01_crawler

December 11, 2018

```
In [1]: import requests
        from IPython.core.display import HTML
        styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-CS109A/maste
        HTML(styles)

        from bs4 import BeautifulSoup
        import re
        import pandas as pd
        import time
        import json
        from pathlib import Path
        import numpy as np
        import os
        from os import listdir
        from os.path import isfile, join
```

Lets define some usefull functions and global variables we will use later:

```
In [2]: def check_or_save_page(filename, url):
            """
                Check if the file exist, if not get the page
                from the url and store in on the disk
                Returns the file content as a soup
            """
            # Check if the page has been stored on disk
            if Path(filename).is_file() is False:
                #print('No page')
                # Get the page
                result = requests.get(url)
                with open(filename,'w') as outfile:
                    outfile.write(result.text)
                time.sleep(2)
            #else:
                #print('We got it')

            with open(filename) as my_file:
                soup = BeautifulSoup(my_file.read(), "html.parser")
```

```
    return soup

# Declare global variables
states = ['Alabama', 'Alaska', 'Arizona', 'Arkansas', 'California', 'Colorado', 'Connec
state_to_iso2 = { 'Alabama': "AL", 'Alaska': "AK", 'Arizona': "AZ", 'Arkansas': "AR",

# National unemployement rate by month from 1948 to 2018
# Source: https://data.bls.gov/pdq/SurveyOutputServlet
national_unemployement_rate = pd.read_csv('data/national_unemployement_1948_2018.csv')

# Get the presidental job approval
# Source: https://www.gallup.com
# https://news.gallup.com/interactives/185273/presidential-job-approval-center.aspx
with open('data/all_presidential_job_approval_gallup.json') as f:
    presidential_approval = json.load(f)
presidential_approval = presidential_approval['AllPresidents']['HistoricalPresident']
presidential_approval_df = pd.DataFrame.from_dict([x['PresidentData'] for x in presiden
print('Presidential approval')
display(presidential_approval_df.head())

print('National unemployement rate')
display(national_unemployement_rate.head())

# List of the US presidents
president_elected_history = pd.read_csv('data/president_elected_history.csv', sep=';')
print('US presidents history')
display(president_elected_history.head())

"""
Get the national level factors
Source: https://en.wikipedia.org/wiki/United_States_presidential_election
"""

# From a tag, extract the number of seats
def extract_seats(tag):
    if tag.findAll('b'):
        d_seats = tag.b.extract().string
    elif tag.sup and tag.sup.decompose():
        d_seats = tag.sup.decompose()
    elif tag.string is None:
        d_seats = tag.text
    else:
        d_seats = tag.string
    return int(d_seats)

def extract_seats_change(tag):
    if tag.sup:
        d_seats_change = tag.text.split('[', 1)[0]
    else:
        d_seats_change = tag.text
    return int(d_seats_change.replace('', '-'))

# Get the house election years
def extract_house_elections_history():
    house_elections_history = []

    # If the file doesn't exist, get the data from the webpage and store the content t
    filename = 'data/list_of_house_elections_page.html'
    if Path(filename).is_file():
        #with open(filename) as my_file:
        with open(filename, encoding='utf-8') as my_file:
            list_of_house_elections_page = my_file.read()
    else:
        print('no file')
        list_of_house_elections_page = requests.get('https://en.wikipedia.org/wiki/Lis
        with open(filename,'w') as outfile:
            outfile.write(list_of_house_elections_page.text)

    soup = BeautifulSoup(list_of_house_elections_page, "html.parser")

    # Find the election years
    data = []
    elections_pages = []
    for t in soup.find_all('a', title=lambda x: x and 'United States House of Represen
        if len(t.string) == 4:
            elections_pages.append({
                'year': int(t.string),
                'url':'https://en.wikipedia.org'+t.attrs['href']
            })
            year = int(t.string)+2

            cols = t.parent.parent.find_all('td')

            # Get the number of Democrat seats
            d_seats = extract_seats(cols[1])

            # Get the change in the number of Democrat seats
            d_seats_change = extract_seats_change(cols[2])

            # Get the number of Republican seats
            r_seats = extract_seats(cols[3])

            # Get the change in the number of Republican seats
            r_seats_change_by_year = extract_seats_change(cols[4])

            #print(1 if year in presidential_years else 0)
```

```
                idx = (np.abs(president_elected_history['year'].values-year+1)).argmin()
                president_can_be_re_elected = president_elected_history['can_be_re_elected
                president_party = president_elected_history['president_elected_party'].loc

                # Look for president overall job approval average
                president_name = president_elected_history['president_elected'].loc[[idx]]
                president_overall_avg_job_approval = presidential_approval_df.loc[presiden
                president_overall_avg_job_approval = float(president_overall_avg_job_appro

                # Get the national unemployement rate for November
                oct_unemployement_rate = national_unemployement_rate.loc[national_unemploy

                oct_unemployement_rate = oct_unemployement_rate.values[0] if oct_unemploye

                data.append({
                    'year': year,
                    'is_presidential_year': 1 if year in president_elected_history['year']
                    'president_party': president_party,
                    'president_can_be_re_elected': president_can_be_re_elected,
                    'president_overall_avg_job_approval': president_overall_avg_job_approva
                    'oct_unemployement_rate': oct_unemployement_rate,
                    'last_democrat_seats': d_seats,
                    'last_republican seats': r_seats,
                    'last_house_majority': 'R' if d_seats < r_seats else 'D'
                })

        return data, elections_pages

    data, house_elections_pages = extract_house_elections_history()
    data_df = pd.DataFrame(data)
    national_level_factors = data_df[[
        'year',
        'is_presidential_year',
        'president_party',
        'president_can_be_re_elected',
        'president_overall_avg_job_approval',
        'oct_unemployement_rate',
        'last_democrat_seats',
        'last_republican seats',
        'last_house_majority']]
    print('National level factors')
    display(national_level_factors.sort_values('year', ascending=False).head())
```

```
Presidential approval


  DatesinOffice DaysInOffice     EndDate FirstTermAverage JobApprovalHigh  \
0  2017-Present         658                          -          45.0
1     2009-2017        2922  2017-01-20             48            67
2     2001-2009        2922  2009-01-20           62.2            90
3     1993-2001        2922  2001-01-20           49.6            73
4     1989-1993        1461  1993-01-20           60.9            89

  JobApprovalLow OverallAverage Party      PresidentName SecondTermAverage  \
0           35.0           39.5  Rep.    Donald J. Trump                 -
1             40             48  Dem.       Barack Obama                47
2             25           49.4  Rep.     George W. Bush              36.5
3             37           55.1  Dem.       Bill Clinton              60.6
4             29           60.9  Rep.  George H. W. Bush                 -

    StartDate
0  2017-01-20
1  2009-01-20
2  2001-01-20
3  1993-01-20
4  1989-01-20


National unemployement rate


   Year  Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
0  1948  3.4  3.8  4.0  3.9  3.5  3.6  3.6  3.9  3.8  3.7  3.8  4.0
1  1949  4.3  4.7  5.0  5.3  6.1  6.2  6.7  6.8  6.6  7.9  6.4  6.6
2  1950  6.5  6.4  6.3  5.8  5.5  5.4  5.0  4.5  4.4  4.2  4.2  4.3
3  1951  3.7  3.4  3.4  3.1  3.0  3.2  3.1  3.1  3.3  3.5  3.5  3.1
4  1952  3.2  3.1  2.9  2.9  3.0  3.0  3.2  3.4  3.1  3.0  2.8  2.7


US presidents history


   year       president_elected president_elected_party  can_be_re_elected
0  1824       John Quincy Adams                      DR                  1
1  1828          Andrew Jackson                       D                  1
2  1832          Andrew Jackson                       D                  0
3  1836        Martin Van Buren                       D                  1
4  1840  William Henry Harrison                       W                  1


National level factors


    year  is_presidential_year president_party  president_can_be_re_elected  \
80  2018                     0               R                            1
79  2016                     1               R                            1
```

```
78  2014                     0               D                            0
77  2012                     1               D                            0
76  2010                     0               D                            1

    president_overall_avg_job_approval  oct_unemployement_rate  \
80                               0.395                     3.7
79                               0.395                     4.9
78                               0.480                     5.7
77                               0.480                     7.8
76                               0.480                     9.4

    last_democrat_seats  last_republican seats last_house_majority
80                  194                    241                   R
79                  188                    247                   R
78                  201                    234                   R
77                  193                    242                   R
76                  257                    178                   D
```

1 Data Collection - Web Scraping - Data Parsing

```
In [3]: # Get once the necessary pages
        presidential_page = requests.get('https://en.wikipedia.org/wiki/United_States_presiden
```

```
In [4]: # List of the US presidents
        president_elected_history = pd.read_csv('data/president_elected_history.csv', sep=';')
        display(president_elected_history.head())
```

```
   year       president_elected president_elected_party  can_be_re_elected
0  1824       John Quincy Adams                      DR                  1
1  1828          Andrew Jackson                       D                  1
2  1832          Andrew Jackson                       D                  0
3  1836         Martin Van Buren                      D                  1
4  1840  William Henry Harrison                       W                  1
```

