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 (https://dl.acm.org/citation.cfm?id=3174220)

Please follow on below and complete all sections.

In [1]:
```
# You should 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 ing matplotlib or seaborn
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

# This asessment also depends on the pylev package.  Ensure you have installe

# to install pylev, run the command below by uncommenting it and running the
#!pip install --user git+https://github.com/toastdriven/pylev

import pylev
```

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 keystroke, and information such as key press time, key release time (both in milliseconds), the sentence prompted, the actual input, and the key code is represented.

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

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

user_frame = []

for file in listdir('KeyStrokes-LabStudies/'):

    if (file[-3:]) == 'txt':
        user_frame.append(pd.read_csv('KeyStrokes-LabStudies/' + file, sep='\

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

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

data_frame.head()
```

Out[3]:

	PARTICIPANT_ID	TEST_SECTION_ID	SENTENCE	USER_INPUT	KEYSTROKE_ID	PRESS_TIME	R
0	145007	1577476	I will be out on Friday, but any other day is ...	I will be out on Frida, but any other day is f...	74993989	1473871859057	
1	145007	1577476	I will be out on Friday, but any other day is ...	I will be out on Frida, but any other day is f...	74993987	1473871859201	
2	145007	1577476	I will be out on Friday, but any other day is ...	I will be out on Frida, but any other day is f...	74993991	1473871859312	
3	145007	1577476	I will be out on Friday, but any other day is ...	I will be out on Frida, but any other day is f...	74993993	1473871859424	
4	145007	1577476	I will be out on Friday, but any other day is ...	I will be out on Frida, but any other day is f...	74993995	1473871859504	

In [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.  Use a data frame

metric_frame = data_frame[['PARTICIPANT_ID', 'TEST_SECTION_ID']].drop_duplica

metric_frame.head()
```

Out[4]:

	PARTICIPANT_ID	TEST_SECTION_ID
0	145007	1577476
1	145007	1577488
2	145007	1577500
3	145007	1577508
4	145007	1577517

Performance Metrics

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.

For each metric, use the following pattern to implement your solution:

Define a function that takes in a data frame that holds a row for each keystroke for a given sentence and returns the metric for that sentence.

Use 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 you will add a column with a new metric), a function that calculates the metric, and a name for the new column.

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

Uncorrected Error Rate

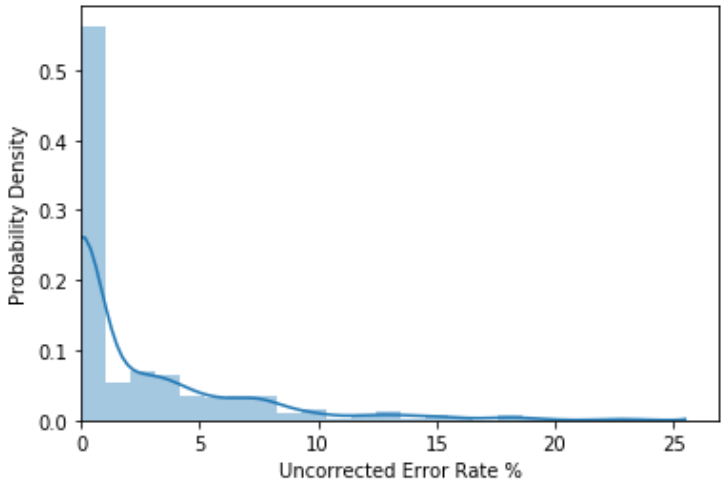
UER is the Levenshtein distance (use the pylev package to calculate this) from the correct string to the string entered by the user devided by the number of characters in the larger of the correct or the entered string.

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

Use this sample solution to structure your solutions for the remaining metrics.

