Introduction to Python for Digital Humanities and Computational Research

Sections

1. Introduction to Python programming
2. Getting started with Python Programming
3. String Manipuation
4. List in Python
5. Dictionaries in Python
6. Control Structures - For Loop
7. Exanple Project: Data Visualization
8. Example Project: Sentiment Analysis

Makanjuola Ogunleye
Virginia Tech

1. Introduction to Python Programing in the Humanities

One of the goal of this chapter is to introduce you to some Technical tools used within the field of Digital Humanities. One powerful and expressive digital humanities tool is the Python Programming. It allows researchers and users to write code that can do almost all they want computationally. You can write codes that can work on various forms of modalities (texts, images, videos, sounds etc) without any limitation.

Installing Python Programming

To begin, we will need to install Python Programming on our computer. There are at least two ways to install Python.

The first approach is to install it from the official website: https://www.python.org/downloads/

As at the time of writing this book, the latest version is 3.11. However Python regularly update the software version.

The second approach is to search for "install python" on a search engine like Google. Usually the first page that Google returns will be the intslallation page.

Either way you are to download the software to your computer. Based on your operating system (OS) the python official website should provide you with the installation file that aligns your OS.

Once you have downloaded the software, install it and follow the prompts on the screen. Agree to the defaut setup suggested to you.

Once you have finished installing Python, search for python from your search bar to open it. Once you open it, you can start practicing the codes below.

Alternative Way to run python code:

Assuming you have difficulty installing python or your computer's capacity is too small to run python or for any reason you are unable to run python, there are available resources online that can allow you to run python without doing any installation.

1. Replit - https://replit.com/languages/python3
2. Google Colaboratory - https://colab.research.google.com/ (Click on the link and click on new notebook. A new notebook will open with cells where you can write you Python codes)

2. Getting started

Congratulations on reaching this step. So you have successfully installed Python and are ready to run your first code!

Let's get started. Here we will introduce you to your first python line of code. We put the string "Hellow World, welcome to Digital Humanities" in a Python print statement. The print command then outputs the string. We will talk more about this print command in coming section.

```
print("Hellow World, welcome to Digital Humanities")
```

Hellow World, welcome to Digital Humanities

Data type

There are 3 basic data types in Python. They are *Strings*, *Integers* and *Floating* Points. An integer is a digit that does not contain a decimal place, i.e. 1 or 2 or 3. This can be a number of any size, such as 100,001, or 20000. A floating point on the other hand is a digit with a decimal point. E.g 2.0, 3.093, 200.9585757585. Strings are a sequence of characters. The characters can be letters, symbols or digits. What distinguishes a string from an integer or float is the presence of quotation marks. We use quotation marks (“ or ‘) to tell python that this object is a string and not an integer or float.

Intitializing datatypes: In the code below, we show how to initiatialize sting, integers and floats in python. Basically we have an identifier (which is a variable name) and we assign a corresponding value to the identifier.

```
number = 10  #here we are ssigning the integer 10 to variable name number
gpa = 5.0    # similarly here we are assigning 5.0 to gpa
name = "John" #And here we are assignn "John" to name
```

Explicit assignment: Python can either infer the type of a value by just assigning the value to a variable or sometimes you can explicitly define the type. For example, we can explicitly describe of the the followinging data using the code below.

```
number = int(10)
gpa = float(5.0)
name = str("John")
```

Type

You can know the type of a variable by applying the type method on it

```
name = "Daniel"
print(name, "is of type", type(name))
a = 5
print(a, "is of type", type(a))
a = 2.0
print(a, "is of type", type(a))

Daniel is of type <class 'str'>
5 is of type <class 'int'>
2.0 is of type <class 'float'>
```

Single vs double quote: It would be good to mention here that you use either a single quote or double quote to represent a string. Python doesn't really make a difference between both approach is this scenario.

```
name = "Daniel"
name1 = 'Daniel'
print(name, "is of type", type(name))
print(name1, "is of type", type(name1))

Daniel is of type <class 'str'>
Daniel is of type <class 'str'>
```

Identifier and Variables

Python is object oriented. Objects can be thought of as something that has properties. For example, we can think of a person as an object. A person has a name, has an age, has associations, can be a student or a teacher and can perform actions like swimming, walking and singing etc. These are the attributes of the object, Person.

Identifier: A variable used to store a value or object
For example

temperature = 98
print(temperature)

temperature here is an identifier or variable name for the object 98.

Sometimes you can assign an identifier to a value directly like what we did above or you can store the result of an excecuted command to an identifier, like we show below:

Number1 = 4
Number2 = 5
Sum = number1 + number 2
print(sum)

Properties of Identifiers and variable names

- They are Case sensitive. ***temperature*** and ***Temperature*** mean different things.

```
temperature = 20
Temperature = 30
print(temperature)
print(Temperature)
20
30
```

- They can be formed by combining letters, numerals, and underscore

```
temp_lagos_1 = 30
person1_age = 25
```

- They cannot start with numbers. Python will throw an error if you try to start a variable with a number.

```
10_name = "daniel"
SyntaxError: invalid decimal literal
```

- When defining identifiers and variables, always give names that make sense. T = 98, temperature = 98

- Can change. You can always change the values you store in a variable.

```
num = 10
print(num)
num = 11
print(num)
10
11
```

- *Multiple variables can store the same value*

```
x = y = z = "same"
```
print (x)
print (y)
print (z)

```
same
same
same
```

Keywords
Keywords are the reserved words in Python. We cannot use a keyword as a variable name, function name, or any other identifier.

Examples of keywords: *if, continue, else, import,* etc. There are 33 reserved keywords in Python. They also cannot be used as identifiers. E.g continue = 5

Assignment statement
Assignment statements are Python statements that assign a value to a variable. For example, the expression temperature = 98 is an *Assignment statement*

Multiline statement:
Sometimes you want to stack a long statement in multiple lines. You will need to use a forward slash to inform Python that the multiline statement is continuous. See the example below:

```
a = 1 + 2 + 3 +\
   4 + 5 + 6 +\
   7 + 8 + 9

print(a)
45
```

Comments
Sometimes you want to explain what a code does to a future programmer who would read your code. Comments come in handy in documenting your code. We use comments to describe a block of code. Comments in Python can either be single-line or multi-line.

Single line: To describe a single line code by adding a comment to it, we use the # command. Python stops processing the statement after the # command.

```
#print out Hello
print('Hello')
```

Multi-line: If we want to add a comment in multiple lines, we can make use of triple quotes.

```
#This is a comment
"""This is also a
perfect example of
multi-line comments"""
```

Input and output

At the center of programming is the ability to receive input and return output to the screen. Python supports this ability using the "print" command for output and the "input" command for accepting inputs.

Output Examples

```
print('This sentence is output to the screen')
a = 5
print('The value of a is', a)
print(1, 2, 3, 4)
print(1, 2, 3, 4, sep='*')

x = 5
y = 10
print('The value of x is {} and y is {}'.format(x,y))
```

Input examples

```
num = input('Enter a number: ')
print(num)

Enter a number: 10
10
```

Operators

Think of basic arithmetic operations (addition, subtraction, multiplication, and division), you can perform them using Python. In this section, we will go through common operators in Python and how to use them.

Arithmetic operators

Let us initialize two numbers below

```
x = 15
y = 4
```

You can add them together

```
# Output: x + y = 19
print('x + y =',x+y)
x + y = 19
```

You can subtract one from the other

```
# Output: x - y = 11
print('x - y =',x-y)
x - y = 11
```

You can multiply them together

```
# Output: x * y = 60
print('x * y =',x*y)
x * y = 60
```

Division: There are three types of divisions in Python.

1. Regular division (/): This is our regular division in mathematics. This type of division divides the number on the left by the number on the right and returns a floating point number. 5 / 2 will return 2.5

```
x = 15
y = 4
# Output: x / y = 3.75
print('x / y =', x/y)
```

x / y = 3.75

2. Integer division (//): Integer division rounds up the result to a whole number. 5 // 2 will return 2.

```
x = 15
y = 6
# Output: x // y = 2
print('x // y =', x//y)
x // y = 2
```

3. Modulus division (%): This rule of division returns the remainder as the result. 5 % 2 will return 1.

Integer, remainder. Modular division

```
x = 15
```

```
y = 6
# Output: x % y = 3
print('x % y =', x % y)
x % y = 3
```

Comparison operators

Similarly, let us initialize two numbers below

```
x = 10
y = 12
```

We can check whether one is greater than the other

```
# Output: x > y is False
print('x > y is',x>y)
x > y is False
```

We can check whether one is less than the other

```
# Output: x < y is True
print('x < y is',x<y)
x < y is True
```

We can check if the number on the left is equal to the number on the right

```
# Output: x == y is False
print('x == y is',x==y)
x == y is False
```

We can check for inequality

```
# Output: x != y is True
print('x != y is',x!=y)
x != y is True
```

We can also check if one is greater or equal to the other

```
# Output: x >= y is False
print('x >= y is',x>=y)
x >= y is False
```

Similarly, we can also check if one is less than or equal to the other.

```
# Output: x <= y is True
print('x <= y is',x<=y)
x <= y is True
```

Logical operator

Logical operators compare two or more Boolean operators, evaluate them, and return a Boolean. Boolean values are either True or False

Consider two Boolean operators below

```
x = True
y = False
```

We can evaluate a statement that returns True if both of them are True. This is an AND statement

```
print(x and y)
False
```

We can evaluate a statement that returns True if one of the Boolean values is true. This is an OR statement

```
print(x or y)
True
```

A NOT Boolean statement evaluates and returns the opposite of the Boolean statement. For example, NOT x will return the opposite of True which is False.

```
print( not x )
False
```

String Manipulation

Many disciplines within the humanities work on texts which is a collection of strings. Quite naturally programming for the humanities will focus a lot on manipulating texts. In previous examples, you have seen how to define a variable to store a string. E.g name = “Daniel”. In this section, we will show you more of the power of strings and the different things you can do with them in Python.

