Student Information

Please provide information about yourself. We will NOT grade this submissing w/o all the information

Name:
NetID:
Recitation (01,02,90,91):
Notes to Grader (optional):

IMPORTANT Your work will not be graded withour your initials below
I certify that this lab represents my own work and I have read the RU academic intergrity policies at
https://www.cs.rutgers.edu/academic-integrity/introduction (https://www.cs.rutgers.edu/academic-integrity/introduction)
Initials:

Grader Notes

Your Grade:
Grader Initials:
Grader Comments (optional):

CS 439 - Introduction to Data Science

Fall 2021

Lab 3: Data Cleaning and Visualization

Due Date: Sunday October 10, 2021 by 11:59 PM

Instructions

This lab is presented as a notebook. Please execute the cells that are already completed and your task is to fill in the code between ### BEGIN SOLUTION ### and ### END SOLUTION ###.

Important: Please do not add any new cells or change the order of cells. If you have questions, please contact the courseS staff.

In this lab, you will be working with a dataset from NYPD containing data on calls to the New York Police Department. Information about the datasets can be found https://opendata.cityofnewyork.us/ (https://opendata.cityofnewyork.us/)

Setup

Note that after activating matplotlib to display figures inline via the IPython magic `%matplotlib inline`, we configure a custom default figure size. Virtually every default aspect of matplotlib can be customized (https://matplotlib.org/users/customizing.html).

In [ ]:
```
!pip install datascience
```

In [7]:
```
import pandas as pd
import numpy as np
import zipfile
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
plt.rcParams['figure.figsize'] = (12, 9)
```

Part 1:Getting Data

We will work with the NYPD Historic complaint data set. Our first task is to estimate the size of this download by looking at the number of rows, columns and using an estimated size for a column (use a reasonable value). The site metadata is available from the page https://data.cityofnewyork.us/Public-Safety/NYPD-Complaint-Data-Historic/qgea-i56i (https://data.cityofnewyork.us/Public-Safety/NYPD-Complaint-Data-Historic/qgea-i56i)

In [ ]:
```
### BEGIN SOLUTION
estimated size of the download (based on metadata information from the site)
# please explain how you reached the answer
### END SOLUTION
```

1.1 Download the data

This file is large (use the estimate you did above). If it takes too long to download, you may want to interrupt and download the file using a browser and URL https://data.cityofnewyork.us/Public-Safety/NYPD-Complaint-Data-Historic/qgea-i56i (https://data.cityofnewyork.us/Public-Safety/NYPD-Complaint-Data-Historic/qgea-i56i)

In [ ]:
```
import utils
data_dir = 'data'
data_url = 'https://data.cityofnewyork.us/Public-Safety/NYPD-Complaint-Data-

file_name = 'NYPD_Complaint_Data_Historic.csv'

# To retrieve the dataset, we will use the `utils.fetch_and_cache` utility f
dest_path = utils.fetch_and_cache(data_url=data_url, file=file_name,data_dir
print(f'Located at {dest_path}')
```

1.2 Inspect the size of the file

It is helpful to get an idea of the size of the file. This can be done using functions in the utils library.

In [5]:
```
# look at the size of the file w/o opening it using OS (https://docs.python.
# variety of operating system related functions from this package.
### BEGIN SOLUTION

#### END SOLUTION
```

```
2352699423
```

1.3 Split the large file

This data file NYPD_Complaint_Data_Historic.csv is too big to load into a single DataFrame. Let us split the large file into smaller files. Let us find out the number of lines in the NYPD_Complaint_Data_Historic.csv file using utils.

In [2]:
```
import utils
```

In [4]:
```
# Using utils library, find the number of lines in the file
# import pysys
### BEGIN SOLUTION


### END SOLUTION
linecount
```

Out[4]:
```
7375994
```

In [8]:
```
# Split the file into 10 smaller files. Estimate the number of lines in each
# files should be created in the data folder and named NYPD_Complaint_Data_H
# NYPD_Complaint_Data_Historic_2.csv, ... NYPD_Complaint_Data_Historic_10.cs
# It is possible that few lines from the original file may not be saved due

### BEGIN SOLUTION



### END SOLUTION
```

```
737600

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-8-b99778a0d1bc> in <module>
     19 i=0
     20 for chunk in pd.read_csv(file, chunksize=chunksize):
---> 21     chunk.to_csv(f'data/NYPD_Complaint_Data_Historic_{i+1}.csv')
     22     print(utils.line_count(f'data/NYPD_Complaint_Data_Historic_{i+1}.cs
v'))
     23     i+=1

~\anaconda3.1\lib\site-packages\pandas\core\generic.py in to_csv(self, path_or_
buf, sep, na_rep, float_format, columns, header, index, index_label, mode, enco
ding, compression, quoting, quotechar, line_terminator, chunksize, date_format,
doublequote, escapechar, decimal, errors, storage_options)
   3385         )
   3386
-> 3387         return DataFrameRenderer(formatter).to_csv(
   3388             path_or_buf,
   3389             line_terminator=line_terminator,

~\anaconda3.1\lib\site-packages\pandas\io\formats\format.py in to_csv(self, pat
h_or_buf, encoding, sep, columns, index_label, mode, compression, quoting, quot
echar, line_terminator, chunksize, date_format, doublequote, escapechar, error
s, storage_options)
   1081             formatter=self.fmt,
   1082         )
-> 1083         csv_formatter.save()
   1084
   1085         if created_buffer:

~\anaconda3.1\lib\site-packages\pandas\io\formats\csvs.py in save(self)
    246             )
    247
--> 248             self._save()
    249
    250     def _save(self) -> None:

~\anaconda3.1\lib\site-packages\pandas\io\formats\csvs.py in _save(self)
    251         if self._need_to_save_header:
    252             self._save_header()
--> 253         self._save_body()
    254
    255     def _save_header(self) -> None:
```

```
~\anaconda3.1\lib\site-packages\pandas\io\formats\csvs.py in _save_body(self)
    289             if start_i >= end_i:
    290                 break
--> 291             self._save_chunk(start_i, end_i)
    292
    293     def _save_chunk(self, start_i: int, end_i: int) -> None:

~\anaconda3.1\lib\site-packages\pandas\io\formats\csvs.py in _save_chunk(self,
 start_i, end_i)
    296         df = self.obj.iloc[slicer]
    297
--> 298         res = df._mgr.to_native_types(**self._number_format)
    299         data = [res.iget_values(i) for i in range(len(res.items))]
    300

~\anaconda3.1\lib\site-packages\pandas\core\internals\managers.py in to_native_
types(self, **kwargs)
    677         in formatting (repr / csv).
    678         """
--> 679         return self.apply("to_native_types", **kwargs)
    680
    681     def is_consolidated(self) -> bool:

~\anaconda3.1\lib\site-packages\pandas\core\internals\managers.py in apply(sel
f, f, align_keys, ignore_failures, **kwargs)
    425                     applied = b.apply(f, **kwargs)
    426                 else:
--> 427                     applied = getattr(b, f)(**kwargs)
    428             except (TypeError, NotImplementedError):
    429                 if not ignore_failures:

~\anaconda3.1\lib\site-packages\pandas\core\internals\blocks.py in to_native_ty
pes(self, na_rep, float_format, decimal, quoting, **kwargs)
   2108                 values = np.array(values, dtype="object")
   2109
-> 2110             values[mask] = na_rep
   2111             return self.make_block(values)
   2112

KeyboardInterrupt:
```

1.4 Check the files in the data folder

Now, we'll use a method of the `Pathlib.Path` class called `glob` to list all files in the `data` directory. You will find useful information in pathlib docs (https://docs.python.org/3/library/pathlib.html).

Below, we use pathlib's `glob` method to store the list of all files' names from the `data_dir` directory in the variable `file_names`. These names should be strings that contain only the file name (e.g. `dummy.txt` not `data/dummy.txt`). The asterisk (*) character is used with the `glob` method to match any string.

In [9]:
```
from pathlib import Path
data_dir_path = Path('data') # creates a Path object that points to the data
file_names = [x.name for x in data_dir_path.glob('*') if x.is_file()]
file_names
```

Out[9]:
```
['ben_kurtovic.py',
 'Berkeley_PD_-_Calls_for_Service.csv',
 'do_not_readme.md',
 'dummy.txt',
 'hello_world.py',
 'lab03_data_fa18.zip',
 'NYPD_Complaint_Data_Historic.csv',
 'NYPD_Complaint_Data_Historic_1.csv',
 'NYPD_Complaint_Data_Historic_2.csv']
```

1.5 Pre-processing of data

It is good to pre-process the data to see if the file can be opened in a Jupyter notebook. We need to avoid large files that can crash notebooks. Typically, files of size around 200 MB is ok to open into a DataFrame. In the following activities we will inspect the file w/o opening it as a DataFrame. Using utils.head

In [10]:
```
# Use the util.head() to read up to 5 lines from the original file (w/o open
from utils import head
head('data/NYPD_Complaint_Data_Historic.csv')
```

