```
In [ ]: # Initialize Otter
        import otter
        grader = otter.Notebook("lab3.ipynb")
```

1 Lab 3: Data Cleaning and Visualization

In this lab, you will be working with a dataset from the City of Berkeley containing data on calls to the Berkeley Police Department. Information about the dataset can be found at this link.

1.1 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.

```
In [1]: import pandas as pd
        import numpy as np
        import zipfile
        import matplotlib
        import matplotlib.pyplot as plt
        import seaborn as sns

        %matplotlib inline
```

```
In [2]: plt.rcParams['figure.figsize'] = (12, 9)
```

2 Part 1: Cleaning and Exploring the Data

A file named `data` exists in the current directory.

```
In [5]: !ls -alh data ## what does -alh do?
```

```
total 688K
drwxr-xr-x 7 jovyan users  224 Apr 20 03:57 .
drwxr-xr-x 7 jovyan users  224 Apr 20 04:09 ..
-rw-r--r-- 1 jovyan users 1.9K Apr 20 03:57 ben_kurtovic.py
-rw-r--r-- 1 jovyan users 672K Apr 20 03:57 Berkeley_PD_-_Calls_for_Service.csv
-rw-r--r-- 1 jovyan users   33 Apr 20 03:57 do_not_readme.md
```

```
-rw-r--r-- 1 jovyan users   26 Apr 20 03:57 dummy.txt
-rw-r--r-- 1 jovyan users   78 Apr 20 03:57 hello_world.py
```

Let's now load the CSV file we have into a `pandas.DataFrame` object.

```
In [5]: calls = pd.read_csv("data/Berkeley_PD_-_Calls_for_Service.csv")
        calls.head()
```

Out[5]:

	CASENO	OFFENSE	EVENTDT	EVENTTM	CVLEGEND	CVDOW	InDbDate	Block_Location	BLKADDR	City	State
0	18022300	DISTURBANCE	04/18/2018 12:00:00 AM	22:17	DISORDERLY CONDUCT	3	09/06/2018 03:30:12 AM	OREGON STREET & MCGEE AVE\nBerkeley, CA\n(…	OREGON STREET & MCGEE AVE	Berkeley	CA
1	18026683	THEFT MISD. (UNDER $950)	05/09/2018 12:00:00 AM	21:25	LARCENY	3	09/06/2018 03:30:13 AM	200 UNIVERSITY AVE\nBerkeley, CA\n(37.865511, …	200 UNIVERSITY AVE	Berkeley	CA
2	18038550	THEFT MISD. (UNDER $950)	05/18/2018 12:00:00 AM	20:00	LARCENY	5	09/06/2018 03:30:09 AM	2200 MILVIA ST\nBerkeley, CA\n(37.868574, -122…	2200 MILVIA ST	Berkeley	CA
3	18014810	BURGLARY AUTO	03/13/2018 12:00:00 AM	08:50	BURGLARY - VEHICLE	2	09/06/2018 03:30:08 AM	1200 SIXTH ST\nBerkeley, CA\n(37.881142, -122…	1200 SIXTH ST	Berkeley	CA
4	18018643	ALCOHOL OFFENSE	03/31/2018 12:00:00 AM	13:29	LIQUOR LAW VIOLATION	6	09/06/2018 03:30:11 AM	CENTER STREET & SHATTUCK AVE\nBerkeley, CA…	CENTER STREET & SHATTUCK AVE	Berkeley	CA

We see that the fields include a case number, the offense type, the date and time of the offense, the "CVLEGEND" which appears to be related to the offense type, a "CVDOW" which has no apparent meaning, a date added to the database, and the location spread across four fields.

Let's also check some basic information about these files using the `DataFrame.describe` and `DataFrame.info` methods.

```
In [6]: calls.info()
        calls.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3788 entries, 0 to 3787
Data columns (total 11 columns):
CASENO            3788 non-null int64
OFFENSE           3788 non-null object
EVENTDT           3788 non-null object
EVENTTM           3788 non-null object
CVLEGEND          3788 non-null object
CVDOW             3788 non-null int64
InDbDate          3788 non-null object
Block_Location    3788 non-null object
BLKADDR           3766 non-null object
City              3788 non-null object
State             3788 non-null object
dtypes: int64(2), object(9)
memory usage: 325.7+ KB
```

Out[6]:

	CASENO	CVDOW
count	3.788000e+03	3788.000000
mean	1.804387e+07	2.997888
std	2.665970e+04	1.952160
min	1.801375e+07	0.000000
25%	1.802444e+07	1.000000
50%	1.803530e+07	3.000000
75%	1.804537e+07	5.000000
max	1.809135e+07	6.000000

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 in Question 1.

Before we go over the fields to see their meanings, the cell below will verify that all the events happened in Berkeley by grouping on the `City` and `State` columns. You should see that all of our data falls into one group.

```
In [7]: calls.groupby(["City","State"]).count()
```

Out[7]:

		CASENO	OFFENSE	EVENTDT	EVENTTM	CVLEGEND	CVDOW	InDbDate	\
City	State								
Berkeley	CA	3788	3788	3788	3788	3788	3788	3788	

