2023-07-14 sql-vs-rd-pattern-analysis_executed

July 14, 2023

1 Final analysis — SQL vs. Visual Diagrams for Matching Relational Query Patterns — June 2023 Study

We are preregistering this study based on the OSF Google Docs template, which is one of several preregistration templates that OSF provides. Our experimental setup is inspired by Leventidis et al. (2020).

See our updates post-registration below in the Other section.

1.1 Study Information

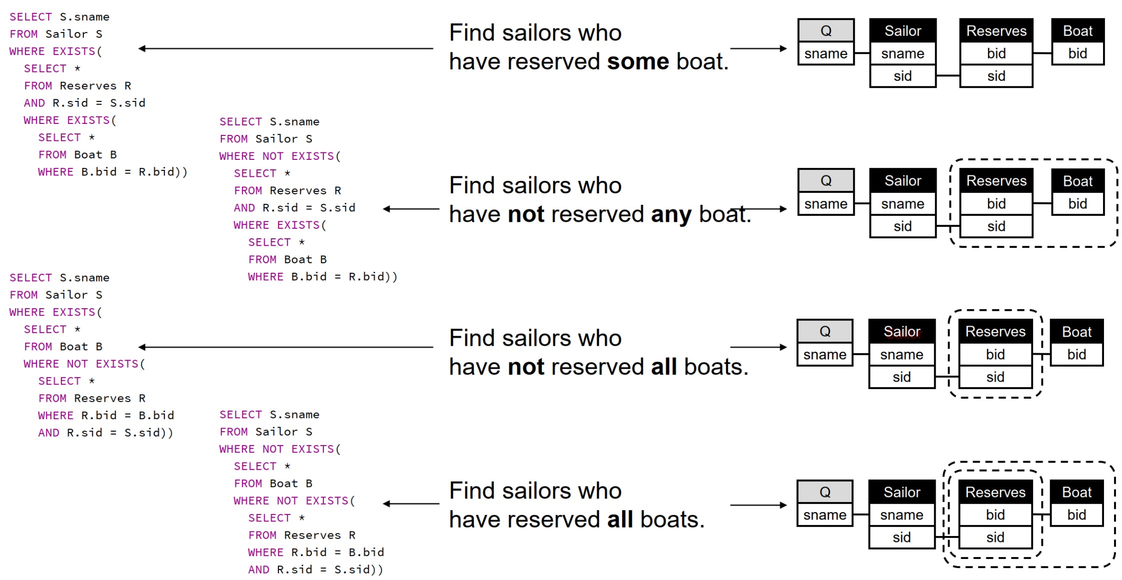
- **Title:** The effect of SQL vs. Visual Diagrams on time and correctness matching relational query patterns
- **Authors:** *Anonymous for peer review. The online form on osf.io will list authors upon publication or embargo expiration.*
- **Description:** Pilot testing has indicated that visual diagrams (RD) improve participant speed at correctly identifying relational query patterns, contrasting with formatted SQL. We will measure participant time and the proportion of correct answers for two conditions (RD and SQL) and 4 relational query patterns across 32 questions.
- **Hypotheses:** We are testing for a total of `3` hypotheses:
  - ***Time:***
    * Let θ_X denote the median time per question in seconds for a given condition X per participant. We hypothesize that (1) θ_{RD} / $\theta_{SQL} < 1$, thus participants are relatively faster using `RD` compared to `SQL`.
    * Corrected to match code & original intent: Let θ_{X1} and θ_{X2} denote the median time per question in seconds for a given condition X per participant in the first $X1$ or second half $X2$. We hypothesize that (2) both $\theta_{RD2}/\theta_{RD1}$ and $\theta_{SQL2}/\theta_{SQL1} < 1$, thus participants are relatively faster in the 2nd half than the 1st.
  - ***Correctness:*** Let δ_X denote the mean proportion of correct responses for a given condition X. We hypothesize that (3) $\delta_{RD} \simeq \delta_{SQL}$, i.e., participants make a comparable number of correct responses using `RD` or `SQL`.

1.2 Design Plan

- **Study type:** Experiment.
- **Blinding:** No blinding is involved in this study.

- **Study design:** We have a within-subjects design, i.e., each participant sees questions in both of our modes: `RD` and `SQL`. Each participant will see a total of 32 questions: 2 modes × 4 patterns × 2 instances in each half × 2 halves. I.e., they will see each pattern-mode combination 2 times per half. For each question, the participant will be given a SQL query presented using one of the modes. They must choose the most appropriate of 4 natural-language interpretations of the query, each corresponding to one of our 4 patterns. Their sequence number determines the mode presented to each participant for a given question — described under the randomization bullet. The stimuli for each mode is:
  - `SQL` — A conventional SQL representation with appropriate indentation. The SQL text is indented and SQL keywords are color-coded appropriately.
  - `RD` — A visual diagram we created of the query.

    See the included `supplement/tutorial.pdf` file for a sample of the stimuli and how to read them, one page extracted here:



- **Randomization:** To reduce ordering effects caused by which mode is presented first, we assign participants as they arrive to alternately start with `SQL` (sequence number `0`) or `RD` (sequence number `1`). We then alternate the modes the presenter sees with each question. I.e., `[SQL, RD, SQL, RD...]`. We randomize the order that patterns are presented in each half separately, ensuring that each combination appears the same number of times and that both halves have the same number of each pattern-mode combination.

1.3 Sampling Plan

- **Existing data:** Registration before creating data: As of the date of submission of this research plan for preregistration, the data have not yet been collected, created, or realized. Only pilot data has been collected.
- **Explanation of existing data:** N/A.
- **Data collection procedures:**

- ***Population:*** Participants will be recruited through Amazon Mechanical Turk (AMT), a popular crowdsourcing website used for a variety of tasks, including academic studies.
- ***Recruitment efforts:*** A Human Intelligence Task (HIT) will be posted on AMT. We may repeatedly re-post the HIT as recruitment slows over time.
- ***Inclusion criteria:*** Participants could accept the AMT HIT if they are all of the following:
  1. Residing in the USA and, thus, we expect English proficiency.
  2. Adults (over 18 years old).
  3. Experienced SQL users, self-determined with the prompt: "Workers should be familiar with SQL at the level of an advanced undergraduate database class, in particular with nested SQL queries."
  4. Have submitted over `500` approved HITs on AMT.
  5. Have more than `97%` approved HIT assignments on AMT.
- ***Exclusion criteria:*** None. Pilot study participants were collected from our institution so did not need to be excluded on AMT.
- ***Timeline:*** Data will be collected from when we start until our **Stopping rule** below is met.
- ***Payment:***
  - *AMT Rejection criteria:* A HIT will be accepted and the participant paid only if they correctly answered $\geq$ `16/32` questions within `50` minutes. Otherwise, the HIT will be rejected.
  - *Base pay:* `$6.00` USD.
  - *Correctness bonus:* For every correctly answered question after the `16th` the participant receive a bonus payment of `$0.20` USD for a total pay of `$9.20` USD.
  - *Time bonus:* Based on total test completion time, the participant will receive a percentage bonus on total pay (including the correctness bonus). Completion within `11` minutes awards a `5%` bonus for a maximum pay of `$9.66`. Each minute faster gets you an additional `5%` bonus up to `40%` for completing within `4` minutes, with a maximum pay of `$12.88`.

- **Sample size:** Our target sample size is `50` participants.
- **Sample size rationale:** As all 13 pilot participants were faster with `RD` than `SQL`, we did not use a power analysis to choose the sample size. Instead, `50` was chosen as a meaningfully large and round number that is still a multiple of 2 to ensure that we have an equal number of participants among the sequences (see **Randomization**, above)
- **Stopping rule:** We will terminate data collection once our number of complete HITs has reached our maximum target sample size. Given the strict ***Inclusion criteria*** in our study, it is possible that we won't be able to hit our target sample size. In that case, we shall restrict our analysis to the data we can collect before paper submission. We will continue collecting data until we reach the maximum target sample size or the camera-ready paper submission deadline.

1.4 Variables

- **Manipulated Variables:**
  - *Mode:* `[SQL, RD]`. See Study Information.

- **Measured Variables:**
  - For each participant and each question they answer, we record:
    - ***Time (quantitative):*** The time they take to answer the question.
    - ***Pattern (categorical/integer):*** The pattern they were provided in one of the modes for that question.
    - ***Choice (integer):*** Their selected pattern from the `4`-option multiple choice question.
    - ***Correct (boolean/integer):*** Whether their answer for the `4`-option multiple choice question was correct.
  - For each participant, we also record:
    - ***Sequence (categorical):*** The sequence the participant was randomly assigned to (see Design Plan).
    - ***Free-text feedback (string):*** The participant's optional answers to a feedback prompt.

- **Indices:** From our collected study data, we will calculate:
  - ***Indices for time***
    - **`Median time per mode per participant:`** This is calculated by taking the median of the `16` ***Time*** records for each mode for each participant. Using that information, we can calculate the median time across all participants.
    - **`Ratio of median time of RD relative to SQL per participant:`** Using the `Median time per mode per participant`, we will calculate `RD/SQL`.
    - **`Quartiles and CIs of median time per mode across all participants:`** Using the `Median time per mode per participant`, we will calculate the 1st, 2nd (median), 3rd quartiles.
    - **`Quartiles and CIs of median time of RD relative to SQL across all participants:`** Using the `Ratio of median time of RD relative to SQL per participant`, we will calculate the 1st, 2nd (median), 3rd quartiles as well as 95% Confidence Intervals using Bias Corrected and Accelerated (BCa) bootstrapping (Efron (1987)).
    - **`Per-half indices:`** We will compute both the `Quartiles and CIs of median time...` indices for each half.
  - ***Indices for correctness***
    - **`Proportion of correct responses per mode per participant:`** For each participant, calculate the proportion of correct responses per mode as: `correct responses / total questions per mode`.
    - **`Mean and CIs of proportion of correct responses per mode across all participants:`** The mean proportion of correct responses across all participants is calculated by taking the arithmetic mean of all the `Proportion of correct responses per mode per participant` values for a given mode across all participants. We will calculate 95% Confidence Intervals using Bias Corrected and Accelerated (BCa) bootstrapping.

1.5 Analysis Plan

- **Statistical Models:**
  - ***Distribution testing:*** We will examine the distributions of our data visually, including for each mode, to ensure there are no problematic distributions.
  - ***Hypothesis Testing:*** We will visually examine the 95% BCa bootstrapped confidence intervals for each mean and median.
- **Transformations:** N/A.
- **Inference criteria:** We will interpret the results using Interval Estimation rather than dichotomous p-value cutoffs (e.g., we will not use p<.05 to determine statistical significance). See Dragicevik (2016) and Besançon & Dragicevic (2019) for a discussion of using estimation for these types of analyses.
- **Data exclusion:** To perform a concrete analysis of our data, we would like to minimize the set of outlier points as they will negatively affect the quality of our statistical analysis and introduce unwanted/non-existent bias. After collecting our data, we will examine the time distribution of each worker. In particular, we will examine how long each worker took to answer the question on a per-question basis. We expect two types of outlier points in our experiments.
  1. **Speeders:** Workers that answered a question much faster than the vast majority of participants (usually in the order of a few seconds, i.e., workers tried to rush answering each question without thinking). The range could vary, but a rough indication of a speeder would be if their time per question is 2-3 standard deviations lower than the mean time per question.
  2. **Unusually delayed answers:** This refers to workers who took unusually long to answer a question. This is most likely attributed to some distraction that made the worker not focus on our question while the timer was running (i.e., a phone call, text message, bathroom break, etc.). As an online test, we can't know exactly what was the cause of it, but usually, we can identify such data points by noticing their much larger value in time. Since we capture a time distribution, it is expected to be left-skewed, and thus a rough measure of an *unusually delayed answer* would be about ≥ 3 times the mean time per question.
  3. **Cheaters:** Previous studies have indicated that workers can leak answers to enable other workers to answer all the questions correctly and quickly. We have used technical measures to limit this possibility and give each user different sets of stimuli. However, if we identify cheaters through log analysis, we will exclude them.
  4. **Median:** To minimize the effect of outliers for all the above reasons and to provide a statistically valid unbiased estimator for the ratio of times, we use the median instead of the mean aggregation for time analysis.
- **Missing data:** We will unlikely have missing data because for a participant to submit their results, they must answer all the questions. However, if we have missing data points from an individual, we will remove the individual completely from our analysis.

1.6 Other

- **Discrepancies between preregistration prose and analysis code:** The intent of our study design is explained in this section. In case of any discrepancy between the analysis code below and this section, what is written in this section takes precedence for the preregistration.
- **Updates post-registration:**
  1. Our second hypothesis was incorrectly stated above, but our intent was clear from the below text and code. We have updated it. This result should be evaluated accordingly as it is a deviation from our prergistration.
  2. Minor error in correctness score calculation fixed.
  3. Anonymization of MTurk worker IDs is removed from this code and now done outside this worksheet to avoid accidental release of worker IDs.
  4. Time spent on tutorials was erroneously collected, and should not be used. We remove this column in the anonymization code.
  5. In total, 177 participants began the study, but many quit before finishing the tutorial or after a few questions. Only 120 participants submitted the HIT. Of those, only 58 reached the 50% correctness threshold for HIT acceptance. We only select the first 50 of those 58 that were submitted to be in accordance with our preregistration.
  6. Added visual emphasis for ratio = 1 in figure 1b.
  7. Added user feedback printing at the end of the worksheet.
  8. Added "variants" for further exploratory analysis based on elevated correctness thresholds.
  9. Added per-pattern exploratory analysis.
  10. Simplified the code for figure 3.
  11. Added additional figure showing difference for question 4.

2 Scripts to analyze the study data

(Q1) TIMING PER PARTICIPANT (SQL vs RD) 1. Per participant, calculate the median time in seconds spent on SQL and RD (32/2=16 per mode and participant, irrespective of correctness) 2. Q1a: show violin plot figure with median times per user compared via gray lines 3. Calculate their ratio per user (also gives fraction of users who are faster with one or the other) 4. Calculate the median of those fractions and the 95% CI 5. Q1b: show violin plot figure with fractions, and also 95% CI

(Q2) TIMING PER PARTICIPANT (SQL vs RD / 1st vs. 2nd half) 1. Per participant, calculate the median time over all questions answered in 1st half in RD (32/2/2=8) and SQL, and in 2nd half. 2. Q2: show repeated measure violin plot figure, showing improvements over time, of 2 halfs 3. Calculate the relative ratio for timing 2nd/1st for RD, and SQL including 95% CI

(Q3) TIMING PATTERNS ACROSS PARTICIPANTS 1. calculate the median time per pattern (4) across the two modes (2). Thus 8 values. 2. show repeated measure violin plot figure

(Q4) CORRECTNESS (SQL vs RD) 1. take mean correct over all questions and all users answered in SQL (32/2*13), or in RD (2 values) 2. calculate 95% CI for each, and sampled p-value (perhaps with difference?)