Out[10]:
```
['CMPLNT_NUM,CMPLNT_FR_DT,CMPLNT_FR_TM,CMPLNT_TO_DT,CMPLNT_TO_TM,ADDR_PCT_CD,RP
T_DT,KY_CD,OFNS_DESC,PD_CD,PD_DESC,CRM_ATPT_CPTD_CD,LAW_CAT_CD,BORO_NM,LOC_OF_O
CCUR_DESC,PREM_TYP_DESC,JURIS_DESC,JURISDICTION_CODE,PARKS_NM,HADEVELOPT,HOUSIN
G_PSA,X_COORD_CD,Y_COORD_CD,SUSP_AGE_GROUP,SUSP_RACE,SUSP_SEX,TRANSIT_DISTRICT,
Latitude,Longitude,Lat_Lon,PATROL_BORO,STATION_NAME,VIC_AGE_GROUP,VIC_RACE,VIC_
SEX\n',
 '394506329,12/31/2019,17:30:00,,,32,12/31/2019,118,DANGEROUS WEAPONS,793,WEAPO
NS POSSESSION 3,COMPLETED,FELONY,MANHATTAN,,STREET,N.Y. POLICE DEPT,0,,,,99993
7,238365,,,,,40.82092679700002,-73.94332421899996,"(40.82092679700002, -73.9433
2421899996)",PATROL BORO MAN NORTH,,UNKNOWN,UNKNOWN,E\n',
 '968873685,12/29/2019,16:31:00,12/29/2019,16:54:00,47,12/29/2019,113,FORGERY,7
29,"FORGERY,ETC.,UNCLASSIFIED-FELO",COMPLETED,FELONY,BRONX,,STREET,N.Y. POLICE
DEPT,0,,,,1022508,261990,,,,,40.885701406000074,-73.86164032499995,"(40.8857014
06000074, -73.86164032499995)",PATROL BORO BRONX,,UNKNOWN,UNKNOWN,E\n',
 '509837549,12/15/2019,18:45:00,,,109,12/29/2019,578,HARRASSMENT 2,638,"HARASSM
ENT,SUBD 3,4,5",COMPLETED,VIOLATION,QUEENS,FRONT OF,STREET,N.Y. POLICE DEPT,
0,,,,1034178,209758,25-44,UNKNOWN,M,,40.74228115600005,-73.81982408,"(40.742281
15600005, -73.81982408)",PATROL BORO QUEENS NORTH,,25-44,WHITE HISPANIC,F\n',
 '352454313,12/28/2019,01:00:00,,,47,12/28/2019,126,MISCELLANEOUS PENAL LAW,11
7,RECKLESS ENDANGERMENT 1,COMPLETED,FELONY,BRONX,REAR OF,STREET,N.Y. POLICE DEP
T,0,,,,1026412,258211,18-24,BLACK,M,,40.87531145100007,-73.84754521099995,"(40.
87531145100007, -73.84754521099995)",PATROL BORO BRONX,,UNKNOWN,UNKNOWN,E\n']
```

1.6 Inspecting and describing data columns

There should be 35 columns in each record. Using header information and data types, describe the type of data in each column. If you are unable to determine, just state so.

BEGIN SOLUTION

- CMPLNT_NUM :
- CMPLNT_FR_DT:
- CMPLNT_FR_TM:
- CMPLNT_TO_DT:
- CMPLNT_TO:
- ....
- VIC_RACE:
- VIC_SEX:

END SOLUTION

Part 2 - Exploratory Data Analysis

Exploratory data analysis (EDA) is the process of examining a subest of a large data set to see what we can know about the data. First we will explore one file NYPD_Complaint_Data_Historic_1.csv to see what we can find out.

2.1 Loading Data into a DataFrame

Load the first CSV file, NYPD_Complaint_Data_Historic_1.csv into a `pandas.DataFrame` object. Also do a time analysis to see how long it took to load the data into a DataFrame. Time should be printed in seconds. The time libraries https://docs.python.org/3/library/time.html (https://docs.python.org/3/library/time.html) can help.

In [12]:
```
1 !pip install memory_profiler
```

```
Collecting memory_profiler
  Downloading memory_profiler-0.58.0.tar.gz (36 kB)
Requirement already satisfied: psutil in c:\users\andyg\anaconda3.1\lib\site-pa
ckages (from memory_profiler) (5.8.0)
Building wheels for collected packages: memory-profiler
  Building wheel for memory-profiler (setup.py): started
  Building wheel for memory-profiler (setup.py): finished with status 'done'
  Created wheel for memory-profiler: filename=memory_profiler-0.58.0-py3-none-a
ny.whl size=30183 sha256=90ab55dae2a67889e3f6c12d9fabf260f679e7c462644eca74a8ed
cac28ae337
  Stored in directory: c:\users\andyg\appdata\local\pip\cache\wheels\6a\37\3e\d
9e8ebaf73956a3ebd2ee41869444dbd2a702d7142bcf93c42
Successfully built memory-profiler
Installing collected packages: memory-profiler
Successfully installed memory-profiler-0.58.0
```

In [13]:
```
### BEGIN SOLUTION


### END SOLUTION
```

```
peak memory: 859.64 MiB, increment: 440.06 MiB
5.297006607055664
```

2.2 Description of Fields

Let's also check some basic information about these files using the `DataFrame.describe` and `DataFrame.info` methods. Describe columns that can be removed based on the information.

In [14]:
```
### BEGIN SOLUTION


# What columns can be removed from the DataFrame? A reasonable rule of thumb
# 50% of the data then it should be removed


### END SOLUTION
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 737599 entries, 0 to 737598
Data columns (total 36 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   Unnamed: 0          737599 non-null  int64  
 1   CMPLNT_NUM          737599 non-null  int64  
 2   CMPLNT_FR_DT        737588 non-null  object 
 3   CMPLNT_FR_TM        737599 non-null  object 
 4   CMPLNT_TO_DT        622586 non-null  object 
 5   CMPLNT_TO_TM        622915 non-null  object 
 6   ADDR_PCT_CD         737255 non-null  float64
 7   RPT_DT              737599 non-null  object 
 8   KY_CD               737599 non-null  int64  
 9   OFNS_DESC           736944 non-null  object 
 10  PD_CD               736576 non-null  float64
 11  PD_DESC             736576 non-null  object 
 12  CRM_ATPT_CPTD_CD    737599 non-null  object 
 13  LAW_CAT_CD          737599 non-null  object 
 14  BORO_NM             736561 non-null  object 
 15  LOC_OF_OCCUR_DESC   596884 non-null  object 
 16  PREM_TYP_DESC       733903 non-null  object 
 17  JURIS_DESC          737599 non-null  object 
 18  JURISDICTION_CODE   736576 non-null  float64
 19  PARKS_NM            5623 non-null    object 
 20  HADEVELOPT          35342 non-null   object 
 21  HOUSING_PSA         54580 non-null   float64
 22  X_COORD_CD          735262 non-null  float64
 23  Y_COORD_CD          735262 non-null  float64
 24  SUSP_AGE_GROUP      458016 non-null  object 
 25  SUSP_RACE           509594 non-null  object 
 26  SUSP_SEX            509474 non-null  object 
 27  TRANSIT_DISTRICT    18509 non-null   float64
 28  Latitude            735262 non-null  float64
 29  Longitude           735262 non-null  float64
 30  Lat_Lon             735262 non-null  object 
 31  PATROL_BORO         736487 non-null  object 
 32  STATION_NAME        18509 non-null   object 
 33  VIC_AGE_GROUP       680318 non-null  object 
 34  VIC_RACE            737594 non-null  object 
 35  VIC_SEX             737594 non-null  object 
dtypes: float64(9), int64(3), object(24)
memory usage: 202.6+ MB
['PARKS_NM', 'HADEVELOPT', 'HOUSING_PSA', 'TRANSIT_DISTRICT', 'STATION_NAME']
```

2.2 Finding Uniques

Notice that the functions above reveal type information for the columns, as well as some basic statistics about the numerical columns found in the DataFrame. However, we still need more information about what each column represents. Let's explore the data further.

find the number of unique values in each DataFrame column and answer the questions below.

In [15]:
```
### BEGIN SOLUTION


# Questions
# 1. How many distinct locations where the complaints have come from? -
# 2. How many age groups are represented in the data set? -
# 3. How many boroughs are included in the data set? -
# 4. How many offense types are listed in this data set? -

### END SOLUTION
```

```
JURISDICTION_CODE        21
PARKS_NM                738
HADEVELOPT              262
HOUSING_PSA             395
X_COORD_CD            54536
Y_COORD_CD            57546
SUSP_AGE_GROUP           25
SUSP_RACE                 7
SUSP_SEX                  3
TRANSIT_DISTRICT         12
Latitude             130582
Longitude            128213
Lat_Lon              130595
PATROL_BORO               8
STATION_NAME            368
VIC_AGE_GROUP            44
VIC_RACE                  7
VIC_SEX                   4
dtype: int64
```

2.3 Offense by Boro

Using GroupBy operation, create a DataFrame that groups offenses by Boro. call the DataFrame calls_by_Boro_and_offense

In [17]:
```
### BEGIN SOLUTION


### END SOLUTION
```

```
2, 96, 99, 101, 102, 103, 104, 105, 106, 109, 110, 116, 119, 120, 123, 124, 1
26, 127, 128, 133, 134, 135, 143, 145, 151, 154, 159, 160, 164, 165, 167, 17
2, 173, 175, 176, 177, 178, 180, 181, 182, 183, 186, 187, 190, 191, 193, 194,
195, 198, 202, 204, 210, 212, 213, 217, 219, 225, 226, 231, 232, 235, 237, 23
9, 240, ...], 'BROOKLYN': [6, 16, 31, 36, 44, 52, 55, 79, 88, 107, 108, 114,
115, 129, 130, 137, 138, 140, 141, 146, 147, 153, 157, 168, 171, 174, 179, 18
4, 189, 196, 201, 205, 211, 214, 215, 218, 224, 241, 245, 246, 247, 256, 258,
262, 266, 272, 281, 286, 292, 295, 307, 308, 313, 314, 323, 325, 329, 338, 35
4, 359, 366, 372, 399, 400, 401, 403, 410, 421, 428, 430, 432, 435, 439, 443,
444, 450, 452, 455, 456, 457, 467, 469, 473, 476, 479, 481, 482, 489, 491, 49
2, 497, 503, 506, 522, 526, 536, 541, 542, 545, 547, ...], 'MANHATTAN': [0,
5, 9, 10, 11, 18, 23, 25, 27, 28, 29, 34, 40, 47, 54, 67, 71, 84, 85, 89, 10
0, 111, 113, 117, 136, 144, 150, 155, 156, 158, 170, 188, 192, 200, 206, 216,
221, 222, 227, 230, 236, 251, 259, 271, 282, 294, 300, 306, 340, 342, 344, 34
5, 353, 355, 358, 361, 362, 368, 369, 371, 377, 381, 387, 393, 397, 414, 424,
425, 426, 431, 433, 441, 448, 454, 459, 462, 465, 480, 494, 499, 504, 505, 52
1, 527, 530, 532, 533, 539, 546, 560, 571, 575, 581, 583, 588, 589, 590, 592,
597, 601, ...], 'QUEENS': [2, 7, 12, 21, 33, 38, 39, 42, 53, 59, 61, 63, 66,
73, 76, 93, 95, 98, 118, 131, 132, 142, 149, 152, 161, 166, 197, 203, 209, 22
0, 228, 250, 255, 257, 298, 301, 310, 315, 328, 370, 373, 388, 404, 408, 420,
```

2.4 Offenses in Bronx

In the cell below, find a list of strings corresponding to the possible values for `OFNS_DESC` when `BORO` is "BRONX". Create an expression that automatically extracts the names of the offenses.