1.1 wikipedia.org

1.1.1 Get the House and Senate election result pages for all the available years

```
In [7]: """
        Get the district level factor
        """
        def get_district_list():
            district_list = []
            url = 'https://en.wikipedia.org/wiki/United_States_House_of_Representatives_electi
            filename = 'data/wikipedia/all_state_districts_list_page.html'
```

```
    # Check if the page has been stored on disk
    soup = check_or_save_page(filename, url)

    # Find the districts page links
    districts = soup.find_all('a', href=re.compile(r'(.*\/wiki\/.* )|(.*_congressional_

    for district in districts:
        if any(substring in district.string for substring in states) \
        and district.string not in district_list \
        and "'s" not in district.string \
        and "12th" not in district.string \
        and '1st' not in district.string:
            district_state = ''
            # Get the corresponding state
            for state in states:
                if state in district.string:
                    district_state = state

            # Format the district name
            if 'at-large' in district.string:
                dist_name = 'At-Large'
            else:
                # Find the district number
                dist_number = [int(s) for s in district.string.split() if s.isdigit()]
                if len(dist_number) > 0:
                    dist_number = dist_number[0]
                    dist_name = 'District {}'.format(dist_number)
                else:
                    continue

            #print(district_state, dist_name)

            district_list.append({
                'name': dist_name,
                'page_url': 'https://en.wikipedia.org{}'.format(district['href']),
                'state': district_state
            })

    # Remove duplicate in the list
    district_list = [dict(t) for t in {tuple(d.items()) for d in district_list}]

    return district_list

def get_wiki_district_pages(districts):
    # Get the district pages if they have not been stored on disk yet
    for district in districts:
        filename = 'data/district_pages/{}.html'.format(district['name'])
        # Check if the page has been stored on disk
        check_or_save_page(filename, district['page_url'])

def parse_district_house_results(filename, district, state):
    undesirable_chars = ['\*', '%', '\(incumbent\)', '\(inc.\)', '\(write-in\)']
    district_house_results = []
    with open(filename) as my_file:
        soup = BeautifulSoup(my_file.read(), "html.parser")

        # Find the election results tables
        caption = soup.find_all('caption')
        elems = []
        for capt in caption:
            x = capt.get_text()
            if ('United States House of Representatives elections,' in x or
                'congressional district election' in x or
                'US House election, ' in x or
                'Congressional District House Election'
            ):
                elems.append(capt)

        for capt in elems:
            # Find the date
            match = re.match(r'.*([1-2][0-9]{3})', capt.text)
            if match is None:
                continue

            # Then it found a match!
            year = int(match.group(1))
            #print(year)

            # Get the result table itself
            table = capt.find_parent('table')
            table_body = table.find('tbody')
            rows = table_body.find_all('tr')

            for row in rows:
                cols = row.find_all('td')
                cols = [ele.text.strip() for ele in cols]
                cols = [ele for ele in cols if ele] # Get rid of empty values

                if len(cols) and cols[0] in ['Republican', 'Democratic']:
                    print(cols)

                    percent = np.NaN
                    if len(cols) > 3 and cols[3] != 'N/A':
                        percent = float(re.sub("|".join(undesirable_chars), "", cols[3
                    votes = np.NaN
                    if len(cols) > 2 and cols[2] == 'N/A':
                        votes = np.NaN
                    elif len(cols) > 2 and '%' not in cols[2] and cols[2] != '100.00':
                        votes = int(cols[2].replace(',', '').replace('.', ''))
                    elif len(cols) > 2 and ('%' in cols[2] or cols[2] == '100.00'):
                        percent = float(re.sub("|".join(undesirable_chars), "", cols[2

                    district_house_results.append({
                        'year': year,
                        'candidate_party': 'R' if cols[0] == "Republican" else 'D',
                        'candidate_name': re.sub("|".join(undesirable_chars), "", cols
                        'votes': votes,
                        'percent': percent
                    })

    return pd.DataFrame(district_house_results)

def get_district_level_factors(district):
    state = districts_df.loc[districts_df['name'] == district]['state'].values[0]
    dist_level_factors = []
    # Get the page of the district
    html_filename = 'data/district_pages/{}.html'.format(district)
    json_filename = 'data/district_pages/{}.json'.format(district)

    # If there is no already formated data, get them from the corresponding Wikipedia 
    if Path(json_filename).is_file() is False:
        district_house_results = parse_district_house_results(html_filename, district,
        display(district_house_results)
    else:
        print('get from json')
        district_house_results = pd.read_json(json_filename)
        display(district_house_results)

    # Now, for each year
    for year in district_house_results['year'].unique():
        # If there is more than 1 candidate this year
        # Get the current year
        if len(district_house_results.loc[district_house_results['year'] == year]) > 1
            curr_year_idx = district_house_results.loc[district_house_results['year'] 
            curr_year = district_house_results.loc[[curr_year_idx]]
        else:
            curr_year = district_house_results

        # Get previous year
        prev_year = district_house_results.loc[district_house_results['year'] == year-

        if prev_year.empty is False:
```

```
        # If there is more than 1 candidate the previous year
        if len(prev_year) > 1:
            prev_year_winner_idx = prev_year['votes'].idxmax()
            prev_year_winner = prev_year.loc[[prev_year_winner_idx]]
        else:
            prev_year_winner = prev_year

        # Get the incumbent name
        incumbent = prev_year_winner['candidate_name'].values[0]

        # Get the elections previously won by the incumbent
        incumbent_history = district_house_results.loc[
            (district_house_results['candidate_name'] == incumbent) &
            (district_house_results['year'] < year)
        ]

        incubent_first_elected_idx = incumbent_history['year'].idxmin()
        incubent_first_elected = incumbent_history.loc[[incubent_first_elected_idx
        incubent_is_candidate = curr_year.loc[curr_year['candidate_name'] == incum

        dist_data = {
            'year': year,
            'state': state,
            'district': district,
            'incumbent': incumbent,
            'incumbent_party': 'R' if prev_year_winner['candidate_party'].values[0
            'incumbent_count_victories': len(incumbent_history),
            'incumbent_first_elected': incubent_first_elected,
            'incumbent_running_re_election': 0 if incubent_is_candidate else 1,
            'candidate_elected_party': curr_year['candidate_party'].values[0]
        }

        dist_level_factors.append(dist_data)
    else:
        print('yop')
        dist_data = {
            'year': year,
            'state': state,
            'district': district,
            'incumbent': np.NaN,
            'incumbent_party': np.NaN,
            'incumbent_count_victories': np.NaN,
            'incumbent_first_elected': np.NaN,
            'incumbent_running_re_election': np.NaN,
            'candidate_elected_party': curr_year['candidate_party'].values[0]
        }
        dist_level_factors.append(dist_data)
    return dist_level_factors

#for district in ['Alabamaă1', 'Alabamaă2']:
#for district in ['Arkansasă1']:
#    district_level_factors = get_district_level_factors(district)
#    display(pd.DataFrame(district_level_factors).sort_values('year', ascending=True))
#    #display(district_level_factors)

def get_wiki_districts_house_results(districts_list):
    candidate_results = []
    wiki_undesirable_chars = [
        '\*', '%', '\(Incumbent\)', '\(incumbent\)', '\(inc.\)', '\(write-in\)',
        '\(as a write-in\)'
    ]
    for district in districts_list:
        # To remove
        if district['state'] != 'Wisconsin' or district['name'] != 'District 2':
        #if district['state'] != 'Wyoming':
            continue

        print('Will get results for house/{}/{}.html'.format(district['state'], distric
        print('Source: {}'.format(district['page_url']))

        # In some cases, the wikipedia page is too messy to crawl
        # So I manually gather the informations into a json file
        # If this file exist, it will be prefered
        json_filename = 'data/wikipedia/house/{}/{}.json'.format(district['state'], di
        if Path(json_filename).is_file() is True:
            print('Data are store in a formated JSON')
            continue

        # Create the directories if necessary
        if not os.path.exists('data/wikipedia/house'):
            os.makedirs('data/wikipedia/house')
        if not os.path.exists('data/wikipedia/house/{}'.format(district['state'])):
            os.makedirs('data/wikipedia/house/{}'.format(district['state']))

        filename = 'data/wikipedia/house/{}/{}.html'.format(district['state'], distric

        # Check if the page has been stored on disk
        soup = check_or_save_page(filename, district['page_url'])

        # Find the results tables
        caption = soup.find_all('caption')
        tables = []
        for capt in caption:
            x = capt.get_text()
            if ('United States House of Representatives elections,' in x or
            'congressional district election' in x or
            'US House election, ' in x or
            'Congressional District House Election'
        ):
            # print(capt)
            table = capt.find_parent('table')
            tables.append(table)

    # For each result table, extract the results
    for table in tables:
        # Get the year
        table_title = table.find('caption')

        # If this is a table about a special election, skip it
        if 'Special' in table_title.text:
            continue

        year_match = re.match(r'.*([1-2][0-9]{3})', table_title.text)

        # If there is no year match, then this table isn't of interest
        if year_match is None:
            continue

        year = int(year_match.group(1))
        # To remove
        if year != 2018:
            continue
        # print(year)

        # Get the result table itself
        rows = table.find('tbody').find_all('tr')
        candidate_rows = []
        for row in rows:
            cols = row.find_all('td')
            cols = [ele.text.strip() for ele in cols]
            # If all the values of the cols are empty strings, continue
            if all(v is '' for v in cols):
                continue

            # print(cols)

            # If this row contains a candidate results
            if len(cols) > 2 and cols[1] in ['Republican', 'Democratic']:
                # print(cols)

                party = 'R' if cols[1] == 'Republican' else 'D'
                name = cols[2]
                votes = int(cols[3].replace(',', '').replace('[8]', '').replace('c
                    percent = float(cols[4].replace('%', '')) if cols[4] != '' else np

                    candidate_rows.append({
                        'year': year,
                        'state': district['state'],
                        'district': district['name'],
                        'is_incumbent': np.NaN,
                        'name': name,
                        'party': party,
                        'percent': percent,
                        'votes': votes,
                        'won': 0
                    })

            # If we found no candidate data, continue
            if len(candidate_rows) == 0:
                continue

            # Enrich the candidates data
            max_percent = max([x['percent'] for x in candidate_rows])
            for candidate in candidate_rows:
                # Check if the candidate won the elections
                if candidate['percent'] == max_percent:
                    candidate['won'] = 1

                # Check if we can determine if the candidate is an incumbent
                if '(inc.)' in candidate['name'] or '(incumbent)' in candidate['name']
                    candidate['is_incumbent'] = 1

                # Clean the candidate name
                candidate['name'] = re.sub("|".join(wiki_undesirable_chars), "", candid

                candidate_results.append(candidate)

            # If we found that one of the candidates is an incumbent, the others are s
            max_incumbent = max([x['is_incumbent'] for x in candidate_rows])
            #print(type(max_incumbent))
            if max_incumbent == 1:
                for candidate in candidate_rows:
                    candidate['is_incumbent'] = 0 if candidate['is_incumbent'] != 1 els

    return candidate_results

districts_list = get_district_list()
# districts_df = pd.DataFrame(districts_list)
# display(districts_df.loc[districts_df['state'] == 'Wyoming'])

wiki_house_history = get_wiki_districts_house_results(districts_list)
```

```
        # Store in disk
        wiki_house_history_df = pd.DataFrame(wiki_house_history)
        display(wiki_house_history_df)
        # wiki_house_history_df.to_csv('data/wikipedia/house_results.csv', encoding='utf-8')
```

```
Will get results for house/Wisconsin/District 2.html
Source: https://en.wikipedia.org/wiki/Wisconsin%27s_2nd_congressional_district


Empty DataFrame
Columns: []
Index: []
```