2.1 Load packages

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
sns.set(style="whitegrid",font_scale=2)
import matplotlib.pyplot as plt
from scipy.stats import bootstrap as scipybootstrap
from IPython.display import display
import matplotlib.ticker as mtick           # allows change to percentage

# Tell matplotlib to export svg text as text not paths
plt.rcParams['svg.fonttype'] = 'none'
plt.rcParams['axes.axisbelow'] = True   # draw axes and grids behind everything␣
↪else

# Set Jupyter and Pandas to show 3 decimal places, does not work for lists of␣
↪numbers
%precision 3
pd.options.display.float_format = '{:,.3f}'.format
np.set_printoptions(precision=3)
# np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})     ␣
↪      # TODO: does not work for lists
def print(*args):
    __builtins__.print(*("%.3f" % a if isinstance(a, float) else a
                         for a in args))
```

2.2 Global Variables Setup

[2]:
```
# A set of constant global variables used throughout the notebook
num_questions = 32
modes = ['SQL', 'RD']
mode_to_name = {0: 'SQL', 1: 'RD'}

# anonymizeddata = 'data/users-table-pilot.csv'                      # pilot
anonymizeddata = 'data/users-table-anonymized.csv'
transformeddata = 'data/transformed_data.csv'    # file with appropriately␣
 ↪transformed data ready for analysis

BOOTSTRAPCONFIDENCE = 0.95       # confidence level used for bootstrap
BOOTSTRAPMETHOD = 'BCa'          # method used for bootstrap, appears to be␣
 ↪better than the textbook version for mean (but not for median), also␣
 ↪available as 'percentage'
BOOTSTRAPSAMPLES = 10000         # number of resamples
VARIANT = 1                      # variant 1: all participants, variant 2: only␣
 ↪for correctness = 1.0, variant 3: only for correctness = 0.9, variant 4:␣
 ↪only for correctness >= 0.66
```

2.3 Define subfolder where figures are stored

By default, figures will not be saved. If you want to save figures, set savefig to `True`. Learned from: https://github.com/jorvlan/open-visualizations/blob/master/Python/tutorial_2/repeated_measures_python_2.ipynb

[3]:
```
savefig = True
if savefig:
    import os
    from os.path import isdir
    cwd = os.getcwd()   # Get current working directory, but you can specify␣
 ↪your own directory of course.
    if  os.path.exists(cwd + "/figs"):
        print("Directory already exists")
        fig_dir = cwd + "/figs"     # Assign the existing directory to a␣
 ↪variable
    elif not os.path.exists(cwd + "/figs"):
        print("Directory does not exist and will be created ......")
        os.makedirs(cwd + "/figs")
        if isdir(cwd + "/figs"):
            print('Directory was created succesfully')
        fig_dir = cwd + "/figs"     # Assign the created directory to a variable
    else:
        print("Something went wrong")
```

```
Directory already exists
```

2.4 Loading full data, transforming it, and saving the transformed version

Loading the full data, transforming it to make available for later analysis, and saving it

[4]:
```
# --- Load anonymized full study data
df = pd.read_csv(anonymizeddata)

# --- Filter on 'current_section=RESULTS'
dfresults = df.loc[(df.current_section == "RESULTS")].copy()            # (7/
↪6/2023: added filter to only focus on RESULTS)

# --- Turn string to array
from ast import literal_eval            # to turn string to array
dfresults['pattern_order']= dfresults['pattern_order'].apply(literal_eval)

# display(dfresults)
# The "current page" is the section of the study the workers are doing to save␣
↪their state & prevent them cheating

# --- The following code block transforms the data frame to have one question␣
↪per row. That simplifies the later analysis.
# reshape dfresults (melt, pivot) to bring multiple question times (e.g.␣
↪'q7_time') per row into separate rows
# https://towardsdatascience.com/
↪wide-to-long-data-how-and-when-to-use-pandas-melt-stack-and-wide-to-long-7c1e0f462a98
df2 = dfresults.melt(id_vars=['worker_id', 'sequence_num', 'pattern_order',
                       'q1', 'q2','q3', 'q4','q5', 'q6','q7', 'q8', 'q9', 'q10',
                       'q11', 'q12','q13', 'q14','q15', 'q16','q17', 'q18',␣
↪'q19', 'q20',
                       'q21', 'q22','q23', 'q24','q25', 'q26','q27', 'q28',␣
↪'q29', 'q30',
                       'q31', 'q32'], value_vars=['q1_time', 'q2_time',␣
↪'q3_time', 'q4_time','q5_time', 'q6_time', 'q7_time', 'q8_time', 'q9_time',␣
↪'q10_time',
                                                  'q11_time', 'q12_time',␣
↪'q13_time', 'q14_time', 'q15_time', 'q16_time', 'q17_time', 'q18_time',␣
↪'q19_time', 'q20_time',
                                                  'q21_time', 'q22_time',␣
↪'q23_time', 'q24_time', 'q25_time', 'q26_time', 'q27_time', 'q28_time',␣
↪'q29_time', 'q30_time',
                                                  'q31_time', 'q32_time'],␣
↪var_name='question', value_name='time')

# replace time in msec with sec in column 'time'
df2['time'] = df2['time'] / 1000

# replace question string 'q7_time' with number '7' in column 'question'
```

```
from re import search as re_search             # regular expression
new_column = []
for values in df2['question']:
    new_column.append(int(re_search(r'\d+', values).group()))
df2['question'] = new_column

# choose the right pattern from the list 'pattern_order' and add as column␣
↪'pattern'
new_column = []
for (pattern_order_list, ind) in zip(df2['pattern_order'], df2['question']):
    new_column.append(pattern_order_list[ind-1])
df2['pattern'] = new_column

# determine the 'mode' (SQL or RD) from 'sequence_num' and 'question'
#   sequence_num = 0 means that the first question is shown in SQL, 1 means we␣
↪start instead with RD. Then alternate between the two.
#   Thus (sequence_num + question_num) % 2 == 1 means SQL
#   Thus (sequence_num + question_num) % 2 == 0 means RD
new_column = []
for (sequence, question) in zip(df2['sequence_num'], df2['question']):
    mode = 'SQL' if (sequence + question) % 2 == 1 else 'RD'
    new_column.append(mode)
df2['mode'] = new_column

# determine the 'choice' (among the 4 patterns) made by the user for this␣
↪question. Requires all the 32 question choices (e.g. 'q7') and index of the␣
↪question at hand ('question')
questionarray = df2[['q1', 'q2','q3', 'q4','q5', 'q6','q7', 'q8', 'q9', 'q10',
                     'q11', 'q12','q13', 'q14','q15', 'q16','q17', 'q18',␣
↪'q19', 'q20',
                     'q21', 'q22','q23', 'q24','q25', 'q26','q27', 'q28',␣
↪'q29', 'q30',
                     'q31', 'q32']].to_numpy()
questionindex = df2[["question"]].to_numpy()

new_array = np.take_along_axis(questionarray,questionindex-1,1)      # take the␣
↪'questionindex'-th entry from each row of the questionarray (notice 1-index␣
↪vs 0-indexin)
df2['choice'] = new_array

# determine whether the choice was correct by comparing the ground truth␣
↪('pattern') against the choice made ('choice'). Saved as 0/1 value in new␣
↪column 'correct'
new_column = []
for (pattern, choice) in zip(df2['pattern'], df2['choice']):
    correct = 1 if pattern == choice else 0
```

```python
    new_column.append(correct)
df2['correct'] = new_column

# sort by worker and question number, and reset the inde
df2.sort_values(by=['worker_id', 'question'], inplace=True)
df2.reset_index(drop=True, inplace=True)
# display(df2)

# select only the relevant subset of columns
df_transformed_data = df2[['worker_id', 'question', 'time', 'pattern', 'mode',␣
↪'choice', 'correct']]
# display(df3)

# pd.write_csv(filename)
df_transformed_data.to_csv(transformeddata,
                           index=False,
                           )
display(dfresults)
```

```
    worker_id  assignment_id  hit_id  qualification_score current_section  \
0          42            NaN     NaN                  NaN         RESULTS
1         147            NaN     NaN                  NaN         RESULTS
3         148            NaN     NaN                  NaN         RESULTS
5          19            NaN     NaN                  NaN         RESULTS
6         161            NaN     NaN                  NaN         RESULTS
..          …              …      …                    …             …
165        57            NaN     NaN                  NaN         RESULTS
166        62            NaN     NaN                  NaN         RESULTS
168         6            NaN     NaN                  NaN         RESULTS
169        28            NaN     NaN                  NaN         RESULTS
170       163            NaN     NaN                  NaN         RESULTS

    current_page  sequence_num  \
0              1         1.000
1              1         1.000
3              1         1.000
5              1         0.000
6              1         1.000
..           …             …
165            1         1.000
166            1         0.000
168            1         0.000
169            1         1.000
170            1         1.000

                                          pattern_order  \
0    [1, 1, 2, 1, 1, 2, 3, 4, 2, 4, 4, 2, 4, 3, 3, …
```

```
1    [3, 1, 3, 3, 2, 2, 4, 4, 2, 1, 1, 2, 1, 3, 4, …
3    [1, 3, 3, 2, 3, 2, 2, 4, 2, 4, 4, 3, 4, 1, 1, …
5    [3, 2, 4, 4, 2, 1, 1, 3, 2, 3, 4, 2, 1, 1, 3, …
6    [2, 3, 4, 2, 2, 1, 1, 4, 3, 3, 1, 1, 3, 4, 4, …
..                                                …
165  [3, 3, 4, 2, 1, 1, 3, 2, 1, 4, 4, 3, 2, 1, 2, …
166  [4, 4, 3, 3, 4, 4, 3, 2, 1, 1, 2, 2, 1, 1, 2, …
168  [3, 3, 3, 4, 2, 2, 4, 3, 1, 4, 4, 1, 2, 1, 1, …
169  [4, 3, 1, 4, 1, 4, 2, 2, 4, 3, 3, 2, 2, 1, 3, …
170  [4, 2, 4, 4, 1, 1, 1, 4, 2, 1, 2, 2, 3, 3, 3, …

                 start_datetime                end_datetime  …   q30_time  \
0    2023-07-06 15:33:53.819738   2023-07-06 15:56:48.16091  … 13,226.000
1    2023-07-06 15:35:09.753802  2023-07-06 15:47:48.277039  … 19,847.000
3    2023-07-07 18:11:52.077358  2023-07-07 18:37:30.385374  … 16,843.000
5    2023-07-06 15:33:42.760353  2023-07-06 15:52:40.496031  …  5,311.000
6    2023-07-07 18:11:09.874051  2023-07-07 18:58:04.501641  …  3,524.000
..                          …                           …  …          …
165  2023-07-07 18:11:49.878986   2023-07-07 18:22:30.45823  …  5,306.000
166  2023-07-06 16:05:50.918429  2023-07-06 16:18:44.773885  …  5,546.000
168  2023-07-07 19:01:38.084371  2023-07-07 19:19:47.324967  …  7,472.000
169  2023-07-06 15:34:02.551439  2023-07-06 16:19:00.534461  … 12,782.000
170  2023-07-07 18:44:54.269601  2023-07-07 19:01:59.264466  …  3,067.000

      q31                   q31_start                     q31_end   q31_time  \
0   2.000  2023-07-06 15:54:44.186955  2023-07-06 15:54:57.575035 13,388.000
1   4.000   2023-07-06 15:47:19.25664  2023-07-06 15:47:24.331627  5,074.000
3   3.000   2023-07-07 18:36:52.45584  2023-07-07 18:36:57.673103  5,217.000
5   3.000  2023-07-06 15:51:03.003979  2023-07-06 15:51:20.250417 17,246.000
6   1.000  2023-07-07 18:40:46.153988  2023-07-07 18:40:49.968978  3,814.000
..    …                           …                           …          …
165 4.000  2023-07-07 18:22:06.119203  2023-07-07 18:22:11.094555  4,975.000
166 2.000  2023-07-06 16:18:14.652581  2023-07-06 16:18:25.846863 11,194.000
168 3.000  2023-07-07 19:19:10.726193  2023-07-07 19:19:32.963664 22,237.000
169 4.000  2023-07-06 16:18:21.318188  2023-07-06 16:18:31.224703  9,906.000
170 2.000  2023-07-07 19:00:31.777433  2023-07-07 19:00:36.612726  4,835.000

      q32                   q32_start                     q32_end   q32_time  \
0   4.000  2023-07-06 15:55:00.530759  2023-07-06 15:55:13.292598 12,761.000
1   1.000  2023-07-06 15:47:26.069748   2023-07-06 15:47:33.66475  7,595.000
3   1.000  2023-07-07 18:37:00.669486  2023-07-07 18:37:12.637864 11,968.000
5   4.000  2023-07-06 15:51:22.991053  2023-07-06 15:51:28.173166  5,182.000
6   3.000  2023-07-07 18:40:53.382182  2023-07-07 18:40:56.839762  3,457.000
..    …                           …                           …          …
165 4.000  2023-07-07 18:22:12.775153  2023-07-07 18:22:18.426053  5,650.000
166 4.000   2023-07-06 16:18:27.91728  2023-07-06 16:18:32.772318  4,855.000
168 1.000  2023-07-07 19:19:34.721309  2023-07-07 19:19:39.613308  4,891.000
169 2.000  2023-07-06 16:18:31.963803  2023-07-06 16:18:39.588003  7,624.000
```

```
170 1.000  2023-07-07 19:00:38.045936  2023-07-07 19:00:40.864742  2,818.000

                                              feedback
0    I found the tutorial to be very helpful in und…
1                                                 good
3                                                 Good
5    Nothing particular. I enjoyed the task very mu…
6                                                  NaN
..                                                 …
165                                                NaN
166                                               none
168                                               good
169                                                NaN
170                                           GOODNESS

[133 rows x 152 columns]
```

2.5 Loading transformed data

[5]:
```python
df_transformed_data = pd.read_csv(transformeddata)
display(df_transformed_data)
```