In [18]:
```
### BEGIN SOLUTION

### END SOLUTION

# How many offenses were committed in Bronx during the analysis period?
print(len(calls_by_Boro_and_offense.get_group('BRONX')))
```

```
['FORGERY' 'MISCELLANEOUS PENAL LAW' 'OFF. AGNST PUB ORD SENSBLTY &'
 'FELONY ASSAULT' 'ASSAULT 3 & RELATED OFFENSES' 'PETIT LARCENY' 'RAPE'
 'GRAND LARCENY OF MOTOR VEHICLE' 'SEX CRIMES' 'ROBBERY'
 'CRIMINAL MISCHIEF & RELATED OF' 'BURGLARY' 'DANGEROUS DRUGS'
 'DANGEROUS WEAPONS' 'VEHICLE AND TRAFFIC LAWS' 'CRIMINAL TRESPASS'
 'HARRASSMENT 2' 'OFFENSES INVOLVING FRAUD'
 'OFFENSES AGAINST PUBLIC ADMINI' 'ARSON' 'GRAND LARCENY' 'THEFT-FRAUD'
 'FRAUDS' 'ADMINISTRATIVE CODE' 'INTOXICATED & IMPAIRED DRIVING'
 'ESCAPE 3' 'NYS LAWS-UNCLASSIFIED FELONY' 'UNAUTHORIZED USE OF A VEHICLE'
 'THEFT OF SERVICES' 'OFFENSES AGAINST THE PERSON'
 'OTHER OFFENSES RELATED TO THEF' 'ENDAN WELFARE INCOMP'
 'POSSESSION OF STOLEN PROPERTY' 'OTHER STATE LAWS (NON PENAL LA'
 'OFFENSES AGAINST PUBLIC SAFETY' nan 'LOITERING/GAMBLING (CARDS, DIC'
 'ALCOHOLIC BEVERAGE CONTROL LAW' 'KIDNAPPING & RELATED OFFENSES'
 "BURGLAR'S TOOLS" 'CHILD ABANDONMENT/NON SUPPORT' 'GAMBLING'
 'AGRICULTURE & MRKTS LAW-UNCLASSIFIED' 'FRAUDULENT ACCOSTING'
 'OFFENSES RELATED TO CHILDREN' 'PROSTITUTION & RELATED OFFENSES'
 'FELONY SEX CRIMES' 'PETIT LARCENY OF MOTOR VEHICLE' 'KIDNAPPING'
 'OFFENSES AGAINST MARRIAGE UNCL' 'DISORDERLY CONDUCT'
 'HOMICIDE-NEGLIGENT-VEHICLE' 'INTOXICATED/IMPAIRED DRIVING' 'JOSTLING'
 'HOMICIDE-NEGLIGENT,UNCLASSIFIE' 'NYS LAWS-UNCLASSIFIED VIOLATION'
 'ANTICIPATORY OFFENSES' 'DISRUPTION OF A RELIGIOUS SERV']
160808
```

2.5 Most Common Crimes in NYC

What are the five crime types of OFNS_DESC that have the most crime events in Bronx? You may need to use `value_counts` to find the answer. Save your results as a list of strings.

Hint: *The `keys` method of the Series class might be useful.*

In [19]:
```
### BEGIN SOLUTION

### END SOLUTION
```

```
Index(['HARRASSMENT 2', 'PETIT LARCENY', 'ASSAULT 3 & RELATED OFFENSES',
       'CRIMINAL MISCHIEF & RELATED OF', 'DANGEROUS DRUGS'],
      dtype='object')
```

2.6 Probability of a Crime in Bronx

What is the probability that a the crime "Arson" can happen in Bronx?

In [20]:
```
### BEGIN SOLUTION

### END SOLUTION
```

Out[20]: 0.002651512793860441

Part 3: Visualizing the Data

Pandas vs. Seaborn Plotting

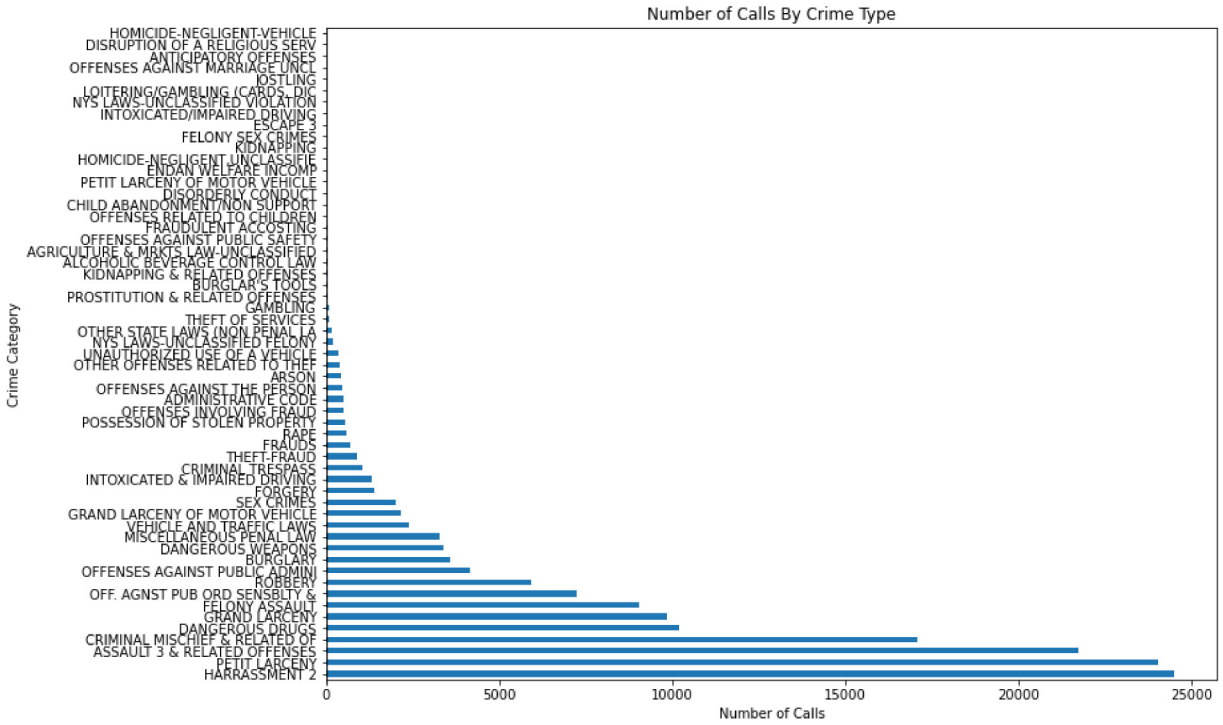
Pandas offers basic functionality for plotting. For example, the `DataFrame` and `Series` classes both have a `plot` method. However, the basic plots generated by pandas are not particularly pretty. While it's possible to manually use matplotlib commands to make pandas plots look better, we'll instead use a high level plotting library called Seaborn that will take care of most of this for us.

As you learn to do data visualization, you may find the pandas documentation (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html) and Seaborn documentation (https://seaborn.pydata.org/api.html) helpful!

3.1 Plotting a Series

Using the built-in plotting functionality of pandas, such as `plot` method of the `Series` class to generate a `barh` plot type, display the value counts for `OFNS_DESC` visually as a barh chart.

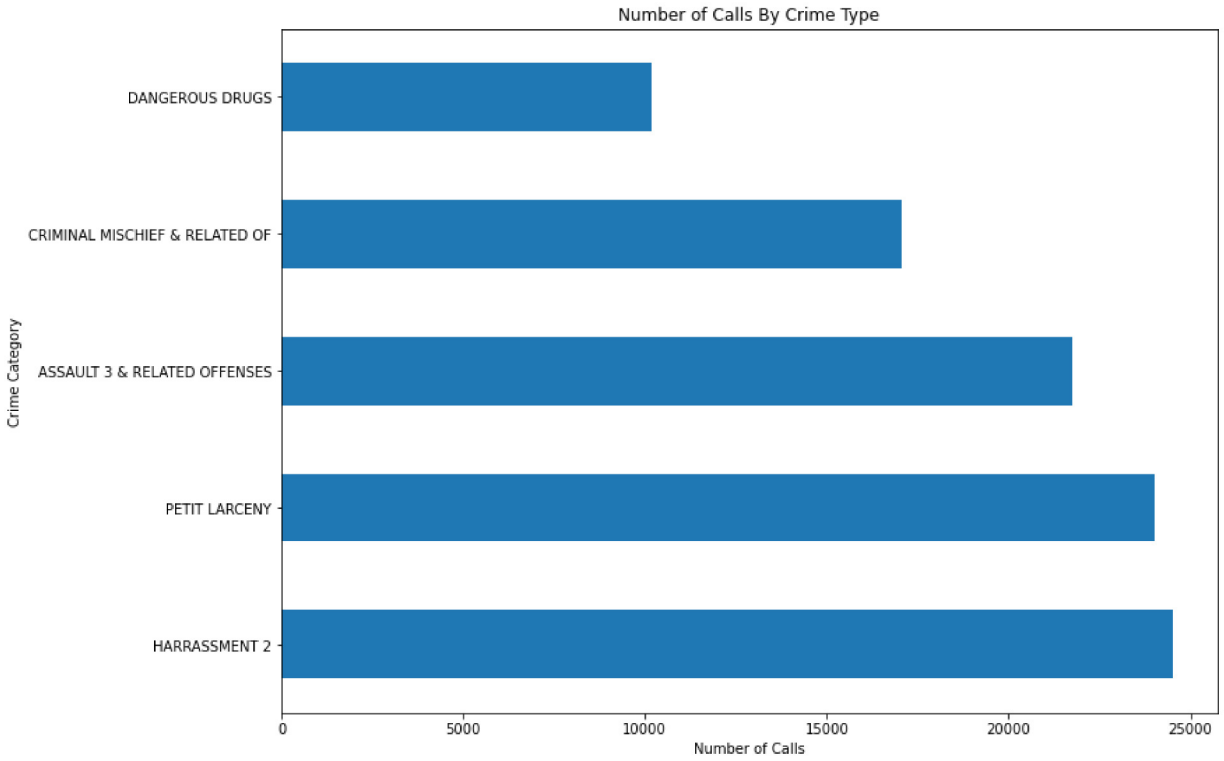
In [21]:
```
### BEGIN SOLUTION


### END SOLUTION
```



