getranges

April 12, 2021

0.0.1 NOTE! HASHTAGGED CELLS THAT HAVE FORMULAS SHOULD HAVE THE # REMOVED BEFORE RUNNING *security reasons*

[4]:
```
# -*- coding: utf-8 -*-



"""
Created on Tue Apr  6 17:35:01 2021

@author: schne
"""
import re
import numpy as np
import pandas as pd
import pickle
from bs4 import BeautifulSoup
from __future__ import print_function
import os.path
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
import email
import base64
import requests
import itertools

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

# If modifying these scopes, delete the file token.json.
SCOPES = ['https://www.googleapis.com/auth/gmail.readonly']


def search_inbox(service, user_id,search_string):
    """
    Search the inbox for emails using standard gmail search parameters
    and return a list of email IDs for each result
    PARAMS:
        service: the google api service object already instantiated
        user_id: user id for google api service ('me' works here if
        already authenticated)
        search_string: search operators you can use with Gmail
        (see https://support.google.com/mail/answer/7190?hl=en for a list)
    RETURNS:
        List containing email IDs of search query
    """
    try:
        # initiate the list for returning
        list_ids = []

        # get the id of all messages that are in the search string
        search_ids = service.users().messages().
 ↪list(userId=user_id,labelIds='INBOX', q=search_string).execute()

        # if there were no results, print warning and return empty string
        try:
            ids = search_ids['messages']

        except KeyError:
            print("WARNING: the search queried returned 0 results")
            print("returning an empty string")
            return ""

        if len(ids)>1:
            for msg_id in ids:
                list_ids.append(msg_id['id'])
            return(list_ids)

        else:
            list_ids.append(ids['id'])
            return list_ids

    except (errors.HttpError, error):
        print("An error occured: %s") % error


def get_message(service, user_id, msg_id):
    """
    Search the inbox for specific message by ID and return it back as a
    clean string. String may contain Python escape characters for newline
    and return line.
    PARAMS
        service: the google api service object already instantiated
        user_id: user id for google api service ('me' works here if
        already authenticated)
        msg_id: the unique id of the email you need
    RETURNS
        A string of encoded text containing the message body
    """
    try:
        # grab the message instance
        message = service.users().messages().get(userId=user_id,␣
 ↪id=msg_id,format='raw').execute()

        # decode the raw string, ASCII works pretty well here
        msg_str = base64.urlsafe_b64decode(message['raw'].encode('ASCII'))

        # grab the string from the byte object
        mime_msg = email.message_from_bytes(msg_str)

        # check if the content is multipart (it usually is)
        content_type = mime_msg.get_content_maintype()
        if content_type == 'multipart':
            # there will usually be 2 parts the first will be the body in text
            # the second will be the text in html
            parts = mime_msg.get_payload()

            # return the encoded text
            final_content = parts[0].get_payload()
            return final_content

        elif content_type == 'text':
            return mime_msg.get_payload()

        else:
            return ""
            print("\nMessage is not text or multipart, returned an empty string")
    # unsure why the usual exception doesn't work in this case, but
    # having a standard Exception seems to do the trick
    except Exception:
        print("An error occured: %s") % error


def get_service():
    """
    Authenticate the google api client and return the service object
    to make further calls
    PARAMS
```

```
        None
    RETURNS
        service api object from gmail for making calls
    """

    creds = None
    # The file token.json stores the user's access and refresh tokens, and is
    # created automatically when the authorization flow completes for the first
    # time.
    if os.path.exists('token.json'):
        creds = Credentials.from_authorized_user_file('token.json', SCOPES)
    # If there are no (valid) credentials available, let the user log in.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)
        # Save the credentials for the next run
        with open('token.json', 'w') as token:
            token.write(creds.to_json())

    service = build('gmail', 'v1', credentials=creds)

    # Call the Gmail API
    results = service.users().labels().list(userId='me').execute()
    labels = results.get('labels', [])

    return service
```

