Python EDA

In [ ]:

```
# Exploratory Data Analysis (EDA)
# Exploratory Data Analysis (EDA) is an approach to analyze the data using visual techni
# It is used to discover trends,patterns,or to check assumptions with the help of statis
# summary and graphical representations.

# Three concepts -
#                 Descriptive Statistics,
#                 Data Visualization,
#                 Data Aggregation/wrangling

# For Numeric data:
# Univariate Statistics - Single variable stats
#                        - The term univariate refers to the analysis of one variable

# Descriptive Statistics - Description of data - Numeric data

# Measures of Central Tendency(Mid point) - Mean, Median and mode

# Measures of Dispersion(Scattering of observations aroung Mean) - Variance, Standard De

# Mesures of Asymmetry(how close data is to normal distribution/Bell curve)
# Skewness and Kurtosis

# Measures of locations - Observations location in terms of quartiles,Percentiles,
#                          Deciles(Multiple of 10).

# Non Numeric data
# Frequency Counts, Cross Tabulation ( frequency table of 2 Non Numeric Varibales)
```

In [1]:

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

In [2]:

```
df = pd.read_csv('D:\Datasets\listings.csv')
```

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7747 entries, 0 to 7746
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   id                              7747 non-null   int64  
 1   name                            7747 non-null   object 
 2   host_id                         7747 non-null   int64  
 3   host_name                       7747 non-null   object 
 4   neighbourhood_group             0 non-null      float64
 5   neighbourhood                   7747 non-null   object 
 6   latitude                        7747 non-null   float64
 7   longitude                       7747 non-null   float64
 8   room_type                       7747 non-null   object 
 9   price                           7747 non-null   int64  
 10  minimum_nights                  7747 non-null   int64  
 11  number_of_reviews               7747 non-null   int64  
 12  last_review                     6254 non-null   object 
 13  reviews_per_month               6254 non-null   float64
 14  calculated_host_listings_count  7747 non-null   int64  
 15  availability_365                7747 non-null   int64  
 16  number_of_reviews_ltm           7747 non-null   int64  
 17  license                         6573 non-null   object 
dtypes: float64(4), int64(8), object(6)
memory usage: 1.1+ MB
```

In [5]:

```
# indexing of accesing specific columns or rows of a dataframe
# Two types of indexing - Character indexing or indexing by column/variable name
# Column/Variable Name must exactly match

# 1)dataframename['column']  or dataframename["column"]
# 2)datatframename.columnname

# if variable name has space dataframename['column']  must be used.
```

In [6]:

```
df['price'].head() # head() - By default first 5 rows
```

Out[6]:

```
0     90
1     65
2    198
3     85
4    202
Name: price, dtype: int64
```

In [7]:

```
# datatframename.columname
df.price.head()
```

Out[7]:

```
0     90
1     65
2    198
3     85
4    202
Name: price, dtype: int64
```

In [8]:

```
# Character indexing - Multiple Columns or Varibles
# dataframename[['col1','col2','cal3']]
```

In [9]:

```
df[['price','minimum_nights','number_of_reviews']].head()
```

Out[9]:

	price	minimum_nights	number_of_reviews
0	90	3	212
1	65	7	7
2	198	2	59
3	85	2	483
4	202	3	598

In [10]:

```
df.columns
```

Out[10]:

```
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 [11]:

```
# Numeric Indexing or Indexing by Column/variable Number. Python strats from 0.
```

In [12]:

```
df.iloc[:,9].head()
# :, before indicates colun selection. if :, not given indicates row selection
```

Out[12]:

```
0     90
1     65
2    198
3     85
4    202
Name: price, dtype: int64
```

In [13]:

```
# Numeric indsing Multicolumns
df.iloc[:,[5,9,10,11,12]].head() # Column indexing
```

Out[13]:

	neighbourhood	price	minimum_nights	number_of_reviews	last_review
0	Hyde Park	90	3	212	2023-03-04
1	Edgewater	65	7	7	2022-12-01
2	North Center	198	2	59	2022-12-31
3	West Town	85	2	483	2023-02-26
4	Logan Square	202	3	598	2023-02-25

In [14]:

```
df.iloc[[5,9,10,11,12]].head() # Row indexing
```

Out[14]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitu
5	207218	Historic Pullman Artist Flat - Artists & Explo...	1019125	Jb	NaN	Pullman	41.688
9	1773021	4 Bedroom Across from Wrigley Field Stadium Suite	9297431	Inn At Wrigleyville	NaN	Lake View	41.947
10	220333	Pullman School House Apartment - monthly rental	1019125	Jb	NaN	Pullman	41.689
11	12140	Lincoln Park Guest House	46734	Sharon And Robert	NaN	Lincoln Park	41.923
12	233933	Lovely Bedroom 3 in a New Renovated Apartment	1224828	Tippi	NaN	Irving Park	41.957

In [15]:

```
df.describe()
# describe() - Count of N, Min, Max, Mean, Stddev, Q1 ,Q2, median, Q3

# Mean and Median must be close to one another +/- 10%. if there is big difference betwe
# mean is distorted by small or large values.

# If mean is distorted stddev is very high
```

Out[15]:

	id	host_id	neighbourhood_group	latitude	longitude	
count	7.747000e+03	7.747000e+03	0.0	7747.000000	7747.000000	7747.0
mean	2.650872e+17	1.617357e+08	NaN	41.895250	-87.662637	184.2
std	3.448603e+17	1.526951e+08	NaN	0.061759	0.043208	1160.0
min	2.384000e+03	2.153000e+03	NaN	41.650640	-87.847243	0.0
25%	3.094478e+07	3.288698e+07	NaN	41.867725	-87.686305	77.0
50%	4.973334e+07	1.074344e+08	NaN	41.898470	-87.657760	124.0
75%	6.629074e+17	2.574644e+08	NaN	41.938337	-87.631890	189.0
max	8.495391e+17	5.056757e+08	NaN	42.022200	-87.529541	99998.0