We now have on disk ALL the available historical district results from Wikipedia
Lets take a look:

```
In [603]: test_df = pd.read_csv('data/wikipedia/house_results.csv', index_col=0)
          display(test_df.head())
```

```
      district  is_incumbent              name party  percent  state  \
0  District 19           NaN  Randy Neugebauer     R     85.0  Texas
1  District 19           NaN  Randy Neugebauer     R     78.0  Texas
2  District 19           NaN       Andy Wilson     D     19.0  Texas
3  District 19           NaN  Randy Neugebauer     R     72.0  Texas
4  District 19           NaN  Dwight Fullingim     D     25.0  Texas

      votes  won  year
0  160136.0    1  2012
1  106059.0    1  2010
2   25984.0    0  2010
3  168501.0    1  2008
4   58030.0    0  2008
```

1.2 ballotpedia.org

So far so good but the 2018 results are missing on Wikipedia and the available data are not always exhaustives. So I decided to get the same informations from a different source: Ballotpedia.
Here we have the complete 2018 results as well as historical date from 2012.
Note that the incumbent information is consistent.

1.2.1 Get the House and Senate election result pages for all the available years

```
In [ ]: def get_house_senate_state_list():
            house_state_list = []
            senate_state_list = []
            filename = 'data/ballotpedia/house_state_list_src.html'
            url = 'https://ballotpedia.org/U.S._House_battlegrounds,_2018'
```

```
    # Check if the page has been stored on disk
    soup = check_or_save_page(filename, url)

    # Find the list of the U.S. Senate Elections by State (2018) pages
    table = soup.find('table', { 'class': 'infobox' })
    for link in table.find_all('a', href=lambda x: x and '/United_States_Senate_electi
        senate_state_list.append({
            'state': link.text,
            'url': 'https://ballotpedia.org{}'.format(link['href'])
        })

    # Find the list of the U.S. House Elections by State (2018) pages
    table = soup.find('table', { 'class': 'infobox' })
    for link in table.find_all('a', href=lambda x: x and (
        '/United_States_House_of_Representatives_election_in_' in x or
        '/United_States_House_of_Representatives_elections_in_' in x
    )):
        house_state_list.append({
            'state': link.text,
            'url': 'https://ballotpedia.org{}'.format(link['href'])
        })

    return house_state_list, senate_state_list

def get_district_pages(dict_page_url, year, state, district):
    """
        Recursively get all available previous election result pages
        for a given district
    """
    print('Will get house/{}/{}/{}.html'.format(state, district, year))
    # Create the directories if necessary
    if not os.path.exists('data/ballotpedia/house/'):
        os.makedirs('data/ballotpedia/house/')
    if not os.path.exists('data/ballotpedia/house/{}'.format(state)):
        os.makedirs('data/ballotpedia/house/{}'.format(state))
    if not os.path.exists('data/ballotpedia/house/{}/{}'.format(state, district)):
        os.makedirs('data/ballotpedia/house/{}/{}'.format(state, district))

    filename = 'data/ballotpedia/house/{}/{}/{}.html'.format(state, district, year)
    dict_soup = check_or_save_page(filename, dict_page_url)

    # Check if there is a link to a previous electoral year for this state
    table = dict_soup.find('table', { 'class': 'infobox' })
    div = table.find('div', style=lambda x: x and '#A3B1BF' in x and 'float:left;' in

    # If there is one
    if div is not None:
        # Extract the link election year
        prev_year = int(re.match(r'.*([1-2][0-9]{3})', div.text).group(1))

        if prev_year < year:
            # Get the link to this disctict House election results parge
            link = div.find('a')
            #print(link['href'])

            # Get this page
            url = 'https://ballotpedia.org{}'.format(link['href'])
            get_district_pages(url, prev_year, state, district)

def get_house_senate_state_districts_list(house_state_list):
    start_year = 2018
    state_district_list = []
    for house_state in house_state_list:
        # To remove
        #if house_state['state'] != 'Maryland':
        #    continue

        filename = 'data/ballotpedia/2018_house_{}.html'.format(house_state['state'])

        # Check if the page has been stored on disk
        soup = check_or_save_page(filename, house_state['url'])
        #print(soup)

        # Get the district page links
        table = soup.find('table', { 'class': 'infobox' })

        links = table.find_all('a', href=lambda x: x and (
            '_Congressional_District_election,_' in x
        ))

        if len(links) == 0:
            title = soup.find('b', text=lambda x : x and 'District Pages' in x)
            links = title.parent.parent.find_all('a', href=lambda x: x and (
                '_Congressional_District_election,_' in x
            ))

        for link in links:
            print(link.text)
            url = 'https://ballotpedia.org{}'.format(link['href'])
            state_district_list.append({
                'state': house_state['state'],
                'district': link.text
            })
            #print(' |-', url)
```

```
                # Get the page
                get_district_pages(url, start_year, house_state['state'], link.text)

        return state_district_list

    house_state_list, senate_state_list = get_house_senate_state_list()
    state_district_list = get_house_senate_state_districts_list(house_state_list)
```

