Overview

In this practical we were required to write python code to fulfill the following requirements related to data analysis with pandas:

1. Check the consistency of the initial (raw) data.

2. In case of any inconsistencies, output refined data for further analysis.

3. Provide an executable Python script to automate the two steps above.

4. Implement unit tests to check at least some of the auxiliary functions.

5. Carry out certain data analysis and visualisation tasks.

6. Provide an executable Python script to regenerate all images and save them in a directory.

7. Provide a reproducible Jupyter notebook combining your report and data analysis.

8. Organise code with minimal duplication for its reuse in Jupyter notebook(s) and Python scripts.

As well as completing the following additional requirements:

Easy: Analyse applications used to send tweets.

Easy: Extend the descriptive analysis, for example, by calculating the average number of times each user being retweeted and the average number of times each user being replied.

Medium: Analyse patterns of user activity over the period covered by the dataset

Plus additional custom extensions.

Discussion about Refining

We removed redundant columns and removed rows that shouldn't be empty. We then check if the columns that are only supposed to contain integers contain any other type of data, if so, remove those rows. Moreover, we checked that the data in the time column is following the correct format dd/mm/yyyy hh: mm. If it isn't in the correct format, remove that row. Afterward, we removed any useless columns from the data frame. We then convert all types of English into one general type en. Lastly, we output it as a CSV file.

In [1]:

```
import pandas as pd
import csv
```

In [2]:

```
df=pd.read_csv("../data/CometLandingRefined.csv")
```

In [3]:

```
df
```

Out[3]:

	id_str	from_user	text	time	user_lang	in_reply_to_us
0	5.409304e+17	amika0078788556	RT @VersaTechnology: Congratulations @Philae20...	05/12/2014 18:07	en	
1	5.409300e+17	ChrisDMarshall	CometWatch 2 December Â» Rosetta - ESA's comet...	05/12/2014 18:05	en	
2	5.409300e+17	MHuuskoL	RT @EUCouncil: After the #CometLanding - Astro...	05/12/2014 18:05	en	
3	5.409293e+17	SaraGomezAranci	RT @EUCouncil: After the #CometLanding - Astro...	05/12/2014 18:03	fr	
4	5.409292e+17	CBCDay6	RT @shaunmajumder: Feels good to be the @CBCDa...	05/12/2014 18:02	en	
...	...	...	...	...	...	
77047	5.324601e+17	ABForScience	This means that the actual landing will be ar...	12/11/2014 09:09	en	
77048	5.324601e+17	atieyK	RT @ObservingSpace: Weâ€™ve been waiting 10 ye...	12/11/2014 09:09	en	
77049	5.324601e+17	j0nny5	RT @maxplanckpress: Accomazzo (flight director...	12/11/2014 09:09	en	
77050	5.324601e+17	nsentse	7 hours of waiting #CometLanding	12/11/2014 09:09	en	
77051	5.324601e+17	grery92	RT @dsdanyds: TopTrendIT: TT ITALIA 09:59\n1.#...	12/11/2014 09:09	it	

77052 rows × 15 columns

In [4]:

```
len(df)
```

Out[4]:

```
77052
```

In [5]:

```
df.dtypes
```

Out[5]:

```
id_str                        float64
from_user                      object
text                           object
time                           object
user_lang                      object
in_reply_to_user_id_str       float64
in_reply_to_screen_name        object
from_user_id_str              float64
in_reply_to_status_id_str     float64
source                         object
profile_image_url              object
user_followers_count          float64
user_friends_count            float64
status_url                     object
entities_str                   object
dtype: object
```

In [6]:

```
df.sort_values(by='time', ascending=True)
```

Out[6]:

	id_str	from_user	text	time	user_lang	in_reply_to_use
1629	5.392087e+17	jorgerojas00	RT @UniversoCnocido: En una sola imagen todos ...	01/12/2014 00:05	es	
1628	5.392089e+17	jorgerojas00	RT @UniversoCnocido: .@Philae2014 ya estÃ¡ seg...	01/12/2014 00:06	es	
1627	5.392089e+17	x5_dc	RT @ScienceNews: Good night, Philae. We may se...	01/12/2014 00:06	en	
1626	5.392089e+17	x5_dc	RT @SNStudents: This photo shows the primary l...	01/12/2014 00:06	en	
1625	5.392091e+17	x5_dc	RT @ScienceNews: Philae lander sent in a surpr...	01/12/2014 00:07	en	
...	...	...	...	...	...	
1654	5.391976e+17	haileysecretary	RT @heyyouapp: â™¡ http://t.co/2XIG5hzBTP 584 ...	30/11/2014 23:21	en	
1642	5.391976e+17	gg_rainey	RT @heyyouapp: â™¡ http://t.co/2XIG5hzBTP 584 ...	30/11/2014 23:21	en	
1648	5.391976e+17	philominax	RT @heyyouapp: â™¡ http://t.co/2XIG5hzBTP 584 ...	30/11/2014 23:21	en	
1631	5.392013e+17	diuk37	RT @davidshukmanbbc: Interesting new take on #...	30/11/2014 23:36	en	
1630	5.392047e+17	ben_economics	RT @rjmlaird: My @ESA_Rosetta t-shirts have ar...	30/11/2014 23:49	fr	

