Title: MarathonResultsAnalysis
Description: Build Postgres Database for analyzing results
Date: Completed 12/14/2024
Steps:

1. Gather all data from the website.
2. Clean data with python.
3. Pull into Postgres database and modify table as needed.
4. Create list of questions to investigate.
5. Build queries and views for analysis.

Data source: https://www.athlinks.com/event/35398/results/Event/1092637/Results

1. Data was copied and pasted page by page into text file. Below is what the raw data looks like.

```
Claim
Alexander Wilson
M 30Bib 470Golden, CO, USA
1
1
1
5:47
MIN/MI
2:31:37
Claim
Jesse Armijo
M 42Bib 53
2
2
1
6:21
MIN/MI
2:46:11
Claim
Willie Kinney
M 34Bib 281Albuquerque, NM, USA
```

Each entry is a total of nine lines, which makes the organization process easy with python.

2. Here is the python script for organizing the data in csv format.

```python
#raw source file.
f = open("Results.txt", "r")

#save each line into a list; ignore blank lines.
lines = []
for line in f:
    if line != "\n":
        lines.append(line[:-1])

#primary key
place = [i+1 for i in range(len(lines)//9)]
#extract other fields of interest
name = lines[slice(1,len(lines)+1,9)]
sexAge = lines[slice(2,len(lines)+1,9)]
sex = [entry[0] for entry in sexAge]
age = [entry[2:4] for entry in sexAge]
time = lines[slice(8,len(lines)+1,9)]

f.close()

#check that each field list is the same length.
print(len(place),len(name),len(sex),len(age),len(time))

#save output.
with open("Results_Cleaned.csv", "w") as output:
    output.write("place,name,sex,age,time\n")
    for i in range(len(time)):
        output.write(",".join([str(place[i]),name[i],sex[i],age[i],time[i]]))
        output.write("\n")
```

Below is a snapshot of the cleaned data set.

File Edit View

```
place,name,sex,age,time
1,Alexander Wilson,M,30,2:31:37
2,Jesse Armijo,M,42,2:46:11
3,Willie Kinney,M,34,2:46:52
4,Chris Michael,M,43,2:51:20
5,Xavier DELATORRE,M,33,2:54:15
6,Judson White,M,35,2:59:30
7,William Miles,M,52,3:00:00
8,Vincent DiMassa,M,31,3:00:34
9,Neil McLaughlin,M,30,3:03:40
10,Aaron Gamez,M,39,3:03:55
```

3. In postgres, we create a table to hold the data using appropriate data types.

```sql
CREATE TABLE year_2024 (
    place integer PRIMARY KEY,
    name text,
    sex char(1),
    age integer,
    racetime text);
```

When we try to copy the csv data into the table, we find an error.

```sql
COPY year_2024 (place,name,sex,age,racetime)
FROM 'C:\users\Public\Results_Cleaned.csv'
DELIMITER ','
CSV HEADER;
```

Data Output | Messages | Notifications

```
ERROR:  invalid input syntax for type integer: "4B"
CONTEXT:  COPY year_2024, line 59, column age: "4B"

SQL state: 22P02
```

This is an example of unexpected data from the age field. When I wrote the python script, I assumed that all runners would be at least 10 years old (exactly two digits for each age value). Apparently, the data set has errors, or there were truly a few runners who were as young as 4 years old. Further investigation would be needed, but for this project, I will remove these entries. This would be difficult if the file were millions of lines. So, one way to get around this is to re-create the table, saving the age field as TEXT type. Then, we will remove the problematic records, and change the data type to integer when it is cleaned.

```sql
DROP TABLE year_2024;
CREATE TABLE year_2024 (
    place integer PRIMARY KEY,
    name text,
    sex char(1),
    age text,
    racetime text);
```

And we run the copy statement again, this time with no errors!

```
COPY year_2024 (place,name,sex,age,racetime)
FROM 'C:\users\Public\Results_Cleaned.csv'
DELIMITER ','
CSV HEADER;
```

Data Output | Messages | Notifications

```
COPY 483

Query returned successfully in 77 msec.
```

Below is a section of the data.

```
SELECT * FROM year_2024
LIMIT 10;
```

Data Output | Messages | Notifications

	place [PK] integer	name text	sex character	age text	racetime text
1	1	Alexander Wilson	M	30	2:31:37
2	2	Jesse Armijo	M	42	2:46:11
3	3	Willie Kinney	M	34	2:46:52
4	4	Chris Michael	M	43	2:51:20
5	5	Xavier DELATORRE	M	33	2:54:15
6	6	Judson White	M	35	2:59:30
7	7	William Miles	M	52	3:00:00
8	8	Vincent DiMassa	M	31	3:00:34
9	9	Neil McLaughlin	M	30	3:03:40
10	10	Aaron Gamez	M	39	3:03:55

To find the problematic rows, let's select the second character of the age column and sort to find the anomalies.
Scanning through the result set, we find that the non-numeric second characters are at the end. We can now delete the problematic records.

```sql
SELECT place,SUBSTRING(age,2,1) AS second_char FROM year_2024
ORDER by second_char;
```

