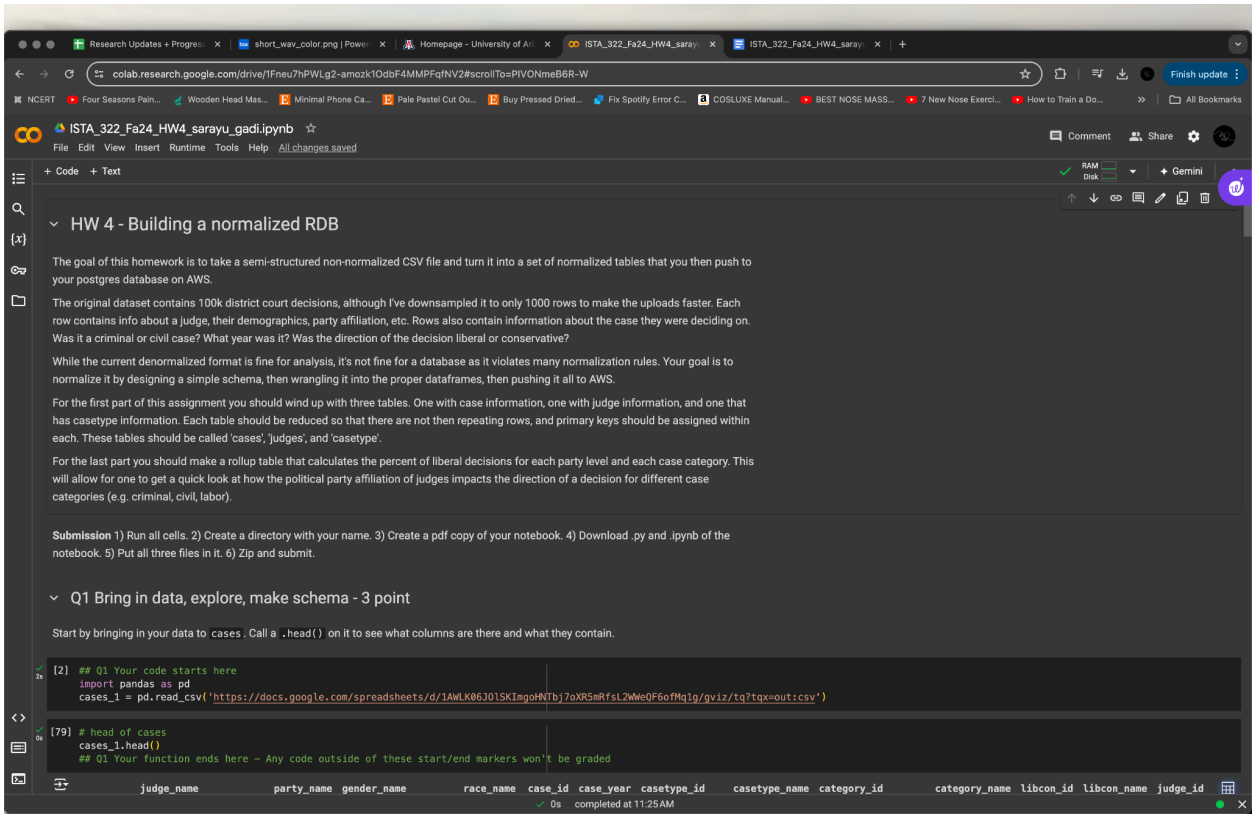
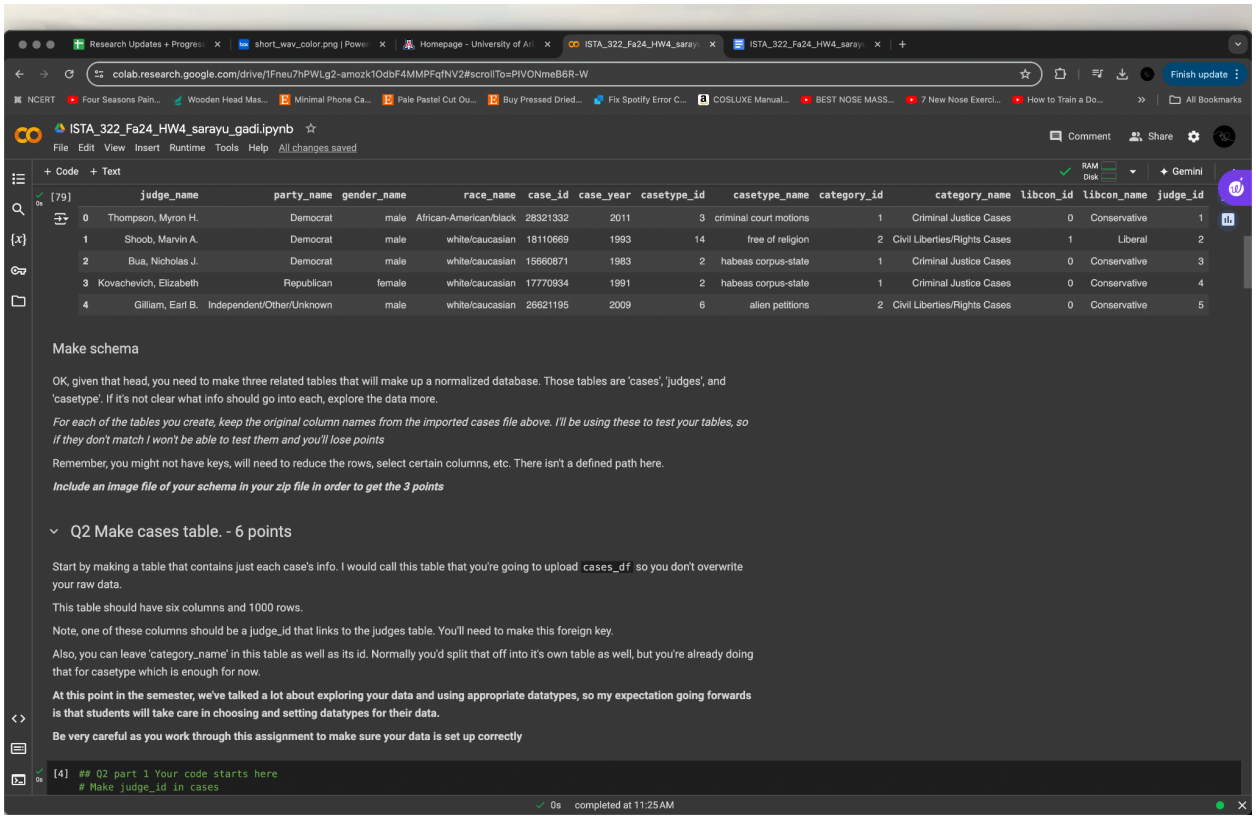
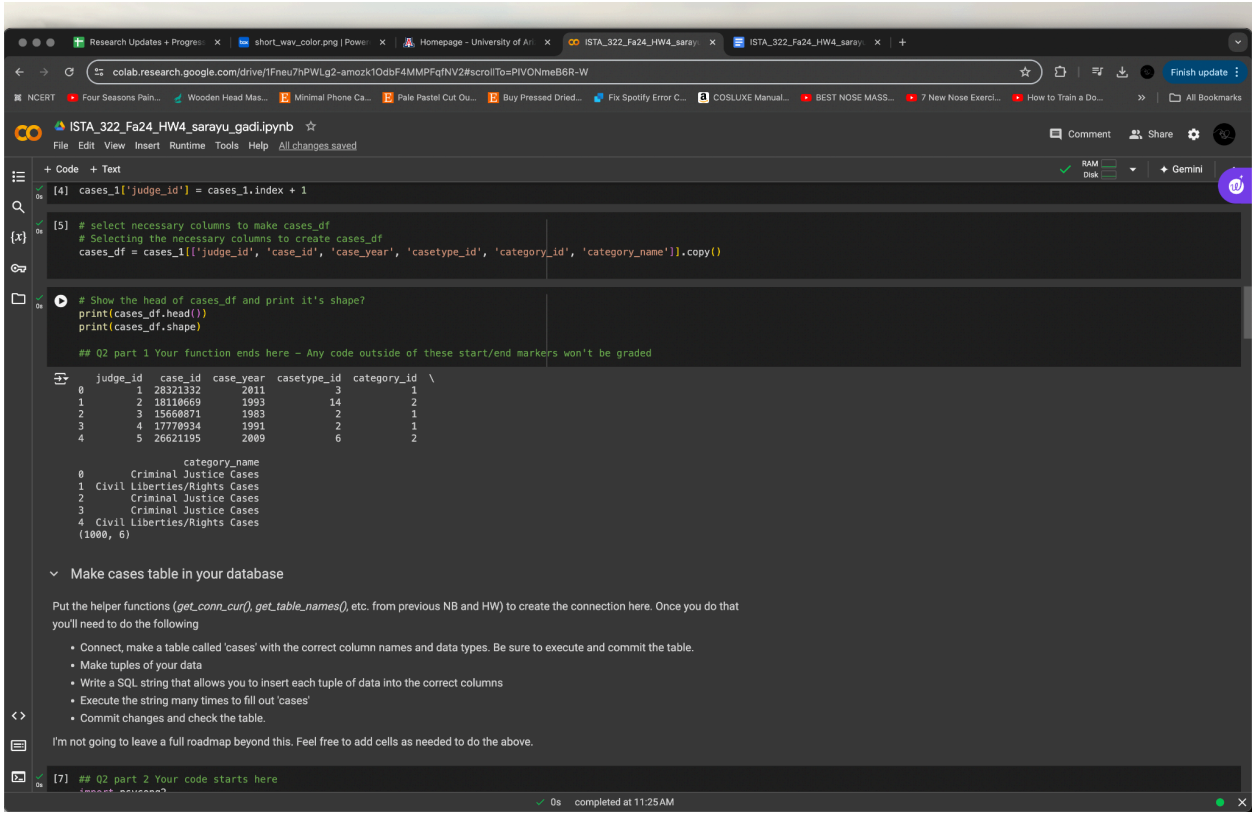