1.2.2 Extract the House election results for every districts and years

```
In [584]: def extract_district_data(state_district_list):
              results = []
              undesirable_chars = ['\*', '%', 'Incumbent', '\(D\)', '\(R\)']
              for item in state_district_list:
                  # To remove
                  #if item['state'] != 'New Hampshire' or item['district'] != 'District 2':
                  #if item['state'] != 'Wyoming':
                  #    continue

                  # Get the pages
                  directory = 'data/ballotpedia/house/{}/{}'.format(item['state'], item['distr
                  files = [f for f in listdir(directory) if isfile(join(directory, f))]

                  # For each year, get the district data
                  for file in files:
                      # Extract the year
                      year = int(re.match(r'.*([1-2][0-9]{3})', file).group(1))
                      candidate_rows = []

                      # To remove
                      #if year != 2018:
                      #    continue

                      # Get the page content
                      filename = 'data/ballotpedia/house/{}/{}/{}'.format(item['state'], item[
                      with open(filename) as my_file:
                          soup = BeautifulSoup(my_file.read(), "html.parser")

                      # The 2018 pages requires a different approach
                      if year == 2018:
                          #print(2018)
                          # Find the result table
                          table = soup.find('table',  { 'class': 'results_table' })
                          rows = table.find_all('tr')

                          for row in rows:
                              cols = row.find_all('td')
                              cols = [ele.text.strip() for ele in cols]
```

```
        cols = [ele for ele in cols if ele] # Get rid of empty values

        # Check if is incumbant
        incumbent = 1 if row.find('b') and row.find('b').find('u') else
        #print(cols)
        if len(cols) == 4 and cols[0] == '':
            is_winner = 1
            name = cols[1] +' Incumbent' if incumbent == 1 else cols[1]
            percent = cols[2] if len(cols) > 1 else np.NaN
            votes = cols[3] if len(cols) > 2 else np.NaN
            party = 'Democratic' if '(D)' in cols[1] else 'Republican'
            candidate_rows.append([party, name, percent, votes, is_winne

        elif len(cols) == 3 and '(D)' in cols[0] or '(R)' in cols[0]:
            is_winner = 0
            name = cols[0] +' Incumbent' if incumbent == 1 else cols[0]
            percent = cols[1] if len(cols) > 1 else np.NaN

            votes = cols[2] if len(cols) > 2 else np.NaN
            if len(cols) > 1:
                party = 'Democratic' if '(D)' in cols[0] else 'Republica
            else:
                party = np.NaN

            candidate_rows.append([party, name, percent, votes, is_winne

else:
    # Find the result table
    th = soup.find('th', colspan='5', style=lambda x: x and 'background-
    table = th.find_parent('table')
    #table_body = table.find('tbody')
    rows = table.find_all('tr')
    #print(rows)

    for row in rows:
        cols = row.find_all('td')
        cols = [ele.text.strip() for ele in cols]
        cols = [ele for ele in cols if ele] # Get rid of empty values

        # Ignore the rows not about the candidates
        if 'Republican' not in cols and not 'Democratic' in cols:
            continue

        # Check if the candidate won the elections
        is_winner = 1 if row.find('a', title="Won") else 0
        cols.append(is_winner)
        candidate_rows.append(cols)
```

```
            # If there was only one candidate
            if len(candidate_rows) == 1:
                if type(candidate_rows[0][3]) is int:
                    candidate_rows[0].append(candidate_rows[0][3])
                    candidate_rows[0][3] = np.NaN

            for candidate in candidate_rows:
                #print(year, item['district'], candidate)

                # Get and format the candidate party
                candidate_party = 'R' if candidate[0] == 'Republican' else 'D'

                # Get and clean the candidate name
                candidate_name = re.sub("|".join(undesirable_chars), "", candidate[1

                # Get and clean the candidate percent
                if type(candidate[2]) is str:
                    candidate_percent = float(candidate[2].replace('%', ''))
                else:
                    candidate_percent = candidate[2]

                # Get and clean the candidate vote
                if type(candidate[3]) is str:
                    candidate_vote = int(candidate[3].replace(',', ''))
                else:
                    candidate_vote = candidate[3]

                # Determine whether or not the candidate is incumbent
                candidate_is_incumbent = 1 if 'Incumbent' in candidate[1] else 0

                results.append({
                    'year': year,
                    'state': item['state'],
                    'district': item['district'] if item['district'] != 'General ele
                    'name': candidate_name,
                    'party': candidate_party,
                    'percent': candidate_percent,
                    'votes': candidate_vote,
                    'is_incumbent': candidate_is_incumbent,
                    'won': candidate[4]
                })

                #print(results)
                #print('')

        #soup = BeautifulSoup(my_file.read(), "html.parser")
        #print(soup)
    return results
```

```
ballo_house_history = extract_district_data(state_district_list)

# Store on disk
ballo_house_history_df = pd.DataFrame(ballo_house_history)
ballo_house_history_df.to_csv('data/ballotpedia/ballo_results.csv', encoding='utf-8'
```

1.3 Merge the data from wikipedia.org and ballotpedia.org

Now we have two dataset with the same columns and some overlaping data. Its time to merge them.
It appears that the data from ballotpedia.org are more consistent so we will favor them.

```
In [9]: ballo_df = pd.read_csv('data/ballotpedia/ballo_results.csv', index_col=0)
        wikipedia_df = pd.read_csv('data/wikipedia/house_results.csv', index_col=0)

        merged_df =  pd.concat([wikipedia_df.loc[wikipedia_df['year'] < 2012], ballo_df])
        display(merged_df.head())
```

```
      district  is_incumbent              name party  percent  state  \
1  District 19           NaN  Randy Neugebauer     R     78.0  Texas
2  District 19           NaN       Andy Wilson     D     19.0  Texas
3  District 19           NaN  Randy Neugebauer     R     72.0  Texas
4  District 19           NaN  Dwight Fullingim     D     25.0  Texas
5  District 19           NaN  Randy Neugebauer     R     68.0  Texas

      votes  won  year
1  106059.0    1  2010
2   25984.0    0  2010
3  168501.0    1  2008
4   58030.0    0  2008
5   92811.0    1  2006
```

1.4 Imput and derive from the data

From this data, we can create new predictors:

- Impute the missing data for is_incumbent
- First time the incumbent has been elected
- Number of incumbents victories

```
In [10]: derived_df = merged_df.copy()
         derived_df = derived_df.sort_values(by=['year'])

         # Number of NaN values for the `is_incumbent` col
         print('NaN is_incumbent values before:', derived_df['is_incumbent'].isna().sum())

         def check_if_is_incumbent(row):
             is_incumbent = row['is_incumbent']

             # Check if there is a previous election for this state, district and candidate
             prev_year = row['year'] - 2
             prev_year_row = derived_df.loc[(derived_df['state'] == row['state']) & (derived_di

             # If the row has NaN for the col `is_incumbent` and the candidate won the last el
             if np.isnan(row['is_incumbent']) and prev_year_row.empty is False and prev_year_ro
                 is_incumbent = 1
             # If the candidate lose the last elections, it is likely he isn't the incumbent
             elif np.isnan(row['is_incumbent']) and prev_year_row.empty is False and prev_year_
                 is_incumbent
             # If the candidate didn't participate to the last election,
             # we can safely assume he isn't the incumbent
             elif np.isnan(row['is_incumbent']) and prev_year_row.empty:
                 is_incumbent = 0

             return is_incumbent

         # Check if the candidate has already been elected the past year (and so is an incumbe
         derived_df['is_incumbent'] = derived_df.apply(check_if_is_incumbent, axis=1)

         print('NaN is_incumbent values after:', derived_df['is_incumbent'].isna().sum())
```

```
NaN is_incumbent values before: 2748
NaN is_incumbent values after: 118
```

We gain the is_incumbent information for 2630 rows. 118 remains NaN.
Now, we will add a new column to know the year of the first election the candidate won:

```
In [11]: def get_first_year_elected(row):
             first_elected = np.NaN
             # Get the first year the candidate has been elected (if exist)
             victories = derived_df.loc[(derived_df['state'] == row['state']) & (derived_df['di

             if victories.empty is False:
                 first_elected = victories['year'].min()

             return first_elected

         derived_df['first_time_elected'] = derived_df.apply(get_first_year_elected, axis=1)
```

Finally, we want to count the number of victories of each candidate:

```
In [12]: def count_victories(row):
             count_victories = 0
             victories = derived_df.loc[(derived_df['state'] == row['state']) & (derived_df['di
             if victories.empty is False:
                 count_victories = len(victories)

             return count_victories

         derived_df['count_victories'] = derived_df.apply(count_victories, axis=1)
```

```
In [13]: display(derived_df.head())
```

```
        district  is_incumbent          name party  percent    state    votes  \
1963  District 1           0.0  Ratliff Boon     D     42.1  Indiana   4281.0
1964  District 1           1.0  Ratliff Boon     D     42.8  Indiana   5202.0
1965  District 1           1.0  Ratliff Boon     D     52.2  Indiana   7272.0
1967  District 1           0.0      John Law     D     49.1  Indiana  10868.0
1966  District 1           1.0  Ratliff Boon     D     50.9  Indiana  11280.0

      won  year  first_time_elected  count_victories
1963    1  1824              1824.0                7
1964    1  1826              1824.0                7
1965    1  1828              1824.0                7
1967    0  1830              1860.0                2
1966    1  1830              1824.0                7
```

1.5 Additional factors

Let's also add the **unemployement rate** at the district level when available, else, at the national level.

```
In [14]: def get_district_number(row):
            district = row['district']

            p = re.compile("District (.*) \(")
            dist = p.search(row['district']).group(1)

            if dist == '(at Large)':
                district = 'At-Large'
            else:
                district = 'District {}'.format(dist)

            return district

        def get_state_dist_unemployement(row):
            unemployement_rate = np.NaN

            # Find the corresponding unemployement rate
            unemp_row = unemp_df.loc[(unemp_df['state'] == row['state']) & \
                                     (unemp_df['district'] == row['district']) & \
                                     (unemp_df['year'] == row['year'])]
            if unemp_row.empty is False:
                unemployement_rate = unemp_row['unemp_rate_16'].values[0]
            else:
                # Use the national unemployement rate of October instead
                nat_oct_unemployement_rate = national_unemployement_rate.loc[national_unemploy
                unemployement_rate = np.NaN if nat_oct_unemployement_rate.empty else nat_oct_u

            return unemployement_rate

        augmented_df = derived_df.copy()

        unemp_df = pd.read_csv('data/unemployment/unemp_2012_2017.csv',sep=';')
        unemp_df['state'] = unemp_df['state'].str.rsplit(',').str[-1].str.strip()
        unemp_df['district'] = unemp_df.apply(get_district_number, axis=1)

        augmented_df['unemployement_rate'] = augmented_df.apply(get_state_dist_unemployement,

        display(augmented_df.loc[augmented_df['state'] == 'Alabama'].head())
```

```
        district  is_incumbent                 name party  percent    state  \
889   District 1           0.0            Judy Belk     D     38.0  Alabama
888   District 1           0.0           Jo Bonner     R     61.0  Alabama
1223  District 4           0.0  Robert B. Aderholt     R     87.0  Alabama
8075  District 7           0.0          Artur Davis     D     92.3  Alabama
4929  District 5           0.0           Bud Cramer     D     73.0  Alabama

         votes  won  year  first_time_elected  count_victories  \
889    67507.0    0  2002                 NaN                0
888   108102.0    1  2002              2002.0                6
1223  139705.0    1  2002              2002.0                5
8075  153735.0    1  2002              2002.0                4
4929  143029.0    1  2002              2002.0                3

      unemployement_rate
889                  5.7
888                  5.7
1223                 5.7
8075                 5.7
4929                 5.7
```