3.2 Getting a Better Plot

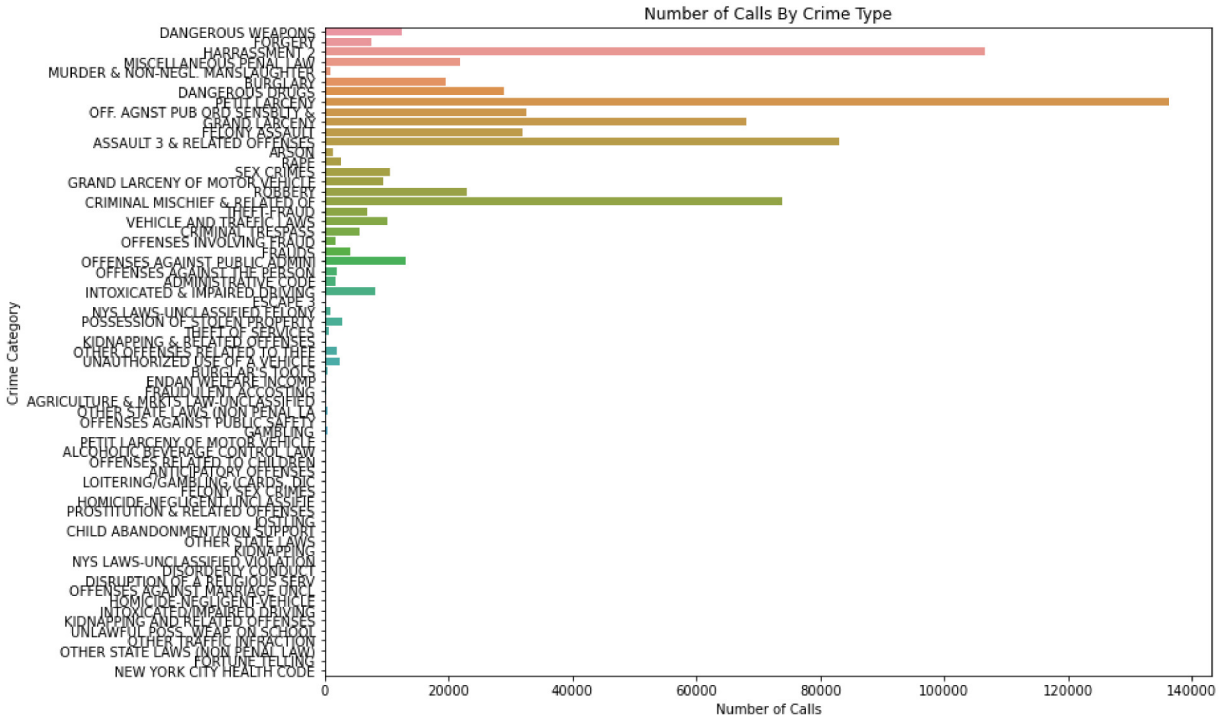
The plot above can be messy as it plots all offenses. Plot only the offenses that has more than 10000 calls

In [22]:
```
### BEGIN SOLUTION


### END SOLUTION
```



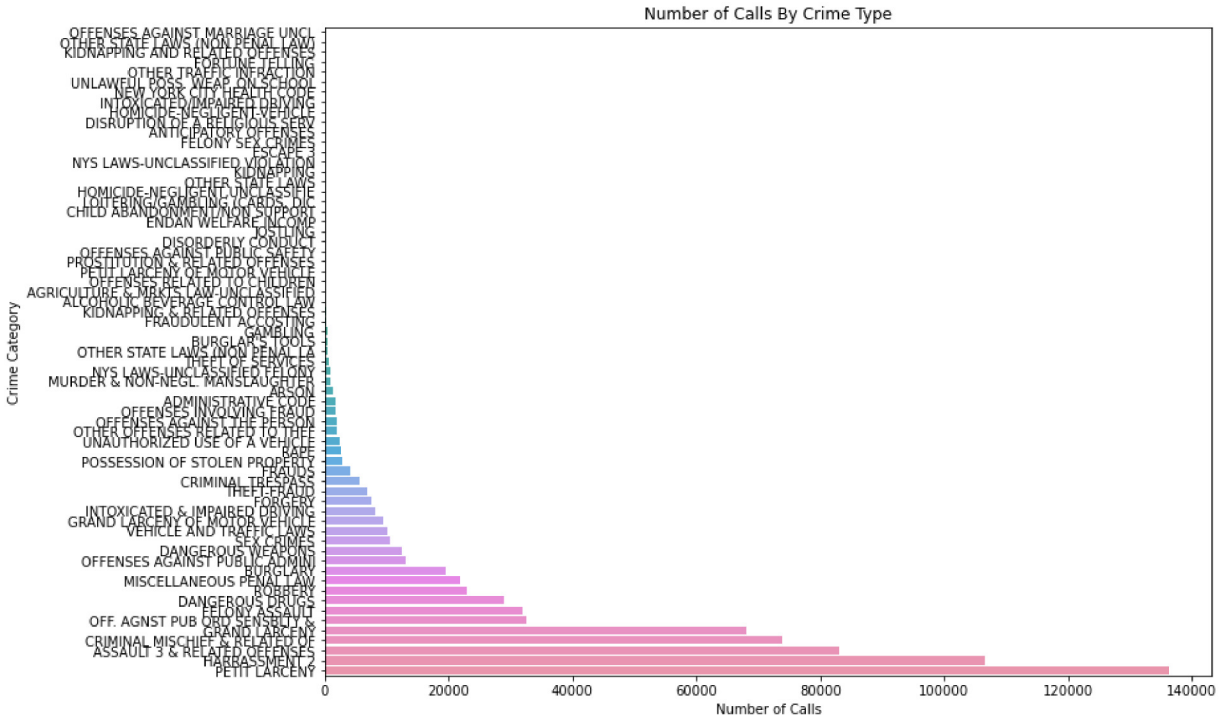
By contrast, the Seaborn library provides a specific function `countplot` built for plotting counts. It operates directly on the DataFrame itself i.e. there's no need to call `value_counts()` at all. This higher level approach makes it easier to work with. Use the y-label ("Crime Category"), x-label("Number of Calls") and title_of_plot("Number of Calls By Crime Type"

In [23]:
```
### BEGIN SOLUTION

### END SOLUTION
```



You may note that the ordering might be different for the seaborn plot (as compared to pandas plot). If we want the same ordering that we had in the pandas plot, we can use the order parameter of the `countplot` method. It takes a list of strings corresponding to the axis to be ordered. By passing the index of the `value_counts`, you can get the order you want.

In [24]:
```
### BEGIN SOLUTION

### END SOLUTION
```



Now we have a pretty bar plot with the bars ordered by size. Though `seaborn` appears to provide a superior plot from a aesthetic point of view, the `pandas` plotting library is also good to understand. You'll get practice using both libraries in the following questions.

An Additional Note on Plotting in Jupyter Notebooks

You may have noticed that many of our code cells involving plotting end with a semicolon (;). This prevents any extra output from the last line of the cell that we may not want to see. Try adding this to your own code in the following questions!

3.3 making more plots

Now it is your turn to make some more plots using `pandas` and `seaborn`. Let's start by looking at the distribution of calls over days of the week.

The CMPLNT_FR_DT field contains the date of the event. We would like to add a new column to the DataFrame that includes Day of the week (DAY_OF_WEEK) that indicates the day of the week. This can help us analyze the crimes on a specific day of the week. For example, we can answer questions such as "what day of the week that a LARSON is likely to happen in NYC?"

Add a new column `DAY_OF_WEEK` into the `calls` dataframe that has the day string (eg. 'Sunday') for the corresponding value in CMPLNT_FR_DT. For example, if the first 3 values of `CMPLNT_FR_DT` are `['01/27/2006, '01/28/2006, '01/29/2006]`, then the first 3 values of the `DAY_OF_WEEK` column should be `["Friday", "Saturday", "Sunday"]`.

