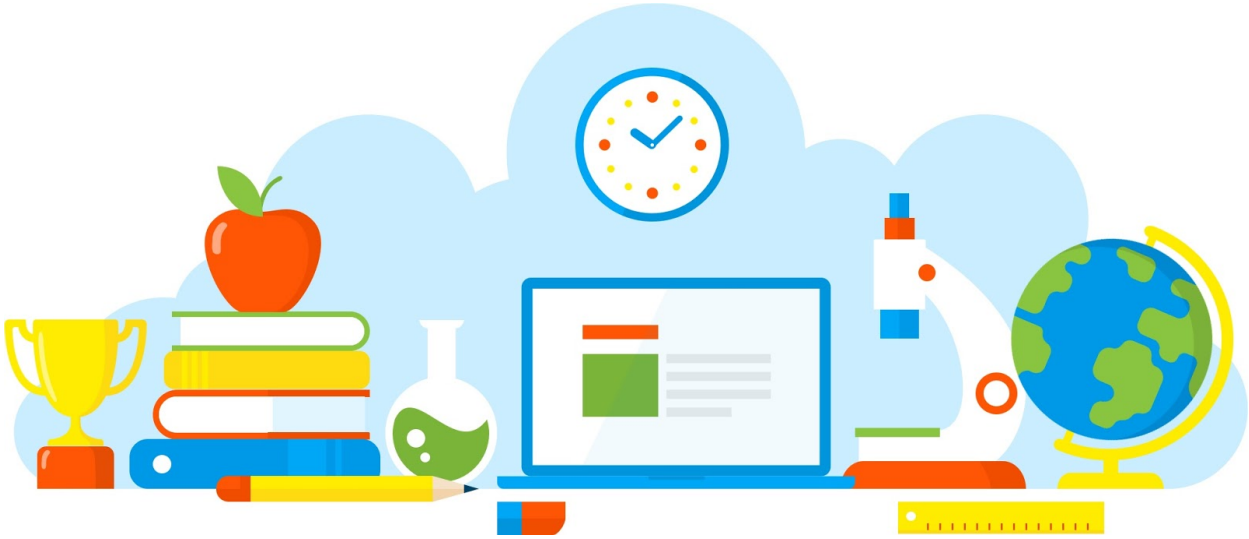


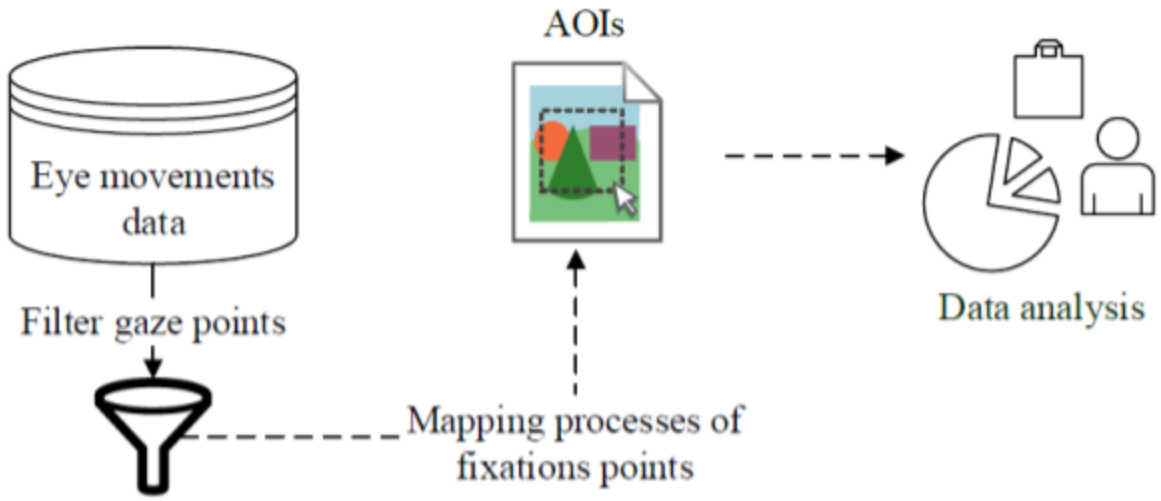
Capstone Project

Intermediate Data Science: Python

Nima Ahmadi
08/02/2019

Introduction

Training interventions are designed to narrow the gap between the performance of novices and experts. This could achieve using various approaches; each of which has its advantages & disadvantages. In the domain of aviation, the process of training is lengthy and expensive. Besides, some flight scenarios (if possible) are dangerous to practice in actual flights. So, the virtual environment is a good option to reduce the cost of training & to improve the transfer of training. Furthermore, pilots scanning pattern is related to piloting skills; the more structured scanning pattern, the better the piloting skills. This makes eye-tracking technology an appropriate tool to assess pilots' performances.



The process of eye movements' data analysis

In the current research, k-means clustering analysis is used to define how much a gaze-based training was effective on novice pilots. There is a reason to apply this method. The current method of assessing pilots' scanning pattern is based on defining Areas of Interests (AOIs), then measuring metrics (such as fixation count and dwell time). This is a time-consuming process. Because, fixation points (of pilots) should be mapped on AOIs. The software does it for a researcher, but the outcome is not as expected. Errors occur in this process. Therefore, the researcher has to manually fix the problem. For one recording, this could take up to 2 hours. If there is only one participant, this is not a problem, but in some studies, more than 50 subjects are needed. So, a method was developed to not map fixation point on AOIs. Instead, raw data and pilots

gaze points were used to assess the effectiveness of training.

What could go wrong in flight?

Pilots have to apply active visual search to keep a high level of Situation Awareness (SA). It is simple, but not easy to apply. Pilots could suffer from divided attention or selective attention. Divided attention is multi-tasking. In this case, pilots distribute their attention to multiple objects in-flight environment. As a result, pilots can not process information. On the other hand, focusing on an object makes pilots blind to maintain situational awareness. The key here is to ignore less important instruments and focus on more valuable information. So, it is expected that the pilots who suffered from tunnel visioning will have less number of clusters than other pilots. The eye movement metric that indicates the scanning pattern of pilots is visual entropy. A high visual entropy shows the preference of pilots to search visual environment. In contrast, a low value relates to narrowing of visual attention to a few number of instruments.

[1]

Experimental Design

Twenty novice pilots were recruited from nearby flight schools. Pilots were divided into two groups: experimental and control groups. Pilots of the experimental group trained on tactical scanning and rest of the pilot received sham treatment. Pilots performed four scenarios, the results of this research is presented for one flight scenario. In the research, flight & eye movement data were collected to assess piloting as well as scanning patterns' of pilots.

Data collection and data wrangling

Eye tracker samples data 50 times per seconds. In other words, the sampling rate of the eye tracker was 50 Hz. It saves the recorded data in .tsv format which has 114 columns.

These columns are provides various information, mainly about project data and eye

[1] https://www.tobiipro.com/product-listing/tobii-pro-glasses-2/

tracking data. Flight data, on the other end, was recorded in .txt format in 21 columns.