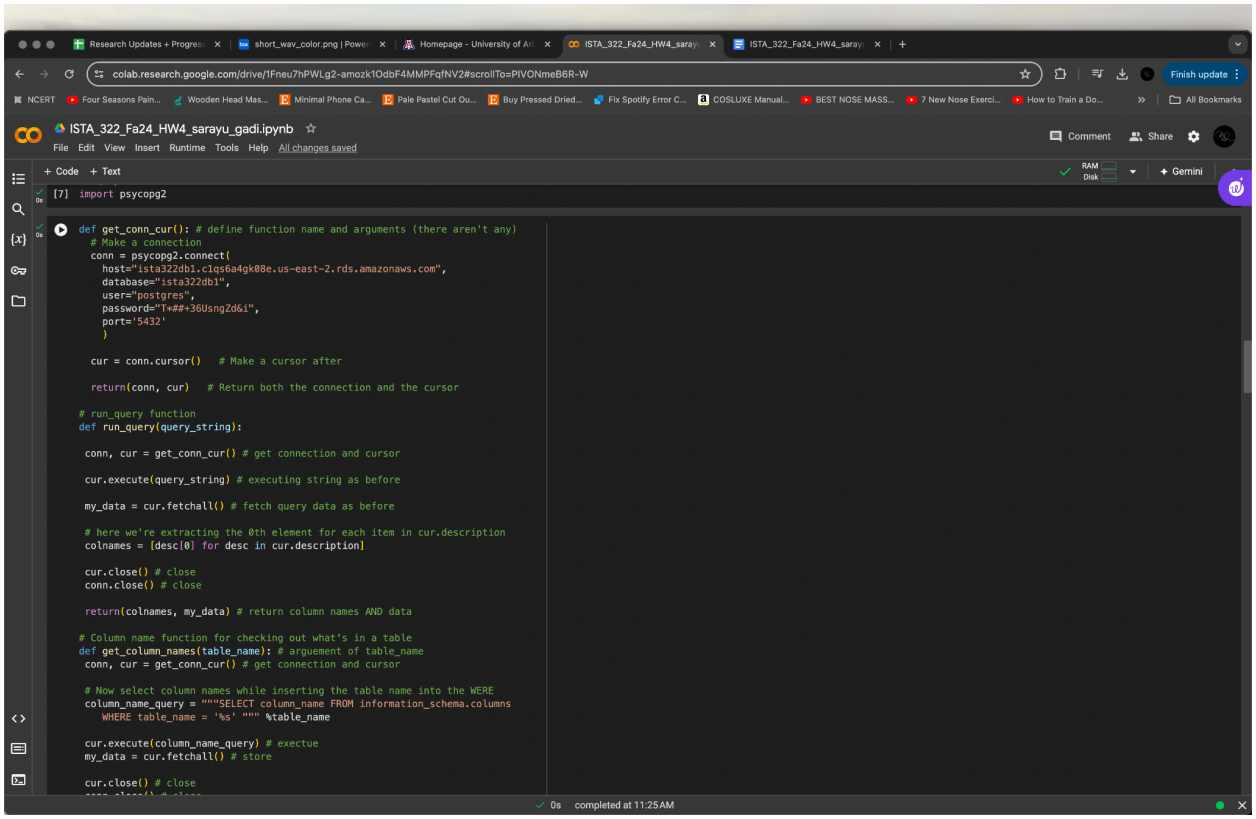
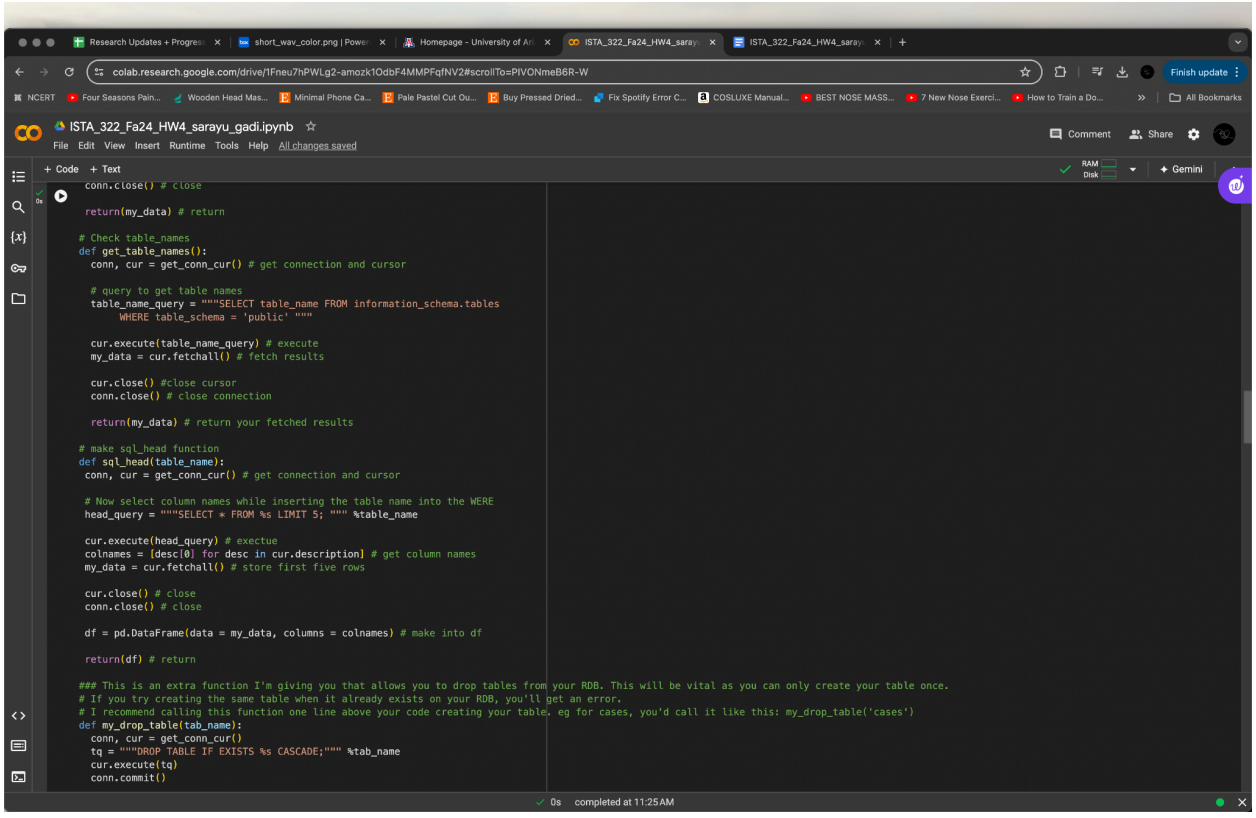

