Pandas 2 (Data Analysis Example)

This lecture will do an example analysis of a reasonably large dataset.

Our data is from NYC Open data (https://nycopendata.socrata.com/) (over 1200+ datasets available!) It contains all 311 complaints from November 1, 2014 until January 6, 2015. This lecture is adapted from here (https://www.wakari.io/sharing/bundle/jvns/PyData%20NYC%202013%20tutorial).

The data is in the file: 311_calls_2months.csv

In [1]:
```
from pandas import Series, DataFrame
import pandas as pd
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [2]:
```
orig_data = pd.read_csv('Pandas_2_data/311_calls_2months.csv',
                        nrows=100000,
                        dtype=str,
                        parse_dates=['Created Date'])
```

In [3]:
```
print(orig_data.columns.values)
```
第一行 表头 的值

```
['Unique Key' 'Created Date' 'Closed Date' 'Agency' 'Agency Name'
 'Complaint Type' 'Descriptor' 'Location Type' 'Incident Zip'
 'Incident Address' 'Street Name' 'Cross Street 1' 'Cross Street 2'
 'Intersection Street 1' 'Intersection Street 2' 'Address Type' 'City'
 'Landmark' 'Facility Type' 'Status' 'Due Date'
 'Resolution Action Updated Date' 'Community Board' 'Borough'
 'X Coordinate (State Plane)' 'Y Coordinate (State Plane)'
 'Park Facility Name' 'Park Borough' 'School Name' 'School Number'
 'School Region' 'School Code' 'School Phone Number' 'School Address'
 'School City' 'School State' 'School Zip' 'School Not Found'
 'School or Citywide Complaint' 'Vehicle Type' 'Taxi Company Borough'
 'Taxi Pick Up Location' 'Bridge Highway Name' 'Bridge Highway Direction'
 'Road Ramp' 'Bridge Highway Segment' 'Garage Lot Name' 'Ferry Direction'
 'Ferry Terminal Name' 'Latitude' 'Longitude' 'Location']
```

In [4]:
```
orig_data.iloc[0]   第一行数据
```

Out[4]:

Unique Key	29641524
Created Date	2015-01-06 02:14:39
Closed Date	NaN
Agency	CHALL
Agency Name	CHALL
Complaint Type	Opinion for the Mayor
Descriptor	PUBLICSAFETY
Location Type	NaN
Incident Zip	NaN
Incident Address	NaN
Street Name	NaN
Cross Street 1	NaN
Cross Street 2	NaN
Intersection Street 1	NaN
Intersection Street 2	NaN
Address Type	NaN
City	NaN
Landmark	NaN
Facility Type	NaN
Status	Email Sent
Due Date	01/20/2015 02:15:41 AM
Resolution Action Updated Date	NaN
Community Board	0 Unspecified
Borough	Unspecified
X Coordinate (State Plane)	NaN
Y Coordinate (State Plane)	NaN

```
Park Facility Name                Unspecified
Park Borough                      Unspecified
School Name                       Unspecified
School Number                     Unspecified
School Region                     Unspecified
School Code                       Unspecified
School Phone Number               Unspecified
School Address                    Unspecified
School City                       Unspecified
School State                      Unspecified
School Zip                        Unspecified
School Not Found                          NaN
School or Citywide Complaint              NaN
Vehicle Type                              NaN
Taxi Company Borough                      NaN
Taxi Pick Up Location                     NaN
Bridge Highway Name                       NaN
Bridge Highway Direction                  NaN
Road Ramp                                 NaN
Bridge Highway Segment        1-1-1052163595
Garage Lot Name                           NaN
Ferry Direction                           NaN
Ferry Terminal Name                       NaN
Latitude                                  NaN
Longitude                                 NaN
Location                                  NaN
Name: 0, dtype: object
```

What do folks complain about?

In [5]:
```
# Complaint Type looks interesting
orig_data['Complaint Type']
```

Out[5]:
```
0          Opinion for the Mayor
1             Noise - Commercial
2                   Animal Abuse
3          Street Sign - Missing
4        Noise - Street/Sidewalk
                   ...
99995              HEAT/HOT WATER
99996              HEAT/HOT WATER
99997              HEAT/HOT WATER
99998              HEAT/HOT WATER
99999               PAINT/PLASTER
Name: Complaint Type, Length: 100000, dtype: object
```

What do they complain about the most?

In [6]:
```
# Let us aggregate all the complaints and see the frequency of each complaint type
vc = orig_data['Complaint Type'].value_counts()
vc[:10]
```

Out[6]:
```
HEAT/HOT WATER            20286
Blocked Driveway           5760
Street Light Condition     5288
Street Condition           4495
Illegal Parking            4045
UNSANITARY CONDITION       3855
PAINT/PLASTER              3384
PLUMBING                   2846
Noise - Commercial         2542
Opinion for the Mayor      2373
Name: Complaint Type, dtype: int64
```

It's November, and New Yorkers really want their HOT WATER flowing.

... and the least?

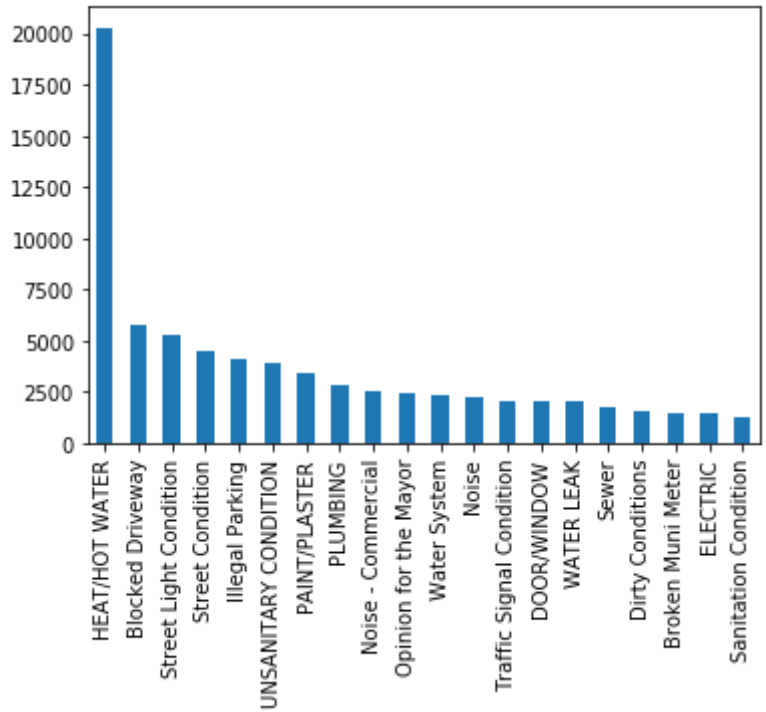
In [7]:
```
vc[-10:]
```

Out[7]:
```
Legal Services Provider Complaint    1
Squeegee                             1
Standpipe - Mechanical               1
Rangehood                            1
Invitation                           1
DHS Income Savings Requirement       1
Highway Sign - Missing               1
Public Assembly                      1
Tanning                              1
Transportation Provider Complaint    1
Name: Complaint Type, dtype: int64
```

It is often easier to plot things. Though we will look at plotting in detail in a later lecture, we can get started now.

In [8]:
```
# Plot a histogram of the top-20 complaints.
top_20_vc = vc[:20]
top_20_vc.plot(kind='bar')
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1441283e988>