In [6]:
```
def uer(sentence):
    distance = float(pylev.levenshtein(sentence['USER_INPUT'].iloc[0], senten
    score = float(max(len(sentence['USER_INPUT'].iloc[0]), len(sentence['SENT
    return (float(distance/score)*100.0)

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

# plt.hist(metric_frame['UER'])
# plt.xlabel("Uncorrected Error Rate %")
# plt.ylabel("Test Section (N)")

ax = sns.distplot(metric_frame['UER'])
ax.set(xlabel="Uncorrected Error Rate %", ylabel="Probability Density", xlim=
```

Out[6]:
```
[Text(0, 0.5, 'Probability Density'),
 (0, 26.894247672509824),
 Text(0.5, 0, 'Uncorrected Error Rate %')]
```



Words Per Minute

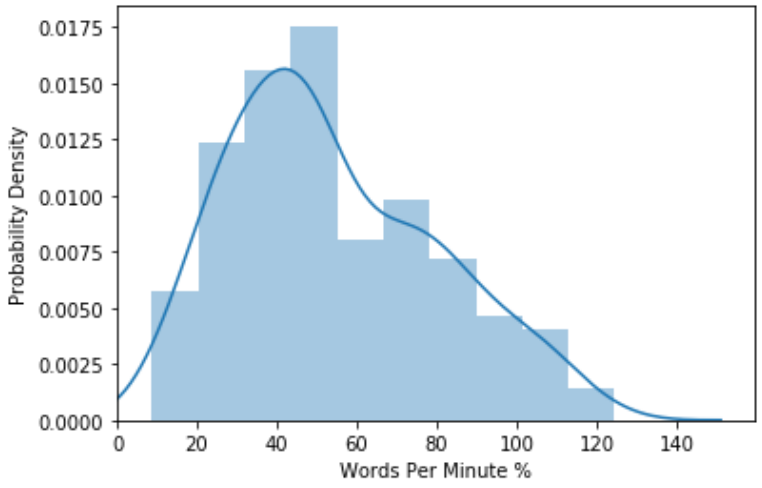
WPM is calculated by taking the word length of the string transcribed (where length is the number of characters in the string divided by five) divided by the time (in minutes) from the first key press to the last key press.

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

In [7]:
```
# Words Per Minute
def WPM(sentence):
    sLength = float(len(sentence['USER_INPUT'].iloc[0]) / 5)
    timing = (sentence['RELEASE_TIME'].iloc[-1]) - (sentence['PRESS_TIME'].il

    return(sLength/timing) *60000

metric_frame = add_column(metric_frame, WPM, 'WPM')

print("Average WPM of all Participants: " + str(metric_frame['WPM'].mean()))

ax = sns.distplot(metric_frame['WPM'])
ax.set(xlabel="Words Per Minute %", ylabel="Probability Density", xlim=(0,Non
```

```
Average WPM of all Participants: 54.3075056271782
```

Out[7]:
```
[Text(0, 0.5, 'Probability Density'),
 (0, 159.5516629962213),
 Text(0.5, 0, 'Words Per Minute %')]
```



Error Corrections

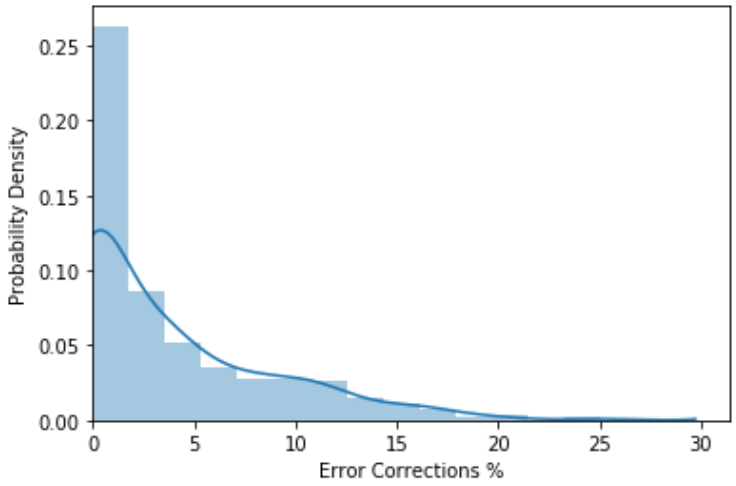
EC is calculated as the percentage of keypresses that are the backspace or delete key.

