STAT 243 Problem Set 5

Treves Li

2024-10-27

Collaboration Statement

I did not collaborate with anyone.

Question 1

This problem asks you to use Dask to process some Wikipedia traffic data and makes use of tools discussed in Section on October 11. The files in `/scratch/users/paciorek/wikistats/dated_2017_small/dated` (on the SCF) contain data on the number of visits to different Wikipedia pages on November 4, 2008 (which was the date of the US election in 2008 in which Barack Obama was elected). The columns are: date, time, language, webpage, number of hits, and page size. (Note that the Unit 7 Dask bag example and Question 2 below use a larger set of the same data.)

Question 1a

In an interactive session on the SCF Linux cluster (ideally on the `low` partition, but if necessary on the `high` partition), start an interactive session on one of the cluster nodes using `srun` (as discussed in section). Request four cores. Then follow these steps:

Question 1a: part i

> Copy the data files to a subdirectory of the `/tmp` directory of the machine your interactive session is running on. (Keep your code files in your home directory.) Putting the files on the local hard drive of the machine you are computing on reduces the amount of copying data across the network (in the situation where you read the data into your program multiple times) and should speed things up in step ii.

I copied the data into my `tmp` subdirectory below.

```
ssh -Y treves@gollum.berkeley.edu

# -t    10 minute time limit
# -N    Use 1 cluster node
# -c    Request 4 cores
# -p    Use low partition
# -mem  5G, based on provided tip in problem set
# --pty Create pseudo terminal for bash while srun is going
srun -t 00:15:00 -N 1 -c 4 -p low --mem-per-cpu=5G --pty /bin/bash

# Make /tmp parent directories as required (-p)
mkdir -p ~/tmp/treves/ps5

# Copy data files /tmp subdirectory
cp /scratch/users/paciorek/wikistats/dated_2017_small/dated/* ~/tmp/treves/ps5
```

Question 1a: part ii

> Write efficient Python code to do the following: Using the `dask` package as seen in Unit 6, with either `map` or a list of delayed tasks, write code that, in parallel, reads in the space-delimited files and filters to only the rows that refer to pages where "Barack_Obama" appears in the page title (column 4). You can use the code from Unit 6 as a template. Collect all the results into a single data frame. In your `srun` invocation and in your code, please use four cores in your parallelization so that other cores are saved for use by other users/students. IMPORTANT: before running the code on the full set of data, please test your code on a small subset first (and test your function on a single input file serially).

I first tested out a script on just one of the `.gz` files. This was to understand both the nature of the data I was working with, and to ensure my code runs smoothly before extrapolating the process to all the other files.

I set up Dask such that I have `n_workers = 4`, which results in four cores being used. It turns out that I had to wrap/guard the Dask code under `if __name__ == '__main__':` (credit to João/Chris for helping me with this). Apparently, without this guard, the process may re-run the main script recursively and potentially create new Dask clients, clusters, or tasks ad infinitum.

Some addiitonal error checking was included to handle empty entries, rows with missing columns, etc.

I used the `map` method to read and filter the `.gz` files in parallel, which in turn calls the function `read_wiki_gz`. The output is collated in a single dataframe `obama_collated_df`.

```
import os
import glob
import csv
import pandas as pd
import dask.dataframe as dd
from dask.distributed import Client


# Define the column names in the wiki data
col_names = ["date", "time", "lang", "page", "num_hits", "pg_size"]


def read_wiki_gz(gz_filepaths):
```

```
"""
Reads and process Wikipedia traffic data in .gz format,
then finds entries related to "Barack_Obama".

Parameters:
gz_filepaths (str): Path to .gz files with Wiki data.

Returns:
pd.DataFrame: Df where "Barack_Obama" was in the wiki page title.
"""

df = dd.read_csv(
  gz_filepaths,
  sep=" ",                # space delimited (given)
  names=col_names,        # column names as above (given)
  compression="gzip",     # define compression type
  blocksize=None,         # read file in one go
  low_memory=False,       # use more memory
  quoting=csv.QUOTE_NONE, # treat all characters literally
  escapechar="\\",        # define escapechar for special chars
  on_bad_lines="skip",    # skip bad lines (fewer than 6 fields)
  dtype="str",
)

# Sanity check that the reading worked
# print(df.head(10))
# print(f"Processing file: {gz_filepaths}")

if "page" in df.columns:
  # Handle NaN values
  df["page"] = df["page"].fillna("").astype(str)

  # Query for Barry-O
  obama_df = df[df["page"].str.contains("Barack_Obama")]

  # If entries were found, return as pandas dataframe
  if len(obama_df) > 0:
    return obama_df.compute()

# Return None if no valid 'page' column or entries found
return None
```

```
if __name__ == "__main__":
  # Create Dask client with four cores, as required
  client = Client(n_workers=4)

  # Test for just one file
  # gz_filepaths = [os.path.expanduser("~/tmp/treves/ps5/part-00122.gz")]

  # Use glob to retrieve all .gz files
  gz_filepaths = glob.glob(os.path.expanduser("~/tmp/treves/ps5/*.gz"))

  # Map the processing function to the tmp file paths
  tasks = client.map(read_wiki_gz, gz_filepaths)

  # Gather results from all tasks
  results = client.gather(tasks)

  # Sanity check if we're getting valid results
  # print(f"Number of results: {len(results)}")

  # Concatenate all filtered DataFrames into a single Dask df
  obama_collated_df = pd.concat(results, axis=0)

  # Export df to csv
  save_location = os.path.expanduser("~/ps5/obama_wiki.csv")
  obama_collated_df.to_csv(save_location, index=False)

  # Sanity check that concatenation worked
  print(f"Rows found with 'Barack_Obama': {len(obama_collated_df)}\n")
  pd.set_option("display.max_columns", None)  # Show all columns
  print(obama_collated_df.iloc[:, 2:5].head(20))
```

