머신러닝 데이터 수집과 분석 시각화

```
import pandas as pd
import numpy as np
import platform
import matplotlib.pyplot as plt
from selenium import webdriver
import time
import re
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from bs4 import BeautifulSoup
from selenium.common.exceptions import WebDriverException
from matplotlib import rc, font_manager

from context.domains import Reader, File

from context.models import Model


class Solution(Reader):
  def __init__(self):
      rc('font', family=font_manager.FontProperties(fname='C:/Windows/Fonts/malgunsl.ttf').get_name())
      self.file = File()
      self.driver = webdriver.Chrome("C:/Users/minseo/Downloads/chromedriver_win32/chromedriver.exe") # chromedriver 설치 후 경로 복사, \
      self.file.context = './data/'

  def hook(self):
          def print_menu():
              print('0. Exit')
              print('1. driver를 이용한 크롤링 후 시,도 이름 뽑기.')
              print('2. 지역별 대선 결과 득표율 시각화 하기 ')
              return input('메뉴 선택 \n')

          while 1:
              menu = print_menu()
              if menu == '0':
                  break
              if menu == '1':
                  self.get_data()
              if menu == '2':
                  self.draw_korea_result()
              elif menu == '0':
                  break

# Selenium과 Beautiful Soup을 이용한 데이터 획득 준비 작업
  def get_data(self):
      driver = self.driver
      try:
          driver.get(
              "http://info.nec.go.kr/main/showDocument.xhtml?electionId=0000000000&topMenuId=VC&secondMenuId=VCCP09")
          driver.get(
              "http://info.nec.go.kr/main/showDocument.xhtml?electionId=0000000000&topMenuId=VC&secondMenuId=VCCP09")
      except WebDriverException:
          print("page down")

      driver.find_element(By.ID, "electionType1").click()
      driver.find_element(By.ID, "electionName").send_keys("제19대")
      time.sleep(1)
      driver.find_element(By.ID, "electionCode").send_keys("대통령선거")
      time.sleep(1)
      sido_list_raw = driver.find_element_by_xpath("""//*[@id="cityCode"]""")
      sido_list = sido_list_raw.find_elements_by_tag_name("option")
      sido_names_values = [option.text for option in sido_list]
      sido_names_values = sido_names_values[2:]
      # print(sido_names_values)
      return sido_names_values

# 19대 대선 개표 결과 데이터 획득하기
  @staticmethod
    def get_num(tmp):
        return float(re.split('\(', tmp)[0].replace(',', ''))

    def move_sido(self, name):
        driver = self.driver
        wait = WebDriverWait(driver, 40)
        element = driver.find_element(By.ID, "cityCode")
        element.send_keys(name)
        make_xpath = """//*[@id="searchBtn"]"""
        wait.until(EC.element_to_be_clickable((By.XPATH, make_xpath)))
        driver.find_element(By.XPATH, make_xpath).click()
```

```
        time.sleep(30)

    def append_data(self, df, sido_name, data):
        for each in df[0].values[1:]:
            data['광역시도'].append(sido_name)
            data['시군'].append(each[0])
            data['pop'].append(self.get_num(each[2]))
            data['moon'].append(self.get_num(each[3]))
            data['hong'].append(self.get_num(each[4]))
            data['ahn'].append(self.get_num(each[5]))

    def get_result_data(self):
        driver = self.driver
        election_result_raw = {'광역시도': [],
                               '시군': [],
                               'pop': [],
                               'moon': [],
                               'hong': [],
                               'ahn': []}
        sido_names_values = self.get_data()
        for each_sido in sido_names_values:
            self.move_sido(each_sido)
            html = self.driver.page_source
            time.sleep(10)
            soup = BeautifulSoup(html, 'html.parser')
            table = soup.find('table')
            df = pd.read_html(str(table))
            self.append_data(df, each_sido, election_result_raw)
        election_result = pd.DataFrame(election_result_raw,
                                       columns=['광역시도', '시군', 'pop', 'moon', 'hong', 'ahn'])
        election_result.to_csv('./data/election_result.csv', encoding='utf-8', sep=',')
        driver.close()

    # 각 후보의 득표율과 지역 ID 정리하기
    def read_election_result(self):
        file = self.file
        file.fname = 'election_result'
        election_result = self.csv(file)
        # print(election_result.head())
        return election_result

