Metrics of successful websites and companies

Danai Avratoglou

January 2017

Contents

1 Introduction

An on-line presence of a company was not an important factor of the overall success that the enterprise would have until a few years ago. Taking although into account the vast spread of the impact that internet has on consumers regarding their brand choices and their products this hypothesis is not valid any more. Companies are obliged by the trends to be active on-line and to maintain a website that depicts the image they want their consumers to perceive. By creating a more consumer orientated image they lead the users to create a positive idea regarding the company and this could potentially lead to bigger sales. The purpose of this paper is to understand the relationship that exists between the website of a company and its success. As success we will consider the amount of revenues of the company. Trying to comprehend this relationship a comparison will take place between the enterprises that were deemed as the more successful ones from Fortune 500 based on their revenues and a number of website metrics in order to see by performing regression models and statistical analysis which metrics do affect the most or are related the most with a company's success.

2 Data gathering

The first step in order to contact this research is to find which companies we are going to examine. Since the purpose of this paper is to see if the website metrics that will be examined are influencing the success of the company it is a good idea to examine websites of some already successful companies and try to find out what they have in common. Thus as it is mentioned in the introduction we will examine the 500 companies that were ranked as the most successful ones from Fortune 500.

2.1 Data Source - Fortune 500

The Fortune 500 is an annual list compiled and published by Fortune magazine that ranks 500 of the largest United States corporations by total revenue for their respective fiscal years. The list includes public companies, along with privately held companies for which revenues are publicly available.[1, 2]
For the purposes of this paper we will use this list of companies and we will examine their websites in order to understand if they indeed have something in common or if their success is irrelevant with their on-line presence.
The first thing that we will need is a list of the names that are include in the fortune 500. The easiest way to obtain this list is from the following article: http://www.zyxware.com/articles/4344/list-of-fortune-500-companies-and-their-websites. The way that we will obtain the list will be explained further along in this paper.
In the following table one can see the 50 most successful companies that are included in the Fortune 500 during the period this paper is taking place. The rest of the list is available in the Appendix A.**??**

Table 1: Fortune 500 - 50 first companies

1. Walmart	2. Exxon Mobil	3. Apple
4. Berkshire Hathaway	5. McKesson	6. UnitedHealth Group
7. CVS Health	8. General Motors	9. Ford Motor
10. AT&T	11. General Electric	12. AmerisourceBergen
13. Verizon	14. Chevron	15. Costco
16. Fannie Mae	17. Kroger	18. Amazon.com
19. Walgreens Boots Alliance	20. HP	21. Cardinal Health
22. Express Scripts Holding	23. J.P. Morgan Chase	24. Boeing
25. Microsoft	26. Bank of America Corp.	27. Wells Fargo
28. Home Depot	29. Citigroup	30. Phillips 66
31. IBM	32. Valero Energy	33. Anthem
34. Procter & Gamble	35. State Farm Insurance Cos.	36. Alphabet
37. Comcast	38. Target	39. Johnson & Johnson
40. MetLife	41. Archer Daniels Midland	42. Marathon Petroleum
43. Freddie Mac	44. PepsiCo	45. United Technologies
46. Aetna	47. Lowe's	48. UPS
49. AIG	50. Prudential Financial	

2.2 Metrics

Now that we have declared the companies that we are going to use we will also need to decide which metrics are we going to examine for each site. Since we cannot have access to metrics such as traffic we will have to examine metrics that are more related to how the site is structure and what exactly does the initial page of each site includes. Initially we can divide the metrics in two major categories:

- What we see?
- What lays behind of what we see?

In the first category we are referring to metrics that can easily be conceived by the naked eye as well. For example the images that a website is using in its landing page. How many there are and if they are big or small.
The second category is not so obvious and it includes informations that usually is visible only to the web developer or the creator of the page. The information here are being given from the html code of a site. For instance we can see the actual type and size of an image, an information not visible with the naked eye.
Now that we have a first understanding of the two main categories that the metrics we will use are divided in, we can see in detail the metrics that will be examined in this paper:

2.2.1 Loading time:

One aspect of a website that is crucial is the time it takes for it to load. Nowadays that the internet speed is going higher and higher most people do not have the patient to wait for a page to load. That is why we think that a metrics that should be definitely included in this research in the loading time of the initial page of a site.

2.2.2 Number of links:

When someone is browsing through the internet in many cases they are not completely sure what exactly they are looking for and that is why in a website it would be wise to have some links that can direct the user to find what he wants.These links can either direct the user in another page of the same site, in which case the link will be characterized as an internal link. Or they can lead the user to another site, where in that case the link is characterized as an external link.For the purpose of this paper since it is not so clear which type of links are more important to a user we will examine both the internal and the external links.

2.2.3 Social media:

Our era is marked by the social media wave that has changed our lifestyle and our daily habits. So it would be considered an overview if we didn't take under consideration the number of social media that the company chooses to participate in. Even though they can also be considered as external links of the website we will examine them separately in order to see if any particular social medium effects the company's revenues.

2.2.4 Number, type and sizes of images:

Since the site is the first thing that a user will see for the company and there is a famous quote that says that *"First impressions counts"* we should also examine how the companies decide to visualize their landing page. In other words to see how many images they include in it and more on that what sizes are those images.
It is completely different to see only one huge image in the landing page of a site with not many words or descriptions than to see many small images with different information. The purpose is to examine if these type of diversities between the examined websites are actually related to how they are doing profit wise.
Moreover a kind of information a little more complicated for a simple user to understand, but plays a very important role in many cases is the type of the image. For example some websites are using specific type of images or banners that are not compatible with all the browsers, leading the user to see some break points in the website and even stop visiting it. That is why we believe is important to review this metric as well.

2.2.5 Content:

They say a picture worth a thousand words but that is not enough in our case. After exploring the number, sizes and type of pictures that a website is using we should also explore the number of words it is using to accompany the images and complete the outcome that a user will come across. The metrics we will use will be two. The first one will be the total words that are being used in the landing page and the other one will be the total unique words that are being used. When we are referring to unique words we mean words that are not so commonly use such as "a" or "and" and they give an air of individuality to the text. By using this metric we would try to see if the words that are being used are just as important as the actual content and if the words can make a difference.
Furthermore we should also take under consideration how comprehending is the text used in the websites for the users. This information can be obtained by calculating the readability index of the website and also the number of sentences that exists in a page (a metric that comes to complete the previous ones).How the readability index is being calculating will be furthered explained in another section.

2.2.6 HTML Validation:

Moreover we will have to check the quality of the html code behind the website we are seeing. Are there any mistakes in the code for example any brackets that opened and never closed or any links that do not work. We will examine again two different metrics here the number of errors and the number of warnings. The warning are parts of the code that even though they work at the time there is a good chance to malfunction if any changes or addition are to be made to the html code.

2.3 Python

After having a first look into the variables/ metrics we will use in order to contact this research we should now see how we are going to obtain all this information.
Since all of this information can be subtract from the html code of a company's website we should use a programming language in order to download the html pages and then to extract the specific metrics we want to examine from them.
For the purposes of this paper the programming language that will be used for downloading the metrics from the websites is Python. More specifically the version of Python that will be used is the 2.7 one.
Python is a widely used high-level programming language used for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using white space indentation to delimit code blocks rather than curly braces or keywords), and a syntax which allows programmers

to express concepts in fewer lines of code than possible in languages such as C++ or Java.
The language provides constructs intended to enable writing clear programs on both a small and large scale. Furthermore the way that Python allows a user to programming is common to all users which gives this language a leverage as a program build in Python can be easily understood from another user without any difficulty.
The environment that is going to be used is from the Anaconda package which is a free open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment. To be more precise from this package we are going to use the Jupyter Notebook.[4]

2.4 Scripts

In order to gather all the needed metrics we had to create a variety of small scripts so as to collect them. In this section we will present in detail the procedure and the actual scripts that were used in order to extract the information that later will help us contact the analysis of the relationship between those metrics and the company's status.

2.4.1 Companies ranking, names and url

For starters we need to create a list with the names, the ranking and the URL of the sites that we will later download and extract the necessary info from them. So as it was mentioned in the previous section the first step that needs to be done is to download and gather those informations into a data frame[1] so as to be able to use them later on.
The most easy way to obtain those informations is by extracting them from an already existing list. This list in our case was available in the following link:http://www.zyxware.com/articles/4344/list-of-fortune-500-companies-and-their-websites.
The way to keep only those three informations as three different variables is by separating from the html code of this page the needed information.
The first step is to create 3 empty lists where we will include the informations we are going to extract. The first list will contain the rank of each site, the second one will contain the name of the company and the 3rd one will contain the actual link of the company's site:

```
list_company_number =[]
list_company_name = []
list_company_website = []
```

The second step is to upload some libraries that will help us create this function but also the ones that are going to follow.

