part1-final

January 3, 2019

0.1 # News Analytics and Stock Price Performance: EDA (part 1)

Can we use news analytics and market data to predict stock price performance? There is no doubt that the ubiquity of data today enables investors at any scale to make better investment decisions but to truly harness this power, we must be able to distinguish signal from noise.

This is a 3 part walkthrough of a Kaggle competition by Two Sigma, with the end result being a model that predicts a signed confidence of an assets fluctuation over a ten-day window.

$$\hat{y}_{ti} \in [-1, 1]$$

Initially, I was a little confused with the evaluation process here. Most people think of stock market predictions as being regression problems but this seemed like a binary classification problem to me. An asset either has a positive or a negative return, with the signed confidence being used to indicate both the direction and the magnitude of this move.

as mentioned above, the signed confidence interval needs to be between [-1 and 1]. Binary classification models are going to output a probability, naturally being a number between 0 and 1. So in order to get the output of my model to conform to this structure, I decided I would multiply my predicted value by 2 and then subtract 1. If the predicted probability is 0, this will cause the output to be -1 and if the predicted probability is 1, then the output will be 1.

For each day in the evaluation time period, we calculate:

$$x_t = \sum_i \hat{y}_{ti} r_{ti} u_{ti}$$

$$score = \frac{\bar{x}_t}{\sigma(x_t)}$$

where r_{ti} is the 10-day market-adjusted leading return for day t for instrument i, and u_{ti} is a 0/1 universe variable that controls whether a particular asset is included in scoring on a particular day.

Your submission score is then calculated as the mean divided by the standard deviation of your daily x_t values:

If the standard deviation of predictions is 0, the score is defined as 0.

Two sources of data for this competition:

Market data (2007 to present) provided by Intrinio - contains financial market information such as opening price, closing price, trading volume, calculated returns, etc.

News data (2007 to present) Source: Thomson Reuters - contains information about news articles/alerts published about assets, such as article details, sentiment, and other commentary.

There are 3 notebooks for this walkthrough. The first includes all of the EDA for both datasets. In the second, I walk through feature preprocessing, exploration and engineering. Finally, in the third notebook, I build, test and tune multiple machine learning models.

In [1]:
```
import numpy as np
import pandas as pd
import os
from kaggle.competitions import twosigmanews
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import seaborn as sns; sns.set()
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
```

In [2]:
```
# Load training data from API
env = twosigmanews.make_env()
(market_train_df, news_train_df) = env.get_training_data()
```

```
Loading the data... This could take a minute.
Done!
```

1 Inspecting market_train_df

In [3]:
```
print(f'market_train_df shape: {market_train_df.shape}')
market_train_df.head()
```

```
market_train_df shape: (4072956, 16)
```

Out[3]:
```
                        time  ...   universe
0 2007-02-01 22:00:00+00:00  ...        1.0
1 2007-02-01 22:00:00+00:00  ...        0.0
2 2007-02-01 22:00:00+00:00  ...        1.0
3 2007-02-01 22:00:00+00:00  ...        1.0
4 2007-02-01 22:00:00+00:00  ...        1.0

[5 rows x 16 columns]
```

In [4]:
```
market_train_df.tail()
```

Out[4]:
```
                              time  ...   universe
4072951 2016-12-30 22:00:00+00:00  ...        0.0
4072952 2016-12-30 22:00:00+00:00  ...        0.0
4072953 2016-12-30 22:00:00+00:00  ...        0.0
4072954 2016-12-30 22:00:00+00:00  ...        1.0
4072955 2016-12-30 22:00:00+00:00  ...        1.0

[5 rows x 16 columns]
```

In [5]: market_train_df.describe()

```
Out[5]:              volume       ...           universe
        count  4.072956e+06       ...       4.072956e+06
        mean   2.665312e+06       ...       5.949365e-01
        std    7.687606e+06       ...       4.909044e-01
        min    0.000000e+00       ...       0.000000e+00
        25%    4.657968e+05       ...       0.000000e+00
        50%    9.821000e+05       ...       1.000000e+00
        75%    2.403165e+06       ...       1.000000e+00
        max    1.226791e+09       ...       1.000000e+00

        [8 rows x 13 columns]
```

2 Inspecting news_train_df

```
In [6]: print(f'news_train_df shape: {news_train_df.shape}')
        news_train_df.head()
```

```
news_train_df shape: (9328750, 35)
```

```
Out[6]:                        time       ...       volumeCounts7D
        0 2007-01-01 04:29:32+00:00       ...                    7
        1 2007-01-01 07:03:35+00:00       ...                    3
        2 2007-01-01 11:29:56+00:00       ...                   17
        3 2007-01-01 12:08:37+00:00       ...                   15
        4 2007-01-01 12:08:37+00:00       ...                    0

        [5 rows x 35 columns]
```

In [7]: news_train_df.tail()

```
Out[7]:                              time       ...       volumeCounts7D
        9328745 2016-12-30 21:56:06+00:00       ...                   10
        9328746 2016-12-30 21:56:28+00:00       ...                   11
        9328747 2016-12-30 21:57:00+00:00       ...                   41
        9328748 2016-12-30 21:58:53+00:00       ...                    3
        9328749 2016-12-30 22:00:00+00:00       ...                    0

        [5 rows x 35 columns]
```

In [8]: news_train_df.describe()

```
Out[8]:             urgency       ...        volumeCounts7D
        count  9.328750e+06       ...          9.328750e+06
        mean   2.321202e+00       ...          4.050544e+01
        std    9.470095e-01       ...          8.948574e+01
```

```
min    1.000000e+00    ...    0.000000e+00
25%    1.000000e+00    ...    4.000000e+00
50%    3.000000e+00    ...    1.300000e+01
75%    3.000000e+00    ...    4.100000e+01
max    3.000000e+00    ...    2.974000e+03

[8 rows x 23 columns]
```

These are fairly large DataFrames. Combined they represent around 13 million rows.

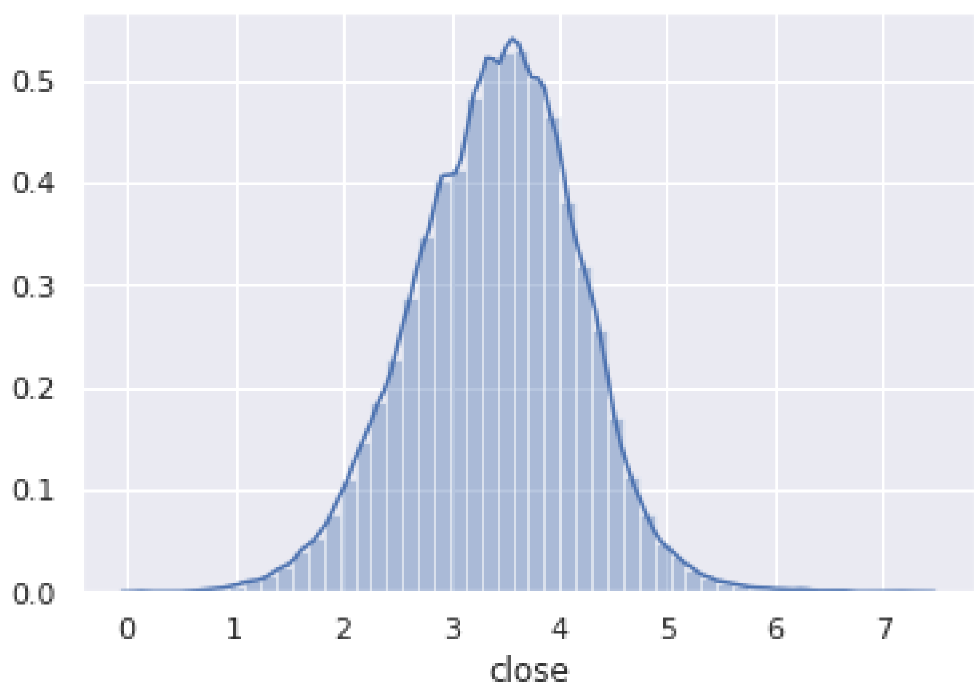
3 EDA

In [9]:
```
market_train_df['returnsOpenNextMktres10'].isnull().sum()
```

Out[9]:
```
0
```

No rows are missing a ground truth label.