Concatenation

We have already seen some basic arithmetic in our previous codes. We will show you here that not only numbers, but strings can also be added, or, more precisely, concatenated, together as well:

```
name = "Daniel"
discipline = "Digital Humanities"
print(name + " likes " + discipline)
```

Daniel likes Digital Humanities

In the example above, we concatenated the string name with " likes " and discipline together. We use the plus operator to join strings together in Python.

Indexing

Indexing allows us to access characters at different positions in a string. In python, it is worth noting that positioning starts at zero instead of 1 like we have been accustomed to as human beings. For example, if we want the first character of the name "Daniel", we will do the following.

```
name = "Daniel"
first_letter = name[0]
print(first_letter)
D
```

Supposing we want to extract the last character of the name daniel. Counting from zero, we can tell that the position of the last character is 5. So we can extract it this way:

```
name = "Daniel"
print("Last character: ", name[5])
```

Last character: l

However, using this approach to extract the last character is not very effective. What happens if the string is very long (i.e containing 1 billion characters) and you do not have knowledge of the position number of index of the last element ? What do you do ? Thankfully, their is a way to extract the last character without knowlede of the character index in python. Let's take a look at the example below:

```
name = "Daniel"
print("First approach of extracting last character: ", name[5])
print("A better approach of extracting last character: ", name[-1])
```

First approach of extracting last character: l
A better approach of extracting last character: l

Length
You can know the length of a string using the **len()** function:

```
print("Length of the string name:", len(name))
```
```
Length of the string name: 6
```

You could also use the length function to print out the last letter

```
print("Another way to extract the last letter:", len(name) - 1)
```
Another way to extract the last letter: 5

Slicing
You are already becoming an expert in indexing strings. Now, what if we would like to find out what the last two or three letters of our name are? In Python, we can use what we call slices to perform this operation. To find the first two letters of our name we type in:

```
first_two_letters = name[0:2]
print("First two letters: ", first_two_letters)
```
First two letters: Da

To extract the name without the first two letters, we can do:

```
without_first_two_letters = name[2:]
print("Without first two letters:", without_first_two_letters)
```
Without first two letters: niel

Note here that we didnt specify the end of the index. We just did name[2:]. If you want to extract from the beginning to a certain index you can do, name[:2] without specifying 0 as the starting index. Similarly to extract from an index to the end, you can do name[2:].

If we want to find what the last two letters of the name Daniel is, we can do the following:

```
last_two_letters = name[-2:]
print("Last two letters:", last_two_letters)
```
Last two letters: el

We have provided an image below to help you consolidate the knowledge of indexing and slicing. Take a look at the image and try to internalize what is happening. To extract the word “pyth”, us can use the slice [6:10].

Exercise

1. Can you define a variable `middle_letters` and assign to it all letters of your name except for the first two and the last two?

```
middle_letters = # insert your code here
print(middle_letters)
```

2. Given the following two words, can you write code that prints out the word *humanities* using only slicing and concatenation? (So, no quotes are allowed in your code.)

```
word1 = "human"
word2 = "opportunities"
# insert your code here
```

Additional Data Structures

List

What is a List:
A list is an in built data structure in Python. It is a sequential collection of objects, where the objects can be integers, strings, functions etc, or combination of any of these. We need List to organize data. Elements of a list can be accessed using indexing and different operations can be performed on a list, some of which we will look into in the following sections.

How to initialize a List:
To initialize a list, we use a square bracket [ ]. We put the elements of the the list in the square bracket and assign the list to a variable name. See example below:

```
names = ["Mackay", "Daniel", "Steve", "Blessing", "Jesus", "Juliet"]
prime_numbers = [2, 3, 5, 7, 11, 13]
income_per_hour = [7.5, 10.5, 50.5, 6.6, 7.7]
data = ["income", 25, 7.5]

print(names)
print(prime_numbers)
print(income_per_hour)
print(data)
```

['Mackay', 'Daniel', 'Steve', 'Blessing', 'Jesus', 'Juliet']
[2, 3, 5, 7, 11, 13]
[7.5, 10.5, 50.5, 6.6, 7.7]
['income', 25, 7.5]

Operations on List

Length of a List

Strings and List seems to have a lot of shared operations and methods. For example if you remember to obtain the length of the characters in a string we use the len() function. Similarly in List, we can use the len() function to obtain the length of items in the List.

```
print("Length of list names:", len(names))
print("Length of prime numbers list:", len(prime_numbers))
```

Length of list names: 6

Length of prime numbers list: 6

Slicing

Also, the same way we index and slice a string we can do the same thing with lists. See several examples below:

```
print(names)
print("First element in the list:", names[0])
```

```
print("Second element in the list:", names[1])
print("The last element:", names[5])

print()
print("slicing a range: ",names[1:4])
print("slicing a range: ",names[1:5])
print("The last element:", names[-1])
print("Extracting first 3 elements: ", names[0:3])
print("Extracting first 3 elements: ", names[:3])
print("Extracting the last 2 elements: ", names[-2:]) #Best solution


print()
print("Extracting the last 2 elements:", names[4:]) #this can work too
```

Resulst of indexing and slicing

```
['Mackay', 'Daniel', 'Steve', 'Blessing', 'Jesus', 'Juliet']
First element in the list: Mackay
Second element in the list: Daniel
The last element: Juliet

slicing a range:  ['Daniel', 'Steve', 'Blessing']
slicing a range:  ['Daniel', 'Steve', 'Blessing', 'Jesus']
The last element: Juliet
Extracting first 3 elements:  ['Mackay', 'Daniel', 'Steve']
Extracting first 3 elements:  ['Mackay', 'Daniel', 'Steve']
Extracting the last 2 elements:  ['Jesus', 'Juliet']

Extracting the last 2 elements: ['Jesus', 'Juliet']
```

Appending

The append() function lets you add an element to a list. Let's assume we want to add the next prime number 17 to the list of prime numbers that we defined above. We can use the append function to achieve this.

```
prime_numbers = [2, 3, 5, 7, 11, 13]
print("Initial list before appending: ", prime_numbers, "Length = ",
len(prime_numbers))
prime_numbers.append(17)
print("Final list after appending: ", prime_numbers, "Length = ",
len(prime_numbers))
```

```
Initial list before appending:  [2, 3, 17, 7, 11, 13] Length =  6
Final list after appending:  [2, 3, 17, 7, 11, 13, 17] Length =  7
```

Example Use case of Append

Suppose we have a list of digits:

```
digits = [20, 1, 35, 16, 2, 3, 7, 9, 8]
```

And we want to extract into a new list the digits whose values are less than 10. We can create a new list, and then go through each of the digits in the digits list, checking whether each digit is less than 10. Once we see a digit that is less than 10, we add this digit to the new list that we created using the **append()** function. One question will immediately come to your mind: How do we go through each of the elements of the digit list? We can use one of the control structures in python called **For loop** to perform the iteration.

```
digits_less_than_10 = []    #create an empty list. We will use this to
store digits less than 10
for item in digits:    # for loop control structure. We are going through
each of the digit in digits
 if item < 10:      #We check of the current digit is less than 10
    digits_less_than_10.append(item)      # If it is, we add it using the
append function to the created new list

print()
print("original digits: ", digits)
print("digits less than 10: ", digits_less_than_10)
original digits:  [20, 1, 35, 16, 2, 3, 7, 9, 8]
digits less than 10:  [1, 2, 3, 7, 9, 8]
```

Inserting into a specific position

When you use the append() function to add an item into a list, you will realize that it adds the item to the end of the list. This is how the append function was designed to work. However, sometimes, you want to insert an item into a particular position instead. List have an insert() function that can come to our aid.

```
prime_numbers = [2, 3, 5, 7, 11, 13]
print("original prime number list: ", prime_numbers)
prime_numbers.insert(2, 17) #insert 17 to position 2
print("prime number list after insertion: ", prime_numbers)
```

```
original prime number list:  [2, 3, 5, 7, 11, 13]
prime number list after insertion:  [2, 3, 17, 5, 7, 11, 13]
```

Replacing an item

You can replace an item at a particular position with another item. Note, this is different from inserting. With insert(), the new item is added to a particular position and the previous item at that position is adjusted to another position. When we replace an item, it means what it says. The previous item at the position of replacement is changed to the new item.

```
prime_numbers = [2, 3, 5, 7, 11, 13]
print("original prime number list: ", prime_numbers)
prime_numbers[2] = 17  # 17 will replace the number 5
print("prime number lists after replacement: ", prime_numbers)
```

```
original prime number list:  [2, 3, 5, 7, 11, 13]
prime number lists after replacement:  [2, 3, 17, 7, 11, 13]
```

Extend

Sometimes you have two lists and you want to combine them together to form a single list. The extend() function allows you to extend a list by another list.

```
prime_numbers = [2, 3, 5, 7, 11, 13]
additional_primes = [17, 23, 29]
print("Before extending:", prime_numbers)

prime_numbers.extend(additional_primes)
print("After extending: ", prime_numbers)
Before extending: [2, 3, 5, 7, 11, 13]
After extending:  [2, 3, 5, 7, 11, 13, 17, 23, 29]
```

Just with appending, extending() adds the second list at the end of the first list. If you want to add the list at a particular position, you can try the following:

```
first_five_nums = [1,2,3,4,5]
next_three = [6,7,8]
#Goal: final list = [1,2,3,6,7,8,4,5]
print()
final_list = first_five_nums[:3] + next_three + first_five_nums[-2:]
print(final_list)
After adding second list to a particular position:  [1, 2, 3, 6, 7, 8, 4,
5]
```

Split

Suppose you have a sentence that contains the string below:

```
sentence = "We love digital humanities and we would love to specialize in it"
```

Assuming you want to extract the first word and you try sentence[0], this will extract the first character instead.

```
print(sentence[0])
W
```

We can use split() function in python to split a long string into a list of words. Let's see it in action.

```
print(sentence.split())
['We', 'love', 'digital', 'humanities', 'and', 'would', 'love', 'to', 'specialize', 'in', 'it']
```

And now you can extract the first word from the split sentence.

```
print(sentence.split()[0])
We
```

List of List

Lists are very powerful. They can contain different types of object. They can even be used to store list themselves. When we use a list to store multiple individual lists, these scenario is called nested list. See example below:

```
name_age_lst = [["Mack", 20], ["Austin", 35], ["Ade", 25], ["Dare", 28], ["Godspower", 50]]
print(name_age_lst)
[['Mack', 20], ['Austin', 35], ['Ade', 25], ['Dare', 28], ['Godspower', 50]]
```

Here we initialized a list that contains other lists of people's first names and ages. This can get you thinking about the potential of data structures and how information in the world are being collected and stored using data structures.

