다음과 네이버의 실시간 검색어 분석

김동영 (Kim Dong Young)

I. 기능 총정리

기능	설명	함수
주기 선택	사용자가 원하는 주기 입력	seasonal
주기에 따른 시간 계산(실시간)	실시간으로 크롤링이 가능하도록 시간 계산 멀티쓰레드의 timer함수, after함수 사용	countdown
크롤링	네이버와 다음의 검색어 크롤링 Chrome driver 사용	crawl
검색어의 워드클라우드	순위에 따라 글씨 크기를 변화하여 시각화 Wordcloud 함수 이용	wordcloud
순위에 진입한 횟수 상위 5개 검색어 찾기	순위에 진입한 횟수를 계산하여 top 5 확인	inserttop
검색어 표현	1~10위 순으로 검색어 나열 Button으로 나열	printword
포털별 검색어 순위 차이 확인	포털별 검색어의 순위 차이 확인 (가장 큰/작은 검색어 확인) 순위 차이 : | Naver 순위 – Daum 순위 |	finddiff
상관관계	포털별 총 검색어 순위의 상관관계 / 특정 단어의 포털별 순위간 상관관계 Numpy 패키지의 corrcoef함수 이용	corrcoef
특정 검색어의 순위 변화	Plot함수를 이용하여 변화 표현	findPlot
연관 검색어	포털사이트에 검색했을 때 제공되는 연관 검색어 크롤링	relate
관련 뉴스	포털사이트에 검색했을 때 제공되는 뉴스 제목 2개씩 크롤링	
특정 검색어의 영상 찾기	Youtube에 검색어를 입력한 경우 나오는 영상 보여주기	video
특정 검색어의 이미지 찾기	Google에 검색어를 입력한 경우 나오는 이미지 보여주기	image

II. 기능 설명

A. 주기 선택

사용자가 원하는 주기를 입력 받는다. 그리고 입력 받은 주기에 따라 검색어가 실시간으로 업데이트 될 것이다.

```
def Seasonal(self):
    self.f = Frame(self.window,bg="snow"); self.f.pack()
    self.ls = Label(self.f, text="주기를 입력하시오 (단위: 분) ", font=("Malgun Gothic",11),bg="snow"); self.ls.pack(side="left", ipadx=10, ipady=10)
    self.es = Entry(self.f); self.es.pack(side="left", ipadx=10, ipady=10)
    self.bs = Button(self.f, text="입력", command=self.insertData, font=("Malgun Gothic",11),bg="snow2"); self.bs.pack(side="left", padx=10, pady=10)

def insertData(self):
    self.seasonal=int(self.es.get())
    self.f.pack_forget()
    self.showInfo()
```

주기를 입력하고 입력 버튼을 선택하게 되면 insertData함수가 실행되어 seasonal변수에 저장된다. 이때, 주기는 분 단위로 입력해야 한다. 그리고 주기에 따라 실시간 검색어를 크롤링하고 다양한 기능을 시각화 할 함수가 실행되게 된다. 우선 showInfo함수가 실행되는데 이 함수에서는 앞으로 분석결과 및 기능을 보여줄 때 필요한 기본 배치를 정하는 함수이다.

B. 주기에 따른 시간 계산

A에서 입력 받은 주기를 이용하여 countdown함수에서 사용된다. 이 때, countdown함수는 초를 기준으로 계산하므로 분으로 입력 받은 주기에 60을 곱해서 매개변수로 사용된다. 그리고 시간을 체크할 때는 threading 패키지의 after함수를 이용한다. after함수는 이용하여 1초가 지날 때마다 다시 시간을 체크하고 프로그램에 명시해주는 역할을 한다. 만약에 입력한 시간이 지나면 저장하고 있던 변수를 모두 초기화하며 다시 crawl 함수를 실행하여 검색어를 업데이트한다. 이 기능을 프로그램이 종료될 때까지 계속 반복진행하기 위해서는 remaining변수에 주기가 끝남에 따라 계속 값을 저장해줘야 한다.

```
def countdown(self, remaining=None):
    if remaining is not None:
        self.remaining = remaining
    if self.remaining>0:
        self.l8.configure(text="update after %d sec" % self.remaining) # 남은 시간을 표시
        self.remaining = self.remaining - 1
        self.window.after(1000, self.countdown) # 1000ms(1초)가 지날 때마다 시간을 체크
    if self.remaining==0: # 모든 시간이 경과한 경우
        # 변수 초기화
        self.n = {}
        self.d = {}
        self.naverbutton = []
        self.daumbutton = []
        self.mw = [[], []]
        self.remaining=self.seasonal*60 # 다음번에 다시 사용하기 위해

        # 크롤링
        self.crawl()
```