0.0.2 Method to Scrape Risk Range Table from the Hedgeye website (currently unable to use as no HTTP access :(

r = requests.get("https://app.hedgeye.com/users/sign_in", auth=('','"))

r

r.headers['content-type']

r.encoding

r.text

s = requests.get(' ')

s.text

soup3=BeautifulSoup(r,'html.parser')

```
[5]: ### Check labels on GMAIL. This is to ensure we only select the correct Label␣
      ↪folder ('INBOX') for the message pull
     #service.users().labels().list(userId='me', x__xgafv=None).execute()
```

```
[6]: ### Check labels on GMAIL. This is to ensure we only select the correct Label␣
      ↪folder ('INBOX') for the message pull
     #service.users().messages().list(userId='me',labelIds='INBOX', x__xgafv=None).
      ↪execute()
```

```
[7]: service=get_service()
     rr_emails=search_inbox(service,'me',"CLICK HERE to submit up to 4 tickers you'd␣
      ↪like to see on the list.")
     rr_emails
```

```
[7]: ['178c5d1ab177f6ab',
      '178b675562417123',
      '178b166d5d3bed46',
      '178ac35ef5b46790',
      '178a7188cb49c5ae',
      '178a1f2520c7d299',
      '1788d4e06e8efd8f',
      '178882dc6ba33b38',
      '17882fe62d70cb4a',
      '1787dcf1ca93c3b2',
      '1786e624bc818228',
      '1786946ed378be54',
      '1786417c71c9df57',
      '1785ef15e49e0838',
      '17859c8091567e0f',
      '1784a47b8ac0fab4',
      '178453d33b1972e9',
      '17840163c9aaa5dc',
      '1783af6421a6d9cb',
      '17835bf5fd97d5b3',
      '17826a85ca0f473a',
      '178215f1ee928c6e',
      '17811e4c42f60a44',
      '177fd49f83332420',
      '177f827fd8a707e4',
      '177f31193a68d85d',
      '177ede2898bb1f8a',
      '177de7542c8d39ee']
```

```
[8]: range_rr_emails=list(iter(range(len(rr_emails))))


     email_series=pd.Series(rr_emails)
```

```
[9]: # Creating a dictionary of email lists
     # using list comprehension

     rr_dict = dict((val, None) for val in rr_emails)

     print(rr_dict)
     rr_dict.keys()
```

```
{'178c5d1ab177f6ab': None, '178b675562417123': None, '178b166d5d3bed46': None,
'178ac35ef5b46790': None, '178a7188cb49c5ae': None, '178a1f2520c7d299': None,
'1788d4e06e8efd8f': None, '178882dc6ba33b38': None, '17882fe62d70cb4a': None,
'1787dcf1ca93c3b2': None, '1786e624bc818228': None, '1786946ed378be54': None,
'1786417c71c9df57': None, '1785ef15e49e0838': None, '17859c8091567e0f': None,
'1784a47b8ac0fab4': None, '178453d33b1972e9': None, '17840163c9aaa5dc': None,
'1783af6421a6d9cb': None, '17835bf5fd97d5b3': None, '17826a85ca0f473a': None,
'178215f1ee928c6e': None, '17811e4c42f60a44': None, '177fd49f83332420': None,
'177f827fd8a707e4': None, '177f31193a68d85d': None, '177ede2898bb1f8a': None,
'177de7542c8d39ee': None}
```

```
[9]: dict_keys(['178c5d1ab177f6ab', '178b675562417123', '178b166d5d3bed46',
     '178ac35ef5b46790', '178a7188cb49c5ae', '178a1f2520c7d299', '1788d4e06e8efd8f',
     '178882dc6ba33b38', '17882fe62d70cb4a', '1787dcf1ca93c3b2', '1786e624bc818228',
     '1786946ed378be54', '1786417c71c9df57', '1785ef15e49e0838', '17859c8091567e0f',
     '1784a47b8ac0fab4', '178453d33b1972e9', '17840163c9aaa5dc', '1783af6421a6d9cb',
     '17835bf5fd97d5b3', '17826a85ca0f473a', '178215f1ee928c6e', '17811e4c42f60a44',
     '177fd49f83332420', '177f827fd8a707e4', '177f31193a68d85d', '177ede2898bb1f8a',
     '177de7542c8d39ee'])
```

```
[10]: ### DO NOT RUN THIS CODE AFTER FIRST PARSING!!! WILL RERUN WHOLE API GET AND␣
       ↪POTENTIALLY INCUR UNWANTED CALL FEES!

      #with open('data_pick.pkl','wb') as pickle_file:
          #pickle.dump(rr_dict, pickle_file)
```

```
[11]: # Hey Morty, I turned myself into a pickle! im_pkl_dict!!
      with open('data_pick.pkl','rb') as pickle_file:
          im_pkl_dict=pickle.load(pickle_file)

      im_pkl_dict
```

```
[11]: {'178c5d1ab177f6ab': None,
       '178b675562417123': None,
       '178b166d5d3bed46': None,
       '178ac35ef5b46790': None,
       '178a7188cb49c5ae': None,
       '178a1f2520c7d299': None,
       '1788d4e06e8efd8f': None,
       '178882dc6ba33b38': None,
 '17882fe62d70cb4a': None,
 '1787dcf1ca93c3b2': None,
 '1786e624bc818228': None,
 '1786946ed378be54': None,
 '1786417c71c9df57': None,
 '1785ef15e49e0838': None,
 '17859c8091567e0f': None,
 '1784a47b8ac0fab4': None,
 '178453d33b1972e9': None,
 '17840163c9aaa5dc': None,
 '1783af6421a6d9cb': None,
 '17835bf5fd97d5b3': None,
 '17826a85ca0f473a': None,
 '178215f1ee928c6e': None,
 '17811e4c42f60a44': None,
 '177fd49f83332420': None,
 '177f827fd8a707e4': None,
 '177f31193a68d85d': None,
 '177ede2898bb1f8a': None,
 '177de7542c8d39ee': None}
```