Suppose we want to print the age of the first person in our nested list "name_age_lst", we can achieve that using the following code:

```
print("Name of first person in the list:", name_age_lst[0][0])
print("Age of first person in the list:", name_age_lst[0][1])
```

```
Name of first person in the list: Mack
Age of first person in the list: 20
```

Sorting:

In life and in many disciplines, we like to arrange things and sort them in either ascending or descending order. Python has a function to do this for us on a list. There are two ways to sort a list:

Approach 1

```
prime_numbers = [17, 23, 27, 2, 3, 5, 7, 11, 13]
print("Original List", prime_numbers)

sorted_list = sorted(prime_numbers) #first way to sort
print("Sorted List (Approach 1): ", sorted_list)
```

Approach 2

```
#second way to sort
prime_numbers.sort()
print("Sorted List (Approach 2): ", prime_numbers)

#supposing you want to sort in descending order
prime_numbers.sort(reverse = True) #in descending order.
print("Sorted List (Approach 2 in reverse order): ", prime_numbers)
```

```
Original List [17, 23, 27, 2, 3, 5, 7, 11, 13]
Sorted List (Approach 1):  [2, 3, 5, 7, 11, 13, 17, 23, 27]
Sorted List (Approach 2):  [2, 3, 5, 7, 11, 13, 17, 23, 27]
Sorted List (Approach 2 in reverse order):  [27, 23, 17, 13, 11, 7, 5, 3,
2]
```

Removing an item from a List

Let's assume we have a list of book titles defined below:

```
good_reads = ["The Hunger games", "A Clockwork Orange",
              "The pursuit of God", "Water for Elephants",
              "The Shadow of the Wind", "Things fall apart"]
```

Python provides the method `remove()` that acts upon a list and takes as its argument the items we would like to remove and removes them.

```
print(good_reads)
```

['The Hunger games', 'A Clockwork Orange', 'The pursuit of God', 'The Shadow of the Wind', 'Things fall apart']

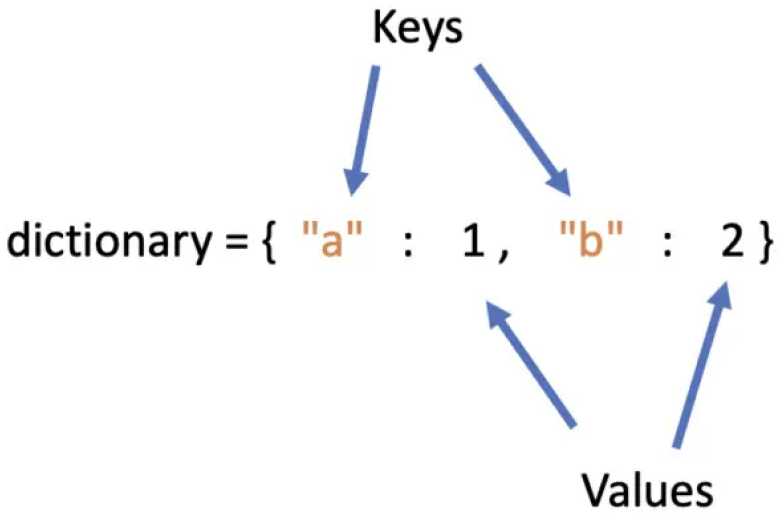
Exercises

1. Define a variable `good_reads` as an empty list. Now add some of your favorite books to it (at least three) and print the last two books you added.
2. Update the `good_reads` collection with some of your own books and give them all a score. Can you print out the score you gave to the first book in the list? (Tip: you can pile up indexes)

Dictionaries

Provides A Dictionary is another great and powerful data structure in Python. It is sometimes called maps or hashmaps. It consists of key and value pairs. The keys are unique and given a key, you can use the key to retrieve a corresponding value. You can think of a dictionary data structure as similar to the dictionaries you have at home. It consists of entries, or keys, that holds a value

Below is an image to graphically describe the dictionary data structure.



Initializing a dictionary

Let’s define a dictionary below:

```
my_dict = {"book": "an object consisting of pages bound together",
           "sword": "a cutting weapon that has a long metal blade",
          "pie": "dish baked in pastry-lined pan often with a pastry top"}
```

Take a close look at the new syntax. Like other objects, we save the dictionary into a variable named `my_dict`. Notice also, the curly brackets and the colons. Keys are located at the left side of the colon; values at the right side.

Accessing a value

To retrieve the value of a given key, we 'index' the dictionary using that key. See an example below:

```
sword_desc = my_dict['sword']    #indexing on my_dict to extract the value
of the key "sword"
print("Sword description: ", sword_desc)
```

Sword description: a cutting weapon that has a long metal blade

---

Inserting an entry

We can insert a new key value pair to an existing dictionary. Let's look at an example below. First we initialize a dictionary names as keys and ages as values.

```
name_age_dict = {"dammy":24,
                 "Michael": 38,
                 "Emma": 22,
                 "Mackay": 15,
                 "Bright": 12,
                 "Yetunde": 10,
                 "Blessing": 20}
```

Supposing we want to insert a new name "Kolade" whose age is 35. We can do it this way:

```
print("Before", name_age_dict)
name_age_dict["Kolade"] = 35      #insertion is happening here. The key
"Kolade" is being added with a value 35.
print("After", name_age_dict)
```

```
Before {'dammy': 24, 'Michael': 38, 'Emma': 22, 'Mackay': 15, 'Bright':
12, 'Yetunde': 10, 'Blessing': 20}
After {'dammy': 24, 'Michael': 38, 'Emma': 22, 'Mackay': 15, 'Bright': 12,
'Yetunde': 10, 'Blessing': 20, 'Kolade': 35}
```

Deleting an entry

We can use the "`del`" command to delete an entry from a dictionary:

```
#Deletion
print("Before", name_age_dict)
del name_age_dict["Michael"]
print("After", name_age_dict)
```

```
Before {'dammy': 24, 'Michael': 38, 'Emma': 22, 'Mackay': 15, 'Bright':
12, 'Yetunde': 10, 'Blessing': 20, 'Kolade': 35}
After {'dammy': 24, 'Emma': 22, 'Mackay': 15, 'Bright': 12, 'Yetunde': 10,
'Blessing': 20, 'Kolade': 35}
```

Length of a dictionary

The same len() function that we used to check the length of a String and the length of a List can be used to check the length of a dictionary.

```
print("Length = ", len(name_age_dict))
```

```
Length =  7
```

Retrieving keys and values

You can also retrieve all the keys of a dictionary or values in form of a list:

```
#List all the keys of a dictionary
print()
print("Keys of the dictionary as a list = ", list(name_age_dict.keys()))

#List all the Values of a dictionary
print()
print("Values of the dictionary as a list = ",
list(name_age_dict.values()))
```

```
Keys of the dictionary as a list =  ['dammy', 'Emma', 'Mackay', 'Bright',
'Yetunde', 'Blessing', 'Kolade']

Values of the dictionary as a list =  [24, 22, 15, 12, 10, 20, 35]
```

Control Structures - Conditionals

Contro Structures - Loops

You have not truly harnessed the power of Python programming if you've not understood and used control structures for conditioning and performing iteration through looping. Here we will teach you the basic syntax of looping and show you some examples. We use loop in python to perform the same operation on a sequence of objects (the iterable). The sequence of objects can be a list, dictionary, or string and the operation can be anything from printing, addition, checking if a condition is true etc.

For example, given a list of words, we would like to know the length of all words, not just one. Now we could technically do this by going through each of the word, getting its length and printing the length, however this would take time and lots of line of code. Imagine if the list contains 1,000,000 words, doing this 1,000,000 times will be infeasible.

Python provides "for" statements that allow us to iterate through any iterable object and perform actions on its elements. The basic format of a "for" statement is:

```
for X in iterable:
 do something
```

Conside the word "humanities". We can print all the letters in the word using for loop.

```
for letter in "humanities":
   print(letter)
```

```
h
u
m
a
n
i
t
i
e
S
```

We iterated through each letters in humanities and printed them starting with the first letter "h" and ending with the last letter "s"

As another example, we can print all the items contained in a list. See example below:

```
names = ["Mackay", "Daniel", "Steve", "Blessing", "Jesus", "Juliet"]
for name in names:
   print("My name is " + name)
```

```
My name is Mackay
```

```
My name is Daniel
My name is Steve
My name is Blessing
My name is Jesus
My name is Juliet
```

Also dictionaries are iterable objects. We can iterate and perform operation on each the key value pairs. The below code will iterate over the keys of the name_age_dict dictionary.

```
name_age_dict = {"dammy":24,
                 "Michael": 38,
                 "Emma": 22,
                 "Mackay": 15,
                 "Bright": 12,
                 "Yetunde": 10,
                 "Blessing": 20}
for name in name_age_dict:
 print(name)
```

```
dammy
Michael
Emma
Mackay
Bright
Yetunde
Blessing
```

We can also iterate over the rows and keys of a dictionary at the same time.

```
for name, value in name_age_dict.items():
   print(name + " is " + str(value) + " years old")
```

```
dammy is 24 years old
Michael is 38 years old
Emma is 22 years old
Mackay is 15 years old
Bright is 12 years old
Yetunde is 10 years old
Blessing is 20 years old
```

When we use items(), at each iteration, a nice pair of the key and the value is returned . In the example above the variable name will loop over the keys of the dictionary, which are the names and the variable value will loop over the respective values which are the ages.