In [10]:
```
ax = sns.distplot(np.log1p(market_train_df['close']))
```



Here we can see the distribution of the closing price of all the assets. As expected this feature is slightly right skewed and nearly normally distributed. This is a common trait among stocks.

In [11]:
```
# Select 5 random companies
randoms = market_train_df.sample(5, random_state=33)
```

With 13+ million rows and thousands of unique assets, it might be challenging to graphically represent this data. Instead, I am going to explore 5 randomly selected assets.

3.1 Closing Prices of 5 Random Assets

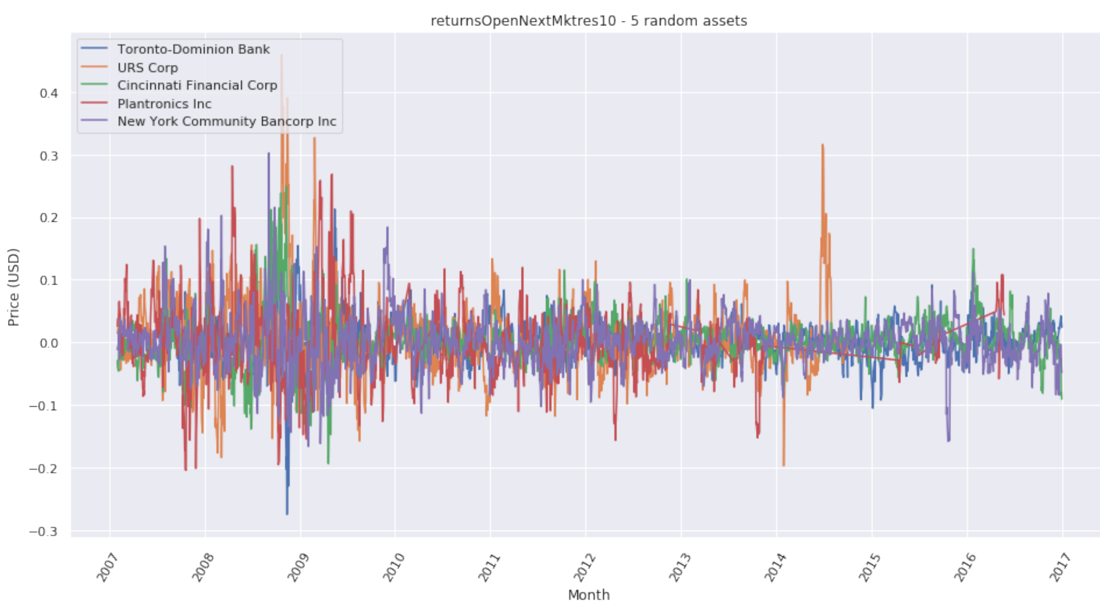
```
In [14]: data = []

         for asset in randoms['assetName']:
             asset_df = market_train_df[(market_train_df['assetName'] == asset)]
             asset_df['date'] = pd.to_datetime(asset_df['time']).dt.strftime(date_format='%Y-%m
             asset_df = asset_df.set_index('time')

             plt.plot(asset_df['close'], label=asset)

         plt.title('Closing prices - 5 random assets')
         plt.xlabel('Month')
         plt.ylabel('Price (USD)')
         plt.legend(loc='upper left')
         plt.xticks(rotation=60)
         plt.rcParams["figure.figsize"] = (16,8)
         plt.show()
```

This is an interesting chart and illustrates exactly what I was hoping to capture. Financial markets are vast and confusing landscapes that likely require large amounts of specific domain knowledge before a predictive strategy can become lucrative. Take a look at the Toronto-Dominion Bank asset around 2014. It appears as if the closing price dropped in half overnight. Does this mean the company is doing really poorly? No. This discrepancy is the direct result of a stock split. A quick google search verified my suspicion. Taking events like market crashes, flash crashes, stock splits etc. into consideration is likely very important and requires a degree of domain knowledge.

3.2 returnsOpenNextMktres10 of 5 Random Assets

The *returnsOpenNextMktres10* feature represents a 10 day, market-residualized return

```
In [15]: data = []

         for asset in randoms['assetName']:
             asset_df = market_train_df[(market_train_df['assetName'] == asset)]
             asset_df['date'] = pd.to_datetime(asset_df['time']).dt.strftime(date_format='%Y-%m
             asset_df = asset_df.set_index('time')

             plt.plot(asset_df['returnsOpenNextMktres10'], label=asset)

         plt.title('returnsOpenNextMktres10 - 5 random assets')
         plt.xlabel('Month')
         plt.ylabel('Price (USD)')
         plt.legend(loc='upper left')
         plt.xticks(rotation=60)
         plt.show()
```

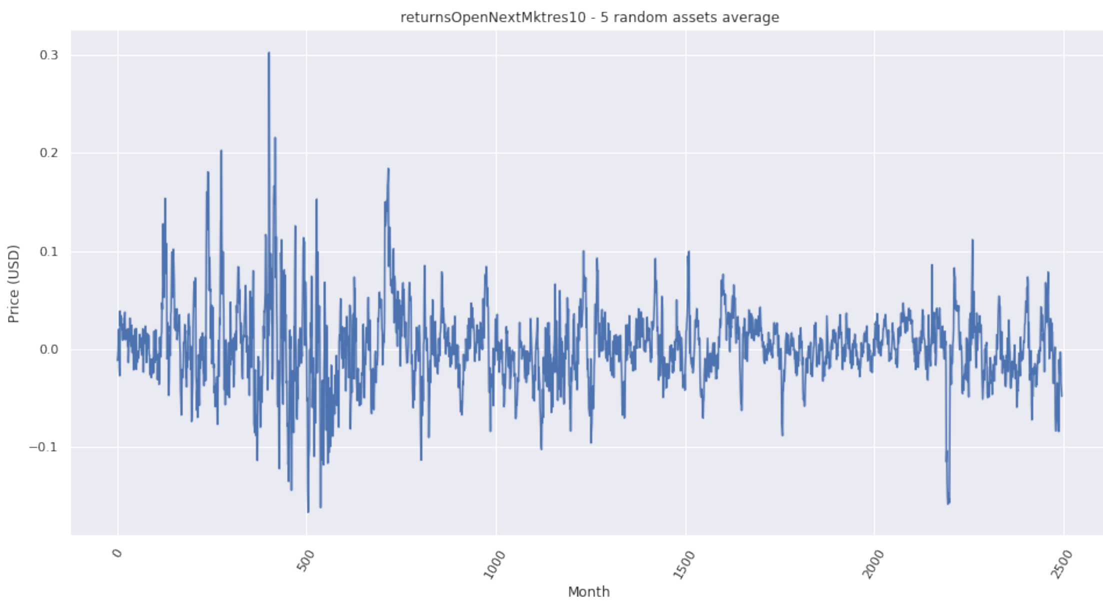


This is a little messy and hard to identify a trend or anything immediately useful. While there appears to be quite a bit of variance, these fluctuations seem consistent among all of the assets.

