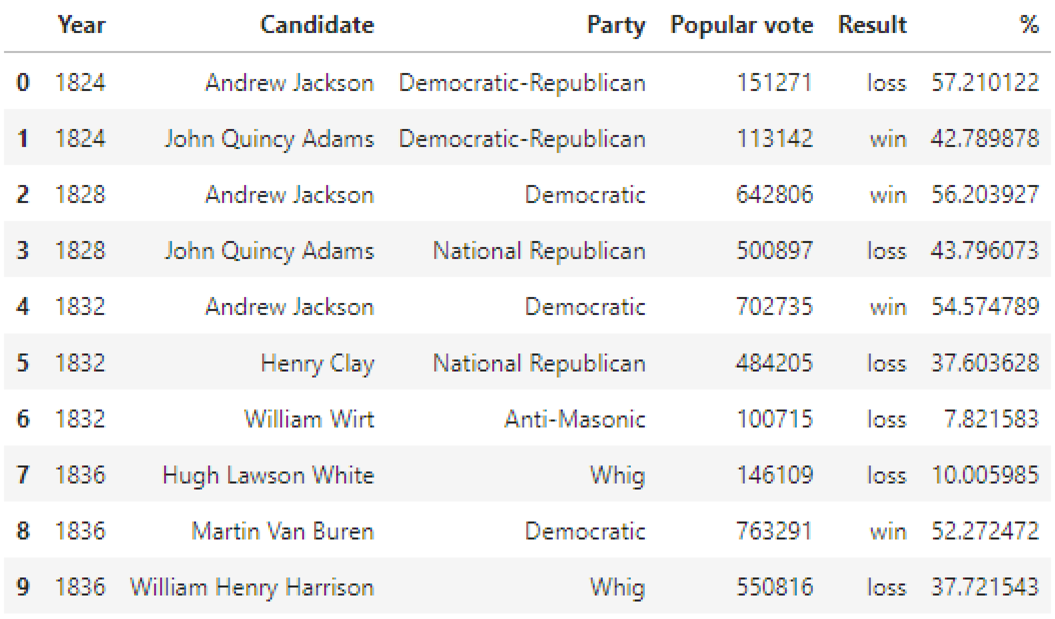
Data 100, Spring 2023 February 1, 2023

Discussion #2 Solutions

Pandas Practice

The purpose of this question is to give you an opportunity to practice interpreting and writing `pandas` code. On exams, you may be asked to write `pandas` code or recognize lines of `pandas` code that can be used to accomplish certain tasks.

The following questions are based off of the `elections.csv` data referenced in lecture.



	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113142	win	42.789878
2	1828	Andrew Jackson	Democratic	642806	win	56.203927
3	1828	John Quincy Adams	National Republican	500897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702735	win	54.574789
5	1832	Henry Clay	National Republican	484205	loss	37.603628
6	1832	William Wirt	Anti-Masonic	100715	loss	7.821583
7	1836	Hugh Lawson White	Whig	146109	loss	10.005985
8	1836	Martin Van Buren	Democratic	763291	win	52.272472
9	1836	William Henry Harrison	Whig	550816	loss	37.721543

1. Assume `elections.csv` is the file containing the elections data, and it is in your current working directory already. **Write a line of code to read the data into a `pandas` DataFrame, storing it in a variable called `elections`.** It's good practice to quickly check the contents of such variables to ensure you loaded up the right dataset, so follow up with a line of code whose output displays the first 10 rows of the dataset.

   Solution:

   ```
   elections = pd.read_csv("elections.csv")
   elections.head(10)
   ```

2. We want to select the "Popular vote" column as a `pd.Series`. Which of the following lines of code will error? (*Hint: think about the valid data-type of arguments to loc/iloc*)

    1. `elections['Popular vote']`
    2. `elections.iloc['Popular vote']`
    3. `elections.loc['Popular vote']`
    4. `elections.loc[:, 'Popular vote']`
    5. `elections.iloc[:, 'Popular vote']`

    Solution: 2, 3, and 5. 2 and 5 error because we cannot pass a string into `.iloc`, as `.iloc` only takes in integers. 3 will error because `.loc` requires both rows and columns to be passed in, this option only passes in a column.