The important columns are as follows:

1. **Collected data by eye tracker**

- **Project data** includes name of project, recording data and recording timestamp.
    - Application of project data:
        - Preparing time index dataframe
        - Merging flight dataframe with eye movements dataframe
        - Debugging eye tracking data on eye tracking software (sometimes the software maps fixation points incorrectly and needs to be fixed. I used the collected data to manually correct mapped data)
- **Eye tracking data** including:
    - eye movements types (fixation, saccade, …)
    - events names
    - fixation points
    - fixation duration
    - gaze points (right and left eyes)
    - pupil diameter (right and left eyes)
    - gaze 3D positions x, y, z (right and left eyes)
    - Location of mapped fixation points (x, y, z) (pixel)

2. **Collected data by flight simulator**

- **Flight data** including**:**
    - airspeed
    - altitude
    - vertical air speed
    - heading
    - rate of turn
    - power setting (RPM)
    - inputs to the yoke, rudder, & trim

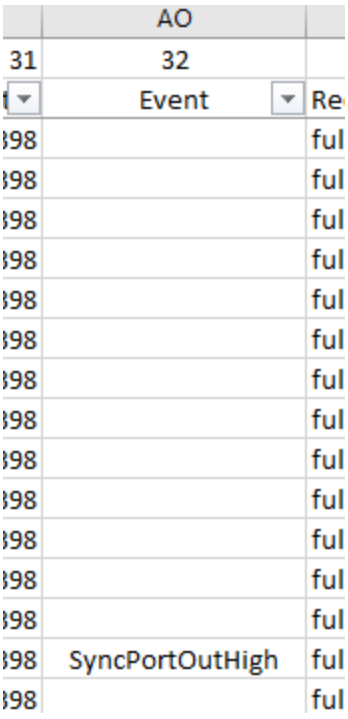
Challenges of Data wrangling

Raw data is not in a form that could be used in analysis. Some steps were taken to make data use able such as generating timestamps for collected data and estimating missing values. In what follows, I provides more details on data wrangling and reasons for doing that:

1. Eye tracker only generate tags for onset and end of custom events; between could be no tags or few unwanted tags. Custom events are used to define scenarios. It indicates, for instance, time that a flight task (such as landing) started and also the time that task was finished. To analyze data and make a master dataframe, the whole period of a given scenario should be tagged. This was challenging as the eye tracking generated others tags and added to the collected data.

A given custom event

A given unwanted events



Code for modification of events' tags

```
list_extra = []
list_extra = df_eye['Event'][df_eye['Event'].notnull()].unique()
list_extra = list_extra.tolist()
list_extra.extend(['X', 'XX'])

list_events = ['Landing', 'Level Turn', 'Level Flight', 'Descent']

for i in list_events:
    if i in list_extra:
        list_extra.remove(i)

for i in list_extra:
    df_eye['Event'] = df_eye['Event'].str.replace(i, '')

df_eye['Event'] = df_eye['Event'].fillna('')
df_eye['Event'] = df_eye['Event'].replace({'': np.NaN})
```

```
for index, row in df_eye.iterrows():
    if  len(df_eye['Event']) == index + 1:
        break

    elif pd.isnull(df_eye['Event'].iloc[index]) == True and pd.isnull(df_eye['Event'].iloc[index+1]) == True:
        flag_event = False

    elif pd.isnull(df_eye['Event'].iloc[index]) == False and pd.isnull(df_eye['Event'].iloc[index+1]) == True and flag_event ==
e:

        if index == 0:
            if pd.isnull(df_eye['Event'].iloc[index]) == False and pd.isnull(df_eye['Event'].iloc[index+1]) == True:
                zero_point = index
        else:

            if pd.isnull(df_eye['Event'].iloc[index]) == False and pd.isnull(df_eye['Event'].iloc[index+1]) == True and
.snull(df_eye['Event'].iloc[index-1]) == True:
                zero_point = index

        df_eye.at[index+1, 'Event'] = df_eye['Event'].iloc[index]

        df_eye.at[index, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index] - df_eye['Recording
stamp'].iloc[zero_point]

        flag_event = False

    elif pd.isnull(df_eye['Event'].iloc[index]) == False and pd.isnull(df_eye['Event'].iloc[index+1]) == False:

        df_eye.at[index, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index]
        df_eye.at[index+1, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index+1]
        df_eye.at[index, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index] - df_eye['Recording
stamp'].iloc[zero_point]
        df_eye.at[index+1, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index+1] - df_eye['Recording
stamp'].iloc[zero_point]

        zero_point = 0
        flag_event = True
```

2. Eye-tracking and flight data is time-series data. So, few columns should be merged to create a timestamp. One point, Timestamps one both data frames needed to have common format (date and time).

Generated data by eye tracker

B	E	F	G	J
Export date	Recording date	Recording start time	Recording duration	Recording timestamp
7/5/2019	7/2/2019	43:05.4	757985.566	0
7/5/2019	7/2/2019	43:05.4	757985.566	0
7/5/2019	7/2/2019	43:05.4	757985.566	20
7/5/2019	7/2/2019	43:05.4	757985.566	40
7/5/2019	7/2/2019	43:05.4	757985.566	60
7/5/2019	7/2/2019	43:05.4	757985.566	80
7/5/2019	7/2/2019	43:05.4	757985.566	100

The result is cleaned as follows:

```
Out[6]:
                              time       timestamp
21164  2018-09-28 15:52:37.583 00:01:41.493000
21165  2018-09-28 15:52:37.584 00:01:41.494000
21166  2018-09-28 15:52:37.590 00:01:41.500000
21167  2018-09-28 15:52:37.591 00:01:41.501000
21168  2018-09-28 15:52:37.595 00:01:41.505000
```

Code to import data, generate timestamp

```
    df_eye = pd.read_csv(filename_eye, encoding = 'utf-16', delimiter='\t', low_memory=False, parse_dates = {'time':['Recording
date','Recording start time']})
    df_eye['Recording timestamp'] = pd.to_timedelta(df_eye['Recording timestamp'], unit ='ms')
    df_eye['time'] = df_eye['time'] + df_eye['Recording timestamp']

    df_eye['scenario duration'] = np.nan
    df_eye['scenario duration'] = pd.to_timedelta(df_eye['scenario duration'])
```

3. there are several missing values which should be estimated using various approaches such as backfill, forward fill, interpolation. Also, nan values were addressed in appropriate ways

Sample of Missed data

	K	L	
	Gaze point X	Gaze point Y	Gaz
	929	372	
	931	365	

Sample of flight data (look at the time)

	timestamp	real, time	...	rpm n, engin	s
2018-09-28 15:50:52.556118		92.596481	...	1495.429321	
2018-09-28 15:50:52.656124		92.690842	...	1495.429321	
2018-09-28 15:50:52.746129		92.783447	...	1495.429321	
2018-09-28 15:50:52.836134		92.875763	...	1495.429321	
2018-09-28 15:50:52.926139		92.967957	...	1495.429321	
2018-09-28 15:50:53.016144		93.060242	...	1495.429321	
2018-09-28 15:50:53.116150		93.154129	...	1495.429321	
2018-09-28 15:50:53.207155		93.247414	...	1495.429321	

Sample of eye tracking data (look at the time)

Out[6]:

	time	timestamp
21164	2018-09-28 15:52:37.583	00:01:41.493000
21165	2018-09-28 15:52:37.584	00:01:41.494000
21166	2018-09-28 15:52:37.590	00:01:41.500000
21167	2018-09-28 15:52:37.591	00:01:41.501000
21168	2018-09-28 15:52:37.595	00:01:41.505000

Timestamps are generated for each 100 ms, next data is interpolated, finally eye tracking and flight data were merged

```
In [14]: synchronized_data['time']
Out[14]:
0      2018-09-28 16:00:28.500
1      2018-09-28 16:00:28.600
2      2018-09-28 16:00:28.700
3      2018-09-28 16:00:28.800
4      2018-09-28 16:00:28.900
5      2018-09-28 16:00:29.000
6      2018-09-28 16:00:29.100
7      2018-09-28 16:00:29.200
8      2018-09-28 16:00:29.300
9      2018-09-28 16:00:29.400
10     2018-09-28 16:00:29.500
```

4. The software has some bugs which should be addressed for analysis of data. Sometimes eye movement type does not specify correctly, but based on gaze duration and mapped fixation, data is c
5. Eye movement data is almost 10 Gigs (20 participants, 160 flight scenarios). I had to move it as a master file and divide it into files (for each participant). Then, using loops, files were imported, processed and analyzed. Finally, wrangled data was saved in a master dataframe and exported as .csv file. The estimated time for running each scenario is between 5 to 7.5 hrs.