[1]a table in Python environment with rows and columns

```
import urllib
import urllib2
import time
import os
from bs4 import BeautifulSoup
import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Finally the third step is to create the function that will firstly download the html code of the url at hand, secondly keep only the part of the code that we need to examine and thirdly save this part into the empty lists we created above. This function is called websites and takes as variable to work only the url of the site we need to examine:

```
def websites (url):
    from time import time
    start = time ()
    browser = urllib2.build_opener()
    browser.addheaders = [('User-agent', 'Mozilla/5.0')]
    response = browser.open(url)
    myHTML = response.read()
    soup = BeautifulSoup(myHTML,"lxml")
    o = 0
    td_list =[]
    for row2 in soup.html.body.findAll('td'):
        td_list.insert(o, row2)
        o = o + 1
    a = 0
    b = 1
    c = 2
    list_numbering = 0
    for i in range (0,500):
        num = str(td_list[a])
        company = str(td_list[b])
        site = str(td_list[c])
        c_num = re.findall('>(.+?)</td>',num)
        c_num = str(c_num[0])
        c_name = re.findall('>(.+?)</td>',company)
        c_name = str(c_name[0])
        c_site = re.findall('">(.+?)</a>',site)
        c_site = str(c_site[0])
        list_company_number.insert(list_numbering,c_num)
        list_company_name.insert(list_numbering,c_name)
        list_company_website.insert(list_numbering,c_site)
        a = a + 3
        b = b + 3
        c = c + 3
        list_numbering =  list_numbering + 1
    end = time ()
    duration = round (end - start , 1)
    minutes = round (duration /60, 1)
    print 'The lists are ready in ', duration , ' seconds'
    print 'The lists are ready in ', minutes , ' minutes'
```

The steps we followed to create the following function are the following:

2.4.1.1 Step 1

We create a fake browser that we are going to use in order to open the page and downloaded. The reason we do that is that many sites do not allow us to download their page because they are afraid of stealing important information. Since we are not using any private information we use this method to avoid issues while trying to open the html page at hand.

2.4.1.2 Step 2

We open the url and we read it while saving it in the variable "myHTML".

2.4.1.3 Step 3

With the help of the BeautifulSoup library we read the page as a lxml file and then for each row of this file we are looking for the "td" parts of the code where the informations we want are included.

2.4.1.4 Step 4

Since we need the names and the urls of all the 500 sites we created a loop from 0 to 500 where for each i we try to isolate the part of the code that contains the information that we want. Moreover even thought it seems that with this loop we calculate 501 numbers since in Python the second bracket is always open we actually count from zero to 499.

2.4.1.5 Step 5

We use reg expressions[2] in order to state precisely what part of the already selected code we want to keep.

[2]A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern.

2.4.1.6 Step 6

We insert with a specific order the names, the ranking and the url to the corresponding lists and finally we create a text that will appear when the function is completed. Here we have also calculated the time that this function took to be completed and we will appear it as well along with the text.

2.4.2 URL validation

Now that we have saved in the three lists the names, the ranking and the URL of the companies that we are going to examine we first have to check whether or not those URL are valid in order to proceed with the download of the html code behind the initial web page of each one of those companies.
We first have to install the validators package in python so as to proceed with the validation of the url. In the command prompt window that opens when you open the Jupyter Notebook you have to write: "pip install validators" and then press enter in order for the package to be installed. This specific package currently supports python versions 2.7, 3.3, 3.4, 3.5 and PyPy.[3] The function that we are going to use from this package is called validators.url. This functions returns True if the url at hand is a valid url or False if it is not.
Below one can see that we created a loop that will run as many time as the length of the list that we created in the previous section.
During this loop we create a string variable where we use the URL that we have saved in the list, one at each time, and we add the "http://" prefix.
Now that we have create the correct way that a url is supposed to be written we will use the function of the validators package and we will create an if function that will check if the answer to a site is different from True and if it is it would add a unit in the variable nv so as to know at the end how many sites did not had valid URL.
Finally we ask from the programme to print the final result so as to know how many URL are not valid. The code that was created for this procedure is the following one:

```
nv = 0
for num in range(len(list_company_website)):
    line = 'http://' + str(list_company_website[num])
    x = validators.url(line)
    if x != True:
        nv = nv +1
print "The validation is complete! There were" , nv,
 "not valid pages"
```

In our case all the URL where valid so we are eligible to proceed at the next part of the code.

[3]https://validators.readthedocs.io/en/latest

2.4.3 Download sites html code

After checking the validity of the URL that we have saved in the list the next step is to download the actual html code of the initial pages of each one of the 500 websites we want to examine.
As we did in the section 2.4.1 we have to create a fake browser so as the to be able to download the html code without contacting any problems. In our case we created a Mozilla browser.
In order for the url to be opened we must first bring it to the correct form. Initially we create again a loop and for each loop we will examine a specific url. We first replace some symbols on the string that will not be recognized if we try to open the url in this form and then we add the "http://" in the begging of each url.
Before starting downloading we create some rules for some exceptions. For example the sites 71 (Best Bay),119 (Arrow Electronics) and 465 (St. Jude Medical) in ranking create problem when we tried to download them and the code is stop working. So in order to avoid such incidents we created an exception and the python code will not even try to download these sites and in their position in the new list with the html pages that we will create a zero will be saved.The same thing would happen if an exception is thrown in the function if that we are creating. In any other case we will open the site and save this action in the variable response2.
After saving the action we will use the command read and we will save the html code which is essentially a long piece of text and then we will save this "text" in the list.
Finally we will wait for 2 seconds in order for the browser not to be suspicious from the extreme speed that we are going to open the next page. In that way we avoid having any crushing incidents on the code.
After completing this procedure for all the site we will have a list that in each position the html code will be held as a very large text. Except from the sites that were not able to be opened.
The code that was created for this procedure is the following:

```
import time
browser2 = urllib2.build_opener()
browser2.addheaders = [('User-agent', 'Mozilla/5.0')]
for i in range (0,500):
    k = str(i + 1)
    lc = str(list_company_website[i])
    lc = lc.replace("'","")
    lc = lc.replace("[","")
    lc = lc.replace("]","")
    lcn = str(list_company_name[i])
    lcn = lcn.replace("'","")
    lcn = lcn.replace("[","")
    lcn = lcn.replace("]","")
    url2= 'http://' + lc
```

```
list500_names.insert(i,lcn)
list500_url.insert(i,lc)
list500_num.insert(i,k)
if i == 118 or i == 464 or i == 70:
    list500_sites.insert(i,0)
    print ("The site " + str(i)
    + " has NOT been downloaded!")
else:
    try:
        response2=browser2.open(url2)
        print ("The site " + str(i)
        + " has been downloaded!")
    except Exception:
        list500_sites.insert(i,0)
        print ("The site " + str(i)
        + " has NOT been downloaded from exception!")
        continue
    myHTML2=response2.read()
    list500_sites.insert(i,myHTML2)
    time.sleep(2)
```

2.4.4 Not downloadable pages

As we explained in the previous section there are some pages that were not able to be downloaded. In order to know which are those pages and more specifically which companies sites will not be available for further exploration we created this part of the code which creates a list with the names of those companies.
In the previous code we used a zero in each position that we weren't able to download the site. Here we will use this information in order to gather in a new list only the parts of the list that did get a zero.
Here we actually created a small function that is called "$not_downloadables$". In the first part of the code we create the function and in the second part we run the function for our lists. Finally we create a data frame with the results so as to be more easy on the eyes.
The code that was created is the following:

```
not_d = []
not_d_n = []
num = []
def not_downloadables (list500_names, list500_sites):
    met = 0
    for i in range(len(list500_names)):
        if list500_sites[i] == 0:
            ct = list500_names[i]
            not_d.insert(met,ct)
            not_d_n.insert(met,str(i))
            num.insert(met,met)
            met = met + 1

not_downloadables (list500_names,list500_sites)

d = {'company' : pd.Series(not_d, index=[num]),
     'number' : pd.Series(not_d_n, index=[num])}
nd = pd.DataFrame(d)
nd
```

In the following table we can see the sites that were not downloaded:

Table 2: Not downloadable sites

16. Fannie Mae	63. HCA Holdings
71. Best Buy	91. Nike
98. Tesoro	119. Arrow Electronics
136. AutoNation	142. Southwest Airlines
162. Southern	165. American Electric Power
196. Office Depot	217. PBF Energy
229. Consolidated Edison	240. Toys R Us
243. Dominion Resources	276. Global Partners
307. PayPal Holdings	327. News Corp.
364. Williams	415. Tractor Supply
442. Old Republic International	465. St. Jude Medical

2.4.5 Content

Now that we have downloaded the html code we should start extracting some of the variables we are going to use for the analysis in the next chapter.
The first thing we are going to check is how easy it is for a user to read the content of each site. In order to check this performance indicator we will need to gather 4 variables:

1. Number of Words
2. Number of Unique Words
3. Number of Sentences
4. Flesch score

The three first are quite clear. The number of words are the total words that appear in the texts that the user is seeing in a website. The number of unique words are the words that are not very common and can attract the readers attention or even make it harder for him to comprehend the text. The number of sentences is related to those variables as we can understand from this is comparison to the number of total words how big or short is the sentence and as a

conclusion how easy or not is the complete text.
Finally regarding the forth variable the flesh score is referring to the Flesh reading ease test score.In the Flesch reading-ease test, higher scores indicate material that is easier to read; lower numbers mark passages that are more difficult to read. The formula for the Flesch reading-ease score (FRES) test is:

$206.835 - 1.015(totalwords/totalsentences) - 84.6(totalsyllables/totalwords)$

We will calculate the flesh reading test and the other three aforementioned variables with the help of an on-line checking site called:
"http://www.webpagefx.com/tools/read-able/".
This site calculates and return all these results that we need. The way to implement them in a new data frame to our python code is by dividing the html code, behind the page with the results for each site, and keeping only the actual numbers and sizes that we seek.