3.3 returnsOpenNextMktres10 Average of 5 Random Assets

```
In [16]: price_df = asset_df.groupby('time')['returnsOpenNextMktres10'].mean().reset_index()

         price_df['time'] = pd.to_datetime(price_df['time']).dt.strftime(date_format='%Y-%m-%d
         price_df = price_df.set_index('time')
```

```
plt.plot(price_df['returnsOpenNextMktres10'].values)

plt.title('returnsOpenNextMktres10 - 5 random assets average')
plt.xlabel('Month')
plt.ylabel('Price (USD)')
plt.xticks(rotation=60)
plt.show()
```

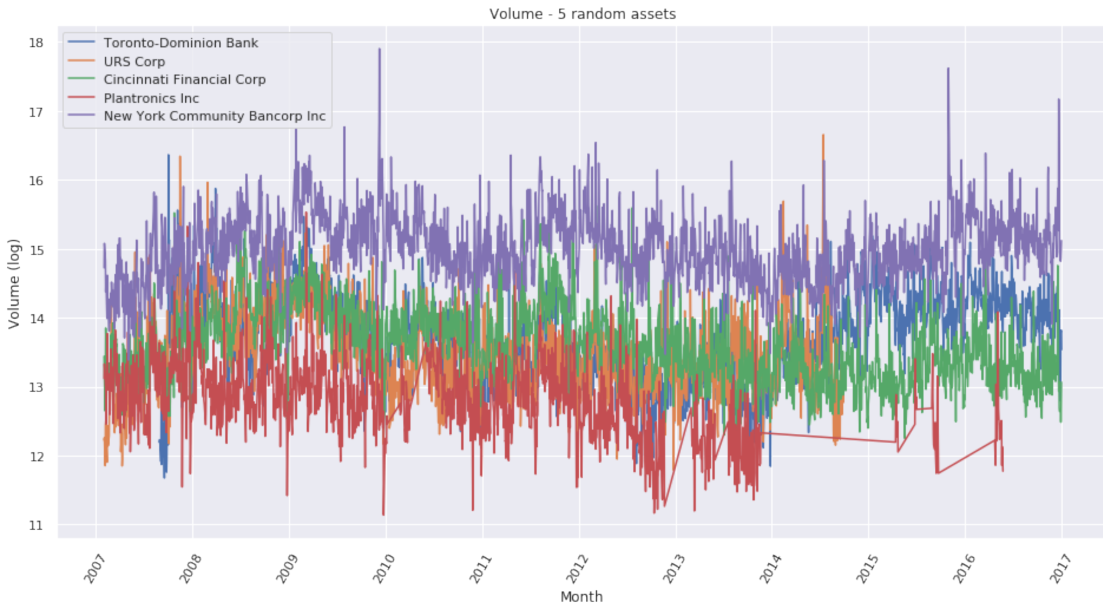


While examining the average *returnsOpenNextMktres10* of all 5 assets is a little cleaner, there still doesn't seem to be a whole lot to learn from this, aside from the fact that it fluctuates quite significantly. It reminds me of the famous quote by J.P. Morgan, "It will fluctuate".

3.4 Volume of 5 Random Assets

```
In [17]: data = []

         for asset in randoms['assetName']:
             asset_df = market_train_df[(market_train_df['assetName'] == asset)]
             asset_df['date'] = pd.to_datetime(asset_df['time']).dt.strftime(date_format='%Y-%
             asset_df = asset_df.set_index('time')

             plt.plot(np.log1p(asset_df['volume']), label=asset)

         plt.title('Volume - 5 random assets')
         plt.xlabel('Month')
         plt.ylabel('Volume (log)')
         plt.legend(loc='upper left')
```

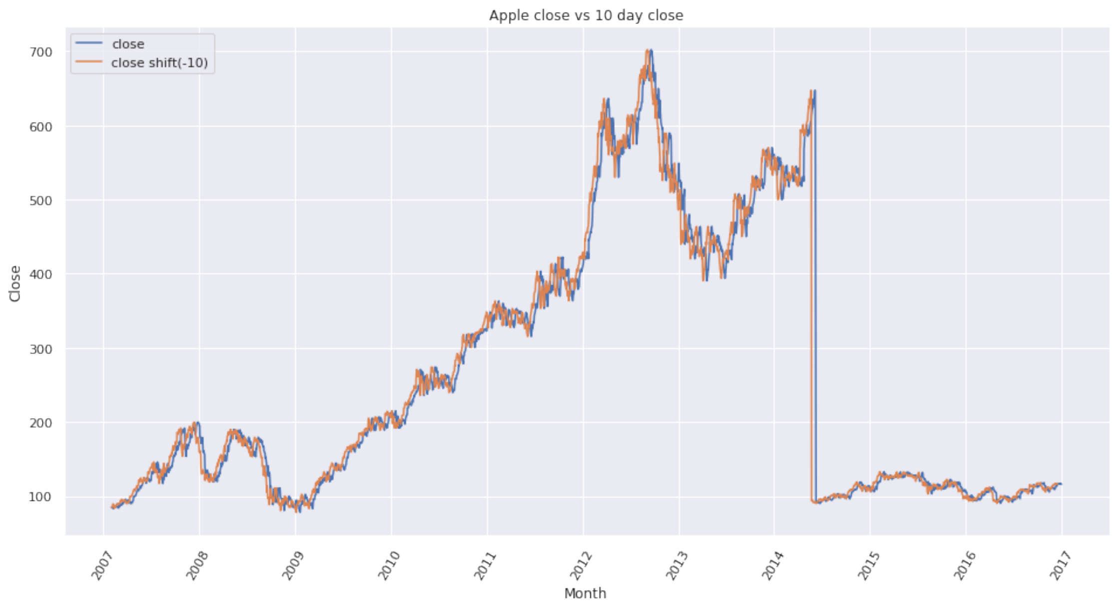
```
plt.xticks(rotation=60)
plt.show()
```



More seemingly stochastic fluctuations. I found something interesting in this plot though. While examining the asset with the highest levels of volume and the high associated variance, I noticed that it was the same asset that had the lowest closing prices and the least amount of variance in the closing price plot shown earlier. It would seem that volume and the price of an asset is uncorrelated. I validte this theory in the next notebook when I create a correlation matrix on my features.

3.5 Predicting Future Prices from Historical Prices

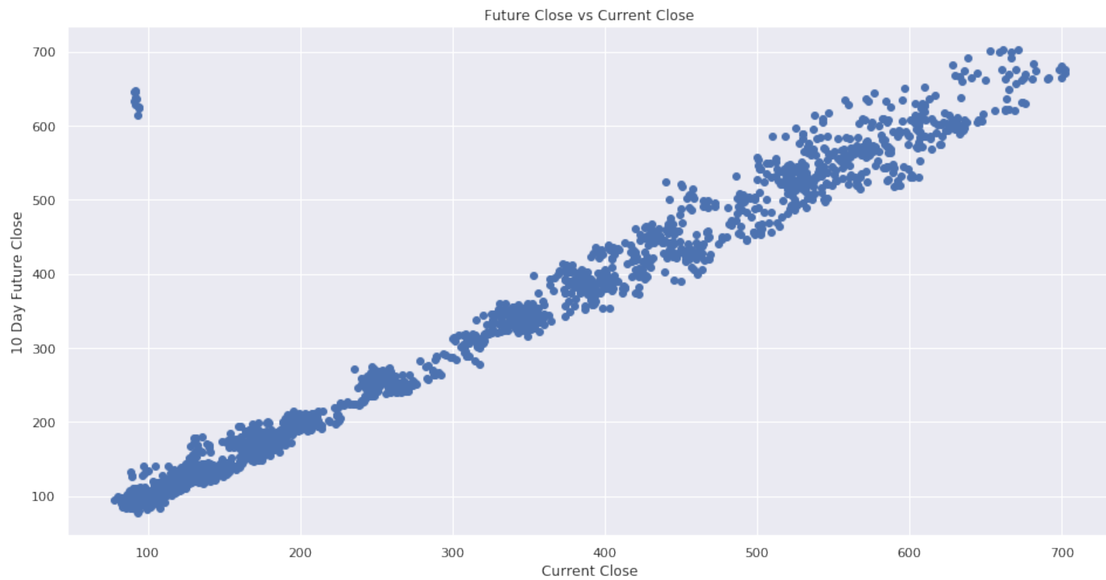
```
In [18]: apple_df = market_train_df[market_train_df['assetName'] == 'Apple Inc']
         apple_df['10d_future_close'] = apple_df['close'].shift(-10)
         apple_df['date'] = pd.to_datetime(apple_df['time']).dt.strftime(date_format='%Y-%m-%d
         apple_df = apple_df.set_index('time')

         plt.plot(apple_df['close'], label='close')
         plt.plot(apple_df['10d_future_close'], label='close shift(-10)')

         plt.title('Apple close vs 10 day close')
         plt.xlabel('Month')
         plt.ylabel('Close')
         plt.legend(loc='upper left')
         plt.xticks(rotation=60)
         plt.show()
```



