

# Eve-GPT

## Group 6

Yang Fan (Blithe)	3035771646
Li Xinran (Anna)	3035767437
Zhu Jiarui (Janet)	3035638791
Fung Ho Kit (Justin)	3035779105





# Table of **contents**



01

**Web Scrapping**

02

**Data Processing**

03

**Sentiment analysis**

04

**EveGPT**



# Step 1: Web Scraping

```

url = f'https://www.wsj.com/'
browser.get(url)
sleep(1)
soup = BeautifulSoup(browser
if p == 1:
    pagenum = soup.find('spc
    pageNum = re.search(r"lc
    print('总页数: ' + str(p
div_list = soup.find_all('di
print(div_list)
for div in div_list:
    title = div.find('span')
    link = div.find('a').get
    try:
        date = div.find('p',
    except:
        date = ''
    try:
        summary = div.find('
    except:
        summary = ''
    dic = {
        'title': title,
        'date': date,
        'summary': summary,
        'detailLink': detail
        'authors': authors,
    }
    print(dic)
    resL.append(dic)
df = pd.DataFrame(resL)
df.to_csv('bloomberg.csv', i
print('已爬' + str(p) + '页')

```

```

main():
keyword = input("Please Input Your Keyword: ")
endindex = 0
while 1:
    url = f'https://api.queryly.com/cnbc/json.aspx?queryly_key=31a35d40a9a64ab3&query={keyword}&end_t_number'
    res = requests.get(url=url, headers=headers).json()
    data = res['results']
    for item in data:
        title = item['cn:title']
        date = item['_pubDate']
        summary = item['description']
        link = item['url']
        section = item['section']
        resp = requests.get(url=link, headers=headers).text
        soup = BeautifulSoup(resp, 'Lxml')
        try:
            result=''
            first_group = soup.select_one('.ArticleBody-articleBody > div.group')
            text = [first_group] + first_group.find_next_siblings()
            for elem in text:
                result += elem.text.strip()
            except:
                result = ''
            # print(resp)
            if soup.find('div', class_='RenderKeyPoints-List') is None:
                summary_ = summary
            else:
                content_li = soup.find('div', class_='group').find('ul').find_all('li')
                content = ''
                for li in content_li:
                    content += li.text
                summary_ = content

```

file:

::]

time("%Y%m%d")

me("%Y%m%d")

.parser")

# The Wall Street Journal

```

import sys
import time
import requests
import json
import re
import datetime
import pandas as pd
from bs4 import BeautifulSoup
from urllib.parse import urljoin

def get_html(url):
    browser = requests.Session()
    browser.get(url)
    sleep(1)
    soup = BeautifulSoup(browser.page_source, 'lxml')
    if p == 1:
        pagenum = soup.find('span', class_='WSJTheme--total-pages--3FkCtMxZ').text
        pageNum = re.search(r"\d+", pagenum).group()
        print('总页数: ' + str(pageNum))
    div_list = soup.find_all('div', class_='WSJTheme--search-text-combined--29JN8aap')
    print(div_list)
    for div in div_list:
        title = div.find('span').text
        link = div.find('a').get('href')
        try:
            date = div.find('p', class_='WSJTheme--timestamp--22sfkNDv').text
        except:
            date = ''
        try:
            summary = div.find('span', class_='WSJTheme--summaryText--2LRaCWgJ').text
        except:
            summary = ''
        dic = {
            'title': title,
            'date': date,
            'summary': summary,
            'link': link,
            'summary': summary
        }
        resL.append(dic)
    resL.append(dic)
    df = pd.DataFrame(resL)
    writer = pd.ExcelWriter(f'{keyword}_wsj.xlsx')
    df.to_excel(writer, index=False, encoding='utf-8')

if __name__ == '__main__':
    keyword = sys.argv[1]
    url = f'https://www.wsj.com/search?query={keyword}&mod=searchresults_viewallresults&page={str(p),
    browser.get(url)
    sleep(1)
    soup = BeautifulSoup(browser.page_source, 'lxml')
    if p == 1:
        pagenum = soup.find('span', class_='WSJTheme--total-pages--3FkCtMxZ').text
        pageNum = re.search(r"\d+", pagenum).group()
        print('总页数: ' + str(pageNum))
    div_list = soup.find_all('div', class_='WSJTheme--search-text-combined--29JN8aap')
    print(div_list)
    for div in div_list:
        title = div.find('span').text
        link = div.find('a').get('href')
        try:
            date = div.find('p', class_='WSJTheme--timestamp--22sfkNDv').text
        except:
            date = ''
        try:
            summary = div.find('span', class_='WSJTheme--summaryText--2LRaCWgJ').text
        except:
            summary = ''
        dic = {
            'title': title,
            'date': date,
            'summary': summary,
            'link': link,
            'summary': summary
        }
        resL.append(dic)
    resL.append(dic)
    df = pd.DataFrame(resL)
    writer = pd.ExcelWriter(f'{keyword}_wsj.xlsx')
    df.to_excel(writer, index=False, encoding='utf-8')

# 爬取华尔街日报新闻
start_number = 10000
url = f'https://news.set
time.sleep(3)
response = requests.get(
soup = BeautifulSoup(res
with open('yahoo_search
    file.write(str(soup)
for article in soup.find
    title = article.find
    raw_link = article.f
    unquoted_link = requ
    pattern = re.compile
    link = re.search(pat
    link = article.find(
    desc = article.find(
    relative_time = arti
    if 'hours' in relatit
        hours = int(relat
        date = (datetime
    else:
        days = int(relat
        date = (datetime
    source = article.fir
    content_response = r
    content_soup = Beaut
    print(content_soup)
    content = content_sc
    articles.append([tit
        print(dic)
        resL.append(dic)
    df = pd.DataFrame(resL)
    df.to_csv('bloomberg.csv')
    print('已爬' + str(p) + '