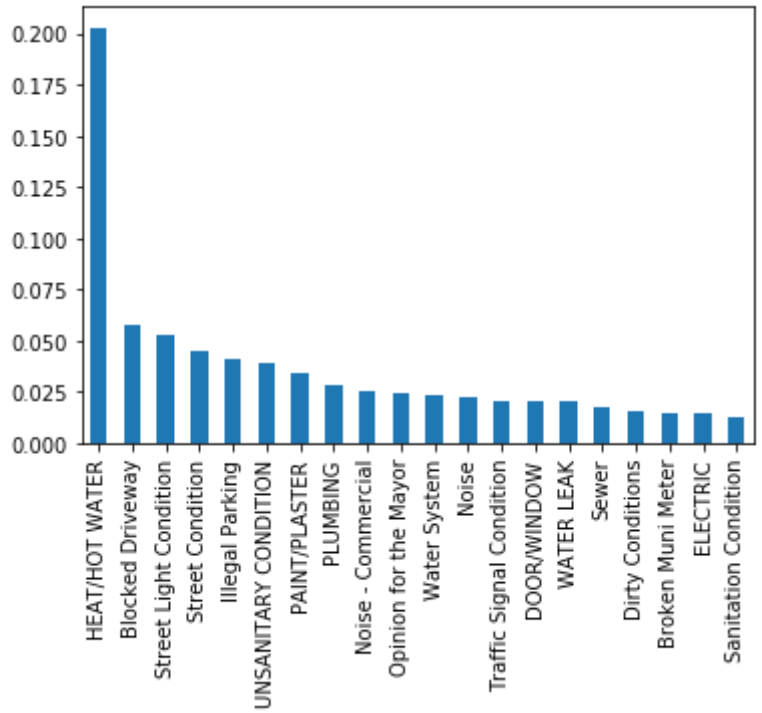


What if we want the y-axis to show the fraction of complaints, instead of the number of complaints?

We must **normalize** the value counts (vc) by the total number of complaints.

