1 Do hot naked selfies get the most likes on Instagram?

1.1 A case study

My favorite instagrammer once said something along the lines of "I have thousands of professional modeling photos, but all you guys want to see is mirror selfies of me in my underwear."

Is this true? Would the average viewer prefer a candid lingerie photo over a glitzy shot on set with hair, makeup, wardrobe, etc? I decided to find out.

1.1.1 Imports

In [1]:
```
import numpy as np
import pandas as pd
from scipy.stats import f_oneway, linregress, kruskal
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import urllib.request
from InstagramAPI import InstagramAPI
```

1.1.2 Instagram API

Shout out to LevPasha (https://github.com/LevPasha/Instagram-API-python) on GitHub for writing these APIs. He did all the hard work.

In [2]:
```
api = InstagramAPI('username', 'password')
```

Login to my account.

In [3]:
```
api.login()
```

```
Request return 405 error!
{'message': '', 'status': 'fail'}
Login success!
```

Out[3]:
```
True
```

1.1.3 Instagrammer Selection

Obviously to do this in real depth would require pulling data from at least hundreds of users. That way you can control for differences between a given user's following and with more data points the trends found tend to be more extrapalatable. For example, different cultures have different beauty standards. People will follow different versions of what they consider attractive on Instagram and this will ripple over into which photos get the most likes and comments. Thus only looking at a single user won't necessarily provide insight into greater trends. However, I don't have that kind of free time.

My crush du jour is Teddy Quinlivan (https://www.instagram.com/teddy_quinlivan/), so we'll be using her account as the data set. She has over 1000 photos ranging from walking the Louis Vuitton runway to pulling lewks at China Chalet to random art finds. There's a lot of hot content on her account, but there's also a plethora of content that might disinterest the typical male @emrata (https://www.instagram.com/emrata/) follower. Hopefully this will give us enough variation to see if the hot selfies really do get all the likes.

Using the API we'll search for her account by handle and find her account number.

```
In [4]: api.searchUsername('teddy_quinlivan')
        api.LastJson
```

```
Out[4]: {'user': {'pk': 1904001441,
          'username': 'teddy_quinlivan',
          'full_name': 'Theodora Quinlivan',
          'is_private': False,
          'profile_pic_url': 'https://scontent-lga3-1.cdninstagram.com/v/t51.2885-19/s150x
        150/47695016_2286338751596772_7740193201025187840_n.jpg?_nc_ht=scontent-lga3-1.cdn
        instagram.com&_nc_ohc=MTnYvmAYn6AAX8yN19a&oh=bb2a4a7887109b66f738e0a5545acc98&oe=5
        E928187',
          'profile_pic_id': '1953616360206528395_1904001441',
          'is_verified': True,
          'has_anonymous_profile_picture': False,
          'media_count': 1178,
          'geo_media_count': 0,
          'follower_count': 122596,
          'following_count': 769,
          'following_tag_count': 0,
          'biography': 'Live Fast, Die Last \n👠🎱🎎🔪🌹🔮\nVision - Los Angeles\nSelect
        - London\nPremium - Paris \nMonster - Milan\nSupreme - NYC',
          'biography_with_entities': {'raw_text': 'Live Fast, Die Last \n👠🎱🎎🔪🌹🔮\nVi
        sion - Los Angeles\nSelect - London\nPremium - Paris \nMonster - Milan\nSupreme
```

1.1.4 Scraping the data

Now that we've found her account number we can get her entire feed and all meta data with a single command.

```
In [5]: user_data = api.getTotalUserFeed('1904001441')
```

From the raw data I'd like to collect 5 features: the timestamp, number of likes, number of comments, the caption, and location.

The API grabs the data with the most recent post as 0, so we'll reverse iterate through it so we can save the photos numbered from 0 being the first photo she ever posted to n being the most recent. I've placed all the feature grabbing lines in try/excepts because I found some photos had comments disabled or no location data as the feature was not part of the original app. For posts without this data 0/nulls were inserted into the data set.

I made a TQ_Data folder on my desktop which will hold all of her photos numbered by post and a csv containing the metadata. The index of the csv will match the photo number.

In [6]:
```
columns = ['timestamp', 'likes', 'comments', 'caption', 'location']
ar = np.zeros((len(user_data), len(columns)))
df = pd.DataFrame(ar, columns=columns)
print(len(user_data))
for i in range(len(user_data) - 1, -1, -1): # loop backwards so first photo is 0 and 
    try:
        time = user_data[i]['taken_at']
    except:
        time = ''
    try:
        likes = user_data[i]['like_count']
    except:
        likes = 0
    try:
        comments = user_data[i]['comment_count'] # comments may be disabled
    except:
        comments = 0
    try:
        caption = user_data[i]['caption']['text']
    except:
        caption = ''
    try:
        location = user_data[i]['location']['name'] # location wasn't always a featur
    except:
        location = ''

    df.loc[len(user_data) - i - 1, :] = [time, likes, comments, caption, location]

    try:
        link = TQ_data[i]['image_versions2']['candidates'][0]['url']
        urllib.request.urlretrieve(link, f'~/Desktop/TQ_Data/TQ_{len(TQ_data) - i - 1
    except:
        try:
            link = TQ_data[i]['carousel_media'][1]['image_versions2']['candidates'][0
            urllib.request.urlretrieve(link, f'~/Desktop/TQ_Data/TQ_{len(TQ_data) - i
        except:
            pass

df.to_csv('~/Desktop/TQ_Data/TQ_Data12.csv')
```

1.2 Data Labeling

The funny thing about machine learning is that the intial steps are pretty human intensive. Before computers and data scientists can do anything with image data the data has to be labeled. Humans know cats vs dogs by sight, but a computer has to relate it to the 3 letter word c-a-t.

That means I went through over 1000 photo manually labeling features: rating, showing, taken_by.