```



# Bloomberg

```

def main():
    keyword = input("Please Input keyword: ")
    endIndex = 0
    while 1:
        url = f'https://api.query'
        res = requests.get(url=url)
        data = res['results']
        for item in data:
            title = item['content']
            date = item['pubDate']
            summary = item['desc']
            link = item['url']
            section = item['sect']
            resp = requests.get(url=link)
            soup = BeautifulSoup(resp.text)
            try:
                results = ''
                first_group = soup.find('div', {'class': 'first_group'})
                for elem in first_group.find_all('div'):
                    result += elem.get_text()
            except:
                result = ''
            # print(result)
            if soup.find('div', {'class': 'summary'}) is not None:
                summary = soup.find('div', {'class': 'summary'}).get_text()
            else:
                content_li = soup.find('div', {'class': 'content'})
                for li in content_li.find_all('li'):
                    content += li.get_text()
                summary = content

```

```

while (start_number <= 10):
    url = f'https://news.bloom'
    time.sleep(3)
    response = requests.get(url=url)
    soup = BeautifulSoup(response.text)
    with open("yahoo_search_results.txt", 'a') as f:
        for article in soup.find_all('div', {'class': 'article'}):
            title = article.find('h2').get_text()
            raw_link = article.find('a').get_text()
            unquoted_link = re.sub(r'"', '', raw_link)
            pattern = re.compile(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$#@]|_|[\u4e00-\u9fa5])+')
            link = re.search(pattern, unquoted_link)
            desc = article.find('p').get_text()
            relative_time = article.find('span', {'class': 'relative_time'}).get_text()
            if 'hour' in relative_time:
                hours = int(relative_time.split('hour')[0])
                date = (datetime.datetime.now() - timedelta(hours=hours)).strftime('%Y-%m-%d %H:%M:%S')
            else:
                days = int(relative_time.split('days')[0])
                date = (datetime.datetime.now() - timedelta(days=days)).strftime('%Y-%m-%d %H:%M:%S')
            source = article.find('p', {'class': 'source'}).get_text()
            content_response = article.find('div', {'class': 'content_response'}).get_text()
            content_soup = BeautifulSoup(content_response)
            print(content_soup.get_text())
            content = content_soup.get_text()
            articles.append([title, date, source, content])

```

```

startPage = int(input("请输入起始页数: "))
url = f'https://www.bloomberg.com/markets2/api/search?query=amazon&page={str(startPage)}&sort=time:desc'
res = requests.get(url=url, headers=headers, proxies=proxies).text
data = json.loads(res)
print(data)
totalPage = int(data['total'] / 10)
for p in range(startPage, totalPage + 1):
    url = f'https://www.bloomberg.com/markets2/api/search?query=amazon&page={str(p)}&sort=time:desc'
    res = requests.get(url=url, headers=headers, proxies=proxies).text
    data = json.loads(res)
    print(data)
    for item in data['results']:
        title = item['headline']
        date = item['publishedAt']
        summary = item['summary']
        detailLink = item['url']
        authors = item['authors']
        dic = {
            'title': title,
            'date': date,
            'summary': summary,
            'detailLink': detailLink,
            'authors': authors,
        }
        print(dic)
        resL.append(dic)
df = pd.DataFrame(resL)
df.to_csv('bloomberg.csv', index=False, encoding='utf-8')
print('已爬' + str(p) + '页')

```

'results'&page={str(p)}

xt

80ap')