		Block_Location	BLKADDR
City	State		
Berkeley	CA	3788	3766

2.1 Question 1

Above, when we called `head`, it seemed like `OFFENSE` and `CVLEGEND` both contained information about the type of event reported. What is the difference in meaning between the two columns? One way to probe this is to look at the `value_counts` for each Series.

```
In [8]: calls['OFFENSE'].value_counts().head(10)
```

```
Out[8]: BURGLARY AUTO                658
        THEFT MISD. (UNDER $950)     633
        DISTURBANCE                  277
        VANDALISM                    248
        VEHICLE STOLEN               235
        THEFT FELONY (OVER $950)     214
        BURGLARY RESIDENTIAL         194
        ASSAULT/BATTERY MISD.        190
        ROBBERY                      168
        NARCOTICS                    118
        Name: OFFENSE, dtype: int64
```

```
In [9]: calls['CVLEGEND'].value_counts().head(10)
```

```
Out[9]: LARCENY                   871
        BURGLARY - VEHICLE        658
        DISORDERLY CONDUCT        279
        ASSAULT                   263
        VANDALISM                 248
        MOTOR VEHICLE THEFT       235
        FRAUD                     208
        BURGLARY - RESIDENTIAL    194
        ROBBERY                   168
        DRUG VIOLATION            118
        Name: CVLEGEND, dtype: int64
```

2.1.1 Question 1a

Above, it seems like `OFFENSE` is more specific than `CVLEGEND`, e.g. "LARCENY" vs. "THEFT FELONY (OVER $950)". For those of you who don't know the word "larceny", it's a legal term for theft of personal property.

To get a sense of how many subcategories there are for each `OFFENSE`, set `calls_by_cvlegend_and_offense` equal to a multi-indexed series where the data is first indexed on the `CVLEGEND` and then on the `OFFENSE`, and the data is equal to the number of offenses in the database that match the respective `CVLEGEND` and

`OFFENSE`. For example, calls_by_cvlegend_and_offense["LARCENY", "THEFT FROM PERSON"] should return 24.

Use methods `.groupby()` and `.size()`. Refer to Hierarchical Indexes example for how to group using multiple columns.

In [10]:
```
calls_by_cvlegend_and_offense = ...

# BEGIN SOLUTION
calls_by_cvlegend_and_offense = calls.groupby(["CVLEGEND", "OFFENSE"]).size()
# END SOLUTION
```

In [ ]:
```
grader.check("q1a")
```

In [12]:
```
calls_by_cvlegend_and_offense
```

Out[12]:

CVLEGEND	OFFENSE	
ALL OTHER OFFENSES	MUNICIPAL CODE	3
ARSON	ARSON	18
ASSAULT	ASSAULT/BATTERY FEL.	73
	ASSAULT/BATTERY MISD.	190
BURGLARY - COMMERCIAL	BURGLARY COMMERCIAL	112
BURGLARY - RESIDENTIAL	BURGLARY RESIDENTIAL	194
BURGLARY - VEHICLE	BURGLARY AUTO	658
DISORDERLY CONDUCT	DISTURBANCE	277
	VICE	2
DRUG VIOLATION	NARCOTICS	118
FAMILY OFFENSE	DOMESTIC VIOLENCE	108
FRAUD	FRAUD/FORGERY	95
	IDENTITY THEFT	113
HOMICIDE	HOMICIDE	1
KIDNAPPING	KIDNAPPING	6
LARCENY	THEFT FELONY (OVER $950)	214
	THEFT FROM PERSON	24
	THEFT MISD. (UNDER $950)	633
LARCENY - FROM VEHICLE	THEFT FROM AUTO	87
LIQUOR LAW VIOLATION	ALCOHOL OFFENSE	53
MISSING PERSON	MISSING ADULT	30
	MISSING JUVENILE	8
MOTOR VEHICLE THEFT	VEHICLE STOLEN	235
NOISE VIOLATION	2ND RESPONSE	3
	DISTURBANCE - NOISE	1
RECOVERED VEHICLE	VEHICLE RECOVERED	15
ROBBERY	ROBBERY	168
SEX CRIME	SEXUAL ASSAULT FEL.	23
	SEXUAL ASSAULT MISD.	12
VANDALISM	VANDALISM	248
WEAPONS OFFENSE	BRANDISHING	43

```
GUN/WEAPON                    23
dtype: int64
```

2.1.2 Question 1b

In the cell below, set `answer1b` equal to a list of strings corresponding to the possible values for `OFFENSE` when `CVLEGEND` is "LARCENY". You can type the answer manually, or you can create an expression that automatically extracts the names.

```
In [13]: # You may use this cell for your scratch work as long as you enter
         # in your final answers in the answer1b variable.
         answer1b = ...


         # BEGIN SOLUTION
         answer1b = list(calls_by_cvlegend_and_offense['LARCENY'].index)
         # END SOLUTION
```

```
In [ ]: grader.check("q1b")
```

2.2 Question 2

What are the five crime types of CVLEGEND that have the most crime events? You may need to use `value_counts` to find the answer. Save your results into `answer2` as a list of strings.

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

```
In [19]: answer2 = ...

         # BEGIN SOLUTION
         answer2 = list(calls["CVLEGEND"].value_counts().head(5).keys())
         # END SOLUTION
```

```
In [ ]: grader.check("q2")
```