```
flesch = []
sentence = []
word = []
unique_w =[]
empty =[]

import time
for num in range(0,500):
    site = list500_sites[num]
    line = list500_url[num]
    url_check = "http://www.webpagefx.com/tools/
␣␣␣␣read-able/check.php?tab=Test+By+Url&uri=http://"+ line
    browser = urllib2.build_opener()
    browser.addheaders = [('User-agent', 'Mozilla/5.0')]
    if site == 0 or num == 107:
        print("Site", str(num), "is␣not␣validated␣from␣sites")
        flesch.insert(num,"n/a")
        sentence.insert(num,"n/a")
        word.insert(num,"n/a")
        unique_w.insert(num,"n/a")
    else:
        try:
            response = browser.open(url_check)
        except Exception:
            flesch.insert(num,"n/a")
            sentence.insert(num,"n/a")
            word.insert(num,"n/a")
            unique_w.insert(num,"n/a")
            print("Site", str(num), "is␣not␣validated␣from␣check")
            continue
        html_r = response.read()
        check = str(html_r)
        if check != empty:
                soup = BeautifulSoup(check,"lxml")
                o = 0
                keyf = []
                for row in soup.html.body.findAll('tr'):
                    keyf.insert(o,row)
                    o = o + 1
                if keyf != empty:
                        print("Site", str(num), "is validated")
                        #Flesh measurement
                        if keyf[0] != empty:
                            readability = str(keyf[0])
                            split1 = readability.split('>')
                            readability2 = str(split1[4])
                            split2 = readability2.split('<')
                            readability3 = str(split2[0])
                            flesch.insert(num, readability3)
                        else:
                            flesch.insert(num,"n/a")
                            sentence.insert(num,"n/a")
                            word.insert(num,"n/a")
                            unique_w.insert(num,"n/a")
                        #Number of sentences
                        if keyf[6] != empty:
                            sentences = str(keyf[6])
                            spli1 = sentences.split('>')
                            sentences2 = str(spli1[4])
                            spli2 = sentences2.split('<')
                            sentences3 = str(spli2[0])
                            sentence.insert(num, sentences3)
                        else:
                            flesch.insert(num,"n/a")
                            sentence.insert(num,"n/a")
                            word.insert(num,"n/a")
                            unique_w.insert(num,"n/a")
                        #Number of words
                        if keyf[7] != empty:
                            words = str(keyf[7])
                            spl1 = words.split('>')
                            words2 = str(spl1[4])
                            spl2 = words2.split('<')
                            words3 = str(spl2[0])
                            word.insert(num,words3)
                        else:
                            flesch.insert(num,"n/a")
                            sentence.insert(num,"n/a")
                            word.insert(num,"n/a")
                            unique_w.insert(num,"n/a")
                        #No. of complex words
                        if keyf[7] != empty:
                            unique_ws = str(keyf[8])
                            sp1 = unique_ws.split('>')
                            unique_ws2 = str(sp1[4])
                            sp2 = unique_ws2.split('<')
                            unique_ws3 = str(sp2[0])
                            unique_w.insert(num,unique_ws3)
                        else:
                            flesch.insert(num,"n/a")
                            sentence.insert(num,"n/a")
                            word.insert(num,"n/a")
                            unique_w.insert(num,"n/a")
                    else:
                        print("Site", str(num), "is not validated from check 2")
                        flesch.insert(num,"n/a")
                        sentence.insert(num,"n/a")
                        word.insert(num,"n/a")
                        unique_w.insert(num,"n/a")
   time.sleep(2)
```

Step 1
We create four lists where we will save the variables that we are looking for. Then we start a loop for the 500 sites where in each loop we change the value of the 3 main variables the sites that contain the html code the url that contain the url of the company at hand each time and the url check where we save the part of the web address that remains the same while doing the on line check and we add at the end the url of the site we want to check. Also we open the browser as we have done in previous scripts as well.

Step 2
Then we create an if part where we check that the variable site is not zero and that we will not examine the site 108 (Tech Data) as the code seems to have a problem in that case. In this check we put on the respective sites n/a values so as to be clear that we did not retrieve these info for them.

Step 3
We open the browser link and we check whether or not we drop to any exception in which case we also put n/a values in the respective companies.

Step 4

We read the link and after checking that it is not empty (this we can check by comparing it with an empty list that we have created for this use) we use the beautiful soup package to extract the part of the code that we need.

Step 5
After taking a look at the html code of the pages we find out that the parts that we need are between the following brackets ¡tr¿...¡/tr¿ so we will extract each such bracket from the code and save them in a list. By checking the list we found which numbers of the list items we want and we saved them in the variables we have created. Always of course checking first that the specific prices exist or in any other way we again put n/a.

Step 6
Now that we have the for key indicators that we will use later on the analysis we should also create a more easily comprehend variable that is related to the flesh measure variable we just created. The numbers that each site gathered mean different things. So we will try to create a correlation and build a variable called readability that will say in text how easily read or not a site is. The score can be interpreted by the following logic:

- Flesch measure < 30 : Very Confusing
- Flesch measure > 30 : Difficult
- Flesch measure > 50 : Fairly Difficult
- Flesch measure > 60 : Standard
- Flesch measure > 70 : Fairly Easy
- Flesch measure > 80 : Easy
- Flesch measure > 90 : Very Easy

Step 7
We created a function that creates a new list where the flesh measure variable is being saved as the above descriptions based on the scores each of the site achieved.

Step 8
Now that we have all the needed informations in lists we should combine them by creating one data frame with the use of the variable company name again in order to have a common key so as to merge all the data frames that we will create in the end.

```
readability = []
def readable (flesch):
    for i in range (len(flesch)):
            f_n = flesch[i]
            if f_n == "n/a":
                readability.insert(i,"n/a")
            else:
                a = int(float(f_n))
                if a > 90:
                    readability.insert(i,"Very easy")
                elif a > 80:
                    readability.insert(i,"Easy")
                elif a > 70:
                    readability.insert(i,"Fairly easy")
                elif a > 60:
                    readability.insert(i,"Standard")
                elif a > 50:
                    readability.insert(i,"Fairly difficult")
                elif a > 30:
                    readability.insert(i,"Difficult")
                else:
                    readability.insert(i,"Very Confusing")
    print "The function is completed!"

readable (flesch)

d1 = {'company' : pd.Series(list500_names, index=[list500_num]),
        'url' : pd.Series(list500_url, index=[list500_num]),
        'Readability' : pd.Series(readability, index=[list500_num]),
        'Flesh_Mesaure' : pd.Series(flesch,index=[list500_num]),
'Sentences' : pd.Series(sentence, index=[list500_num]),
'Words' : pd.Series(word, index=[list500_num]),
'Unique words' : pd.Series(unique_w, index=[list500_num])}
fre = pd.DataFrame(d1)
```

2.4.6 HTML Validation

After downloading and saving in data frames the first batch of metrics that we will need for the analysis that we will do further along we should also check the quality of the html code.
Most pages on the World Wide Web are written in computer languages (such as HTML). One of the advantages of writing in a computer language is that it allows the developer to structure text, add multimedia content, and specify what appearance, or style, the result should have based on his needs.
As every speaking language,so the computer languages do have their own grammar, vocabulary and syntax too. Each document that is written in these computer languages is supposed to follow these rules in order to consider it well structure and written.
However, just as texts in a natural language can include spelling or grammar errors, documents using computer languages may (for various reasons) not be

following these rules as well.
The process of verifying whether or not a document actually follows the rules for the language it uses is called validation, and the tool used for that is a validator. A document that passes this process with success is called valid.
With these concepts in mind, we can define ”validation” as the process of checking a web document against the grammar (generally a DTD[4]) it claims to be using.
For the purposes of this paper we will use an on-line validator site called W3C[5] which will help as see how many errors and warnings does each site have.

Step 1
Initially we create the empty lists in which we will save the variables we will extract that will give as a clear glance on how many errors does each page has, how many warnings, how many pages weren't recognized as documents and how many pages did open during the process.

Step 2
The next step is to create a function that will do a similar job as the script we used to extract the previous variables. Initially we locate the part of the url that remains the same when the check is completed and we see the part that does change where the name of the url we want to examine should go. In order for the code to recognize that we save the result in a variable that in the end is a new url that when we open it, it will show as the page of the results for each site in each loop that we are examining.

Step 3
One difference in this script is that now the part of the code that we need is not between ¡tr¿..¡/tr¿ but between ¡div¿...¡/div¿ parts of the code. So we follow the same procedure as before and we locate in the list of all the divs that we create with the help of the Beatuful soup package the parts that gives us the informations we want to keep.

Step 4
One other difference is that here after the procedure ends we have counted the actual time we needed to complete this procedure. Then next step is to run the function we created.

Step 5
Finally we save the lists that we created in a new data frame where the common column remains the name of the company as it already is in the previous dataframes we have created.