Here we can see a current closing price and a closing price for 10 days in the past. This allows us to look into the future and calculate percentage changes.

```
In [19]: apple_df = market_train_df[market_train_df['assetName'] == 'Apple Inc']
         apple_df['10d_future_close'] = apple_df['close'].shift(-10)

         corr = apple_df[['10d_future_close', 'close']].corr()
         print(corr)

         # Scatter the current 10-day change
         plt.scatter(apple_df['10d_future_close'], apple_df['close'])
         plt.title('Future Close vs Current Close')
         plt.xlabel('Current Close')
         plt.ylabel('10 Day Future Close')
         plt.show()
```

```
                  10d_future_close     close
10d_future_close          1.000000  0.975551
close                     0.975551  1.000000
```



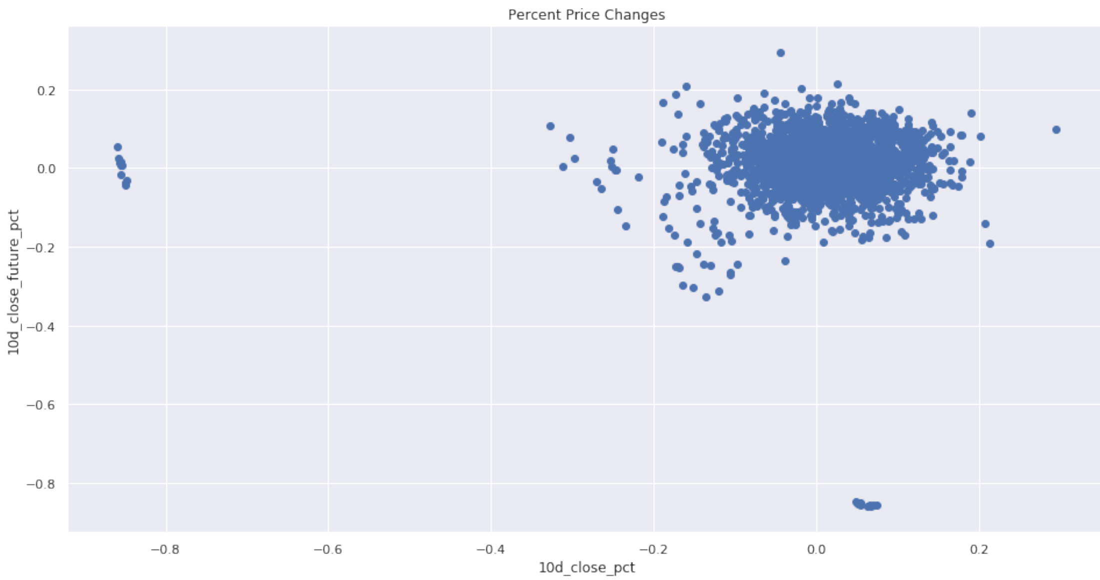
While this is highly correlated and looks like we should be able to predict future prices based on past prices. This is actually just a mirage! The range of future prices compared to current prices is simply too large.

3.6 Percent Price Changes

Let's see if previous percent price changes can be used to predict future percent price changes.

```
In [20]: # 10-day perent price changes
         apple_df['10d_future_close'] = apple_df['close'].shift(-10)
         apple_df['10d_close_future_pct'] = apple_df['10d_future_close'].pct_change(10)
         apple_df['10d_close_pct'] = apple_df['close'].pct_change(10)

         # Calculate the correlation
         corr = apple_df[['10d_close_pct', '10d_close_future_pct']].corr()
         print(corr)

         plt.scatter(apple_df['10d_close_pct'], apple_df['10d_close_future_pct'])
         plt.title('Percent Price Changes')
         plt.xlabel('10d_close_pct')
         plt.ylabel('10d_close_future_pct')
         plt.show()
```

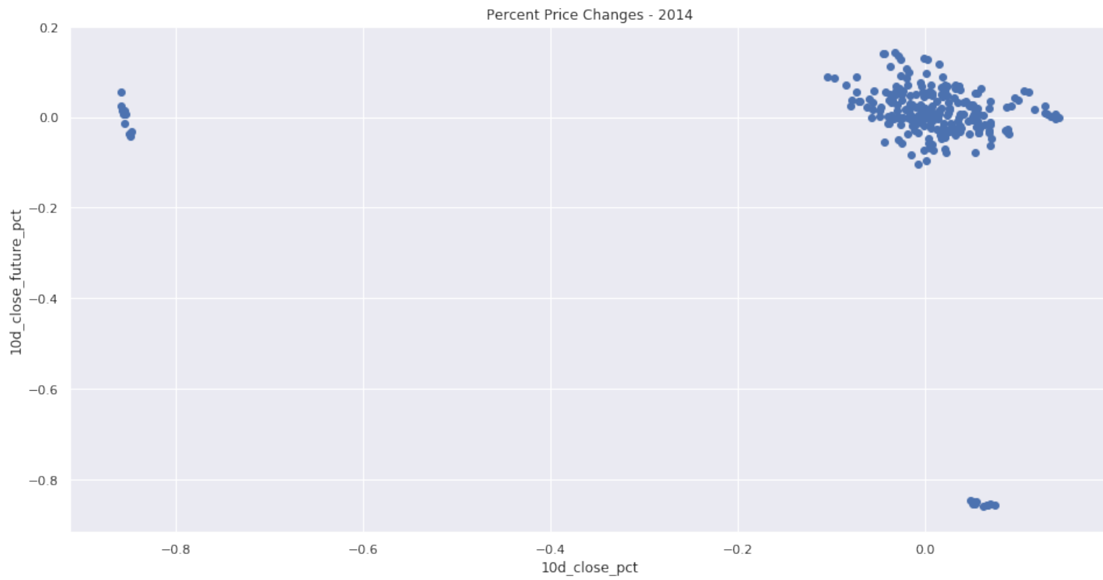
```
                      10d_close_pct  10d_close_future_pct
10d_close_pct              1.000000              0.012574
10d_close_future_pct       0.012574              1.000000
```



This looks a lot different than the correlation between closing price and future closing price. With the exception of a few, it seems like the vast majority of points fall around the 0,0 mark, indicating low correlation. This might be too high level to identify patterns or correlations. Let's drill down and examine a few specific years.