.text

# Yahoo Finance

```
url = f'https://www.wsj.com/seo'
'douser.get(url)
sleep(3)
soup = BeautifulSoup(browser.pa
if p == 1:
    pagenum = soup.find('span',
    pagelium = re.search(r"ld+",
    print('dlldl: ' + str(pageh
    div_list = soup.find_all('div',
    print(div_list)
for div in div_list:
    title = div.find('span').te
    link = div.find('a').get('h
    try:
        date = div.find('p', cl
    except:
        date = ''
    try:
        summary = div.find('spa
    except:
        summary = ''
    dic = {
        'title': title,
        'date': date,
        'link': link,
        'summary': summary
    }
    resL.append(dic)
df = pd.DataFrame(resL)
writer = pd.ExcelWriter(f'keyw
    excel(writer, index=False
```

```
main():
keyword = input("Please I
endindex = 0
while 1:
    url = f'https://api.q
    res = requests.get(ur
    data = res['results']
    for item in data:
        title = item['_cn:
        date = item['_pub
        summary = item['_d
        link = item['_url'
        section = item['_s
        resp = requests.g
        soup = Beautifuls
        try:
            result = ''
            first_group =
            text = [first
            for elem in t
                result +=
        except:
            result = ''
        # print(resp)
        if soup.find('div
            summary_ = su
        else:
            content_li =
            content = ''
            for li in con
                content +=
            summary_ = con
```

```
le (start_number <= 10000):
url = f"https://news.search.yahoo.com/search?p={keyword}&b={start_number}"
time.sleep(3)
response = requests.get(url, headers = headers)
soup = BeautifulSoup(response.content, "html.parser")
with open("yahoo_search_results.txt", "w", encoding="utf-8") as file:
    file.write(str(soup))
for article in soup.find_all("div", class_="NewsArticle"):
    title = article.find("h4", class_="s-title").text.strip()
    raw_link = article.find('a').get('href')
    unquoted_link = requests.utils.unquote(raw_link)
    pattern = re.compile(r'RU=(.+)\|RK')
    link = re.search(pattern, unquoted_link).group(1)
    link = article.find('a').get('href')
    desc = article.find('p', 's-desc').text.strip()
    relative_time = article.find('span', 's-time').text.strip()[2:]
    if 'hour' in relative_time:
        hours = int(relative_time.split()[0])
        date = (datetime.utcnow() - timedelta(hours=hours)).strftime("%Y%m%d")
    else:
        days = int(relative_time.split()[0])
        date = (datetime.utcnow() - timedelta(days=days)).strftime("%Y%m%d")
    source = article.find('span', 's-source').text
    content_response = requests.get(link, headers=headers)
    content_soup = BeautifulSoup(content_response.content, "html.parser")
    print(content_soup)
    content = content_soup.select_one(".caas-body").get_text()
    articles.append([title, source, link, desc, date])
```

artPage]}&sort=time:desc

r(p)}&sort=time:desc'



# Twitter—Twint



726232111 commented on Mar 10 • edited ▾

1. install twint (need install git) , don't use pip install twint command.

```
pip3 install --user --upgrade
```

```
git+https://github.com/twintproject/twint.git@origin/master#egg=twint
```

2. get file path

```
python.exe -c "import twint;print(twint.__file__);"
```

3. replace token.py file

<https://gist.github.com/moxak/ed83dd4169112a0b1669500fe855101a>



2

## #1433

This repository has been archived by the owner on Mar 30, 2023. It is now read-only.

# Twitter—Selenium

```

while driver.execute_script(js1) > old_scroll_height: # compare the height
    div_list = driver.find_elements(By.XPATH,
                                     '//*[@id="react-root"]/div/div/div[2]/main/div/div/div/div/div[1]')
    for div in div_list:
        try:
            content = div.find_element(By.XPATH, '//*[@data-testid="tweetText"]')
        except Exception as e:
            print(e)
            continue
        if content.get_attribute('id') in content_l:
            continue
        cnt += 1
        print(content.text)
        try:
            time_element = div.find_element(By.XPATH, '//*[@time]').text
        except Exception as e:
            print(e)
            continue
        link_div = div.find_element(By.XPATH, '//*[@class="css-1dbjc4n r-18u37iz r-1q142lx"]')
        link_element = link_div.find_element(By.XPATH, '//*[@a]').get_attribute('href')
        content_l.append(content.get_attribute('id'))
        dic = {
            'Content': content.text,
            'Time': time_element,
            'Link': link_element
        }
        print(dic)
        resL.append(dic)
        df = pd.DataFrame(resL)
        writer = pd.ExcelWriter(f'{key}_twi.xlsx')
        df.to_excel(writer, index=False, encoding='utf-8')
        writer.save()
    old_scroll_height = driver.execute_script(js1) # get current height
    driver.execute_script(js2) # pull the weber
    for j in range(5):
        actions.send_keys(Keys.PAGE_DOWN).perform()
        time.sleep(1)
    time.sleep(4) #leave load time
    if (driver.execute_script(js1) <= old_scroll_height): # deal with the crush of web pages
        cnt = 0
        winsound.Beep(3000, 3000) #sound alert
        input('Enter:')
        old_scroll_height = 0 #
        js1 = 'return document.body.scrollHeight'

```

# Step 2: Data Processing



## — Text Data: Overview

	Bloomberg	CNBC	WSJ	Yahoo
Apple	10000 rows link, date, title, summary, author	7400 rows section, date time, title, summary...	1883 rows link, date time, title, summary	1000 rows link, source, date, title, description
Google	9890 rows link, date, title, summary, author	3080 rows section, date time, title, summary...	2384 rows link, date time, title, summary	1000 rows link, source, date, title, description
:	:	:	:	:
Nvidia	2980 rows link, date, title, summary, author	2090 rows section, date time, title, summary...	147 rows link, date time, title, summary	1000 rows link, source, date, title, description