C. 검색어 크롤링

처음부터 실행하였던 크롬드라이버를 이용하여 네이버와 다음사이트에 접근하여 검색어 정보를 가져온다. 다음이 10개의 검색어를 제공하므로 네이버도 10개만 가져온다. 그리고 나중에 시각화를 할 때 필요하므로 언제 크롤링을 했는지에 관한 시간 정보도 저장해 놓는다. 그 다음으로는 크롤링으로부터 생성된 raw data를 분석과 시각화에 필요한 데이터의 형태로 바꿔주는 preprocess함수를 실행한다.

```
def crawl(self):
    # naver
    now = time.localtime() # 현재 시간
    h=now.tm_hour
    m=now.tm_min

    tim="""//*[@id="content"]/div/div[2]/div[1]/div[1]/div/div/div/div[2]/a[3]/span[1]"""
    tim=ndriver.find_element_by_xpath(tim).click()

    # 현재시간
    hou="""//*[@id="u.c.layer.time"]/div[6]/a[1]"""
    hour=ndriver.find_element_by_xpath(hou).click()

    ok="""//*[@id="u.c.layer.time"]/div[6]/a[2]"""
    send=ndriver.find_element_by_xpath(ok).click()

    html1 = ndriver.page_source
    soup1 = BeautifulSoup(html1, 'html.parser')
    rank1 = soup1.find_all("span",{"class":"item_title"})
    self.naver=[]
    for i in range(10):
        for a in rank1[i]:
            self.naver.append(a) # naver 검색어 10개 저장
    self.naverl.append(self.naver)

    # daum
    url = "https://www.daum.net/"
    page = urlopen(url)  ## web page read
    soup2 = BeautifulSoup(page, 'html.parser')
    rank2 = soup2.find_all("a",{"class":"link_issue"})
    self.daum=[]
    for i in range(20):
        self.daum.append(rank2[i].text) # 다음 검색어 10개 저장
    self.daum = self.daum[::2]
    self.dauml.append(self.daum)

    self.t.append(str(now.tm_mon)+"/"+str(now.tm_mday)+" "+str(h)+":"+str(m)) # 시간 저장

    # 분석과 시각화에 필요한 데이터의 형태로 바꿔주기 위한 전처리 함수
    self.preprocess()
```

D. 검색어의 워드클라우드

Preprocess함수에서 순위와 단어를 dataframe으로 전환하여 저장한 n, d변수를 이용하여 wordcloud를 만든다. 글씨의 크기는 순위가 높을수록(1위에 가까울수록) 크게 나타나도록 한다. 워드클라우드를 사진으로 저장하여 gui로 구현할 때 사용된다.

in preprocess함수

```
# wordcloud용
for i in range(10):
    self.n[self.naverday.loc[i,"word"]]=11-self.naverday.loc[i,"rank"]
    self.d[self.daumday.loc[i,"word"]]=11-self.daumday.loc[i,"rank"]
```

```
def wordcloud(self):
    colors = ["#BF0A30", "#002868"]
    cmap = LinearSegmentedColormap.from_list("mycmap", colors)

    # 네이버
    wordcloud = WordCloud(font_path="BMJUA_ttf.ttf",background_color="white",random_state=5,min_font_size=10,max_font_size=80,colormap=cmap).generate_from_frequencies(self.n)
    fig = plt.figure(figsize=[4, 1.6])
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    fig.savefig(self.name[0]+".png")
    plt.clf()

    self.imga = PhotoImage(file=self.name[0] + ".png")
    self.l1.config(image=self.imga)

    # 다음
    wordcloud = WordCloud(font_path="BMJUA_ttf.ttf", background_color="white", random_state=5, min_font_size=10, max_font_size=80, colormap=cmap).generate_from_frequencies(self.d)
    fig = plt.figure(figsize=[4, 1.6])
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    fig.savefig(self.name[1] + ".png")

    self.imgb = PhotoImage(file=self.name[1] + ".png")
    self.l2.config(image=self.imgb)
    self.l1.image = self.imga
    self.l2.image = self.imgb
```