In [21]:
```
# Calculate the correlation
corr = apple_df[apple_df['time'].dt.year == 2014][['10d_close_pct', '10d_close_future
print(corr)

plt.scatter(apple_df[apple_df['time'].dt.year == 2014]['10d_close_pct'], apple_df[app
plt.title('Percent Price Changes - 2014')
plt.xlabel('10d_close_pct')
plt.ylabel('10d_close_future_pct')
plt.show()
```

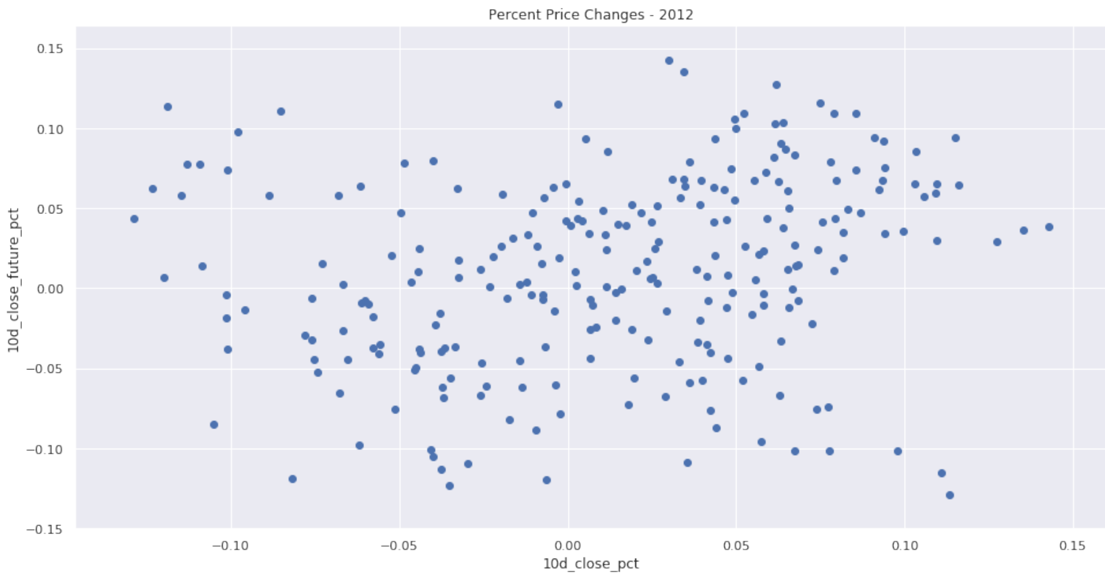
```
                    10d_close_pct  10d_close_future_pct
10d_close_pct            1.000000             -0.090782
10d_close_future_pct    -0.090782              1.000000
```



It seems like the year 2014 is generally consistent with what we observed over all of the data. What happens if we try another year?

```
In [22]: # Calculate the correlation
         corr = apple_df[apple_df['time'].dt.year == 2012][['10d_close_pct', '10d_close_future_
         print(corr)

         plt.scatter(apple_df[apple_df['time'].dt.year == 2012]['10d_close_pct'], apple_df[appl
         plt.title('Percent Price Changes - 2012')
         plt.xlabel('10d_close_pct')
         plt.ylabel('10d_close_future_pct')
         plt.show()
```

```
                      10d_close_pct  10d_close_future_pct
10d_close_pct              1.000000              0.211207
10d_close_future_pct       0.211207              1.000000
```



This plot looks very different from the two above and in doing so, uncovers an interesting observation. When examining the year 2012, there appears to be a correlation of 0.21 between the current and future percentage price changes. This could be indicative of percentage price changes being a valuable feature at certain points in time.

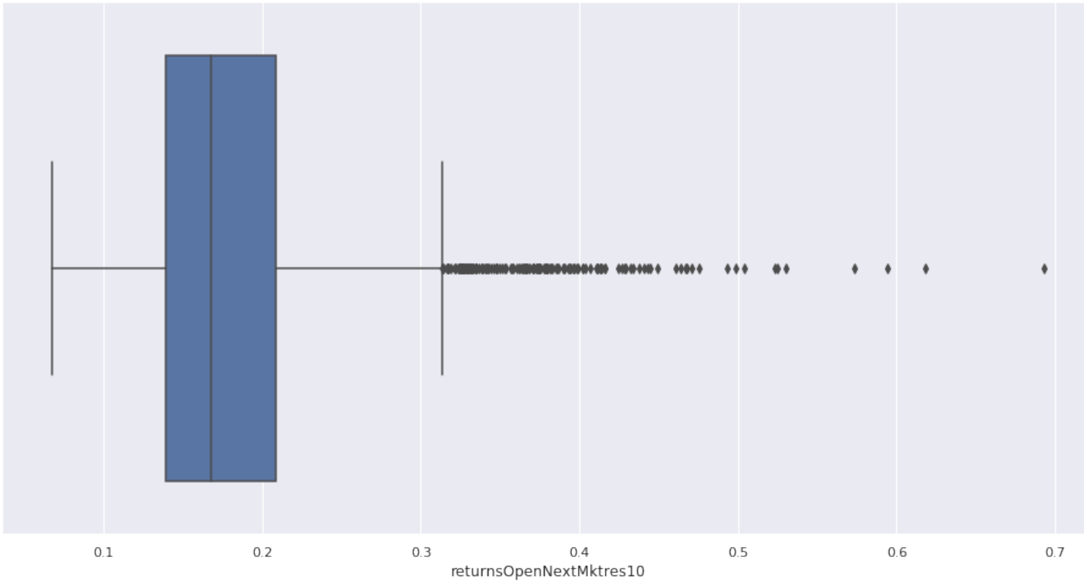
3.7 Outliers

```
In [23]: q1_df = market_train_df.groupby('time')['returnsOpenNextMktres10'].quantile(0.01).rese
         q99_df = market_train_df.groupby('time')['returnsOpenNextMktres10'].quantile(0.99).res
```

```
In [24]: print(f'Min q1: {q1_df["returnsOpenNextMktres10"].min()}')
         print(f'Max q99: {q99_df["returnsOpenNextMktres10"].max()}')
```

```
Min q1: -0.4941219830456521
Max q99: 0.6932446723840386
```

```
In [25]: ax = sns.boxplot(x=q99_df['returnsOpenNextMktres10'])
```



Above is the boxplot of the 99th percentile for *returnsOpenNextMktres10*. we can see that there are quite a few outliers. Outliers can be somewhat of a subjective topic and I imagine they are rampant in the financial industry.

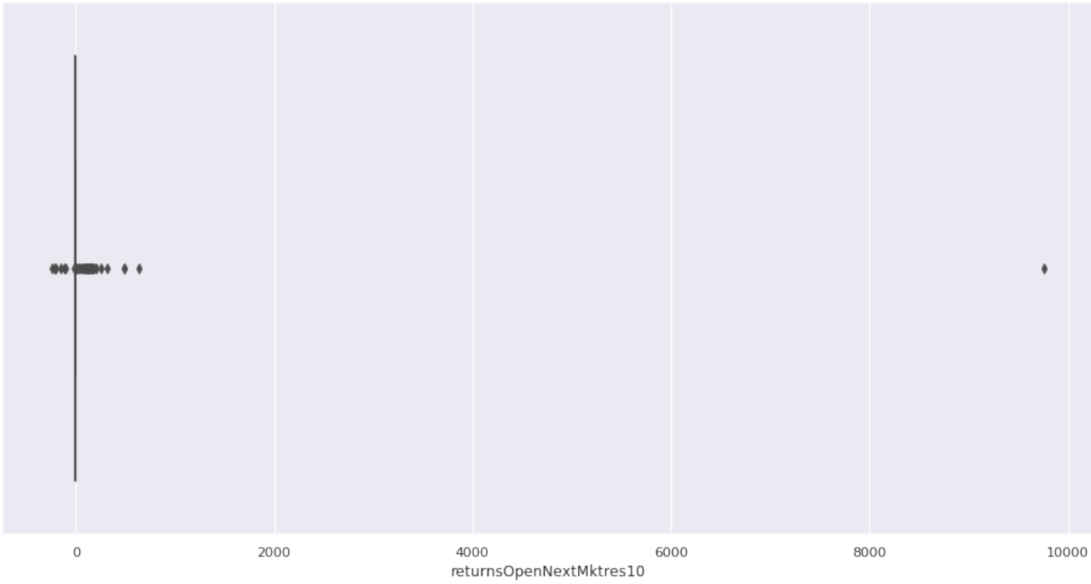
Let's inspect the asset with the highest *returnsOpenNextMktres10* and see how far outside of the 99th percentile it lies.

```
In [26]: market_train_df[market_train_df['returnsOpenNextMktres10'] == market_train_df['returns
```

```
Out[26]:                                 time  ...  universe
         91022 2007-05-02 22:00:00+00:00  ...       1.0

         [1 rows x 16 columns]
```