---

3 Part 2: Visualizing the Data

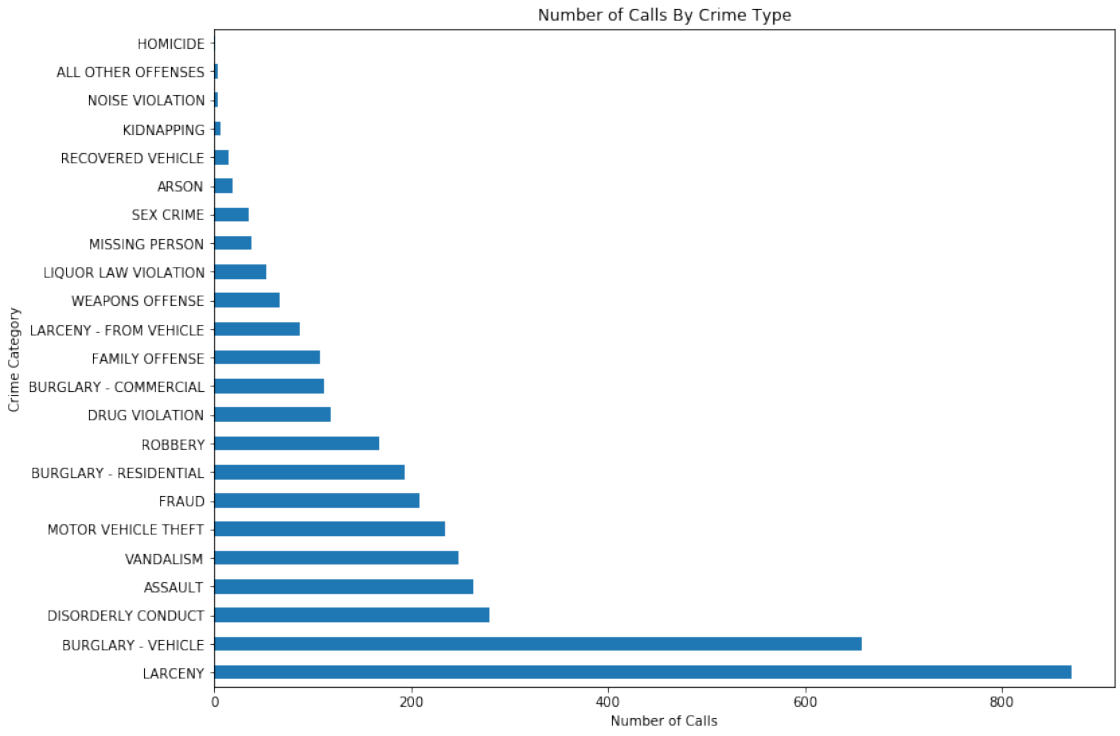
3.1 Pandas vs. Seaborn Plotting

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 and Seaborn documentation helpful!

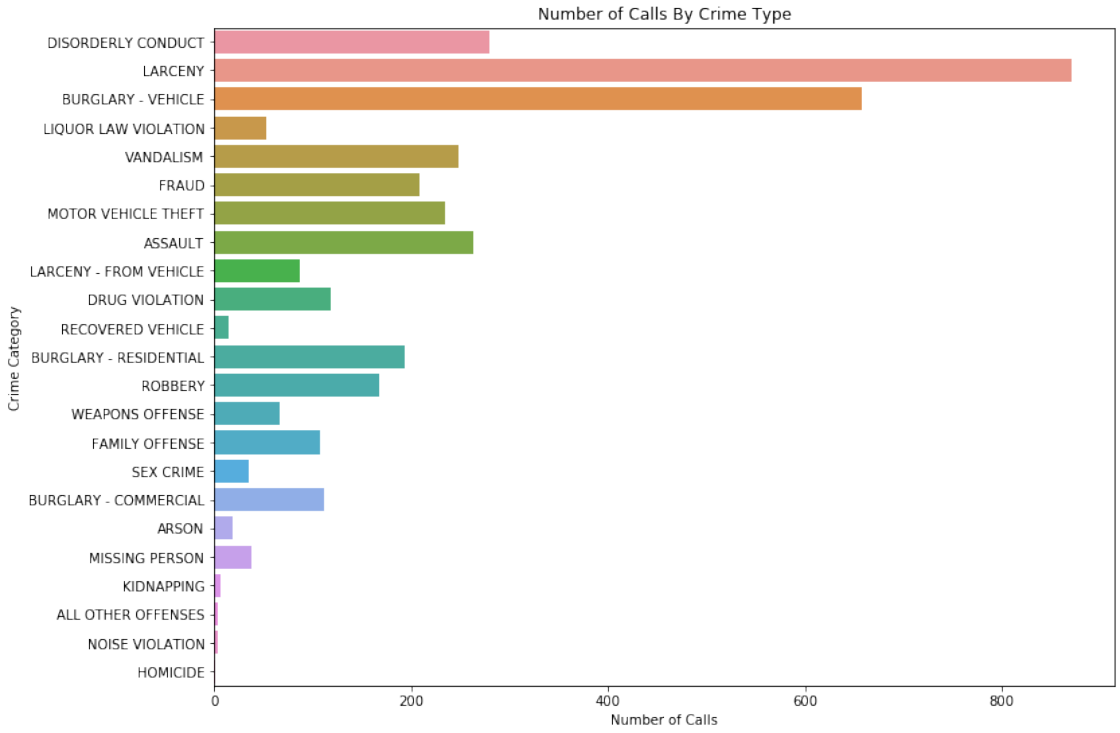
As an example of the built-in plotting functionality of pandas, the following example uses `plot` method of the `Series` class to generate a `barh` plot type to visually display the value counts for `CVLEGEND`.

```
In [24]: ax = calls['CVLEGEND'].value_counts().plot(kind='barh')
         ax.set_ylabel("Crime Category")
         ax.set_xlabel("Number of Calls")
         ax.set_title("Number of Calls By Crime Type");
```



