Problem Statement:

During the recently completed PSY-001 course, various click actions, generated by users, were

logged by the web application and stored for later usage. Calculate any 2 of the analytic features listed below:

- List of most active users
- Peak usage time
- Most Used Internet Browser or Device
- Users most used language
- Most viewed videos

This document is split into two parts, the first part will be the data preprocessing that I had to do to make a clean data, the second part is about my analysis to find the answers for the above questions.

PART 1: DATA PREPROCESSING

Type of data in the file:

Text file with the clickthrough data in the JSON format.

What is a clickstream data?

The clickstream data is the data which helps us to determine the user behaviour. The users always leave some trail with this data. Mostly used for sentimental analysis and identify consumer behaviour.

Initial thought process seeing the data:

This is the first time I am working with the clickstream analysis and initially I was a little overwhelmed seeing a huge 2GB data file. After some analysis by reading the snippet data provided, I came to conclusion to break down the problem into smaller chunks.

The first thing I noticed was the file had data was unstructured (I should call it as semi-structured) as I could find some patterns in the data. It was in the JSON format (Key-Value pair).

1st Step:

- So, my 1st step was to convert this semi structured data to excel format or machine-readable format, but the data was dirty, so, I had to again break the data into small chunks.
- I could see that the JSON values were nested, which required me to write a code to split them.
- I decided to use python as my language as it had better high-level libraries.

2nd Step:

- Identifying the libraries required, python has a good number of libraries which made my work easier. The list of libraries which I used are given below.
    - Pandas
    - Json
    - Matplotlib
    - UAParser

- Datetime

3rd Step:

Preprocessing:

Breakdown of the problem and logic used:

- I initially wanted to test my code with very few rows, so I decided to extract 1000 rows from the 2GB file. And convert this small file from text to XLSX format, which would be easier to analyse.

```
counter = 0
click_data = []
for line in open('psy-001_clickstream_export.txt', 'r'):
    json_val = json.loads(line)
    # val = pd.io.json.json_normalize(json_val)
    click_data.append(json_val)
    counter += 1
    if counter == 1001:
        break
```

- The counter helps to stop the code from running once it reaches my condition as 1001.

Output:

```
df = pd.DataFrame.from_dict(click_data)

print(df.head(5))
```

...

0 ...

1 ...

2 ...

3 ...

4 ...

As you can see this was not the desired output, so I had to do something to make the display the entire output. I found something to get my desired output with the code shown below.

```
"""
To print the whole dataset
"""
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', -1)
df = pd.DataFrame.from_dict(click_data)
```

```
print(df.head(5))
```

The above code allowed to print the entire dataset as I desired.

4th step:

Extraction and removal of unwanted expressions from the nested column: (To make a clean data)

- Next my job was to split and remove all the unwanted regular expressions like [],{}, '' etc. The code to remove the regular expressions is given below:
- As we can see from the file some of the important data about the video is given inside the value column, which I had to split to extract information in machine-readable format.
- The output of the code is given below.

```
"""
code to remove the regular expressions from the data and the  nested column
"""
new = df['value'].str.split(",", expand=True)

print(len(new.columns))

for i in range(len(new.columns)):
    if i == len(new.columns) - 1:
        split_colon = new[i].str.split(":", expand=True)
        ident = split_colon[0][0].split('"')[1]
        split_brac = split_colon[1].str.split("}", expand=True)
        df[ident] = split_brac[0]
    else:
        split_colon = new[i].str.split(":", expand=True)
        ident = split_colon[0][0].split('"')[1]
        df[ident] = split_colon[1]

print(df.head(5))
```

currentTime	laybackRat	paused	error	etworkStat	readyState	eventTimestamp	initTimestamp	type
138.83799743652344	2	false	null	1	4	1368217522545	1368217496709	"seeked"
9e62014cf146709f8dfceddb3afc170								
9e62014cf146709f8dfceddb3afc170								
152.39935302734375	2	false	null	1	4	1368217523726	1368217496709	"seeked"
167.00390625	2	false	null	1	4	1368217524570	1368217496709	"seeked"
196.21298217773438	2	false	null	1	4	1368217526248	1368217496709	"seeked"
31.39028549194336	2	false	null	2	4	1368217527202	1368217496709	"seeked"
31.39028549194336	2	false	null	2	4	1368217527203	1368217496709	"seeked"
0	2	false	null	2	4	1368217528244	1368217496709	"seeked"

The information is clear now.

5th Step:

To delete useless information from the table after extraction.