colab.research.google.com/drive/1Fneu7hPWLg2-amozk1OdbF4MMPFqfNV2#scrollTo=PIVONmeB6R-W
ISTA_322_Fa24_HW4_sarayu_gadi.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
HW 4 - Building a normalized RDB
The goal of this homework is to take a semi-structured non-normalized CSV file and turn it into a set of normalized tables that you then push to your postgres database on AWS.
The original dataset contains 100k district court decisions, although I've downsampled it to only 1000 rows to make the uploads faster. Each row contains info about a judge, their demographics, party affiliation, etc. Rows also contain information about the case they were deciding on. Was it a criminal or civil case? What year was it? Was the direction of the decision liberal or conservative?
While the current denormalized format is fine for analysis, it's not fine for a database as it violates many normalization rules. Your goal is to normalize it by designing a simple schema, then wrangling it into the proper dataframes, then pushing it all to AWS.
For the first part of this assignment you should wind up with three tables. One with case information, one with judge information, and one that has casetype information. Each table should be reduced so that there are not then repeating rows, and primary keys should be assigned within each. These tables should be called 'cases', 'judges', and 'casetype'.
For the last part you should make a rollup table that calculates the percent of liberal decisions for each party level and each case category. This will allow for one to get a quick look at how the political party affiliation of judges impacts the direction of a decision for different case categories (e.g. criminal, civil, labor).
Submission 1) Run all cells. 2) Create a directory with your name. 3) Create a pdf copy of your notebook. 4) Download .py and .ipynb of the notebook. 5) Put all three files in it. 6) Zip and submit.
Q1 Bring in data, explore, make schema - 3 point
Start by bringing in your data to cases. Call a .head() on it to see what columns are there and what they contain.
[2] ## Q1 Your code starts here
import pandas as pd
cases_1 = pd.read_csv('https://docs.google.com/spreadsheets/d/1AWLK06JOlSKImgoHNTbj7oXR5mRfsL2WWeQF6ofMq1g/gviz/tq?tqx=out:csv')
[79] # head of cases
cases_1.head()
Q1 Your function ends here - Any code outside of these start/end markers won't be graded
judge_name party_name gender_name race_name case_id case_year casetype_id casetype_name category_id category_name libcon_id libcon_name judge_id
0s completed at 11:25 AM