Code to read all eye movements at a time, then making a file for each participants

```
dividing_file = 0

if dividing_file == 1:
    print('================================== READING MASTER FILE OF EYE MOVEMENT ======
    tobii = pd.read_csv('Tobii.tsv', encoding = 'utf-16', delimiter='\t', low_memory=False

    print('================================== DIVIDING MASTER FILE OF EYE MOVEMENT INTO
==================================')
    tobii_recording_names = tobii['Participant name'].unique().tolist()
    tobii_recording_names

    name_counter = 0
    for name in tobii_recording_names:
        name_counter += 1
        tsv_flag = True

        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
        print(name_counter , '=====>', name)
        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++')

        divided_file = tobii.loc[(tobii['Participant name'] == name)]

        if name == 'N04-C-B-LDST':
            divided_file.loc[:, 'Participant name'] = 'N11-E-B-DSTL'
            name = 'N11-E-B-DSTL'

        elif name == 'N04-C-A-STDL':
            divided_file.loc[:, 'Participant name'] = 'N11-E-A-LSDT'
            name = 'N11-E-A-LSDT'
```

Code to read eye tracking file of each participants using loop

```
importing_tsv_files =  [
                        'E01-E-B-LSDT',                                    'E01-E-A-STDL',
                        'N01-C-B-TSDL',                                    'N01-C-A-LTSD',
                        'N02-C-B-TSDL',                                    'N02-C-A-DSTL',
                        'N03-C-B-TSDL',                                    'N03-C-A-DSTL','N
                        'N04-C-A-STDL',                                    'N04-C-B-LDST',
                        'N05-C-B-LSTD',                                    'N05-C-A-STDL','N
                        'N06-C-B-LTDS',                                    'N06-C-A-LDTS',
                        'N07-C-B-SDTL',                                    'N07-C-A-DSTL',
                        'N08-C-B-LDTS',                                    'N08-C-A-LSTD',
                        'N09-C-B-LTDS',                                    'N09-C-A-STDL',

                        'N07-E-B-LSTD',                                    'N07-E-A-LSTD-DE'
                        'N08-E-B-LDST',                                    'N08-E-A-DTSL-LF-
                        'N09-E-B-LTDS',                                    'N09-E-A-SDTL',
                        'N10-E-B-DTSL',                                    'N10-E-A-LTDS',
                        'N11-E-B-DSTL',                                    'N11-E-A-LSDT',
                        'N12-E-B-TDSL-LF-LA','N12-E-B-TDSL-LT-DE',         'N12-E-A-LTSD',
                        'N13-E-B-LDST-LF-LA','N13-E-B-LDST-LT-DE',         'N13-E-A-LTDS',
                        'N14-E-B-LDST',                                    'N14-E-A-SDTL',
                        'N15-E-B-LSTD',                                    'N15-E-A-LDST'
                        ]

p = 0

for file_name in importing_tsv_files:
    p += 1
    print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    print('FILENAME: ', file_name, '                ',p, ' OUT OF:', len(importing_tsv_files),
```

6. Two flight parameter (true air speed of airplane and rate of turn) had not recorded during the experiment, so using polynomial regression it was estimated.

Using polynomial regression to estimate a flight data

```
print('REGRESSION ==> BEFORE ENTERING THE LOOP')
print('')
list_data=[]

for filename in ['Data_20_40.xlsx' , 'Data_15_35.xlsx']:
     data = pd.read_excel(filename)
     list_data.append(data)

dataset = pd.concat(list_data, ignore_index=True)

dataset = dataset[['Vtrue,_ktas', '_Vind,_kias', '__alt,__ind', 'pitch,__deg', '_roll,__deg', 'hding,_true', '

X = dataset[['_Vind,_kias', '__alt,__ind']]
y = dataset['Vtrue,_ktas']

poly_features= PolynomialFeatures(degree = 2)
X_poly = poly_features.fit_transform(X)
poly_features.get_feature_names()

model = Pipeline([('poly', PolynomialFeatures(degree=2)), ('linear', LinearRegression(fit_intercept=False))])
model = model.fit(X, y)
```

7. Eye movement transition matrix was constructed and visualized to measure visual entropy. Then use it in gaze point assessment. I had to construct a dataframe for eye movement to act as a node. It provides information on start of eye movement and its end. Also, I had to consider cases such not counting transition from an area to itself. I added lines of code to make sure it works fine.

Starts of code for construction of transition matrix

```
print('TRANSITION MATRIX')
columns = ['From','To']
index = range(0, math.ceil(len(df_fixation)/2))

df_from_to = pd.DataFrame(index = index, columns = columns)

counter_index = 0

for i in range(0, math.ceil(len(df_fixation)/2)):
    #time.sleep(0.01)
    #print(counter_index, i, df_fixation['AOI'].iloc[counter_index], df_fi
    df_from_to['From'].iloc[i] = df_fixation['AOI'].iloc[counter_index]
    df_from_to['To'].iloc[i] = df_fixation['AOI'].iloc[counter_index+1]

    counter_index += 1

# INDENT
df_from_to = df_from_to[df_from_to['From'] != df_from_to['To']]
df_from_to = df_from_to.reset_index(drop=True)
df_crosstab = pd.crosstab(df_from_to.From, df_from_to.To)
```

Addressing common problems when constructing transition matrix (for example there is not data about look point)

```
for i in AOI_list:
    if (df_from_to['From'].str.contains(i).any() == False and df_from_to['To'].str.contains(i).any() == False) or \
    (df_from_to['From'].str.contains(i).any() == True and df_from_to['To'].str.contains(i).any() == False) or \
    (df_from_to['From'].str.contains(i).any() == False and df_from_to['To'].str.contains(i).any() == True):

        s = pd.Series([i, i], index=['From', 'To'])
        df_from_to = df_from_to.append(s, ignore_index=True)

row_index = 0
for row in df_from_to.iterrows():

    if row_index == df_from_to.index[-1]:
        break
    else:
        if pd.isnull(df_from_to['To'].iloc[row_index]) == True and \
        pd.isnull(df_from_to['From'].iloc[row_index+1]) == True:
            print(index)
            df_from_to.at[row_index, 'To'] = df_from_to['To'].iloc[row_index+1]
            df_from_to.drop(row_index+1, inplace=True)
            df_from_to = df_from_to.reset_index(drop=True)
            row_index -= 1

    row_index += 1

df_crosstab = pd.crosstab(df_from_to.From, df_from_to.To)

for i in AOI_list:
    df_crosstab[i][i] = 0
```