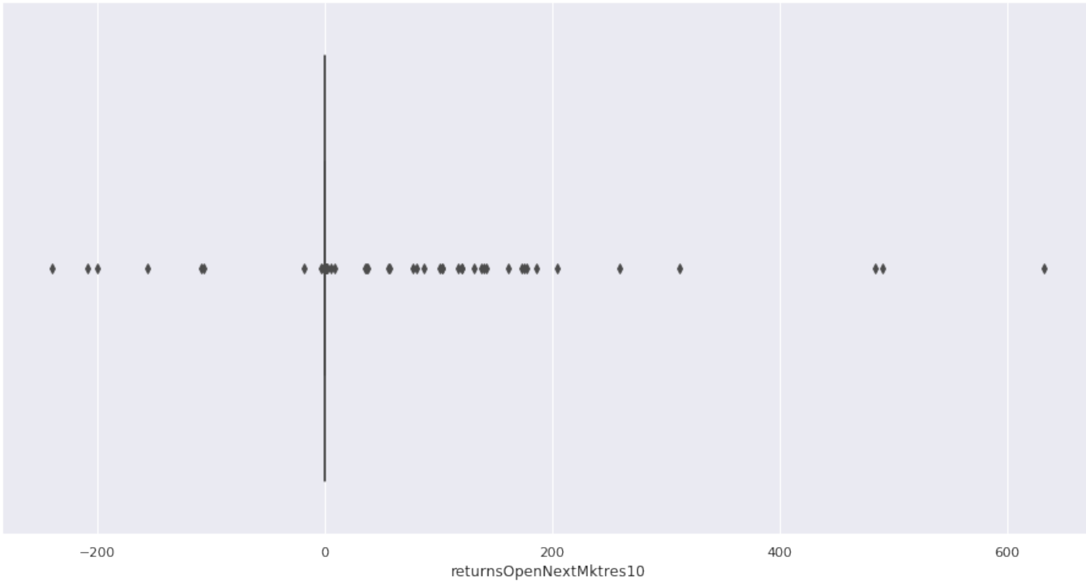
With a mean of 0.01 for the *returnsOpenNextMktres10* feature, it seems kind of surprising that there would be a value of 9761 that exists. That's 976,100 times larger than the average.

```
In [27]: max_asset = market_train_df[market_train_df['assetName'] == market_train_df[market_tra
         ax = sns.boxplot(x=max_asset['returnsOpenNextMktres10'])
```



Data points like this seem extreme and can probably be safely dropped. It would be interesting to talk to a professional in the financial industry and get their take on "outliers".

```
In [28]: max_df = market_train_df[market_train_df['assetName'] == market_train_df[market_train_
         max_df = max_df.set_index('time')

         drop_max = max_df[max_df['returnsOpenNextMktres10'] != max_df['returnsOpenNextMktres1
         ax = sns.boxplot(x=drop_max['returnsOpenNextMktres10'])
```



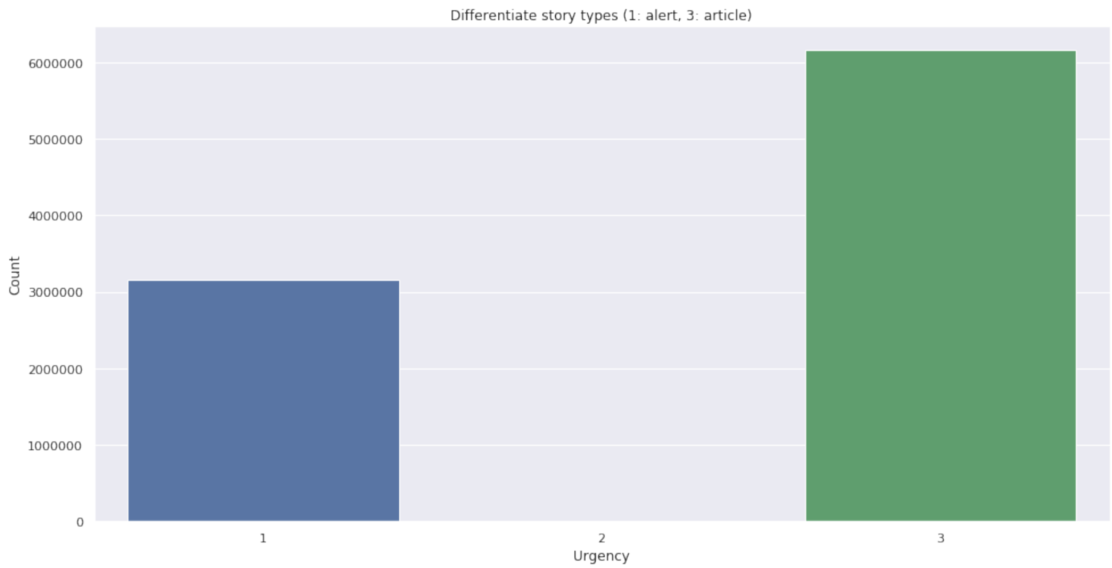
The result of dropping the outlier.

4 News EDA

Now I am going to examine the news data.

```
In [29]: news_train_df = news_train_df.set_index('time')
```

```
In [30]: _ = sns.barplot(news_train_df['urgency'].value_counts().index, news_train_df['urgency
         _ = plt.xlabel('Urgency')
         _ = plt.ylabel('Count')
         _ = plt.title('Differentiate story types (1: alert, 3: article)')
```




```
In [31]: news_train_df['urgency'].value_counts()
```

```
Out[31]: 3    6162567
         1    3166158
         2         25
         Name: urgency, dtype: int64
```

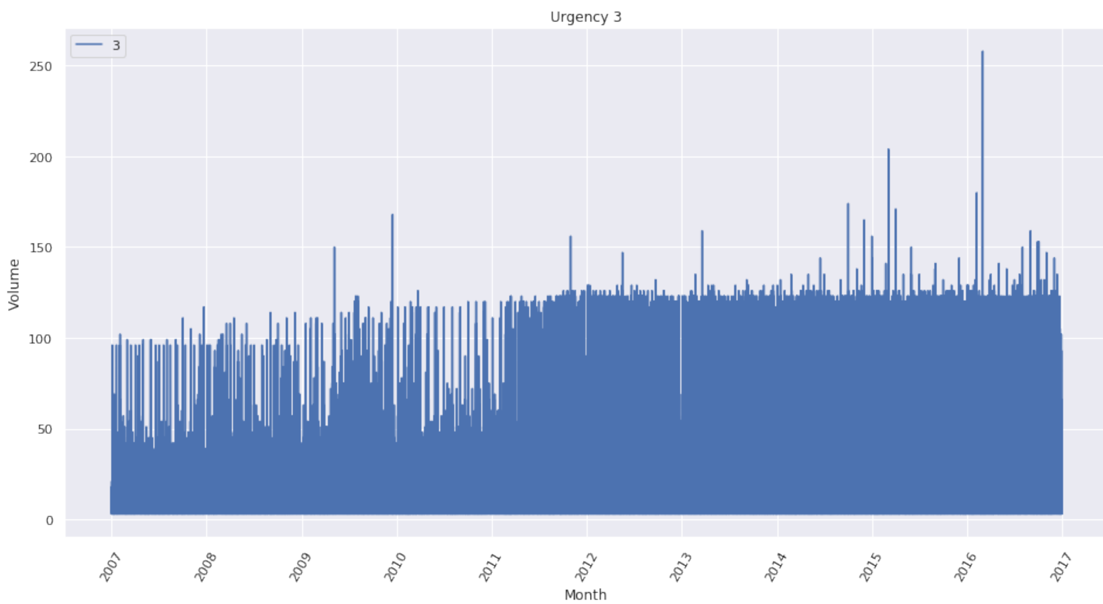
The *Urgency* feature differentiates story types (1: alert, 3: article). Here we can see the vast majority of news data are classified as articles, with about 50% of that number being classified as alerts. The classification of 2 is nearly nonexistent and was not described in the dataset.