As you can see, I have extracted the information from the nested JSON data, now I can delete the column which can reduce the size of the file. To drop the column, I used the below code. Some of the unwanted columns are shown below:

	A	B	C	D	E	F	G	H	I
		12	13	client	from	key	language	page_url	session
0				spark	https://cla	user.video	es,en-US;q	https://cla	857960598
1				spark	https://cla	pageview	es-ES,es;q=	https://cla	131890403
2				spark	https://cla	pageview	es-ES,es;q=	https://cla	131890403
3				spark	https://cla	user.video	es,en-US;q	https://cla	857960598
4				spark	https://cla	user.video	es,en-US;q	https://cla	857960598
5				spark	https://cla	user.video	es,en-US;q	https://cla	857960598
6				spark	https://cla	user.video	es,en-US;q	https://cla	857960598
7				spark	https://cla	user.video	es,en-US;q	https://cla	857960598
8				spark	https://cla	user.video	es,en-US;q	https://cla	857960598
9				spark	https://cla	user.video	es,en-US;q	https://cla	857960598
10				spark	https://cla	user.video	es,en-US;q	https://cla	857960598
11				spark	https://cla	user.video	es,en-US;q	https://cla	857960598
12				spark	https://cla	user.video	es,en-US;q	https://cla	857960598
13				spark	https://cla	user.video	es,en-US;q	https://cla	857960598
14				spark	https://cla	user.video	es,en-US;q	https://cla	857960598
15				hg.client	https://cla	pageview	es-ES,es;q=	https://cla	131890403

```
"""
Dropping the split column to avoid redundancy
"""
df = df.drop(columns="value")
df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')  # Converting timestamp
to human readable
df.drop(df.columns[[0, 1]], axis=1, inplace=True)  # To remove the unwanted first two
columns
```

Below you can see the changes in the timestamp column, the code for that is shown.

	I	J
	session	timestamp
	8579605985-1368217057801	1.36822E+12
	1318904033-1368216996291	1.36822E+12
	1318904033-1368216996291	1.36822E+12
	8579605985-1368217057801	1.36822E+12
	8579605985-1368217057801	1.36822E+12
	8579605985-1368217057801	1.36822E+12
	8579605985-1368217057801	1.36822E+12

	G	H
	session	timestamp
	8579605985-1368217057801	2013-05-10 20:25:15
	1318904033-1368216996291	2013-05-10 20:25:15
	1318904033-1368216996291	2013-05-10 20:25:15
	8579605985-1368217057801	2013-05-10 20:25:16
	8579605985-1368217057801	2013-05-10 20:25:17
	8579605985-1368217057801	2013-05-10 20:25:19
	8579605985-1368217057801	2013-05-10 20:25:20

6th Step:

Use of API to extract information from User_agent column:

Some information about User agent in webpages:

User agent:

User agent

From Wikipedia, the free encyclopedia

In computing, a **user agent** is software (a software agent) that is acting on behalf of a user. One common use of the term refers to a web browser that "retrieves, renders and facilitates end user interaction with Web content".[1]

I could see the user agent column has lots of information which we need to do some analytics, a sample of the information is given below.

API used:

https://github.com/ua-parser/uap-python This is the API which I used to extract the information from the column. Please download it from the same before executing the code.

user_agent
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.64 Safari/537.31
Mozilla/5.0 (Windows NT 6.0) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.64 Safari/537.31
Mozilla/5.0 (Windows NT 6.0) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.64 Safari/537.31
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.64 Safari/537.31
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.64 Safari/537.31
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.64 Safari/537.31
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.64 Safari/537.31
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.64 Safari/537.31
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.64 Safari/537.31

This is the column which is going to give us browser and OS information, the code to extract that is given below.

```
"""
Code snippet to split the OS and browser name from User agent
"""
df['User browser'] = 'dummy'
for index, row in df.iterrows():
    # print(row['user_agent'])
    ua_string = row['user_agent']
    parsed_string = user_agent_parser.ParseUserAgent(ua_string)
    # print(parsed_string['family'])
    df.at[index, 'User browser'] = parsed_string['family']

df['User Device/OS'] = 'dummy'
for index, row in df.iterrows():
    # print(row['user_agent'])
    ua_string = row['user_agent']
```

```
    parsed_string = user_agent_parser.ParseOS(ua_string)
    print(parsed_string['family'])
    df.at[index, 'User Device/OS'] = parsed_string['family']  # To replace dummy as
the value found
```

7th Step:

The final step of preprocessing:

My final step was to convert the extracted information to an excel format which will make my analysis easier. The code snippet which I used to convert it to the recommended format is given below:

```
"""
Code to convert to Excel
"""

writer = pd.ExcelWriter('click_1L.xlsx', engine='xlsxwriter',
options={'strings_to_urls': False})
df.to_excel(writer, sheet_name='1L values')
writer.save()
```

There its over, the process I went through to convert the semi structured data to useful and machine-readable information. Thanks to all the existing libraries. The excel file is uploaded for further information.

PART 2: DATA ANALYSIS

My next job was to use this useful data to extract insights.