77052 rows × 15 columns

In [7]:

```
df['time']=pd.to_datetime(df['time'], dayfirst=True)
```

In [8]:

```
day=df.groupby(df['time'].dt.date)
```

In [9]:

```
day.size()
```

Out[9]:

```
time
2014-11-12    73094
2014-11-26      396
2014-11-27      485
2014-11-28      700
2014-11-29      416
2014-11-30      331
2014-12-01      591
2014-12-02      463
2014-12-03      303
2014-12-04      195
2014-12-05       78
dtype: int64
```

In [10]:

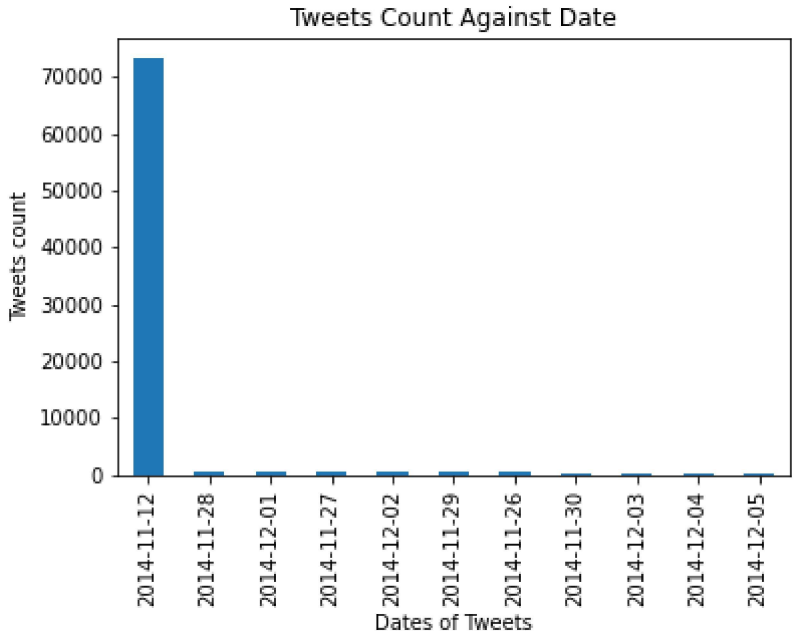
```
import matplotlib.pyplot as plt
%matplotlib inline
```

Graphs to show tweet activity over time.

First, a bar chart:

In [11]:

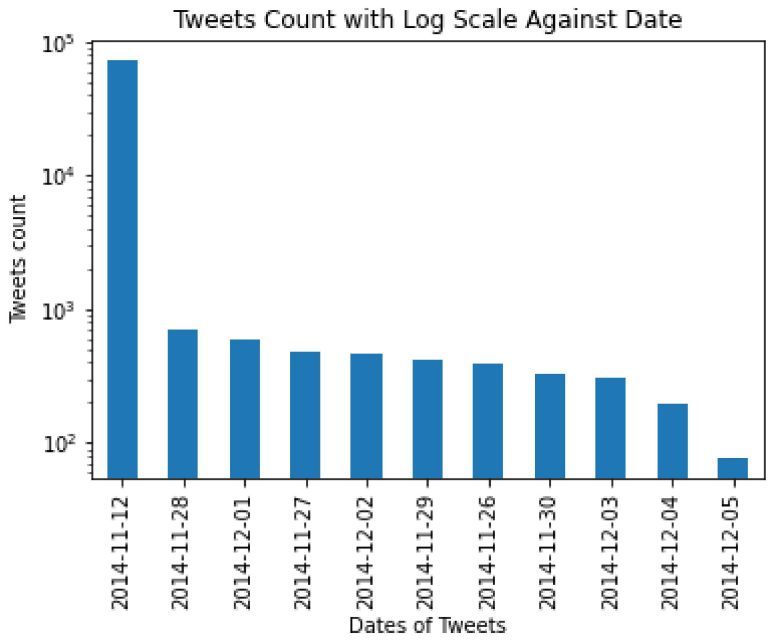
```
df['time'].dt.date.value_counts().plot(kind="bar").set(xlabel='Dates of Tweets', ylabel='Tw
plt.show()
```



Let's get a better look with a log scale:

In [12]:

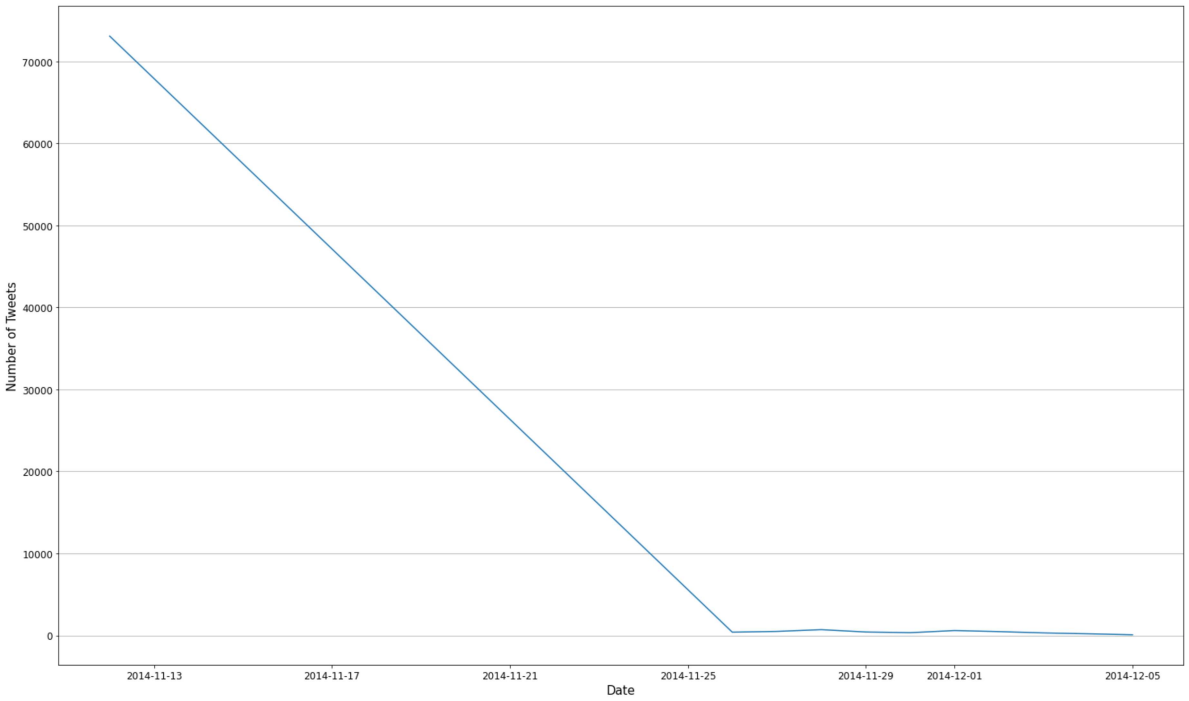
```
df['time'].dt.date.value_counts().plot(kind="bar").set(yscale='log', xlabel='Dates of Tweet
plt.show()
```



Now a line graph:

In [13]:

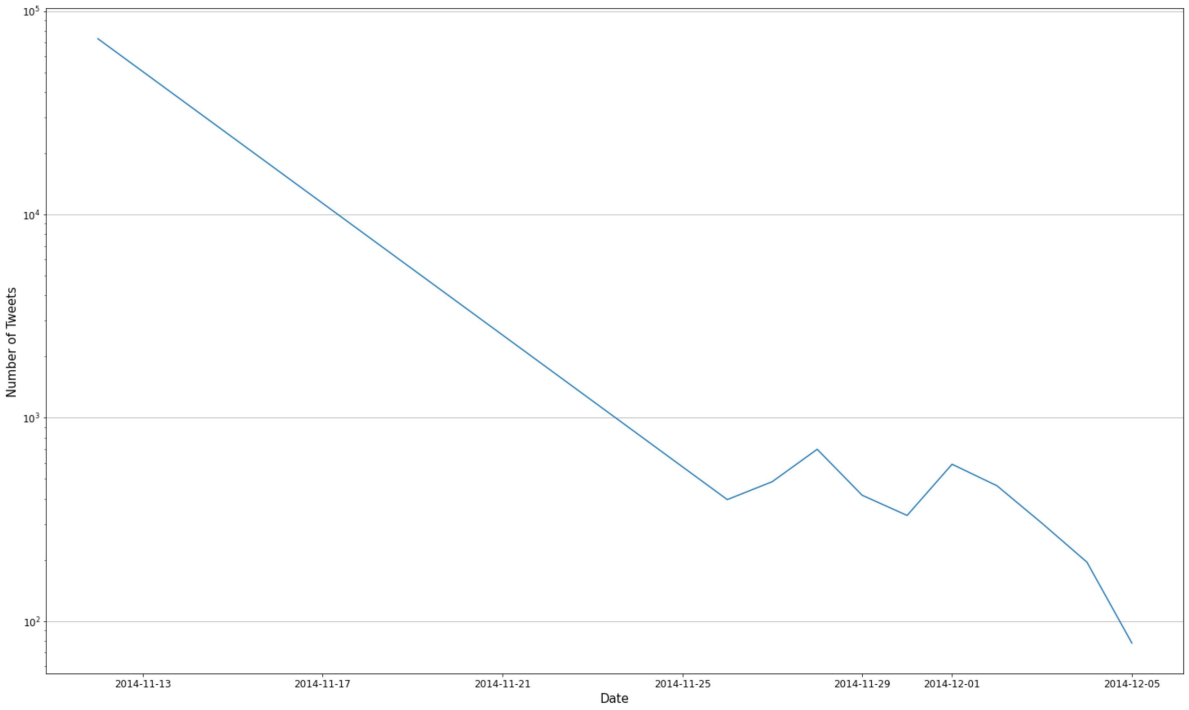
```
df.sort_values(by='time', ascending=True)
df['time']=pd.to_datetime(df['time'], dayfirst=True)

df['time'].dt.date.value_counts().plot(figsize=(25,15))
plt.grid(axis = 'y')
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.xlabel("Date", fontsize=15)
plt.ylabel("Number of Tweets", fontsize=15)
plt.show()
```



Once again, let's get a better look by using a log scale on the y-axis

In [14]:

```
df['time'].dt.date.value_counts().plot(figsize=(25,15), logy=True)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.xlabel("Date", fontsize=15)
plt.ylabel("Number of Tweets", fontsize=15)
plt.grid(axis = 'y')
plt.show()
```



