Over the past decade, **memes** has became very popular in many areas such as computer science, marketing, health care, sociology, technology etc and day by day as people are spending more time over internet, it's popularity is increasing, it's all because it passes informations with a pinch of humor. During this some of the memes becomes viral and there should be a reason for becoming a meme viral.

**In this project I will try to find the reason behind becoming a meme viral and try to predict whether a meme will become viral or not based on data provided for meme and information available in the meme**.

For this project I will collect data from many subreddits of **Reddit**, from largest meme subreddit to lowest one in terms of subscribers, so that we get memes of every flavours. For this project I will use memes in the form of images (no gif and no video) and data available for the post of the meme such as time of post, number of upvotes, size of image etc. After data collection I will frame the problem as binary classification by denoting popular memes based on upvotes and number of subscribers of subreddit as 1 and non-popular meme as 0. Now, this is a binary classification problem.

```
#importing libraries
import pandas as pd
import numpy as np
import datetime as dt
import requests
import matplotlib.pyplot as plt
from matplotlib import pyplot
import seaborn as sns
from tqdm import tqdm
import warnings
warnings.filterwarnings("ignore")
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client

Enter your authorization code:
4/1AY0e-g5sAB80AF2-H_H2dtNhAozFGnQxT7Ljx9Pi9m7MEgRuQiX1_d8qcXo
Mounted at /content/drive
```

# Data Collection

I am going to collect data for this from **Reddit**. Reddit has many subreddits for memes and also Reddit provides an API called **Pushshift API** to easily get data from Reddit. So, I am goind to use

that API to collect data from Reddit.

```
#installing psaw (Pushshift.io API Wrapper)
!pip install psaw
```

```
Collecting psaw
  Downloading https://files.pythonhosted.org/packages/01/fe/e2f43241ff7545588
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: Click in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /us
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/
Installing collected packages: psaw
Successfully installed psaw-0.1.0
```

```
#importing pushshift API
from psaw import PushshiftAPI


api = PushshiftAPI()
```

I am collecting data on 31st May 2021. I am not going to collect very latest or very old data, I am going to collect 6 month old data so that memes gets enough time to be viral and it should also show the latest trend of memes.

```
#dates between which memes are being collected
start_date=int(dt.datetime(2020, 11, 1).timestamp())
end_date=int(dt.datetime(2020, 11, 20).timestamp())


#list of subreddits from whom I am going to collect data
lst_subreddits = ['meme','memes','4PanelCringe','teenagers','dankmemes','PrequelMe
                  'MemePiece','funny','MadeOfStyrofoam']


#Creating an empty dataframe whith required columns to concatenate data from each
dataset = pd.DataFrame(columns=['author','created_utc','is_original_content','is_v
                                'over_18','subreddit','subreddit_subscribers','thu
                                'url','thumbnail_height', 'thumbnail_width','score


#iterating over each subreddits and collecting necessary data
for i in lst_subreddits:
  api_request_generator = api.search_submissions(subreddit = i,
                                                 after = start_date,
                                                 before = end_date,
                                                 filter=['author','created_utc','i
                                                         'subreddit','subreddit_su
                                                         'thumbnail_height', 'thum
                                                 limit=30000)
```

```
                                                                limit=30000)
  aita_submissions = pd.DataFrame([submission.d_ for submission in api_request_gen
  dataset = pd.concat([dataset, aita_submissions], ignore_index=True)
```

```
/usr/local/lib/python3.7/dist-packages/psaw/PushshiftAPI.py:192: UserWarning:
  warnings.warn("Got non 200 code %s" % response.status_code)
/usr/local/lib/python3.7/dist-packages/psaw/PushshiftAPI.py:180: UserWarning:
  warnings.warn("Unable to connect to pushshift.io. Retrying after backoff.")
```

```
dataset.shape
```

```
(166993, 16)
```

I have collected 166993 data with 16 columns

```
#numbers of data from each subreddits
dataset.subreddit.value_counts()
```

```
funny              30000
memes              30000
teenagers          30000
dankmemes          30000
meme               25336
PrequelMemes       10380
HistoryMemes        8405
MadeOfStyrofoam     1674
MemePiece            882
4PanelCringe         316
Name: subreddit, dtype: int64
```

```
#Saving the dataset because these processes takes time, so that I don't have to do
dataset.to_csv('dataset.csv', index=False)
```

```
dataset = pd.read_csv("/content/drive/MyDrive/Applied_ai/dataset.csv")
```

```
#here I will deal with only images, so removing data which are not images
df = dataset[dataset['url'].str.endswith(('jpg','png','jpeg'))]
```

```
df.shape[0]
```

```
123557
```

Now we are left with 123557 image data, now we will remove all those data whose links are not working of giving error.

```
'''
removing all those data whose links are not starting with 'https' because,
I am also going to extract data from images so, if the link to image will not work
I can't extract data from image for that post
'''
```

```
df = df[df['url'].str.startswith(('https'))]
```

**Some of the data collections are very time consuming, so for that I am going to use multiprocessing**

```
#importing multiprocessing library
import multiprocessing
from multiprocessing import  Pool


#Getting status code of each URLs. If status code is 200 then url is working otherw
def status_code(urls):
  stat_code = []
  for i in tqdm(urls, position=0):
    try:
      stat_code.append(requests.get(i).status_code)
    except:
      stat_code.append('error')
  return pd.DataFrame(stat_code)


#Reference : https://towardsdatascience.com/make-your-own-super-pandas-using-multi
def parallelize_dataframe(urls, func, n_cores=4):
    df_split = np.array_split(urls, n_cores)
    pool = Pool(n_cores)
    stat_code = pd.concat(pool.map(func, df_split), ignore_index=True)
    pool.close()
    pool.join()
    return stat_code


stat_code = parallelize_dataframe(df['url'], status_code)
```

```
 53%|█████     | 16519/30887 [40:43<51:36,  4.64it/s]
```

```
df['status_code'] = 0


#making status code as new column of the dataset
df['status_code'] = stat_code[0]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCo
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
```

```
#removing all data whose status code is not 200
df = df[df['status_code']==200]


df.shape
```

```
(84359, 17)
```

Now we are left with only **84359** data

```
#dropping unnecessary columns
df.drop(['created','status_code','thumbnail','author'], axis=1,inplace=True)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:4174: SettingWith
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  errors=errors,
```

Now we are left with **84359** rows and **13** columns.

```
#Saving the dataset because these processes takes time, so that I don't have to do
df.to_csv('df_wrk_lnk.csv')
```

```
df = pd.read_csv('/content/drive/MyDrive/Applied_ai/df_wrk_lnk.csv')
```

## Getting sentiment score for title column

To compute sentiment score I have used pre-trained model by:
https://medium.com/analytics-vidhya/sentiment-analysis-for-text-with-deep-learning-2f0a0c6472b5

This is a LSTM based model trained on **Stanford sentiment treebank data** (dictionary.txt) which has 239233 rows. The index is used to match each of the sentences to a sentiment score in the file “labels.txt”. The score ranges from 0 to 1, 0 being very negative and 1 being very positive.

I am computing sentiment score before extracting text from the image using OCR and combining with title column and cleaning, because I am going to use stemming so the original word will be lost and also the text extracted from the image will have many errors. So, I decided to compute sentiment score on title only and also title is the reflection of content in the meme template.

```
from tensorflow import keras
import json
from nltk.tokenize import RegexpTokenizer
```

```
#loading pre trained model and word index
pred_model = keras.models.load_model('/content/drive/MyDrive/Applied_ai/best_model
word_idx = json.load(open("/content/drive/MyDrive/Applied_ai/word_idx.txt"))


#function to compute sentiment score
def get_sentiment_DL(prd_model, text_data, word_idx):

    live_list = []
    batchSize = len(text_data)
    live_list_np = np.zeros((56,batchSize))

    mask = [isinstance(item, (str, bytes)) for item in text_data['title']]
    text_data = text_data.loc[mask]

    for index, row in text_data.iterrows():


        text_data_sample = text_data['title'][index]

        # split the sentence into its words and remove any punctuations.
        tokenizer = RegexpTokenizer(r'\w+')
        text_data_list = tokenizer.tokenize(text_data_sample)
        if len(text_data_list)>56:
          text_data_list = text_data_list[:56]

        labels = np.array(['1','2','3','4','5','6','7','8','9','10'], dtype = "int

        # get index for the live stage
        data_index = np.array([word_idx[word.lower()] if word.lower() in word_idx e
        data_index_np = np.array(data_index)

        # padded with zeros of length 56 i.e maximum length
        padded_array = np.zeros(56)
        padded_array[:data_index_np.shape[0]] = data_index_np
        data_index_np_pad = padded_array.astype(int)


        live_list.append(data_index_np_pad)

    live_list_np = np.asarray(live_list)
    score = prd_model.predict(live_list_np, batch_size=64, verbose=0)
    single_score = np.round(np.dot(score, labels)/10,decimals=2)
    score_all  = []
    for each_score in score:

        top_3_index = np.argsort(each_score)[-3:]
        top_3_scores = each_score[top_3_index]
        top_3_weights = top_3_scores/np.sum(top_3_scores)
        single_score_dot = np.round(np.dot(top_3_index, top_3_weights)/10, decimals
        score_all.append(single_score_dot)

    text_data['Sentiment_Score'] = pd.DataFrame(score_all)

    return text_data
```

```
df = get_sentiment_DL(pred_model, df, word_idx)
```