4.1 Let's examine the frequency of the urgency categories

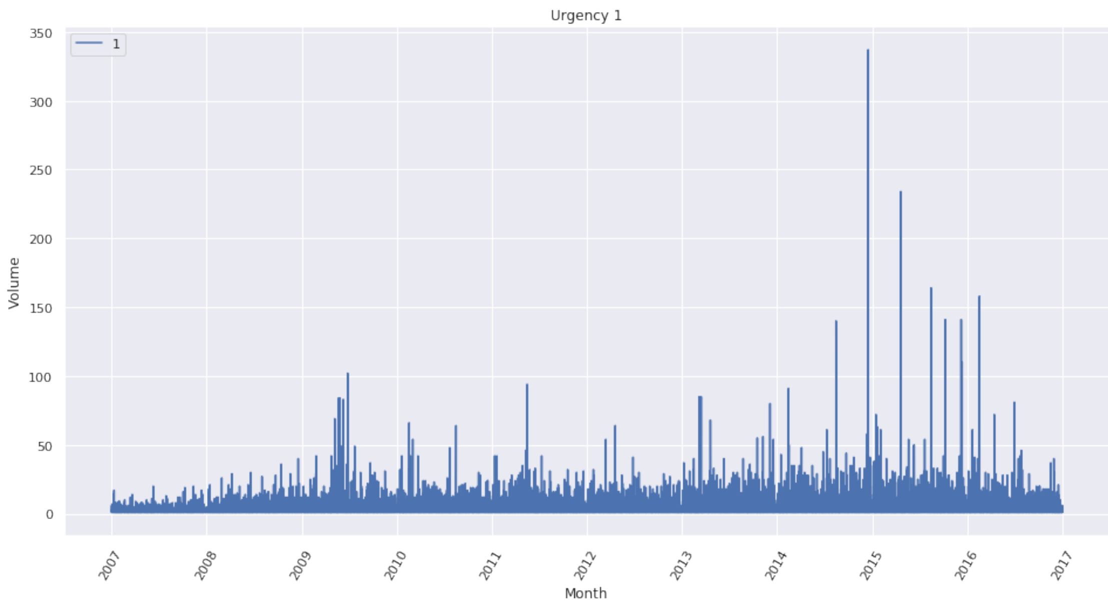
```
In [32]: grouped_3 = news_train_df[news_train_df['urgency'] == 3].groupby(news_train_df[news_t
         grouped_1 = news_train_df[news_train_df['urgency'] == 1].groupby(news_train_df[news_t
```

```
In [33]: plt.plot(grouped_3['urgency'].agg(np.sum), label='3')
```

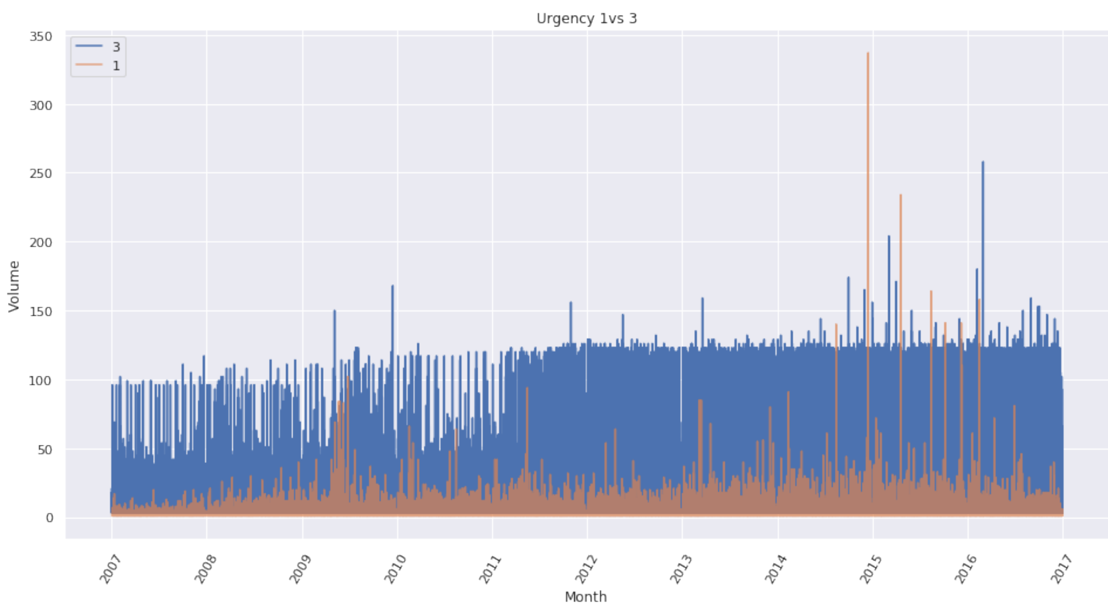
```
plt.title('Urgency 3')
plt.xlabel('Month')
plt.ylabel('Volume')
plt.legend(loc='upper left')
plt.xticks(rotation=60)
plt.show()
```




```
In [34]: plt.plot(grouped_1['urgency'].agg(np.sum), label='1')

         plt.title('Urgency 1')
         plt.xlabel('Month')
         plt.ylabel('Volume')
         plt.legend(loc='upper left')
         plt.xticks(rotation=60)
         plt.show()
```



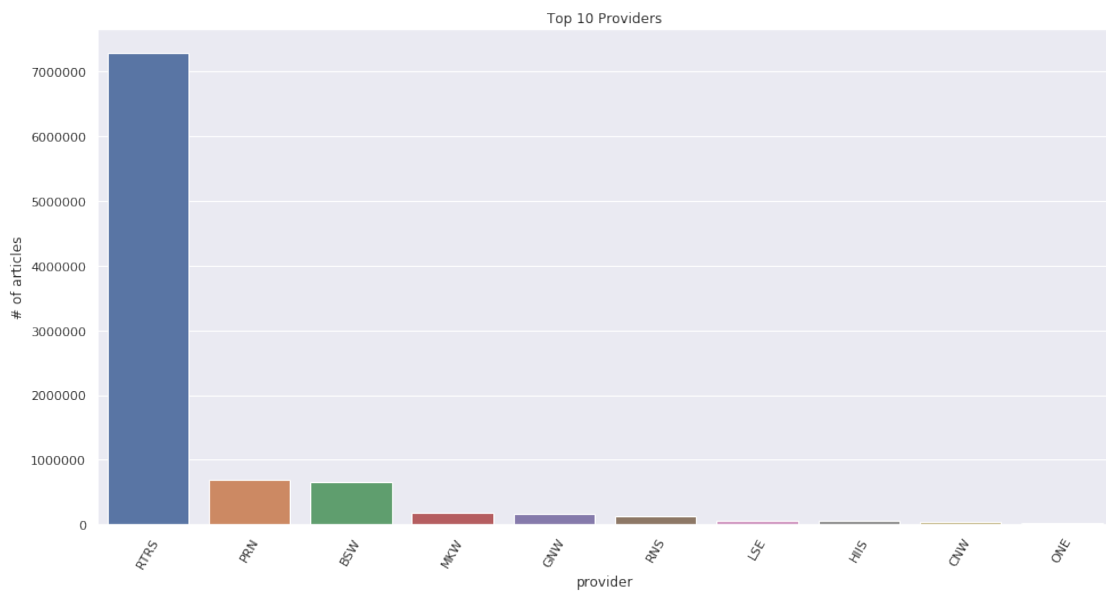
In [35]:
```
plt.plot(grouped_3['urgency'].agg(np.sum), label='3')
plt.plot(grouped_1['urgency'].agg(np.sum), label='1', alpha=0.7)

plt.title('Urgency 1vs 3')
plt.xlabel('Month')
plt.ylabel('Volume')
plt.legend(loc='upper left')
plt.xticks(rotation=60)
plt.show()
```



We can see a trend of the plot becoming denser as we move towards more modern years. This seems understandable considering the rate at which we are generating data today.