Let's break this down because this is the most controversial/important part. Classifying images in the context of this study is subjective, so I made the decision to add a couple new features to help define each image. That being said, the labels used were predominantly a product of the model chosen. Given this account's content there was no need to create labels such as gym pic or booty pic as Teddy is neither a "fit-chick" nor bikini model. This once again reduces the likelihood that trends found are reflected in the greater model/influencer Instamgram community. At any rate, these are the features and associated labels defined and I already regret not including thirst trap.

Rating: a measure of attractiveness

- Casual: a photo of the user in a normal day-to-day context
- Glamour Shot: a photo that evidences prior setup (eg. a fashion show or something clearly not candid)
- Hot: a photo of the user clothed where it seems a more than usual effort was made to look attractive
- Not Self: a photo of anything other than the user (may include other people, objects, drawings, etc)
- Nude: a photo where the user is naked
- Scantily Clad: a photo where the user is wearing highly revealing clothing

Showing: a measure how much of the user's body is shown

- Body: a photo containing either the entirety or majority of the user's body
- Bust: a photo of the user from the bust up
- Head: a photo of the user's head
- null: photos not of the user are marked null

Taken By: tagging the photographer

- Amateur: a photo taken by any non-professional
- Professional: a photo taken by what appears to be a career photographer
- Self: a photo taken by the user - a selfie in the modern parlance
- null: photos not of the user are marked null

1.2.1 Subjectivity

The choosing of these features and each one of their facets is a highly subjective matter. One could argue that just by the items chosen I have already skewed that data and introduced a bias. Probably. I acknowledge there are better ways to do this, for example conducting preliminary studies to see how other people categorize the images or having multiple people label the photos to control for personal preferences.

I'm doing this for fun, so doing all of that would have been a lot more than I bargained for. It just means that the results in this study are dependant on the labels used and the account chosen. Whether the conclusions in here map over to the rest of Instagram... who knows. Maybe another day I or another may take this up again and conduct it with more scientific rigor.

1.2.2 Examples

Before going on, it might be helpful to see an example of each of the categories defined above. Examples of several label combinations are shown in the appendix. First let's make a function to easily plot images by their number.

In [7]:
```
def plot_img(num):
    plt.figure(figsize=(5, 5))
    plt.axis('off')
    plt.imshow(mpimg.imread(f'/Users/XXI/Desktop/TQ_Data/TQ_{num}.jpg'));
```

1.3 Analysis

1.3.1 Data: import, clean, augment

In [8]:
```
data = pd.read_csv('/Users/XXI/Desktop/TQ_Data/TQ_Data.csv', header=0, low_memory=Fal
display(data.head(3))
```

	photo_num	timestamp	likes	comments	caption	location	rating	showing	taken_by
0	1192	1588103199	4765	113	A-K-R-I-S spring 2020 Campaign 💙 Thank you so ...	Paris, France	glamour shot	body	professional
1	1191	1588003187	1102	9	👁️	Los Angeles, California	not self	NaN	NaN
2	1190	1587754491	13576	291	Good Sex, No Stress, One Boo, No Ex, Small Cir...	Los Angeles, California	scantily clad	body	self

dsfsd

In [9]:
```
data.fillna('null', inplace=True)
print(data['rating'].unique())
print(data['showing'].unique())
print(data['taken_by'].unique())
```

```
['glamour shot' 'not self' 'scantily clad' 'casual' 'hot' 'nude']
['body' 'null' 'bust' 'head']
['professional' 'null' 'self' 'amateur']
```

dsfdsf

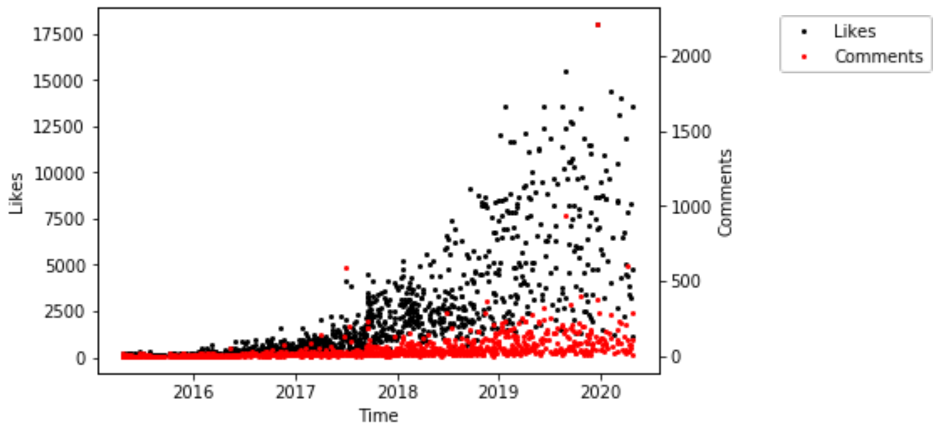
In [10]:
```
data['datetime'] = pd.to_datetime(data['timestamp'], unit='s')
data['date'] = data['datetime'].dt.date
data['year'] = data['datetime'].dt.year.astype(int)
data['day'] = data['datetime'].dt.dayofweek.astype(int)
data['hour'] = data['datetime'].dt.strftime('%H').astype(int)
display(data.head(3))
```

	photo_num	timestamp	likes	comments	caption	location	rating	showing	taken_by	datetime
0	1192	1588103199	4765	113	A-K-R-I-S spring 2020 Campaign 💙 Thank you so ...	Paris, France	glamour shot	body	professional	2020-04-28 19:46:39
1	1191	1588003187	1102	9	👁	Los Angeles, California	not self	null	null	2020-04-27 15:59:47
2	1190	1587754491	13576	291	Good Sex, No Stress, One Boo, No Ex, Small Cir...	Los Angeles, California	scantily clad	body	self	2020-04-24 18:54:51

1.3.2 First look

The plot below shows likes and comments on photos over time. What's interesting to notice is the large spread in likes on the more recent photos. While some photos begin to amass over 10k likes regularly, others still fail to get over 2.5k. Not all photos necessarily got more likes as her following grew.

