Download FOMC Data (Template)

Scrape data from the Federal Reserve website and archives. Also a basic Colab Environment template.

Environment

```
# -*- coding: utf-8 -*-

# ENVIRONMENT CHECK:
import sys, os, inspect, site, pprint
# Check whether in Colab:
IN_COLAB = 'google.colab' in sys.modules
if IN_COLAB == True:
  print('YES, this is a Google Colaboratory environment.')
else:
  print('NO, this is not a Google Colaboratory environment.')
print(' ')

# Python installation files:
stdlib = os.path.dirname(inspect.getfile(os))
python_version = !python --version
print('Python Standard Library is located in:\n' + stdlib)
print(' ')
print('This environment is using {}'.format(str(python_version[0])))
print(' ')
print('Local system packages are located in:')
pprint.pprint(site.getsitepackages())
print(' ')
print('Local user packages are located in:\n' + site.getusersitepackages())

# Installed packages:
#!pip list -v
#!pip list --user -v
```

```
YES, this is a Google Colaboratory environment.

Python Standard Library is located in:
/usr/lib/python3.6

This environment is using Python 3.6.9

Local system packages are located in:
```

```
     ['/usr/local/lib/python3.6/dist-packages',
      '/usr/lib/python3/dist-packages',
      '/usr/lib/python3.6/dist-packages']

     Local user packages are located in:
     /root/.local/lib/python3.6/site-packages


# Mount Google Drive:
if IN_COLAB:
  from google.colab import drive
  drive.mount('/content/drive', force_remount=True)


     Mounted at /content/drive
```

Saved successfully!

```
%cd "/content/drive/MyDrive/Colab Notebooks/proj2/src"
!ls -al

     /content/drive/MyDrive/Colab Notebooks/proj2/src
     total 12484
     -rw------- 1 root root   70492 Jan 25 08:47 0_FOMC_Analysis_Download_Data.ipynb
     -rw------- 1 root root 1297662 Jan 25 08:42 1_FOMC_Analysis_Preliminary.ipynb
     -rw------- 1 root root 1881863 Jan 25 06:22 2_FOMC_Analysis_Preprocess_NonText.ipynb
     -rw------- 1 root root  675104 Jan 25 08:46 3_FOMC_Analysis_Preprocess_Text.ipynb
     -rw------- 1 root root 2659457 Jan 25 00:59 4_FOMC_Analysis_EDA_FE_NonText.ipynb
     -rw------- 1 root root 1282658 Jan 25 00:59 5_FOMC_Analysis_Baseline.ipynb
     -rw------- 1 root root 4138442 Jan 25 06:30 6_FOMC_Analysis_Model_Train.ipynb
     -rw------- 1 root root  358128 Jan 24 19:04 7_FOMC_Analysis_Sentence.ipynb
     -rw------- 1 root root  355706 Jan 25 08:41 7_FOMC_Corpora.ipynb
     drwx------ 2 root root    4096 Jan 25 07:52 C:
     drwx------ 2 root root    4096 Nov  6 04:45 data
     drwx------ 2 root root    4096 Jan 25 06:54 final
     -rw------- 1 root root    7138 Jan 25 08:27 FomcGetCalendar.py
     drwx------ 2 root root    4096 Nov  6 04:45 fomc_get_data
     -rw------- 1 root root    4088 Jan 25 08:46 FomcGetData.py
     drwx------ 2 root root    4096 Nov  6 14:19 .idea
     drwx------ 2 root root    4096 Nov  6 04:45 .ipynb_checkpoints
     -rw------- 1 root root       0 Jan 25 07:21 log_model.txt
     drwx------ 2 root root    4096 Nov 21 01:36 original
     -rw------- 1 root root     368 Nov 14 17:10 pdf2text.py
     -rw------- 1 root root    1932 Jan 24 01:10 QuandlGetData.py
     -rw------- 1 root root   18358 Jan 25 06:41 README.md
     -rw------- 1 root root     222 Nov 14 17:10 requirements.txt


# Define Path Variables:
if IN_COLAB:
  employment_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/Employment/'
  cpi_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/CPI/'
```

```
    fed_rates_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/FEDRates/'
    fx_rates_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/FXRates/'
    gdp_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/GDP/'
    ism_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/ISM/'
    sales_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/Sales/'
    treasury_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/Treasury/'
    fomc_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/FOMC/'
    preprocessed_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/preprocessed/'
    train_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/train_data/'
    output_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/result/'
    keyword_lm_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/LoughranMcDonald/'
    glove_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/GloVe/'
    model_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/models/'

else:
    employment_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/Employment/'
    cpi_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/CPI/'
    fed_rates_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/FEDRates/'
    fx_rates_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/FXRates/'
    gdp_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/GDP/'
    ism_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/ISM/'
    sales_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/Sales/'
    treasury_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/Treasury/'
    fomc_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/FOMC/'
    preprocessed_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/preprocessed/'
    train_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/train_data/'
    output_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/result/'
    keyword_lm_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/LoughranMcDonald/'
    glove_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/GloVe/'
    model_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/models/'
```

Saved successfully!