E. 순위에 진입한 상위 5개의 검색어 찾기

1~10위에 진입한 횟수를 preprocess함수에서 계산한 뒤 상위 5개를 선택하여 파이차트를 이용하여 시각화 한다. 주기를 짧게 하여 크롤링한 경우, 상위 5개의 검색어의 변화는 없지만 갑자기 광고가 순위에 오르거나 무슨 사건이 터진 경우에는 크게 변동이 있다. 이 파이차트를 사진으로 저장하여 gui로 구현할 때 사용된다.

```
def inserttop(self):
    plt.rc('font', family="Malgun Gothic")
    plt.rcParams.update({'figure.max_open_warning': 0})

    # 순위 안으로 진입한 횟수가 많은 순으로 정렬 + 상위 5개 선택
    self.ni=self.ni.sort_values(by="진입",ascending=False)
    self.topni=self.ni[:5]
    self.di=self.di.sort_values(by="진입",ascending=False)
    self.topdi = self.di[:5]

    labels=[]
    for i in range(5):
        labels.append(self.topni.iloc[i,0]+"("+str(self.topni.iloc[i,1])+")")

    fig3 = plt.figure(figsize=[4, 2])
    colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue','orange']
    explode = (0.1, 0.1, 0.1, 0.1,0.1)
    # 네이버 pie plot
    plt.pie(self.topni["진입"], explode=explode, labels=labels, autopct='%1.1f%%', colors=colors, shadow=True, startangle=140)
    plt.axis('equal')
    fig3.savefig("np.png")
    plt.clf()

    labels = []
    for i in range(5):
        labels.append(self.topdi.iloc[i, 0] + "(" + str(self.topdi.iloc[i, 1]) + ")")

    fig4 = plt.figure(figsize=[4, 2])
    # 다음 pie plot
    plt.pie(self.topdi["진입"], explode=explode, labels=labels, autopct='%1.1f%%', colors=colors, shadow=True, startangle=140)
    plt.axis('equal')
    fig4.savefig("dp.png")

    self.np = PhotoImage(file="np.png")
    self.dp = PhotoImage(file="dp.png")
    self.l1_2.config(image=self.np)
    self.l2_2.config(image=self.dp)
    self.l1_2.image = self.np
    self.l2_2.image = self.dp
```

in preprocess함수

```
# 진입 횟수용
self.ni = pd.DataFrame({"word":self.nd.index,"진입":nl})
self.di = pd.DataFrame({"word":self.dd.index,"진입":dl})

self.findDiff()
self.wordcloud()
self.inserttop()
self.printWord()
self.corrcoef(self.nd.iloc[:,-2],self.dd.iloc[:,-2],1)
```

F. 검색어 표현

크롤링한 검색어를 보기 쉽게 포털사이트에서 제공되었던 형태 그대로 배치하여 제공한다. 네이버 옆에 다음을 배치하여 두 포털 간 비교가 쉽게 한다. 또한 궁금한 검색어가 존재하면 정보를 얻을 수 있게 버튼형식으로 검색어를 배치하였다.

```
def printWord(self):
    # 검색어 시각화
    self.labels = []
    for i in range(10):
        c = lambda index=i: self.create_window(self.naver[index])
        self.naverbutton.append(Button(self.f2_1, text=self.naver[i],command=c,font=("Malgun Gothic",11),width=30,bg="white"))
        self.labels.append(Label(self.f2_1, text=str(i+1)+"위",font=("Malgun Gothic",11,'bold'),bg="ghost white"))
        self.labels[i].grid(row=i + 1, column=0,padx=10, pady=2)
        self.naverbutton[i].grid(row=i + 1, column=1, padx=10, pady=2)

    self.labels = []
    for i in range(10):
        c = lambda index=i: self.create_window(self.daum[index])
        self.daumbutton.append(Button(self.f3_1, text=self.daum[i],command=c,font=("Malgun Gothic",11),width=30,bg="white"))
        self.labels.append(Label(self.f3_1, text=str(i+1)+"위",font=("Malgun Gothic",11,'bold'),bg="ghost white"))
        self.labels[i].grid(row=i + 1, column=0, padx=10, pady=2)
        self.daumbutton[i].grid(row=i + 1, column=1, padx=10, pady=2)
```