I invoke the code above with `srun`:

```
# -t    15 minute time limit
# -N    Use 1 cluster node
# -c    Request 4 cores
# -p    Use low partition
# -mem  5G, based on provided tip in problem set
# -J    job name

srun -t 00:15:00 -N 1 -c 4 -p low --mem-per-cpu=5G -J \
ps5-q1a_ii python ~/ps5/q1a_ii.py
```

```
Rows found with 'Barack_Obama':  8274


        lang                                                page num_hits
7030      en    Duncans.Tv_And_Hillary_Clinton_And_Barack_Obama        1
12553     eo               Dosiero:Sen._Barack_Obama_smiles.jpg        1
33557     en                 Media:En-Barack_Obama-article1.ogg       17
35410     en           Spielberg_not_DDB_Group_or_Barack_Obama        1
40175     hr                              Slika:Barack_Obama.jpg        1
105339    eu                                        Barack_Obama        3
121437  en.s       Statement_from_Barack_Obama_on_Darfur,_Sudan        1
142044    vi                          H%C3%ACnh:Barack_Obama.jpg        1
151744    en       Special:WhatLinksHere/Family_of_Barack_Obama        1
163072    yo                                        Barack_Obama        5
224842    fr                              Image:Barack_Obama.jpg       27
224843    fr  Image:Barack_Obama_and_supporters_5,_February_...       14
297989    ku                                        Barack_Obama        5
304258    bs                                        Barack_Obama        6
329341    br                                        Barack_Obama        6
331795  en.n        Talk:Grandmother_of_Barack_Obama_dies_at_86        1
332052  en.n  US_candidate_Barack_Obama_announces_Joe_Biden_...        5
332060  en.n  US_presidential_candidate_Barack_Obama%27s_lea...        1
332593  en.n  Wikinews:Story_preparation/Barack_Obama_wins_U...        5
332666  en.n  Wikinews_talk:Story_preparation/Barack_Obama_w...        1
Name: page, dtype: object
```

Question 1a: part iii

> Tabulate the number of hits for each hour of the day and make a (time-series) plot showing how the number of visits varied over the day (I don't care how you do this - using either Python or R is fine.). Note that the time zone is UTC/GMT, so you won't actually see the evening times when Obama's victory was announced - we'll see that in Question 2. Feel free to do this step outside the context of the parallelization. You'll want to use datetime functions from Pandas or the datetime package to manipulate the timing information (i.e., don't use string manipulations).

I exported the dataframe I got from part ii, and copied it to my local machine:

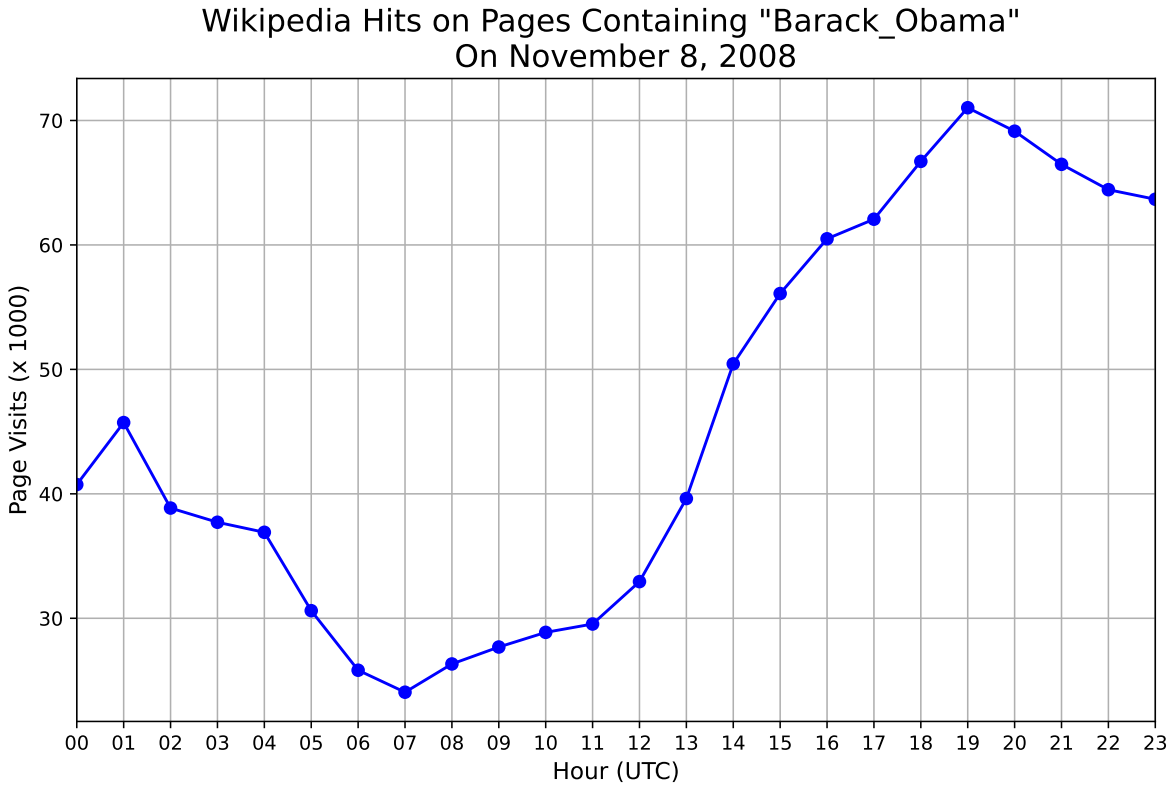
```
scp treves@gollum.berkeley.edu:~/ps5/obama_wiki.csv ~/treves/ps5/obama_wiki.csv
```