The above way is the most elegant way of looping over dictionaries. Although there is one other method as well. Take a look at it below:

```
for name in name_age_dict:
   print(name + " is " + str(name_age_dict[name]) + " years old")
```

```
dammy is 24 years old
Michael is 38 years old
Emma is 22 years old
Mackay is 15 years old
Bright is 12 years old
Yetunde is 10 years old
Blessing is 20 years old
```

In this example, we iterate over the keys and we use the keys to select the values.

Exercises

1. The function len() returns the length of an iterable item. We can use this function to print the length of each word in the color list. Write your code in the box below:

```
colors = ["yellow", "red", "green", "blue", "purple"]

# insert your code here
```

2. Now write a small program that iterates through the list colors and appends all colors that contain the letter r to the list colors_with_r. (Tip: use colors_with_r.append)

```
colors = ["yellow", "red", "green", "blue", "purple"]
colors_with_r = []
# insert you code here
```

Example Digital Humanities Project Using Python

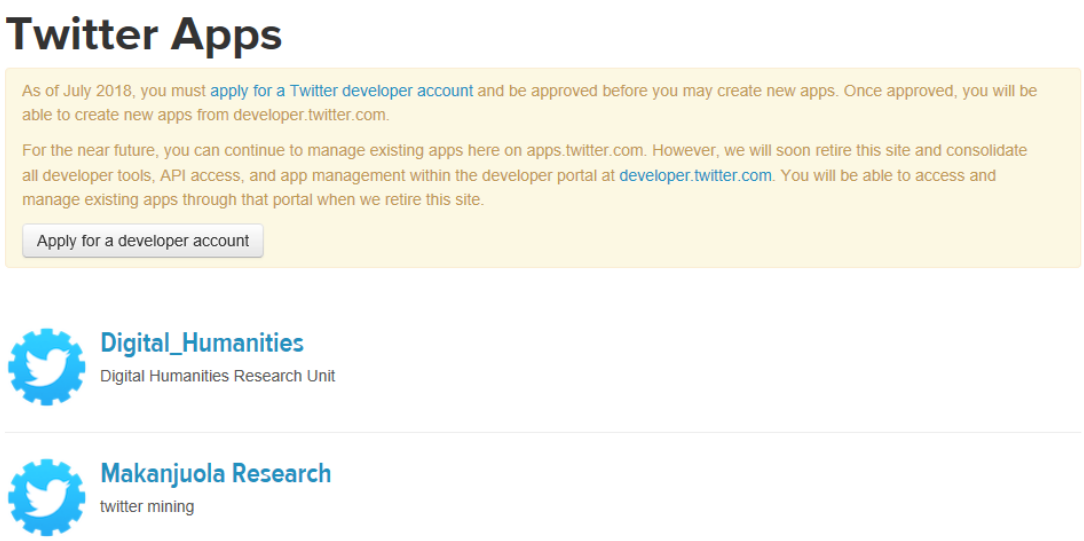
Project Motivation

In this project, our motivation is gotten from a Reality TV show competition in Nigeria called Big Brother Nigeria that took place in 2018. This show has gathered public attraction and has generated a lot of controversies among Nigerians. Perhaps one thing that makes this show popular is its eviction system, it requires the public to vote through Text messages or an online voting platform. Each contestant with the lowest vote during a voting period would be evicted. The interesting thing is that most Nigerians now take to Twitter to share their opinions and perception about their favorite contestant. Thus, tons of thousands of tweets are added daily mostly from Nigerians about this reality show. We want to extract this data from Twitter, analyze them, and check for interesting information about them. [5]

Data Set And Data Collection:

We took advantage of the fact that the show is currently ongoing, as a result, we were able to gather tweets about the show using the hashtag **#bbnaija**. We connected to Twitter's using Twitter streaming API and we were able to gather **580,000 tweets**.

The good thing is that Twitter has a feature that allows developers to download streams of tweets as they are being posted into a database. These features granted us access to Twitter. We used Twitter streaming API to collect the tweets.



By creating a developer account on Twitter, it is possible to extract tons of thousands of tweets. Twitter gives each developer account a unique key and access token. It is with this key and token that the developer interacts with Twitter in downloading tweets.

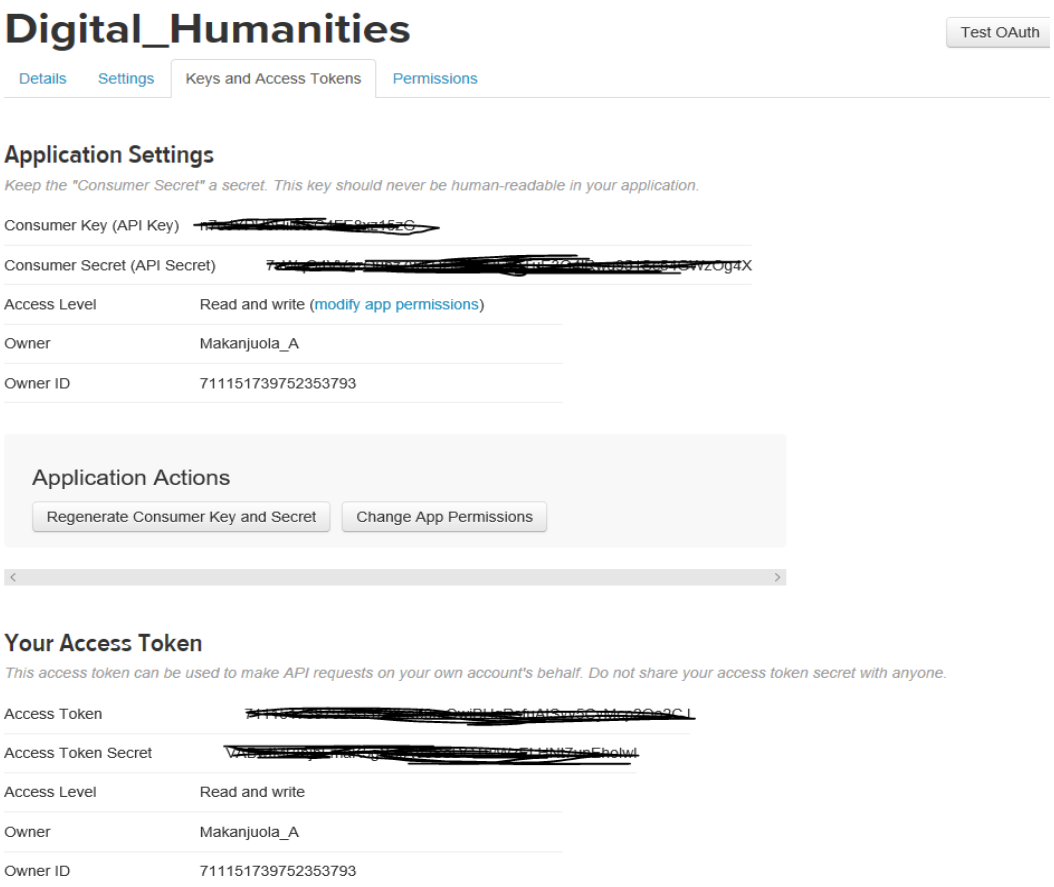
Account creation and keys

You can create a developer account on Twitter using the link below. Once you create an account, you would be provided with an API key, API Secret key, Access Token and Access Token secret. APIs are Application Programming Interfaces. They are set of keys by which we can access an application. You would need these keys to be able to retrieve texts from Twitter.

https://developer.twitter.com/en/portal/dashboard

It is worth noting that access to Twitter has changed over the years. Now in 2023, Twitter provides three (3) types of access to their API. They are Essential, Elevated, and Academic. Academic access provides the best experience - through the access you can download up to 10 million tweets. However, you would need to fill out an application indicating that you need an Academic access. The Essential access allows you to download up to 500,000 tweets and it is free. It can be a starting point for a beginner to explore the possibilities of retrieving tweets from Twitter.

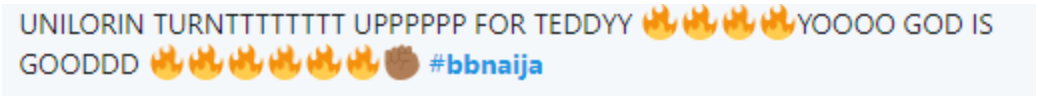
After creating a developer account and generating keys, you should have a tab that has a similar look to this:



Data Preprocessing:

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. In a similar fashion, data extracted from Twitter usually come raw and unprocessed. A typical raw Twitter tweet contains Emoticons, Irregular use of upper and lower-case letters in a sentence, HTML links, stop words, and lots of punctuation, etc. For data to be useful for analysis, it needs to undergo a data-cleaning process. Data cleaning or preprocessing is therefore an essential aspect of text analytics. We exploit some Natural language processing packages in python and some other useful python modules such as Lambda functions, Pandas, and NumPy in the data cleaning process.

Below is an example of uncleaned data.



Here is an example of the same data that is now cleaned and useful for analysis:

unilorin turnttttttttt upppppp for teddyy yoooo god is gooddd bbnaija

Our Data cleaning process was in the following stages. We will show codes of these stages in the coming sections

1. **Tweet case transformation:** Here we transformed all the tweet text to lowercase to preserve uniformity in the data.
2. **Emoticons removal:** All emoticons and smileys objects were removed at this stage
3. **Stop words removal**: Stop words are prepositions and conjunctions that have little or no usefulness in Natural Language Processing or Text Mining.
4. **Url Links removal:** We removed urls from the texts becaus they are not needed in textual analysis
5. **Stemming:** We observed some contestants were pronounced in different ways, we try to conserve data by stemming these names as one.
6. **Tokenization:** In text analysis it is sometimes useful to break text into word tokens, this word tokens can be used to calculate frequencies and perform quantitative analysis. We split the whole tweet collection into word tokens.