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

In [8]:
```
# Error Corrections
def eCorrections(sentence):
    tCorrections =0
    tKeypress = len(sentence)

    for input in sentence['LETTER']:

        if (input == 'DELETE' or input == 'BKSP'):
            tCorrections +=1

    return(tCorrections/tKeypress) *100

metric_frame = add_column(metric_frame, eCorrections, 'Error Corrections')

tUsers = 0
tError = 0

for user in metric_frame['Error Corrections']:
    tUsers +=1
    tError +=user

print("Average Error Corrections of all Participants: " + str(metric_frame['E

ax = sns.distplot(metric_frame['Error Corrections'])
ax.set(xlabel="Error Corrections %", ylabel="Probability Density", xlim=(0,No
```

```
Average Error Corrections of all Participants: 3.874126643012982
```

Out[8]:
```
[Text(0, 0.5, 'Probability Density'),
 (0, 31.451848165648883),
 Text(0.5, 0, 'Error Corrections %')]
```



Keystrokes Per Character

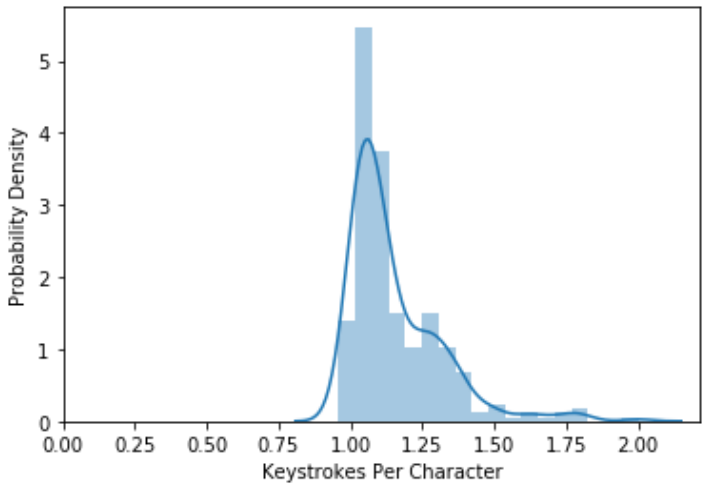
KPC is the number of total keystrokes divided by the number of characters in the final sentence.

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

In [9]:
```
# Keystrokes per character
def KpC(sentence):
    tKeystrokes =0
    tInput = len(sentence['USER_INPUT'].iloc[-1])

    for input in sentence['LETTER']:
        tKeystrokes +=1

    return tKeystrokes / tInput

metric_frame = add_column(metric_frame, KpC, 'KPC')

print("Average Keystrokes Per Character of all Participants: " + str(metric_f

ax = sns.distplot(metric_frame['KPC'])
ax.set(xlabel="Keystrokes Per Character", ylabel="Probability Density", xlim=
```

```
Average Keystrokes Per Character of all Participants: 1.1542178248152415
```

Out[9]:
```
[Text(0, 0.5, 'Probability Density'),
 (0, 2.2165886965657213),
 Text(0.5, 0, 'Keystrokes Per Character')]
```



Interkey Interval

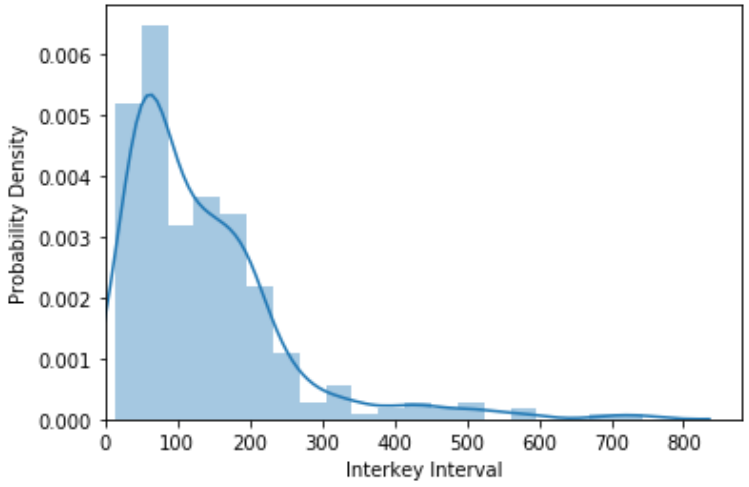
IKI is the time difference in milliseconds between two keypress events. Remove IKI intervals of greater than 5000 milliseconds from your analysis.