In [11]:
```
fig, ax = plt.subplots()
plt1 = ax.plot_date(data['date'], data['likes'], color='k', markersize=2)
ax0 = ax.twinx()
plt2 = ax0.plot_date(data['date'], data['comments'], color='r', markersize=2)
ax.set_xlabel('Time')
ax.set_ylabel('Likes')
ax0.set_ylabel('Comments')
plts = plt1 + plt2
labels = ['Likes', 'Comments']
ax.legend(plts, labels, bbox_to_anchor=[1.5, 0, 0, 1]);
```



1.3.3 Detrend

When someone joins Instagram they have 0 followers. The increase in likes over time is directly correlated to the number of followers she's gained over the years. Unfortunately, it's not possible get the number of followers in time, so we can't detrend it directly based on follower count. To truly compare likes in one photo to the next, especially over time we need to detrend the data another way. Let's first take the natural log of the likes and comments.

In [12]:
```
data0 = data.copy()
data0['likes'] = np.log(data0['likes'] + 1)
data0['comments'] = np.log(data0['comments'] + 1)
```

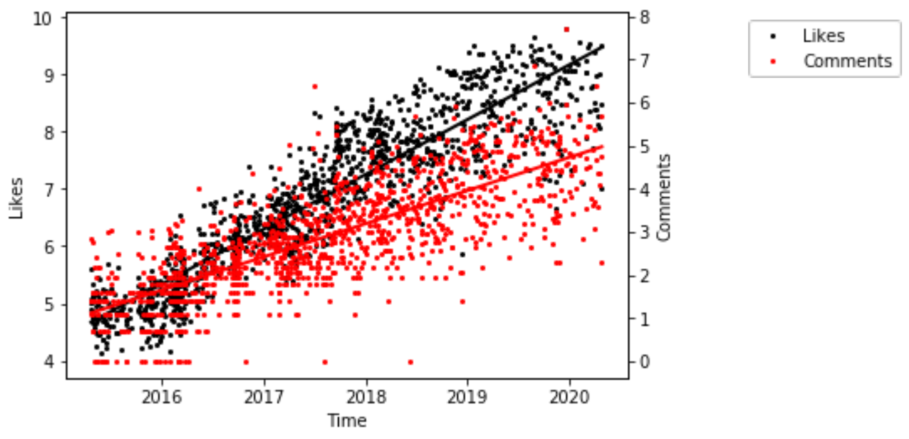
In the plot below we'll see that the data is already a lot neater and a more evident trend appears. So let's go ahead and make a linear regression of the likes and comments.

In [13]:
```
linreg_likes = linregress(data0['timestamp'], data0['likes'])
linreg_comments = linregress(data0['timestamp'], data0['comments'])
print(linreg_likes)
print(linreg_comments)
```

```
LinregressResult(slope=3.033517640182872e-08, intercept=-38.69763788807153, rvalue
=0.913692480812176, pvalue=0.0, stderr=3.909767975785163e-10)
LinregressResult(slope=2.4329443186707576e-08, intercept=-33.667519565206305, rval
ue=0.7765580821432169, pvalue=3.3540393430796435e-241, stderr=5.719714814442726e-1
0)
```

Plotting the data now with the linear regressions it seems there's a pretty good fit. Note that the streaking pattern in comments on the lower left is caused by photos receiving few comments early in her account history. If photos are only getting 1-10 comments the spread is quite small relative to photos getting 10-100 comments, hence why the comments scatter plot starts off streaked and then becomes more scattered.

In [14]:
```
fig, ax = plt.subplots()
plt1 = ax.plot_date(data0['date'], data0['likes'], color='k', markersize=2)
ax.plot_date(data0['date'], data0['timestamp']*linreg_likes[0] + linreg_likes[1], co
ax0 = ax.twinx()
plt2 = ax0.plot_date(data0['date'], data0['comments'], color='r', markersize=2)
ax0.plot_date(data0['date'], data0['timestamp']*linreg_comments[0] + linreg_comments
ax.set_xlabel('Time')
ax.set_ylabel('Likes')
ax0.set_ylabel('Comments')
plts = plt1 + plt2
labels = ['Likes', 'Comments']
ax.legend(plts, labels, bbox_to_anchor=[1.5, 0, 0, 1]);
```

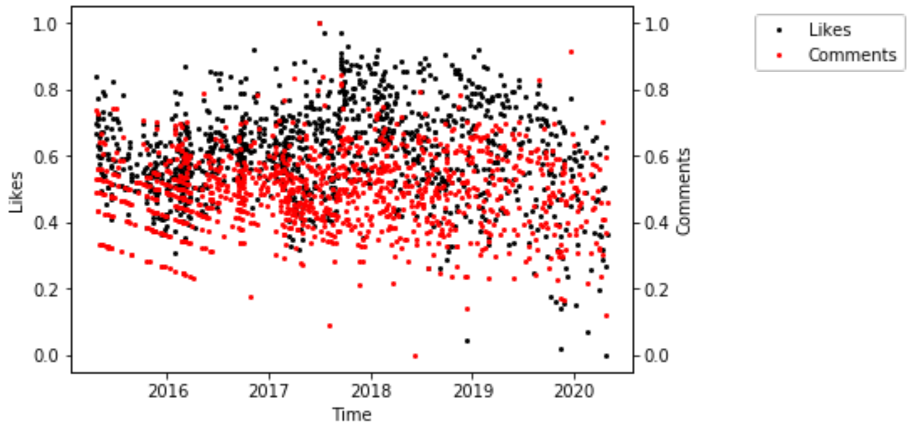


Now let's subtract out the regression and normalize the values between 0 and 1. This will fully detrend the data a give an easy to use metric: normalized likes and comments.

