Assessment 2

December 3, 2021

1 Assessment 2

This assessment involves reproducing an experimental analysis using an openly available dataset. The datasets and accompanying research paper can be found at: https://dl.acm.org/citation.cfm?id=3174220

Please follow on below and complete all sections.

See also the accompanying IS AE2 sheet on Moodle for submission details etc.

[1]:
```
# You can use pandas to work with the dataset as shown in this notebook.
import pandas as pd
from os import listdir
import numpy as np


# Generate visualisations in matplotlib or seaborn
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

# This assessment also depends on the pylev package.  Ensure you have installed␣
 ↪this before beginning.

# %pip install pylev
import pylev
print("Everything imported OK")
```

```
Everything imported OK
```

2 Keystroke Dataset

The dataset you will use for this assessment is a subset of the 136 million keystrokes analysed in the paper. Each row in the dataset represents a single keystroke (e.g. pressing "a", or a modifier key like "SHIFT"), and contains information such as key press time, key release time (both in milliseconds), the sentence prompted, the actual input, and the key code.

Each user's data is stored in a separate txt file. Load in the data and familiarise yourself with its structure.

```
[2]: # load in the dataset provided at https://dl.acm.org/citation.cfm?id=3174220

     # Shown here how to load into a single data frame

     user_frame = []

     for file in listdir('KeyStrokes-LabStudies/'):
         if (file[-3:]) == 'txt':
             user_frame.append(pd.read_csv('KeyStrokes-LabStudies/' + file, sep='\t'))

     data_frame = pd.concat(user_frame, axis=0, ignore_index=True)
```

```
[3]: # if in doubt, always check that the data appears as you expect

     data_frame.head()
```

```
[3]:    PARTICIPANT_ID  TEST_SECTION_ID  \
     0          145007          1577476
     1          145007          1577476
     2          145007          1577476
     3          145007          1577476
     4          145007          1577476

                                            SENTENCE  \
     0  I will be out on Friday, but any other day is ...
     1  I will be out on Friday, but any other day is ...
     2  I will be out on Friday, but any other day is ...
     3  I will be out on Friday, but any other day is ...
     4  I will be out on Friday, but any other day is ...

                                          USER_INPUT  KEYSTROKE_ID  \
     0  I will be out on Frida, but any other day is f...      74993989
     1  I will be out on Frida, but any other day is f...      74993987
     2  I will be out on Frida, but any other day is f...      74993991
     3  I will be out on Frida, but any other day is f...      74993993
     4  I will be out on Frida, but any other day is f...      74993995

           PRESS_TIME   RELEASE_TIME LETTER  KEYCODE
     0  1473871859057  1473871859304  SHIFT       16
     1  1473871859201  1473871859297      I       73
     2  1473871859312  1473871859408              32
     3  1473871859424  1473871859520      w       87
     4  1473871859504  1473871859568      i       73
```

```
[4]: # setup a new data structure to hold each of the metrics you have calculated

     # All metrics should be calculated on a per-sentence basis.  Store the metrics␣
      ↪you calculate.

     metric_frame = data_frame[['PARTICIPANT_ID', 'TEST_SECTION_ID']].
      ↪drop_duplicates().reset_index(drop=True)

     metric_frame.head()
```

```
[4]:    PARTICIPANT_ID  TEST_SECTION_ID
     0          145007          1577476
     1          145007          1577488
     2          145007          1577500
     3          145007          1577508
     4          145007          1577517
```

3 1. Performance Metrics (10 marks)

Keystroke studies have a number of standard performance metrics. For this assessment, you must calculate each of the metrics below and generate an accompanying diagram/visualisation for each. All metrics should be calculated for each test section (on a sentence by sentence basis) but visualisations can be produced using any grouping of the data.

Marks will be given in each case for finding the correct result and any well-formed visualisation (correctly labelled axes etc) that illustrates some aspect of the data

For each metric below, you can use the following pattern to implement your solution. * Define a function that: * takes in a data frame holding data for a single sentence (i.e. all data for unique TEST_SECTION_ID), where each row of the data frame is a keystroke for that sentence * returns the metric for that sentence * Call the add_column function below to add the metric to the metric_frame as a new column. add_column takes in a data frame (to which we're adding a column), your new function to calculate the metric, and a name for the new column.

```
[5]: def add_column(frame, fn, name):
         new_col = data_frame.groupby(['TEST_SECTION_ID', 'PARTICIPANT_ID']).
      ↪apply(fn).reset_index(name=name)
         return frame.merge(new_col)
```

To illustrate the format, the first metric below is completed for you

3.1 a) Uncorrected Error Rate

UER is the Levenshtein distance (use the pylev package to calculate this) from the correct string to the string entered by the user, divided by the number of characters in whichever is larger of the correct or entered string.