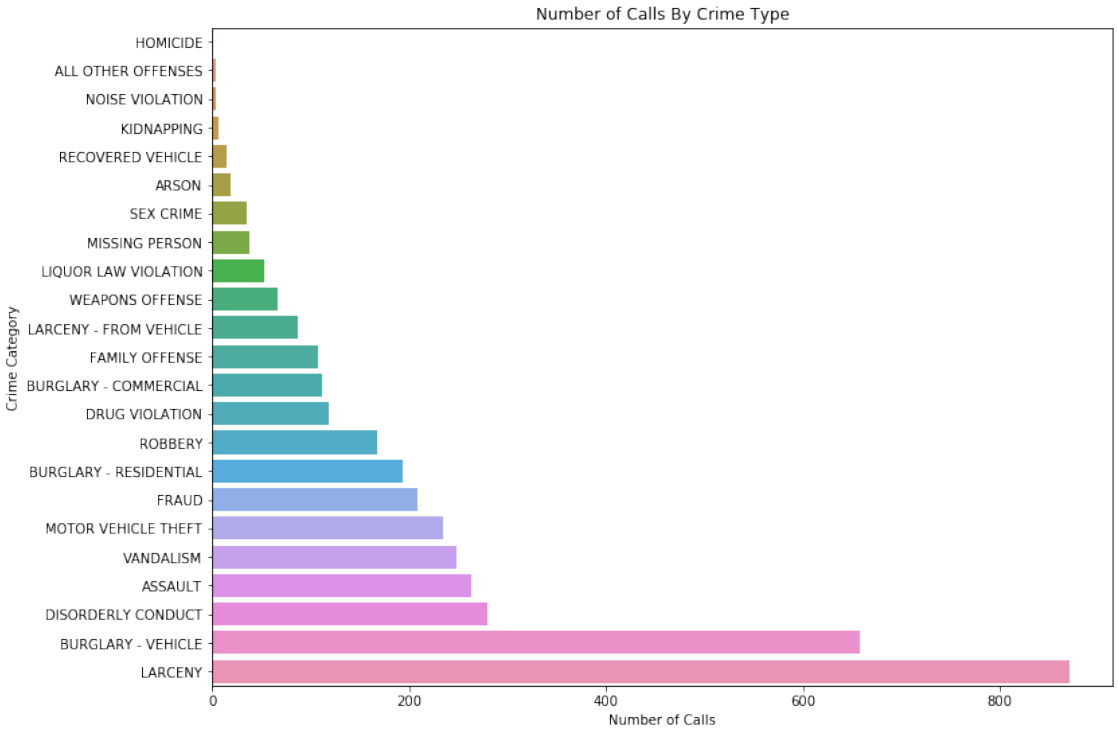
By contrast, the Seaborn library provides a specific function `countplot` built for plotting counts. It operates directly on the DataFrame itself i.e. there's no need to call `value_counts()` at all. This higher level approach makes it easier to work with. Run the cell below, and you'll see that the plot is much prettier (albeit in a weird order).

```
In [25]: ax = sns.countplot(data=calls, y="CVLEGEND")
         ax.set_ylabel("Crime Category")
         ax.set_xlabel("Number of Calls")
         ax.set_title("Number of Calls By Crime Type");
```



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`, we get the order we want.

```
In [26]: ax = sns.countplot(data=calls, y="CVLEGEND", order=calls["CVLEGEND"].value_counts(ascending=Tr
         ax.set_ylabel("Crime Category")
         ax.set_xlabel("Number of Calls")
         ax.set_title("Number of Calls By Crime Type");
```



Voilà! 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.

3.2 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 Question 3

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

The CVDOW field isn't named helpfully and it is hard to see the meaning from the data alone. According to the website linked at the top of this notebook, CVDOW is actually indicating the day that events happened. 0->Sunday, 1->Monday … 6->Saturday.

3.3.1 Question 3a

Add a new column `Day` into the `calls` dataframe that has the string weekday (eg. 'Sunday') for the corresponding value in CVDOW. For example, if the first 3 values of `CVDOW` are `[3, 6, 0]`, then the first 3 values of the `Day` column should be `["Wednesday", "Saturday", "Sunday"]`.

Hint: *Try using the Series.map function on `calls["CVDOW"]`. Can you assign this to the new column `calls["Day"]`?*

