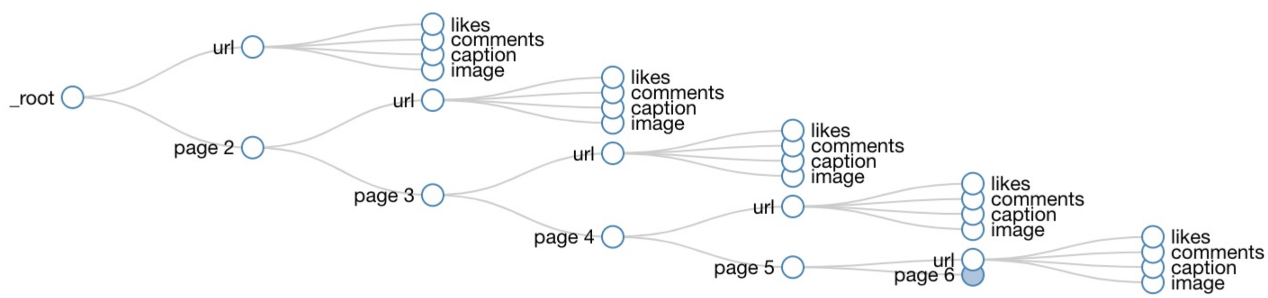
National Geographic - Instagram and image analytics

01. Scraping Instagram and collecting posts

I used a dedicated browser scraper extension for extracting image urls and corresponding likes and comments from the National Geographic's (NatGeo) instagram page. Since this scraper does not automatically scroll infinitely on the official instagram application, I used a secondary website 'picbon.com' which is an exact reflection of the instagram application and allows auto infinite scrolling while scraping.



02. Calculating Engagement

In [176]:
```python
import pandas as pd
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', 10000)
```

After scrapping posts from instagram I came up with a list of 634 posts, including information about the number of likes and comments realted to each post, the caption (description of the post) and the url of the post's image.

In [177]:
```python
instagram_posts = pd.read_csv('instagram.csv')
```

In [178]:
```
instagram_posts.head()[:1]
```

Out[178]:

	likes	comments	caption	
0	444841	2107	Photo By @BrianSkerry\nA West Indian Manatee calf nurses from its mom as they settle down on the sandy sea floor in the waters off the coast of Belize, an important country for these endangered animals. Manatees here live in mangroves and often sleep there overnight and feed on nearby sea grass beds during the day. Unlike Florida manatees, these animals are not nearly as acclimated to humans and are shyer. On this morning however, this very relaxed mom and calf allowed me into their world.\nFor more images and stories about ocean wildlife follow @BrianSkerry\n#manatees #belize #mesoamericanreef #endangeredspecies	1.cdninstagram.com/vp/3a1c818472deb760c090c7f3 15/fr/e15/s1080x1080/39625475_248078652711752_6 ig_cache_key=MTg1MzAwNjU3Nz

Using the above collected urls and the Google Vision API, I came up with a list of labels for each post, while using the number of likes and comments I calculated engagement values for each post using the below formula:

$$EngagementValue = 0.4\frac{likes}{maxLikes} + 0.6\frac{comments}{maxComments}$$

Using this formula and the overall median engagement value of all the posts, I set the "Engagement" parameter as 1 (there was engagement related to a post) for all posts that had EngagementValue above or equal to the median and 0 for the rest of the posts, classifying all our posts into two discrete categories.

It was important to take into account the relevant engagement of each post, identifying the posts that were popular in relation to the rest of the National Geographic posts.