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

In [10]:
```
# Interkey Interval
#taken to be the time when no key is pressed down
def IkI(sentence):
    iki =0
    tTests = 0
    for i in range(len(sentence['PRESS_TIME'])-1):
        difference = sentence['PRESS_TIME'].iloc[i+1] - sentence['RELEASE_TIM

        #if there is rollover take interval to be 0
        if difference < 0:
            difference = 0
        #exclude long skewing data
        if difference <= 5000:
            iki +=difference
            tTests +=1

    return iki/tTests

metric_frame = add_column(metric_frame, IkI, 'IKI')

print("Average Interkey Intervals of all Participants: " + str(metric_frame['

ax = sns.distplot(metric_frame['IKI'])
ax.set(xlabel="Interkey Interval", ylabel="Probability Density", xlim=(0,None
```

```
Average Interkey Intervals of all Participants: 136.18346301110566
```

Out[10]:
```
[Text(0, 0.5, 'Probability Density'),
 (0, 881.2925418098722),
 Text(0.5, 0, 'Interkey Interval')]
```



Keypress Duration

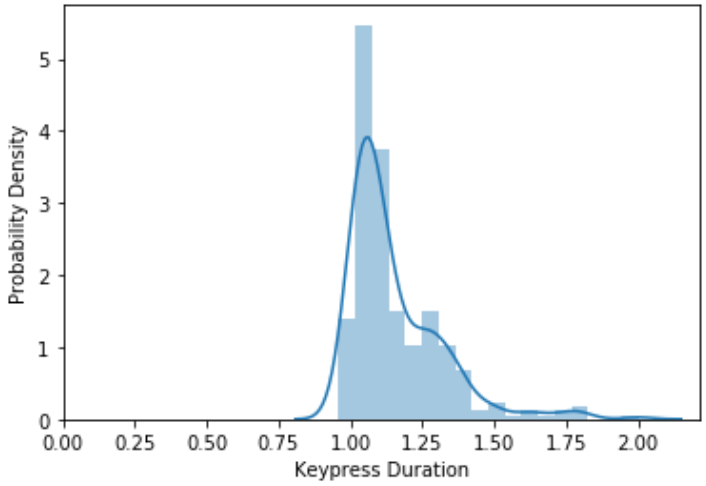
KPD is the duration in milliseconds between a keypress down and keypress up event.

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

In [11]:
```
# Keypress Duration
def KpD(sentence):
    dur =0
    tTests = 0

    for i in range(len(sentence['PRESS_TIME'])):
        dur += sentence['RELEASE_TIME'].iloc[i] - sentence['PRESS_TIME'].iloc
        tTests +=1

    return dur / tTests

metric_frame = add_column(metric_frame, KpD, 'KPD')

tUsers = 0
tKpD = 0

for user in metric_frame['KPD']:
    tUsers +=1
    tKpD +=user

print("Average Keypress Duration of all Participants: " + str(tKpD/tUsers))

ax = sns.distplot(metric_frame['KPC'])
ax.set(xlabel="Keypress Duration", ylabel="Probability Density", xlim=(0,None
```

```
Average Keypress Duration of all Participants: 127.67547747179412
```

Out[11]:
```
[Text(0, 0.5, 'Probability Density'),
 (0, 2.2165886965657213),
 Text(0.5, 0, 'Keypress Duration')]
```



Key Grouping Analysis

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 a why (up to 150 words). Generate suitable visualisations to illustrate your point and give a summary of your results (up to 150 words).

