Data 8 **Name:** _____
Worksheet 2
Conceptual Office Hours

This worksheet can serve as a general overview of materials you have learned this week and give you an opportunity to practice solving them in an exam-like format. NOTE: This is not necessarily representative of what will be tested on the actual exams. For any issues or questions, contact Robert Sweeney Blanco at robertsweeneyblanco@berkeley.edu.

For all problems, you may assume you have imported datascience and numpy as np.

The table *nba* has the score of every NBA game ever played. Each game (denoted by **game_id**) appears in the table twice, one row for each team involved. Each row contains information of whether the team won the game. The **result** column has a "W" if the team won and a "L" if the team lost. The **is_playoffs** column determines if the game was during the playoffs (1) or during the regular season (0).

game_id	year	date_game	is_playoffs	team	opponent	result
194611010TRH	1947	11/1/1946	0	Huskies	Knicks	L
194611010TRH	1947	11/1/1946	0	Knicks	Huskies	W
194611020CHS	1947	11/2/1946	0	Stags	Knicks	W
194611020CHS	1947	11/2/1946	0	Knicks	Stags	L
194611020DTF	1947	11/2/1946	0	Falcons	Capitols	L
194611020DTF	1947	11/2/1946	0	Capitols	Falcons	W
194611020PRO	1947	11/2/1946	0	Celtics	Steamrollers	L
194611020PRO	1947	11/2/1946	0	Steamrollers	Celtics	W
194611020STB	1947	11/2/1946	0	Ironmen	Bombers	L
194611020STB	1947	11/2/1946	0	Bombers	Ironmen	W

... (118006 rows omitted)

1. Is the **is_playoffs** column numerical or categorical?
   Categorical

2. Set the variable *warriors* to the table with only rows with "Warriors" as the team. The table should look like the one below.

game_id	year	date_game	is_playoffs	team	opponent	result
194611070GSW	1947	11/7/1946	0	Warriors	Ironmen	W
194611140GSW	1947	11/14/1946	0	Warriors	Capitols	W
194611190GSW	1947	11/19/1946	0	Warriors	Bombers	L
194611210GSW	1947	11/21/1946	0	Warriors	Stags	L
194611260GSW	1947	11/26/1946	0	Warriors	Celtics	W
194611280GSW	1947	11/28/1946	0	Warriors	Falcons	L
194611300NYK	1947	11/30/1946	0	Warriors	Knicks	L
194612030GSW	1947	12/3/1946	0	Warriors	Steamrollers	W
194612050GSW	1947	12/5/1946	0	Warriors	Knicks	W
194612070STB	1947	12/7/1946	0	Warriors	Bombers	W

... (5647 rows omitted)

warriors=nba.where("team", "Warriors")

3. What percent of games have the warriors won? (HINT: Use the *warriors* variable you defined above)

np.count_nonzero(warriors.column("result") == "W") / warriors.num_rows

4. Filter the *warriors* table to only include rows where the opponents are the "Lakers", "Kings", or "Suns". Set that new table to the variable *rivals*.

game_id	year	date_game	is_playoffs	team	opponent	result
194811110GSW	1949	11/11/1948	0	Warriors	Kings	L
194811280LAL	1949	11/28/1948	0	Warriors	Lakers	L
194812040ROC	1949	12/4/1948	0	Warriors	Kings	L
194812210GSW	1949	12/21/1948	0	Warriors	Lakers	W
194901110ROC	1949	1/11/1949	0	Warriors	Kings	L
194901230LAL	1949	1/23/1949	0	Warriors	Lakers	L
194902030GSW	1949	2/3/1949	0	Warriors	Lakers	L
194902220ROC	1949	2/22/1949	0	Warriors	Kings	L
194903100GSW	1949	3/10/1949	0	Warriors	Kings	L
194903200LAL	1949	3/20/1949	0	Warriors	Lakers	L

... (1050 rows omitted)

rivals=warriors.where("opponent", are.contained_in(make_array("Lakers", "Kings", "Suns")))