Packages

```
#if IN_COLAB:
#  # Uninstall existing versions:
#  !pip uninstall bs4 -y
#  !pip uninstall textract -y
#  !pip uninstall numpy -y
#  !pip uninstall pandas -y
#  !pip uninstall requests -y
#  !pip uninstall tqdm -y
#  !pip uninstall nltk -y
#  !pip uninstall quandl -y
#  !pip uninstall scikit-plot -y
#  !pip uninstall seaborn -y
```

```
#  !pip uninstall sklearn -y
#  !pip uninstall torch -y
#  !pip uninstall transformers -y
#  !pip uninstall wordcloud -y
#  !pip uninstall xgboost -y
#
#  # Install packages:
#  !pip install bs4==0.0.1
#  !pip install textract==1.6.3
#  !pip install numpy==1.19.4
#  !pip install pandas==1.1.4
#  !pip install requests==2.24.0
#  !pip install tqdm==4.51.0
#  !pip install nltk==3.5
#  !pip install quandl==3.5.3
```

Saved successfully!

```
#  !pip install sklearn==0.0
#  !pip install torch==1.7.1+cu101 torchvision==0.8.2+cu101 -f https://download.pytorch.org/whl/torch_stable.html
#  !pip install transformers==3.5.0
#  !pip install wordcloud==1.8.0
#  !pip install xgboost==1.2.1
##  os.kill(os.getpid(), 9)
```

Inspect Packages

```
!pip list -v
!pip list --user -v
```

```
    sympy                         1.1.1          /usr/local/lib/python3.6/dist-packages pip
    tables                        3.4.4          /usr/local/lib/python3.6/dist-packages pip
    tabulate                      0.8.7          /usr/local/lib/python3.6/dist-packages pip
    tblib                         1.7.0          /usr/local/lib/python3.6/dist-packages pip
    tensorboard                   2.4.0          /usr/local/lib/python3.6/dist-packages pip
    tensorboard-plugin-wit        1.7.0          /usr/local/lib/python3.6/dist-packages pip
    tensorboardcolab              0.0.22         /usr/local/lib/python3.6/dist-packages pip
    tensorflow                    2.4.0          /usr/local/lib/python3.6/dist-packages pip
    tensorflow-addons             0.8.3          /usr/local/lib/python3.6/dist-packages pip
    tensorflow-datasets           4.0.1          /usr/local/lib/python3.6/dist-packages pip
    tensorflow-estimator          2.4.0          /usr/local/lib/python3.6/dist-packages pip
    tensorflow-gcs-config         2.4.0          /usr/local/lib/python3.6/dist-packages pip
    tensorflow-hub                0.11.0         /usr/local/lib/python3.6/dist-packages pip
    tensorflow-metadata           0.26.0         /usr/local/lib/python3.6/dist-packages pip
    tensorflow-privacy            0.2.2          /usr/local/lib/python3.6/dist-packages pip
    tensorflow-probability        0.12.1         /usr/local/lib/python3.6/dist-packages pip
    termcolor                     1.1.0          /usr/local/lib/python3.6/dist-packages pip
    terminado                     0.9.2          /usr/local/lib/python3.6/dist-packages pip
```

```
terminado                     0.9.2          /usr/local/lib/python3.6/dist-packages pip
testpath                      0.4.4          /usr/local/lib/python3.6/dist-packages pip
text-unidecode                1.3            /usr/local/lib/python3.6/dist-packages pip
textblob                      0.15.3         /usr/local/lib/python3.6/dist-packages pip
textgenrnn                    1.4.1          /usr/local/lib/python3.6/dist-packages pip
textract                      1.6.3          /usr/local/lib/python3.6/dist-packages pip
Theano                        1.0.5          /usr/local/lib/python3.6/dist-packages pip
thinc                         7.4.0          /usr/local/lib/python3.6/dist-packages pip
tifffile                      2020.9.3       /usr/local/lib/python3.6/dist-packages pip
tokenizers                    0.9.3          /usr/local/lib/python3.6/dist-packages pip
toml                          0.10.2         /usr/local/lib/python3.6/dist-packages pip
toolz                         0.11.1         /usr/local/lib/python3.6/dist-packages pip
torch                         1.7.1+cu101    /usr/local/lib/python3.6/dist-packages pip
torchsummary                  1.5.1          /usr/local/lib/python3.6/dist-packages pip
torchtext                     0.3.1          /usr/local/lib/python3.6/dist-packages pip
torchvision                   0.8.2+cu101    /usr/local/lib/python3.6/dist-packages pip
tornado                       5.1.1          /usr/local/lib/python3.6/dist-packages pip
tqdm                          4.51.0         /usr/local/lib/python3.6/dist-packages pip
traitlets                     4.3.3          /usr/local/lib/python3.6/dist-packages pip
transformers                  3.5.0          /usr/local/lib/python3.6/dist-packages pip
tweepy                        3.6.0          /usr/local/lib/python3.6/dist-packages pip
typeguard                     2.7.1          /usr/local/lib/python3.6/dist-packages pip
typing-extensions             3.7.4.3        /usr/local/lib/python3.6/dist-packages pip
tzlocal                       1.5.1          /usr/local/lib/python3.6/dist-packages pip
umap-learn                    0.4.6          /usr/local/lib/python3.6/dist-packages pip
uritemplate                   3.0.1          /usr/local/lib/python3.6/dist-packages pip
urllib3                       1.24.3         /usr/local/lib/python3.6/dist-packages pip
vega-datasets                 0.9.0          /usr/local/lib/python3.6/dist-packages pip
wasabi                        0.8.0          /usr/local/lib/python3.6/dist-packages pip
wcwidth                       0.2.5          /usr/local/lib/python3.6/dist-packages pip
webencodings                  0.5.1          /usr/local/lib/python3.6/dist-packages pip
Werkzeug                      1.0.1          /usr/local/lib/python3.6/dist-packages pip
wheel                         0.36.2         /usr/local/lib/python3.6/dist-packages pip
widgetsnbextension            3.5.1          /usr/local/lib/python3.6/dist-packages pip
wordcloud                     1.8.0          /usr/local/lib/python3.6/dist-packages pip
wrapt                         1.12.1         /usr/local/lib/python3.6/dist-packages pip
xarray                        0.15.1         /usr/local/lib/python3.6/dist-packages pip
xgboost                       1.2.1          /usr/local/lib/python3.6/dist-packages pip
xkit                          0.0.0          /usr/lib/python3/dist-packages
xlrd                          1.2.0          /usr/local/lib/python3.6/dist-packages pip
XlsxWriter                    1.3.7          /usr/local/lib/python3.6/dist-packages pip
xlwt                          1.3.0          /usr/local/lib/python3.6/dist-packages pip
yellowbrick                   0.9.1          /usr/local/lib/python3.6/dist-packages pip
```

Saved successfully!