```
      worker_id  question    time  pattern mode  choice  correct
0             0         1 139.259        2  SQL   1.000        0
1             0         2  10.359        2   RD   2.000        1
2             0         3  41.813        4  SQL   4.000        1
3             0         4   8.190        4   RD   4.000        1
4             0         5  80.589        3  SQL   3.000        1
…          …       …      …       …   …      …      …
4251        169        28   4.797        2   RD   2.000        1
4252        169        29   4.589        4  SQL   4.000        1
4253        169        30   4.384        4   RD   4.000        1
4254        169        31   5.703        4  SQL   4.000        1
4255        169        32   4.874        4   RD   4.000        1

[4256 rows x 7 columns]
```

2.6 Filter users down to first 50 (VARIANT filters), and total time users took (in minutes)

[6]:
```python
# New dataframe with worker id and when they started the HITS (allowing to sort␣
↪by starting time)
dfendtime = dfresults[["worker_id", "start_datetime", "sequence_num"]]
dfendtime.set_index("worker_id", inplace=True)

# New dataframe with worker ids and fraction correct (allowing to filter out␣
↪those who did not pass the 0.5 correctness criterion)
```

```
dftemp = df_transformed_data.groupby(['worker_id']).agg(
    time=('time', np.sum),
    correct=('correct', np.mean))
dftemp['time'] = dftemp['time'] / 60
dftemp.sort_values(by=['correct'], ascending=False, inplace=True)
# display(dftemp)

# Joining the dataframes
dftemp = dftemp.join(dfendtime)

# Filtering the dataframes for those who passed the 0.5 correctness criterion
dftemp = dftemp.loc[dftemp.correct >= 0.5]

# Keep only first 50 participants (creates imbalance: 26/24 between sequence␣
↪numbers)
# dftemp = dftemp.sort_values(by="start_datetime", ascending=True)
# dftemp = dftemp.head(50)                              # only keep the first 50␣
↪participants

# Keep only first 50 balanced participants, thus first 25 from sequence 0, and␣
↪first 25 from sequence 1
dftemp0 = dftemp.loc[(dftemp.sequence_num == 0.0)].copy()
dftemp0 = dftemp0.sort_values(by="start_datetime", ascending=True)
dftemp0 = dftemp0.head(25)                              # only keep the first 25␣
↪participants
dftemp1 = dftemp.loc[(dftemp.sequence_num == 1.0)].copy()
dftemp1 = dftemp1.sort_values(by="start_datetime", ascending=True)
dftemp1 = dftemp1.head(25)                              # only keep the first 25␣
↪participants
dftemp = pd.concat([dftemp0, dftemp1])

if VARIANT == 2:
    dftemp = dftemp.loc[dftemp.correct == 1.0]                                  ␣
↪# 12/50
if VARIANT == 3:
    dftemp = dftemp.loc[dftemp.correct >= 0.9]                                  ␣
↪# up to 3 mistakes, 27/50
if VARIANT == 4:
    dftemp = dftemp.loc[dftemp.correct >= 0.66]                                 ␣
↪# up to 12 mistakes, thus 2/3 correct, 34/50
print('dftemp:')
display(dftemp)

print('Number of participants who started with RD first:', np.sum(dftemp.
↪sequence_num))
```

```
df_filtered_data = df_transformed_data[df_transformed_data.worker_id.
    ↪isin(dftemp.index)]      # only retain those that pass the 0.5 correctness␣
    ↪criterium
print('df_filtered_data:')
display(df_filtered_data)
```

```
dftemp:

             time  correct              start_datetime  sequence_num
worker_id
166        13.110    0.969   2023-07-06 15:33:42.37047          0.000
5           9.802    1.000  2023-07-06 15:33:43.176727          0.000
110        21.080    0.656  2023-07-06 15:33:57.191982          0.000
169         3.517    1.000  2023-07-06 15:34:39.261401          0.000
2           8.065    1.000  2023-07-06 15:34:43.567259          0.000
146         8.134    0.625  2023-07-06 15:37:12.663507          0.000
96         18.454    0.719  2023-07-06 15:42:20.453162          0.000
17          3.744    0.500  2023-07-06 16:05:20.144093          0.000
87         21.996    0.812  2023-07-06 16:34:33.028051          0.000
154         8.953    1.000  2023-07-06 16:36:07.050799          0.000
159         8.125    0.969  2023-07-06 17:12:49.060194          0.000
21          7.874    1.000  2023-07-06 17:18:50.168761          0.000
92         12.873    0.531  2023-07-06 18:15:32.990734          0.000
158         8.139    0.531  2023-07-07 18:10:46.043268          0.000
43         22.682    0.875  2023-07-07 18:11:03.035618          0.000
162        14.924    0.969  2023-07-07 18:11:10.045972          0.000
125         7.049    0.906  2023-07-07 18:11:10.432369          0.000
153        27.547    0.656  2023-07-07 18:11:30.090096          0.000
136         7.021    1.000  2023-07-07 18:14:34.271669          0.000
130        16.834    1.000  2023-07-07 18:28:15.617866          0.000
119         3.985    0.594  2023-07-07 18:43:12.941313          0.000
6           8.273    1.000  2023-07-07 19:01:38.084371          0.000
121         6.495    0.969  2023-07-07 19:01:39.418515          0.000
83         19.664    0.844  2023-07-07 19:03:08.590044          0.000
0          16.758    0.625  2023-07-07 19:16:27.964457          0.000
58         13.759    0.656  2023-07-06 15:33:42.871209          1.000
66         11.329    1.000  2023-07-06 15:33:43.726871          1.000
50         10.841    0.906  2023-07-06 15:33:53.523919          1.000
42         14.206    0.938  2023-07-06 15:33:53.819738          1.000
28         14.293    0.938  2023-07-06 15:34:02.551439          1.000
80         18.198    0.906  2023-07-06 15:35:42.063363          1.000
117         2.562    0.562  2023-07-06 16:06:22.118244          1.000
52         15.721    0.812  2023-07-06 16:24:41.817076          1.000
72         14.869    0.969  2023-07-06 16:24:45.985302          1.000
75         10.113    0.719  2023-07-06 16:46:59.265172          1.000
115        10.982    1.000  2023-07-07 18:10:46.107235          1.000
39          8.165    0.906  2023-07-07 18:10:52.555864          1.000
77         14.199    0.594  2023-07-07 18:11:07.532996          1.000
```

```
168        17.812    0.562  2023-07-07 18:11:11.306257         1.000
32         14.009    0.500  2023-07-07 18:11:12.976612         1.000
143         3.266    0.969  2023-07-07 18:11:22.386833         1.000
57          7.527    0.875  2023-07-07 18:11:49.878986         1.000
148        13.752    0.531  2023-07-07 18:11:52.077358         1.000
165         6.050    0.969   2023-07-07 18:12:26.00741         1.000
89         19.763    0.531  2023-07-07 18:12:34.892116         1.000
81         12.004    1.000   2023-07-07 18:13:35.80157         1.000
60         13.981    0.938  2023-07-07 18:14:47.210084         1.000
141        11.045    0.562  2023-07-07 18:16:33.378571         1.000
91          3.584    1.000  2023-07-07 18:34:13.912492         1.000
10         14.587    0.938  2023-07-07 19:01:52.970403         1.000
```

```
Number of participants who started with RD first: 25.000
df_filtered_data:
```

```
      worker_id  question    time  pattern mode  choice  correct
0             0         1 139.259        2  SQL   1.000        0
1             0         2  10.359        2   RD   2.000        1
2             0         3  41.813        4  SQL   4.000        1
3             0         4   8.190        4   RD   4.000        1
4             0         5  80.589        3  SQL   3.000        1
…         …       …       …        …  …      …       …
4251        169        28   4.797        2   RD   2.000        1
4252        169        29   4.589        4  SQL   4.000        1
4253        169        30   4.384        4   RD   4.000        1
4254        169        31   5.703        4  SQL   4.000        1
4255        169        32   4.874        4   RD   4.000        1

[1600 rows x 7 columns]
```

3 Question 1. Figure 1a

[7]:

```python
# create two columns mode and median, with 2 rows per worker (used for Fig 1a␣
 ↪violines)
dfq1a = df_filtered_data.groupby(['worker_id', 'mode']).time.agg(['median'])
dfq1a.reset_index(inplace=True)
# print('dfq1a:')
# display(dfq1a)

# pivot to have one row per worker (used for Fig 1a individual points)
dfq1b = pd.pivot_table(dfq1a, values=['median'], index=['worker_id'],␣
 ↪columns=['mode'])
dfq1b=dfq1b.droplevel(0, axis=1)
print('dfq1b:')
display(dfq1b)
```

```
modes = ['RD', 'SQL']
median_time = {}
ci = {}
ci_delta = {}
for mode in modes:
    median_time[mode] = np.median(dfq1b[mode])
    ci[mode] = scipybootstrap((dfq1b[mode],), statistic=np.median,␣
↪n_resamples=BOOTSTRAPSAMPLES, confidence_level=BOOTSTRAPCONFIDENCE,␣
↪method='percentile', axis=0).confidence_interval        #convert array to␣
↪sequence
    ci_delta[mode] = [median_time[mode] - ci[mode].low, ci[mode].high -␣
↪median_time[mode]]

    print(f'Median time {mode}: {median_time[mode]:.2f}, 95% CI [{ci[mode].low:.
↪2f}, {ci[mode].high:.2f}]')
```

dfq1b:

```
mode           RD    SQL
worker_id
0           7.018 28.357
2           9.144 12.800
5           9.378 16.846
6           9.357 16.023
10          9.960 13.986
17          5.756  5.691
21          8.377 12.603
28         10.586 12.822
32         17.987 25.491
39          7.220  9.152
42         13.366 23.652
43          7.929 13.225
50          5.414  8.546
52         13.305 17.604
57          5.921  8.445
58         10.258  8.849
60         11.616  9.700
66          8.540 12.380
72         18.521 28.279
75         12.252 12.471
77         18.601 31.567
80         15.697 30.716
81         14.380 18.619
83         10.471 38.456
87         19.988 42.377
89         20.476 54.861
91          4.865  6.268
92         17.733 18.866
```

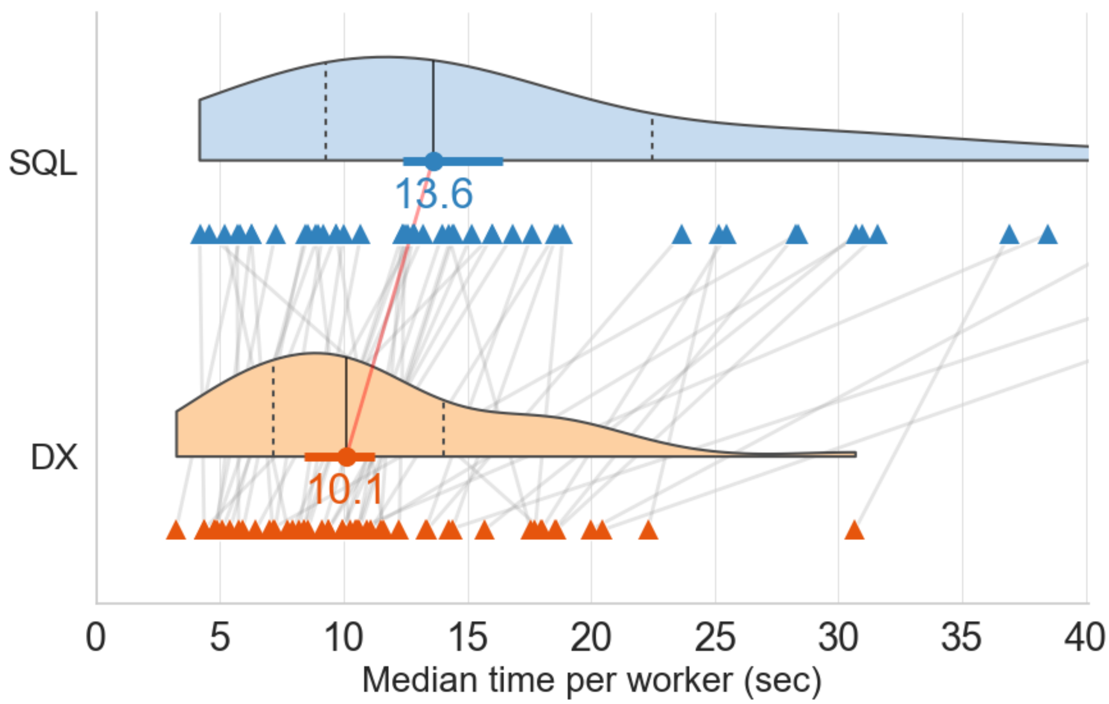
```
96        10.912 30.937
110       22.293 25.159
115       11.087 18.552
117        4.360  4.181
119        7.749  5.184
121        8.207 10.006
125        4.792  8.986
130        7.131  6.258
136       10.635 14.470
141       18.003  4.572
143        3.221  5.837
146        4.337 16.021
148       17.566 14.229
153       14.242 50.465
154       10.534 14.382
158        6.451 12.857
159        9.951 15.182
162        5.090  9.694
165        8.392 10.698
166       11.477 12.369
168       30.665 36.904
169        4.778  7.235

Median time RD: 10.11, 95% CI [8.38, 11.26]
Median time SQL: 13.61, 95% CI [12.37, 16.43]
```

[8]:
```
# Define pre-settings
figwidth = 10
figheight = 6
xlab_size = 20
ylab_size = 20
figfont_size = 24

# Define consistent color maps
my_cmap_sns_light = [(0.9921568627450981, 0.8156862745098039, 0.
 ↪6352941176470588), (0.7764705882352941, 0.8588235294117647, 0.
 ↪9372549019607843)]        # light blue, light orange
my_cmap_sns_dark = [(0.9019607843137255, 0.3333333333333333, 0.
 ↪050980392156862744), (0.19215686274509805, 0.5098039215686274, 0.
 ↪7411764705882353)]       # dark blue, dark orange
my_cmap_dark = sns.color_palette(my_cmap_sns_dark, as_cmap=True)
my_cmap_light = sns.color_palette(my_cmap_sns_light, as_cmap=True)


# Create empty figure and plot the individual datapoints
fig, ax = plt.subplots(figsize=(figwidth,figheight))
```

```
# 1. Violinplots
axsns = sns.violinplot(x='median', y='mode', data=dfq1a,
                       hue=True, hue_order=[False, True], split=True,   # half␣
 ↪violinplots https://stackoverflow.com/questions/53872439/
 ↪half-not-split-violin-plots-in-seaborn
                       inner='quartile',
                       cut=0,               # 0 means ending sharp at end points
                       width=.7,
                       orient = 'h',
                       zorder=20,)

# change the medium default line to full (https://stackoverflow.com/questions/
 ↪60638344/quartiles-line-properties-in-seaborn-violinplot)
for l in axsns.lines[1::3]:
    l.set_linestyle('-')
    l.set_linewidth(1.2)
    l.set_color('black')
    l.set_alpha(0.8)

