(http://www.pieriandata.com)

Sentiment Analysis Assessment - Solution

Task #1: Perform vector arithmetic on your own words

Write code that evaluates vector arithmetic on your own set of related words. The goal is to come as close to an expected word as possible. Please feel free to share success stories in the Q&A Forum for this section!

In [1]:
```
# Import spaCy and load the language library. Remember to use a larger m

import spacy
nlp = spacy.load('en_core_web_lg')
```

In [2]:
```
# Choose the words you wish to compare, and obtain their vectors
word1 = nlp.vocab['wolf'].vector
word2 = nlp.vocab['dog'].vector
word3 = nlp.vocab['cat'].vector
```

In [3]:
```
# Import spatial and define a cosine_similarity function

from scipy import spatial

cosine_similarity = lambda x, y: 1 - spatial.distance.cosine(x, y)
```

In [4]:
```
# Write an expression for vector arithmetic
# For example: new_vector = word1 - word2 + word3
new_vector = word1 - word2 + word3
```

In [5]:
```
# List the top ten closest vectors in the vocabulary to the result of th
computed_similarities = []

for word in nlp.vocab:
    if word.has_vector:
        if word.is_lower:
            if word.is_alpha:
                similarity = cosine_similarity(new_vector, word.vector)
                computed_similarities.append((word, similarity))


computed_similarities = sorted(computed_similarities,
                               key = lambda item: -item[1])

print([w[0].text for w in computed_similarities[:10]])
```

```
['wolf', 'cat', 'i', 'dare', 'dog', 'she', 'ai', 'ca', 'lovin', 'would
']
```

CHALLENGE: Write a function that takes in 3 strings, performs a-b+c arithmetic, and returns a top-ten result

In [7]:
```
def vector_math(a,b,c):
    word1 = nlp.vocab[a].vector
    word2 = nlp.vocab[b].vector
    word3 = nlp.vocab[c].vector
    new_vector = word1 - word2 + word3

    computed_similarities = []
    for word in nlp.vocab:
        if word.has_vector:
            if word.is_lower:
                if word.is_alpha:
                    similarity = cosine_similarity(new_vector,
                                                   word.vector)
                    computed_similarities.append((word, similarity))


    computed_similarities = sorted(computed_similarities,
                                   key = lambda item: -item[1])

    return( [w[0].text for w in computed_similarities[:10]] )
```

In [8]:
```
# Test the function on known words:
vector_math('king','man','woman')
```

Out[8]:
```
['king', 'woman', 'she', 'who', 'wolf', 'when', 'dare', 'cat', 'was',
'not']
```

Task #2: Perform VADER Sentiment Analysis on your own review

Write code that returns a set of SentimentIntensityAnalyzer polarity scores based on your own written review.

In [9]:
```
# Import SentimentIntensityAnalyzer and create an sid object

from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
```

In [10]:
```
# Write a review as one continuous string (multiple sentences are ok)
review = 'This movie was based on a true story.'
```

In [11]:
```
# Obtain the sid scores for your review
sid.polarity_scores(review)
```

Out[11]:
```
{'neg': 0.0, 'neu': 0.682, 'pos': 0.318, 'compound': 0.4215}
```

CHALLENGE: Write a function that takes in a review and returns a score of "Positive", "Negative" or "Neutral"

In [12]:
```
def review_rating(string):
    scores = sid.polarity_scores(string)
    if scores['compound'] == 0:
        return("Neutral")
    elif scores['compound'] > 0:
        return("Positive")
    else:
        return("Negative")
```

In [13]:
```
# Test the function on your review above:
review_rating(review)
```

Out[13]:
```
'Positive'
```

In [14]:
```
review = 'This movie portrayed real people and was based on actual event
```

In [ ]:
```

```

Great job!