We will also add the following national level factors:

- is_presidential_year: 1 if Yes, 0 if No
- president_can_be_re_elected: Can the president stand for re-election ? 1 = Yes, 0 = No

- president_party: D or R

- president_overall_avg_job_approval: Only available from 1953 to 2018
- last_D_house_seats: # of Democrat seats at the last elections
- last_R_house_seats: # of Republican seats at the last elections
- last_house_majority: Which party had the majority at the last House elections. D or R

```
In [18]: nat_augmented_df = augmented_df.copy()

         def add_is_presidential_year(row):
             result = np.NaN
             df = national_level_factors.loc[national_level_factors['year'] == row['year'], 'is

             if df.empty is False:
                 result = df.values[0]

             return result

         def add_president_can_be_re_elected(row):
             # idx = (np.abs(president_elected_history['year'].values-row['year']+1)).argmin()
             # return president_elected_history['can_be_re_elected'].loc[[idx]].values[0]
             result = np.NaN
             df = national_level_factors.loc[national_level_factors['year'] == row['year'], 'pr

             if df.empty is False:
                 result = df.values[0]

             return result

         def add_president_party(row):
             # idx = (np.abs(president_elected_history['year'].values-row['year']+1)).argmin()
             # return president_elected_history['president_elected_party'].loc[[idx]].values[0
             result = np.NaN
             df = national_level_factors.loc[national_level_factors['year'] == row['year'], 'pr

             if df.empty is False:
                 result = df.values[0]

             return result

         def add_president_overall_avg_job_approval(row):
             # idx = (np.abs(president_elected_history['year'].values-row['year']+1)).argmin()
             # president_name = president_elected_history['president_elected'].loc[[idx]].valu
             # president_overall_avg_job_approval = presidential_approval_df.loc[presidential_
             # return float(president_overall_avg_job_approval.values[0])/100 if president_ove
             result = np.NaN
             df = national_level_factors.loc[national_level_factors['year'] == row['year'], 'pr

             if df.empty is False:
                 result = df.values[0]
```

```
    return result

def add_last_D_house_seats(row):
    result = np.NaN
    df = national_level_factors.loc[national_level_factors['year'] == row['year'], 'la

    if df.empty is False:
        result = df.values[0]

    return result

def add_last_R_house_seats(row):
    result = np.NaN
    df = national_level_factors.loc[national_level_factors['year'] == row['year'], 'la

    if df.empty is False:
        result = df.values[0]

    return result

def add_last_house_majority(row):
    result = np.NaN
    df = national_level_factors.loc[national_level_factors['year'] == row['year'], 'la

    if df.empty is False:
        result = df.values[0]

    return result

# is_presidential_year
nat_augmented_df['is_presidential_year'] = nat_augmented_df.apply(add_is_presidential_

# president_can_be_re_elected
nat_augmented_df['president_can_be_re_elected'] = nat_augmented_df.apply(add_presiden

# president_party
nat_augmented_df['president_party'] = nat_augmented_df.apply(add_president_party, axis

# president_overall_avg_job_approval
nat_augmented_df['president_overall_avg_job_approval'] = nat_augmented_df.apply(add_pr

# last_D_house_seats
nat_augmented_df['last_D_house_seats'] = nat_augmented_df.apply(add_last_D_house_seats

# last_R_house_seats
nat_augmented_df['last_R_house_seats'] = nat_augmented_df.apply(add_last_R_house_seats
# last_house_majority
nat_augmented_df['last_house_majority'] = nat_augmented_df.apply(add_last_house_majori
```

```
In [19]: display(nat_augmented_df.loc[nat_augmented_df['year'] == 1958].head())
         #display(nat_augmented_df.head())
```

```
         district  is_incumbent             name party  percent       state  \
5838  District 15           1.0    Joe M. Kilgore     D   100.00       Texas
3347  District 19           0.0   Roy E. Reynolds     R    24.70  California
2080   District 1           1.0     Ray J. Madden     D    66.40     Indiana
6868   District 4           1.0   Wayne Aspinall      D    63.61    Colorado
1319  District 17           1.0    Cecil R. King      D    75.30  California

         votes  won  year  first_time_elected  count_victories  \
5838   28404.0    1  1958              1956.0                4
3347   26092.0    0  1958                 NaN                0
2080   95801.0    1  1958              1942.0               17
6868   43785.0    1  1958              1950.0               11
1319  182965.0    1  1958              1942.0               13

      unemployement_rate  is_presidential_year  president_can_be_re_elected  \
5838                 6.7                   0.0                          0.0
3347                 6.7                   0.0                          0.0
2080                 6.7                   0.0                          0.0
6868                 6.7                   0.0                          0.0
1319                 6.7                   0.0                          0.0

     president_party  president_overall_avg_job_approval  last_D_house_seats  \
5838               R                                0.65               232.0
3347               R                                0.65               232.0
2080               R                                0.65               232.0
6868               R                                0.65               232.0
1319               R                                0.65               232.0

      last_R_house_seats last_house_majority
5838               203.0                   D
3347               203.0                   D
2080               203.0                   D
6868               203.0                   D
1319               203.0                   D
```

Finally, we add the fundraising data we got on followthemoney.org (from 2009 to 2018). The candidate names are not formated the same way as our data from wikipedia and ballotpedia so we will use a fuzzy search algorithm to match them.

```
In [42]: # You need to install the fuzzywuzzy package
         # Source: https://github.com/seatgeek/fuzzywuzzy
         from fuzzywuzzy import fuzz
         from fuzzywuzzy import process

         fundraising_df = pd.read_csv('data/Fundraising/followthemoney_2009-2018.csv')
         # display(fundraising_df.head())

         def add_fundraising(row):
             fundraising = np.NaN

             if row['year'] > 1991:
                 # Get the code ISO 2 of the state
                 iso2 = state_to_iso2[row['state']]
                 # Find the corresponding rows for this year and state from the fundraising da
                 fundraising_rows = fundraising_df.loc[(fundraising_df['Election_Year'] == row
                                   (fundraising_df['Election_Jurisdiction'] == iso2)]
                 # Try to match with the candidate name
                 for index, line in fundraising_rows.iterrows():
                     # Get the district number
                     district_numb = int([int(s) for s in line['Office_Sought'].split() if s.i

                     if 'District {}'.format(district_numb) != row['district']:
                         continue

                     ratio = fuzz.token_sort_ratio(row['name'], line['Candidate'].lower())

                     if ratio > 79:
                         fundraising = line['Total_$']

             return fundraising

         nat_augmented_df['fundraising'] =  nat_augmented_df.apply(add_fundraising, axis=1)
```

```
In [47]: display(nat_augmented_df.loc[nat_augmented_df['year'] > 2017].head())
```

```
         district  is_incumbent             name party  percent       state  \
357   District 28           1.0      Adam Schiff     D     76.4  California
3003   District 2           0.0  Shireen Ghorbani     D     38.4        Utah
380   District 31           1.0     Pete Aguilar     D     56.1  California
2979  District 35           1.0    Lloyd Doggett     D     71.2       Texas
2986  District 36           1.0      Brian Babin     R     72.6       Texas

         votes  won  year  first_time_elected  count_victories  \
357   127153.0    1  2018              2012.0                4
3003   84011.0    0  2018                 NaN                0
380    61747.0    1  2018              2014.0                3
2979  137325.0    1  2018              2012.0                4
2986  160592.0    1  2018              2014.0                3

      unemployement_rate  is_presidential_year  president_can_be_re_elected  \
357                   3.7                    0.0                          1.0
3003                  3.7                    0.0                          1.0
380                   3.7                    0.0                          1.0
2979                  3.7                    0.0                          1.0
2986                  3.7                    0.0                          1.0

     president_party  president_overall_avg_job_approval  last_D_house_seats  \
357                R                               0.395               194.0
3003               R                               0.395               194.0
380                R                               0.395               194.0
2979               R                               0.395               194.0
2986               R                               0.395               194.0

      last_R_house_seats last_house_majority  fundraising
357                241.0                   R   4606209.66
3003               241.0                   R    109604.65
380                241.0                   R   2046673.91
2979               241.0                   R    543906.56
2986               241.0                   R    891968.14
```

2 Store the final dataset on disk

```
In [49]: nat_augmented_df.to_csv('data/ready_to_use_dataset.csv', index=False)
```