## — Text Data: What Our Model Need

FinBERT

- supports many common English punctuations
- can split a paragraph into sentences

date	platform	title	summary
2023-04-17	bloomberg	text	text
⋮	⋮	⋮	⋮
2020-07-04	wsj	text	text



## — Text Data: Ready for Model Input

	A	B	C	D	E	F
1	date	platform	title	summary		
2	2023-04-17	bloomberg	Transcript: Sc	Over time, we expect the world to get r		
3	2023-04-17	bloomberg	Wells Fargo F	Wells Fargo & Co. leaders are privately		
4	2023-04-17	bloomberg	Apple's Buyb	One of the most important numbers in		

## — Stock Data: Where We Get It

```
>>> import yfinance as yf
```

```
In [229]: #df = yf.download(tickers = 'TSLA', start = '2022-03-11',end = '2023-03-11')
          company = "TSLA"
          Starting = "2020-07-20"
          Ending = "2023-04-14"

          tickers = yf.Tickers(company)
          StockPrice = tickers.tickers[company].history(start=Starting,end=Ending)
          StockPrice.index = pd.to_datetime(StockPrice.index).date
          StockPrice
```

executed in 168ms, finished 22:42:44 2023-04-26

	Open	High	Low	Close	Volume	Dividends	Stock Splits
2020-07-20	101.267334	110.000000	99.199997	109.533333	256821000	0.0	0.0
2020-07-21	109.328667	111.666664	103.866669	104.557335	241608000	0.0	0.0
2020-07-22	106.599998	108.428001	104.133331	106.155334	212416500	0.0	0.0
2020-07-23	111.930000	112.599998	98.718002	100.871330	364927500	0.0	0.0
2020-07-24	94.400665	97.666664	91.102669	94.466667	290949000	0.0	0.0
...	...	...	...	...	...	...	...
2023-04-06	183.080002	186.389999	179.740005	185.059998	123857900	0.0	0.0

## — Stock Data: Calculate Technical Analysis Indicators

```
>>> import pandas_ta as ta
```

```
In [28]: Full_data['RSI']=ta.rsi(Full_data.Close, length=30)
         Full_data['EMA']=ta.ema(Full_data.Close, length=30)
```

```
In [29]: data_set = Full_data.iloc[30:-1, 6:14]
         data_set
```

executed in 14ms, finished 03:39:03 2023-04-27

	score_title	score_summary	RSI	EMA	TargetNextClose
2020-08-31	0.064944	0.178783	78.587101	115.660970	158.350006
2020-09-01	0.139595	0.208545	72.223916	118.415101	149.123337
2020-09-02	-0.175306	0.031887	65.679839	120.396277	135.666672
2020-09-03	-0.072719	-0.187810	57.780923	121.381464	139.440002
2020-09-04	-0.066646	-0.387875	59.204111	122.546531	110.070000
...	...	...	...	...	...
2023-04-05	0.159216	0.101471	50.021287	189.355496	185.059998
2023-04-06	-0.080997	0.019024	49.882707	189.078367	184.509995
2023-04-10	-0.066649	-0.001051	49.712360	188.783633	186.789993
2023-04-11	-0.112037	-0.080313	50.438172	188.655011	180.539993
2023-04-12	0.051657	0.028311	48.454957	188.131462	185.899994

658 rows x 5 columns

# Step 3: Sentiment Analysis

## FinBERT

### FinBERT: Financial Sentiment Analysis with Pre-trained Language Models

- Araci, D. (2019).
- Improve BERT with further pretraining on
  - TRC2-financial.
  - Financial PhraseBank.\*

(lr)2-4 (lr)5-7 Model	All data			Data with 100% agreement		
	Loss	Accuracy	F1 Score	Loss	Accuracy	F1 Score
LSTM	0.81	0.71	0.64	0.57	0.81	0.74
LSTM with ELMo	0.72	0.75	0.7	0.50	0.84	0.77
ULMFit	0.41	0.83	0.79	0.20	0.93	0.91
LPS	-	0.71	0.71	-	0.79	0.80
HSC	-	0.71	0.76	-	0.83	0.86
FinSSLX	-	-	-	-	0.91	0.88
FinBERT	<b>0.37</b>	<b>0.86</b>	<b>0.84</b>	<b>0.13</b>	<b>0.97</b>	<b>0.95</b>



## FinBERT

### FinBERT: Financial Sentiment Analysis with Pre-trained Language Models

- Araci, D. (2019).
- Improve BERT with further pretraining on
  - TRC2-financial.
  - Financial PhraseBank.\*

