HOMEWORK #6 SOLUTION - STUDENTS

Problem 1 (5 points):

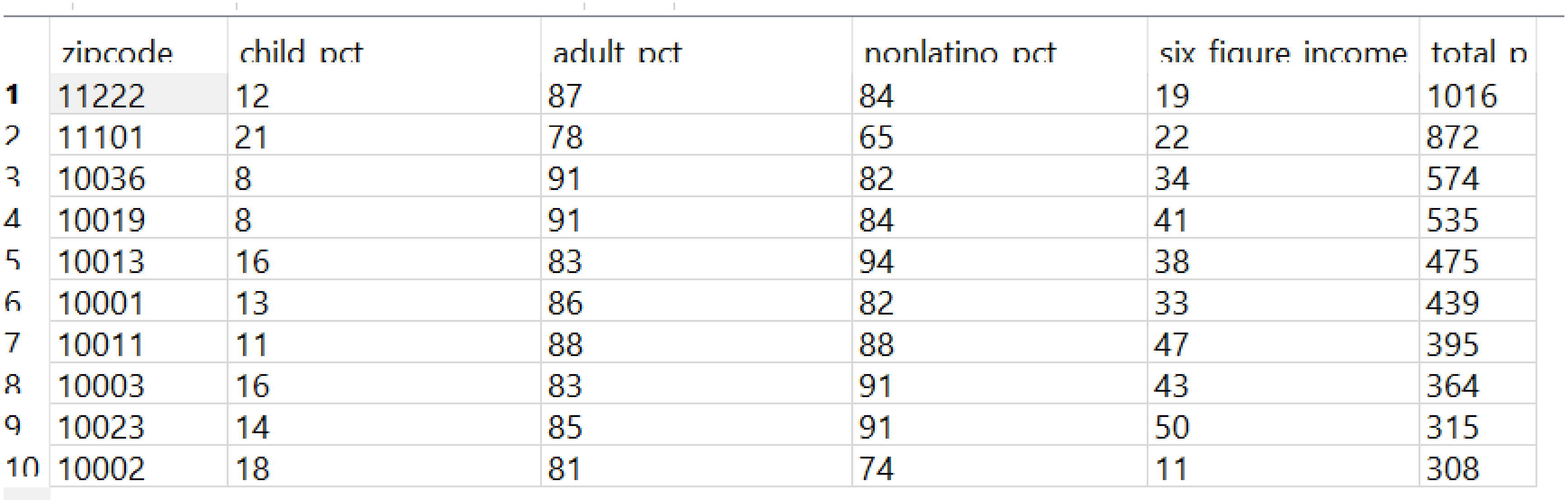
Find the top zip codes for filming permits in 2015 and the percentages of children, adults, non-latino individuals and six_figure_income returns for these zip codes. (HINT: you should have 6 columns: zipcode, 4 percentage columns, and a filming permit count column). Submit your SQL code along with its resulting output.

Code:

```
WITH flat_irs AS (
    SELECT
        year,
        zipcode,
        SUM(CASE WHEN agi_map_id IN (1,2,3,4) THEN return_count ELSE NULL END) AS agi_under100K,
        SUM(CASE WHEN agi_map_id IN (5,6) THEN return_count ELSE NULL END) AS agi_over100k,
        SUM(return_count) AS total_returns
    FROM irs_nyc_tax_returns
    GROUP BY
        year,
        zipcode
),

flat_films AS (
    SELECT
        STRFTIME('%Y',StartDateTime) AS year
        ,zipcode
        ,COUNT(DISTINCT EventID) as permit_count
    FROM nyc_film_permits
    GROUP BY
        year
        ,zipcode
)
```

```
SELECT
    cen.zipcode,
    100 * (SUM(cen.Age_under5) + SUM(cen.Age_5to9) + SUM(cen.Age_10to14) + SUM(cen.Age_15to19)) /
SUM(Age_all) AS child_pct,
    100 * (SUM(Age_all) - (SUM(cen.Age_under5) + SUM(cen.Age_5to9) + SUM(cen.Age_10to14) +
SUM(cen.Age_15to19) )) / SUM(Age_all) AS adult_pct,
    100 * (SUM(cen.Age_all) - SUM(cen.Ethnicity_All_HispancLatino_Descent)) / SUM(cen.Age_all) AS
nonlatino_pct,
    100 * SUM(irs.agi_over100k) / SUM(irs.total_returns) AS six_figure_income_pct,
    SUM(nfp.permit_count) as total_permits
FROM nyc_census_data cen
    INNER JOIN flat_irs irs
        ON cen.zipcode = irs.zipcode
    INNER JOIN flat_films nfp
        ON (
            cen.zipcode = nfp.zipcode
            AND irs.year = nfp.year
        )
WHERE
    irs.year = 2015
GROUP BY
    cen.zipcode
ORDER BY
    total_permits DESC
LIMIT 10;
```