Now back to Python on my machine, so that I can group and plot the data:

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Read in the Obama wiki data
df = pd.read_csv('obama_wiki.csv', dtype={'time': 'str'})

# Manipulate timing information
df['time'] = pd.to_datetime(df['time'], format='%H%M%S')
df['time'] = df['time'].dt.strftime('%H:%M')

# Group by time/hour, sum by page hits
df = df.groupby("time")["num_hits"].sum()
print(df)

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(df.index, df.values/1000, marker='o', linestyle='-', color='b')

# Set plot labels and title
plt.title(
  'Wikipedia Hits on Pages Containing "Barack_Obama" \n\
  On November 8, 2008', fontsize=16)
```

```
plt.xlabel('Hour (UTC)', fontsize=12)
plt.ylabel('Page Visits (x 1000)', fontsize=12)

# Configure x-axis for hourly ticks from 00 to 23
plt.xticks(range(24), [f"{str(i).zfill(2)}" for i in range(24)])
plt.xlim(0, 23)

# Show the plot
plt.grid(True)
plt.show()
```

```
time
00:00    40744
01:00    45727
02:00    38857
03:00    37716
04:00    36909
05:00    30611
06:00    25826
07:00    24059
08:00    26330
09:00    27690
10:00    28868
11:00    29539
12:00    32946
13:00    39622
14:00    50446
15:00    56094
16:00    60494
17:00    62067
18:00    66715
19:00    71025
20:00    69140
21:00    66476
22:00    64439
23:00    63670
Name: num_hits, dtype: int64
```


Wikipedia Hits on Pages Containing "Barack_Obama"
On November 8, 2008
Page Visits (x 1000)
70
60
50
40
30
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Hour (UTC)

Question 1a: part iv

Remove the files from `/tmp`.

Done.

```
rm -rf ~/tmp/treves/ps5
```

Question 1b

> Now replicate steps (i) and (ii) but using `sbatch` to submit your job as a batch job to the SCF Linux cluster, where step (ii) involves running Python from the command line (e.g., `python your_file.py`. You don't need to make the plot again. As discussed here in the Dask documentation, put your Python/Dask code inside an if `__name__ == '__main__'` block.
>
> Note that you need to copy the files to `/tmp` in your submission script, so that the files are copied to `/tmp` on whichever node of the SCF cluster your job gets run on. Make sure that as part of your `sbatch` script you remove the files in `/tmp` at the end of the script. (Why? In general `/tmp` is cleaned out when a machine is rebooted, but this might take a while to happen and many of you will be copying files to the same hard drive so otherwise `/tmp` could run out of space.)

The bash script below takes the inputs I had previously used with `srun`, but this time I use `sbatch`. I made sure to follow the instructions on copying the files in `/tmp`, then deleting them as required after the task is completed. I submit the job to Slurm using `sbatch q1b_submit.sh`.

Also note that I had to change my code in part a so that it read the Wikipedia data from a relative filepath (i.e., from the `\tmp` subdirectory in the cluster node, instead of from my home folder on the login node). Other than that, the code is the same as what I wrote in part a. Indeed, the output is also the same as in part a.

```
#!/bin/bash

################
# SBATCH OPTIONS
################

#SBATCH --partition=low   # use low partition (optional)
#SBATCH --error=q1b.error # file for stderr (optional)
#SBATCH --output=q1b.out  # file for stdout (optional)
#SBATCH --time=00:15:00   # max runtime of job hr:min:sec
#SBATCH --nodes=1         # use 1 cluster node
#SBATCH --ntasks=1        # use 1 task
#SBATCH --cpus-per-task=4 # use 4 CPU cores
#SBATCH --mem-per-cpu=5G  # memory per CPU core

###################
```

```
# Commands to run
###################

# Create tmp dir on cluster node, and copy files
TMP_DIR=/tmp/treves/ps5
mkdir -p $TMP_DIR
cp /scratch/users/paciorek/wikistats/dated_2017_small/dated/* $TMP_DIR/