Import Packages

```
# Python libraries
import pprint
import datetime as dt
import re
import io
```

```
import io
import os
import pickle
from tqdm.notebook import tqdm
import time
import logging
import random
from collections import defaultdict, Counter
import xgboost as xgb
import codecs
pprint.pprint(sys.path)

# Data Science modules
import numpy as np
import pandas as pd
```



```
="darkgrid")
#plt.style.use('ggplot')

# Import Scikit-learn models
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import accuracy_score, f1_score, plot_confusion_matrix
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, ExtraTreesClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression, Perceptron, SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn import model_selection
from sklearn.model_selection import GridSearchCV, cross_val_score, cross_validate, StratifiedKFold, learning_curve, RandomizedSearchCV
import scikitplot as skplt

# Import nltk modules and download dataset
import nltk
from nltk.corpus import stopwords
from nltk.util import ngrams
from nltk.tokenize import word_tokenize, sent_tokenize

# Import Pytorch modules
import torch
from torch import nn, optim
import torch.nn.functional as F
from torch.utils.data import (DataLoader, RandomSampler, SequentialSampler, TensorDataset)
from torch.autograd import Variable
from torch.optim import Adam, AdamW
```

```
# Import Transformers
#from transformers import *
from transformers import BertTokenizer, BertForSequenceClassification, BertModel
```

```
['',
 '/env/python',
 '/usr/lib/python36.zip',
 '/usr/lib/python3.6',
 '/usr/lib/python3.6/lib-dynload',
 '/usr/local/lib/python3.6/dist-packages',
 '/usr/lib/python3/dist-packages',
 '/usr/local/lib/python3.6/dist-packages/IPython/extensions',
 '/root/.ipython']
```

Saved successfully!

```
## Use TPU Runtime:
#if IN_COLAB:
#  assert os.environ['COLAB_TPU_ADDR'], 'Make sure to select TPU from Edit > Notebook setting > Hardware accelerator'
#  VERSION = "20200220"
#  !curl https://raw.githubusercontent.com/pytorch/xla/master/contrib/scripts/env-setup.py -o pytorch-xla-env-setup.py
#  !python pytorch-xla-env-setup.py --version $VERSION
```

```
## Use GPU Runtime:
#if IN_COLAB:
#  if torch.cuda.is_available():
#    torch.cuda.get_device_name(0)
#    gpu_info = !nvidia-smi
#    gpu_info = '\n'.join(gpu_info)
#    print(gpu_info)
#  else:
#    print('Select the Runtime > "Change runtime type" menu to enable a GPU accelerator, and then re-execute this cell.')
#    #os.kill(os.getpid(), 9)
```

Finalize Setup

```
#  Download nltk dataset
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

stop = set(stopwords.words('english'))
```

```
stop = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
```

```
# Set logger
logger = logging.getLogger('mylogger')
logger.setLevel(logging.INFO)

timestamp = time.strftime("%Y.%m.%d_%H.%M.%S", time.localtime())
                                    ctime)s][%(levelname)s] ## %(message)s')

fh = logging.FileHandler('log_model.txt')
fh.setLevel(logging.DEBUG)
fh.setFormatter(formatter)
logger.addHandler(fh)

ch = logging.StreamHandler()
ch.setLevel(logging.INFO)
ch.setFormatter(formatter)
logger.addHandler(ch)
```

Saved successfully!

```
# Set Random Seed
random.seed(42)
np.random.seed(42)
torch.manual_seed(42)
torch.cuda.manual_seed(42)
rand_seed = 42
```

```
# Set Seaborn Style
sns.set(style='white', context='notebook', palette='deep')
```

Load preprocessed data

```
if IN_COLAB:
  employment_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/Employment/'
  cpi_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/CPI/'
  fed_rates_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/FEDRates/'
  fx_rates_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/FXRates/'
```

```
  gdp_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/GDP/'
  ism_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/ISM/'
  sales_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/Sales/'
  treasury_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/Treasury/'
  fomc_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/FOMC/'
  preprocessed_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/preprocessed/'
  train_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/train_data/'
  output_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/result/'
  keyword_lm_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/LoughranMcDonald/'
  glove_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/GloVe/'
  model_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/models/'
  graph_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/graphs/'
else:
  employment_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/Employment/'
  cpi_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/CPI/'
  fed_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/FEDRates/'
  fx_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/FXRates/'
  gdp_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/GDP/'
  ism_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/ISM/'
  sales_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/Sales/'
  treasury_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/Treasury/'
  fomc_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/FOMC/'
  preprocessed_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/preprocessed/'
  train_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/train_data/'
  output_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/result/'
  keyword_lm_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/LoughranMcDonald/'
  glove_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/GloVe/'
  model_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/models/'
  graph_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/graphs/'
```

Saved successfully!

Save Data

```
if IN_COLAB:
  def save_data(df, file_name, dir_name=train_dir, index_csv=True):
    if not os.path.exists(dir_name):
      os.mkdir(dir_name)
    # Save results to a picke file
    file = open(dir_name + file_name + '.pickle', 'wb')
    pickle.dump(df, file)
    file.close()
    print('Successfully saved {}.pickle. in {}'.format(file_name, dir_name + file_name + '.pickle'))
    # Save results to a csv file
    df.to_csv(dir_name + file_name + '.csv', index=True)
    print('Successfully saved {}.csv. in {}'.format(file_name, dir_name + file_name + '.csv'))
```

```
else:
  def save_data(df, file_name, dir_name=train_dir):
    # Save results to a .picke file
    file = open(dir_name + file_name + '.pickle', 'wb')
    pickle.dump(df, file)
    file.close()
    print('Successfully saved {}.pickle. in {}'.format(file_name, dir_name + file_name + '.pickle'))
    # Save results to a .csv file
    df.to_csv(dir_name + file_name + '.csv', index=True)
    print('Successfully saved {}.csv. in {}'.format(file_name, dir_name + file_name + '.csv'))
```

FomcGetData Testing:

Saved successfully!