Objectives:

In this project we will attempt to answer the following research questions:

1. How are tweeters distributed by location?
2. Who are the top contestant by popularity on Twitter?
3. The number of tweets that contain each contestant?
4. Words with the highest correlation with each contestant and generating N-grams?

   .*We generate n-grams using python to group words that have high correlation together.*
5. Who are the Top tweeters and their location?
6. What are the sentiments around each contestant?

Methods:

Python Modules used:

The following modules were used for analysis in python

- *Matplotlib, seaborn* and *word cloud pylab* for creating charts

- *pandas* and *numpy* for operating with Dataframes and Arrays
- *re* for Regular Expressions
- *textblob* and *nltk* for Sentiment analysis and Natural Language Processing

All modules used

Code

```
%matplotlib inline
import matplotlib
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import nltk
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from operator import itemgetter
from wordcloud import WordCloud
from textblob import TextBlob
from pylab import figure, axes, pie, title, show
```

Reading in Data

The extracted Tweets from Twitter were stored in a JSON file. Each tweet's Json dictionary contains information about the tweet, such as creation date, tweet text, location, etc. Retweeters information were also stored in each tweet's JSON file. We read the entire JSON file and we extracted three pieces of information that was needed to answer our research questions. ***tweet text, tweet location, and tweet date***

Code

```
import json
tweetText = []
location = []
date = []
with open('pythonbbnaija.json') as dataIn:
    for row in dataIn:
        try:
            tweet = json.loads(row)
```

```
            tweetText.append(tweet['text'])
            location.append(tweet['user']['location'])
            date.append(tweet['created_at'])
        except:
            continue
```

Creating a dataframe

In order to exploit the power of Python's Pandas library, we created a data frame of *tweet text, tweet location, and tweet date.* This new data frame is a subset of the entire information obtained whenever a tweet is extracted from Twitter. The tweet text is the most important component needed for our analysis.

Code

```
import pandas as pd
twtData = pd.DataFrame({
   'Tweet' : tweetText,
   'location' : location,
   'date' : date,
})
```

User-defined Functions

We defined and implemented some python functions that were useful for our analyis.

1. **emoji_pattern** – This function creates a regular expression pattern for emojis to be later removed during the cleaning process

2. **rePart:** This function creates a dictionary of some regular expression pattern that would be used in replacing some words during the cleaning process

3. **reExpr:** This function creates a pattern that would be used in counting word occurrences as part of the **CountWrd** function

4. **RemoveStopWord:** This function receives a string, sentence, or tweet, splits it, and returns a new string with no stop word. Stop words are prepositions and conjunctions that have little or no usefulness in Natural Language Processing or Text Mining.

5. **RemoveURL:** This function Removes URLs from tweets and returns a string with no URL

6. **ReplacePattern:** Replace similar patterns of words with one single word. Somewhat related to stemming.

7. **CountWrd:** Receives two arguments - A data (could be a list, array, or series) and a search pattern. counts the occurrences of that pattern in the data, then returns the count.

8. **Ngram:** Receives two arguments - A data (could be a list, array, or series) and a keyword. Creates and returns a list of bi-gram of words around the keyword.

9. **GetTweetSentiment:** Receives a string, here a Tweet, the Python module textBlob is then used to get sentiment polarity. Polarity is a metric that measures sentiments or opinions. It ranges between -1 and 1. if polarity is less than 0; This funtion returns a negative sentiment, if polarity is equal to 0: it returns a neutral sentiment. If greater than zero, it returns a positive sentiment.

10. **PlotWordCloud:** This function receives a list of words, concatenates it to a big text, and then returns a word cloud plot of the text.

Code

```
emoji_pattern = re.compile("["
       u"\U0001F600-\U0001F64F"  # emoticons
       u"\U0001F300-\U0001F5FF"  # symbols & pictographs
       u"\U0001F680-\U0001F6FF"  # transport & map symbols
       u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                          "]+", flags=re.UNICODE)

rePart = {
   'cee-c' : r'cee',
   'miracle': r'mira'
}
reExpr = {
   'cee-c' : re.compile(r'cee'),
   'miracle': re.compile(r'mira'),
   'nina': re.compile(r'nina'),
   'lolu': re.compile(r'lolu'),
   'alex': re.compile(r'alex'),
   'anto': re.compile(r'anto'),
   'khloe':re.compile(r'khloe'),
   'tobi' :re.compile(r'tobi')
}

def RemoveStopWord(data):
   word = ''
   for wrd in data.split():
       if wrd not in stopwrd:
           word = word + ' ' + wrd
   return word
def RemoveUrl(wordseries):
```

```
    word = ''
    for wrd in wordseries.split():
        if wrd[:4] != 'http':
            word = word + wrd + ' '
    return word

def ReplacePattern(data, part):
    word = ''
    dataSplit = data.split()
    for i in range(len(dataSplit)):
        m = re.search(rePart[part], dataSplit[i])
        if m:
            dataSplit[i] = part
    for wrd in dataSplit:
        word = word + wrd + ' '
    return word
def CountWrd(data, pattern):
    cnt = 0
    for word in data:
        m = reExpr[pattern].search(word)
        if m:
            cnt+= 1
    return cnt

def Ngram(data, keyword):
    gramLst = []
    for i in range(len(data)):
        if data[i] == keyword :
            if data[i-1] == data[i] or data[i] == data[i+1]:
                continue
            elif data[i-1] == data[i+1]:
                needOnly = data[i-1] + '-' + data[i]
                gramLst.append(needOnly)
            else:
                needLeft = data[i-1] + '-' + data[i]
                needRight = data[i] + '-' + data[i + 1]
                gramLst.append(needLeft)
                gramLst.append(needRight)
    return gramLst
```

```
def GetTweetSentiment(tweet):
        analysis = TextBlob(tweet)
        if analysis.sentiment.polarity > 0:
            return 'positive'
        elif analysis.sentiment.polarity == 0:
            return 'neutral'
        else:
            return 'negative'

def PlotWordCloud(AList):
   text = ' '.join(AList)
   wc = WordCloud(width=800, height=600, margin=5,
              stopwords=[], prefer_horizontal = 1.5).generate(text)

   n = plt.figure(figsize=(10,14), dpi=100)
   plt.imshow(wc, interpolation='bilinear')
   plt.axis('off')
   plt.show()
```

Data Preprocessing

This is our data cleaning stage. Here we transformed tweets text to lower case, removed emoji, removed stop words, and removed URL links. We also observed some contestants were pronounced in different ways, in order not to loose data, we stemmed the different ways pronouncing a contestant as one. For example, Miracle can be pronounced as Mira, Mirac, or Miracle. We combined all of this options as one.

```
stopwrd = set(stopwords.words('english'))

twtData['Tweet'] = twtData['Tweet'].apply(lambda x: x.lower())
twtData['Tweet'] = twtData['Tweet'].apply(lambda x: emoji_pattern.sub(r'',
x))
twtData['Tweet'] = twtData['Tweet'].apply(lambda x: removeStopWord(x))
twtData['Tweet'] = twtData['Tweet'].apply(lambda x: removeUrl(x))
twtData['Tweet']   =   twtData['Tweet'].apply(lambda   x:   replacePattern(x,
'cee-c'))
twtData['Tweet']   =   twtData['Tweet'].apply(lambda   x:   replacePattern(x,
'miracle'))
wordlst = [wrd for tw in twtData['Tweet'] for wrd in tw.split()]
wordseries = pd.Series(wordlst)
```

Findings And Analysis

Data Visualizations

Result 1: How are tweeters distributed by location?

This graph below tells us that Lagos possesses the highest contribution in terms of online engagement during the BBnaija show. In addition, we can tell that other African countries like Kenya, Uganda, Ghana, and Ivory Coast have a good online engagement for the BBnaija show. One of the observations gotten from the data is that most Twitter users leave their locations off. We got a count for such users and categorized them as “NONE”, but we had to filter out that category from the graph since it carries no useful information.