Check number of different (unique) users

In [15]:

```
# https://stackoverflow.com/questions/36106490/how-to-get-unique-values-from-multiple-colum
# By Yaakov Bressler (Last accessed date: 08-04-2022)
users=df.groupby(df['from_user_id_str']).agg(['unique'])
unique_users = len(users.index)
```

Number of replies

In [16]:

```
df_replies=df[['in_reply_to_user_id_str', 'in_reply_to_screen_name']]
df_replies=df_replies.dropna(subset=['in_reply_to_user_id_str', 'in_reply_to_screen_name'],
```

In [17]:

```
len(df_replies.index)
```

Out[17]:

1718

Lets have a look at what languages the tweets are in:

In [18]:

```
#Only display languages that appear frequently, lets say more than 400 times.
df2 = df.groupby('user_lang').filter(lambda x : len(x)<400)
lst = df2['user_lang'].tolist()
lst = list(dict.fromkeys(lst))
lst
for ln in lst:
    df['user_lang'].replace(ln, 'other', inplace=True)
```

In [19]:

```
lang=df.groupby(df['user_lang'])
```

In [20]:

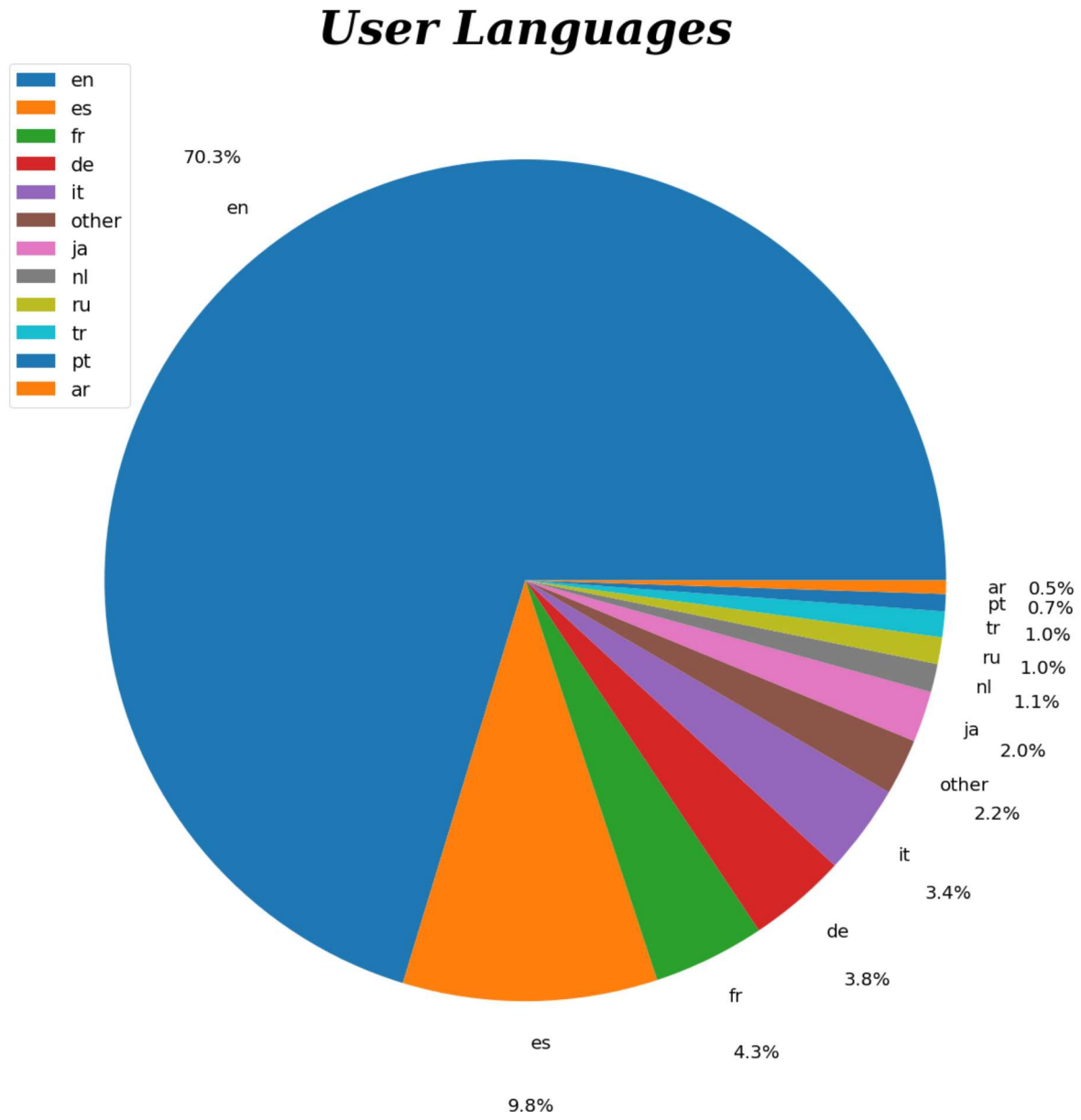
```
lang.size()
```

Out[20]:

```
user_lang
ar          401
de         2914
en        54166
es         7534
fr         3309
it         2655
ja         1508
nl          838
other      1669
pt          508
ru          791
tr          759
dtype: int64
```