Packages

Inspect Packages

```
!cat fomc_get_data/FomcBase.py
```

```
            filepath = dir_name + file_name + '.pickle'
            print("")
            print("Writing to ", filepath)
            with open(filepath, "wb") as output_file:
                pickle.dump(df, output_file)
            file.close()
            print('Successfully saved {}.pickle in {}'.format(file_name, filepath))
            filepath = dir_name + file_name + '.csv'
            print("Writing to ", filepath)
            df.to_csv(filepath, index=index_csv)
            print('Successfully saved {}.csv in {}'.format(file_name, filepath))

        def save_data(df, file_name, dir_name='/content/drive/My Drive/Colab Notebooks/proj2/src/data/FOMC/', index_csv=False):
            if not os.path.exists(dir_name):
              os.mkdir(dir_name)
            # Save results to a picke file
            pickle_path = dir_name + file_name + '.pickle'
            with open(pickle_path, "wb") as output_file:
                pickle.dump(df, output_file)
            file.close()
            print('Successfully saved {}.pickle in {}'.format(file_name, pickle_path))
            # Save results to a csv file
            csv_path = dir_name + file_name + '.csv'
            df.to_csv(csv_path, index=index_csv)
            print('Successfully saved {}.csv in {}'.format(file_name, csv_path))
```

```
        def dump_df(df, file_name, dir_name='/content/drive/My Drive/Colab Notebooks/proj2/src/data/FOMC/', index_csv=False):
            if not os.path.exists(dir_name):
                os.mkdir(dir_name)
            filepath = dir_name + file_name + '.pickle'
            print("")
            print("Writing to ", filepath)
            with open(filepath, "wb") as output_file:
                pickle.dump(df, output_file)
            file.close()
            print('Successfully saved {}.pickle in {}'.format(file_name, filepath))
            filepath = dir_name + file_name + '.csv'
            print("Writing to ", filepath)
            df.to_csv(filepath, index=index_csv)
            print('Successfully saved {}.csv in {}'.format(file_name, filepath))

        def __init__(self, df, prefix="FOMC_", target="contents"):
            '''
            Save each row in the dataframe df to text files
            '''
            tmp_dates = []
            tmp_seq = 1
            for i, row in self.df.iterrows():
                cur_date = row['date'].strftime('%Y-%m-%d')
                if cur_date in tmp_dates:
                    tmp_seq += 1
                    filepath = self.base_dir + prefix + cur_date + "-" + str(tmp_seq) + ".txt"
                else:
                    tmp_seq = 1
                    filepath = self.base_dir + prefix + cur_date + ".txt"
                tmp_dates.append(cur_date)
                if self.verbose: print("Writing to ", filepath)
                os.makedirs(os.path.dirname(filepath), exist_ok=True)
                with open(filepath, "w", encoding="utf-8") as output_file:
                    output_file.write(row[target])
```

Saved successfully!

```
!cat fomc_get_data/FomcMeetingScript.py
```

```
            self.speakers = []
            self.dates = []

            r = requests.get(self.calendar_url)
            soup = BeautifulSoup(r.text, 'html.parser')

            # Meeting Script can be found only in the archive as it is published after five years
            if from_year > 2014:
                print("Meeting scripts are available for 2014 or older")
            if from_year <= 2014:
                for year in range(from_year, 2015):
                    yearly_contents = []
                    fomc_yearly_url = self.base_url + '/monetarypolicy/fomchistorical' + str(year) + '.htm'
                    r_year = requests.get(fomc_yearly_url)
                    soup_yearly = BeautifulSoup(r_year.text, 'html.parser')
                    meeting_scripts = soup_yearly.find_all('a', href=re.compile('^/monetarypolicy/files/FOMC\d{8}meeting.pdf'))
```

```
                for meeting_script in meeting_scripts:
                    self.links.append(meeting_script.attrs['href'])
                    self.speakers.append(self._speaker_from_date(self._date_from_link(meeting_script.attrs['href'])))
                    self.titles.append('FOMC Meeting Transcript')
                    self.dates.append(datetime.strptime(self._date_from_link(meeting_script.attrs['href']), '%Y-%m-%d'))
                if self.verbose: print("YEAR: {} - {} meeting scripts found.".format(year, len(meeting_scripts)))
            print("There are total ", len(self.links), ' links for ', self.content_type)

    def _add_article(self, link, index=None):
        '''
        Override a private function that adds a related article for 1 link into the instance variable
        The index is the index in the article to add to.
        Due to concurrent processing, we need to make sure the articles are stored in the right order
        '''
        if self.verbose:
            sys.stdout.write(".")

        link_url = self.base_url + link
        pdf_filepath = self.base_dir + 'script_pdf/FOMC_MeetingScript_' + self._date_from_link(link) + '.pdf'

        # Scripts are provided only in pdf. Save the pdf and pass the content
        res = requests.get(link_url)
        with open(pdf_filepath, 'wb') as f:
            f.write(res.content)

        # Extract text from the pdf
        # pdf_file_parsed = parser.from_file(pdf_filepath)
        # paragraphs = re.sub('(\n)(\n)+', '\n', pdf_file_parsed['content'].strip())
        pdf_file_parsed = textract.process(pdf_filepath).decode('utf-8-sig')
        paragraphs = re.sub('(\n)(\n)+', '\n', pdf_file_parsed.strip())
        paragraphs = paragraphs.split('\n')

        section = -1
        paragraph_sections = []
        for paragraph in paragraphs:
            if not re.search('^(page|january|february|march|april|may|june|july|august|september|october|november|december|jan|feb|mar|apr|may
                if len(re.findall(r'[A-Z]', paragraph[:10])) > 5 and not re.search('(present|frb/us|abs cdo|libor|rp-ioer|lsaps|cusip|nairu|s
                    section += 1
                    paragraph_sections.append("")
                if section >= 0:
                    paragraph_sections[section] += paragraph
        self.articles[index] = "\n\n[SECTION]\n\n".join([paragraph for paragraph in paragraph_sections])
```

