EDA

April 16, 2018

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
table00 = pd.read_json("data/merged-01-00", lines=True)
small = pd.read_json("data/merged-01-00-snippet", lines=True)
```

In [ ]:
```
# extras
table01 = pd.read_json("data/merged-01-01", lines=True)
table14 = pd.read_json("data/merged-02-14", lines=True)
table20 = pd.read_json("data/merged-03-20", lines=True)
```

In [220]:
```
pd.set_option('display.max_columns', 72)
 # table00.head(20)
```

In [40]:
```
# Testing for any bid_responses at all
# Somehow there are none in these 4 random samples
# Going to drop the columns for now bc apparently they are useless
print(len(table00[[rq != [] for rq in table00['bid_responses']]]), end=" ")
print(len(table01[[rq != [] for rq in table01['bid_responses']]]), end=" ")
print(len(table14[[rq != [] for rq in table14['bid_responses']]]), end=" ")
print(len(table20[[rq != [] for rq in table20['bid_responses']]]), end=" ")
```

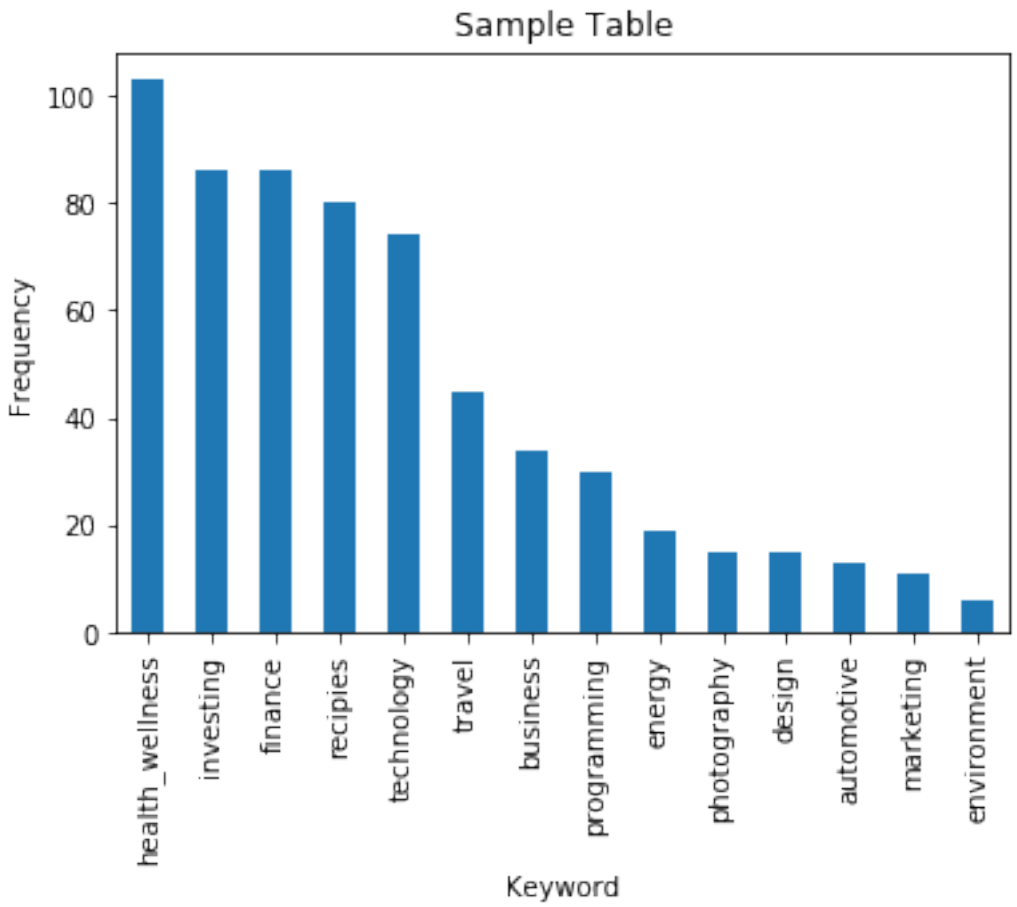
```
0 0 0 0
```

In [16]:
```
# splitting table up between the ads and the people clicking the ads
ads = small[small['keywords'] == '']
clickers = small[small['keywords'] != '']
```

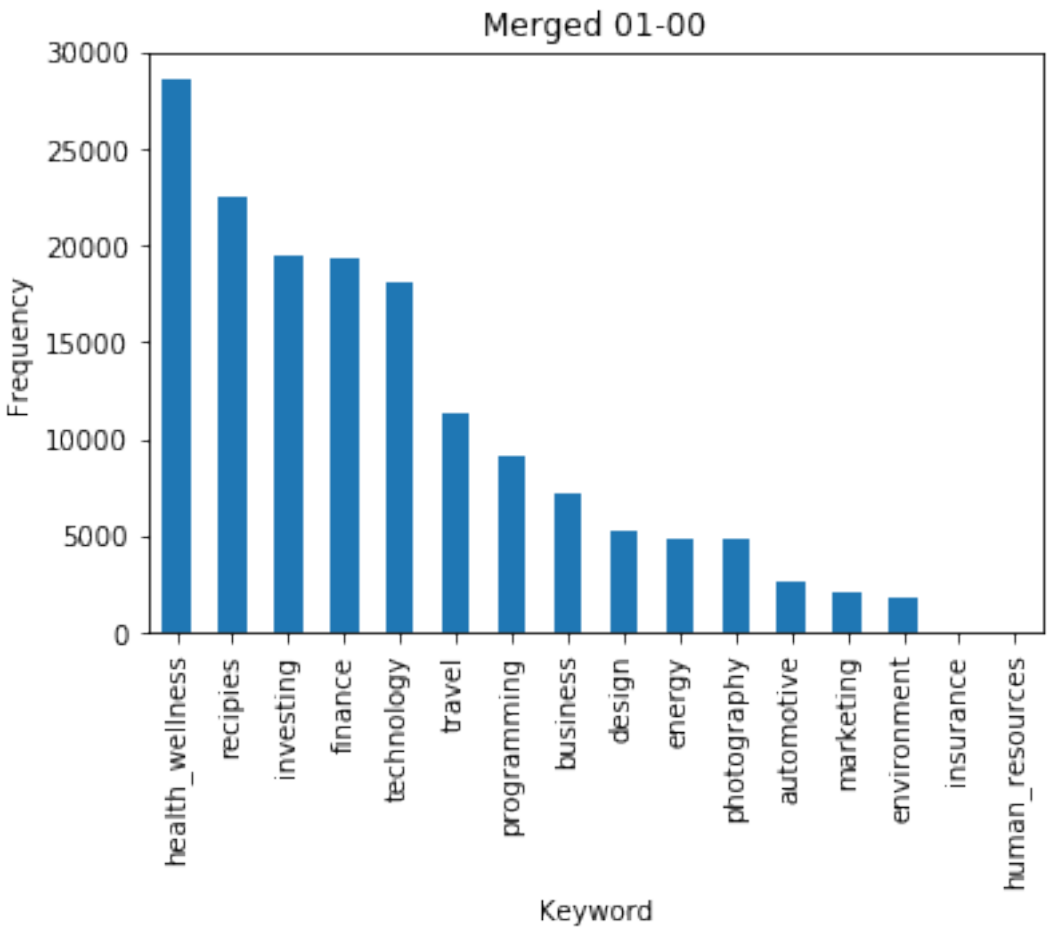
In [221]:
```
# for col in small.columns:
#     print(col + ":", end=" ")
#     try:
#         print(len(set(small[col])))
#     except TypeError as e:
#         print(e)
```

0.0.1 Graphs of what keywords were present for user rows from several different tables

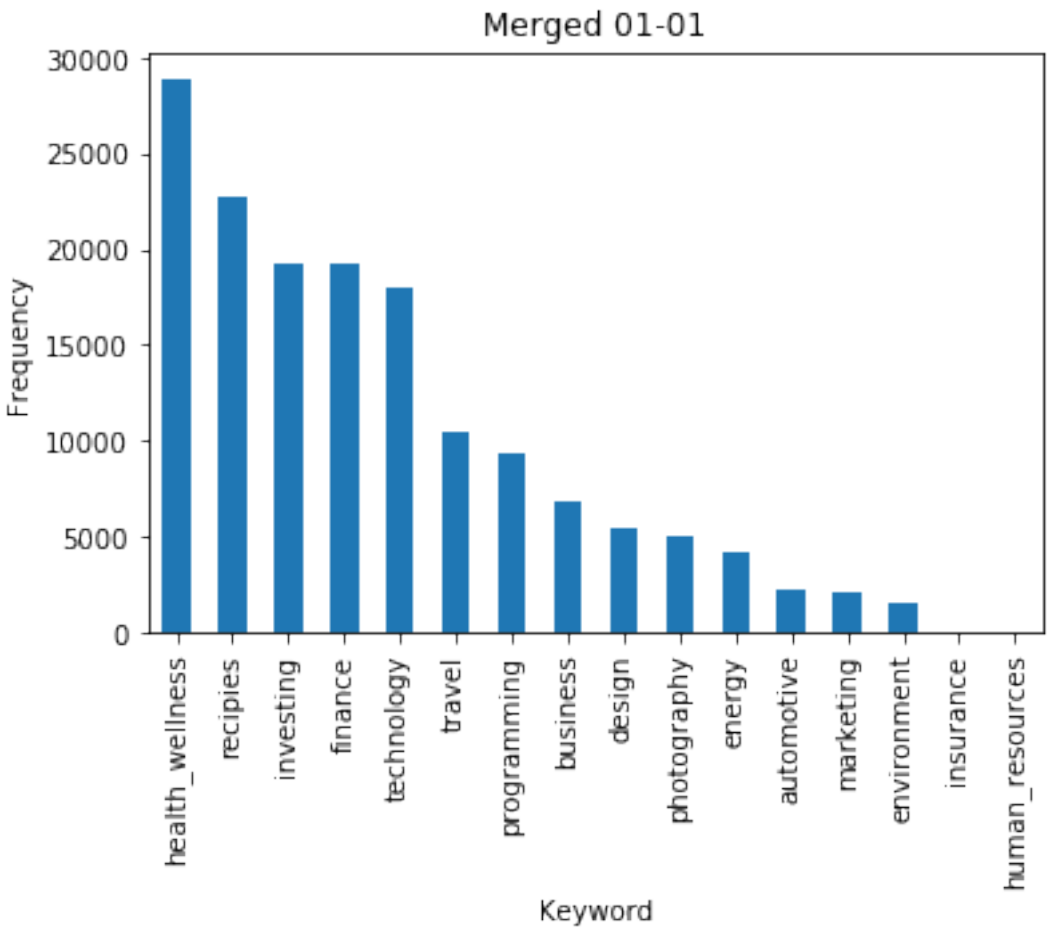
```
In [79]: from collections import defaultdict

