data source: https://www.rankedchoicevoting.org/data_clearinghouse (https://www.rankedchoicevoting.org/data_clearinghouse)

RCV definition: https://ballotpedia.org/Ranked-choice_voting_(RCV (https://ballotpedia.org/Ranked-choice_voting_(RCV))

Additional analysis:

- http://archive3.fairvote.org/press/san-leandro-facts/ (http://archive3.fairvote.org/press/san-leandro-facts/)
- https://laurendo.wordpress.com/2010/11/24/running-the-numbers/ (https://laurendo.wordpress.com/2010/11/24/running-the-numbers/)
- http://www.acgov.org/rov/rcv/results/index.htm (http://www.acgov.org/rov/rcv/results/index.htm)

Objective for this notebook: separate the elections into the following categories:

1. Leading candidate in the first round has greater than 50% first choice votes
2. Leading candidate in the first round has between 45-50% first choice votes
3. Leading candidate in the first round has less than 45% of first choice votes

In [1]:
```
import glob
import pandas
print('pandas',pandas.__version__)
```

```
pandas 0.23.4
```

The Google drive contains folders and subfolders, with .txt files being the desired content

First attempt to get data: download individual txt file from Google drive

This approach works, but does not scale well. The manual labor of getting the relevant links would be tedious. Also the filename is not preserved

In [2]:
```
# https://stackoverflow.com/questions/38511444/python-download-files-from-google-drive-using-url
# https://github.com/ndrplz/google-drive-downloader
!pip install googledrivedownloader
```

```
Requirement already satisfied: googledrivedownloader in /opt/conda/lib/python3.6/site-packages (0.4)
```

In [3]:
```
from google_drive_downloader import GoogleDriveDownloader as gdd
```

In [4]:
```
gdd.download_file_from_google_drive(file_id='16lMt83ZaI_iLnyr1_R2fx43BimLwi01B',
                                    dest_path='./file.txt')
```

Rather than download one file, I got a zip

data gathering attempt 2: download all folders from drive manually

all the data: https://drive.google.com/drive/folders/1DJzIrTaDW3GSGJTkPTGAlpAMbozFG_pm (https://drive.google.com/drive/folders/1DJzIrTaDW3GSGJTkPTGAlpAMbozFG_pm)

Then download all content as a zip. Size is 1.5 GB. Of this, Sante Fe is 1.4GB

I started with just "Alameda County, CA (Berkeley, Oakland, San Leandro)" which is 18MB as a .zip

https://drive.google.com/drive/folders/1u_airJzoLC2PMYMHcF2KYJEKxxKBi5H7 (https://drive.google.com/drive/folders/1u_airJzoLC2PMYMHcF2KYJEKxxKBi5H7)

!mkdir voting_data !mkdir voting_data/Alameda !unzip voting_data/Alameda/drive-download-20190724T221439Z-001.zip

import a file

In [5]:
```
list_of_files = glob.glob('voting_data/Alameda/Alameda (Oakland, San Leandro, Berkeley) 2010/*')
len(list_of_files)
```

Out[5]:
```
34
```

In [6]:
```
list_of_files[0]
```

Out[6]:
```
'voting_data/Alameda/Alameda (Oakland, San Leandro, Berkeley) 2010/ballot_image_ Member, City Council, District 4 - Oakland_Nov 2010.txt'
```

Normally this is where I would want to create a tuple of `ballot_image_` and `master_lookup_` in order to decode the data.

Since I don't care about the name of who actually won each election, I can stick with numeric data for now

In [7]:
```
# I could analyze the fixed-width file as plain text
with open(list_of_files[0],'r') as fil:
    #file_contents = fil.read()  # do not split on newline -- one long s
tring
    file_contents = fil.readlines()  # creates a list of entries

len(file_contents)
```

Out[7]:
```
71652
```

Rather than manually split each row in the plain text file, I'll use Pandas

In [8]:
```
df = pandas.read_fwf(list_of_files[0],
                     header=None,
                     widths=[7,9,7,3,7,3,7,1,1])
df.columns=['contest_id','pref_voter_id',
            'serial_number','tally_type_id',
            'precinct_id','vote_rank',
            'candidate_id','over_vote','under_vote']
df.shape
```

Out[8]:
```
(71652, 9)
```

In [9]:
```
df.head()
```

Out[9]:

	contest_id	pref_voter_id	serial_number	tally_type_id	precinct_id	vote_rank	candidat
0	37	12307	2	3	152	1	408
1	37	12307	2	3	152	2	409
2	37	12307	2	3	152	3	406
3	37	12313	6	3	293	1	411
4	37	12313	6	3	293	2	408

analyze dataframe

In [10]:
```
df.nunique()
```

Out[10]:
```
contest_id           1
pref_voter_id    23884
serial_number       81
tally_type_id        5
precinct_id         45
vote_rank            3
candidate_id         9
over_vote            2
under_vote           2
dtype: int64
```