In [9]:
```
top_20_vc_fraction = top_20_vc / vc.sum()
top_20_vc_fraction.plot(kind='bar')
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1441283e3c8>



Which locations complain the most?

We have the incident zipcode, and we have the borough. Let's look at these.

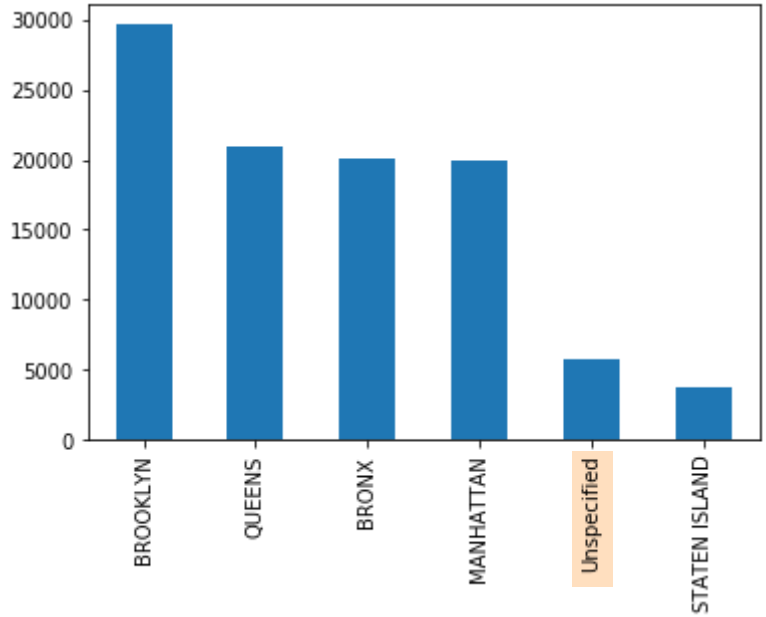
In [10]:
```
orig_data['Borough'][:10]
```

Out[10]:
```
0    Unspecified
1         QUEENS
2         QUEENS
3       BROOKLYN
4      MANHATTAN
5         QUEENS
6    Unspecified
7       BROOKLYN
8      MANHATTAN
9    Unspecified
Name: Borough, dtype: object
```

How do we plot the number of complaints for each Borough?

In [11]:
```
orig_data['Borough'].value_counts().plot(kind='bar')
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x144195f6688>

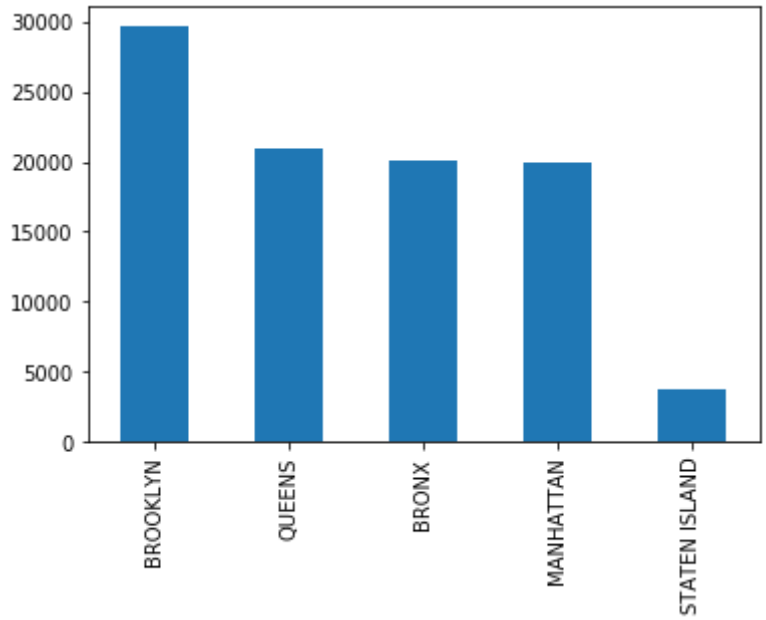


We are getting "Unspecified", but that's really a missing value. We should set missing values to NaN so that they are not counted.

In [12]:
```
mask = (orig_data['Borough'] == 'Unspecified')
orig_data.loc[mask, 'Borough'] = np.nan # This sets the value to NaN
              row    column

# Let's redo the bar plot
orig_data['Borough'].value_counts().plot(kind='bar')
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1441348b2c8>
```



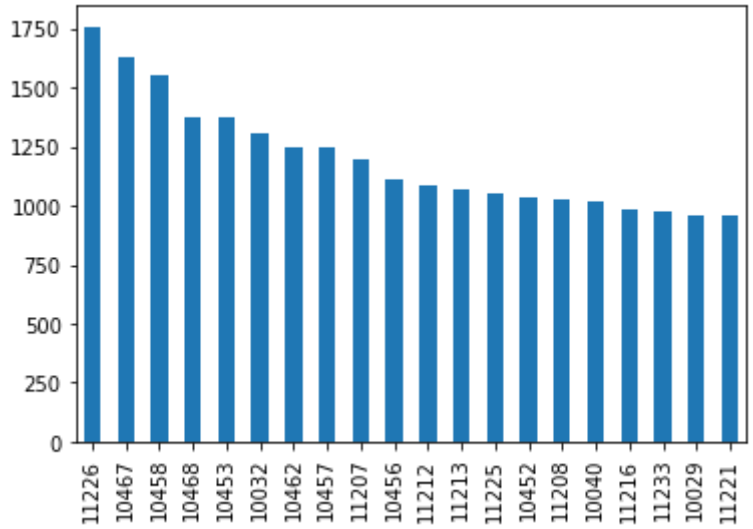
Let's do the same by zipcode.

In [13]:
```
orig_data['Incident Zip'][:5]
```

Out[13]:
```
0      NaN
1    11372
2    11416
3    11233
4    10022
Name: Incident Zip, dtype: object
```

In [14]:
```
# Number of complaints by zipcode
orig_data['Incident Zip'].value_counts()[:20].plot(kind='bar')
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x144134fb408>
```



Find the Borough for each zip-code

Let's do something a bit more complex. We have the top zip-codes, but that gives little understanding. Let us find the Borough for each zipcode.