Result:



	zipcode	child pct	adult pct	nonlatino pct	six figure income	total p
1	11222	12	87	84	19	1016
2	11101	21	78	65	22	872
3	10036	8	91	82	34	574
4	10019	8	91	84	41	535
5	10013	16	83	94	38	475
6	10001	13	86	82	33	439
7	10011	11	88	88	47	395
8	10003	16	83	91	43	364
9	10023	14	85	91	50	315
10	10002	18	81	74	11	308

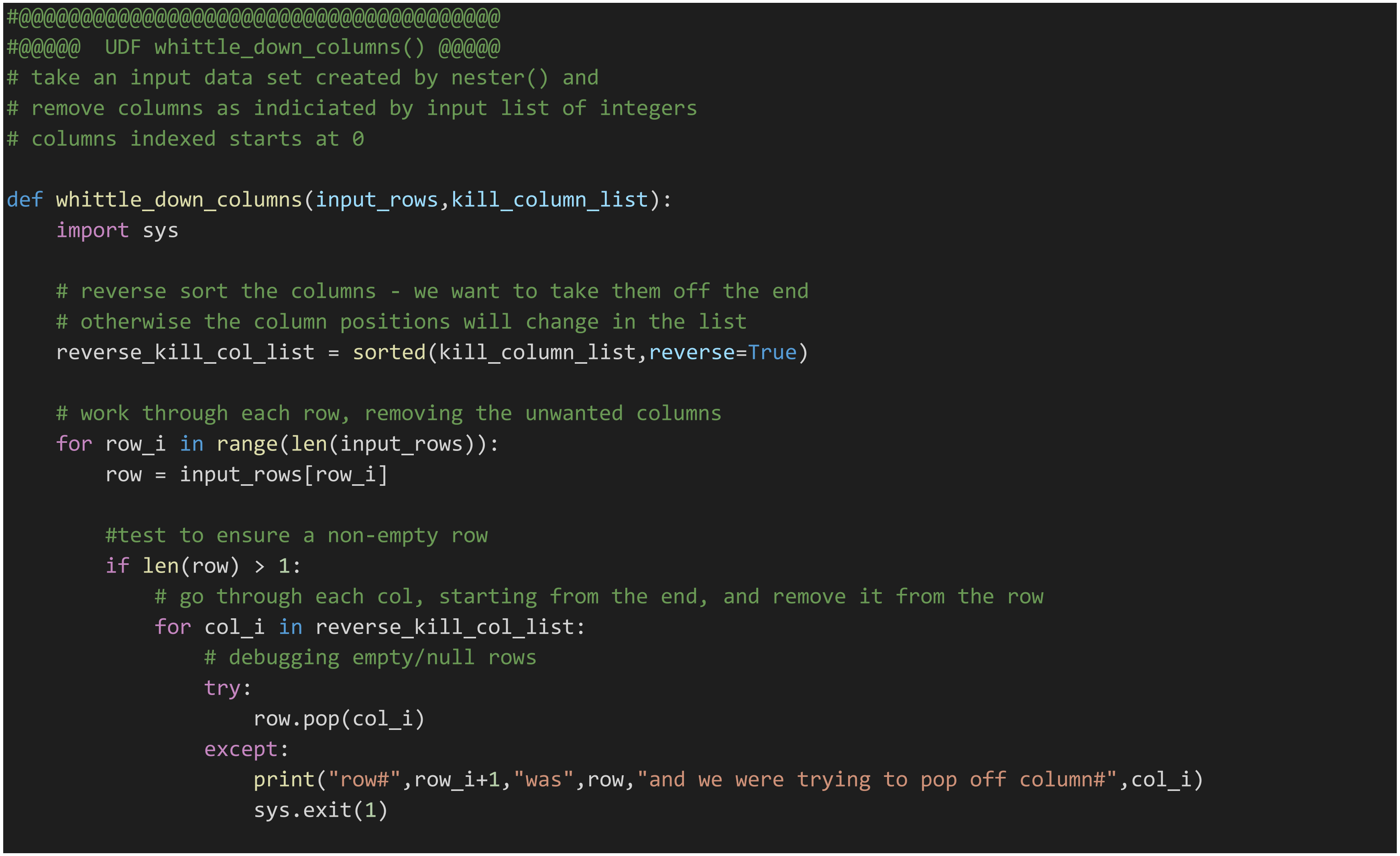
Problem 2 (4 points):