8. whereas data was neutralized, names of recorded eye movement files were used to distinguish research group using methods such as np.where and lambda

Sample of code to add columns to dataframe to distinguish groups

```
df_eye['Participant name'] = df_eye['Participant name'].str[:12]
df_eye['Number'] = df_eye['Participant name'].str[1:3]
df_eye['Skill'] = np.where(df_eye['Participant name'].str[0:1]=='E', 'Expert', 'Novice')
df_eye['Group'] = np.where(df_eye['Participant name'].str[0:1]=='E', 'Expert',
      np.where(df_eye['Participant name'].str[4:5]=='E', 'Experimental', 'Control'))
```

K-means clustering on extreme cases:

To understand whether k-means clustering is beneficial to detect scanning pattern, eye movements of one expert, trained pilots and untrained pilots were compared. I conducted it on extreme cases. It means that **three of the best trained pilots** were selected from experimental group and **three of the worst pilots** were selected from control group. The worst pilots were selected from those with the highest values of visual entropy who lost control of the airplane. In other words, pilots who applied random scanning pattern with poor piloting performance were selected from control group. A high value of visual entropy indicate less structured scanning pattern and random scanning pattern. Less structured scanning pattern has adverse effect in flight as it affects pilots' information processing capabilities (pilots' behaviors were studied prior & post treatment training. This report is about post training assessment) Low visual entropy, on the other hand, indicates limited visual search. Visual entropy is measured according to Shannon entropy which is defined as:

$$H(X) = -\sum_{i=1}^{N} P(x_i) \log_2 P(x_i)$$

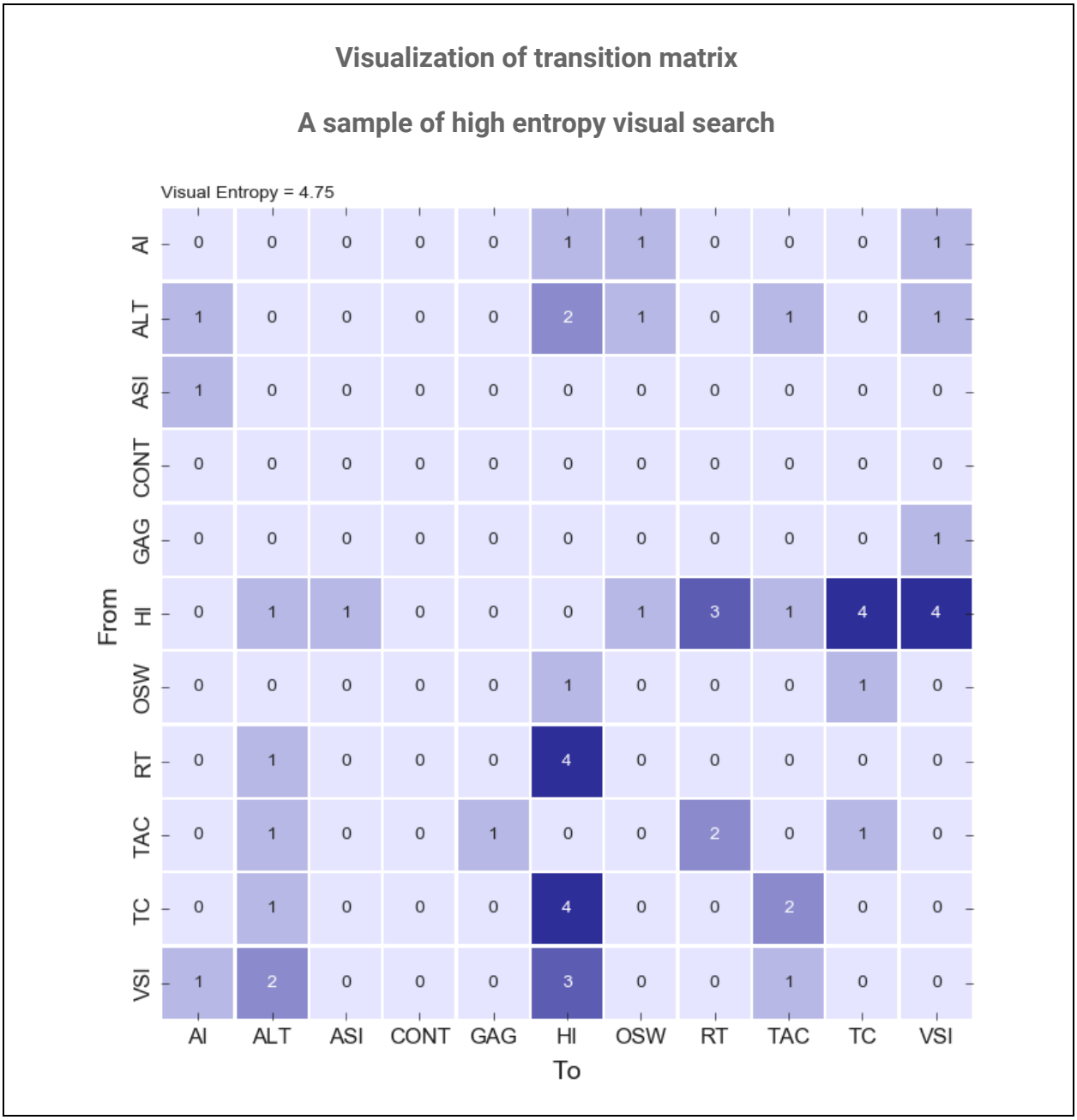
Where P(x) is probability of transition from a given AOI to another AOI. To compute this, fixation on AOIs were counted and divided by the total number of fixation. In this research, 11 AOIs were defined.