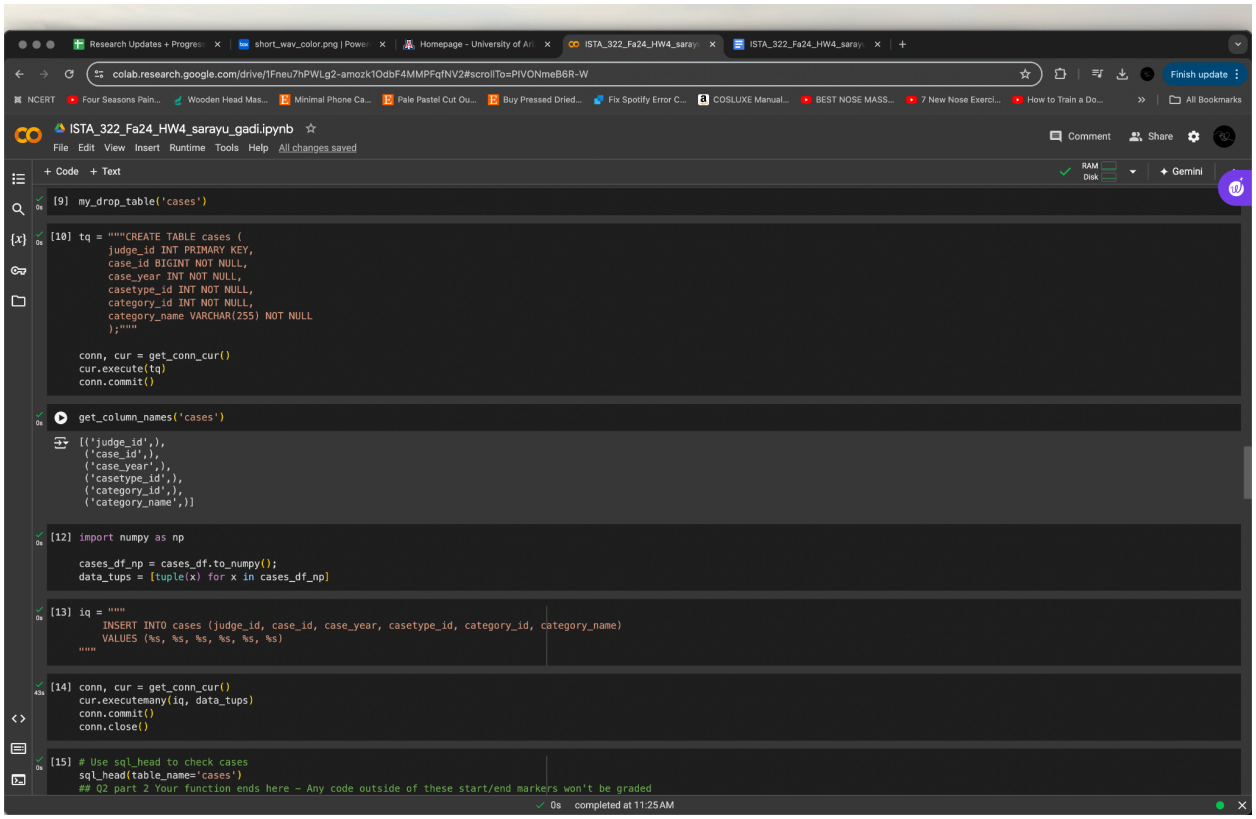

colab.research.google.com/drive/1Fneu7hPWLg2-amozk1OdbF4MMPFqfNV2#scrollTo=PIVONmeB6R-W
ISTA_322_Fa24_HW4_sarayu_gadi.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[79]
judge_name party_name gender_name race_name case_id case_year casetype_id casetype_name category_id category_name libcon_id libcon_name judge_id
0 Thompson, Myron H. Democrat male African-American/black 28321332 2011 3 criminal court motions 1 Criminal Justice Cases 0 Conservative 1
1 Shoob, Marvin A. Democrat male white/caucasian 18110669 1993 14 free of religion 2 Civil Liberties/Rights Cases 1 Liberal 2
2 Bua, Nicholas J. Democrat male white/caucasian 15660871 1983 2 habeas corpus-state 1 Criminal Justice Cases 0 Conservative 3
3 Kovachevich, Elizabeth Republican female white/caucasian 17770934 1991 2 habeas corpus-state 1 Criminal Justice Cases 0 Conservative 4
4 Gilliam, Earl B. Independent/Other/Unknown male white/caucasian 26621195 2009 6 alien petitions 2 Civil Liberties/Rights Cases 0 Conservative 5
Make schema
OK, given that head, you need to make three related tables that will make up a normalized database. Those tables are 'cases', 'judges', and 'casetype'. If it's not clear what info should go into each, explore the data more.
For each of the tables you create, keep the original column names from the imported cases file above. I'll be using these to test your tables, so if they don't match I won't be able to test them and you'll lose points
Remember, you might not have keys, will need to reduce the rows, select certain columns, etc. There isn't a defined path here.
Include an image file of your schema in your zip file in order to get the 3 points
Q2 Make cases table. - 6 points
Start by making a table that contains just each case's info. I would call this table that you're going to upload cases_df so you don't overwrite your raw data.
This table should have six columns and 1000 rows.
Note, one of these columns should be a judge_id that links to the judges table. You'll need to make this foreign key.
Also, you can leave 'category_name' in this table as well as its id. Normally you'd split that off into it's own table as well, but you're already doing that for casetype which is enough for now.
At this point in the semester, we've talked a lot about exploring your data and using appropriate datatypes, so my expectation going forwards is that students will take care in choosing and setting datatypes for their data.
Be very careful as you work through this assignment to make sure your data is set up correctly
[4] ## Q2 part 1 Your code starts here
Make judge_id in cases
0s completed at 11:25 AM


ISTA_322_Fa24_HW4_sarayu_gadi.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[4] cases_1['judge_id'] = cases_1.index + 1
[5] # select necessary columns to make cases_df
# Selecting the necessary columns to create cases_df
cases_df = cases_1[['judge_id', 'case_id', 'case_year', 'casetype_id', 'category_id', 'category_name']].copy()
# Show the head of cases_df and print it's shape?
print(cases_df.head())
print(cases_df.shape)
Q2 part 1 Your function ends here — Any code outside of these start/end markers won't be graded
judge_id case_id case_year casetype_id category_id \
0 1 28321332 2011 3 1
1 2 18110669 1993 14 2
2 3 15660871 1983 2 1
3 4 17770934 1991 2 1
4 5 26621195 2009 6 2
category_name
0 Criminal Justice Cases
1 Civil Liberties/Rights Cases
2 Criminal Justice Cases
3 Criminal Justice Cases
4 Civil Liberties/Rights Cases
(1000, 6)
Make cases table in your database
Put the helper functions (get_conn_cur(), get_table_names(), etc. from previous NB and HW) to create the connection here. Once you do that you'll need to do the following
Connect, make a table called 'cases' with the correct column names and data types. Be sure to execute and commit the table.
Make tuples of your data
Write a SQL string that allows you to insert each tuple of data into the correct columns
Execute the string many times to fill out 'cases'
Commit changes and check the table.
I'm not going to leave a full roadmap beyond this. Feel free to add cells as needed to do the above.
[7] ## Q2 part 2 Your code starts here
0s completed at 11:25 AM