# Apply colorscheme to violinplots https://stackoverflow.com/questions/70442958/
 ↪seaborn-how-to-apply-custom-color-to-each-seaborn-violinplot
from matplotlib.collections import PolyCollection
for ind, violin in enumerate(axsns.findobj(PolyCollection)):
    violin.set_facecolor(my_cmap_light[ind])


# 2. Plot individual points
y_tilt = -0.25                                     # Set some delta for the␣
 ↪points below the violinplot
y_base = np.zeros(dfq1b.values.shape[0]) + y_tilt   # base vector to which to␣
 ↪broadcast y-tilt values

for i, col in enumerate(modes):
    ax.plot(dfq1b[col],
            y_base + i,
            # 'o',          # circles
            '^',            # triangles_up
            alpha=1,
            zorder=20,      # higher means more visible
            markersize=11,
            markeredgewidth=0,
            # markerfacecolor='none',
            markerfacecolor=my_cmap_dark[i],
            markeredgecolor=my_cmap_dark[i],)
    ax.plot(dfq1b[col],
            y_base + i,
            # 'o',          # circles
```

```
            '^',            # triangles_up
            markersize=11,
            markerfacecolor='white',
            markeredgewidth=1,
            color ='white',
            linewidth = None,
            zorder=1,)


# 3. Plot gray lines connecting modes
for i, idx in enumerate(dfq1b.index):
    ax.plot(dfq1b.loc[idx, modes],
            [y_tilt, y_tilt+1],
            color ='gray', linewidth = 2, linestyle ='-', alpha = .2,
            zorder=0)


# 4. Plot red line connecting medians
ax.plot(np.median(dfq1b, axis=0), [0, 1], color ='red', linewidth = 2,␣
 ↪linestyle ='-', alpha = .4)


# 5. CI Errorbars
for i, mode in enumerate(modes):
    plt.errorbar(median_time[mode], i, xerr=np.array([[ci_delta[mode][0],␣
 ↪ci_delta[mode][1]]]).T,
                 fmt='o', markersize=10,
                 # lw = 3,           # if end line for CI
                 lw = 5,             # if no ned line for CI
                 alpha=1,
                 zorder=100,         # higher means more visible
                 capsize = 0,       # 10
                 # capthick = 4,     # end line for CI
                 capthick = 0,       # no end line for CI
                 # color = 'black',
                 color = my_cmap_dark[i],
                 )   # my_cmap[1])
    ax.text(median_time[mode],
            # i+0.36,
            i-0.16,
            f'{median_time[mode]:.1f}', horizontalalignment='center',
            # color='black',
            color= my_cmap_dark[i],
            fontsize=figfont_size)
    # ax.text(ci[mode].low, i+0.1, f'{ci[mode].low:.1f}',␣
 ↪horizontalalignment='center', color='black', fontsize=20)
```

```python
    # ax.text(ci[mode].high, i+0.1, f'{ci[mode].high:.1f}',␣
 ↪horizontalalignment='center', color='black', fontsize=20)


#Additional settings
ax.set_xticks(range(0, 100, 5))
ax.set_yticks(range(len(dfq1b.columns)))
ax.set_yticklabels(modes, size= ylab_size)
ax.set_xlim(0, 40.1)
ax.set_ylim(-0.5, 1.5)
ax.set_xlabel('Median time per worker (sec)', size = xlab_size)
ax.set_ylabel(None)
ax.set_yticklabels(['DX', 'SQL', ])
# ax.set_title('Median times per worker', size = title_size)
sns.despine()
ax.legend_.remove()

plt.grid(axis = 'x', linewidth = 0.5, color = 'lightgray')

if savefig:
    plt.savefig(fig_dir + f'/q1_figure1_variant{VARIANT}.pdf',␣
 ↪bbox_inches='tight')
    plt.savefig(fig_dir + f'/q1_figure1_variant{VARIANT}.svg',␣
 ↪bbox_inches='tight')
```



4 Question 1. Figure 1b

[9]:
```
dfq1c = df_filtered_data.groupby(['worker_id', 'mode']).time.agg(['median'])␣
↪                             # for each worker, calculate median for both modes
dfq1c = pd.pivot_table(dfq1c, values=['median'], index=['worker_id'],␣
↪columns=['mode'])        # pivot to have one row per worker
dfq1c['ratio median'] = dfq1c['median','RD'] / dfq1c['median','SQL']           ␣
↪                # add the ratio between medians of the two modes

print('dfq1c:')
display(dfq1c)

sample = np.array(dfq1c['ratio median'])                                        ␣
↪              # extract the sample and then create the boostrapped medians
data_ratio = dfq1c['ratio median']

median_ratio = np.median(data_ratio)
ci_ratio = scipybootstrap((data_ratio,), statistic=np.median,␣
↪n_resamples=BOOTSTRAPSAMPLES, confidence_level=BOOTSTRAPCONFIDENCE,␣
↪method=BOOTSTRAPMETHOD, axis=0).confidence_interval        #convert array to␣
↪sequence
ci_ratio_delta = [median_ratio - ci_ratio.low, ci_ratio.high - median_ratio]

print(f'Median ratio: {median_ratio:.3f}, 95% CI [{ci_ratio.low:.3f}, {ci_ratio.
↪high:.3f}]')
print(f'Number (fraction) of users faster with RD: {np.sum(data_ratio<1.0)}␣
↪({np.sum(data_ratio<1.0)/np.sum(data_ratio>0.0):.3f})')
```

```
dfq1c:

          median        ratio median
mode          RD    SQL
worker_id
0          7.018 28.357        0.248
2          9.144 12.800        0.714
5          9.378 16.846        0.557
6          9.357 16.023        0.584
10         9.960 13.986        0.712
17         5.756  5.691        1.011
21         8.377 12.603        0.665
28        10.586 12.822        0.826
32        17.987 25.491        0.706
39         7.220  9.152        0.789
42        13.366 23.652        0.565
43         7.929 13.225        0.600
50         5.414  8.546        0.633
52        13.305 17.604        0.756
```

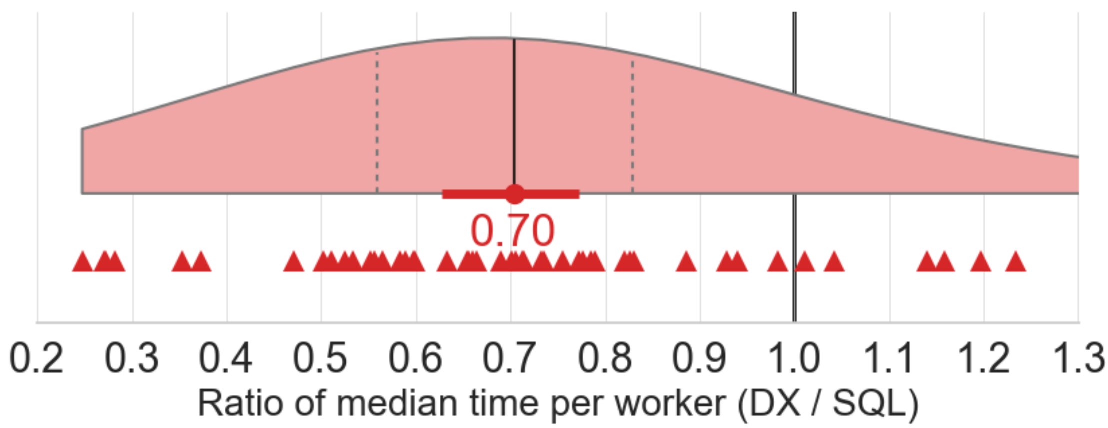
```
57          5.921  8.445        0.701
58         10.258  8.849        1.159
60         11.616  9.700        1.198
66          8.540 12.380        0.690
72         18.521 28.279        0.655
75         12.252 12.471        0.982
77         18.601 31.567        0.589
80         15.697 30.716        0.511
81         14.380 18.619        0.772
83         10.471 38.456        0.272
87         19.988 42.377        0.472
89         20.476 54.861        0.373
91          4.865  6.268        0.776
92         17.733 18.866        0.940
96         10.912 30.937        0.353
110        22.293 25.159        0.886
115        11.087 18.552        0.598
117         4.360  4.181        1.043
119         7.749  5.184        1.495
121         8.207 10.006        0.820
125         4.792  8.986        0.533
130         7.131  6.258        1.140
136        10.635 14.470        0.735
141        18.003  4.572        3.937
143         3.221  5.837        0.552
146         4.337 16.021        0.271
148        17.566 14.229        1.235
153        14.242 50.465        0.282
154        10.534 14.382        0.732
158         6.451 12.857        0.502
159         9.951 15.182        0.655
162         5.090  9.694        0.525
165         8.392 10.698        0.784
166        11.477 12.369        0.928
168        30.665 36.904        0.831
169         4.778  7.235        0.661

Median ratio: 0.703, 95% CI [0.627, 0.772]
Number (fraction) of users faster with RD: 42 (0.840)
```

[10]:
```
# Define figure settings
figwidth = 10
figheight = 3
xlab_size = 20
ylab_size = 20
figfont_size = 24

# Define consistent color maps
```

```
my_cmap_sns_dark = [(0.8392156862745098, 0.15294117647058825, 0.
↪1568627450980392)]
my_cmap_sns_light = [(0.984313725490196, 0.6039215686274509, 0.6)]
my_cmap_dark = sns.color_palette(my_cmap_sns_dark, as_cmap=True)
my_cmap_light = sns.color_palette(my_cmap_sns_light, as_cmap=True)

# Create data frame for split violinplot
dfvp = pd.DataFrame()
dfvp["values"] = sample
dfvp["all"] = ""                                          # attribute that is␣
↪shared by all entries
# print(dfvp)

# Create empty figure and plot the individual datapoints
fig, ax = plt.subplots(figsize=(figwidth,figheight))


# 1. Violinplot
axsns = sns.violinplot(x='values', y='all',     # y='all' just need to group␣
↪both types together
                       data=dfvp,
                       hue = True, hue_order = [False, True],
                       split = True, inner = 'quartile',
                       cut=0,                   # 0 means ending sharp at end␣
↪points
                       width=.6, scale = 'width',
                       # dodge = False,         # When using ``hue`` nesting,␣
↪setting this to ``True`` will separate the strips for different hue levels␣
↪along the categorical axis.
                       orient = 'h',
                       color=my_cmap_light[0],)

# change the medium default linke to full
for l in axsns.lines[1::3]:
    l.set_linestyle('-')
    l.set_linewidth(1.5)
    l.set_color('black')
    l.set_alpha(0.8)


# 2. Plot individual points
y_tilt = -0.13                                  # Set some delta for the␣
↪points below the violinplot
y_base = np.zeros(len(data_ratio)) + y_tilt     # base vector to which to␣
↪broadcast y-tilt values
```

```
ax.plot(data_ratio, y_base,
        # 'o',
        '^',
        alpha=1,
        zorder=20,      # higher means more visible
        markersize=11,
        markeredgewidth=0,
        # markerfacecolor='none',
        markerfacecolor=my_cmap_dark[0],
        markeredgecolor=my_cmap_dark[0],
        )


# 3. CI Errorbars & show numbers
axeb = plt.errorbar(median_ratio, 0, xerr=np.array([[ci_ratio_delta[0],␣
 ↪ci_ratio_delta[1]]]).T,
                    fmt='o',
                    markersize=10, alpha=1,
                    # lw = 3,
                    lw = 5,
                    zorder=100,      # higher means more visible
                    capsize = 0,     # 10
                    # capthick = 4,
                    capthick = 0,
                    # color = 'black',
                    color = my_cmap_dark[0],
                    )

med = np.median(sample)
# ax.text(med, 0.32, f'{100*med:.1f}%', horizontalalignment='center',␣
 ↪color='black', fontsize=20)
# ax.text(med, 0.32, f'{med:.2f}', horizontalalignment='center', color='black',␣
 ↪fontsize=20)
ax.text(med, -0.1, f'{med:.2f}', horizontalalignment='center',
        # color='black',
        color = my_cmap_dark[0],
        fontsize=figfont_size)
# ax.text(ci_ratio.low, 0.04, f'{100*ci_ratio.low:.1f}%',␣
 ↪horizontalalignment='center', color='black', fontsize=20)
# ax.text(ci_ratio.high, 0.04, f'{100*ci_ratio.high:.1f}%',␣
 ↪horizontalalignment='center', color='black', fontsize=20)


# 4. vertical bar for x-axis = 1
plt.plot([1, 1], [-10, 10], color = 'black', zorder = 0, linewidth = 2)
```

```
# Additional settings
# ax.set_ylim(-0.2, 0.4)
ax.set_xticks(np.linspace(0, 2, num=21))
ax.set_ylim(-0.25, 0.35)
ax.set_ylabel(None)          # remove the 'all'
ax.set_xlim(0.2, 1.25)
if VARIANT == 1:
    ax.set_xlim(0.2, 1.301)
if VARIANT == 3:
    ax.set_xlim(0.499, 1.205)
# ax.set_xlabel('Ratio of median time per worker (RD / SQL)', size = xlab_size)
ax.set_xlabel('Ratio of median time per worker (DX / SQL)', size = xlab_size)
sns.despine(left=True)                # remove bounding box
plt.grid(axis = 'x', linewidth = 0.5, color = 'lightgray')
ax.legend_.remove()

if savefig:
    plt.savefig(fig_dir + f'/q1_figure2_variant{VARIANT}.pdf',␣
↪bbox_inches='tight')
    plt.savefig(fig_dir + f'/q1_figure2_variant{VARIANT}.svg',␣
↪bbox_inches='tight')
```



5 Question 2

[11]:
```
# Create df6, df7
df0 = df_filtered_data[['worker_id', 'question', 'time', 'mode']].copy()
df0['H1'] = np.where(df0['question'].between(1, 16, inclusive='both'), 'H1', '')
df0['H2'] = np.where(df0['question'].between(17, 32, inclusive='both'), 'H2',␣
↪'')
# display(df0)
```

```
df1 = df0[['worker_id','question','time','mode','H1']].rename(columns={'H1':␣
↪'section'}) # Two sections: 1st half (H1) and 2nd half (H2)
df2 = df0[['worker_id','question','time','mode','H2']].rename(columns={'H2':␣
↪'section'})
df4 = pd.concat([df1, df2])
df4 = df4.loc[df4['section'] != '']
df4 = df4.reset_index(inplace=False, drop=True)
# display(df4)

df5 = df4.groupby(['worker_id', 'mode', 'section']).time.agg(['median'])        ␣
↪         # for each worker, calculate median for both modes and section
df5.reset_index(inplace=True)
# display(df5)