G. 포털 별 검색어 순위 차이 확인

궁금한 검색어에 따라 포털 별 검색어 순위를 비교한다. 순위의 차는 (네이버의 순위 - 다음의 순위)에 절댓값을 씌운 값을 이용한다. 만약에 이전에 등장하지 않았던 검색어가 존재하면 그 검색어는 11위라고 가정하고 계산했다. 순위차가 가장 큰 경우와 작은 경우의 검색어가 모두 프로그램에서 제시된다.

in preprocess함수

```
self.naverday=pd.DataFrame({"word":self.naver,"rank":range(10)})
self.naverday["rank"]+=1

self.daumday=pd.DataFrame({"word":self.daum,"rank":range(10)})
self.daumday["rank"]+=1

self.totalword=pd.concat([self.naverday["word"],self.daumday["word"]]).unique()
# 차이 구하는 용
self.td=pd.DataFrame(0,index=range(len(self.totalword)),columns=["totalword","naver","daum"])
self.td["totalword"]=self.totalword

for i in range(10):
    for j in range(len(self.td["totalword"])):
        if self.naverday.iloc[i,0]==self.td.iloc[j,0]:
            self.td.iloc[j,1]=self.naverday.iloc[i,1]
        if self.daumday.iloc[i,0]==self.td.iloc[j,0]:
            self.td.iloc[j,1]=self.naverday.iloc[i,1]
```

```
def findDiff(self):
    # 포털별 순위차이가 가장 큰/작은 검색어 분석
    self.td["absdiff"]=abs(self.td["naver"]-self.td["daum"])
    maxdiff=self.td["absdiff"].max()
    mindiff=self.td["absdiff"].min()
    for i in range(len(self.td)):
        if self.td.loc[i,"absdiff"]==maxdiff:
            self.mw[0].append(self.td.loc[i,"totalword"])
        if self.td.loc[i,"absdiff"]==mindiff:
            self.mw[1].append(self.td.loc[i,"totalword"])

    self.l5.config(text="포털별 순위차이가 가장 큰 검색어: "+", ".join(self.mw[0])+"₩n포털별 순위차이가 가장 작은 검색어: "+", ".join(self.mw[1]))
```

H. 상관 관계

포털사이트 별 전체 단어의 상관관계와 선택한 특정 검색어의 시간에 따른 순위 변화의 상관관계를 구할 수 있다. 상관관계는 numpy 패키지의 corrcoef함수를 이용하여 구한다. 이때의 상관관계는 중요도를 기준으로 구한다. 중요도는 1순위일수록 높다고 생각하여 모든 순위에 12를 뺀 값을 이용하여 중요도를 지정한다. 그리고 10위 이내에 들지 못한 검색어는 중요도 1을 준다. 포털사이트 별 전체 검색어간의 상관관계는 그 시점의 중요도를 이용하여 구하지만, 특정 검색어의 시간에 따른 순위 변화의 상관관계를 구할 때는 프로그램을 시작하기 전의 정보가 없기 때문에 따로 데이터 값을 부여한다. 이전 시간의 순위는 모르기 때문에 그에 해당하는 중요도는 0이라고 정하고 그 이후에 각각의 순위로 이동했다고 가정하여 맨처음에 크롤링한 검색어의 중요도의 초기값으로 0을 모두 부여하여 중요도에 따른 상관계수를 구했다. 즉, t_0시간의 순위의 중요도를 0이라고 하고 t_1시간 이후의 순위를 크롤링하여 얻은 순위에 해당하는 중요도를 가졌다고 지정한다.