Data Output | Messages | Notifications

	place [PK] integer	second_char text
468	80	9
469	203	9
470	418	9
471	24	9
472	19	9
473	421	9
474	132	9
475	10	9
476	313	9
477	33	9
478	427	9
479	237	b
480	158	B
481	468	B
482	58	B
483	100	B

If this was a larger deletion set, we would need to use a subquery. But here, we can directly enter the places for the records we want to remove.

```sql
DELETE FROM year_2024
WHERE place IN (237,158,468,58,100);
```

Now, we are ready to change the age column to an integer type.

```sql
ALTER TABLE year_2024
ALTER COLUMN age TYPE integer USING age::integer;
```

Data Output | Messages | Notifications

```
ALTER TABLE

Query returned successfully in 60 msec.
```

For analysis, we want to include a new calculated column that contains the time values in numeric form. For this, I converted each time to its value in seconds.

```
ALTER TABLE year_2024
ADD COLUMN racetime_seconds integer;
UPDATE year_2024
SET racetime_seconds = 3600*CAST(SUBSTRING(racetime FROM 1 FOR 1) AS integer)
                       +  60*CAST(SUBSTRING(racetime FROM 3 FOR 2) as integer)
                       +  CAST(SUBSTRING(racetime FROM 6 FOR 1) as integer);
```

Data Output | Messages | Notifications

```
UPDATE 478

Query returned successfully in 51 msec.
```

Here is a snapshot of the transformed data set, sorted by age.

```
SELECT *
FROM year_2024
ORDER BY age
LIMIT 10;
```

Data Output | Messages | Notifications

	place [PK] integer	name text	sex character	age integer	racetime text	racetime_seconds integer
1	15	Ean Elias	M	15	3:10:40	11404
2	358	Brandon Hanert	M	18	5:25:56	19505
3	112	Sami Schuh	F	18	4:04:50	14645
4	293	Natalie Griebel	F	19	4:58:16	17881
5	281	Liam Kramer	M	19	4:56:35	17763
6	312	Anne Elias	F	19	5:07:05	18420
7	132	Emily Barker	F	19	4:09:55	14945
8	221	Conrad Black	M	19	4:39:16	16741
9	174	Chloe Bryson	F	20	4:23:31	15783
10	207	Kian Hill	M	20	4:33:14	16381

4. Now, we are ready to answer some questions. Here are a few ideas.
    a. What was the total time of the marathon (the range) in minutes and hours?
    b. How many men ran the race? How many women?
    c. What are the quartile time ranges? How fast did you need to run to be in the top ten percent?

The first several questions are easy to answer with simple queries. We find that the total recorded race time was about five and a half hours.

```
SELECT (MAX(racetime_seconds) - MIN(racetime_seconds))/3600 AS hours,
((MAX(racetime_seconds) - MIN(racetime_seconds))%3600)/60 AS minutes
FROM year_2024;
```

Data Output | Messages | Notifications

	hours integer	minutes integer
1	5	33

```
SELECT COUNT(*) AS count
FROM year_2024
GROUP BY sex;
```

Data Output | Messages | Notifications

	count bigint
1	331
2	147

And there were about twice as many males as females.

To answer the final questions, it is convenient to create a quantiles function that will enable us to show the time windows for any equally-divided set over time. For quartiles, we use 4, and for percentages, we use 10.
Below is the body of the function.

quantiles(bins integer)

General | Definition | Code | Options | Parameters | Security | SQL

```
1
2  BEGIN
3  RETURN QUERY
4      WITH cte AS
5      (SELECT racetime,
6       NTILE(bins) OVER (ORDER BY racetime) AS bin_number
7       FROM year_2024)
8
9      SELECT MIN(racetime), MAX(racetime), COUNT(*)::integer as runners_count, bin_number
10     FROM cte
11     GROUP BY bin_number
12     ORDER BY bin_number;
13 END;
14
```

The results desired are:

```
1  SELECT *
2  FROM quantiles(4);
```

Data Output | Messages | Notifications

	min_time text	max_time text	runners integer	bin integer
1	2:31:37	4:07:16	120	1
2	4:07:36	4:48:26	120	2
3	4:48:26	5:28:49	119	3
4	5:29:40	8:04:5	119	4

```
1  SELECT *
2  FROM quantiles(10);
```

Data Output | Messages | Notifications

	min_time text	max_time text	runners integer	bin integer
1	2:31:37	3:39:42	48	1
2	3:40:53	3:58:42	48	2
3	3:59:10	4:13:57	48	3
4	4:14:05	4:29:46	48	4
5	4:29:53	4:48:26	48	5
6	4:48:26	4:58:10	48	6
7	4:58:16	5:17:05	48	7
8	5:17:20	5:39:10	48	8
9	5:40:00	6:05:15	47	9
10	6:05:43	8:04:5	47	10

So, to be in the first quartile, you had to be under 4:07, and to be in the top 10%, you had to be under 3:39.

This concludes the case study.

—----------