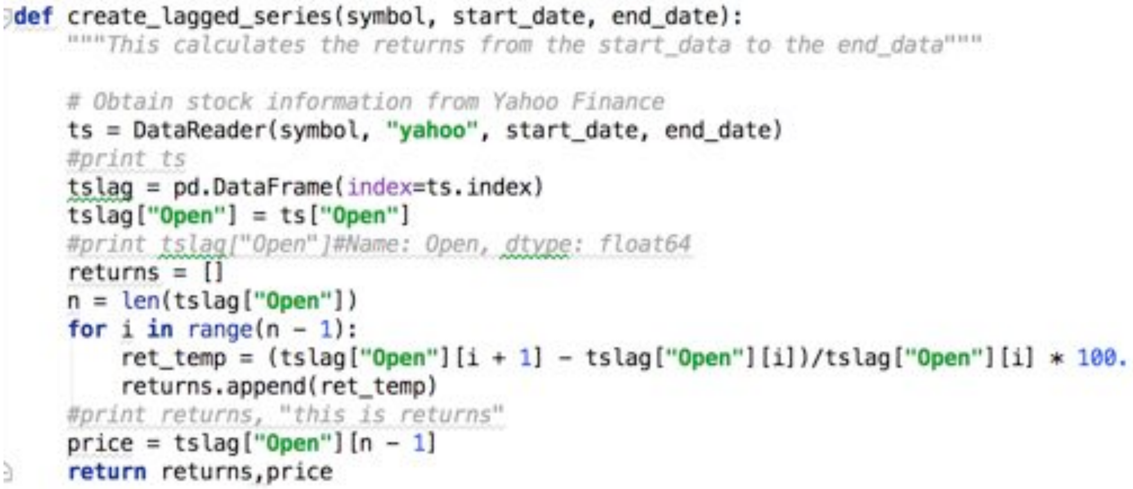
EECS E6895 Advanced Big Data Analytics

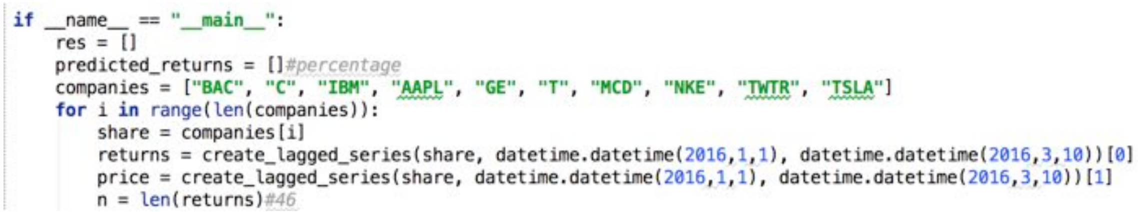
Homework 2
Name: Jingyi Yuan
UNI: jy2736

Part 1:Regression using returns

First, I calculate the returns of each day from Jan 1st 2016 to March 10th 2016.

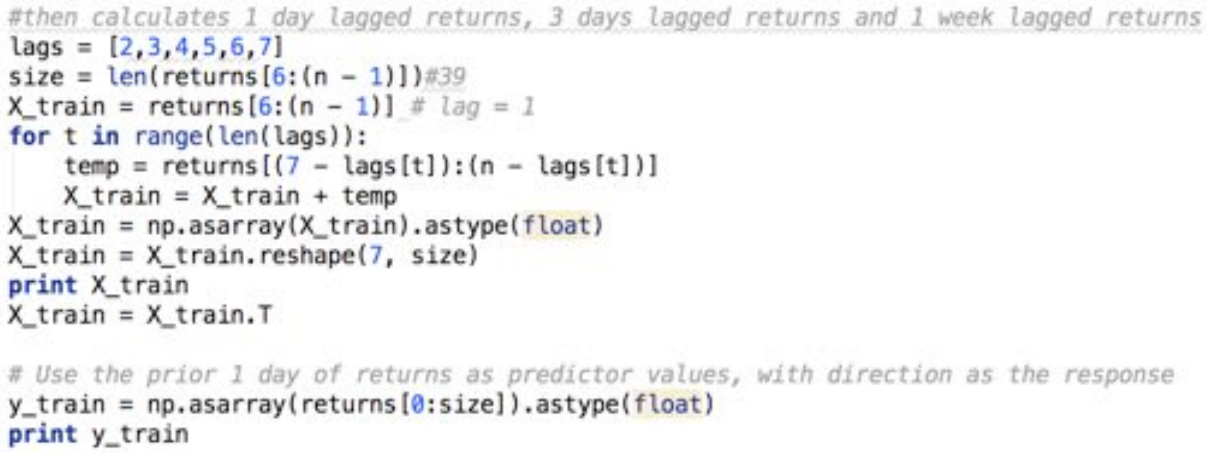


```
def create_lagged_series(symbol, start_date, end_date):
    """This calculates the returns from the start_data to the end_data"""

    # Obtain stock information from Yahoo Finance
    ts = DataReader(symbol, "yahoo", start_date, end_date)
    #print ts
    tslag = pd.DataFrame(index=ts.index)
    tslag["Open"] = ts["Open"]
    #print tslag["Open"]#Name: Open, dtype: float64
    returns = []
    n = len(tslag["Open"])
    for i in range(n - 1):
        ret_temp = (tslag["Open"][i + 1] - tslag["Open"][i])/tslag["Open"][i] * 100.
        returns.append(ret_temp)
    #print returns, "this is returns"
    price = tslag["Open"][n - 1]
    return returns,price
```