in preprocess함수

```
# 상관관계, 그래프용
self.nd = pd.DataFrame(np.nan, index=self.totalword, columns=self.t)
self.dd = pd.DataFrame(np.nan, index=self.totalword, columns=self.t)
nl=[0]*len(self.nd.index)
dl=[0]*len(self.dd.index)

for i in range(len(self.t)):
    for j in range(len(self.nd.index)):
        for k in range(10):
            if self.naverl[i][k] == self.nd.index[j]:
                self.nd.iloc[j,i] = k + 1
                nl[j]+=1
self.nd["site"]="Naver"

for i in range(len(self.t)):
    for j in range(len(self.dd.index)):
        for k in range(10):
            if self.dauml[i][k] == self.dd.index[j]:
                self.dd.iloc[j, i] = k + 1
                dl[j] += 1
self.dd["site"] = "Daum"
```

```
# 상관관계
    def corrcoef(self, a, b, num):
        a=a.copy() # 순위값이 변하면 안되므로 copy
        b=b.copy() # 순위값이 변하면 안되므로 copy
        for i in range(len(a)):
            if a.iloc[i]!=np.nan:
                a.iloc[i]=12-a.iloc[i]
            if b.iloc[i]!=np.nan:
                b.iloc[i]=12-b.iloc[i]
        a = a.fillna(1)
        b = b.fillna(1)

        if num == 1:  # 포털사이트 별 전체 순위의 상관관계
            ndfl = a.tolist()
            ddfl = b.tolist()
            self.coco = round(np.corrcoef(ndfl, ddfl)[0, 1], 3)
            self.l6.config(text="포털별 상관관계 정도 : " + str(round(self.coco, 3)))
```

I. 특정 검색어의 순위 변화

선택한 검색어의 시간에 따른 순위 변화를 위해 크롤링할 때마다 모든 단어와 순위를 저장한 naverl, dauml 변수를 preprocess함수에서 시간대별로 나눠서 dataframe으로 저장한다. 이 때, nd, dd에 새로 만든 dataframe을 저장하고 선택한 단어와 맞는 순위만을 가져와 시간에 따른 순위 변화를 시각화 한다.

in preprocess함수

```
# 상관관계, 그래프용
self.nd = pd.DataFrame(np.nan, index=self.totalword, columns=self.t)
self.dd = pd.DataFrame(np.nan, index=self.totalword, columns=self.t)
nl=[0]*len(self.nd.index)
dl=[0]*len(self.dd.index)

for i in range(len(self.t)):
    for j in range(len(self.nd.index)):
        for k in range(10):
            if self.naverl[i][k] == self.nd.index[j]:
                self.nd.iloc[j,i] = k + 1
                nl[j]+=1
self.nd["site"]="Naver"

for i in range(len(self.t)):
    for j in range(len(self.dd.index)):
        for k in range(10):
            if self.dauml[i][k] == self.dd.index[j]:
                self.dd.iloc[j, i] = k + 1
                dl[j] += 1
self.dd["site"] = "Daum"
```

```
def findPlot(self):
    plt.rc('font', family="Malgun Gothic")
    plt.rcParams['axes.unicode_minus'] = False
    plt.rcParams.update({'figure.max_open_warning': 0})

    # 선택한 단어와 맞는 순위 만을 가져와 시간에 따른 순위 변화를 시각화
    np = self.nd[self.nd.index == self.w]
    dp = self.dd[self.dd.index == self.w]
    p = pd.concat([np, dp])
    p2 = p.reset_index(drop=True)
    p3 = p2.set_index('site')
    del p3.index.name
    self.df = p3.T

    fig2 = plt.figure(figsize=[8, 4])
    plt.plot(self.df["Daum"], marker="o", lw=2, label="Daum")
    plt.plot(self.df["Naver"], marker="x", lw=2, ms=11, label="Naver")
    plt.title(self.w + "의 순위 변화")
    plt.ylim(11, -1)
    plt.xlabel("시간")
    plt.ylabel("실시간 검색어 순위 (단위:위)")
    plt.legend()
    can = FigureCanvasTkAgg(fig2, self.canvas)
    can.get_tk_widget().pack()
```