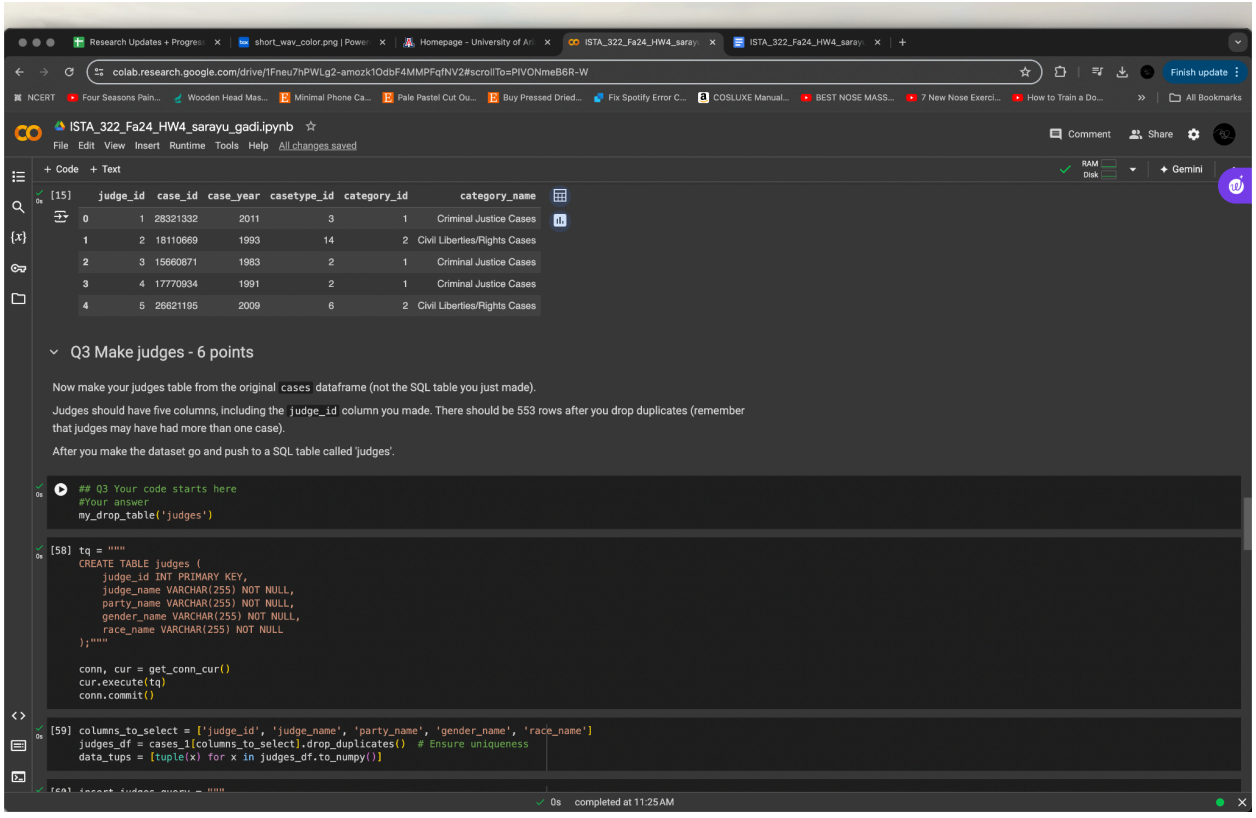

ISTA_322_Fa24_HW4_sarayu_gadi.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[7] import psycopg2
def get_conn_cur(): # define function name and arguments (there aren't any)
# Make a connection
conn = psycopg2.connect(
host="ista322db1.c1qs6a4gk08e.us-east-2.rds.amazonaws.com",
database="ista322db1",
user="postgres",
password="T+##+36UsngZd&i",
port='5432'
)
cur = conn.cursor() # Make a cursor after
return(conn, cur) # Return both the connection and the cursor
# run_query function
def run_query(query_string):
conn, cur = get_conn_cur() # get connection and cursor
cur.execute(query_string) # executing string as before
my_data = cur.fetchall() # fetch query data as before
# here we're extracting the 0th element for each item in cur.description
colnames = [desc[0] for desc in cur.description]
cur.close() # close
conn.close() # close
return(colnames, my_data) # return column names AND data
Column name function for checking out what's in a table
def get_column_names(table_name): # arguement of table_name
conn, cur = get_conn_cur() # get connection and cursor
# Now select column names while inserting the table name into the WERE
column_name_query = """SELECT column_name FROM information_schema.columns
WHERE table_name = '%s' """ %table_name
cur.execute(column_name_query) # exectue
my_data = cur.fetchall() # store
cur.close() # close
0s completed at 11:25 AM