```
num_errors = []
num_info_warnings = []
```

[4]DTT: Document Type definition
[5]https://validator.w3.org/

```
num_non_doc = []
nm = []
num_open_page = []
empty = ""

def html_validation (list500_url ,list500_names):
    from time import time # I used it to see how much time it does to run the fu
    start = time ()
    for num in range(len(list500_names)):
        line = list500_url[num]
        url_check = "https://validator.w3.org/nu/?doc=https://" + line
        browser = urllib2.build_opener()
        browser.addheaders = [('User-agent', 'Mozilla/5.0')]
        response = browser.open(url_check)
        html_check = response.read()
        html_check
        check = str(html_check)
        er = 0
        err = 0
        errr = 0
        e = False
        if check != empty:
            e = True
            soup = BeautifulSoup(check,"lxml")
            o = 0
            keyf = []
            for row in soup.html.body.findAll('div'):
                keyf.insert(o,row)
                o = o + 1
            if len(keyf) != 0:
                    keyfin = str(keyf[2])
                    dol= re.findall('class="error"',keyfin)
                    er = er + len(dol)
                    doll= re.findall('class="info warning"'
                                    ,keyfin)
                    err = err + len(doll)
                    dolll= re.findall('class="non-document-error io"'
                                    ,keyfin)
                    errr = errr + len(dolll)
        num_errors.insert(num,er)
        num_info_warnings.insert(num,err)
        num_non_doc.insert(num,errr)
        nm.insert(num,num)
        num_open_page.insert(num,e)
    end = time ()
    duration = round (end - start , 3)
    minutes = round (duration /60, 1)
    print 'The lists are ready in ', minutes, ' minutes'


html_validation (list500_url ,list500_names)


d8 = {'company' : pd.Series(list500_names, index=[nm]),
        'The_page_opened' : pd.Series(num_open_page, index=[nm])
        ,'number_of_errors' : pd.Series(num_errors, index=[nm]),
        'number_of_warning' : pd.Series(num_info_warnings, index=[nm])
        ,'non-document-error' : pd.Series(num_non_doc, index=[nm])}
html_val = pd.DataFrame(d8)
html_val.head(3)
```

2.4.7 Social media

One other group of variables that we should include in our analysis is the social media that each company choose to use. Since this era is being characterized from the massive use of social media we can't help but wonder if some of them could actually play a crucial role in a business success.
The way we are going to see which social media does a company use is quite simple. We will create a list with the name of each of the six most well known social media with the suffix ".com" and then we will check if the expression appears in the html code of each company. If an expression does exists that means that the specific company does indeed uses this specific social media and as so it appears a respective link in it's home page so as the user to have the opportunity to subscribe in it.
The procedure we will use is similar to the previous ones. Firstly we create the lists for the social media that we want to examine. More specifically the social media we will search for are the following:

- **Facebook**: a social media that lets you upload images, thoughts, songs and lets you interact with your friends
- **Twitter**: an on-line news and social networking service where users post and interact with messages, "tweets," restricted to 140 characters
- **Pinterest**: an internet photo sharing and publishing service that allows users to "Pin" pictures they like and upload their own recommendations to their "pinboards".
- **YouTube**: a free video sharing website that lets people upload, view, and share videos.
- **Instagram**: an online photo and video sharing social networking service. It allows users to take pictures and videos, apply digital filters to them and share them to their followers.

- **Linkedin**: a social networking website for people in professional jobs. Users can make connections with other people they have worked with, post their work experience and skills, look for jobs, and look for workers.

The next step is to create a function that first creates the lists with the 6 elements we would search on each html code. Then after checking that the html code for each specific site has been downloaded properly (in other words is not equal to zero) we search in a loop for each of the social media at hand if it exists in the html code. If it exists we put True in the respective list.After completing this procedure we export also the time we did to run the function.
Now that we have the function ready we run it and afterwards we again save the results in a new data frame.

```
sm_f = []
sm_t = []
sm_i = []
sm_p = []
sm_y = []
sm_l = []
sm_nm = []
nm = []
sm_url = []

def socialmedia (list500_sites ,list500_names , list500_url ):
    from time import time
    start = time ()
    for i in range(len(list500_names)):
            myHTML = list500_sites[i]
            sm = ['facebook.com','twitter.com',
                  'instagram.com','pinterest.com',
                  'youtube.com','linkedin.com']
            if myHTML == 0:
                sm_nm.insert(i,list500_names[i])
                nm.insert(i,i)
                sm_url.insert(i,list500_url[i])
                sm_f.insert(i,'n/a')
                sm_t.insert(i,'n/a')
                sm_i.insert(i,'n/a')
                sm_p.insert(i,'n/a')
                sm_y.insert(i,'n/a')
                sm_l.insert(i,'n/a')
            else:
                for index in range(len(sm)):
                    x = sm[index]
                    social = re.findall(x,myHTML)
                    if (len(social) > 0):
                        if x == 'facebook.com':
```

```
                    answerf = 'TRUE'
                if x == 'twitter.com':
                    answert = 'TRUE'
                if x == 'instagram.com':
                    answeri = 'TRUE'
                if x == 'pinterest.com':
                    answerp = 'TRUE'
                if x == 'youtube.com':
                    answery = 'TRUE'
                if x =='linkedin.com':
                    answerl = 'TRUE'
            else:
                 if x == 'facebook.com':
                    answerf = 'FALSE'
                 if x == 'twitter.com':
                    answert = 'FALSE'
                 if x == 'instagram.com':
                    answeri = 'FALSE'
                 if x == 'pinterest.com':
                    answerp = 'FALSE'
                 if x == 'youtube.com':
                    answery = 'FALSE'
                 if x =='linkedin.com':
                    answerl = 'FALSE'
        sm_nm.insert(i,list500_names[i])
        nm.insert(i,i)
        sm_url.insert(i,list500_url[i])
        sm_f.insert(i,answerf)
        sm_t.insert(i,answert)
        sm_i.insert(i,answeri)
        sm_p.insert(i,answerp)
        sm_y.insert(i,answery)
        sm_l.insert(i,answerl)
    end = time ()
    duration = round (end - start, 3)
    minutes = round (duration /60, 1)
    print 'The lists are completed in ', minutes, ' minutes'
    print 'The lists are ready in ', duration, ' seconds'

socialmedia (list500_sites,list500_names,list500_url)

d2 = {'company' : pd.Series(sm_nm, index=[nm]),
      'facebook' : pd.Series(sm_f, index=[nm]),
       'twitter' : pd.Series(sm_t, index=[nm]),
      'instagram' : pd.Series(sm_i, index=[nm]),
       'pinterest' : pd.Series(sm_p, index=[nm]),
      'youtube' : pd.Series(sm_y, index=[nm]),
       'linkedin' : pd.Series(sm_l, index=[nm]),}
social_media = pd.DataFrame(d2)
```

2.4.8 Links - Internal and External

Next step is to see how many internal, external and finally total links each html code has. The links show in how many other pages does this home page leads to. If the links are external that means as we said before that they lead in a page that is not of this website. While the internal links lead to other pages in the same website.
By extracting this information we want to see if it is important for the user to have the ability to browse in various pages inside the site from the home page and also if it plays any role the use of external links that give the user the opportunity to be transferred in another site that is relative of course with the one he is looking in.
One easy way to spot the links in an html code is the prefix that they all use which is "href". So the first step (after creating the initial lists of course and the loop we do in each script) is to locate how many times does the expression "href" appears in the code. The result of this search will give us the total links of the site.
Next step is to separate somehow the internal and the external ones. An easy way to do that is to find all the "href" references that are followed by "="https: ". By locating specifically those expressions we have located all the external links because unlike the internal links that do not need to be written in this form the external links since they lead to other sites they should always begin with this expression.
Now that we have the total links and the external links as well we can easily find the internal links with a simple subtraction.:

total links - external links = internal links

Finally we put the results in the respective lists for each site and we export the time we did to run the script.
Now that the function is completed we run it and then we create a data frame as we have done in the previous scripts as well.

```
l_nm = []
l_ex = []
l_in = []
l_t = []
nm = []
l_url = []


def links (list500_sites ,list500_names ,list500_url):
    from time import time
    start = time ()
    for num in range(len(list500_names)):
            myHTML = list500_sites[num]
            if myHTML == 0:
                l_nm.insert(num,list500_names[num])
                l_ex.insert(num,'n/a')
                l_t.insert(num,'n/a')
                l_in.insert(num,'n/a')
                nm.insert(num,num)
            else:
                href = re.findall('href',myHTML)
                external = re.findall('href="https:',myHTML)
                ex = (len(external))
                alllinks = (len(href))
                internal =  (len(href) - len(external))
                l_nm.insert(num,list500_names[num])
                l_ex.insert(num,ex)
                l_t.insert(num,alllinks)
                l_in.insert(num,internal)
                nm.insert(num,num)
    end = time ()
    duration = round (end - start, 3)
    minutes = round (duration /60, 1)
    print 'The lists are ready in ', minutes, ' minutes'
    print 'The lists are ready in ', duration, ' seconds'

links (list500_sites,list500_names,list500_url)

d3 = {'company' : pd.Series(l_nm, index=[nm]),
      'external' : pd.Series(l_ex, index=[nm]),
      'internal' : pd.Series(l_in, index=[nm]),
     'total_links' : pd.Series(l_t, index=[nm])}
sites_links = pd.DataFrame(d3)
```

2.4.9 Loading time

One very important factor as we mentioned before is the time that the site does to be loaded and with this part of the code we will count exactly that.
We have already counted in previous scripts the time they did to be completed. In a similar logic we will count the time it does for the browser create after opening the url to read it.
The procedure is the same, firstly we create the empty list where we will save the time it does to open for each site. Then we create the function and we create the loop for the 500 sites. In case of an exception we put "n/a" in the respective list position or if the site is the number 119(Arrow Electronics) or

465(St. Jude Medical)[6] which create a problem in the code and we continue. In the end of the function we appear a text that lets us know that the process has been completed.
Next step is to run the function and finally create a data frame with the results.