[12]:
```
###Add New Keys with each run

for val in list(rr_emails):  # Use a list instead of a view
    if val not in im_pkl_dict.keys():
        im_pkl_dict.update((val, None ))
print('All Risk Range Emails have been pulled')
```

```
All Risk Range Emails have been pulled
```

[13]:
```
### Add Message to Values in Dictionary
for key, val in im_pkl_dict.items():
    if val == None:
        msg = get_message(service,'me',key)
        thisdict= dict((key, msg) for val in list(im_pkl_dict.values()))
        im_pkl_dict.update(thisdict)
        print("Message Successfully Pulled")
    else:

        print("Already Pulled")

#print(im_pkl_dict)




# Easy way to check if data came through because of the size of dictionary␣
 ↪values returned
```

```
im_pkl_dict.keys()
```

```
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
Message Successfully Pulled
```

```
[13]: dict_keys(['178c5d1ab177f6ab', '178b675562417123', '178b166d5d3bed46',
      '178ac35ef5b46790', '178a7188cb49c5ae', '178a1f2520c7d299', '1788d4e06e8efd8f',
      '178882dc6ba33b38', '17882fe62d70cb4a', '1787dcf1ca93c3b2', '1786e624bc818228',
      '1786946ed378be54', '1786417c71c9df57', '1785ef15e49e0838', '17859c8091567e0f',
      '1784a47b8ac0fab4', '178453d33b1972e9', '17840163c9aaa5dc', '1783af6421a6d9cb',
      '17835bf5fd97d5b3', '17826a85ca0f473a', '178215f1ee928c6e', '17811e4c42f60a44',
      '177fd49f83332420', '177f827fd8a707e4', '177f31193a68d85d', '177ede2898bb1f8a',
      '177de7542c8d39ee'])
```

```
[14]: ## Pickle the file again to new pkl file
      with open('im_pkl_dict.pkl','wb') as pickle_file:
          pickle.dump(im_pkl_dict, pickle_file)


      with open('im_pkl_dict.pkl','rb') as pickle_file:
          im_pkl_dict=pickle.load(pickle_file)
```

```
[15]: #Code to format each email message str pull and apply formatted results (a␣
       ↪DataFrame) into a list of DataFrame, then append list to new df

      ##The new df to which formatted dataframe data (final_df) will be appended to.
      rr_dfs=pd.
       ↪DataFrame(columns=['Asset','LERR','TERR','LastPrice','Trend','Ticker','Date']).
       ↪reset_index(drop=True)
      rr_dfs

      #The list of dataframes
      appended_data=[]

      for val in list(im_pkl_dict.values()):
          ## Beautiful Soup
          soup = BeautifulSoup(val, 'lxml')

          #print(soup.prettify())

          rr=soup.p.contents[1]
          rr=rr.contents[1]
          #rr.contents
          #Convert soup to unicode
          unicode_string=str(rr)
          #print(unicode_string)

          #Find date indicies in email
          date_found=re.search(r"\d\d/\d\d/\d\d",unicode_string) ## searching for date␣
       ↪in message (this is a constant regular expression we can take advantage of) )'
          date_found=np.asarray(date_found.span())
          #print(date_found[0])

          #grab date string
          rr_dates=unicode_string[date_found[0]:date_found[1]]
          rr_dates

          #### Grab RRs From Email Message

          #Find RR indicies in email
          #print(unicode_string.find("Bullish"))
          #print(unicode_string.find("Hedgeye's Risk Ranges"))


          #grab rr string
          rr_rrs=unicode_string[231:2063]
          #print(type(rr_rrs))
          rr_rrs
```

```
test_split = rr_rrs.split('\r\n')
test_split

test_series=pd.Series(test_split)
test_series.head()

##Break out DataFrame Columns from Series Data