Suppose you want to study how many times the Warriors played their rivals throughout the year. Write a line of code that evaluates to the table below. Set it equal to the variable *game_counts*.

year	Kings	Lakers	Suns
1949	5	5	0
1950	6	6	0
1951	6	6	0
1952	6	6	0
1953	6	6	0
1954	8	8	0
1955	9	9	0
1956	9	9	0
1957	9	9	0
1958	9	9	0

... (57 rows omitted)

game_counts = rival.pivot("opponent", "year")

5. Use the *game_counts* table to find the number of times the Warriors have played the Lakers in NBA history.

   sum(game_counts.column("Lakers"))

6. It seems like the Warriors didn't play the Suns for the first few years. That is because the Suns weren't created yet. Write a line of code that finds the first year the Warriors played the Suns.

   game_counts.where("Suns", are.above(0)).column("year").item(0)

The table *passwords* has the top 500 most used passwords. Its never a good idea to use a common password, since it makes your account more vulnerable to hackers.

Ranking	Password
1	123456
2	password
3	12345678
4	qwerty
5	123456789
6	12345
7	1234
8	111111
9	1234567
10	dragon

... (490 rows omitted)

7. Define a function that takes in a String (new_password) and returns a boolean indicating whether the password is good (not in the list).

```
def is_safe(new_password):
    top_passwords = passwords.column("Password")
    for password in top_passwords:
        if new_password == password:
            return False
    return True
```

John is playing cards with a standard 52 card deck. Recall that a deck of cards has 4 suits: clubs, diamonds, hearts and spades. There are 13 cards in each suit: 2 through 10, jack, queen, king, ace. Assume each scenario is performed independently, and leave each answer in the form of an expression. What is the probability that:

8. John deals a card: it is a king.

$\frac{4}{52} = \frac{1}{13}$

9. John deals two cards: both are aces.

$\frac{4}{52} \cdot \frac{3}{51}$

10. John deals two cards: one is a heart and one is a club.

$\frac{13}{52} \cdot \frac{13}{51} + \frac{13}{52} \cdot \frac{13}{51}$

11. John deals three cards: at least one is a heart.

$1 - (\frac{39}{52} \cdot \frac{38}{51} \cdot \frac{37}{50})$

Suppose you conduct a survey where you ask people to pick a random from 1 to 100 and record your results in the table *survey*.

Respondent #	Favorite Number
0	95
1	10
2	87
3	26
4	6
5	87
6	36
7	100
8	91
9	24

... (990 rows omitted)

12. Calculate the missing density value. (You can leave your answer as an expression to avoid tedious math)

Bin	1-20	21-40	41-60	61-80	81-100
Height	0.5	1	???	0.5	1

$(100 - (20 \cdot 0.5 + 20 \cdot 1 + 20 \cdot 0.5 + 20 \cdot 1))/20 = 2$

13. Suppose you are curious what proportion of people picked 10 or less. Write a line of code to calculate that value.

survey.where("Favorite Number", are.below_or_equal_to(10)).num_rows / survey.num_rows

14. Suppose you ran survey.hist("Favorite Number", bins=make_array(21,31,41)). The first density is .8, what is the the second density?

$(100 - .8 \cdot 10)/10$

In the US, the presidency is won by the candidate who can get the most electoral votes. Each state has a certain number of electoral votes and the candidate that gets the most votes in a particular state gets all their electoral votes. (For those political savants, please ignore odd cases like the 'faithless elector'). Suppose it is the morning after election night in the US and you want to know who won the presidential election, Candidate 0 or Candidate 1, without looking at the news. You get the results in a table below set to the variable *election*. The table below contains the results from all the districts in every state. Each row represents a district. The 'State' column indicates what State the district is in, the Candidate 1 and Candidate 2 columns indicate how many votes each candidate got in that district. Follow the instructions below to find out who won!