How do we do this?

We want to create a **Series** of Borough indexed by zipcode. In this Series, we must have:

- for each zipcode, there should be one borough, and
- there should be no duplicate zipcodes in the index.

In [15]:
```
borough_zip = orig_data[['Borough', 'Incident Zip']]
borough_zip[:5]
```

Out[15]:

	Borough	Incident Zip
0	NaN	NaN
1	QUEENS	11372
2	QUEENS	11416
3	BROOKLYN	11233
4	MANHATTAN	10022

Trouble: both have missing values, and we must get rid of these.

In [16]:
```
# Detect missing boroughs
mask_borough = borough_zip['Borough'].notnull()
mask_borough[:5]
```

Out[16]:
```
0    False
1     True
2     True
3     True
4     True
Name: Borough, dtype: bool
```

In [17]:
```
# Similarly, we get a mask for the non-null zipcodes
mask_zip = borough_zip['Incident Zip'].notnull()
```

Which are the rows we want to keep?

In [18]:
```
# We combine the two masks
mask = (mask_borough & mask_zip)  # mask is True only if both mask_borough and mask_zip are True
```

In [19]:
```
# Apply the mask
borough_zip_clean = borough_zip[mask]
borough_zip_clean[:5]
```

Out[19]:

	Borough	Incident Zip
1	QUEENS	11372
2	QUEENS	11416
3	BROOKLYN	11233
4	MANHATTAN	10022
5	QUEENS	11368

Another option is to use **dropna()**

In [20]:
```
borough_zip_clean = borough_zip.dropna(how='any')
borough_zip_clean[:5]
```

Out[20]:

	Borough	Incident Zip
1	QUEENS	11372
2	QUEENS	11416
3	BROOKLYN	11233
4	MANHATTAN	10022
5	QUEENS	11368

We have a DataFrame of Borough and zipcode with no missing values. However this has two

problems:

- The (Borough, Incident Zip) pairs could be repeated multiple times.
- Some zipcodes could span multiple Boroughs (!)

We need to get rid of these **duplicates**.

In [21]:
```
borough_zip_dedup = borough_zip_clean.drop_duplicates(subset='Incident Zip')

print("Initial length of DataFrame =", len(borough_zip_clean))
print("After removing duplicates, length =", len(borough_zip_dedup))
```

```
Initial length of DataFrame = 87629
After removing duplicates, length = 196
```

DataFrame.drop_duplicates()

- By default, it removes duplicate rows
- So if **(Queens, 11372)** is repeated multiple times, only one such row remains.

DataFrame.drop_duplicates(subset='Incident Zip')

- This means we want Pandas to use only the zip-code while determining duplicates

- instead of both Borough and zip-code

- For any zip-code, only one **(Borough, Incident Zip)** row will be retained

We now have a good DataFrame of unique (Zipcodes, Borough) pairs.

In [22]:
```
borough_zip_dedup[:5]
```

Out[22]:

	Borough	Incident Zip
1	QUEENS	11372
2	QUEENS	11416
3	BROOKLYN	11233
4	MANHATTAN	10022
5	QUEENS	11368

However, we need to create a **Series** with the zipcode as index.

In [23]:
```
tmp_df = borough_zip_dedup.set_index('Incident Zip')
tmp_df[:5]
```

Out[23]:

Incident Zip	Borough
11372	QUEENS
11416	QUEENS
11233	BROOKLYN
10022	MANHATTAN
11368	QUEENS

In [24]:
```
# Recall that each column of a DataFrame is a Series.
borough_zip_series = tmp_df['Borough']
borough_zip_series[:5]
```

Out[24]:
```
Incident Zip
11372       QUEENS
11416       QUEENS
11233     BROOKLYN
10022    MANHATTAN
11368       QUEENS
Name: Borough, dtype: object
```

Summary (find the Borough for each zip-code)

We wanted to get a Series of Boroughs, indexed by Zipcode.

- We selected zipcodes and boroughs from the full data

```
borough_zip = orig_data[['Borough', 'Incident Zip']]
```

- We removed missing values by applying a mask.
  - there is also a dropna() method which does what we did

- We removed duplicates
  - drop_duplicates() method

- We set the zipcode to be the index
  - set_index() method

- Finally, we selected the 'Borough' Series, now indexed by zipcode.

Plot the most *interesting* zip-codes

Let us again plot the number of incidents by zip-code, but with the zip-code labels replaced by the corresponding Boroughs.

How do we do this?

1. Get the number of complaints by zipcode. This gives a Series, indexed by zipcode.
2. Get a Series of Boroughs, again indexed by zipcode.
3. **Rename** the index of the first series using the Series of Step 2.

Step 1: Get the number of complaints by zipcode.

In [25]:
```
# Step 1: Get the number of complaints by zipcode.
vc = orig_data['Incident Zip'].value_counts()
vc[:5]
```

Out[25]:
```
11226    1758
10467    1632
10458    1554
10468    1373
10453    1372
Name: Incident Zip, dtype: int64
```

Step 2: Get a Series of Boroughs, indexed by zipcode.