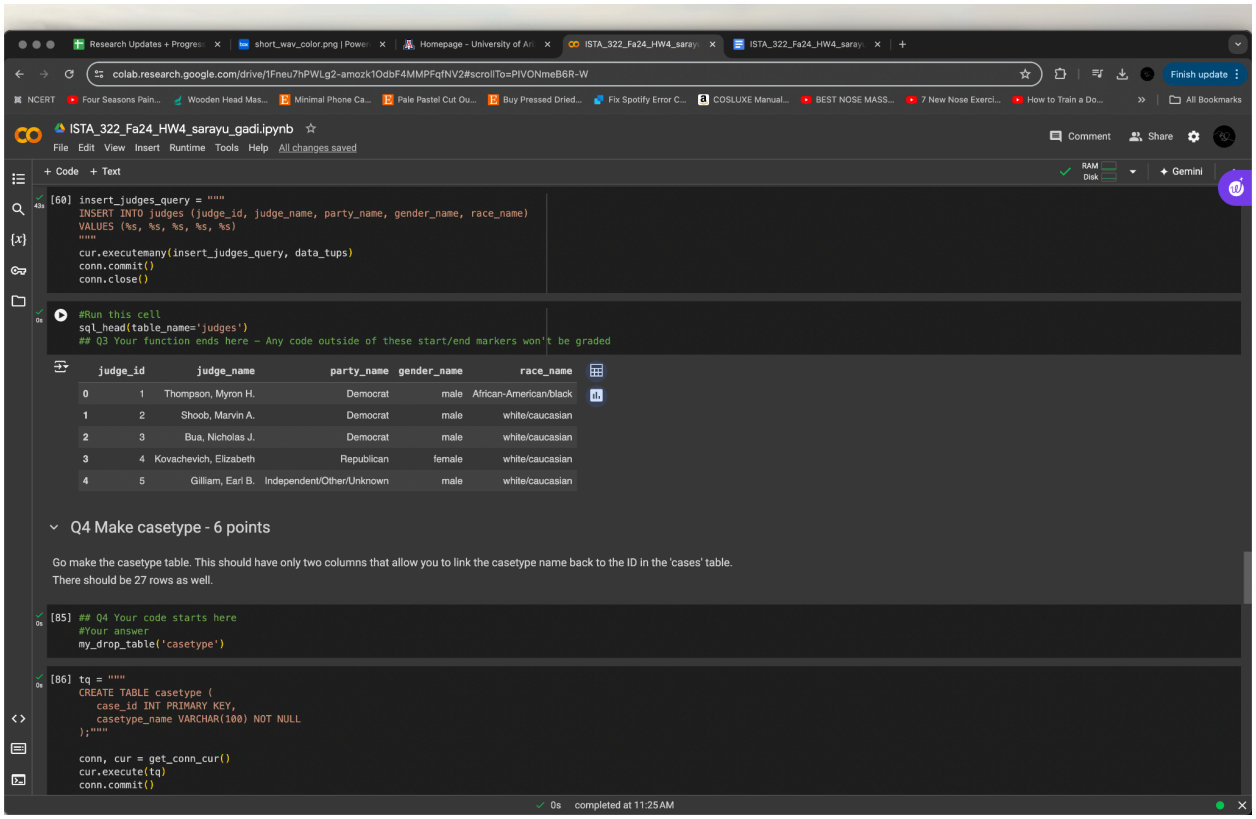


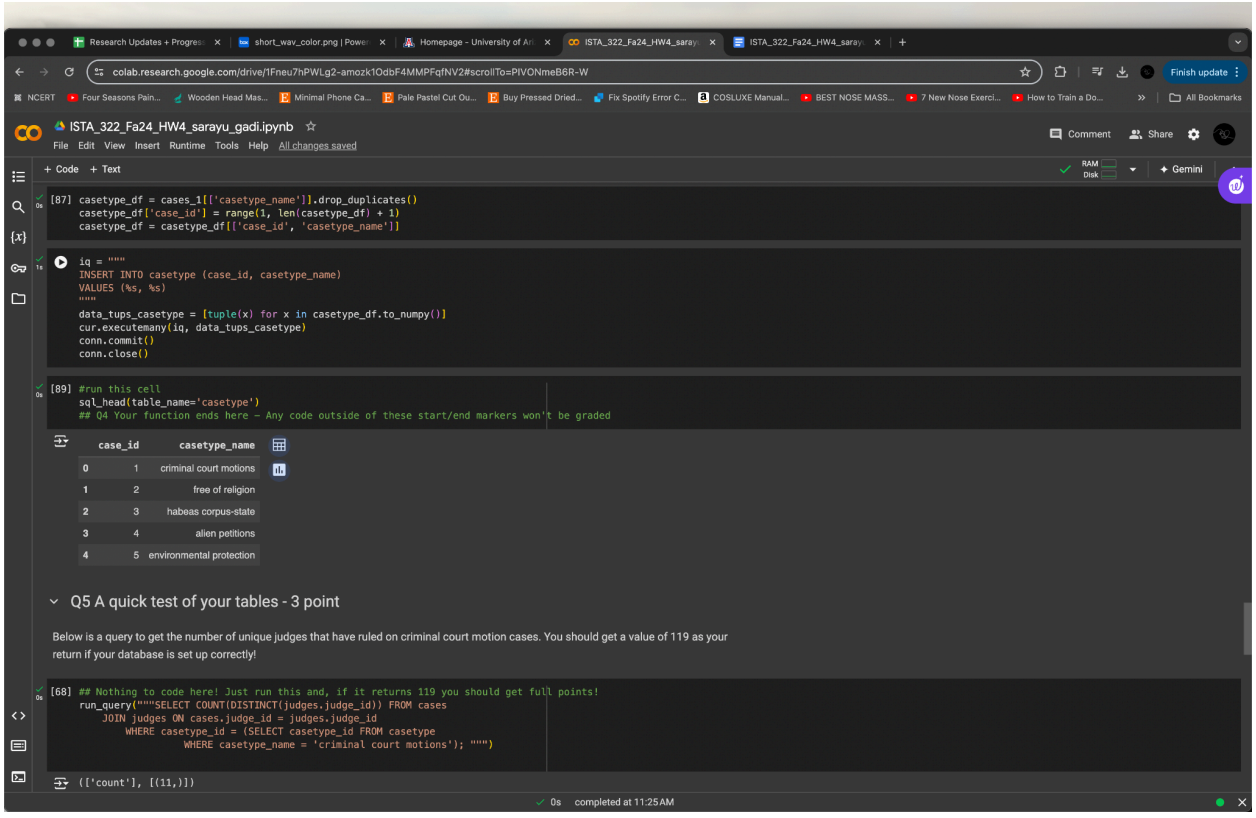
```
    conn.close() # close

    return(my_data) # return

# Check table_names
def get_table_names():
  conn, cur = get_conn_cur() # get connection and cursor

  # query to get table names
  table_name_query = """SELECT table_name FROM information_schema.tables
       WHERE table_schema = 'public' """

  cur.execute(table_name_query) # execute
  my_data = cur.fetchall() # fetch results

  cur.close() #close cursor
  conn.close() # close connection

  return(my_data) # return your fetched results

# make sql_head function
def sql_head(table_name):
  conn, cur = get_conn_cur() # get connection and cursor

  # Now select column names while inserting the table name into the WERE
  head_query = """SELECT * FROM %s LIMIT 5; """ %table_name

  cur.execute(head_query) # exectue
  colnames = [desc[0] for desc in cur.description] # get column names
  my_data = cur.fetchall() # store first five rows

  cur.close() # close
  conn.close() # close

  df = pd.DataFrame(data = my_data, columns = colnames) # make into df

  return(df) # return

### This is an extra function I'm giving you that allows you to drop tables from your RDB. This will be vital as you can only create your table once.
# If you try creating the same table when it already exists on your RDB, you'll get an error.
# I recommend calling this function one line above your code creating your table. eg for cases, you'd call it like this: my_drop_table('cases')
def my_drop_table(tab_name):
  conn, cur = get_conn_cur()
  tq = """DROP TABLE IF EXISTS %s CASCADE;""" %tab_name
  cur.execute(tq)
  conn.commit()
```