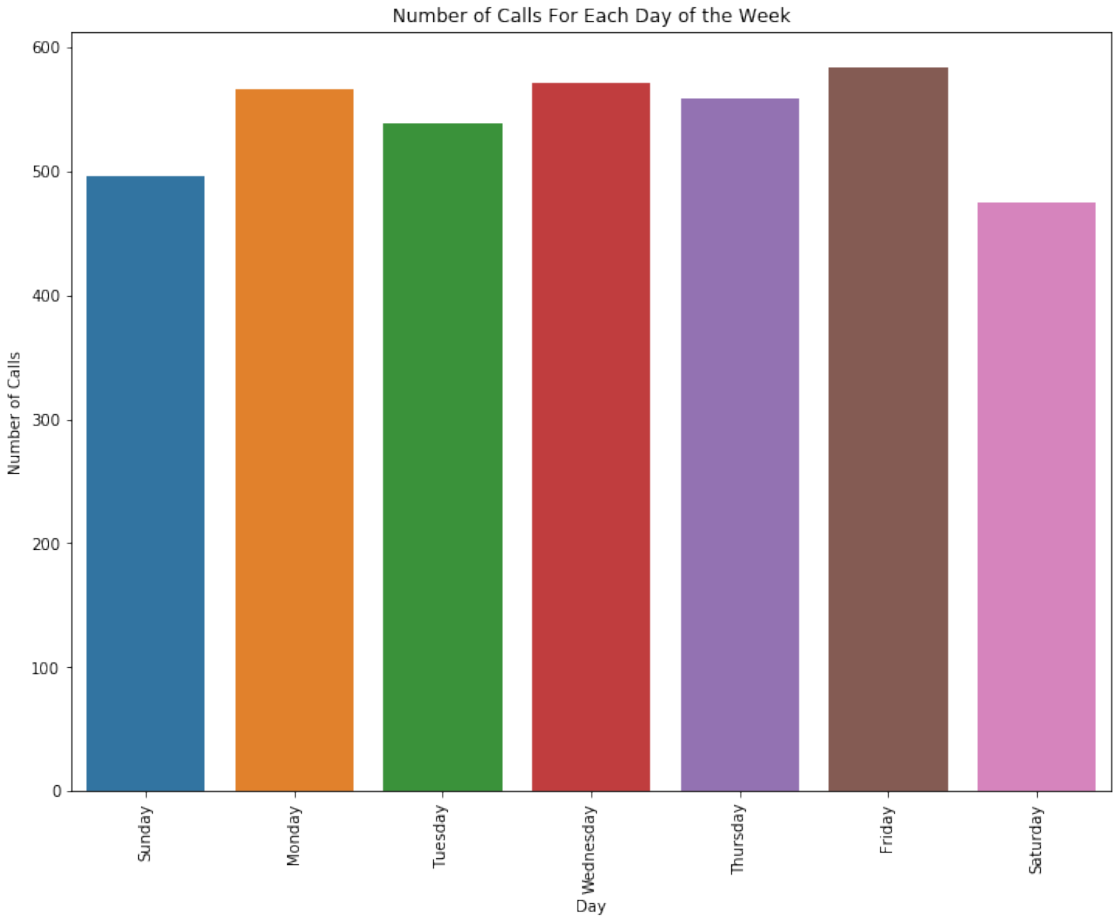
```
In [27]: days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
         day_indices = range(7)
         indices_to_days_dict = dict(zip(day_indices, days)) # Should look like {0:"Sunday", 1:"Monday"
         ...
         # BEGIN SOLUTION
         calls["Day"] = calls["CVDOW"].map(indices_to_days_dict)
         # END SOLUTION
```

```
In [ ]: grader.check("q3a")
```

3.3.2 Question 3b

Run the cell below to create a `seaborn` plot. This plot shows the number of calls for each day of the week. Notice the use of the `rotation` argument in `ax.set_xticklabels`, which rotates the labels by 90 degrees.

```
In [30]: ax = sns.countplot(data=calls, x='Day', order=days)
         ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
         ax.set_ylabel("Number of Calls")
         ax.set_title("Number of Calls For Each Day of the Week");
```

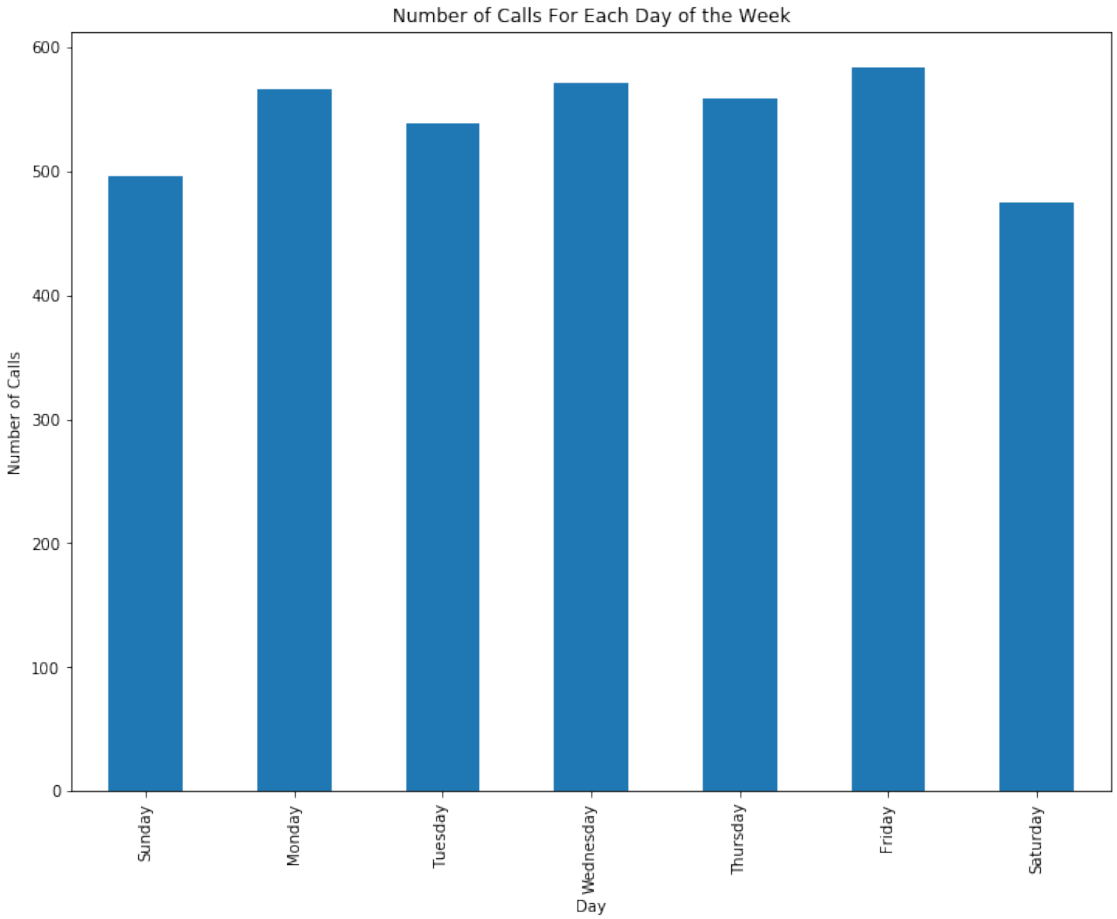


Now, let's make the same plot using `pandas`. Construct a vertical bar plot with the count of the number of calls (entries in the table) for each day of the week **ordered by the day of the week** (eg. `Sunday`, `Monday`, ...). Do not use `sns` for this plot. Be sure that your axes are labeled and that 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 [31]: # BEGIN SOLUTION
         ax = calls['Day'].value_counts()[days].plot(kind='bar')
         ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
         ax.set_xlabel("Day")
         ax.set_ylabel("Number of Calls")
         ax.set_title("Number of Calls For Each Day of the Week");
         # END SOLUTION