(lr)2-4 (lr)5-7 Model	All data			Data with 100% agreement		
	Loss	Accuracy	F1 Score	Loss	Accuracy	F1 Score
LSTM	0.81	0.71	0.64	0.57	0.81	0.74
LSTM with ELMo	0.72	0.75	0.7	0.50	0.84	0.77
ULMFit	0.41	0.83	0.79	0.20	0.93	0.91
LPS	-	0.71	0.71	-	0.79	0.80
HSC	-	0.71	0.76	-	0.83	0.86
FinSSLX	-	-	-	-	0.91	0.88
FinBERT	<b>0.37</b>	<b>0.86</b>	<b>0.84</b>	<b>0.13</b>	<b>0.97</b>	<b>0.95</b>

### FinBERT: Financial Sentiment Analysis with BERT

- Prosus AI Team

## Code base

📁 **ProsusAI / finBERT** Public

Financial Sentiment Analysis with BERT

📄 Apache-2.0 license

⭐ 998 stars 🍴 336 forks

★ Starred Watch

<> Code Issues 3 Pull requests 1

## Pre-trained parameters

😊 Search models, datasets, users...

📁 **ProsusAI / finbert** like 204

Text Classification PyTorch TensorFlow JAX

Transformers English bert financial-sentiment-analysis

sentiment-analysis arxiv:1908.10063

## Model import API

### 😊 Transformers

State-of-the-art Machine Learning for [PyTorch](#), [TensorFlow](#), and [JAX](#).

😊 Transformers provides APIs and tools to easily download and train state-of-the-art pretrained models. Using pretrained models can reduce your compute costs, carbon footprint, and save you the time and resources required to train a model from scratch. These models support common tasks in different modalities, such as:

📄 **Natural Language Processing:** text classification, named entity recognition, question answering, language modeling, summarization, translation, multiple choice, and text generation.

🖼️ **Computer Vision:** image classification, object detection, and segmentation.

🔊 **Audio:** automatic speech recognition and audio classification.

📄 **Multimodal:** table question answering, optical character recognition, information extraction from scanned documents, video classification, and visual question answering.

## Demo

⚙️ Text Classification Examples ▾

Stocks rallied and the British pound gained.

Compute

Computation time on Intel Xeon 3rd Gen Scalable cpu: cached

positive	0.898
neutral	0.067
negative	0.034

</> JSON Output 🔍 Maximize

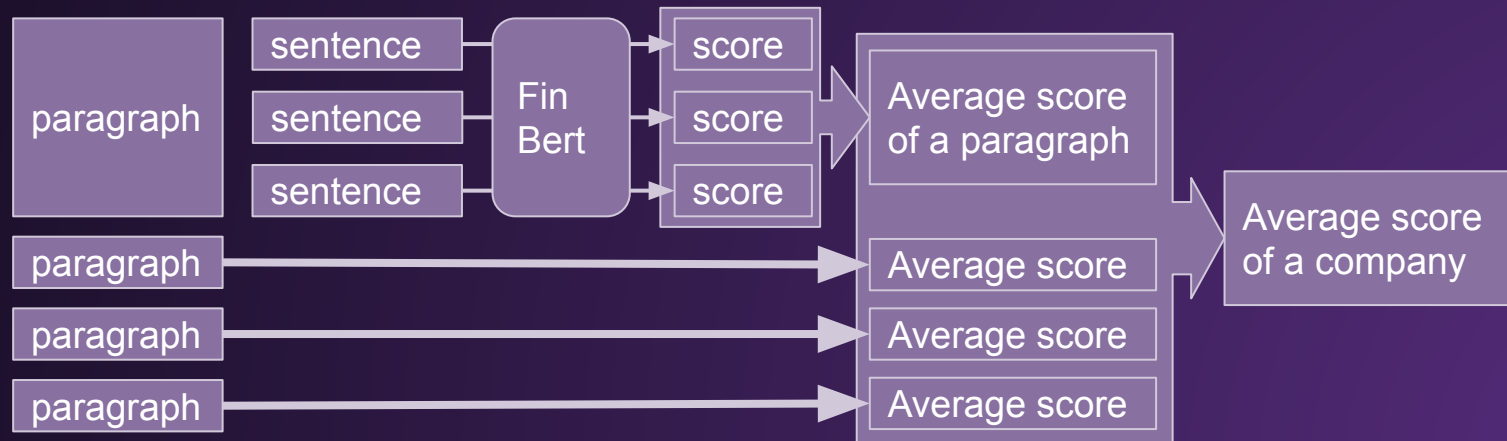
## — **Changes and Innovations**

- Add text validity checking before process

## — Changes and Innovations

- Add text validity checking before process
- One sentiment score per sentence

⇒ Average score of the company per day



## — Changes and Innovations

- Add text validity checking before process
- One sentiment score per sentence
  - ⇒ Average score of the company per day
- Output baseline scores for dates without textual information
  - ⇒ Ensure data have consecutive dates for time series analysis in next step



## — Processing Results

- 12000 records / hour with HKU GPU farm
- Daily news semitation analysis score for:  
6 companies:
  - Amazon, Apple, Google, Microsoft, Nvidia, Tesla
- 1000+ days:
  - 2020-07-04 ~ 2023-04-17



# Step 4: EveGPT

# Workflow

News  
articles  
Blogs  
Tweets

Scrape  
(Part 1)

Process  
(Part 2)

Finbert  
(Part 3)

Stock  
Price

Date	2023.03.11	2023.03.12	...	2023.03.24
Classification result Title_Score Summary_Score	0.1	-0.3	...	???
Indicators: RSI EMA	62.0372 206.845 3	50.5564 209.2611	...	