cols=test_series[2:3]
#print(cols)

header = [n.split(' ') for n in cols]
#print(header)

header_df=pd.DataFrame(header)
#print(header_df)

header_df.columns = header_df.iloc[0]

header_df=header_df.drop(header_df.index[0])

#print(header_df)

##Grabbing the mislocated Trend value for implementation back into dataframe␣
↪later on
trend_header=header_df.columns[-1]
#print(trend_header)
##Grabbing the mislocated Security Name (UST10 YR) for implementation back␣
↪into dataframe later on
sec_header=header_df.columns[-2]
#print(sec_header)

#### So we have some excess columns that we need to get rid of, and we also␣
↪need to save this information into it's own dataframe b/c it is actually data,␣
↪and not data attributes or descriptions

### Drop excess columns

#### THIS CODE HAS APPENDEGAGES!!!!, if need to access old header_df, rerurn␣
↪the code ABOVE THIS CELL
header_cols=[-1,-2]
header_df.drop(header_df.columns[header_cols],axis=1,inplace=True)
header_df

raw=test_series.reset_index(drop=True)
raw=raw[3:].reset_index(drop=True)
raw.head()
```

```
## Replace (remove) spaces between indexes that do not work with regex logic␣
↪(ie Russell 2000, Nikkei 225, SP 500, etc)
raw=raw.str.replace(r"(?<=[liP]) (?=\d)",'') ## searching for first␣
↪instances of '1 str' + ' ' + '1 int'
raw=raw.str.replace(r"(?<=\d) (?=[I])",'') ## Regex specific to Nikkei 225␣
↪Index
raw.head()

##Begin Splitting series for compatible DataFrame creation
raw = raw.str.split(r"(?![a-z]) (?=\d)", expand=True)
raw.head()

squoze_raw = raw.iloc[:,-1:].squeeze()
#print(squoze_raw.head())

lrs=squoze_raw.str.split(r"(?<=\d) (?=[A-Z])", expand=True)
lrs.tail()

squoze_lrs= lrs.iloc[:,-1:].squeeze()
#print(len(squoze_lrs))
squoze_lrs.tail()

final_split=squoze_lrs.str.split(r"(?<=[A-Z]) (?=[(])" , expand=True)
ticker_split = final_split.iloc[:,0]
ticker_split=ticker_split.reset_index(drop=True)
#print(len(ticker_split))
#print(ticker_split.tail())


trend_split=final_split.iloc[:,-1:]
trend_split=trend_split.reset_index(drop=True)
#print(len(trend_split))
#print(trend_split.tail())

### Adding the Trend Value that was originally located in the df header

#Shift whole column down
trend_split=trend_split.shift(periods=1,axis=0)

#Add trend value to new empty cell
trend_split.iloc[0]=trend_header
trend_split.tail()

### Adding the Ticker Value that was originally located in the df header

#Shift whole column down
ticker_split=ticker_split.shift(periods=1,axis=0)
```

```
    #Add trend value to new empty cell
    ticker_split.iloc[0]=sec_header
    ticker_split.tail()

    final_df = raw.assign(LastPrice = lrs[0])
    final_df = final_df.assign(Trend = trend_split)
    final_df = final_df.assign(Ticker = ticker_split)
    final_df = final_df.drop([3], axis =1)
    final_df= final_df.rename(columns={0: "Asset", 1: "LERR",2: "TERR"})
    final_df['Date']=rr_dates

    appended_data.append(final_df)
rr_dfs = pd.concat(appended_data)
```

```
<ipython-input-15-8bd67882cc45>:87: FutureWarning: The default value of regex
will change from True to False in a future version.
  raw=raw.str.replace(r"(?<=[liP]) (?=\d)",'') ## searching for first instances
of '1 str' + ' ' + '1 int'
<ipython-input-15-8bd67882cc45>:88: FutureWarning: The default value of regex
will change from True to False in a future version.
  raw=raw.str.replace(r"(?<=\d) (?=[I])",'') ## Regex specific to Nikkei 225
Index
```

