 'isis': Power_divergenceResult(statistic=1.5443509082853344, pvalue=0.213971
34128528295),
 'ringlike': Power_divergenceResult(statistic=1.295042460408538, pvalue=0.255
12076479610835),
 'sass': Power_divergenceResult(statistic=1.295042460408538, pvalue=0.2551207
6479610835),
 'vince': Power_divergenceResult(statistic=3.8851273812256135, pvalue=0.04871
556686149284)}
```

In [113]:
```
jeopardy['round'].value_counts()
```

Out[113]:
```
Jeopardy!           9901
Double Jeopardy!    9762
Final Jeopardy!      335
Tiebreaker             1
Name: round, dtype: int64
```

Looking at the data and making its cross table with the value_level column, we can tell that Doube Jeopardy round holds the most high-value questions. But how do we know whether this phenomenon if just by chance (for this sample) or is this true for the population.

In [114]:
```
cross_table = pd.crosstab(jeopardy['round'],jeopardy['high_value'])
cross_table
```

Out[114]:

| high_value | 0 | 1 |
|---|---|---|
| round | | |
| **Double Jeopardy!** | 3507 | 6255 |
| **Final Jeopardy!** | 335 | 0 |
| **Jeopardy!** | 7442 | 2459 |
| **Tiebreaker** | 1 | 0 |

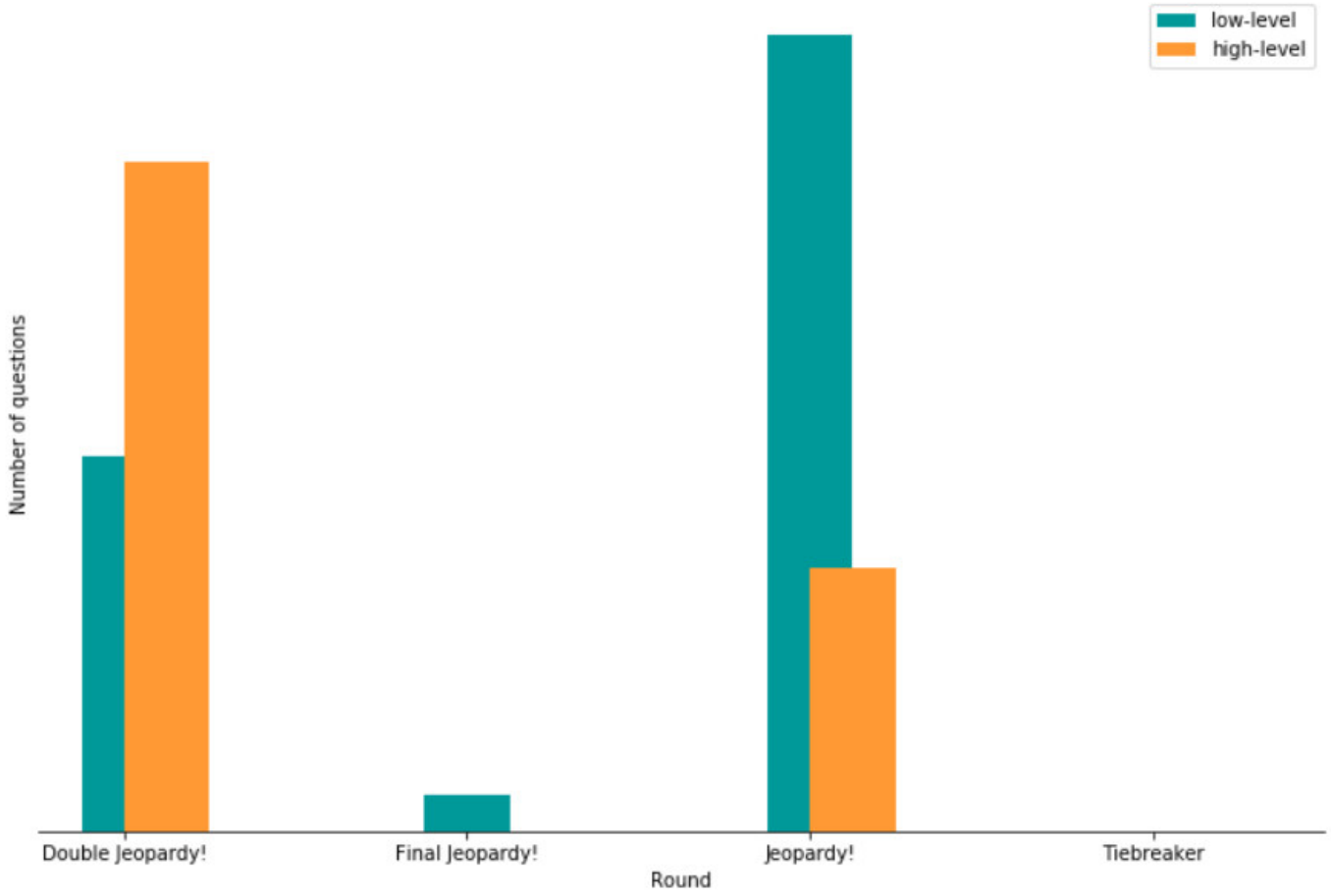
For this purpose, let us perform a chi-square test using the scipy.stats.chi2_contingency function on the cross table. The null hypothesis is that there is no correlation between the rounds and the value level of the questions.

The alternative hypothesis is that there exists some correlation between the rounds and value level of the questions.

In [115]:
```
from scipy.stats import chisquare,chi2_contingency
chi_sq,p_value,dof,expected = chi2_contingency(cross_table)
p_value
```

Out[115]:
```
0.0
```

In [116]:
```
plt.figure(figsize=(12,8))
cross_table[0].plot.barh(align='center',color='#009999',label='low-level',widt
h=0.25)
cross_table[1].plot.barh(align='edge',color = '#ff9933',label='high-level',wid
th=0.25)
plt.legend()
plt.yticks([])
plt.xticks(rotation=0)
plt.ylabel('Number of questions')
plt.xlabel('Round')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['left'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
```




Looking at the cross table and the plot, we can then conclude the direction of the difference i.e. The first round Jeopardy hosts more low-level questions whereas the second round Double Jeopardy hosts more high-level questions. The final round Final Jeopardy and the Tiebreaker round host more of low-level questions.

The category column has the topic for the question. We have analysed which rounds have a higher chance of having high-value questions. Now let us look into the categories (topics) and its correlation with the value levels. This was we can have an idea whether there exists a relationship/correlation between the value level and topic of question.