```
lt_nm = []
lt_time = []
nm = []
lt_url = []

def loadtime (list_company_website ,list500_names ,list500_url ):
    from time import time
    browser2 = urllib2.build_opener()
    browser2.addheaders = [('User-agent', 'Mozilla/5.0')]
    for num in range(len(list500_names)):
        lc = str(list_company_website[num])
        lc = lc.replace("'","")
        lc = lc.replace("[","")
        lc = lc.replace("]","")
        url2 = 'http://' + lc
        if num == 118 or num == 464:
            lt_nm.insert(num, list500_names[num])
            lt_time.insert(num, 'n/a')
            nm.insert(num,num)
            lt_url.insert(num, list500_url[num])
        else:
            try:
                response2 = browser2.open(url2)
            except Exception:
                lt_time.insert(num, 'n/a')
                lt_nm.insert(num, list500_names[num])
                nm.insert(num,num)
                print ("The site " + str(num)+ " has NOT been loaded!")
                continue
            start_time = time()
            myHTML2 = response2.read()
            end_time = time()
            response2.close()
            l_t = round(end_time-start_time , 3)
            #in order to be more readable we rounded the time
            loadt = str(l_t)
            lt_nm.insert(num, list500_names[num])
            lt_time.insert(num, loadt)
            nm.insert(num,num)
            lt_url.insert(num, list500_url[num])
```

[6] or 118/464 in the loop's numbering since it start from zero

```
            #print ("The site " + str(num) + " has been loaded!")
    print "The function is completed!"

loadtime (list_company_website,list500_names,list500_url)

d4 = {'company' : pd.Series(lt_nm, index=[nm]),
      'loading_time' : pd.Series(lt_time, index=[nm])}
loading_time = pd.DataFrame(d4)
```

2.4.10 Number, type and sizes of images

We reach to the final group of variables that we are going to examine regarding the home pages of the websites of the companies at hand. This group has to do with the images of each site. We can divide the informations that we want to extract into 3 major categories:

1. Type of images
2. Number of images
3. Different image sizes

2.4.10.1 Type of images

There are many different types of images that can be used in a site. Here we will try to see between the most common types which ones are the most preferable from the sites and later on in the analysis to see if this has any correlation to their success. The type of images that we will examine are the following ones:

- **PNG**: Portable Network Graphics (PNG) is a raster graphics file format that supports lossless data compression[7].
- **BMP**: Bitmap (BMP) is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter), especially on Microsoft Windows and OS operating systems.
- **DIB**: Device-Independent Bitmap (DIB) is a graphics file format used by Windows. DIB files are bitmapped graphics that represent color formats. Similar to BMP format, except they have a different header. DIB files can be opened and edited in most image editing programs.
- **JPEG/JPG/JPE**: The term (JPEG) is an acronym for the Joint Photographic Experts Group, which created the standard method of lossy compression for digital images, particularly for those images produced by

[7]Lossless compression is a class of data compression algorithms that allows the original data to be perfectly reconstructed from the compressed data

digital photography. The degree of compression can be adjusted, allowing a selectable trade-off between storage size and image quality. JPEG typically achieves 10:1 compression with little perceptible loss in image quality.

- **GIF**: The Graphics Interchange Format (GIF) is a bitmap image format that supports up to 8 bits per pixel for each image, allowing a single image to reference its own palette of up to 256 different colors chosen from the 24-bit RGB color space. It also supports animations and allows a separate palette of up to 256 colors for each frame. These palette limitations make the GIF format less suitable for reproducing color photographs and other images with continuous color, but it is well-suited for simpler images such as graphics or logos with solid areas of color.
- **TIFF/TIF**:Tagged Image File Format (TIFF or TIF) is a computer file format for storing raster graphics images, popular among graphic artists, the publishing industry and photographers. The TIFF format is widely supported by image-manipulation applications, by publishing and page layout applications, and by scanning, faxing, word processing, optical character recognition and other applications.

As we can see some of the type of images have more than one ways that can appear so in order to be precise we will examine all nine different endings.
The procedure we will follow is the same one. Initially we create the lists we will use then we create the function and the loop for the 500 sites. For each site we check if the html code is not empty and then we search for all the endings that exists in each site and we save in the lists the number of times the ending appeared so as to know how many such images the site has.

2.4.10.2 Number of images

The total number of images is the sum of all the different types of images and the number of each of those in a site. So in order to catch to birds with one stone we create a variable in the same function where in each loop for each different type of image it adds them in this variable so as to have the total images in the end.
In the end of the function we export the time the function did to run again. The next step is to run the function and finally create the data frame with the variables we created.

```
p_p = []
p_d = []
p_jpg = []
p_jpeg = []
p_gif = []
p_tif = []
p_tiff = []
p_bmp = []
```

```
p_jpe = []
p_nm = []
p_tt =[]
nm = []
p_url = []

def images (list500_sites,list500_names,list500_url):
    from time import time
    start = time ()
    for num in range(len(list500_names)):
            myHTML = list500_sites[num]
            image = ['.png','.dib','.jpg','.jpeg',
                     '.bmp','.jpe','.gif','.tif','.tiff']
            totalnumber = 0
            if myHTML == 0:
                p_nm.insert(num,list500_names[num])
                p_p.insert(num,'n/a')
                p_d.insert(num,'n/a')
                p_jpg.insert(num,'n/a')
                p_jpeg.insert(num,'n/a')
                p_gif.insert(num,'n/a')
                p_tif.insert(num,'n/a')
                p_tiff.insert(num,'n/a')
                p_bmp.insert(num,'n/a')
                p_jpe.insert(num,'n/a')
                p_tt.insert(num,'n/a')
                nm.insert(num,num)
                p_url.insert(num,list500_url[num])
            else:
                for index in range(len(image)):
                    x = image[index]
                    photo = re.findall(x,myHTML)
                    if x == '.png':
                        p = str (len(photo))
                    if x == '.dib':
                        d = str (len(photo))
                    if x == '.jpg':
                        jpg = str (len(photo))
                    if x == '.jpeg':
                        jpeg = str (len(photo))
                    if x == '.gif':
                        gif = str (len(photo))
                    if x == '.tif':
                        tif = str (len(photo))
                    if x == '.tiff':
                        tiff = str (len(photo))
                    if x == '.bmp':
                        bmp = str (len(photo))
                    if x == '.jpe':
                        jpe = str (len(photo))
                    totalnumber = len(photo) + totalnumber
                total = str (totalnumber)
                p_nm.insert(num,list500_names[num])
                p_p.insert(num,p)
                p_d.insert(num,d)
                p_jpg.insert(num,jpg)
                p_jpeg.insert(num,jpeg)
                p_gif.insert(num,gif)
                p_tif.insert(num,tif)
                p_tiff.insert(num,tiff)
                p_bmp.insert(num,bmp)
                p_jpe.insert(num,jpe)
                p_tt.insert(num,total)
                nm.insert(num,num)
                p_url.insert(num,list500_url[num])
    end = time ()
    duration = round (end - start , 3)
    minutes = round (duration /60, 1)
    print 'The lists are ready in ', minutes , ' minutes'
    print 'The lists are ready in ', duration , ' seconds'

images (list500_sites ,list500_names ,list500_url)

d5 = {'company' : pd.Series(p_nm, index=[nm]),
      '.png' : pd.Series(p_p, index=[nm]),
      '.dib' : pd.Series(p_d, index=[nm]),
      '.jpg' : pd.Series(p_jpg, index=[nm]),
      '.jpeg' : pd.Series(p_jpeg, index=[nm]),
      '.bmp' : pd.Series(p_bmp, index=[nm]),
      '.jpe' : pd.Series(p_jpe, index=[nm]),
      '.gif' : pd.Series(p_gif, index=[nm]),
      '.tif' : pd.Series(p_tif, index=[nm]),
      '.tiff' : pd.Series(p_tiff, index=[nm]),
      'total_images' : pd.Series(p_tt, index=[nm])}
images_types = pd.DataFrame(d5)
```

2.4.10.3 Different image sizes

Aside from the number of images and the types that are being used another important factor is the size of each image. There is a vast majority of different sizes and it won't be wise to restrict the research to specific sizes so in the following code will find the different sizes that are being used in the companies.

Step 1: The first step is to find the different dimensions that each site uses. We initially create the lists that we will use. In the script there are also comments of what each list represents. Then we create a function that will gather all the different dimensions from all the sites and in another variable all the times each of this dimension appears. Next with the help of the package Beautiful soup we search and gather every mention of the ¡img¿...¡/img¿ part of the code. Then we try to separate the height and the weight since in some cases there aren't available both the dimensions and in order to be more precise we will keep only the sizes that have both the dimensions available.

```
nm = []
s_comp = []
s_dimensions = []
s_times = []
s_tt_dif_dim = []
ht = [] #list of different heights in each case
wt = [] #list of different widths in each case
h_w = [] # combinations of height and width
dif_size = []
un_size = []
s_url = []

def find_dif_sizes (list_company_website,list500_names,list500_url):
    from time import time
    start = time ()
    for num in range(len(list500_names)):
            nm.insert(num,num)
            s_comp.insert(num,list500_names[num])
            s_url.insert(num,list500_url[num])
            myHTML = list500_sites[num]
            if myHTML == 0:
                s_dimensions.insert(num,0)
                s_times.insert(num,0)
            else:
                soup = BeautifulSoup(myHTML, "lxml")
                s_dimensions_local = []
                s_times_local = []
                hw = 0
                for tag in soup.find_all('img'):
                    h = tag.attrs.get('height', None)
                    w = tag.attrs.get('width', None)
                    if h != None:
                        if w != None:
                            ht.insert(hw,h)
                            wt.insert(hw,w)
                            hw = hw + 1
                hw2 = 0
                for l in range(len(ht)):
                    h_w_c = ht[l] + 'x' + wt[l]
                    #we create a str with the form (300x300)
                    #so as to be more easily to read later on
                    h_w.insert(hw2,h_w_c)
                    #we put it in a new list
                    hw2 = hw2 + 1
                if h_w == []:
                    nm.insert(num,num)
                    s_comp.insert(num,list500_names[num])
                    s_dimensions.insert(num,0)
                    s_times.insert(num,0)
                if h_w != []:
                    hw_unique = Counter(h_w)
                    hw_unique2 = str(hw_unique)
                    split1 = hw_unique2.split('{')
                    a = split1[1]
                    split2 = a.split('}')
                    b = split2[0]
                    split3 = b.split(',')
                    finalsplit = []
                    fs = []
                    z = 0
                    m = 1
                    j = 0
                    z1 = 0
                    m1 = 1
                    for numb in split3:
                        oldstring = numb
                        newstring = oldstring.replace("'", "")
                        new = newstring.replace("'","")
                        string = new.replace(" ","")
                        finalstring = string.split(':')
                        for xx in range(len(finalstring)):
                            ax = finalstring[xx]
                            if 'x' in ax:
                                s_dimensions_local.insert(z1,finalstring[xx])
                                z1 = z1 + 1
                            else:
                                s_times_local.insert(m1,finalstring[xx])
                                m1 = m1 + 1
                    s_dimensions.insert(num,s_dimensions_local)
                    s_times.insert(num,s_times_local)
    end = time ()
    duration = round (end - start , 3)
```