In [15]:
```
data0['likes'] -= data0['timestamp']*linreg_likes[0] + linreg_likes[1]
data0['comments'] -= data0['timestamp']*linreg_comments[0] + linreg_comments[1]

data0['likes'] = (data0['likes'] - min(data0['likes']))/(max(data0['likes']) - min(d
data0['comments'] = (data0['comments'] - min(data0['comments']))/(max(data0['comment
```

Finally we have a fully detrended and normalized data set. These values will make it possible to compare likes and comments over time from one image to another.

In [38]:
```
fig, ax = plt.subplots()
plt1 = ax.plot_date(data0['date'], data0['likes'], color='k', markersize=2)
ax0 = ax.twinx()
plt2 = ax0.plot_date(data0['date'], data0['comments'], color='r', markersize=2)
ax.set_xlabel('Time')
ax.set_ylabel('Likes')
ax0.set_ylabel('Comments')
plts = plt1 + plt2
labels = ['Likes', 'Comments']
ax.legend(plts, labels, bbox_to_anchor=[1.5, 0, 0, 1]);
```



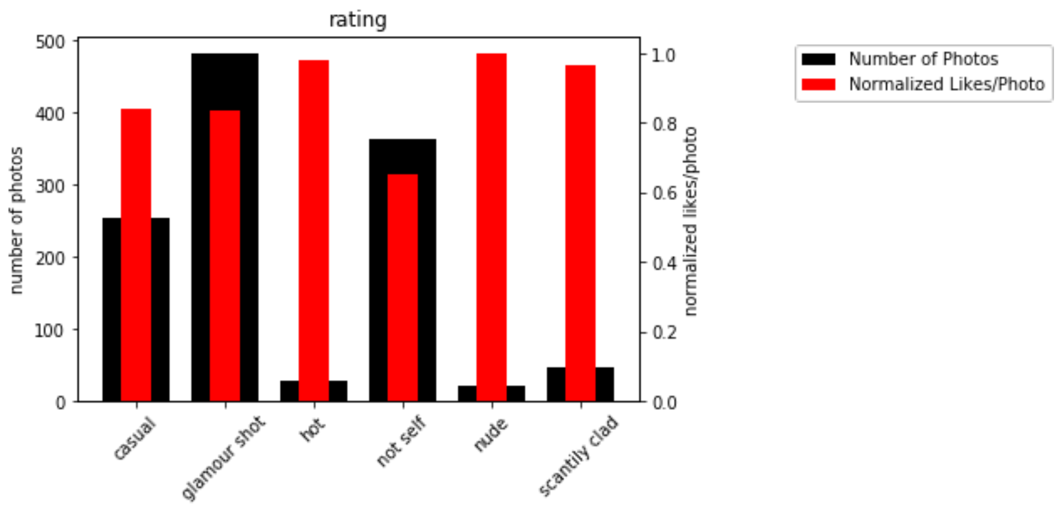
1.3.4 So do hot naked selfies get the most likes?

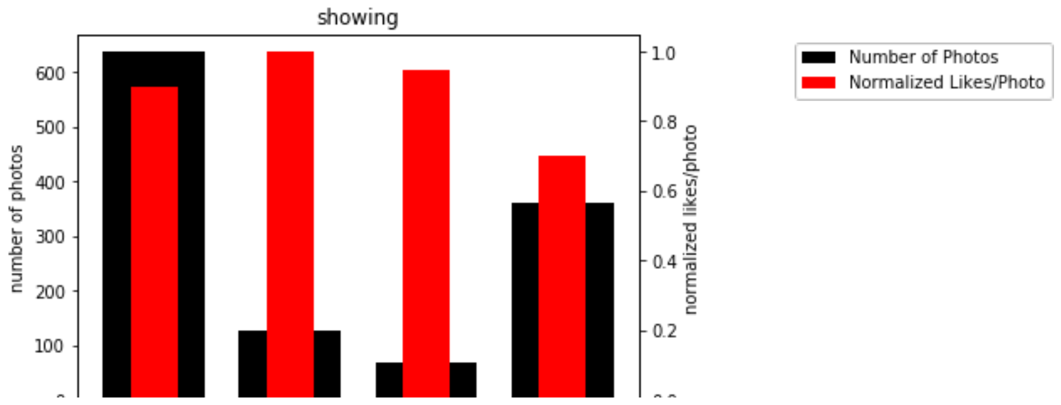
Yes, but not really. For each of the three main features there are plots below of the number of photos for each label in the feature and the number of normalized likes per photo for the label. Photos rated as nude, of the bust, taken by the user receive the most likes as independant features . However, photos rated as glamour shots bring in undefined normalized likes for every normalized like a nude photo gets. Even casual photos get about the same amount. Photos labeled as not self are the real losers bringing in only undefined normalized likes.

In the other categories we see a similar pattern among the labels. The spread between photos of the head, bust, and body isn't that large, but all three of those values differ a lot from the value for the label null. The same is true for the feature taken_by. All of this is to say that the biggest difference is between photos of the user vs photos not of the user. Hot photos do better than not hot photos, but as long as they're of the user the variance isn't that large.

