In [1]:

```
# Datastructures in Python
# List or Array - List or Array starts and end with square brackets or []
# and elements in the List are seperated by comma.
# List/Array are mutable or appendable, unordered and allow duplicates
# and multi data types.
numarray=[24,78,12,15,34,8,10,78,12]
print(numarray)
chrarray=['AAA',"john",'C',"BOY",'AAA']
print(chrarray)
mixedarray=[24,"DDD","A",78,90,199,"KKK","A","A"]
print(mixedarray)
```

```
[24, 78, 12, 15, 34, 8, 10, 78, 12]
['AAA', 'john', 'C', 'BOY', 'AAA']
[24, 'DDD', 'A', 78, 90, 199, 'KKK', 'A', 'A']
```

In [2]:

```
# Tupples - Tupples start and end with round brackets or paranthesis or()
# and elements are seperated by comma. Tupples are immutable and allow
# duplicates, and multi datatypes
numtupple=(24,78,12,15,34,8,10,78,12)
print(numtupple)
chrtupple=('AAA',"john",'C',"BOY",'AAA')
print(chrtupple)
mixedtupple=(24,"DDD","A",78,90,199,"KKK","A","A")
print(mixedtupple)
```

```
(24, 78, 12, 15, 34, 8, 10, 78, 12)
('AAA', 'john', 'C', 'BOY', 'AAA')
(24, 'DDD', 'A', 78, 90, 199, 'KKK', 'A', 'A')
```

In [3]:

```
# Sets - Sets start and end with curly brackets or {} and elements
# seperated by comma. Sets are mutable or appendable and do not allow
# duplicates.
numset={24,78,12,15,34,8,10,78,12}
print(numset)
chrset={'AAA',"john",'C',"BOY",'AAA'}
print(chrset)
mixedset={24,"DDD","A",78,90,199,"KKK","A","A"}
print(mixedset)
```

```
{34, 8, 24, 10, 12, 78, 15}
{'C', 'BOY', 'john', 'AAA'}
{'A', 90, 199, 24, 'KKK', 'DDD', 78}
```

In [4]:

```
# Dictionary - Dictionary start and end with curly brackets or {} and
# elements of dictionary are Key:Value pairs. Key is the character and
# value is numerical
dict_1={'Male':1,'Female':0}
print(dict_1)
```

```
{'Male': 1, 'Female': 0}
```

List Vs Set Vs Dictionary Vs Tuple

Lists	Sets	Dictionaries	Tuples
List = [10, 12, 15]	Set = {1, 23, 34} Print(set) -> {1, 23,24} Set = {1, 1} print(set) -> {1}	Dict = {"Ram": 26, "mary": 24}	Words = ("spam", "egss") Or Words = "spam", "eggs"
Access: print(list[0])	Print(set). Set elements can't be indexed.	print(dict["ram"])	Print(words[0])
Can contains duplicate elements	Can't contain duplicate elements. Faster compared to Lists	Can't contain duplicate keys, but can contain duplicate values	Can contains duplicate elements. Faster compared to Lists
List[0] = 100	set.add(7)	Dict["Ram"] = 27	Words[0] = "care" ->TypeError
Mutable	Mutable	Mutable	Immutable - Values can't be changed once assigned
List = []	Set = set()	Dict = {}	Words = ()
Slicing can be done print(list[1:2]) -> [12]	Slicing: Not done.	Slicing: Not done	Slicing can also be done on tuples
Usage: Use lists if you have a collection of data that doesn't need random access. Use lists when you need a simple, iterable collection that is modified frequently.	Usage: - Membership testing and the elimination of duplicate entries. - when you need uniqueness for the elements.	Usage: - When you need a logical association b/w key:value pair. - when you need fast lookup for your data, based on a custom key. - when your data is being constantly modified.	Usage: Use tuples when your data cannot change. A tuple is used in combination with a dictionary, for example, a tuple might represent a key, because its immutable.

6/25/2016 Rajkumar Rampelli - Python 15

In [5]:

```
# List/Array Methods
numarray.append(20)
numarray.append([50,79,100]) # Adds elements at the end of list
print(numarray)
print(numarray.count(20)) # Returns number of times element appears
print(numarray.index(20)) # Python starts with 0. Index Position
numarray.insert(5,88) # insert element at specified position
numarray
```

```
[24, 78, 12, 15, 34, 8, 10, 78, 12, 20, [50, 79, 100]]
1
9
```

Out[5]:

```
[24, 78, 12, 15, 34, 88, 8, 10, 78, 12, 20, [50, 79, 100]]
```

In [6]:

```
numarray=[24,78,12,15,34,8,10,78,12]
numarray.sort()
numarray
```

Out[6]:

```
[8, 10, 12, 12, 15, 24, 34, 78, 78]
```

In [7]:

```
# numarray.clear() removes all elements of list
numarray.pop(6) # remove element at a specified position
print(numarray)
numarray.remove(12) # removes the first item with specified value
print(numarray)
```

```
[8, 10, 12, 12, 15, 24, 78, 78]
[8, 10, 12, 15, 24, 78, 78]
```

In [8]:

```
# Libraries
# Pandas - Dataframe based operations library for read/write data, concat,
# reshape,replace,resample, date functions, etc. Dataframe is by default
# Rows are observations and columns are variables. Excel - Data Table.

# matplotlib.pyplot - Data Vizualization Library for creating charts
# and most importanly resizing the plot window, adding axis labels,
# adding title, etc.

# Numpy - Array/List based operations. All computations and calculations
# are performed by this library. Sub libraries in Numpy
# scipy.stats - Statistics in Python
# statsmodels - Timeseries or IOT Data
# sklearn - Machine Learning in Python

# io - inputoutput library for setting up working directory

# Other Libraries to be installed
# nltk, textblob, wordcloud - Natural Language Processing
# tensorflow, keras - Deep Learning & Image Processing
```

In [9]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [10]:

```
import io
%cd '/Users/rajeshprabhakarkaila/Desktop/Datasets'
```

```
/Users/rajeshprabhakarkaila/Desktop/Datasets
```

In [11]:

```
NYBnB=pd.read_excel("New York AirBnB.xlsx",sheet_name="New York AirBnB")
# pd refers to Library Name
# read_excel() - predefined function. Every Function must have paranthesis
# or (). Either paranthesis will be null or arguments given within
# Function Syntax - libraryname.functionname()
# Dataframe Function Sytntax - dataframename.functionname()
```

In [12]:

```
NYBnB.info()
# Class - Dataframe , Rows/Observation-41533, Columns/Variables-18
# Individual Variable Data Type - int64,float64,object, datetime64[ns]
# Non Null count highlights whether a variable has missing values.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41533 entries, 0 to 41532
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              41533 non-null  int64
 1   name                            41517 non-null  object
 2   host_id                         41533 non-null  int64
 3   host_name                       41527 non-null  object
 4   neighbourhood_group             41533 non-null  object
 5   neighbourhood                   41533 non-null  object
 6   latitude                        41533 non-null  float64
 7   longitude                       41533 non-null  float64
 8   room_type                       41533 non-null  object
 9   price                           41533 non-null  int64
 10  minimum_nights                  41533 non-null  int64
 11  number_of_reviews               41533 non-null  int64
 12  last_review                     32140 non-null  datetime64[ns]
 13  reviews_per_month               32140 non-null  float64
 14  calculated_host_listings_count  41533 non-null  int64
 15  availability_365                41533 non-null  int64
 16  number_of_reviews_ltm           41533 non-null  int64
 17  license                         1 non-null      object
dtypes: datetime64[ns](1), float64(3), int64(8), object(6)
memory usage: 5.7+ MB
```