Saved successfully!

```
!cat fomc_get_data/FomcMinutes.py
                    self.titles.append('FOMC Meeting Minutes')
                    self.dates.append(datetime.strptime(self._date_from_link(yearly_content.attrs['href']), '%Y-%m-%d'))
                    # Sometimes minutes carries the first day of the meeting before 2000, so update them to the 2nd day
                    if self.dates[-1] == datetime(1996,1,30):
                        self.dates[-1] = datetime(1996,1,31)
                    elif self.dates[-1] == datetime(1996,7,2):
                        self.dates[-1] = datetime(1996,7,3)
```

```
                elif self.dates[-1] == datetime(1997,2,4):
                    self.dates[-1] = datetime(1997,2,5)
                elif self.dates[-1] == datetime(1997,7,1):
                    self.dates[-1] = datetime(1997,7,2)
                elif self.dates[-1] == datetime(1998,2,3):
                    self.dates[-1] = datetime(1998,2,4)
                elif self.dates[-1] == datetime(1998,6,30):
                    self.dates[-1] = datetime(1998,7,1)
                elif self.dates[-1] == datetime(1999,2,2):
                    self.dates[-1] = datetime(1999,2,3)
                elif self.dates[-1] == datetime(1999,6,29):
                    self.dates[-1] = datetime(1999,6,30)

        if self.verbose: print("YEAR: {} - {} links found.".format(year, len(yearly_contents)))
print("There are total ", len(self.links), ' links for ', self.content_type)

, index=None):

tion that adds a related article for 1 link into the instance variable
The index is the index in the article to add to.
Due to concurrent prcessing, we need to make sure the articles are stored in the right order
'''
if self.verbose:
    sys.stdout.write(".")
    sys.stdout.flush()

res = requests.get(self.base_url + link)
html = res.text

# p tag is not properly closed in many cases
html = html.replace('<P', '<p').replace('</P>', '</p>')
html = html.replace('<p', '</p><p').replace('</p><p', '<p', 1)

# remove all after appendix or references
x = re.search(r'(<b>references|<b>appendix|<strong>references|<strong>appendix)', html.lower())
if x:
    html = html[:x.start()]
    html += '</body></html>'
# Parse html text by BeautifulSoup
article = BeautifulSoup(html, 'html.parser')

#if link == '/fomc/MINUTES/1994/19940517min.htm':
#    print(article)

# Remove footnote
for fn in article.find_all('a', {'name': re.compile('fn\d')}):
    # if fn.parent:
    #     fn.parent.decompose()
    # else:
    #     fn.decompose()
    fn.decompose()
# Get all p tag
paragraphs = article.findAll('p')
```

Saved successfully!

```
!cat fomc_get_data/FomcSpeech.py
```

```
                    # Sometimes the same link is put for watch live video. Skip those.
                    if speech_link.find({'class': 'watchLive'}):
                        continue

                    # Add link, title and date
                    self.links.append(speech_link.attrs['href'])
                    self.titles.append(speech_link.get_text())
                    self.dates.append(datetime.strptime(self._date_from_link(speech_link.attrs['href']), '%Y-%m-%d'))

                    # Add speaker
                    # Somehow the speaker is before the link in 1997 only, whereas the others is vice-versa
                    if year == 1997:
                        # Somehow only the linke for December 15 speech has speader after the link in 1997 page.
                        if speech_link.get('href') == '/boarddocs/speeches/1997/19971215.htm':
                            tmp_speaker = speech_link.parent.next_sibling.next_element.get_text().replace('\n', '').strip()
                        else:
                            tmp_speaker = speech_link.parent.previous_sibling.previous_sibling.get_text().replace('\n', '').strip()
                    else:
                        # Somehow 20051128 and 20051129 are structured differently
                        if speech_link.get('href') in ('/boarddocs/speeches/2005/20051128/default.htm', '/boarddocs/speeches/2005/20051129/default
                            tmp_speaker = speech_link.parent.previous_sibling.previous_sibling.get_text().replace('\n', '').strip()
                        tmp_speaker = speech_link.parent.next_sibling.next_element.get_text().replace('\n', '').strip()
                        # When a video icon is placed between the link and speaker
                        if tmp_speaker in ('Watch Live', 'Video'):
                            tmp_speaker = speech_link.parent.next_sibling.next_sibling.next_sibling.next_element.get_text().replace('\n', '').stri
                    self.speakers.append(tmp_speaker)
                if self.verbose: print("YEAR: {} - {} speeches found.".format(year, len(speech_links)))

        def _add_article(self, link, index=None):
            '''
            Override a private function that adds a related article for 1 link into the instance variable
            The index is the index in the article to add to.
            Due to concurrent prcessing, we need to make sure the articles are stored in the right order
            '''
            if self.verbose:
                sys.stdout.write(".")
                sys.stdout.flush()

            res = requests.get(self.base_url + link)
            html = res.text
            # p tag is not properly closed in many cases
            html = html.replace('<P', '<p').replace('</P>', '</p>')
            html = html.replace('<p', '</p><p').replace('</p><p', '<p', 1)
            # remove all after appendix or references
            x = re.search(r'(<b>references|<b>appendix|<strong>references|<strong>appendix)', html.lower())
            if x:
                html = html[:x.start()]
                html += '</body></html>'
            # Parse html text by BeautifulSoup
            article = BeautifulSoup(html, 'html.parser')
```