         # Leave this for grading purposes
         ax_3b = plt.gca()
```



In [ ]: grader.check("q3b")

3.4 Question 4

It seems weekdays generally have slightly more calls than Saturday or Sunday, but the difference does not look significant.

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

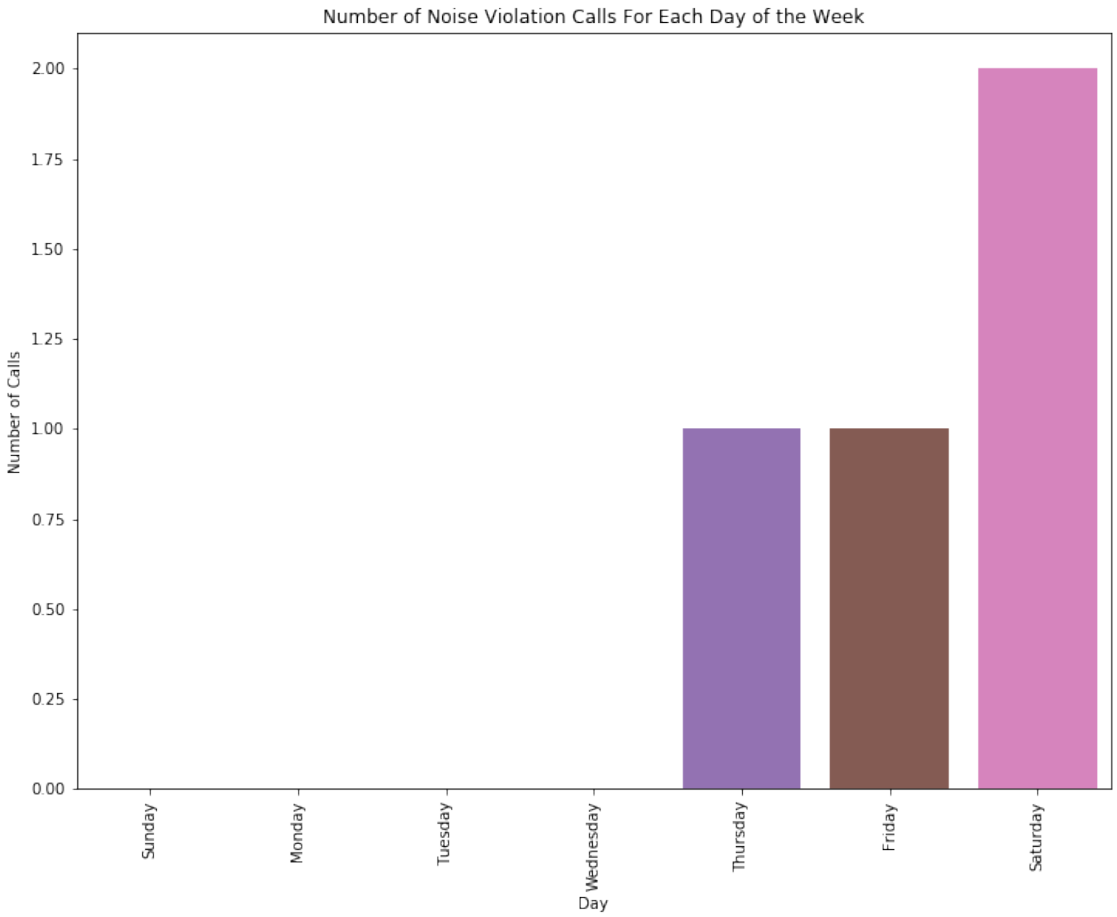
3.4.1 Question 4a

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 Question 3b as a starting point.*

```
In [34]: # BEGIN SOLUTION
         ax = sns.countplot(data= calls[calls["CVLEGEND"] == "NOISE VIOLATION"], x = "Day", order = days
         ax.set_xticklabels(ax.get_xticklabels(),rotation=90)
         ax.set_ylabel("Number of Calls")
         ax.set_title("Number of Noise Violation Calls For Each Day of the Week");
         # END SOLUTION