3 Manually fix errors and add some missing results by hand

In some edge-cases, the crawler didn't do a good job or the data was particularly messy, resulting in more than 1 winner for an election, 0 winner or duplicate candidates. Here, we fix have to manually check every of this occurences and fix them.

```
In [50]: house_df = pd.read_csv('data/ready_to_use_dataset.csv')
         house_df.loc[(house_df['year'] == 1858) & \
                      (house_df['state'] == 'Indiana') & \
                      (house_df['district'] == 'District 1') &\
                      (house_df['party'] == 'D'), 'won'] = 1

         house_df.loc[(house_df['year'] == 1876) & \
                      (house_df['state'] == 'California') & \
                      (house_df['district'] == 'District 4') &\
                      (house_df['party'] == 'R'), 'won'] = 1

         house_df.loc[(house_df['year'] == 1878) & \
                      (house_df['state'] == 'Indiana') & \
                      (house_df['district'] == 'District 1') &\
                      (house_df['party'] == 'D'), 'won'] = 0
```

```
house_df.loc[(house_df['year'] == 1876) & \
             (house_df['state'] == 'California') & \
             (house_df['district'] == 'District 4') &\
             (house_df['party'] == 'D'), 'won'] = 0

house_df.loc[(house_df['year'] == 1888) & \
             (house_df['state'] == 'Indiana') & \
             (house_df['district'] == 'District 1') &\
             (house_df['party'] == 'R'), 'won'] = 0

house_df = house_df.drop(house_df.loc[(house_df['year'] == 1930) & \
             (house_df['state'] == 'Utah') & \
             (house_df['district'] == 'District 2') &\
             (house_df['percent'] == 44.13)].index)
house_df = house_df.drop(house_df.loc[(house_df['year'] == 1930) & \
             (house_df['state'] == 'Utah') & \
             (house_df['district'] == 'District 2') &\
             (house_df['percent'] == 42.34)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 1940) & \
             (house_df['state'] == 'Colorado') & \
             (house_df['district'] == 'District 3') &\
             (house_df['name'] == 'William E. Burney')].index)
house_df = house_df.drop(house_df.loc[(house_df['year'] == 1940) & \
             (house_df['state'] == 'Colorado') & \
             (house_df['district'] == 'District 3') &\
             (house_df['name'] == 'Henry Leonard')].index)

house_df.loc[(house_df['year'] == 1956) & \
             (house_df['state'] == 'Colorado') & \
             (house_df['district'] == 'District 3') &\
             (house_df['party'] == 'D'), 'won'] = 0

house_df.loc[(house_df['year'] == 1958) & \
             (house_df['state'] == 'Colorado') & \
             (house_df['district'] == 'District 3') &\
             (house_df['party'] == 'D'), 'won'] = 0

house_df = house_df.drop(house_df.loc[(house_df['year'] == 1966) & \
             (house_df['state'] == 'California') & \
             (house_df['district'] == 'District 14') &\
             (house_df['percent'] != 56.4) &\
             (house_df['percent'] != 43.6)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 1970) & \
             (house_df['state'] == 'California') & \
             (house_df['district'] == 'District 24') &\
             (house_df['percent'] != 65.1) &\
             (house_df['percent'] != 32.4)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 1970) & \
             (house_df['state'] == 'California') & \
             (house_df['district'] == 'District 35') &\
             (house_df['percent'] != 67.0) &\
             (house_df['percent'] != 30.3)].index)

house_df = house_df.append({'district': 'District 6', 'is_incumbent': 0.0, 'name': 'Ri
                            'party': 'D', 'percent': np.NaN, 'state': 'Minnesota', 'vo
                            'won': 1, 'year': 1972, 'first_time_elected': np.NaN, 'cou
                            'unemployement_rate': 5.6, 'is_presidential_year': 1.0, \
                            'president_can_be_re_elected': 0.0, 'president_party': 'R'
                            'president_overall_avg_job_approval': 0.49, 'last_D_house_
                            'last_R_house_seats': 180.0, 'last_house_majority': 'D', '
                           ignore_index=True)
house_df.loc[(house_df['year'] == 1972) & \
             (house_df['state'] == 'Minnesota') & \
             (house_df['district'] == 'District 6') &\
             (house_df['name'] == 'John M. Zwach {incumbent}'), 'is_incumbent'] = 1.0
house_df.loc[(house_df['year'] == 1972) & \
             (house_df['state'] == 'Minnesota') & \
             (house_df['district'] == 'District 6') &\
             (house_df['name'] == 'John M. Zwach {incumbent}'), 'name'] = 'John M. Zwa

house_df = house_df.drop(house_df.loc[(house_df['year'] == 1974) & \
             (house_df['state'] == 'California') & \
             (house_df['district'] == 'District 13') &\
             (house_df['percent'] != 52.6) &\
             (house_df['percent'] != 42.4)].index)

house_df = house_df.append({'district': 'District 6', 'is_incumbent': 1.0, 'name': 'Ri
                            'party': 'D', 'percent': 55.4, 'state': 'Minnesota', 'vote
                            'won': 1, 'year': 1974, 'first_time_elected': 1972, 'count
                            'unemployement_rate': 6.0, 'is_presidential_year': 0.0, \
                            'president_can_be_re_elected': 0.0, 'president_party': 'R'
                            'president_overall_avg_job_approval': 0.49, 'last_D_house_
                            'last_R_house_seats': 192.0, 'last_house_majority': 'D', '
                           ignore_index=True)

house_df = house_df.append({'district': 'District 6', 'is_incumbent': 1.0, 'name': 'Ri
                            'party': 'D', 'percent': 59.6, 'state': 'Minnesota', 'vote
                            'won': 1, 'year': 1976, 'first_time_elected': 1972, 'count
                            'unemployement_rate': 7.7, 'is_presidential_year': 1.0, \
                            'president_can_be_re_elected': 1.0, 'president_party': 'D'
                            'president_overall_avg_job_approval': 0.455, 'last_D_house
                            'last_R_house_seats': 144.0, 'last_house_majority': 'D', '
                           ignore_index=True)
```

```
house_df.loc[(house_df['year'] == 1978) & \
             (house_df['state'] == 'Colorado') & \
             (house_df['district'] == 'District 3') &\
             (house_df['party'] == 'R'), 'won'] = 0

house_df = house_df.drop(house_df.loc[(house_df['year'] == 1982) & \
             (house_df['state'] == 'California') & \
             (house_df['district'] == 'District 30') &\
             (house_df['percent'] != 53.9) &\
             (house_df['percent'] != 46.1)].index)

house_df = house_df.append({'district': 'District 6', 'is_incumbent': 1.0, 'name': 'G
                           'party': 'D', 'percent': np.NaN, 'state': 'Minnesota', 'vo
                           'won': 1, 'year': 1984, 'first_time_elected': 1982, 'count
                           'unemployement_rate': 7.4, 'is_presidential_year': 1.0, \
                           'president_can_be_re_elected': 0.0, 'president_party': 'R'
                           'president_overall_avg_job_approval': 0.528, 'last_D_house
                           'last_R_house_seats': 166.0, 'last_house_majority': 'D', '
                          ignore_index=True)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 1986) & \
             (house_df['state'] == 'Hawaii') & \
             (house_df['district'] == 'District 1') &\
             (house_df['percent'] != 59.20) &\
             (house_df['percent'] != 37.45)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 1990) & \
             (house_df['state'] == 'Hawaii') & \
             (house_df['district'] == 'District 2') &\
             (house_df['percent'] != 66.27) &\
             (house_df['percent'] != 30.64)].index)

house_df.loc[(house_df['year'] == 1994) & \
             (house_df['state'] == 'Connecticut') & \
             (house_df['district'] == 'District 2') &\
             (house_df['party'] == 'R'), 'won'] = 0

house_df = house_df.drop(house_df.loc[(house_df['year'] == 1996) & \
             (house_df['state'] == 'Colorado') & \
             (house_df['district'] == 'District 5') &\
             (house_df['percent'] != 71.94) &\
             (house_df['percent'] != 28.06)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 1996) & \
             (house_df['state'] == 'Oregon') & \
             (house_df['district'] == 'District 3') &\
             (house_df['percent'] != 66.93) &\
             (house_df['percent'] != 26.32)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 1998) & \
             (house_df['state'] == 'New Mexico') & \
             (house_df['district'] == 'District 1') &\
             (house_df['percent'] != 48.44) &\
             (house_df['percent'] != 41.88)].index)

house_df.loc[(house_df['year'] == 2002) & \
             (house_df['state'] == 'Colorado') & \
             (house_df['district'] == 'District 7') &\
             (house_df['party'] == 'D'), 'won'] = 0

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2002) & \
             (house_df['state'] == 'Hawaii') & \
             (house_df['district'] == 'District 2') &\
             (house_df['percent'] != 56.16) &\
             (house_df['percent'] != 39.98)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2006) & \
             (house_df['state'] == 'California') & \
             (house_df['district'] == 'District 50') &\
             (house_df['percent'] != 53.2) &\
             (house_df['percent'] != 43.5)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2006) & \
             (house_df['state'] == 'Colorado') & \
             (house_df['district'] == 'District 5') &\
             (house_df['percent'] != 59.62) &\
             (house_df['percent'] != 40.35)].index)

house_df.loc[(house_df['year'] == 2006) & \
             (house_df['state'] == 'Connecticut') & \
             (house_df['district'] == 'District 2') &\
             (house_df['party'] == 'R'), 'won'] = 0

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2006) & \
             (house_df['state'] == 'Louisiana') & \
             (house_df['district'] == 'District 2') &\
             (house_df['percent'] != 56.55) &\
             (house_df['percent'] != 43.45)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2006) & \
             (house_df['state'] == 'Mississippi') & \
             (house_df['district'] == 'District 4') &\
             (house_df['percent'] != 79.79) &\
             (house_df['percent'] != 20.21)].index)
```

```
house_df.loc[(house_df['year'] == 2006) & \
             (house_df['state'] == 'Pennsylvania') & \
             (house_df['district'] == 'District 8') &\
             (house_df['party'] == 'R'), 'won'] = 0

house_df.loc[(house_df['year'] == 2008) & \
             (house_df['state'] == 'Alabama') & \
             (house_df['district'] == 'District 2') &\
             (house_df['party'] == 'R'), 'won'] = 0

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2008) & \
             (house_df['state'] == 'California') & \
             (house_df['district'] == 'District 12') &\
             (house_df['percent'] != 75.2) &\
             (house_df['percent'] != 18.5)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2008) & \
             (house_df['state'] == 'Colorado') & \
             (house_df['district'] == 'District 5') &\
             (house_df['percent'] != 60.0) &\
             (house_df['percent'] != 37.0)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2008) & \
             (house_df['state'] == 'Colorado') & \
             (house_df['district'] == 'District 6') &\
             (house_df['percent'] != 61.0) &\
             (house_df['percent'] != 39.0)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2008) & \
             (house_df['state'] == 'Maryland') & \
             (house_df['district'] == 'District 4') &\
             (house_df['percent'] != 85.83) &\
             (house_df['percent'] != 12.85)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2008) & \
             (house_df['state'] == 'Massachusetts') & \
             (house_df['district'] == 'District 1') &\
             (house_df['name'] == 'Robert Feuer')].index)
house_df.loc[(house_df['year'] == 2008) & \
             (house_df['state'] == 'Massachusetts') & \
             (house_df['district'] == 'District 1') &\
             (house_df['party'] == 'D'), 'won'] = 1

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2008) & \
             (house_df['state'] == 'New Mexico') & \
             (house_df['district'] == 'District 1') &\
             (house_df['percent'] != 55.65) &\
             (house_df['percent'] != 44.35)].index)
```

```
house_df = house_df.drop(house_df.loc[(house_df['year'] == 2008) & \
            (house_df['state'] == 'New Mexico') & \
            (house_df['district'] == 'District 3') &\
            (house_df['percent'] != 56.74) &\