# pivot to have one row per worker
df6 = pd.pivot_table(df5, values=['median'], index=['worker_id'],␣
↪columns=['mode', 'section'])
df6=df6.droplevel(0, axis=1)

# relative improvements per user per mode
df6['RD', 'ratio'] = df6['RD', 'H2'] / df6['RD', 'H1']
df6['SQL', 'ratio'] = df6['SQL', 'H2'] / df6['SQL', 'H1']
# relative improvements of RD over SQL per user half
df6['H1', 'ratio'] = df6['RD', 'H1'] / df6['SQL', 'H1']
df6['H2', 'ratio'] = df6['RD', 'H2'] / df6['SQL', 'H2']
print('df6:')
display(df6)

# Median of median task time for each mode and section
modes = ['RD', 'SQL']
sections = ['H1', 'H2', 'ratio']
median_time = {}
ci = {}
ci_delta = {}
for mode in modes:
    for section in sections:
        column = (mode, section)
        median_time[column] = np.median(df6[column])
        ci[column] = scipybootstrap((df6[column],), statistic=np.median,
                                    n_resamples=BOOTSTRAPSAMPLES,
                                    confidence_level=BOOTSTRAPCONFIDENCE,
                                    method='percentile',
                                    axis=0).confidence_interval        #convert␣
↪array to sequence
        ci_delta[column] = [median_time[column] - ci[column].low, ci[column].
↪high - median_time[column]]
```

```
        print(f'{mode}, {section}: {median_time[column]:.3f}, 95% CI␣
↪[{ci[column].low:.3f}, {ci[column].high:.3f}]')

for half in ['H1', 'H2']:
    column = half
    median_time[column] = np.median(df6[column])
    ci[column] = scipybootstrap((df6[column],), statistic=np.median,
                                n_resamples=BOOTSTRAPSAMPLES,
                                confidence_level=BOOTSTRAPCONFIDENCE,
                                method='percentile',
                                axis=0).confidence_interval        #convert␣
↪array to sequence
    ci_delta[column] = [median_time[column] - ci[column].low, ci[column].high -␣
↪median_time[column]]
    print(f'{column}: {median_time[column]:.3f}, 95% CI [{ci[column].low[0]:.
↪3f}, {ci[column].high[0]:.3f}]')

# uses df5 to make df7 (used for later plot)
modes = ['SQL', 'RD']
sections = ['H1', 'H2']
df7 = df5.loc[df5['section'].isin(sections)]
# display(df7)
```

df6:

mode	RD		SQL		RD	SQL	H1	H2
section	H1	H2	H1	H2	ratio	ratio	ratio	ratio
worker_id								
0	7.991	5.639	54.237	6.925	0.706	0.128	0.147	0.814
2	9.747	8.585	13.806	11.911	0.881	0.863	0.706	0.721
5	12.148	7.399	22.152	16.363	0.609	0.739	0.548	0.452
6	11.375	7.215	17.726	14.008	0.634	0.790	0.642	0.515
10	14.007	6.982	26.168	9.383	0.498	0.359	0.535	0.744
17	6.242	4.905	7.510	4.858	0.786	0.647	0.831	1.009
21	8.774	7.048	13.952	12.367	0.803	0.886	0.629	0.570
28	12.962	9.658	14.064	9.983	0.745	0.710	0.922	0.967
32	15.100	19.551	23.410	30.148	1.295	1.288	0.645	0.648
39	8.720	6.993	19.712	7.722	0.802	0.392	0.442	0.906
42	15.617	11.213	55.898	15.980	0.718	0.286	0.279	0.702
43	14.982	6.216	65.841	8.636	0.415	0.131	0.228	0.720
50	10.453	3.932	35.611	7.218	0.376	0.203	0.294	0.545
52	19.684	10.204	20.617	15.413	0.518	0.748	0.955	0.662
57	11.396	5.369	25.916	5.913	0.471	0.228	0.440	0.908
58	14.386	7.908	10.149	8.407	0.550	0.828	1.417	0.941
60	15.075	10.655	10.811	9.302	0.707	0.860	1.394	1.146
66	9.150	8.181	14.344	11.183	0.894	0.780	0.638	0.732
72	26.365	13.165	40.867	17.230	0.499	0.422	0.645	0.764
75	22.348	10.909	22.399	11.322	0.488	0.505	0.998	0.964

```
77         12.448 25.276 28.640 36.115 2.031 1.261 0.435 0.700
80         12.041 35.852 43.927 24.125 2.977 0.549 0.274 1.486
81         14.380 13.649 20.355 15.739 0.949 0.773 0.706 0.867
83         21.773  9.941 71.220 13.771 0.457 0.193 0.306 0.722
87         22.406 15.235 82.963 20.297 0.680 0.245 0.270 0.751
89         20.476 20.556 67.059 13.220 1.004 0.197 0.305 1.555
91          4.927  4.822  7.377  5.995 0.979 0.813 0.668 0.804
92         18.061 17.733 23.764 18.866 0.982 0.794 0.760 0.940
96         23.182  5.483 63.761 19.377 0.237 0.304 0.364 0.283
110        29.262 20.494 36.913 23.960 0.700 0.649 0.793 0.855
115        13.155  9.254 29.754 12.825 0.703 0.431 0.442 0.722
117         4.472  4.218  4.181  4.517 0.943 1.080 1.069 0.934
119         7.749  7.720  4.866  5.247 0.996 1.079 1.593 1.471
121         9.007  7.050 11.855  9.004 0.783 0.759 0.760 0.783
125        10.337  3.229 18.303  6.014 0.312 0.329 0.565 0.537
130         7.131  6.832  6.951  6.143 0.958 0.884 1.026 1.112
136        10.238 10.995 11.290 15.410 1.074 1.365 0.907 0.713
141        21.690  6.064  7.111  3.978 0.280 0.559 3.050 1.524
143         3.986  3.192  8.069  4.089 0.801 0.507 0.494 0.781
146         5.294  3.913 17.922  8.953 0.739 0.500 0.295 0.437
148        25.429 13.747 12.210 14.229 0.541 1.165 2.083 0.966
153        19.672 12.869 62.773 23.929 0.654 0.381 0.313 0.538
154        15.026  7.610 18.871  9.690 0.506 0.513 0.796 0.785
158         7.914  4.925 24.032 10.136 0.622 0.422 0.329 0.486
159        10.122  7.675 15.822 13.043 0.758 0.824 0.640 0.588
162         7.441  3.383 11.116  8.642 0.455 0.777 0.669 0.391
165         8.486  8.392 11.799  9.719 0.989 0.824 0.719 0.864
166        17.133 10.287 12.759 10.253 0.600 0.804 1.343 1.003
168        24.126 30.665 39.257 35.037 1.271 0.893 0.615 0.875
169         5.588  4.401  7.989  5.506 0.788 0.689 0.700 0.799

RD, H1: 12.298, 95% CI [10.180, 14.982]
RD, H2: 7.814, 95% CI [7.022, 9.941]
RD, ratio: 0.712, 95% CI [0.634, 0.793]
SQL, H1: 19.291, 95% CI [14.064, 23.764]
SQL, H2: 10.718, 95% CI [9.299, 13.495]
SQL, ratio: 0.700, 95% CI [0.507, 0.785]
H1: 0.643, 95% CI [0.548, 0.713]
H2: 0.782, 95% CI [0.721, 0.864]
```

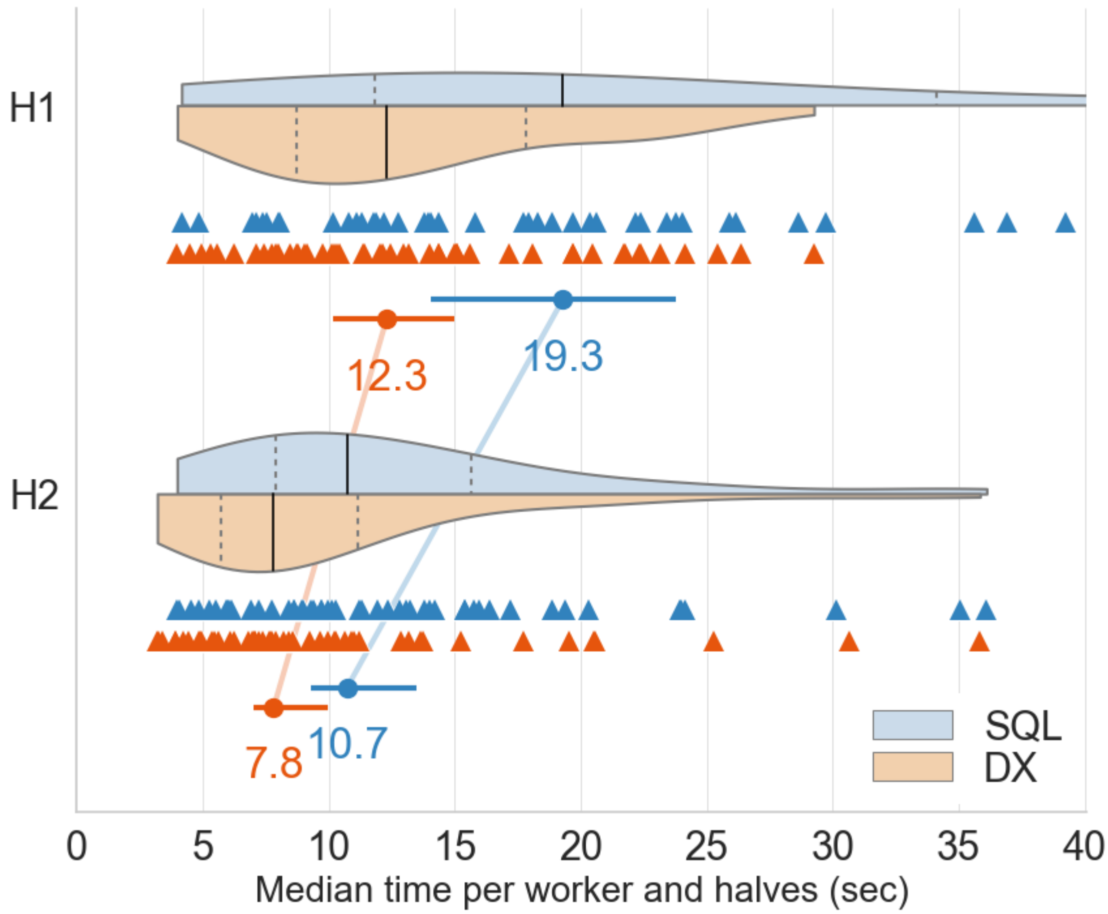
[12]:
```python
# Plot (uses df6, df7)

# Define pre-settings
figwidth = 10
figheight = 6
if VARIANT == 1:
    figheight = 8
xlab_size = 20
```

```
ylab_size = 20
figfont_size = 24

# Define consistent color maps
my_cmap_sns_light = [(0.7764705882352941, 0.8588235294117647, 0.
↪9372549019607843), (0.9921568627450981, 0.8156862745098039, 0.
↪6352941176470588)]        # light orange, light blue
my_cmap_sns_dark = [(0.19215686274509805, 0.5098039215686274, 0.
↪7411764705882353), (0.9019607843137255, 0.3333333333333333, 0.
↪050980392156862744)]      # dark orange, dark blue
my_cmap_dark = sns.color_palette(my_cmap_sns_dark, as_cmap=True)
my_cmap_light = sns.color_palette(my_cmap_sns_light, as_cmap=True)


# Create empty figure and plot the individual datapoints
fig, ax = plt.subplots(figsize=(figwidth,figheight))


# 1. Violinplots
axsns = sns.violinplot(x='median', y='section', data=df7,
                       hue='mode',
                       hue_order=['SQL', 'RD'],
                       split=True,   # half violinplots https://stackoverflow.
↪com/questions/53872439/half-not-split-violin-plots-in-seaborn
                       inner='quartile',
                       cut=0,                # 0 means ending sharp at end points
                       width=.4,
                       orient = 'h',
                       zorder=20,
                       palette = my_cmap_light,)

# change the medium default line to full (https://stackoverflow.com/questions/
↪60638344/quartiles-line-properties-in-seaborn-violinplot)
for l in axsns.lines[1::3]:
    l.set_linestyle('-')
    l.set_linewidth(1.2)
    l.set_color('black')
    # l.set_alpha(0.8)


# 2. Plot individual points
y_base = np.zeros(df6.values.shape[0])  # base vector to which to broadcast␣
↪y-tilt values
y_tilt_mode = [0.3, 0.38]
y_tilt_section = [0, 1]
for i, mode in enumerate(modes):
    for j, section in enumerate(sections):
```

```
        column = (mode, section)
        ax.plot(df6[column],
                y_base + y_tilt_mode[i] + y_tilt_section[j],
                # 'o',
                # '|',
                '^',
                alpha=1,
                zorder=20,      # higher means more visible
                markersize=11,
                markeredgewidth=0,
                # markerfacecolor='none',
                markerfacecolor=my_cmap_sns_dark[i],
                markeredgecolor=my_cmap_sns_dark[i],)
        ax.plot(df6[column],          # white background
                y_base + y_tilt_mode[i] + y_tilt_section[j],
                # 'o',
                # '|',
                '^',
                markersize=11,
                markeredgewidth=1,
                markerfacecolor='white',
                color ='white',
                linewidth = None,
                zorder=1,)


# # 3. Plot lines connecting points
# for idx in df6.index:
#     for i, mode in enumerate(modes):
#         for j in range(len(sections)-1):
#             start = (mode, sections[j])
#             end = (mode, sections[j+1])
#             ax.plot(df6.loc[idx, [start, end]],
#                     [y_tilt_mode[i] + y_tilt_section[j], y_tilt_mode[i] +␣
 ↪y_tilt_section[j+1]],
#                     color=my_cmap_sns_dark[i], linewidth=2, linestyle='-',␣
 ↪alpha=.2, zorder=0)


# 4. CI Errorbars & numbers
y_tilt_mode = [0.5, 0.55]
# y_tilt_section_bar = [0.23, 0.8]
# y_tilt_section_number = [0.19, 0.89]
for i, mode in enumerate(modes):
    for j, section in enumerate(sections):
        column = (mode, section)
        plt.errorbar(median_time[column], y_tilt_mode[i]+y_tilt_section[j],
```