Saved successfully!

```
            # Remove footnote
            for fn in article.find_all('a', {'name': re.compile('fn\d')}):
                if fn.parent:
                    fn.parent.decompose()
                else:
                    fn.decompose()
            # Get all p tag
            paragraphs = article.findAll('p')
            self.articles[index] = "\n\n[SECTION]\n\n".join([paragraph.get_text().strip() for paragraph in paragraphs])
```

```
!cat fomc_get_data/FomcStatement.py
```

```
            # Getting links from current page. Meetin scripts are not available.
            if self.verbose: print("Getting links for statements...")
            contents = soup.find_all('a', href=re.compile('^/newsevents/pressreleases/monetary\d{8}[ax].htm'))
            self.links = [content.attrs['href'] for content in contents]
            self.speakers = [self._speaker_from_date(self._date_from_link(x)) for x in self.links]
            self.titles = ['FOMC Statement'] * len(self.links)
            self.dates = [datetime.strptime(self._date_from_link(x), '%Y-%m-%d') for x in self.links]
            # Correct some date in the link does not match with the meeting date
            for i, m_date in enumerate(self.dates):
                if m_date == datetime(2019,10,11):
                    self.dates[i] = datetime(2019,10,4)

            if self.verbose: print("{} links found in the current page.".format(len(self.links)))

            # Archived before 2015
            if from_year <= 2014:
                print("Getting links from archive pages...")
                for year in range(from_year, 2015):
                    yearly_contents = []
                    fomc_yearly_url = self.base_url + '/monetarypolicy/fomchistorical' + str(year) + '.htm'
                    r_year = requests.get(fomc_yearly_url)
                    soup_yearly = BeautifulSoup(r_year.text, 'html.parser')
                    yearly_contents = soup_yearly.findAll('a', text = 'Statement')
                    for yearly_content in yearly_contents:
                        self.links.append(yearly_content.attrs['href'])
                        self.speakers.append(self._speaker_from_date(self._date_from_link(yearly_content.attrs['href'])))
                        self.titles.append('FOMC Statement')
                        self.dates.append(datetime.strptime(self._date_from_link(yearly_content.attrs['href']), '%Y-%m-%d'))
                        # Correct some date in the link does not match with the meeting date
                        if self.dates[-1] == datetime(2007,6,18):
                            self.dates[-1] = datetime(2007,6,28)
                        elif self.dates[-1] == datetime(2007,8,17):
                            self.dates[-1] = datetime(2007,8,16)
                        elif self.dates[-1] == datetime(2008,1,22):
                            self.dates[-1] = datetime(2008,1,21)
                        elif self.dates[-1] == datetime(2008,3,11):
                            self.dates[-1] = datetime(2008,3,10)
                        elif self.dates[-1] == datetime(2008,10,8):
                            self.dates[-1] = datetime(2008,10,7)
```

```
                if self.verbose: print("YEAR: {} - {} links found.".format(year, len(yearly_contents)))

        print("There are total ", len(self.links), ' links for ', self.content_type)

    def _add_article(self, link, index=None):
        '''
        Override a private function that adds a related article for 1 link into the instance variable
        The index is the index in the article to add to.
        Due to concurrent processing, we need to make sure the articles are stored in the right order
        '''
        if self.verbose:
            sys.stdout.write(".")
            sys.stdout.flush()

        res = requests.get(self.base_url + link)
        html = res.text
        (html, 'html.parser')
        ndAll('p')
        "\n\n[SECTION]\n\n".join([paragraph.get_text().strip() for paragraph in paragraphs])
```

Saved successfully!

```
!cat fomc_get_data/FomcTestimony.py
                            speaker = doc_link.parent.parent.next_element.next_element.get_text().replace('\n', '').strip()
                            date_str = doc_link.parent.parent.next_element.replace('\n', '').strip()
                        elif doc_link.get('href') in ('/boarddocs/testimony/1997/19970121.htm'):
                            title = doc_link.parent.parent.find_next('em').get_text().replace('\n', '').strip()
                            speaker = doc_link.parent.parent.find_next('strong').get_text().replace('\n', '').strip()
                            date_str = doc_link.get_text()
                        else:
                            title = doc_link.get_text()
                            speaker = doc_link.parent.find_next('div').get_text().replace('\n', '').strip()

                            # When a video icon is placed between the link and speaker
                            if speaker in ('Watch Live', 'Video'):
                                speaker = doc_link.parent.find_next('p').find_next('p').get_text().replace('\n', '').strip()
                            date_str = doc_link.parent.parent.next_element.replace('\n', '').strip()

                        self.titles.append(doc_link.get_text())
                        self.speakers.append(speaker)
                        self.dates.append(datetime.strptime(date_str, '%B %d, %Y'))

                if self.verbose: print("YEAR: {} - {} testimony docs found.".format(year, len(doc_links)))

    def _add_article(self, link, index=None):
        if self.verbose:
            sys.stdout.write(".")
            sys.stdout.flush()

        link_url = self.base_url + link
        # article_date = self._date_from_link(link)

        #print(link_url)
```

```
            # date of the article content
            # self.dates.append(article_date)

            res = requests.get(self.base_url + link)
            html = res.text

            # p tag is not properly closed in many cases
            html = html.replace('<P', '<p').replace('</P>', '</p>')
            html = html.replace('<p', '</p><p').replace('</p><p', '<p', 1)

            # remove all after appendix or references
            x = re.search(r'(<b>references|<b>appendix|<strong>references|<strong>appendix)', html.lower())
            if x:
                html = html[:x.start()]
                html += '</body></html>'

            autifulSoup
            (html, 'html.parser')

            # Remove footnote
            for fn in article.find_all('a', {'name': re.compile('fn\d')}):
                # if fn.parent:
                #     fn.parent.decompose()
                # else:
                fn.decompose()

            # Get all p tag
            paragraphs = article.findAll('p')
            self.articles[index] = "\n\n[SECTION]\n\n".join([paragraph.get_text().strip() for paragraph in paragraphs])
```