In [179]:
```
import statistics
import csv
from google.oauth2 import service_account
from google.cloud import vision
```

```
SCRAPED_POSTS_FILE_PATH = 'instagram.csv'
MAX_POSTS = 634

class InstaPost:
    def __init__(self, nlikes, ncomments, url='', caption='', labels=
        self.url = url
        self.caption = caption
        self.nlikes = nlikes
        self.ncomments = ncomments
        self.labels = labels
        self.engagement = engagement


def calc_engagement(instaposts):
    max_likes = max(instaposts,key=lambda post: post.nlikes).nlikes
    max_comments = max(instaposts, key=lambda post: post.ncomments).n
    for i in range(0,len(instaposts)):
        instaposts[i].nlikes = (instaposts[i].nlikes * 1.0) / max_lik
        instaposts[i].ncomments = (instaposts[i].ncomments * 1.0) / m
        instaposts[i].engagement = (4 * instaposts[i].nlikes) + (6 *
        instaposts[i].engagement_dec = (4 * instaposts[i].nlikes) + (
    median_engagement = statistics.median(map(lambda post: post.engag
    for post in instaposts:
        post.engagement = 1 if post.engagement >= median_engagement e


def get_posts_attrs(instaposts):
    credentials = service_account.Credentials.from_service_account_fi
    client = vision.ImageAnnotatorClient(credentials=credentials)
    image = vision.types.Image()
    for post in instaposts:
        image.source.image_uri = post.url
        response = client.label_detection(image=image)
        post.labels = [label.description for label in response.label_


def import_scraped_posts(file_path):
    post_counter = 0
    instaposts = []
    with open(file_path,'r', encoding="utf8") as csv_file:
        csv_reader = csv.reader(csv_file, quotechar='"', delimiter=',
        next(csv_reader)
        for row in csv_reader:
            if post_counter > MAX_POSTS:
                break
            post = InstaPost(int(row[0]),int(row[1]),row[3],row[2])
            instaposts.append(post)
            post_counter += 1
    return instaposts
```

```

###Testing only###
#post1 = InstaPost(20, 30, 'https://www.gettyimages.com/gi-resources/
#post2 = InstaPost(30, 40, 'https://i1.wp.com/thefreshimages.com/wp-c
#instaposts = [post1,post2]

#calc_engagement(instaposts)
#get_posts_attrs(instaposts)

#print(instaposts[0].engagement)
#print(instaposts[1].engagement)
#print(instaposts[0].labels)
#print(instaposts[1].labels)

instaposts = import_scraped_posts(SCRAPED_POSTS_FILE_PATH)
calc_engagement(instaposts)
get_posts_attrs(instaposts)
for post in instaposts:
    print(post.ncomments,post.nlikes,post.engagement,post.labels)

#################


```

```
, 'wood', 'animal source foods']
0.19176983035304906 0.6878308130727165 1 ['surgeon', 'operating theate
r', 'service', 'medical', 'hospital']
0.012408298945437872 0.2946170420115282 1 ['sea', 'coastal and oceanic
landforms', 'body of water', 'ocean', 'shore', 'sky', 'horizon', 'coas
t', 'wave', 'water']
0.006189821182943604 0.15481301267366243 0 ['bird', 'seabird', 'albatr
oss', 'water', 'sea', 'wave', 'ocean', 'fauna', 'beak', 'wind wave']
0.025876891334250345 0.23144040515075376 1 ['climbing', 'rock climbing
', 'rock', 'sport climbing', 'adventure', 'tree', 'outdoor recreation'
, 'rock climbing equipment', 'outcrop', 'free climbing']
0.006476386978450252 0.13288870824711793 0 ['bazaar', 'public space',
'marketplace', 'market', 'shopkeeper', 'city']
0.015780223139232767 0.40485458080845405 1 ['wildlife', 'lion', 'mamma
l', 'fauna', 'terrestrial animal', 'masai lion', 'big cats', 'zoo', 'o
rganism', 'cat like mammal']
1.0 0.8895702085796631 1 ['rhinoceros', 'mammal', 'wildlife', 'terrest
rial animal', 'fauna', 'horn', 'eye', 'snout', 'organism', 'grass']
0.006973101023995109 5 0.08569559654892107 0 ['fun', 'recreation', 'gir
l', 'leisure']
```

In [222]:

```
all_posts = [(post.ncomments, post.nlikes, post.labels) for post in
df_posts = pd.DataFrame(all_posts, columns=['ncomments', 'nlikes', '
df_posts['caption'] = instagram_posts.iloc[:]['caption']
df_posts['engagement'] = [post.engagement for post in instaposts]
df_posts['engagement_dec']  = [post.engagement_dec for post in insta
```

