To avoid Robocallers, use Pandas

Justus Perlwitz

2015-11-15

Contents

I found a really neat data source online on unwanted robocalls that the FCC (Federal Communications Commission, a United States government agency) has created and published openly. The data source provides times and dates of unwanted robocalls that consumers have reported to the FCC. We can use this data source to find out all kinds of things, but today we will be content with just finding out the time of the day households are most likely to receive robocalls.

First, we need to fire up our trusty IPython notebook and download the data. The data is freely available on the fcc.gov website and is encoded as a `.csv` (comma separated values) file. Fortunately for us, we don't even need to leave IPython. We can execute bash commands right here!

```
%%bash
wget https://consumercomplaints.fcc.gov/hc/\
theme_assets/513073/200051444/Telemarketing_RoboCall_Weekly_Data.csv
```

0.1 Reading the Data

Using the convenient `read_csv` method, we can automatically turn a `.csv` file into a pandas DataFrame.

```
from pandas import read_csv
s = read_csv('Telemarketing_RoboCall_Weekly_Data.csv')
```

Excellent! As we can see, the DataFrame contains a lot of useful information. There is the number of a caller, the type of call, the reason for reporting it, in which state of the US it happened and finally time and date.

0.2 Extracting Information

Now, in order to retrieve the time of day when the unwanted robocall was received, we need to take a closer look at the `Time of Issue` column.

```
s['Time of Issue']
```

Time of Issue
0
-
1
8:20 AM
2
2:00 p.m.
3
11:46 AM
4
3:19 p.m.
5
12:00 PM

6
11:00 AM
7
10:21 AM
8
1:02 PM
9
12:58 p.m.
10
11:11 AM
11
1:30 PM
12
10:35 AM
13
6:50 a.m.
14
11:59 AM
15
10:49 a.m.
16
10:51 a.m.
17
8:41 PM
18
11:21 a.m.
19
4:24 PM
20
11:30 AM
21
-
22
12:00 P.M.
23
1:10 PM
24
2:00 PM
25
1:45 PM
26
11:25 AM
27
2:58 p.m.
28
2:15 PM
29
9:43 AM
...
...
16284
9:17 AM
16285
7:07 P.M.
16286
10:30 AM

16287
1:19 PM
16288
2:17 P.M.
16289
8:30 PM
16290
11:27 a.m.
16291
3:00 PM
16292
10:12 PM
16293
10:19 AM
16294
1:31 p.m.
16295
1:00 PM
16296
2:55 PM
16297
6:16 PM
16298
7:00 PM
16299
6:02 p.m.
16300
5:13 PM
16301
11:16 AM
16302
4:47 P.M.
16303
8:31 p.m.
16304
6:00 AM
16305
1:11 p.m.
16306
5:39 p.m.
16307
1:00 PM
16308
3:40 p.m.
16309
3:13 p.m.
16310
3:32 p.m.
16311
5:45 a.m.
16312
3:54 PM
16313
3:59 p.m.
16314 rows × 1 columns

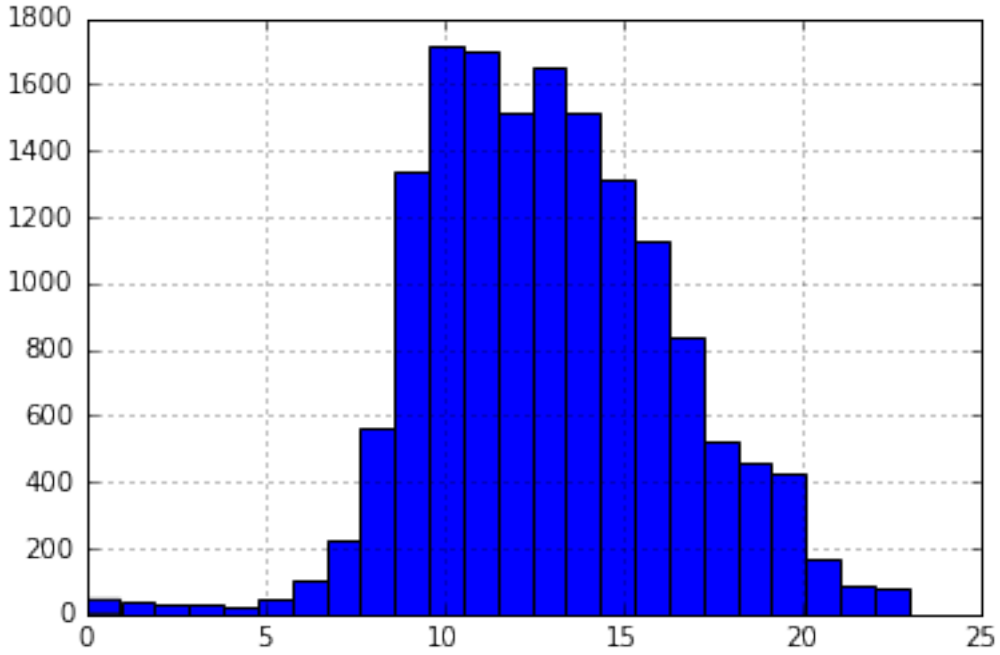

Figure 1: Matplotlib Chart

The column is not well formatted, as 'AM' and 'PM' appear in different spellings. Sometimes irregular characters such as "?" and "" appear. Perhaps this is OCR gone awry. In order to extract useful information, we need to parse the time information first and ignore a lot of the spelling/encoding errors.

```
from datetime import datetime, time

def parse_time(raw_time):
    if raw_time == '-':
        return None
    # This is why we can't have nice things
    raw_time = raw_time.upper().replace(".", "").replace(",", "").replace(
        ":", "").replace(">", "").replace("?", "").replace(
        "  ", " ").replace("MM", "M").zfill(7)
    if raw_time[:2] == "00":
        raw_time = raw_time[1:]
    # Need to use datetime, time has no "strptime"
    dt =  datetime.strptime(
        raw_time,
        "%I%M %p",
    )
    return dt.hour # All this sweat and labor for an hour! Oh heavens!

actual_times = s['Time of Issue'].apply(parse_time)
```

Wasn't that fun? Dealing with messy data is a challenge, but can be mastered through continuous application of brute-force and being in denial about reality. A steady supply of coffee helps as well.

Now that we have the actual times, I would like to retrieve the most frequent hour. Let's do this!

```
%matplotlib inline
actual_times.hist(bins=24)
```

Aha! Most robo calls get placed around 10-11 AM!