Alternative Data in Investment and Tading

Julian Kaljuvee / DataScrum.co.uk

LondonAI Meetup September 2019

DataScrum - Overview

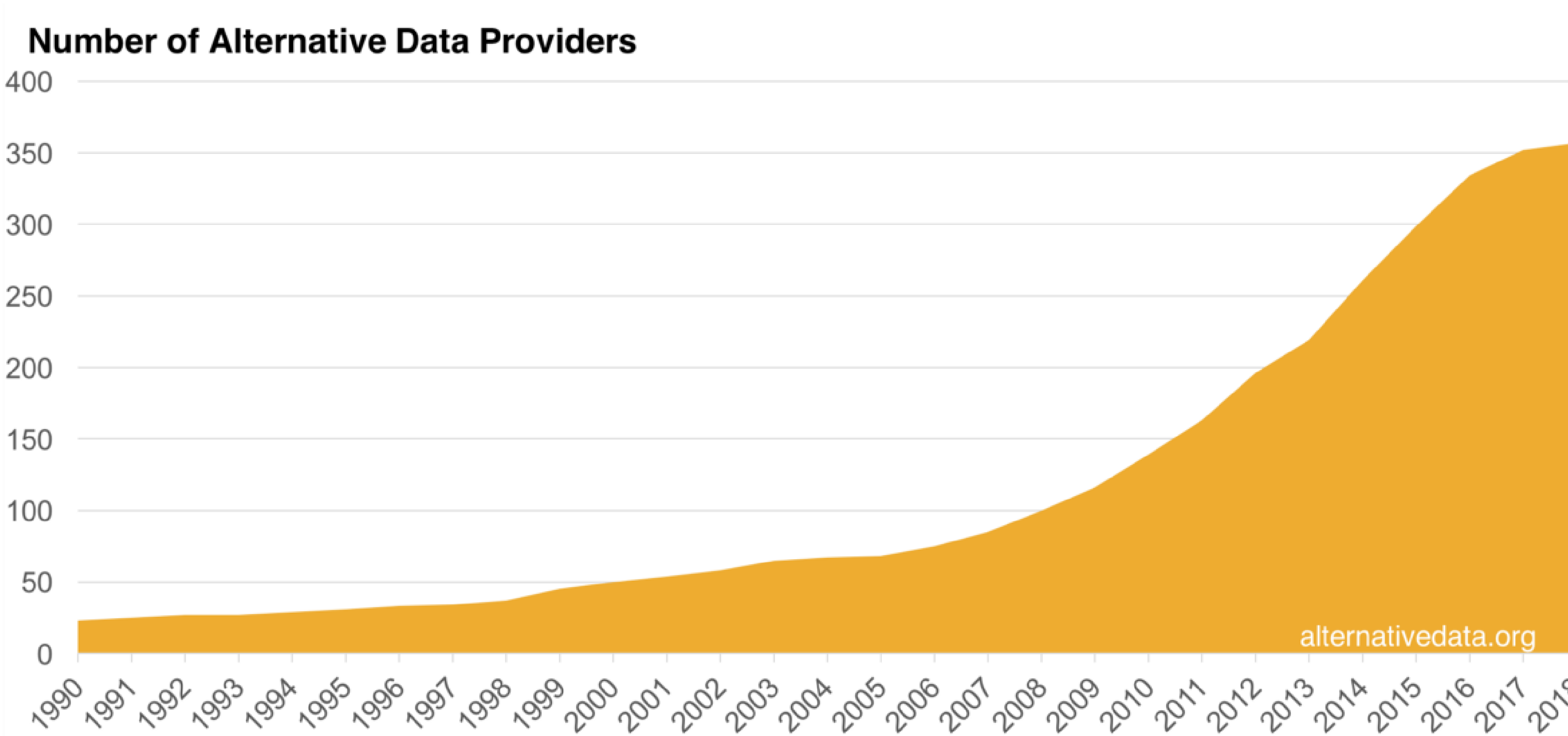
DataScrum – an alternative data focused meetup (every 4-6 weeks) bringing together a community of:

- **Alternative data providers -** looking to educate the audience about their products
- **Alternative data users** – such as investment managers and insurance companies looking to gain an edge and to improve investment performance based on alternative data
- **Data Scientists** – looking to learn more about leveraging alternative data in financial models and beyond
- **Data Science consultancies and specialists recruiters** - looking to provider alternative data solutions and resources

Focus is on information sharing via high quality speakers and content and we will also aim to organise informal mixers and hackathons