- There are 81 unique voting machines
- There are 23,885 voters
- There are 9 candidates

In [11]:
```
# what are the candidate IDs?
df['candidate_id'].unique()
```

Out[11]:
```
array([408, 409, 406, 411, 410,   0, 405, 407,  76])
```

In [12]:
```
# how many rows does each candidate appear in?
df['candidate_id'].value_counts()
```

Out[12]:
```
0      18090
408    14790
405    12113
410     7521
407     7468
409     5019
406     3529
411     2723
76       399
Name: candidate_id, dtype: int64
```

In [13]:
```
df[df['candidate_id']==0].head()
```

Out[13]:

	contest_id	pref_voter_id	serial_number	tally_type_id	precinct_id	vote_rank	candida
8	37	12769	7	3	152	3	0
17	37	12772	7	3	152	3	0
18	37	12773	7	3	152	1	0
19	37	12773	7	3	152	2	0
20	37	12773	7	3	152	3	0

In [14]:
```
# drop rows where candidate_id==0

df_cand = df[df['candidate_id']!=0]
```

did any candidate win the first round?

In [15]:
```
# compare only the candidates with rank==1
df_cand[df_cand['vote_rank']==1].head()
```

Out[15]:

	contest_id	pref_voter_id	serial_number	tally_type_id	precinct_id	vote_rank	candida
0	37	12307	2	3	152	1	408
3	37	12313	6	3	293	1	411
6	37	12769	7	3	152	1	408
9	37	12770	7	3	152	1	406
12	37	12771	7	3	152	1	408

In [16]:
```
# https://jakevdp.github.io/PythonDataScienceHandbook/03.08-aggregation-and-grouping.html
# https://www.geeksforgeeks.org/python-pandas-dataframe-groupby/
df_cand[df_cand['vote_rank']==1].groupby('candidate_id')['vote_rank'].count()
```

Out[16]:
```
candidate_id
76       67
405    4794
406     878
407    2448
408    8746
409    1133
410    2345
411     526
Name: vote_rank, dtype: int64
```

In RCV, if a candidate wins a majority of first-preference votes, he or she is declared the winner.

Caveat: this dataframe ignores rows where no candidate preference was provided

In [17]:
```
# majority = half the sum of first choices
majority_first_round = df_cand[df_cand['vote_rank']==1].groupby('candidate_id')['vote_rank'].count().sum()/2
majority_first_round
```

Out[17]:
```
10468.5
```

In [18]:
```
df_cand[df_cand['vote_rank']==1].groupby('candidate_id')['vote_rank'].co
unt()>majority_first_round
```

Out[18]:
```
candidate_id
76     False
405    False
406    False
407    False
408    False
409    False
410    False
411    False
Name: vote_rank, dtype: bool
```

Since no candidate won the majority of the first round, the next step in the algorithm is to eliminate the candidate with the lowest count.

In [19]:
```
df_cand[df_cand['vote_rank']==1].groupby('candidate_id')['vote_rank'].co
unt().idxmin()
```

Out[19]:
```
76
```

In [20]:
```
df_second_round = df_cand[df_cand['candidate_id']!=76]
df_second_round.head()
```

Out[20]:

	contest_id	pref_voter_id	serial_number	tally_type_id	precinct_id	vote_rank	candidat
0	37	12307	2	3	152	1	408
1	37	12307	2	3	152	2	409
2	37	12307	2	3	152	3	406
3	37	12313	6	3	293	1	411
4	37	12313	6	3	293	2	408

Then first-preference votes cast for the failed candidate are eliminated, lifting the second-preference choices indicated on those ballots.

However, for my analysis I only care about the first round. Recall my objective is to categorize elections by

- Leading candidate in the first round has greater than 50% first choice votes
- Leading candidate in the first round has between 45-50% first choice votes
- Leading candidate in the first round has less than 45% of first choice votes

In [21]:
```
number_of_first_choice_votes = df_cand[df_cand['vote_rank']==1].groupby(
'candidate_id')['vote_rank'].count().sum()
```

In [22]:
```
outcome = df_cand[df_cand['vote_rank']==1].groupby('candidate_id')['vote
_rank'].count()>number_of_first_choice_votes*0.5
outcome
```

Out[22]:
```
candidate_id
76     False
405    False
406    False
407    False
408    False
409    False
410    False
411    False
Name: vote_rank, dtype: bool
```

In [23]:
```
outcome.any()
```

Out[23]:
```
False
```

My next step is to generalize the above analysis to all the elections