J. 연관 검색어, 관련 뉴스 크롤링

선택한 검색어에 관한 정보를 알고자할 때 검색어를 클릭하면 연관 검색어와 관련된 뉴스가 나오도록

구현했다. 검색어를 입력하였을 때 제공되는 연관 검색어 전체와 관련 뉴스를 4개 크롤링한다. 프로그램에 나타낼 때는 중복되는 것들은 빼고 제공한다.

```
def relate(self):
    # 네이버 연관검색어
    url="https://search.naver.com/search.naver?sm=top_hty&fbm=1&ie=utf8&query=" + quote(self.w)
    page = urlopen(url)
    soup2 = BeautifulSoup(page, 'html.parser')
    rank2 = soup2.find_all("a", {"data-area": "*q"})
    self.relateword = []
    for i in range(len(rank2)):
        self.relateword.append(rank2[i].text)

    # 다음 연관검색어
    url="https://search.daum.net/search?w=tot&DA=YZR&t__nil_searchbox=btn&sug=&sugo=&q=" + quote(self.w)
    page = urlopen(url)
    soup2 = BeautifulSoup(page, 'html.parser')
    rank2 = soup2.find_all("a", {"class": "keyword"})
    for i in range(len(rank2)):
        self.relateword.append(rank2[i].text)

    # 연관 뉴스
    url = "https://search.naver.com/search.naver?where=news&sm=tab_jum&query=" + quote(self.w)
    page = urlopen(url)
    soup2 = BeautifulSoup(page, 'html.parser')
    rank2 = soup2.find_all("a", {"class": "_sp_each_title"})
    self.relatenews = []
    for i in range(4):
        self.relatenews.append(rank2[i].text)

    # 연관 검색어와 뉴스의 포털별 중복 제거
    self.relateword = list(set(self.relateword))
    self.relatenews = list(set(self.relatenews))

    self.lt = Label(self.rel, text=" [ 연관 검색어 ] ", font=("Malgun Gothic", 15), bg="snow")
    self.lt.grid(row=0, column=0, columnspan=4)

    self.rw = []
    self.rn = []
    length = len(self.relateword) // 4 + 1
    cnt = 0
    if len(self.relateword) == 0:
        self.words = Label(self.rel, text="없음", font=("Malgun Gothic", 12), bg="snow");
        self.words.grid(row=1, column=0, columnspan=4)
    else:
        for i in range(length):
            for j in range(4):
                if i * 4 + j != len(self.relateword):
                    self.rw.append(
                        Label(self.rel, text=self.relateword[cnt], font=("Malgun Gothic", 11), bg="snow"))
                    self.rw[cnt].grid(row=i + 1, column=j, padx=10, pady=2)
                    cnt += 1
                else:
                    break

    self.lt = Label(self.rel, text=" [ 연관 뉴스 ] ", font=("Malgun Gothic", 15), bg="snow")
    self.lt.grid(row=length + 2, column=0, columnspan=4)
    cnt = 0
    if len(self.relatenews) == 0:
        self.news = Label(self.rel, text="없음", font=("Malgun Gothic", 12), bg="snow");
        self.news.grid(row=1, column=0, columnspan=4)
    else:
        for i in range(len(self.relatenews)):
            self.rn.append(Label(self.rel, text=self.relatenews[cnt], font=("Malgun Gothic", 11), bg="snow"))
            self.rn[cnt].grid(row=length + i + 3, column=0, columnspan=4, padx=10, pady=2)
            cnt += 1
```

K. 특정 검색어의 영상, 이미지 찾기

Youtube 사이트를 chromedriver로 열어서 선택한 검색어에 관한 영상을 제공한다.