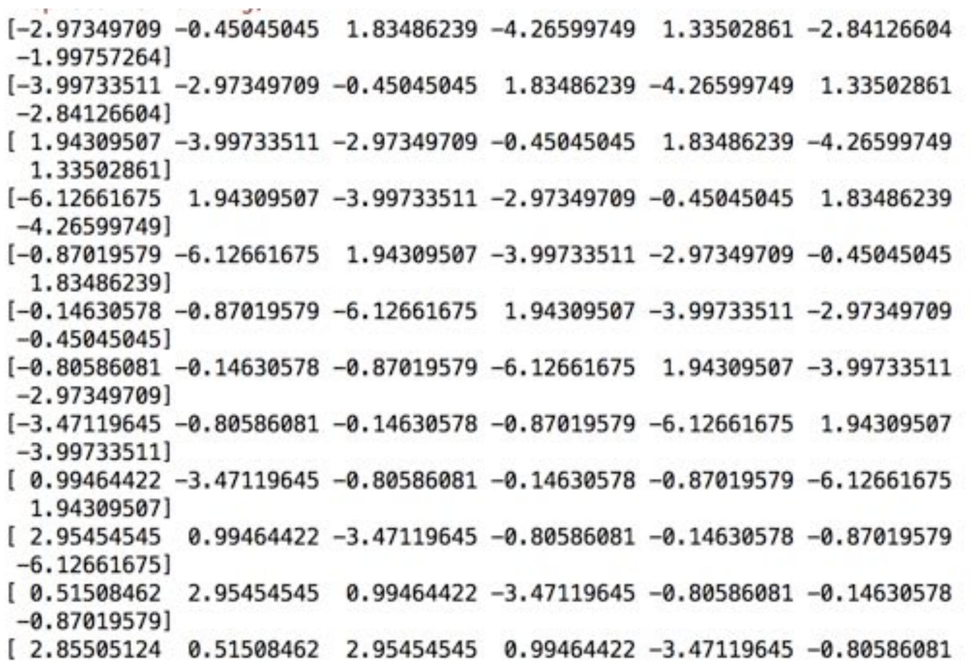
```
if __name__ == "__main__":
    res = []
    predicted_returns = []#percentage
    companies = ["BAC", "C", "IBM", "AAPL", "GE", "T", "MCD", "NKE", "TWTR", "TSLA"]
    for i in range(len(companies)):
        share = companies[i]
        returns = create_lagged_series(share, datetime.datetime(2016,1,1), datetime.datetime(2016,3,10))[0]
        price = create_lagged_series(share, datetime.datetime(2016,1,1), datetime.datetime(2016,3,10))[1]
        n = len(returns)#46
```

Then I use the returns to calculate the 1day to 7 days lagged returns as X_train and use the returns as the y_train:



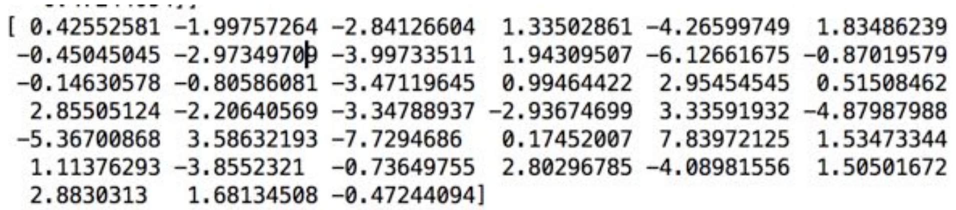
```
#then calculates 1 day lagged returns, 3 days lagged returns and 1 week lagged returns
lags = [2,3,4,5,6,7]
size = len(returns[6:(n - 1)])#39
X_train = returns[6:(n - 1)] # lag = 1
for t in range(len(lags)):
    temp = returns[(7 - lags[t]):(n - lags[t])]
    X_train = X_train + temp
X_train = np.asarray(X_train).astype(float)
X_train = X_train.reshape(7, size)
print X_train
X_train = X_train.T

# Use the prior 1 day of returns as predictor values, with direction as the response
y_train = np.asarray(returns[0:size]).astype(float)
print y_train
```

X_train is like this:



```
[-2.97349709 -0.45045045  1.83486239 -4.26599749  1.33502861 -2.84126604
 -1.99757264]
[-3.99733511 -2.97349709 -0.45045045  1.83486239 -4.26599749  1.33502861
 -2.84126604]
[ 1.94309507 -3.99733511 -2.97349709 -0.45045045  1.83486239 -4.26599749
  1.33502861]
[-6.12661675  1.94309507 -3.99733511 -2.97349709 -0.45045045  1.83486239
 -4.26599749]
[-0.87019579 -6.12661675  1.94309507 -3.99733511 -2.97349709 -0.45045045
  1.83486239]
[-0.14630578 -0.87019579 -6.12661675  1.94309507 -3.99733511 -2.97349709
 -0.45045045]
[-0.80586081 -0.14630578 -0.87019579 -6.12661675  1.94309507 -3.99733511
 -2.97349709]
[-3.47119645 -0.80586081 -0.14630578 -0.87019579 -6.12661675  1.94309507
 -3.99733511]
[ 0.99464422 -3.47119645 -0.80586081 -0.14630578 -0.87019579 -6.12661675
  1.94309507]
[ 2.95454545  0.99464422 -3.47119645 -0.80586081 -0.14630578 -0.87019579
 -6.12661675]
[ 0.51508462  2.95454545  0.99464422 -3.47119645 -0.80586081 -0.14630578
 -0.87019579]
[ 2.85505124  0.51508462  2.95454545  0.99464422 -3.47119645 -0.80586081
```

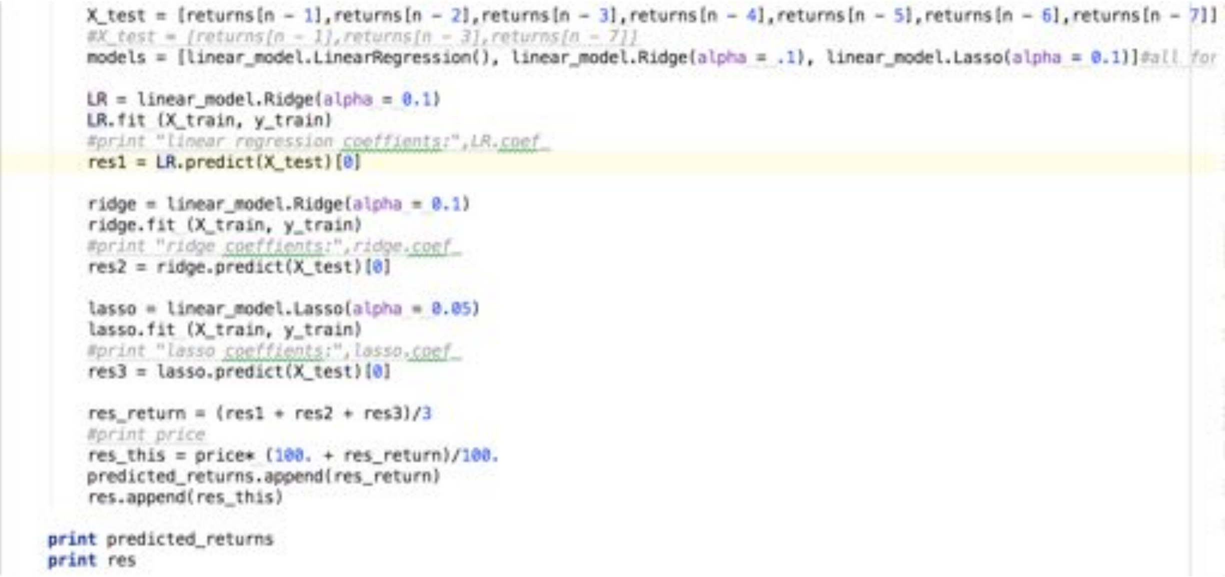
y_train is:



```
[ 0.42552581 -1.99757264 -2.84126604  1.33502861 -4.26599749  1.83486239
 -0.45045045 -2.97349709 -3.99733511  1.94309507 -6.12661675 -0.87019579
 -0.14630578 -0.80586081 -3.47119645  0.99464422  2.95454545  0.51508462
  2.85505124 -2.20640569 -3.34788937 -2.93674699  3.33591932 -4.87987988
 -5.36700868  3.58632193 -7.7294686   0.17452007  7.83972125  1.53473344
  1.11376293 -3.8552321  -0.73649755  2.80296785 -4.08981556  1.50501672
  2.8830313   1.68134508 -0.47244094]
```