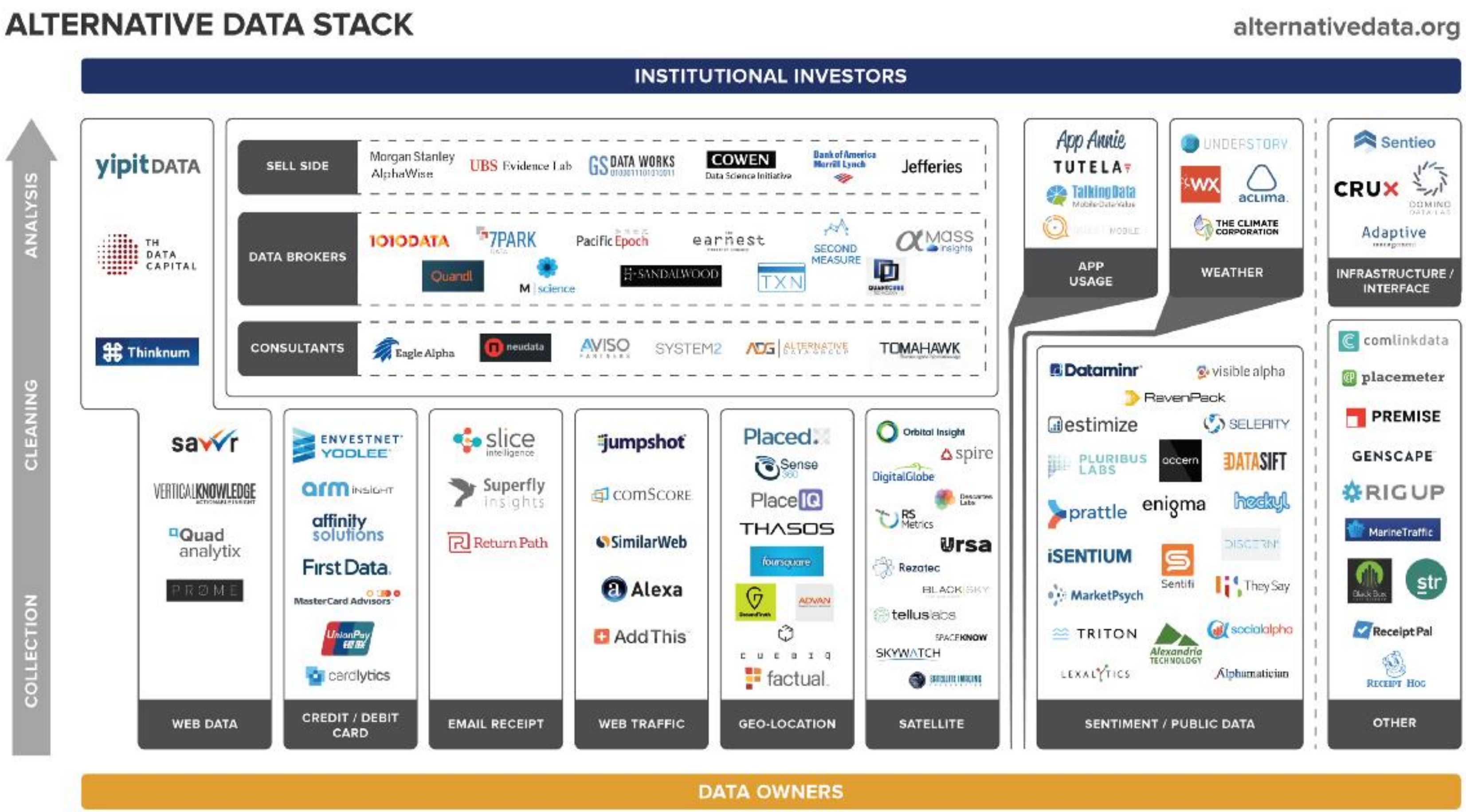
Growth of Alternative Data

- Growth is been exponential over past decade, primarily in the US but recently also growing in Europe and Asia



+ Alternative data stack

- Large number (400+) data providers globally



+
What is Alternative Data in Financial Markets?

- **Definition** – all **non-financial data**, not generated by the trading activity in the markets, and / or not directly reported by companies
- **Individuals** - social/sentiment, web traffic, app usage, survey
- **Business Processes** - credit/debit card, web data, public Data, email/consumer Receipts
- **Sensors** - geo-location, satellite , weather
- **Data cost** - typically Business Processes > Sensors > Individuals

Preferences by buyers

- **Data source with the greatest number of providers** – social and sentiment
- **Highest grossing data source** – credit and debit card
- **Most utilized datasets** - web data, credit and debit card
- **Most insightful datasets** – credit and debit card, web data
- **Least insightful datasets -** geo-location and satellite

+ So what? Show me the trade!