            (house_df['percent'] != 30.47)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2010) & \
            (house_df['state'] == 'Alabama') & \
            (house_df['district'] == 'District 5') &\
            (house_df['percent'] != 58.0) &\
            (house_df['percent'] != 42.0)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2010) & \
            (house_df['state'] == 'California') & \
            (house_df['district'] == 'District 42') &\
            (house_df['percent'] != 62.0) &\
            (house_df['percent'] != 32.0)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2010) & \
            (house_df['state'] == 'Hawaii') & \
            (house_df['district'] == 'District 1') &\
            (house_df['percent'] != 53.23) &\
            (house_df['percent'] != 46.77)].index)

house_df = house_df.drop(house_df.loc[(house_df['year'] == 2010) & \
            (house_df['state'] == 'Texas') & \
            (house_df['district'] == 'District 14') &\
            (house_df['percent'] != 76.0) &\
            (house_df['percent'] != 24.0)].index)

house_df.loc[(house_df['year'] == 2018) & \
            (house_df['state'] == 'Arkansas') & \
            (house_df['district'] == 'District 4') &\
            (house_df['party'] == 'R'), 'won'] = 1

house_df.loc[(house_df['year'] == 2018) & \
            (house_df['state'] == 'California') & \
            (house_df['district'] == 'District 10') &\
            (house_df['party'] == 'R'), 'won'] = 1

house_df.loc[(house_df['year'] == 2018) & \
            (house_df['state'] == 'California') & \
            (house_df['district'] == 'District 39') &\
            (house_df['party'] == 'R'), 'won'] = 1

house_df.loc[(house_df['year'] == 2018) & \
            (house_df['state'] == 'California') & \
            (house_df['district'] == 'District 45') &\
            (house_df['party'] == 'R'), 'won'] = 1

house_df.loc[(house_df['year'] == 2018) & \
            (house_df['state'] == 'Florida') & \
            (house_df['district'] == 'District 8') &\
            (house_df['party'] == 'R'), 'won'] = 1

house_df.loc[(house_df['year'] == 2018) & \
            (house_df['state'] == 'Georgia') & \
            (house_df['district'] == 'District 4') &\
            (house_df['party'] == 'D'), 'won'] = 1

house_df.loc[(house_df['year'] == 2018) & \
            (house_df['state'] == 'Georgia') & \
            (house_df['district'] == 'District 7') &\
            (house_df['party'] == 'R'), 'won'] = 1

house_df.loc[(house_df['year'] == 2018) & \
            (house_df['state'] == 'Maine') & \
            (house_df['district'] == 'District 2') &\
            (house_df['party'] == 'R'), 'won'] = 1

house_df.loc[(house_df['year'] == 2018) & \
            (house_df['state'] == 'Minnesota') & \
            (house_df['district'] == 'District 1') &\
            (house_df['party'] == 'R'), 'won'] = 1

house_df.loc[(house_df['year'] == 2018) & \
            (house_df['state'] == 'New Hampshire') & \
            (house_df['district'] == 'District 2') &\
            (house_df['name'] == 'Steve Negron'), 'party'] = 'R'

house_df = house_df.append({'district': 'District 2', 'is_incumbent': 1.0, 'name': 'A
                            'party': 'D', 'percent': np.NaN, 'state': 'New Hampshire',
                            'won': 1, 'year': 2018, 'first_time_elected': 2012, 'count
                            'unemployement_rate': 3.7, 'is_presidential_year': 0.0, \
                            'president_can_be_re_elected': 1.0, 'president_party': 'R'
                            'president_overall_avg_job_approval': 0.395, 'last_D_house
                            'last_R_house_seats': 241.0, 'last_house_majority': 'R', '
                           ignore_index=True)

house_df.loc[(house_df['year'] == 2018) & \
            (house_df['state'] == 'New Jersey') & \
            (house_df['district'] == 'District 3') &\
            (house_df['name'] == 'Tom MacArthur'), 'party'] = 'R'
house_df.loc[(house_df['year'] == 2018) & \
            (house_df['state'] == 'New Jersey') & \
             (house_df['district'] == 'District 3') &\
             (house_df['name'] == 'Tom MacArthur'), 'percent'] = 48.8
house_df.loc[(house_df['year'] == 2018) & \
             (house_df['state'] == 'New Jersey') & \
             (house_df['district'] == 'District 3') &\
             (house_df['name'] == 'Tom MacArthur'), 'votes'] = 147036
house_df = house_df.append({'district': 'District 3', 'is_incumbent': 0.0, 'name': 'An
                            'party': 'D', 'percent': 49.9, 'state': 'New Jersey', 'vote
                            'won': 1, 'year': 2018, 'first_time_elected': np.NaN, 'cour
                            'unemployement_rate': 3.7, 'is_presidential_year': 0.0, \
                            'president_can_be_re_elected': 1.0, 'president_party': 'R'
                            'president_overall_avg_job_approval': 0.395, 'last_D_house
                            'last_R_house_seats': 241.0, 'last_house_majority': 'R', 'i
                           ignore_index=True)

house_df.loc[(house_df['year'] == 2018) & \
             (house_df['state'] == 'New York') & \
             (house_df['district'] == 'District 22') &\
             (house_df['party'] == 'D'), 'won'] = 1

house_df.loc[(house_df['year'] == 2018) & \
             (house_df['state'] == 'Utah') & \
             (house_df['district'] == 'District 4') &\
             (house_df['party'] == 'D'), 'won'] = 1

house_df = house_df.append({'district': 'District 2', 'is_incumbent': 1.0, 'name': 'Ma
                            'party': 'D', 'percent': 97.5, 'state': 'Wisconsin', 'votes
                            'won': 1, 'year': 2018, 'first_time_elected': 2012, 'count
                            'unemployement_rate': 3.7, 'is_presidential_year': 0.0, \
                            'president_can_be_re_elected': 1.0, 'president_party': 'R'
                            'president_overall_avg_job_approval': 0.395, 'last_D_house
                            'last_R_house_seats': 241.0, 'last_house_majority': 'R', 'i
                           ignore_index=True)

house_df = house_df.append({'district': 'District 7', 'is_incumbent': 1.0, 'name': 'Te
                            'party': 'D', 'percent': 100, 'state': 'Alabama', 'votes':
                            'won': 1, 'year': 2018, 'first_time_elected': 2010, 'count
                            'unemployement_rate': 3.7, 'is_presidential_year': 0.0, \
                            'president_can_be_re_elected': 1.0, 'president_party': 'R'
                            'president_overall_avg_job_approval': 0.395, 'last_D_house
                            'last_R_house_seats': 241.0, 'last_house_majority': 'R', 'i
                           ignore_index=True)

house_df = house_df.append({'district': 'District 5', 'is_incumbent': 1.0, 'name': 'Jo
                            'party': 'D', 'percent': 100, 'state': 'Georgia', 'votes':
                            'won': 1, 'year': 2018, 'first_time_elected': 2002, 'count
                            'unemployement_rate': 3.7, 'is_presidential_year': 0.0, \
                            'president_can_be_re_elected': 1.0, 'president_party': 'R'
```

```
                                 'president_overall_avg_job_approval': 0.395, 'last_D_house_
                                 'last_R_house_seats': 241.0, 'last_house_majority': 'R', ':
                                ignore_index=True)

house_df = house_df.append({'district': 'District 24', 'is_incumbent': 1.0, 'name': 'I
                                 'party': 'D', 'percent': 100, 'state': 'Florida', 'votes':
                                 'won': 1, 'year': 2018, 'first_time_elected': 2012, 'count_
                                 'unemployement_rate': 3.7, 'is_presidential_year': 0.0, \
                                 'president_can_be_re_elected': 1.0, 'president_party': 'R'
                                 'president_overall_avg_job_approval': 0.395, 'last_D_house_
                                 'last_R_house_seats': 241.0, 'last_house_majority': 'R', ':
                                ignore_index=True)

house_df = house_df.append({'district': 'District 21', 'is_incumbent': 1.0, 'name': 'I
                                 'party': 'D', 'percent': 100, 'state': 'Florida', 'votes':
                                 'won': 1, 'year': 2018, 'first_time_elected': 2016, 'count_
                                 'unemployement_rate': 3.7, 'is_presidential_year': 0.0, \
                                 'president_can_be_re_elected': 1.0, 'president_party': 'R'
                                 'president_overall_avg_job_approval': 0.395, 'last_D_house_
                                 'last_R_house_seats': 241.0, 'last_house_majority': 'R', ':
                                ignore_index=True)

house_df = house_df.append({'district': 'District 14', 'is_incumbent': 1.0, 'name': 'I
                                 'party': 'D', 'percent': 100, 'state': 'Florida', 'votes':
                                 'won': 1, 'year': 2018, 'first_time_elected': 2010, 'count_
                                 'unemployement_rate': 3.7, 'is_presidential_year': 0.0, \
                                 'president_can_be_re_elected': 1.0, 'president_party': 'R'
                                 'president_overall_avg_job_approval': 0.395, 'last_D_house_
                                 'last_R_house_seats': 241.0, 'last_house_majority': 'R', ':
                                ignore_index=True)

house_df = house_df.append({'district': 'District 10', 'is_incumbent': 1.0, 'name': 'V
                                 'party': 'D', 'percent': 100, 'state': 'Florida', 'votes':
                                 'won': 1, 'year': 2018, 'first_time_elected': 2016, 'count_
                                 'unemployement_rate': 3.7, 'is_presidential_year': 0.0, \
                                 'president_can_be_re_elected': 1.0, 'president_party': 'R'
                                 'president_overall_avg_job_approval': 0.395, 'last_D_house_
                                 'last_R_house_seats': 241.0, 'last_house_majority': 'R', ':
                                ignore_index=True)

house_df = house_df.append({'district': 'District 4', 'is_incumbent': 1.0, 'name': 'Jo
                                 'party': 'D', 'percent': 100, 'state': 'Massachusetts', 'vo
                                 'won': 1, 'year': 2018, 'first_time_elected': 2012, 'count_
                                 'unemployement_rate': 3.7, 'is_presidential_year': 0.0, \
                                 'president_can_be_re_elected': 1.0, 'president_party': 'R'
                                 'president_overall_avg_job_approval': 0.395, 'last_D_house_
                                 'last_R_house_seats': 241.0, 'last_house_majority': 'R', ':
                                ignore_index=True)
```

```
        house_df = house_df.append({'district': 'District 1', 'is_incumbent': 1.0, 'name': 'Ri
                                    'party': 'D', 'percent': 100, 'state': 'Massachusetts', 'vo
                                    'won': 1, 'year': 2018, 'first_time_elected': 2012, 'count_
                                    'unemployement_rate': 3.7, 'is_presidential_year': 0.0, \
                                    'president_can_be_re_elected': 1.0, 'president_party': 'R'
                                    'president_overall_avg_job_approval': 0.395, 'last_D_house_
                                    'last_R_house_seats': 241.0, 'last_house_majority': 'R', 'i
                                   ignore_index=True)

        house_df = house_df.append({'district': 'District 8', 'is_incumbent': 1.0, 'name': 'St
                                    'party': 'D', 'percent': 100, 'state': 'Massachusetts', 'vo
                                    'won': 1, 'year': 2018, 'first_time_elected': 2012, 'count_
                                    'unemployement_rate': 3.7, 'is_presidential_year': 0.0, \
                                    'president_can_be_re_elected': 1.0, 'president_party': 'R'
                                    'president_overall_avg_job_approval': 0.395, 'last_D_house_
                                    'last_R_house_seats': 241.0, 'last_house_majority': 'R', 'i
                                   ignore_index=True)

        house_df = house_df.append({'district': 'District 7', 'is_incumbent': 0.0, 'name': 'Ay
                                    'party': 'D', 'percent': 100, 'state': 'Massachusetts', 'vo
                                    'won': 1, 'year': 2018, 'first_time_elected': 2018, 'count_
                                    'unemployement_rate': 3.7, 'is_presidential_year': 0.0, \
                                    'president_can_be_re_elected': 1.0, 'president_party': 'R'
                                    'president_overall_avg_job_approval': 0.395, 'last_D_house_
                                    'last_R_house_seats': 241.0, 'last_house_majority': 'R', 'i
                                   ignore_index=True)

        house_df.to_csv('data/ready_to_use_dataset.csv', index=False)
```