## Extracting texts from images using OCR

Memes will have texts written in images, here I am going to extract those texts.

```
#installing pytesseract
!pip install pytesseract
```

```
Collecting pytesseract
  Downloading https://files.pythonhosted.org/packages/a0/e6/a4e9fc8a93c131854
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packag
Building wheels for collected packages: pytesseract
  Building wheel for pytesseract (setup.py) ... done
  Created wheel for pytesseract: filename=pytesseract-0.3.7-py2.py3-none-any.
  Stored in directory: /root/.cache/pip/wheels/81/20/7e/1dd0daad1575d5260916b
Successfully built pytesseract
Installing collected packages: pytesseract
Successfully installed pytesseract-0.3.7
```

```
!sudo apt install tesseract-ocr
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  tesseract-ocr-eng tesseract-ocr-osd
The following NEW packages will be installed:
  tesseract-ocr tesseract-ocr-eng tesseract-ocr-osd
0 upgraded, 3 newly installed, 0 to remove and 39 not upgraded.
Need to get 4,795 kB of archives.
After this operation, 15.8 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/universe amd64 tesseract-ocr-en
Get:2 http://archive.ubuntu.com/ubuntu bionic/universe amd64 tesseract-ocr-os
Get:3 http://archive.ubuntu.com/ubuntu bionic/universe amd64 tesseract-ocr am
Fetched 4,795 kB in 0s (28.9 MB/s)
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based fro
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Selecting previously unselected package tesseract-ocr-eng.
(Reading database ... 160772 files and directories currently installed.)
Preparing to unpack .../tesseract-ocr-eng_4.00~git24-0e00fe6-1.2_all.deb ...
Unpacking tesseract-ocr-eng (4.00~git24-0e00fe6-1.2) ...
Selecting previously unselected package tesseract-ocr-osd.
Preparing to unpack .../tesseract-ocr-osd_4.00~git24-0e00fe6-1.2_all.deb ...
Unpacking tesseract-ocr-osd (4.00~git24-0e00fe6-1.2) ...
Selecting previously unselected package tesseract-ocr.
```

```
Preparing to unpack .../tesseract-ocr_4.00~git2288-10f4998a-2_amd64.deb ...
Unpacking tesseract-ocr (4.00~git2288-10f4998a-2) ...
Setting up tesseract-ocr-osd (4.00~git24-0e00fe6-1.2) ...
Setting up tesseract-ocr-eng (4.00~git24-0e00fe6-1.2) ...
Setting up tesseract-ocr (4.00~git2288-10f4998a-2) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

```
#importing libraries
import cv2
import pytesseract
from skimage import io
from google.colab.patches import cv2_imshow


pytesseract.pytesseract.tesseract_cmd = r'/usr/bin/tesseract'


#creating a column to store extracted text
df['img_txt']=0


"""
This function will extract text from image. It takes url of image as input and
extracts text from image using pytesseract.
"""
def extract_text(url):
  try:
    im = io.imread(url)
    im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    _, im = cv2.threshold(im, 127, 255, cv2.THRESH_BINARY)
    custom_config = r"--oem 3 --psm 11 -c tessedit_char_whitelist= 'ABCDEFGHIJKLMN
    text = pytesseract.image_to_string(im, lang='eng', config=custom_config)
    return text.replace('\n', ' ')
  except:
    return " "


tqdm.pandas(position=0)
df['img_txt'] = df['url'].progress_apply(extract_text)


#Saving the dataset because these processes takes time, so that I don't have to do
df.to_csv("df_img_txt.csv")


df = pd.read_csv('/content/drive/MyDrive/Applied_ai/df_img_txt.csv')
```

```
#combining all text columns to one column and dropping the columns which are being
df['text'] = df['title']+df['img_txt']
df.drop(['title','img_txt'], axis=1, inplace=True)
```

## Cleaning text column

```
#importing libraries
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
import re


#downloading punkt and stopwords
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
True
```

```
#this function will take text as input and will clean it and return cleaned text
def clean_text(text):
    text = re.sub('[^A-Za-z]',' ',text).lower()    #Removing all characters from t
    words = word_tokenize(text)
    stopWords = set(stopwords.words('english'))
    ps = PorterStemmer()
    cleanedText = []
    for word in words:
        word = ps.stem(word)
        if word in stopWords or len(word)<=4:      #Removing stopwords and words w
            continue
        else:
            cleanedText.append(word)
    return " ".join(set(cleanedText))


tqdm.pandas(position=0)
#calling function to clean the text
df['text'] = df['text'].progress_apply(clean_text)
```

```
/usr/local/lib/python3.7/dist-packages/tqdm/std.py:658: FutureWarning: The Pa
  from pandas import Panel
100%|██████████| 84359/84359 [01:23<00:00, 1015.38it/s]
```

```
#new feature containing number of words in text
```

```
df['num_words'] = df['text'].str.split().apply(len)


df = pd.read_csv('/content/drive/MyDrive/Applied_ai/df_img_feature_pred.csv')
```

## Extracting HSV value from the image

Extracting Hue, Saturation and Value of the image.

```
#Reference : https://www.codespeedy.com/splitting-rgb-and-hsv-values-in-an-image-us
#This function will take url as input and return average Hue, Saturation and Value
def get_avg_hsv(url):
  try:
    im = io.imread(url)
    hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
    h,s,v = cv2.split(hsv)
    return [h.mean(), s.mean(), v.mean()]
  except:
    return [0,0,0]


#this function will apply the above function to url column of the dataset
def hsv(df):
  tqdm.pandas(position=0)
  avg_hsv = df.progress_apply(get_avg_hsv)
  return avg_hsv


#This function will help to parallelize our process
def parallelize_dataframe(urls, func, n_cores=4):
    df_split = np.array_split(urls, n_cores)
    pool = Pool(n_cores)
    avg_hsv = pd.concat(pool.map(func, df_split))
    pool.close()
    pool.join()
    return avg_hsv


avg_hsv = parallelize_dataframe(df['url'], hsv)
```

```
    /usr/local/lib/python3.7/dist-packages/tqdm/std.py:658: FutureWarning: The Pa
      from pandas import Panel
      0%|          | 0/21090 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-packag
      from pandas import Panel
      0%|          | 0/21090 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-packag
      from pandas import Panel
    /usr/local/lib/python3.7/dist-packages/tqdm/std.py:658: FutureWarning: The Pa
      from pandas import Panel
      4%|▍         | 888/21090 [02:27<1:24:55,  3.96it/s]/usr/local/lib/python3.7
      DecompressionBombWarning,
    100%|██████████| 21090/21090 [1:00:20<00:00,  5.82it/s]
    100%|██████████| 21090/21090 [1:02:13<00:00,  5.65it/s]
```

```
100%|██████████| 21090/21090 [1:03:45<00:00,  5.51it/s]
100%|██████████| 21089/21089 [1:13:28<00:00,  4.78it/s]
```

```
#creating columns to store average Hue, Saturation and Value of the image
df['avg_h'] = np.matrix(avg_hsv.tolist())[:,0]
df['avg_s'] = np.matrix(avg_hsv.tolist())[:,1]
df['avg_v'] = np.matrix(avg_hsv.tolist())[:,2]


#Saving the dataset because these processes takes time, so that I don't have to do
df.to_csv("df_hsv.csv")


df = pd.read_csv('/content/drive/MyDrive/Applied_ai/df_hsv.csv')
```

## Extracting colors from images

Extracting normalized number of pixels with certain color in the image.

```
#Reference : https://www.pyimagesearch.com/2014/08/04/opencv-python-color-detectio
#Reference : https://realpython.com/python-opencv-color-spaces/
#Reference : https://alloyui.com/examples/color-picker/hsv.html
'''
gimp uses HSV value in range of H = 0-360, S = 0-100 and V = 0-100.
But OpenCV uses H: 0-179, S: 0-255, V: 0-255.
So, when we will try to get a HSV value it will be in gimp's form. We have to conv
that so that it can be used by Opencv.
H(opencv) = H(gimp)/2
S(opencv) = (S(gimp)*255)/100
V(opencv) = (V(gimp)*255)/100
'''
boundries = [([0,0,200],[180,25,255]),      #white
             ([0,0,0],[180,255,3]),         #black
             ([0,0,100],[180,20,180]),        #gray
             ([0,90,115],[17,255,190]),     #brown
             ([20,50,240],[30,75,255]),     #off-white
             ([0,140,155],[12,255,230]),      #dark red
             ([0,140,230],[12,255,255]),     #light red
             ([13,190,155],[17,255,230]),     #dark orange
             ([13,140,230],[115,255,255]),    #light orange
             ([18,140,155],[140,255,230]),     #goldish
             ([23,140,230],[165,255,255]),    #yellow
             ([28,90,155],[80,255,230]),    #dark green
             ([33,90,230],[77,255,255]),   #light green
             ([85,77,153],[93,255,230]),   #dark cyan
             ([85,77,230],[93,255,255]),  #cyan
             ([100,128,90],[125,255,190]),   #dark blue
             ([100,128,193],[125,255,255]),  #light blue
             ([142,102,153],[150,255,255]),  #purple
```