Calculate the Levenshtein distance for each test section (sentence), print the average value across all users, and produce a visualisation of this data.

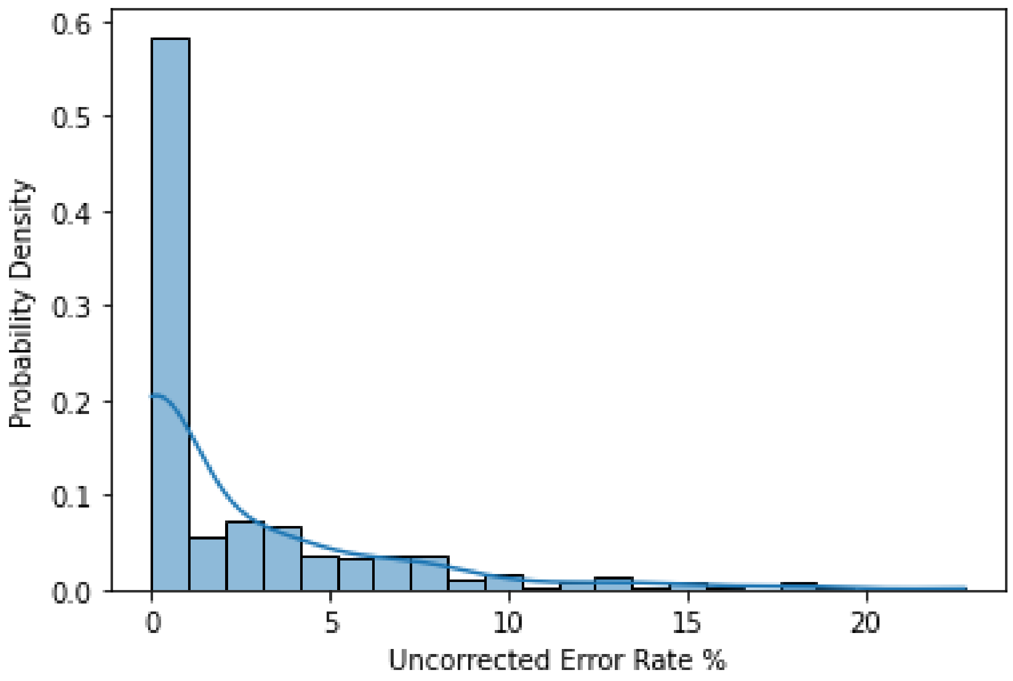
Sample solution provided

– You can use the format of this sample answer to structure your solutions for the remaining metrics.

[6]:
```python
def uncorrErrRate(test_section):
    dist = pylev.levenshtein(test_section['SENTENCE'].iloc[0],␣
 ↪test_section['USER_INPUT'].iloc[0])
    maxChars = max(len(test_section['SENTENCE'].iloc[0]),␣
 ↪len(test_section['USER_INPUT'].iloc[0]))
    return 100.0*float(dist)/float(maxChars)

metric_frame = add_column(metric_frame, uncorrErrRate, 'UER')

uer_mean = metric_frame['UER'].mean()
print(f"Average Uncorrected Error Rate: {uer_mean:.6f}%")
```

```
Average Uncorrected Error Rate: 2.394441%
```

[7]:
```python
#show probability
ax = sns.histplot(data=metric_frame['UER'], kde="true", stat="probability")
ax.set(xlabel="Uncorrected Error Rate %", ylabel="Probability Density");
```



3.2 b) Words Per Minute

WPM is calculated by taking the word count of the typed string divided by the time in minutes from the first key press to the last key press.