```
                     xerr=np.array([[ci_delta[column][0]␣
↪,ci_delta[column][1]]]).T,
                     fmt='o', markersize=10,
                     lw = 3, alpha=1,
                     zorder=100,        # higher means more visible
                     #             capsize = 10, capthick = 4,
                     capsize = 0,
                     color = my_cmap_sns_dark[i]    # 'black'
                     )
        ax.text(median_time[column], y_tilt_mode[i]+y_tilt_section[j] + 0.18,␣
↪f'{median_time[column]:.1f}',
                horizontalalignment='center', color = my_cmap_sns_dark[i],
                fontsize=figfont_size)


# 5. Plot red line connecting medians
for i, mode in enumerate(modes):
    ax.plot([median_time[(mode, 'H1')], median_time[(mode, 'H2')]],
            [y_tilt_mode[i]+y_tilt_section[0],␣
↪y_tilt_mode[i]+y_tilt_section[1]],
            color=my_cmap_sns_dark[i], linewidth = 3, linestyle ='-',
            alpha=.3,
            zorder=0)


# #Additional settings
ax.set_xticks(range(0, 100, 5))
ax.set_xlabel('Median time per worker and halves (sec)', size = xlab_size)
ax.set_ylabel(None)
ax.set_xlim(0, 40.1)
ax.set_ylim(1.82, -0.25)
leg = plt. legend(loc='lower right',
                  borderaxespad= 0.2,
                  frameon = True,
                  labelspacing = 0.1)
leg.get_frame().set_alpha(1)
leg.get_frame().set_linewidth(0.0)
for text, text2 in zip(leg.get_texts(), ['SQL', 'DX']):
    text.set_text(text2)

plt.grid(axis = 'x', linewidth = 0.5, color = 'lightgray')
sns.despine()               # remove bounding box

if savefig:
    plt.savefig(fig_dir + f'/q2_figure_variant{VARIANT}.pdf',␣
↪bbox_inches='tight')
```

```
    plt.savefig(fig_dir + f'/q2_figure_variant{VARIANT}.svg',␣
↪bbox_inches='tight')
```



6 Question 3: four patterns

(1) calculate the median time per pattern (4) across the two modes (2). Thus 8 values.
(2) show repeated measure violin plot figure

[13]:
```
# Create df8, df9
df0 = df_filtered_data[['worker_id', 'pattern', 'time', 'mode']]
# print(df)

df8 = df0.groupby(['worker_id', 'mode', 'pattern']).time.agg(['median'])      ␣
↪ # for each worker, calculate median for both modes
df8.reset_index(inplace=True)
# print('df8:')
# display(df8)
```

```
# Pivot to have one row per worker
df9 = pd.pivot_table(df8, values=['median'], index=['worker_id'],␣
↪columns=['mode', 'pattern'])
df9=df9.droplevel(0, axis=1)
print('df9:')
display(df9)

# Median of median task time for each mode and section (for error plots)
modes = ['RD', 'SQL']
patterns = [1, 2, 3, 4]
median_time = {}
ci = {}
ci_delta = {}
for mode in modes:
    for pattern in patterns:
        column = (mode, pattern)
        median_time[column] = np.median(df9[column])
        ci[column] = scipybootstrap((df9[column],), statistic=np.median,
                                    n_resamples=BOOTSTRAPSAMPLES,
                                    confidence_level=BOOTSTRAPCONFIDENCE,
                                    method='percentile',
                                    axis=0).confidence_interval        #convert␣
↪array to sequence
        ci_delta[column] = [median_time[column] - ci[column].low, ci[column].
↪high - median_time[column]]
        print(f'{mode}, {pattern}: {median_time[column]:.3f}, 95% CI␣
↪[{ci[column].low:.3f}, {ci[column].high:.3f}]')

# Median ratio RD/SQL per pattern
for pattern in patterns:
    column = ('ratio', pattern)
    df9['ratio', pattern] = df9['RD', pattern] / df9['SQL', pattern]
    median_time[column] = np.median(df9[column])
    ci[column] = scipybootstrap((df9[column],), statistic=np.median,
                                n_resamples=BOOTSTRAPSAMPLES,
                                confidence_level=BOOTSTRAPCONFIDENCE,
                                method='percentile',
                                axis=0).confidence_interval        #convert␣
↪array to sequence
    ci_delta[column] = [median_time[column] - ci[column].low, ci[column].high -␣
↪median_time[column]]
    print(f'ratio, {pattern}: {median_time[column]:.3f}, 95% CI [{ci[column].
↪low:.3f}, {ci[column].high:.3f}]')


# print(median_time)
```

```
# print(ci)
# print(ci_delta)
```

df9:

mode	RD				SQL			
pattern	1	2	3	4	1	2	3	4
worker_id								
0	10.743	8.359	6.698	6.008	14.905	36.797	59.405	20.626
2	5.763	8.801	13.316	12.110	21.611	13.085	13.359	9.433
5	4.341	11.566	12.451	5.909	16.144	17.267	22.487	15.869
6	4.598	12.098	10.607	9.812	17.572	14.300	30.809	11.163
10	11.838	13.354	9.617	7.479	22.340	12.156	29.614	9.487
17	5.732	6.175	6.480	4.373	5.058	6.283	6.177	6.156
21	9.734	23.887	6.872	7.048	15.326	12.707	13.424	12.530
28	7.678	7.417	17.119	14.781	10.409	13.274	44.967	13.194
32	13.519	34.927	11.123	16.664	26.592	37.562	17.396	24.245
39	4.110	8.950	9.537	6.993	15.761	8.436	8.331	11.702
42	9.339	15.576	12.989	11.909	51.422	23.992	20.569	42.055
43	39.382	8.934	7.427	7.880	12.249	10.734	45.501	10.773
50	9.282	8.398	5.043	4.728	28.998	10.566	10.888	6.618
52	14.657	12.528	13.305	17.442	14.966	12.585	26.383	55.478
57	5.646	6.652	18.717	7.854	7.042	9.678	10.959	14.056
58	20.224	10.258	14.566	9.842	14.597	7.352	8.555	7.309
60	10.117	16.352	10.577	11.575	9.776	10.035	10.053	9.177
66	9.035	9.623	8.552	7.316	9.283	13.579	12.380	11.319
72	24.194	30.939	20.398	14.492	31.044	37.365	28.279	15.788
75	19.223	13.901	10.663	23.811	14.259	22.319	11.861	12.069
77	10.902	29.985	20.251	25.949	28.447	36.307	22.822	30.817
80	21.343	15.039	16.091	27.936	61.486	27.852	31.846	29.410
81	8.212	11.478	29.560	15.845	44.715	11.319	21.013	11.835
83	7.583	11.575	20.982	10.124	38.788	44.747	42.904	25.588
87	16.300	19.988	22.802	24.525	32.172	55.120	51.922	76.430
89	13.842	19.970	25.956	21.399	50.622	56.785	58.758	42.281
91	4.175	7.650	6.031	4.373	4.236	9.535	12.401	3.961
92	13.043	29.417	15.970	20.207	20.310	28.922	11.677	29.362
96	4.561	15.671	13.224	6.077	34.899	23.767	45.007	57.634
110	25.987	21.697	25.157	16.268	25.159	24.812	46.842	22.615
115	10.288	10.971	10.514	12.494	19.202	21.965	25.297	13.241
117	4.501	5.051	4.184	4.068	4.128	3.881	5.637	4.769
119	3.929	11.073	9.399	8.395	5.261	4.867	4.465	5.851
121	8.557	12.180	7.963	7.330	9.715	14.059	10.139	6.388
125	6.665	6.969	7.196	4.274	12.033	13.067	8.127	11.806
130	6.500	7.206	6.030	10.046	5.974	6.951	7.649	6.117
136	9.207	11.666	12.430	7.110	17.006	20.848	10.535	12.446
141	4.412	18.003	30.385	5.393	5.205	6.478	4.095	4.957
143	2.710	4.406	3.758	2.865	5.851	6.253	7.363	3.783
146	3.819	7.717	11.749	4.819	13.605	17.785	15.294	11.594

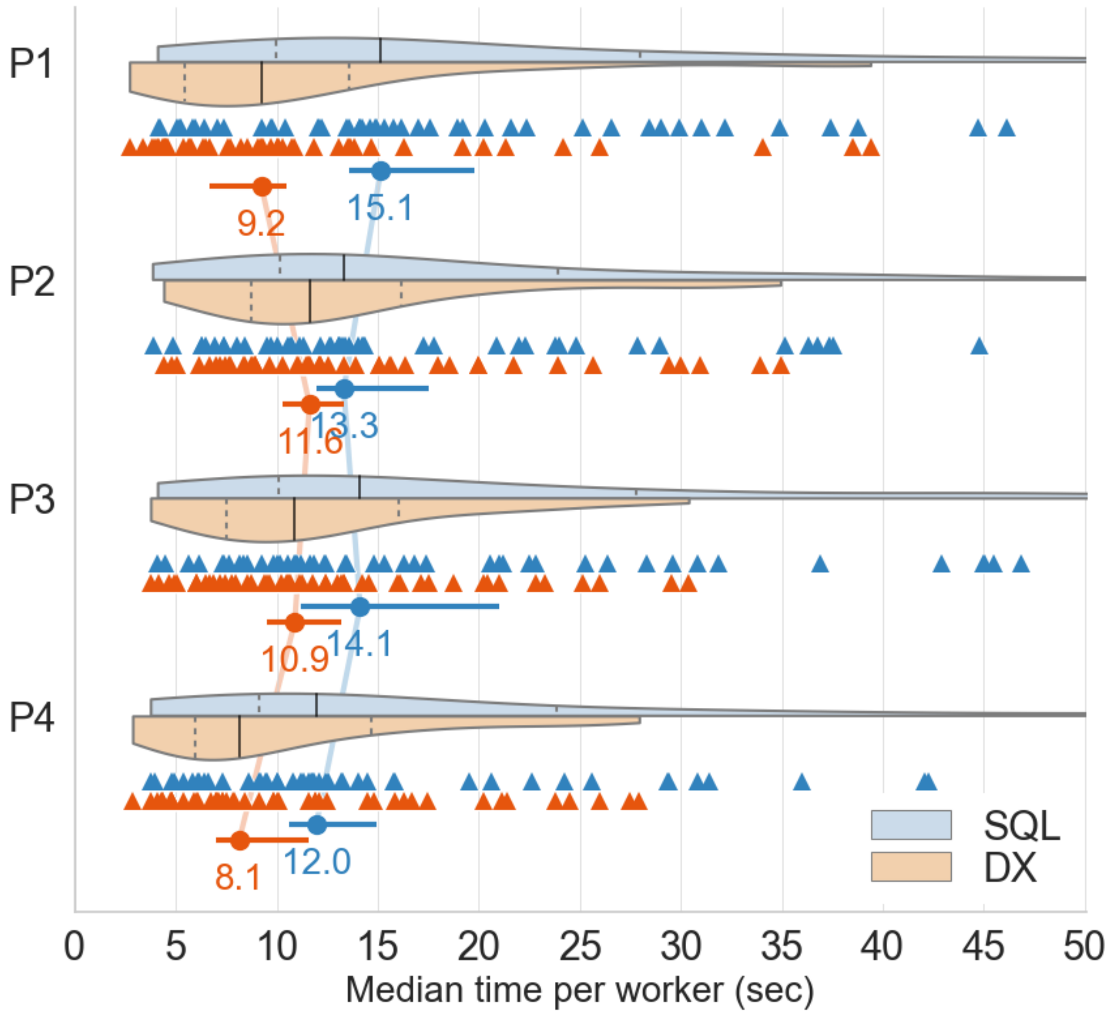
```
148       38.515 25.681 17.514  9.155 29.904 14.229 11.193 14.480
153       13.610 12.074 14.245 21.149 46.125 63.980 21.224 54.346
154       13.622 15.070 11.265  6.736 14.081 13.113 16.783 10.037
158        6.357  8.718  6.011  3.740 18.930  8.051  9.223 36.023
159        5.360 11.519 10.209  6.990 13.451 14.365 14.837 19.510
162        3.383 12.178  4.913  4.809  7.378 11.100  9.827  8.625
165       10.000  8.908  8.761  4.774 14.120 10.698 10.992 10.002
166       10.038 18.571  7.737 11.921 12.090 13.363 16.262  9.098
168       34.071 33.947 23.260 27.477 37.389 35.180 36.904 31.392
169        4.458  4.822  4.658  4.772  6.428  7.386  7.333  5.377

RD, 1: 9.245, 95% CI [6.665, 10.471]
RD, 2: 11.620, 95% CI [10.258, 13.354]
RD, 3: 10.893, 95% CI [9.508, 13.224]
RD, 4: 8.137, 95% CI [7.021, 11.575]
SQL, 1: 15.146, 95% CI [13.605, 19.756]
SQL, 2: 13.319, 95% CI [11.951, 17.526]
SQL, 3: 14.130, 95% CI [11.193, 21.013]
SQL, 4: 11.952, 95% CI [10.600, 14.922]
ratio, 1: 0.639, 95% CI [0.486, 0.779]
ratio, 2: 0.830, 95% CI [0.698, 0.965]
ratio, 3: 0.662, 95% CI [0.533, 0.768]
ratio, 4: 0.712, 95% CI [0.598, 0.859]
```

[14]:
```python
# needs df8 for violin, df9 for points, dictionaries (median_time, ci,␣
↪ci_delta) for error plots, df6 for individual points

modes = ['SQL', 'RD']
patterns = [1, 2, 3, 4]
y_tilt_section = [0, 1, 2, 3]

# Define pre-settings
figwidth = 10
figheight = 9
xlab_size = 20
ylab_size = 20
figfont_size = 20

# Define consistent color maps
my_cmap_sns_light = [(0.7764705882352941, 0.8588235294117647, 0.
↪9372549019607843), (0.9921568627450981, 0.8156862745098039, 0.
↪6352941176470588)]        # light orange, light blue
my_cmap_sns_dark = [(0.19215686274509805, 0.5098039215686274, 0.
↪7411764705882353), (0.9019607843137255, 0.3333333333333333, 0.
↪050980392156862744)]      # dark orange, dark blue
my_cmap_dark = sns.color_palette(my_cmap_sns_dark, as_cmap=True)
my_cmap_light = sns.color_palette(my_cmap_sns_light, as_cmap=True)
```

```python
# Create empty figure and plot the individual datapoints
fig, ax = plt.subplots(figsize=(figwidth,figheight))


# 1. Violinplots
axsns = sns.violinplot(x='median', y='pattern', data=df8,
                       hue='mode',
                       hue_order=['SQL', 'RD'],
                       split=True,   # half violinplots https://stackoverflow.
 ↪com/questions/53872439/half-not-split-violin-plots-in-seaborn
                       inner='quartile',
                       cut=0,                 # 0 means ending sharp at end points
                       width=.4,
                       orient = 'h',
                       zorder=20,
                       palette = my_cmap_light,)