         def graph_keywords(table, title):
             nonemptykeywords = table[table['keywords'] != ''] # this pulls out all of the use
             keywords = nonemptykeywords['keywords'] # pulls out only keywords column

             # We manually aggregate values because every cell value contains a string with po
             keyword_count = defaultdict(lambda: 0, {})
             for elem in keywords:
                 words = elem.split(',')
                 for word in words:
                     keyword_count[word] += 1

             PREFIX = "Pocket_SaveTopic_Month_"

         #     KEYWORDS = [k[len(PREFIX):] for k in keyword_count.keys()]
             KEYWORDS = list(keyword_count.keys())
             print("KEYWORDS =", KEYWORDS)

             count_map = {key[len(PREFIX):]: value for key, value in keyword_count.items()}
             s = pd.Series(count_map).sort_values(ascending=False)
             graph = s.plot(kind="bar", title=title)
             graph.set_xlabel("Keyword")
             graph.set_ylabel("Frequency")
             plt.show()



         graph_keywords(small, "Sample Table")
         graph_keywords(table00, "Merged 01-00")
         graph_keywords(table01, "Merged 01-01")
         graph_keywords(table14, "Merged 02-14")
         graph_keywords(table20, "Merged 03-20")

KEYWORDS = ['Pocket_SaveTopic_Month_photography', 'Pocket_SaveTopic_Month_design', 'Pocket_Save
```



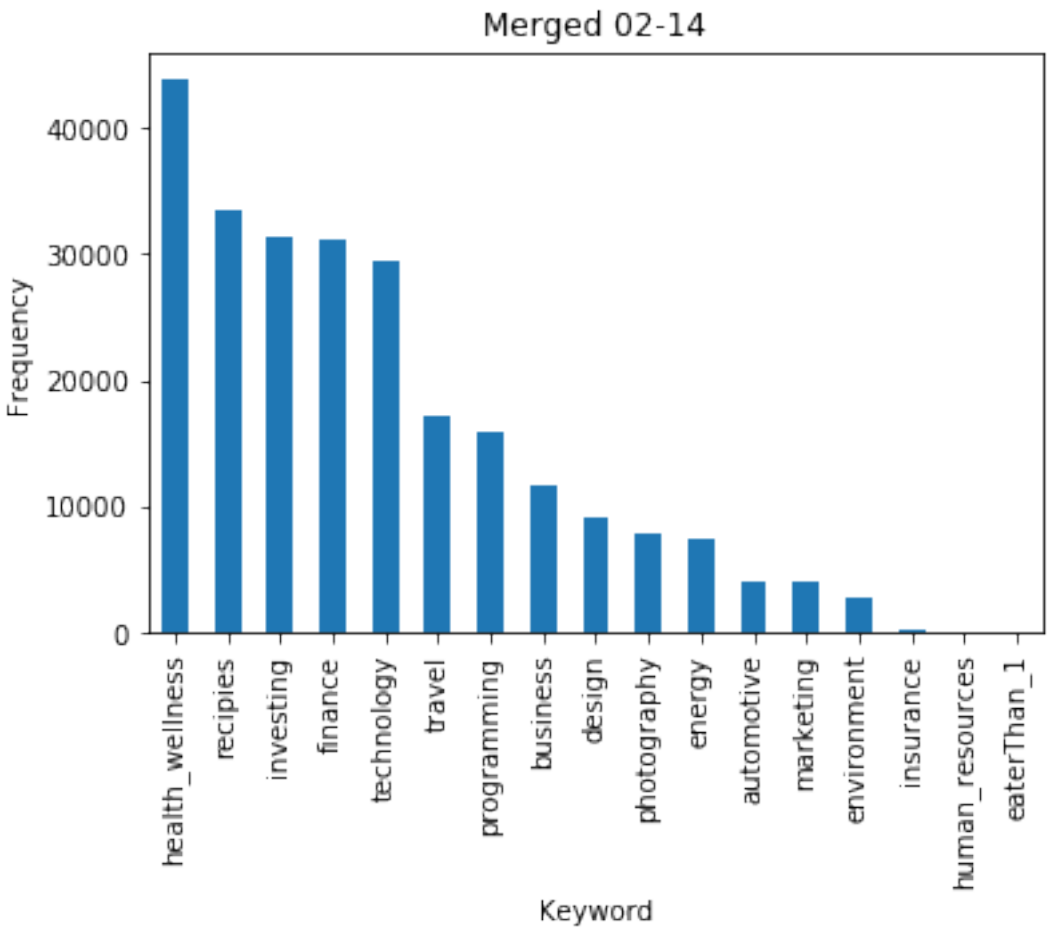