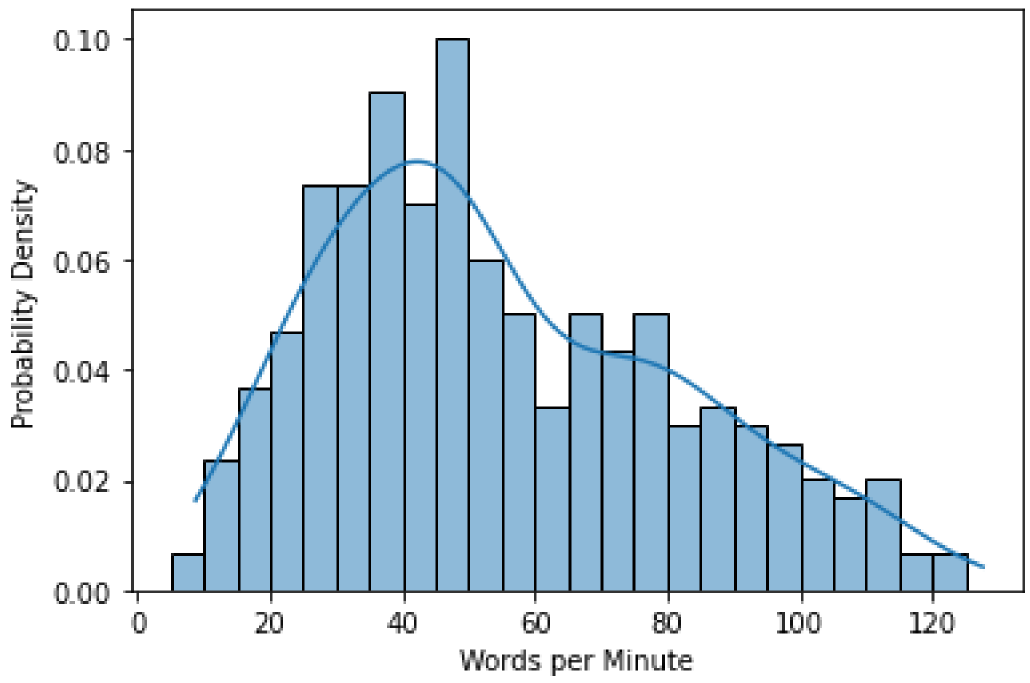
NOTE the definition of 'word' in the paper for the purposes of these calculations: a word is considered as any 5 characters of text the user submits (including spaces, punctuation etc.). Note also that this is considering the string the user typed - not the target string.

Calculate the WPM for each test section, print the average value across all users, and produce a visualisation of this data.

2 marks

[8]:
```python
# Words per minute
def wpm(test_section):
    word_len = 5 # as per paper
    words = len(test_section['USER_INPUT'].iloc[0]) / word_len
    # Using last press time as per question, but I would argue last release time␣
 ↪would make more sense for complete time
    minutes = (test_section['PRESS_TIME'].iloc[-1] - test_section['PRESS_TIME'].
 ↪iloc[0]) / 60000
    return float(words) / float(minutes)


metric_frame = add_column(metric_frame, wpm, 'WPM')
wpm_mean = metric_frame['WPM'].mean()
print(f"Average Words Per Minute: {wpm_mean:.6f}")

ax = sns.histplot(data=metric_frame['WPM'], kde="true", stat="probability", bins␣
 ↪= np.arange(5, 130, 5))
ax.set(xlabel="Words per Minute", ylabel="Probability Density");
```

```
Average Words Per Minute: 55.145475
```



3.3 c) Error Corrections

We count error correcting as pressing a key associated with deleting a character (remember there are usually 2 different keys to do this on most keyboards). EC is calculated as the percentage of keypresses that are the backspace ***or*** delete key.

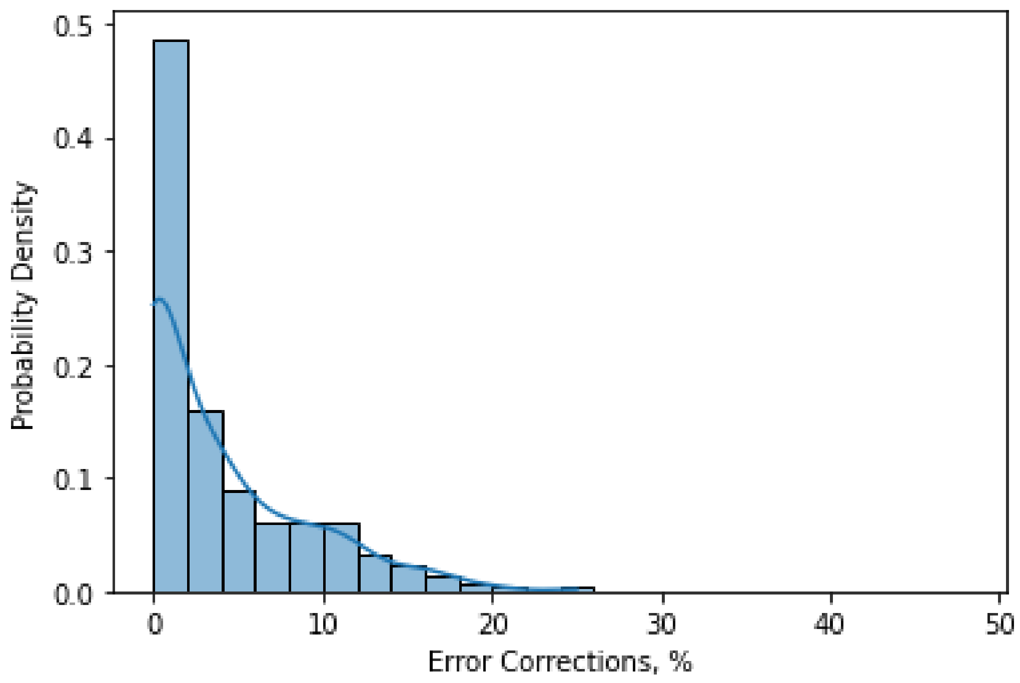
Calculate the EC for each test section, print the average across all users, and produce a visualisation of this data.