In [29]:
```
features = ['rating', 'showing', 'taken_by']
def bar_category(df, category):
    category_x = np.sort(df[category].unique())
    category_y = []
    category_likes = []
    cat_var = []
    for i in category_x:
        category_y.append(len(df.loc[df[category] == i]))
        category_likes.append(sum(df.loc[df[category] == i]['likes'])/category_y[-1]

    category_likes = np.array(category_likes)/max(category_likes)
    fig, ax = plt.subplots()
    plt1 = ax.bar(category_x, category_y, width=0.75, color='k')
    ax0 = ax.twinx()
    plt2 = ax0.bar(category_x, category_likes, width=0.35, color='r')
    ax.set_xticklabels(category_x, rotation=45)
    ax.set_ylabel('number of photos')
    ax0.set_ylabel('normalized likes/photo')
    plts = [plt1, plt2]
    labels = ['Number of Photos', 'Normalized Likes/Photo']
    ax.set_title(category)
    ax0.legend(plts, labels, bbox_to_anchor=[1.75, 0, 0, 1])
    print(f'{category_x[np.argmax(category_likes)]} photos get the most normalized l
#     print(np.var(category_likes)/np.mean(category_likes))

for i in features:
    bar_category(data0, i)
```

```
nude photos get the most normalized likes per photo and not self get the least
bust photos get the most normalized likes per photo and null get the least
self photos get the most normalized likes per photo and null get the least
```





To dig a little deeper we can run an ANOVA test. The function below is written as to compare all the normalized likes for labels within a feature or all the the labels except one. If our p value is greater than 0.05 then we can say the difference betweeen normalized likes for each label is not statistically signifacant.

In the results we see incredibly small values indicating that the differences between normalized likes for each label cannot be ignored, however when the test is run omitting photos not of the user the p value shoots up by a factor ~10^{75} for all the features. In a separate notebook the test was run omitting other labels, but the p value never changed by as much. This massive difference further confirms that photos not of the user are bringing down the overall like count. This also further evidences that as long as she posts photos of herself her normalized likes per photo is quite high.

In [35]:
```
def ANOVA(df, category, omit=False):
    string = 'f_oneway('
    base_filter = 'df.loc[df[category] == i]["likes"]'
    for i in np.sort(df[category].unique()):
        if(i not in ['not self', 'null'] and omit):
            string += f'df.loc[df[category] == "{i}"]["likes"], '
        elif(not omit):
            string += f'df.loc[df[category] == "{i}"]["likes"], '
    string = string[:len(string) - 2]
    string += ')'
    if(not omit):
        print(f'{eval(string)} - all labels')
    else:
        print(f'{eval(string)} - all except photos not of herself\n')

for i in features:
    print(i)
    ANOVA(data0, i)
    ANOVA(data0, i, omit=True)

# print(data0.loc[~data0['rating'].isin(['0', 'not self'])]['likes'].mean())
# print(f_oneway(data0.loc[data0['rating'] == 'hot']['likes'], data0.loc[data0['rati
```

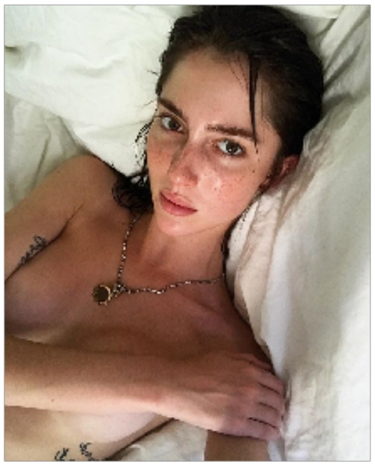
```
rating
F_onewayResult(statistic=104.32548294998702, pvalue=2.3862365396895665e-91) - all
labels
F_onewayResult(statistic=21.357563821153818, pvalue=8.828624263805188e-17) - all e
xcept photos not of herself

showing
F_onewayResult(statistic=159.79317463450153, pvalue=5.143068595098031e-87) - all l
abels
F_onewayResult(statistic=24.271514994386187, pvalue=5.7073929423833817e-11) - all
except photos not of herself

taken_by
F_onewayResult(statistic=145.76771064040003, pvalue=1.9584171470638345e-80) - all
labels
F_onewayResult(statistic=6.732497394207687, pvalue=0.0012578623402667059) - all ex
cept photos not of herself
```

1.3.5 The 🔥 HOTTEST🔥 pic

For kicks let's see what photos match the highest liked labels for each category: nude, bust, self. There are only three images that match this combinaiton of labels and after looking at one I'd probaby throw it a like too. The photo shown below got essentially half as many likes as photo 1104 though...

```
In [30]: top_img = data.loc[(data['rating'] == 'nude') & (data['showing'] == 'bust') & (data[
         display(top_img)
         plot_img(875)
```

	photo_num	timestamp	likes	comments	caption	location	rating	showing	taken_by	datetime
88	1104	1568841126	10552	185	That Thotty girl from Euphoria Dick Sucking ma...	Los Angeles, California	nude	bust	self	2019-09-18 21:12:06
317	875	1531264385	5606	81	Well-behaved women seldom make history	New York, New York	nude	bust	self	2018-07-10 23:13:05
1110	82	1444949583	126	3	Water nymph looks autumn/winter 2015 💧🐚 #Londo...	London, United Kingdom	nude	bust	self	2015-10-15 22:53:03



Well, these pics were over taken over a year apart and thus while it has lower absolute likes, it has a much higher normalzied like value of undefined. This means that relative to other photos and her follower count at the time getting undefined likes was an incredible amount.

```
In [57]: time_diff = top_img['timestamp'].diff()[top_img.index[1]]
         print(f'{round(abs(time_diff)/60/60/24/365, 1)} years apart')
         top_img_norm = data0.loc[data0.index.isin(top_img.index)]
         top_img.loc[:, 'normalized likes'] = top_img_norm.loc[:, 'likes']
         display(top_img.loc[:, ['photo_num', 'likes', 'normalized likes', 'date']].head(2))

1.2 years apart
```