    def sigun_candi(self):
        election_result = self.read_election_result()
        # '광역시도' 이름에서 앞 두 글자 추출
        sido_candi = election_result['광역시도']
        sido_candi = [name[:2] if name[:2]
                                  in ['서울', '부산', '대구', '광주', '인천', '대전', '울산']
                      else '' for name in sido_candi]
        sigun_candi = [''] * len(election_result)

        # 광역시가 아닌데 행정구를 가지고 있는 '시군'에 대해 처리
        for n in election_result.index:
            each = election_result['시군'][n]
            if each[:2] in ['수원', '성남', '안양', '안산', '고양',
                            '용인', '청주', '천안', '전주', '포항', '창원']:
                sigun_candi[n] = re.split('시', each)[0] + ' ' + \
                                 self.cut_char_sigu(re.split('시', each)[1])
            else:
                sigun_candi[n] = self.cut_char_sigu(each)
        print(sigun_candi)

    @staticmethod
    def cut_char_sigu(name):
        return name if len(name) == 2 else name[:-1]

    def organize_data(self):
        file = self.file
        file.fname = 'election_result'
        election_result = self.csv(file)
        # print(election_result.head())
        sido_candi = election_result['광역시도']
        sido_candi = [name[:2] if name[:2]
                                  in ['서울', '부산', '대구', '광주', '인천', '대전', '울산']
                      else '' for name in sido_candi]
        sigun_candi = [''] * len(election_result)
        # print(sigun_candi)

        for n in election_result.index:
            each = election_result['시군'][n]
            if each[:2] in ['수원', '성남', '안양', '안산', '고양',
                            '용인', '청주', '천안', '전주', '포항', '창원']:
                sigun_candi[n] = re.split('시', each)[0] + ' ' + \
                                 self.cut_char_sigu(re.split('시', each)[1])
            else:
                sigun_candi[n] = self.cut_char_sigu(each)
        ID_candi = [sido_candi[n] + ' ' + sigun_candi[n] for n in range(0, len(sigun_candi))]
        # 세종시의 경우 예외로 처리
```

```
        ID_candi = [name[1:] if name[0] == ' ' else name for name in ID_candi]
        ID_candi = [name[:2] if name[:2] == '세종' else name for name in ID_candi]
        # print(ID_candi)

        election_result['ID'] = ID_candi
        # print(election_result.head(10))

        election_result[['rate_moon', 'rate_hong', 'rate_ahn']] = \
            election_result[['moon', 'hong', 'ahn']].div(election_result['pop'], axis=0)
        election_result[['rate_moon', 'rate_hong', 'rate_ahn']] *= 100
        # print(election_result.head())
        # print(election_result.sort_values(['rate_moon'], ascending=[False]).head(10))
        # print(election_result.sort_values(['rate_hong'], ascending=[False]).head(10))
        # print(election_result.sort_values(['rate_ahn'], ascending=[False]).head(10))

        draw_korea = pd.read_csv('./data/draw_korea.csv', encoding='utf-8',
                                 index_col=0)
        # print(draw_korea.head())
        set(draw_korea['ID'].unique()) - set(election_result['ID'].unique())
        set(draw_korea['ID'].unique()) - set(election_result['ID'].unique())
        set(draw_korea['ID'].unique()) - set(election_result['ID'].unique())
        election_result.loc[125, 'ID'] = '고성(강원)'
        election_result.loc[233, 'ID'] = '고성(경남)'
        # print(election_result[election_result['시군'] == '고성군'])

        election_result.loc[125, 'ID'] = '고성(강원)'
        election_result.loc[233, 'ID'] = '고성(경남)'
        # print(election_result[election_result['시군'] == '고성군'])

        election_result.loc[228, 'ID'] = '창원 합포'
        election_result.loc[229, 'ID'] = '창원 회원'
        # print(election_result[election_result['광역시도'] == '경상남도'])

        set(draw_korea['ID'].unique()) - set(election_result['ID'].unique())
        set(election_result['ID'].unique()) - set(draw_korea['ID'].unique())
        # print(election_result[election_result['시군'] == '부천시'])
        election_result.tail()

        ahn_tmp = election_result.loc[85, 'ahn'] / 3
        hong_tmp = election_result.loc[85, 'hong'] / 3
        moon_tmp = election_result.loc[85, 'moon'] / 3
        pop_tmp = election_result.loc[85, 'pop'] / 3

        rate_moon_tmp = election_result.loc[85, 'rate_moon']
        rate_hong_tmp = election_result.loc[85, 'rate_hong']
        rate_ahn_tmp = election_result.loc[85, 'rate_ahn']
        '''
        election_result.loc[250] = [ahn_tmp, hong_tmp, moon_tmp, pop_tmp,