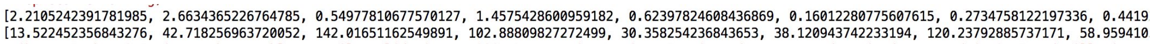
which is the value of everyday returns.

Then we I 1 to 7 days lagged returns from now as the test data:



```
        X_test = [returns[n - 1],returns[n - 2],returns[n - 3],returns[n - 4],returns[n - 5],returns[n - 6],returns[n - 7]]
        #X_test = [returns[n - 1],returns[n - 3],returns[n - 7]]
        models = [linear_model.LinearRegression(), linear_model.Ridge(alpha = .1), linear_model.Lasso(alpha = 0.1)]#all for

        LR = linear_model.Ridge(alpha = 0.1)
        LR.fit (X_train, y_train)
        #print "linear regression coeffients:",LR.coef_
        res1 = LR.predict(X_test)[0]

        ridge = linear_model.Ridge(alpha = 0.1)
        ridge.fit (X_train, y_train)
        #print "ridge coeffients:",ridge.coef_
        res2 = ridge.predict(X_test)[0]

        lasso = linear_model.Lasso(alpha = 0.05)
        lasso.fit (X_train, y_train)
        #print "lasso coeffients:",lasso.coef_
        res3 = lasso.predict(X_test)[0]

        res_return = (res1 + res2 + res3)/3
        #print price
        res_this = price* (100. + res_return)/100.
        predicted_returns.append(res_return)
        res.append(res_this)

    print predicted_returns
    print res
```

And fit 3 models: linear regression, ridge regression with alpha = 0.1, and lasso regression with alpha = 0.05, and I use the average of these 3 predictions as the predicted returns, Finally, I use the price to calculate the predicted value:



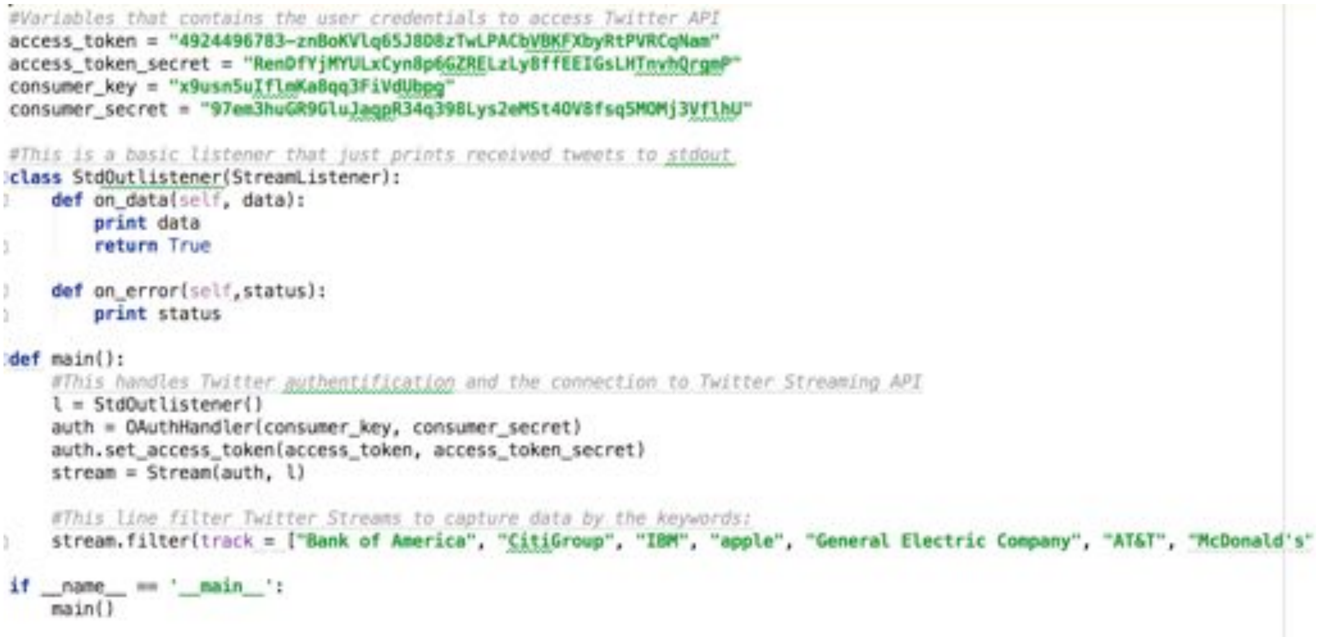
```
[2.2105242391781985, 2.6634365226764785, 0.54977810677570127, 1.4575428600959182, 0.62397824608436869, 0.16012280775607615, 0.2734758122197336, 0.4419
[13.522452356843276, 42.718256963720052, 142.01651162549891, 102.88809827272499, 30.358254236843653, 38.120943742233194, 120.23792885737171, 58.959410
```