# change the medium default line to full (https://stackoverflow.com/questions/
 ↪60638344/quartiles-line-properties-in-seaborn-violinplot)
for l in axsns.lines[1::3]:
    l.set_linestyle('-')
    l.set_linewidth(1.2)
    l.set_color('black')
    l.set_alpha(0.8)


# 2. Plot individual points
y_base = np.zeros(df6.values.shape[0])  # base vector to which to broadcast␣
 ↪y-tilt values
y_tilt_mode = [0.3, 0.39]
for i, mode in enumerate(modes):
    for j, pattern in enumerate(patterns):
        column = (mode, pattern)
        ax.plot(df9[column],
                y_base + y_tilt_mode[i] + y_tilt_section[j],
                '^',
                alpha=1,
                zorder=20,      # higher means more visible
                markersize=10,
                markeredgewidth=0,
                # markerfacecolor='none',
                markerfacecolor=my_cmap_sns_dark[i],
                markeredgecolor=my_cmap_sns_dark[i],)
        ax.plot(df9[column],    # white background behind the markers, but in␣
 ↪front of the connecting lines
                y_base + y_tilt_mode[i] + y_tilt_section[j],
                '^',
```

```
                markersize=10,
                markeredgewidth=1,
                markerfacecolor='white',
                color ='white',
                linewidth = None,
                zorder=1,)


# # 3. Plot lines connecting individual points
# for idx in df9.index:
#     for i, mode in enumerate(modes):
#         for j in range(len(patterns)-1):
#             start = (mode, patterns[j])
#             end = (mode, patterns[j+1])
#             ax.plot(df9.loc[idx, [start, end]],
#                     [y_tilt_mode[i] + y_tilt_section[j], y_tilt_mode[i] +␣
↪y_tilt_section[j+1]],
#                     color =my_cmap_sns_dark[i], linewidth = 2, linestyle␣
↪='-', alpha = .2, zorder=0)


# 4. CI Errorbars & numbers
y_tilt_mode = [0.5, 0.57]
for i, mode in enumerate(modes):
    for j, section in enumerate(patterns):
        column = (mode, section)
        plt.errorbar(median_time[column],
                     y_tilt_mode[i]+y_tilt_section[j],
                     xerr=np.array([[ci_delta[column][0]␣
↪,ci_delta[column][1]]]).T,
                     fmt='o', markersize=10,
                     capsize = 0,
                     lw = 3, alpha=1,
                     zorder=100,         # higher means more visible
                     color = my_cmap_sns_dark[i])     # 'black'
        ax.text(median_time[column], y_tilt_mode[i]+y_tilt_section[j] + 0.22,␣
↪f'{median_time[column]:.1f}',
                horizontalalignment='center', color = my_cmap_sns_dark[i],
                fontsize=figfont_size)


# 5. Plot red line connecting medians
for i, mode in enumerate(modes):
    for j in range(len(patterns)-1):
        start = (mode, patterns[j])
        end = (mode, patterns[j+1])
        ax.plot([median_time[start], median_time[end], ],
```

```
                [y_tilt_mode[i] + y_tilt_section[j], y_tilt_mode[i] +␣
↪y_tilt_section[j+1]],
                color =my_cmap_sns_dark[i], linewidth = 3, linestyle ='-',␣
↪alpha = .3, zorder=0)



# # #Additional settings
ax.set_xticks(range(0, 100, 5))
ax.set_xlabel('Median time per worker (sec)', size = xlab_size)
ax.set_ylabel(None)
ax.set_yticklabels(['P1', 'P2', 'P3', 'P4'])
ax.set_xlim(0, 50.05)
ax.set_ylim(3.9, -0.25)
leg = plt.legend(loc='lower right',
                 borderaxespad= 0.2,
                 frameon = True,
                 labelspacing = 0.1)
leg.get_frame().set_alpha(1)
leg.get_frame().set_linewidth(0.0)

for text, text2 in zip(leg.get_texts(), ['SQL', 'DX']):
    text.set_text(text2)

plt.grid(axis = 'x', linewidth = 0.5, color = 'lightgray')
sns.despine()               # remove bounding box

if savefig:
    plt.savefig(fig_dir + f'/q3_figure_variant{VARIANT}.pdf',␣
↪bbox_inches='tight')
    plt.savefig(fig_dir + f'/q3_figure_variant{VARIANT}.svg',␣
↪bbox_inches='tight')
```



7 Question 4: Correctness

(1) take mean correct over all questions and all users answered in SQL or RD (32/2*13) and the difference in correctness score
(2) calculate 95% CI for each

[15]:
```
# dfq4a: Create two columns mode and mean, with 2 rows per worker
dfq4a = df_filtered_data.groupby(['worker_id', 'mode']).correct.agg(['mean'])
dfq4a.reset_index(inplace=True)
# display(dfq4a)

# dfq4b: Pivot to have one row per worker
dfq4b = pd.pivot_table(dfq4a, values=['mean'], index=['worker_id'],␣
 ↪columns=['mode'])
dfq4b=dfq4b.droplevel(0, axis=1)
```

```
dfq4b['diff'] = dfq4b['RD'] - dfq4b['SQL']
print("dfq4b:\n")
display(dfq4b)

# Calculate fraction of those better in either mode
num_SQLbetter = np.where(dfq4b['SQL'] > dfq4b['RD'], 1, 0).sum()
num_RDbetter = np.where(dfq4b['SQL'] < dfq4b['RD'], 1, 0).sum()
num_workers =len(dfq4b)
print(f'{num_SQLbetter}/{num_workers} ({num_SQLbetter/num_workers:.3f}) better␣
↪with SQL.')
print(f'{num_RDbetter}/{num_workers} ({num_RDbetter/num_workers:.3f}) better␣
↪with RD.')
print(f'{num_workers-num_RDbetter-num_SQLbetter}/{num_workers}␣
↪({(num_workers-num_RDbetter-num_SQLbetter)/num_workers:.3f}) equally good.')␣
↪ # (7/6/2023): fixed: was incorrectly deducing RD twice, instead of␣
↪num-RD-SQL

# Mean of mean correctness for each mode Plus 95% CI
modes_diff = ['RD', 'SQL', 'diff']
mean_correct = {}
ci = {}
ci_delta = {}
for mode in modes_diff:
    mean_correct[mode] = np.mean(dfq4b[mode])
    ci[mode] = scipybootstrap((dfq4b[mode],), statistic=np.mean,␣
↪n_resamples=BOOTSTRAPSAMPLES, confidence_level=BOOTSTRAPCONFIDENCE,␣
↪method='percentile', axis=0).confidence_interval        #convert array to␣
↪sequence
    ci_delta[mode] = [mean_correct[mode] - ci[mode].low, ci[mode].high -␣
↪mean_correct[mode]]

print(f"mean RD correct = {mean_correct['RD']:.3f}, 95% CI [{ci['RD'].low:.3f},␣
↪{ci['RD'].high:.3f}]")
print(f"mean SQL correct = {mean_correct['SQL']:.3f}, 95% CI [{ci['SQL'].low:.
↪3f}, {ci['SQL'].high:.3f}]")
print(f"mean difference in correctness = {mean_correct['diff']:.3f}, 95% CI␣
↪[{ci['diff'].low:.3f}, {ci['diff'].high:.3f}]")
```

```
dfq4b:


mode          RD    SQL   diff
worker_id
0          0.938 0.312  0.625
2          1.000 1.000  0.000
5          1.000 1.000  0.000
6          1.000 1.000  0.000
```

10	1.000	0.875	0.125
17	0.500	0.500	0.000
21	1.000	1.000	0.000
28	0.875	1.000	-0.125
32	0.875	0.125	0.750
39	1.000	0.812	0.188
42	1.000	0.875	0.125
43	1.000	0.750	0.250
50	1.000	0.812	0.188
52	0.938	0.688	0.250
57	0.812	0.938	-0.125
58	0.625	0.688	-0.062
60	0.875	1.000	-0.125
66	1.000	1.000	0.000
72	0.938	1.000	-0.062
75	0.812	0.625	0.188
77	0.875	0.312	0.562
80	0.938	0.875	0.062
81	1.000	1.000	0.000
83	0.938	0.750	0.188
87	1.000	0.625	0.375
89	0.875	0.188	0.688
91	1.000	1.000	0.000
92	0.625	0.438	0.188
96	1.000	0.438	0.562
110	0.938	0.375	0.562
115	1.000	1.000	0.000
117	1.000	0.125	0.875
119	1.000	0.188	0.812
121	1.000	0.938	0.062
125	0.938	0.875	0.062
130	1.000	1.000	0.000
136	1.000	1.000	0.000
141	0.875	0.250	0.625
143	1.000	0.938	0.062
146	0.750	0.500	0.250
148	0.875	0.188	0.688
153	0.812	0.500	0.312
154	1.000	1.000	0.000
158	0.812	0.250	0.562
159	0.938	1.000	-0.062
162	1.000	0.938	0.062
165	1.000	0.938	0.062
166	0.938	1.000	-0.062
168	0.875	0.250	0.625
169	1.000	1.000	0.000

7/50 (0.140) better with SQL.

```
30/50 (0.600) better with RD.
13/50 (0.260) equally good.
mean RD correct = 0.924, 95% CI [0.891, 0.953]
mean SQL correct = 0.718, 95% CI [0.632, 0.800]
mean difference in correctness = 0.206, 95% CI [0.131, 0.285]
```

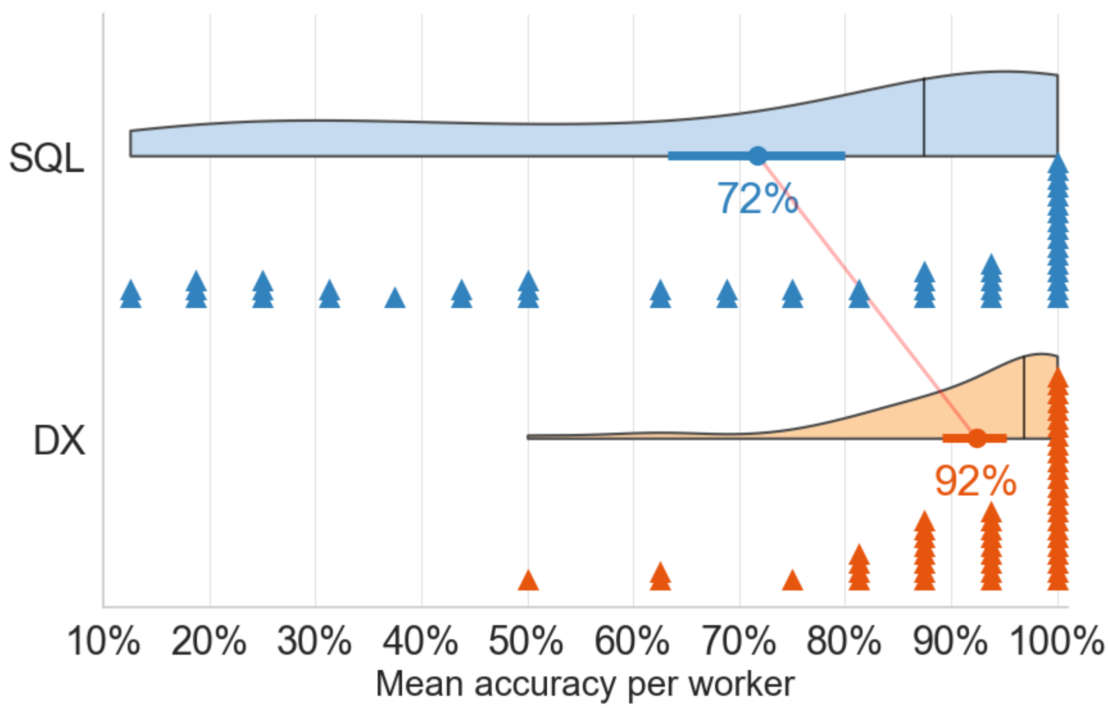
[16]:

```
# Removing 'diff' from variables
dfq4c=dfq4b.copy()
dfq4c.drop('diff', inplace=True, axis=1)
modes = ['RD', 'SQL']

# Define pre-settings
figwidth = 9.7
figheight = 6
xlab_size = 20
ylab_size = 20
figfont_size = 24

# Define consistent color maps
my_cmap_sns_light = [(0.9921568627450981, 0.8156862745098039, 0.
 ↪6352941176470588), (0.7764705882352941, 0.8588235294117647, 0.
 ↪9372549019607843)]        # light blue, light orange
my_cmap_sns_dark = [(0.9019607843137255, 0.3333333333333333, 0.
 ↪050980392156862744), (0.19215686274509805, 0.5098039215686274, 0.
 ↪7411764705882353)]      # dark blue, dark orange
my_cmap_dark = sns.color_palette(my_cmap_sns_dark, as_cmap=True)
my_cmap_light = sns.color_palette(my_cmap_sns_light, as_cmap=True)

# Create empty figure and plot the individual datapoints
fig, ax = plt.subplots(figsize=(figwidth,figheight))

# 1. Violinplots
axsns = sns.violinplot(x='mean', y='mode', data=dfq4a,
                       hue=True, hue_order=[False, True], split=True,   # half␣
 ↪violinplots https://stackoverflow.com/questions/53872439/
 ↪half-not-split-violin-plots-in-seaborn
                       inner='quartile',
                       cut=0,               # 0 means ending sharp at end points
                       width=.6,
                       orient = 'h',
                       zorder=20,)

# change the medium default line to full (https://stackoverflow.com/questions/
 ↪60638344/quartiles-line-properties-in-seaborn-violinplot)
for l in axsns.lines[1::3]:
    l.set_linestyle('-')
    l.set_linewidth(1.2)
```

```
    l.set_color('black')
    l.set_alpha(0.8)
for i in [0, 2, 3, 5]:          # remove the 25% and 75% quartiles
    l = axsns.lines[i]
    l.set_linestyle('-')
    l.set_linewidth(0)
    l.set_color('red')
    l.set_alpha(0.8)

# Apply colorscheme to violinplots https://stackoverflow.com/questions/70442958/
 ↪seaborn-how-to-apply-custom-color-to-each-seaborn-violinplot
from matplotlib.collections import PolyCollection
for ind, violin in enumerate(axsns.findobj(PolyCollection)):
    violin.set_facecolor(my_cmap_light[ind])
# plt.setp(ax.collections, alpha=.999)  # semi-transparent (https://
 ↪stackoverflow.com/questions/62597959/seaborn-violinplot-transparency)