                                    '경기도', '부천시', '부천 소사',
                                    rate_moon_tmp, rate_hong_tmp, rate_ahn_tmp]
        election_result.loc[251] = [ahn_tmp, hong_tmp, moon_tmp, pop_tmp,
                                    '경기도', '부천시', '부천 오정',
                                    rate_moon_tmp, rate_hong_tmp, rate_ahn_tmp]
        election_result.loc[252] = [ahn_tmp, hong_tmp, moon_tmp, pop_tmp,
                                    '경기도', '부천시', '부천 원미',
                                    rate_moon_tmp, rate_hong_tmp, rate_ahn_tmp]
        election_result[election_result['시군'] == '부천시']

        '''
    def draw_election_result(self):
        pas
```

```
    @staticmethod
    def drawKorea(targetData, blockedMap, cmapname):
        BORDER_LINES = [
            [(5, 1), (5, 2), (7, 2), (7, 3), (11, 3), (11, 0)],  # 인천
            [(5, 4), (5, 5), (2, 5), (2, 7), (4, 7), (4, 9), (7, 9),
             (7, 7), (9, 7), (9, 5), (10, 5), (10, 4), (5, 4)],  # 서울
            [(1, 7), (1, 8), (3, 8), (3, 10), (10, 10), (10, 7),
             (12, 7), (12, 6), (11, 6), (11, 5), (12, 5), (12, 4),
             (11, 4), (11, 3)],  # 경기도
            [(8, 10), (8, 11), (6, 11), (6, 12)],  # 강원도
            [(12, 5), (13, 5), (13, 4), (14, 4), (14, 5), (15, 5),
             (15, 4), (16, 4), (16, 2)],  # 충청북도
            [(16, 4), (17, 4), (17, 5), (16, 5), (16, 6), (19, 6),
             (19, 5), (20, 5), (20, 4), (21, 4), (21, 3), (19, 3), (19, 1)],  # 전라북도
            [(13, 5), (13, 6), (16, 6)],  # 대전시
            [(13, 5), (14, 5)],  # 세종시
            [(21, 2), (21, 3), (22, 3), (22, 4), (24, 4), (24, 2), (21, 2)],  # 광주
            [(20, 5), (21, 5), (21, 6), (23, 6)],  # 전라남도
            [(10, 8), (12, 8), (12, 9), (14, 9), (14, 8), (16, 8), (16, 6)],  # 충청북도
            [(14, 9), (14, 11), (14, 12), (13, 12), (13, 13)],  # 경상북도
            [(15, 8), (17, 8), (17, 10), (16, 10), (16, 11), (14, 11)],  # 대구
```

```
            [(17, 9), (18, 9), (18, 8), (19, 8), (19, 9), (20, 9), (20, 10), (21, 10)],  # 부산
            [(16, 11), (16, 13)],  # 울산
            [(27, 5), (27, 6), (25, 6)],
        ]
        gamma = 0.75

        whitelabelmin = 20.

        datalabel = targetData

        tmp_max = max([np.abs(min(blockedMap[targetData])),
                       np.abs(max(blockedMap[targetData]))])
        vmin, vmax = -tmp_max, tmp_max

        mapdata = blockedMap.pivot_table(index='y', columns='x', values=targetData)
        masked_mapdata = np.ma.masked_where(np.isnan(mapdata), mapdata)

        plt.figure(figsize=(9, 11))
        plt.pcolor(masked_mapdata, vmin=vmin, vmax=vmax, cmap=cmapname,
                   edgecolor='#aaaaaa', linewidth=0.5)


        # 지역 이름 표시
        for idx, row in blockedMap.iterrows():
            # 광역시는 구 이름이 겹치는 경우가 많아서 시단위 이름도 같이 표시한다.
            # (중구, 서구)
            if len(row['ID'].split()) == 2:
                dispname = '{}\n{}'.format(row['ID'].split()[0], row['ID'].split()[1])
            elif row['ID'][:2] == '고성':
                dispname = '고성'
            else:
                dispname = row['ID']

            # 서대문구, 서귀포시 같이 이름이 3자 이상인 경우에 작은 글자로 표시한다.
            if len(dispname.splitlines()[-1]) >= 3:
                fontsize, linespacing = 10.0, 1.1
            else:
                fontsize, linespacing = 11, 1.

            annocolor = 'white' if np.abs(row[targetData]) > whitelabelmin else 'black'
            plt.annotate(dispname, (row['x'] + 0.5, row['y'] + 0.5), weight='bold',
                         fontsize=fontsize, ha='center', va='center', color=annocolor,
                         linespacing=linespacing)

        # 시도 경계 그린다.
        for path in BORDER_LINES:
            ys, xs = zip(*path)
            plt.plot(xs, ys, c='black', lw=2)

        plt.gca().invert_yaxis()

        plt.axis('off')

        cb = plt.colorbar(shrink=.1, aspect=10)
        cb.set_label(datalabel)

        plt.tight_layout()
        plt.show()

        model = Model()
        return plt.savefig(f'{model.get_sname()}{targetData}.png')


    def draw_korea_result(self):
        final_elect_data = self.organize_data()
        self.drawKorea('moon_vs_hong', final_elect_data, 'RdBu')
        self.drawKorea('moon_vs_ahn', final_elect_data, 'RdBu')
        self.drawKorea('ahn_vs_hong', final_elect_data, 'RdBu')

if __name__ == '__main__':
    Solution().organize_data()
    Solution().hook()
```