Overview – Sample Trading Strategy

A sample strategy could look as follows:

- **Model inputs** – historical realised quarterly sales (dependent variable / label) vs web traffic
- **Objective –** predict / forecast sales from the web traffic data (potentially augmented with app usage, social and card transaction data)
- **Model form** – start with OLS but try more complex models, eg GLM, random forrest etc. It can also become multi class problem (eg also predict earnings per share, not only sales)
- **Market consensus –** compare your prediction against earnings / sales estimates
- **Strategy -** if our estimate >> consensus, go long, else if our estimate << consensus, go short (the analysts are wrong)

Consensus estimates

Mean Rating	2.54	2.50	2.32	2.32

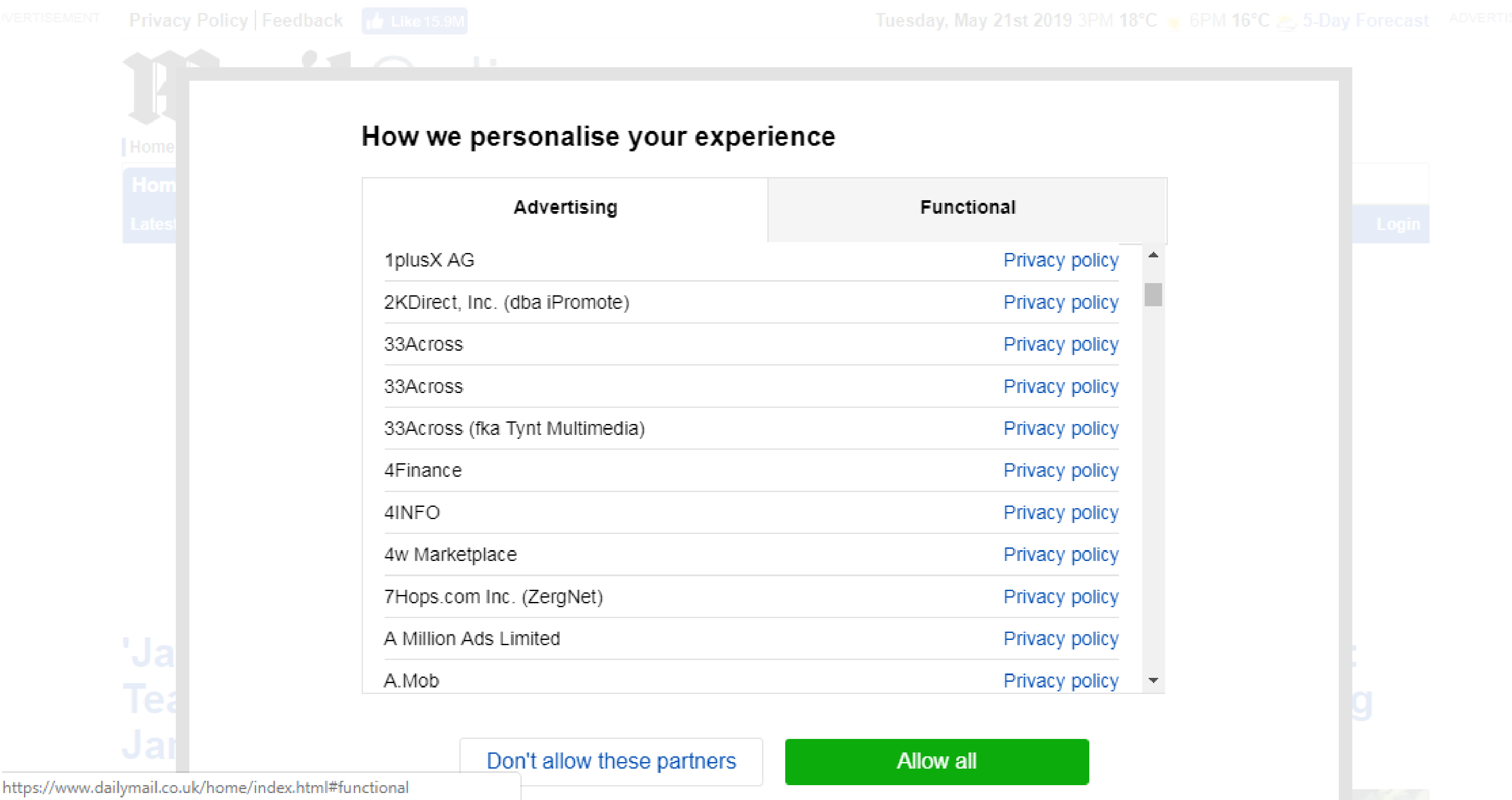
CONSENSUS ESTIMATES ANALYSIS

Sales and Profit Figures in British Pound (GBP)
Earnings and Dividend Figures in British Pound (GBP)

	# of Estimates	Mean	High	Low	1 Year Ago
SALES (in millions)					
Year Ending Aug-19	25	2,783.15	2,808.43	2,752.40	3,051.22
Year Ending Aug-20	25	3,247.88	3,358.10	3,176.26	3,721.94
Earnings (per share)					
Quarter Ending Feb-19	1	34.40	34.40	34.40	--
Quarter Ending Aug-19	1	83.10	83.10	83.10	--
Year Ending Aug-19	23	51.94	59.40	32.43	118.57
Year Ending Aug-20	23	82.11	105.78	57.12	147.41
LT Growth Rate (%)	2	7.18	7.26	7.10	24.84

How do they estimate web traffic?