```
In [126]: jeopardy .category.value_counts().head(10)
```

```
Out[126]: TELEVISION          51
          U.S. GEOGRAPHY      50
          LITERATURE          45
          BEFORE & AFTER      40
          HISTORY             40
          AMERICAN HISTORY    40
          AUTHORS             39
          WORD ORIGINS        38
          WORLD CAPITALS      37
          BODIES OF WATER     36
          Name: category, dtype: int64
```

Similar to the approach in finding the correlation between the words and value level of the questions. We will find the observed and expected frequencies of high-value and low-value questions for each category, to this we will apply a chi-square test to determine.

The null hypothesis is that there is no correlation between the level of the value and the topic of the question. The alternative hypothesis is that there is a correlation between the level of the value and the topic of the question.

In simple words we are checking for each category whether it is associated mostly with high-level questions or low-level questions. We will perform this analysis on the top 10 frequent topics in the sample data.

In [118]:
```
catgs = jeopardy.category.value_counts().sort_values(ascending=False)[:10].ind
ex

def observed(catg):
    high_count = 0
    low_count = 0

    for i,row in jeopardy.iterrows():
        if row.category == catg:
            if row.high_value == 1:
                high_count += 1
            else:
                low_count += 1

    return high_count,low_count

observed_values = []
for catg in catgs:
    observed_values.append(observed(catg))

observed_values
```

Out[118]:
```
[(9, 42),
 (20, 30),
 (21, 24),
 (24, 16),
 (12, 28),
 (18, 22),
 (12, 27),
 (15, 23),
 (14, 23),
 (10, 26)]
```

In [119]:
```
chi_squared = {}

def chi_test(observed_values,catgs):
    high_value_count = np.count_nonzero(jeopardy.high_value == 1)
    low_value_count = np.count_nonzero(jeopardy.high_value == 0)

    total_rows = len(jeopardy)