```
                ([158,102,180],[160,255,255]),  #pink
                ([0,0,255],[180,25,255])      #faded colors
    ]


    #function to extract color from image
    def extract_color(url):
      try:
        im = io.imread(url)
        num_pixel = im.shape[0] * im.shape[1]
        hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
        clr_pixel = []

        for(lower, upper) in boundries:
          lower = np.array(lower, dtype = "uint8")
          upper = np.array(upper, dtype = "uint8")
          mask = cv2.inRange(hsv, lower, upper)
          clr_pixel.append(round(((mask==255).sum())/num_pixel,5)) #counting and norma

        return clr_pixel

      except:
        return [0]*20


    #this function will apply the above function to url column of the dataset
    def color(df):
      tqdm.pandas(position=0)
      norm_pix = df.progress_apply(extract_color)
      return norm_pix


    #This function will help to parallelize our process
    def parallelize_data_color(urls, func, n_cores=4):
        df_split = np.array_split(urls, n_cores)
        pool = Pool(n_cores)
        norm_pix = pd.concat(pool.map(func, df_split))
        pool.close()
        pool.join()
        return norm_pix


    norm_pix_img = parallelize_data_color(df['url'], color)
```

```
     /usr/local/lib/python3.7/dist-packages/tqdm/std.py:658: FutureWarning: The Pa
       from pandas import Panel
       0%|          | 0/21090 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-packag
       from pandas import Panel
     /usr/local/lib/python3.7/dist-packages/tqdm/std.py:658: FutureWarning: The Pa
       from pandas import Panel
       0%|          | 0/21090 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-packag
       from pandas import Panel
       4%|▌         | 863/21090 [04:44<1:21:55,  4.12it/s]/usr/local/lib/python3.7
       DecompressionBombWarning,
      39%|███▉      | 8265/21090 [48:23<1:03:23,  3.37it/s]
```

```
colors = ['white','black','gray','brown','off-white','dark red','light red','dark
          'goldish','yellow','dark green','light green','dark cyan','cyan','dark b
          'purple','pink','faded colors'
         ]


#adding new color features to the dataset
for i, color in enumerate(colors):
  df[color] = np.matrix(norm_pix_img.tolist())[:,i]


#Saving the dataset because these processes takes time, so that I don't have to do
df.to_csv('df_color.csv')
```

## Extracting features from images

Each meme is in the form of image so, there will be image of objects in the image. Here I am going to extract those objects with their respective probabilities.


Image above is a meme image, here we can see that there are human faces so we have to extract those objects. We can also see that there are texts in the images which can be extracted

using OCR.

```
import keras
from keras.applications.vgg16 import VGG16
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
~/.local/lib/python3.8/site-packages/keras/__init__.py in <module>
      2 try:
----> 3     from tensorflow.keras.layers.experimental.preprocessing import
RandomRotation
      4 except ImportError:

ModuleNotFoundError: No module named 'tensorflow'

During handling of the above exception, another exception occurred:

ImportError                               Traceback (most recent call last)
<ipython-input-4-eceeb2eee374> in <module>
----> 1 import keras
      2 from keras.applications.vgg16 import VGG16

~/.local/lib/python3.8/site-packages/keras/__init__.py in <module>
      3     from tensorflow.keras.layers.experimental.preprocessing import
RandomRotation
      4 except ImportError:
----> 5     raise ImportError(
      6         'Keras requires TensorFlow 2.2 or higher. '
      7         'Install TensorFlow via `pip install tensorflow`')

ImportError: Keras requires TensorFlow 2.2 or higher. Install TensorFlow via
`pip install tensorflow`
```

```
!pip install tensorflow
```

```
Collecting tensorflow
  Downloading tensorflow-2.5.0-cp38-cp38-manylinux2010_x86_64.whl (454.4 MB)
     |                                | 2.5 MB 71 kB/s eta 1:46:02^C

ERROR: Operation cancelled by user
```

```
df = pd.read_csv('/content/drive/MyDrive/Applied_ai/df_color.csv')
```

```
#importing libraries
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
import tensorflow as tf
# load the model
model = VGG16()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applica
553467904/553467096 [==============================] - 8s 0us/step
553476096/553467096 [==============================] - 8s 0us/step
```

```
#this function will take url of image as input and will get the image and convert i
def load_image(link, target_size=None):
    import requests
    import shutil
    import os

    _, ext = os.path.splitext(link)

    r = requests.get(link, stream=True)
    with open('temp.' + ext, 'wb') as f:
        r.raw.decode_content = True
        shutil.copyfileobj(r.raw, f)

    img = image.load_img('temp.' + ext, target_size=target_size)
    return image.img_to_array(img)


#this function will extract features and their probabilities from image and return
def extract_feature_from_image(url):
  try:
    im = load_image(url, target_size=(224, 224))
    im = im.reshape((1, im.shape[0], im.shape[1], im.shape[2]))
    im = preprocess_input(im)
    yhat = model.predict(im)
    label = decode_predictions(yhat, top=3)
    obj = []
    pred = []
    for i in label[0]:
      obj.append(i[1])
      pred.append(i[2])
    return [obj,pred]

  except:
    return [[""]*3, [0]*3]


tqdm.pandas(position=0)
img_feature_pred = df['url'].progress_apply(extract_feature_from_image)
```

```
/usr/local/lib/python3.7/dist-packages/tqdm/std.py:658: FutureWarning: The Pa
  from pandas import Panel
  0%|          | 0/20000 [00:00<?, ?it/s]Downloading data from https://storag
40960/35363 [==================================] - 0s 0us/step
49152/35363 [==========================================] - 0s 0us/step
100%|██████████| 20000/20000 [1:11:26<00:00,  4.67it/s]
```

```
#creating new columns in the dataset to store new features
df['img_feature'] = np.empty(84359, dtype=object)
df['img_feature_pred'] = np.empty(84359, dtype=object)


#adding newly generated features to the dataset
df['img_feature'] = [i[0] for i in img_feature_pred]
```

```
df['img_feature_pred'] = [i[1] for i in img_feature_pred]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCo
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCo
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
```

```
#Saving the dataset because these processes takes time, so that I don't have to do
df.to_csv('df_img_feature_pred.csv',index=False)
```

## Dropping Duplicates

In this dataset we have many such rows in which urls are repeated means duplicate urls. Here I am going to drop such data.

```
df = pd.read_csv("/content/drive/MyDrive/Applied_ai/df_senti.csv")
```

```
#dropping rows based on duplicate urls, because urls are unique for each posts
df.drop_duplicates(subset=['url'], inplace=True)
```

```
df.shape[0]
```

```
83908
```

After dropping duplicate values we are left with **83908** data points.

```
df.to_csv('df_dup.csv')
```

## Dank or Not

Marking data points whether they are dank or not based on normalized score.

```
df = pd.read_csv("/content/drive/MyDrive/Applied_ai/df_dup.csv")
```

```
'''
```

```
'''
Subreddits with more subscribers tend to get more upvotes, so I am normalizing the
the number of subscribers from the respective subreddit where it was posted.
'''
df['score'] = df['score']/df['subreddit_subscribers']


'''
Checking percentile value of normalized score, so that I can get clear understandi
as dank or not and making the problem as binary classification problem.
'''
#checking for top 5 precentile value
top_5_percentile_value = []
for i in range(95,101):
  top_5_percentile_value.append(np.percentile(df['score'], i))
print(top_5_percentile_value)
```

```
[3.325298850927006e-05, 5.056514246890454e-05, 9.213297312491056e-05, 0.00027
```

Here we can see that there is huge difference between 97th percentile and 98th percentile values. Now lets see between 97th and 98th percentile value.

```
between_97_98_percentile_value = []
for i in np.arange(0,1,0.1):
  between_97_98_percentile_value.append(np.percentile(df['score'], (97+i)))
print(between_97_98_percentile_value)
```

```
[9.213297312491056e-05, 9.876765417343272e-05, 0.00010782892342374918, 0.0001
```

Here we can see that, there is huge difference between 97.1 percentile value and 97.2 percentile value.
So, I will mark data with normalized score value greater than 97.1 percentile as 1 and others as 0.

```
df['dank_or_not'] = (df['score']>np.percentile(norm_score, 97.1)).astype('int')


df['dank_or_not'].value_counts()
```

```
0    81474
1     2434
Name: dank_or_not, dtype: int64
```

Here we have only **2434** dank data points i.e. only **2.9%** of the dataset.

```
df.to_csv('df_dankornot.csv', index=False)
```

## Data cleaning and feature encoding

cleaning the dataset and encoding the categorical features of the dataset.

```
df = pd.read_csv("/content/drive/MyDrive/Applied_ai/df_dankornot.csv")
```

## Encoding created_utc column and creating new 'hour' column

The created_utc feature contains the timestamp when the post appeared on Reddit in the Coordinated Universal Time zone (UTC). Since most active Reddit users reside inthe USA, we converted this to North American Central Time Zone.