# Run Python script
python ~/ps5/q1b.py

# Remove tmp files
rm -rf $TMP_DIR
```

```
Rows found with 'Barack_Obama': 8274

        lang                                                page num_hits
10597     en  User:Giftlite/List_of_Nobel_laureates_who_endo...        1
11261     en        User:Jfire/Kenyan_relatives_of_Barack_Obama        1
41223     lb             Spezial:Export%C3%A9ieren/Barack_Obama        1
41250     lb                         Spezial:Export/Barack_Obama        1
41383     lb                         special:export/Barack_Obama        1
49253   en.d                         Special:Search/Barack_Obama        1
79865     en  Special:Export/Early_life_and_career_of_Barack...        1
104139    es                  Especial:Buscar/Barack_Obama,_Sr.        1
122921    en  Image:Ted_Kennedy_at_Barack_Obama_rally,_Febru...        3
200959    fi                  Kuva:Sen._Barack_Obama_smiles.jpg        3
214668    en       Cultural_and_political_image_of_Barack_Obama        1
279094  en.s                                 Author:Barack_Obama        3
279367  en.s                                        Barack_Obama        2
279368  en.s                        Barack_Obama%27s_Iraq_Speech        9
279369  en.s  Barack_Obama's_Announcement_to_Run_for_U.S._Pr...        2
279370  en.s                          Barack_Obama's_Iraq_Speech       30
323600    ur  %D8%AA%D8%B5%D9%88%DB%8C%D8%B1:Barack_Obama_at...        1
324163    uz                                        Barack_Obama        2
336590    en                Steady_and_America_and_Barack_Obama        1
343679    it                            Discussione:Barack_Obama        4
```

Question 2

> Consider the Wikipedia traffic data for October 15-November 15, 2008 (already available in `/var/local/s243/wikistats/dated_2017_sub` on all of the SCF cluster nodes in the low or high partitions). As in Question 1, explore the variation over time in the number of visits to Barack Obama-related Wikipedia sites, based on searching for "Barack_Obama" on English language Wikipedia pages. You should use Dask with distributed data structures to do the reading and filtering, as seen in Unit 7. Then group by day-hour (it's fine to do the grouping/counting in Python in a way that doesn't use Dask data structures). You can do this either in an interactive session using `srun` or a batch job using `sbatch`. And if you use `srun`, you can run Python itself either interactively or as a background job. Time how long it takes to do the Dask part of the computations to get a sense for how much time is involved working with this much data. Once you have done the filtering and gotten the counts for each day-hour, you can simply use standard Python or R code on your laptop to do some plotting to show how the traffic varied over the days of the full month-long time period and particularly over the hours of November 3-5, 2008 (election day was November 4 and Obama's victory was declared at 11 pm Eastern time on November 4).

For this question, I relied on the example Dask code provided in Unit 7 of the course notes. I ensured that I tested my code on a subset of the Wikipedia data first, before submitting my job for the whole ~40 GB dataset.

Similar to the course note examples, I created a function `find()` to look for pages related to "Barack_Obama" on *English* wiki pages. The matches were read and filtered using Dask bags, and the computation was timed accordingly (it took 12 minutes using 16 cores, as seen in the output below).

```
import os
import re
import time
import pandas as pd
import dask.multiprocessing
import dask.bag as db

# Use SLURM environment variable to set the number of workers
# Default to 16 cores if not set
num_workers = int(os.environ.get("SLURM_CPUS_PER_TASK", 16))
dask.config.set(scheduler="processes", num_workers=num_workers)
```

```
# Set global file paths
path = "/var/local/s243/wikistats/dated_2017_sub/"                    # Full path
# path = "/scratch/users/paciorek/wikistats/dated_2017_small/dated/" # Demo path


# Search for "Barack_Obama" on English language Wikipedia pages
def find(line, regex="Barack_Obama"):
  """
  Search for "Barack_Obama" on English language Wikipedia pages

  Parameters:
  line (str): A space-delimited string of wiki page data.
  regex (str): Pattern to find "Barack_Obama".

  Returns:
  bool: True if the conditions are met, False otherwise.
  """

  # Split into fields,; data is space-delimited
  vals = line.split(" ")

  # Error if number of fields is less than 6
  if len(vals) < 6:
    return False

  # We only want English language wiki pages
  if vals[2] != "en":
    return False

  tmp = re.search(regex, vals[3])

  # Return True if a match is found, else False
  return tmp is not None


def make_tuple(line):
  """Convert a space-delimited string into a tuple."""
  return tuple(line.split(" "))


if __name__ == "__main__":
  # Read the data in using Dask bags
```

```
  wiki = db.read_text(path + "part-0*gz")

  # Filter for "Barry-O" using Dask bags
  matches = wiki.filter(find)

  # Convert tuple Dask bag to a Dask df for grouping and summarization
  dtypes = {
    "date": "object",
    "time": "object",
    "language": "object",
    "webpage": "object",
    "hits": "float64",
    "size": "float64",
  }
  df = matches.map(make_tuple).to_dataframe(dtypes)

  # Do the computation and time it
  start_time = time.time()
  result = df.compute()  # Return **Pandas** (not Dask) df to the main process.
  end_time = time.time()

  # Calculate the computation duration
  duration = end_time - start_time
  minutes, seconds = divmod(duration, 60)
  print(f"Compute time: {int(minutes)}:{int(seconds):02d} mins.\n")

  # Export df to csv
  save_location = os.path.expanduser("~/ps5/obama_wiki_extended.csv")
  result.to_csv(save_location, index=False)

  # Sanity check the code worked correctly
  print(f"Rows found with 'Barack_Obama': {len(result)}\n")
  pd.set_option("display.max_columns", None)  # Show all columns
  print(result.iloc[:, 2:5].head(20))
```

I ran the code above using `sbatch`:

```
#!/bin/bash

#################
# SBATCH OPTIONS
#################
```

```
#SBATCH --error=q2.err      # file for stderr (optional)
#SBATCH --output=q2.out     # file for stdout (optional)
#SBATCH --time=00:50:00     # max runtime of job hr:min:sec
#SBATCH --nodes=1           # use 1 node
#SBATCH --ntasks=1          # use 1 task
#SBATCH --cpus-per-task=16  # use 16 CPU core

####################
# Command(s) to run
####################

python q2.py
```

```
Compute time: 11:57 mins.