In [26]:
```
borough_zip_series[:5]
```

Out[26]:
```
Incident Zip
11372        QUEENS
11416        QUEENS
11233      BROOKLYN
10022     MANHATTAN
11368        QUEENS
Name: Borough, dtype: object
```

Step 3: Replace the index of the value-counts Series (vc) by the corresponding Borough from borough_zip_series.

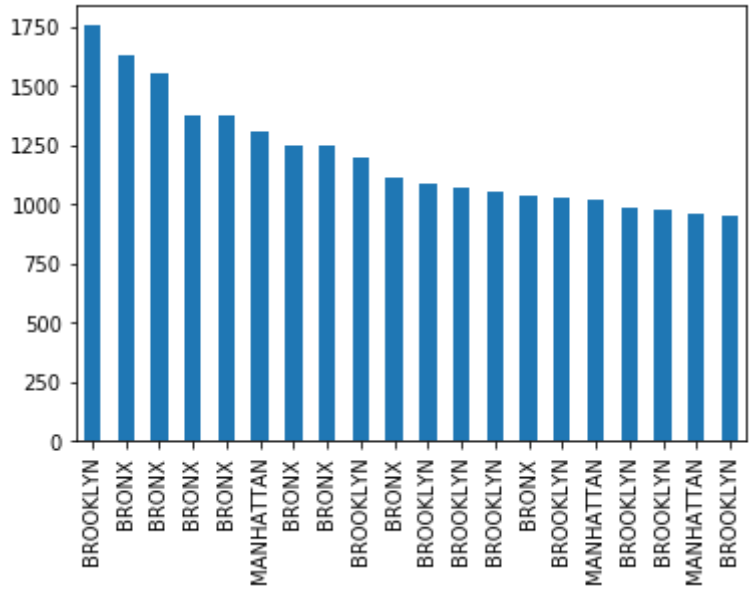
In [27]:
```
# Step 3: Replace index of vc with borough_zip_series
vc_renamed = vc.rename(borough_zip_series)
vc_renamed[:5]
```

把index从zip code换成boro

Out[27]:
```
BROOKLYN    1758
BRONX       1632
BRONX       1554
BRONX       1373
BRONX       1372
Name: Incident Zip, dtype: int64
```

In [28]:
```
# Finally, let us re-plot the value counts by zip-code, but with the Borough name as the label
vc_renamed[:20].plot(kind='bar')
```

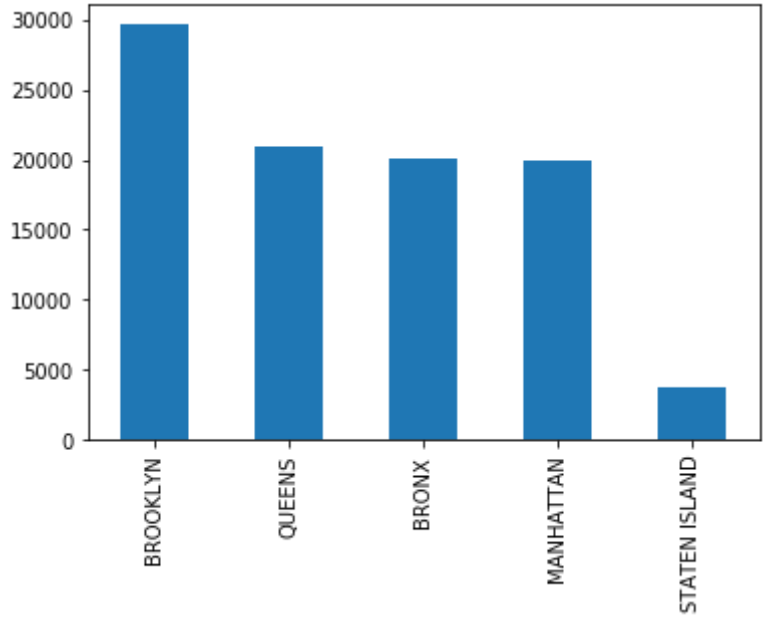
Out[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x144135e7708>
```



- **Surprise!** The top-complaining zipcodes seem to be mostly from the Bronx, but we'd earlier seen that Brooklyn complains the most?

In [29]:
```
orig_data['Borough'].value_counts().plot(kind='bar')
```

Out[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14413f3c9c8>
```



Why? Maybe Brooklyn just has more zipcodes?

Find the number of zipcodes for each Borough.

How would we solve this?

In [30]:
```
zip_per_borough = orig_data[['Borough', 'Incident Zip']].dropna(how='any') \
                                                        .drop_duplicates() \
                                                        ['Borough'] \
                                                        .value_counts()

zip_per_borough
```

Out[30]:
```
QUEENS           65
MANHATTAN        58
BROOKLYN         40
BRONX            26
STATEN ISLAND    12
Name: Borough, dtype: int64
```

So, it isn't the case that Brooklyn has far more zipcodes than everyone else... Still, it has quite a few more than the Bronx.

Plot complaints per zipcode for each Borough.