```
from datetime import datetime
from pytz import timezone
import pytz


fmt = "%Y-%m-%d %H:%M:%S %Z%z"


datetime_converted = df['created_utc'].apply(datetime.fromtimestamp)


datetime_converted=pd.DatetimeIndex(datetime_converted).tz_localize('UTC').tz_conv


df['created_utc'] = pd.to_datetime(datetime_converted)


#creating hour column
df['hour'] = df['created_utc'].apply(lambda x: x.hour)


df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83908 entries, 0 to 83907
Data columns (total 42 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   created_utc            83908 non-null  datetime64[ns, America/New_York]
 1   is_original_content    83908 non-null  bool
 2   is_video               83908 non-null  bool
 3   num_comments           83908 non-null  int64
 4   over_18                83908 non-null  bool
 5   subreddit              83908 non-null  object
 6   subreddit_subscribers  83908 non-null  int64
 7   upvote_ratio           83908 non-null  float64
 8   url                    83908 non-null  object
 9   thumbnail_height       83842 non-null  float64
```

```
 10  thumbnail_width         83842 non-null  float64
 11  text                    81766 non-null  object
 12  num_words               83908 non-null  int64
 13  avg_h                   83908 non-null  float64
 14  avg_s                   83908 non-null  float64
 15  avg_v                   83908 non-null  float64
 16  img_feature             83908 non-null  object
 17  img_feature_pred        83908 non-null  object
 18  sentiment_score         83908 non-null  float64
 19  score                   83908 non-null  float64
 20  dank_or_not             83908 non-null  int64
 21  white                   83908 non-null  float64
 22  black                   83908 non-null  float64
 23  gray                    83908 non-null  float64
 24  brown                   83908 non-null  float64
 25  off-white               83908 non-null  float64
 26  dark red                83908 non-null  float64
 27  light red               83908 non-null  float64
 28  dark orange             83908 non-null  float64
 29  light orange            83908 non-null  float64
 30  goldish                 83908 non-null  float64
 31  yellow                  83908 non-null  float64
 32  dark green              83908 non-null  float64
 33  light green             83908 non-null  float64
 34  dark cyan               83908 non-null  float64
 35  cyan                    83908 non-null  float64
 36  dark blue               83908 non-null  float64
 37  light blue              83908 non-null  float64
 38  purple                  83908 non-null  float64
 39  pink                    83908 non-null  float64
 40  faded colors            83908 non-null  float64
 41  hour                    83908 non-null  int64
dtypes: bool(3), datetime64[ns, America/New_York](1), float64(28), int64(5),
memory usage: 25.2+ MB
```

Since we have only images so lets see 'is_video' column.

```
df['is_video'].value_counts()
```

```
False    83908
Name: is_video, dtype: int64
```

'is_video' column have only one value i.e. False so, dropping this column.

```
df.drop(['is_video'], axis=1, inplace=True)
```

thumbnail_height, thumbnail_width and text columns have some null values.

For text column there can be possibility that memes have only image no titles and no texts on images so in this case there will be no text for some data points. We can fill that NaN value with empty string.

```
df['text'] = df['text'].fillna("")
```

Now checking thumbnail_height and thumbnail_width columns.

```
df[df['thumbnail_height'].isna()]
```

| | created_utc | is_original_content | num_comments | over_18 | subreddit |
|---|---|---|---|---|---|
| **5393** | 2020-11-14 23:27:53-05:00 | False | 1 | False | meme |
| **5636** | 2020-11-14 16:23:22-05:00 | False | 0 | False | meme |
| **5637** | 2020-11-14 16:23:21-05:00 | False | 0 | False | meme |
| **5900** | 2020-11-14 10:50:22-05:00 | False | 48 | False | meme |
| **6171** | 2020-11-14 03:22:19-05:00 | False | 1 | False | meme |
| **...** | ... | ... | ... | ... | ... |
| **81524** | 2020-11-04 07:36:48-05:00 | False | 0 | False | funny |
| **81701** | 2020-11-03 22:57:57-05:00 | False | 1 | False | funny |
| **82041** | 2020-11-03 14:27:17-05:00 | False | 1 | False | funny |
| **82597** | 2020-11-02 18:17:40-05:00 | False | 0 | False | funny |
| **83657** | 2020-11-06 16:50:10-05:00 | False | 5 | False | MadeOfStyrofoam |

66 rows × 41 columns

Here we can see that the data points which have thumbnail_height as NaN value also have thumbnail_width value as NaN and all other data are present for these data points. So, we can't drop these data points.

```
print(df['thumbnail_height'].value_counts())
print()
print(df['thumbnail_width'].value_counts())
```

```
140.0    46553
105.0     2214
78.0      2208
138.0     1126
139.0     1064
          ...
9.0          1
6.0          1
11.0         1
7.0          1
1.0          1
Name: thumbnail_height, Length: 136, dtype: int64

140.0    83776
70.0        54
56.0         3
42.0         2
68.0         1
62.0         1
1.0          1
9.0          1
66.0         1
4.0          1
8.0          1
Name: thumbnail_width, dtype: int64
```

Since we have only 66 values missing in these two columns we can fill them with most frequent values because more than half data have 140 thumbnail_height and almost 99% of the data have 140 thumbnail_width.

```
df['thumbnail_height'].fillna(df['thumbnail_height'].mode().iloc[0], inplace=True)
df['thumbnail_width'].fillna(df['thumbnail_width'].mode().iloc[0], inplace=True)
```

```
df.describe()
```

| | num_comments | subreddit_subscribers | upvote_ratio | thumbnail_height | th |
|---|---|---|---|---|---|
| **count** | 83908.000000 | 8.390800e+04 | 83908.000000 | 83908.000000 | |
| **mean** | 15.108154 | 8.977504e+06 | 0.956594 | 124.999404 | |
| **std** | 78.849474 | 1.078705e+07 | 0.081024 | 22.762394 | |

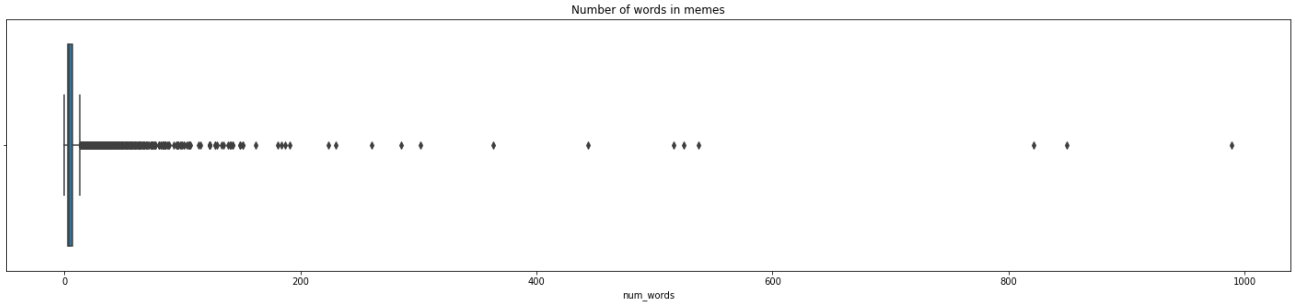
Here we can see that num_words feature have 989 maximum number of words which can be outlier because for a meme such large number of words are not obvious and also the mean value is around 6.

Lets check num_words column.

| | | | | |
|---|---|---|---|---|
| **max** | 5252.000000 | 3.424276e+07 | 1.000000 | 140.000000 |

```
plt.figure(figsize=(25,5))
sns.boxplot(df['num_words'])
plt.title('Number of words in memes')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarni
  FutureWarning
```




Here we can see that most of the memes have words less than 100.
Lets see with the help of percentiles.

```
np.percentile(df['num_words'],99.9)
```

```
75.0
```

```
for i in range(90,101):
  print(i,'percentile : ',np.percentile(df['num_words'],i))
```

```
90 percentile :  10.0
91 percentile :  11.0
92 percentile :  12.0
93 percentile :  12.0
94 percentile :  13.0
95 percentile :  14.0
96 percentile :  15.0
```

```
97 percentile :  17.0
98 percentile :  21.0
99 percentile :  28.0
100 percentile :  989.0
```

There is huge jump between 99th and 100th percentile. Lets visualize between them.

```
for i in np.arange(99,100.1,0.1):
  print(i,'percentile : ',np.percentile(df['num_words'],i))
```

```
99.0 percentile :  28.0
99.1 percentile :  29.0
99.19999999999999 percentile :  31.0
99.29999999999998 percentile :  33.0
99.39999999999998 percentile :  35.0
99.49999999999997 percentile :  37.0
99.59999999999997 percentile :  41.0
99.69999999999996 percentile :  47.0
99.79999999999995 percentile :  56.0
99.89999999999995 percentile :  75.0
99.99999999999994 percentile :  988.9999999938882
```

There is huge jump between 99.9th and 100th percentile. So, we can remove the data points which have number of words greater than 75.

```
df = df[df['num_words']<=75]
```

```
df.shape[0]
```

```
83826
```

Now we are left with **83826** data points.

## Dropping unimportant columns

```
df.drop(['created_utc','subreddit','url'], axis=1, inplace=True)
```

## Encoding 'is_original_content' column

```
df['is_original_content'] = df['is_original_content'].astype('int')
```

## Encoding 'over_18' column

```
df['over_18'] = df['over_18'].astype('int')
```

## Encoding image feature column

Idea to encode this feature is that, I will take top 10 objects from top 5% (dank) data points, lower 5% from least popular memes and 10 features which are shared among most and least popular memes.