DailyMail.co.uk has 1200+ companies collecting visitor traffic on their web site. Similar to most other ecommerce web sites



Case Study – Estimating Sales of Booking.com Using Social Media Data

Get Data – Actual Realised Sales

We get quarterly sales (revenues) via eodhistoricaldata.com API

Use EODHistorical API provided to source revenue / sales data for Booking.com (BKNG.US) under the Financial -> Income_Statement -> Quarterly data dictionary keys

The API supports fields filtering with the parameter ‘filter=’. With ability to specify a block, field or Code required within the json data. Also different layers can be divided with “::” and it’s possible to have any number of layers. In our case, we want the quarterly Income_Statement and hence have configured API accordingly.

In [2]:
```
# Specify API url link to for quarterly income statement retrival
eodhist_url = 'http://eodhistoricaldata.com/api/fundamentals/BKNG.US?api_token=5cc0ea63d1cda3.37070012&filter=Financials::Income_
```

In [3]:
```
#Use a combination of pandas json reader to read json data as series
#And normalise with json normaliser and finally retrive the required column fields
fund_df = json_normalize(pd.read_json(eodhist_url,typ='series'))[['date','totalRevenue','costOfRevenue']]
```

In [4]:
```
#Explore the data
print(fund_df.head())
print()
print(fund_df.info())
```

```
         date   totalRevenue costOfRevenue
0  2019-03-31  2837000000.00  501000000.00
1  2018-12-31  3212615000.00  242000000.00
2  2018-09-30  4849090000.00          0.00
3  2018-06-30  3537094000.00          0.00
4  2018-03-31  2928201000.00          0.00
```

Get Features Data – Tweets

We get Twitter data for Booking.com via an open-source Twitter wrapper API and process the Tweets with a basic sentiment analysis library (TextBlob)

Load tweet data extracted from DataScrum database for BKNG.US into a dataframe and process data for senitments

In [6]:
```
#Load tweet data from filesystem
tweet_df = pd.read_csv('C:\\dev\\datascrum\\data\\tweets-bkng.csv', parse_dates = ['date'])
```

In [7]:
```
# define function to be used for tweet senitments analysis
def clean_get_twt_sentiment(tweet):
        '''
        Utility function to clean tweet text by removing links, special characters
        using simple regex statements and to classify sentiment of passed tweet
        using textblob's sentiment method
        '''
        #clean tweet
        clean_tweet = ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)", " ", tweet).split())
        # create TextBlob object of passed tweet text
        analysis = TextBlob(clean_tweet)
        # set sentiment
        if analysis.sentiment.polarity > 0:
            return 'positive'
        elif analysis.sentiment.polarity == 0:
            return 'neutral'
        else:
            return 'negative'
```

Get Data – Get Tweets

We get Twitter data for Booking.com via an open-source Twitter wrapper API and process the Tweets with a basic sentiment analysis library (TextBlob)

Load tweet data extracted from DataScrum database for BKNG.US into a dataframe and process data for senitments

In [6]:
```
#Load tweet data from filesystem
tweet_df = pd.read_csv('C:\\dev\\datascrum\\data\\tweets-bkng.csv', parse_dates = ['date'])
```

In [7]:
```
# define function to be used for tweet senitments analysis
def clean_get_twt_sentiment(tweet):
        '''
        Utility function to clean tweet text by removing links, special characters
        using simple regex statements and to classify sentiment of passed tweet
        using textblob's sentiment method
        '''
        #clean tweet
        clean_tweet = ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)", " ", tweet).split())
        # create TextBlob object of passed tweet text
        analysis = TextBlob(clean_tweet)
        # set sentiment
        if analysis.sentiment.polarity > 0:
            return 'positive'
        elif analysis.sentiment.polarity == 0:
            return 'neutral'
        else:
            return 'negative'
```

Explore Data – Tweets and Sentiment

Raw data comes in at relatively high-frequency (n minute intervals)

Out[8]:

	date	text	likes	retweet	count_comments	ticker
0	2015-04-09 08:45:00	Q5. What activity organised by an accommodatio...	0	0	0	BKNG.O
1	2015-04-09 08:45:00	Q5. What activity organised by an accommodatio...	0	0	0	BKNG.O
2	2015-04-09 08:51:00	Q6 #BookingsBest lists are based on reviews wi...	3	1	5	BKNG.O
3	2015-04-09 08:52:00	#BookingsBest lists are based on reviews with ...	1	1	0	BKNG.O
4	2015-04-09 08:52:00	#BookingsBest lists are based on reviews with ...	0	0	0	BKNG.O

In [9]:

```
#Process the tweet text for senitment indicator
tweet_df['senti']  = tweet_df['text'].apply(clean_get_twt_sentiment)
```