Analysis

The lettering pairings for the analysis of key grouping have been selected as, "oo" , “op” and "qu". Code "oo" was been selected as it is a duplicate letter (therefore on one side of the keyboard), it will generally be conducted by a single finger. Code “op” was picked as it is close together on the keyboard but not a duplicate letter, generally conducted with one hand but with different fingers. Code "qu" was chosen as the letters are distributed physically far apart on the keyboard and will generally be typed using both hands. Codes selected to be used by one hand were chosen at the same end of the keyboard to ensure the same hand would be used. It was found that these codes are used a similar number of times in the dataset which gives a further reason they should be chosen; "oo"20- occurrences, "op"-11 occurrences, "oo"-8 occurrences.

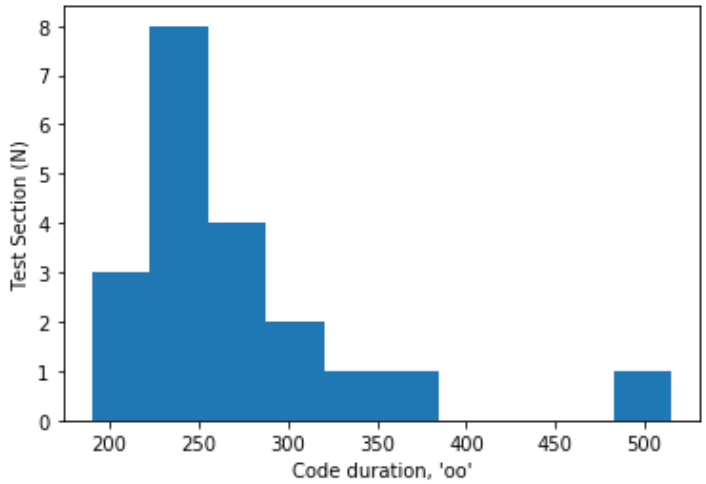
The analysis that was carried out was the "Code Press Duration", given as the time from the first letter of code being pressed to the time the second letter of the code being released. The times accountable were limited to 600 milliseconds as skewed results were present otherwise. The average of these results was then taken, and a histogram was plotted. Average "Code Press Durations" in milliseconds as follows; "oo" = 269.45, "op" =305.0, “qu” = 233.625. These results show that two keys were pressed one after the other the fastest when the participants would use both of theirs hand, the letters being physically far apart on the keyboard. The results also indicate that when the participants types two consecutive letters with the same hand, the two letters are typed faster if the are the same letter. This is because the same finger would be used in the same location.

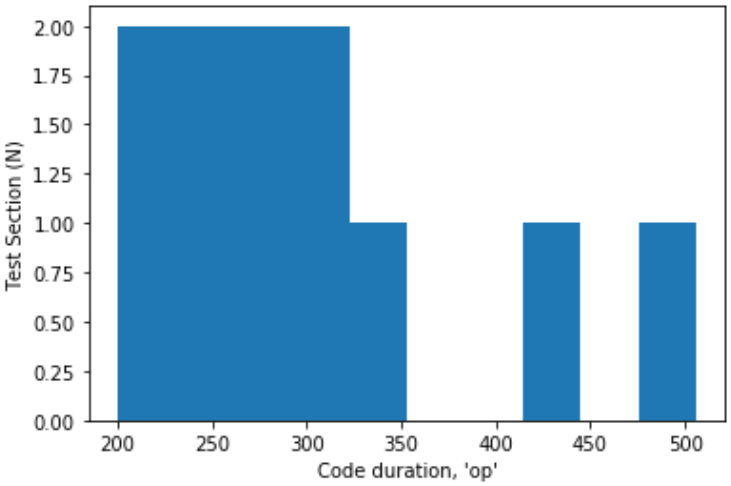
In [12]:
```
# Code press duration of oo
def oo(sentence):
    code_dur =0
    tTests = 0
    difference = 0
    for i in range(len(sentence['PRESS_TIME'])-1):
        if(sentence['LETTER'].iloc[i]+sentence['LETTER'].iloc[i+1]  == "oo"):
            difference = sentence['RELEASE_TIME'].iloc[i+1] - sentence['PRESS

            if difference < 0:
                difference = 0
            if difference <= 600:
                code_dur +=difference
                tTests +=1
    try:
        #print(code_dur/tTests)
        return code_dur/tTests
    except:
        return None

#create frame frame
metric_frame = add_column(metric_frame, oo, 'oo')
#find average
print("Average 'oo' Code Press Duration of all Participants: " + str(metric_f
#plot on histogram
plt.figure()
plt.hist(metric_frame['oo'].dropna())
#label axis
plt.xlabel("Code duration, 'oo'")
plt.ylabel("Test Section (N)")


# Code press duration of op
def op(sentence):
    code_dur =0
    tTests = 0
    difference = 0
    for i in range(len(sentence['PRESS_TIME'])-1):
        if(sentence['LETTER'].iloc[i]+sentence['LETTER'].iloc[i+1]  == "op"):
            difference = sentence['RELEASE_TIME'].iloc[i+1] - sentence['PRESS

            if difference < 0:
                difference = 0
            if difference <= 600:
                code_dur +=difference
                tTests +=1
    try:
        #print(code_dur/tTests)
        return code_dur/tTests
    except:
        return None

#create frame frame
metric_frame = add_column(metric_frame, op, 'op')
#find average
print("Average 'op' Code Press Duration of all Participants: " + str(metric_f
#plot on histogram
plt.figure()
plt.hist(metric_frame['op'].dropna())
#label axis
plt.xlabel("Code duration, 'op'")
plt.ylabel("Test Section (N)")

# Code press duration of qu
```