[16]:
```
rr_df=rr_dfs.reset_index(drop=True)
```

[17]:
```
print(rr_df)
```

```
                          Asset    LERR    TERR LastPrice      Trend  Ticker
Date
0    10-Year U.S. Treasury Yield    1.78    1.63      1.67  (BULLISH)  UST10Y
04/12/21
1     2-Year U.S. Treasury Yield    0.18    0.13      0.16  (BULLISH)   UST2Y
04/12/21
2                     S&amp;P500   4,022   4,172     4,128  (BULLISH)     SPX
04/12/21
3                   Russell2000   2,173   2,304     2,243  (BULLISH)     RUT
04/12/21
4             NASDAQ Composite  13,122  14,152    13,900  (BULLISH)   COMPQ
04/12/21
..                          ...     ...     ...       ...        ...     ...
...
913                 Amazon Inc.   3,019   3,214     3,057  (BEARISH)    AMZN
02/26/21
914               Facebook Inc.     247     270       254  (BEARISH)      FB
02/26/21
915               Alphabet Inc.   2,001   2,140     2,015  (BULLISH)   GOOGL
02/26/21
```

```
916                   Netflix Inc.     531     565       546  (BULLISH)    NFLX
02/26/21
917                     Tesla Inc.    644     None      None  (BEARISH)    TSLA
02/26/21

[918 rows x 7 columns]
```

0.1 SINGLE DATAFRAME LOAD in CODE BELOW

[18]:
```
test=get_message(service,'me','177de7542c8d39ee')
#print(test)

## Beautiful Soup
soup = BeautifulSoup(test, 'lxml')

#print(soup.prettify())

rr=soup.p.contents[1]
rr=rr.contents[1]
#rr.contents[1]
```

Grab Dates From Email Message

[19]:
```
#Convert soup to unicode
unicode_string=str(rr)
#print(unicode_string)

#Find date indicies in email
date_found=re.search(r"\d\d/\d\d/\d\d",unicode_string) ## searching for date in␣
 ↪message (this is a constant regular expression we can take advantage of) )'
date_found=np.asarray(date_found.span())
#print(date_found[0])

#grab date string
rr_dates=unicode_string[date_found[0]:date_found[1]]
#rr_dates
```

Grab RRs From Email Message

[20]:
```
#Find RR indicies in email
#print(unicode_string.find("Bullish"))
#print(unicode_string.find("Hedgeye's Risk Ranges"))


#grab rr string
rr_rrs=unicode_string[231:2063]
#print(type(rr_rrs))
#rr_rrs
```