In [223]:
```
df_posts[:1]
```

Out[223]:

	ncomments	nlikes	labels	caption	engagement	engagement_dec
0	0.020126	0.256822	[marine mammal, fauna, water, mammal, marine biology, underwater, manatee, organism, wildlife, dugong]	Photo By @BrianSkerry\nA West Indian Manatee calf nurses from its mom as they settle down on the sandy sea floor in the waters off the coast of Belize, an important country for these endangered animals. Manatees here live in mangroves and often sleep there overnight and feed on nearby sea grass beds during the day. Unlike Florida manatees, these animals are not nearly as acclimated to humans and are shyer. On this morning however, this very relaxed mom and calf allowed me into their world.\nFor more images and stories about ocean wildlife follow @BrianSkerry\n#manatees #belize #mesoamericanreef #endangeredspecies	1	1.148048

In [224]:
```
df_posts.to_csv('all_data.csv')
```

03. Logistic Regression using image labels and engagement

Having collected and preprocessed my data, I decided to run a logistic regression model to make predictions and identify the key features of a successful image post. But before I could do that, I first created a list of the features, which in this case were the unique labels from all the posts.

In [225]:
```
all_data = pd.read_csv('all_data.csv')
```

In [184]:
```
# Getting distinct labels to convert it into features
import ast

labels_list=[]

for post_labels in all_data.iloc[:]['labels']:
     labels_list += ast.literal_eval(post_labels)
label_set=set(labels_list)
label_set=sorted(list(label_set))

print('The total number of features is: '+ str(len(label_set)))
```

```
The total number of features is: 913
```

Having set the list of features, we created a dataframe indicating the features that appear in every post along with the engagement related to the post.

In [185]:
```
import pandas as pd
import numpy as np

# creating an empty dataframe
labels = [ast.literal_eval(post_labels) for post_labels in all_data.

colmns = label_set
df = pd.DataFrame(0.0, index = np.arange(len(labels)), columns = col
df['Engagement'] = all_data.iloc[:]['engagement']
```

In [186]:
```
import math
from textblob import TextBlob as tb

def tf(word, blob):
    return blob.words.count(word) / len(blob.words)

def n_containing(word, bloblist):
    return sum(1 for blob in bloblist if word in blob.words)

def idf(word, bloblist):
    return math.log(len(bloblist) / (1 + n_containing(word, bloblist

def tfidf(word, blob, bloblist):
    return tf(word, blob) * idf(word, bloblist)
```

In [187]:

```
# make label_sent which is a list that contains all the labels conca
label_sent = []

for i in range(len(labels)):
    label_sent.append(" ".join(labels[i]))
```

In [188]:

```
# convert the post list to textblob format for passing into tf-idf f
labels_tb = []

for labels in label_sent:
    labels_tb.append(tb(labels))
```

In [189]:

```
bloblist = labels_tb

tfidf_list = []

# list containing all the labels with tfidf score

for i, blob in enumerate(bloblist):
    scores = {word: tfidf(word, blob, bloblist) for word in blob.wor
    tfidf_list.append(scores)

```

In [190]:

```
#Assigning value of feature according to the label set of each post

data_ind = 0
for index, row in df.iterrows():
    for column in df:
        if column in tfidf_list[data_ind]:
#               print(index,column,tfidf_list[data_ind][column])

                df.at[index,column] = tfidf_list[data_ind][column]
    data_ind=data_ind+1
```

In [191]:

```
columns=['Post']+label_set
#Divide data set into Features and Outcome
X=df[label_set]
y = df['Engagement']
```

In [192]:
```
import matplotlib
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

#Divide into test and train data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=0)

#Logistic Regression
lr = LogisticRegression(C=2, random_state=42)
lr.fit(X_train, y_train)
```

Out[192]:
```
LogisticRegression(C=2, class_weight=None, dual=False, fit_intercept=T
rue,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs
=1,
          penalty='l2', random_state=42, solver='liblinear', tol=0.000
1,
          verbose=0, warm_start=False)
```

In [193]:
```
#Predicting test outcome
y_pred = lr.predict(X_test)

#Calculate Accuracy
print('Accuracy: %.2f' % accuracy_score(y_test,y_pred))

#####Extra part to calculate probability of prediction
# y_pred = lr.predict_proba(X_test)
# from sklearn.metrics import roc_curve, auc, roc_auc_score
# false_positive_rate, true_positive_rate, thresholds = roc_curve(y_
# print (auc(false_positive_rate, true_positive_rate))
# print (roc_auc_score(y_train, lr.predict(X_train)))
```

```
Accuracy: 0.74
```