In [10]:

```
#Explore tweet data for senitment indicator
tweet_df.head()
```

Out[10]:

	date	text	likes	retweet	count_comments	ticker	senti
0	2015-04-09 08:45:00	Q5. What activity organised by an accommodatio...	0	0	0	BKNG.O	neutral
1	2015-04-09 08:45:00	Q5. What activity organised by an accommodatio...	0	0	0	BKNG.O	neutral
2	2015-04-09 08:51:00	Q6 #BookingsBest lists are based on reviews wi...	3	1	5	BKNG.O	positive
3	2015-04-09 08:52:00	#BookingsBest lists are based on reviews with ...	1	1	0	BKNG.O	positive
4	2015-04-09 08:52:00	#BookingsBest lists are based on reviews with ...	0	0	0	BKNG.O	positive

Explore Data – Tweets and Sentiment

Raw data comes in at relatively high-frequency (n minute intervals)

Out[8]:

	date	text	likes	retweet	count_comments	ticker
0	2015-04-09 08:45:00	Q5. What activity organised by an accommodatio...	0	0	0	BKNG.O
1	2015-04-09 08:45:00	Q5. What activity organised by an accommodatio...	0	0	0	BKNG.O
2	2015-04-09 08:51:00	Q6 #BookingsBest lists are based on reviews wi...	3	1	5	BKNG.O
3	2015-04-09 08:52:00	#BookingsBest lists are based on reviews with ...	1	1	0	BKNG.O
4	2015-04-09 08:52:00	#BookingsBest lists are based on reviews with ...	0	0	0	BKNG.O

In [9]:

```
#Process the tweet text for senitment indicator
tweet_df['senti']  = tweet_df['text'].apply(clean_get_twt_sentiment)
```

In [10]:

```
#Explore tweet data for senitment indicator
tweet_df.head()
```

Out[10]:

	date	text	likes	retweet	count_comments	ticker	senti
0	2015-04-09 08:45:00	Q5. What activity organised by an accommodatio...	0	0	0	BKNG.O	neutral
1	2015-04-09 08:45:00	Q5. What activity organised by an accommodatio...	0	0	0	BKNG.O	neutral
2	2015-04-09 08:51:00	Q6 #BookingsBest lists are based on reviews wi...	3	1	5	BKNG.O	positive
3	2015-04-09 08:52:00	#BookingsBest lists are based on reviews with ...	1	1	0	BKNG.O	positive
4	2015-04-09 08:52:00	#BookingsBest lists are based on reviews with ...	0	0	0	BKNG.O	positive

Aggregate Data – Tweets and Sentiment

Aggregate the minute-by-minute data to quarterly buckets

```
In [18]: #Explore the resampling by Quarterly and aggregation.. this should be similar to cumsun function
         tweet_sum_df.resample('Q', label='right').sum()
```

Out[18]:

	likes	retweet	count_comments	senti_negative	senti_neutral	senti_positive
2015-06-30	896	348	152	4	39	48
2015-09-30	263	190	139	0	10	17
2015-12-31	154	57	35	1	9	9
2016-03-31	171	85	15	0	3	3
2016-06-30	585	141	64	3	6	12
2016-09-30	1051	327	278	3	16	27
2016-12-31	556	120	45	2	7	14
2017-03-31	494	130	64	4	7	20
2017-06-30	742	183	313	2	13	47
2017-09-30	1160	250	679	9	32	82
2017-12-31	1492	303	500	8	34	105
2018-03-31	1199	159	222	9	27	62
2018-06-30	897	112	217	5	5	50
2018-09-30	344	68	385	0	2	8
2018-12-31	246	66	260	0	2	8
2019-03-31	302	53	285	2	0	11
2019-06-30	202	28	205	1	1	9

Join Revenues with Tweets and Sentiment Data

Join our revenue (labels) with Tweet count and sentiment data (features)

In [22]:
```
#Join tweet and revenue dataframe by quarterly date
tweet_fund_df = fund_df[['totalRevenue','costOfRevenue']].join(tweet_sum_quarterly_df, how='inner')
tweet_fund_df
```

Out[22]:

	totalRevenue	costOfRevenue	likes	retweet	count_comments	senti_negative	senti_neutral	senti_positive
2019-03-31	2837000000.00	501000000.00	302	53	285	2	0	11
2018-12-31	3212615000.00	242000000.00	246	66	260	0	2	8
2018-09-30	4849090000.00	0.00	344	68	385	0	2	8
2018-06-30	3537094000.00	0.00	897	112	217	5	5	50
2018-03-31	2928201000.00	0.00	1199	159	222	9	27	62
2017-12-31	2803093000.00	45150000.00	1492	303	500	8	34	105
2017-09-30	4434029000.00	54181000.00	1160	250	679	9	32	82
2017-06-30	3024556000.00	67425000.00	742	183	313	2	13	47
2017-03-31	2419404000.00	80401000.00	494	130	64	4	7	20
2016-12-31	2348433000.00	72072000.00	556	120	45	2	7	14
2016-09-30	3690552000.00	101489000.00	1051	327	278	3	16	27
2016-06-30	2555902000.00	126084000.00	585	141	64	3	6	12
2016-03-31	2148119000.00	128669000.00	171	85	15	0	3	3
2015-12-31	1999995000.00	120612000.00	154	57	35	1	9	9
2015-09-30	3102901000.00	169274000.00	263	190	139	0	10	17
2015-06-30	2280397000.00	187491000.00	896	348	152	4	39	48