```
top_5 = df[df['score']>np.percentile(df['score'],95)]['img_feature']
least_5 = df[df['score']<np.percentile(df['score'],5)]['img_feature']
```

```
#list data are in form of string, converting them to list
from ast import literal_eval
top_5 = top_5.apply(literal_eval)
least_5 = least_5.apply(literal_eval)
```

```
#storing all the words present im top 5% features
top_list=[]
for value in top_5:
    top_list.extend(value)
```

```
#storing all the words present im bottom 5% features
least_list=[]
for value in least_5:
    least_list.extend(value)
```

Getting top 10 **shared** objects

```
top_obj = pd.DataFrame(top_list).value_counts()[:11].index.to_frame(index=False)[0
```

```
least_obj = pd.DataFrame(least_list).value_counts()[:10].index.to_frame(index=Fals
```

```
shared = set(top_obj).intersection(set(least_obj))
```

Getting top 10 **most popular** objects which are not shared

```
top_obj = pd.DataFrame(top_list).value_counts()[:47].index.to_frame(index=False)[0
```

```
least_obj = pd.DataFrame(least_list).value_counts()[:47].index.to_frame(index=Fals
```

```
most_pop = set(top_obj) - set(least_obj)
```

Getting top 10 **least popular** objects which are not shared

```
least_obj = pd.DataFrame(least_list).value_counts()[-10:].index.to_frame(index=Fal

top_obj = pd.DataFrame(top_list).value_counts()[:47].index.to_frame(index=False)[0

least_pop = set(least_obj) - set(top_obj)

#showing top 10 words in top 5%, bottom 5% and shared data
from prettytable import PrettyTable
x = PrettyTable()

x.add_column("Shared",list(shared))
x.add_column("Top 5%",list(most_pop))
x.add_column("Bottom 5%",list(least_pop))

print(x)
```

```
+---------------+------------------+-------------+
|     Shared    |      Top 5%      |  Bottom 5%  |
+---------------+------------------+-------------+
|     screen    | bulletproof_vest |   milk_can  |
|   television  |   oxygen_mask    |  mud_turtle |
| jigsaw_puzzle |    Chihuahua     |  bee_eater  |
|    web_site   |    can_opener    | mixing_bowl |
|  book_jacket  |    ballplayer    |     mink    |
|     packet    |     marimba      |    otter    |
|    envelope   |       pick       |    beaver   |
|      menu     |     bookshop     |    organ    |
|    monitor    |   balance_beam   |   meerkat   |
|   comic_book  |  academic_gown   |   zucchini  |
+---------------+------------------+-------------+
```

```
#list in img_feature column are in form of string, converting them to list
df['img_feature'] = df['img_feature'].apply(literal_eval)

#now combining all elements of list as one string
df['img_feature'] = df['img_feature'].apply(lambda x: ' '.join(map(str, x)))

#storing all the 30 words so that we can use them during bag of words creation
vocab = []
vocab.extend(list(most_pop))
vocab.extend(list(least_pop))
vocab.extend(list(shared))

#using countvectorizer to create bag of words
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(vocabulary=vocab)
```

```
img_feature_vec = cv.fit_transform(df['img_feature'])

img_feature_vec = pd.DataFrame(img_feature_vec.toarray(),columns=cv.get_feature_na

#joining all bag of word features to original dataset
df = df.reset_index().join(img_feature_vec)

df.drop(['index','img_feature'], axis=1, inplace=True)

#saving the dataset
df.to_csv('df_clean.csv', index=False)
```

# Exploratory Data Analysis

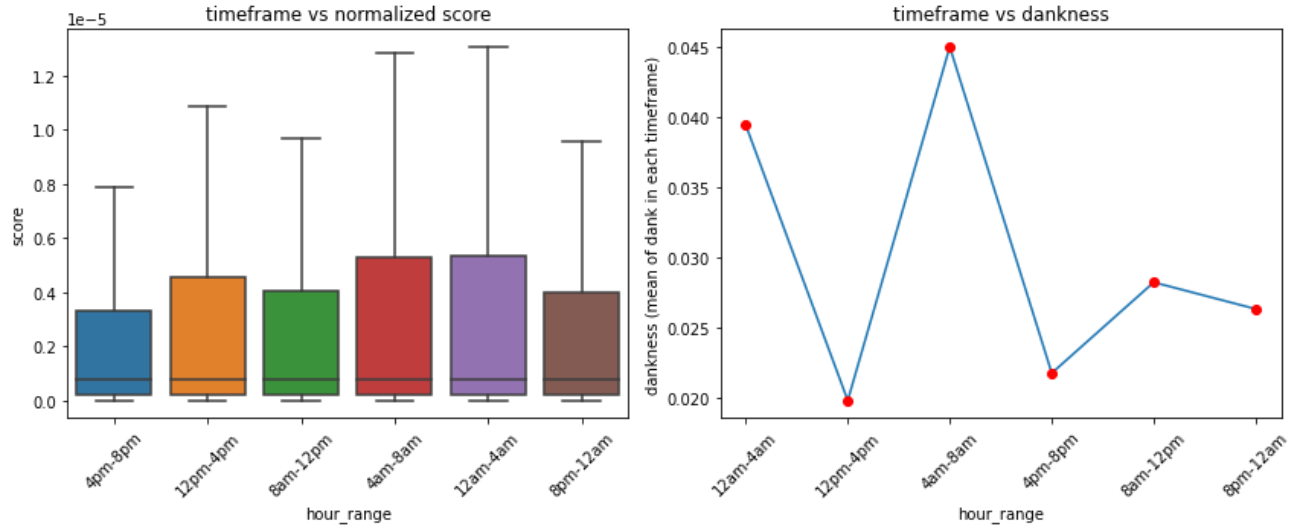
```
df = pd.read_csv("/content/drive/MyDrive/Applied_ai/df_clean.csv")
```

## Time vs normalized score

Here I am trying to visualize when in a day meme should be posted so that it's chance of becoming dank increases.
To do that I will divide the time in different ranges and visualise each range.

```
#Dividing hours in six groups, each group has four hour timeframe.
hour_range = []
for i in df['hour']:
  if i>=0 and i<=4:
    hour_range.append('12am-4am')
  elif i>4 and i<=8:
    hour_range.append('4am-8am')
  elif i>8 and i<=12:
    hour_range.append('8am-12pm')
  elif i>12 and i<=16:
    hour_range.append('12pm-4pm')
  elif i>16 and i<=20:
    hour_range.append('4pm-8pm')
  elif i>20 and i<=23:
    hour_range.append('8pm-12am')

df['hour_range'] = hour_range
```

```
'''
groupby the dataset on hour_range column and getting mean of dank_or_not column in
so that we can find relation between timeframe and dankness of the memes.
'''
grpby_hr_don = df.groupby('hour_range')['dank_or_not'].mean()


#plotting graphs
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
sns.boxplot(x='hour_range', y='score', data=df,showfliers=False)
plt.title('timeframe vs normalized score')
plt.xticks(rotation=45)

plt.subplot(1,2,2)
plt.plot(grpby_hr_don.index,grpby_hr_don.values)
plt.plot(grpby_hr_don.index,grpby_hr_don.values, 'ro')
plt.title('timeframe vs dankness')
plt.xlabel('hour_range')
plt.ylabel('dankness (mean of dank in each timeframe)')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```



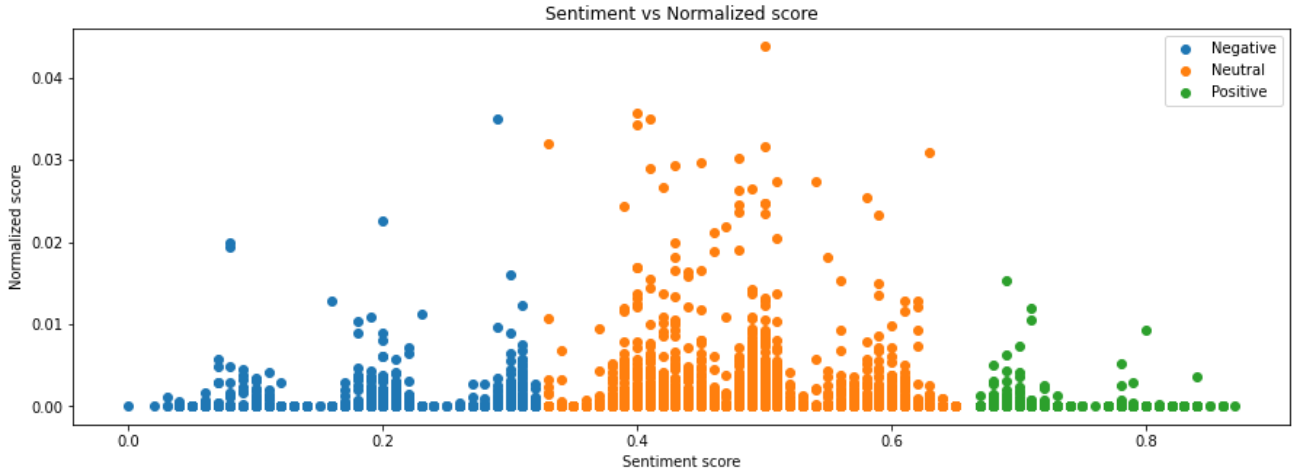

From the plot above we can see that the memes which are posted between 12am to 4am have highest score values followed by 4am to 8am. Reason for this can be that most of the users scrolls social networking sites just before sleeping for the case of 12am to 4am and just after waking up for case of 4am to 8 am. One more reason can be, people need humor just before sleeping and just after waking up.