companies = ["BAC", "C", "IBM", "AAPL", "GE", "T", "MCD", "NKE", "TWTR", "TSLA"]
predicted returns:
[2.2105242391781985, 2.6634365226764785, 0.54977810677570127, 1.4575428600959182, 0.62397824608436869, 0.16012280775607615, 0.2734758122197336, 0.44192390757883021, -1.6770297658623738, -2.7045978997942739]
predicted values for 10 companies:
[13.522452356843276, 42.718256963720052, 142.01651162549891, 102.88809827272499, 30.358254236843653, 38.120943742233194, 120.23792885737171, 58.959410338168006, 17.393334417648649, 204.320344410432]

References:
1. https://www.quantstart.com/articles/Forecasting-Financial-Time-Series-Part-1
2. Session 5: Financial Analytics, IEOR4574 Business Analytics for OR (Spring 2016)

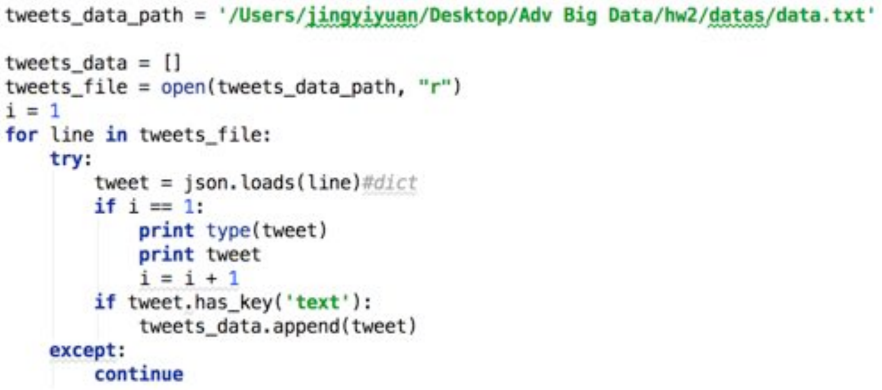
Part 2: Downloading data from Twitter

One day before the result comes out (Mar 10th), I collected data for 30 minutes about these ten companies and saved all data to file "data.txt"