Normalise All Data

Normalise data with *sklearn.preprocessing.MinMaxScaler* or *sklearn.preprocessing.MinMaxScaler*

The transformation is calculated as:

```
X_scaled = scale * X + min - X.min(axis=0) * scale
where scale = (max - min) / (X.max(axis=0) - X.min(axis=0))
```

The standard score of a sample x is calculated as:

z = (x - u) / s

Out[25]:

	totalRevenue	likes	retweet	count_comments	senti_negative	senti_neutral	senti_positive
2019-03-31	-0.224631	-0.886782	-1.151780	0.320742	-0.415227	-1.081182	-0.743672
2018-12-31	0.261087	-1.025690	-1.014412	0.179290	-1.079591	-0.917985	-0.846543
2018-09-30	2.377258	-0.782600	-0.993278	0.886550	-1.079591	-0.917985	-0.846543
2018-06-30	0.680680	0.589121	-0.528339	-0.064007	0.581318	-0.673189	0.593652
2018-03-31	-0.106696	1.338234	-0.031700	-0.035717	1.910046	1.121982	1.005136
2017-12-31	-0.268477	2.065023	1.489917	1.537230	1.577864	1.693173	2.479620
2017-09-30	1.840531	1.241495	0.929877	2.550026	1.910046	1.529975	1.690942
2017-06-30	0.017903	0.204642	0.221903	0.479169	-0.415227	-0.020400	0.490781
2017-03-31	-0.764636	-0.410524	-0.338137	-0.929693	0.249136	-0.509992	-0.435059
2016-12-31	-0.856410	-0.256733	-0.443805	-1.037197	-0.415227	-0.509992	-0.640801
2016-09-30	0.879121	0.971119	1.743520	0.281136	-0.083045	0.224396	-0.195026
2016-06-30	-0.588126	-0.184798	-0.221903	-0.929693	-0.083045	-0.591590	-0.709381
2016-03-31	-1.115442	-1.211728	-0.813643	-1.206939	-1.079591	-0.836386	-1.017995
2015-12-31	-1.306985	-1.253897	-1.109513	-1.093778	-0.747409	-0.346794	-0.812252
2015-09-30	0.119213	-0.983522	0.295870	-0.505337	-1.079591	-0.265196	-0.537930
2015-06-30	-0.944389	0.586640	1.965422	-0.431782	0.249136	2.101166	0.525071

Explore Correlations

Revenues are positively correlated with comment counts (75%), likes (25%) and positive sentiment (21%)

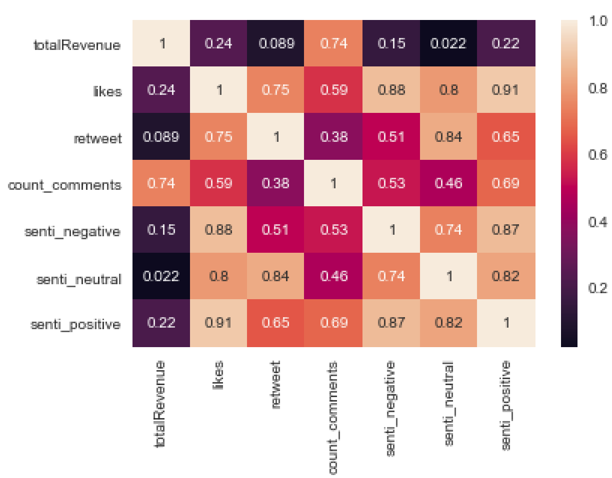
Out[26]:

	totalRevenue	likes	retweet	count_comments	senti_negative	senti_neutral	senti_positive
totalRevenue	1.000000	0.242425	0.089407	0.740516	0.154957	0.021636	0.215627
likes	0.242425	1.000000	0.749258	0.585070	0.884749	0.795292	0.911202
retweet	0.089407	0.749258	1.000000	0.377470	0.510280	0.836692	0.652346
count_comments	0.740516	0.585070	0.377470	1.000000	0.532460	0.462752	0.686259
senti_negative	0.154957	0.884749	0.510280	0.532460	1.000000	0.742016	0.874409
senti_neutral	0.021636	0.795292	0.836692	0.462752	0.742016	1.000000	0.822844
senti_positive	0.215627	0.911202	0.652346	0.686259	0.874409	0.822844	1.000000