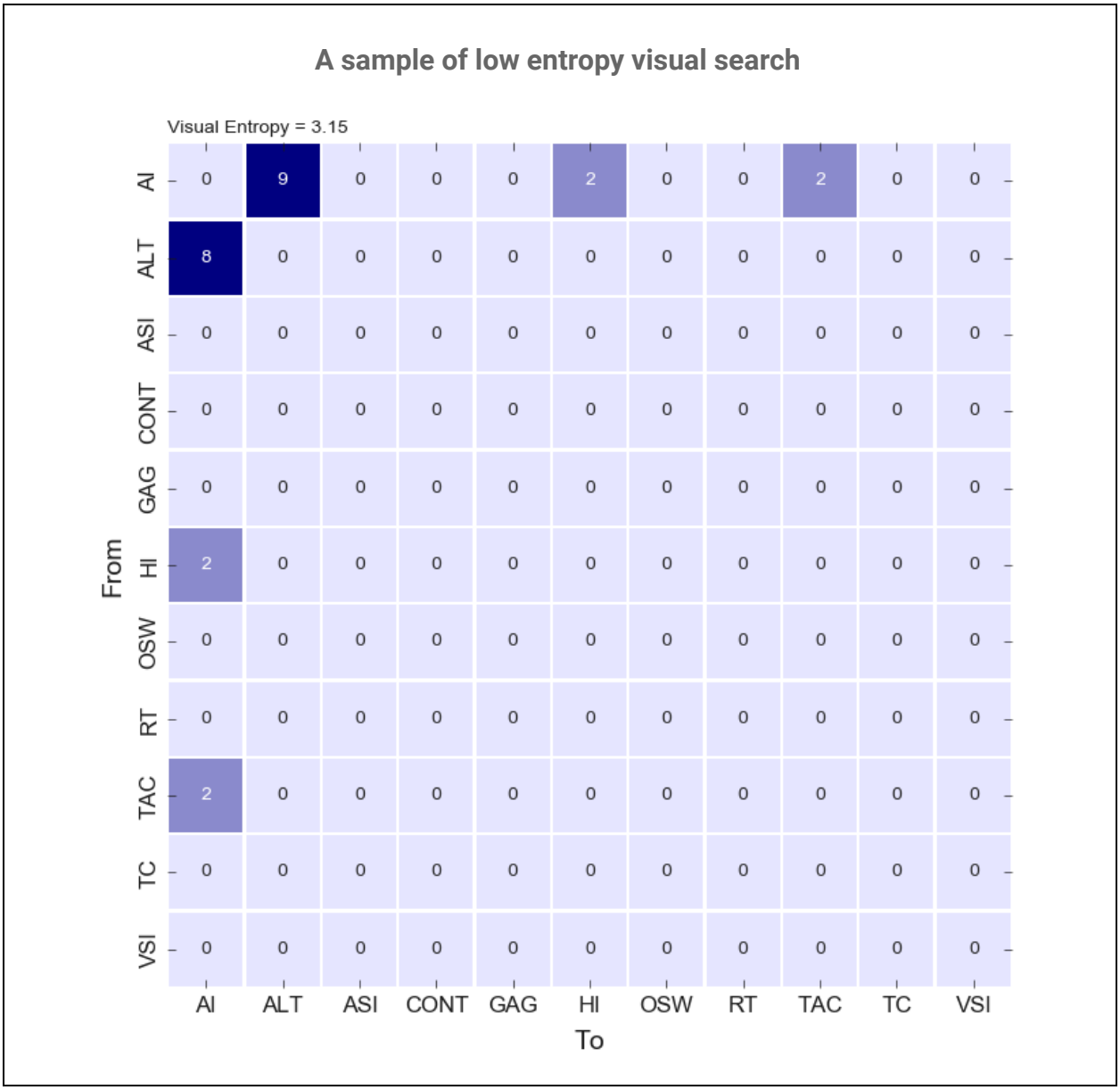
Computing Visual Entropy

```
probability_matrix = transition_matrix.div(transition_matrix.values.sum())
log_probability_matrix = probability_matrix.copy()
log_probability_matrix.iloc[:,:] = 0

for i in range (0, len(AOI_list)):
    log_probability_matrix[i] = probability_matrix[i].apply(lambda x: math.log2(x) if x > 0 else 0)

entropy_matrix = probability_matrix * log_probability_matrix
entropy = -entropy_matrix.values.sum()


scenario['entropy'] = entropy
print('ENTROPY: ', entropy)
```

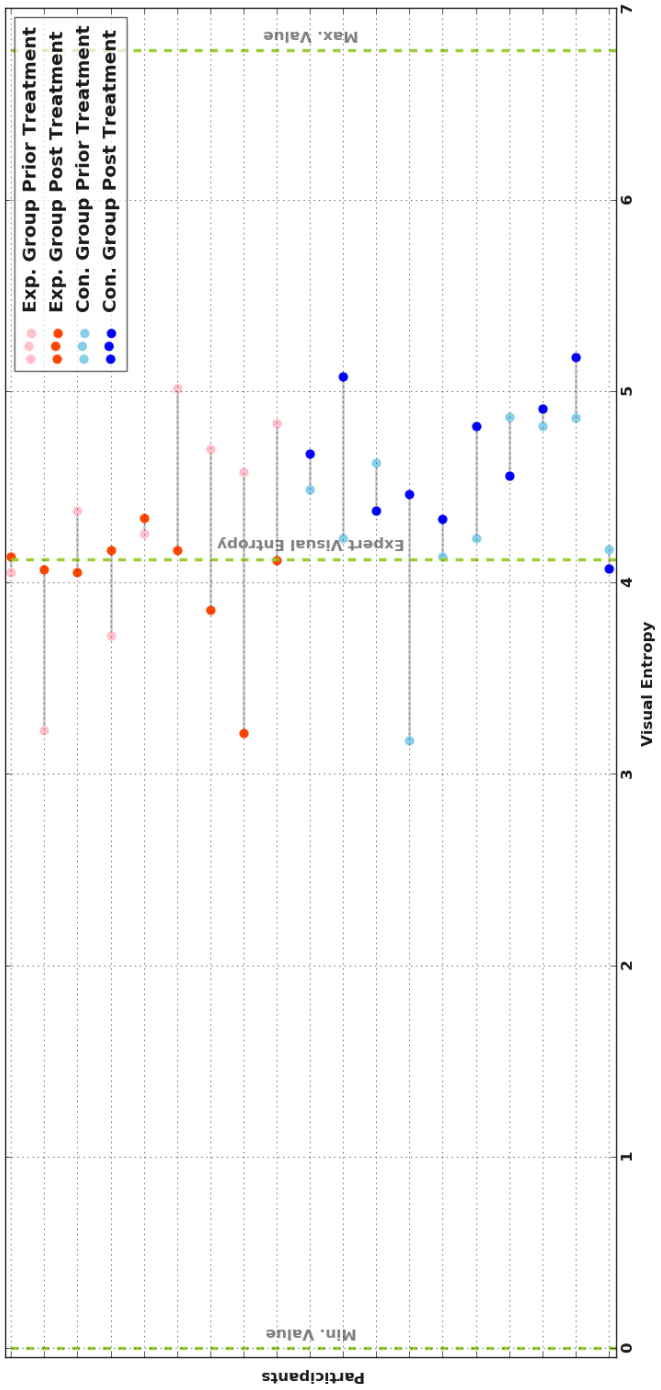
Visualization of transition matrix using heatmap in seaborn

```
style.use('classic')

# heatmap of transion matrix
# cmap='Set3'

f, ax = plt.subplots(figsize=(8, 8))


#my_cmap ='Set3'
my_cmap = sns.light_palette("Navy", as_cmap=True)

sns.set(font_scale = 1.0)
sns.heatmap(df_crosstab_plot, annot=True, cbar=False, cmap=my_cmap, fmt='d', square=True, linewidths= 1.
ax.figure.axes[0].yaxis.label.set_size(6)

a1 = ax.get_xlabel()
a2 = ax.get_ylabel()

ax.set_xlabel(a1, fontsize = 14)
ax.set_ylabel(a2, fontsize = 14)
#ax.get_xaxis().set_label_coords(0,0.5)
ax.get_yaxis().set_label_coords(-0.05, 0.5)
ax.get_xaxis().set_label_coords(0.5, -0.05)
ax.set_title('Visual Entropy = '+ str(np.round_(entropy,2)), loc='left', fontname='arial', fontsize=10)
```


Visualization of transition matrix
A sample of high entropy visual search
Visual Entropy = 4.75
From
AI 0 0 0 0 0 1 1 0 0 0 1
ALT 1 0 0 0 0 2 1 0 1 0 1
ASI 1 0 0 0 0 0 0 0 0 0 0
CONT 0 0 0 0 0 0 0 0 0 0 0
GAG 0 0 0 0 0 0 0 0 0 0 1
HI 0 1 1 0 0 0 1 3 1 4 4
OSW 0 0 0 0 0 1 0 0 0 1 0
RT 0 1 0 0 0 4 0 0 0 0 0
TAC 0 1 0 0 1 0 0 2 0 1 0
TC 0 1 0 0 0 4 0 0 2 0 0
VSI 1 2 0 0 0 3 0 0 1 0 0
AI ALT ASI CONT GAG HI OSW RT TAC TC VSI
To