In [194]:
```
#Confusion Matrix
cnf_matrix = confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[194]:
```
array([[64, 32],
       [17, 78]], dtype=int64)
```

The small accuracy of the model was mostly attributed to the false positive values (based on the confusion matrix), while I saw that it improved drastically with the introduction of more data. Indicatively, with the use of 450 posts was only getting an accuracy slightly above 50%, while the above was a result of using 634.

04. Logistic Regression using captions and engagement

Not being satisfied with my results, my second attempt included the captions instead of the Google vision API image labels. In order to complete the regression this time, I first ran through the captions and made the relevant cleaning (removing stop words) and replacements in order to optimize my results.

In [195]:
```
from nltk import pos_tag
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from collections import Counter
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\lovek\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[195]: True

In [196]:
```
def count_words(dataframe,column):
    captions = []
    for i in range(dataframe.shape[0]):
        captions.append(pd.DataFrame(dataframe.iloc[:][column]).loc[
    wordcount = Counter(pos_tag(word_tokenize(''.join(captions))))
    word_list = sorted(list(wordcount.items()), key = lambda w: -w[1
    stoplist = nltk.corpus.stopwords.words('english')
    word_list = [word_list[i] for i in range(len(word_list))
                if len(word_list[i][0][0]) > 2 and word_list[i][0][
                and word_list[i][0][1] in ['NN', 'NNP', 'NNS', 'JJ'
    return word_list
```

In [197]:
```
word_list = count_words(instagram_posts, 'caption')
len(word_list)
```

Out[197]: 9453

In [198]:
```
animal = ['species', 'wildlife', 'animals', 'elephants', 'lions', 'b
```

```
           'birds', 'lion', 'ivory', 'bear', 'pride', 'habitat', 'fis
           'jaguar', 'horses', 'snake', 'animal', 'dragon', 'lewa_wil
           'whale', 'penguins', 'bull', 'rhinos', 'horse', 'flamingo'
           'tortoises', 'rhinos', 'cat', 'tigers', 'insects', 'goat',

human = ['people', 'work', 'family', 'women', 'children', 'refugee',
         'mother', 'camp', 'men', 'girls', 'rohingya', 'humans', 'ki
         'son', 'baby', 'tourists', 'rangers', 'friends', 'father',
         'families', 'woman', 'wife', 'residents', 'brother', 'poach
         'americans', 'president', 'survivor', 'kid', 'chancellor',
         'visitors']

photographer = ['timlaman', 'stephenwilkes', 'stevewinterphoto', 'mi
                'gabrielegalimbertiphoto', 'williamalbertallard', 'c
                'renaeffendiphoto', 'jimmy_chin', 'muhammedmuheisenp
                'mmuheisen', 'george', 'paleyphoto', 'muheisen', 'ca
                'hammond_robin', 'beverlyjoubert', 'brianskerry', 'p
                'christineeckstrom', 'salvarezphoto', 'florianschulz
                'katieorlinsky', 'davidalanharvey', 'ljohnphoto', 'p

place = ['myanmar', 'china', 'india', 'bangladesh', 'peru', 'borneo'
         'california', 'matera', 'afghanistan', 'africa', 'capital',
         'alaska', 'indonesia', 'madagascar', 'nation', 'greenland',
         'ghana', 'france', 'north', 'south', 'east', 'west', 'greec
         'kenya', 'japan', 'florida']

advertisment = ['africanparksnetwork', 'samsungmobileusa', 'withgala

nature = ['water', 'river', 'sea', 'ice', 'place', 'image', 'area',
          'coast', 'mountain', 'mountains', 'ocean', 'tree', 'island
          'rivers', 'cannabis', 'rock', 'waters', 'forest', 'dunes',
          'environment', 'everglades', 'trees', 'garden', 'plants',
          'field', 'sky', 'plains', 'rain', 'ecosystems', 'lakes', '
          'glacier', 'socotra', 'flames', 'storm', 'fog', 'tide', 'o
          'natureern', 'naturen', 'natures']

civilization = ['city', 'park', 'country', 'war', 'village', 'commun
                'border', 'crisis', 'school', 'culture', 'boat', 'ho
                'market', 'toys', 'education', 'cities', 'york', 'po
                'research', 'prayer', 'organizations', 'conflict', '
                'hotel', 'toy', 'poverty', 'cultures', 'music', 'art
                'schools', 'motel', 'bridge', 'law', 'street', 'room

time = ['years', 'time', 'day', 'night', 'days', 'today', 'year', 'c
        'summer', 'months', 'future', 'hours', 'times', 'august', 'w
        'monument', 'weeks', 'month', 'decade', 'moments', 'season',
        'weekend', 'spring']

size = ['big', 'small', 'tiny', 'large', 'long', 'high', 'massive',
        'narrow', 'thin', 'enormous', 'tall', 'heavy', 'distant', 'i
        'hundreds', 'temperatures', 'half', 'size', 'distance', 'len
```

```

different = ['new', 'great', 'old', 'different', 'important', 'tradi
             'perfect', 'popular', 'dangerous', 'famous', 'dramatic'
             'private', 'successful', 'endangered', 'valuable', 'ori
             'vulnerable', 'vital', 'spiritual', 'distinct', 'devast

nice = ['beautiful', 'incredible', 'amazing', 'magical', 'sweet', 'w
        'stunning', 'excellent']
```