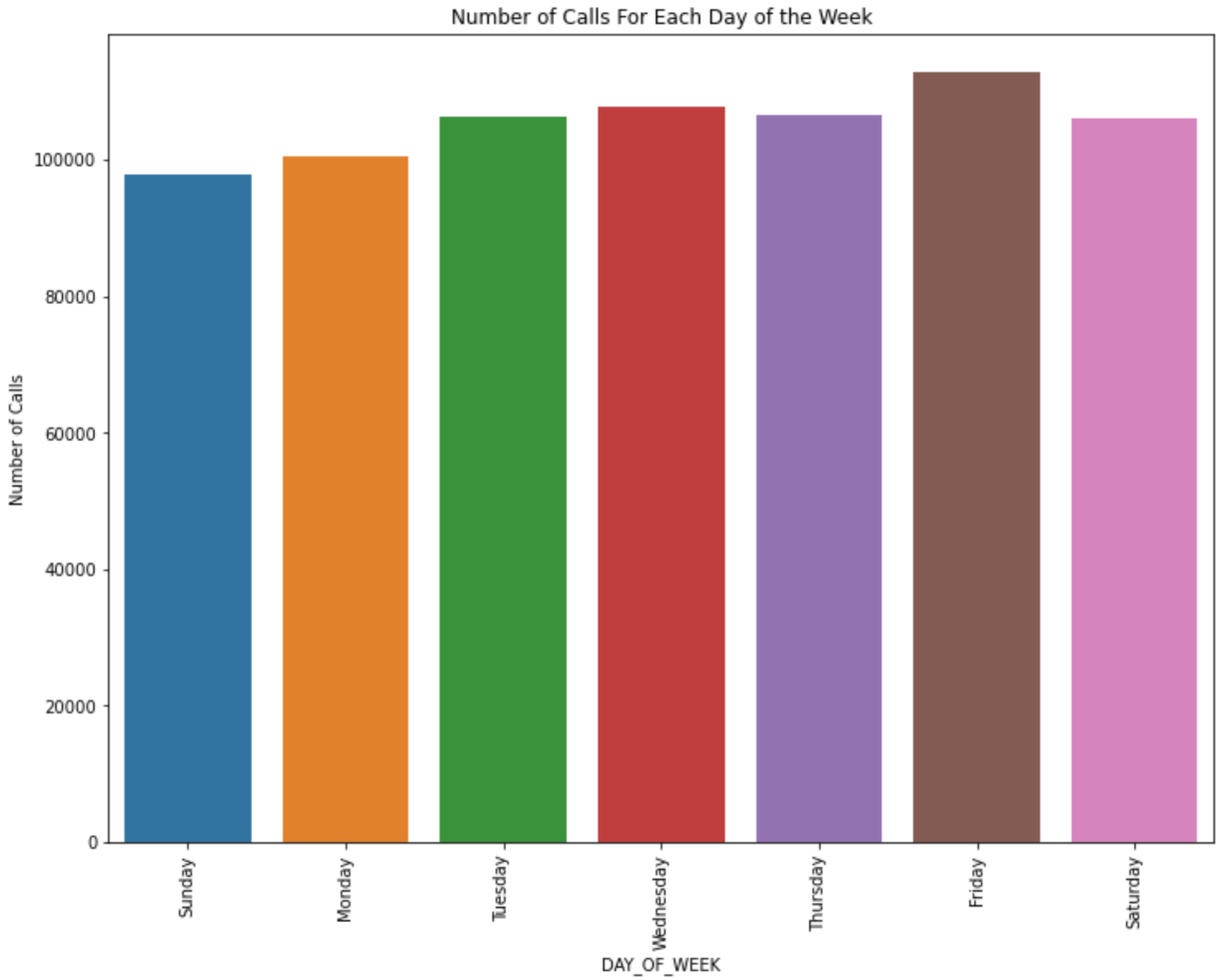
Hint: *Try using the Series.map (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.map.html) function on `calls["OFNS_DESC"]`. Can you assign this to the new column `calls["DAY_OF_WEEK"]`?*

In [25]:
```
days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "S
### BEGIN SOLUTION

### END SOLUTION
```

3.4 Seaborn plots

Create a `seaborn` plot that shows the number of calls for each day of the week. You may want to use of the `rotation` argument in `ax.set_xticklabels`, which rotates the labels by 90 degrees.

In [26]:
```
### BEGIN SOLUTION

### END SOLUTION
```

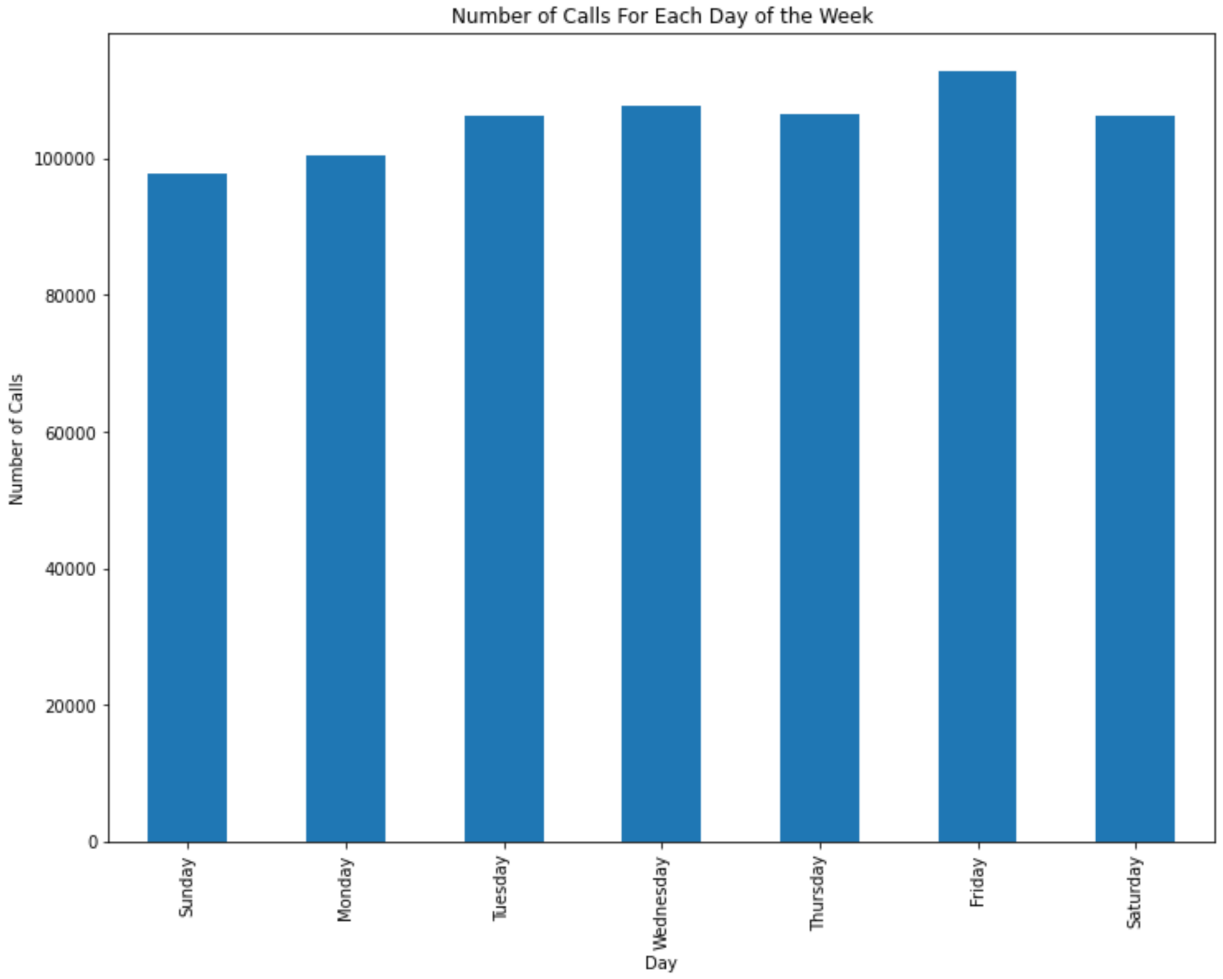


Now, let's make the same plot using `pandas`. Construct a vertical bar plot with the count of the number of calls (entries in the table) for each day of the week **ordered by the day of the week** (eg. `Sunday`, `Monday`, ...). Do not use `sns` for this plot. Be sure that your axes are labeled and

that your plot is titled.

Hint: *Given a series `s`, and an array `coolIndex` that has the same entries as in `s.index`, `s[coolIndex]` will return a copy of the series in the same order as `coolIndex`.*

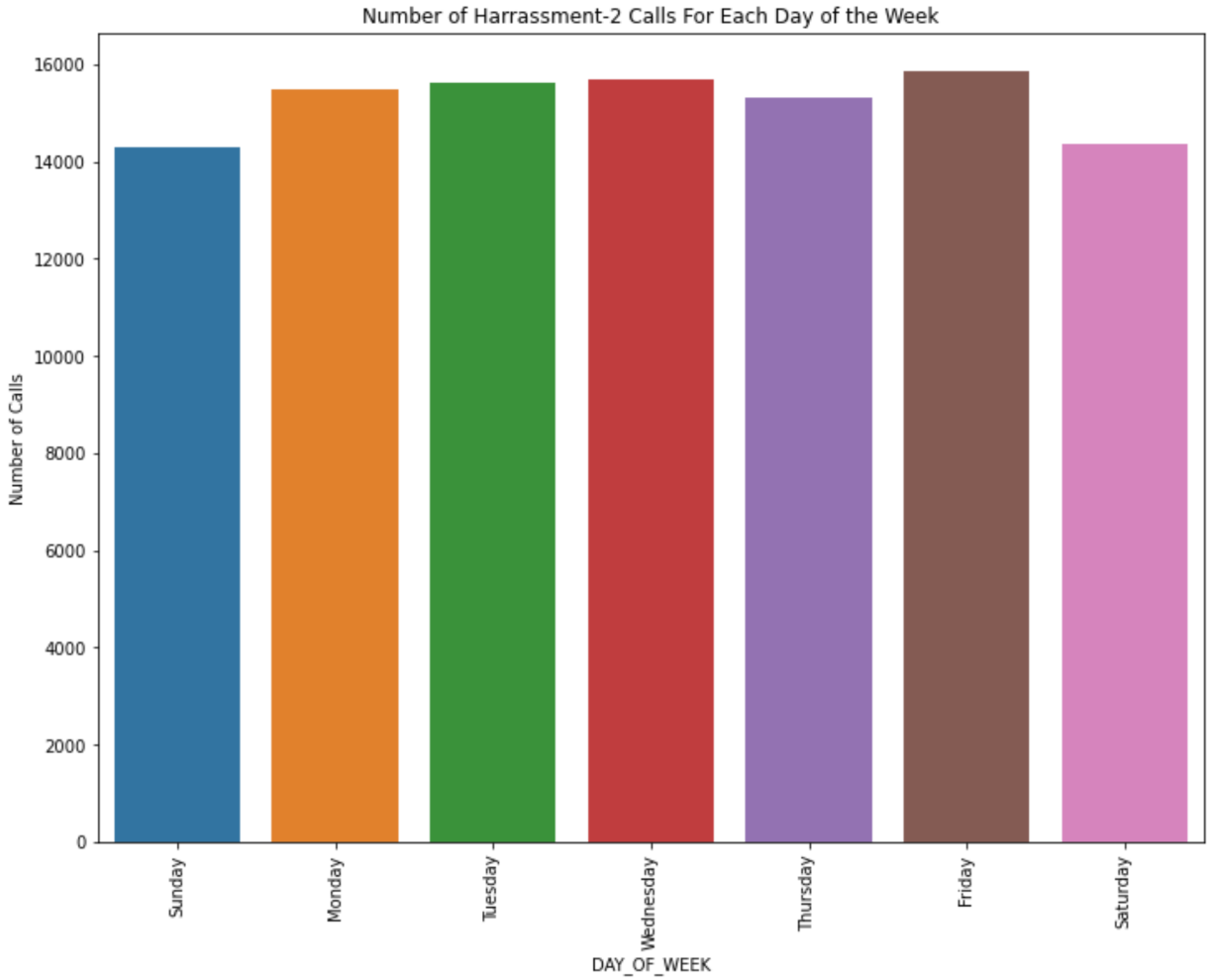
In [27]:
```
### BEGIN SOLUTION

### END SOLUTION
```




```
## 3.5 What Day of the Week is more calls?

Is it true that weekdays generally have slightly more calls than Saturday
or Sunday? What can you say about the difference?

##### BEGIN SOLUTION

##### END SOLUTION

We can break down into some particular types of events to see their
distribution. For example, let's make a bar plot for the OFNS_DESC
"HARRASSMENT 2". Which day is the peak for "HARRASSMENT 2"?

This time, use `seaborn` to create a vertical bar plot of the number of
total noise violations reported on each day of the week, again ordered by
the days of the week starting with Sunday. Do not use `pandas` to plot.
```

```
**Hint:** *If you're stuck, use the code for the seaborn plot in above question as a starting point.*
```