         # Leave this for grading purposes
         ax_4a = plt.gca()
```



```
In [ ]: grader.check("q4a")
```

3.4.2 Question 4b

Do you realize anything interesting about the distribution of NOISE VIOLATION calls over a week? Type a 1-sentence answer in the cell below.

Type your answer here, replacing this text.

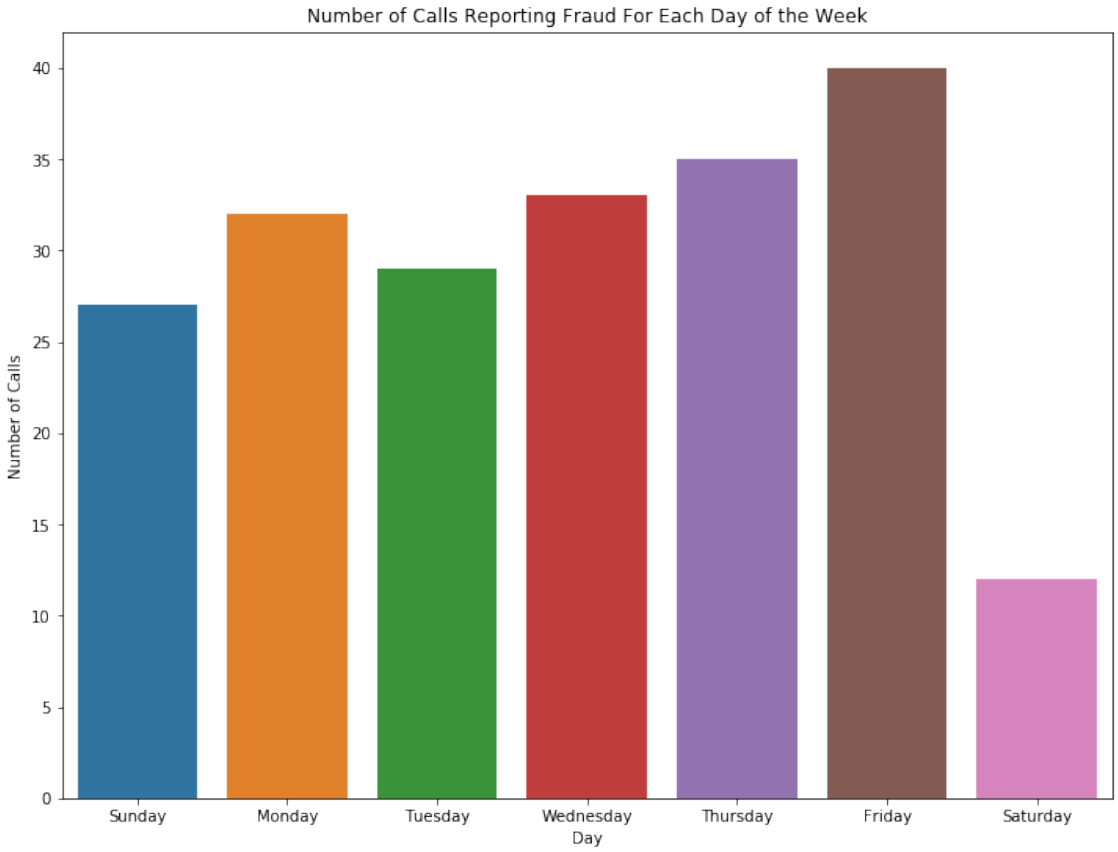
SOLUTION

3.5 Question 5

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 Question 4, but now looking at instances of the CVLEGEND "FRAUD" (instead of "NOISE VIOLATION"). Use either `pandas` or `seaborn` plotting as you desire.

```
In [37]: # BEGIN SOLUTION
         ax = sns.countplot(data=calls[calls["CVLEGEND"] == "FRAUD"], x="Day", order=days)
         ax.set_xlabel("Day")
         ax.set_ylabel("Number of Calls")
         ax.set_title("Number of Calls Reporting Fraud For Each Day of the Week");
         # END SOLUTION