Questions to be asked from the data:

1. Most Used Internet Browser or Device
2. List of most active users
3. Users most used language

The reason for all the preprocessing is to extract insights from the data, this is because “Data is the new gold”. To get this information I had to do the following steps.

1st step:

Loading and reading the Excel file into python

- The code for the above operation is given below

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style

style.use('bmh')  # style format for plotting

if __name__ == '__main__':
    df = pd.read_excel(r'click_1L.xlsx')
```

```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', -1)
print(df.head())
```

- Now the data is loaded into the data frame.

PS: I have loaded just 100000 rows into the data frame for the ease of analysis.

2nd Step:

Removing the null values

- Null values are basically useless, so to see how many nulls there in our data are, I used the below code.

```
print(df.shape)
print(df.isnull().sum())
```

- This code returned me the number of null values and which can be avoided during the analysis.

(100001, 22) → The number of rows and columns

The list of null values present in the data

```
client            0
from              2455
key               0
language          0
page_url          0
session           0
timestamp         0
user_agent        0
user_ip           0
username          0
currentTime       44652
playbackRate      51969
paused            44650
error             98978
networkState      46673
readyState        46673
eventTimestamp    44766
initTimestamp     44766
type              44766
prevTime          44768
User browser      0
User Device/OS    0
```

- As we can see some of the columns are of no use to us, so I decided to drop those. And for the “from” row I just dropped the rows not the entire column.

3rd Step:

Dropping the columns and Null values

```
df.drop(df.columns[[7, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]], axis=1, inplace=True)
print(df.isnull().sum())
df = df.dropna(axis=0) # To drop the null rows in the from column
print(df.isnull().sum())
```

Output:

```
User browser      0
User Device/OS    0
client            0
from              0
key               0
language          0
page_url          0
session           0
timestamp         0
user_ip           0
username          0
User browser      0
User Device/OS    0
```

- And there are null values now in the data, now we are good to go with our analysis.

4th step:

Another BIG problem

- Though the data is clean now, we have lots of duplicates as the same user can login to the website multiple times, next my goal was to find the unique users and group them accordingly

```
"""
Removing the duplicates in the dataframe
"""
MUOS = (df.groupby(df['User
Device/OS'])['username'].nunique().nlargest(10).reset_index(name='count'))
print(MUOS)
MOB = (df.groupby(df['User
browser'])['username'].nunique().nlargest(10).reset_index(name='count'))
print(MOB)
MFU =
(df.groupby(df['username']).size().nlargest(10).reset_index(name='count'))
print(MFU)
print(MFU.count(axis=0))
```

- This helped me group all the unique users and sort them accordingly.

PS: I have taken just the top 10 users in all the cases.

```
"""
Terminologies:
MUOS - MOST USED OPERATING SYSTEM
MOB - MOST USED BROWSER
MFU - MOST FREQUENT USER
MUL - MOST USED TO LANGUAGE
"""
```

CASE MISMATCH WHILE FINDING THE MOST USED LANGUAGE:

- When I was trying to find the most used language, it had something called q which meant **relative quality factor,** it says about the probability of knowing other mention language in the row.
- I felt that information is not needed right now and there was also a case mismatch, which I had to resolve.

language
es,en-US;q=0.8,en;q=0.6
es-ES,es;q=0.8
es-ES,es;q=0.8
es,en-US;q=0.8,en;q=0.6
es,en-US;q=0.8,en;q=0.6
es,en-US;q=0.8,en;q=0.6
es,en-US;q=0.8,en;q=0.6

- Using the code below, I was able to do the above-mentioned things.

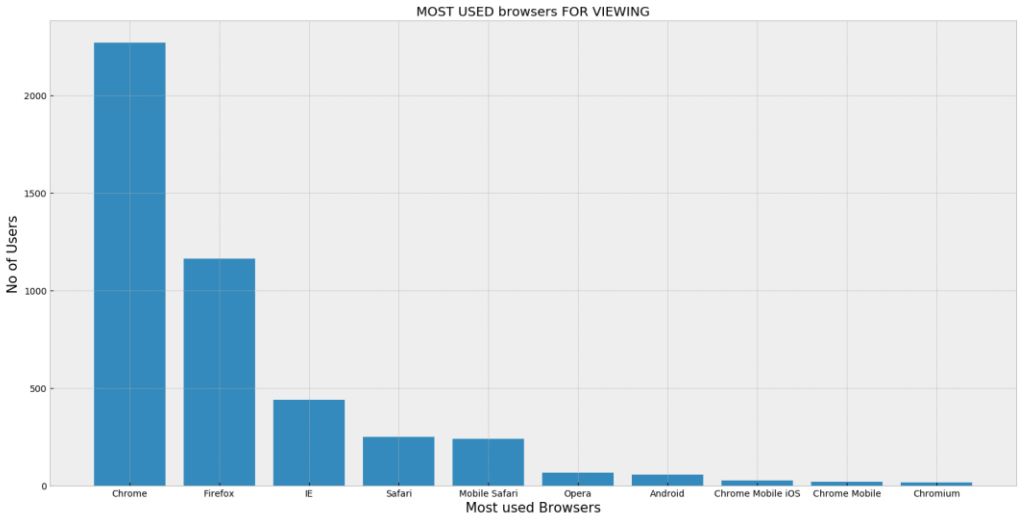
```
"""
Correcting the case mismatch in the dataset to avoid duplicates
"""