Rows found with 'Barack_Obama': 62781
```

	language	webpage	hits
0	en	Special:RecentChangesLinked/Image:Barack_Obama...	3.0
1	en	The_Case_Against_Barack_Obama	3.0
2	en	Illinois_Senate_career_of_Barack_Obama	18.0
3	en	User:Giftlite/List_of_Nobel_laureates_who_endo...	1.0
4	en	User:Jfire/Kenyan_relatives_of_Barack_Obama	1.0
5	en	Special:Filepath/Barack_Obama_vs._John_McCain_...	1.0
6	en	Special:Filepath/Barack_Obama_vs._John_McCain_...	1.0
7	en	Special:Filepath/Barack_Obama_vs._John_McCain_...	1.0
8	en	Special:PrefixIndex/Barack_Obama_pictures	1.0
9	en	Republican_and_conservative_support_for_Barack...	2.0
10	en	Category:Barack_Obama	12.0
11	en	Illinois_Senate_career_of_Barack_Obama	32.0
12	en	Talk:List_of_Barack_Obama_presidential_campaig...	1.0
13	en	Talk:Public_image_of_Barack_Obama	1.0
14	en	Image:Barack_Obama.jpg	100.0
15	en	Image:Barack_Obama_Sr_Jr.jpg	33.0
16	en	Image:Barack_Obama_and_Ted_Kennedy_in_Hartford...	4.0
17	en	Image:Barack_Obama_in_New_Hampshire.jpg	1.0
18	en	Image:Barack_Obama_signature.svg	36.0
19	en	Template_talk:Barack_Obama	1.0

To plot the data, I again copied the `csv` output, as well as the error log and the ouput log, from the SCF to my local machine:

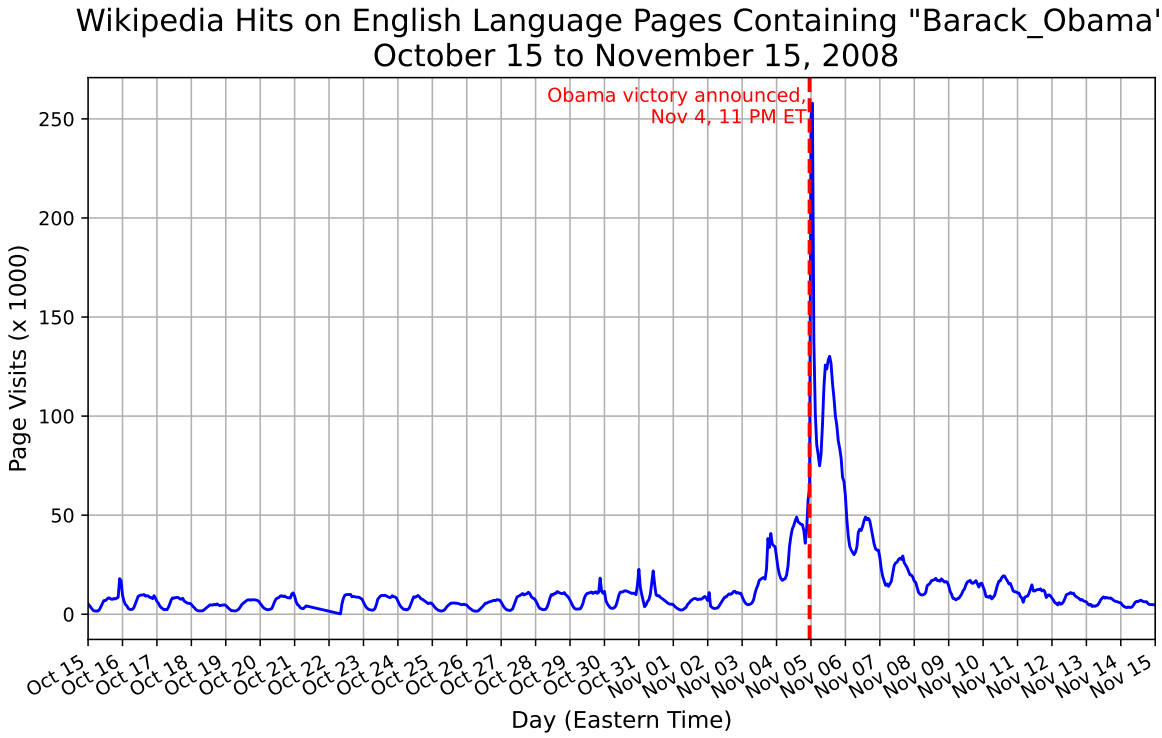
```
scp treves@gollum.berkeley.edu:~/ps5/{obama_wiki_extended.csv,q2.out,q2.err} \
  ~/treves/ps5/
```

The grouping and plotting was executed with the following code, which makes use of `panda`'s `groupby()` method to group the page counts by day-hour. Compared to Question 1, I now converted all the times to Eastern Time (using the `pytz` library), so it would be easier to identify the number of page hits and Obama's victory declaration at 11 pm Eastern Time.

Indeed, the chart below shows a clear spike of Wikipedia hits when Obama's victory was announced. Interestingly, there is a second (albeit smaller) spike around midday of November 5. Further below, we can zoom in on that time to see if we can glean any further information.