In [13]:

```
# Indexing or Accessing Specific Rows or Columns
# Character Indexing - Accessing by Column or Variable Name. Name must
# exactly match
# Numeric Indexing - Accessing By Column or Variable Index Number. Python
# starts with 0 or Zero
```

In [14]:

```
# Character Indexing - 2 types
# 1) dataframename.columnname
# 2) dataframename["columnname"] used when Variable Name has space in
# between

NYBnB.price.head() # head() - By default First 5 Rows of Data
```

Out[14]:

```
0    275
1     75
2     60
3     68
4    175
Name: price, dtype: int64
```

In [15]:

```
NYBnB["price"].head()
```

Out[15]:

```
0    275
1     75
2     60
3     68
4    175
Name: price, dtype: int64
```

In [16]:

```
#Character Indexing-multicolumn/variable - dataframename[["var1","var2"]]
NYBnB[['price','minimum_nights','number_of_reviews']].head(10)
# First 10 Rows of Data
```

Out[16]:

	price	minimum_nights	number_of_reviews
0	275	21	3
1	75	2	118
2	60	30	50
3	68	2	559
4	175	30	49
5	65	30	1
6	124	4	218
7	68	30	187
8	220	3	316
9	62	30	242

In [17]:

```
# Numeric Indexing - Column Number
NYBnB.iloc[:,9].head() # :, before indicates Column/Variable Selection
```

Out[17]:

```
0    275
1     75
2     60
3     68
4    175
Name: price, dtype: int64
```

In [18]:

```
NYBnB.iloc[9] # Row Selection
```

Out[18]:

```
id                                            6990
name                          UES Beautiful Blue Room
host_id                                      16800
host_name                                      Cyn
neighbourhood_group                      Manhattan
neighbourhood                          East Harlem
latitude                                  40.78778
longitude                                -73.94759
room_type                           Private room
price                                          62
minimum_nights                                 30
number_of_reviews                             242
last_review                   2022-10-21 00:00:00
reviews_per_month                            1.52
calculated_host_listings_count                  1
availability_365                              308
number_of_reviews_ltm                           8
license                                       NaN
Name: 9, dtype: object
```

In [19]:

```
# Numeric Indexing - Multicolumn/variable
NYBnB.iloc[:,[9,10,11,14,17]].head() # Column Selection
```

Out[19]:

	price	minimum_nights	number_of_reviews	calculated_host_listings_count	license
0	275	21	3	1	NaN
1	75	2	118	1	NaN
2	60	30	50	2	NaN
3	68	2	559	1	NaN
4	175	30	49	3	NaN

In [20]:

```
NYBnB.iloc[[9,10,11,14,17]] # Row Selection
```

Out[20]:

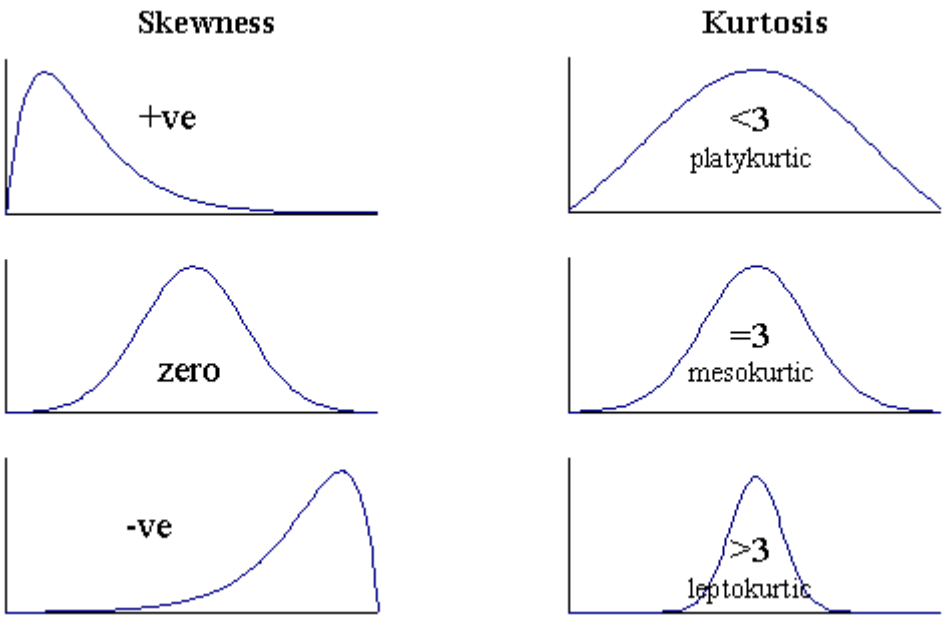
	id	name	host_id	host_name	neighbourhood_group	neighbourhood	l
9	6990	UES Beautiful Blue Room	16800	Cyn	Manhattan	East Harlem	4(
10	8490	Maison des Sirenes1,bohemian, luminous apartment	25183	Nathalie	Brooklyn	Bedford-Stuyvesant	4(
11	7801	Sweet and Spacious Brooklyn Loft	21207	Chaya	Brooklyn	Williamsburg	4(
14	9357	Midtown Pied-a-terre	30193	Tommi	Manhattan	Hell's Kitchen	4(
17	12937	1 Stop fr. Manhattan! Private Suite,Landmark B...	50124	Orestes	Queens	Long Island City	4(

In [21]:

```
# Exploratory Data Analysis (EDA) - Descriptive Statistics, Data
# Vizualization and Data Aggregation/Grouping/Wranggling
# EDA provides complete understanding of Data.

# Descriptive Statistics - Describe the Data. Numeric Variables -
# Univariate Statistics.
# Measures of Central Tendency (Midpoint) - Mean, Median, Mode
# Measures of Dispersion(scattering of observations around mean) -
# Variance and Standard Deviation
# Measures of Location - Quartiles, Percentiles and Deciles
# Measures of Asymmetry(how close data is to Normal Distribution/Bell Curve)
# Skewness and Kurtosis

# Mean and Median must be closer to one another as mean gets distorted by
# small or large values. IF mean is distorted median must be used
# STandard Deviation must be lower and it doesnot have fixed range and
# it depends on mathematical unit like tens,hundreds, thousands, etc.
# Positive Skewness indicates Peak of Curve is on Left Side and Negative
# Skewness indicates Peak of Curve on Right Side.
# Positive Kurtosis indicates tall and narrow peak and Negative Kurtosis
# indicates flat and wide peak
```



In [22]:

```
NYBnB.price.describe() # N, Min, Max, Mean, Median/50%, std, 25%/Q1,75%/Q3
# Significant difference between Mean and Median
# Std Deviation is High
# Median is best mid point
```

Out[22]:

```
count    41533.000000
mean       221.978282
std        919.502236
min          0.000000
25%         80.000000
50%        131.000000
75%        220.000000
max      98159.000000
Name: price, dtype: float64
```

In [23]:

```
print("Skewness",NYBnB.price.skew()) # Very High Positive Skewness
print("Kurtosis",NYBnB.price.kurt()) # Very High Positive Kurtosis
```

```
Skewness 78.22503185691421
Kurtosis 7498.176026467221
```

In [24]:

```
NYBnB.columns # Name of the Variable/Columns
```

Out[24]:

```
Index(['id', 'name', 'host_id', 'host_name', 'neighbourhood_group',
       'neighbourhood', 'latitude', 'longitude', 'room_type', 'price',
       'minimum_nights', 'number_of_reviews', 'last_review',
       'reviews_per_month', 'calculated_host_listings_count',
       'availability_365', 'number_of_reviews_ltm', 'license'],
      dtype='object')
```