    for catg,l in zip(catgs,observed_values):
        total = sum(l)
        total_prop = total / total_rows

        expected_high = total_prop * high_value_count
        expected_low = total_prop * low_value_count

        observed = np.array([l[0],l[1]])
        expected = np.array([expected_high,expected_low])

        chi_squared[catg] = chisquare(observed,expected)

chi_test(observed_values,catgs)
chi_squared
```

Out[119]:
```
{'AMERICAN HISTORY': Power_divergenceResult(statistic=0.03316692443543142, pv
alue=0.855490212383109),
 'AUTHORS': Power_divergenceResult(statistic=2.6000518043290746, pvalue=0.106
86022199140573),
 'BEFORE & AFTER': Power_divergenceResult(statistic=4.390534336396016, pvalue
=0.03613898538801975),
 'BODIES OF WATER': Power_divergenceResult(statistic=3.652634806702691, pvalu
e=0.05598058556613405),
 'HISTORY': Power_divergenceResult(statistic=2.9967917586670154, pvalue=0.083
42957570170428),
 'LITERATURE': Power_divergenceResult(statistic=0.17526192502981316, pvalue=
0.6754770906301193),
 'TELEVISION': Power_divergenceResult(statistic=13.941489027465813, pvalue=0.
00018858955620803958),
 'U.S. GEOGRAPHY': Power_divergenceResult(statistic=0.25949785783631446, pval
ue=0.6104653431793821),
 'WORD ORIGINS': Power_divergenceResult(statistic=0.2596149692997901, pvalue=
0.6103847994683275),
 'WORLD CAPITALS': Power_divergenceResult(statistic=0.4948415535552552, pvalu
e=0.4817755106192815)}
```

Here we can see the majority of topics do not have p_value <= 0.05, meaning for these topics we fail to reject the null hypothesis. However, for two topics - TELEVISION and BEFORE & AFTER, the null hypothesis is rejected and hence can be said that it does have a correlation with the value levels.

We have only performed these tests for the top 10 most frequent categories (topics) in the data. Let us perform the same for the top 20 categories (topics).

In [120]:
```
catgs = jeopardy.category.value_counts().sort_values(ascending=False)[:20].ind
ex

observed_values = []
for catg in catgs:
    observed_values.append(observed(catg))