```
# Read in the Obama wiki data
dtypes = {"date": "str", "time": "str"}
df = pd.read_csv("obama_wiki_extended.csv", dtype=dtypes)

# Manipulate timing information
df["time"] = pd.to_datetime(df["time"], format="%H%M%S")
df["time"] = df["time"].dt.strftime("%H:%M")

# Manipulate date information
df["date"] = pd.to_datetime(df["date"], format="%Y%m%d")
df["date"] = df["date"].dt.strftime("%Y-%m-%d")

# Combine 'date' and 'time', localise to UTC
df["datetime"] = pd.to_datetime(df["date"] + " " + df["time"])
df["datetime"] = df["datetime"].dt.tz_localize("UTC")

# Convert timezone to ET, and fix format for ET
df["datetime_ET"] = df["datetime"].dt.tz_convert("America/New_York")
df["datetime_ET"] = df["datetime_ET"].dt.tz_localize(None)

# Group by day-hour
df_grouped = df.groupby("datetime_ET")[["hits"]].sum()
# print(df_grouped)

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(df_grouped.index, df_grouped["hits"].values / 1000, color="b")

# Set plot labels and title
plt.title(
```

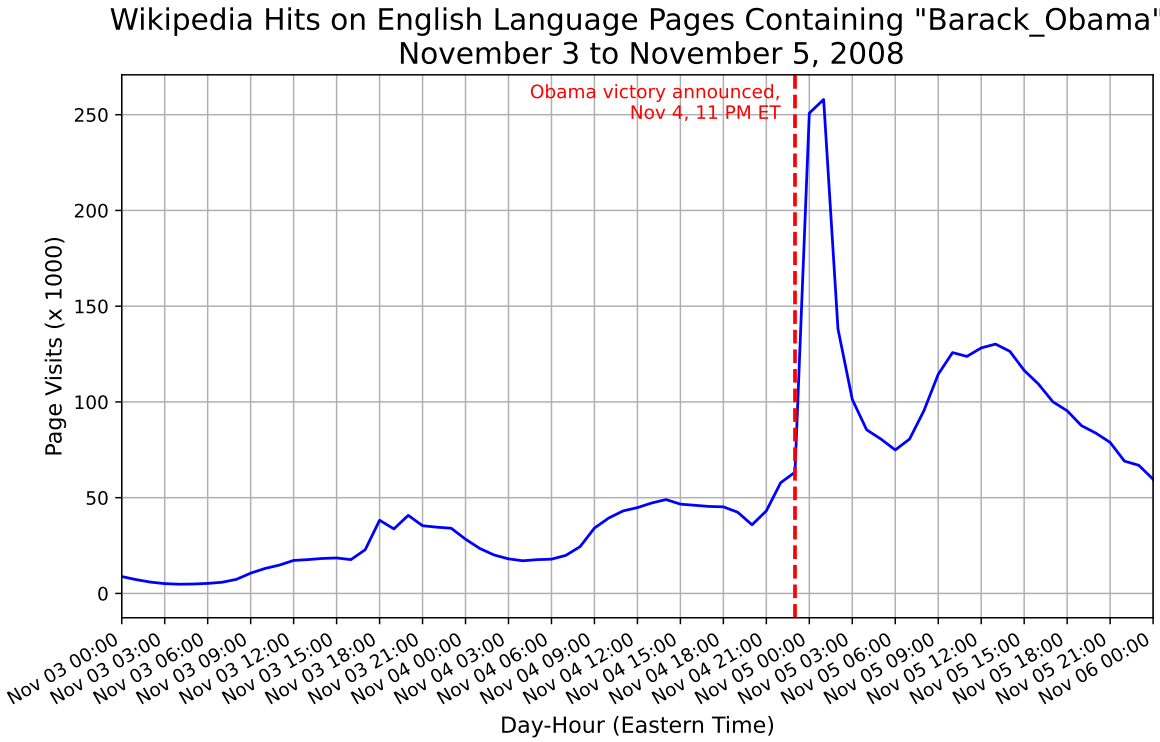
```
  'Wikipedia Hits on English Language Pages Containing "Barack_Obama"\n \
  October 15 to November 15, 2008',
  fontsize=16,
)
plt.xlabel("Day (Eastern Time)", fontsize=12)
plt.ylabel("Page Visits (x 1000)", fontsize=12)

# Set x-ticks start and end
start_date = pd.Timestamp("2008-10-15 00:00:00")
end_date = pd.Timestamp("2008-11-15 00:00:00")
plt.xlim(start_date, end_date)

# Make every tick at 00:00
plt.gca().xaxis.set_major_locator(mdates.DayLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%b %d"))

# Highlight Obama victory (Eastern Time)
obama_victory = pd.Timestamp("2008-11-04 23:00:00")
plt.axvline(obama_victory, color="red", linestyle="--", linewidth=2)
plt.text(
  obama_victory - pd.Timedelta(hours=2),
  plt.ylim()[1] - 25,
  "Obama victory announced,\nNov 4, 11 PM ET",
  color="red",
  fontsize=10,
  ha="right",
  va="bottom",
)

# Show plot
plt.grid(True)
plt.gcf().autofmt_xdate() # Auto-format date
plt.show()
```



We can zoom in over the hours of November 3-5, 2008, to have a closer look at the number of page hits around Election Day and around the victory declaration.