In [31]:
```
# We have the borough, zipcode DataFrame with null values removed.
# Each row corresponds to one complaint.
borough_zip_clean[:5]
```

Out[31]:

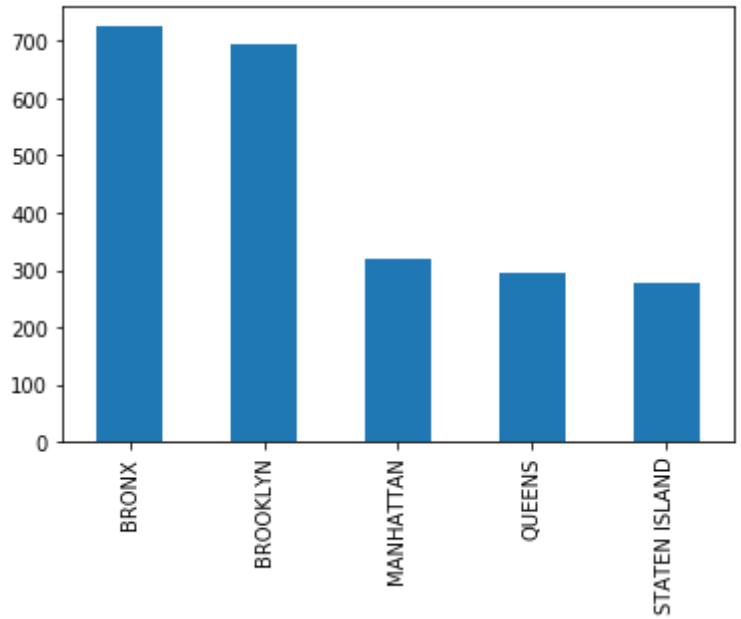
	Borough	Incident Zip
1	QUEENS	11372
2	QUEENS	11416
3	BROOKLYN	11233
4	MANHATTAN	10022
5	QUEENS	11368

In [32]:
```
# Get the number of complaints by borough
borough_counts = borough_zip_clean['Borough'].value_counts()
borough_counts
```

Out[32]:
```
Borough    Value counts
BROOKLYN          27748
QUEENS            19159
BRONX             18884
MANHATTAN         18523
STATEN ISLAND      3315
Name: Borough, dtype: int64
```

In [33]:
```
# Divide this by zip_per_borough, and plot.
borough_counts_per_zip = borough_counts / zip_per_borough
borough_counts_per_zip.plot(kind='bar')
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x144150b9e88>



The Bronx claims the throne!

I'm going to NYC and I hate noise. Which streets should I avoid?

Let us now focus on one particular kind of complaint: Noise complaints. How do we find all types of noise-related complaints?

In [34]:
```
orig_data['Complaint Type'].unique()  # Get all types of complaints
```

Out[34]:
```
array(['Opinion for the Mayor', 'Noise - Commercial', 'Animal Abuse',
       'Street Sign - Missing', 'Noise - Street/Sidewalk',
       'Illegal Parking', 'Consumer Complaint', 'Blocked Driveway',
       'Food Establishment', 'Street Condition', 'Graffiti', 'Rodent',
       'Noise - Helicopter', 'Homeless Person Assistance', 'Complaint',
       'Street Light Condition', 'Noise - Vehicle',
       'Overgrown Tree/Branches', 'Traffic Signal Condition',
       'Found Property', 'Benefit Card Replacement', 'PLUMBING',
       'HEAT/HOT WATER', 'UNSANITARY CONDITION', 'PAINT/PLASTER',
       'WATER LEAK', 'FLOORING/STAIRS', 'Sewer', 'Water System', 'Noise',
       'Request for Information', 'Dirty Conditions',
       'DCA / DOH New License Application Request', 'Highway Condition',
       'SCRIE', 'Missed Collection (All Materials)', 'Snow',
       'Agency Issues', 'Non-Residential Heat', 'Vending',
       'Derelict Vehicle', 'Other Enforcement', 'Litter Basket / Request',
       'Derelict Vehicles', 'Air Quality', 'Taxi Complaint',
       'Food Poisoning', 'DOF Literature Request',
       'Street Sign - Damaged', 'DOT Literature Request', 'Construction',
       'Root/Sewer/Sidewalk Condition', 'Indoor Air Quality',
       'Sweeping/Missed', 'Broken Muni Meter', 'Damaged Tree', 'Mold',
       'Ferry Inquiry', 'Homeless Encampment', 'Smoking',
       'Sanitation Condition', 'Derelict Bicycle',
       'Unsanitary Animal Pvt Property', 'Asbestos', 'School Maintenance',
       'Water Quality', 'Street Sign - Dangling', 'Taxi Compliment',
       'Indoor Sewage', 'Building/Use', 'Electrical', 'Drinking',
       'Elevator', 'General Construction/Plumbing', 'Traffic',
```

