In [1]:

```
import textblob
from textblob import TextBlob
text = 'Today is a beautiful day. Tomorrow looks like bad weather.'
blob = TextBlob(text)
blob
```

Out[1]:

```
TextBlob("Today is a beautiful day. Tomorrow looks like bad weather.")
```

In [2]:

```
# One of the most common and valuable NLP tasks is sentiment analysis, which determines
# whether text is positive, neutral or negative. For instance, companies might use this
# determine whether people are speaking positively or negatively online about their prod
# Consider the positive word “good” and the negative word “bad.” Just because a sentence
# contains “good” or “bad” does not mean the sentence’s sentiment necessarily is
# positive or negative.
# Sentiment analysis is a complex machine-learning problem. However, libraries like
# TextBlob have pretrained machine learning models for performing sentiment analysis.
# A TextBlob’s sentiment property returns a Sentiment object indicating whether the text
# is positive or negative and whether it’s objective or subjective:

blob.sentiment

# the polarity indicates sentiment with a value from -1.0 (negative)
# to 1.0 (positive) with 0.0 being neutral. The subjectivity is a value from 0.0
# (objective) to 1.0 (subjective). Based on the values for our TextBlob, the overall sen
# is close to neutral, and the text is mostly subjective.
```

Out[2]:

```
Sentiment(polarity=0.07500000000000007, subjectivity=0.8333333333333333)
```

In [3]:

```
# The values displayed above probably provide more precision that you need in most cases
# This can detract from numeric output’s readability. The IPython magic %precision
# allows you to specify the default precision for standalone float objects and float obj
# in built-in types like lists, dictionaries and tuples. Let’s use the magic to round th
# and subjectivity values to three digits to the right of the decimal point:

%precision 3
```

Out[3]:

```
'%.3f'
```

In [4]:

```
blob.sentiment.polarity
```

Out[4]:

```
0.075
```

In [5]:

```
blob.sentiment.subjectivity
```

Out[5]:

```
0.833
```

In [6]:

```
# You also can get the sentiment at the individual sentence level. Let's use the senten
# property to get a list of Sentence objects, then iterate through them and display eac
# sentiment property:

for sentence in blob.sentences:
    print(sentence.sentiment)

# This might explain why the entire TextBlob's sentiment is close to 0.0 (neutral)—one
# sentence is positive (0.85) and the other negative (-0.6999999999999998).
```

```
Sentiment(polarity=0.85, subjectivity=1.0)
Sentiment(polarity=-0.6999999999999998, subjectivity=0.6666666666666666)
```

In [7]:

```
# By default, a TextBlob and the Sentences and Words you get from it determine sentiment
# using a PatternAnalyzer, which uses the same sentiment analysis techniques as in the P
# library. The TextBlob library also comes with a NaiveBayesAnalyzer9 (module textblob.
# sentiments), which was trained on a database of movie reviews. Naive Bayes10 is a
# commonly used machine learning text-classification algorithm. The following uses the
# analyzer keyword argument to specify a TextBlob's sentiment analyzer.

from textblob.sentiments import NaiveBayesAnalyzer
```

In [8]:

```
blob = TextBlob(text, analyzer=NaiveBayesAnalyzer())
```

In [9]:

```
blob
```

Out[9]:

```
TextBlob("Today is a beautiful day. Tomorrow looks like bad weather.")
```

In [10]:

```
# Let's use the TextBlob's sentiment property to display the text's sentiment using the
# NaiveBayesAnalyzer:

blob.sentiment
```

Out[10]:

```
Sentiment(classification='neg', p_pos=0.47662917962091056, p_neg=0.5233708
203790892)
```

In [11]:

```
# In this case, the overall sentiment is classified as negative (classification='neg').
# Sentiment object's p_pos indicates that the TextBlob is 47.66% positive, and its p_neg
# indicates that the TextBlob is 52.34% negative. Since the overall sentiment is just sl
# more negative we'd probably view this TextBlob's sentiment as neutral overall.
# Now, let's get the sentiment of each Sentence:

for sentence in blob.sentences:
    print(sentence.sentiment)

# Notice that rather than polarity and subjectivity, the Sentiment objects we get from
# the NaiveBayesAnalyzer contain a classification—'pos' (positive) or 'neg' (negative)—
# and p_pos (percentage positive) and p_neg (percentage negative) values from 0.0 to 1.0
# Once again, we see that the first sentence is positive and the second is negative.
```

```
Sentiment(classification='pos', p_pos=0.8117563121751951, p_neg=0.18824368
782480477)
Sentiment(classification='neg', p_pos=0.174363226578349, p_neg=0.825636773
4216521)
```