In [21]:

```
csfont = {'fontname':'Serif'}
df['user_lang'].value_counts().plot(kind="pie", figsize=(21,21), fontsize=20, autopct='%1.1
plt.xlabel("")
plt.ylabel("")
plt.title(label="User Languages", fontsize=50, fontstyle="italic", **csfont, fontweight="bo
plt.legend(loc='upper left',fontsize=21)
plt.show()
```



We can also determine the frequency of hashtags

In [22]:

```
import json
import re
df = pd.read_csv("../data/CometLandingRefined.csv")['entities_str']
df.reset_index()
```

Out[22]:

	index	entities_str
0	0	{"hashtags":[{"text":"Philae","indices":[49,56...
1	1	{"hashtags":[{"text":"CometLanding","indices":...
2	2	{"hashtags":[{"text":"CometLanding","indices":...
3	3	{"hashtags":[{"text":"CometLanding","indices":...
4	4	{"hashtags":[{"text":"MiniMansbridge","indices...
...	...	...
77047	77047	{"hashtags":[{"text":"CometLanding","indices":...
77048	77048	{"hashtags":[{"text":"cometlanding","indices":...
77049	77049	{"hashtags":[{"text":"CometLanding","indices":...
77050	77050	{"hashtags":[{"text":"CometLanding","indices":...
77051	77051	{"hashtags":[{"text":"GUERRIERO","indices":[44...

77052 rows × 2 columns

In [23]:

```
lst = []
for index in range(df.shape[0]):
    results = re.findall(r"{\"text\":\".+\",", str(df.iloc[index]))
    for i in results:
        if(i.startswith('{\"text\":\"') and i.endswith("\",")):
            ents=i.split(",")
            ents=filter(lambda x: "text" in str(x), ents)
            for j in ents:
                lst.append(j)
```

In [24]:

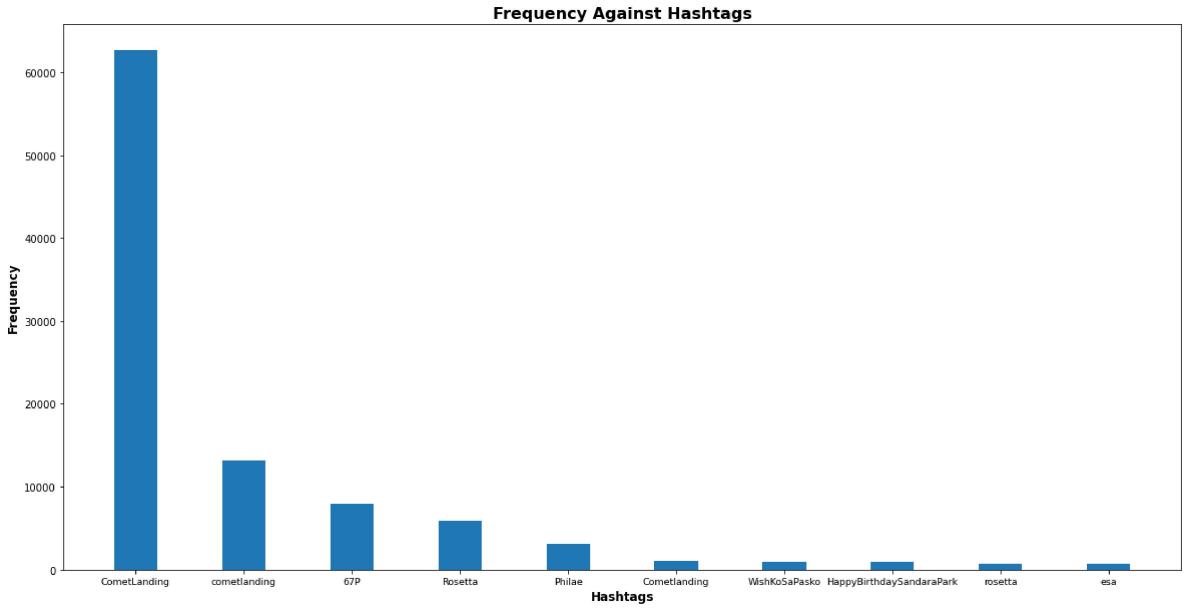
```
cleanlst = []
lst=filter(lambda x: "{\"text\":" in str(x), lst)
for i in lst:
    lst2 = i.split(":")
    cleanlst.append(lst2[1].replace("\"", ""))
print('Total number of hashtags: ' + str(len(cleanlst)))
```

```
Total number of hashtags: 115113
```

iterate down the rows take final column code that extract and take top 10

In [25]:

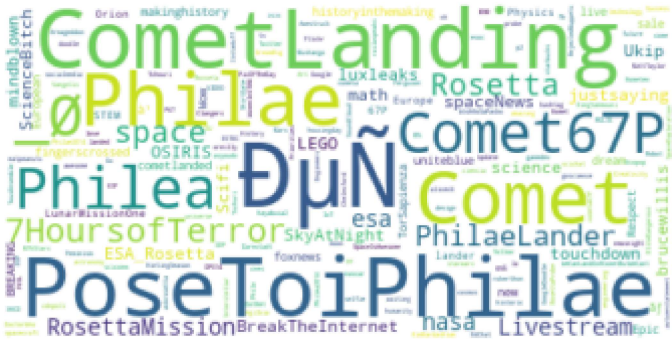
```
from collections import Counter
from itertools import chain
hashtag_counts=Counter(cleanlst)
plt.figure(figsize=(20,10))
hashtag_plot=plt.bar(*zip(*hashtag_counts.most_common()[:10]), width=0.4) # Getting the top
plt.xticks(fontsize=9.5)
plt.xlabel("Hashtags", fontsize=12, fontweight="bold")
plt.ylabel("Frequency", fontsize=12, fontweight="bold")
plt.title("Frequency Against Hashtags", fontsize=16, fontweight="bold")
plt.show()
```