In [27]:

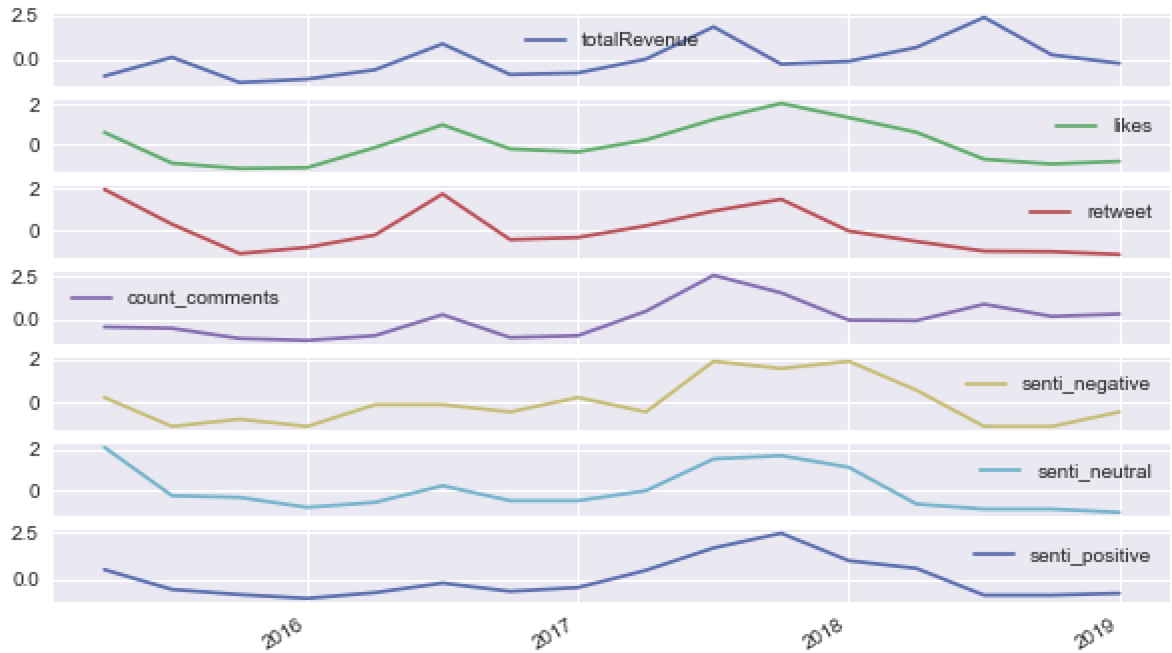
```
#Explore visual correlation between Revenue and other feature values
sns.heatmap(tweet_fund_df_nom.corr(), annot=True)
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x17b91f85160>



Explore Correlations – Time Series

Exploring correlations over time, the correlations appear to be tradeable



Model Definition - OLS

Model definition – pick targets (labels, or dependent variables) and features (independent variables)

```
In [30]: # Select target(dependent/label) and feature(independent) variables
         targets = tweet_fund_df_nom['totalRevenue']
         features = tweet_fund_df_nom.drop(['totalRevenue'], axis=1)
         feature_names = features.columns
         feature_names = features.columns
         features.shape, targets.shape

Out[30]: ((16, 6), (16,))

In [31]: # Using a linear OLS model here
         import statsmodels.api as sm

In [32]: # Add a constant to the features
         linear_features = sm.add_constant(features)

In [33]: #Explore features columns
         linear_features.columns

Out[33]: Index(['const', 'likes', 'retweet', 'count_comments', 'senti_negative',
                'senti_neutral', 'senti_positive'],
               dtype='object')

In [34]: # Create a size for the training set that is 85% of the total number of samples
         train_size = int(0.85 * targets.shape[0])
         train_features = linear_features[:train_size]
         train_targets = targets[:train_size]
         test_features = linear_features[train_size:]
         test_targets = targets[train_size:]
         print(linear_features.shape, train_features.shape, test_features.shape)
```

Model Run - OLS

Ordinary-least squares (OLS) has high R-squared on low p-values for tweet counts

In [34]:
```
# Create a size for the training set that is 85% of the total number of samples
train_size = int(0.85 * targets.shape[0])
train_features = linear_features[:train_size]
train_targets = targets[:train_size]
test_features = linear_features[train_size:]
test_targets = targets[train_size:]
print(linear_features.shape, train_features.shape, test_features.shape)
```