```
    minutes = round (duration /60, 1)
    print 'The lists are ready in ', minutes, ' minutes'
    print 'The lists are ready in ', duration, ' seconds'

find_dif_sizes (list500_sites ,list500_names ,list500_url)

def unique_dif_sizes (s_dimensions ,list500_names):
    ds = 0
    for num in range(len(list500_names)):
        asw = s_dimensions[num]
        if asw != 0 :
            for s in range(len(asw)):
                ss = asw[s]
                dif_size.insert(ds,ss)
                ds = ds + 1
    dsu = 0
    for i in dif_size:
        if i not in un_size:
            un_size.insert(dsu,i)
            dsu = dsu + 1
    print(un_size)


unique_dif_sizes (s_dimensions ,list500_names)

#The lists we will need for the next function
t_f_s = []
ttf = []
nm = []
com = []

#Function in order to check whether or not each
#company has these dimensions
def dimensions_per_company (un_size ,list500_names):
    from time import time
    # I used it to see how much time it does to run the function
    start = time ()
    #t_f_s.insert(0,un_size)
    #ttf.insert(0,t_f_s)
    for num in range(len(list500_names)):
        #print(str(num))
        s1a = s_dimensions[num]
        #dimensions of site num
        where = [] #empty list
        wh = 0
        haveornot = []
        for er in range (len(un_size)):
            if s1a != 0 :
                for sizea in s1a:
                    if sizea == un_size[er]:
                        where.insert(wh,str(er))
                        wh = wh +1
                        break
            if str(er) in where:
                haveornot.insert(er,True)
            else:
                haveornot.insert(er,False)

        t_f_s.insert(num,haveornot)
        ttf.insert(num,t_f_s)
        nm.insert(num,num)
        com.insert(num,list500_names[num])
    end = time ()
    duration = round (end - start , 3)
    minutes = round (duration /60, 1)
    print 'The lists are ready in ', minutes, ' minutes'
    print 'The lists are ready in ', duration, ' seconds'

#Run the function dimensions_per_company
dimensions_per_company (un_size,list500_names)

#Create an initial dataframe where we will add the sizes later on
d6 = {'company' : pd.Series(com, index=[nm])}
sizess = pd.DataFrame(d6)

#Now we want to break the variable t_f_s
#in order to add the columns to the dataframe
#Finally we create the data frame with the elements we found
def final_dimensions_dataframe (un_size,t_f_s,list500_names):
    from time import time
    # I used it to see how much time it does to run the function
    start = time ()
    for q in range(len(un_size)):
        names = un_size[q]
        var = []
        for num in range(len(list500_names)):
            a = t_f_s[num]
            var.insert(num,a[q])
        sizess[names] = pd.Series(var, index=sizess.index)
    end = time ()
    duration = round (end - start , 3)
    minutes = round (duration /60, 1)
    print 'The lists are ready in ', minutes, ' minutes'
    print 'The lists are ready in ', duration, ' seconds'

final_dimensions_dataframe (un_size, t_f_s, list500_names)
```

2.4.11 Fortune 500 - Metrics download

Just in order to have some metrics that will depict the status of the companies we are going to analyse we will also download some metrics from the fortune 500 sites for each company.

```
#The next step is to take some informations from the fortune 500 site for each c
#In order to achieve that we should open the pages for each one of the sites sep
#Since there is a pattern in the way the pages are named it shouldn't be difficu
#Firstly we should create the pattern with which we will download the pages
#By running the code we can see that the names of each comany are not
#written exactly as we have saved them
#So we do need to alter the names first in order for the below function to run




#creating a new list with alterations in order for the names
#to match the ones that fortune 500 uses so that we can download the html page
list_company_name_new = []
for num in range (0,500):
    cn = list_company_name[num]
    cn = cn.replace(" ", "-")
    cn = cn.replace("&", "")
    cn = cn.replace("         ", "")
    cn = cn.replace(".", "-")
    cn = cn.replace("amp;", "")
    company = cn.lower()
    list_company_name_new.insert(num,cn)




fortune_pages = []
def fortune500 (list_company_name_new):
    from time import time # I used it to see how much time it does to run the fu
    start = time ()
    for num3 in range (0,500):
        i = str (num3 +1)
        companyname =  list_company_name_new[num3]
        browser = urllib2.build_opener()
```

```
          #because i work from different computers with different
          #pyhton version some commands are not recognizable in each version
          browser.addheaders = [('User-agent', 'Mozilla/5.0')]
          site_fortune = "http://beta.fortune.com/fortune500/"+companyname+"-"+ i
          page_fortune = browser.open(site_fortune)
          html_fortune = page_fortune.read()
          #print("fortune page for company: ", list_company_name_new[num3], i)
          fortune_pages.insert(num3, html_fortune)
      end = time ()
      duration = round (end - start, 3)
      minutes = round (duration /60, 1)
      print 'The lists are ready in ', minutes, ' minutes'
      print 'The lists are ready in ', duration, ' seconds'



#Run the function we created
fortune500 (list_company_name_new)


    The lists are ready in  21.6  minutes
    The lists are ready in  1295.917  seconds



#Now that we have opened the url we are going to extract
#some informations that we need from them
#In order to do that initially we have to create
#the variables we will need
keyf =[]
per =[]
rev_dol = []
rev_per = []
prof_dol = []
prof_per = []
assets_dol = []
assets_per = []
tse_dol = []
tse_per = []
mar_dol = []
mar_per = []
market = []
nm = []
ln = []
```

```
urln = []
empty = []



def fortune_metrics (list_company_name,list_company_website):
    x = 0
    for n in range (0,500):    #we put 25 for testing
        nm.insert(x,x)
        ln.insert(x,list_company_name[n])
        urln.insert(x,list_company_website[n])
        files = fortune_pages[x]
        soup = BeautifulSoup(files,"lxml")
        o=0
        for row in soup.html.body.findAll('tbody'):
            keyf.insert(o,row)
            o=o+1
        keyfin = keyf[0]
        #the elements we need is in the first tbody of the code
        data = keyfin.findAll('td')

        one = str(data[0])
        # revenue
        two = str(data[1])
        # revenue in dollars we need to extract this
        revdol= re.findall('>\$(.+?)</td>',two)
        #we keep only the numbers
        if revdol[0] != empty:
            w = revdol[0]
            a = w.replace("[", "")
            r = a.replace("]","")
            rev_dol.insert(x,r)
        else:
            rev_dol.insert(x,'not available')
        tria = str(data[2])
        # revenue in percentage we need to extract this as well
        revper= re.findall('>(.+?)%</td>',tria)
        #we keep only the numbers
        if revper != empty:
            w = revper[0]
            a = w.replace("[", "")
            r1 = a.replace("]","")
            rev_per.insert(x,r1)
        else:
            rev_per.insert(x,'not available')
```

```
four = str(data[3])    # profit
five = str(data[4])
# profit in dollars we need to extract this
profdol= re.findall('>\$(.+?)</td>',five)
#we keep only the numbers
if profdol != empty:
    w = profdol[0]
    a = w.replace("[", "")
    p = a.replace("]","")
    prof_dol.insert(x,p)
else:
    prof_dol.insert(x,'not available')
six = str(data[5])
# profit in percentage we need to extract this as well
profper = re.findall('>(.+?)%</td>',six)
#we keep only the numbers
if profper != empty:
    w = profper[0]
    a = w.replace("[", "")
    p1 = a.replace("]","")
    prof_per.insert(x,p1)
else:
    prof_per.insert(x,'not available')
seven = str(data[6]) #assets
eight = str(data[7]) #assets in dollars we need to extract this
assetsdol= re.findall('>\$(.+?)</td>',eight)
#we keep only the numbers
if assetsdol != empty:
    w = assetsdol[0]
    a = w.replace("[", "")
    ass = a.replace("]","")
    assets_dol.insert(x,ass)
else:
    assets_dol.insert(x,'not available')
ten = str(data[9]) #Total Stockholder Equity ($M)
eleven = str(data[10])
#Total Stockholder Equity ($M) in dollars we need to extract this
tsedol= re.findall('>\$(.+?)</td>',eleven)
#we keep only the numbers
if tsedol != empty:
    w = tsedol[0]
    a = w.replace("[", "")
    ts = a.replace("]","")
    tse_dol.insert(x,ts)
else:
    tse_dol.insert(x,'not available')
```

```
        thirteen = str(data[12]) # market value
        fourteen = str(data[13])
        # market value in dollars we need to extract this
        mardol= re.findall('>\$(.+?)</td>',fourteen)
        #we keep only the numbers
        if mardol != empty:
            w = mardol[0]
            a = w.replace("[", "")
            mar = a.replace("]","")
            mar_dol.insert(x,mar)
        else:
            mar_dol.insert(x,'not␣available')
        x = x + 1
    print "The␣function␣is␣complete!"
```

```
fortune_metrics (list_company_name,list_company_website)
```

```
    The function is complete!
```

```
d9 = {'company' : pd.Series(ln, index=[nm]),
      'Revenues␣$' : pd.Series(rev_dol, index=[nm]),
      'Revenues␣%' : pd.Series(rev_per, index=[nm]),
      'Assets␣$' : pd.Series(assets_dol, index=[nm]),
      'Total␣Stockholder␣Equity␣$' : pd.Series(tse_dol, index=[nm]),
      'Market␣value␣$' : pd.Series(mar_dol, index=[nm])}
fort500 = pd.DataFrame(d9)
fort500.head(3)
```

2.4.12 Merge data-frames and extract final csv file

The final step of the code is to create a final data frame which will include all the information gathered and then create a csv file that will be used for the further analysis that will be take place with the use of the program language R.