In [16]:

```
df.price.describe()
```

Out[16]:

```
count     7747.000000
mean       184.285917
std       1160.005899
min          0.000000
25%         77.000000
50%        124.000000
75%        189.000000
max      99998.000000
Name: price, dtype: float64
```

In [17]:

```
df.number_of_reviews.describe()
```

Out[17]:

```
count    7747.000000
mean       45.938815
std        86.832868
min         0.000000
25%         1.000000
50%        13.000000
75%        54.000000
max      3091.000000
Name: number_of_reviews, dtype: float64
```

In [18]:

```
# in case of Asymmetry
# Skewness - Positive Skewness indicates peak of curve on left side
# Skewness = o - Normal Distribution
# Skewness - Nevative skewness indicates peak of curve on right side

# Kurtosis - >3 indicates Tall and Narrow Peak
# Kurtosis - =3 indicates Normal Distibution
# Kurtosis - <3 indicates Wide and Broad peak
```

In [19]:

```
print("Skewness is:",df.price.skew())
print("Kurtosis is:",df.price.kurt())
```

```
Skewness is: 82.41216412031832
Kurtosis is: 7080.294967055464
```

In [20]:

```
print("Skewness is:",df.number_of_reviews.skew())
print("Kurtosis is:",df.number_of_reviews.kurt())
```

```
Skewness is: 8.337482754590493
Kurtosis is: 209.3503187270444
```

In [21]:

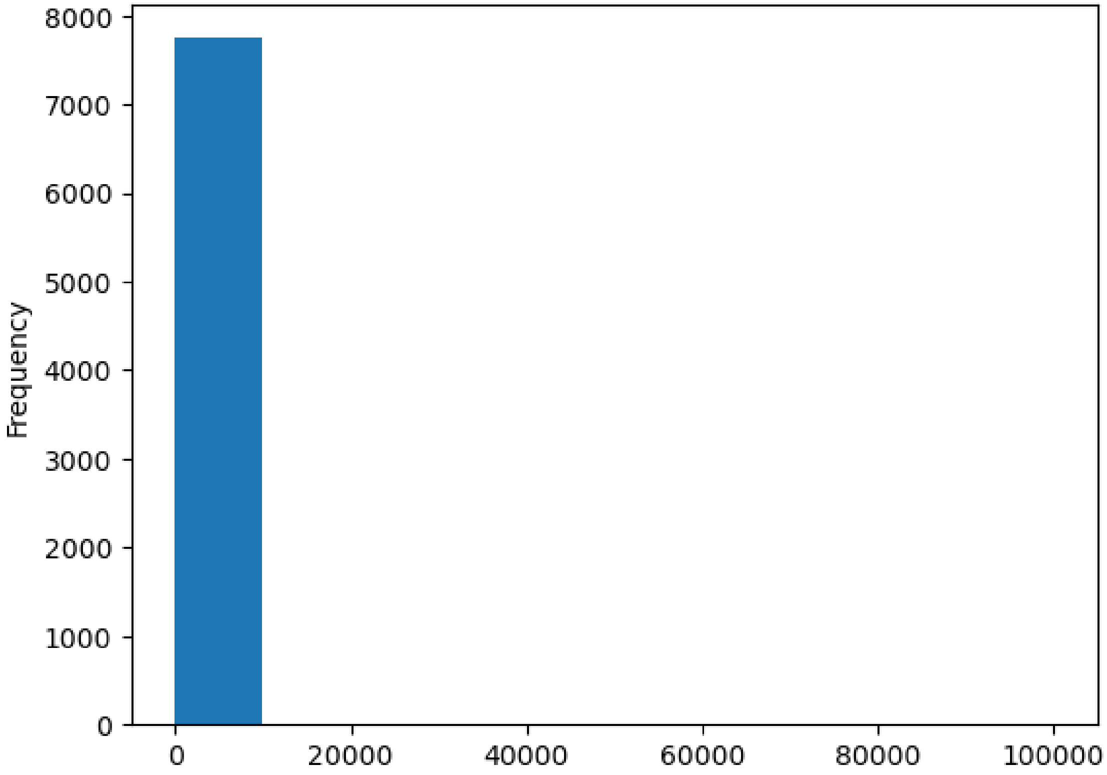
```
# EDA -  Data Vizualizations -  matplotlib.pyplot, seaborn
# Plots like line,Bar,Pie,Stacked Plot, etc.

# 3 Most important plots in ML are

# 1) Histogram - Based on frequency Distribution Table. Class interval(lower limit-upper
#    Bar plot of frequency distribution table is Histogram. Histogram indicates skewness

# 2) Boxplot - Based on Quartiles. Q1, Q2 or Median, Q3 and Inter Quartile range(Q3-Q1)
#    Boxplot identifies skewness and most importantly outliers. Outliers are extreme val
#    Outliers are identified using formula
#    Minimum Side - Q1 - 1.5 * IQR
#    Maximum Side - Q3 + 1.5 * IQR

# 3) Density Curve or Kernal Density Curve - Based on standard Normal Disribution scores
```

In [22]:

```
# pandas - plot() - kind = 'line','bar','pie','hist','box','density'

df.price.plot(kind='hist')
```

Out[22]:

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

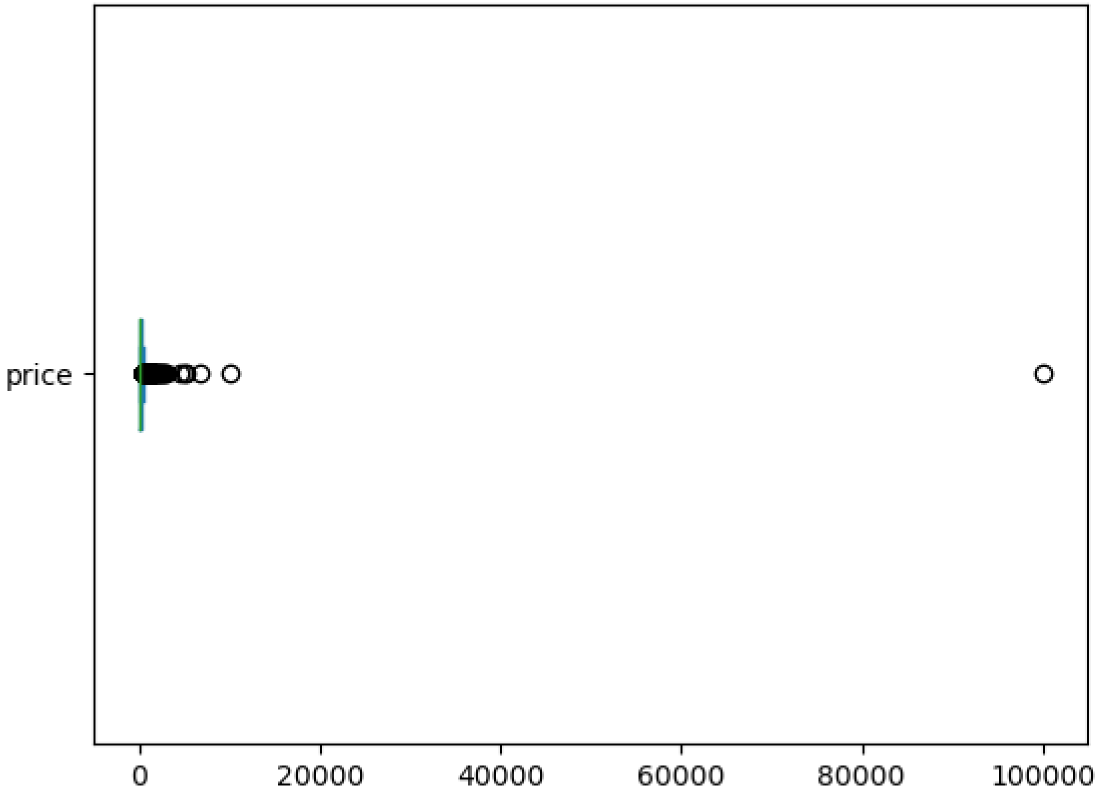


In [23]:

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

Out[23]:

```
<Axes: >
```

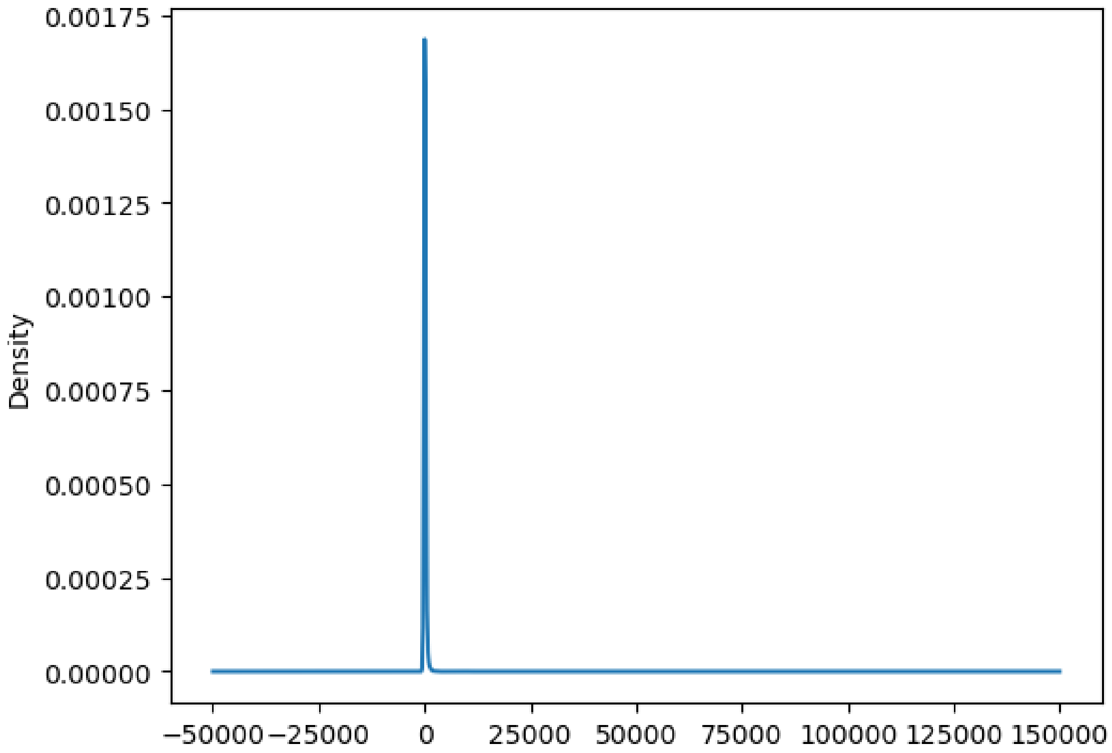


In [24]:

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

Out[24]:

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

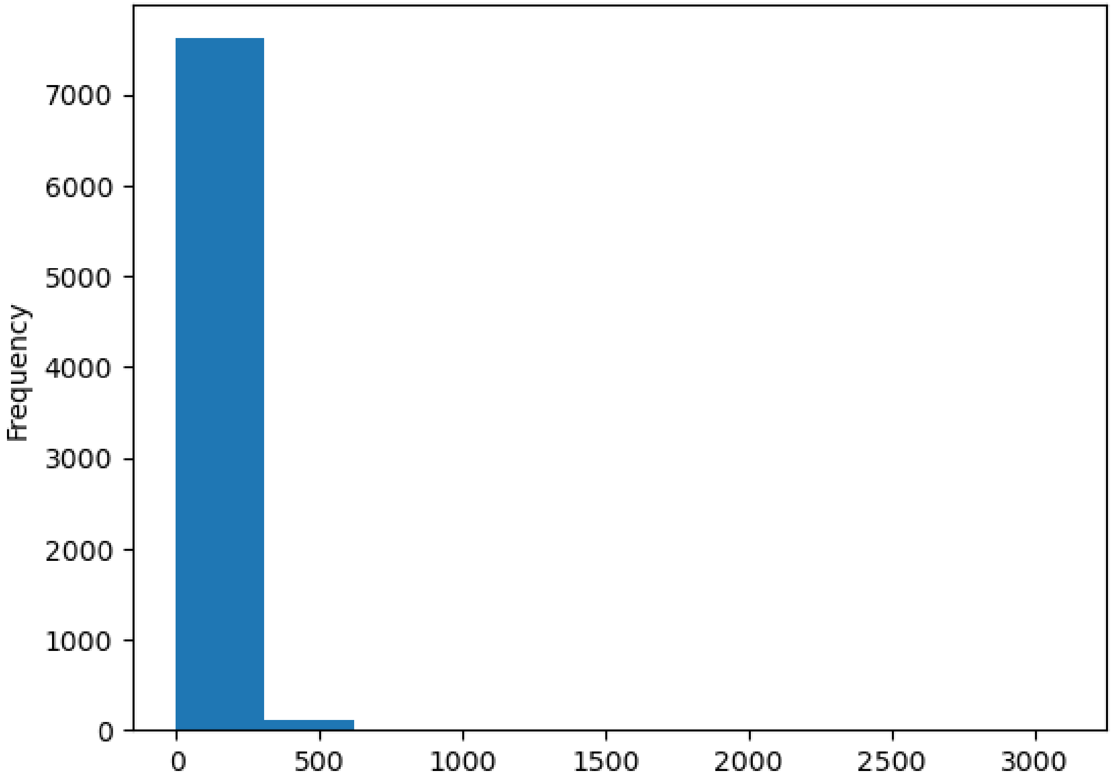


In [25]:

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

Out[25]:

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

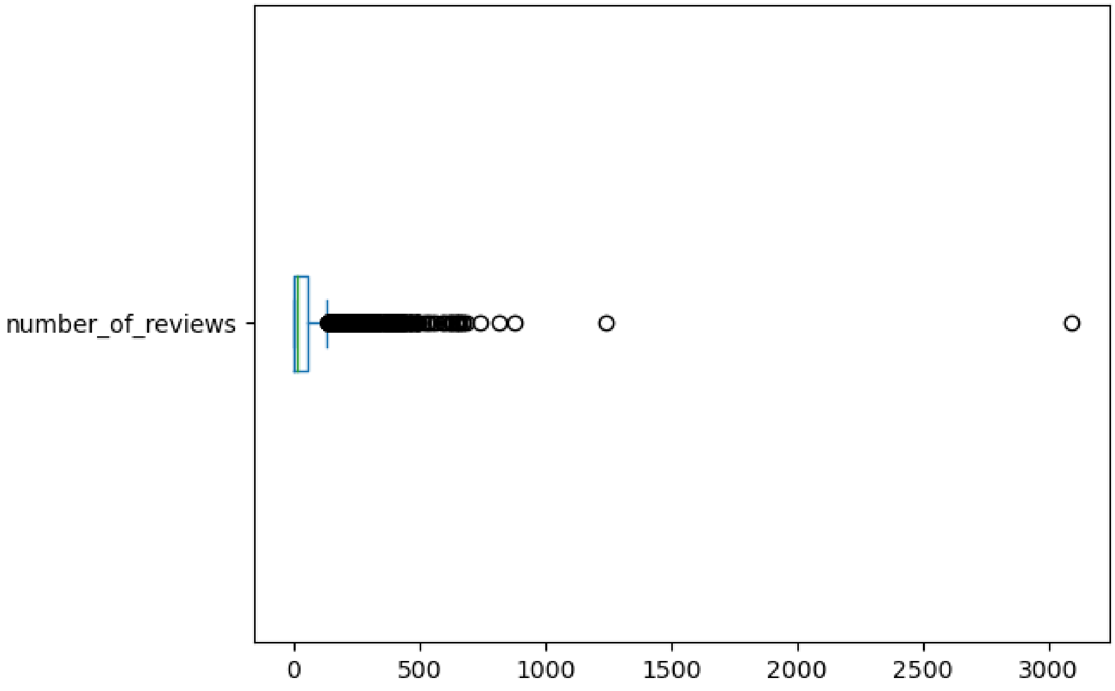


In [26]:

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

Out[26]:

```
<Axes: >
```

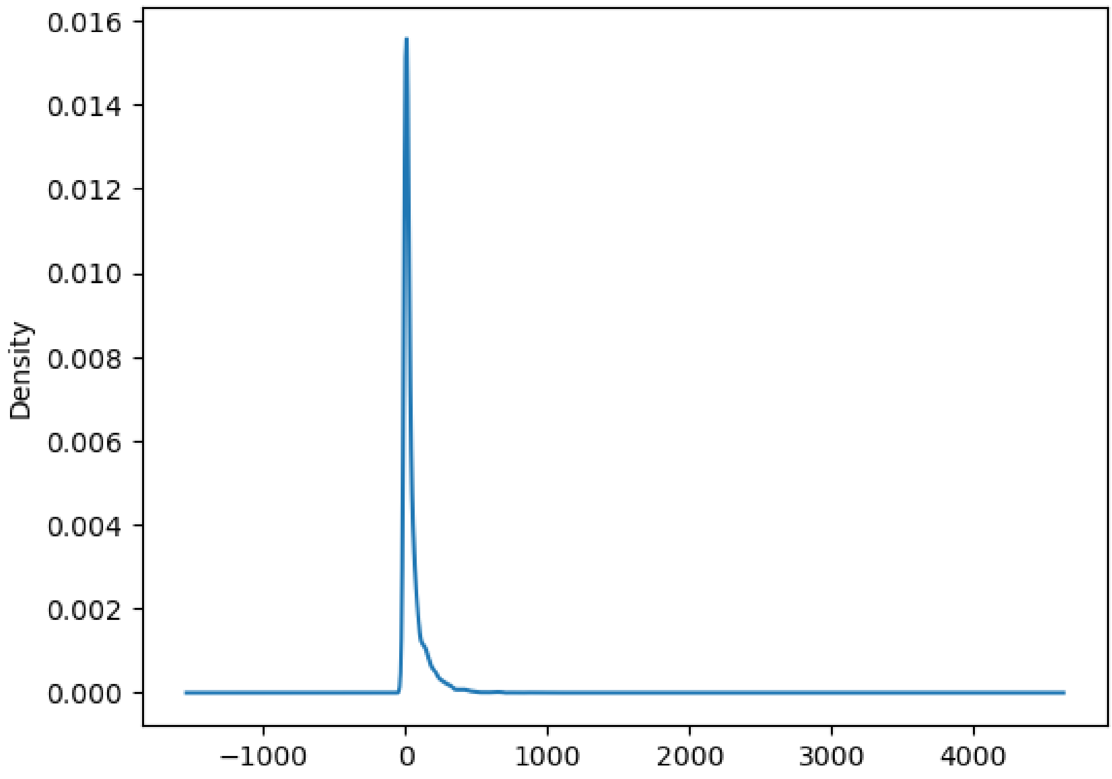


In [27]:

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

Out[27]:

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



In [28]:

```
# Non Numeric Data - Frequency Counts

df.room_type.value_counts()
```

Out[28]:

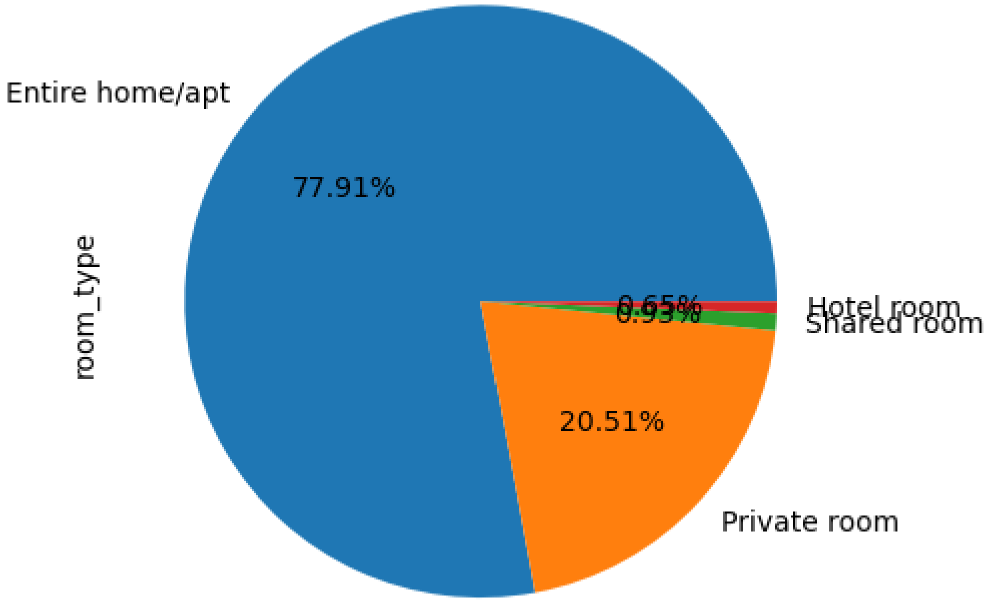
```
Entire home/apt    6036
Private room       1589
Shared room          72
Hotel room           50
Name: room_type, dtype: int64
```

In [29]:

```
df.room_type.value_counts().plot(kind='pie',autopct="%.2f%%")
```

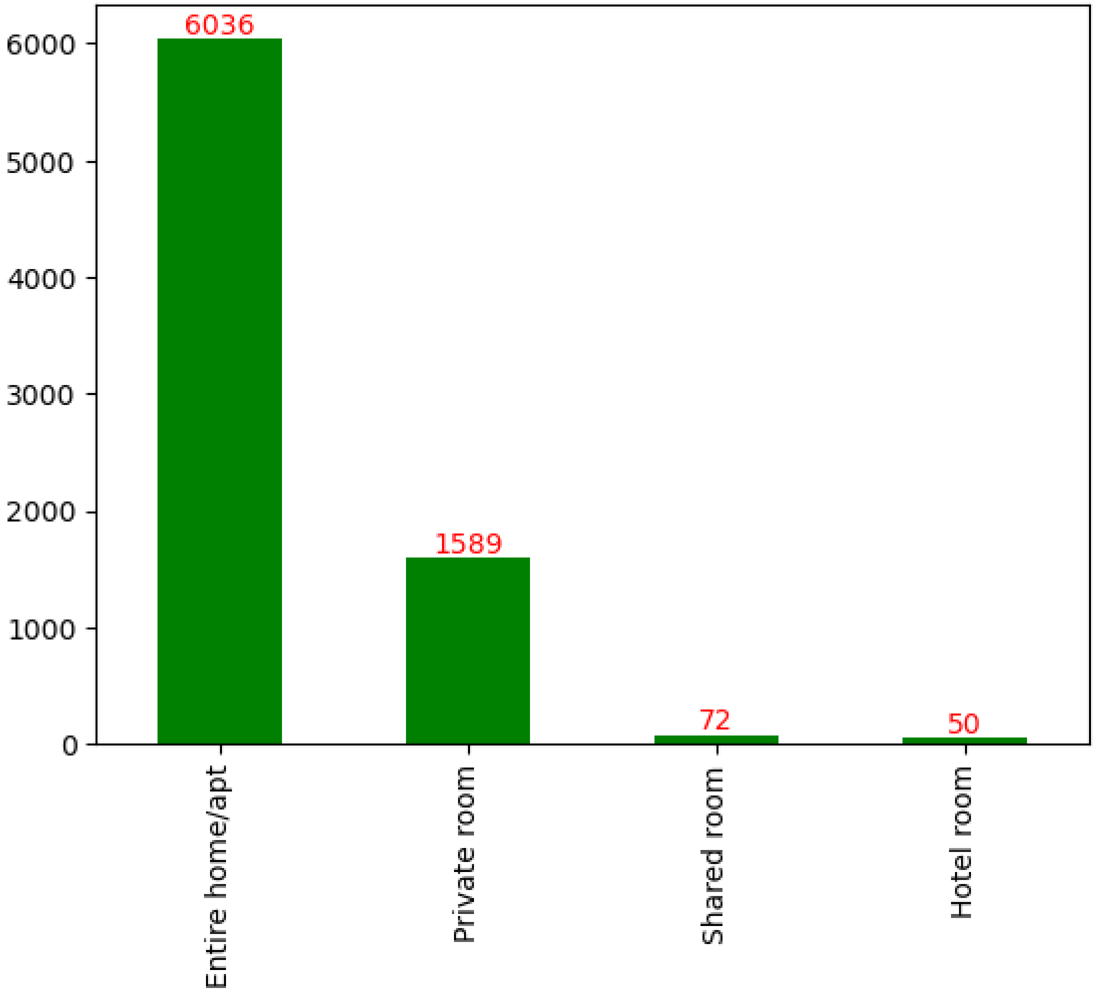
Out[29]:

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



In [30]:

```
ax = df.room_type.value_counts().plot(kind='bar',color = 'green')
for i in ax.containers:
    ax.bar_label(i,color='red')
```



In [31]:

```
df.columns
```

Out[31]:

```
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 [32]:

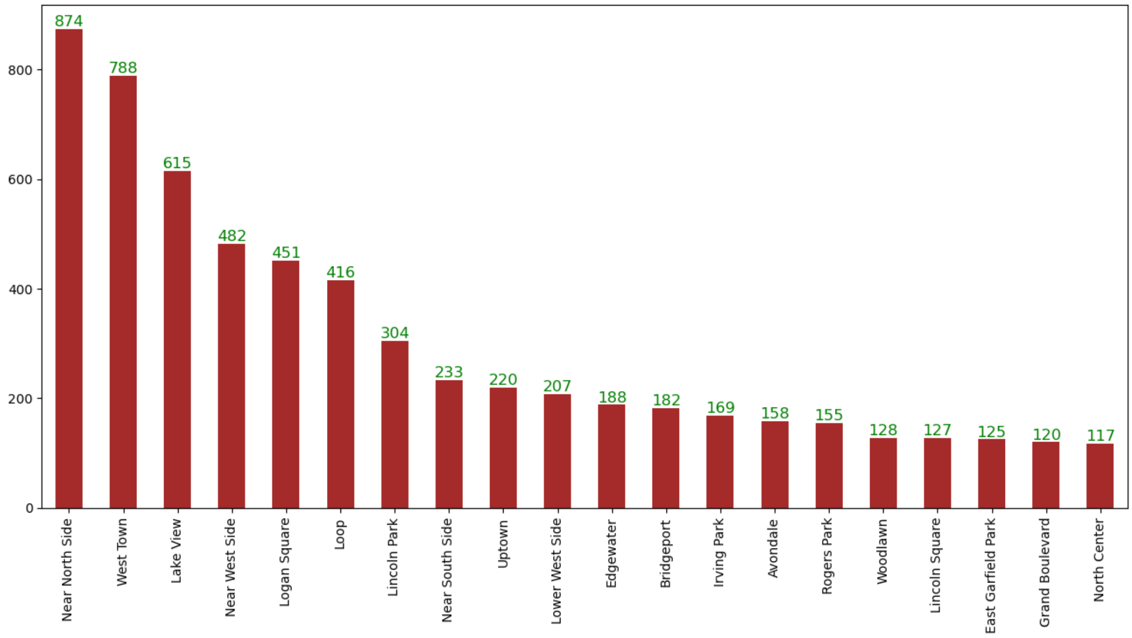
```
pd.set_option("display.max_rows",7)
df.neighbourhood.value_counts()
```

Out[32]:

```
Near North Side    874
West Town          788
Lake View          615
                  ...
Mount Greenwood      2
Edison Park          1
Burnside             1
Name: neighbourhood, Length: 76, dtype: int64
```

In [33]:

```
plt.figure(figsize=(15,7))
ax=df.neighbourhood.value_counts().nlargest(20).plot(kind='bar',color='brown')
for i in ax.containers:
    ax.bar_label(i,fontsize=12,color='green')
```



In [34]:

```
# Both Numeric and Non Numeric - groupby() function is used
# Left side of group by must be numerical
# Right side of group by within brackets Non Numeric
# Statistical Fucntion like, sum, mean,median,sd,etc must be specified.
```

In [35]:

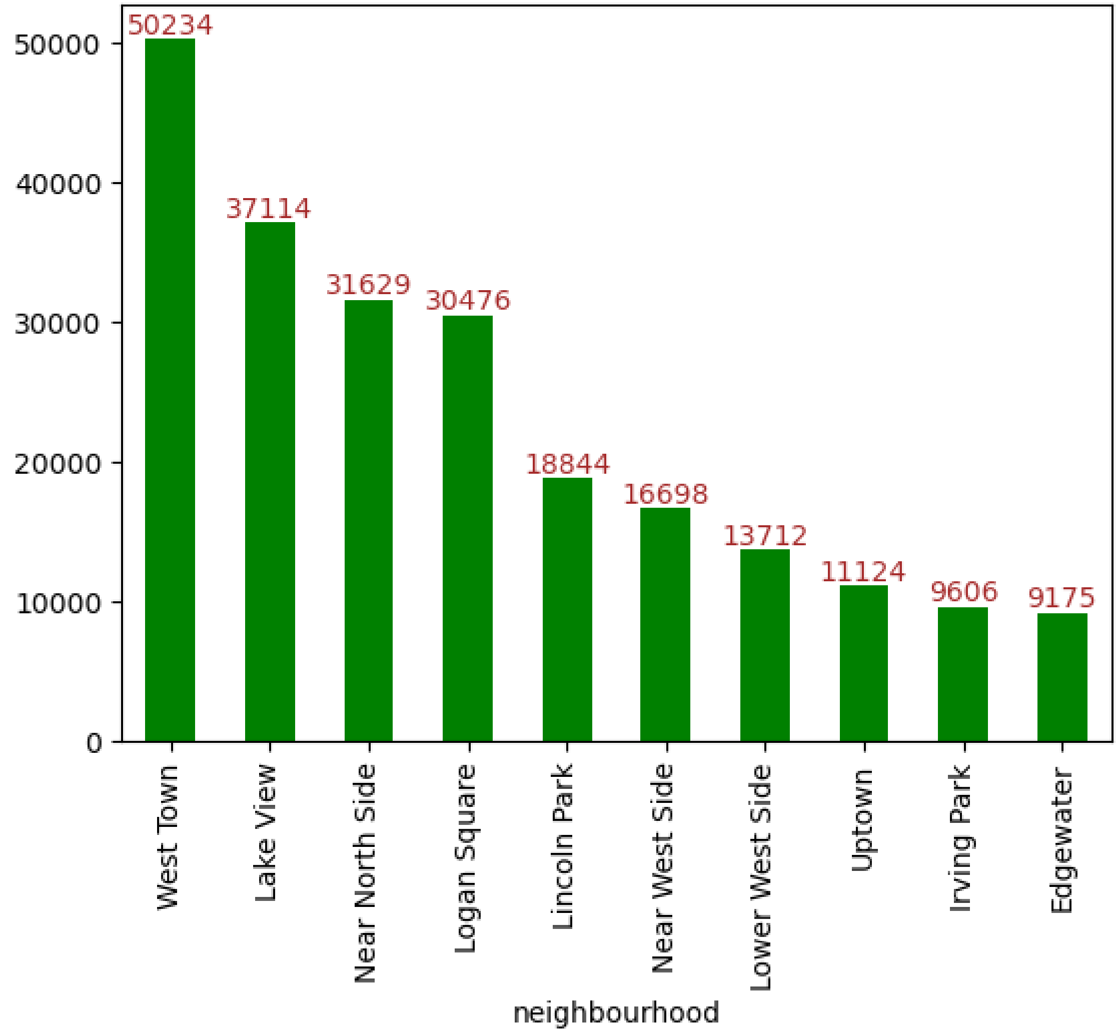
```
# Total number_of_reviews for Top 20 neighborhood

df.number_of_reviews.groupby(df.neighbourhood).sum().sort_values(ascending = False).nlar
```

Out[35]:

```
neighbourhood
West Town           50234
Lake View           37114
Near North Side     31629
                    ...
Uptown              11124
Irving Park          9606
Edgewater            9175
Name: number_of_reviews, Length: 10, dtype: int64
```

In [36]:

```
ax = df.number_of_reviews.groupby(df.neighbourhood).sum().sort_values(ascending = False)
for i in ax.containers:
    ax.bar_label(i,color='brown')
```



In [37]:

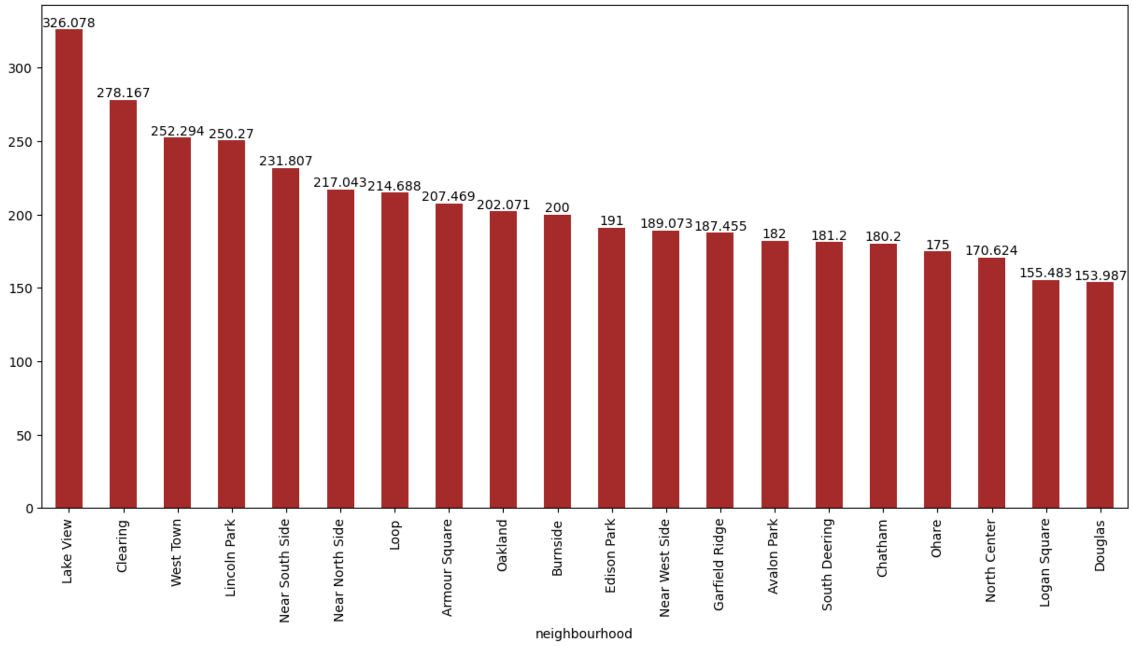
```
# Average price of top 20 neighbourhood
df.price.groupby(df.neighbourhood).mean().sort_values(ascending=False).nlargest(20)
```

Out[37]:

```
neighbourhood
Lake View        326.078049
Clearing         278.166667
West Town        252.294416
                    ...
North Center     170.623932
Logan Square     155.483370
Douglas          153.987013
Name: price, Length: 20, dtype: float64
```

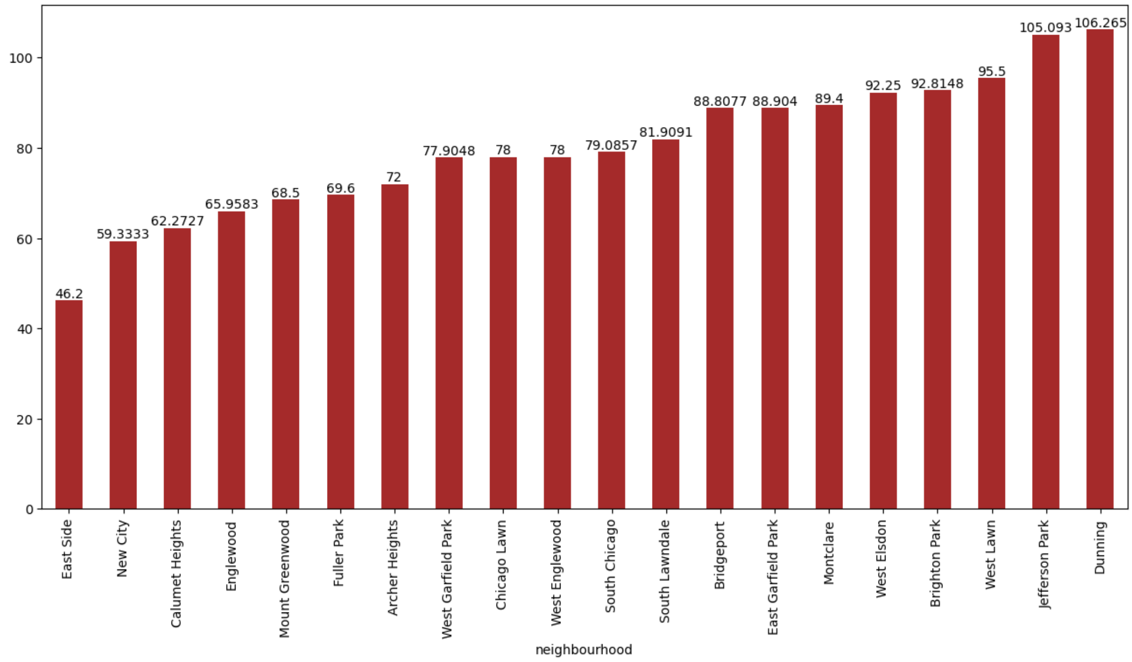
In [38]:

```
plt.figure(figsize=(15,7))
ax= df.price.groupby(df.neighbourhood).mean().sort_values(ascending=False).nlargest(20).
for i in ax.containers:
    ax.bar_label(i)
```



In [39]:

```
# Average price of bottom 20 neighbourhood
plt.figure(figsize=(15,7))
ax= df.price.groupby(df.neighbourhood).mean().sort_values(ascending=False).nsmallest(20)
for i in ax.containers:
    ax.bar_label(i)
```



In [40]:

```
# Mean -  Σ(N)/N
# Median - Sort data in ascending order, even(m1+m2)/2, Odd-Exact Midpoint
# Mode - Most Frequent or repetitve

# Variance = Σ(x-mean)^2/N-1 N -population & N-1 Sample
# Stddev - sqrt(variance) or sqrt(Σ(x-mean)^2/N-1)

# Skeweness = Σ(x-mean)^3/stdev^3
# Kurtosis  = Σ(x-mean)^4/stdev^4

# Quartiles - Sort data in ascending Order. Q1 - N/4, Q2-2N/4, Q3-3N/4
```

In [41]:

```
# np.sum(), np.mean(), np.std(), **2,**3,**4
```

In [42]:

```
sf = [54,18,22,27,28,32,29,34,21,18]

np.sum(sf)
```

Out[42]:

```
283
```

In [43]:

```
# Mean
print(np.mean(sf))
print(np.sum(sf)/len(sf))
```

```
28.3
28.3
```

In [56]:

```
# Median
sf.sort()
```

In [57]:

```
sf
```

Out[57]:

```
[18, 18, 21, 22, 27, 28, 29, 32, 34, 54]
```

In [62]:

```
sf1 = [54,18,22,27,28,32,29,34,21]

sf1

sf1.sort()

sf1

print(np.median(sf1))
```

```
28.0
```

In [45]:

```
sf

print(np.median(sf))
```

```
27.5
```

In [46]:

```
# Mode
import statistics as st
```

In [47]:

```
st.mode(sf)
```

Out[47]:

```
18
```

In [48]:

```
# Variance

sf

((18-28.3)**2+(18-28.3)**2+(21-28.3)**2+(22-28.3)**2+
 (27-28.3)**2+(28-28.3)**2+(29-28.3)**2+(32-28.3)**2+
 (34-28.3)**2+(54-28.3)**2)/len(sf)
```

Out[48]:

```
101.41
```

In [49]:

```
np.var(sf)
```

Out[49]:

```
101.41000000000001
```

In [50]:

```
# Standard Deviation

np.sqrt(
((18-28.3)**2+(18-28.3)**2+(21-28.3)**2+(22-28.3)**2+
 (27-28.3)**2+(28-28.3)**2+(29-28.3)**2+(32-28.3)**2+
 (34-28.3)**2+(54-28.3)**2)/len(sf))
```

Out[50]:

```
10.070253224224304
```

In [51]:

```
np.std(sf)
```

Out[51]:

```
10.070253224224304
```

In [52]:

```
# skewness

((18-28.3)**3+(18-28.3)**3+(21-28.3)**3+(22-28.3)**3+
 (27-28.3)**3+(28-28.3)**3+(29-28.3)**3+(32-28.3)**3+
 (34-28.3)**3+(54-28.3)**3)/((len(sf)-1)*np.std(sf)**3)
```

Out[52]:

```
1.5650102943080282
```

In [53]:

```
pd.DataFrame(sf).skew()
```

Out[53]:

```
0    1.670287
dtype: float64
```

In [54]:

```
from scipy.stats import skew
```

In [55]:

```
skew(sf)
```

Out[55]:

```
1.4085092648772253
```

In [ ]:

```

```