print(catgs)
observed_values
```

```
Index(['TELEVISION', 'U.S. GEOGRAPHY', 'LITERATURE', 'BEFORE & AFTER',
       'HISTORY', 'AMERICAN HISTORY', 'AUTHORS', 'WORD ORIGINS',
       'WORLD CAPITALS', 'BODIES OF WATER', 'SPORTS', 'RHYME TIME',
       'SCIENCE & NATURE', 'SCIENCE', 'MAGAZINES', 'WORLD GEOGRAPHY',
       'WORLD HISTORY', 'ANNUAL EVENTS', 'HISTORIC NAMES',
       'IN THE DICTIONARY'],
      dtype='object')
```

Out[120]:
```
[(9, 42),
 (20, 30),
 (21, 24),
 (24, 16),
 (12, 28),
 (18, 22),
 (12, 27),
 (15, 23),
 (14, 23),
 (10, 26),
 (7, 29),
 (12, 23),
 (21, 14),
 (21, 14),
 (12, 23),
 (11, 22),
 (10, 22),
 (11, 21),
 (15, 17),
 (22, 9)]
```

In [121]:
```
chi_squared = {}
chi_test(observed_values,catgs)
chi_squared
```

Out[121]:
```
{'AMERICAN HISTORY': Power_divergenceResult(statistic=0.03316692443543142, pv
alue=0.855490212383109),
 'ANNUAL EVENTS': Power_divergenceResult(statistic=1.1009222808234171, pvalue
=0.2940637912962565),
 'AUTHORS': Power_divergenceResult(statistic=2.6000518043290746, pvalue=0.106
86022199140573),
 'BEFORE & AFTER': Power_divergenceResult(statistic=4.390534336396016, pvalue
=0.03613898538801975),
 'BODIES OF WATER': Power_divergenceResult(statistic=3.652634806702691, pvalu
e=0.05598058556613405),
 'HISTORIC NAMES': Power_divergenceResult(statistic=0.14197686997349648, pval
ue=0.7063235918777161),
 'HISTORY': Power_divergenceResult(statistic=2.9967917586670154, pvalue=0.083
42957570170428),
 'IN THE DICTIONARY': Power_divergenceResult(statistic=9.462798794299626, pva
lue=0.002096808976711425),
 'LITERATURE': Power_divergenceResult(statistic=0.17526192502981316, pvalue=
0.6754770906301193),
 'MAGAZINES': Power_divergenceResult(statistic=1.2276265582942991, pvalue=0.2
6786911298547267),
 'RHYME TIME': Power_divergenceResult(statistic=1.2276265582942991, pvalue=0.
26786911298547267),
 'SCIENCE': Power_divergenceResult(statistic=3.8417175443465155, pvalue=0.049
99228562898841),
 'SCIENCE & NATURE': Power_divergenceResult(statistic=3.8417175443465155, pva
lue=0.04999228562898841),
 'SPORTS': Power_divergenceResult(statistic=8.523795485944486, pvalue=0.00350
532671847749),
 'TELEVISION': Power_divergenceResult(statistic=13.941489027465813, pvalue=0.
00018858955620803958),
 'U.S. GEOGRAPHY': Power_divergenceResult(statistic=0.25949785783631446, pval
ue=0.6104653431793821),
 'WORD ORIGINS': Power_divergenceResult(statistic=0.2596149692997901, pvalue=
0.6103847994683275),
 'WORLD CAPITALS': Power_divergenceResult(statistic=0.4948415535552552, pvalu
e=0.4817755106192815),
 'WORLD GEOGRAPHY': Power_divergenceResult(statistic=1.4070623489237597, pval
ue=0.23554467279401614),
 'WORLD HISTORY': Power_divergenceResult(statistic=1.9761614326845232, pvalue
=0.1597953765739848)}
```

We have new additions to our list of topics that have correlation with the value levels, they are - SPORTS, SCIENCE, SCIENCE & NATURE, BIRDS and the ones from previous analysis as well as this, TELEVISION and BEFORE & AFTER.

Let us make a cross table for these topics, to understand the frequencies of these topics with respect to the value level.

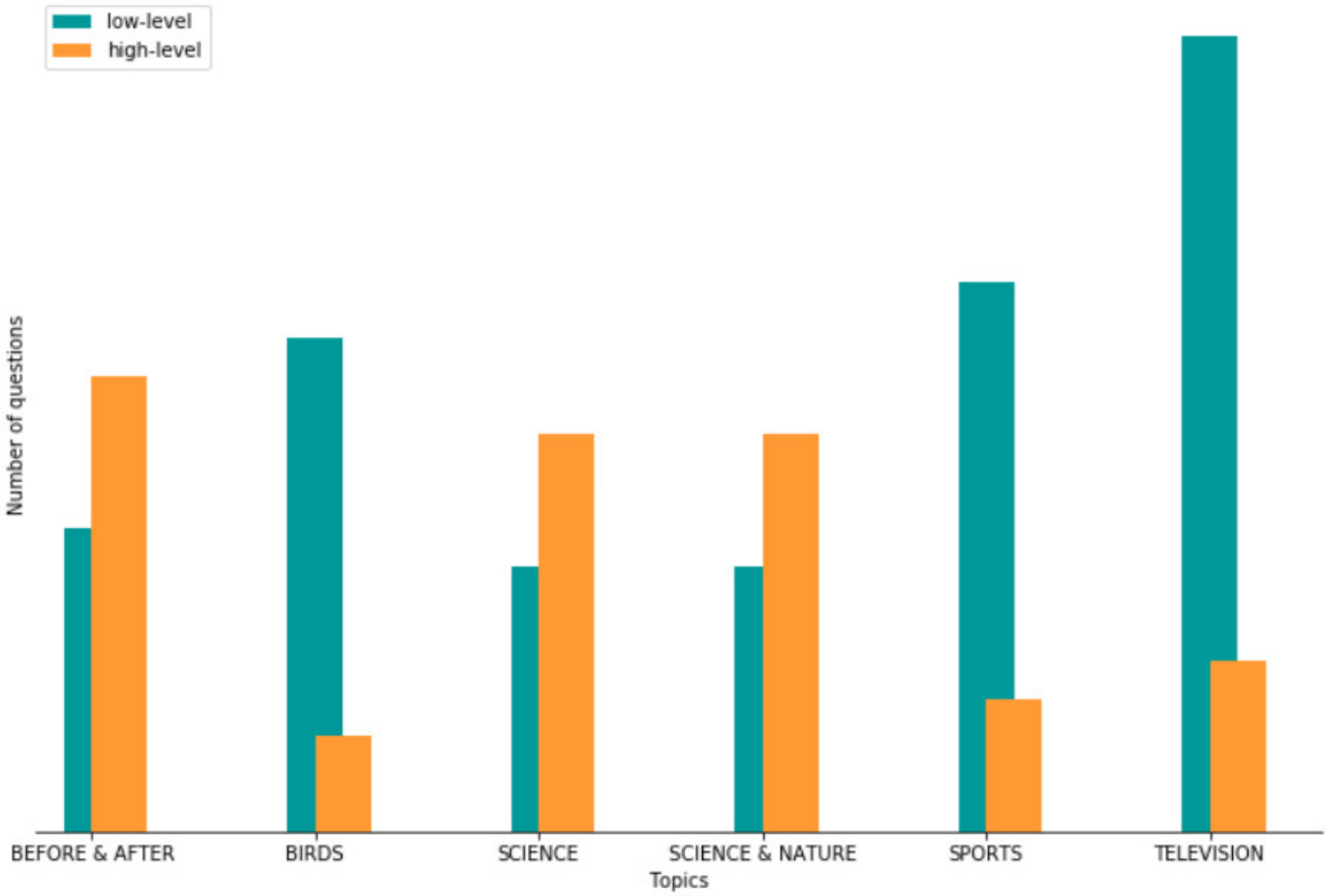
In [122]:
```
catg_interest = [
    'SPORTS',
    'SCIENCE',
    'SCIENCE & NATURE',
    'BIRDS',
    'TELEVISION',
    'BEFORE & AFTER'
]