```
KEYWORDS = ['Pocket_SaveTopic_Month_photography', 'Pocket_SaveTopic_Month_design', 'Pocket_Save
```



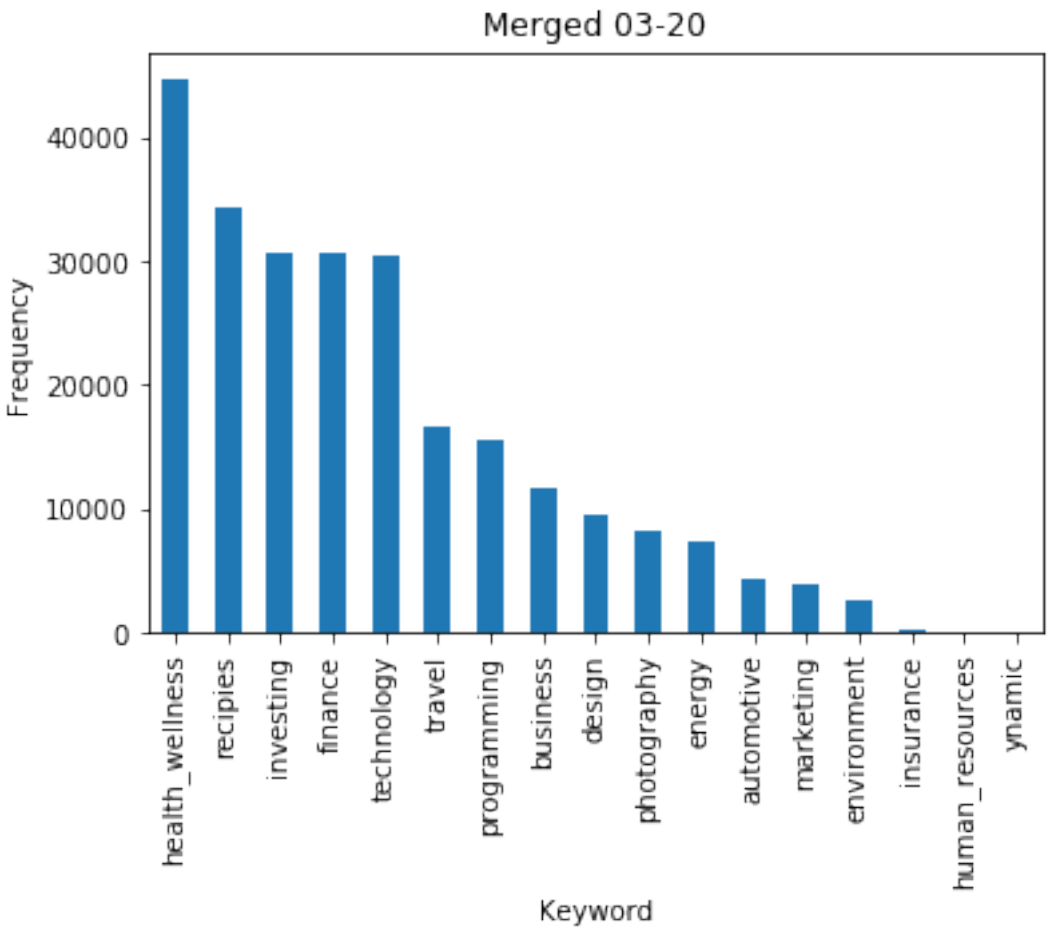