# 2. Plot individual points [new dot plot]
y_tilt = -0.5       # how far below each violinplot

def dotplot(input_x, y0, delta, **args):
    unique_values, counts = np.unique(input_x, return_counts=True)  # Count how␣
 ↪many times does each value occur

    # Convert 1D input into 2D array
    scatter_x = [] # x values
    scatter_y = [] # corresponding y values
    for idx, value in enumerate(unique_values):
        for counter in range(0, counts[idx]):
            scatter_x.append(value)
            scatter_y.append(y0+counter*delta)
    plt.scatter(scatter_x, scatter_y, **args)

for i, col in enumerate(dfq4c):
    dotplot(input_x=dfq4c[col], y0=y_tilt + i, delta=0.03,      # y-axis tilt␣
 ↪change with each column
            marker='^',
            alpha=1,
            zorder=20,      # higher means more visible
            color=my_cmap_dark[i],
            s=150,
            linewidth=0,)


# 4. Plot red line connecting means
```

```
ax.plot(np.mean(dfq4c, axis=0), [0, 1], color ='red', linewidth = 2, linestyle␣
↪='-',
        alpha = .3,
        zorder=4,
        )

# 5. CI Errorbars
for i, mode in enumerate(modes):
    plt.errorbar(mean_correct[mode], i, xerr=np.array([[ci_delta[mode][0],␣
↪ci_delta[mode][1]]]).T,
                 fmt='o', markersize=10,
                 lw = 5, alpha=1,
                 zorder=100,        # higher means more visible
                 capsize = 0,
                 # capthick = 4,
                 capthick = 0,
                 # color = 'black',
                 color=my_cmap_dark[i],
                 )   # my_cmap[1])

    # ax.text(median_time[column], y_tilt_mode[i]+y_tilt_section[j] + 0.18,␣
↪f'{median_time[column]:.1f}',
    #         horizontalalignment='center', color = my_cmap_sns_dark[i],␣
↪fontsize=20)
    ax.text(mean_correct[mode],
            i-0.2,
            # f'{mean_correct[mode]:.2f}', horizontalalignment='center',
            f'{100*mean_correct[mode]:.0f}%', horizontalalignment='center',
            # color='black',
            color=my_cmap_dark[i],
            fontsize=figfont_size)
    # ax.text(ci[mode].low, i+0.1, f'{ci[mode].low:.2f}',␣
↪horizontalalignment='center', color='black', fontsize=20)
    # ax.text(ci[mode].high, i+0.1, f'{ci[mode].high:.2f}',␣
↪horizontalalignment='center', color='black', fontsize=20)


#Additional settings
# ax.set_yticklabels(modes, size= ylab_size)
# ax.set_xlim(0.7499, 1.0003)
# ax.set_xlim(0.74, 1.01)
ax.set_xlim(0.1, 1.01)
if VARIANT == 1:
    ax.set_xticks(np.linspace(0.1, 1, num=10))
if VARIANT == 3:
    ax.set_xlim(0.8, 1.01)
```

```
if VARIANT == 4:
    ax.set_xlim(0.4, 1.01)
ax.set_ylim(-0.6, 1.5)
ax.set_xlabel('Mean accuracy per worker', size = xlab_size)
ax.set_ylabel(None)
ax.set_yticklabels(['DX', 'SQL'])
ax.legend_.remove()
plt.grid(axis = 'x', linewidth = 0.5, color = 'lightgray')
sns.despine()                # remove bounding box

# import matplotlib.ticker as mtick
ax.xaxis.set_major_formatter(mtick.PercentFormatter(1.0))          # show in␣
↪percentage

if savefig:
    plt.savefig(fig_dir + f'/q4_figure_variant{VARIANT}.pdf',␣
↪bbox_inches='tight')
    plt.savefig(fig_dir + f'/q4_figure_variant{VARIANT}.svg',␣
↪bbox_inches='tight')
```

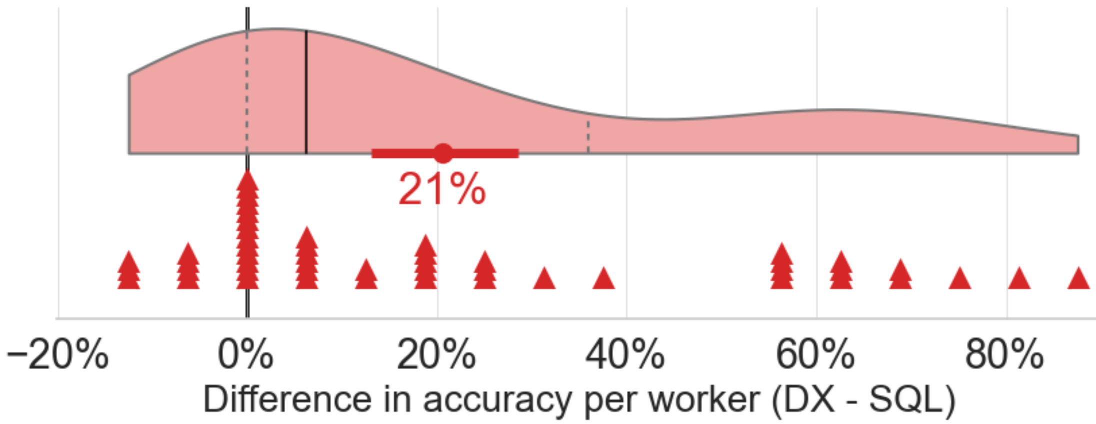


8 Question 4. Figure 4b

[17]:
```
# Define figure settings
figwidth = 10
figheight = 3
xlab_size = 20
ylab_size = 20
figfont_size = 24

# Define consistent color maps
my_cmap_sns_dark = [(0.8392156862745098, 0.15294117647058825, 0.
↪1568627450980392)]
my_cmap_sns_light = [(0.984313725490196, 0.6039215686274509, 0.6)]
my_cmap_dark = sns.color_palette(my_cmap_sns_dark, as_cmap=True)
my_cmap_light = sns.color_palette(my_cmap_sns_light, as_cmap=True)

# Create data frame for split violinplot
sample = np.array(dfq4b['diff'])                                            ␣
↪      # extract the sample and then create the boostrapped medians
data_difference = dfq4b['diff']
dfvp = pd.DataFrame()
dfvp["values"] = sample
dfvp["all"] = ""                                         # attribute that is␣
↪shared by all entries

# Create empty figure and plot the individual datapoints
fig, ax = plt.subplots(figsize=(figwidth,figheight))


# 1. Violinplot
axsns = sns.violinplot(x='values', y='all',     # y='all' just need to group␣
↪both types together
                       data=dfvp,
                       hue = True, hue_order = [False, True],
                       split = True, inner = 'quartile',
                       cut=0,                     # 0 means ending sharp at end␣
↪points
                       width=.6, scale = 'width',
                       # dodge = False,         # When using ``hue`` nesting,␣
↪setting this to ``True`` will separate the strips for different hue levels␣
↪along the categorical axis.
                       orient = 'h',
                       color=my_cmap_light[0],
                       zorder = 3)

# change the medium default linke to full
for l in axsns.lines[1::3]:
```

```
    l.set_linestyle('-')
    l.set_linewidth(1.5)
    l.set_color('black')
    l.set_alpha(0.8)


# 2. Plot individual points [new dot plot]
y_tilt = -0.3                                     # Set some delta for the␣
 ↪points below the violinplot

def dotplot(input_x, y0, delta, **args):
    unique_values, counts = np.unique(input_x, return_counts=True)  # Count how␣
 ↪many times does each value occur

    # Convert 1D input into 2D array
    scatter_x = [] # x values
    scatter_y = [] # corresponding y values
    for idx, value in enumerate(unique_values):
        for counter in range(0, counts[idx]):
            scatter_x.append(value)
            scatter_y.append(y0+counter*delta)
    plt.scatter(scatter_x, scatter_y, **args)

dotplot(input_x=data_difference, y0=y_tilt, delta=0.02,
        marker='^',
        alpha=1,
        zorder=20,      # higher means more visible
        color=my_cmap_dark[0],
        s=150,
        linewidth=0,)


# 3. CI Errorbars & show numbers
# axeb = plt.errorbar(median_ratio, 0, xerr=np.array([[ci_ratio_delta[0],␣
 ↪ci_ratio_delta[1]]]).T,
axeb = plt.errorbar(mean_correct['diff'], 0, xerr=np.
 ↪array([[ci_delta['diff'][0], ci_delta['diff'][1]]]).T,
                    fmt='o',
                    markersize=10, alpha=1,
                    # lw = 3,
                    lw = 5,
                    zorder=100,      # higher means more visible
                    capsize = 0,     # 10
                    # capthick = 4,
                    capthick = 0,
                    # color = 'black',
                    color = my_cmap_dark[0],
```

```
                    )



# 4. vertical bar for x-axis = 1
plt.plot([0, 0], [-10, 10], color = 'black', zorder = 0, linewidth=2)


meandiff = np.mean(sample)           # rename
# ax.text(meandiff, -0.12, f'{meandiff:.2f}', horizontalalignment='center',
ax.text(meandiff, -0.12, f'{100*meandiff:.0f}%', horizontalalignment='center',
        # color='black',
        color = my_cmap_dark[0],
        fontsize=figfont_size)
# ax.text(ci_ratio.low, 0.04, f'{100*ci_ratio.low:.1f}%',␣
 ↪horizontalalignment='center', color='black', fontsize=20)
# ax.text(ci_ratio.high, 0.04, f'{100*ci_ratio.high:.1f}%',␣
 ↪horizontalalignment='center', color='black', fontsize=20)




# Additional settings
ax.set_ylim(-0.4, 0.35)
ax.set_ylabel(None)         # remove the 'all'

ax.set_xticks(np.linspace(-1, 1, num=11))
# ax.set_xticks(np.linspace(-1, 1, num=21), minor=True)
ax.set_xlim(-0.201, 0.901)

if VARIANT == 3:
    ax.set_xticks(np.linspace(-0.2, 0.2, num=9))
    ax.set_xlim(-0.151, 0.201)
if VARIANT == 4:
    ax.set_xlim(-0.201, 0.601)
ax.set_xlabel('Difference in accuracy per worker (DX - SQL)', size = xlab_size)

sns.despine(trim=False, left=True)              # remove bounding box
plt.grid(axis = 'x', linewidth = 0.5, color = 'lightgray', which='both')
ax.legend_.remove()
ax.xaxis.set_major_formatter(mtick.PercentFormatter(1.0, decimals=0), )        ␣
 ↪  # show in percentage

if savefig:
    plt.savefig(fig_dir + f'/q4_figure2_variant{VARIANT}.pdf',␣
 ↪bbox_inches='tight')
    plt.savefig(fig_dir + f'/q4_figure2_variant{VARIANT}.svg',␣
 ↪bbox_inches='tight')
```



9 Print user feedback

Prints all comments received from participants who passed the requirements (0.5 correctness)

[18]:
```
feedback = dfresults.loc[dfresults.worker_id.isin(dftemp.index),'feedback']
for i, text in enumerate(feedback):
    print(f'{i}: "{text}"')
```

```
0: "I found the tutorial to be very helpful in understanding the queries. In
particular, the side-by-side examples were very helpful. "
1: "Good"
2: "good little bit in deficit in the coding"
3: "the instructions are quite clear and easy to follow, the diagrams are easy
to understand and questions are not very difficult. And the tutorials especially
the last summary page is quite helpful. I think the diagrams can be applied to
my own usage but I am not quite sure now. "
4: "risky "
5: "It's very understanding and easy to play, also interesting too."
6: "good survey . i really love it. this is very helpful for some database
studies. "
7: "the example was very helpful
None"
8: "GOOD"
9: "nan"
10: "Really very nice survey I enjoyed lot to do SQL very easy. Thank you so
much. I m very Happy to complete This survey."
11: "It is easy and creates curiosity while answering. It's all good."
12: "nan"
13: "I will learn more and feel good for contain this survey."
14: "GOOD"
15: "Hard "
16: "It is easy and very curious to participate."
```

17: "none"
18: "The SQL queries was understanding, No that's are good, it's most useful of query patterns."
19: "1.your diagrams and explanation are awesome to understand.
2.All of your queries and content are interesting that just like daily activities.
3.I fix my mid itself some logical pattern to do your hit fast .
3. your organizational chart based explanation is very useful. Diagram easily understand compare to text."
20: "nan"
21: "nan"
22: "All Diagrams of SQL queries helpful for understanding the queries."
23: "Yes, the diagrams were helpful. It was very easy to understand both code and diagram. I found the exercise to be very straightforward and understandable. I noted the visual patterns in the diagrams, and the syntax patterns in the code. The last page of the tutorial with a summary of the diagrams was most helpful. I don't know how these diagrams would be helpful in real life. I can't imagine a scenario wherein that would be true."
24: "none"
25: "Very hard "
26: "nan"
27: "good"
28: "NONE"
29: "I found the Diagram of the SQL to be extremely helpful in determining the answer.

The overview of the four query patterns I liked.

The Diagrams I think were much easier to understand than the SQL.

After answering a few questions I felt I got the patterns down quicker. The diagrams felt much easier to answer than the SQL questions.

Overview of the four patterns was helpful.

Yes."
30: "your SQl queries very helpful that has presented with daily activities.
Your diagrams awesome that has great explanation to understand even beginner.
I believe your both pattern useful.
Tutorials helped very much for my knowledge.
It is interesting task to do in future also"
31: "NONE"
32: "NONE"
33: "The diagrams were very easy to understand. The dotted lines helped group the queries. "
34: "It was great study for me."
35: "The instructions/diagrams were quite clear.
The logical patterns became more apparent with practice, even making mistakes

```
helped firm up future comprehension.
"
36: "nan"
37: "none"
38: "good survey. i really enjoyed it . i know some database knowledge from this
study."
39: "It's very interesting to participate and easy to understand."
40: "I found it helpful when using the guide. They were all useful. I could see
the pattern. The examples helped the most. Thank you."
41: "The SQL  PDF really help to me to do this survey.
It's really understandable.
First 5 question I unable to understand after that I gave all answer correctly.
I take some practice and time.
Thank You!!"
42: "This survey understandable I adapted quickly.
Instruction PDF clearly explained
I suggest Add extra 3 question to practice to make any mistake to do complete
this task"
43: "good"
44: "I found your diagrams very helpful in understanding the queries. At first I
didn't get it, but after staring at the diagrams for a few minutes it clicked
and everything became super simple. I saw the patterns and it became just
looking for the correct pattern to know which query was being used."
45: "good"
46: "Hard one "
47: "nan"
48: "good"
49: "nan"
```

[19]:
```
# end
```

[20]:
```
# end
```