In [199]:

```
def less_caption_words(dataframe,column):
#   creating list of captions with lower cases
    captions = []
    for i in range(dataframe.shape[0]):
        captions.append(pd.DataFrame(dataframe.iloc[:][column]).loc[

#   introducing replacements
    word_lists = [animal, human, photographer, place, advertisment,
                  different, nice]
    replacement_words = ['animal', 'human', 'photographer', 'place',
                         'time', 'size', 'different', 'nice']

#   replacing words and creating new captions
    simple_captions = []
    for c in captions:
        for i in range(len(replacement_words)):
            for word in word_lists[i]:
                c = c.replace(word, replacement_words[i])
        simple_captions.append(c)
    token_captions = [word_tokenize(c) for c in simple_captions]
#   counting words using the simplified captions with the replaced w
    wordcount = Counter(pos_tag(word_tokenize(''.join(simple_caption
    word_list = sorted(list(wordcount.items()), key = lambda w: -w[1
    stoplist = nltk.corpus.stopwords.words('english')
    word_list = [word_list[i] for i in range(len(word_list))
                 if len(word_list[i][0][0]) > 2 and word_list[i][0][
                 and word_list[i][0][1] in ['NN', 'NNP', 'NNS', 'JJ'
    return [simple_captions, token_captions, word_list]
```

In [200]:

```
[simple_captions, token_captions, new_word_list] = less_caption_word
```

In [201]:

```
# creating an empty dataframe
wrds = [new_word_list[i][0][0] for i in range(len(new_word_list))]
engmnt = list(all_data['engagement'])
cllmns = wrds+['Engagement']
df_caption = pd.DataFrame(0.0, index = np.arange(len(token_captions)
df_caption['Engagement'] = ([engmnt[i] for i in range(len(token_capt
```

In [202]:
```
df_caption.shape
```

Out[202]: (634, 8894)

As seen through the extensive word replacements, I achieved the reduction of the captions' count by 500 words, reducing the complexity of the model and the features sacrificing only part of its accuracy of information.

In [203]:
```
# convert the post list to textblob format for passing into tf-idf f
captions_tb = []

for caption in list(simple_captions):
    captions_tb.append(tb(caption))
```

In [204]:
```
# import time
# t0 = time.time()

caption_tfidf_list = []

# list containing all the captions with tfidf score

for i, blob in enumerate(captions_tb):
    scores = {word: tfidf(word, blob, captions_tb) for word in blob.
    caption_tfidf_list.append(scores)

# t1 = time.time()
# print(t1-t0)
```

In [205]:
```
#Assigning value of feature according to the caption set of each pos

caption_data_ind = 0
for index, row in df_caption.iterrows():
    for column in df_caption:
        if column in caption_tfidf_list[caption_data_ind]:
            df_caption.loc[index, column] = caption_tfidf_list[capti
    caption_data_ind=caption_data_ind+1
```