EveGPT

Group 6: Yang Fan (Blithe) Li Xinran (Anna) Zhu Jiarui (Janet) Fung Ho Kit (Justin)  
Instructor: Dr. Matthias Buehlmaier

Stock price prediction  
(Part 4)

# Task Construction

```
Full_data['TargetNextClose'] = Full_data['Close'].shift(-1)  
Full_data['TargetNextClose'] = Full_data['Close'].shift(-1)
```

Input Variables

	score_title	score_summary	RSI	EMA	TargetNextClose
2020-08-31	0.568099	0.671994	0.872132	0.000000	0.166462
2020-09-01	0.611594	0.690165	0.763835	0.011589	0.135897
2020-09-02	0.428118	0.582304	0.652460	0.019925	0.091320
2020-09-03	0.487890	0.448165	0.518026	0.024070	0.103820
2020-09-04	0.491428	0.326013	0.542247	0.028972	0.006526
...	...	...	...	...	...
2023-04-05	0.623026	0.624790	0.385962	0.310083	0.254944
2023-04-06	0.483067	0.574451	0.383604	0.308917	0.253122
2023-04-10	0.491427	0.562193	0.380705	0.307677	0.260675
2023-04-11	0.464982	0.513799	0.393057	0.307136	0.239971
2023-04-12	0.560357	0.580121	0.359304	0.304933	0.257727

Output Target

658 rows × 5 columns

# Data Preparation

```
X = []  
backcandles = 10 # Look back period  
  
print(data_set_scaled.shape[0])  
for j in range(4):  
    X.append([])  
    for i in range(backcandles, data_set_scaled.shape[0]):  
        X[j].append(data_set_scaled[i-backcandles:i, j])  
  
#move axis from 0 to position 2  
X=np.moveaxis(X, [0], [2])  
  
X, yi =np.array(X), np.array(data_set_scaled[backcandles:,-1])  
y=np.reshape(yi,(len(yi),1))  
  
print(X.shape) #658 data, 10 days loop back, 6 indicators  
print(y.shape) #Start from 10 days since need 10 to process data
```

658

Shape of X: (648, 10, 4)

Shape of y: (648, 1)

## Look back method

Considering past 10 days data to  
do the prediction

**648 Data Volume**  
**10 days lookahead**  
**4 Indicators**



# Model Training

```
In [48]: indicators = 4
lstm_input = Input(shape=(backcandles, indicators), name='lstm_input')
inputs = LSTM(150, name='first_layer')(lstm_input)
inputs = Dense(1, name='dense_layer')(inputs)
output = Activation('linear', name='output')(inputs)
model = Model(inputs=lstm_input, outputs=output)
adam = optimizers.Adam()
model.compile(optimizer=adam, loss='mse')
history = model.fit(x=X_train, y=y_train, batch_size=15, epochs=200, shuffle=True)
history
```

Model Setting

executed in 41.2s, finished 03:43:34 2023-04-27

```
32/32 [=====] - 0s 7ms/step - loss: 0.0025 - val_loss: 0.0026
Epoch 197/200
32/32 [=====] - 0s 7ms/step - loss: 0.0020 - val_loss: 0.0028
Epoch 198/200
32/32 [=====] - 0s 7ms/step - loss: 0.0019 - val_loss: 0.0029
Epoch 199/200
32/32 [=====] - 0s 7ms/step - loss: 0.0019 - val_loss: 0.0026
Epoch 200/200
32/32 [=====] - 0s 8ms/step - loss: 0.0018 - val_loss: 0.0028
```

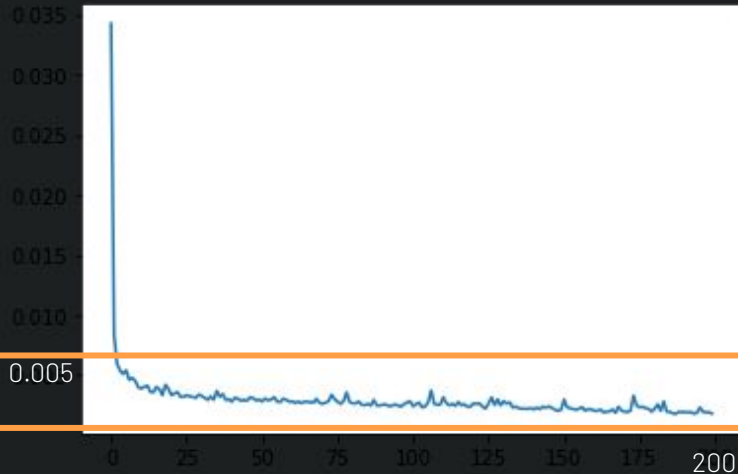
Prevent Overfitting

# Model Evaluation - Loss

```
In [58]: loss_values = history.history['loss']

# print out the loss values for each epoch
for i in range(len(loss_values)):
    print("Epoch:", i + 1, "Loss:", loss_values[i])

plt.plot(loss_values)
```