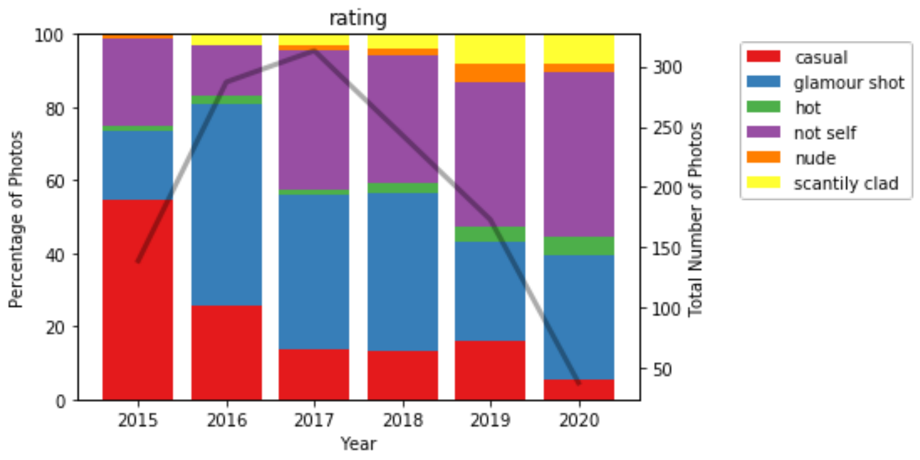
	photo_num	likes	normalized likes	date
88	1104	10552	0.704459	2019-09-18
317	875	5606	0.830360	2018-07-10

1.3.6 So why does it seem like hot pics get all the attention?

On Teddy's Instagram stories she complained that her modeling photos weren't getting any love. It's unlikely this belief came out of nowhere, so let's try to see if there's some other factor at play that makes the contrast more stark.

```
In [103]: def plot_categories_in_time(df, category):
              cmap = 'Set1'
              plt.rcParams['image.cmap'] = cmap
              plt.rcParams['axes.prop_cycle'] = plt.cycler(color=eval(f'plt.cm.{cmap}.colors'
          #     plt.figure()
              fig, ax = plt.subplots()

              years = np.sort(data['year'].unique())
              categories = list(np.sort(data[category].unique()))
              category_base = np.zeros(len(years))
              category_adders = np.zeros(len(years))
              category_trend = []
              for i in categories:
                  category_y = []
                  for j in range(len(years)):
                      subset = df.loc[(df['year'] == years[j]) & (df[category] == i)]
                      category_y.append(len(subset)/len(df.loc[(df['year'] == years[j])])*100
          #             category_y.append(len(subset)) # unhash to change plot to number of p
          #             category_y.append(subset['likes'].sum()) # unhash to change plot to s
                      category_adders[j] = category_y[-1]
                  ax.bar(years, category_y, bottom=category_base)
                  category_trend.append(linregress(years, category_y)[0])
                  category_base += category_adders
              print(f'{categories[np.argmax(category_trend)]} photos are on the rise while {c
              ax0 = ax.twinx()
              total = []
              for i in years:
                  total.append(len(df.loc[data['year'] == i])) # get y array for total photos
          #         total.append(sum(df.loc[data['year'] == i]['likes'])) # get y array for t
              ax0.plot(years, total, color='k', linewidth=3, alpha=0.33) # unhash to plot one
          #     categories.append('total')
              ax.set_xlabel('Year')
              ax.set_ylabel('Percentage of Photos')
              ax0.set_ylabel('Total Number of Photos')
              ax.set_title(category)
              ax.legend(categories, bbox_to_anchor=[1.5, 0, 0, 1])

          for i in features:
              plot_categories_in_time(data0, i)
```

```
not self photos are on the rise while casual images are being posted less
null photos are on the rise while body images are being posted less
null photos are on the rise while amateur images are being posted less
```

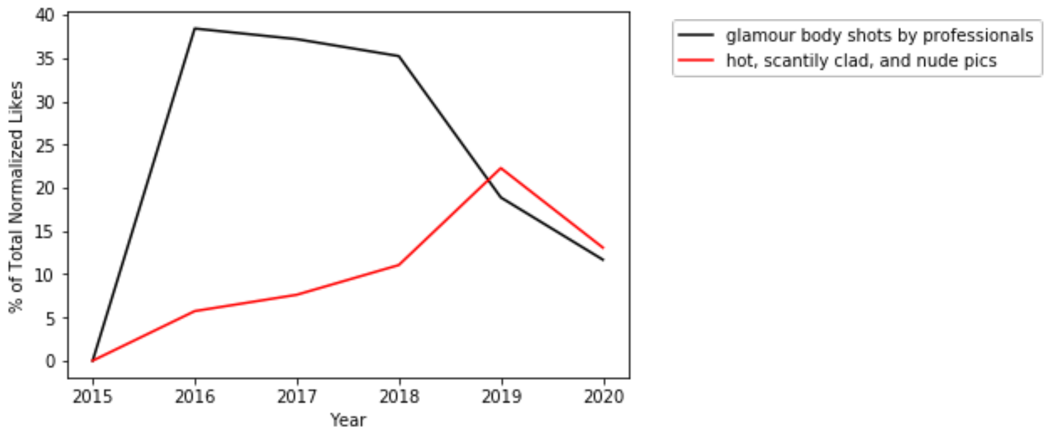


In [62]:
```
glam = data0.loc[(data0['rating'] == 'glamour shot') & (data0['showing'] == 'body') &
hot = data0.loc[data0['rating'].isin(['hot', 'scantily clad', 'nude'])]
not_self = data0.loc[data0['rating'] == 'not self']
```

Glamour shots of the body by professionals are 32.4% of all the photos Teddy posts, whereas photos rated hot, scantily clad, or nude combined make up only 8.0% of her posts to date. Simlarly, the glamour photos have brought in 33.5% of the normalized likes while hot pics brought in 9.8%. Note that the percentage of photos usually corresponds highly with the percentage of likes.

The real killers are the photos which are not of herself which are 30.3% of all her content, but bring in only 25.0% of the likes.