Memes have least score when posted between 4pm to 8pm.

## Sentiment score vs Normalized score

Here I am trying to find, what should be the sentiment of text in the meme to be dank.

```
#plotting negative, neutral and positive sentiments on same graph
plt.figure(figsize=(15,5))
#positive
plt.scatter(df[df['sentiment_score']<0.33]['sentiment_score'], df[df['sentiment_sc
#neutral
plt.scatter(df[(df['sentiment_score']>=0.33) & (df['sentiment_score']<0.66)]['sent
#negative
plt.scatter(df[df['sentiment_score']>0.66]['sentiment_score'], df[df['sentiment_sc

plt.title('Sentiment vs Normalized score')
plt.xlabel('Sentiment score')
plt.ylabel('Normalized score')
plt.legend(['Negative','Neutral','Positive'])
plt.show()
```



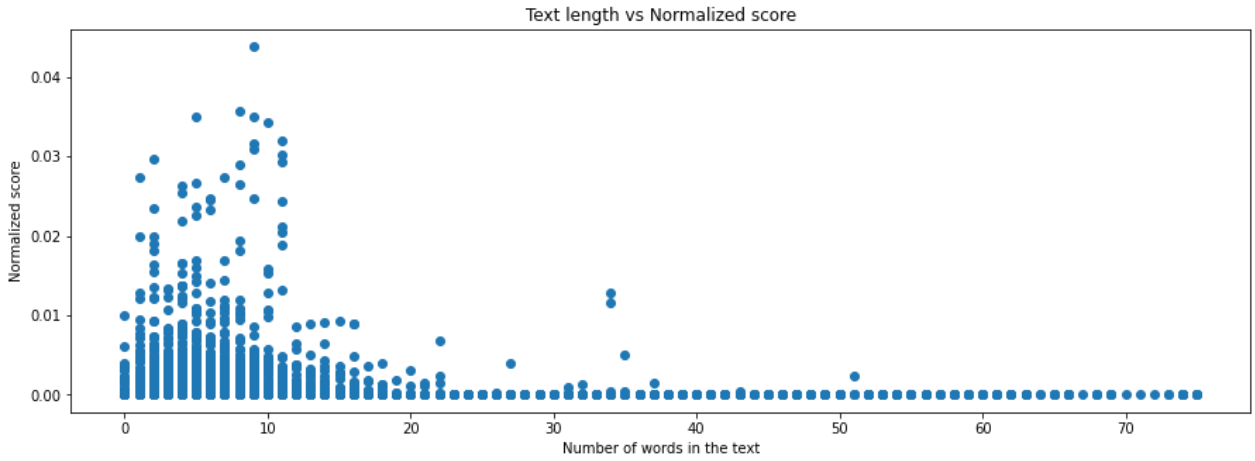

Here we can see that,

- neutral texts have highest chances of becoming dank.
- Between positive and negative sentiments, text with negative sentiment in the meme have higher chances of becoming dank.

## Text length vs Normalized score

```
plt.figure(figsize=(15,5))
plt.scatter(df['num_words'], df['score'])

plt.title('Text length vs Normalized score')
```

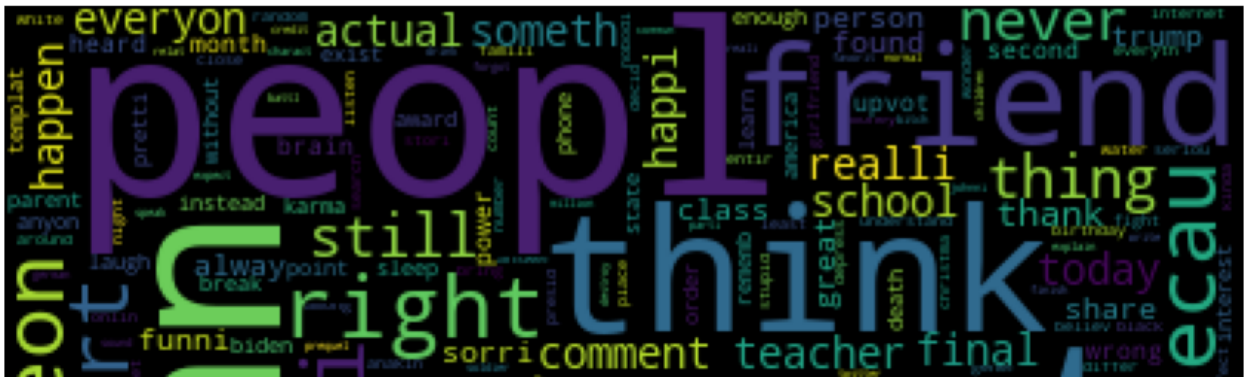
```
plt.xlabel('Number of words in the text')
plt.ylabel('Normalized score')
plt.show()
```




- Text with word length less than 12 is most preferable and have high chances of becoming dank and it's obvious, peoplr don't like to read long texts for a pinch of humor.
- Texts with word length greater than 11 is least preferable.

## Word cloud for text column

```
from wordcloud import WordCloud
plt.figure(figsize=(15,10))
text = " ".join(i for i in df['text'].astype(str))
wordcloud = WordCloud().generate(text)
# Generate plot
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```




Here we can see that some of the words are not so important for meme such as 'memat' which is mematic, a website where memers create memes and all the memes created from mematic have it's watermark and OCR reads those watermarks. 'reddit', from where we have extracted data. 'nan', rows where no words are present.
So, let's remove these words and generate word cloud again.

```
text = text.replace('memat','').replace('reddit','').replace('nan','')
```

```
plt.figure(figsize=(12,10))
wordcloud = WordCloud().generate(text)
# Generate plot
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```

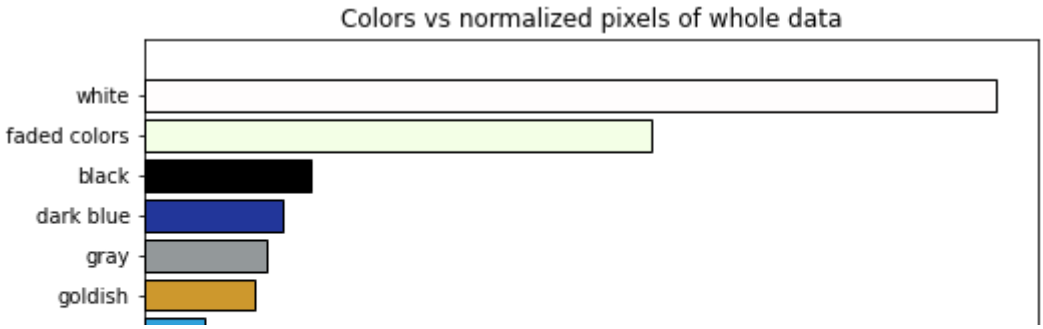

Some of the most frequent words in the memes are 'people','think','because','someone','something','still','really','friend','teacher' etc.

## Color vs Normalized pixels

We have extracted normalized pixel (numper of pixel of a color divider by total umber of pixels of the image) for colors in the images.
We will sum all the normalized pixels of a color and those color which have maximum value are most frequent color in the meme.

```
#adding all the normalized pixels of a color and dividing it by number of data ava
white     = df['white'].sum()/df.shape[0]
black     = df['black'].sum()/df.shape[0]
gray      = df['gray'].sum()/df.shape[0]
brown     = df['brown'].sum()/df.shape[0]
off_white = df['off-white'].sum()/df.shape[0]
dark_red  = df['dark red'].sum()/df.shape[0]
light_red = df['light red'].sum()/df.shape[0]
dark_orange   = df['dark orange'].sum()/df.shape[0]
light_orange  = df['light orange'].sum()/df.shape[0]
goldish       = df['goldish'].sum()/df.shape[0]
yellow        = df['yellow'].sum()/df.shape[0]
dark_green    = df['dark green'].sum()/df.shape[0]
light_green   = df['light green'].sum()/df.shape[0]
dark_cyan     = df['dark cyan'].sum()/df.shape[0]
cyan          = df['cyan'].sum()/df.shape[0]
dark_blue     = df['dark blue'].sum()/df.shape[0]
light_blue    = df['light blue'].sum()/df.shape[0]
purple        = df['purple'].sum()/df.shape[0]
pink          = df['pink'].sum()/df.shape[0]
faded_colors  = df['faded colors'].sum()/df.shape[0]


#creating a dictionary to store colors and their corresponding pixel value
x = {'white':white,'black':black,'gray':gray,'brown':brown,'off white':off_white,
     'dark red':dark_red,'light red':light_red,'dark orange':dark_orange,'light ora
     'goldish':goldish,'yellow':yellow,'dark_green':dark_green,'light green':light_
     'dark cyan':dark_cyan,'cyan':cyan,'dark blue':dark_blue,'light blue':light_blu
     'pink':pink,'faded colors':faded_colors}


#sorting the dictionary
x = dict(sorted(x.items(), key=lambda item: item[1]))


#plotting Color vs Normalized pixels
plt.figure(figsize=(8,8))
color = plt.barh(list(x.keys()),list(x.values()), height=0.8)

#setting different colors for different bars
color[-1].set_color('#FFFDFD')
color[-1].set_edgecolor('black')
color[-2].set_color('#F3FFE6')
color[-2].set_edgecolor('black')
color[-3].set_color('#000000')
color[-3].set_edgecolor('black')
color[-4].set_color('#22369A')
color[-4].set_edgecolor('black')
color[-5].set_color('#93989A')
```

```
    color[-5].set_edgecolor('black')
    color[-6].set_color('#CD982D')
    color[-6].set_edgecolor('black')
    color[-7].set_color('#30A1DA')
    color[-7].set_edgecolor('black')
    color[-8].set_color('#F3EA66')
    color[-8].set_edgecolor('black')
    color[-9].set_color('#94533B')
    color[-9].set_edgecolor('black')
    color[-10].set_color('#E6953E')
    color[-10].set_edgecolor('black')
    color[-11].set_color('#599F36')
    color[-11].set_edgecolor('black')
    color[-12].set_color('#44C8BF')
    color[-12].set_edgecolor('black')
    color[-13].set_color('#C8462C')
    color[-13].set_edgecolor('black')
    color[-14].set_color('#35F1F1')
    color[-14].set_edgecolor('black')
    color[-15].set_color('#D7832F')
    color[-15].set_edgecolor('black')
    color[-16].set_color('#EC5234')
    color[-16].set_edgecolor('black')
    color[-17].set_color('#71EC34')
    color[-17].set_edgecolor('black')
    color[-18].set_color('#C8C098')
    color[-18].set_edgecolor('black')
    color[-19].set_color('#AE2CC8')
    color[-19].set_edgecolor('black')
    color[-20].set_color('#E633AA')
    color[-20].set_edgecolor('black')

    plt.title('Colors vs normalized pixels of whole data')
    plt.xlabel('Normalized pixels')
    plt.ylabel('Colors')
    plt.show()
```