3. Write one line of `pandas` code that returns a `pd.DataFrame` that only contains election results from the 1900s.

    Solution:

    `elections[(elections['Year'] >= 1900) & (elections['Year'] < 2000)]`

    `elections.query("Year >= 1900 & Year < 2000")`

    There are other correct solutions, but we present these two to showcase how to utilize multiple Boolean conditions.

4. Write one line of `pandas` code that returns a `pd.Series`, where the index is the Party, and the values are how many times that party won an election.

    `Hint:` use `value_counts()`.

    Solution:

    `elections[elections['Result'] == 'win']['Party'].value_counts()`

    You can also do this with `groupby`, but it's more complex:
    `elections[elections['Result'] == 'win'].groupby('Party').size()`

    Notice the slight difference in the output of the two lines of code.

5. Which of the following lines of code returns a `pandas` Series with the **mean** vote percentage **for each political party**, for all years given, sorted in **decreasing** order?

1. `elections.groupby(‘Party’)[‘%’].agg(‘mean’).sort_values()`
2. `elections.groupby(‘Party’)[‘%’].agg(‘mean’).sort_values(ascending = False)`
3. `elections.groupby(‘Party’)[‘%’].mean().sort_values()`
4. `elections.groupby(‘Party’)[‘%’].mean().sort_values(ascending = False)`

Solution: Only 2 and 4. It is important to note that `sort_values` defaults to arranging a series in increasing order, so `ascending = False` must be specified. Using `agg(’mean’)` and `mean()` on a `groupby` object are equally valid ways of obtaining the means of each group.

6. Write a line of `pandas` code that returns a `pandas` series with the year as the index, and the total number of votes that were cast across all parties for that year.

Solution: `elections.groupby(‘Year’)[‘Popular vote’].agg(sum)`
or
`elections.groupby(‘Year’)[‘Popular vote’].sum()`
or
`elections.groupby(‘Year’).sum()[‘Popular vote’]`

7. Finally, write a line of `pandas` code that returns a `pandas` Series whose index are the years and whose values are the number of candidates that participated in those years’ elections.

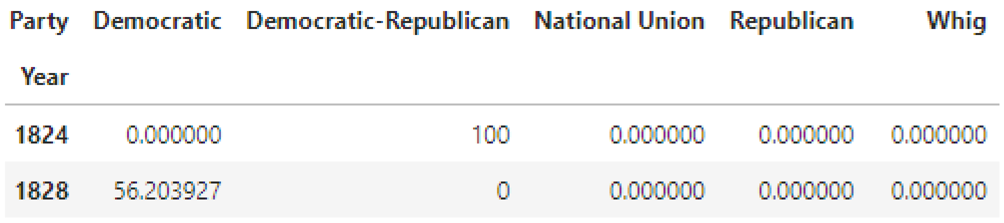
Solution: `elections.groupby(’Year’).size()`

8. Write a line of `pandas` code that creates a filtered DataFrame named filtered_parties from the elections dataset and keeps only the parties that have at least one election % more than 50%.

Solution:

```
filtered_parties = elections.groupby("Party")
                            .filter(lambda sf: sf["\%"].max() >= 50)
```

9. Write a line of `pandas` code that uses the `filtered_parties` DataFrame to return a new DataFrame with row indices that correspond to the year and columns that correspond to each party. Each entry should be the total percentage of votes for all the candidates that ran during that particular year for the specified party. *Below is an example.*



Party	Democratic	Democratic-Republican	National Union	Republican	Whig
Year					
1824	0.000000	100	0.000000	0.000000	0.000000
1828	56.203927	0	0.000000	0.000000	0.000000

Solution:

```
elections_pivot = filtered_parties.pivot_table(
                                            index="Year",
                                            columns="Party",
                                            values=["%"],
                                            aggfunc=np.sum,
                                            fill_value = 0
                                                )
```