2 marks

[9]:
```
# Error corrections
def error_corrections(test_section):
    count = 0
    i = 0
    for key in test_section['LETTER']:
        if (key == 'DELETE' or key == 'BKSP'):
            count += 1
        i += 1
    total_keys = len(test_section['KEYCODE'])
    return 100 * float(count) / float(total_keys)

metric_frame = add_column(metric_frame, error_corrections, 'EC')
ec_mean = metric_frame['EC'].mean()
print(f"Average Error Corrections: {ec_mean:.6f}%")
```

```
ax = sns.histplot(data=metric_frame['EC'], kde="true", stat="probability", bins␣
↪= np.arange(0, 50, 2))
ax.set(xlabel="Error Corrections, %", ylabel="Probability Density");
```

```
Average Error Corrections: 3.874127%
```



3.4 d) Keystrokes Per Character

KPC is the number of total keystrokes divided by the number of characters in the final sentence the user produced (like WPM, note that this refers to the typed not target string).

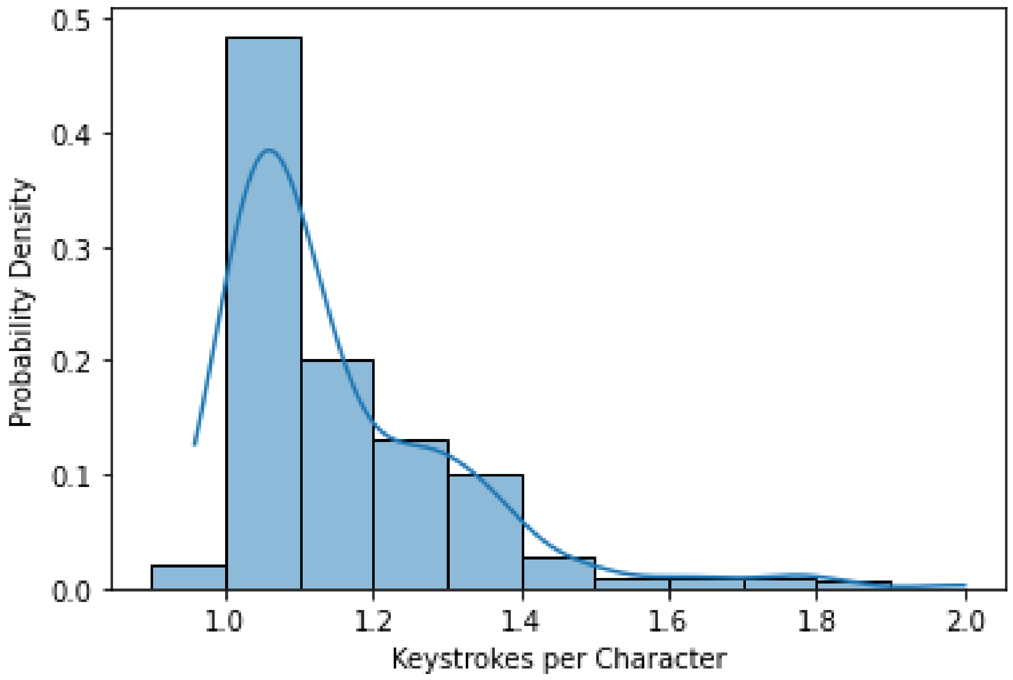
Calculate the KPC for each test section, print the average across all users, and produce a visualisation of this data.

2 marks

[10]:
```
# Keystrokes per character
def kpc(test_section):
    return test_section['LETTER'].size / len(test_section['USER_INPUT'].iloc[0])


metric_frame = add_column(metric_frame, kpc, 'KPC')
kpc_mean = metric_frame['KPC'].mean()
print(f"Average Keystrokes per Character: {kpc_mean:.6f}")
```

```
ax = sns.histplot(data=metric_frame['KPC'], kde="true", stat="probability", bins␣
↪= np.arange(0.9, 2, 0.1))
ax.set(xlabel="Keystrokes per Character", ylabel="Probability Density");
```

```
Average Keystrokes per Character: 1.154218
```