Just for fun - lets create a wordcloud for hashtag content

In [26]:

```
from wordcloud import WordCloud
# removing duplicates from cleanlst
unique_cleanlst = list(dict.fromkeys(cleanlst))
text_for_word_cloud = ' '.join(unique_cleanlst)
word_cloud = WordCloud(collocations=False, background_color='white').generate(text_for_word
plt.imshow(word_cloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



In [27]:

```
df = pd.read_csv("../data/CometLandingRefined.csv")
size_all = len(df)
#Remove all retweets
def retweet(inp):
    if(str(inp).startswith("RT @")):
        return True
    else:
        return False
df=df[df['text'].map(retweet) == False]
num_retweets = size_all - len(df)
```

In [28]:

```
num_retweets
```

Out[28]:

```
59857
```

In [29]:

```
df_replies=df[['in_reply_to_user_id_str', 'in_reply_to_screen_name']]
df_replies=df_replies.dropna(subset=['in_reply_to_user_id_str', 'in_reply_to_screen_name'],
replies = len(df_replies.index)
```

In [30]:

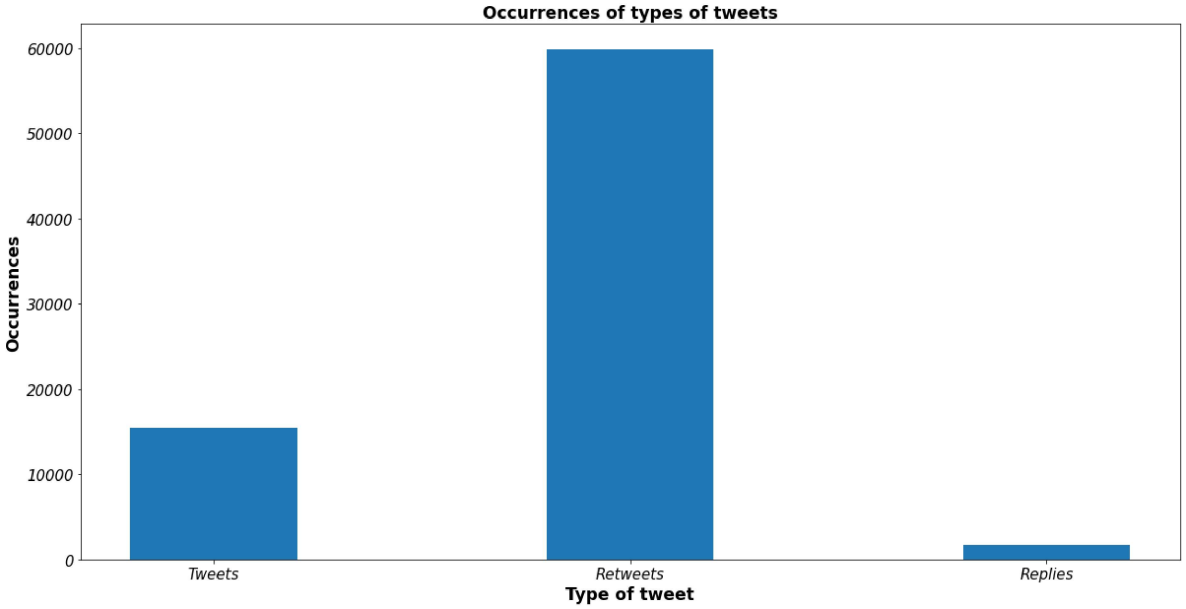
```
standard_tweets = len(df) - replies
standard_tweets
```

Out[30]:

```
15507
```

In [31]:

```
data = {'Tweets':standard_tweets, 'Retweets':num_retweets, 'Replies':replies}
types = list(data.keys())
values = list(data.values())
plt.figure(figsize=(20,10))
plt.bar(types, values, width=0.4)
plt.xticks(fontsize=15, fontstyle='italic')
plt.yticks(fontsize=15, fontstyle='italic')
plt.xlabel('Type of tweet', fontsize=17, fontweight='bold')
plt.ylabel('Occurrences', fontsize=17, fontweight='bold')
plt.title('Occurrences of types of tweets', fontsize=17, fontweight='bold')
plt.show()
```



In [32]:

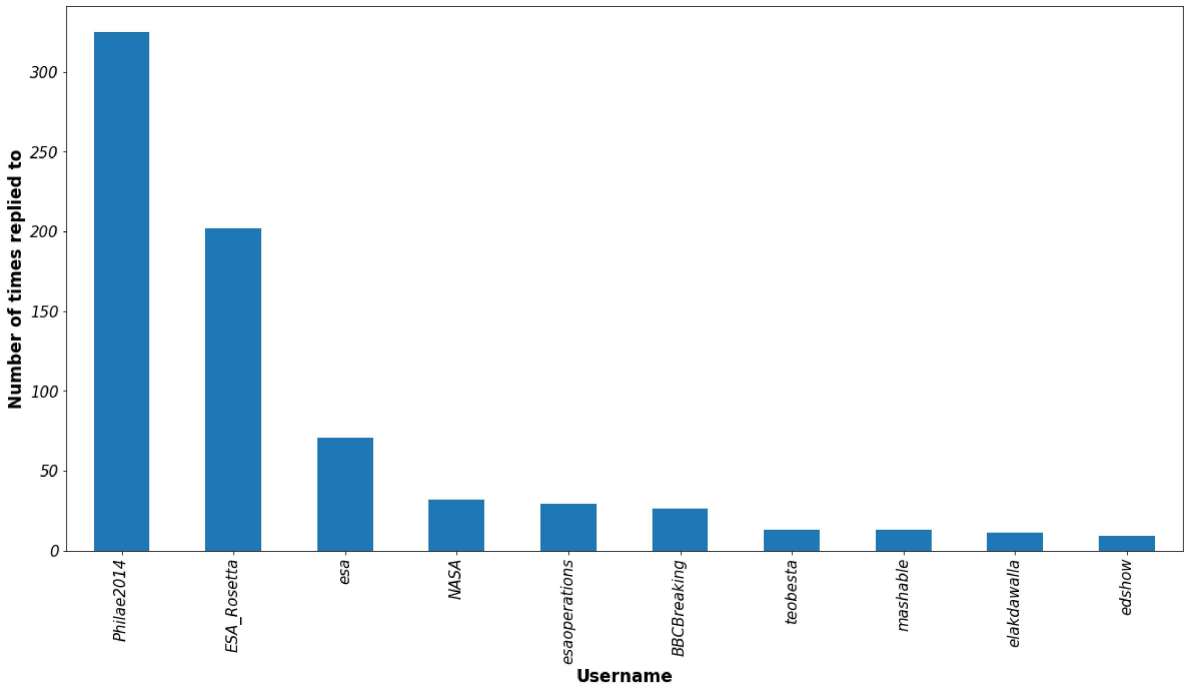
```
print("Average number of tweets per user: " + str(standard_tweets/unique_users))
print("Average number of retweets per user: " + str(num_retweets/unique_users))
print("Average number of replies per user: " + str(replies/unique_users))

print("\nAverage number of tweets of any kind per user: " + str(size_all/unique_users))
```

```
Average number of tweets per user: 0.3093233862602729
Average number of retweets per user: 1.1939878720178727
Average number of replies per user: 0.03367110827415623

Average number of tweets of any kind per user: 1.536982366552302
```

Top 10 most replied to users:

In [33]:

```
repl = df['in_reply_to_screen_name'].value_counts()[:10]

plt.figure(figsize=(20,10))
plt.xticks(fontsize=15, fontstyle='italic')
plt.yticks(fontsize=15, fontstyle='italic')
plt.xlabel('Username', fontsize=17, fontweight='bold')
plt.ylabel('Number of times replied to', fontsize=17, fontweight='bold')
repl.plot(kind='bar')
plt.show()
```



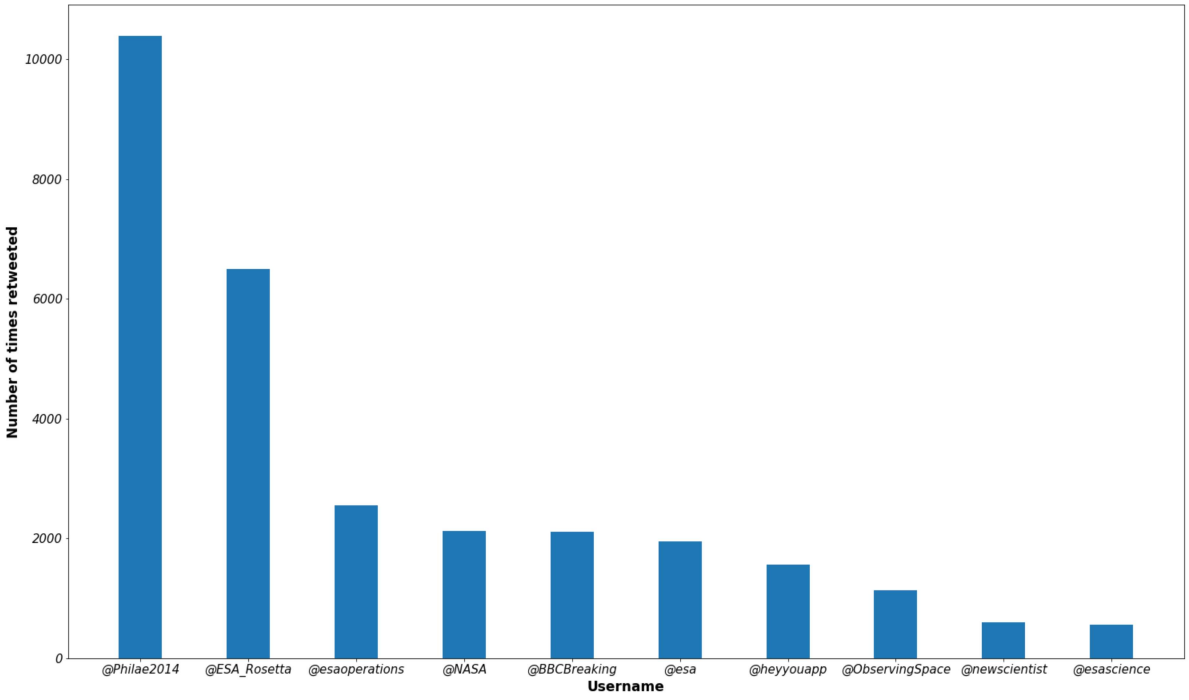
Now let's find out which users were retweeted the most (Top 10):

In [34]:

```
#Only keep retweets
df = pd.read_csv("../data/CometLandingRefined.csv")
size_all = len(df)
#Remove all retweets
def retweet(inp):
    if(str(inp).startswith("RT @")):
        return True
    else:
        return False
df=df[df['text'].map(retweet) == True]
num_retweets = len(df)
```

In [35]:

```
usrs = []
usrsclean = []
for i in range(num_retweets):
    txt_vals = str(df.loc[:, 'text'].values[i]).split(" ")
    usrs.append(txt_vals[1])
for strng in usrs:
    usrsclean.append(strng[:-1])
usrsclean

rt_counts=Counter(usrsclean)
plt.figure(figsize=(25,15))
plt.xticks(fontsize=15, fontstyle='italic')
plt.yticks(fontsize=15, fontstyle='italic')
plt.xlabel('Username', fontsize=17, fontweight='bold')
plt.ylabel('Number of times retweeted', fontsize=17, fontweight='bold')
plt.bar(*zip(*rt_counts.most_common()[:10]), width=0.4)
plt.show()
```



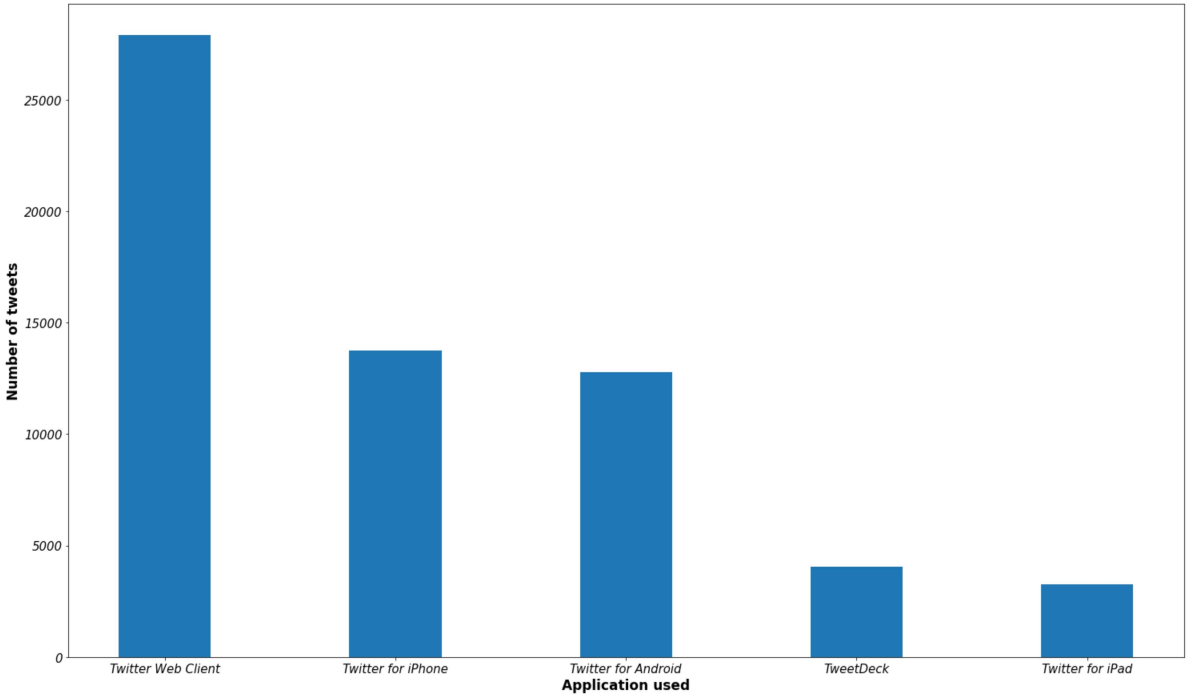
Applications Used for Sending Tweets (Top 5):

In [36]:

```
df = pd.read_csv("../data/CometLandingRefined.csv")
srces = []
for i in range(len(df)):
    html = str(df.loc[:, 'source'].values[i])
    relst = re.findall(">.+<", html)
    for i in relst:
        srces.append(i[1:-1])
```

In [37]:

```
src_counts=Counter(srces)
plt.figure(figsize=(25,15))
plt.xticks(fontsize=15, fontstyle='italic')
plt.yticks(fontsize=15, fontstyle='italic')
plt.xlabel('Application used', fontsize=17, fontweight='bold')
plt.ylabel('Number of tweets', fontsize=17, fontweight='bold')
plt.bar(*zip(*src_counts.most_common()[:5]), width=0.4)
plt.show()
```



Problem Discussion:

We wanted to include the medium to hard requirement, however, we found the data set to be to big to analyse within a reasonable time.

Given more time we would have liked to look more into the harder additional requirements.

Summary of Provenance:

Data Cleaning and Get Images: Kane (Debugged by Auden and Johan)

Data Analysis: Johan and Kane

Testing: Auden

Borrowed code is commented!