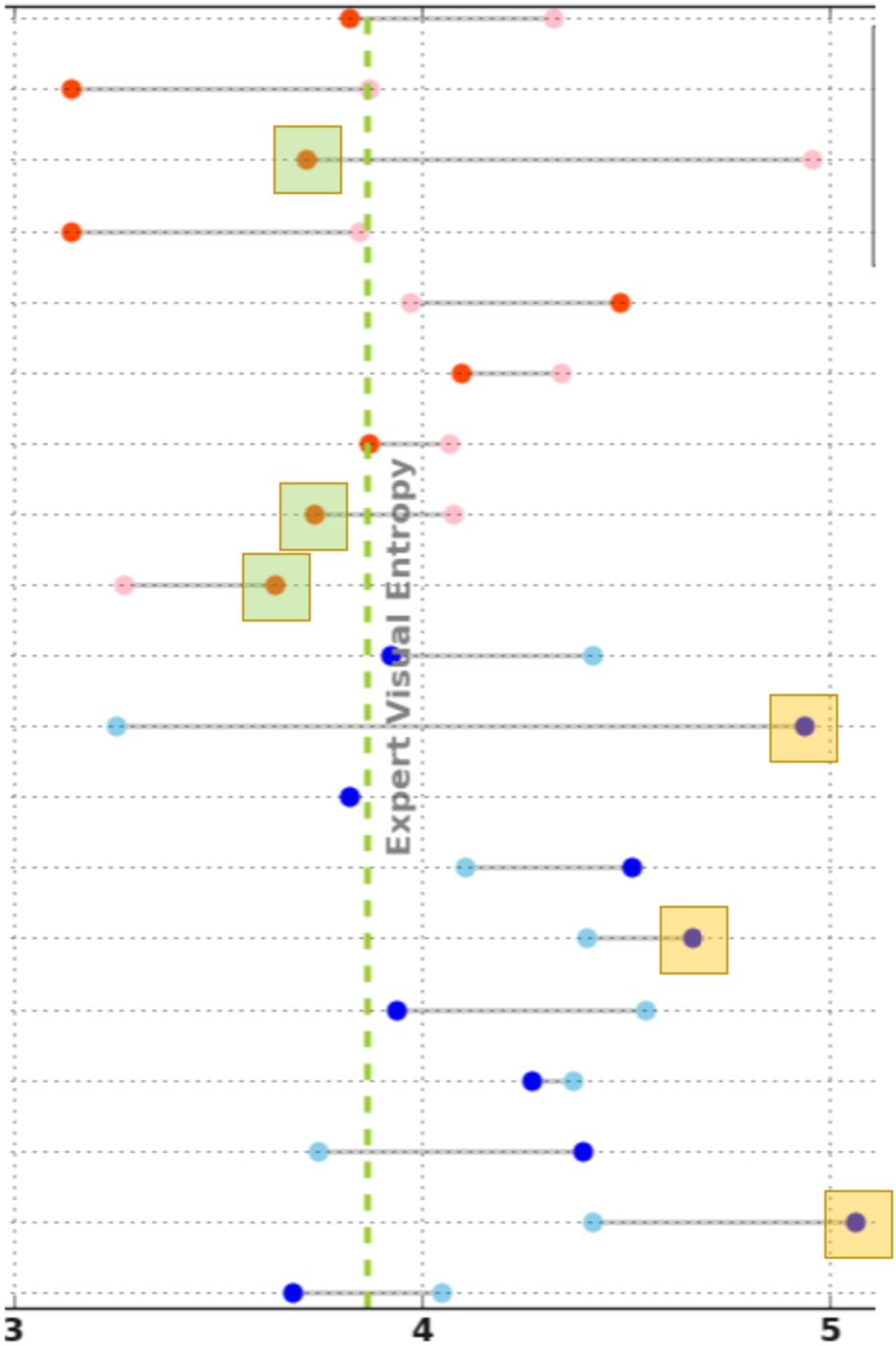

A sample of low entropy visual search
Visual Entropy = 3.15
From
AI
ALT
ASI
CONT
GAG
HI
OSW
RT
TAC
TC
VSI
To
AI
ALT
ASI
CONT
GAG
HI
OSW
RT
TAC
TC
VSI
0 9 0 0 0 2 0 0 2 0 0
8 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0

Visualization of change in visual entropy prior & post treatment (using lolliplot)



I used Lolli plot to monitor the changes in scanning pattern prior and post training. I used it to select the highest values of visual entropy after training.

The selection of participants from experimental & control group based on visual entropy



Visualization code (first part)

```
plt.rcParams.update({'font.size': 12})

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(15, 7), sharex=True)

dwell_df = pd.read_csv(filename)
entropy_df = pd.read_csv(filename)

entropy_df = entropy_df.drop_duplicates(subset=['Participant name', 'Training'], keep='first')

subject_name = dwell_df['Participant name'].unique().tolist()
group_name = dwell_df['Group'].unique().tolist()
entropy_df['TOI'] = np.nan

for name_ in subject_name:
    for group_ in group_name:

        entropy_df['TOI'].loc[(entropy_df['Participant name'] == name_) & (entropy_df['Group'] == group_)]

entropy_df = entropy_df[['Participant name', 'Training', 'Group', 'entropy', 'TOI']]

entropy_df = entropy_df.reset_index(drop = True)
entropy_df
for index, row in entropy_df.iterrows():
    if row['Participant name'][0] == 'E':
        entropy_df = entropy_df.drop(entropy_df.index[index])

entropy_df = entropy_df.reset_index(drop = True)

entropy_df['Training'] = np.where(entropy_df['Participant name'].str[6:7]=='A', 'After', 'Before')


entropy_df['Group'] = entropy_df['Group'].str.replace('Control','Con.')
entropy_df['Group'] = entropy_df['Group'].str.replace('Experimental','Exp.')

con_group_range = entropy_df['Group'].loc[entropy_df['Group'] == 'Con.'].count()
exp_group_range = entropy_df['Group'].loc[entropy_df['Group'] == 'Exp.'].count()

entropy_df['no'] = np.nan

row_counter = 1
con_counter = 1

for index, row in entropy_df.iterrows():
    if index == entropy_df.index[-1]:
        break

    elif entropy_df['Group'].iloc[index] == 'Con.':
        if row_counter % 2 != 0:
            entropy_df['no'].iloc[index] = con_counter
            entropy_df['no'].iloc[index+1] = con_counter
            con_counter += 1
```