3.5 e) Interkey Interval

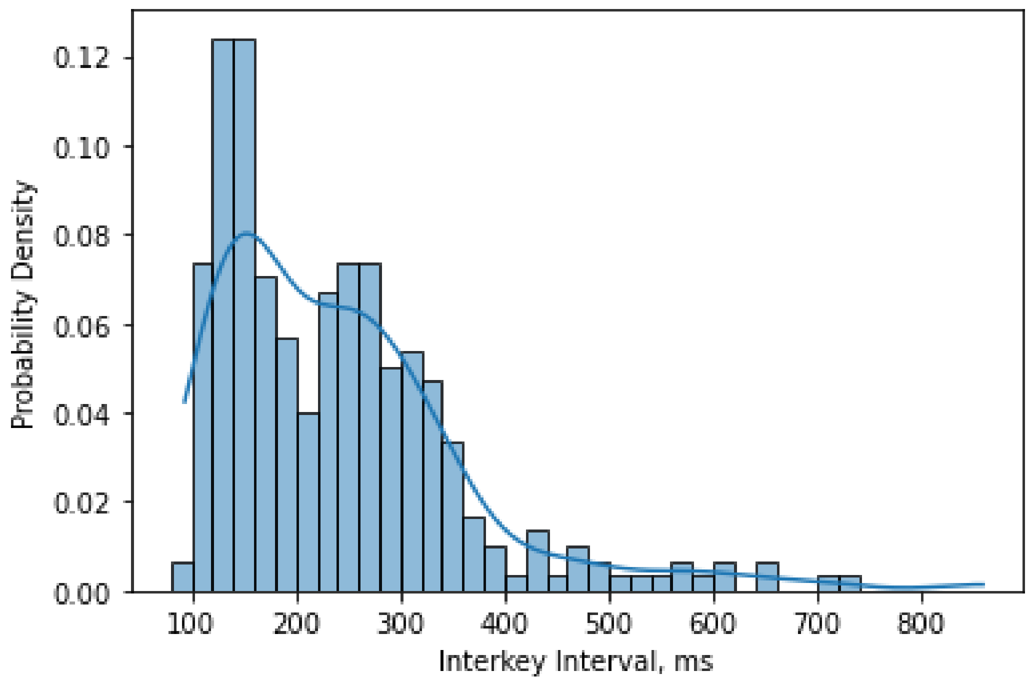
IKI is the time difference in milliseconds between two keypress events. Remove IKI intervals of greater than 5000 milliseconds from your analysis.

Calculate the average IKI for each test section, print the average across all users, and produce a visualisation of this data.

2 marks

```
[11]: # Interkey Interval
def iki(test_section):
    total_time = 0.0
    events = 0
    for i in range(1, test_section['LETTER'].size):
        time = test_section['PRESS_TIME'].iloc[i] - test_section['PRESS_TIME'].
↪iloc[i-1]
        if (time < 5000):
            total_time += time
            events += 1
    return total_time / events

metric_frame = add_column(metric_frame, iki, 'IKI')
iki_mean = metric_frame['IKI'].mean()
print(f"Average Interkey Interval: {iki_mean:.6f} ms")

ax = sns.histplot(data=metric_frame['IKI'], kde="true", stat="probability", bins␣
 ↪= np.arange(80, 860, 20))
ax.set(xlabel="Interkey Interval, ms", ylabel="Probability Density");
```

```
Average Interkey Interval: 242.552694 ms
```



3.6 f) Keypress Duration

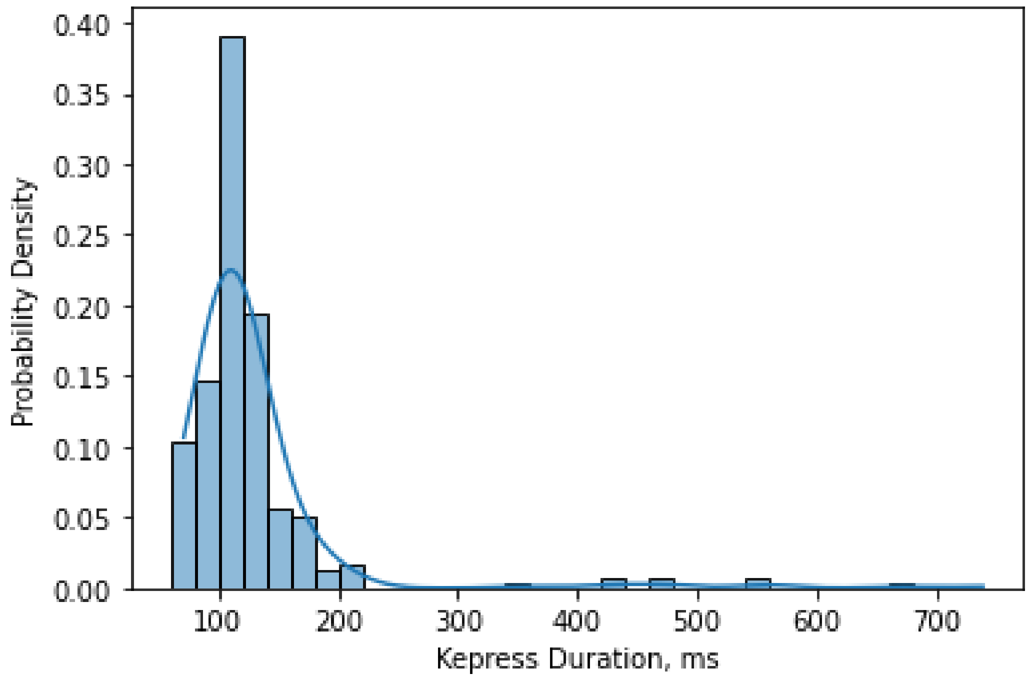
KPD is the duration in milliseconds between a keypress down and keypress up event.