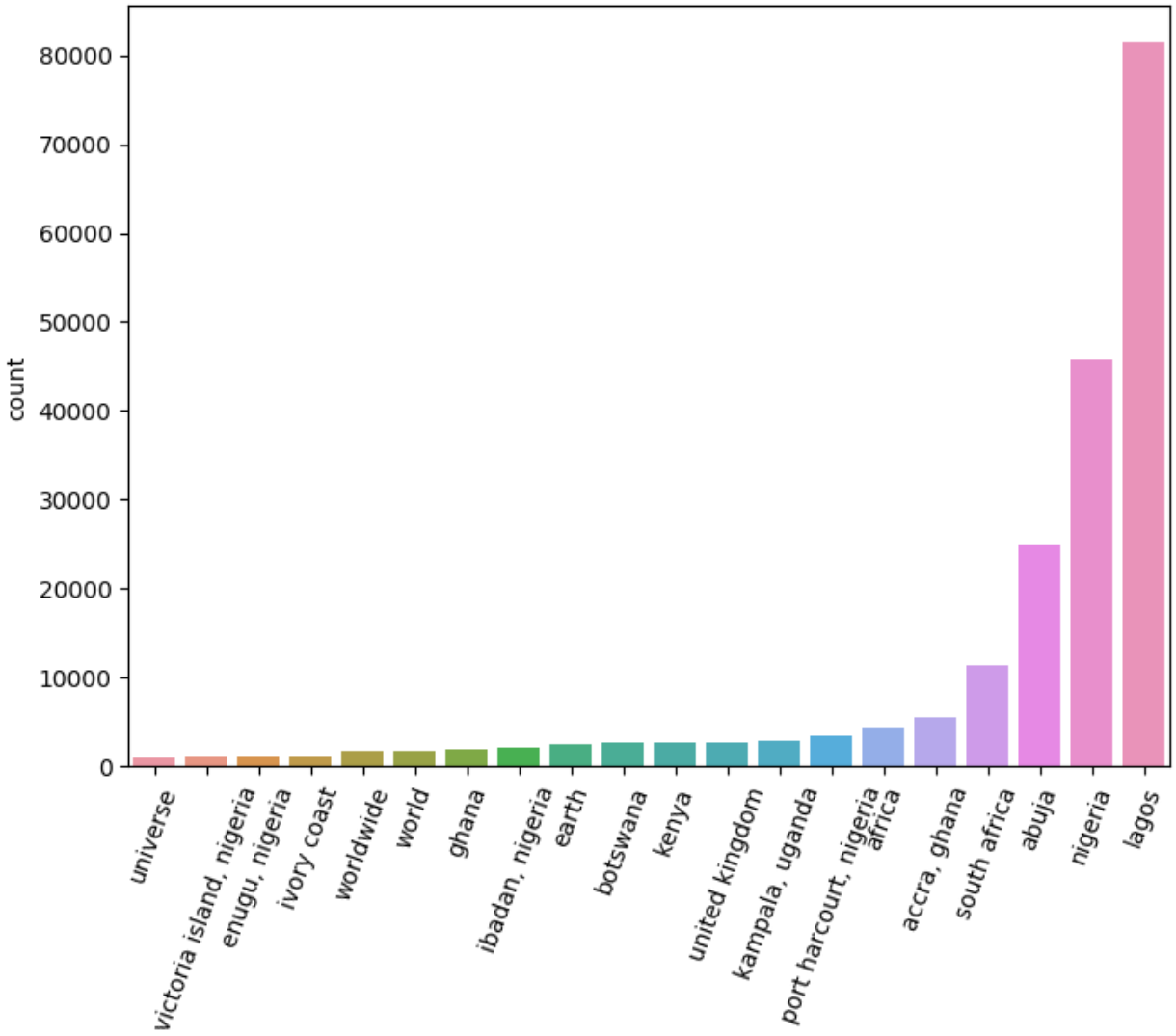
Code

Using the location column in our Data frame, we were able to answer this question, the location series was converted to lowercase. Replicate locations were stemmed as one. And then *sns countplot* was used to plot the distribution of tweeters by location.

```
location = twtData['location'].str.lower().tolist()

for i in range(len(location)):
   if location[i] != None:
       m = re.match(r'lagos|lasgidi', location[i])
             g = re.match(r'pretoria, south africa|johannesburg, south
africa|durban, south africa', location[i])
       n = re.match(r'abuja|federal capital territory, nig', location[i])
              j = re.match(r'england, united kingdom|united kingdom',
location[i])
       k = re.match(r', nigeria|nigeria.|nigeria', location[i])
       u = re.match(r'nairobi, kenya|kenya', location[i])
       if m:
           location[i] = 'lagos'
       if n:
           location[i] = 'abuja'
       if k:
           location[i] = 'nigeria'
       if g:
           location[i] = 'south africa'
       if j:
           location[i] = 'united kingdom'
       if u:
```

```
            location[i] = 'kenya'

locat = [i for i in location if i != None ]
lstDict= {}
for i in locat:
   lstDict[i] = lstDict.get(i, 0) + 1
sortedlist = sorted(lstDict.items(), key=itemgetter(1), reverse = True)
sortedlist

lst2 = [j[0] for j in sortedlist[0:20][::-1]]
lst3 = [item for item in locat if item in lst2]

plt.figure(figsize=(10,6), dpi = 100)
ax = sns.countplot(x = lst3, order = lst2)
ax.set_xticklabels(lst2, rotation=70)
plt.show()
```

Distribution of tweets by Location



Result 2: Who are the top contestants by popularity on Twitter?

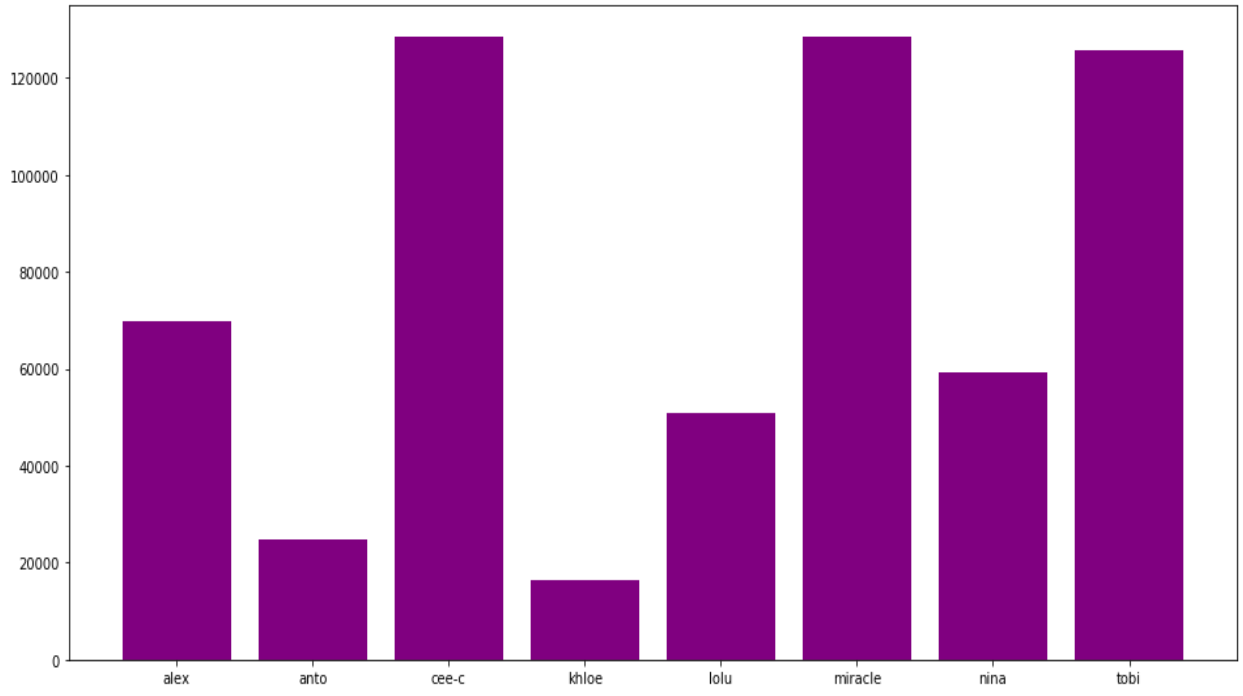
We created a list of contestants. This list was used to generate counts for each contestant using the countWrd function that was defined above. These counts were saved in a list using list comprehension (List comprehension is a technique used to quickly generate a list from a for

loop statement in one line of code). Finally, we plotted a bar chart of Contestants versus name counts using matplotlib. **Cee-C, Miracle, and Tobi take the lead in the most talked about contestant**

```
keys = ['miracle', 'cee-c','alex', 'nina', 'lolu', 'khloe', 'tobi',
'anto' ]
cnt = [countWrd(wordlst,x) for x in keys]
cntDict2 = dict(zip(keys,cnt))
sortedlist = sorted(cntDict2.items(), key=itemgetter(1), reverse = True)

plt.figure(figsize=(16,8), dpi = 100)
ax = plt.bar(cntDict2.keys(), cntDict2.values(), color = 'purple')
plt.show()

pd.DataFrame(sortedlist, columns = ['Names','Counts'])
```

	Names	Counts
0	cee-c	128625
1	miracle	128494
2	tobi	125635
3	alex	69863
4	nina	59147
5	lolu	50859
6	anto	24681
7	khloe	16389

Graphical Display of Top Contestants by Popularity



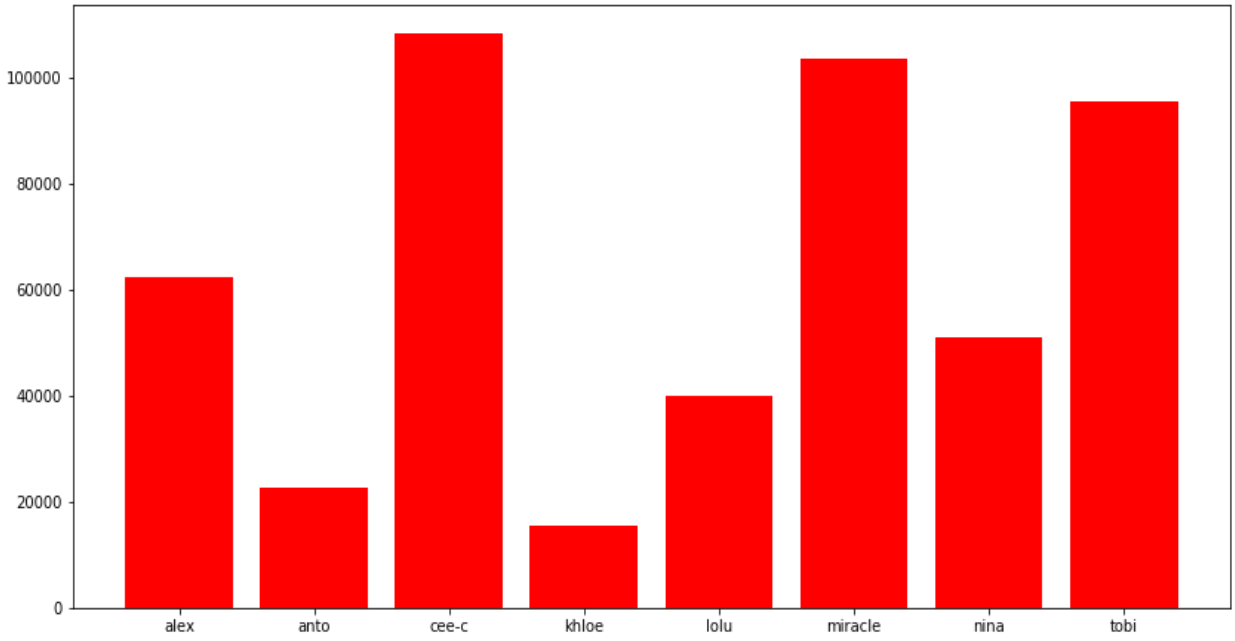
Result 3: Number of tweets that contain each contestant?