```
KEYWORDS = ['Pocket_SaveTopic_Month_design', 'Pocket_SaveTopic_Month_insurance', 'Pocket_SaveTo
```




```
KEYWORDS = ['Pocket_SaveTopic_Month_technology', 'Pocket_SaveTopic_Month_design', 'Pocket_Save
```



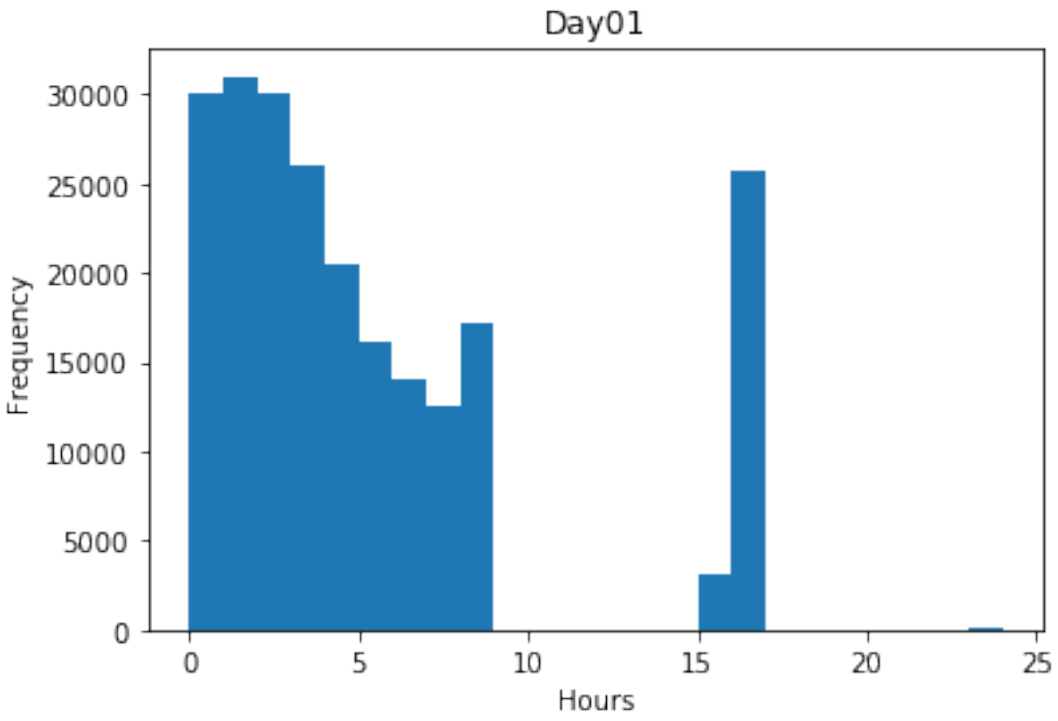