Calculate the average KPD for each test section, print the average across all users, and produce a visualisation of this data.

2 marks

```
[12]: # Keypress Duration
def kpd(test_section):
    total_time = 0.0
    n = test_section['LETTER'].size
    for i in range(n):
        total_time += test_section['RELEASE_TIME'].iloc[i] -␣
 ↪test_section['PRESS_TIME'].iloc[i]
    return total_time / n

metric_frame = add_column(metric_frame, kpd, 'KPD')
kpd_mean = metric_frame['KPD'].mean()
print(f"Average Keypress Duration: {kpd_mean:.6f} ms")

ax = sns.histplot(data=metric_frame['KPD'], kde="true", stat="probability", bins␣
 ↪= np.arange(60, 740, 20))
ax.set(xlabel="Kepress Duration, ms", ylabel="Probability Density");
```

```
Average Keypress Duration: 127.675477 ms
```



4 2. Key Grouping Analysis (10 marks)

There are certain key combinations and orderings in typing behaviour that can be seen in a dataset like this. In the paper, the authors describe some of these results such as lettering pairings "as" "an" and "ll" which require typing across different areas of a QWERTY keyboard.

Using your knowledge of keyboard layouts and typing behaviour, complete a novel analysis of a key grouping of your choice. You may need to be creative in exploring the data to find and analyse a typing behaviour.

Use the space below to describe what key grouping you are analysing and why (up to 200 words).

Generate suitable visualisations to illustrate your point and give a summary of your results (up to 200 words). Comment on whether your results complement or contradict any other findings or discussions in the paper or are investigating something new, and/or general implications of your findings (up to 200 words).

10 marks

4.1 Part 1: Reasoning

The key groupings chosen to analyse are "SHIFT+i" and "CAPS_LOCK, i" as an experiment to measure words per minute of participants using each method of capitalising the letter *i*.

Firstly, the letter *i* is chosen because the capitalised form of it can often be found as a word in the middle of sentences; thus, capital *I* is found much more often than other capital letters.

Secondly, these key groupings are chosen because of the following hypothesis: ***"Participants using CAPS_LOCK to capitalise letters are slower typists than participants using SHIFT"*** and its null hypothesis (There is no difference between participants using CAPS_LOCK to capitalise letter and those using SHIFT) and the following reasons: * CAPS_LOCK capitalisation method requires 2 key presses to capitalise and stop capitalising, while SHIFT requires just 1 (albeit with more control to time press and release times more precisely), which might affect IKI and WPM; * SHIFT and CAPS_LOCK are almost equi-distant to the key *i*, reducing this potential confounding variable; * the differences in WPM and/or IKI might be increased by other confounding factors characterising these two types of people, e.g., experience, frequency of use, etc, which is an interesting potential aspect to analyse out of the scope of this assessed exercise.