Saved successfully!

```
!cat pdf2text.py

    def pdf2text(filename):
        from tika import parser
        raw = parser.from_file(filename + '.pdf')

        f = open(filename + '.txt', 'w+')
        f.write(raw['content'].strip())
        f.close

    import sys
    pg_name = sys.argv[0]
    args = sys.argv[1:]

    if len(sys.argv) != 2:
        print("Usage: ", pg_name)
        print("Please specify One argument")
        sys.exit(1)

    pdf2text(args[0])
```

Import Packages

```
from fomc_get_data.FomcStatement import FomcStatement
from fomc_get_data.FomcMinutes import FomcMinutes
from fomc_get_data.FomcMeetingScript import FomcMeetingScript
from fomc_get_data.FomcPresConfScript import FomcPresConfScript
from fomc_get_data.FomcSpeech import FomcSpeech
from fomc_get_data.FomcTestimony import FomcTestimony
```

Download text data

Saved successfully!

```
# Inspect script:
!cat FomcGetData.py
```

```
        print( writing to  , filepath)
        df.to_csv(filepath, index=index_csv)
        print('Successfully saved {}.csv in {}'.format(file_name, filepath))

    if __name__ == '__main__':
        pg_name = sys.argv[0]
        args = sys.argv[1:]
        content_type_all = ('statement', 'minutes', 'meeting_script', 'presconf_script', 'speech', 'testimony', 'all')

        if (len(args) != 1) and (len(args) != 2):
            print("Usage: ", pg_name)
            print("Please specify the first argument from ", content_type_all)
            print("You can add from_year (yyyy) as the second argument.")
            print("\n You specified: ", ','.join(args))
            sys.exit(1)

        if len(args) == 1:
            from_year = 1990
        else:
            from_year = int(args[1])

        content_type = args[0].lower()
        if content_type not in content_type_all:
            print("Usage: ", pg_name)
            print("Please specify the first argument from ", content_type_all)
            sys.exit(1)

        if (from_year < 1980) or (from_year > 2020):
            print("Usage: ", pg_name)
            print("Please specify the second argument between 1980 and 2020")
```

```
            sys.exit(1)

        if content_type == 'all':
            fomc = FomcStatement()
            download_data(fomc, from_year)
            fomc = FomcMinutes()
            download_data(fomc, from_year)
            fomc = FomcMeetingScript()
            download_data(fomc, from_year)
            fomc = FomcPresConfScript()
            download_data(fomc, from_year)
            fomc = FomcSpeech()
            download_data(fomc, from_year)
            fomc = FomcTestimony()
            download_data(fomc, from_year)
        else:
                                    tement':
                                    t()
                                    inutes':
                fomc = FomcMinutes()
            elif content_type == 'meeting_script':
                fomc = FomcMeetingScript()
            elif content_type == 'presconf_script':
                fomc = FomcPresConfScript()
            elif content_type == 'speech':
                fomc = FomcSpeech()
            elif content_type == 'testimony':
                fomc = FomcTestimony()

            download_data(fomc, from_year)
```

Saved successfully!

Execution arguments

```
# Execute Script:
!python FomcGetData.py all 1980
#!python FomcGetData.py all 1980 > FomcGetData_debug.txt # Save output for debugging
```

```
Getting links for statements...
45 links found in the current page.
Getting links from archive pages...
YEAR: 1980 - 0 links found.
YEAR: 1981 - 0 links found.
YEAR: 1982 - 0 links found.
YEAR: 1983 - 0 links found.
YEAR: 1984 - 0 links found.
YEAR: 1985 - 0 links found.
YEAR: 1986 - 0 links found.
YEAR: 1987 - 0 links found.
YEAR: 1988 - 0 links found.
YEAR: 1989 - 0 links found.
```

```
YEAR: 1990 - 0 links found.
YEAR: 1991 - 0 links found.
YEAR: 1992 - 0 links found.
YEAR: 1993 - 0 links found.
YEAR: 1994 - 6 links found.
YEAR: 1995 - 3 links found.
YEAR: 1996 - 1 links found.
YEAR: 1997 - 1 links found.
YEAR: 1998 - 3 links found.
YEAR: 1999 - 6 links found.
YEAR: 2000 - 8 links found.
YEAR: 2001 - 11 links found.
YEAR: 2002 - 8 links found.
YEAR: 2003 - 8 links found.
YEAR: 2004 - 8 links found.
YEAR: 2005 - 8 links found.
```

Saved successfully!

```
YEAR: 2009 - 8 links found.
YEAR: 2010 - 9 links found.
YEAR: 2011 - 8 links found.
YEAR: 2012 - 8 links found.
YEAR: 2013 - 8 links found.
YEAR: 2014 - 8 links found.
There are total  194  links for  statement
Getting articles - Multi-threaded...
.....................................................................................................................................
  File "FomcGetData.py", line 81, in <module>
    download_data(fomc, from_year)
  File "FomcGetData.py", line 14, in download_data
    df = fomc.get_contents(from_year)
  File "/content/drive/My Drive/Colab Notebooks/proj2/src/fomc_get_data/FomcBase.py", line 128, in get_contents
    self._get_articles_multi_threaded()
  File "/content/drive/My Drive/Colab Notebooks/proj2/src/fomc_get_data/FomcBase.py", line 115, in _get_articles_multi_threaded
    t.join()
  File "/usr/lib/python3.6/threading.py", line 1056, in join
    self._wait_for_tstate_lock()
  File "/usr/lib/python3.6/threading.py", line 1072, in _wait_for_tstate_lock
    elif lock.acquire(block, timeout):
KeyboardInterrupt
```