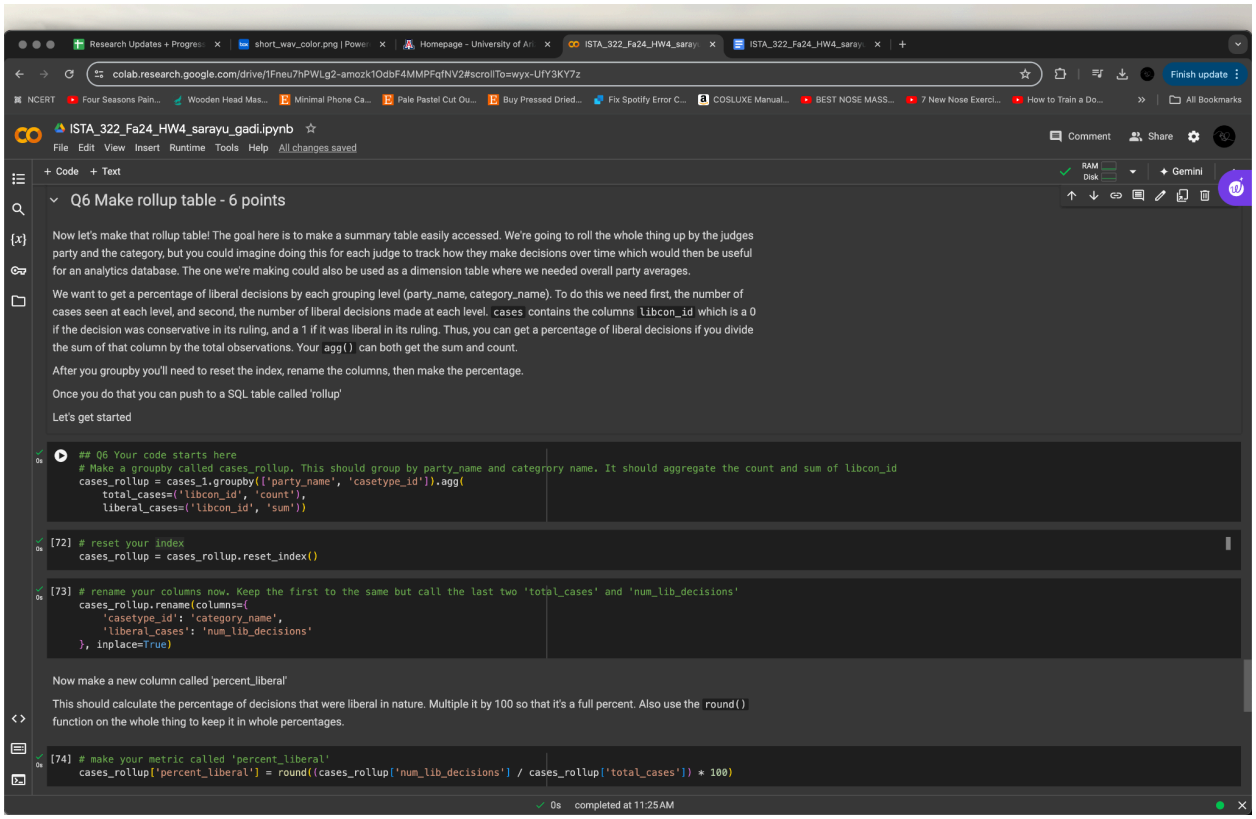


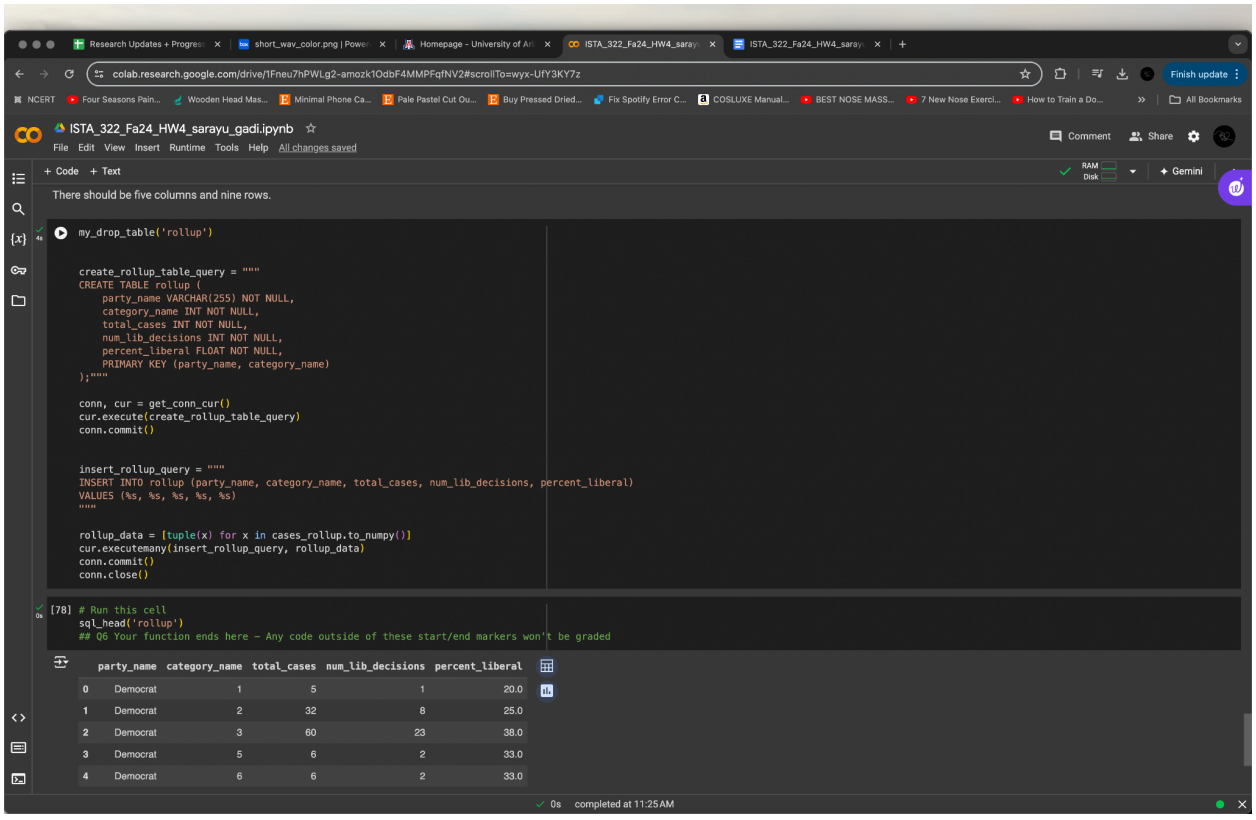
```
[9] my_drop_table('cases')

[10] tq = """CREATE TABLE cases (
        judge_id INT PRIMARY KEY,
        case_id BIGINT NOT NULL,
        case_year INT NOT NULL,
        casetype_id INT NOT NULL,
        category_id INT NOT NULL,
        category_name VARCHAR(255) NOT NULL
        );"""

    conn, cur = get_conn_cur()
    cur.execute(tq)
    conn.commit()

get_column_names('cases')

[('judge_id',),
 ('case_id',),
 ('case_year',),
 ('casetype_id',),
 ('category_id',),
 ('category_name',)]

[12] import numpy as np

    cases_df_np = cases_df.to_numpy();
    data_tups = [tuple(x) for x in cases_df_np]

[13] iq = """
       INSERT INTO cases (judge_id, case_id, case_year, casetype_id, category_id, category_name)
       VALUES (%s, %s, %s, %s, %s, %s)
    """

[14] conn, cur = get_conn_cur()
    cur.executemany(iq, data_tups)
    conn.commit()
    conn.close()

[15] # Use sql_head to check cases
    sql_head(table_name='cases')
    ## Q2 part 2 Your function ends here - Any code outside of these start/end markers won't be graded
```

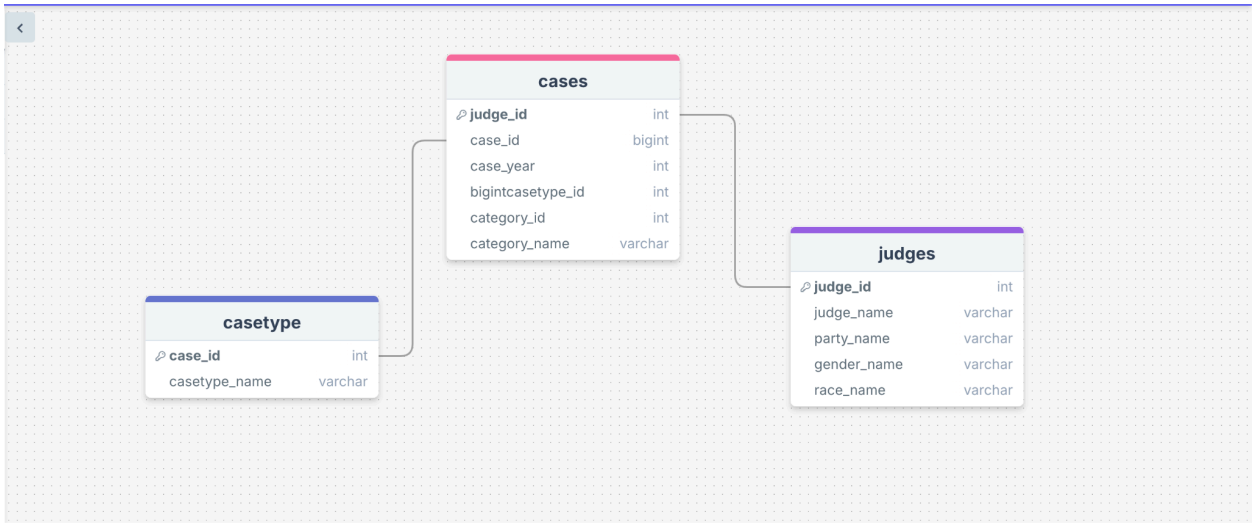

colab.research.google.com/drive/1Fneu7hPWLg2-amozk1OdbF4MMPFqfNV2#scrollTo=PIVONmeB6R-W
ISTA_322_Fa24_HW4_sarayu_gadi.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[15]
judge_id case_id case_year casetype_id category_id category_name
0 1 28321332 2011 3 1 Criminal Justice Cases
1 2 18110669 1993 14 2 Civil Liberties/Rights Cases
2 3 15660871 1983 2 1 Criminal Justice Cases
3 4 17770934 1991 2 1 Criminal Justice Cases
4 5 26621195 2009 6 2 Civil Liberties/Rights Cases
Q3 Make judges - 6 points
Now make your judges table from the original cases dataframe (not the SQL table you just made).
Judges should have five columns, including the judge_id column you made. There should be 553 rows after you drop duplicates (remember that judges may have had more than one case).
After you make the dataset go and push to a SQL table called 'judges'.
Q3 Your code starts here
#Your answer
my_drop_table('judges')
[58] tq = """
CREATE TABLE judges (
judge_id INT PRIMARY KEY,
judge_name VARCHAR(255) NOT NULL,
party_name VARCHAR(255) NOT NULL,
gender_name VARCHAR(255) NOT NULL,
race_name VARCHAR(255) NOT NULL
);"""
conn, cur = get_conn_cur()
cur.execute(tq)
conn.commit()
[59] columns_to_select = ['judge_id', 'judge_name', 'party_name', 'gender_name', 'race_name']
judges_df = cases_1[columns_to_select].drop_duplicates() # Ensure uniqueness
data_tups = [tuple(x) for x in judges_df.to_numpy()]
0s completed at 11:25AM