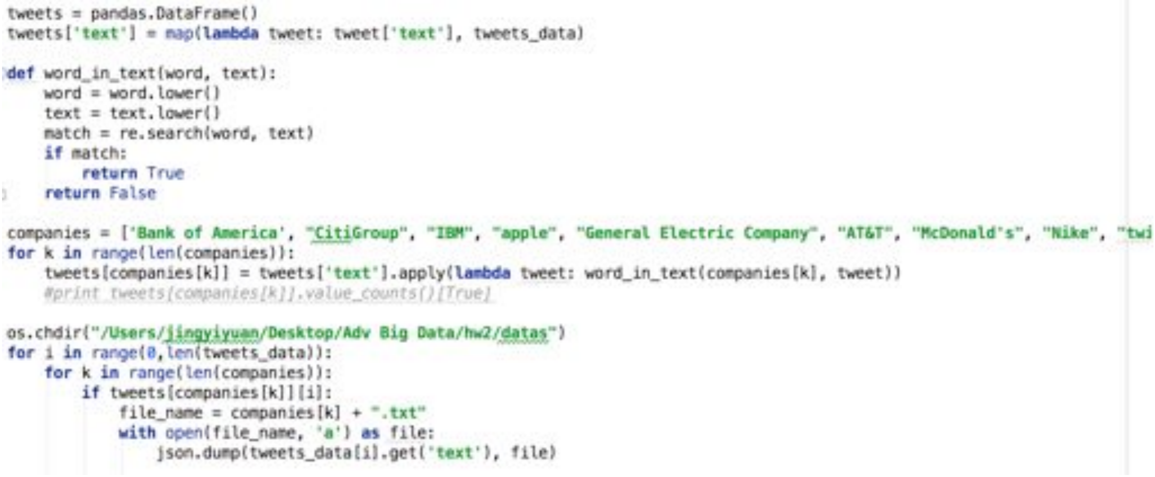


```
#Variables that contains the user credentials to access Twitter API
access_token = "4924496783-znBoKVlq65J8D8zTwLPACbVBKFXbyRtPVRCqNam"
access_token_secret = "RenDfYjMYULxCyn8p6GZRELzLyBffEEIGsLHTnvhQrgmP"
consumer_key = "x9usn5uIflmKa8qq3FiVdUbpg"
consumer_secret = "97em3huGR9GluJaqpR34q39BLys2eMSt4OV8fsq5MOMj3VflhU"

#This is a basic listener that just prints received tweets to stdout
class StdOutlistener(StreamListener):
    def on_data(self, data):
        print data
        return True

    def on_error(self,status):
        print status

def main():
    #This handles Twitter authentification and the connection to Twitter Streaming API
    l = StdOutlistener()
    auth = OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_token, access_token_secret)
    stream = Stream(auth, l)

    #This line filter Twitter Streams to capture data by the keywords:
    stream.filter(track = ["Bank of America", "CitiGroup", "IBM", "apple", "General Electric Company", "AT&T", "McDonald's"

if __name__ == '__main__':
    main()
```

Then created 10 files for these companies:



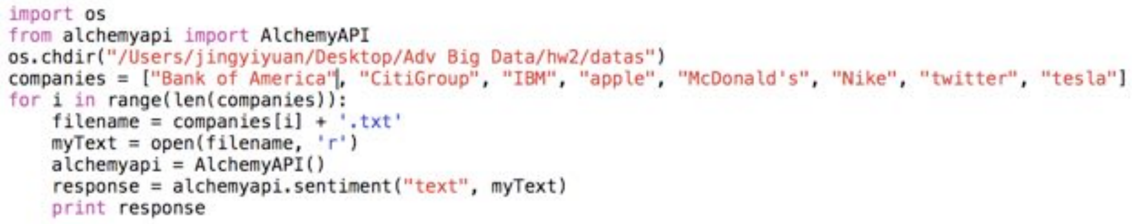
```
tweets_data_path = '/Users/jingyiyuan/Desktop/Adv Big Data/hw2/datas/data.txt'

tweets_data = []
tweets_file = open(tweets_data_path, "r")
i = 1
for line in tweets_file:
    try:
        tweet = json.loads(line)#dict
        if i == 1:
            print type(tweet)
            print tweet
            i = i + 1
        if tweet.has_key('text'):
            tweets_data.append(tweet)
    except:
        continue
```