```
fort500.merge(html_val, left_on='company', right_on='company',
how='outer')
fort500.head(3)
result = pd.merge(fort500, html_val, how='inner', on=['company', 'company'])
result2 = pd.merge(social_media, fre, how='inner', on=['company', 'company'])
```

```
result3 = pd.merge(wordss, sizess, how='inner', on=['company', 'company'])
result4 = pd.merge(images_types, loading_time, how='inner',
 on=['company', 'company'])
result5 = pd.merge(result, sites_links, how='inner', on=['company', 'company'])
result6 = pd.merge(result5, result2, how='inner', on=['company', 'company'])
result7 = pd.merge(result6, result3, how='inner', on=['company', 'company'])
final3 = pd.merge(result7, result4, how='inner', on=['company', 'company'])
final3.head(3)
final3.to_csv('total_500.csv', sep=';')

data500 = pd.read_csv("total_500.csv", sep=';')
data500.head(3)
```

3 Data Analysis

Now that we have gathered all the information needed the next step is to proceed to their analysis. For the purposes of this assignment we will use the programming language R to perform this analysis.

3.1 R and R-Studio

R is a programming language and software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis.
Furthermore R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems. While R has a command line interface, there are several graphical front-ends available. For the purpose of this paper we will use the R-Studio which is a free and open-source integrated development environment (IDE) for R.

3.2 Scripts

In this chapter we are going to present the scripts that we used throughout the analysis and we will explain elaborately how we reached to the conclusions. The analysis that we are going to perform will be explained by the following order as we performed it:

1. Data cleansing
2. Variable analysis and correlation
   (a) Fortune variables correlation
   (b) Social media analysis and correlation
   (c) Links analysis and correlations
   (d) Words analysis and correlation
   (e) HTML validation variables analysis and correlation
   (f) Image types analysis and correlation
3. Data manipulation
   (a) Image sizes variables reconstruction
   (b) Remove variables
4. Logistic Regression
   (a) Training and test set creation
   (b) Null and Full model creation
   (c) Lasso Method

    (d) Both Method
    (e) Predictions and comparison of models

5. Comparisons and other methods
    (a) Correlation testing
    (b) Clustering testing
    (c) Final testing
    (d) Final model

3.2.1 Data cleansing

The first step in order to perform the analysis is to upload the final csv we have created and then change any miss fitted value or type of variable so as to be easier for us to examine them.

3.2.2 Variable analysis and correlation

Now that the variables are cleansed we can perform the first step of the analysis. We should see how the prices are distributed and get a first look of what we are examining.

3.2.2.1 Fortune variables correlation

The first step is to determine the variable with which we will compare all the other ones. Obviously this variable will be one of the metrics we downloaded from the fortune 500 site so as to have an actual variable that shows the status of the company and compare the rest of the variables with it.

3.2.2.2 Social media analysis and correlation

Now that we have reach to the conclusion that the metric we will use from the Fortune 500 is the Revenues we will proceed with analysing the social media variables to see how they are distributed and then to see how they are correlated with the chosen variable.

3.2.2.3 Links analysis and correlations

Next step is to analyse the distribution of the internal, external and total links and of course their correlation with each other and with the Revenues.

3.2.2.4 Words analysis and correlation

Continuing with the analysis of the unique words and the total words with correlation to the readability index.

3.2.2.5 HTML validation variables analysis and correlation

During the extraction of the html validation variables we created 4 different metrics: the number of errors, the number of warnings the non document error and the page not opened one. In this chapter we are going to analyse the distributions of those variables and also their correlation with the Revenues.

3.2.2.6 Image types analysis and correlation

Last but not least we should analyse the distribution of the different types of images that are being used on the websites that we are examining and also compare the total images with the Revenues to see the correlation.

3.2.3 Data manipulation

In this chapter we are going to change some of the variables that we have created in order to make them more easy to be analysed and explained in relationship to the Revenues.

3.2.3.1 Image sizes variables reconstruction

After implementing the python code for the size images (2.4.10) we created almost 700 different variables. This number is almost prohibited when it comes to regression models especially when the actual number of the examined cases are a lot fewer (500). That is why we decide to group this 700 variables to 5 new ones that we will be able to incorporate to this analysis.

3.2.3.2 Remove variables

Now that we have created and analysed all the needed variables we can subtract from the data frame the variables that we have decide not to include in the regression analysis that will follow.

3.2.4 Logistic Regression

In this chapter we will perform logistic regression to the chosen data frame in order to determine the variables that affect most the price of the Revenues of a company

3.2.4.1 Training and test set creation

Before we begin with the regression we have to divide the data frame to 2 data frames so as to be able to test the model we will create. We will make a training set and a test set.

3.2.4.2 Null and Full model creation

The first step is by using the training set to create a null model (that includes only the Revenues) and a full model (that includes all the variables)

3.2.4.3 Lasso Method

The next step is to implement the Lasso method

3.2.4.4 Both Method

We will continue by applying the both method of the logistic regression in order to see if the variables that will be kept are the same or less than the lasso method.

3.2.4.5 Predictions and comparison of models

Now we will make the predictions using the test set in order to compare how well do the model that we created work in practise.

3.2.5 Comparisons and other methods

Now that we have a first idea of the metrics that are considered more important it would be wise to use other methods to double check the results.

3.2.5.1 Correlation testing

By seeing the correlations of the chosen variables we could create models by subtracting some of them to see if the results will be improved.

3.2.5.2 Clustering testing

We will use the clustering method to see how the variables are being group together. And which variables play the most important role.

3.2.5.3 Final testing

Now we will compare the results to see if we can test any other theory so as to be sure about the final results.

3.2.5.4 Final model

After completing the analysis we can conclude that the most important variable of the websites that can play a crucial role to the actual revenues of a company are the images. More specifically the sizes and the type of the images along with the external links.

4 Conclusions

This conclusion can be easily explained from the fact that the eye is drowned to specific stimulations. There are many studies that shows that where the eye is going in the first seconds that a user is visiting a page can be crucial. So the right type of image with a correct structure and use of specific image sizes can make a site more likeable and friendly to the users.
Example - Testing of where does the eye drops first
Example - Images and the importance to the psychology
Example - Type of images more easily and rapid loading
Example - Information that the user is looking for not confined only inside the same website

5 Bibliography

References

[1] $https://en.wikipedia.org/wiki/Fortune_500$

[2] $http://beta.fortune.com/fortune500$

[3] $http://www.tablesgenerator.com/$

[4] $https://www.continuum.io/downloads$

[5] $https://validator.w3.org/$

[6] Rui Miguel Forte,Mastering Predictive Analytics with R,Packt Publishing Ltd.,June 2015

A Appendix A: Fortune 500 Companies

Table 3: Fortune 500 - Companies Ranked: 51 - 100

51. Intel	52. Humana	53. Disney
54. Cisco Systems	55. Pfizer	56. Dow Chemical
57. Sysco	58. FedEx	59. Caterpillar
60. Lockheed Martin	61. N.Y. Life Insurance	62. Coca-Cola
63. HCA Holdings	64. Ingram Micro	65. Energy Transfer Equity
66. Tyson Foods	67. American Airlines Group	68. Delta Air Lines
69. Nationwide	70. Johnson Controls	71. Best Buy