After refelecting on the dataset, we decided that the count_victories variable should only count the number of victories of the candidate *to date* rather than the overall (past and future) number of victories.

```
In [51]: house_df = pd.read_csv('data/ready_to_use_dataset.csv')
         def get_count_victories(row):
             count_victories = 0
             # Try to find previous years in this state and district where this candidate won
             previous_won = house_df.loc[(house_df['name'] == row['name']) & \
                                         (house_df['won'] == 1) & \
                                         (house_df['state'] == row['state']) & \
                                         (house_df['district'] == row['district']) & \
                                         (house_df['year'] < row['year'])]

             if len(previous_won):
                 count_victories = len(previous_won)

             return count_victories
```

```
        house_df['count_victories'] = house_df.apply(get_count_victories, axis=1)

In [52]: house_df.to_csv('data/ready_to_use_dataset.csv', index=False)
```

4 Dataset preparation

```
In [49]: #house_df = pd.read_csv('data/ready_to_use_dataset.csv')

         years = house_df['year'].unique()

         for year in [2018]:
             print(year)
             grouped_df = house_df.loc[house_df['year'] == year].groupby(['state', 'district']
             dist_count = 0
             for state in house_df.loc[house_df['year'] == year, 'state'].unique():
                 dist_list = house_df.loc[(house_df['state'] == state), 'district'].unique()
                 n_dist = len(dist_list)
                 dist_count = dist_count + n_dist

                 for dist in dist_list:
                     check = grouped_df.loc[(grouped_df['state'] == state) & (grouped_df['dist
                     if len(check) == 0:
                         print(state, dist)

             print(dist_count)

2018
435
```

With the cleaned raw data at hand, we can now format them in a way that will facilitate the analysis:

```
In [327]: house_df = pd.read_csv('data/ready_to_use_dataset.csv')

          # Get the list of distinct years

          for year in house_df['year'].unique():
              for state in house_df['state'].unique():
                  for district in house_df.loc[house_df['state'] == state, 'district'].unique(
                      rows = house_df.loc[(house_df['year'] == year) & (house_df['state'] == s
                                   (house_df['district'] == district)]
                      if (len(rows) > 2):
                          print(year, state, district, len(rows))

1912 Utah District 2 4
1912 Utah District 1 4
```

```
1936 Delaware At-Large 3
1938 Delaware At-Large 3
1946 California District 16 3
1963 California District 23 8
1975 California District 37 12
1982 California District 43 3
1989 California District 15 7
1996 California District 2 3
1996 California District 8 3
1996 California District 9 3
1996 California District 15 4
1996 California District 21 3
1996 California District 41 3
1996 California District 44 3
1998 California District 1 3
1998 California District 2 3
1998 California District 15 3
1998 California District 38 3
2000 California District 34 3
2000 California District 36 3
2002 California District 9 3
2002 New Mexico District 2 3
2002 Louisiana District 2 4
2002 Louisiana District 1 3
2003 Hawaii District 2 5
2004 New Mexico District 1 3
2004 Louisiana District 5 3
2004 Louisiana District 1 6
2004 Louisiana District 6 3
2005 California District 5 9
2006 California District 10 3
2006 Colorado District 6 3
2006 Texas District 15 3
2006 Texas District 21 3
2006 Texas District 22 3
2006 New Mexico District 2 3
2006 Louisiana District 4 4
2006 Louisiana District 1 3
2006 Louisiana District 3 3
2010 California District 19 3
2012 Maryland District 1 3
2012 Maryland District 2 3
2012 Maryland District 7 4
2012 Louisiana District 2 4
2012 Louisiana District 1 3
2014 Louisiana District 1 3
2014 Wisconsin District 7 3
2016 Louisiana District 2 3
```

```
2016 Louisiana District 1 4
2016 Louisiana District 6 4
2018 Louisiana District 1 4
2018 Louisiana District 3 6
2018 Louisiana District 6 3
```

```
In [ ]:
```