```
KEYWORDS = ['Pocket_SaveTopic_Month_design', 'Pocket_SaveTopic_Month_insurance', 'Pocket_SaveTo
```



0.0.2 Histogram of times during the day when ads were displayed

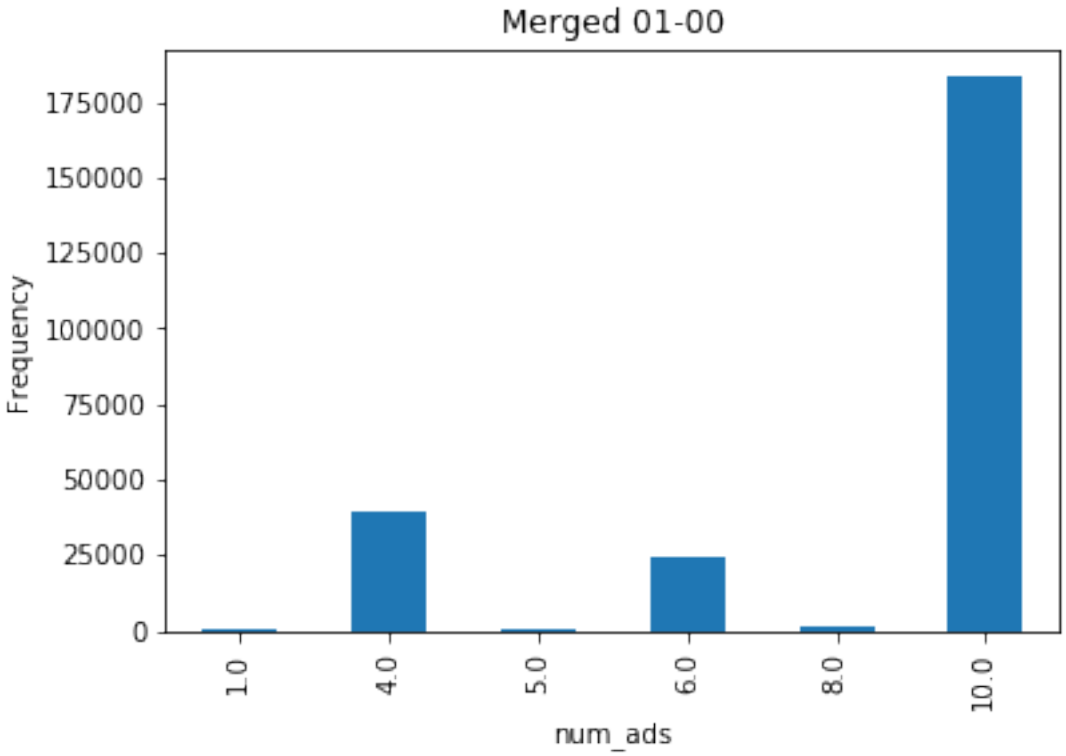
```
In [188]: import parse
          def graph_time(table, title):
              # Replace values in timestamp columns
              # We are using r_timestamp which is when an ad is requested
              # An alternative is i_timestamp which is when the impression is made,
              # but there are many more NaNs because only the winning ad is displayed/impressi
              FORMAT = "{0}T{1}:{2}:{3}Z"
              copy = table["r_timestamp"].copy()
              assert len(copy.shape) == 1, "Incorrect copy dimensions"
              for i in range(copy.size):
                  t = copy.iloc[i]
                  date, h, m, s = parse.parse(FORMAT, t)
                  # NOTE we assume everything is in one day
                  h, m, s = float(h), float(m), float(s)
                  inseconds = h*60*60 + m*60 + s
                  inhours = inseconds / 3600
```

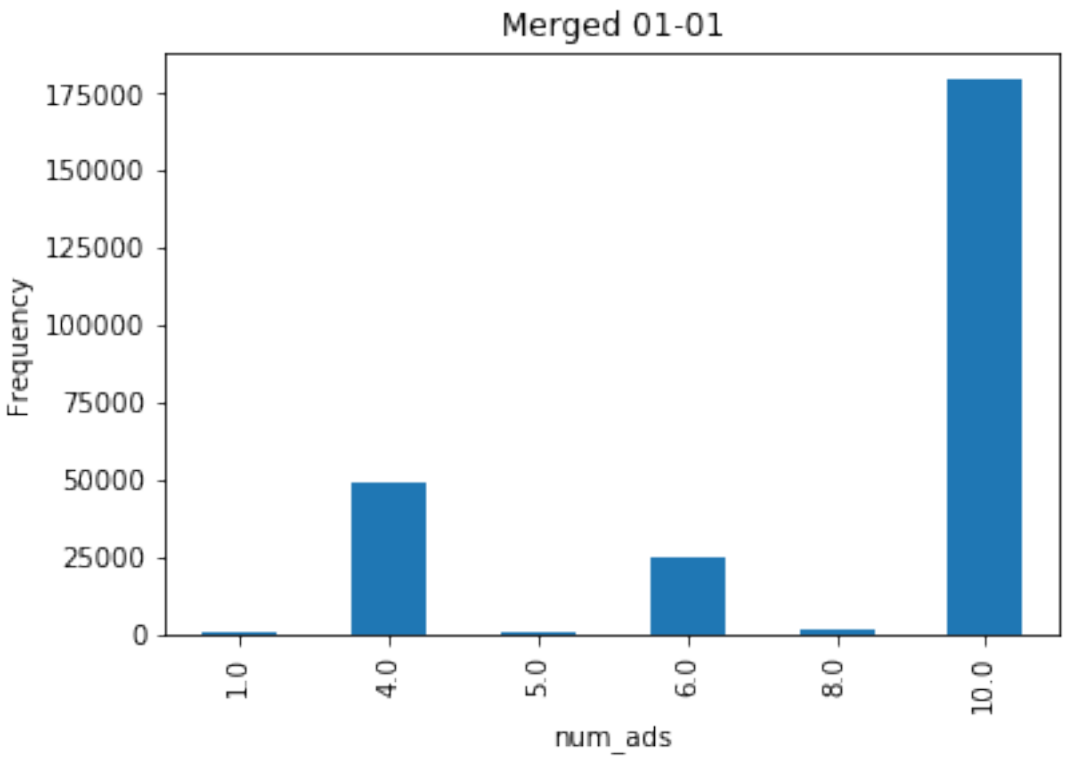
```
#         if inseconds > 3600:
#             print(t)
        copy.iloc[i] = inhours
    timeplot = copy.plot(kind="hist", bins=24, title=title)
    timeplot.set_xlabel("Hours")