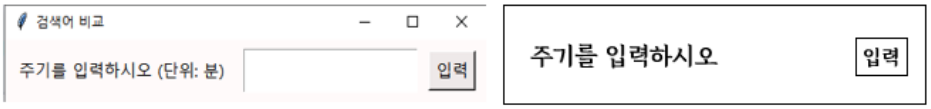
```
def video(self):
    self.ydriver = webdriver.Chrome('driver/chromedriver')  # 크롬 드라이버 지정 - 유투브
    self.ydriver.get("https://www.youtube.com/results?search_query="+self.w)
```

Google 사이트를 chromedriver로 열어서 선택한 검색어에 관한 이미지를 제공한다.

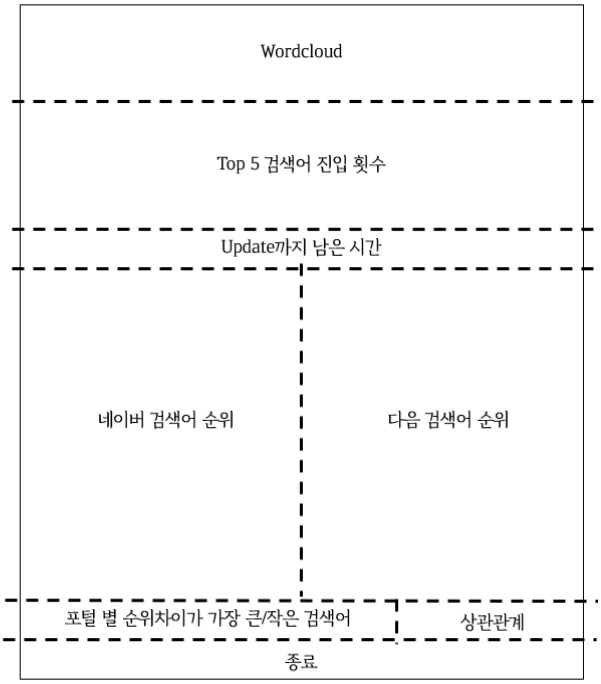
```
def image(self):
    self.gdriver = webdriver.Chrome('driver/chromedriver')  # 크롬 드라이버 지정 - 구글
    self.gdriver.get("https://www.google.co.kr/imghp?hl=ko&tab=wi&ogbl")
    self.gdriver.find_element_by_name('q').send_keys(self.w)
    self.gdriver.find_element_by_xpath("""//*[@id="sbtc"]/button/div/span""").click()
```