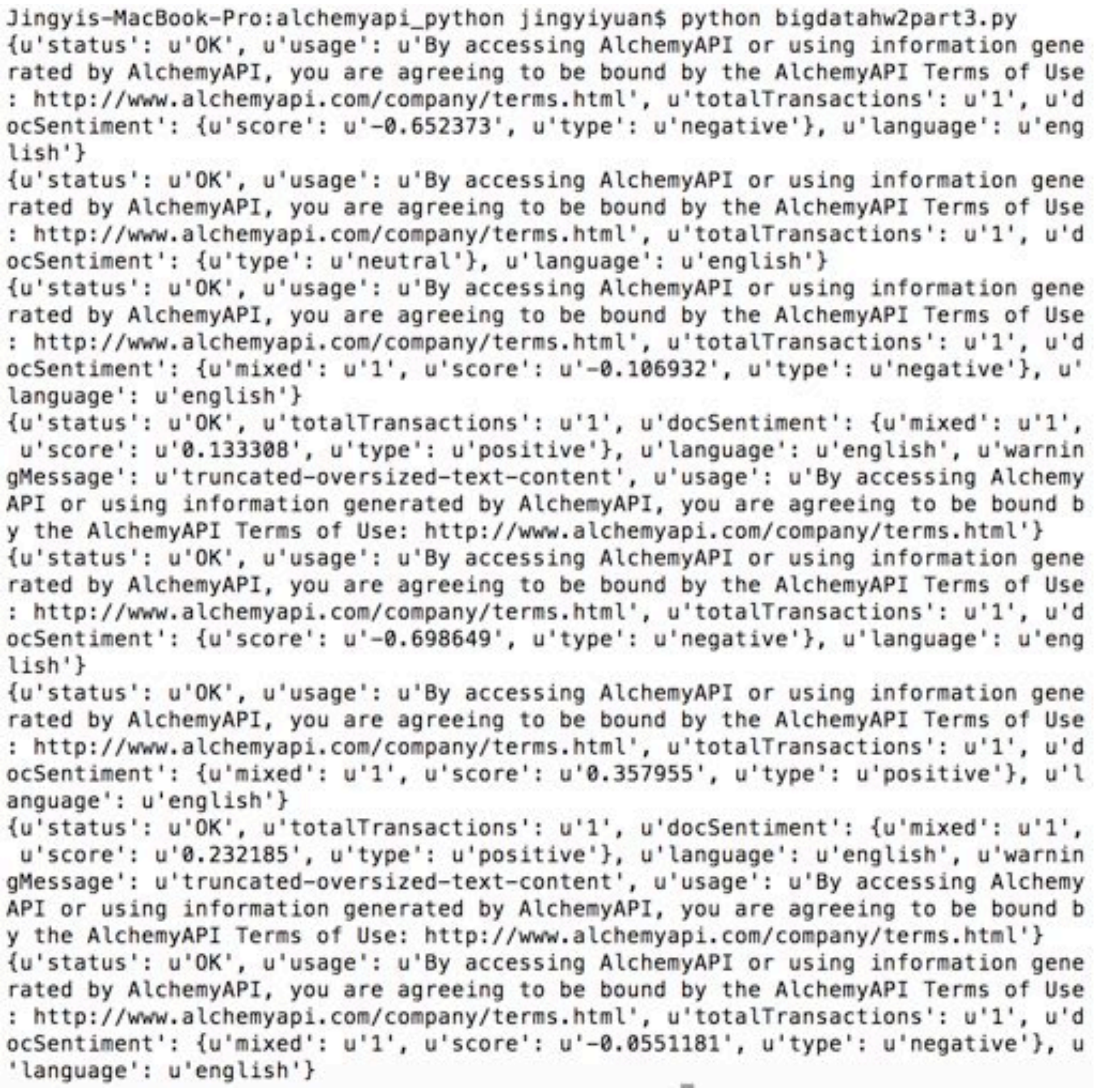
```
tweets = pandas.DataFrame()
tweets['text'] = map(lambda tweet: tweet['text'], tweets_data)

def word_in_text(word, text):
    word = word.lower()
    text = text.lower()
    match = re.search(word, text)
    if match:
        return True
    return False

companies = ['Bank of America', "CitiGroup", "IBM", "apple", "General Electric Company", "AT&T", "McDonald's", "Nike", "twi
for k in range(len(companies)):
    tweets[companies[k]] = tweets['text'].apply(lambda tweet: word_in_text(companies[k], tweet))
    #print tweets[companies[k]].value_counts()[True]

os.chdir("/Users/jingyiyuan/Desktop/Adv Big Data/hw2/datas")
for i in range(0,len(tweets_data)):
    for k in range(len(companies)):
        if tweets[companies[k]][i]:
            file_name = companies[k] + ".txt"
            with open(file_name, 'a') as file:
                json.dump(tweets_data[i].get('text'), file)
```