State	Candidate 1	Candidate 2
Alaska	567986	417957
Alabama	634748	422345
Alabama	646046	383950
Alabama	639883	401502
Alabama	768271	279735
Alabama	656317	398211
Alabama	763135	274250
Alabama	278344	784355
Arkansas	626531	430024
Arkansas	561824	485971

... (426 rows omitted)

To find out who won every state, we need a table that shows the aggregate results at the state level, like the one shown below. Write a line of code that evaluates to that table and set it equal to *state_results*.

State	Candidate 1 sum	Candidate 2 sum
Alabama	4.38674e+06	2.94435e+06
Alaska	567986	417957
Arizona	4.75034e+06	4.62056e+06
Arkansas	2.49585e+06	1.69816e+06
California	1.99442e+07	3.51259e+07
Colorado	3.34321e+06	4.09291e+06
Connecticut	2.09323e+06	3.32391e+06
Delaware	410840	679043
District of Columbia	5455	5515
Florida	1.33472e+07	1.53152e+07

... (41 rows omitted)

state_results = election.group("State", sum)

Define a function winner that takes in two numbers. If the first number is bigger than the second, return the string "Candidate 1", else return "Candidate 2".

```
def winner(num_votes1, num_votes2):
    if num_votes1 > num_votes2:
        return "Candidate 1"
    else:
        return "Candidate 2"
```

Set a variable *state_winners* to an array of the winner of each state. Then use it to append a column to the *state_results* table like the one shown below that shows the state and its winner. If both lines of code are run, the *state_results* table should look like the one below. (HINT: The function you defined in the previous problem might be useful)

State	Candidate 1 sum	Candidate 2 sum	Candidate
Alabama	4.31884e+06	2.90087e+06	Candidate 1
Alaska	559194	411785	Candidate 1
Arizona	4.6768e+06	4.55233e+06	Candidate 1
Arkansas	2.45722e+06	1.67308e+06	Candidate 1
California	1.96355e+07	3.46072e+07	Candidate 2
Colorado	3.29146e+06	4.03246e+06	Candidate 2
Connecticut	2.06083e+06	3.27482e+06	Candidate 2
Delaware	404480	669015	Candidate 2
District of Columbia	5541	5312	Candidate 1
Florida	1.31405e+07	1.5089e+07	Candidate 2

... (41 rows omitted)

```
state_winners = state_results.apply(winner, "Candidate 1 sum", "Candidate 2 sum")
```

```
state_results = state_results.with_column("Candidate", state_winners)
```

Now we need to know how many electoral votes each state has. Use the table *electoral* (left) to append the number of electoral votes each state has to the *state_results* table. Once the line is executed, the *state_results* table should look like the table on the right.

State	Electoral Votes
Alabama	9
Alaska	3
Arizona	11
Arkansas	6
California	55
Colorado	9
Connecticut	7
District of Columbia	3
Delaware	3
Florida	29

... (41 rows omitted)

State	Candidate 1 sum	Candidate 2 sum	Candidate	Electoral Votes
Alabama	4.38674e+06	2.94435e+06	Candidate 1	9
Alaska	567986	417957	Candidate 1	3
Arizona	4.75034e+06	4.62056e+06	Candidate 1	11
Arkansas	2.49585e+06	1.69816e+06	Candidate 1	6
California	1.99442e+07	3.51259e+07	Candidate 2	55
Colorado	3.34321e+06	4.09291e+06	Candidate 2	9
Connecticut	2.09323e+06	3.32391e+06	Candidate 2	7
Delaware	410840	679043	Candidate 2	3
District of Columbia	5455	5515	Candidate 2	3
Florida	1.33472e+07	1.53152e+07	Candidate 2	29

... (41 rows omitted)

state_results = state_results.join("State", electoral)

Now write a line of code that shows how many electoral votes each candiate got. Make it so that the resulting table should only have two columns, one with the candidates names and the other with their electoral votes total.

state_results.group("Candidate", sum).select("Candidate", "Electoral Votes sum")

The last line should evaluate to the following table.

Candidate	Electoral Votes sum
Candidate 1	191
Candidate 2	347