We created a list of contestants. This list was used to generate counts for each contestant using the **countWrd** function we defined above. We passed the series of our tweet text into the countWrd function. This counts unique appearances of contestants' names in a tweet. These counts were saved in a list using list comprehension. Finally, we plotted a bar chart of Contestant versus The number of tweets that contain each contestant using matplotlib.

```
keys = ['miracle', 'cee-c','alex', 'nina', 'lolu', 'khloe', 'tobi',
'anto' ]
cnt = [countWrd(twtData['Tweet'],x) for x in keys]

cntDict1 = dict(zip(keys,cnt))
sortedlist = sorted(cntDict1.items(), key=itemgetter(1), reverse = True)

plt.figure(figsize=(14,7))
ax = plt.bar(keys, cnt, color = 'red')
plt.show()

pd.DataFrame(sortedlist, columns = ['Names','Counts'])
```

	Names	Counts
0	cee-c	108143
1	miracle	103604
2	tobi	95517
3	alex	62336
4	nina	51027
5	lolu	40019
6	anto	22519
7	khloe	15559

Graphical Display of Tweets that Contains each Contestant



We can tell from this graph that for every tweet made by Twitter users, Cee-c appears most, followed by Mira and then Tobi. The difference between this result and the previous one is that in this result we counted all tweets that contain each contestant regardless of the number of times the contestant was mentioned per tweet. This tends to give us an idea of the probability of a contestant being mentioned in a tweet

Result 4: Words with the highest correlation with each contestant and generating bi-grams?

In text mining, an n-gram is a phrase or combination of words that may take on a meaning that is different from, or greater than the meaning of each word individually.

Here we created a bigram on words around each contestant as posted by Twitter users to see the words that are associated with them the most.

So basically, an n-gram can give us an attribute of a person. For example, words around miracle reveal he is a people person. As you can see on the Word Cloud; In text mining, a word cloud is a very powerful tool for data visualization. Here we have created a commonality word cloud. Examples of words around him that show he seems to be a people person are *'support miracle', 'voting miracle', 'like miracle', 'miracle fan',* and so on.

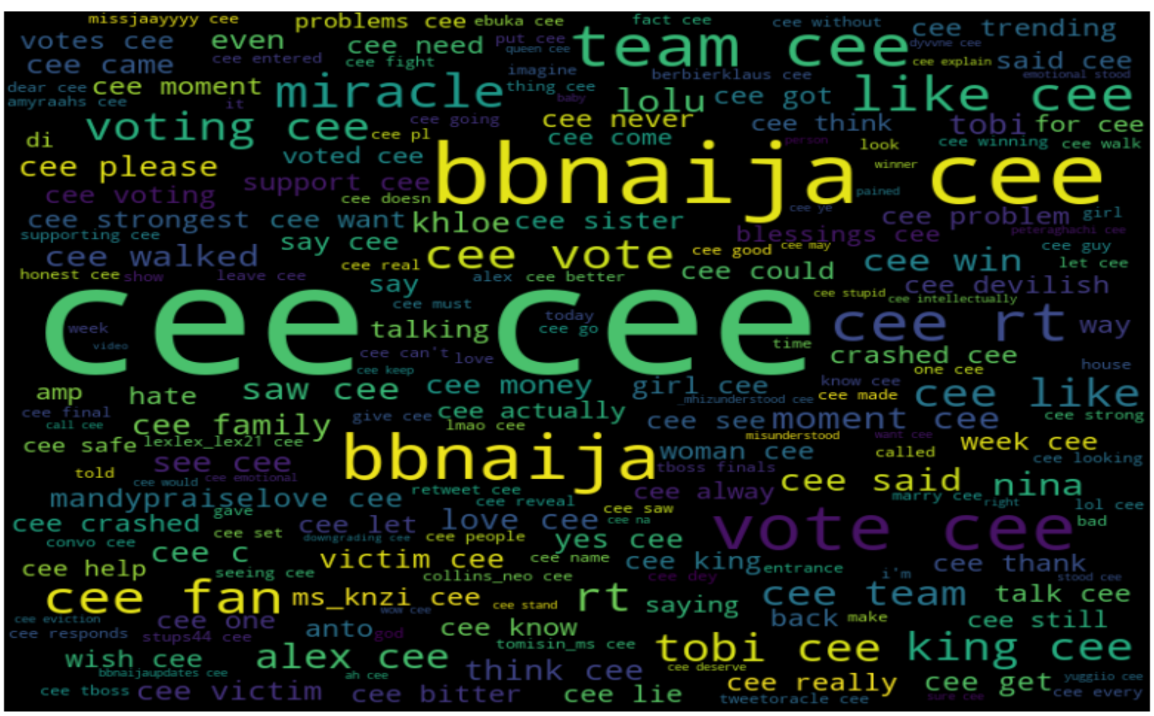
Words around Cee-c reveal she might be an active person. As you can see on the word cloud; *'cee-c talked', 'cee-c can't', 'cee-c fight', 'cee-c crashed'*, and so on.

Words around Nina reveal that she might be a Talking type of person. As can be seen on the word cloud; *'telling nina', 'nina said', 'nina talk'* e.t.c

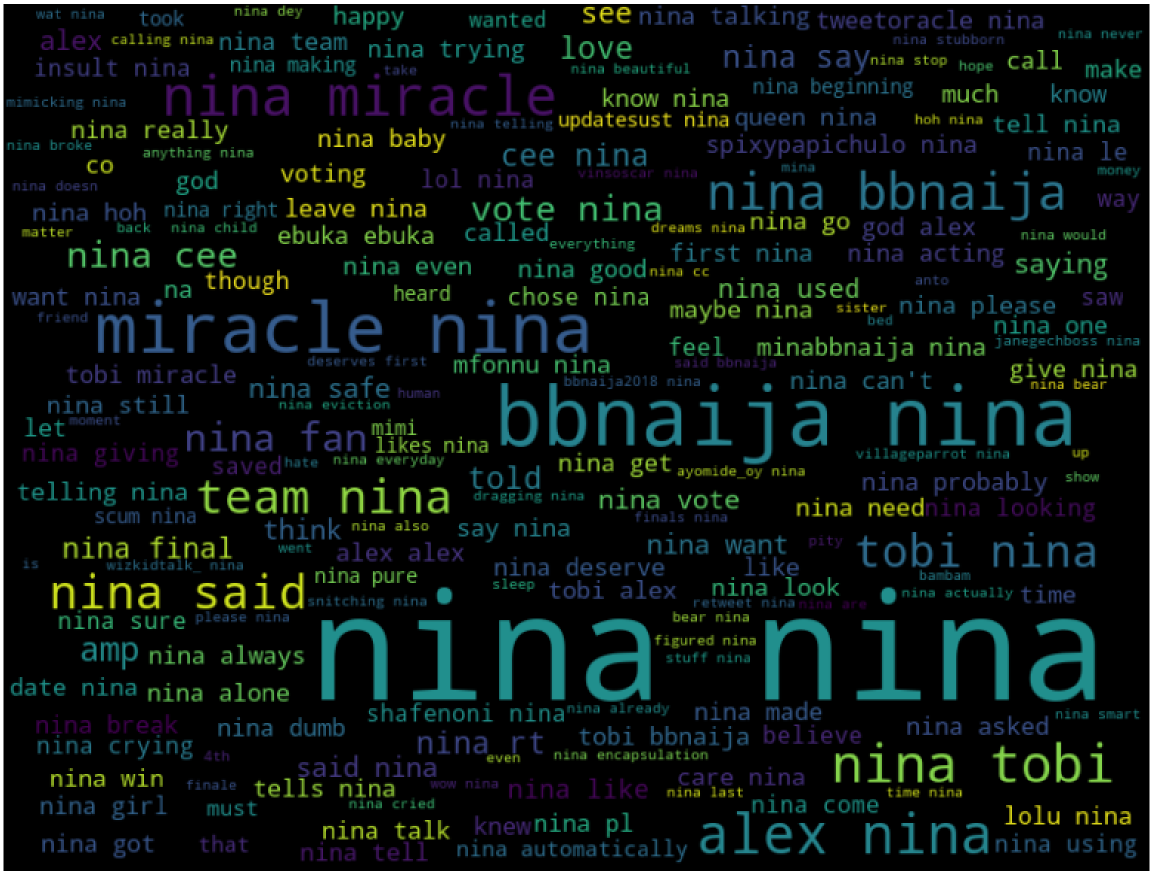
Code

```
contNames = ['miracle', 'cee-c','alex', 'nina', 'lolu',  'khloe', 'tobi',
'anto' ]
interestNgram = [Ngram(wordlst, x) for x in contNames]

PlotWordCloud(interestNgram[0])
PlotWordCloud(interestNgram[1])
```


miracle miracle
vote miracle
bbnaija miracle
team miracle
miracle bbnaija
rt miracle
miracle tobi
like miracle
miracle nina
miracle vote
miracle fan
miracle like
nina miracle
voting miracle
miracle win


cee cee
bbnaija cee
bbnaija
vote cee
team cee
miracle
like cee
voting cee
cee rt
cee fan
tobi cee
king cee
cee vote
alex cee
cee team
lolu
cee like
cee please
cee family
crashed cee
cee said
nina
victim cee
love cee
cee win
cee devilish
cee trending

```
PlotWordCloud(interestNgram[3])
```



Result 5: Who are the Top tweeters and their location?