         # Leave this for grading purposes
         ax_5 = plt.gca()
```




```
In [ ]: grader.check("q5")
```

3.6 Question 6

3.6.1 Question 6a

Now let's look at the EVENTTM 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 to select the substring.

- Series.astype to change the type.

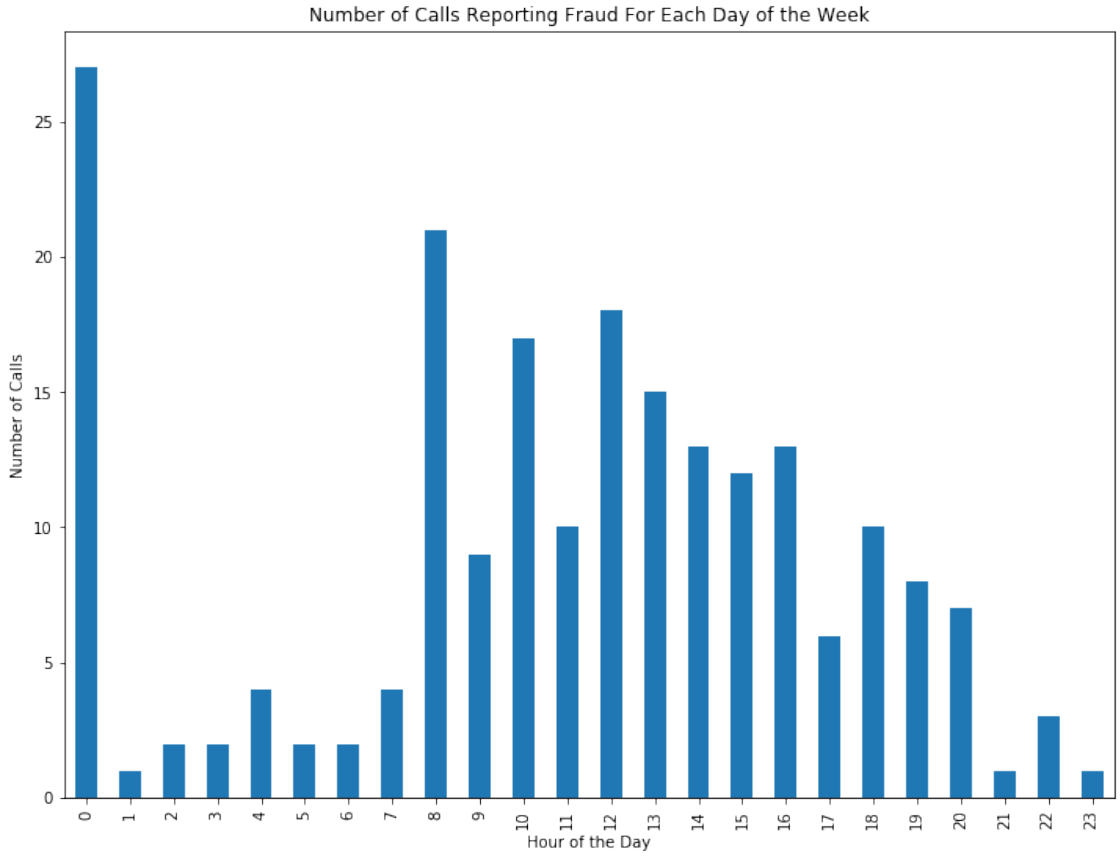
Hint: *The* `str` *helper member of a series can be used to grab substrings. For example,* `calls["EVENTTM"].str.slice(3,5)` *returns the minute of each hour of the* `EVENTTM`*.*

In [40]:
```
# BEGIN SOLUTION
calls["Hour"] = calls["EVENTTM"].str.slice(0,2).astype('int')
# END SOLUTION
```

In [ ]:
```
grader.check("q6a")
```

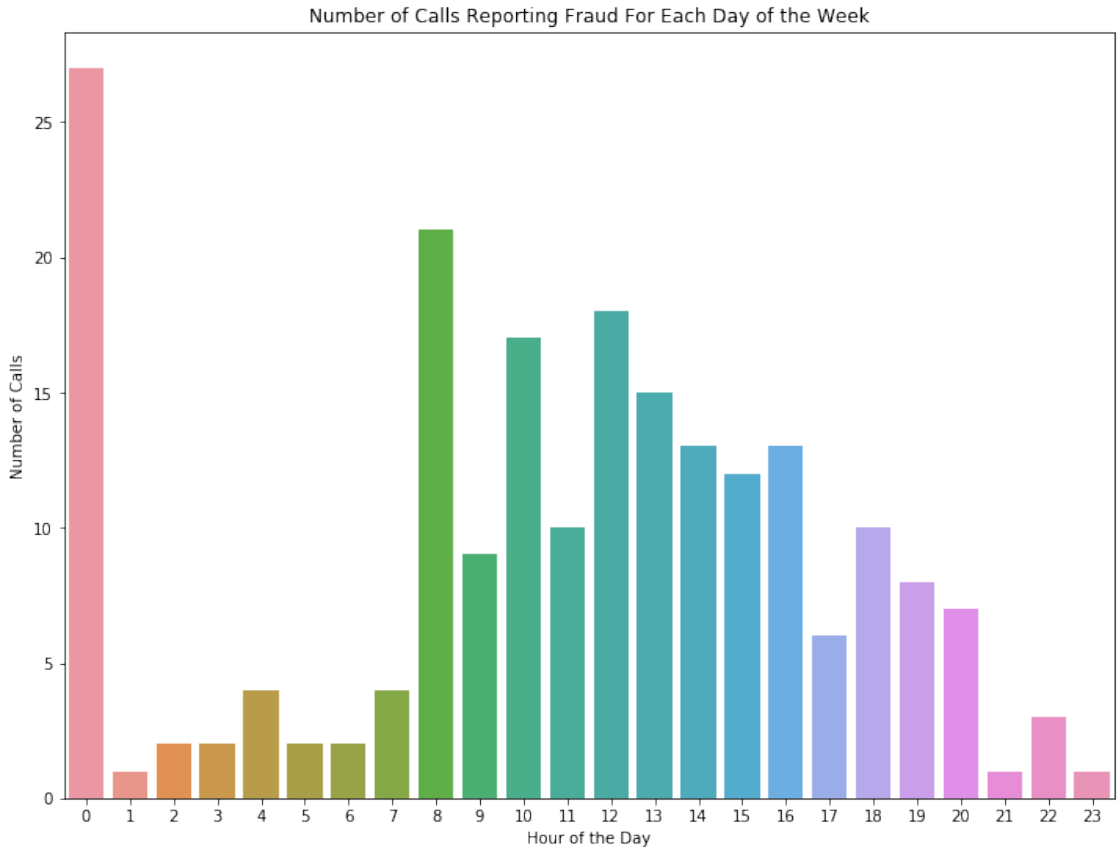
The code in the cell below creates a pandas bar plot showing the number of FRAUD crimes committed at each hour of the day.

In [45]:
```
ax = calls[calls["CVLEGEND"] == "FRAUD"]['Hour'].value_counts().sort_index().plot(kind='bar')
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");
```



The cell below contains a seaborn plot of the same data.

```
In [46]: ax = sns.countplot(calls[calls["CVLEGEND"] == "FRAUD"]['Hour'])
         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");
         #alternate solution: sns.countplot(data=calls[calls["CVLEGEND"] == "FRAUD"], x = 'Hour');
```



3.6.2 Question 6b

According to our plots, there seems to be a spike in calls reporting fraud at midnight. Do you trust that this spike is legitimate, or could there be an issue with our data? Explain your reasoning in 1-2 sentences in the cell below.

Type your answer here, replacing this text.

SOLUTION

Cell Intentionally Blank

To double-check your work, the cell below will rerun all of the autograder tests.

```
In [ ]: grader.check_all()
```

3.7 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

```
In [ ]: # Save your notebook first, then run this cell to export your submission.
        grader.export()
```