III. GUI 구현

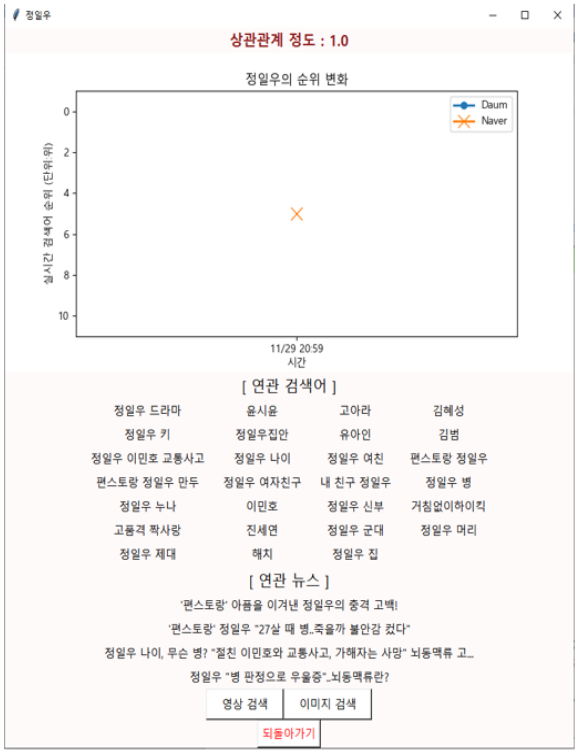
1) 주기 입력

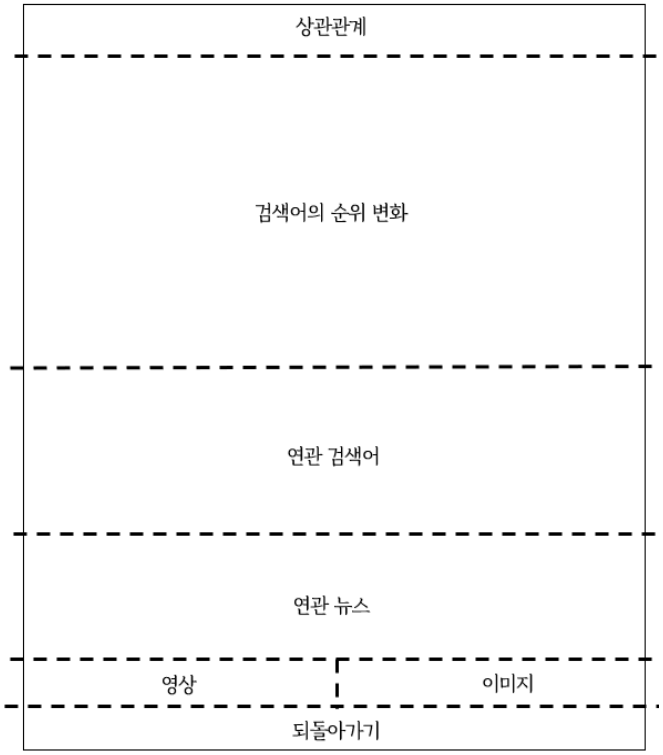


2) 주기를 입력한 경우, 기본 페이지

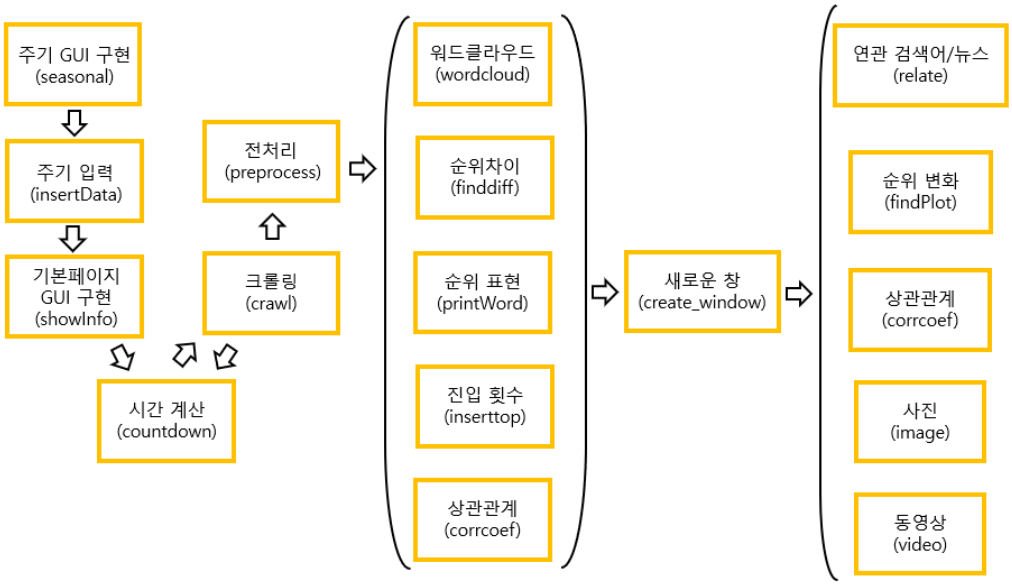


3) 특정 검색어를 입력했을 때,





IV. 관계도



V. 프로그램 실행 환경

1. selenium, bs4, pandas, numpy, wordcloud 패키지를 다운 받아 프로그램을 실행한다.

크롤링용 : selenium, bs4, urllib

전처리용 : time, pandas, numpy

그래프용 : wordcloud, matpotlib

GUI용 : tkinter, threading

```
from selenium import webdriver
from bs4 import BeautifulSoup
from urllib.request import urlopen  # URL open import

import time
import pandas as pd
import numpy as np

from wordcloud import WordCloud
from matplotlib.colors import LinearSegmentedColormap
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt

from tkinter import *
from tkinter import messagebox
from threading import Timer
```