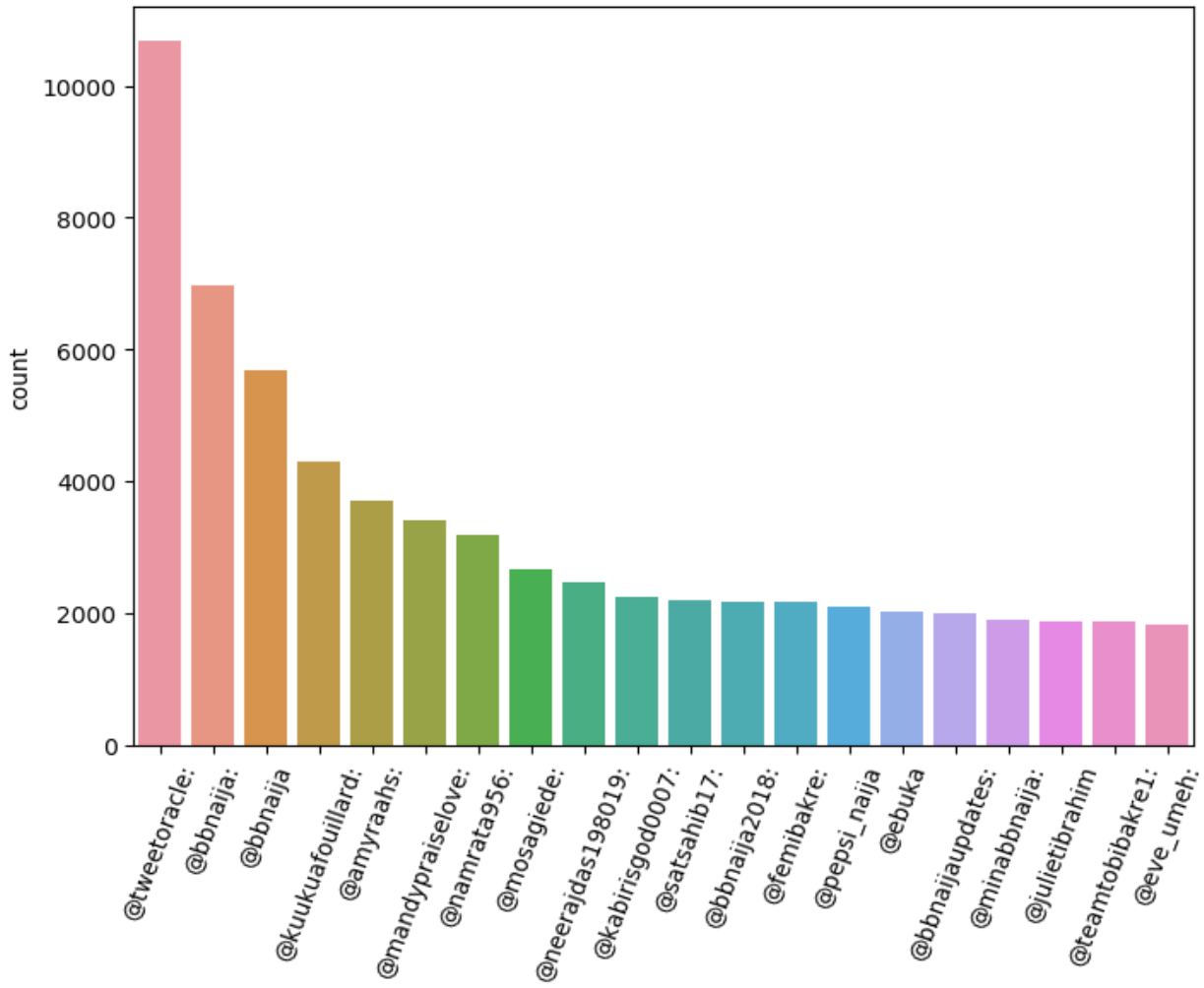
Using subsetting and applying Lambda functions in Pandas series we extracted a subset of the list of word tokens such that the first character in the word is an '@'. These are the user names that occurred throughout the tweets. The top 15 userNames were then plotted using sns count plot.

Code:

```
wordat = wordseries[wordseries.apply(lambda x: x[0] == '@')]

topUser = wordat.value_counts()[:20]
topUserDf = pd.DataFrame({
```

```
    'Username': topUser.index,
    'Counts' : topUser.values})
print(topUserDf)

needLst = [x for x in wordat if x in topUser.index ]

plt.figure(figsize=(8,6), dpi=100)
ax = sns.countplot(x = needLst, order = topUser.index)
ax.set_xticklabels(topUser.index, rotation=70)
plt.show()
```

Graphical Representation of Top Twitter Usernames in the collection of 2018 BBnaija Tweets



We can infer from our research analysis that @tweetoracle is the highest poster as regards BBnaija followed by @bbnaija, @kuukuafouillard, and so on.

Some said @tweetoracle is one of the most influential social media personalities in Nigeria as of 2018, well maybe that is true!

Using Python for Sentiment Analysis

Sentiment Analysis is the process of computationally identifying and categorizing opinions expressed in a piece of text, especially to determine whether the writer's attitude towards a particular topic, product, etc., is positive, negative, or neutral.

In this research, we computationally categorize every tweet around a contestant into either positive, negative, or neutral sentiments. This helps us determine further those contestants who have little or more love from fans.

Tobi seems to be the one with the highest positive, whereas Cee-c even with a high positive has the highest Negative. However, Miracle has a high positive, low negative and has more tweets compared to Tobi.

As statisticians who do not exactly conclude on a hypothesis, however we can infer and make predictions. *We feel Miracle cee-c and Tobi have the top shots in the game*

We made these predictions before the result was released using our analysis. When the results were finally released, the results were exactly as we predicted.

Methodology:

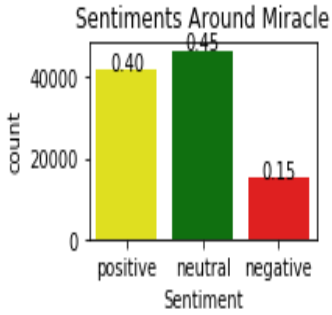
We implemented a **GetTweetSentiment** function. The GetTweetSentiment function was applied to the tweet column. This function returns the sentiment of a sentence or text passed to it. It returns either a positive, neutral, or negative sentiment. This function was applied to each element in the tweet column, using The apply and lambda operations in Python. A new column was then created in the data frame that contains the corresponding sentiment for each tweet.

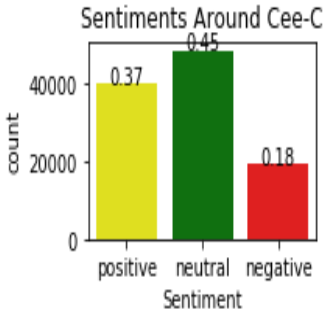
Code:

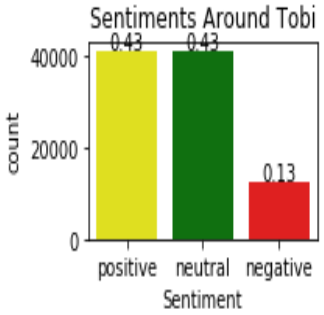
```
twtData['Sentiment'] = twtData['Tweet'].apply(lambda x: get_tweet_sentiment(x))
contExpr = [r'mira', r'cee', r'alex', r'nina', r'lolu', r'khloe', r'tobi', r'anto' ]
```

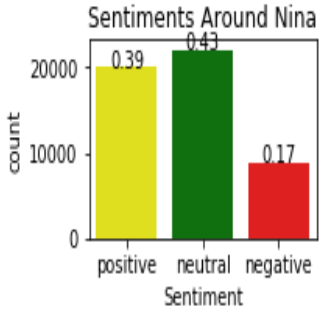
```
contId = [0,1,2,3,4,5,6,7]
contDict = dict(zip(contId,contExpr))
ContestantFrame     =     [twtData.loc[twtData['Tweet'].apply(lambda     x:
bool(re.search(contDict[item], x)))] for item in contId]

def subPlot(data, m, n, title):
   orD = ['positive', 'neutral', 'negative']
            pt    =    sns.countplot(x="Sentiment",    data=data,    palette=
['yellow','green','red'],
                      order = orD , ax=ax[m][n]).set_title(title)

   total = float(len(data))
   for p in ax[m][n].patches:
       height = p.get_height()
                  ax[m][n].text(p.get_x()+p.get_width()/2.,   height   +   3,
'{:1.2f}'.format(height/total),  ha="center")
       plt.subplots_adjust(left  = 0.008, right = 1.7, bottom = 0.1, top =
1.0, wspace = 1.0,hspace = 1.0)
   return pt

fig, ax =plt.subplots(2,3)
subPlot(ContestantFrame[0], 0, 0, 'Sentiments Around Miracle')
subPlot(ContestantFrame[1], 0, 1,'Sentiments Around Cee-C')
subPlot(ContestantFrame[6], 0, 2, 'Sentiments Around Tobi')
subPlot(ContestantFrame[3], 1, 0,'Sentiments Around Nina')
subPlot(ContestantFrame[2], 1, 1,'Sentiments Around Alex')
subPlot(ContestantFrame[5], 1, 2,'Sentiments Around Khloe')

plt.show()
```

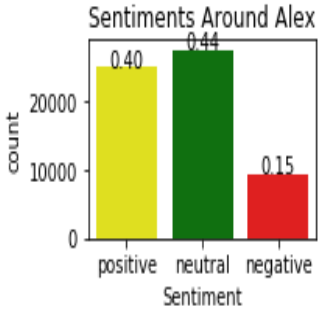
Result 6: Graphical representation of the sentiment Analysis on each contestant

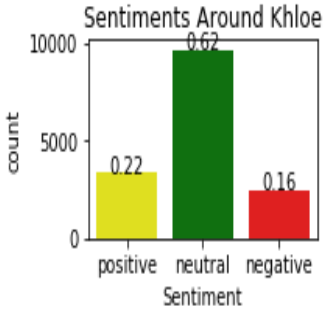












References

1. https://en.wikipedia.org/wiki/Digital_humanities
2. https://digitalhumanities.duke.edu/about-digital-humanities
3. https://www.thebritishacademy.ac.uk/blog/what-are-digital-humanities/
4. https://www.researchgate.net/publication/367510779_Sentiment_Analysis_of_Big_Brother_Nigeria_Tweets
5. https://www.researchgate.net/publication/367511542_Hate_Speech_Identification_from_Tweets_Using_Deep_Learning