In [206]:
```
#Divide data set into Features and Outcome
XX = df_caption[wrds]
yy = df_caption['Engagement']
```

In [207]:
```
import matplotlib
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

#Divide into test and train data
XX_train, XX_test, yy_train, yy_test = train_test_split(XX, yy, test

#Logistic Regression
caption_lr = LogisticRegression(C=3, random_state=0)
caption_lr.fit(XX_train, yy_train)
```

Out[207]:
```
LogisticRegression(C=3, class_weight=None, dual=False, fit_intercept=T
rue,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs
=1,
          penalty='l2', random_state=0, solver='liblinear', tol=0.0001
,
          verbose=0, warm_start=False)
```

In [208]:
```
#Predicting test outcome
yy_pred = caption_lr.predict(XX_test)

#Calculate Accuracy
print('Accuracy: %.2f' % accuracy_score(yy_test,yy_pred))
```

```
Accuracy: 0.73
```

Based on the above, it appeared that the overall accuracy did not improve over the original model. Interstingly enough, it appears that the information collected from the image recognition labeling was of the same value as the thorough description of the image by the photographer.

05. Logistic Regression using (captions, labels) and engagement

In this step, I combined all the information together to a third model making engagement predictions using both image labels and captions.

In [209]:
```
# tfidf values of labels and tfidf values of captions
all_tfidf_table = pd.concat([df.loc[:, :'zoo'], df_caption], axis=1)
all_tfidf_table.shape
```

Out[209]:
```
(634, 9807)
```

In [210]:
```
#Divide data set into Features and Outcome
X_all = all_tfidf_table.loc[:, :'benefits']
y_all = all_tfidf_table['Engagement']
```

In [211]:
```
#Divide into test and train data
X_all_train, X_all_test, y_all_train, y_all_test = train_test_split(

#Logistic Regression
all_lr = LogisticRegression(C=2, random_state=42)
all_lr.fit(X_all_train, y_all_train)
```

Out[211]:
```
LogisticRegression(C=2, class_weight=None, dual=False, fit_intercept=T
rue,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs
=1,
          penalty='l2', random_state=42, solver='liblinear', tol=0.000
1,
          verbose=0, warm_start=False)
```

In [212]:
```
#Predicting test outcome
y_all_pred = all_lr.predict(X_all_test)

#Calculate Accuracy
print('Accuracy: %.2f' % accuracy_score(y_all_test,y_all_pred))
```

```
Accuracy: 0.75
```

While it appeared to be a slight increase in the overall accuracy on our results, the added complexity of the model is not justified. It seems that the information of the image labels and description are somewhat correlated essentially describing the same thing and carrying the same amount of information. The image caption is describing what the google vision API is sumarising in its labels.

06. Clustering images - LDA on image labels

Aiming to identify what makes an image popular and attracts engagement, I used LDA (Latent Dirichlet allocation) on image labels. Running the clustering algorithm multiple times, in order to find the right number of topics that best separated / described my images, I ensured that the topics where clearly distinctive from each other while also their themes were intuitive.

In [232]:
```
import gensim
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from nltk.stem.porter import *
import numpy as np

import nltk
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\lovek\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[232]:
```
True
```

In [233]:
```
post_labels = [ast.literal_eval(post) for post in all_data.iloc[:]['

dict_labels = {}
for i in range(len(post_labels)):
    for j in range(len(post_labels[i])):
        if post_labels[i][j] in dict_labels:
            a = dict_labels[post_labels[i][j]] + 1
            dict_labels[post_labels[i][j]] = a
        else:
            dict_labels[post_labels[i][j]] = 1
```

In [3]:
```
import lda
from sklearn.feature_extraction.text import CountVectorizer
```

In [235]:
```
stopwords_nltk=set(stopwords.words('english'))
```

In [236]:
```
words_freq_vec = CountVectorizer(input=post_labels,stop_words=stopwo
```

In [237]:
```
labels_under_pro = words_freq_vec.fit_transform([' '.join(p) for p i
```

In [238]:
```
ntopics = 4
corpus_arr = np.array(corpus)
model = lda.LDA(n_topics=int(ntopics), n_iter=500, random_state=1)
model.fit(labels_under_pro)
```