In [25]:

```
NYBnB[['minimum_nights', 'number_of_reviews']].describe()
```

Out[25]:

	minimum_nights	number_of_reviews
count	41533.000000	41533.000000
mean	18.592204	26.204994
std	30.699921	56.178847
min	1.000000	0.000000
25%	2.000000	1.000000
50%	10.000000	5.000000
75%	30.000000	25.000000
max	1250.000000	1666.000000

In [26]:

```
print("skewness",NYBnB[['minimum_nights','number_of_reviews']].skew())
print("kurtosis",NYBnB[['minimum_nights','number_of_reviews']].kurt())
```

```
skewness minimum_nights       14.331509
number_of_reviews     5.570549
dtype: float64
kurtosis minimum_nights       375.692050
number_of_reviews     65.747075
dtype: float64
```

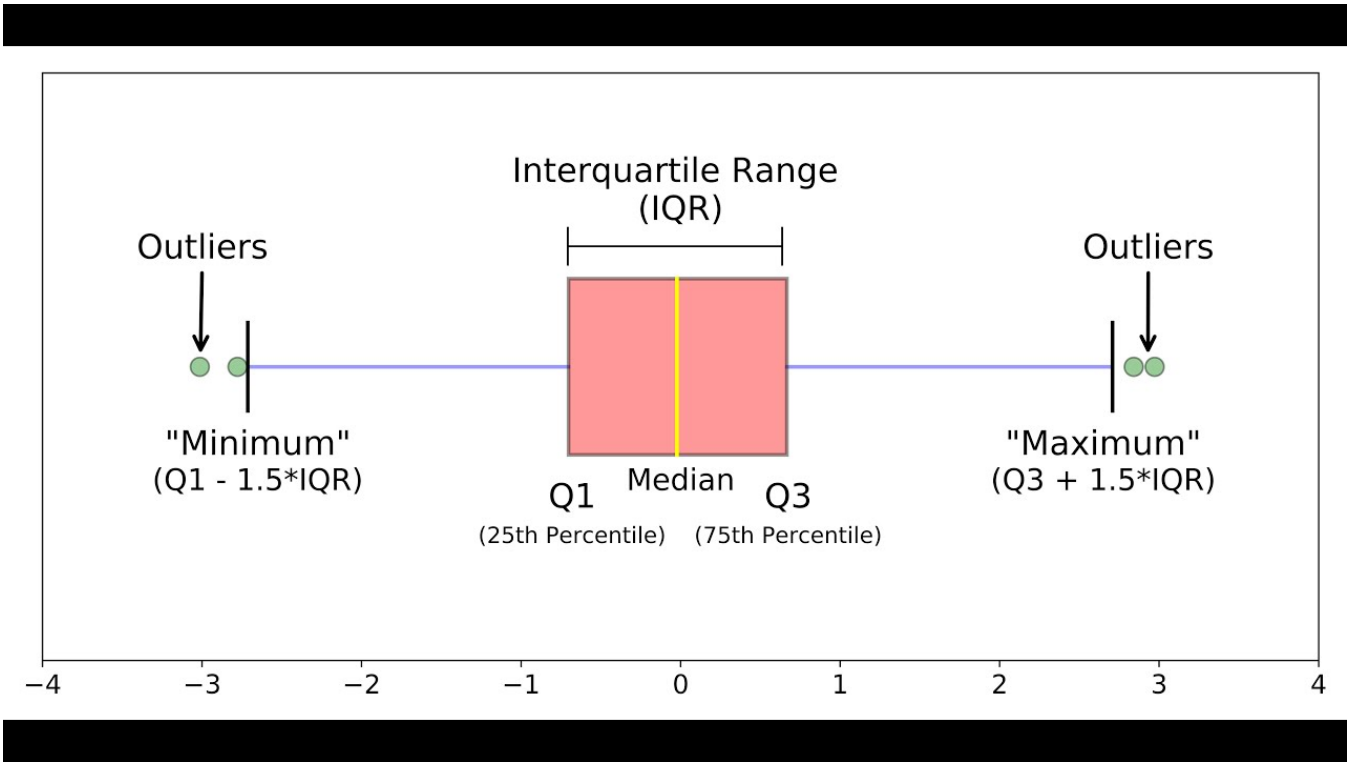
In [27]:

```
# Univariate Plots _ Line, Pie, Bar, Stacked Bar Charts, etc.
# pandas plots - dataframename.variablename.plot(kind="")
# kind=line or pie or bar or bar,stacked=True

# 3 Most Important plots
# 1) Histogram - Bar chart of Frequency Distribution Table. Frequency
# Distribution Table has 2 Columns ClassInterval(LL - UL) & Frequency
# Histogram highlights skewness, kurtosis and outliers

# 2) Boxplot - Based on Quartiles. Q1, Q2, Q3 and InterQuartile Range are
# required for Boxplot. Boxplot highlights skewness and most importantly
# outliers. outliers are extreme values that fall outside normal datarange
# Minimum Side - Q1 - 1.5*IQR
# Maximum Side - Q3 + 1.5*IQR

# 3) Density Curve - Based on Standard Normal Distribution Scores. Line
# plot shows how close data is to normal distribution highlights both
# skewness and kurtosis
```