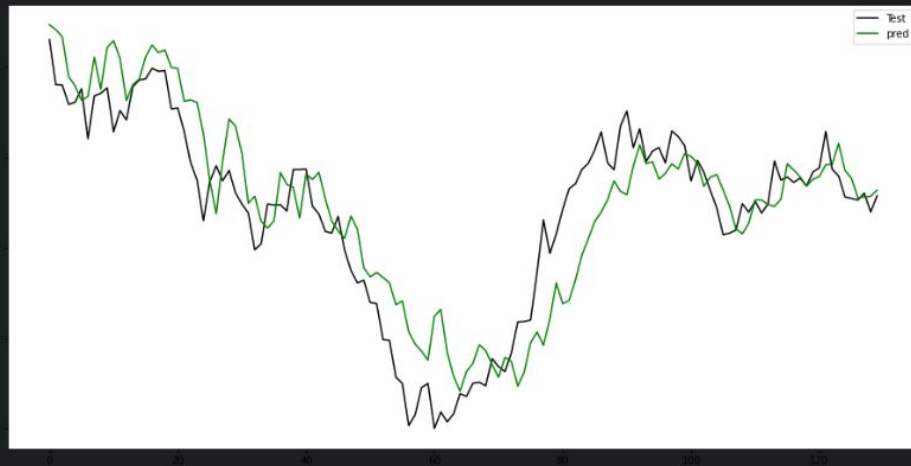


**Loss below 0.005  
and become stable**

# Model Demo & Storage

```
plt.figure(figsize=(16,8))
plt.plot(y_test, color = 'black', label = 'Test')
plt.plot(y_pred, color = 'green', label = 'pred')
plt.legend()
plt.show()
```

executed in 110ms, finished 03:53:35 2023-04-27



```
model.save('EveGPT')
```

```
loaded_model = keras.models.load_model('EveGPT')
loaded_model.summary()
```

```
y_pred = loaded_model.predict(X_test)
for i in range(10):
    print(y_pred[i], y_test[i])
```

executed in 1.14s, finished 03:54:44 2023-04-27

Model: "model"

Layer (type)	Output Shape	Param #
=====		
lstm_input (InputLayer)	[(None, 10, 6)]	0
first_layer (LSTM)	(None, 150)	94200
dense_layer (Dense)	(None, 1)	151
output (Activation)	(None, 1)	0

```
=====
Total params: 94,351
Trainable params: 94,351
Non-trainable params: 0
```

# Web App Design

## WebApp Setting Html & CSS coding

```
#####  
#  
#           Setting  
#  
#####
```

```
def local_css(file_name):  
    with open(file_name) as f:  
        st.markdown(f"<style>{f.read()}</style>", unsafe_allow_html=True)  
st.set_page_config(page_title="EveGPT", page_icon="👽")  
st.markdown("<div style='background-color:#F4D03F;padding:10px;border-radius:10px;'>", unsafe_allow_html=True)  
st.markdown("<h1 style='text-align: center; color: #FF9E44;'>EveGPT</h1>", unsafe_allow_html=True)  
Group = ""  
  
    <div style="background-color:#F4D03F;padding:10px;border-radius:10px;">  
        <div style="color: black;">  
            Group 6:  
            Yang Fan (Blithe)  
            Li Xinran (Anna)  
            Zhu Jiarui (Janet)  
            Fung Ho Kit (Justin)  
            </br>  
            Instructor: Dr. Matthias Buehlmaier  
        </div>  
    </div>  
    ""  
st.markdown(Group, unsafe_allow_html=True)  
local_css("style.css")
```

## Instant Data retrieval

```
#####  
#  
#           Raw Data  
#  
#####  
  
start = "2020-07-20"  
today = date.today().strftime("%Y-%m-%d")  
st.write(f"Today is {today}")  
  
stocks = ["AAPL",  
          "TSLA",  
          "GOOG",  
          "AMZN",  
          "MSFT",  
          "NVDA"]  
  
selected_stocks = st.selectbox("Select stock", stocks)  
#n_years = st.slider("Year of prediction",1,4)  
#period = n_years*365  
  
def load_data(ticker):  
    tickers = yf.Tickers(ticker)  
    data = tickers.tickers[ticker].history(start=start)  
    data.index = pd.to_datetime(data.index).date  
    #data.reset_index(inplace = True)  
    return data  
  
data_load_state = st.text("load data...")  
StockPrice = load_data(selected_stocks)  
data_load_state.text("load data...done!")  
  
st.write("Raw data")  
st.write(StockPrice.tail())  
  
def plot_raw_data(Data):  
    fig = go.Figure()  
    fig.add_trace(go.Scatter(x=Data.index, y= Data["Open"], name = "stock_open"))  
    fig.add_trace(go.Scatter(x=Data.index, y= Data["Close"], name = "stock_close"))  
    fig.layout.update(title_text="Time series data", xaxis_rangeslider_visible=True)  
    st.plotly_chart(fig)  
    plot_raw_data(StockPrice)  
  
st.markdown("<div style='background-color:#F4D03F;padding:10px;border-radius:10px;'>", unsafe_allow_html=True)
```