In most of the memes white, faded and dark colors are used. Light and bright colors are rarely used in the memes.In general, muted colors are more abundant than bright colors in memes. Perhaps because memes tend to be mundane photos, often blurry in self-made way, unlike professional photography.

## Let's visualize this for only dank memes

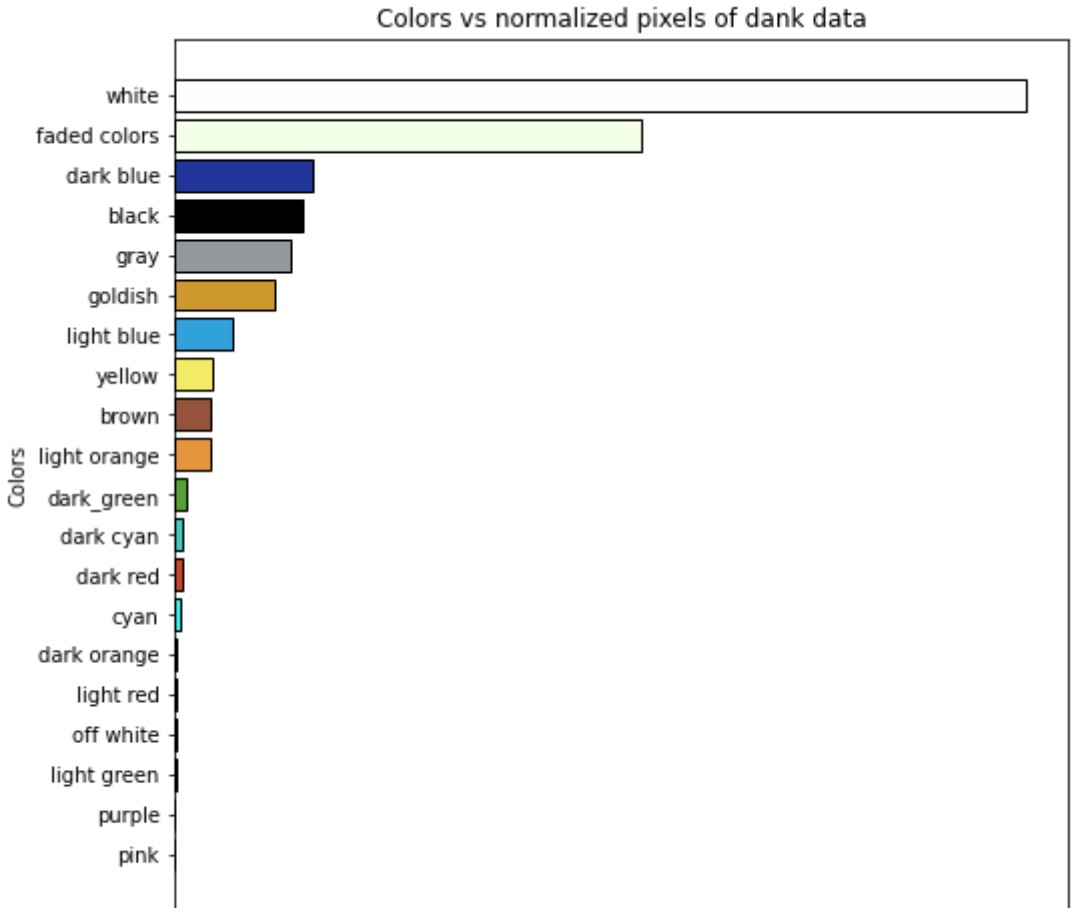

```
#Getting on dank meme data
dank = df[df['dank_or_not']==1]
```

```
#adding all the normalized pixels of a color and dividing it by number of data ava
white     = dank['white'].sum()/dank.shape[0]
black     = dank['black'].sum()/dank.shape[0]
gray      = dank['gray'].sum()/dank.shape[0]
brown     = dank['brown'].sum()/dank.shape[0]
off_white = dank['off-white'].sum()/dank.shape[0]
dark_red  = dank['dark red'].sum()/dank.shape[0]
light_red = dank['light red'].sum()/dank.shape[0]
dark_orange   = dank['dark orange'].sum()/dank.shape[0]
light_orange  = dank['light orange'].sum()/dank.shape[0]
goldish       = dank['goldish'].sum()/dank.shape[0]
yellow        = dank['yellow'].sum()/dank.shape[0]
dark_green    = dank['dark green'].sum()/dank.shape[0]
light_green   = dank['light green'].sum()/dank.shape[0]
dark_cyan     = dank['dark cyan'].sum()/dank.shape[0]
cyan          = dank['cyan'].sum()/dank.shape[0]
dark_blue     = dank['dark blue'].sum()/dank.shape[0]
light_blue    = dank['light blue'].sum()/dank.shape[0]
purple        = dank['purple'].sum()/dank.shape[0]
pink          = dank['pink'].sum()/dank.shape[0]
faded_colors  = dank['faded colors'].sum()/dank.shape[0]
```

```
#creating a dictionary to store colors and their corresponding pixel value
x = {'white':white,'black':black,'gray':gray,'brown':brown,'off white':off_white,
     'dark red':dark_red,'light red':light_red,'dark orange':dark_orange,'light ora
     'goldish':goldish,'yellow':yellow,'dark_green':dark_green,'light green':light_
     'dark cyan':dark_cyan,'cyan':cyan,'dark blue':dark_blue,'light blue':light_blu
     'pink':pink,'faded colors':faded_colors}
```

```
#sorting the dictionary
x = dict(sorted(x.items(), key=lambda item: item[1]))
```

```
#plotting Color vs Normalized pixels
plt.figure(figsize=(8,8))
color = plt.barh(list(x.keys()),list(x.values()), height=0.8)

#setting different colors for different bars
color[-1].set_color('#FFFDFD')
color[-1].set_edgecolor('black')
color[-2].set_color('#F3FFE6')
color[-2].set_edgecolor('black')
color[-3].set_color('#22369A')
color[-3].set_edgecolor('black')
color[-4].set_color('#000000')
color[-4].set_edgecolor('black')
color[-5].set_color('#93989A')
color[-5].set_edgecolor('black')
color[-6].set_color('#CD982D')
color[-6].set_edgecolor('black')
color[-7].set_color('#30A1DA')
color[-7].set_edgecolor('black')
color[-8].set_color('#F3EA66')
color[-8].set_edgecolor('black')
color[-9].set_color('#94533B')
color[-9].set_edgecolor('black')
color[-10].set_color('#E6953E')
color[-10].set_edgecolor('black')
color[-11].set_color('#599F36')
color[-11].set_edgecolor('black')
color[-12].set_color('#44C8BF')
color[-12].set_edgecolor('black')
color[-13].set_color('#C8462C')
color[-13].set_edgecolor('black')
color[-14].set_color('#35F1F1')
color[-14].set_edgecolor('black')
color[-15].set_color('#D7832F')
color[-15].set_edgecolor('black')
color[-16].set_color('#EC5234')
color[-16].set_edgecolor('black')
color[-17].set_color('#71EC34')
color[-17].set_edgecolor('black')
color[-18].set_color('#C8C098')
color[-18].set_edgecolor('black')
color[-19].set_color('#AE2CC8')
color[-19].set_edgecolor('black')
color[-20].set_color('#E633AA')
color[-20].set_edgecolor('black')

plt.title('Colors vs normalized pixels of dank data')
plt.xlabel('Normalized pixels')
plt.ylabel('Colors')
plt.show()
```