```
INFO:lda:n_documents: 634
INFO:lda:vocab_size: 935
INFO:lda:n_words: 6691
INFO:lda:n_topics: 4
INFO:lda:n_iter: 500
C:\Users\lovek\Anaconda3\lib\site-packages\lda\utils.py:55: FutureWarn
ing: Conversion of the second argument of issubdtype from `int` to `np
.signedinteger` is deprecated. In future, it will be treated as `np.in
t32 == np.dtype(int).type`.
  if sparse and not np.issubdtype(doc_word.dtype, int):
INFO:lda:<0> log likelihood: -59701
INFO:lda:<10> log likelihood: -42999
INFO:lda:<20> log likelihood: -41478
INFO:lda:<30> log likelihood: -40933
INFO:lda:<40> log likelihood: -40750
INFO:lda:<50> log likelihood: -40545
INFO:lda:<60> log likelihood: -40366
INFO:lda:<70> log likelihood: -40233
INFO:lda:<80> log likelihood: -40263
INFO:lda:<90> log likelihood: -40169
INFO:lda:<100> log likelihood: -40192
INFO:lda:<110> log likelihood: -40129
INFO:lda:<120> log likelihood: -40141
INFO:lda:<130> log likelihood: -40126
INFO:lda:<140> log likelihood: -40102
INFO:lda:<150> log likelihood: -40085
INFO:lda:<160> log likelihood: -40067
INFO:lda:<170> log likelihood: -39967
INFO:lda:<180> log likelihood: -39998
INFO:lda:<190> log likelihood: -39919
INFO:lda:<200> log likelihood: -39802
INFO:lda:<210> log likelihood: -39783
INFO:lda:<220> log likelihood: -39737
INFO:lda:<230> log likelihood: -39771
INFO:lda:<240> log likelihood: -39702
INFO:lda:<250> log likelihood: -39770
INFO:lda:<260> log likelihood: -39707
INFO:lda:<270> log likelihood: -39636
INFO:lda:<280> log likelihood: -39684
INFO:lda:<290> log likelihood: -39566
INFO:lda:<300> log likelihood: -39618
INFO:lda:<310> log likelihood: -39610
INFO:lda:<320> log likelihood: -39597
INFO:lda:<330> log likelihood: -39545
INFO:lda:<340> log likelihood: -39603
INFO:lda:<350> log likelihood: -39561
INFO:lda:<360> log likelihood: -39566
INFO:lda:<370> log likelihood: -39551
INFO:lda:<380> log likelihood: -39590
INFO:lda:<390> log likelihood: -39487
```

```
INFO:lda:<400> log likelihood: -39519
INFO:lda:<410> log likelihood: -39559
INFO:lda:<420> log likelihood: -39534
INFO:lda:<430> log likelihood: -39526
INFO:lda:<440> log likelihood: -39584
INFO:lda:<450> log likelihood: -39575
INFO:lda:<460> log likelihood: -39623
INFO:lda:<470> log likelihood: -39598
INFO:lda:<480> log likelihood: -39552
INFO:lda:<490> log likelihood: -39612
INFO:lda:<499> log likelihood: -39534
```

Out[238]: <lda.lda.LDA at 0x27187329358>

In [239]:
```
type(labels_under_pro)
# dataframe_label = pd.DataFrame(total_features_words)
dataframe_label = pd.DataFrame({'post_no': range(len(post_labels)),
```

In [240]:
```
topic_label = model.topic_word_
topic_post=model.doc_topic_
topic_post=pd.DataFrame(topic_post)
dataframe_label=dataframe_label.join(topic_post)
Insta_Posts=all_data.iloc[:,[3, 5, 6]]
```

In [241]:
```
all_data[:1]
```

Out[241]:

	Unnamed: 0	ncomments	nlikes	labels	caption	engagement	engag
0	0	0.020126	0.256822	['marine mammal', 'fauna', 'water', 'mammal', 'marine biology', 'underwater', 'manatee', 'organism', 'wildlife', 'dugong']	Photo By @BrianSkerry\r\nA West Indian Manatee calf nurses from its mom as they settle down on the sandy sea floor in the waters off the coast of Belize, an important country for these endangered animals. Manatees here live in mangroves and often sleep there overnight and feed on nearby sea grass beds during the day. Unlike Florida manatees, these animals are not nearly as acclimated to humans and are shyer. On this morning however, this very relaxed mom and calf allowed me into their world.\r\nFor more images and stories about ocean wildlife follow @BrianSkerry\r\n#manatees #belize #mesoamericanreef #endangeredspecies	1	

In [242]:
```
for i in range(int(ntopics)):
    topic="topic_"+str(i)
    Insta_Posts[topic]=dataframe_label.groupby(['post_no'])[i].mean(
```