In [28]:
```
### BEGIN SOLUTION


### END SOLUTION
```

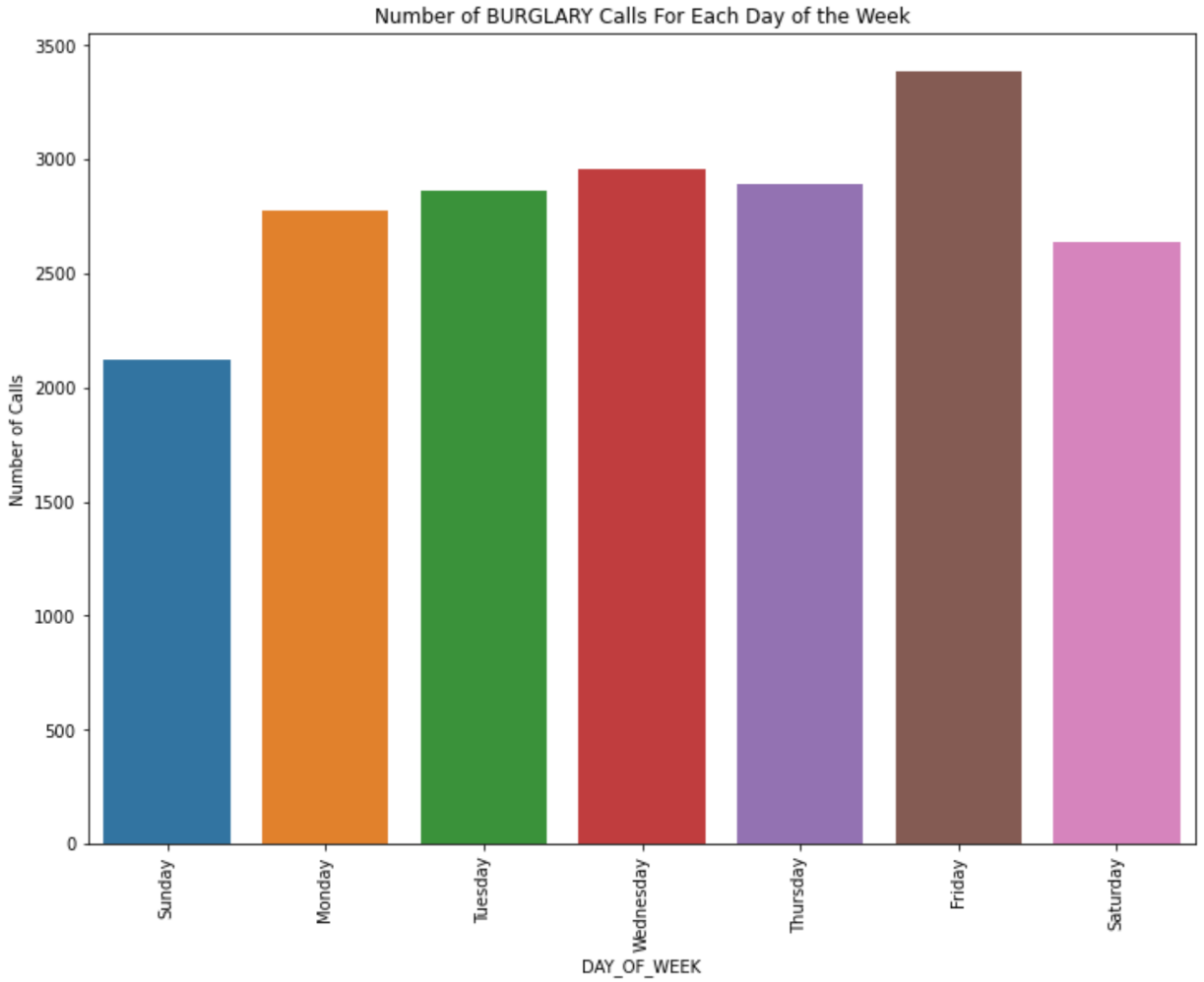



```
### 3.6

Do you see anything interesting about the distribution of HARRASSMENT 2 calls over a week? Type a short answer below.
##### BEGIN SOLUTION


##### END SOLUTION
```

3.7 More Plots

Let's look at a similar distribution but for a crime we have much more calls data about. In the cell below, create the same plot as you did in previous questions, but now looking at instances of the OFNS_DESC "BURGLARY" (instead of "HARRASSMENT 2"). Use either `pandas` or `seaborn` plotting as you desire.

In [29]:
```
### BEGIN SOLUTION


### END SOLUTION


```



3.8 time of events

Now let's look at the CMPLNT_TO_TM column which indicates the time for events. Since it contains hour and minute information, let's extract the hour info and create a new column named `Hour` in the `calls` dataframe. You should save the hour as an `int`. Then plot the frequency of each hour in the table (i.e., `value_counts()`) sorted by the hour of the day (i.e., `sort_index()`).

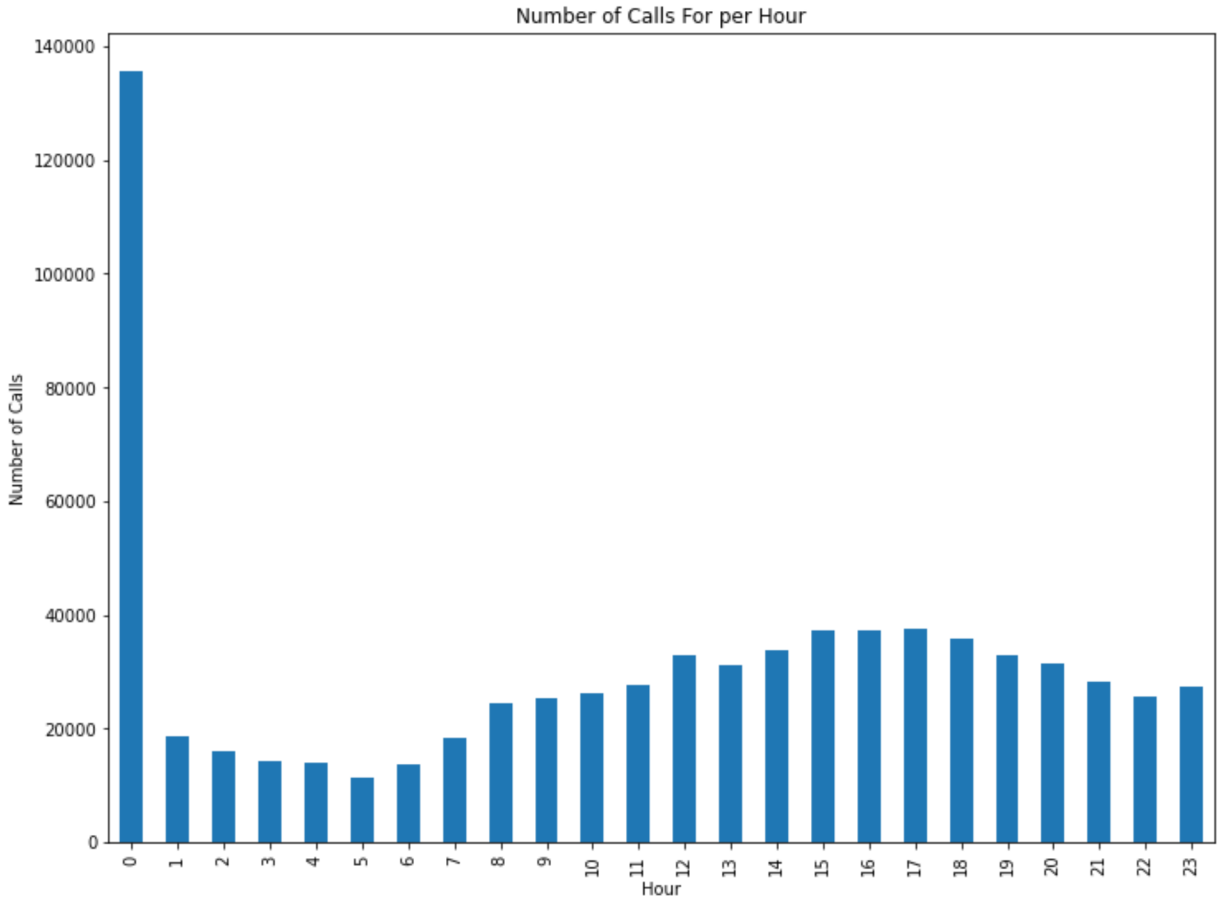
You will want to look into how to use:

- Series.str.slice (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.str.slice.html#pandas.Series.str.slice) to select the substring.
- Series.astype (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.astype.html) to change the type.