# To save us a lot of time from converting the entire file to a DataFrame
def extract_times(filename):
    f = open(filename, 'r')
    FINDSTR = '"r_timestamp": '
    TIMESTAMP_FORMAT = '"YYYY-MM-DDTHH:MM:SS.SSSSSSZ"'
    count = 0
    rows_list = []
    for line in f:
        index = line.find(FINDSTR)
        if index == -1:
            print("Did not find timestamp at index", count)
        else:
            timestamp = line[index+len(FINDSTR)+1:index+len(FINDSTR)+len(TIMESTAMP_FO
        rows_list.append({"r_timestamp": timestamp})
        count += 1
    return pd.DataFrame(rows_list)

# TODO uncomment following line when kernel is restarted; avoiding rerunning for now
########## df = extract_times("data/merged-01-xx") ##########
graph_time(df.sample(frac=0.1), "Day01")
```

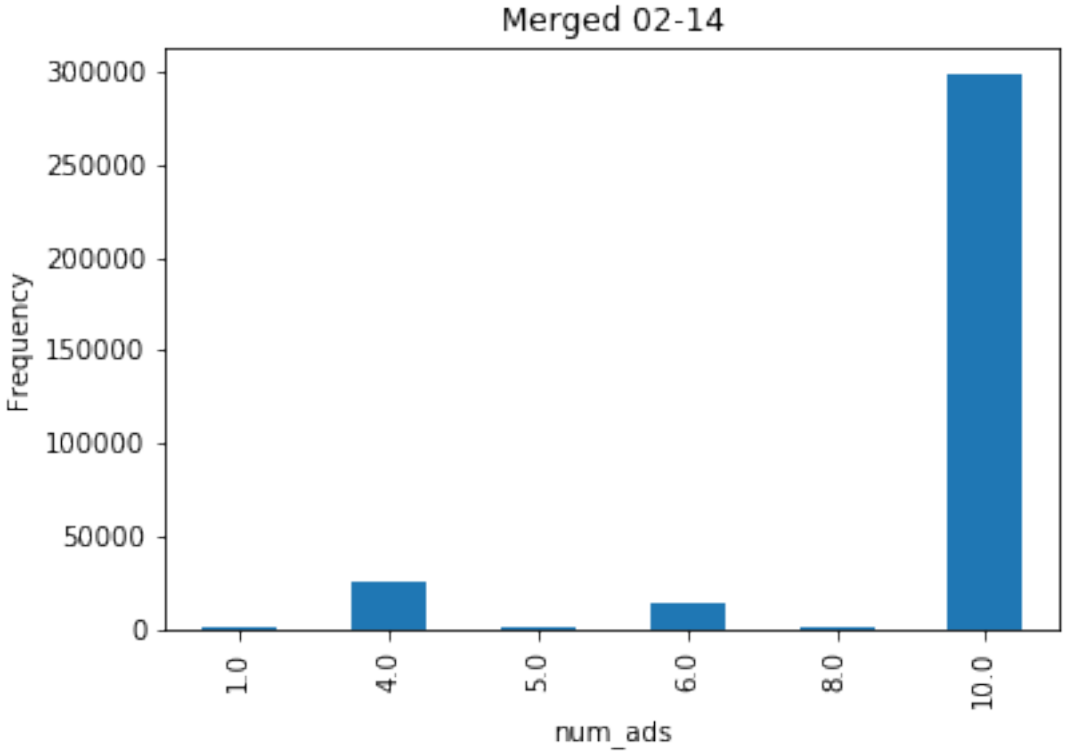


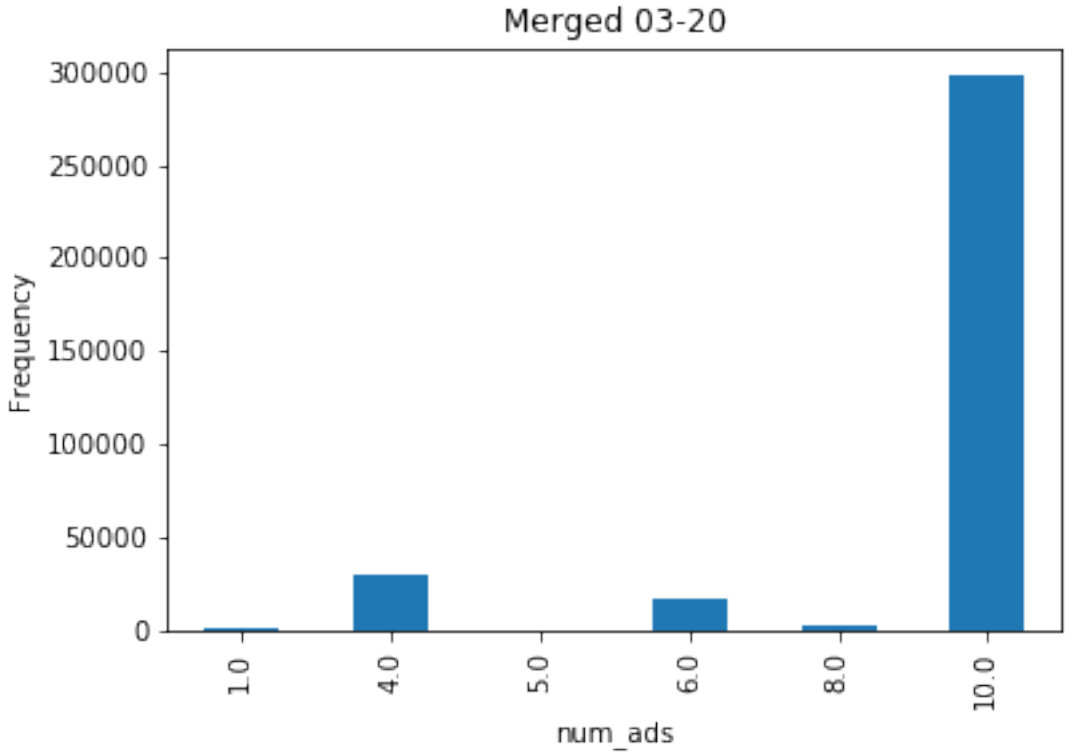
0.0.3 Number of Ads Displayed