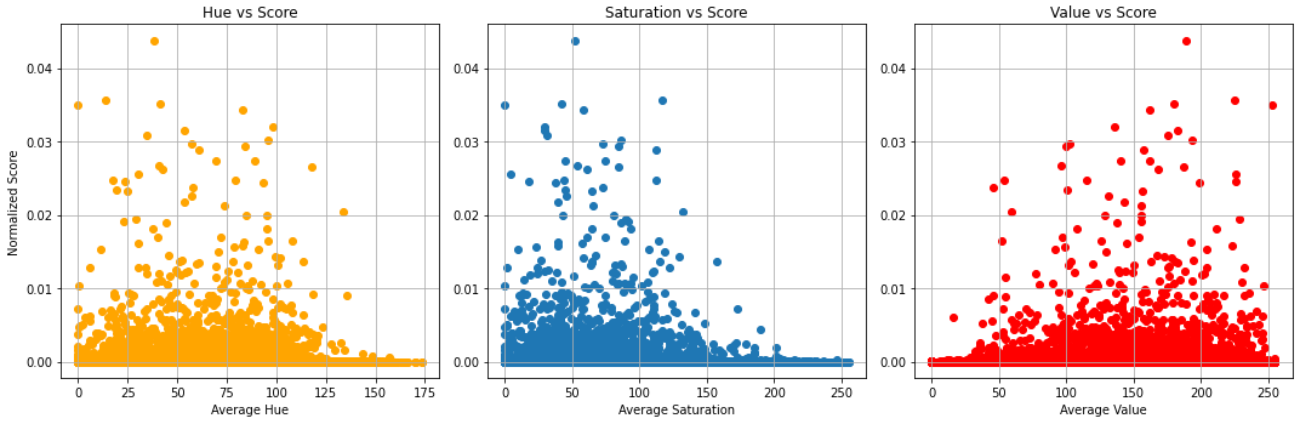

Dank memes colors are approximately same as the whole data. Means memers should not bother about the picture quality or putting bright tangy colors on the memes, the content of the meme should be good and people love them.

## HSV vs Normalized score

Visualizing average Hue, Saturation and Value vs Normalized score.

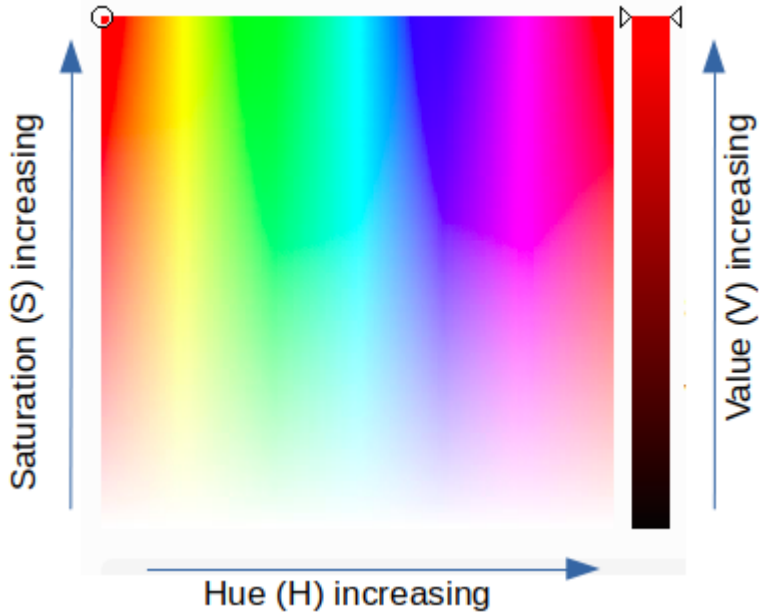
```
plt.figure(figsize=(15,5))

#plotting average hue vs normalized score
plt.subplot(1,3,1)
plt.scatter(df['avg_h'],df['score'], color='orange')
plt.title('Hue vs Score')
plt.xlabel('Average Hue')
plt.ylabel('Normalized Score')
plt.grid()

#plotting average saturation vs normalized score
plt.subplot(1,3,2)
plt.scatter(df['avg_s'],df['score'])
plt.title('Saturation vs Score')
plt.xlabel('Average Saturation')
plt.grid()

#plotting average value vs normalized score
plt.subplot(1,3,3)
```

```
plt.scatter(df['avg_v'],df['score'], color='red')
plt.title('Value vs Score')
plt.xlabel('Average Value')
plt.grid()

plt.tight_layout()
plt.show()
```




From above plots we can see that:

- For a meme to get high score (upvotes minus downvotes) hue should be low (approximately between 10 to 125 in opencv).
- For a meme to get high score the saturation value should also be low (approximately between 0 to 140 in opencv).
- For a meme to get high score the value should be higher (approximately between 100 to 200 in opencv).
  Figure below will provide a rough idea:

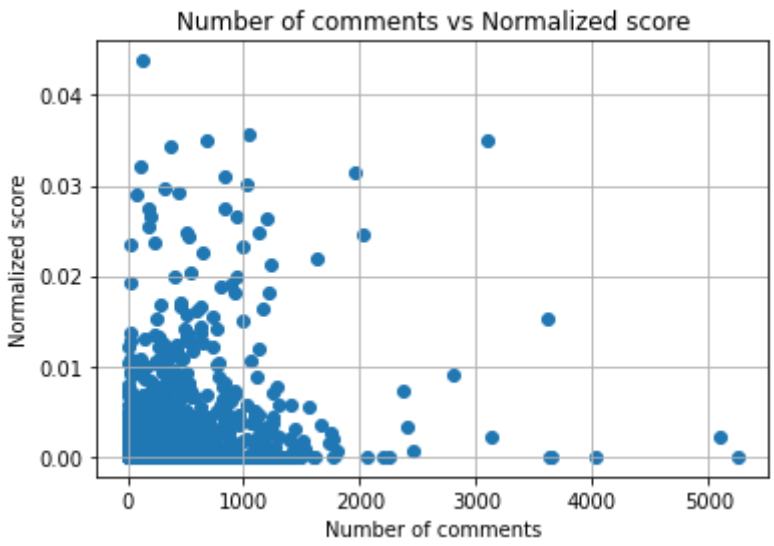



```
df['is_original_content'].value_counts()
```

```
0    83154
1      672
Name: is_original_content, dtype: int64
```

## Number of comments vs Normalized score

```
plt.scatter(df['num_comments'], df['score'])
plt.title('Number of comments vs Normalized score')
plt.xlabel('Number of comments')
plt.ylabel('Normalized score')
plt.grid()
plt.show()
```

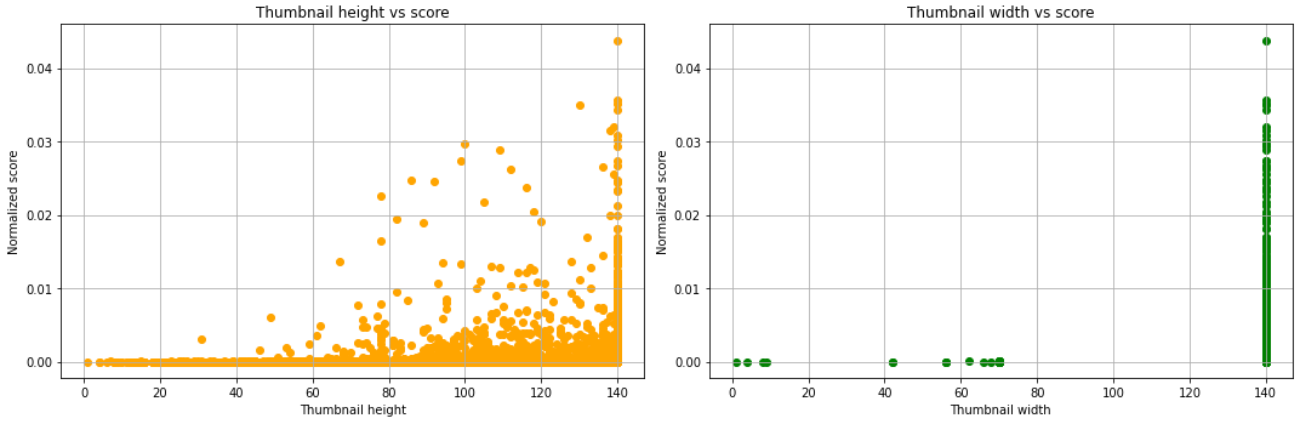



Here we can see that there is no clear relation between number of comments and Normalized score, because even memes with zero number of comments have high scores and memes with

large number of comments have low scores.

## Thumbnail height and width vs normalized score

```
#thumbnail height
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.scatter(df['thumbnail_height'], df['score'], color='orange')
plt.title('Thumbnail height vs score')
plt.xlabel('Thumbnail height')
plt.ylabel('Normalized score')
plt.grid()

#thumbnail width
plt.subplot(1,2,2)
plt.scatter(df['thumbnail_width'], df['score'],color='green')
plt.title('Thumbnail width vs score')
plt.xlabel('Thumbnail width')
plt.ylabel('Normalized score')
plt.grid()

plt.tight_layout()
plt.show()
```




Mostly thumbnail with height greater than 80 have high scores and almost all of the thumbnails have 140 width means there is no correlation between score and thumbnail width.