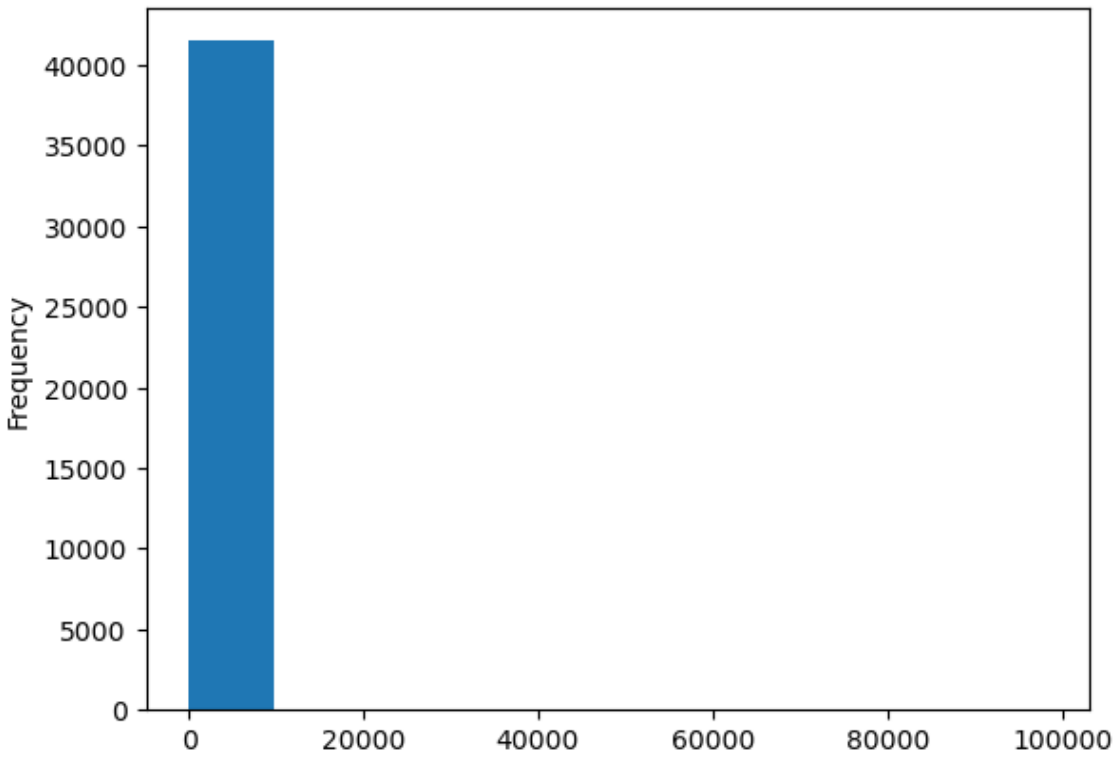


In [28]:

```
NYBnB.price.plot(kind='hist') # Histogram
```

Out[28]:

```
<AxesSubplot:ylabel='Frequency'>
```

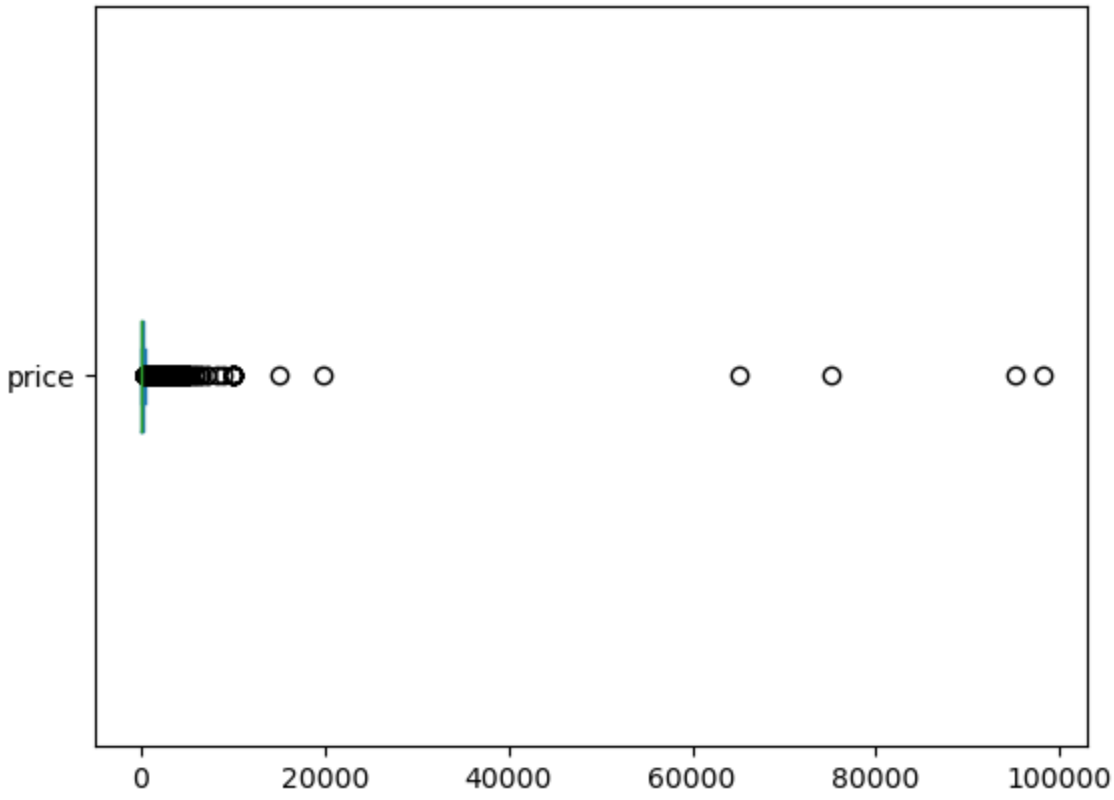


In [29]:

```
NYBnB.price.plot(kind='box',vert=False)
```

Out[29]:

<AxesSubplot:>



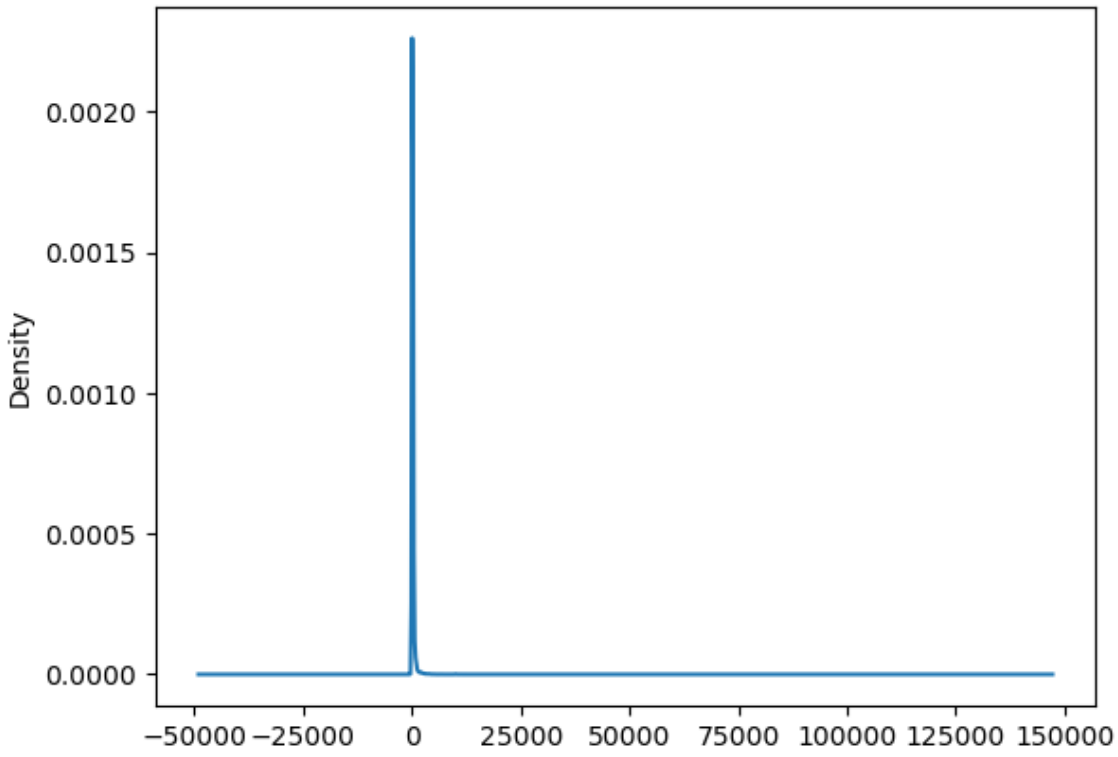
In [30]:

```
NYBnB.price.plot(kind='density')
```

```
<frozen importlib._bootstrap>:228: RuntimeWarning: scipy._lib.messagestrea
m.MessageStream size changed, may indicate binary incompatibility. Expecte
d 56 from C header, got 64 from PyObject
```