```
In [219]: def graph_numads(table, title):
              numads = table[["num_ads"]].dropna()
              grouped = numads.groupby("num_ads").size()
              graph = grouped.plot.bar()
              graph.set_ylabel("Frequency")
              graph.set_title(title)
              plt.show()

          graph_numads(table00, "Merged 01-00")
          graph_numads(table01, "Merged 01-01")
          graph_numads(table14, "Merged 02-14")
          graph_numads(table20, "Merged 03-20")
```




Merged 01-01
Frequency
175000
150000
125000
100000
75000
50000
25000
0
1.0
4.0
5.0
6.0
8.0
10.0
num_ads


Merged 02-14
Frequency
300000
250000
200000
150000
100000
50000
0
1.0
4.0
5.0
6.0
8.0
10.0
num_ads



In [ ]:

0.0.4 IGNORE BELOW

In [42]: small.columns

Out[42]: Index(['_host', 'ad_network_id', 'ad_type', 'adlog_count', 'advertiser_id',
'bid_requests', 'bid_responses', 'c_cnt', 'c_flag_cnt', 'c_timestamp',
'campaign_id', 'campaign_type', 'ck', 'cr_cnt', 'creative_id',
'exp_mode', 'f_cnt', 'flag', 'geo_area_code', 'geo_city_code',
'geo_city_name', 'geo_continent_code', 'geo_country_code2',
'geo_country_code3', 'geo_dma_code', 'geo_postal_code',
'geo_region_name', 'geo_timezone', 'i_cnt', 'i_flag_cnt', 'i_timestamp',
'i_txn_fee', 'i_txn_rate', 'ip_address', 'is_bot', 'is_fraud',
'keywords', 'num_ads', 'pub_campaign_id', 'pub_network_id', 'r_cnt',
'r_num_ads_requested', 'r_num_ads_returned', 'r_num_ads_third_party',
'r_timestamp', 'rate_metric', 'referer', 'session_id', 'site_id',
'token', 'txn_fee', 'txn_rate', 'ua', 'ua_device', 'ua_device_type',
'ua_major', 'ua_minor', 'ua_name', 'ua_os', 'ua_os_name', 'url',
'user_agent', 'uuid', 'vi_cnt', 'vi_flag_cnt', 'vi_timestamp', 'vv_cnt',

```
       'widget_id', 'zone_id'],
      dtype='object')
```

In [ ]:
```
# EDIT THIS as needed
# NOTE somehow (in 01-00, 01-01, 02-14, 03-20 at least) there was not a single filled
useless_all = []
# only one value
useless_all.extend(['adlog_count', 'cr_cnt', 'r_num_ads_third_party'])
# no values
useless_all.extend(['bid_requests', 'bid_responses'])

useless_ads = []
useless_clicks = []
filtered = small.drop(useless)
```

In [ ]:
```

```