72. Merck	73. Liberty Mutual I.G.	74. Goldman Sachs Group
75. Honeywell International	76. Massachusetts Mutual L.I.	77. Oracle
78. Morgan Stanley	79. Cigna	80. U.C. Holdings
81. Allstate	82. TIAA	83. INTL FCStone
84. CHS	85. American Express	86. Gilead Sciences
87. Publix Super Markets	88. General Dynamics	89. TJX
90. ConocoPhillips	91. Nike	92. World Fuel Services
93. 3M	94. Mondelez International	95. Exelon
96. Twenty-First Century Fox	97. Deere	98. Tesoro
99. Time Warner	100. Northwestern Mutual	

Table 4: Fortune 500 - Companies Ranked: 101 - 150

101. DuPont	102. Avnet	103. Macy's
104. Enterprise Products Partners	105. Travelers Cos.	106. Philip Morris International
107. Rite Aid	108. Tech Data	109. McDonald's
110. Qualcomm	111. Sears Holdings	112. Capital One Financial
113. EMC	114. USAA	115. Duke Energy
116. Time Warner Cable	117. Halliburton	118. Northrop Grumman
119. Arrow Electronics	120. Raytheon	121. Plains GP Holdings
122. US Foods Holding	123. AbbVie	124. Centene
125. Community Health Systems	126. Alcoa	127. International Paper
128. Emerson Electric	129. Union Pacific	130. Amgen
131. U.S. Bancorp	132. Staples	133. Danaher
134. Whirlpool	135. Aflac	136. AutoNation
137. Progressive	138. Abbott Laboratories	139. Dollar General
140. Tenet Healthcare	141. Eli Lilly	142. Southwest Airlines
143. Penske Automotive Group	144. ManpowerGroup	145. Kohl's
146. Starbucks	147. Paccar	148. Cummins
149. Altria Group	150. Xerox	

Table 5: Fortune 500 - Companies Ranked: 151 - 200

151. Kimberly-Clark	152. Hartford F.S.G.	153. Kraft Heinz
154. Lear	155. Fluor	156. AECOM
157. Facebook	158. Jabil Circuit	159. CenturyLink
160. Supervalu	161. General Mills	162. Southern
163. NextEra Energy	164. Thermo Fisher Scientific	165. American Electric Power
166. PG&E Corp.	167. NGL Energy Partners	168. Bristol-Myers Squibb
169. Goodyear Tire & Rubber	170. Nucor	171. PNC F.S.G.
172. Health Net	173. Micron Technology	174. Colgate-Palmolive
175. Freeport-McMoRan	176. ConAgra Foods	177. Gap
178. Baker Hughes	179. Bank of N.Y. Mellon C.	180. Dollar Tree
181. Whole Foods Market	182. PPG Industries	183. Genuine Parts
184. Icahn Enterprises	185. Performance Food Group	186. Omnicom Group
187. DISH Network	188. FirstEnergy	189. Monsanto
190. AES	191. CarMax	192. National Oilwell Varco
193. NRG Energy	194. Western Digital	195. Marriott International
196. Office Depot	197. Nordstrom	198. Kinder Morgan
199. Aramark	200. DaVita HealthCare Partners	

Table 6: Fortune 500 - Companies Ranked: 201 - 250

201. Molina Healthcare	202. WellCare Health Plans	203. CBS
204. Visa	205. Lincoln National	206. Ecolab
207. Kellogg	208. C.H. Robinson Worldwide	209. Textron
210. Loews	211. Illinois Tool Works	212. Synnex
213. Viacom	214. HollyFrontier	215. Land O'Lakes
216. Devon Energy	217. PBF Energy	218. Yum Brands
219. Texas Instruments	220. CDW	221. Waste Management
222. Marsh & McLennan	223. Chesapeake Energy	224. Parker-Hannifin
225. Occidental Petroleum	226. Guardian Life I.C.A.	227. Farmers Ins. Exchange
228. J.C. Penney	229. Consolidated Edison	230. Cognizant Tech. Solutions
231. VF	232. Ameriprise Financial	233. Computer Sciences
234. L Brands	235. Jacobs Engineering Group	236. Principal Financial
237. Ross Stores	238. Bed Bath & Beyond	239. CSX
240. Toys R Us	241. Las Vegas Sands	242. Leucadia National
243. Dominion Resources	244. United States Steel	245. L-3 Communications
246. Edison International	247. Entergy	248. ADP
249. First Data	250. BlackRock	

Table 7: Fortune 500 - Companies Ranked: 251 - 300

251. WestRock	252. Voya Financial	253. Sherwin-Williams
254. Hilton Worldwide Holdings	255. R.R. Donnelley & Sons	256. Stanley Black & Decker
257. Xcel Energy	258. Murphy USA	259. CBRE Group
260. D.R. Horton	261. Estee Lauder	262. Praxair
263. Biogen	264. State Street Corp.	265. Unum Group
266. Reynolds American	267. Group 1 Automotive	268. Henry Schein
269. Hertz Global Holdings	270. Norfolk Southern	271. Reinsurance G. of America
272. Public Service E. G.	273. BB&T Corp.	274. DTE Energy
275. Assurant	276. Global Partners	277. Huntsman
278. Becton Dickinson	279. Sempra Energy	280. AutoZone
281. Navistar International	282. Precision Castparts	283. Discover F. S.
284. Liberty Interactive	285. W.W. Grainger	286. Baxter International
287. Stryker	288. Air Products & Chemicals	289. Western Refining
290. Universal Health Services	291. Owens & Minor	292. Charter Communications
293. Advance Auto Parts	294. MasterCard	295. Applied Materials
296. Eastman Chemical	297. Sonic Automotive	298. Ally Financial
299. CST Brands	300. eBay	

Table 8: Fortune 500 - Companies Ranked: 301 - 350

301. Lennar	302. GameStop	303. Reliance Steel & Aluminum
304. Hormel Foods	305. Celgene	306. Genworth Financial
307. PayPal Holdings	308. Priceline Group	309. MGM Resorts International
310. Autoliv	311. Fidelity National Financial	312. Republic Services
313. Corning	314. Peter Kiewit Sons'	315. Univar
316. Mosaic	317. Core-Mark Holding	318. Thrivent F. for Lutherans
319. Cameron International	320. HD Supply Holdings	321. Crown Holdings
322. EOG Resources	323. Veritiv	324. Anadarko Petroleum
325. Laboratory C. of A.	326. Pacific Life	327. News Corp.
328. Jarden	329. SunTrust Banks	330. Avis Budget Group
331. Broadcom	332. American Family I. G.	333. Level 3 Communications
334. Tenneco	335. United Natural Foods	336. Dean Foods
337. Campbell Soup	338. Mohawk Industries	339. BorgWarner
340. PVH	341. Ball	342. O'Reilly Automotive
343. Eversource Energy	344. Franklin Resources	345. Masco
346. Lithia Motors	347. KKR	348. Oneok
349. Newmont Mining	350. PPL	

Table 9: Fortune 500 - Companies Ranked: 351 - 400

351. SpartanNash	352. Quanta Services	353. XPO Logistics
354. Ralph Lauren	355. Interpublic Group	356. Steel Dynamics
357. WESCO International	358. Quest Diagnostics	359. Boston Scientific
360. AGCO	361. Foot Locker	362. Hershey
363. CenterPoint Energy	364. Williams	365. Dick's Sporting Goods
366. Live Nation Entertainment	367. Mutual of Omaha Ins.	368. W.R. Berkley
369. LKQ	370. Avon Products	371. Darden Restaurants
372. Kindred Healthcare	373. Weyerhaeuser	374. Casey's General Stores
375. Sealed Air	376. Fifth Third Bancorp	377. Dover
378. Huntington Ingalls Industries	379. Netflix	380. Dillard's
381. EMCOR Group	382. Jones Financial	383. AK Steel Holding
384. UGI	385. Expedia	386. salesforce.com
387. Targa Resources	388. Apache	389. Spirit AeroSystems H.
390. Expeditors Inter. of Washington	391. Anixter International	392. Fidelity N. Inf. S.
393. Asbury Automotive Group	394. Hess	395. Ryder System
396. Terex	397. Coca-Cola Eur. P.	398. Auto-Owners Insurance
399. Cablevision Systems	400. Symantec	

Table 10: Fortune 500 - Companies Ranked: 401 - 450

401. Charles Schwab	402. Calpine	403. CMS Energy
404. Alliance Data Systems	405. JetBlue Airways	406. Discovery Communic.
407. Trinity Industries	408. Sanmina	409. NCR
410. FMC Technologies	411. Erie Insurance Group	412. Rockwell Automation
413. Dr Pepper Snapple Group	414. iHeartMedia	415. Tractor Supply
416. J.B. Hunt Transport Services	417. Commercial Metals	418. Owens-Illinois
419. Harman Inter. Ind.	420. Baxalta	421. American F. G.
422. NetApp	423. Graybar Electric	424. Oshkosh
425. Ameren	426. A-Mark Precious Metals	427. Barnes & Noble
428. Dana Holding	429. Constellation Brands	430. LifePoint Health
431. Zimmer Biomet H.	432. Harley-Davidson	433. PulteGroup
434. Newell Brands	435. Avery Dennison	436. Jones Lang LaSalle
437. WEC Energy Group	438. Marathon Oil	439. TravelCenters of A.
440. United Rentals	441. HRG Group	442. Old Republic Inter.
443. Windstream Holdings	444. Starwood Hotels & Resorts	445. Delek US Holdings
446. Packaging Corp. of A.	447. Quintiles Transnational H.	448. Hanesbrands
449. Realogy Holdings	450. Mattel	

Table 11: Fortune 500 - Companies Ranked: 451 - 500

451. Motorola Solutions	452. J.M. Smucker	453. Regions Financial
454. Celanese	455. Clorox	456. Ingredion
457. Genesis Healthcare	458. Peabody Energy	459. Alaska Air Group
460. Seaboard	461. Frontier Communic.	462. Amphenol
463. Lansing Trade Group	464. SanDisk	465. St. Jude Medical
466. Wyndham Worldwide	467. Kelly Services	468. Western Union
469. Envision Healthcare H.	470. Visteon	471. Arthur J. Gallagher
472. Host Hotels & Resorts	473. Ashland	474. Insight Enterprises
475. Energy Future Holdings	476. Markel	477. Essendant
478. CH2M Hill	479. Western & Southern F.G.	480. Owens Corning
481. S&P Global	482. Raymond James Financial	483. NiSource
484. Airgas	485. ABM Industries	486. Citizens F.G.
487. Booz Allen Hamilton H.	488. Simon Property Group	489. Domtar
490. Rockwell Collins	491. Lam Research	492. Fiserv
493. Spectra Energy	494. Navient	495. Big Lots
496. Telephone & Data Systems	497. First American Financial	498. NVR
499. Cincinnati Financial	500. Burlington Stores	