```
C:\Users\lovek\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
(http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-vi
ew-versus-copy)
  This is separate from the ipykernel package so we can avoid doing im
ports until
```

In [243]:
```
Insta_Posts=Insta_Posts.reset_index()
topics=pd.DataFrame(topic_label)
topics.columns=words_freq_vec.get_feature_names()
topics1=topics.transpose()
print ("Topics label distribution written in file topic_label_dist.x
topics1.to_csv("topic_label_dist.csv")
Insta_Posts.to_csv("Insta_posts_topic_dist.csv",index=False)
print ("Restaurant topic distribution written in file Insta_posts_to
```

```
Topics label distribution written in file topic_label_dist.xlsx
Restaurant topic distribution written in file Insta_posts_topic_dist.x
lsx
```

Analysing influential topics

Based on my analysis and the labels related to each category, I identified that sorting my labels into 4 topics was the most effective way of splitting them, as the labels were split to the below self explanatory topics:

1. Topic I: plants
2. Topic II: human factor
3. Topic III: Animal
4. Topic IV: Scenery

NOTE: The actual words per each topic are listed in the "topic_word_dist.xlsx" file.

In [244]:

```
import csv
posts = []
with open('Insta_posts_topic_dist.csv','r', encoding="utf8") as csv_
    csv_reader = csv.reader(csv_file, quotechar='"', delimiter=',',
    next(csv_reader)
    for row in csv_reader:
        posts.append((float(row[3]),float(row[4]),float(row[5]),floa

posts.sort(key=lambda x: x[4], reverse=True)
quartile_amount = int(len(posts)/4)

top_quartile_topic1_avg = sum(map(lambda x: x[0], posts[:quartile_am
top_quartile_topic2_avg = sum(map(lambda x: x[1], posts[:quartile_am
top_quartile_topic3_avg = sum(map(lambda x: x[2], posts[:quartile_am
top_quartile_topic4_avg = sum(map(lambda x: x[3], posts[:quartile_am

bottom_quartile_topic1_avg = sum(map(lambda x: x[0], posts[-quartile
bottom_quartile_topic2_avg = sum(map(lambda x: x[1], posts[-quartile
bottom_quartile_topic3_avg = sum(map(lambda x: x[2], posts[-quartile
bottom_quartile_topic4_avg = sum(map(lambda x: x[3], posts[-quartile

print('Top quartile topic avgs: ',top_quartile_topic1_avg,top_quarti
print('\n')
print('Bottom quartile topic avgs: ',bottom_quartile_topic1_avg,bott
```

```
Top quartile topic avgs:  1.0242003584503878 0.06629849408088037 0.051
1766861781931 0.023865919085372403


Bottom quartile topic avgs:  1.1922729380081662 0.2750263838998251 0.1
9092801061018513 0.5266541625872856
```

Based on my analysis of the collected data, it appeared that the first topic (related to nature and plants) was prominent in all the images irrespective of their engagement, which was expected given that the images were collected from national geographic's page and thus it did not appear to be the differentiating factor.

On the other hand, the rest of the topics (human factor, animal and scenery) played a crusial role in separating images from mediocre to great and it appeared that more unsuccessful images had twice the amount of "human factor", more than 5 times "animal" related themes, and twice the "scenery" aspect.

07. Advice for National Geographic

Based on my analysis, I would advice the people responsible to keep focusing on natural themes, but rather adding more context to their images enhancing the human factor and somewhat avoiding the plain classic beautiful scenery themes.

In [ ]:

```

```