Out[30]:

```
<AxesSubplot:ylabel='Density'>
```

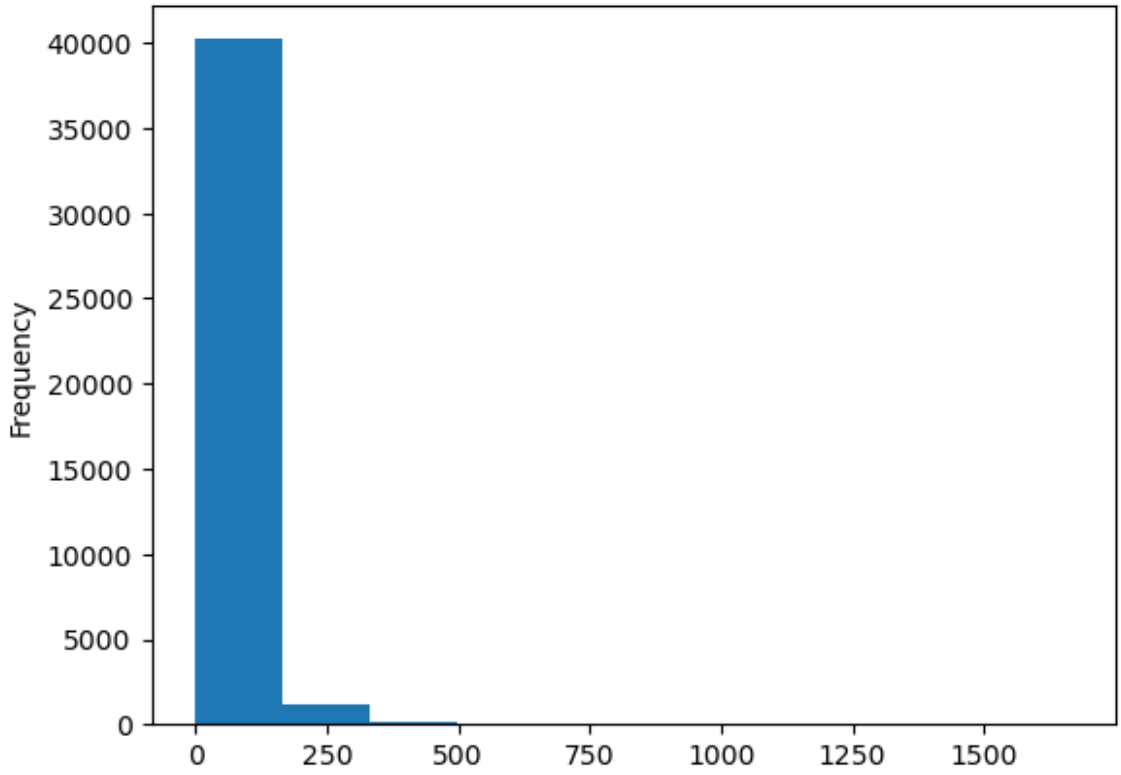


In [31]:

```
NYBnB.number_of_reviews.plot(kind='hist')
```

Out[31]:

```
<AxesSubplot:ylabel='Frequency'>
```

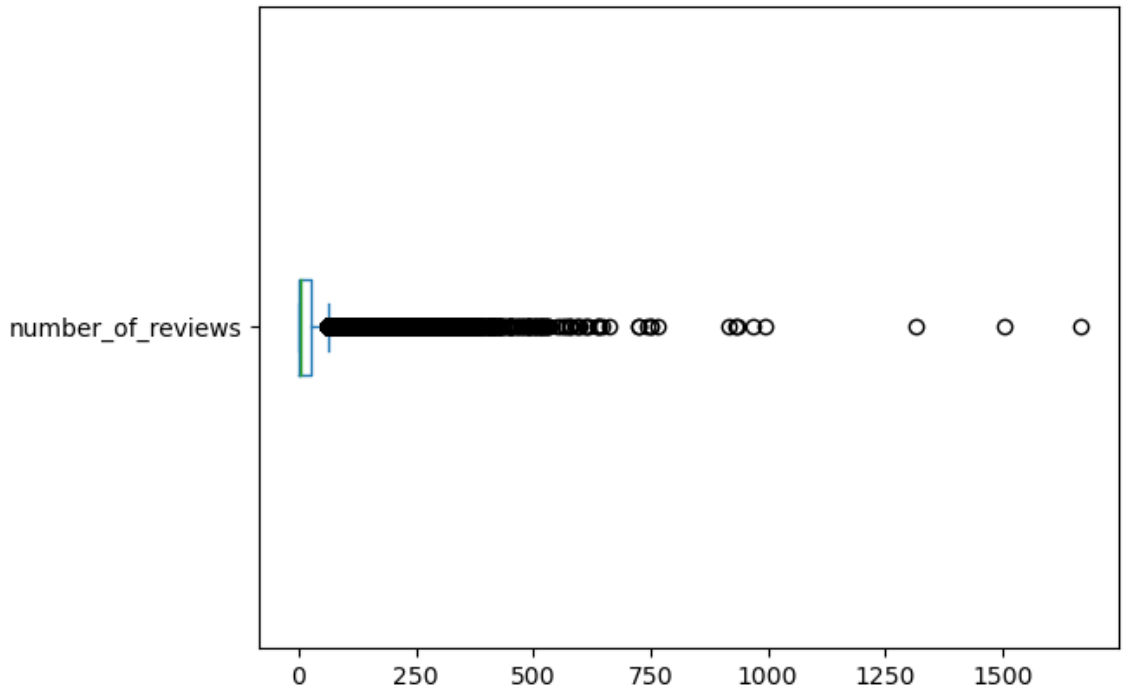


In [32]:

```
NYBnB.number_of_reviews.plot(kind='box',vert=False)
```

Out[32]:

<AxesSubplot:>

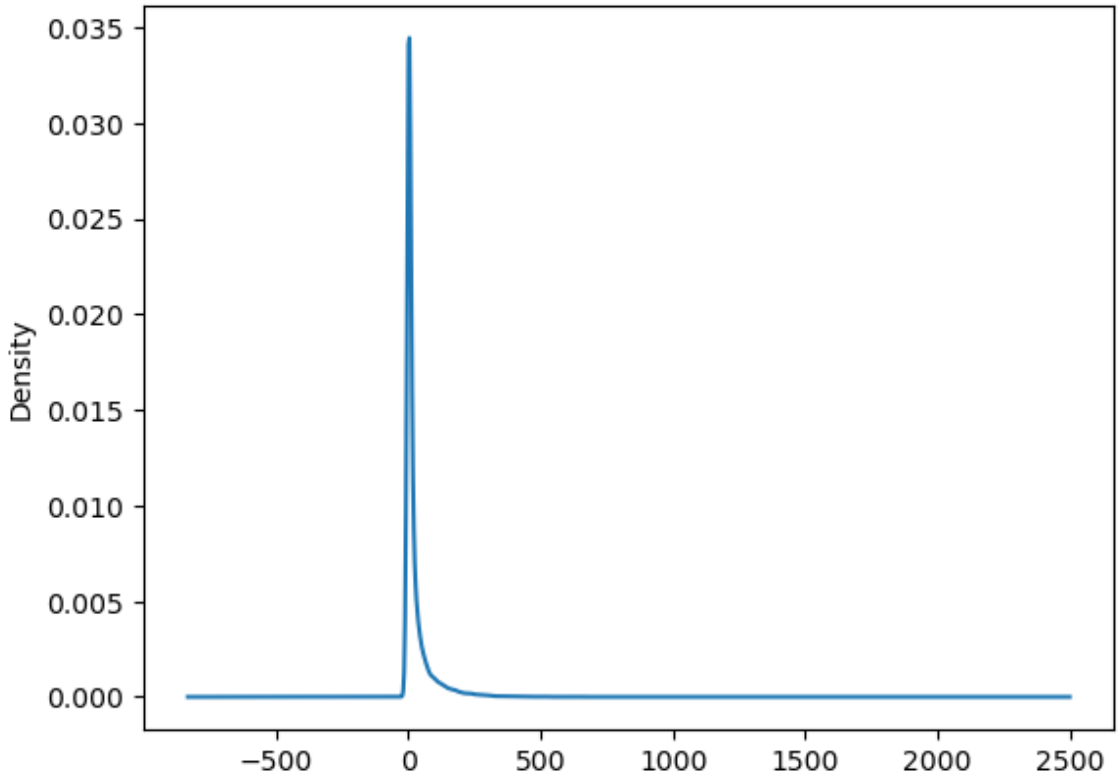


In [33]:

```
NYBnB.number_of_reviews.plot(kind='density')
```

Out[33]:

```
<AxesSubplot:ylabel='Density'>
```



In [34]:

```
# Non Numeric Data - Only Frequency Counts and Cross Tabulations
NYBnB.room_type.value_counts()
```

Out[34]:

```
Entire home/apt    23526
Private room       17287
Shared room          532
Hotel room           188
Name: room_type, dtype: int64
```

In [35]:

```
pd.set_option("display.max_rows",225)
NYBnB.neighbourhood.value_counts()
```