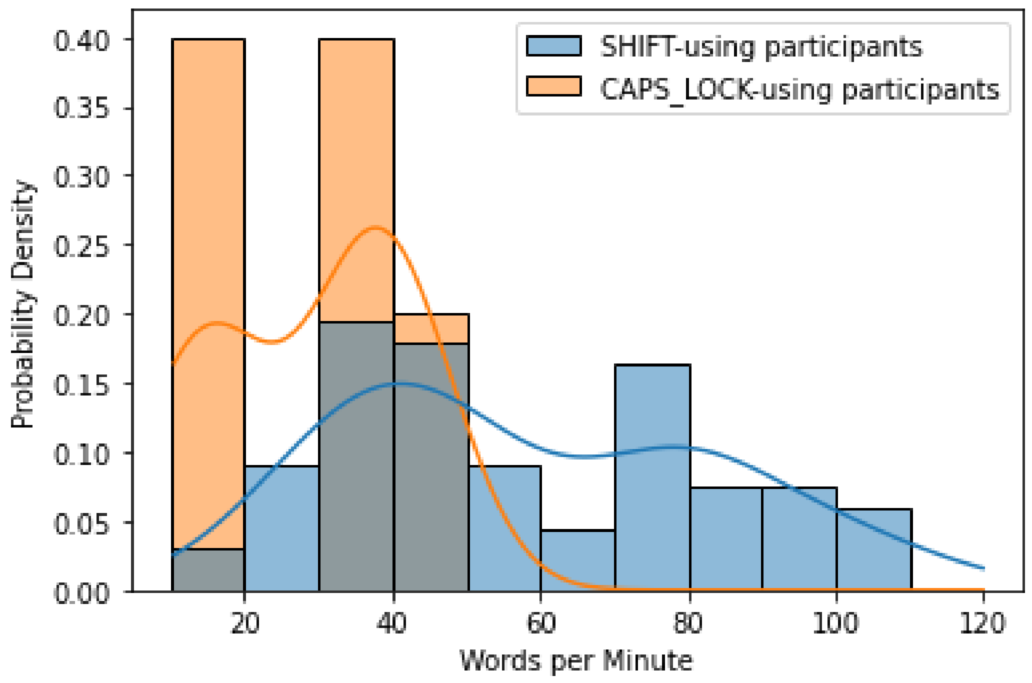
4.2 Part 2: Results

4.2.1 Code and raw data:

[13]:
```
unique_sentences = []

shift_participants = []
shift_wpm = []
shift_kpc = []

caps_participants = []
caps_wpm = []
caps_kpc = []

idgaf_participants = []

i = 0
for sentence in data_frame['SENTENCE']:
    if (sentence not in unique_sentences):
        unique_sentences.append(sentence)
        if ('I' in sentence):
            test_section = data_frame['TEST_SECTION_ID'].iloc[i]
            participant = data_frame['PARTICIPANT_ID'].iloc[i]
            wpm = metric_frame[(metric_frame['PARTICIPANT_ID'] == participant) &
```

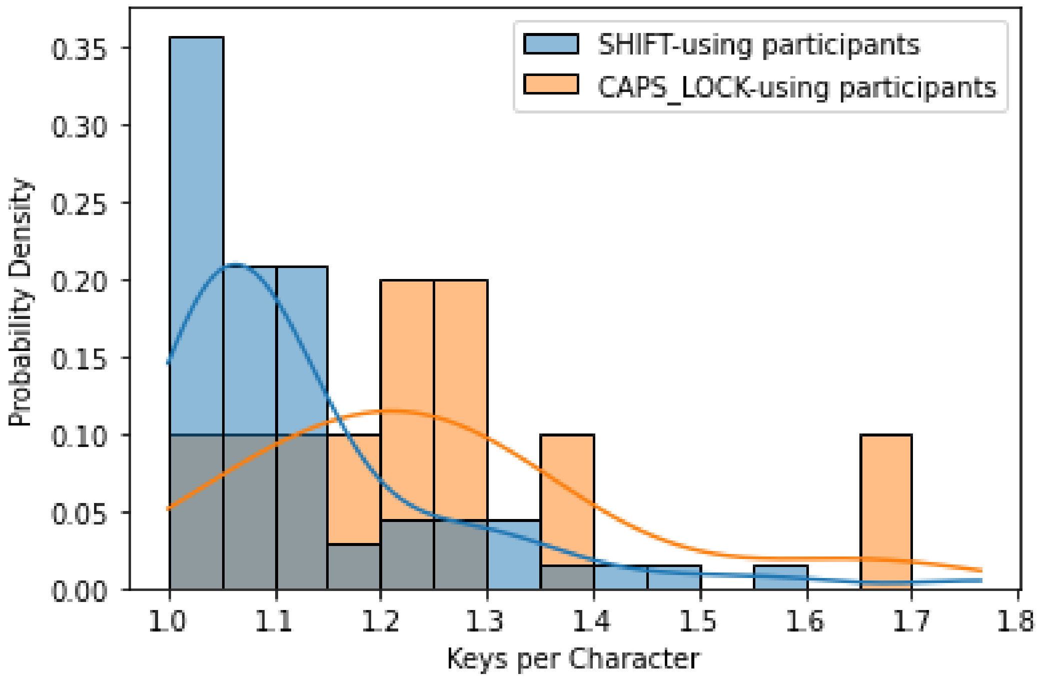
```
                                (metric_frame['TEST_SECTION_ID'] ==␣
↪test_section)]['WPM'].tolist()[0]

            kpc = metric_frame[(metric_frame['PARTICIPANT_ID'] == participant) &
                                (metric_frame['TEST_SECTION_ID'] ==␣
↪test_section)]['KPC'].tolist()[0]

            keys = data_frame[data_frame.TEST_SECTION_ID ==␣
↪test_section]['LETTER']
            if (len(keys[keys == "SHIFT"]) > 0):
                shift_participants.append(participant)
                shift_wpm.append(wpm)
                shift_kpc.append(kpc)

            elif (len(keys[keys == "CAPS_LOCK"]) > 0):
                caps_participants.append(participant)
                caps_wpm.append(wpm)
                caps_kpc.append(kpc)

            else:
                idgaf_participants.append(data_frame['PARTICIPANT_ID'].iloc[i])
    i += 1
print("SHIFT min, max:", np.min(shift_wpm), np.max(shift_wpm))
print("CAPS_LOCK min, max:",np.min(caps_wpm), np.max(caps_wpm))

print("SHIFT median:", np.median(shift_wpm))
print("CAPS_LOCK median:", np.median(caps_wpm))

print("SHIFT standard deviation: " + str(np.std(shift_wpm) / np.mean(shift_wpm)␣
↪* 100) + "%")
print("CAPS_LOCK standard deviation: " + str(np.std(caps_wpm) / np.
↪mean(caps_wpm) * 100) + "%")

wpm_plot = sns.histplot(data={"SHIFT-using participants": shift_wpm,␣
↪"CAPS_LOCK-using participants": caps_wpm},
                  kde="true", stat="probability", common_norm=False, bins = np.
↪arange(10, 120, 10))
wpm_plot.set(xlabel="Words per Minute", ylabel="Probability Density");
```

```
SHIFT min, max: 17.120437522292235 119.88398324202385
CAPS_LOCK min, max: 10.48165710007487 45.409674234945705
SHIFT median: 52.96319377981556
CAPS_LOCK median: 34.558157528445946
SHIFT standard deviation: 43.03982225622428%
CAPS_LOCK standard deviation: 42.39245259930327%
```