```
'Maintenance or Facility', 'Dead Tree', 'Hazardous Materials',
'For Hire Vehicle Complaint', 'Fire Safety Director - F58',
'Special Projects Inspection Team (SPIT)', 'Invitation',
'Sidewalk Condition', 'Bus Stop Shelter Placement',
'EAP Inspection - F59', 'Public Payphone Complaint', 'Taxi Report',
'Broken Parking Meter', 'DOF Property - Reduction Issue',
'Vacant Lot', 'Sweeping/Inadequate', 'Water Conservation',
'Industrial Waste', 'Recycling Enforcement', 'Curb Condition',
'DCA Literature Request', 'DOF Parking - Tax Exemption',
'DPR Internal', 'Illegal Tree Damage', 'Plumbing',
'Unsanitary Pigeon Condition', 'DPR Literature Request',
'Illegal Animal Kept as Pet', 'Animal in a Park',
'Violation of Park Rules', 'Unleashed Dog', 'Lead', 'Boilers',
'Special Enforcement', 'BEST/Site Safety', 'Urinating in Public',
'Beach/Pool/Sauna Complaint', 'Fire Alarm - Reinspection',
'Unsanitary Animal Facility', 'Window Guard',
'DHS Income Savings Requirement', 'For Hire Vehicle Report',
'ELECTRIC', 'DOOR/WINDOW', 'SAFETY', 'OUTSIDE BUILDING', 'GENERAL',
'APPLIANCE', 'ELEVATOR', 'Noise - House of Worship',
'Overflowing Litter Baskets', 'Bike Rack Condition', 'Panhandling',
'Tanning', 'Collection Truck Noise', 'Bike/Roller/Skate Chronic',
'Bridge Condition', 'Overflowing Recycling Baskets',
'Noise - Park', 'Open Flame Permit',
'Emergency Response Team (ERT)', 'Disorderly Youth',
'DEP Literature Request', 'Senior Center Complaint',
'Investigations and Discipline (IAD)', 'Illegal Fireworks',
'Illegal Animal Sold', 'Special Natural Area District (SNAD)',
'Cranes and Derricks', 'Ferry Complaint', 'Lifeguard',
```

```
'Municipal Parking Facility', 'Parking Card',
'Highway Sign - Damaged', 'Literature Request', 'Scaffold Safety',
'Fire Alarm - Addition', 'Fire Alarm - Modification', 'Rangehood',
'Legal Services Provider Complaint', 'Sprinkler - Mechanical',
'Plant', 'X-Ray Machine/Equipment', 'Misc. Comments',
'Posting Advertisement', 'Adopt-A-Basket',
'OEM Literature Request', 'Ferry Permit', 'Tattooing',
'Drinking Water', 'Compliment', 'City Vehicle Placard Complaint',
'Fire Alarm - New System', 'Miscellaneous Categories',
'Highway Sign - Missing', 'Internal Code',
'Transportation Provider Complaint', 'Squeegee', 'Stalled Sites',
'Standpipe - Mechanical', 'Public Assembly', 'SG-98'], dtype=object)
```

How do we find the noisiest streets?

1. Create a function that checks if the complaint type contains 'Noise'
    - How?
2. Select all noise-related complaints
    - Build a mask using the function of Step 1
3. Pick the streets that occur most frequently
    - value_counts()

Step 1: Create a function that checks if the complaint type contains 'Noise'.

In [35]:
```
# Side note: Pandas actually has functions to do regular expressions easily,
#           but we won't get into that here.
import re
def noisy(s):
    """Given a Complaint Type string, return True if it is
       a noise-related complaint."""
    return (len(re.findall('Noise', s)) > 0)

# Test
print(noisy('Noise - Commercial'))
print(noisy('ELEVATOR'))
```

```
True
False
```

Step 2: Build a mask using this function.

map is for Series; apply is for DataFrames

In [36]:
```
noise_mask = orig_data['Complaint Type'].map(noisy)
```

We have the mask that is True if the Complaint is noise-related; now we select those rows.

In [37]:
```
noise_complaints = orig_data[noise_mask]
noise_complaints[:5]
```

Out[37]:

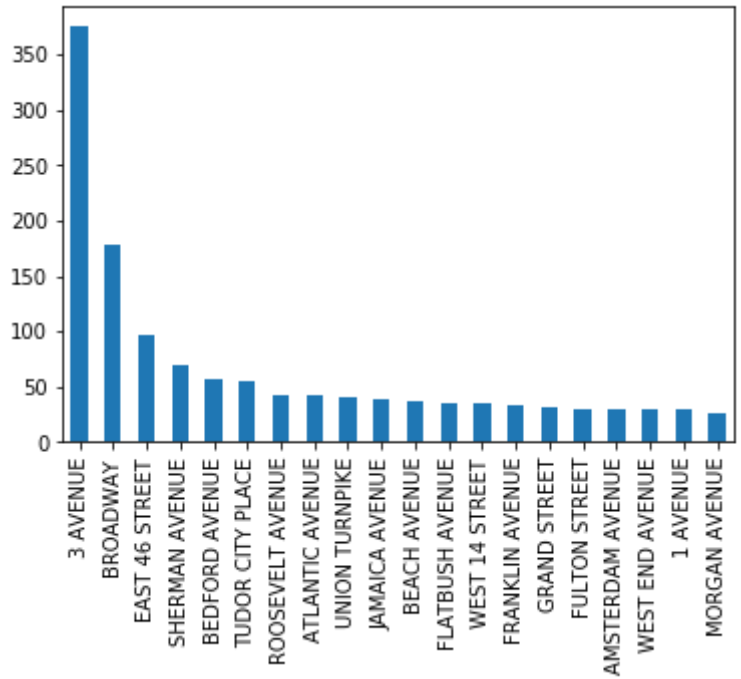
	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	...	Bridge Highway Name
1	29636054	2015-01-06 02:09:30	NaN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11372	70-06 ROOSEVELT AVENUE	...	NaN
4	29641040	2015-01-06 02:03:11	01/06/2015 02:36:38 AM	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Talking	Street/Sidewalk	10022	238 EAST 58 STREET	...	NaN
20	29639511	2015-01-06 01:32:51	NaN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11372	70-06 ROOSEVELT AVENUE	...	NaN
24	29641827	2015-01-06 01:27:24	NaN	EDC	Economic Development Corporation	Noise - Helicopter	Other	Above Address	10040	89 THAYER STREET	...	NaN
27	29638620	2015-01-06 01:24:16	01/06/2015 02:19:00 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10011	355 WEST 16 STREET	...	NaN

5 rows × 52 columns

Step 3: Pick the streets that occur most frequently.

In [38]:
```
# Which streets have the most noise complaints?
noise_vc = noise_complaints['Street Name'].value_counts()
noise_vc[:20].plot(kind='bar')
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x1441283e948>