Visualization code (second part)

```
entropy_df = entropy_df[['Participant name', 'no', 'Group', 'Training', 'entropy', 'TOI']]
entropy__expert_threshold = entropy_df['entropy'].iloc[0:2].mean()

#my_color=np.where((ordered_df['group']=='experimental') , 'orange', 'skyblue')
#my_size=np.where(ordered_df ['group']=='B', 70, 30)

axes.hlines(y=entropy_df['no'].loc[entropy_df['Training'] =='Before'],
           xmin=entropy_df['entropy'].loc[(entropy_df['Training'] =='Before')],
           xmax=entropy_df['entropy'].loc[(entropy_df['Training'] =='After')],
           color='grey', alpha=0.4, zorder=1, linewidth=2.0)

axes.axvline(entropy__expert_threshold, linestyle='dashed', linewidth=2.5, color = 'yellowgreen')
axes.text(x = entropy__expert_threshold + 0.05, y = (entropy_df['no'].max() +1) / 2, s='Expert Visual

axes.axvline(-math.log2(110**-1), linestyle='dashed', linewidth=2.5, color = 'yellowgreen')
axes.text(x = 6.83, y = (entropy_df['no'].max() +1) / 2, s='Max. Value',  rotation=90, verticalalignm

axes.axvline(-math.log2(1**-1), linestyle='dashed', linewidth=2.5, color = 'yellowgreen')
axes.text(x = 0.05, y = (entropy_df['no'].max() +1) / 2 , s='Min. Value',  rotation=90, verticalalign

E1 = axes.scatter(entropy_df['entropy'].loc[(entropy_df['Training'] =='Before') & (entropy_df['Group'
            entropy_df['no'].loc[(entropy_df['Training'] =='Before') & (entropy_df['Group'] =='Exp.')

E2 = axes.scatter(entropy_df['entropy'].loc[(entropy_df['Training'] =='After') & (entropy_df['Group']
            entropy_df['no'].loc[(entropy_df['Training'] =='After') & (entropy_df['Group'] =='Exp.')]

E3 = axes.scatter(entropy_df['entropy'].loc[(entropy_df['Training'] =='Before') & (entropy_df['Group'
            entropy_df['no'].loc[(entropy_df['Training'] =='Before') & (entropy_df['Group'] =='Con.')

E4 = axes.scatter(entropy_df['entropy'].loc[(entropy_df['Training'] =='After') & (entropy_df['Group']
            entropy_df['no'].loc[(entropy_df['Training'] =='After') & (entropy_df['Group'] =='Con.')]

#plt.rc('grid', linestyle="-", color='black')
axes.set_yticklabels([])
# Add title and axis names
axes.grid()

axes.legend(loc="upper right")
axes.legend([E1, E2, E3, E4],['Exp. Group Prior Treatment', 'Exp. Group Post Treatment', 'Con. Group
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

plt.xlabel('Visual Entropy', fontweight='bold')
plt.ylabel('Participants', fontweight='bold')
plt.xlim(-0.05 , 7)
plt.yticks(np.arange(1, entropy_df['no'].max()+1 , 1.0))

# Set spacing between y ticks,
plt.gca().margins(y=0)
```

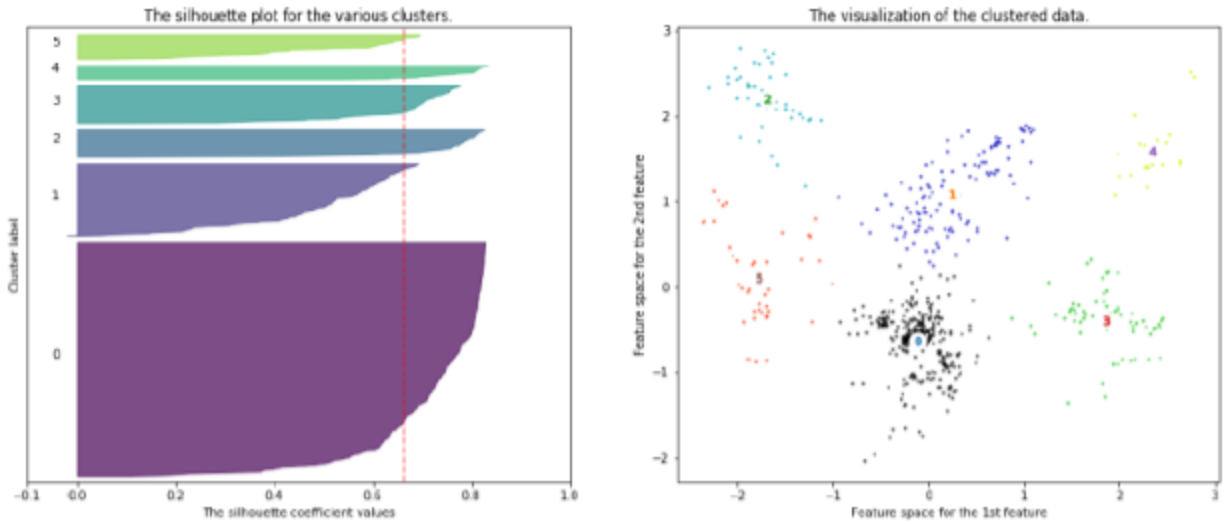
Result:

I checked the optimal number of gaze points' clusters using k-mean clustering. The silhouette score was computed for 2 to 6 clusters. The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.

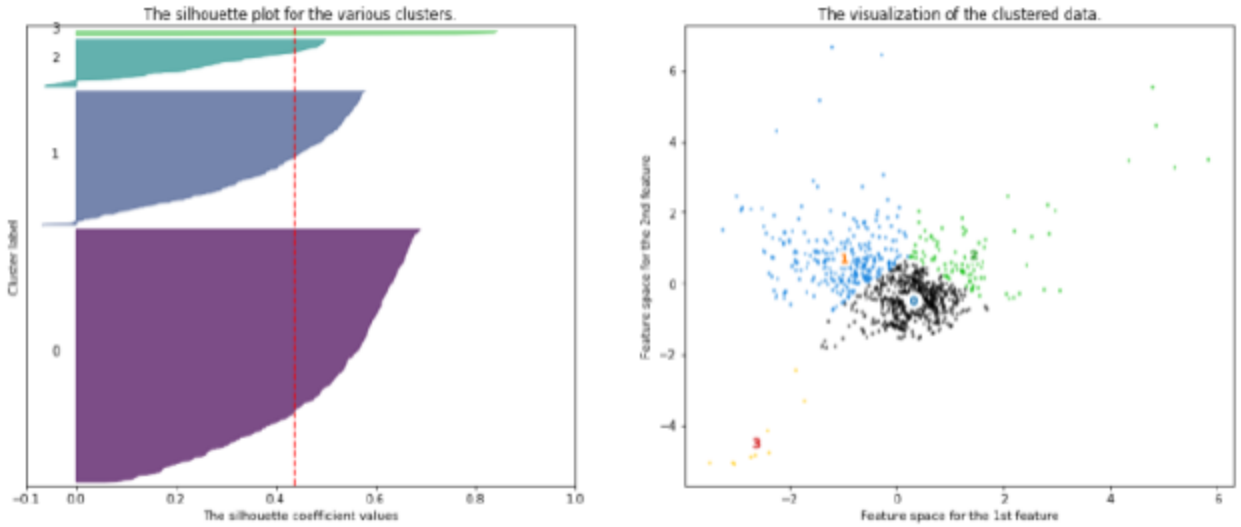
The result showed there is a pattern in the gaze points of pilots: the higher the performance of the pilots, the more number of clustered points. In other words, gaze points of better pilots were grouped in higher number of clusters than less skilled pilots. In means that piloting and number of gaze point clusters are related to each other.