Hint: *The `str` helper member of a series can be used to grab substrings. For example, `calls["CMPLNT_TO_TM"].str.slice(3,5)` returns the minute of each hour of the `CMPLNT_TO_TM` .*

In [30]:
```
calls["Hour"] = calls["CMPLNT_TO_TM"].str.slice(0,2).replace(np.NaN,0).astyp
```

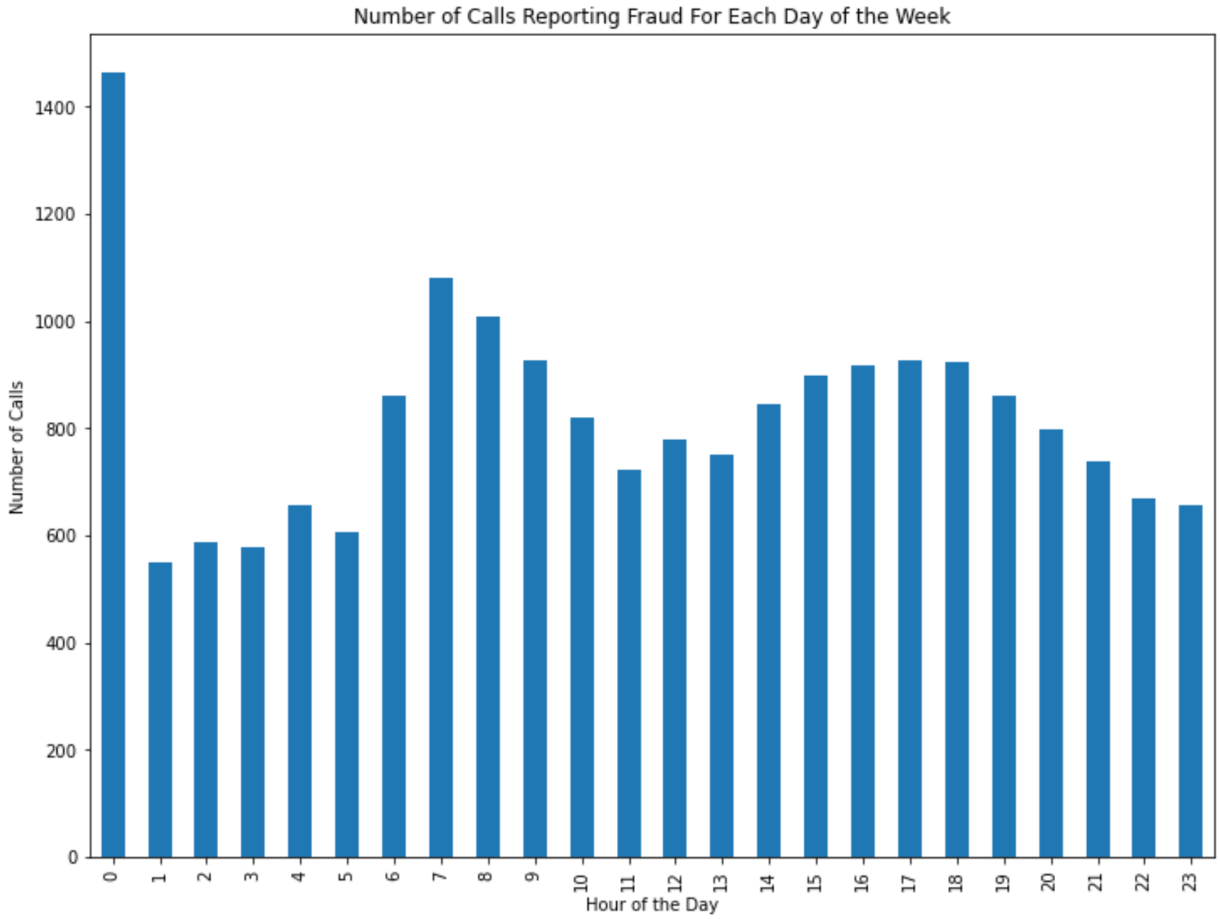
In [31]:
```
### BEGIN SOLUTION

### END SOLUTION
```



Create a pandas bar plot showing the number of BURGLARY crimes committed at each hour of the day. Use the labels

- ax.set_xlabel("Hour of the Day")
- ax.set_ylabel("Number of Calls")
- ax.set_title("Number of Calls Reporting Fraud For Each Day of the Week");

In [32]:
```
### BEGIN SOLUTION


### END SOLUTION
```

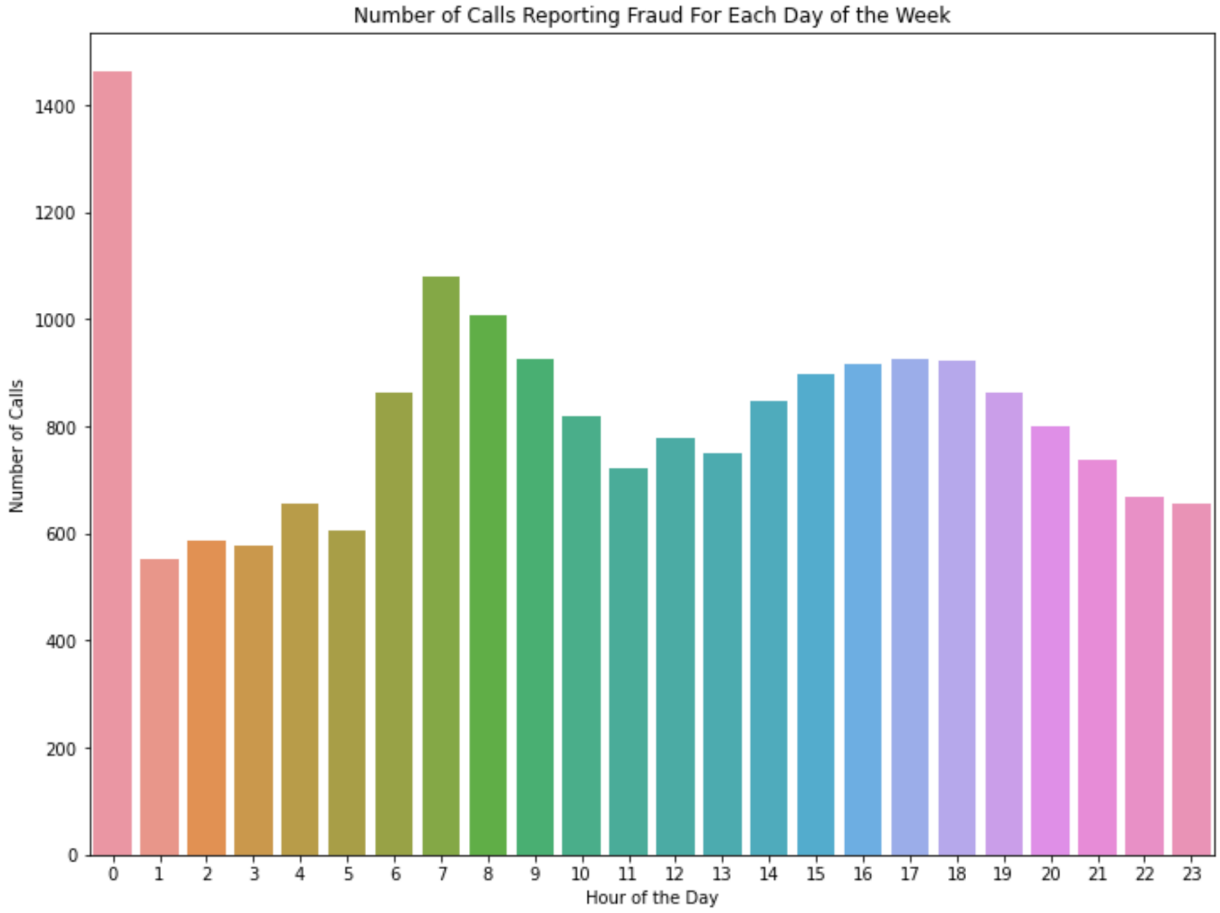


3.9 More plots

In the cell below, create a seaborn plot of the same data. Again, make sure you provide axes labels and a title for your plot.