[21]:
```
test_split = rr_rrs.split('\r\n')
#test_split
```

[22]:
```
test_series=pd.Series(test_split)
#test_series.head()
```

[23]:
```
##Break out DataFrame Columns from Series Data
cols=test_series[2:3]
#print(cols)

header = [n.split(' ') for n in cols]
#print(header)

header_df=pd.DataFrame(header)
#print(header_df)

header_df.columns = header_df.iloc[0]

header_df=header_df.drop(header_df.index[0])

#print(header_df)

##Grabbing the mislocated Trend value for implementation back into dataframe␣
 ↪later on
trend_header=header_df.columns[-1]
#print(trend_header)
##Grabbing the mislocated Security Name (UST10 YR) for implementation back into␣
 ↪dataframe later on
sec_header=header_df.columns[-2]
#print(sec_header)
```

So we have some excess columns that we need to get rid of, and we also need to save this information into it's own dataframe b/c it is actually data, and not data attributes or descriptions

[24]:
```
### Drop excess columns

#### THIS CODE HAS APPENDEGAGES!!!!, if need to access old header_df, rerurn the␣
 ↪code ABOVE THIS CELL
header_cols=[-1,-2]
header_df.drop(header_df.columns[header_cols],axis=1,inplace=True)
#header_df
```

[25]:
```
raw=test_series.reset_index(drop=True)
raw=raw[3:].reset_index(drop=True)
#raw.head()
```

[26]:
```
## Replace (remove) spaces between indexes that do not work with regex logic (ie␣
↪Russell 2000, Nikkei 225, SP 500, etc)
raw=raw.str.replace(r"(?<=[liP]) (?=\d)",'') ## searching for first instances of␣
↪'1 str' + ' ' + '1 int'
raw=raw.str.replace(r"(?<=\d) (?=[I])",'') ## Regex specific to Nikkei 225 Index
#raw.head()
```

```
<ipython-input-26-207d894cc2c1>:2: FutureWarning: The default value of regex
will change from True to False in a future version.
  raw=raw.str.replace(r"(?<=[liP]) (?=\d)",'') ## searching for first instances
of '1 str' + ' ' + '1 int'
<ipython-input-26-207d894cc2c1>:3: FutureWarning: The default value of regex
will change from True to False in a future version.
  raw=raw.str.replace(r"(?<=\d) (?=[I])",'') ## Regex specific to Nikkei 225
Index
```

[27]:
```
##Begin Splitting series for compatible DataFrame creation
raw = raw.str.split(r"(?![a-z]) (?=\d)", expand=True)
#raw.head()
```

[28]:
```
squoze_raw = raw.iloc[:,-1:].squeeze()
#print(squoze_raw.head())

lrs=squoze_raw.str.split(r"(?<=\d) (?=[A-Z])", expand=True)
#lrs.tail()
```

[29]:
```
squoze_lrs= lrs.iloc[:,-1:].squeeze()
#print(len(squoze_lrs))
#squoze_lrs.tail()
```

[30]:
```
final_split=squoze_lrs.str.split(r"(?<=[A-Z]) (?=[(])" , expand=True)
ticker_split = final_split.iloc[:,0]
ticker_split=ticker_split.reset_index(drop=True)
#print(len(ticker_split))
#print(ticker_split.tail())


trend_split=final_split.iloc[:,-1:]
trend_split=trend_split.reset_index(drop=True)
#print(len(trend_split))
#print(trend_split.tail())
```

[31]:
```
### Adding the Trend Value that was originally located in the df header

#Shift whole column down
trend_split=trend_split.shift(periods=1,axis=0)

#Add trend value to new empty cell
```

```
trend_split.iloc[0]=trend_header
#trend_split.tail()
```

[32]:
```
### Adding the Ticker Value that was originally located in the df header

#Shift whole column down
ticker_split=ticker_split.shift(periods=1,axis=0)

#Add trend value to new empty cell
ticker_split.iloc[0]=sec_header
#ticker_split.tail()
```

[33]:
```
final_df = raw.assign(LastPrice = lrs[0])

final_df = final_df.assign(Trend = trend_split)
final_df = final_df.assign(Ticker = ticker_split)

final_df = final_df.drop([3], axis =1)
final_df= final_df.rename(columns={0: "Asset", 1: "LERR",2: "TERR"})
#final_df.head()
```