Out[35]:

```
Bedford-Stuyvesant           2936
Williamsburg                 2570
Harlem                       1949
Midtown                      1918
Bushwick                     1752
Hell's Kitchen               1533
Upper West Side              1514
Upper East Side              1428
Crown Heights                1299
East Village                 1171
Chelsea                       889
Lower East Side               847
East Harlem                   821
Greenpoint                    759
Astoria                       727
Washington Heights            634
East Flatbush                 585
Financial District            570
```

In [36]:

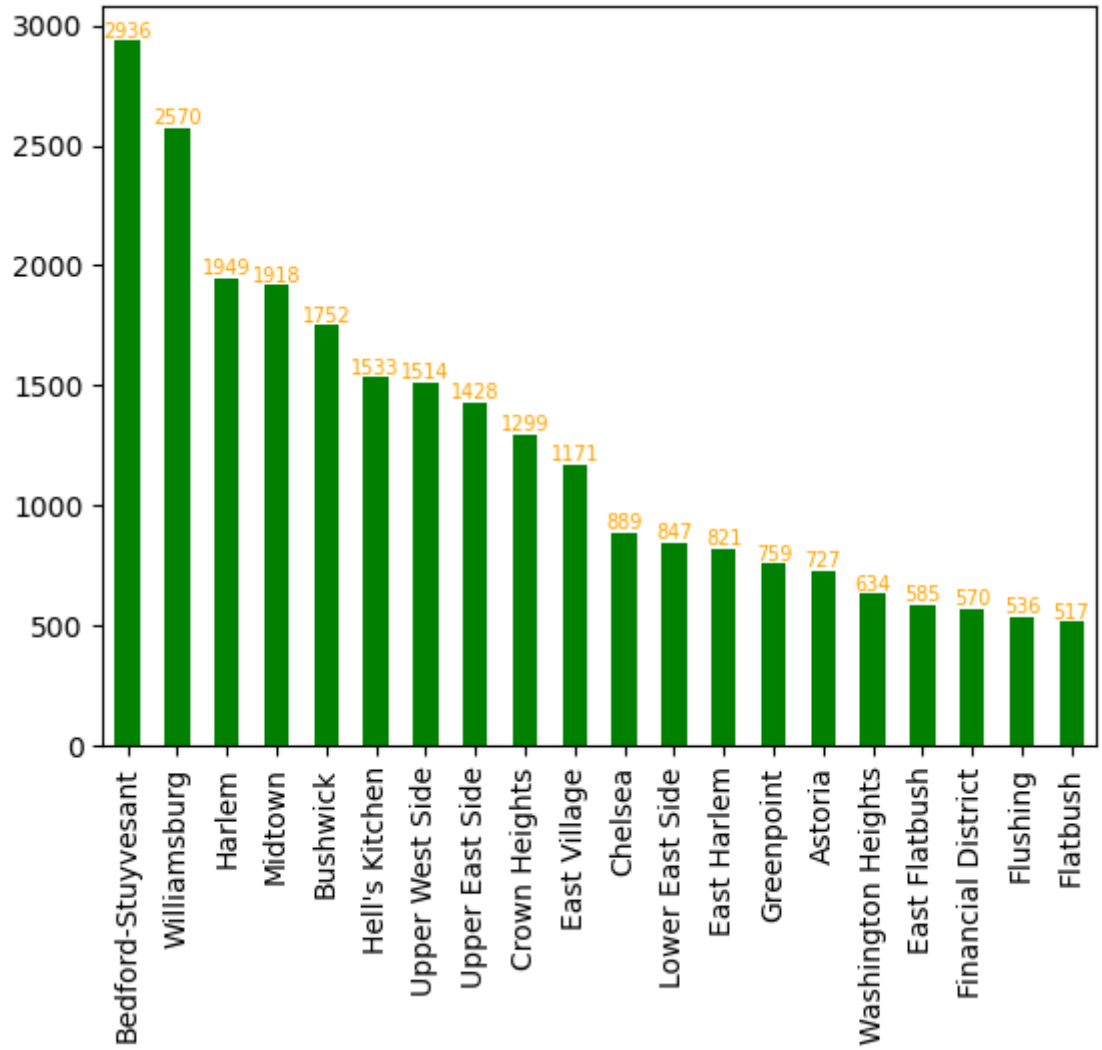
```
# Top 20 neighborhood
NYBnB.neighbourhood.value_counts().nlargest(20)
```

Out[36]:

```
Bedford-Stuyvesant    2936
Williamsburg          2570
Harlem                1949
Midtown               1918
Bushwick              1752
Hell's Kitchen        1533
Upper West Side       1514
Upper East Side       1428
Crown Heights         1299
East Village          1171
Chelsea                889
Lower East Side        847
East Harlem            821
Greenpoint             759
Astoria                727
Washington Heights     634
East Flatbush          585
Financial District     570
Flushing               536
Flatbush               517
Name: neighbourhood, dtype: int64
```

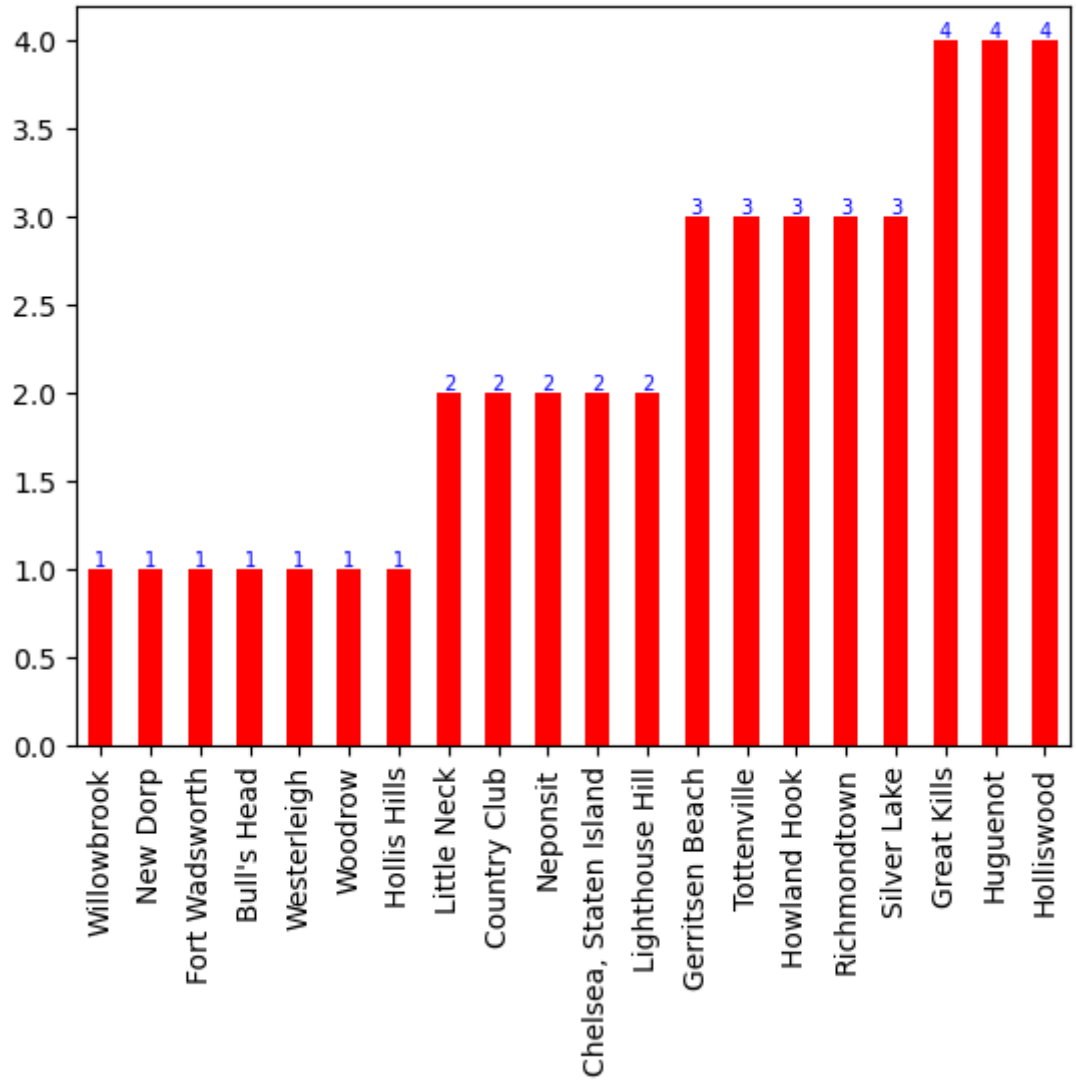
In [37]:

```
df=NYBnB.neighbourhood.value_counts().nlargest(20)
ax=df.plot.bar(color='green')
for i in ax.containers:
    ax.bar_label(i,fontsize=7,color='orange')
```