Create a User Defined Function that removes all other columns from the data set in the nested structure EXCEPT the following columns. The following 29 columns MUST remain in the dataset, and no others:

1. Id2
2. Number-SEX_AND_AGE_Total_population
3. Number-SEX_AND_AGE_Total_population_Under_5_years
4. Number-SEX_AND_AGE_Total_population_5_to_9_years
5. Number-SEX_AND_AGE_Total_population_10_to_14_years
6. Number-SEX_AND_AGE_Total_population_15_to_19_years
7. Number-SEX_AND_AGE_Total_population_20_to_24_years
8. Number-SEX_AND_AGE_Total_population_25_to_29_years
9. Number-SEX_AND_AGE_Total_population_30_to_34_years
10. Number-SEX_AND_AGE_Total_population_35_to_39_years
11. Number-SEX_AND_AGE_Total_population_40_to_44_years
12. Number-SEX_AND_AGE_Total_population_45_to_49_years
13. Number-SEX_AND_AGE_Total_population_50_to_54_years
14. Number-SEX_AND_AGE_Total_population_55_to_59_years
15. Number-SEX_AND_AGE_Total_population_60_to_64_years
16. Number-SEX_AND_AGE_Total_population_65_to_69_years
17. Number-SEX_AND_AGE_Total_population_70_to_74_years
18. Number-SEX_AND_AGE_Total_population_75_to_79_years
19. Number-SEX_AND_AGE_Total_population_80_to_84_years
20. Number-SEX_AND_AGE_Total_population_85_years_and_over
21. Number-SEX_AND_AGE_Male_population
22. Number-SEX_AND_AGE_Female_population
23. Number-RACE_Race_alone_or_in_combination_with_one_or_more_other_races-4-White
24. Number-RACE_Race_alone_or_in_combination_with_one_or_more_other_races-4-Black_or_African_American
25. Number-RACE_Race_alone_or_in_combination_with_one_or_more_other_races-4-American_Indian_and_Alaska_Native
26. Number-RACE_Race_alone_or_in_combination_with_one_or_more_other_races-4-Asian
27. Number-RACE_Race_alone_or_in_combination_with_one_or_more_other_races-4-Native_Hawaiian_and_Other_Pacific_Islander
28. Number-RACE_Race_alone_or_in_combination_with_one_or_more_other_races-4-Some_Other_Race