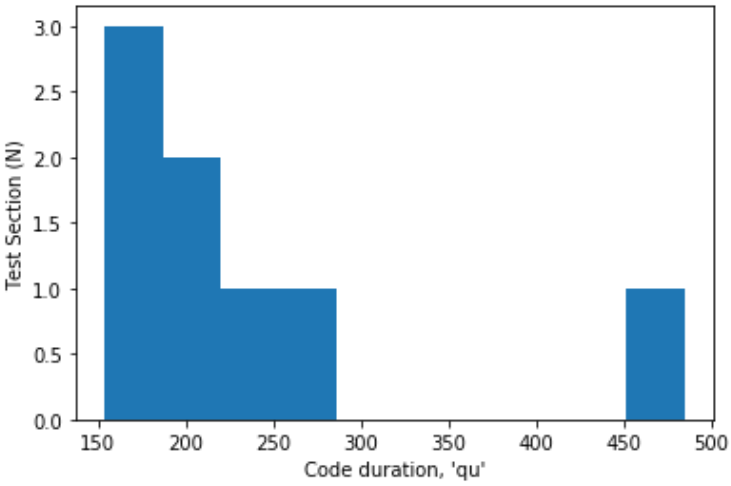
```
def qu(sentence):
    code_dur =0
    tTests = 0
    difference = 0
    for i in range(len(sentence['PRESS_TIME'])-1):
        if(sentence['LETTER'].iloc[i]+sentence['LETTER'].iloc[i+1]  == "qu"):
            difference = sentence['RELEASE_TIME'].iloc[i+1] - sentence['PRESS

            if difference < 0:
                difference = 0
            if difference <= 600:
                code_dur +=difference
                tTests +=1
    try:
        #print(code_dur/tTests)
        return code_dur/tTests
    except:
        return None

#create frame
metric_frame = add_column(metric_frame, qu, 'qu')
#find average
print("Average 'qu' Code Press Duration of all Participants: " + str(metric_f
#plot on histogram
plt.figure()
plt.hist(metric_frame['qu'].dropna())
#label axis
plt.xlabel("Code duration, 'qu'")
plt.ylabel("Test Section (N)")
```

```
Average 'oo' Code Press Duration of all Participants: 269.45
Average 'op' Code Press Duration of all Participants: 305.0
Average 'qu' Code Press Duration of all Participants: 233.625
```

Out[12]: Text(0, 0.5, 'Test Section (N)')







Big Data?

If you are interested to see how you analysis stacks up to the full dataset (16 GB uncompressed) than rerun your analysis with the full dataset available here: https://userinterfaces.aalto.fi/136Mkeystrokes/ (https://userinterfaces.aalto.fi/136Mkeystrokes/)