subset = jeopardy[jeopardy.category.isin(catg_interest)]
cross_table = pd.crosstab(subset.category,subset.high_value)
cross_table
```

Out[122]:

| high_value | 0 | 1 |
|---|---|---|
| category | | |
| BEFORE & AFTER | 16 | 24 |
| BIRDS | 26 | 5 |
| SCIENCE | 14 | 21 |
| SCIENCE & NATURE | 14 | 21 |
| SPORTS | 29 | 7 |
| TELEVISION | 42 | 9 |

In [123]:
```
plt.figure(figsize=(12,8))
cross_table[0].plot.bar(align='center',color='#009999',label='low-level',width
=0.25)
cross_table[1].plot.bar(align='edge',color = '#ff9933',label='high-level',widt
h=0.25)
plt.legend()
plt.yticks([])
plt.xticks(rotation=0)
plt.ylabel('Number of questions')
plt.xlabel('Topics')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['left'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
```




Looking into the cross table, the plot and the p_values obtained from before, we can say that the topics SPORTS, TELEVISION, BIRDS have a higher chance of being low-level questions, whereas the topics BEFORE & AFTER, SCIENCE and SCIENCE & NATURE have a higher chance of being high-level questions.

From our analysis, we can conclude :-

1. The answers are hardly hidden in the questions and hence the participant has to be revised with all categories (topics).
2. The repetition of questions is rare, the participant must not rely on reading previous questions only to win the game.
3. No relationship was found between the level of the question (>750 or <750 dollars) and the words present in the questions. Thus the participant cannot estimate the level of the question with respect to words in the question.
4. The first round, Jeopardy! hosts mostly low-level (<750 dollars) questions. Whereas the second round Double Jeopardy! hosts high-level (>750 dollars) questions. Participant's aim to win more money can utilize these findings and play accordingly.
5. The categories (topics) - SPORTS, TELEVISION and BIRDS have a higher chance of having low-level (<750 dollars) questions, whereas the categories (topics) BEFORE & AFTER, SCIENCE and SCIENCE & NATURE have a highe chance of having high-level (> 750 dollars) questions.

From the above conclusions, the participant can accordingly prepare and choose to answer questions in the game in order to win more money and overall be successful in the game.

In [ ]: