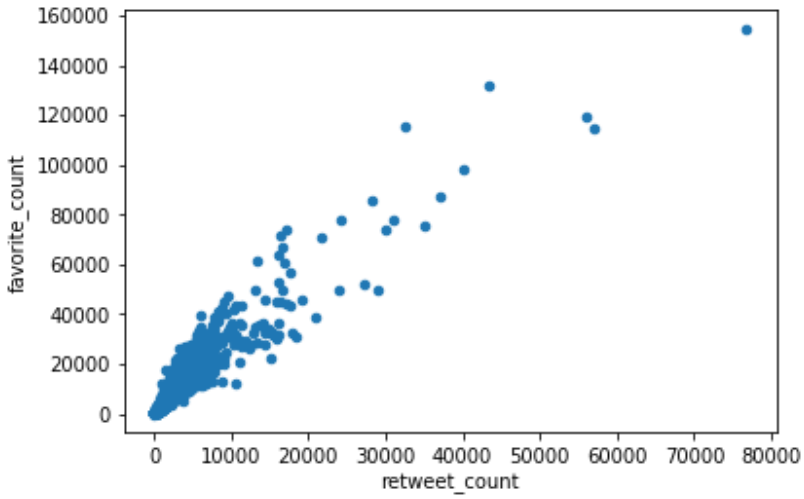
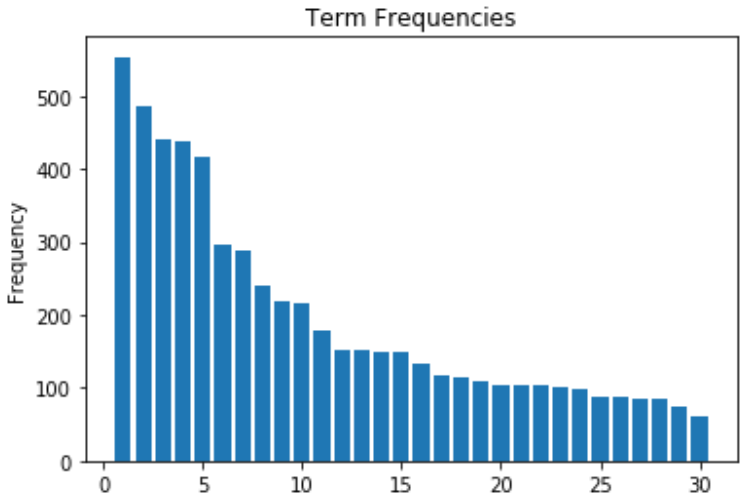
In [73]:
```
twitter_archive_master.plot(x='retweet_count', y= 'favorite_count', kind =('scatter'));
```



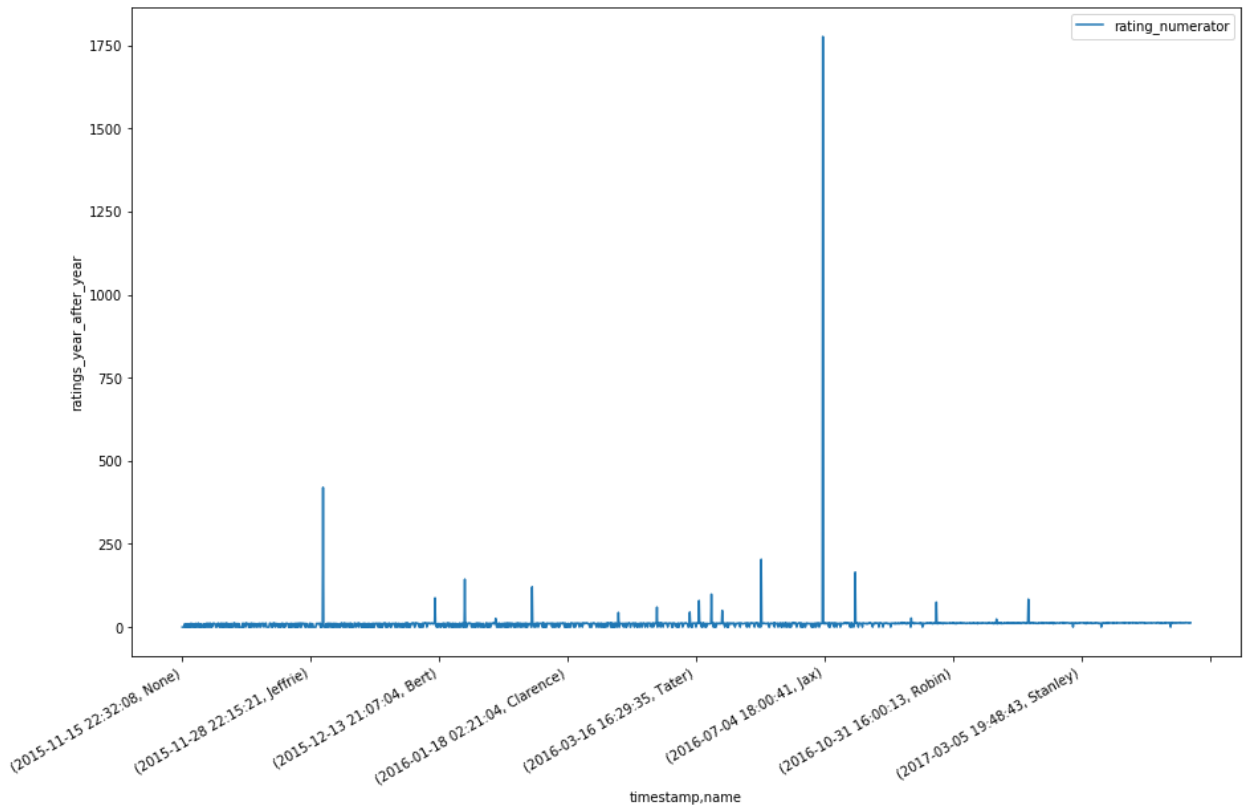
This analysis focuses on unstructured data, that is, the raw text of the tweet

In [71]:
```
y = [count for tag, count in tf.most_common(30)]
x = range(1, len(y)+1)
plt.bar(x, y)
plt.title("Term Frequencies")
plt.ylabel("Frequency")
plt.savefig('term_distribution.png')
```



In [76]:
```
Dogs_with_high_ratings_year_after_year = df_group1.groupby(['timestamp', 'name'])
Dogs_with_high_ratings_year_after_year.mean().plot(subplots=True, figsize=(15,10));
plt.ylabel('ratings_year_after_year')
```

Out[76]:
```
Text(0, 0.5, 'ratings_year_after_year')
```


rating_numerator
1750
1500
1250
1000
750
500
250
0
ratings_year_after_year
(2015-11-15 22:32:08, None)
(2015-11-28 22:15:21, Jeffrie)
(2015-12-13 21:07:04, Bert)
(2016-01-18 02:21:04, Clarence)
(2016-03-16 16:29:35, Tater)
(2016-07-04 18:00:41, Jax)
(2016-10-31 16:00:13, Robin)
(2017-03-05 19:48:43, Stanley)
timestamp,name

This first analysis figure 2 above, focuses on unstructured data, that is, the raw text of the tweet.

Aspects of text analytics include but are not limited to the following:

1. Text pre-processing
2. Normalization
3. etc

1. Text pre-processing:

Tokenization is one of the important steps in the preprocessing phase. Given a stream of text (such as a tweet status), tokenization is the process of breaking this text down into individual units called tokens. In the simplest form, these units are words, but we could also work on a more complex tokenization that deals with phrases, symbols, and so on. Tokenization sounds like a trivial task, and it's been widely studied by the natural language processing community.

Twitter changes the rules of tokenization because the content of a tweet includes emoticons, hashtags, user mentions, URLs, and is quite different from standard English.

For this reason, when using the Natural Language Toolkit (NLTK) library, we showcased the TweetTokenizer class as a tool to tokenize Twitter content. l made use of this tool in this project.

Another preprocessing step that is worth considering is stop word removal. Stop words are words that, when taken in isolation, are not content-bearing. This category of words includes articles, propositions, adverbs, and so on.

Frequency analysis will show that these words are typically the most common in any dataset. While a list of stop words can be compiled automatically by analyzing the data (for example, by including words that are present in more than an arbitrary percentage of the documents, such as 95%), in general, it pays off to be more conservative and use only common English stop words.

NLTK provides a list of common English stop words via the nltk.corpus.stopwords module.

Stop word removal can be extended to include symbols as well (such as punctuation) or domain-specific words.

In our Twitter context, common stop words are the terms ‘rt' (short for Retweet) and via, often used to mention the author of the content being shared.

1. 2, Normalization

Another important preprocessing step is normalization. This is an umbrella term that can consist of several types of processing. In general, normalization is used when we need to aggregate different terms in the same unit. The special case of normalization considered here is case normalization, where every term is lowercase so that strings with originally different casing will match (for example, 'python' == 'Python'.lower()). The advantage of performing case normalization is that the frequency of a given term will be automatically aggregated, rather than being dispersed into the different variations of the same term.

The core of the preprocessing logic is implemented by the process() function. The function takes a string as input and returns a list of strings as output. All the preprocessing steps mentioned earlier, case normalization, tokenization, and stop words removal are implemented here in a few lines a code.

The function also takes two more optional arguments: a tokenizer, that is, an object that implements a tokenize() method and a list of stop words, so the stop word-removal process can be customized. When applying stop word removal, the function also removes numerical tokens (for example, '5' or '42') using the isdigit() function over a string.

The script initializes TweetTokenizer used for tokenization, and then defines a list of stop words. Such list is made up of common English stop words coming from NLTK, as well as punctuation symbols defined in string.punctuation.

To complete the stop word list, we also include the ‘rt’, ‘via’ tokens.

The second figure above doesn't report the terms per se, as the focus of this analysis is frequency distribution. As we can see from the second figure, there are a few terms on the left with very high frequency, for example, the most frequent term is twice more frequent than the ones after position 10 or so. As we move towards the right-hand side of the figure, the curve becomes less steep, meaning that the terms on the right share similar frequencies.

The curve that we can observe in the second figure *r*epresents an approximation of a Power law (https://en.wikipedia.org/wiki/Power_law).

In statistics, a power law is a functional relationship between two quantities; in this case, the frequency of a term and its position within the ranking of terms by frequency.

This type of distribution always shows a long tail (https://en.wikipedia org/wiki/Long_tail), meaning that a small portion of frequent items dominate the distribution, while there is a large number of items with smaller frequencies. Another name for this phenomenon is the 80-20 rule or Pareto principle

(https://en.wikipedia.org/wiki/Pareto_principle), which states that roughly 80% of the effect comes from 20% of the cause (in our context, 20% of the unique terms account for 80% of all term occurrences).

A few decades ago, the American linguist George Zipf popularized what is nowadays known as the Zipf's law (https://en.wikipedia.org/wiki/Zipf%27s_law).

This empirical law states that given a collection of documents, the frequency of any word is inversely proportional to its rank in the frequency table. In other words, the most frequent word will be seen twice as often as the second most frequent one, three times as often as the third most frequent one, and so on. In practice, this law describes a trend rather than the precise frequencies. Interestingly enough, Zipf's law can be generalized for many different natural languages, as well as many language-unrelated rankings studies in social sciences.

In the third figure, I tried to show dogs with high ratings year after year and the result from this third figure suggests that the dog(s) called jax have higher ratings