MUL = (df.groupby(df['language'].str.split(",",
expand=True)[0].str.upper())['username'].nunique().nlargest(
    10).reset_index(name='count'))
print(MUL)
print(MUL.keys())
```

- All the data analysis work was done, now my job was to find the answers for the questions. I decided to use plots to visualize my data.

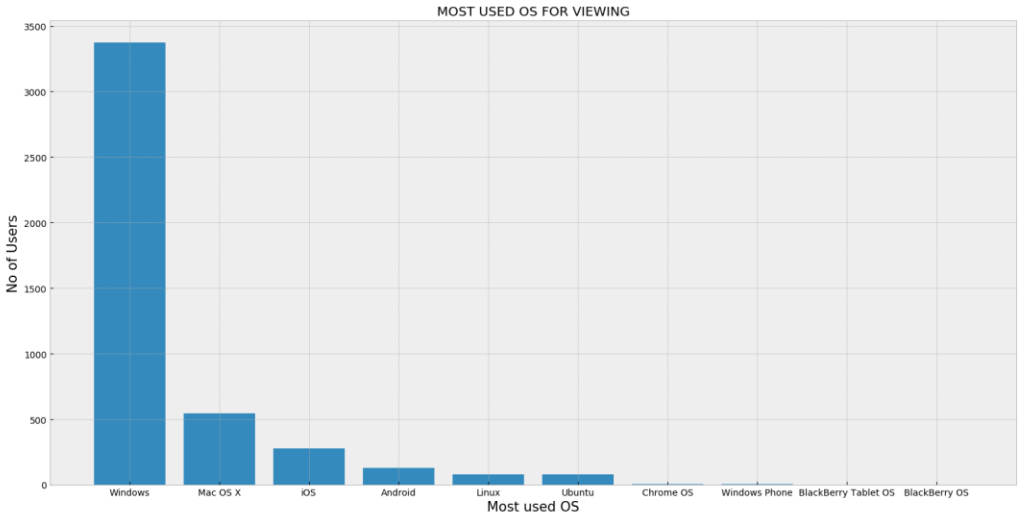
ANSWERS

1) Most used browser?



As, we can see chrome is the most used browser for viewing the videos, followed by Firefox.

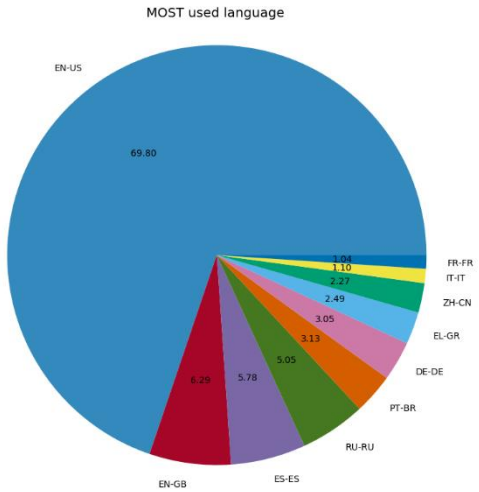
2) Most used Operating systems?



This was an easy guess, windows topping the operating system followed by the Mac OS

3) Most used language?

I tried using the pie to plot my data and below is the output.



Note: The percentage displayed is computed with the other top 10 languages used, the reason was the pie chart was too cluttered when I was trying to plot all the values together.

- The most used language also shows us the popularity of Coursera in different parts of the world.

4) List of Most frequent users?

- I was able to make use of the session column to find out the list of most frequent users, the answer is below.

	username	count
0	0da7bb68bf6f29d10c5476af42de1f2c9c6ba153	1060
1	d333771a15bba985586d6ec53db437855b2f1cc7	979
2	756817966ea3918aa14ed2b50b2184cbbc2ccb81	731
3	7141d9efcbea448b931eea78c0af123da610a3b1	707
4	d851f993a4d0a439cd1b16a1968523b5b4f34d40	664
5	910a0e6309bdf253ba7a33e9e6a3fe14db7f6e74	584

- As, the username is hidden for security purposes, I could not make it look beautiful in the plots, and hence I decided to leave it in the text format.