The result showed that the expert had the highest optimal number of clusters: 6. Then, pilots of experimental groups with 3, 4 and 3 clusters of gaze points. While, gaze points of pilots with poor piloting performance (control group) were groups in 2, 3, and 2 clusters.

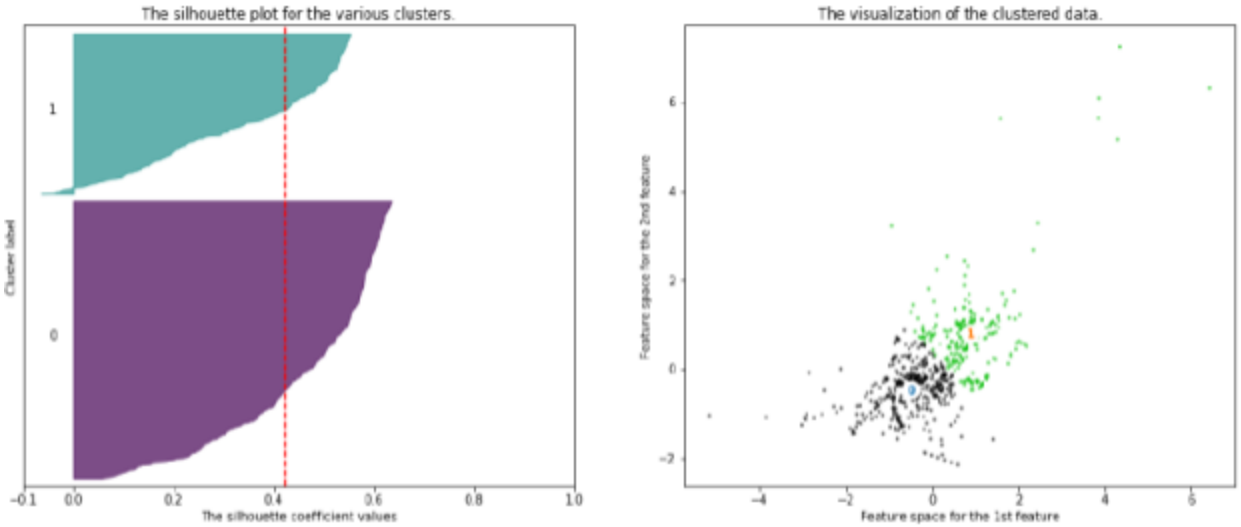
This finding is for one flight scenario could expand for other flight scenario.



Fixation clustering of a flight instructor



Fixation clustering of the experimental group (did not lose control)



Fixation clustering of a pilot of control group (lost control)

Clustering code (first part)

```
clmns = ['Gaze point X', 'Gaze point Y']

df_tr_std = stats.zscore(eye_final[['Gaze point X', 'Gaze point Y']])

kmeans = KMeans(n_clusters=3, random_state=0).fit(df_tr_std)

labels = kmeans.labels_
eye_final['clusters'] = labels
#Add the column into our list
clmns.extend(['clusters'])
#Lets analyze the clusters
print (eye_final[clmns].groupby(['clusters']).mean())

sns.lmplot('Gaze point X', 'Gaze point Y',
           data=eye_final,
           fit_reg=False,
           hue="clusters",
           scatter_kws={"marker": "D",
                        "s": 100})
plt.title('Clusters Wattage vs Duration')
plt.xlabel('Wattage')
plt.ylabel('Duration')

# Defining number of cluster using elbow method
sse = []
list_k = list(range(1, 10))

for k in list_k:
    km = KMeans(n_clusters=k)
    km.fit(df_tr_std)
    sse.append(km.inertia_)

# Plot sse against k
plt.figure(figsize=(6, 6))
plt.plot(list_k, sse, '-o')
plt.xlabel(r'Number of clusters *k*')
plt.ylabel('Sum of squared distance');

# defining the silhouette_scores for 2 clusters to 6 clusters
cluster_range = range( 2, 6 )

for n_clusters in cluster_range:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)
```

Clustering code (second part)

```
  # The 1st subplot is the silhouette plot
  # The silhouette coefficient can range from -1, 1 but in this example all
  # lie within [-0.1, 1]
ax1.set_xlim([-0.1, 1])
  # The (n_clusters+1)*10 is for inserting blank space between silhouette
  # plots of individual clusters, to demarcate them clearly.
ax1.set_ylim([0, len(df_tr_std) + (n_clusters + 1) * 10])

  # Initialize the clusterer with n_clusters value and a random generator
  # seed of 10 for reproducibility.
clusterer = KMeans(n_clusters=n_clusters, random_state=10)
cluster_labels = clusterer.fit_predict( df_tr_std )

  # The silhouette_score gives the average value for all the samples.
  # This gives a perspective into the density and separation of the formed
  # clusters
silhouette_avg = silhouette_score(df_tr_std, cluster_labels)
print("For n_clusters =", n_clusters, "The average silhouette_score is :", silhouette_avg)

  # Compute the silhouette scores for each sample
sample_silhouette_values = silhouette_samples(df_tr_std, cluster_labels)
y_lower = 10

for i in range(n_clusters):
  # Aggregate the silhouette scores for samples belonging to
  # cluster i, and sort them
    ith_cluster_silhouette_values = \
    sample_silhouette_values[cluster_labels == i]

    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    cmap = plt.cm.viridis
    color = cmap(float(i) / n_clusters)

    ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values, facec

      # Label the silhouette plots with their cluster numbers at the middle
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

      # Compute the new y_lower for next plot
    y_lower = y_upper + 10  # 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

The vertical line for average silhoutte score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
```

Clustering code (third part)

```
  ax1.set_yticks([])  # Clear the yaxis labels / ticks
  ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

# 2nd Plot showing the actual clusters formed
  colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
  ax2.scatter(df_tr_std[:, 0], df_tr_std[:, 1], marker='.', s=30, lw=0, alpha=0.7,c=colors)

# Labeling the clusters
  centers = clusterer.cluster_centers_
# Draw white circles at cluster centers
  ax2.scatter(centers[:, 0], centers[:, 1], marker='o', c="white", alpha=1, s=200)

  for i, c in enumerate(centers):
      ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1, s=50)

  ax2.set_title("The visualization of the clustered data.")
  ax2.set_xlabel("Feature space for the 1st feature")
  ax2.set_ylabel("Feature space for the 2nd feature")

  plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                "with n_clusters = %d" % n_clusters),
               fontsize=14, fontweight='bold')
  plt.show()
```

This is two tailed test and p-value is less than 0.05 (0.097 / 2). With 95%, I reject the null hypothesis that number of gaze points clusters for good pilots and bad pilots are equal. I do believe I need more samples of pilots that lost control of the airplane to draw a solid conclusion here. The finding could be used to develop a warning system for airplane (in order to take control of the airplane in cases pilots lost control) based on pilots scanning pattern.

•**Null hypothesis (H0)**: the mean of number of clusters of novice pilots does not differ from trained pilots or expert pilot

•**Alternative Hypothesis (H1)** : the mean of number of clusters of novice pilots does differ from trained pilots or expert pilot

•**Two tailed T-test**

•**Confidence Level = 95%**

```
import scipy.stats as stats
import numpy as np

sample1 = [3,4,3,6]
sample2 = [2,3,2]

t_stat, p_val = stats.ttest_ind(sample1, sample2, equal_var = False)
```

```
t_stat, p_val
```

```
(2.132007163556104, 0.09713437094229309)
```