[14]:
```
print("SHIFT min, max:", np.min(shift_kpc), np.max(shift_kpc))
print("CAPS_LOCK min, max:", np.min(caps_kpc), np.max(caps_kpc))

print("SHIFT median:", np.median(shift_kpc))
print("CAPS_LOCK median:", np.median(caps_kpc))

print("SHIFT standard deviation: " + str(np.std(shift_kpc) / np.mean(shift_kpc)␣
 ↪* 100) + "%")
print("CAPS_LOC standard deviation: " + str(np.std(caps_kpc) / np.mean(caps_kpc)␣
 ↪* 100) + "%")

iki_plot = sns.histplot(data={"SHIFT-using participants": shift_kpc,␣
 ↪"CAPS_LOCK-using participants": caps_kpc},
                  kde="true", stat="probability", common_norm=False, bins=np.
 ↪arange(1, 1.8, 0.05))
iki_plot.set(xlabel="Keys per Character", ylabel="Probability Density");
```

```
SHIFT min, max: 1.0 1.7647058823529411
CAPS_LOCK min, max: 1.0434782608695652 1.6578947368421053
SHIFT median: 1.0816666666666666
CAPS_LOCK median: 1.2267080745341614
SHIFT standard deviation: 12.605438027533863%
CAPS_LOC standard deviation: 13.436811988785188%
```



4.2.2 Results

There is a clear difference between the SHIFT group and the CAPS_LOCK group, both in their Words per Minute and their Keys per Character metrics, which gives the ability to reject the null hypothesis. Additionally, analysing the graphs and metrics gives some insight into the hypothesis itself.

By looking at the Keys per Character aspect, the median for the CAPS_LOCK group is around 1.227 (STD: 12.6%), while for the SHIFT group - around 1.082 (STD: 13.4%), which is 0.145 less. While the standard deviations are pretty high, there seems to be an obvious correlation between the capitalisation method used and the Keys per Character metric, as described in Part 1 (SHIFT requiring 1 press, while CAPS_LOCK requiring 2)

By looking at the Words per Minute aspect, the median for the CAPS_LOCK group is around 34.56 (STD: 42%), while for the SHIFT group - around 52.96 (STD: 43%), which is 18.40 words higher. While the STDs are crazy high, a relatively lower WPM metric for the CAPS_LOCK group makes sense, if at least based from the additional keys to press for each capital *I*. The difference could also be explained by confounding variables, e.g., trained or untrained typists, etc.

4.3 Part 3: Discussion

Some of the results have direct correspondence with findings in the paper, e.g., the median KPC of CAPS_LOCK-users (1.227) is very close to the KPC of slow typists reported in the paper (1.24), both of which are far from the average KPC of all typists (1.17), which makes sense because of the additional CAPS_LOCK key required for each capitalised section. Additionally, both "SHIFT+i" and “CAPS_LOCK, i” are a very similar distance to the “ai” bigram, which the paper categorises as a hand-alternation one.

Furthermore, the median WPM of SHIFT-users (52.96) is close to the WPM of all typists in the paper (51.56), which means that SHIFT-users, the vast majority of all typists in the data, are a good representation of the average typist’s WPM.

However, analysing special keys with letter keys is rather different from any analysis in the paper. This difference in capitalisation methods is rather striking, not just in WPM and KPC, but also in the number of key sequences in the data using each method (SHIFT in 68 sequences, CAPS_LOCK in 10). Thus, it seems that typists could improve their WPM by just changing the method with which they capitalise letters. What it will definitely improve, however, is their KPC.