[35]:
```
final_df['Date']=rr_dates
final_df
```

[35]:
```
                                            Asset    LERR    TERR LastPrice  \
Trend   Ticker      Date
0                                                    None    None      None
Trend  Neutral  02/26/21
1   INDEX BUY TRADE SELL TRADE PREV. CLOSE UST10Y ...    None    None      None
None      None  02/26/21
2                          10-Year U.S. Treasury Yield    1.56    1.22      1.54
None      None  02/26/21
3                           2-Year U.S. Treasury Yield    0.18    0.12      0.17
(BULLISH)    UST2Y  02/26/21
4                                         S&amp;P500   3,811   3,957     3,829
(BULLISH)      SPX  02/26/21
5                                       Russell2000   1,980   2,301     2,200
(BULLISH)      RUT  02/26/21
6                                 NASDAQ Composite  12,992  14,326    13,119
(BULLISH)    COMPQ  02/26/21
7                   Technology Select Sector SPDR Fund  129.96  140.15    130.00
(BULLISH)      XLK  02/26/21
8                       Energy Select Sector SPDR Fund   46.17   51.02     49.32
(BULLISH)      XLE  02/26/21
9                   Financials Select Sector SPDR Fund   31.39   33.81     32.94
(BULLISH)      XLF  02/26/21
10                   Utilities Select Sector SPDR Fund   58.56   61.28     59.46
```

(BEARISH)	XLU	02/26/21				
11			VanEck Vectors Gold Miners ETF	31.60	34.45	32.33
(BEARISH)	GDX	02/26/21				
12			Shanghai Composite	3,473	3,720	3,509
(BULLISH)	SSEC	02/26/21				
13			Nikkei225Index	28,906	30,652	28,966
(BULLISH)	NIKK	02/26/21				
14			German DAX Composite	13,718	14,140	13,879
(BULLISH)	DAX	02/26/21				
15			Volatility Index	17.80	30.99	28.89
(BEARISH)	VIX	02/26/21				
16			U.S. Dollar Index	89.71	90.90	90.14
(BEARISH)	USD	02/26/21				
17			Euro to U.S. Dollar	1.205	1.223	1.218
(BULLISH)	EUR/USD	02/26/21				
18			U.S. Dollar to Japanese Yen	104.89	106.49	106.21
(BULLISH)	USD/JPY	02/26/21				
19			British Pound to U.S. Dollar	1.383	1.421	1.401
(BULLISH)	GBP/USD	02/26/21				
20			Canadian Dollar to U.S. Dollar	0.79	0.80	0.79
(BULLISH)	CAD/USD	02/26/21				
21			U.S. Dollar to Swiss Franc	0.88	0.91	0.90
(BEARISH)	USD/CHF	02/26/21				
22			Light Crude Oil Spot Price	58.41	64.14	63.53
(BULLISH)	WTIC	02/26/21				
23			Natural Gas Spot Price	2.65	3.24	2.78
(BULLISH)	NATGAS	02/26/21				
24			Gold Spot Price	1,748	1,818	1,775
(BEARISH)	GOLD	02/26/21				
25			Copper Spot Price	3.99	4.43	4.26
(BULLISH)	COPPER	02/26/21				
26			Silver Spot Price	26.49	28.20	27.68
(BULLISH)	SILVER	02/26/21				
27			Microsoft Corp.	226	250	228
(BULLISH)	MSFT	02/26/21				
28			Apple Inc.	118	130	120
(BEARISH)	AAPL	02/26/21				
29			Amazon Inc.	3,019	3,214	3,057
(BEARISH)	AMZN	02/26/21				
30			Facebook Inc.	247	270	254
(BEARISH)	FB	02/26/21				
31			Alphabet Inc.	2,001	2,140	2,015
(BULLISH)	GOOGL	02/26/21				
32			Netflix Inc.	531	565	546
(BULLISH)	NFLX	02/26/21				
33			Tesla Inc.	644	None	None
(BEARISH)	TSLA	02/26/21				

From here, these ranges can be grabbed daily and uploaded to a csv or excel file. The process of loading is not included in this repository because the same logic is in another one of my repositories and I wanted to get this out quickly for the #HedgeyeNation.

```
[ ]:
```