colab.research.google.com/drive/1Fneu7hPWLg2-amozk1OdbF4MMPFqfNV2#scrollTo=PIVONmeB6R-W
ISTA_322_Fa24_HW4_sarayu_gadi.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[60] insert_judges_query = """
INSERT INTO judges (judge_id, judge_name, party_name, gender_name, race_name)
VALUES (%s, %s, %s, %s, %s)
"""
cur.executemany(insert_judges_query, data_tups)
conn.commit()
conn.close()
#Run this cell
sql_head(table_name='judges')
Q3 Your function ends here — Any code outside of these start/end markers won't be graded
judge_id judge_name party_name gender_name race_name
0 1 Thompson, Myron H. Democrat male African-American/black
1 2 Shoob, Marvin A. Democrat male white/caucasian
2 3 Bua, Nicholas J. Democrat male white/caucasian
3 4 Kovachevich, Elizabeth Republican female white/caucasian
4 5 Gilliam, Earl B. Independent/Other/Unknown male white/caucasian
Q4 Make casetype - 6 points
Go make the casetype table. This should have only two columns that allow you to link the casetype name back to the ID in the 'cases' table.
There should be 27 rows as well.
[85] ## Q4 Your code starts here
#Your answer
my_drop_table('casetype')
[86] tq = """
CREATE TABLE casetype (
case_id INT PRIMARY KEY,
casetype_name VARCHAR(100) NOT NULL
);"""
conn, cur = get_conn_cur()
cur.execute(tq)
conn.commit()
0s completed at 11:25AM


ISTA_322_Fa24_HW4_sarayu_gadi.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[87] casetype_df = cases_1[['casetype_name']].drop_duplicates()
casetype_df['case_id'] = range(1, len(casetype_df) + 1)
casetype_df = casetype_df[['case_id', 'casetype_name']]
iq = """
INSERT INTO casetype (case_id, casetype_name)
VALUES (%s, %s)
"""
data_tups_casetype = [tuple(x) for x in casetype_df.to_numpy()]
cur.executemany(iq, data_tups_casetype)
conn.commit()
conn.close()
[89] #run this cell
sql_head(table_name='casetype')
Q4 Your function ends here - Any code outside of these start/end markers won't be graded
case_id casetype_name
0 1 criminal court motions
1 2 free of religion
2 3 habeas corpus-state
3 4 alien petitions
4 5 environmental protection
Q5 A quick test of your tables - 3 point
Below is a query to get the number of unique judges that have ruled on criminal court motion cases. You should get a value of 119 as your return if your database is set up correctly!
[68] ## Nothing to code here! Just run this and, if it returns 119 you should get full points!
run_query("""SELECT COUNT(DISTINCT(judges.judge_id)) FROM cases
JOIN judges ON cases.judge_id = judges.judge_id
WHERE casetype_id = (SELECT casetype_id FROM casetype
WHERE casetype_name = 'criminal court motions'); """)
(['count'], [(11,)])
0s completed at 11:25AM


ISTA_322_Fa24_HW4_sarayu_gadi.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Q6 Make rollup table - 6 points
Now let's make that rollup table! The goal here is to make a summary table easily accessed. We're going to roll the whole thing up by the judges party and the category, but you could imagine doing this for each judge to track how they make decisions over time which would then be useful for an analytics database. The one we're making could also be used as a dimension table where we needed overall party averages.
We want to get a percentage of liberal decisions by each grouping level (party_name, category_name). To do this we need first, the number of cases seen at each level, and second, the number of liberal decisions made at each level. cases contains the columns libcon_id which is a 0 if the decision was conservative in its ruling, and a 1 if it was liberal in its ruling. Thus, you can get a percentage of liberal decisions if you divide the sum of that column by the total observations. Your agg() can both get the sum and count.
After you groupby you'll need to reset the index, rename the columns, then make the percentage.
Once you do that you can push to a SQL table called 'rollup'
Let's get started
Q6 Your code starts here
# Make a groupby called cases_rollup. This should group by party_name and category name. It should aggregate the count and sum of libcon_id
cases_rollup = cases_1.groupby(['party_name', 'casetype_id']).agg(
total_cases=('libcon_id', 'count'),
liberal_cases=('libcon_id', 'sum'))
[72] # reset your index
cases_rollup = cases_rollup.reset_index()
[73] # rename your columns now. Keep the first to the same but call the last two 'total_cases' and 'num_lib_decisions'
cases_rollup.rename(columns={
'casetype_id': 'category_name',
'liberal_cases': 'num_lib_decisions'
}, inplace=True)
Now make a new column called 'percent_liberal'
This should calculate the percentage of decisions that were liberal in nature. Multiple it by 100 so that it's a full percent. Also use the round() function on the whole thing to keep it in whole percentages.
[74] # make your metric called 'percent_liberal'
cases_rollup['percent_liberal'] = round((cases_rollup['num_lib_decisions'] / cases_rollup['total_cases']) * 100)
0s completed at 11:25AM


colab.research.google.com/drive/1Fneu7hPWLg2-amozk1OdbF4MMPFqfNV2#scrollTo=wyx-UfY3KY7z
ISTA_322_Fa24_HW4_sarayu_gadi.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
There should be five columns and nine rows.
my_drop_table('rollup')
create_rollup_table_query = """
CREATE TABLE rollup (
party_name VARCHAR(255) NOT NULL,
category_name INT NOT NULL,
total_cases INT NOT NULL,
num_lib_decisions INT NOT NULL,
percent_liberal FLOAT NOT NULL,
PRIMARY KEY (party_name, category_name)
);"""
conn, cur = get_conn_cur()
cur.execute(create_rollup_table_query)
conn.commit()
insert_rollup_query = """
INSERT INTO rollup (party_name, category_name, total_cases, num_lib_decisions, percent_liberal)
VALUES (%s, %s, %s, %s, %s)
"""
rollup_data = [tuple(x) for x in cases_rollup.to_numpy()]
cur.executemany(insert_rollup_query, rollup_data)
conn.commit()
conn.close()
[78] # Run this cell
sql_head('rollup')
Q6 Your function ends here - Any code outside of these start/end markers won't be graded
party_name category_name total_cases num_lib_decisions percent_liberal
0 Democrat 1 5 1 20.0
1 Democrat 2 32 8 25.0
2 Democrat 3 60 23 38.0
3 Democrat 5 6 2 33.0
4 Democrat 6 6 2 33.0
0s completed at 11:25 AM


cases
judge_id int
case_id bigint
case_year int
bigintcasetype_id int
category_id int
category_name varchar
judges
judge_id int
judge_name varchar
party_name varchar
gender_name varchar
race_name varchar
casetype
case_id int
casetype_name varchar