Part 3: Analyze the data and come up with the results

Analyze whether these data is positive, negative of neural using alchemy api:



```
import os
from alchemyapi import AlchemyAPI
os.chdir("/Users/jingyiyuan/Desktop/Adv Big Data/hw2/datas")
companies = ["Bank of America", "CitiGroup", "IBM", "apple", "McDonald's", "Nike", "twitter", "tesla"]
for i in range(len(companies)):
    filename = companies[i] + '.txt'
    myText = open(filename, 'r')
    alchemyapi = AlchemyAPI()
    response = alchemyapi.sentiment("text", myText)
    print response
```



```
Jingyis-MacBook-Pro:alchemyapi_python jingyiyuan$ python bigdatahw2part3.py
{u'status': u'OK', u'usage': u'By accessing AlchemyAPI or using information gene
rated by AlchemyAPI, you are agreeing to be bound by the AlchemyAPI Terms of Use
: http://www.alchemyapi.com/company/terms.html', u'totalTransactions': u'1', u'd
ocSentiment': {u'score': u'-0.652373', u'type': u'negative'}, u'language': u'eng
lish'}
{u'status': u'OK', u'usage': u'By accessing AlchemyAPI or using information gene
rated by AlchemyAPI, you are agreeing to be bound by the AlchemyAPI Terms of Use
: http://www.alchemyapi.com/company/terms.html', u'totalTransactions': u'1', u'd
ocSentiment': {u'type': u'neutral'}, u'language': u'english'}
{u'status': u'OK', u'usage': u'By accessing AlchemyAPI or using information gene
rated by AlchemyAPI, you are agreeing to be bound by the AlchemyAPI Terms of Use
: http://www.alchemyapi.com/company/terms.html', u'totalTransactions': u'1', u'd
ocSentiment': {u'mixed': u'1', u'score': u'-0.106932', u'type': u'negative'}, u'
language': u'english'}
{u'status': u'OK', u'totalTransactions': u'1', u'docSentiment': {u'mixed': u'1',
 u'score': u'0.133308', u'type': u'positive'}, u'language': u'english', u'warnin
gMessage': u'truncated-oversized-text-content', u'usage': u'By accessing Alchemy
API or using information generated by AlchemyAPI, you are agreeing to be bound b
y the AlchemyAPI Terms of Use: http://www.alchemyapi.com/company/terms.html'}
{u'status': u'OK', u'usage': u'By accessing AlchemyAPI or using information gene
rated by AlchemyAPI, you are agreeing to be bound by the AlchemyAPI Terms of Use
: http://www.alchemyapi.com/company/terms.html', u'totalTransactions': u'1', u'd
ocSentiment': {u'score': u'-0.698649', u'type': u'negative'}, u'language': u'eng
lish'}
{u'status': u'OK', u'usage': u'By accessing AlchemyAPI or using information gene
rated by AlchemyAPI, you are agreeing to be bound by the AlchemyAPI Terms of Use
: http://www.alchemyapi.com/company/terms.html', u'totalTransactions': u'1', u'd
ocSentiment': {u'mixed': u'1', u'score': u'0.357955', u'type': u'positive'}, u'l
anguage': u'english'}
{u'status': u'OK', u'totalTransactions': u'1', u'docSentiment': {u'mixed': u'1',
 u'score': u'0.232185', u'type': u'positive'}, u'language': u'english', u'warnin
gMessage': u'truncated-oversized-text-content', u'usage': u'By accessing Alchemy
API or using information generated by AlchemyAPI, you are agreeing to be bound b
y the AlchemyAPI Terms of Use: http://www.alchemyapi.com/company/terms.html'}
{u'status': u'OK', u'usage': u'By accessing AlchemyAPI or using information gene
rated by AlchemyAPI, you are agreeing to be bound by the AlchemyAPI Terms of Use
: http://www.alchemyapi.com/company/terms.html', u'totalTransactions': u'1', u'd
ocSentiment': {u'mixed': u'1', u'score': u'-0.0551181', u'type': u'negative'}, u
'language': u'english'}
```

From the picture, we can see that:
companies = ["Bank of America", "CitiGroup", "IBM", "apple", "McDonald's", "Nike", "twitter", "tesla"]
responses = [negative, neural, negative, positive, negative, positive, positive, negative]
These is no information form twitter about GT and T.

Then I try to combine these two results:

BAC	C	IBM	AAPL	GE	T	MCD	NKE	TWTR	TSLA
2.211	2.663	0.550	1.457	0.624	0.160	0.270	0.442	-1.677	-5.704
-1	0	-1	1	none	none	-1	1	1	-1

For those whose response is neural, none and which is same direction as its return, I keep the value of its return. Otherwise, we increase or decrease the return by 30% according to the response.

Finally, I have my predicted values as:

BAC	13.43476071
C	42.71825696
IBM	141.7827791
AAPL	102.8880983
GE	30.35825424
T	38.12094374
MCD	120.1366339
NKE	58.95941034
TWTR	17,48233709
TSLA	204.3203444