# Web App Design

## # Loading Scraping Results and Sentiment Scores

```
def load_scraped_text(ticker):
    Path = f"data/{ticker}.xlsx"
    scraped_text = pd.read_excel(Path).iloc[:,1].set_index("date")
    scraped_text.index = pd.to_datetime(scraped_text.index).date
    return scraped_text

def load_sentiment_score(ticker):
    Path = f"../Part3_FinBERT/output/predictions_{ticker}.csv"
    Sentiment = pd.read_csv(Path).iloc[:,1]\
        .set_index("date")\
        .drop(['Unnamed: 0'],axis=1)
    #Sentiment.index = pd.to_datetime(Sentiment["date"]).date
    Sentiment.column = ["Scores for title", "Score for summary"]
    return Sentiment

data_load_state = st.text("load data...")
scraped_text = load_scraped_text(selected_stocks)
Sentiment = load_sentiment_score(selected_stocks)
data_load_state.text("load data...done!")

st.write("Scraped Text")
st.write(scraped_text.tail(10))

st.write("Sentiment score retrieval")
st.write(Sentiment.tail(10))
```

```
#####
#
# Run prediction
#
#####

company = selected_stocks
Starting = "2020-07-20"
Ending = "2023-04-14"

tickers = yf.Tickers(company)
StockPrice = tickers.tickers[company].history(start=Starting, end=Ending)
StockPrice.index = pd.to_datetime(StockPrice.index).date

Path = f"../Part3_FinBERT/output/predictions_{company}.csv"
Sentiment = pd.read_csv(Path).iloc[:,1]\
    .reset_index()\
    .drop(['Unnamed: 0'],axis=1)

Sentiment = Sentiment[Sentiment["date"] >= Starting]
Sentiment["date"] = pd.to_datetime(Sentiment["date"])
Sentiment.set_index("date", inplace=True)
Sentiment.index = pd.to_datetime(Sentiment.index).date

missing_row_index = StockPrice.index.difference(Sentiment.index)
new_data = {'score.title': 0, 'score.summary': 0}
new_row = pd.DataFrame(new_data, index=missing_row_index)
Sentiment = pd.concat([new_row, Sentiment], axis=0, sort=False)
index_to_keep = pd.Series(True, index=StockPrice.index)
mask = Sentiment.index.isin(index_to_keep.index)
Sentiment = Sentiment[mask]
Full_data = pd.concat([StockPrice, Sentiment], axis=1)

Full_data['RSI'] = ta.rsi(Full_data.Close, length=30)
Full_data['EMAP'] = ta.ema(Full_data.Close, length=30)
Full_data['EMAW'] = ta.ewm(Full_data.Close, length=30)
Full_data['EMAS'] = ta.ewm(Full_data.Close, length=30)

Full_data['TargetNextClose'] = Full_data['Close'].shift(-1)
Full_data['TargetNextClose'] = Full_data['Close'].shift(-1)
Full_data.iloc[:, 0:-1].dropna(inplace=True)
Full_data.drop(['Volume'], axis=1, inplace=True)
data_set = Full_data.iloc[10:-1, 0:12]
st.write(data_set)

sc = MinMaxScaler(feature_range=(0,1))
data_set_scaled = sc.fit_transform(data_set)
X = []
backcandles = 10 # Look back period

print(data_set_scaled.shape[0])
for i in range(0):
    X.append([])
    for j in range(backcandles, data_set_scaled.shape[0]):
        X[j].append(data_set_scaled[i-backcandles:i, j])

#move axis from 0 to position 2
X=np.moveaxis(X, [0], [2])
print(X.shape)

loaded_model = keras.models.load_model('EveGPT')
loaded_model.summary()
y_pred = loaded_model.predict(X)

y = Full_data.iloc[40:-1, 12:]
print(y)
sc = MinMaxScaler(feature_range=(0,1))
y_test = sc.fit_transform(y)

def plot_result():
    fig = go.Figure()
    fig.add_trace(go.Scatter(x=y.index, y=y_test.ravel(), name='original', line=dict(color='white')))
    fig.add_trace(go.Scatter(x=y.index, y=y_pred.ravel(), name='prediction', line=dict(color='darkblue')))
    fig.layout.update(title_text='Prediction', axis_rangeslider_visible=True)
    st.plotly_chart(fig)
plot_result()
```

## Loading Pretrained Model For Prediction Of Stock Price

# Web App Demo

**240 lines  
to build the web app**



<https://justinfunji-fina4350-nlp-part4-evegptwebapp-fxtphb.streamlit.app/>



# Thank you!