In [119]:
```
years = np.sort(data0['year'].unique())
glam_likes = []
hot_likes = []
not_self_likes = []
for i in years:
    glam_likes.append(data0.loc[(data0['year'] == i) & (data0['rating'] == 'glamour
    hot_likes.append(data0.loc[(data0['year'] == i) & (data0['rating'].isin(['hot',
    not_self_likes.append(data0.loc[(data0['year'] == i) & (data0['rating'] == 'not
plt1 = plt.plot(years, glam_likes, color='k')
plt2 = plt.plot(years, hot_likes, color='r')
# plt3 = plt.plot(years, not_self_likes)
plt.xlabel('Year')
plt.ylabel('% of Total Normalized Likes')
plt.legend(['glamour body shots by professionals', 'hot, scantily clad, and nude pi
```



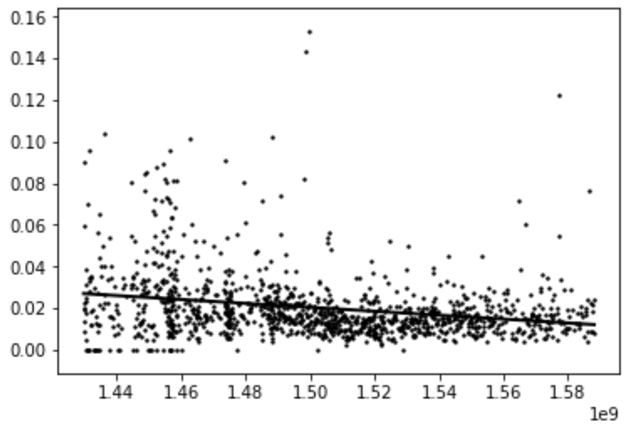
1.4 Conclusion

yes, but no

In [231]:
```
top_img = data.loc[(data['rating'] == 'glamour shot') & (data['showing'] == 'body')
print(round(len(top_img)/len(data), 3))

print(sum(top_img['likes'])/sum(data['likes']))

print(len(data.loc[data['rating'] == 'not self'])/len(data))
```

```
0.285
0.24719340011917385
0.2625318606627018
```

```
In [ ]:

In [7]:

In [ ]:

In [ ]:

In [10]:

In [ ]:

In [ ]:

In [97]: linreg_engagement = linregress(data['timestamp'], data['comments']/(data['likes'] +
         print(linreg_engagement)
         plt.scatter(data['timestamp'], data['comments']/data['likes'], color='k', s=2)
         plt.plot(data['timestamp'], data['timestamp']*linreg_engagement[0] + linreg_engageme

          LinregressResult(slope=-9.416168425715098e-11, intercept=0.1616232855650165, rvalu
          e=-0.24241962334704623, pvalue=2.0429280132605432e-17, stderr=1.0919407525737933e-
          11)
```



```
In [ ]: ##### print(data['comments'].idxmax())
        data.loc[data['comments'].idxmin(), :]
        # # print(len(data))
        # print(min(data['likes']))
        # TeddyQ correlation

In [ ]:
```

In [117]:
```
data.loc[data['likes'] < 500].head()
```

Out[117]:

	photo_num	timestamp	likes	comments	caption	location	rating	showing	taken_by
208	968	1544459378	353	3	REDKEN 🖤 \n#colorgelslacquers #redken @redken		not self		
452	724	1512403703	460	4	Music: Julian Winding - Disconnected	New York, New York	0		
540	636	1504047935	495	9	Drew Barrymore - Scream 1996	New York, New York	0		
545	631	1503358512	477	6	Ghost in the Shell - 1995	New York, New York	0		
555	621	1502471017	423	6	ああ助かった、金曜日だ	New York, New York	0		

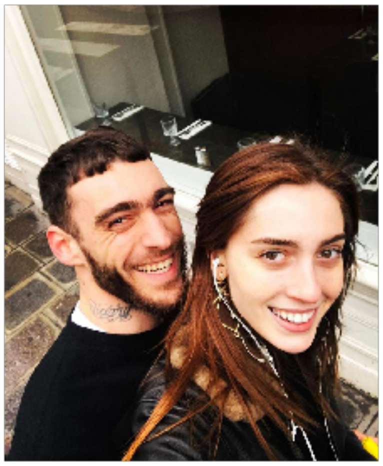
1.5 Appendix

1.5.1 Examples of Photo Labels

1.5.1.1 casual, head, self

This casual selfie with a friend only includes her head.

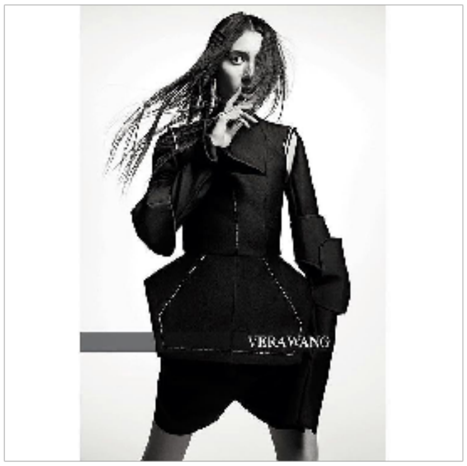
In [129]:
```
plot_img(477)
```



1.5.1.2 glamour shot, body, professional

The fashion brand name across the photo makes it obvious that this photo has been staged and was shot by a professional. We can see Teddy's entire body here.

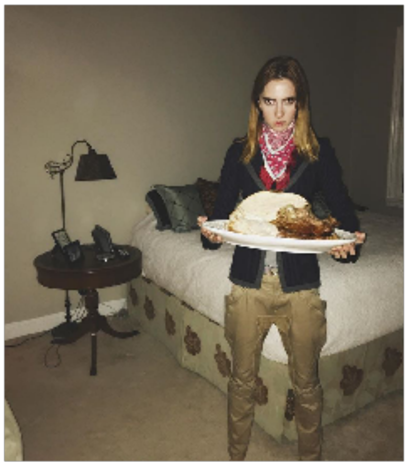
In [119]:
```
plot_img(893)
```



1.5.1.3 glamour shot, body, amateur

Here Teddy is clearly wearing a couture ensemble and her whole body is visible. However, this photo is not on a set and likely shot by a friend.