FOMCGetCalendar Testing

Packages

Inspect Script

```
# Inspect script:
!cat FomcGetCalendar.py
```

```
                if "*" in date_text:
                    date_text = date_text.replace("*", "").strip()
                    is_forecast = True

                if "/" in month_name:
                    month_name = re.findall(r".+/(.+)$", month_name)[0]
                    is_month_short = True

                if "-" in date_text:
                    findall(r".+-(.+)$", date_text)[0]

                meeting_date_str = m_year + "-" + month_name + "-" + date_text
                if is_month_short:
                    meeting_date = datetime.strptime(meeting_date_str, '%Y-%b-%d')
                else:
                    meeting_date = datetime.strptime(meeting_date_str, '%Y-%B-%d')

                date_list.append({"date": meeting_date, "unscheduled": is_unscheduled, "forecast": is_forecast, "confcall": False})

        # Retrieve FOMC Meeting date older than 2015
        for year in range(from_year, 2015):
            hist_url = base_url + '/monetarypolicy/fomchistorical' + str(year) + '.htm'
            r = requests.get(hist_url)
            soup = BeautifulSoup(r.text, 'html.parser')
            if year in (2011, 2012, 2013, 2014):
                panel_headings = soup.find_all('h5', {"class": "panel-heading"})
            else:
                panel_headings = soup.find_all('div', {"class": "panel-heading"})
            print("YEAR: {} - {} meetings found.".format(year, len(panel_headings)))
            for panel_heading in panel_headings:
                date_text = panel_heading.get_text().strip()
                #print("Date: ", date_text)
                regex = r"(January|February|March|April|May|June|July|August|September|October|November|December).*\s(\d*-)*(\d+)\s+(Meeting|Confe
                date_text_ext = re.findall(regex, date_text)[0]
                meeting_date_str = date_text_ext[4] + "-" + date_text_ext[0] + "-" + date_text_ext[2]
                #print("   Extracted:", meeting_date_str)
                if meeting_date_str == '1992-June-1':
                    meeting_date_str = '1992-July-1'
                elif meeting_date_str == '1995-January-1':
                    meeting_date_str = '1995-February-1'
                elif meeting_date_str == '1998-June-1':
                    meeting_date_str = '1998-July-1'
                elif meeting_date_str == '2012-July-1':
                    meeting_date_str = '2012-August-1'
                elif meeting_date_str == '2013-April-1':
                    meeting_date_str = '2013-May-1'
```

Saved successfully!

```
                meeting_date = datetime.strptime(meeting_date_str, '%Y-%B-%d')
                is_confcall = "Conference Call" in date_text_ext[3]
                is_unscheduled = "unscheduled" in date_text_ext[3]
                date_list.append({"date": meeting_date, "unscheduled": is_unscheduled, "forecast": False, "confcall": is_confcall})

    df = pd.DataFrame(date_list).sort_values(by=['date'])
    df.reset_index(drop=True, inplace=True)
    print(df)

    # Save
    dump_df(df, 'fomc_calendar')
    save_data(df, 'fomc_calendar')
```

Saved successfully!

Execution arguments

```
# Execute Script:
!python FomcGetCalendar.py 1980
#!python FomcGetCalendar.py 1980 > FomcGetCalendar_debug.txt # Save output for debugging
```

```
    YEAR: 2018 - 8 meetings found.
    YEAR: 2017 - 8 meetings found.
    YEAR: 2016 - 8 meetings found.
    YEAR: 1980 - 21 meetings found.
    YEAR: 1981 - 14 meetings found.
    YEAR: 1982 - 11 meetings found.
    YEAR: 1983 - 14 meetings found.
    YEAR: 1984 - 9 meetings found.
    YEAR: 1985 - 10 meetings found.
    YEAR: 1986 - 9 meetings found.
    YEAR: 1987 - 13 meetings found.
    YEAR: 1988 - 12 meetings found.
    YEAR: 1989 - 14 meetings found.
    YEAR: 1990 - 12 meetings found.
    YEAR: 1991 - 19 meetings found.
    YEAR: 1992 - 12 meetings found.
    YEAR: 1993 - 16 meetings found.
    YEAR: 1994 - 13 meetings found.
    YEAR: 1995 - 11 meetings found.
    YEAR: 1996 - 8 meetings found.
    YEAR: 1997 - 8 meetings found.

    YEAR: 1998 - 10 meetings found.
    YEAR: 1999 - 8 meetings found.
    YEAR: 2000 - 8 meetings found.
```

```
YEAR: 2001 - 13 meetings found.
YEAR: 2002 - 8 meetings found.
YEAR: 2003 - 13 meetings found.
YEAR: 2004 - 8 meetings found.
YEAR: 2005 - 8 meetings found.
YEAR: 2006 - 8 meetings found.
YEAR: 2007 - 11 meetings found.
YEAR: 2008 - 14 meetings found.
YEAR: 2009 - 11 meetings found.
YEAR: 2010 - 10 meetings found.
YEAR: 2011 - 10 meetings found.
YEAR: 2012 - 8 meetings found.
YEAR: 2013 - 9 meetings found.
YEAR: 2014 - 9 meetings found.
          date  ...  confcall
0   1980-01-09  ...     False
```

Saved successfully!

```
4   1980-03-18  ...     False
..         ...  ...       ...
441 2021-06-16  ...     False
442 2021-07-28  ...     False
443 2021-09-22  ...     False
444 2021-11-03  ...     False
445 2021-12-15  ...     False

[446 rows x 4 columns]

Writing to  /content/drive/My Drive/Colab Notebooks/proj2/src/data/FOMC/fomc_calendar.pickle
Successfully saved fomc_calendar.pickle in /content/drive/My Drive/Colab Notebooks/proj2/src/data/FOMC/fomc_calendar.pickle
Writing to  /content/drive/My Drive/Colab Notebooks/proj2/src/data/FOMC/fomc_calendar.csv
Successfully saved fomc_calendar.csv in /content/drive/My Drive/Colab Notebooks/proj2/src/data/FOMC/fomc_calendar.csv
Successfully saved fomc_calendar.pickle in /content/drive/My Drive/Colab Notebooks/proj2/src/data/FOMC/fomc_calendar.pickle
Successfully saved fomc_calendar.csv in /content/drive/My Drive/Colab Notebooks/proj2/src/data/FOMC/fomc_calendar.csv
```

Saved successfully!