```
(16, 7) (13, 7) (3, 7)
```

In [35]:
```
# Create the linear model and complete the least squares fit
model = sm.OLS(train_targets, train_features)
results = model.fit()  # fit the model
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           totalRevenue   R-squared:                       0.825
Model:                            OLS   Adj. R-squared:                  0.649
Method:                 Least Squares   F-statistic:                     4.701
Date:                Mon, 17 Jun 2019   Prob (F-statistic):             0.0408
Time:                        00:40:04   Log-Likelihood:                -7.1495
No. Observations:                  13   AIC:                             28.30
Df Residuals:                       6   BIC:                             32.25
Df Model:                           6
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const             -0.2129      0.217     -0.979      0.365      -0.745      0.319
likes              1.4794      0.794      1.862      0.112      -0.464      3.423
retweet           -0.4985      0.531     -0.938      0.384      -1.798      0.801
count_comments     1.2680      0.250      5.069      0.002       0.656      1.880
senti_negative    -0.0727      0.511     -0.142      0.892      -1.324      1.178
```

Feature Evaluation – Random Forest

Random forest model indicates again tweet counts most important

```
In [34]: # Create a size for the training set that is 85% of the total number of samples
         train_size = int(0.85 * targets.shape[0])
         train_features = linear_features[:train_size]
         train_targets = targets[:train_size]
         test_features = linear_features[train_size:]
         test_targets = targets[train_size:]
         print(linear_features.shape, train_features.shape, test_features.shape)

         (16, 7) (13, 7) (3, 7)

In [35]: # Create the linear model and complete the least squares fit
         model = sm.OLS(train_targets, train_features)
         results = model.fit()  # fit the model
         print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           totalRevenue   R-squared:                       0.825
Model:                            OLS   Adj. R-squared:                  0.649
Method:                 Least Squares   F-statistic:                     4.701
Date:                Mon, 17 Jun 2019   Prob (F-statistic):             0.0408
Time:                        00:40:04   Log-Likelihood:                -7.1495
No. Observations:                  13   AIC:                             28.30
Df Residuals:                       6   BIC:                             32.25
Df Model:                           6
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const             -0.2129      0.217     -0.979      0.365      -0.745       0.319
likes              1.4794      0.794      1.862      0.112      -0.464       3.423
retweet           -0.4985      0.531     -0.938      0.384      -1.798       0.801
count_comments     1.2680      0.250      5.069      0.002       0.656       1.880
senti_negative    -0.0727      0.511     -0.142      0.892      -1.324       1.178
```

Feature Evaluation – Random Forest

Random forest model indicates again tweet count feature most important

In [39]:
```
# Try with Random Forest
from sklearn.ensemble import RandomForestRegressor

# Create the random forest model and fit to the training data
rfr = RandomForestRegressor(n_estimators=200)
rfr.fit(train_features, train_targets)

# Look at the R^2 scores on train and test
print(rfr.score(train_features, train_targets))
print(rfr.score(test_features, test_targets))
```

```
0.8223484970574497
-1.615363389914215
```

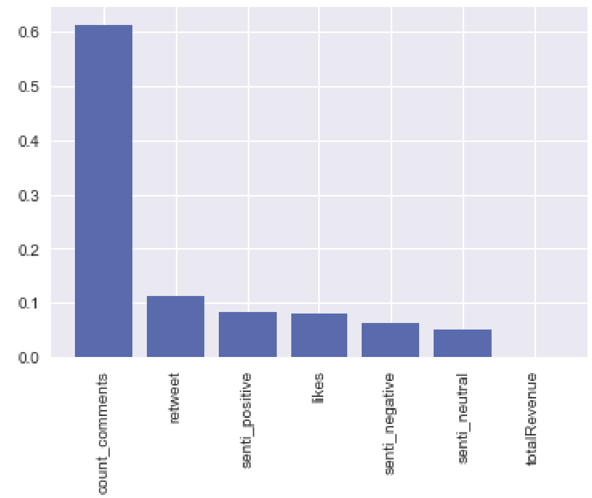
In [40]:
```
# Get feature importances from our random forest model
importances = rfr.feature_importances_

#importances = importances[1:]

# Get the index of importances from greatest importance to least
sorted_index = np.argsort(importances)[::-1]
x = range(len(importances))

# Create tick labels
labels = np.array(tweet_fund_df_nom.columns.values)[sorted_index]
plt.bar(x, importances[sorted_index], tick_label=labels)

# Rotate tick labels to vertical
plt.xticks(rotation=90)
plt.show()
```



DataScrum

Upcoming events:

- **Implications of Alternative Data for Cryptocurrencies** on 15th Oct.
  - Apply Discount Code **CRYPTODATA** for 10% off on tickets
- **Alternative Data Bootcamp:** Workshop in November (Date TBD)
- https://www.meetup.com/Datascrum/
- Website: http://www.datascrum.co.uk
- Medium.com/@datascrum
- Email: team@datascrum.co.uk
- Twitter: @dataScrum

Resources

- Hackathon on September 14th at Microsoft Reactor London
- Quandl (now Nasdaq) – www.quandl.com
- Open FactSet - open.factset.com
- AlternativeData.org – www.alternartivedata.org
- BattleFin - BattleFin.com
- NeuData Alt Data London 2019 Summit – NeuData.co