In [33]:
```
### BEGIN SOLUTION

### END SOLUTION
```

```
C:\Users\andyg\anaconda3.1\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variable as a keyword arg: x. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments with
out an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



3.8 Spike in burglary?

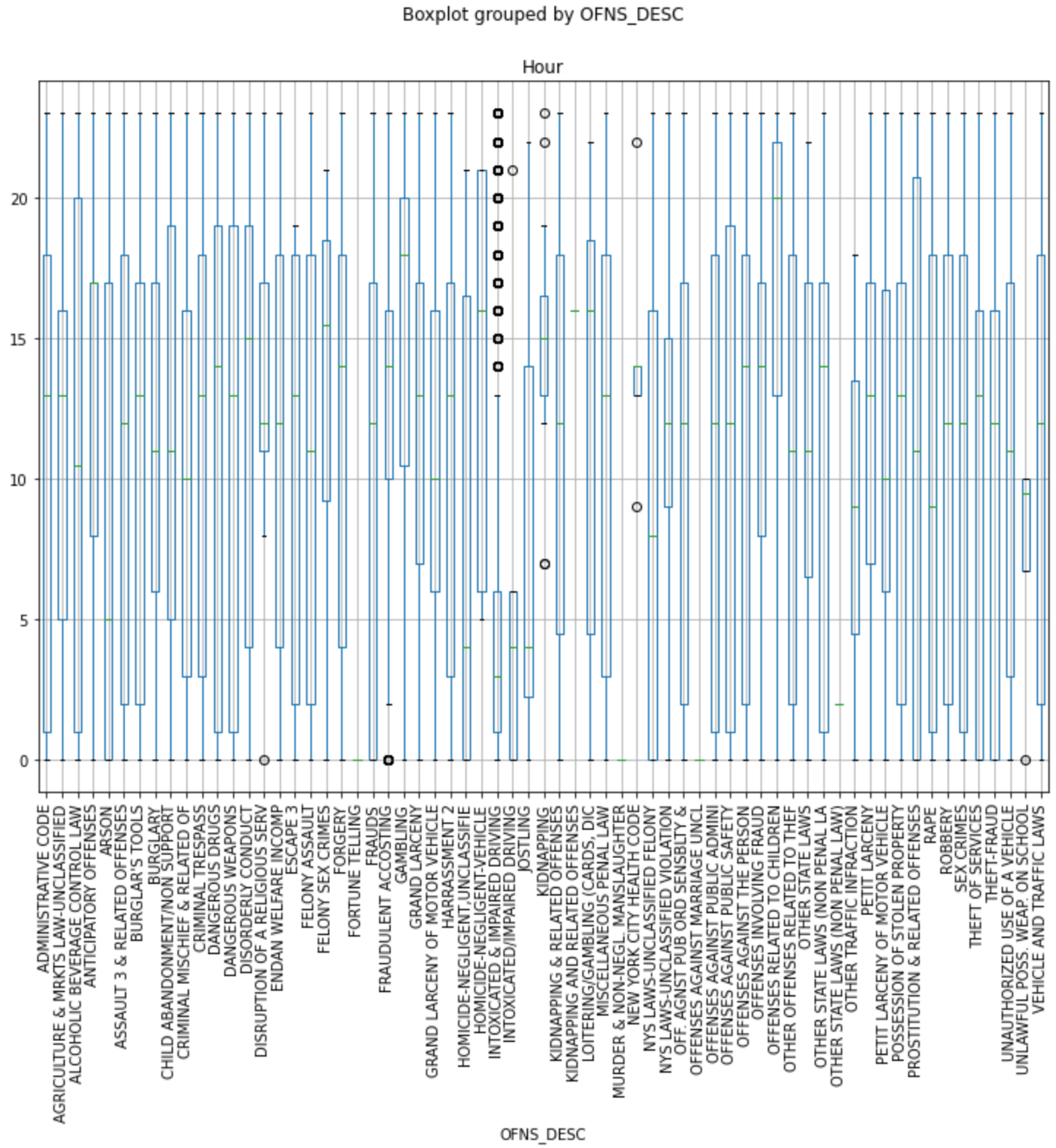
According to your plots, is there a spike in calls reporting BURGLARY at any particular time? If so, Do you trust that this spike is legitimate, or could there be an issue with our data? Explain your reasoning in 1-2 sentences below.

BEGIN SOLUTION

END SOLUTION

In the cell below, we generate a boxplot which examines the hour of day of each crime broken down by the `OFNS_DESC` value. To construct this plot we used the DataFrame.boxplot (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.boxplot.html) documentation.

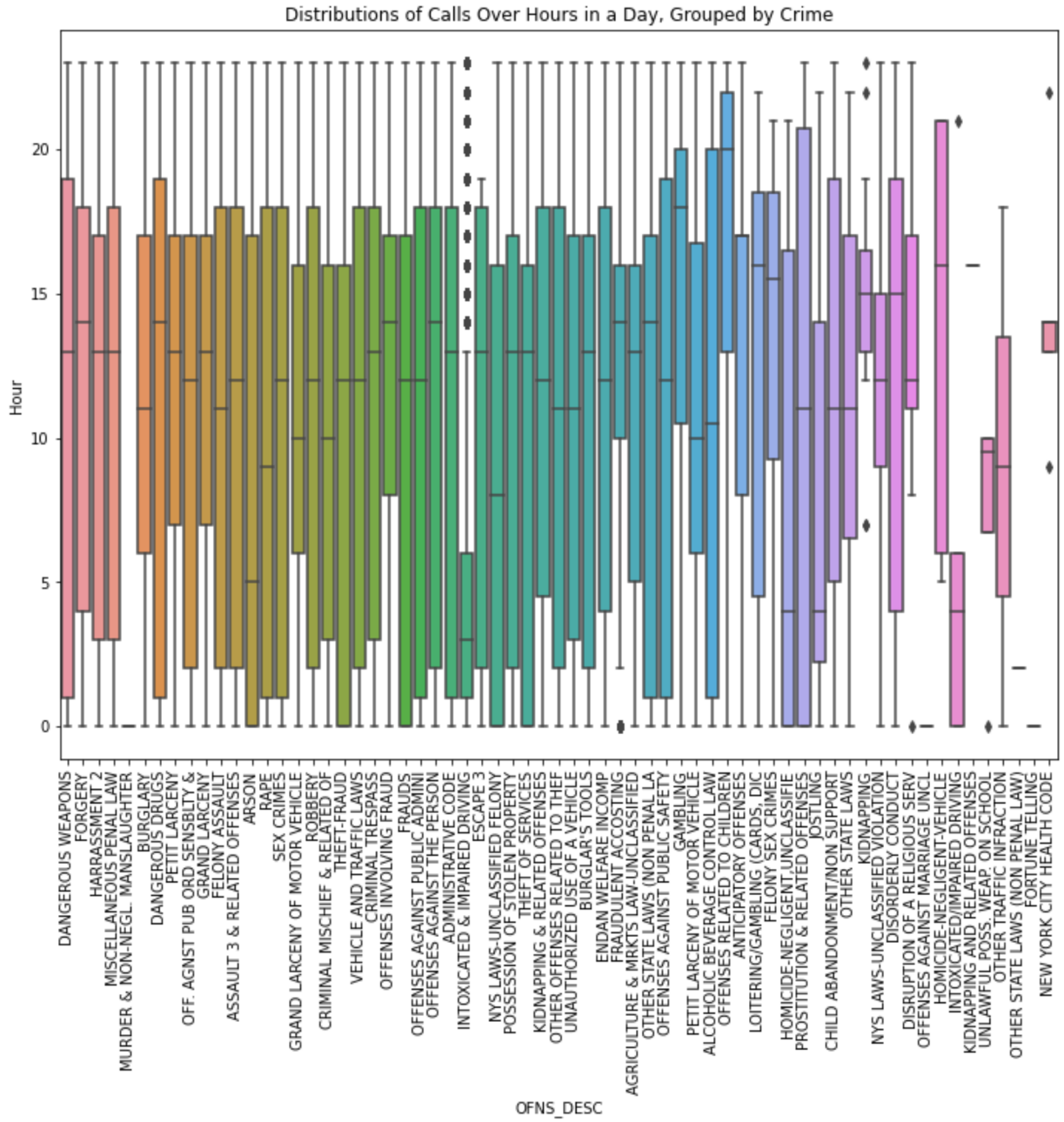
In [34]:
```
1 calls.boxplot(column="Hour", by='OFNS_DESC', rot=90);
```



While the pandas boxplot is informative, we can use seaborn to create a more visually-appealing plot. Using seaborn, regenerate a better box plot. See either the textbook (https://www.textbook.ds100.org/ch/06/viz_quantitative.html (https://www.textbook.ds100.org/ch/06/viz_quantitative.html)) or the seaborn boxplot documentation (https://seaborn.pydata.org/generated/seaborn.boxplot.html).

Looking at your plot, which crime type appears to have the largest interquartile range? Put your results into `answer` as a string.

In [36]:
```
# Todo: Make a boxplot with seaborn
### BEGIN SOLUTION


### END SOLUTION
```



3.9 - Visualization of crimes on a Map of NYC

finally we attempt to visualize the crimes committed in NYC on a Map. First we need to installing some mapping software. run the cell below to install folium package for mapping software.

In [37]:
```
!pip install --upgrade folium
```

```
Requirement already satisfied: folium in c:\users\andyg\anaconda3.1\lib\site-pa
ckages (0.12.1)
Requirement already satisfied: numpy in c:\users\andyg\anaconda3.1\lib\site-pac
kages (from folium) (1.20.1)
Requirement already satisfied: branca>=0.3.0 in c:\users\andyg\anaconda3.1\lib
\site-packages (from folium) (0.4.2)
Requirement already satisfied: jinja2>=2.9 in c:\users\andyg\anaconda3.1\lib\si
te-packages (from folium) (2.11.3)
Requirement already satisfied: requests in c:\users\andyg\anaconda3.1\lib\site-
packages (from folium) (2.25.1)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\andyg\anaconda3.1\l
ib\site-packages (from jinja2>=2.9->folium) (1.1.1)
Requirement already satisfied: chardet<5,>=3.0.2 in c:\users\andyg\anaconda3.1
\lib\site-packages (from requests->folium) (4.0.0)
Requirement already satisfied: idna<3,>=2.5 in c:\users\andyg\anaconda3.1\lib\s
ite-packages (from requests->folium) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\andyg\anaconda3.1
\lib\site-packages (from requests->folium) (2020.12.5)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\andyg\anaconda
3.1\lib\site-packages (from requests->folium) (1.26.4)
```

In [38]:
```
buNYC = calls[calls["OFNS_DESC"] == "BURGLARY"][:20]
len(buNYC)
```

Out[38]:
```
20
```

In [39]:
```
### Plot the maps for BURGLARY in NYC
### if it takes too much time or map does not show up, try plotting a subset
import folium
### BEGIN SOLUTION

### END SOLUTION
```

In [40]:
```
1 map1
```

Out[40]: Make this Notebook Trusted to load map: File -> Trust Notebook

Congratulations !!!

Submission Instructions

Output: Please **remove all output** from your notebook prior to submission
Data: **DO NOT** submit any data files. You may lose points if you do so
File Name: Please name the file as your_section_your_netID_Lab3.ipynb (eg. 01_adg133_Lab3.ipynb
Submit To: Canvas → Assignments → Lab3
Warning: Failure to follow directions may result in loss of points.