29. Number-HISPANIC_OR_LATINO_Total_population_Hispanic_or_Latino_of_any_race

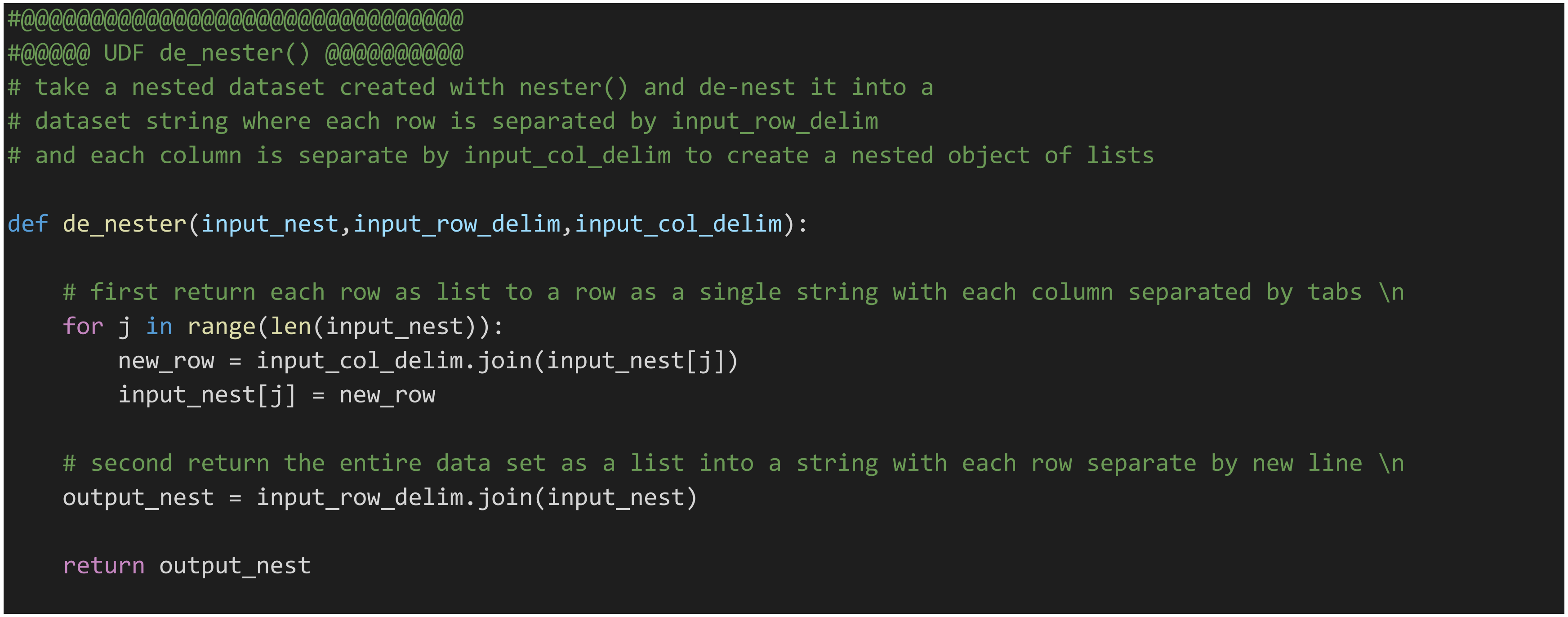
CODE



```
#@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
#@@@@@  UDF whittle_down_columns() @@@@@
# take an input data set created by nester() and
# remove columns as indiciated by input list of integers
# columns indexed starts at 0

def whittle_down_columns(input_rows,kill_column_list):
    import sys

    # reverse sort the columns - we want to take them off the end
    # otherwise the column positions will change in the list
    reverse_kill_col_list = sorted(kill_column_list,reverse=True)

    # work through each row, removing the unwanted columns
    for row_i in range(len(input_rows)):
        row = input_rows[row_i]

        #test to ensure a non-empty row
        if len(row) > 1:
            # go through each col, starting from the end, and remove it from the row
            for col_i in reverse_kill_col_list:
                # debugging empty/null rows
                try:
                    row.pop(col_i)
                except:
                    print("row#",row_i+1,"was",row,"and we were trying to pop off column#",col_i)
                    sys.exit(1)
                    # after each column is removed, write updated row back to file
            input_rows[row_i] = row
        else:
            continue

    return input_rows
```

Problem 3 (1 point):
Create a User Defined Function that transforms your nested data structure back into a string so it can be written to a file as a Comma-Separated-Values or Tab-Separate-Values data file.

CODE:



```python
#@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
#@@@@@ UDF de_nester() @@@@@@@@@@
# take a nested dataset created with nester() and de-nest it into a
# dataset string where each row is separated by input_row_delim
# and each column is separate by input_col_delim to create a nested object of lists

def de_nester(input_nest,input_row_delim,input_col_delim):

    # first return each row as list to a row as a single string with each column separated by tabs \n
    for j in range(len(input_nest)):
        new_row = input_col_delim.join(input_nest[j])
        input_nest[j] = new_row

    # second return the entire data set as a list into a string with each row separate by new line \n
    output_nest = input_row_delim.join(input_nest)

    return output_nest
```