Sweet.

No problems, right?

Are we sure there's only one "3 AVENUE"?

mask

In [39]:
```
noise_complaints[noise_complaints['Street Name'] == '3 AVENUE']['Borough'].value_counts()
```

Out[39]:
```
MANHATTAN    358
BROOKLYN      16
BRONX          1
Name: Borough, dtype: int64
```

There is a "3 AVENUE" in Manhattan, Brooklyn, and the Bronx. To find the noisy streets, we need to differentiate between these.

What do we do?

1. Create a new column 'Street & Borough', which will look like '3AVENUE (MANHATTAN)'
2. We will do value_counts() on this 'Street & Borough' column.

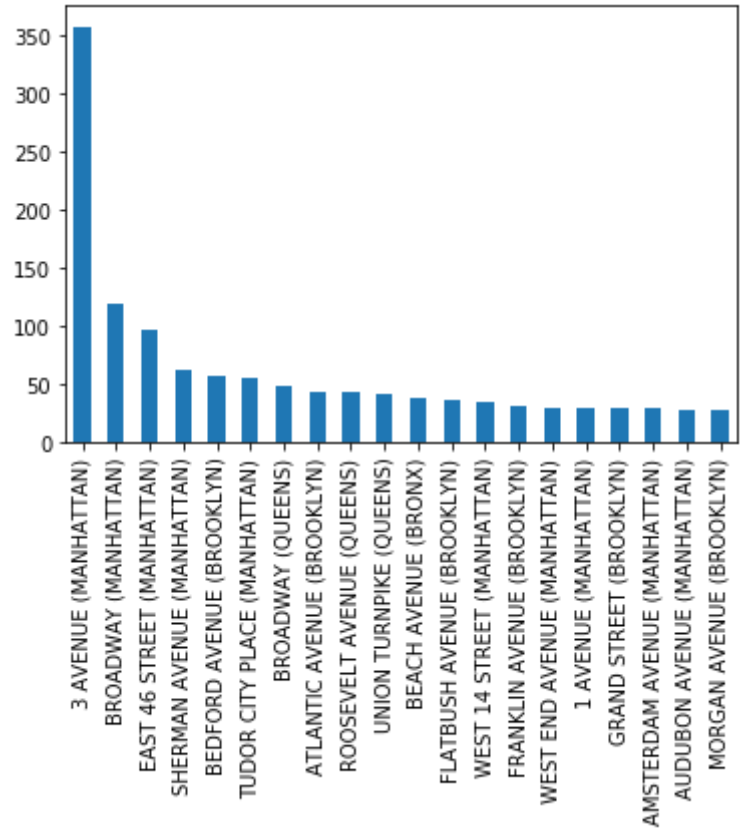
In [40]:
```
# Step 1: Create the new column
noise_complaints_copy = noise_complaints.copy()
noise_complaints_copy['Street & Borough'] = noise_complaints['Street Name'] + \
                                            ' (' + noise_complaints['Borough'] + ')'
noise_complaints_copy['Street & Borough'][:5]
```

add together as string

Out[40]:
```
1      ROOSEVELT AVENUE (QUEENS)
4     EAST 58 STREET (MANHATTAN)
20     ROOSEVELT AVENUE (QUEENS)
24     THAYER STREET (MANHATTAN)
27    WEST 16 STREET (MANHATTAN)
Name: Street & Borough, dtype: object
```

In [41]:
```
# Now we can do value_counts
noise_complaints_copy['Street & Borough'].value_counts()[:20].plot(kind='bar')
```

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x1441578c288>



Avoid noisy Manhattan streets!

Summary

We saw several common use cases:

- Find the most common or most uncommon items in a Series
  - _value_counts()_
- Deal with missing data
  - *dropna()*
  - or just create masks with *isnull()* and *notnull()*
- Deal with duplicates
  - *drop_duplicates()*
- Operate on rows rather than columns
  - *DataFrame.T*
  - (stands for *transpose*)

- Plotting
  - *plot(kind="bar")*