```
# Create the day-hour plot
plt.figure(figsize=(10, 6))
plt.plot(df_grouped.index, df_grouped["hits"].values / 1000, color="b")

# Set plot labels and title
plt.title(
  'Wikipedia Hits on English Language Pages Containing "Barack_Obama"\n \
  November 3 to November 5, 2008',
  fontsize=16,
)
plt.xlabel("Day-Hour (Eastern Time)", fontsize=12)
plt.ylabel("Page Visits (x 1000)", fontsize=12)

# Set x-ticks start and end
start_date = pd.Timestamp("2008-11-03 00:00:00")
end_date = pd.Timestamp("2008-11-06 00:00:00")
plt.xlim(start_date, end_date)
```

```
# Make every tick at 3 hours
plt.gca().xaxis.set_major_locator(mdates.HourLocator(interval=3))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter("%b %d %H:%M"))

# Highlight Obama victory (Eastern Time)
obama_victory = pd.Timestamp("2008-11-04 23:00:00")
plt.axvline(obama_victory, color="red", linestyle="--", linewidth=2)
plt.text(
  obama_victory - pd.Timedelta(hours=1),
  plt.ylim()[1] - 25,
  "Obama victory announced,\nNov 4, 11 PM ET",
  color="red",
  fontsize=10,
  ha="right",
  va="bottom",
)

# Show plot
plt.grid(True)
plt.gcf().autofmt_xdate() # Auto-format date
plt.show()
```



In the plot above, we see a clear spike in the hours immediately after Obama's victory was announced. The other (smaller) spike between 6 am and 12 pm on November 5 probably corresponds to people who did not catch the victory at night and were waking up to the news (accounting for the six time zones across America.)

Question 3

SQL practice using the Stack Overflow database.

Question 3a

Find all the users (including their `displayname`) who have asked but never answered a question. You're welcome to set up your solution using views, but you should not need to. In your solution, provide as many separate queries as needed to illustrate all of the following SQL functionality: a subquery in the WHERE statement, a subquery in the FROM statement, a set operation using INTERSECT, UNION or EXCEPT), and an outer join (note that SQLite does not have a right outer join). Check the count of the number of results from the different queries to make sure they are the same.

Note that for reasons I haven't looked into, both the questions and answers tables contain rows where the `ownerid` is `NULL`. These can mess up certain versions of the queries I'm asking you to create. So first create views that omit those cases from those two tables. (One could do the omitting within the main query, but this is a good situation for using views to make the main query easier to understand.)

I first downloaded a copy of the SQLite version of the Stack Overflow 2021 database, and created views that omitted rows where `ownerid` is `NULL`, as required.

I then set up queries that satisfy the four requirements, including checking the count of fields. In anticipation of part b, I also timed how long each query took and saved the durations as variables. Since these steps are repeated, I created a function `query_and_time()` that would both query and time.

The count at the end confirms that the four separate query methods return the same count of results.

```
import time
import sqlite3 as sq

db_filename = "stackoverflow-2021.db"

# Connect to db
con = sq.connect(db_filename)

# Create cursor object for interacting with the db
```

```
db = con.cursor()

# Create views that omit where ownerid is NULL
# in 'questions' and 'answers'
db.execute(
  """
  CREATE VIEW IF NOT EXISTS valid_questions AS
  SELECT * FROM questions WHERE ownerid IS NOT NULL
  """
)
db.execute(
  """
  CREATE VIEW IF NOT EXISTS valid_answers AS
  SELECT * FROM answers WHERE ownerid IS NOT NULL
  """
)
con.commit()  # Commit changes


def query_and_time(query):
  """Execute and time SQL query."""
  start_time = time.time()
  result = db.execute(query).fetchall()
  timing = time.time() - start_time
  return result, timing


# Solution 1: a subquery in the WHERE statement
result1, timing1 = query_and_time(
  """
  SELECT userid, displayname FROM users
  WHERE userid IN (
    SELECT q.ownerid FROM valid_questions q
    WHERE q.ownerid NOT IN (
      SELECT a.ownerid FROM valid_answers a
    )
  )
  """
)
print("\nExample results using a subquery in the WHERE statement:")
display(result1[:5])
```

```
# Solution 2: a subquery in the FROM statement
result2, timing2 = query_and_time(
  """
  SELECT userid, displayname FROM users
  WHERE userid IN (
    SELECT q.ownerid
    FROM (
      SELECT ownerid FROM valid_questions
    ) AS q
    WHERE q.ownerid NOT IN (
      SELECT a.ownerid
      FROM (
        SELECT ownerid FROM valid_answers
      ) AS a
    )
  )
  """
)
print("\nExample results using a subquery in the FROM statement:")
display(result2[:5])

# Solution 3: a set operation using INTERSECT, UNION or EXCEPT
result3, timing3 = query_and_time(
  """
  SELECT userid, displayname FROM users
  WHERE userid IN (
    SELECT q.ownerid FROM valid_questions q
    EXCEPT
    SELECT a.ownerid FROM valid_answers a
  )
  """
)
print("\nExample results using INTERSECT, UNION or EXCEPT):")
display(result3[:5])

# Solution 4: an outer join
result4, timing4 = query_and_time(
  """
  SELECT userid, displayname FROM users u
  WHERE userid IN (
    SELECT q.ownerid FROM valid_questions q
    LEFT OUTER JOIN valid_answers a ON q.ownerid = a.ownerid
```

```
    WHERE a.ownerid IS NULL
  )
  """
)
print("\nExample results using an outer join:")
display(result4[:5])

# Check count of the number of results from the different queries
# to make sure they are the same.
results = [result1, result2, result3, result4]
solution_id = [1, 2, 3, 4]
for id, result in zip(solution_id, results):
  print(f"Number of results for Solution {id}: {len(result)}")

# Close connection
con.close()
```

```
Example results using a subquery in the WHERE statement:


[(114.0, 'Craig Smitham'),
 (159.0, 'Holtorf'),
 (719.0, 'Marie Fischer'),
 (1912.0, 'robintw'),
 (1975.0, 'Webjedi')]


Example results using a subquery in the FROM statement:


[(114.0, 'Craig Smitham'),
 (159.0, 'Holtorf'),
 (719.0, 'Marie Fischer'),
 (1912.0, 'robintw'),
 (1975.0, 'Webjedi')]


Example results using INTERSECT, UNION or EXCEPT):
```

```
[(114.0, 'Craig Smitham'),
 (159.0, 'Holtorf'),
 (719.0, 'Marie Fischer'),
 (1912.0, 'robintw'),
 (1975.0, 'Webjedi')]
```

Example results using an outer join:

```
[(114.0, 'Craig Smitham'),
 (159.0, 'Holtorf'),
 (719.0, 'Marie Fischer'),
 (1912.0, 'robintw'),
 (1975.0, 'Webjedi')]

Number of results for Solution 1: 512091
Number of results for Solution 2: 512091
Number of results for Solution 3: 512091
Number of results for Solution 4: 512091
```

Question 3b

> Do the different approaches vary much in terms of how much time they take? What about using SQLite versus DuckDB?

The timings from part a using SQLite are printed below:

```
print(f"Time with subquery in WHERE: {timing1}")
print(f"Time with subquery in FROM:  {timing2}")
print(f"Time using EXCEPT:           {timing3}")
print(f"Time using outer join:       {timing4}")
```

```
Time with subquery in WHERE: 3.581627130508423
Time with subquery in FROM:  3.4857184886932373
Time using EXCEPT:           4.066743612289429
Time using outer join:       6.620959758758545
```

The results show that while the timing of the queries are all in the same order of magnitude, it seems like using an outer join takes the longest time, while using a subquery in a FROM statement is the quickest (followed shortly by using an EXCEPT set operation). The different speeds are probably a function of their logic and execution plans, with some being better than suited than others for this particular task.

We could try running the same code with DuckDB, and time the similarly results:

```
import duckdb as dd

db_filename = "stackoverflow-2021.duckdb"

# Connect to db (now with DuckDB!)
con = dd.connect(db_filename)

# Create cursor object for interacting with the db
db = con.cursor()

# Create views that omit where ownerid is NULL
# in 'questions' and 'answers'
db.execute(
  """
```

```
  CREATE VIEW IF NOT EXISTS valid_questions AS
  SELECT * FROM questions WHERE ownerid IS NOT NULL
  """
)
db.execute(
  """
  CREATE VIEW IF NOT EXISTS valid_answers AS
  SELECT * FROM answers WHERE ownerid IS NOT NULL
  """
)
con.commit()  # Commit changes


def query_and_time(query):
  """Execute and time SQL query."""
  start_time = time.time()
  result = db.execute(query).fetchall()
  timing = time.time() - start_time
  return result, timing


# Solution 1: a subquery in the WHERE statement
result1_dd, timing1_dd = query_and_time(
  """
  SELECT userid, displayname FROM users
  WHERE userid IN (
    SELECT q.ownerid FROM valid_questions q
    WHERE q.ownerid NOT IN (
      SELECT a.ownerid FROM valid_answers a
    )
  )
  """
)

# Solution 2: a subquery in the FROM statement
result2_dd, timing2_dd = query_and_time(
  """
  SELECT userid, displayname FROM users
  WHERE userid IN (
    SELECT q.ownerid
    FROM (
      SELECT ownerid FROM valid_questions
```

```
    ) AS q
    WHERE q.ownerid NOT IN (
      SELECT a.ownerid
      FROM (
        SELECT ownerid FROM valid_answers
      ) AS a
    )
  )
  """
)

# Solution 3: a set operation using INTERSECT, UNION or EXCEPT
result3_dd, timing3_dd = query_and_time(
  """
  SELECT userid, displayname FROM users
  WHERE userid IN (
    SELECT q.ownerid FROM valid_questions q
    EXCEPT
    SELECT a.ownerid FROM valid_answers a
  )
  """
)

# Solution 4: an outer join
result4_dd, timing4_dd = query_and_time(
  """
  SELECT userid, displayname FROM users u
  WHERE userid IN (
    SELECT q.ownerid FROM valid_questions q
    LEFT OUTER JOIN valid_answers a ON q.ownerid = a.ownerid
    WHERE a.ownerid IS NULL
  )
  """
)

print("DUCKDB TIMINGS:")
print(f"Time with subquery in WHERE: {timing1_dd}")
print(f"Time with subquery in FROM:  {timing2_dd}")
print(f"Time using EXCEPT:           {timing3_dd}")
print(f"Time using outer join:       {timing4_dd}")

# Close connection
```

```
con.close()
```

```
DUCKDB TIMINGS:
Time with subquery in WHERE: 0.4081897735595703
Time with subquery in FROM:  0.37999534606933594
Time using EXCEPT:           0.38228559494018555
Time using outer join:       0.4474198818206787
```

We find that the timing is much quicker (an order of magnitude) with DuckDB! Based on this blog post, it seems like DuckDB has been designed to be optimised for *analytical* queries that involve a lot of rows, as may be the case with the Stack Overflow database. Furthermore, based on the course notes, DuckDB stores data *column-wise*, and it can process the data without needing to read it all into memory. In short, DuckDB proves itself to be a compelling candidate for querying tasks that involve large datasets.