In [38]:

```
# Bottom 20
df=NYBnB.neighbourhood.value_counts().nsmallest(20)
ax=df.plot.bar(color='red')
for i in ax.containers:
    ax.bar_label(i,fontsize=7,color='blue')
```



In [39]:

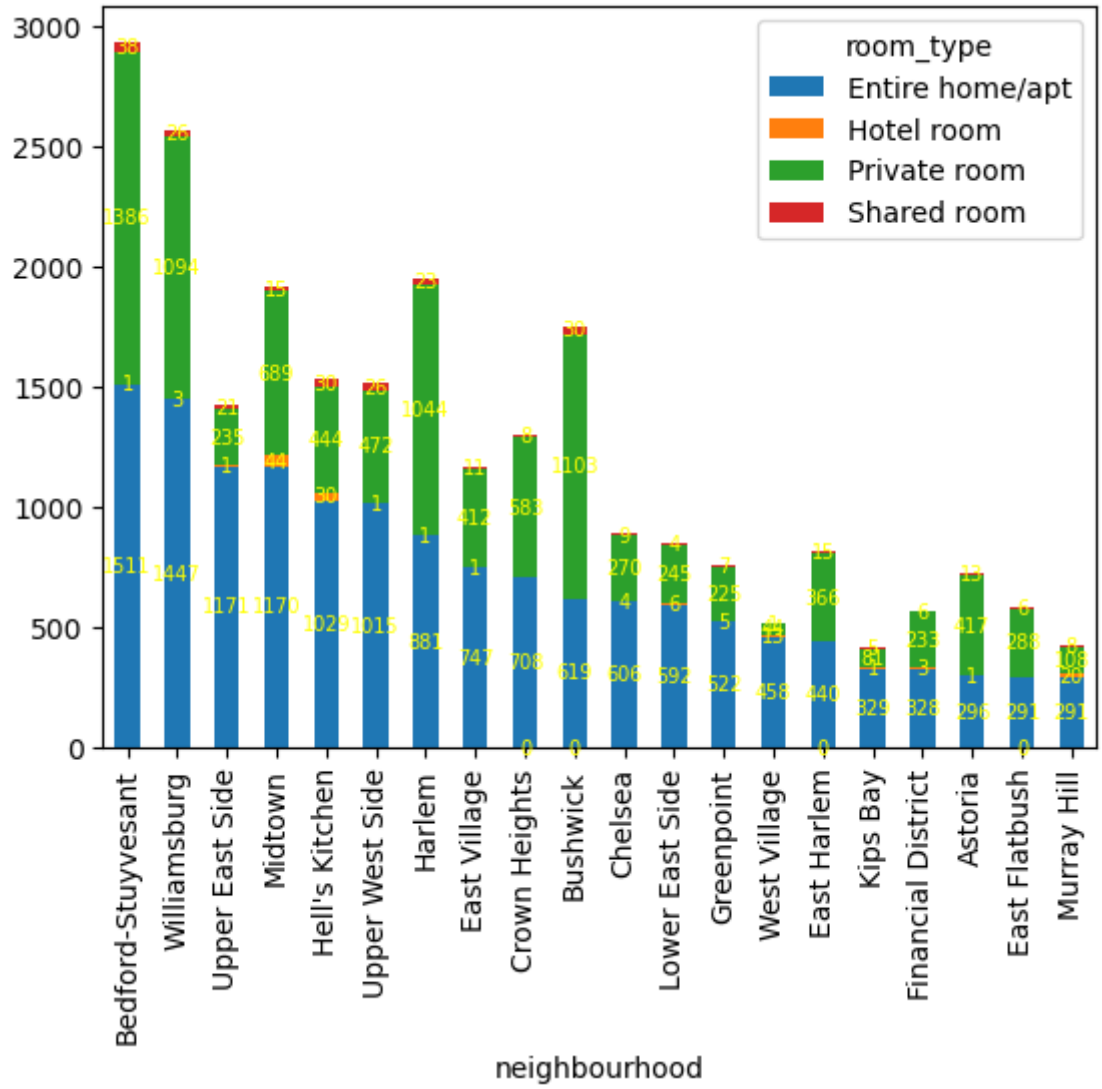
```
# Cross tabulation - Frequency Table of 2 Categorical Variables
pd.crosstab(NYBnB.neighbourhood,
            NYBnB.room_type).nlargest(20,columns="Entire home/apt")
```

Out[39]:

room_type neighbourhood	Entire home/apt	Hotel room	Private room	Shared room
Bedford-Stuyvesant	1511	1	1386	38
Williamsburg	1447	3	1094	26
Upper East Side	1171	1	235	21
Midtown	1170	44	689	15
Hell's Kitchen	1029	30	444	30
Upper West Side	1015	1	472	26
Harlem	881	1	1044	23
East Village	747	1	412	11
Crown Heights	708	0	583	8
Bushwick	619	0	1103	30
Chelsea	606	4	270	9
Lower East Side	592	6	245	4
Greenpoint	522	5	225	7
West Village	458	13	44	1
East Harlem	440	0	366	15
Kips Bay	329	1	81	5
Financial District	328	3	233	6
Astoria	296	1	417	13
East Flatbush	291	0	288	6
Murray Hill	291	20	108	8

In [40]:

```
df=pd.crosstab(NYBnB.neighbourhood,
               NYBnB.room_type).nlargest(20,columns="Entire home/apt")
ax=df.plot.bar(stacked=True)
for i in ax.containers:
    ax.bar_label(i,fontsize=7,color='yellow',label_type="center")
```



In [41]:

```
d1=pd.crosstab(NYBnB.neighbourhood,NYBnB.room_type)
```

In [48]:

```
d1[(d1['Entire home/apt']>=500)&(d1['Entire home/apt']<=1000)]
```

Out[48]:

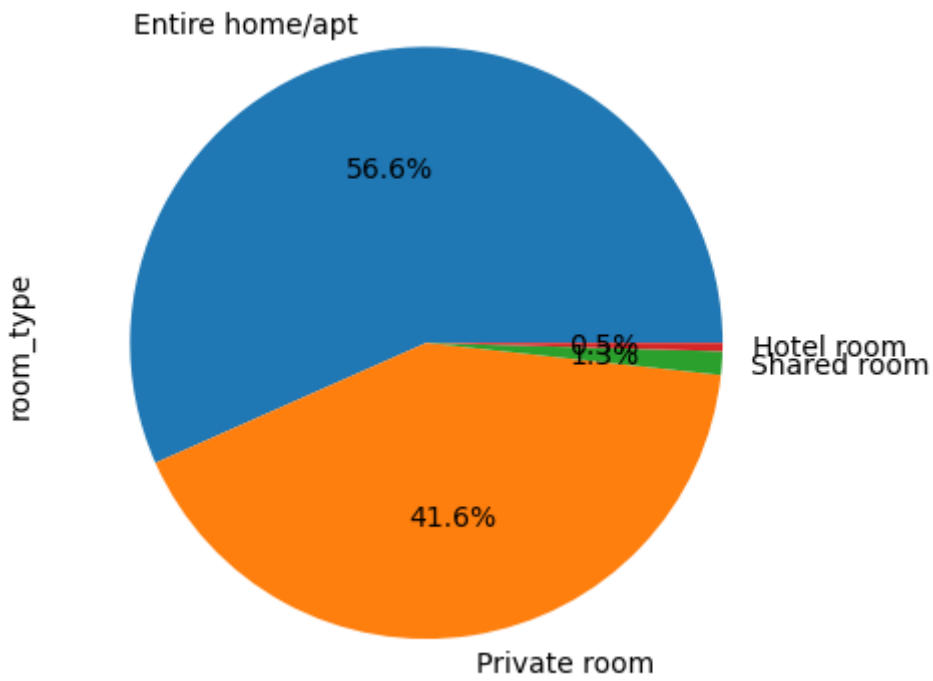
room_type neighbourhood	Entire home/apt	Hotel room	Private room	Shared room
Bushwick	619	0	1103	30
Chelsea	606	4	270	9
Crown Heights	708	0	583	8
East Village	747	1	412	11
Greenpoint	522	5	225	7
Harlem	881	1	1044	23
Lower East Side	592	6	245	4

In [60]:

```
NYBnB.room_type.value_counts().plot(kind='pie',autopct='%.1f%%')
# %.2f%% is for 2 decimal with % symbol
# %.1f%% is for 1 decimal with % symbol
```

Out[60]:

```
<AxesSubplot:ylabel='room_type'>
```



In [64]:

```
# Data Aggregation - groupby() function
NYBnB.price.groupby(NYBnB.room_type).mean()
# Left side of groupby must be numerical
# Right side of groupby within paranthesis categorical variable
# Statistical Function like mean, median,std,sum, etc. must be speciifed.
```

Out[64]:

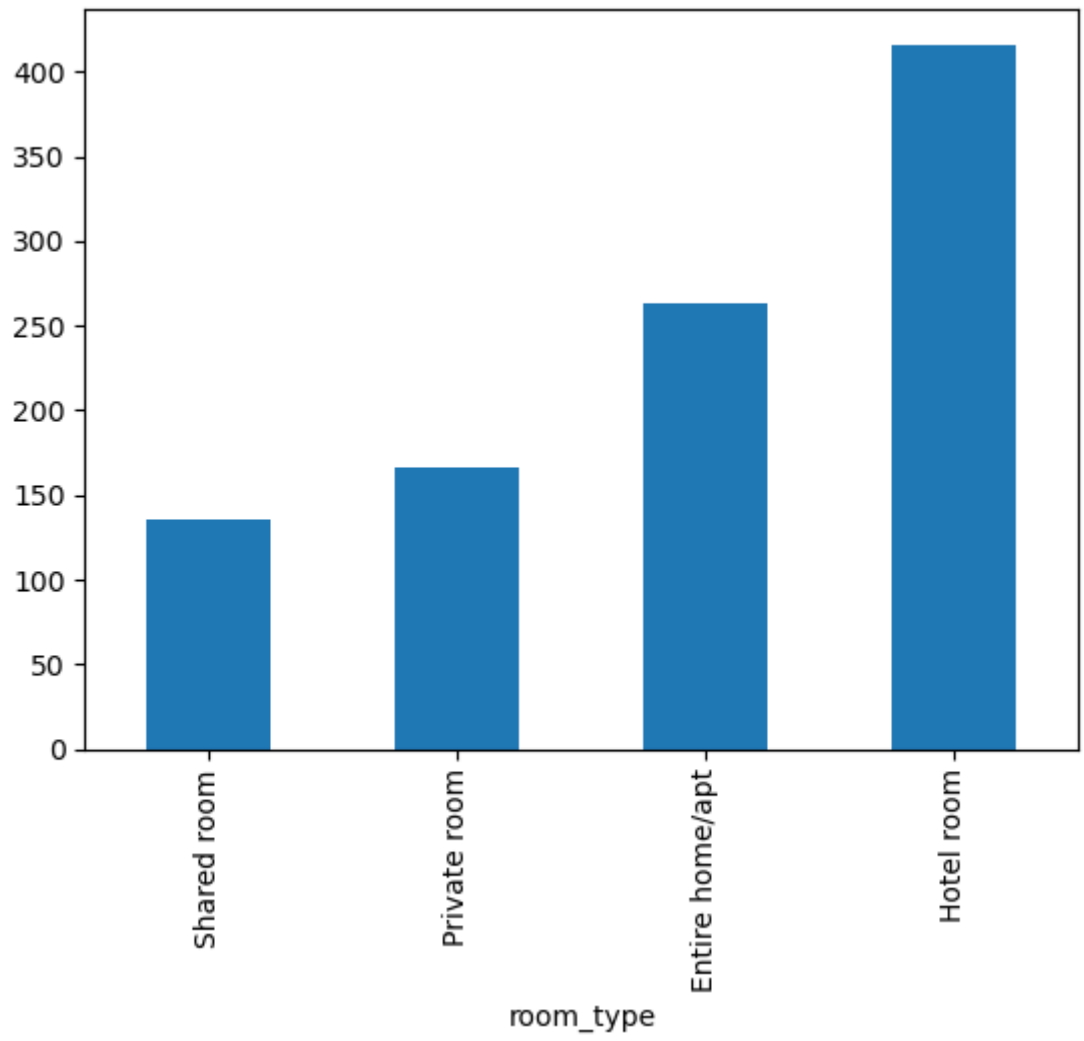
```
room_type
Entire home/apt    263.442404
Hotel room         416.164894
Private room       166.102042
Shared room        135.400376
Name: price, dtype: float64
```

In [69]:

```
NYBnB.price.groupby(NYBnB.room_type).mean().sort_values().plot(kind='bar')
```

Out[69]:

```
<AxesSubplot:xlabel='room_type'>
```



In [72]:

```
NYBnB.price.groupby(NYBnB.neighbourhood).mean().sort_values(
    ascending=False)
```

```
Middle Village                157.206897
Schuylerville                 154.777778
Harlem                        154.136993
Oakwood                       150.857143
Breezy Point                  150.000000
Crown Heights                 149.278676
Mill Basin                    147.785714
Whitestone                    146.750000
Flatlands                     146.339286
Little Neck                   146.000000
Fresh Meadows                 145.941176
Queens Village                145.670213
South Ozone Park              145.581395
Two Bridges                   145.245614
Jamaica                       145.179348
Unionport                     143.882353
Midland Beach                 143.400000
Brownsville                   142.637097
Roosevelt Island              142.333333
City Island                   141.928571
```

In [76]:

```
# What is the Average Number of reviews for each room type
np.round(NYBnB.number_of_reviews.groupby(NYBnB.room_type).mean(),2)
# np.round(output,decimal)
```

Out[76]:

```
room_type
Entire home/apt    27.05
Hotel room         55.65
Private room       24.93
Shared room        19.97
Name: number_of_reviews, dtype: float64
```

In [75]:

```
# What is the Median price for each room type
NYBnB.price.groupby(NYBnB.room_type).median()
```

Out[75]:

```
room_type
Entire home/apt    180.0
Hotel room         335.0
Private room        78.0
Shared room         65.0
Name: price, dtype: float64
```

In [ ]:

```

```