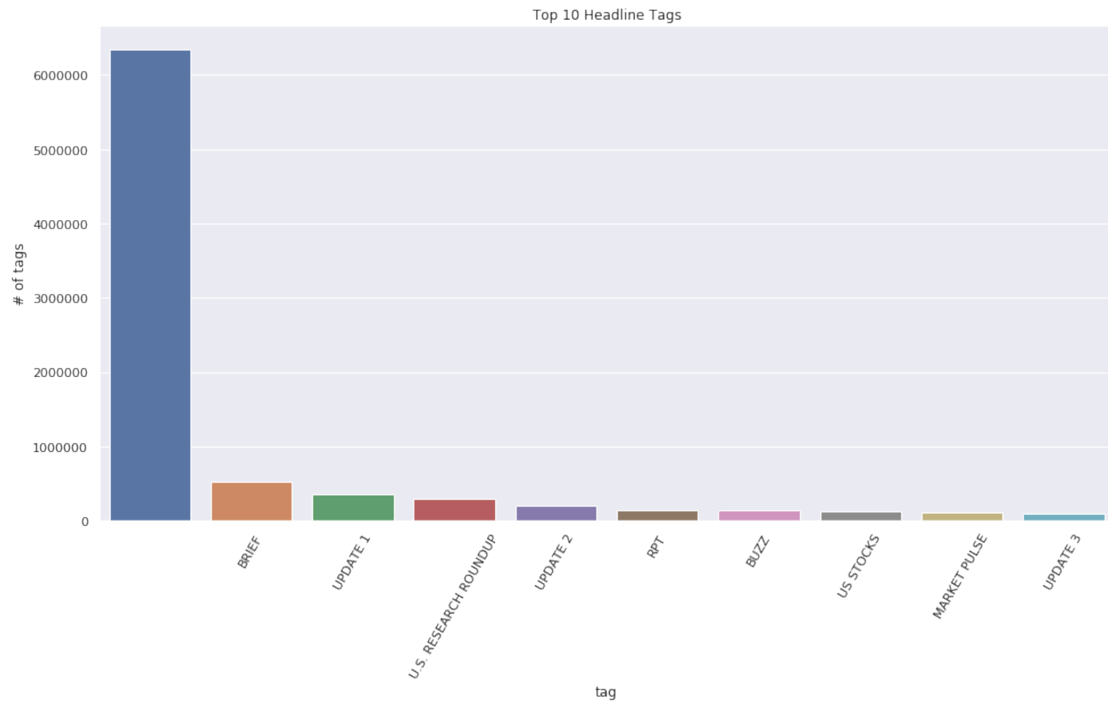
Basically, what I was trying to identify was clear signs of increases/decreases in alerts. For example, did the number of alerts in relation to articles increase leading up to the 2008 crash? There are evident spikes in the volume of alerts beings produced and this could prove to be an insightful feature.

```
In [36]: _ = sns.barplot(np.array(news_train_df['provider'].value_counts().index[0:10]), news_
         _ = plt.title('Top 10 Providers')
         _ = plt.xlabel('provider')
         _ = plt.ylabel('# of articles')
         _ = plt.xticks(rotation=60)
```



Reuters News is the most popular news provider by a factor of magnitude. No big surprise here.

```
In [37]: _ = sns.barplot(news_train_df['headlineTag'].value_counts()[0:10].index, news_train_d
         _ = plt.title('Top 10 Headline Tags')
         _ = plt.xlabel('tag')
         _ = plt.ylabel('# of tags')
         _ = plt.xticks(rotation=60)
```



A substantial number of news articles are tagless. Again, no big surprise here.

4.2 Analyzing Sentiment

sentimentClass - indicates the predominant sentiment class for this news item with respect to the asset. The indicated class is the one with the highest probability.

sentimentNegative - probability that the sentiment of the news item was negative for the asset
sentimentNeutral - probability that the sentiment of the news item was neutral for the asset
sentimentPositive - probability that the sentiment of the news item was positive for the asset

In [38]:
```
grouped_news = news_train_df.groupby('assetName')
```

4.2.1 Most Negative

In [39]:
```
grouped_news['sentimentClass'].agg(np.sum).sort_values(ascending=True)[0:20]
```

Out[39]:
```
assetName
Citigroup Inc                              -12798.0
Bank of America Corp                       -12291.0
JPMorgan Chase & Co                        -11927.0
BP PLC                                      -9674.0
Goldman Sachs Group Inc                     -9023.0
UBS AG                                      -7262.0
Motors Liquidation Co                       -6249.0
Federal Home Loan Mortgage Corp             -5791.0
Latam Airlines Group SA                     -5457.0
```

```
American International Group Inc         -5136.0
Federal National Mortgage Association    -5026.0
Morgan Stanley                           -4987.0
Royal Bank of Scotland Group PLC         -4804.0
Toyota Motor Corp                        -4712.0
HSBC Holdings PLC                        -4576.0
Exxon Mobil Corp                         -4468.0
Credit Suisse AG                         -4275.0
Transocean Ltd                           -3966.0
Credit Suisse Group AG                   -3945.0
Apple Inc                                -3932.0
Name: sentimentClass, dtype: float64
```

4.2.2 Most Positive

```
In [40]: grouped_news['sentimentClass'].agg(np.sum).sort_values(ascending=False)[0:20]
```

```
Out[40]: assetName
General Electric Co                     8566.0
Verizon Communications Inc              8349.0
AT&T Inc                                7373.0
Ball Corp                               6558.0
Barclays PLC                            6258.0
Anheuser Busch Inbev SA                 5901.0
Rio Tinto PLC                           4959.0
Aviva PLC                               4653.0
Bank of Montreal                        4609.0
Royal Dutch Shell PLC                   4479.0
International Business Machines Corp    4468.0
Steris Corp                             4452.0
Shire PLC                               4214.0
Invesco Ltd                             4020.0
BHP Billiton PLC                        3386.0
Equinix Inc                             3303.0
Comcast Corp                            3068.0
ARRIS Group Inc                         2987.0
Blackstone Group LP                     2723.0
Vodafone Group PLC                      2707.0
Name: sentimentClass, dtype: float64
```

```
In [41]: for i, j in zip([-1, 0, 1], ['negative', 'neutral', 'positive']):
             df_sentiment = news_train_df.loc[news_train_df['sentimentClass'] == i, 'assetName
             print(f'Top mentioned companies for {j} sentiment are:')
             print(f'{df_sentiment.value_counts().head(5)} \n')
```

```
Top mentioned companies for negative sentiment are:
Citigroup Inc           30823
JPMorgan Chase & Co     29129
Bank of America Corp    28197
```

```
Apple Inc                  26702
Goldman Sachs Group Inc    25044
Name: assetName, dtype: int64

Top mentioned companies for neutral sentiment are:
Barclays PLC         24898
HSBC Holdings PLC    23191
Deutsche Bank AG     20702
BHP Billiton PLC     18019
Rio Tinto PLC        16782
Name: assetName, dtype: int64

Top mentioned companies for positive sentiment are:
Barclays PLC             22855
Apple Inc                22770
General Electric Co      20055
Royal Dutch Shell PLC    18206
Citigroup Inc            18025
Name: assetName, dtype: int64
```

It appears that big banks like Citigroup, Bank of America, JPMorgan Chase & Co and Goldman are the most negatively viewed. I was somewhat surprised to see Apple in this category.

It was interesting to see Barclays, a big bank, at the top of the most positively viewed list. Especially considering it was mainly big banks at the top of the negative list. The main difference here is that Barclays is a British based bank.

Probably the most interesting insight here is that Apple is on both the most positively and negatively viewed lists. This could mean that most people take an extreme view towards Apple. They either love the company or they hate them.