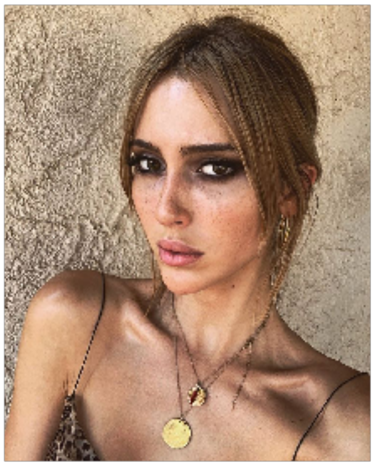
In [121]:
```
plot_img(410)
```



1.5.1.4 hot, bust, amateur

Between the makeup, jewely, and spaghetti strap top I think it's safe to say she didn't roll out of bed like this. The image is framed from slightly below the shoulders and up and most likely not taken by a professional.

In [122]:
```
plot_img(1079)
```



1.5.1.5 scantily clad, body, amateur

In this photo there is no guesswork. Just read the caption:

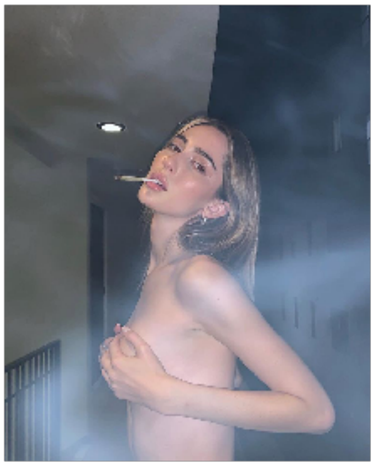
"Ok so last night I was invited to this **messy lingerie themed** Halloween party, and of course I'm so down because I'm basically a hooker and an attention whore so this party was right up my ally. I didn't end up having any fun cuz they make you sign an NDA at the door and it was a pussy fest (there were only girls there). Not only that, but I'm so used to being the only tall hot slut at parties, and this particular party was filled with so many hot girls who had way better bodies than me, so I didn't even stand out or feel special, I was just 1 in 1000 sexy hot slutty girls. I tried to take this picture in the bathroom to discreetly capture my glam and I would like to **thank the real MVP @giselleadorabrune (https://www.instagram.com/giselleadorabrune/) for Kneeling in a pile of toilette paper to take this picture** of me. Halloween parties are a overrated scam, next year I'm going as a ghost."

In [123]:
```
plot_img(1125)
```

1.5.1.6 nude, bust, amateur

This is about as naked as you can get on Instagram.

In [124]:
```
plot_img(1137)
```



1.5.1.7 not self, null, null

For this post Teddy used a meme for the image.

In [128]:
```
plot_img(1150)
```

1.5.2 All the permutations

These photos show all the possible labels in each catgory, but don't even begin to capture all the possible permutations between rating, showing, and taken by.

1.5.3 Refreshiging the Data

Naturally I didn't do this whole analysis in a day, so as new photos were posted it was essential to keep the numbers and records up to date. The code below is a modified version of the collection code in section 1.1.4 that reads what the last photo collected on my desktop was and updates from there. It also recollects all of the meta data as the likes and comments on past photos may have changed.

In [3]:
```
api = InstagramAPI('500daysofsingle', 'Blackmill1@')
```

In [4]:
```
api.login()
```

```
Request return 429 error!
{'message': 'Please wait a few minutes before you try again.', 'status': 'fail'}
Request return 405 error!
{'message': '', 'status': 'fail'}
Login success!
```

Out[4]:
```
True
```

In [5]:
```
api.getUserFeed('1904001441')
```

Out[5]:
```
True
```

In [6]:
```
# user_data = api.getTotalUserFeed('1904001441') # use this if they have more than 12
user_data = api.LastJson['items'] # otherwise this is much faster
print(len(user_data))
```

```
In [112]: columns = ['photo_num', 'timestamp', 'likes', 'comments', 'caption', 'location', 'r
          ar = np.zeros((len(user_data), len(columns)))
          df = pd.DataFrame(ar, columns=columns)
          for i in range(len(user_data) - 1, -1, -1):
          #     if(user_data[i]['taken_at'] == data.loc[data['timestamp'].idxmax(), 'timestam
              if(user_data[i]['taken_at'] == data.loc[data['timestamp'].idxmax(), 'timestamp'
                  counter = 0
                  for j in range(i - 1, -1, -1):
                      try:
                          time = user_data[j]['taken_at']
                      except:
                          time = ''
                      try:
                          likes = user_data[j]['like_count']
                      except:
                          likes = 0
                      try:
                          comments = user_data[j]['comment_count'] # comments may be disabled
                      except:
                          comments = 0
                      try:
                          caption = user_data[j]['caption']['text']
                      except:
                          caption = ''
                      try:
                          location = user_data[j]['location']['name'] # location wasn't alway
                      except:
                          location = ''

                      df.loc[j, :] = [0, time, likes, comments, caption, location, '', '', ''

                      try:
                          link = user_data[j]['image_versions2']['candidates'][0]['url']
                          urllib.request.urlretrieve(link, f'/Users/XXI/Desktop/TQ_Data/TQ_{l
                      except:
                          try:
                              link = user_data[j]['carousel_media'][1]['image_versions2']['ca
                              urllib.request.urlretrieve(link, f'/Users/XXI/Desktop/TQ_Data/T
                          except:
                              pass
                      counter += 1

          df = df.loc[df['timestamp'] != 0]
          df['photo_num'] = np.arange(len(data) + len(df) - 1, len(data) - 1, -1)
          display(df)
          df = pd.concat([data, df], sort=False)
          df.sort_values(by='timestamp', inplace=True)
          # df.to_csv('/Users/XXI/Desktop/TQ_Data/TQ_Data.csv', index=False)

In [ ]:
```